**Machine Learning**

**Aprendizaje basado en ejemplos.**

*In whitch we describe agents that can improve their behavior through diligent study of their own experiences.*

**¿Por qué queremos que un agente aprenda?**

Si es posible un mejor diseño, **¿por qué no lo diseñamos mejor desde el principio?**

* No es posible anticipar todas las situaciones
* No se puede anticipar todos los cambios
* El programador no tiene ni idea de cómo programar una solución

**Componentes a aprender**

* Mapeo de condiciones de un estado a acciones.
* Inferir propiedades relevantes del mundo desde la secuencia de percepciones.
* Como impactan las acciones.
* Utilidad de los estados.
* Información acerca de las preferencias de las acciones.

**Representación del conocimiento.**

* **Representación factorizada** como entradas (un vector de atributos)
* **Aprendizaje inductivo** es aprender una función general desde casos específicos.

**Feedback. ¿es necesario?**

3 tipos de feedback:

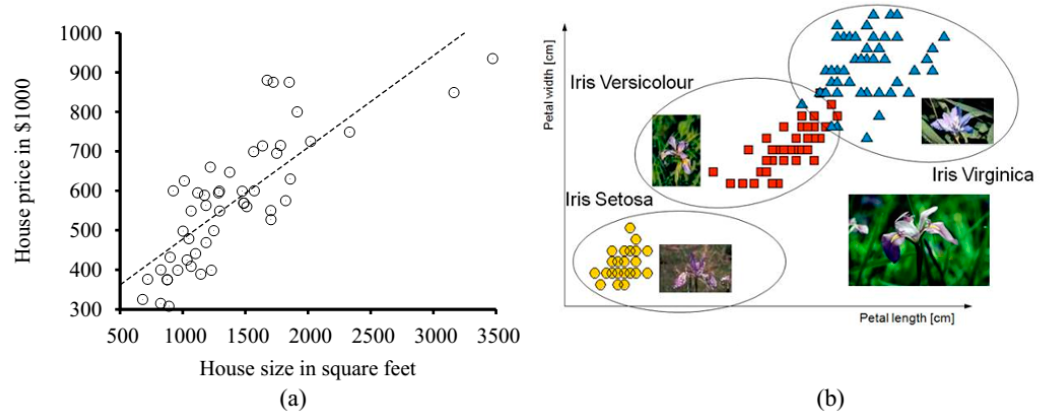
1. **Aprendizaje no supervisado:** aprende sin feedback. El algoritmo clasifica datos pero no sabe qué significan. Lo más común son algoritmos de **clustering**.
2. **Aprendizaje por refuerzo:** el agente aprende a través de una serie de refuerzos (recompensas o castigos)
3. **Aprendizaje supervisado:** el agente cuenta con entradas y las salidas esperadas y aprende una función para realizar el mapeo.
4. **Aprendizaje semi-supervisado** es un gris entre 1 y 3. Se debe al ruido o a la falta de etiquetas.

**Aprendizaje supervisado**

Dado un **Conjunto de entrenamiento** donde cada es generado mediante , donde es desconocida; se trata de encontrar una hipótesisque aproxime a .

El aprendizaje consiste en buscar una hipótesis que se ajuste bien a los datos.

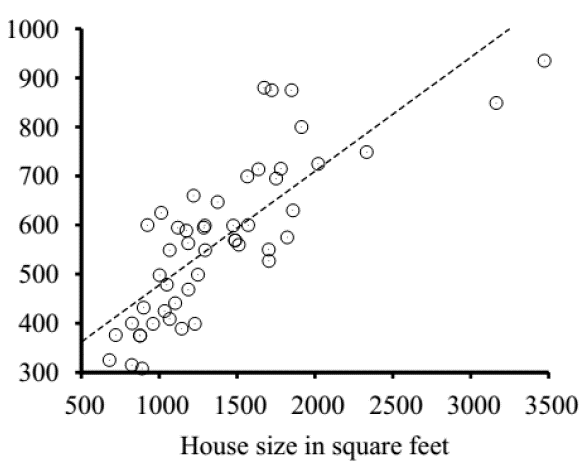
X es un vector de atributos. Y puede ser un valor o un vector.



