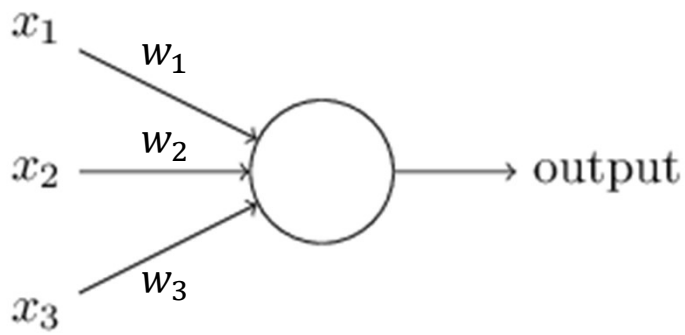
Redes neuronales

Perceptron

Perceptron

Representación matemática de una neurona

Entradas binarias



Salida binaria

Pesos expresan la importancia de una entrada dada para generar una salida. Existe una regla para la salida:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Umbral permite a la neurona activarse o inhibirse

Perceptron

La regla del perceptrón puede ser escrita en una forma distinta:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Donde $b = -threshold$, comúmente conocido como el bias.

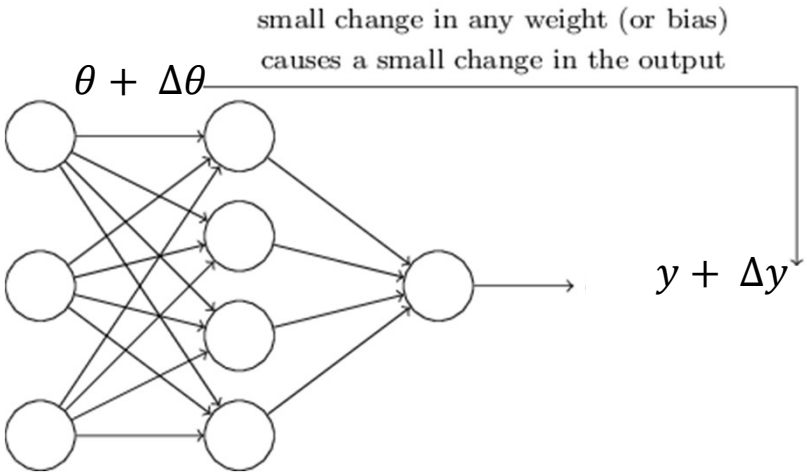
El preceptrón se activa solo si el producto interno entre pesos y entradas es más grande que el bias. **El bias es el parámetro que representa que tan fácil es activar una neurona.**

Aprendizaje de perceptrones

Dada una red neuronal T, la salida es una función de la entrada x y el conjunto de parámetros θ. El conjunto de parámetros son los pesos y biases.

$$y = T(x, \theta)$$

La red es capaz de aprender si y solo si es posible medir cuan grande es el cambio en la salida si nosotros cambiamos los parámetros.

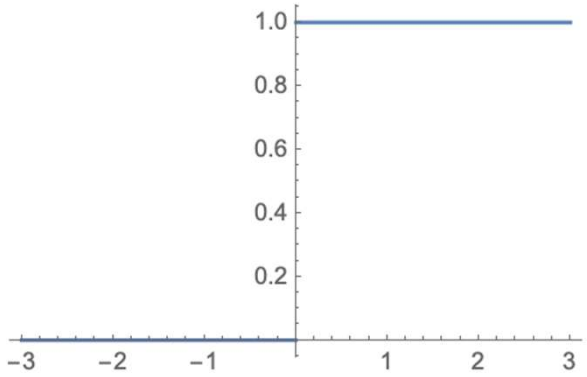


El problema con el perceptrón es que un pequeño cambio en los parámetros puede causar un cambio grande en la salida (de 0 a 1 o viceversa)

Razón: umbralización simple es una función no suave.

