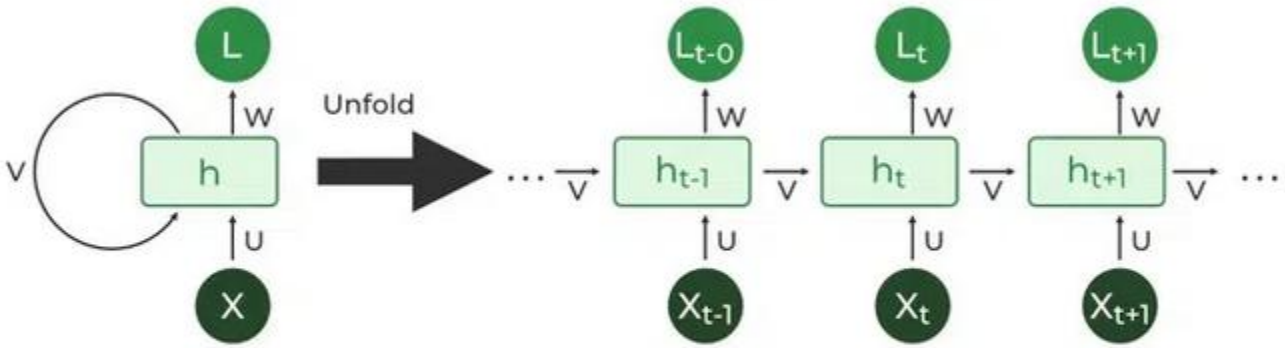
RNN

Recurrent Neural Networks (RNNs) work a bit different from regular neural networks.

In neural network the information flows in one direction from input to output.

However in RNN information is fed back into the system after each step.

- RNNs allow the network to “remember” past information by feeding the output from one step into next step.
- This helps the network understand the context of what has already happened and make better predictions based on that.



Recurrent Neural Network

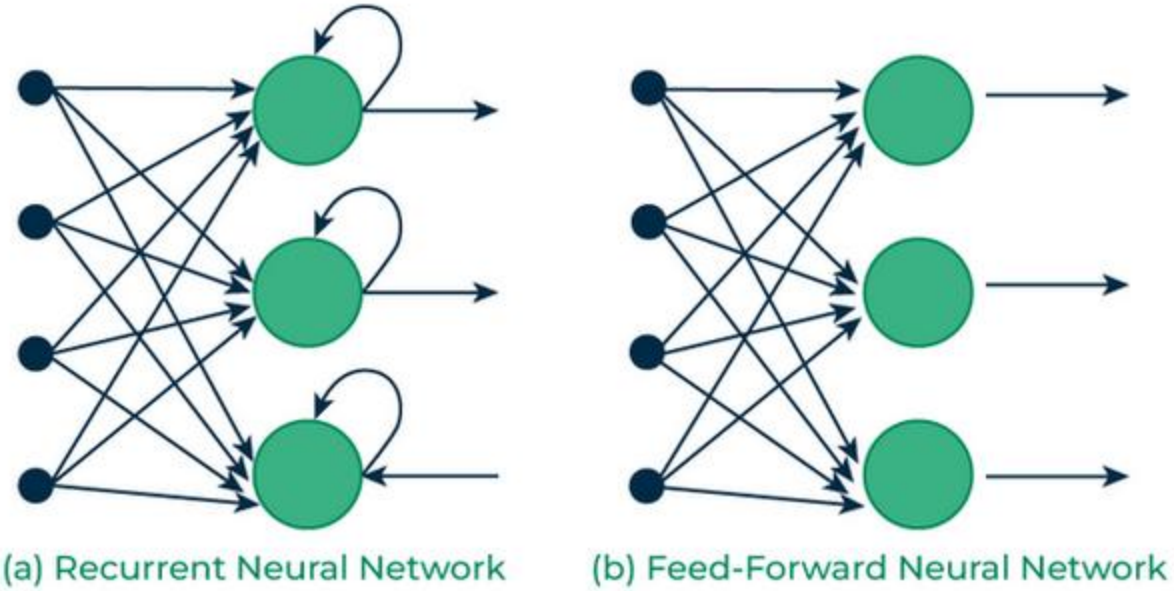
How RNN Differs from Feedforward Neural Networks?

How RNN Differs from Feedforward Neural Networks?

- Feedforward Neural Networks (FNNs) process data in one direction from input to output without retaining information from previous inputs.

- This makes them suitable for tasks with independent inputs like image classification. However FNNs struggle with sequential data since they lack memory.

- Recurrent Neural Networks (RNNs) solve this **by incorporating loops that allow information from previous steps to be fed back into the network**. This feedback enables RNNs to remember prior inputs making them ideal for tasks where context is important.



Recurrent Vs Feedforward networks

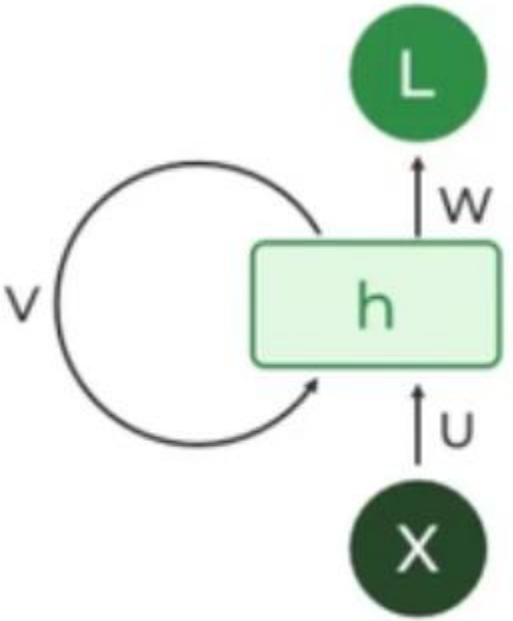
Key Components of RNNs

Key Components of RNNs

- **1. Recurrent Neurons**
- **2. RNN Unfolding**

Recurrent Neurons

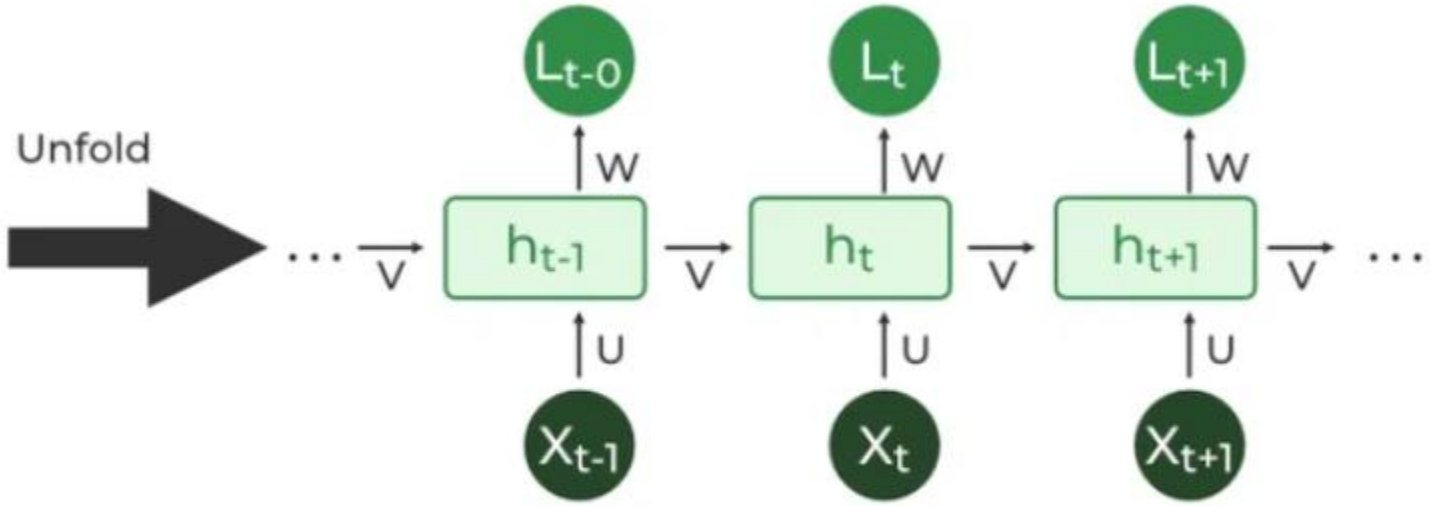
- The fundamental processing unit in RNN is a **Recurrent Unit.** Recurrent units hold a hidden state that maintains information about previous inputs in a sequence.
- Recurrent units can “remember” information from prior steps by feeding back their hidden state, allowing them to capture dependencies across time.



Recurrent Neuron

RNN Unfolding

- RNN unfolding or unrolling is the process of expanding the recurrent structure over time steps. During unfolding each step of the sequence is represented as a separate layer in a series illustrating how information flows across each time step.



RNN Unfolding

How does RNN work?

- At each time step RNNs process units with a fixed activation function. These units have an internal hidden state that acts as memory that retains information from previous time steps. This memory allows the network to store past knowledge and adapt based on new inputs.