* *x e y pueden adoptar cualquier tipo de datos, es un vector de atributos.*
* *Un* ***conjunto de test*** *es usado para medir la* ***precisión*** *de una hipótesis.*
* *Decimos que una hipótesis* ***generaliza*** *bien, cuando predice correctamente con entradas no conocidas.*
* *Cuando la salida es un* ***valor de un conjunto finito****, el problema es llamado* ***clasificación****.*
* *Cuando la salida es un* ***número*** *el problema es llamado* ***regresión****.*

**Regresión lineal univariada**

Se conoce como regresión a la tarea de encontrar un h que minimice el **error** o **costo.**



• Tiene la forma de una recta: *y = w1 x + w0*

• usamos *w* porque vamos a pensar los coeficientes como pesos (weights)

• vamos a usar *w* y Theta indistintamente

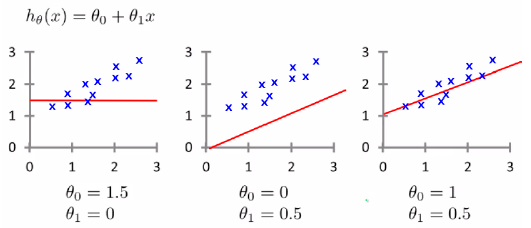
• **w** es el vector definido por [w0, w1]

• hw(x) = w1 x + w0

**Función de costo “J”**

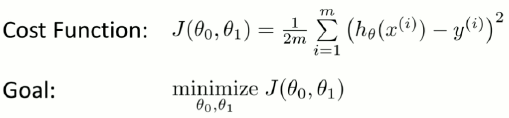
Es el error que tiene las hipótesis.

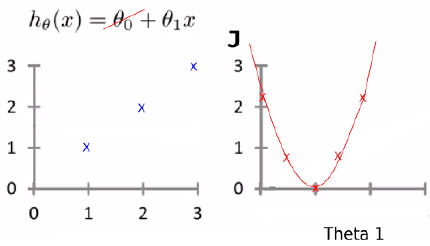
¿Cómo elegir **W**?



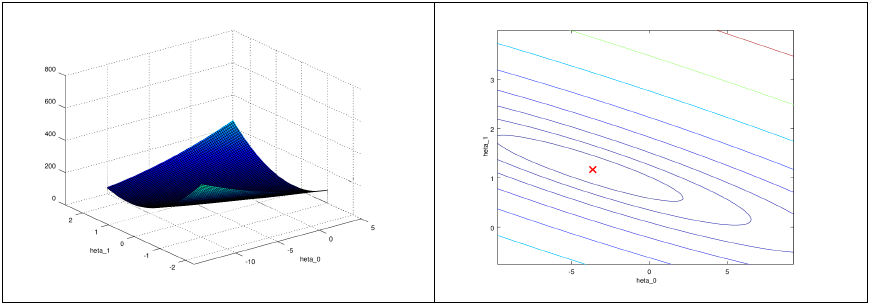
La última es mejor, se llama error cuadrático medio.

Error cuadrático medio:





La hipótesis de recta que mejor predice es la que tiene el menor error (theta). Sería el mínimo.

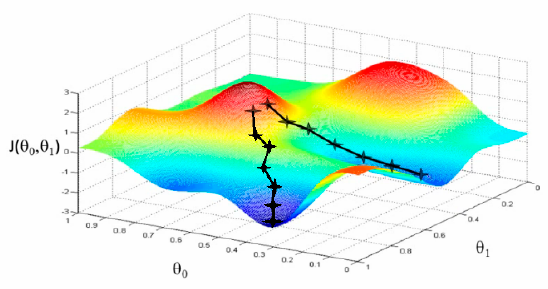


La función de costo es estrictamente convexa, con lo cual, solo tiene un mínimo global

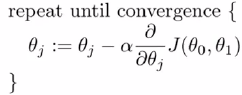
**Algoritmo para encontrar mínimo: Descenso por el gradiente (GD)**

El gradiente indica en qué dirección hay que moverse para encontrar el mínimo. Usa las derivadas para saber indicar la dirección.

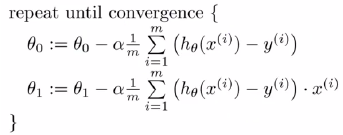
Se parte de un valor aleatorio para los parámetros y se van adaptando los parámetros tratando de reducir hasta encontrar un mínimo.



