CS 581

Advanced Artificial Intelligence

March 27, 2024

Announcements / Reminders

- Please follow the Week 10 To Do List instructions (if you haven't already)

- Programming Assignment #02 due on Sunday (04/07) at 11:59 PM CST
- Written Assignment #03 due on Sunday (03/31) at 11:59 PM CST

Plan for Today

- Convolutional Neural Networks
- Recurrent Neural Networks
  - Basic RNNs
  - Long Term Short Term Memory (LSTM)

Neural Networks

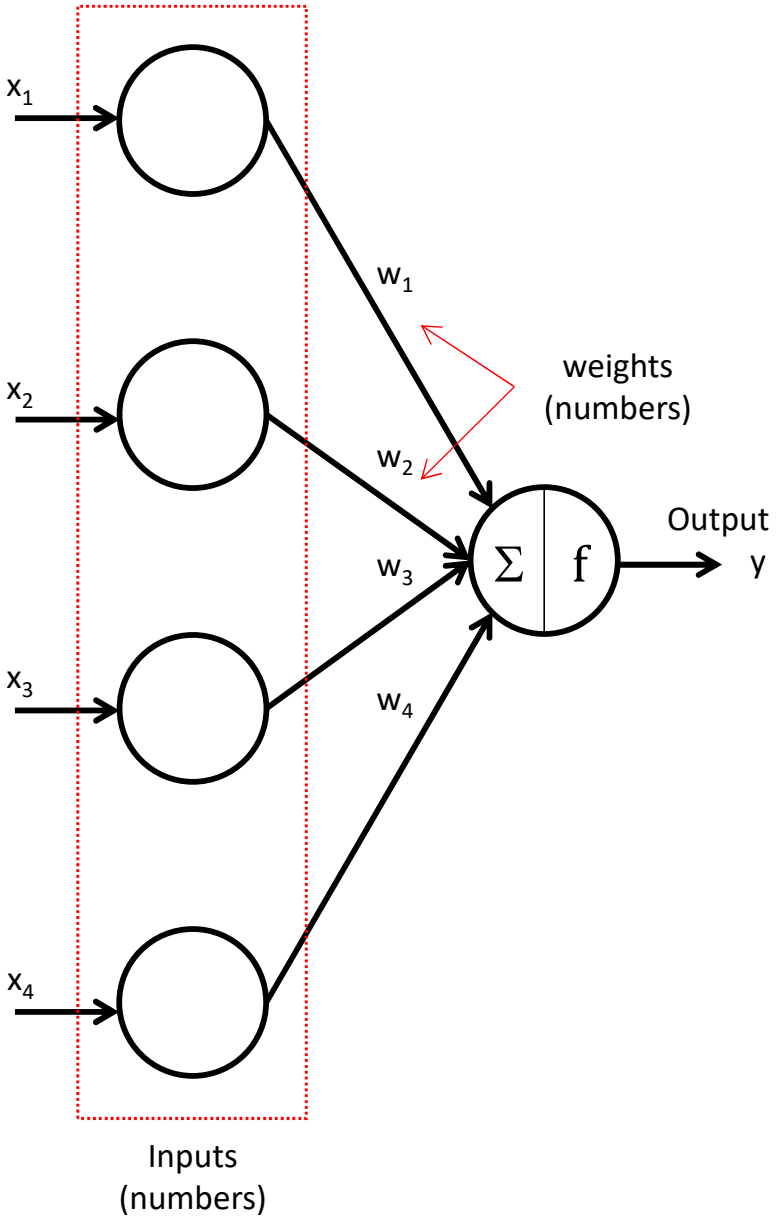
Main Machine Learning Categories

Supervised learning	Unsupervised learning	Reinforcement learning
Supervised learning is one of the most common techniques in machine learning. It is based on **known relationship(s) and patterns within data** (for example: relationship between inputs and outputs). Frequently used types: **regression**, and **classification**.	**Unsupervised learning** involves finding underlying patterns within data. Typically used in **clustering** data points (similar customers, etc.)	Reinforcement learning is inspired by behavioral psychology. It is **based on a rewarding / punishing an algorithm**. Rewards and punishments are based on algorithm's action within its environment.

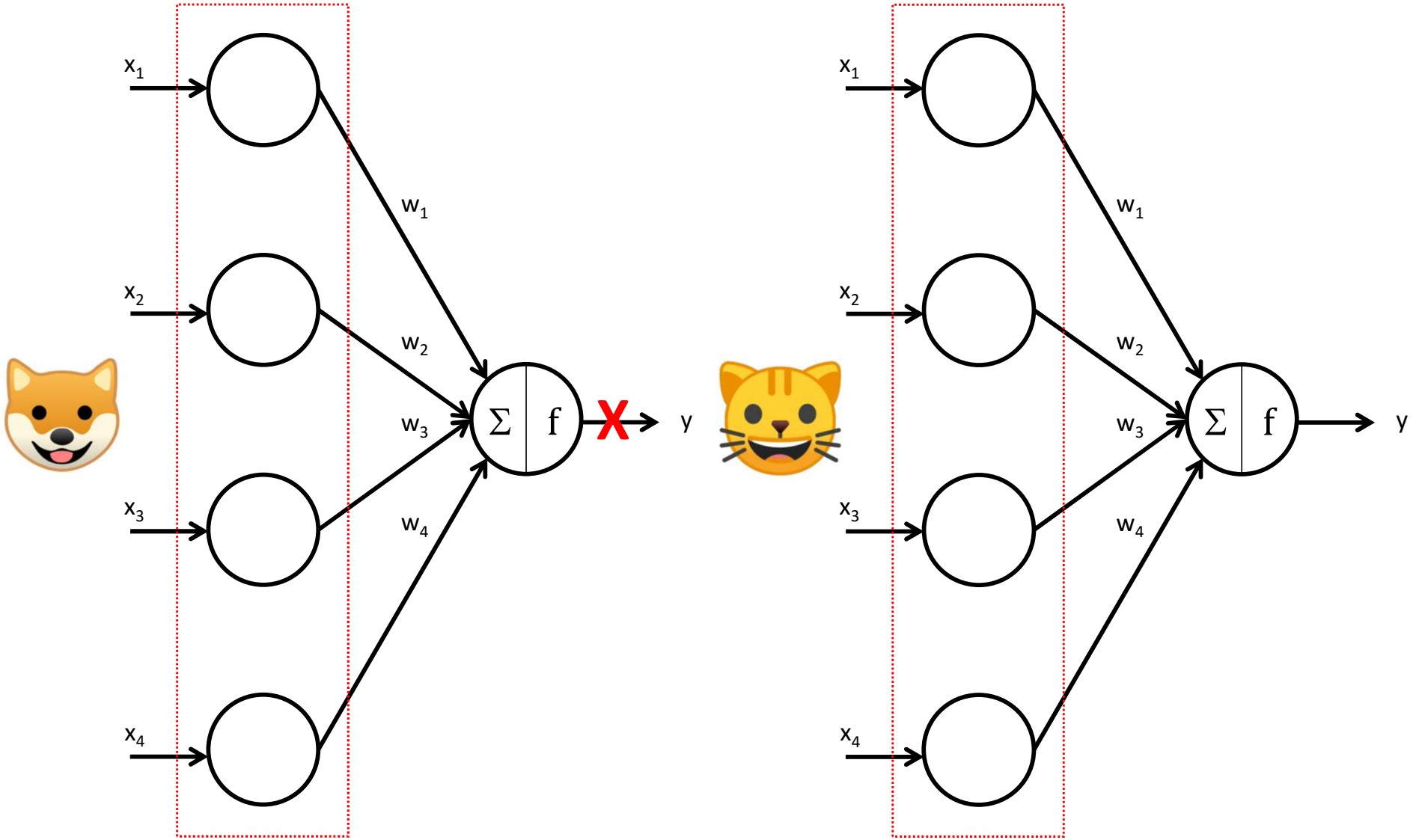
Artificial Neuron (Perceptron)

A (**single-layer**) **perceptron** is a model of a biological neuron. It is made of the following components:

- inputs x_i - numerical values representing information
- weights w_i - numerical values representing how "important" corresponding input is
- weighted sum: $\Sigma\ w_i * x_i$
- activation function f that decides if the neuron "fires"



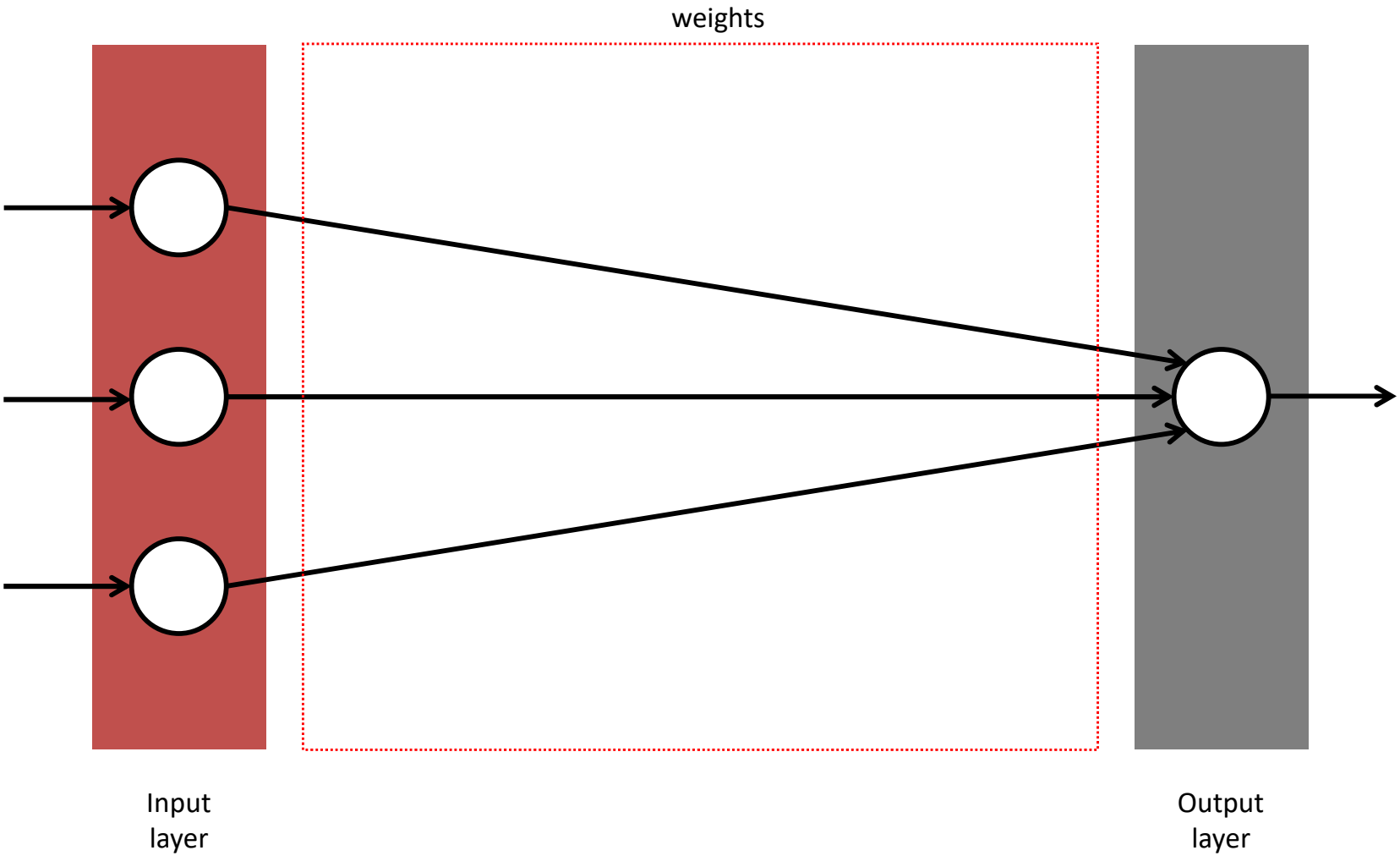
Single-layer Perceptron as a Classifier



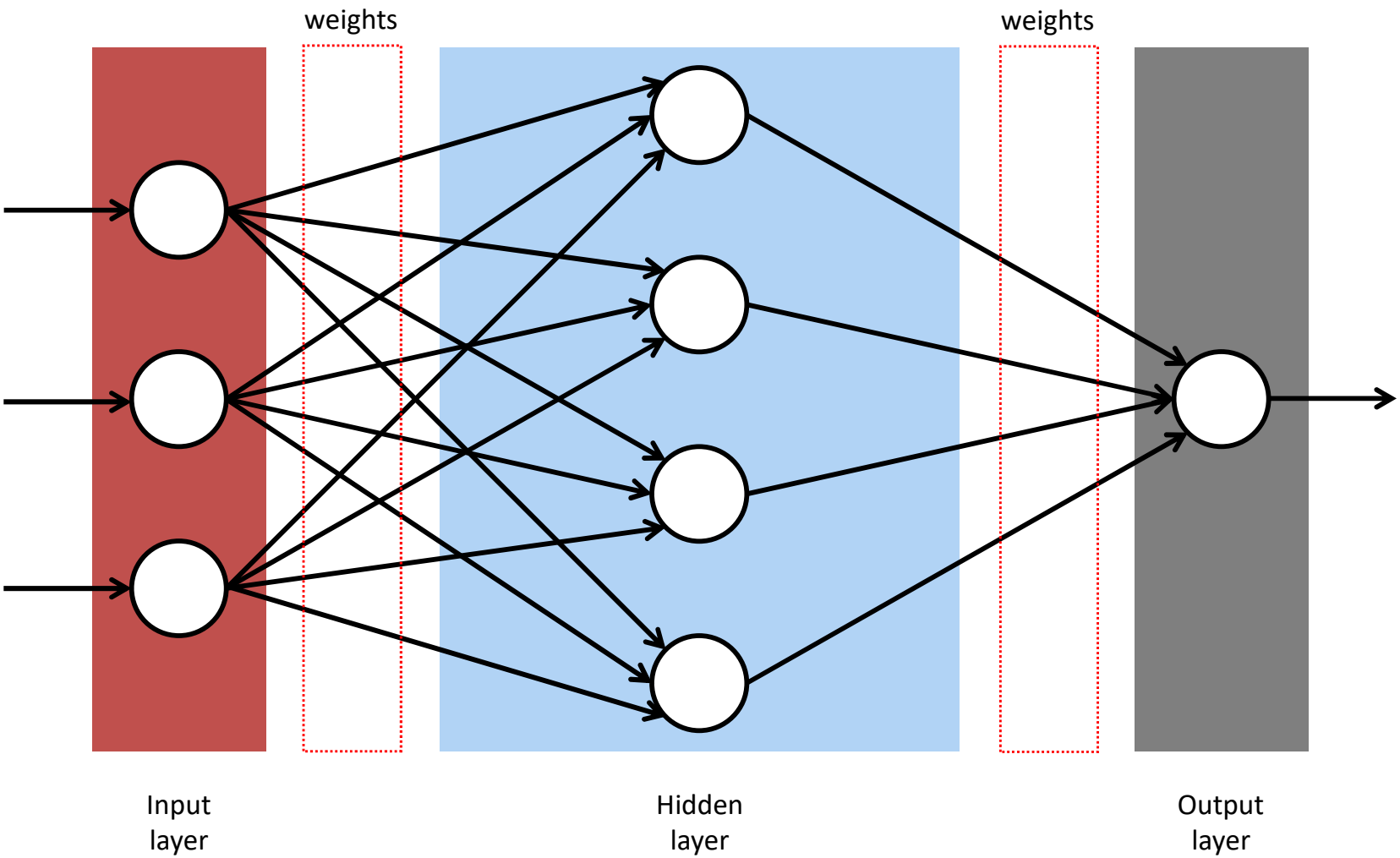
$\Sigma\, w_i * x_i < 0 \rightarrow f = 0$ → **NOT CAT**

$\Sigma\, w_i * x_i \geq 0 \rightarrow f = 1$ → **CAT**

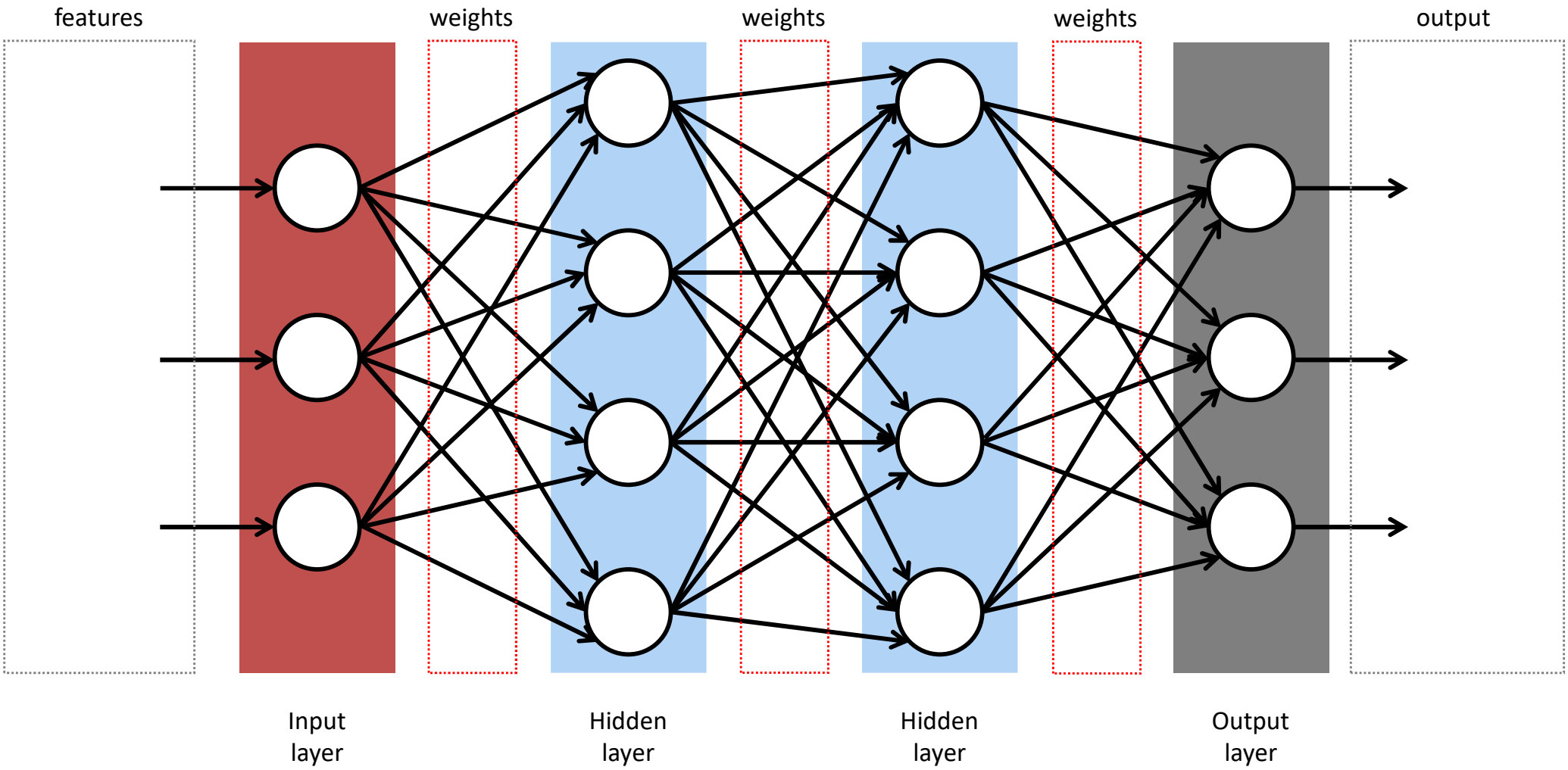
Basic Neural Unit



Hidden Layer



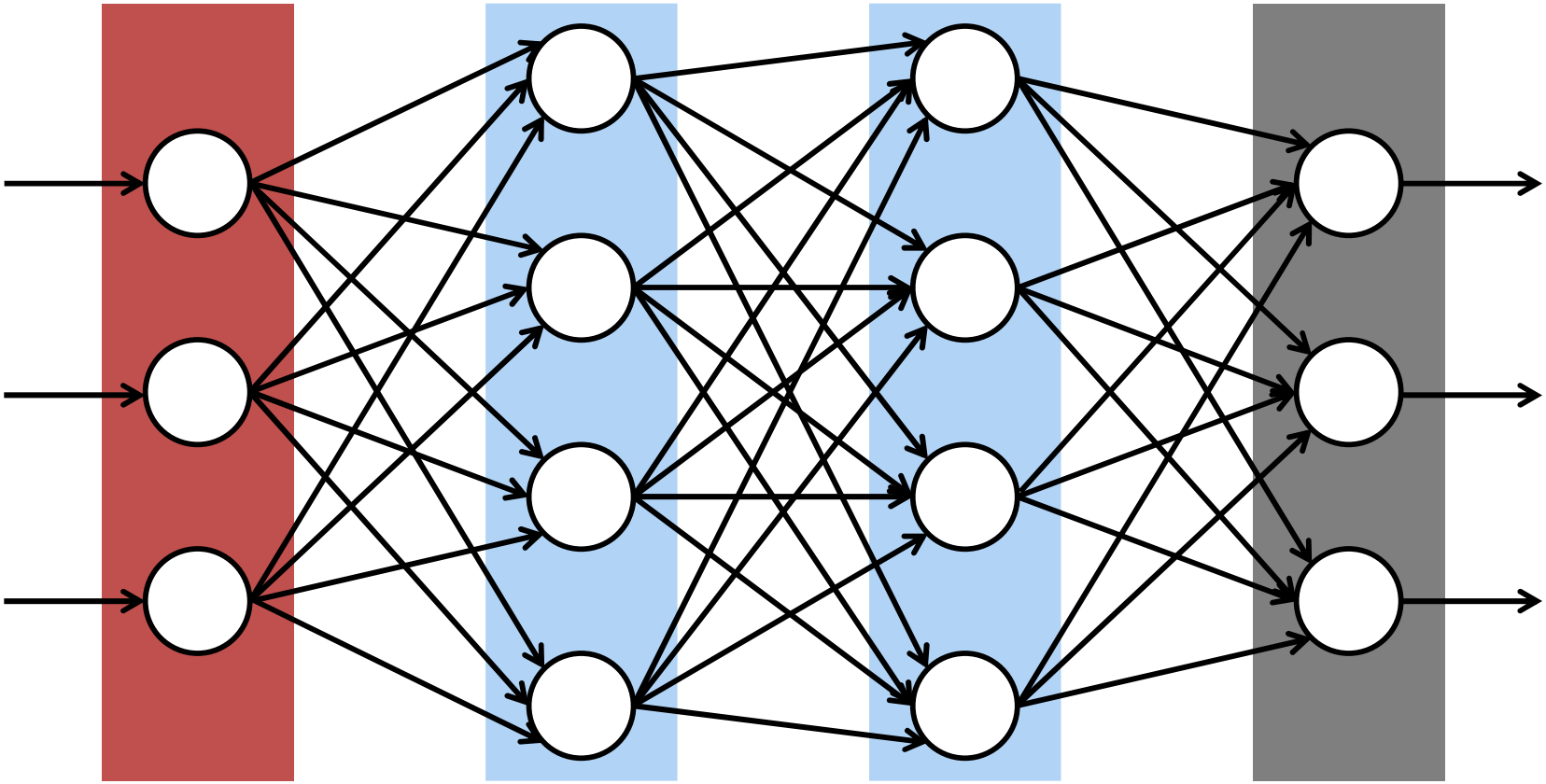
Feedforward Neural Network



Also called (historically): multi-layer perceptron

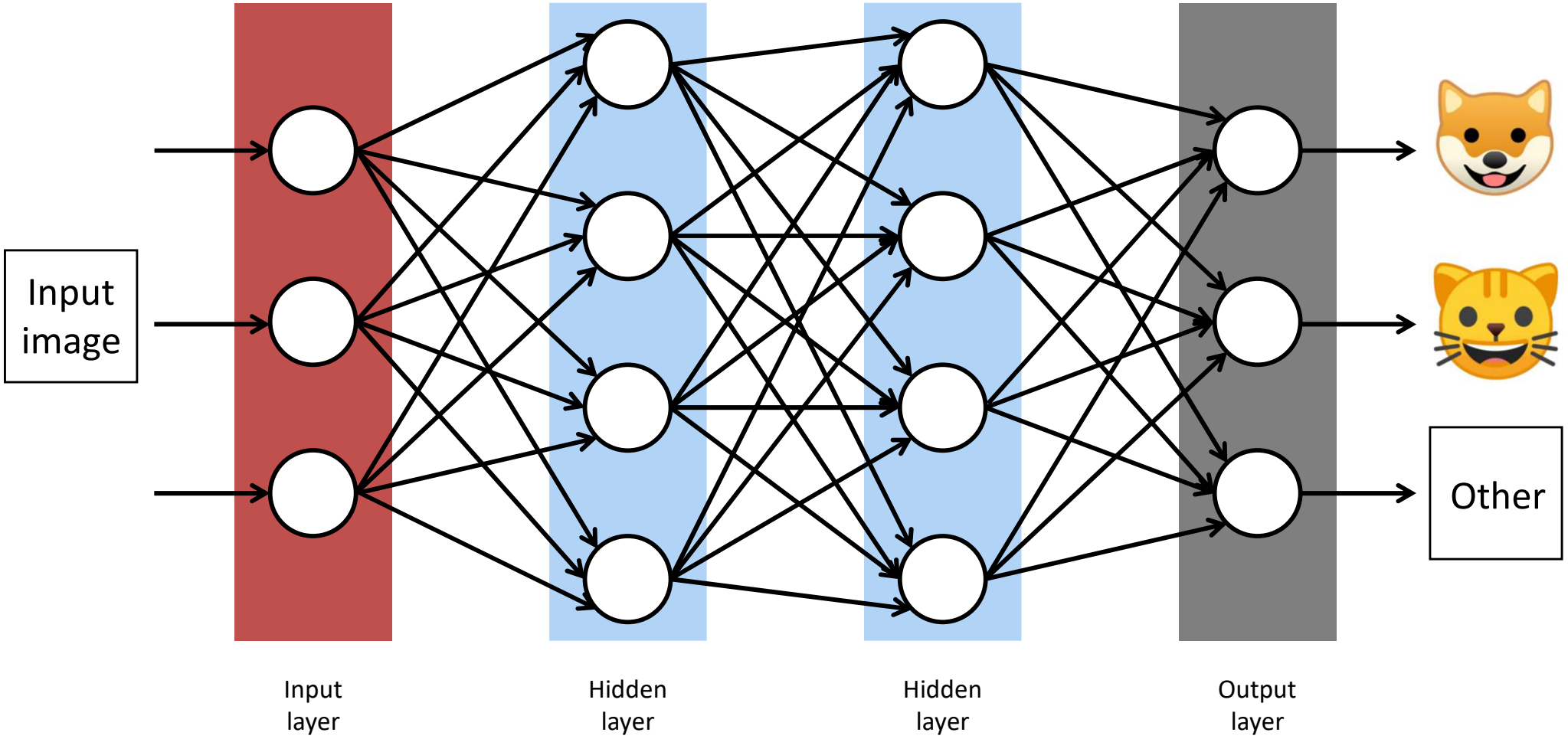
Artificial Neural Network (ANN)

An artificial neural network is made of **multiple artificial neuron layers**.

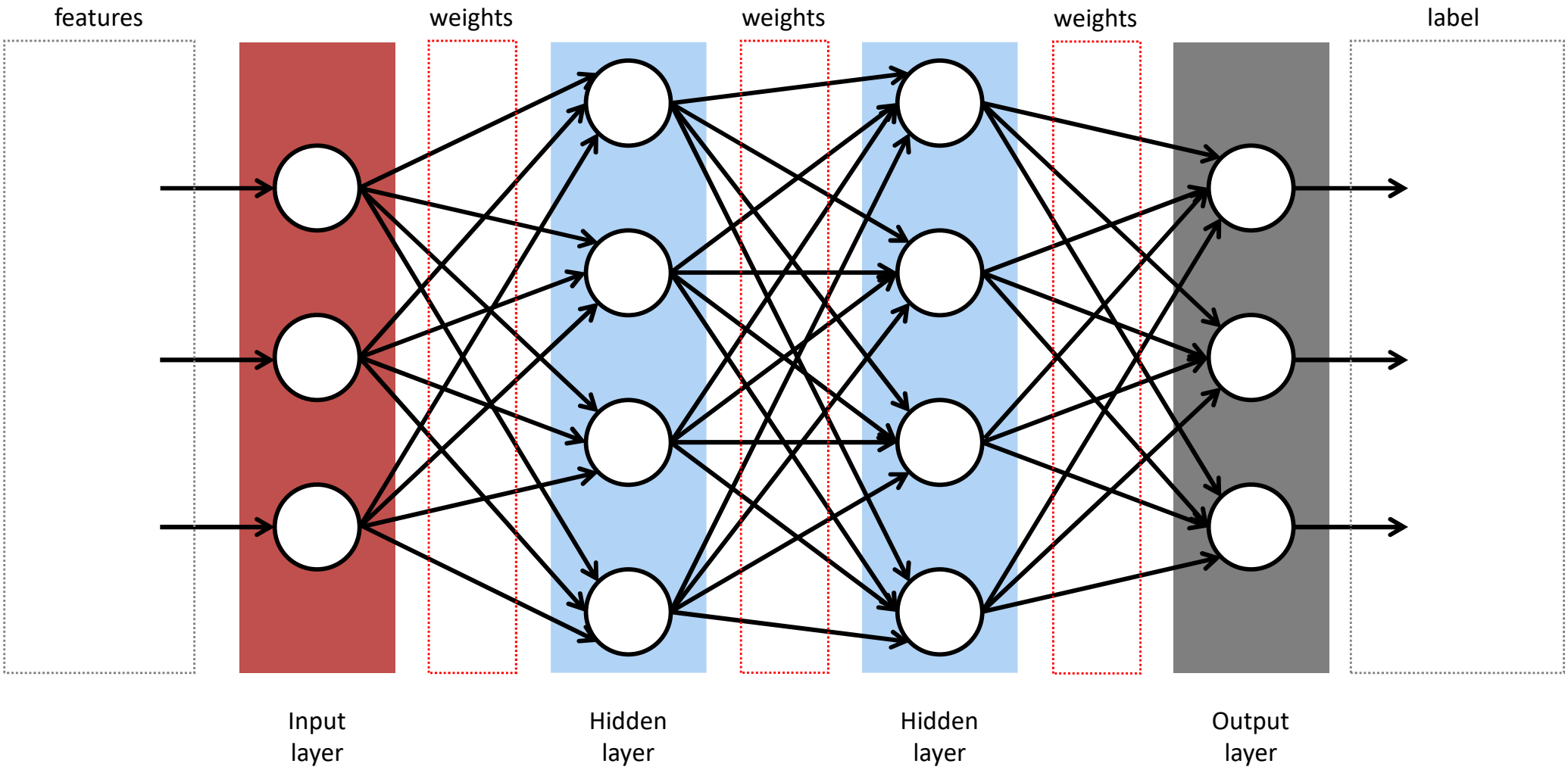


ANN as an Image Classifier

An artificial neural network can be used as a **classifier** as well.

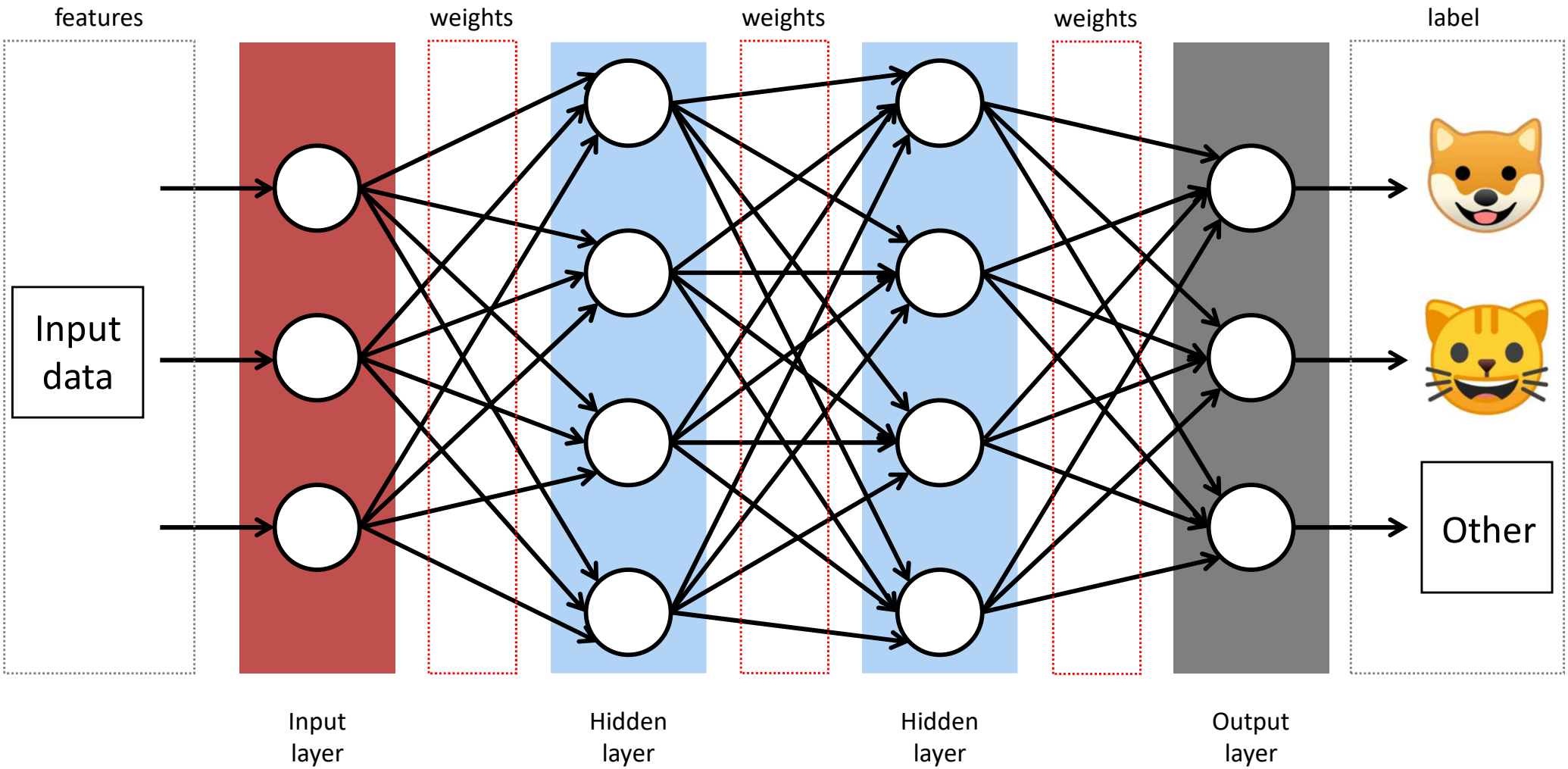


ANN as a Classifier



ANN: Supervised Learning

In order to work properly a classifier **needs to be trained** first with **labeled data**.

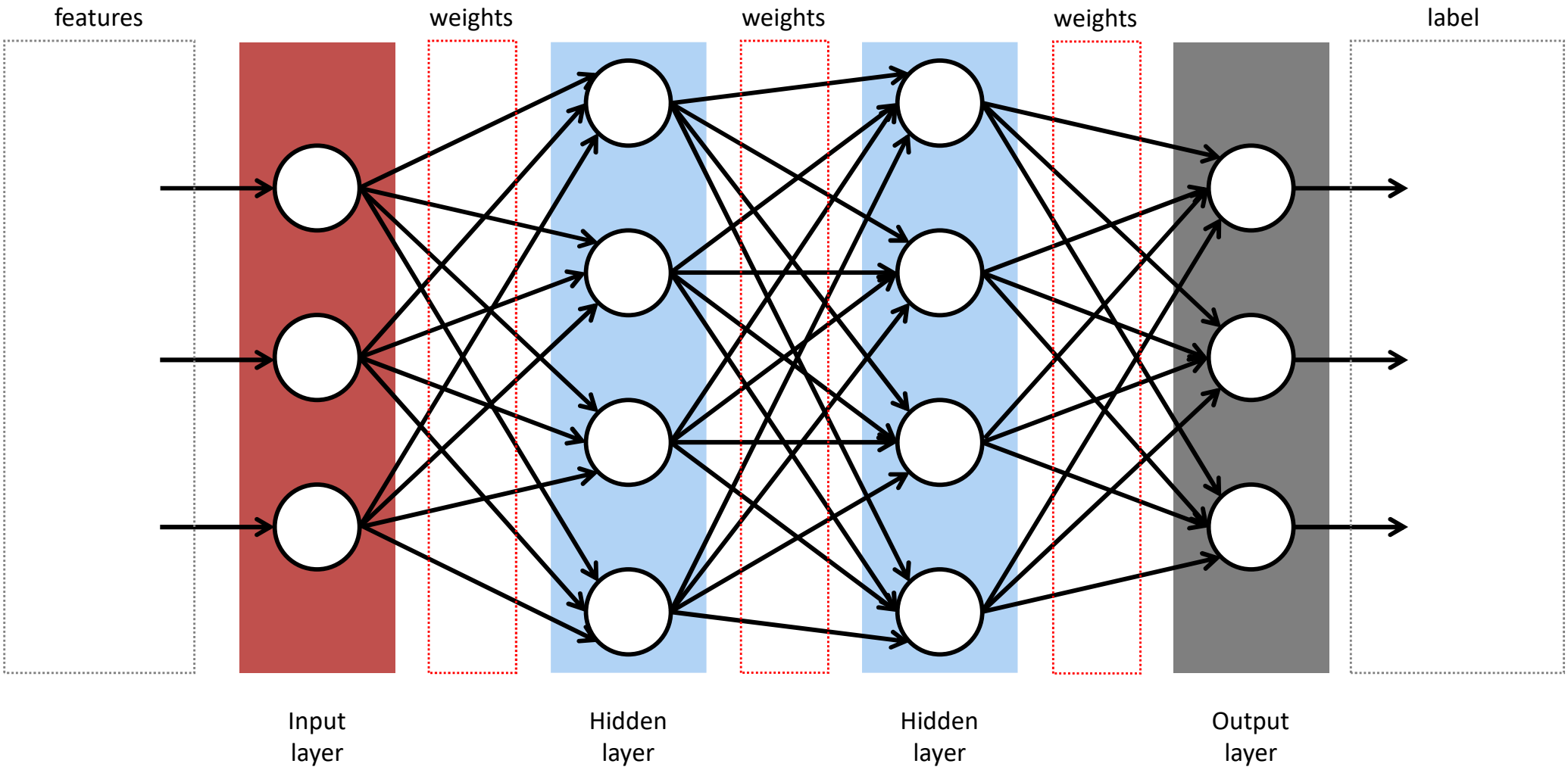


Training will **adjust all the weights** within this artificial neural network.

