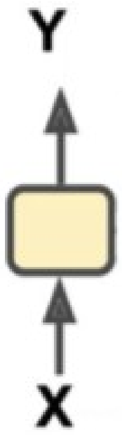
SCS-3546 Deep Learning
Session 5

Recurrent Neural Networks

One-to-One Processing

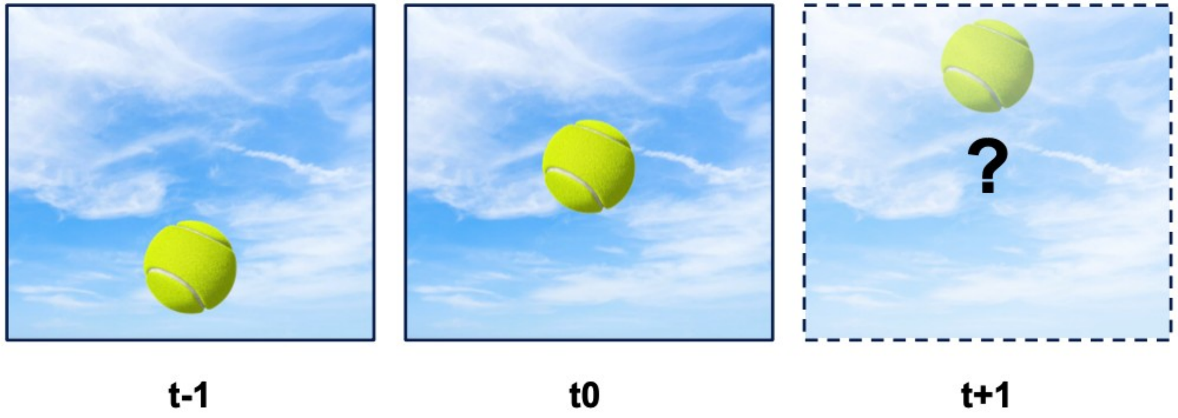


Class: “Dog”

- Fixed input & output dimensions.
- Predictions are independent; non-contextual.

What direction is the ball moving in?

Some prediction problems inherently involve **sequences**.



Sequence data and sequence predictions

A sequence:

1. an array of elements
2. a *maximum* number of elements that the array may contain (i.e. its allocated size), and
3. a logical *length* indicating how many of the allocated elements are valid (between 0 and the maximum (inclusive))

- it is not permissible to access an element at an index greater than or equal to the length
- Examples of sequence data :
  - DNA, protein,
  - customer purchase history
  - web surfing history

Sequence data and sequence predictions

Time-series data:

- a collection of integer values collected over a period of time
- indexed (or listed) in time order
- values are usually taken at regular intervals (e.g. minute, hour, or day)

Important considerations:

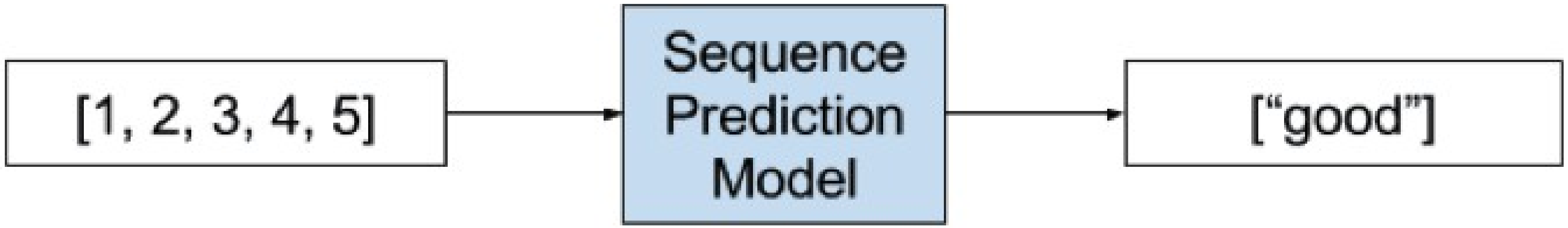
- the points in the dataset are reliant on the other points in the dataset
- A Timeseries, such as a stock price or sensor data

Sequence prediction

- **Weather Forecasting**
  - Given a sequence of observations about the weather over time, predict the expected weather tomorrow
- **Stock Market Prediction**
  - Given a sequence of movements of a security over time, predict the next movement of the security.
- **Product Recommendation**
  - Given a sequence of past purchases of a customer, predict the next purchase of a customer.

Sequence Classification

predicting a class label for a given input sequence.

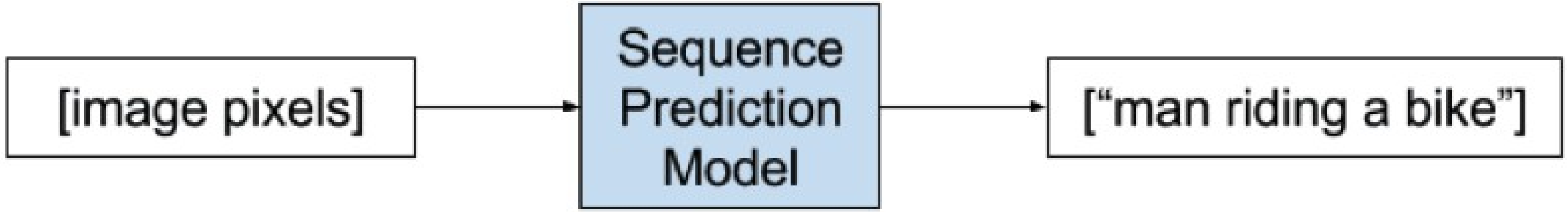


- **DNA Sequence Classification**
  - Given a DNA sequence of ACGT values
  - predict whether the sequence codes for a coding or non-coding region
- **Anomaly Detection**
  - Given a sequence of observations, predict whether the sequence is anomalous or not
- **Sentiment Analysis**
  - Given a sequence of text such as a review or a tweet, predict whether sentiment of the text is positive or negative.

Sequence Generation

Either:

- From a sequence -> generating a new output sequence that has the same general characteristics as other sequences in the corpus
- from a single observation as input -> generate a sequence



- **Image Caption Generation**
  - Given an image as input, generate a sequence of words that describe an image

- **Text Generation**
  - Given a corpus of text, generate new sentences or paragraphs of text

- **Handwriting Prediction**
  - Given a corpus of handwriting examples, generate handwriting for new phrases (with properties of handwriting in the corpus)
- **Music Generation**
  - Given a corpus of examples of music, generate new musical

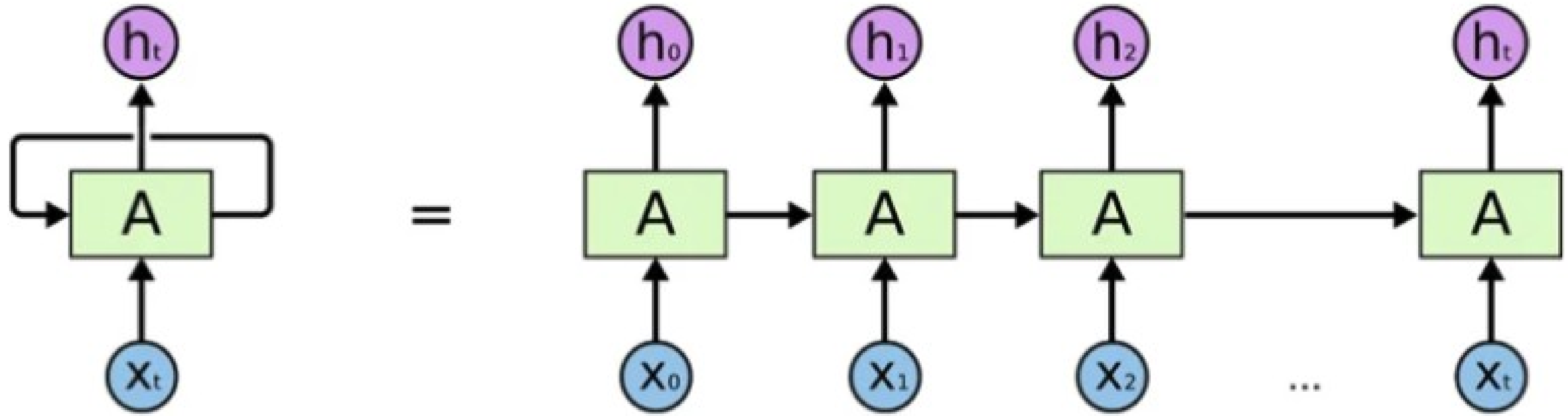
Sequence-to-Sequence Prediction

- involves predicting an output sequence given an input sequence

To succeed at sequence modelling, we need to be able to:

1. Handle variable sequence lengths
2. Maintain a **state** or **memory** that tracks temporal dependencies
3. Be sensitive to the **order** of information

Intro to RNNs



Recurrent neural networks (RNN)

- a class of neural networks
- powerful for **modeling sequence data**
  - time series
  - natural language

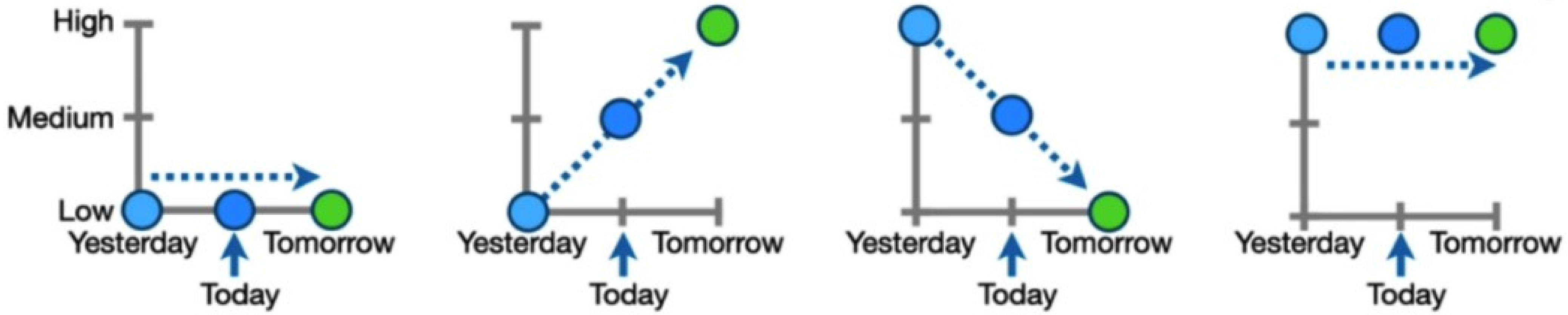
Ordinary feedforward neural networks

- only meant for data points that are independent of each other

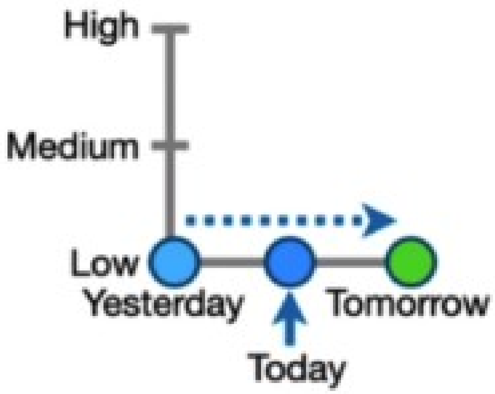
RNNs

- Capable for being used with data that is in a sequence
  - **one data point depends upon the previous data point**
- incorporate the dependencies between data points
- Concept of "memory" that helps store the states or information of previous inputs to generate the next output of sequence

How can we ensure that a neuron's prediction at a given timestep t takes into account the inputs it saw at previous timesteps?