Aprendizaje de perceptrones

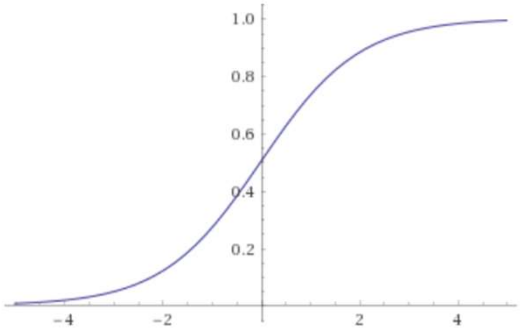
Razón: umbralización simple es una función no suave



Solución: Reemplazar la función step con una función suave de comportamiento similar

Función Sigmoide

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

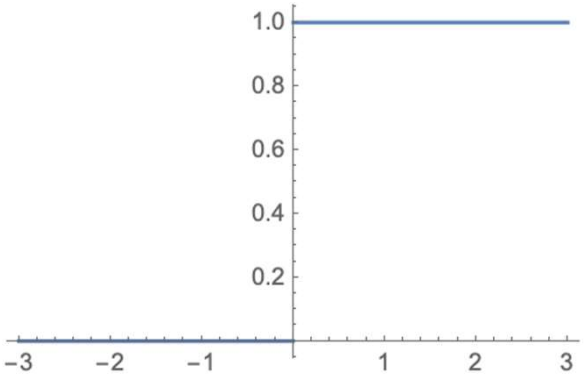


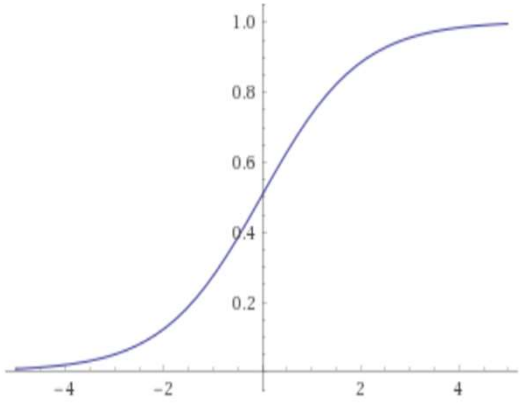
Así que la salida de la neurona es:

$$\sigma(w.z + b) = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

Aprendizaje de perceptrones

Razonamiento: ambas funciones lucen muy distintas matemáticamente, sin embargo tienen un comportamiento muy similar





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

- Cuando z es un número positivo grande, entonces $e^{-z} \approx 0$, $\sigma(z) \approx 1$ — **Like step function**
- Cuando z es un número negativo grande, entonces $e^{-z} \to \infty$, $\sigma(z) \approx 0$ — **Like step function**
- La diferencia es cuando z tiene valores cercanos a cero

$\sigma(z)$ es suave alrededor de cero, también diferenciable

Resumen

- Perceptrones
- Función Sigmoide y propiedades
- Multilayer perceptron

Redes neuronales

Gradiente descendiente Parte 1

Gradiente descendiente

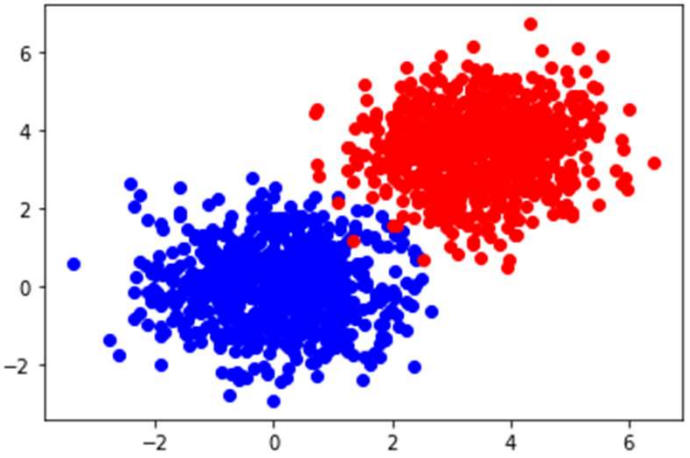
Explicaremos el aprendizaje con un ejemplo. Clasificación binaria de puntos en 2D

$$D = \{(x_1, x_2, y)\}$$

Donde:

- (x_1, x_2) son coordenadas en 2D
- y es una etiqueta que indica si el punto pertenece a la clase 0 o a la clase 1

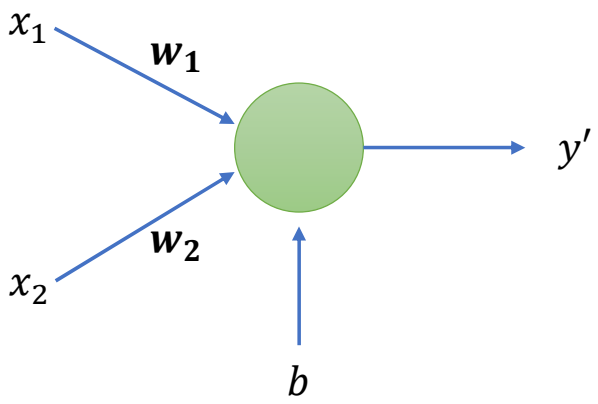
Class 0



Class 1

Gradiente descendiente

Para este problema, usaremos una sola neurona sigmoide con los siguientes componentes:



Donde:

$$z = \boldsymbol{w_1}x_1 + \boldsymbol{w_2}x_2 + b$$

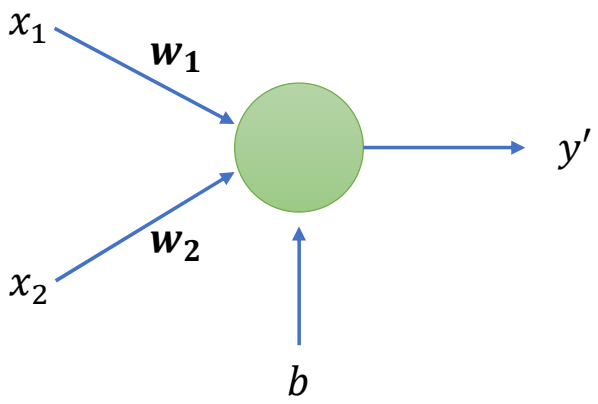
$$y' = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Dado un sample (x_1, x_2), la red producirá una salida y'
Cómo podemos comparar esta salida con la salida deseada y?

$$C = \frac{1}{2}(y' - y)^2$$

Gradiente descendiente

Para este problema, usaremos una sola neurona sigmoide con los siguientes componentes:



Donde:

$$z = \boldsymbol{w_1}x_1 + \boldsymbol{w_2}x_2 + b$$

$$y' = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Dado un sample (x_1, x_2), la red producirá una salida y'
Cómo podemos comparar esta salida con la salida deseada y?

$$C(\boldsymbol{w_1}, \boldsymbol{w_2}, b) = \frac{1}{2}(y'(x_1, x_2) - y)^2$$

Costo es una función de los parámetros!

Gradiente descendiente

Con n muestras de entrenamiento, el costo es

$$C(\boldsymbol{w_1}, \boldsymbol{w_2}, b) = \frac{1}{2n} \sum_{(x_1, x_2)} (y'(x_1, x_2) - y)^2$$

Función de costo cuadrático o
Mean Squared Error o MSE

Propiedades de la función de costo:

- $C(\boldsymbol{w_1}, \boldsymbol{w_2}, b)$ es no negativa
- $C(\boldsymbol{w_1}, \boldsymbol{w_2}, b) \approx 0$ cuando y' es muy similar a y

Una buena red neuronal:

- Tiene pesos y biases tal que $C \approx 0$

Minimizar C con respecto a parámetros $\boldsymbol{w_1}, \boldsymbol{w_2}, b$

A mala red neuronal:

- Tiene pesos y biases tal que C es grande

Gradiente descendiente al rescate!

