Nöron Ağlarına Giriş
(Introduction to Neural Networks)

Perceptron

Slides modified from Neural Network Design

by Hagan, Demuth and Beale

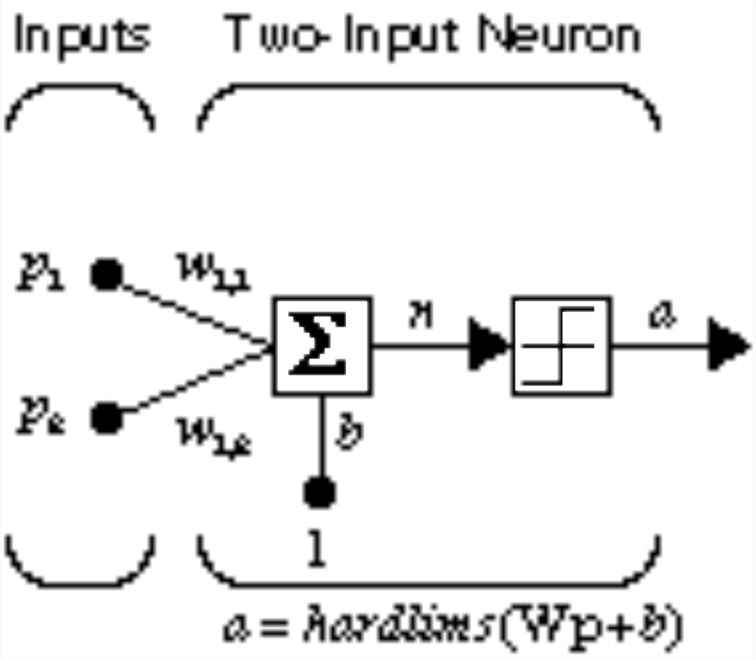
(Berrin Sabancı)

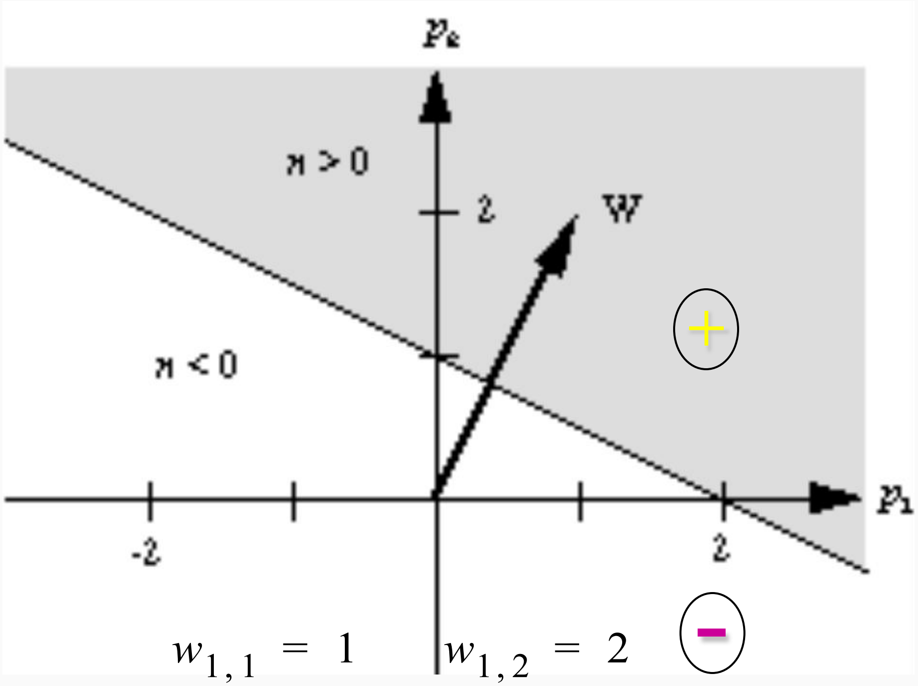
Perceptron

- A single artificial neuron that computes its weighted input and uses a threshold activation function.
- It is also called a TLU (Threshold Logic Unit)
- It effectively separates the input space into two categories by the hyperplane:

$$\mathbf{w}^T\mathbf{x} + b_i = 0$$

Two-Input Case



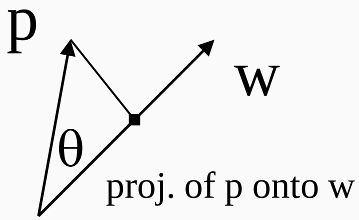


$w_{1,1} = 1 \qquad w_{1,2} = 2$

$$a = hardlims(n) = hardlims(\begin{bmatrix}1 & 2\end{bmatrix}\mathbf{p} + (-2))$$

Decision Boundary

$$\mathbf{Wp} + b = 0 \qquad \begin{bmatrix}1 & 2\end{bmatrix}\mathbf{p} + (-2) = 0$$



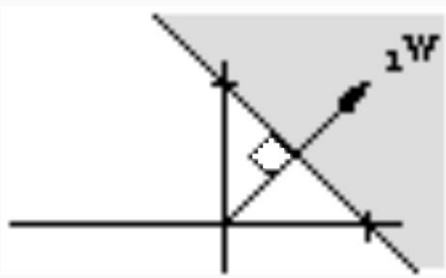
Decision Boundary

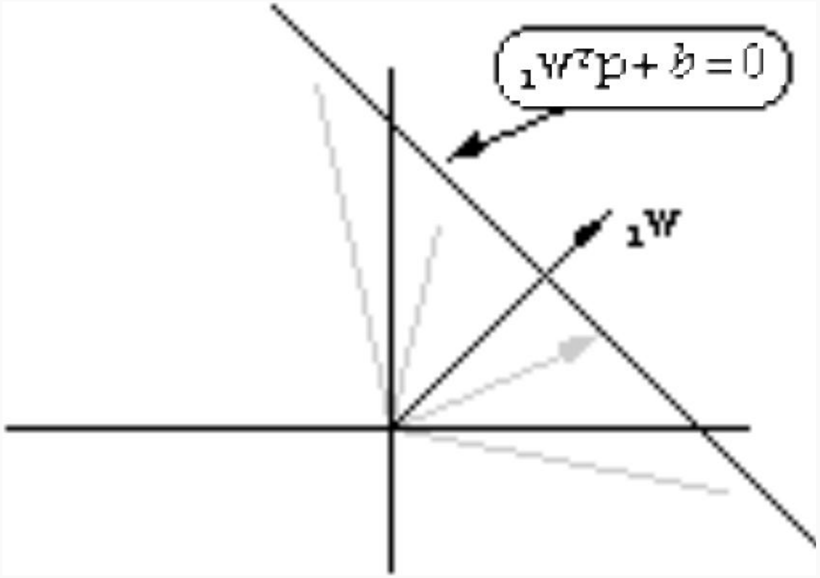
$w^T.p$ = ||w||||p||Cosθ

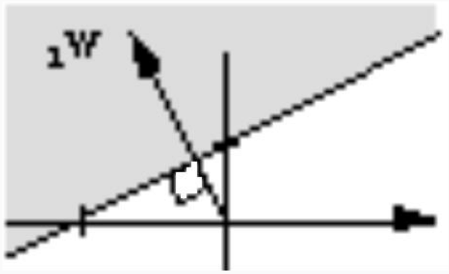
proj. of p onto w
= ||p||Cosθ
= $w^T.p$/||w||

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

- All points on the decision boundary have the same inner product (= -b) with the weight vector
- Therefore they have the same projection onto the weight vector; so they must lie on a line orthogonal to the weight vector







Decision Boundary

The weight vector should be orthogonal to the decision boundary

- see previous slide

The weight vector should point 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 opposite labels)

