M5MS10
Machine Learning

Spring 2018

Lecture 10

Dr Ben Calderhead

b.calderhead@imperial.ac.uk

M5MS10
Machine Learning

Spring 2018

Lecture 10

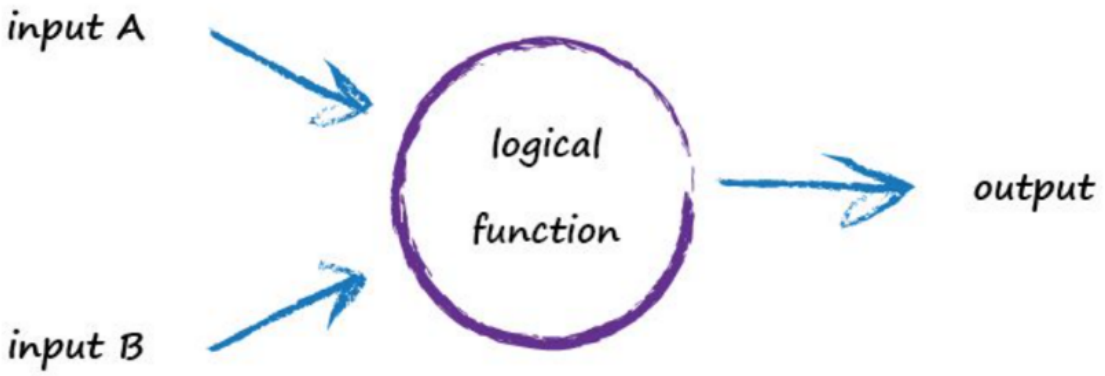
Dr Ben Calderhead

b.calderhead@imperial.ac.uk

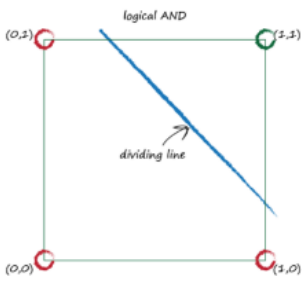
Sometimes One Classifier Is Not Enough

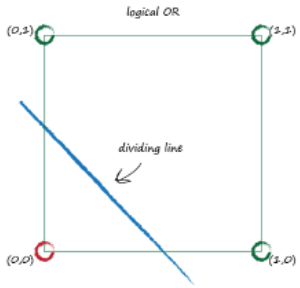
We've seen a few approaches to classification problems, along with the kernel trick for dealing with awkward data that may not otherwise be linearly classified.

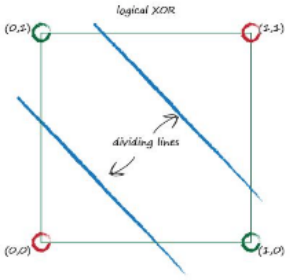
In fact we can construct a very simple function that defies linear classification.



Main idea: Construct a model that combines linear classifiers!





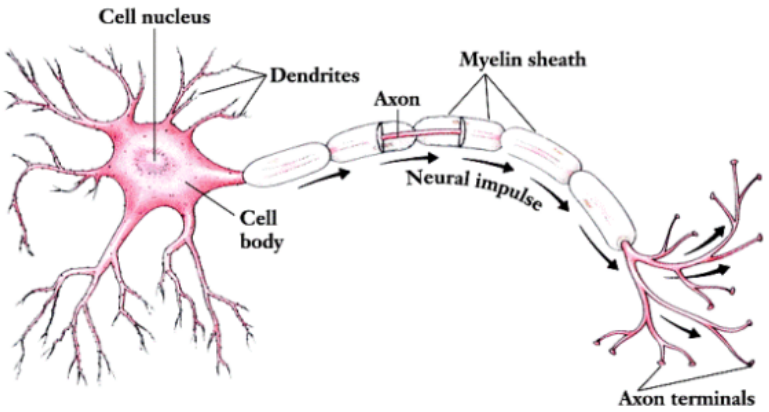


Neurons - Nature's Computing Machines

Neurons, although there are various forms of them, all transmit an electrical signal from one end to the other, from the dendrites along the axons to the terminals.

These signals are then passed from one neuron to another - this is how your body senses light, sound, touch pressure, heat and so on.

Signals from specialised sensory neurons are transmitted along your nervous system to your brain, which itself is mostly made of neurons too!



How Many Neurons Do We Need?

How many neurons do we need to perform interesting, more complex, tasks?

The human brain has about 100 billion neurons! A fruit fly has about 100,000 neurons and is capable of flying, feeding, evading danger, finding food, and many more fairly complex tasks.

100,000 neurons is well within the realm of modern computers to try to replicate. A nematode worm has just 302 neurons, which is miniscule compared to today's digital computer resources!

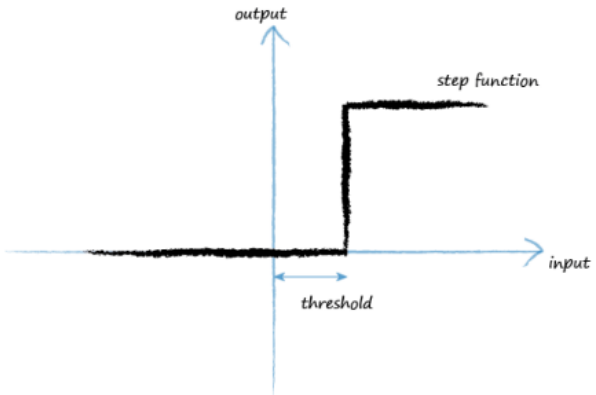
The complete functioning of brains, consciousness for example, is still a mystery, but enough is known about neurons to suggest different ways of doing computation, that is, different ways to solve problems

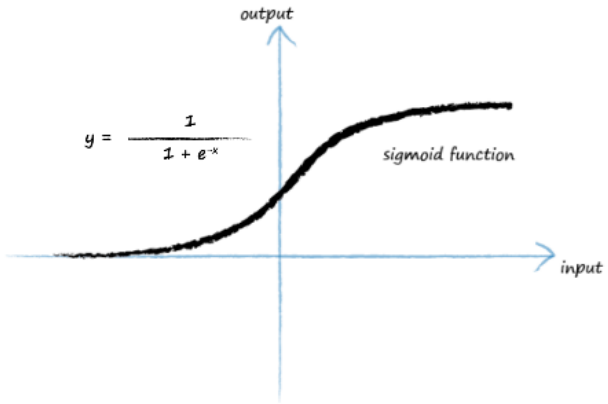
Activation functions

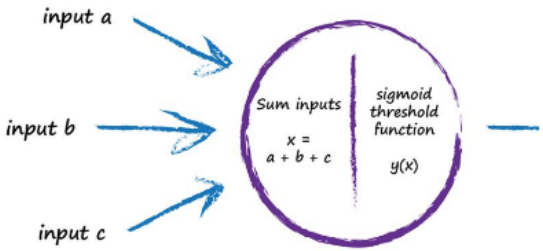
Observations suggest that neurons don't react readily, but instead suppress the input until it has grown so large that it triggers an output.

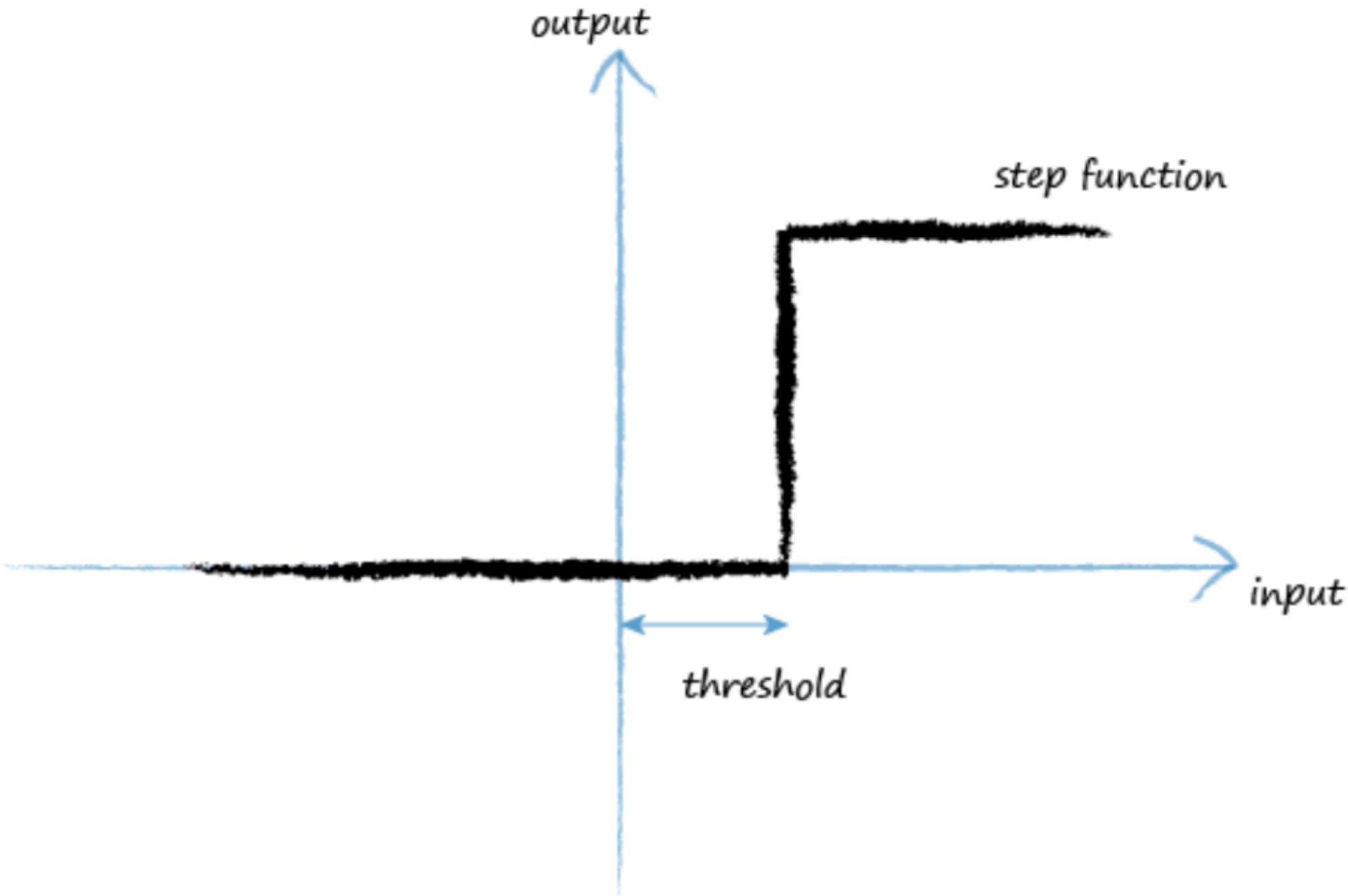
Intuitively this makes sense - the neurons don't want to be passing on tiny noise signals, only emphatically strong intentional signals.

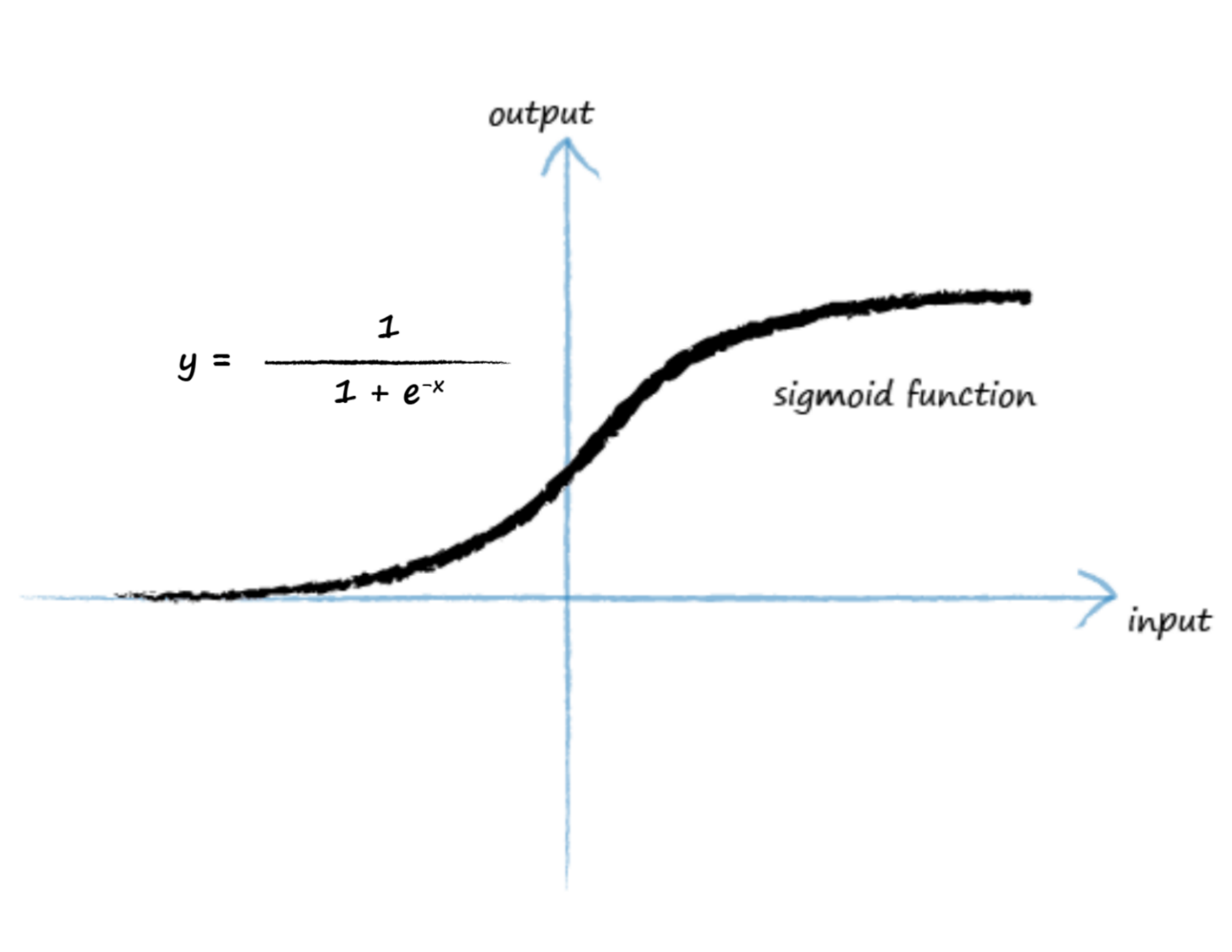
The following illustrates this idea of only producing an output signal if the input is sufficiently dialed up to pass a threshold.

