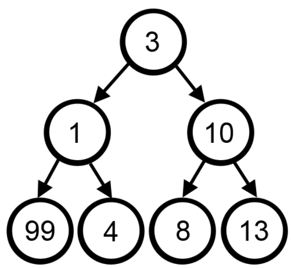
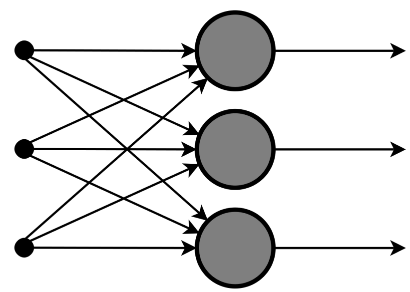
# CSE 40171:
# Artificial Intelligence

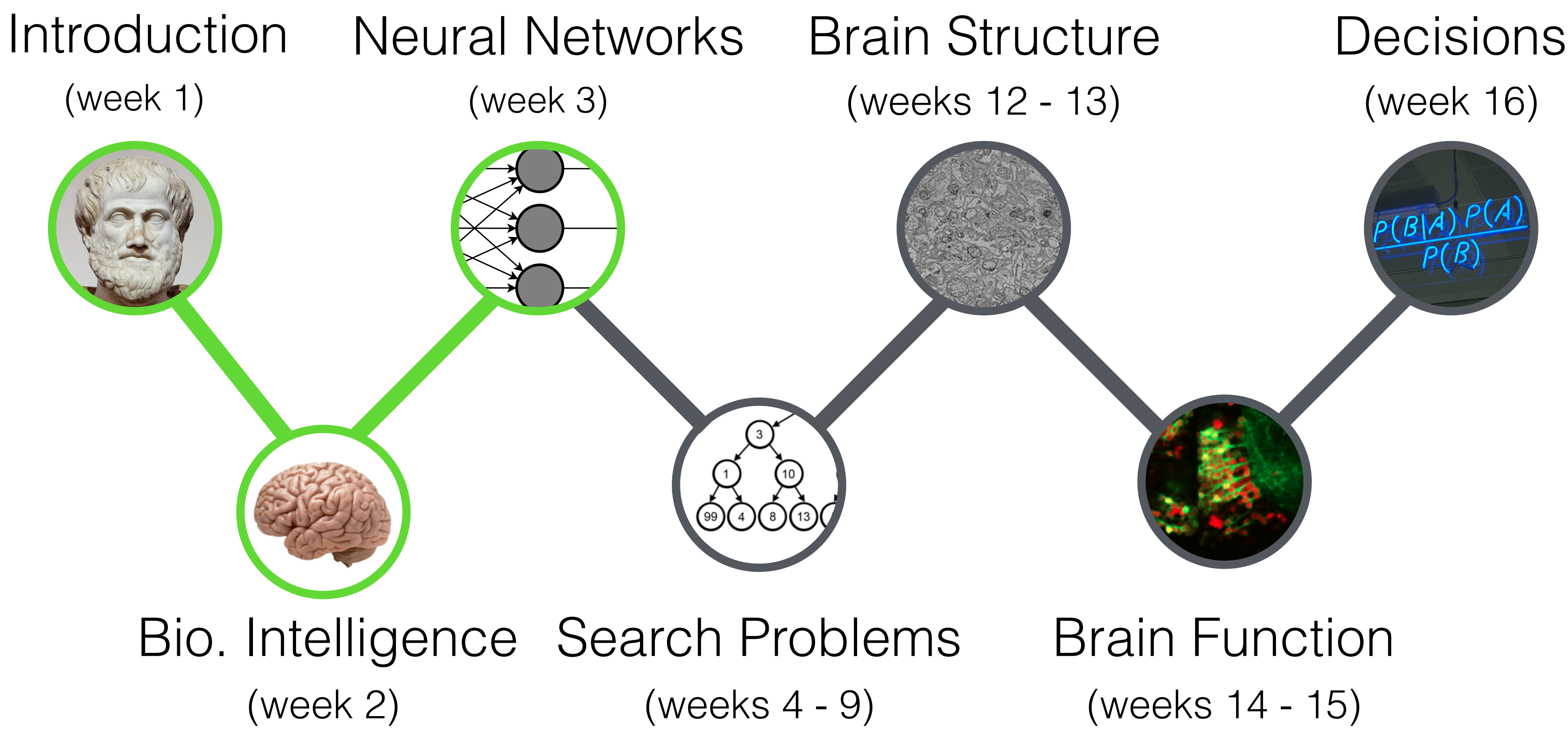


## Artificial Neural Networks: Structure of Neural Networks

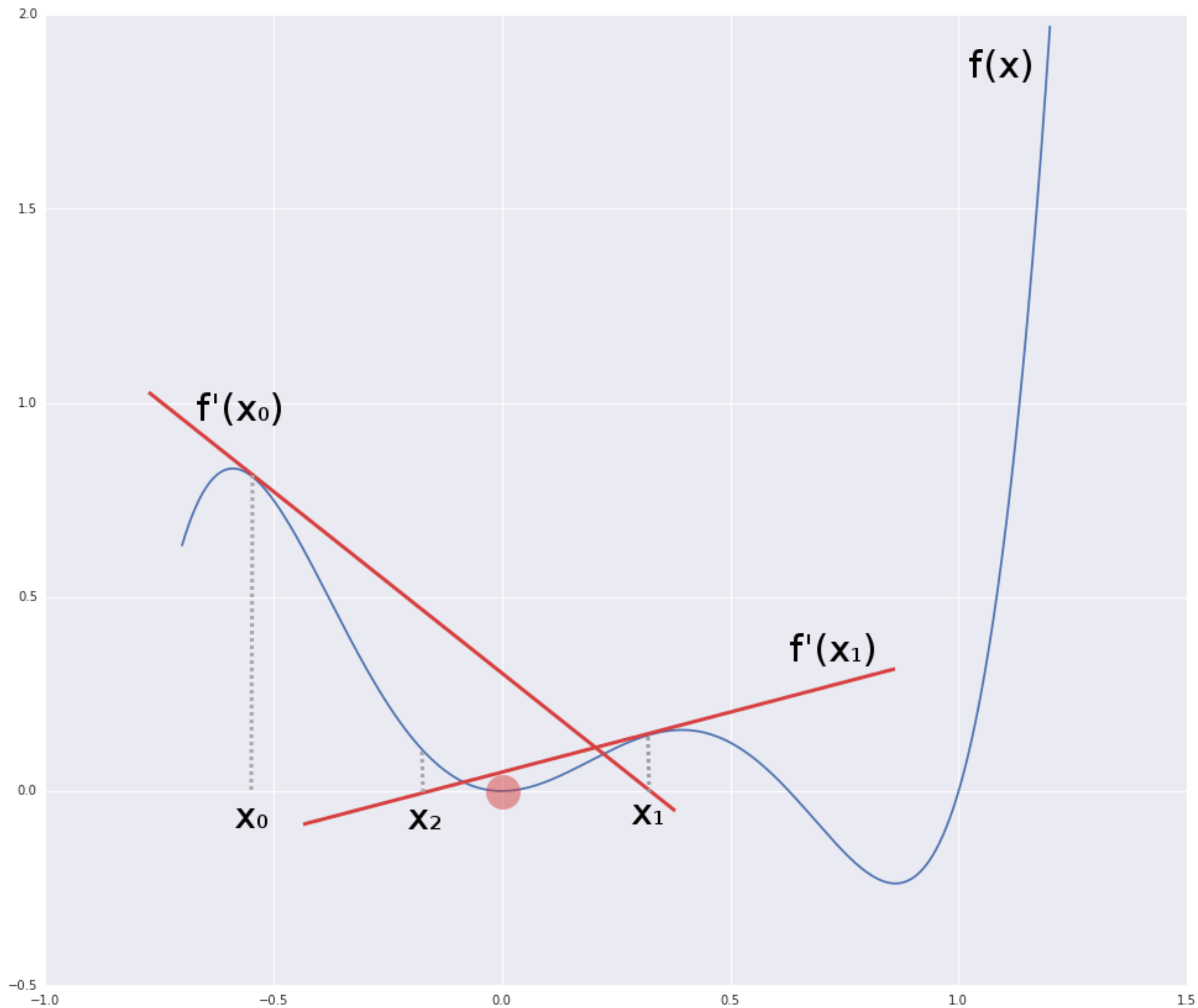
Homework #1 has been released
It is due at 11:59PM on 9/16