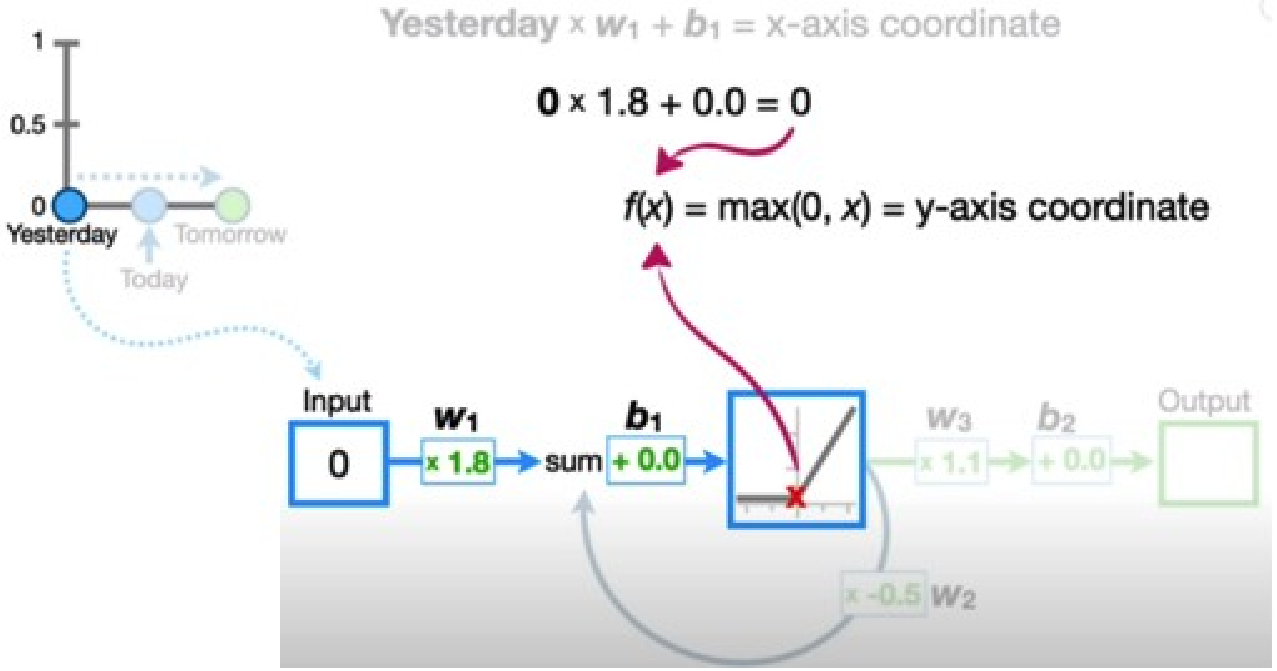
Scenario 1

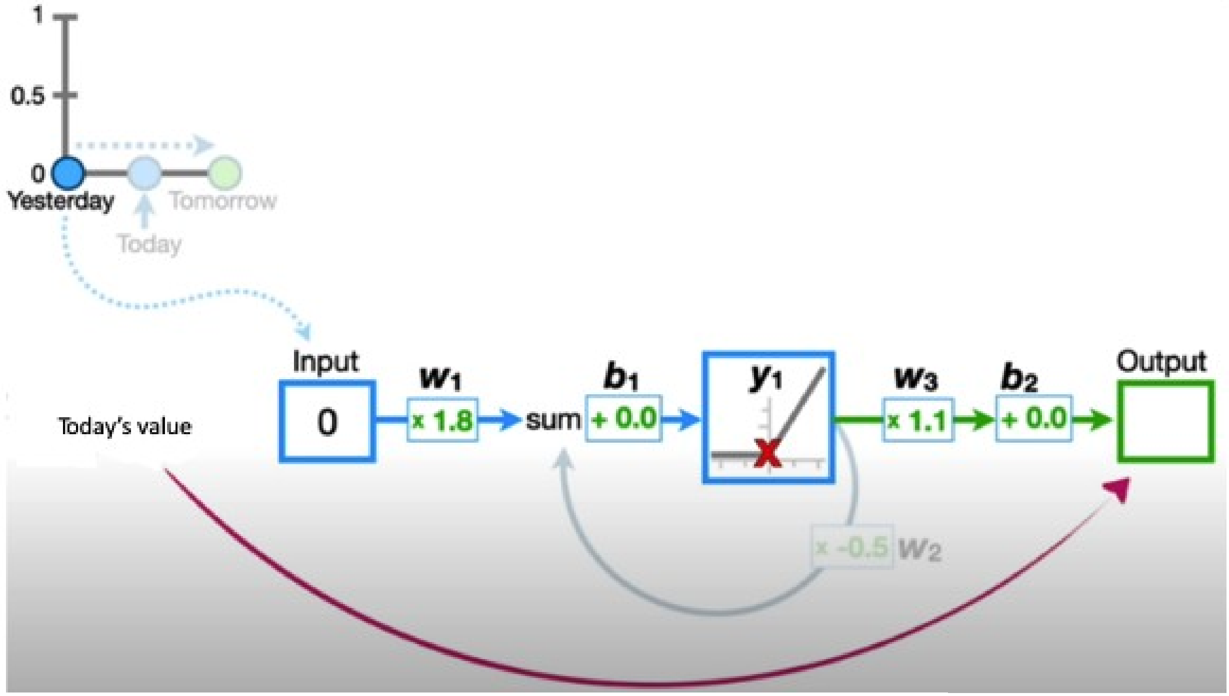


[x(yesterday), x(Today)] -> Y(Tomorrow)

[0, 0]

Let's predict using an ordinary DNN model





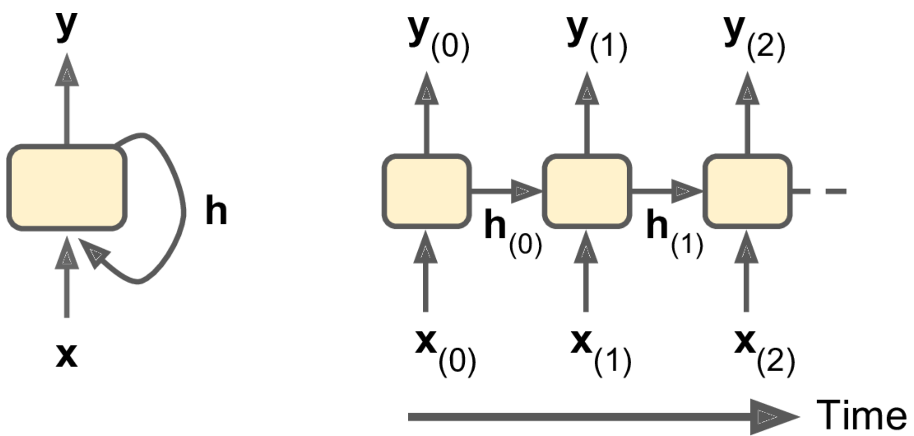
But, we need to predict tomorrow's value:

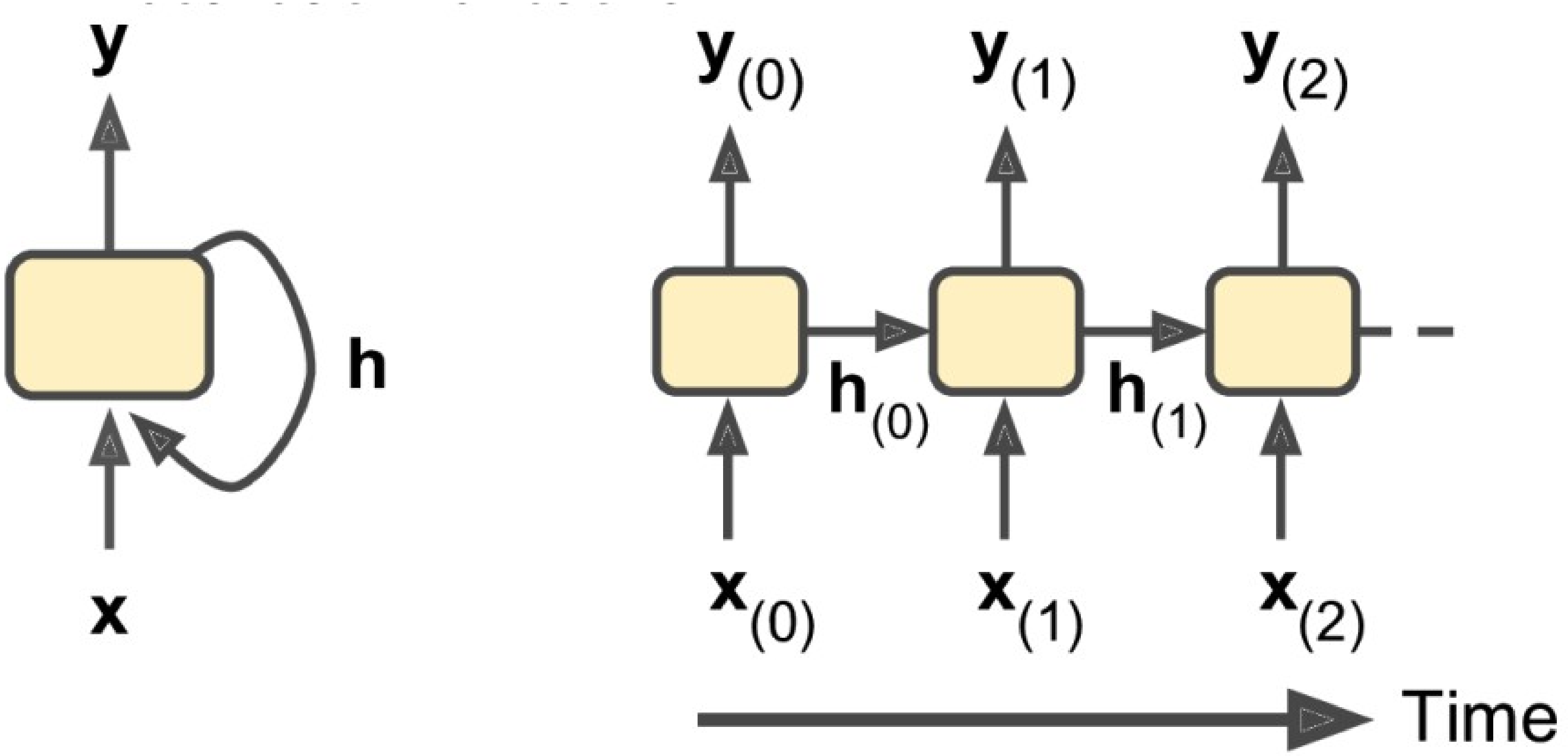
how to incorporate today's value to predict tomorrow's value?

Memory

What makes RNNs different from other networks

- maintain a hidden internal state $h(t)$ that serves as a form of memory
- Internal state $h(t)$ depends on:
  - the input $x(t)$ at time step t
  - the internal state in the previous timestep : $h(t-1)$
  - the information cycles through a loop to the middle hidden layer
  - The middle layer 'h' can consist of multiple hidden layers, each with its own activation functions





- The output of a recurrent neuron at time t is a function of all the inputs from previous times
- In general, a cell's state $h_{(t)}$ is a function of some inputs $x_{(t)}$ and its state at the previous time step: $h_{(t)} = f(h_{(t-1)}, x_{(t)})$
- The network's output $y_{(t)}$, is also a function of the previous $y_{(t-1)}$ and $x_{(t)}$.

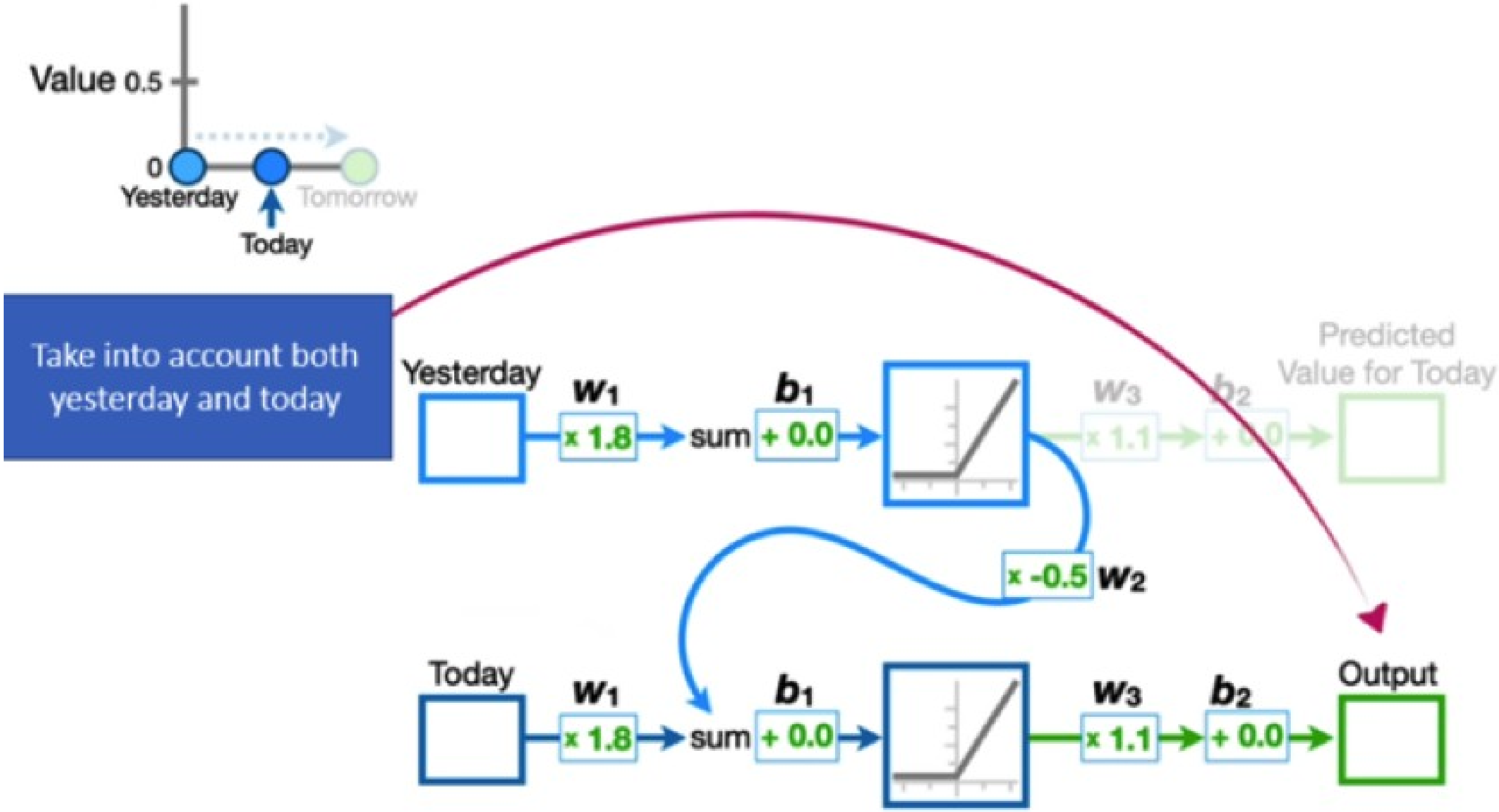
Feed-forward through an RNN:

