Better Stock Prediction with Long Short-Term Memory (LSTM)!!!

LSTM: Main Ideas

1 **A Problem:** Although *basic* **Recurrent Neural Networks** (**RNNs**) are cool because they can expand, by unrolling, to handle any size dataset without increasing the number of weights and biases we need to train, they have the **Vanishing/Exploding Gradient Problems**.

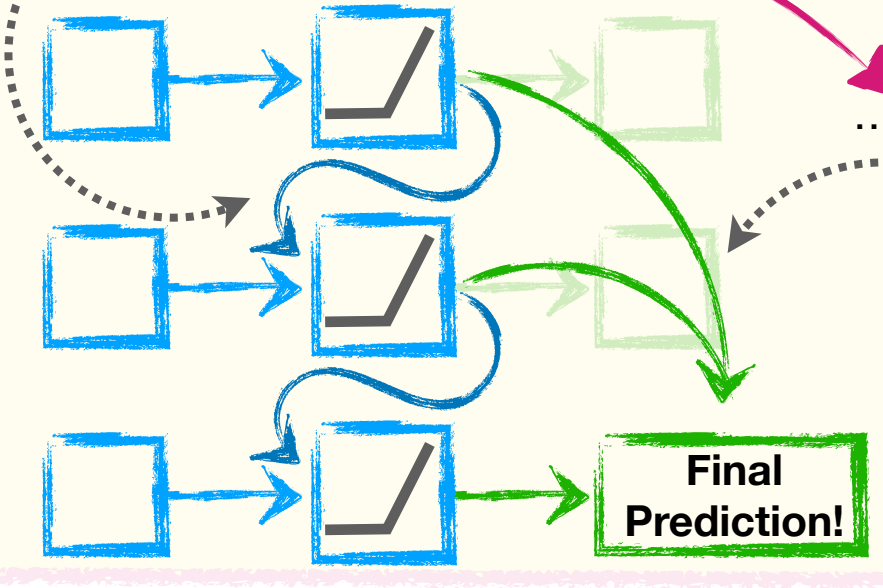
When the weight that connects each unrolled copy of the network is between **-1** and **1**, we can't actually use that much data to train the RNN because it will scale the older data points so much that they essentially vanish, preventing them from contributing to the gradient.

And when the weight that connects each unrolled copy of the network is less than **-1** or greater than **1**, it doesn't take many data points to completely overwhelm, or explode, the gradient.

2 **A Solution: Long Short-Term Memory** networks (**LSTMs**) extend the *basic* RNN in a relatively simple way that can reduce the effects of this problem.

Essentially, instead of using a single connection to combine all of the data points into a prediction, **LSTMs** use two separate paths:

…one for **Short-Term Memories**…

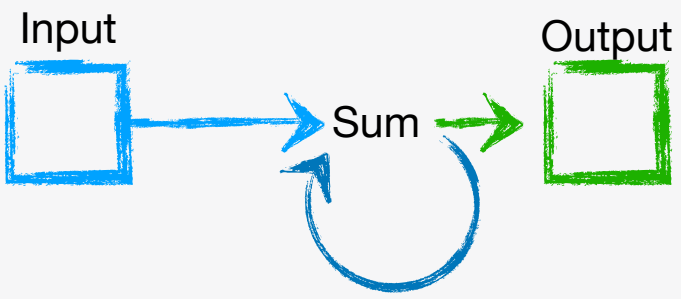


…and one for **Long-Term Memories**.

Bam!

Now let's talk about the details!

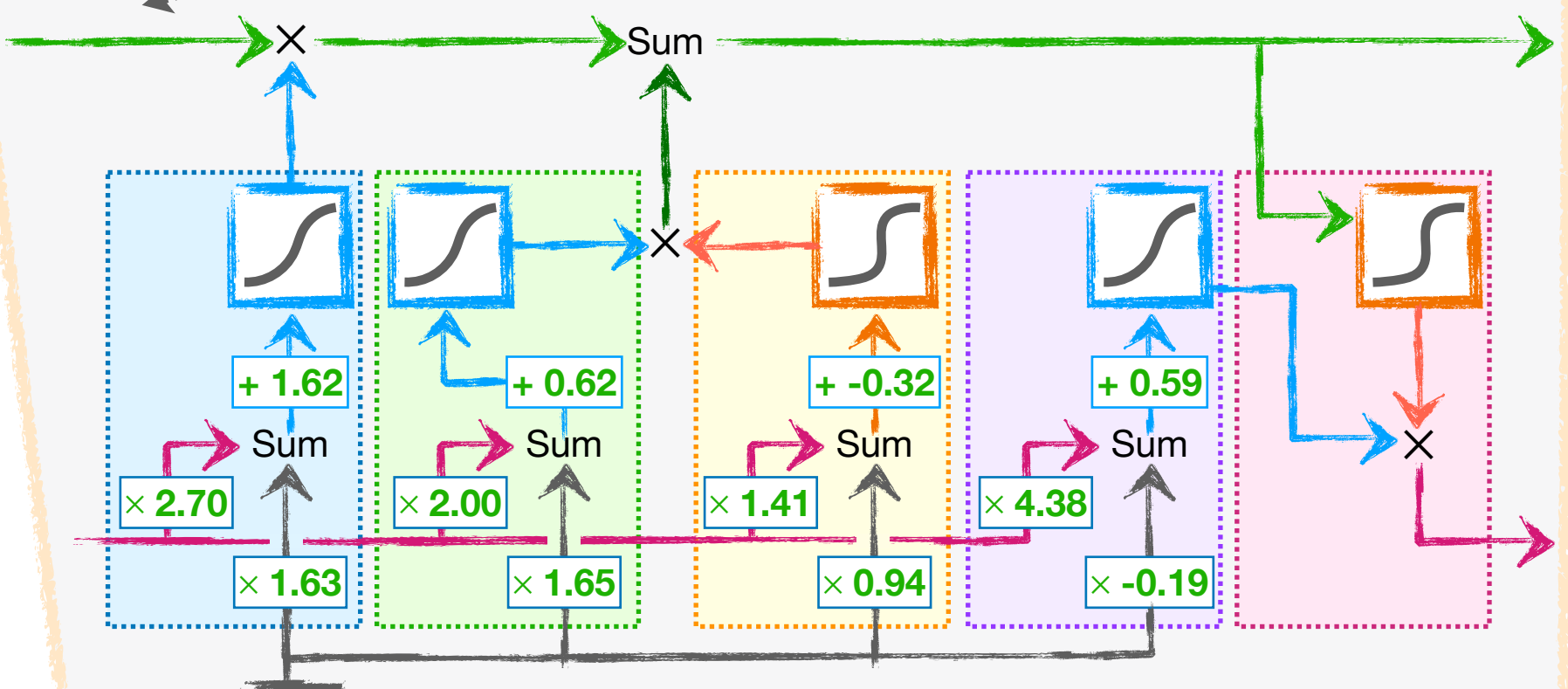
LSTM: Details



1 Compared to *basic* RNNs…

…LSTMs look pretty complicated.

However, LSTMs, like all neural networks, are made from relatively simple parts.

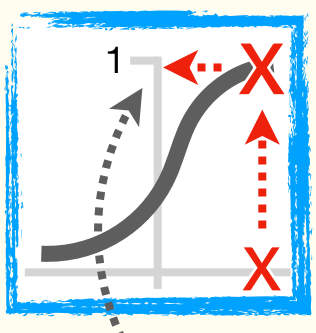


2 Specifically, LSTMs make use of two activation functions that we haven't worked with yet: the **Sigmoid**…

…and the **hyperbolic tangent**, which we refer to by the shorthand, and somewhat backward, term, **Tanh**.

3 The equation for the Sigmoid activation function looks fancy, but all it does is convert any x-axis coordinate into a y-axis coordinate between **0** and **1**.

$$f(x) = \frac{e^x}{e^x + 1}$$



For example, if we plug in this x-axis coordinate, **5**, into the equation for the Sigmoid activation function…

$$f(5) = \frac{e^5}{e^5 + 1} = 0.99$$

Remember '***e***' is **Euler's number**, which is roughly **2.72**.

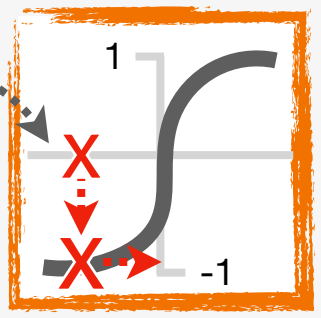
…then we get **0.99** as the y-axis coordinate.

LSTM: Details

4 The **Tanh** activation function looks even fancier, but all it does is convert any x-axis coordinate into a y-axis coordinate between **-1** and **1**.

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

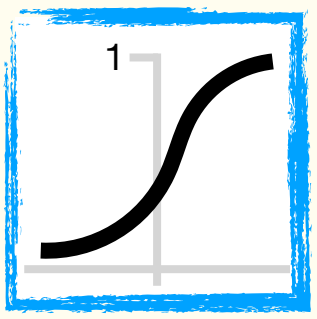
For example, if we plug in this x-axis coordinate, **-2**, into the equation for the Tanh activation function…



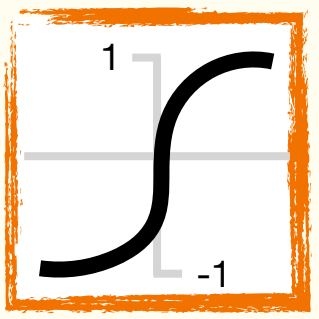
$$f(-2) = \frac{e^{-2} - e^{2}}{e^{-2} + e^{2}} = -0.96$$

…then we get **-0.96** as the y-axis coordinate.

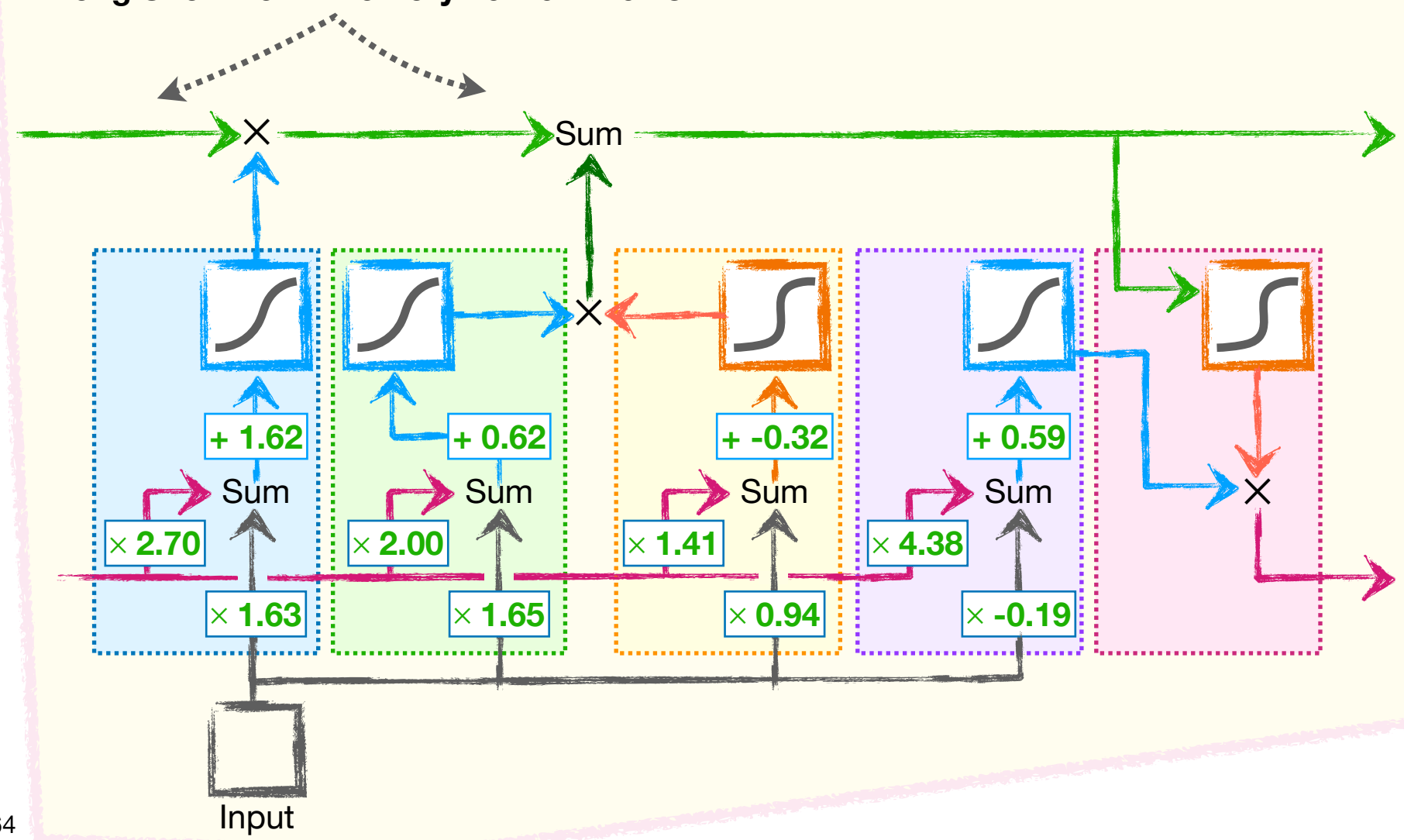
5 So, now that we know that the Sigmoid activation function turns any input into a number between **0** and **1**…



…and the Tanh activation function turns any input into a number between **-1** and **1**…



…let's talk about how a single unit of a **Long Short-Term Memory** network works!