# Course Roadmap



# Numerical Approximation

$$f(x) = 6x^5 - 5x^4 - 4x^3 + 3x^2$$




Slide Credit: http://danielhomola.com/2016/02/09/newtons-method-with-10-lines-of-python/

A function exists. We don't know it. We have data to learn it.

# Components of Learning

Input: $\mathrm{x} \in \mathbb{R}^d = \mathcal{X}$

Output: $y \in -1, +1 = \mathcal{Y}$

Target Function: $f : \mathcal{X} \mapsto \mathcal{Y}$

(the target $f$ is *unknown*)

Data set: $\mathcal{D} = (\mathrm{x}_1, y_1), \ldots, (\mathrm{x}_N, y_N)$

($y_n = f(\mathbf{x}_n)$ is *unknown*)

$\mathcal{X}$ $\mathcal{Y}$ and $\mathcal{D}$ are given by the learning problem;

The target $f$ is fixed but unknown

We learn the function $f$ from the data $\mathcal{D}$

Slide credit: Abu-Mostafa, Magdon-Ismail, Lin, Learning from Data

# Components of Learning

Start with a set of candidate hypotheses $\mathcal{H}$ that you think are likely to represent $f$.

$$\mathcal{H} = \{h_1, h_2, \ldots, \}$$

is called the hypothesis set or **model.**

Select a hypothesis $g$ from $\mathcal{H}$. The way we do this is called a **learning algorithm**.

We hope $g \approx f$

We choose $\mathcal{H}$ and the learning algorithm

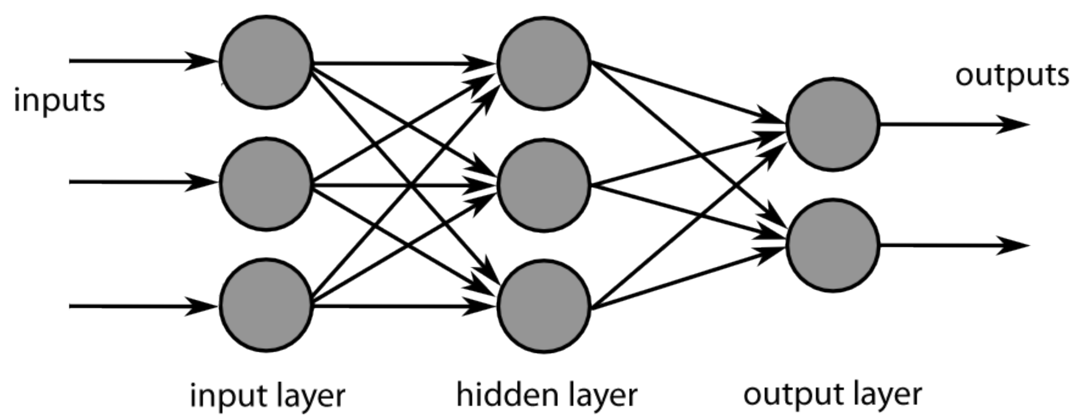
Slide credit: Abu-Mostafa, Magdon-Ismail, Lin, Learning from Data

# Basic Artificial Neural Networks

# Artificial Neural Networks

Artificial "neurons" are connected together to form a network

- Contains sets of adaptive weights
  - These are learned during training
- Capable of approximating non-linear functions of their input




