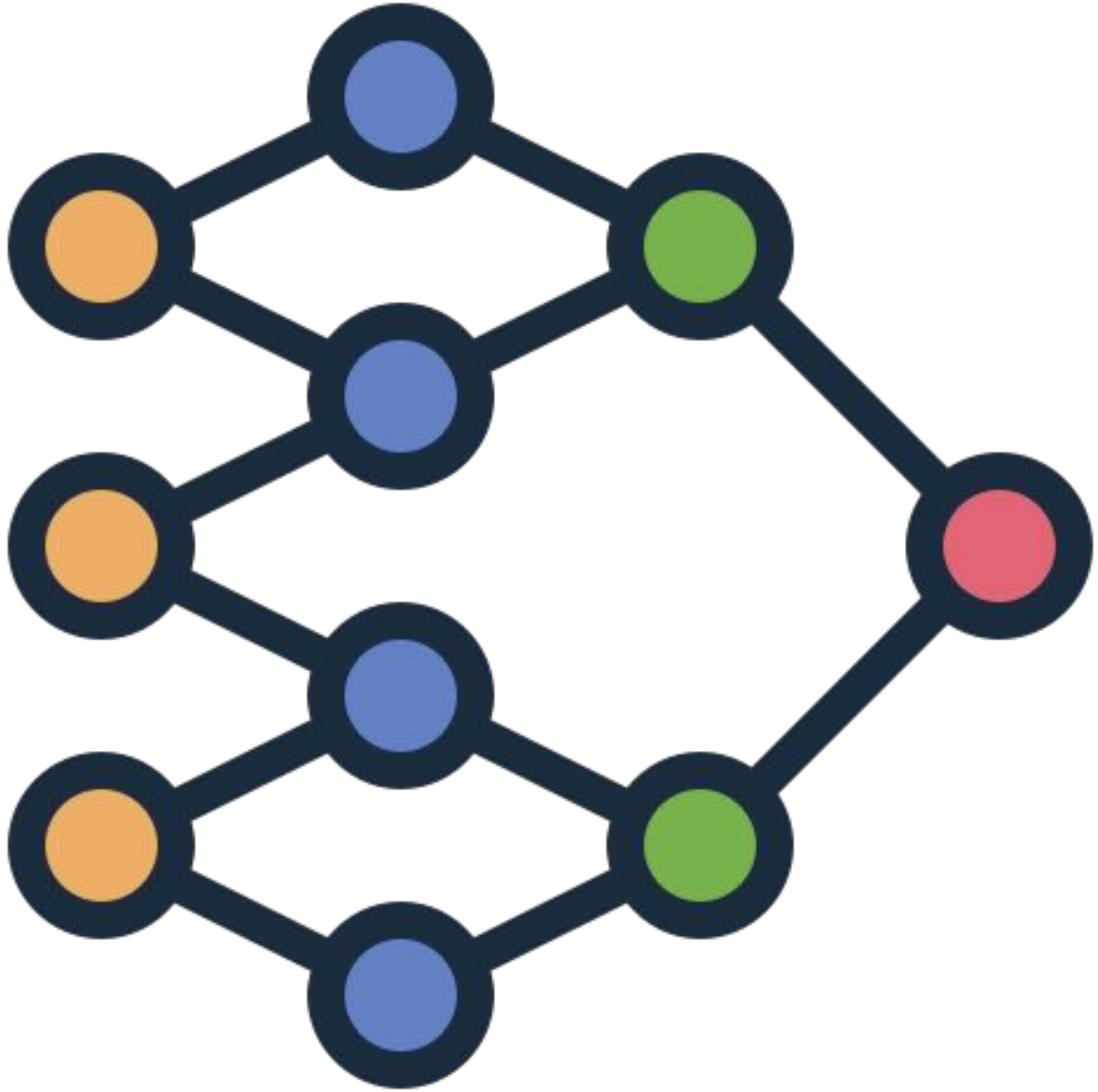
Introduction to Neural Networks



By Imène ALLOUCHE

TABLE OF CONTENTS

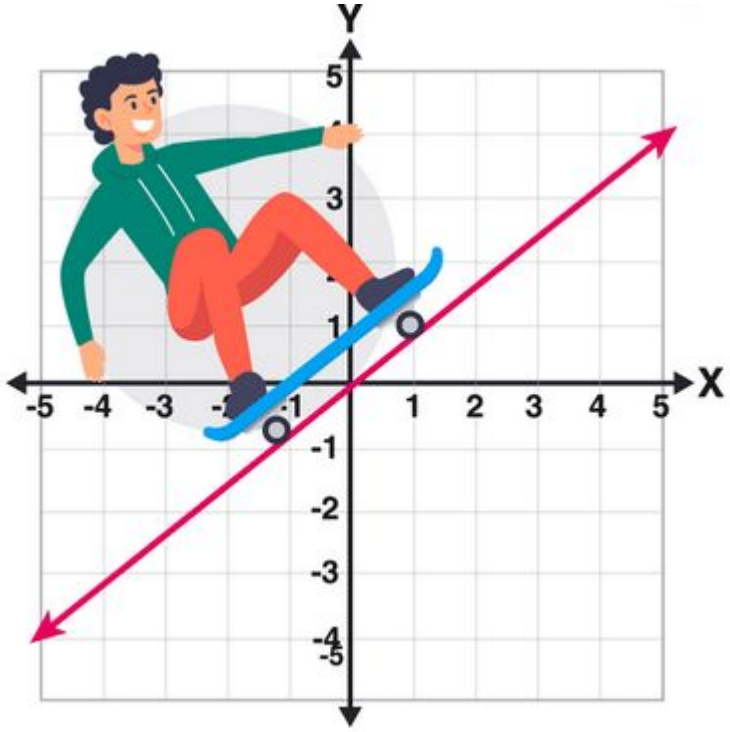
01

Why Neural Network?

01

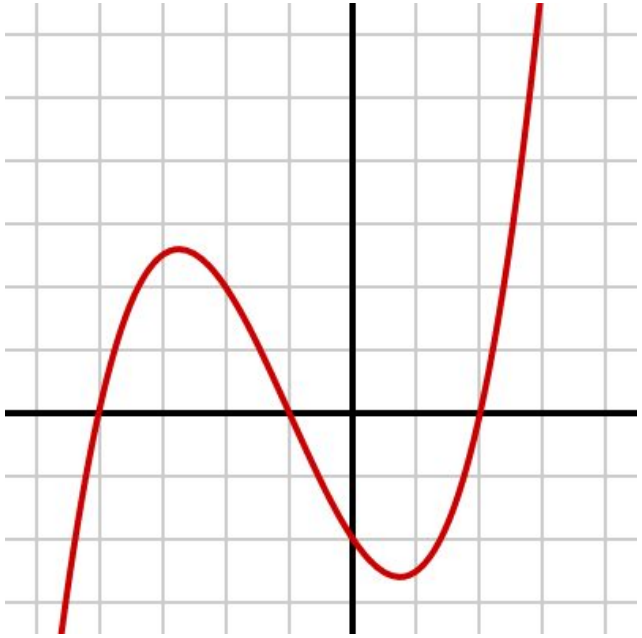
Why Neural Network?

Linear Relations



y= ax+b

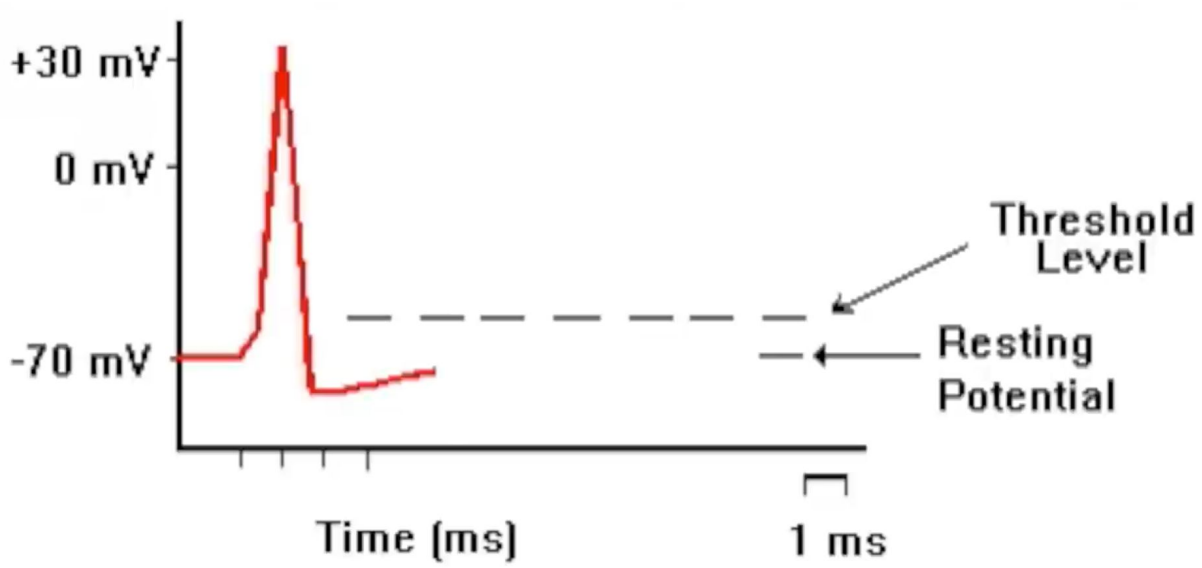
Non Linear Relations

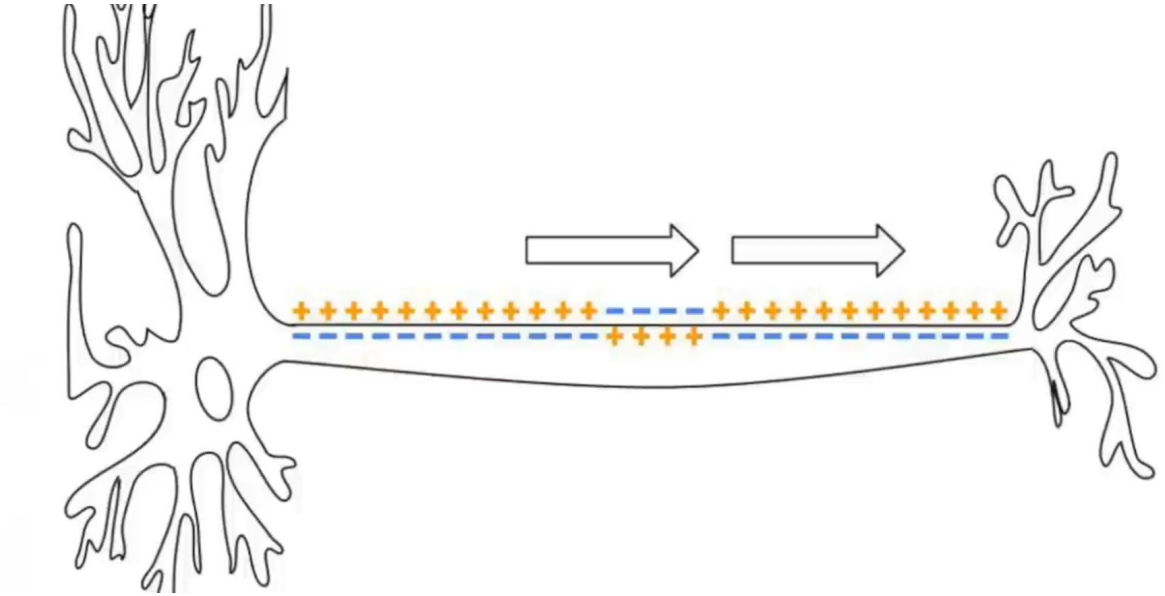


y= ?

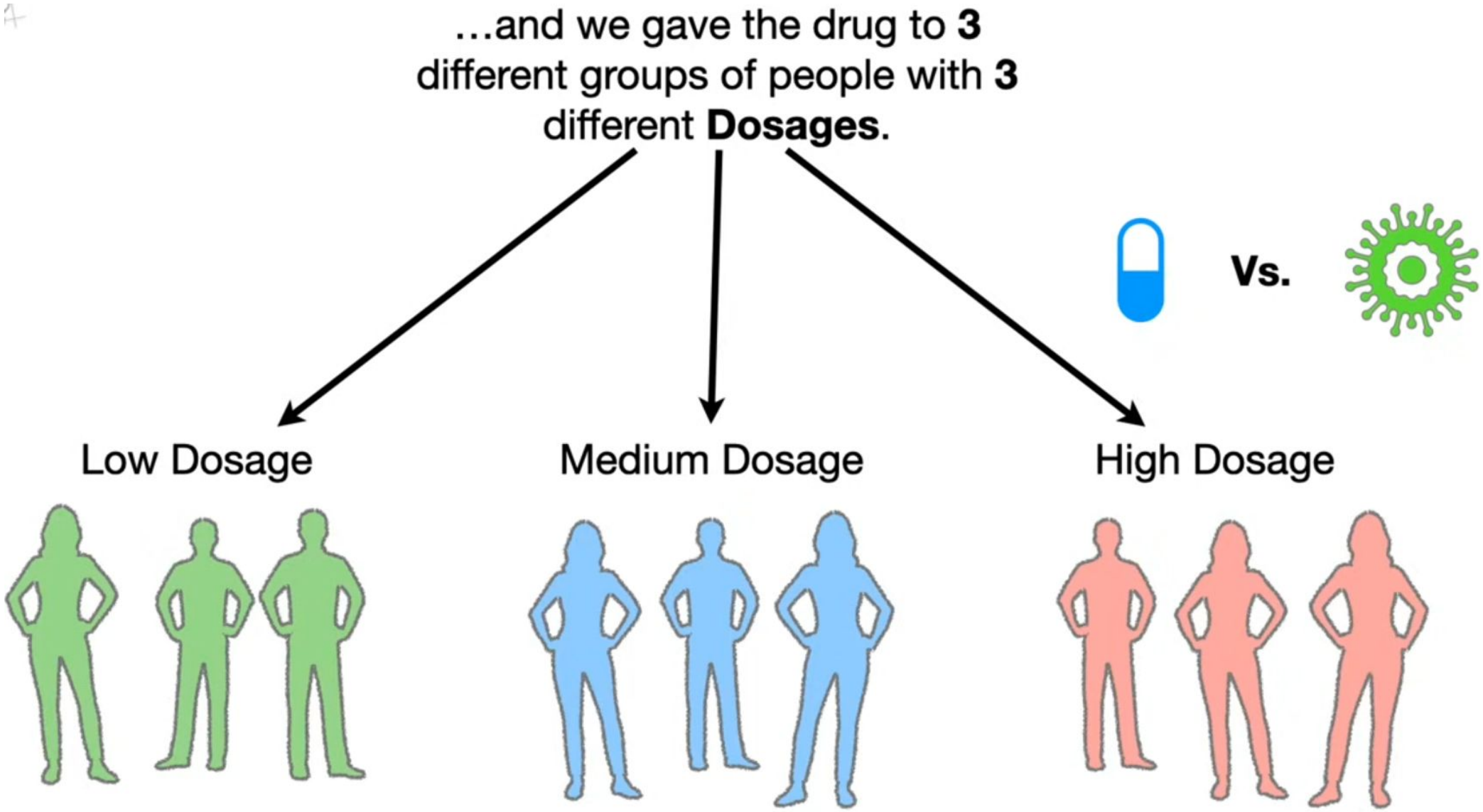
01

Why Neural Network?

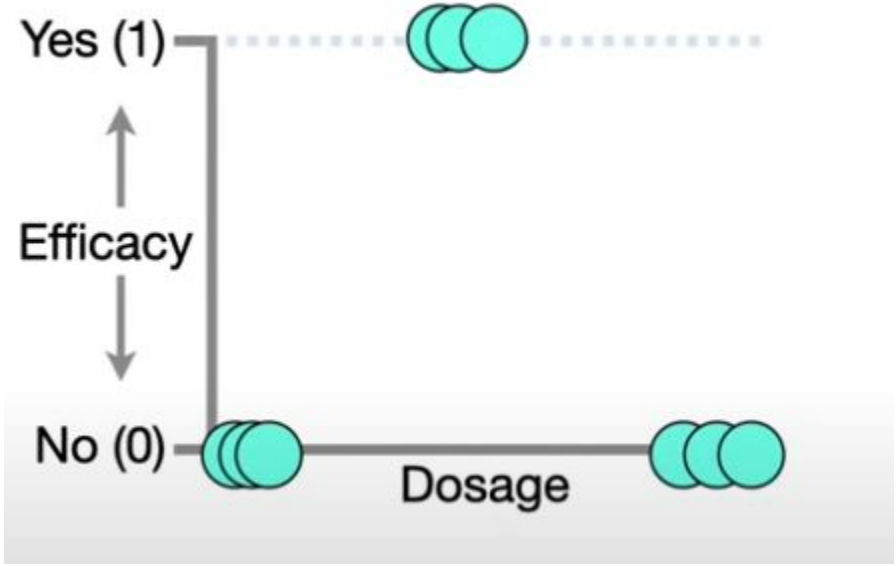


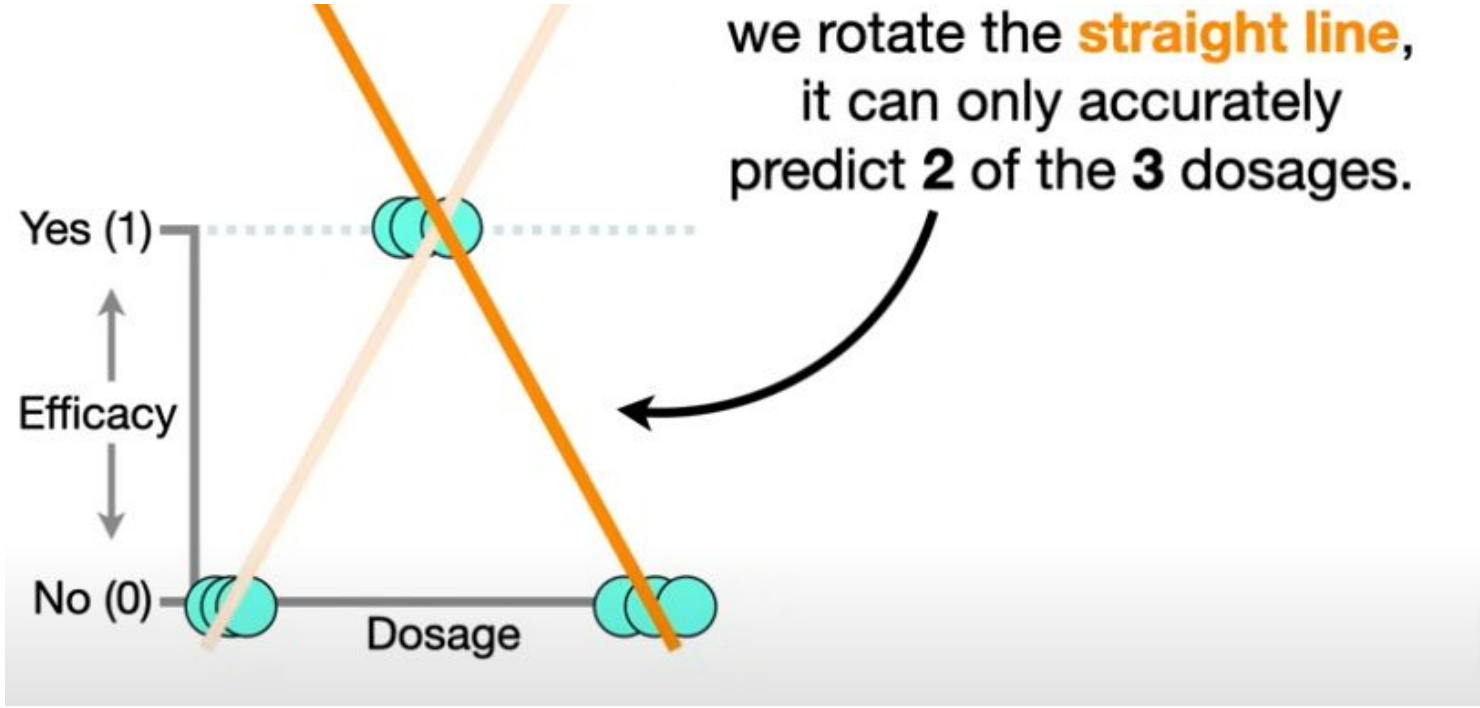


How Neural Network?

