

# *Inteligencia Artificial*

## Aprendizaje Supervisado

## 4. Aprendizaje Supervisado

4.1 Perceptrón Binario

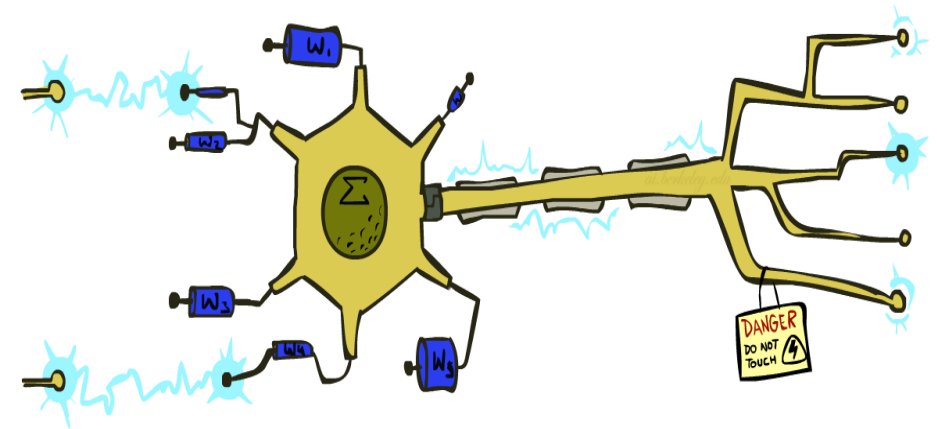
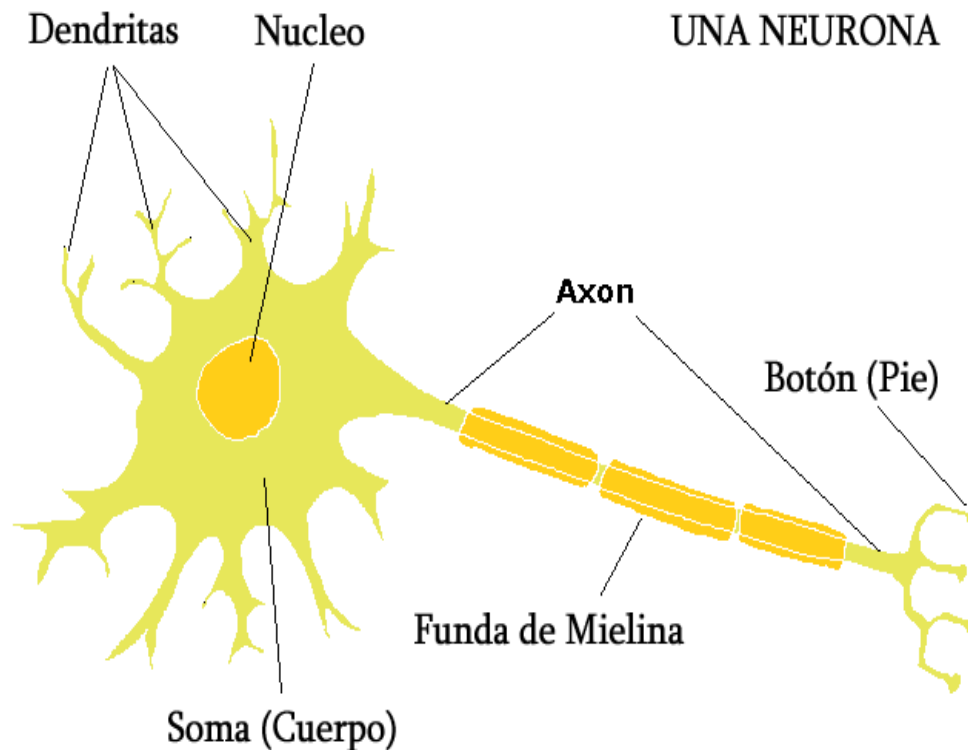
4.2 Perceptrón Multiclase

4.3 Averaged Perceptron

4.4 MIRA y SVM















# Un poco de Biología (Simplificada)

- Una ligera inspiración: neuronas humanas

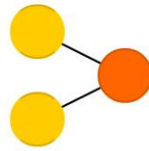


*Cuando las dendritas reciben mensajes químicos, las neuronas cambian el balance de iones (átomos cargados electrónicamente) entre su interior y el exterior de la membrana celular. Cuando este cambio alcanza un nivel umbral, este efecto se expande a través de la membrana de la célula hasta el axón. Cuando alcanza al axón, se inicia un potencial de acción (Dr. C. George Boeree).*

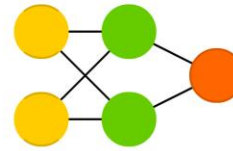
# Redes Neuronales (the asimov institute 2019)

-  Input Cell
-  Backfed Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Capsule Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Gated Memory Cell
-  Kernel
-  Convolution or Pool

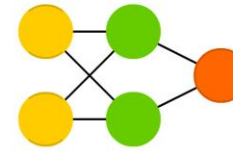
Perceptron (P)



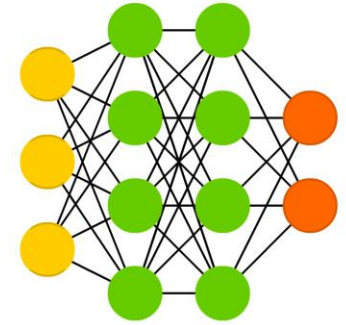
Feed Forward (FF)



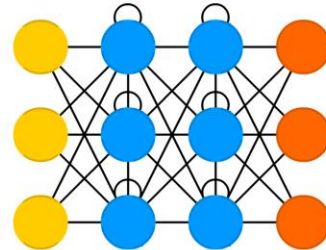
Radial Basis Network (RBF)



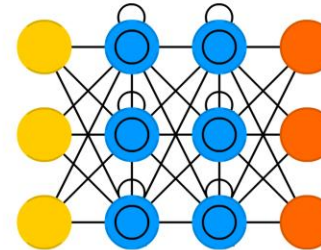
Deep Feed Forward (DFF)



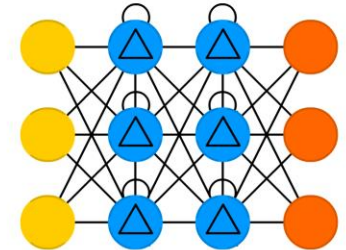
Recurrent Neural Network (RNN)



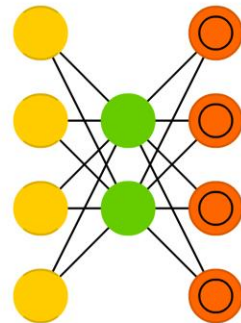
Long / Short Term Memory (LSTM)



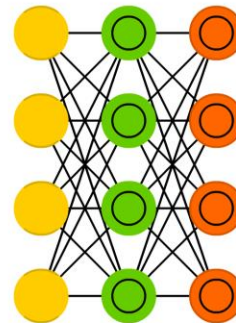
Gated Recurrent Unit (GRU)



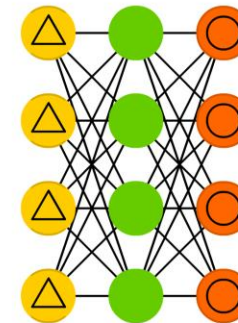
Auto Encoder (AE)



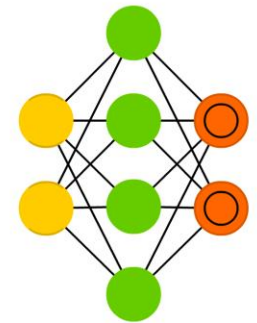
Variational AE (VAE)



Denoising AE (DAE)

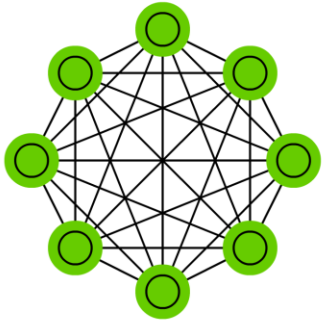


Sparse AE (SAE)

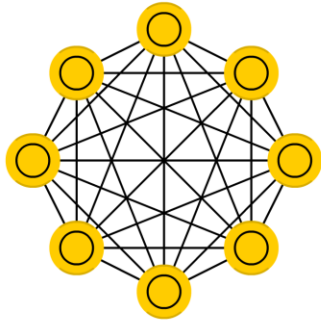


# Redes Neuronales (the asimov institute 2019)

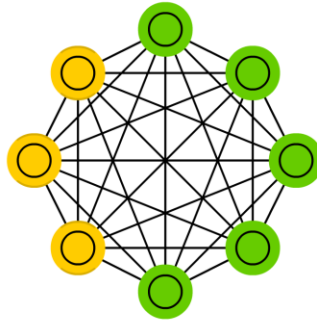
Markov Chain (MC)



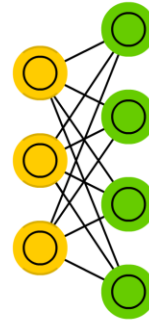
Hopfield Network (HN)



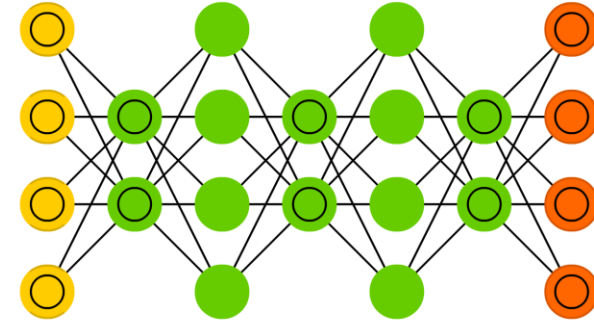
Boltzmann Machine (BM)



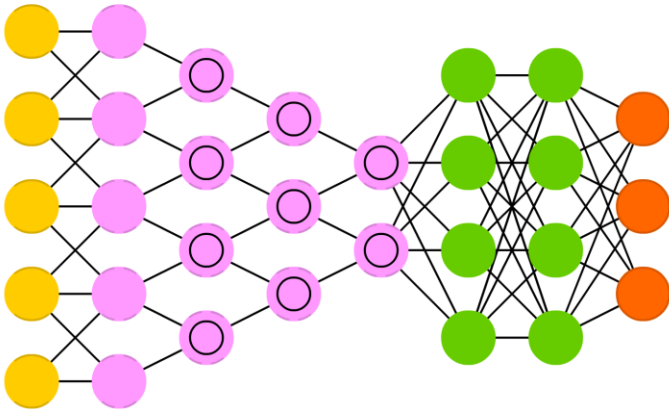
Restricted BM (RBM)



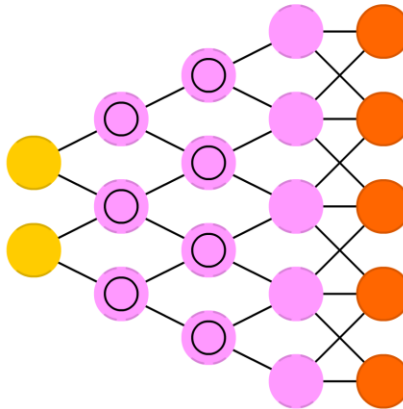
Deep Belief Network (DBN)



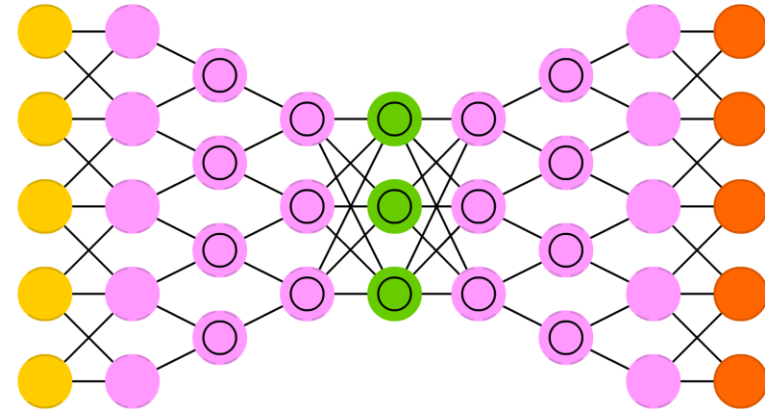
Deep Convolutional Network (DCN)



Deconvolutional Network (DN)



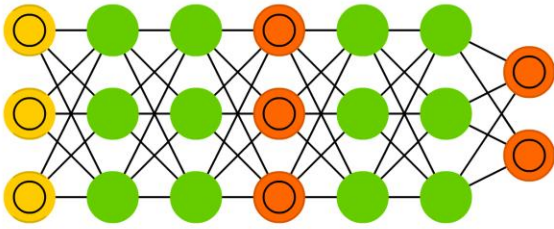
Deep Convolutional Inverse Graphics Network (DCIGN)



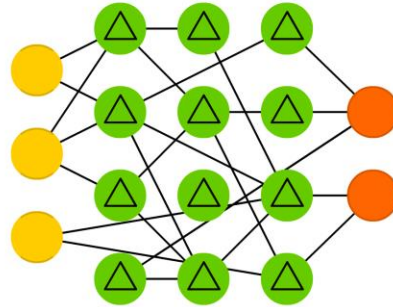


# Redes Neuronales (the asimov institute 2019)

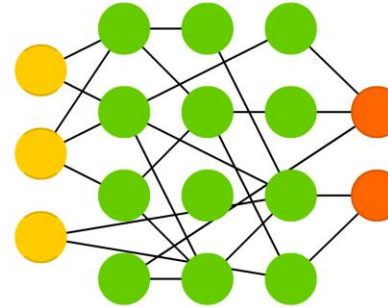
Generative Adversarial Network (GAN)



