

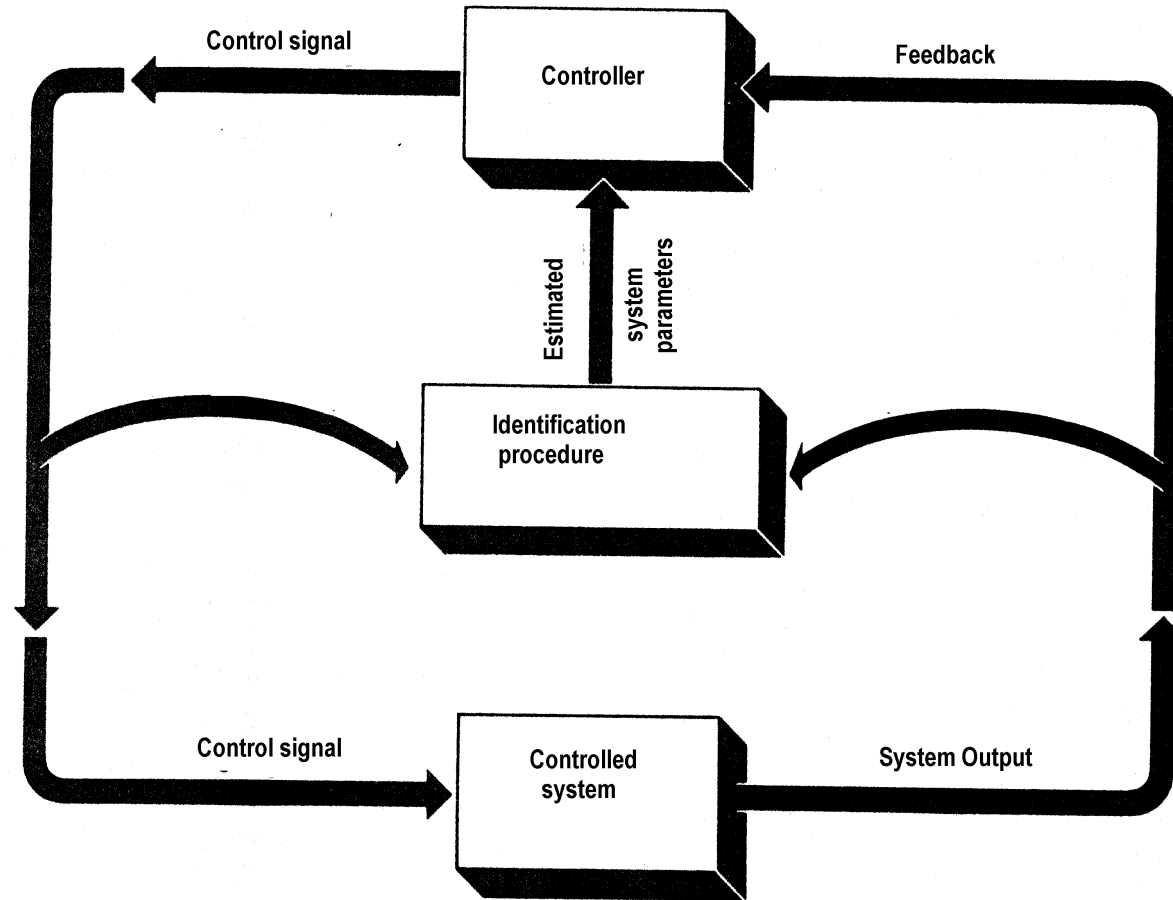
Teoría del Cerebro y Neuroinformática

Redes Adaptativas

Maestría en Ciencias en
Computación



Procedimientos de identificación y control adaptativo



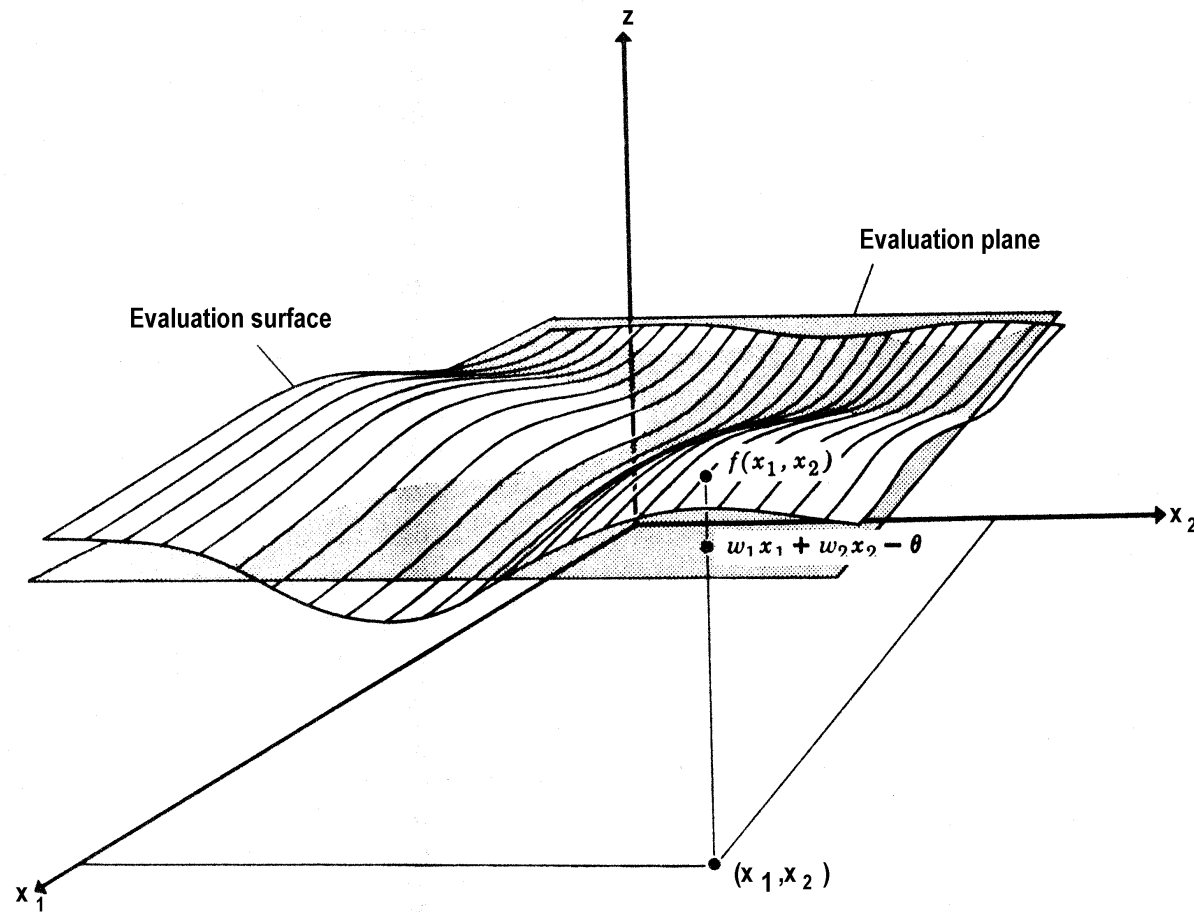
Jugador de damas. Samuel 1959

- Programación de una computadora para jugar a las damas.
- Tomaba en cuenta varios movimientos por delante: Estrategia Max-min
- Podaba el árbol de búsqueda
- Daba a la computadora una manera de determinar el valor de las posiciones del tablero.

Jugador de damas. Samuel 1959

- Sin saber cómo evaluamos un tablero, podemos al menos estar seguros de que su valor depende de cosas como
 - el número de piezas que cada jugador tiene
 - el número de reinas
 - Equilibrio del tablero
 - movilidad
 - control del centro.
- A estos podemos asignar números precisos.
- Eligió 16 parámetros que contribuían a la evaluación del tablero.
- Evaluación = $f(x_1, x_2, \dots, x_{16})$ ¿Qué es f ?

Aproximación de una superficie de evaluación mediante un (hiper)plano



Aproximación de una superficie de evaluación mediante un (hiper)plano

- La estrategia de Samuel en 1959 fue
 - además de reducir el "lookahead"
- adivinar que
 - la función de evaluación era aproximadamente lineal.
- ... utilizando una aproximación de un hiperplano a la evaluación real para jugar un buen juego:
 - $z = w_1 x_1 + \dots + w_{16} x_{16} - \theta$ (una aproximación lineal)
- para algunas opciones de los 16 pesos w_1, \dots, w_{16} , y θ .
- Al decidir cuál es mejor de dos tableros la constante θ es irrelevante
- por lo que sólo hay 16 parametros que encontrar para obtener la mejor aproximación lineal.

La regla de aprendizaje

- Oriente un hiperplano de evaluación en el espacio de dimensión 17:
- Con base en el ajuste actual de los pesos, la computadora elige un movimiento que parece conducir a tableros de alto valor para ella.
- Si al cabo de un tiempo se da cuenta de que el juego parece ir mal, ya que sobrevaloró el tablero que eligió, entonces reducirá los parámetros que contribuyeron positivamente mientras que aumentará los que no lo hicieron.
- Barto en "Adaptive Critics" para el aprendizaje por refuerzo
 - sustituye el refuerzo por "refuerzo esperado"

Modelos clásicos para redes adaptativas

- Los dos esquemas de aprendizaje clásicos para neuronas formales del tipo McCulloch-Pitts ($\sum_i w_i x_i \geq \theta$):
- Aprendizaje Hebbiano - Predisposiciones Amplificadoras
 - El esquema de Hebb en “La organización del comportamiento” (1949)
 - reforzar una sinapsis cuya actividad coincide con el disparo de la neurona postsináptica
- El Perceptrón - Aprendiendo con un Maestro (Rosenblatt 1962)
 - fortalecer una sinapsis activa si la neurona eferente no dispara cuando debería haberlo hecho;
 - debilitar una sinapsis activa si la neurona eferente se dispara cuando no debería haber disparado.

El esquema de Hebb

- Hebb (p. 62, 1949):
 - Cuando un axón de la célula A está lo suficientemente cerca como para excitar una célula B y repetidamente o persistentemente participa en su disparo, se produce algún proceso de crecimiento o cambio metabólico en una o ambas células, de manera que la eficiencia de A como una de las células que disparan B, se incrementa.
- Hebb (1949) desarrolló un modelo multinivel de percepción y aprendizaje, en el cual las **unidades de pensamiento** fueron codificadas por **conjuntos de células**, cada una definida por actividad reverberante en un conjunto de vías neurales cerradas.
- La esencia de la sinapsis de Hebb es aumentar el acoplamiento entre las células coactivas para que puedan ser enlazadas en ensambles en crecimiento.
- Hebb desarrolló hipótesis similares en un nivel jerárquico superior de organización, vinculando eventos cognitivos y su recuerdo en secuencias de fases, series de activaciones organizadas temporalmente de ensambles de células.

Regla de Hebb

- La formalización de la regla de Hebb más simple es incrementar w_{ij} por: $\Delta w_{ij} = k y_i x_j$



- donde la sinapsis w_{ij} conecta una neurona presináptica con una tasa de disparo x_j a una neurona postsináptica con tasa de disparo y_i .

Regla de Hebb

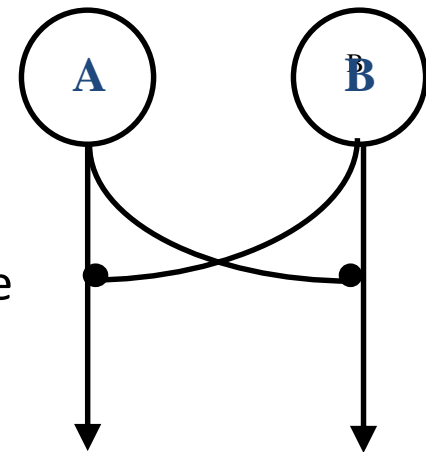
- Peter Milner señaló el problema de la saturación
- von der Malsburg 1973 (modelado del desarrollo de detectores de orientación de bordes en la corteza visual del gato [Hubel-Wiesel: V1 simple cells]) aumentó las sinapsis de tipo Hebb con
 - una forma de normalización para detener la saturación de todas las sinapsis

$$\sum w_i = \text{Constante}$$

- inhibición lateral para evitar que la primer “experiencia” “asuma” todos los “circuitos de aprendizaje”: impide que las células cercanas adquieran el mismo patrón permitiendo así que el conjunto de neuronas “abarque el espacio de características”

Inhibición lateral entre

- La idea es utilizar la competencia entre las neuronas, para que si una neurona se convierte en experta en responder a un patrón, inhiba a otras neuronas de hacerlo.
- Si la celda A se dispara mejor que la celda B para una orientación determinada θ , entonces se dispara más que B y se reduce la respuesta de B por inhibición lateral
- para que A se adapte más hacia θ y
- B se adaptará menos, y la tendencia continuará con cada presentación de θ .
- El conjunto final de pesos de entrada a la neurona depende tanto
 - en el ajuste inicial de los pesos, y
 - en el patrón de agrupamiento del conjunto de estímulos al que está expuesto.
- Capturando las estadísticas del conjunto de patrones

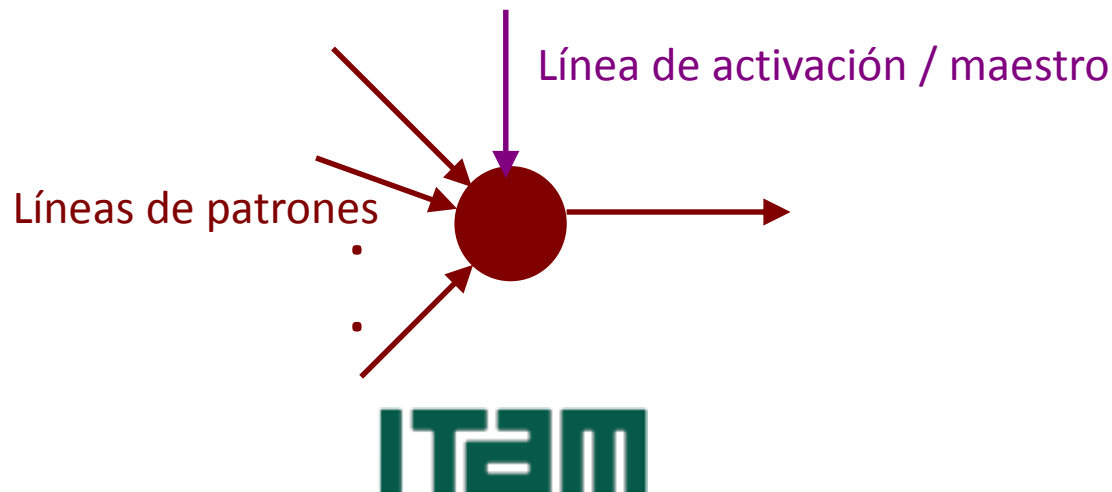


Aprendizaje Hebbiano no supervisado

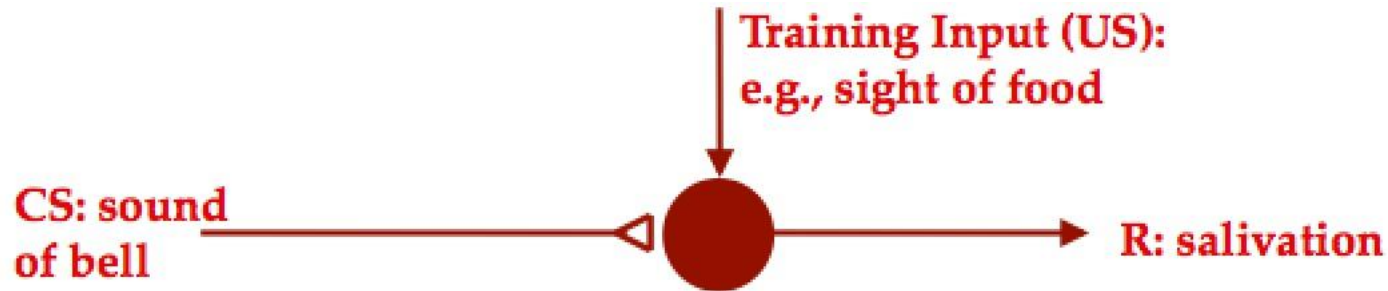
- Tiende a agudizar la predisposición de una neurona **“sin un maestro”**
- el disparo de la neurona se hace mejor y más correlacionado con un conjunto de patrones de estímulos

El aprendizaje Hebbiano puede ser supervisado

- El aprendizaje supervisado de Hebbian está basado en tener una línea de activación separada de las líneas de patrón con sinapsis entrenables.
- Usa la línea de activación para ordenar a una neurona a disparar, asociando así el disparo de la neurona con los patrones de entrada utilizados en las ocasiones en que fue activado.
- [Esto se relaciona con la idea de la memoria asociativa.]



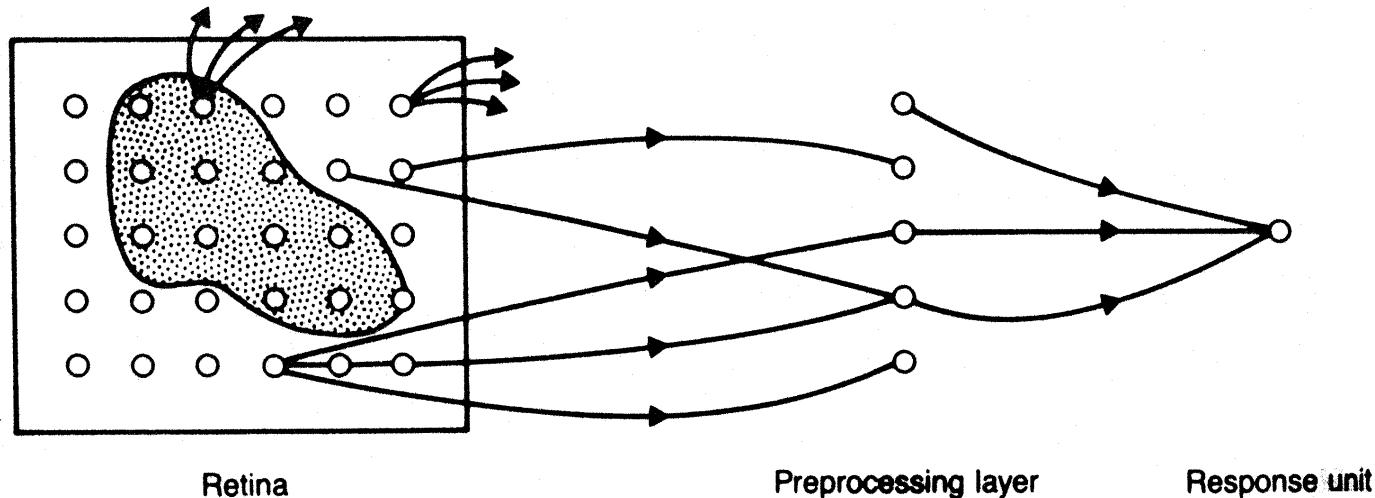
Aprendizaje Hebbiano supervisado está estrechamente relacionado con condicionamiento Pavloviano



- La respuesta de la célula que se está entrenando corresponde a la respuesta condicionada e incondicionada (R),
- la entrada de entrenamiento corresponde al estímulo incondicionado (US), y
- la entrada entrenable corresponde al estímulo condicionado (CS).
- Puesto que los US por sí mismos pueden disparar R, mientras que el CS inicialmente puede ser incapaz de disparar R,
- la actividad conjunta de US y CS crea las condiciones para que la regla de Hebb fortalezca la sinapsis US→R,
- de modo que finalmente el CS es suficiente para obtener una respuesta.

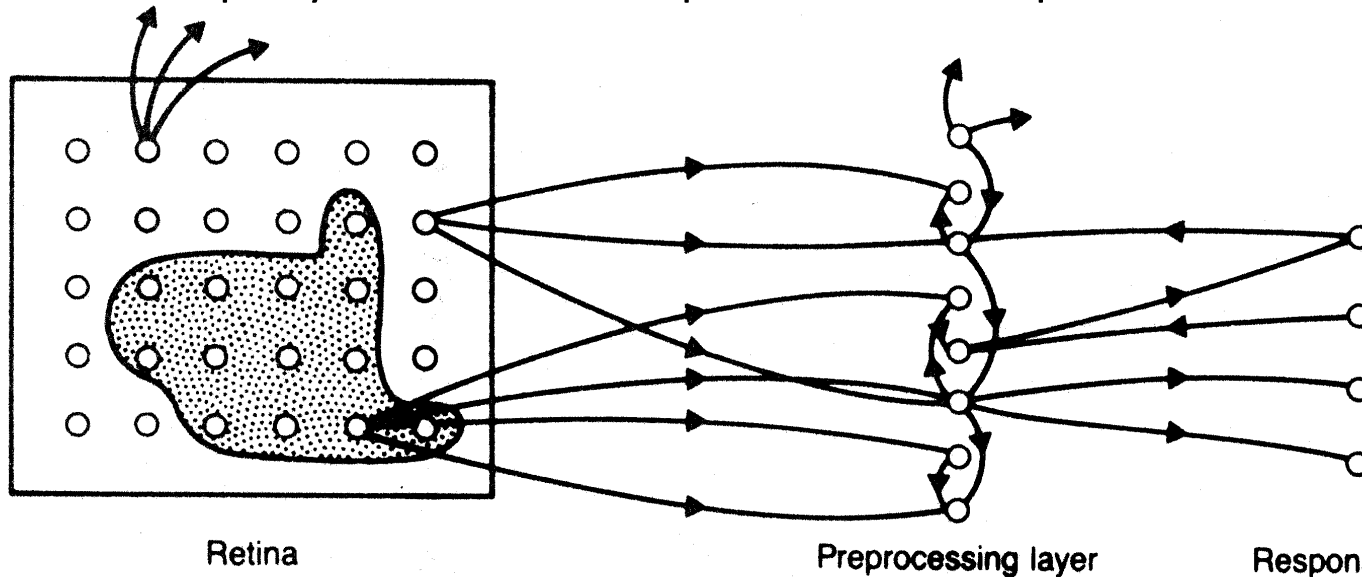
Clasificación de patrones por neuronas

- Rosenblatt (1958) consideró explícitamente el problema del reconocimiento de patrones cuando un maestro es esencial,
 - por ejemplo colocar b , B , b y \mathcal{B} en la misma categoría.
- Introdujo los Perceptrones - redes neuronales que cambian con "experiencia" usando una regla de corrección de errores diseñada para cambiar los pesos de cada unidad de respuesta cuando da respuestas erróneas a estímulos presentados a la red.
- Un Perceptrón simple no tiene ciclos en la red, y sólo los pesos de las unidades de salida pueden cambiar:



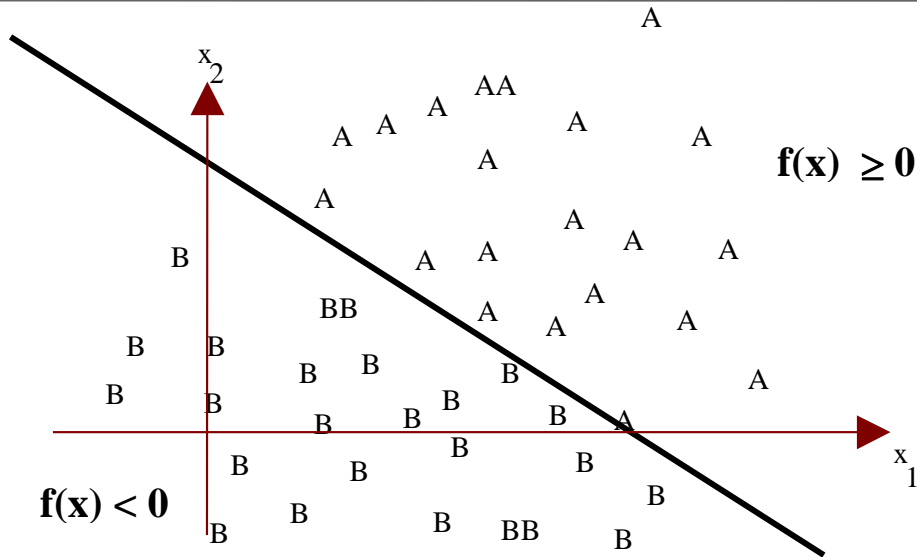
Perceptrones simples vs. generales

- Las unidades asociativas no están interconectadas, y así el perceptrón simple no tiene memoria a corto plazo.
- Si las unidades están acopladas de forma cruzada, la red puede tener varias capas y ciclos de una capa "anterior" a "posterior".



- Se discutirá la retropropagación: extender las técnicas de perceptrón a las redes de propagación hacia delante multicapa sin ciclos mediante "asignación de crédito" para "capas ocultas" entre la entrada y la salida.

Linealmente separables



Dos categorías de patrones son linealmente separables si sus miembros pueden ser separados por una línea o (más generalmente) un hiperplano.

- Una función lineal de la forma
- $f(x) = w_1x_1 + w_2x_2 + \dots + w_dx_d + w_{d+1}$ ($w_{d+1} = -\theta$)
- Es un calificador de patrones de dos categorías
- $f(x) = 0 \iff w_1x_1 + w_2x_2 + \dots + w_dx_d + w_{d+1} = \theta$
- Da un hiperplano como una superficie de decisión
- El entrenamiento consiste en ajustar los coeficientes $(w_1, w_2, \dots, w_d, w_{d+1})$ de forma que la superficie de decisión produzca una separación aceptable de las dos clases.

Estrategia general de entrenamiento

- Elija un "conjunto representativo de patrones"
- Utilice un conjunto de entrenamiento para ajustar los pesos sinápticos usando una regla de aprendizaje.
- Con los pesos ajustados después del entrenamiento, evalúe los pesos anotando las tasas de éxito en un conjunto de pruebas diferente al conjunto de entrenamiento.
- Rosenblatt (1958) proporcionó un esquema de aprendizaje con la propiedad que
 - si los patrones del conjunto de entrenamiento (es decir, un conjunto de vectores de características, cada uno clasificado con 0 o 1) pueden ser separados por una selección de pesos y umbral,
 - entonces el esquema producirá finalmente un ajuste satisfactorio de los pesos.