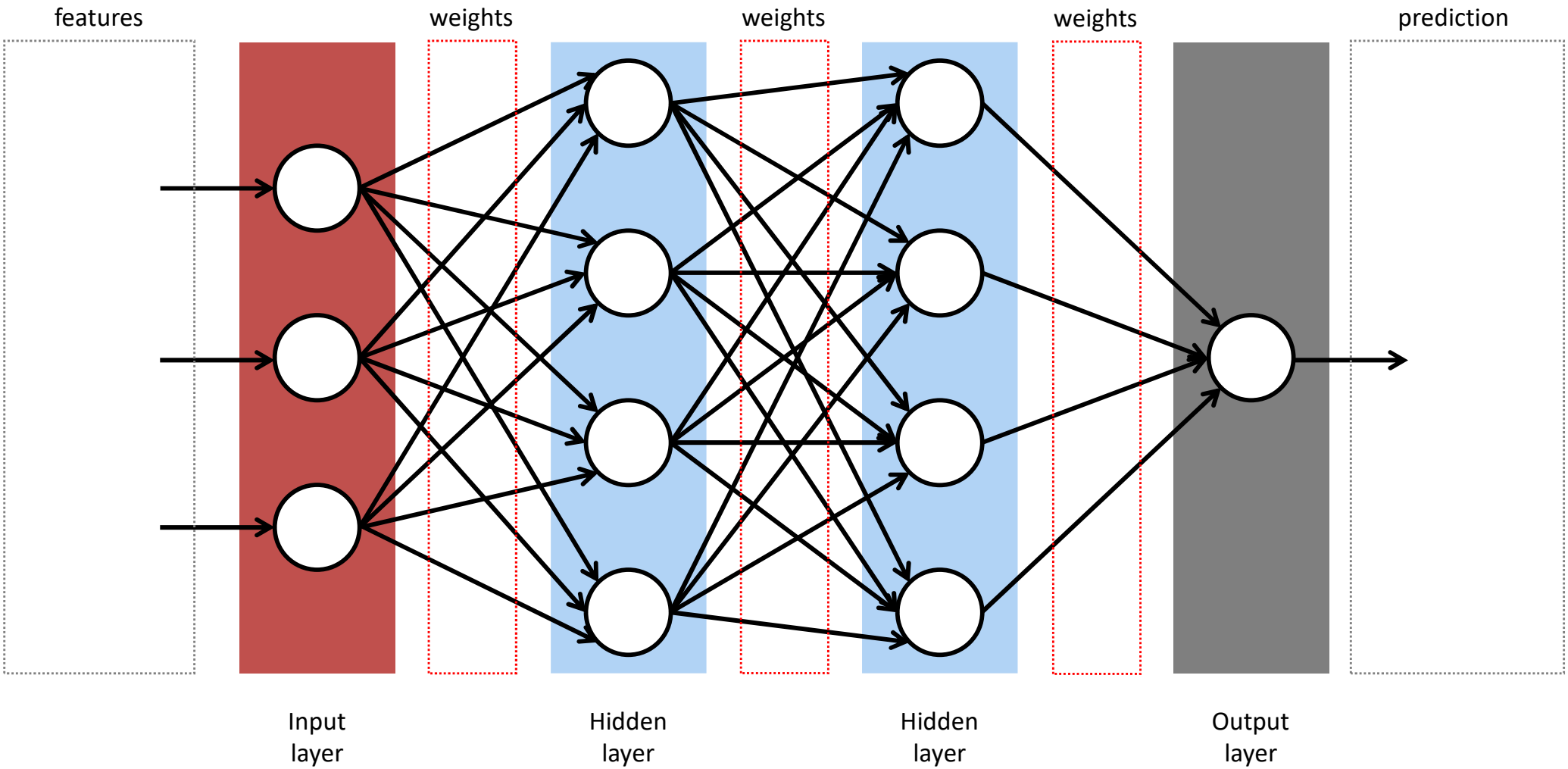
Exercise: ANN Demo

http://playground.tensorflow.org/

ANN for Classification



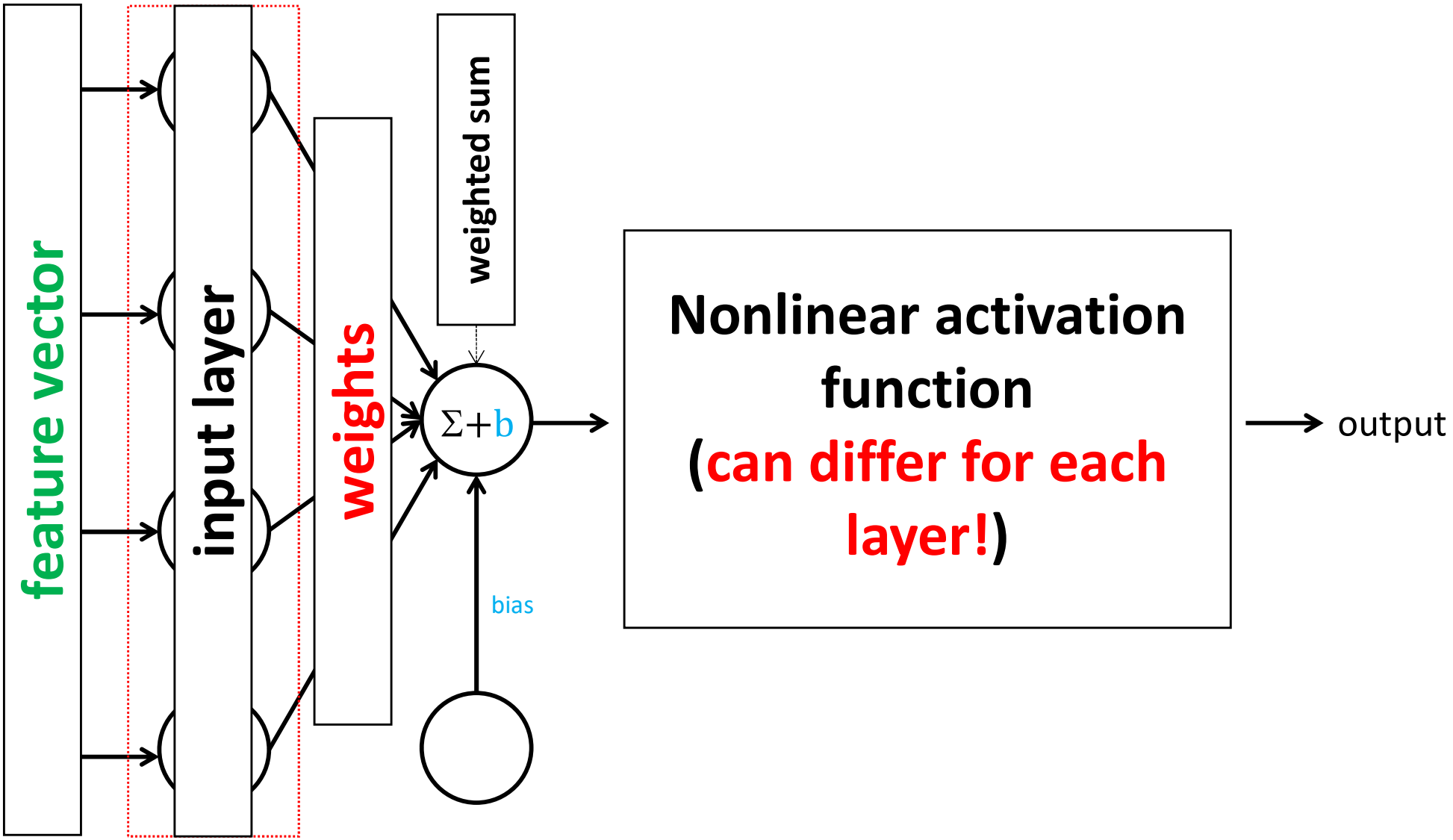
ANN for Regression



Main Machine Learning Categories

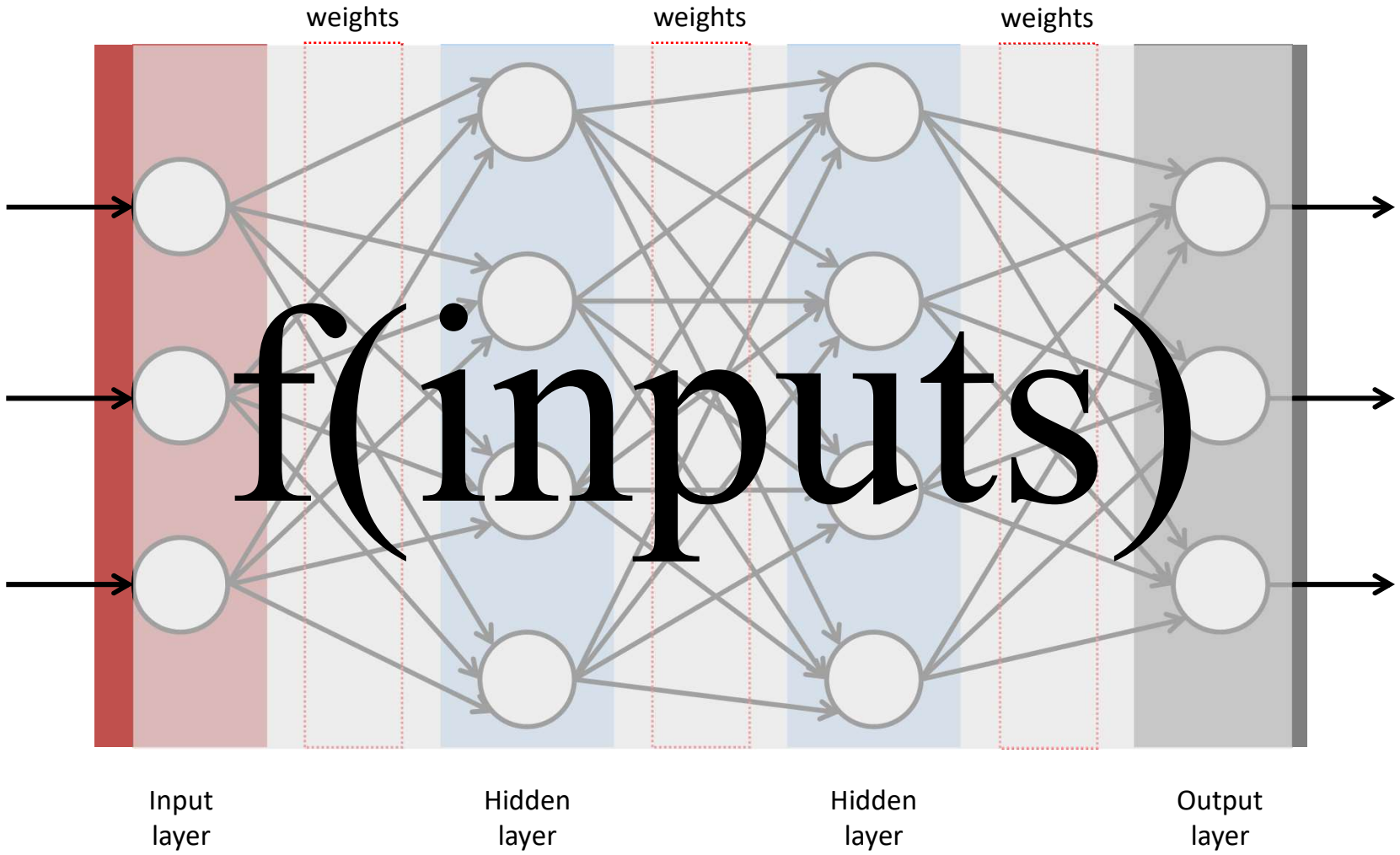
Supervised learning	Unsupervised learning	Reinforcement learning
Supervised learning is one of the most common techniques in machine learning. It is based on **known relationship(s) and patterns within data** (for example: relationship between inputs and outputs). Frequently used types: **regression**, and **classification**.	**Unsupervised learning** involves finding underlying patterns within data. Typically used in **clustering** data points (similar customers, etc.)	Reinforcement learning is inspired by behavioral psychology. It is **based on a rewarding / punishing an algorithm**. Rewards and punishments are based on algorithm's action within its environment.

Basic Neural Unit

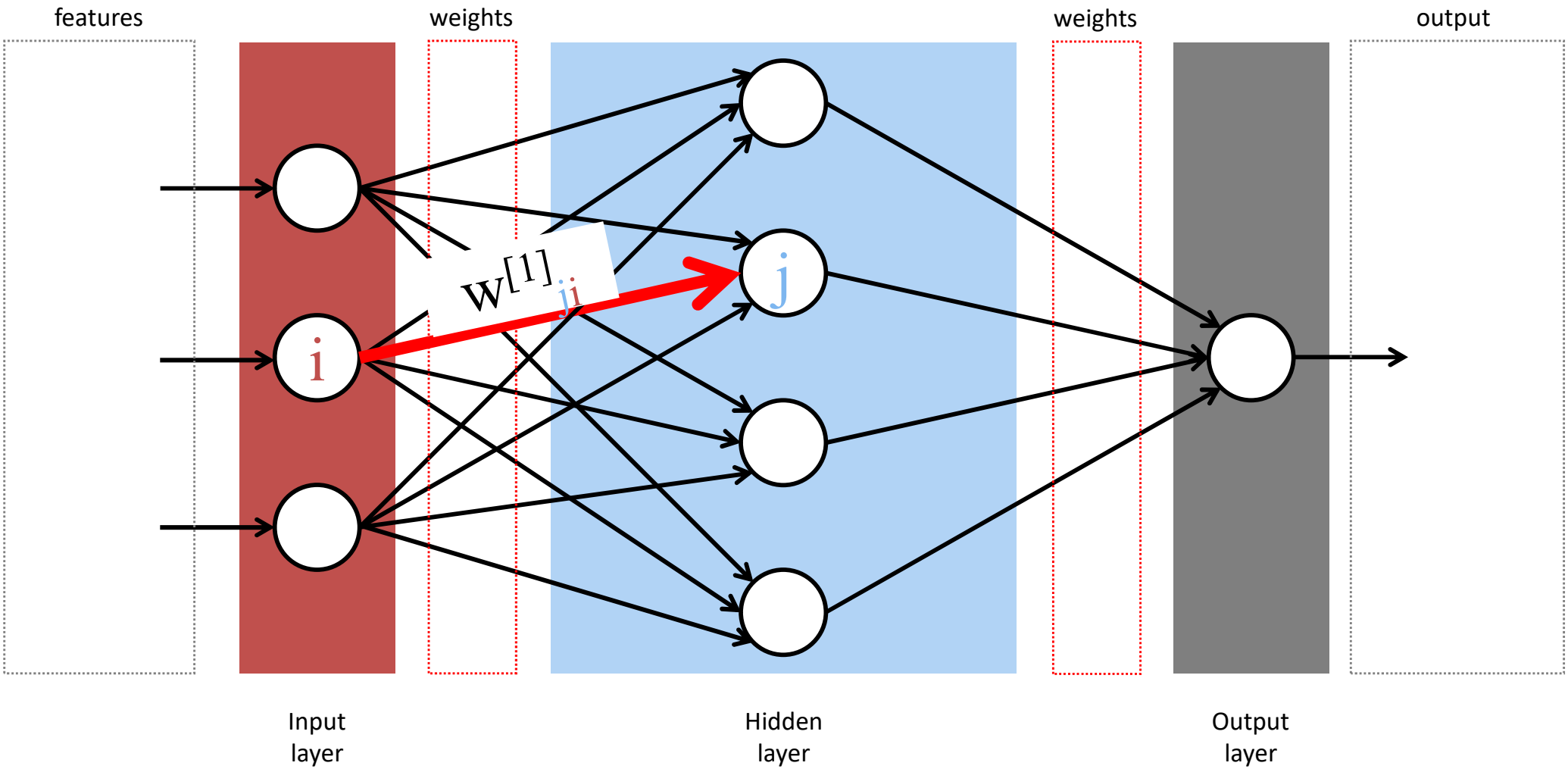


ANN as a Complex Function

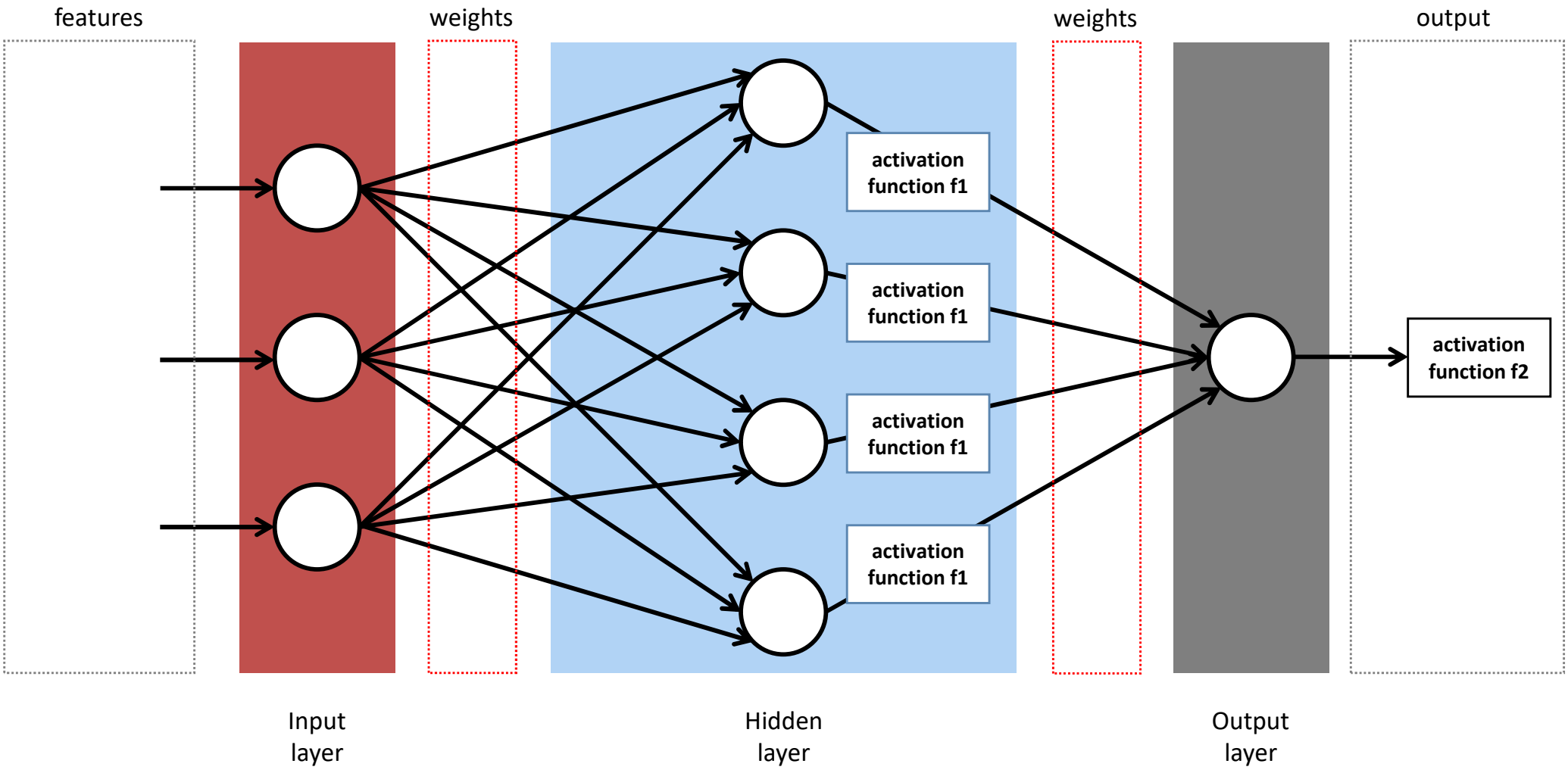
In ANNs **hypotheses take form of complex algebraic circuits** with tunable connection strengths (weights).



2 Layer Network

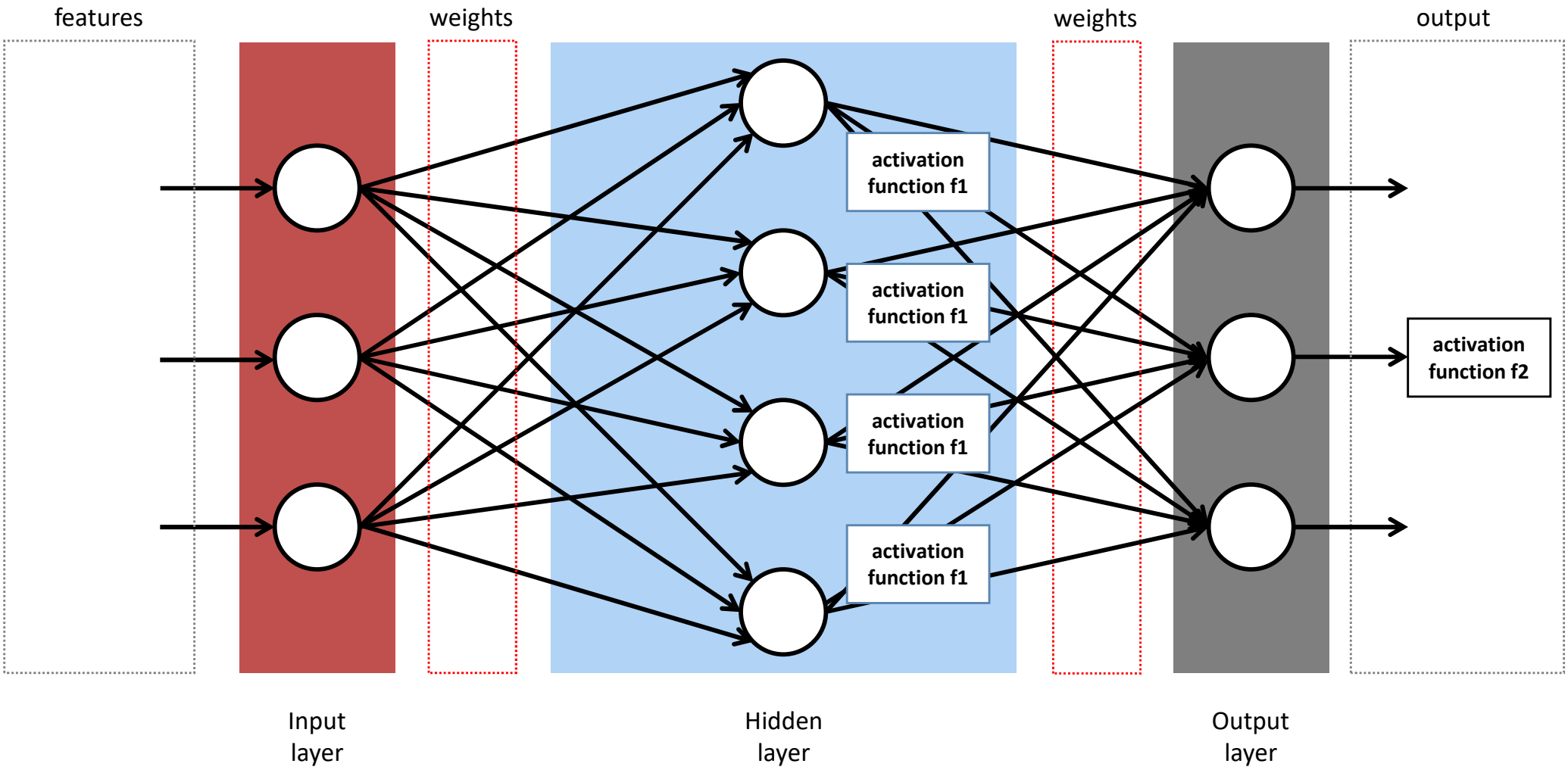


2 Layer Network



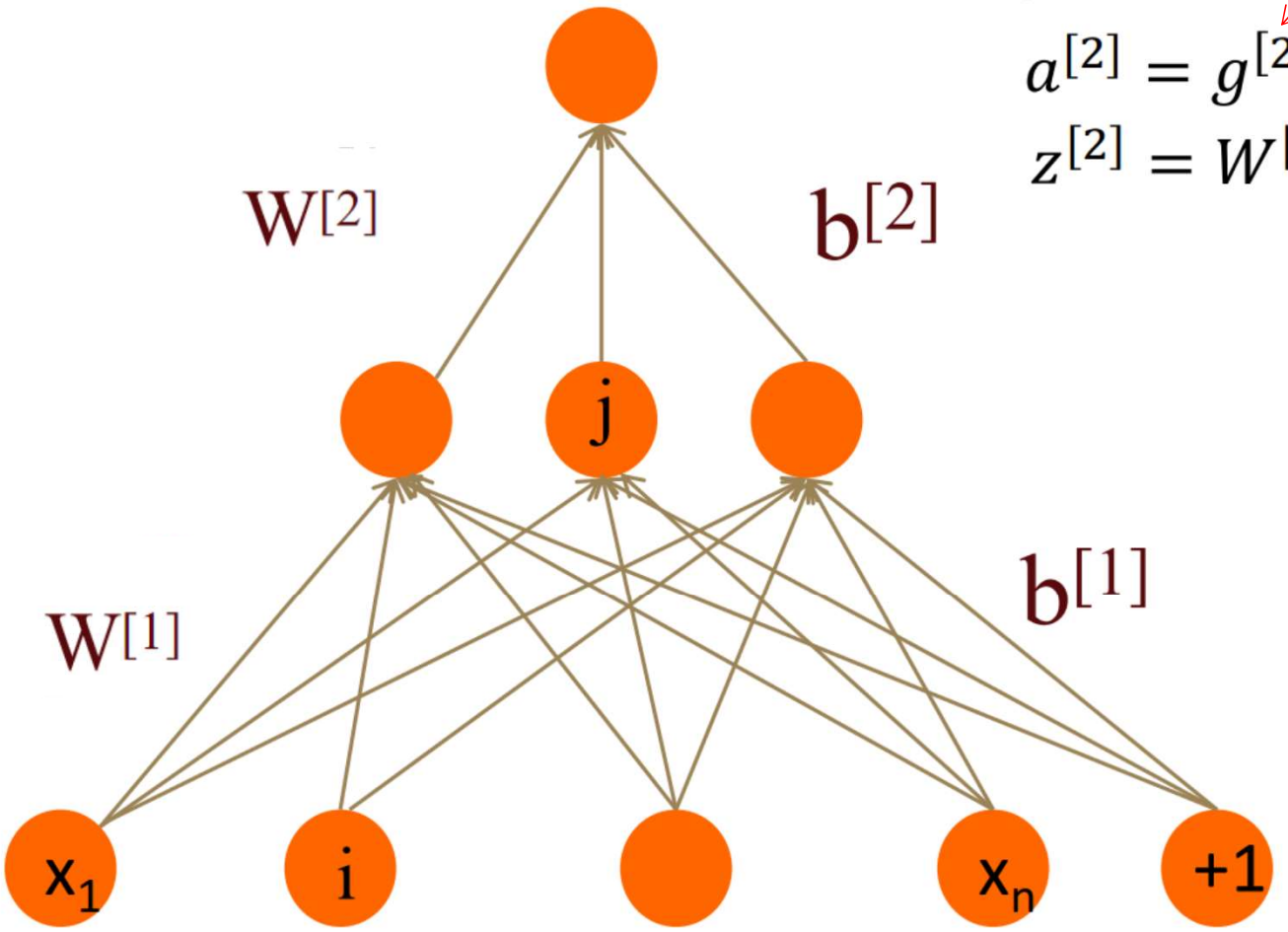
Activation function f1: sigmoid, tanh, ReLU, etc. | Activation function f2: sigmoid

2 Layer Network



Activation function f1: sigmoid, tanh, ReLU, etc. | Activation function f2: softmax

Multilayer Neural Net: Notation



$$y = a^{[2]}$$ sigmoid or softmax

$$a^{[2]} = g^{[2]}(z^{[2]})$$

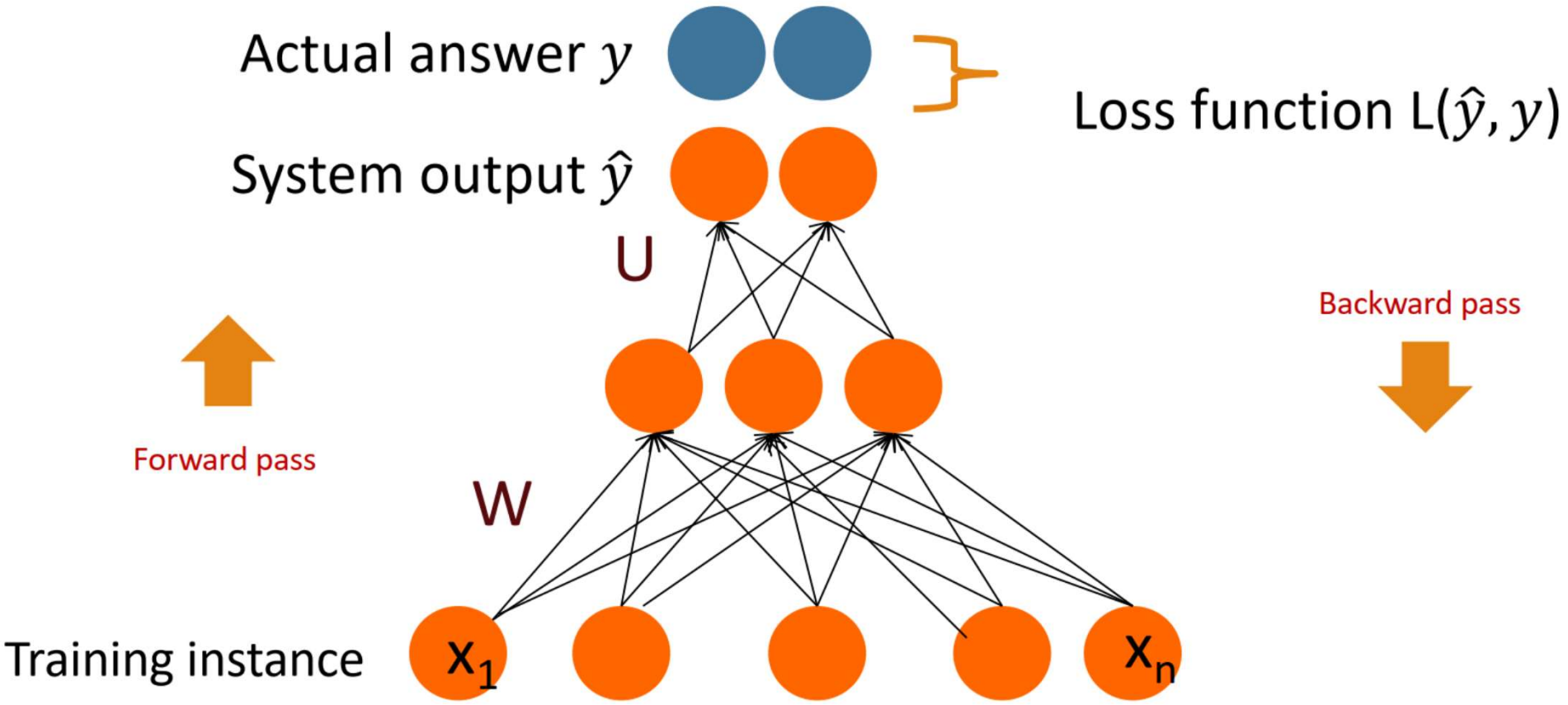
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

ReLU

$$a^{[1]} = g^{[1]}(z^{[1]})$$

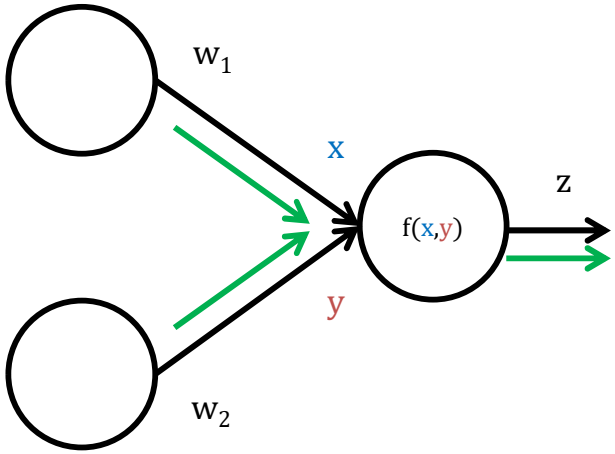
$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

Training Neural Networks



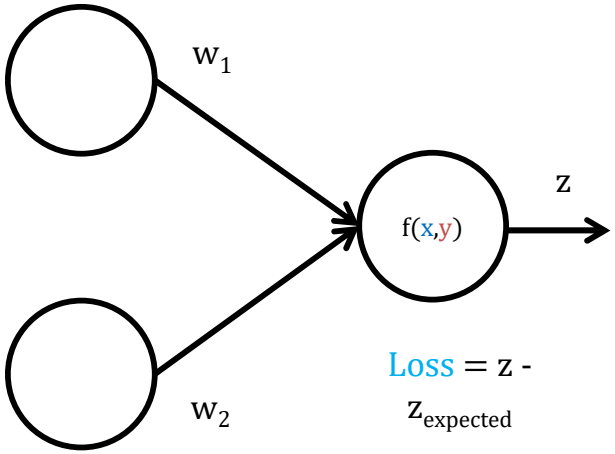
Back-propagation

Feed forward



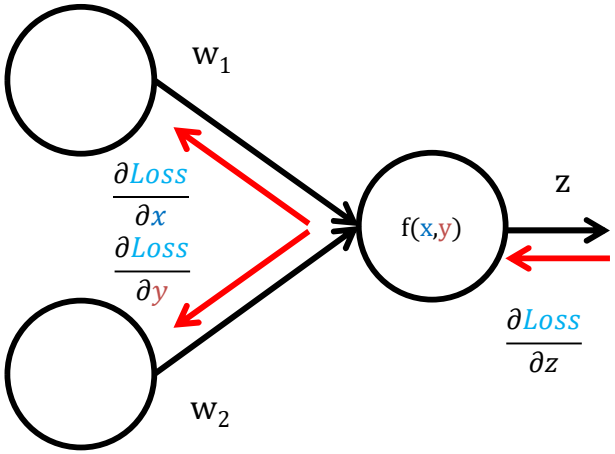
Feed a **labeled sample** through the network

Evaluate Loss



How "incorrect" is the result compare to the label?

Back-propagation



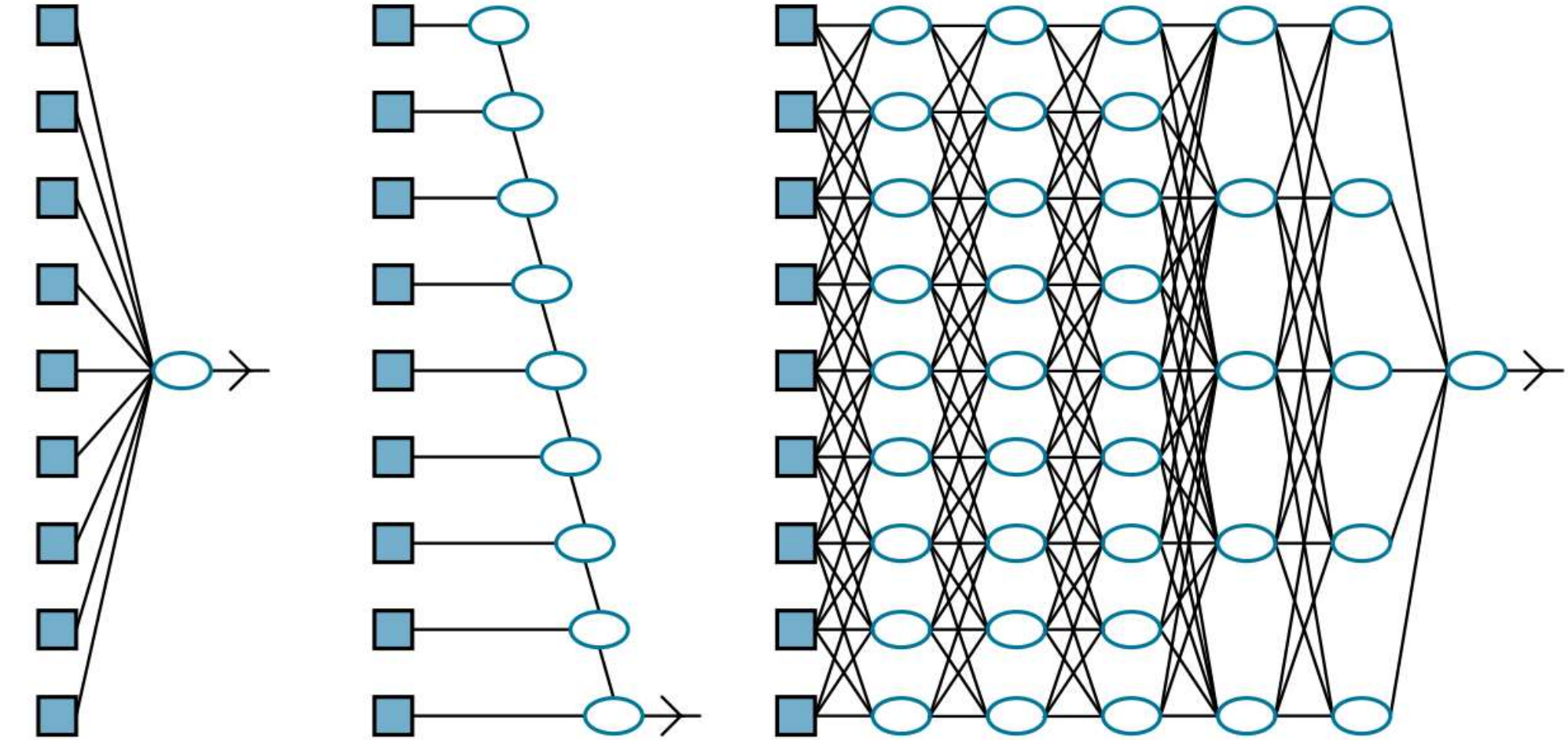
Update weights (use **Gradient Descent**)

Deep Learning

Deep Learning

Deep learning is a broad family of techniques for machine learning (also a sub-field of ML) in which hypotheses take the form of complex algebraic circuits with tunable connections. The word "deep" refers to the fact that the circuits are typically organized into many layers, which means that computation paths from inputs to outputs have many steps.