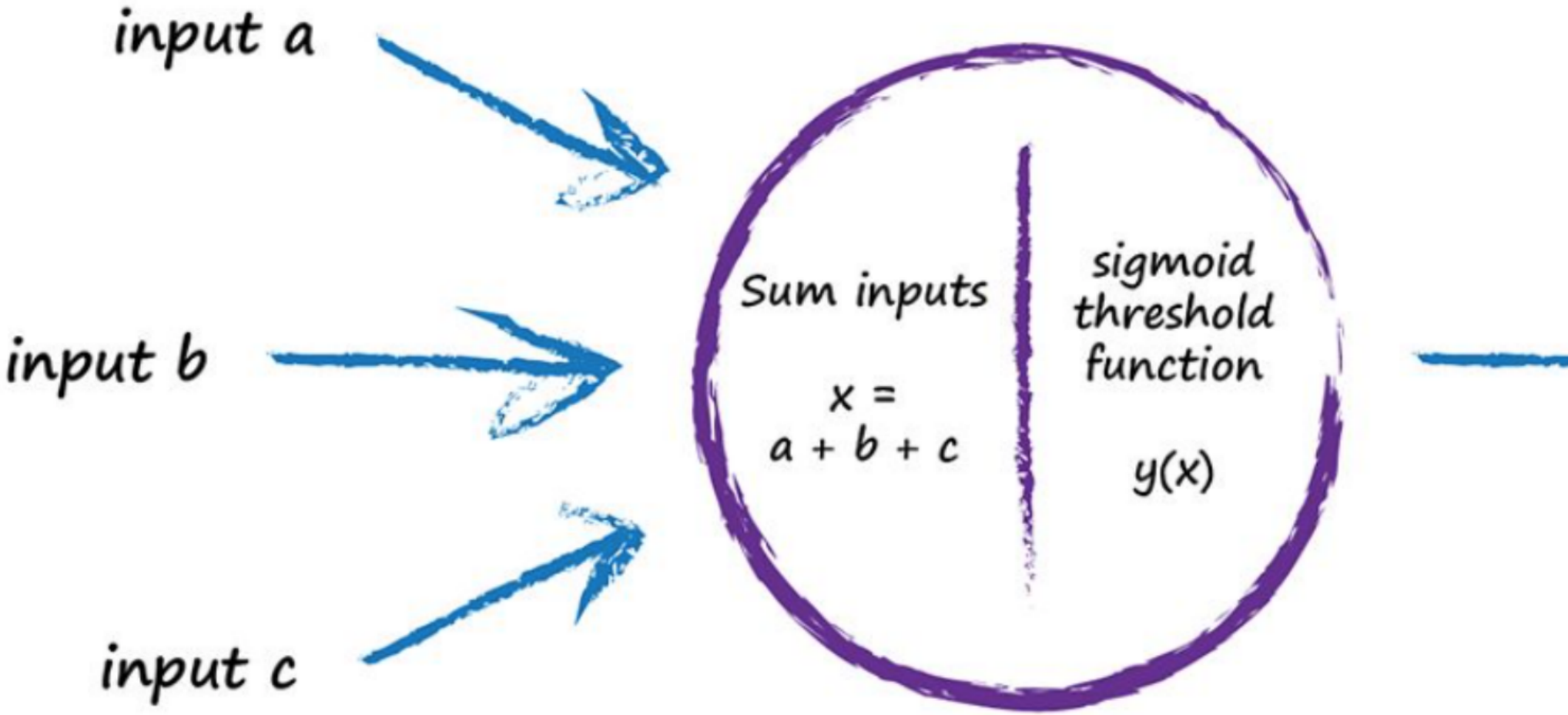







output
step function
input
threshold


output
$y = \frac{1}{1 + e^{-x}}$
sigmoid function
input

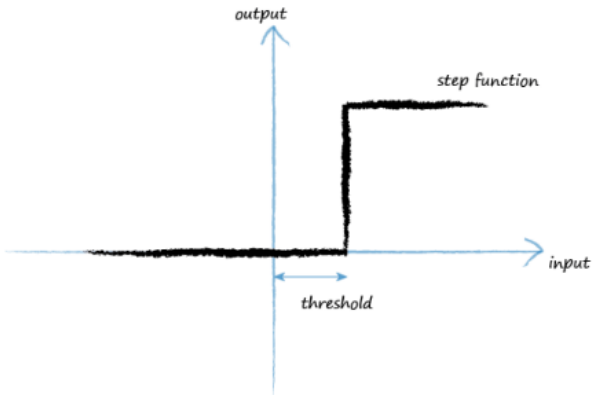

input a
input b
input c
Sum inputs
x =
a + b + c
sigmoid
threshold
function
y(x)

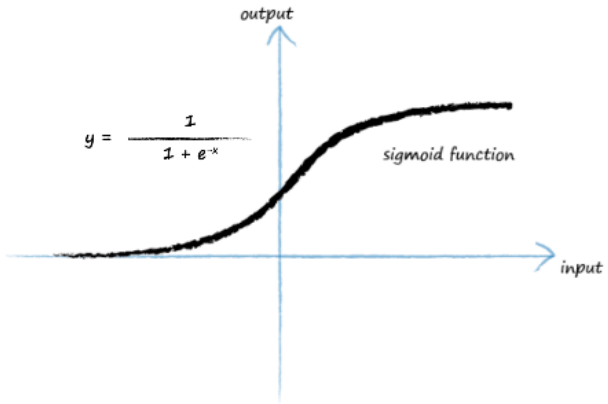
Activation functions

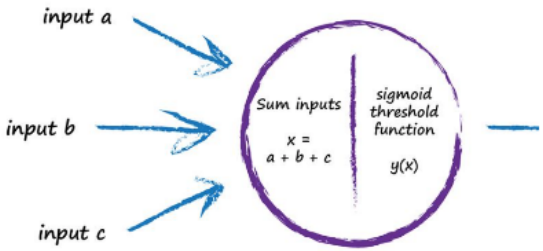
Observations suggest that neurons don't react readily, but instead suppress the input until it has grown so large that it triggers an output.

Intuitively this makes sense - the neurons don't want to be passing on tiny noise signals, only emphatically strong intentional signals.

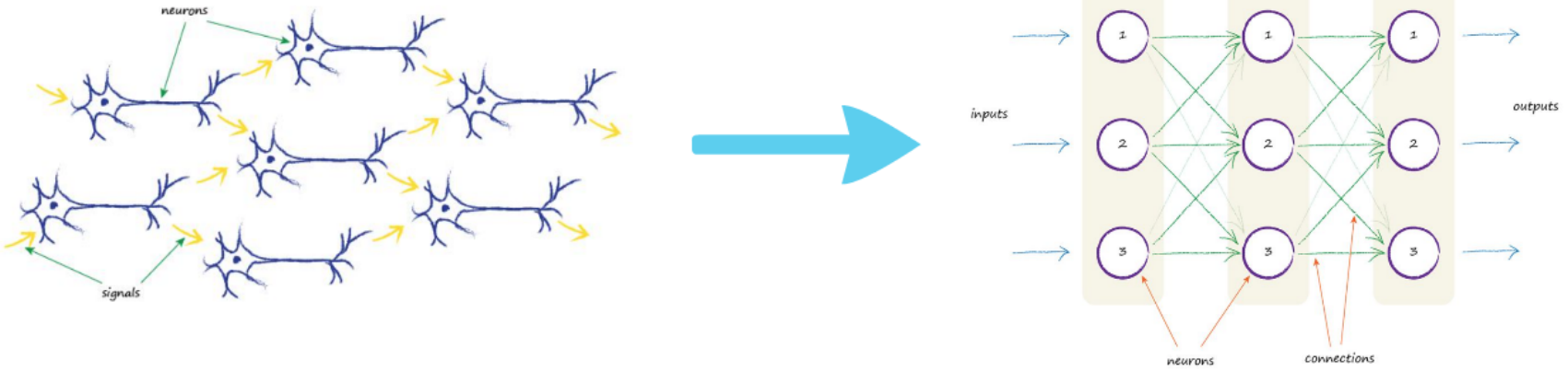
The following illustrates this idea of only producing an output signal if the input is sufficiently dialed up to pass a threshold.







A Biologically Inspired Model



If the combined signal is not large enough then the effect of the sigmoid threshold function is to suppress the output signal.

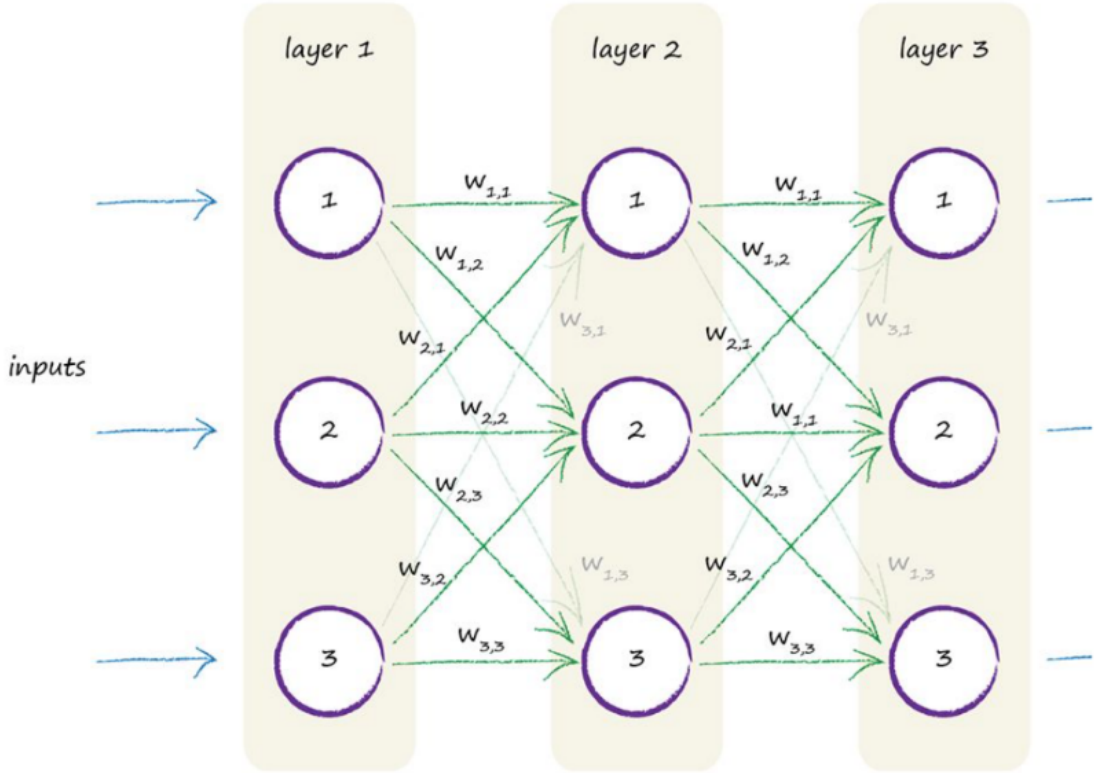
If the sum of inputs is large enough, the effect of the sigmoid is to fire the neuron. If only one of the several inputs is large and the rest small, this may be enough to fire the neuron.

