

# I Introducción

Odin Eufracio

# Motivación

Algoritmos clásicos en **aprendizaje máquina** generalmente usan métodos de **optimización**, como descenso de gradiente, para la etapa de **entrenamiento o aprendizaje**.

Las redes neuronales, ***neural networks (NN)***, son **modelos paramétricos** que también usan métodos de optimización en su etapa de aprendizaje.

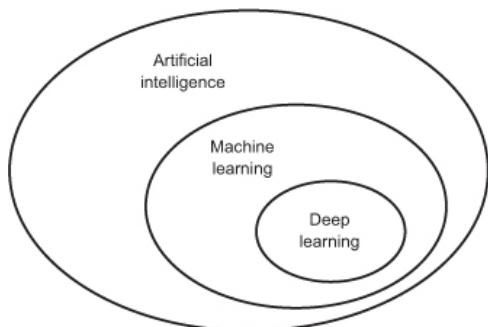
En esta primera sección, implementaremos un modelo de **Regresión Lineal y Logística** para recordar conceptos claves como:

- Modelo paramétrico.
- Funciones de activación y pérdida.
- Optimización (descenso de gradiente)
- Aprendizaje o entrenamiento.

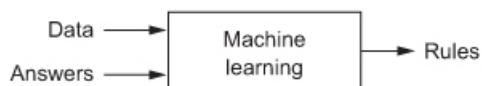
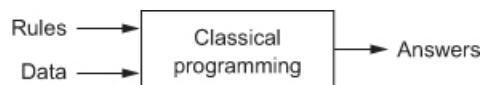
Más adelante veremos que las NN, **deep learning (DL)**, puede verse como un **conjunto apilado de modelos simples** como la Regresión Lineal o Logística.

# Aprendizaje máquina

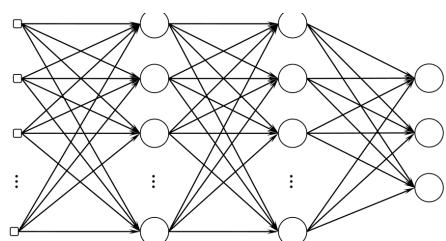
# Inteligencia artificial



1950-1980. Inteligencia artificial (sistemas expertos)

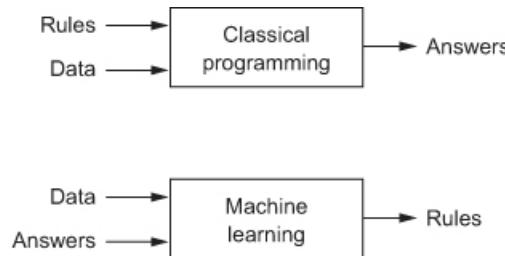


~1990. Aprendizaje máquina



2012 Aprendizaje profundo  
ImageNet - Alex Krizhevsky, Geoffrey Hinton

# "Aprendizaje" en Aprendizaje Máquina



Un **modelo** en aprendizaje máquina transforma los datos de entrada en una salida significativa.

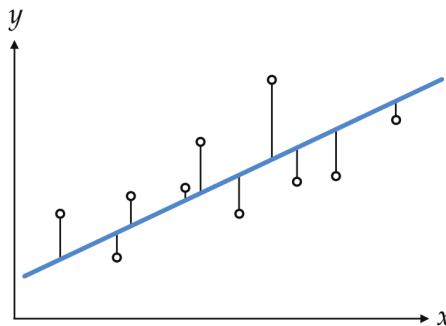
Por tanto, el problema principal en aprendizaje máquina es **aprender representaciones útiles**.

**Aprender**, en el contexto de *aprendizaje máquina*, se puede ver como un proceso automático de **búsqueda** de mejores representaciones.

La *búsqueda no es arbitraria*, se propone un **modelo paramétrico**, y el aprendizaje consiste en **ajustar los parámetros** usando los **datos disponibles**.

# Optimización numérica

# Función de perdida y función de costo



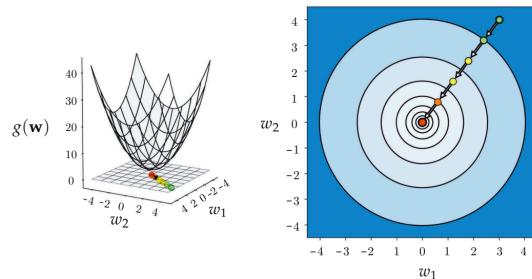
La **función de pérdida** debe ayudarnos a "medir" que tan bien nuestro modelo propuesto se **ajusta a los datos ( en forma individual )**

$$\mathcal{L}(\hat{y}_i, y_i) = \frac{1}{2}(\hat{y}_i - y_i)^2$$

La **función de costo** representa un error promedio de de nuestro modelo sobre **todos los datos**.

$$J(w, b) = \frac{1}{m} \sum_i \mathcal{L}(\hat{y}_i, y_i) = \frac{1}{m} \sum_i \frac{1}{2}(\hat{y}_i - y_i)^2$$

# Descenso de gradiente



Cuando proponemos un modelo, no conocemos sus **parámetros óptimos**, es decir, aquellos parámetros que minimizan la función de costo.

Pregunta: cómo estimar estos parámetros óptimos?

En contexto de *big data*, se recomiendan métodos (*aleatorizados*) de **primer orden**, aquellos que sólo usan información del **gradiente** (por qué?)

$$x^{t+1} \leftarrow x^t + \alpha p \quad p = -\nabla F(x)$$

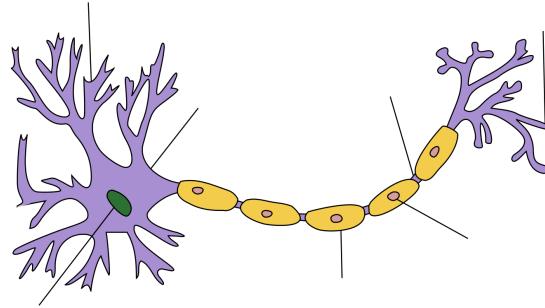
En descenso de gradiente, tenemos un algoritmo iterativo y en cada iteración,  $x$  se mueve *un poco* en la dirección de  $p$ , donde  $\alpha$  es el tamaño de paso o **learning rate**.

# Regresión Lineal y Logística como una simple neurona

# Motivación

El funcionamiento del cerebro de los animales ha interesado a los científicos:  
*cerebros tan pequeños son capaces de realizar tareas tan complicadas.*

- Las **computadoras** procesan datos a una *gran velocidad*, son *secuenciales* y *predecibles*.
- Los **cerebros de los animales**, si bien son “*lentos*”, parece que procesan la información (señales) en *paralelo* y con algo de *caos*.

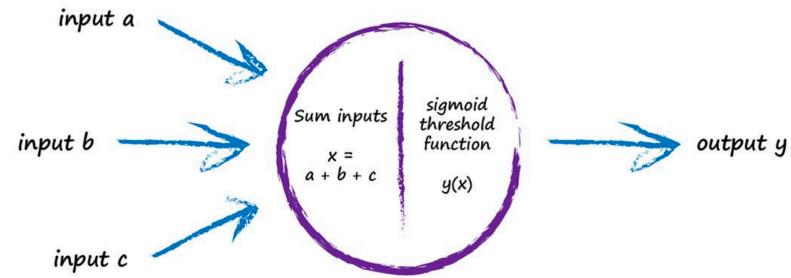
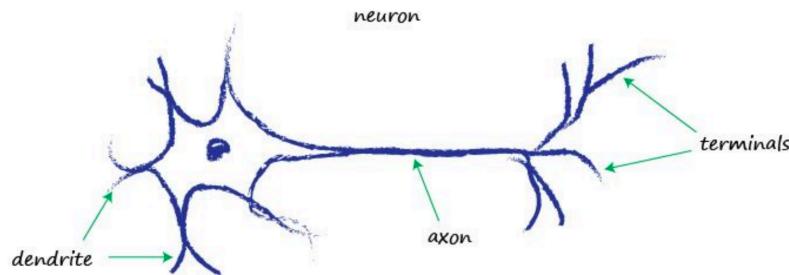


La **arquitectura** del cerebro (red de **neuronas**) parece ser la clave.

En esta sección implementaremos la Regresión Lineal y Regresión como una **simple neurona**, lo cual nos servirá como introducción a **Deep Learning**.