Shallow vs. Deep Models



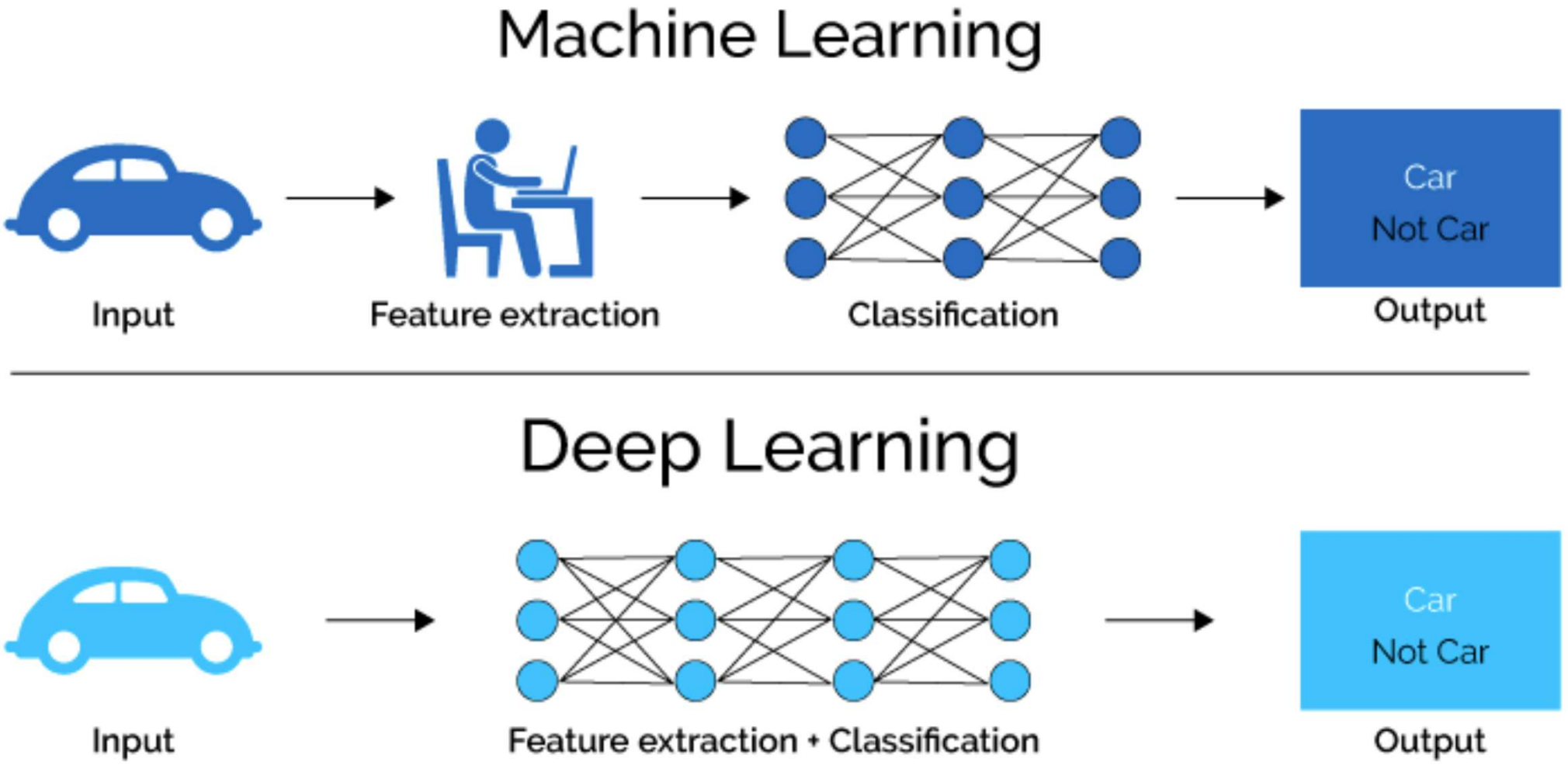
Shallow Model

Shallow Model

Deep Model

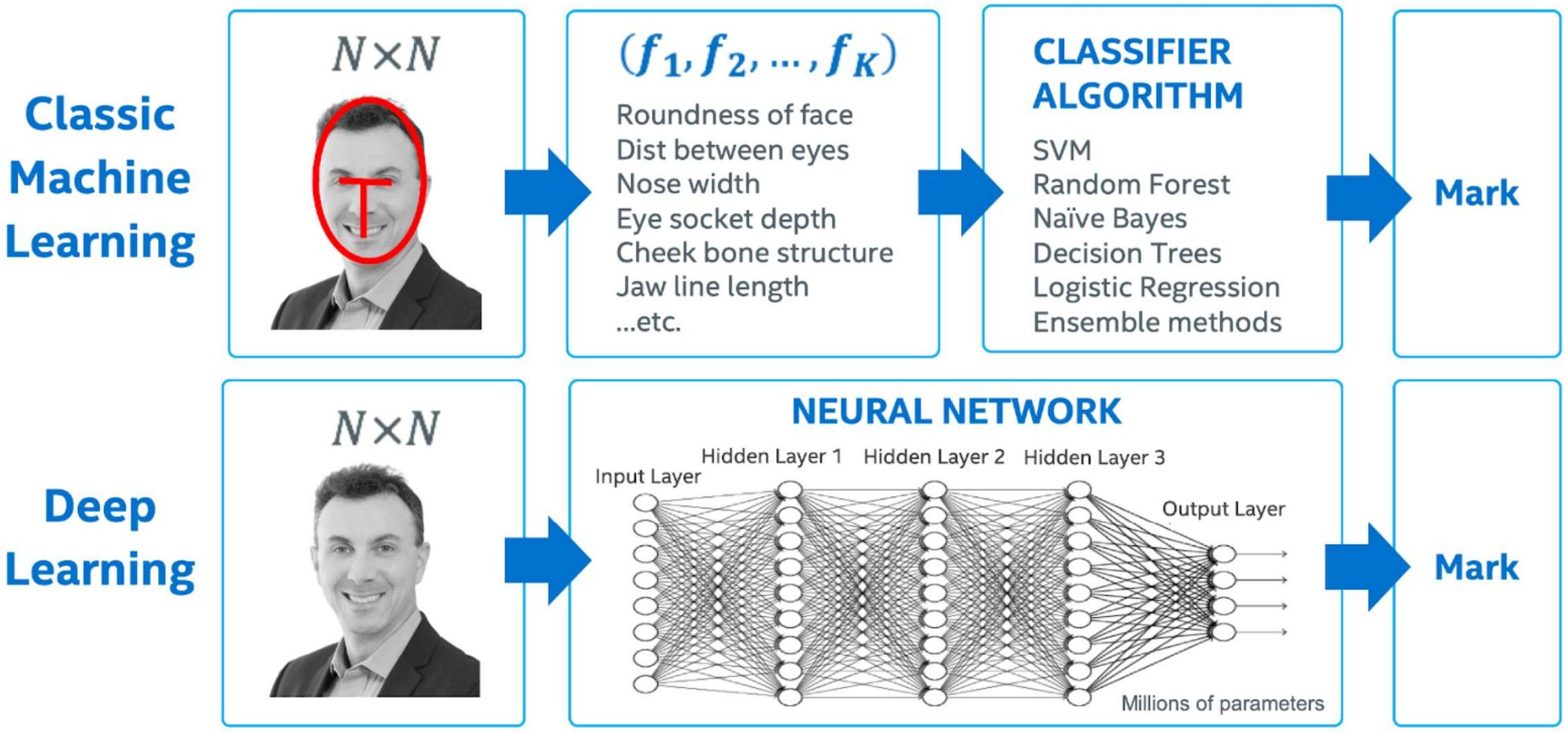
Longer computation path

Machine Learning vs. Deep Learning



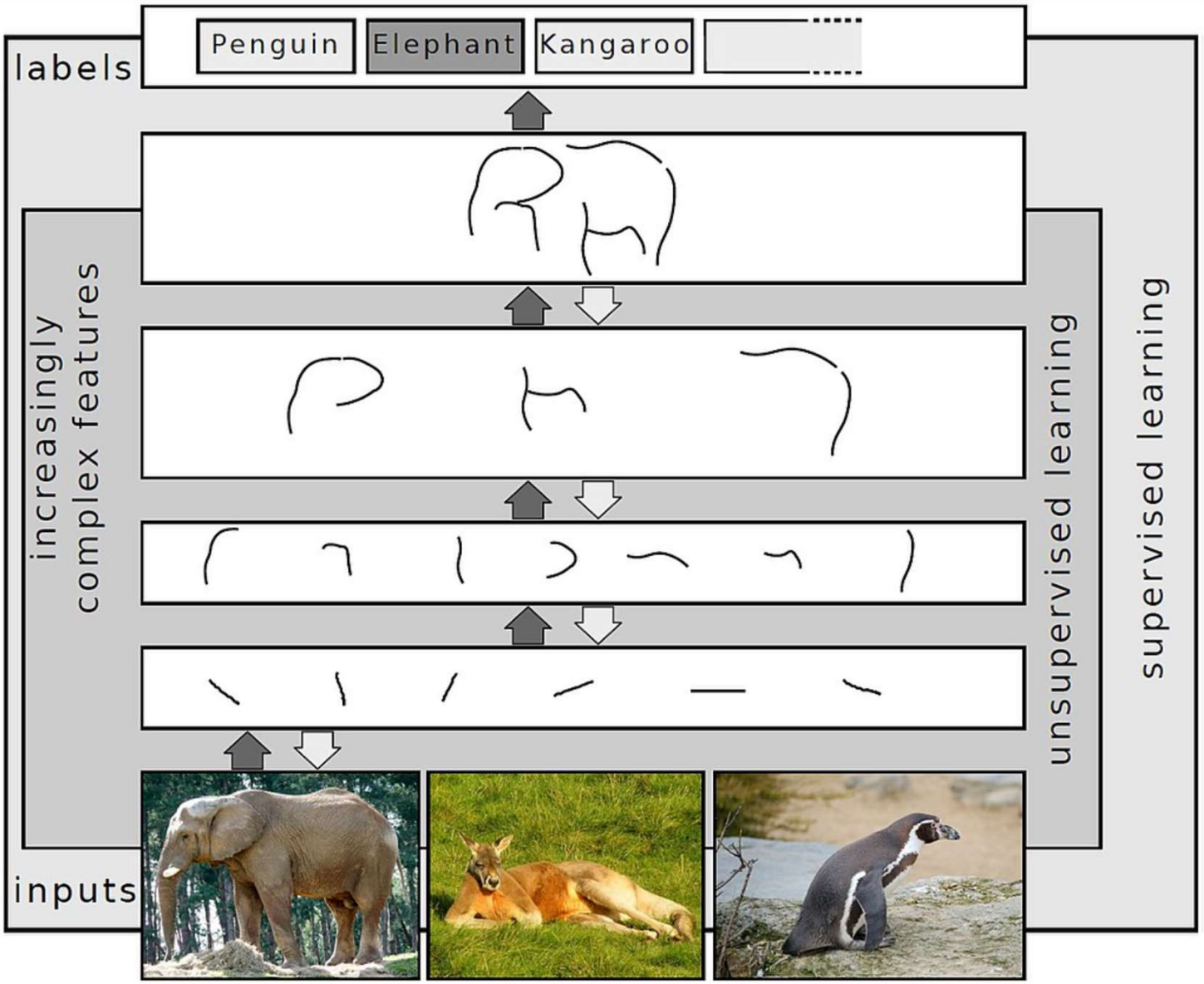
Source: https://www.quora.com/What-is-the-difference-between-deep-learning-and-usual-machine-learning

Machine Learning vs. Deep Learning



Source: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/difference-between-ai-machine-learning-deep-learning.html

Deep Learning: Feature Extraction



Source: https://en.wikipedia.org/wiki/Deep_learning

Convolutional Neural Networks

Convolutional Neural Networks

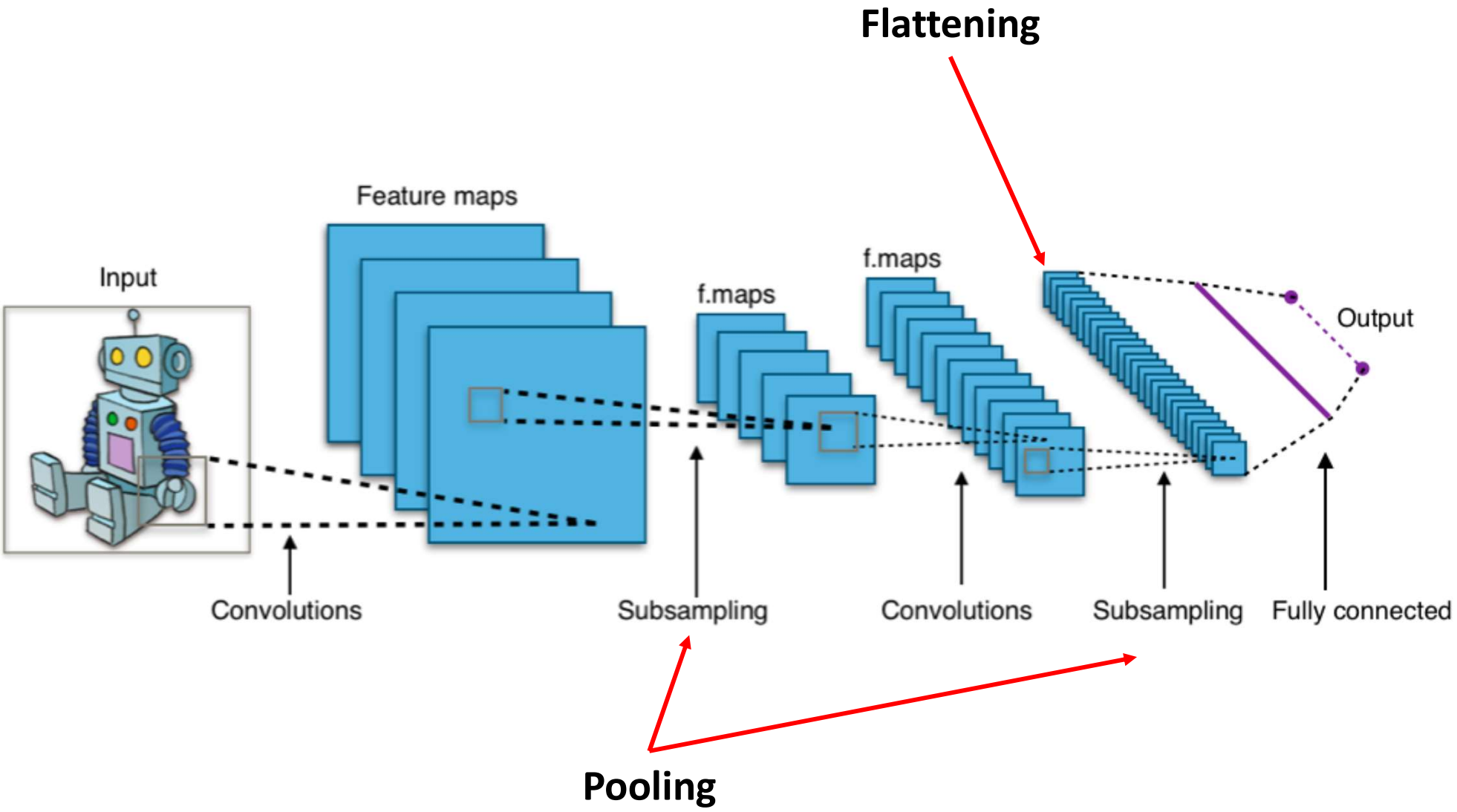
The name Convolutional Neural Network (CNN) indicates that the network employs a mathematical operation called convolution.

Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

CNN is able to successfully capture the spatial dependencies in an image (data grid) through the application of relevant filters.

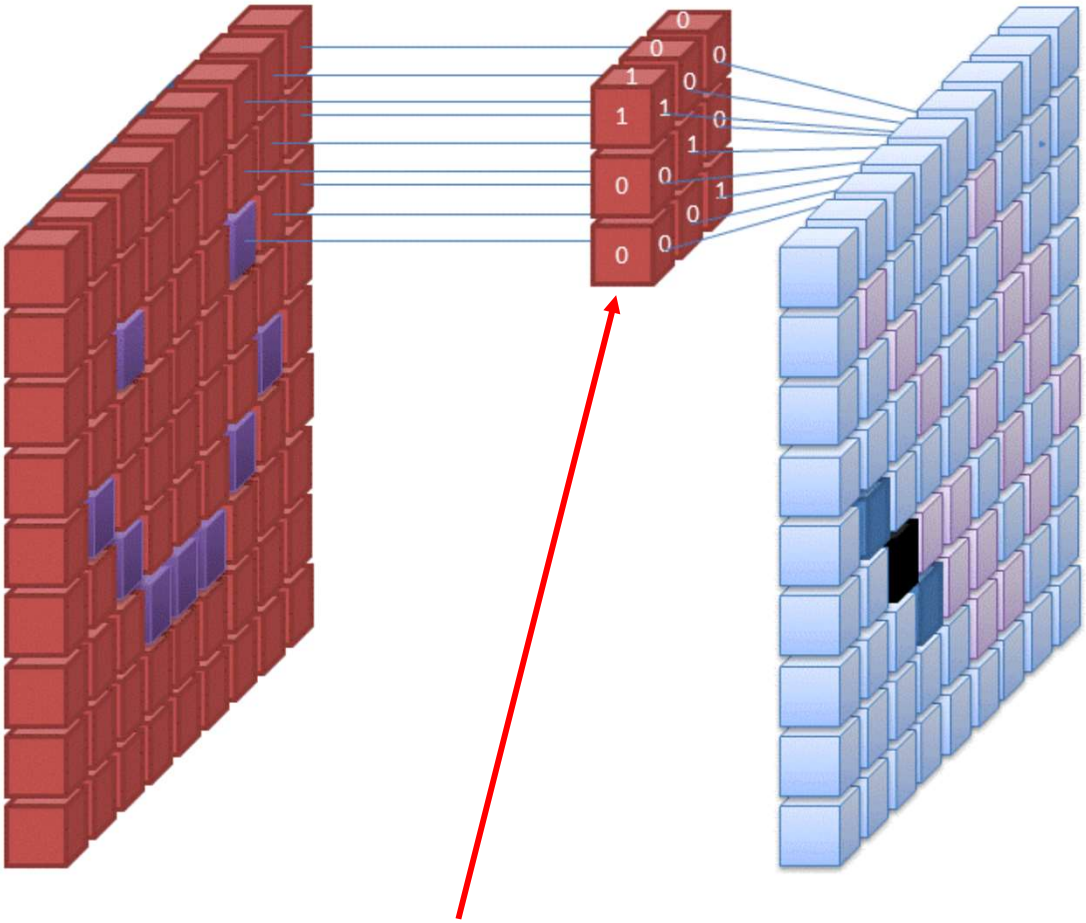
CNNs can reduce images (data grids) into a form which is easier to process without losing features that are critical for getting a good prediction.

Convolutional Neural Networks



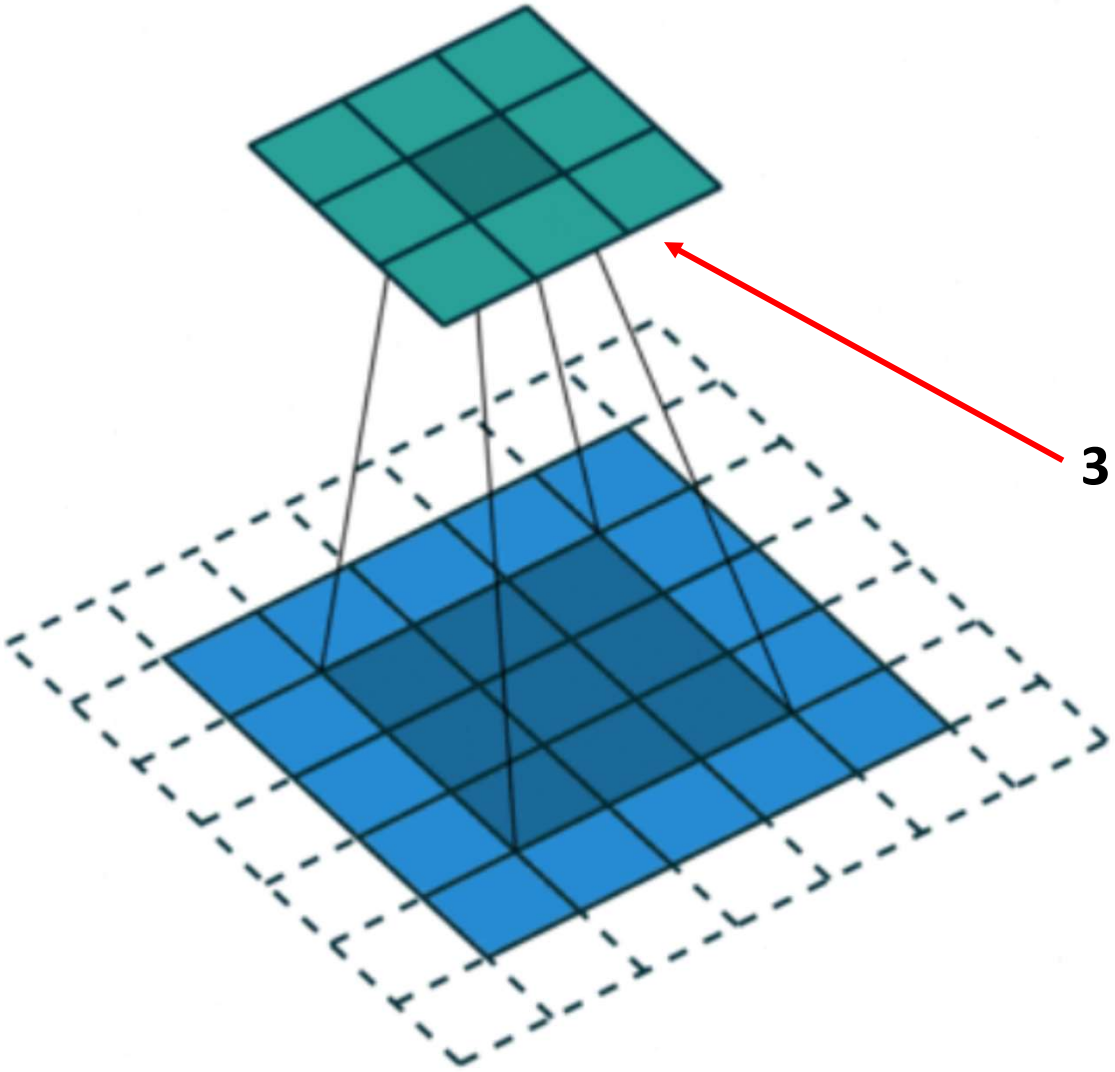
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

Convolution: The Idea



Source: https://commons.wikimedia.org/wiki/File:Convolutional_Neural_Network_NeuralNetworkFilter.gif

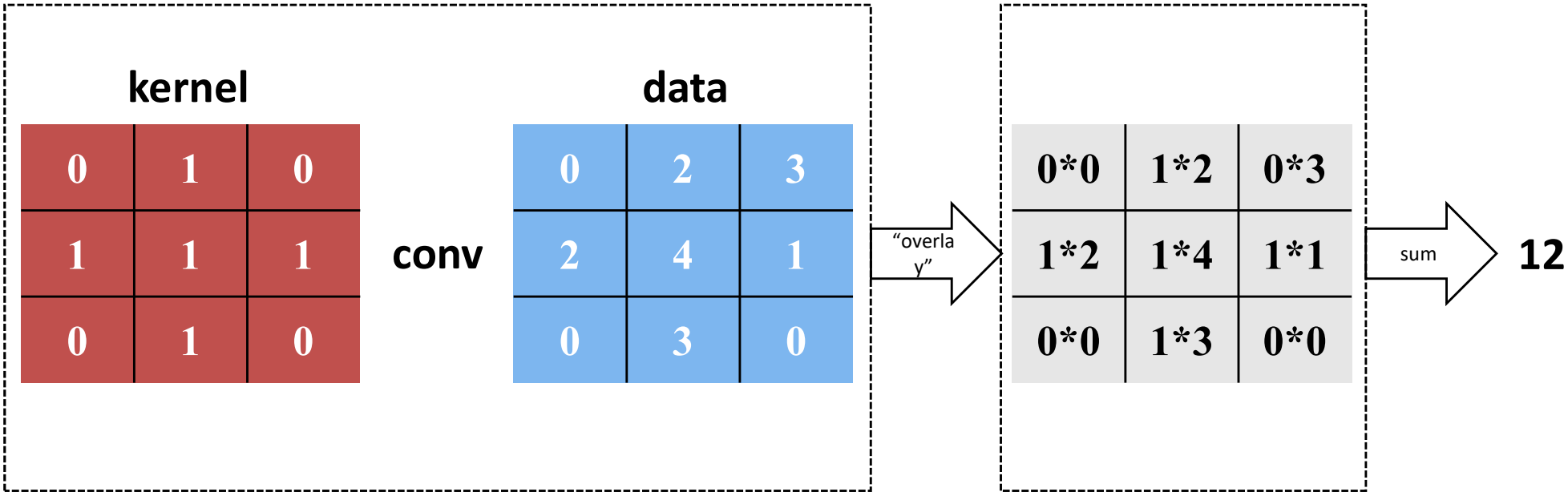
Kernel / Filter: The Idea



Source: https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_Padding_strides.gif

Convoluting Matrices

Convolution (and Convolutional Neural Networks) can be applied to any grid-like data (tensors: matrices, vectors, etc.).



Selected Image Processing Kernels

Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Mean Blur

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Gaussian Blur

$$\begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 1/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Laplacian

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Prewitt (Edge)

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Prewitt (Edge)

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

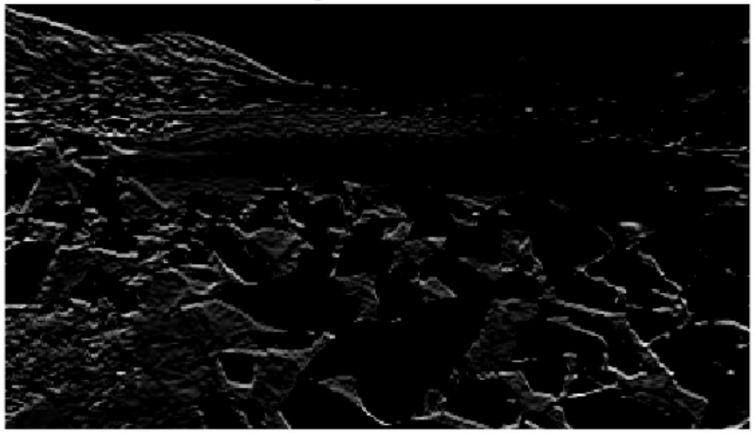
Image Processing: Kernels / Filters

Original

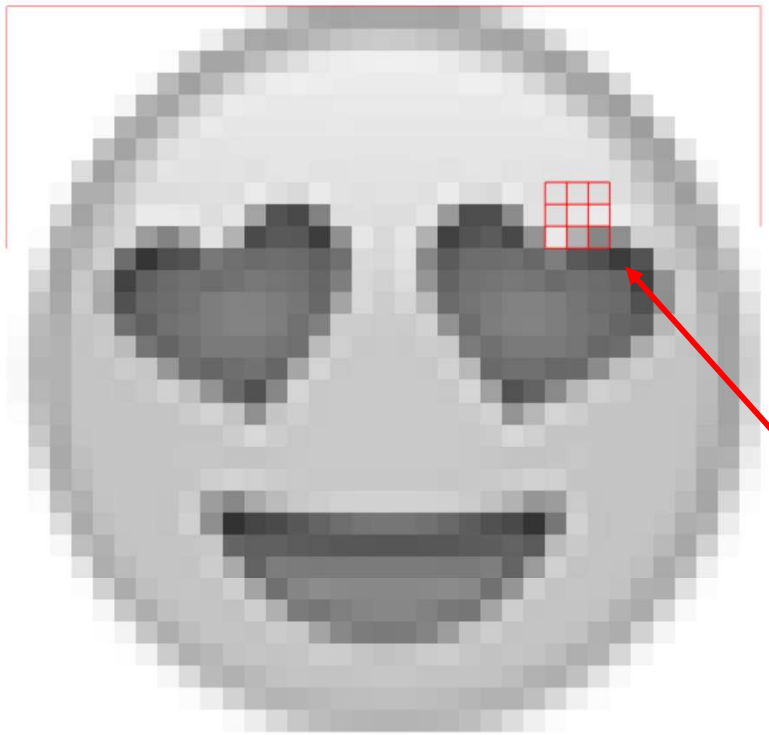
Sobel

Gaussian Blur

Edge detection



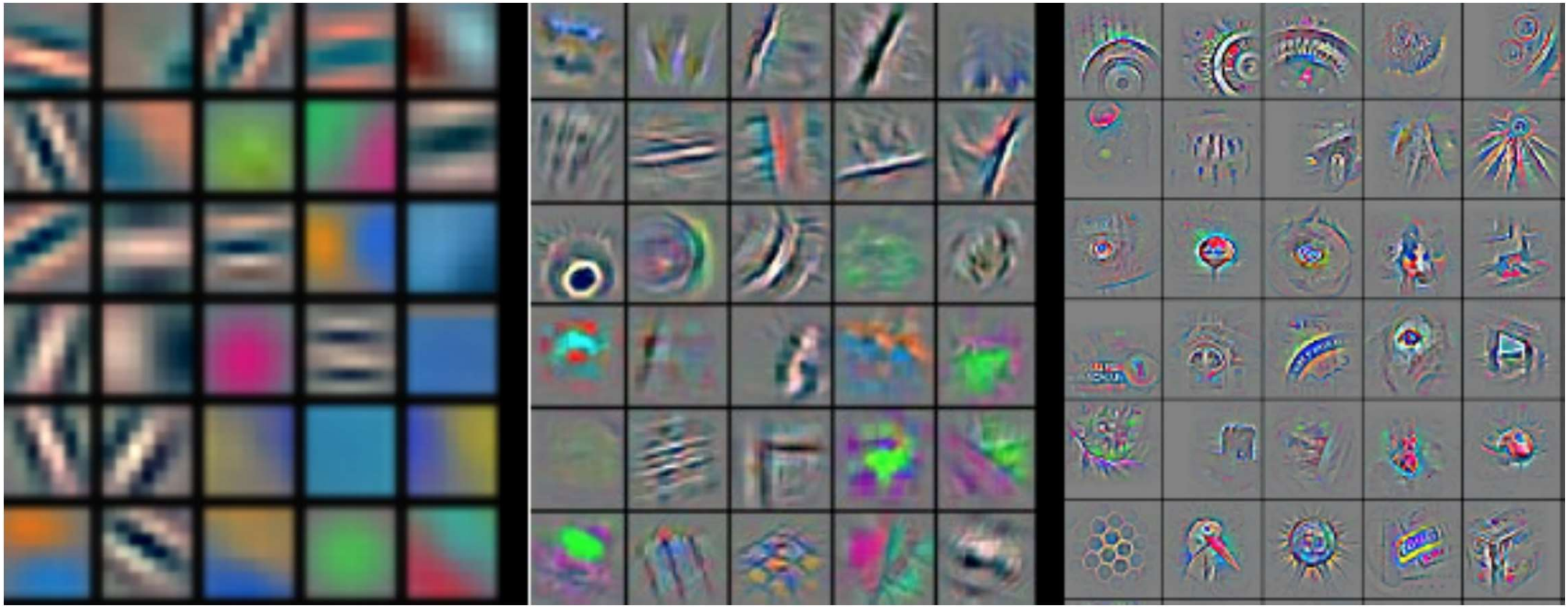
Applying Kernels / Filters



3 x 3 Kernel / Filter

Convolutional NN Kernels

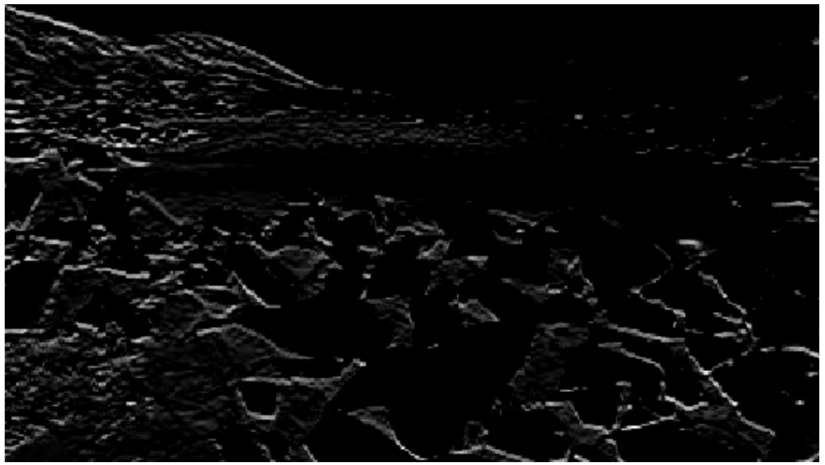
In practice, Convolutional Neural Network kernels can be larger than 3x3 and are learned using back propagation.



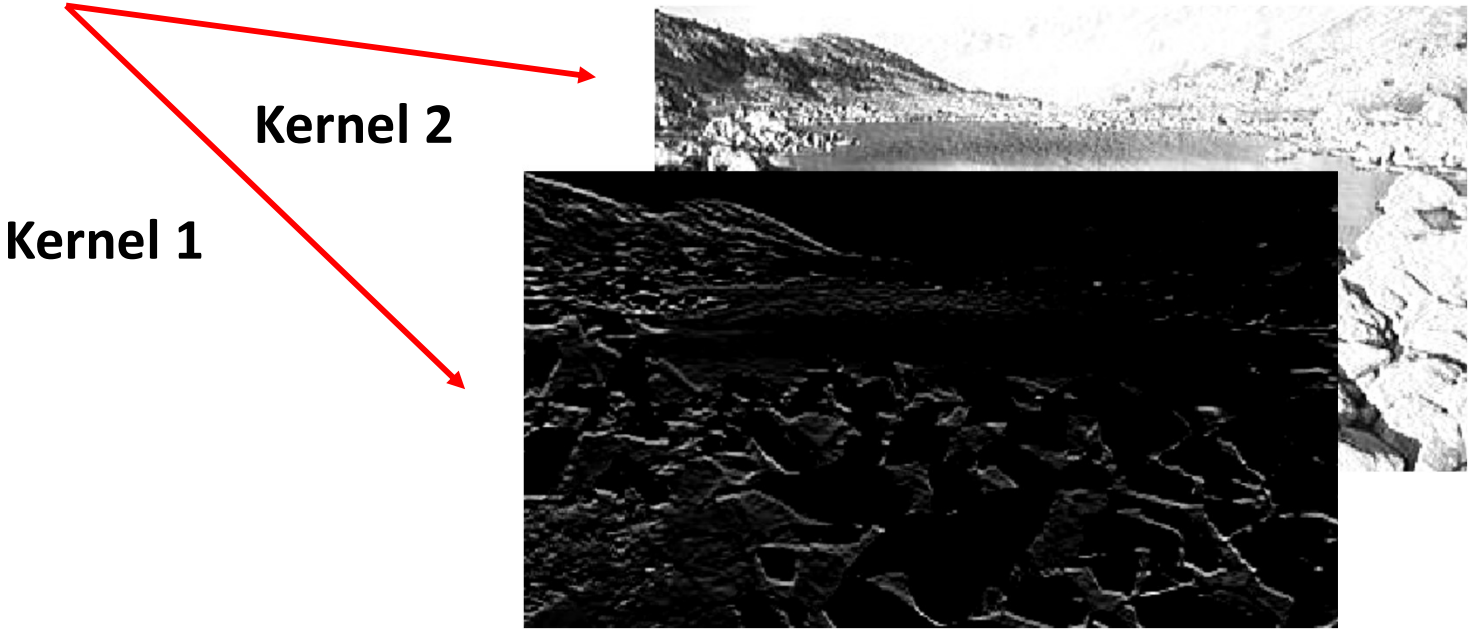
Convolution Layer 1 | Convolution Layer 2 | Convolution Layer 3

