Artificial Neural Networks

Adaline & Perceptron

Slides modified from Neural Network Design
by Hagan, Demuth and Beale

Berrin Yanikoglu

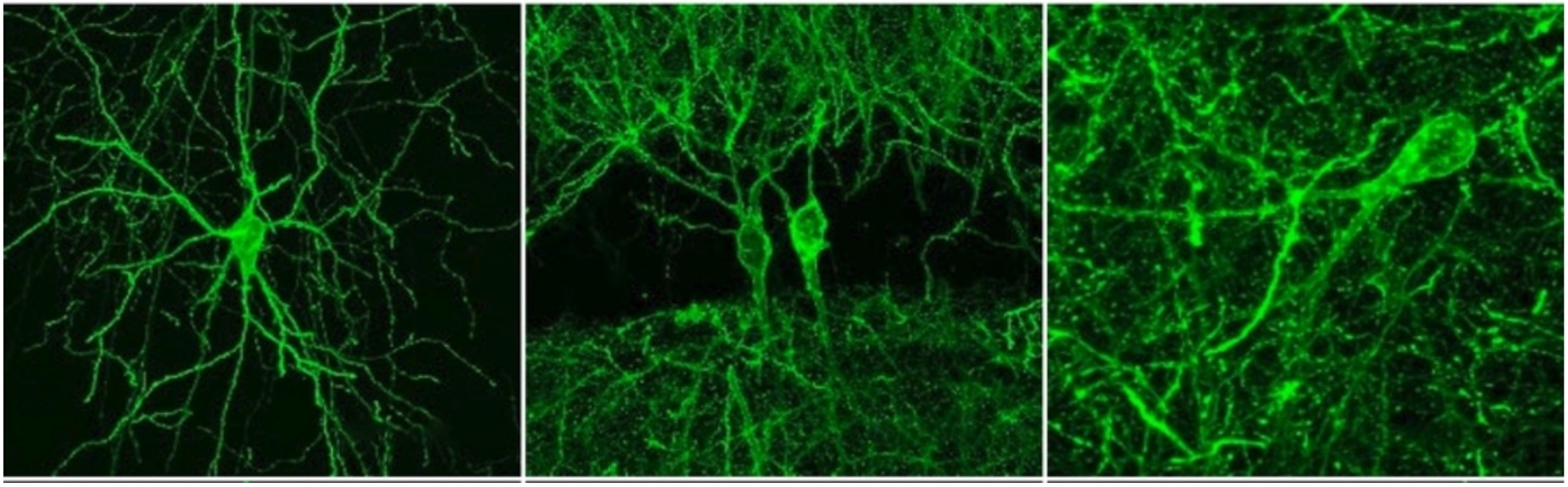
Biological Inspirations

Humans perform complex tasks like vision, motor control, or language understanding very well.

One way to build intelligent machines is to try to imitate the (organizational principles of) human brain.

Human Brain

- The brain is a highly complex, non-linear, and parallel computer, composed of some 10^{11} neurons that are densely connected with around 10^{4} connection per neuron.



Human Brain

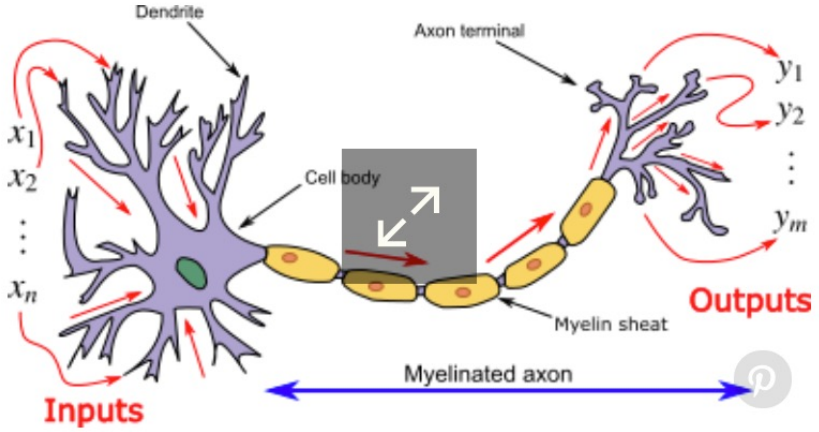
- A neuron is much slower (10^{-3}sec) compared to a silicon logic gate (10^{-9}sec) – **million times slower** - however the massive interconnection between neurons make up for the comparably slow rate.

- 100-Steps rule: Since individual neurons operate in a few milliseconds, calculations do not involve more than about 100 serial steps and the information sent from one neuron to another is very small (a few bits)
  - Complex perceptual decisions are arrived at quickly (within a few hundred milliseconds)

- Plasticity: Some of the neural structure of the brain is present at birth, while other parts are developed through learning, especially in early stages of life, to adapt to the environment (new inputs).

Biological Neuron

- dendrites: nerve fibres carrying electrical signals to the cell
- cell body: computes a non-linear function of its inputs
- synapse: the point of contact between the axon of one cell and the dendrite of another
- axon: single long fiber that carries the electrical signal from the cell body to other neurons

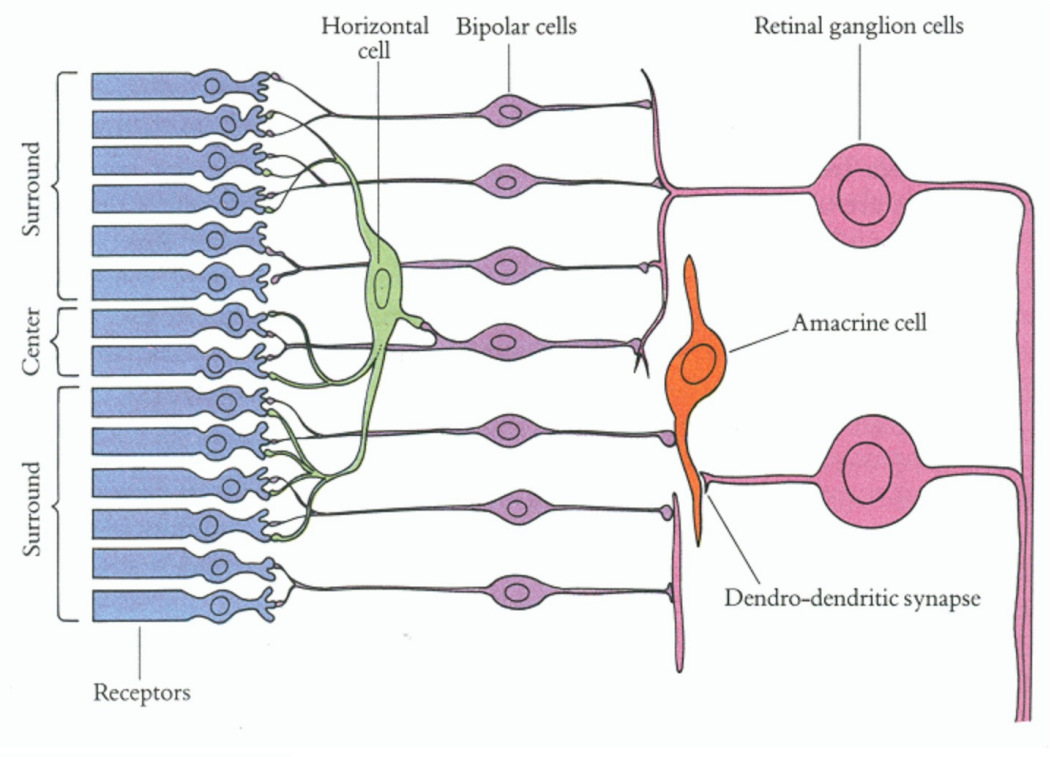


Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals

Biological Neuron

A variety of different neurons exist (motor neuron, on-center off-surround visual cells…), with different connectivity.

The connections of the network and the strengths of the individual synapses establish the function of the network.



