Artificial Neural Networks

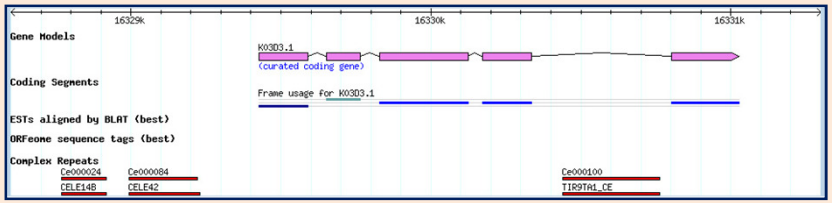
Genome 559: Introduction to Statistical and Computational Genomics

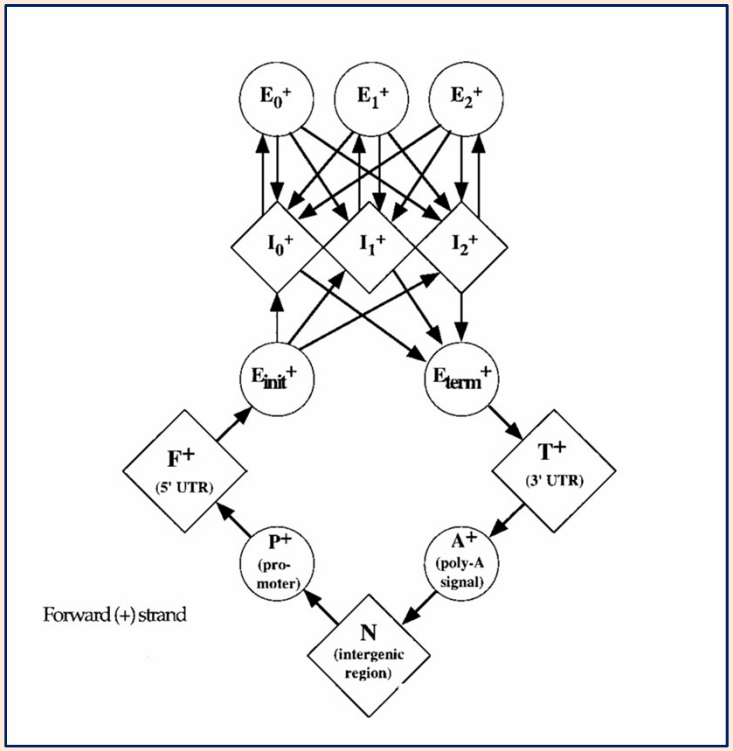
Elhanan Borenstein

Some slides adapted from Geoffrey Hinton and Igor Aizenberg

A quick review

- Ab initio gene prediction
- Parameters:
  - Splice donor sequence model
  - Splice acceptor sequence model
  - Intron and exon length distribution
  - Open reading frame
  - More …
- Markov chain
  - States
  - Transition probabilities
- Hidden Markov Model (HMM)





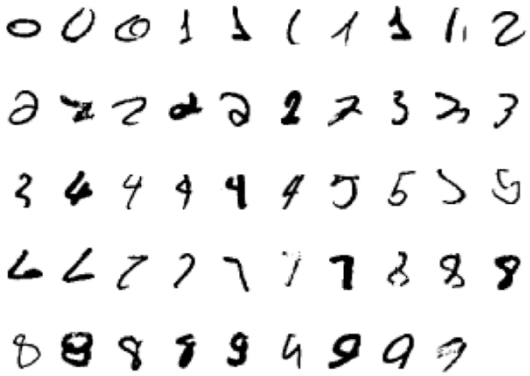
Machine learning

"A field of study that gives computers the ability to learn without being explicitly programmed."

Arthur Samuel (1959)

Tasks best solved by learning algorithms

- **Recognizing patterns**:
  - Facial identities or facial expressions
  - Handwritten or spoken words



- **Recognizing anomalies**:
  - Unusual sequences of credit card transactions
- **Prediction:**
  - Future stock prices
  - Predict phenotype based on markers
  - Genetic association, diagnosis, etc.

Why machine learning?

- It is very hard to write programs that solve problems like recognizing a face.
  - We don't know what program to write.
  - Even if we had a good idea of how to do it, the program might be horrendously complicated.
- Instead of writing a program by hand, we collect lots of examples for which we know the correct output
  - A machine learning algorithm then takes these examples, **trains**, and "produces a program" that does the job.
  - If we do it right, the program works for new cases as well as the ones we trained it on.

Why neural networks?