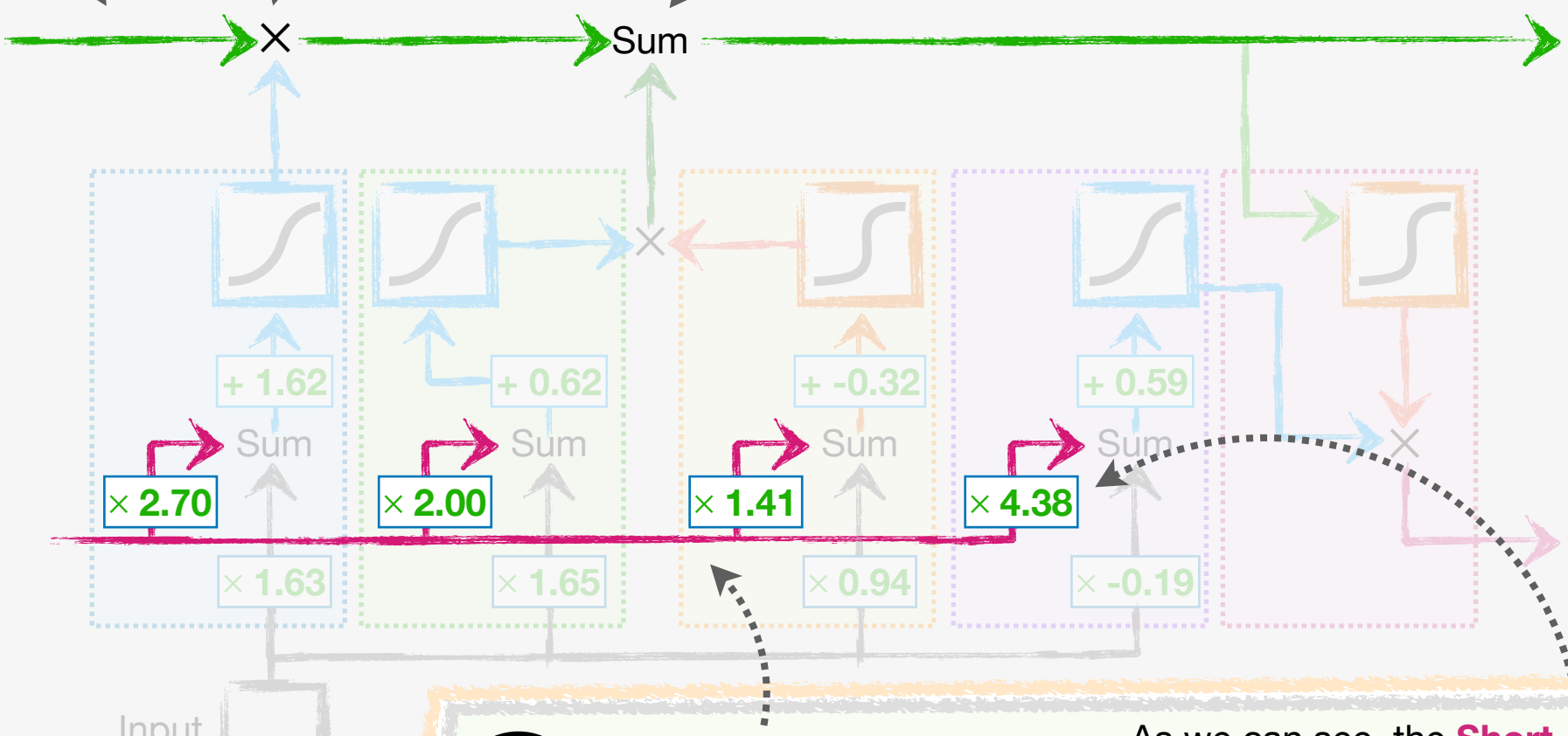
LSTM: Details

6 First, the **green line** that runs all the way across the top of the unit is called the **Cell State** and represents the **Long-Term Memory**.

And, although the **Long-Term Memory** can be modified by this multiplication…

…and then later by this addition…

…you'll notice that there are no weights or biases that can modify it directly. The lack of weights allows the **Long-Term Memory** to flow through a series of unrolled units without causing the gradient to **vanish** or **explode**.



7 This **pink line**, called the **Hidden State**, represents the **Short-Term Memory**.

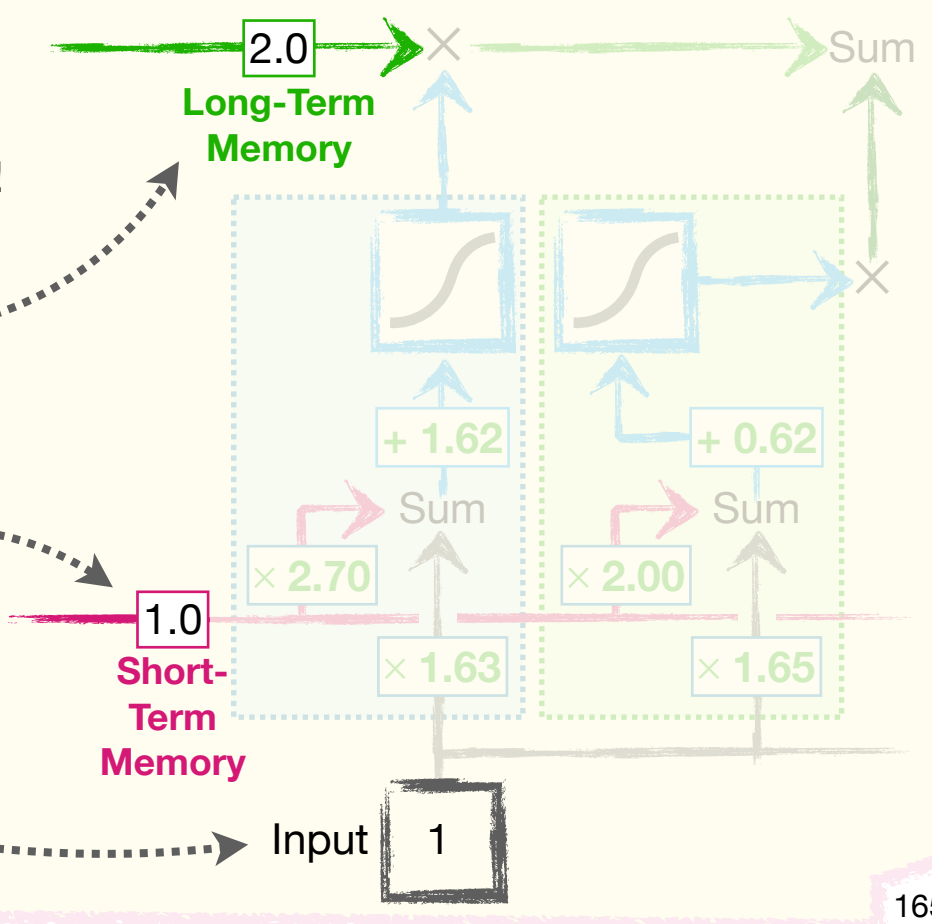
As we can see, the **Short-Term Memory** is directly connected to weights that can modify it.

8 To understand how the **Long-Term Memory** and **Short-Term Memory** interact to create predictions, let's run some numbers through the LSTM unit!

To make the math interesting, let's just assume that the previous **Long-Term Memory** is **2**…

…the previous **Short-Term Memory** is **1**…

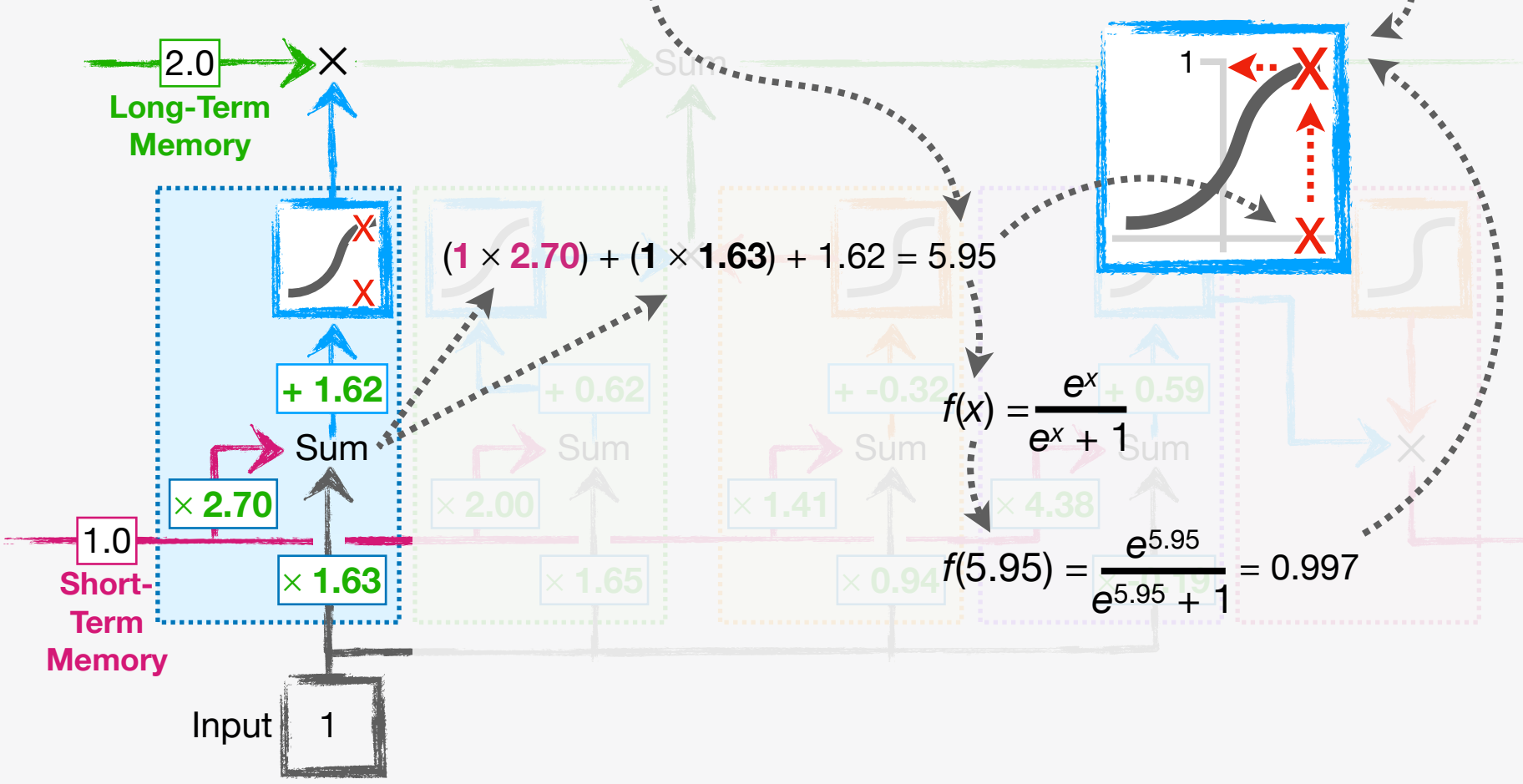
…and the **Input** value is **1**.



LSTM: Details

9 Plugging in the **Short-Term Memory**, **1**, and the **Input** value, **1**, gives us **5.95** as the x-axis coordinate going into the Sigmoid activation function…

…and the Sigmoid activation function gives us the corresponding y-axis coordinate, **0.997**.

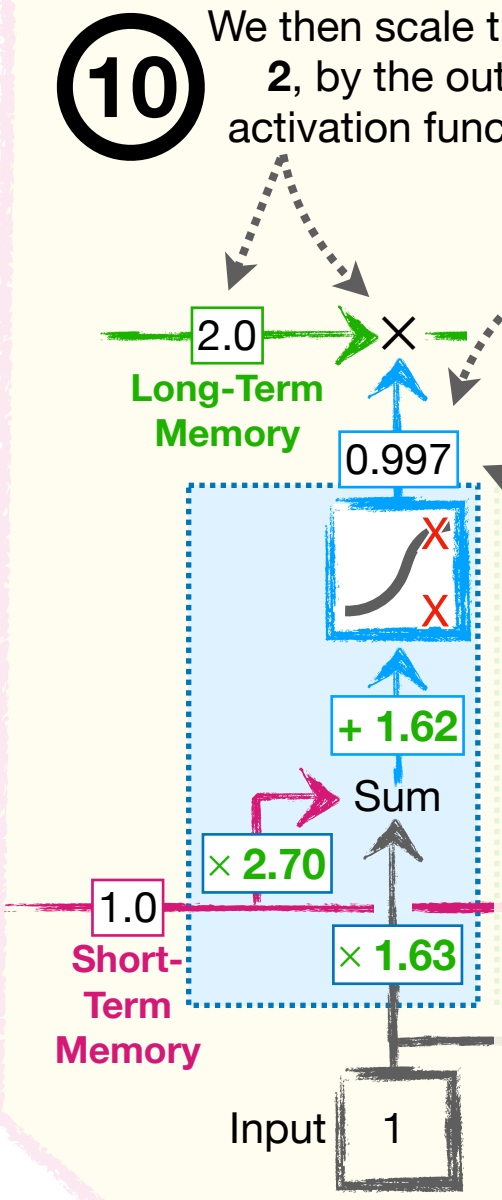


10 We then scale the **Long-Term Memory**, **2**, by the output from the Sigmoid activation function, **0.997**, to get **1.99**.