A Neural network with two layers 

# Neural Network Structures

**Neural networks are composed of:**

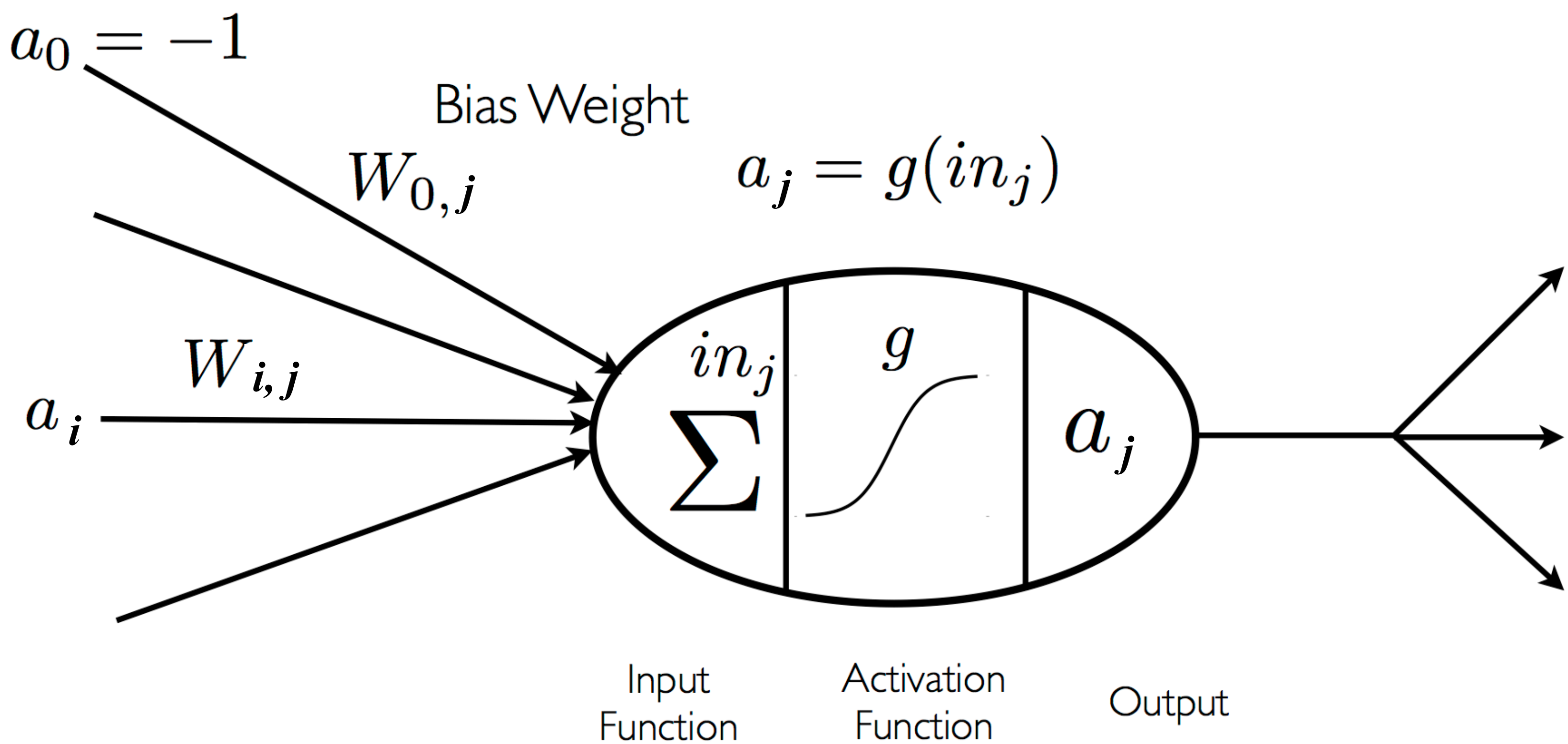
Units (vertices in a graph)

Links (connecting unit $i$ to unit $j$)

Activations (propagating signals from $i$ to $j$)

Weights (determining the strength and sign of the connection)

# Units




Slide credit: Russell and Norvig

# Units

Each unit $j$ first computes a weighted sum of its inputs:

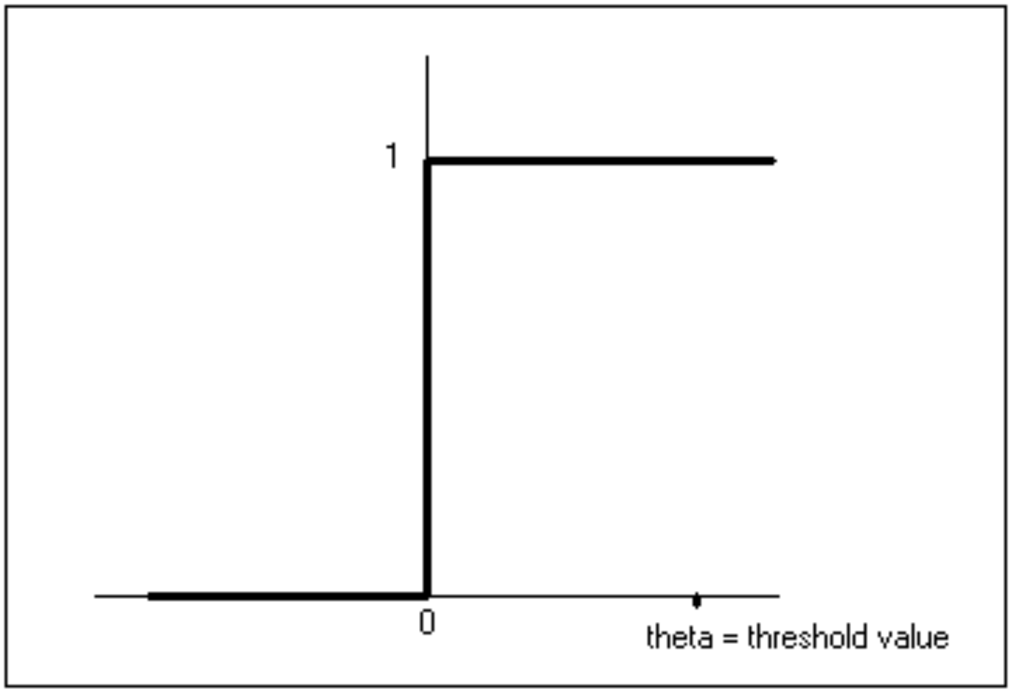
$$in_j = \sum_{i=0}^{n} w_{i,j} a_i$$

# Activation Functions

A unit then applies an **activation function** $g$ to the sum to derive the output:

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$

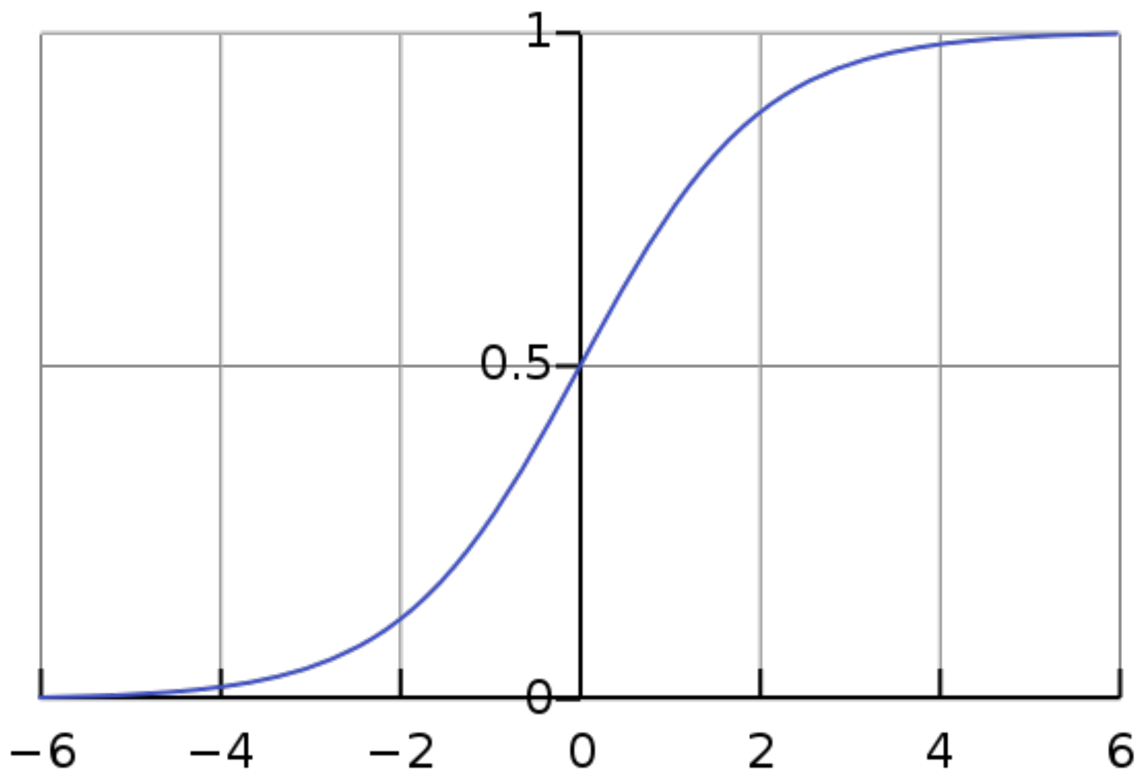
# Activation Function: Hard Threshold




**Unit: Perceptron**

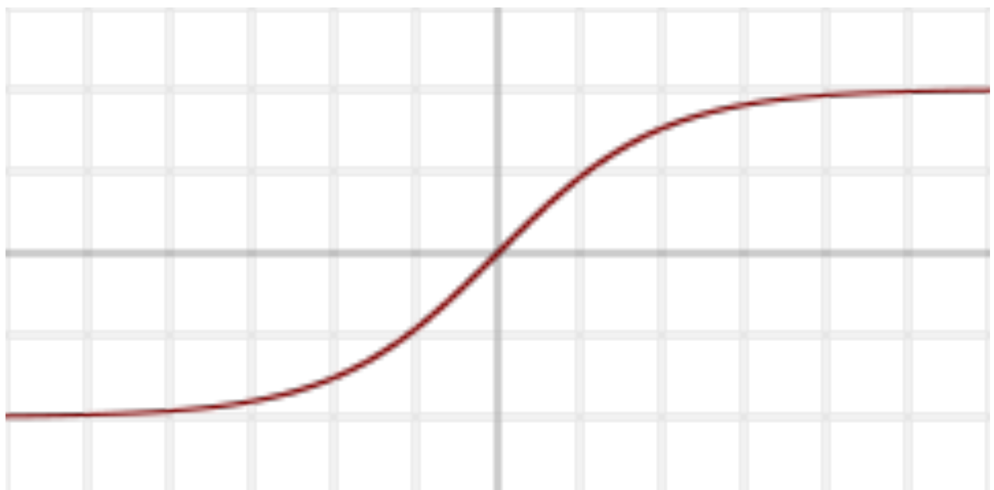
Image Credit: http://www.pirobot.org/blog/0005/

# Activation Function: Logistic Function




**Unit: Sigmoid Perceptron**

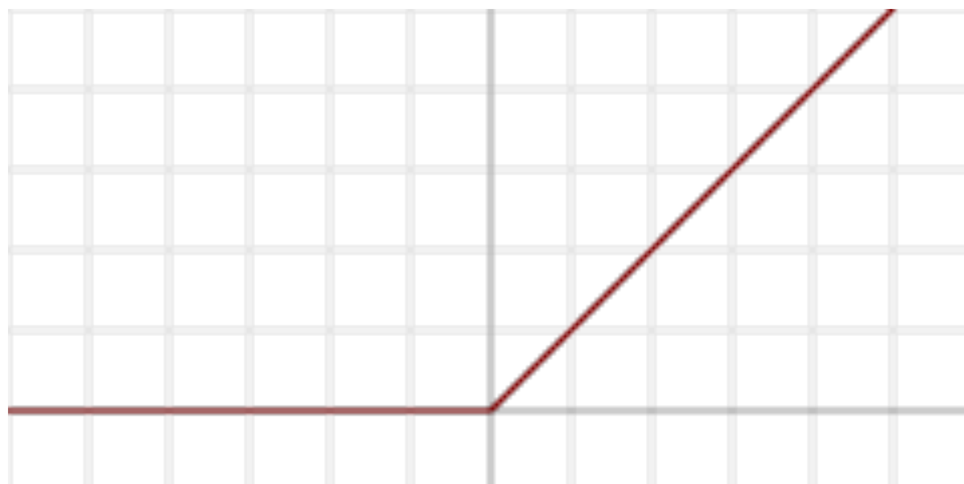
# Activation Function: tanh



A plot of the tanh activation function. 

**Popular in the early days of neural networks**

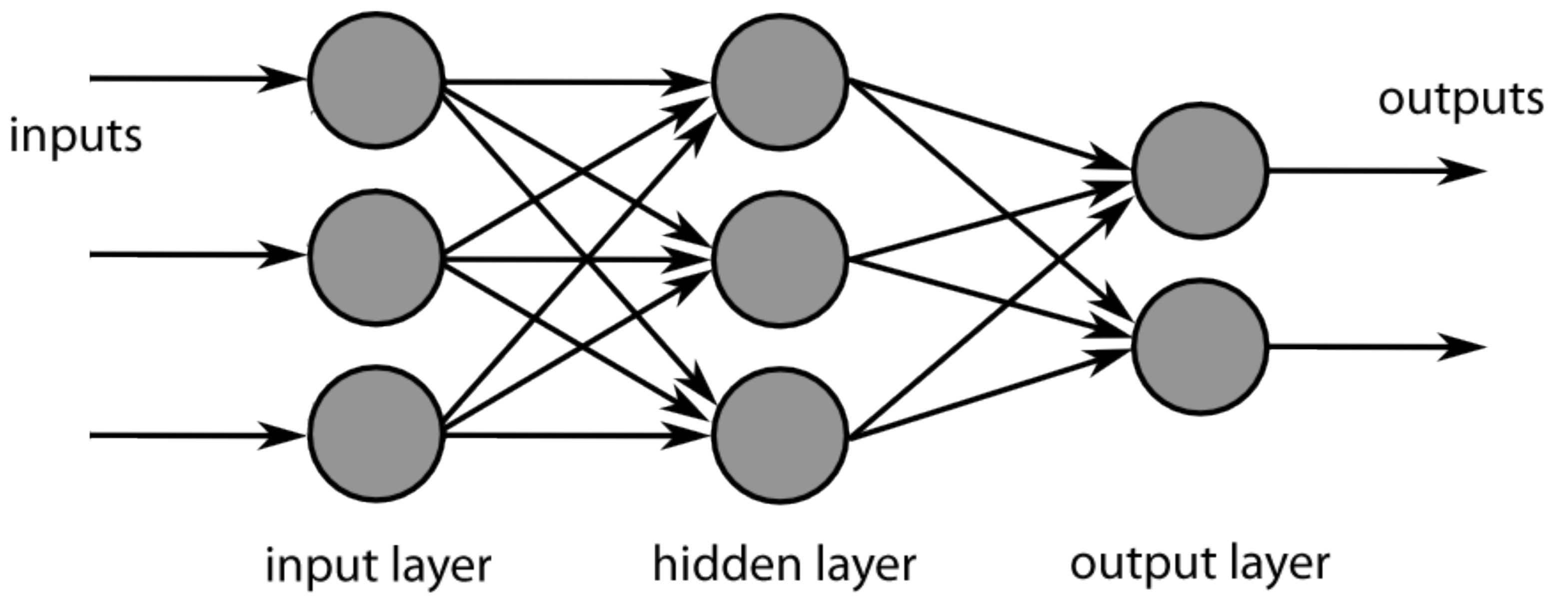
# Activation Function: Rectified Linear (ReLU)



A plot of the rectified linear activation function. 