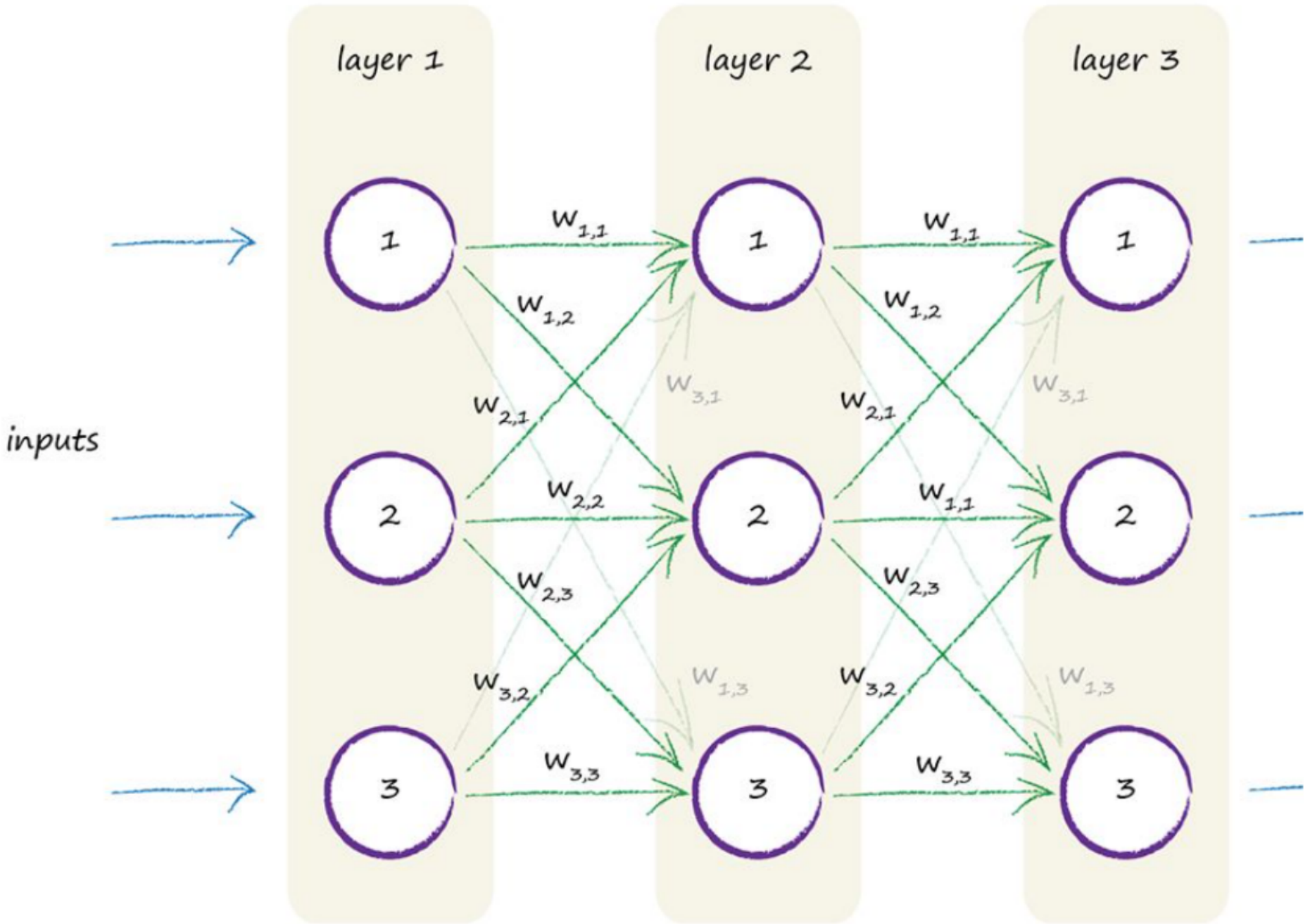
The neuron can fire if some of the inputs are individually almost, but not quite, large enough because when combined the signal is large enough to overcome the threshold.

Parameterising a Neural Network Model

The weight w(2,3) is simply the weight associated with the signal that passed between node 2 in a layer to node 3 in the next layer.

And so w(1,2) is the weight that diminishes or amplifies the signal between node 1 and node 2 in the next layer.

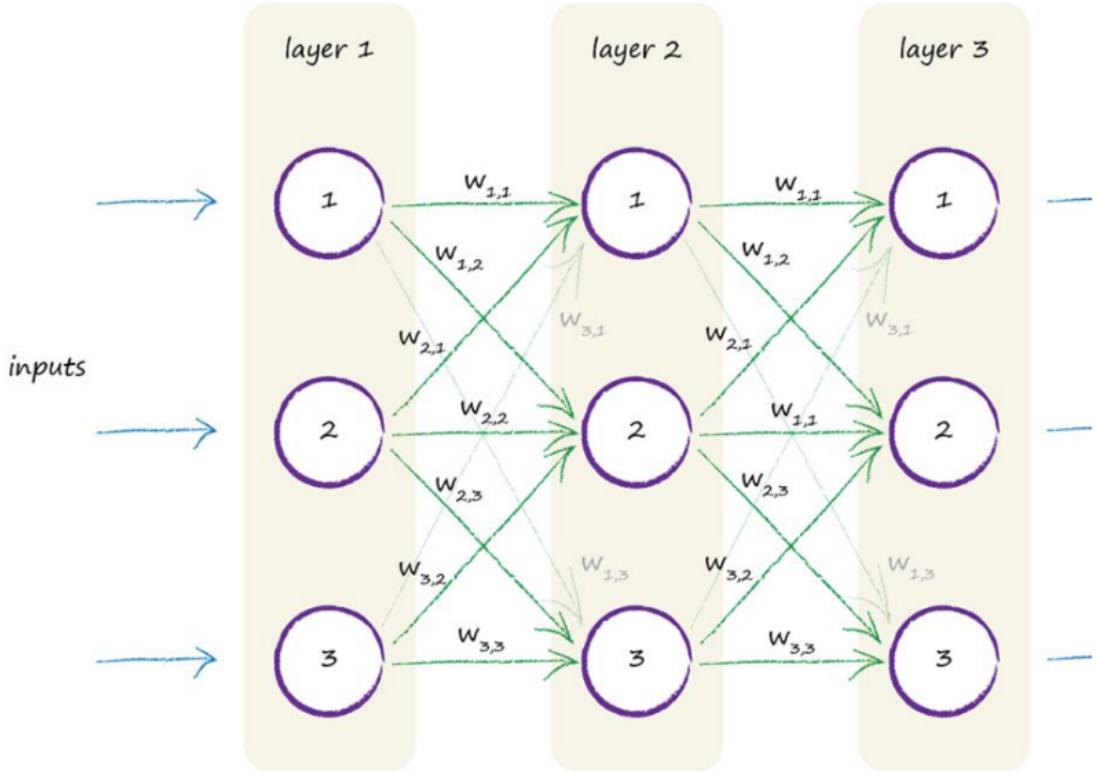



layer 1
layer 2
layer 3
inputs
1
2
3
1
2
3
1
2
3
$w_{1,1}$
$w_{1,2}$
$w_{2,1}$
$w_{2,2}$
$w_{2,3}$
$w_{3,2}$
$w_{3,3}$
$w_{3,1}$
$w_{1,3}$
$w_{1,1}$
$w_{1,2}$
$w_{2,1}$
$w_{1,1}$
$w_{2,3}$
$w_{3,2}$
$w_{3,3}$
$w_{3,1}$
$w_{1,3}$

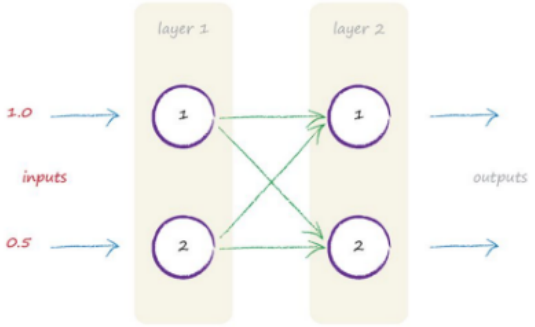
Parameterising a Neural Network Model

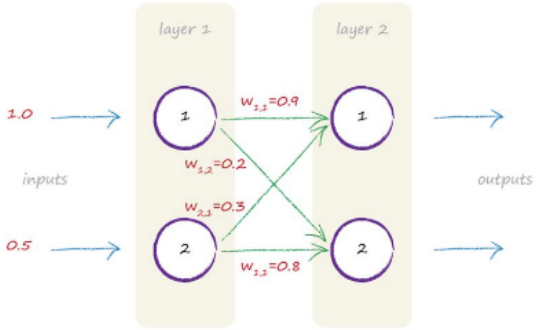
The weight w(2,3) is simply the weight associated with the signal that passed between node 2 in a layer to node 3 in the next layer.

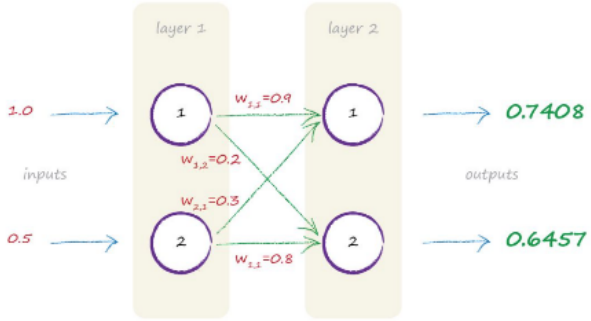
And so w(1,2) is the weight that diminishes or amplifies the signal between node 1 and node 2 in the next layer.

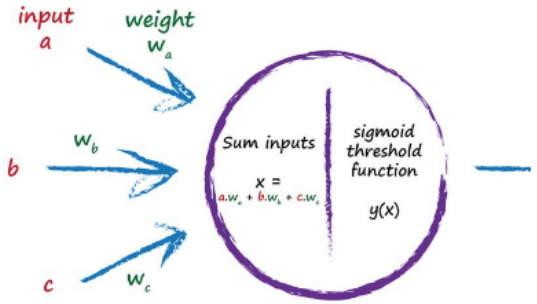


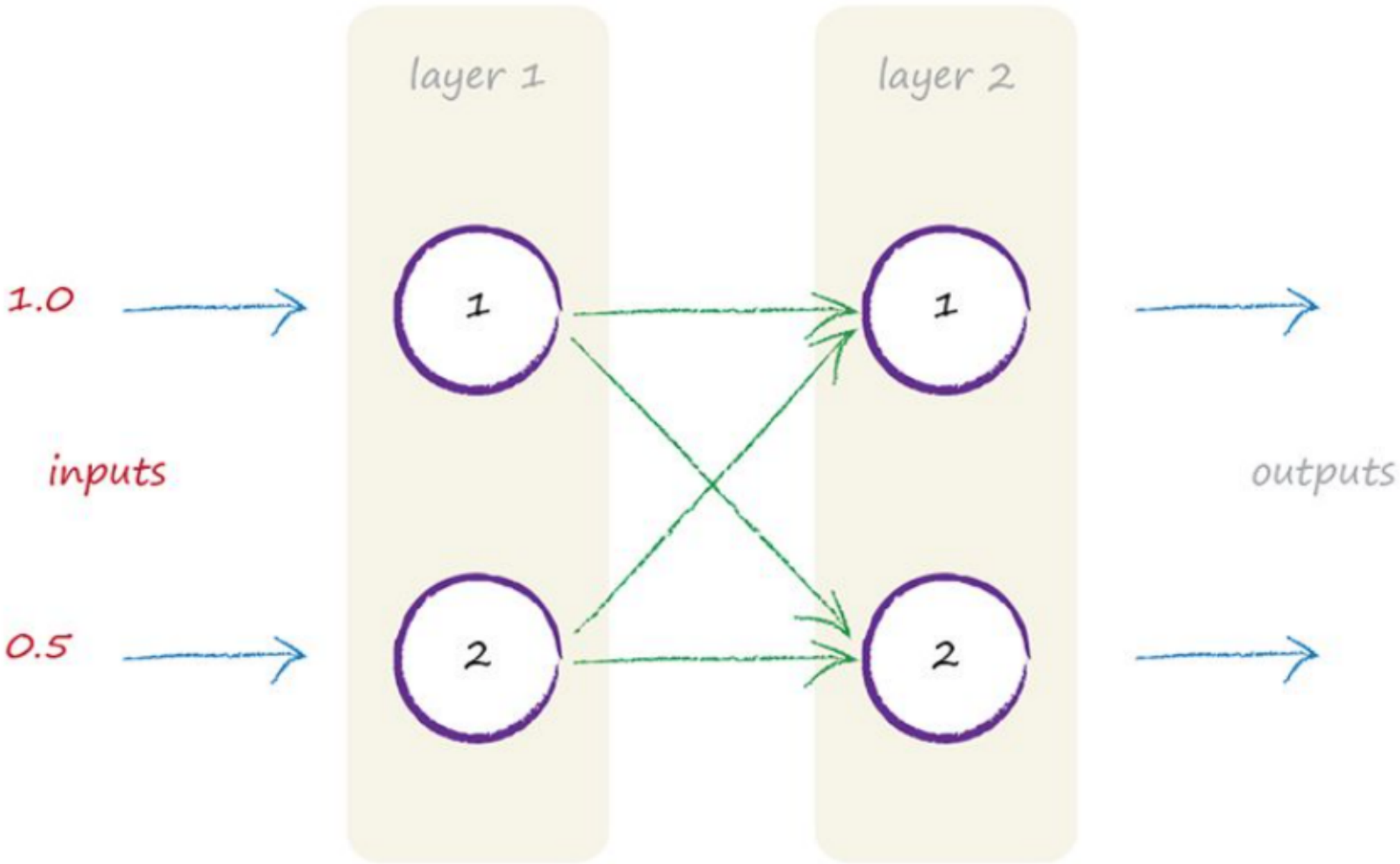
Following Signals Through A Neural Network