So the first stage of the LSTM unit reduced the **Long-Term Memory** by a little bit.

In general, because the Sigmoid activation function turns any input into a number between **0** and **1**, its output determines what percentage of the **Long-Term Memory** is remembered.

In this case, we ***remember*** **99.7%** of the **Long-Term Memory**, which is **1.99**.



NOTE: Even though the first part of an LSTM unit determines what percentage of the **Long-Term Memory** ***will be remembered***, it's usually called the **Forget Gate**.

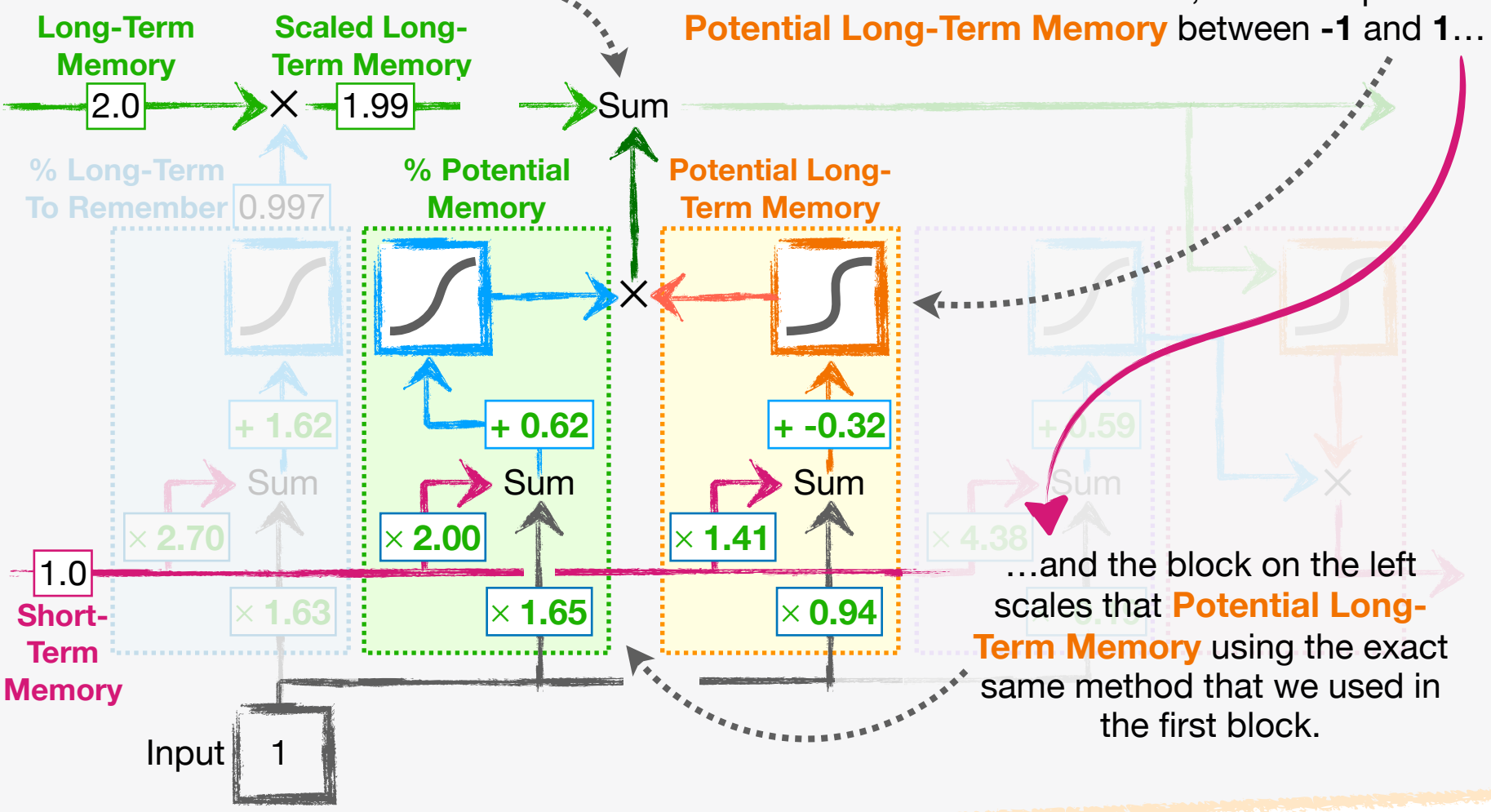
small bam.

LSTM: Details

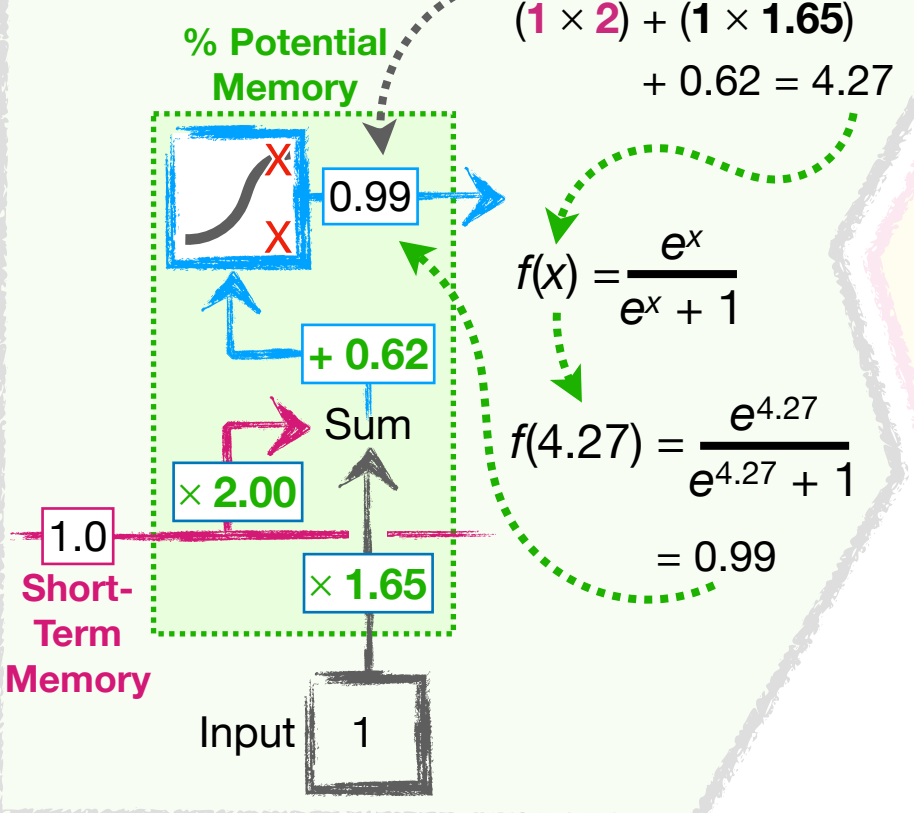
11 In a nutshell, the next two blocks create a **Potential Long-Term Memory** and determine what percentage of that **Potential Long-Term Memory** to add to the **Scaled Long-Term Memory**.

The block on the right uses the **Input** value and the **Short-Term Memory** to create an x-axis coordinate for the Tanh activation function, which outputs a **Potential Long-Term Memory** between **-1** and **1**…

…and the block on the left scales that **Potential Long-Term Memory** using the exact same method that we used in the first block.



12 Doing the math in the block on the left tells us to scale the **Potential Long-Term Memory** by **0.99**…



$(1 \times 2) + (1 \times 1.65) + 0.62 = 4.27$

$$f(x) = \frac{e^x}{e^x + 1}$$

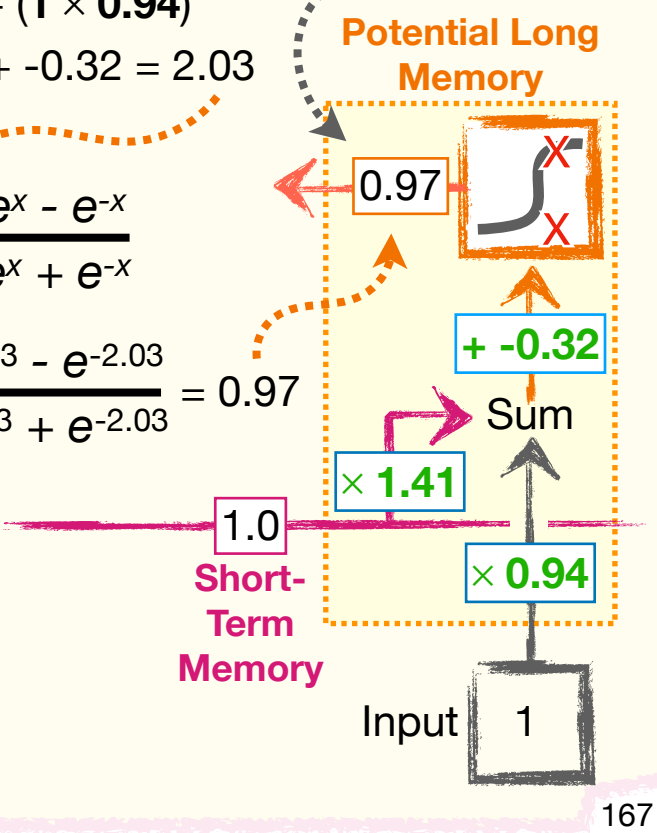
$$f(4.27) = \frac{e^{4.27}}{e^{4.27} + 1} = 0.99$$

13 …and doing the math in the block on the right tells us that the new **Potential Long-Term Memory** is **0.97**.

$(1 \times 1.41) + (1 \times 0.94) + -0.32 = 2.03$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f(2.03) = \frac{e^{2.03} - e^{-2.03}}{e^{2.03} + e^{-2.03}} = 0.97$$

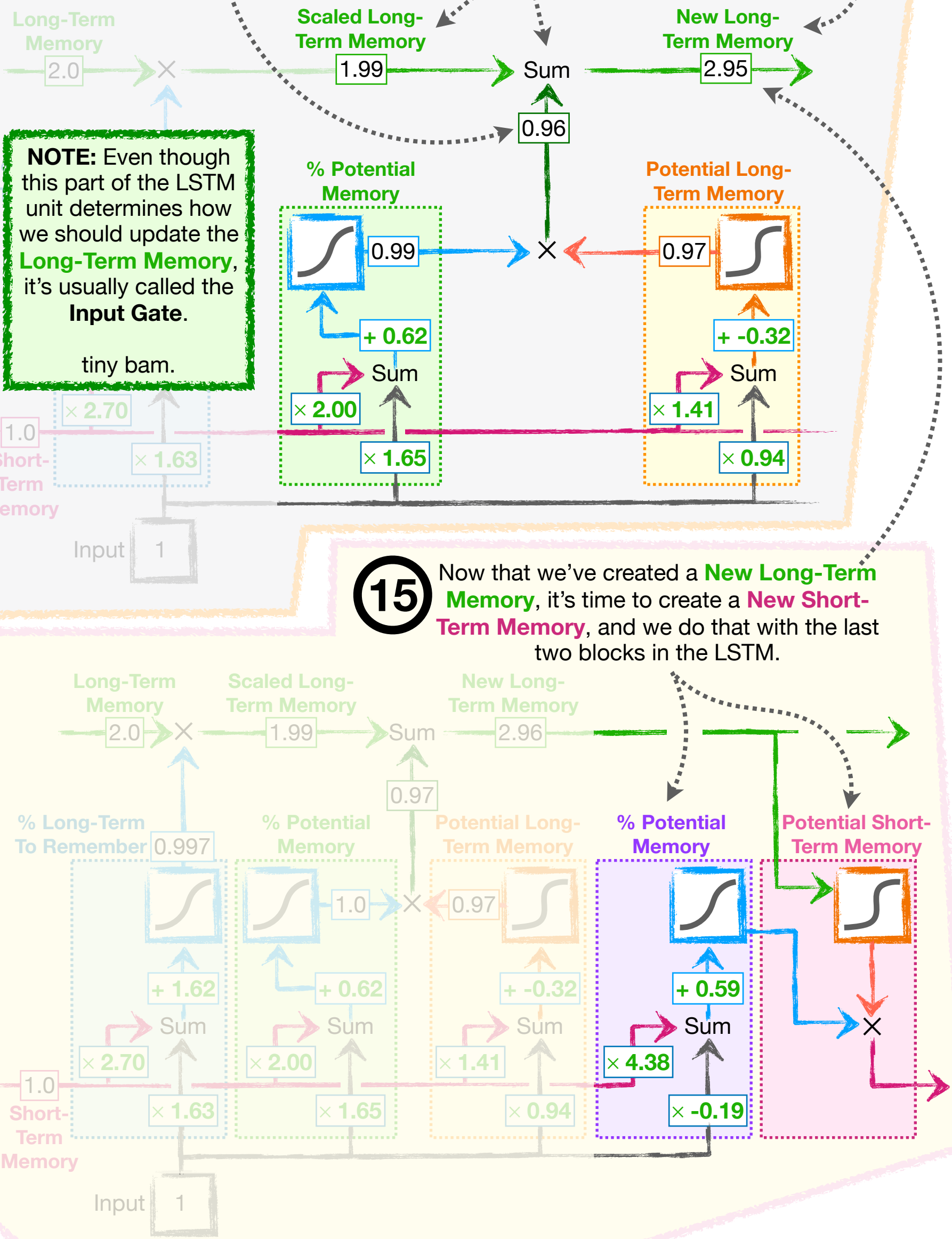


LSTM: Details

14 We then scale the **Potential Long-Term Memory**, **0.97**, by **0.99** to get **0.96**…

…which we add to the **Scaled Long-Term Memory**…

…to get a **New Long-Term Memory**, **2.95**.



