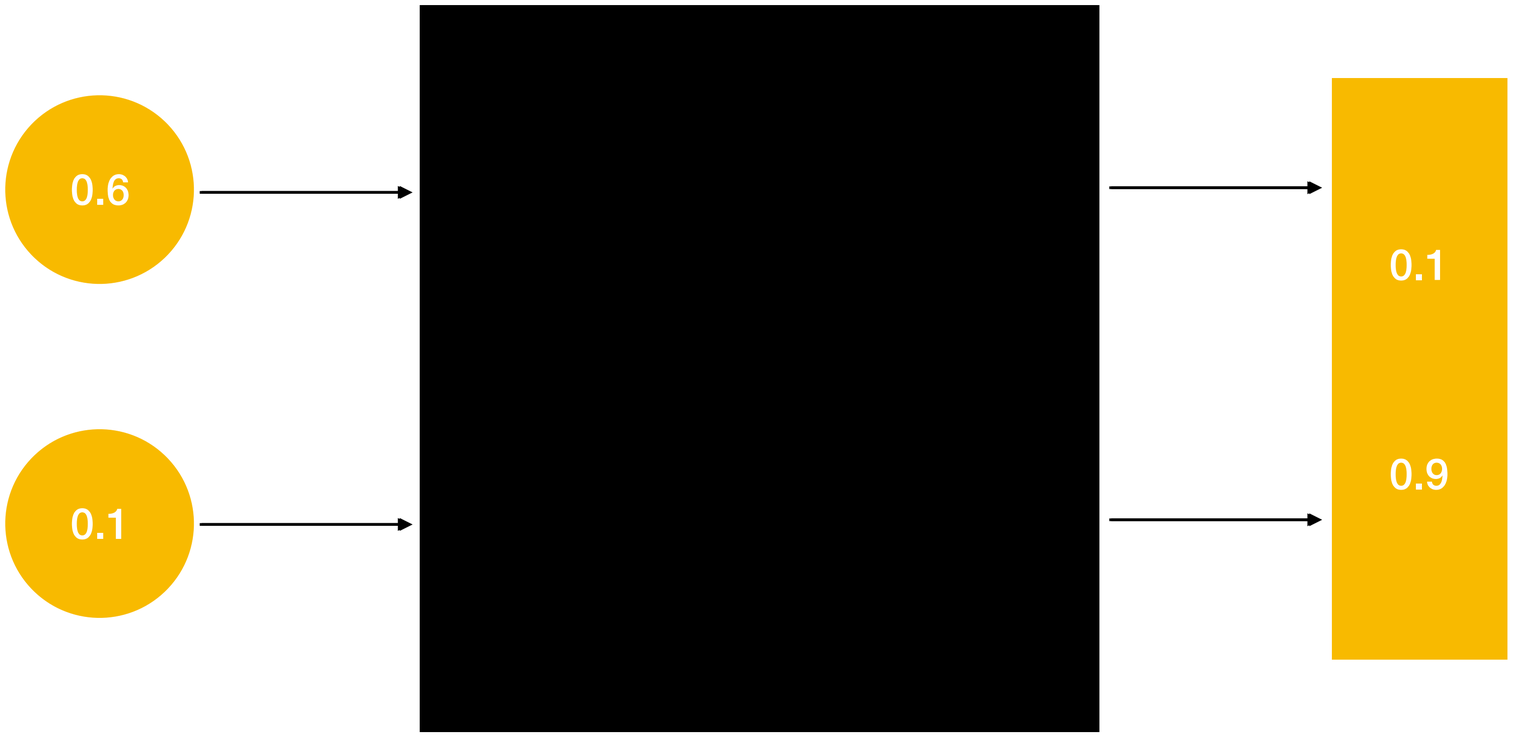
Neural Networks

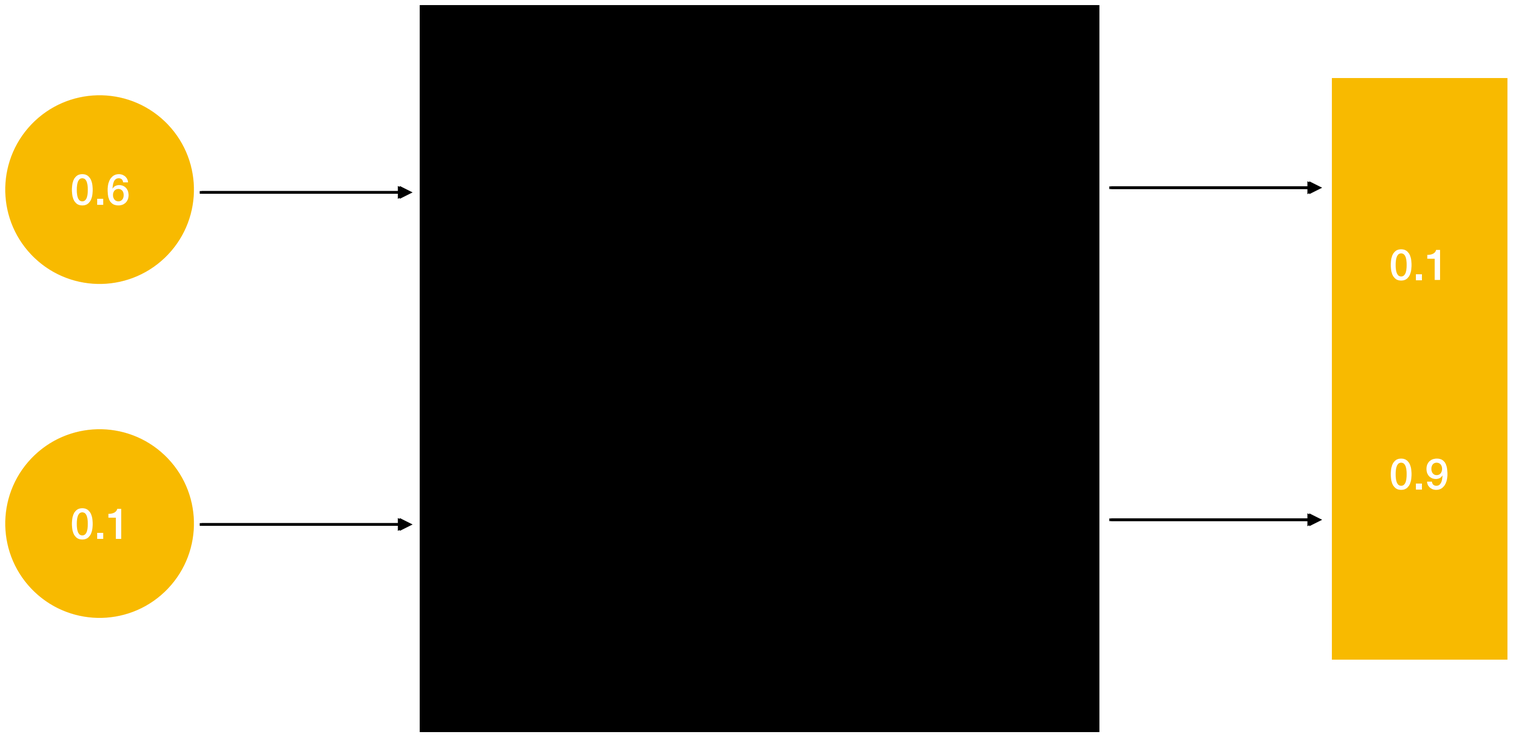
What are they?
How do they work?
Where am I?
Please, somebody, help me!

What is a neural network?



A neural network is just a *thing* that takes some input values and gives back some outputs

What is a neural network?

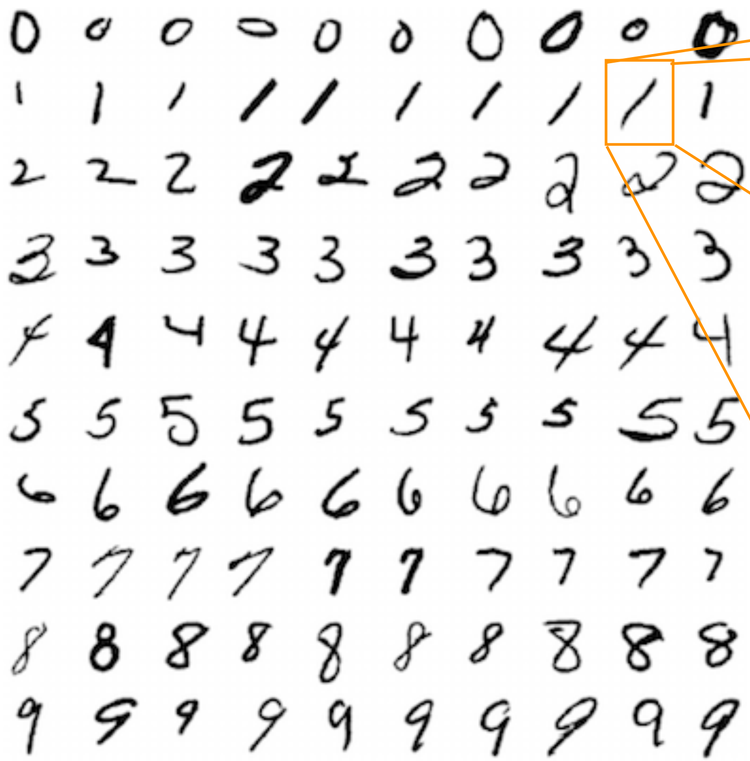


The input to a network can be anything that can be *meaningfully* converted to numbers

What is a neural network?

Like… pictures of handwritten numbers

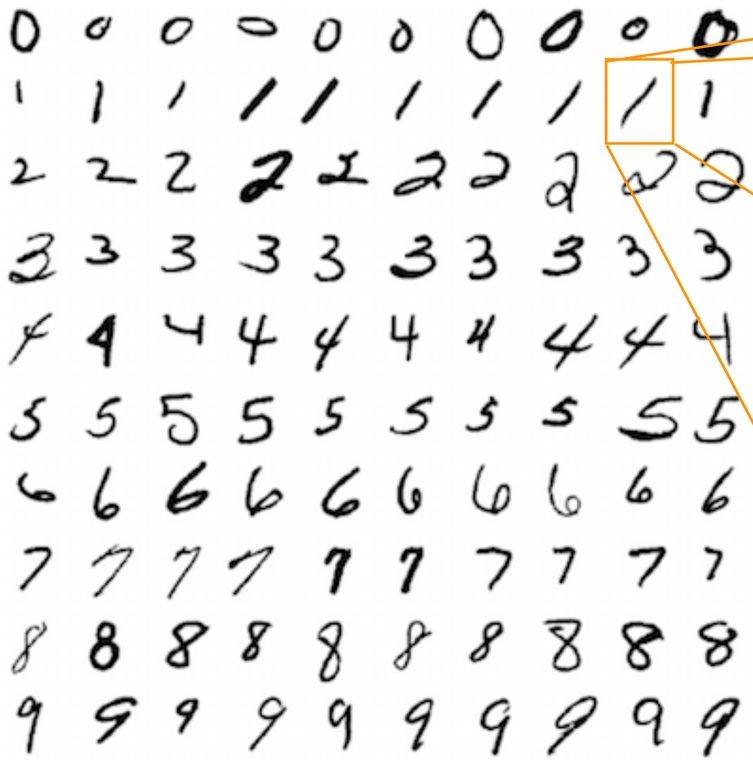
What is a neural network?



0	0	0	0	0	0	0	0
0	0	0	0	0	0.5	0	0
0	0	0	0	0.7	0.7	0	0
0	0	0	0.3	1	0.3	0	0
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0
0	0	0.5	0	0	0	0	0
0	0	0	0	0	0	0	0

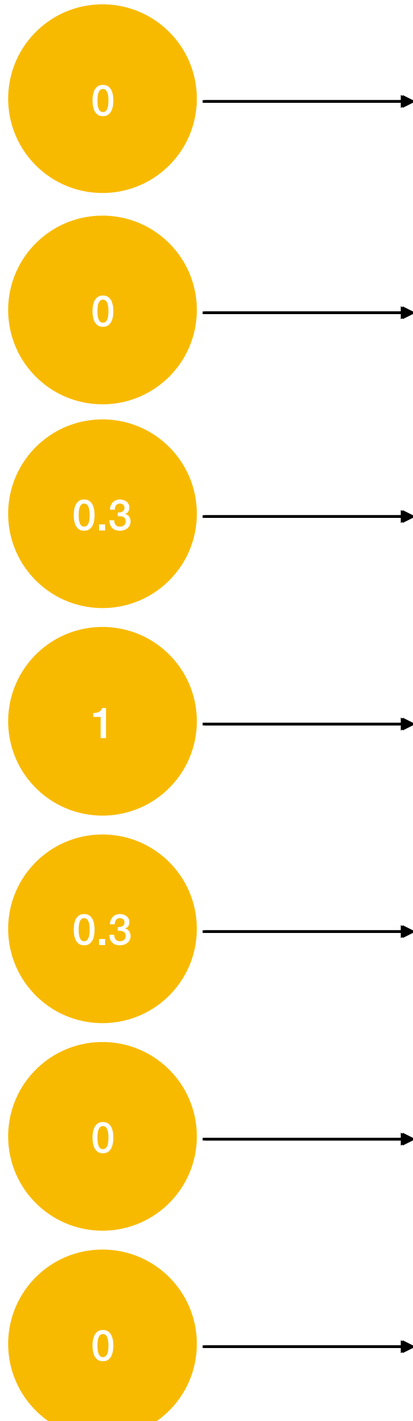
We can turn each one into values representing how dark each of the pixels are

What is a neural network?



0	0	0	0	0	0	0	0
0	0	0	0	0	0.5	0	0
0	0	0	0	0.7	0.7	0	0
0	0	0	0.3	1	0.3	0	0
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0
0	0	0.5	0	0	0	0	0
0	0	0	0	0	0	0	0

And then each of those values can be an input to our network…



In this case, the goal might be to work out what digit we are looking at

It's a 1

It's a 2

It's a 3

It's a 4

It's a 5

It's a 6

It's a 7...

Each output neuron can represent one digit, and so we have 10 outputs for 0, 1, 2, 3, all the way to 9

In this case, the goal might be to work out what digit we are looking at

It's a 1

It's a 2

It's a 3

It's a 4

It's a 5

It's a 6

It's a 7...

If an output value is 1, the network thinks it's the corresponding digit. If it's 0, it does not.

To begin with, the network won't really have any idea which digits it's looking at, so the outputs will be all over the place

0.3

0.5

0.3

0.7

0.4

0.6

0.2

Err, it might be a "1"

Nah, maybe it's a 2.

Ermmm, I know this… is it a "3" maybe??

oh, oh, oh! I think it might be a 4?!


you have to train it by
telling it what's
what...
0.3
0.5
0.3
0.7
0.4
0.6
0.2


It's a "1"!!!

And by doing so, it gradually learns..

0.9

0.2

0

0

0

0

0.5

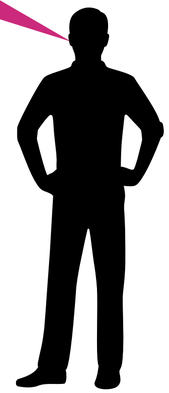
I am pretty damn sure it's a "1".

There's a wee chance that it might be a "2" though..

OK, it's DEFINITELY not a "3"

It's a "1"!!!

Umm, but it could be a "7". Damn these digits for looking so similar..silly humans



…until eventually, it has learned what the digits are

0.9

0.01

0

0

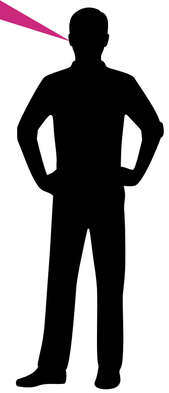
0

0

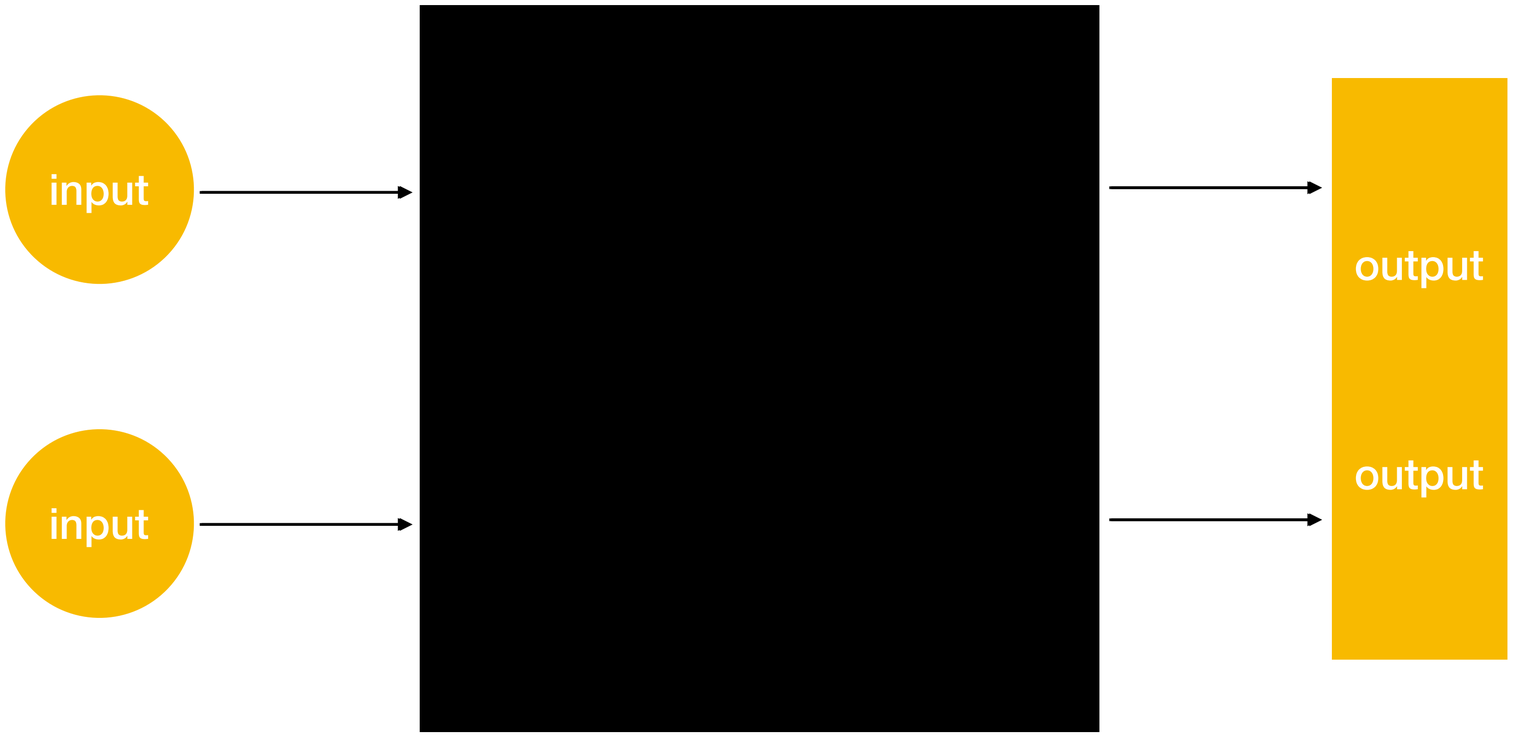
0.02

Defo a "1"

Woohoo! Good job!