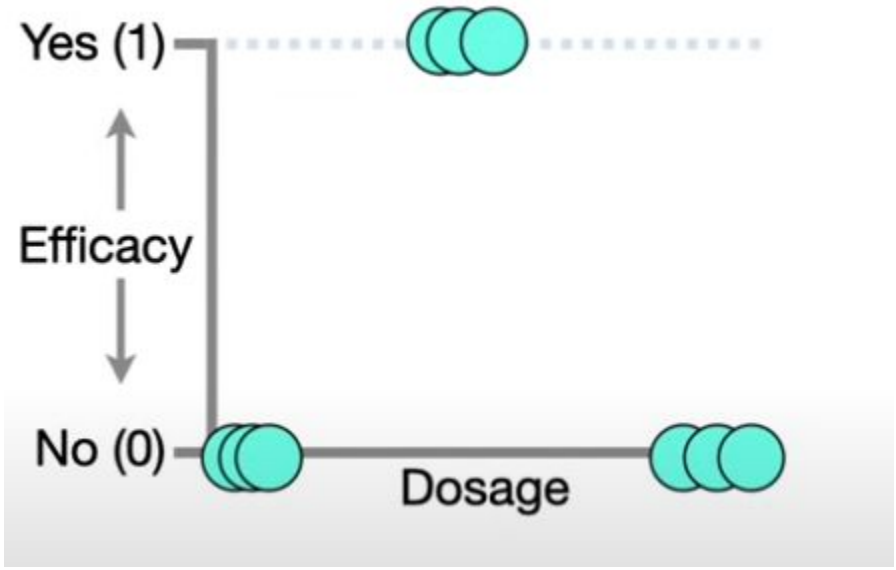


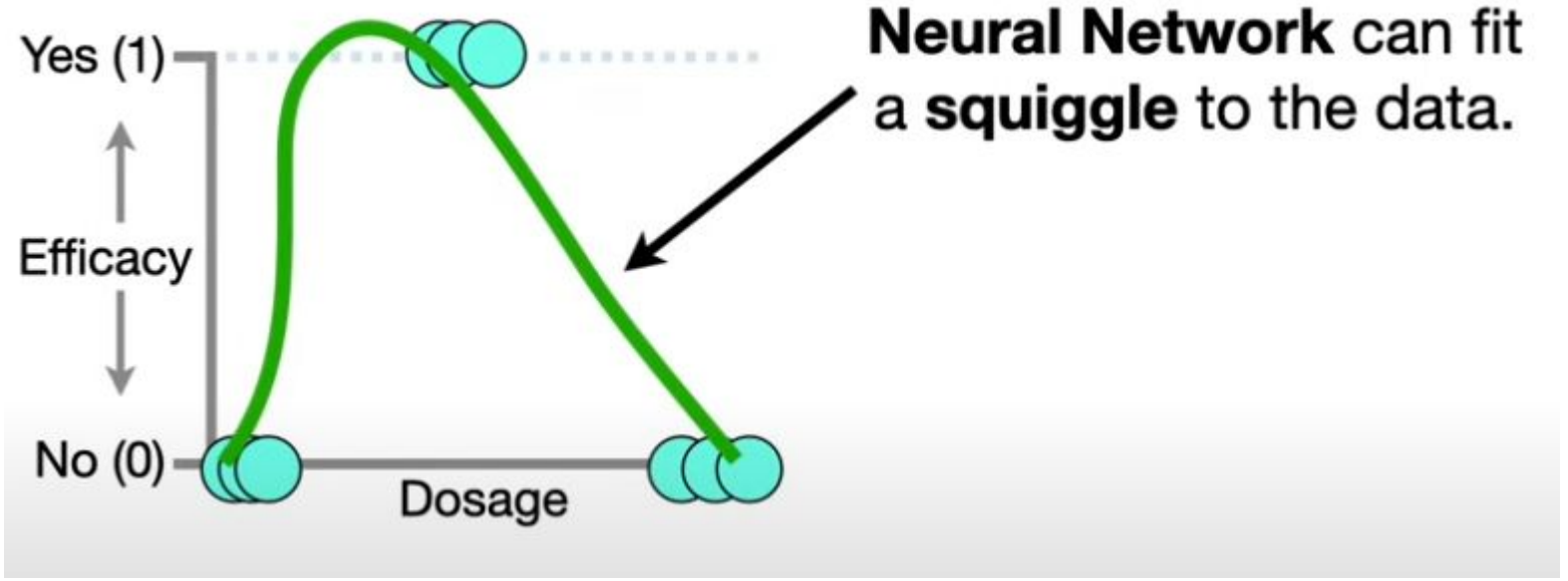
How Neural Network?



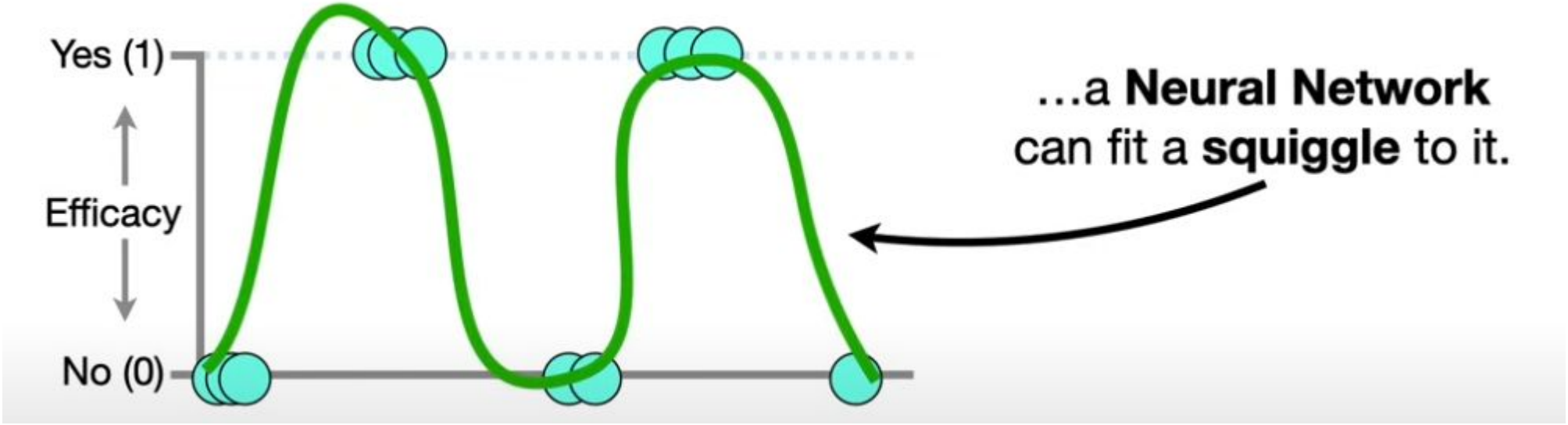


How Neural Network?

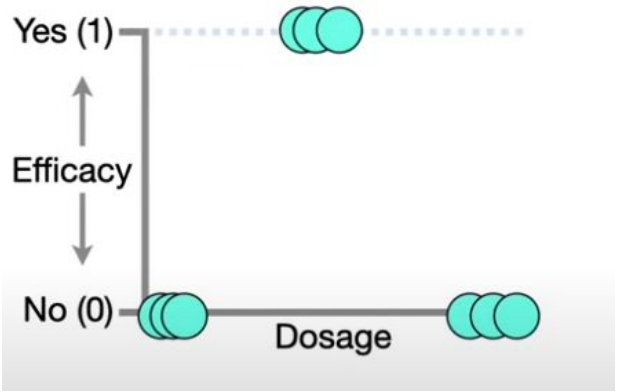


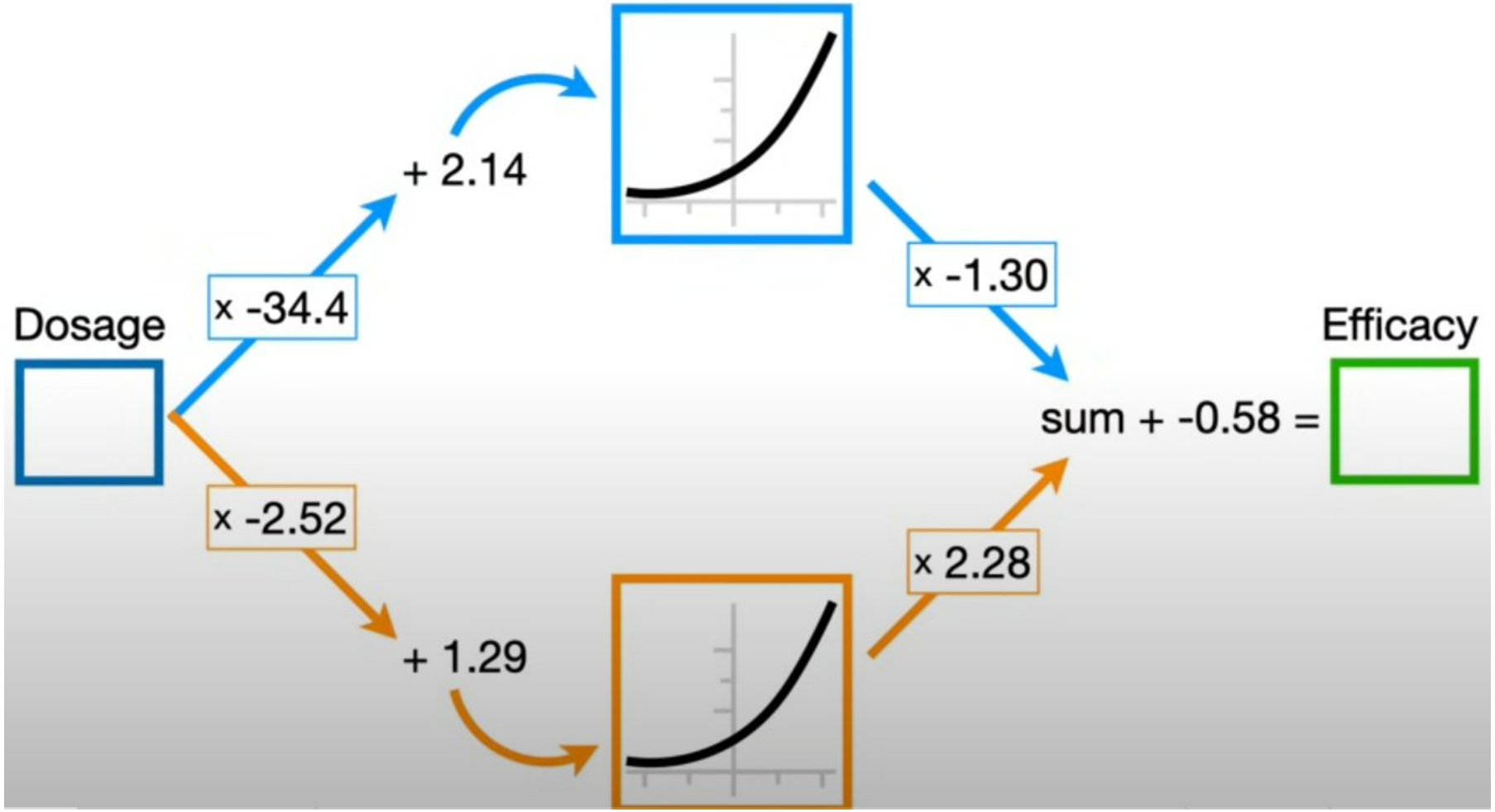


How Neural Network?



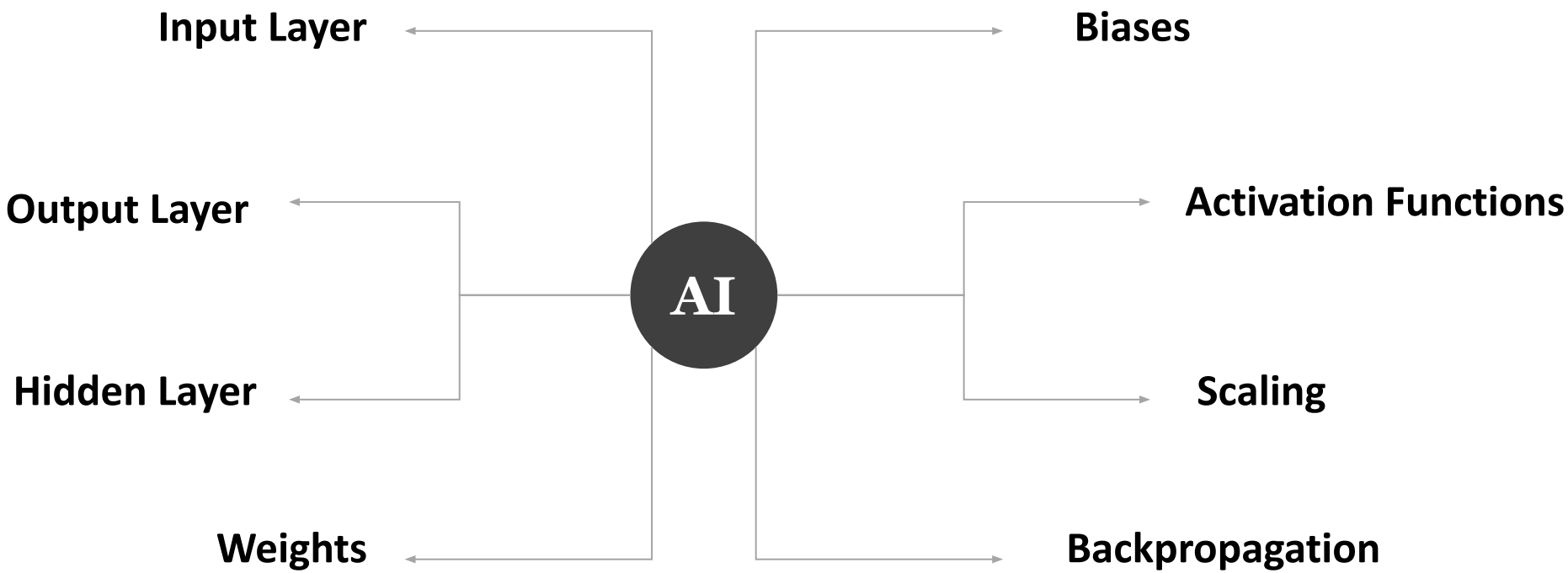
How Neural Network?