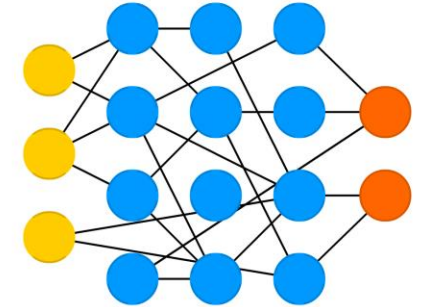
Liquid State Machine (LSM)



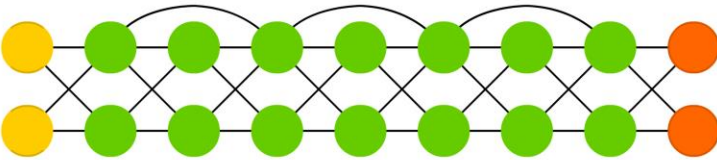
Extreme Learning Machine (ELM)



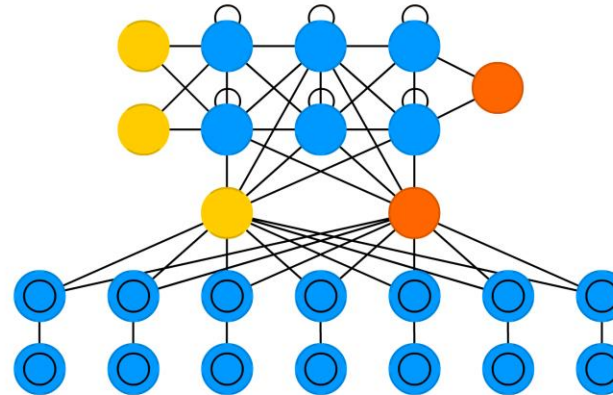
Echo State Network (ESN)



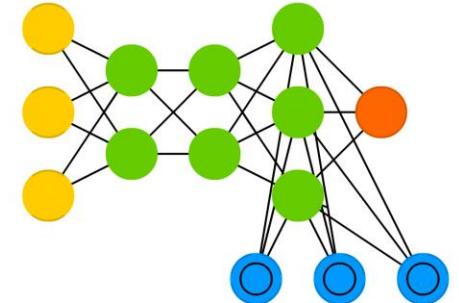
Deep Residual Network (DRN)



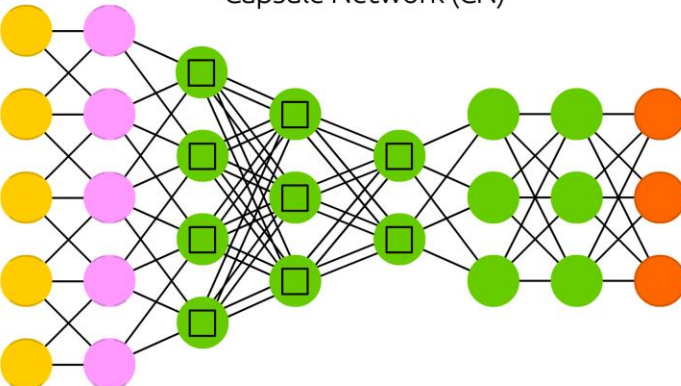
Differentiable Neural Computer (DNC)



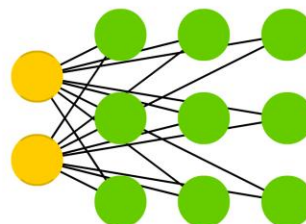
Neural Turing Machine (NTM)



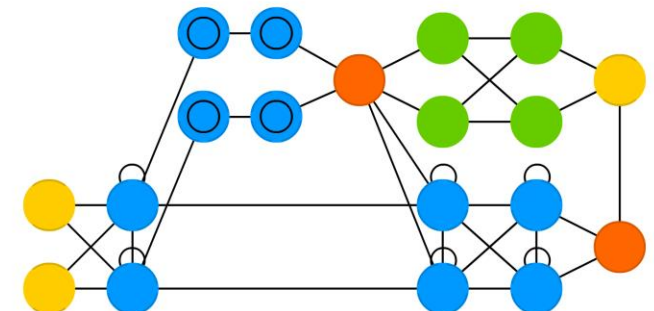
Capsule Network (CN)



Kohonen Network (KN)



Attention Network (AN)



# Clasificación guiada por el error



# Clasificación

## ➤ Ejemplos: es SPAM o no?

hola,  
¿Quieres recibir gratis un Iphone? No tienes más  
que clickar en el siguiente link ...

¡¡Hechamos la casa por la ventana!! ... descuentos  
increibiles para los primeros 100 usuarios que  
clicken en el siguiente link ...

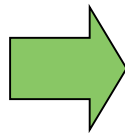


# Vectores de características

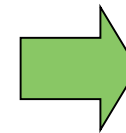
 $x$  $f(x)$  $y$ 

Hola,

¿Quieres cratuchos de  
impresora gratuitos?  
¿Porque pagar mas  
cuando los puedes  
conseguir  
¡COMPLETAMENTE GRATIS!  
Solo

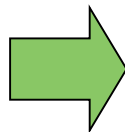


GRATIS	:	2
TU_NOMBRE	:	0
ORTOGR_ERR	:	2
EMISOR_CONOCIDO	:	0
...		

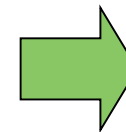


SPAM  
o  
NO SPAM

2



PIXEL 7,12	:	1
PIXEL 7,13	:	0
...		
NUM_LOOPS	:	1
...		



"2"

# ¿Qué hacer frente a los errores?

## ➤ Problema: todavía hay Spam en tu e-mail

Querido cliente de GlobalSCAPE,  
GlobalSCAPE se ha aliado a ScanSoft para ofrecerle la última versión de OmniPage Pro, por solo 99.99 euros- el precio habitual es ;499!-. La más frecuente de las preguntas sobre esta oferta es: ¿Es real? Queremos asegurarle que esta oferta está autorizada por ScanSoft, y es genuina y veraz. Puede recibir ...

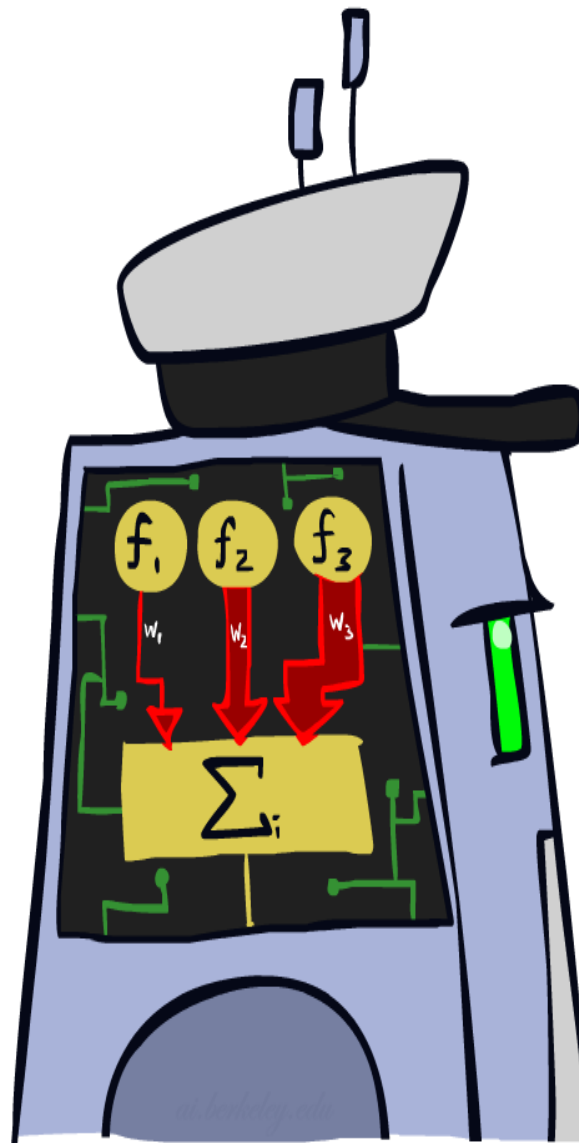
... Para recibir el certificado promocional de 30 euros, clicar sobre el siguiente link  
<http://www.amazon.com/apparel>  
Y verá el link que aparece resaltado de 30 euros de oferta. Podrá encontrar todos los detalles allí. Tenga en cuenta, que si no quiere recibir futuros e-mails anunciando nuevos lanzamientos, por favor haga click en ...

## ➤ Necesitas más **características** ; las palabras contenidas en el e-mail no son suficientes!

- ¿El emisor de ese e-mail **pertenece a tus contactos**?
- ¿Hay **más usuarios** que han **recibido el mismo e-mail**?
- ¿**Las URL** que aparecen realmente **te dirigen a los lugares que anuncian**?
- ¿Se dirigen a ti por **tu nombre**?

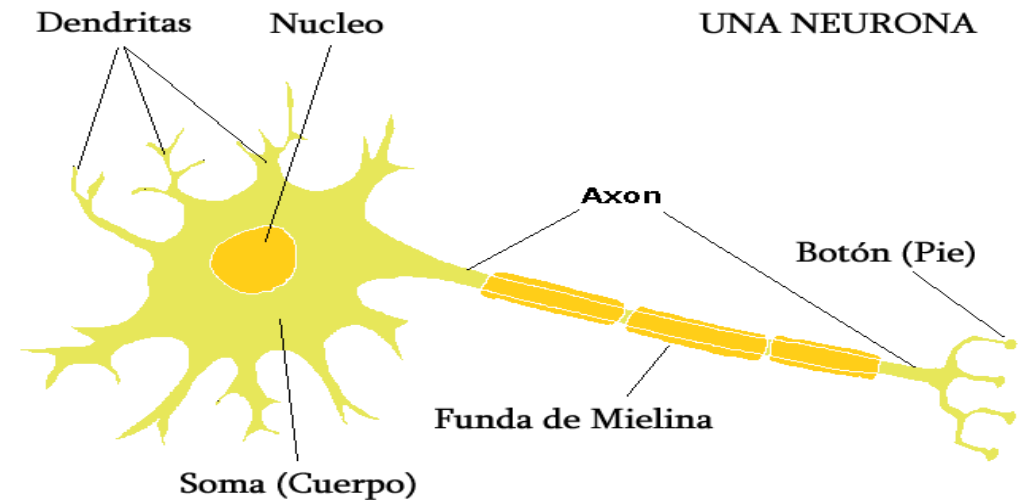
## ➤ **Importante:** Si no se extraen características relevantes, de nada sirve el entrenamiento de la red!

# Clasificadores

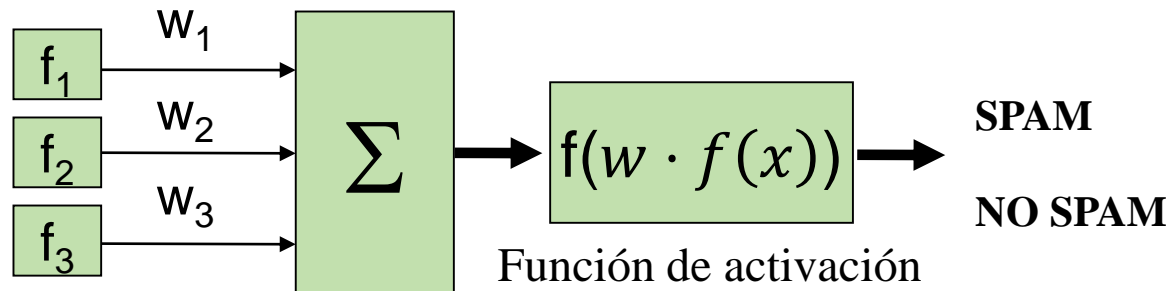


# Clasificadores

- Las entradas son **valores de una característica**
- A cada característica se le asigna un **peso**
- La suma **es la activación**

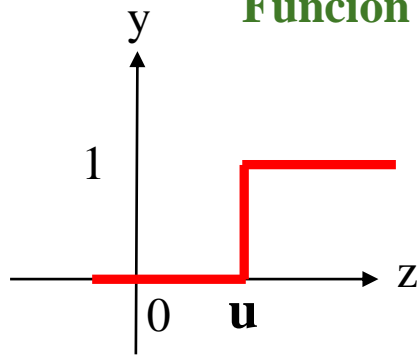


$$\text{activación } n_w(x) = \sum_i w_i \cdot f_i(x) = \mathbf{w} \cdot \mathbf{f}(x)$$