$$h_{(t)} = \phi\left(W_{hh}^{T} h_{(t-1)} + W_{xh}^{T} x_{(t)}\right)$$

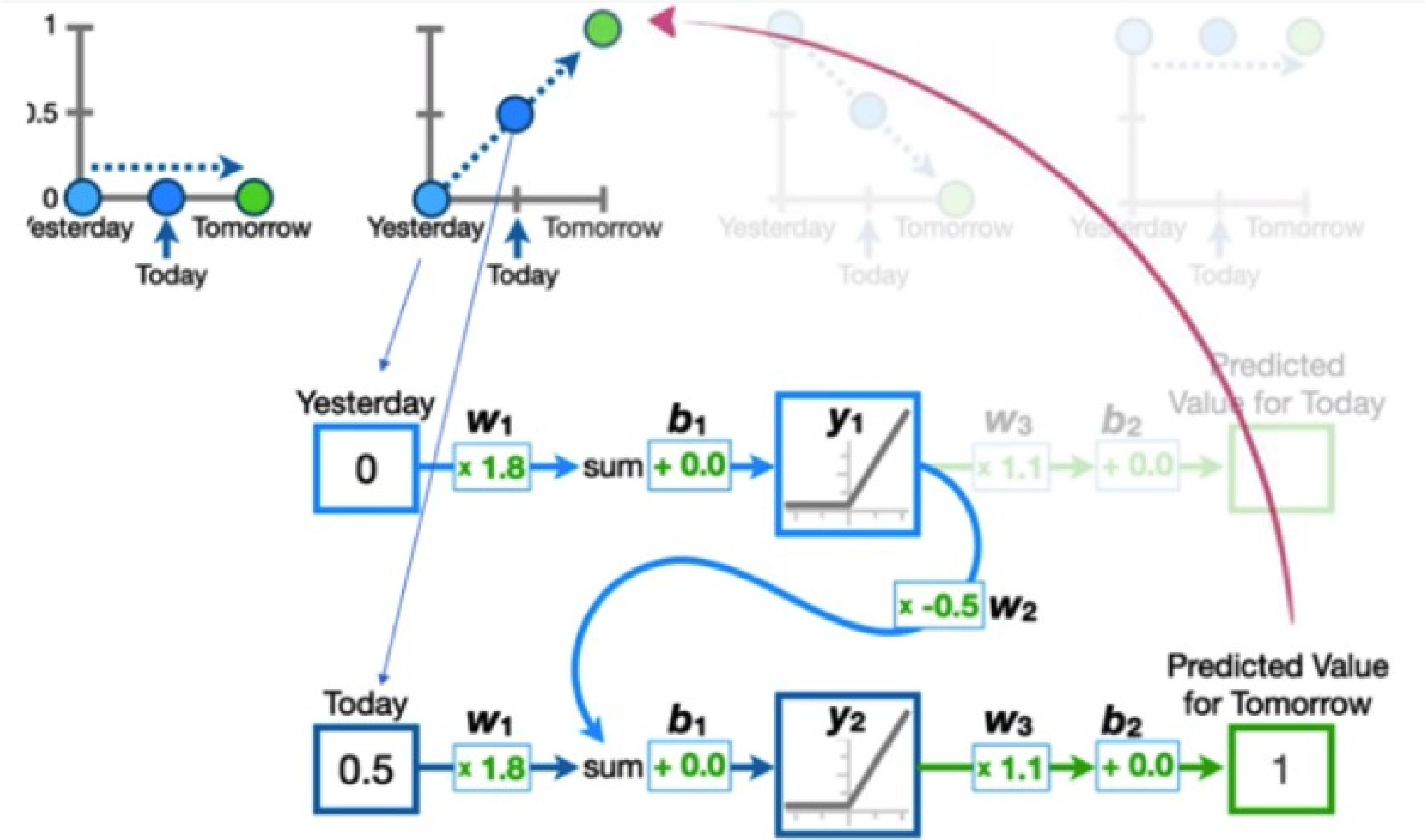
$$y_{(t)} = \phi\left(W_{hy}^{T} h_{(t)}\right)$$

- W_{xh} = a weights matrix that connects the input to the hidden cell state
- W_{hh} = a weights matrix that connects the hidden cell state to itself in the previous timestep
- W_{hy} = a weights matrix connecting the hidden state to the predicted output
- $\phi(.)$ = a nonlinear activation function

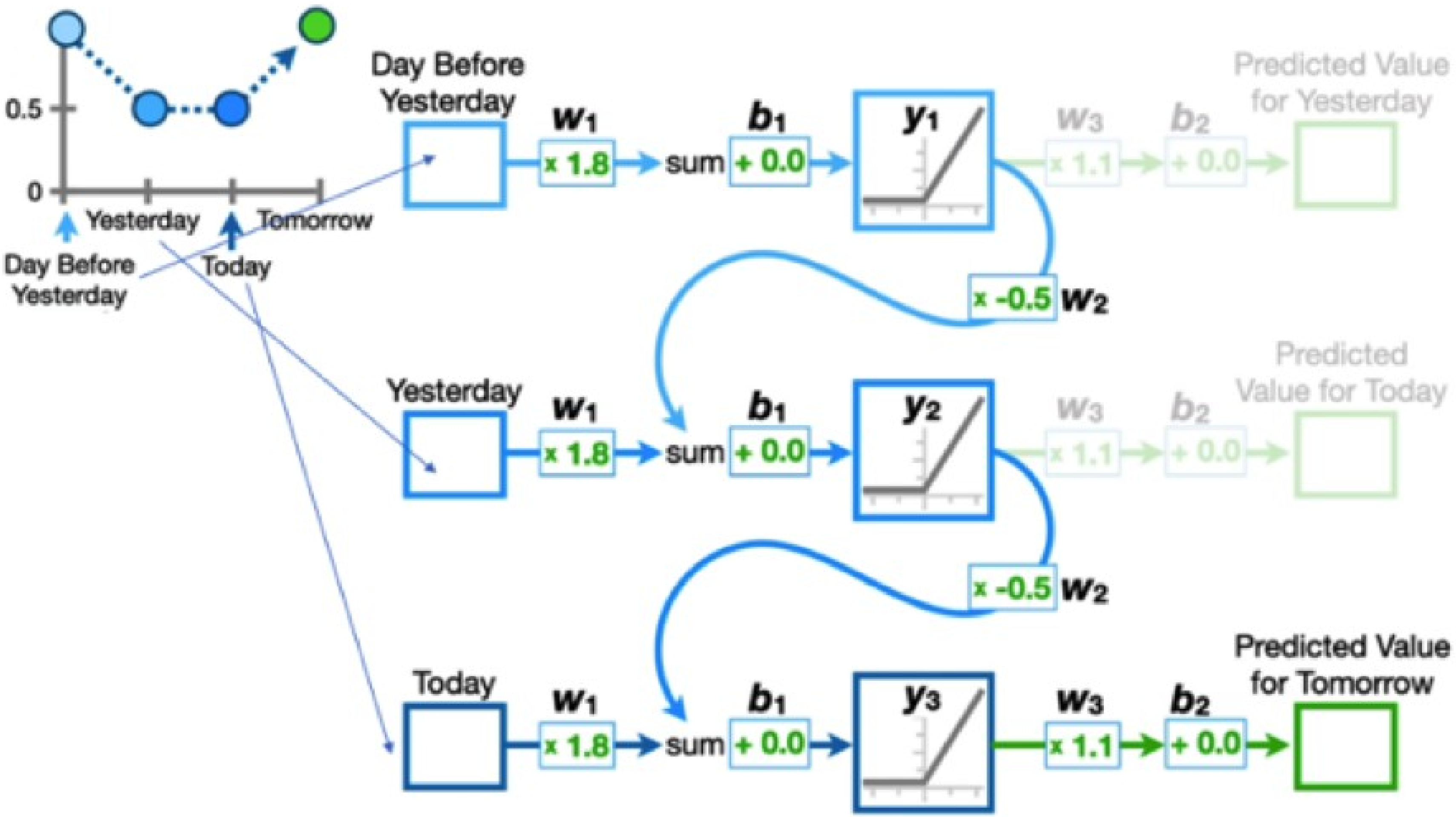
Prediction using RNNs



Review the same with scenario 2:



How about if we have more data points (longer sequence)?

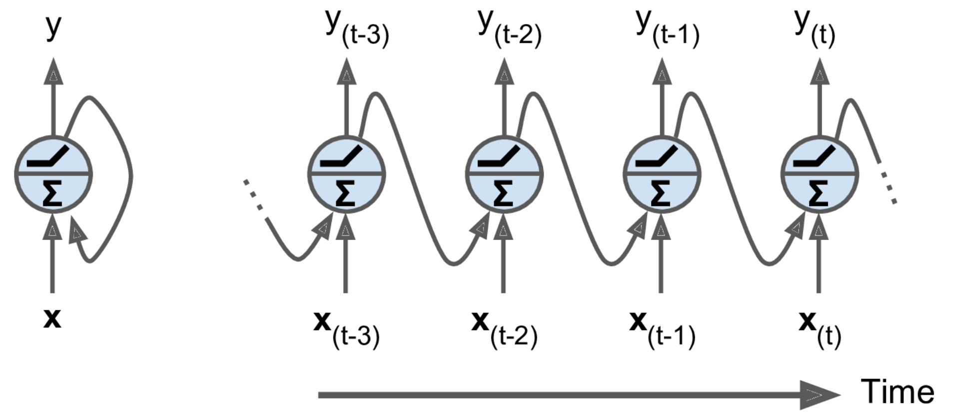


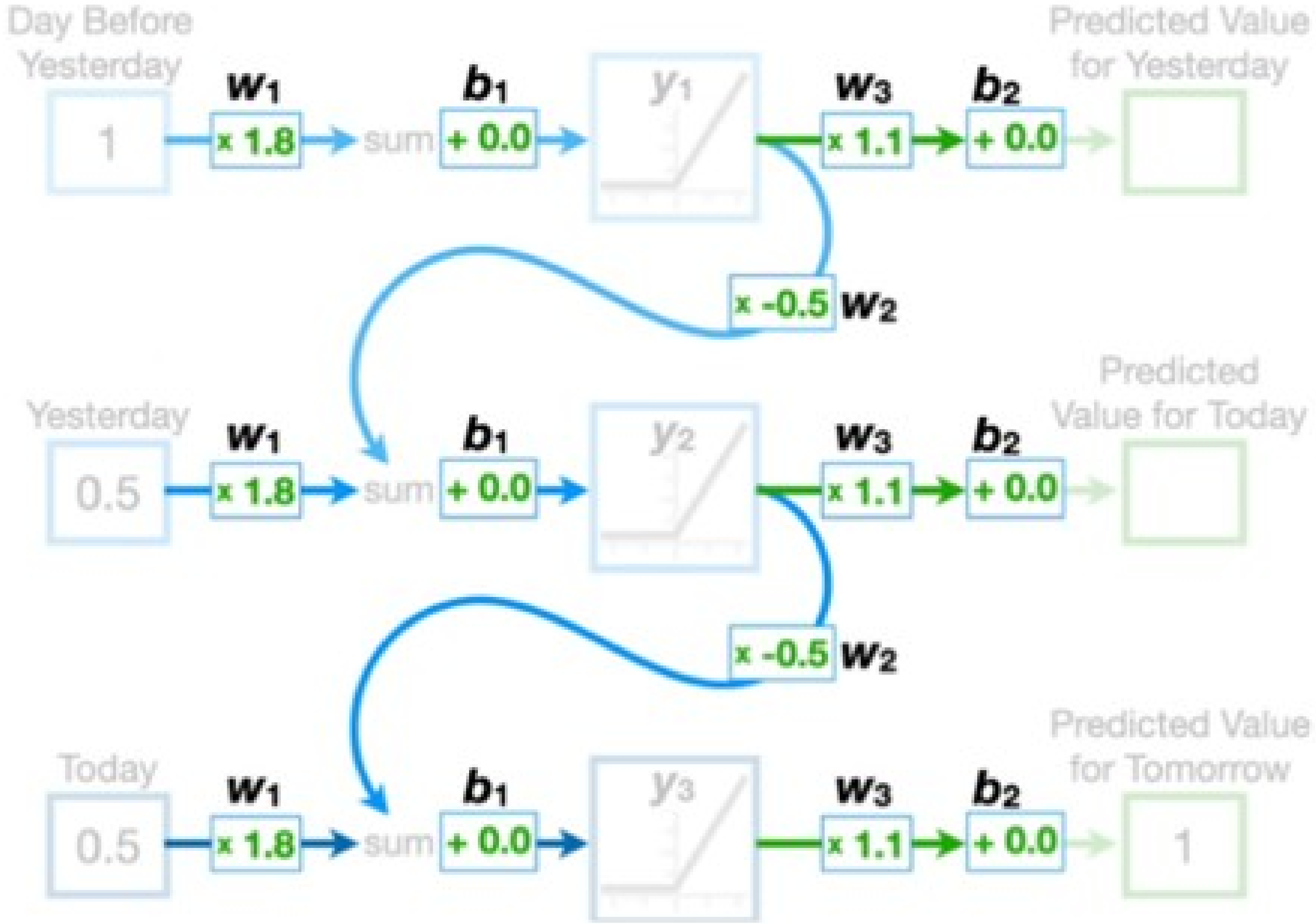
feed each element of the sequence into the RNN (one at a time)

- Each 'steps' is a timestep t
- the network would see the input vector $x(0)$ during timestep $t0$, the vector $x(1)$ at timestep $t1$, and …
- $x(0)$ a single value at time t
  - A data point - e.g. stock price
  - A vector of values- e.g. the pixels of a video frame