Convolution Layer 1

Kernel 1



Convolution Layer 1



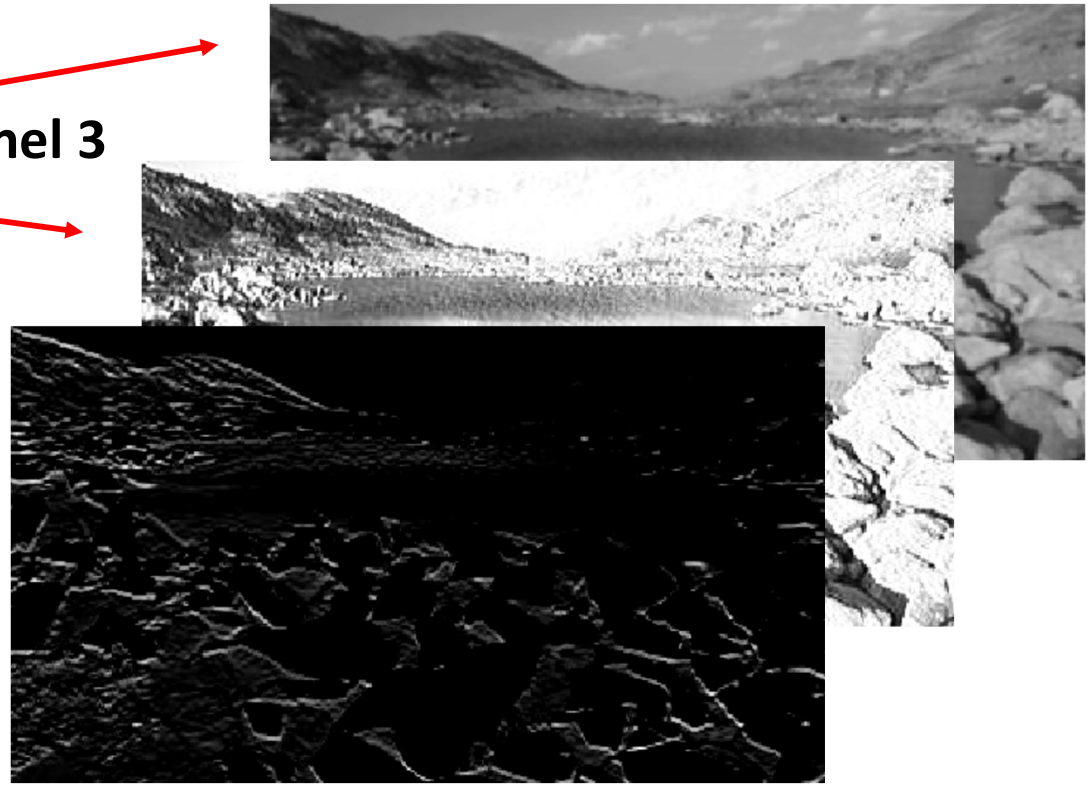
Convolution Layer 1

Original image

Kernel 3

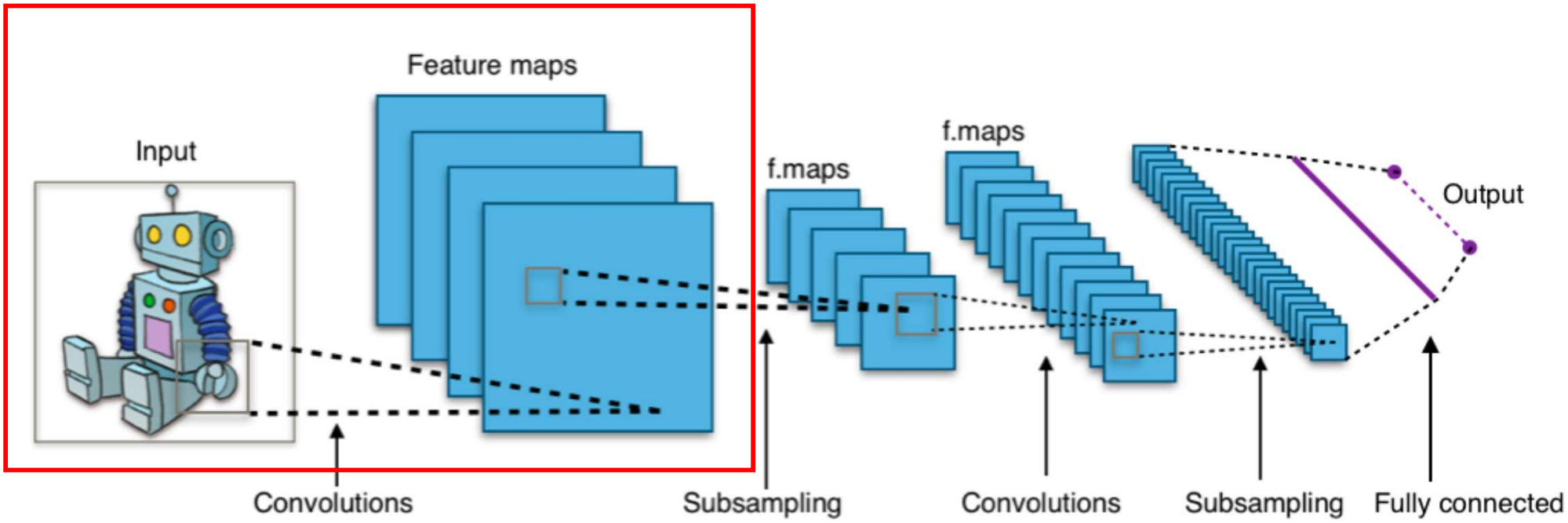
Kernel 2

Kernel 1



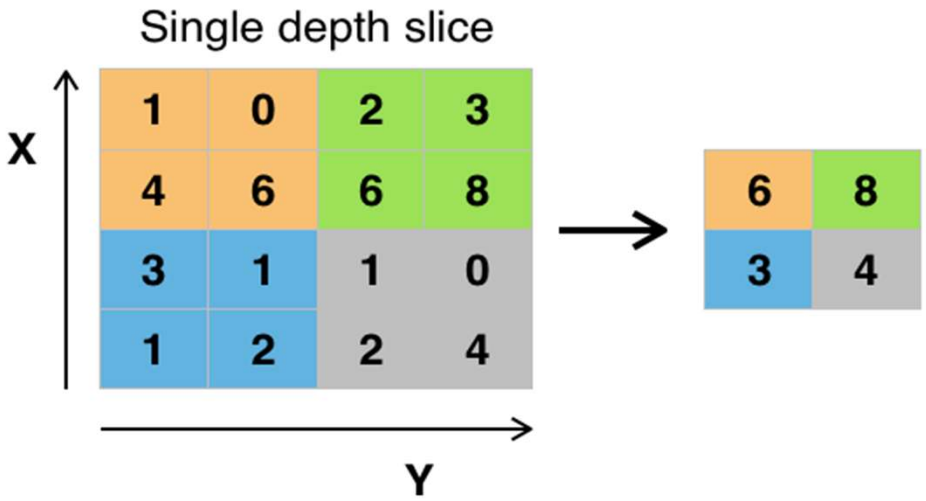
Convolution 1

Convolutional Neural Networks

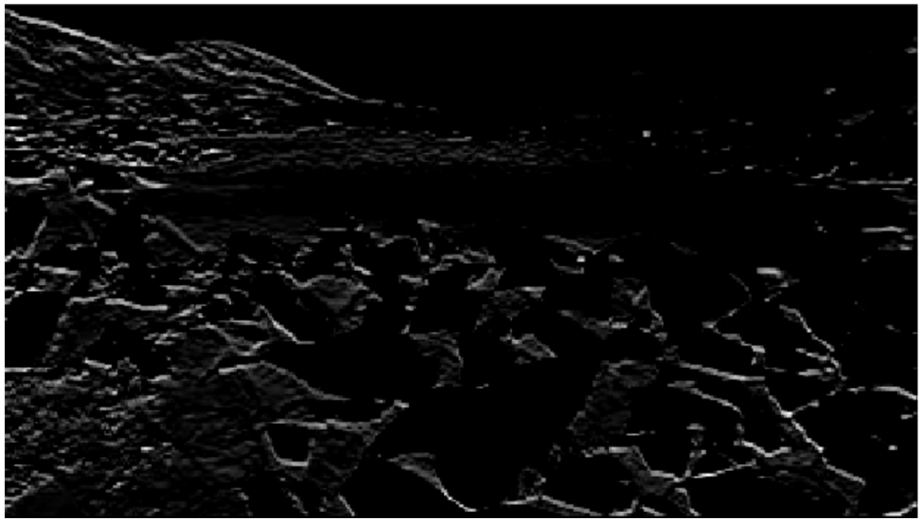


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

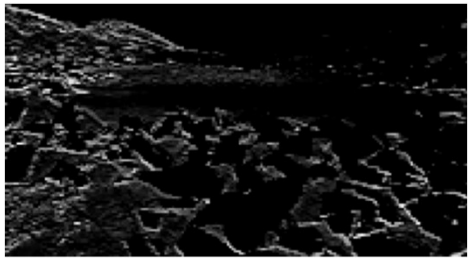
Max Pooling Layer



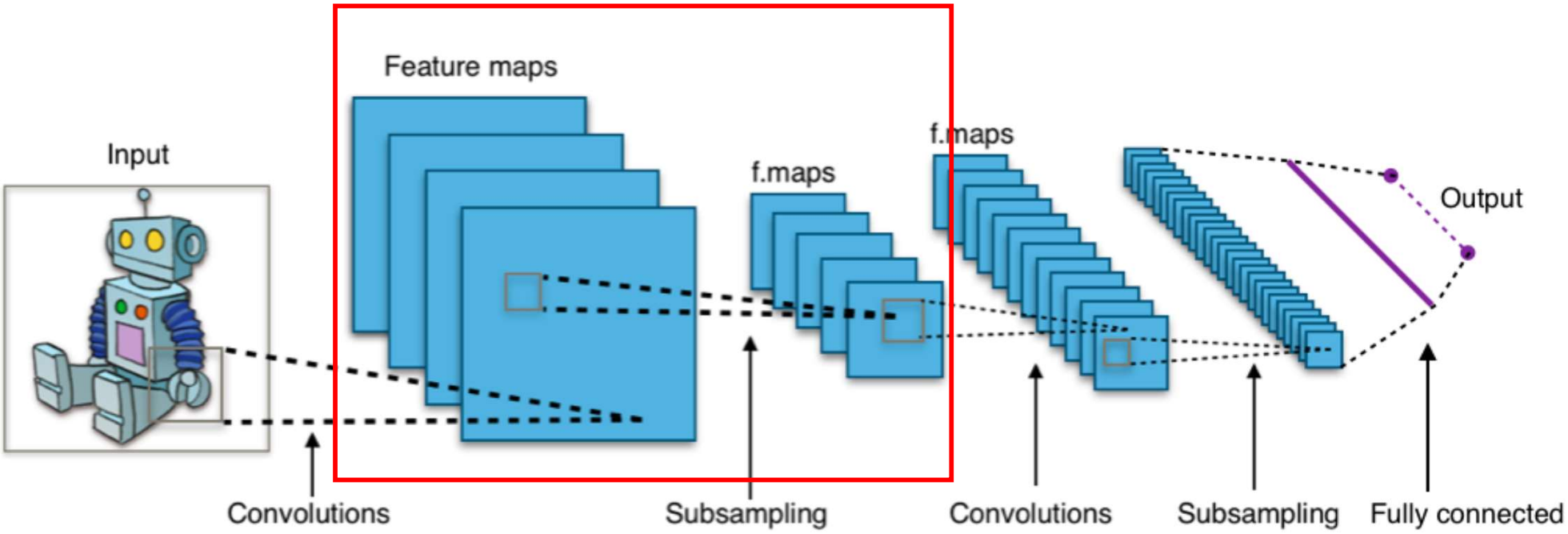
Convolution 1



Max Pooling

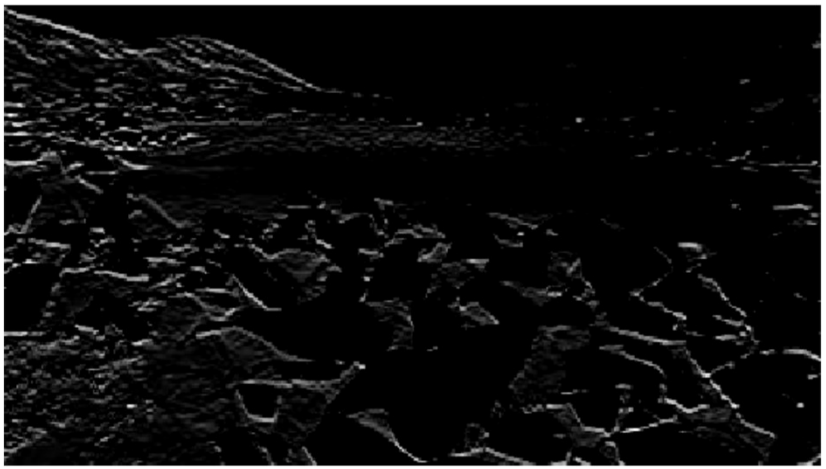


Convolutional Neural Networks



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

Convolution Layer 2

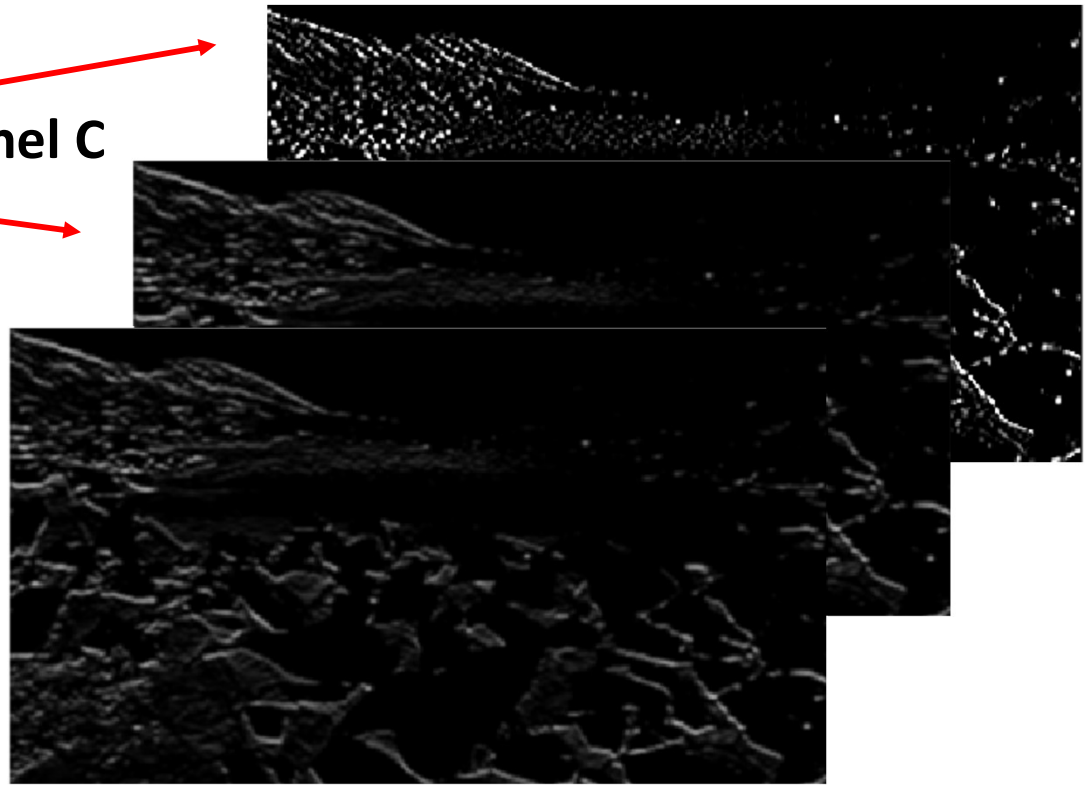


Original convolution after pooling

Kernel C

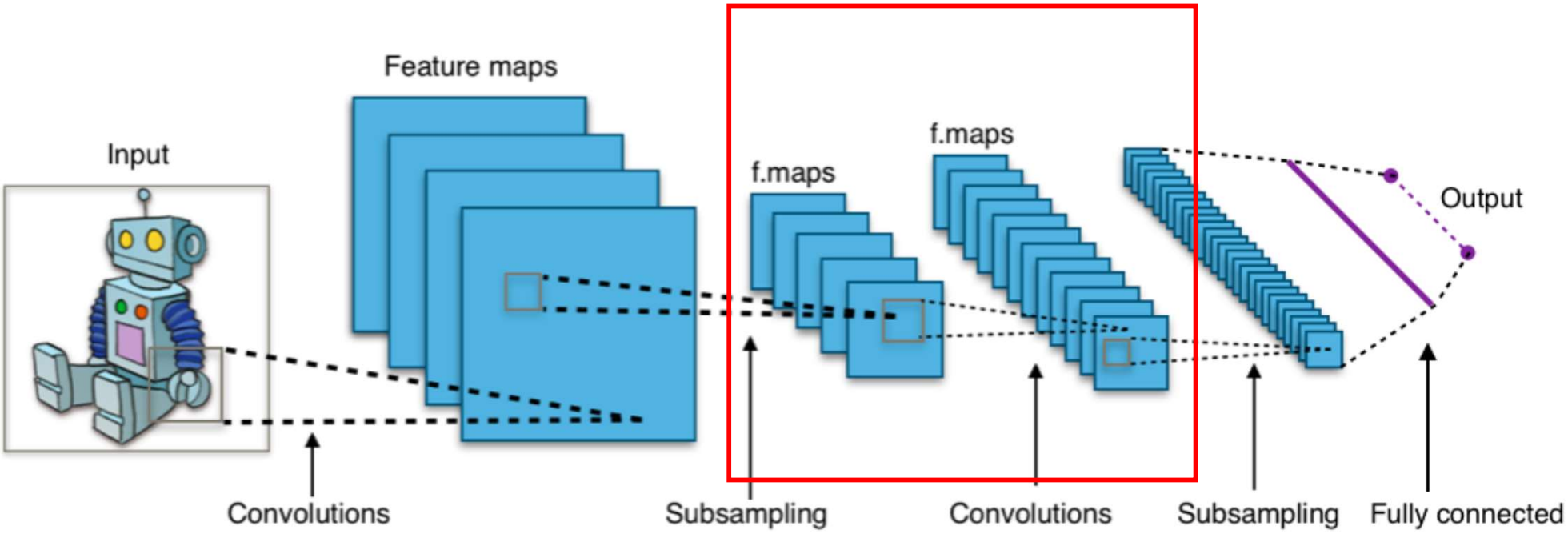
Kernel B

Kernel A



Convolution A

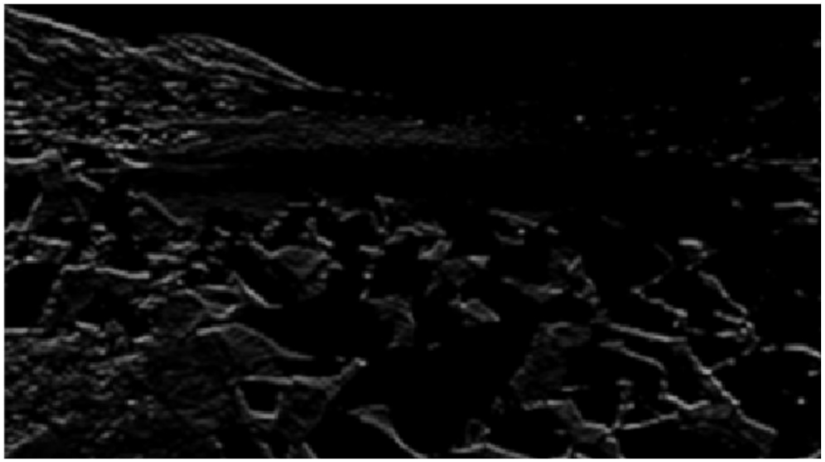
Convolutional Neural Networks



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

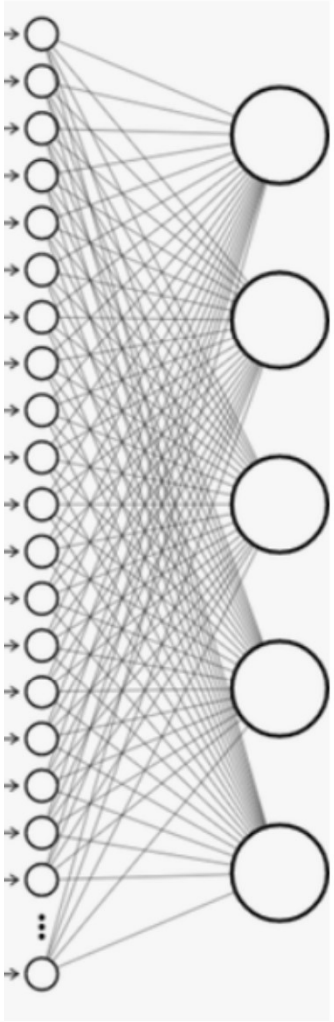
Flattening

Final output of convolution layers is "**flattened**" to become **a vector of features**.



Final convolution layer output

Convert to vector

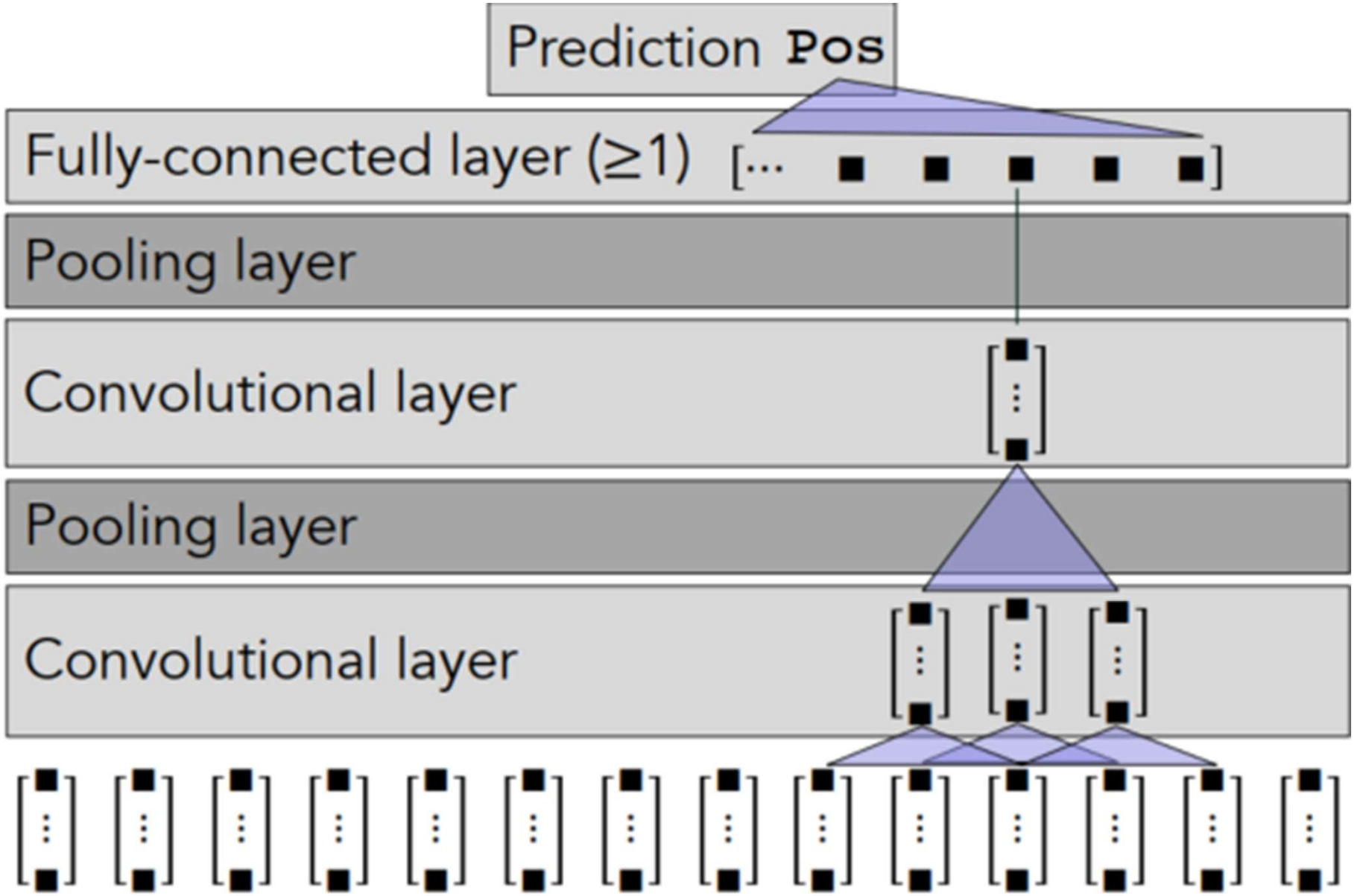


Source: https://nikolanews.com/not-just-introduction-to-convolutional-neural-networks-part-1/

Convolutions for Text

- Convolutions are local feature extractors
  - In vision, detection of edges, corners, facial features, …
  - In NLP, detection of negation, tense, local syntactic features, …
- If word embeddings learn good representations for words, convolutions learn good higher-level representations for making predictions

CNNs for Text Classification/Prediction



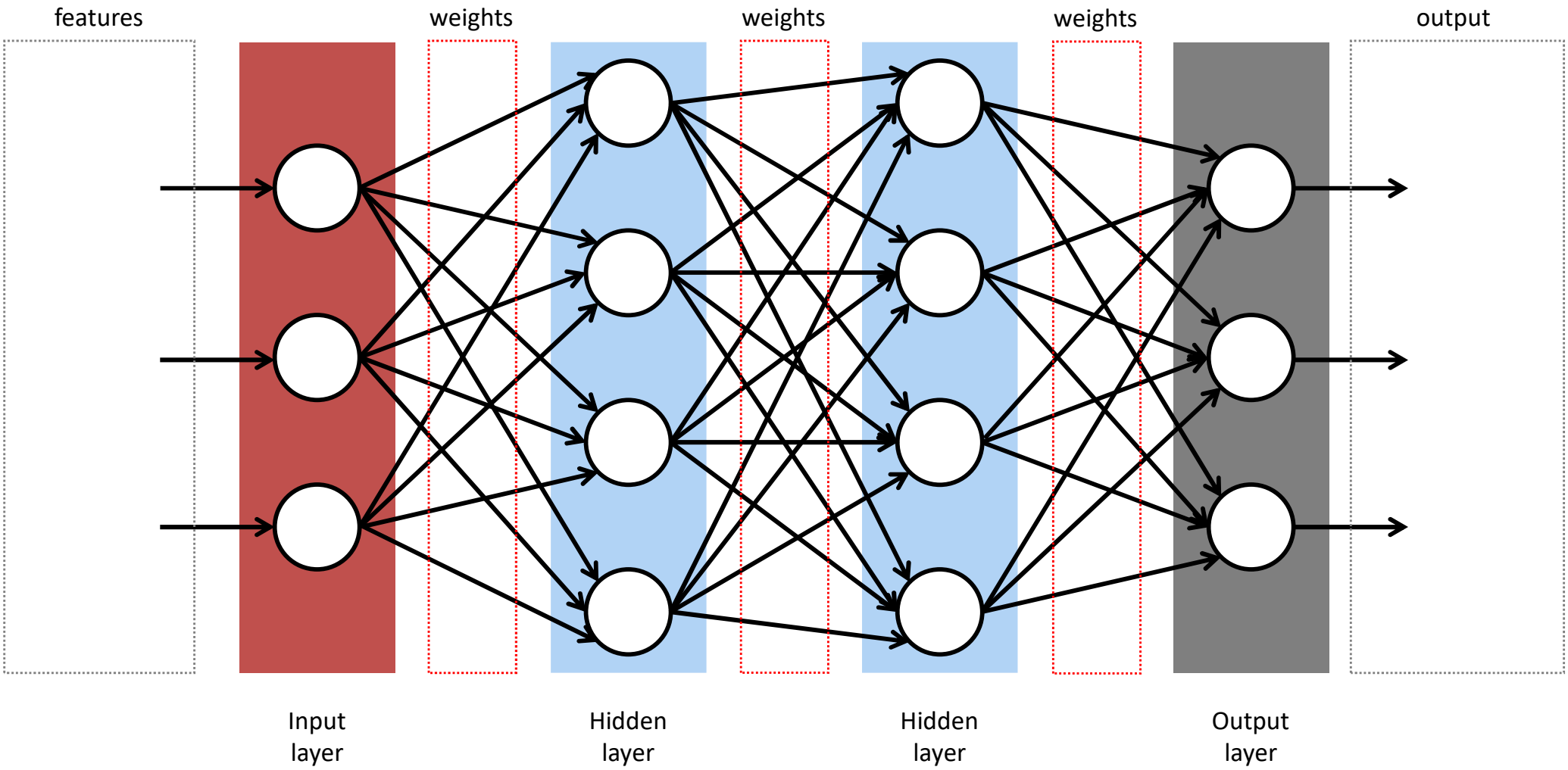
CNNs for Text Classification

- In a convolutional model, we can use pooling operations to aggregate features across the entire sentence / text

- And then use this representation as an input to a standard feed-forward neural network for text categorization

Recurrent Neural Networks

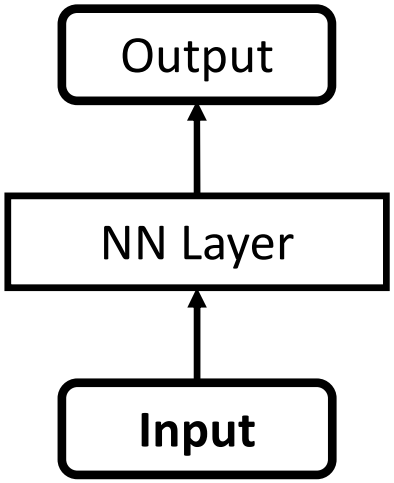
Feedforward Neural Network



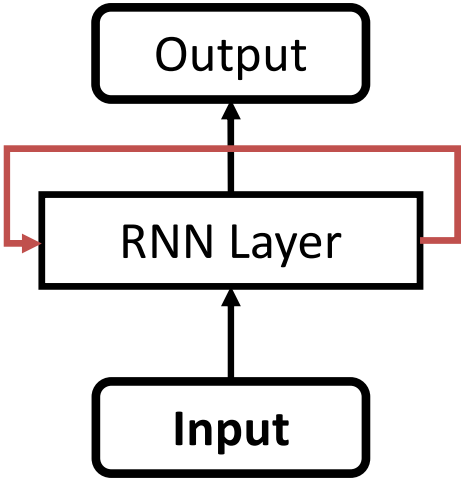
Also called (historically): multi-layer perceptron

Regular vs. Recurrent NNs

Regular NN

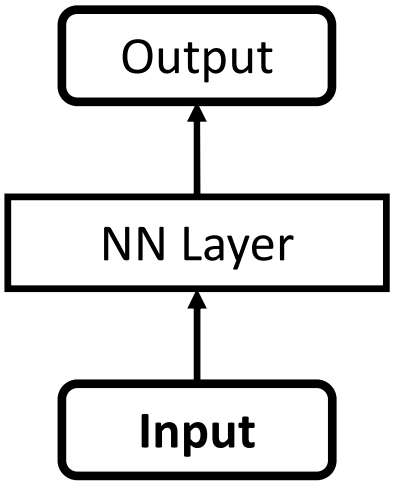


Recurrent NN



Regular vs. Recurrent NNs

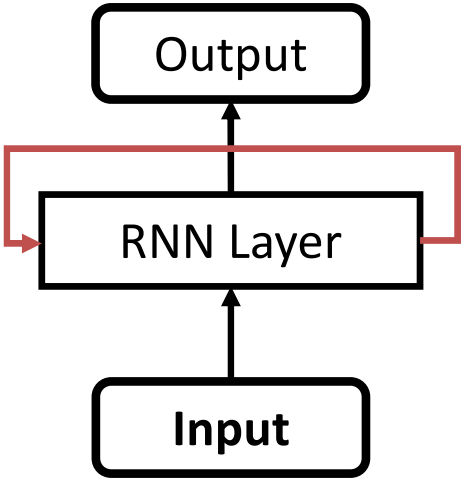
Regular NN



Does NOT have memory (does not "remember" previous state/input)

NOT suitable for sequential data

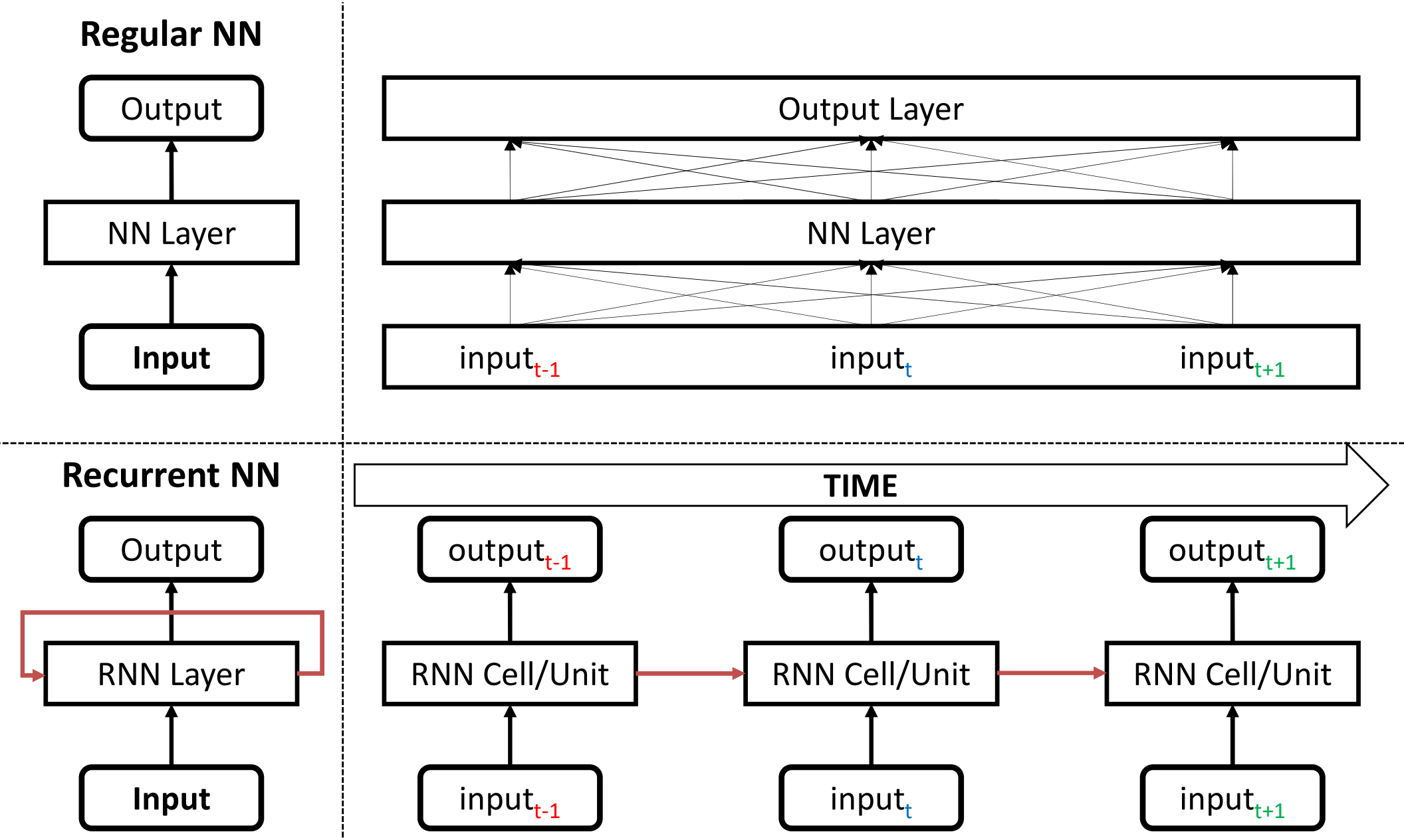
Recurrent NN



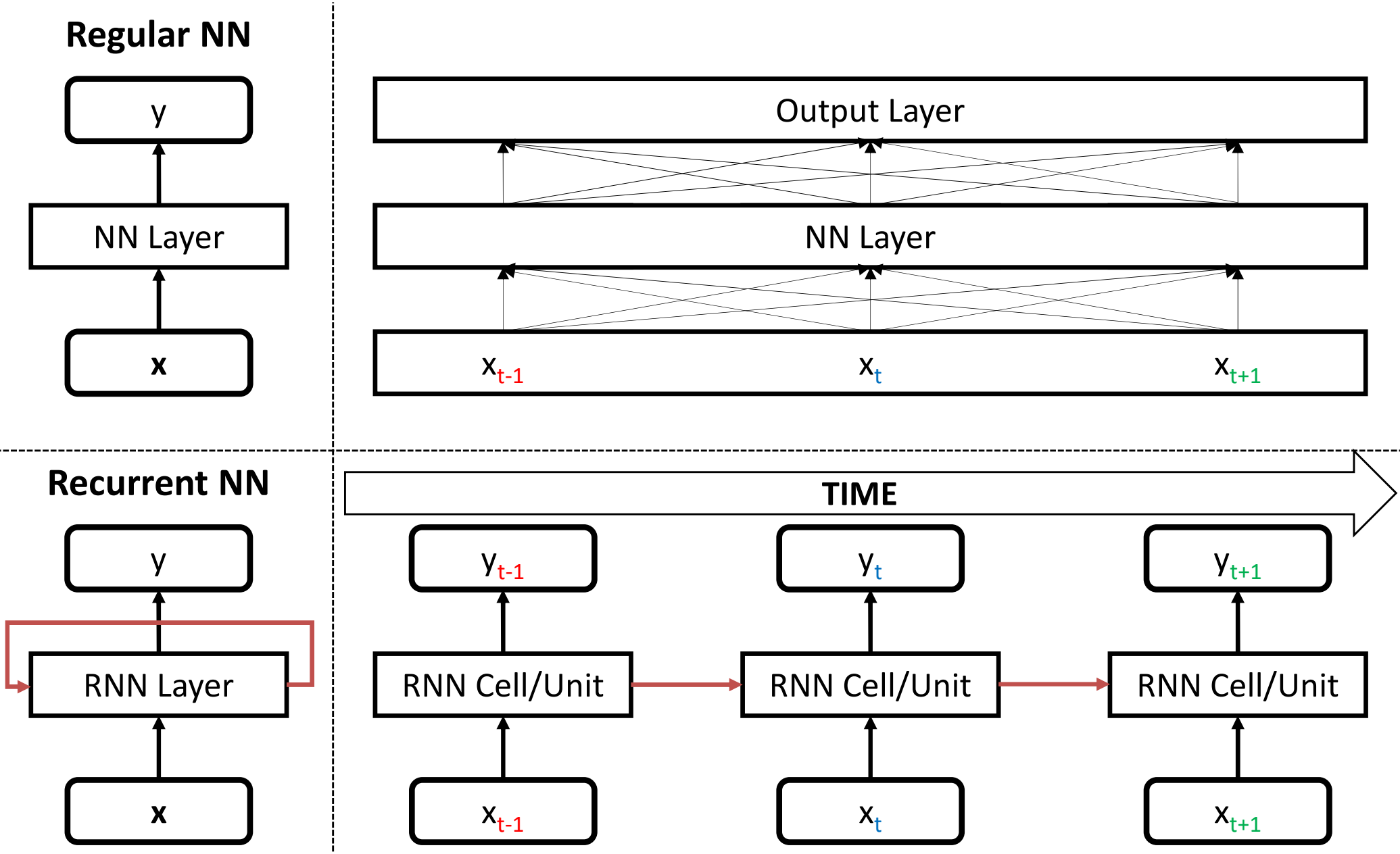
Does HAVE memory ("remembers" previous state/input)

Suitable for sequential data

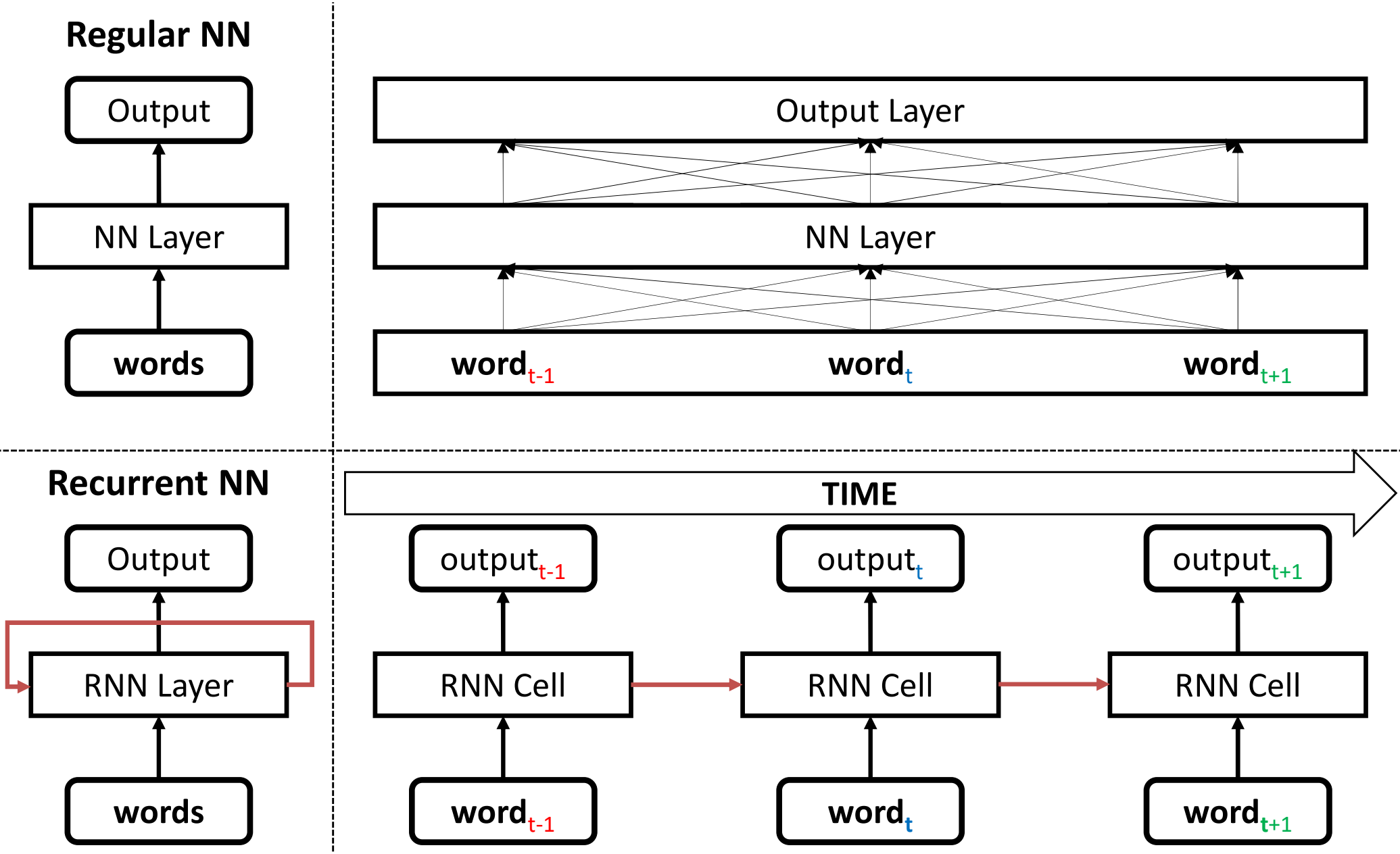
Regular vs. Recurrent NNs



Regular vs. Recurrent NNs

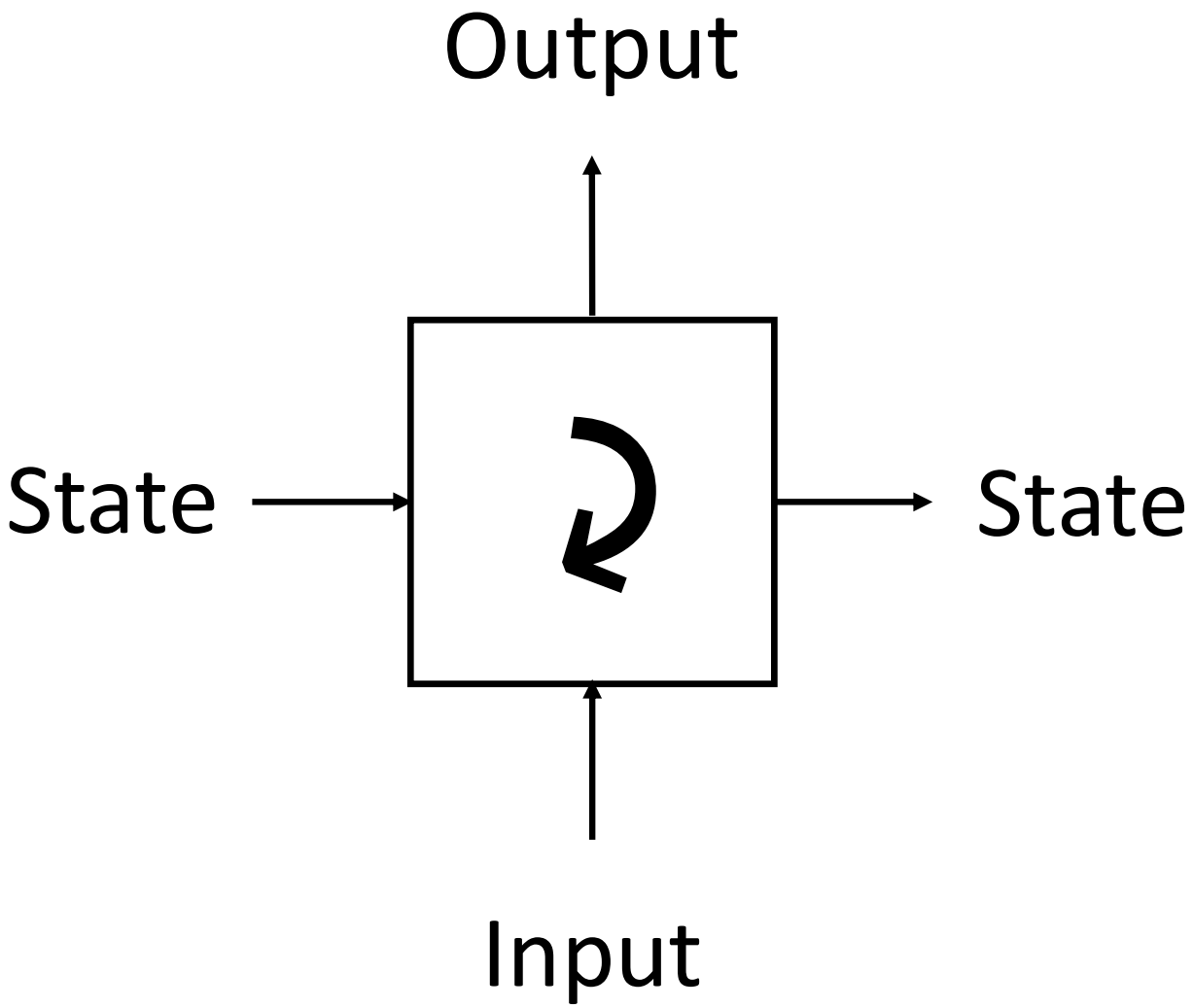


Regular vs. Recurrent NNs

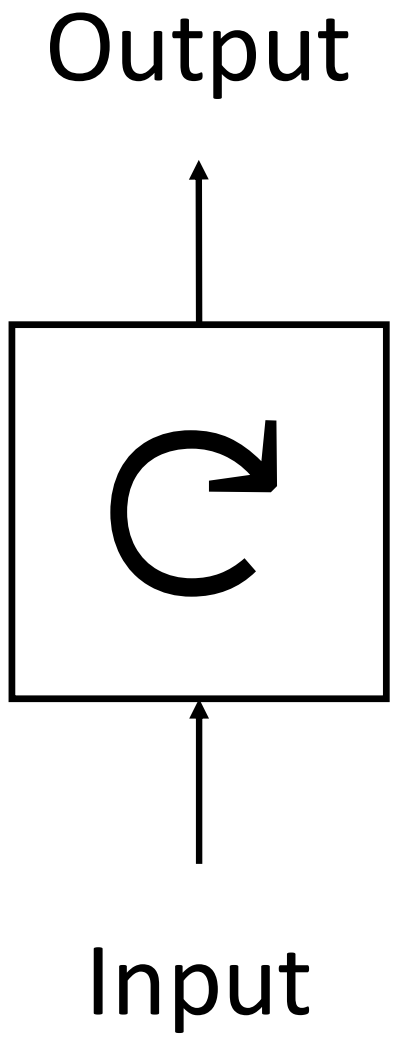


Recurrent Cells/Layers: Symbols

Recurrent Cell/Unit

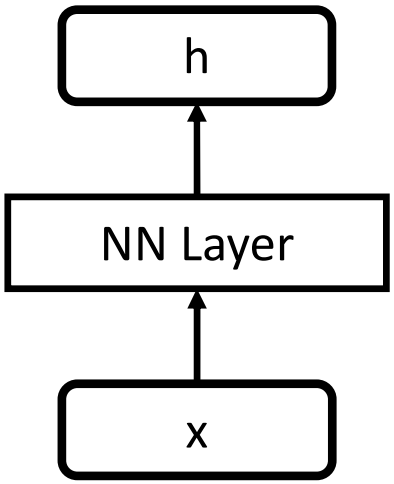


Recurrent Layer



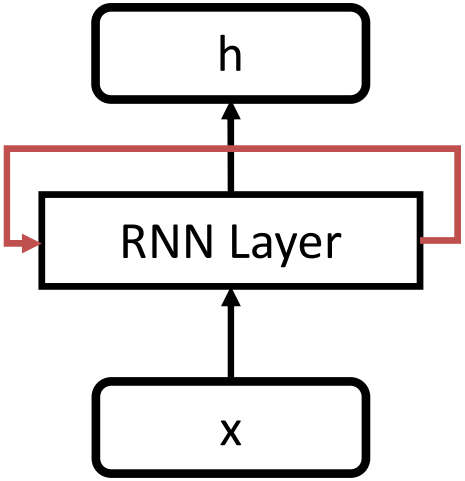
Regular vs. Recurrent NNs

Regular NN



Easier to parallelize

Recurrent NN



Difficult to parallelize

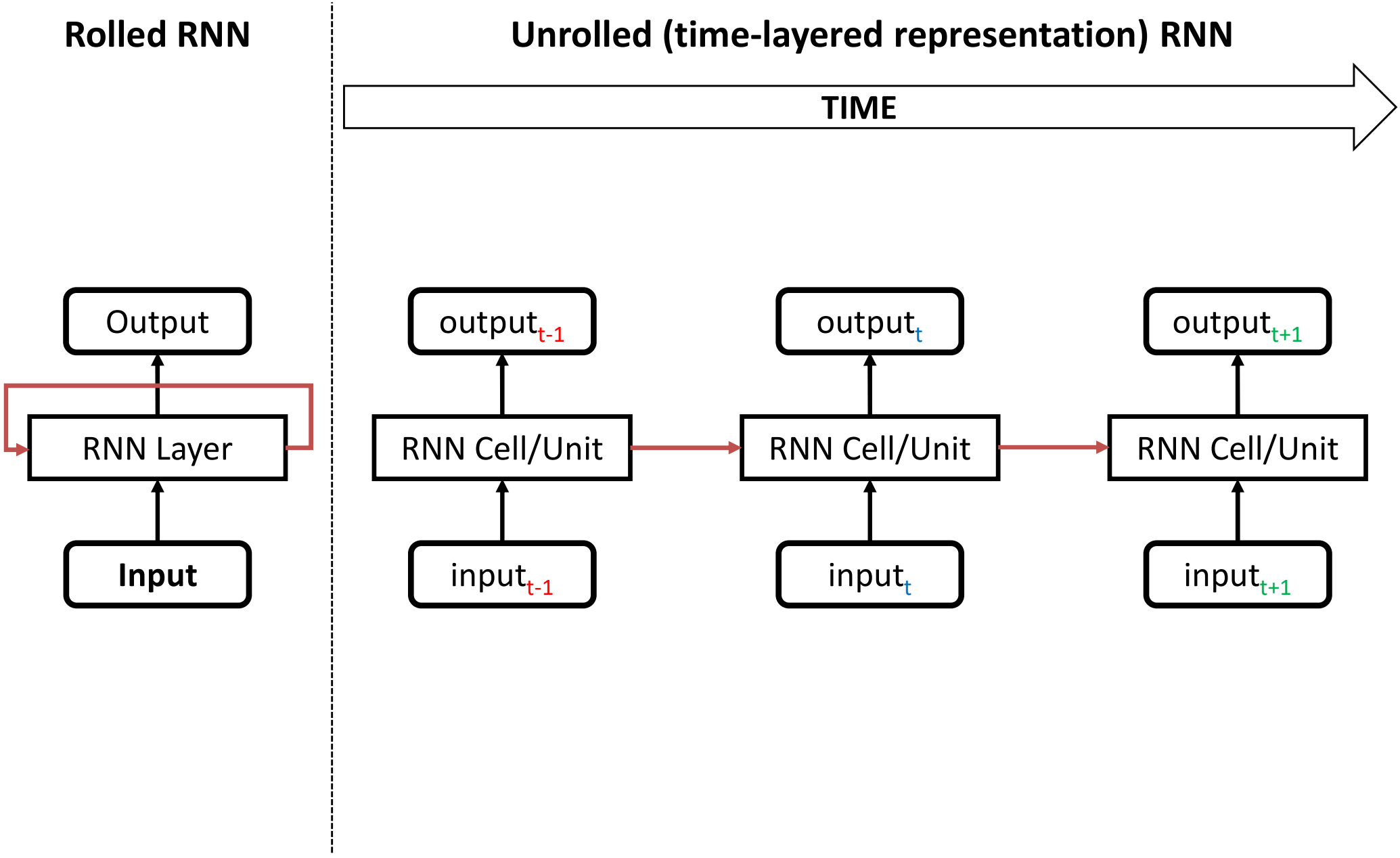
Recurrent Neural Networks (RNNs)

