Perceptron Learning 1

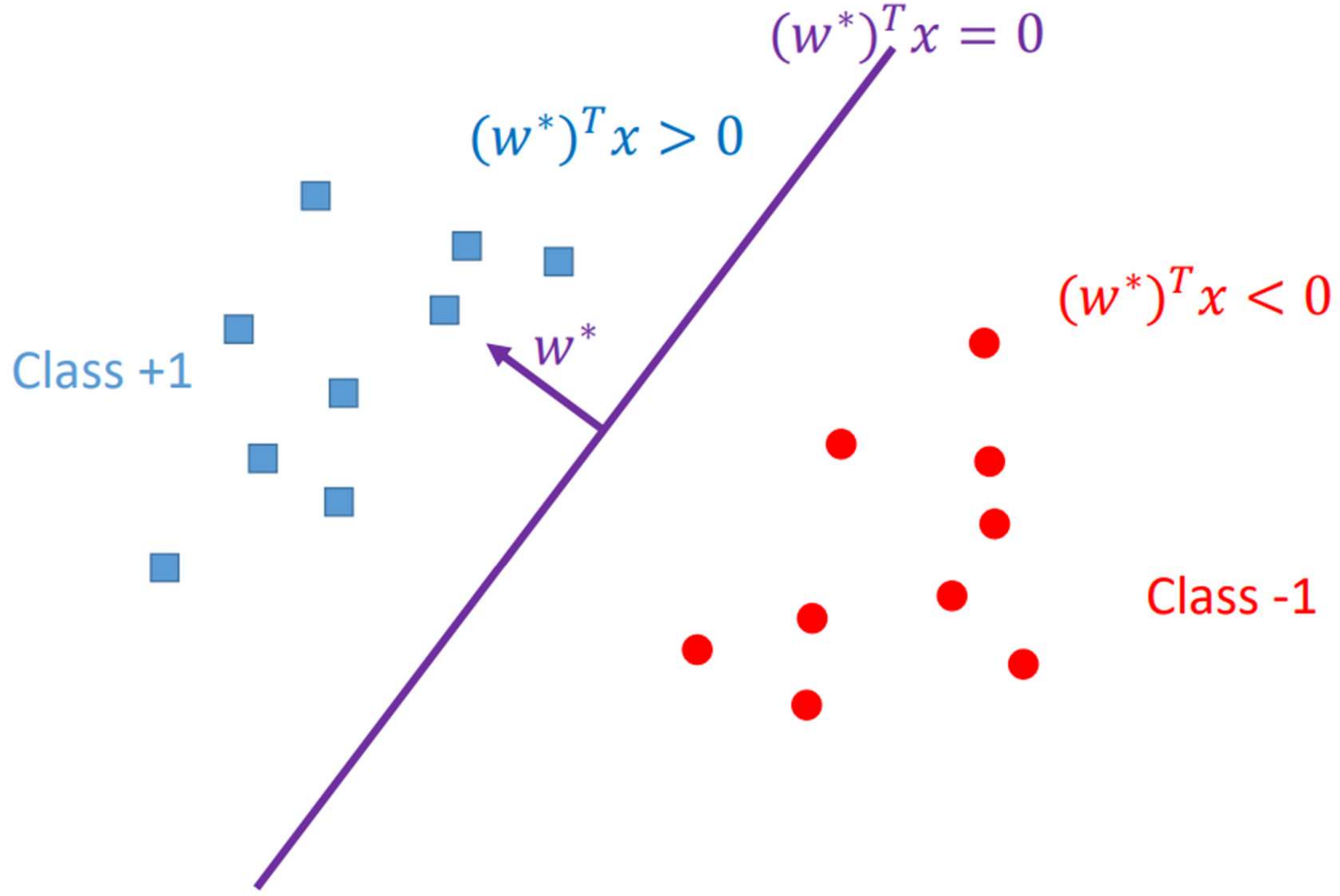
Last Lecture

- MP Neuron and Thresholding Logic

Today’s Topics

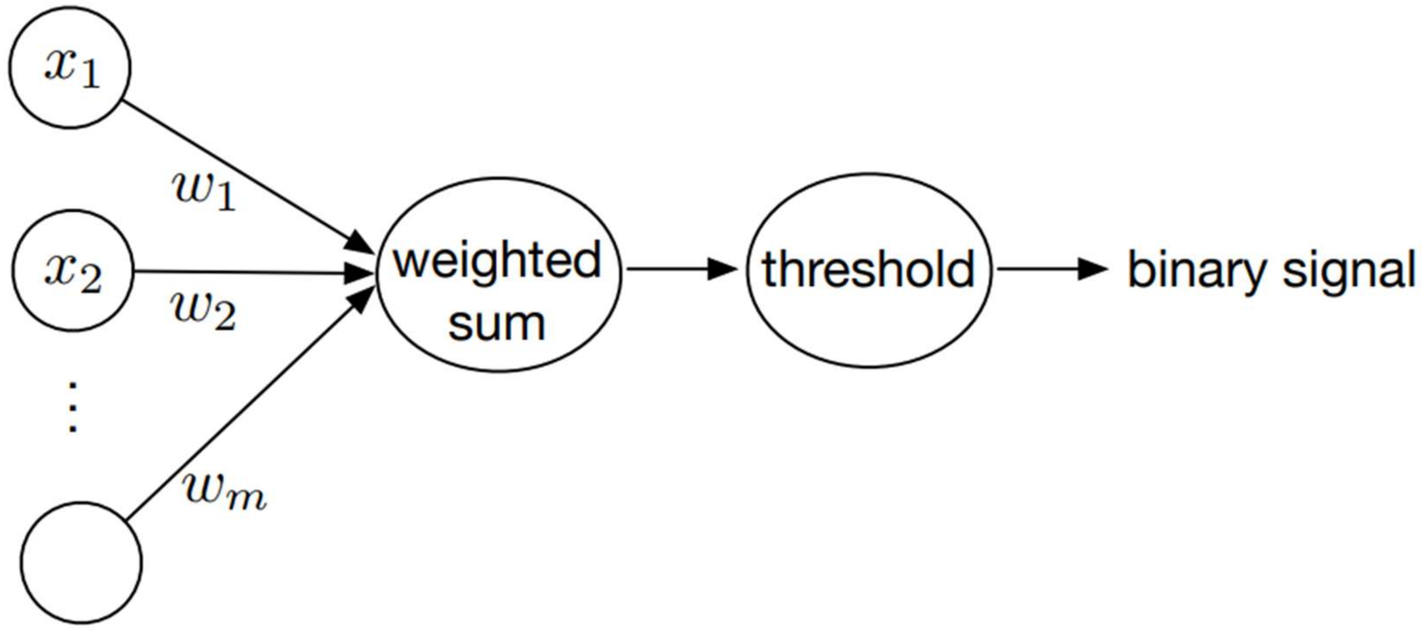
- Perceptron Model
- Mathematical Interpretation
- **Boolean Functions Using Perceptron**
- Learning Algorithm

Task



Problems with MP neuron

- Only models binary input
- Structure doesn't change
- Weights are set by hand
  - No learning!!
- But nonetheless is basis for all future work on neural nets



Perceptron

- What about non-boolean (say, real) inputs ?
- Do we always need to hand code the threshold ?
- Are all inputs equal ? What if we want to assign more weight (importance) to some inputs ?
- What about functions which are not linearly separable ?

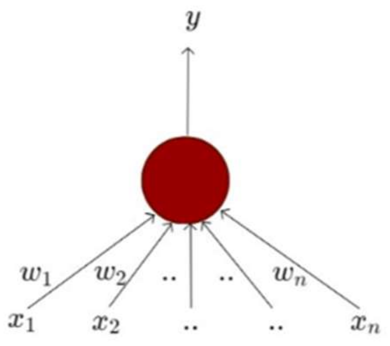
Rosenblatt's Perceptron

A learning rule for the computational/mathematical neuron model

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton. Project Para*. Cornell Aeronautical Laboratory.

Source: http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/Members/wilex4/Rosen-2.jpg

Perceptron Model



It overcomes some of the limitations of the M-P neuron by introducing

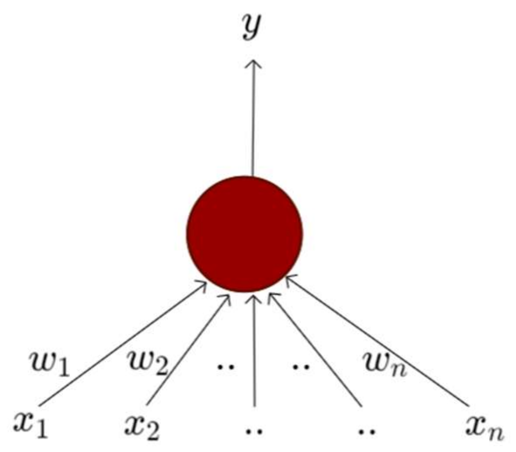
- the concept of numerical weights (a measure of importance) for inputs,
- a mechanism for learning those weights.
- Inputs are no longer limited to boolean values like in the case of an M-P neuron,