Gradiente descendiente

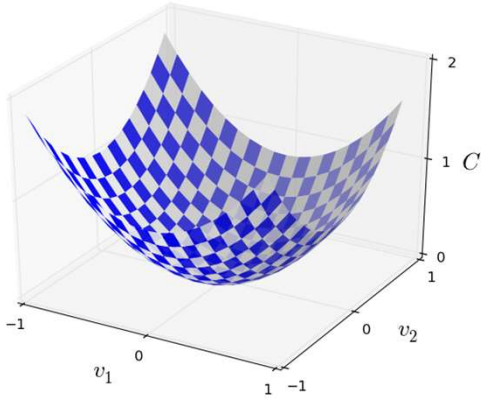
Aprendizaje como un problema de optimización:

Encontrar pesos y biases los cuales minimizan la función de costo cuadrático

Cómo podemos hacer eso? Veamos alguna intuición

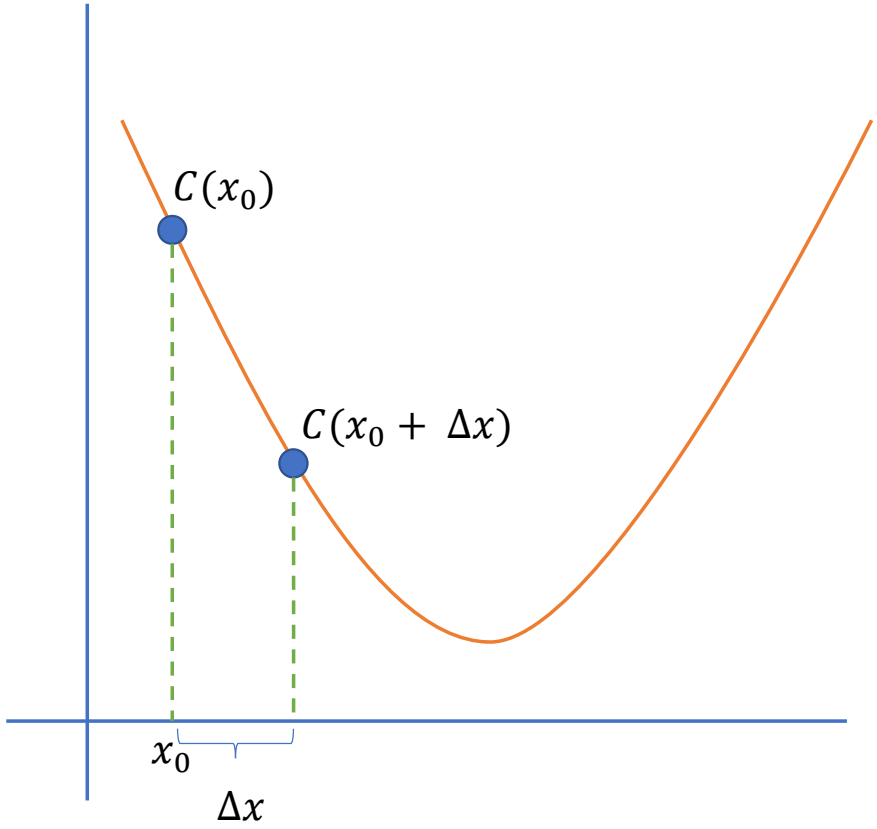
Supongamos que tenemos una función de valores reales de muchas variables $C(v)$, definida como

$$C\colon R^n \to R$$



El problema es encontrar el mínimo global de la función de optimización
Podríamos intentar analíticamente con cálculo, pero recuerda que las redes tienen millones de parámetros

Gradiente descendiente



Derivada de C en el punto x_0

$$\frac{\partial C}{\partial x} = \frac{C(x_0 + \Delta x) - C(x_0)}{\Delta x} = \frac{\Delta C}{\Delta x}$$

$$\Delta C = \frac{\partial C}{\partial x} \Delta x$$

El cambio en la función depende de la derivada de la función y el cambio en los parámetros

