History of the Perceptron

Dr Dayal Kumar Behera

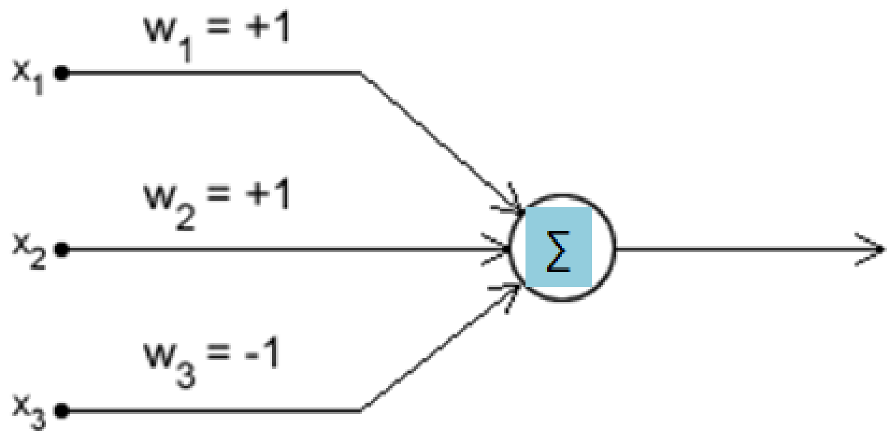
McCulloch-Pitts Neuron

- The first computational model of a neuron was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943.

McCulloch-Pitts Neuron

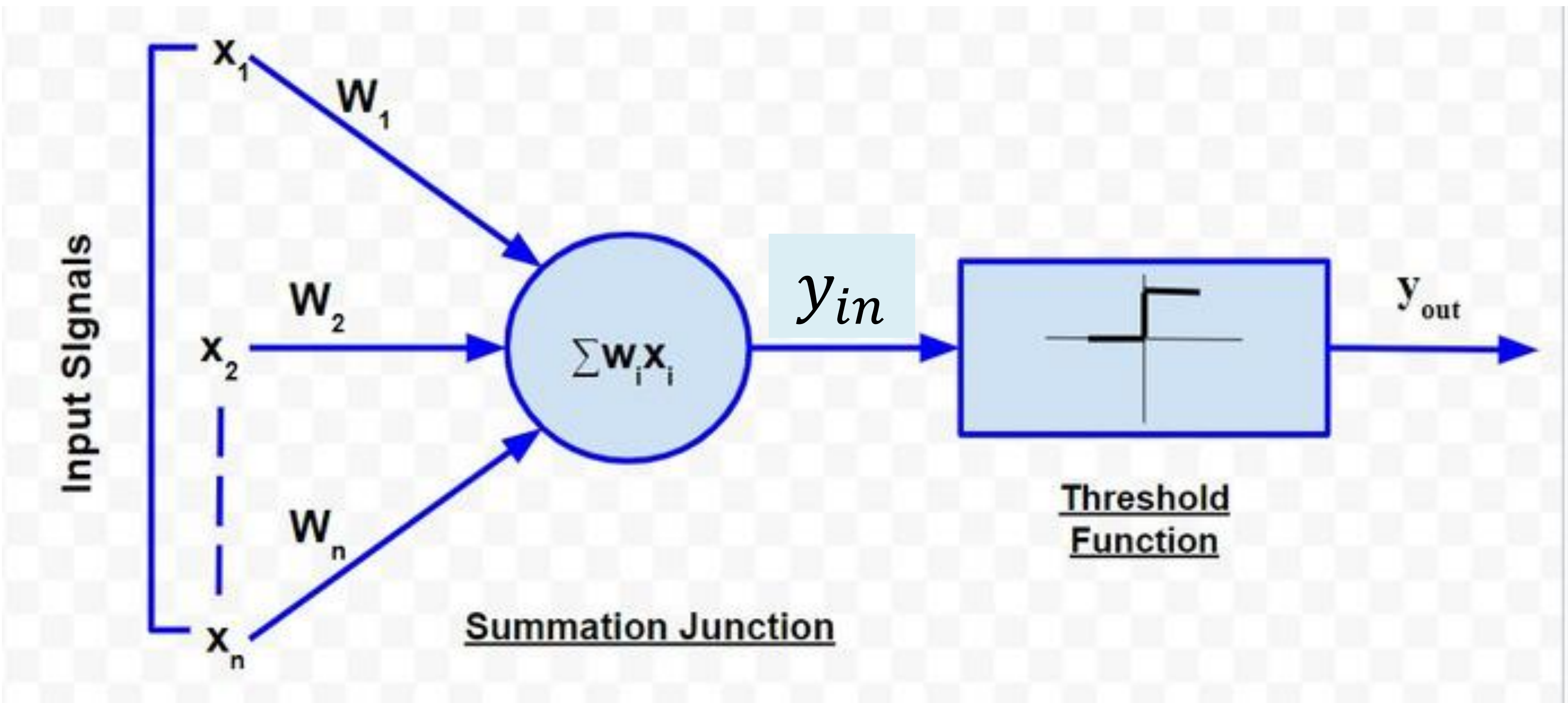
- The inputs of the McCulloch-Pitts neuron could be either 0 or 1.
- The output could be 0 or 1.
- Each input could be either excitatory or inhibitory. if a weight is 1, it is an excitatory input. If weight is -1, it is an inhibitory input.
- It has a threshold function as an activation function.