- One of those things you always hear about but never know exactly what they actually mean…
- A good example of a **machine learning** framework
- In and out of fashion …
- An important part of machine learning history
- A powerful framework

The goals of neural computation

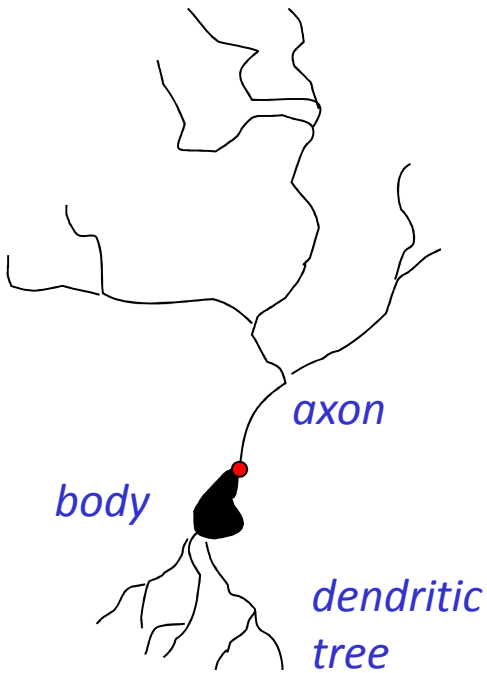
1. To understand how the brain actually works
   - Neuroscience is hard!
2. To develop a new style of computation
   - Inspired by neurons and their adaptive connections
   - **Very** different style from sequential computation
3. **To solve practical problems by developing novel learning algorithms**
   - Learning algorithms can be very useful even if they have nothing to do with how the brain works

How the brain works (sort of)

- Each neuron receives inputs from many other neurons
  - Cortical neurons use spikes to communicate
  - Neurons spike once they “aggregate enough stimuli” through input spikes
  - The effect of each input spike on the neuron is controlled by a synaptic weight. Weights can be positive or negative
  - Synaptic weights adapt so that the whole network learns to perform useful computations
- A huge number of weights can affect the computation in a very short time. Much better bandwidth than a computer.

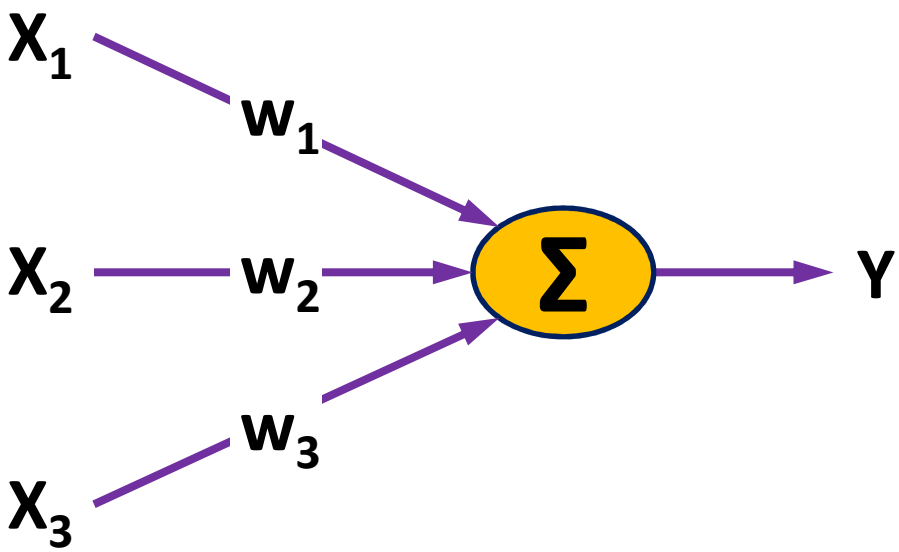
A typical cortical neuron

- Physical structure:
  - There is one axon that branches
  - There is a dendritic tree that collects input from other neurons
  - Axons typically contact dendritic trees at synapses
- A spike of activity in the axon causes a charge to be injected into the post-synaptic neuron