# Neurona simple

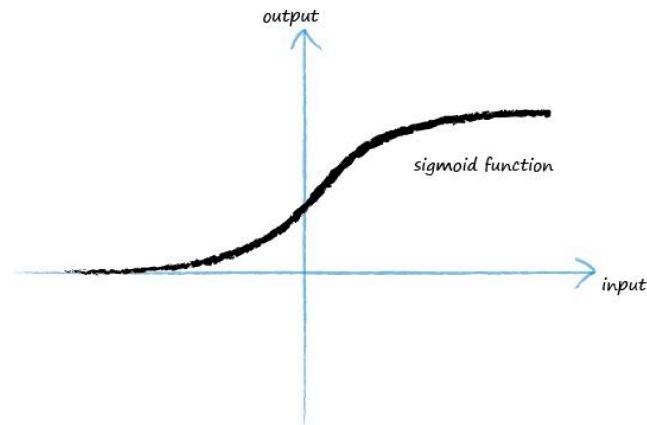
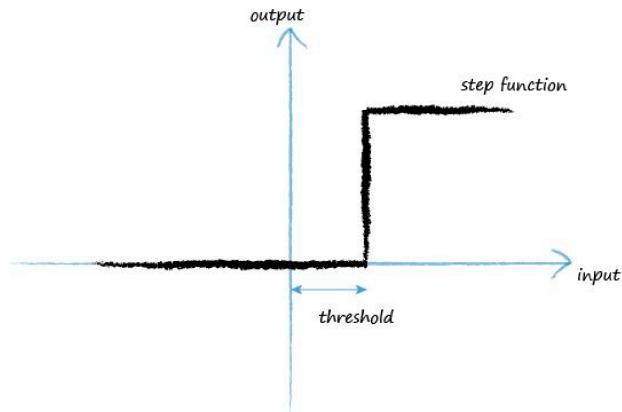
Las neuronas transmiten una señal eléctrica de un lado a otro, de las dendritas a las terminales a través de los axones. Toma una entrada y da una salida!



$$y_i = \sigma(w^T x_i + b)$$

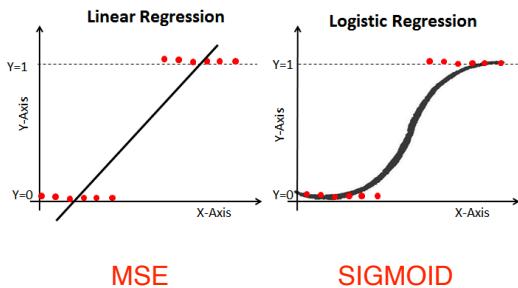
# Neurona simple

Observaciones sugieren que una neurona no **reacciona** inmediatamente a cualquier señal. La neurona **suprime** las señales de entrada hasta que dicha entrada alcanza un **umbral** máximo, una **función de activación** es necesaria.



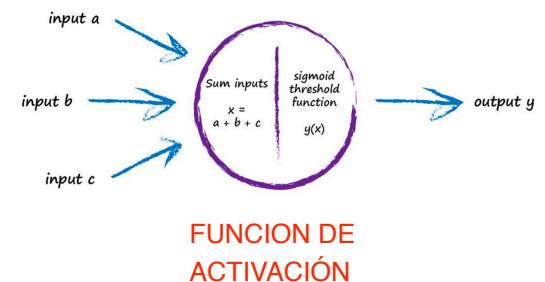
“La neurona se activa cuando un umbral es alcanzado”

# Neurona simple



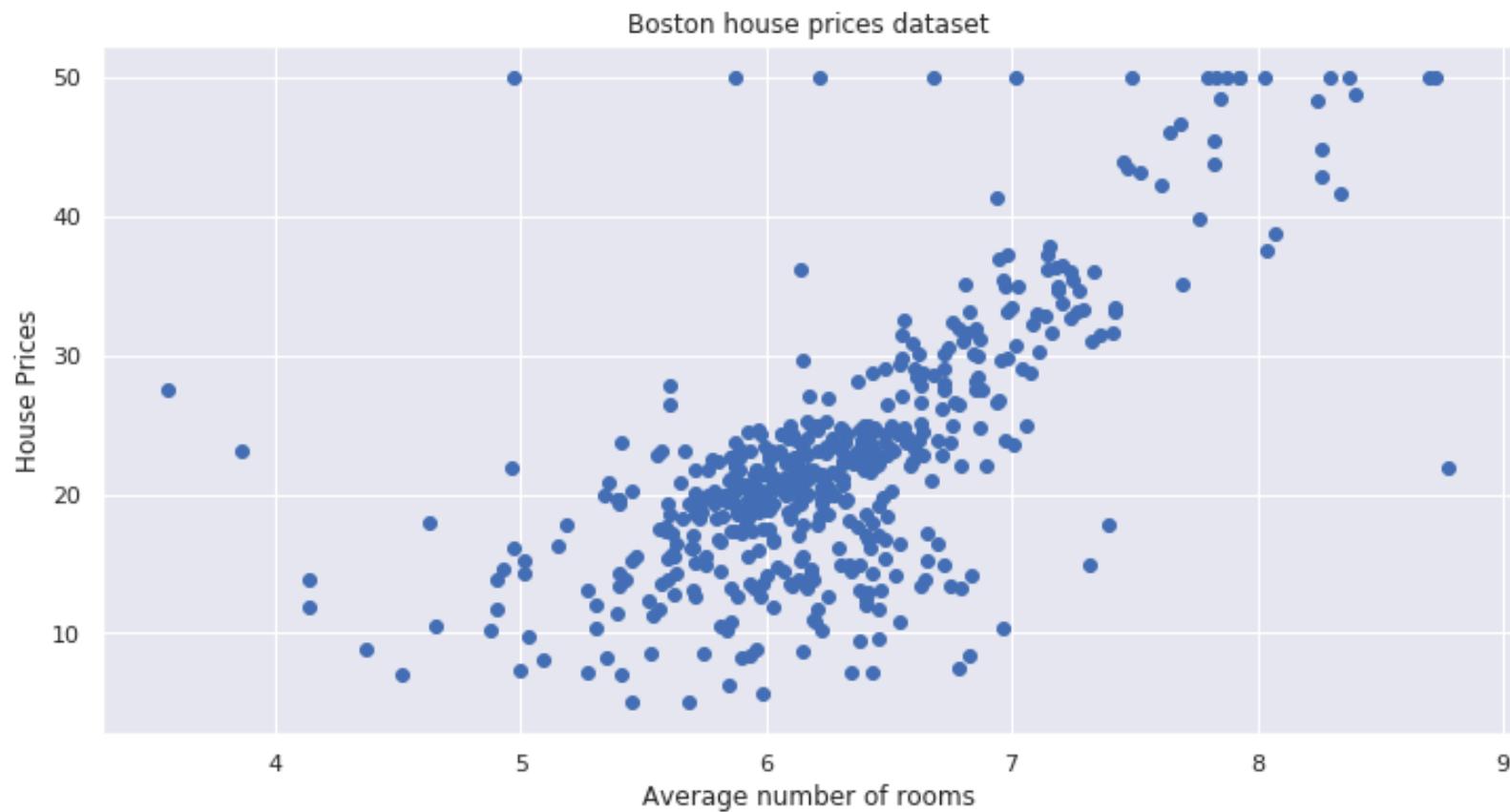
$$y_i = \sigma(w^T x_i + b)$$

Podemos ver a una **neurona simple** como una **generalización** de la regresión lineal y logística, donde la **función de activación** es la clave! En especial, funciones de activación **no-lineales**.



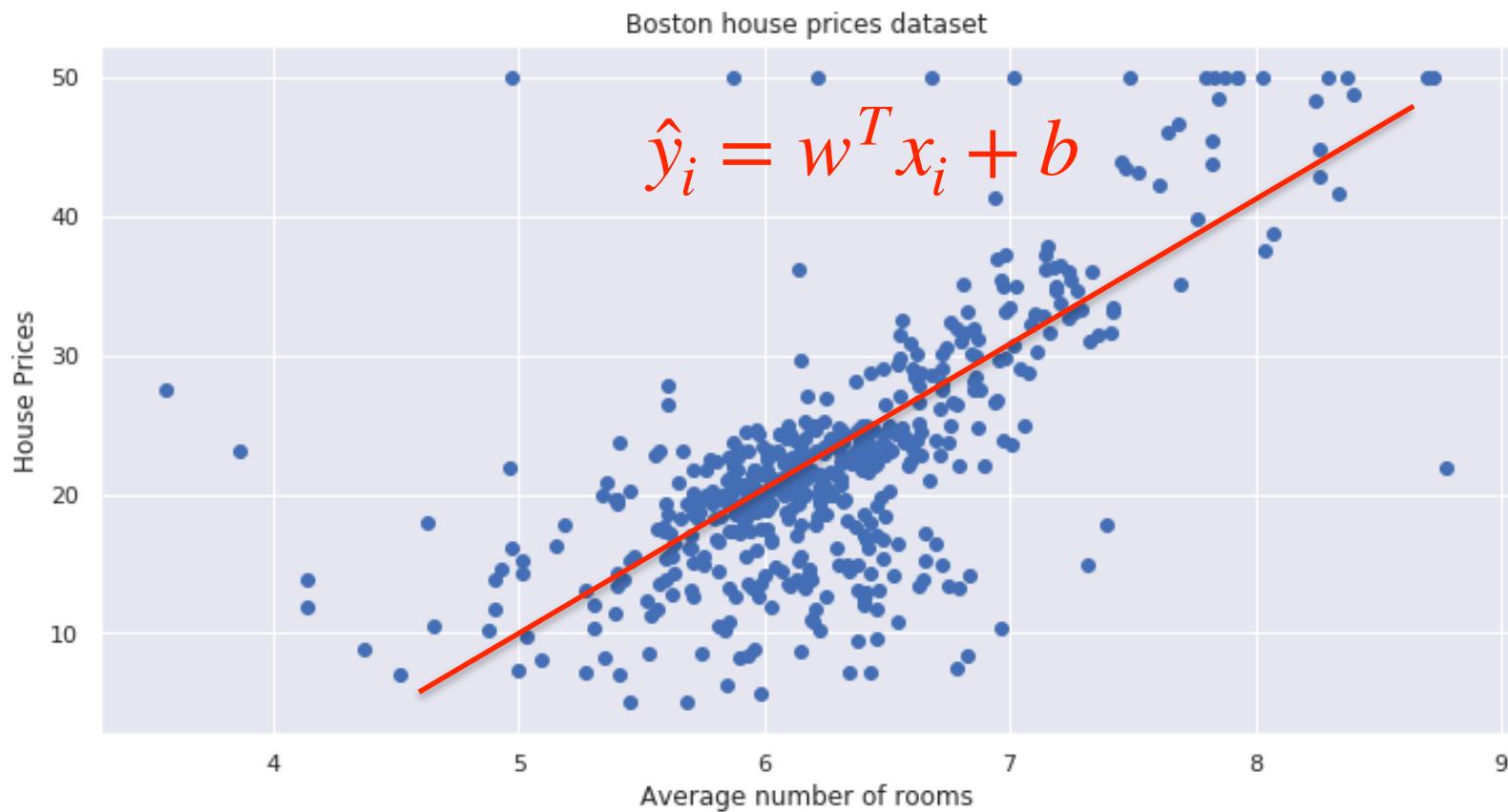
# Regresión Lineal

# Regresión Lineal: Boston house prices data set



Cómo podemos modelar estos datos?