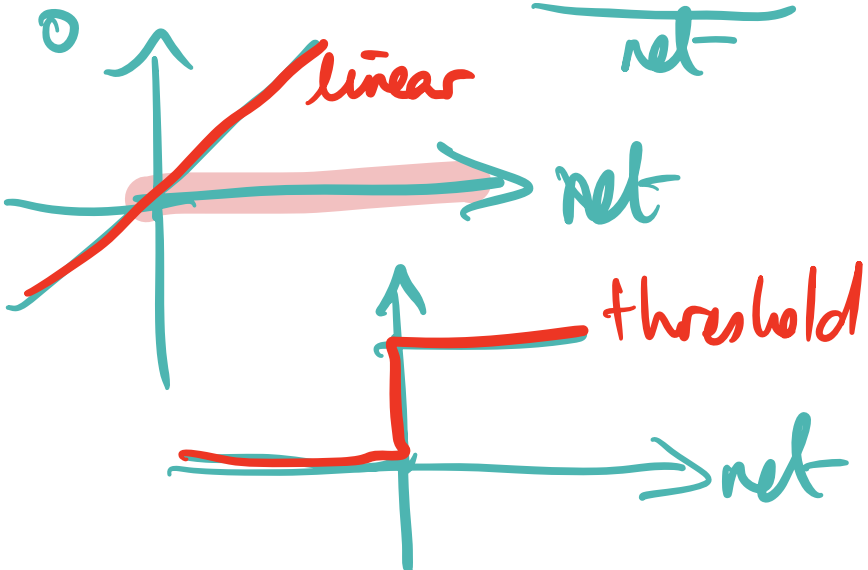
http://fourier.eng.hmc.edu/e180/lectures/retina/node6.html

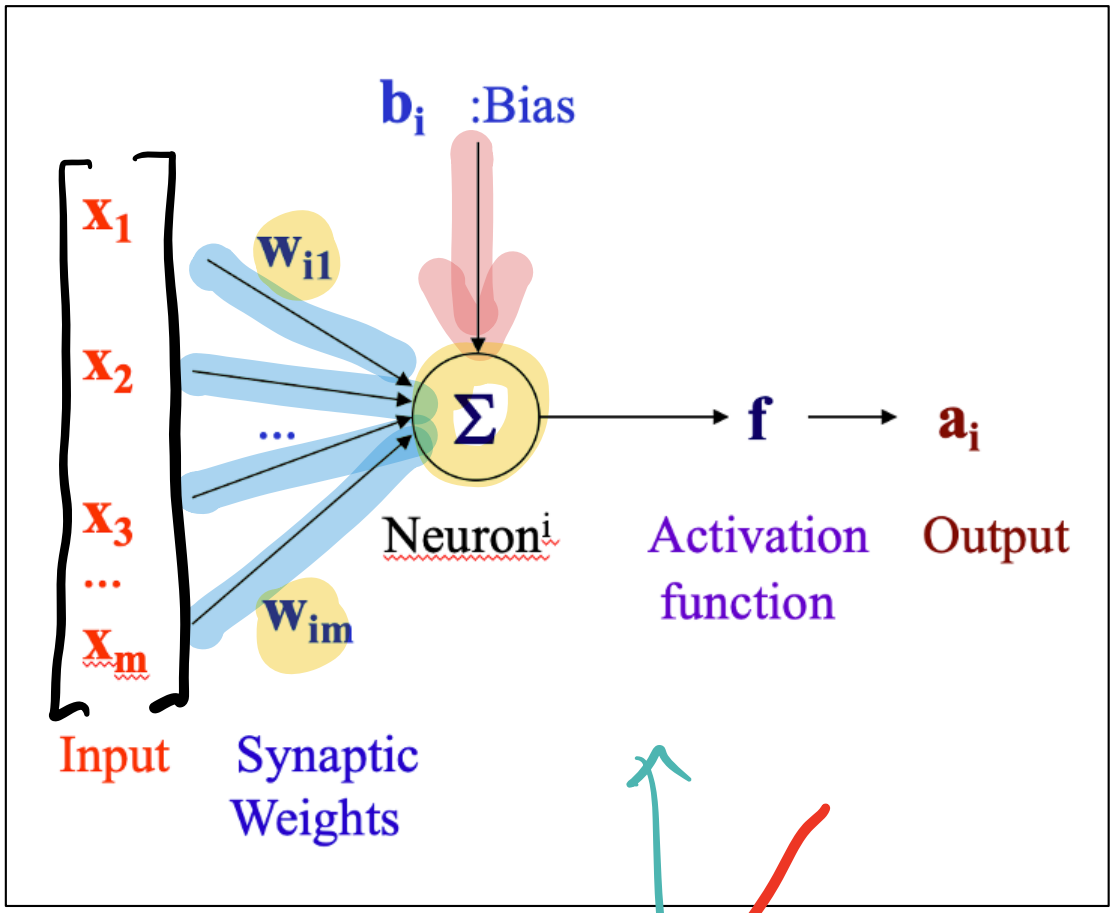
Artificial Neuron: Perceptron and Adaline

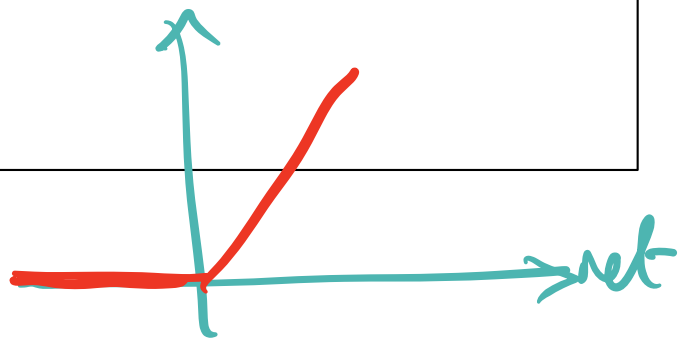
- A single artificial neuron computes its output, based on passing its net input through an activation function.

$$net = \mathbf{w}^T\mathbf{x} + b$$

$$output = f(\mathbf{w}^T\mathbf{x} + b)$$





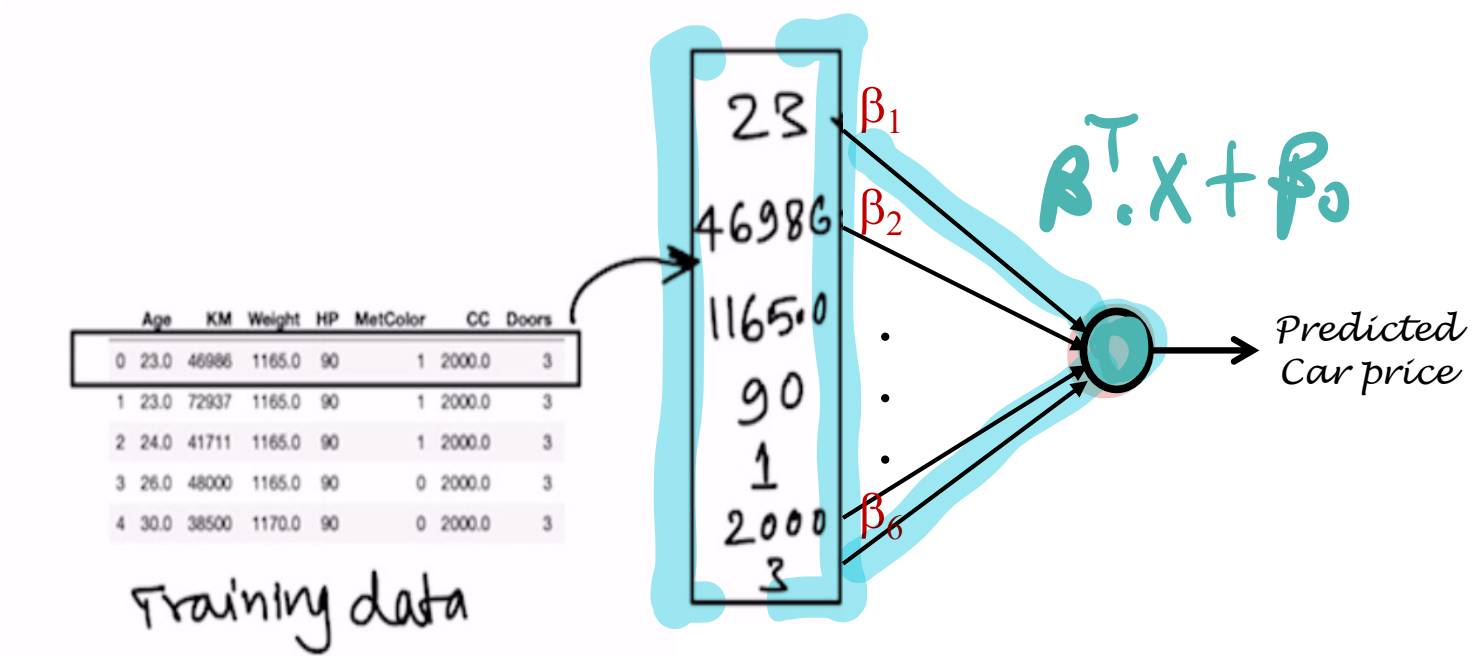


ADALINE

Regression problem: Predict the price of a car from given input parameters (model, k, age, color,…) and model weights, to match the prices of previous sales data (used as training data).

Loss: Mean squared error between known and predicted prices.

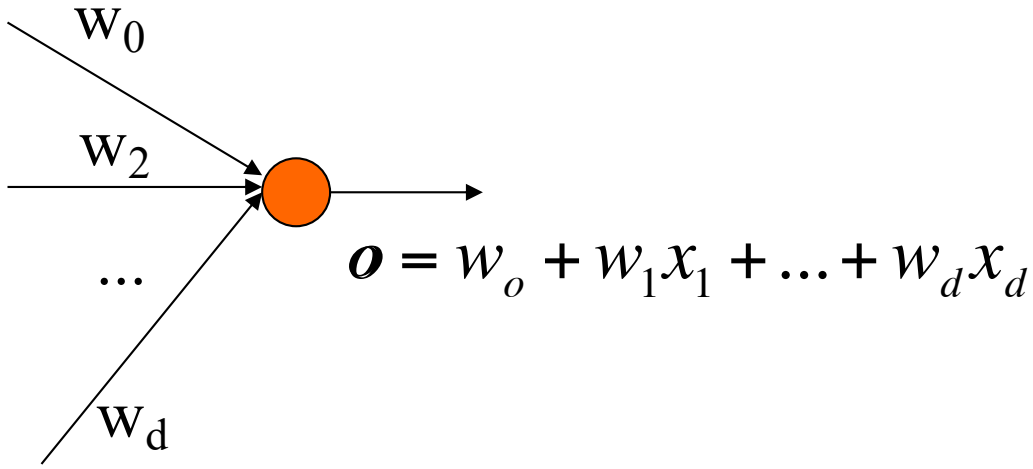
Optimization: Want to change the weights (β_i) to minimize the error over the training set.