Regla de aprendizaje del perceptrón

- La regla de aprendizaje del perceptrón más conocida
 - fortalece una sinapsis activa si la neurona eferente no dispara, cuando debería haber disparado, y
 - debilita una sinapsis activa si la neurona se dispara cuando no debería haberlo hecho:

$$\Delta w_{ij} = k (Y_i - y_i) x_j$$

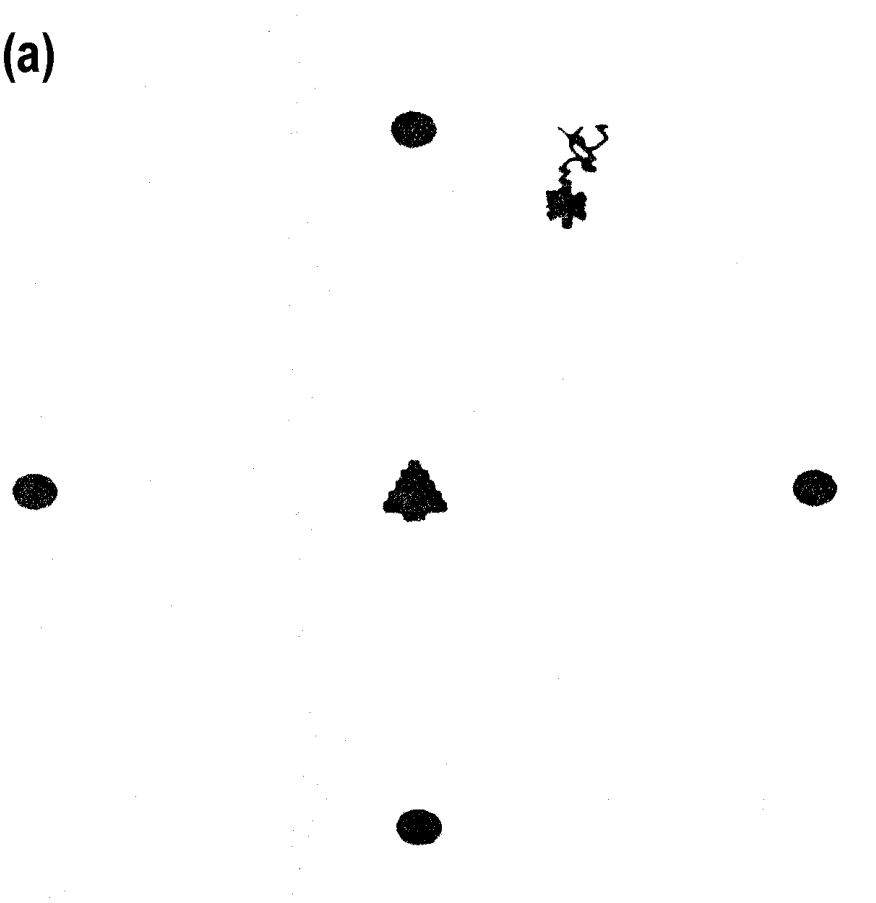
- Como antes, la sinapsis w_{ij} conecta una neurona con tasa de disparo x_j a una neurona con la tasa de disparo y_i , pero ahora
- Y_i es la salida "correcta" suministrada por el "maestro".
 - (Esto es similar al modelo de control adaptativo de Widrow-Hoff [1960]).
- La regla cambia la respuesta a x_j en la dirección correcta:
 - Si la salida es correcta, $Y_i = y_i$ y no hay cambio, $\Delta w_{ij} = 0$.
 - Si la salida es demasiado pequeña, entonces $Y_i - y_i > 0$, y el cambio en w_{ij} añadirá $\Delta w_{ij} = k (Y_i - y_i) x_j$ cuando $x_j > 0$ a la respuesta de la unidad de salida a (x_1, \dots, x_d) .
 - Si la salida es demasiado grande, Δw_{ij} disminuirá la respuesta de la unidad de salida.

El teorema de convergencia del perceptrón

- Así, $w + \Delta w$ clasifica el patrón de entrada x "más correctamente" que w .
- Desafortunadamente, al clasificar x "más correctamente" corremos el riesgo de clasificar otro patrón "menos correctamente".
- Sin embargo, el teorema de convergencia del perceptrón muestra que el procedimiento de Rosenblatt no produce un va y ven sin fin, sino que eventualmente convergerá a un conjunto correcto de pesos si este existe, aunque quizás después de muchas iteraciones, a través del conjunto de patrones de entrenamiento.

El robot, cuatro puntos de referencia (N, E, W, S) y la meta (el árbol "en la cima de la colina")

- La "función de pago" z (a) es una cantidad escalar que aumenta a medida que el robot se acerca a la meta:
 - Piense en z como "altura en una colina" para que se busque el objetivo de "escalar la colina"

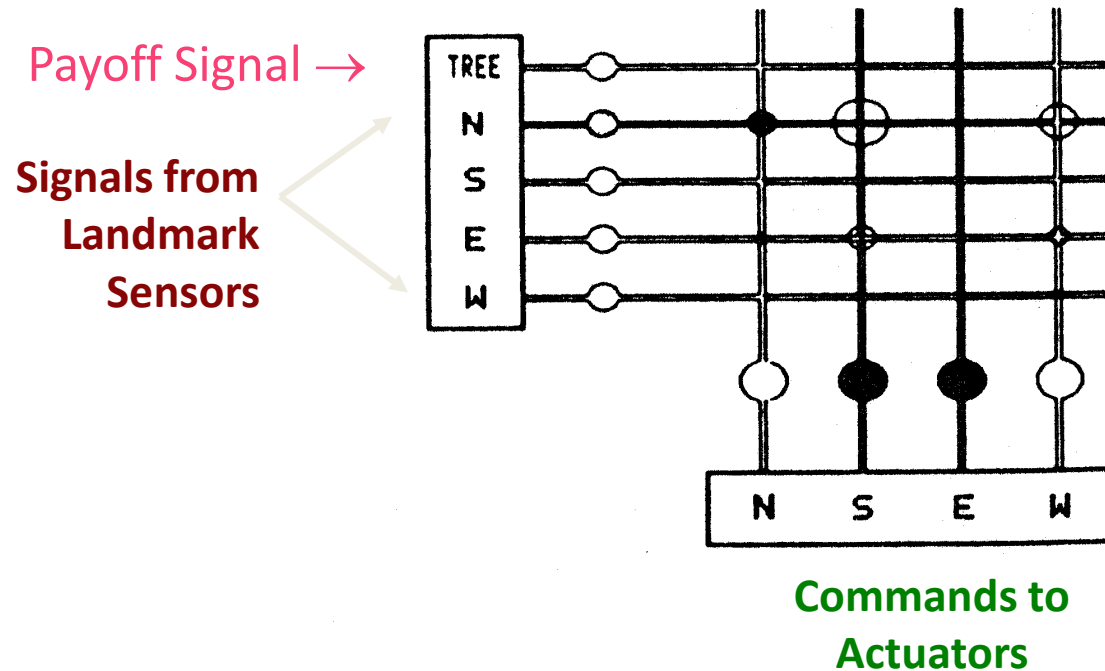


Escalar la colina y aprender referencias

- "Escalar la colina en la niebla":
- En cualquier instante t , el robot toma un solo paso en la dirección $i(t)$, moviéndose desde una posición con compensación $z(t)$ a una con recompensa $z(t + 1)$.
- Si $z(t + 1) - z(t) > 0$, entonces el siguiente paso del robot está en la misma dirección, $i(t + 1) = i(t)$, con alta probabilidad.
- Si $z(t + 1) - z(t) < 0$, entonces $i(t + 1)$ se elige aleatoriamente.
- cf. Quimiotaxis bacteriana: mecanismo de correr y girar.
- Aprendizaje de referencia
 - Barto y Sutton muestran cómo equipar a nuestro robot con un sistema nervioso simple (¡cuatro neuronas!) Que puede ser entrenado para usar "pistas olfativas" de los cuatro puntos de referencia para mejorar su dirección de búsqueda con la experiencia.

El controlador adaptativo de cuatro neuronas

Problema de aprendizaje: Encontrar los pesos sinápticos correctos



- Nota: La señal de pago no proporciona señales de error explícitas a cada comando del actuador.
- Piense en $z(t) - z(t-1)$ como un refuerzo (positivo o negativo).

Escalar la colina en el espacio de pesos

- La red puede "aprender" valores apropiados para los pesos.
- Aquí elevamos el escalar la colina a un nivel más abstracto:
- en lugar de trepar en el espacio físico - eligiendo una dirección otra vez si toma al robot cuesta arriba - ahora llevamos a cabo el **escalar la colina en el espacio de pesos**
- En cada paso, los pesos se ajustan de tal manera que mejoran el rendimiento de la red.
- La entrada z , sin ningún peso asociado propio, es el "maestro" utilizado para ajustar las ponderaciones que vinculan las entradas sensoriales a las salidas motrices.

Aprendizaje de referencias

- Sea la salida $s_j(t) = \sum_i w_{ji}(t)x_i(t)$ (1)
- Puesto que los pesos actuales w pueden no ser correctos, añadimos un término de ruido, ajustando la salida del elemento j en el tiempo t a

$$y_j(t) = 1 \text{ if } s(t) + \text{RUIDO}_j(t) > 0, \text{ else } 0 \text{ (2)}$$

- donde cada $\text{RUIDO}_j(t)$ es una variable aleatoria normalmente distribuida con media cero (cada una con la misma varianza).
- Los pesos cambian según:
$$\Delta w_{ji}(t) = c[z(t) - z(t-1)]y_j(t-1)x_i(t-1) \text{ (3)}$$
- donde c es una "tasa de aprendizaje" positiva.
- Piense en $z(t) - z(t-1)$ como refuerzo (positivo o negativo).

$$\Delta w_{ji}(t) = c[z(t)-z(t-1)]y_j(t-1)x_i(t-1)$$

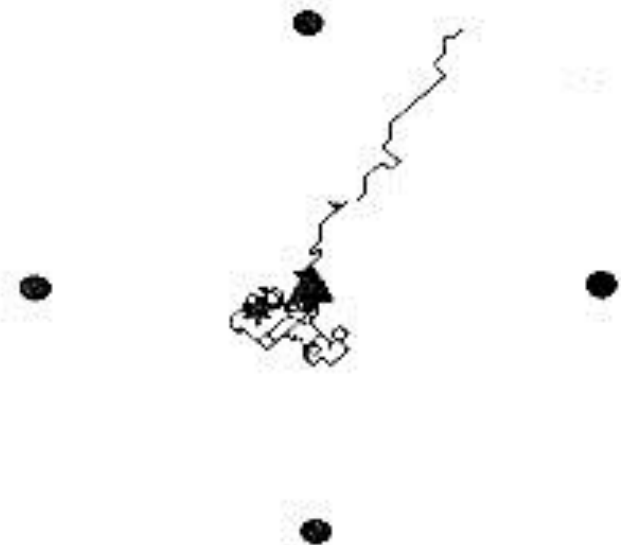
- w_{ji} sólo cambiará si
 - se produce un movimiento j ($y_j(t-1) > 0$) y
 - el "robot" está cerca de la referencia i ($x_i(t-1) > 0$)
- Entonces cambiará en la dirección de $z(t) - z(t-1)$.
- Otra vez vea $z(t)$ como "altura de una colina":
 - w_{ji} aumenta y un movimiento j es más probable si z aumenta (el "robot" se mueve hacia arriba); mientras
 - w_{ji} disminuye y un movimiento j es menos probable si el robot se mueve hacia abajo.
- Los w están cambiando en un espacio abstracto de 16 dimensiones de los ajustes de peso.

Antes y después de aprender

(a)



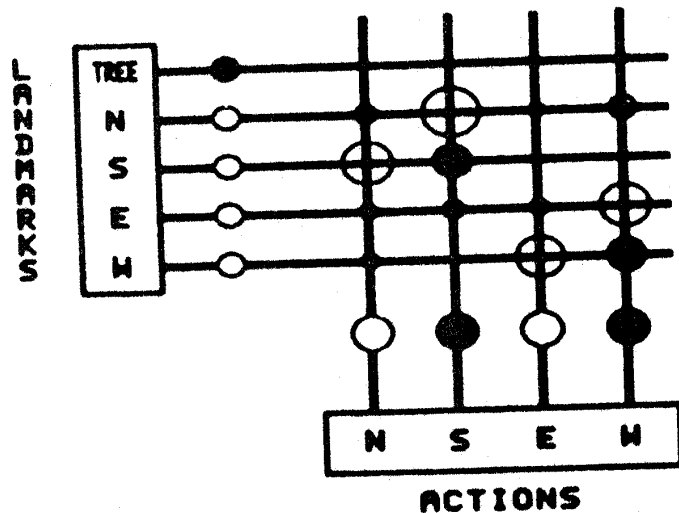
(b)



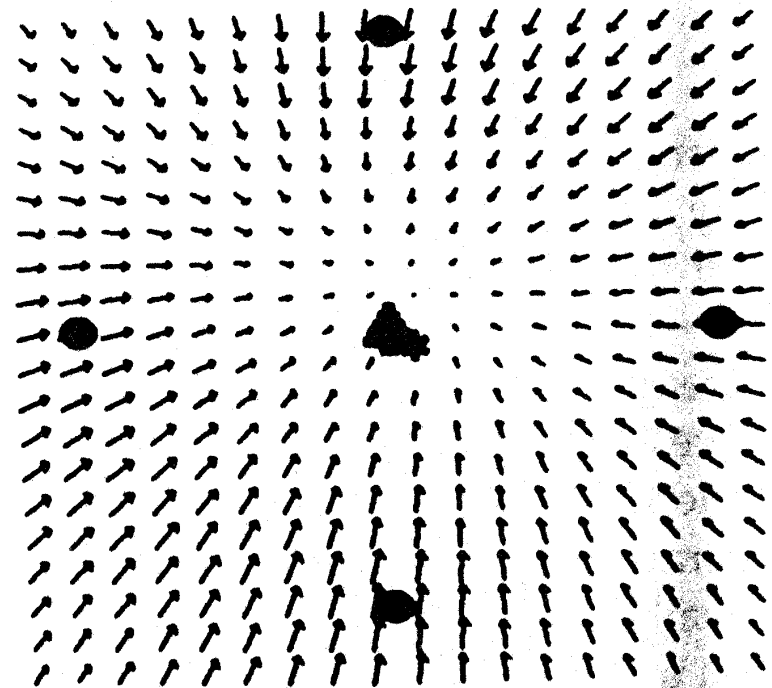
Se aprende un mapa implícitamente

16 pesos sinápticos codifican vectores de movimiento en 400 lugares en el espacio

(a)

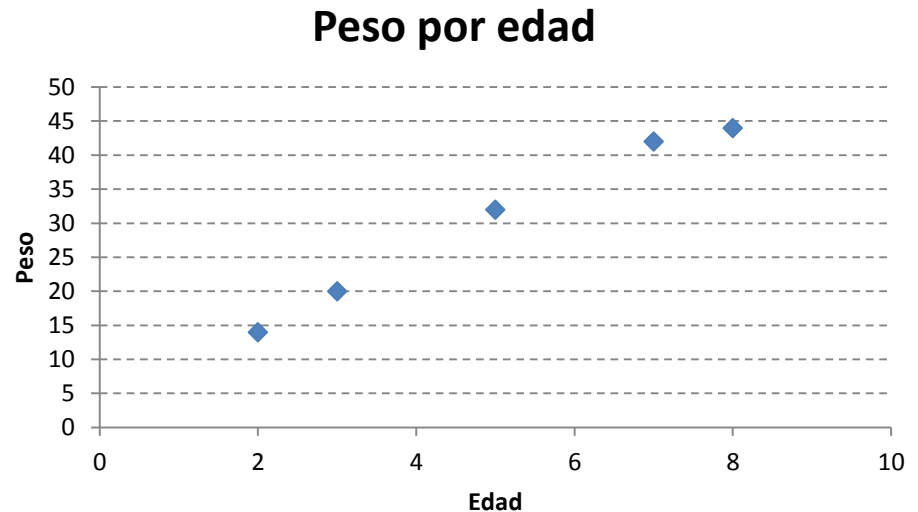


(b)



Regresión lineal

Edad	Peso
2	14
3	20
5	32
7	42
8	44

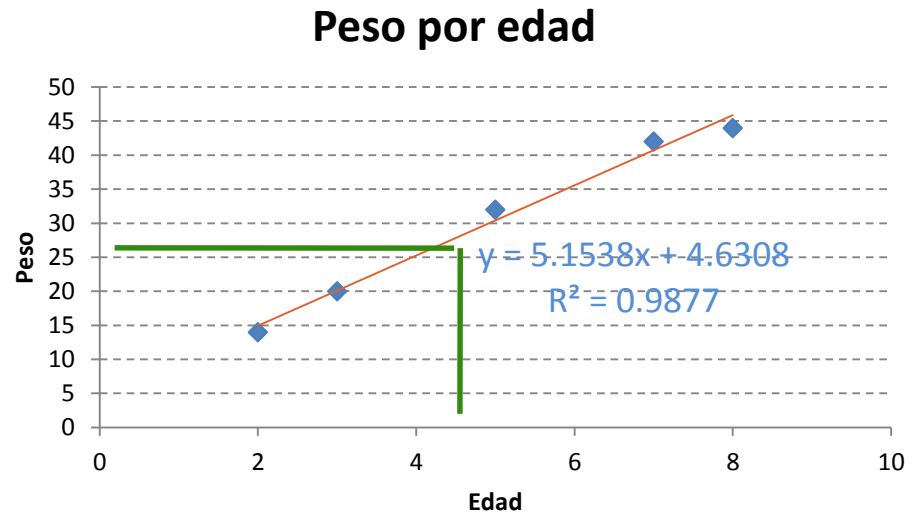


¿Cuál será el peso de alguien de 4 años de edad?

Regresión lineal

5 valores de entrenamiento (m)

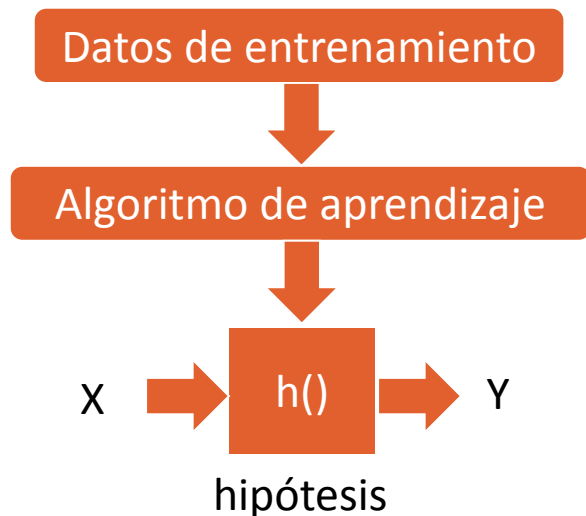
Entrada (x)	Salida (y)
Edad	Peso
2	14
3	20
5	32
7	42
8	44



$$\text{peso} = 5.1538 * \text{edad} + 4.6308$$
$$\text{edad}=4 \Rightarrow \text{peso}=25.25$$

Regresión lineal

- Para obtener la ecuación de la recta puedo hacerlo de dos formas:
 - Descenso por gradiente (algoritmo de aprendizaje)
 - Analítica



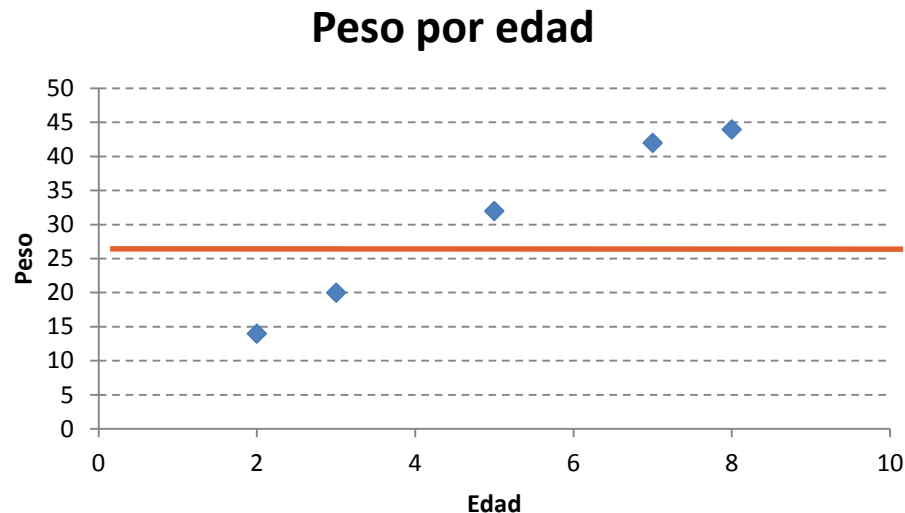
¿Cómo representamos $h()$?

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Regresión lineal univariada

Regresión lineal

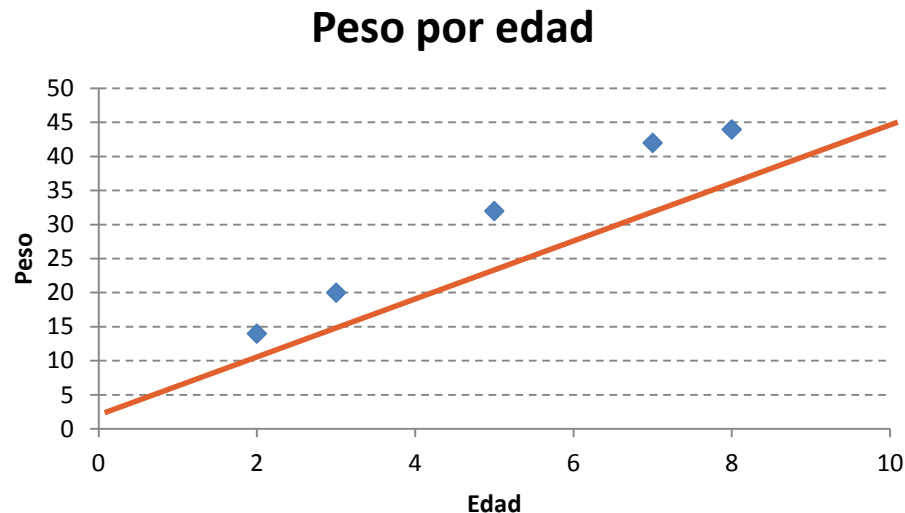
- ¿Cómo obtenemos los valores de los parámetros θ_0 y θ_1 ?



$$\theta_0 = 25 \text{ y } \theta_1 = 0$$

Regresión lineal

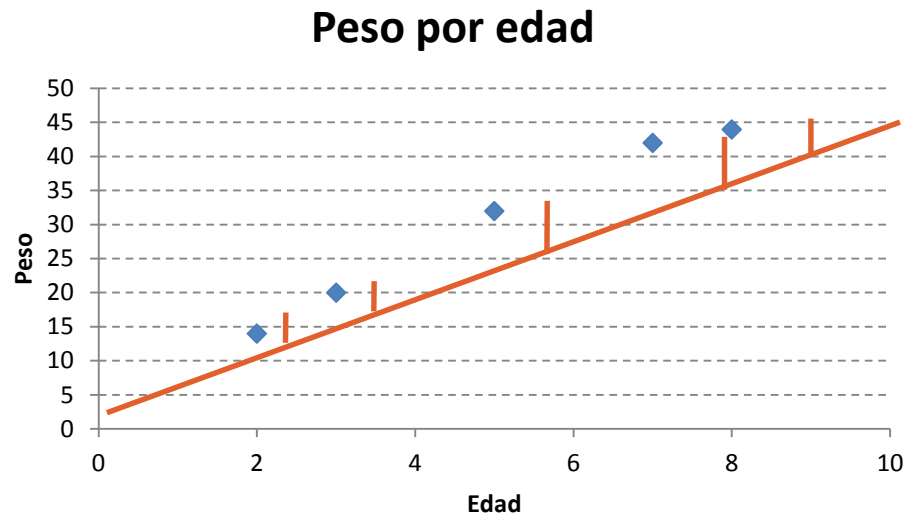
- ¿Cómo obtenemos los valores de los parámetros θ_0 y θ_1 ?



$$\theta_0 = 0 \text{ y } \theta_1 = 5$$

Regresión lineal

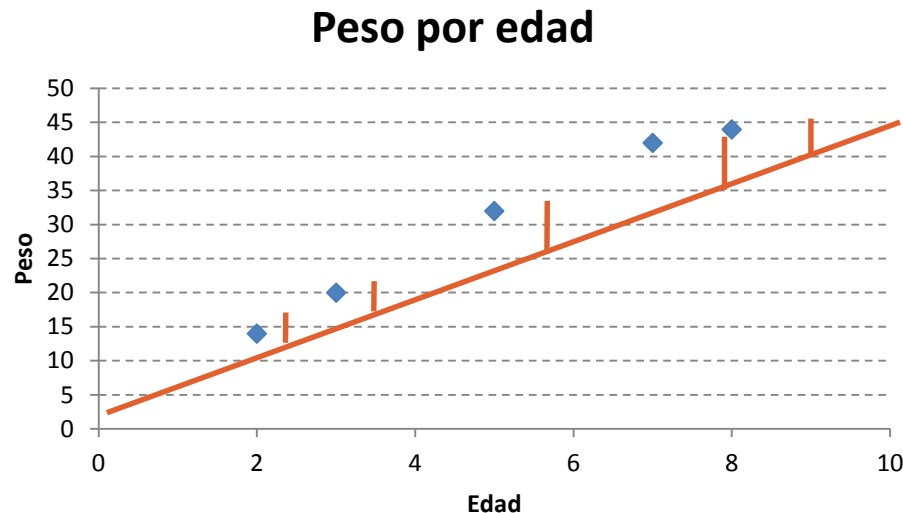
- ¿Cómo obtenemos los valores de los parámetros θ_0 y θ_1 ?