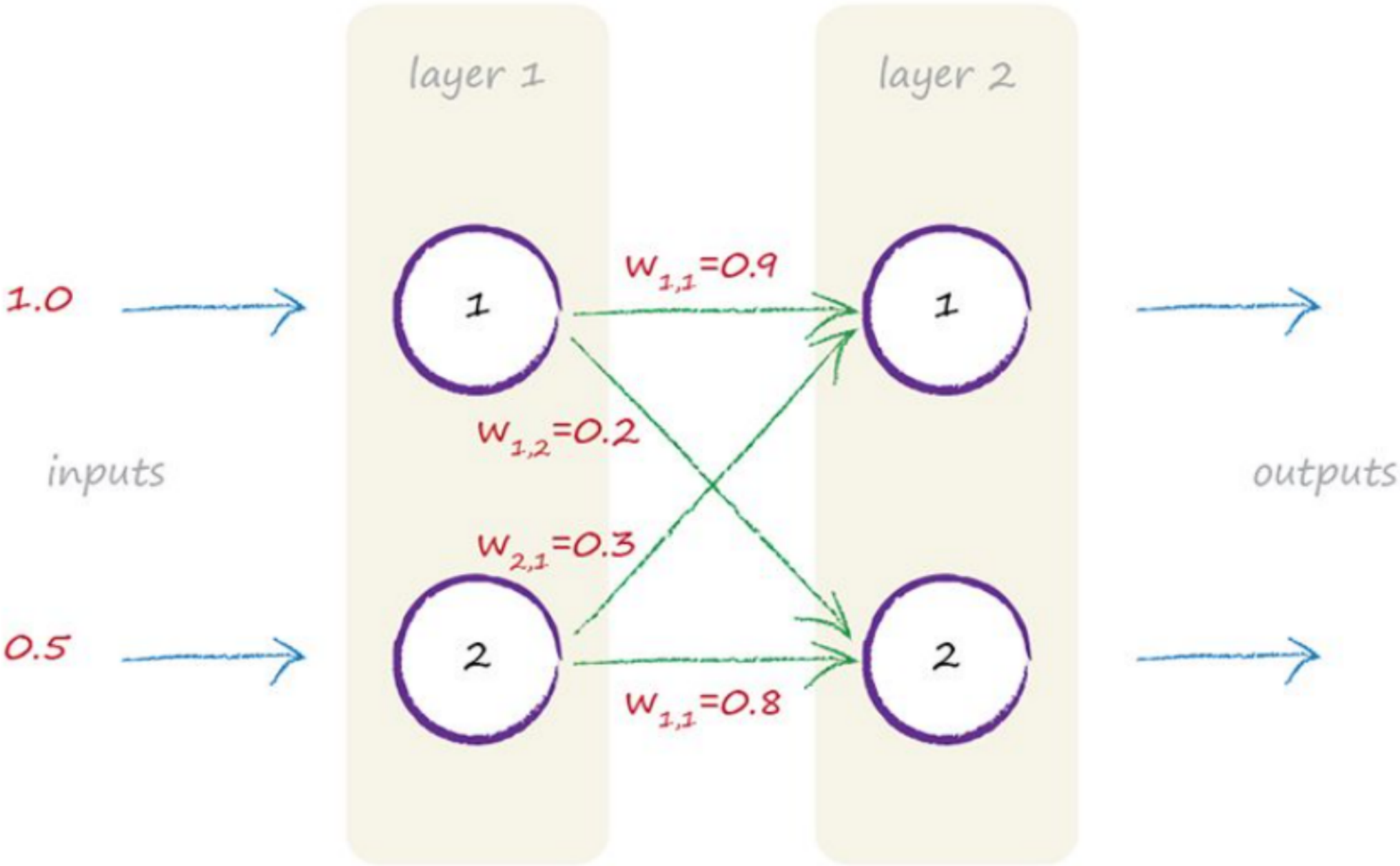


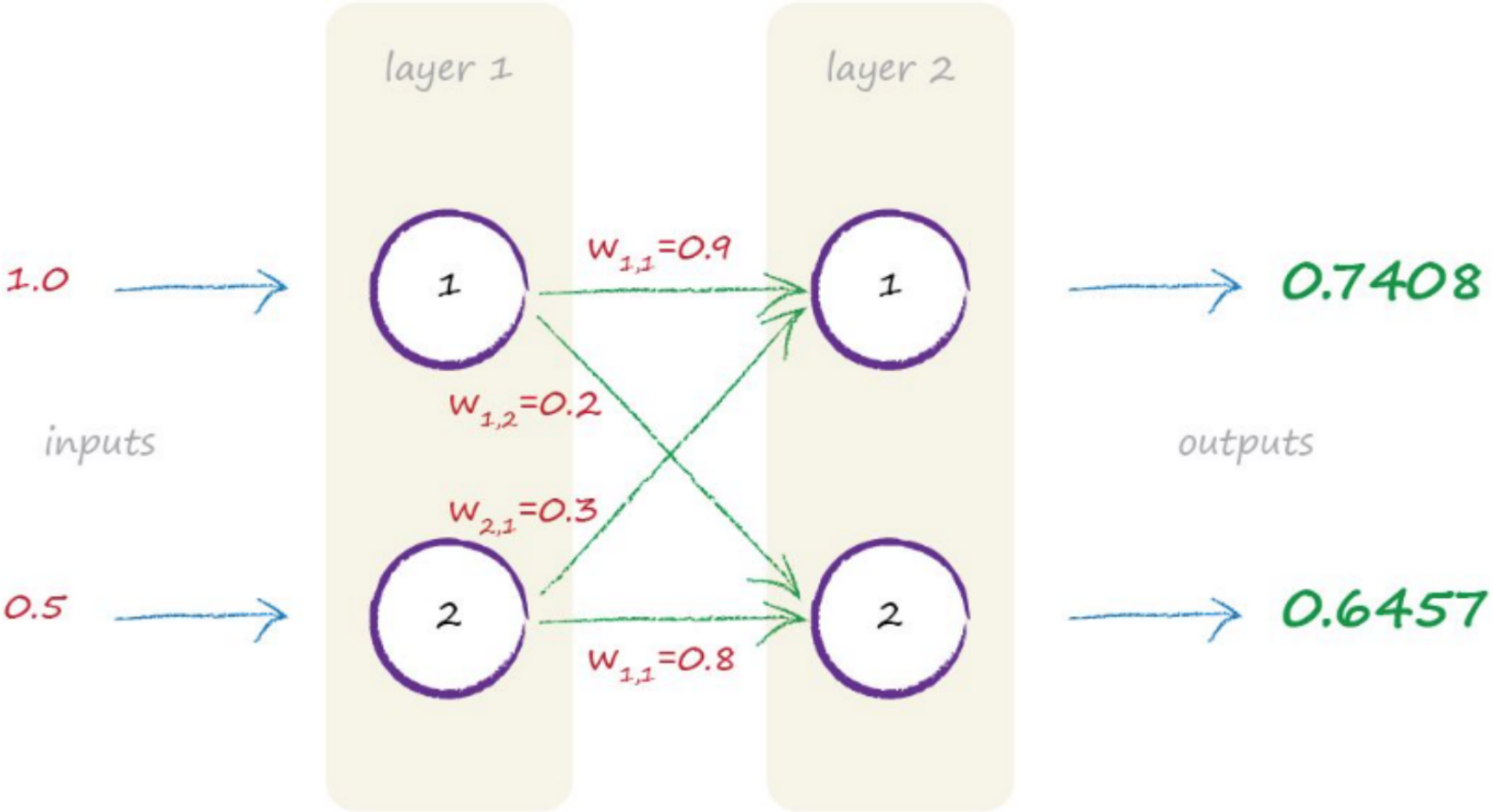


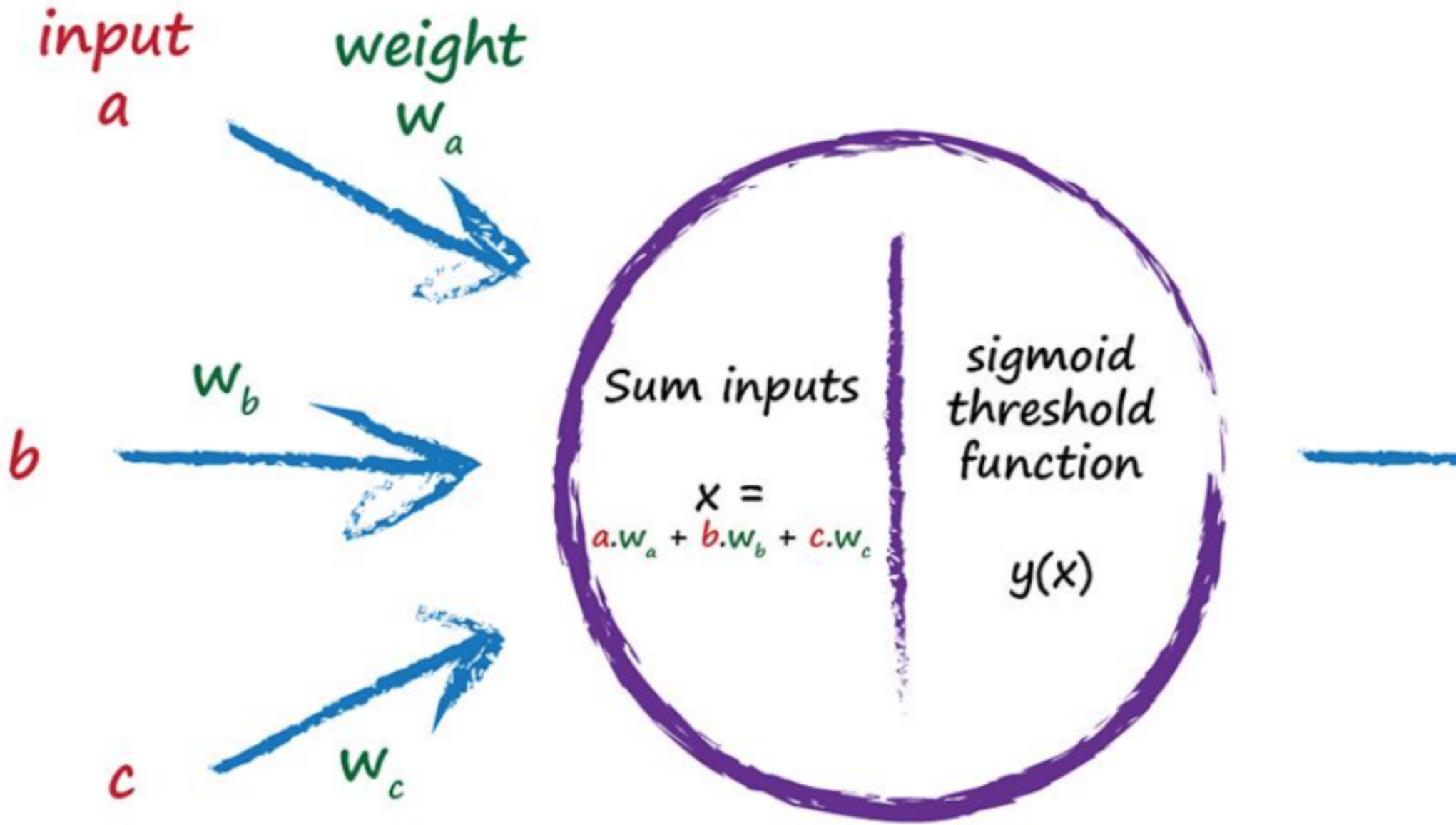




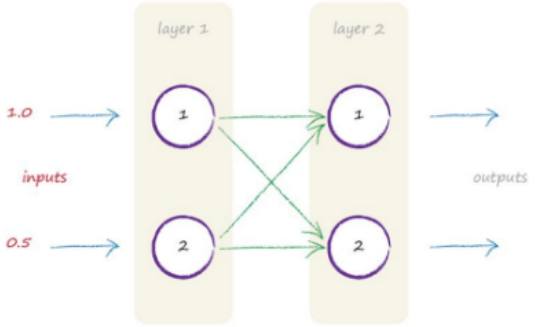

layer 1
layer 2
1.0
0.5
inputs
1
2
1
2
outputs

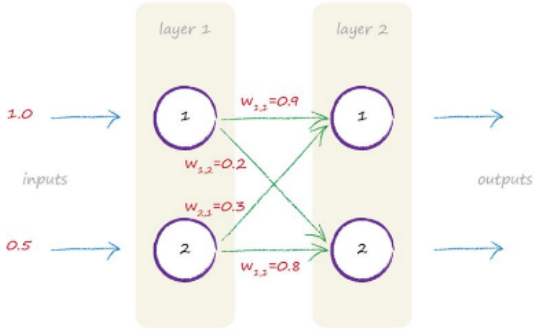

layer 1
layer 2
1.0
0.5
inputs
outputs
1
2
1
2
$w_{1,1}$=0.9
$w_{1,2}$=0.2
$w_{2,1}$=0.3
$w_{1,1}$=0.8

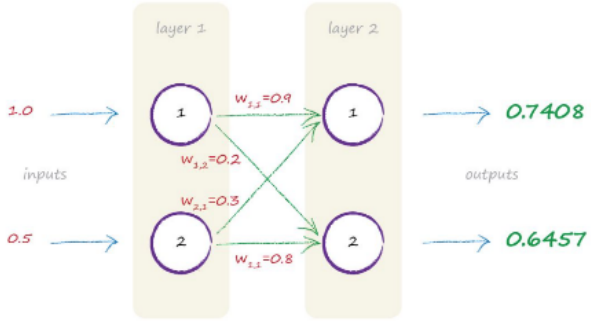

layer 1
layer 2
1.0
0.5
inputs
1
2
1
2
$w_{1,1}$=0.9
$w_{1,2}$=0.2
$w_{2,1}$=0.3
$w_{1,1}$=0.8
outputs
0.7408
0.6457

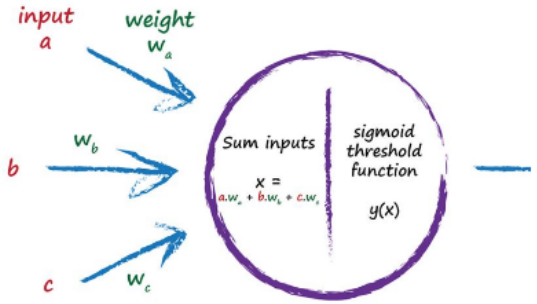

input
a
weight
w_a
b
w_b
c
w_c
Sum inputs
$x = a.w_a + b.w_b + c.w_c$
sigmoid threshold function
$y(x)$