- **Updating the Hidden State in RNNs**
- The current hidden state $h_t$depends on the previous state h_{t-1} and the current input x_t, and is calculated using the following relations:

1. State Update:

$$h_t = f(h_{t-1}, x_t)$$

where:

- h_t is the current state
- h_{t-1} is the previous state
- x_t is the input at the current time step

Activation Function Application

$$h_t = f(w_{hh} \cdot h_{t-1} + w_{xh} . x_t)$$

Here, W_{hh} is the weight matrix for the recurrent neuron, and W_{xh} is the weight matrix for the input neuro

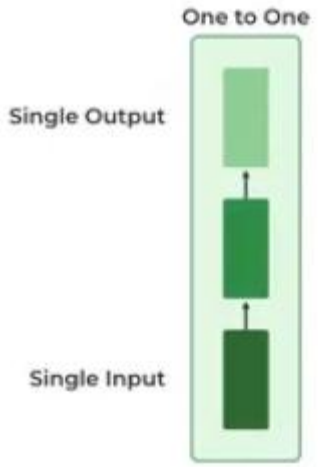
Output Calculation:

$$y_t = W_{hy} \cdot h_t$$

where y_t is the output and W_{hy} is the weight at the output layer.

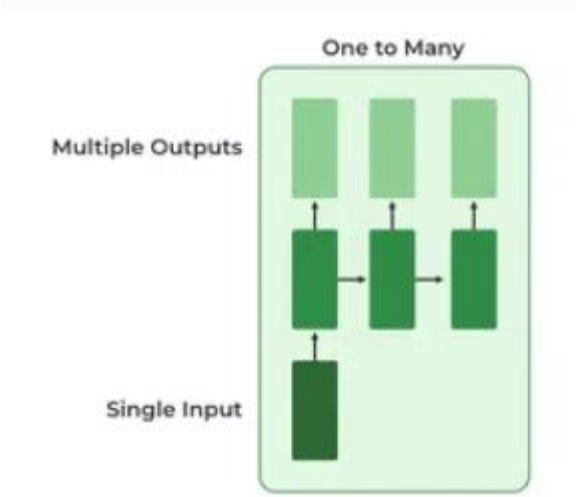
Types Of Recurrent Neural Networks

- **One-to-One RNN**



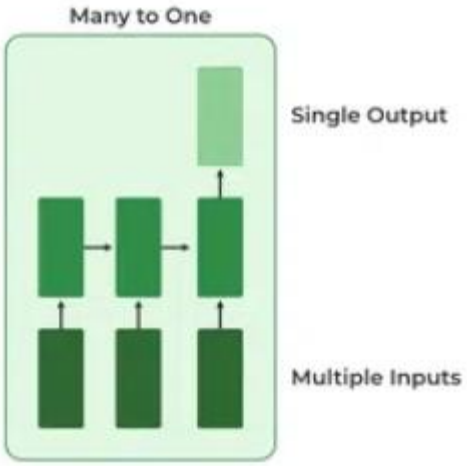
One to One RNN

- **One-to-Many RNN**



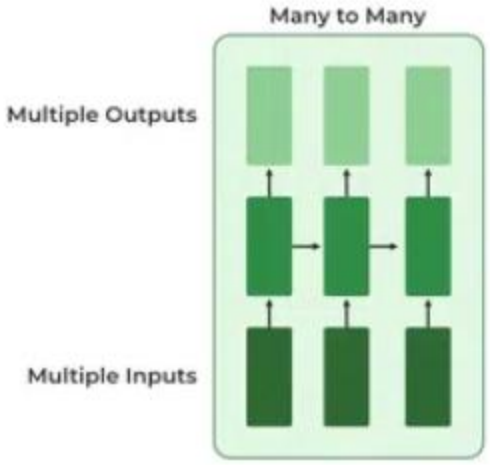
One to Many RNN

- **Many-to-One RNN**



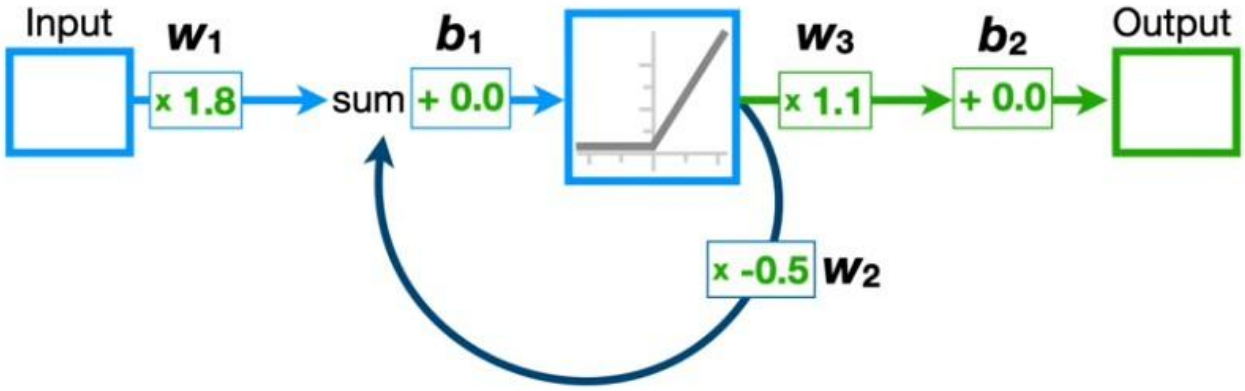
Many to One RNN

- **Many-to-Many RNN**

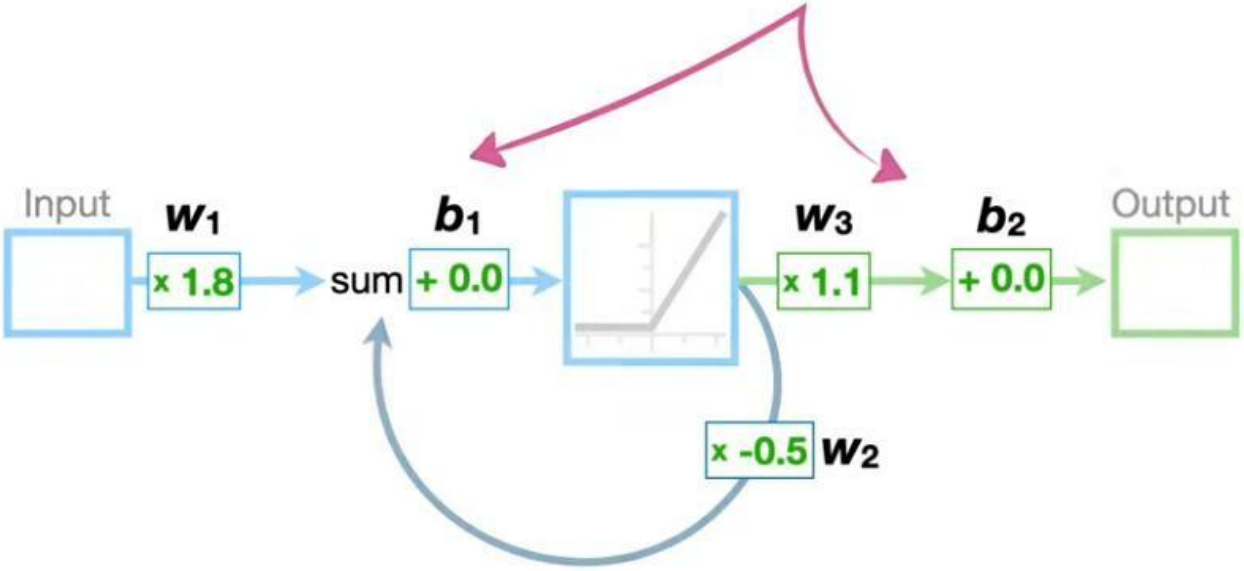


Many to Many RNN

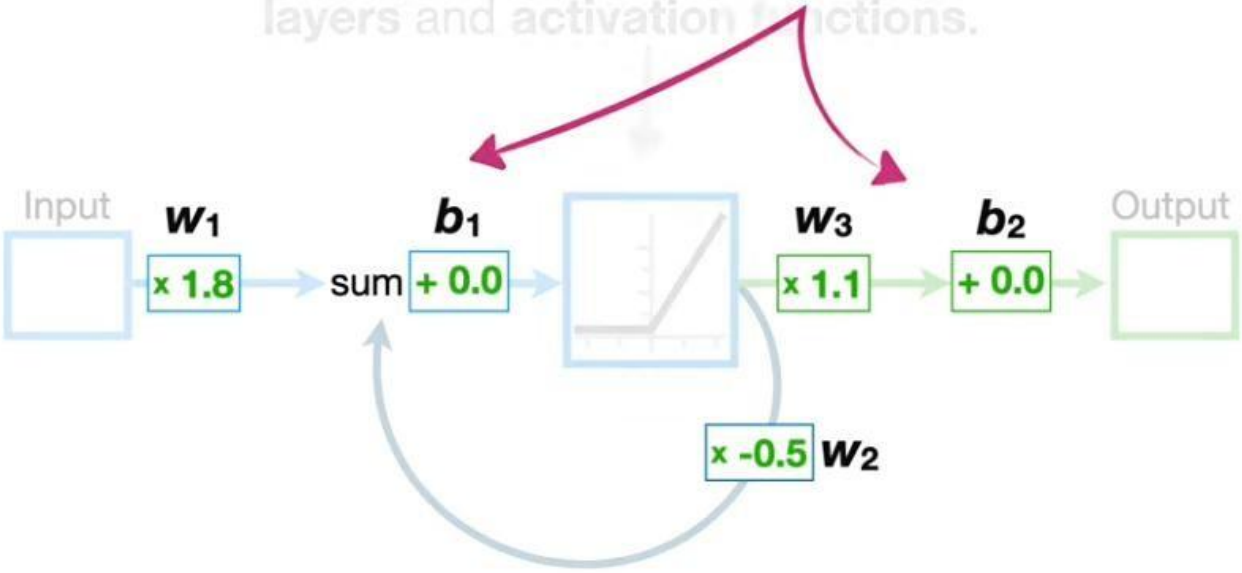
Just like the other neural networks that we've seen before, **Recurrent Neural Networks** have **weights**, **biases**,