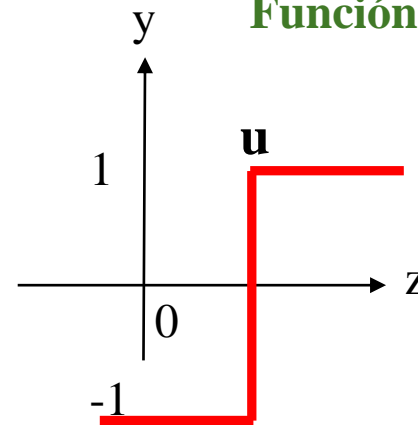
# Función de activación

Función umbral

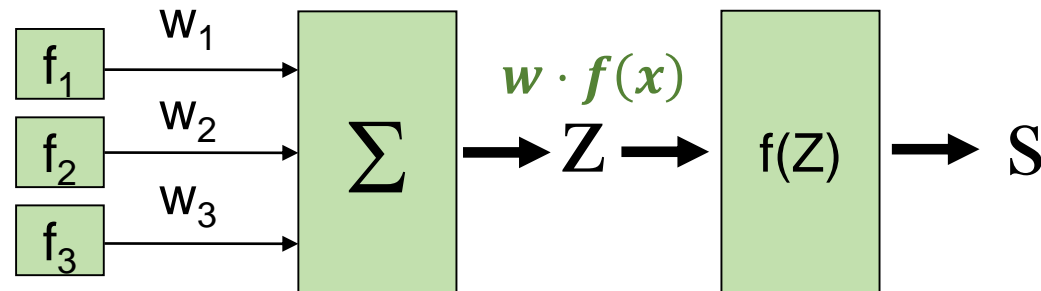


$$y = \begin{cases} 1 & \text{if } z > u \\ 0 & \text{else} \end{cases}$$

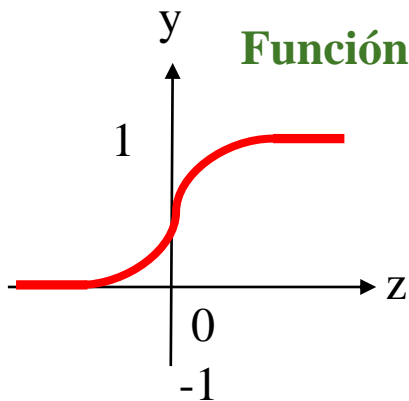
Función signo



$$y = \begin{cases} 1 & \text{if } z \geq u \\ -1 & \text{else} \end{cases}$$

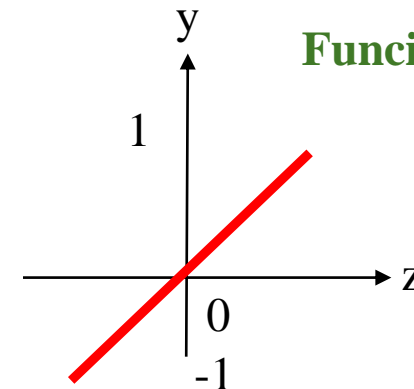


Función sigmoide



$$y = \frac{1}{1 + e^{-z}}$$

Función lineal



$$y = z$$

# Pesos

- **Clasificación binaria:** multiplicar el vector de las **características** con el vector de los **pesos**  $> 0$  o  $< 0$  (umbral)
- **Aprendizaje:** a partir de los ejemplos ajustar/adequar los pesos

**pesos**

$$\begin{pmatrix} \text{GRATIS} & : & 4 \\ \text{TU\_NOMBRE} & : & -1 \\ \text{ORTOGR\_ERR} & : & 1 \\ \text{EMISOR\_CONOCIDO} & : & -3 \\ \dots & & \end{pmatrix}$$

$w$

**Producto escalar**  $w \cdot f$

**Por ejemplo, salida:**

**Positivo** ➡ **SPAM**

**Negativo** ➡ **NO SPAM**

**Características del EMAIL 1**

$$f(x_1) \begin{pmatrix} \text{GRATIS} & : & 2 \\ \text{TU\_NOMBRE} & : & 0 \\ \text{ORTOGR\_ERR} & : & 2 \\ \text{EMISOR\_CONOCIDO} & : & 0 \\ \dots & & \end{pmatrix}$$

$$4*2 + -1*0 + 1*2 + -3*0 = 10 \rightarrow \text{SPAM}$$

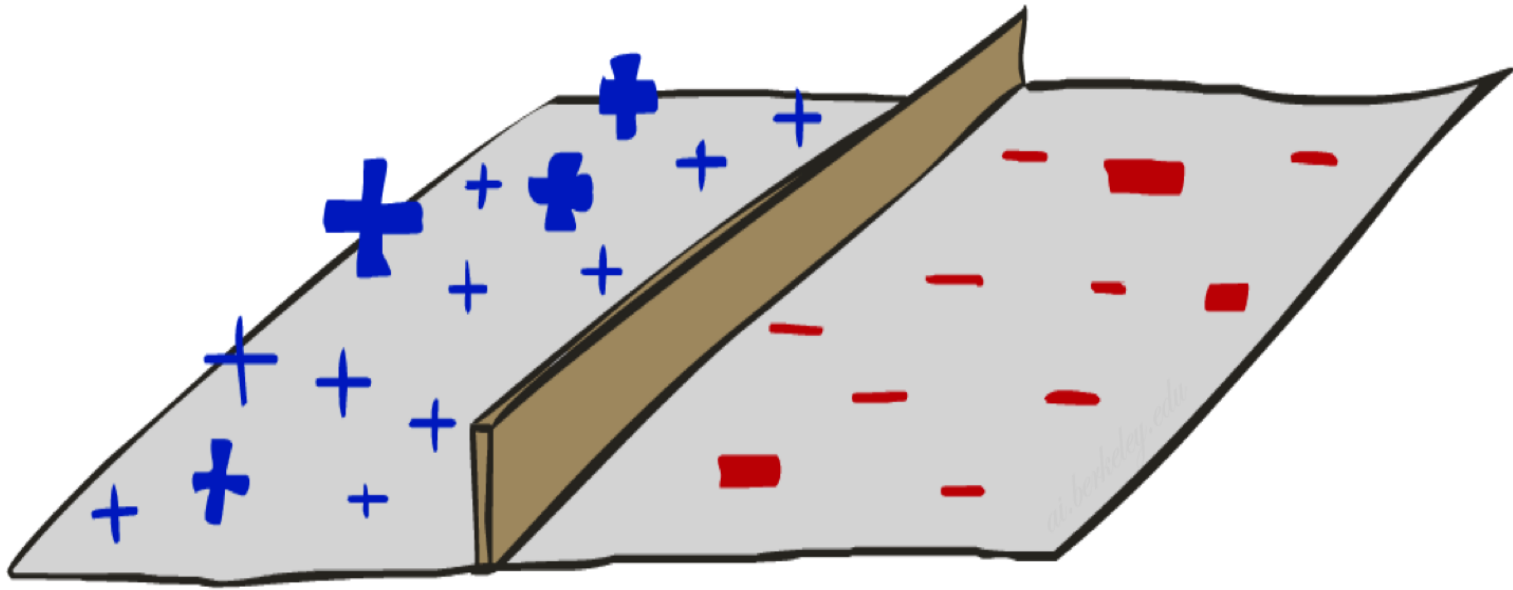
$$4*0 + -1*1 + 1*1 + -3*1 = -3 \rightarrow \text{NO SPAM}$$

$$f(x_2) \begin{pmatrix} \text{GRATIS} & : & 0 \\ \text{TU\_NOMBRE} & : & 1 \\ \text{ORTOGR\_ERR} & : & 1 \\ \text{EMISOR\_CONOCIDO} & : & 1 \\ \dots & & \end{pmatrix}$$

**Características del EMAIL 2**



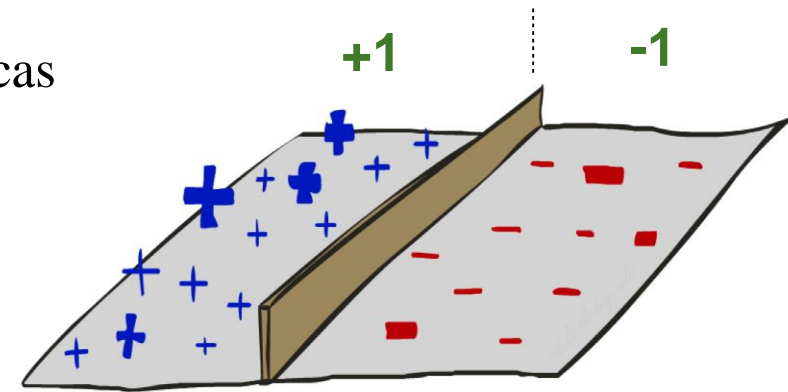
# Fronteras de decisión



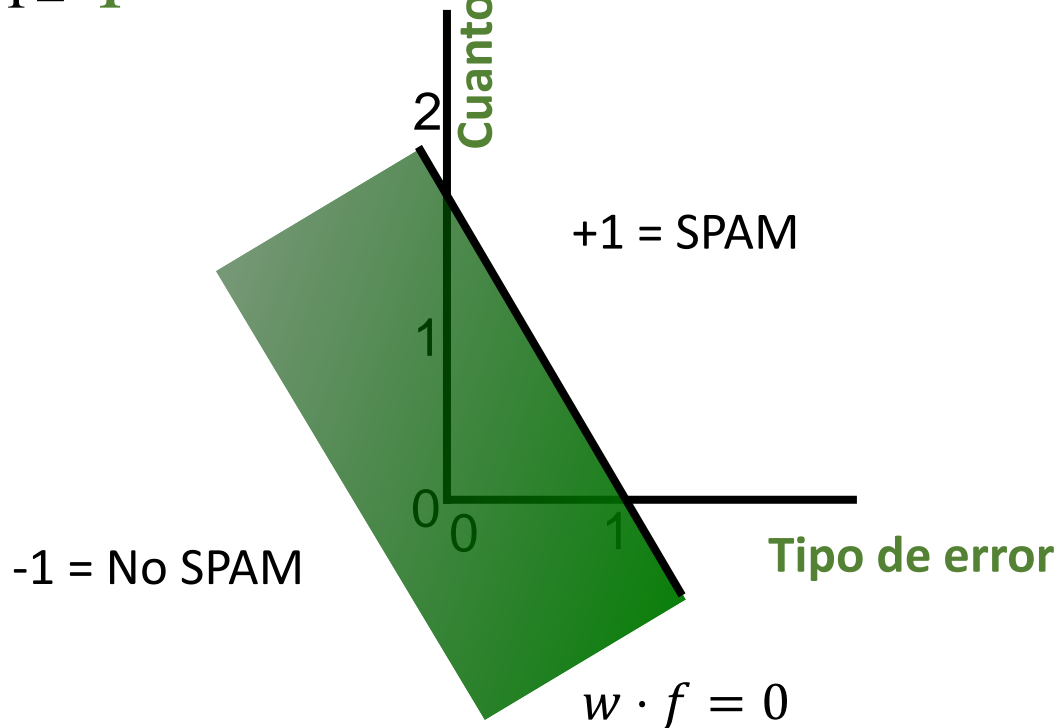
# Decisión binaria

➤ En el espacio generado por los vectores de características

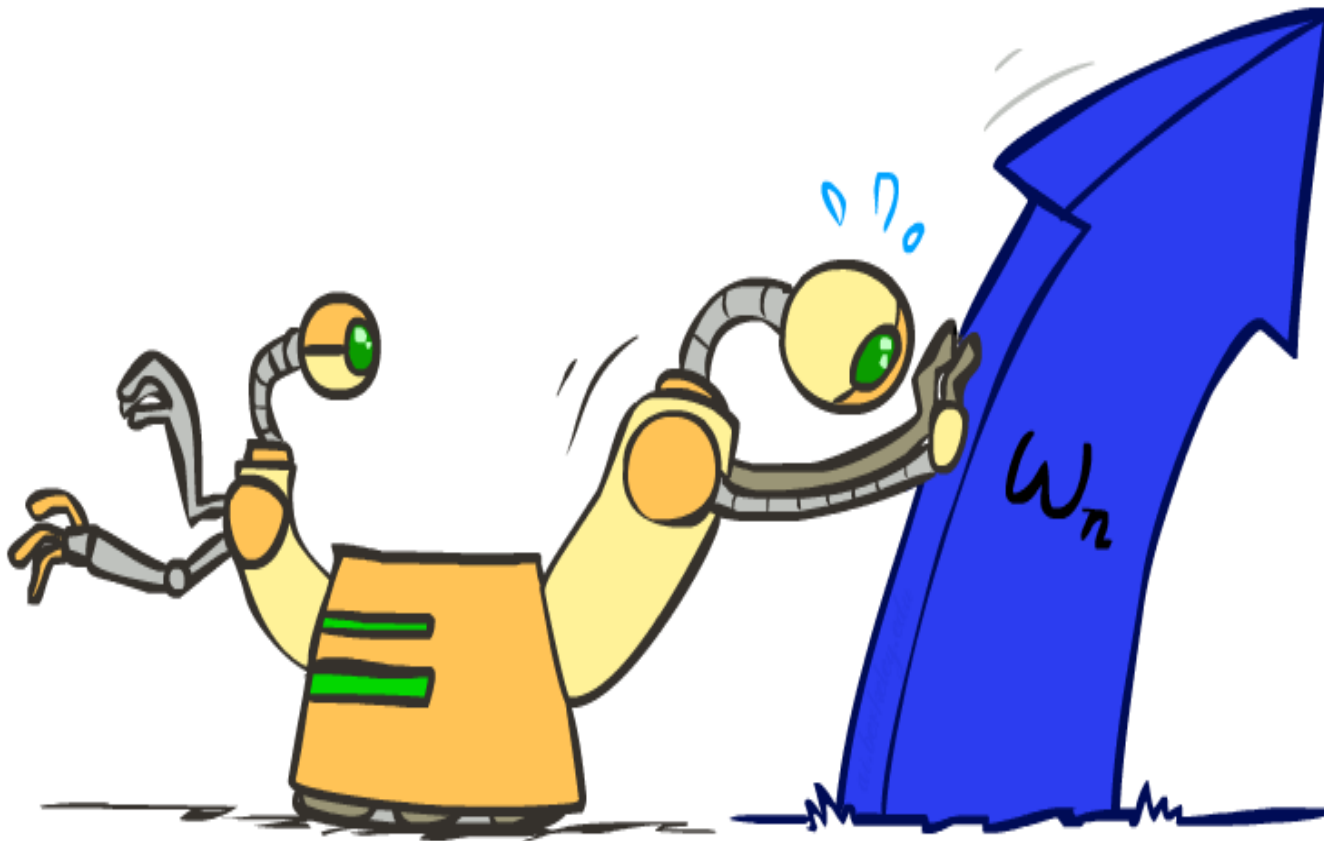
- Los ejemplos son puntos
- Cualquier vector de pesos será un **hiperplano**
- Un lado se corresponde con  $Y = +1$
- El otro se corresponde con  $Y = -1$



$w$	
BIAS	: -3
Gratis	: 4
dinero	: 2
...	