Gradiente descendiente

Para generalizar la regla a muchas más variables, la función C puede depender de un conjunto de variables $v = (v_1, v_2, \cdots, v_m)$

$$\Delta C = \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 + \cdots + \frac{\partial C}{\partial v_m}\Delta v_m$$

Necesitamos escoger $\Delta v_1, \Delta v_2, \cdots, \Delta v_m$ tal que ΔC es negativo (un cambio negativo en C)

Escribimos la ecuación en una forma conveniente:

$$\Delta C \approx \nabla C \cdot \Delta v$$

$$\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}, \cdots, \frac{\partial C}{\partial v_m}\right)^T \qquad \Delta v = (\Delta v_1, \Delta v_2, \cdots, \Delta v_m)^T$$

Gradiente descendiente

Para hacer ΔC negativo, escogeremos

$$\Delta v = -\eta \nabla C$$

Note que si reemplazamos en la ecuación

$$\Delta C \approx \nabla C \cdot \Delta v$$

Tendremos $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$

Como $\|\nabla C\|^2 \geq 0$, esto garantiza que $\Delta C \leq 0$

Conclusión: tomando $\Delta v = -\eta \nabla C$ siempre hará que la función C se decremente

Regla de actualización iterativa $v = v - \eta \nabla C$

Para cada parámetro en la función

Gradiente descendiente

El rol del hiper-parámetro η

$$v = v - \eta \nabla C$$

- Si η es muy pequeño — Realizaremos pasos pequeños en la minimización, GD será lento
- Si η es muy grande — Cerca del mínimo, podemos obtener $\Delta C > 0$. Salto alrededor del mínimo

En la práctica, heurísticas empíricas son útiles, las veremos después en el curso!

Resumen

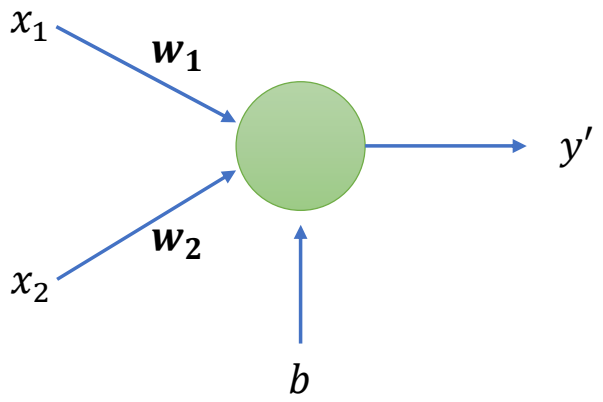
- Gradiente descendiente
- Derivadas de funciones multi-variadas
- Reglas de actualización

Redes neuronales

Gradiente descendente Parte 2

Gradiente descendente en redes neuronales

Regresemos a nuestro problema de clasificación binaria



$$C(\boldsymbol{w_1}, \boldsymbol{w_2}, b) = \frac{1}{2}(y'(x_1, x_2) - y)^2$$

Objetivo

Encontrar $\boldsymbol{w_1}, \boldsymbol{w_2}, b$ que minimiza C

Reglas de actualización

$$\boldsymbol{w_1} = \boldsymbol{w_1} - \eta \frac{\partial C}{\partial \boldsymbol{w_1}} \qquad \boldsymbol{w_2} = \boldsymbol{w_2} - \eta \frac{\partial C}{\partial \boldsymbol{w_2}} \qquad b = b - \eta \frac{\partial C}{\partial b}$$

Gradiente descendente en redes neuronales

Necesitamos encontrar una forma de computar las derivadas parciales. Dada una muestra (x_1, x_2, y), la red neuronal computa la salida como sigue

$$z = \boldsymbol{w_1} x_1 + \boldsymbol{w_2} x_2 + b$$

$$y' = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Finalmente, la función de costo es $C(y') = \frac{1}{2}(y' - y)^2$

Podemos computar las derivadas usando la regla de la cadena

$$\frac{\partial C}{\partial \boldsymbol{w_1}} = \frac{\partial C}{\partial y'} \cdot \frac{\partial y'}{\partial z} \cdot \frac{\partial z}{\partial \boldsymbol{w_1}}$$