Función para calcular cuánto hay que modificar theta:



* Esta regla de actualización se aplica sobre todos los thetas simultáneamente. Modificar cuando ya se tenga calculado cuánto hay que modificar.
* **Alpha** se denomina **learning rate**. Indica cuánto hay que modificar el conocimiento.
  + Si es demasiado chico, GD puede ser lento.
  + Si es demasiado grande, GD puede no converger o divergir.



Este algoritmo también se conoce como **Batch gradient descent**: en cada paso se usan **todos** los ejemplos de entrenamiento.

**Regresión lineal multivariada**

Es la misma idea, pero **x** es un vector



Si creamos una entrada *xj, 0* que siempre valga 1 podemos expresar **h** como el producto punto o producto de matrices.



La regla de actualización es:



**Cosas a tener en cuenta al aplicar los algoritmos:**

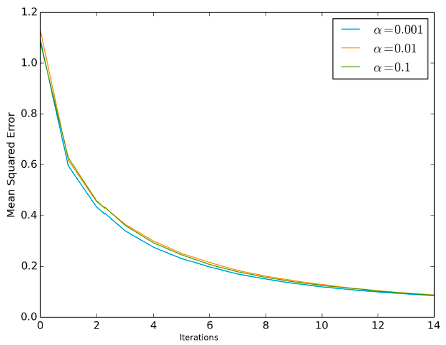
Feature scaling o normalización de datos

Se aplica cuando tenemos multiples atributos de distintas magnitudes. Hace que gradient descent (y otros algoritmos) converja más rápido.

* Una posibilidad es **(**
  + Todos los valores quedan entre 0 y 1.
* Otra posibilidad para cuando trabajamos en papel es

Eligiendo un valor de alpha adecuado

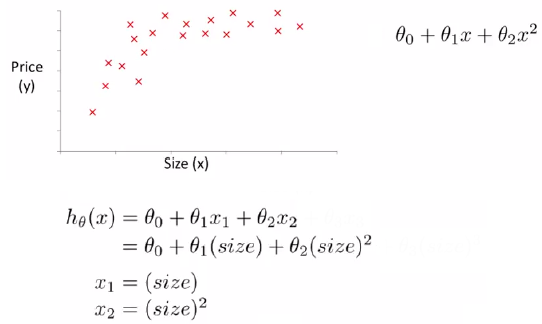
* Una alternativa es dibujar **Jw** respecto de la cantidad de iteraciones y elegir.



* Si sería creciente es porque el alpha es muy grande.
* Si alpha es muy chico, la función decrecería lentamente.

Regresión polinómica

• Puede ser deseable usar como hipótesis un polinomio de grado mayor a 1.



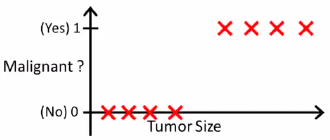
**Problemas de clasificación**

Clasificación: Etiquetar las entradas. Las etiquetas son finitas.

Ejemplos de etiquetas:

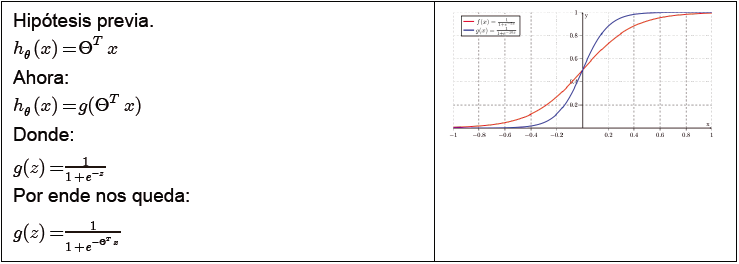
* Email: Spam / Not Spam
* Tumores: Malignos / Benignos
* Clientes: Malos pagadores / Buenos Pagadores

**¿Es posible aplicar regresión lineal para resolver estos problemas? Nop**



Vamos a usar etiquetas 0 y 1.