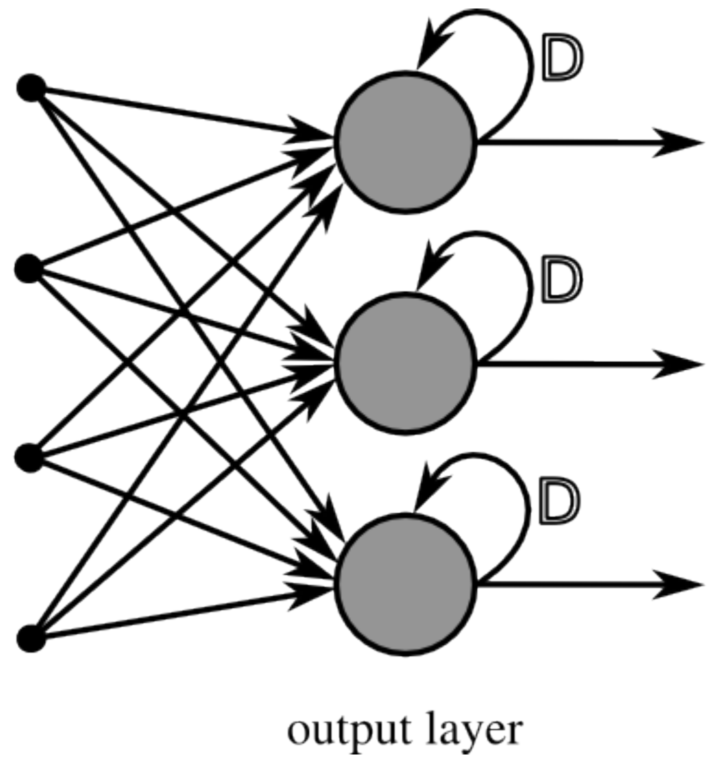
**This is the activation function you should use by default**