McCulloch-Pitts Neuron



Excitatory input ?
Inhibitory input?

McCulloch-Pitts Neuron

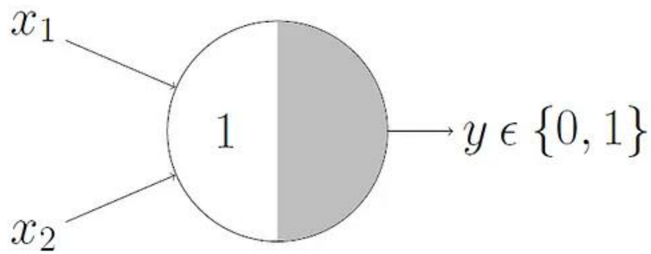


Example: Logic Gate OR

x_1	x_2	Y_{in}	y_{out}
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

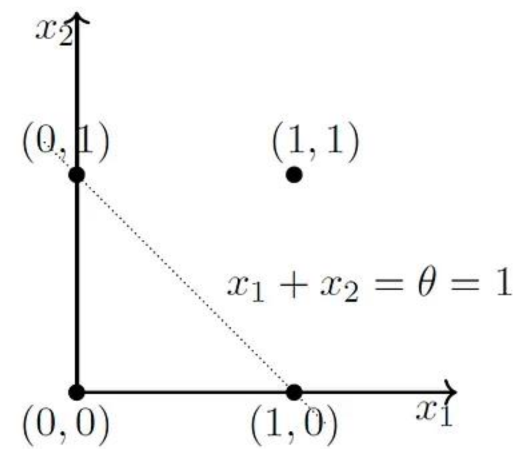
w1 = 1, w2 = 1 and Threshold (θ) = 1

Geometric Interpretation of M-P Model: OR Gate



$OR\ function$

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$



$$w1x1 + w2x2 + w0 \ = 0$$

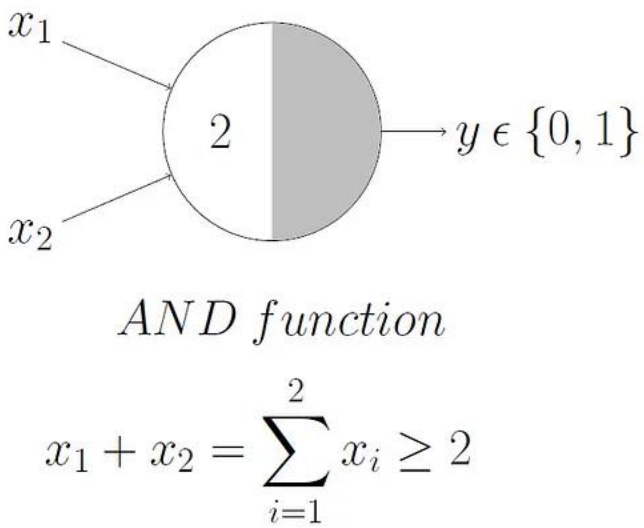
$$x2 = \frac{-w1}{w2}\, x1 + \ \frac{-w0}{w2}$$

