

Welcome to the introduction  
to computational cognitive  
modelling workshop!

## **Part 2: Introduction to artificial neural networks**

# Part 2: **Introduction to artificial neural networks**

*Olivia Guest*

*Chris Brand*

*Nick Sexton*

*Nicole Cruz De Echeverria Loebell*

# What is a neural network?

A mathematical model

- Inspired by the nervous system

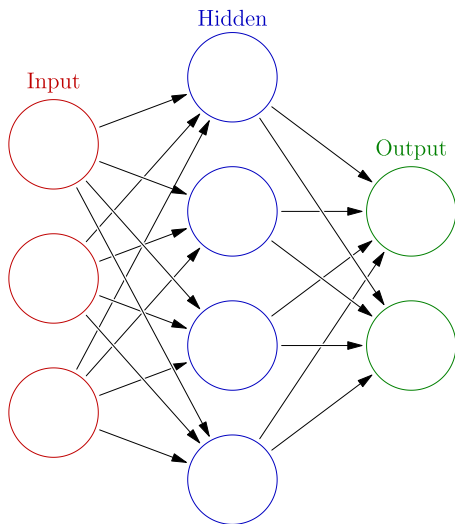


Figure: Glosser.ca / CC-BY-SA-3.0

# What is a neural network?

A mathematical model

- ▶ Inspired by the nervous system
- ▶ A set of *units*, connected by *weights*

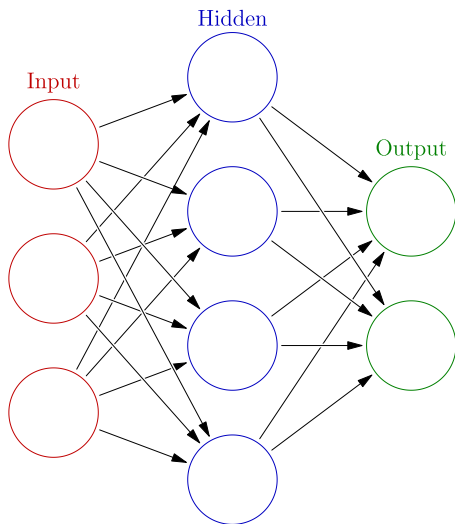


Figure: Glosser.ca / CC-BY-SA-3.0

# What is a neural network?

A mathematical model

- ▶ Inspired by the nervous system
- ▶ A set of *units*, connected by *weights*
- ▶ The network *runs* by passing *activations* from the *input* (to the *hidden*) to the *output* units

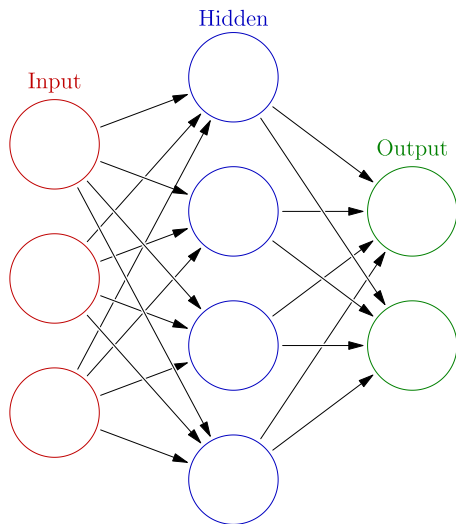


Figure: Glosser.ca / CC-BY-SA-3.0

# Why use artificial neural networks for modelling?

Some aspects of their behaviour are like their namesake!

- ▶ Learn pretty much any input-output data

# Why use artificial neural networks for modelling?

Some aspects of their behaviour are like their namesake!

- ▶ Learn pretty much any input-output data
- ▶ Uncover rules on their own about data

# Why use artificial neural networks for modelling?

Some aspects of their behaviour are like their namesake!

- ▶ Learn pretty much any input-output data
- ▶ Uncover rules on their own about data
- ▶ Generalise from what they have learnt



# Why use artificial neural networks for modelling?

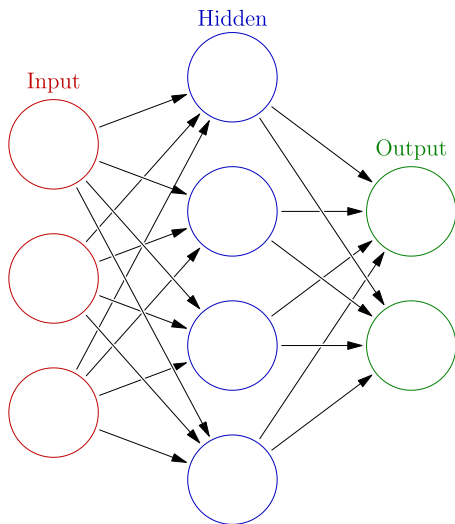
Some aspects of their behaviour are like their namesake!

- ▶ Learn pretty much any input-output data
- ▶ Uncover rules on their own about data
- ▶ Generalise from what they have learnt
- ▶ Cope with noise and damage

# How does an artificial neural network run?

By using maths, predictably!

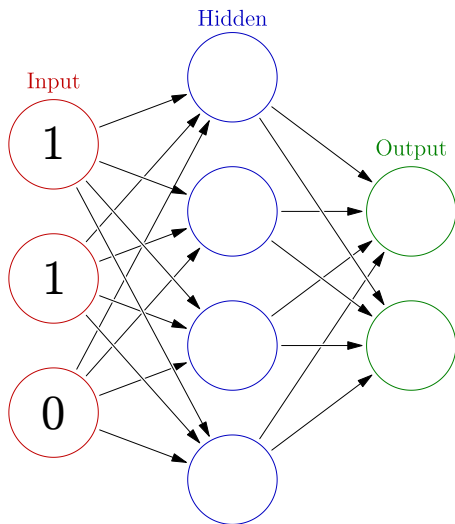
1. **Input units** are set to a *pattern*



# How does an artificial neural network run?

By using maths, predictably!

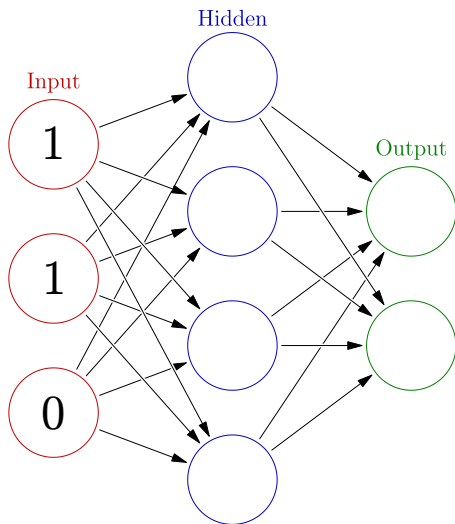
1. **Input units** are set to a *pattern*



# How does an artificial neural network run?

By using maths, predictably!

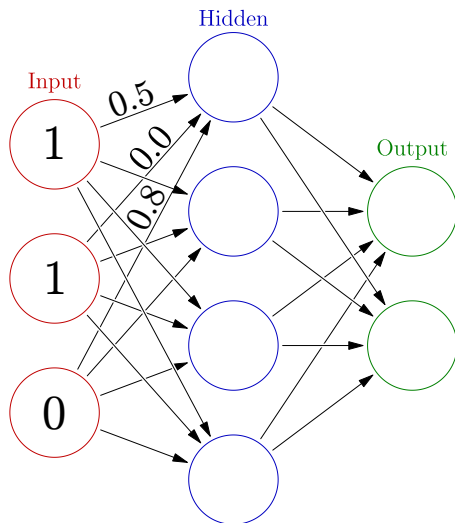
1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states



# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states

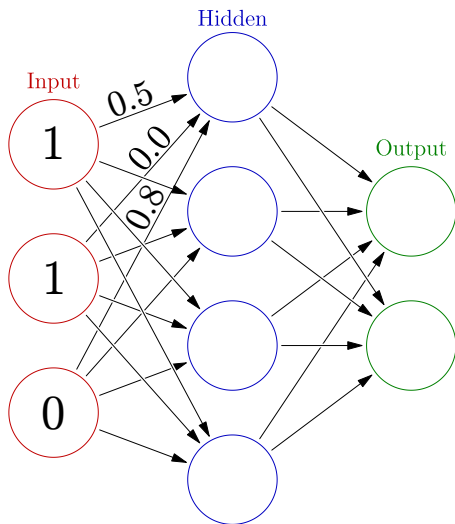


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states:

$$1 \times 0.5 = 0.5$$



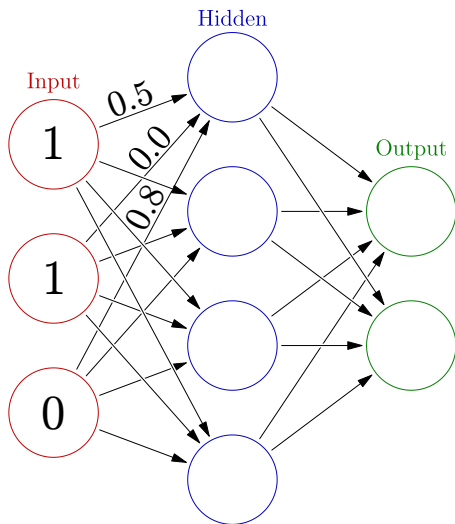
# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states:

$$1 \times 0.5 = 0.5$$

$$1 \times 0.0 = 0.0$$



# How does an artificial neural network run?

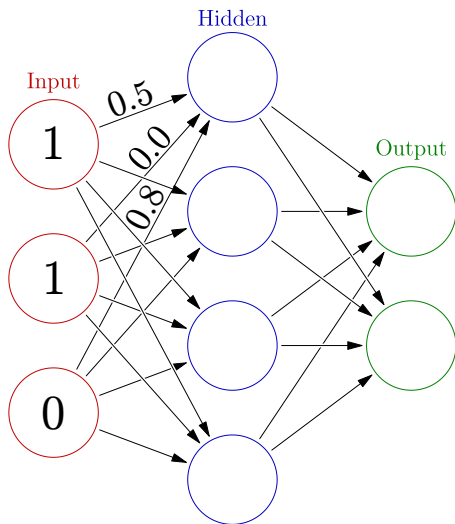
By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states:

$$1 \times 0.5 = 0.5$$

$$1 \times 0.0 = 0.0$$

$$0 \times 0.8 = 0.0$$



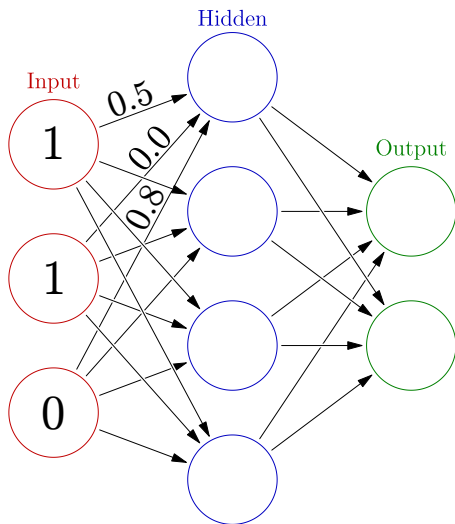


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states:

$$\begin{array}{rcl} 1 \times 0.5 & = & 0.5 \\ 1 \times 0.0 & = & 0.0 \\ 0 \times 0.8 & = & 0.0 \quad + \\ \hline \end{array}$$

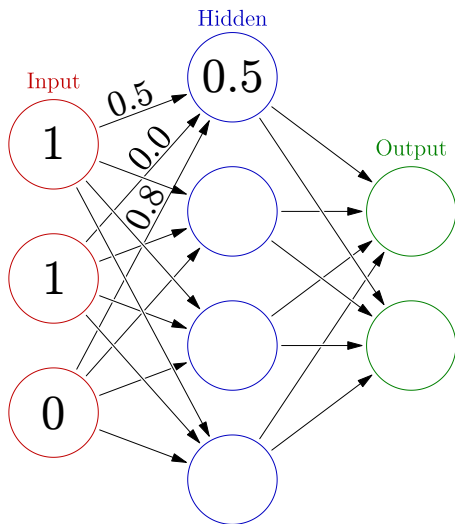


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states:

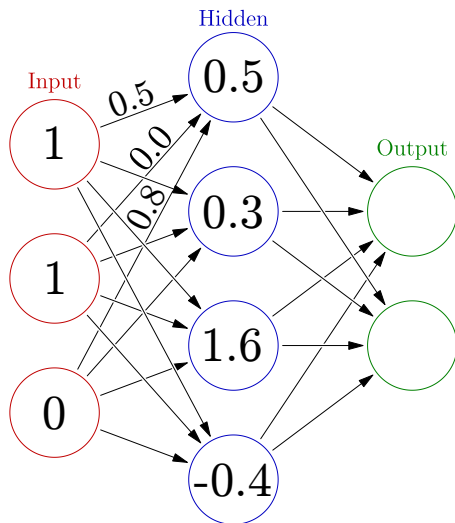
$$\begin{array}{rclcl} 1 \times 0.5 & = & 0.5 & & \\ 1 \times 0.0 & = & 0.0 & & \\ 0 \times 0.8 & = & 0.0 & + & \\ \hline & & 0.5 & & \end{array}$$



# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states

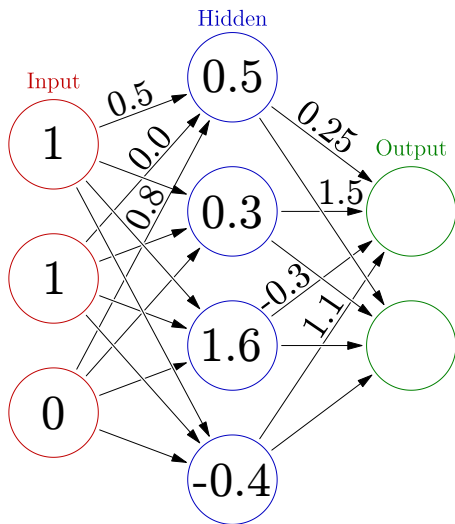




# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states
3. Same for **output units**

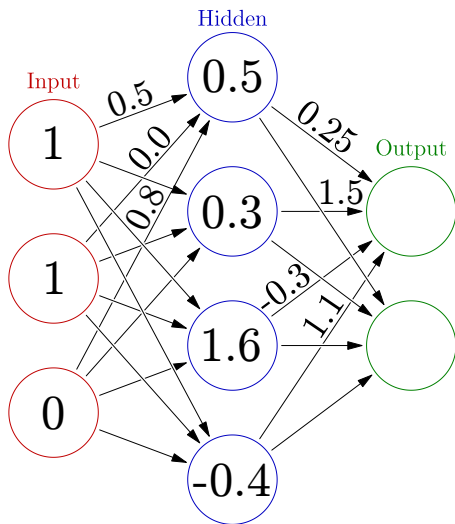


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states
3. Same for **output units**:

$$0.5 \times 0.25 = 0.125$$

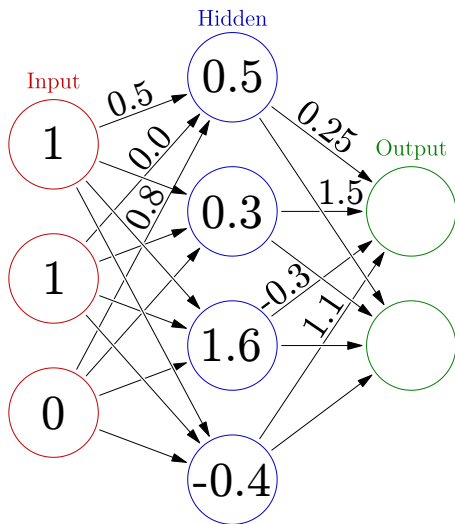


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states
3. Same for **output units**:

$$\begin{aligned} 0.5 \times 0.25 &= 0.125 \\ 0.3 \times 1.5 &= 0.45 \end{aligned}$$

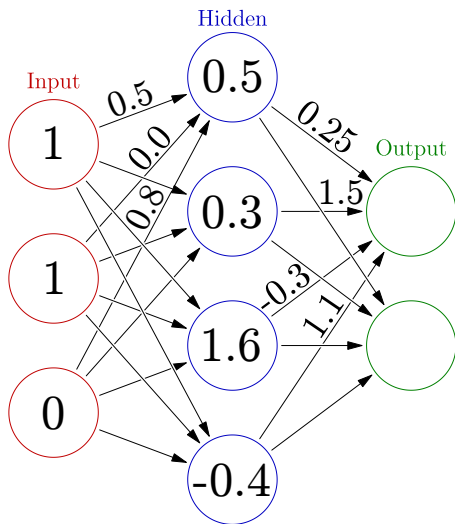


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states
3. Same for **output units**:

$$\begin{aligned}0.5 \times 0.25 &= 0.125 \\0.3 \times 1.5 &= 0.45 \\1.6 \times -0.3 &= -0.48\end{aligned}$$



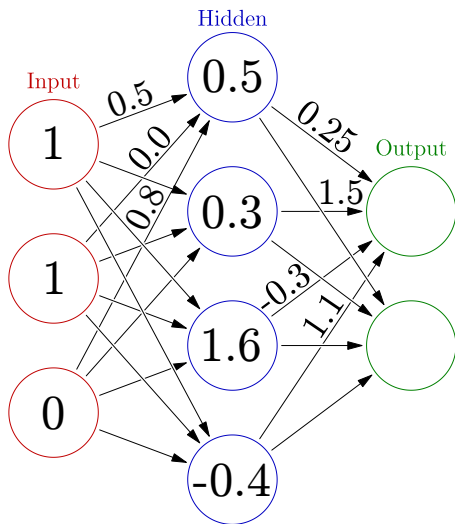


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states
3. Same for **output units**:

$$\begin{aligned} 0.5 \times 0.25 &= 0.125 \\ 0.3 \times 1.5 &= 0.45 \\ 1.6 \times -0.3 &= -0.48 \\ -0.4 \times 1.1 &= -0.44 \end{aligned}$$

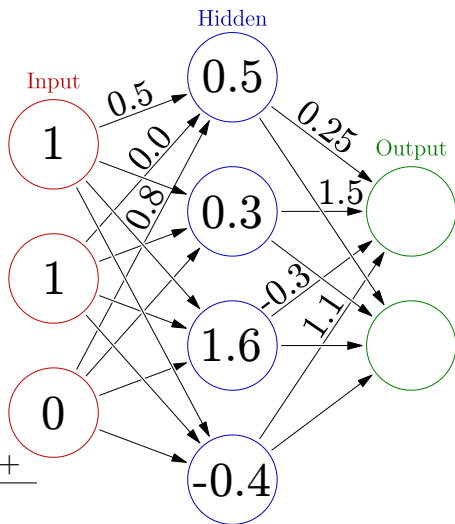


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states
3. Same for **output units**:

$$\begin{array}{rcl} 0.5 \times 0.25 & = & 0.125 \\ 0.3 \times 1.5 & = & 0.45 \\ 1.6 \times -0.3 & = & -0.48 \\ -0.4 \times 1.1 & = & -0.44 \quad + \\ \hline & & -0.345 \end{array}$$

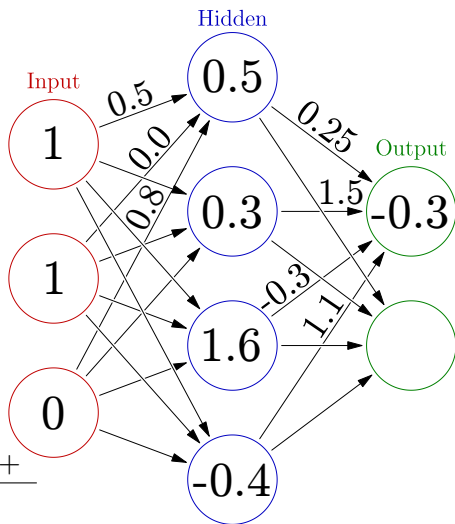


# How does an artificial neural network run?

By using maths, predictably!

1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states
3. Same for **output units**:

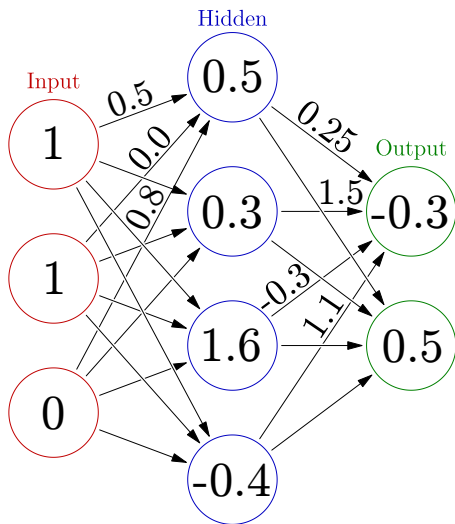
$$\begin{array}{rclcl} 0.5 \times 0.25 & = & 0.125 & & \\ 0.3 \times 1.5 & = & 0.45 & & \\ 1.6 \times -0.3 & = & -0.48 & & \\ -0.4 \times 1.1 & = & -0.44 & + & \\ \hline & & -0.345 & & \end{array}$$



# How does an artificial neural network run?

By using maths, predictably!

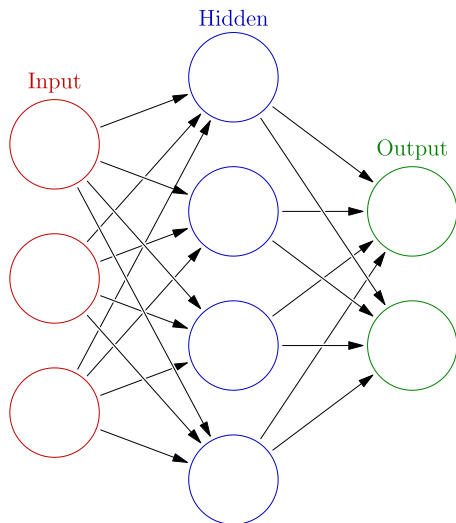
1. **Input units** are set to a *pattern*
2. Calculate **hidden units'** states
3. Same for **output units**



# How does an artificial neural network run?

By using maths, predictably!

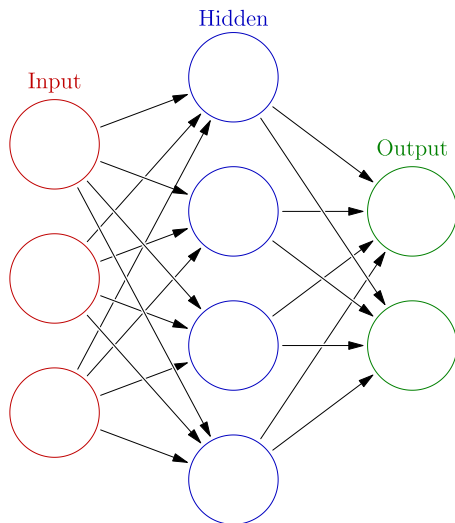
- ▶ But programmers are *lazy*!



# How does an artificial neural network run?

By using maths, predictably!

- General names save time

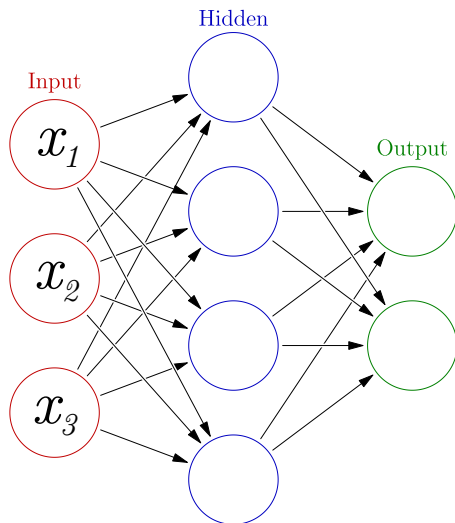


# How does an artificial neural network run?

By using maths, predictably!

- ▶ General names save time

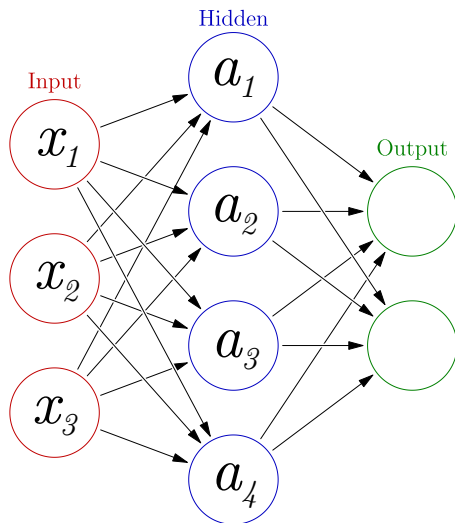
- ▶ **input units:**  $x_i$



# How does an artificial neural network run?

By using maths, predictably!

- ▶ General names save time
- ▶ **input units:**  $x_i$
- ▶ **hidden units:**  $a_j$

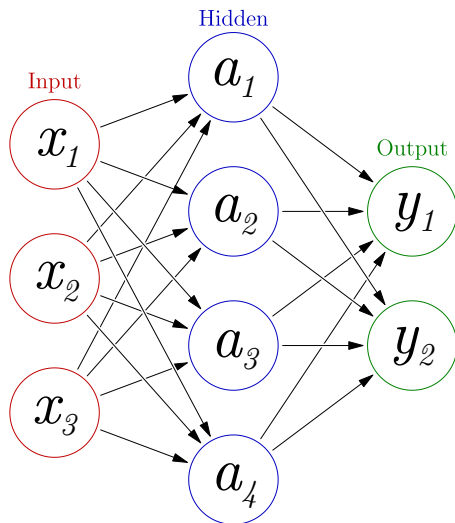




# How does an artificial neural network run?

By using maths, predictably!

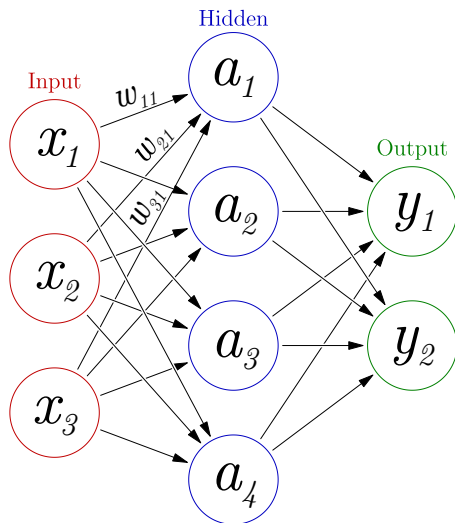
- ▶ General names save time
- ▶ **input units:**  $x_i$
- ▶ **hidden units:**  $a_j$
- ▶ **output units:**  $y_k$



# How does an artificial neural network run?

By using maths, predictably!

- ▶ General names save time
- ▶ **input units:**  $x_i$
- ▶ **hidden units:**  $a_j$
- ▶ **output units:**  $y_k$
- ▶ connection weights:  $w_{ij}$



# How does an artificial neural network run?

By using maths, predictably!

- ▶ General names save time

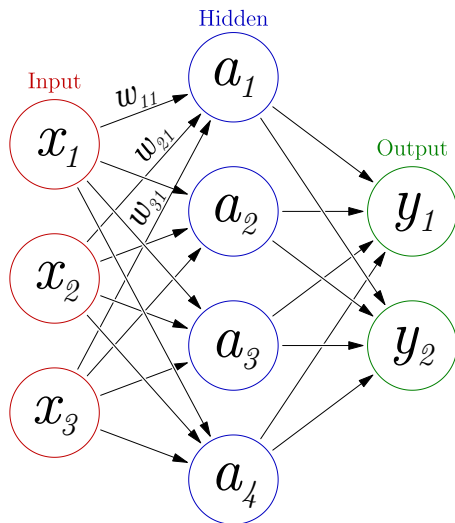
- ▶ **input units:**  $x_i$

- ▶ **hidden units:**  $a_j$

- ▶ **output units:**  $y_k$

- ▶ connection weights:  $w_{ij}$

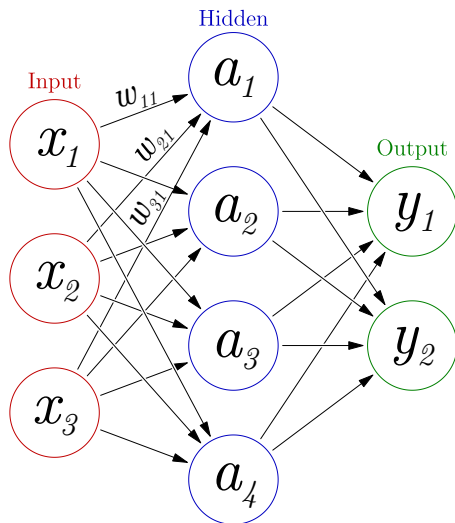
- ▶ subscripts  
general:  $ijklm\dots$   
specific: 12345...



# How does an artificial neural network run?

By using maths, predictably!

We use general names to write a general equation:

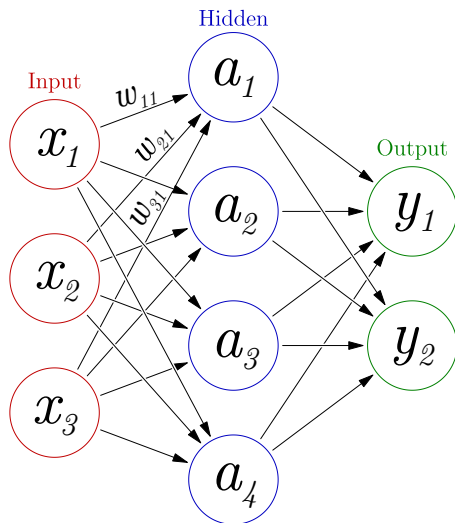


# How does an artificial neural network run?

By using maths, predictably!

We use general names to write a general equation:

$$a_i =$$

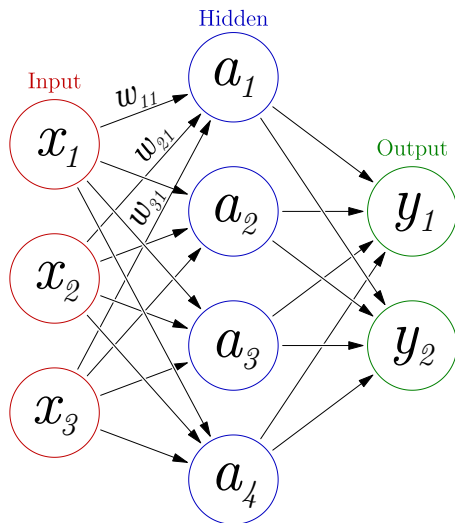


# How does an artificial neural network run?

By using maths, predictably!

We use general names to write a general equation:

$$a_i = x_j \times w_{ji}$$

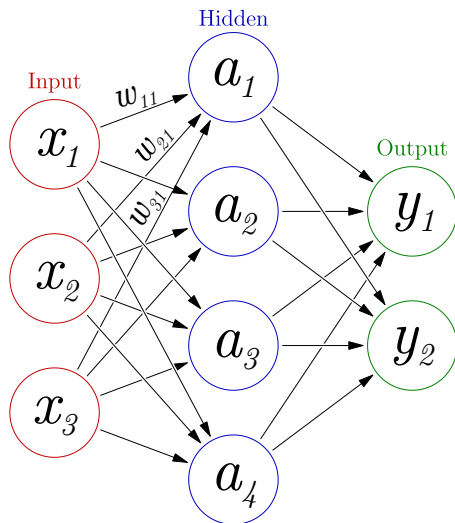


# How does an artificial neural network run?

By using maths, predictably!

We use general names to write a general equation:

$$a_i = \sum_{j=1}^N x_j \times w_{ji}$$

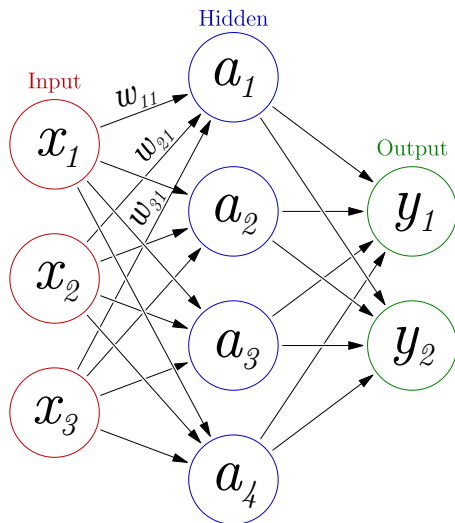


# How does an artificial neural network run?

By using maths, predictably!

We use general names to write a general equation:

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$





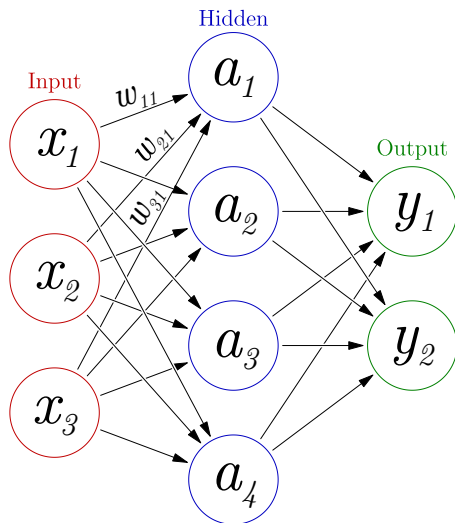
# How does an artificial neural network run?

By using maths, predictably!

We use general names to write a general equation:

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

where  $a_i$  is the unit whose state we want to calculate,  $N$  is the number of units on the previous layer,  $w_{ji}$  is the weight on the connection between  $i$  and  $j$ , and  $f$  is a function that the unit applies.

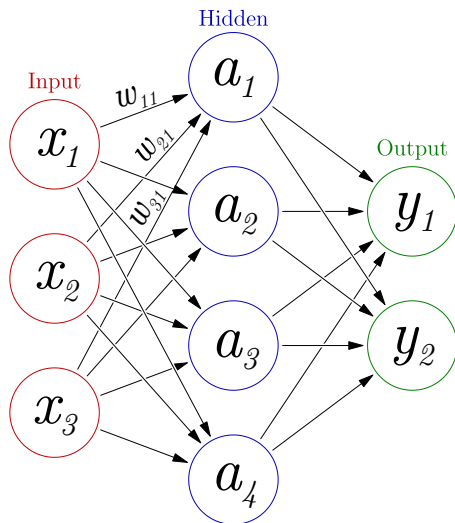


# How does an artificial neural network run?

By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$



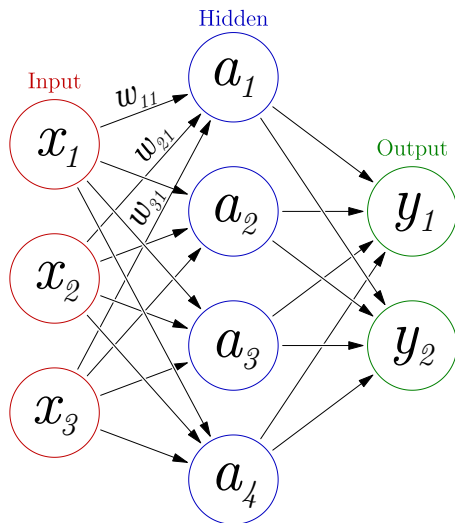
# How does an artificial neural network run?

By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\mathbf{1}} = f \left( \sum_{j=1}^N x_j \times w_{j\mathbf{1}} \right)$$



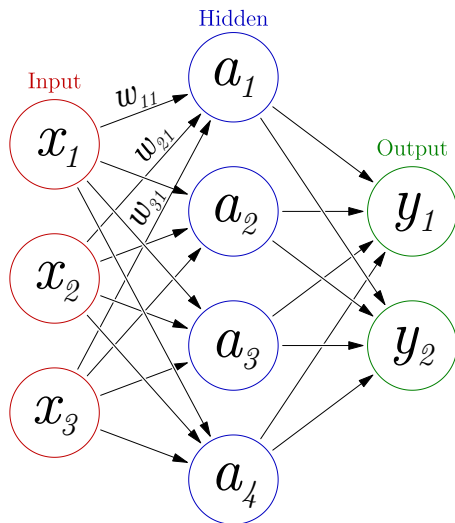
# How does an artificial neural network run?

By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\textcolor{blue}{1}} = f \left( \sum_{j=1}^{\textcolor{red}{3}} x_j \times w_{j\textcolor{blue}{1}} \right)$$



# How does an artificial neural network run?

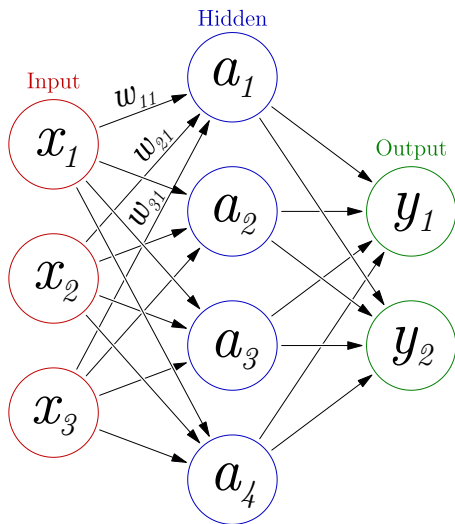
By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\textcolor{blue}{1}} = f \left( \sum_{j=1}^{\textcolor{red}{3}} x_j \times w_{j\textcolor{blue}{1}} \right)$$

$$x_{\textcolor{red}{1}} \times w_{\textcolor{blue}{1}\textcolor{blue}{1}}$$



# How does an artificial neural network run?

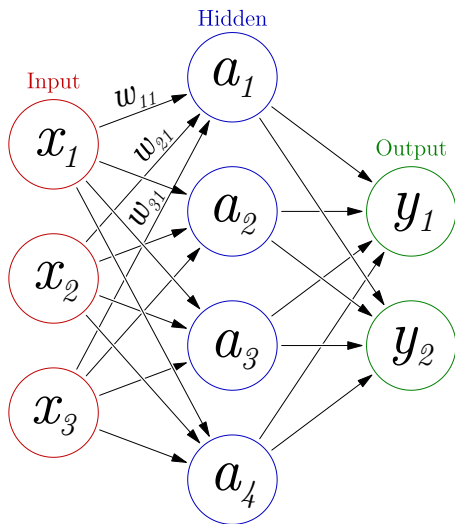
By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\textcolor{blue}{1}} = f \left( \sum_{j=1}^{\textcolor{red}{3}} x_j \times w_{j\textcolor{blue}{1}} \right)$$

$$x_{\textcolor{red}{1}} \times w_{\textcolor{blue}{1}\textcolor{red}{1}} + x_{\textcolor{red}{2}} \times w_{\textcolor{blue}{2}\textcolor{red}{1}}$$



# How does an artificial neural network run?

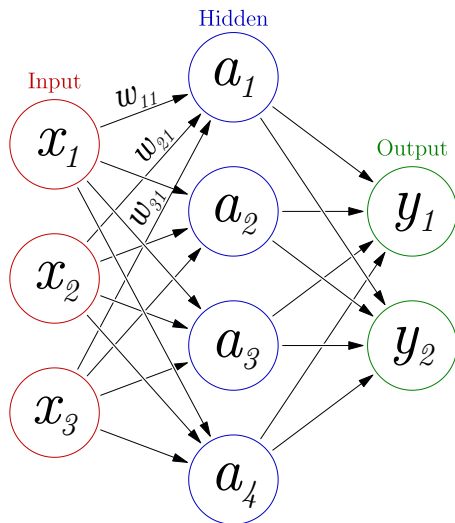
By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\textcolor{blue}{1}} = f \left( \sum_{j=1}^{\textcolor{red}{3}} x_j \times w_{j\textcolor{blue}{1}} \right)$$

$$x_{\textcolor{red}{1}} \times w_{\textcolor{red}{1}\textcolor{blue}{1}} + x_{\textcolor{red}{2}} \times w_{\textcolor{red}{2}\textcolor{blue}{1}} + x_{\textcolor{red}{3}} \times w_{\textcolor{red}{3}\textcolor{blue}{1}}$$



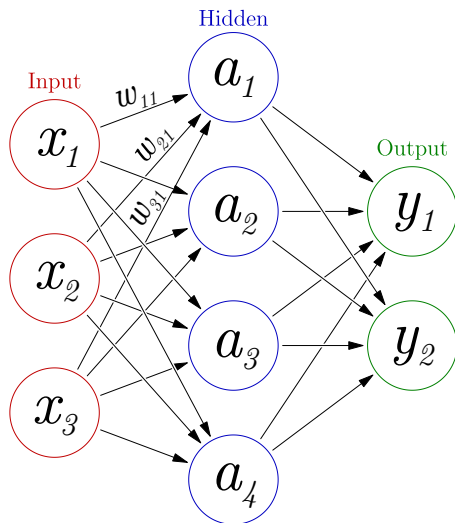
# How does an artificial neural network run?

By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\textcolor{blue}{2}} = f \left( \sum_{j=1}^{\textcolor{red}{3}} x_j \times w_{j\textcolor{blue}{2}} \right)$$





# How does an artificial neural network run?

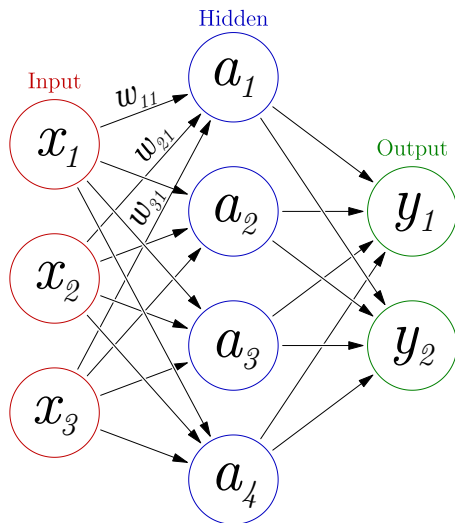
By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\textcolor{blue}{2}} = f \left( \sum_{j=1}^{\textcolor{red}{3}} x_j \times w_{j\textcolor{blue}{2}} \right)$$

$$x_{\textcolor{red}{1}} \times w_{\textcolor{blue}{1}\textcolor{blue}{2}}$$



# How does an artificial neural network run?

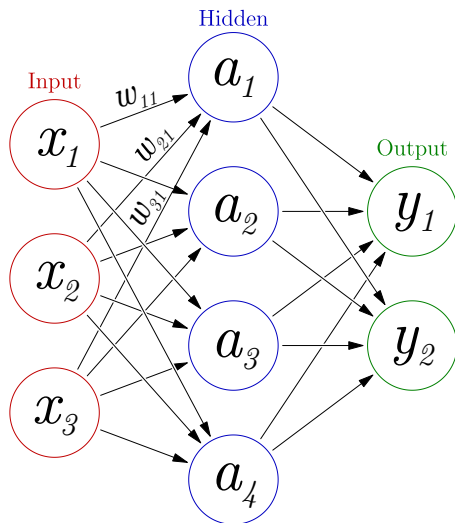
By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\textcolor{blue}{2}} = f \left( \sum_{j=1}^{\textcolor{red}{3}} x_j \times w_{j\textcolor{blue}{2}} \right)$$

$$x_{\textcolor{red}{1}} \times w_{\textcolor{blue}{1}\textcolor{blue}{2}} + x_{\textcolor{red}{2}} \times w_{\textcolor{blue}{2}\textcolor{blue}{2}}$$



# How does an artificial neural network run?

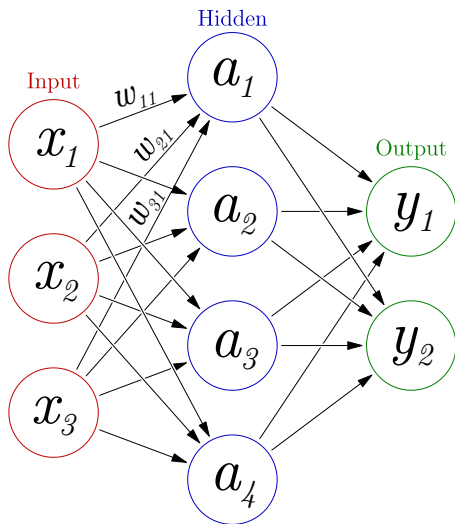
By using maths, predictably!

We use this equation by replacing *iterators*  $i$  and  $j$ :

$$a_i = f \left( \sum_{j=1}^N x_j \times w_{ji} \right)$$

$$a_{\textcolor{blue}{2}} = f \left( \sum_{j=1}^{\textcolor{red}{3}} x_j \times w_{j\textcolor{blue}{2}} \right)$$

$$x_{\textcolor{red}{1}} \times w_{\textcolor{blue}{1}\textcolor{blue}{2}} + x_{\textcolor{red}{2}} \times w_{\textcolor{blue}{2}\textcolor{blue}{2}} + x_{\textcolor{red}{3}} \times w_{\textcolor{blue}{3}\textcolor{blue}{2}}$$



# How do networks learn?

Cunning!

- ▶ Many options: Hebbian learning, back-propagation of error, Boltzmann machine learning, self-organising map algorithm, etc.

# How do networks learn?

Cunning!

- ▶ Many options: Hebbian learning, back-propagation of error, Boltzmann machine learning, self-organising map algorithm, etc.
- ▶ All learning algorithms work by changing the connection weights

# How do networks learn?

Cunning!

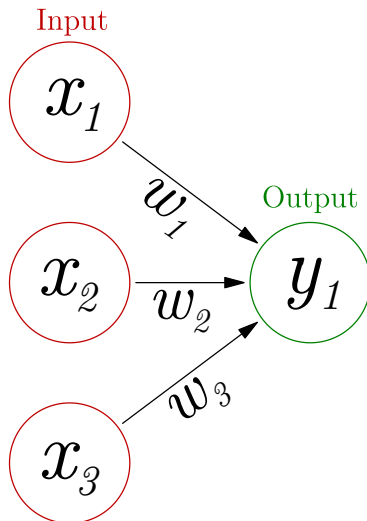
- ▶ Many options: Hebbian learning, back-propagation of error, Boltzmann machine learning, self-organising map algorithm, etc.
- ▶ All learning algorithms work by changing the connection weights
- ▶ Learning can be divided into *supervised*, *unsupervised*, and *reinforcement*

# Hebbian learning

A very simple learning rule

“Cells that fire together, wire together”

— Carla Shatz



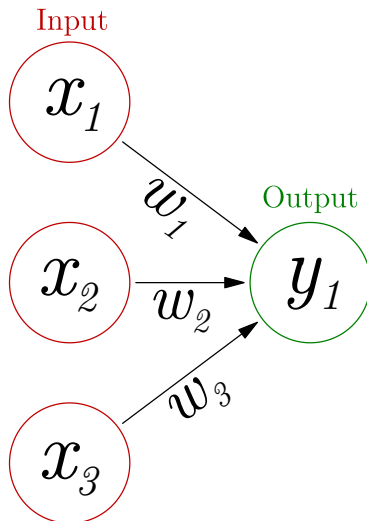
# Hebbian learning

A very simple learning rule

“Cells that fire together, wire together”

— Carla Shatz

$$w_i =$$





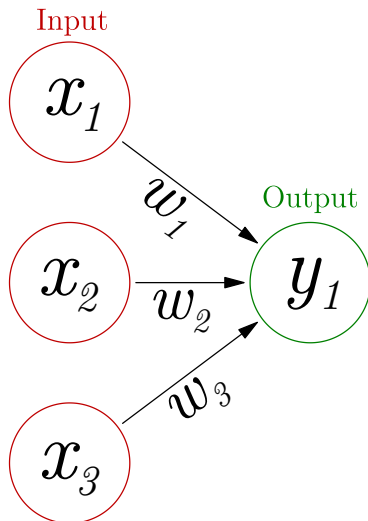
# Hebbian learning

A very simple learning rule

“Cells that fire together, wire together”

— Carla Shatz

$$w_i = x_i \times y_j$$



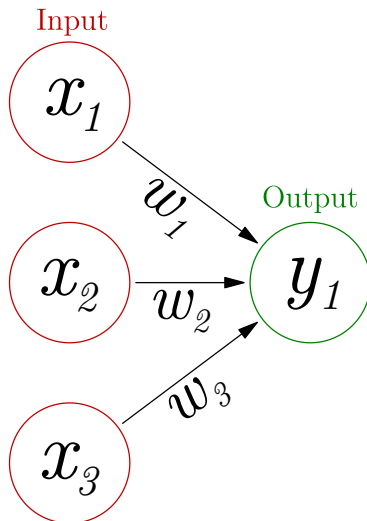
# Hebbian learning

A very simple learning rule

“Cells that fire together, wire together”

— Carla Shatz

$$w_i = \eta \times x_i \times y_j$$



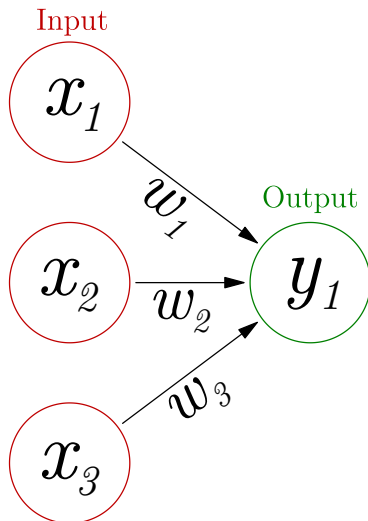
# Hebbian learning

A very simple learning rule

“Cells that fire together, wire together”

— Carla Shatz

$$\Delta w_i = \eta \times x_i \times y_j$$



# Hebbian learning

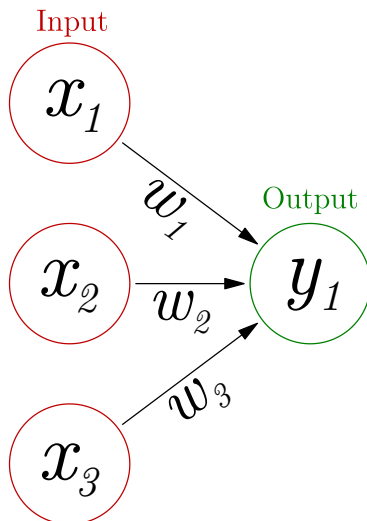
A very simple learning rule

“Cells that fire together, wire together”

— Carla Shatz

$$\Delta w_i = \eta \times \textcolor{red}{x}_i \times \textcolor{green}{y}_j$$

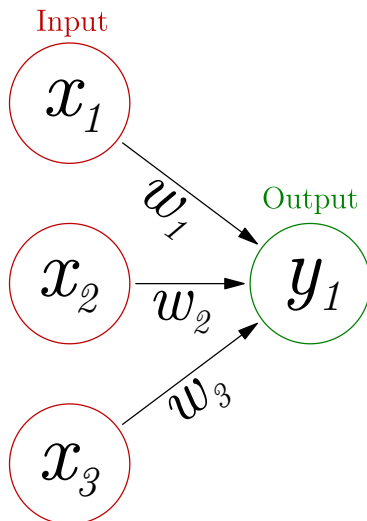
which means each weight is changed by a small in/decrement for every pattern



# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but very *unstable*!

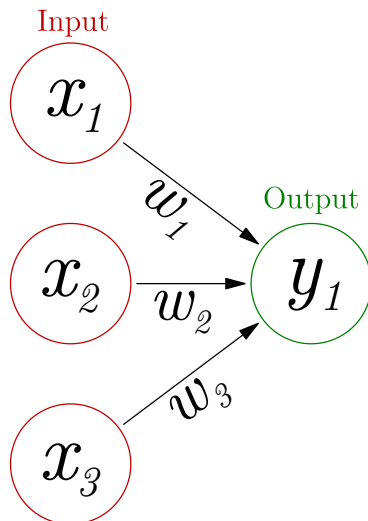


# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but very *unstable*!

$$\Delta w_i = \eta \times x_i \times y_j$$



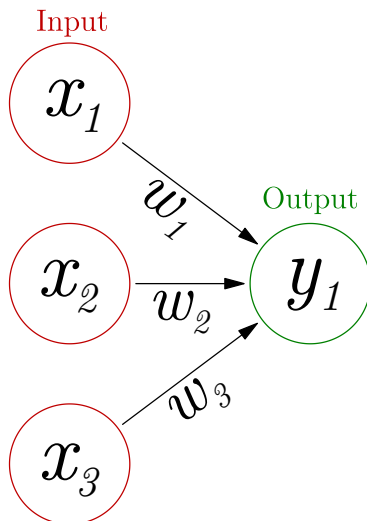
# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = \eta \times x_i \times y_j$$



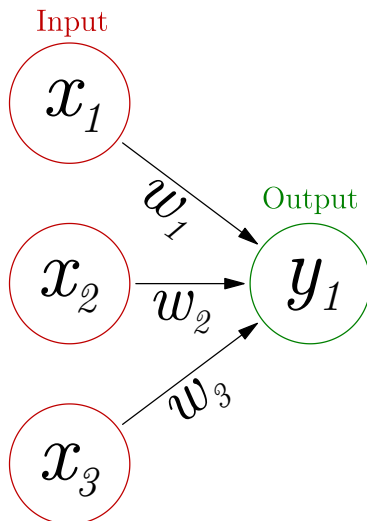
# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.5 \times x_i \times y_j$$





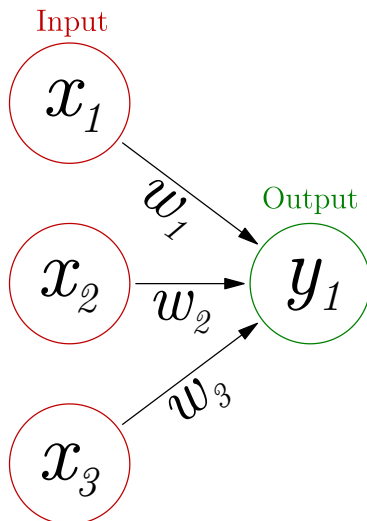
# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.5 \times x_1 \times y_j$$



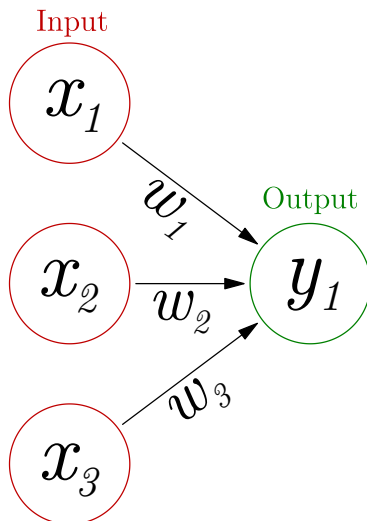
# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.5 \times x_1 \times y_1$$



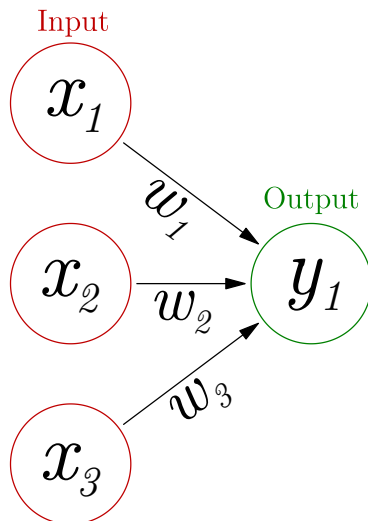
# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.5 \times 1.0 \times y_1$$



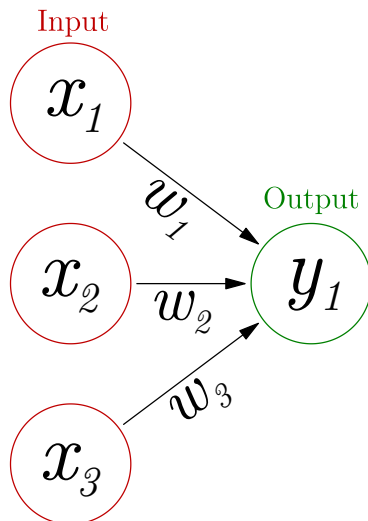
# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.5 \times 1.0 \times 0.3$$



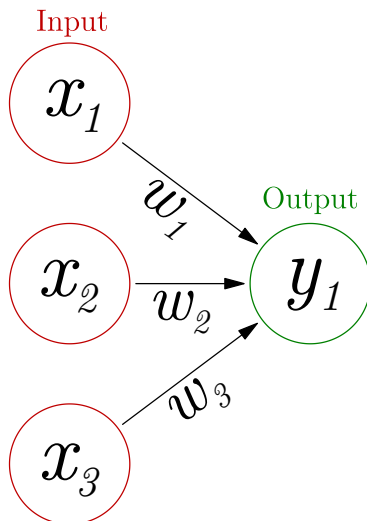
# Hebbian learning

“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.15$$



# Hebbian learning

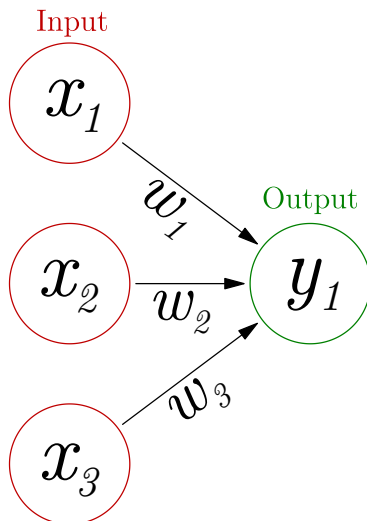
“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.15$$

$$\mathbf{new} \ w_1 = \mathbf{old} \ w_1 + \Delta w_1$$



# Hebbian learning

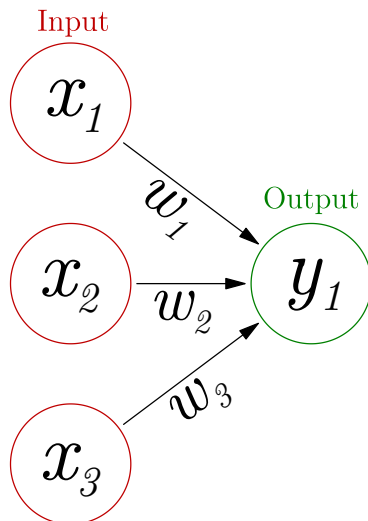
“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.15$$

$$\mathbf{new} \ w_1 = 0.0 + \Delta w_1$$



# Hebbian learning

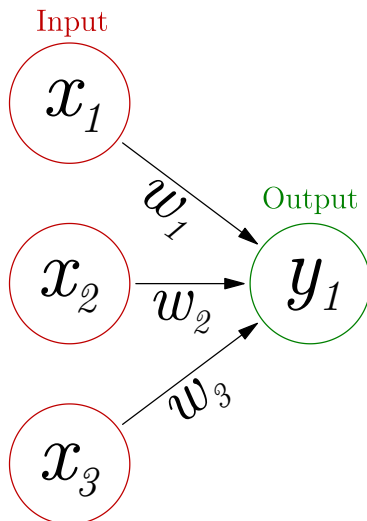
“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.15$$

$$\text{new } w_1 = 0.0 + 0.15$$





# Hebbian learning

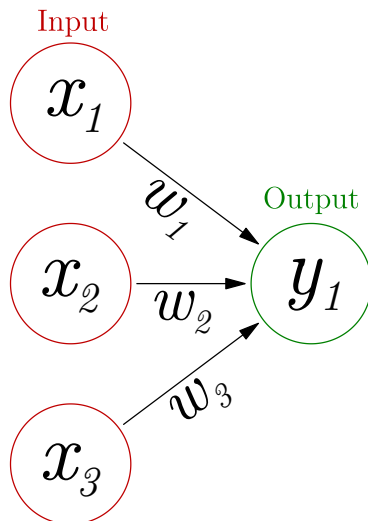
“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.15$$

$$\text{new } w_1 = 0.15$$



# Hebbian learning

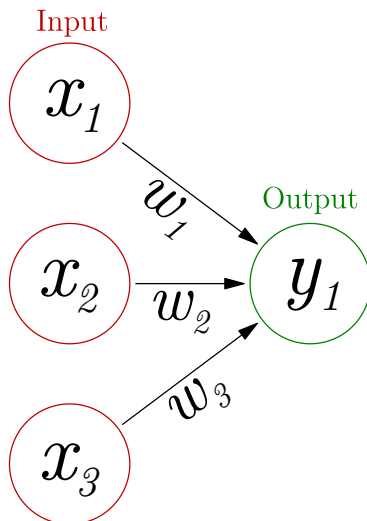
“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.15$$

$$w_1 = 0.15$$



# Hebbian learning

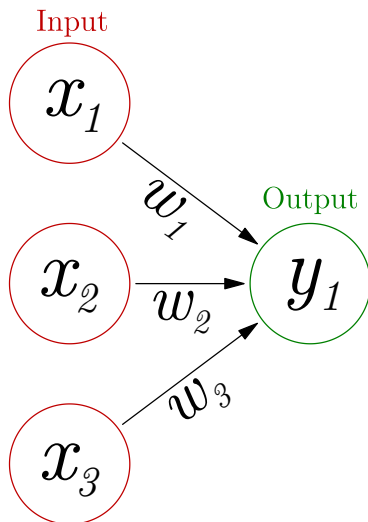
“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.15$$

$$w_1 = 0.15 + \text{something positive}$$



# Hebbian learning

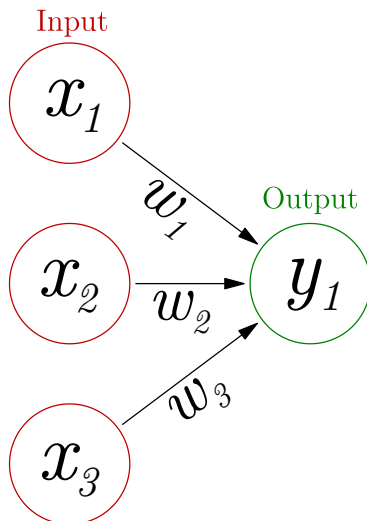
“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

$$\Delta w_1 = 0.15$$

$w_1 = 0.15 + \text{something}$   
positive + something else  
positive +



# Hebbian learning

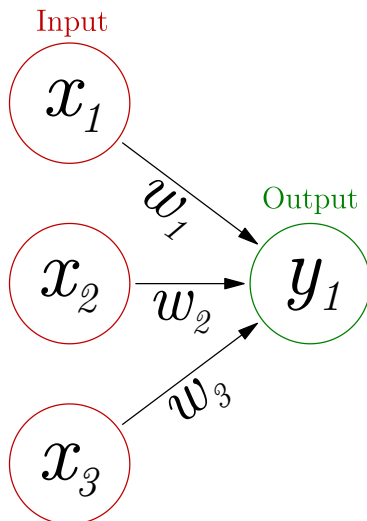
“Cells that fire together, wire together” — Carla Shatz

Hebb's rule is simple, but *very unstable!*

$$\Delta w_i = \eta \times x_i \times y_j$$

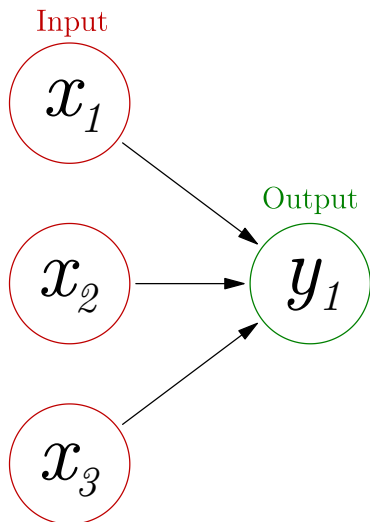
$$\Delta w_1 = 0.15$$

$w_1 = 0.15 + \text{something}$   
positive + something else  
positive + another positive  
value + ...



# The perceptron

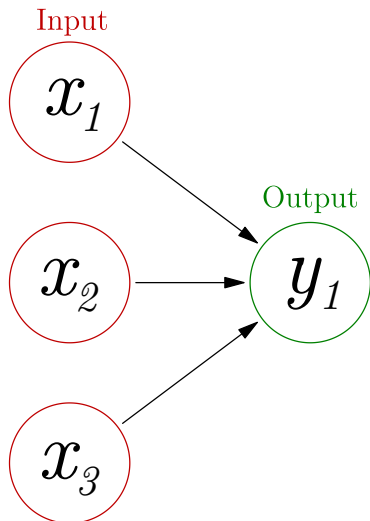
A simple classifier



# The perceptron

A simple classifier

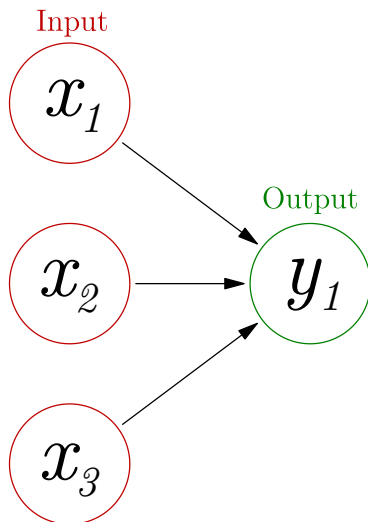
- Created in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt



# The perceptron

A simple classifier

- ▶ Created in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt
- ▶ Linear classifier

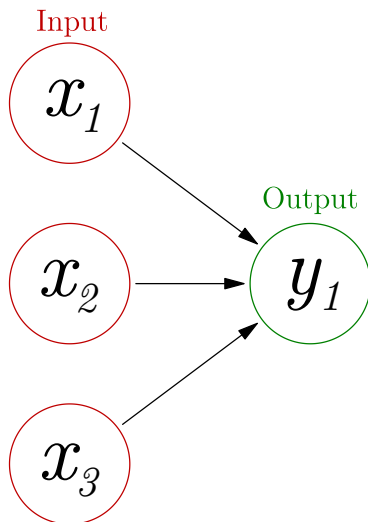




# The perceptron

## A simple classifier

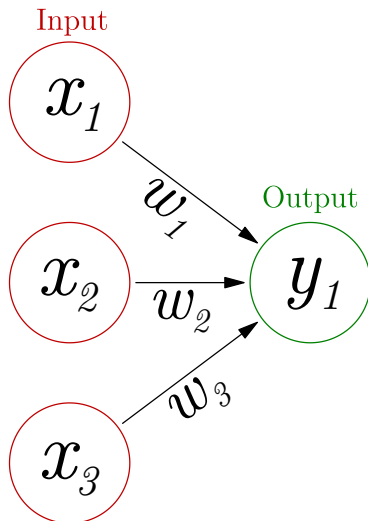
- ▶ Created in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt
- ▶ Linear classifier
- ▶ Simplest form of feedforward network



# How does the perceptron learn?

Maths again!

1. Initialise weights



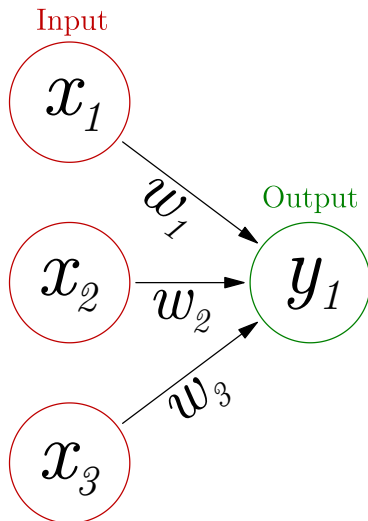
# How does the perceptron learn?

Maths again!

1. Initialise weights
2. Run network using:

$$y_j = f\left(\sum_1^N w_i \times x_i\right)$$

same as always!

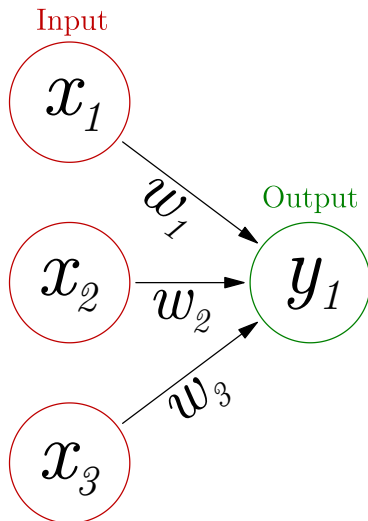


# How does the perceptron learn?

Maths again!

1. Initialise weights
2. Run network
3. Update weights using:

$$\Delta w_i = \eta \quad y_j \times x_i$$

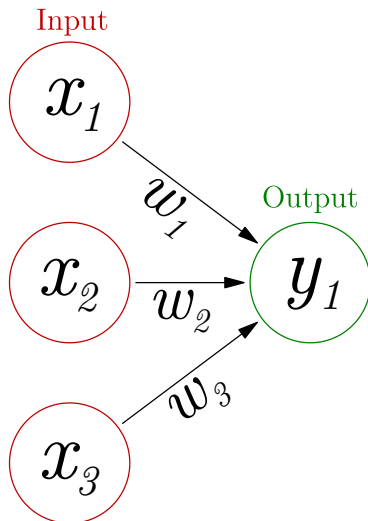


# How does the perceptron learn?

Maths again!

1. Initialise weights
2. Run network
3. Update weights using:

$$\Delta w_i = \eta (d_j - y_j) \times x_i$$



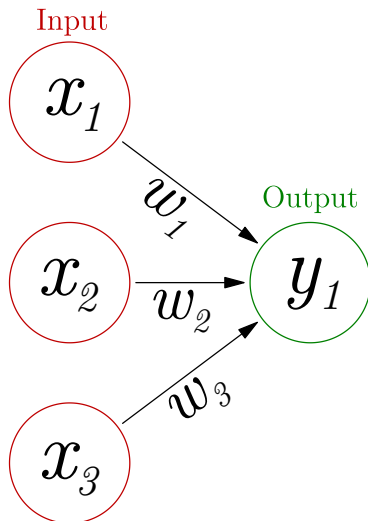
# How does the perceptron learn?

Maths again!

1. Initialise weights
2. Run network
3. Update weights using:

$$\Delta w_i = \eta (d_j - y_j) \times x_i$$

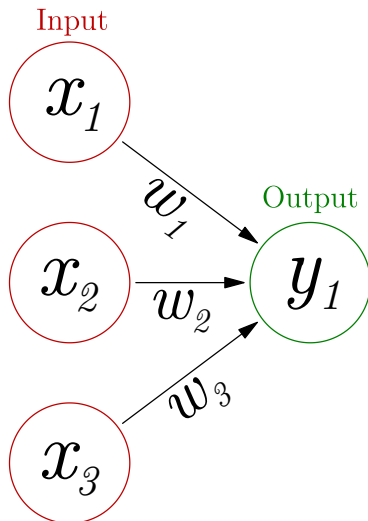
where  $d$  is what we want  $y$  to be given  $x$ , and  $\eta$  is the learning rate.



# How does the perceptron learn?

Maths again!

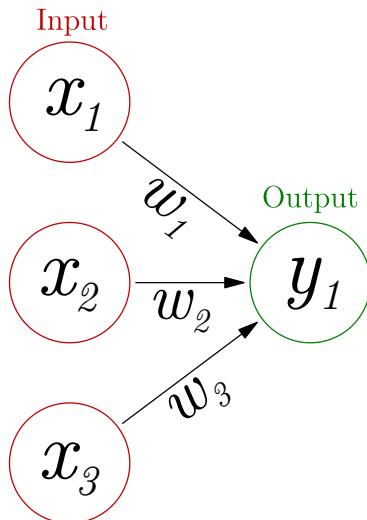
1. Initialise weights
2. Run network
3. Update weights
4. Repeat 2 and 3



# How does the perceptron learn?

Maths again!

1. Initialise weights
2. Run network
3. Update weights
4. Repeat 2 and 3
5. When do we stop?





The end

Time to program a perceptron!