NOTE: Even though this part of the LSTM unit determines how we should update the **Long-Term Memory**, it's usually called the **Input Gate**.

tiny bam.

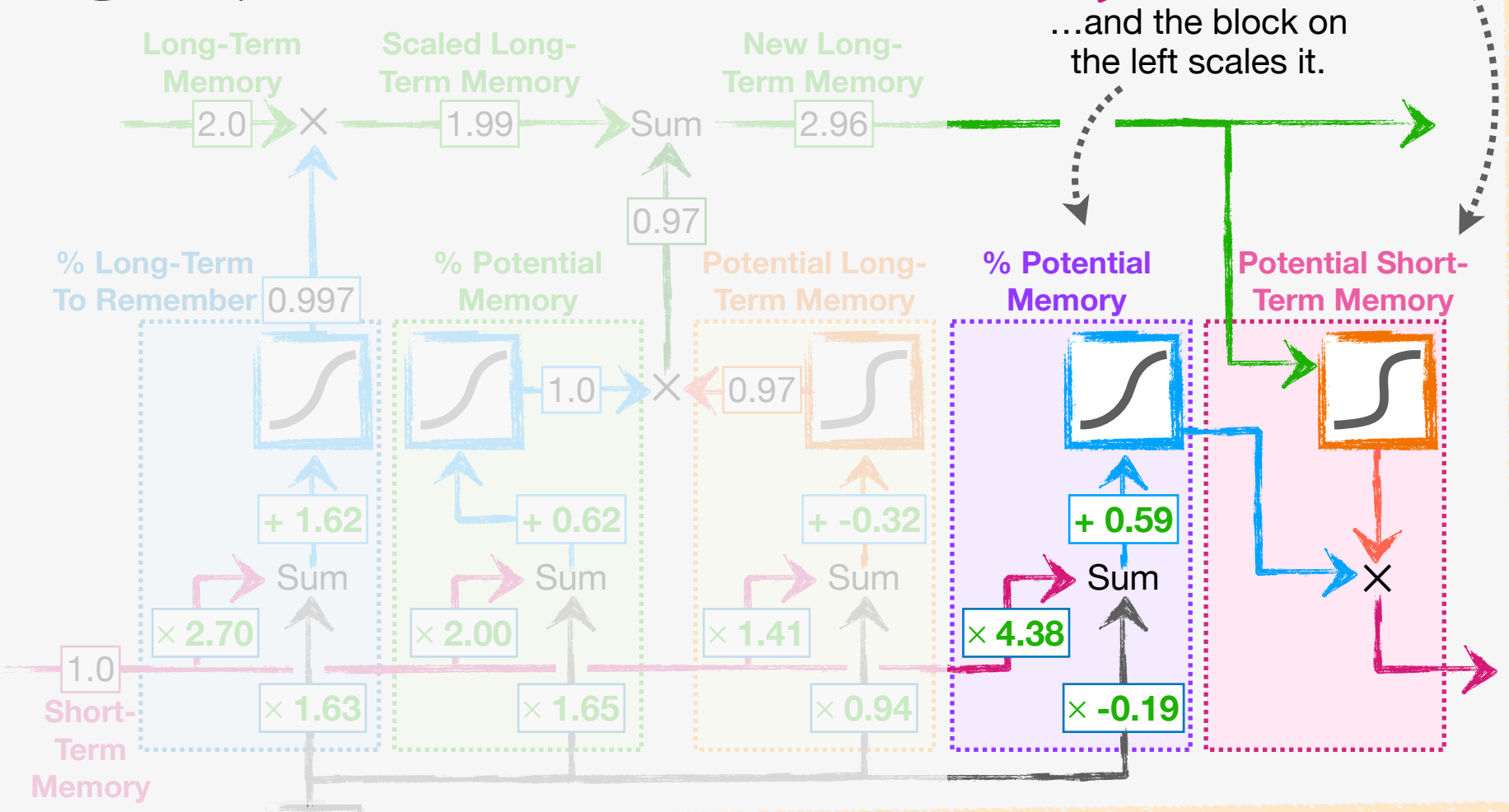
15 Now that we've created a **New Long-Term Memory**, it's time to create a **New Short-Term Memory**, and we do that with the last two blocks in the LSTM.

LSTM: Details

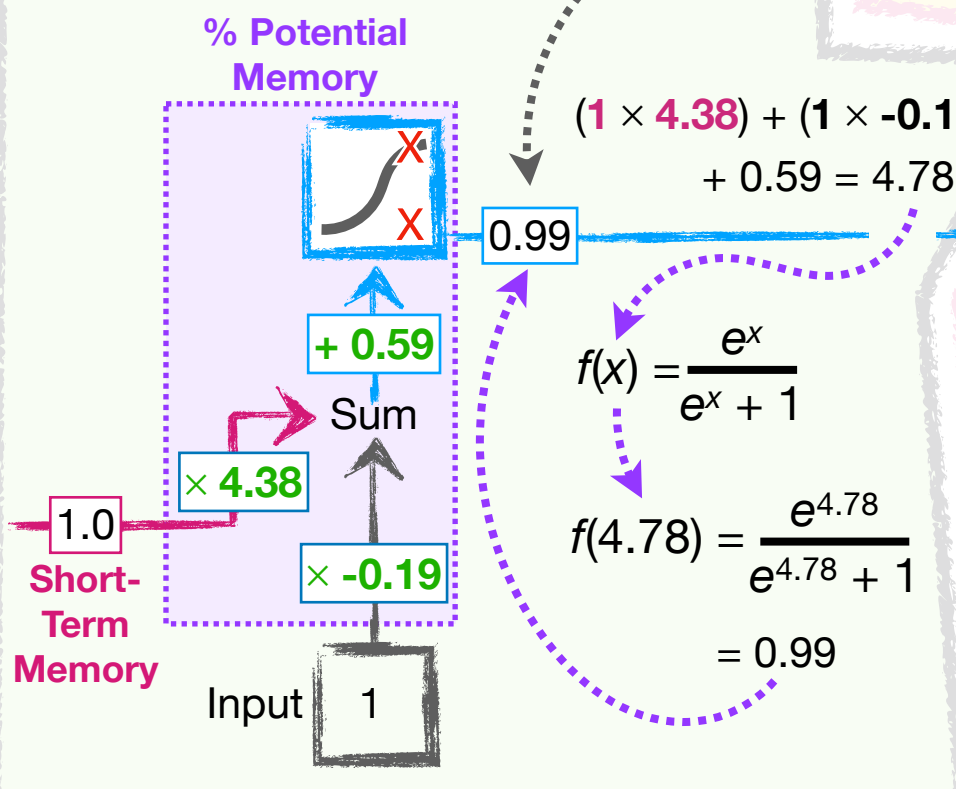
16 The **New Short-Term Memory** is created in a way that's very similar to how we updated the **Long-Term Memory**.

The block on the right creates a **Potential Short-Term Memory**…

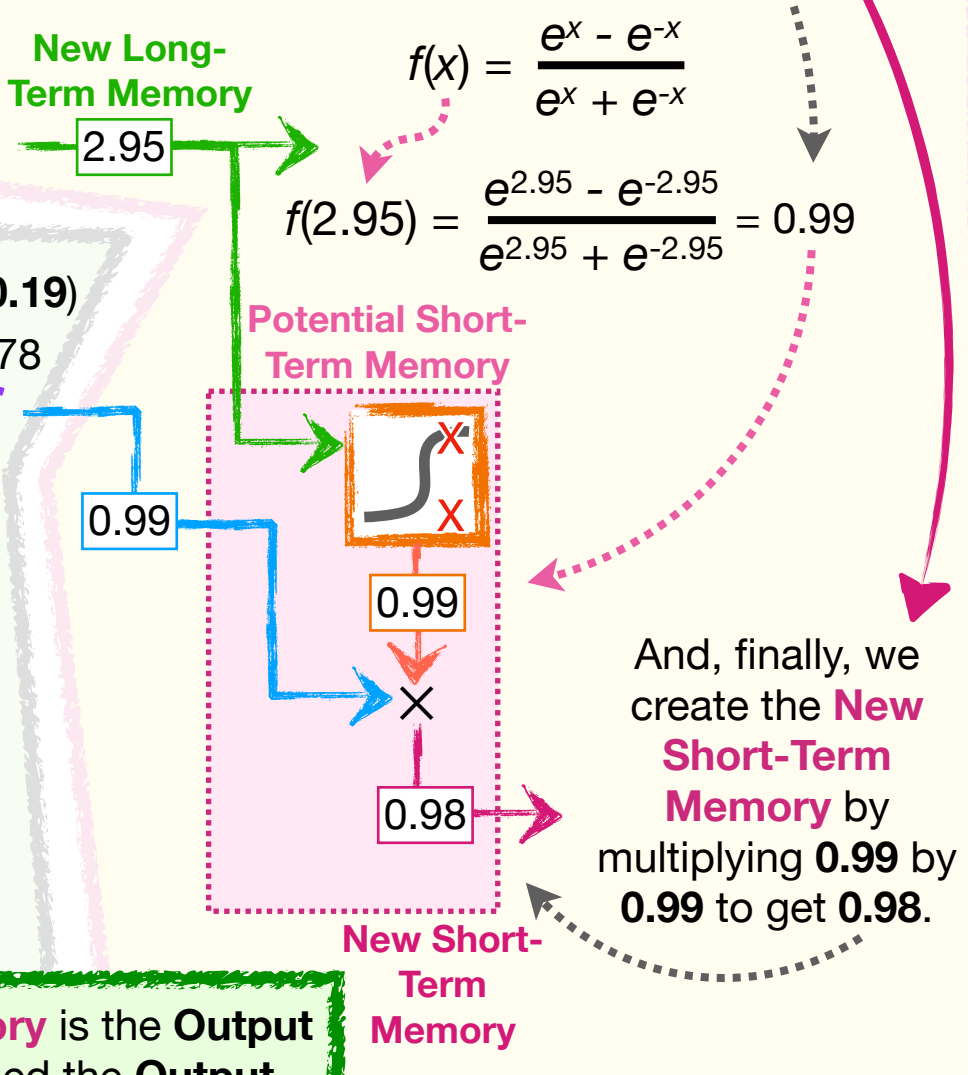
…and the block on the left scales it.



17 Doing the math in the block on the left tells us to scale the **Potential Short-Term Memory** by **0.99**…



18 …and doing the math in the block on the right first tells us that the **Potential Short-Term Memory** is **0.99**…



NOTE: Because the **New Short-Term Memory** is the **Output** from this entire LSTM unit, this stage is called the **Output Gate**. And for once, the terminology makes sense. **BAM!**