- Recurrent neural networks apply the same operation to the input at each time step, producing an output, but also updating an internal memory state that encodes relevant history to be used in prediction

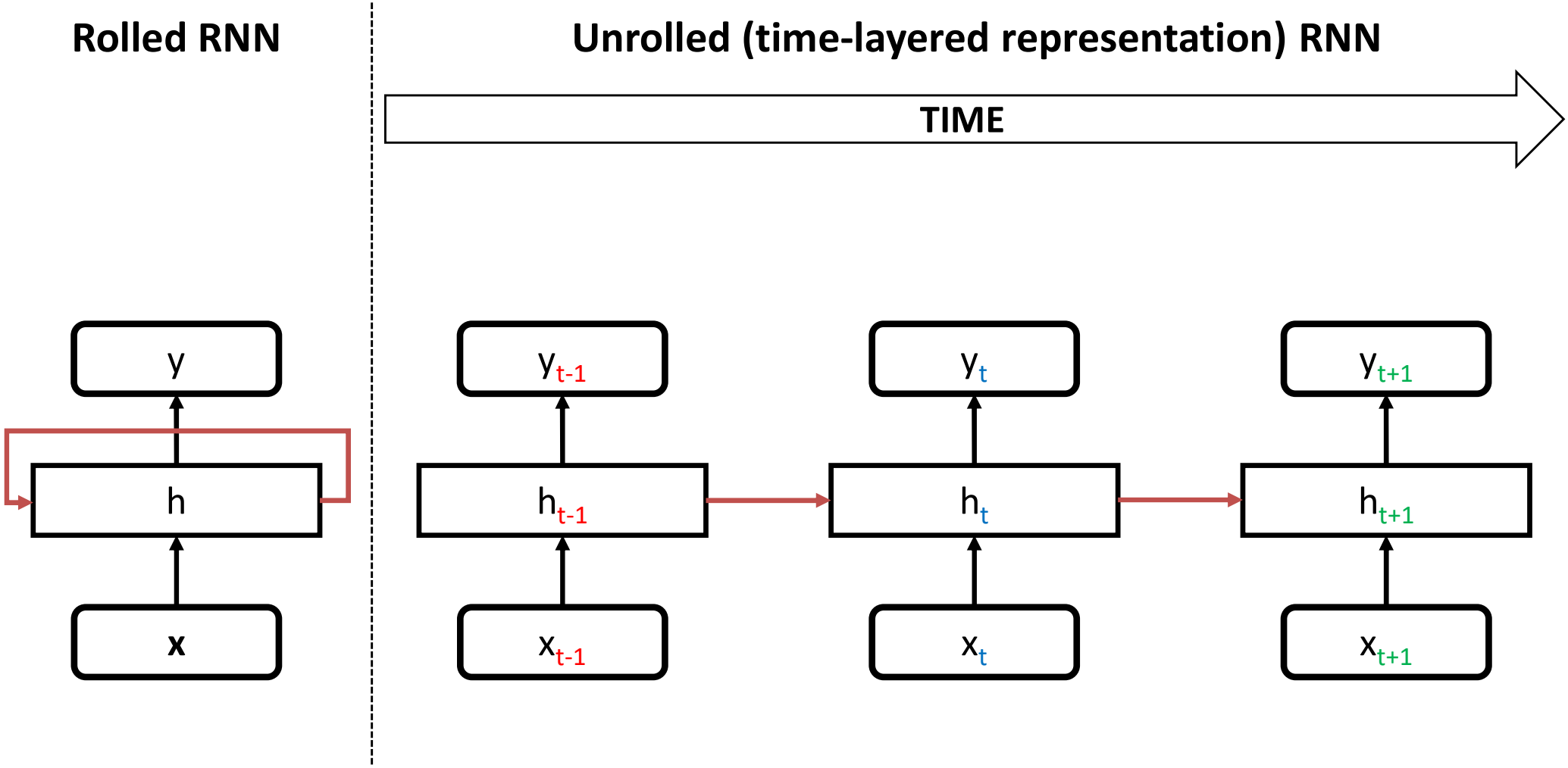
- This memory state can allow distant information to influence the prediction made for a given word/label

- Because the memory state is transferred from time step to time step, the network is intrinsically sequential – it cannot be effectively parallelized

Rolled vs. Unrolled RNN



Rolled vs. Unrolled RNN

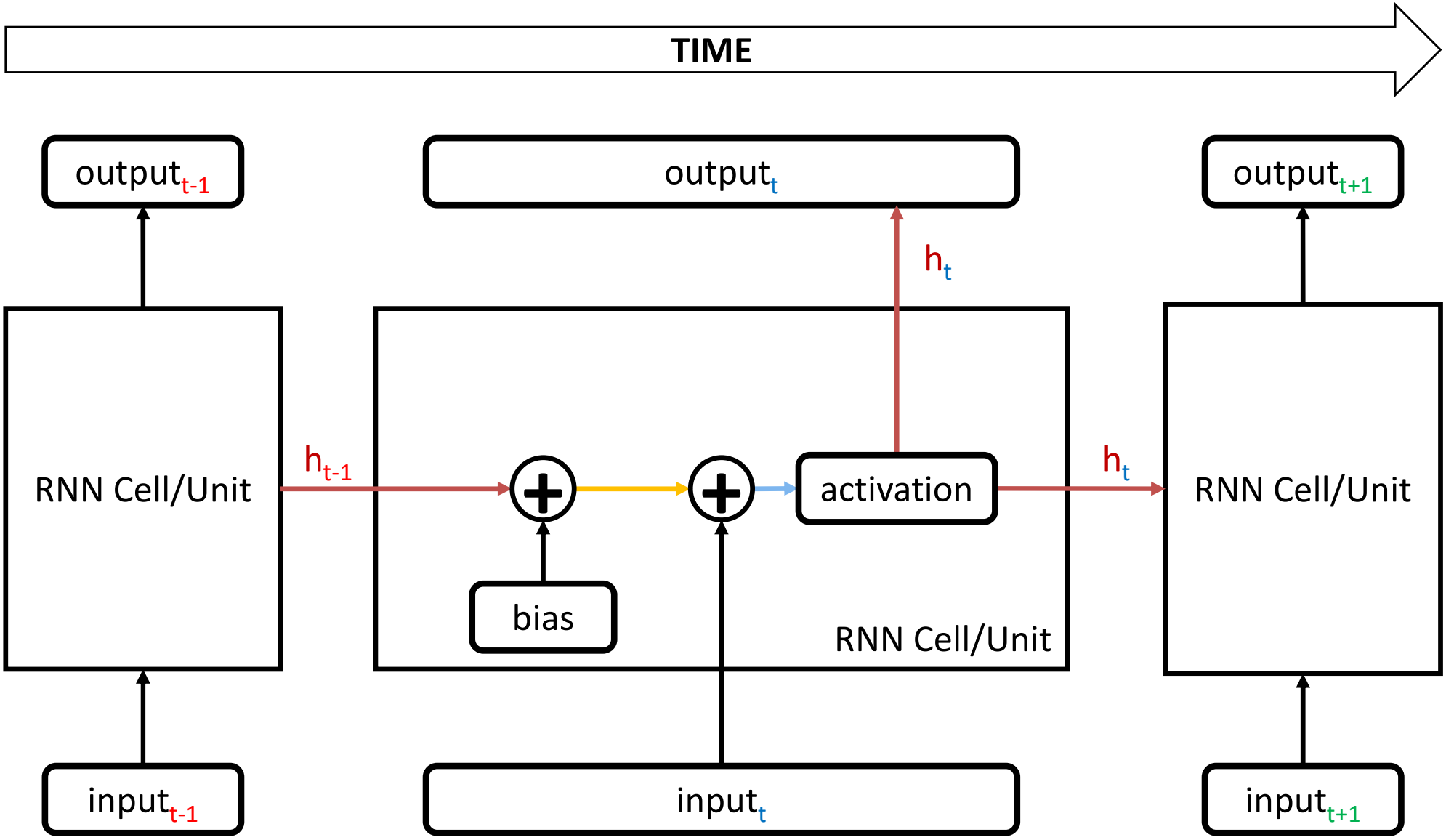


x_i – single input/feature (can be a scalar or a **vector**)

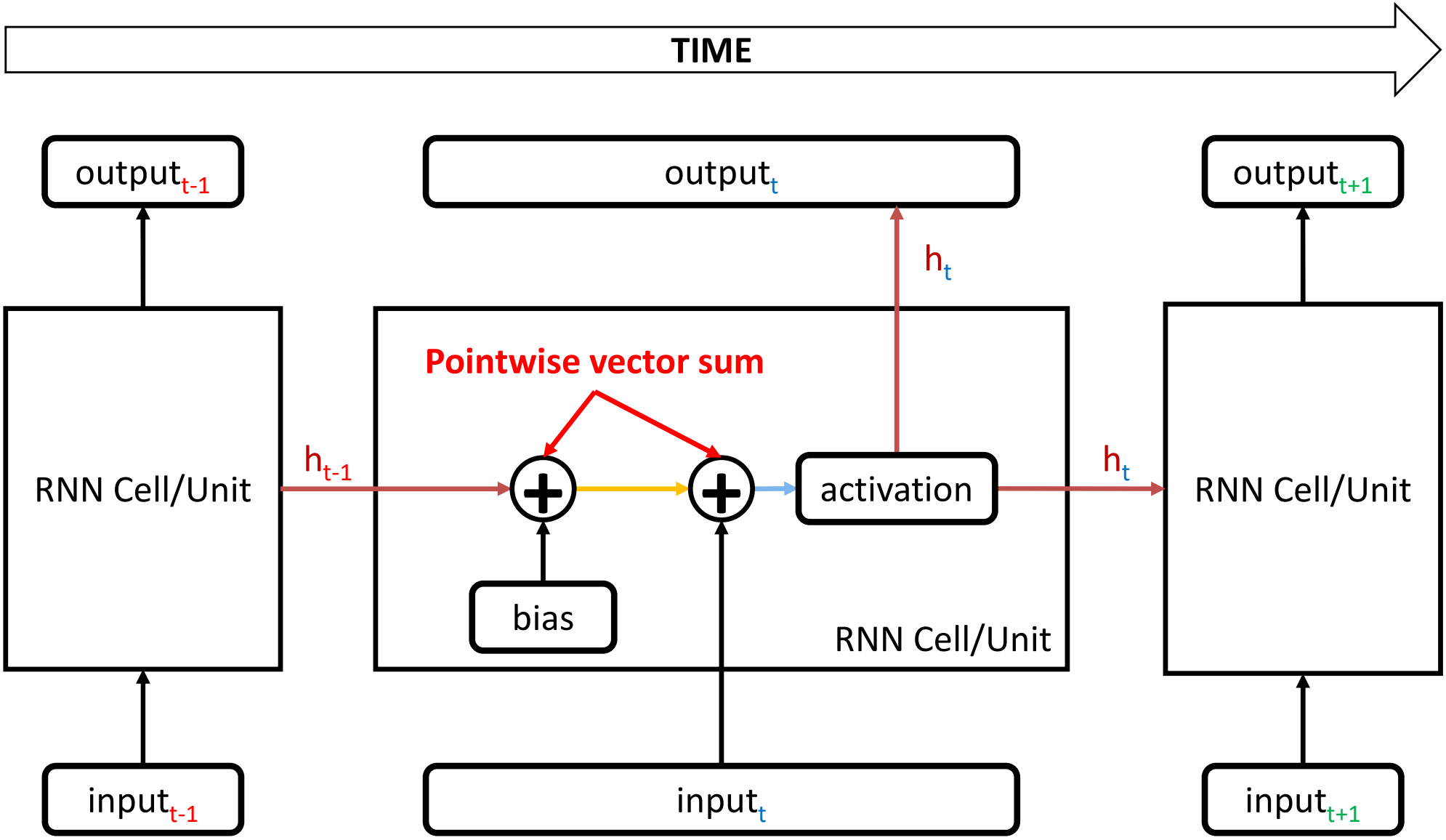
h_i – single memory/hidden representation/state (can be a scalar or a **vector**)

y_i – single output (can be a scalar or a **vector**)

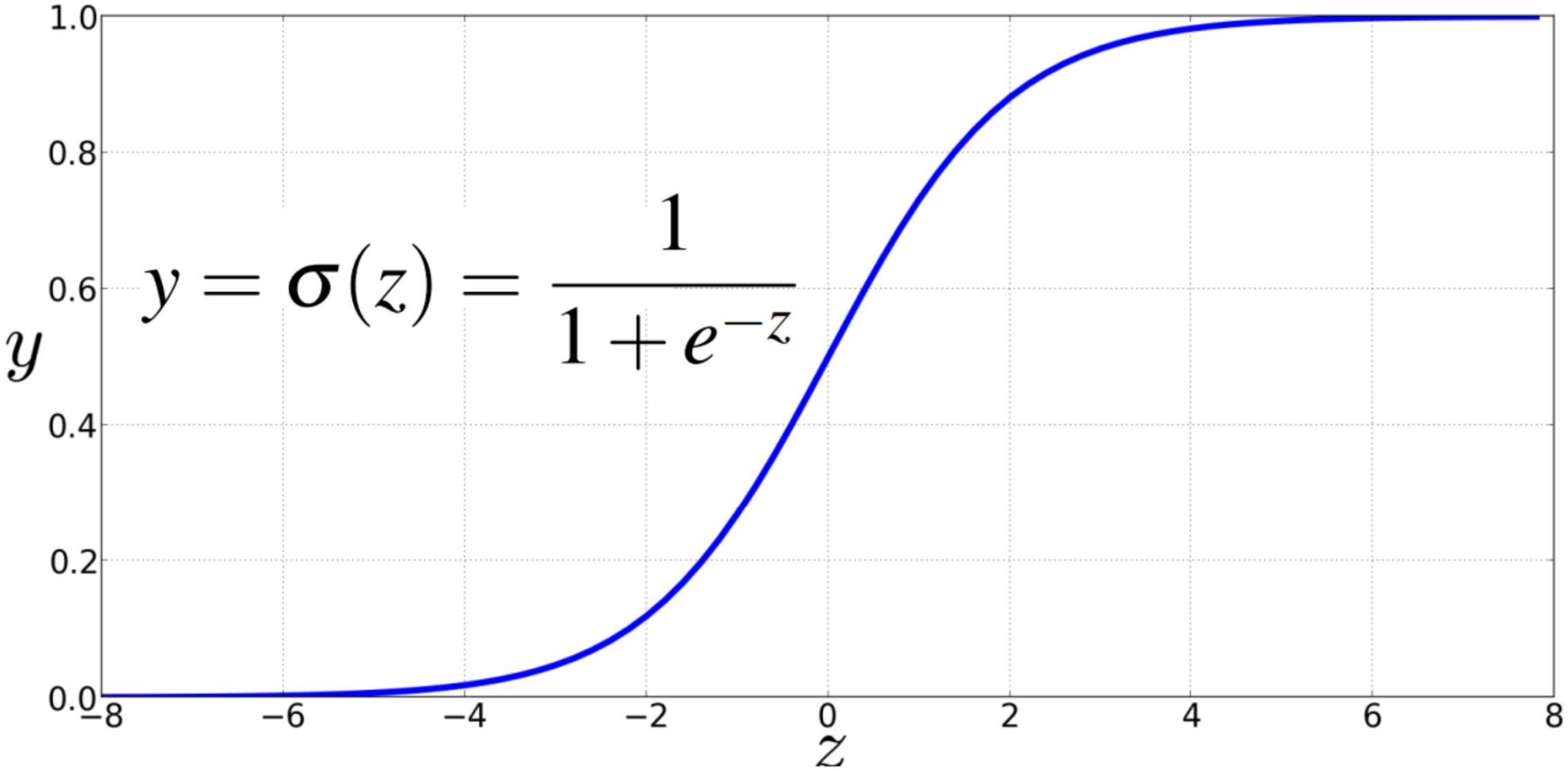
RNN Cell/Unit



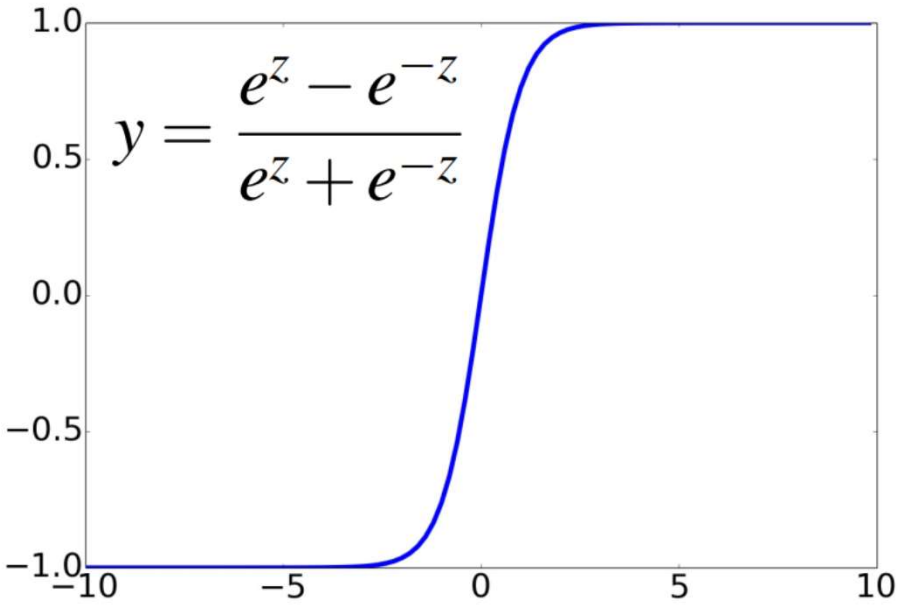
RNN Cell/Unit



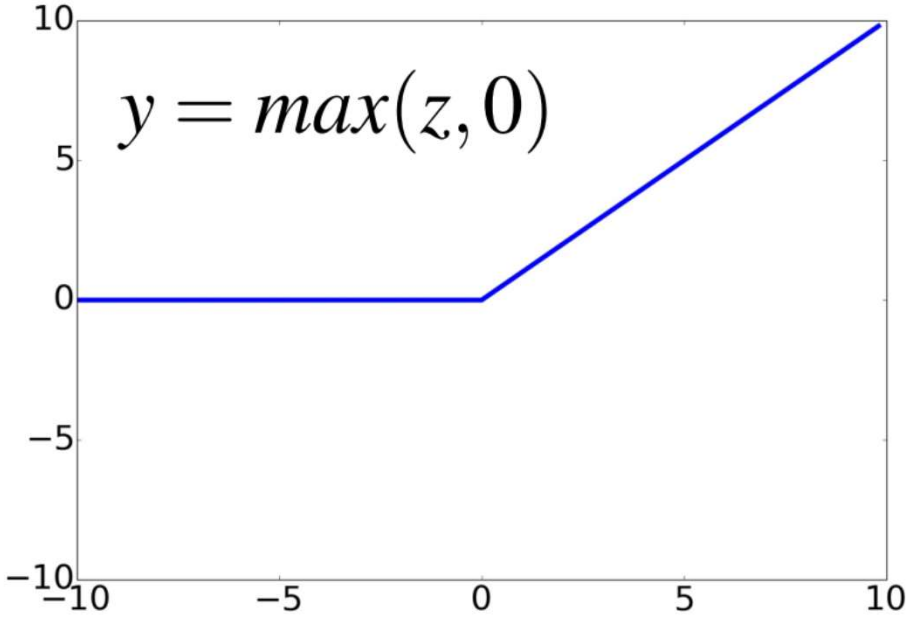
Sigmoid / Logistic Activation Function



Other Nonlinear Activation Functions

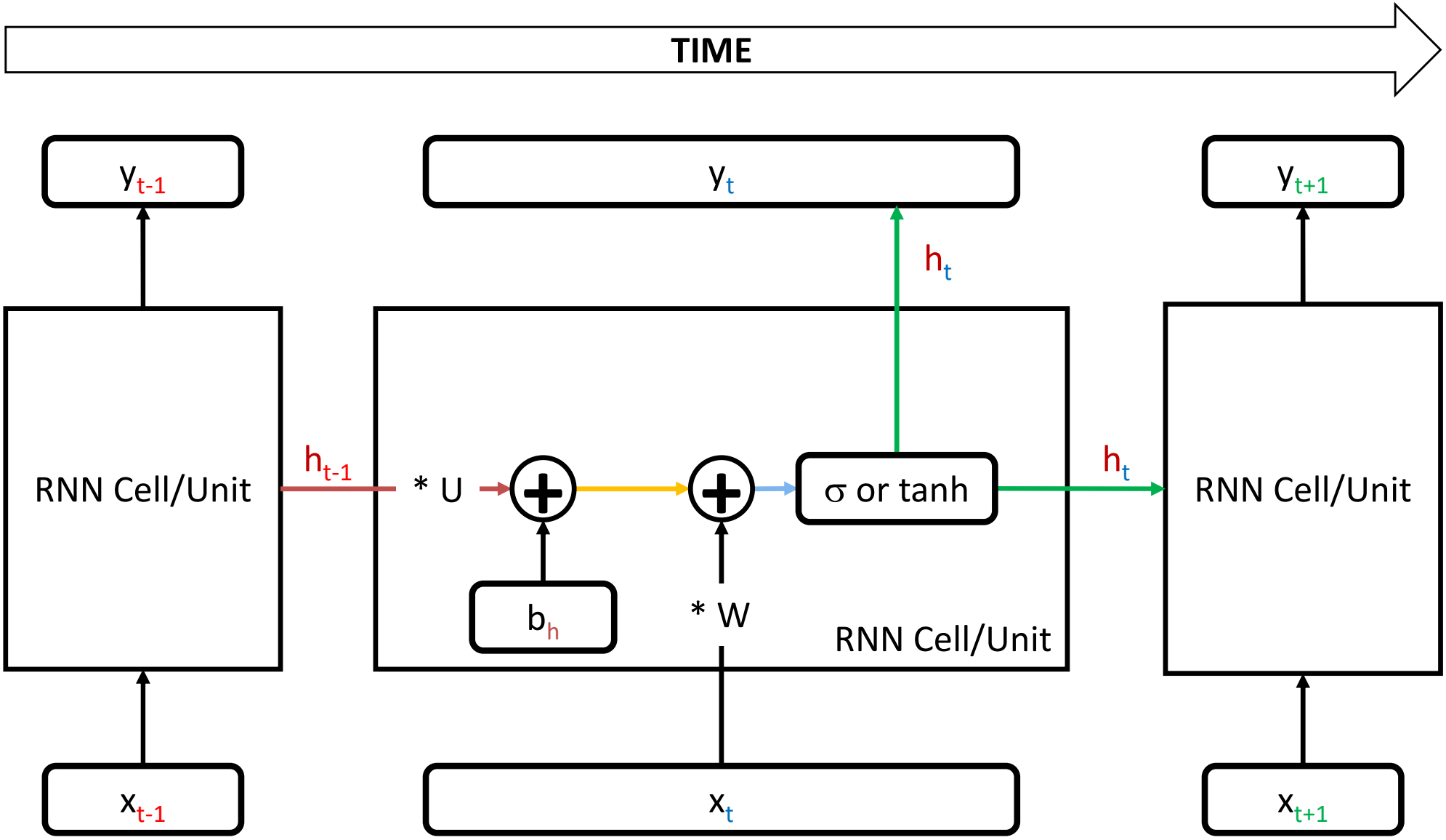


tanh
(hyperbolic tangent)

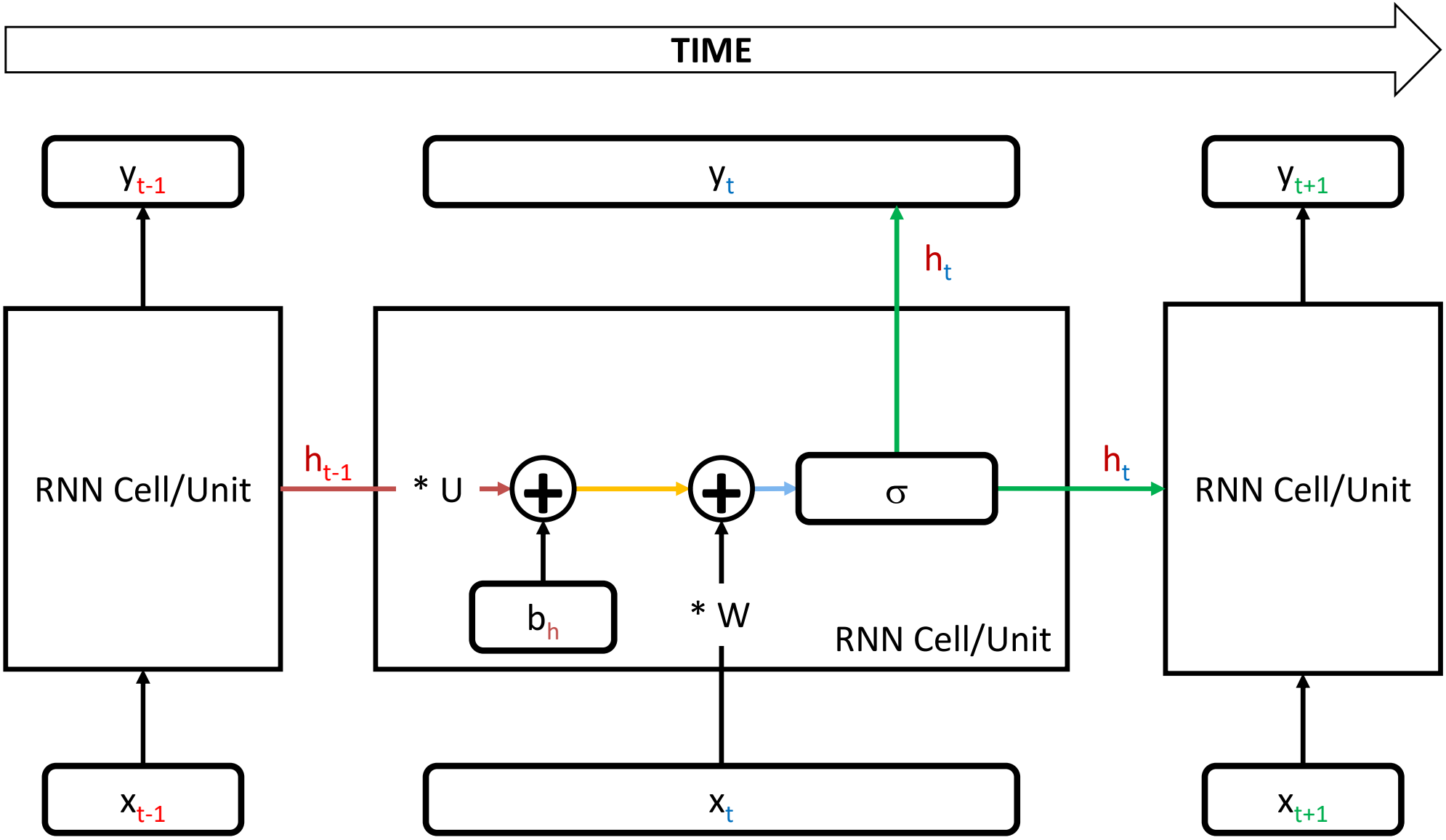


ReLU
(Rectified Linear Unit)

RNN Cell/Unit

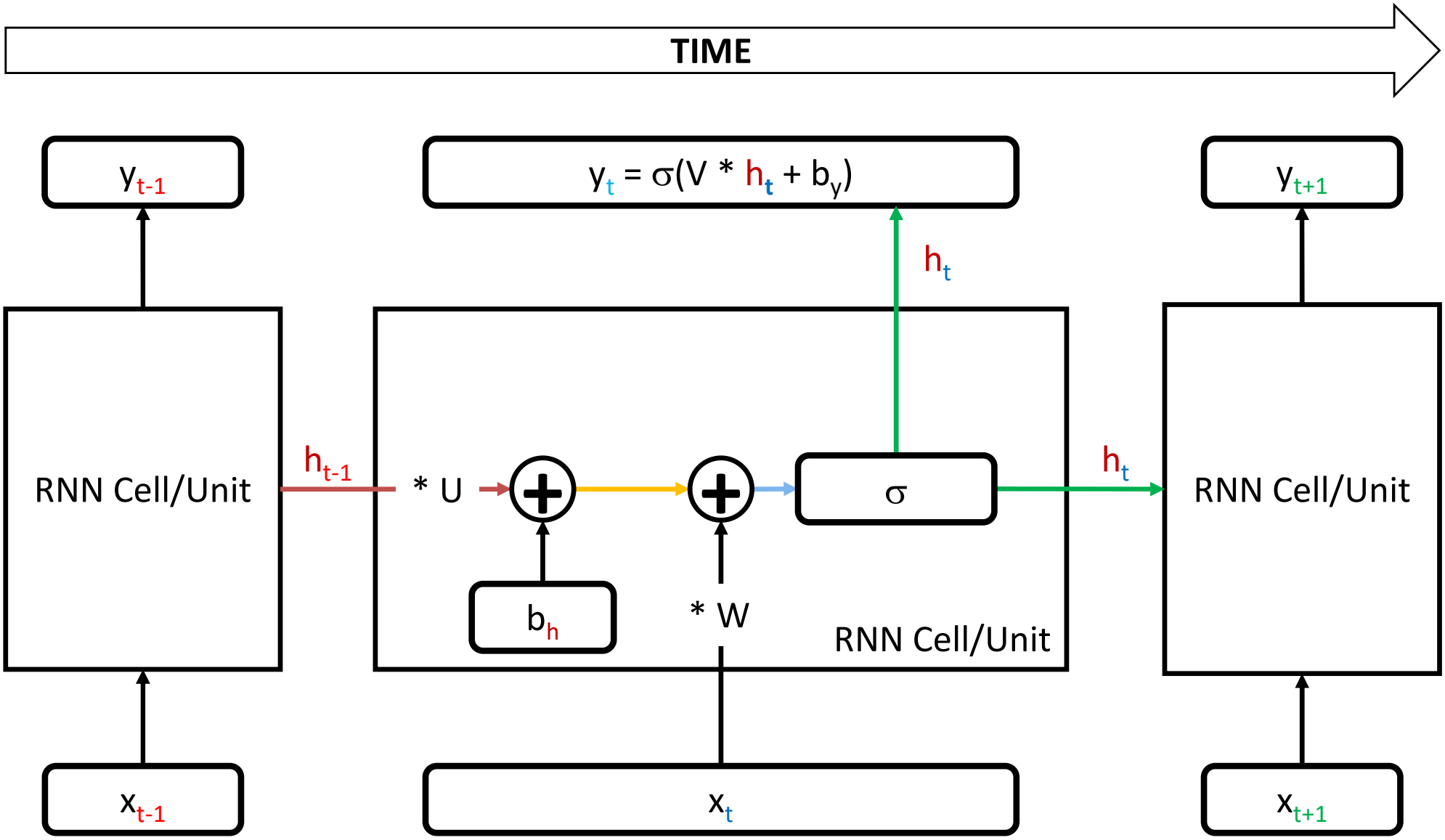


RNN Cell/Unit



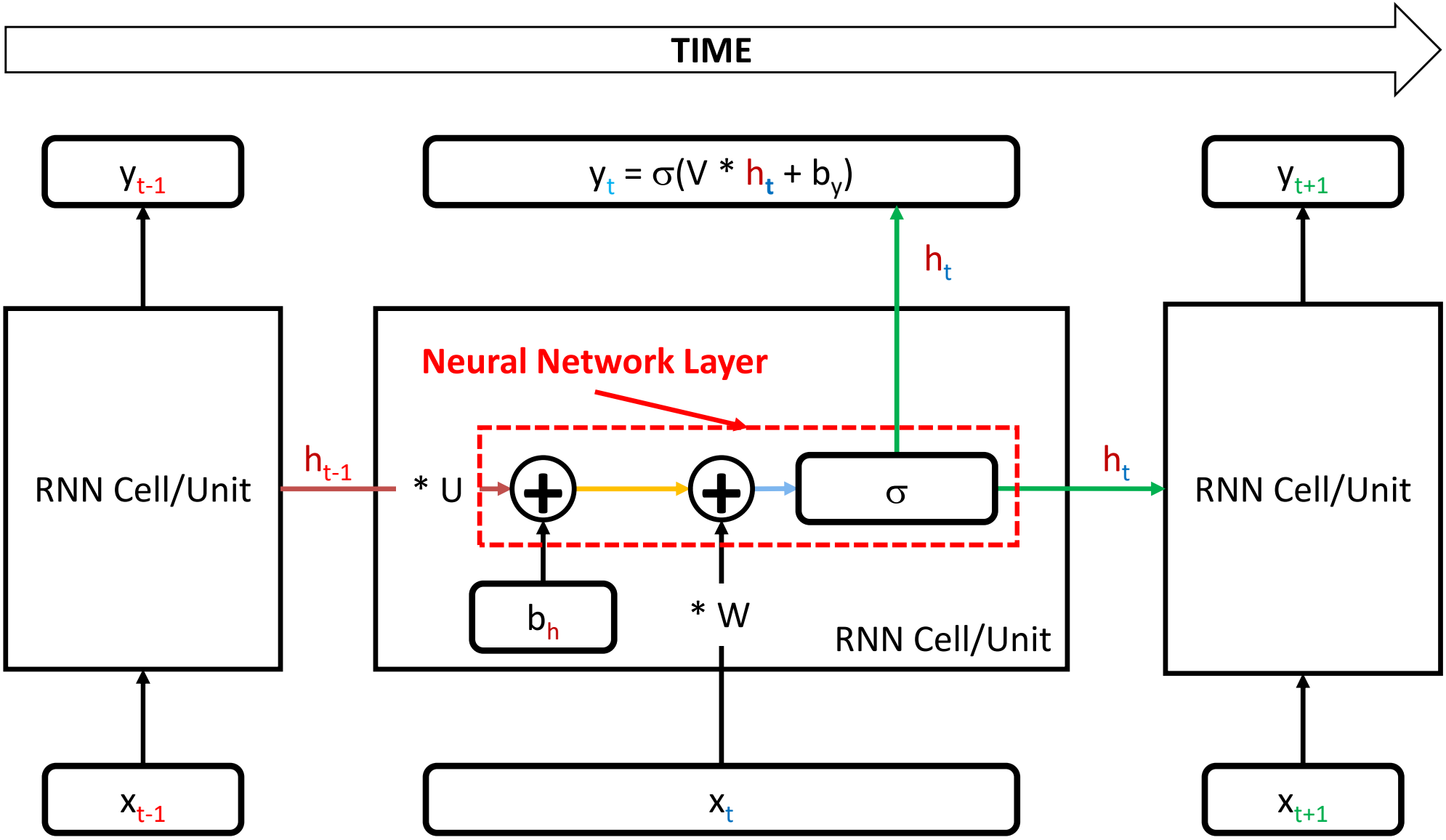
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

RNN Cell/Unit



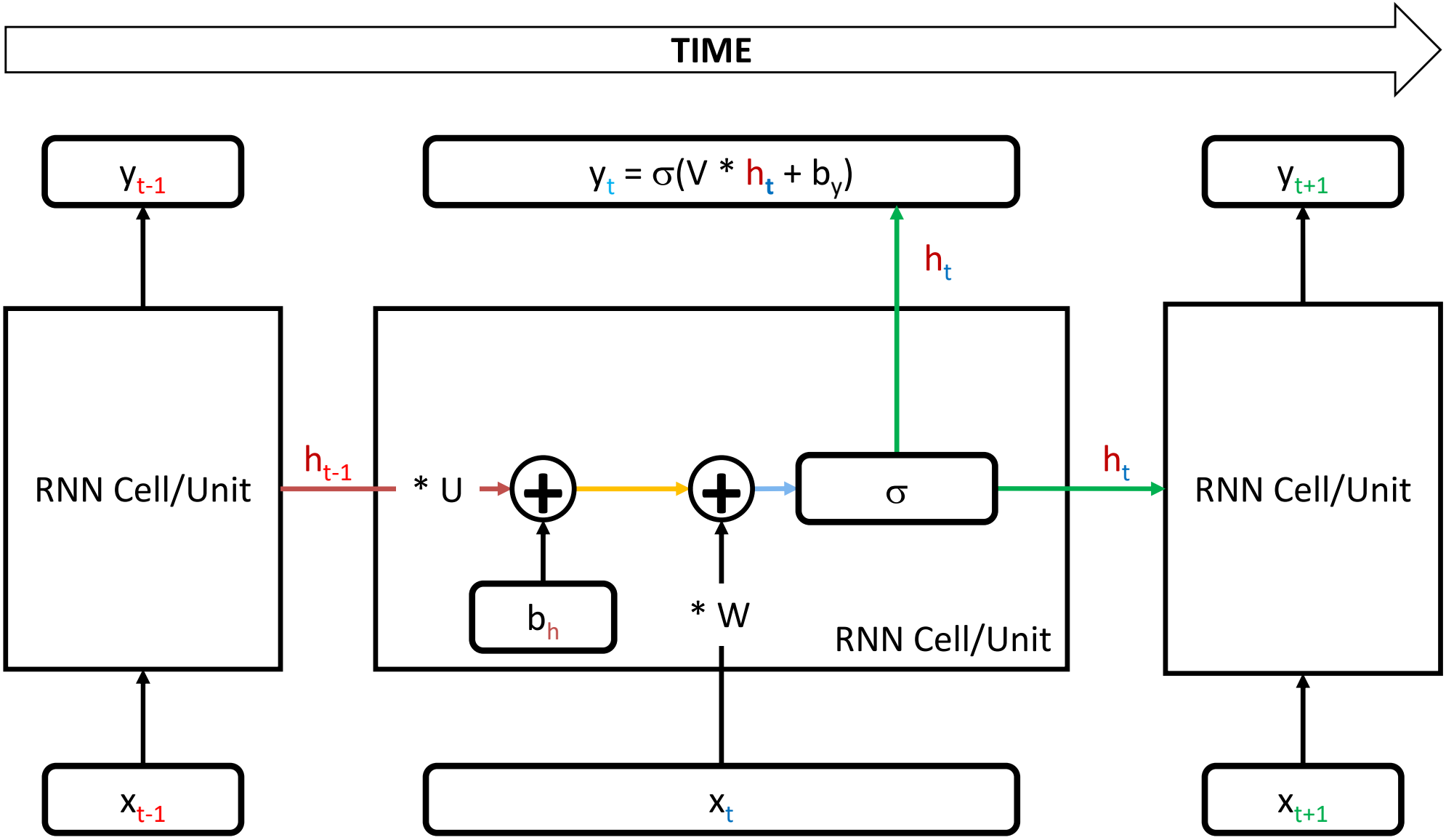
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