ADALINE

t_i: target

o_i: actual neuron output



Let's find/learn the weights w_j to minimize the squared error over the training set with N samples:

$$E(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} (t_i - o_i)^2$$

where N is the number of training samples;
t_i is the target for the ith sample and
o_i is the output for the ith sample.

ADALINE Learning

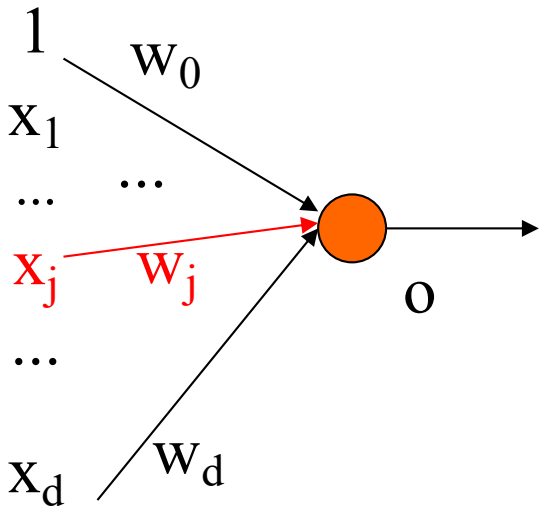
We use gradient descent to learn the weights:

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(t_i - o_i)^2$$

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial (t - o_i)^2}{\partial w_j}$$

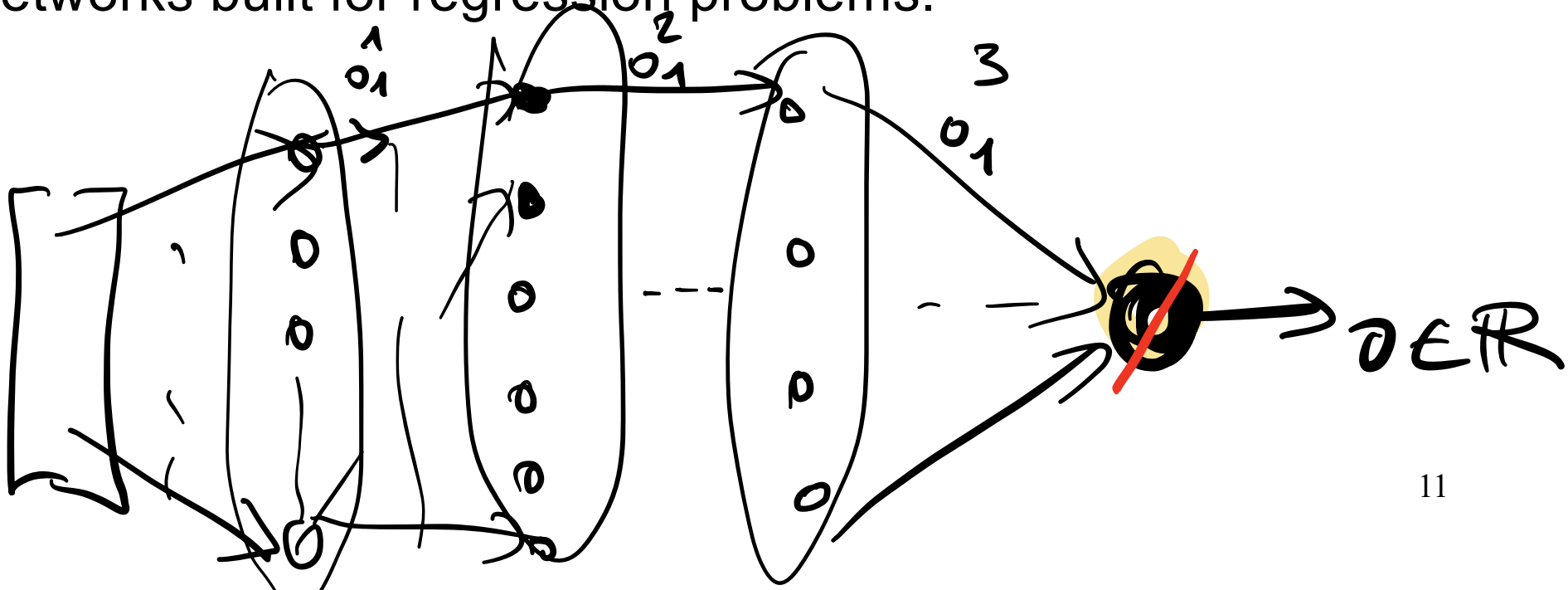
$$= \frac{1}{2N}\sum_{i=1}^{N} 2(t_i - o_i)(-1)\frac{\partial o_i}{\partial w_j}$$

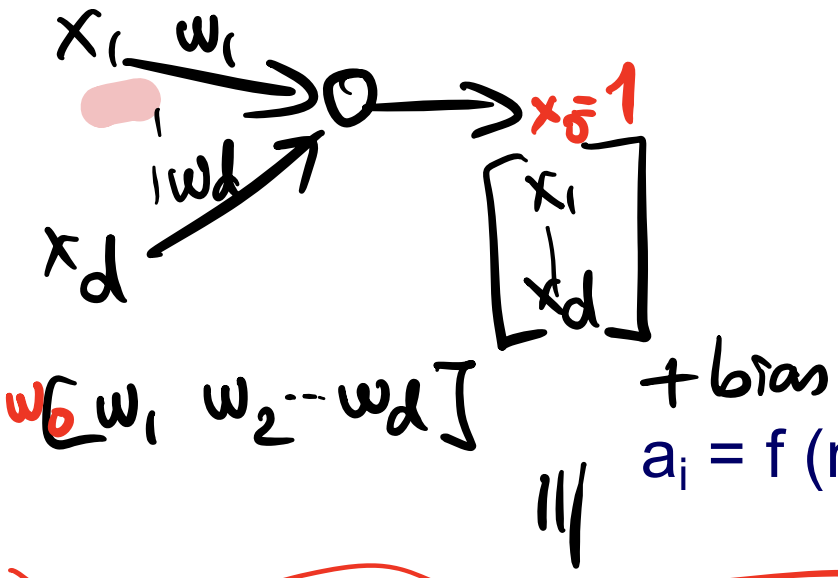
$$= -\frac{1}{N}\sum_{i=1}^{N}(t_i - o_i)\frac{\partial (\mathbf{w}^T \mathbf{x}_i)}{\partial w_j} = -\frac{1}{N}\sum_{i=1}^{N}(t_i - o_i)x_{ij}$$



A single neuron with linear activation may not seem too impressive, but it is nonetheless impressive that a single neuron can learn a regression problem in an adaptive fashion.

A single linear neuron will appear in the output layer of neural networks built for regression problems.





Bias

$$a_i = f(n_i) = f\left(\sum_{j=1}^{m} w_{ij} x_j + b_i\right)$$

An artificial neuron:

- computes the weighted sum of its input and adds its bias
- passes this value (called its **net input**) through an activation function

- This extra free variable (bias) makes the neuron more powerful.
  - Decision boundary does not have to pass through the origin

Bias

Bias can be incorporated as another weight clamped to a fixed input of +1.0, so as to process the computation as matrix multiplication.

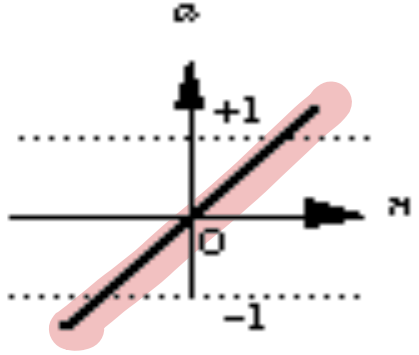
$$a_i = f(n_i) = f\left(\sum_{j=0}^{m} w_{ij} x_j\right) = f(\mathbf{w}_i . \mathbf{x})$$

Activation functions

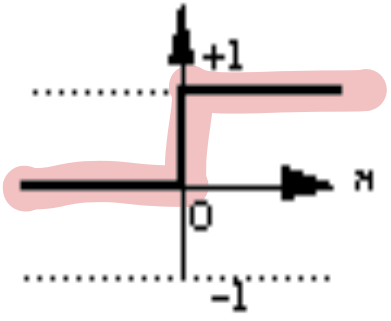
Also called the squashing function as it limits the amplitude of the output of the neuron.