LSTM: Details

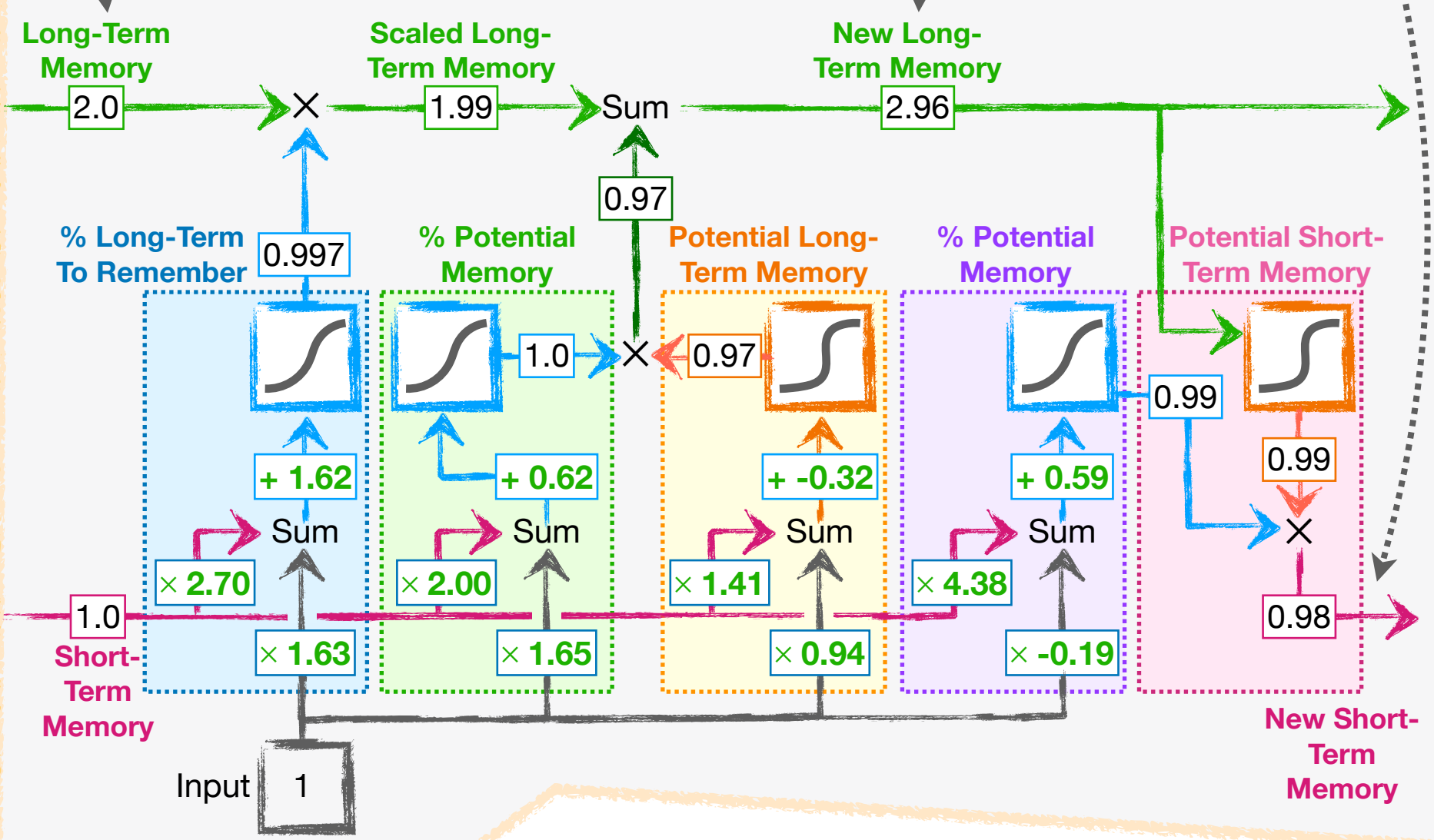
19

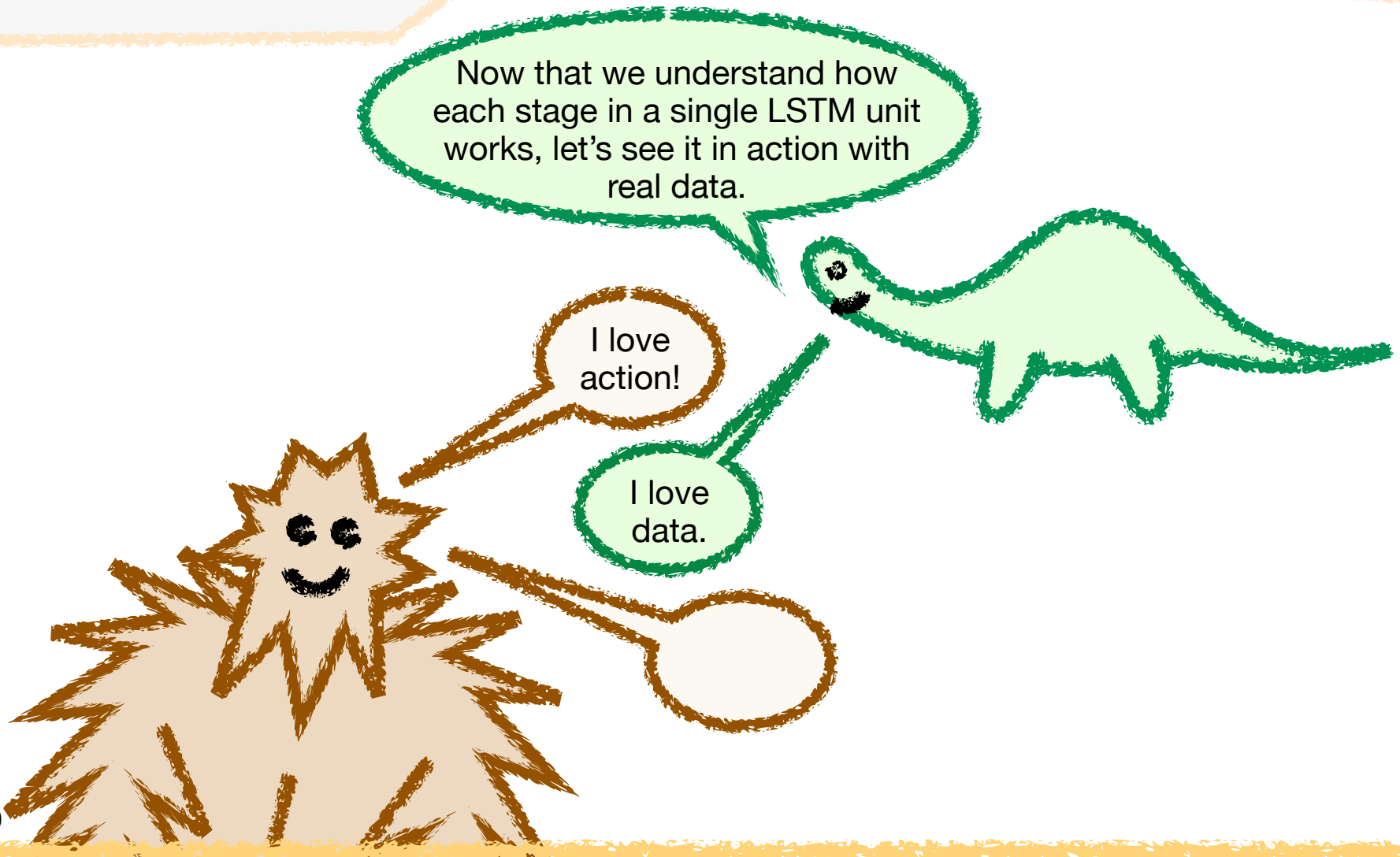
Putting everything together, we can now see how an LSTM unit combines a **Long-Term Memory** with a **Short-Term Memory** and an **Input** value…

…to create a **New Long-Term Memory**…

…and a **New Short-Term Memory**.

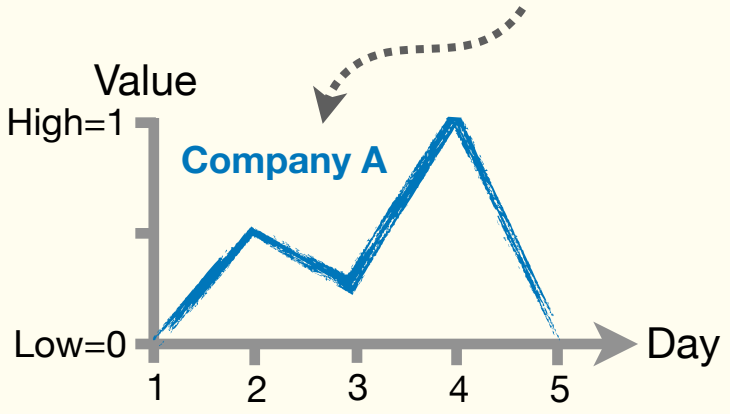
BAM!!!

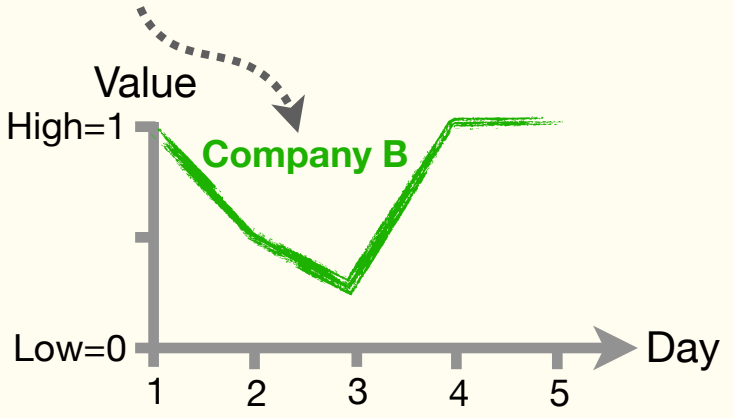




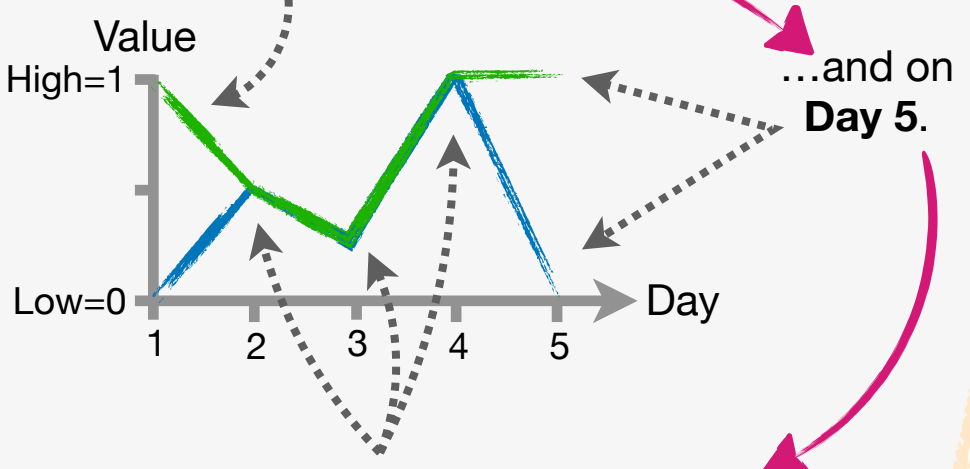
LSTM: Step-by-Step

1 To illustrate how an LSTM works, we're going to apply one to stock prices from two different companies, **Company A** and **Company B**.





2 If we overlap the data from the two companies, we see that the only differences occur on **Day 1**…

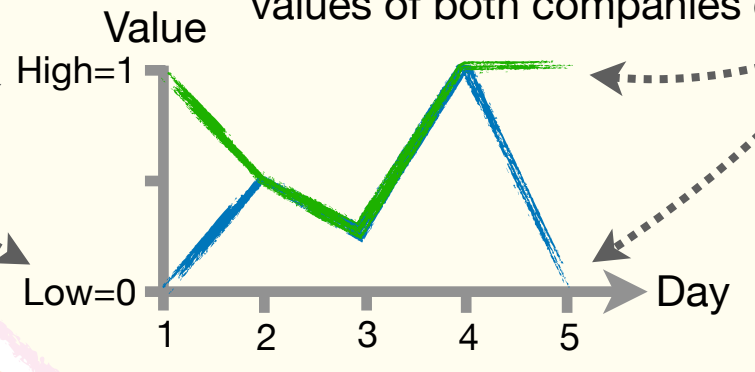


…and on **Day 5**.

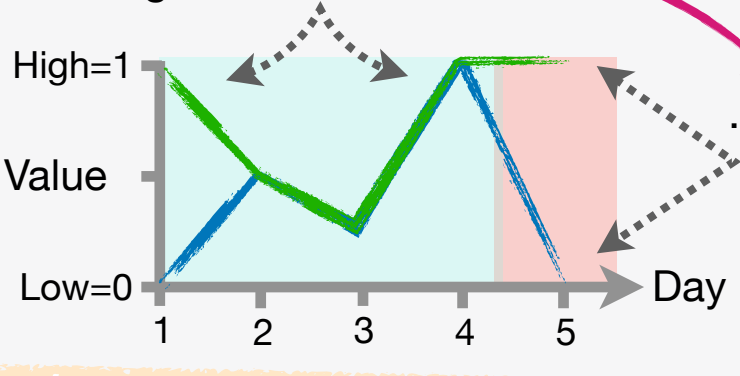
On all of the days in between, **Days 2**, **3** and **4**, both companies have the exact same values.

3 That means that given this sequential data, the LSTM has to keep track of what happened on **Day 1**…

…so it can correctly predict what will happen to the stock market values of both companies on **Day 5**.



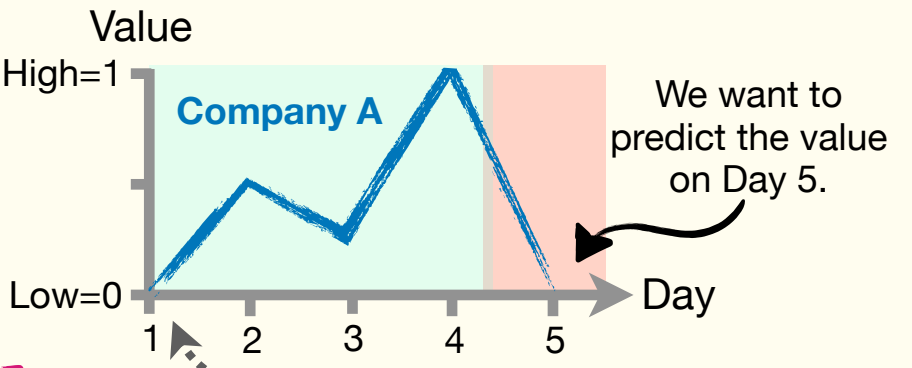
4 So we're going to sequentially run the data from **Days 1** through **4** through an unrolled LSTM…



…and see if those data can correctly predict the stock market values for **Day 5** for both companies.