**Regresión logística**



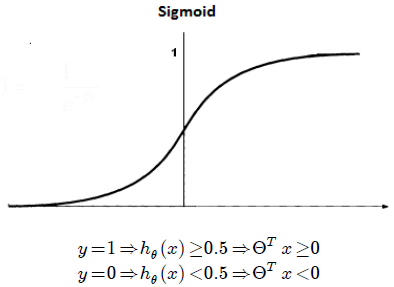
* g(z) se denomina **función logistica** o **función sigmoidea**
* h(x) devuelve la probabilidad de que **y = 1**

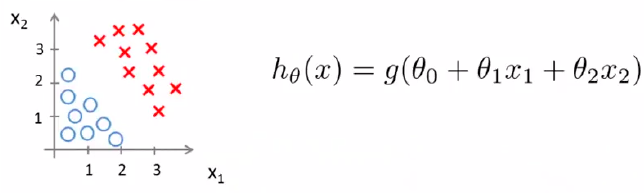
**Frontera de decisión**

Línea que separa las clasificaciones.

X tiene que ser mayor a cero para que y sea 1.

X tiene que ser menor a cero para que y sea 0.

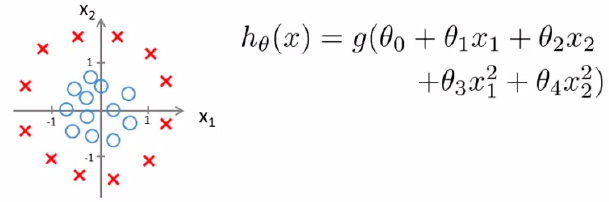




La frontera de decisión es la región donde h(x) = 0.5.

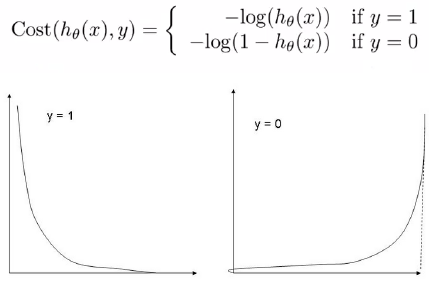
Es la frontera donde la predicción cambia de valor.

**Frontera de decisión no lineal**

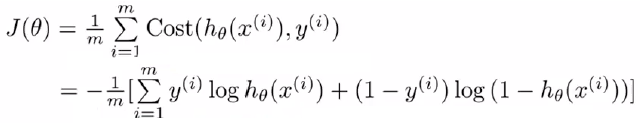


**Función de costo**

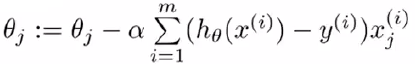
* Si usamos error cuadrático como función de costo, J(w) será no convexa. (no tiene un mínimo global)
* En cambio, vamos a usar:



Si juntamos las dos gráficas:

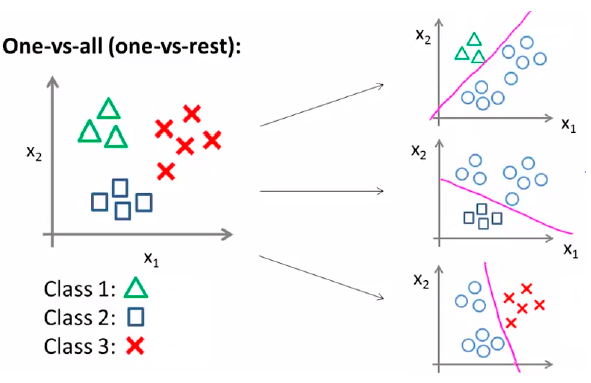


La regla de actualización para descenso por el gradiente queda:



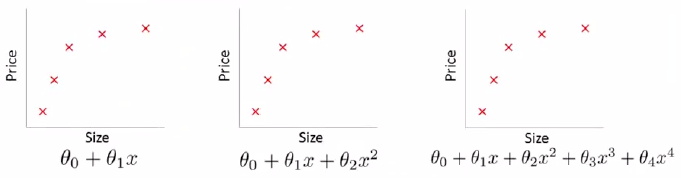
**Clasificación multiclase**

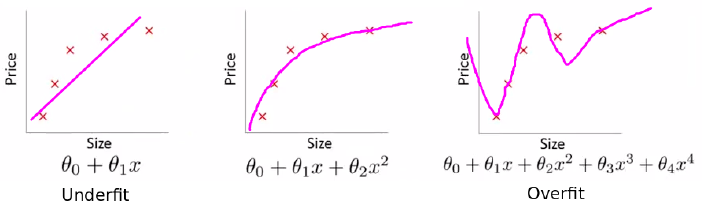
La Regresión Logística no permite muchas clasificaciones. Se puede separar en grupos de a dos.