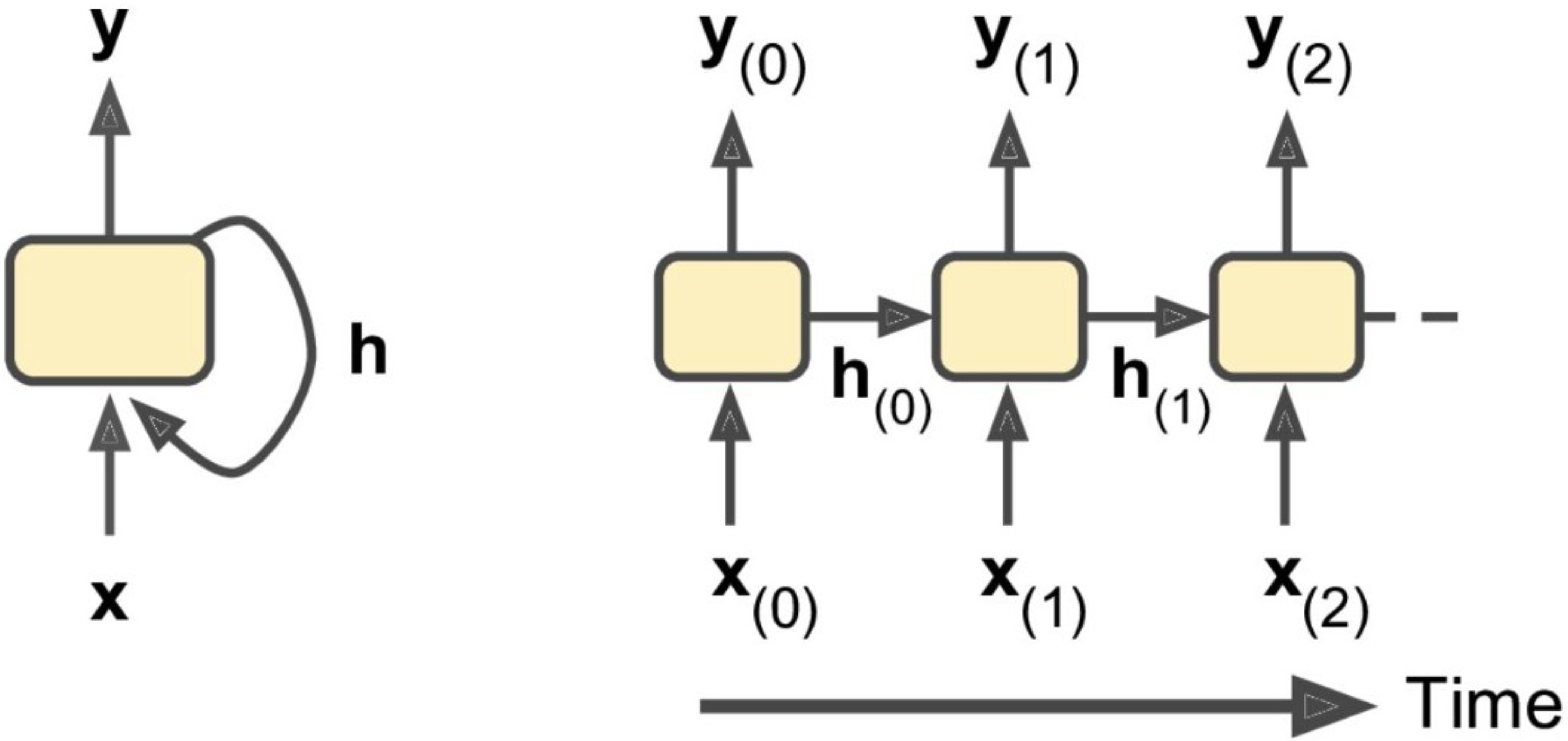
RNNs combine the input vector with their state vector with a fixed (but learned) function to produce a new state vector

- all of the weights and biases are shared
- Each hidden layer has the same parameters
- So, instead of creating multiple hidden layers, it will create one and loop over it as many times as required




Day Before Yesterday
1
w1
x 1.8
sum
b1
+ 0.0
y1
w3
x 1.1
b2
+ 0.0
Predicted Value for Yesterday
x -0.5
w2
Yesterday
0.5
w1
x 1.8
sum
b1
+ 0.0
y2
w3
x 1.1
b2
+ 0.0
Predicted Value for Today
x -0.5
w2
Today
0.5
w1
x 1.8
sum
b1
+ 0.0
y3
w3
x 1.1
b2
+ 0.0
Predicted Value for Tomorrow
1

A Recurrent Unit



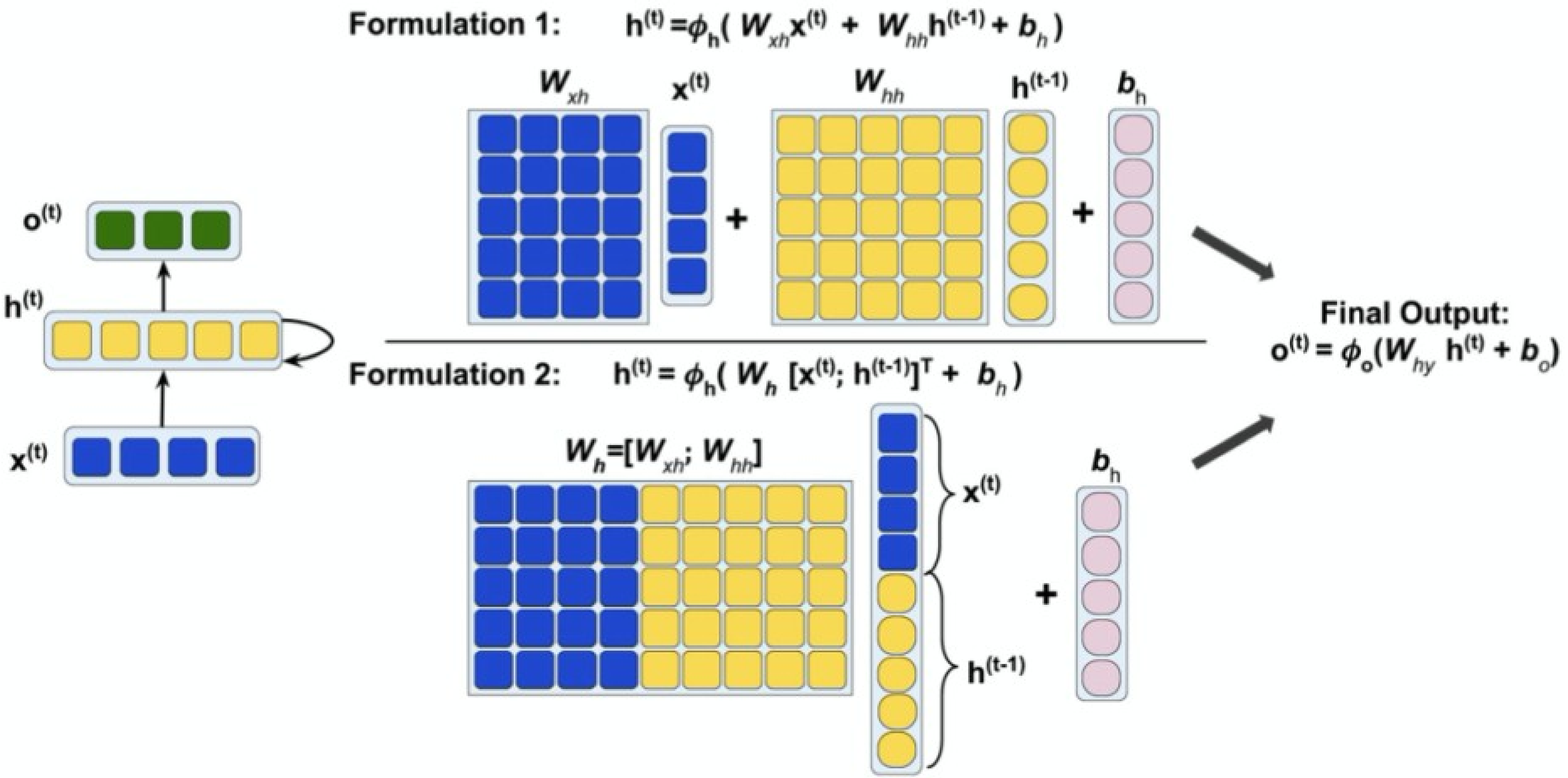
Each recurrent neuron has two sets of weights:

- Wx = for the inputs $x(t)$
- Wy = for the outputs of the previous time step, $y(t–1)$:
- Wx and Wy -> two weight matrices of all the weight vectors for all neurons
- The **output vector of the whole recurrent layer** is computed as:

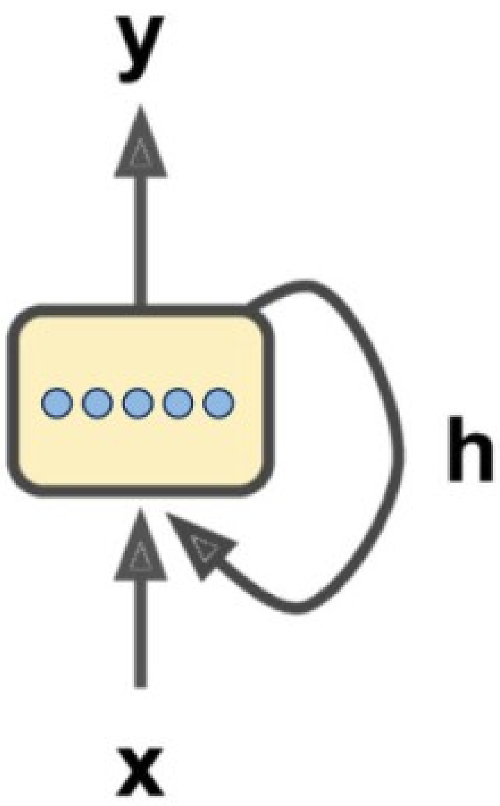
$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x{}^T \mathbf{x}_{(t)} + \mathbf{W}_y{}^T \mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

- b = bias vector
- ϕ(phi) = the nonlinear activation function

'Compact' Weights Representation



A Recurrent Layer



hidden internal state

$$h_{(t)} = \phi\left(W_{hh}^{T} h_{(t-1)} + W_{xh}^{T} x_{(t)}\right)$$

$$y_{(t)} = \phi\left(W_{hy}^{T} h_{(t)}\right)$$

For each layer of recurrent neurons, we now need to learn 3 weight matrices during training.

Note: weights are updated between training batches, **but are fixed constants** between time steps.

Compute a **recurrent layer's output** :

- The output vector of the whole recurrent layer:

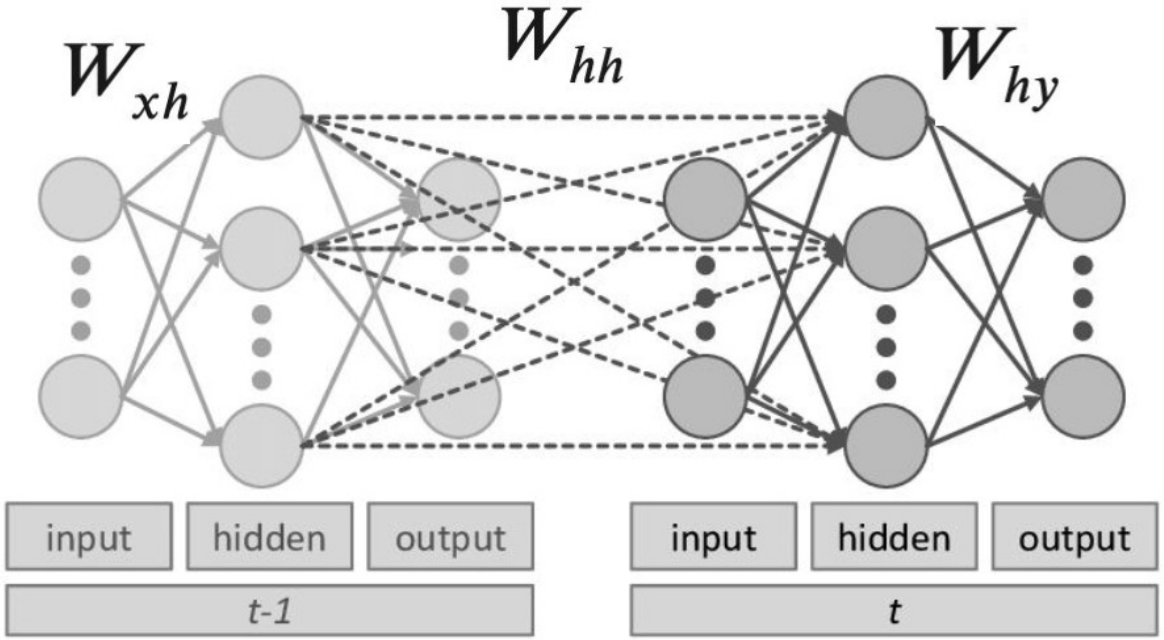
$$\mathbf{Y}_{(t)} = \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \mathbf{Y}_{(t-1)}\mathbf{W}_y + \mathbf{b}\right)$$
$$= \phi\left(\left[\mathbf{X}_{(t)} \quad \mathbf{Y}_{(t-1)}\right]\mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix} \mathbf{W}_x \\ \mathbf{W}_y \end{bmatrix}$$

  - $Y(t)$ = an $m \times n_{neurons}$ matrix
    - the layer's outputs at time step t for each instance in the mini-batch (m is the number of instances in the mini-batch)
  - $X(t)$ = $m \times n_{inputs}$ matrix (the inputs for all instances)
  - W_x = $n_{inputs} \times n_{neurons}$ matrix containing the connection weights of the current step.
  - W_y = $n_{neurons} \times n_{neurons}$ matrix (the connection weights for the outputs of the previous time step)
  - b = a vector of size $n_{neurons}$ (each neuron's bias term)

weight matrices W_x and W_y

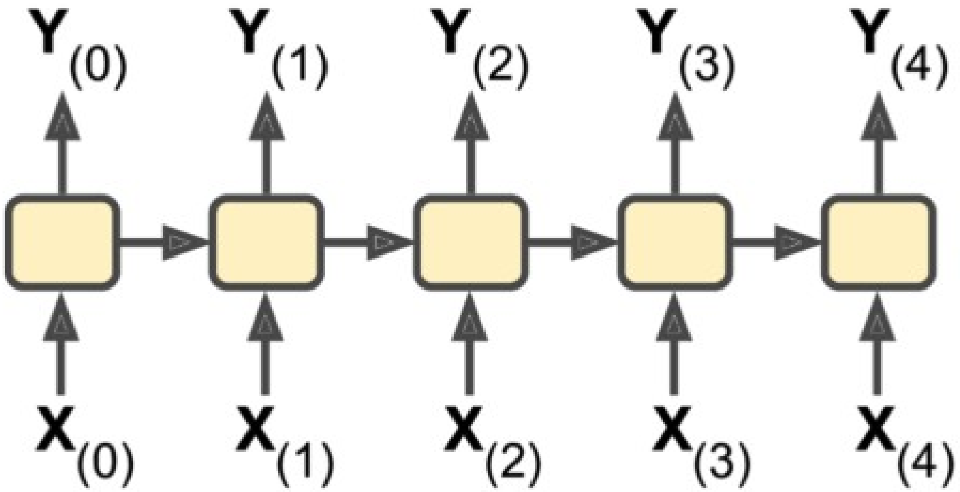
- often concatenated vertically into a single weight matrix
- The notation $[X(t) Y(t−1)]$ represents the horizontal concatenation of the matrices $X(t)$ and $Y(t−1)$.