Idealized Neuron

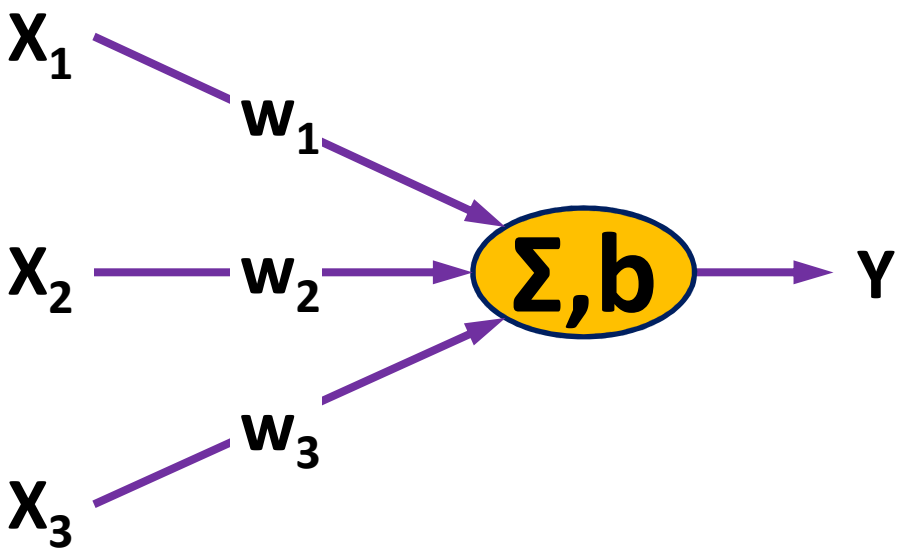
- Basically, a weighted sum!



$$y = \sum_i x_i w_i$$

Adding bias

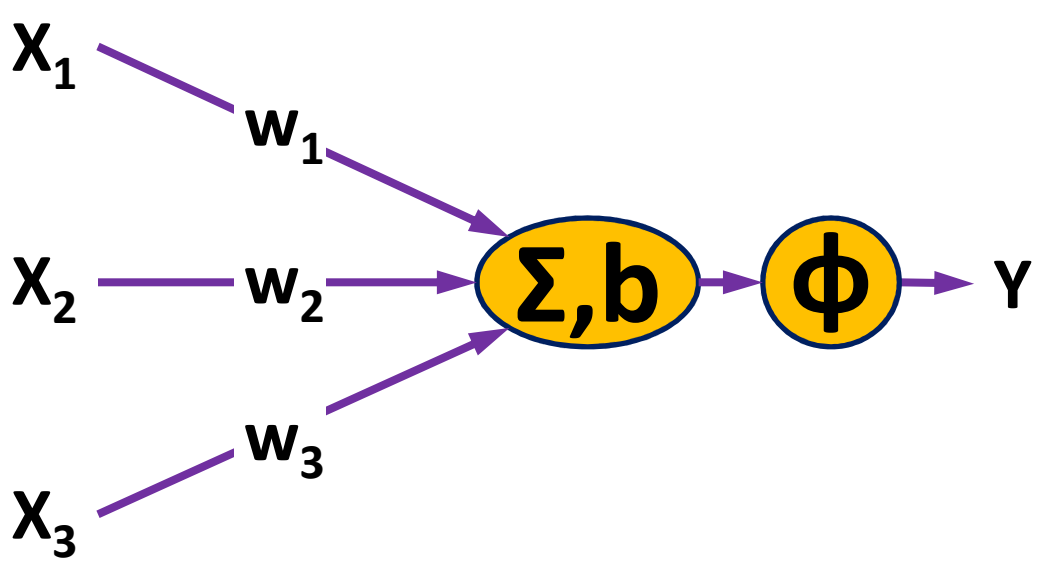
- Function does not have to pass through the origin



$$y = \sum_i x_i w_i - b$$

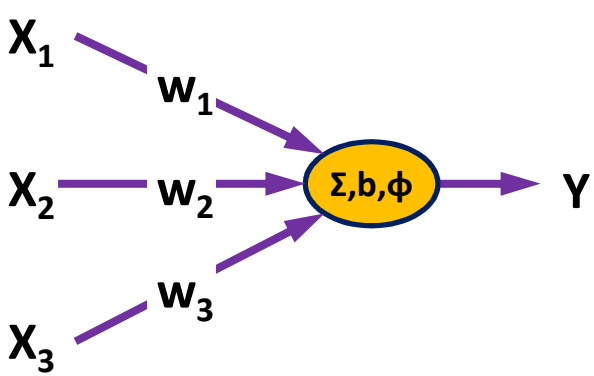
Adding an “activation” function

- The “field” of the neuron goes through an activation function



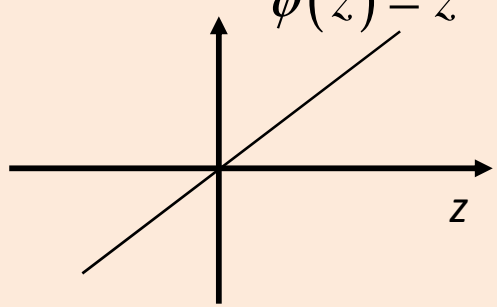
$$y = \Phi(\overbrace{\sum_i x_i w_i - b}^{Z,\text{ (the field of the neuron)}})$$