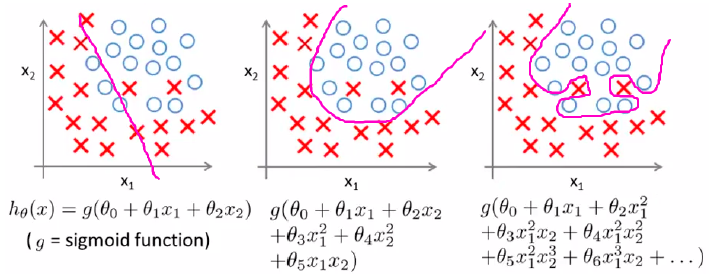
* Entrenamos un clasificador binario por cada clase.
* Para hacer una predicción:
  + Obtenemos la probabilidad de y = 1 por cada predictor.
  + Elegimos la clase con la probabilidad más alta.
* La normalización de los features sigue siendo necesaria.

**Sobre-entrenamiento**





Decimos que el modelo está **sobreentrenado** cuando la hipótesis se ajusta muy bien al dataset de entrenamiento, pero falla en **generalizar** nuevos ejemplos.



**¿Cómo podemos preveer el sobreentrenamiento?**

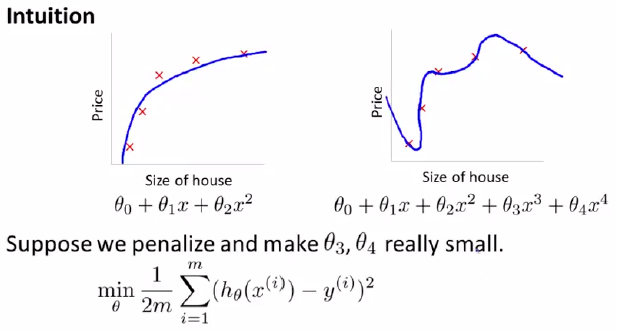
1. Reducir el número de features:

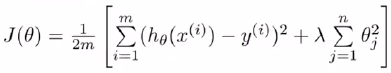
* Selección manual o Algoritmo de selección.
* Puede ser que todos los features contribuyan en la predicción.

1. Regularización:

* Mantener todos los features pero reducir el valor de los parámetros.
* Funciona bien cuando todos los features contribuyen en la predicción.

**Regularización**

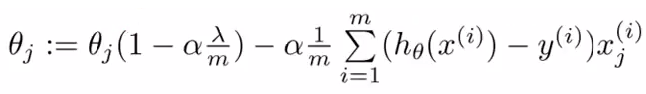


Costo: 

**lambda** es el parámetro de regularización. Establece un tradeoff entre "complejidad de la hipótesis" y "mantener bajo el error".

¿Cómo funciona?

* A valores bajos de lambda, la hipótesis es compleja y el error va a ser bajo (sobreentrenamiento)
* A valores altos de lambda, la hipótesis es simple y el error va a ser alto (bajo entrenamiento)

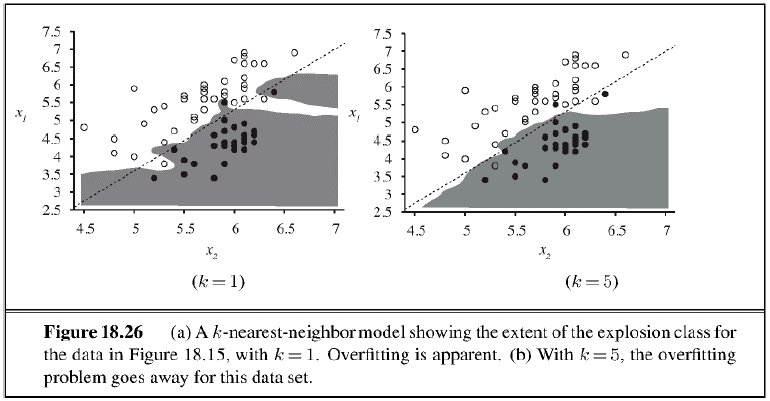


**Modelos no paramétricos**

* Modelo paramétrico: conjunto fijo de **parámetros**
* Modelo no-paramétrico: la cantidad de parámetros es ilimitada; cada ejemplo es un parámetro
* Se lo conoce también como **Aprendizaje basado en instancia** o **Aprendizaje basado en memoria**
* Una posible implementación es la **tabla de búsqueda**

**Modelo de los vecinos cercanos (KNN)**

El resultado se basa en los k vecinos más cercanos



Con un k muy chico puede haber sobreentrenamiento.