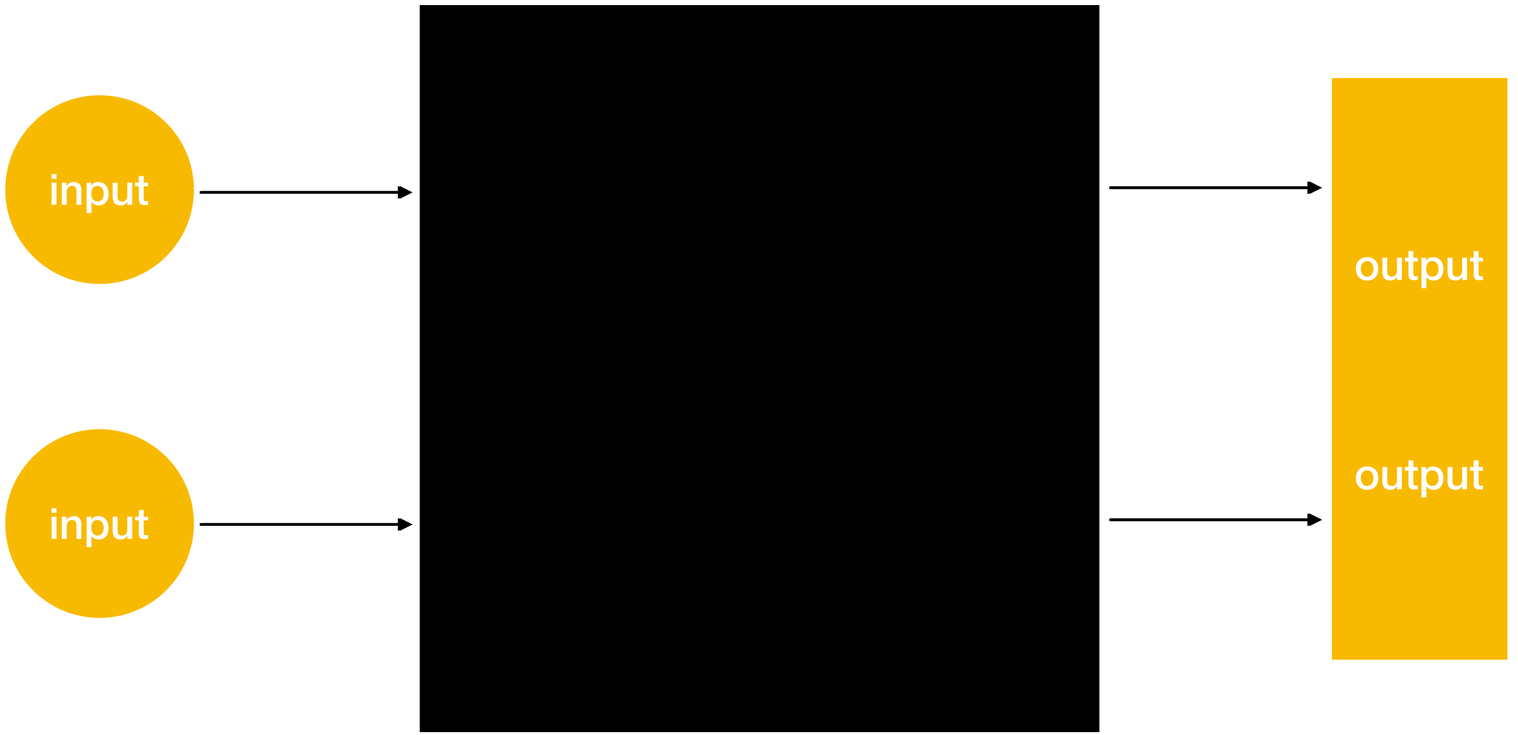


What is a neural network?



So, a network takes in some inputs that represent something, and can be trained to output appropriate values as a result.

What is a neural network?



In fact, a neural network can, in theory, represent *any* mapping from input to output values!

- "universal approximation theorem"

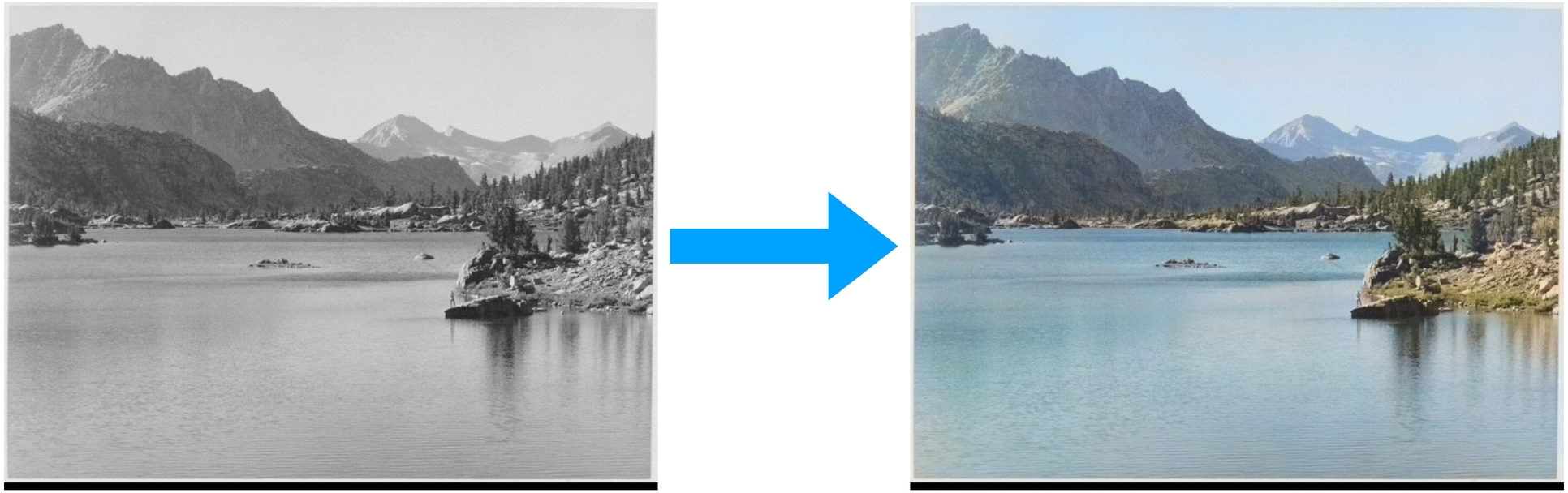
It can learn to map pixels representing handwritten numbers to their value…

“I think it’s a 1”

“I think it’s a 3”

“I think it’s a 7”

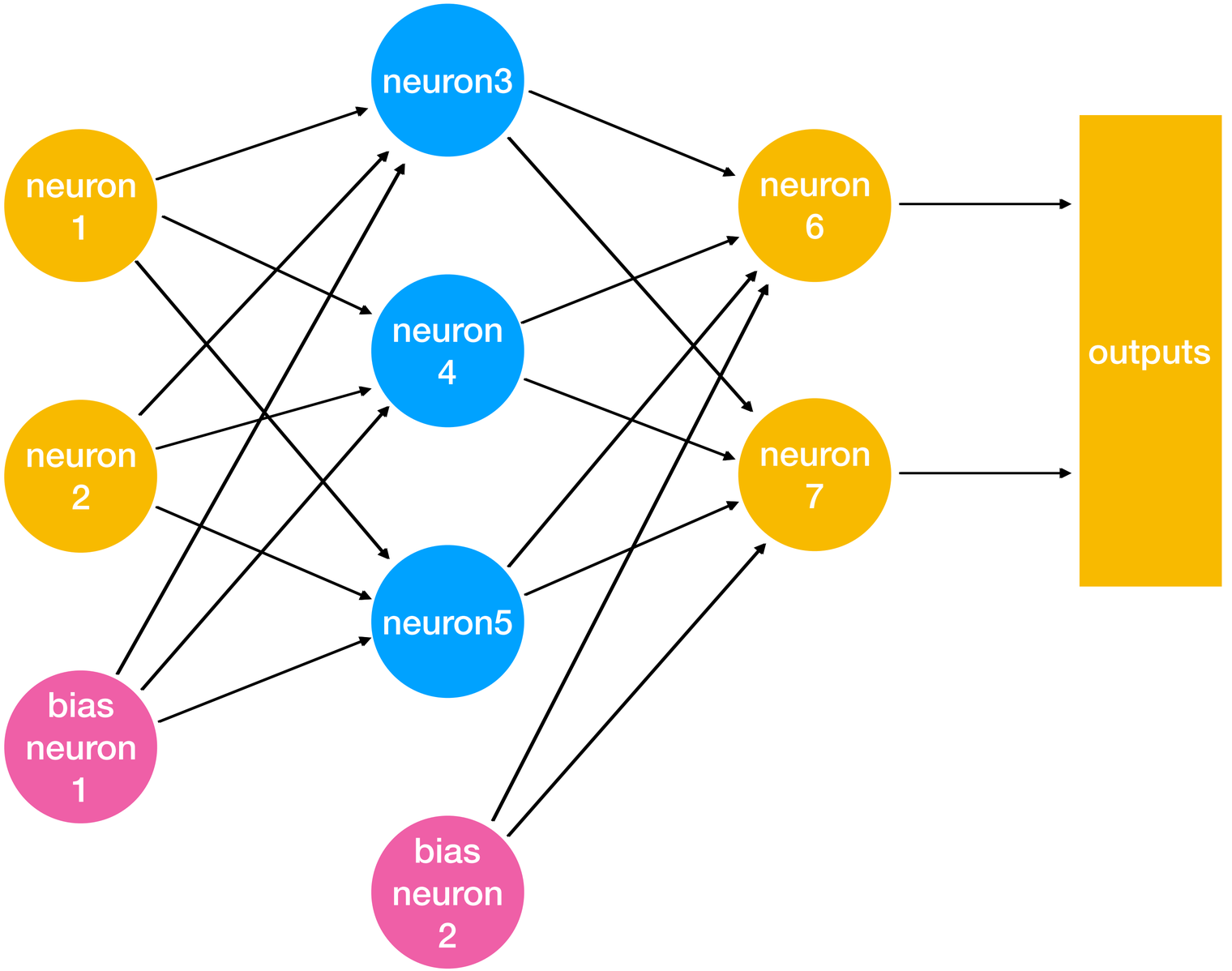
It can learn to map black and white pictures to colour pictures…



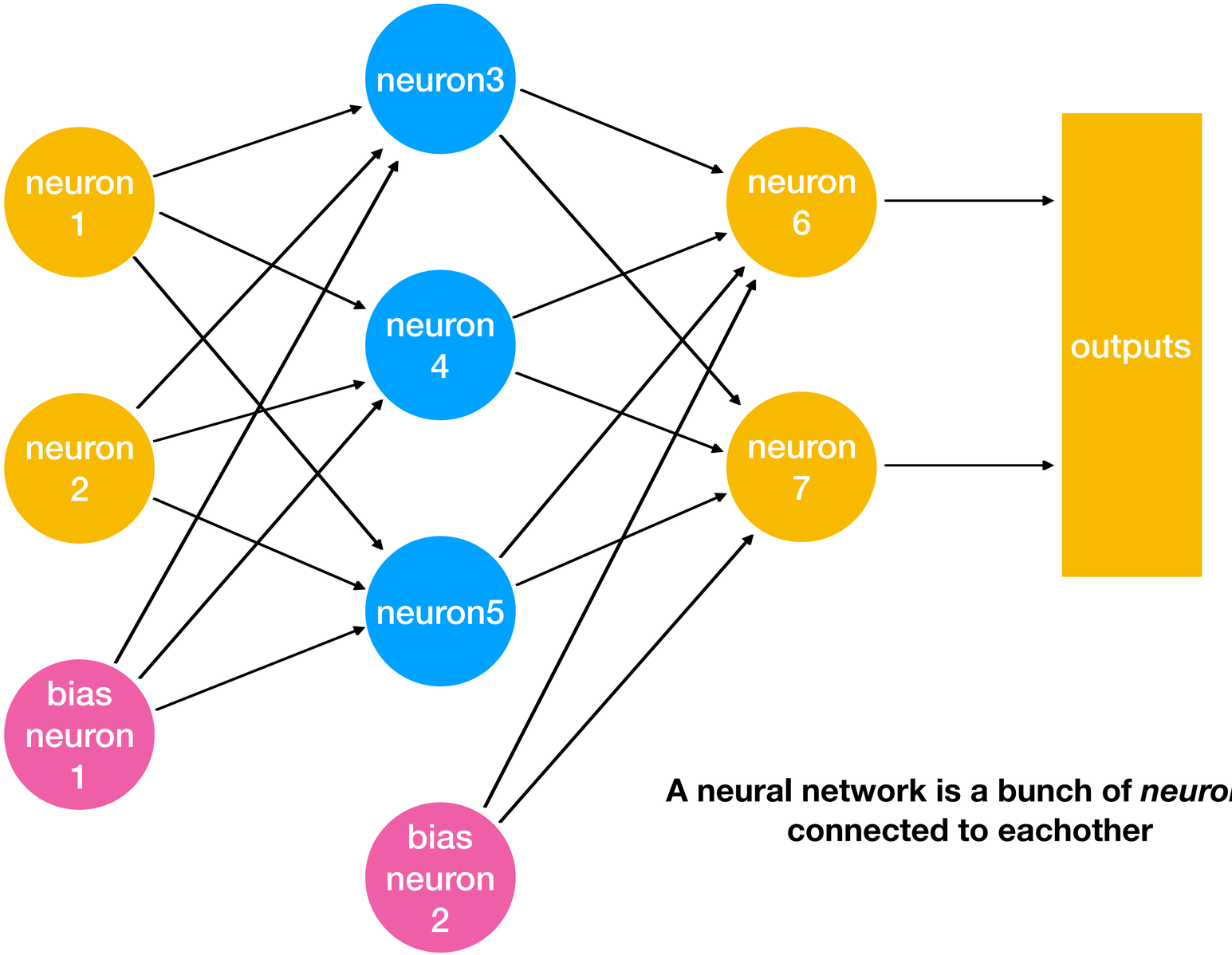
And loads of other things!