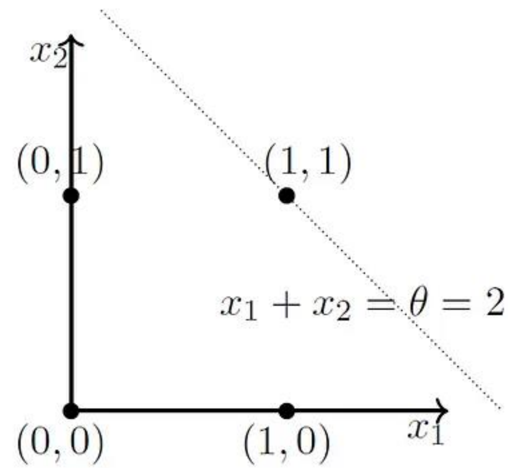
Separating line equation

Here, w1=1, w2=1, w0=-1

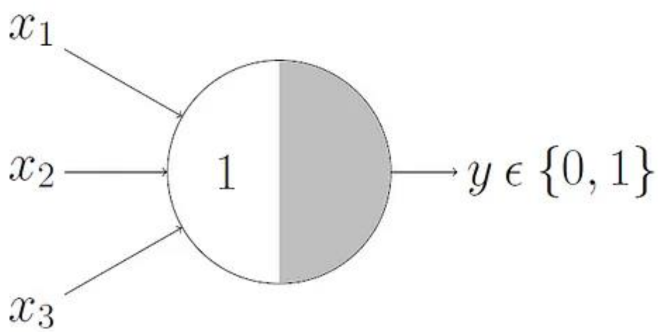
Let x1 = 1 ➔ x2 = 0

Geometric Interpretation of M-P Model: AND Gate



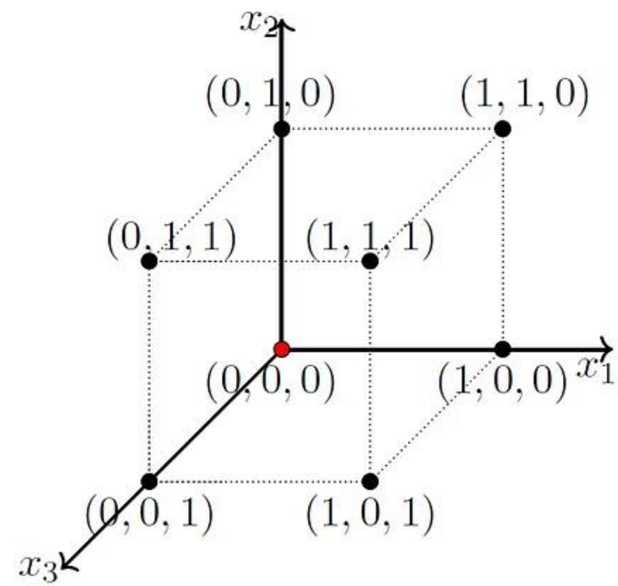


Geometric Interpretation of M-P Model: OR Gate (3-inputs/features)



$OR\ function$

$$x_1 + x_2 + x_3 = \sum_{i=1}^{3} x_i \geq 1$$



Limitations

- The model does not work for non-binary inputs. (Only used for binary inputs)
- The threshold had to be decided beforehand and needed manual computation instead of the model deciding itself.

Rosenblatt's Perceptron

- Overcoming the limitations of the M-P neuron, Frank Rosenblatt, an American psychologist, proposed the classical perception model, the mighty *artificial neuron*, in 1958.
- It is more generalized computational model than the McCulloch-Pitts neuron where weights and thresholds can be learnt over time.

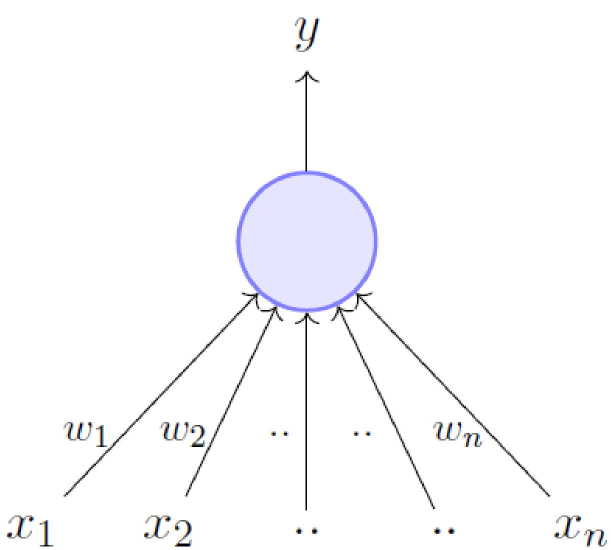
What is Perceptron?

- A perceptron is **a simple model of a biological neuron in an artificial neural network**.
- It consists of a single neuron with adjustable synaptic weights and bias.
- In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers.
- It is used for the classification of patterns said to be *linearly separable* (i.e., patterns that lie on opposite sides of a hyperplane).

- Definition: Sets of points in 2D space (R^2) are linearly separable, if the sets can be separated by a straight line.

- Set of points in n-dimensional space (R^n) are linearly separable, if there is a hyper plane of (n-1) dimensions separating the sets.

Perceptron without Bias



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

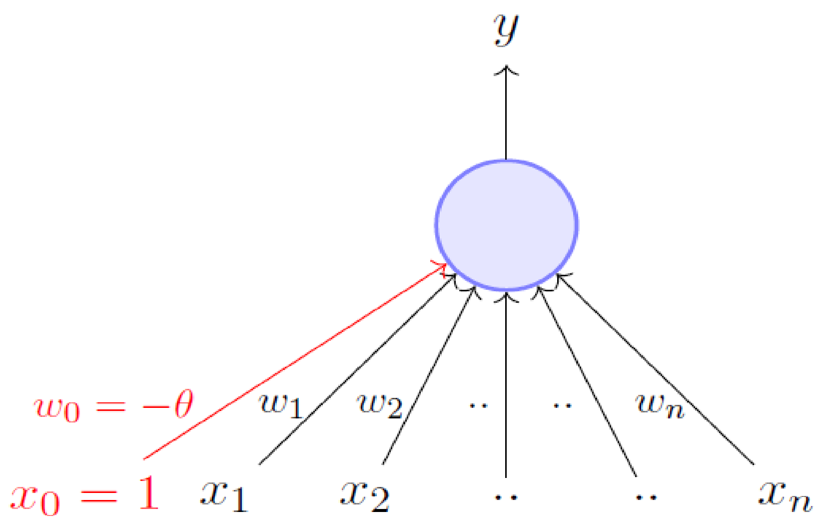
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

Perceptron with Bias



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

M-P vs. Perceptron

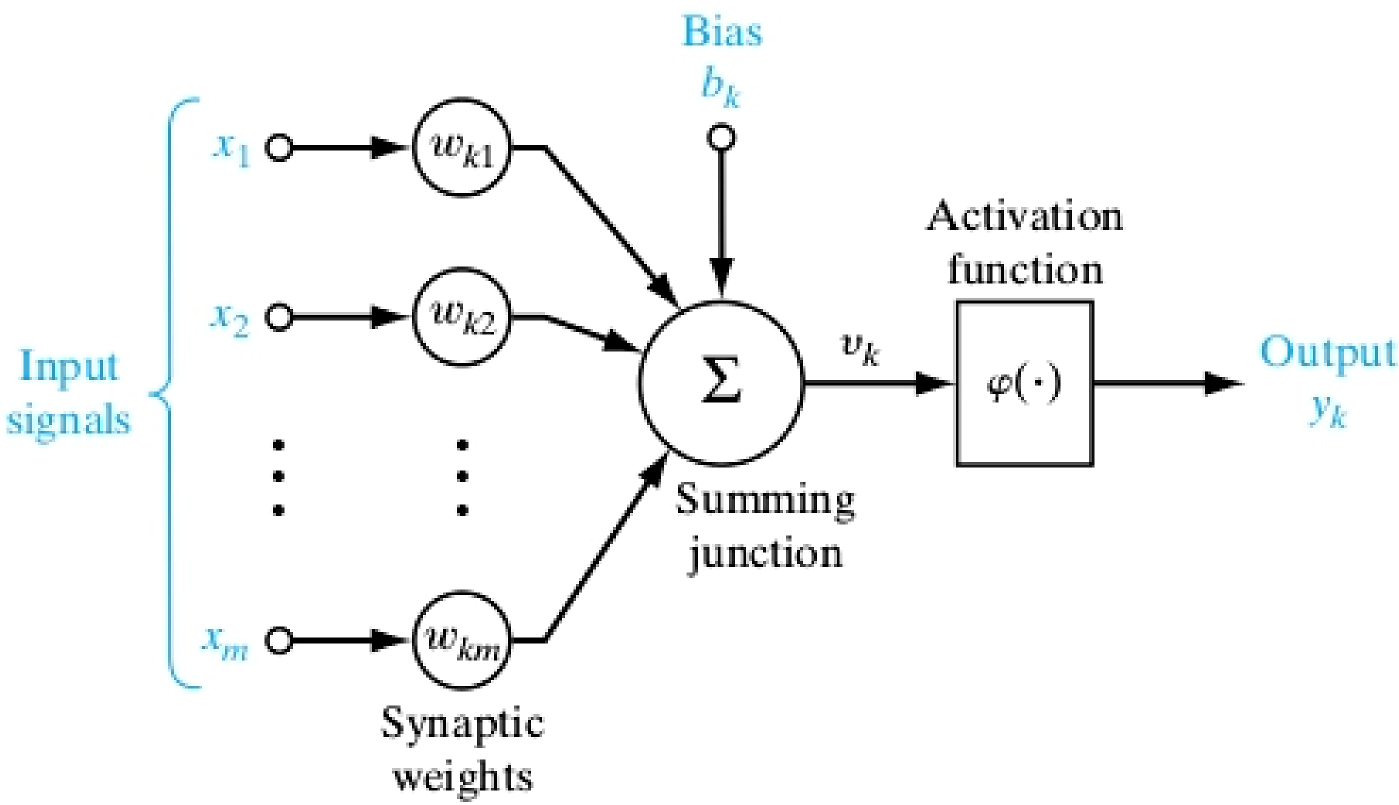
McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=0}^{n} x_i \geq 0 \\
&= 0 \quad if \sum_{i=0}^{n} x_i < 0
\end{aligned}
$$

Perceptron

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0 \\
&= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0
\end{aligned}
$$

Perceptron



Bias
b_k
x_1
x_2
x_m
w_{k1}
w_{k2}
w_{km}
Input signals
Σ
v_k
Activation function
$\varphi(\cdot)$
Output
y_k
Summing junction
Synaptic weights

Perceptron Learning

```
Algorithm 1 Perceptron Learning
w = [w0, w1, w2, . . . , wn]
x = [1, x1, x2, . . . , xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
   Pick random x ∈ P ∪ N
   if x ∈ P and wᵀx < 0 then
      w = w + x
   if x ∈ N and wᵀx ≥ 0 then
      w = w − x
end
```

Perceptron Learning

- Equation of line

$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^{\mathbf{T}}\mathbf{x} = \sum_{i=0}^{n} w_i * x_i$$

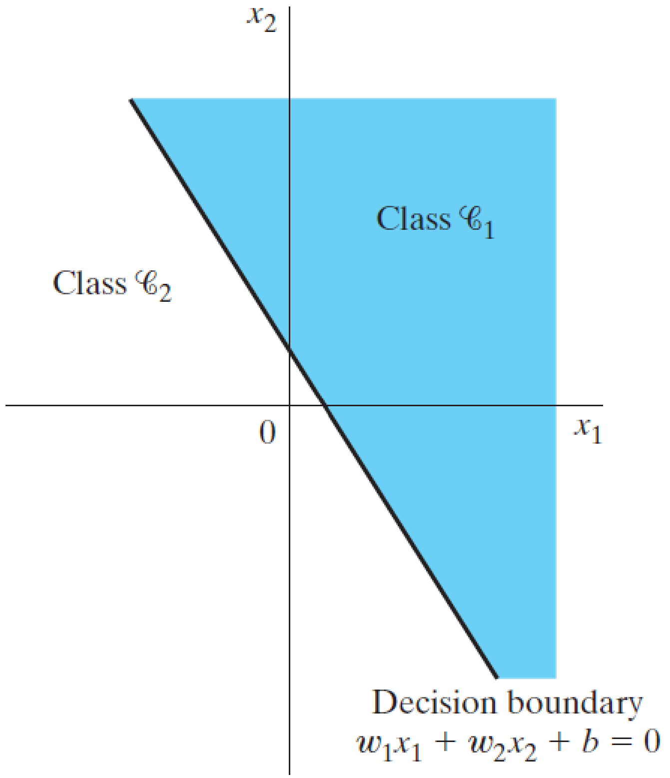
- We can rewrite the perceptron rule as

$$\begin{aligned} y &= 1 \quad if \quad \mathbf{w}^{\mathbf{T}}\mathbf{x} \geq 0 \\ &= 0 \quad if \quad \mathbf{w}^{\mathbf{T}}\mathbf{x} < 0 \end{aligned}$$

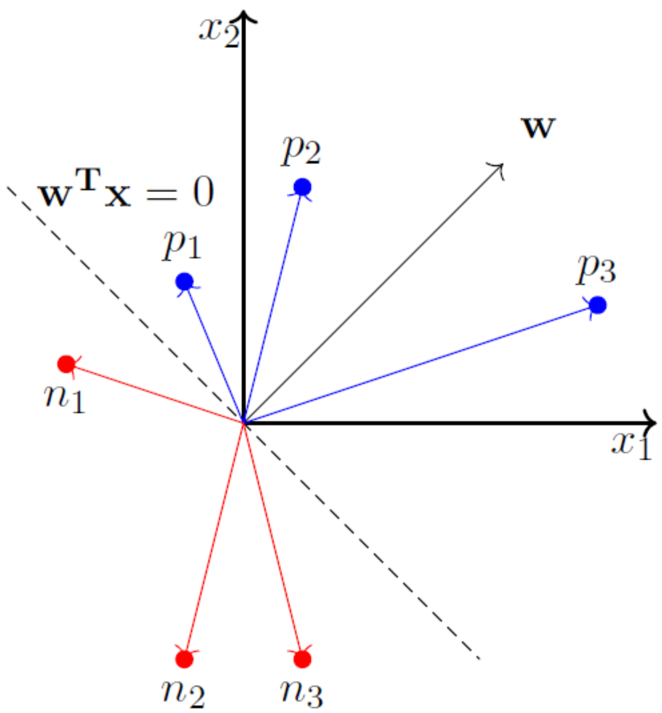
We are interested in finding the line $\mathbf{w}^{\mathbf{T}}\mathbf{x} = 0$ which divides the input space into two halves

Every point $(\mathbf{x})$ on this line satisfies the equation $\mathbf{w}^{\mathbf{T}}\mathbf{x} = 0$

2D Decision Boundary



Perceptron Learning



Negative-Half (N Class)

$w^T x < 0$

➔ $\alpha > 90^0$

Positive-Half (P Class)

$w^T x \geq 0$

➔ $\alpha < 90^0$

Changing

$w(new) = w(old) + x$

➔ $\alpha(new) < \alpha(old)$

Changing

$w(new) = w(old) - x$

➔ $\alpha(new) > \alpha(old)$

Perceptron Learning

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w.x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w.x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

---

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||}$$

- For $\mathbf{x} \in P$ if $\mathbf{w.x} < 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is greater than 90° (but we want α to be less than 90°)
- What happens to the new angle (α_{new}) when $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\ &\propto (\mathbf{w} + \mathbf{x})^T\mathbf{x} \\ &\propto \mathbf{w}^T\mathbf{x} + \mathbf{x}^T\mathbf{x} \\ &\propto cos\alpha + \mathbf{x}^T\mathbf{x} \\ cos(\alpha_{new}) &> cos\alpha \end{aligned}$$

- Thus α_{new} will be less than α and this is exactly what we want

Perceptron Learning

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||}$$

- For $\mathbf{x} \in N$ if $\mathbf{w}.\mathbf{x} \geq 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is less than 90° (but we want α to be greater than 90°)
- What happens to the new angle (α_{new}) when $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$

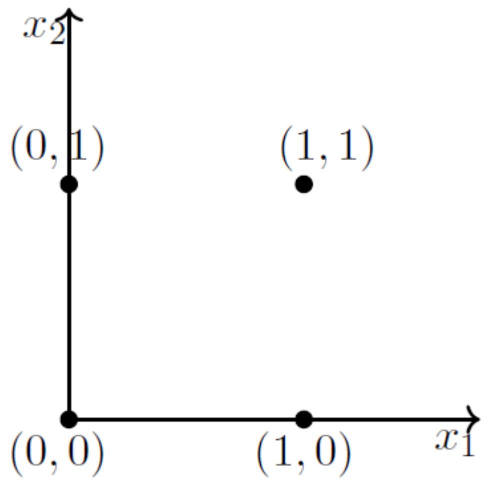
$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\ &\propto (\mathbf{w} - \mathbf{x})^T\mathbf{x} \\ &\propto \mathbf{w}^T\mathbf{x} - \mathbf{x}^T\mathbf{x} \\ &\propto cos\alpha - \mathbf{x}^T\mathbf{x} \\ cos(\alpha_{new}) &< cos\alpha \end{aligned}$$

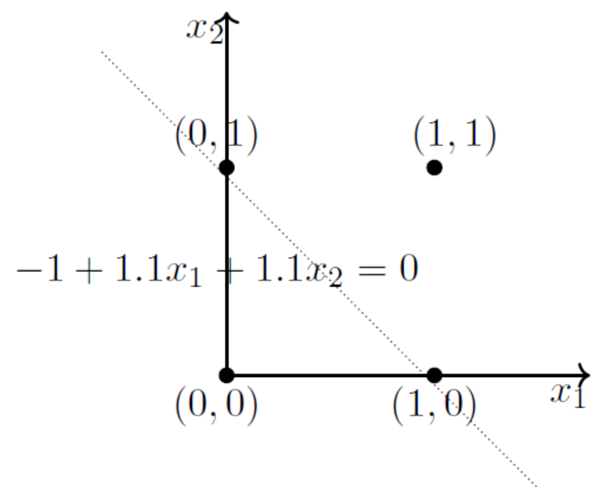
- Thus α_{new} will be greater than α and this is exactly what we want

OR Gate

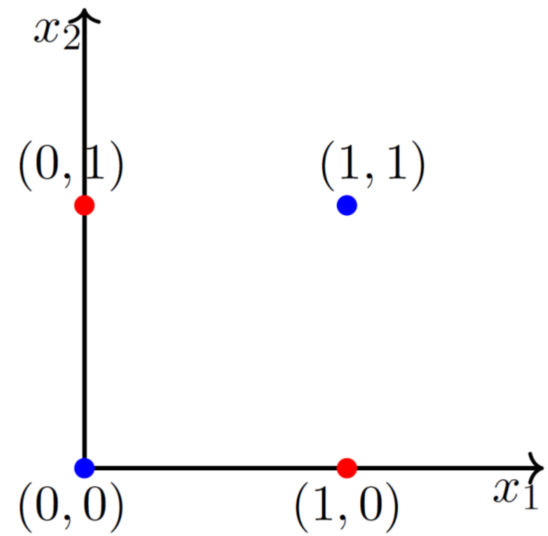
x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0$$





XOR



Perceptron Learning Rule

```
Algorithm 1 Perceptron Learning
w = [w0, w1, w2, . . . , wn]
x = [1, x1, x2, . . . , xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
   Pick random x ∈ P ∪ N
   if x ∈ P and wᵀx < 0 then
      w = w + x
   if x ∈ N and wᵀx ≥ 0 then
      w = w − x
end
```

Target (t)

x1	x2	OR	
0	0	0	N
0	1	1	P
1	0	1	P
1	1	1	P

If t=1 and computed output y = 0, then w(new) = w(old) + x

If t=0 and computed output y = 1, then w(new) = w(old) - x

If t == y, then w(new) = w(old)

Perceptron Learning Rule

Target (t)

x1	x2	OR	
0	0	0	N
0	1	1	P
1	0	1	P
1	1	1	P

The three rules above can be rewritten as a single expression.

Let the perceptron error:

$$e = t - y$$

If e=1, then
w(new) = w(old) + x

If e=-1, then
w(new) = w(old) - x

If e=0, then w(new) = w(old)

If t=1 and computed output y = 0, then w(new) = w(old) + x

If t=0 and computed output y = 1, then w(new) = w(old) - x

If t == y, then w(new) = w(old)

Perceptron Learning Rule

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

$if\ \ y \neq t,$ $\qquad w(new) = w(old) + \eta\, t\, x$

$else$ $\qquad w(new) = w(old)$

η : Learning rate or step size

t : target

We can also use d to represent target or desired variable.

Perceptron Convergence Theorem

- **For any finite set of linearly separable labelled examples, the Perceptron Learning Algorithm will halt after a finite number of iterations**.
- In other words, after a finite number of iterations, the algorithm yields a vector w that classifies perfectly all the examples.

Perceptron Convergence Algorithm

TABLE 1.1 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

$\mathbf{x}(n) = (m + 1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n) = (m + 1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as
$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$
where sgn($\cdot$) is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
where
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$
5. *Continuation.* Increment time step n by one and go back to step 2.