Common activation functions



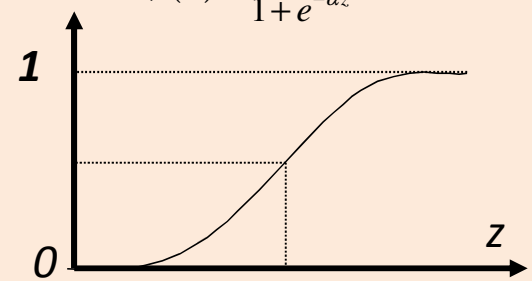
Linear activation

$$\phi(z) = z$$



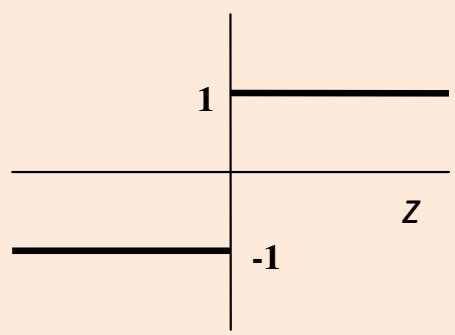
Logistic activation

$$\phi(z) = \frac{1}{1+e^{-\alpha z}}$$



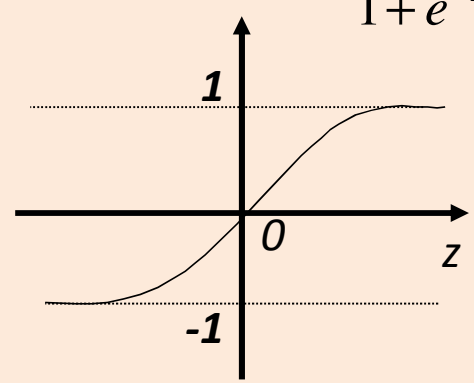
Threshold activation

$$\phi(z) = \text{sign}(z) = \begin{cases} 1, & if \quad z \geq 0, \\ -1, & if \quad z < 0. \end{cases}$$



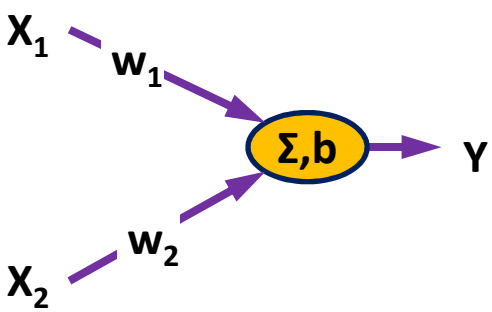
Hyperbolic tangent activation

$$\varphi(u) = tanh(\gamma u) = \frac{1-e^{-2\gamma u}}{1+e^{-2\gamma u}}$$



McCulloch-Pitts neurons

- Introduced in 1943 (and influenced Von Neumann!)
  - Threshold activation function
  - Restricted to binary inputs and outputs