- it supports real inputs as well which makes it more useful and generalized.

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

Rewriting the above,

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

Perceptron



A more accepted convention,

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

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

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

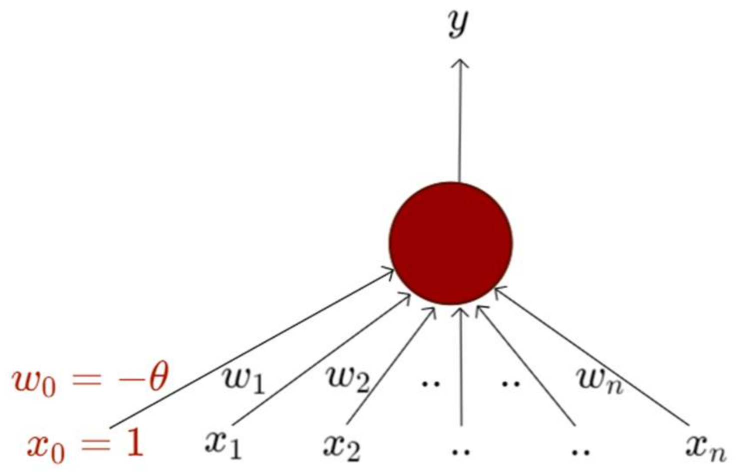
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

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



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

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

Rewriting the above,

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

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

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

Difference between MP and Perceptron

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

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

Perceptron

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

From the equations it should be clear that even a perceptron separates the input space into two halves

All inputs which produce a 1 lie on one side and all inputs which produce a 0 lie on the other side

In other words, a single perceptron can only be used to implement linearly separable functions

Then what is the difference?