Following Signals Through A Neural Network







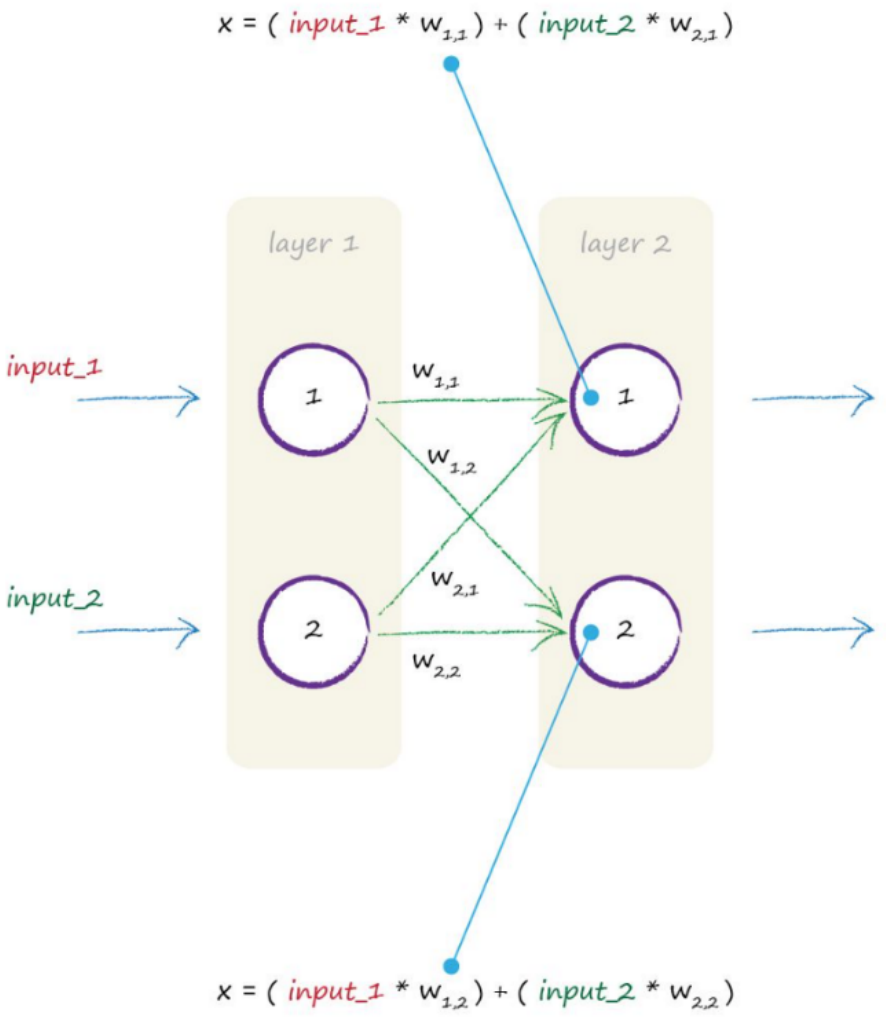


Writing More Succinctly in Matrix Notation

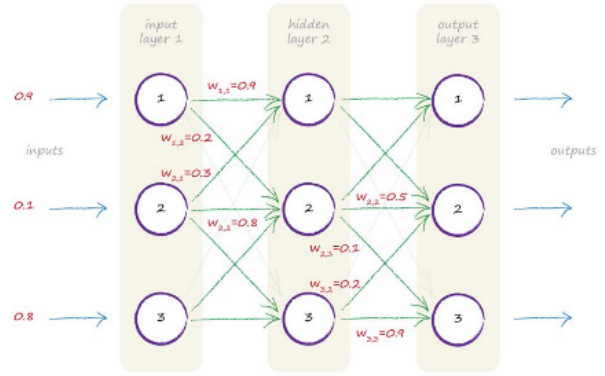
$$\begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} input\_1 \\ input\_2 \end{pmatrix}$$

$X = WI$

$O = Sigmoid(X)$



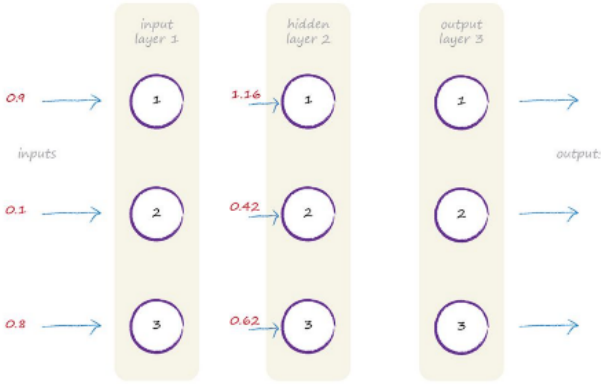
A More Complex Example

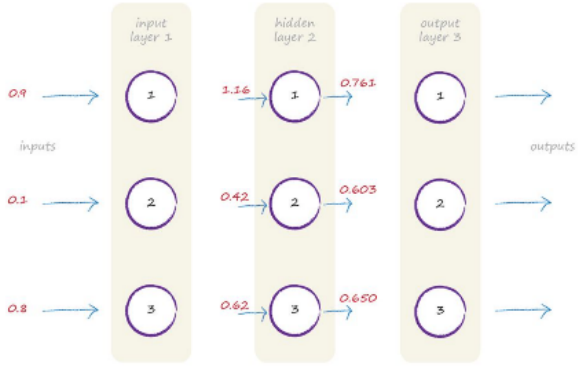


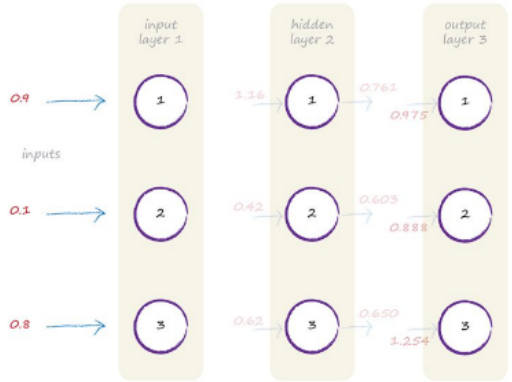
$$I = \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$$

$$W_{input\_hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix}$$

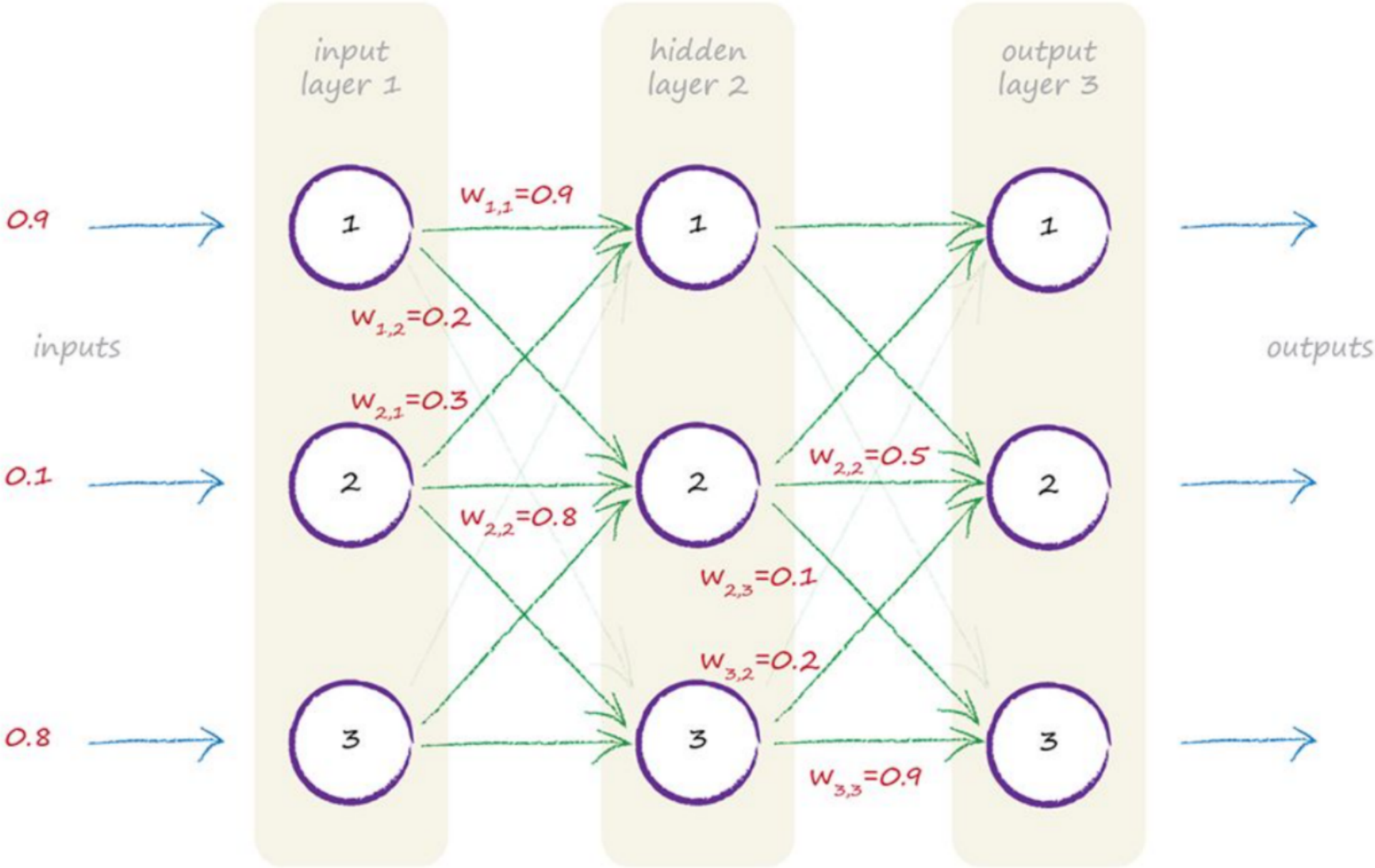
$$W_{hidden\_output} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix}$$