http://www.yaronhadad.com/deep-learning-most-amazing-applications/

Let's peel away the black box…

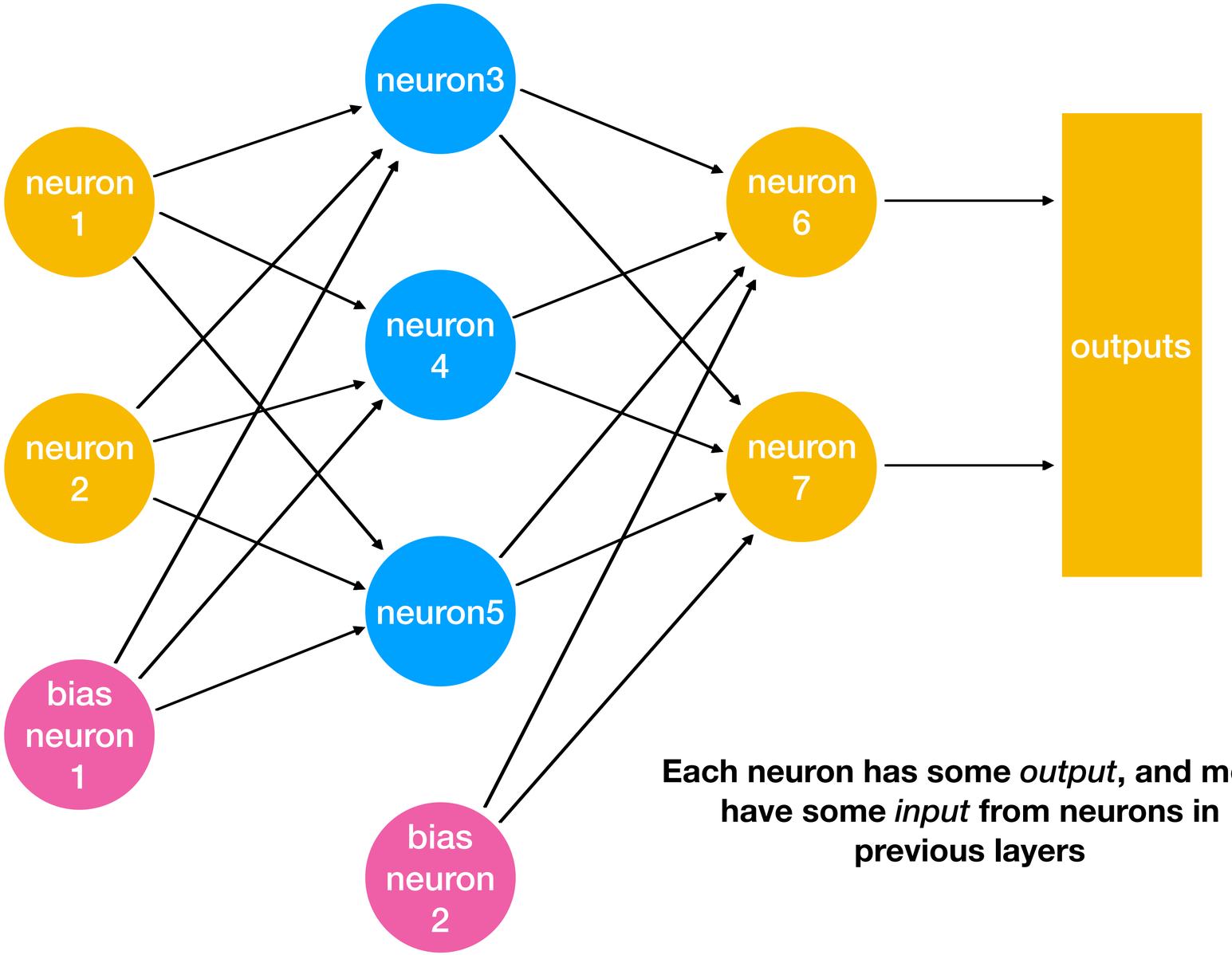


Let's peel away the black box…



A neural network is a bunch of *neurons*, connected to eachother

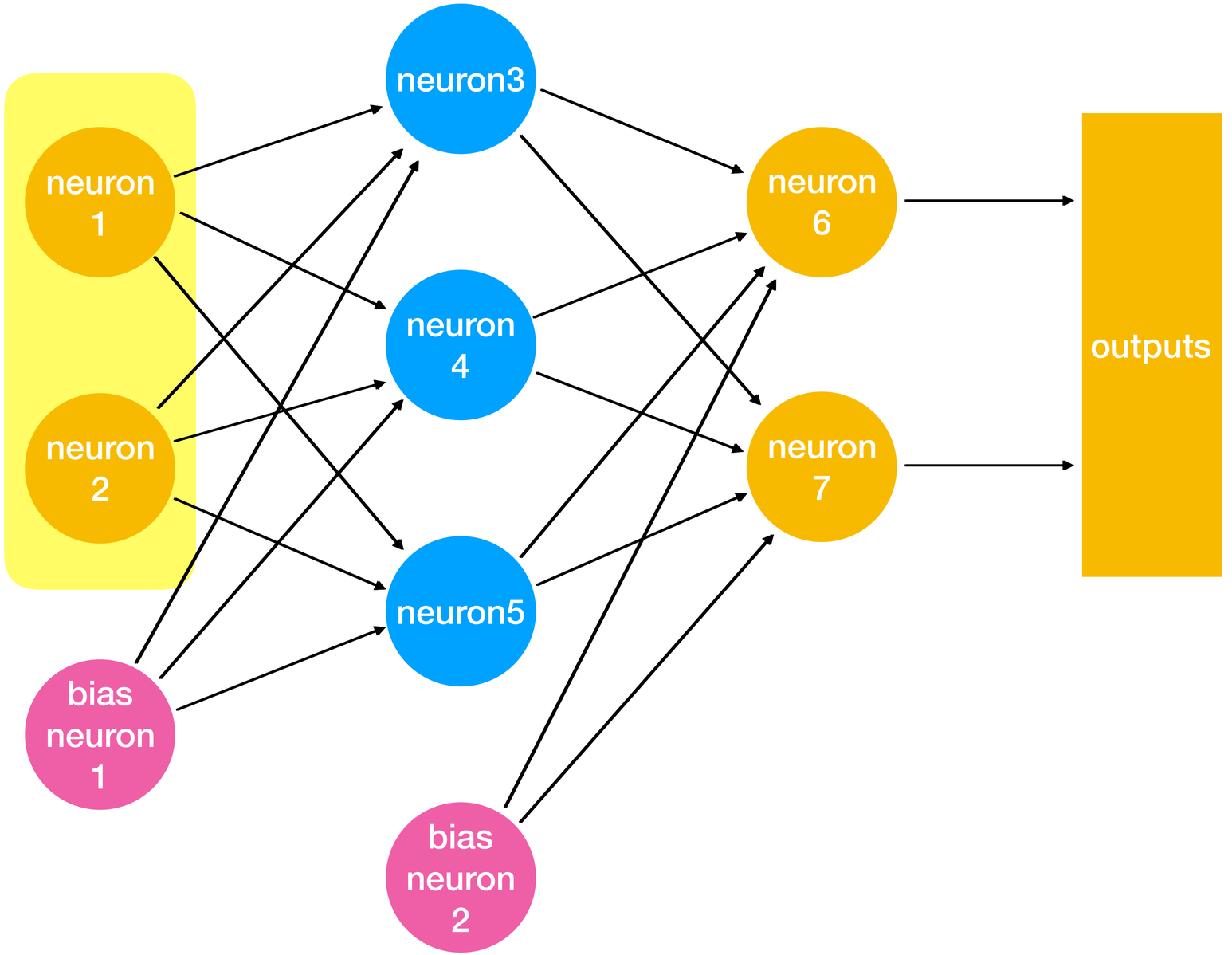
Let's peel away the black box…



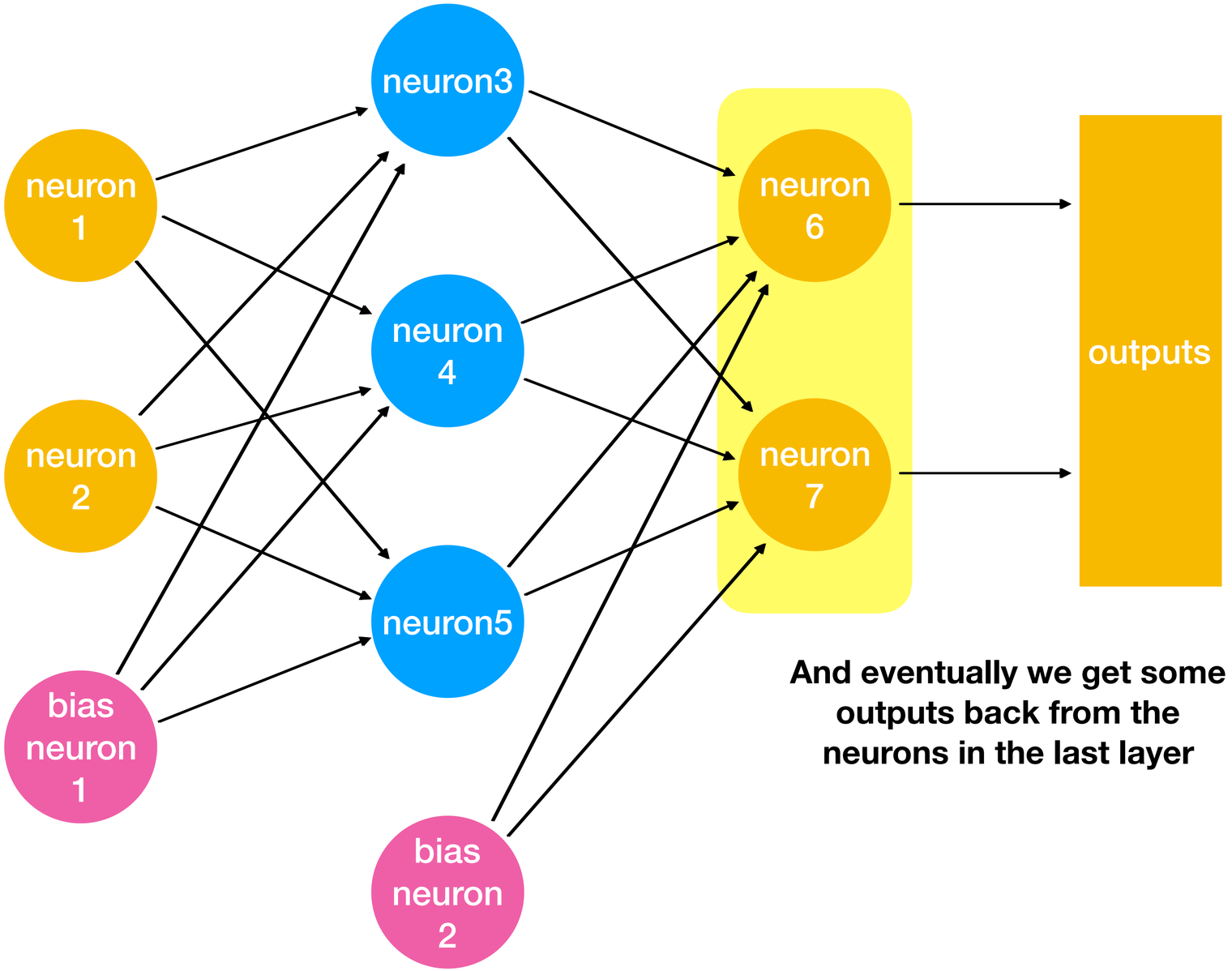
Each neuron has some *output*, and most have some *input* from neurons in previous layers

Let's peel away the black box…

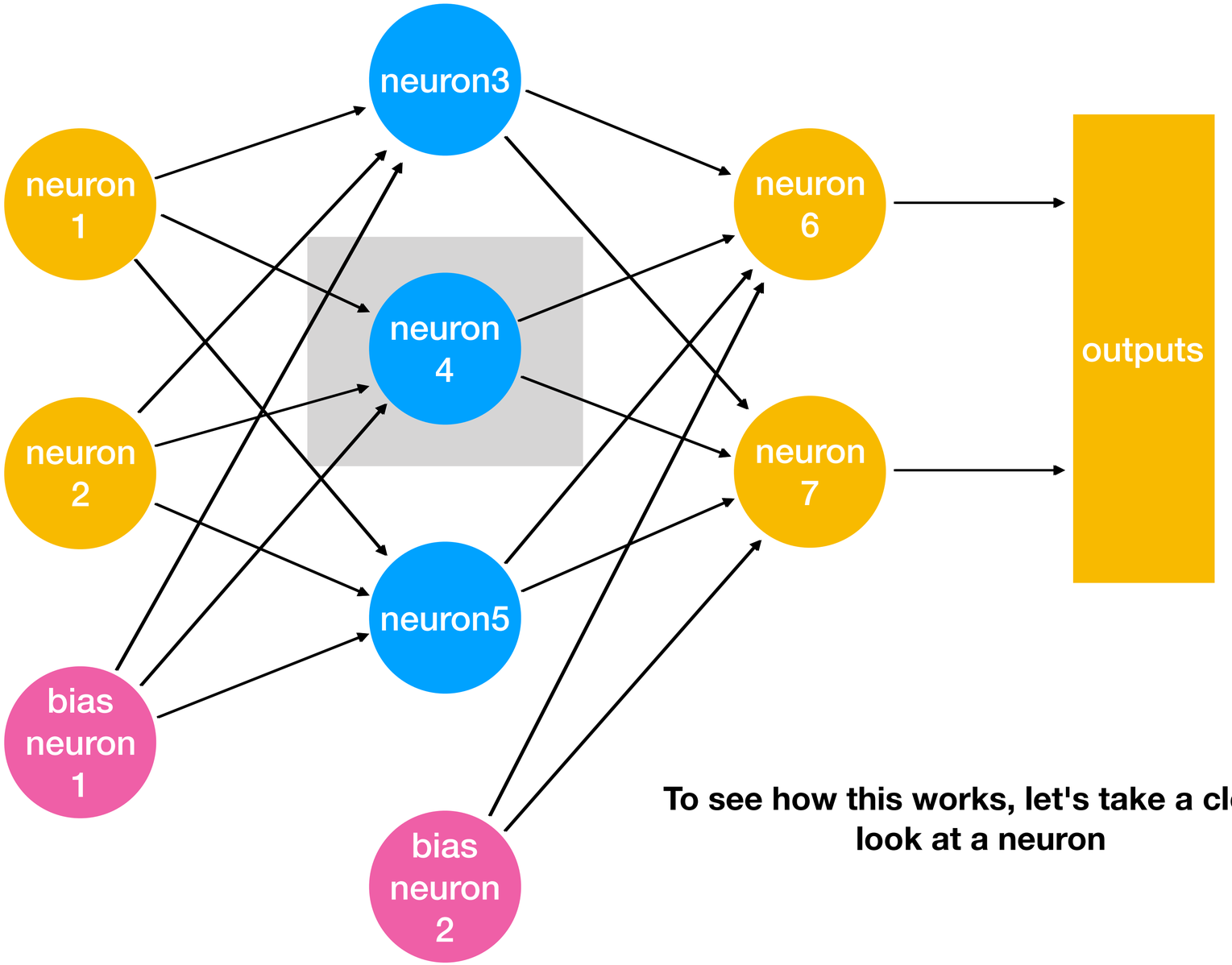
We set the neurons on the first (leftmost) layer to our input values



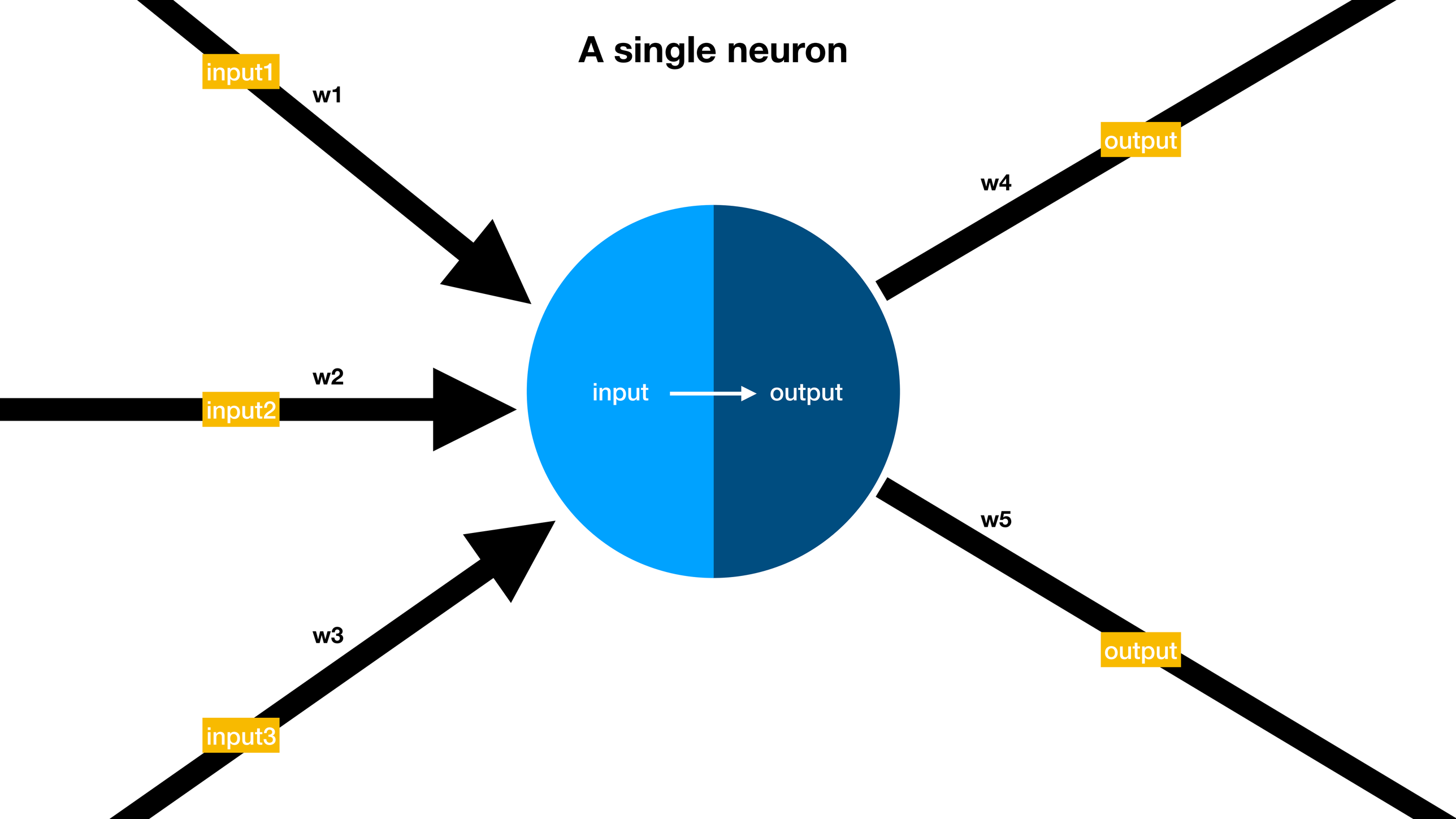
Let's peel away the black box…

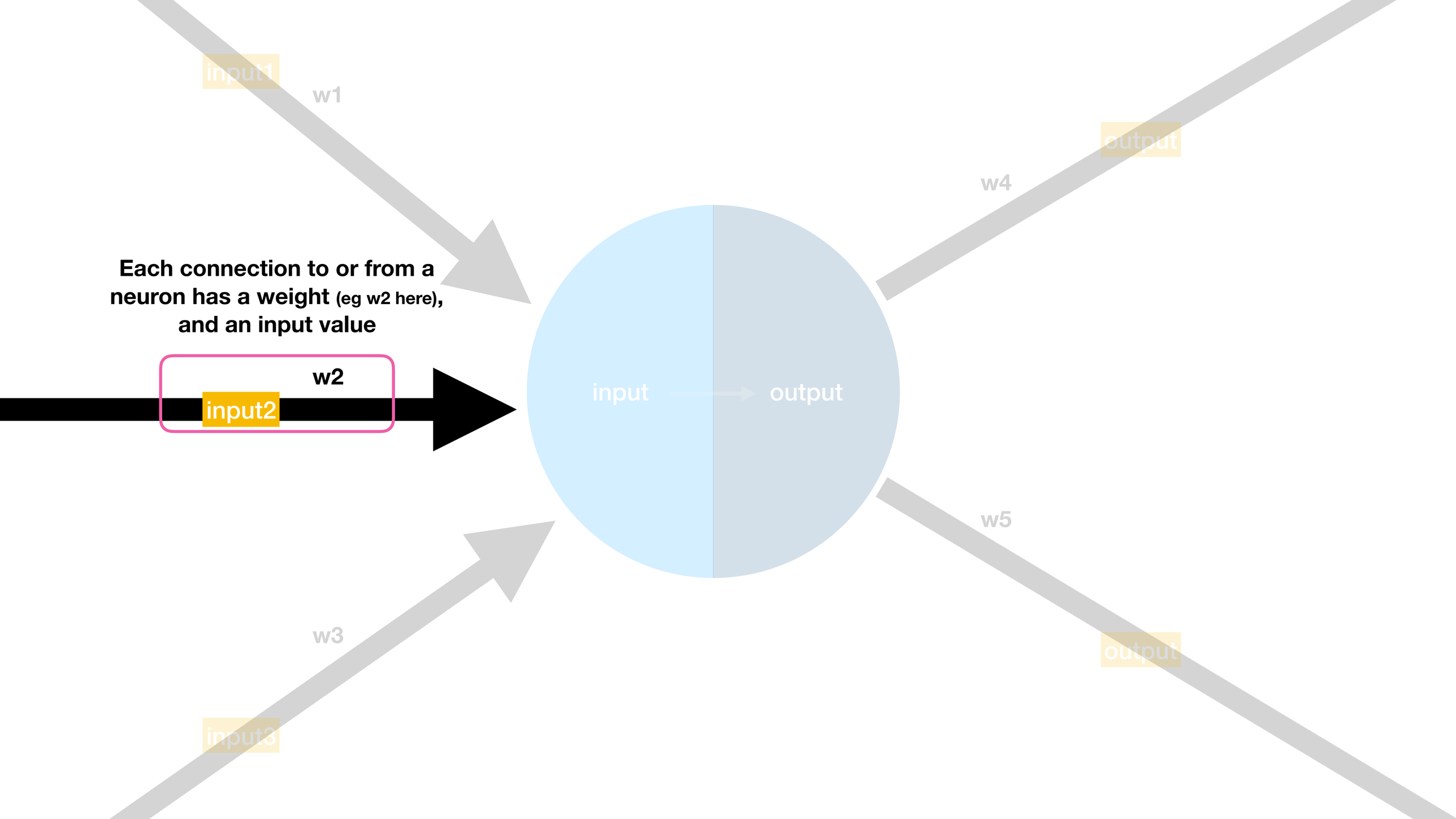


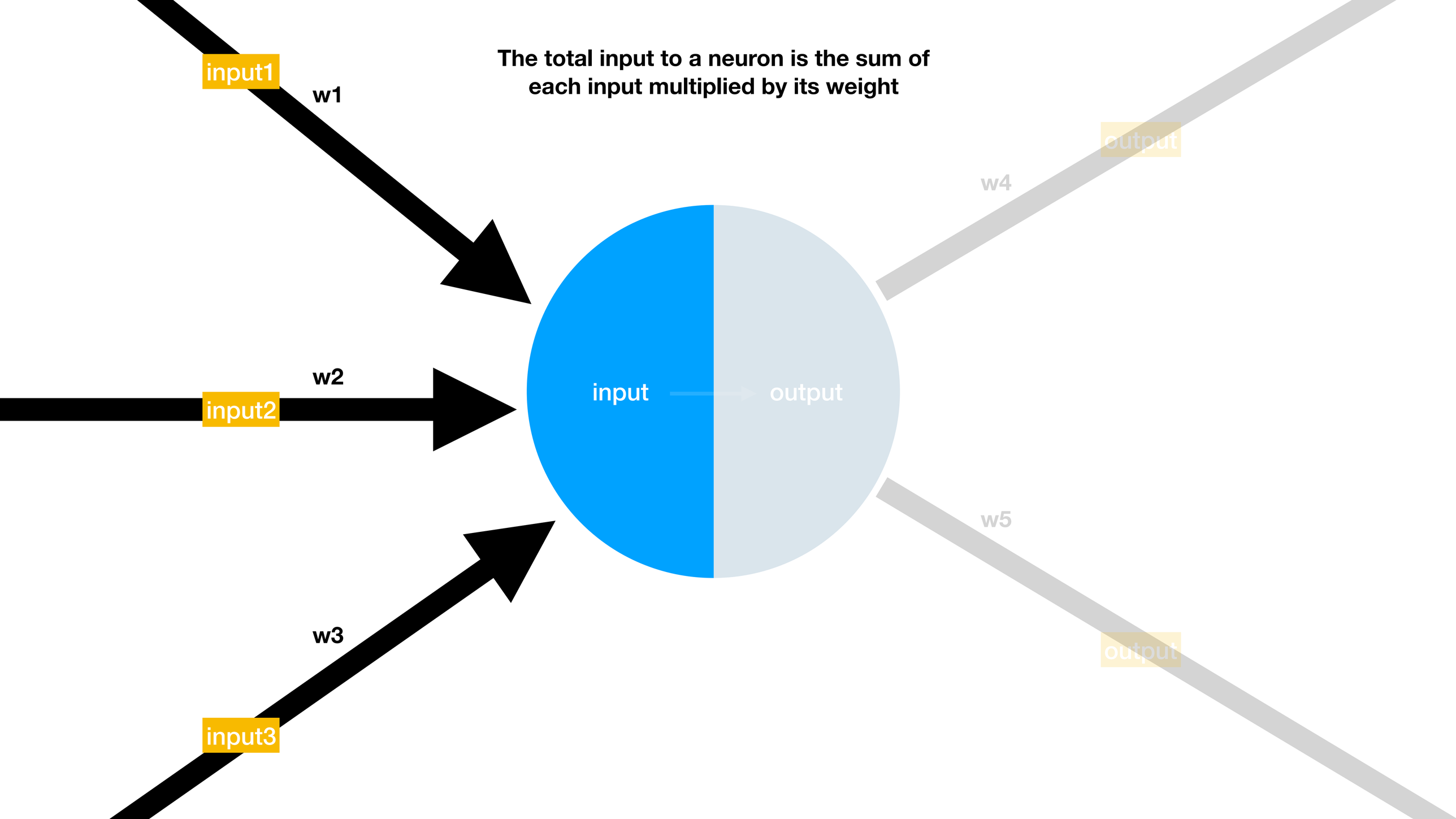
And eventually we get some outputs back from the neurons in the last layer