The weights (including threshold) can be learned and the inputs can be real valued

We will first revisit some boolean functions and then see the perceptron learning algorithm (for learning weights)

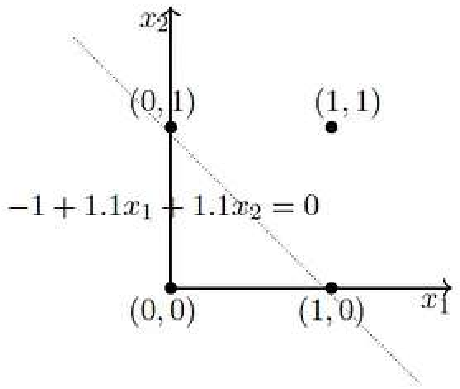
Boolean Functions Using Perceptron

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

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

One possible solution is

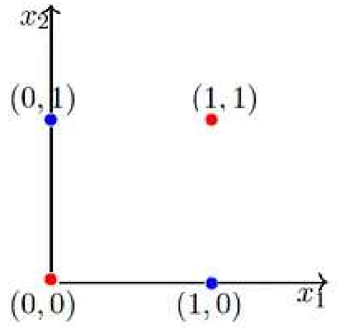
$$w_0 = -1,\ w_1 = 1.1,\ w_2 = 1.1$$



Boolean Functions Using Perceptron

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

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



Notice that the fourth equation contradicts the second and the third equation. Point is, there are no *perceptron* solutions for non-linearly separated data. So the key take away is that a **single** *perceptron* cannot learn to separate the data that are non-linear in nature.

Components of the Perceptron

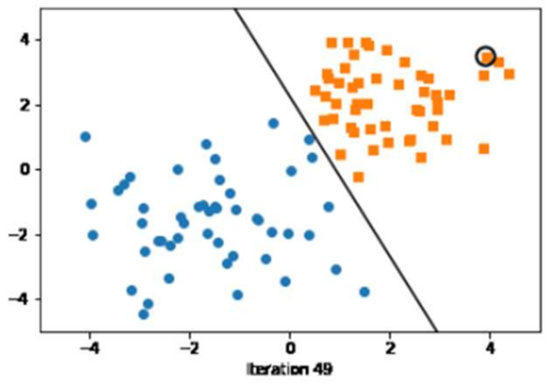
- **Input (x):** These are the feature values of a data sample, typically represented as a vector.
- **Weights (w):** Each feature has an associated weight that determines its influence on the output.
- **Bias (b):** A constant term added to the weighted sum to shift the decision boundary.
- **Activation Function:** Usually a step function that produces the final binary output (1 or 0).

Perceptron Learning Algorithm

1. **Initialize the Weights and Bias:** Start with random small values for the weights and the bias term.
2. **For Each Training Example:**
   - **Compute the Weighted Sum:**
   - **Apply the Activation Function:** The perceptron uses a step function that outputs 1 if the weighted sum is greater than or equal to 0 and 0 otherwise:
   - **Update the Weights and Bias:**
     - If the predicted output matches the true label, no changes are made.
     - If ypred differs from y, adjust the weights and bias using the update rules
3. **Repeat:** Iterate through the training data for multiple epochs until the perceptron converges (i.e., there are no more misclassifications) or a predefined number of epochs is reached.

The Perceptron Learning Algorithm

- If correct: Do nothing if the prediction if output is equal to the target
- If incorrect, scenario **a)**:
  If output is 0 and target is 1, add input vector to weight vector
- If incorrect, scenario **b)**:
  If output is 1 and target is 0, subtract input vector from weight vector