# Regresión Lineal: Boston house prices data set

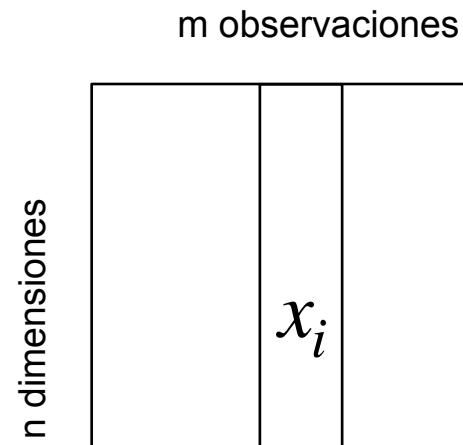


Cuál es el modelo?

Cuáles son los parámetros del modelo?

Cómo estimamos los parámetros (óptimos) del modelo propuesto?

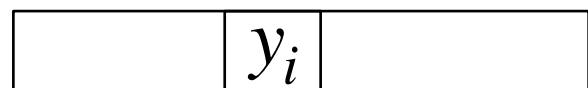
# Regresión Lineal: natación



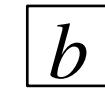
$$X_{n \times m}$$



$$w_{n \times 1}$$

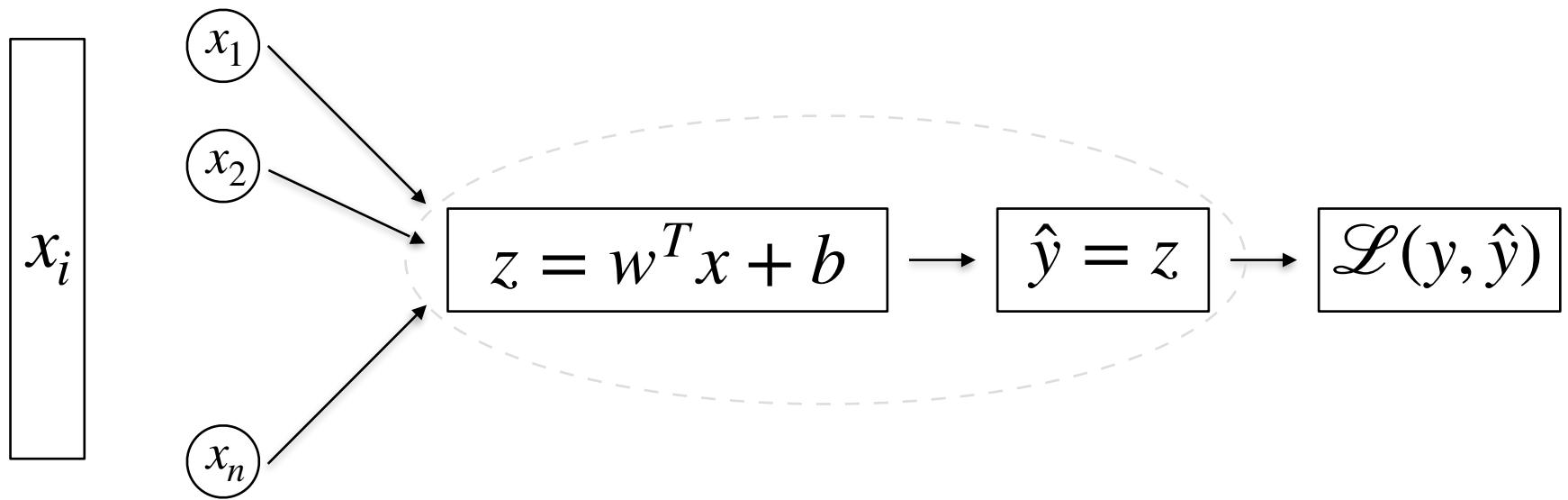


$$y_{1 \times m}$$



$$b_{1 \times 1}$$

# Regresión Lineal: neurona



$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}_i, y_i) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (\hat{y}_i - y_i)^2$$

# Regresión Lineal: Boston house prices data set

Dada la función de costo:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}_i, y_i) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (\hat{y}_i - y_i)^2$$

Formulamos el problema de optimización

$$\min_{w,b} J(w, b)$$

Ahora, cómo resolvemos este problema de optimización? (**GD**).

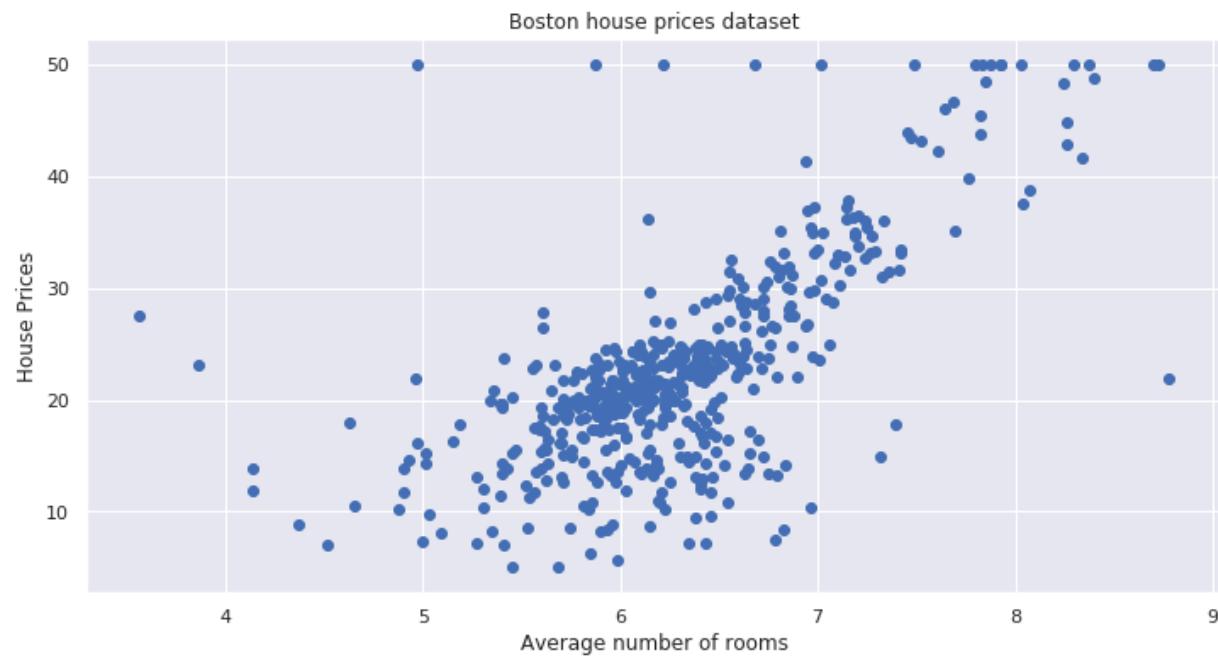
$$w^{t+1} = w^t + \alpha \nabla_w J \quad b^{t+1} = b^t + \alpha \nabla_b J$$

Usando GD, nuestro modelo **aprenderá** los parámetros óptimos.

Vamos a entrenar nuestro modelo.

# Actividad 1

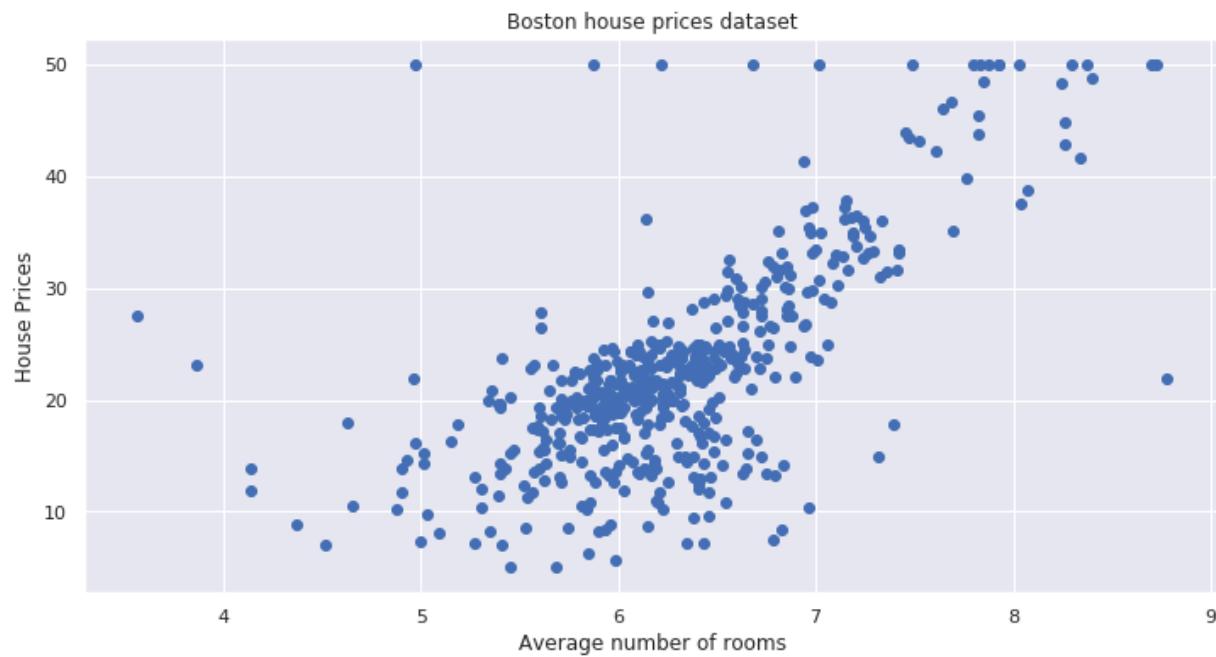
Terminar el notebook [1.1\\_RegresionLineal.ipynb](#)



Retroalimentación.