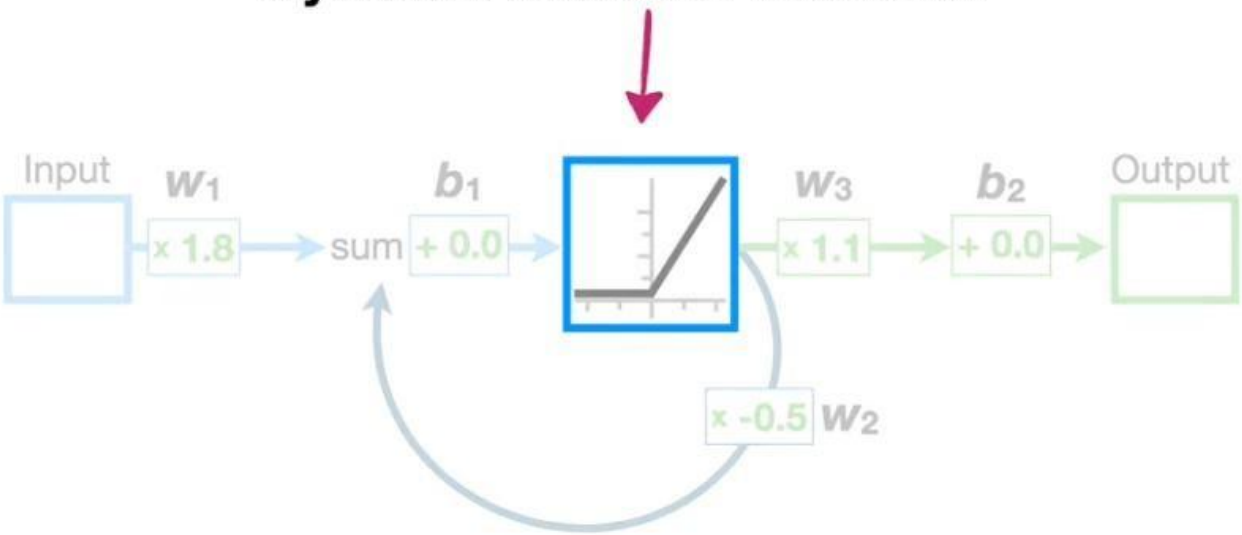
Just like the other neural networks that we've seen before, **Recurrent Neural Networks** have **weights, biases,**



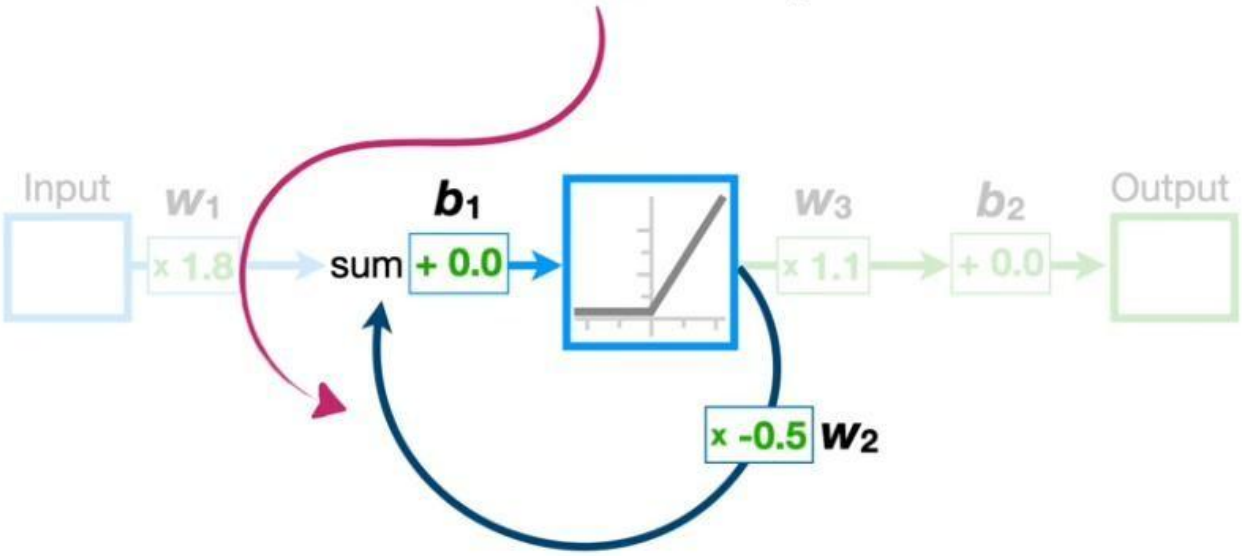
Just like the other neural networks that we've seen before, **Recurrent Neural Networks** have **weights**, **biases**, layers and activation functions.

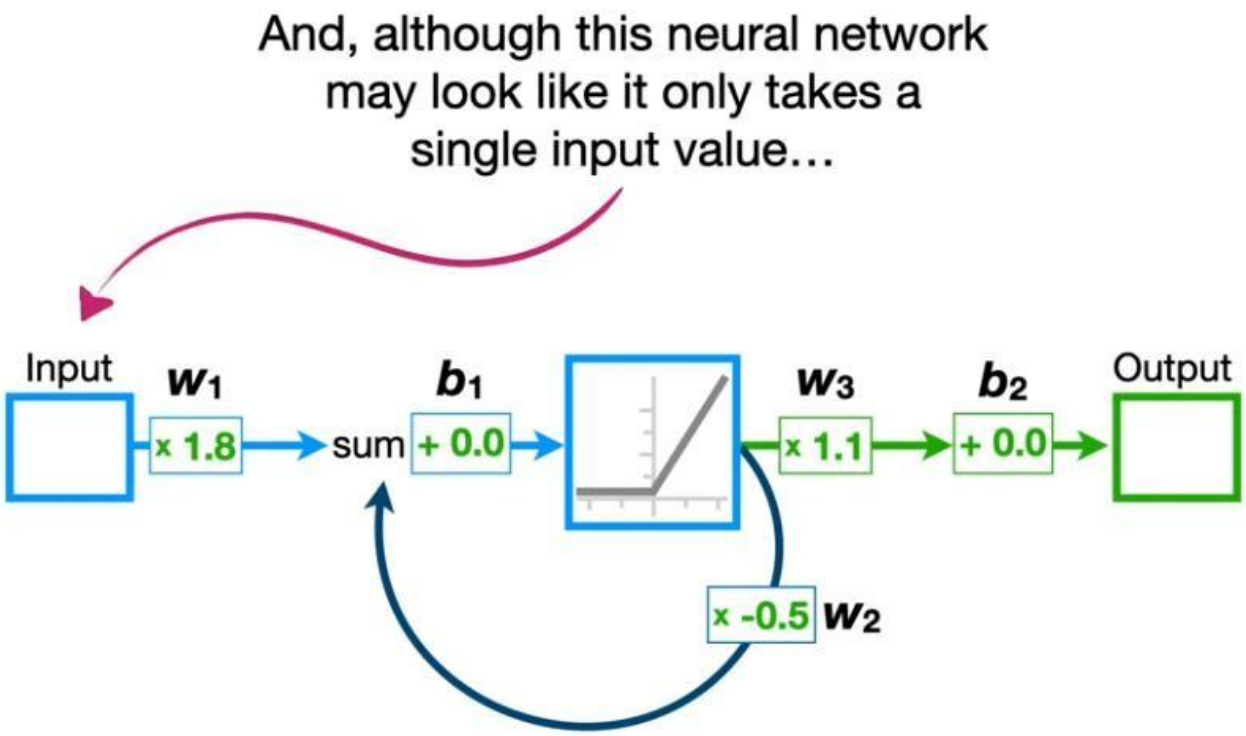


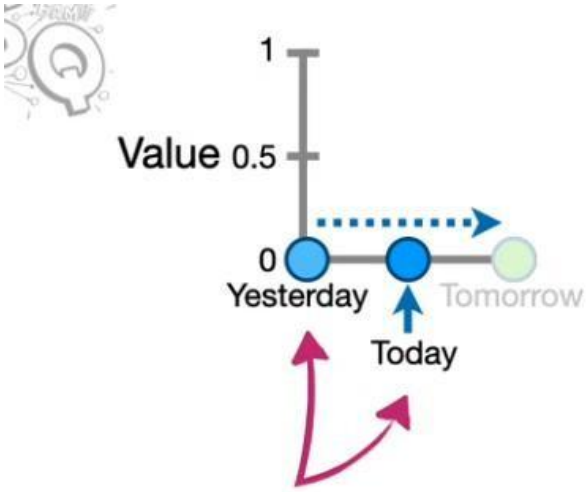
Just like the other neural networks that we've seen before, **Recurrent Neural Networks** have **weights**, **biases**, **layers** and **activation functions**.