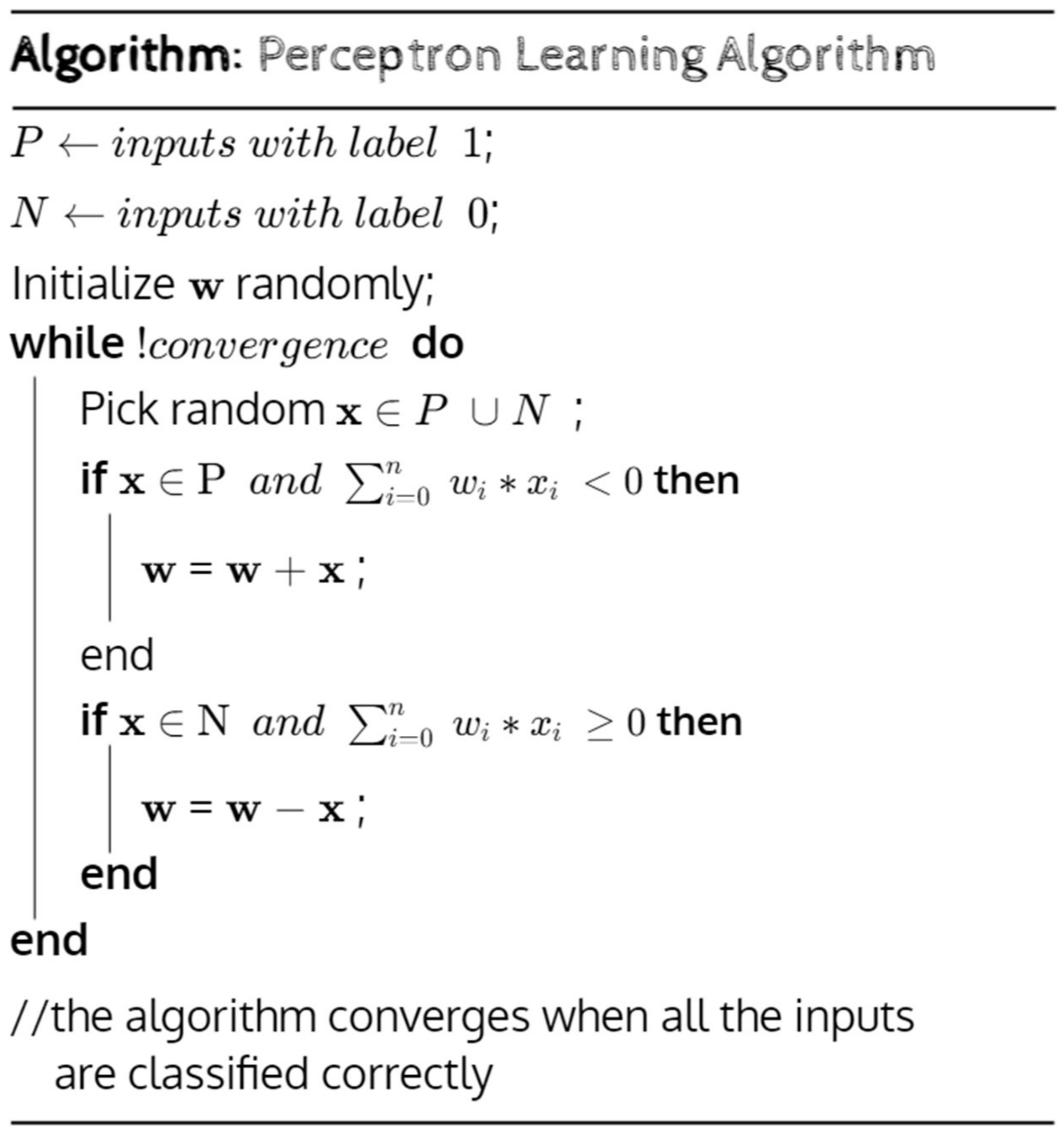
Guaranteed to converge if a solution exists
(more about that later...)

Perceptron Learning Algorithm

```
Algorithm: Perceptron Learning Algorithm
$P \leftarrow inputs\ with\ label\ 1$;
$N \leftarrow inputs\ with\ label\ 0$;
Initialize $\mathbf{w}$ randomly;
while !$convergence$ do

end
```

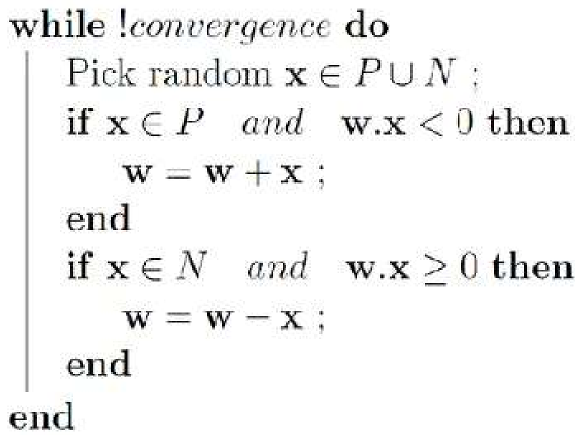
P m



Algorithm: Perceptron Learning Algorithm

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

Why add/subtract input vector?



```
while !$convergence$ do
    Pick random $\mathbf{x} \in P \cup N$ ;
    if $\mathbf{x} \in P$   $and$   $\mathbf{w.x} < 0$ then
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$ ;
    end
    if $\mathbf{x} \in N$   $and$   $\mathbf{w.x} \geq 0$ then
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$ ;
    end
end
```

Case 1: When **x** belongs to *P* and its dot product **w.x** < 0
Case 2: When **x** belongs to *N* and its dot product **w.x** ≥ 0

Vector Representations

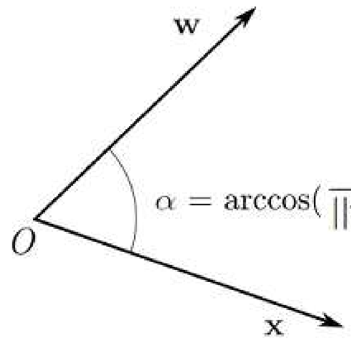
Dot Product Of Two Vectors

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_n]$$
$$\mathbf{x} = [1, x_1, x_2, ..., x_n]$$
$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w^T x} = \sum_{i=0}^{n} w_i * x_i$$

Angle Between Two Vectors

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

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



$$\alpha = \arccos(\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||})$$

Dot product

The dot product of two vectors is written as $\mathbf{m}^T\mathbf{x}$ or $\mathbf{m} \cdot \mathbf{x}$, which is defined as: $\mathbf{m}^T\mathbf{x} = \sum_{i=1}^{k} m_i x_i$