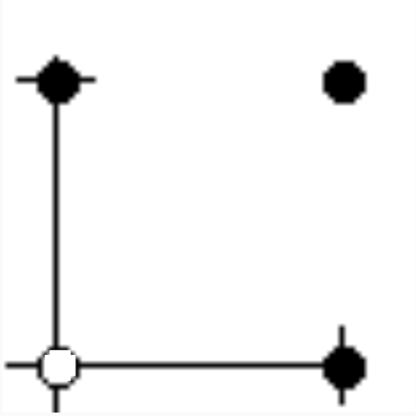
The bias determines the position of the boundary

- solve for wp+b = 0 to find the decision boundary

An Illustrative Example

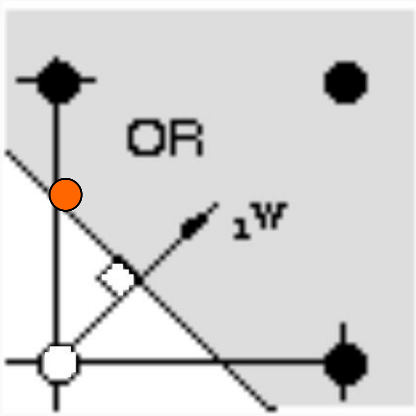
Boolean OR

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



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

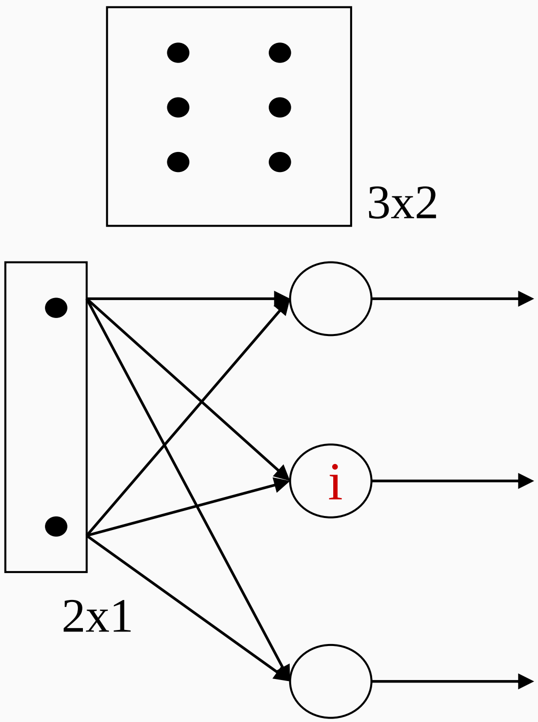
2) Weight vector should be orthogonal to the decision boundary.

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

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

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

Multiple-Neuron Perceptron



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix} \qquad {}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}$$

$$\boxed{a_i = hardlim(n_i) = hardlim({}_i\mathbf{w}^T\mathbf{p} + b_i)}$$

Multiple-Neuron Perceptron

Each neuron will have its own decision boundary.

$$ {}_i\mathbf{w}^T \mathbf{p} + b_i = 0 $$

A single neuron can classify input vectors into two categories.

An S-neuron perceptron can classify input vectors into 2^S categories.

Perceptron Learning Rule

Types of Learning

- Supervised Learning

  Network is provided with a set of examples of proper network behavior (inputs/targets)

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

- Reinforcement Learning

  Network is only provided with a grade, or score, which indicates network performance

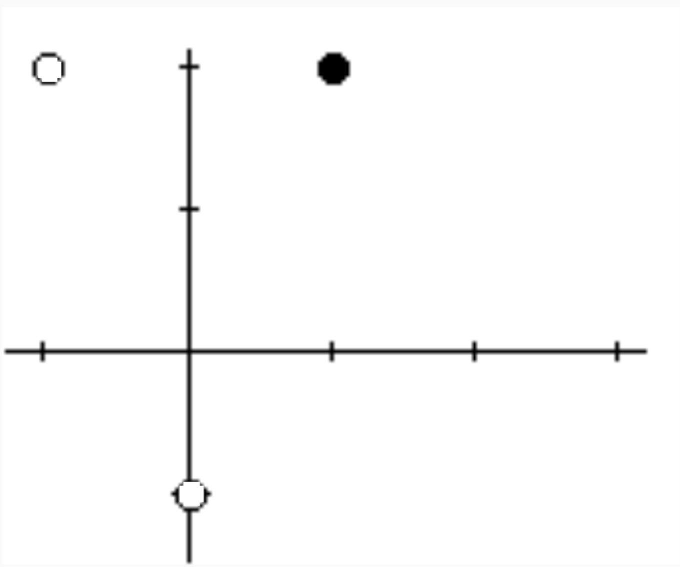
- Unsupervised Learning

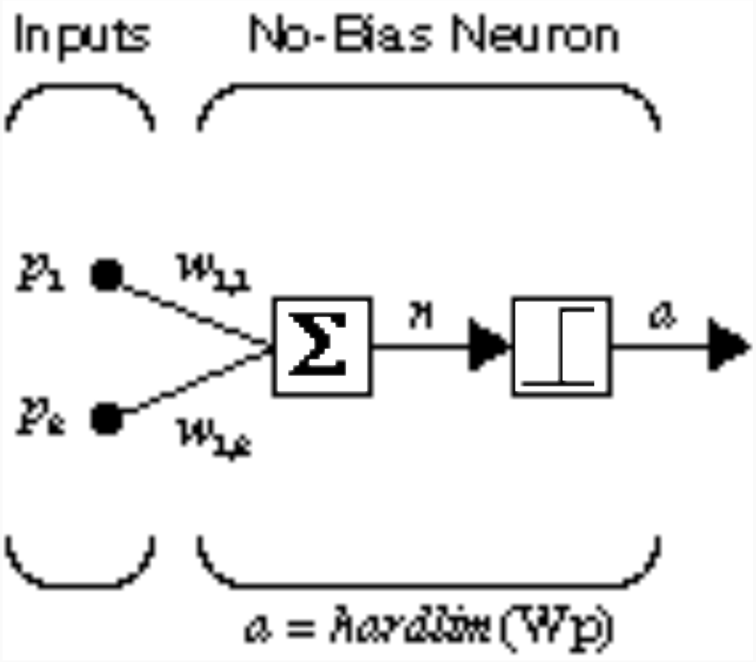
  Only network inputs are available to the learning algorithm. Network learns to categorize (cluster) the inputs.

Learning Rule Test Problem

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

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

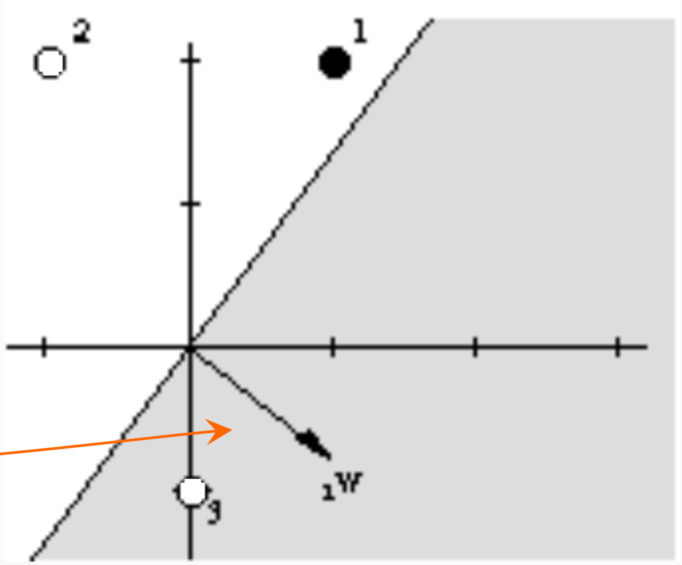




Starting Point

Random initial weight:

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



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

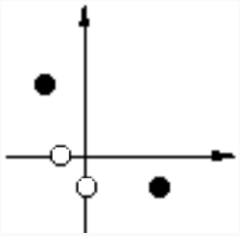
$$ a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) $$

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

Incorrect Classification.

Tentative Learning Rule

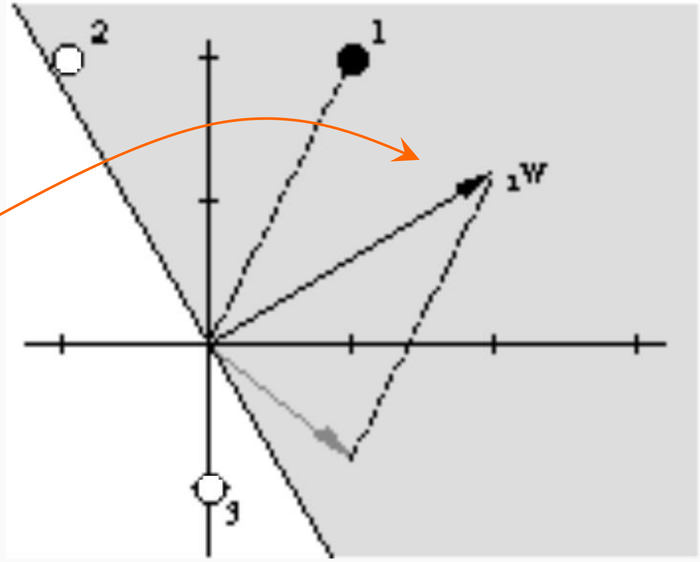
- Set ${}_1\mathbf{w}$ to $\mathbf{p}_1$ ✗
  - Not stable



- Add $\mathbf{p}_1$ to ${}_1\mathbf{w}$ ✓

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

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



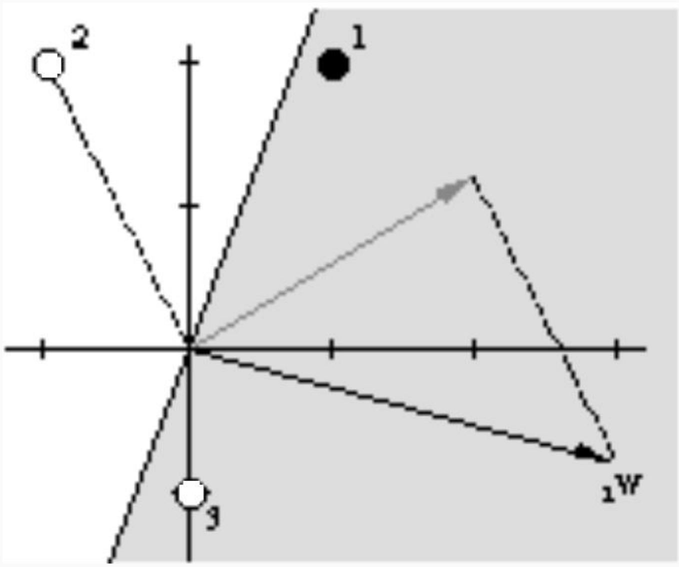
Second Input Vector

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

$$a = hardlim(0.4) = 1$$ (Incorrect Classification)

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

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

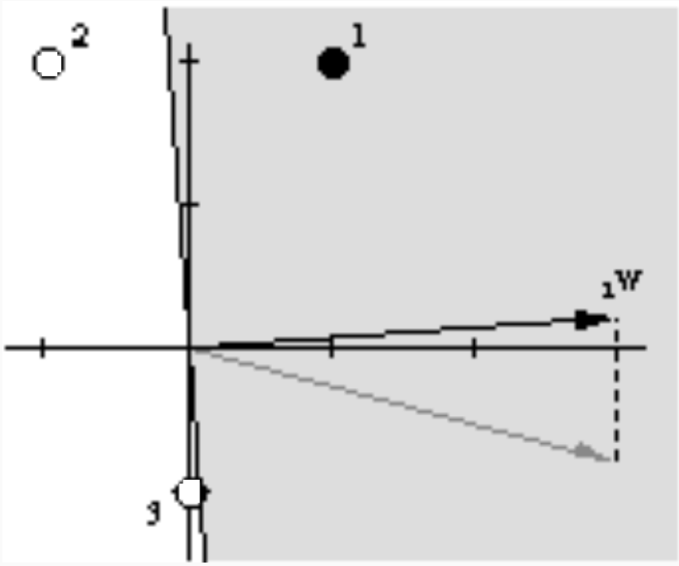


Third Input Vector

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

$$a = \mathrm{hardlim}(0.8) = 1$$ (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.

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

Unified Learning Rule

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

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

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

Unified Learning Rule

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

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

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

Define: $e = t - a$

$$\text{If } e = 1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$$

$$\text{If } e = -1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$$

$$\text{If } e = 0, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$$

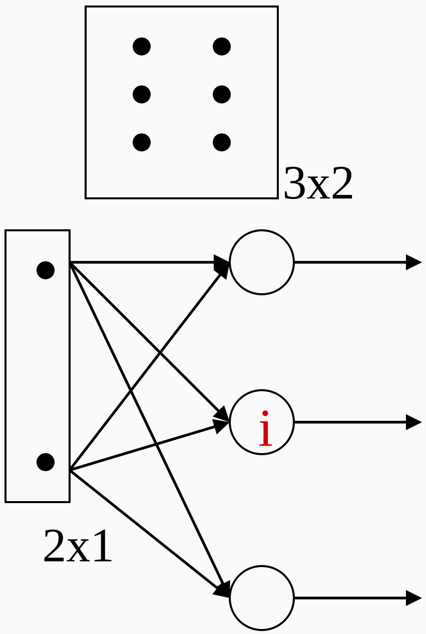
=>

$$\boxed{\begin{gathered} {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t - a)\mathbf{p} \\ b^{new} = b^{old} + e \end{gathered}}$$

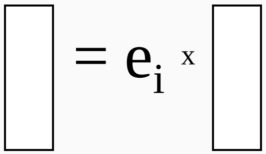
A bias is a weight with an input of 1.

Multiple-Neuron Perceptrons

To update the ith row of the weight matrix:



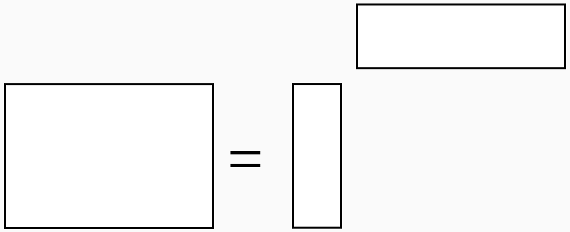
$${}_i\mathbf{w}^{new} = {}_i\mathbf{w}^{old} + e_i\,\mathbf{p}$$



$$b_i^{new} = b_i^{old} + e_i$$

Matrix form:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T$$



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

You should not need it, but if you were to write your own NN toolbox, you need to use matrices in order to greatly improve speed compared to dummy algorithm.

Perceptron Learning Rule (Summary)

How do we find the weights using a learning procedure?

1 - Choose initial weights randomly

2 - Present a randomly chosen pattern x

3 - 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 Thm.

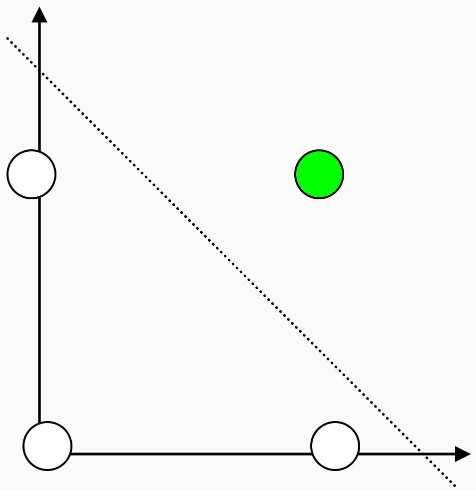
The perceptron rule will always converge to weights which accomplish the desired classification, assuming that such weights exist.

Perceptron Limitations

Perceptron Limitations

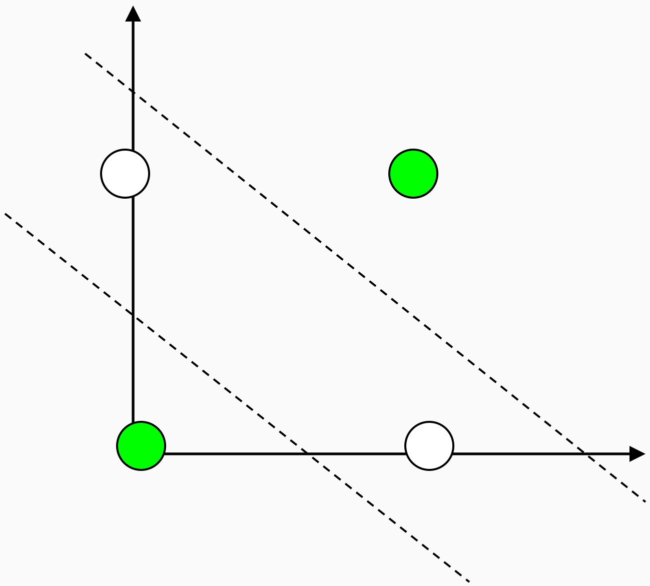
- A single layer perceptron can only learn linearly separable problems.
  - Boolean AND function is linearly separable, whereas Boolean XOR function (and the parity problem in general) is not.