# Feed-Forward Networks



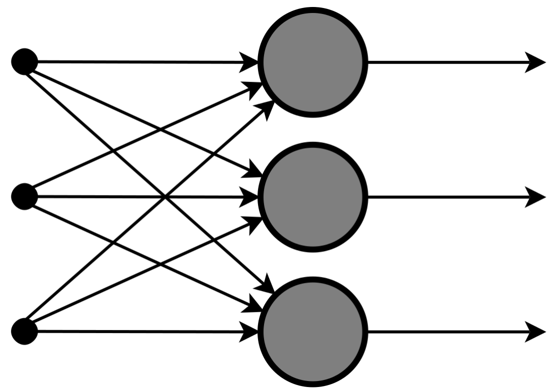

A Neural network with two layers 

# Recurrent Networks

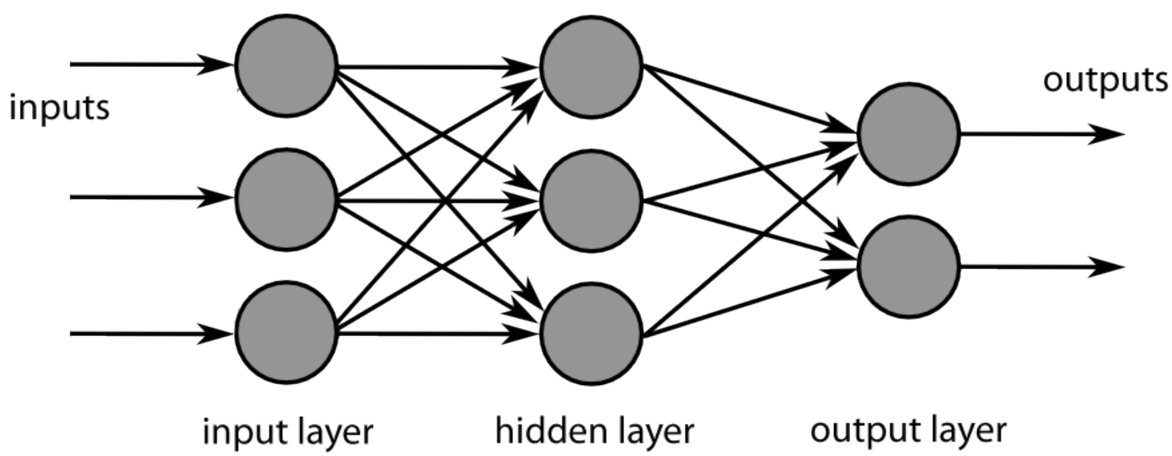


# Layers

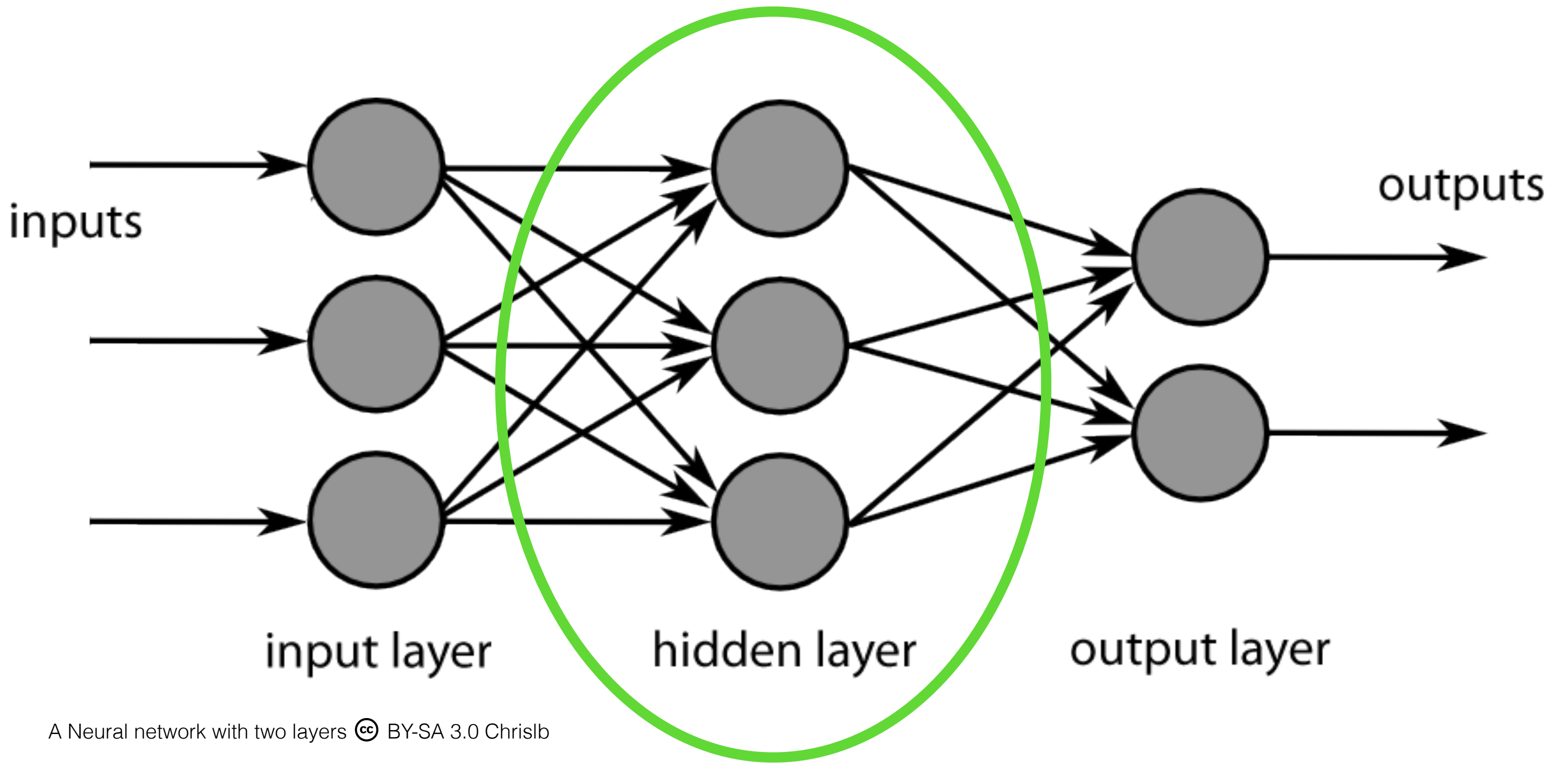
## One Layer



## Multiple Layers



# Hidden Units

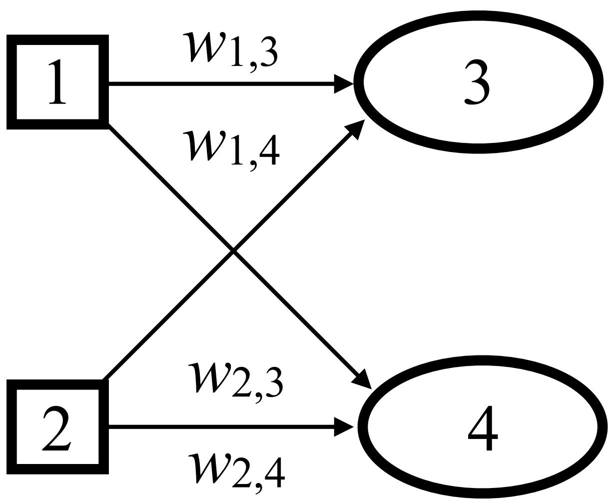





What role do these units play?

# Single-Layer Feed Forward Networks (Perceptrons)

Consider this network:

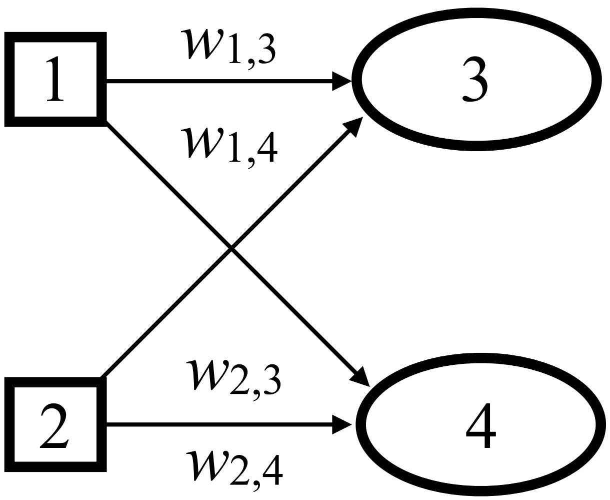



And this training data:

| $x_1$ | $x_2$ | $y_3$ (carry) | $y_4$ (sum) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Objective: Learn a two-bit adder function