Example

$$\mathbf{m} = < 5.13, 1.08, -0.03, 7.29 >$$
$$\mathbf{x} = < x_1, x_2, x_3, x_4 >$$
$$\mathbf{m}^T\mathbf{x} = 5.13x_1 + 1.08x_2 - 0.03x_3 + 7.29x_4$$

If dot product of two vectors is zero: means the two vectors are perpendicular (90° angle)

Length or norm of a vector

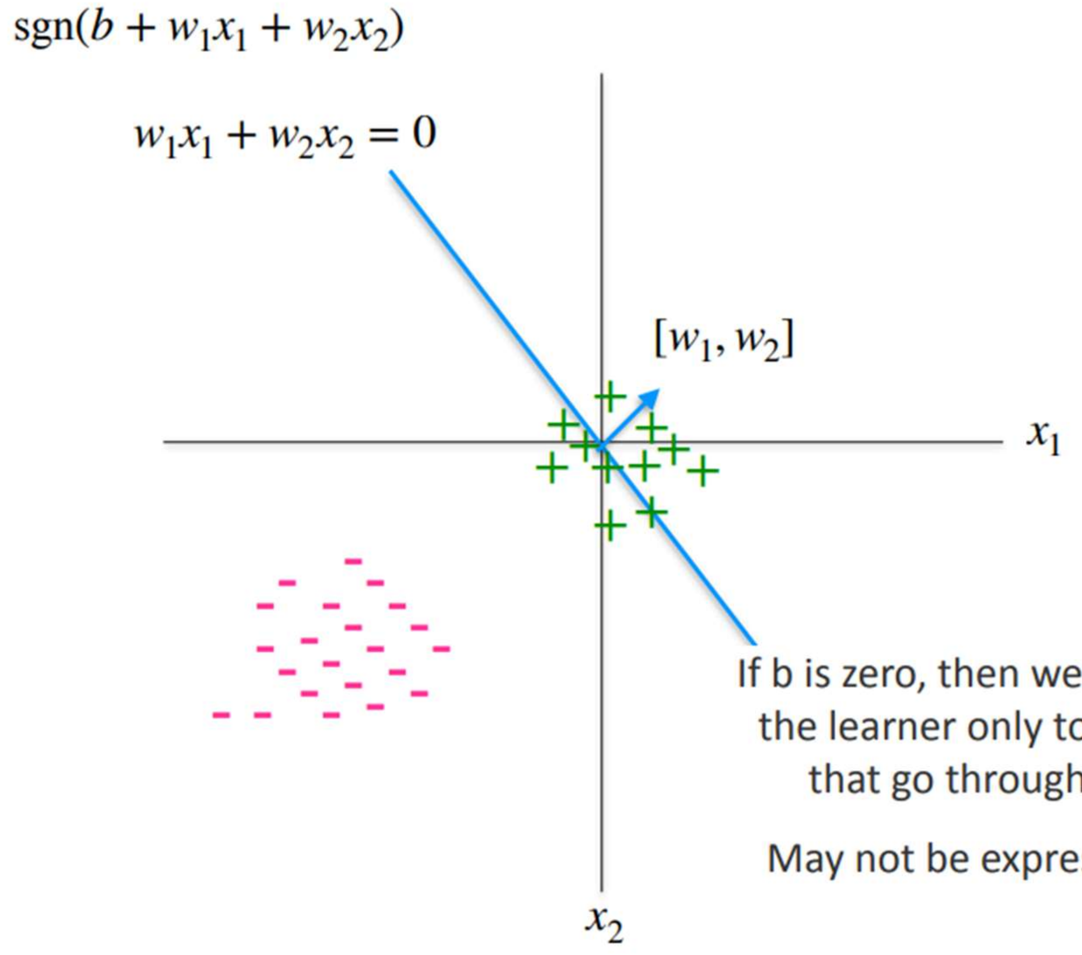
The length or norm of a vector $\mathbf{v}$ is the square root of length $= ||\mathbf{v}|| = \sqrt{\mathbf{v} \cdot \mathbf{v}}$

Dot product here is not zero ($\mathbf{v}$ is not perpendicular to itself), so now have $0°$ angle: product of $\mathbf{v} \cdot \mathbf{v}$ gives length of $\mathbf{v}$ squared

In 2 dimensions: length $= ||\mathbf{v}|| = \sqrt{v_1^2 + v_2^2}$

Why is the bias term needed?