01

Terminology



The Math Behind The Hidden Layer

01 Input (Dosages in out example)

02 X1 = w1 * Input + b1

02 X2 = w2 * Input + b2

03 Y1 = f1(X1)

03 Y2 = f2(X2)

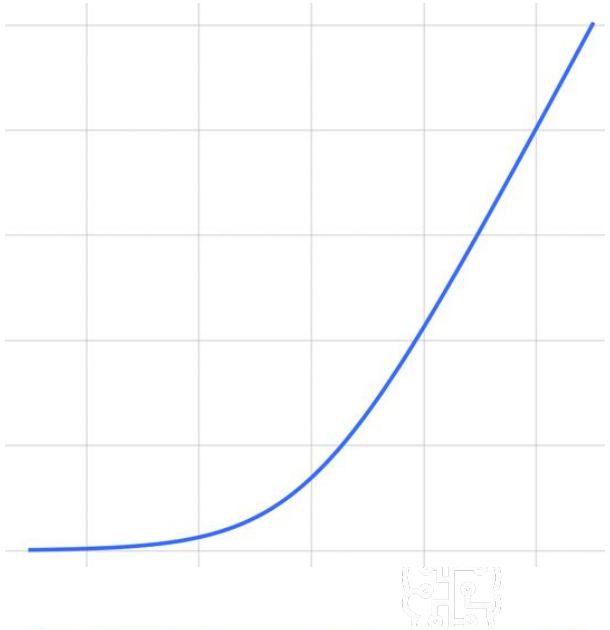
04 Y1 = Y1 * w3

04 Y2 = Y2 * w4

05 Y = Y1 + Y2 + b3

Artificial intelligence development phase

Softplus

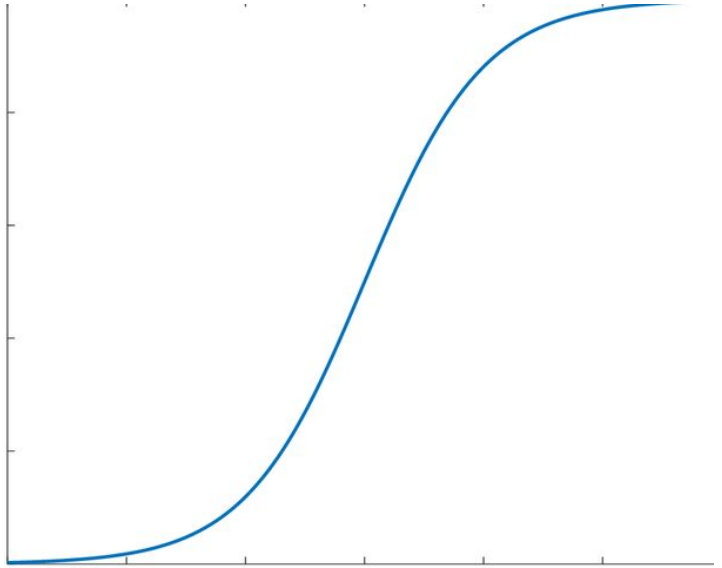


$$f(x) = \log(1 + e^x)$$

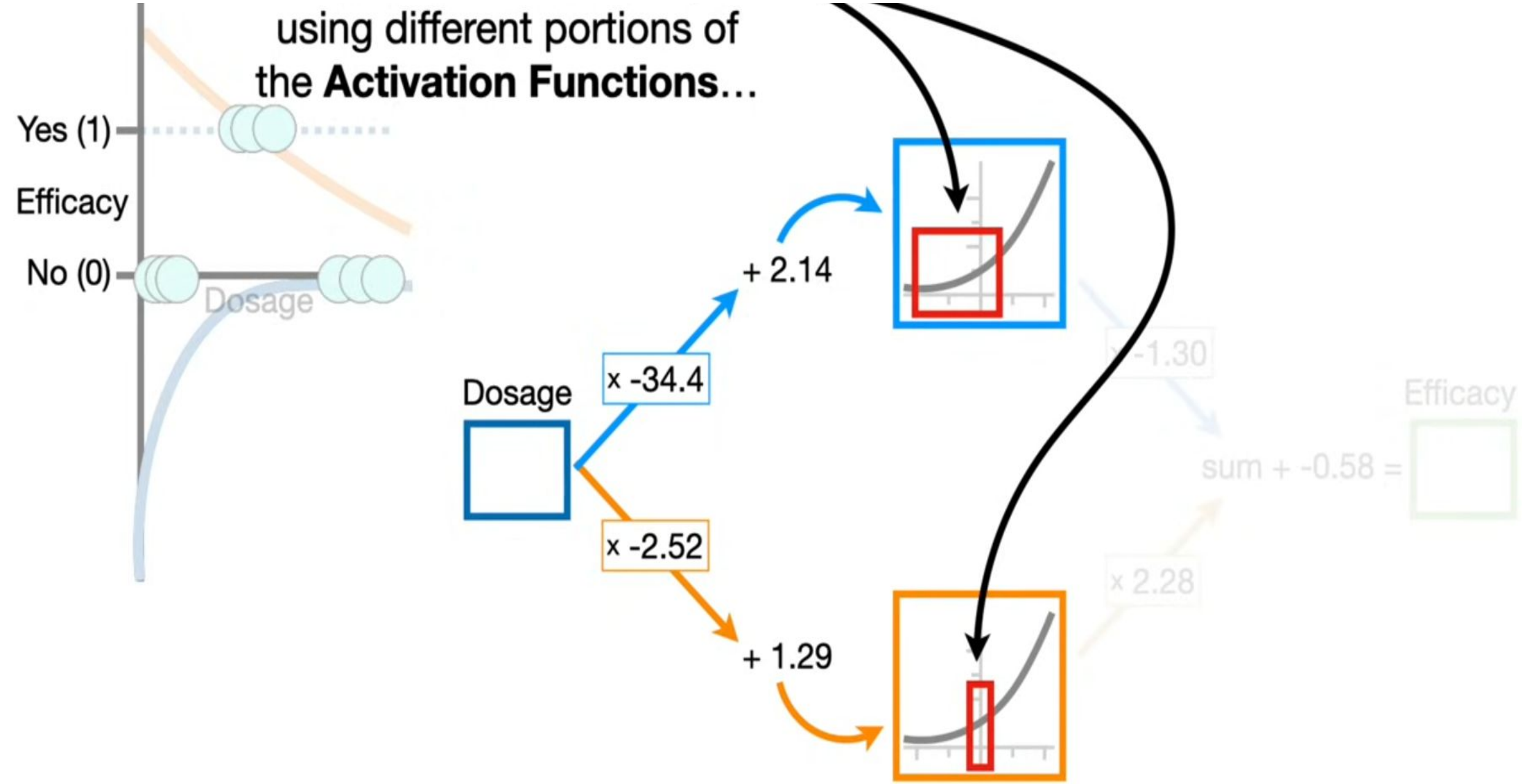
ReLU

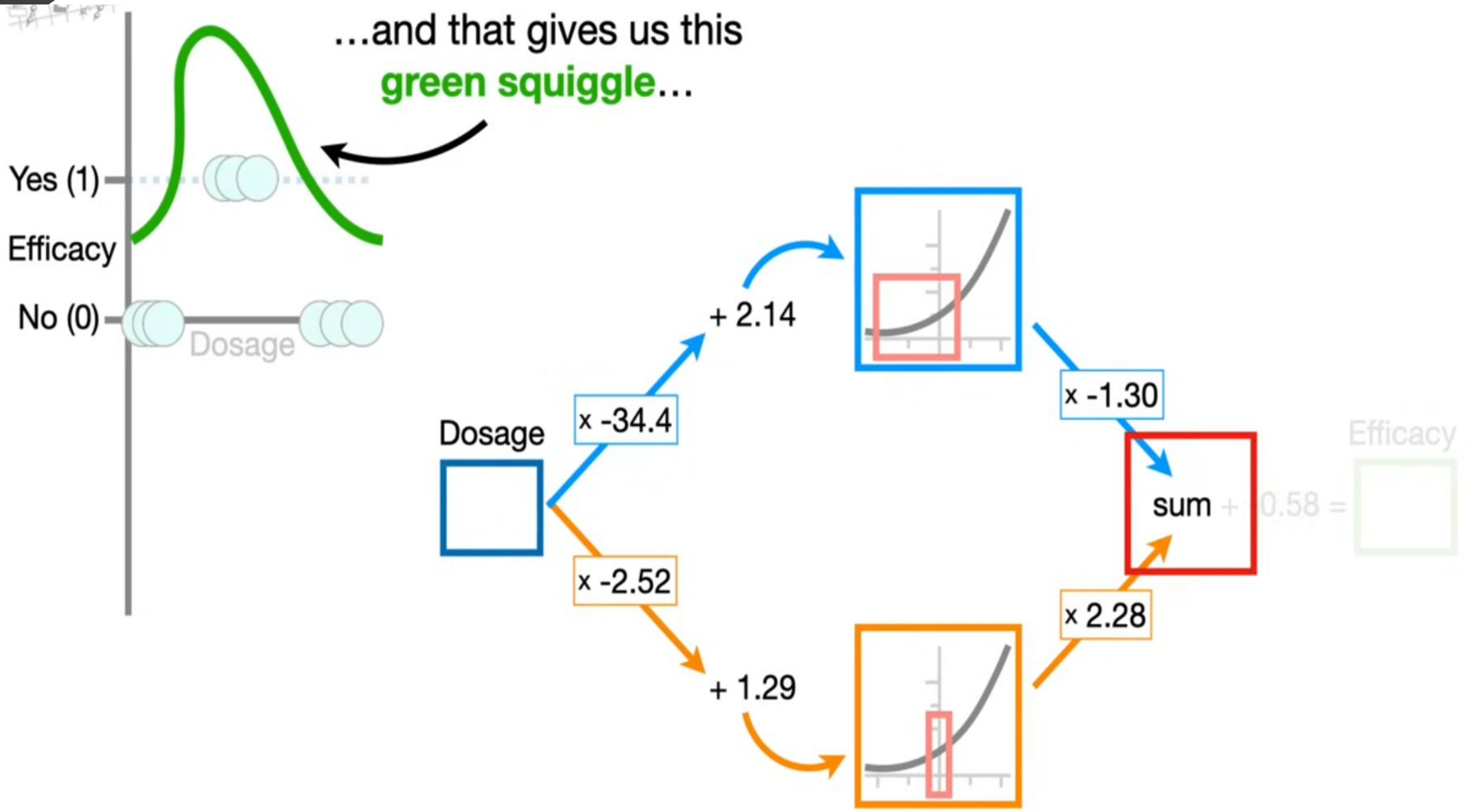
$$f(x) = \max(0, x)$$

Sigmoid

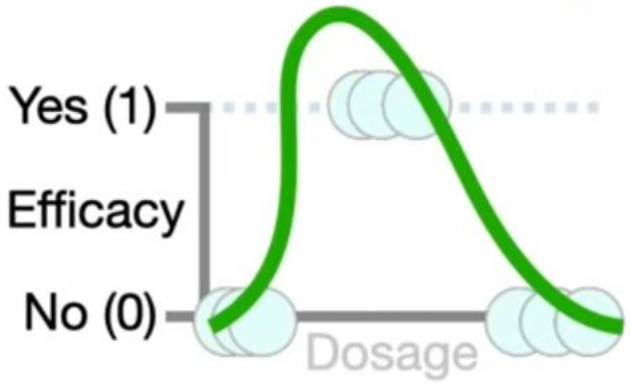


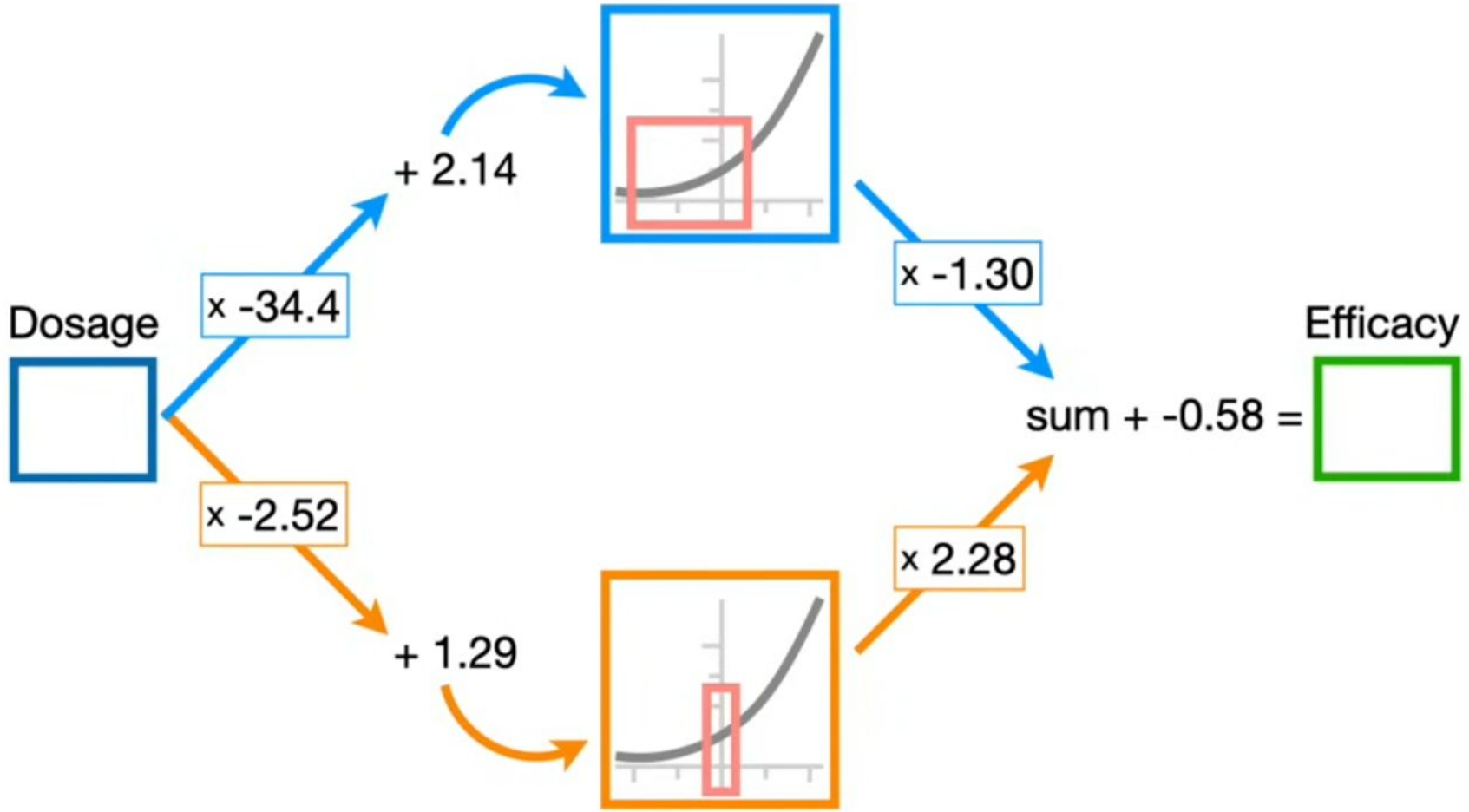
$$f(x) = \frac{e^x}{e^x + 1}$$


using different portions of
the **Activation Functions**…
Yes (1)
Efficacy
No (0)
Dosage
Dosage
x -34.4
x -2.52
+ 2.14
+ 1.29
x -1.30
x 2.28
sum + -0.58 =
Efficacy


...and that gives us this
green squiggle...
Yes (1)
Efficacy
No (0)
Dosage
Dosage
x -34.4
x -2.52
+ 2.14
+ 1.29
x -1.30
x 2.28
sum
Efficacy

squiggle that fits the data.





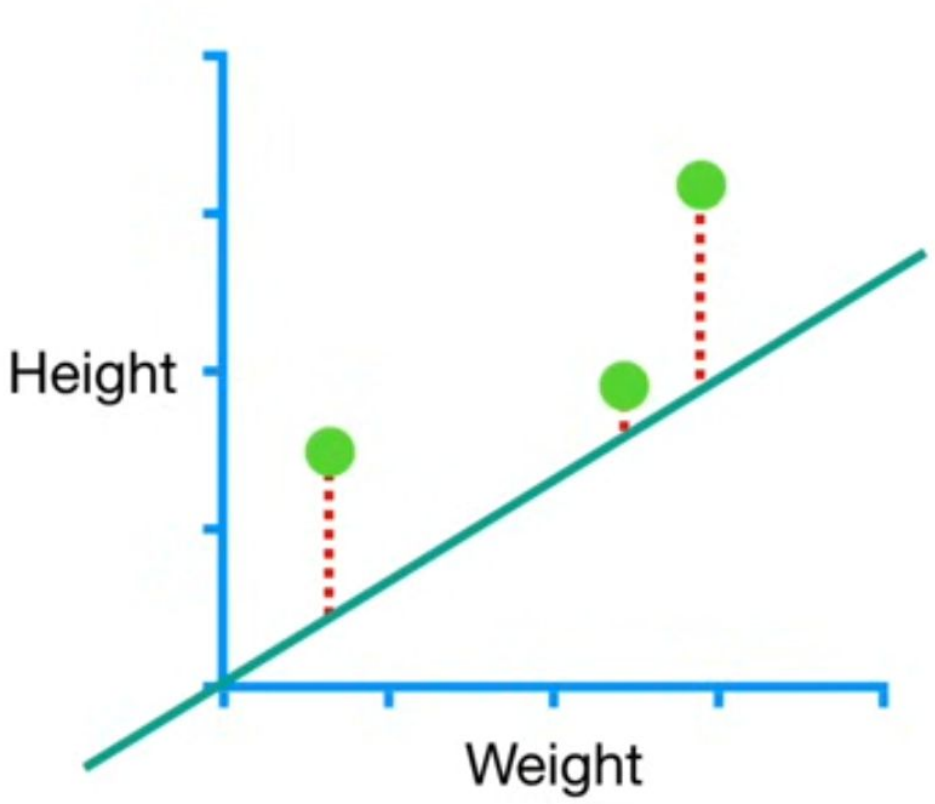
02

Gradient Descent

Gradient Descent

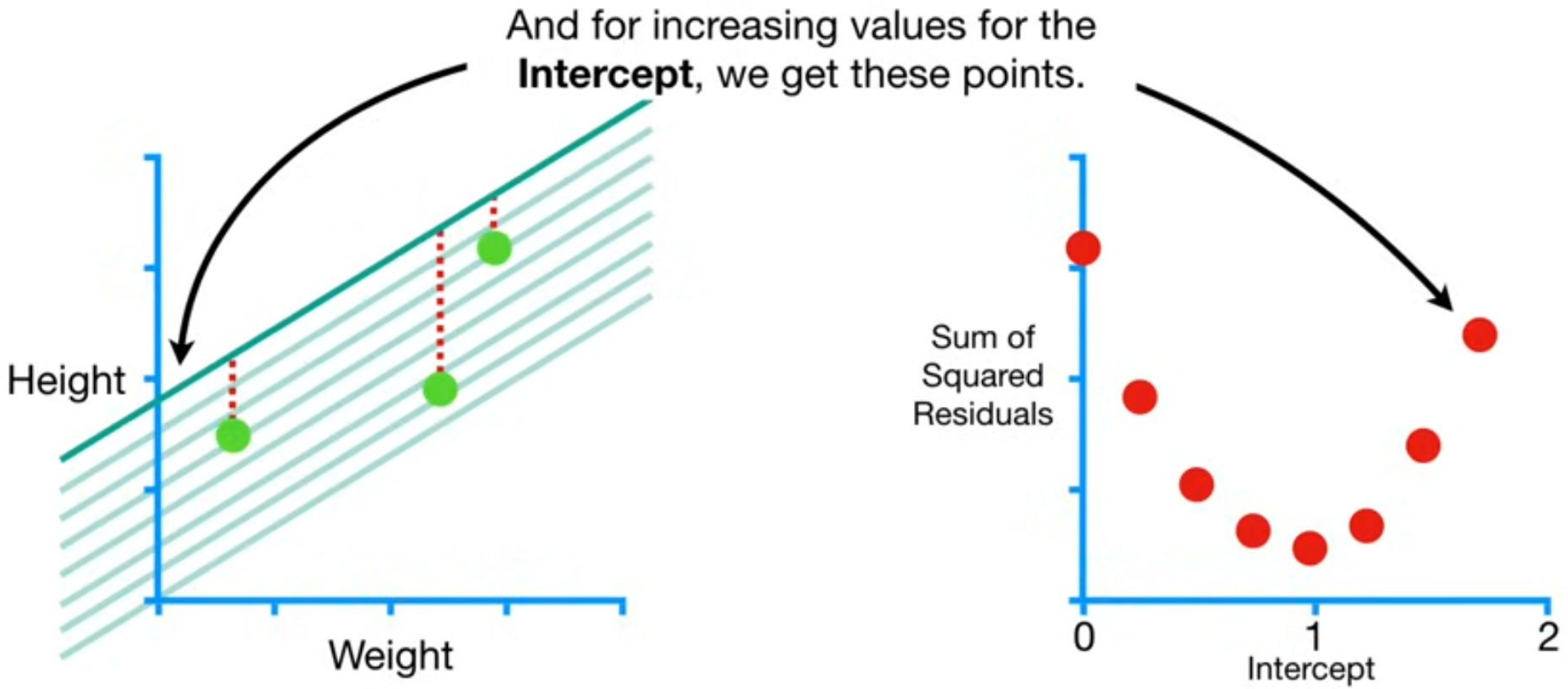
Sum of squared residuals = $1.1^2 + 0.4^2 + 1.3^2 = 3.1$

In the end, **3.1** is the Sum of the Squared Residuals.



02

Gradient Descent

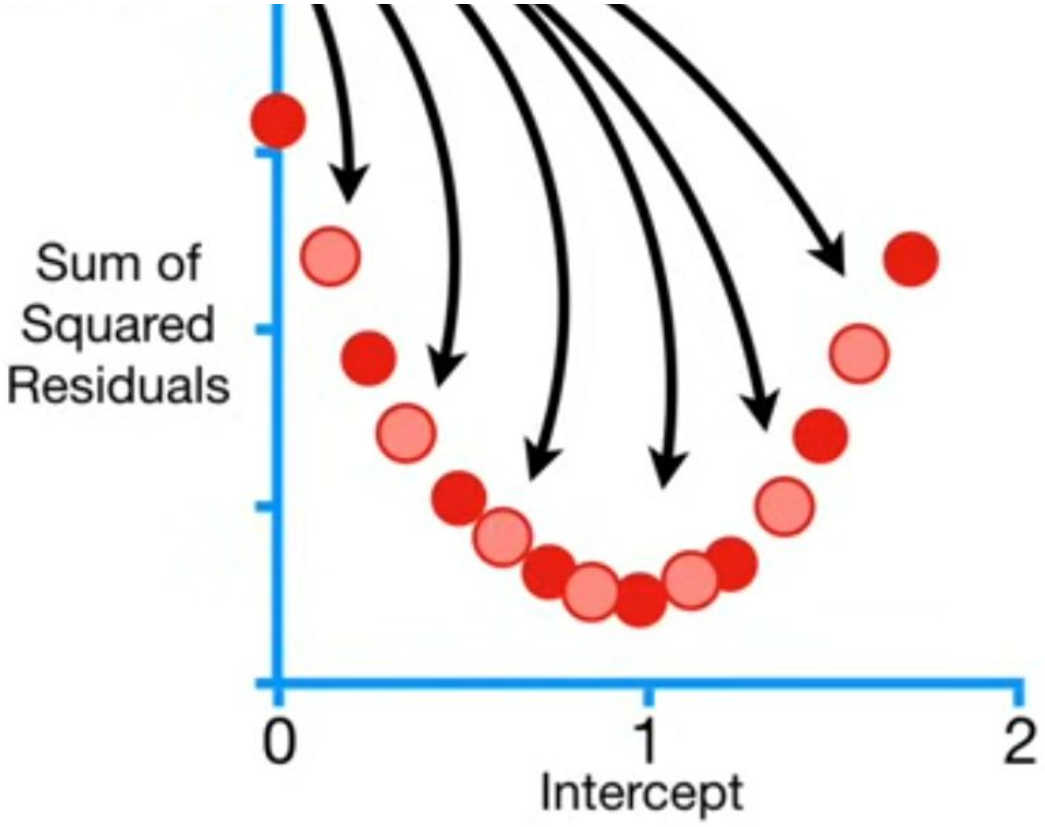


02

Gradient Descent

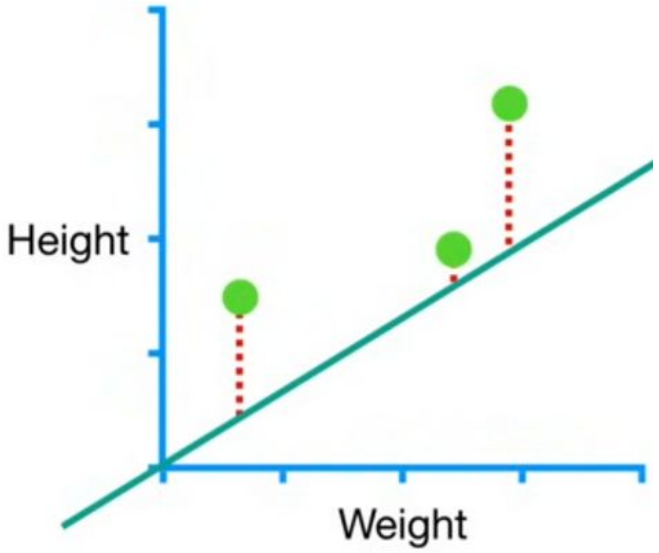
A slow and painful method for finding minimal sum of the squared Residuals is to plug a bunch of values of the Intercept