# Actualización de pesos



# Aprendizaje: Perceptrón Binario

$y$  la clase correcta,  $y'$  la predecida

➤ Inicializar pesos = 0 (**único vector de pesos**)

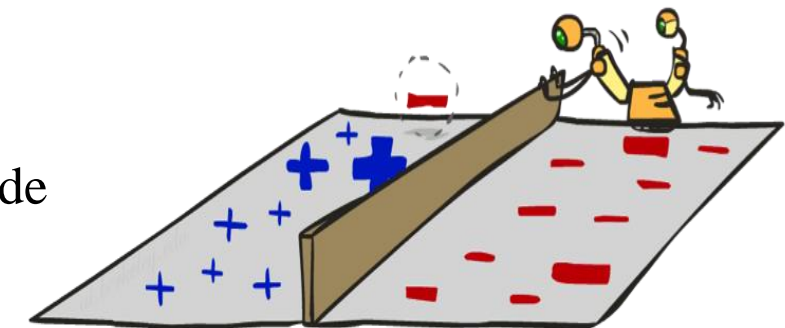
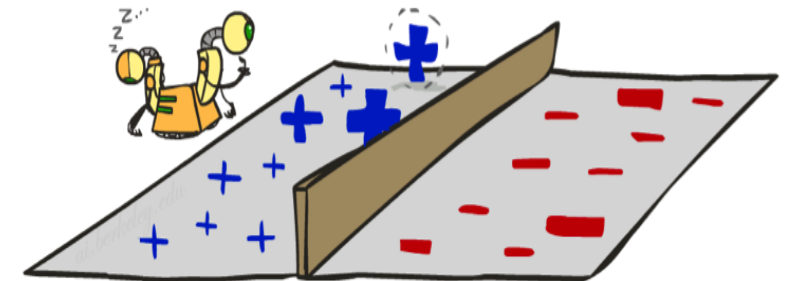
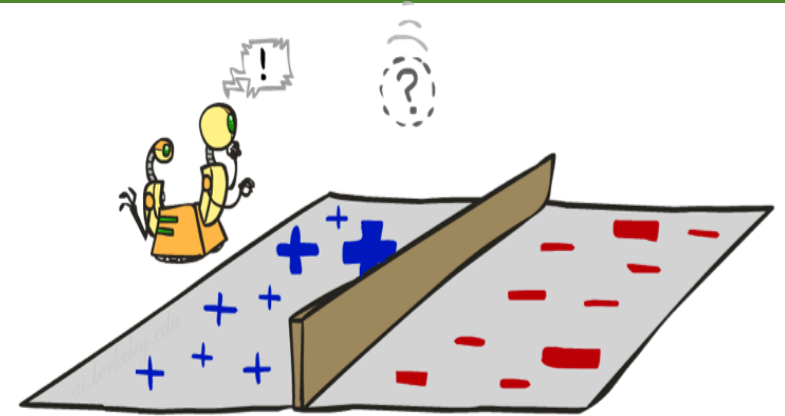
➤ Por cada instancia de entrenamiento:

– Clasificar empleando los pesos actuales

$$y' = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

– Si es correcto ( $y' = y$ ), ¡no cambiar!

– Si es incorrecto: ajustar (**actualizar**) el vector de pesos



# Aprendizaje: Perceptrón Binario

➤ Inicializar pesos = 0 (**único vector de pesos**)

➤ Por cada instancia de entrenamiento:

– Clasificar empleando los pesos actuales

$$y' = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

– Si es correcto ( $y' = y$ ), ¡no cambiar los pesos!

– Si es incorrecto: ajustar (**actualizar**) el vector de pesos sumando o restando el vector de las características.

▪ Restar si la clase  $y$  (la real) es -1

▪ Sumar si la clase  $y$  (la real) es +1

– Entonces si  $y'$  ha sido 1 &  $y$  es -1  $\rightarrow w = w - f(x)$

– Entonces si  $y'$  ha sido -1 &  $y$  es 1  $\rightarrow w = w + f(x)$

$$w = w + (y * f(x))$$

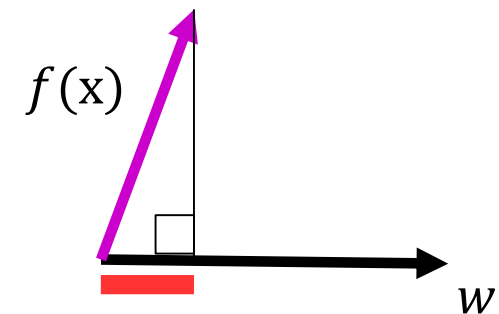
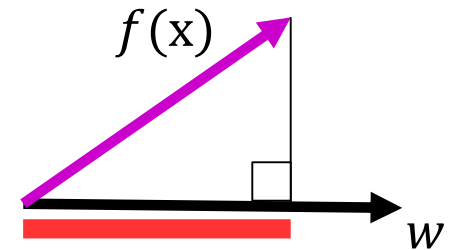
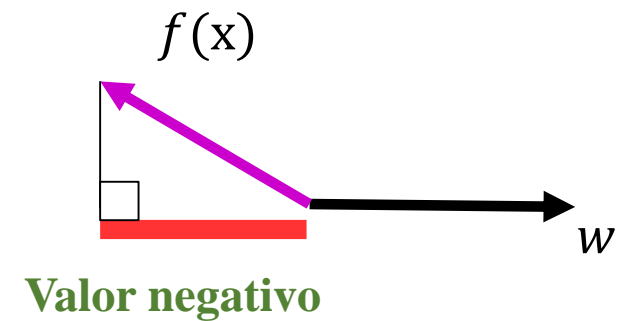
# Aprendizaje: producto escalar de vectores y su significado

- Producto escalar de dos vectores: **Distancia del coseno**
- Proyección de un vector sobre el otro: **Coseno**
- Por cada instancia de entrenamiento:
  - Clasificar empleando los pesos actuales

$$y' = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{f}(\mathbf{x}) \geq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{f}(\mathbf{x}) < 0 \end{cases}$$

Intuición tras el producto escalar:

*Cuanto de  $f$  va en la misma dirección que  $w$  o cuanto se asemejan*



*¿Qué sucede cuándo son ortogonales? ¿Y qué sucede cuándo son iguales?*



# Aprendizaje: Perceptrón Binario

- Inicializar pesos = 0
- Por cada instancia de entrenamiento:
  - Clasificar empleando los pesos actuales

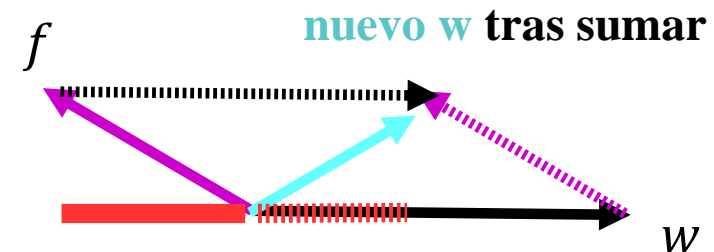
$$y' = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{f}(\mathbf{x}) \geq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{f}(\mathbf{x}) < 0 \end{cases}$$

- Si es correcto ( $y' = y$ ), ¡no cambiar!
- Si es incorrecto: ajustar (actualizar) el vector de pesos sumando o restando el vector de las características.

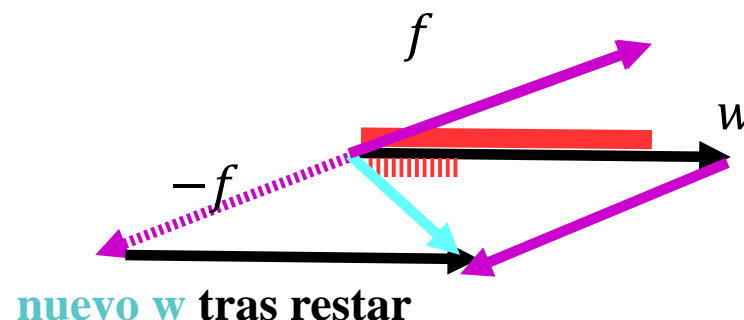
$$\mathbf{w}' = \mathbf{w} - \mathbf{f}(\mathbf{x})$$

$$\mathbf{w}' = \mathbf{w} + \mathbf{f}(\mathbf{x})$$

- Restar si la clase  $y$  (real) es -1
- Sumar en caso contrario



Te ha dado  $y' = -1$   
pero debería de haber sido **1** luego tenemos  
que intentar **acercar** los vectores para que el  
coseno si se repitiese  $f$  (o similar) te de  
positivo  
Por eso **sumamos**



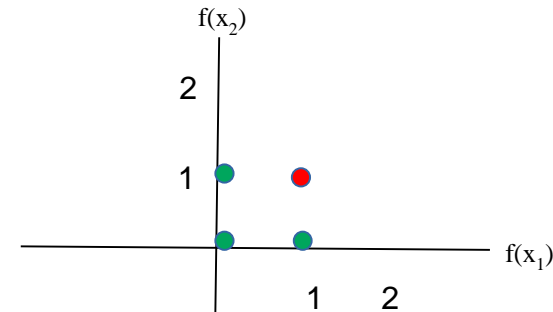
Te ha dado  $y' = 1$   
pero debería de haber  
sido **-1** luego tenemos  
que intentar **alejar** los  
vectores para que el  
coseno si se repitiese  $f$   
(o similar) te de  
negativo  
Por eso **restamos**

# Ejemplo: AND

- Se quiere hacer un perceptrón que entienda el operador AND

Inputs		Output
$f(x_1)$	$f(x_2)$	Z
0	0	0 <span style="color: green;">●</span>
0	1	0 <span style="color: green;">●</span>
1	0	0 <span style="color: green;">●</span>
1	1	1 <span style="color: red;">●</span>

Graficando los puntos:



- ¿Existe recta para dividir los puntos? Inicialización de pesos por RANDOM y valor BIAS = 1

$f(x_1)$	$f(x_2)$	Z	Bias	Clase correcta: <span style="color: orange;">y</span>
0	0	0	1	-1
0	1	0	1	-1
1	0	0	1	-1
1	1	1	1	1

$$w_0 = 0.03 \quad w_1 = 0.66 \quad w_2 = 0.80$$

$$y' = w_0 * \mathbf{BIAS} + w_1 * \mathbf{f}(x_1) + w_2 * \mathbf{f}(x_2) \dots$$

$$y' = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

- Primer conjunto:  $f(x_1) = 0, f(x_2) = 0$ , **clase = -1**

$(0.03 * 1) + (0.66 * 0) + (0.8 * 0) = 0.03 \Rightarrow$  al ser  $\geq 0 \Rightarrow$  Predicción de clase: 1  $\Rightarrow$  **INCORRECTO**

- Ajuste de los pesos:  $w_0 = w_0 + \text{claseCorrecta} * \text{bias}$   $w_n = w_n + \text{claseCorrecta} * f(x_n)$

$$w_0 = 0.03 + (-1) * 1 = -0.97$$

$$w_1 = 0.66 + (-1) * 0 = 0.66$$

$$w_2 = 0.8 + (-1) * 0 = 0.8$$

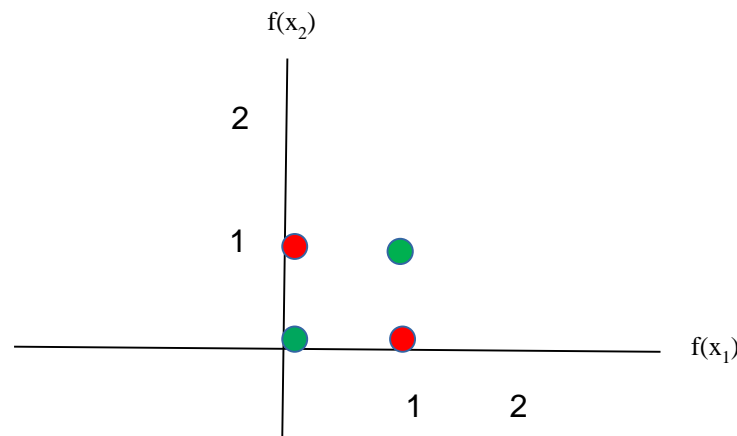
# Ejercicio: XOR

- Se quiere hacer un perceptrón que entienda el operador XOR
  - Inicialización de pesos por RANDOM y valor BIAS = 1

$$w_0 = 0.03 \quad w_1 = 0.66 \quad w_2 = 0.80$$

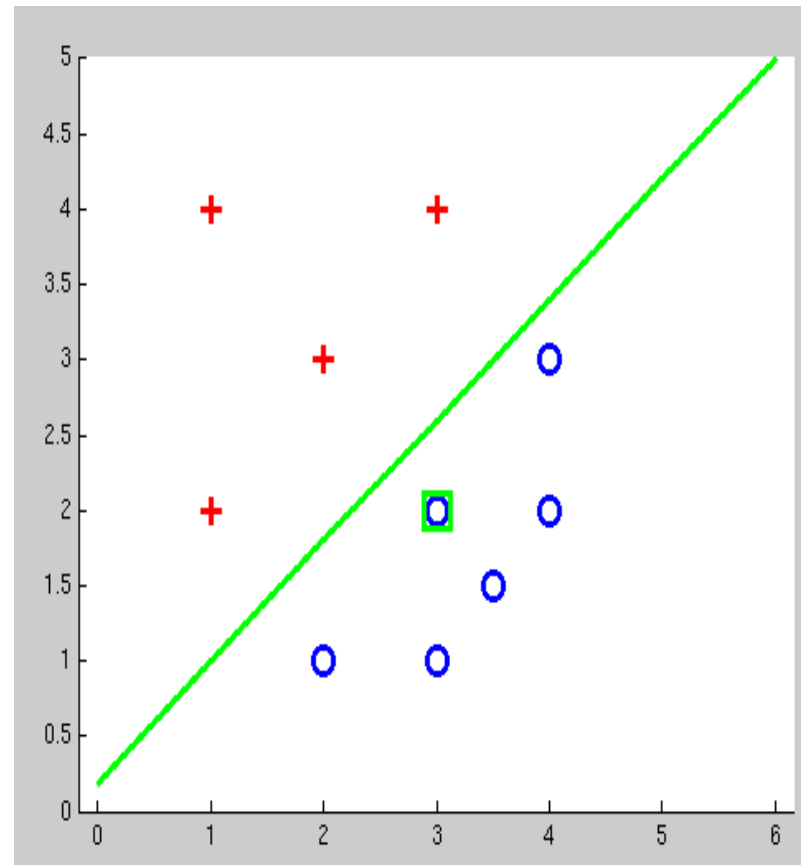
Inputs		Output
$f(x_1)$	$f(x_2)$	Z
0	0	0
0	1	1
1	0	1
1	1	0

- Cual sería la ecuación de la recta?