$$z = \sum_i x_i w_i - b$$

$$y = \begin{cases} 1 \text{ if } z>0 \\ 0 \text{ otherwise} \end{cases}$$

w_1=1, w_2=1, b=1.5

X_1	X_2	y
0	0	0
0	1	0
1	0	0
1	1	1

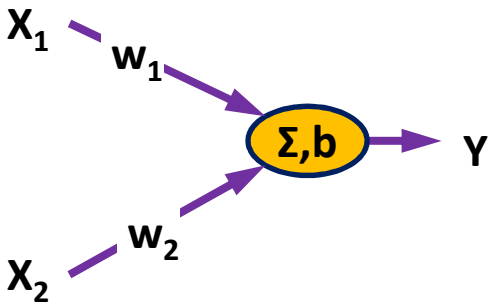
X_1 AND X_2

w_1=1, w_2=1, b=0.5

X_1	X_2	y
0	0	0
0	1	1
1	0	1
1	1	1

X_1 OR X_2

Beyond binary neurons



$$z = \sum_i x_i w_i - b$$

$$y = \begin{cases} 1 \text{ if } z>0 \\ 0 \text{ otherwise} \end{cases}$$

$w_1=1,\ w_2=1,\ b=1.5$

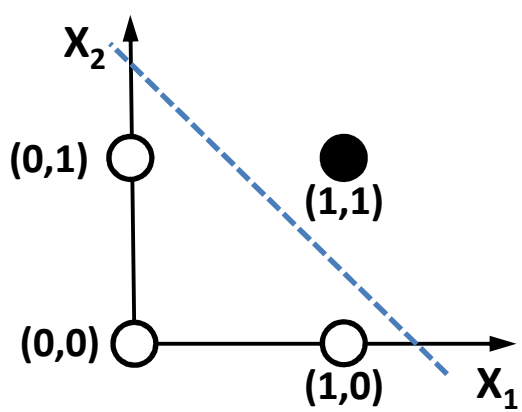
X_1	X_2	y
0	0	0
0	1	0
1	0	0
1	1	1

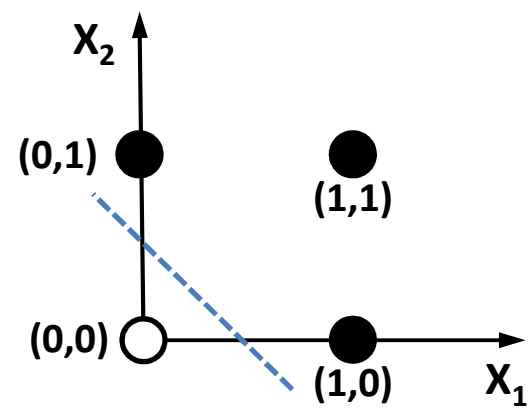
X_1 AND X_2

$w_1=1,\ w_2=1,\ b=0.5$

X_1	X_2	y
0	0	0
0	1	1
1	0	1
1	1	1

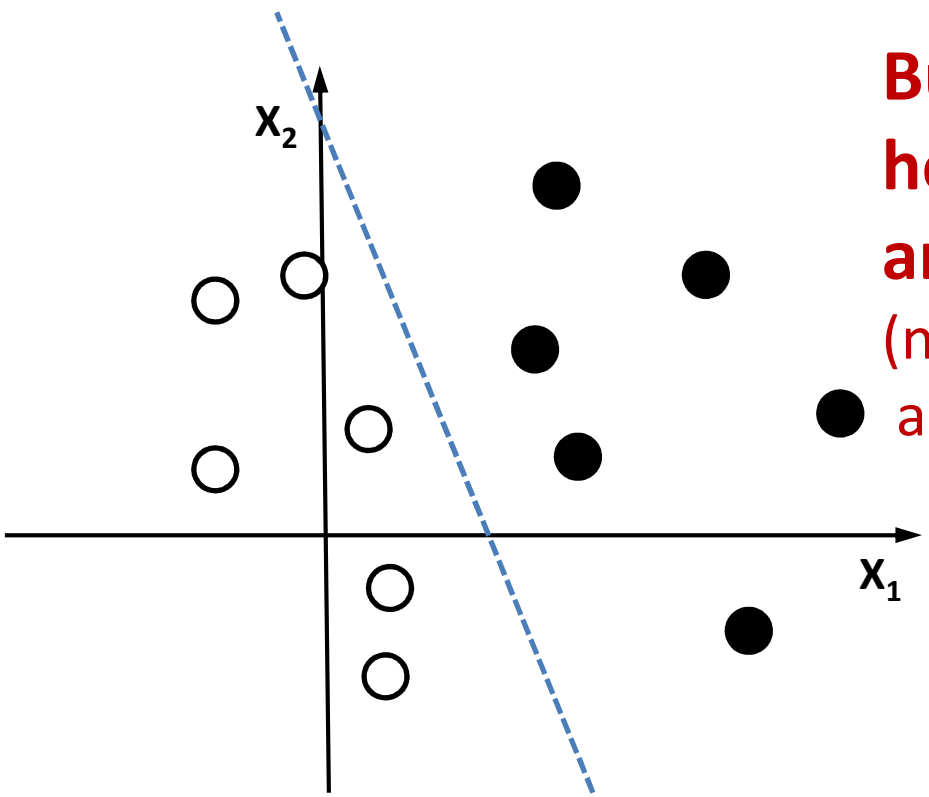
X_1 OR X_2





Beyond binary neurons

- A general classifier
  - The weights determine the slope
  - The bias determines the distance from the origin



But … how would we know how to set the weights and the bias?