The big difference is that
Recurrent Neural Networks also
have **feedback loops**.

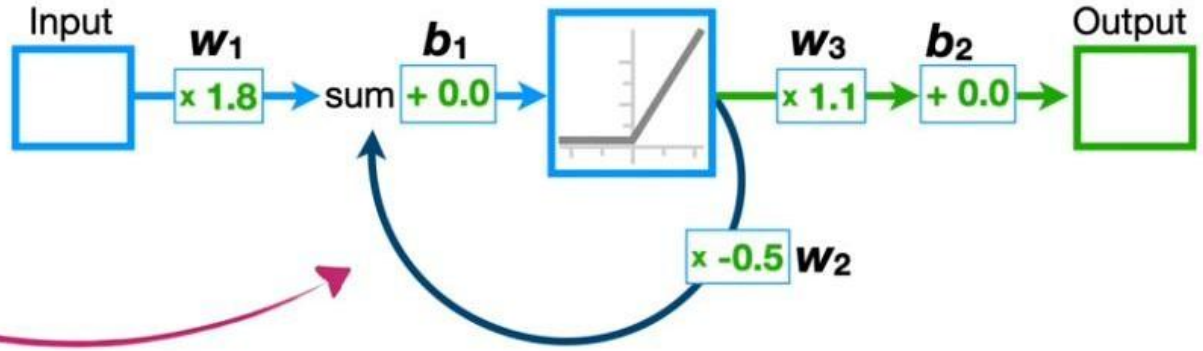



And, although this neural network
may look like it only takes a
single input value…
Input
w1
x 1.8
sum
b1
+ 0.0
w3
x 1.1
b2
+ 0.0
Output
x -0.5
w2

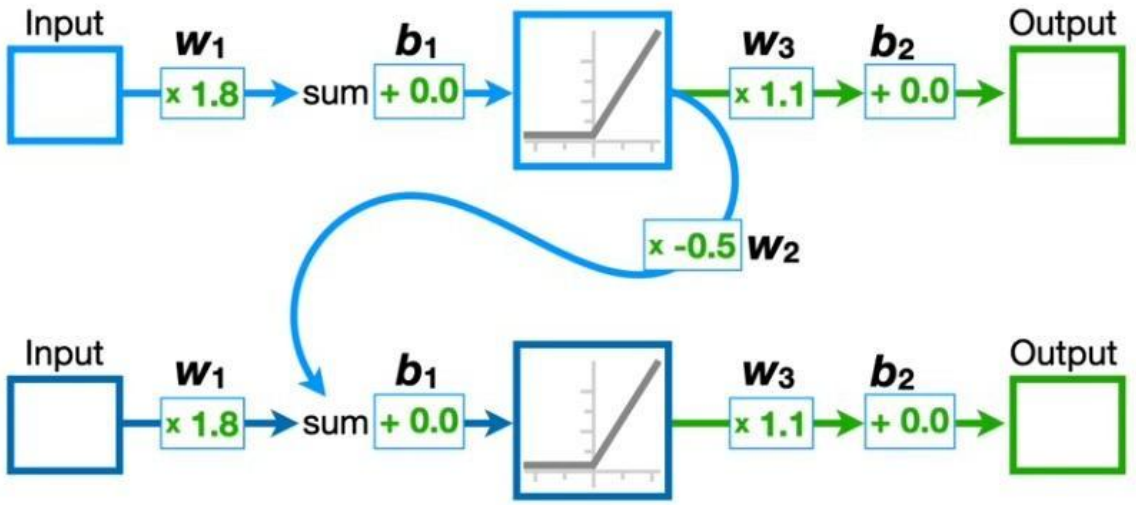


Now let's run the values for **yesterday** and **today**...

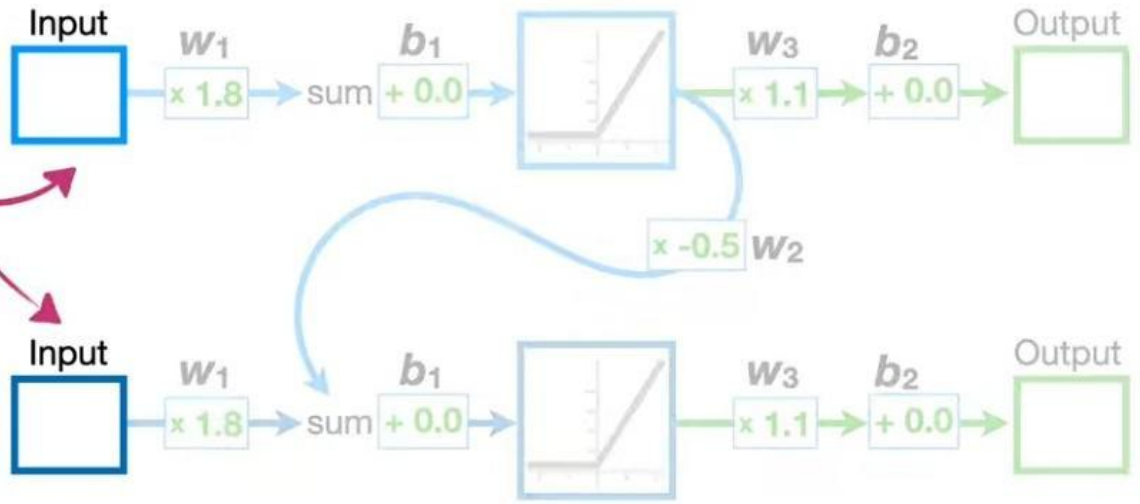
Now, because the recurrent neural network has a **feedback loop**…