Visualizing RNN Connections

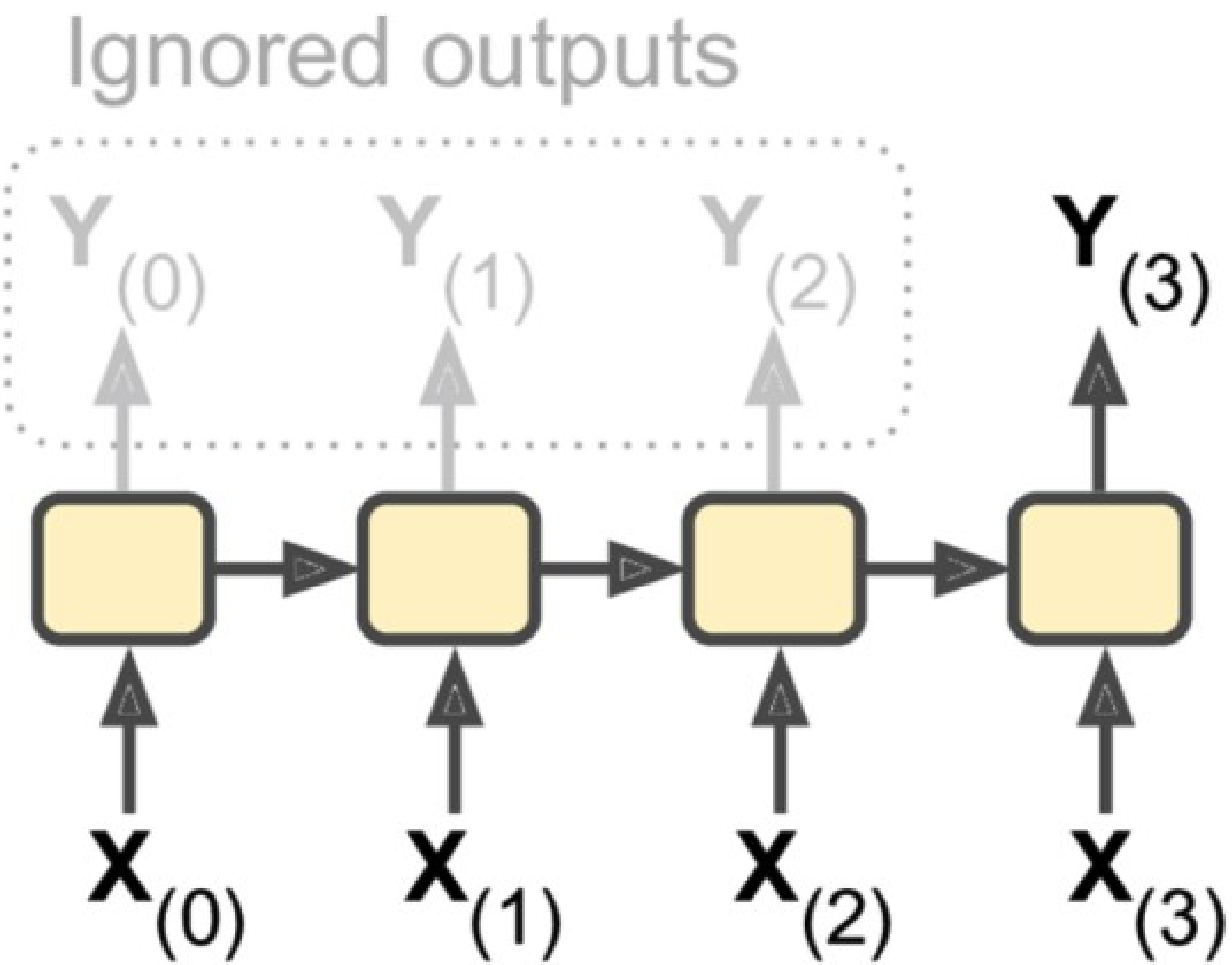


Prediction using RNN

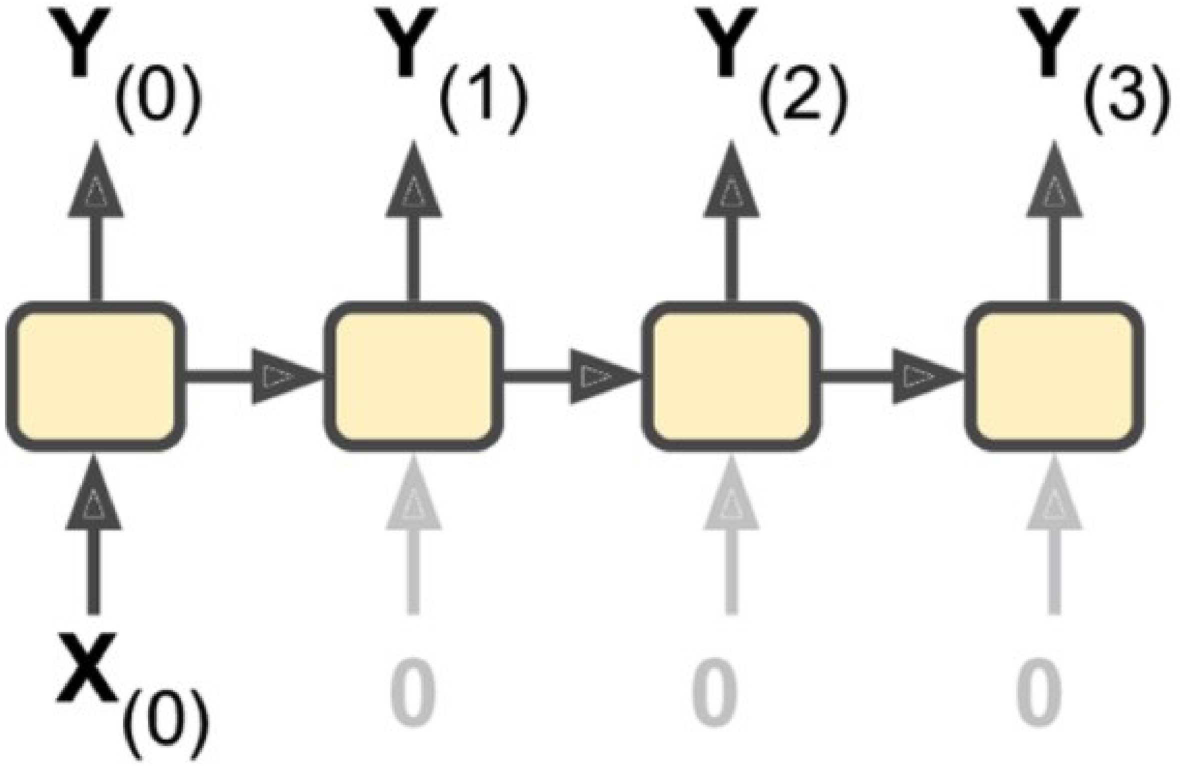
- **Many-to-many sequence prediction:**
  - simultaneously take a sequence of inputs and produce a sequence of outputs



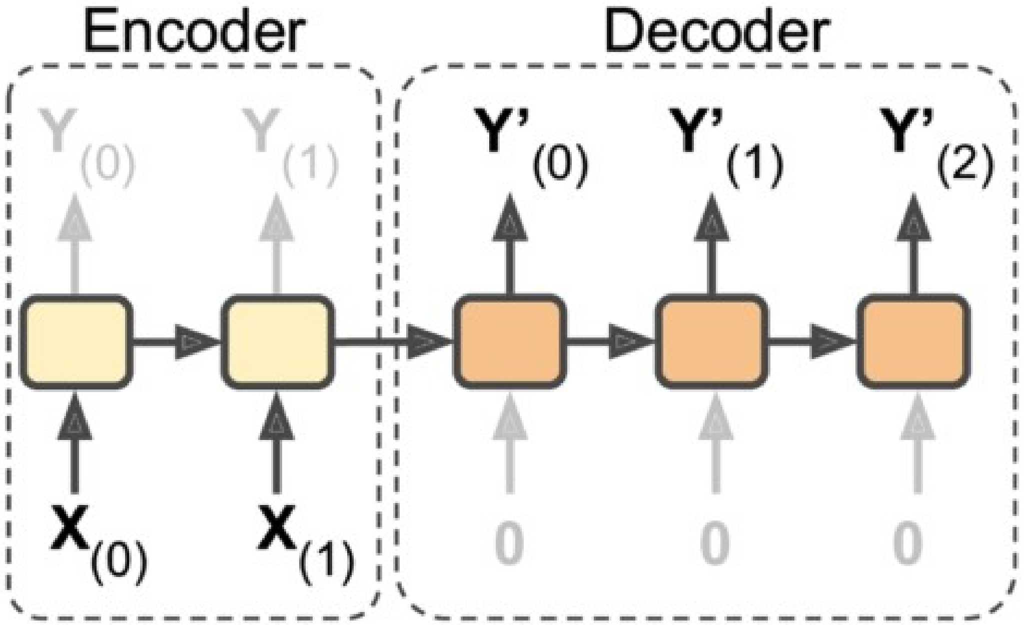
- **Many-to-one prediction:**
  - take a sequence of inputs, and ignore all outputs except for the last one,
  - sequence of words (movie review) and predict a sentiment score



- **One-to-many prediction:**
  - takes a single input at the first time step and zeros for all others
  - ingests an image on the first timestep, and exports a text description of that image across multiple subsequent time steps



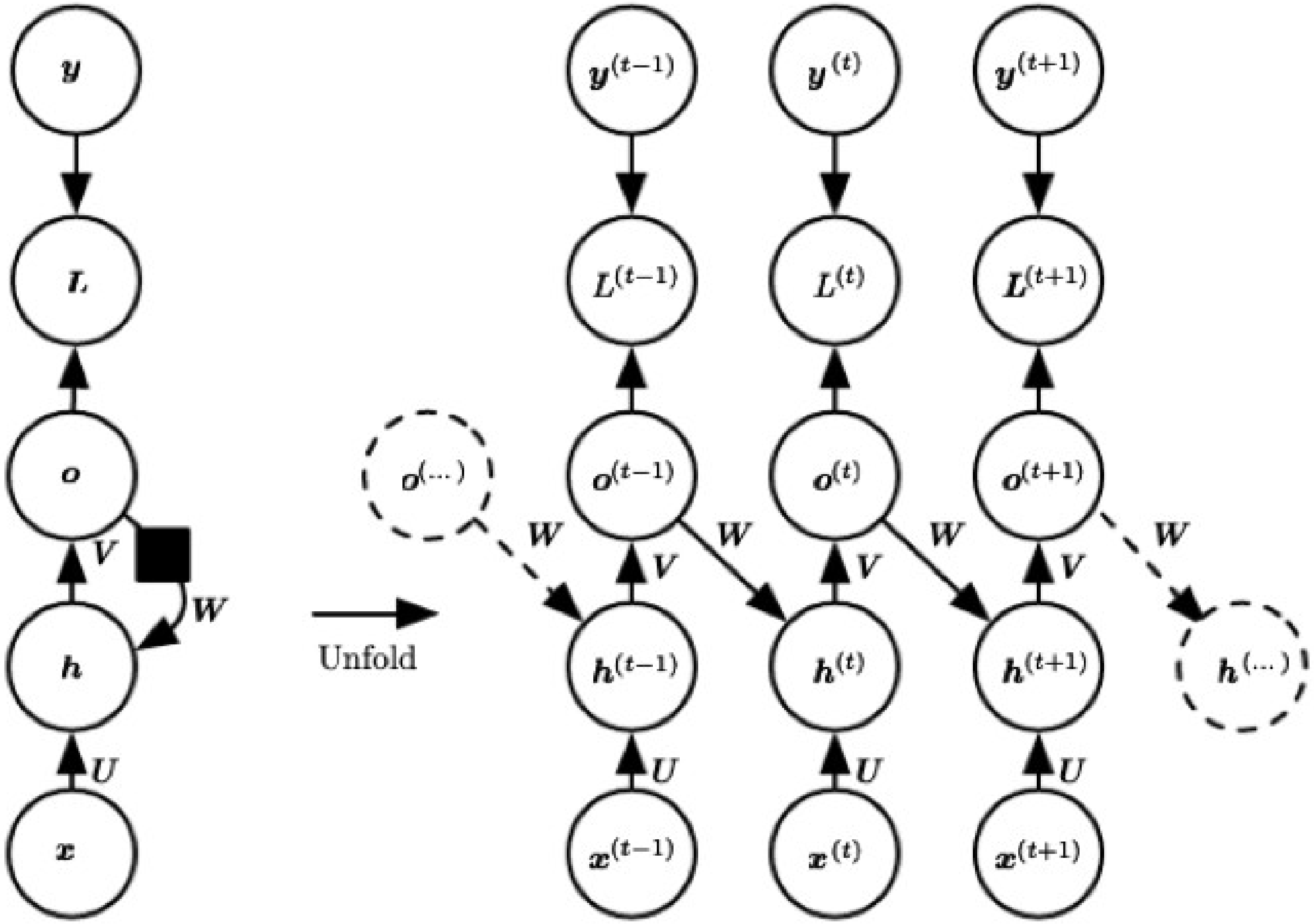
- **Encoder-Decoder**:
  - many-to-one network (e.g. encoder) + a one-to-many network (e.g. decoder)
  - two-step model
  - sentence translation



