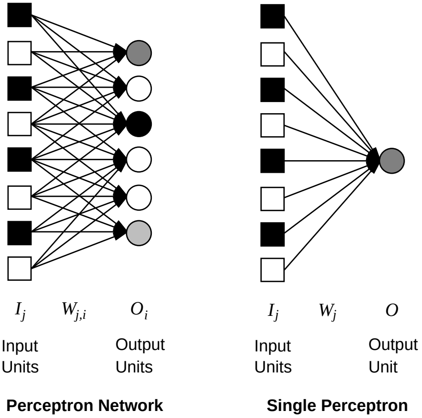
Perceptron and Multilayer Perceptron

Perceptron

a feed-forward network with only one layer of adjustable (learnable) weights connected to one or more threshold units (as output units)



Model

input: $I_1, I_2, \ldots, I_n$

- signals from the other neurons

weights: $w_1, w_2, \ldots, w_n$

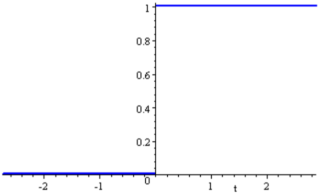
- can be negative

activation function:

- relating the input and output

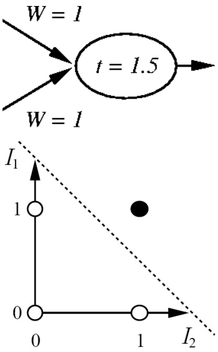
$$O = \text{step}(\textstyle\sum_{j=1}^{n} w_j I_j - \theta)$$

$$\text{step}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \qquad \text{(step function)}$$



What Boolean Functions can Perceptrons Represent?

AND?

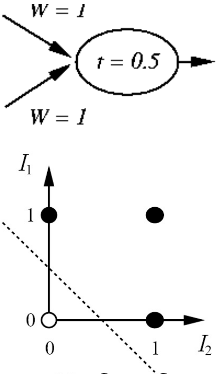


(a) I_1 and I_2

- Perceptron output: $O = \text{step}(\sum_{j=0}^{n} w_j I_j)$
  - decision boundary: $\sum_{j=0}^{n} w_j I_j = 0$

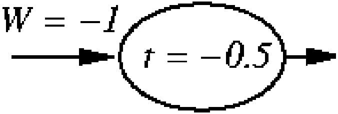
What Boolean Functions can Perceptrons Represent?...

OR?



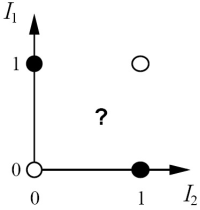
(b) I_1 **or** I_2

NOT?



What Boolean Functions can Perceptrons Represent?...

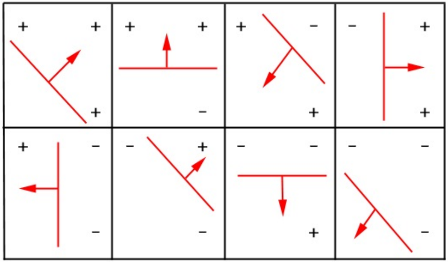
XOR?



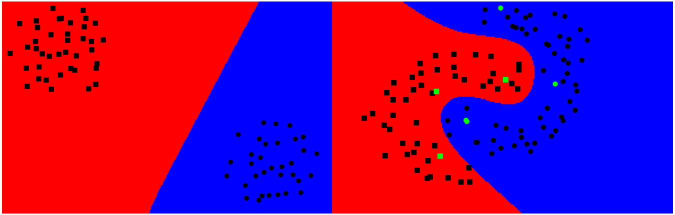
(c) I_1 **xor** I_2

Linearly Separable Functions

- a function can be represented by a single perceptron if and only if the function is linearly separable