Debemos reducir la distancia entre los datos de entrenamiento y el valor de la recta, es decir, minimizar $\sum (h_{\theta}(x) - y)^2$ para todos los datos de entrenamiento

Regresión lineal

- ¿Cómo obtenemos los valores de los parámetros θ_0 y θ_1 ?

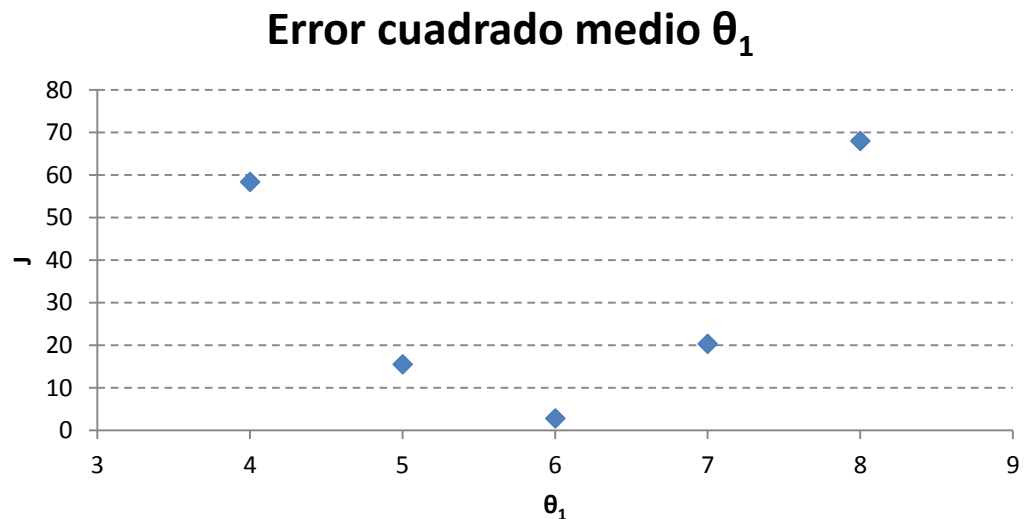


Minimizar $J(\theta_0, \theta_1) = 1/2m \cdot \sum (h_{\theta}(x) - y)^2$ (función de costo, error cuadrado medio)

Error cuadrado medio

- Error cuadrado medio (sólo θ_1)

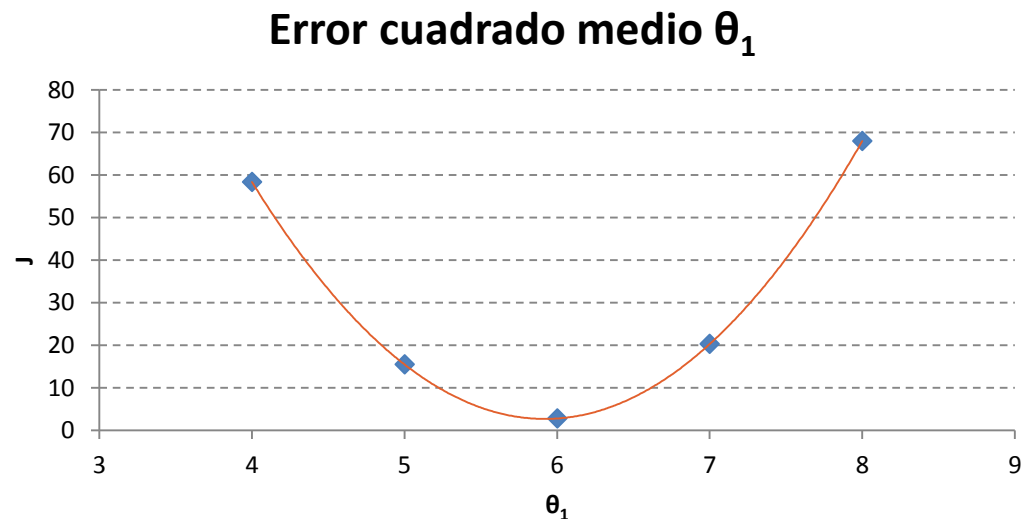
J(4)	J(5)	J(6)	J(7)	J(8)
36	16	4	0	4
64	25	4	1	16
144	49	4	9	64
196	49	0	49	196
144	16	16	144	400
58.4	15.5	2.8	20.3	68



Error cuadrado medio

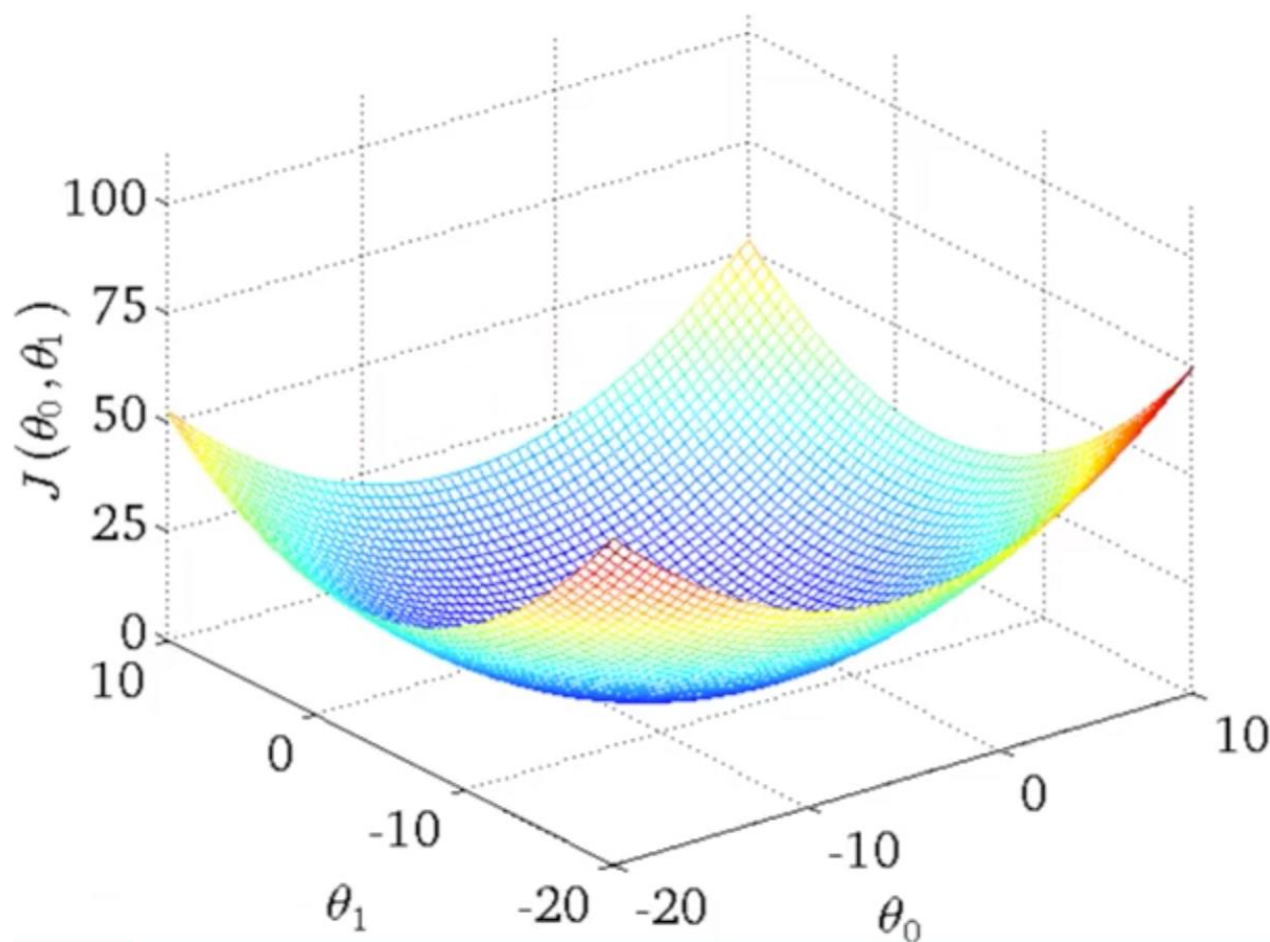
- Error cuadrado medio (sólo θ_1)

J(4)	J(5)	J(6)	J(7)	J(8)
36	16	4	0	4
64	25	4	1	16
144	49	4	9	64
196	49	0	49	196
144	16	16	144	400
58.4	15.5	2.8	20.3	68

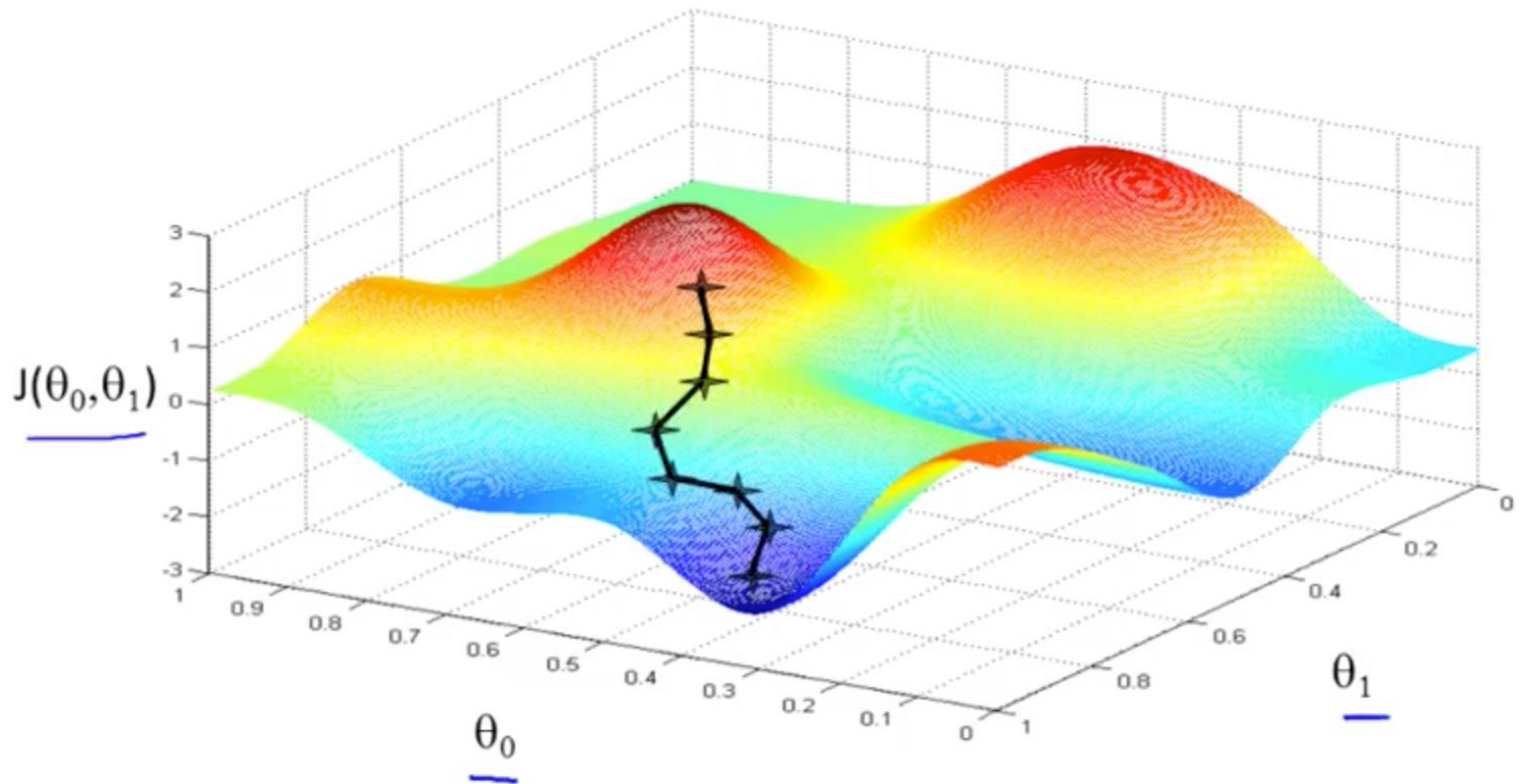


El valor mínimo del ECM se obtiene
con $\theta_1=5.1538$ y $\theta_0=4.6308$

Error cuadrado medio



Descenso por gradiente



Descenso por gradiente

- Algoritmo de descenso por gradiente

Repetir hasta que converja {

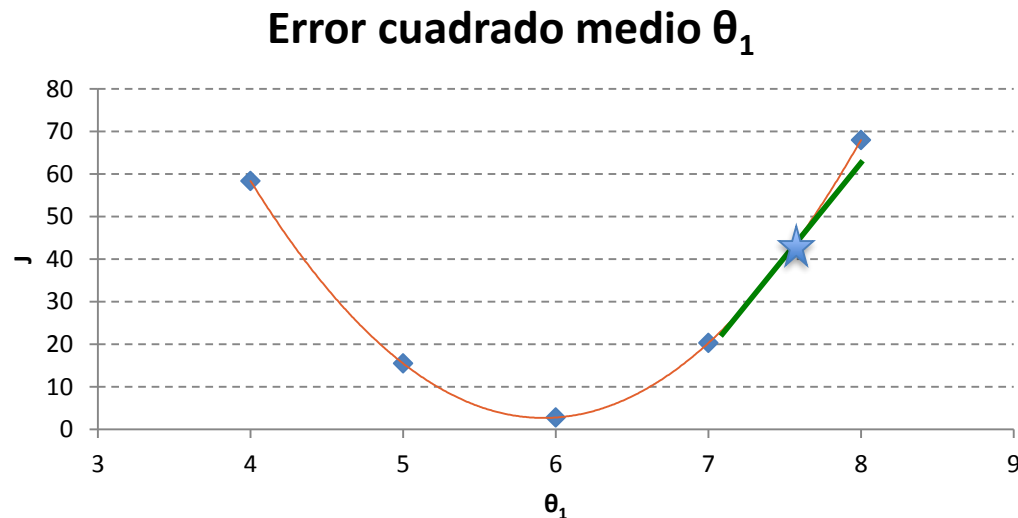
$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{Para } j=1 \text{ y } j=2$$

}

α : tasa de aprendizaje

Descenso por gradiente

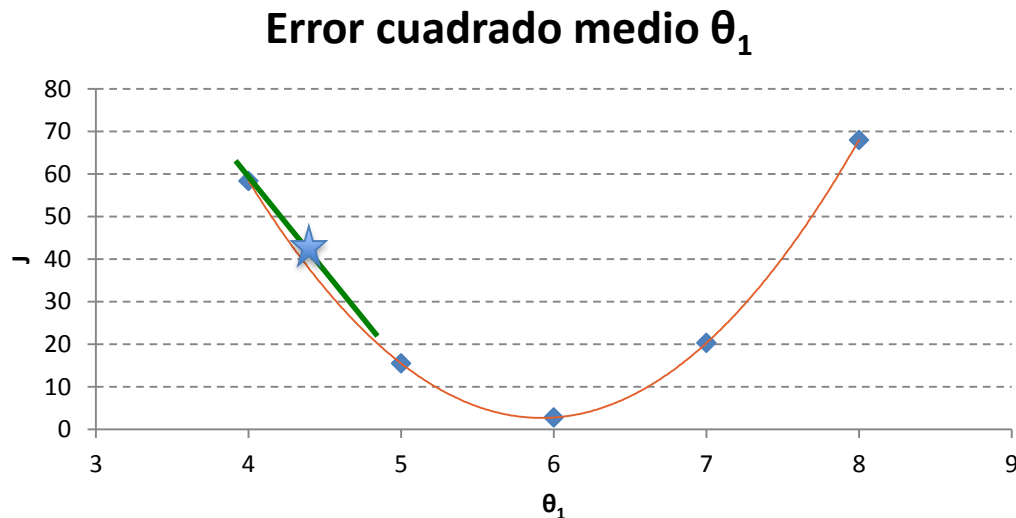
- Error cuadrado medio (sólo θ_1)



$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Descenso por gradiente

- Error cuadrado medio (sólo θ_1)

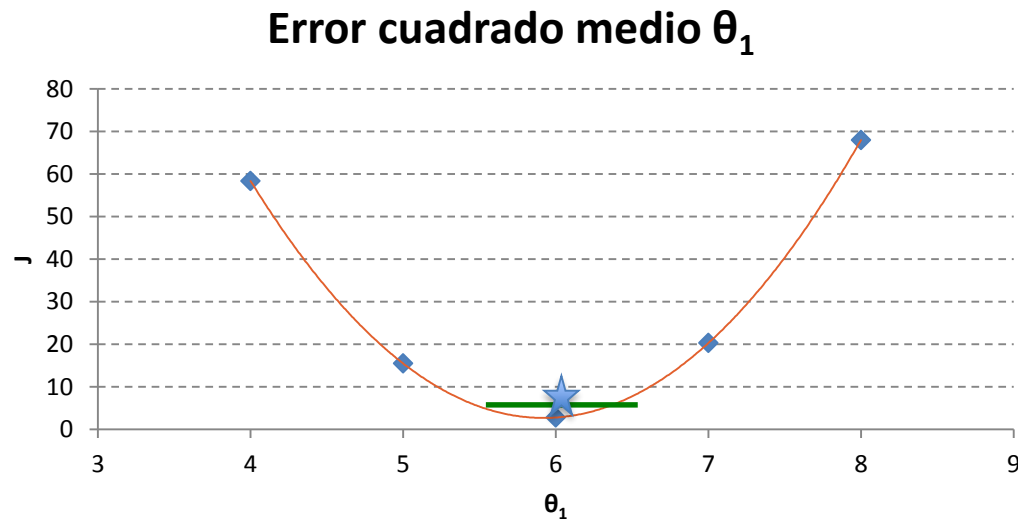


➔

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Descenso por gradiente

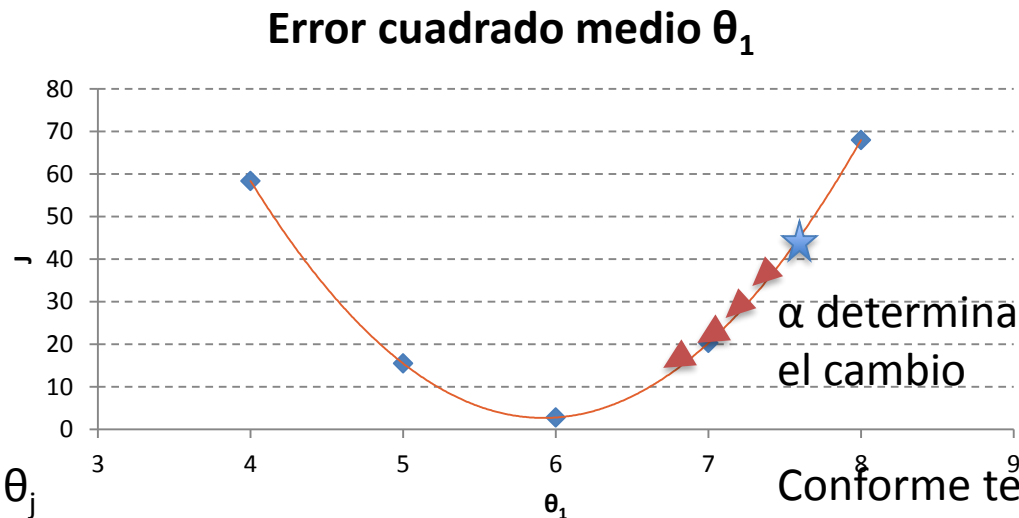
- Error cuadrado medio (sólo θ_1)



$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Descenso por gradiente

- Error cuadrado medio (sólo θ_1)



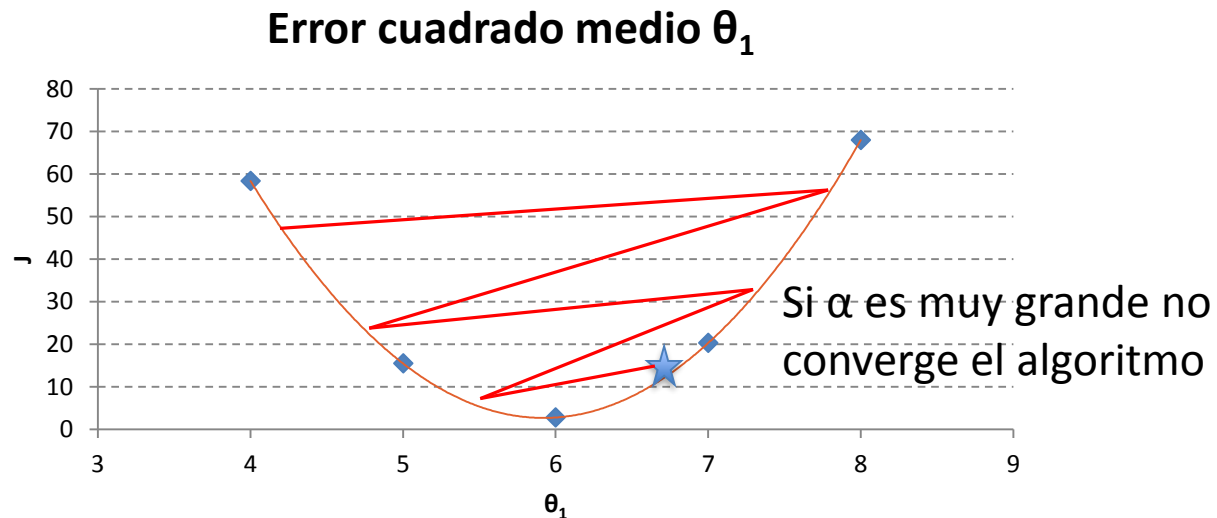
Si la variación de θ_j
entre un paso y otro es $< 10^{-3}$
declaramos que el algoritmo
convergió

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Conforme te acercas
al mínimo disminuye
el cambio (sin modificar α)

Descenso por gradiente

- Error cuadrado medio (sólo θ_1)



$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Descenso por gradiente

- ¿Cuál es la derivada parcial de J con respecto a θ ?

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

Descenso por gradiente

- Algoritmo de descenso por gradiente

Repetir hasta que converja {

$$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

}

- A este algoritmo se le conoce como descenso por gradiente por lotes, ya que en cada paso utiliza todo el conjunto de entrenamiento

Regresión lineal

- Para mejorar mi predicción puedo utilizar más características (variables)
 - No sólo la edad, sino la estatura, los metros que camina a diario, etc.
- ¿Cómo representamos la hipótesis?
$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n$$
- Esto es una regresión lineal multivariada

Regresión lineal

- $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n$
- Esto se puede representar mediante notación vectorial
 - Agregamos una característica x_0 que siempre es 1
 - $h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix} \quad X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad h_{\theta}(X) = \theta^T X$$

Descenso por gradiente

- Algoritmo de descenso por gradiente multivariado

Repetir hasta que converja {

$$\theta_j := \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \quad \text{Para } j=1 \text{ hasta } n$$

}

Escalamiento de características

- Con el fin de que el algoritmo de descenso por gradiente funcione adecuadamente es necesario que las variables se encuentren en la misma escala:

– Edad: 18 a 65, Sueldo: 10,000 a 100,000

Transformarlas para que estén en el rango ≈ -1 a 1 :

$$x_j := \frac{x_j - \mu_{x_j}}{\max(x_j) - \min(x_j)}$$

$$x_j := \frac{x_j - \mu_{x_j}}{\sigma_{x_j}} \quad \begin{array}{l} -3 \text{ a } 3 \\ -1/3 \text{ a } 1/3 \end{array}$$

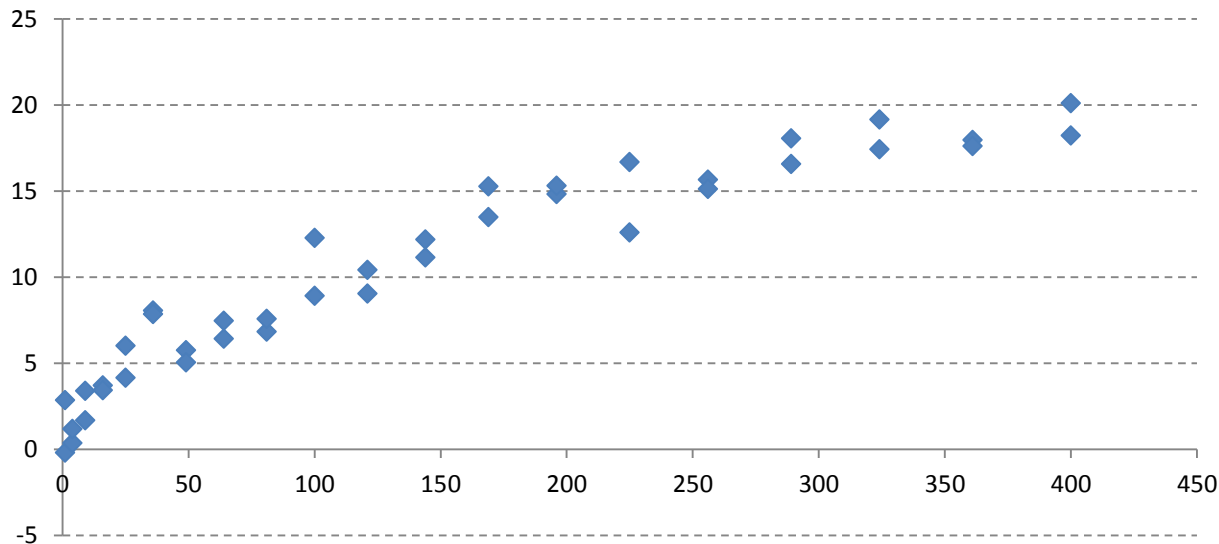
normalización

Tasa de aprendizaje

- Si la tasa de aprendizaje es muy pequeña, el algoritmo se tardará mucho en converger
- Si la tasa de aprendizaje es muy grande, el algoritmo puede no converger
- Valores comunes para α son .001, .01, .1, 1, ...