An illustration in two dimensions



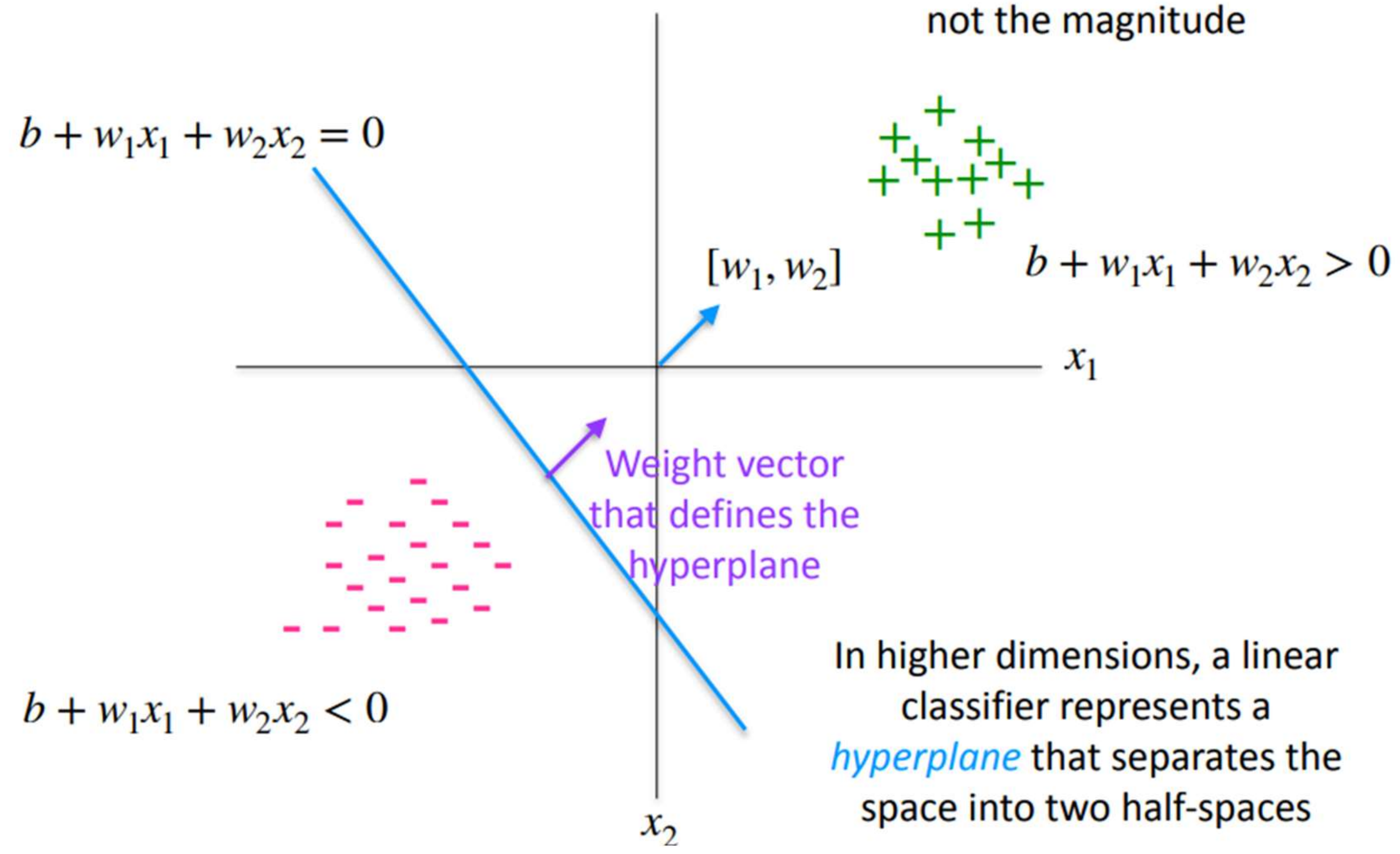
If b is zero, then we are restricting the learner only to hyperplanes that go through the origin

May not be expressive enough

The geometry of a linear classifier

$\mathrm{sgn}(b + w_1x_1 + w_2x_2)$

We only care about the sign, not the magnitude


$b + w_1x_1 + w_2x_2 = 0$
$[w_1, w_2]$
$b + w_1x_1 + w_2x_2 > 0$
x_1
Weight vector that defines the hyperplane
$b + w_1x_1 + w_2x_2 < 0$
x_2

In higher dimensions, a linear classifier represents a *hyperplane* that separates the space into two half-spaces

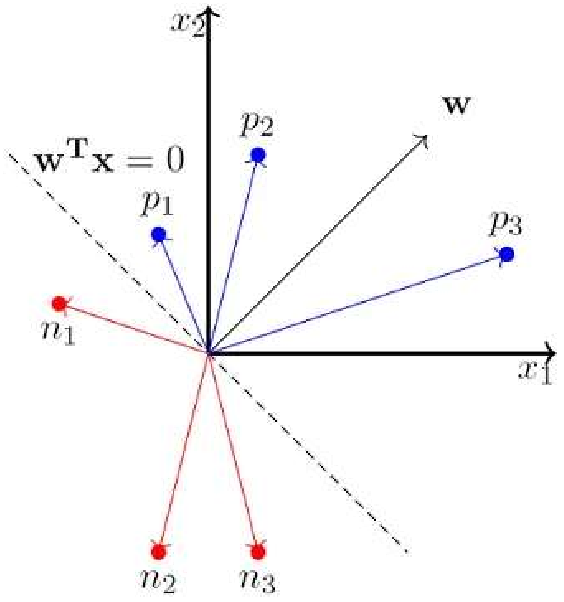
Why Would The Specified Update Rule Work?

when **x** belongs to *P*, the angle between **w** and **x** should be less than 90 because the cosine of the angle is proportional to the dot product.