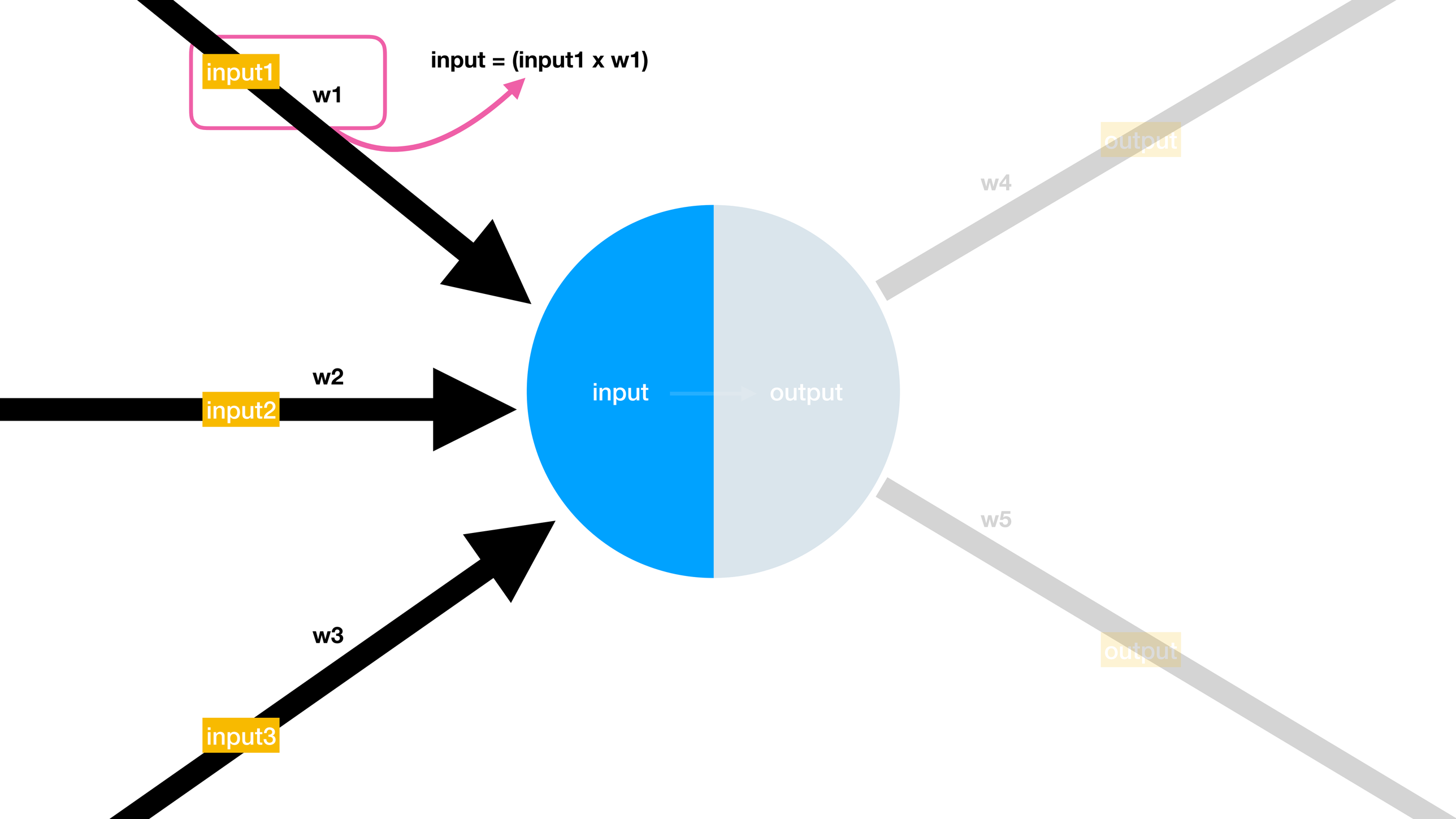


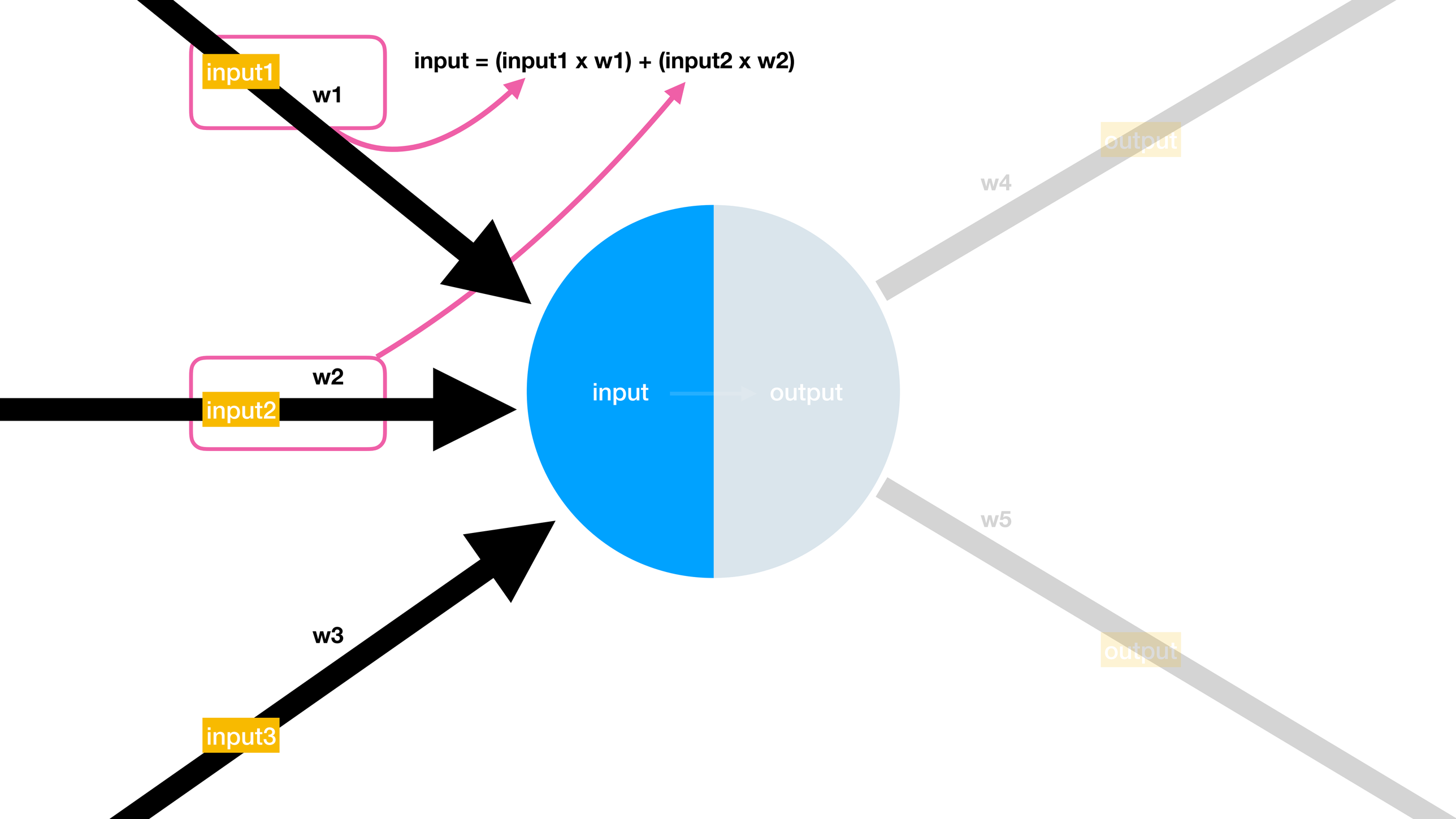
To see how this works, let's take a closer look at a neuron

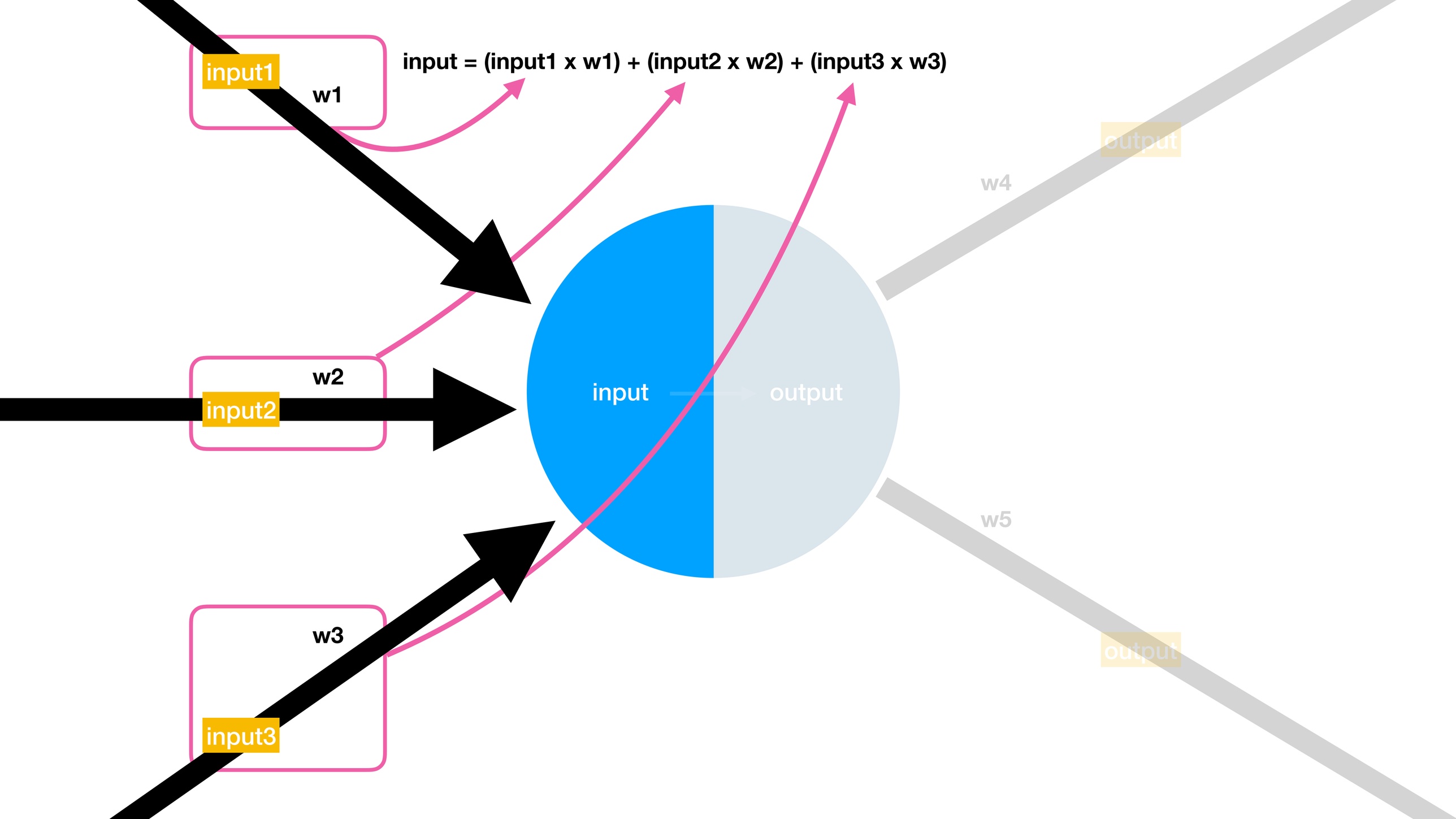

A single neuron
input1
w1
w2
input2
w3
input3
input
output
w4
output
w5
output

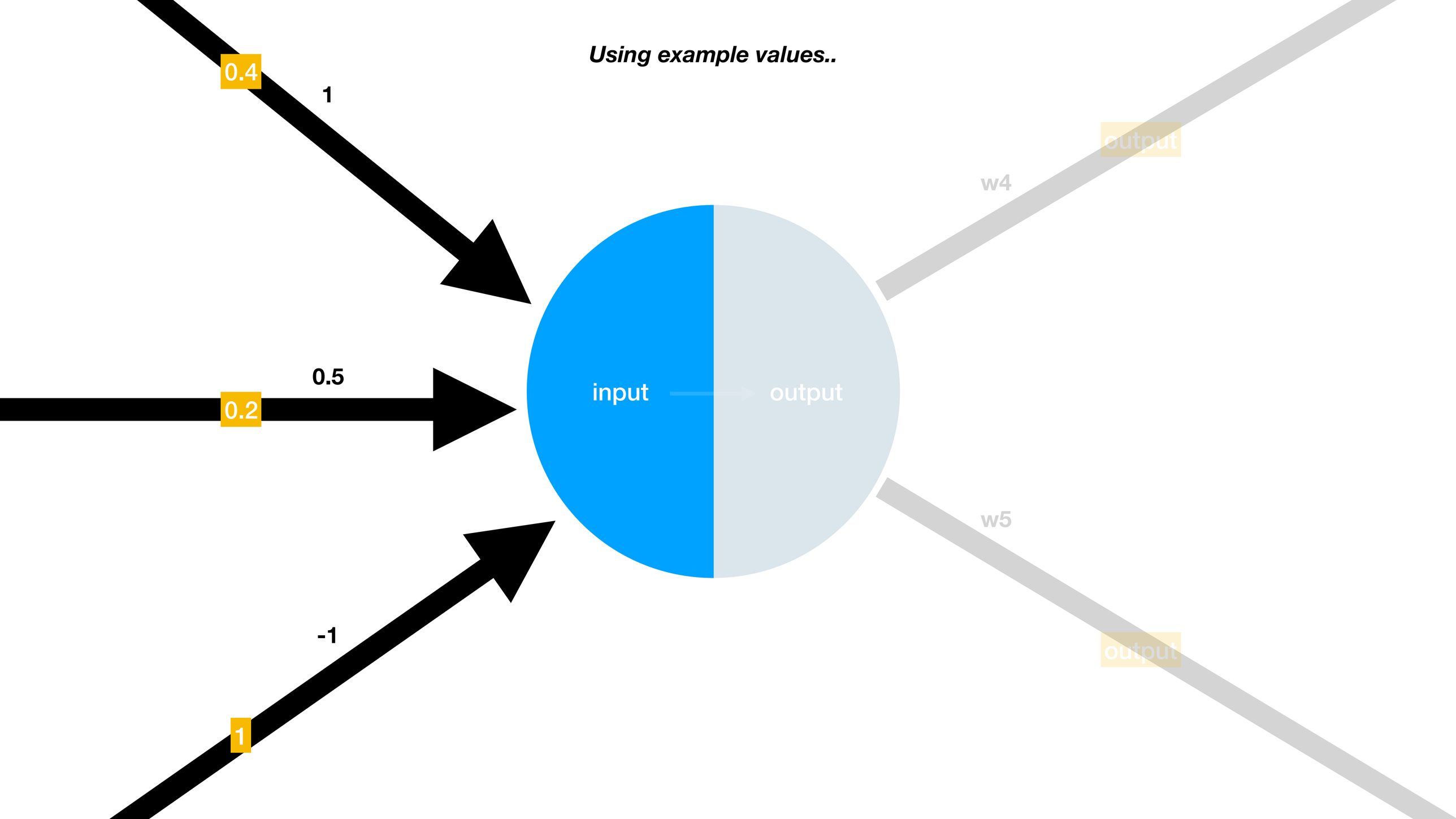

input1
w1
w4
output
Each connection to or from a neuron has a weight (eg w2 here), and an input value
w2
input2
input
output
w5
w3
output
input3

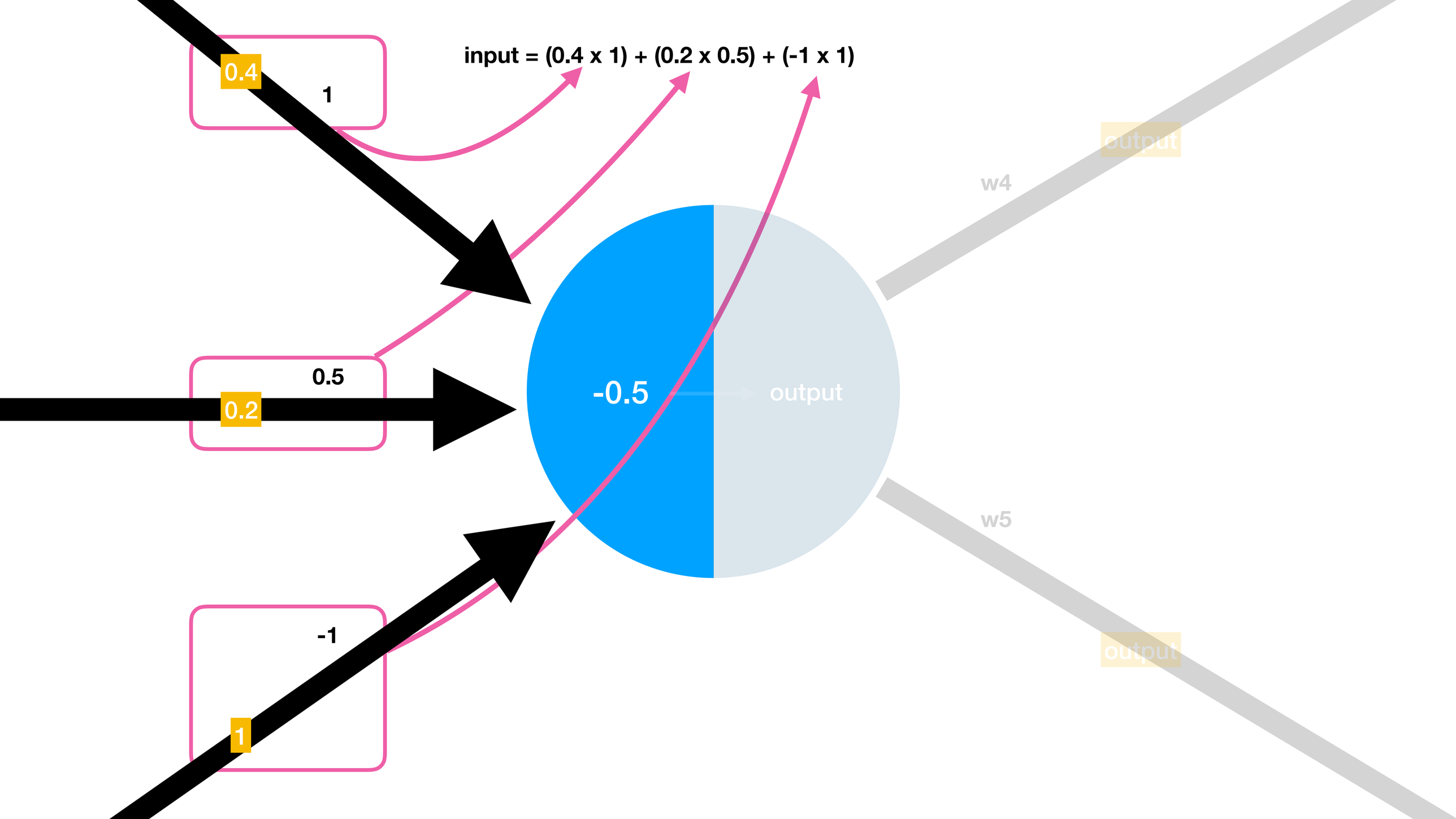

The total input to a neuron is the sum of each input multiplied by its weight
input1
w1
w2
input2
w3
input3
input
output
w4
output
w5
output