RNN Variants

Single recurrence

feedback connection from the output to the hidden layer instead of the hidden cell state to itself

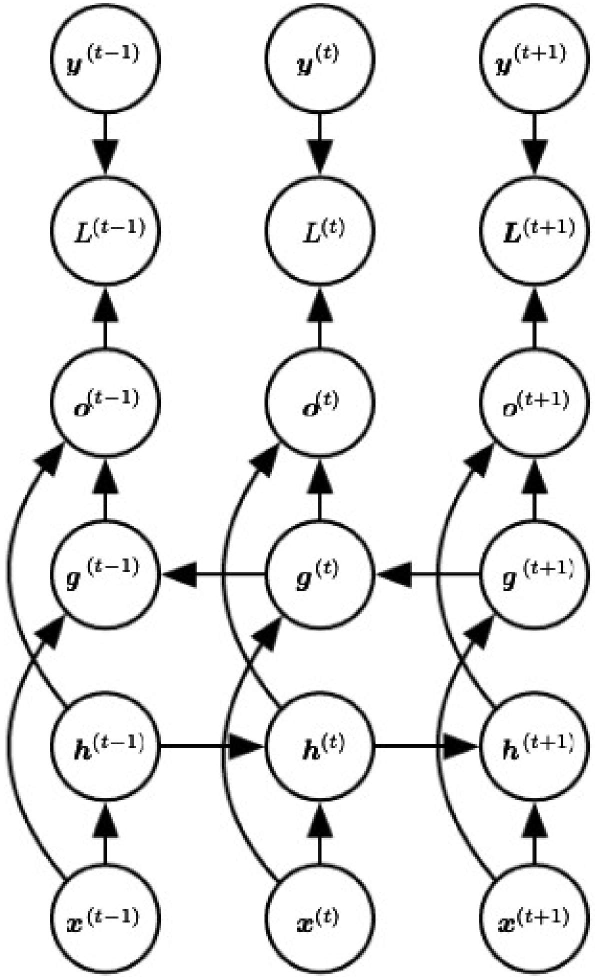


Bidirectional RNNs

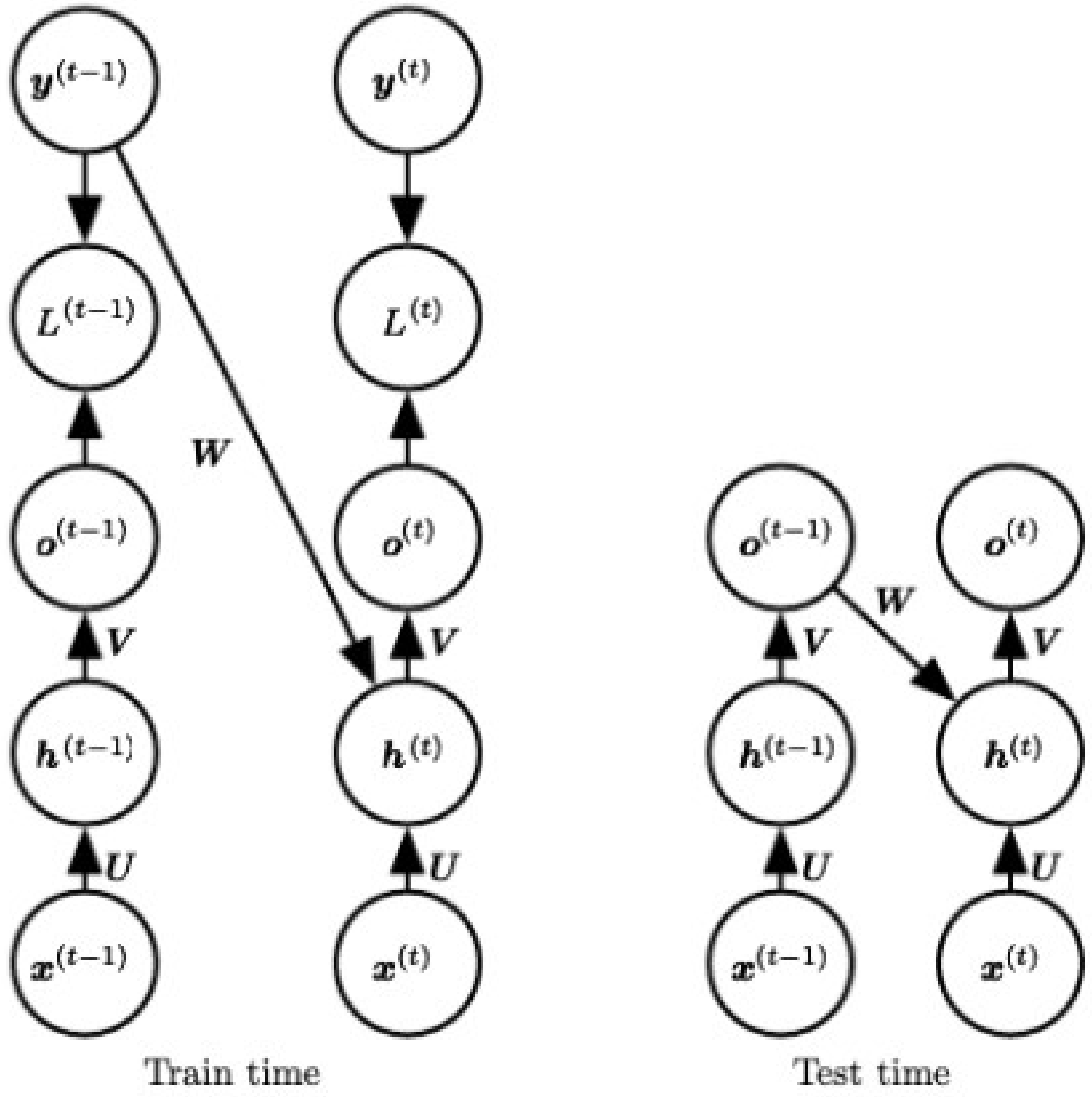
two RNNs training the network in opposite directions

- One RNN starts from the beginning of the sequence and works its way to the end
- The other RNN starts at the end and works its way to the beginning

The two RNNs are represented by the g and h hidden cell states

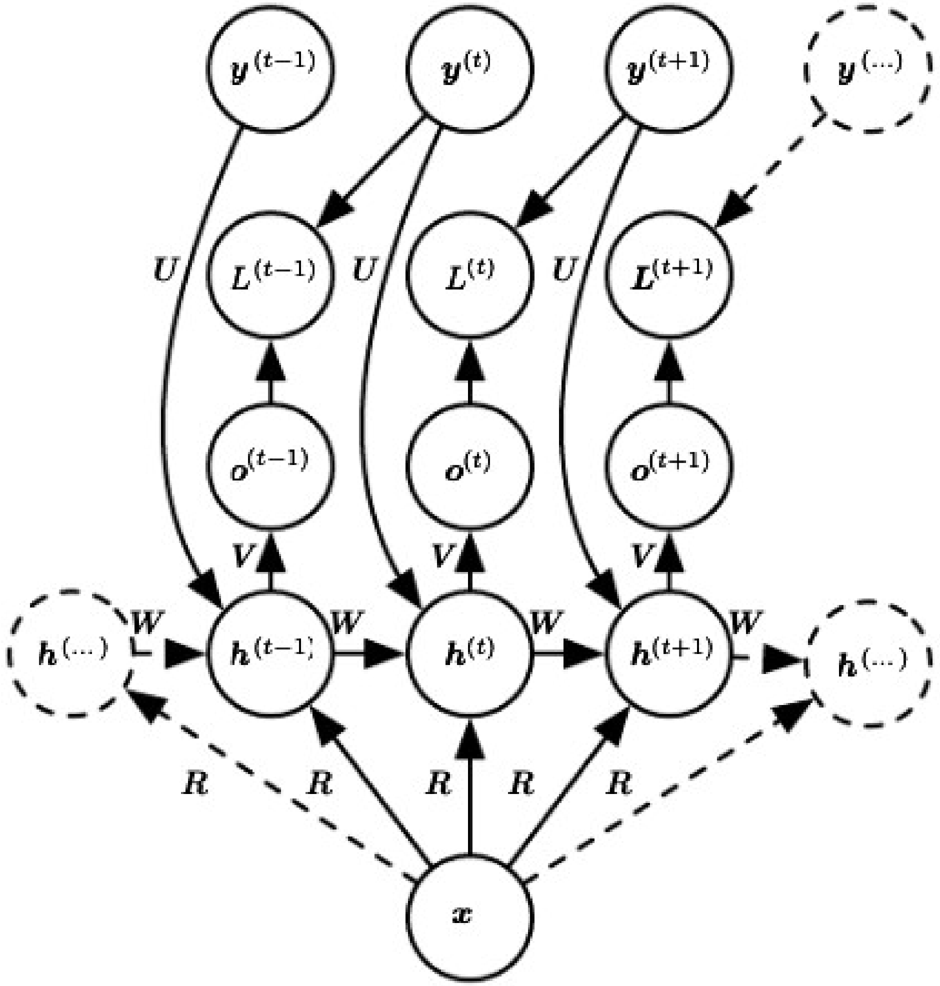


Teacher forcing and networks with output recurrence



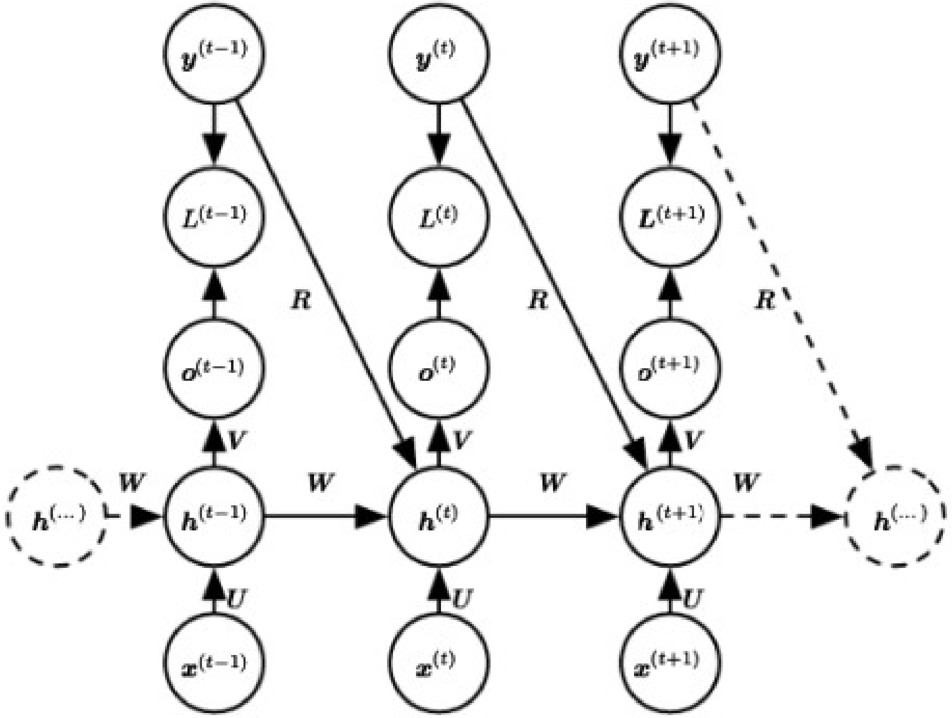
Architectures used for modeling sequences conditioned on context

an RNN that maps a fixed-length vector x into a distribution over sequences Y



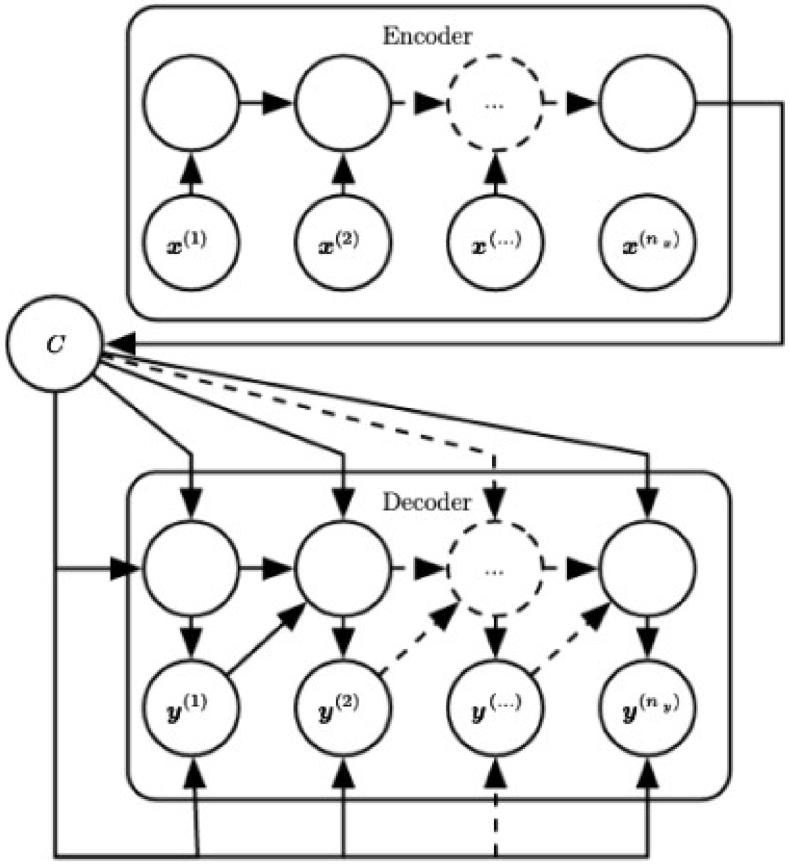
Conditional recurrent neural network architectures

map a variable-length sequence of x values into a distribution over sequences of y values of the same length