## Actividad 2

Terminar el notebook **1.2\_Neurona\_RegresionLineal.ipynb**



Retroalimentación.

# Regresión Logística

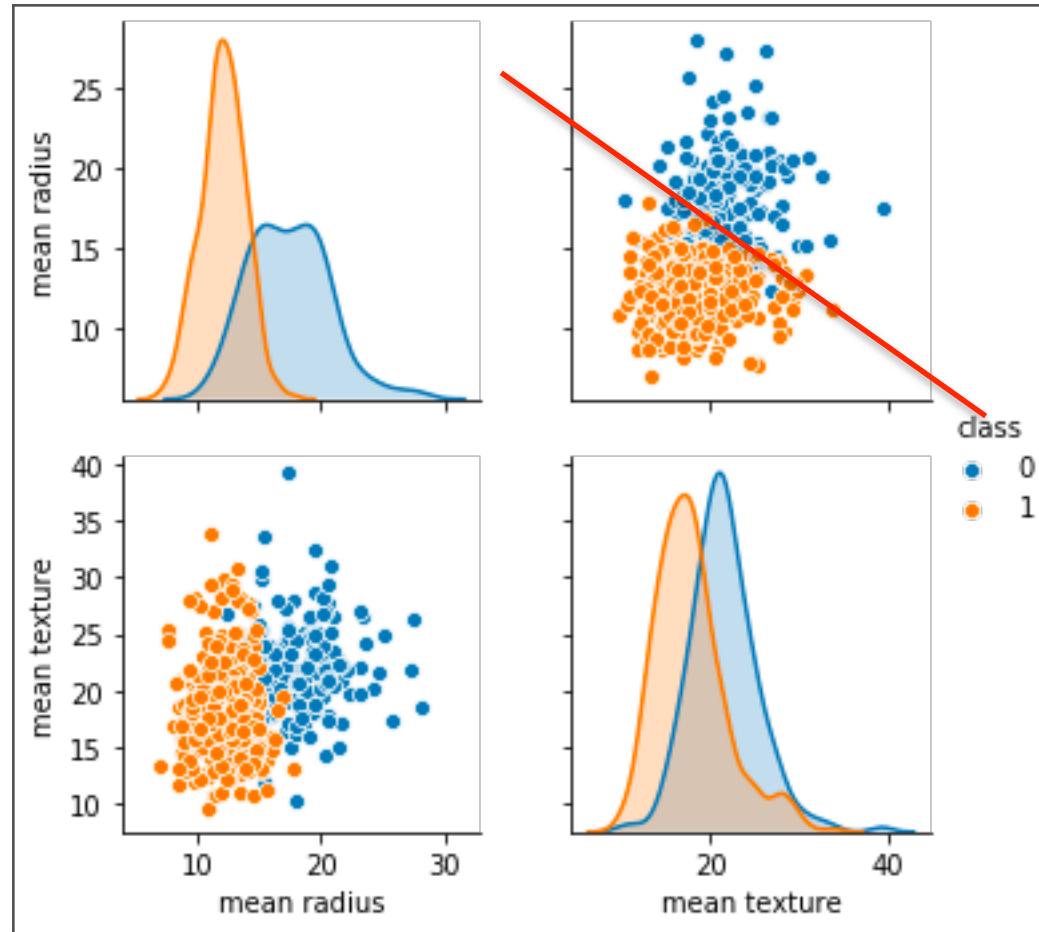
# Regresión Logística: Breast cancer wisconsin dataset



23

# Regresión Logística: Breast cancer wisconsin dataset

$$\hat{y}_i = \sigma(w^T x_i + b)$$



# Regresión Logística: Breast cancer wisconsin dataset

$$\hat{y}_i = \sigma(w^T x_i + b)$$

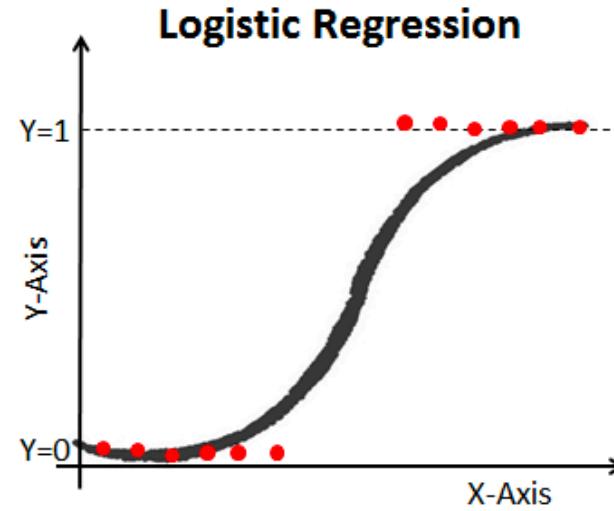
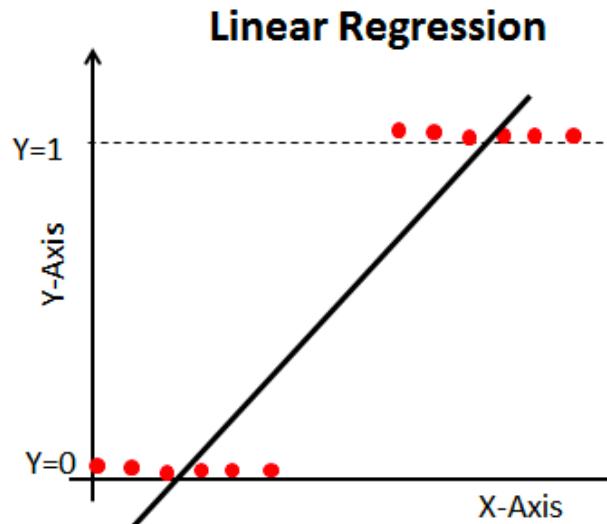
$$\hat{y}_i = \sigma(w^T x_i + b)$$

$$\sigma(w^T x + b) = w^T x + b$$

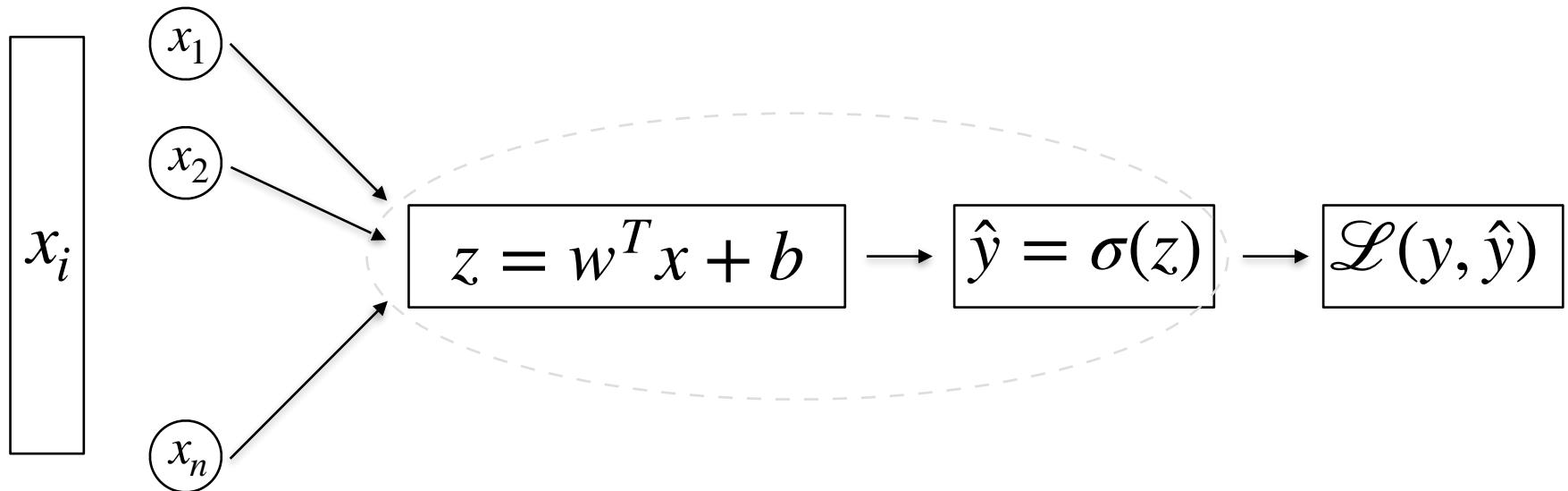
$$\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

$$\sigma(z) = z$$

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



# Regresión Logística: neurona



$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}_i, y_i) = -\frac{1}{m} \sum_{i=1}^m y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

# Regresión Logística: Breast cancer wisconsin dataset

De igual forma, definimos una función de costo  $J(w, b)$ , con  $\sigma(x)$  como una función de activación (**activation function**)

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}_i, y_i) = -\frac{1}{m} \sum_{i=1}^m y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Formulamos el problema de optimización

$$\min_{w,b} J(w, b)$$

Ahora, cómo resolvemos este problema de optimización? (**GD**).

$$w^{t+1} = w^t + \alpha \nabla_w J \quad b^{t+1} = b^t + \alpha \nabla_b J$$

# Actividad 3

Terminar el notebook [1.3\\_RegresionLogistica.ipynb](#)



Retroalimentación.