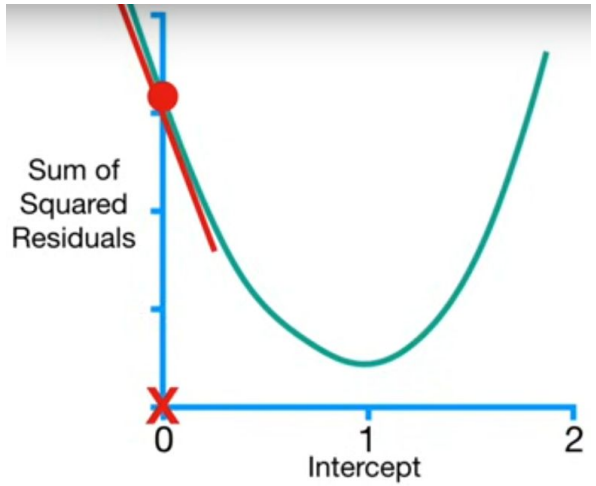
Y = Slope * x + Intercept



02

Gradient Descent





$$\text{Sum of squared residuals} = (\mathbf{1.4} - (\text{intercept} + 0.64 \times \mathbf{0.5}))^2 + (\mathbf{1.9} - (\text{intercept} + 0.64 \times \mathbf{2.3}))^2 + (\mathbf{3.2} - (\text{intercept} + 0.64 \times \mathbf{2.9}))^2$$

$$\frac{d}{d\,\text{intercept}} \text{Sum of squared residuals} = -2(\mathbf{1.4} - (0 + 0.64 \times \mathbf{0.5}) + -2(\mathbf{1.9} - (0 + 0.64 \times \mathbf{2.3})) + -2(\mathbf{3.2} - (0 + 0.64 \times \mathbf{2.9}))$$

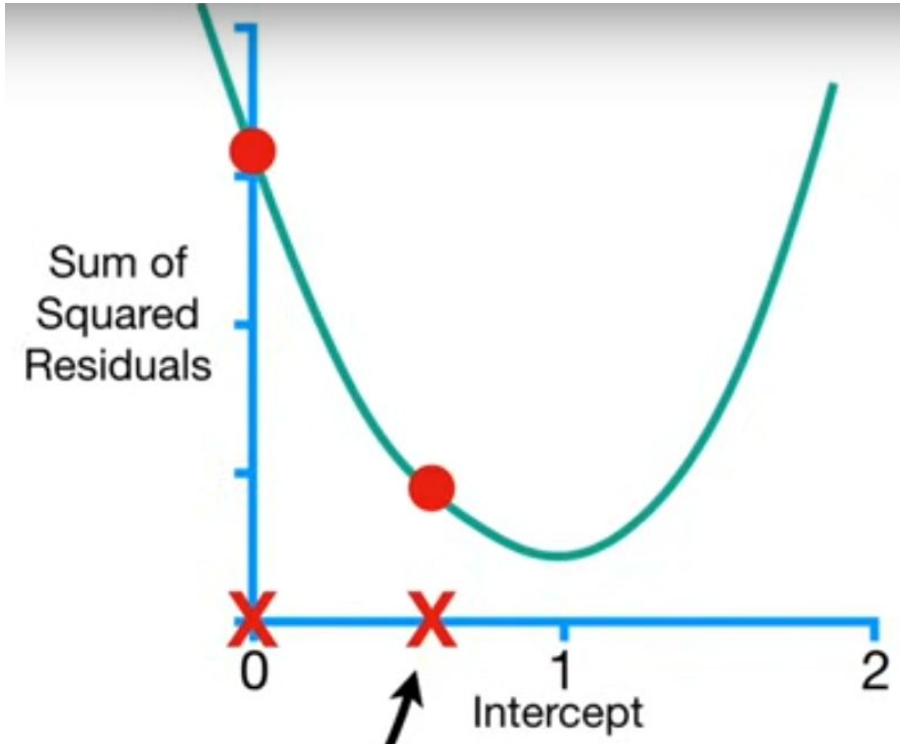
$$= -5.7$$

Step Size = -5.7 × 0.1 = -0.57

When the **Intercept** = **0**, the **Step Size** = **-0.57**.

Step Size = -5.7 * Learning Rate

Gradient Descent



New Intercept = Old Intercept - Step Size

New Intercept = 0 - (-0.57) = **0.57**

…and the the **New Intercept** = **0.57**.

When do we stop?

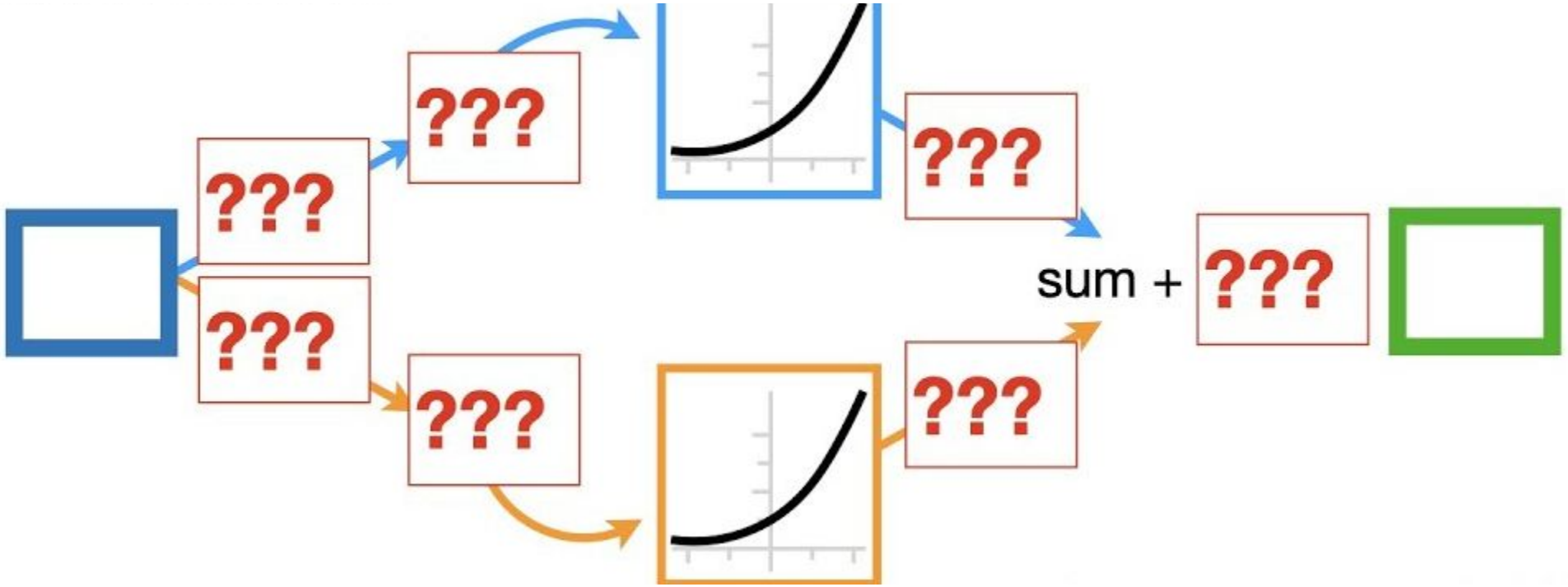
- **when reaching an SSR <= 0.001**
- **Could be changed by the developer**

- **When reaching a maximum number of steps is reached**
- **To be fixed by the developer**

03

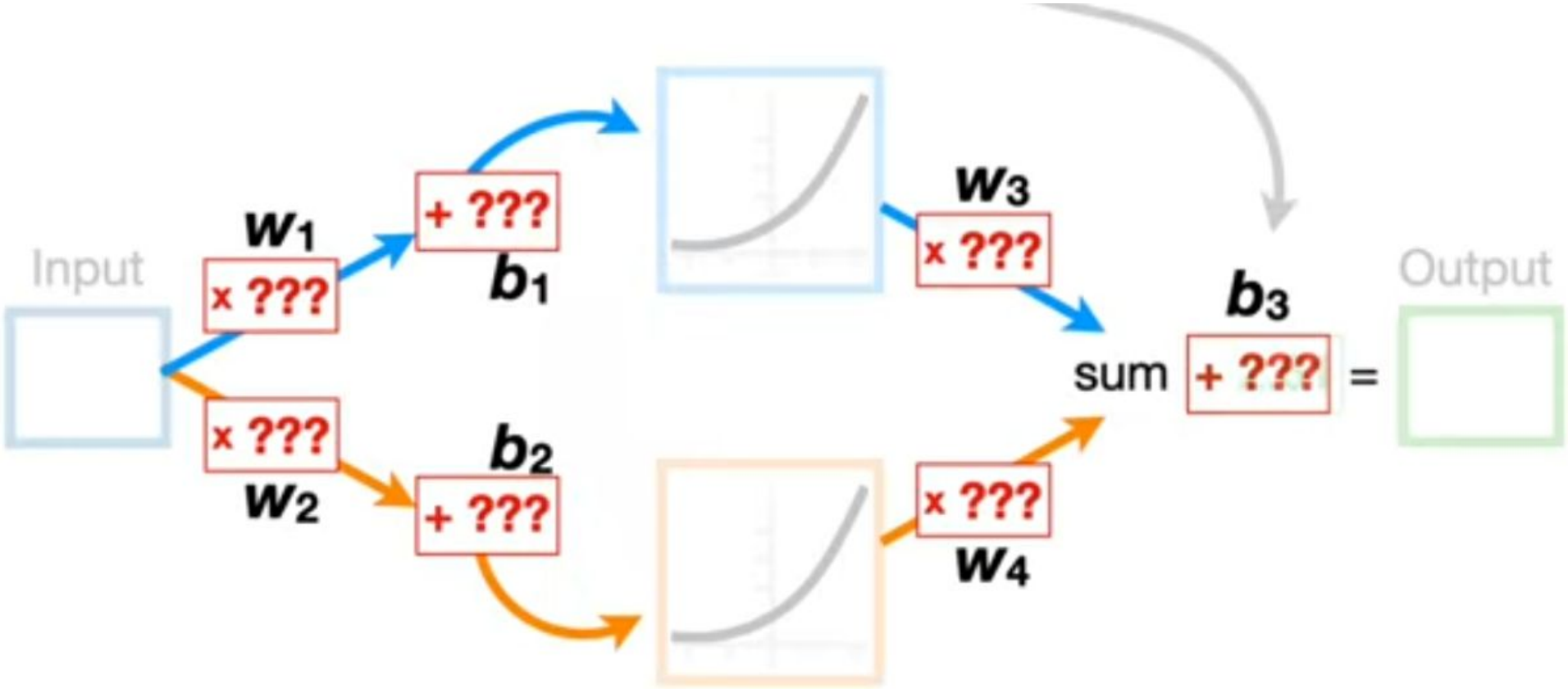
Backpropagation

How to find the best Weights and biases?



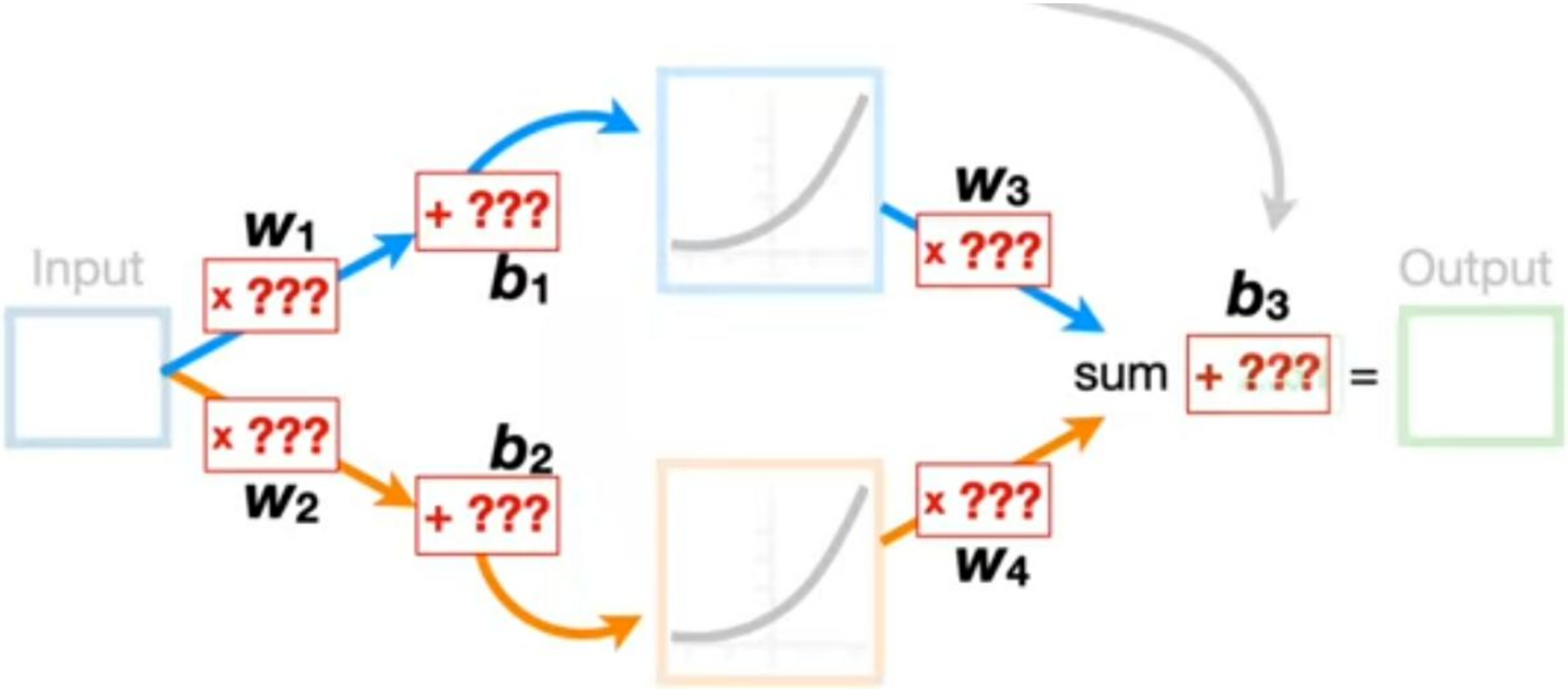
$$\frac{d\,SSR}{d\,w_1} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,y_1} \times \frac{d\,y_1}{d\,x_1} \times \frac{d\,x_1}{d\,w_1}$$

$$\frac{d\,SSR}{d\,w_1} = \sum_{i=1}^{n=3} -2 \times (Observed_i - Predicted_i) \times w_3 \times \frac{e^x}{1 + e^x} \times Input_i$$



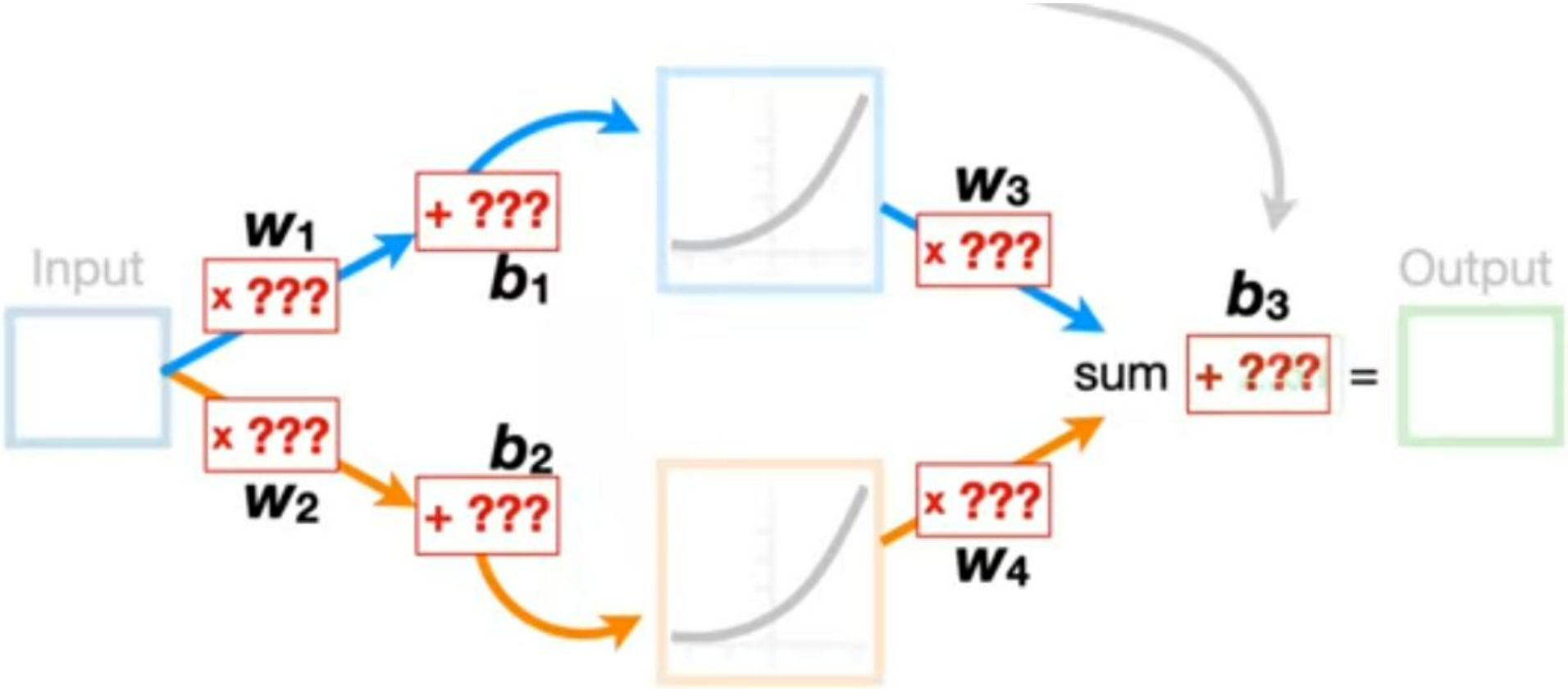
$$\frac{d\,SSR}{d\,b_1} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,y_1} \times \frac{d\,y_1}{d\,x_1} \times \frac{d\,x_1}{d\,b_1}$$

$$\frac{d\,SSR}{d\,b_1} = \sum_{i=1}^{n=3} -2 \times (Observed_i - Predicted_i) \times w_3 \times \frac{e^x}{1 + e^x} \times 1$$

