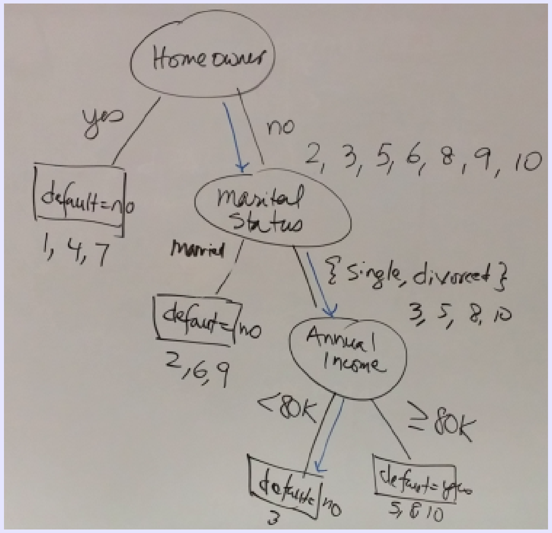
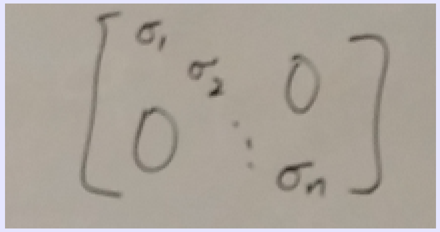
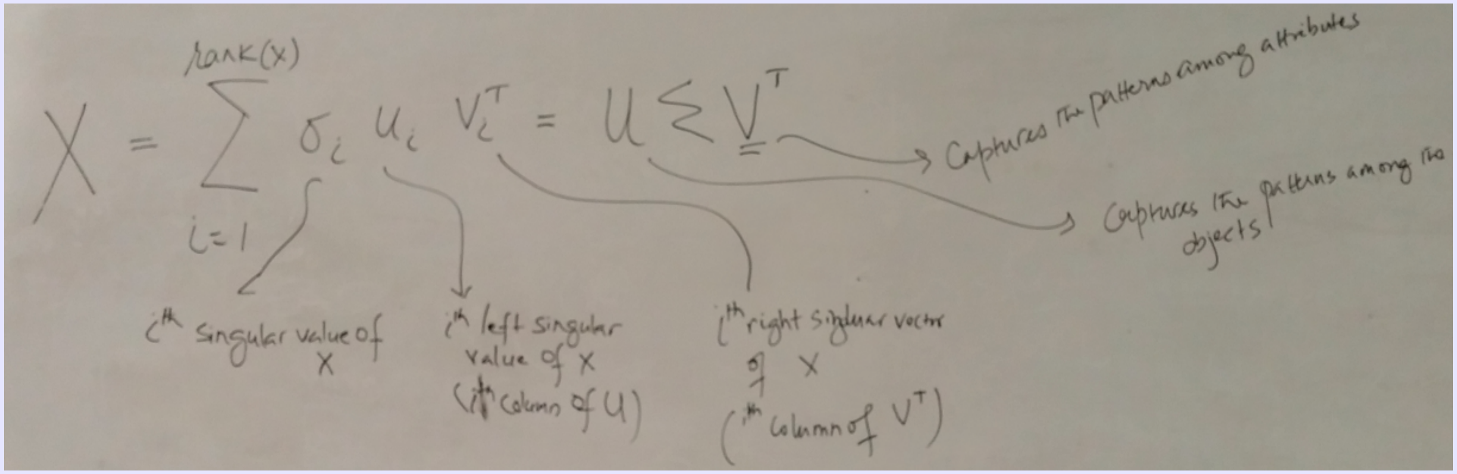
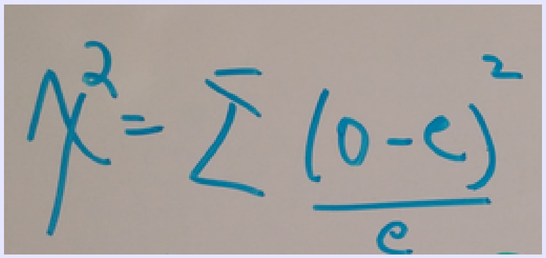
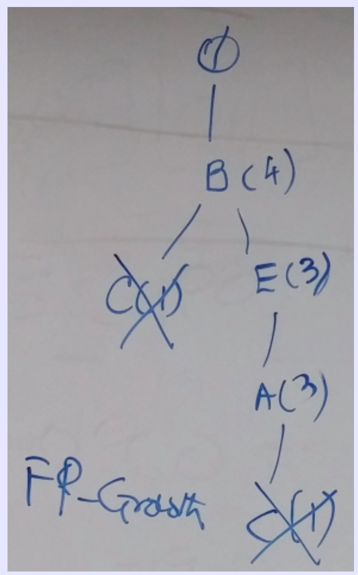
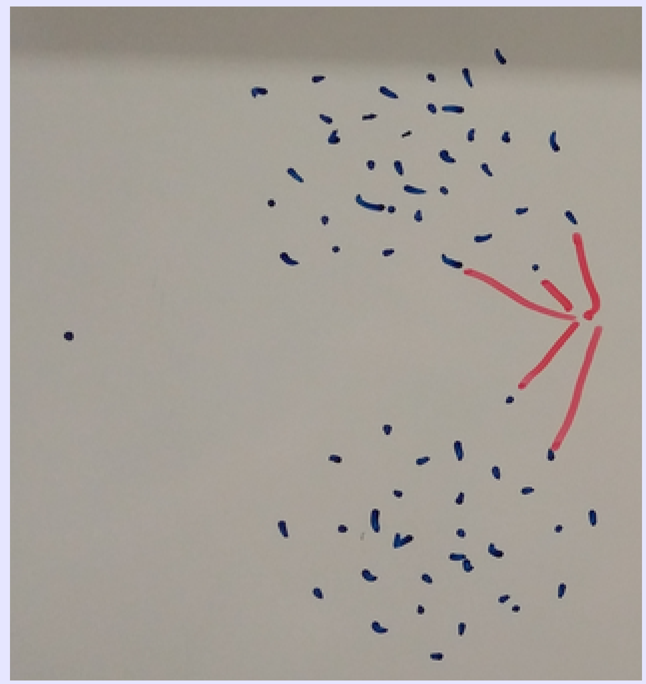












CS 422: Data Mining
Vijay K. Gurbani, Ph.D.,
Illinois Institute of Technology

Lecture: **The Perceptron**

Introduction

- … Except, of course, the human brain is in a league of its own.

Organism	Number of neurons
Jellyfish	5,600
Fruit fly	250,000
Frog	16,000,000
Cat	760,000,000
Humans	100,000,000,000

Introduction

- … Except, of course, the human brain is in a league of its own.

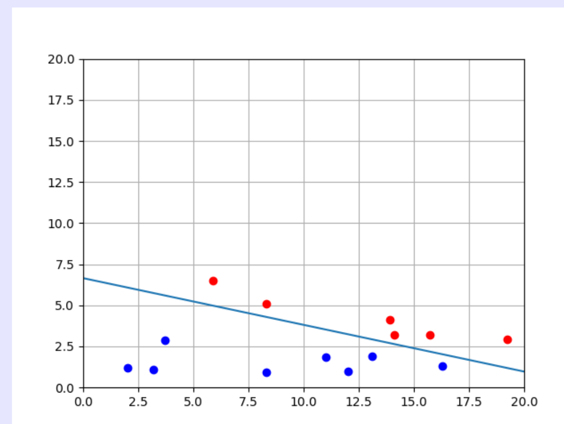
Organism	Number of neurons
Jellyfish	5,600
Fruit fly	250,000
Frog	16,000,000
Cat	760,000,000
Humans	100,000,000,000

- Currently (ca 2018), the largest artificial neural network, built on supercomputers, is about as smart as a frog (16,000,000 neurons).
  (Source: https://phys.org/news/2018-06-ai-method-power-artificial-neural.html)

Perceptron

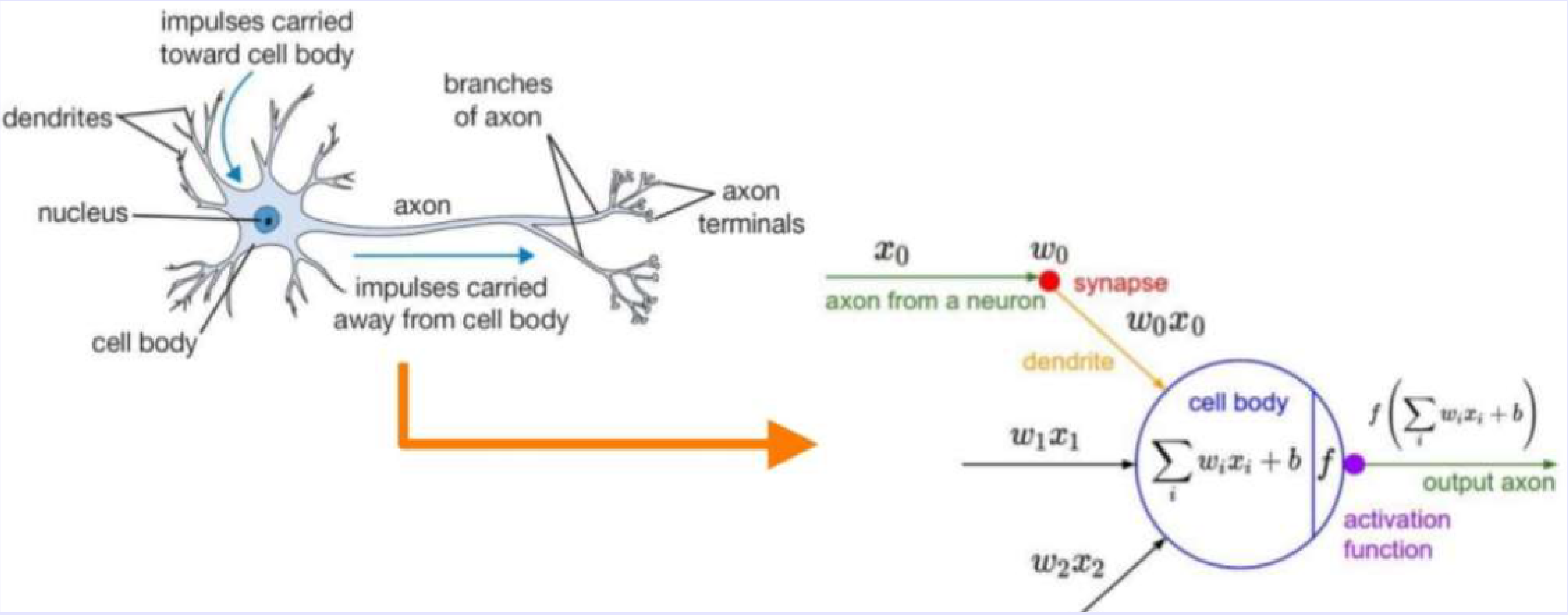
- The perceptron was the first learning algorithm based on the concept of a neural network applied to CS.

- Investigated in 1957 by Frank Rosenblatt at Cornell, funded by the US Office of NavalResearch.
- A perceptron was guaranteed to find the linear boundary between two classes, if such a boundary existed.



- Very popular till 1969, when Minsky Papert showed that a perceptron could not learn an XOR function.
- Interest waned, leading to the first AI winter.

Perceptron: The details



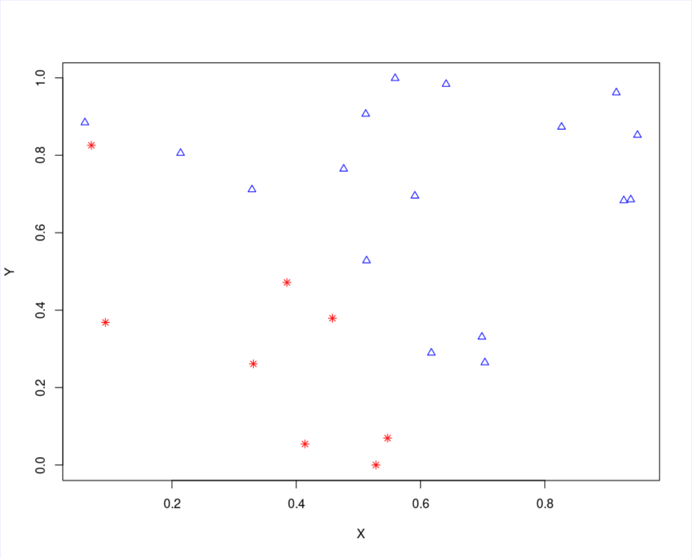
- Perceptron is a simple “one-cell” neural network. Forming more complex artificial neural networks allow us to solve a data mining or a learning task more effectively by neurons collectively working together on the task.

Perceptron: The details

- The most simplest type of neural network.
- Characterized as a “feed-forward” neural network that can be used to solve linearly separable problems.
  - **Strong guarantee**: If the data is linearly separable, a perceptron will find the hyperplane that separates it.
- An online algorithm, processes one observation at a time.
  - Several other variants exist.

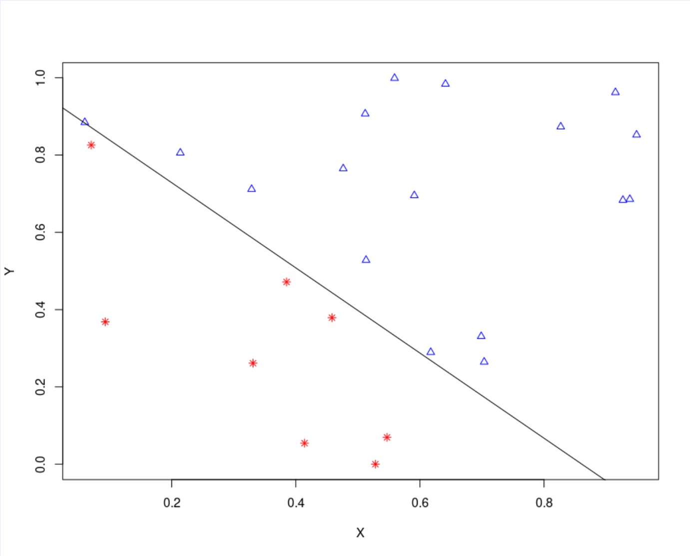
Perceptron: The details

Consider the problem of linearly separating two classes using a hyperplane.



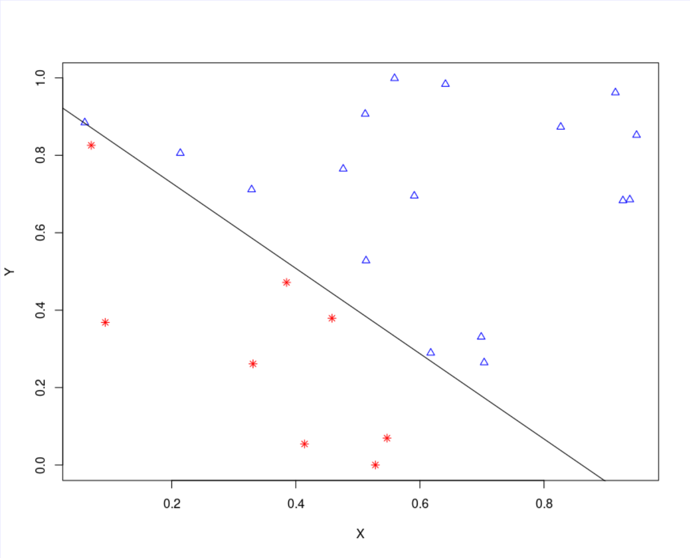
Perceptron: The details

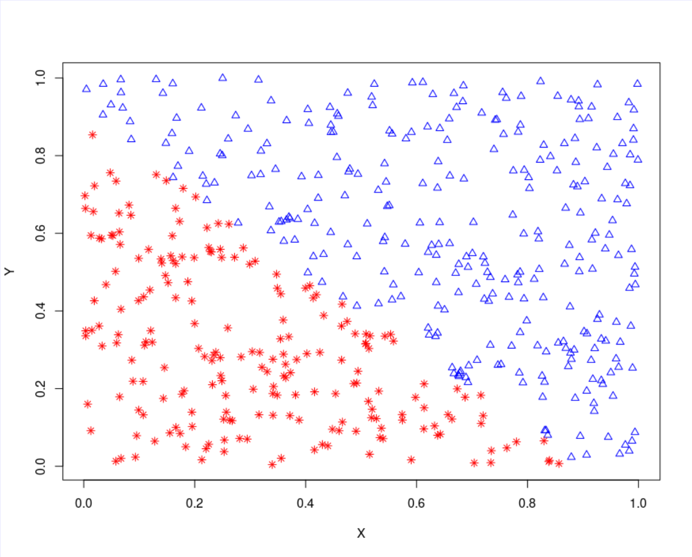
Consider the problem of linearly separating two classes using a hyperplane.



Perceptron: The details

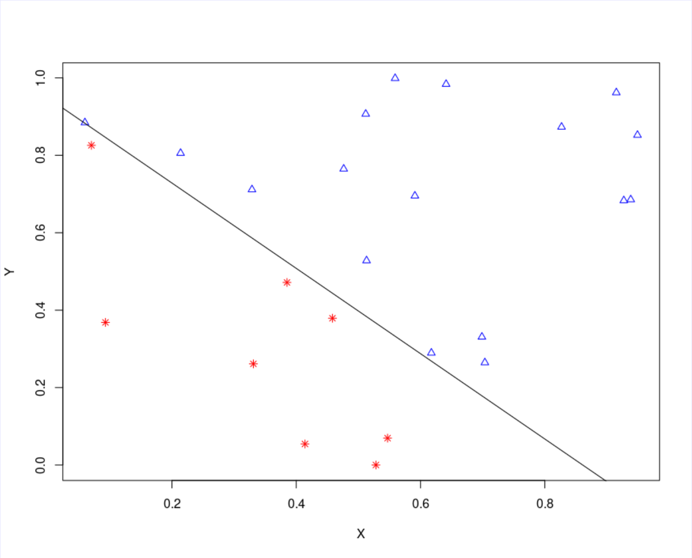
Consider the problem of linearly separating two classes using a hyperplane.

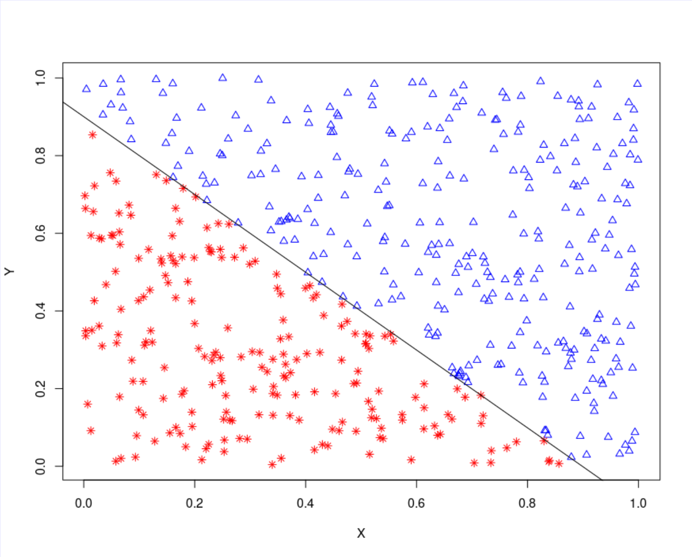




Perceptron: The details

Consider the problem of linearly separating two classes using a hyperplane.
Question:
    Can we train a model to derive a hyperplane that will separate the classes?





Perceptron: The details

- Answer: Yes, but ...
- … under the assumptions:
  - (1) We are dealing with a binary classification problem, i.e., $y_i \in \{-1, +1\}$, and
  - (2) Data is linearly separable.

Perceptron: The demo

- Demonstration of the perceptron.

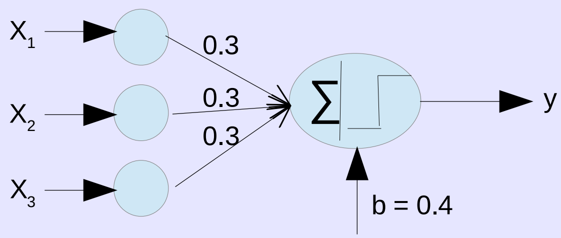
  perceptron.r

Perceptron: The details

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

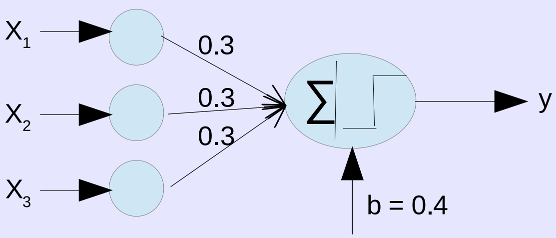
Perceptron: The details

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Perceptron: The details

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

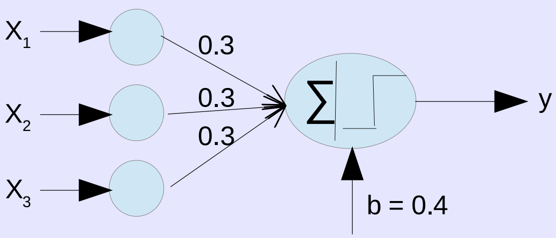


So how does a perceptron work?

$$\hat{y} = \begin{cases} 1: & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 > 0 \\ -1: & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 < 0 \end{cases}$$

Perceptron: The details

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

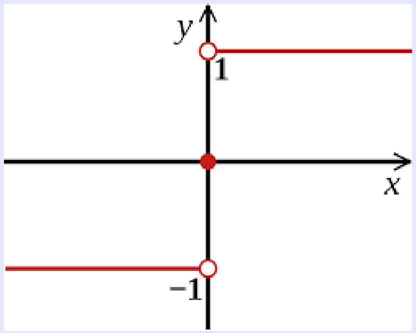


So how does a perceptron work?

$$\hat{y} = \begin{cases} 1 : & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 > 0 \\ -1 : & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 < 0 \end{cases}$$

$$\hat{y} = \text{sign}\left(\left(\sum_{i=i}^{n} w_i x_i\right) + b\right)$$

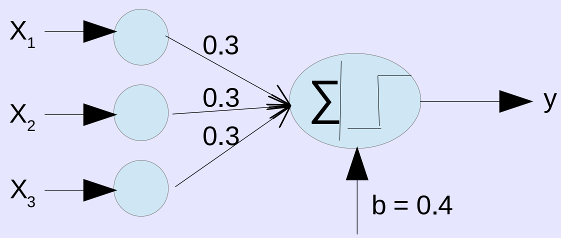
$$\text{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$



Source: https://en.wikipedia.org/wiki/Sign_function

Perceptron: The details

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



So how does a perceptron work?

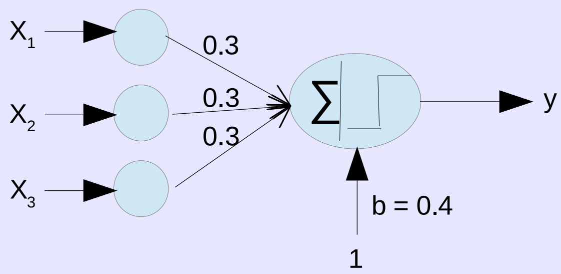
$$\hat{y} = \begin{cases} 1: & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 > 0 \\ -1: & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 < 0 \end{cases}$$

$$\hat{y} = \text{sign}\left(\left(\sum_{i=i}^{n} w_i x_i\right) + b\right)$$

But note that carrying this bias term is inconvenient
What does the bias represent?
Can we generalize it?

Perceptron: The details

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



So how does a perceptron work?

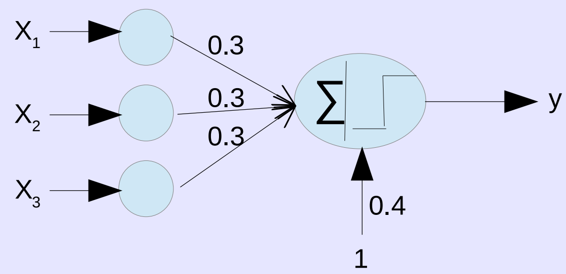
$$\hat{y} = \begin{cases} 1: & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 > 0 \\ -1: & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 < 0 \end{cases}$$

$$\hat{y} = \text{sign}\left(\left(\sum_{i=i}^{n} w_i x_i\right) + b\right)$$

$$\hat{y} = \text{sign}\left(\sum_{i=0}^{n} w_i * x_i\right)$$

Perceptron: The details

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



So how does a perceptron work?

$$\hat{y} = \begin{cases} 1 : & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 > 0 \\ -1 : & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 < 0 \end{cases}$$

$$\hat{y} = \operatorname{sign}\left(\left(\sum_{i=i}^{n} w_i x_i\right) + b\right)$$

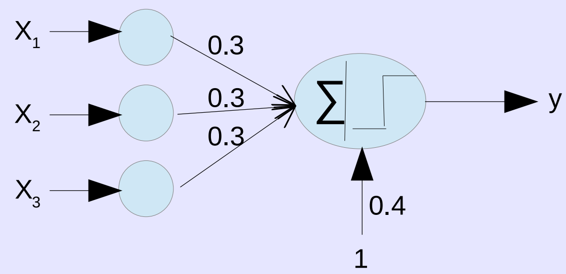
$$\hat{y} = \operatorname{sign}\left(\sum_{i=0}^{n} w_i * x_i\right)$$

$\hat{y} = \operatorname{sign}(w \cdot x)$, where w_0 is the bias and x_0 is 1

$$\vec{x} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \vec{w} = \begin{bmatrix} 0.4 \\ 0.3 \\ 0.3 \\ 0.3 \end{bmatrix}$$

Perceptron: The details

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



So how does a perceptron work?

$$\hat{y} = \begin{cases} 1: & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 > 0 \\ -1: & 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.4 < 0 \end{cases}$$

$$\hat{y} = \text{sign}\left(\left(\sum_{i=i}^{n} w_i x_i\right) + b\right)$$

$$\hat{y} = \text{sign}\left(\sum_{i=0}^{n} w_i * x_i\right)$$

$\hat{y} = \text{sign}(w \cdot x)$, where w_0 is the bias and x_0 is 1

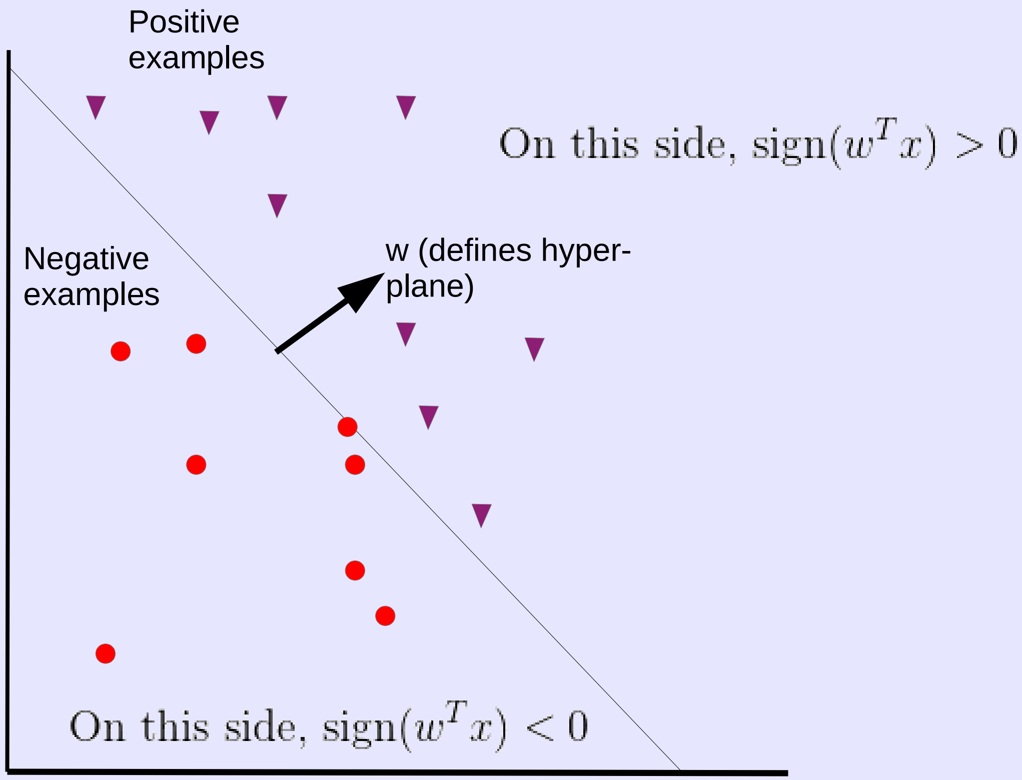
$$\vec{x} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \vec{w} = \begin{bmatrix} 0.4 \\ 0.3 \\ 0.3 \\ 0.3 \end{bmatrix}$$

Note that $w^T x = \sum_{i=0}^{n} w_i x_i$, so we can rewrite

$$\hat{y} = \text{sign}\left(\sum_{i=0}^{n} w_i x_i\right) \text{ as } \hat{y} = \text{sign}(w^T x)$$

Perceptron: The training phase

- The big idea picture.



- Training is all about adjusting (learning) the weight parameter, ***w***, until the response predicted by the perceptron becomes consistent with the true response.

Perceptron: The algorithm

1. Let $D = \{(x_i, y_i) | i = 1, 2, ...n\}$ be the set of training examples.
2. k $\leftarrow 0$
3. Initialize the weight vector with random values, $w^{(0)}$
4. **repeat**
5.   **for** each training example $(x_i, y_i) \in D$ **do**
6.   Compute the predicted output $\hat{y}_i^{(k)}$ using $w^{(k)}$
7.   **for** each weight component w_j **do**
8.   Update the weight, $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$
9.   **end for**
10. $k \leftarrow k + 1$
11.   **end for**
12. **until** $\left(\sum_{i=1}^{n} \left| y_i - \hat{y}_i^{(k)} \right| / n \right) < \gamma$

Perceptron: The algorithm

1. Let $D = \{(x_i, y_i) | i = 1, 2, ...n\}$ be the set of training examples.
2. k $\leftarrow$ 0
3. Initialize the weight vector with random values, $w^{(0)}$
4. **repeat**
5. **for** each training example $(x_i, y_i) \in D$ **do**
6. Compute the predicted output $\hat{y}_i^{(k)}$ using $w^{(k)}$
7. **for** each weight component w_j **do**
8. Update the weight, $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$
9. **end for**
10. $k \leftarrow k + 1$
11. **end for**
12. **until** $\left(\sum_{i=1}^{n} \left| y_i - \hat{y}_i^{(k)} \right| / n \right) < \gamma$

How does the perceptron predict?

Perceptron: The algorithm

1. Let $D = \{(x_i, y_i) | i = 1, 2, ...n\}$ be the set of training examples.
2. k $\leftarrow$ 0
3. Initialize the weight vector with random values, $w^{(0)}$
4. **repeat**
5. **for** each training example $(x_i, y_i) \in D$ **do**
6. Compute the predicted output $\hat{y}_i^{(k)}$ using $w^{(k)}$
7. **for** each weight component w_j **do**
8. Update the weight, $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$
9. **end for**
10. $k \leftarrow k + 1$
11. **end for**
12. **until** $\left(\sum_{i=1}^{n} \left| y_i - \hat{y}_i^{(k)} \right| / n \right) < \gamma$

How does the perceptron predict?

$\text{sign}(w^T x)$

Perceptron: The algorithm

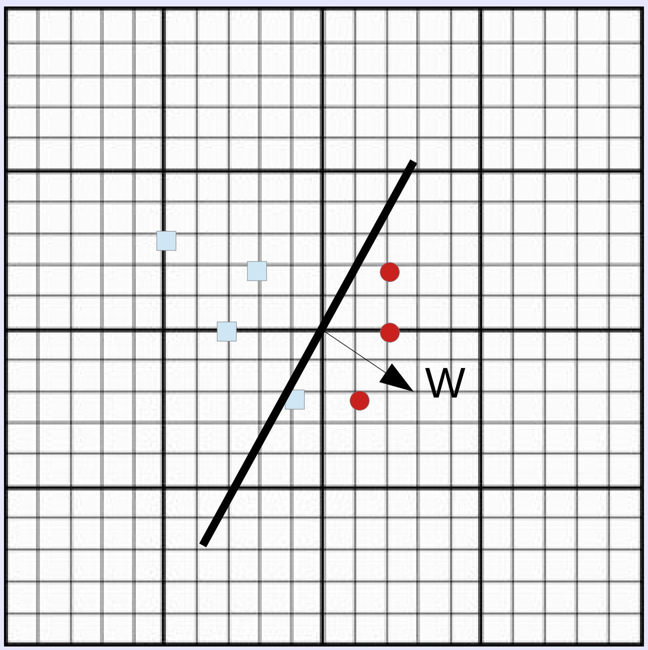
1. Let $D = \{(x_i, y_i) | i = 1, 2, ...n\}$ be the set of training examples.
2. k $\leftarrow$ 0
3. Initialize the weight vector with random values, $w^{(0)}$
4. **repeat**
5. **for** each training example $(x_i, y_i) \in D$ **do**
6. Compute the predicted output $\hat{y}_i^{(k)}$ using $w^{(k)}$
7. **for** each weight component w_j **do**
8. Update the weight, $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$
9. **end for**
10. $k \leftarrow k + 1$
11. **end for**
12. **until** $\left(\sum_{i=1}^{n} \left| y_i - \hat{y}_i^{(k)} \right| / n \right) < \gamma$

Perceptron: Geometric intuition

Case 1: A negative example is misclassified.

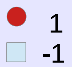
1
-1

$$\vec{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}, \text{Point misclassified} = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \text{True label = -1, Predicted label = 1}$$

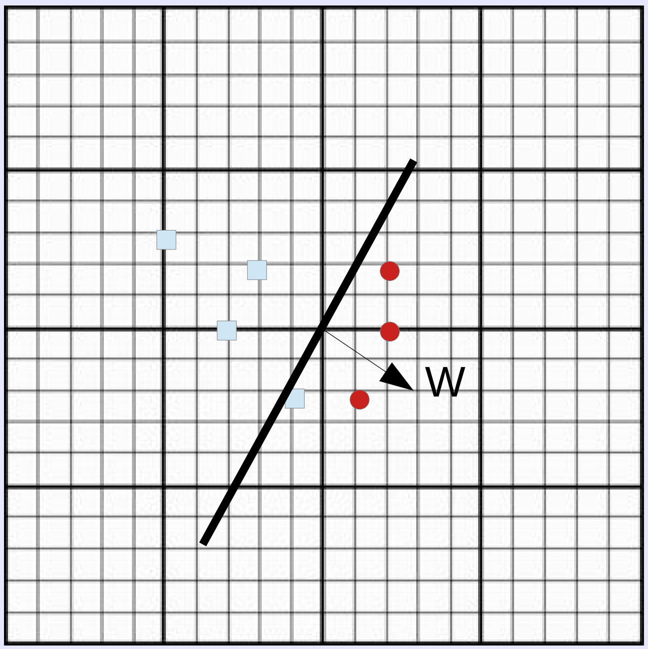


Perceptron: Geometric intuition

Case 1: A negative example is misclassified.



$\vec{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$, Point misclassified $= \begin{bmatrix} -1 \\ -2 \end{bmatrix}$, True label = -1, Predicted label = 1



Update the weights, $w = w + \lambda(y - \hat{y})x$

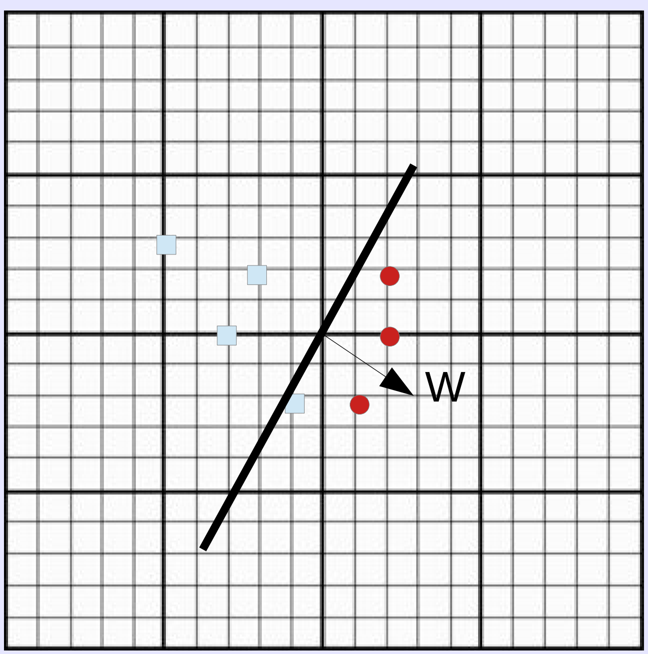
$$w = \begin{bmatrix} 3 \\ -2 \end{bmatrix} + (-1 - 1)\begin{bmatrix} -1 \\ -2 \end{bmatrix}$$
$$= \begin{bmatrix} 3 \\ -2 \end{bmatrix} + -2\begin{bmatrix} -1 \\ -2 \end{bmatrix}$$
$$= \begin{bmatrix} 3 \\ -2 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$
$$= \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

Perceptron: Geometric intuition

Case 1: A negative example is misclassified.

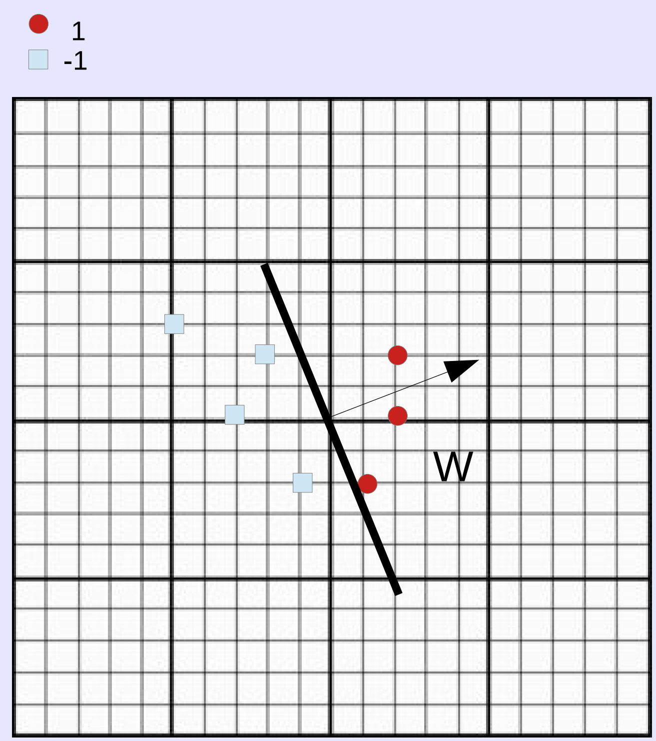
● 1
□ -1

$\vec{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$, Point misclassified $= \begin{bmatrix} -1 \\ -2 \end{bmatrix}$, True label = -1, Predicted label = 1



Update the weights, $w = w + \lambda(y - \hat{y})x$

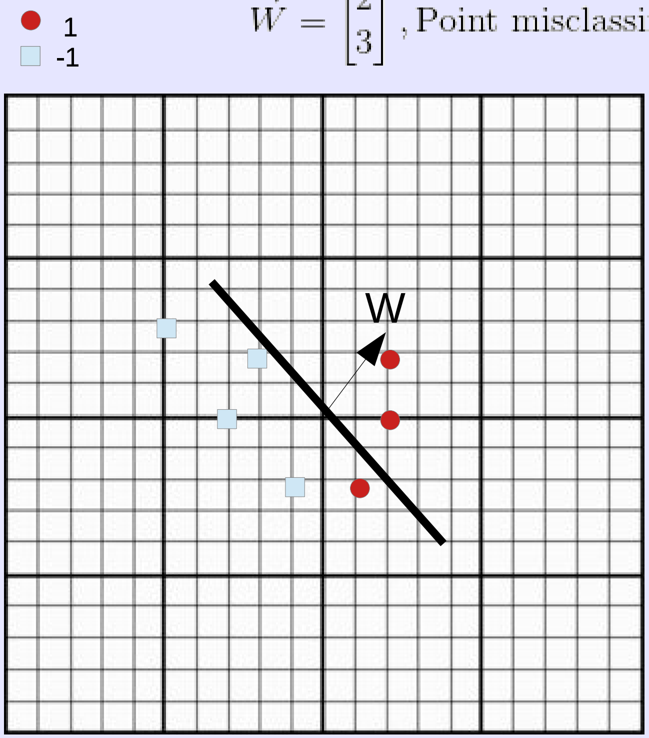
$$\begin{aligned} w &= \begin{bmatrix} 3 \\ -2 \end{bmatrix} + (-1 - 1) \begin{bmatrix} -1 \\ -2 \end{bmatrix} \\ &= \begin{bmatrix} 3 \\ -2 \end{bmatrix} + -2 \begin{bmatrix} -1 \\ -2 \end{bmatrix} \\ &= \begin{bmatrix} 3 \\ -2 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix} \\ &= \begin{bmatrix} 5 \\ 2 \end{bmatrix} \end{aligned}$$



Perceptron: Geometric intuition

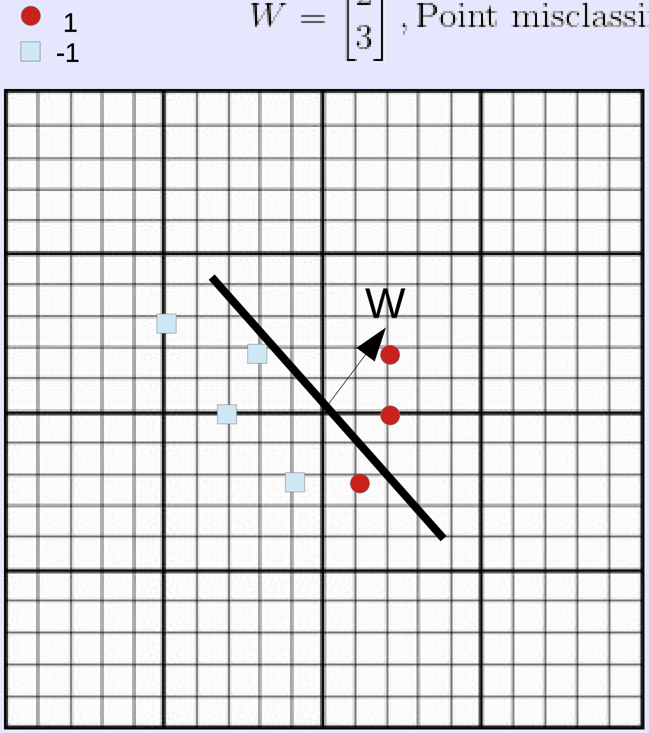
Case 2: A positive example is misclassified.

$\vec{W} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$, Point misclassified $= \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, True label = 1, Predicted label = -1



Perceptron: Geometric intuition

Case 2: A positive example is misclassified.



$\vec{W} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$, Point misclassified $= \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, True label $= 1$, Predicted label $=$ -1

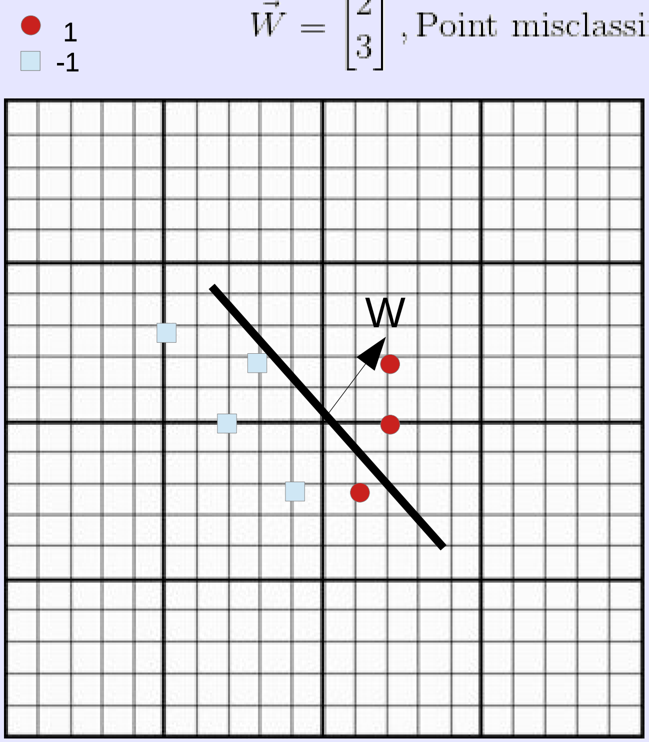
Update the weights, $w = w + \lambda(y - \hat{y})x$

$$\begin{aligned} w &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} + (1 - (-1)) \begin{bmatrix} 1 \\ -2 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} + 2 \begin{bmatrix} 1 \\ -2 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 2 \\ -4 \end{bmatrix} \\ &= \begin{bmatrix} 4 \\ -1 \end{bmatrix} \end{aligned}$$

Perceptron: Geometric intuition

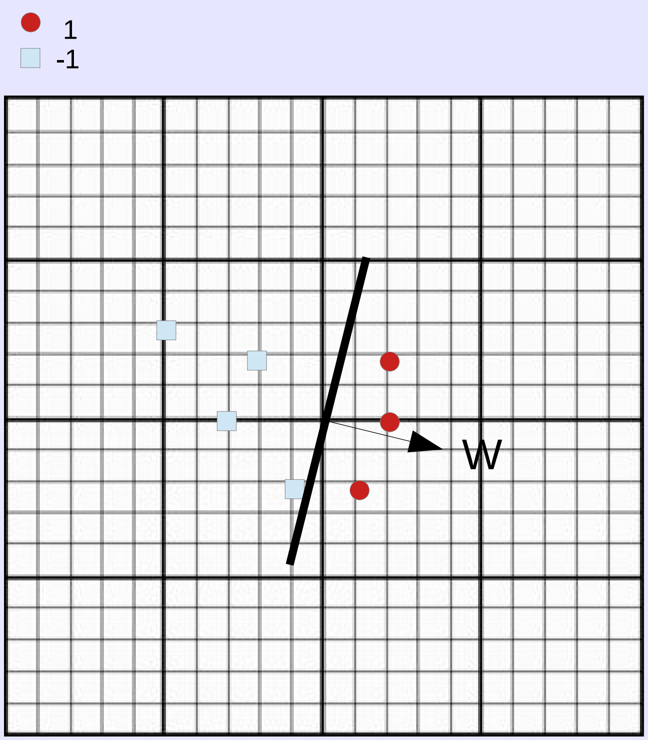
Case 2: A positive example is misclassified.

$\vec{W} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$, Point misclassified $= \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, True label $= 1$, Predicted label = -1



Update the weights, $w = w + \lambda(y - \hat{y})x$

$$\begin{aligned} w &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} + (1 - (-1)) \begin{bmatrix} 1 \\ -2 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} + 2 \begin{bmatrix} 1 \\ -2 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 2 \\ -4 \end{bmatrix} \\ &= \begin{bmatrix} 4 \\ -1 \end{bmatrix} \end{aligned}$$



Perceptron: Geometric intuition

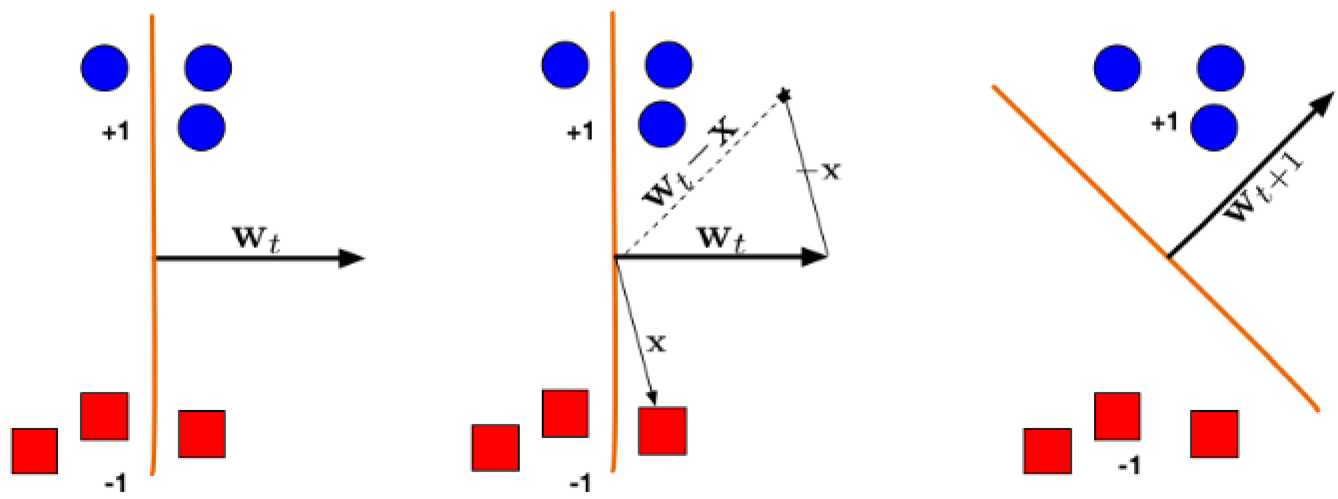


Illustration of a Perceptron update. (Left:) The hyperplane defined by $\mathbf{w}_t$ misclassifies one red (-1) and one blue (+1) point. (Middle:) The red point $\mathbf{x}$ is chosen and used for an update. Because its label is -1 we need to ***subtract*** *$\mathbf{x}$ from $\mathbf{w}_t$. (Right:) The udpated hyperplane $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$ separates the two classes and the Perceptron algorithm has converged.*

Slide credit: Kilian Weinberger, Cornell University

Perceptron: Hyperplane in 2D

Task: Figure out the hyperplane (find x-intercepts and y-intercepts of the hyperplane, and the slope).

Recall, $(w \cdot x) + b = 0$ (the dot product of inputs, weight vector, and bias)
$w_1 x_1 + w_2 x_2 + b = 0$, or
$w_1 x_1 + w_2 x_2 = -b$, What does this remind you of?

Perceptron: Hyperplane in 2D

Task: Figure out the hyperplane (find x-intercepts and y-intercepts of the hyperplane, and the slope).

Recall, $(w \cdot x) + b = 0$ (the dot product of inputs, weight vector, and bias)
$w_1 x_1 + w_2 x_2 + b = 0,$ or
$w_1 x_1 + w_2 x_2 = -b$, [(Ax + By = C), standard equation of a line]

Perceptron: Conclusion

- Very versatile model, capable of detecting class boundaries if they exist.
  - Can be extended for multi-class problems.
- Limited.
  - Even for binary class, a perceptron cannot recognize an XoR function.

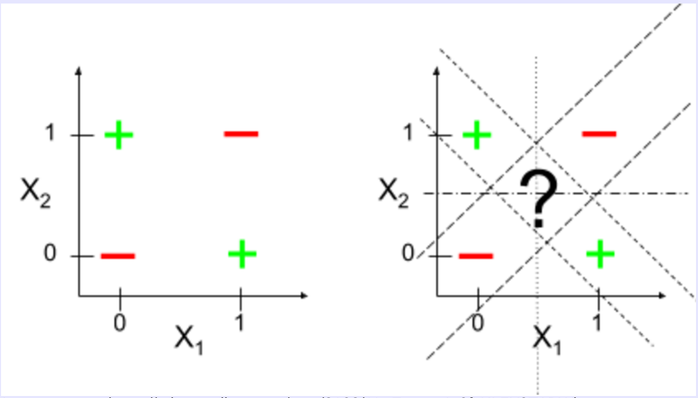
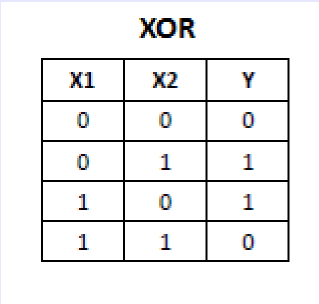


Image source: https://miro.medium.com/max/2108/1*wEomny4n9fzKL7X9TABWjw.png



XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

- To do this, you needed the multiple perceptron model: a general neural network.