(note: the bias can be represented as an additional input)

Perceptron learning

- Use a **"training set"** and let the perceptron **learn** from its mistakes
  - Training set: A set of input data for which we know the correct answer/classification!
  - Learning principle: Whenever the perceptron is wrong, make a small correction to the weights in the right direction.
- Note: **Supervised** learning
- Training set vs. testing set

Perceptron learning

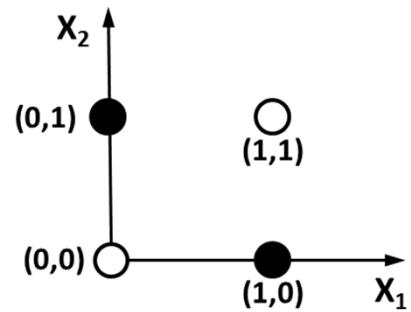
1. Initialize weights and threshold (e.g., use small random values).
2. Use input X and desired output *d* from training set
3. Calculate the actual output, *y*
4. Adapt weights: $w_i(t+1) = w_i(t) + \alpha(d - y)x_i$ for all weights. α is the **learning rate** (don't overshoot)

Repeat 3 and 4 until the $d-y$ is smaller than a user-specified error threshold, or a predetermined number of iterations have been completed.

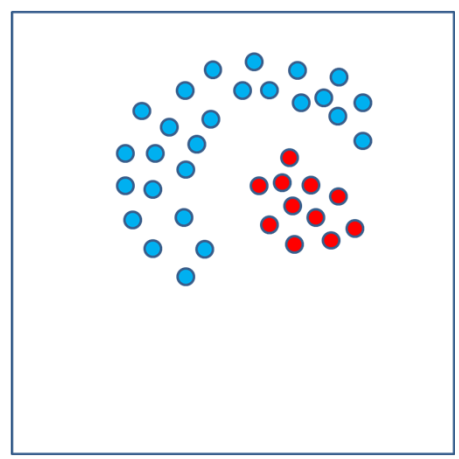
If solution exists – guaranteed to converge!

Linear separability

- What about the XOR function?

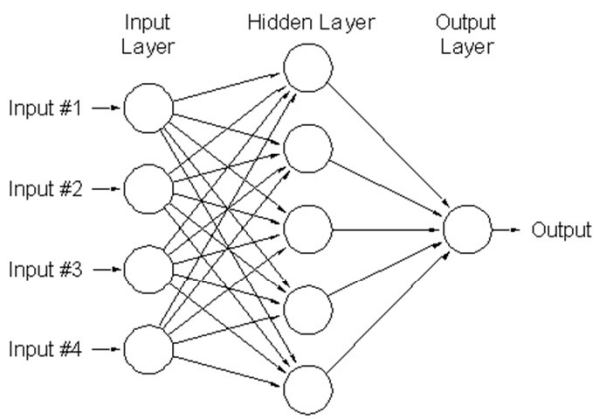


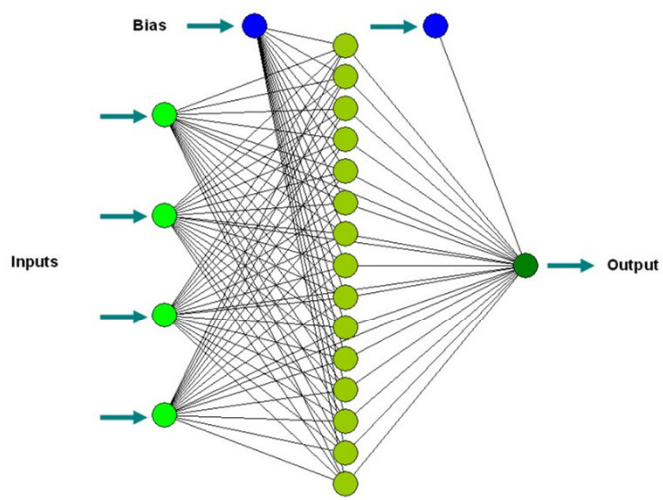
- Or other non linear separable classification problems such as:

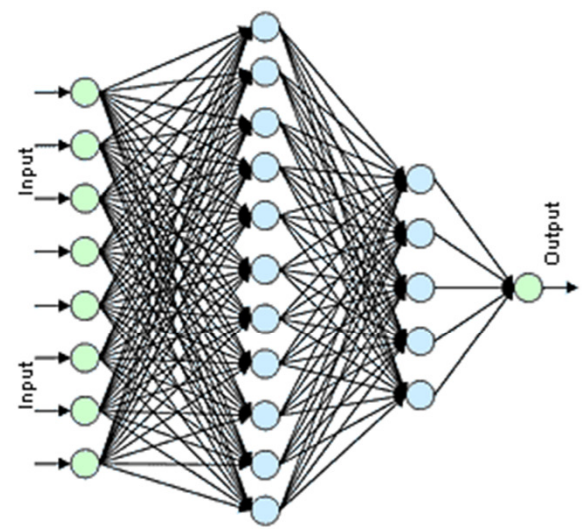


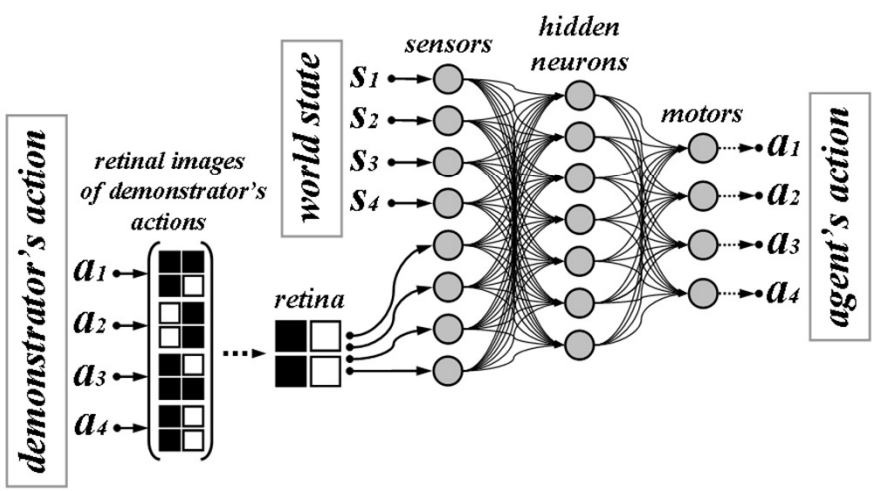
Multi-layer feed-forward networks

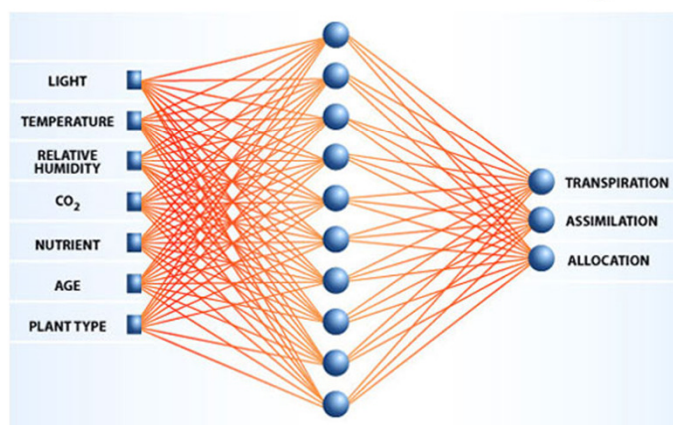
- We can connect several neurons, where the output of some is the input of others.





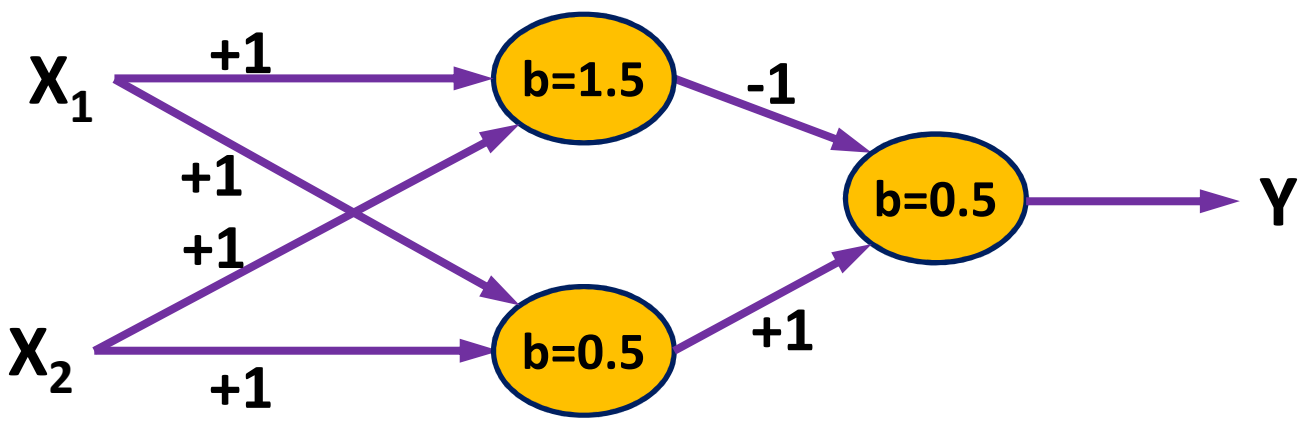






Solving the XOR problem

- **Only 3 neurons are required!!!**



In fact …

- **With one hidden layer you can solve ANY classification task!**

- **But …. How do you find the right set of weights?**

  (note: we only have an error delta for the output neuron)