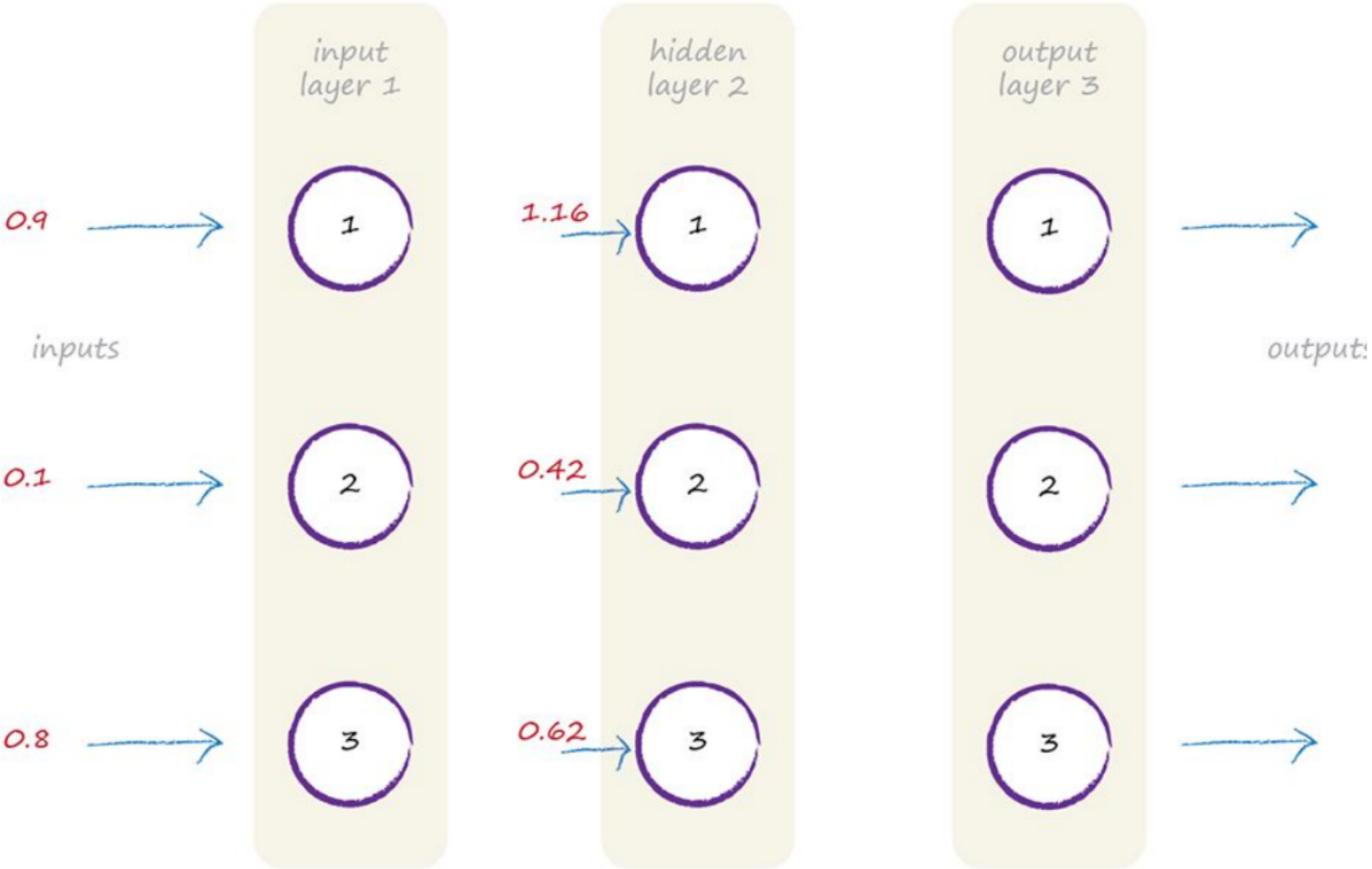


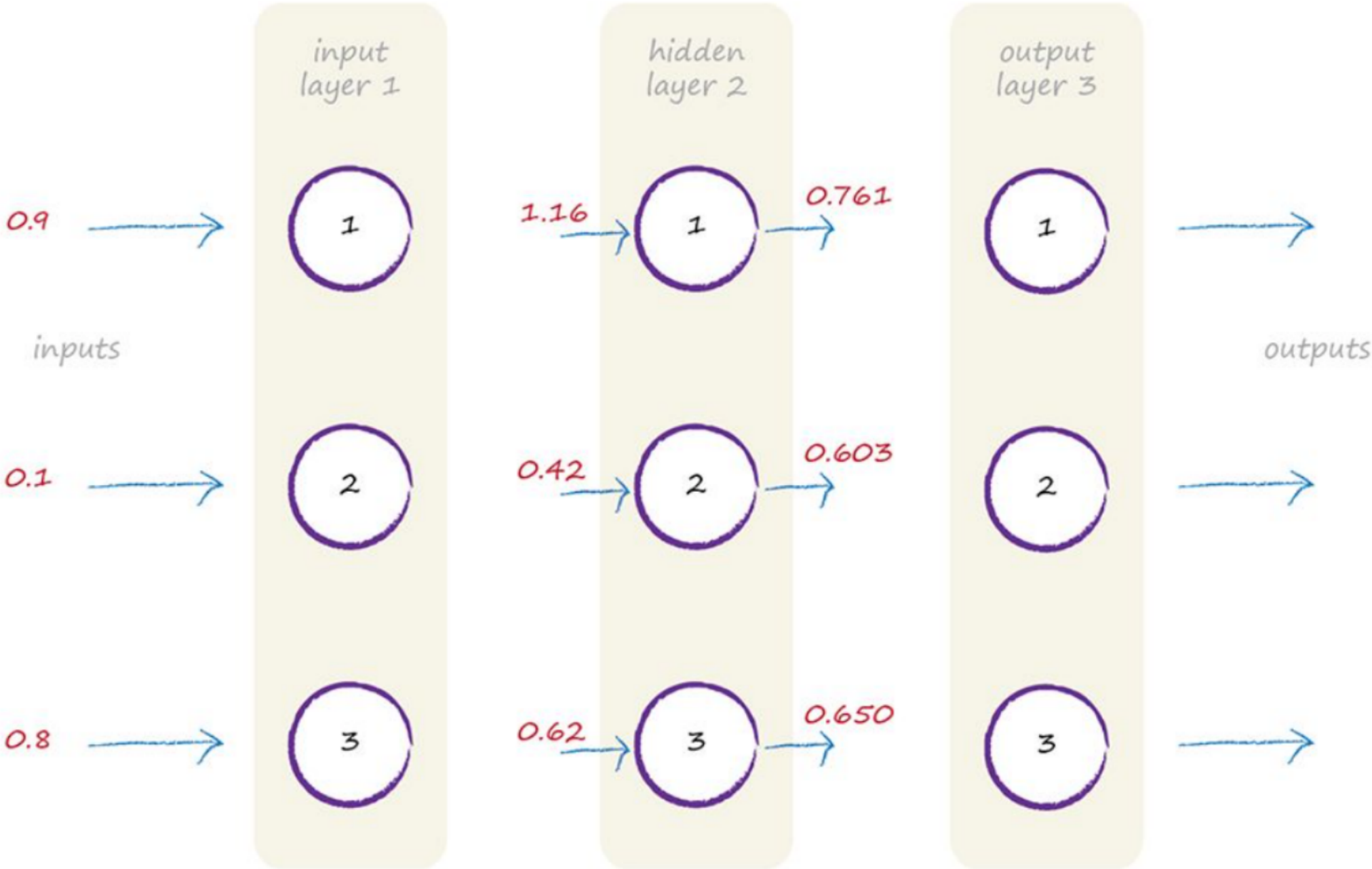


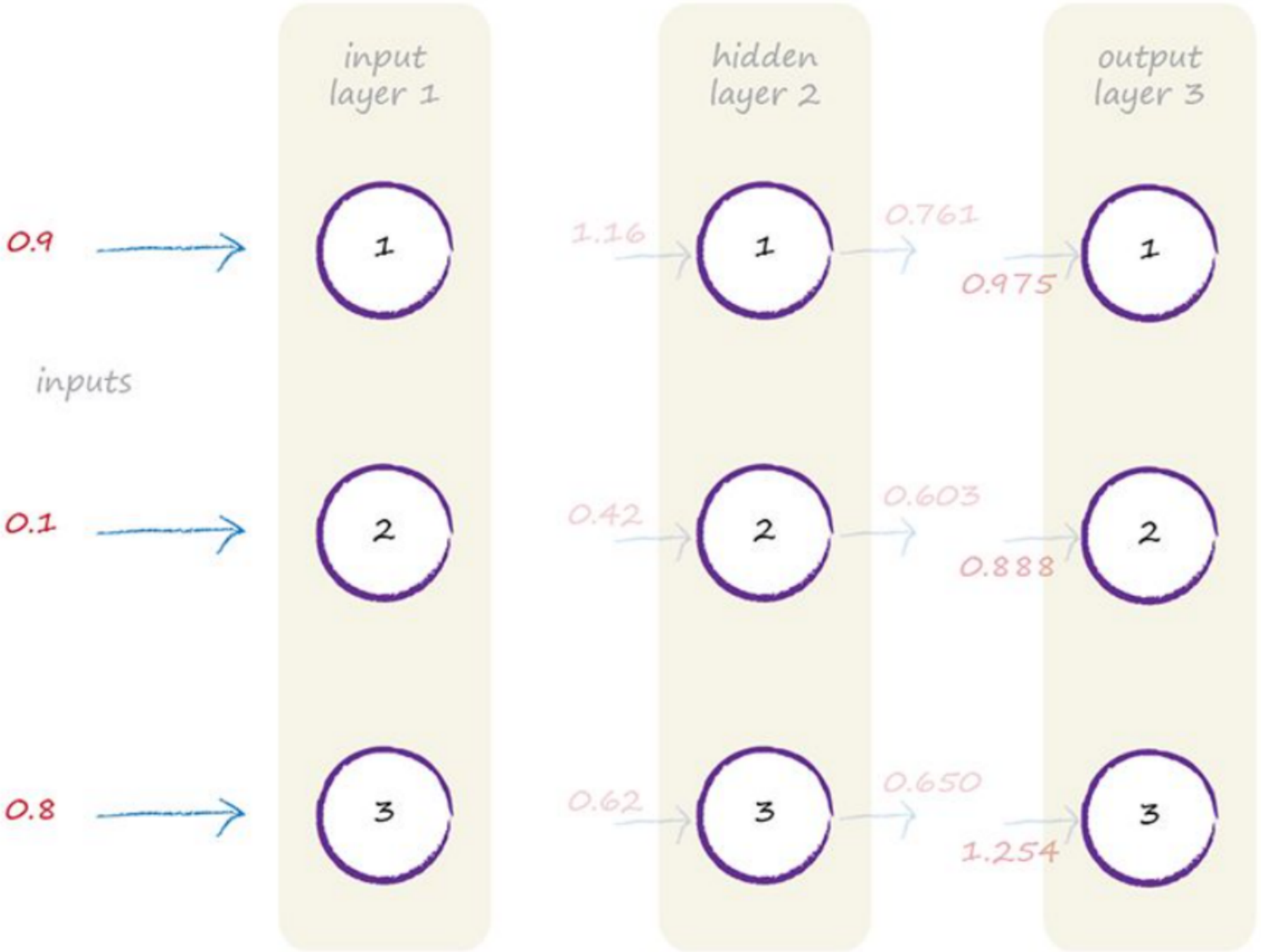


$$= \begin{pmatrix} 0.726 \\ 0.708 \\ 0.778 \end{pmatrix}$$


input layer 1
hidden layer 2
output layer 3
0.9
0.1
0.8
inputs
outputs
1
2
3
1
2
3
1
2
3
$w_{1,1}=0.9$
$w_{1,2}=0.2$
$w_{2,1}=0.3$
$w_{2,2}=0.8$
$w_{2,2}=0.5$
$w_{2,3}=0.1$
$w_{3,2}=0.2$
$w_{3,3}=0.9$


input
layer 1
hidden
layer 2
output
layer 3
0.9
1.16
1
1
1
inputs
output:
0.1
0.42
2
2
2
0.8
0.62
3
3
3


input
layer 1
hidden
layer 2
output
layer 3
0.9
1.16
0.761
1
1
1
inputs
outputs
0.1
0.42
0.603
2
2
2
0.8
0.62
0.650
3
3
3

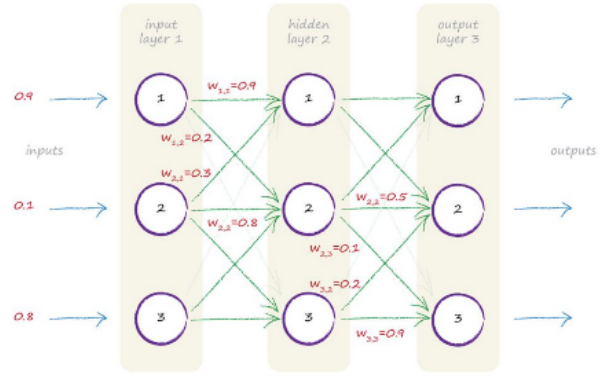

input layer 1
hidden layer 2
output layer 3
inputs
0.9
0.1
0.8
1
2
3
1.16
0.42
0.62
0.761
0.603
0.650
0.975
0.888
1.254

1

2

$$= \begin{pmatrix} 0.726 \\ 0.708 \\ 0.778 \end{pmatrix}$$

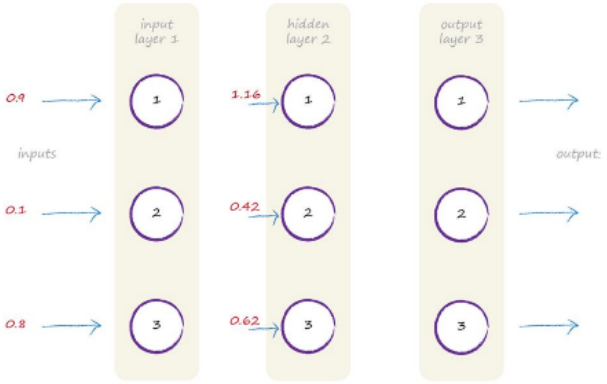
A More Complex Example

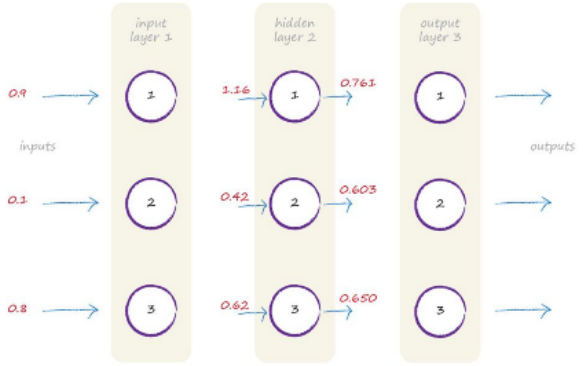


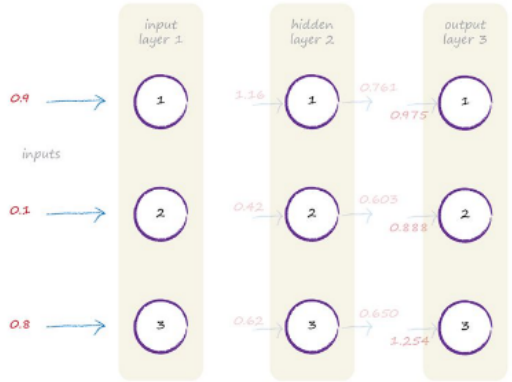
$$I = \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$$

$$W_{input\_hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix}$$

$$W_{hidden\_output} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix}$$

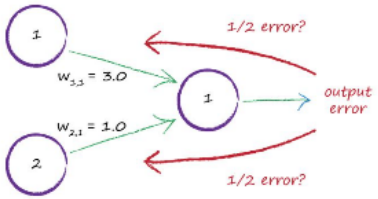


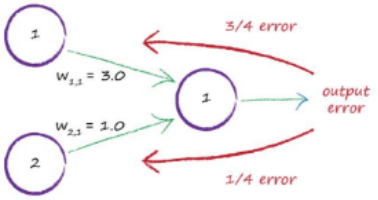


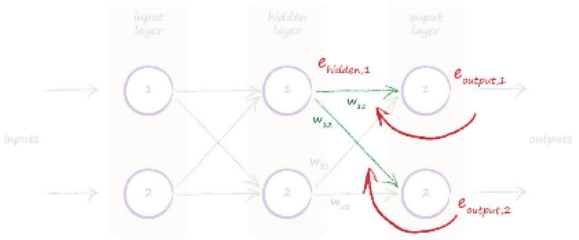


$$= \begin{pmatrix} 0.726 \\ 0.708 \\ 0.778 \end{pmatrix}$$

Backpropagation of Errors

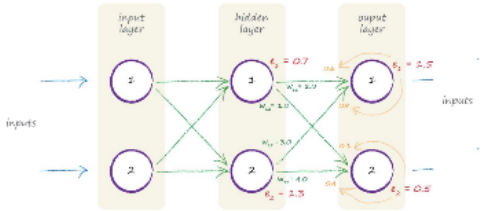






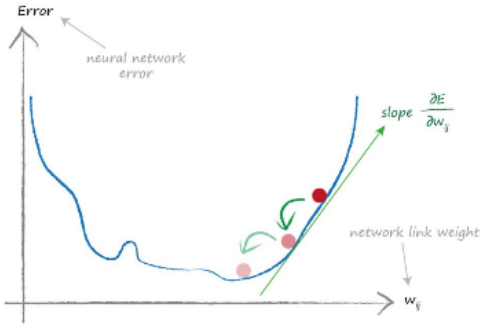
$e_{hidden,1}$ = sum of split errors on links w_{11} and w_{12}