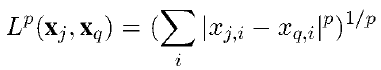
Con k muy alto predice siempre lo mismo.

¿Es rápido?

* Hay que calcular los más cercanos 🡪 alguna función matemática.
  + Tiene que calcular para cada punto 🡪 lento.

**¿Cómo medir la distancia?**

**Distancia de Minkowski**



* Si p=2 se conoce como distancia Euclidea y si p=1 como distancia de Manhattan.
* **Es importante la normalización de los datos**
* Una posibilidad es (Xi - media) / desvio
* Otra es (Xi - min) / (max - min)
* **Los algoritmos knn funcionan bien con muchos ejemplos y pocas dimensiones**
* Variantes para mejorar la velocidad:
  + **k-d tree**
  + **Locality- sensitive hashing**

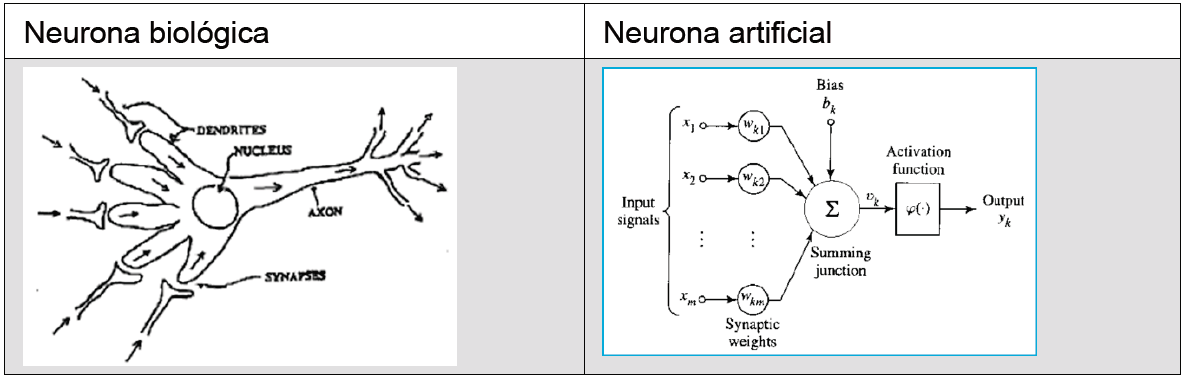
**También se puede hacer regresión no paramétrica**

**Redes neuronales artificiales**

Es un modelo paramétrico

Una red neuronal es una gran cantidad de unidades simples de cómputo interconectadas entre sí, que tratan de asemejarse al cerebro en dos aspectos:

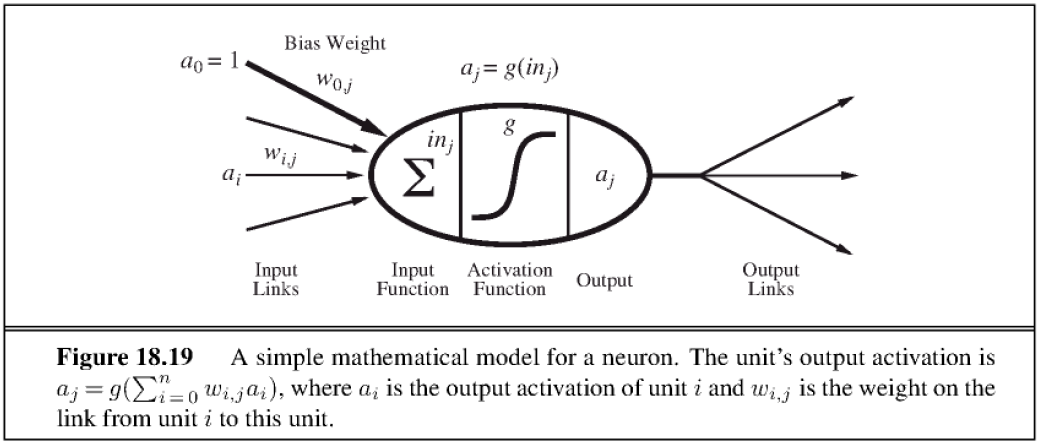
1. El conocimiento es adquirido por la red desde el ambiente a través de un proceso de aprendizaje
2. El conocimiento adquirido es almacenado en las conexiones entre neuronas



Una neurona es una regresión lineal multivariada.