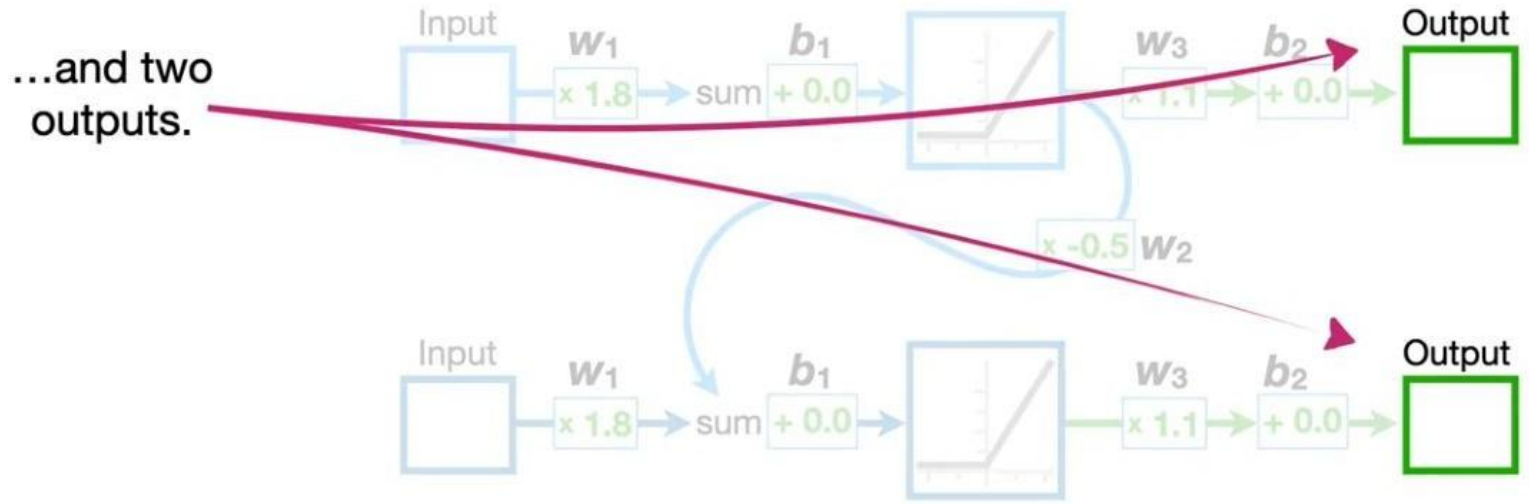


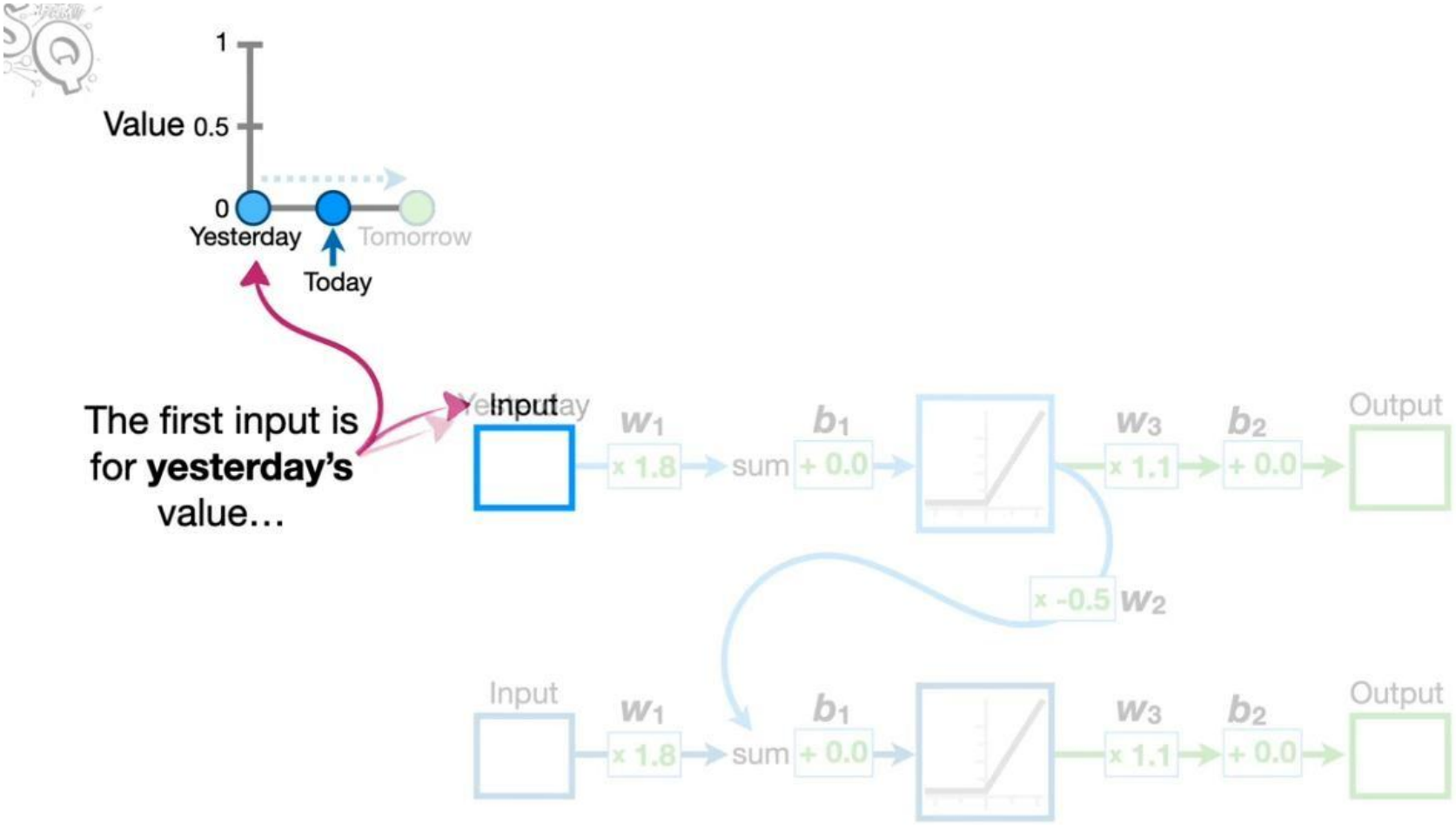
By **unrolling** the recurrent neural network, we end up with an new network that has two inputs…

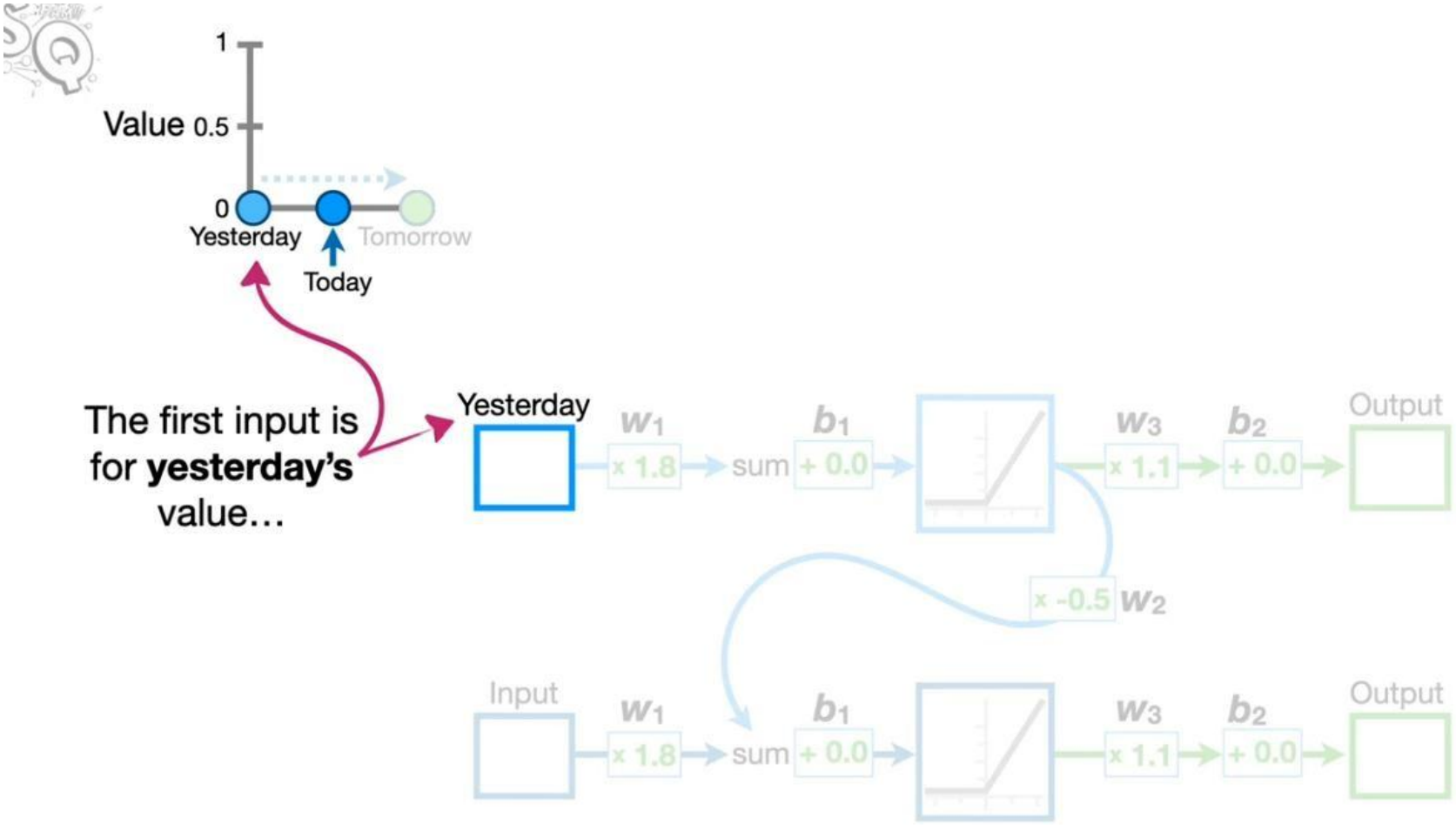


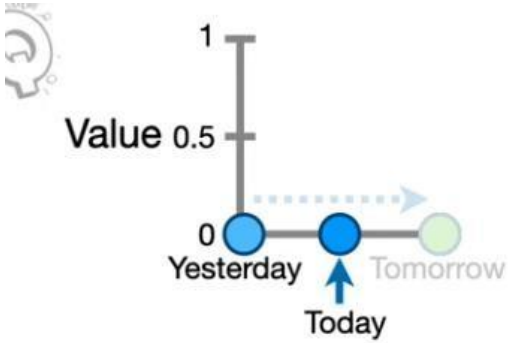
By **unrolling** the recurrent neural network, we end up with an new network that has two inputs…



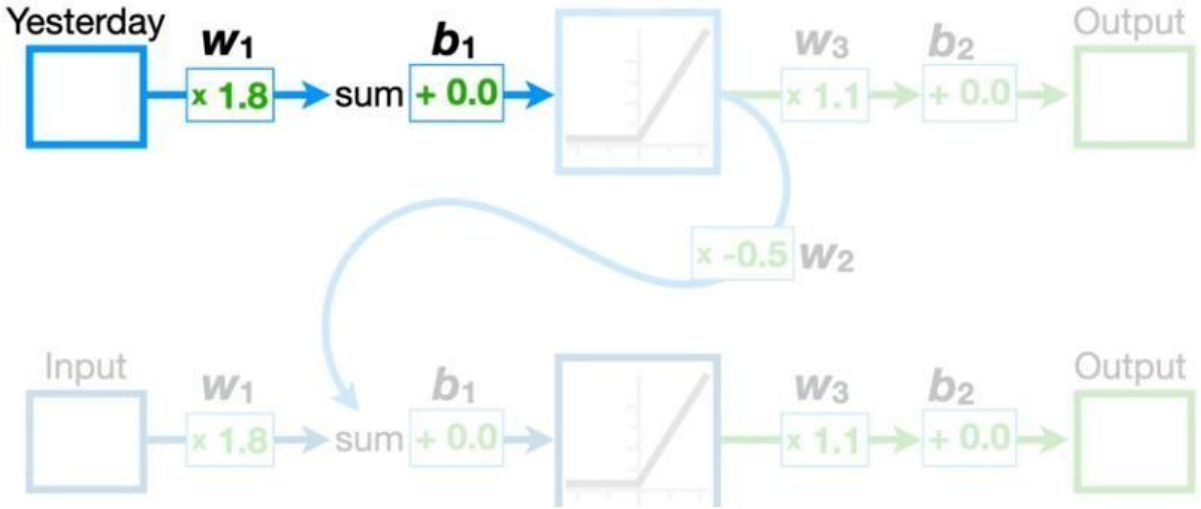

…and two
outputs.
Input
w_1
× 1.8
sum
b_1
+ 0.0
w_3
× 1.1
b_2
+ 0.0
Output
× -0.5
w_2
Input
w_1
× 1.8
sum
b_1
+ 0.0
w_3
× 1.1
b_2
+ 0.0
Output