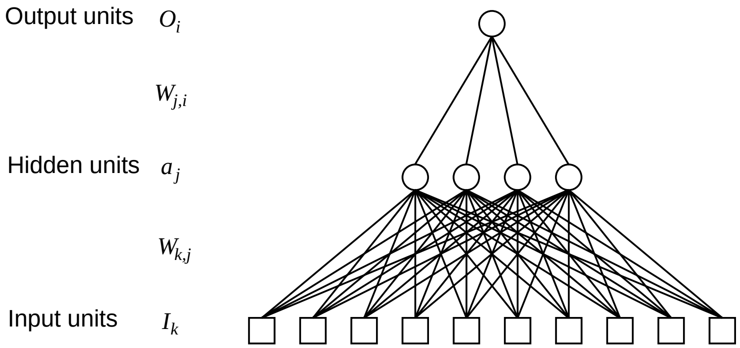


Three points in a plane shattered by a half-space.



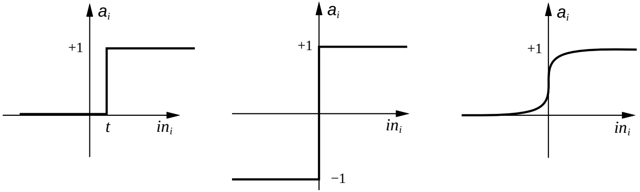
Multi-layer Feedforward Networks

- Generalization of simple perceptrons
- Multi-layer perceptrons (MLP)



Sigmoid unit

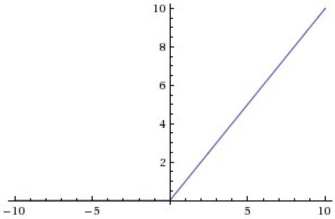
- a unit very much like a perceptron, but based on a smoothed, differentiable threshold function: $\sigma(x) = \frac{1}{1+e^{-x}}$



(a) Step function **(b) Sign function** **(c) Sigmoid function**

Rectified Linear Unit (ReLU)

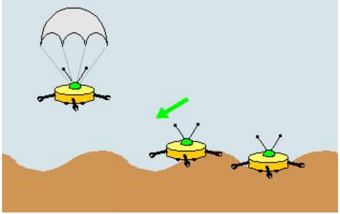
- $f(x) = \max(0, x)$



- the most popular activation function for deep networks
- more efficient computation
- simple gradient
    - if > 0, gradient $= 1$
    - if ≤ 0, gradient $= 0$

Training: Finding the Weight

- use gradient descent to search the space of possible weight vectors to find the weights that minimizes the error
- start at any point and keep going downhill



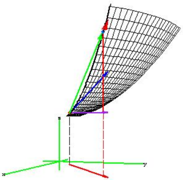
Idea: Gradient descent

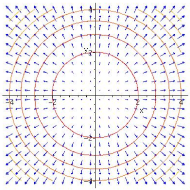
- start with initial value for $\mathbf{w}$
- repeat until convergence
  - compute the gradient vector of the error function for current $\mathbf{w}$
  - move in the opposite direction

Gradient Descent

gradient $\nabla E[\vec{w}]$ at $\vec{w}$:

$$\left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$





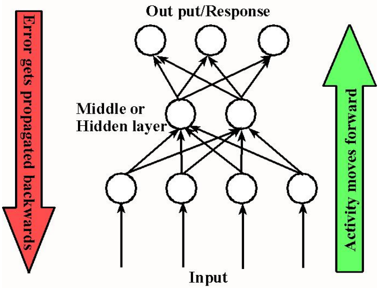
move $\vec{w}$:

- direction: opposite to $\nabla E[\vec{w}]$
- magnitude: a small fraction of $\nabla E[\vec{w}]$

$$\begin{aligned} \vec{w} &\leftarrow \vec{w} + \Delta\vec{w} \\ \Delta\vec{w} &= -\eta\nabla E[\vec{w}] \\ \Delta w_i &= -\eta\frac{\partial E}{\partial w_i} \end{aligned}$$

in general, the error surface can be very complicated

Back-Propagation



- we need to "propagate error back" when computing the gradient ector