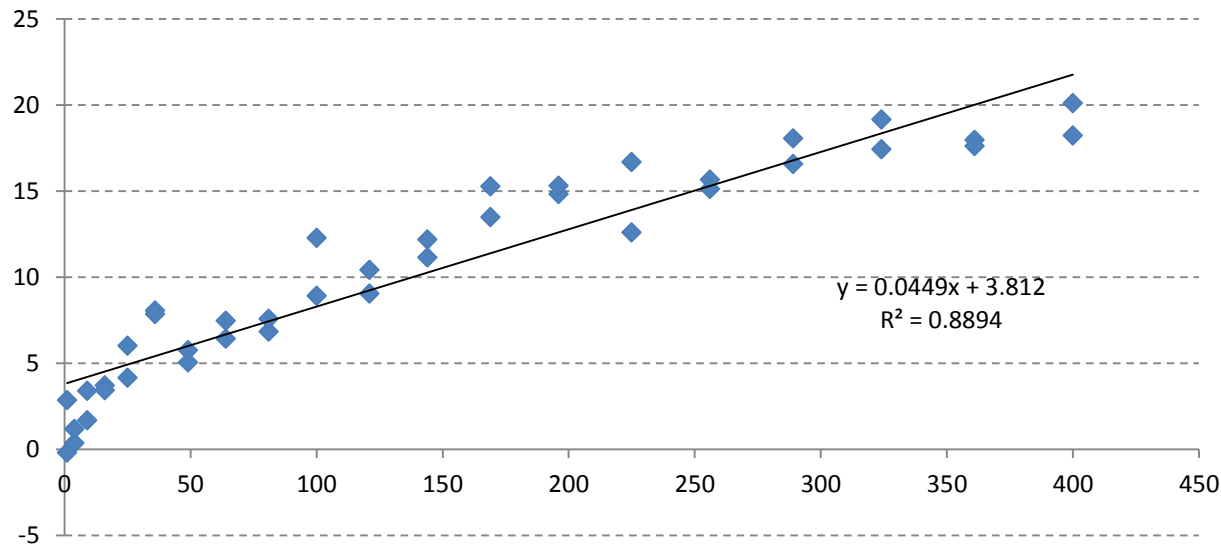
Diferentes características

- Hay ocasiones en que una línea recta no se ajusta bien a la información que vemos



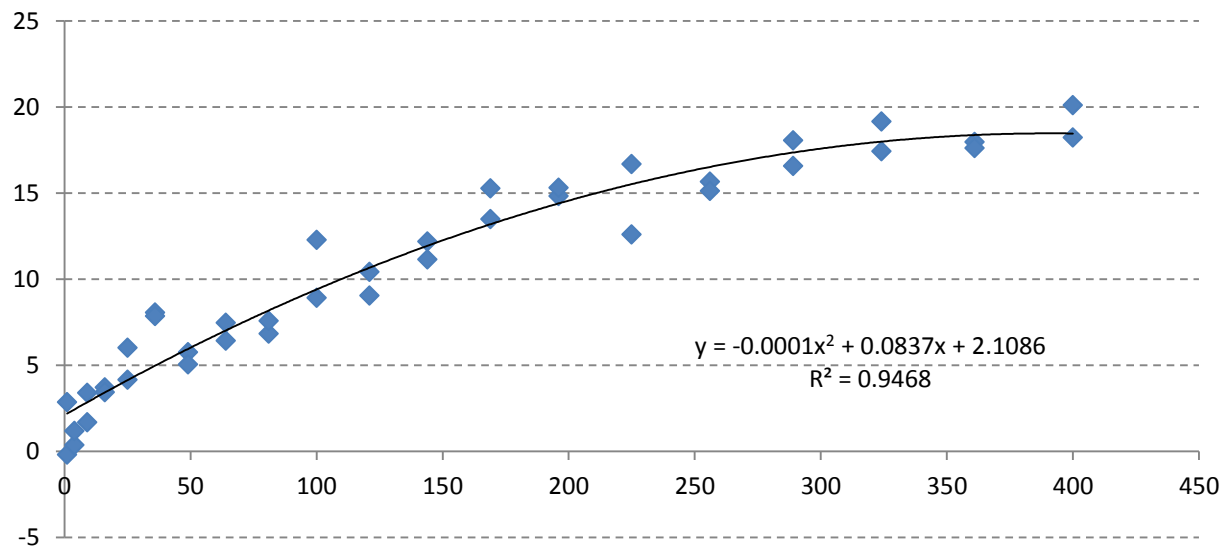
Diferentes características

- Hay ocasiones en que una línea recta no se ajusta bien a la información que vemos



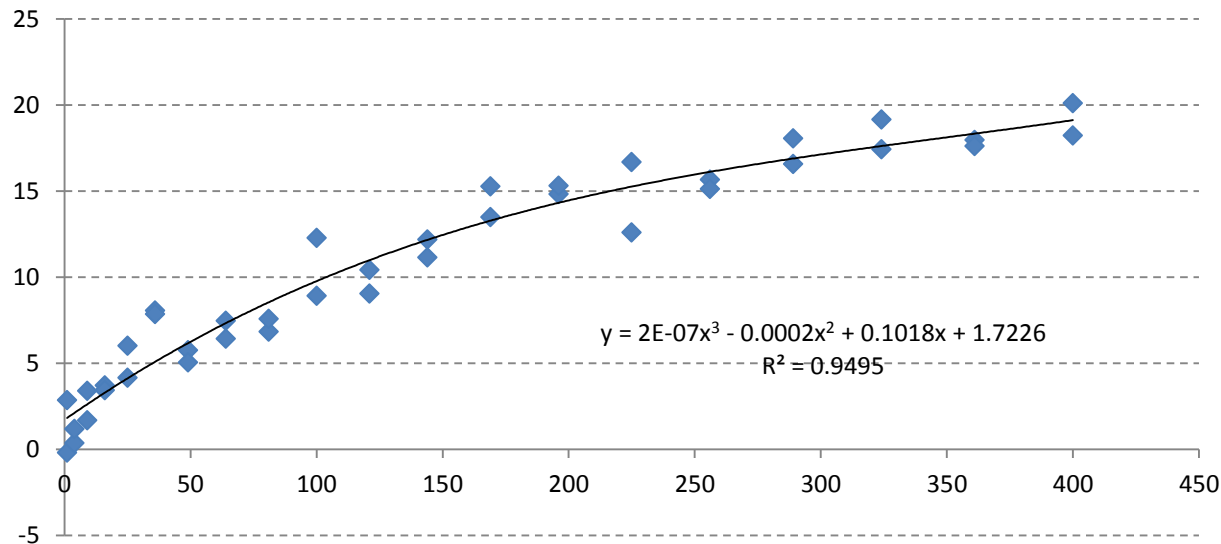
Diferentes características

- Podríamos usar en este caso una cuadrática. Se ajusta mejor, pero como es una parábola se regresaría para número mayores



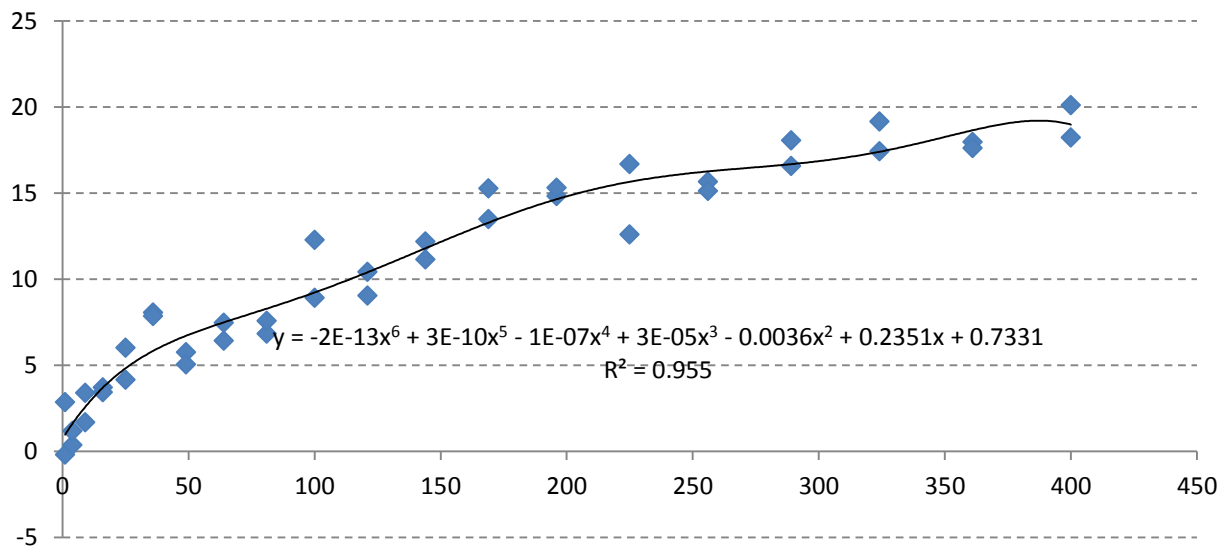
Diferentes características

- Una cúbica resolvería el problema. Para que esto funcione, podemos usar la misma hipótesis que antes, pero con 2 nuevas características, $x_2 = x_1^2$ y $x_3 = x_1^3$
 - $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$



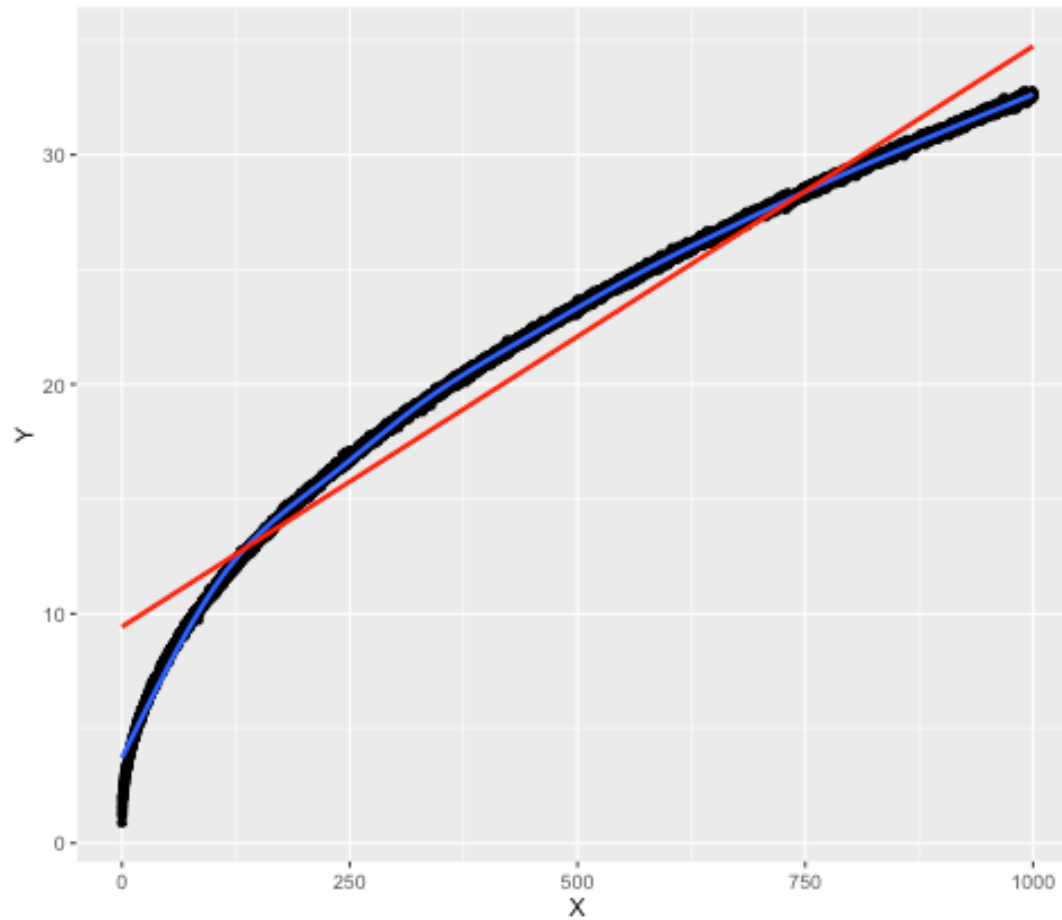
Diferentes características

- En este conjunto de datos, conforme aumento el grado del polinomio, mejoro el ajuste, pero me alejo del modelo que generó los datos (en este caso \sqrt{x}) y no predecirá adecuadamente

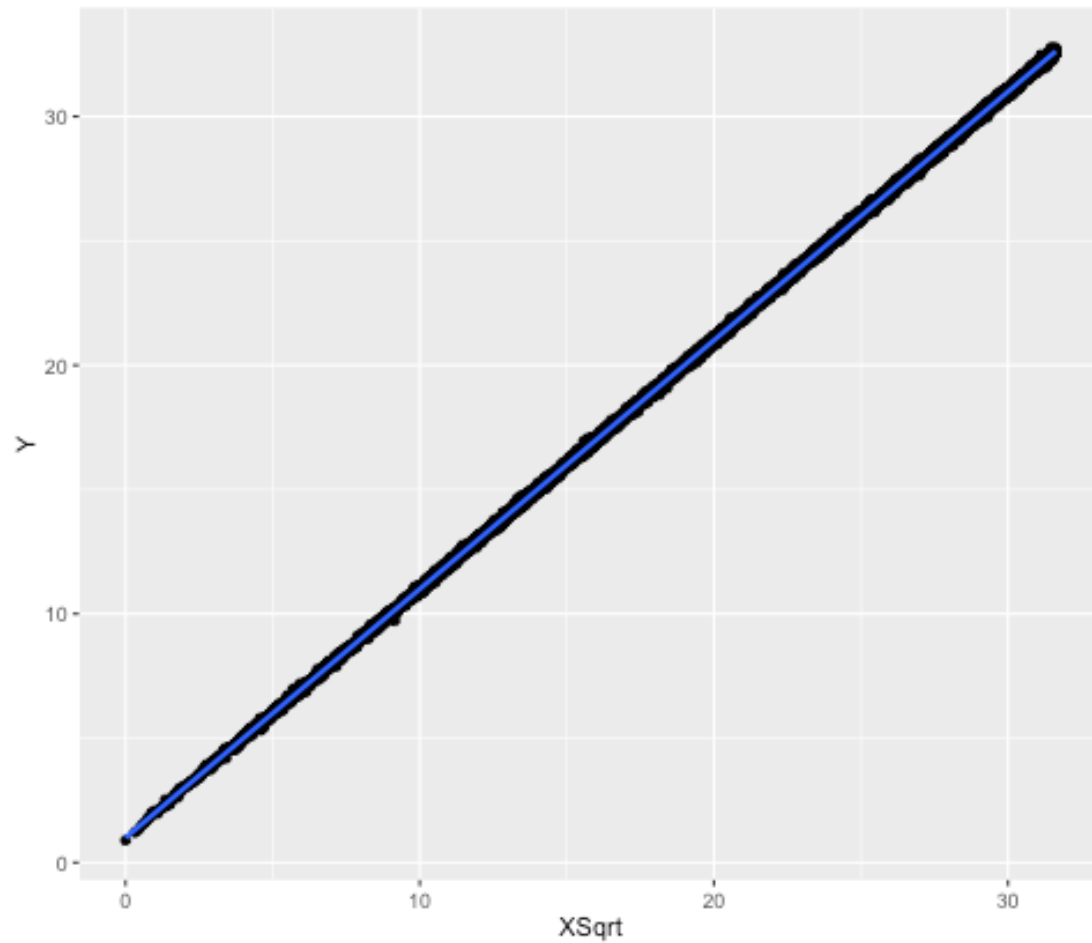


¿Cómo podríamos ajustar nuestros datos a una función de raíz cuadrada?

Raíz cuadrada

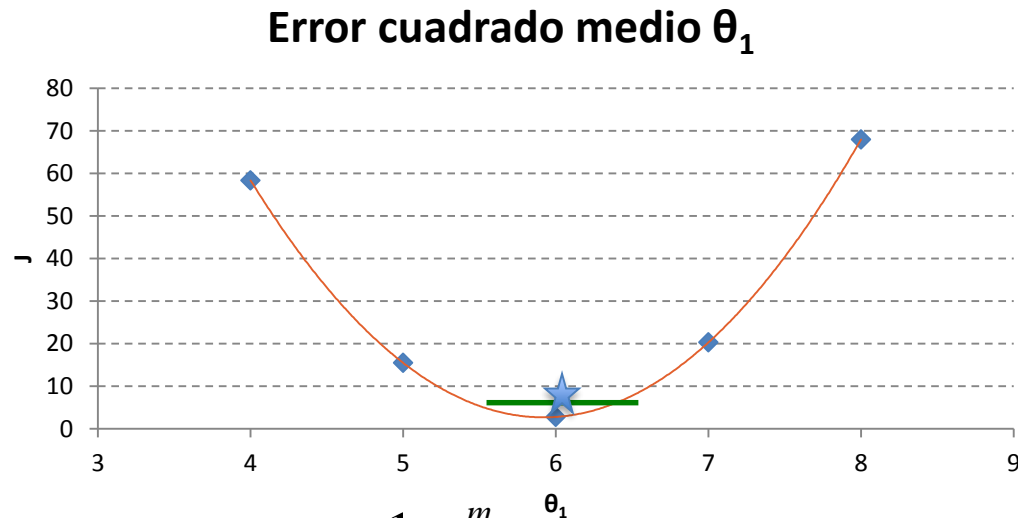


Raíz cuadrada



Solución analítica

- Error cuadrado medio (sólo θ_1)



$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_k} J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_k^{(i)} = 0$$

Solución analítica

- Resolviendo el sistema de ecuaciones obtenemos:

$$\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_k^{(i)} = 0$$

$$\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_k^{(i)} = 0$$

$$X^T \cdot (X \cdot \theta - y) = 0$$

$$X^T X \cdot \theta = X^T y$$

$$\theta = (X^T X)^{-1} X^T y$$

Descenso por gradiente vs. Solución analítica

Descenso por gradiente

- Se necesita escoger α
- Se necesitan muchas iteraciones
- Funciona bien aún para un número grande de características

Solución analítica

- No se necesita escoger α
- No se necesita iterar
- Se necesita calcular $(X^T X)^{-1}$.
La inversión de matrices $O(n^3)$
- Muy lento cuando hay un número muy grande de características ($>10,000$)



Clasificación

- Es el problema de identificar a cuál de un conjunto de categorías (sub-poblaciones) una nueva observación pertenece, sobre la base de un conjunto de entrenamiento de datos que contiene las observaciones (o instancias) cuya pertenencia a una categoría se conoce.

Clasificación

- Recolectar datos
- Seleccionar características
- Escoger modelo (descripciones)
- Entrenar clasificador
- Evaluar clasificador

Clasificación

- Binaria

- Ejemplos:

- Correo electrónico: Es o no spam.
 - Transacciones: Es o no fraudulenta.

$$y \in \{0,1\}$$

0 clase negativa
1 clase positiva

- Multiclase

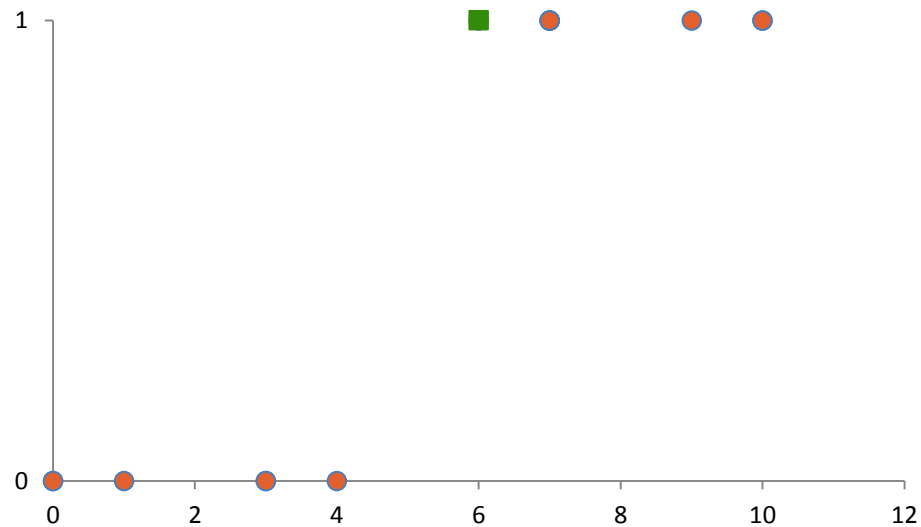
- Ejemplo:

- Reconocimiento de rostros

$$y \in \{0,1,2,\dots,k\}$$

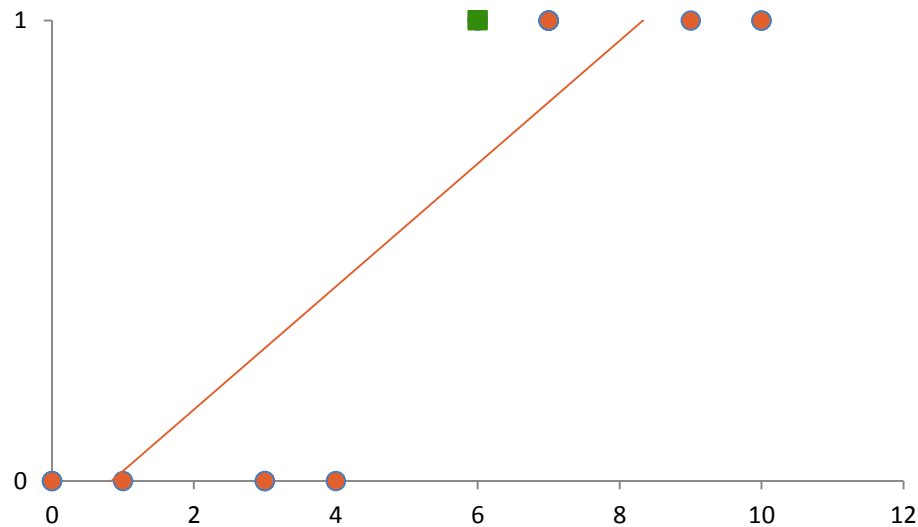
Clasificación

- Debemos poder separar de alguna forma



Clasificación

- Podríamos tratar de utilizar la regresión lineal
 - En muchos casos, como en este, una línea no separa de forma correcta los valores



Regresión logística

- Las predicciones de este método siempre están entre 0 y 1:

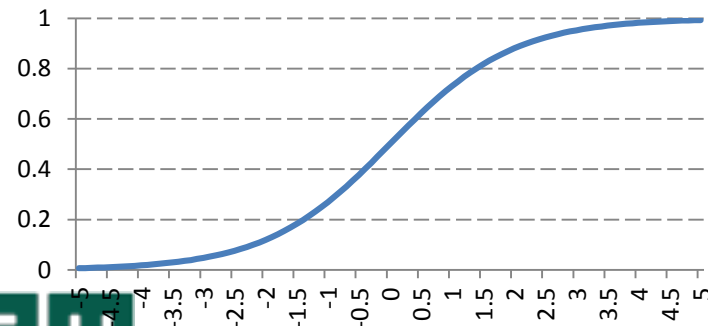
$$0 \leq h(x) \leq 1$$

- Por motivos históricos se le llama regresión, pero es un método para clasificación.

$$h_{\theta}(X) = g(\theta^T X)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Función Logística



Regresión logística

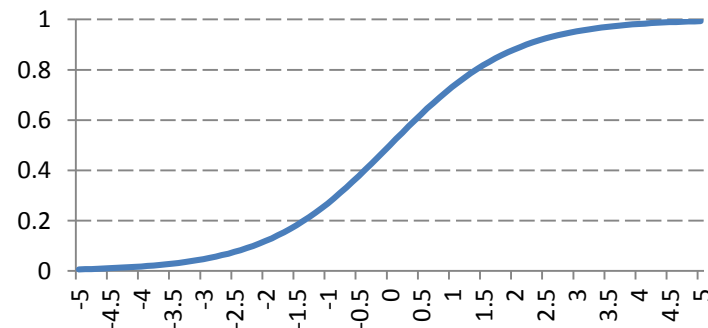
- El resultado lo tomaré como la probabilidad de que la clasificación sea 1.
 - 0.7 indicaría que con 70% de probabilidad las características se clasificarían como 1

$$h_{\theta}(X) = \frac{1}{1 + e^{-\theta^T X}}$$

$$h_{\theta}(X) = p(y = 1 \mid x; \theta)$$

Probabilidad de que “y” sea igual a 1
dado x, parametrizado por θ

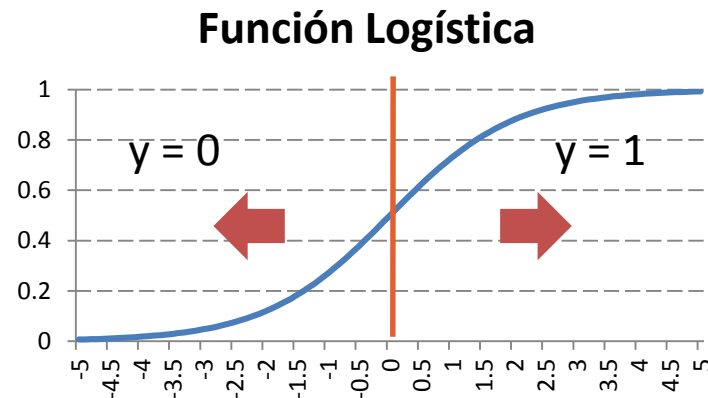
Función Logística



Límite de decisión

- $\theta^T X$ define el límite de decisión con el cual clasificamos a nuestra entrada
 - Dependiendo de cómo se concentren los datos de entrada, se determina que tipo de límite utilizo