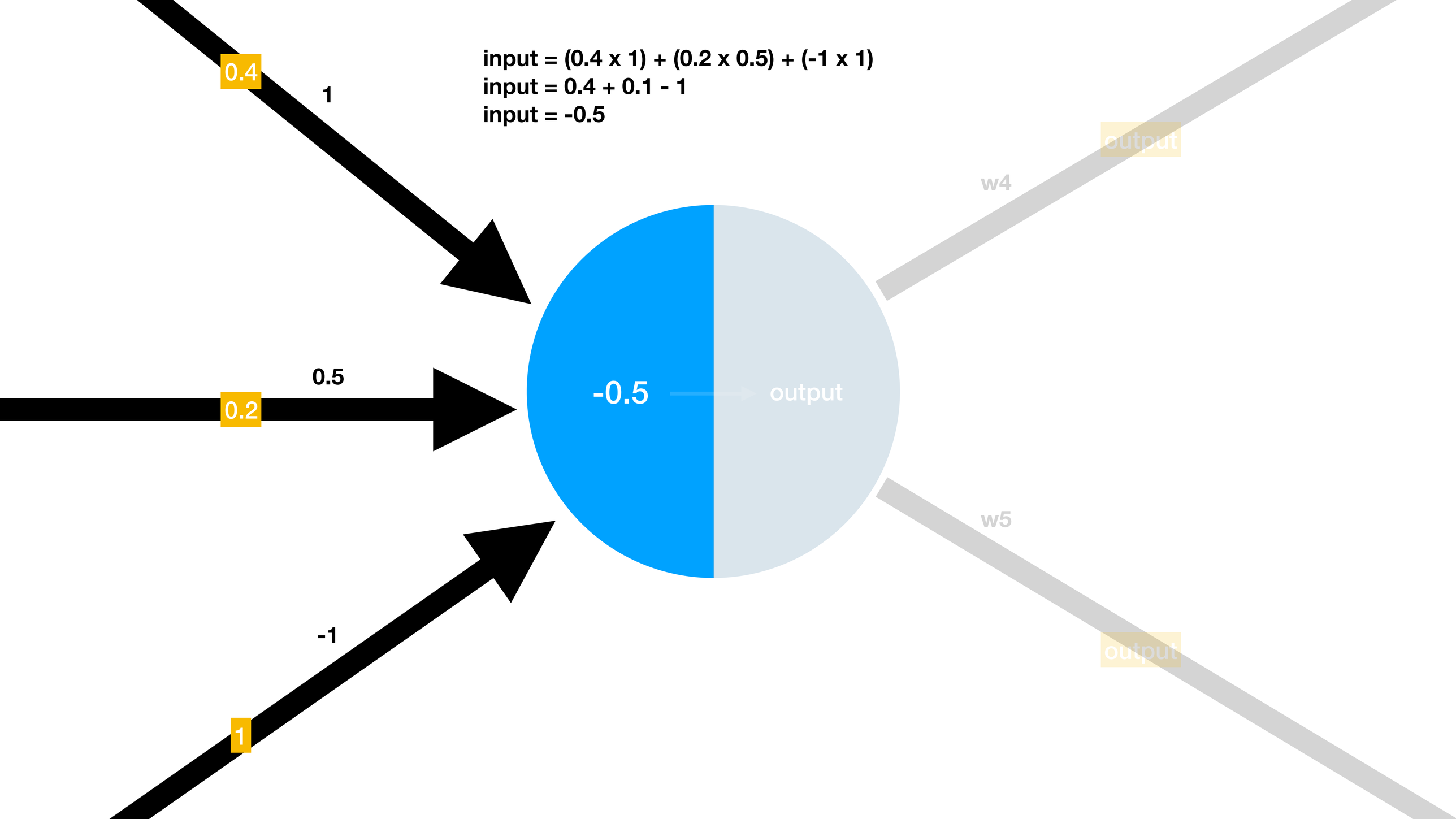

input1
w1
input = (input1 x w1)
input2
w2
input3
w3
input
output
w4
output
w5
output


input1
w1
input = (input1 x w1) + (input2 x w2)
output
w4
input
output
w2
input2
w5
output
w3
input3

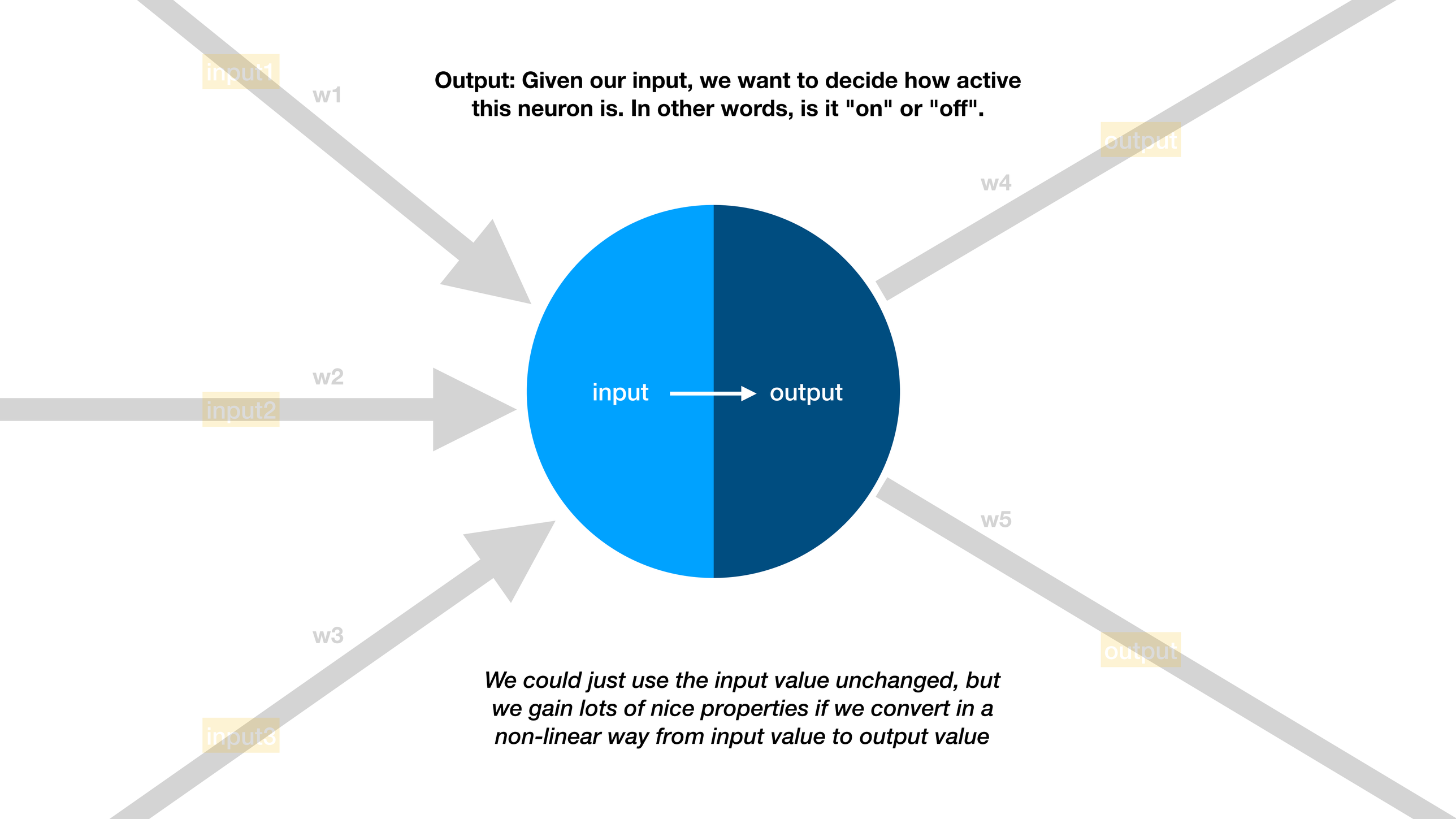

input1
w1
input = (input1 x w1) + (input2 x w2) + (input3 x w3)
output
w4
w2
input2
input
output
w5
w3
output
input3


Using example values..
0.4
1
0.5
0.2
-1
1
input
output
w4
w5
output
output


input = (0.4 x 1) + (0.2 x 0.5) + (-1 x 1)
0.4
1
0.5
0.2
-1
1
-0.5
output
w4
w5
output
output

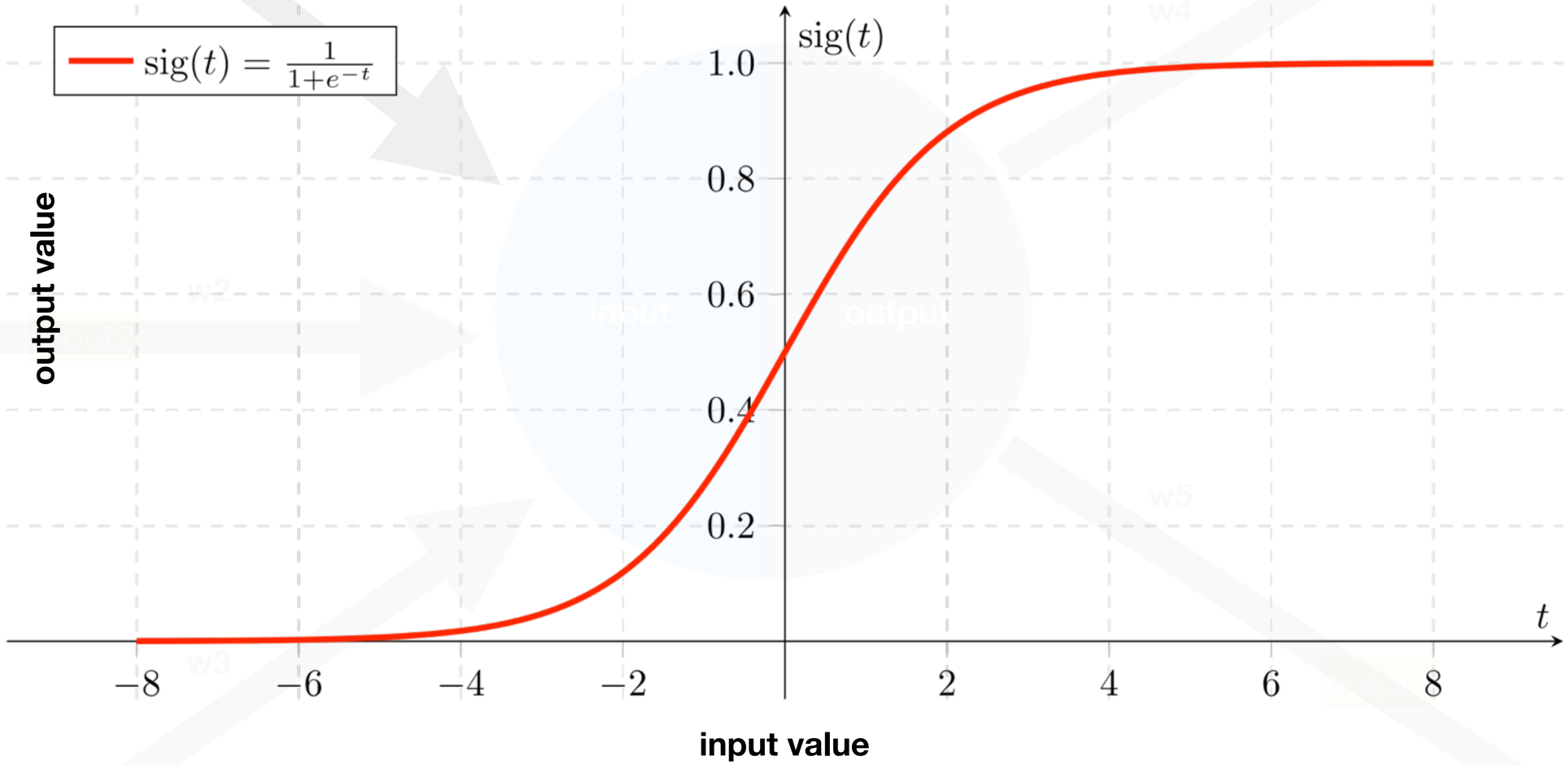

input = (0.4 x 1) + (0.2 x 0.5) + (-1 x 1)
input = 0.4 + 0.1 - 1
input = -0.5
0.4
1
0.5
0.2
-1
1
-0.5
output
w4
output
w5
output

Output: Given our input, we want to decide how active this neuron is. In other words, is it "on" or "off".

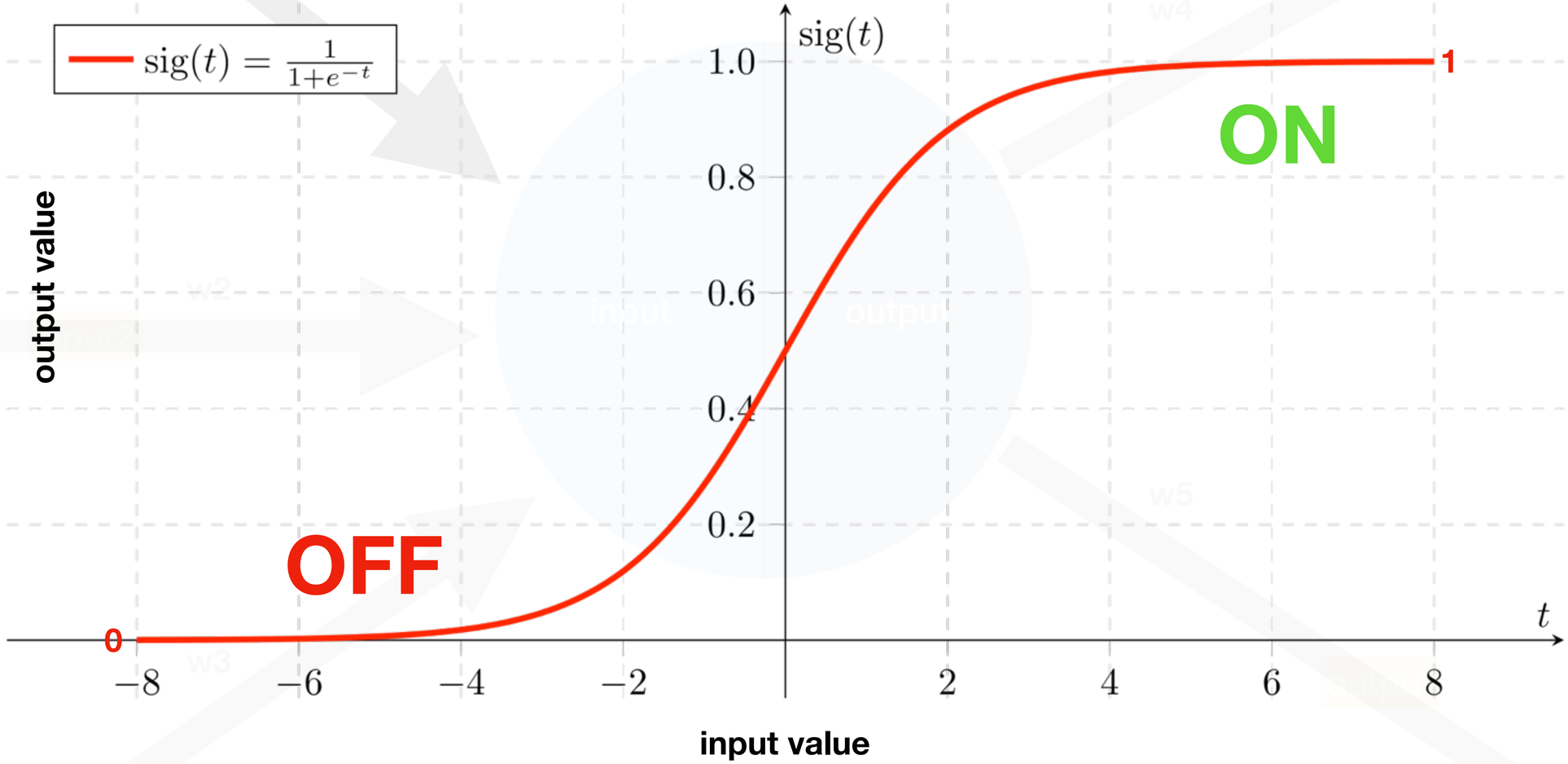


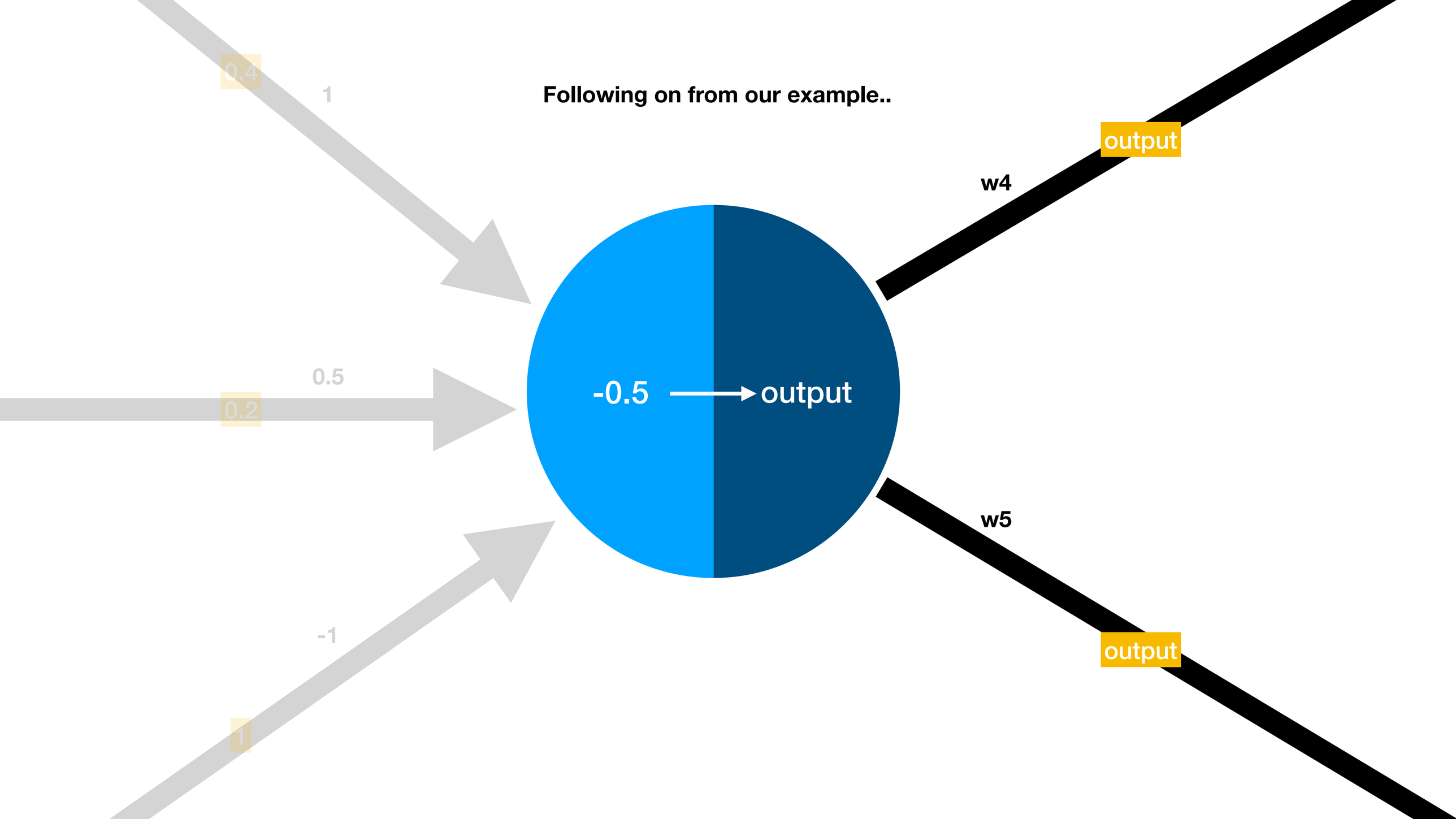
We could just use the input value unchanged, but we gain lots of nice properties if we convert in a non-linear way from input value to output value