Encoder-decoder sequence-to-sequence architecture

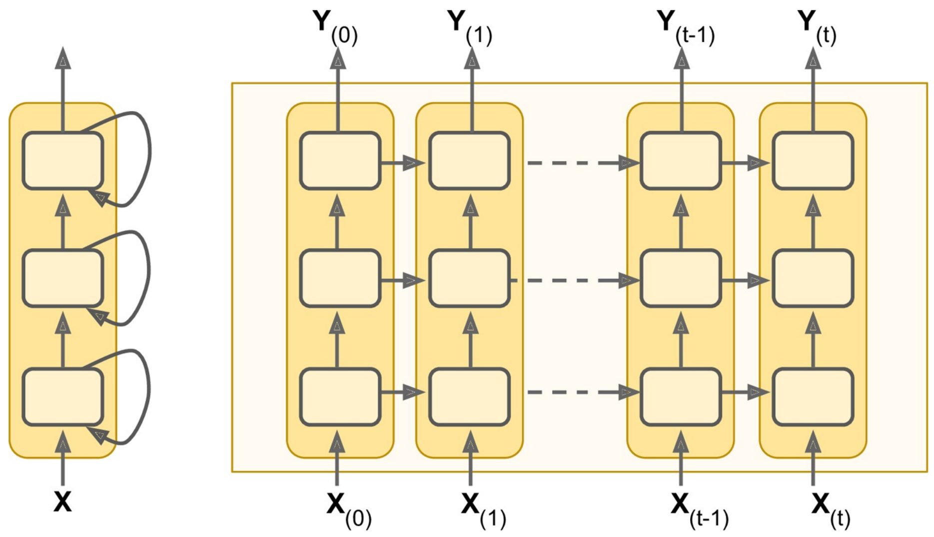
- Multiple RNN cells stacked together to form the encoder. RNN reads each inputs sequentially
- The Encoder will convert the input sequence into a single-dimensional vector (hidden vector)
- The decoder will convert the hidden vector into the output sequence



Overfitting and underfitting

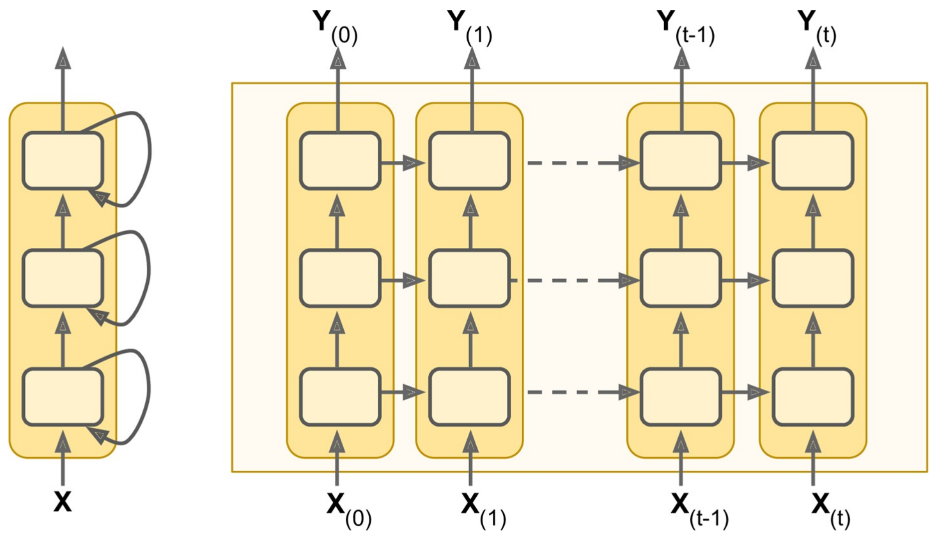
Applying Dropout

- There are two arguments in Keras RNN layers that you can use
  - **dropout**: Float between 0 and 1
  - Fraction of the units to drop for the linear transformation of the inputs
  - **recurrent_dropout**: Float between 0 and 1
  - Fraction of the units to drop for the linear transformation of the recurrent state



Layer Normalization

- Preferred approach
- instead of normalizing across the batch, you normalize across the feature dimension



Important considerations

Handling Long Sequences

- need to run it over many time steps, making the unrolled RNN a very deep network
- Best practices:
  - proper weight initialization
  - non-saturating activation functions
  - gradient clipping
  - faster optimizers

- Training take a long time for RNNs
  - truncated backpropagation through time
    - unroll the RNN only over a limited number of time steps during training
  - truncating the input sequences
    - Reduce n_steps during training.
    - model will not be able to learn long-term patterns
    - One workaround could be to make sure that these shortened sequences contain both old and recent data

Big issue:

For RNNs, the memory of the first inputs gradually fades away

- Some information is lost after each time step.
- After a while, the RNN’s state contains virtually no trace of the first inputs
  - Example: sentiment analysis on a long review:
  - Review starts with the four words “I loved this movie,”
  - The rest of the review lists the many things that could have made the movie even better
  - To solve this problem, various types of cells with long-term memory have been introduced

Advantages and Shortcomings of RNNs

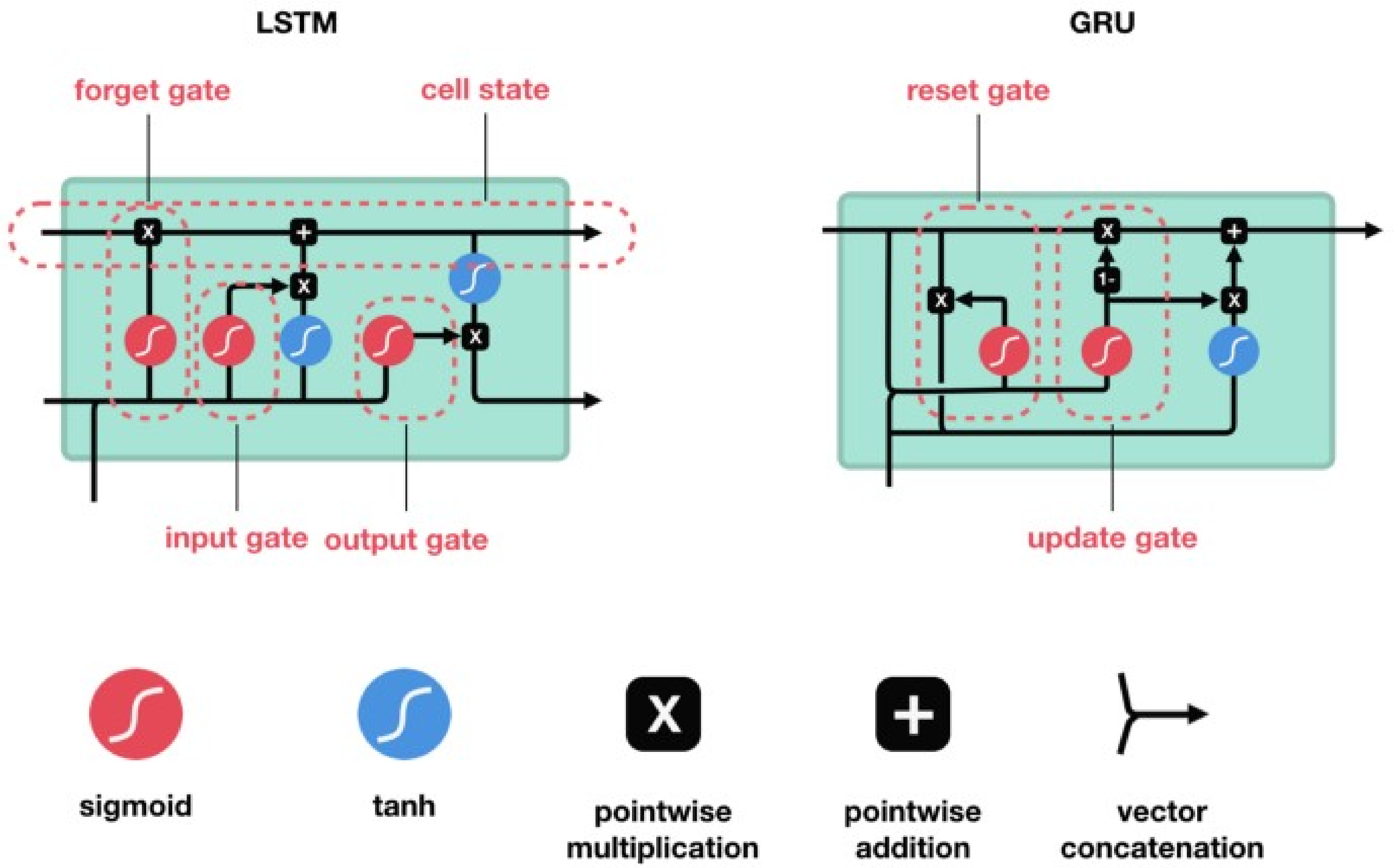
Advantages:

- Ability to handle sequence data
- Ability to handle inputs of varying lengths
- Ability to store or “memorize” historical information

Disadvantages:

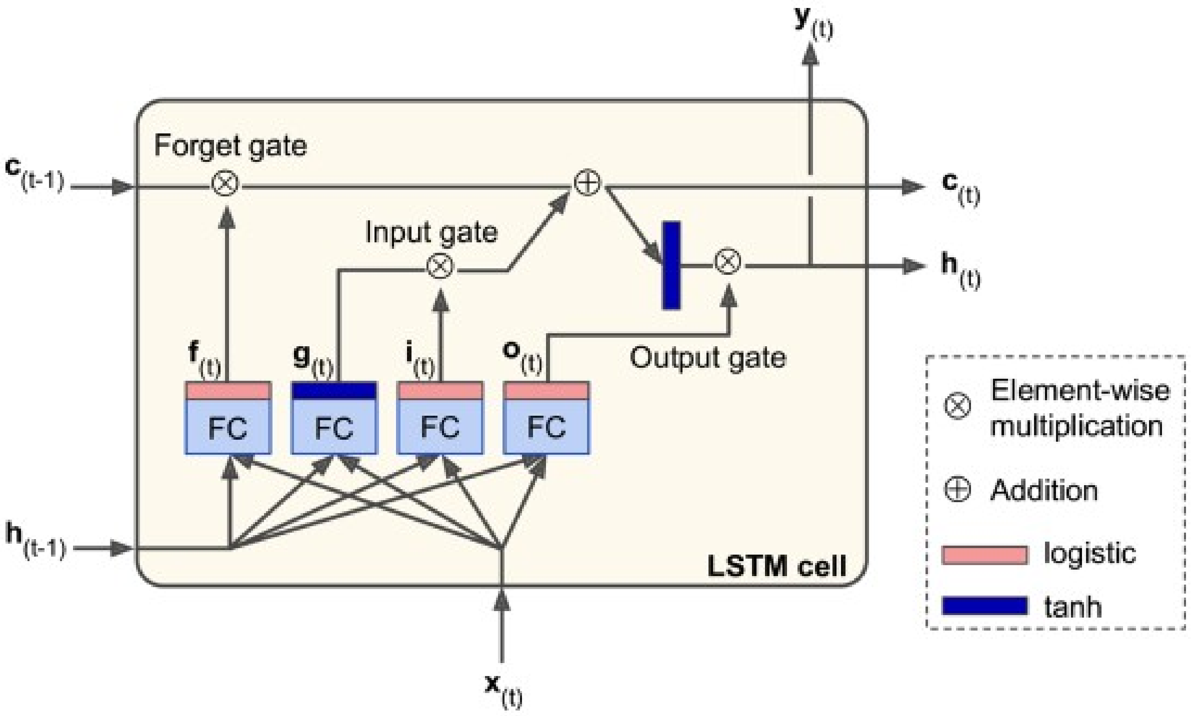
- The computation can be very slow
- The network does not take into account future inputs to make decisions
- Vanishing gradient problem
  - The deeper the network, the more pronounced this problem is

Long Short-Term Memory (LSTM)