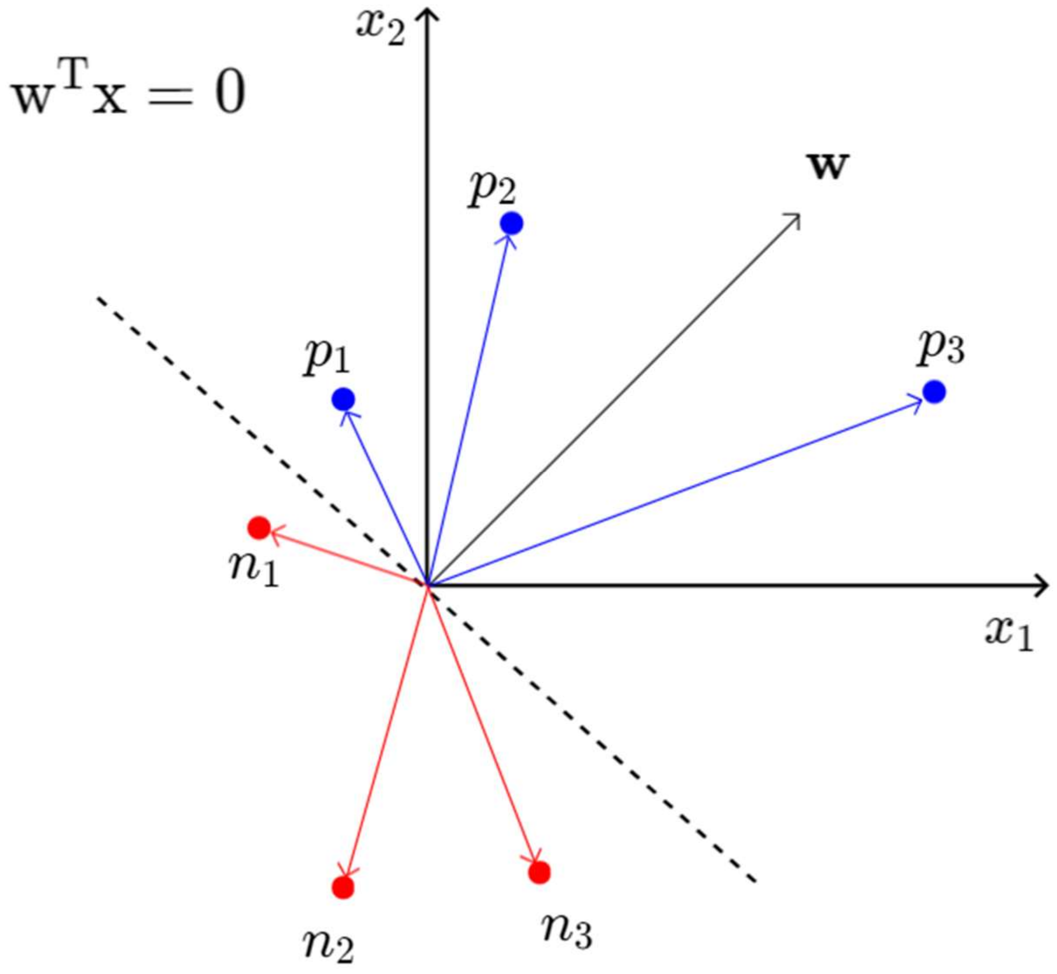
$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||} \quad \Big| \quad cos\alpha \propto \mathbf{w}^T\mathbf{x}$$

$$\text{So if } \mathbf{w}^T\mathbf{x} > 0 \quad \Rightarrow cos\alpha > 0 \quad \Rightarrow \alpha < 90$$

$$\text{Similarly, if } \mathbf{w}^T\mathbf{x} < 0 \quad \Rightarrow cos\alpha < 0 \quad \Rightarrow \alpha > 90$$



Why Would The Specified Update Rule Work?



What will be the angle between p vector and w – less than 90

What will be the angle between n vector and w – greater than 90

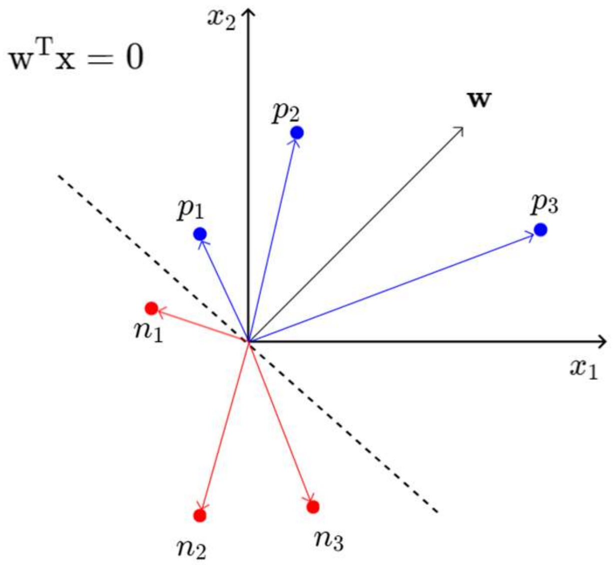
Why Would The Specified Update Rule Work?