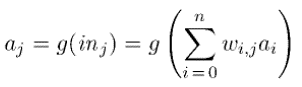
Una vez que se tiene el valor de la regresión, para determinar cuál es la salida, pasa por una función de activación (como la función logística).

**El modelo de McCulloch y Pitts data de 1943.**



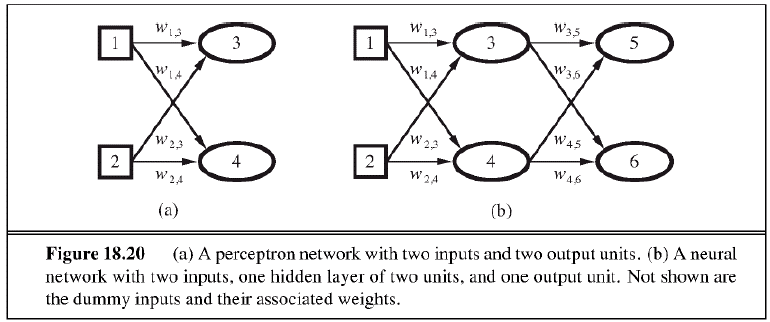
**Estructura de una red neuronal**

* Una red está compuesta por **unidades** (o neuronas) conectadas por **links**.
* Un link tiene asociado un peso y es usado para propagar la activación desde una neurona a otra.
* La activación de una neurona aj se calcula:



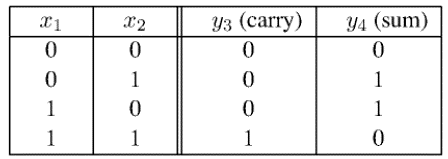
Donde:

* g es la función de activacion
* w es el peso sináptico
* ai es la entrada relacionada
* Si **g** es la función threshold, la neurona se denomina **perceptron**
* Hay redes **propagación hacia adelante** y **recurrentes**.
* Las redes **feed-forward** se pueden organizar en **capas** o **layers**.
* Si la red es multicapa, las capas que no se conectan con la salida se denominan **ocultas**.

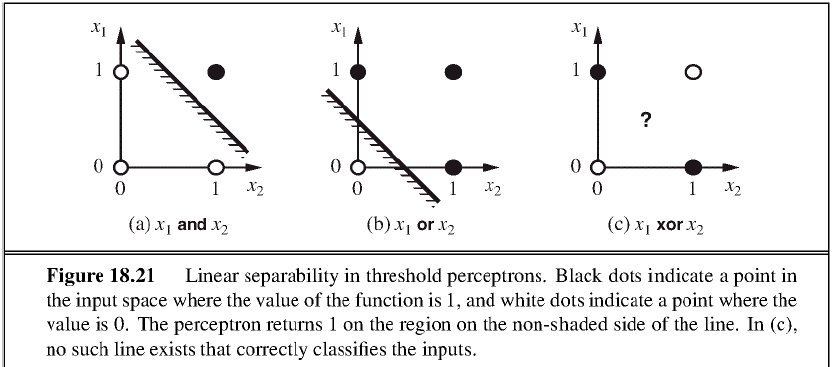


**Redes de una capa.**

* Si hay *m* salidas, entonces hay *m* redes distintas, cada una con su propio entrenamiento
* Cada neurona es un clasificador lineal



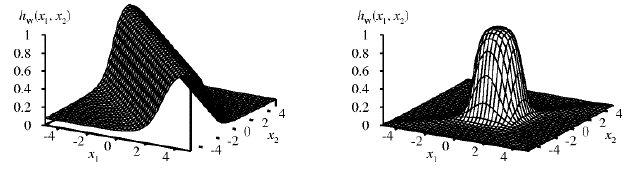
No puede hacer un XOR



Esto era antes.. Solución: conectar las salidas a las entradas de otras

**Redes multicapa. MLP**

Mediante este tipo de redes es posible representar cualquier tipo de función, incluso funciones no lineales.



Las salidas de una capa son entradas a otra capa.