- This problem caused this framework to fall out of favor … until …

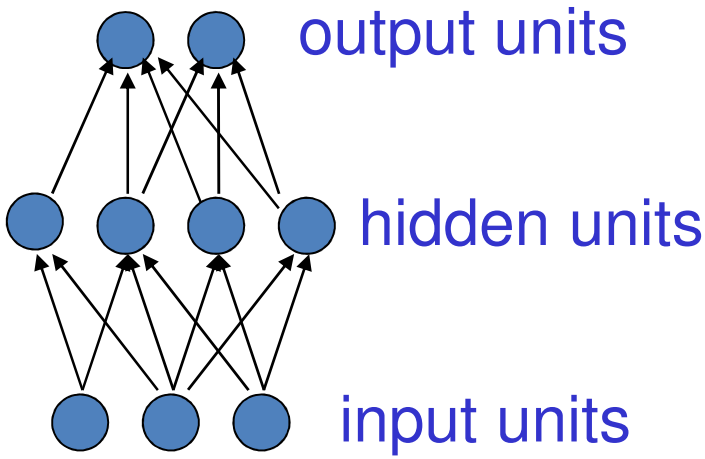
Back-propagation

<u>Main idea:</u>

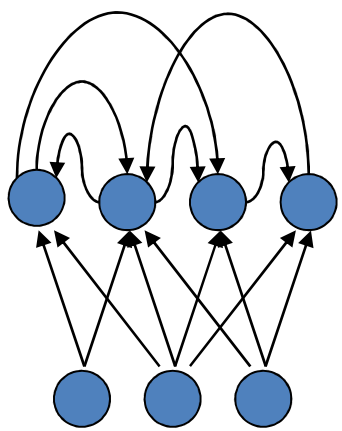
- First propagate a training input data point forward to get the calculated output
- Compare the calculated output with the desired output to get the error (delta)
- **Now, propagate the error <u>back</u> in the network to get an error estimate for each neuron**
- Update weights accordingly

Types of connectivity

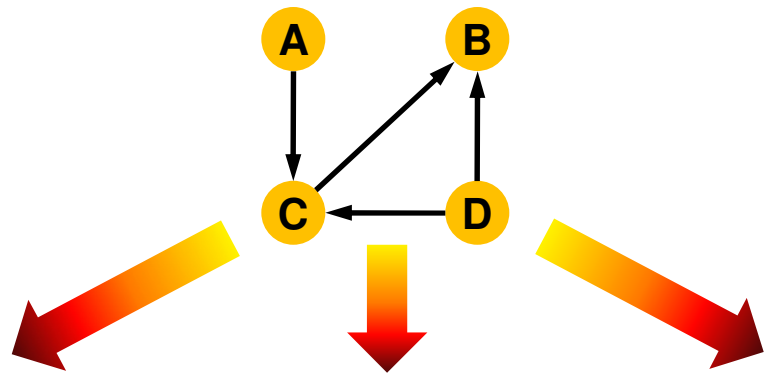
- **Feed-forward networks**
  - Compute a series of transformations
  - Typically, the first layer is the input and the last layer is the output.



- **Recurrent networks**
  - Include directed cycles in their connection graph.
  - Complicated dynamics.
  - **Memory.**
  - More biologically realistic?



Computational representation of networks



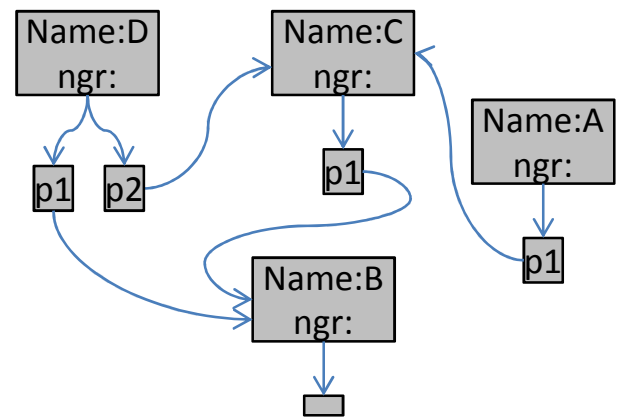
List of edges:
(ordered) pairs of nodes

[(A,C) , (C,B) , (D,B) , (D,C)]

Connectivity Matrix

	A	B	C	D
A	0	0	1	0
B	0	0	0	0
C	0	1	0	0
D	0	1	1	0

Object Oriented



- **Which is the most useful representation?**