Linear Separability



Boolean AND

✓



Boolean XOR

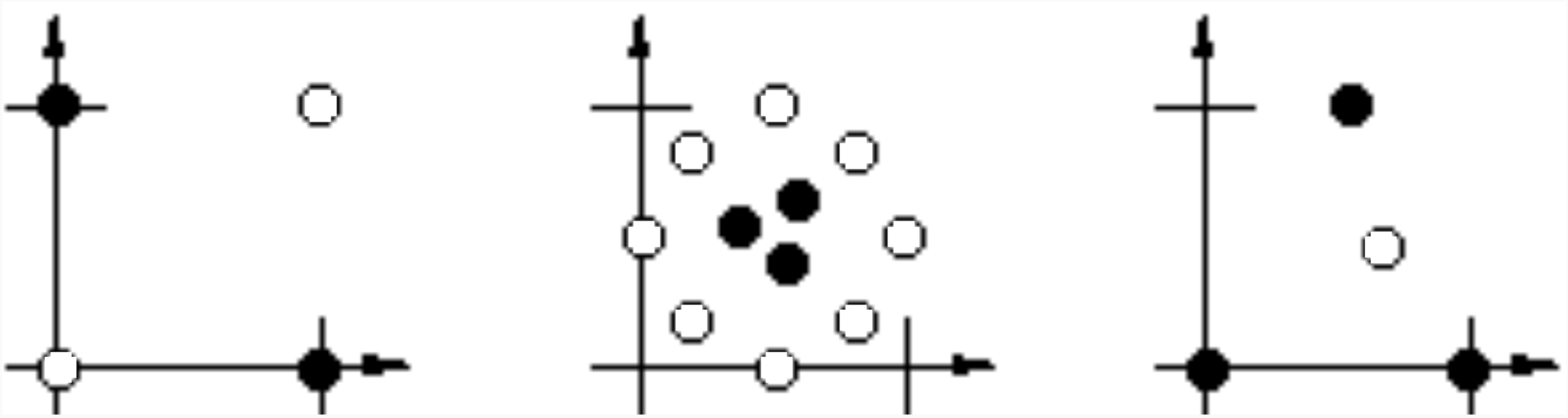
✗

Perceptron Limitations

Linear Decision Boundary

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

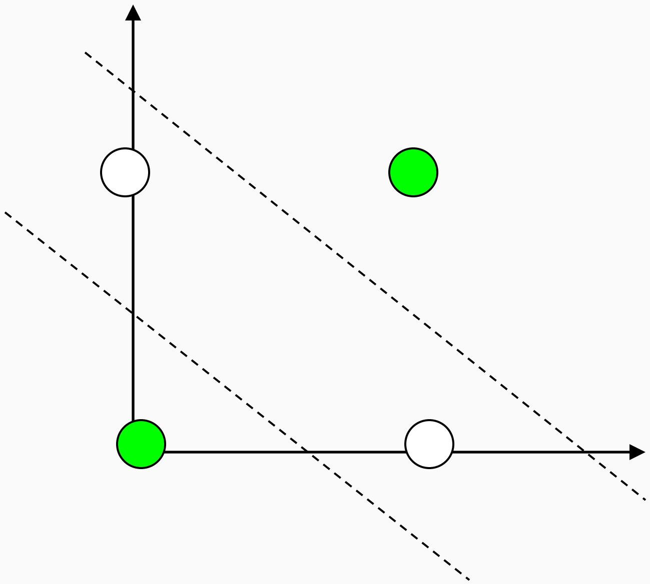
Linearly Inseparable Problems



Perceptron Limitations

XOR problem: What if we use more layers of neurons in a perceptron?

- Each neuron implementing one decision boundary and the next layer combining the two?



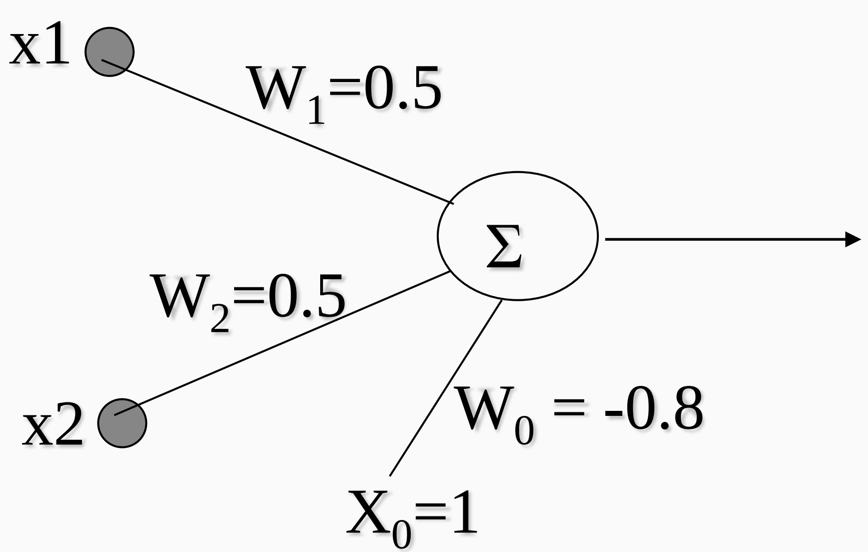
What could be the learning rule for each neuron?

Multilayer networks and the backpropagation learning algorithm

- Perceptrons (in this context of limitations, they refer to single layer perceptrons) can learn many Boolean functions:
    - AND, OR, NAND, NOR, but not XOR

- Multi-layer perceptron can solve this problem

AND:



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 with a sigmoid activation function)

History of Artificial Neural Networks (ANNs)

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

- General theories of learning, vision, conditioning
- 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
- New learning algorithms not forthcoming, **most research suspended**

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

- Progress continues, although at a slower pace

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

- Important new developments cause a **resurgence in the field** (Backpropagation algorithm)

History of Artificial Neural Networks (Details)

- **McCulloch and Pitts (1943):** first neural network model
- **Hebb (1949):** proposed a mechanism for learning, as increasing the synaptic weight between two neurons, by repeated activation of one neuron by the other across that synapse (lacked the inhibitory connection)
- **Rosenblatt (1958):** Perceptron network and the associated learning rule
- **Widrow & Hoff (1960):** a new learning algorithm for linear neural networks (ADALINE)
- **Minsky and Papert (1969):** widely influential book about the limitations of single-layer perceptrons, causing the research on NNs mostly to come to an end.
- **Some that still went on:**
  - **Anderson, Kohonen (1972):** Use of ANNs as associative memory
  - **Grossberg (1980):** Adaptive Resonance Theory
  - **Hopfield (1982):** Hopfield Network
  - **Kohonen (1982):** Self-organizing maps
- **Rumelhart and McClelland (1982):** Backpropagation algorithm for training multilayer feed-forward networks. Started a resurgence on NN research again.