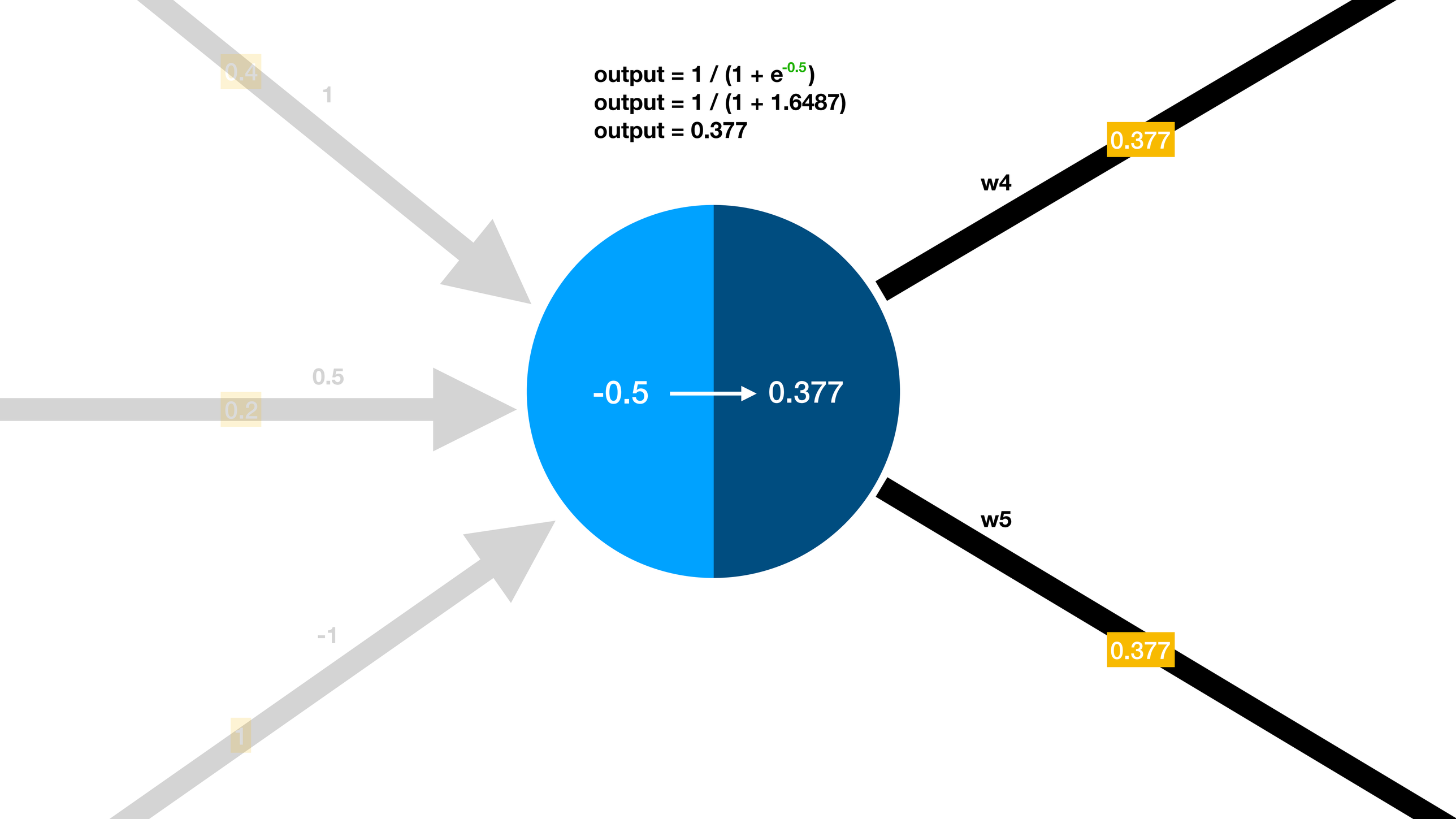
We call how we convert from input to output an *activation function.*
This is one possible choice (almost anything can be used in theory):



The activation function determines how "on" or "off" the neuron is, given the input to it

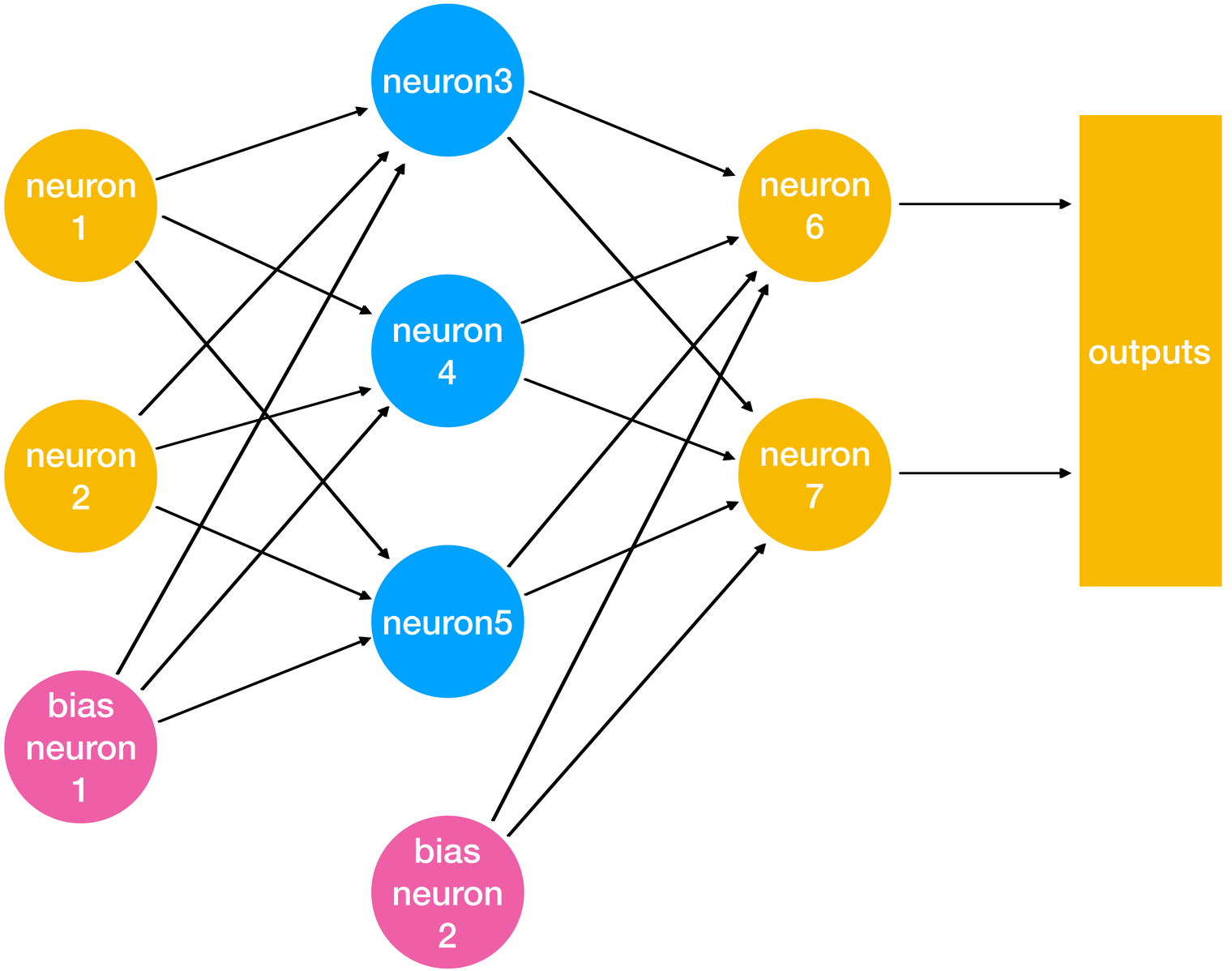



Following on from our example..
0.4
1
0.5
0.2
-1
1
-0.5
output
w4
w5
output
output

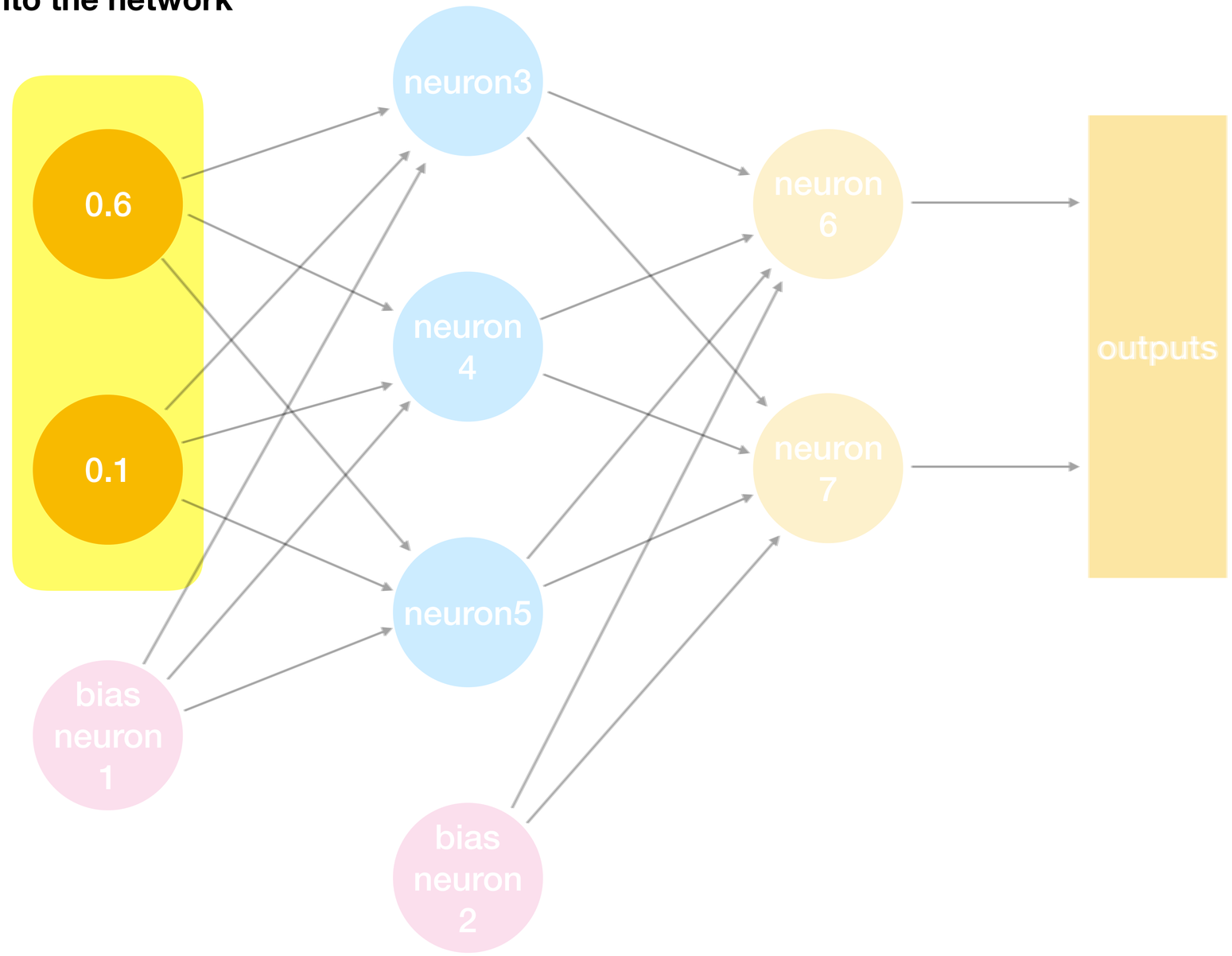

output = 1 / (1 + e^-0.5)
output = 1 / (1 + 1.6487)
output = 0.377
0.4
1
0.5
0.2
-1
1
-0.5
0.377
w4
w5
0.377
0.377

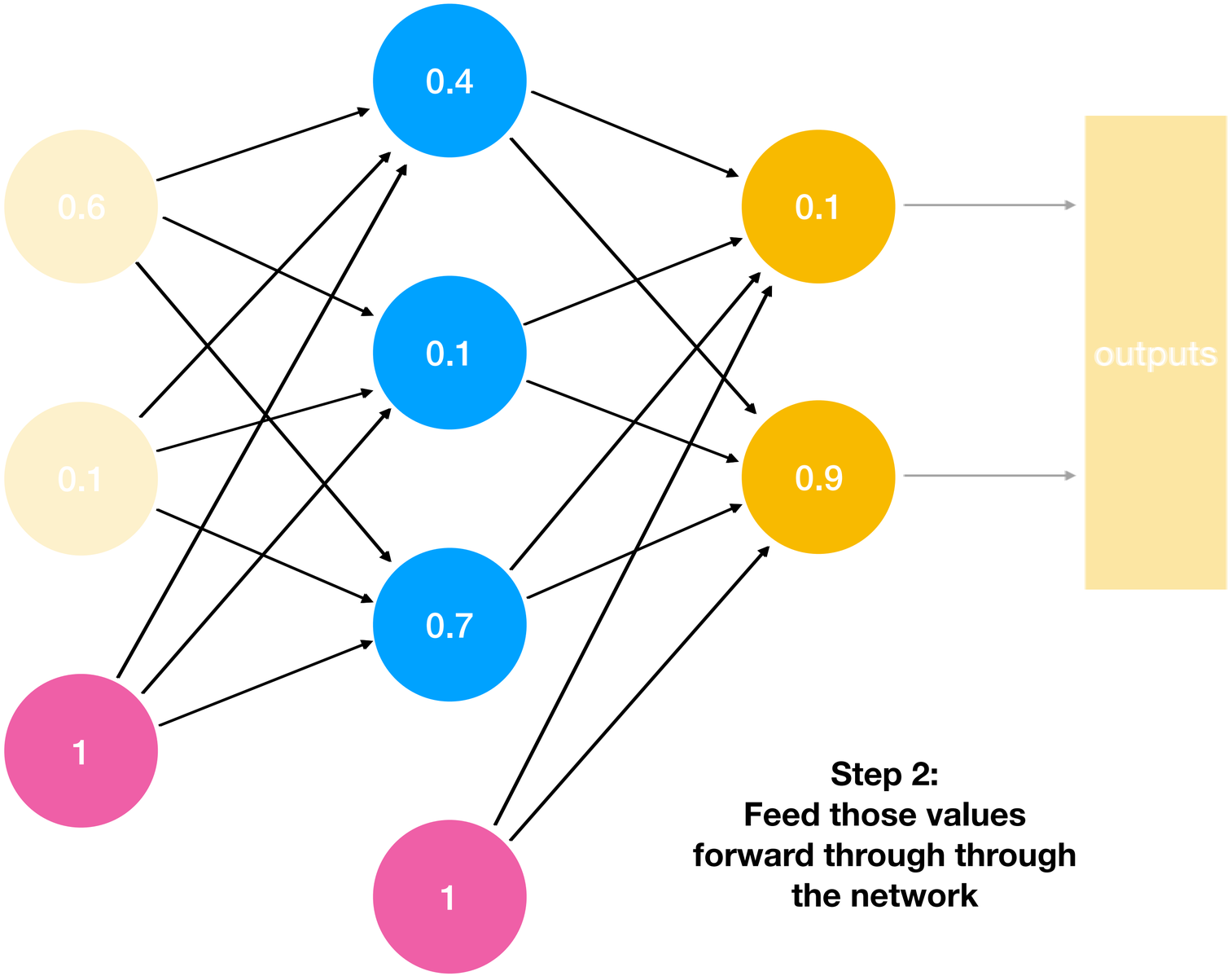
- A neuron takes a bunch of inputs from each of the ones before it
- Each input value is multiplied by the weight - the *strength* - of that connection
- We add these together to get the input to this neuron
- We apply an activation function to determine how "on" or "off" the neuron is given this input

Back to the bigger picture..

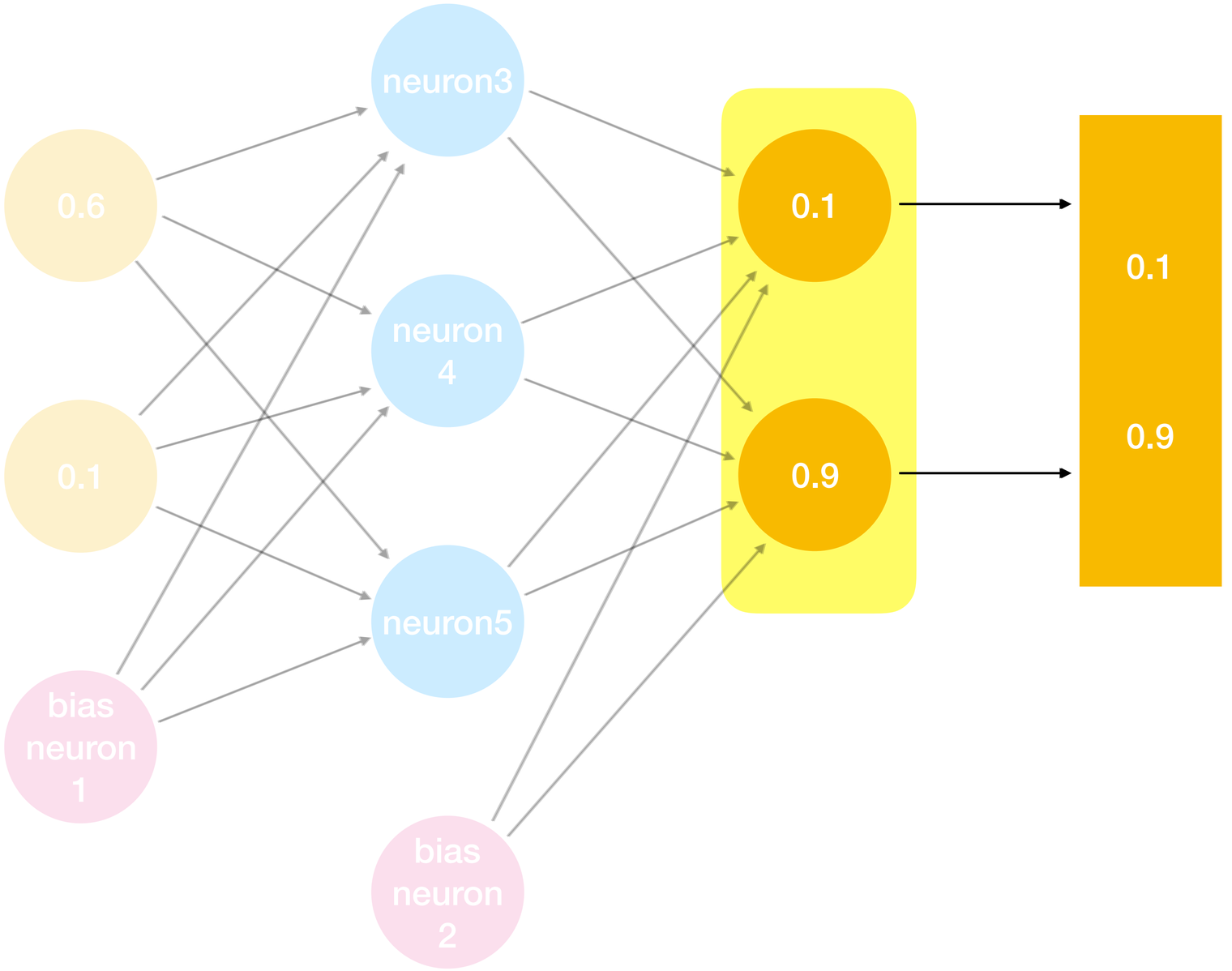


Step 1:
Put some values into the network



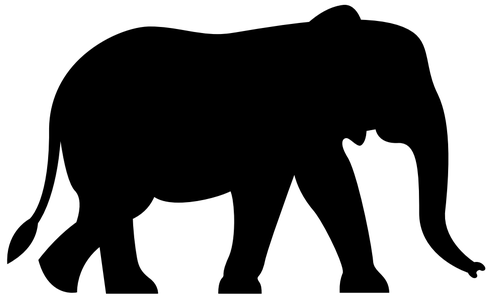


Step 2:
Feed those values forward through through the network



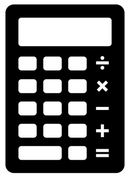
Step 3:
Take the output from the network!

Training the network



The Goal

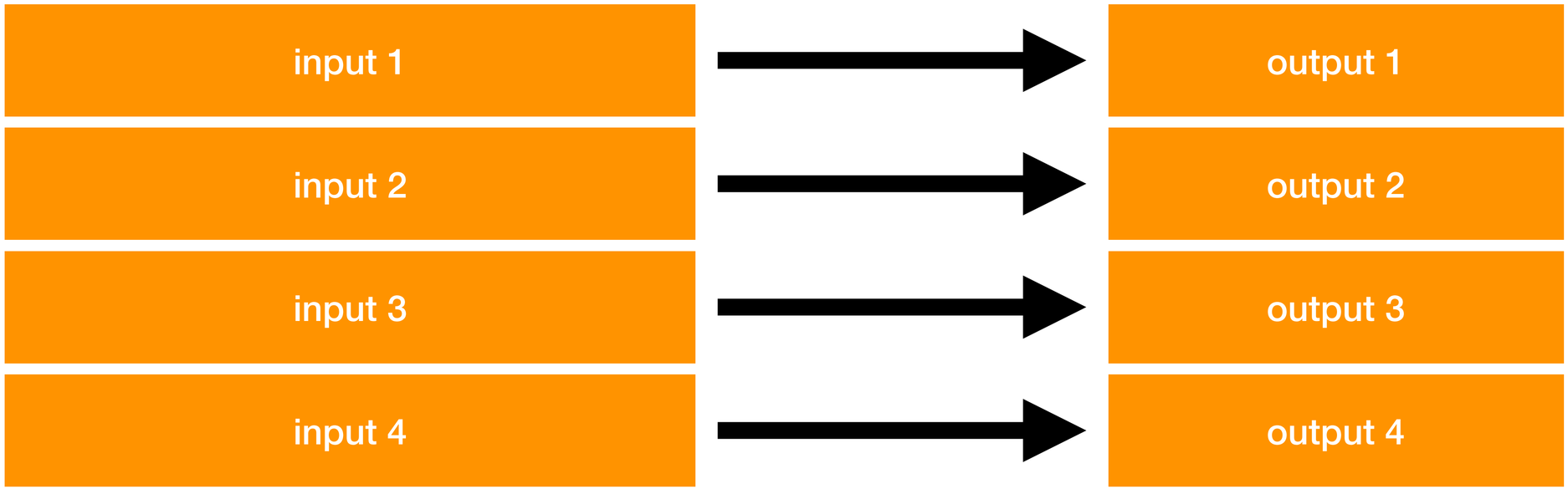
Given some input, we want to update each of the *weights* that connect neurons to each other *a wee bit* in order to reduce the difference between what we want the output to be, and what it is.



What we need

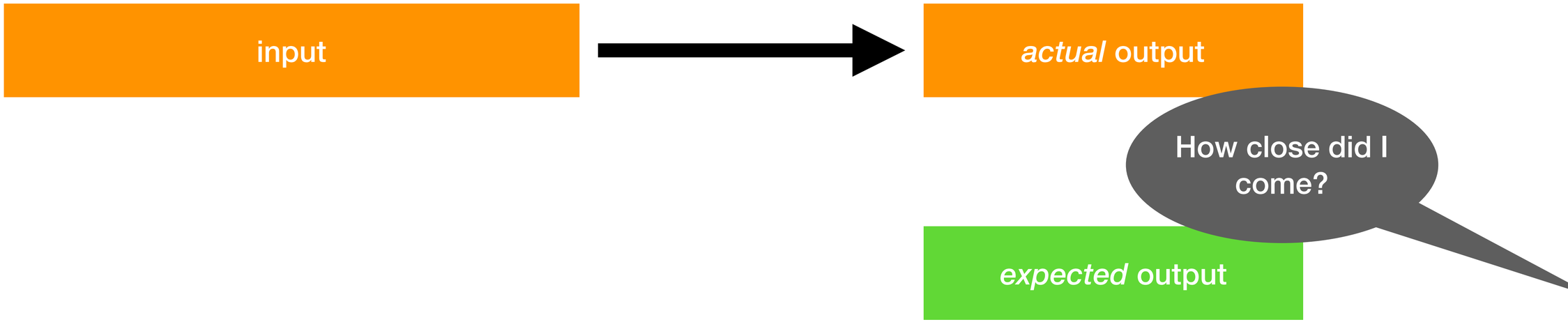
What we need

1. **A *training set*. That is, a set of inputs to the network, and the output we want back for each one.**



What we need

2. **A way to calculate the error given some *expected* output values and some *actual* output values**



What we need

2. **A way to calculate the error given some *expected* output values and some *actual* output values. This is one possibility:**

$$error = \sum \frac{1}{2}(expected - actual)^2$$

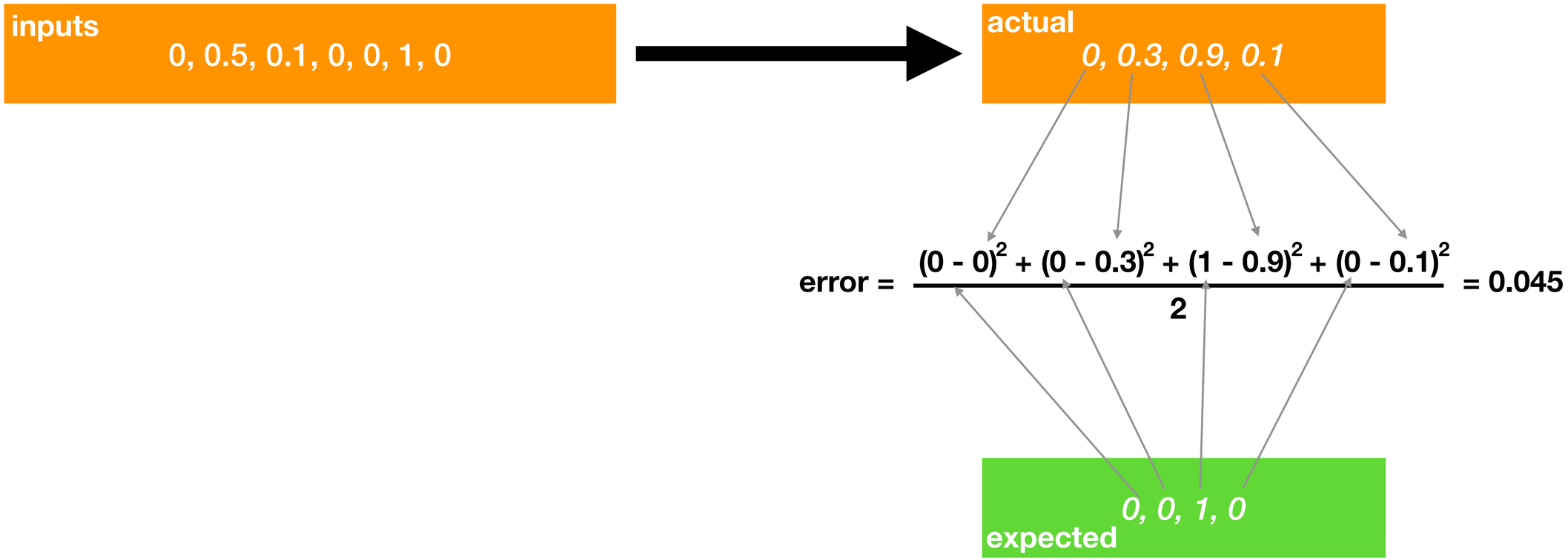
This roughly says "find the difference between each actual output value and the one we expected, and square it, then add each of these together to get our final total"

Squaring the difference means that larger differences are treated more aggressively than small ones. As the difference gets smaller, we work much less to change it. It also means that the best output value to converge on if there are 2 possibilities is one in the middle of those, since that has the smallest average error with a simple abs() function, arr values between the two would be equally appealing.

What we need

2. **A way to calculate the error given some *expected* output values and some *actual* output values. This is one possibility:**

$$error = \sum \frac{1}{2}(expected - actual)^2$$

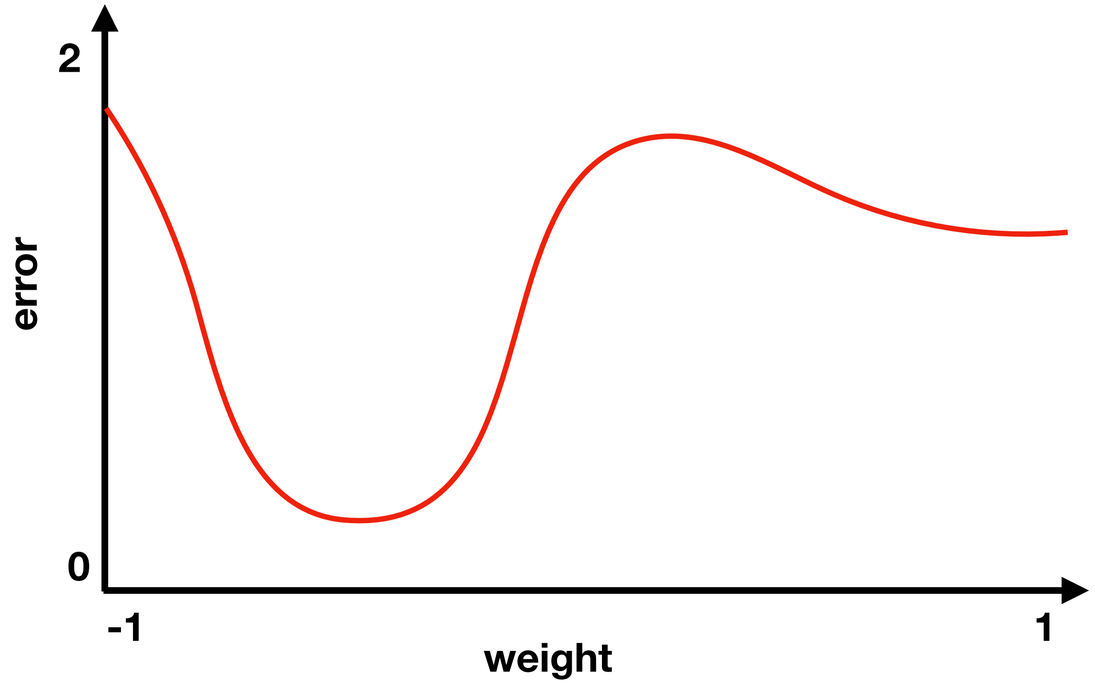


What we need

3. **A way to figure out how to update each weight in order to reduce this error. The method we'll use here is called *back propagation*.**

Back Propagation

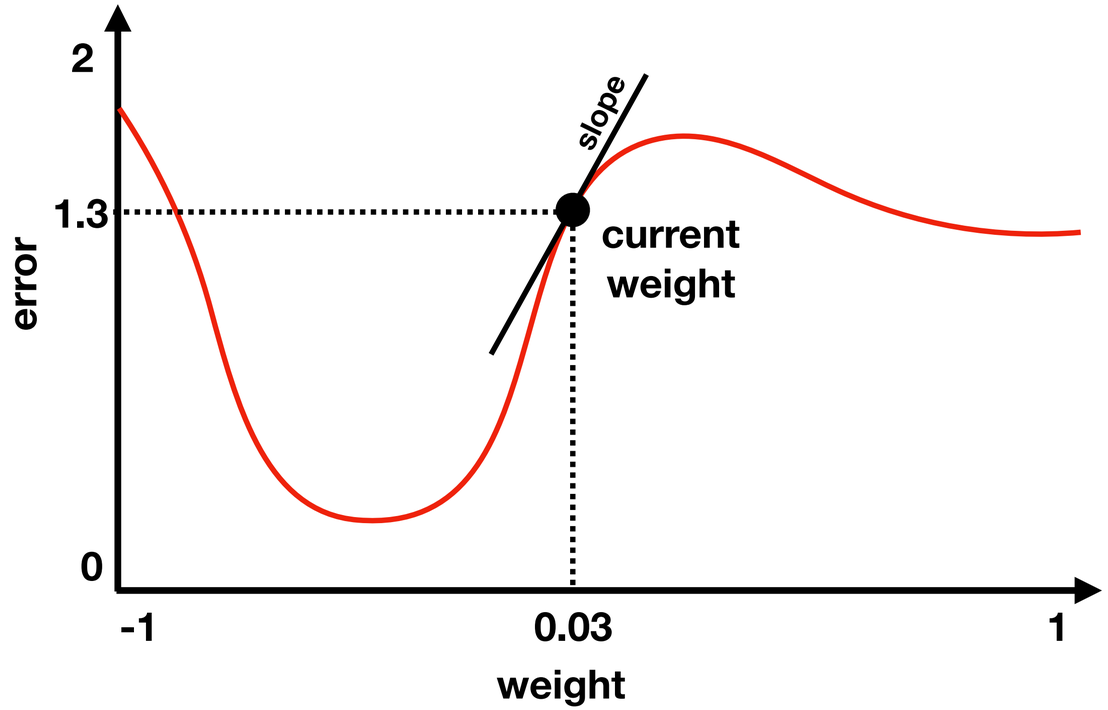
Given a training sample, for each weight we want to know how it changes with respect to the error between actual and expected output



Back Propagation

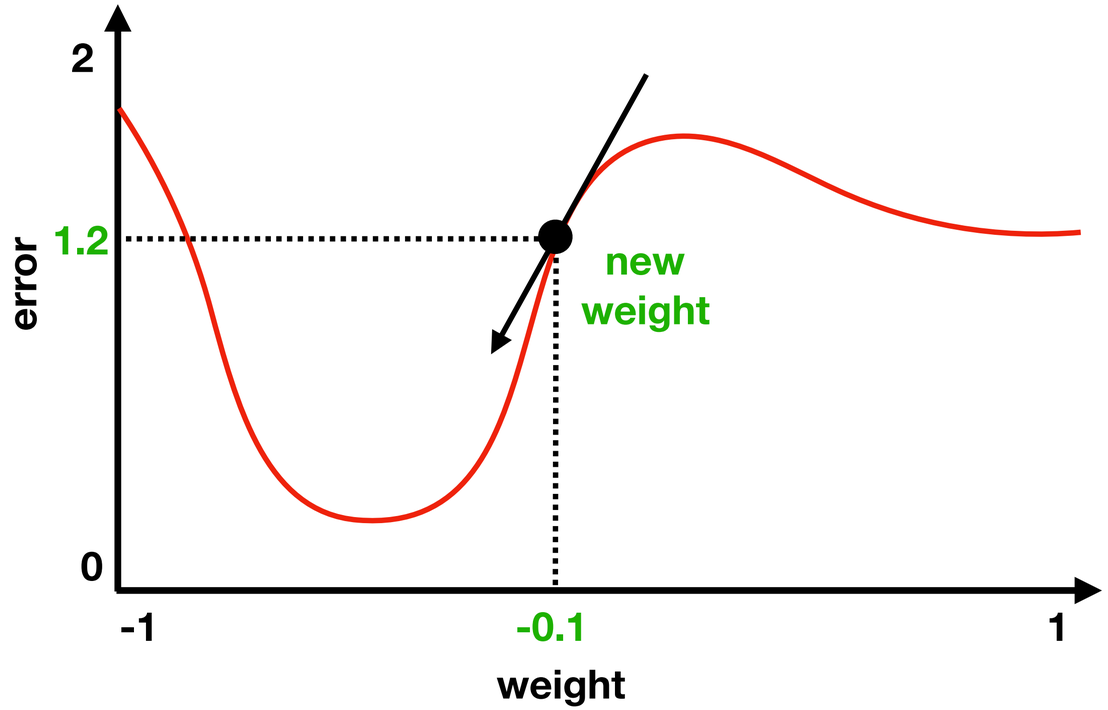
Given the current value of a weight, we can work out what the slope is at that point

(we differentiate the error with respect to one of our weights)

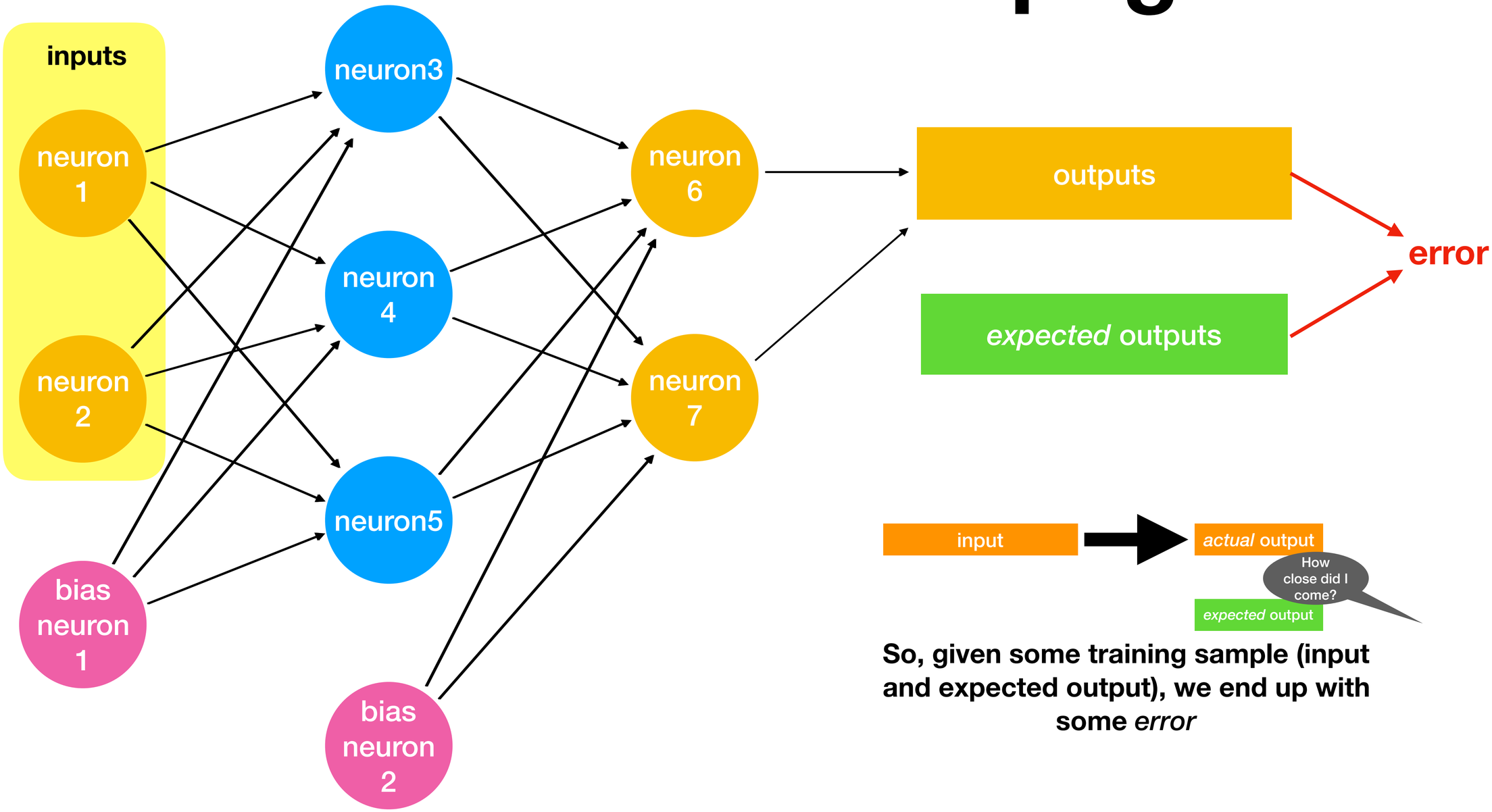


Back Propagation

And based on this, we can decide what the new weight should be by moving a little in the direction that reduces our error