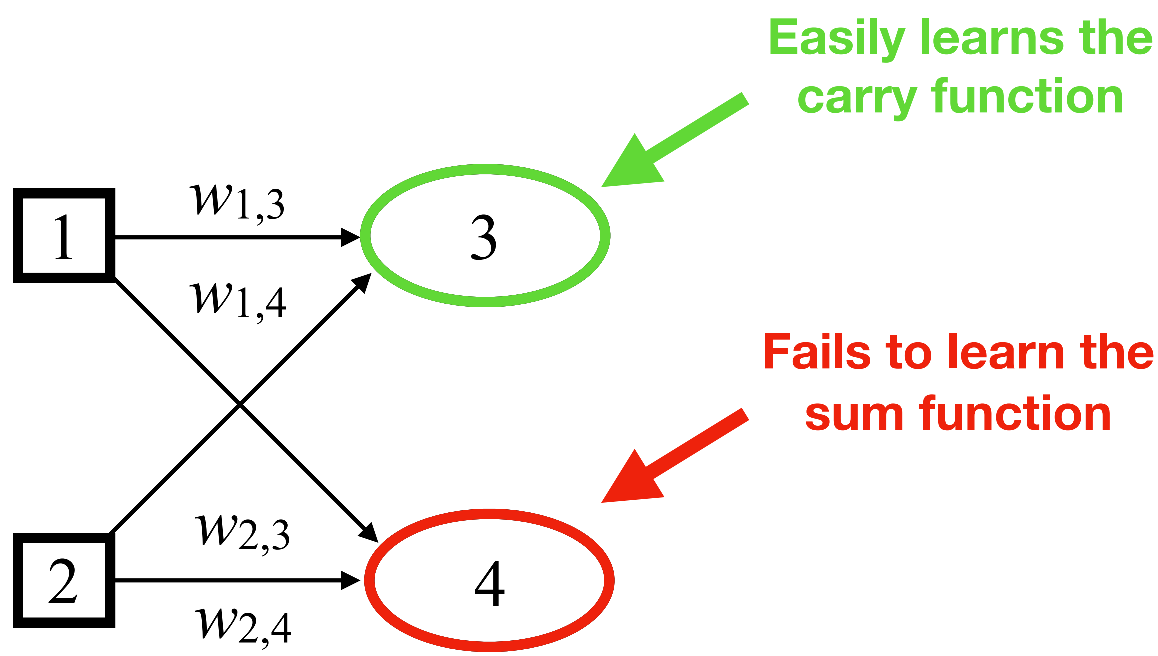
# Network structure




A perceptron network with $m$ outputs is really $m$ separate networks

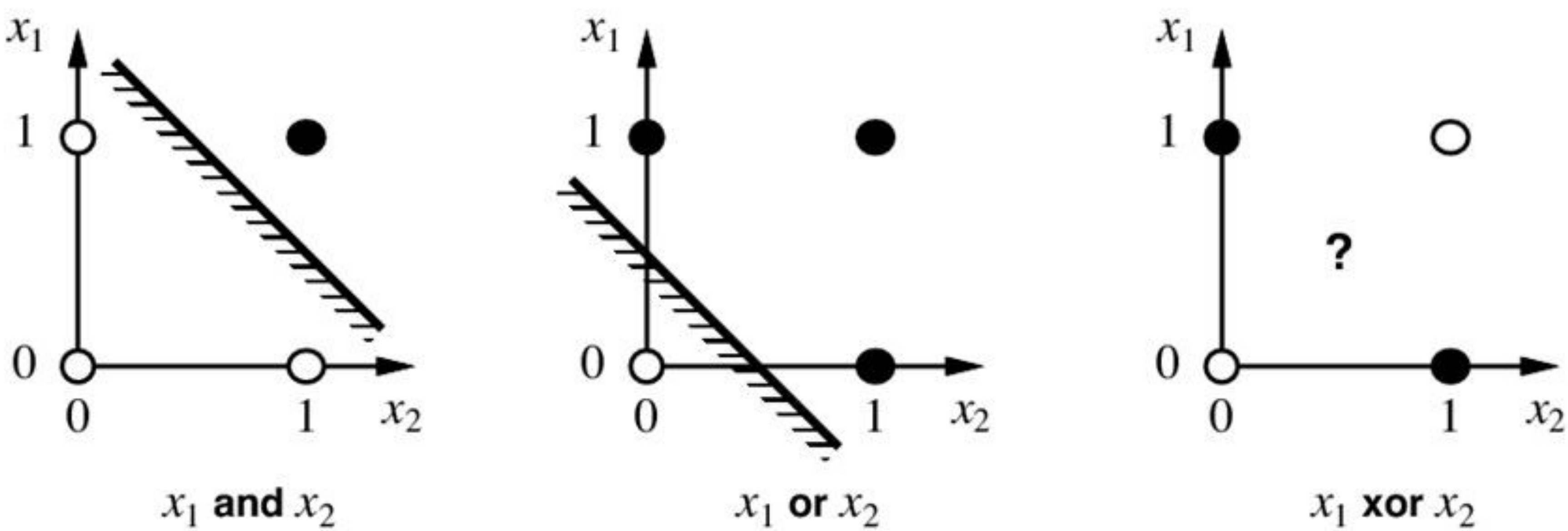
- Each weight affects only one of the outputs
- $m$ separate training processes

# Some trouble with the math…




Why does this happen?

# Linear Separability in Threshold Perceptrons

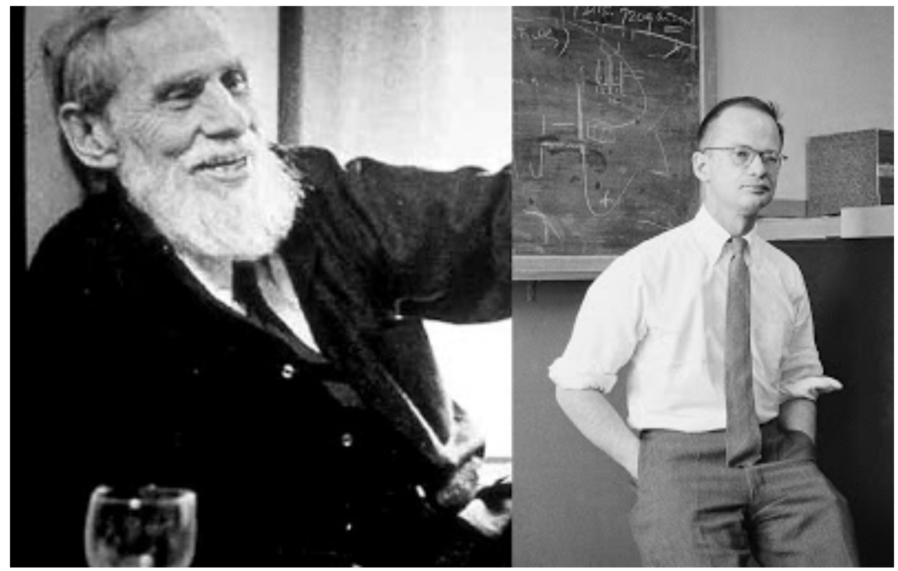



Slide credit: Russell and Norvig

# Multilayer Feed-Forward Neural Networks

**Problem:** The brain can solve XOR

McCulloch and Pitts argued that any desired functionality might be achieved by connecting large numbers of units

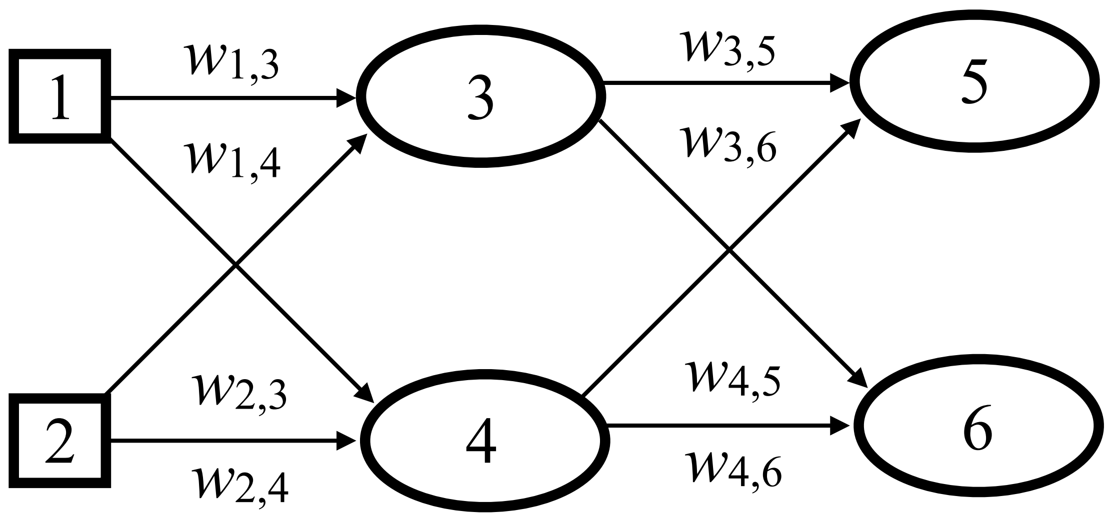


**How does the training work?**

Image credit: http://infonintelli.blogspot.com/2017/11/psychon-and-mcculloch-pitts-neurons.html