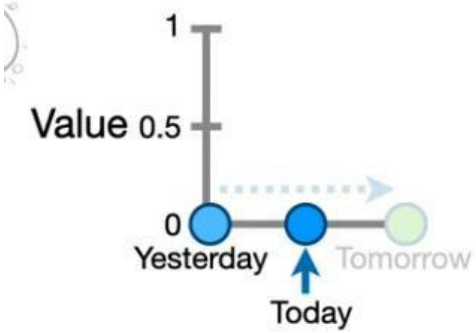

1
Value 0.5
0
Yesterday
Today
Tomorrow
The first input is for **yesterday's** value…
Input
w_1
x 1.8
sum
b_1
+ 0.0
w_3
x 1.1
b_2
+ 0.0
Output
x -0.5 w_2
Input
w_1
x 1.8
sum
b_1
+ 0.0
w_3
x 1.1
b_2
+ 0.0
Output


1
Value 0.5
0
Yesterday
Tomorrow
Today
The first input is for **yesterday's** value…
Yesterday
w_1
x 1.8
sum
b_1
+ 0.0
w_3
x 1.1
b_2
+ 0.0
Output
x -0.5 w_2
Input
w_1
x 1.8
sum
b_1
+ 0.0
w_3
x 1.1
b_2
+ 0.0
Output

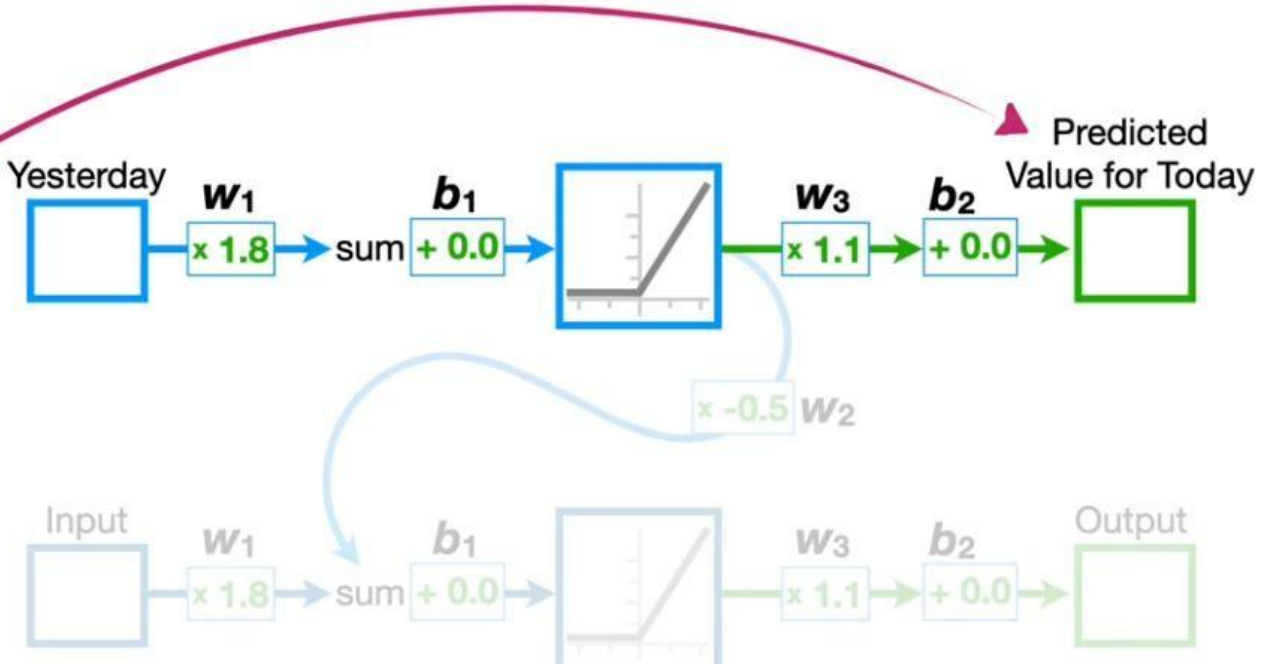


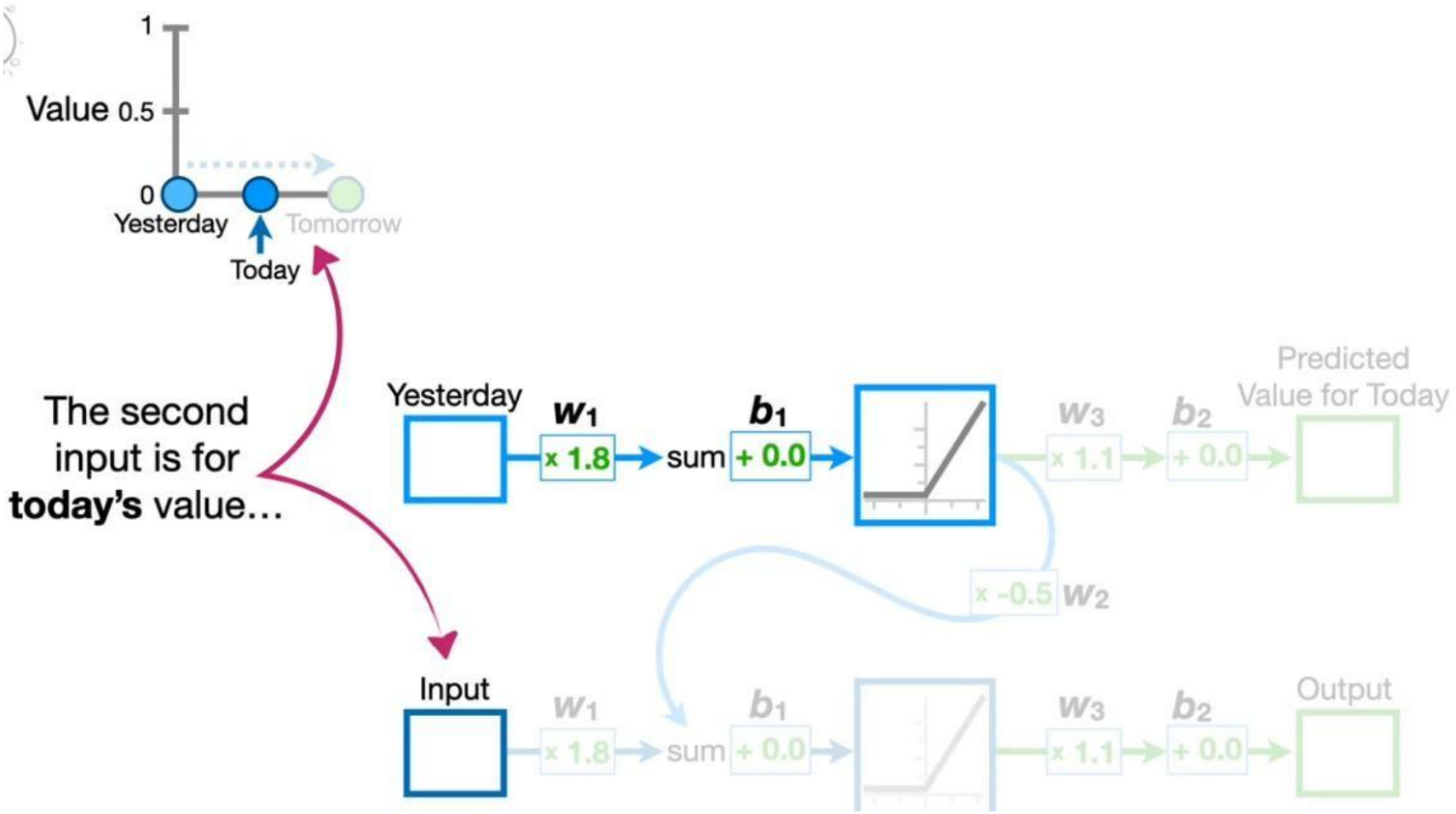
…and if we do the math straight through to the first output like we did earlier…