# Actividad 4

Terminar el notebook **1.4\_Neurona\_RegresionLogistica.ipynb**



Retroalimentación.

# Métodos de optimización

# Descenso de gradiente (versiones)

$$g^t = \nabla_x J(x^t)$$

(batch) Gradient Descent

$$x^{t+1} = x^t - \alpha g^t$$

Stochastic Gradient Descent

$$x^{t+1} = x^t - \alpha g_{(x_i, y_i)}^t$$

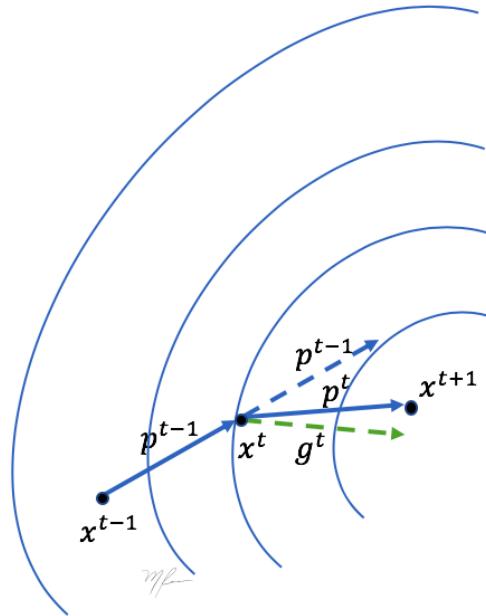
mini-batch Gradient Descent

$$x^{t+1} = x^t - \alpha g_{(x_{(i,i+n)}, y_{(i,i+n)})}^t$$

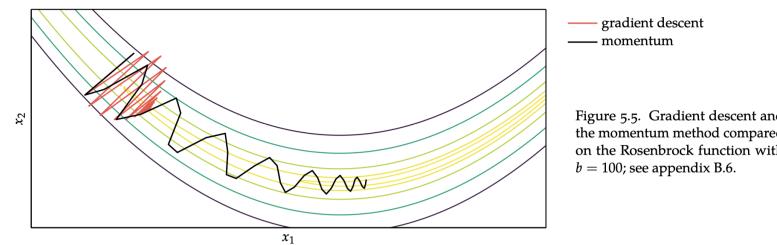
# Gradient Descent with momentum

$$p^t = g^t + \eta p^{t-1}$$

$$x^{t+1} = x^t - \alpha p^t$$



[Imagen tomada de ] [Apuntes-Dr.MarianoRivera](#)

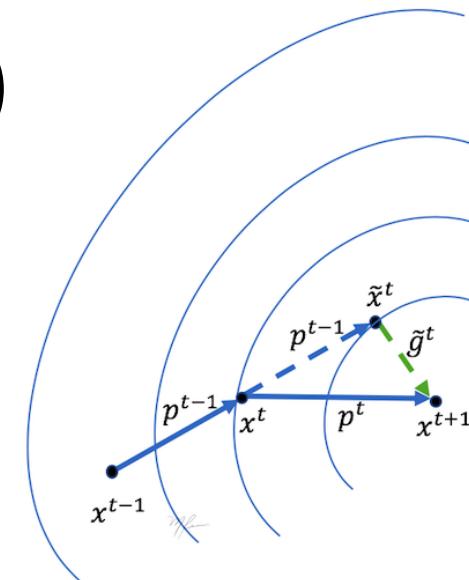


# Nesterov Accelerated Gradient (NAG)

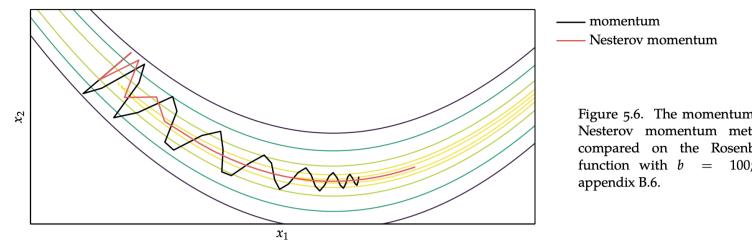
$$\tilde{x}^t = x^t - \alpha p^{t-1} \quad g^t = \nabla_x J(\tilde{x}^t)$$

$$p^t = g^t + \eta p^{t-1}$$

$$x^{t+1} = x^t - \alpha p^t$$



[Imagen tomada de ] [Apuntes-Dr.MarianoRivera](#)



# Adaptive Subgradient Method (Adagrad)

$$g_i^t = [\nabla_x J(x^t)]_i$$

$$s_i^t = \sum_{k=1}^t (g_i^k)^2$$

$$x_i^{t+1} = x_i^t - \frac{\alpha}{\epsilon + \sqrt{s_i^t}} g_i^t$$

# Root Mean Square Propagation (RMSProp)

$$g_i^t = [\nabla_x J(x^t)]_i$$

$$\hat{s}_i^t = \eta \hat{s}^{t-1} + (1 - \eta)(g^t \circ g^t) \quad \eta = 0.9$$

$$x_i^{t+1} = x_i^t - \frac{\alpha}{\epsilon + \sqrt{\hat{s}_i^t}} g_i^t = x_i^t - \frac{\alpha}{\epsilon + RMS(g_i)} g_i^t$$

# Adaptive Moment Estimation (ADAM)

$$v^t = \eta_v v^{t-1} + (1 - \eta_v) g^t \quad \eta_v = 0.9$$

$$s^t = \eta_s s^{t-1} + (1 - \eta_s)(g^t \circ g^t) \quad \eta_s = 0.999$$

$$\hat{v}^t = \frac{1}{1 - (\eta_v)^t} v^t$$

$$\hat{s}^t = \frac{1}{1 - (\eta_s)^t} s^t$$

$$x^{t+1} = x^t - \frac{\alpha}{\epsilon + \sqrt{\hat{s}^t}} \hat{v}^t$$

# Links de interés



Keras <https://keras.io/api/optimizers/>

Descenso de Gradiente y Variaciones Sobre el Tema - Mariano Rivera 2018

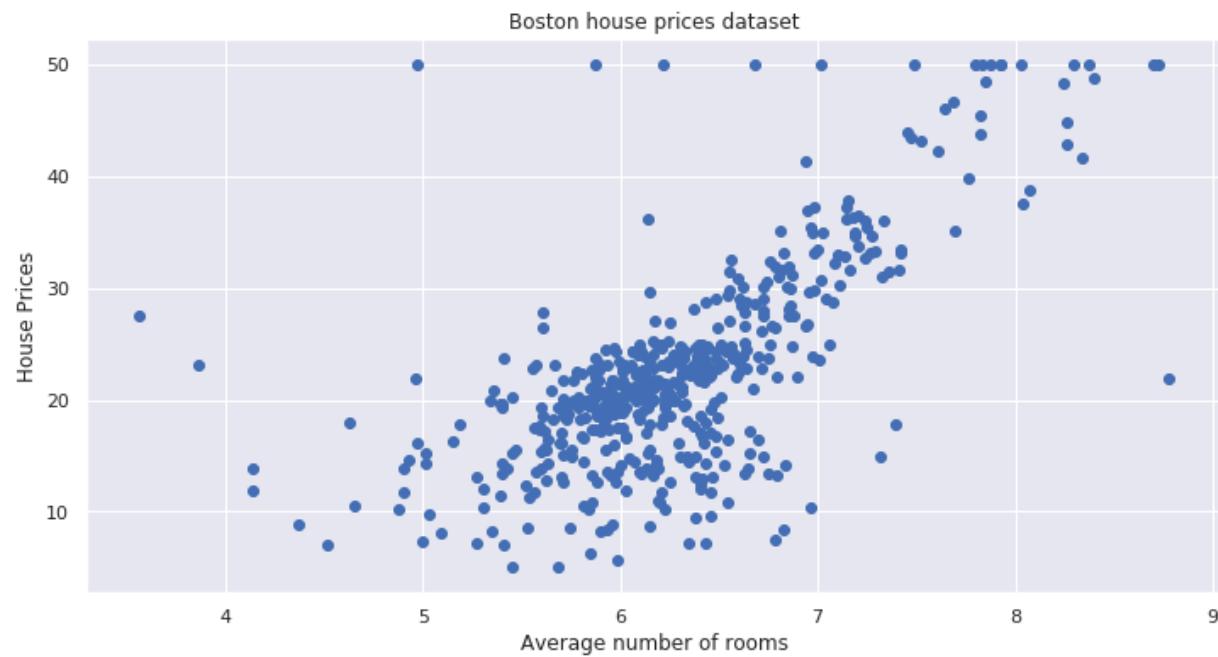
[Apuntes-Dr.MarianoRivera](#)