# Solution: add hidden units

As a function $h_{\mathbf{w}}(\mathbf{x})$ parameterized by weights $w$



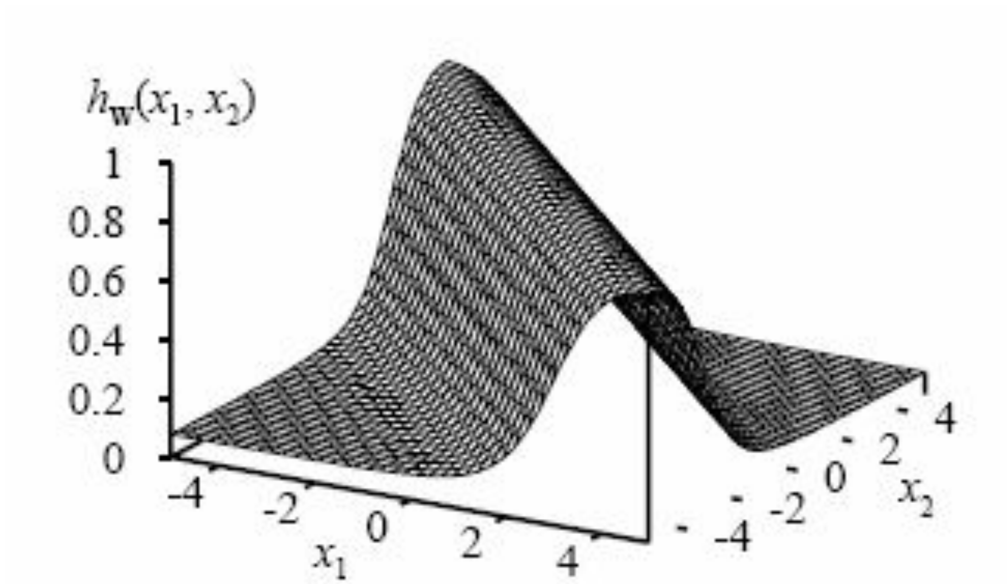
# Output expressed as a function of the inputs and the weights

Given an input vector $\mathbf{x} = (x_1, x_2)$, the activations of the inputs are set to $(a_1, a_2) = (x_1, x_2)$

The output at unit 5 is given by:

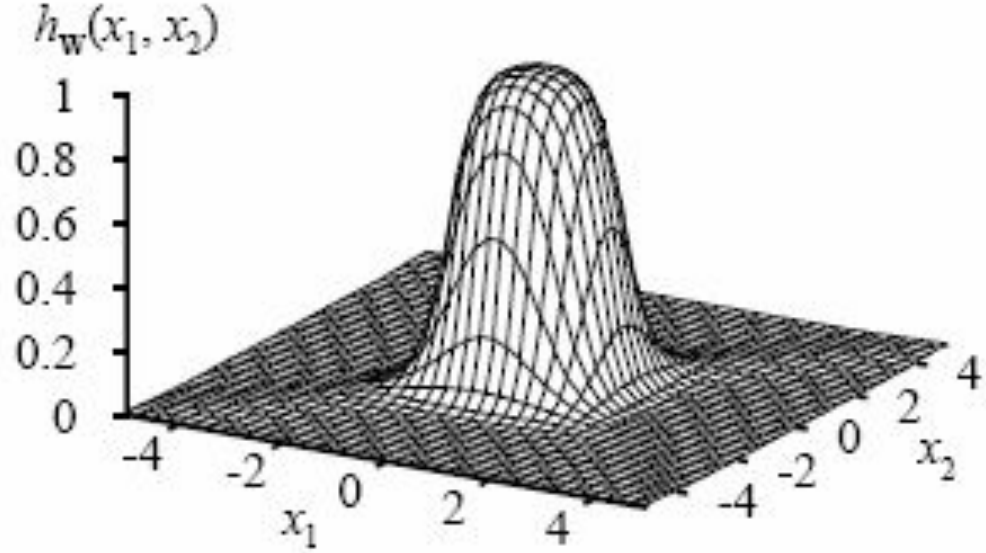
$$
\begin{aligned}
a_5 &= g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\
&= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}a_1 + w_{2,3}a_2) + w_{4,5}g(w_{0,4} + w_{1,4}a_1 + w_{2,4}a_2)) \\
&= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2))
\end{aligned}
$$

# Result of combining two opposite-facing soft threshold functions to produce a ridge




Slide credit: Russell and Norvig

# Result of combining two ridges to produce a bump




Slide credit: Russell and Norvig

# Utility of hidden layers

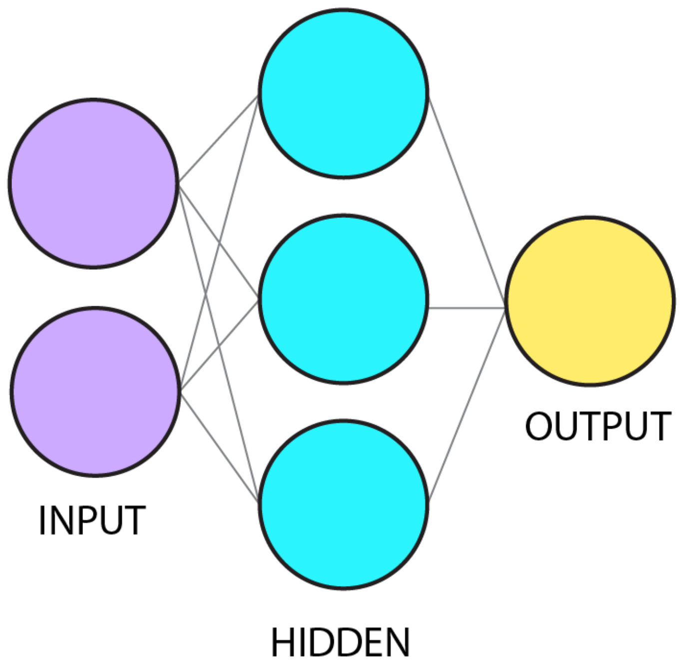
- With a single, sufficiently large hidden layer, it is possible to represent any continuous function of the inputs with arbitrary accuracy

- Two layers: even discontinuous functions can be represented

- Problem: for any *particular* network, it is harder to characterize which functions can be represented and which cannot

# PyTorch Example

Let's predict grades achieved on a test ($y$) based on hours spent studying and sleeping ($X$) using the following network:




Example adapted from: http://bit.ly/2Er1Cy0

Consider the brain: what is missing from this model?