LSTM: Step-by-Step

5 Now, if we want to sequentially run **Company A**'s values from **Days 1** through **4** through an LSTM, we start by initializing the **Long-Term Memory** and **Short-Term Memory** to **0**…

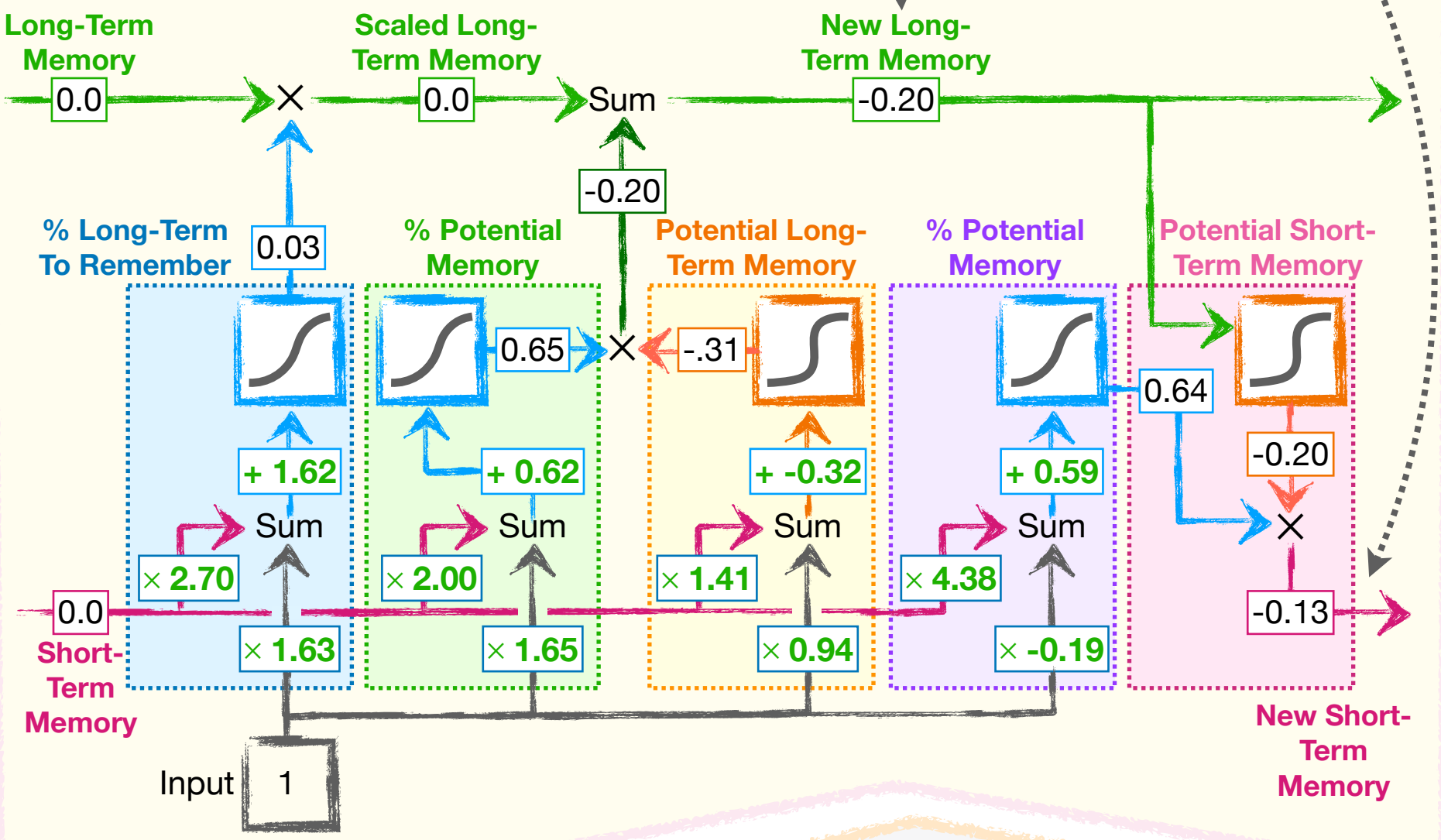


NOTE: The fact that the **Long-Term Memory** and **Short-Term Memory** start out as **0** doesn't mean the LSTM has no memory, but rather that the memories it has are set to **0**. In other words, previous LSTM units may have calculated these memories and the results were **0**.

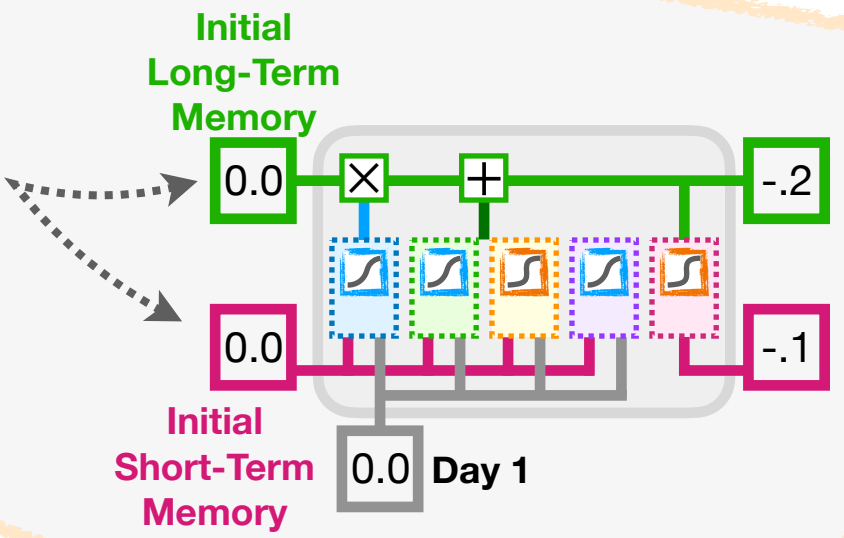
…then we plug the value for **Day 1**, which is **0**, into the **Input**.

When we do the math, we see that the new, or updated, **Long Term-Memory** is **-0.20**…

…and the new, updated **Short-Term Memory** is **-0.13**.

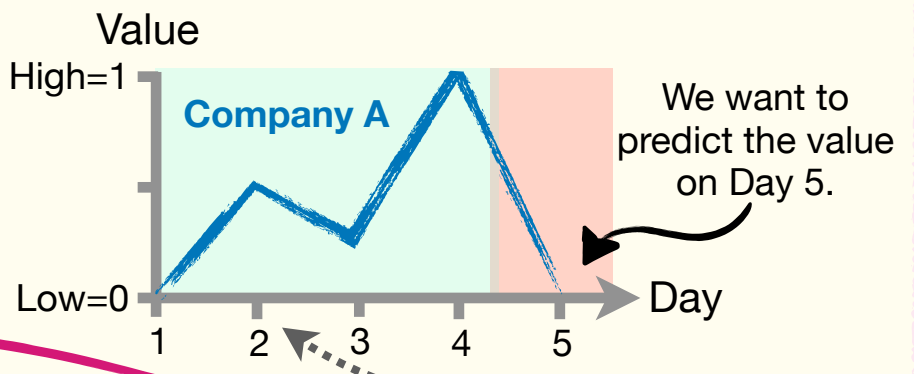


6 Since our LSTM diagram takes up a lot of space, we'll use this simplified diagram to represent each unrolled copy as we input stock market values from each day.

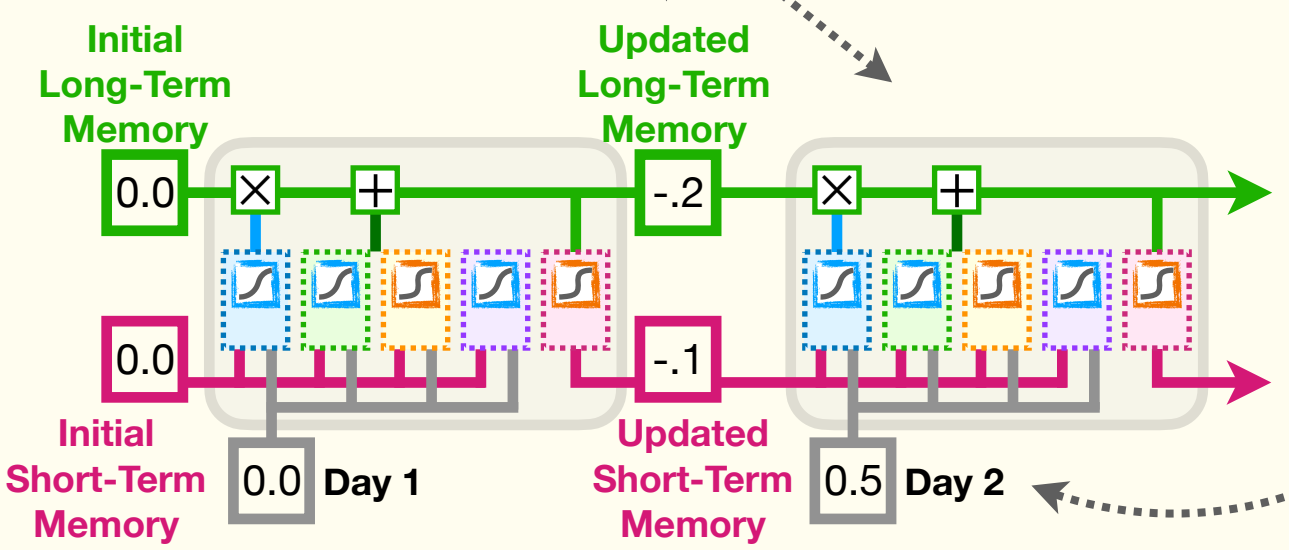


NOTE: The output values are rounded to the nearest tenth.

LSTM: Step-by-Step

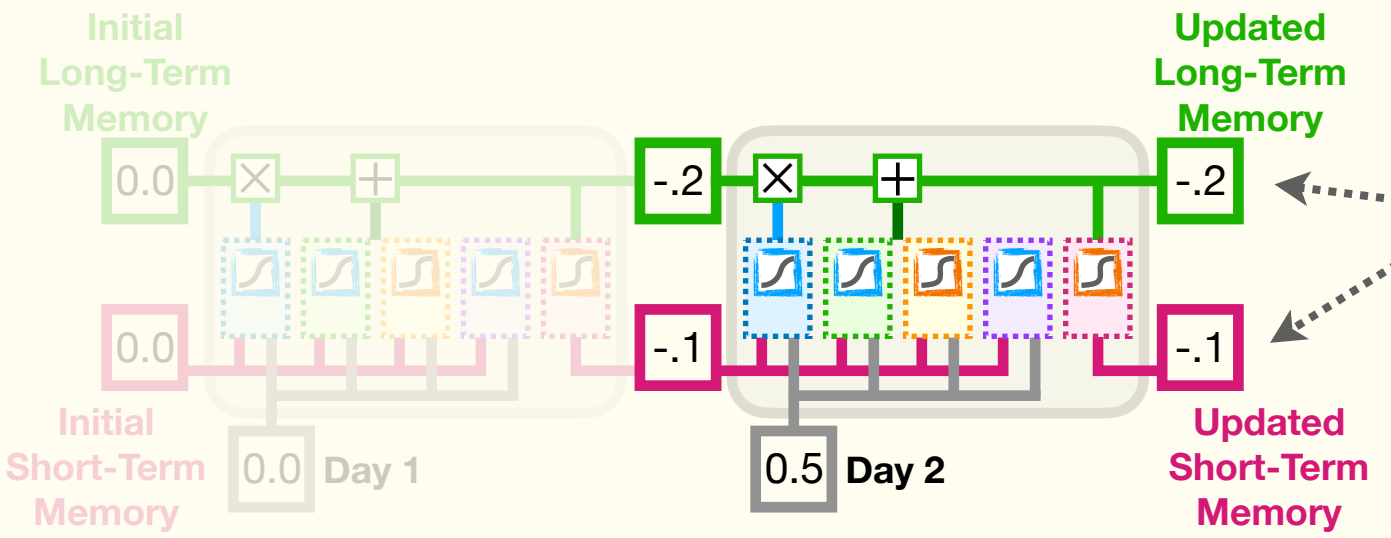


7 Now we unroll the LSTM, using the updated memories…

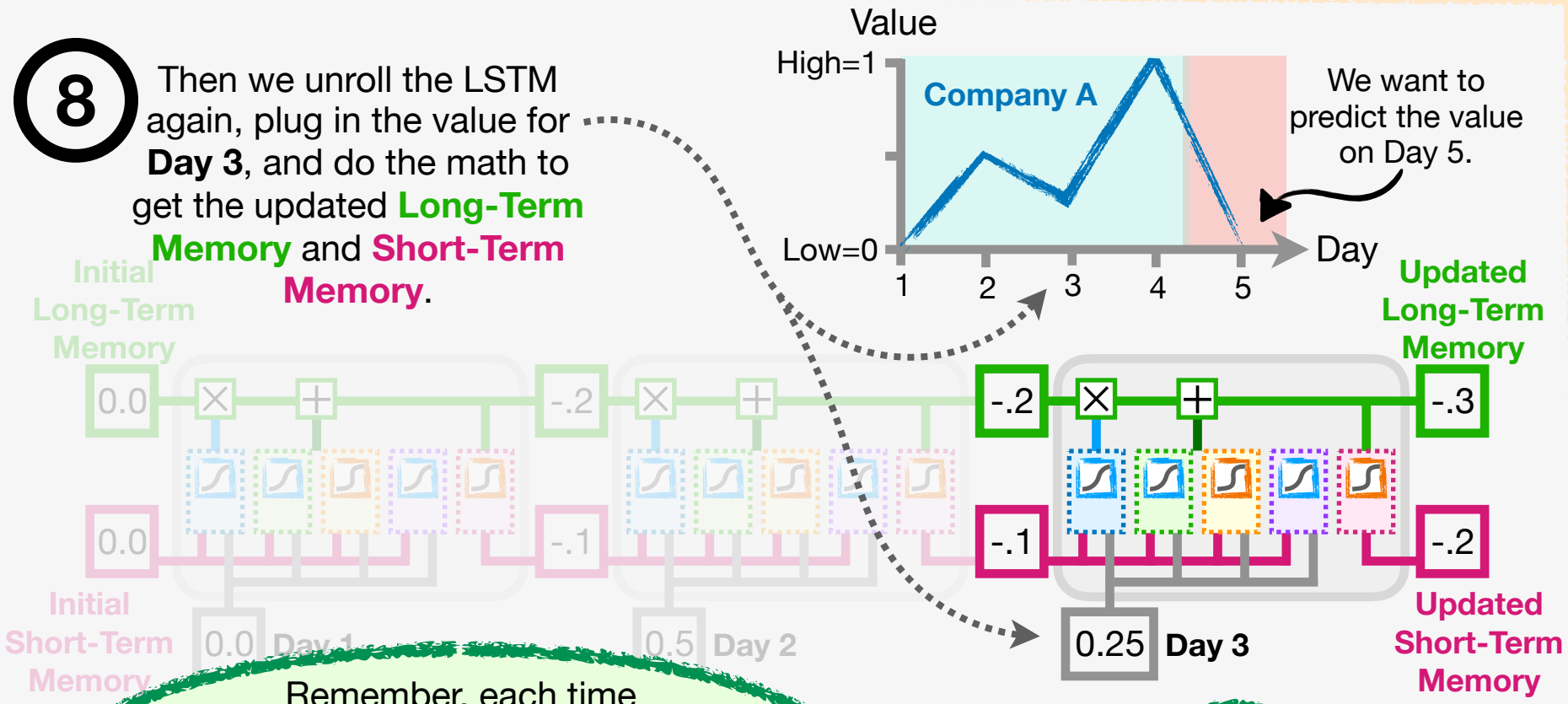


…plug the value from **Day 2**, **0.5**, into the **Input**…

…and do the math to update the **Long-Term Memory** and **Short-Term Memory**.