An overview of gradient descent optimization algorithms - Sebastian Ruder 2016

<https://ruder.io/optimizing-gradient-descent/index.html>

# Actividad 5

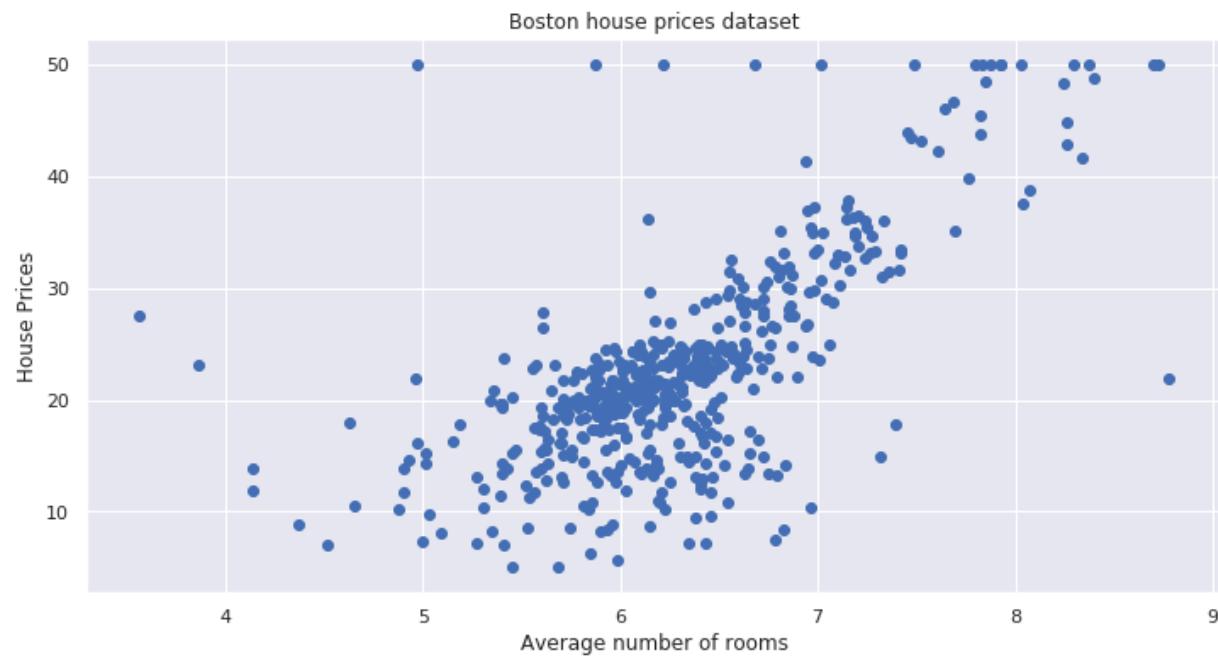
Terminar el notebook [1.5\\_RegresionLineal\\_NAG-ADAM.ipynb](#)



Retroalimentación.

# Actividad 6

Terminar el notebook [1.6\\_Neurona\\_RegresionLineal\\_NAG-ADAM.ipynb](#)

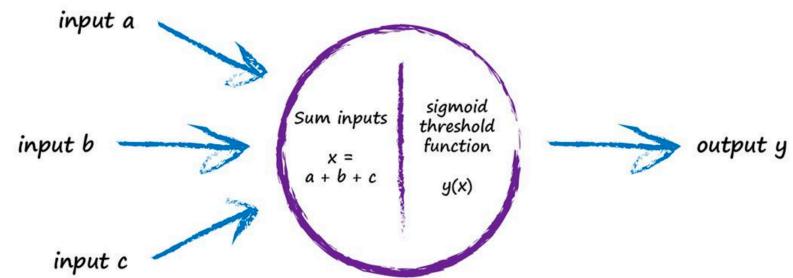
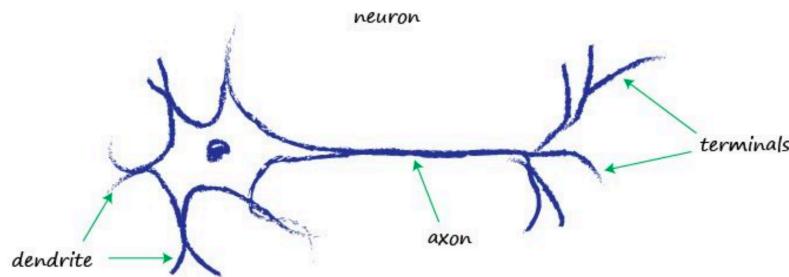


Retroalimentación.

# (shallow) Neural Network

# Motivación

Ya vimos como *modelar el funcionamiento* de una sola neurona.

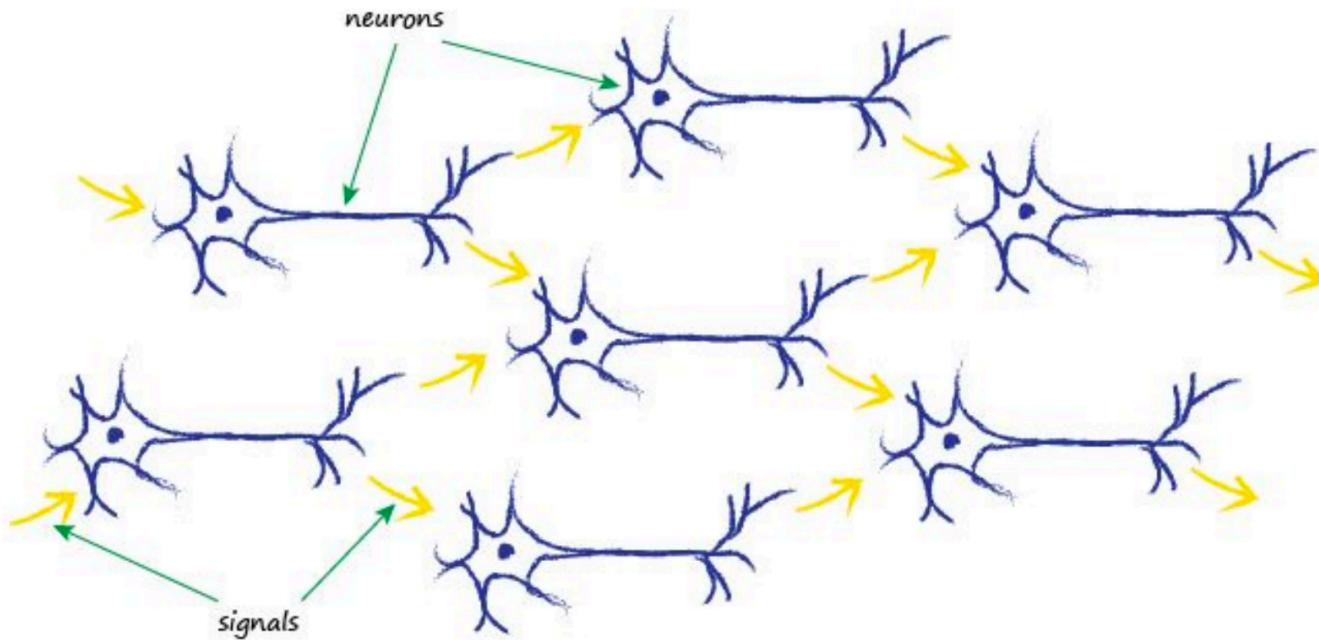


**Podemos realmente construir o modelar un cerebro?**

El objetivo en esta sección es **extender el modelo de una sola neurona\*** y ver su implementación en **TensorFlow**.

# Neural Network

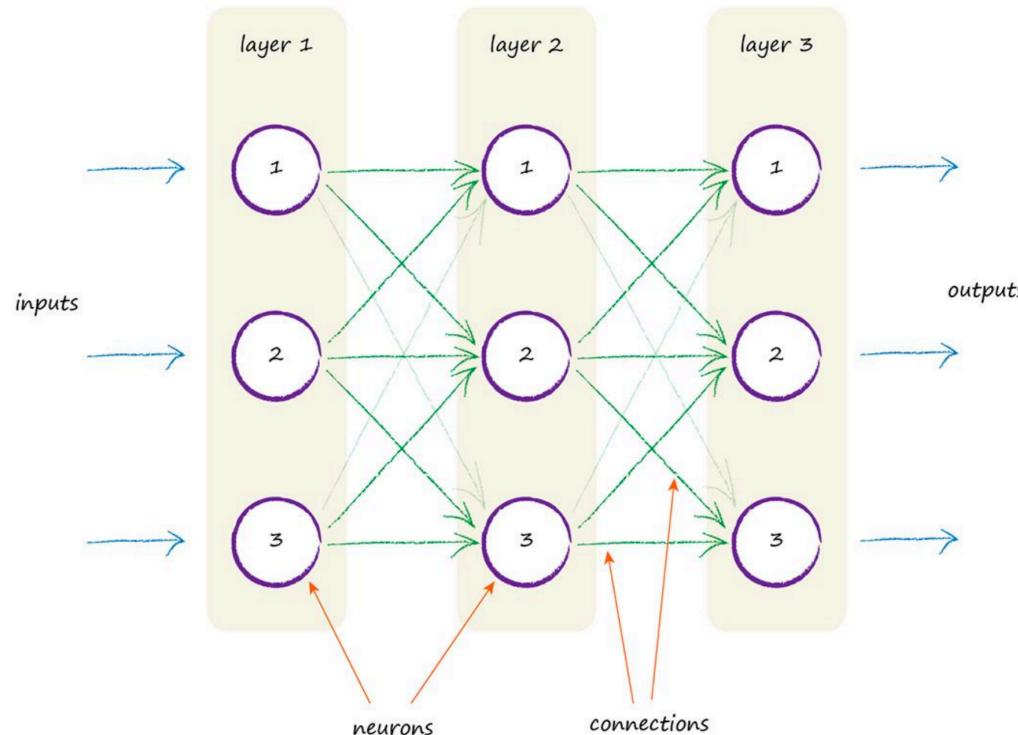
En realidad, una neurona toma las entradas de muchas neuronas, así como envía una señal a muchas otras.