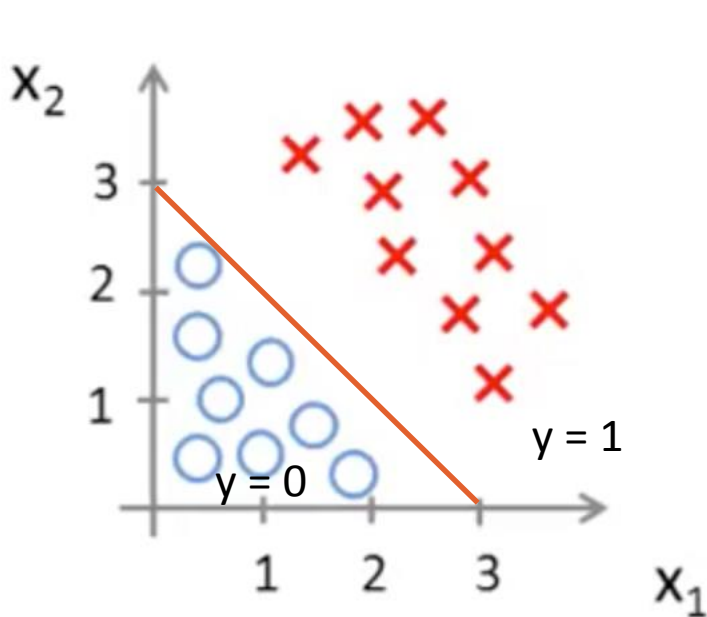
$$h_{\theta}(X) = \frac{1}{1 + e^{-\theta^T X}}$$



Si $\theta^T X \geq 0$ entonces lo clasificamos como 1

Si $\theta^T X < 0$ entonces lo clasificamos como 0

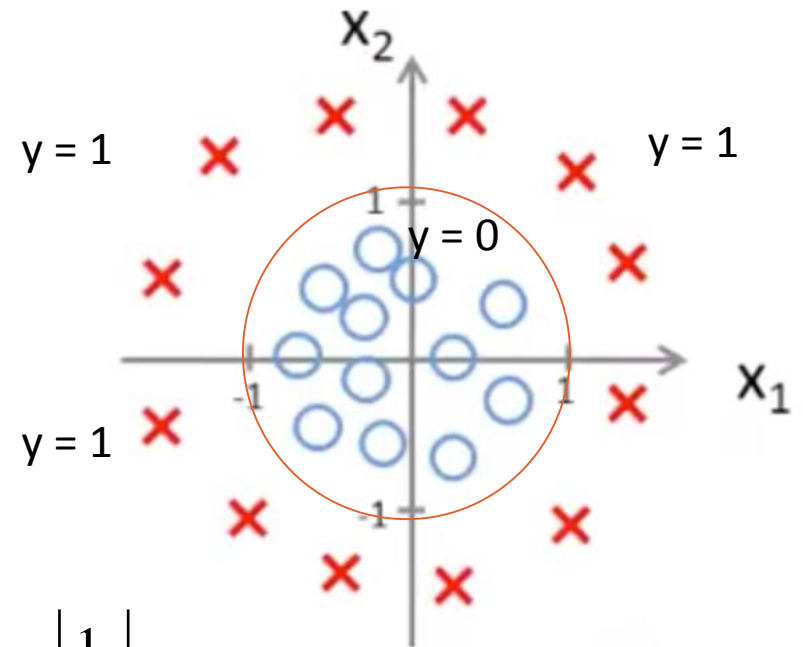
Límite de decisión



$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$x_1 + x_2 \geq 3$$



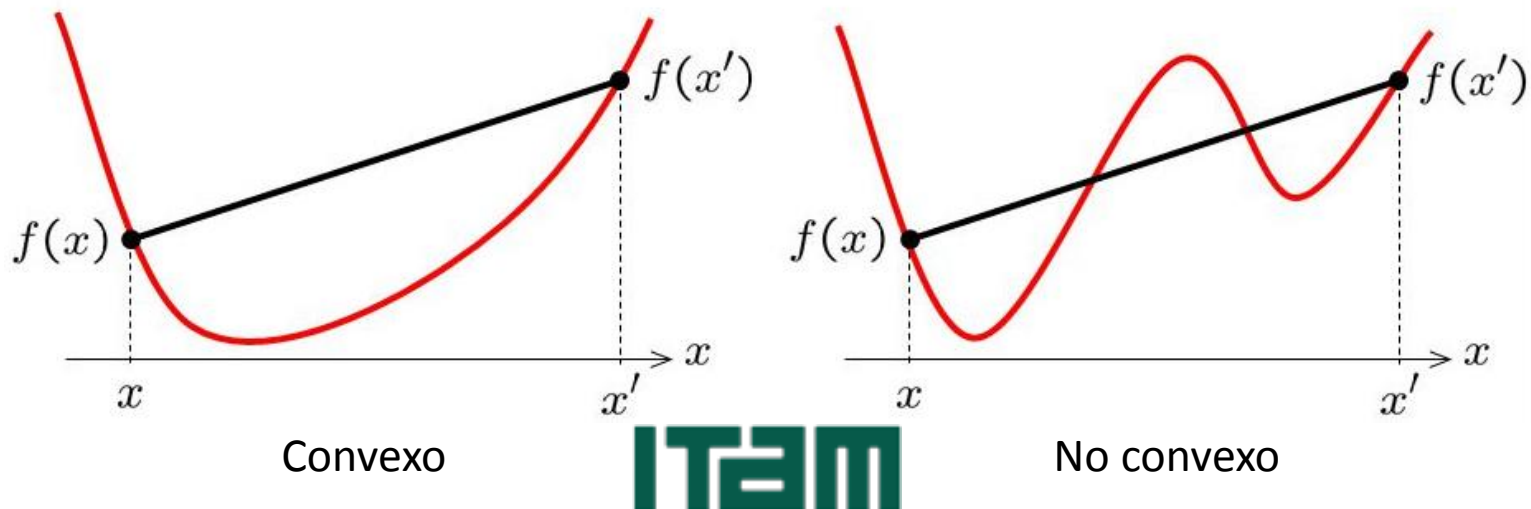
$$\theta = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$$

$$x_1^2 + x_2^2 \geq 1$$

Función de costo

- Si utilizamos la misma función de costo que en regresión logística no obtendríamos los resultados requeridos debido a que la no linealidad (función logística) generaría una función no convexa.



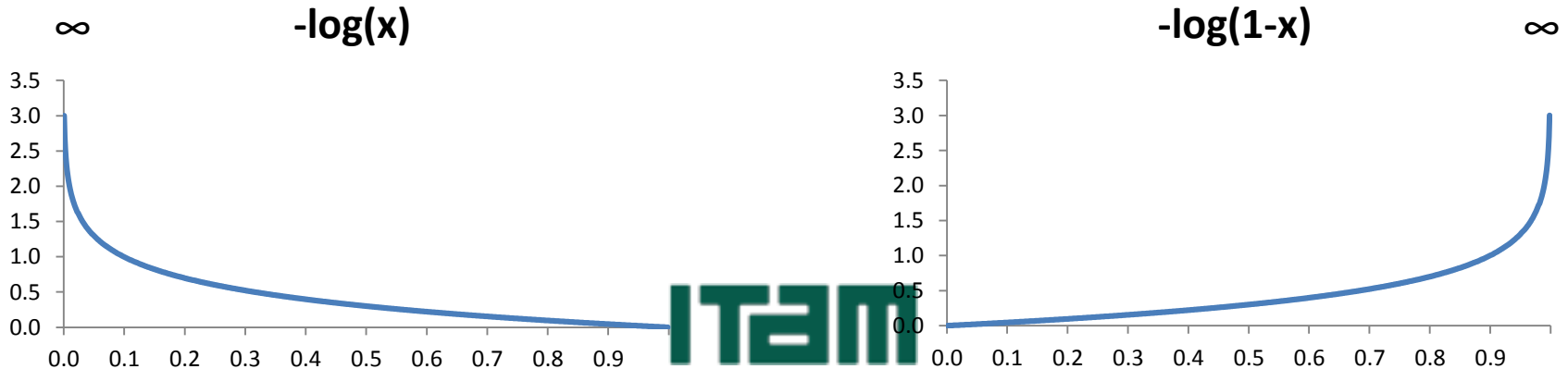
Función de costo

- Para asegurar que sea convexa usamos:

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{i=1}^m \text{costo}(h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\text{costo}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{si } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{si } y = 0 \end{cases}$$



Función de costo

- Reescribimos la función de costo

$$\text{costo}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{si } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{si } y = 0 \end{cases}$$

$$\text{costo}(h_{\theta}(x), y) = -y \cdot \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x))$$

- Derivamos la función

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n)$$

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{i=1}^m \text{costo}(h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Descenso por gradiente

- Algoritmo de descenso por gradiente para regresión lineal


Repetir hasta que converja {


$$\theta_j := \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad \text{Para } j=0 \text{ hasta } n$$


}

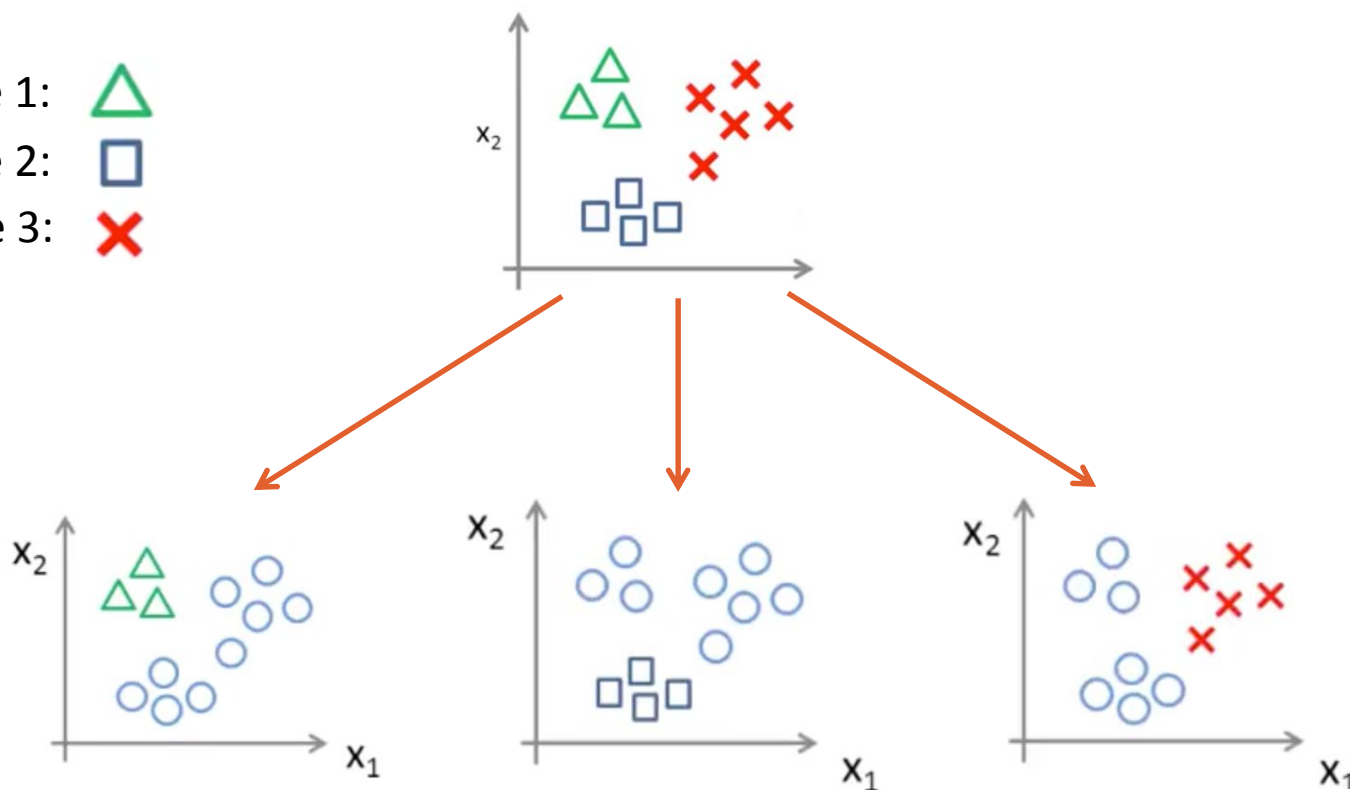
Clasificación multiclase

- Uno vs. Resto

Clase 1: 




Clase 2: 

Clase 3: 



Clasificación multiclase

- Uno vs. Resto

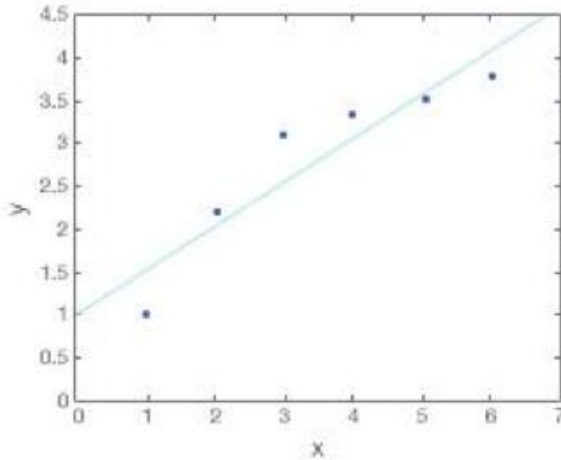
Clase 1: 
Clase 2: 
Clase 3: 



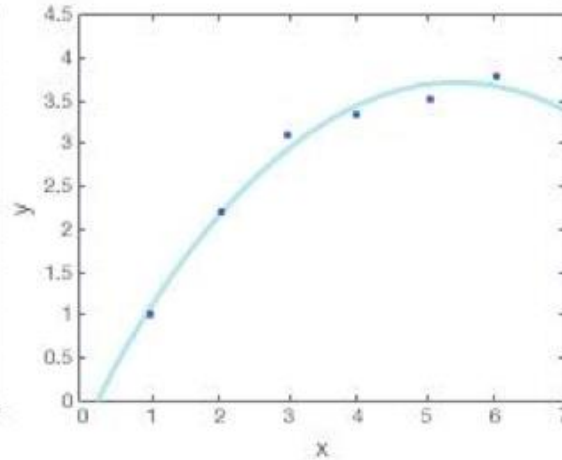
$$h_{\theta}^{(i)}(X) = p(y = i | x; \theta)$$

$$\max_i h_{\theta}^{(i)}(X)$$

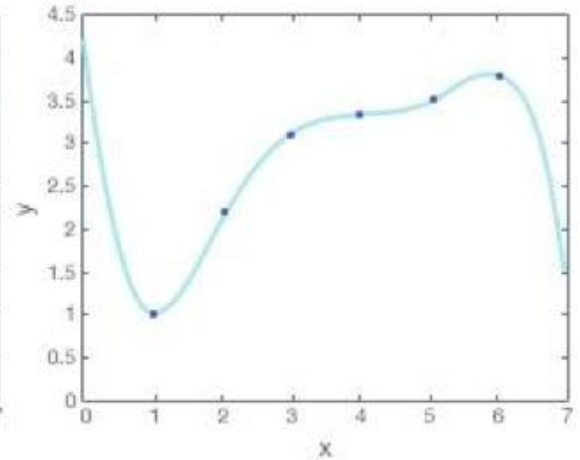
El problema del sobre-ajuste



Pobremente ajustado
Muy Sesgado



Correctamente ajustado

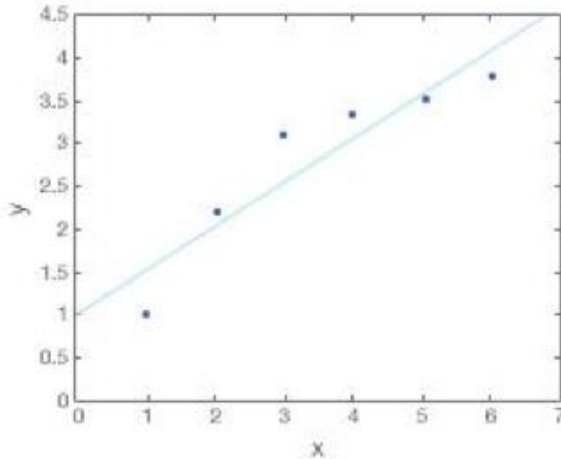


Sobre-ajustado
Mucha Varianza

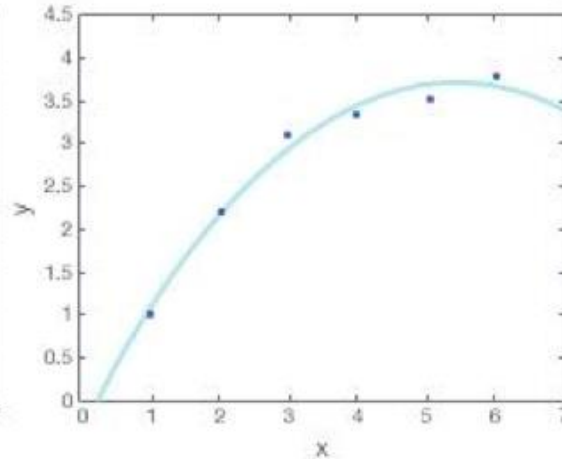
La maldición de la
Dimensionalidad (M^D)

Necesita muchos datos para
que no se sobre-ajuste

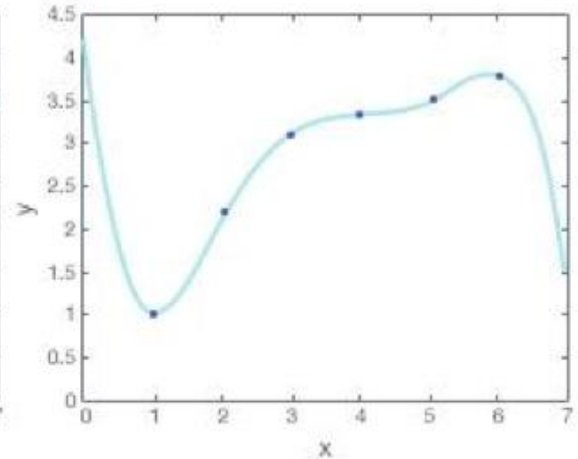
El problema del sobre-ajuste



Pobremente ajustado
Muy Sesgado



Correctamente ajustado



Sobre-ajustado
Mucha Varianza

Para corregirlo:

- Reducimos el número de variables

 - Manualmente

 - Modelo

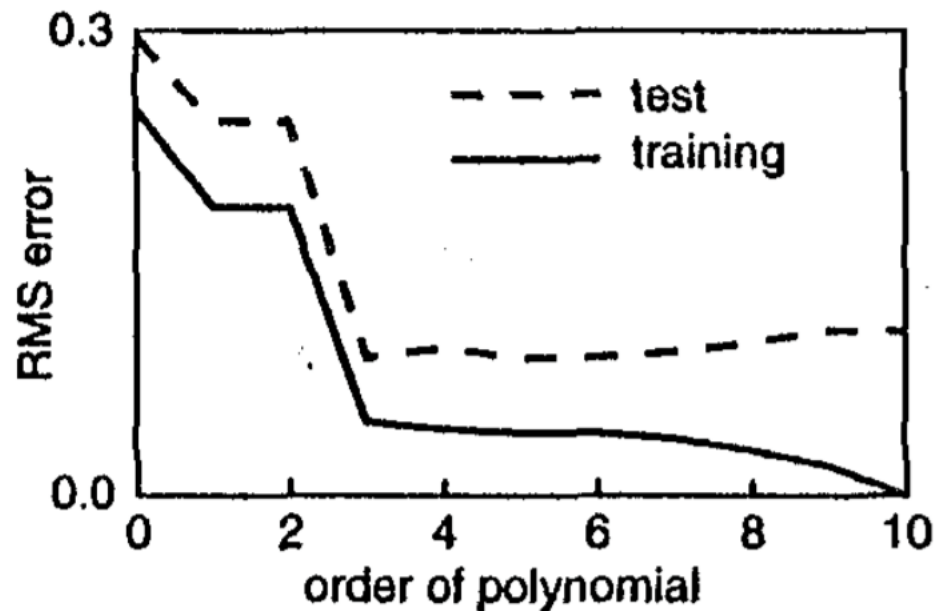
- Regularización

 - Disminuir la magnitud de θ

 - Funciona bien cuando tenemos muchas características

Root Mean Square

$$E^{RMS} = \sqrt{\frac{1}{m} \sum_{i=1}^m \left(h_{\theta}(x^{(i)}) - y^{(i)} \right)^2}$$



Regularización

- Al reducir la magnitud de θ
 - Obtenemos hipótesis más simples
 - Minimizamos la probabilidad de sobre-ajuste

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

$$\frac{\partial}{\partial \theta_k} J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_k^{(i)} + \frac{\lambda}{m} \theta_k$$

λ parámetro de regularización

Descenso por gradiente

- Algoritmo de descenso por gradiente para regresión lineal y logística (h_θ apropiada)

Repetir hasta que converja {

$$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \cdot \left[\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

} Para j=1 hasta n

$$\theta_j := \theta_j \cdot \left(1 - \alpha \cdot \frac{\lambda}{m} \right) - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Solución analítica

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \right)^{-1} \cdot X^T y$$

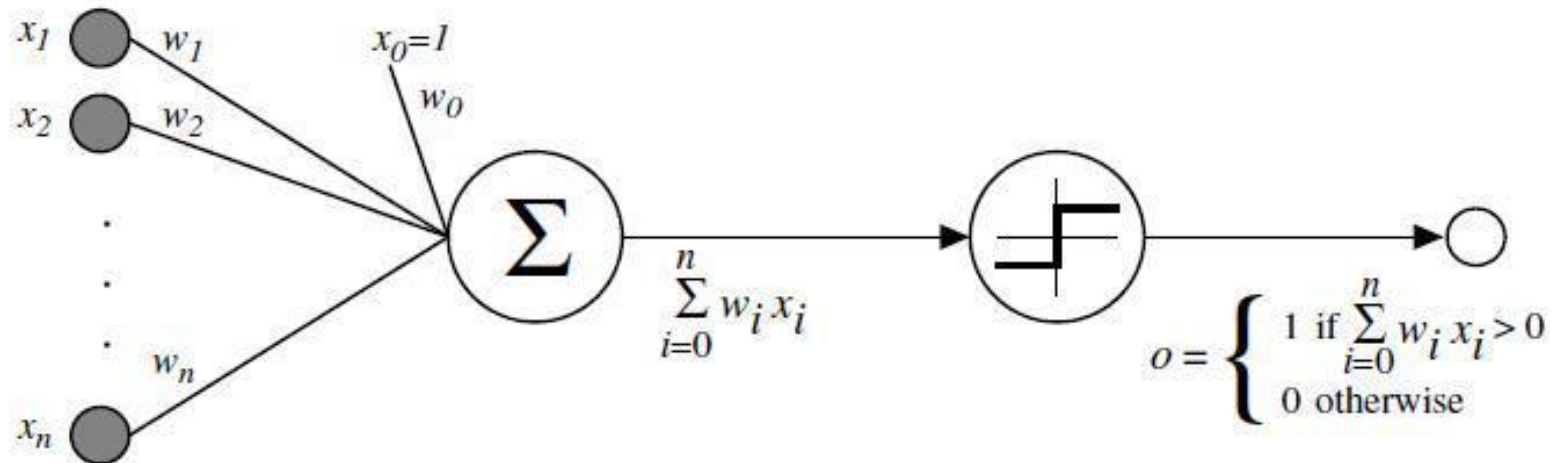
Agregando este término además la hacemos
la matriz invertible si era singular

Hipótesis no lineales

- Cuando se tienen muchas características (100 o más) y requiero de un polinomio de grado mayor a uno para clasificar de manera correcta, el número de parámetros aumenta de forma importante
 - Para 100 requeriría 5,000 si se necesita un polinomio de grado 2
 - Una imagen de 50x50 pixeles requeriría 3.1 millones si se necesita un polinomio de grado 2

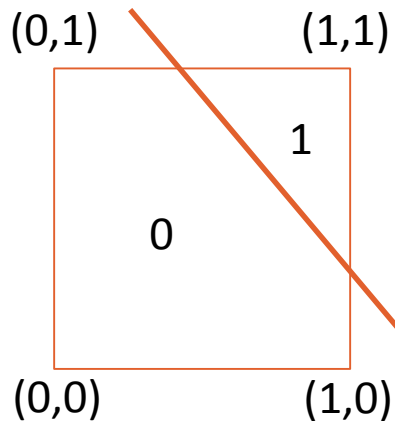
Redes neuronales

- Perceptrón

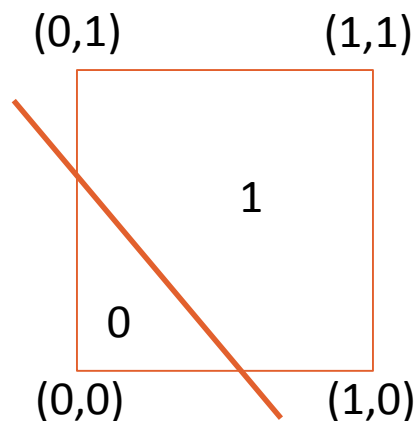


Clasificación

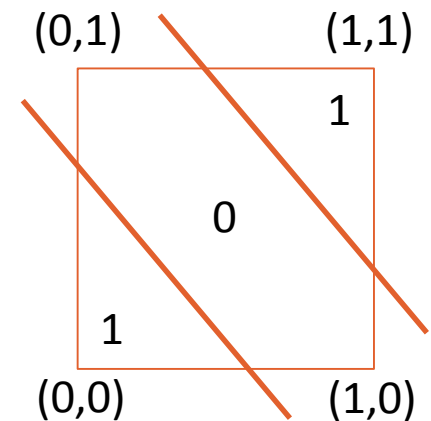
- Un problema tan simple como una operación XNOR no puede ser resuelto con un solo perceptrón.
 - Pero sí con varios de ellos