# Ejemplos: Perceptrón

## ➤ Casos separables



# Fronteras en decisiones Multiclase

- En clasificación multiclase:
  - Un vector de pesos por cada clase:

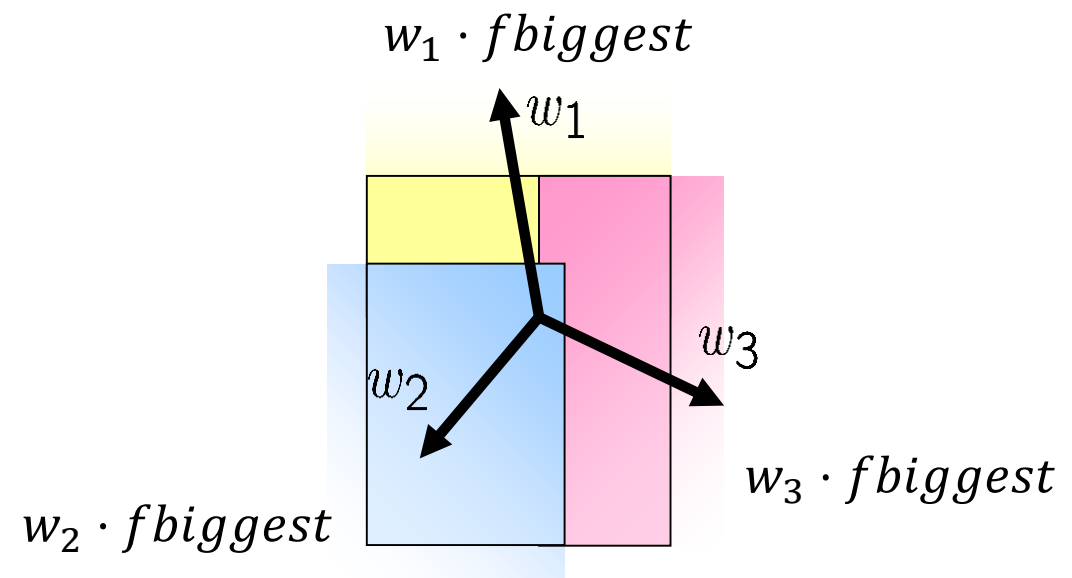
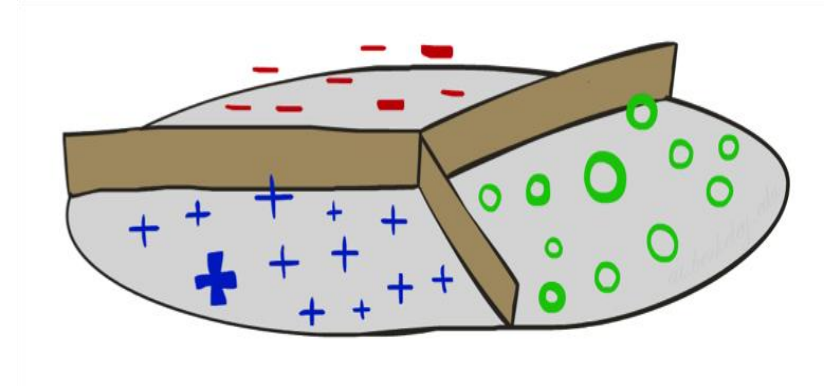
$$w_y$$

- Score (**activación**) de la clase  $y$ :

$$w_y \cdot f(x)$$

- Predicción con mayor score gana

$$y = \arg \max_y w_y \cdot f(x)$$



# Aprendizaje: Perceptrón Multiclase

- Inicializar los pesos = 0
- Por cada ejemplo
  - Clasificar empleando los pesos actuales

$$y = \arg \max_y w_y \cdot f(x)$$

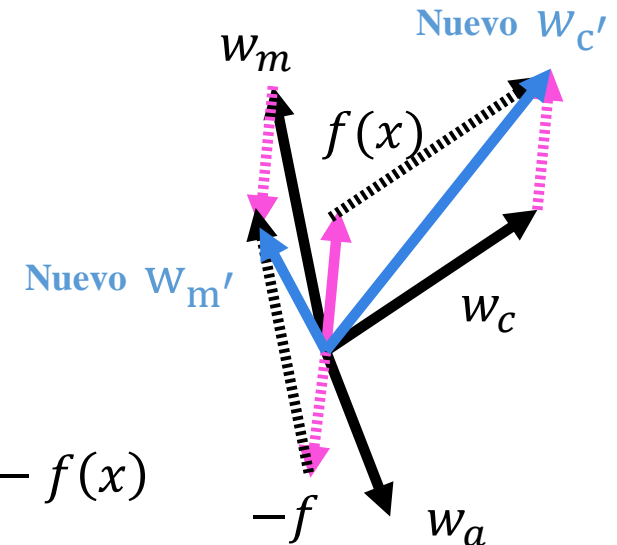
- Si es **correcto**, ¡no cambiar!

- Si es **incorrecto**:

- decrementar el score de la incorrecta  $\rightarrow w_{m'} = w_m - f(x)$
- aumentar el de la correcta  $\rightarrow w_{c'} = w_c + f(x)$

$w_c$  ➡ coche  
 $w_m$  ➡ moto  
 $w_a$  ➡ autobús

Me dice que es **moto**, pero sé que es **¡coche!**  
Sumo a coche y resto a moto, y los de autobús no los toco



$w_c$  Vector de pesos asociado a la clase correcta

$w_m$  Vector de pesos asociado a la clase predecida



# Ejemplo: Perceptrón Multiclase

$X_1$ : “ganar el voto”

$f(x_1) =$

BIAS	: 1
ganar	: 1
juego	: 0
voto	: 1
el	: 1
La	: 0
elección:	0
...	

$X$  : “ganar la elección”

$f(x_2) =$

BIAS	: 1
ganar	: 1
juego	: 0
voto	: 0
el	: 0
La	: 1
elección:	1
...	

$X$  : “ganar el juego”

$f(x_3) =$

BIAS	: 1
ganar	: 1
juego	: 1
voto	: 0
el	: 1
La	: 0
elección:	0
...	

Deporte

BIAS	: 1
ganar	: 0
juego	: 0
voto	: 0
el	: 0
La	: 0
elección:	0
...	

Política

BIAS	: 0
ganar	: 0
juego	: 0
voto	: 0
el	: 0
La	: 0
elección:	0
...	

Tecnología

BIAS	: 0
ganar	: 0
juego	: 0
voto	: 0
el	: 0
La	: 0
elección:	0
...	

Pesos iniciales:

$W_D$

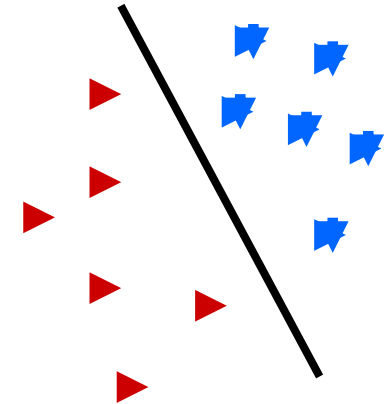
$W_P$

$W_T$

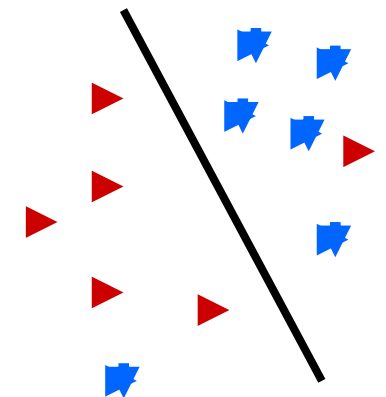
# Propiedades del Perceptron

- **Separabilidad:** Existen vectores de pesos tal que los ejemplos del entrenamiento queden correctamente clasificados
- **Convergencia:** Si los ejemplos del conjunto de entrenamiento son separables, el perceptrón convergerá
- **Limite de error:** El número máximo de errores asociados al margen o grado de separabilidad

Separable

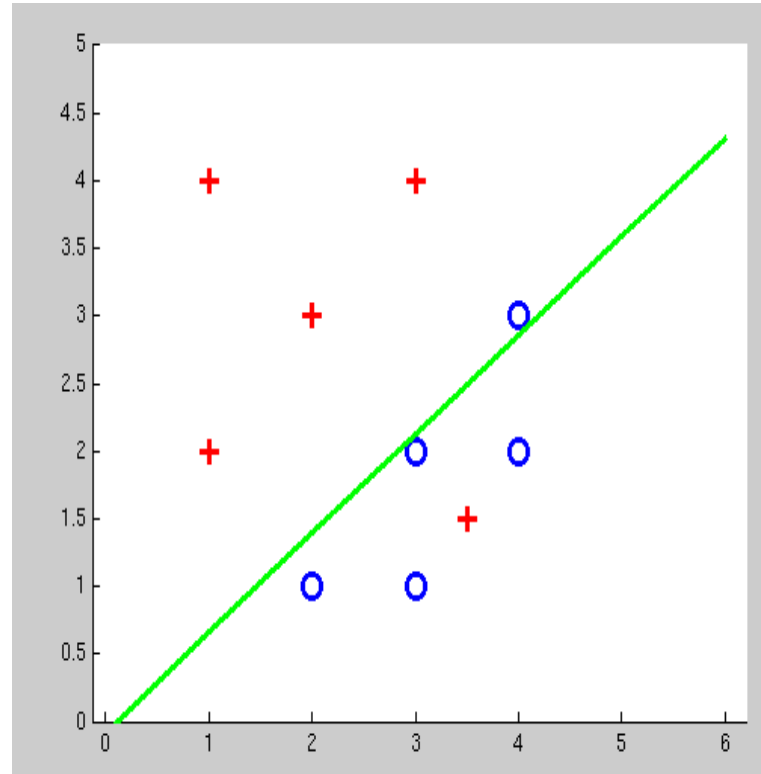


No-Separable

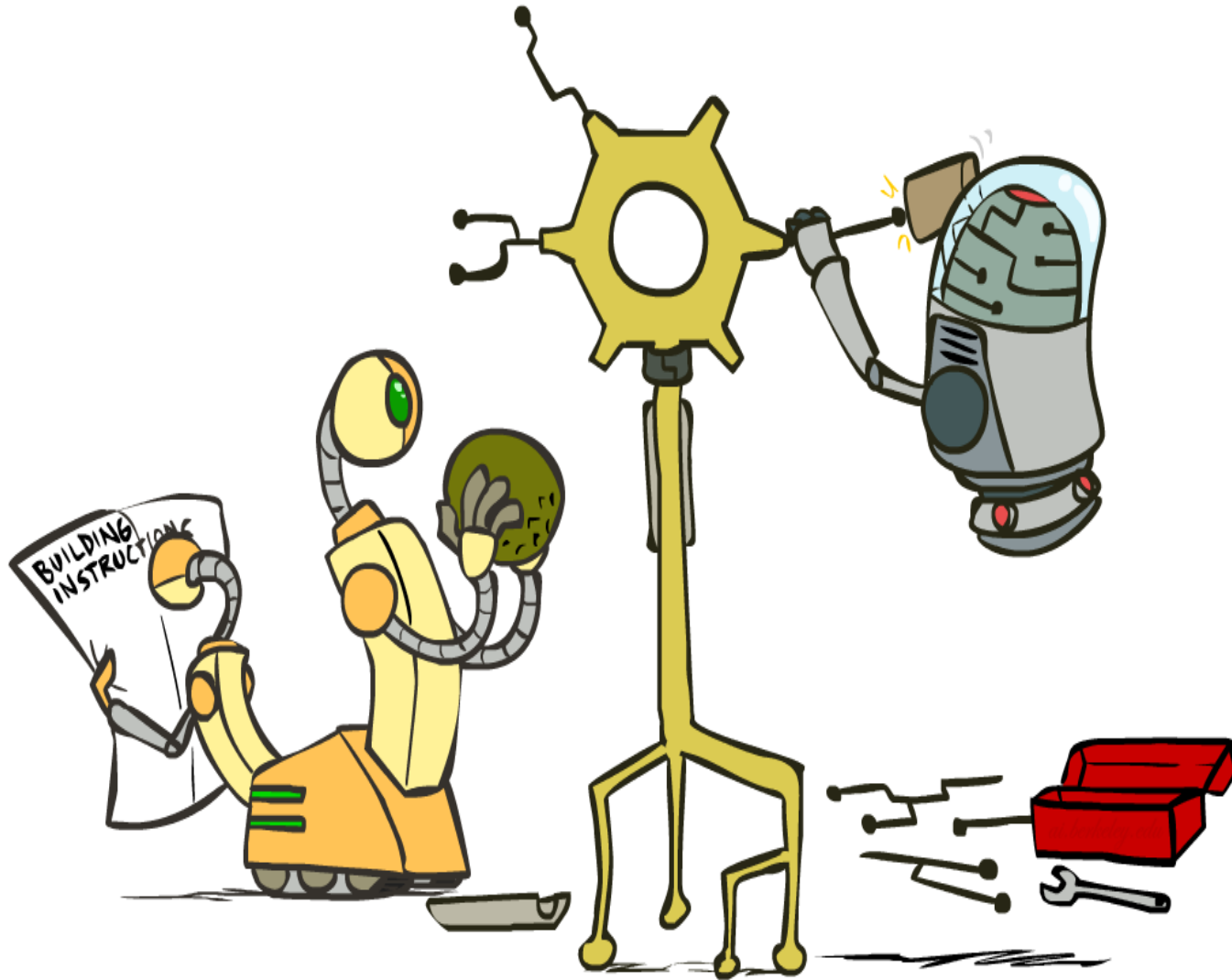


# Ejemplos: Perceptrón

## ➤ Caso no separable



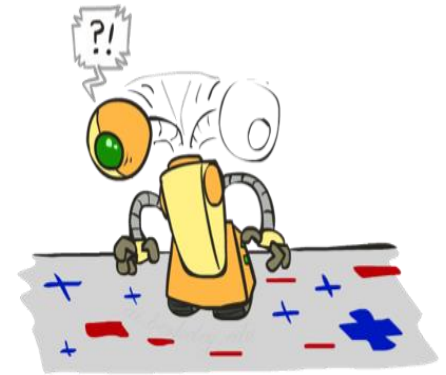
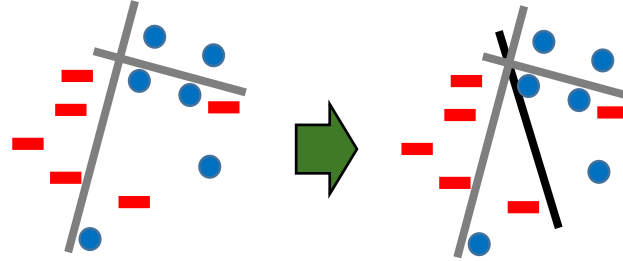
# Mejorando el Perceptrón



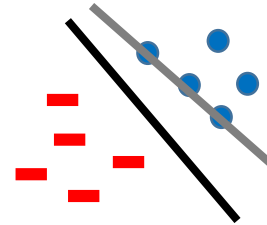
# Problemas con el Perceptrón

➤ **Ruido:** si los datos no son separables, pesos basura

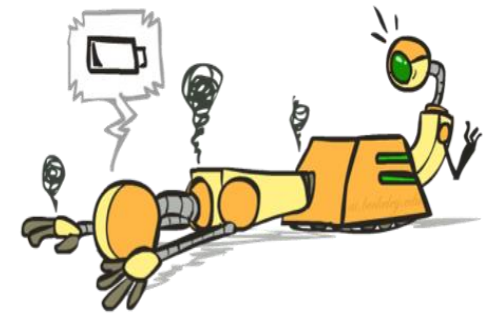
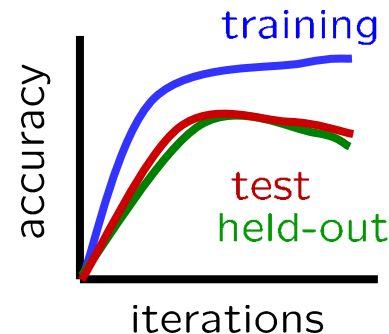
- Obtener medias de los vectores de pesos puede ayudar (averaged perceptron)



➤ **Generalización mediocre:** Encuentra “una” solución separable



➤ **Sobreajuste:** Los resultados en el test suelen ser peores



# Averaged Perceptron

➤ Imaginemos el ejercicio de (Deporte, Política, Tecnología)

– **Primera iteración** con pesos:  $w_D=(1,0,0,0,0,0,0)$   $w_P=(0,0,0,0,0,0,0)$   $w_T=(0,0,0,0,0,0,0)$

- Características  $f(x_1)$ : predicción errónea de  $y_D$  sabiendo que es  $y_P$ . **Recalcular pesos:**

$$w_D=(0,-1,0,-1,-1,0,0) \quad w_P=(1,1,0,1,1,0,0) \quad w_T=(0,0,0,0,0,0,0)$$

- Características  $f(x_2)$ : predicción correcta, seguimos con los mismos pesos:

$$w_D=(0,-1,0,-1,-1,0,0) \quad w_P=(1,1,0,1,1,0,0) \quad w_T=(0,0,0,0,0,0,0)$$

- Características  $f(x_3)$ : predicción errónea de  $y_P$  sabiendo que es  $y_D$ . **Recalcular pesos:**

$$w_D=(1,0,1,-1,0,0,0) \quad w_P=(0,0,-1,1,0,0,0) \quad w_T=(0,0,0,0,0,0,0)$$

– **Segunda iteración: Imaginemos que el algoritmo converge:**

- Características  $f(x_1)$ : predicción correcta, seguimos con los mismos pesos:

$$w_D=(1,0,1,-1,0,0,0) \quad w_P=(0,0,-1,1,0,0,0) \quad w_T=(0,0,0,0,0,0,0)$$

- Características  $f(x_2)$ : predicción correcta, seguimos con los mismos pesos:

$$w_D=(1,0,1,-1,0,0,0) \quad w_P=(0,0,-1,1,0,0,0) \quad w_T=(0,0,0,0,0,0,0)$$

- Características  $f(x_3)$ : predicción correcta, seguimos con los mismos pesos:

$$w_D=(1,0,1,-1,0,0,0) \quad w_P=(0,0,-1,1,0,0,0) \quad w_T=(0,0,0,0,0,0,0)$$

➤ **Los pesos que se usarían en el Test:**

$$W_D=((1+0+0+1+1+1+1)/7, (0-1-1+0+0+0+0)/7, (0+0+0+1+1+1+1)/7, \dots) = w_D=(0.71, -0.28, 0.57, \dots)$$

$$W_P=((0+1+1+0+0+0+0)/7, (0+1+1+0+0+0+0)/7, (0+0+0-1-1-1-1)/7, \dots) = w_P=(0.28, 0.28, -0.57, \dots) \quad w_T=(0,0,0,0,0,0,0)$$



# Averaged Perceptron

- Este proceso necesita guardar todos los  $w$  calculados en cada iteración.
  - **uso excesivo de memoria**
- Solución?
- El vector de pesos  $w_3$  es la suma de las actualizaciones anteriores:

$$\begin{aligned}w_0 &= (0, \dots, 0) \\w_1 &= w_0 + \Delta_1 = \Delta_1 \\w_2 &= w_1 + \Delta_2 = \Delta_1 + \Delta_2 \\w_3 &= w_2 + \Delta_3 = \Delta_1 + \Delta_2 + \Delta_3 \\&\dots\end{aligned}$$

- La media de los tres vectores podríamos escribir de la siguiente forma:

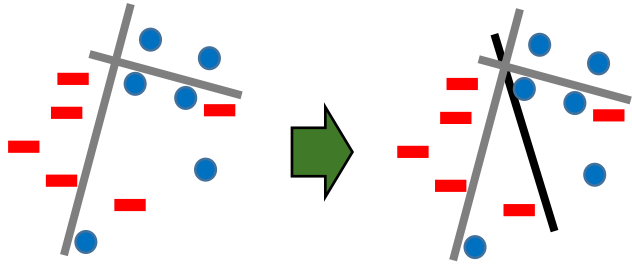
$$\frac{w_1 + w_2 + w_3}{3} = \frac{\Delta_1}{3} + \frac{\Delta_1 + \Delta_2}{3} + \frac{\Delta_1 + \Delta_2 + \Delta_3}{3} = \frac{3}{3}\Delta_1 + \frac{2}{3}\Delta_2 + \frac{1}{3}\Delta_3$$

# Averaged Perceptron

➤ El algoritmo:

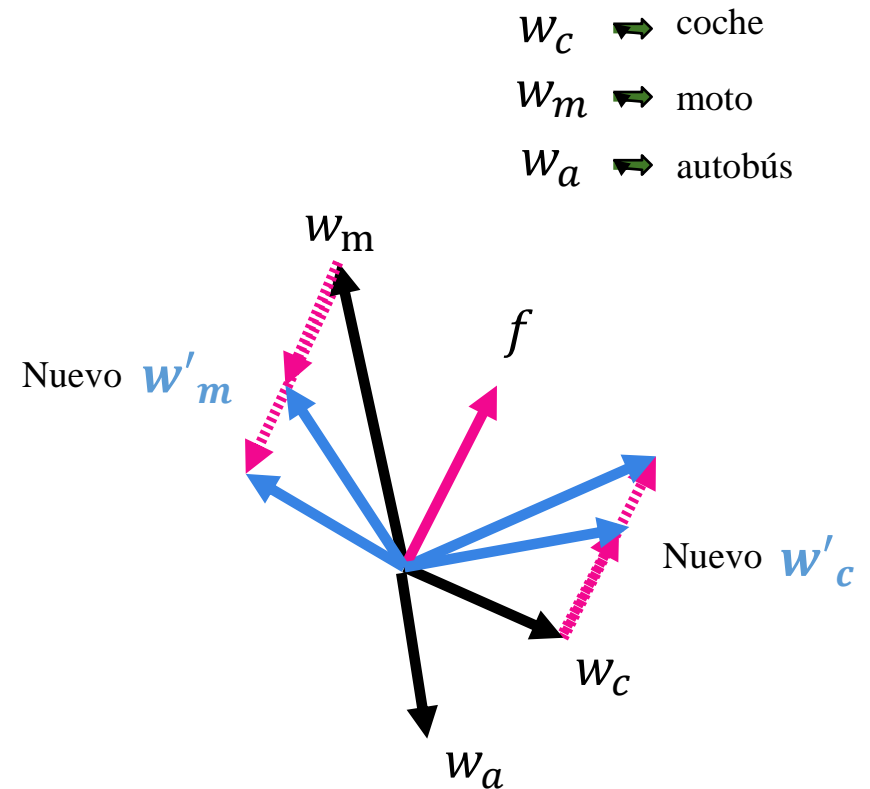
```
Inicializar los pesos  $w$  , bias  
Inicializar los pesos  $w_u$  , bias $_u$  #(media)  
cont=1  
Repetir hasta máxima iteración o converge:  
    Repetir para cada instancia (x,y):  
        Calcular clase predecida  
        Si mal precedido:  
             $w = w + y*x$  ,  $bias = bias + y$   
             $w_u = w_u + y*cont*x$  ,  $bias_u = bias_u + y*cont$   
        cont=cont+1  
  
Return  $w - \frac{w_u}{cont}$  ,  $bias - \frac{bias_u}{cont}$ 
```

# Arreglando el Perceptrón



- Otra solución? ajustar la actualización de pesos para mitigar estos efectos adversos
- MIRA (**M**argin **I**nfused **R**elaxed **A**lgorithm): elegir una actualización que arregle la errónea clasificación del ejemplo actual de entrenamiento...

... pero, que minimice el cambio sobre  $w$



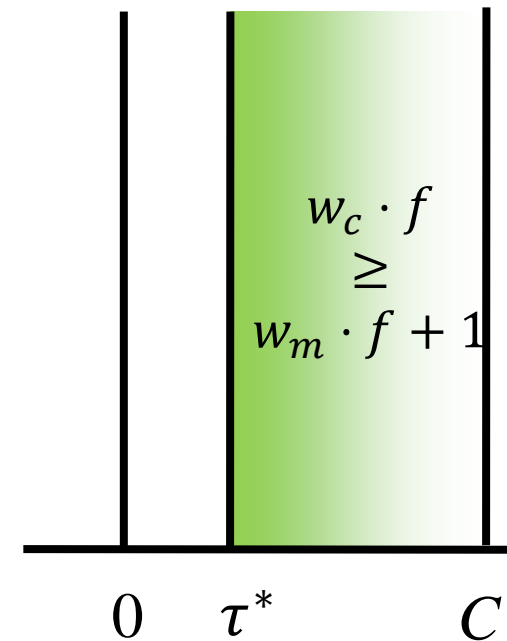
Me dice que es moto, pero sé que es coche! Sumo a coche y resto a moto, y los de autobús no los toco

$$\begin{aligned}w'_m &= w_m - \tau f(x) \\w'_c &= w_c + \tau f(x)\end{aligned}$$

# La actualización Máxima

- En la práctica hacer actualizaciones demasiado generosas no es apropiado
  - El ejemplo puede ser incorrectamente etiquetado
    - **Solución:** capar el valor máximo de  $\tau$  (tau) con alguna constante  $C$

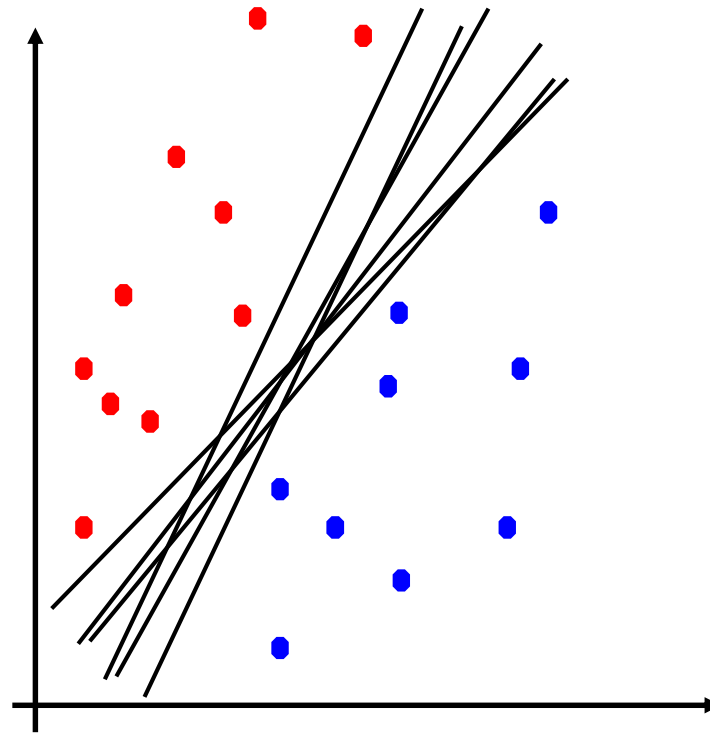
$$\tau = \min \left( \frac{(w_m - w_c) \cdot f + 1}{2f \cdot f}, C \right)$$



- Normalmente **converge antes** que el perceptrón
- Normalmente **es mejor, especialmente frente a ejemplos ruidosos**

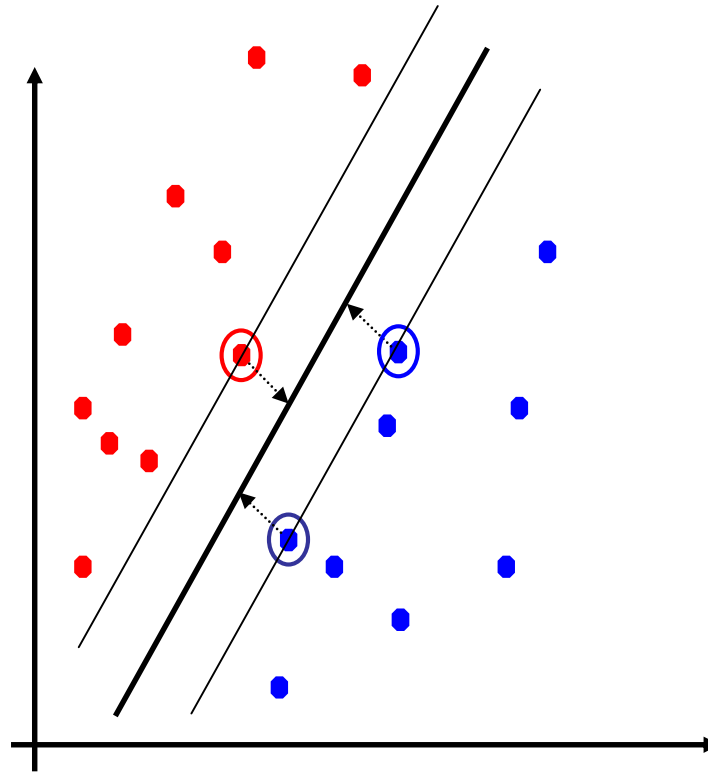
# Separadores lineales

➤ ¿Cual de estos separadores lineales es el óptimo?

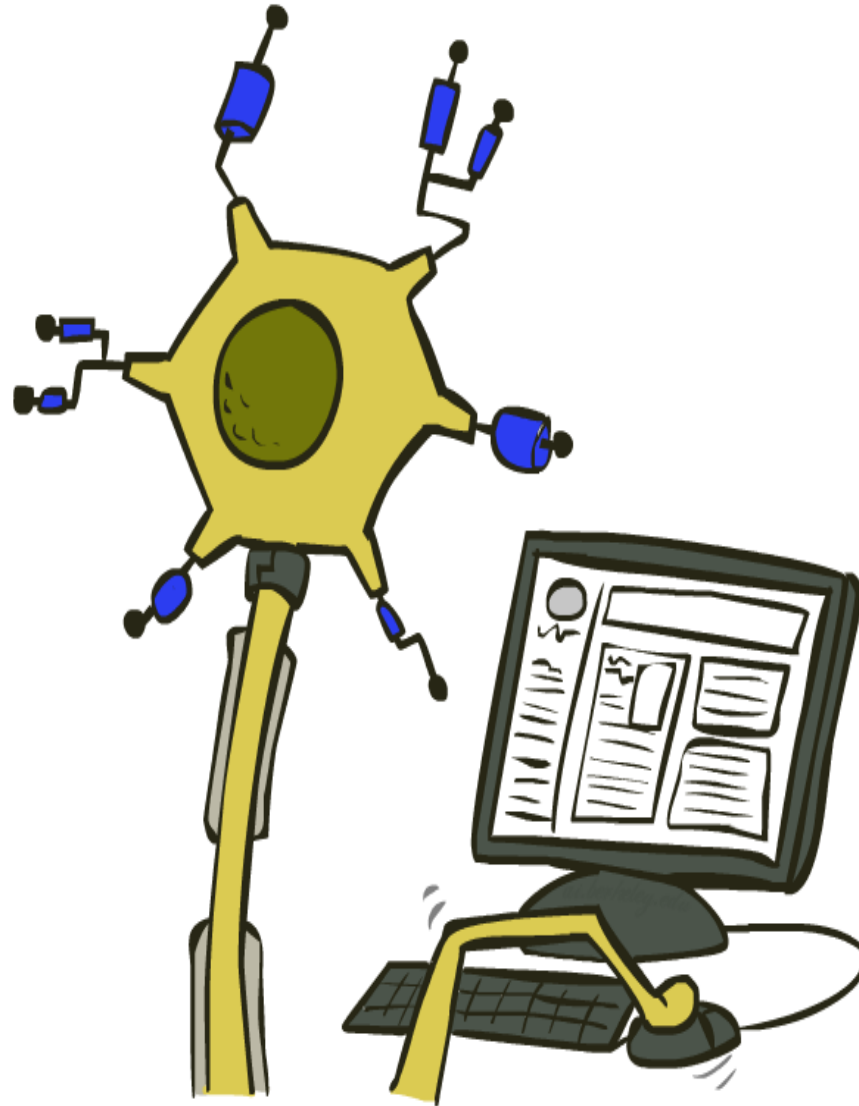


# Support Vector Machines (SVM)

- Support vector machines (SVMs); Maximizar el margen, es intuitivo
- Solo importan los support vectors; el resto de los ejemplos de entrenamiento se ignoran




# Búsqueda Web




# Búsqueda Web: Ranking basado en características

$x = \text{“Apple Computer”}$

$$f(x, \text{Apple}) = [0.3 \ 5 \ 0 \ 0 \ \dots]$$


---

$$f(x, \text{Apple Inc.}) = [0.8 \ 4 \ 2 \ 1 \ \dots]$$




# Perceptrón para Ranking

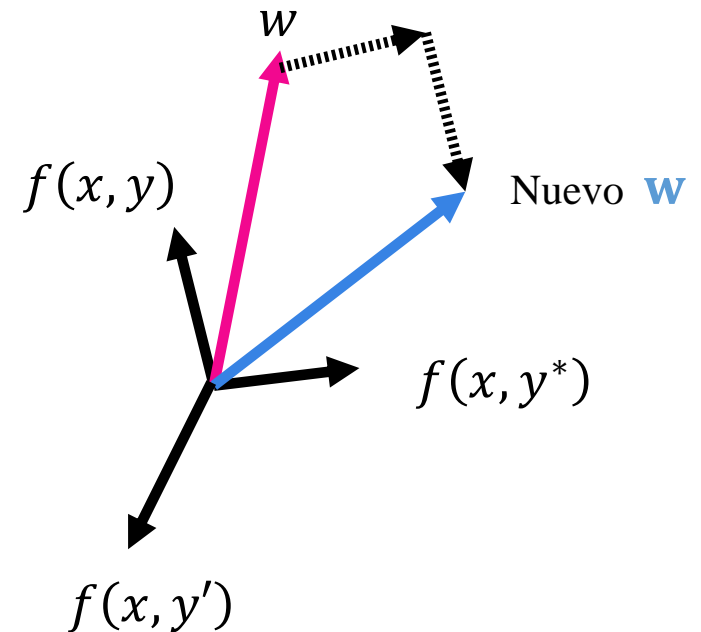
- Inputs:  $x$
- Candidatos:  $y$
- Muchos vectores de características:  $f(x, y)$
- Único vector de pesos:  $w$ 
  - predicción:

$$y = \operatorname{argmax}_y w \cdot f(x, y)$$

- actualización (si incorrecto):

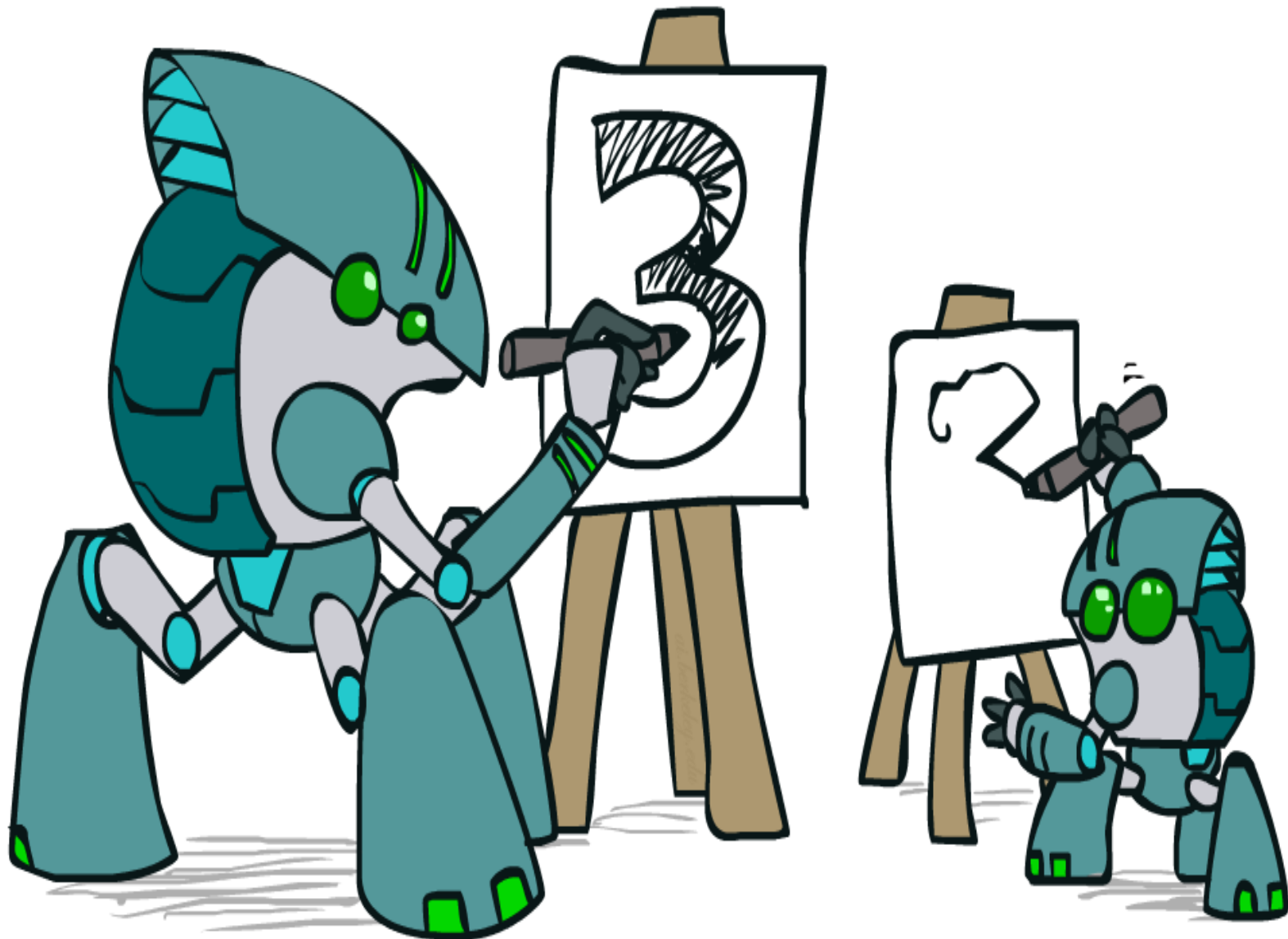
$$\mathbf{w} = w + f(x, y^*) - f(x, y)$$

Me dice que es  $y$ , pero se parece más a  $y^*$ !  
Sumo a  $w$  el valor de  $y^*$  y resto  $y$



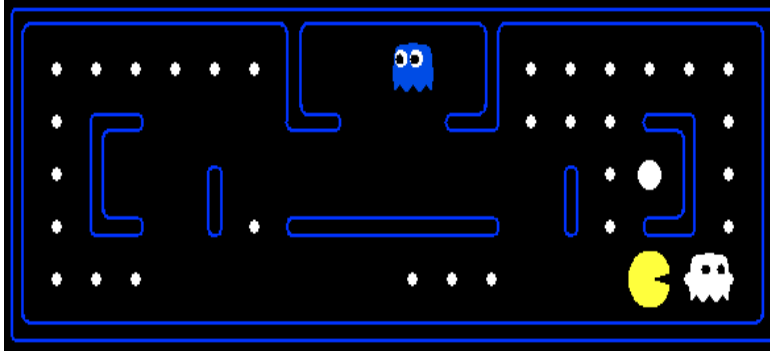
- Hay solo un vector de pesos porque se aplica el mismo a todos los ejemplos, dado que no hay una clase correcta o una incorrecta sino cuanto se parece cada ejemplo a la solución deseada

# Aprendices



# ¡Aprendiz de Pacman!

- Los ejemplos son estados:  $S$



- Los candidatos son pares:  $(S, a)$
- Las acciones “Correctas”: son tomadas por un experto
- Las características se calculan por cada par  $(S, a)$ :  $f(S, a)$
- Único vector de pesos:  $\mathbf{w}$
- El score del par  $(S, a)$  se obtiene:  $\mathbf{w} \cdot \mathbf{f}(S, a)$



$$\forall a \neq a^*, \\ \mathbf{w} \cdot \mathbf{f}(a^*) > \mathbf{w} \cdot \mathbf{f}(a)$$

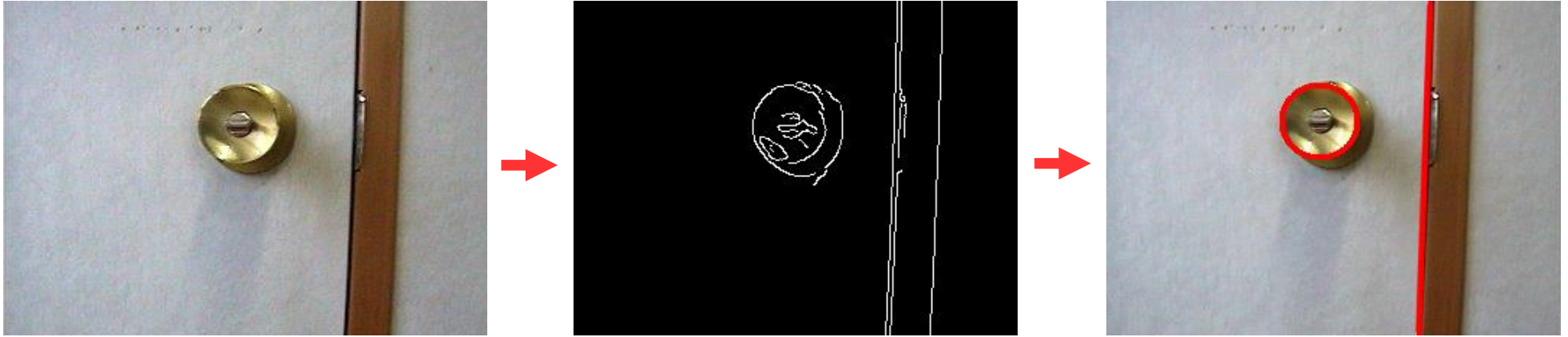
$$y = \operatorname{argmax}_y \mathbf{w} \cdot \mathbf{f}(S, a)$$

$\mathbf{w} = \mathbf{w} - \mathbf{f}(S, a) \Rightarrow$  acción predecida

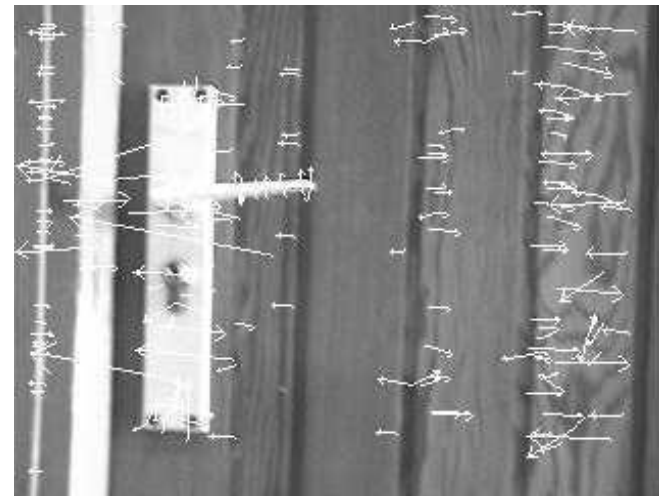
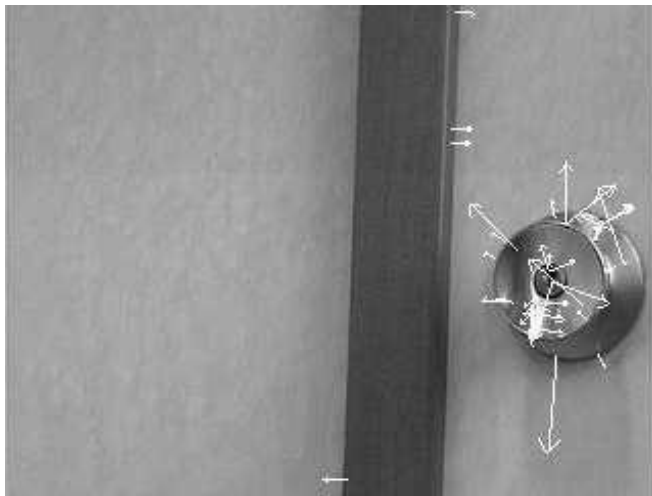
$\mathbf{w} = \mathbf{w} + \mathbf{f}(S, a^*) \Rightarrow$  acción correcta

# Métodos tradicionales para Visión por computador

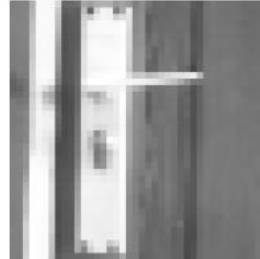
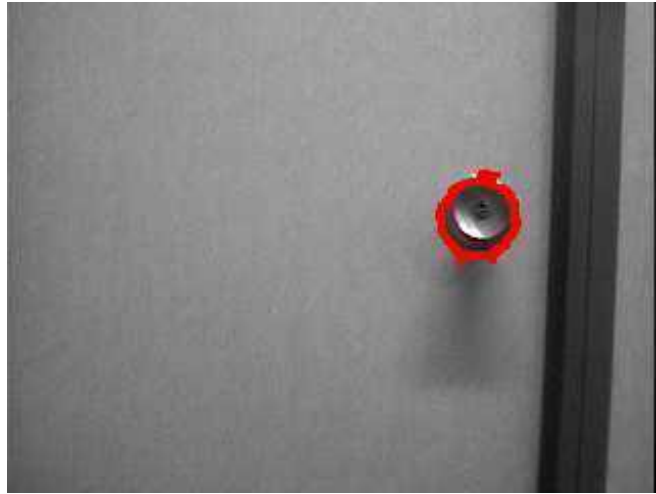
## ➤ Detección de puertas



## ➤ Extracción de características con SIFT, SURF, USURF ... (se almacenan en un vector)



# Métodos tradicionales para Visión por computador



**= ?**



BD de manillas

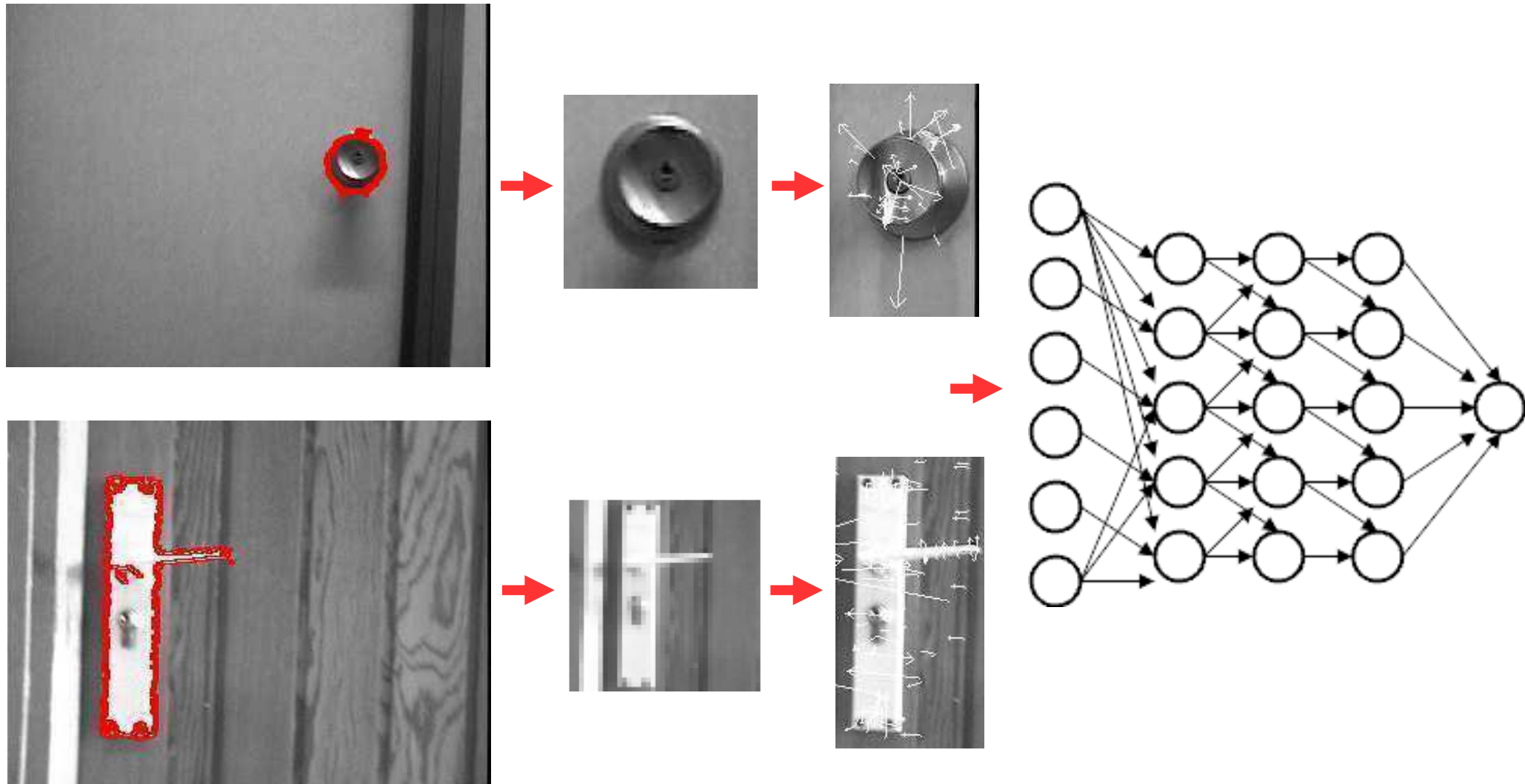
# Métodos tradicionales para Visión por computador

- Mismo programa anterior: detección de paso de peatones, cruce y bicicletas





# Más allá del perceptron



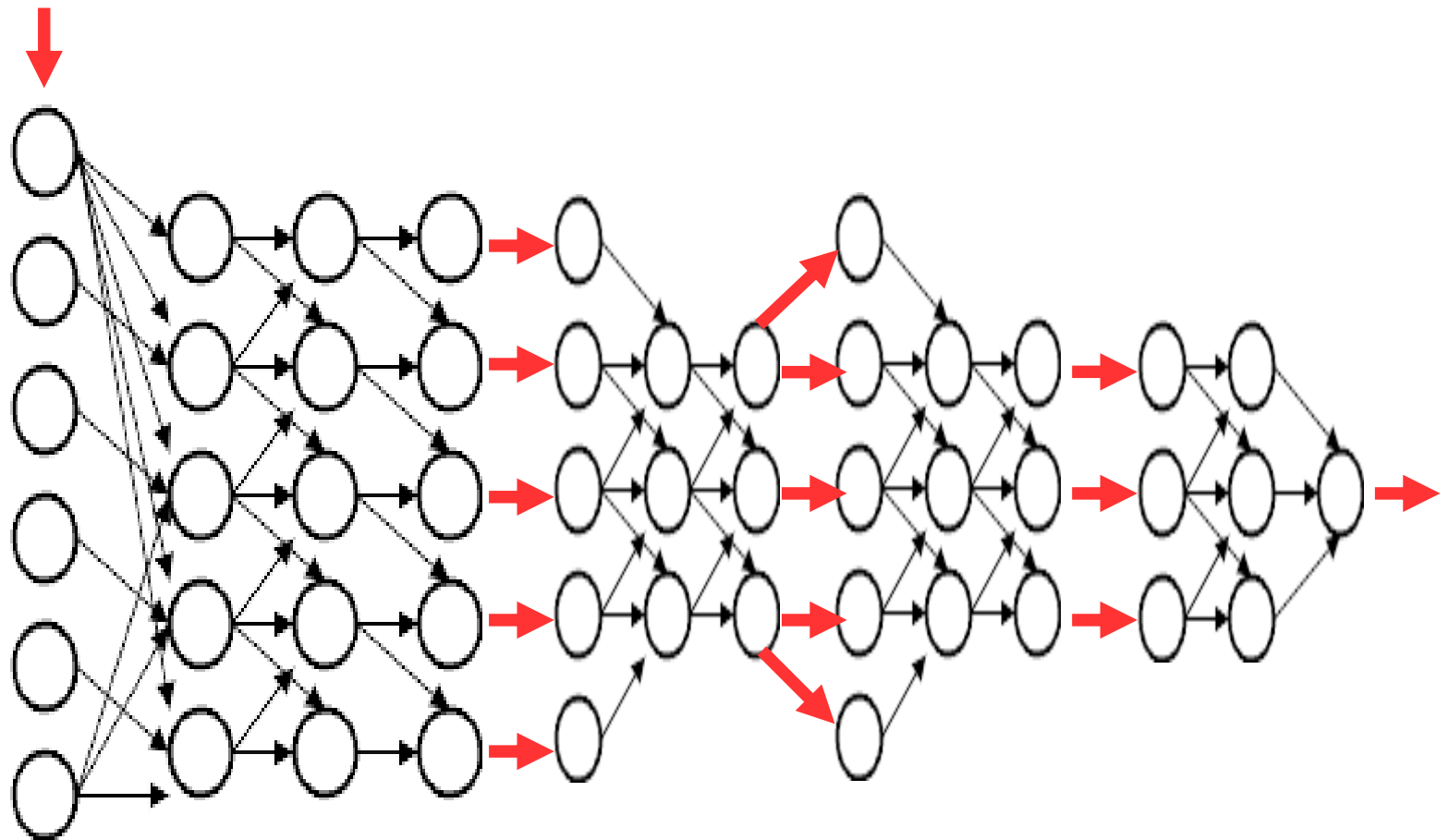
# Más allá del perceptron

10 x 10 x 3

Entrada (10 x 10)



					185	187	209	58	7
					14	125	233	201	98
253	144	120	251	41	147	204			
67	100	32	241	23	165	30			
209	118	124	27	58	201	79			
210	236	105	169	19	218	156			
35	178	199	197	4	14	218			
115	104	34	111	19	196				
32	69	231	203	74					



Capa 1  
(input)

Capa 2  
(feature  
extraction)

Capa 3  
(object  
detection)

Capa 4  
(output)



# Ejercicio Perceptrón Multiclase

**Klases** = {ingles, castellano, francés}

$$f(x_{\text{ingles}}) = (0,0,1,1,0,0)$$

$$f(x_{\text{castellano}}) = (0,0,1,0,1,0)$$

$$f(x_{\text{francés}}) = (0,1,1,1,0,1)$$

Teniendo en cuenta que los vectores de pesos son  $(0,0,0,0,0,0)$  para todas las clases y el valor de **BIAS** es **1**, es decir,  $(\text{bias}, f_0, f_1, f_2, f_3, f_4, f_5)$ . En caso de empate, la clase será **Francés**:

- converge el perceptrón?
- Si converge, cuantas iteraciones ha necesitado?
- Cuales son los valores finales de los pesos?