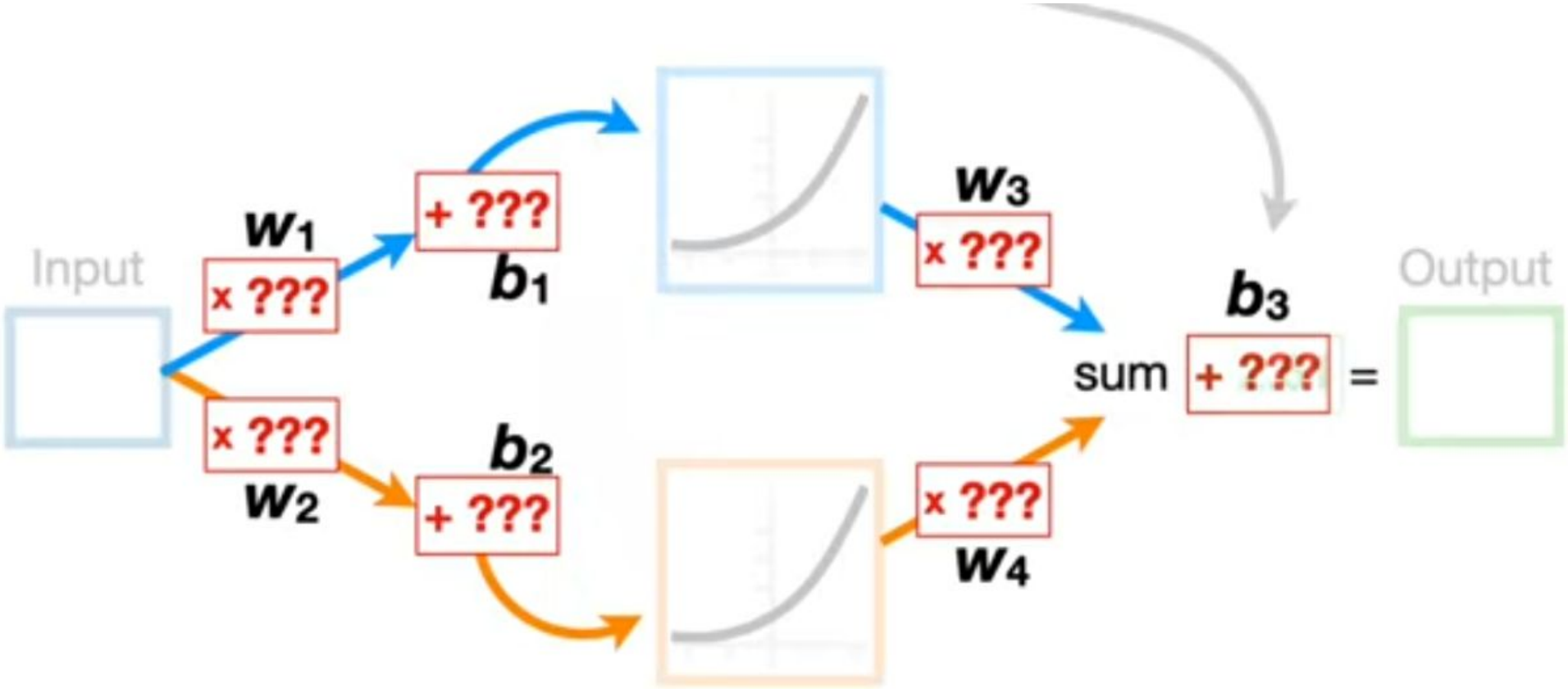


$$\frac{d\,SSR}{d\,w_2} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,y_2} \times \frac{d\,y_2}{d\,x_2} \times \frac{d\,x_2}{d\,w_2}$$

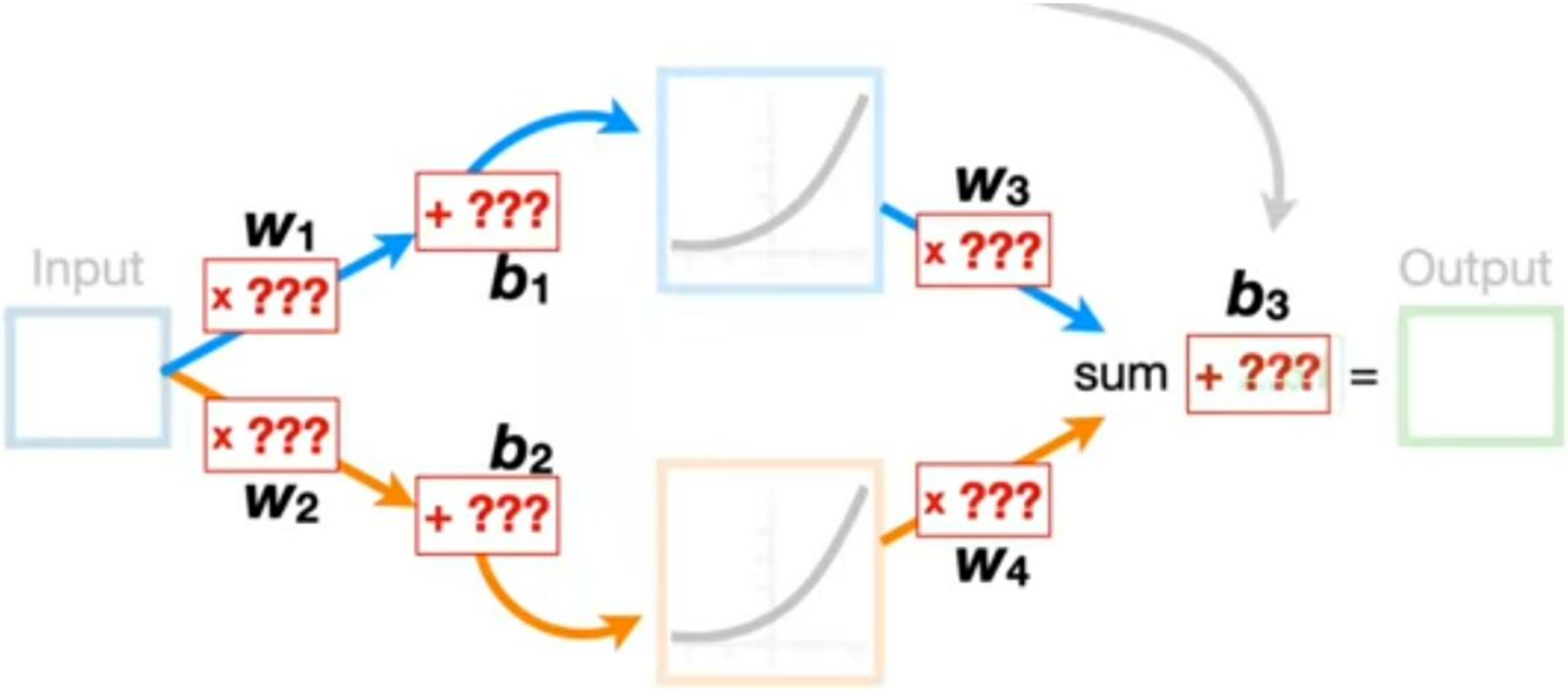
$$\frac{d\,SSR}{d\,w_2} = \sum_{i=1}^{n=3} -2 \times (Observed_i - Predicted_i) \times w_4 \times \frac{e^x}{1+e^x} \times Input_i$$



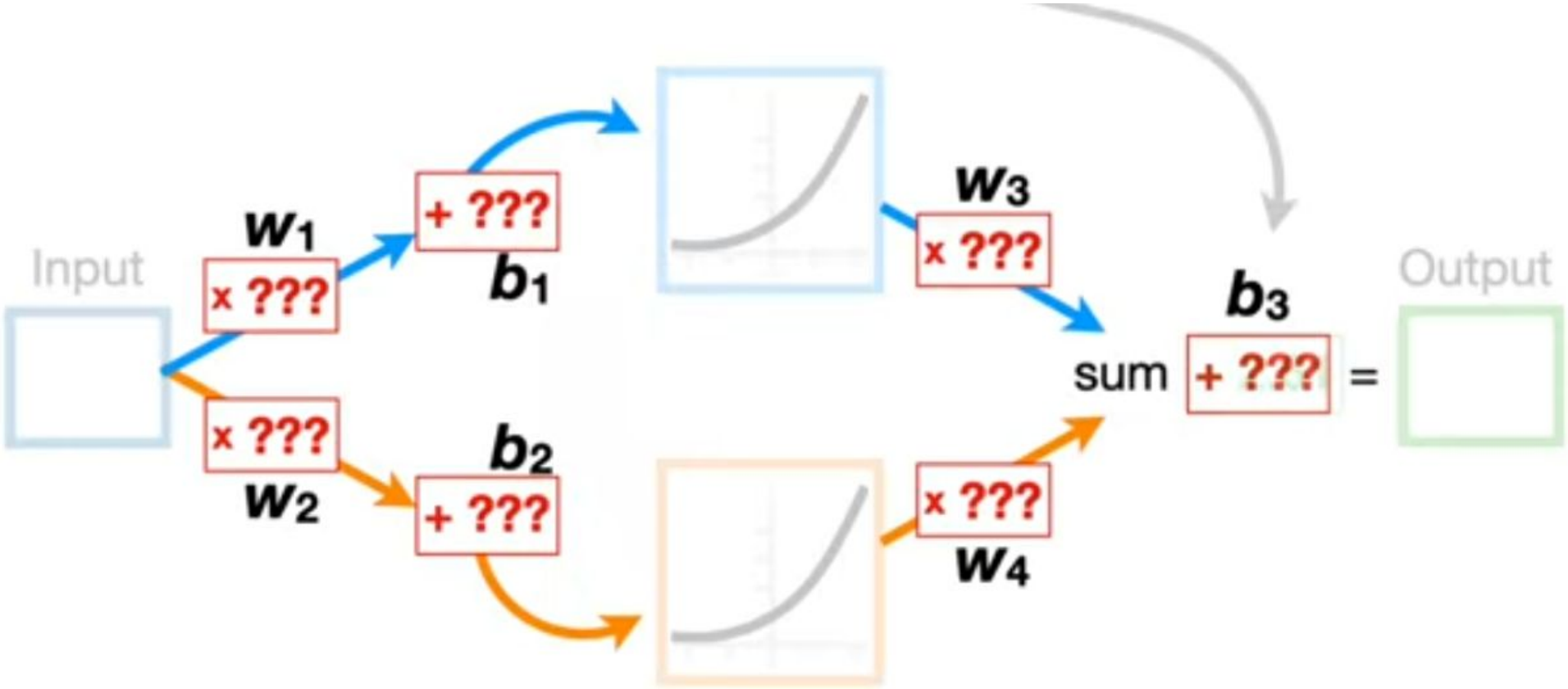
$$\frac{d\,SSR}{d\,b_2} = \sum_{i=1}^{n=3} -2 \times (\text{Observed}_i - \text{Predicted}_i) \times w_4 \times \frac{e^x}{1+e^x} \times 1$$



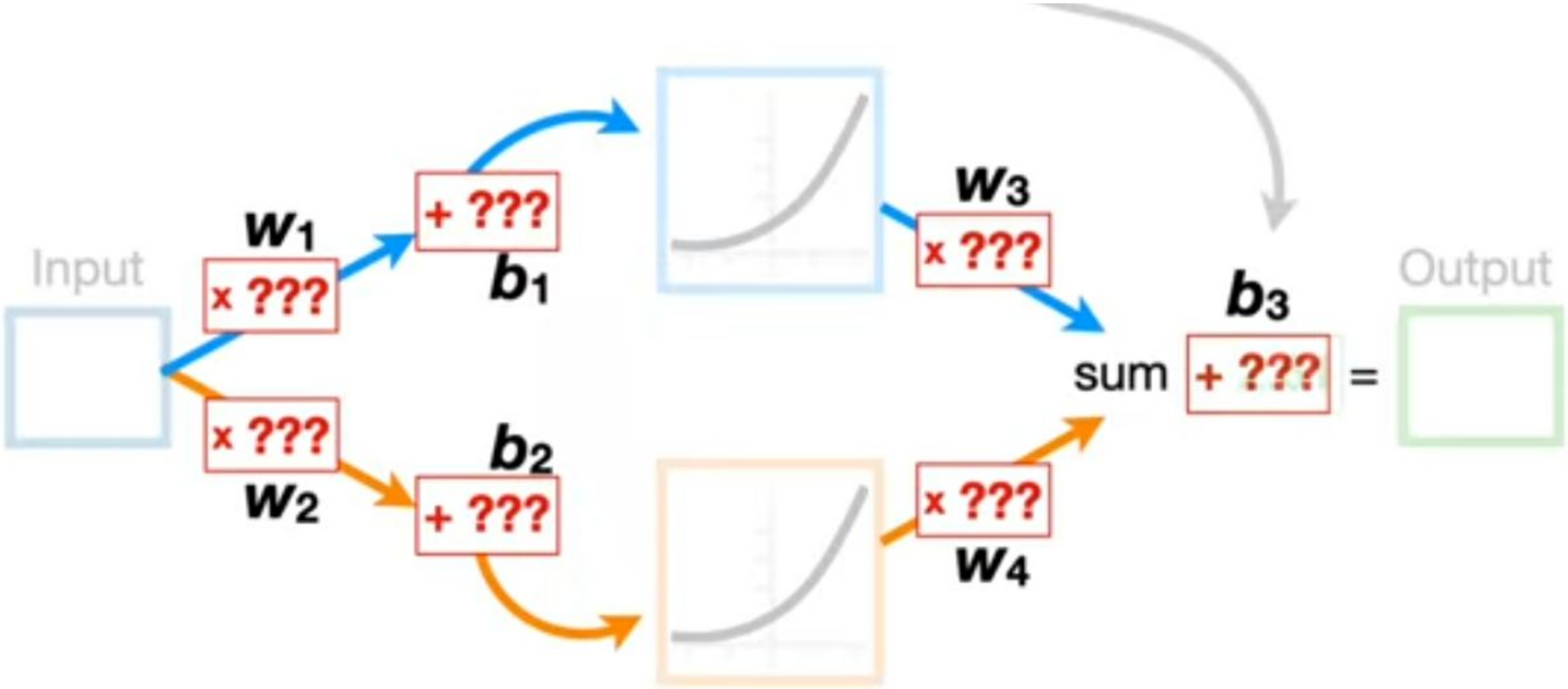
$$\frac{d\,SSR}{d\,w_3} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,w_3} = \sum_{i=1}^{n=3} -2 \times (Observed_i - Predicted_i) \times y_{1,i}$$



$$\frac{d\,SSR}{d\,w_4} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,w_4} = \sum_{i=1}^{n=3} -2 \times (Observed_i - Predicted_i) \times y_{2,i}$$



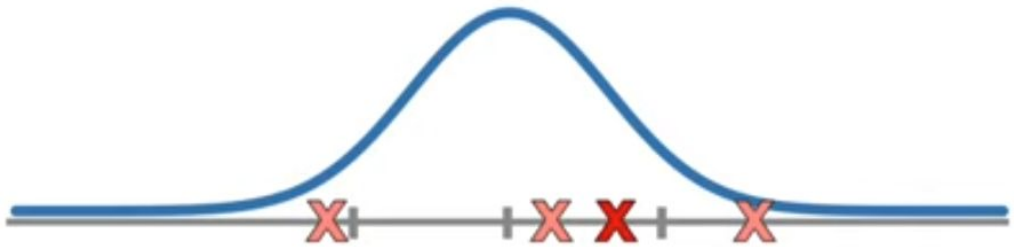
$$\frac{d\,SSR}{d\,b_3} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,b_3} = \sum_{i=1}^{n=3} -2 \times (Observed_i - Predicted_i) \times 1$$



How do we pick the initial weights and biases

Weights

we pick initial numbers from a standard normal Distribution

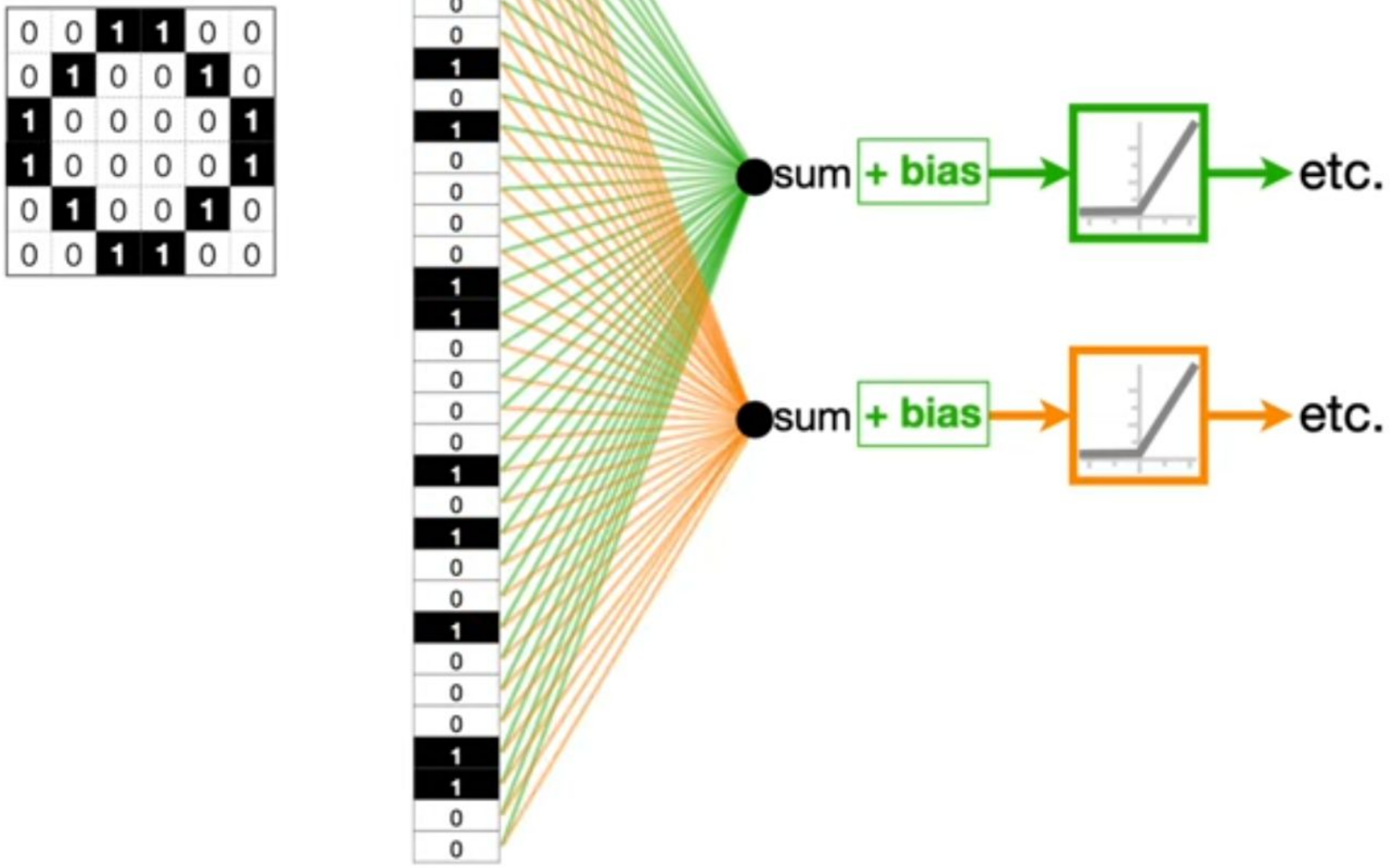


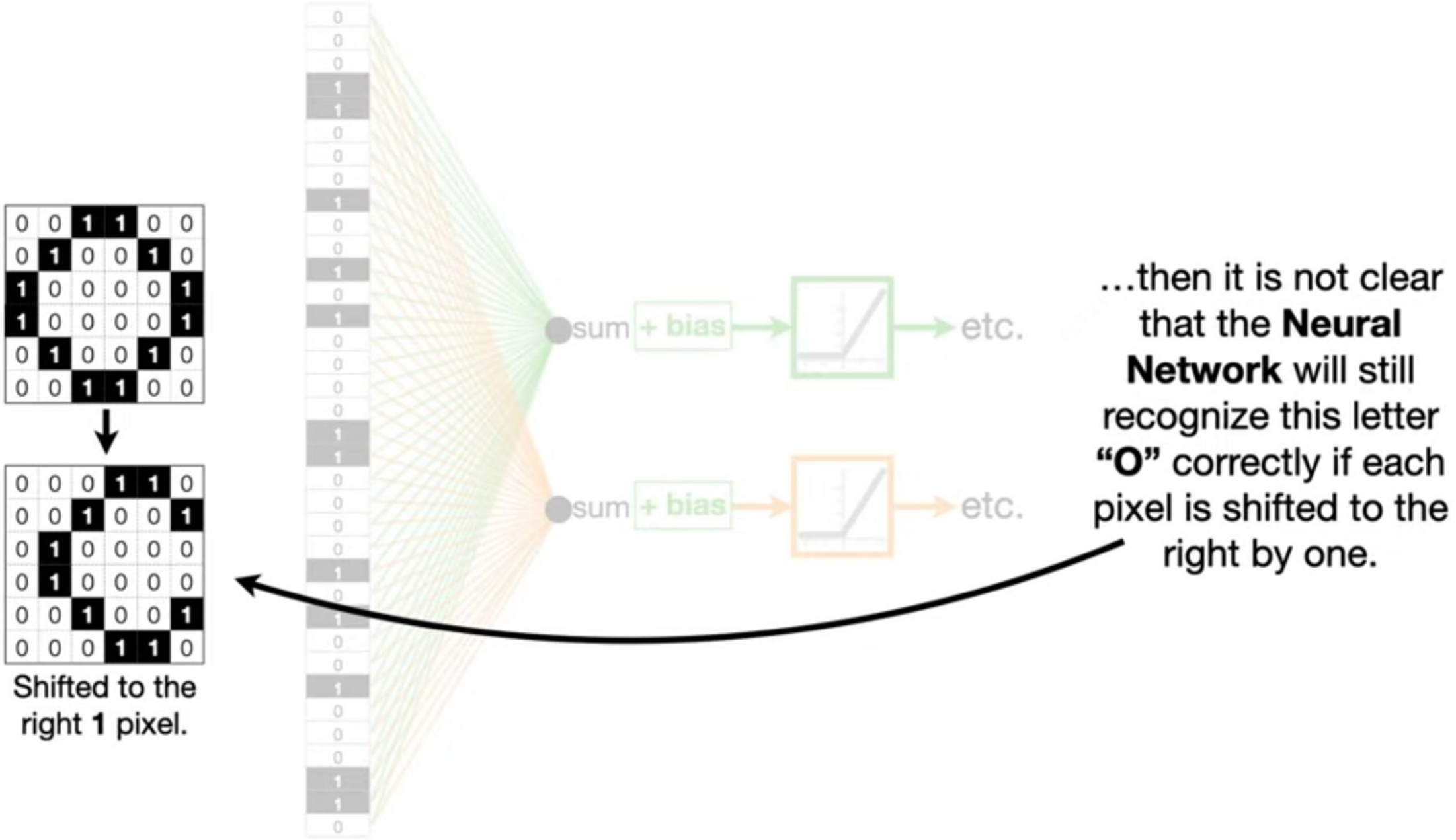
Biases

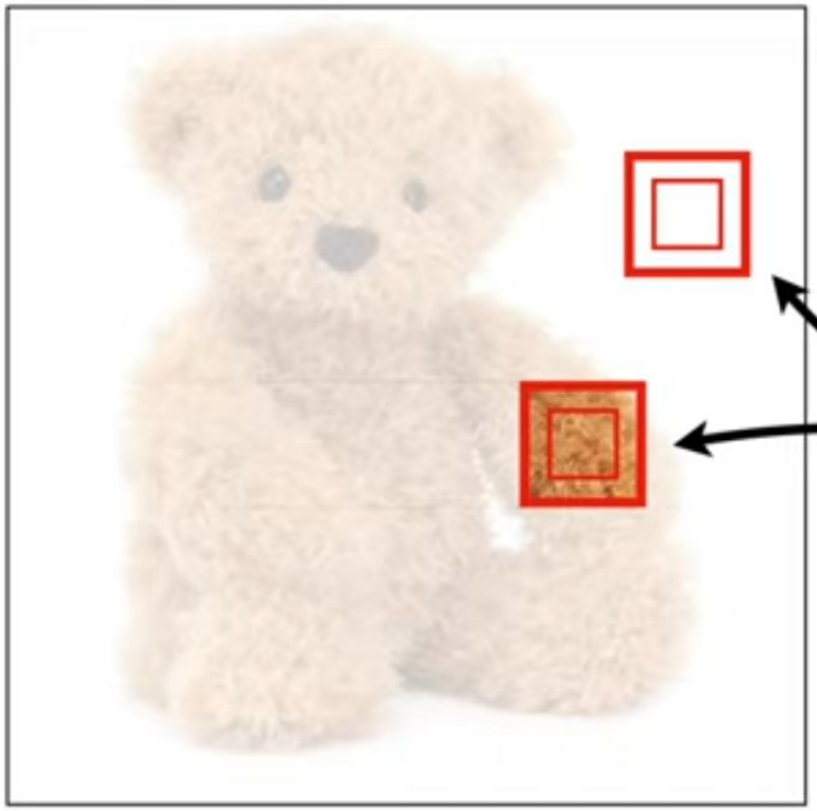
We initialize them to o by standard

04

Convolutional Neural Networks

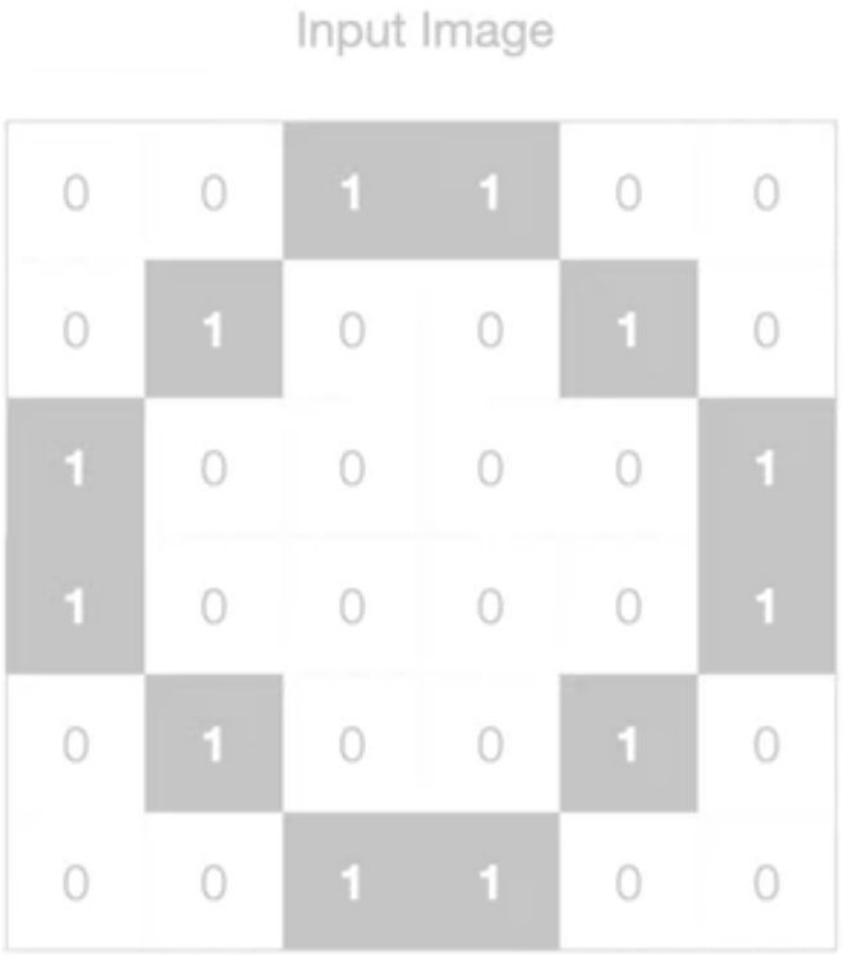

0 0 1 1 0 0
0 1 0 0 1 0
1 0 0 0 0 1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
0 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 1 0 0
sum + bias
etc.
sum + bias
etc.

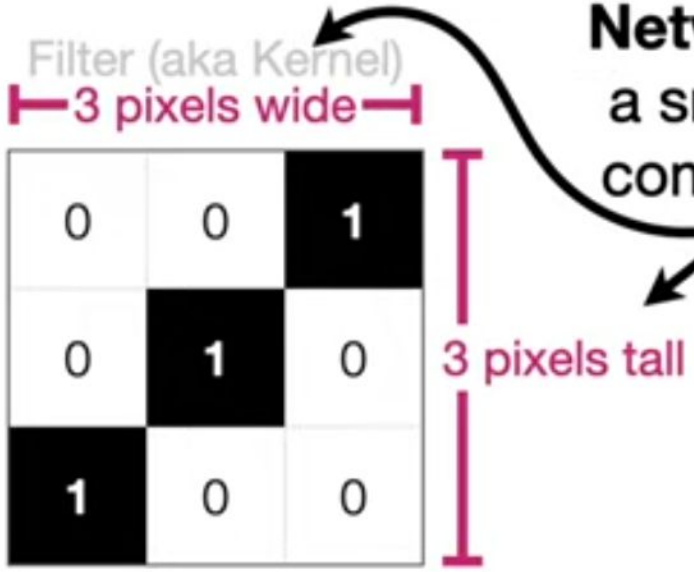

sum + bias
etc.
sum + bias
etc.
Shifted to the right 1 pixel.
…then it is not clear that the Neural Network will still recognize this letter "O" correctly if each pixel is shifted to the right by one.



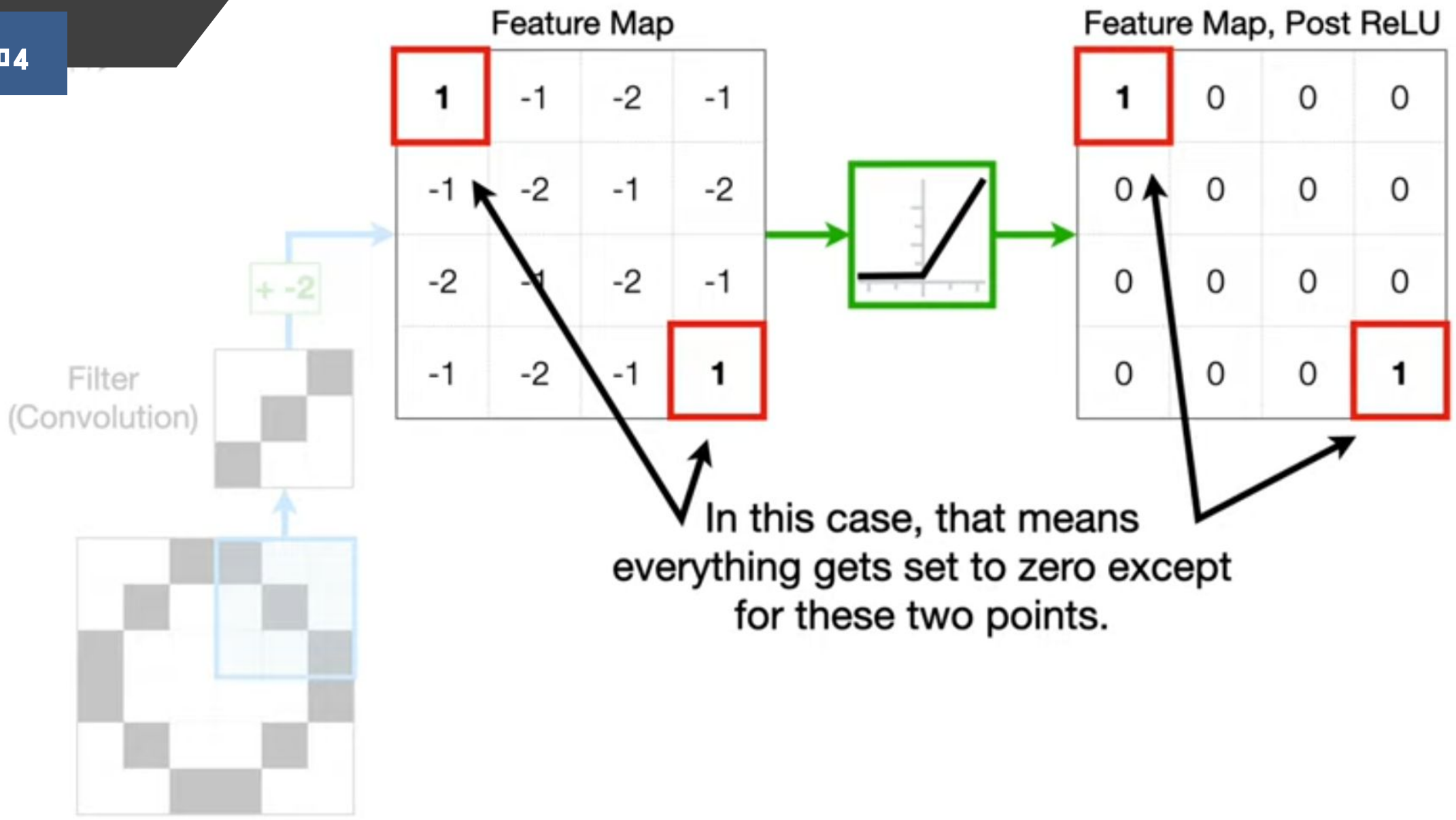
Convolutional Neural Networks do **3** things to make image classification practical:

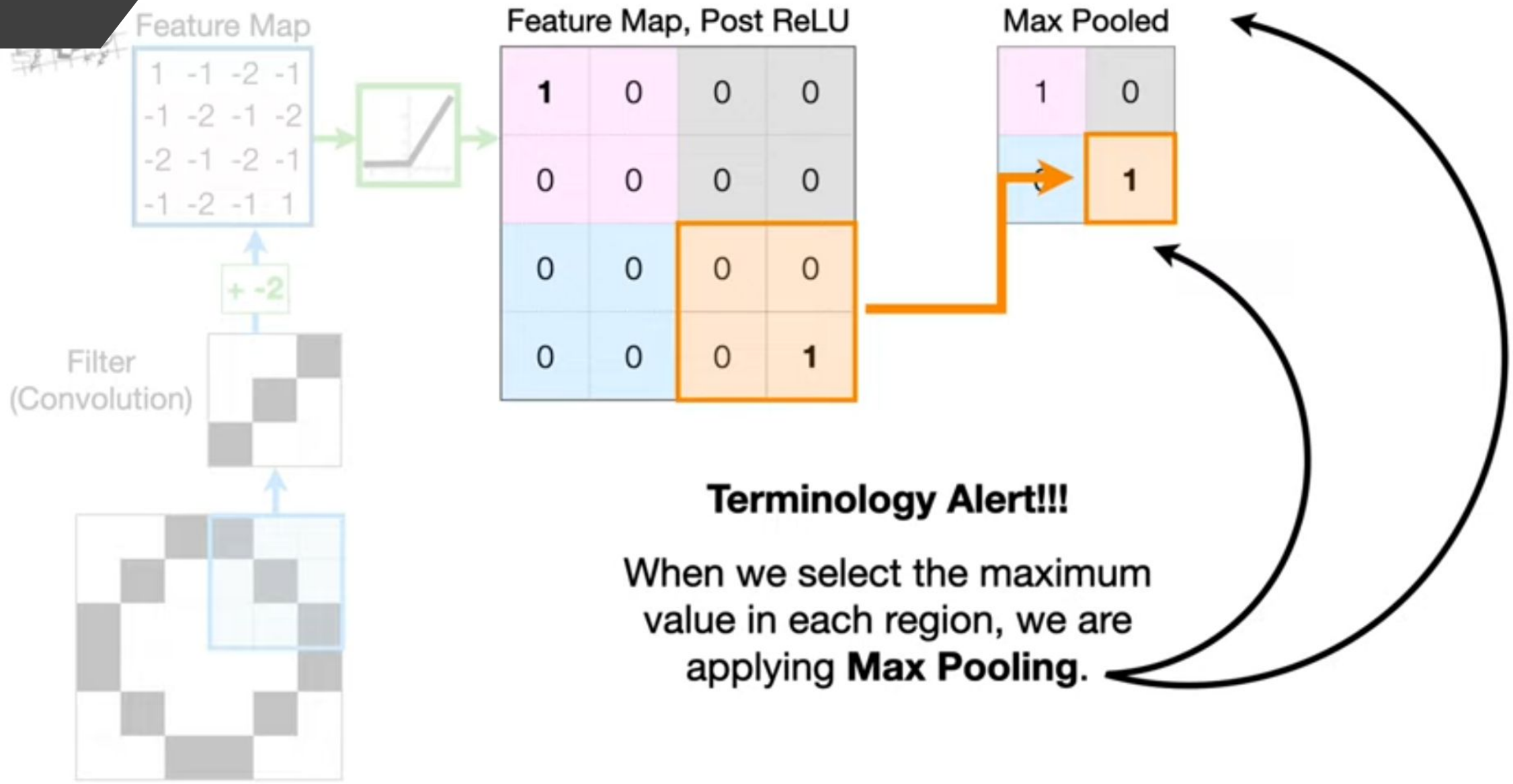
1) Reduce the number of input nodes.
2) Tolerate small shifts in where the pixels are in the image.
3) Take advantage of the correlations that we observe in complex images.





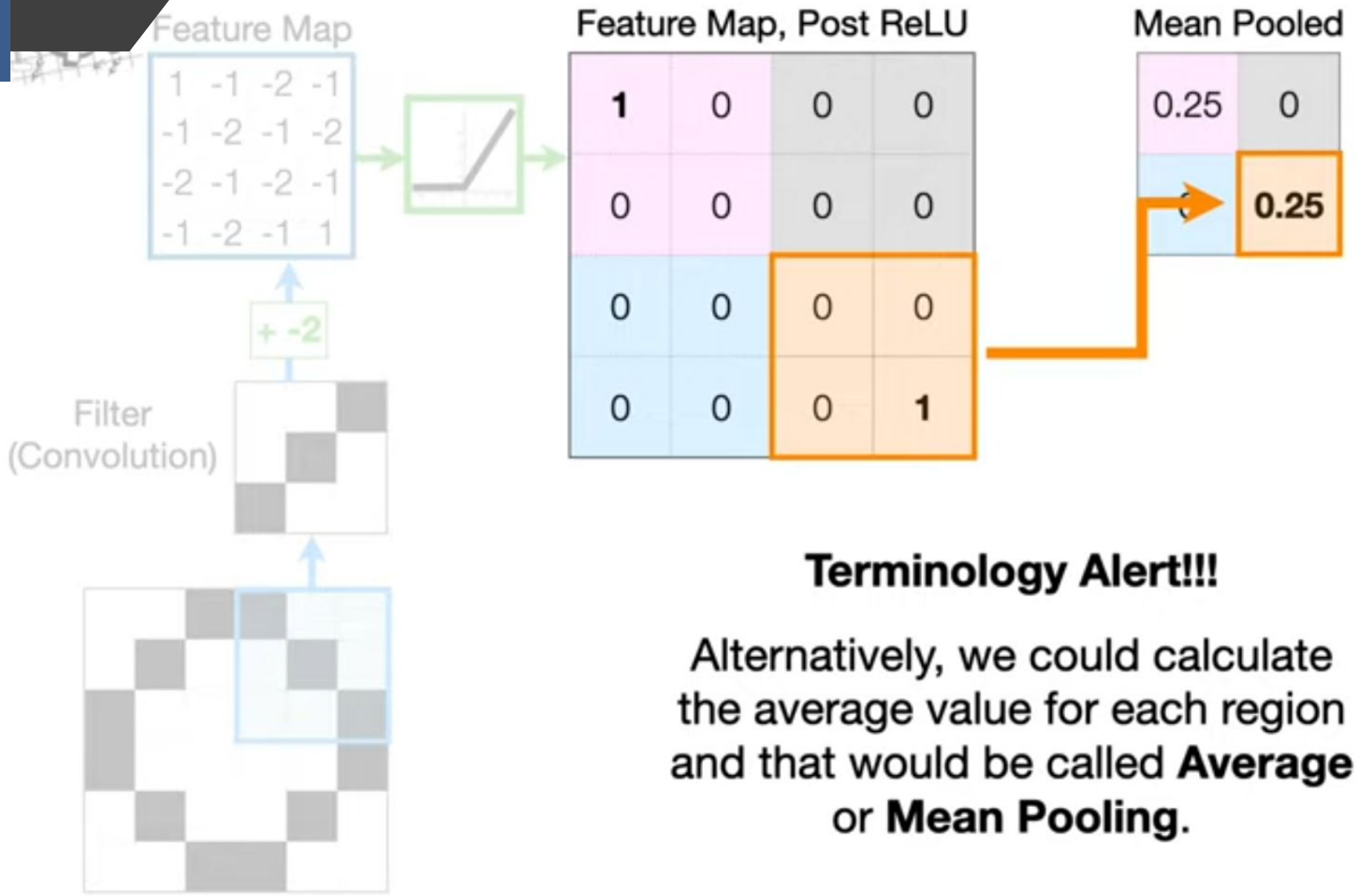
In **Convolutional Neural Networks**, a **filter** is just a smaller square that is commonly **3** pixels by **3** pixels…


Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
Feature Map, Post ReLU
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
+ -2
Filter
(Convolution)
In this case, that means
everything gets set to zero except
for these two points.



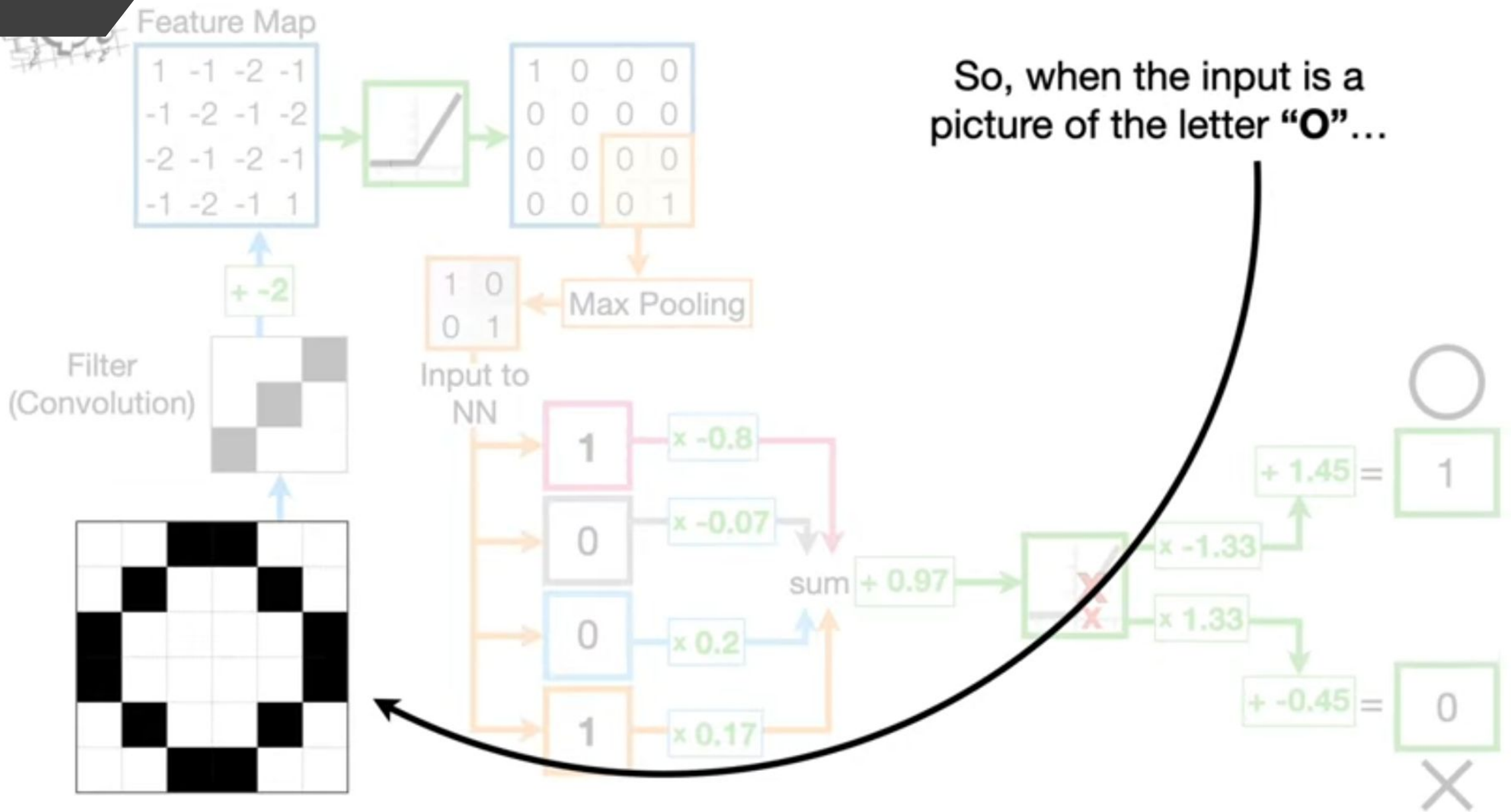
Terminology Alert!!!

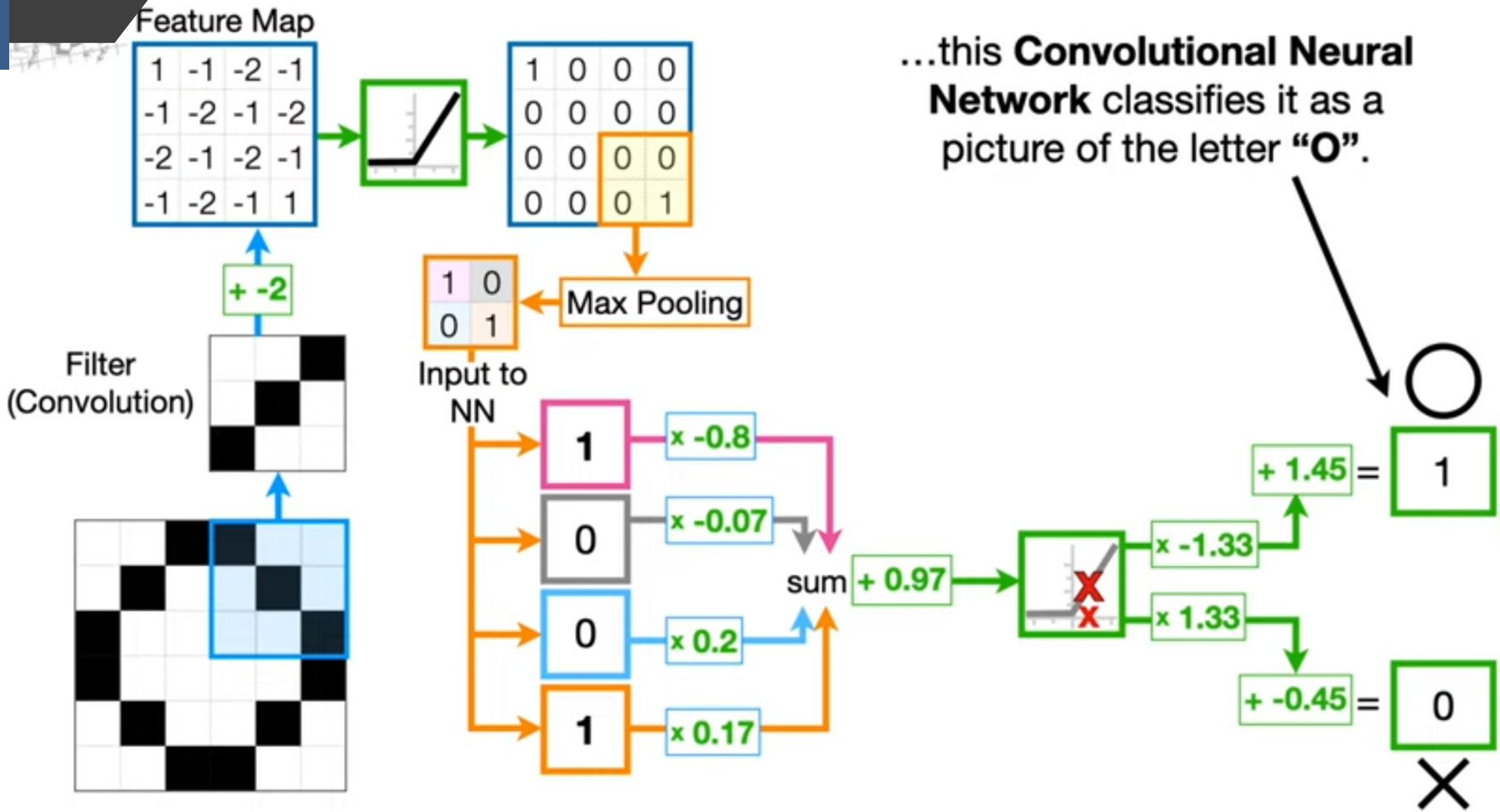
When we select the maximum value in each region, we are applying **Max Pooling**.



Terminology Alert!!!

Alternatively, we could calculate the average value for each region and that would be called **Average** or **Mean Pooling**.

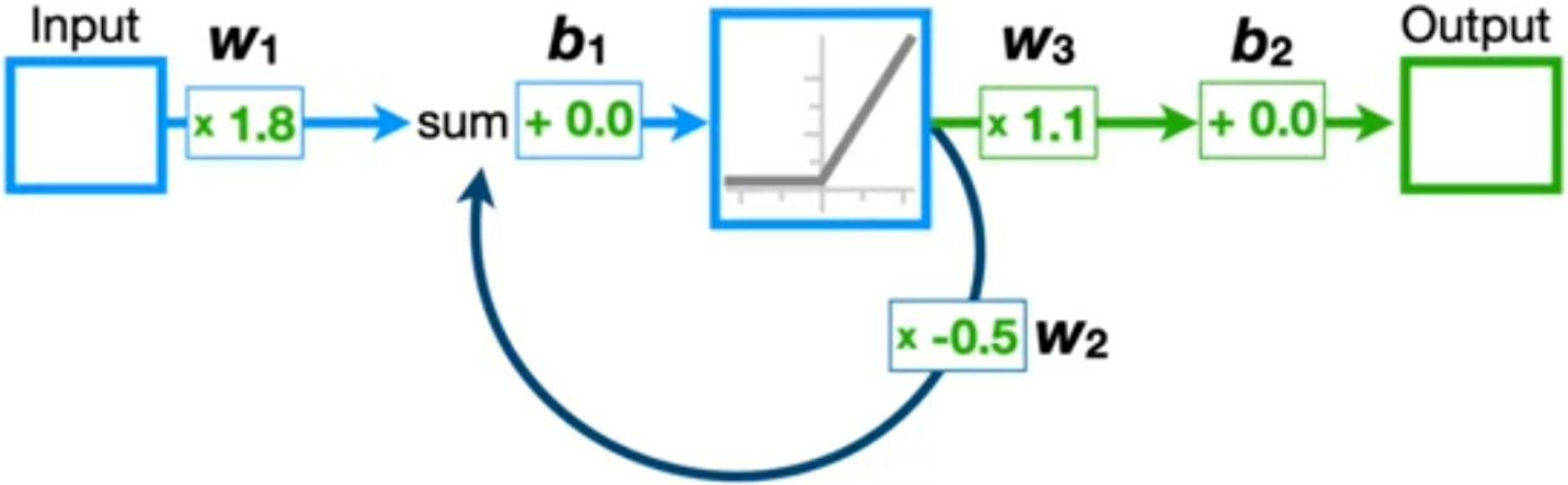

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
+ -2
Filter
(Convolution)
1 0
0 1
Max Pooling
Input to
NN
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 = 1
+ -0.45 = 0
So, when the input is a
picture of the letter "O"…

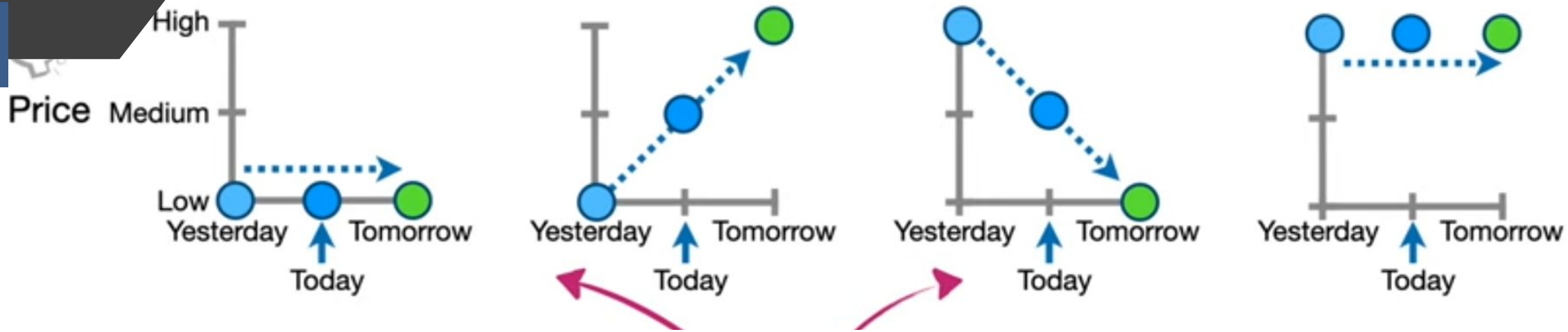

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
Max Pooling
1 0
0 1
Input to
NN
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
1
+ -0.45 =
0
…this Convolutional Neural Network classifies it as a picture of the letter "O".

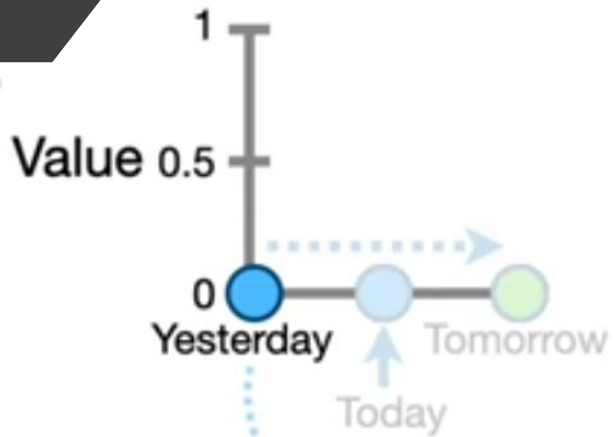
05

Recurrent Neural Networks

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




Price
High
Medium
Low
Yesterday
Today
Tomorrow
Yesterday
Today
Tomorrow
Yesterday
Today
Tomorrow
Yesterday
Today
Tomorrow

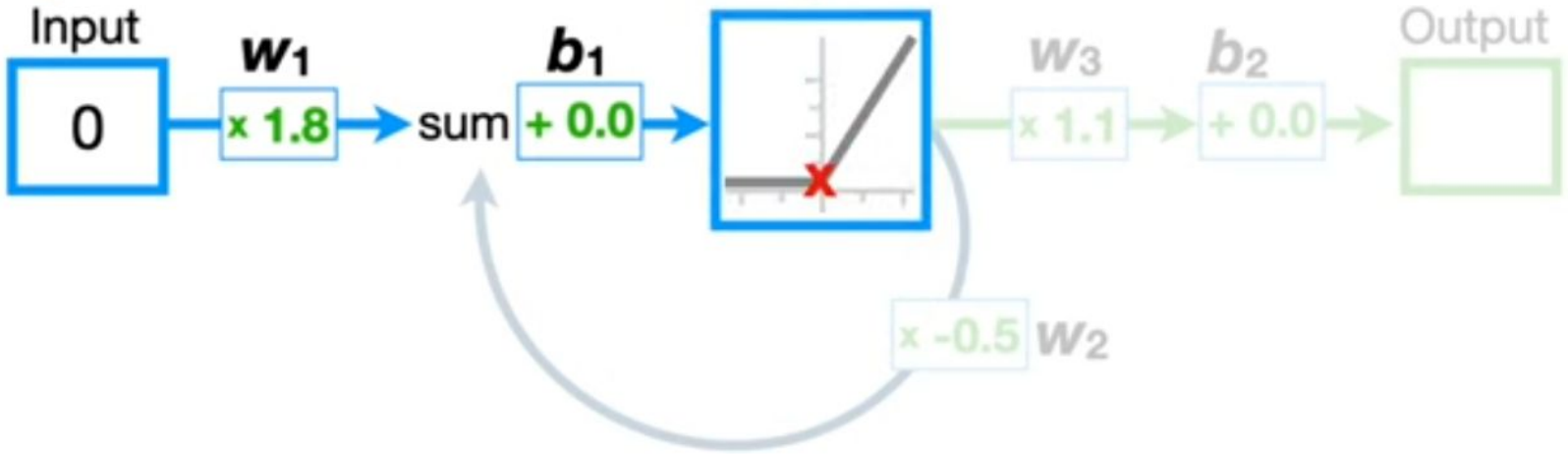


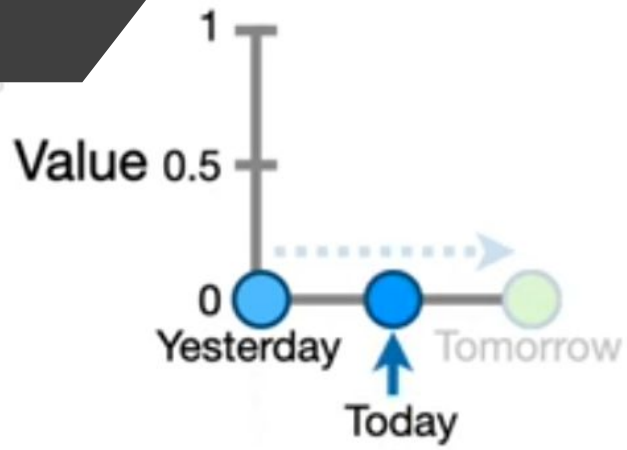
Yesterday × w_1 + b_1 = x-axis coordinate

0 × 1.8 + 0.0 = 0

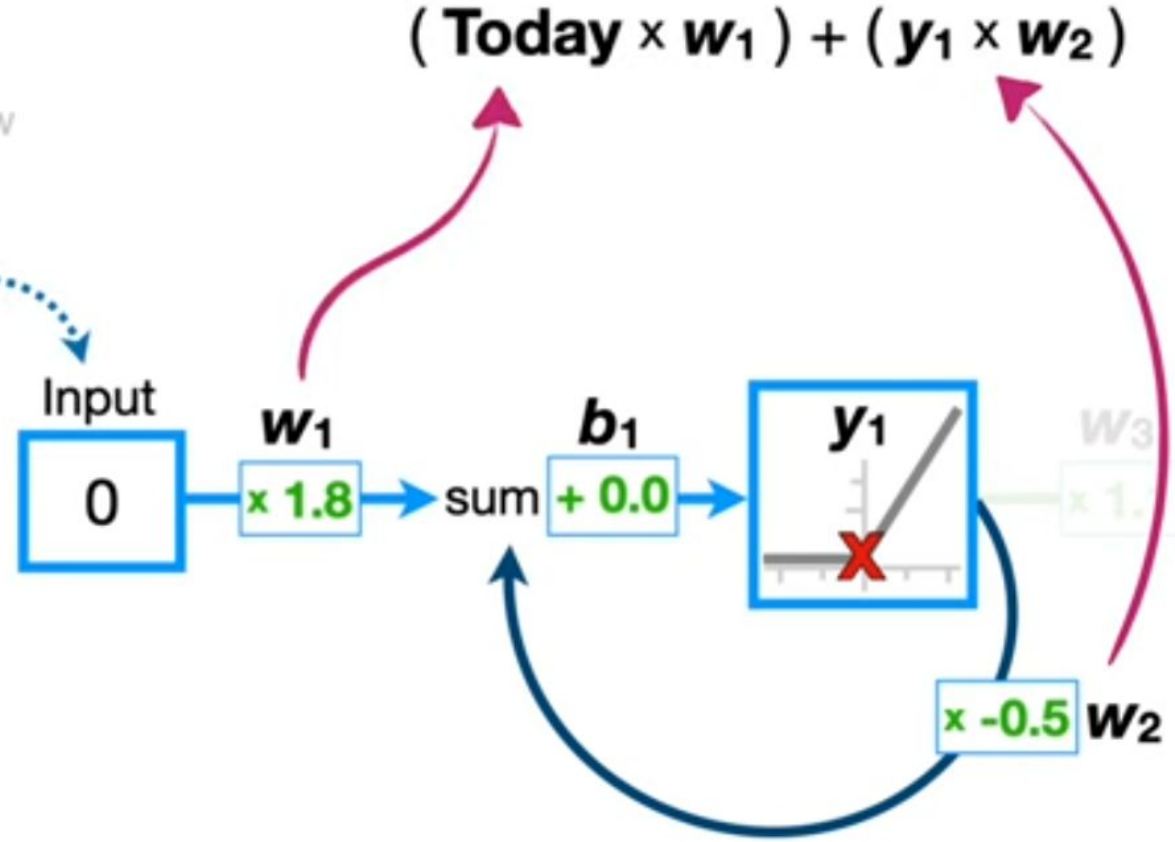
$f(0) = \max(0, 0)$ = y-axis coordinate

Now we can do the math just like we would for any other neural network.