And



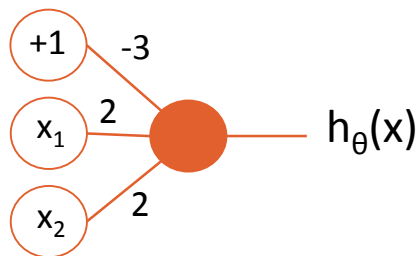
Or



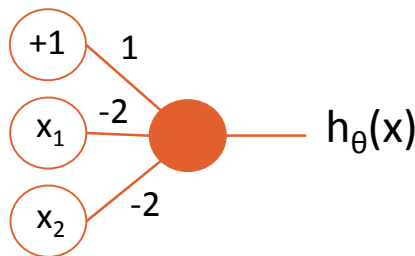
Xnor

Clasificación

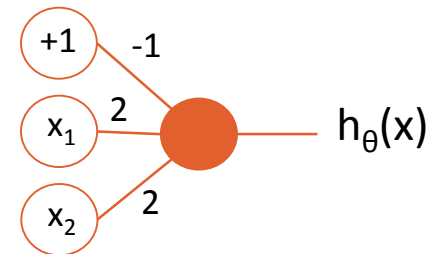
- Un problema tan simple como una operación XNOR no puede ser resuelto con un solo perceptrón.
 - Pero sí con varios de ellos



x_1 and x_2



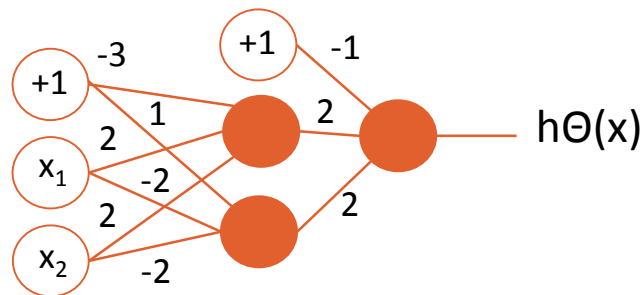
not x_1 and not x_2



x_1 or x_2

Clasificación

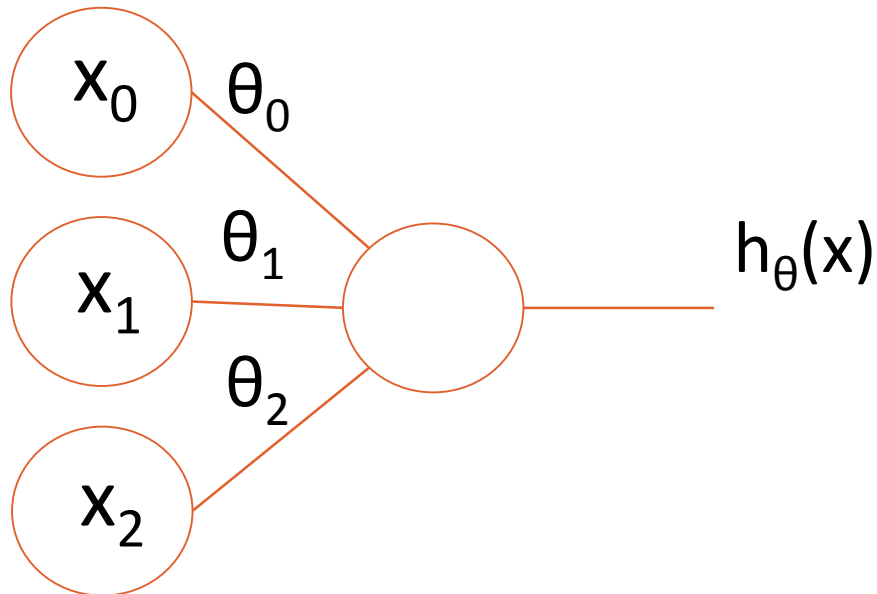
- Un problema tan simple como una operación XNOR no puede ser resuelto con un solo perceptrón.
 - Pero sí con varios de ellos



$x_1 \text{ xnor } x_2$

Modelo de neurona

$x_0 = 1$ (unidad de sesgo)

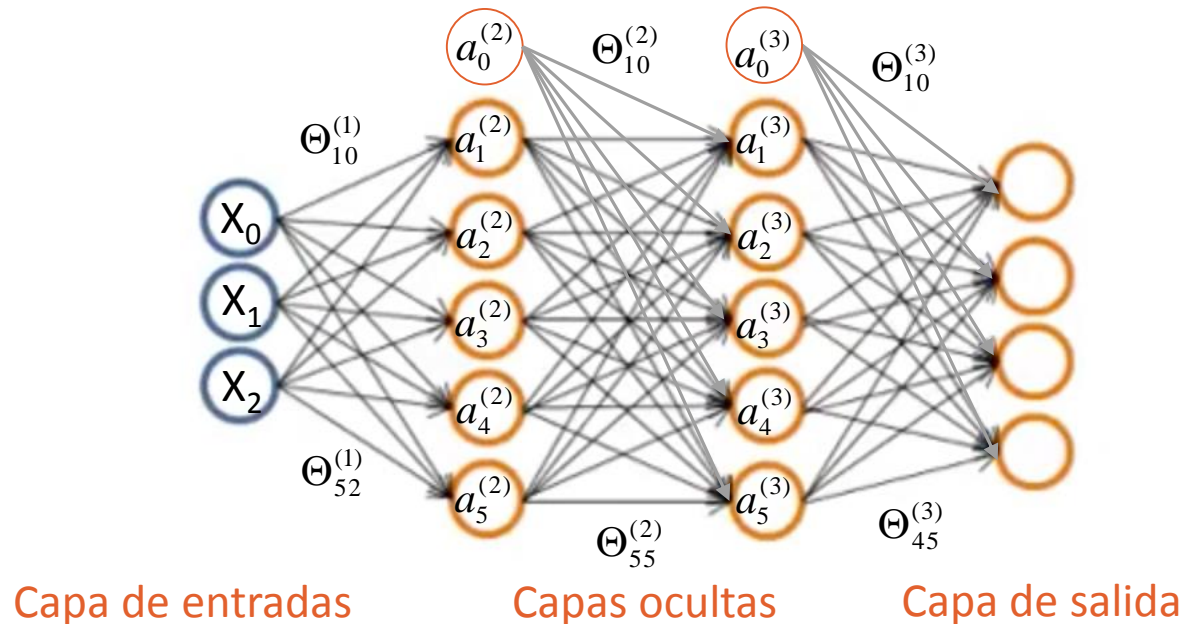


$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix}$$

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Función de activación
sigmoidea o logística

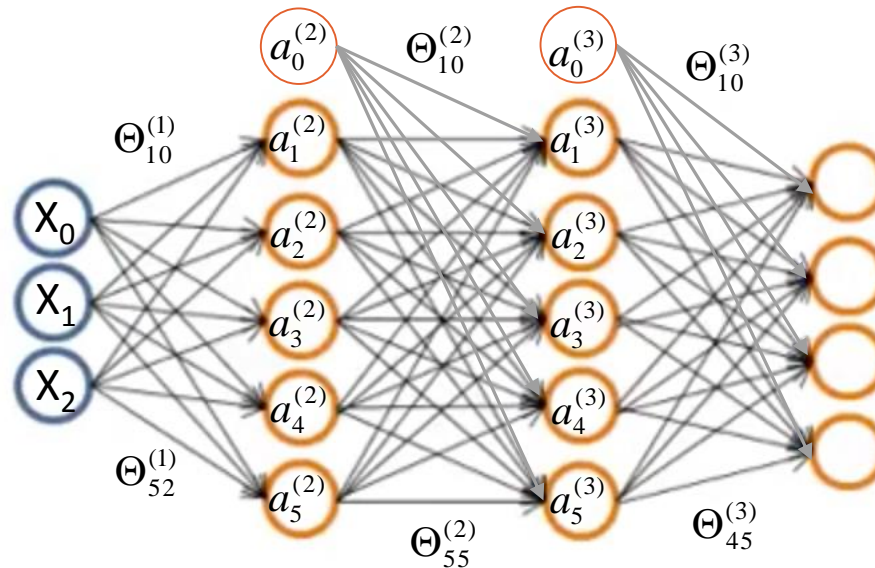
Redes Neuronales



$a_i^{(j)}$ Unidad i de “activación” en la capa j

$\Theta^{(j)}$ Matriz de pesos controlando el mapeo de la función de la capa j a la $j+1$

Redes Neuronales



Capa de entradas

Capas ocultas

Capa de salida

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2)$$

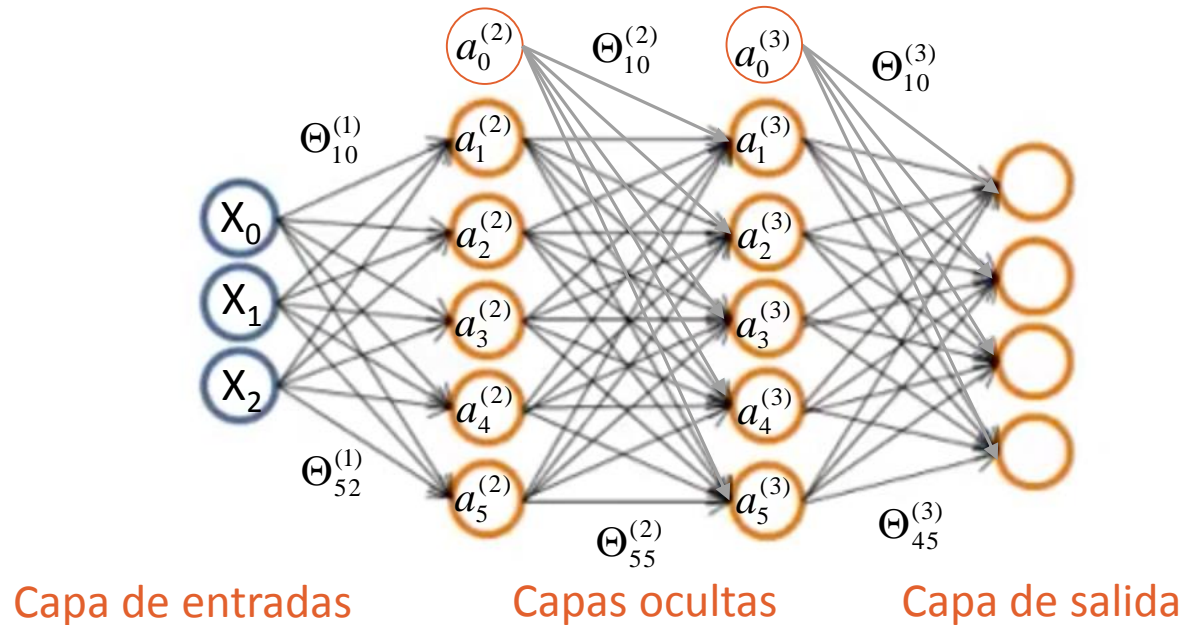
$$a_4^{(2)} = g(\Theta_{40}^{(1)}x_0 + \Theta_{41}^{(1)}x_1 + \Theta_{42}^{(1)}x_2)$$

$$a_5^{(2)} = g(\Theta_{50}^{(1)}x_0 + \Theta_{51}^{(1)}x_1 + \Theta_{52}^{(1)}x_2)$$

Si hay N_j neuronas en la capa j y N_{j+1} neuronas en la capa $j+1$, la matriz $\Theta^{(i)}$ tendrá un tamaño $N_{j+1} \cdot (N_j+1)$

$$h_{\Theta}(x) = a_1^{(4)} = g(\Theta_{10}^{(3)}a_0^{(3)} + \Theta_{11}^{(3)}a_1^{(3)} + \Theta_{12}^{(3)}a_2^{(3)} + \Theta_{13}^{(3)}a_3^{(3)} + \Theta_{14}^{(3)}a_4^{(3)} + \Theta_{15}^{(3)}a_5^{(3)})$$

Redes Neuronales



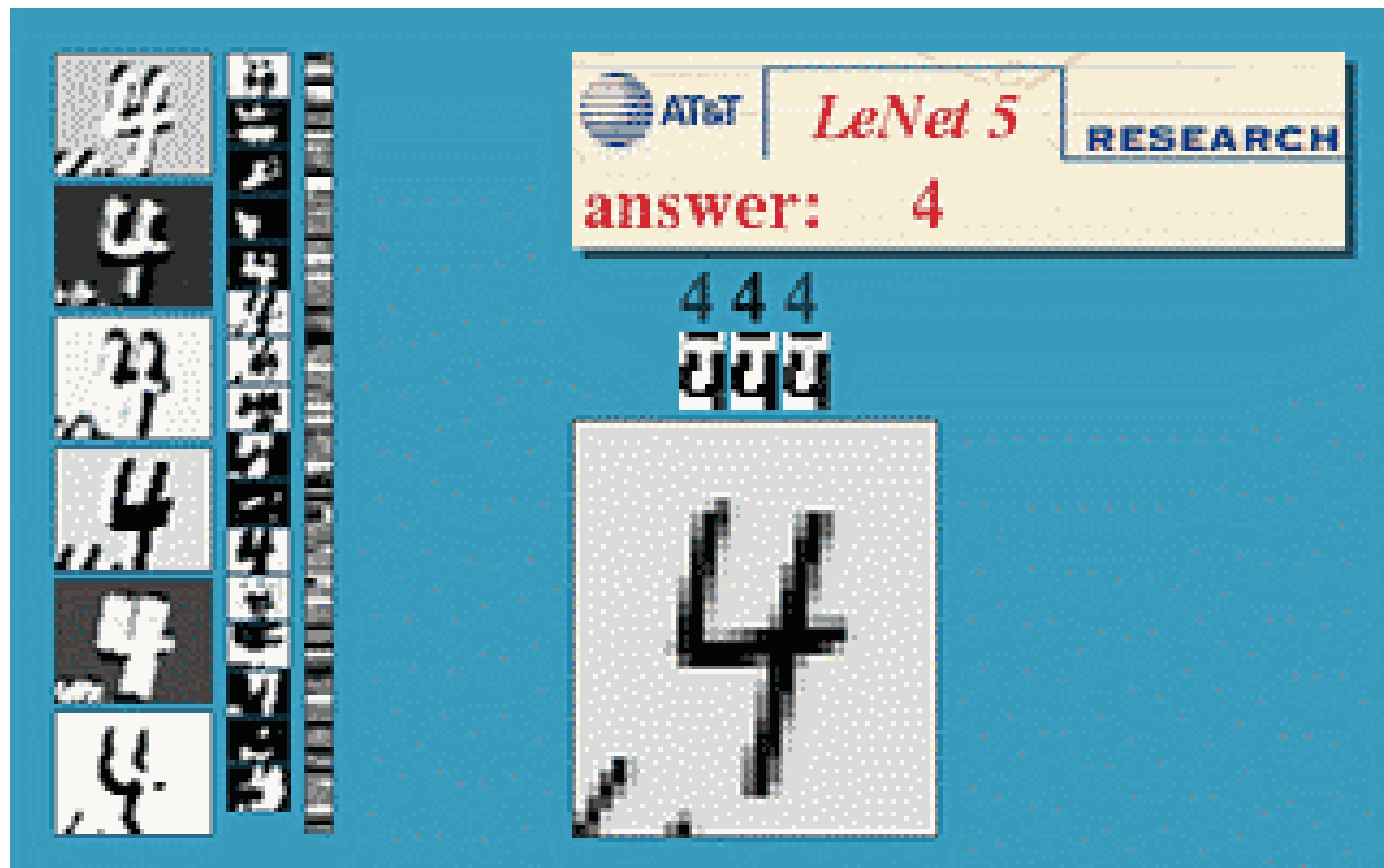
$$a^{(1)} = x$$

$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$

$$a^{(3)} = g(\Theta^{(2)} a^{(2)})$$

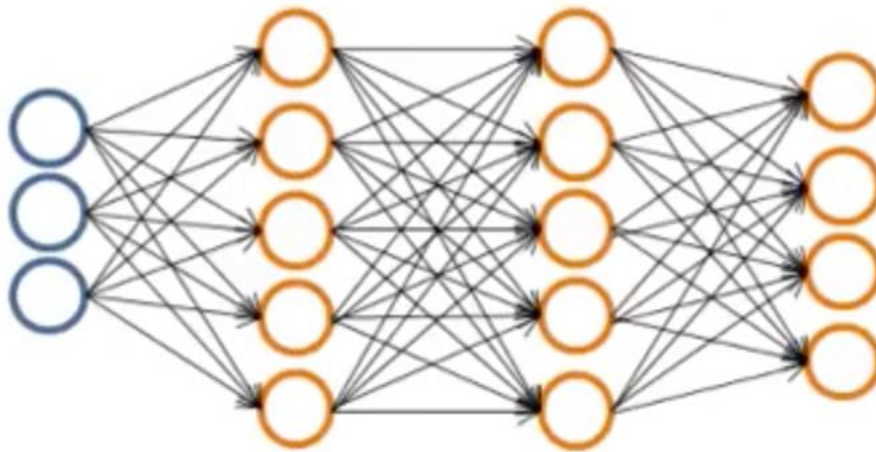
$$h_{\Theta}(x) = a^{(4)} = g(\Theta^{(3)} a^{(3)})$$

Ejemplo



Clasificación multiclase

- Uno vs. Resto



$$h_{\Theta}(x) \in \mathbb{R}^4$$

$$h_{\Theta}(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Clase 1

$$h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Clase 2

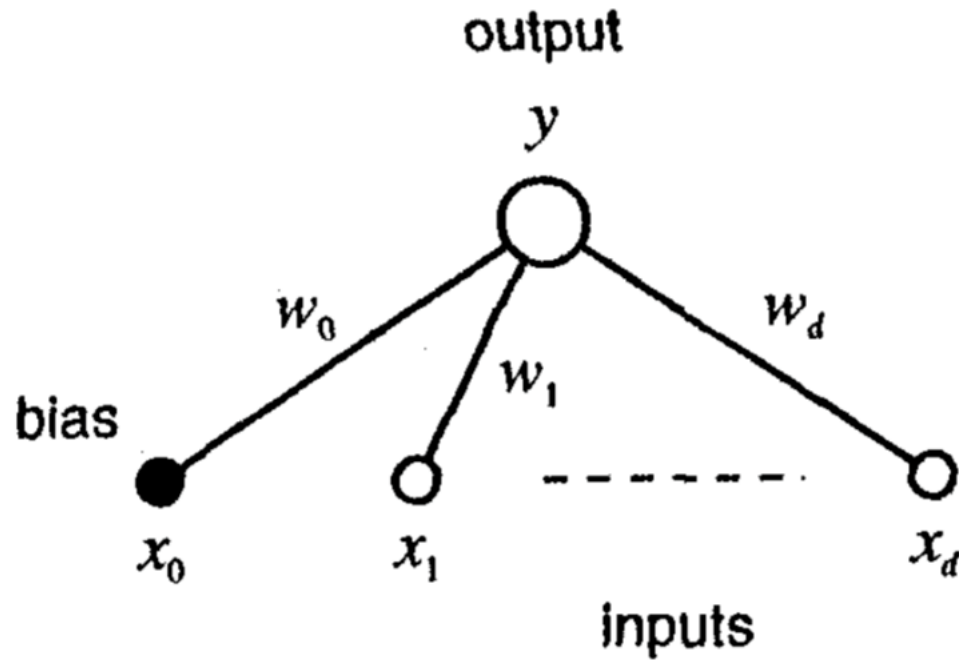
$$h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Clase 3

$$h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Clase 4

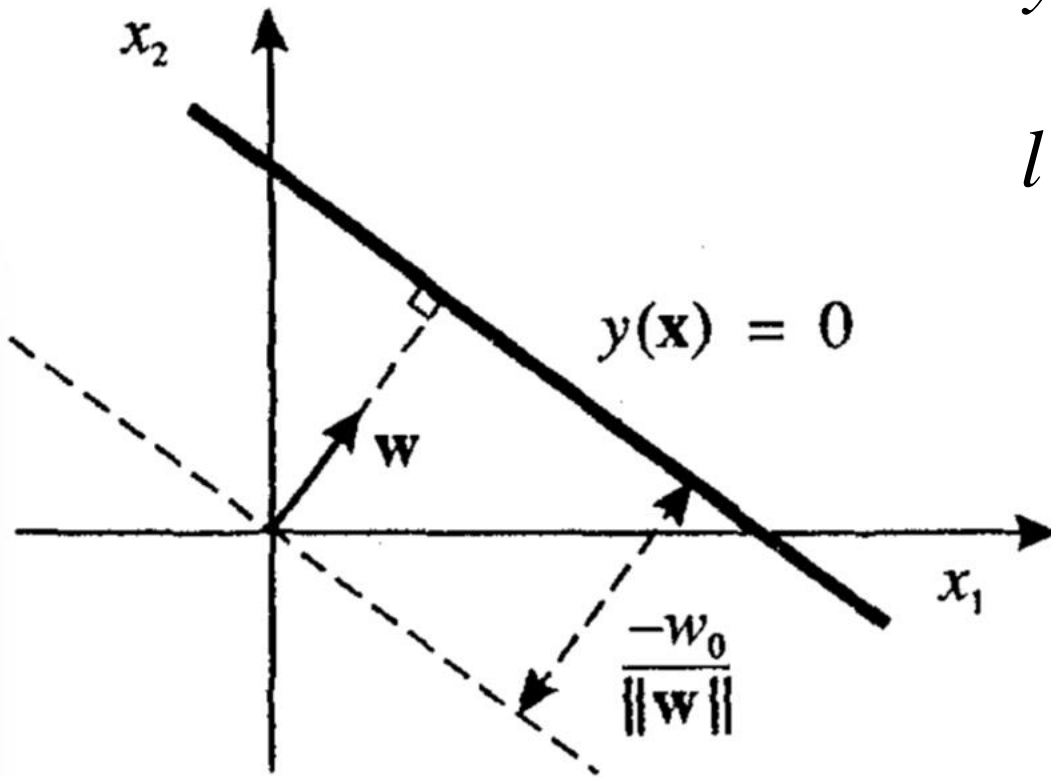
Redes de una capa



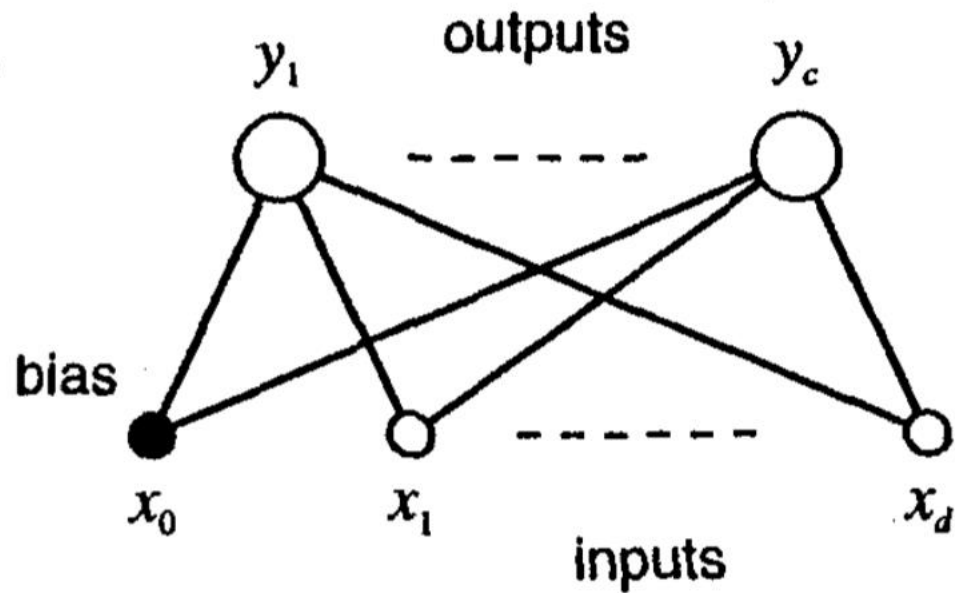
Redes de una capa

$$y(x) = w^T x + w_0$$

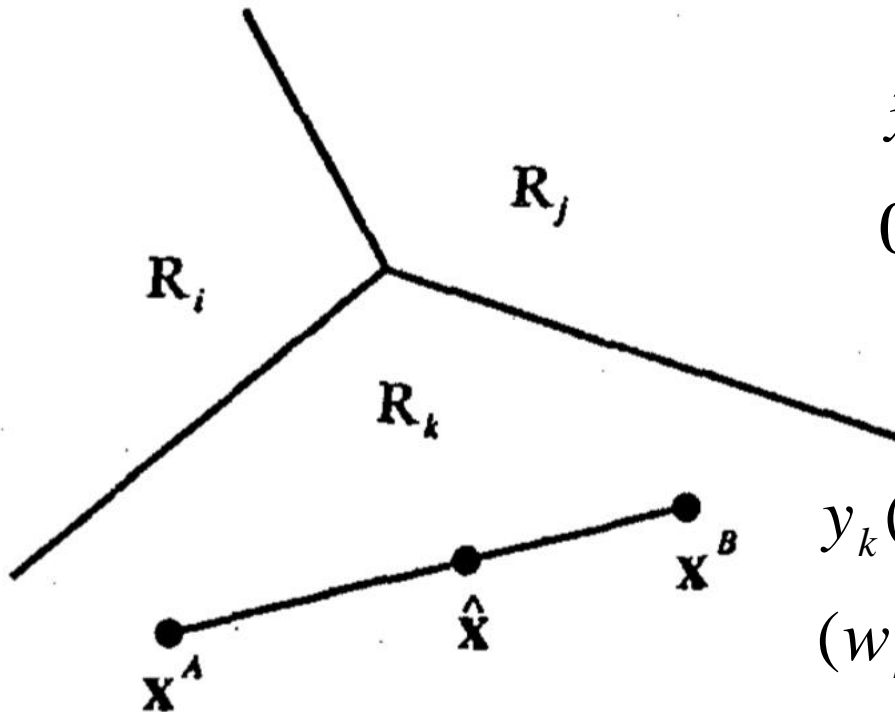
$$l = \frac{w^T x}{\|w\|} = -\frac{w_0}{\|w\|}$$