RNN Cell/Unit



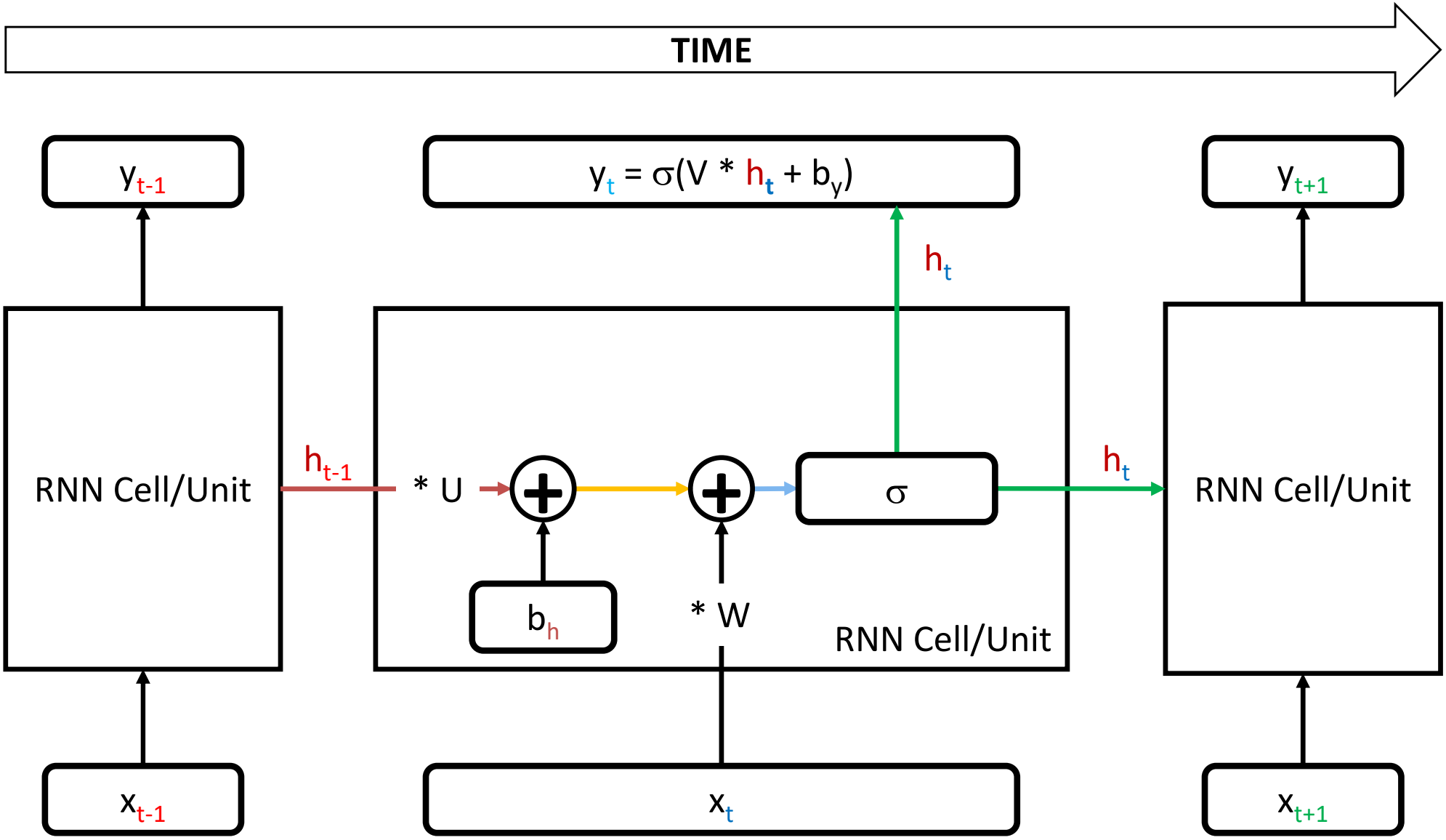
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

RNN Cell/Unit



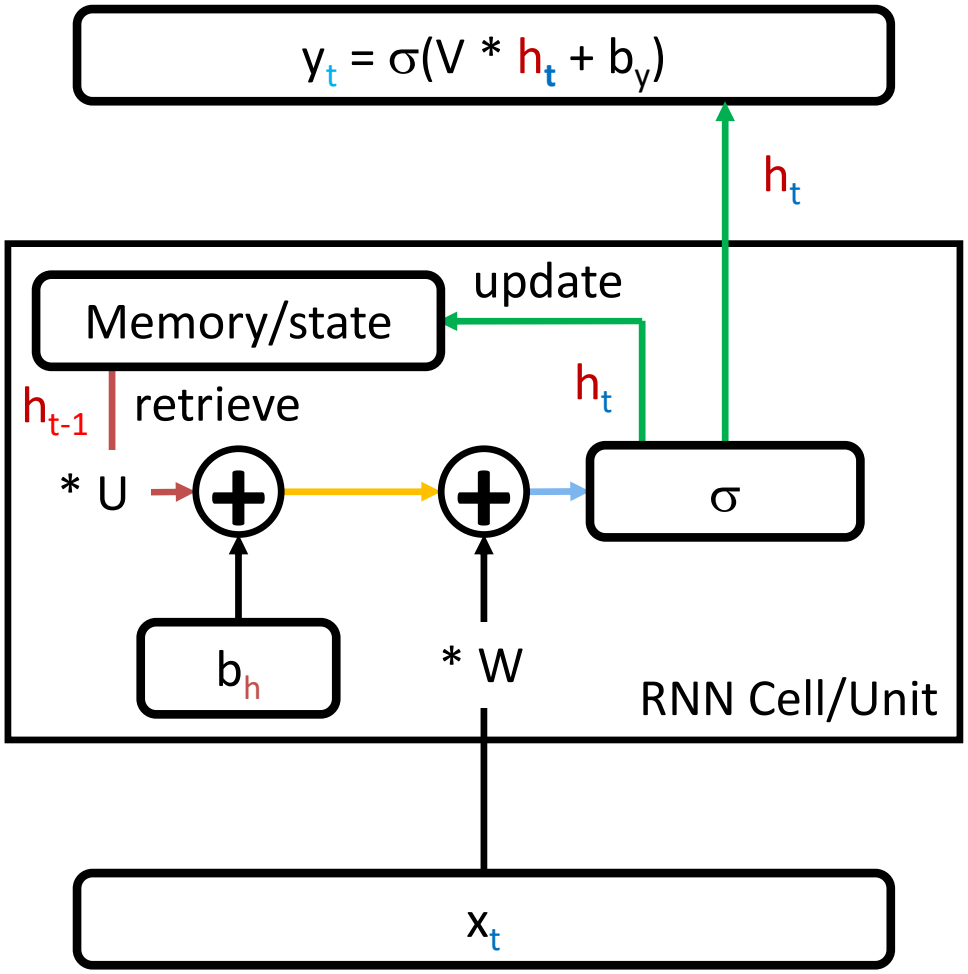
W: Input weight matrix | U: Recurrent weight matrix | V: Output weight matrix | b_y: output bias

RNN Cell/Unit



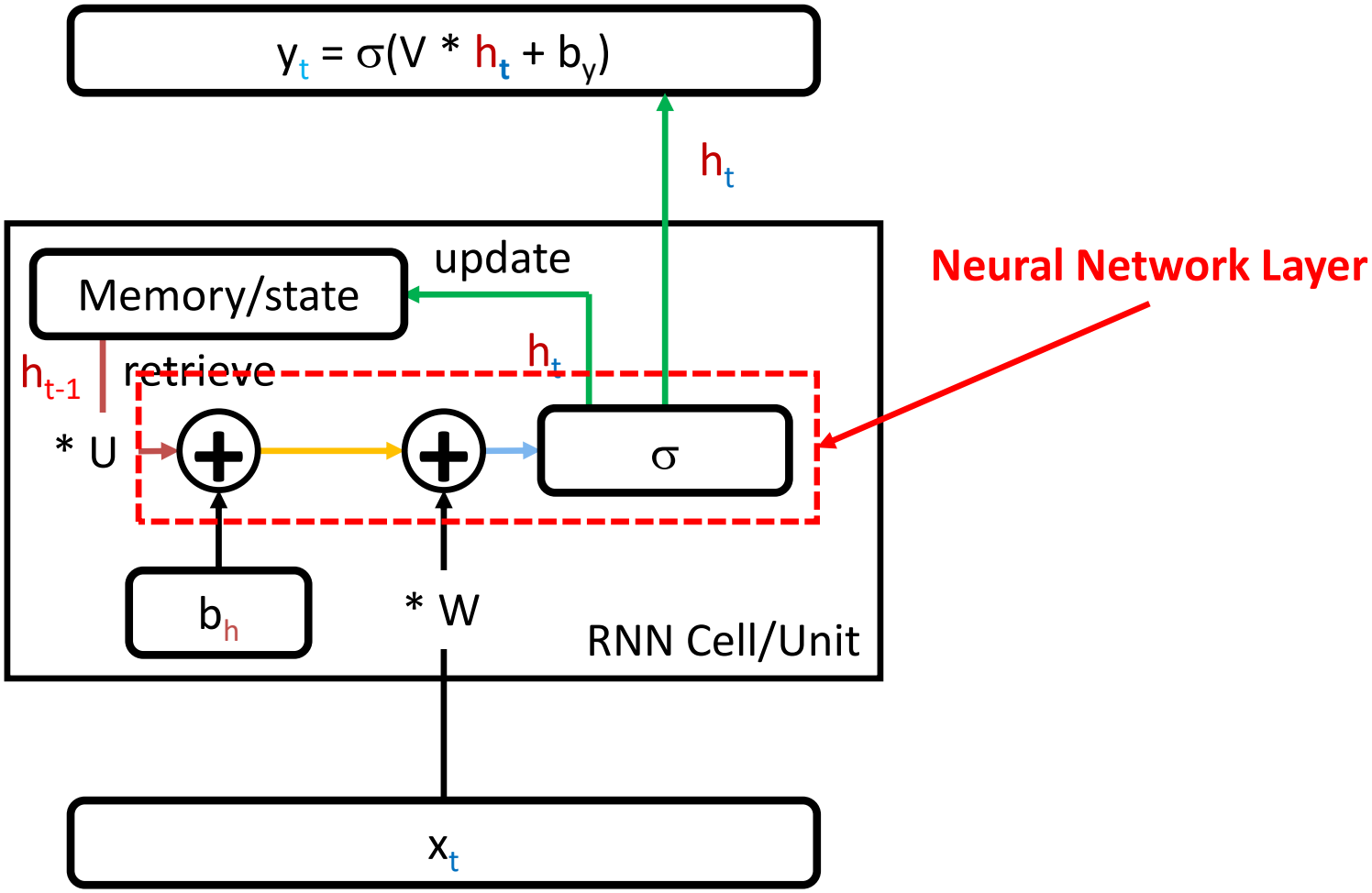
x_i: size d | h_i: size p | y_i: size d | W: p x d | U: p x p | V: d x p | b_h: size p | b_y: size d

In Practice: RNN Cell/Unit



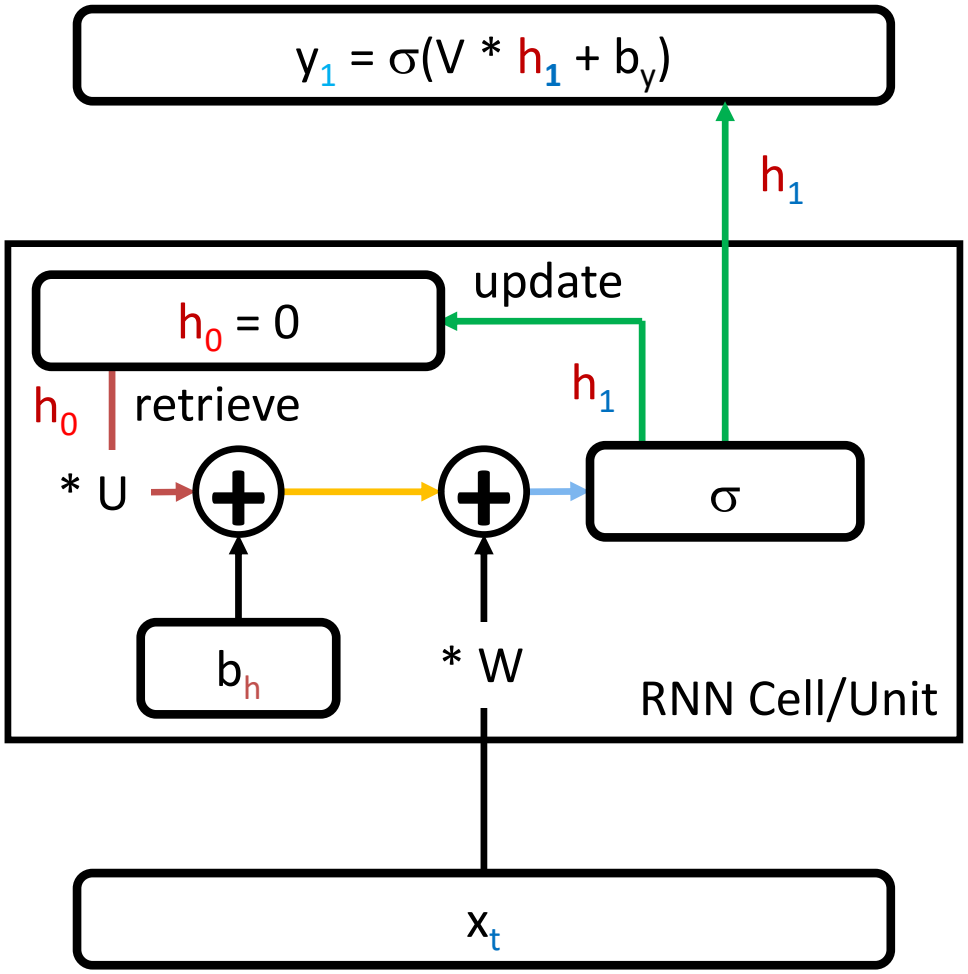
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

In Practice: RNN Cell/Unit



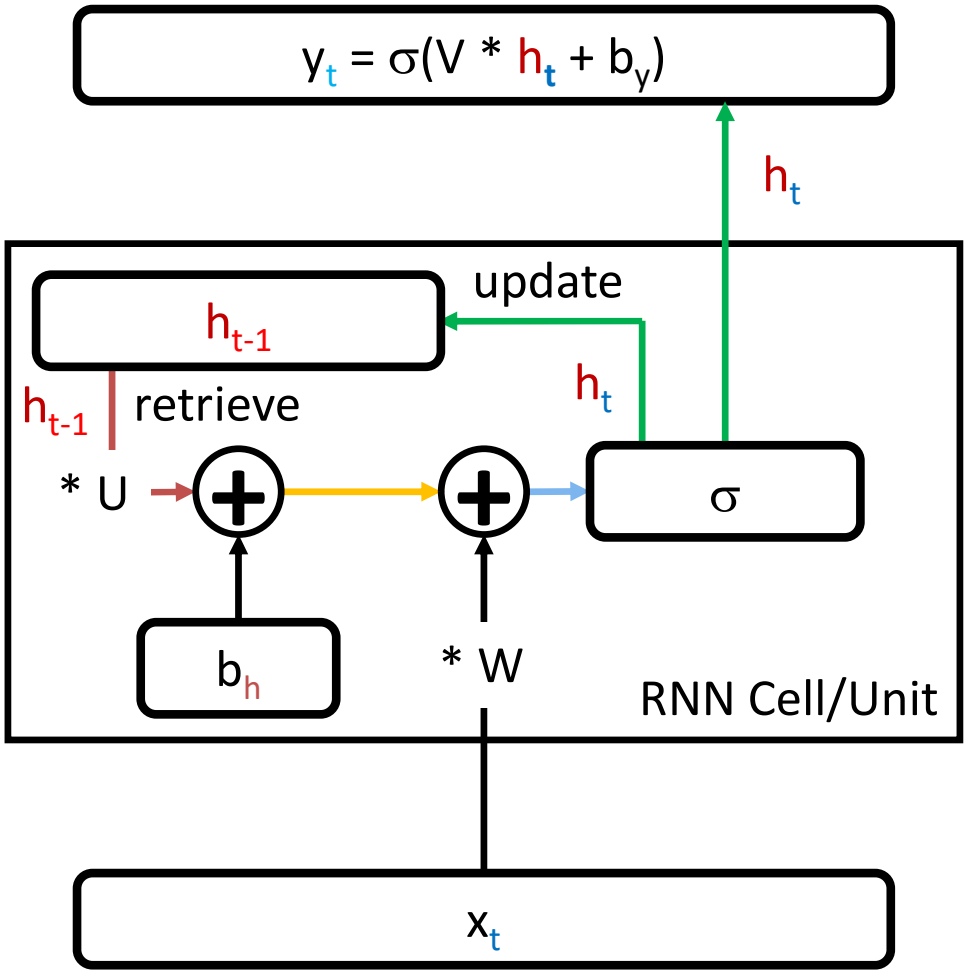
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

In Practice: RNN Cell/Unit | T = 1



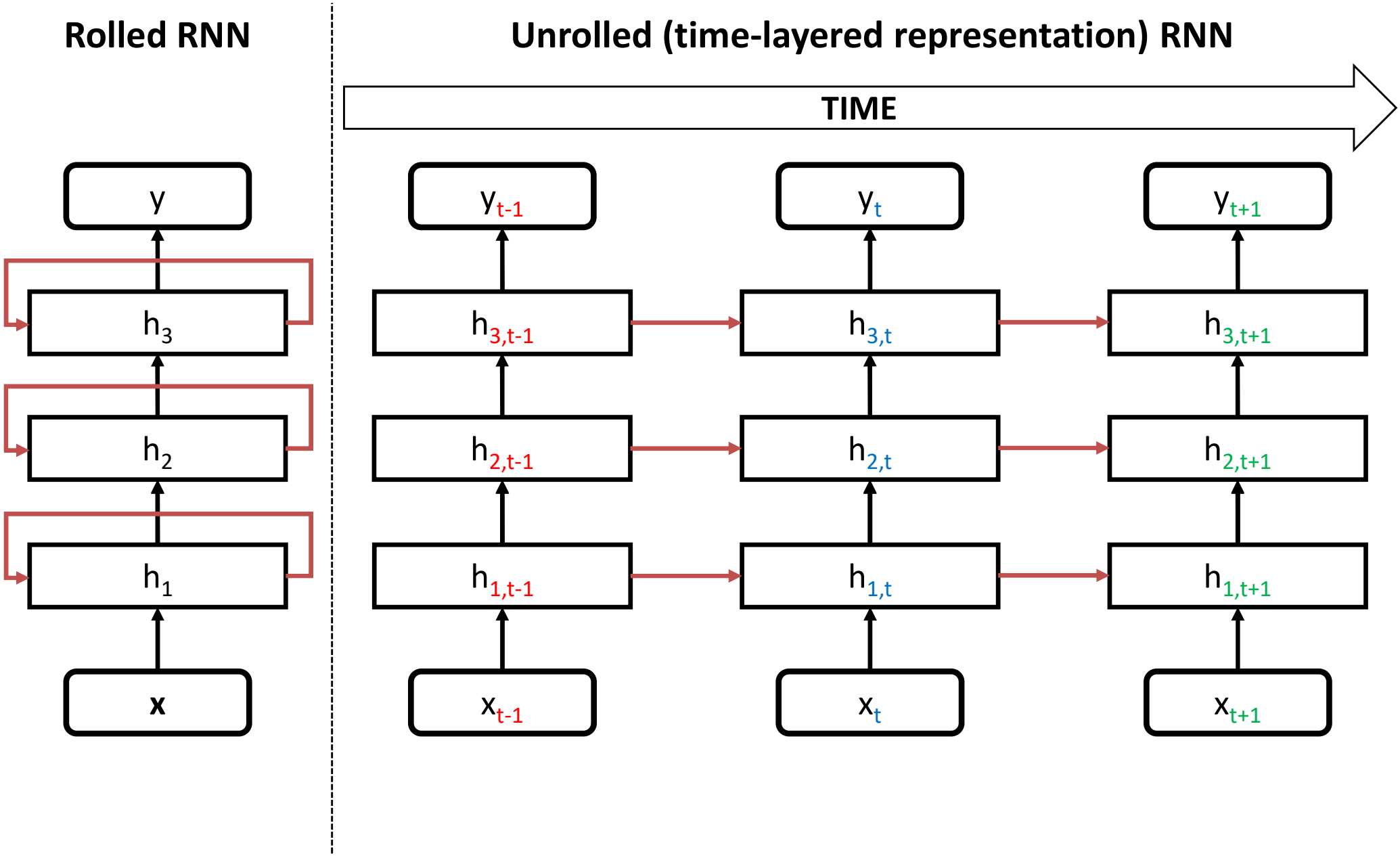
$h_1 = \sigma(W * x_1 + U * h_0 + b_h) = h_1 = \sigma(W * x_1 + U * \mathbf{0} + b_h) = h_1 = \sigma(W * x_1 + b_h)$

In Practice: RNN Cell/Unit | T = t

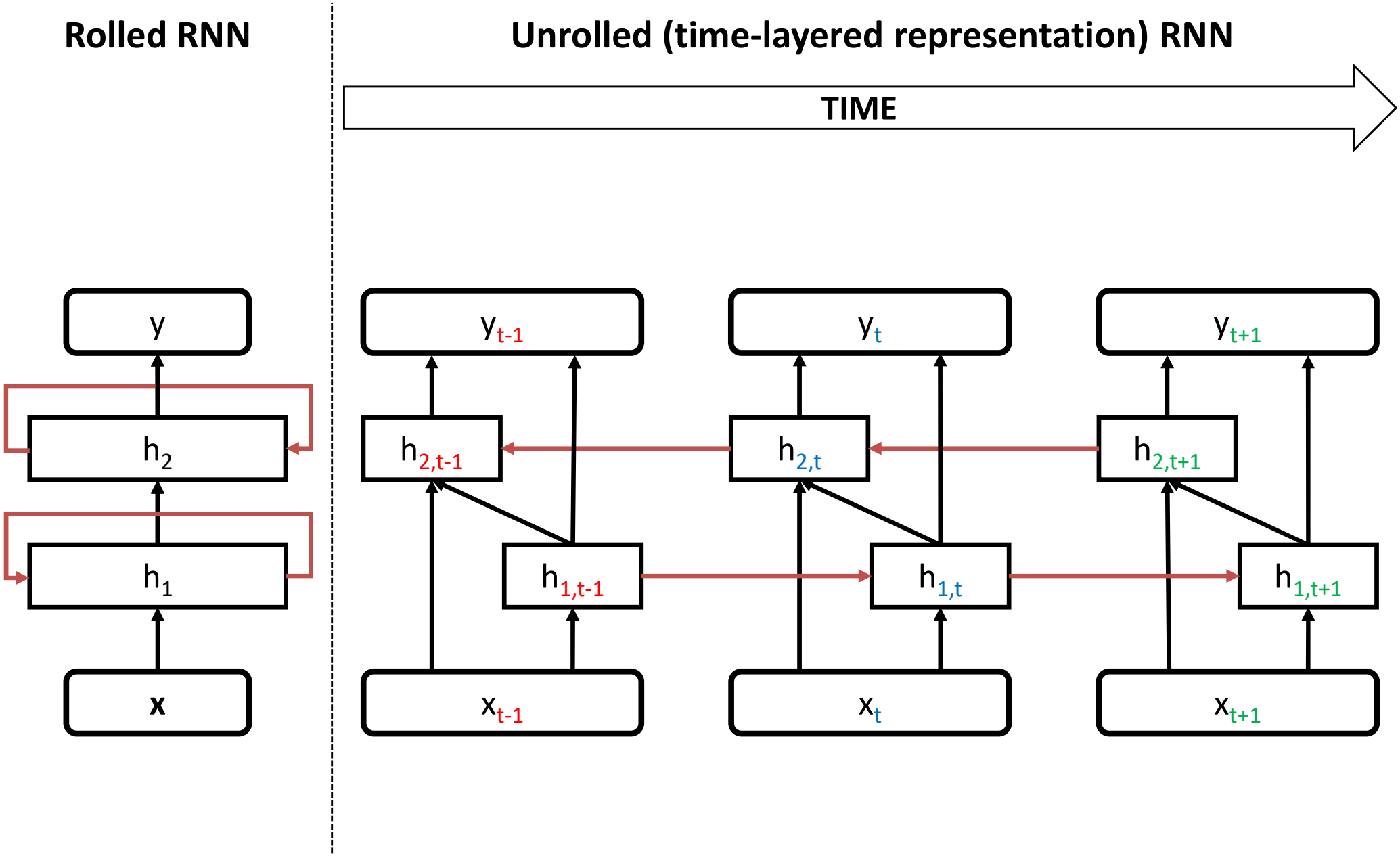


$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

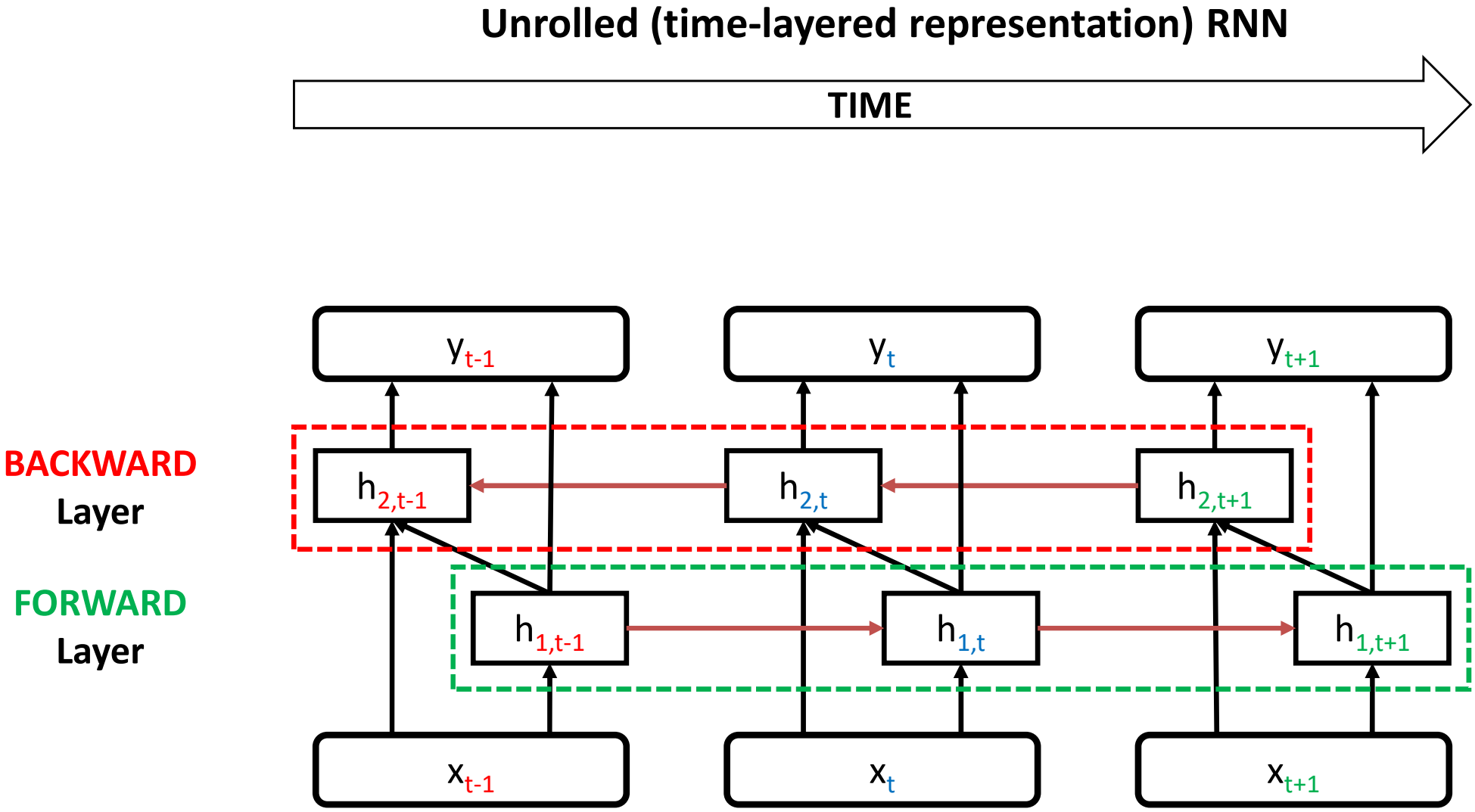
In Practice: Multi-Layer RNNs



In Practice: Bi-directional RNNs



In Practice: Bi-directional RNNs



RNN: Input - Output

$$y_5 = \sigma(Vh_5 + b_y)$$

$$\sigma(Wx_5 + Uh_4 + b_h)$$

$$\sigma(Wx_4 + Uh_3 + b_h)$$

$$\sigma(Wx_3 + Uh_2 + b_h)$$

$$\sigma(Wx_2 + Uh_1 + b_h)$$

$$\sigma(Wx_1 + U\mathbf{0} + b_h)$$

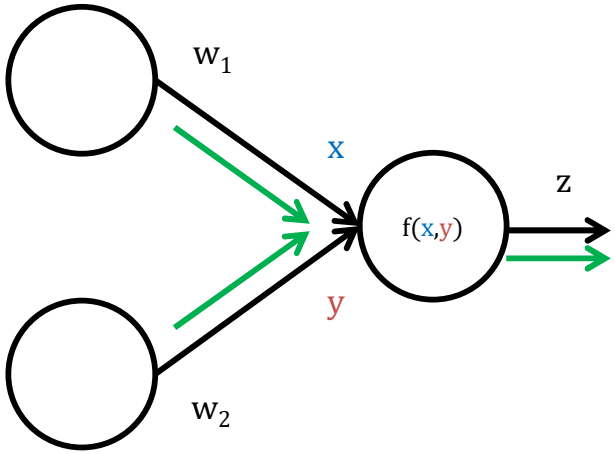
Training Neural Networks: Intuition

For every training tuple (x ,y) = (feature vector, label)

- **Run forward computation to find estimate ŷ**
- **Run backward computation to update weights:**
  - **For every output node**
    - **Compute loss L between true y and the estimated ŷ**
    - **For every weight w from hidden layer to the output layer**
    - **Update the weight**
  - **For every hidden node**
    - **Assess how much blame it deserves for the current answer**
    - **For every weight w from input layer to the hidden layer**
      - **Update the weight**

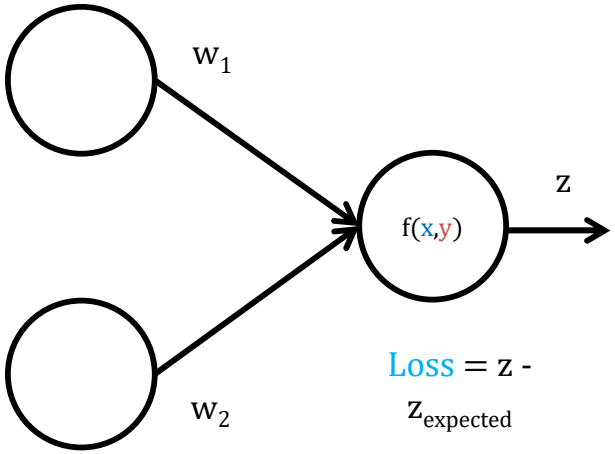
Back-propagation

Feed forward



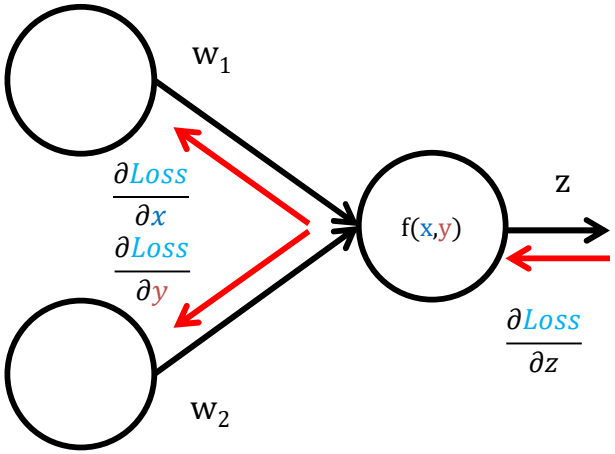
Feed a **labeled sample** through the network

Evaluate Loss



How "incorrect" is the result compare to the label?

Back-propagation



Update weights (use **Gradient Descent**)

Gradients and Learning Rate

- The value of the gradient (slope in our example) $\frac{d}{dw} L(f(x;w), y)$ weighted by a **learning rate** η

- Higher learning rate means move **w** faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)$$

Vanishing and Exploding Gradients

- As we've seen, information needs to travel a long way in an RNN to get from the error signal / loss function (y) to some inputs (xi). By the chain rule of differentiation, the gradient of the loss function will have the form

$$W \times \sigma'(z_1) \times U \times \sigma'(z_2) \times U \times \sigma'(z_3) \cdots$$

- Vanishing gradients: Elements of U are less than one, and gradients drop off to zero
- Exploding gradients: Elements of U are greater than one and gradients increase without limit

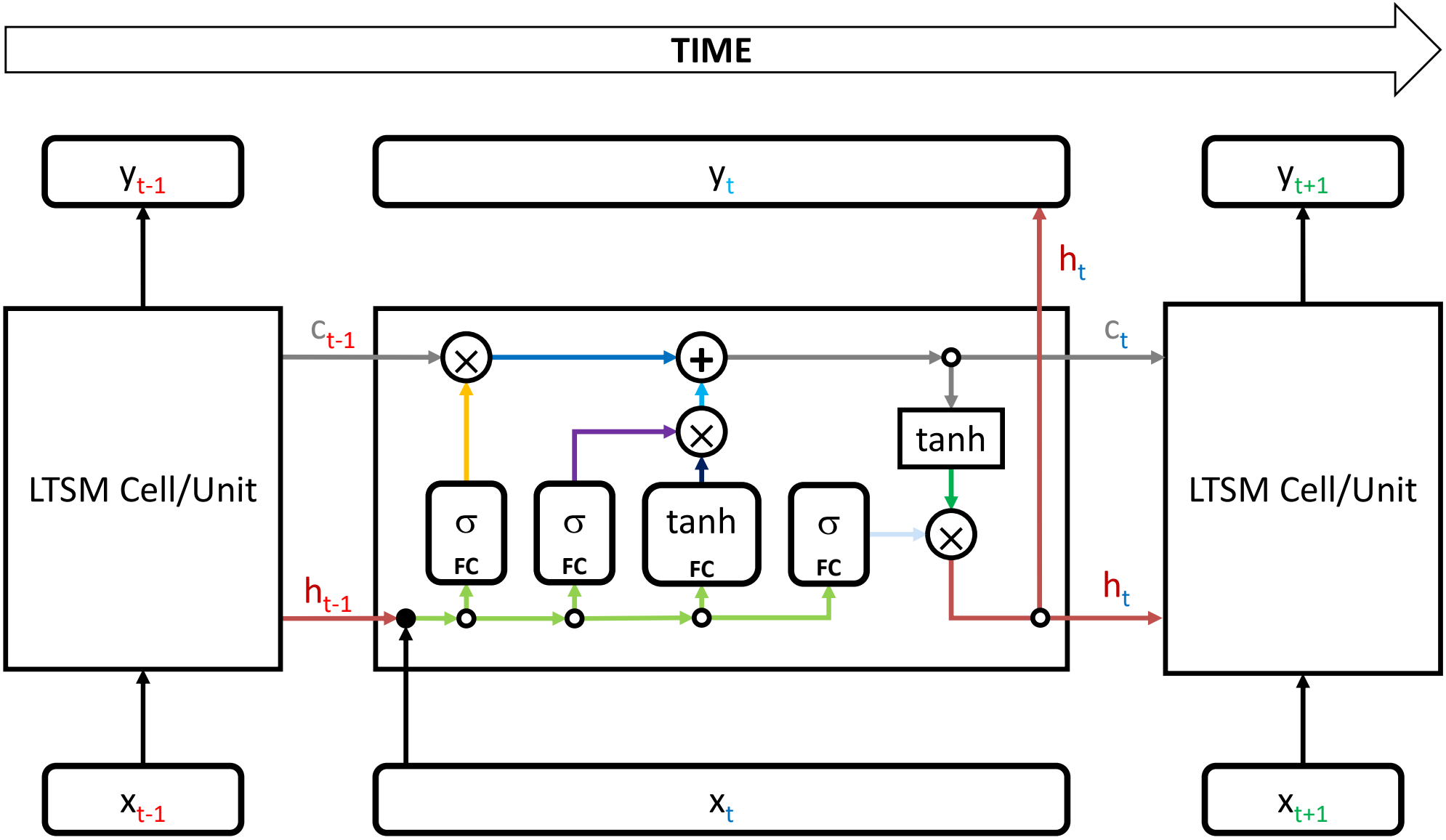
Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM)

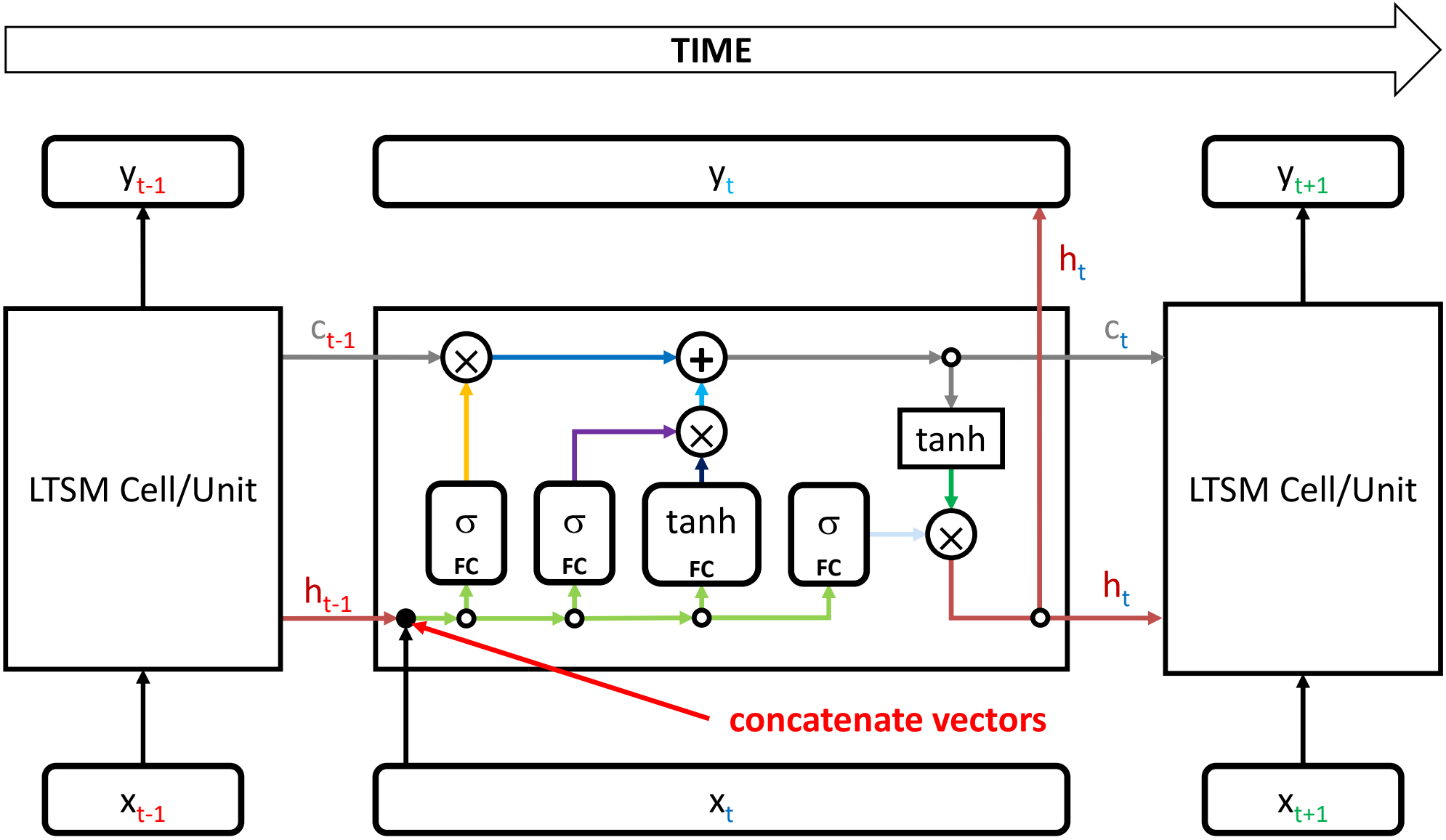
- A more sophisticated version of the recurrent neural network is the Long Short Term Memory (LSTM)

- An LSTM uses gates to determine what information feeds forward from one time step of the network to the next
  - this helps to address the vanishing/exploding gradient problems and make learning more stable