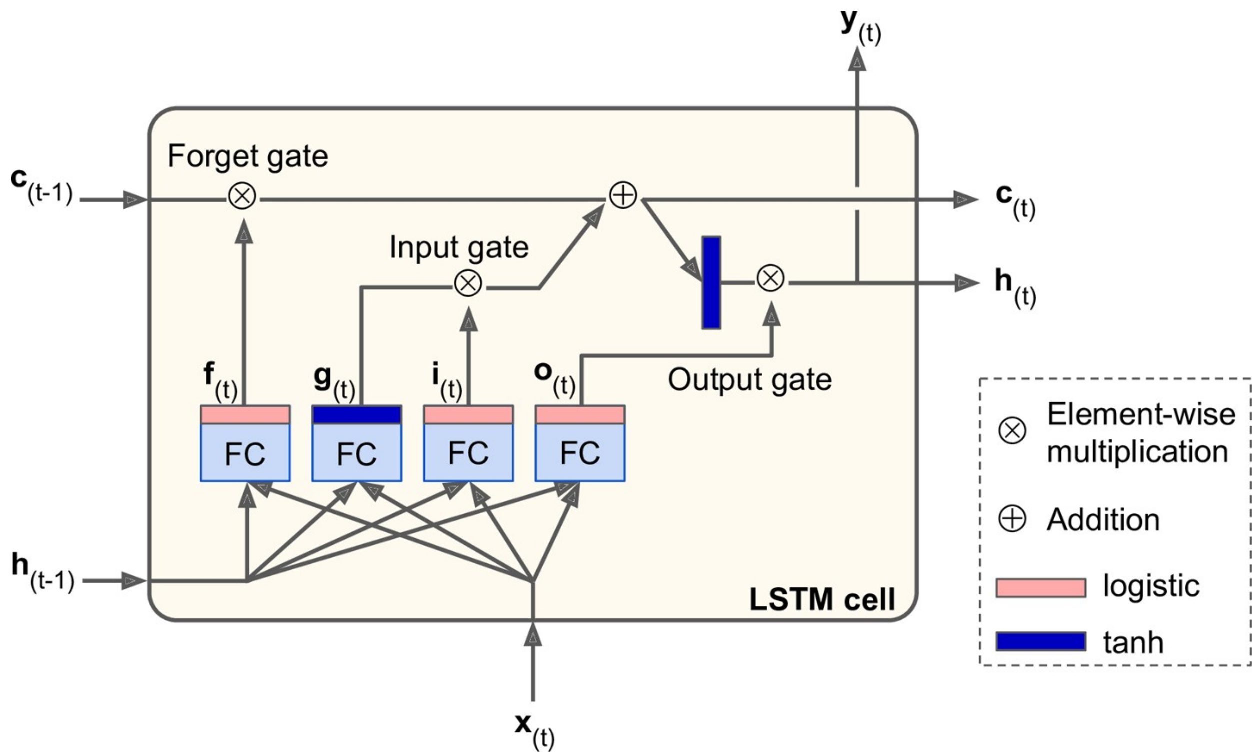


LSTM
GRU
forget gate
cell state
input gate
output gate
reset gate
update gate
1-
sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

- The LSTM cell looks exactly like a regular cell, except that its state is split in two vectors:
  - $h(t)$ the short-term state and
  - $c(t)$ as the long-term state

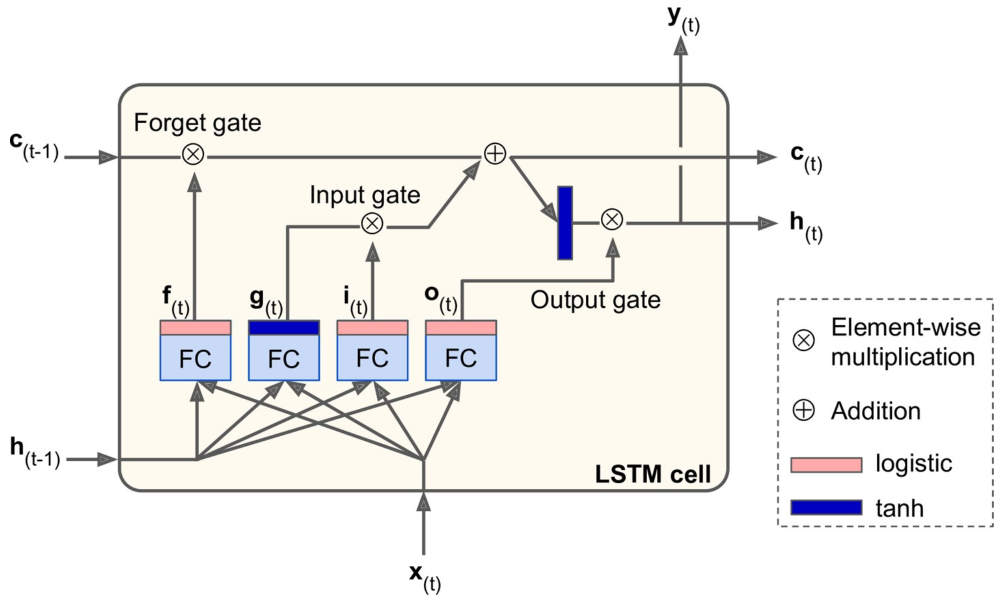


- As the long-term state $c(t-1)$ traverses the network:
  - it first goes through a forget gate, dropping some memories, then
  - adds some new memories via the addition operation
  - At each time step, some memories are dropped and some memories are added.
- The long-term state is then copied and passed through the tanh function, the result is filtered by the output gate. This produces the short-term state $h(t)$h(t).



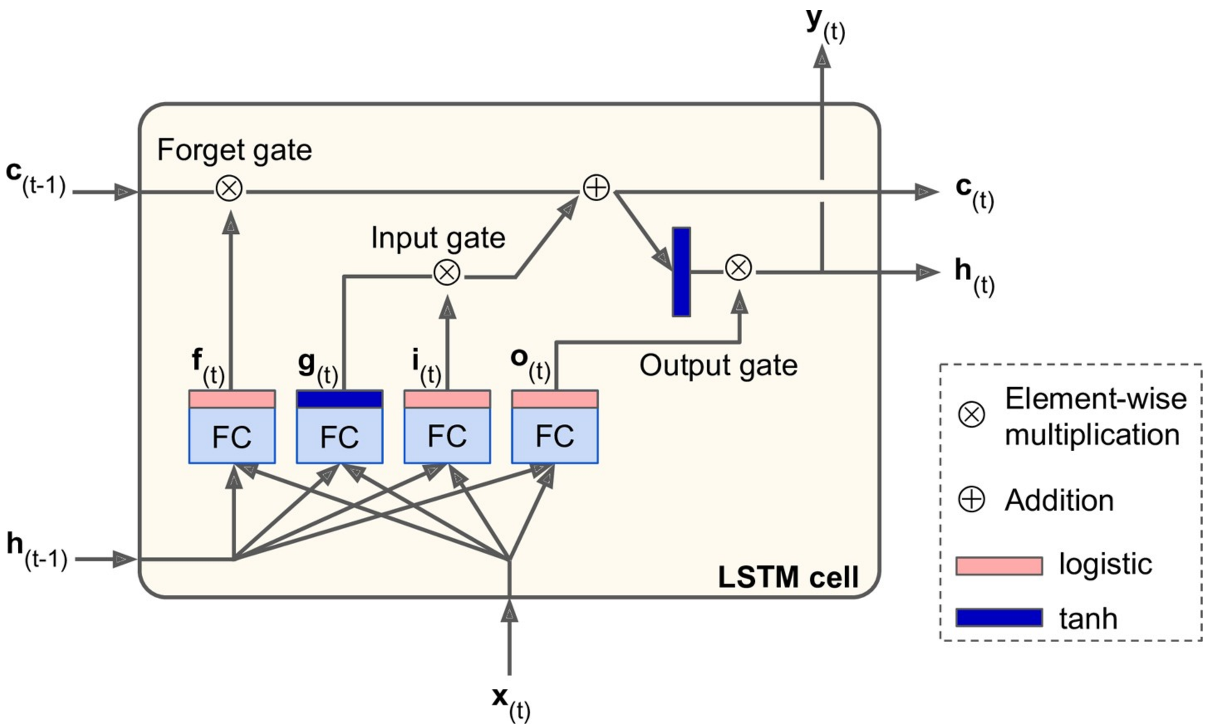
The current input vector $x(t)$ and the previous short-term state $h(t-1)$ are fed to four different fully connected layers

- The main layer is the one that outputs $g(t)$
- It has the usual role of analyzing the current inputs $x(t)$ and the previous $h(t-1)$
  - this layer's output is partially stored in the long-term state
- The three other layers are gate controllers.
- Their outputs range from 0 to 1
  - Their outputs are fed to element-wise multiplication operations
  - if they output 0s, they close the gate, and if they output 1s, they open it



- The forget gate controls which parts of the long-term state should be erased
- The input gate controls which parts of $g(t)$ should be added to the long-term state
- The output gate controls which parts of the long-term state should be read and output at this time step.

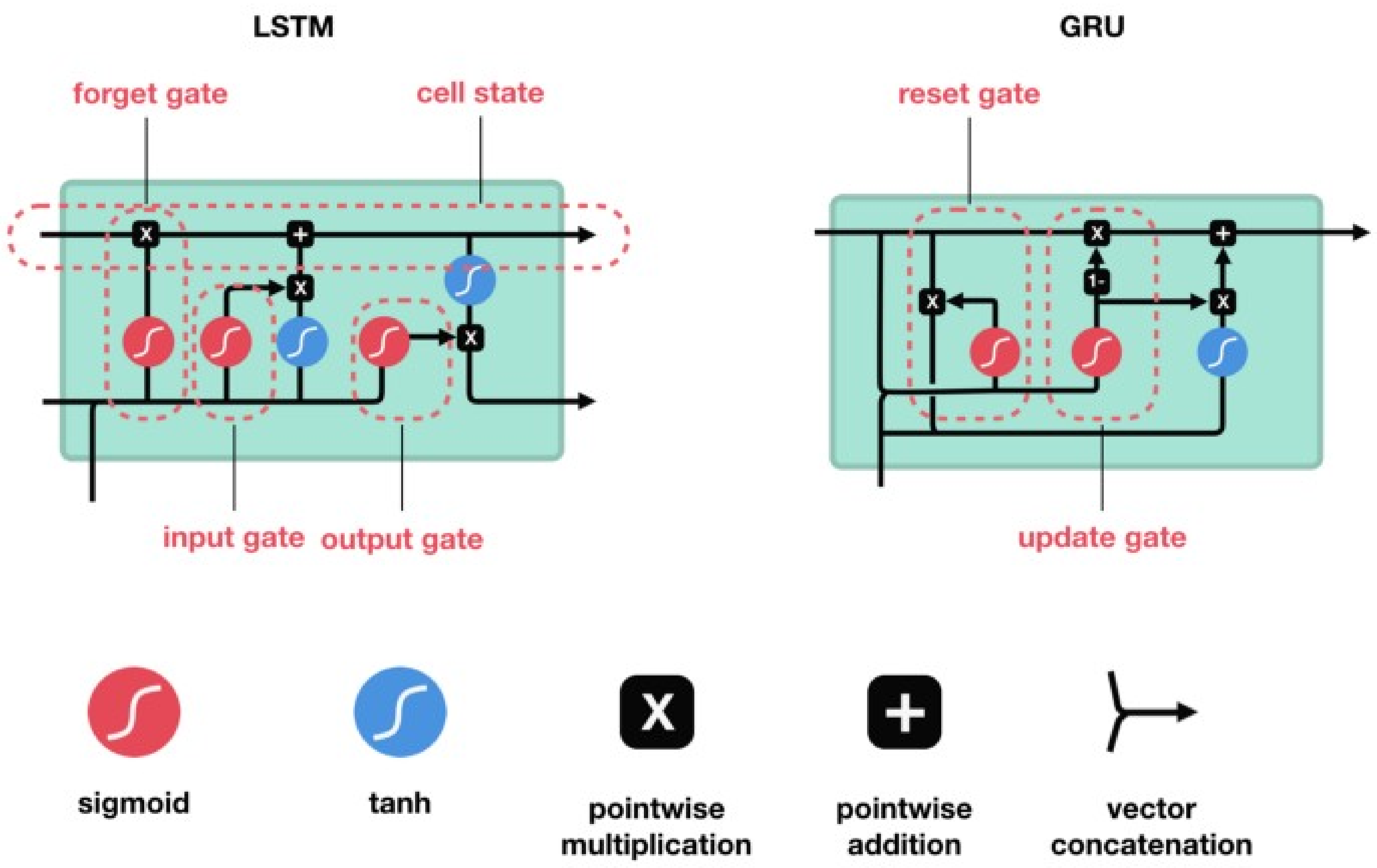
• An LSTM cell can learn to recognize an important input, store it in the long-term state, learn to preserve it for as long as it is needed, and learn to extract it whenever it is needed.



Peephole LSTM Connections

- The previous long-term state $c(t{-}1)$ is added as an input to the controllers of the forget gate and the input gate, and the current long-term state $c(t)$ is added as input to the controller of the output gate

Gated Recurrent Unit (GRU)


LSTM
forget gate
cell state
input gate
output gate
GRU
reset gate
1-
update gate
sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

- designed to handle the vanishing gradient problem
- have a reset and update gate that determine which information is to be retained for future predictions

  - Both state vectors are merged into a single vector $h(t)$
  - A single gate controller controls both the forget gate and the input gate:
    - If the gate controller outputs a 1, the forget gate is open and the input gate is closed
    - If it outputs a 0, the opposite happens
    - Whenever a memory must be stored, the location is erased first
  - There is no output gate
  - The full state vector is output at every time step
  - There is a new gate controller that controls which part of the previous state will be shown to the main layer