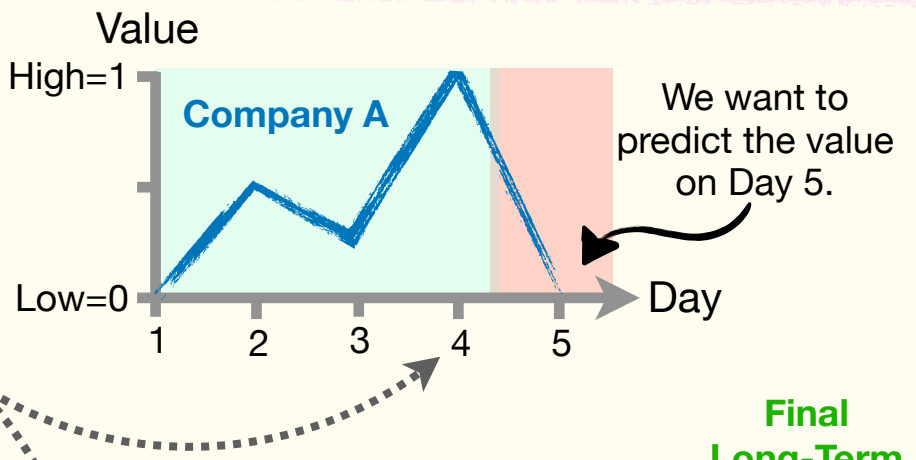
8 Then we unroll the LSTM again, plug in the value for **Day 3**, and do the math to get the updated **Long-Term Memory** and **Short-Term Memory**.

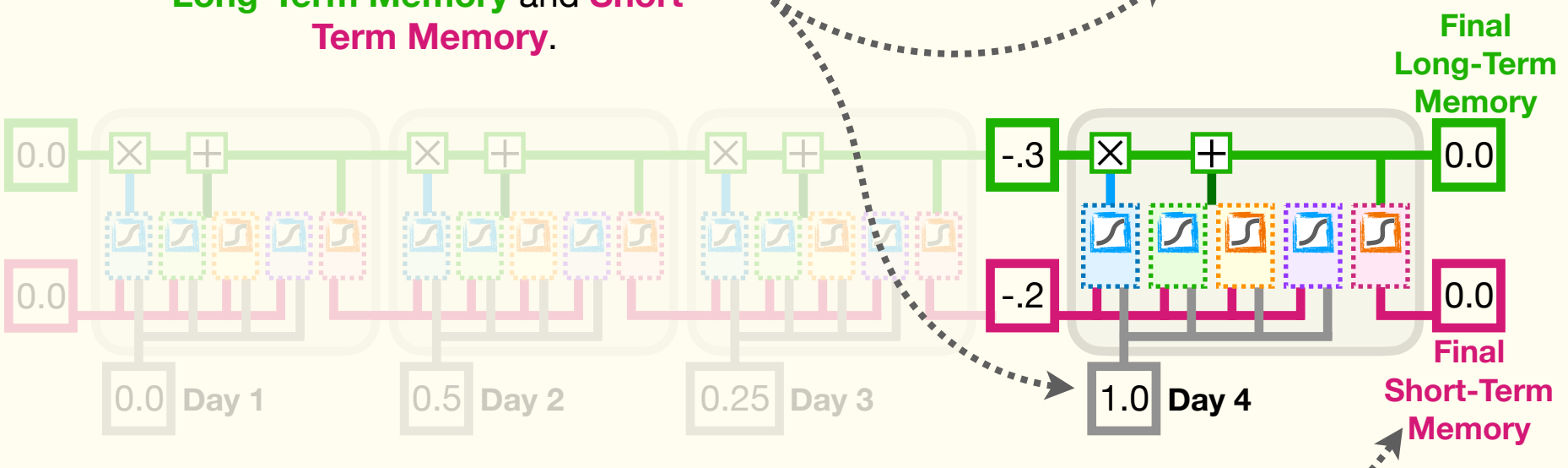


Remember, each time we unroll an RNN—and an LSTM is a type of RNN—we use the exact same weights and biases.

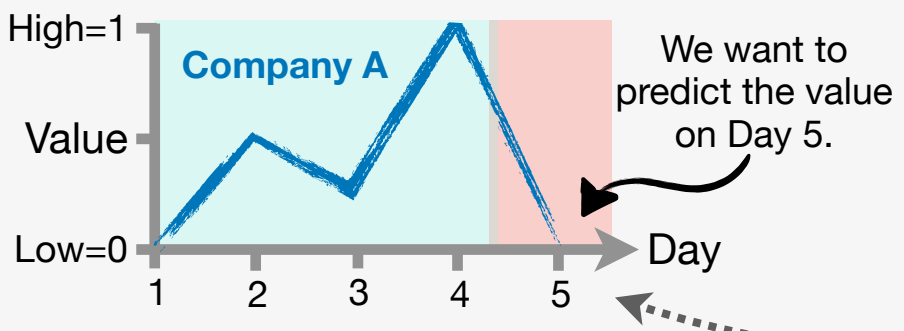
LSTM: Step-by-Step

9 Then we unroll the LSTM one last time, plug in the value for **Day 4**, and do the math to get the final **Long-Term Memory** and **Short-Term Memory**.





10 The final **Short-Term Memory**, **0.0**, is the **Output** from the unrolled LSTM.

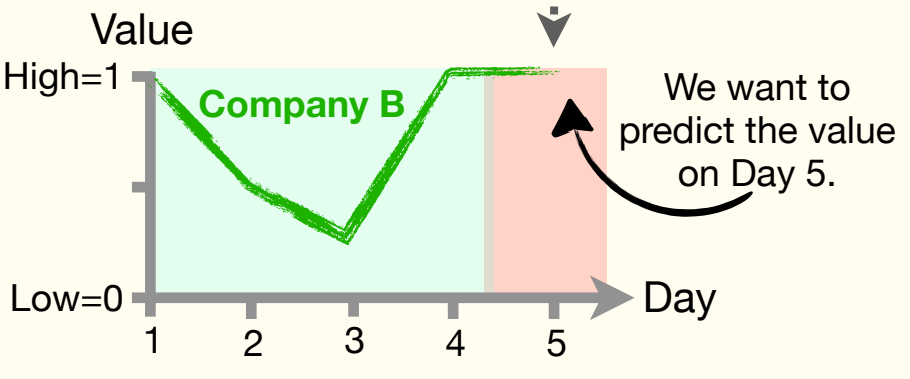


And that means that the **Output** from the LSTM correctly predicts **Company A**'s value for **Day 5**.

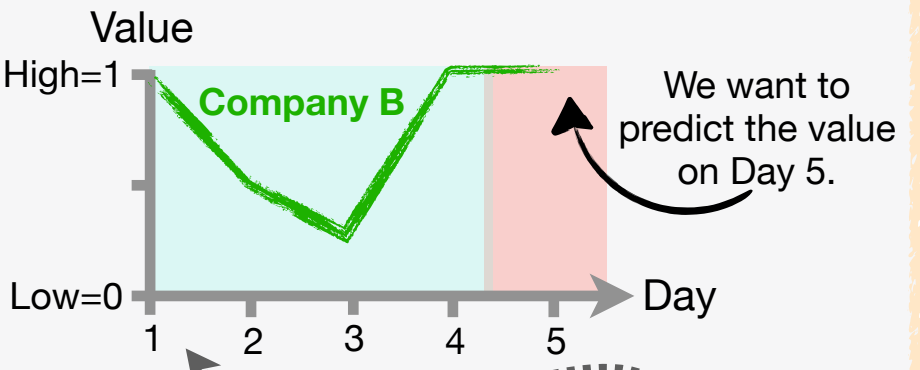
NOTE: To keep this example as simple as possible, this LSTM was only trained to predict the value on **Day 5** and not on any of the other days.

BAM!!!

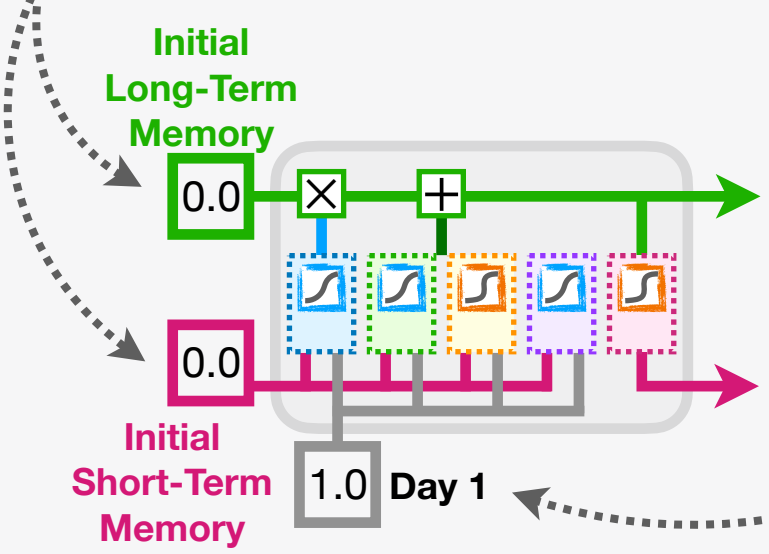
11 Now that we've shown that the LSTM can correctly predict the value on **Day 5** for **Company A**, let's show how the exact same LSTM, with the exact same weights and biases, can correctly predict the value on **Day 5** for **Company B**.



LSTM: Step-by-Step

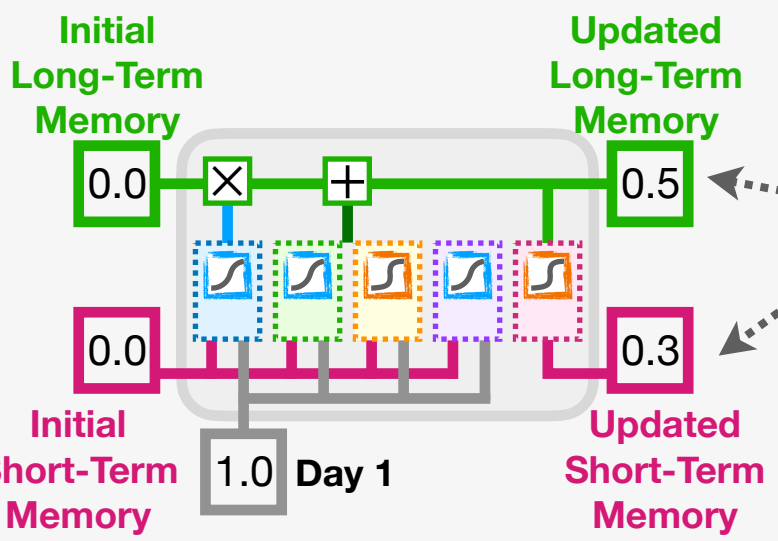


12 So, just like we did with **Company A**, we initialize the **Long-Term Memory** and **Short-Term Memory** to **0**…

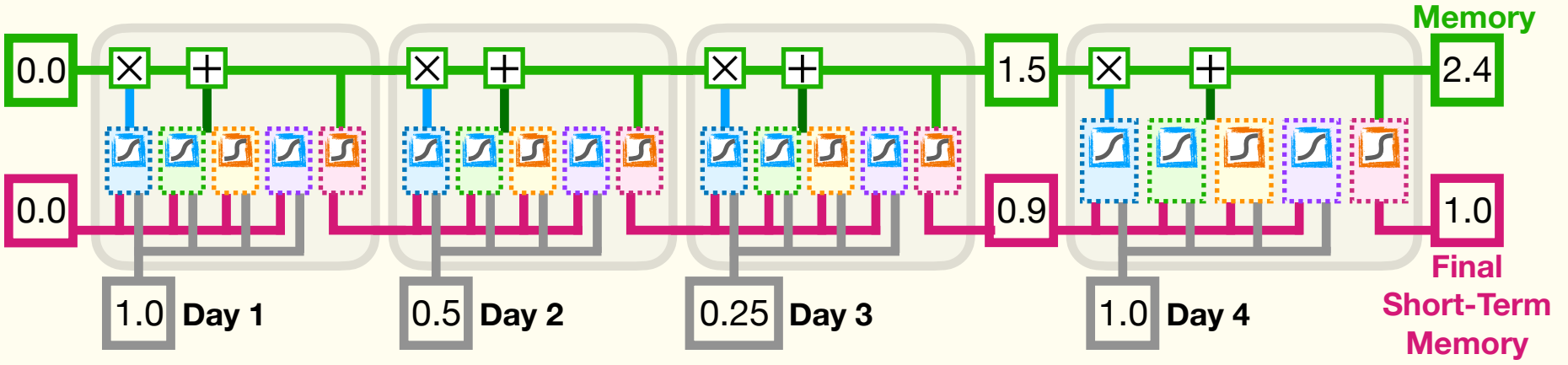


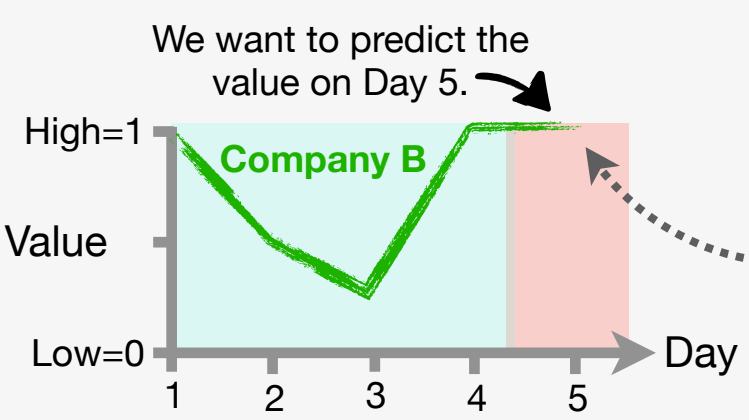
…then we plug in the value for **Day 1**, which in this case is **1.0**.