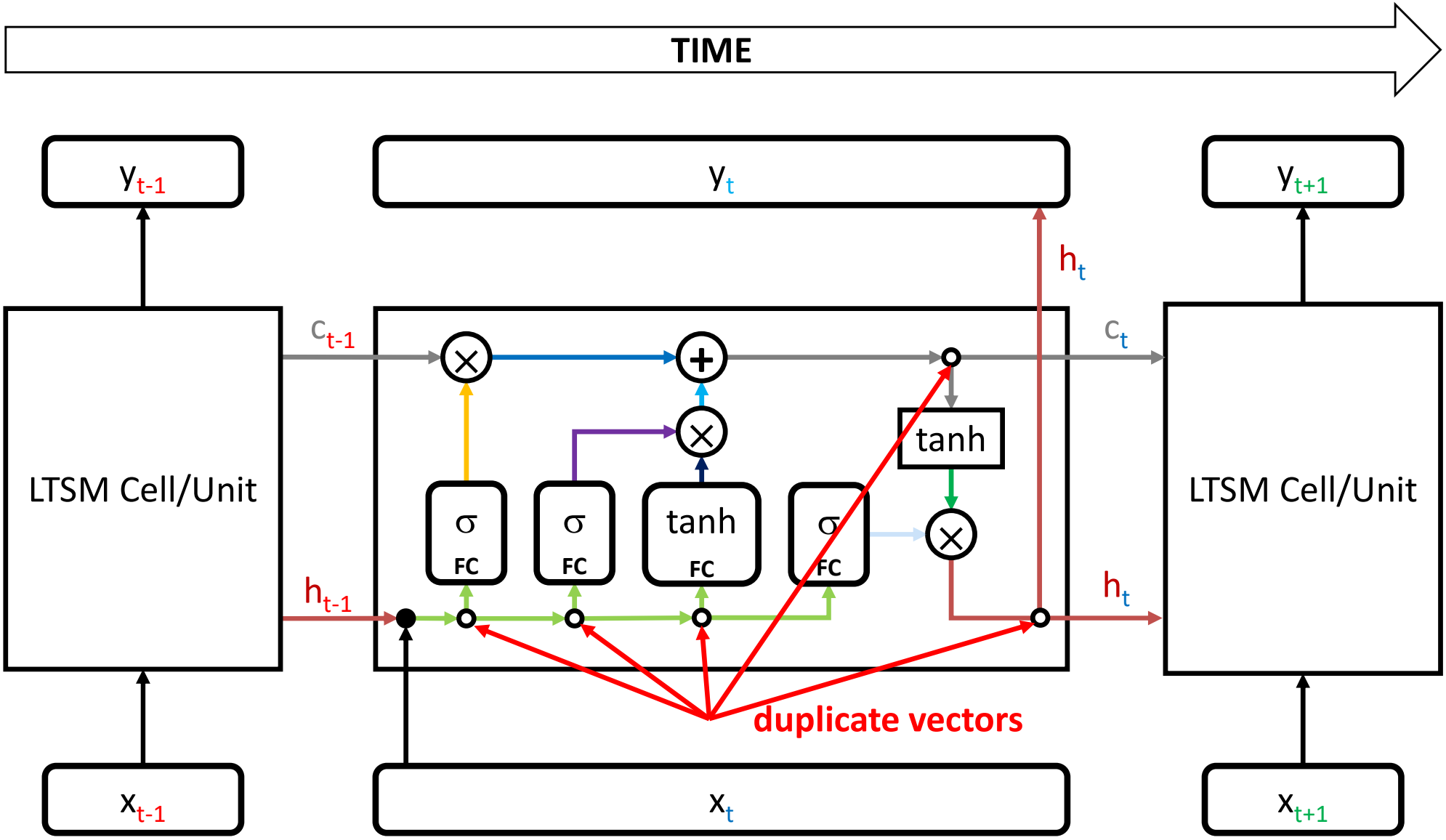
LSTM Cell/Unit



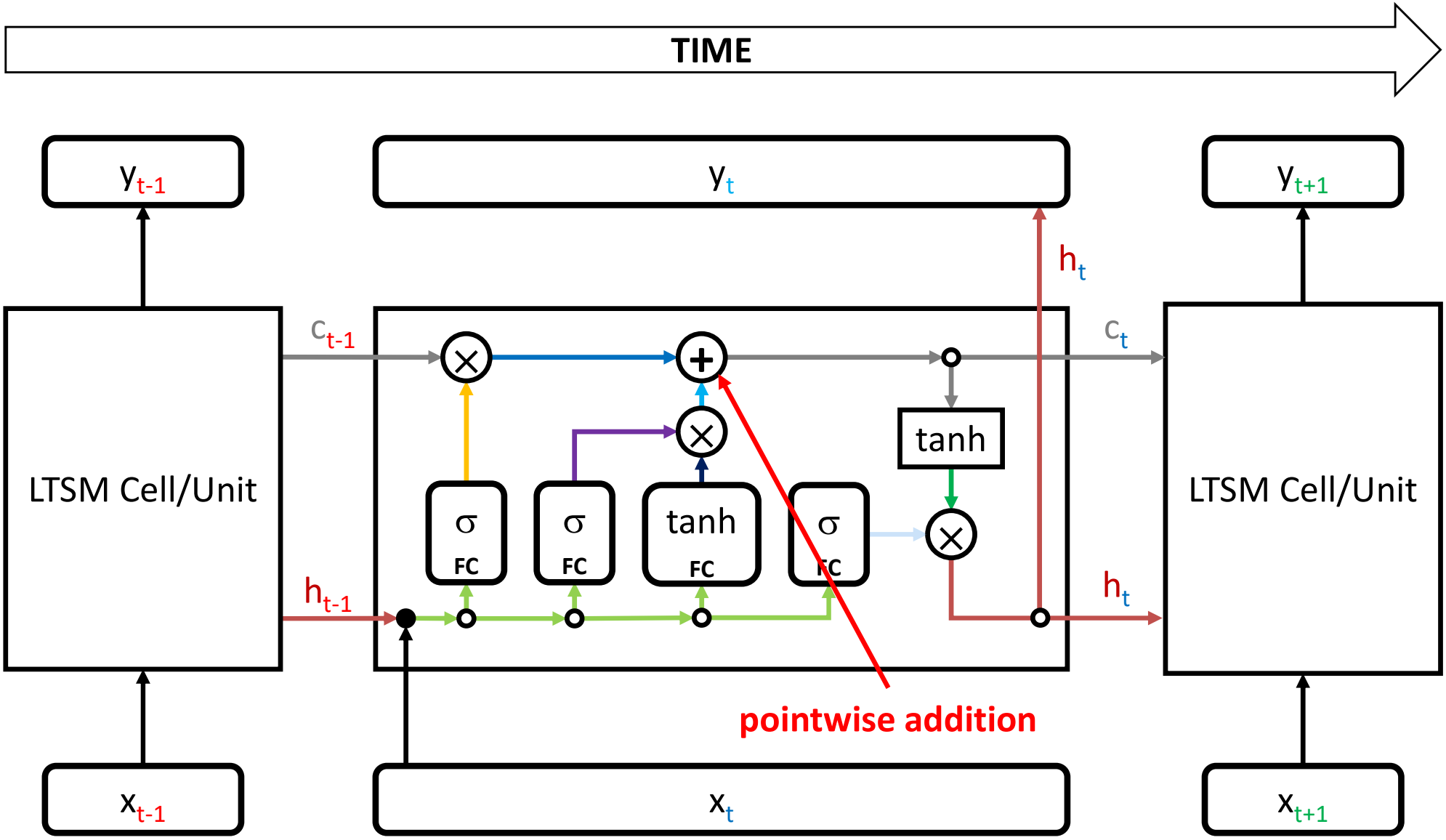
LSTM Cell/Unit



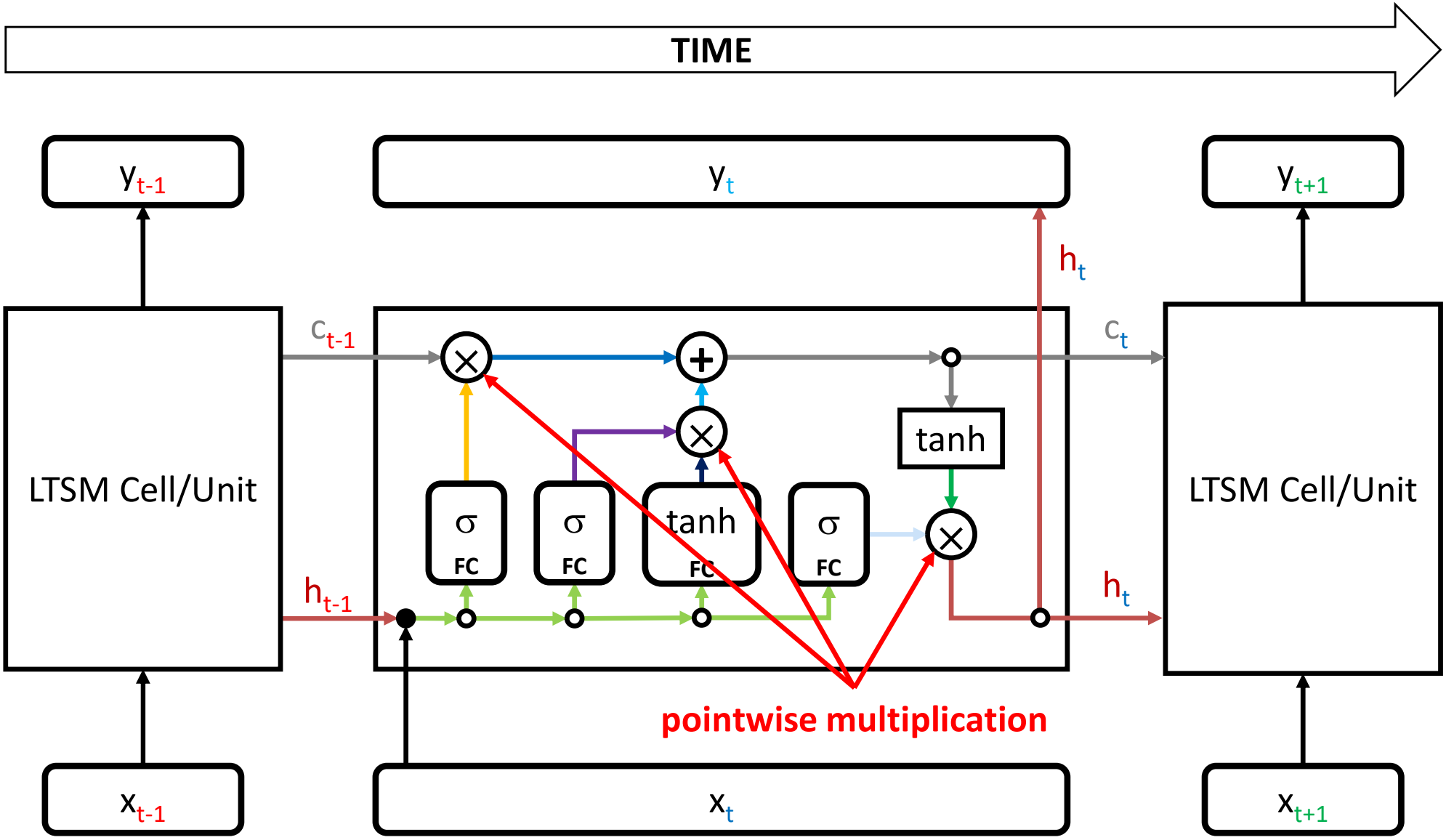
LSTM Cell/Unit



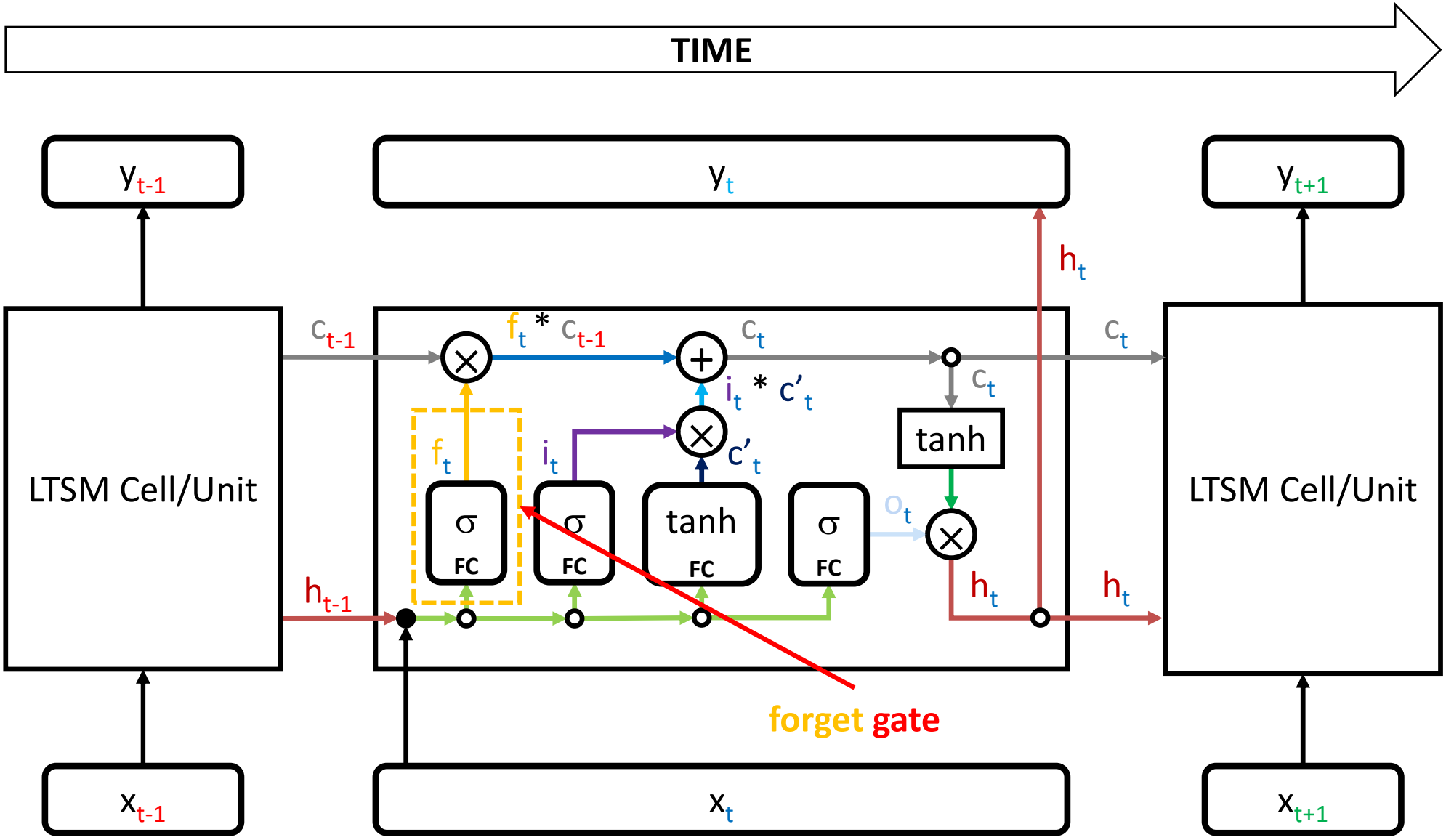
LSTM Cell/Unit



LSTM Cell/Unit

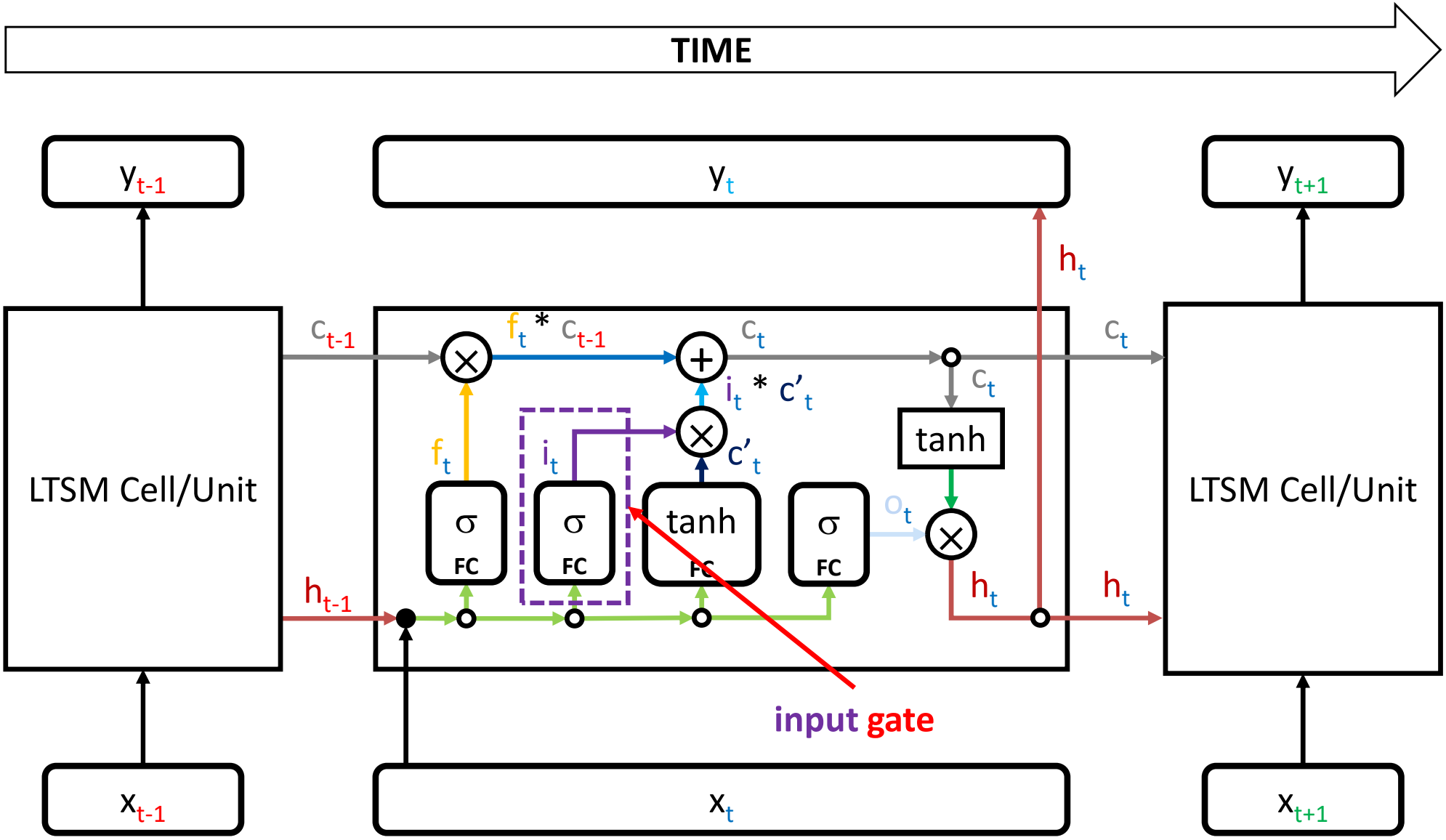


LSTM Cell/Unit



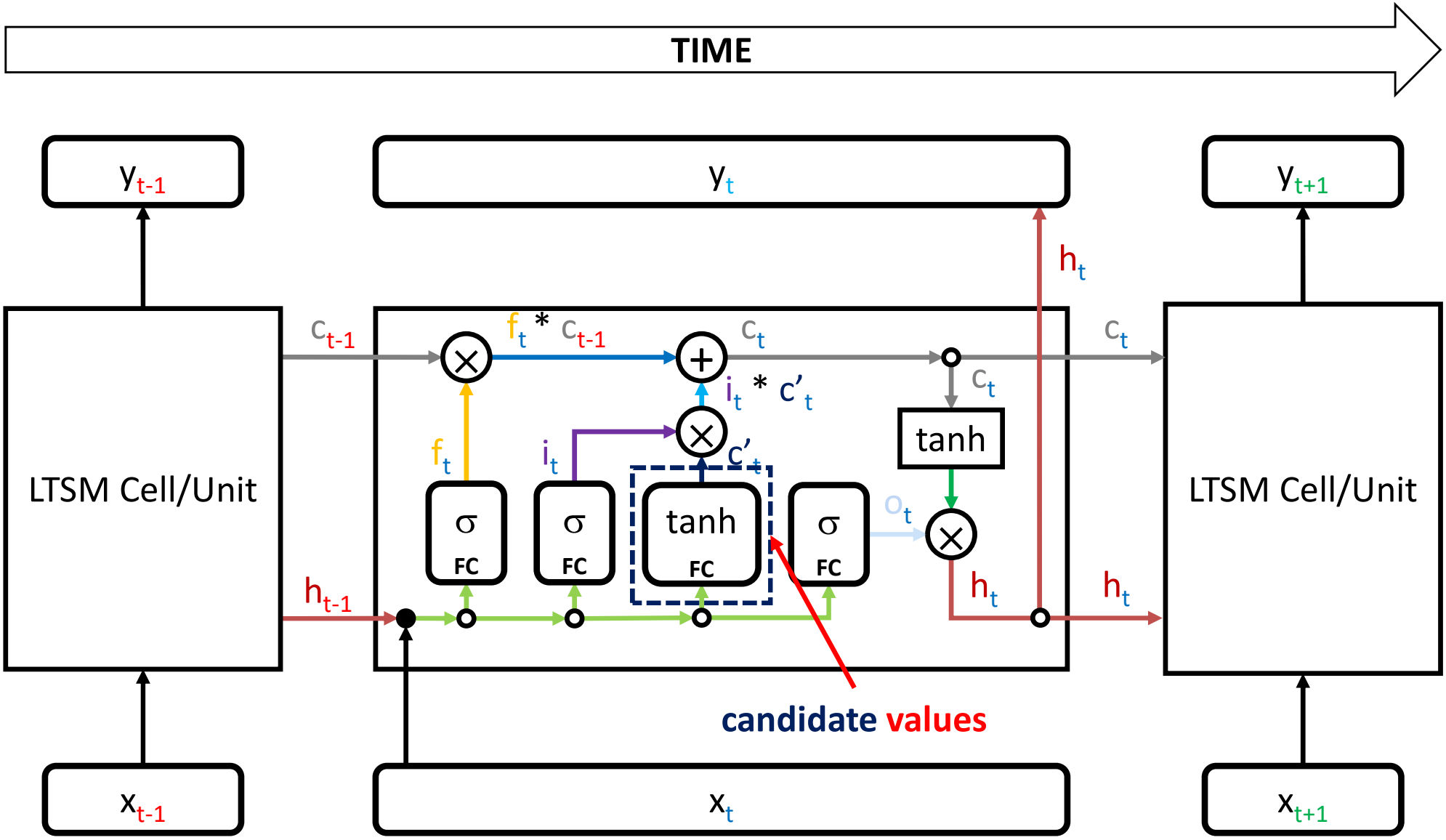
forget gate: determines how much of previous cell state is incorporated into current cell state

LSTM Cell/Unit

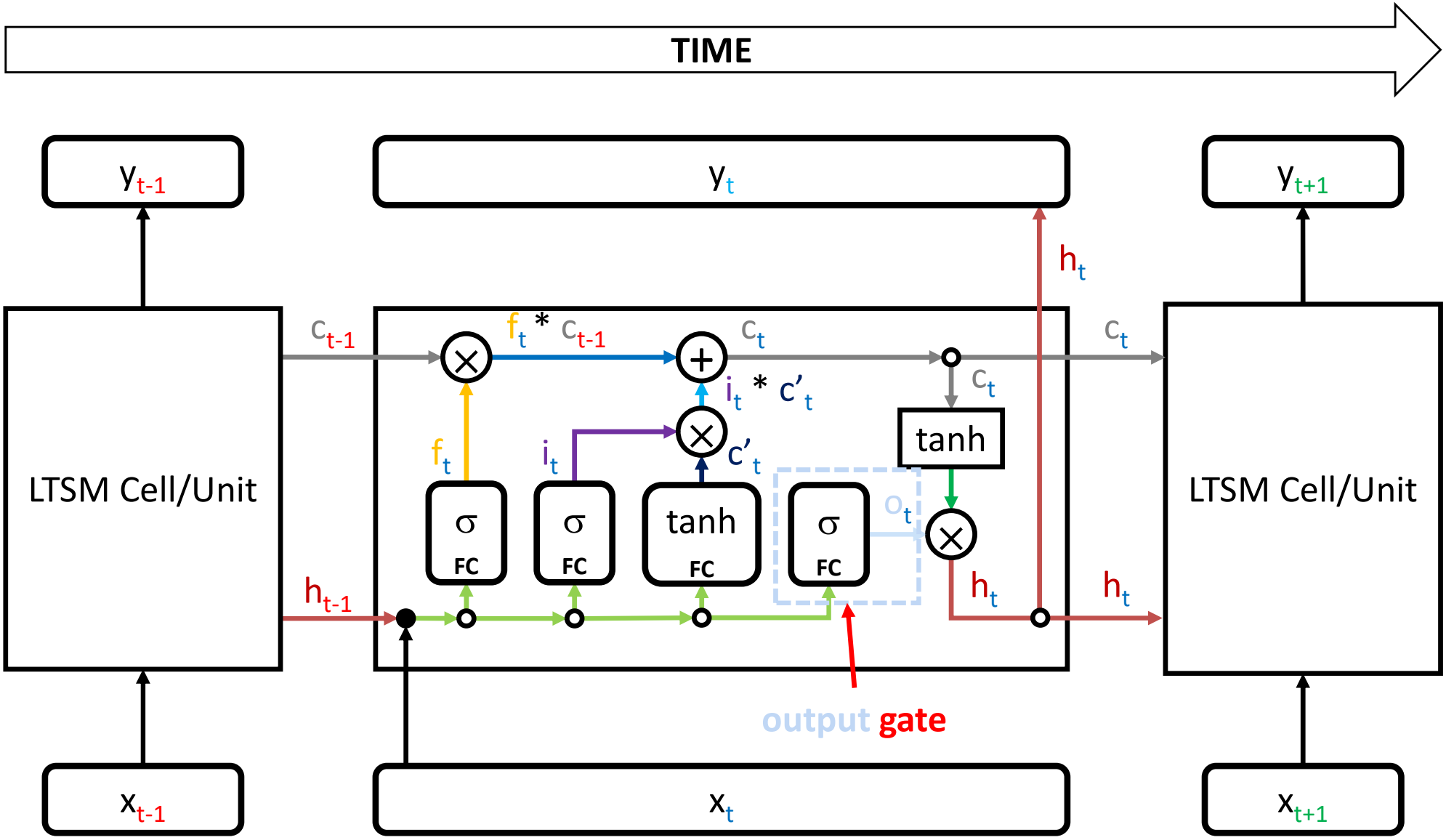


input gate: determines how much of input is incorporated into cell state

LSTM Cell/Unit

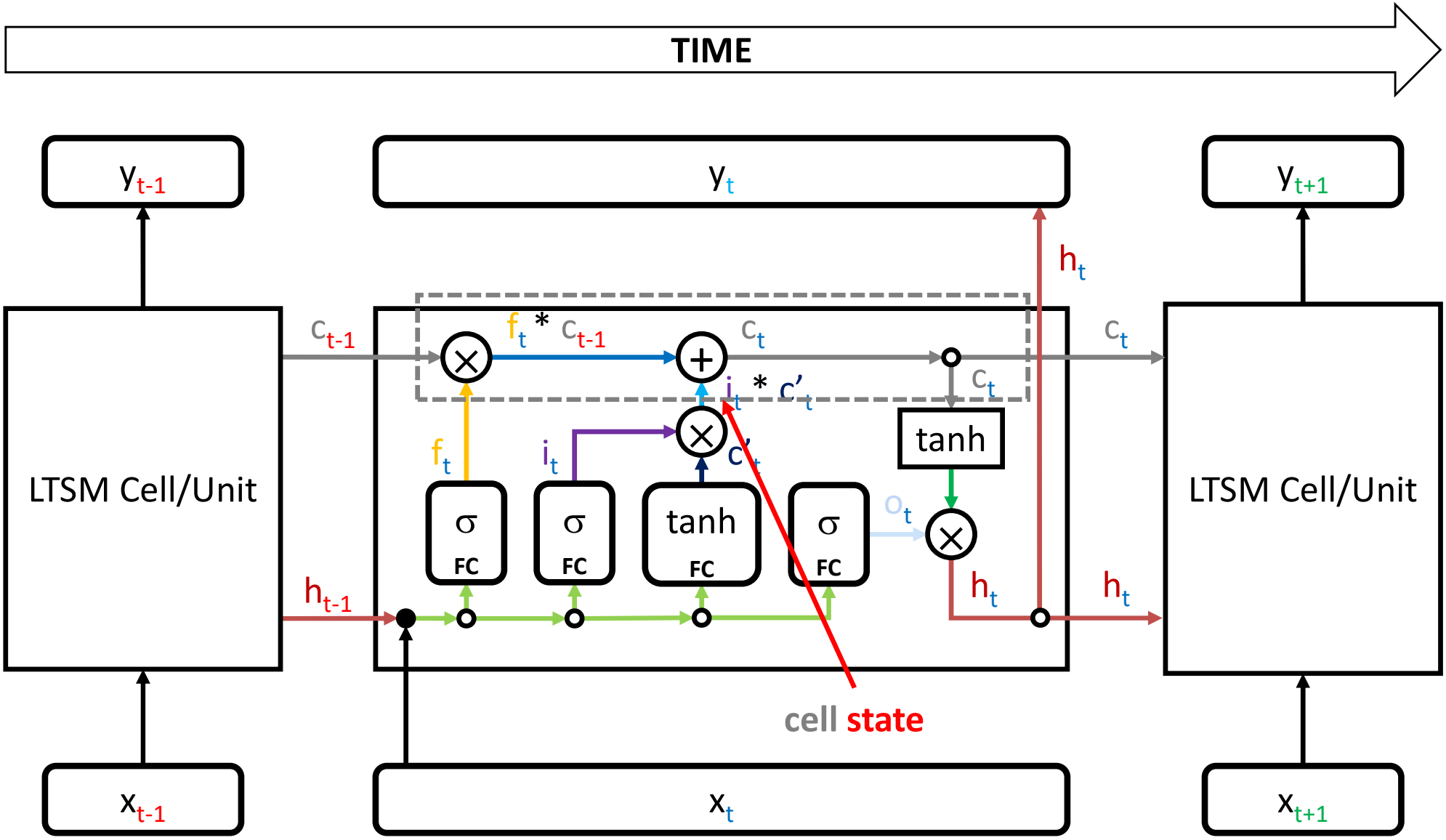


LSTM Cell/Unit

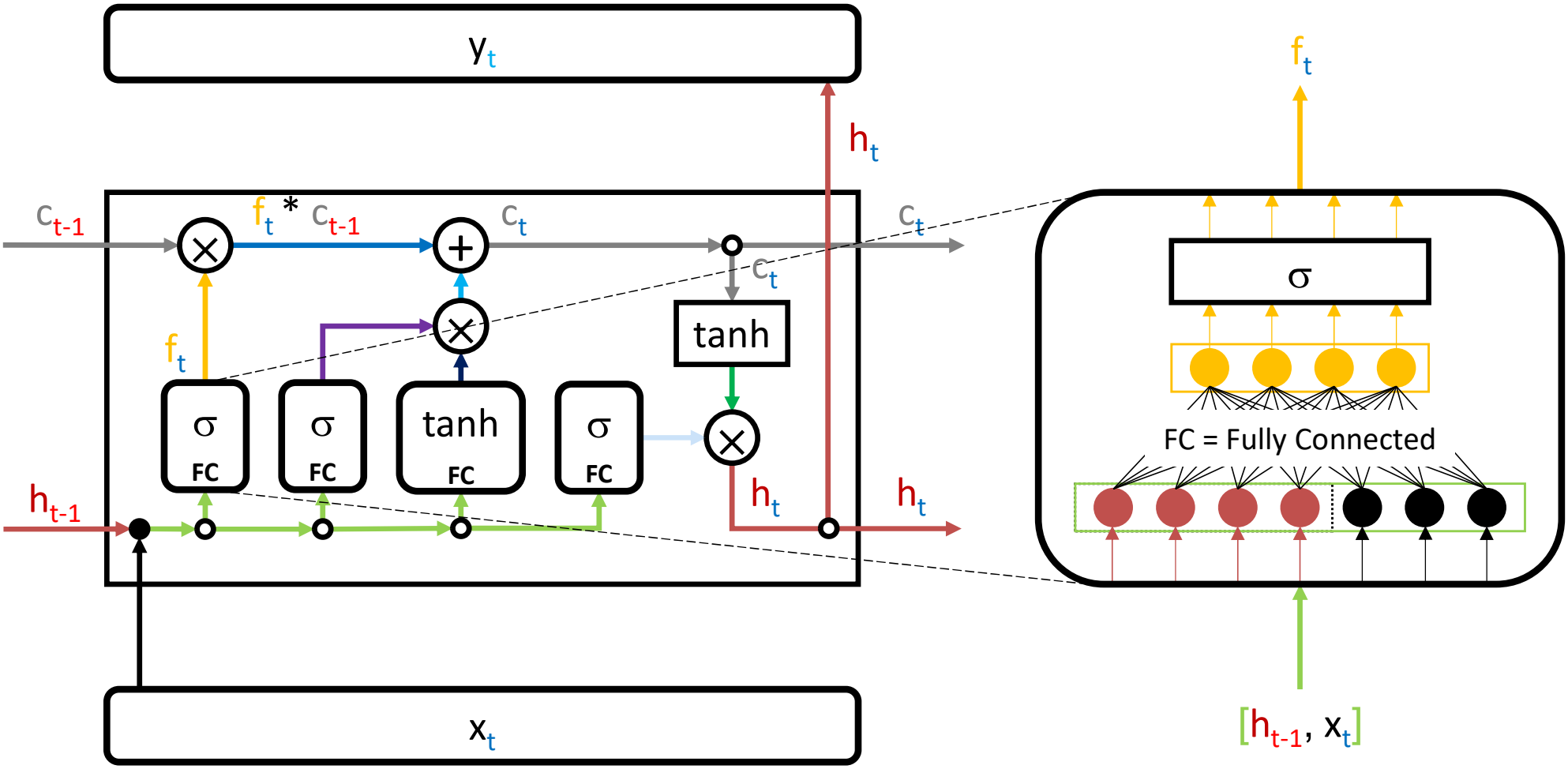


output gate: determines how much of current cell state is incorporated into current output

