

# Welcome to the computational cognitive modelling workshop!

## **Part 2: Artificial neural networks**

# Part 2: **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
- ▶ A set of *units*, connected by *weights*
- ▶ The network *runs* by passing *activations* from the *input* (to the *hidden*) to the *output* units

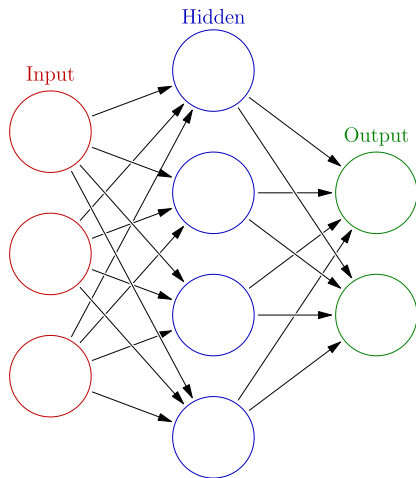


Figure: Glosser.ca / CC-BY-SA-3.0

# Why use artificial neural networks for modelling?

Some aspects of their behaviour are like their namesake!

- ▶ Learn pretty much any input-output data
- ▶ 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!

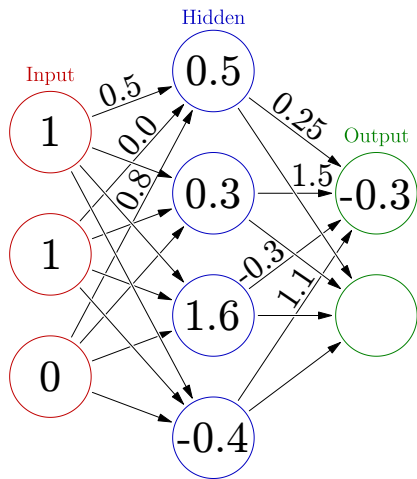
Input units are set to a *pattern*

Calculate hidden units' states:

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

Same for output units:

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



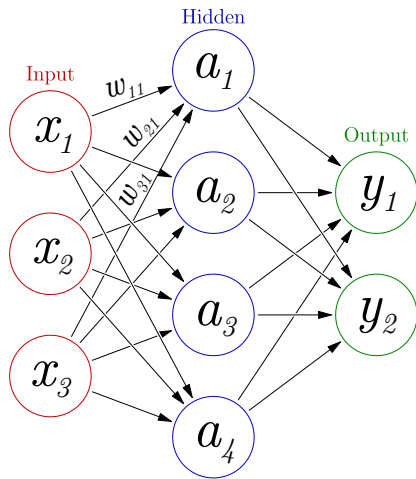
# How does an artificial neural network run?

By using maths, predictably!

But we/programmers are lazy:

$$a_i = f \left( \sum_1^N x_j \times w_{ji} \right)$$

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



# How do networks learn?

Cunning!

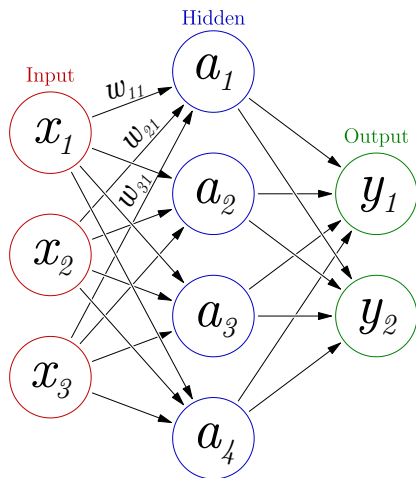
- ▶ Many options: Hebbian learning, back-propagation of error, Boltzmann machine learning, self-organising map algorithm, etc.
- ▶ All learning algorithms work by changing the connection weights
- ▶ Learning can be divided into *supervised*, *unsupervised*, and *reinforcement*

# Hebbian learning

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

$$\Delta w_{ij} = \eta \sum_i^N x_i \times a_j$$

which means each weight,  $w_{ij}$  is changed by a small in/decrement





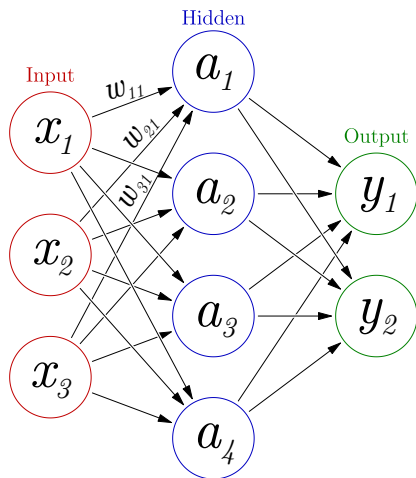
# Hebbian learning

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

$$\Delta w_{ij} = \eta \sum_i x_i \times a_j$$

which means each weight,  $w_{ij}$  is changed by a small in/decrement

If  $x_i$  or  $a_j$  is on, then the other will also be on



# Hebbian learning

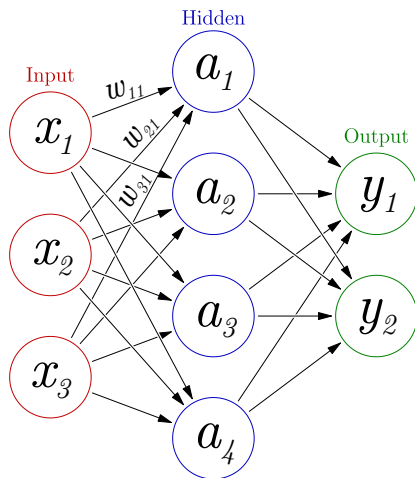
“Cells that fire together, wire together” — Carla Shatz

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

$$\Delta w_{ij} = \eta \sum_i x_i \times a_j$$

which means each weight,  $w_{ij}$  is  
changed by a small in/decrement

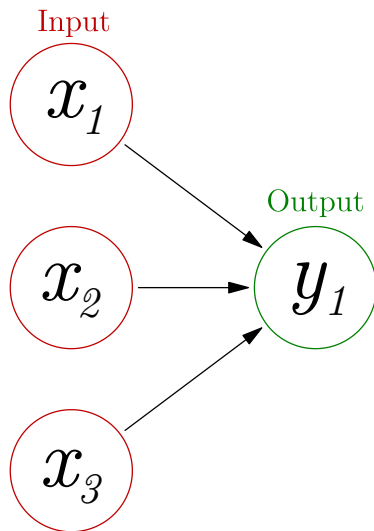
If  $x_i$  or  $a_j$  is on, then the other  
will also be on



# The perceptron

## A simple classifier

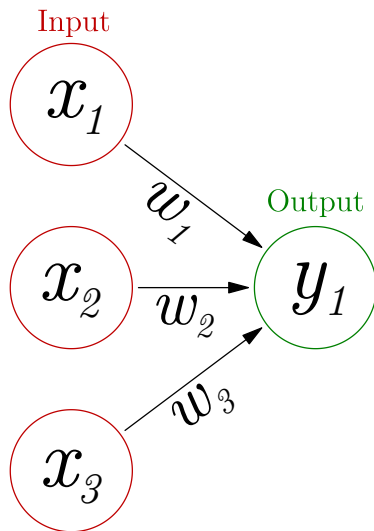
- ▶ Created in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt
- ▶ Linear classifier
- ▶ Simplest form of feedforward network



# How does the perceptron learn?

Maths again!

1. Initialise weights



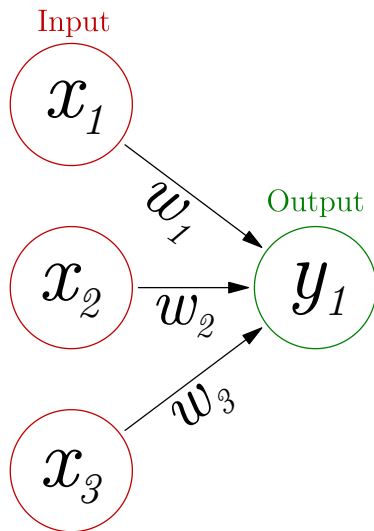
# How does the perceptron learn?

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!



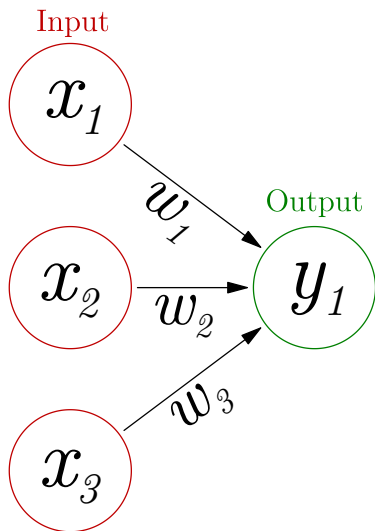
# How does the perceptron learn?

Maths again!

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

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

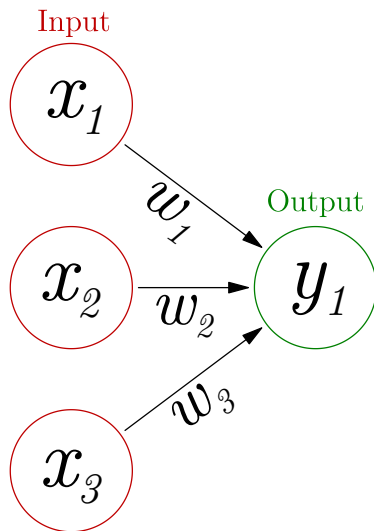
where  $d$  is what we want  $y$  to be given  $x_i$ , and  $\eta$  is the learning rate.



# How does the perceptron learn?

Maths again!

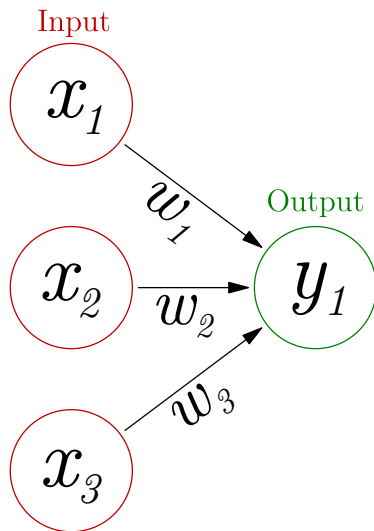
1. Initialise weights
2. Run network
3. Update weights
4. Repeat 2 and 3



# 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!