Gradiente descendente en redes neuronales

Computemos las derivadas:

$$C(y') = \frac{1}{2}(y' - y)^2$$

- Derivada de C con respecto al valor de salida

$$\frac{\partial C}{\partial y'} = (y' - y)$$

- Derivada de z con respecto a $\boldsymbol{w_1}$

$$z = \boldsymbol{w_1} x_1 + \boldsymbol{w_2} x_2 + b$$

$$\frac{\partial z}{\partial \boldsymbol{w_1}} = x_1$$

Gradiente descendente en redes neuronales

- Derivada de y' con respecto a z

$$y' = \sigma(z) = \frac{1}{1 + e^{-z}}$$

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

$$\sigma'(z) = -1 \cdot (1 + e^{-z})^{-2} \cdot e^{-z} \cdot -1$$

$$\sigma'(z) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

Un poco de álgebra para simplificar la derivada

$$\sigma'(z) = \frac{e^{-z} + 1 - 1}{(1 + e^{-z})^2} \longrightarrow \sigma'(z) = \frac{1}{(1 + e^{-z})} - \frac{1}{(1 + e^{-z})^2}$$

Factorizando

$$\sigma'(z) = \frac{1}{(1 + e^{-z})}\left(1 - \frac{1}{(1 + e^{-z})}\right) \longrightarrow \frac{\partial y'}{\partial z} = \sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$$

Gradiente descendente en redes neuronales

Las fórmulas finales son

$$\frac{\partial C}{\partial \boldsymbol{w_1}} = (y' - y) \cdot \sigma(z)(1 - \sigma(z)) \cdot x_1$$

$$\frac{\partial C}{\partial \boldsymbol{w_2}} = (y' - y) \cdot \sigma(z)(1 - \sigma(z)) \cdot x_2$$

$$\frac{\partial C}{\partial b} = (y' - y) \cdot \sigma(z)(1 - \sigma(z))$$

Algoritmo Gradiente Descendente

Entrada: Conjunto de muestras $\{(x_1, x_2, y)\}$, learning rate η, número de épocas E

Para cada época

Para cada muestra de entrenamiento

- Computar y'
- Computar el costo C
- Computar los gradientes
- Actualizar Parámetros

Computar el costo promedio

Resumen

- Regla de la cadena para computar derivadas parciales
- Algoritmo de aprendizaje

Redes neuronales

Gradiente descendente estocástico

Gradiente descendente estocástico (SGD)

Recordemos la función de costo

$$C = \frac{1}{2n}\sum (y'(x) - y)^2 = \frac{1}{n}\sum_x C_x \qquad C_x = \frac{\|y(x)-y\|^2}{2}$$

Para computar el gradiente ∇C, necesitamos computar los gradientes individuales ∇C_x Operación costosa

SGD idea: Aleatoriamente escoger un pequeño número m de muestras de entrenamiento

- Llamaremos estas muestras $X = \{X_1, X_2, \cdots, X_m\}$ como un *mini-batch*
- Si m es suficientemente grande, entonces

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C \quad \Rightarrow \quad \nabla C \approx \frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m}$$

El gradiente completo puede ser estimado con gradientes de un mini-batch escogido aleatoriamente

Gradiente descendente estocástico (SGD)

Reglas de actualización ahora cambian a

$$\boldsymbol{w_k} = \boldsymbol{w_k} - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial \boldsymbol{w}_k}$$

$$b = b - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial \boldsymbol{w}_k}$$

Algoritmo SGD

Input: Conjunto de muestras $\{(x_1, x_2, y)\}$, learning rate η, número de épocas E, tamaño de mini-batch m

Para cada época

Para cada mini-batch X

- Computar y'
- Computar el costo C
- Computar los gradientes
- Actualizar parámetros

Computar el costo promedio

Resumen

- Aproximación estocástica del error
- Procesamiento de mini-batch