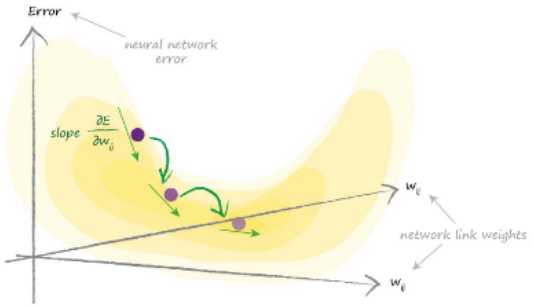
$$= e_{output,1} * \frac{w_{11}}{w_{11} + w_{21}} + e_{output,2} * \frac{w_{12}}{w_{12} + w_{22}}$$

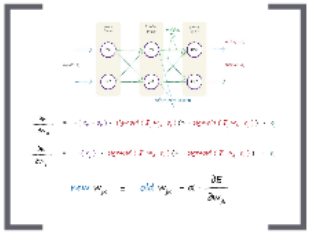


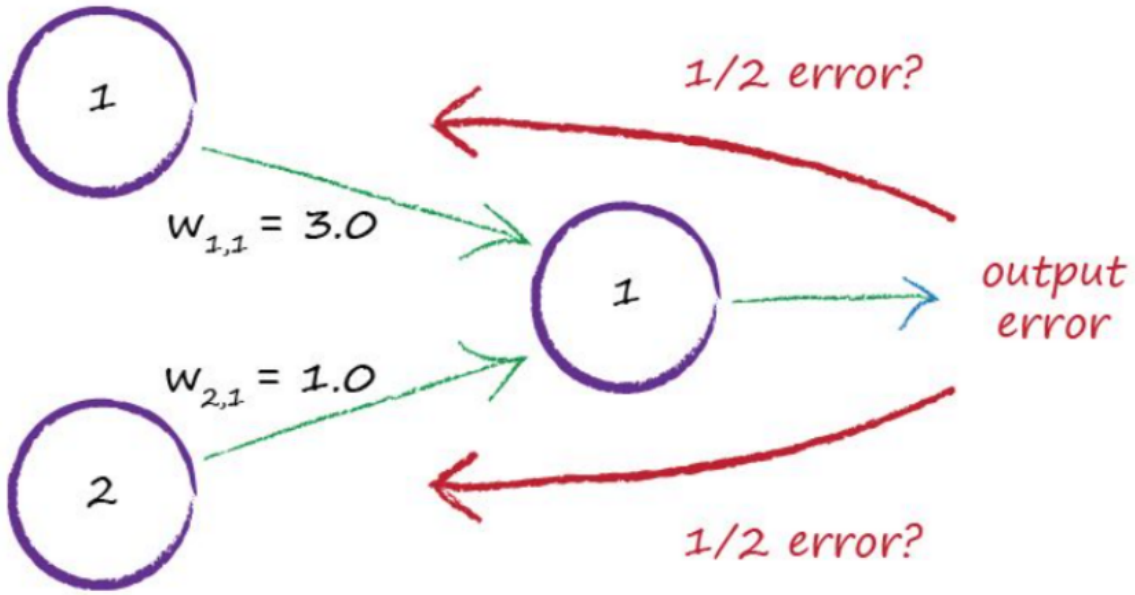
$$error_{hidden} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

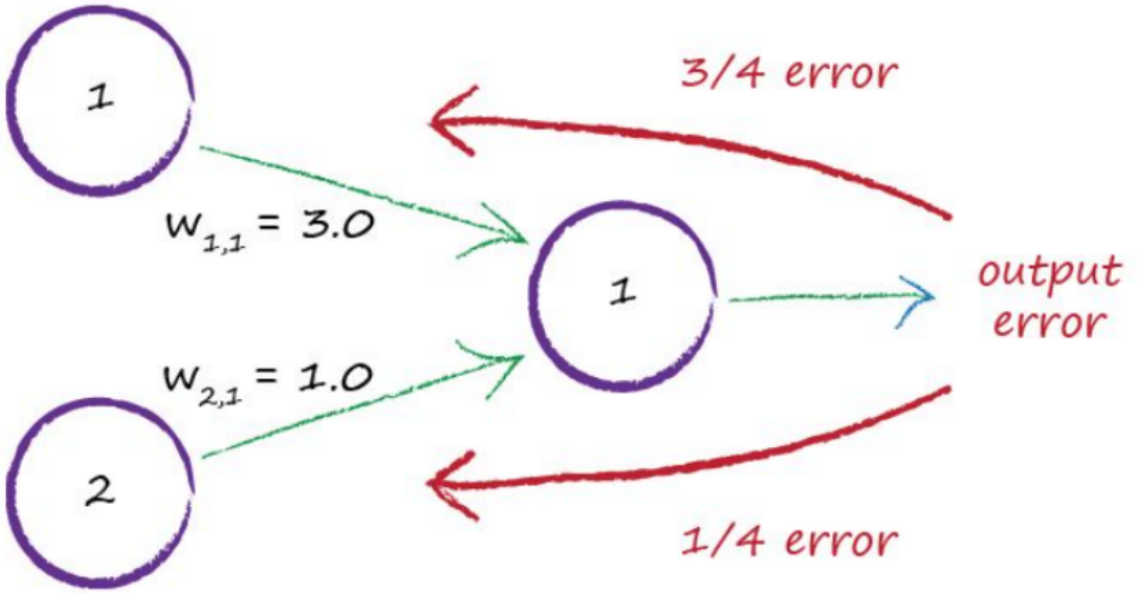
We can compute errors simply using matrix multiplication!

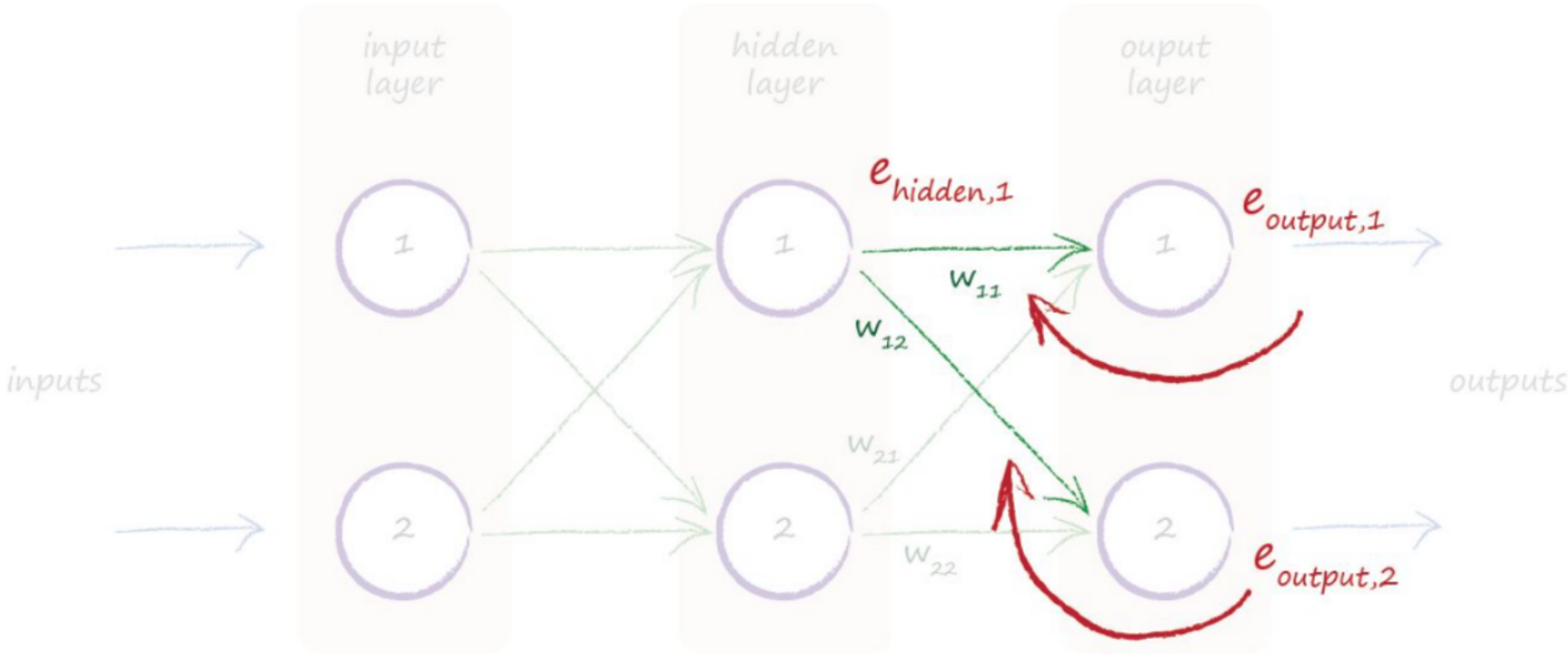






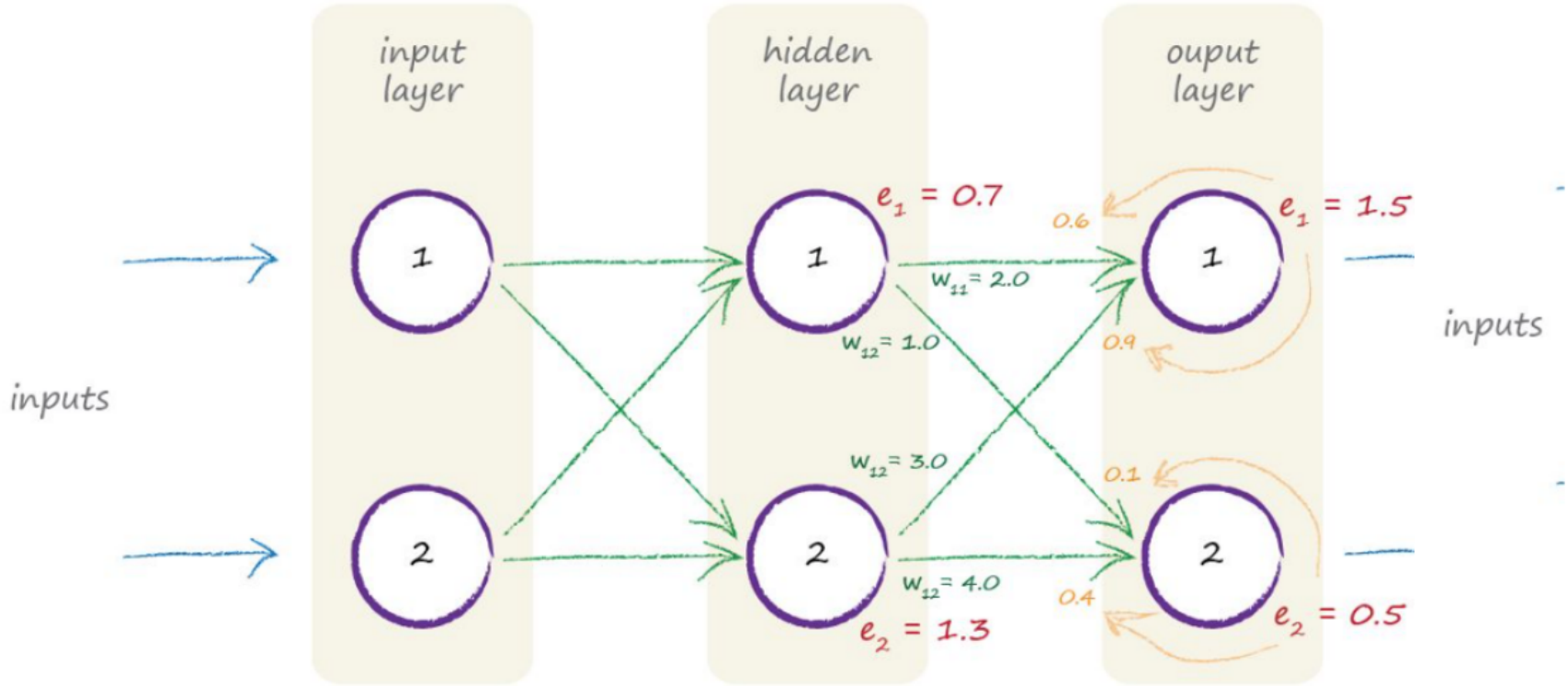

1
1/2 error?
$w_{1,1} = 3.0$
1
output
error
$w_{2,1} = 1.0$
2
1/2 error?