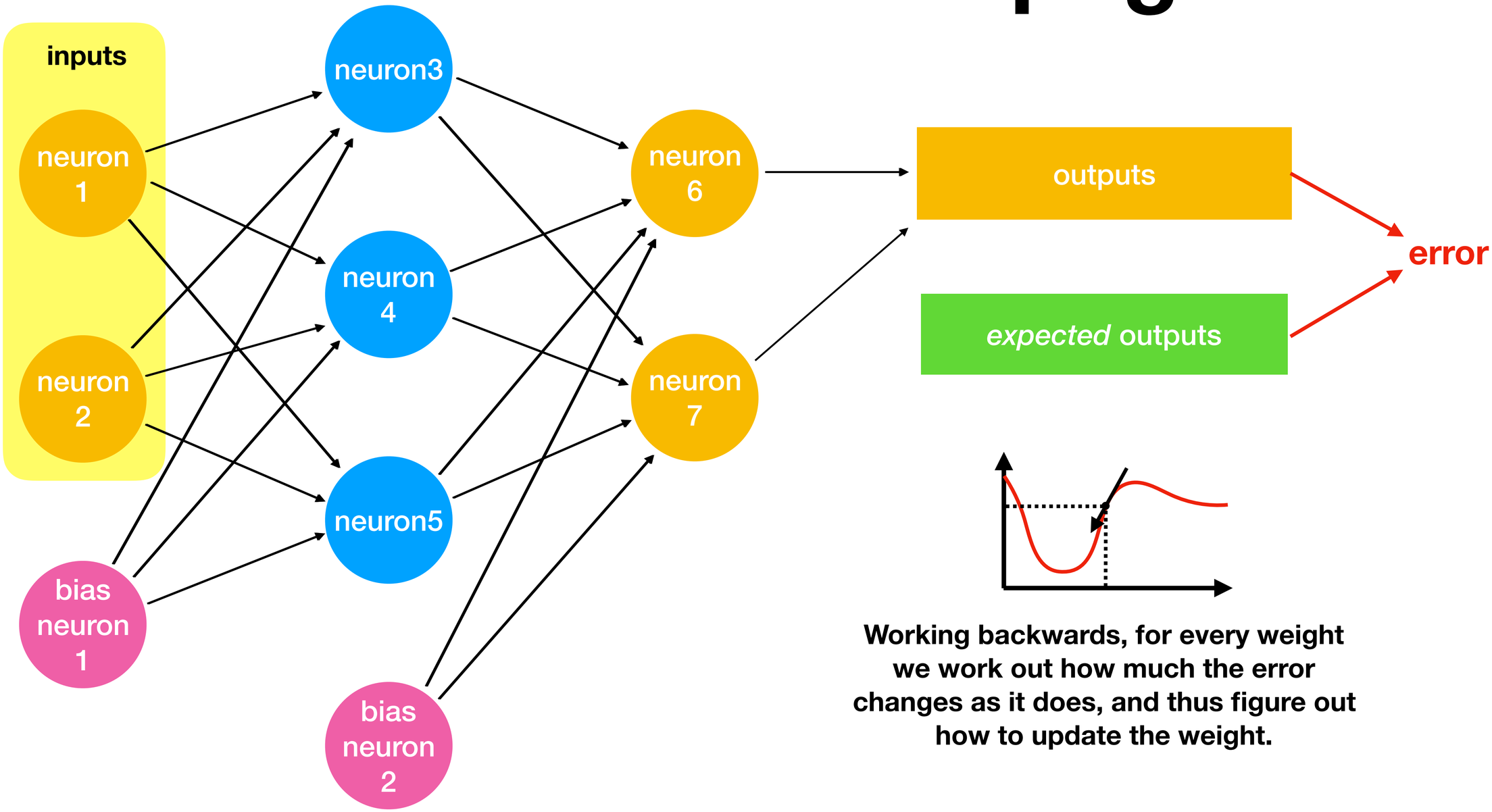


Back Propagation



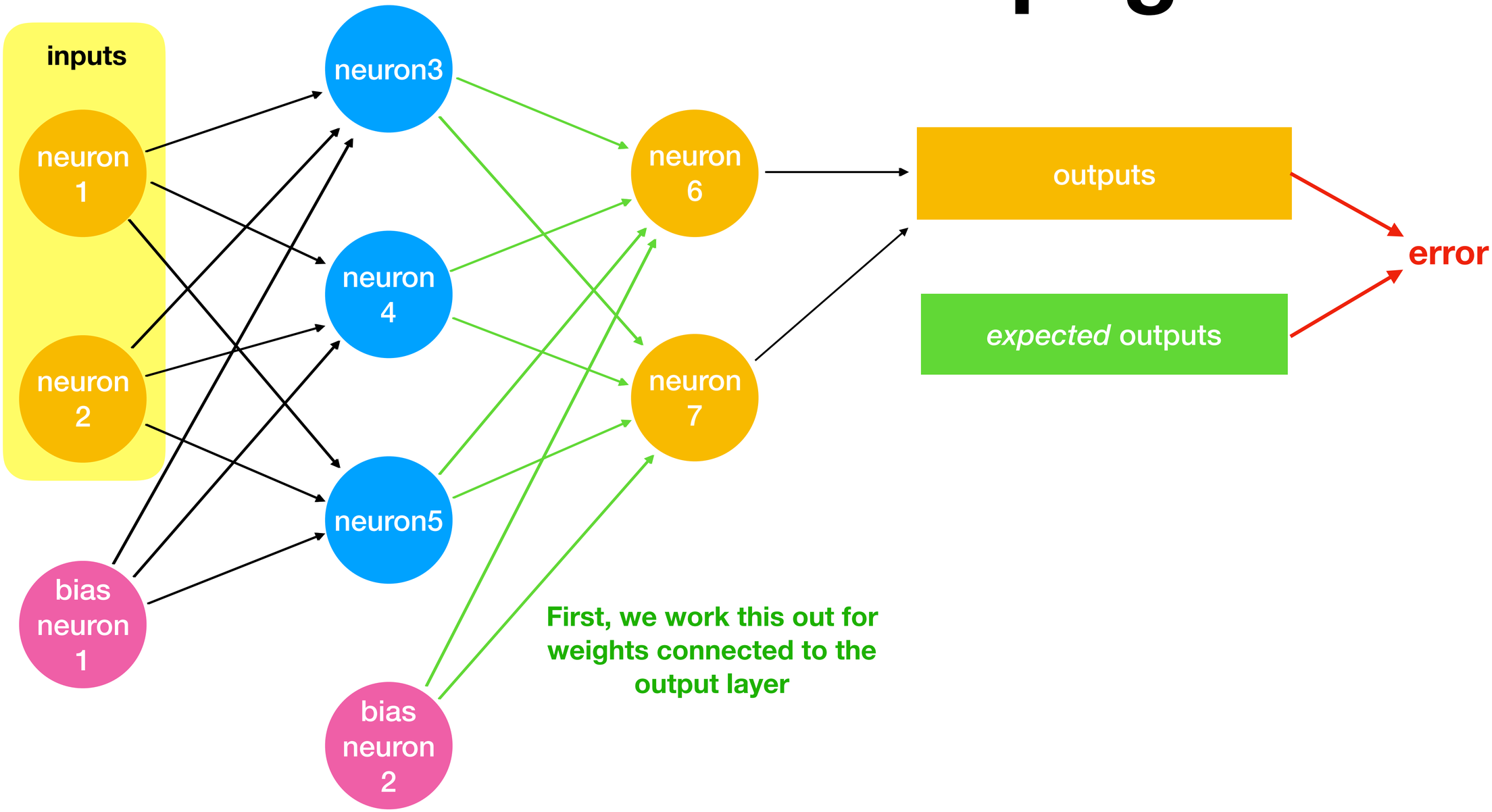
So, given some training sample (input and expected output), we end up with some *error*

Back Propagation

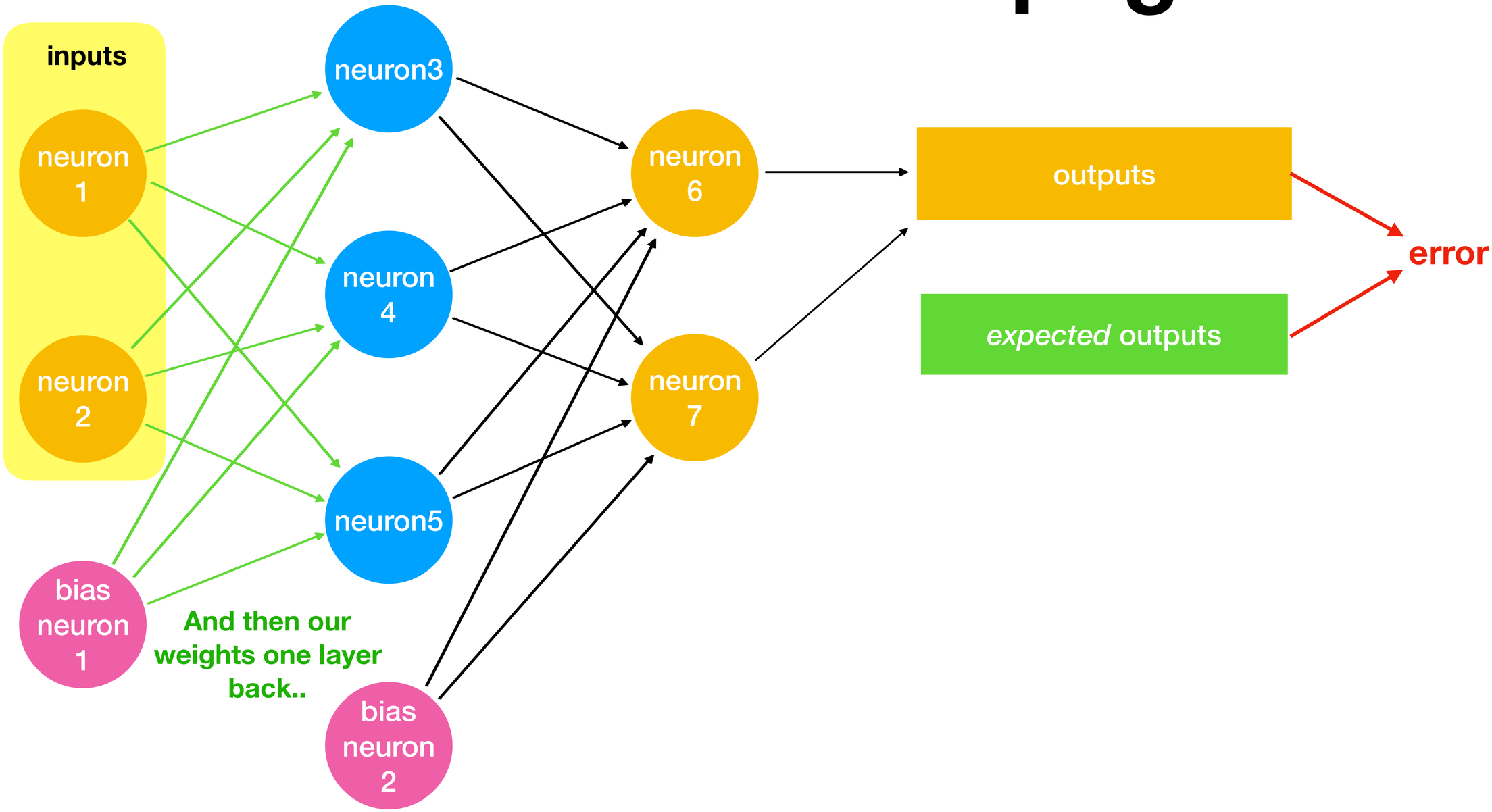


Working backwards, for every weight we work out how much the error changes as it does, and thus figure out how to update the weight.

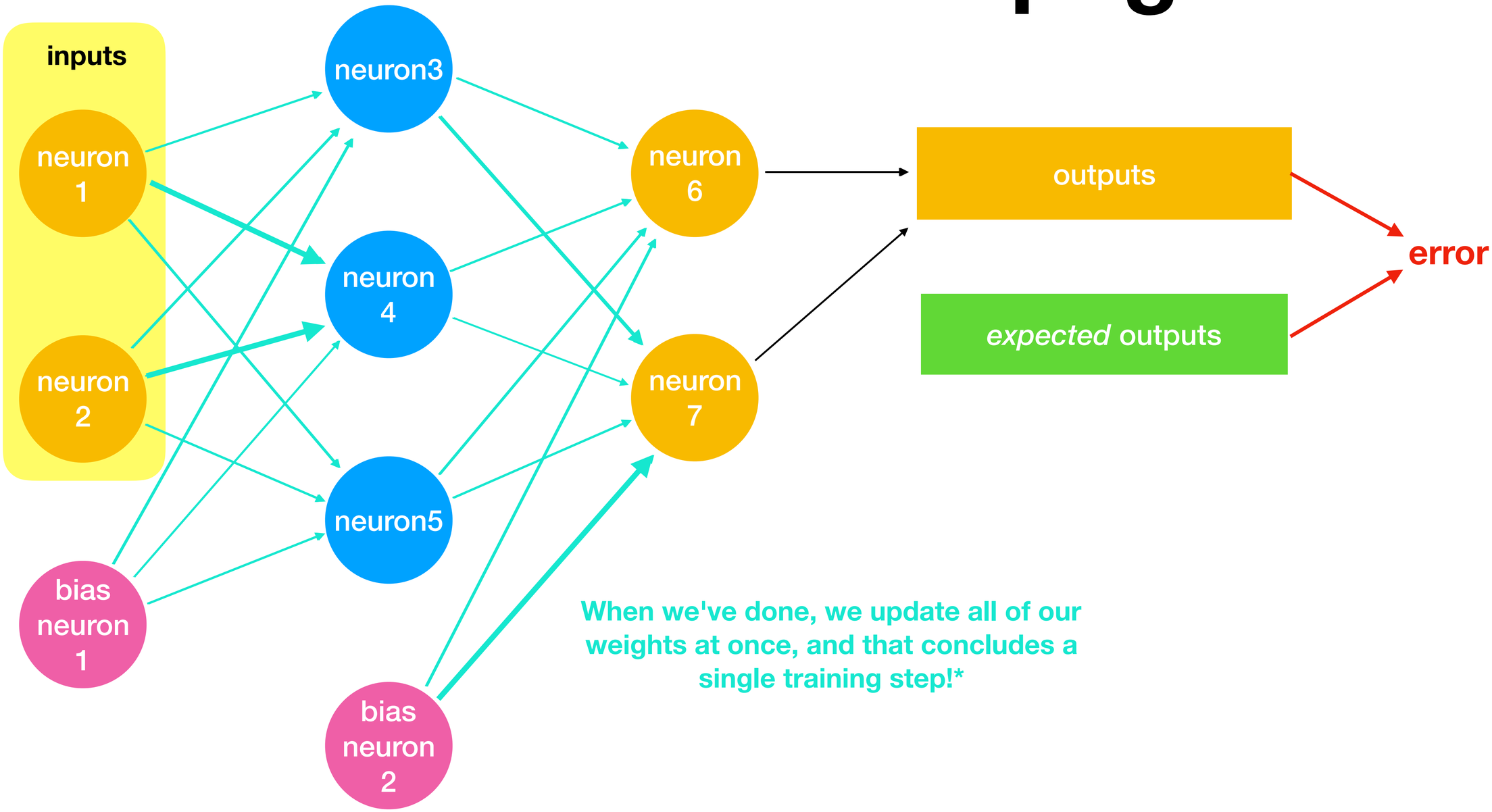
Back Propagation



Back Propagation

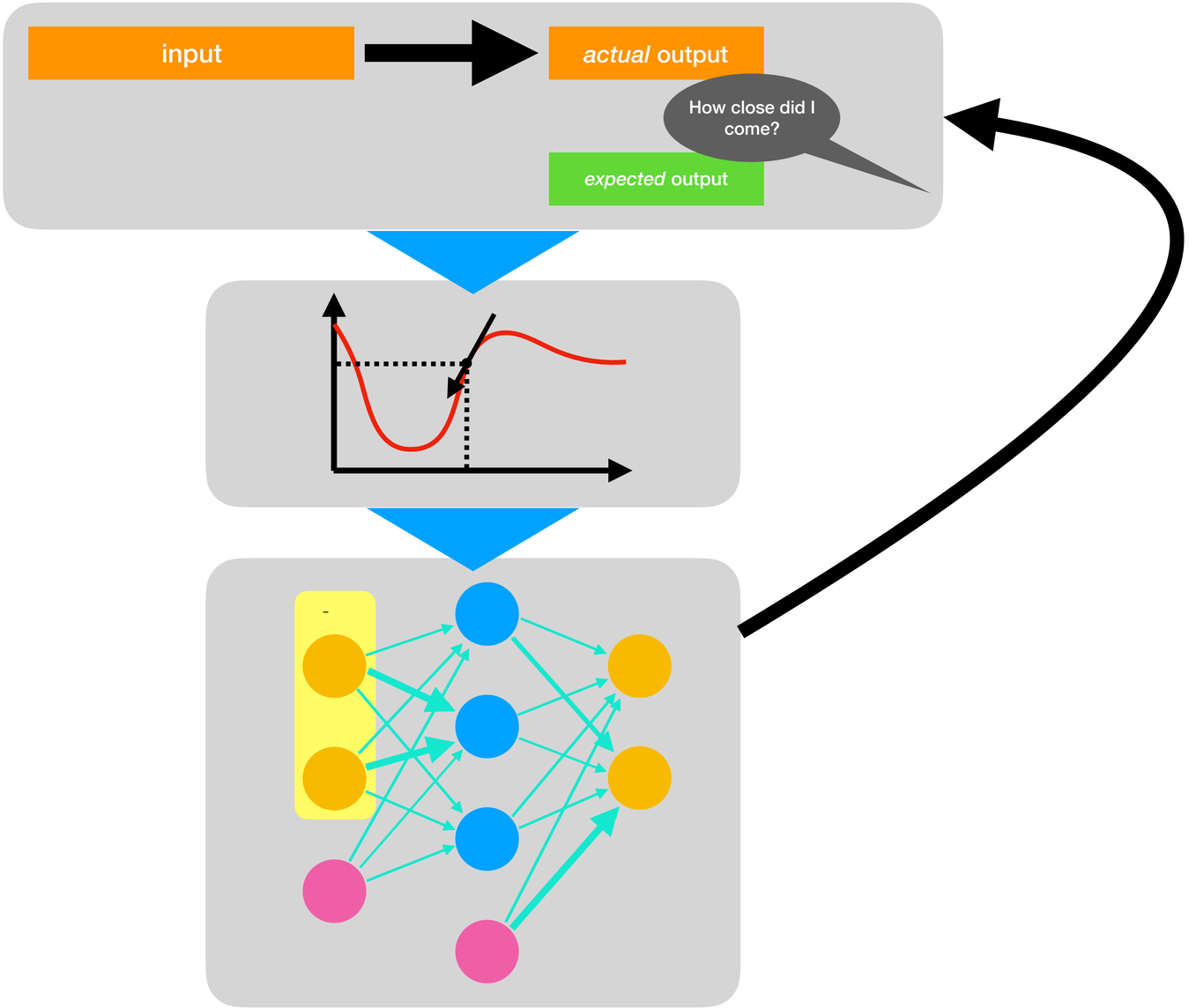


Back Propagation



When we've done, we update all of our weights at once, and that concludes a single training step!*

Training



To train the network, you simply repeat this process over and over with each of the input-output pairs in your training set until the network stops improving

Demo

http://playground.tensorflow.org

“Deep Learning”

Is based on the exact same principles described.

The End

https://jsdw.me/posts/neural-nets
https://github.com/jsdw/neural-net-example