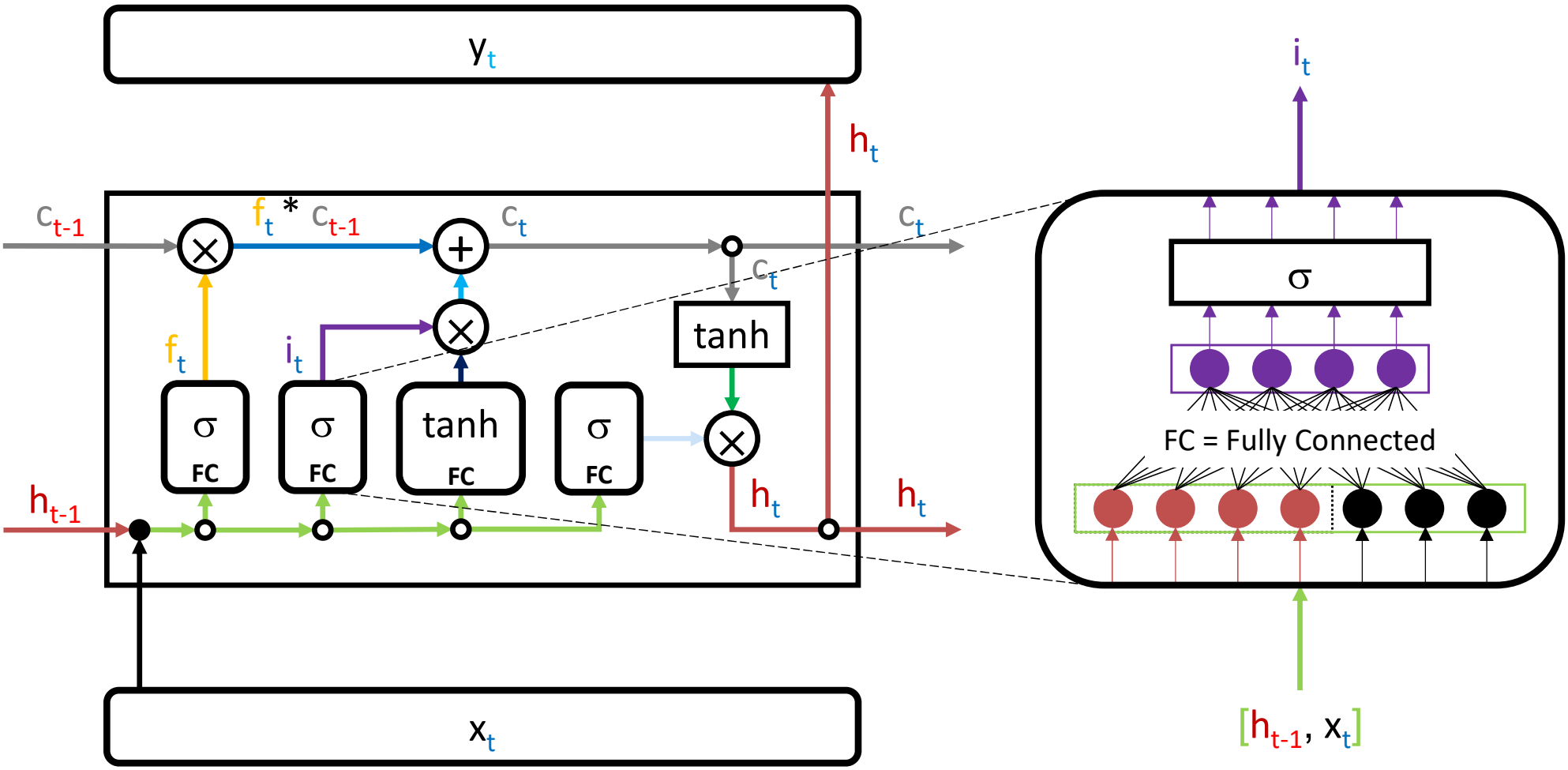
LSTM Cell/Unit



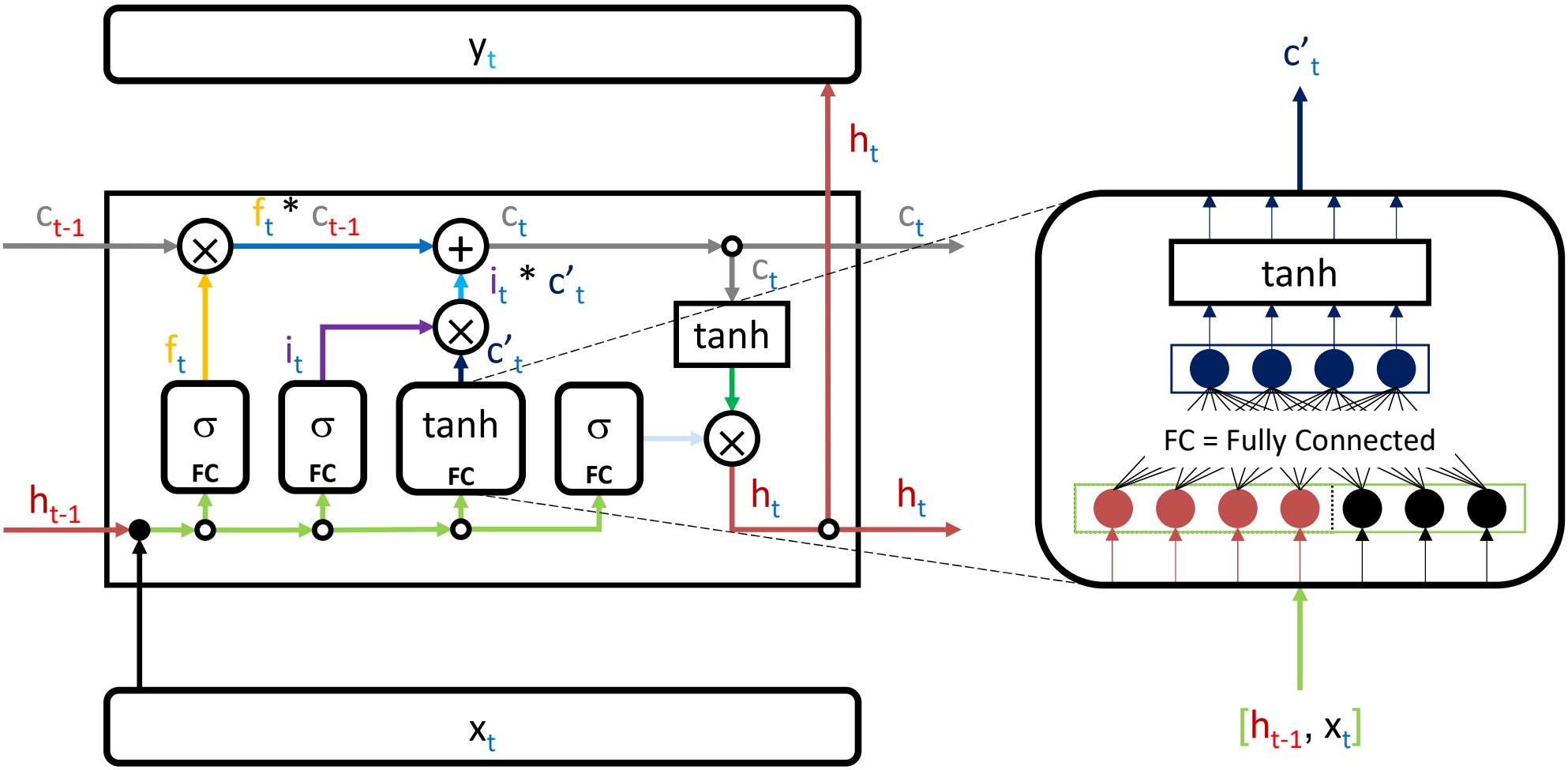
LSTM Cell/Unit



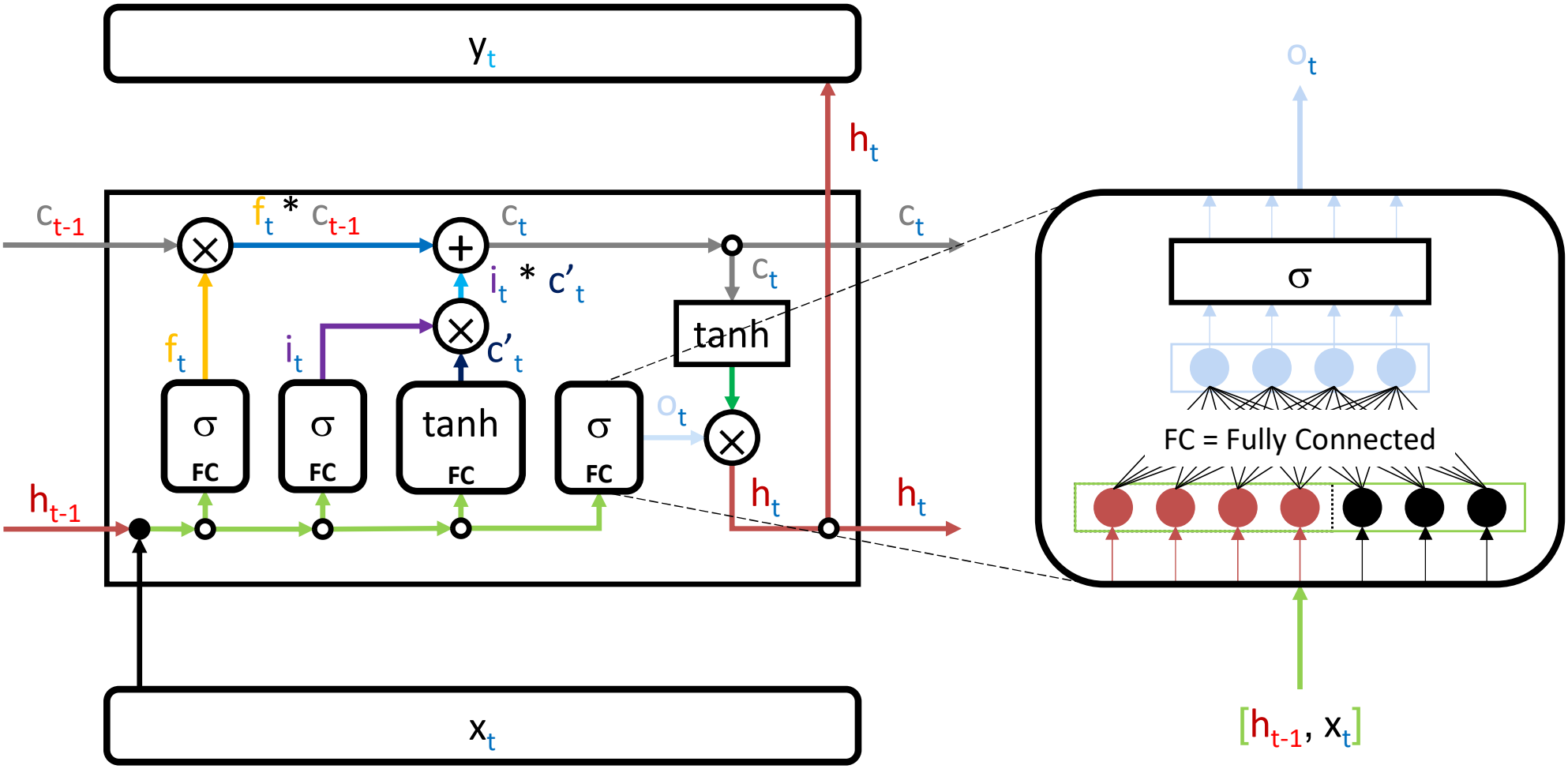
LSTM Cell/Unit



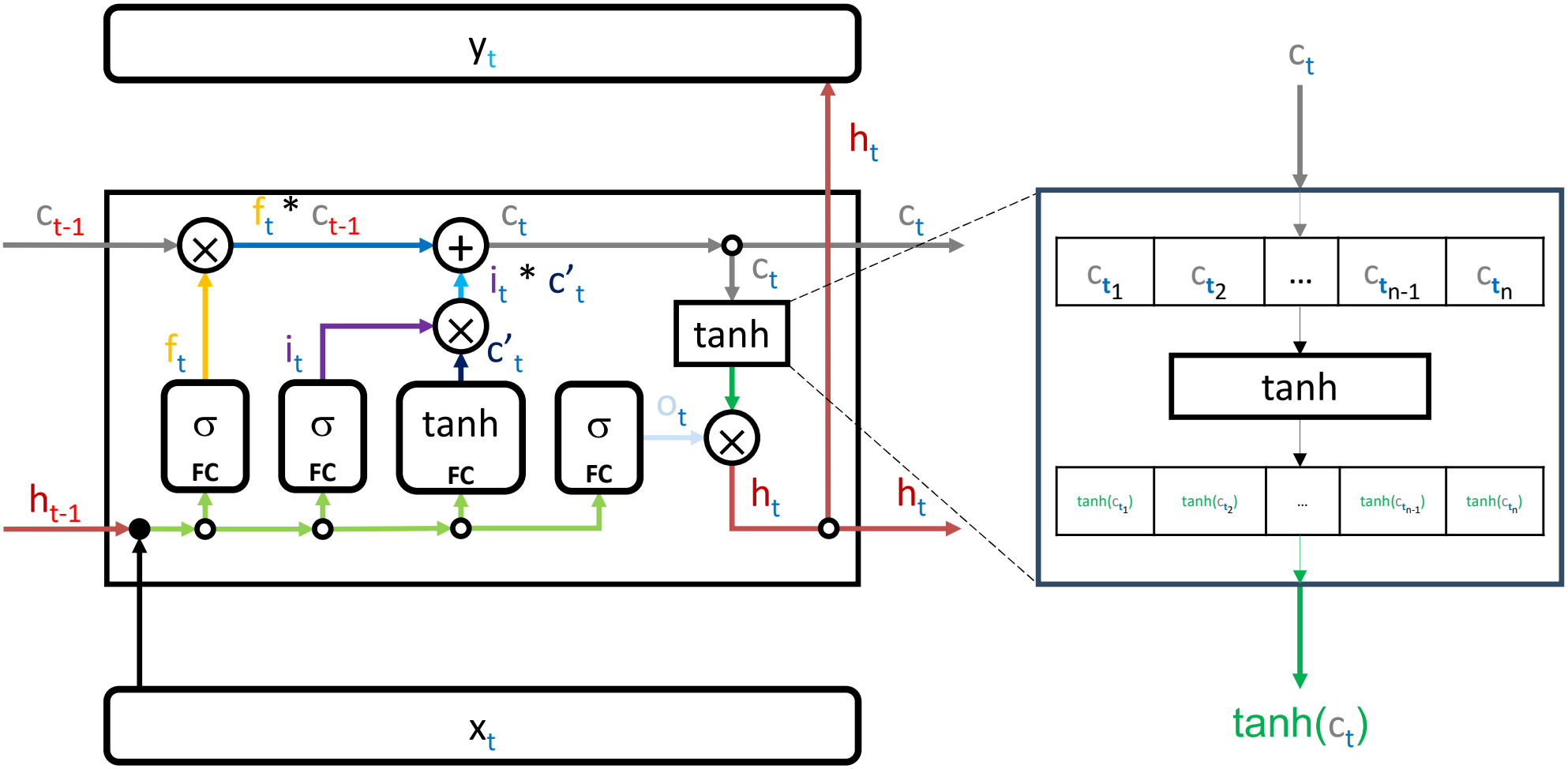
LSTM Cell/Unit



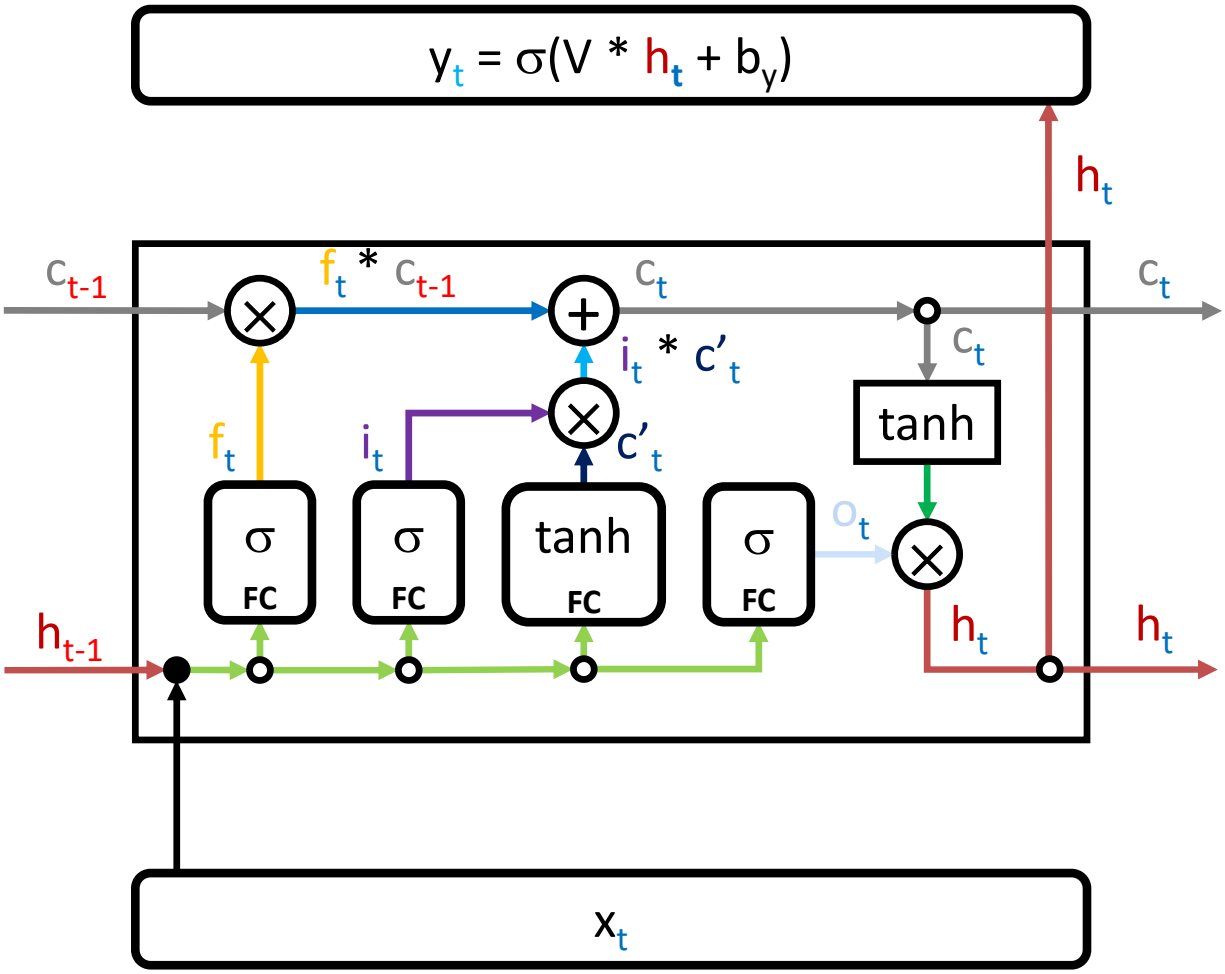
LSTM Cell/Unit



LSTM Cell/Unit

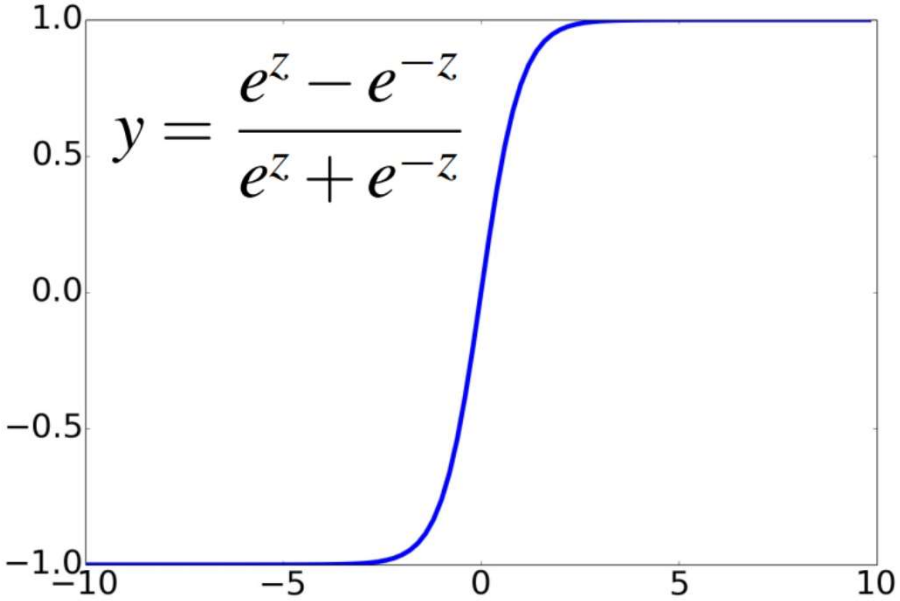


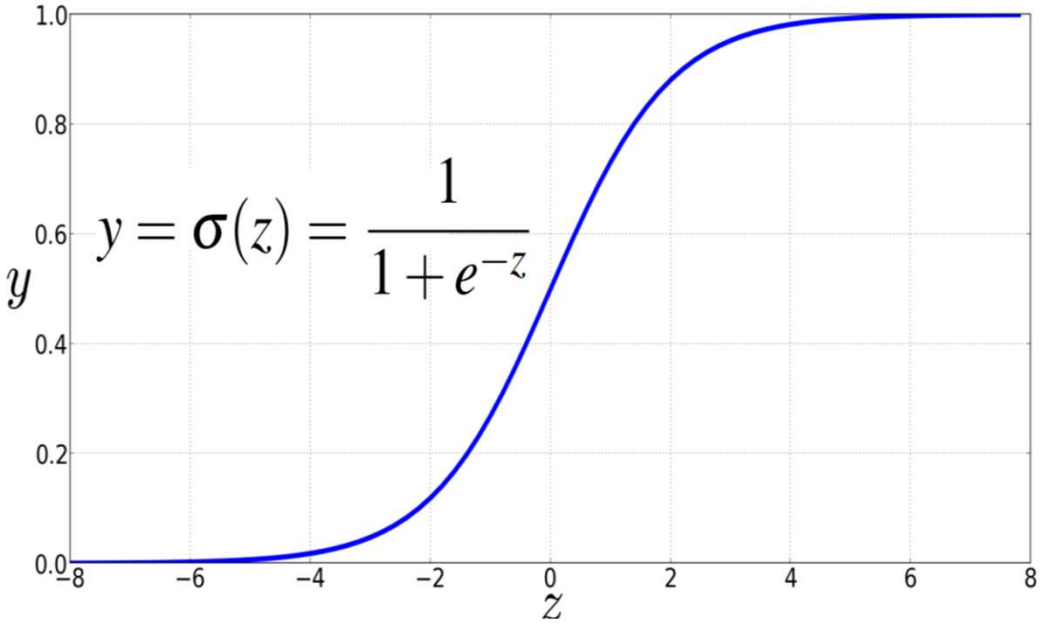
LSTM Cell/Unit



f_t = forget gate output
i_t = input gate output
c'_t = candidate values
o_t = output gate value
h_t = new hidden state
c_t = new cell state

tanh and sigmoid Activation Functions

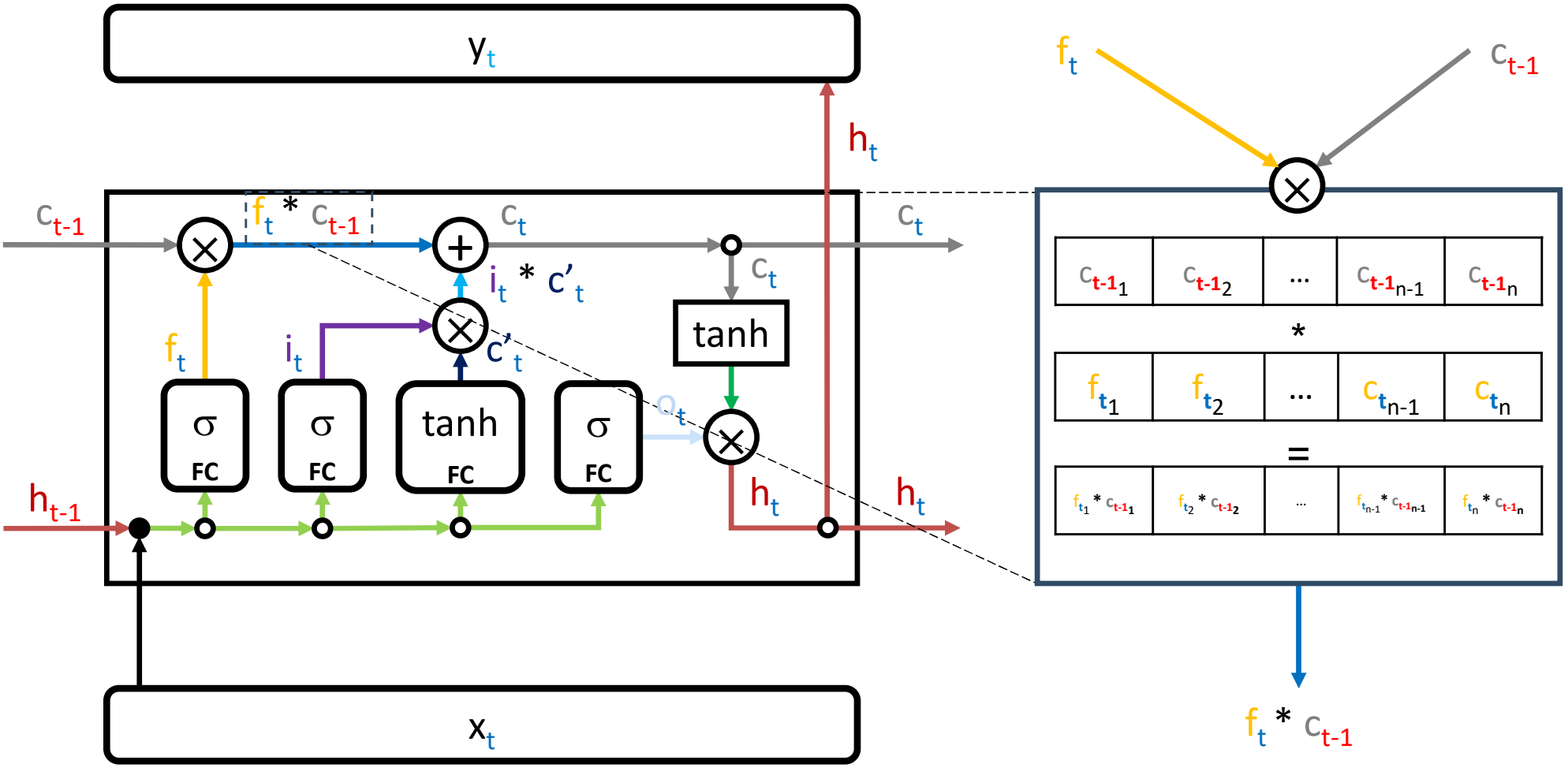




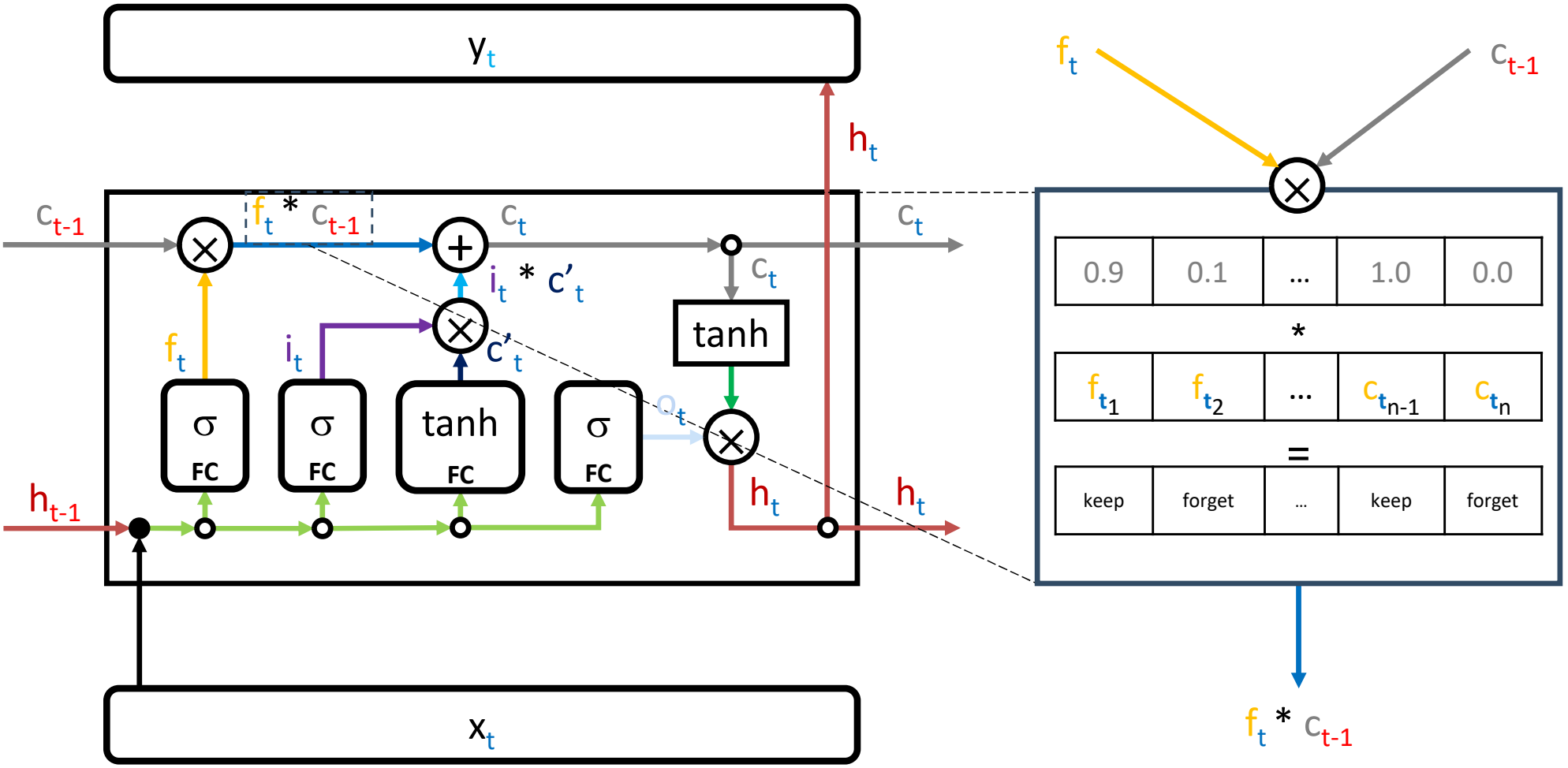
tanh
→ [-1,1] range

Sigmoid
→ [0,1] range

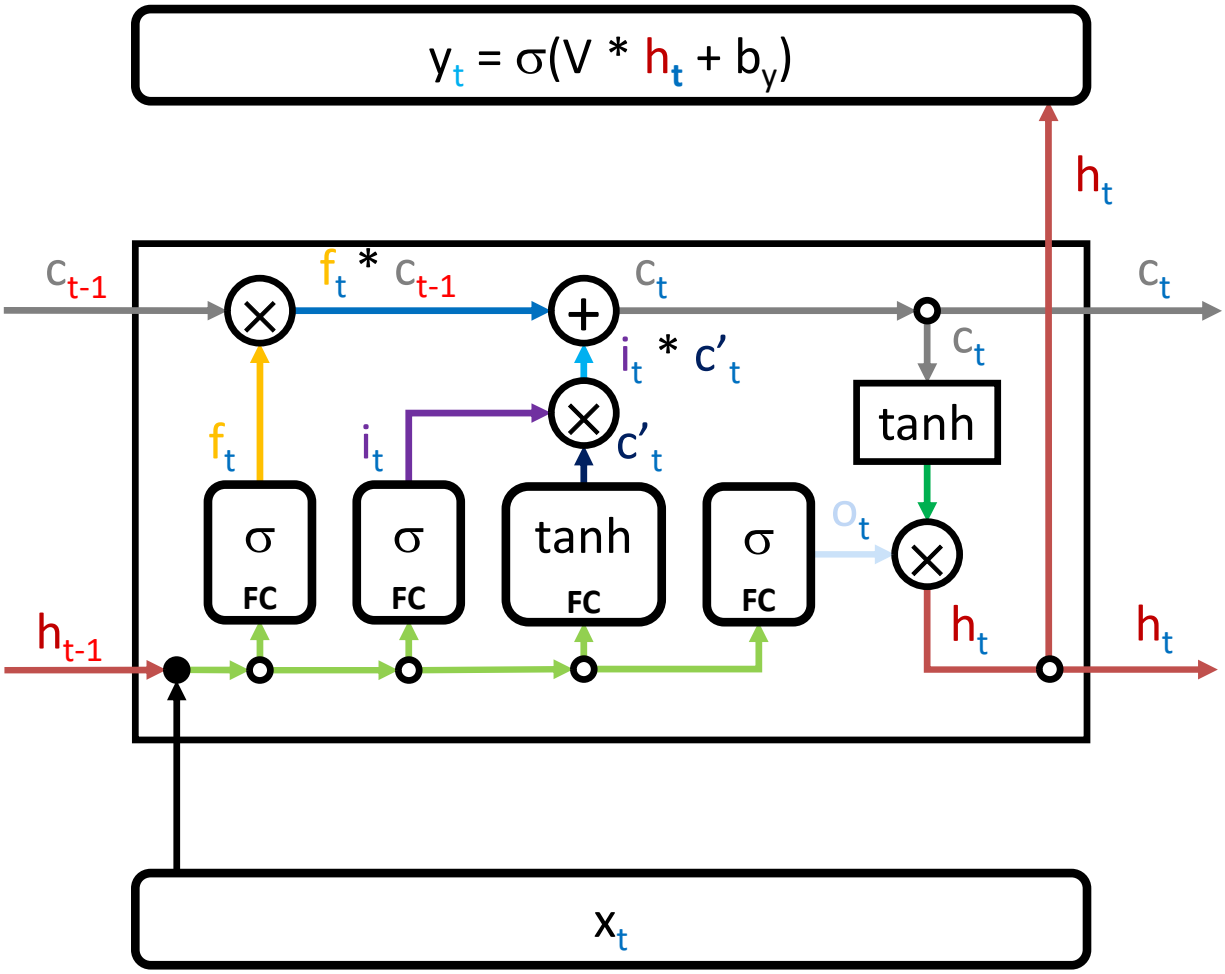
LSTM Cell/Unit



LSTM Cell/Unit



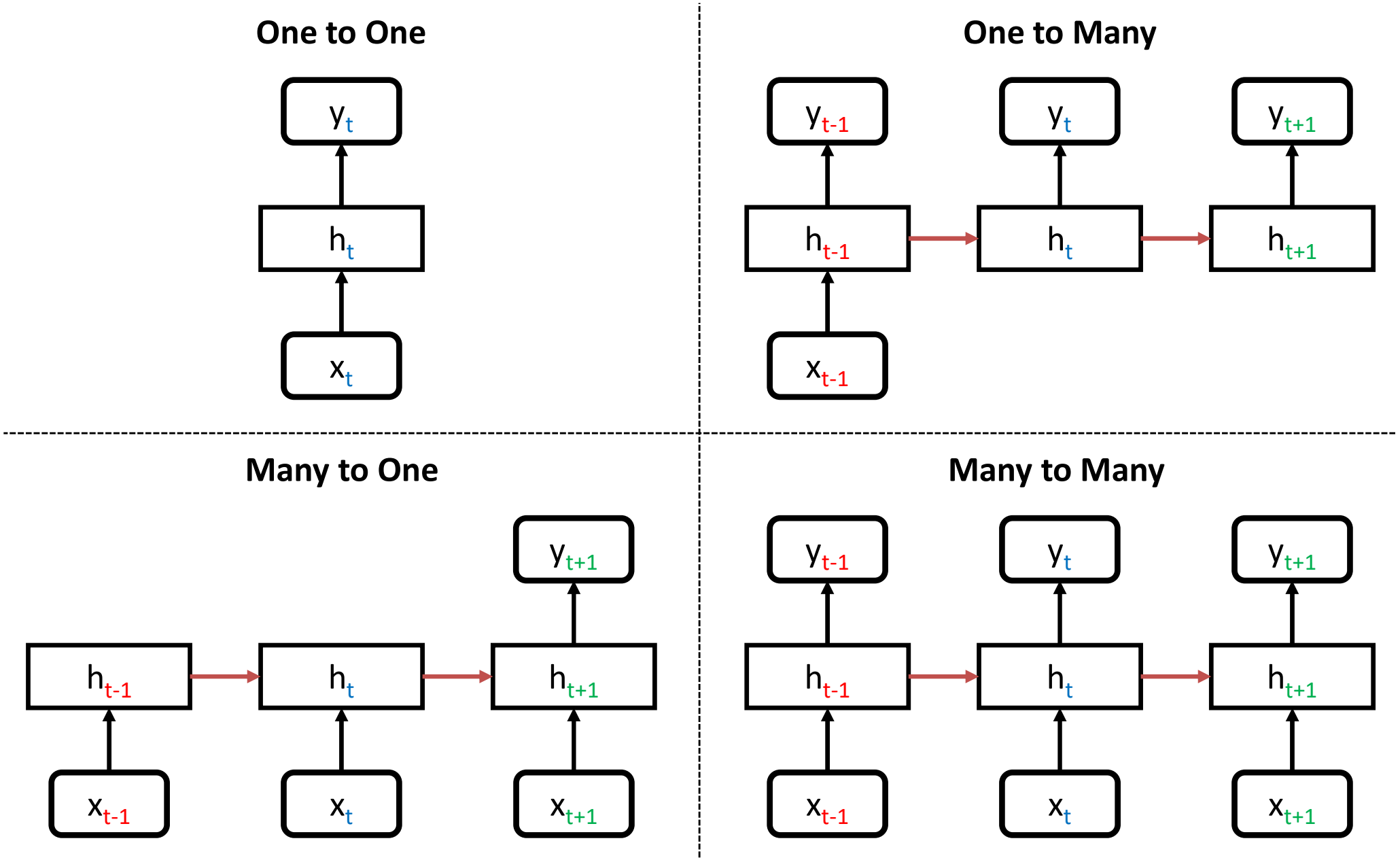
LSTM Cell/Unit



$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$
$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$
$c'_t = \tanh(W_c * [h_{t-1}, x_t] + b_c)$
$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_i)$
$h_t = o_t * \tanh(c_t)$
$c_t = f_t * c_{t-1} + i_t * c'_t$

biases (b_f, b_i, b_c, b_i, b_y) not shown on diagrams
W_f, W_i, W_c, W_i, V are neural network weight matrices

RNN/LSTM Structure Types



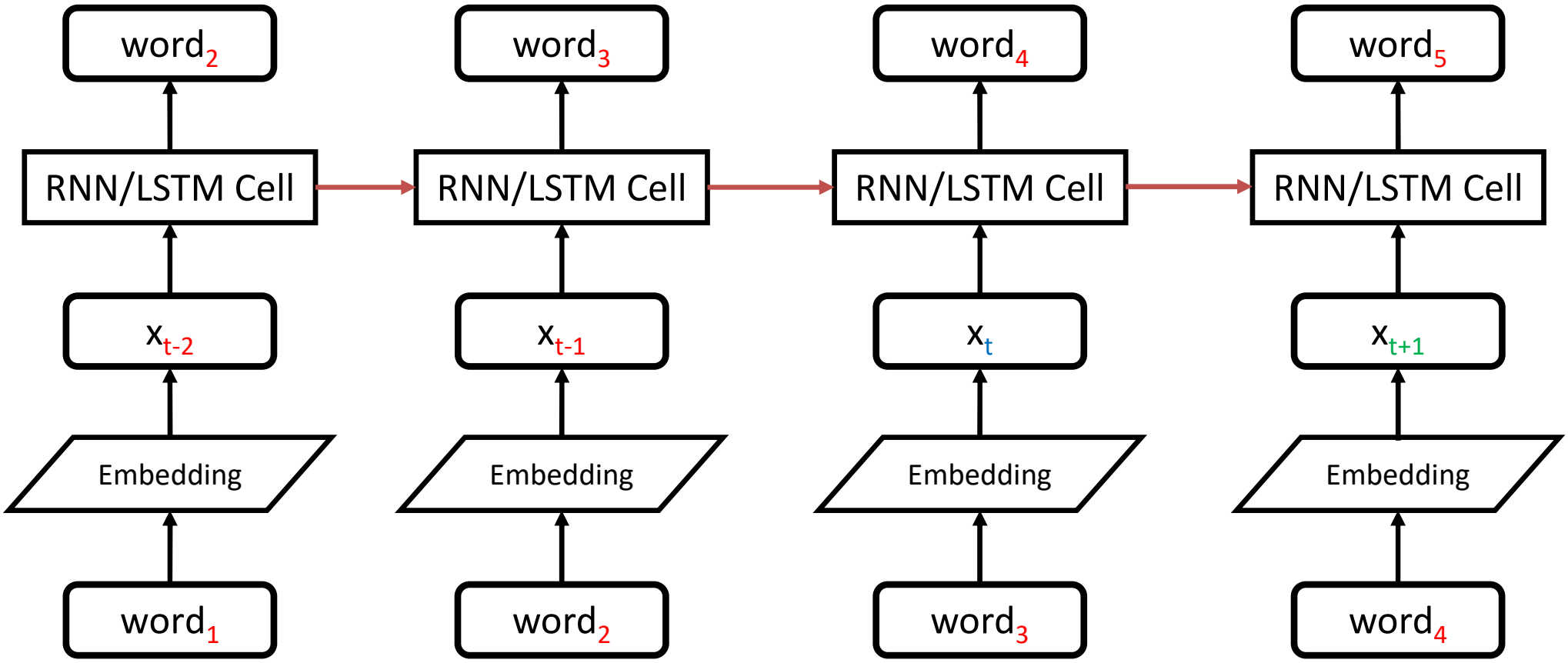
CNNs for Text Classification/Prediction

We noted before that some text categorization tasks (like sentiment analysis) could also benefit from using sequential information about the words in a text

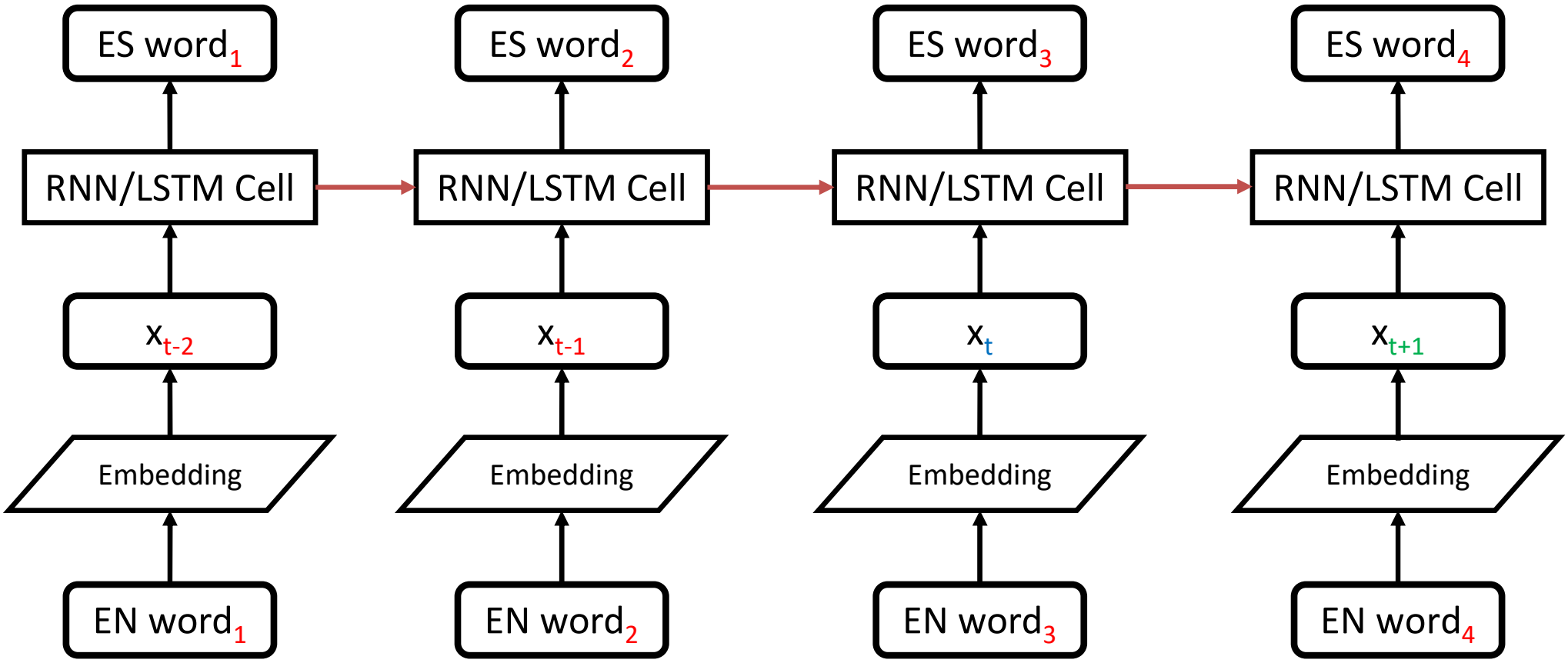
I would never buy this product again. It clearly failed under high-stress testing in my home.

I would clearly buy this product again. It never failed under high-stress testing in my home.

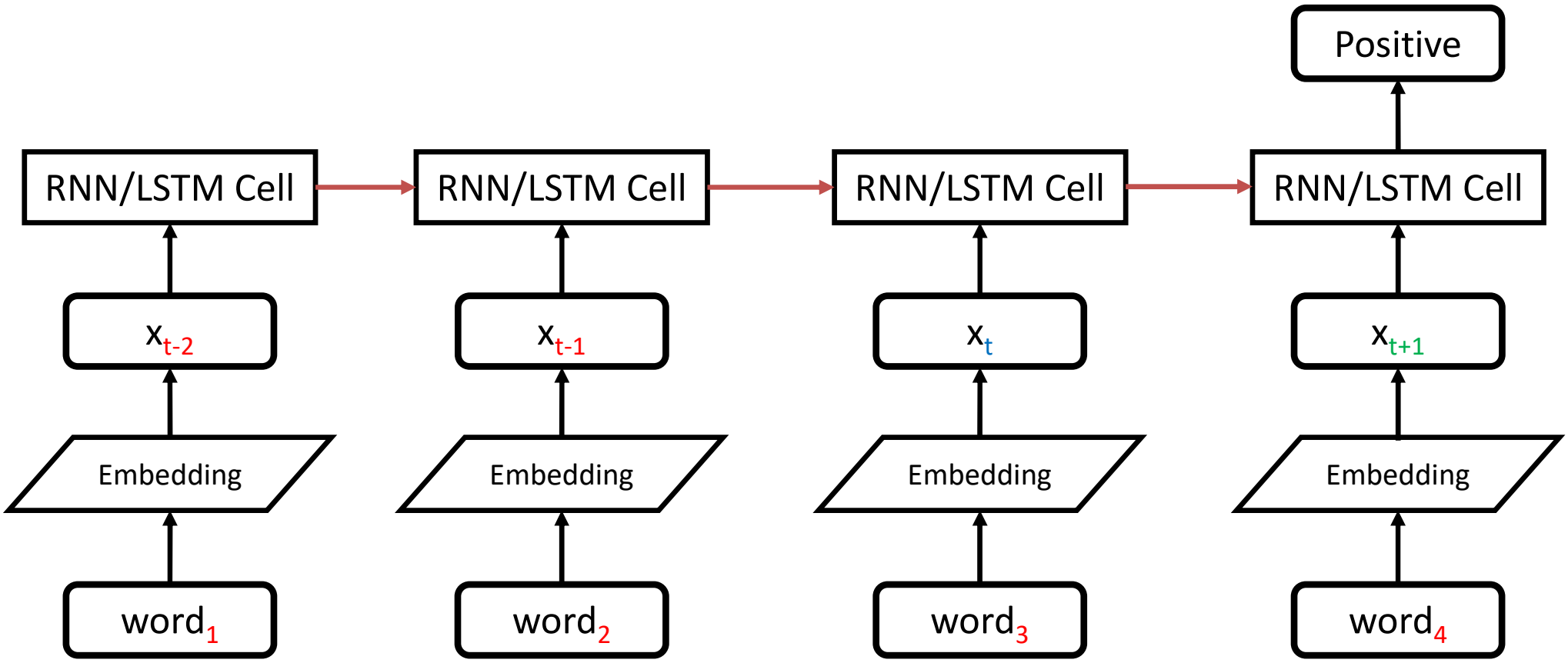
Many to Many: Word Prediction



Many to Many: Translation



Many to One: Classification



One to Many: Image Captioning