Many types of activations functions are used:

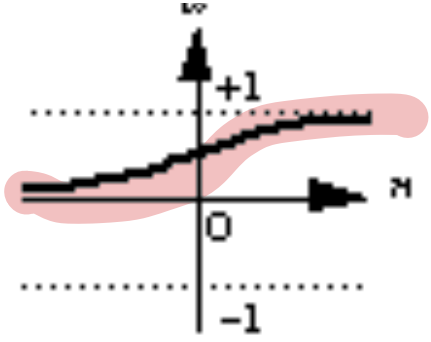
- linear: a = f(net) = net



- threshold: a = {1 if net >= 0
  (hardlimiting)
  0 if net < 0

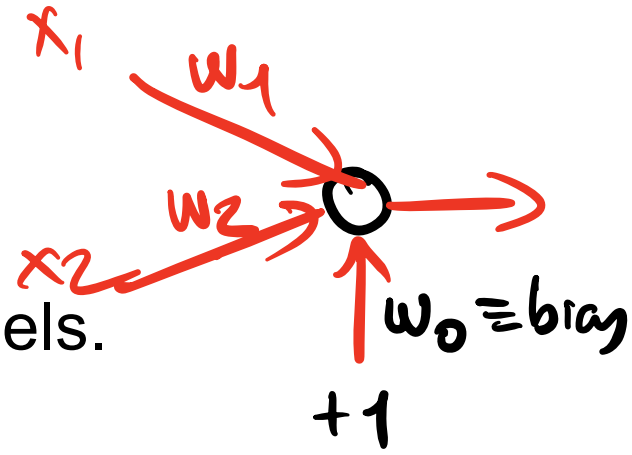


- sigmoid: $a = 1/(1+e^{-net})$



- ...

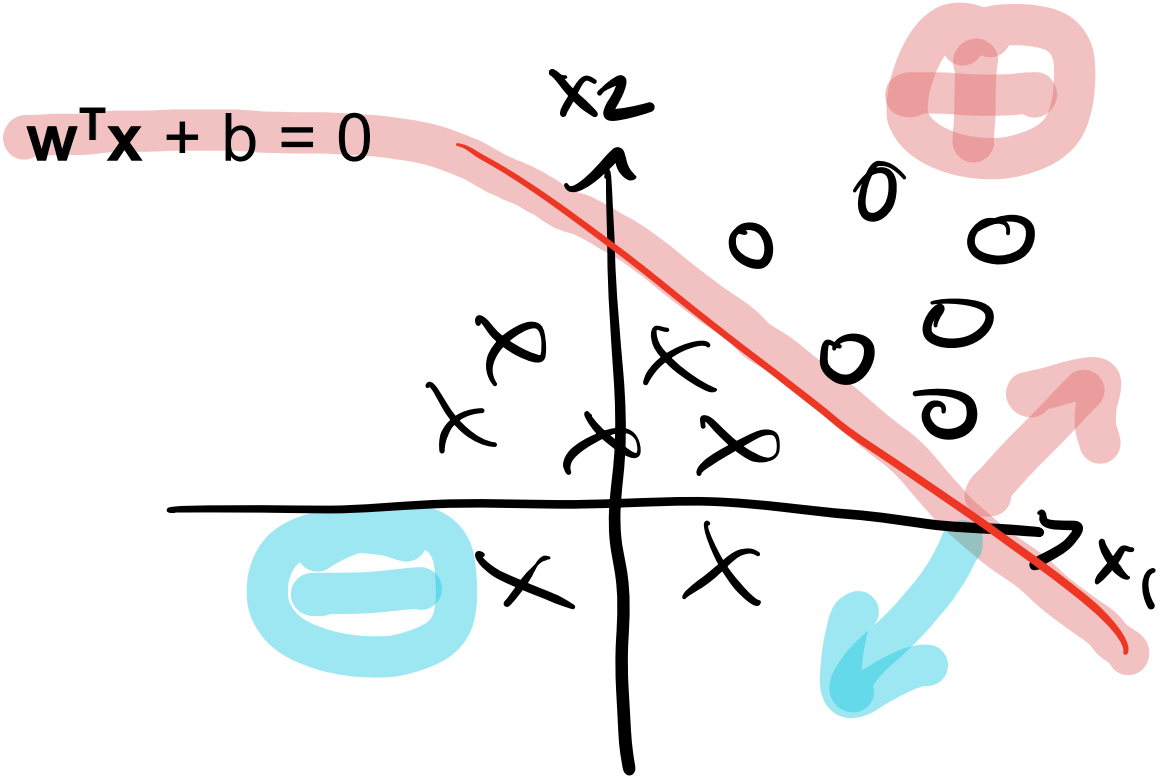
Perceptron



- One of the two original artificial neuron models.
- Uses a hardlimiting (threshold) activation function.

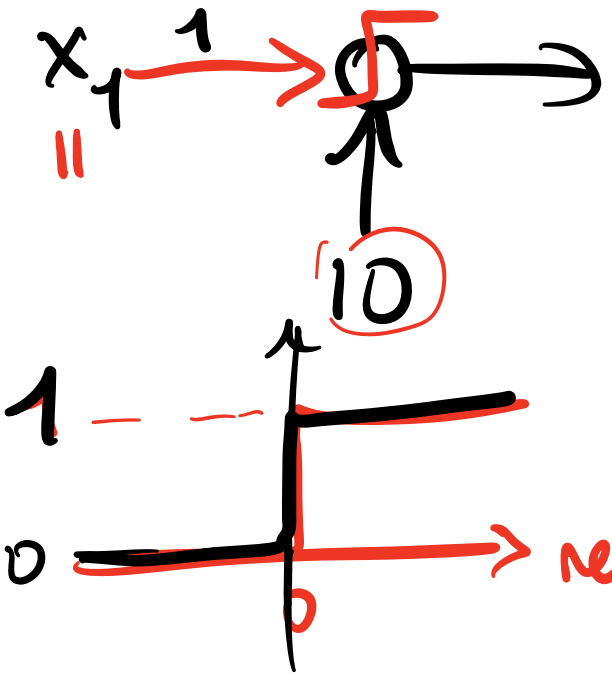
$$w^T x + b \equiv w^T x$$

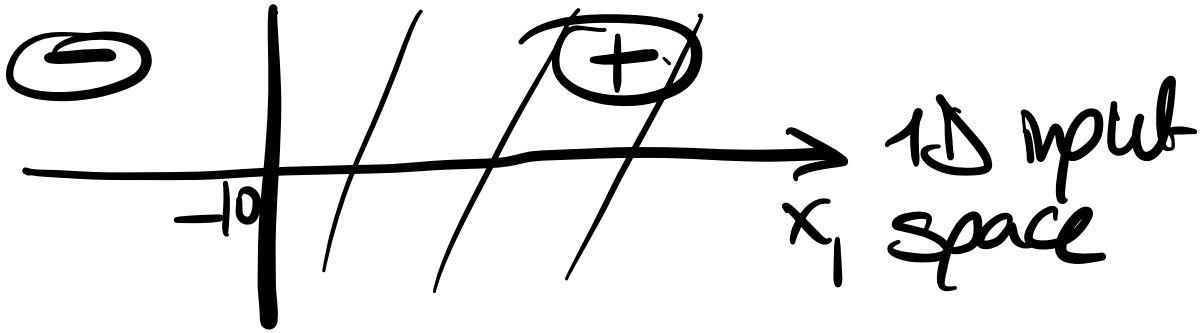
- It effectively separates the input space into two categories by the hyperplane:



➢ *What does this mean?*

Ex. 1





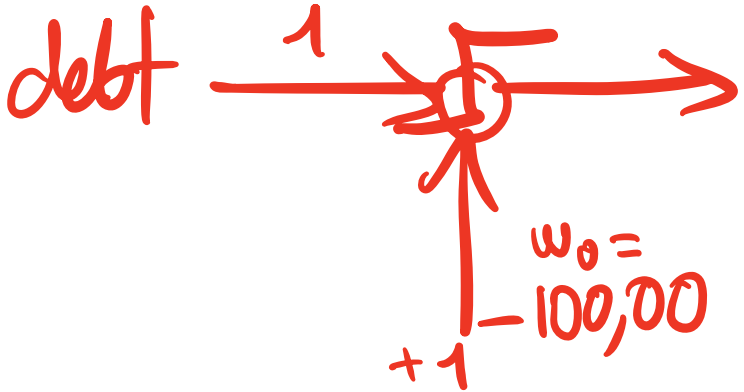
$net = 1 \cdot x_1 + 10$

$net \geq 0$ if $x_1 \geq -10$

$thr(net) = 1$ if $net \geq 0$

$= 0$ ——

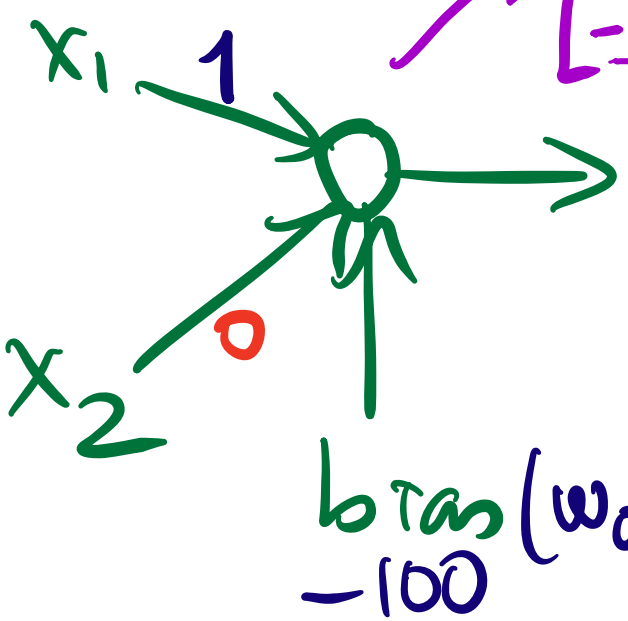
Ex. 2



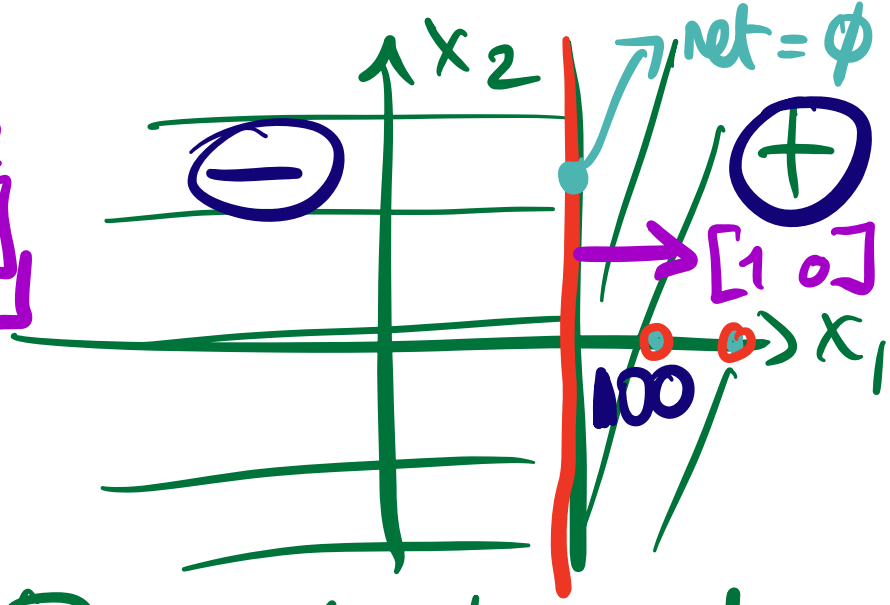
debt $\times 1 - 100,000 \geq 0$

neuron will be active

Ex. 3 :

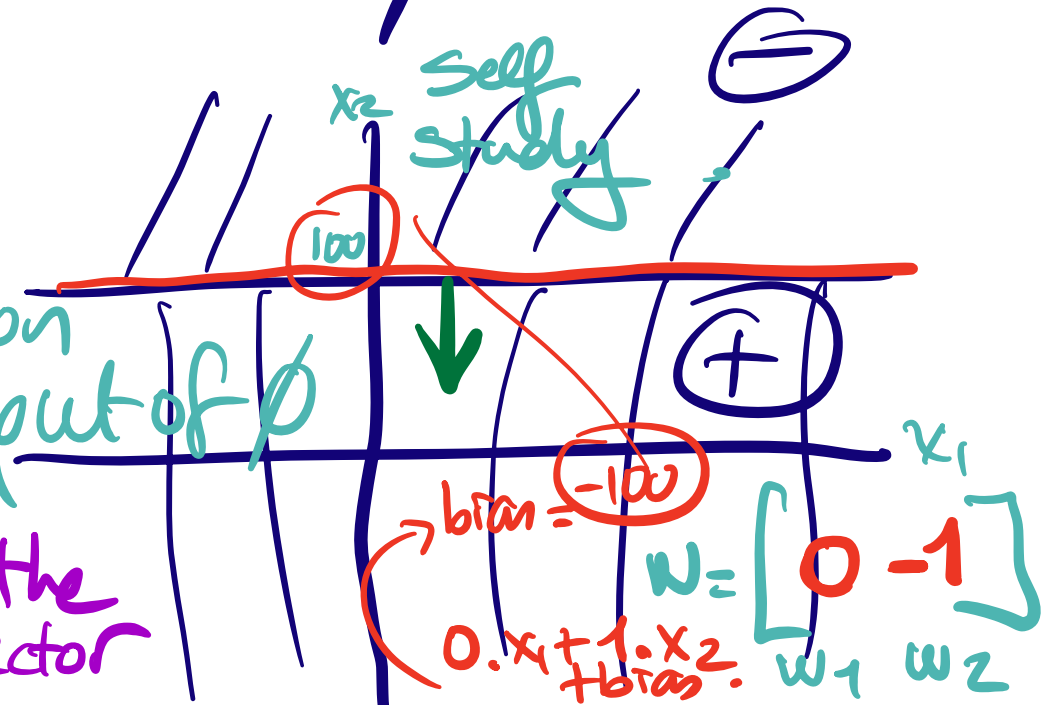


w_0 w_1 w_2
$[-100 \quad 1 \quad 0]$



③ Weight vector points in the direct° of the ⊕ half space.

$$x_1 \cdot w_1 + x_2 \cdot 0 + \text{bias} = \phi$$
$$100 \cdot 1 + \phi +$$

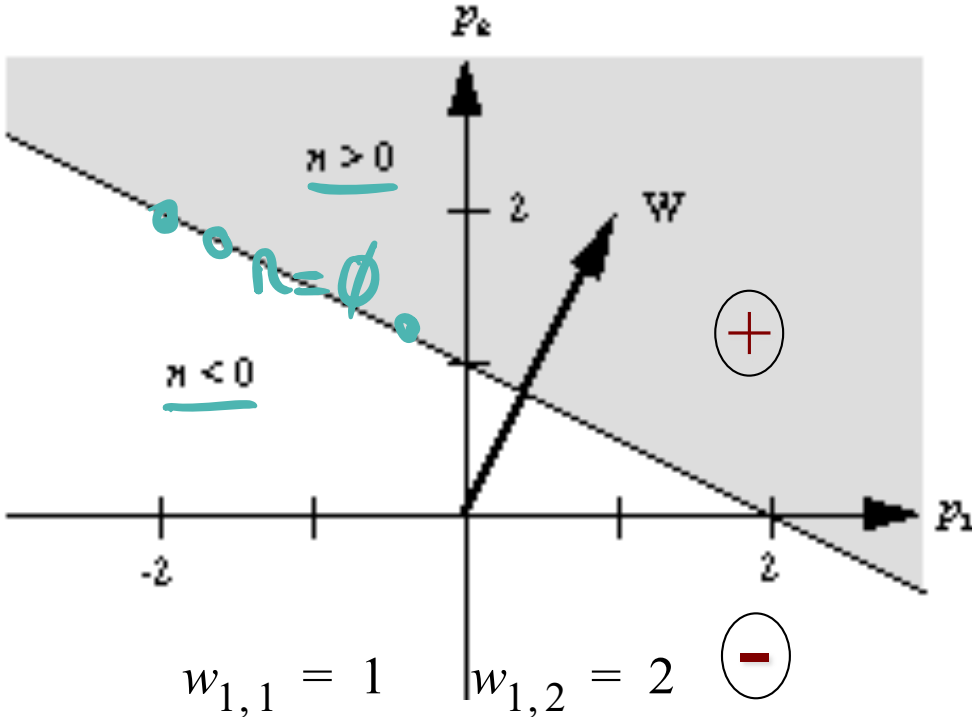


① Points on the decision boundary has net input of ϕ

② Dec. boundary is ⊥ to the weight vector

Two-Input Case

$a = \text{hardlim}(n) = [1\ 2]\mathbf{p} + -2$



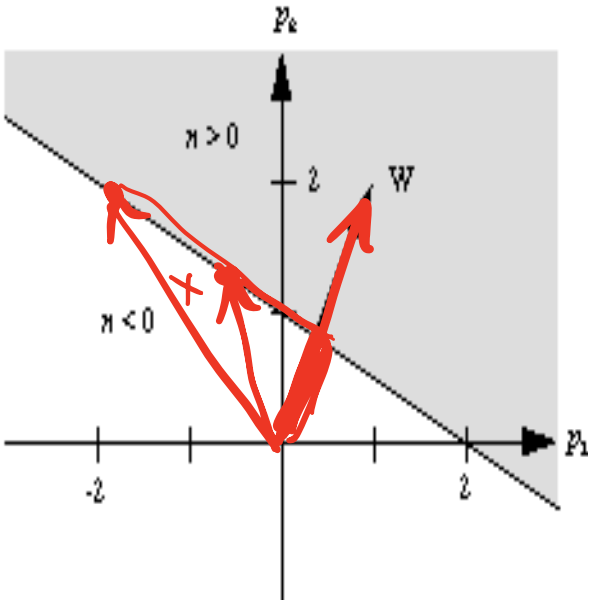
Decision Boundary: all points p for which $\mathbf{w}^T\mathbf{p} + b = 0$

If we know the weights and not the bias, we can take a point on the decision boundary, e.g. $p=[2\ 0]^T$, and if solve for $\mathbf{w}^T\mathbf{p} + b = 0$. Hence $[1\ 2]\mathbf{p} + b = 0$, we see that $b=-2$.