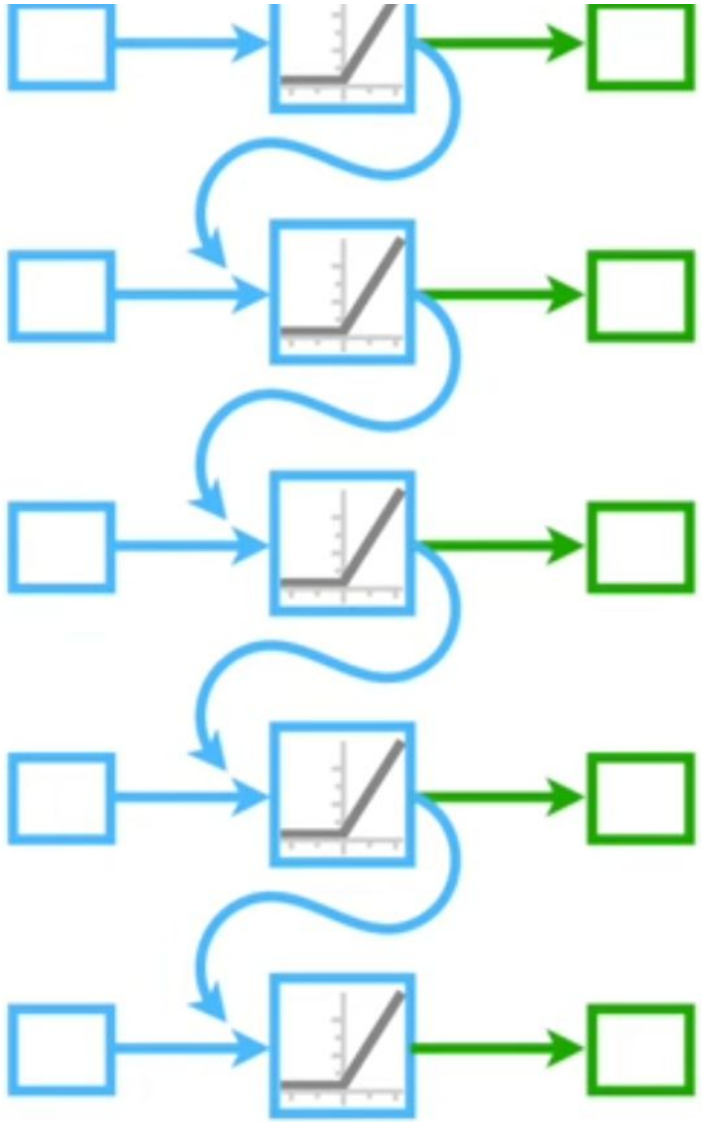




In other words, the **feedback loop** allows both **yesterday** and **today's** values to influence the prediction.



This problem is called **The Vanishing/Exploding Gradient Problem.**



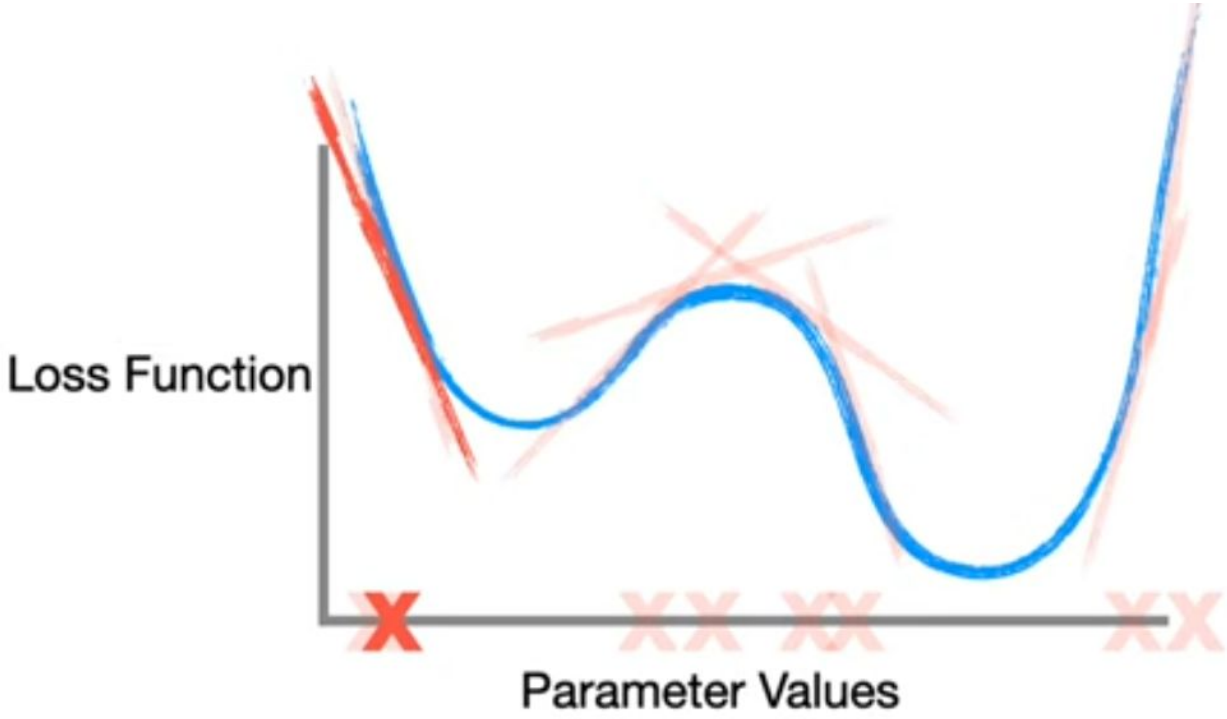
…we multiply the input value by w_2, which is **2**, raised to the number of times we **unrolled**, which is **4**.

$$\text{Input}_1 \times 2 \times 2 \times 2 \times 2$$

$$= \text{Input}_1 \times 2^4$$

$$= \text{Input}_1 \times w_2^{\text{Num. Unroll}}$$

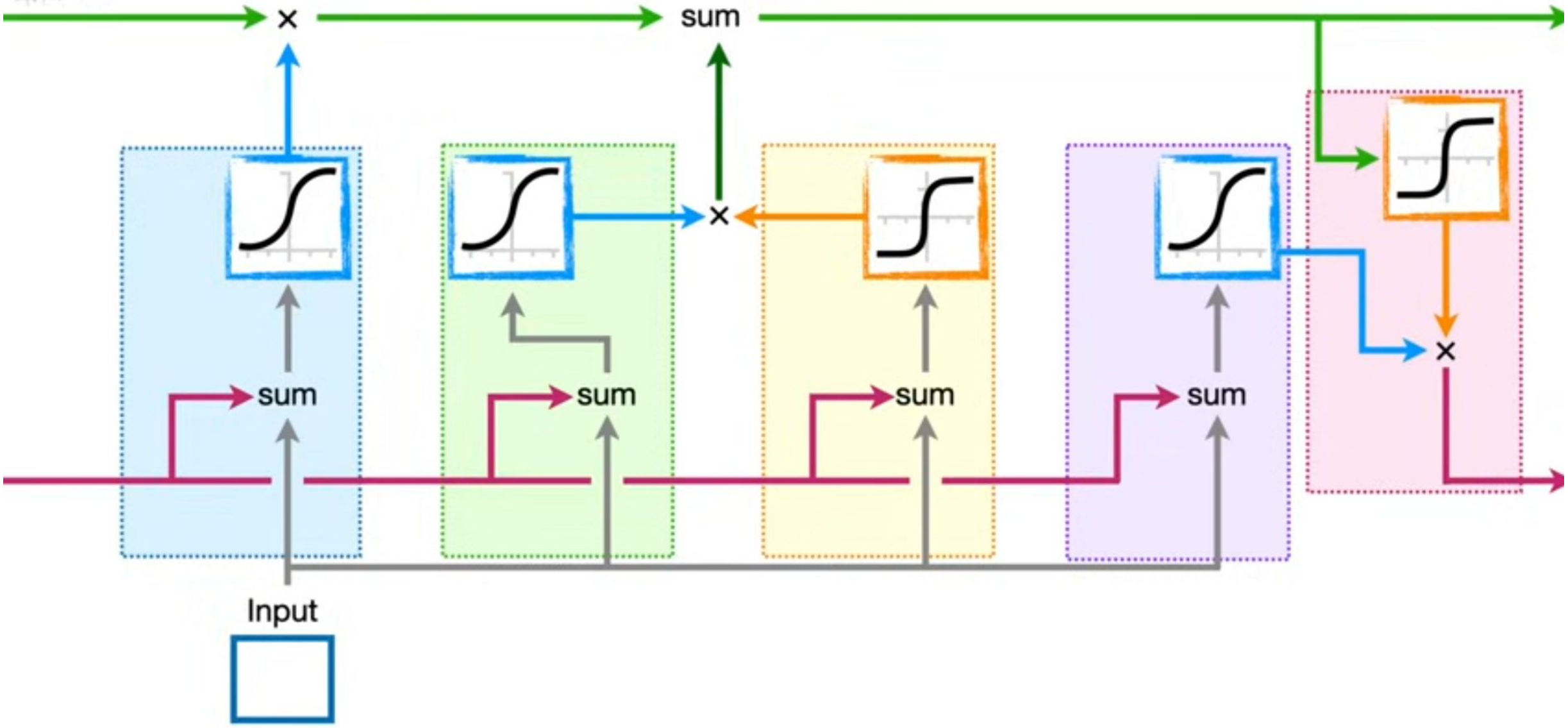
One way to prevent **The Exploding Gradient Problem** would be to limit w_2 to values **< 1**.

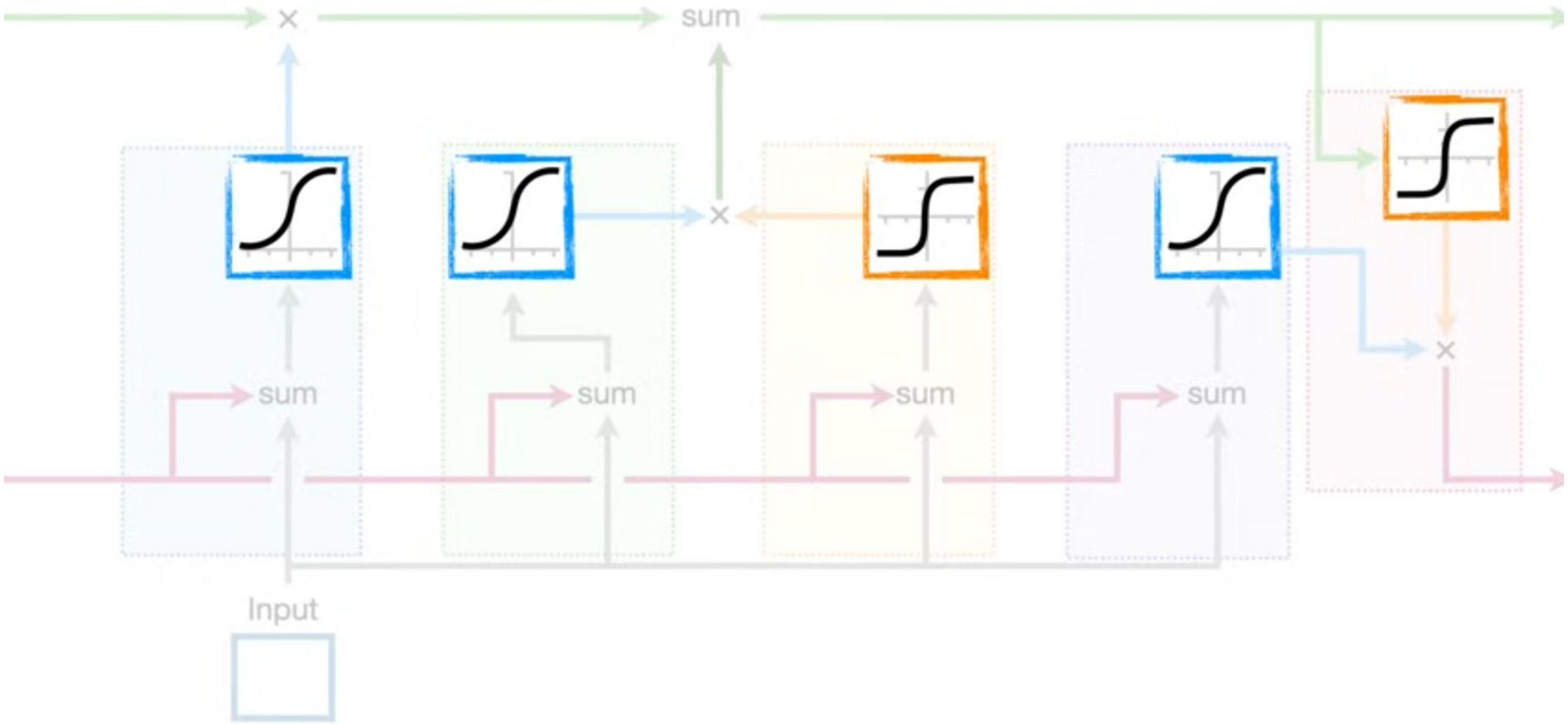


Now, just like before, we multiply the first input by w_2 raised to the number of times we **unroll** the network.

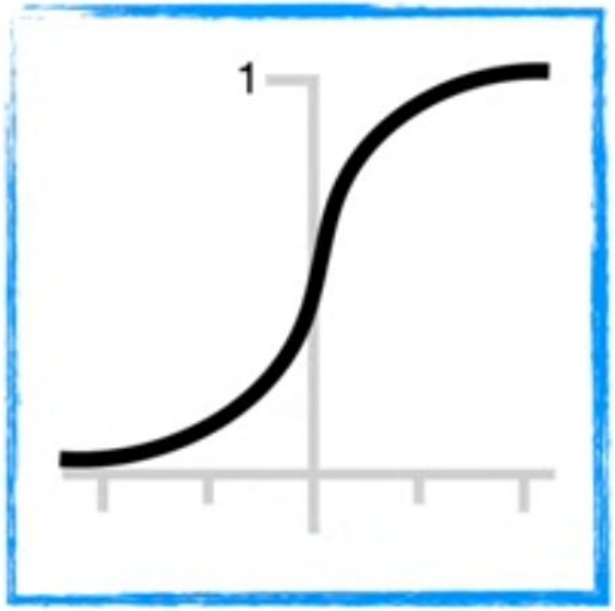
$$\text{Input}_1 \times w_2^{\text{Num. Unroll}}$$

…**Long Short-Term Memory** is based on a much more complicated unit.

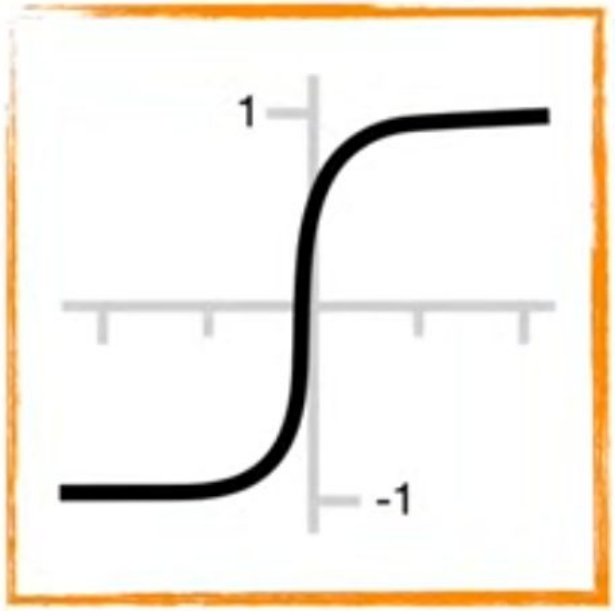


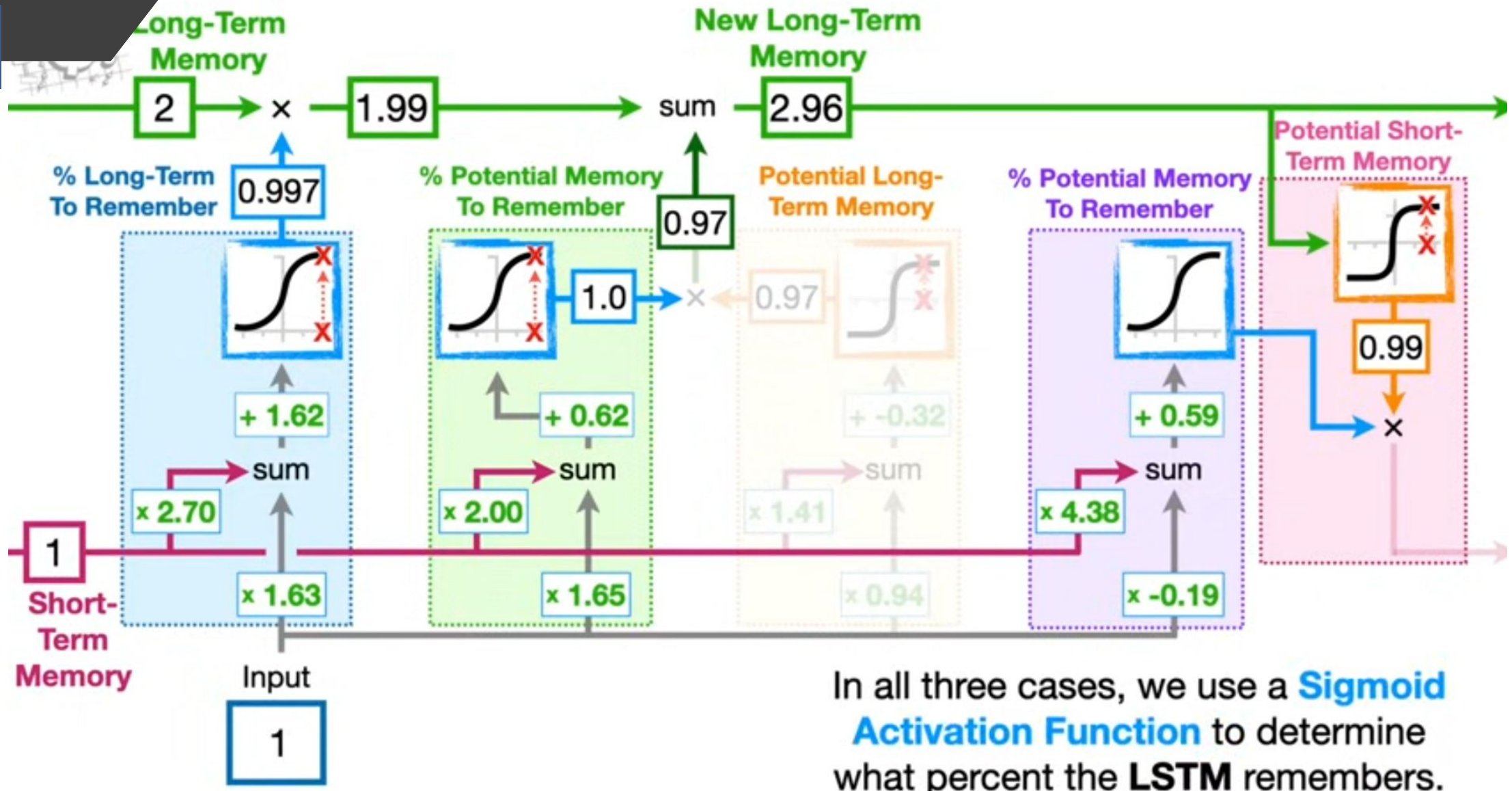

sum
sum
sum
sum
sum
Input

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**...





In all three cases, we use a **Sigmoid Activation Function** to determine what percent the **LSTM** remembers.

Conclusion


Thank You