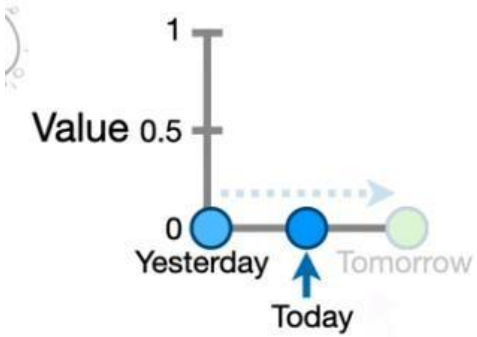




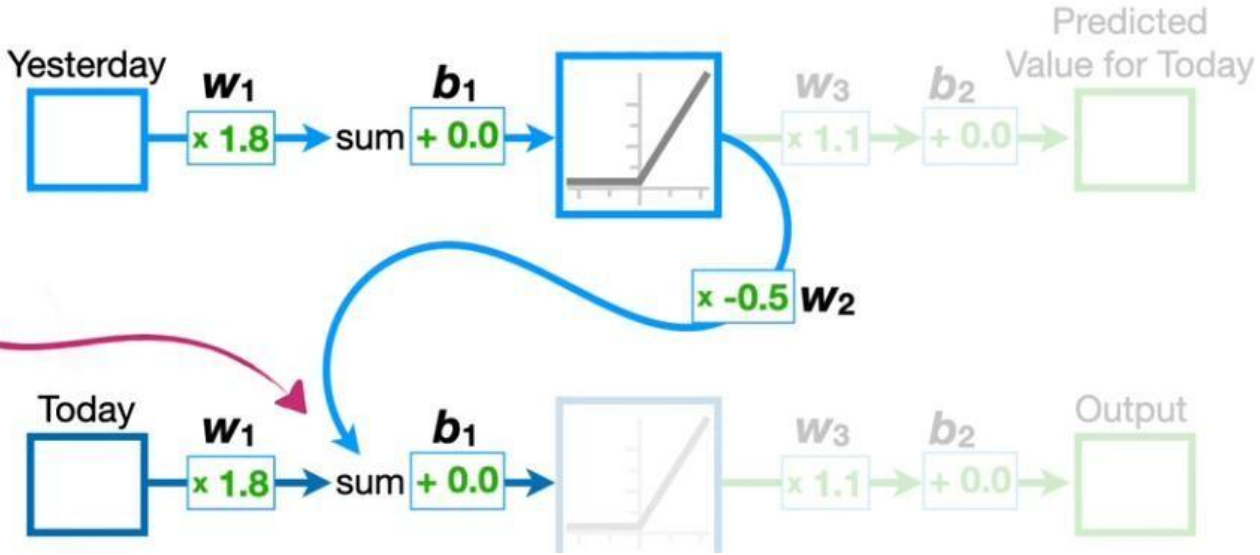
However, as we saw earlier, we can ignore this output.

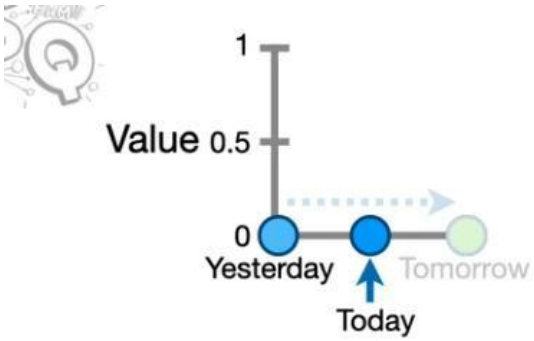



1
Value 0.5
0
Yesterday
Today
Tomorrow
The second input is for **today's** value…
Yesterday
w_1
x 1.8
sum
b_1
+ 0.0
w_3
x 1.1
b_2
+ 0.0
Predicted Value for Today
x -0.5
w_2
Input
w_1
x 1.8
sum
b_1
+ 0.0
w_3
x 1.1
b_2
+ 0.0
Output

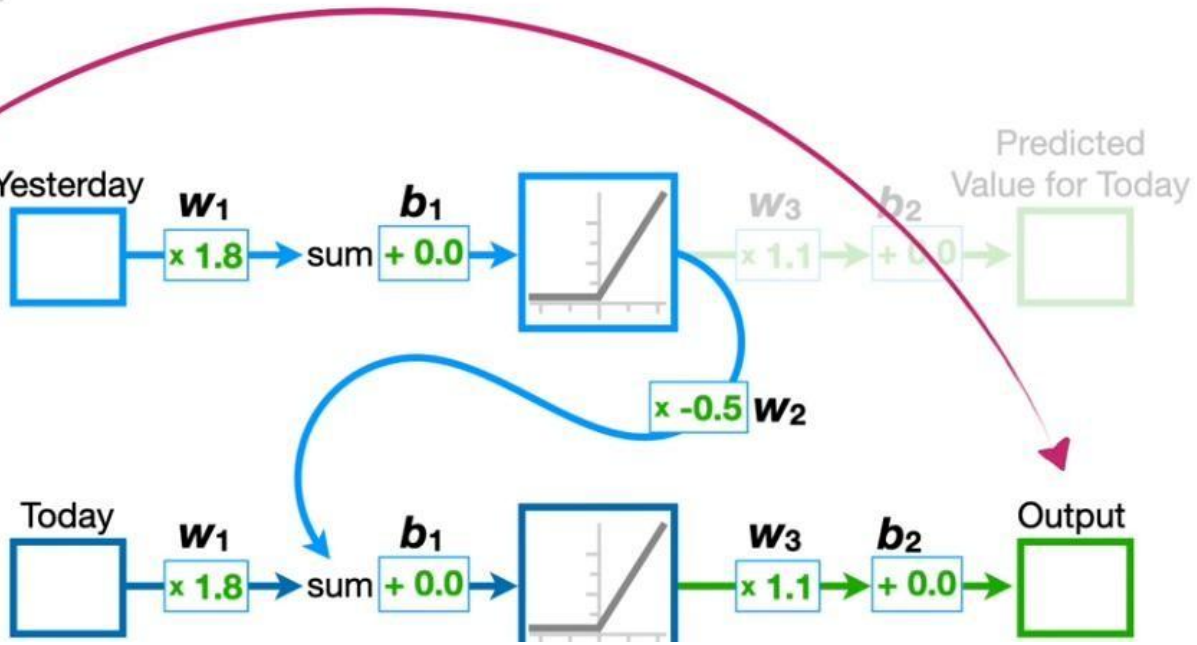


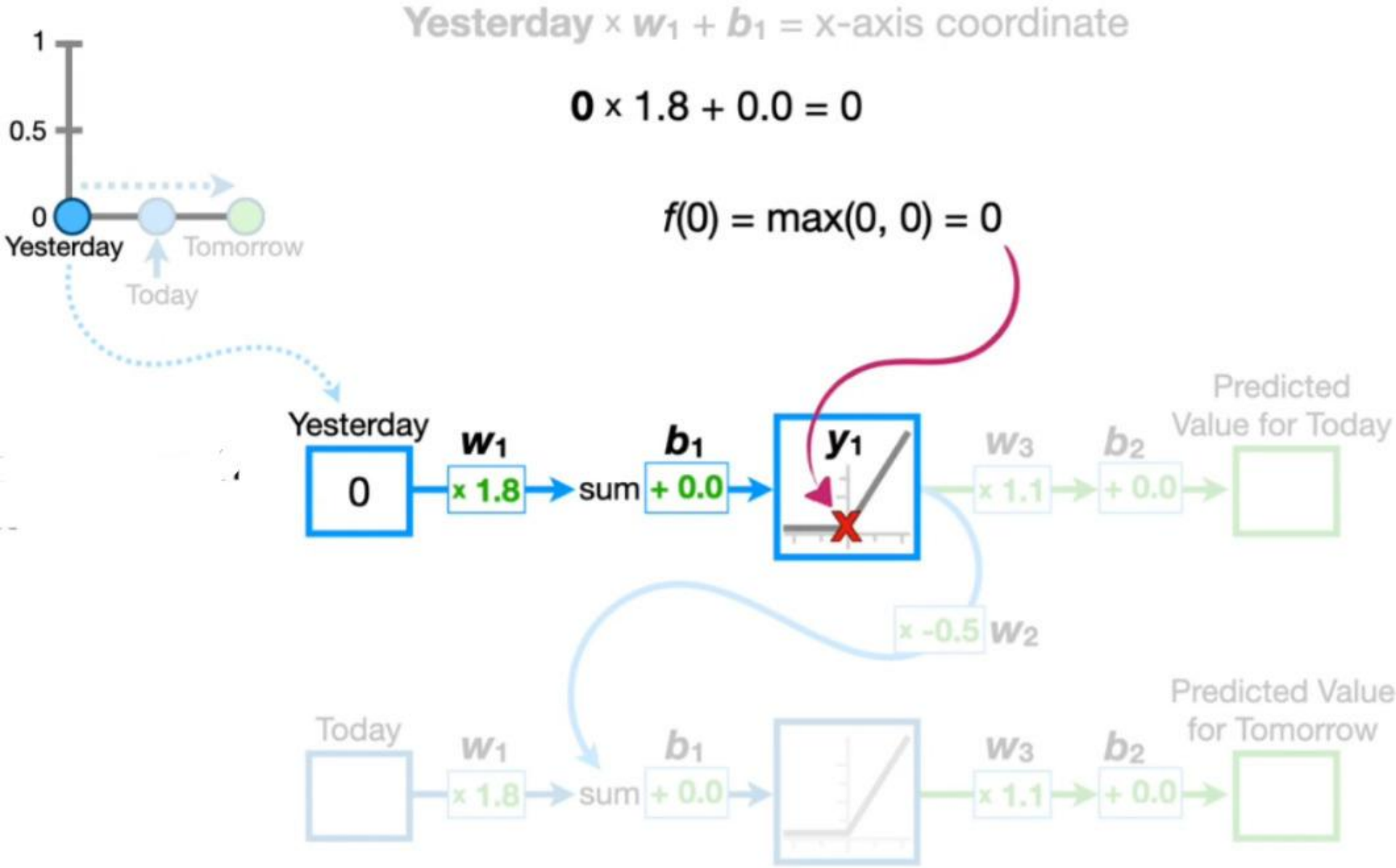
and the connection between the first ivation function and second summation…