Redes de una capa



Redes de una capa



$$\hat{x} = \alpha x^A + (1 - \alpha) x^B$$

$$0 \leq \alpha \leq 1$$

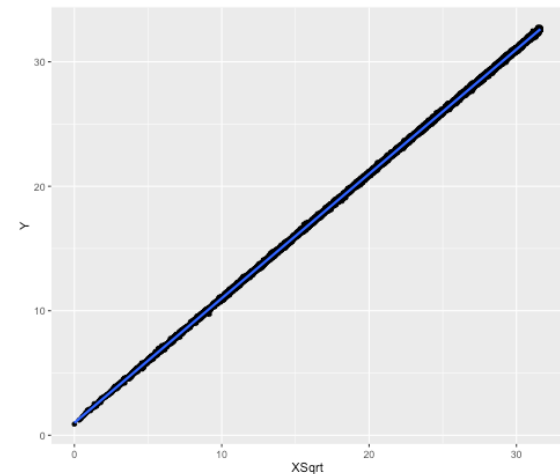
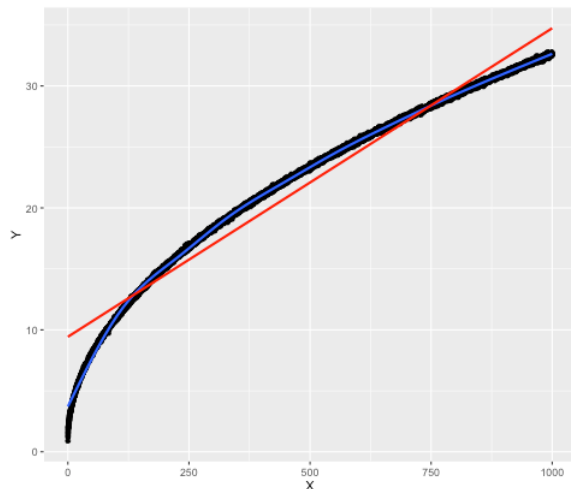
$$y_k(x) = w_k^T x + w_{k0}$$

$$(w_k - w_j)^T x + (w_{k0} - w_{j0}) = 0$$

$$l = -\frac{(w_{k0} - w_{j0})}{\|w_k - w_j\|}$$

Discriminante lineal generalizado

$$h_{k\theta}(x) = \sum_{j=0}^n w_{kj} \phi_j(x)$$



Categorizador logístico

$$p(x | C_k) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} e^{\left\{ -\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) \right\}}$$

$$P(C_1 | x) = \frac{p(x | C_1)P(C_1)}{p(x | C_1)P(C_1) + p(x | C_2)P(C_2)}$$

$$= \frac{1}{1 + e^{-a}}$$

$$= g(a)$$

$$a = \ln \frac{p(x | C_1)P(C_1)}{p(x | C_2)P(C_2)}$$

$$a = w^T x + w_0$$

Perceptrón

- Algoritmo de aprendizaje

Repetir hasta que converja {

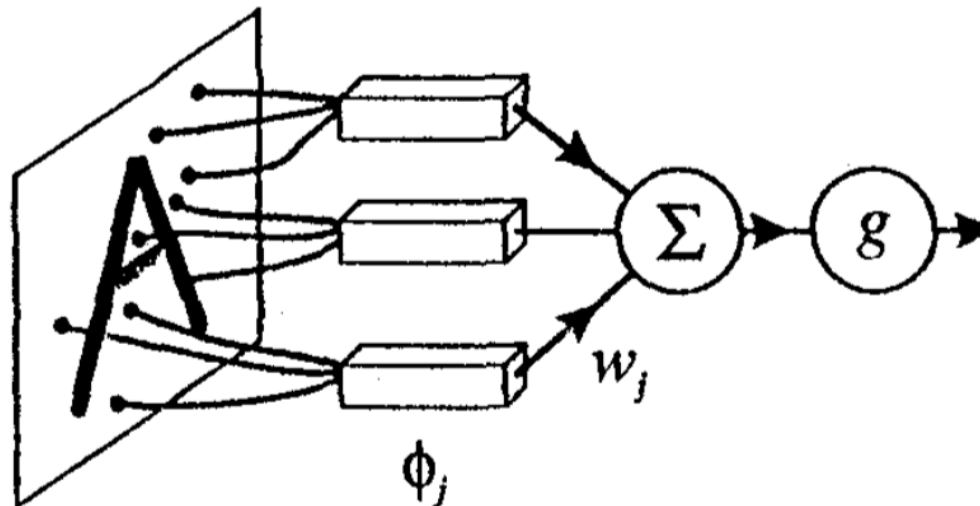
for i=0 to m {

$$\theta_j := \theta_j + (y^{(i)} - g(\theta^T x^{(i)})) \cdot x_j^{(i)}$$

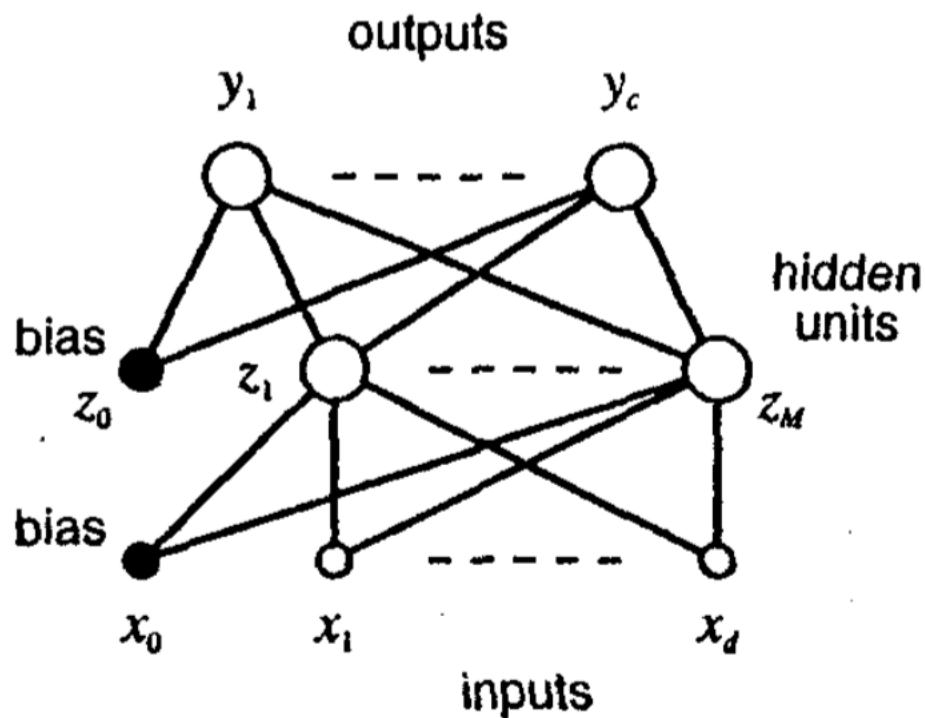
Para j=0 hasta n

}

}



Redes de múltiples capas



Redes de múltiples capas

$$a_j = \sum_{i=0}^d w_{ji}^{(1)} x_i.$$

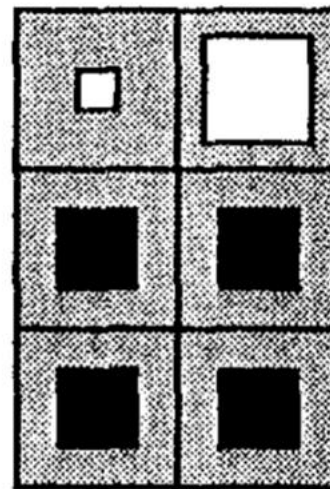
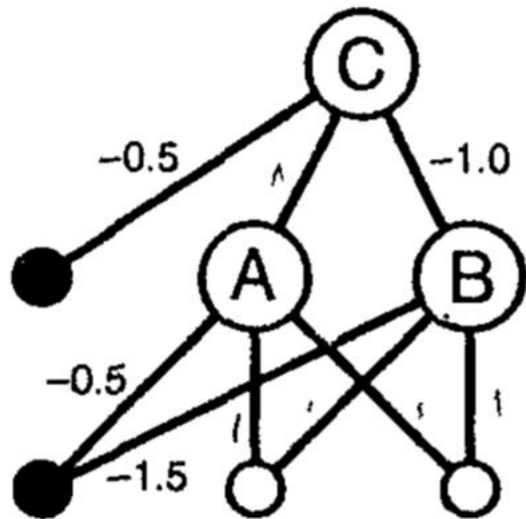
$$z_j = g(a_j).$$

$$a_k = \sum_{j=0}^M w_{kj}^{(2)} z_j$$

$$y_k = \tilde{g}(a_k).$$

$$y_k = \tilde{g} \left(\sum_{j=0}^M w_{kj}^{(2)} g \left(\sum_{i=0}^d w_{ji}^{(1)} x_i \right) \right).$$

Redes de múltiples capas



A



B



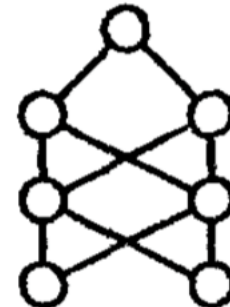
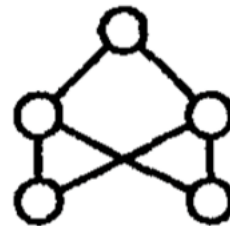
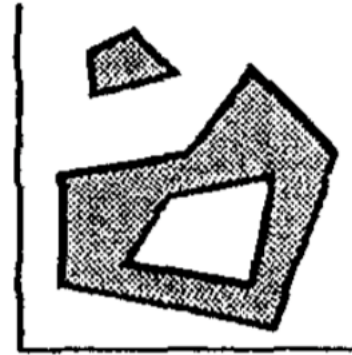
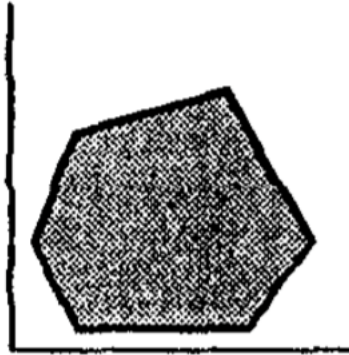
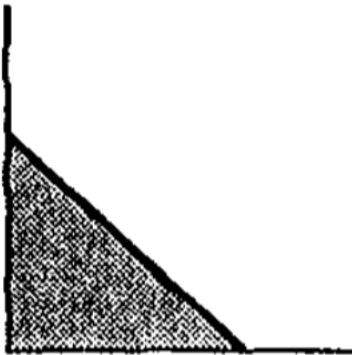
C

biases

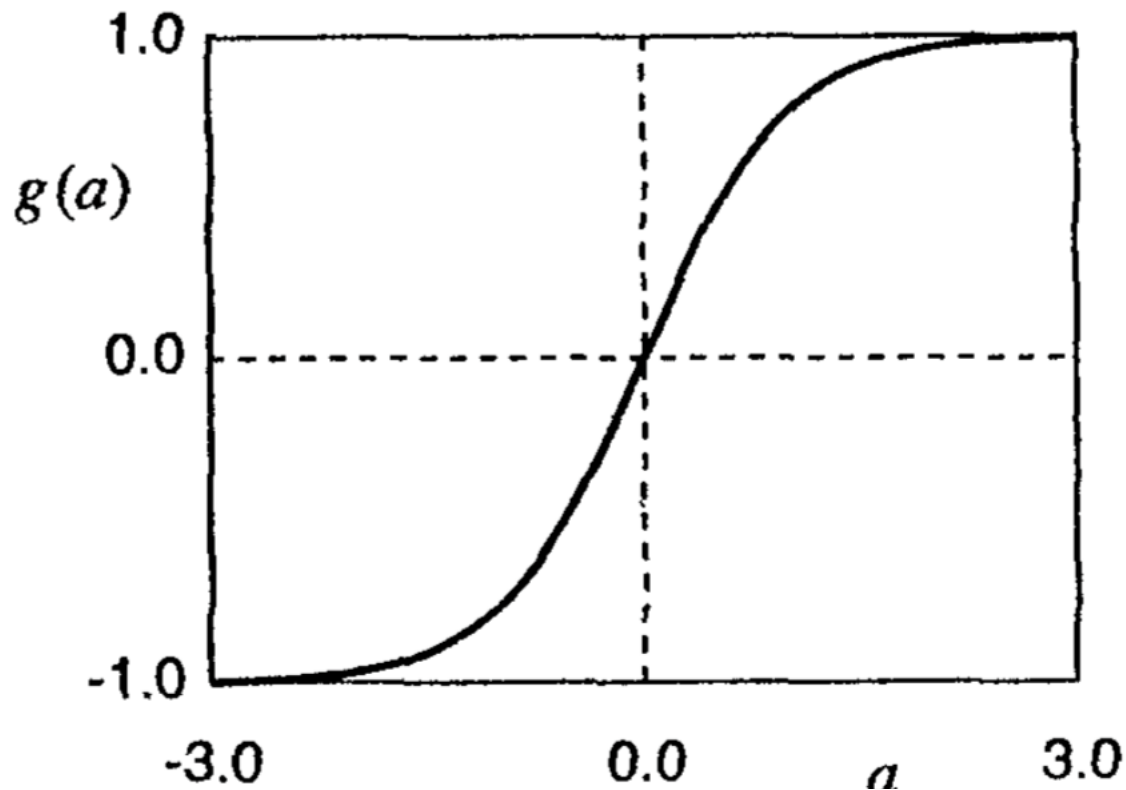
weights

weights

Redes de múltiples capas



Redes de múltiples capas



$$g(a) \equiv \tanh(a) \equiv \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

Descenso por gradiente (incremental)

- Algoritmo de descenso por gradiente para regresión lineal

Repetir hasta que converja {

for i=1 to m {

$$\theta_j := \theta_j - \alpha \cdot (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad \text{Para } j=0 \text{ hasta } n$$

}

}

Back propagation

$$a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial a_j}{\partial w_{ji}} = z_i.$$

$$E = \sum_n E^n \quad \text{n datos de entrenamiento}$$

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j z_i.$$

$$E^n = E^n(y_1, \dots, y_c).$$

$$\delta_k \equiv \frac{\partial E^n}{\partial a_k} = g'(a_k) \frac{\partial E^n}{\partial y_k}$$

Usando la regla de la cadena

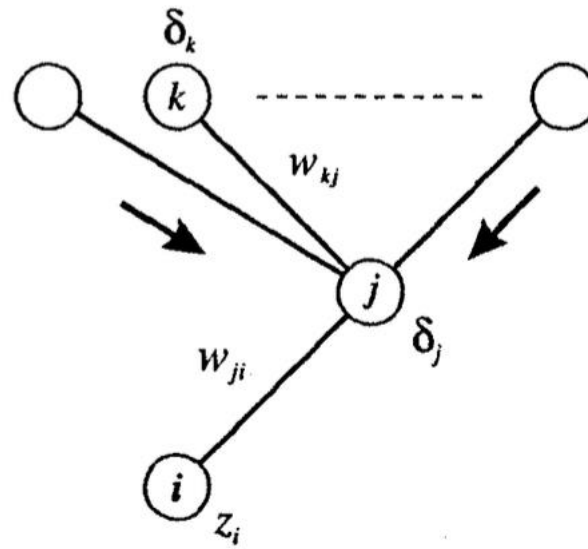
$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}.$$

Usando la regla de la cadena

$$\delta_j \equiv \frac{\partial E^n}{\partial a_j} = \sum_k \frac{\partial E^n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

$$\delta_j \equiv \frac{\partial E^n}{\partial a_j}$$

Back propagation



Fórmula de retro-propagación

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k$$

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E^n}{\partial w_{ji}}.$$

Back propagation

- Propagación hacia delante de las entradas
- Evaluar todas las δ_k de las neuronas de salida
- Retro-propaga las δ 's usando la fórmula de retro-propagación
- Se usa $\frac{\partial E^n}{\partial w_{ji}} = \delta_j z_i$ para evaluar las derivadas

Back propagation

- Ejemplo (salidas lineales, capas ocultas logísticas)

$$g(a) \equiv \frac{1}{1 + \exp(-a)}.$$

$$g'(a) = g(a)(1 - g(a)).$$

$$E^n = \frac{1}{2} \sum_{k=1}^c (y_k - t_k)^2$$

$$\delta_k = y_k - t_k$$

$$\delta_j = z_j(1 - z_j) \sum_{k=1}^c w_{kj} \delta_k$$

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j x_i, \quad \frac{\partial E^n}{\partial w_{kj}} = \delta_k z_j.$$

Actualización de la primera capa (en línea)

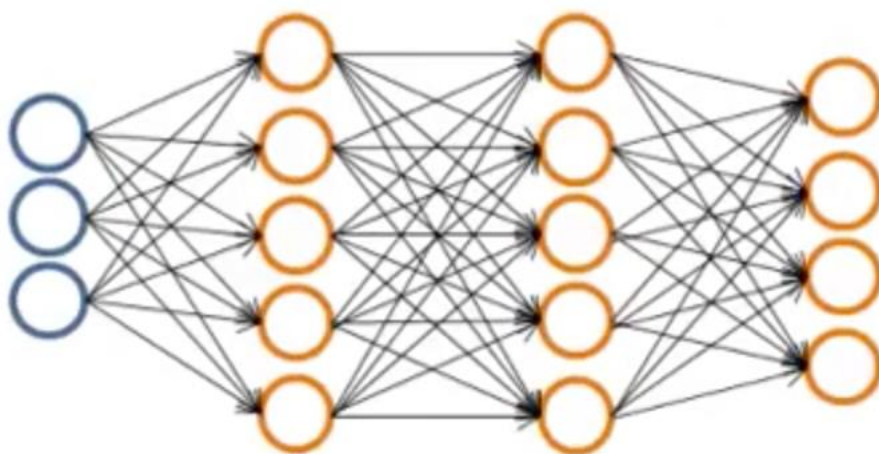
$$\Delta w_{ji} = -\eta \delta_j x_i$$

Actualización de la primera capa (por lotes)

$$\Delta w_{ji} = -\eta \sum_n \delta_j^n x_i^n$$

Clasificación multiclase

- Softmax



$$h_{\Theta}(x) \in \mathbb{R}^4$$

$$h_{\Theta}(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Clase 1

$$h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Clase 2

$$h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

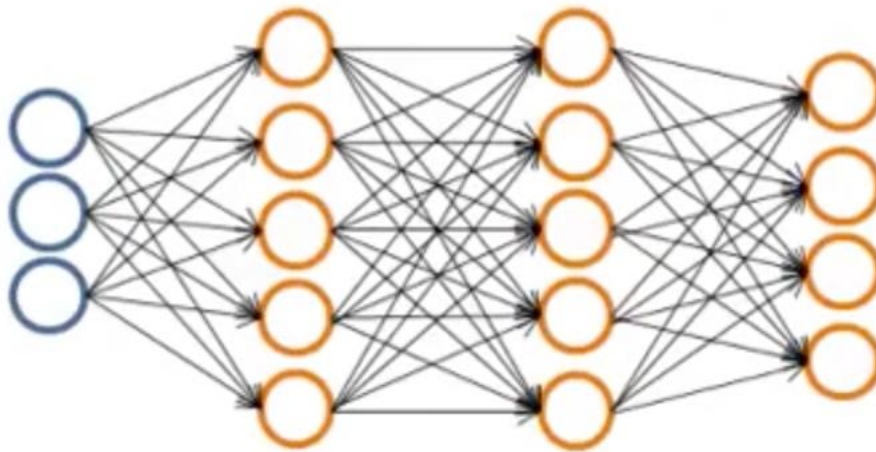
Clase 3

$$h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Clase 4

Clasificación multiclase

- Softmax



$$h_{\Theta}(x) \in \mathbb{R}^4$$

$$\sigma(x^T w)_j = P(y = j | x) = \frac{e^{x^T w_j}}{\sum_{i=1}^K e^{x^T w_i}} \quad \text{for } j = 1, \dots, K$$

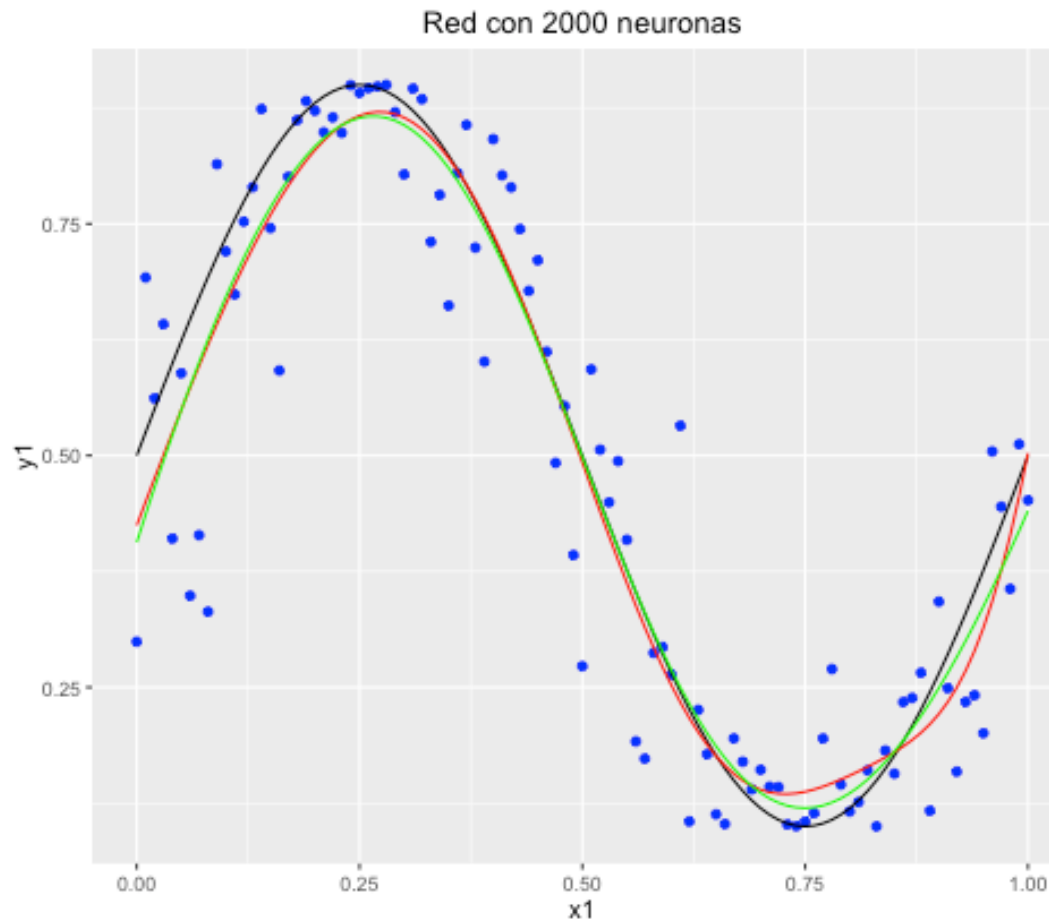
$$\frac{\partial \mathcal{E}^n}{\partial a_k} = y_k - t_k$$

Regularización

Utilizamos el mismo regularizador de las regresiones, sumando el la suma de los cuadrados de los pesos a nuestra función de costo.

$$\Delta w_{ji} = -\eta \delta_j x_i - \eta \lambda w_{ji}$$

Regularización



Momento

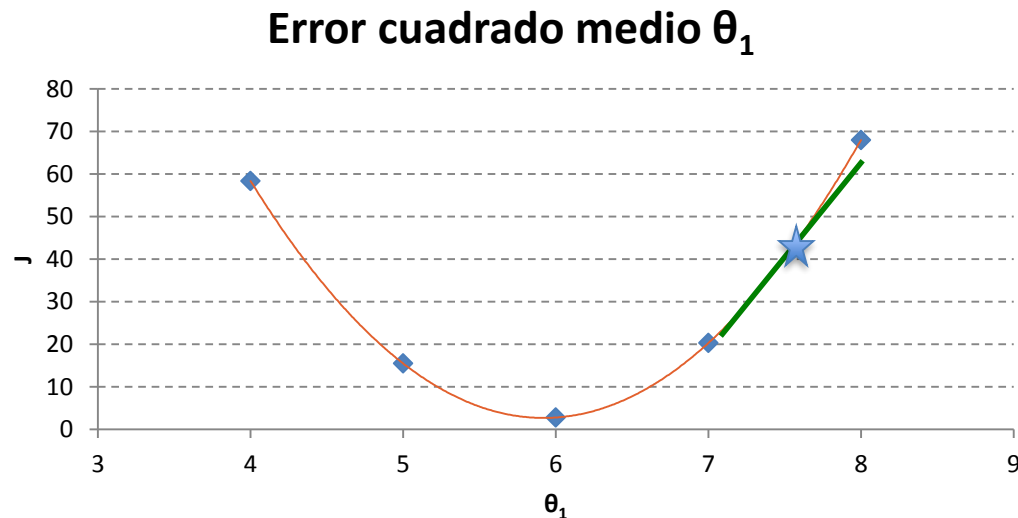
Las funciones de error muchas veces tienen mesetas, regiones donde la pendiente es muy pequeña.

Momento permite aprender más rápido cuando existen mesetas

$$\Delta w_{ji} = -\eta \delta_j x_i + \alpha \Delta w_{ji}$$

Racional detrás del back propagation

- Error cuadrado medio (sólo θ_1)



$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

A red arrow points from the θ_1 argument in the function $J(\theta_0, \theta_1)$ to the left, indicating the direction of the update.