Case 1 x belongs to P and **w.x** < 0 , alpha > 90

we are adding **x** to **w - increasing the *cos(alpha)*** value, which means, we are **decreasing the *alpha* value**, the angle between **w** and **x**

Case 2 x belongs to N and **w.x** > 0 , alpha < 90

we are subtracting **x** from **w - decreasing the *cos(alpha)*** value, which means, we are in**creasing the *alpha* value**, the angle between **w** and **x**



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

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

Algorithm: Perceptron Learning Algorithm

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

For $\mathbf{x} \in P$ if $\mathrm{w}^{\mathrm{T}}\mathrm{x} < 0$ then it means that the angle (α) between this x and the current w is greater than 90°

(but we want to be less than 90°)

What happens to the new angle (α_{new}) when $\mathrm{w}_{\mathrm{new}} = \mathrm{w} + \mathrm{x}$

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

$$cos(\alpha_{new}) > cos\alpha \qquad \left(cos\alpha = \frac{w^T x}{\|w\|\|x\|}\right)$$

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

Algorithm: Perceptron Learning Algorithm

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

For $\mathbf{x} \in N$ if $\mathrm{w}^{\mathrm{T}}\mathrm{x} \geq 0$ then it means that the angle (α) between this x and the current w is less than 90° (but we want to be greater than 90°)

What happens to the new angle (α_{new}) when $\mathrm{w}_{\mathrm{new}} = \mathrm{w} - \mathrm{x}$

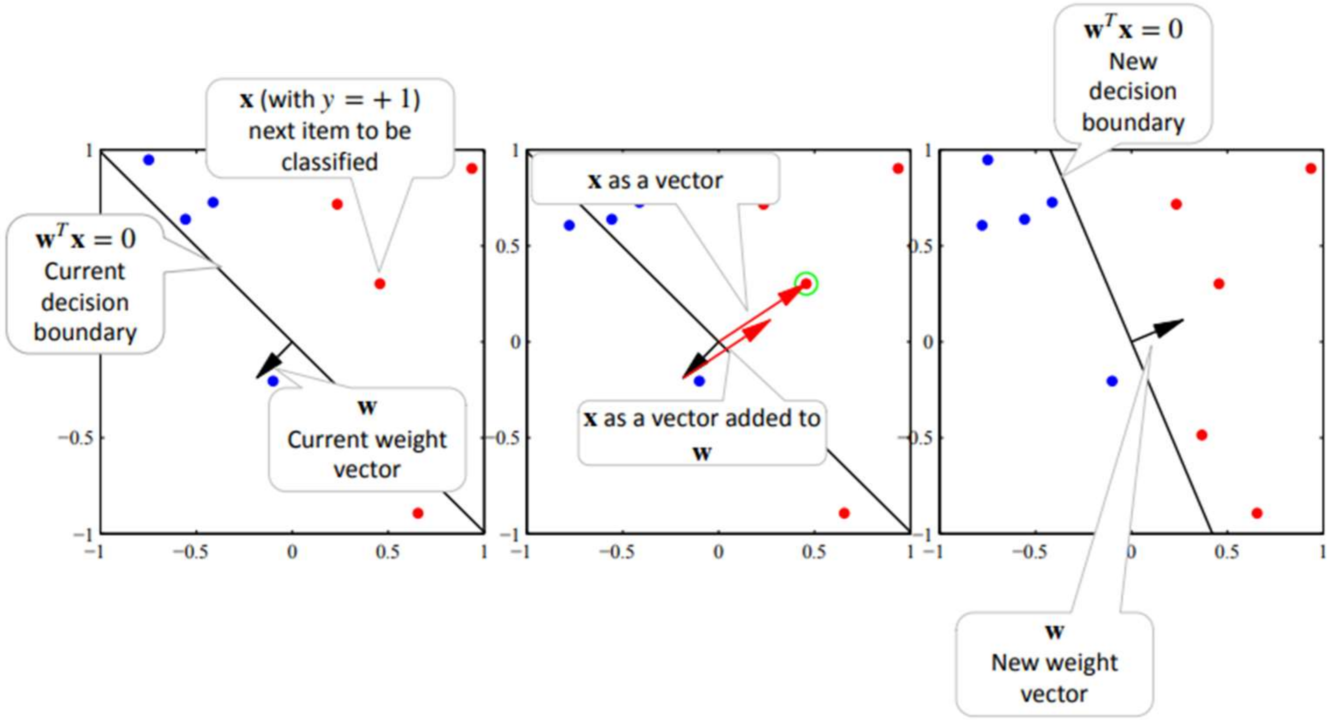
$$\begin{aligned} cos(\alpha_{new}) &\propto (\mathrm{w}_{\mathrm{new}})^{\mathrm{T}} + \mathrm{x} \\ &\propto (\mathrm{w} - \mathrm{x})^{\mathrm{T}}\mathrm{x} \\ &\propto \mathrm{w}^{\mathrm{T}}\mathrm{x} - \mathrm{x}^{\mathrm{T}}\mathrm{x} \\ &\propto cos\alpha - \mathrm{x}^{\mathrm{T}}\mathrm{x} \end{aligned}$$

$$cos(\alpha_{new}) < cos\alpha \qquad (cos\alpha = \frac{w^T x}{||w|| ||x||})$$

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

Perceptron in action

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$



(Figures from Bishop 2006)

● Positive
● Negative

Perceptron Convergence

Perceptron Convergence Theorem: If there exist a set of weights that are consistent with the data (i.e., the data is linearly separable), the perceptron learning algorithm will converge. Further, the number of times the perceptron must adjust weights before convergence is upper bounded

Perceptron Cycling Theorem: If the training data is not linearly separable the perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop.