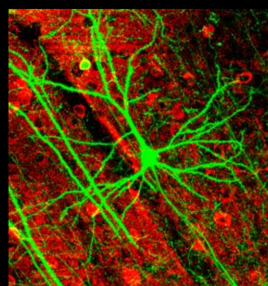
Neurons as Classifiers and Supervised Learning



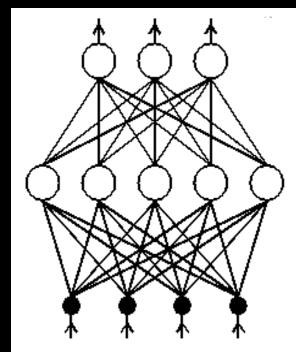


Image Source: Wikimedia Commons

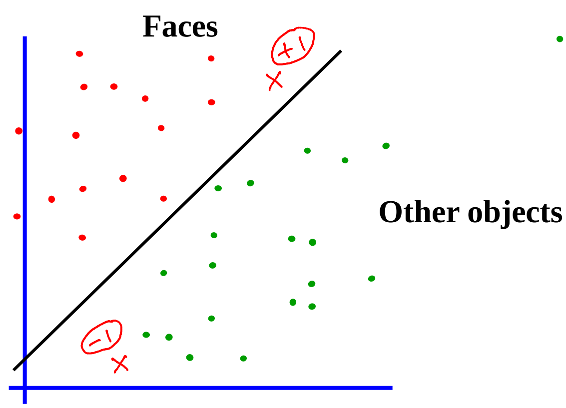
The Classification Problem

How do we build a classifier to distinguish between faces and other objects?

Image Source: Wikimedia Commons

The Classification Problem

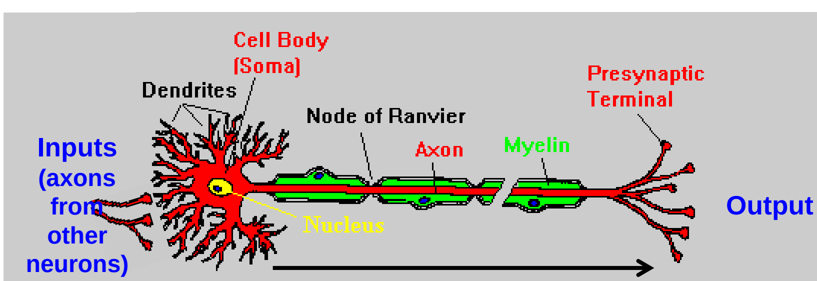
- denotes +1 (faces)
- denotes -1 (other)



Idea: Find a separating hyperplane (line in this case)

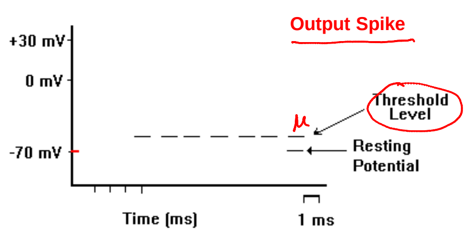
Can neurons do that?

The Idealized Neuron



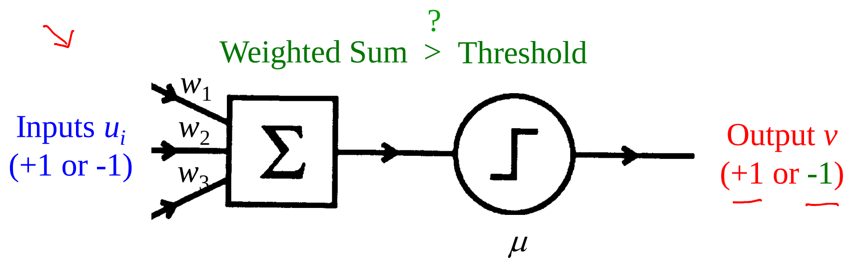
u_i w_i

$\sum_i w_i u_i > \mu$



Images by Eric Chudler, UW

The "Perceptron"



$$v = \Theta\left(\sum_i w_i u_i - \mu\right)$$

$\Theta(x)$ = +1 if $x > 0$ and -1 if $x \le 0$

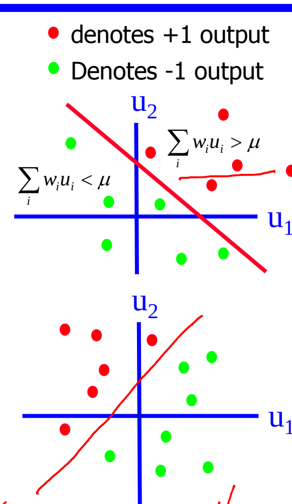
[Introduced by Rosenblatt (1958) building on McCulloch and Pitts (1943)]

What does a Perceptron do?