Decision Boundary

2. **The weight vector is orthogonal to the decision boundary**

1. **The weight vector points in the direction of the vector which should produce an output of 1**
   - so that the vectors with the positive output are on the right side of the decision boundary
     - if w pointed in the opposite direction, the dot products of all input vectors would have the opposite sign
     - would result in same classification but with opposite labels



3. **The bias determines the position of the boundary**
   - solve for $\mathbf{w}^T\mathbf{p}+b = 0$ using *one point* on the decision boundary to find b.

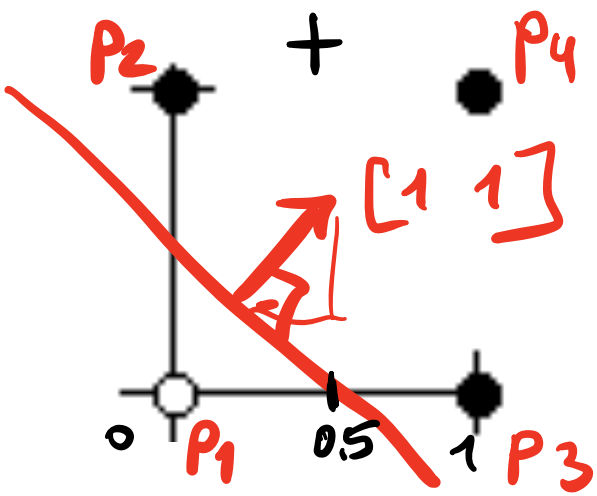
Sample Weights & Corresponding Decsion Boundaries

Shown in prev. hand-drawn slides.

An Illustrative Example

An Illustrative Example: Boolean OR

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\} \quad \left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 1\right\}$$



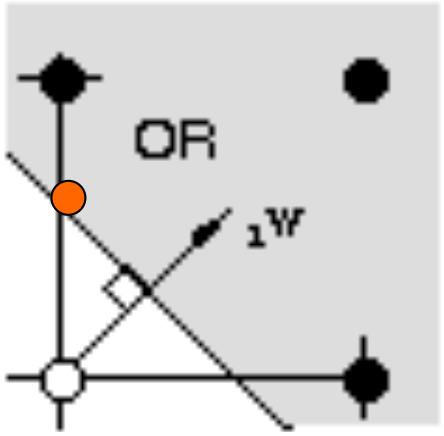
$$\begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}0.5\\0\end{bmatrix} + \text{bias}$$

$$0.5 + \text{bias} = 0$$

$$\Rightarrow \text{bias} = -0.5$$

Given the above input-output pairs (p,t), can you find (manually) the weights of a perceptron to do the job?

Boolean OR Solution



1) Pick an admissable decision boundary

2) Weight vector should be orthogonal to the decision boundary.

$$_1\mathbf{w} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

3) Pick a point on the decision boundary to find the bias.

$$_1\mathbf{w}^T\mathbf{p} + b = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} + b = 0.25 + b = 0 \quad \Rightarrow \quad b = -0.25$$

√√ a diff. bias for $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$

Perceptron Learning Rule

Perceptron Learning Rule (draw network)

How do we find the weights using a learning procedure?

1 – Choose initial weights randomly

2 – Present a randomly chosen pattern

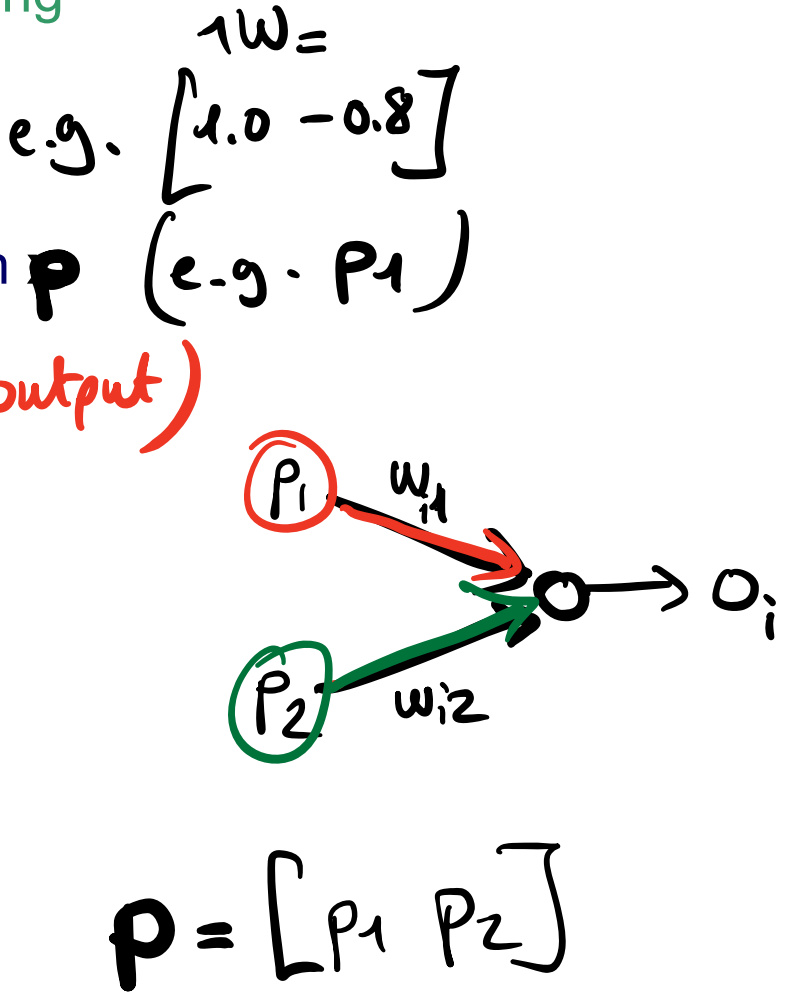
3 – Compute error at node i

4 – Update weights using Delta rule:

$$w_{ij}(t+1) = w_{ij}(t) + err_i * x_j$$

where $err_i = (target_i - output_i)$

4 - Repeat steps 2 and 3 until the stopping criterion (convergence, max number of iterations) is reached



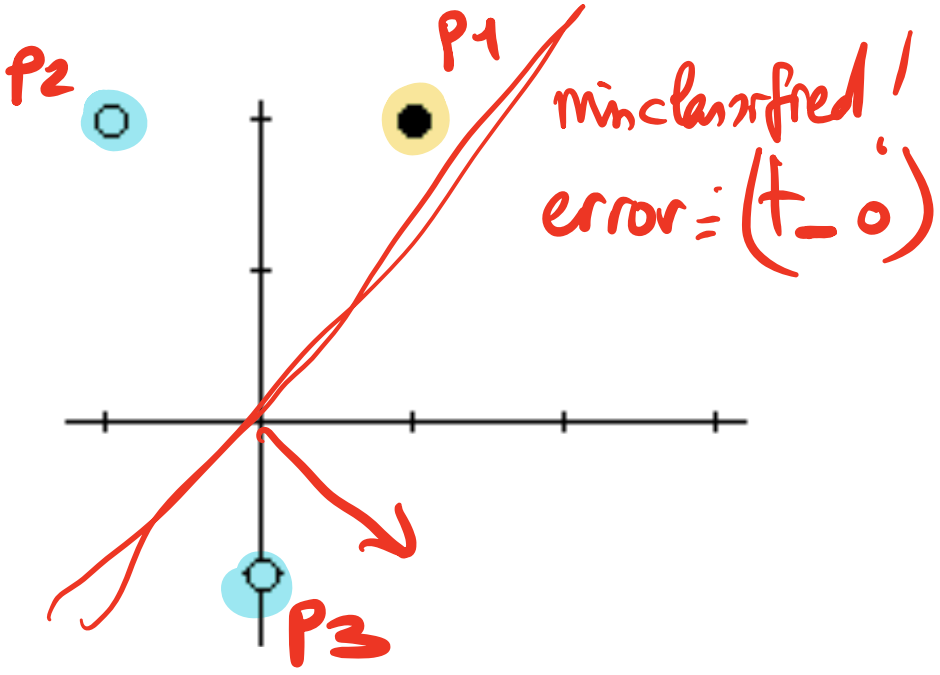
Perceptron Convergence Theorem

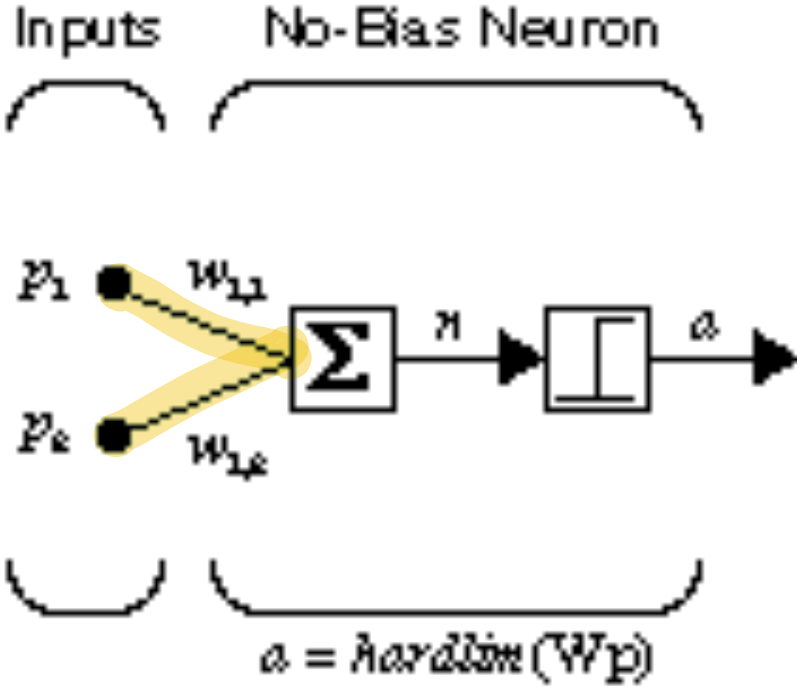
The perceptron rule will always converge to weights which accomplish the desired classification, assuming that such weights exist.

Learning Rule Illustration

Input-output: $\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = 1\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}-1\\2\end{bmatrix}, t_2 = 0\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 0\right\}$$

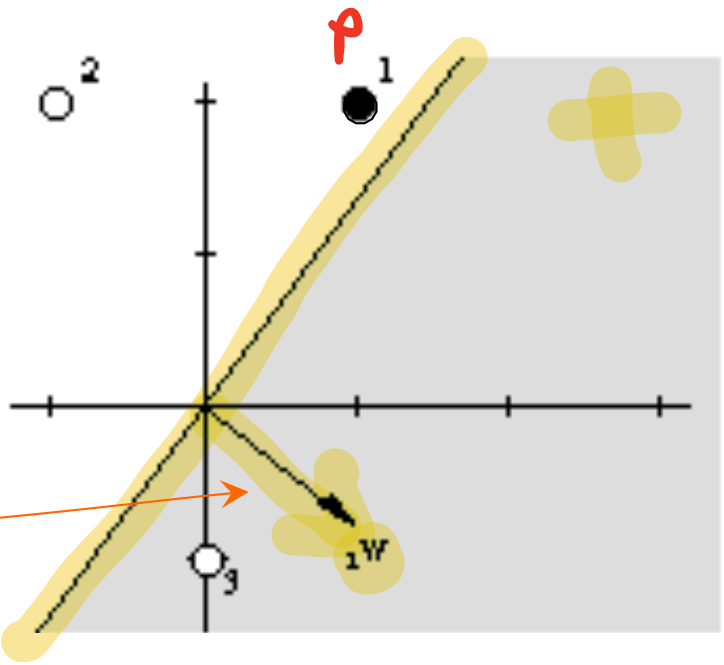




Starting Point

Random initial weight:

$$ {}_1\mathbf{w} = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} $$



Present $\mathbf{p}_1$ to the network:

$$ a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) $$

$$ a = hardlim(-0.6) = 0 $$

Incorrect Classification.

err = 1 = (t - o)

Tentative Learning Rule

${}_1\mathbf{w}$ needs to point more towards $\mathbf{p}_1$ ⟶ Add $\mathbf{p}_1$ to ${}_1\mathbf{w}$

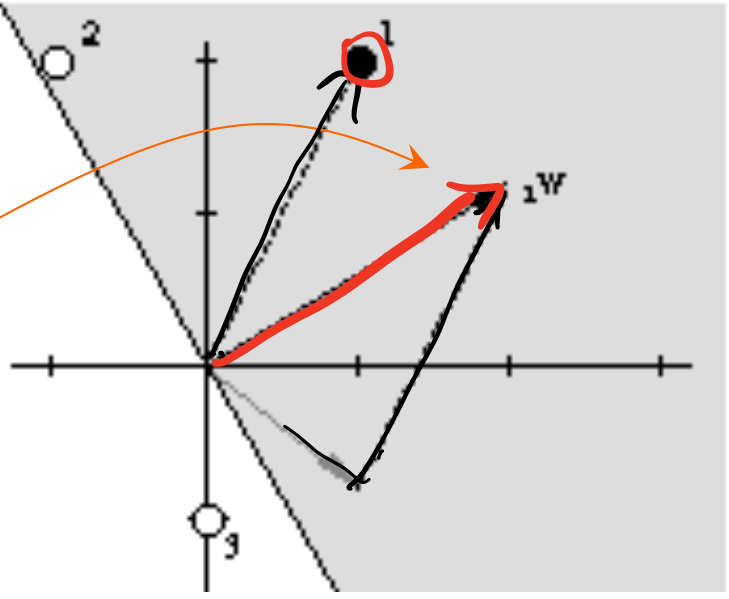
$$w_1(t+1) = w_1(t) + err \cdot p_1$$
$$w_2(t+1) = w_2(t) + err \cdot p_2$$

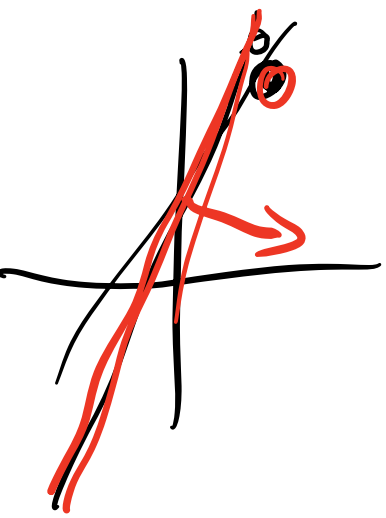
Tentative Rule: If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \overset{err}{\cdot}\mathbf{p}_1 = \begin{bmatrix}1.0\\-0.8\end{bmatrix} + \begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}2.0\\1.2\end{bmatrix}$$

$w^{old} + 1 \cdot p_1$

err.





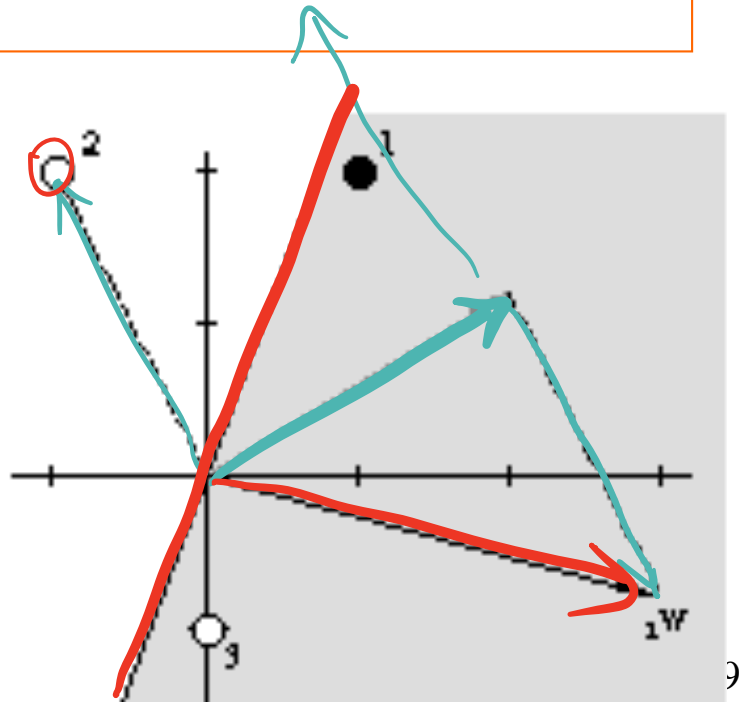
Second Input Vector

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_2) = hardlim\left(\begin{bmatrix}2.0 & 1.2\end{bmatrix}\begin{bmatrix}-1\\2\end{bmatrix}\right)$$

$$a = hardlim(0.4) = 1 \quad \text{(Incorrect Classification)}$$

Modification to Rule: If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix}2.0\\1.2\end{bmatrix} - \begin{bmatrix}-1\\2\end{bmatrix} = \begin{bmatrix}3.0\\-0.8\end{bmatrix}$$

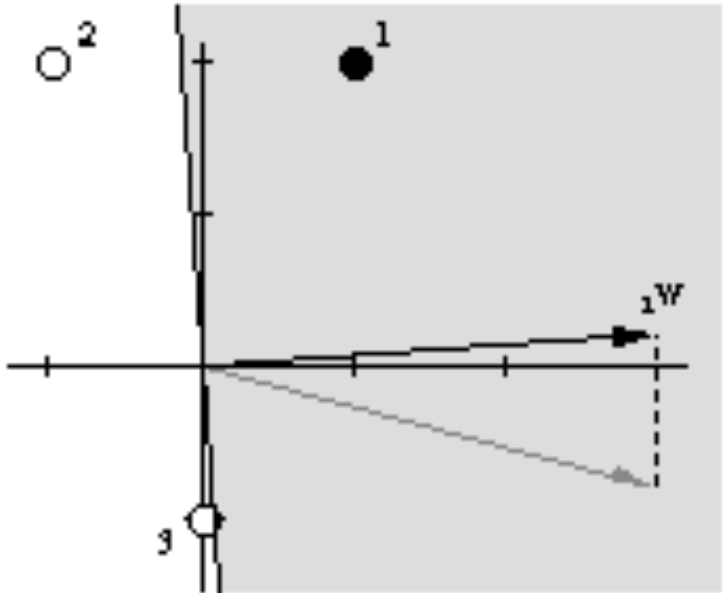


Third Input Vector

$$a = hardlim({}_1\mathbf{w}^T \mathbf{p}_3) = hardlim\left(\begin{bmatrix}3.0 & -0.8\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix}\right)$$

$$a = \text{hardlim}(0.8) = 1 \qquad \text{(Incorrect Classification)}$$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix}3.0\\-0.8\end{bmatrix} - \begin{bmatrix}0\\-1\end{bmatrix} = \begin{bmatrix}3.0\\0.2\end{bmatrix}$$



Patterns are now correctly classified.

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$.

Unified Learning Rule

If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$\equiv w^{old} + err \cdot p$ (err = 1, −1, 0; p = p, p, φ)

Unify using an error term:

$$e = t - a$$

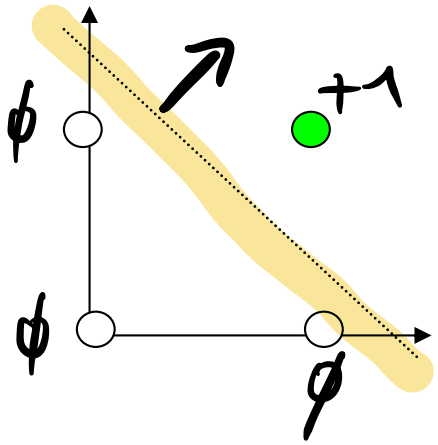
$${}_1w^{new} = {}_1w^{old} + ep$$

$$b^{new} = b^{old} + e$$

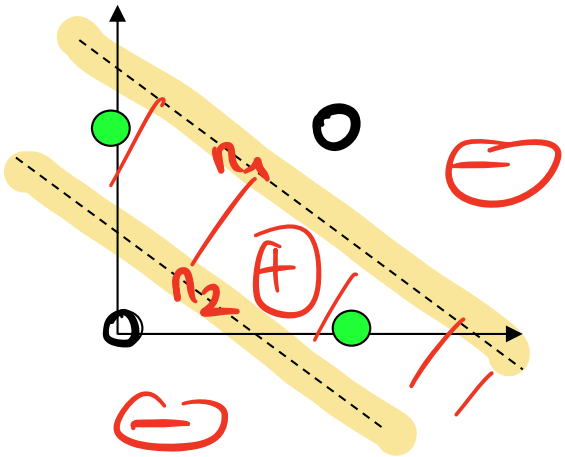
Perceptron Limitations

Perceptron Limitations

- A single layer perceptron can only learn linearly separable problems.
  - Boolean AND function is linearly separable, whereas Boolean XOR function is not.



Boolean AND



Boolean XOR

output = A AND B
x_1 AND x_2

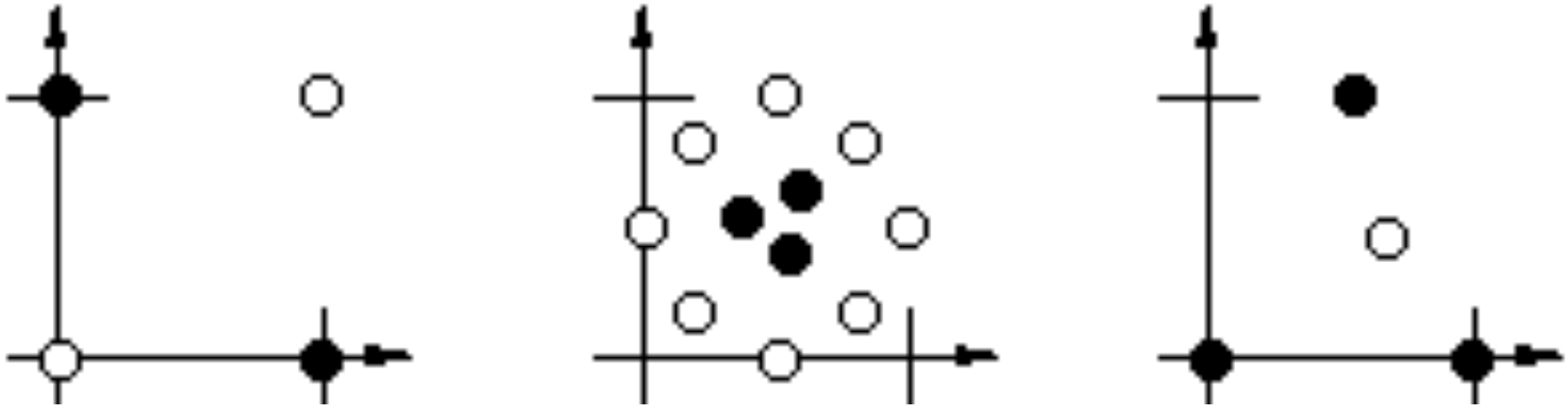
OR		XOR
0 0	0	0
0 1	1	1
1 0	1	1
1 1	1	ϕ

Perceptron Limitations

Linear Decision Boundary

$$_1\mathbf{w}^T\mathbf{p} + b = 0$$

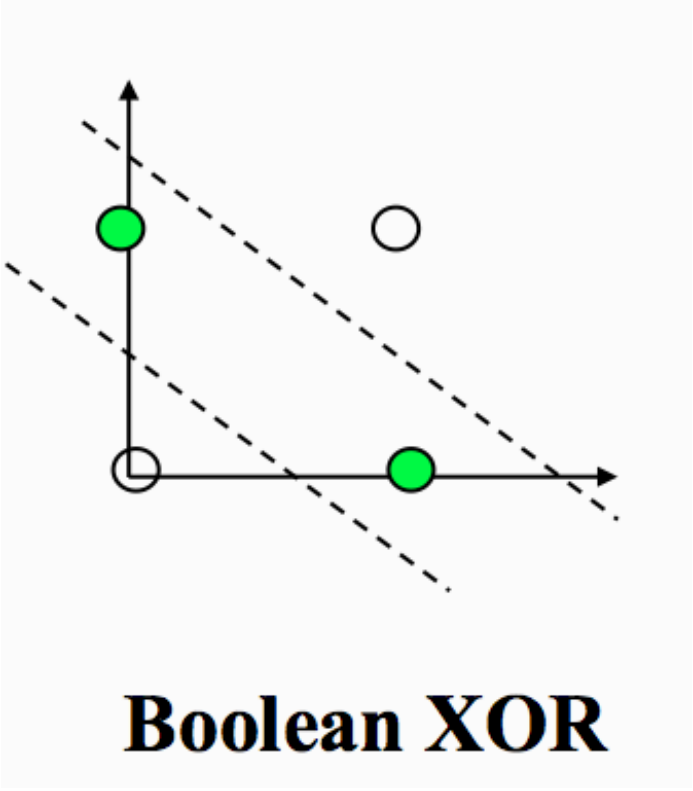
Linearly Inseparable Problems



Perceptron Limitations

For a linearly not-separable problem:

- Would it help if we use more layers of neurons?
- What could be the learning rule for each neuron?



Boolean XOR

Solution: Multilayer networks and the backpropagation learning algorithm

- More than one layer of perceptrons (with a hardlimiting activation function) can learn any Boolean function.
- However, a learning algorithm for multi-layer perceptrons has not been developed until much later
  - backpropagation algorithm
  - replacing the hardlimiter in the perceptron with a sigmoid activation function

Historical Sketch

Pre-1940: von Hemholtz, Mach, Pavlov, etc.

- General theories of learning, vision
- No specific mathematical models of neuron operation

1940s: Hebb, McCulloch and Pitts

- **Hebb: Explained mechanism for learning in biological neurons**
- **McCulloch and Pitts: First neural model**

1950s: Rosenblatt, Widrow and Hoff

- **First practical networks (Perceptron and Adaline) and corresponding learning rules**

1960s: Minsky and Papert

- **Demonstrated limitations of existing neural networks**
- **Most research suspended**

1970s: Amari, Anderson, Fukushima, Grossberg, Kohonen

- Progress continues, although at a slower pace

1980s: Grossberg, Hopfield, Kohonen, Rumelhart, etc.

- **Backpropagation algorithm,...**

2000s: Hinton, LeCun, Bengio et al.

- **Deep learning arrives**