1
3/4 error
$w_{1,1} = 3.0$
1
output
error
$w_{2,1} = 1.0$
2
1/4 error



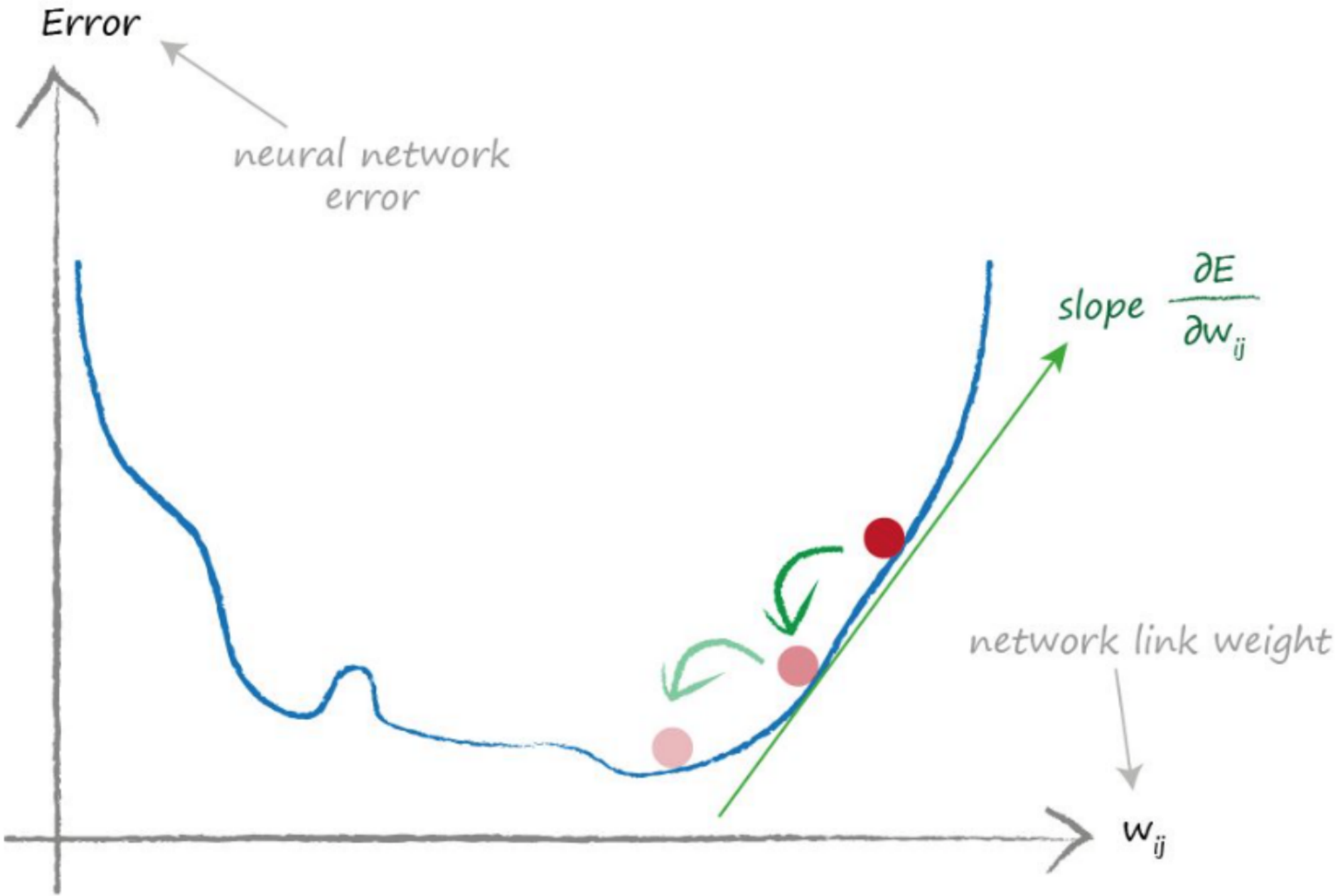
$e_{hidden,1}$ = sum of split errors on links w_{11} and w_{12}

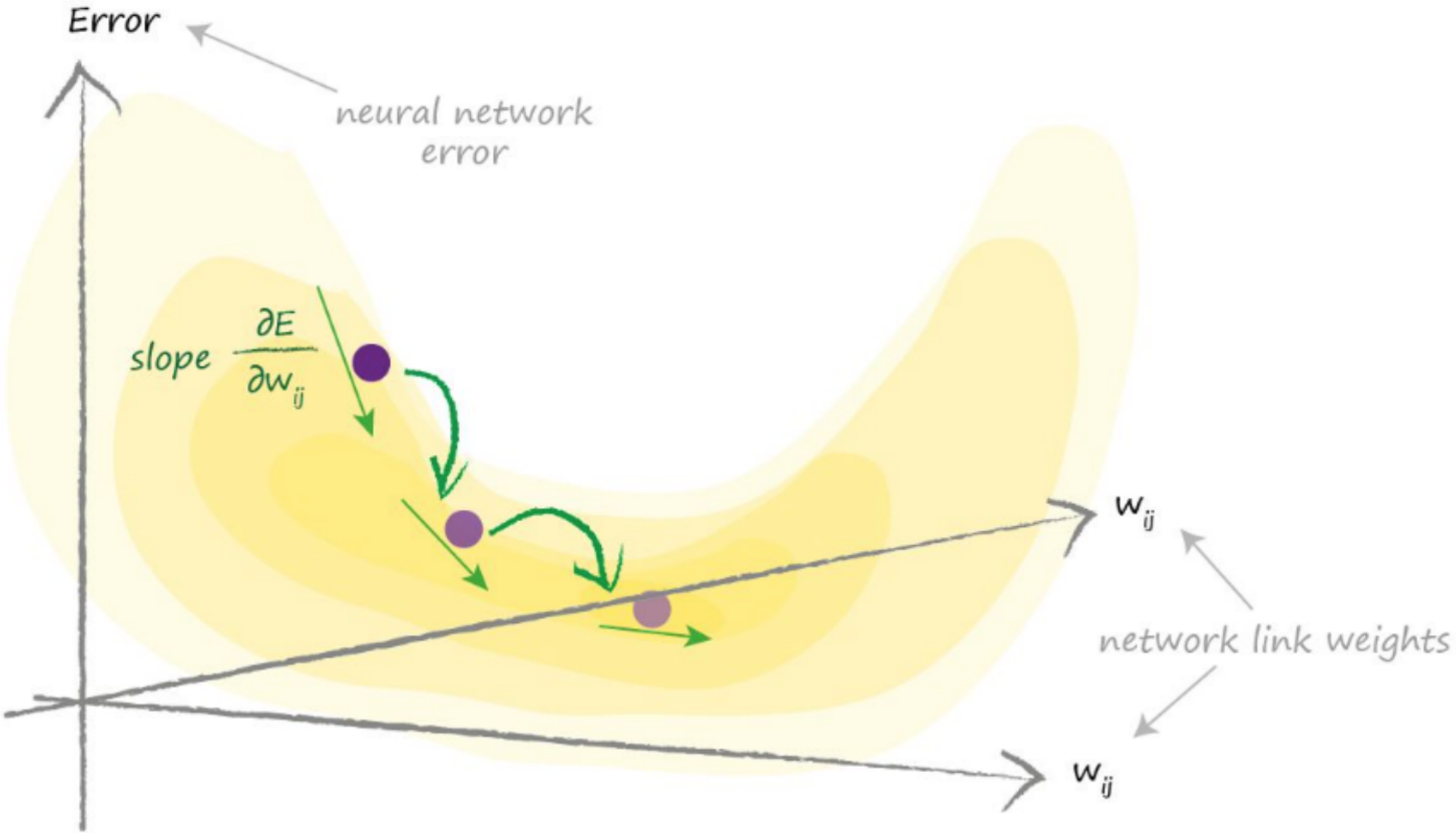
$$= e_{output,1} * \frac{w_{11}}{w_{11} + w_{21}} + e_{output,2} * \frac{w_{12}}{w_{12} + w_{22}}$$

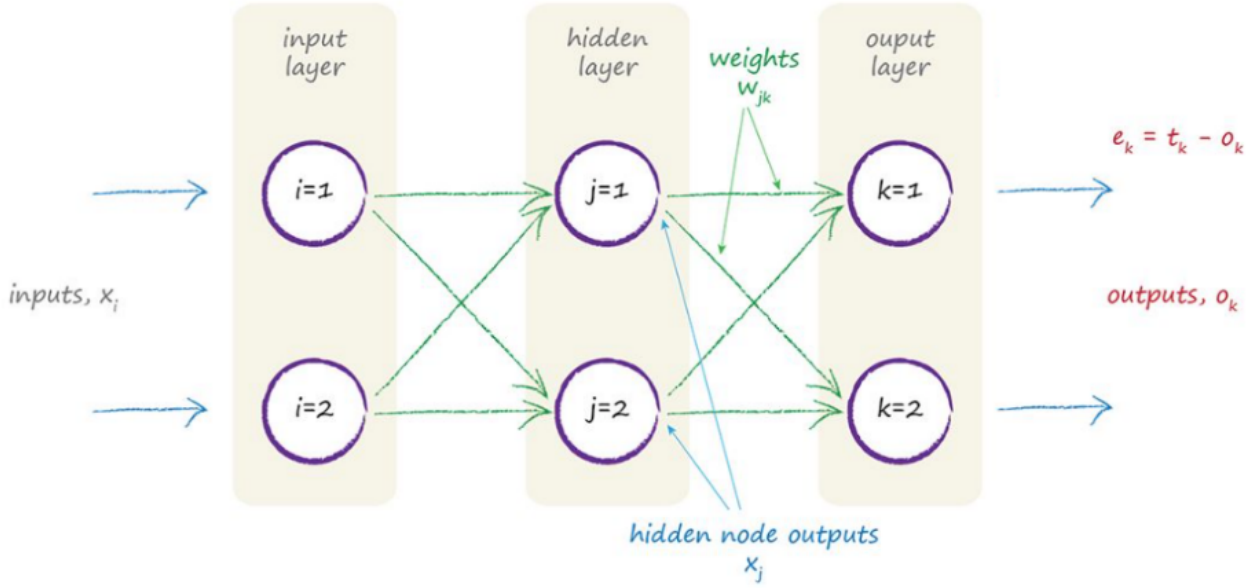


$$error_{hidden} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

We can compute errors simply using matrix multiplication!


Error
neural network error
slope $\frac{\partial E}{\partial w_{ij}}$
network link weight
w_{ij}


Error
neural network error
slope $\frac{\partial E}{\partial w_{ij}}$
w_{ij}
network link weights
w_{ij}

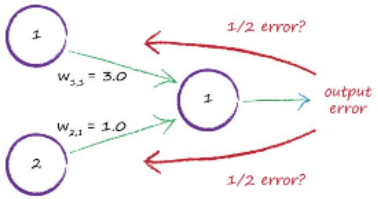


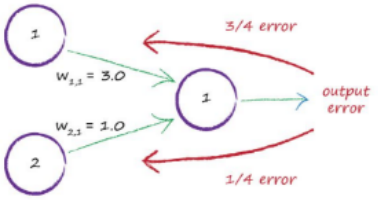
$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) \cdot \text{sigmoid}(\Sigma_j w_{jk} \cdot o_j)(1 - \text{sigmoid}(\Sigma_j w_{jk} \cdot o_j)) \cdot o_j$$

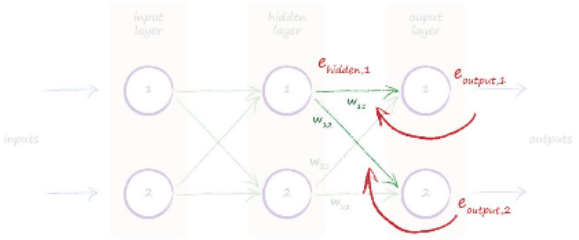
$$\frac{\partial E}{\partial w_{ij}} = -(e_j) \cdot \text{sigmoid}(\Sigma_i w_{ij} \cdot o_i)(1 - \text{sigmoid}(\Sigma_i w_{ij} \cdot o_i)) \cdot o_i$$

$$\text{new } w_{jk} = \text{old } w_{jk} - \alpha \cdot \frac{\partial E}{\partial w_{jk}}$$

Backpropagation of Errors

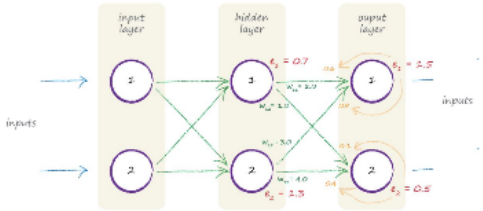






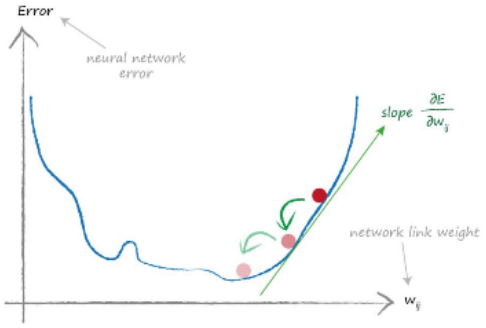
$e_{hidden,1}$ = sum of split errors on links w_{11} and w_{12}

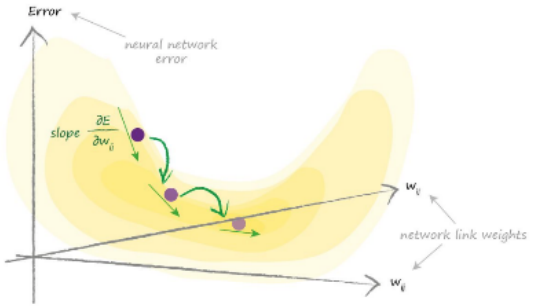
$$= e_{output,1} * \frac{w_{11}}{w_{11} + w_{21}} + e_{output,2} * \frac{w_{12}}{w_{12} + w_{22}}$$

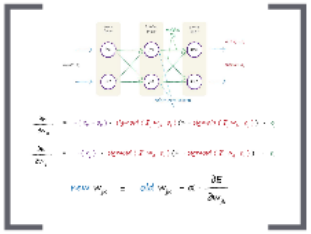


$$error_{hidden} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

We can compute errors simply using matrix multiplication!







Further Reading

http://www.deeplearningbook.org/

Chapter 28
Machine Learning: A Probabilistic Perspective
Kevin Murphy

M5MS10
Machine Learning

Spring 2018

Lecture 10

Dr Ben Calderhead

b.calderhead@imperial.ac.uk