- Weighted sum defines a *hyperplane (line, plane, ...)*

  $$\sum_i w_i u_i - \mu = 0$$

- All inputs *on one side* of hyperplane have output = +1 ("class 1"); all inputs *on other side* have output = -1 ("class 2")
- Perceptrons can classify!
  - Can perform linear classification



How do we learn the weights and threshold?

Perceptron Learning Rule

Given input **u**, output $v = \Theta(\sum_i w_i u_i - \mu)$, and desired output v^d:

Adjust w_i and μ according to output error $(v^d - v)$:

$$\Delta w_i = \varepsilon(v^d - v)u_i$$

1 -1

-1 1

For positive input (u_i = +1):
Increases weight if error is positive
Decreases weight if error is negative
(opposite for u_i = -1)

$$\Delta \mu = -\varepsilon(v^d - v)$$

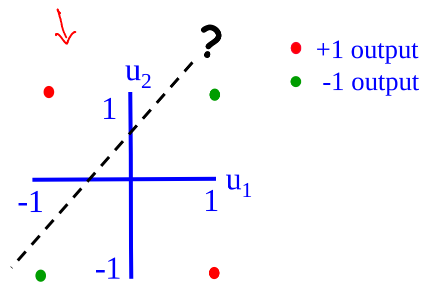
1 -1

-1 1

Decreases threshold if error is positive
Increases threshold if error is negative

Can Perceptrons learn any function?

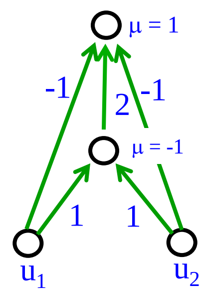
u_1	u_2	XOR
-1	-1	-1
1	-1	+1
-1	1	+1
1	1	-1



Perceptrons can only classify linearly separable data
How do we handle linear inseparability?

Multilayer Perceptrons

- Can classify linearly inseparable data
  - Can solve XOR
- An example of a two-layer perceptron that computes XOR



(Inputs and outputs are +1 or -1)

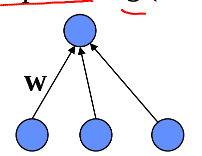
What if you want *continuous* outputs rather than +1/-1 outputs (i.e., regression)?

E.g., Teaching a network to drive a truck

Image Source: Wikimedia Commons

Continuous Outputs with Sigmoid Networks

Output $v = g(\mathbf{w}^T\mathbf{u}) = g(\sum_i w_i u_i)$

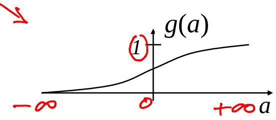


$\mathbf{u} = (u_1 \quad u_2 \quad u_3)^T$

Input nodes

Sigmoid output function:

$$g(a) = \frac{1}{1+e^{-\beta a}}$$

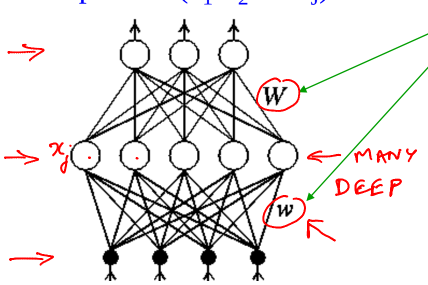


Parameter β controls the slope

Learning Multilayer Sigmoid Networks

$v_i = g(\sum_j W_{ij} g(\sum_k w_{jk} u_k))$

Output $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_J)^T$



Input $\mathbf{u} = (u_1 \ u_2 \ \dots \ u_K)^T$

$$\frac{dE}{dw_{jk}} = \frac{dE}{dx_j} \cdot \frac{dx_j}{dw_{jk}}$$

Desired output **d** also given

Learn weights that minimize output error:

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

Use gradient descent!

$$\Delta W_{ij} = -\varepsilon \frac{dE}{dW_{ij}} = \varepsilon \cdot \underbrace{(d_i - v_i) g'(\sum_j W_{ij} x_j)}_{\delta} x_j$$

Delta rule

$$\boxed{\Delta w_{jk} = -\varepsilon \frac{dE}{dw_{jk}}}$$

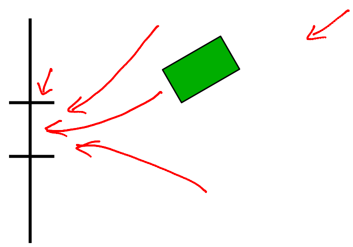
Backpropagation learning rule

(see Supplementary Materials for details)

Example: Backing up a Truck (courtesy of Keith Grochow)

Teaching a Network to back a truck into a loading dock

- Input: x, y, θ of truck
- Output: Steering angle



Next: Predicting Rewards and Reinforcement Learning

Image Source: Wikimedia Commons