# Neural Network

Por tanto, dos formas sencillas de extender el modelo de una sola neurona:

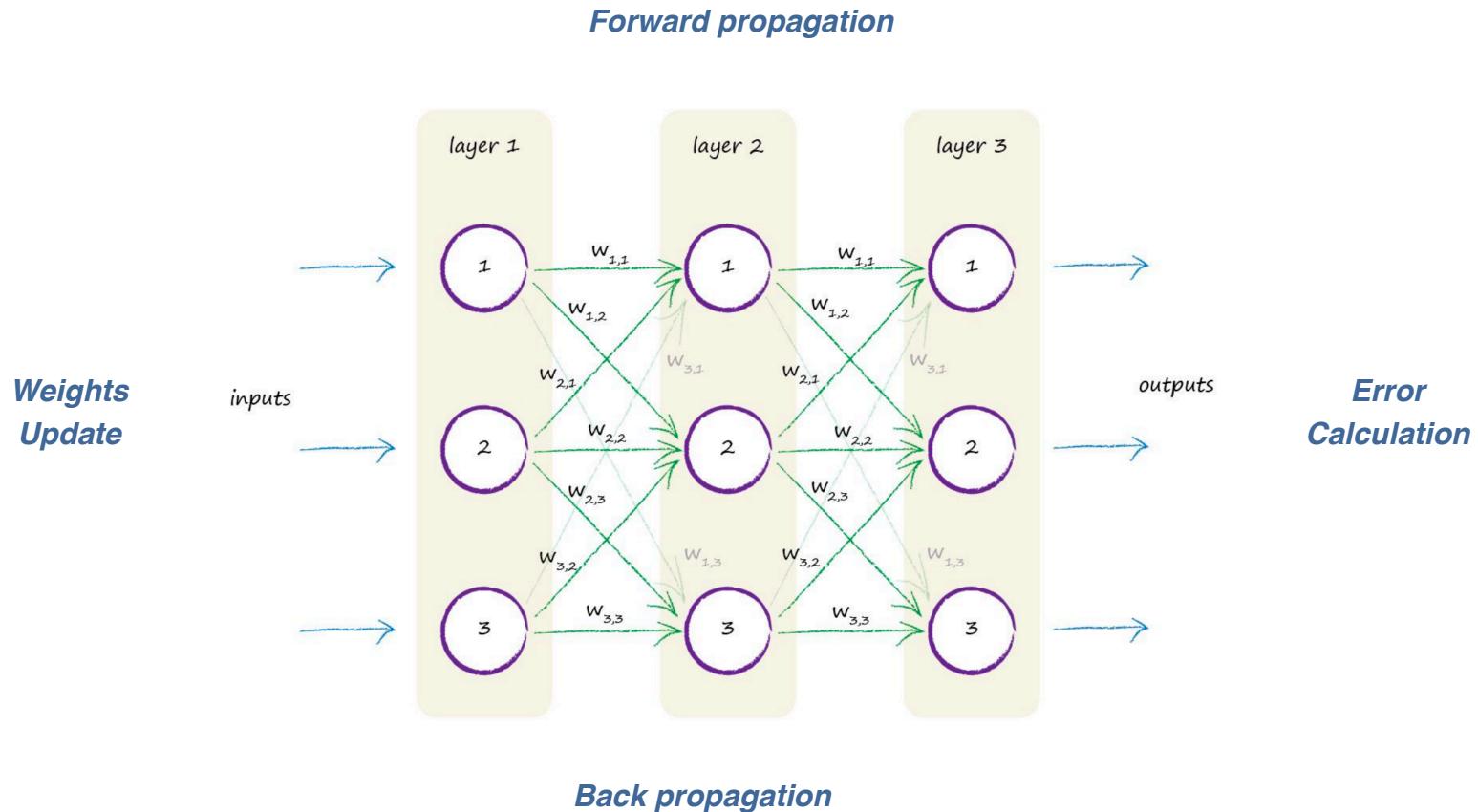
1. La misma entrada puede alimentar a múltiples neuronas, más neuronas por capa o *layer*.
2. Una neurona en una capa puede servir de entrada o otras capas, múltiples capas.



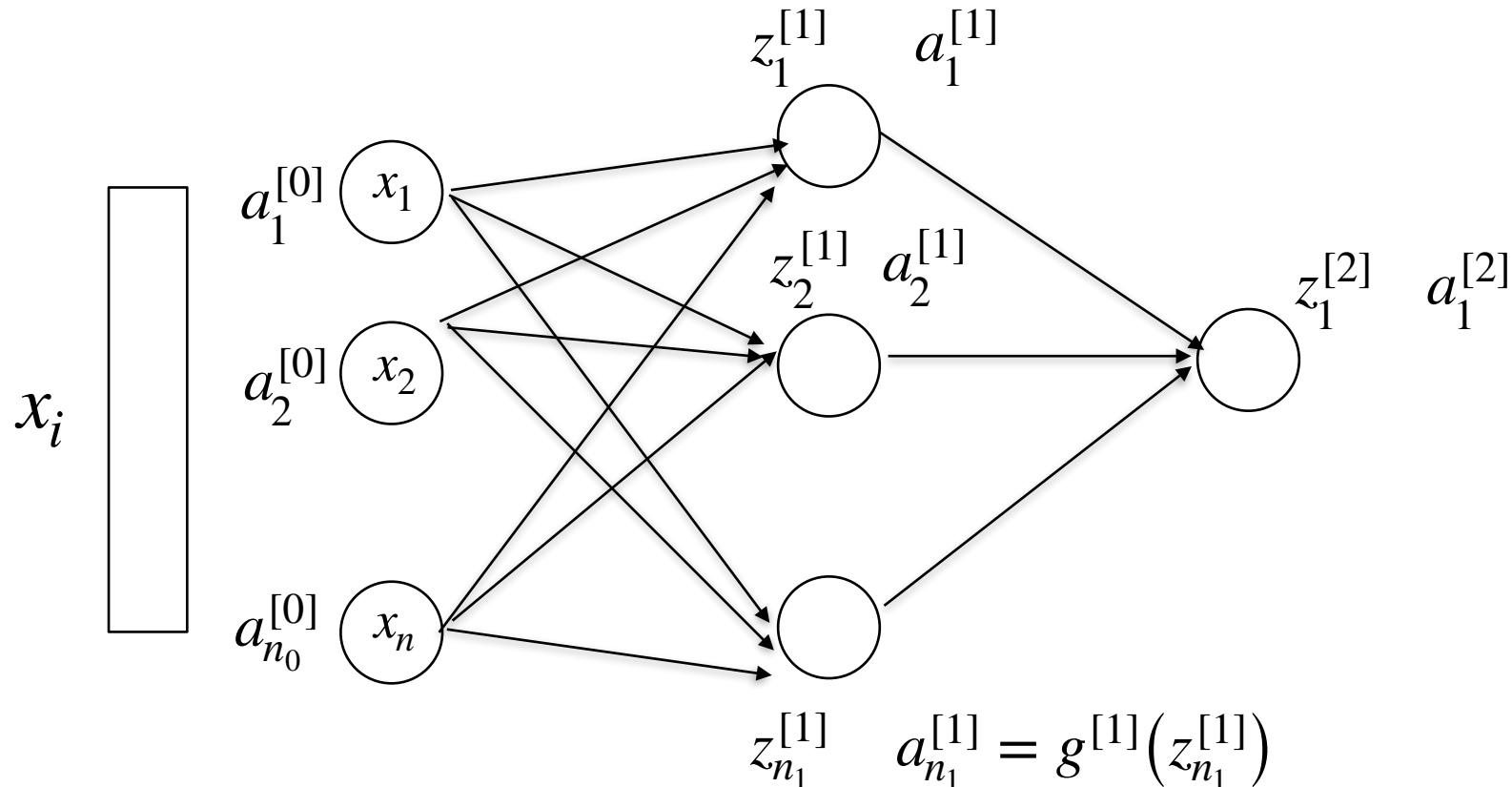
# Neural Network

Pero, cómo se entrena una red de neuronas?

Una red neuronal aprende a mejorar la salida al actualizar los pesos de las conexiones internas.



# Neural Network: notación



$l$

número de capas

$n_l$

número de neuronas en la capa  $l$

$g^{[l]}$

función de activación en la capa  $l$

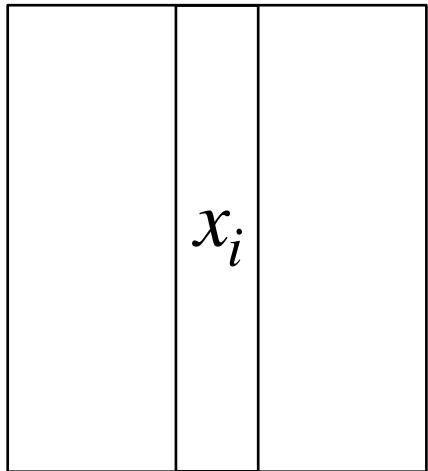
$z_i^{[l]} \quad a_i^{[l]}$

combinación / activación de la neurona  $i$  en la capa  $l$

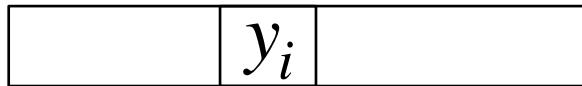
# Neural Network: notación

n dimensiones

m observaciones



$$X = A^{[0]}$$



$$y_1 \times m$$

# entrada  $l - 1$

$w_1^{[l]}$
$w_2^{[l]}$
$w_{n_l}^{[l]}$

$$W^{[l]}$$

# neuronas en capa  $l$

$b_1^{[l]}$
$b_2^{[l]}$
$b_{n_l}^{[l]}$

$$b^{[l]}$$

# Neural Network: notación

una sola observación

The diagram illustrates the computation of a single observation  $z^l$  from inputs  $a^l$  and weights  $b^l$ . It shows three vertical stacks of rectangles. The left stack, labeled  $W^{[l]}$ , contains three horizontal rectangles labeled  $w_1^{[l]}$ ,  $w_2^{[l]}$ , and  $w_{n_l}^{[l]}$ . The middle stack, labeled  $a^{[l]}$ , contains two horizontal rectangles, with the bottom one labeled  $a_i^l$ . The right stack, labeled  $b^{[l]}$ , contains three horizontal rectangles labeled  $b_1^{[l]}$ ,  $b_2^{[l]}$ , and  $b_{n_l}^{[l]}$ . A plus sign (+) is placed between the  $a^{[l]}$  and  $b^{[l]}$  stacks, and an equals sign (=) is placed to the right of the  $b^{[l]}$  stack. Above the first plus sign, the text "una sola observación" is centered. Above the second plus sign, the text "una sola observación" is centered.