Then we do the math, using the exact same weights that we used before with **Company A**, to get the updated **Long-Term Memory** and **Short-Term Memory**.



13 And we unroll the LSTM and do the math with the remaining input values…





14 …and the final **Short-Term Memory**, **1.0**, is the **Output** from the unrolled LSTM.

And that means that the **Output** from the LSTM correctly predicts **Company B**'s value for **Day 5**.

DOUBLE BAM!!

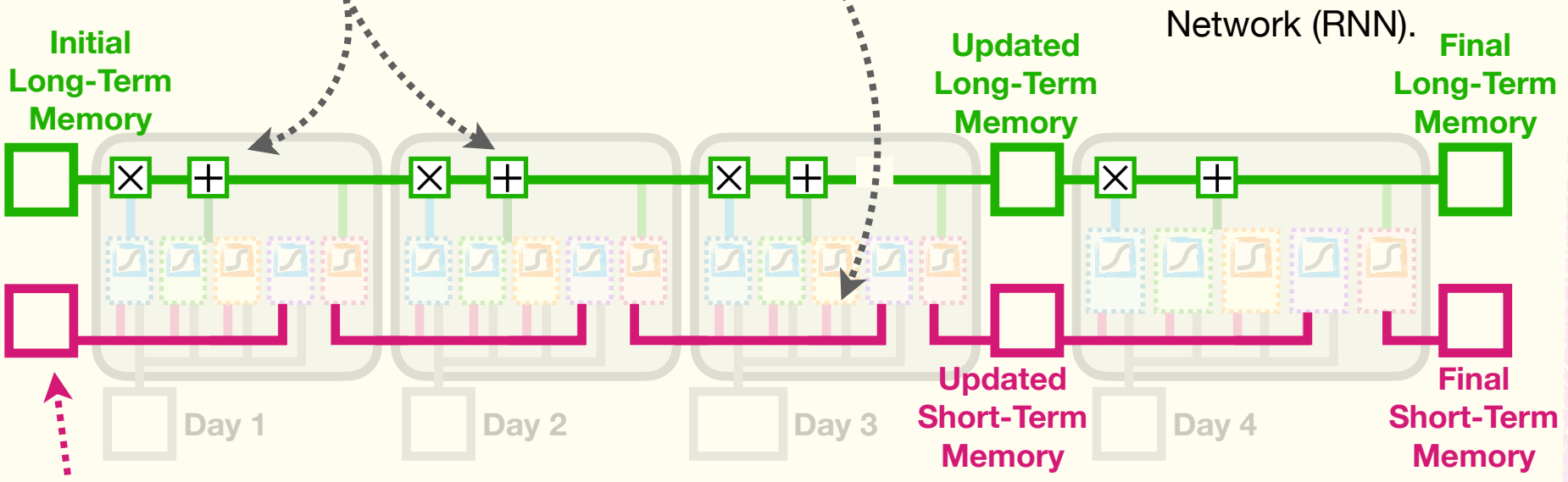
LSTM: Step-by-Step

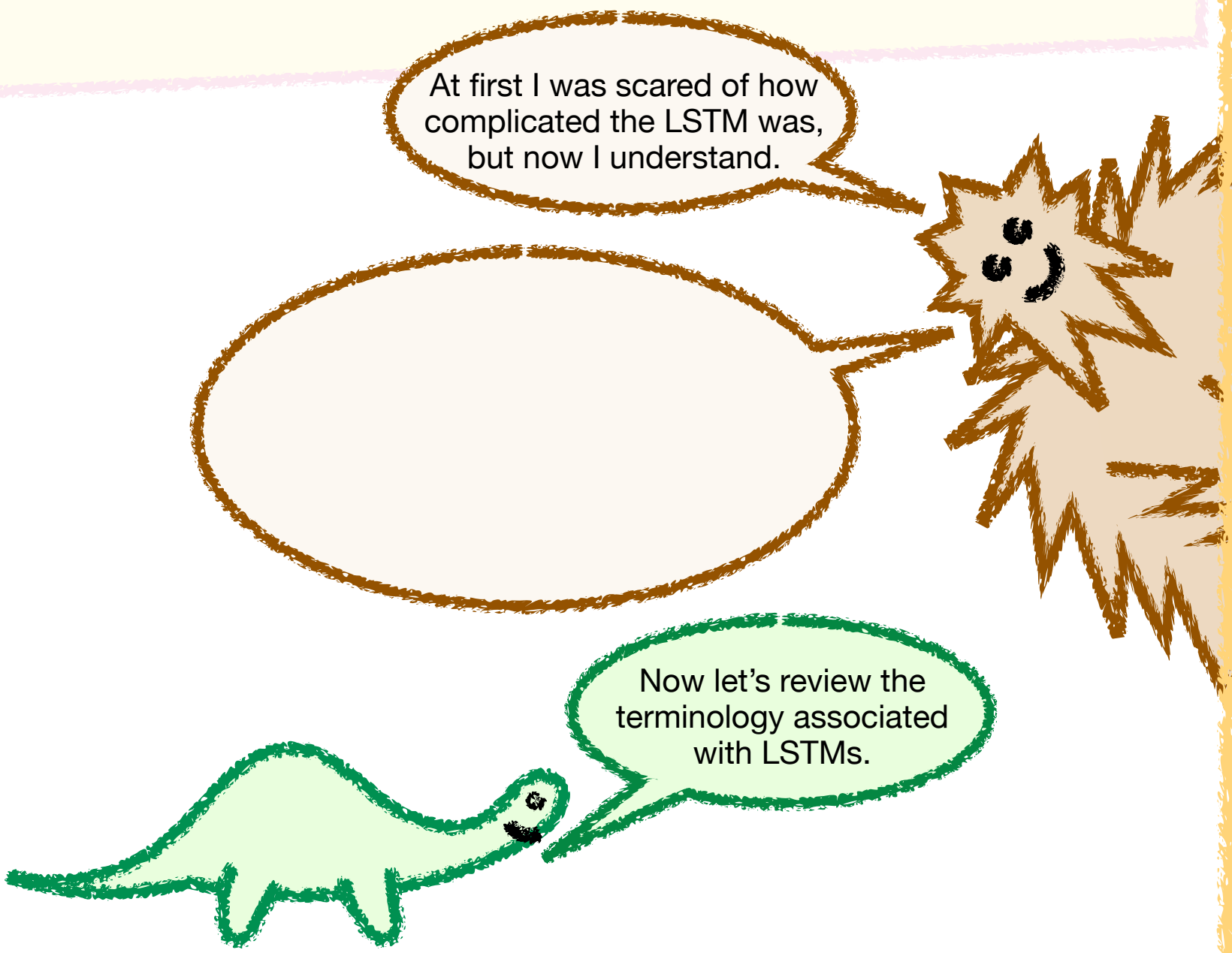
15

In summary, by using separate paths for **Long-Term Memories**…

…and **Short-Term Memories**…

…Long Short-Term Memory networks (LSTMs) reduced the effects of **The Vanishing/ Exploding Gradient Problems**, and that means we can unroll them more times to accommodate longer sequences of input data than a *basic* Recurrent Neural Network (RNN).

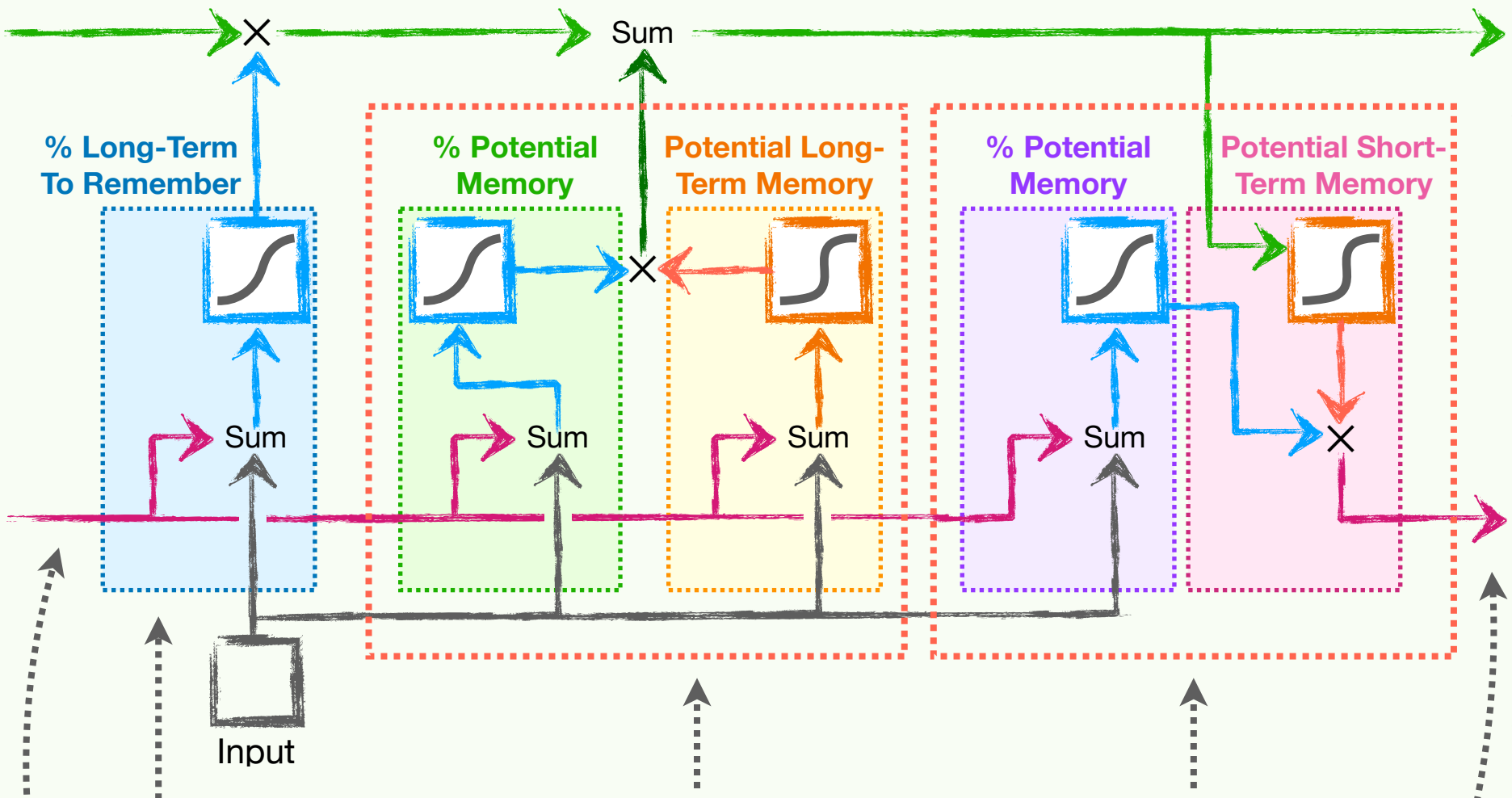




Terminology Alert!!! LSTM Parts

1 Because there is a lot of terminology associated with LSTMs, here's a diagram that puts it all in one place.

The **Long-Term Memory** is called the **Cell State**.



The first part, called the **Forget Gate**, calculates the percent of the **Long-Term Memory** to remember.

The second part, called the **Input Gate**, calculates a new potential **Long-Term Memory** and determines what percentage should be remembered.

The third part, called the **Output Gate**, calculates a new potential **Short-Term Memory** and determines what percentage should be remembered.

The **Short-Term Memory**, which is what we use to make predictions, is called the **Hidden State**.