Racional detrás del back propagation

- Consideremos la expresión:

$$e=(a+b)*(b+1)$$

- Podemos reescribirla como

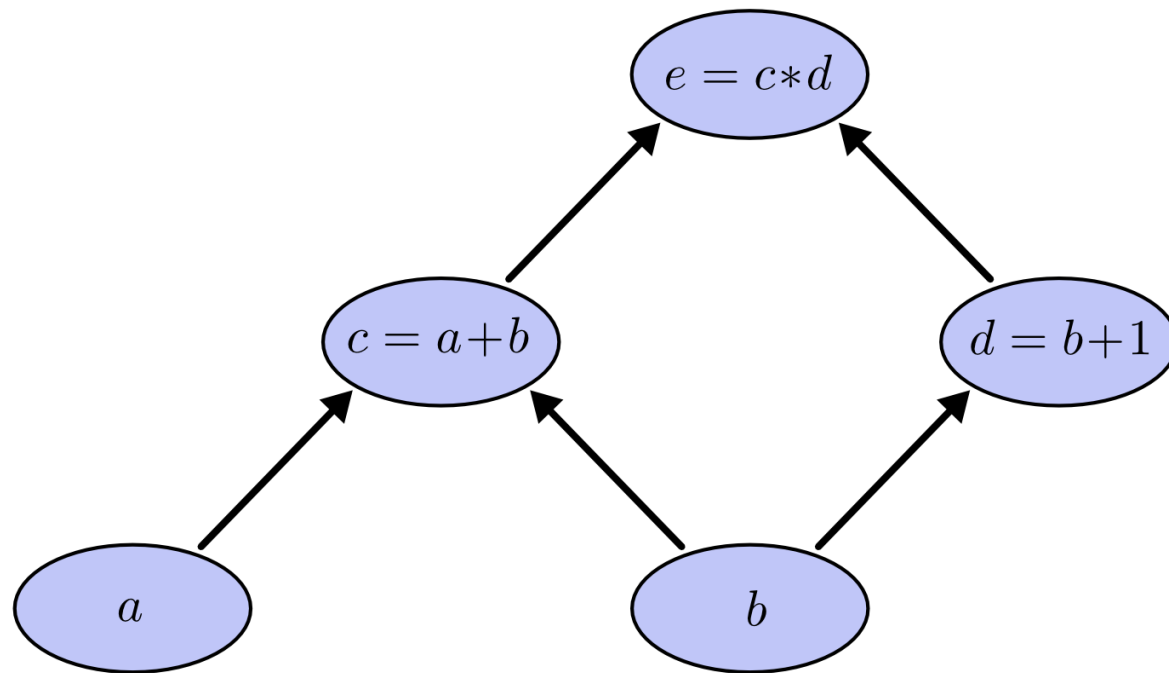
$$c=a+b$$

$$d=b+1$$

$$e=c*d$$

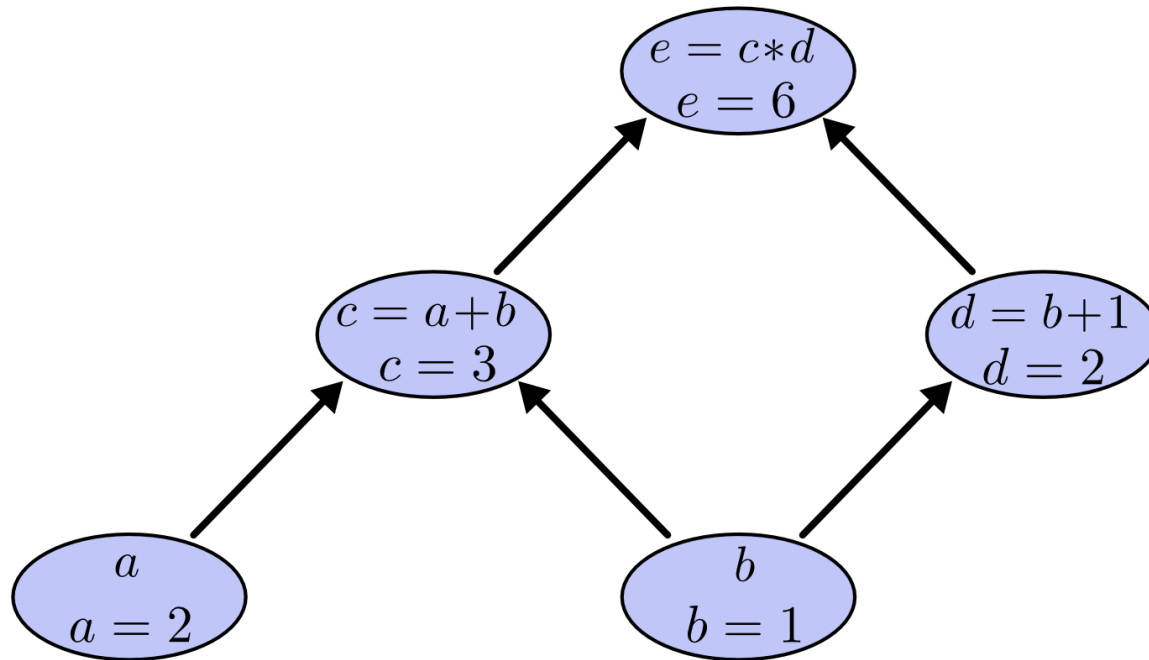
Racional detrás del back propagation

- Podemos representar la expresión mediante el siguiente grafo computacional:



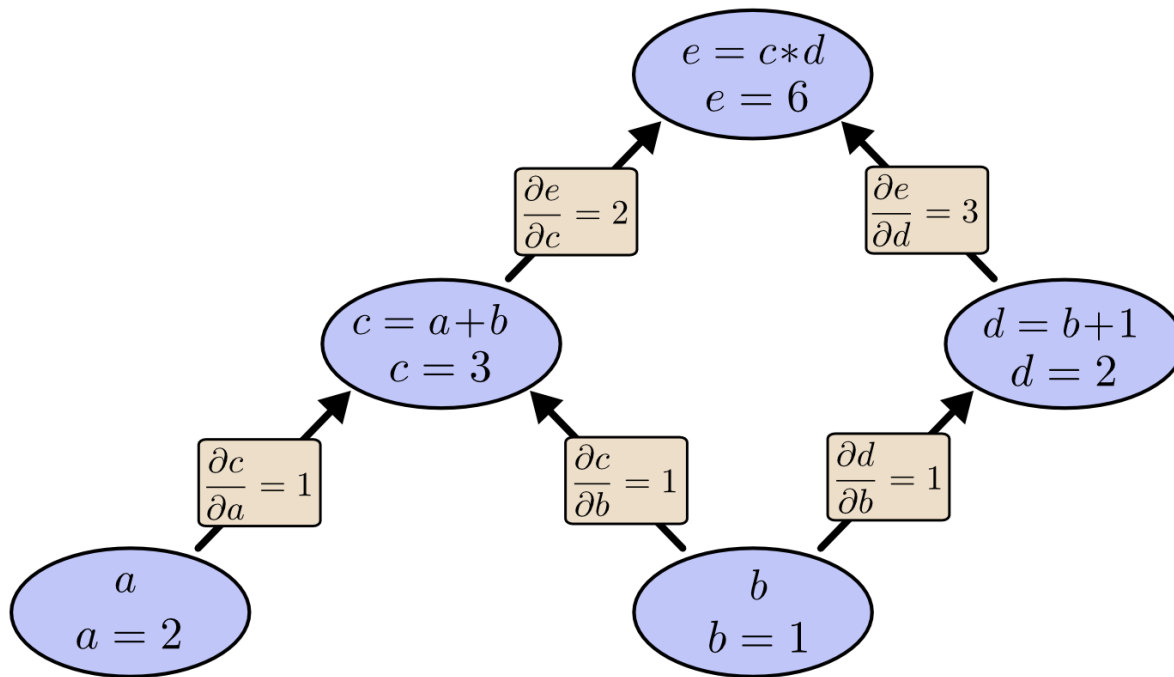
Racional detrás del back propagation

- Podemos evaluar la expresión asignando valores a las variables (por ejemplo $a=2$ y $b=1$):



Racional detrás del back propagation

- Para entender las derivadas en el grafo debemos analizar las derivadas en las aristas:



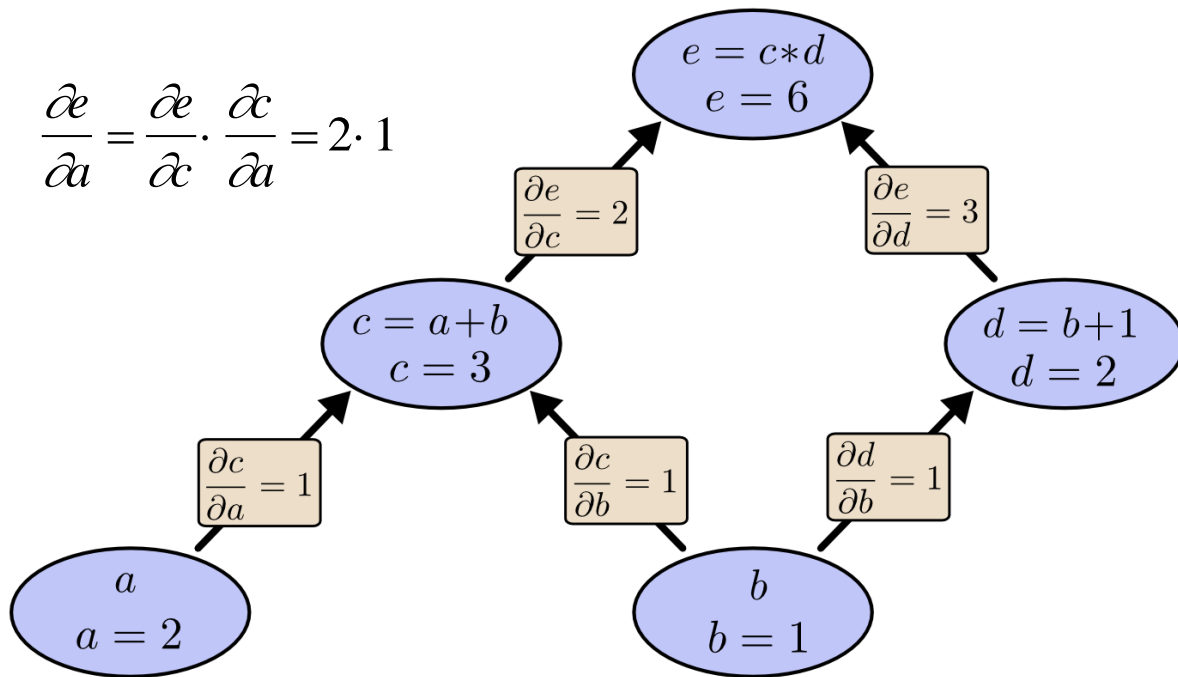
Reglas de la suma y el producto:

$$\frac{\partial}{\partial a}(a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a}$$

$$\frac{\partial}{\partial u}uv = u \frac{\partial v}{\partial u} + v \frac{\partial u}{\partial u}$$

Racional detrás del back propagation

- Para entender las derivadas en el grafo debemos analizar las derivadas en las aristas:



$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a} = 2 \cdot 1$$

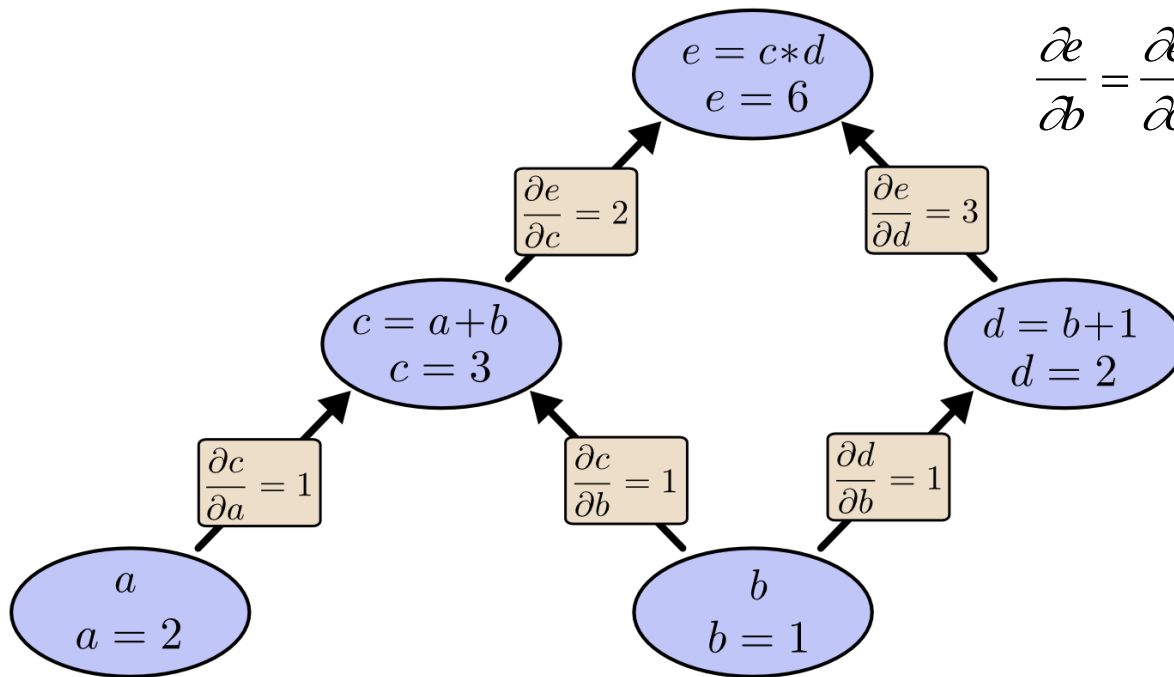
Reglas de la suma y el producto:

$$\frac{\partial}{\partial a}(a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a}$$

$$\frac{\partial}{\partial u}uv = u \frac{\partial v}{\partial u} + v \frac{\partial u}{\partial u}$$

Racional detrás del back propagation

- Para entender las derivadas en el grafo debemos analizar las derivadas en las aristas:



$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b} = 2 \cdot 1 + 3 \cdot 1$$

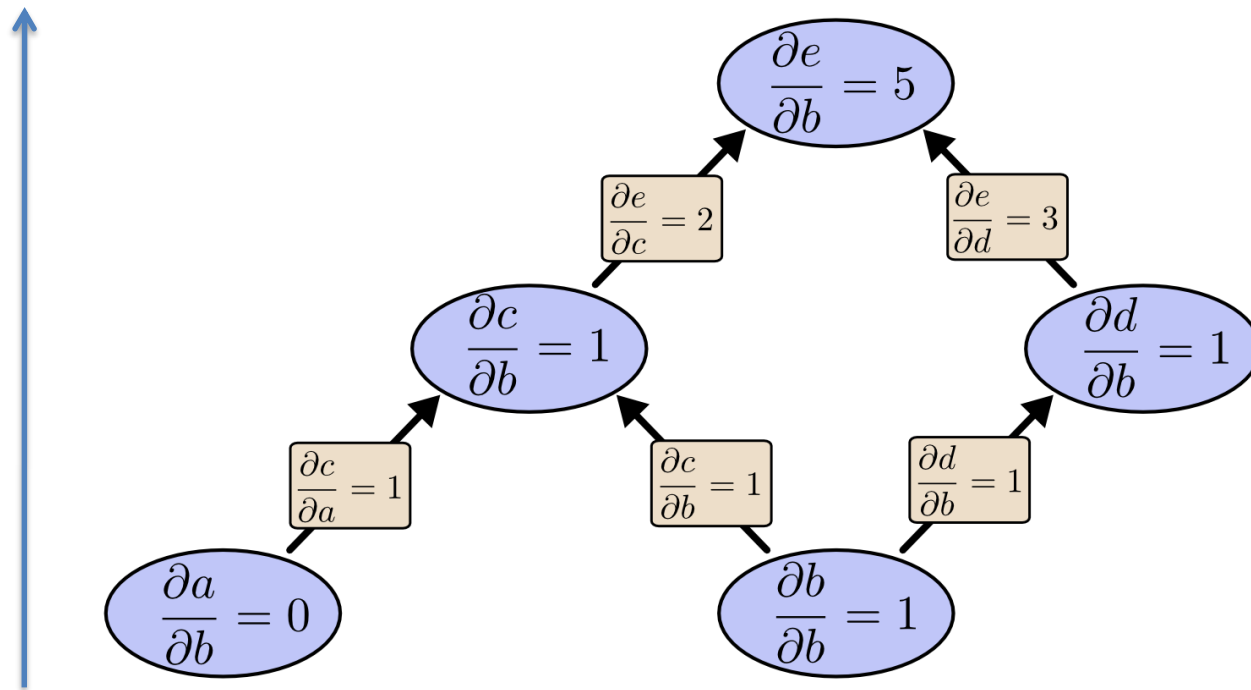
Reglas de la suma y el producto:

$$\frac{\partial}{\partial a}(a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a}$$

$$\frac{\partial}{\partial u}uv = u \frac{\partial v}{\partial u} + v \frac{\partial u}{\partial u}$$

Racional detrás del back propagation

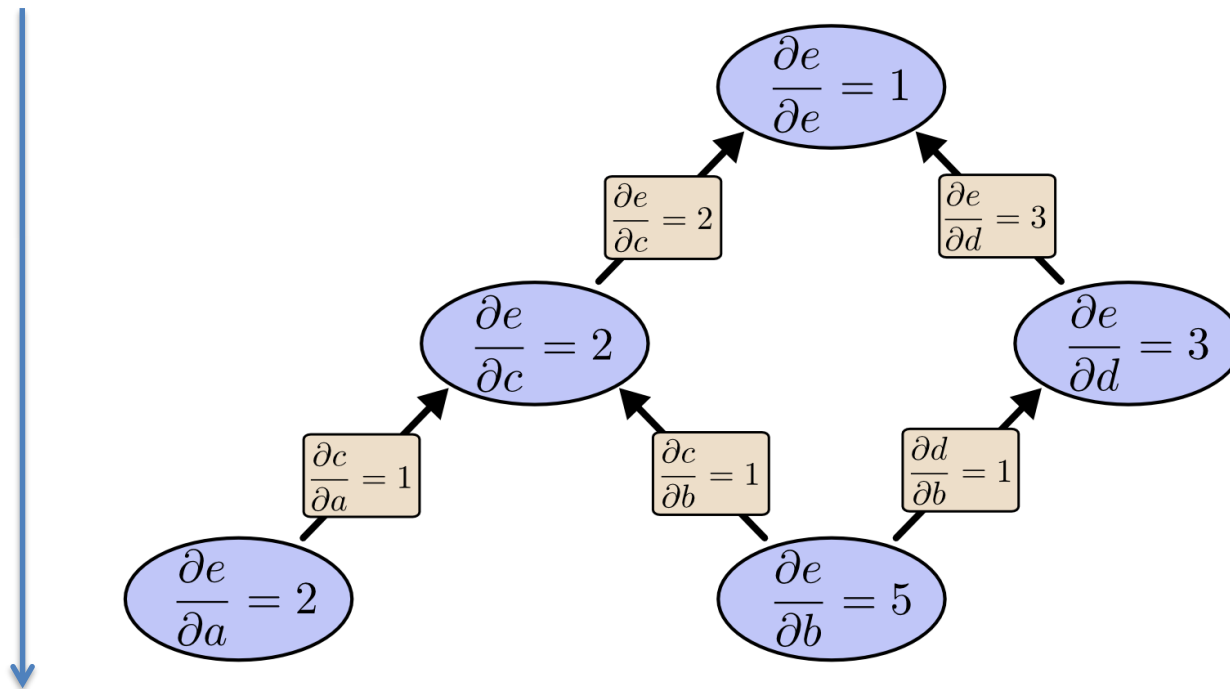
- Derivación de b hacia delante, obtenemos la derivada de la salida con respecto a b



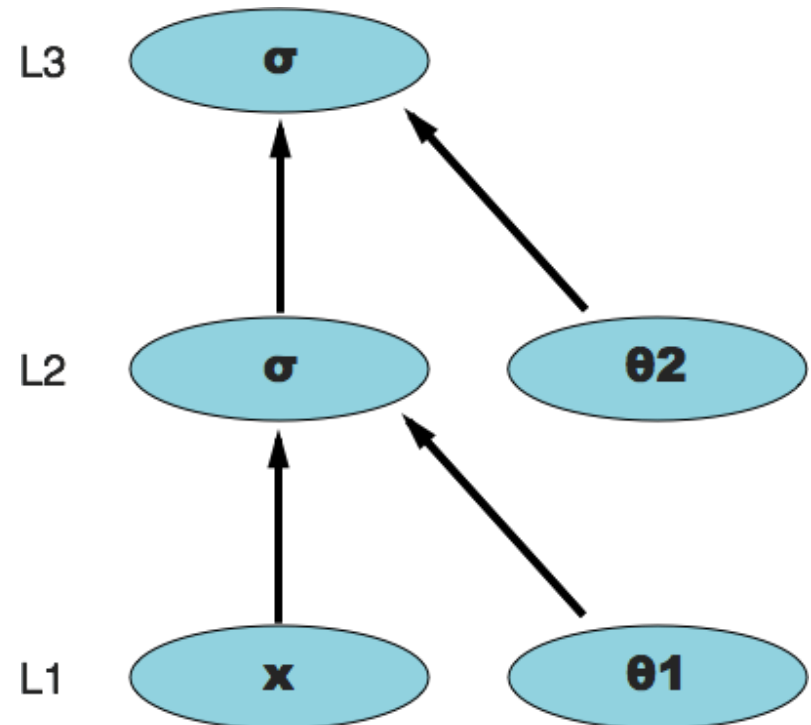
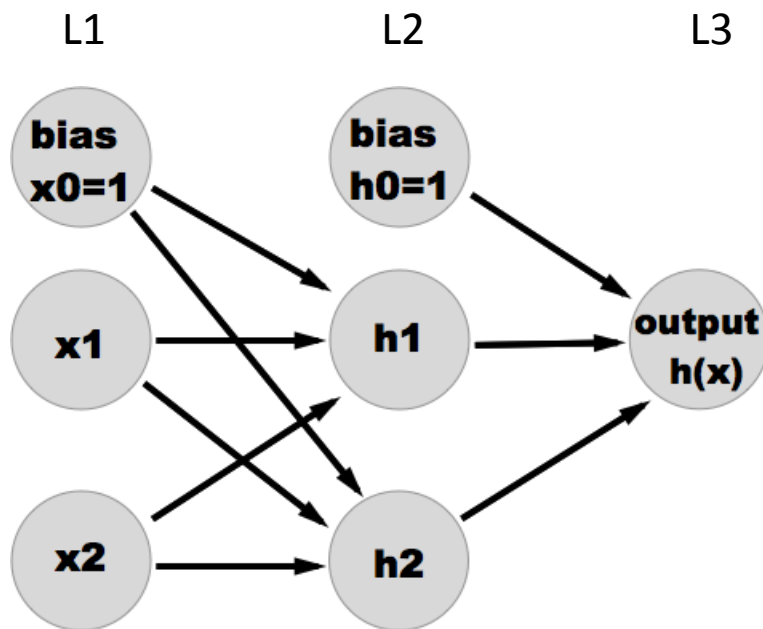
Repetir para cada entrada

Racional detrás del back propagation

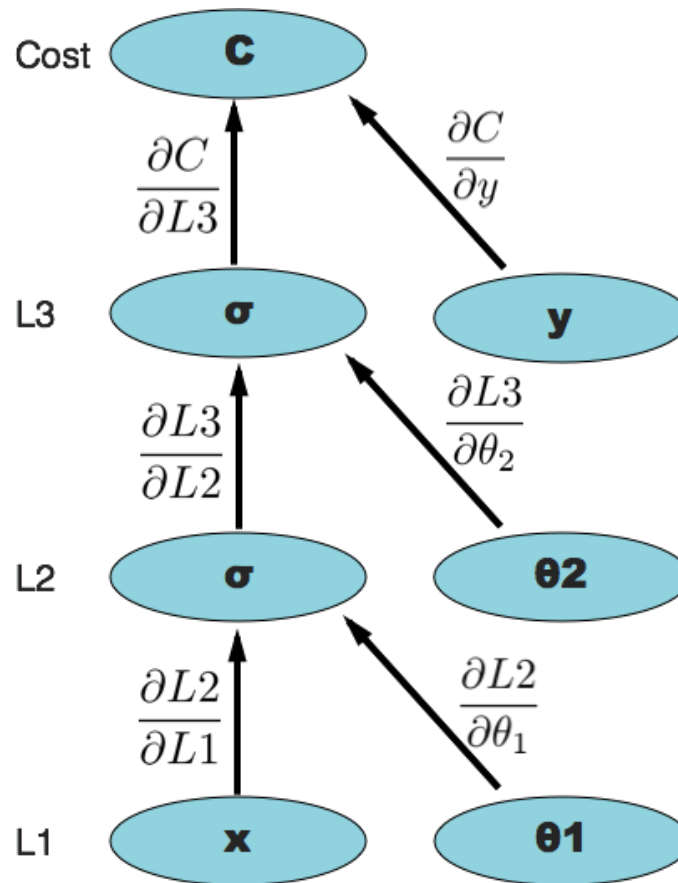
- Derivación de e hacia atrás, obtenemos la derivada de la salida con respecto a todos



Racional detrás del back propagation

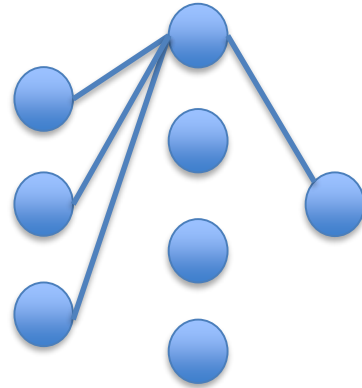


Racional detrás del back propagation



Importancia relativa de las variables de entrada

- Algoritmo de Garsón



$$RI_x = \sum_y^m \frac{|w_{xy} w_{yz}|}{\sum_{x'}^n |w_{x'y} w_{yz}|}$$

- n = dimensión del vector de entrada
- m = neuronas en la capa oculta

Inicialización de pesos

- Hallazgos empíricos indican que para funciones de activación logística o tangente hiperbólica, se obtienen tasas mas bajas de error y menor tiempo de convergencia si los pesos se inicializan de la siguiente forma:

$$W \sim U \left[-\frac{\sqrt{6}}{n^{(l)} + n^{(l+1)}}, \frac{\sqrt{6}}{n^{(l)} + n^{(l+1)}} \right]$$

$n^{(l)}$ número de entradas hacia W

$n^{(l+1)}$ número de salidas desde W