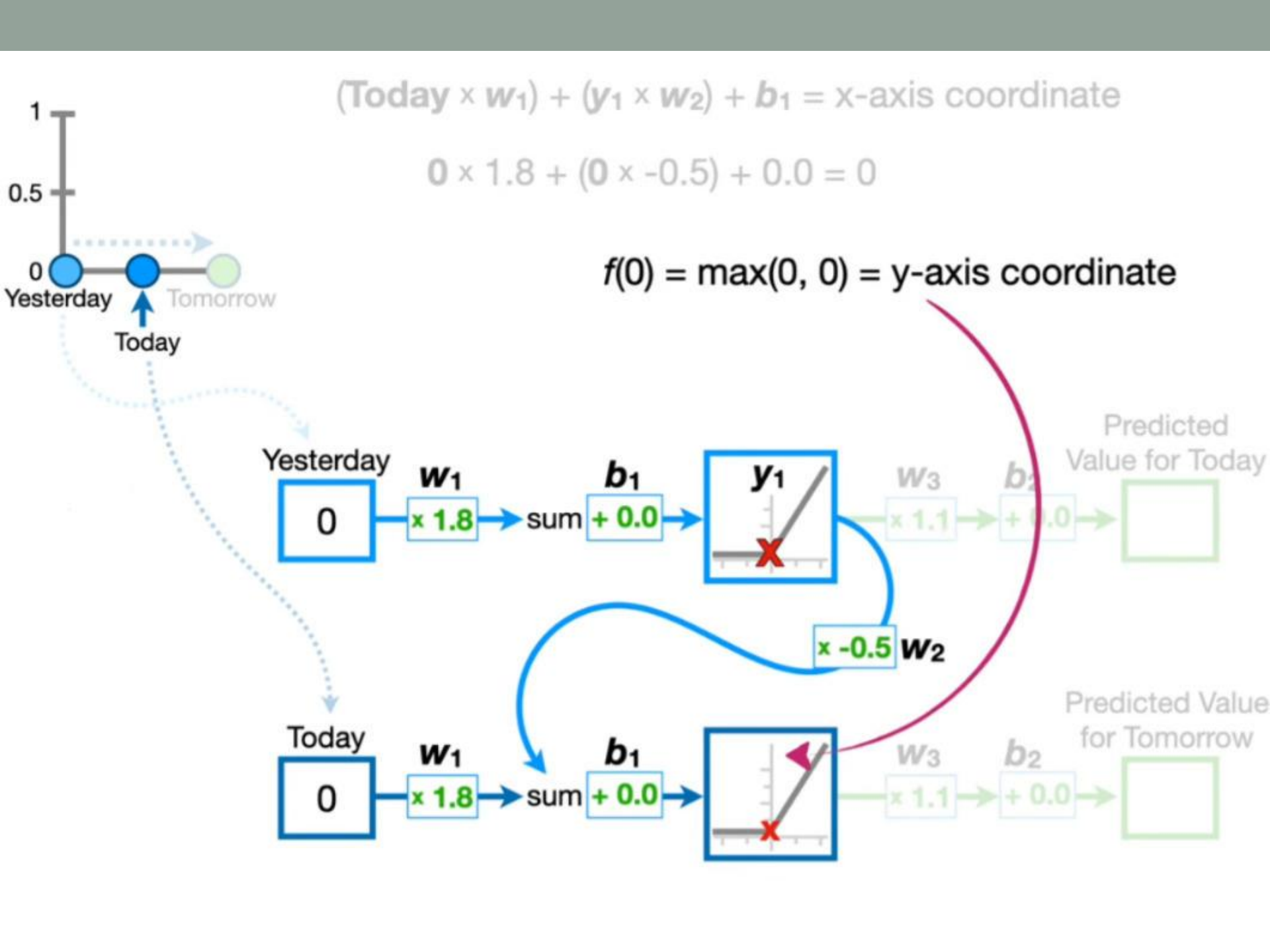


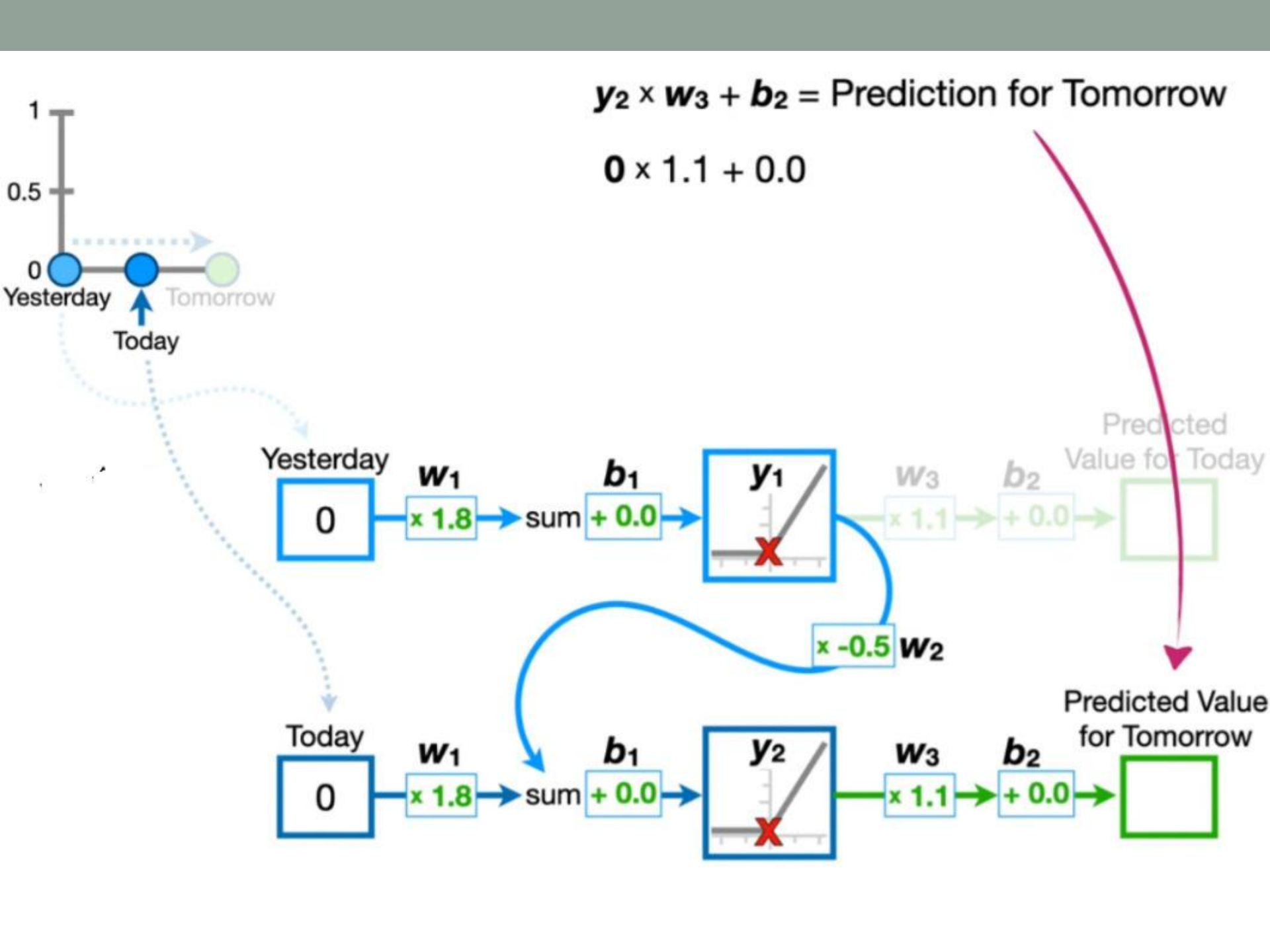


…allows both **yesterday** and **today's** values to influence the final output…




Yesterday × w_1 + b_1 = x-axis coordinate
0 × 1.8 + 0.0 = 0
f(0) = max(0, 0) = 0
1
0.5
0
Yesterday
Tomorrow
Today
Yesterday
0
w_1
× 1.8
sum
b_1
+ 0.0
y_1
w_3
× 1.1
b_2
+ 0.0
Predicted Value for Today
× -0.5 w_2
Today
w_1
× 1.8
sum
b_1
+ 0.0
w_3
× 1.1
b_2
+ 0.0
Predicted Value for Tomorrow


(Today x w_1) + (y_1 x w_2) + b_1 = x-axis coordinate
0 x 1.8 + (0 x -0.5) + 0.0 = 0
1
0.5
0
Yesterday
Tomorrow
Today
$f(0)$ = max(0, 0) = y-axis coordinate
Yesterday
0
w_1
x 1.8
sum
b_1
+ 0.0
y_1
w_3
x 1.1
b_2
+ 0.0
Predicted Value for Today
x -0.5
w_2
Today
0
w_1
x 1.8
sum
b_1
+ 0.0
w_3
x 1.1
b_2
+ 0.0
Predicted Value for Tomorrow


$y_2 \times w_3 + b_2$ = Prediction for Tomorrow
0 x 1.1 + 0.0
1
0.5
0
Yesterday
Tomorrow
Today
Yesterday
0
w_1
x 1.8
sum
b_1
+ 0.0
y_1
w_3
x 1.1
b_2
+ 0.0
Predicted Value for Today
x -0.5
w_2
Predicted Value for Tomorrow
Today
0
w_1
x 1.8
sum
b_1
+ 0.0
y_2
w_3
x 1.1
b_2
+ 0.0