$$W^{[l]} \quad a^{[l]} \quad + \quad b^{[l]} = z^{[l]}$$

$$a^{[l+1]} = g^{[l]}(z^{[l]})$$

# Neural Network: notación

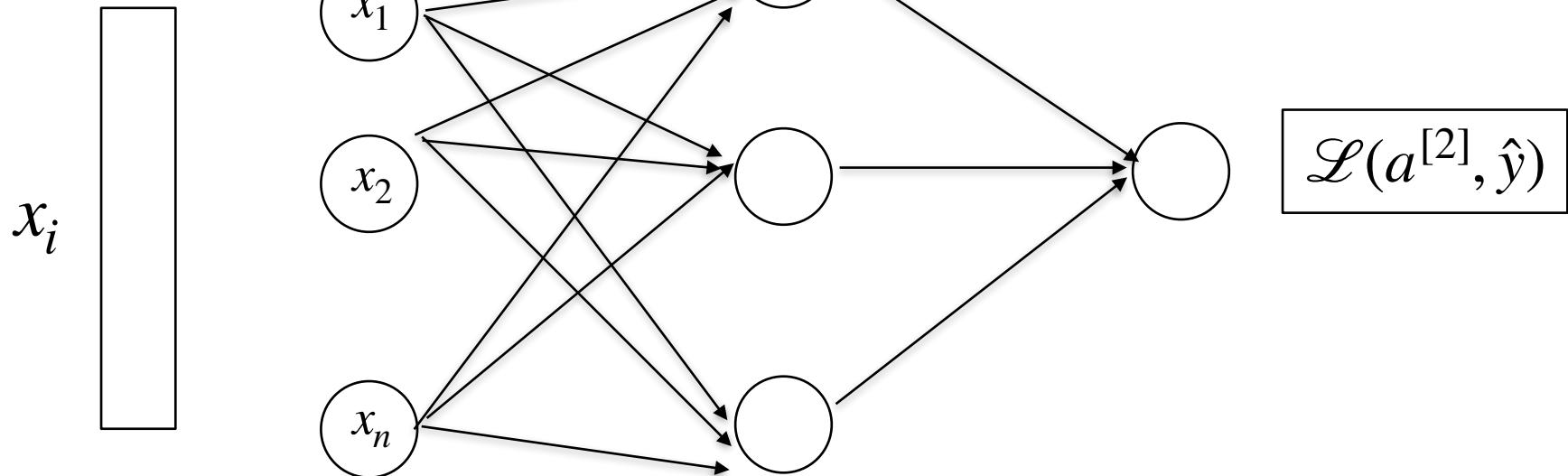
$$\begin{array}{c} \text{m observaciones} \\ w_1^{[l]} \\ w_2^{[l]} \\ \vdots \\ w_{n_l}^{[l]} \end{array} + \begin{array}{c} \text{m observaciones} \\ a_i^l \\ \vdots \\ \end{array} = \begin{array}{c} \text{m observaciones} \\ b_1^{[l]} \\ b_2^{[l]} \\ \vdots \\ b_{n_l}^{[l]} \end{array} = \begin{array}{c} \# \text{neuronas en capa } l \\ z_i^l \\ \vdots \\ \end{array}$$

$W^{[l]}$        $A^{[l]}$       +       $b^{[l]}$       =       $Z^{[l]}$

$$A^{[l+1]} = g^{[l]}(Z^{[l]})$$

# Neural Network: ejercicio (clasificación binaria)

0	1	1	1	1
0	1	1	0	0
1	0	0	1	0
1	0	0	1	1
0	1	1	0	0



$$n_0 = (28 \times 28) = 784$$

$$n_1 = 128$$

$$n_2 = 1$$

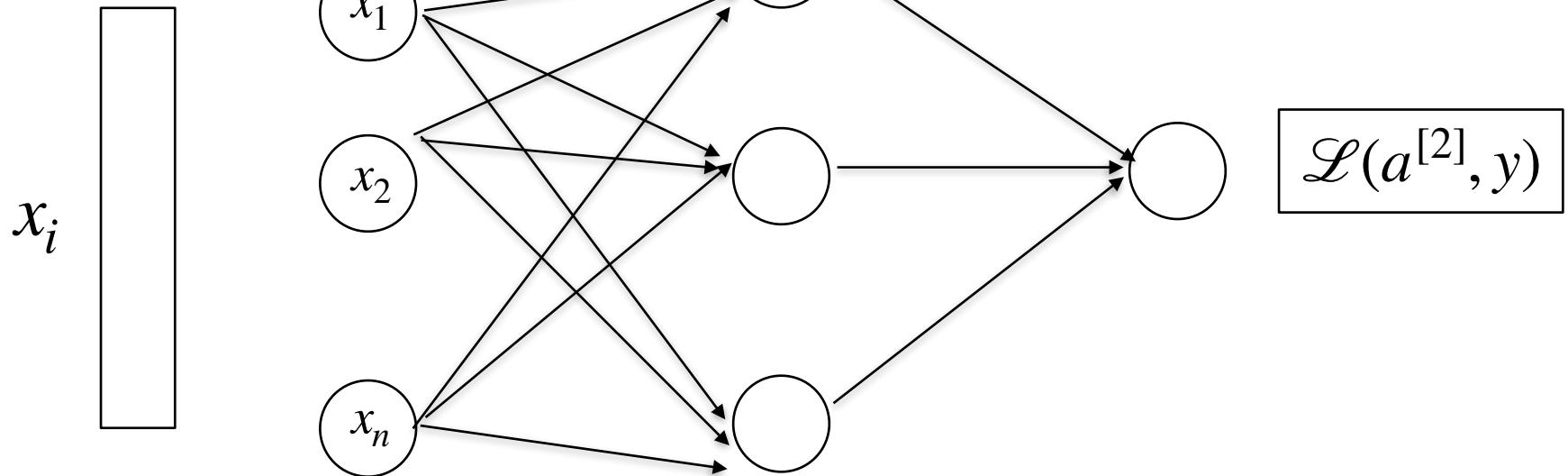
$$g^{[1]} = \tanh(z)$$

$$g^{[2]} = \sigma(z)$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(a_{(i)}^{[2]}, y_i) = -\frac{1}{m} \sum_{i=1}^m y_i \log(a_{(i)}^{[2]}) + (1 - y_i) \log(1 - a_{(i)}^{[2]})$$

# Neural Network: ejercicio (clasificación binaria)

0	1	1	1	1
0	1	1	0	0
1	0	0	1	0
1	0	0	1	1
0	1	1	0	0

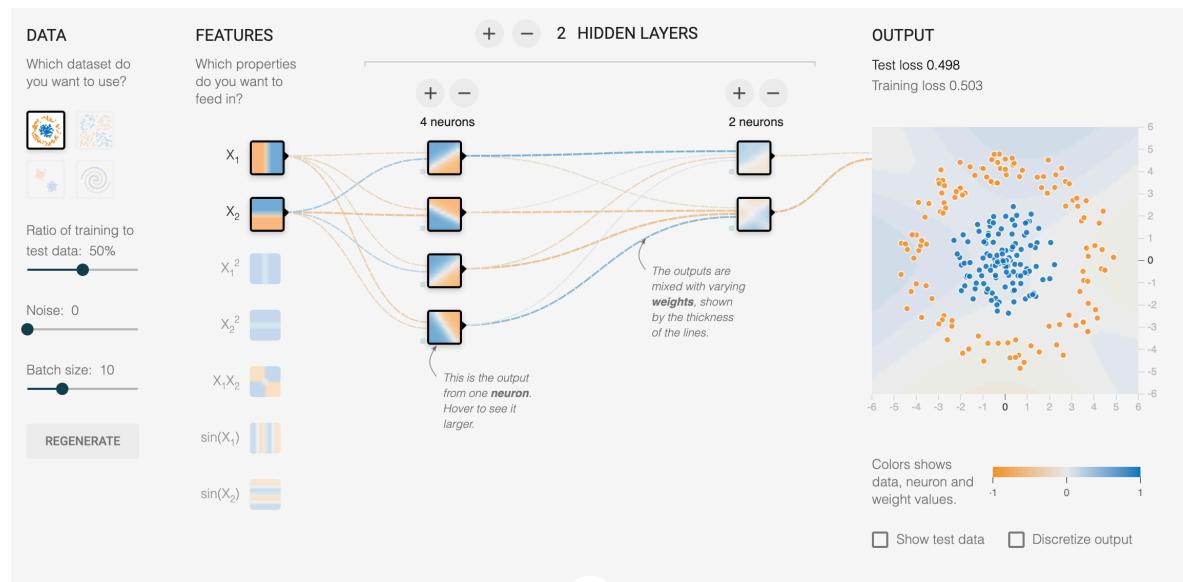


$$A^{[2]} = g^{[2]}(W^{[2]}g^{[1]}(W^{[1]}A^{[0]} + b^{[0]}) + b^{[2]})$$

Riqueza o Complejidad: composición y "no linealidad"

# Actividad 7

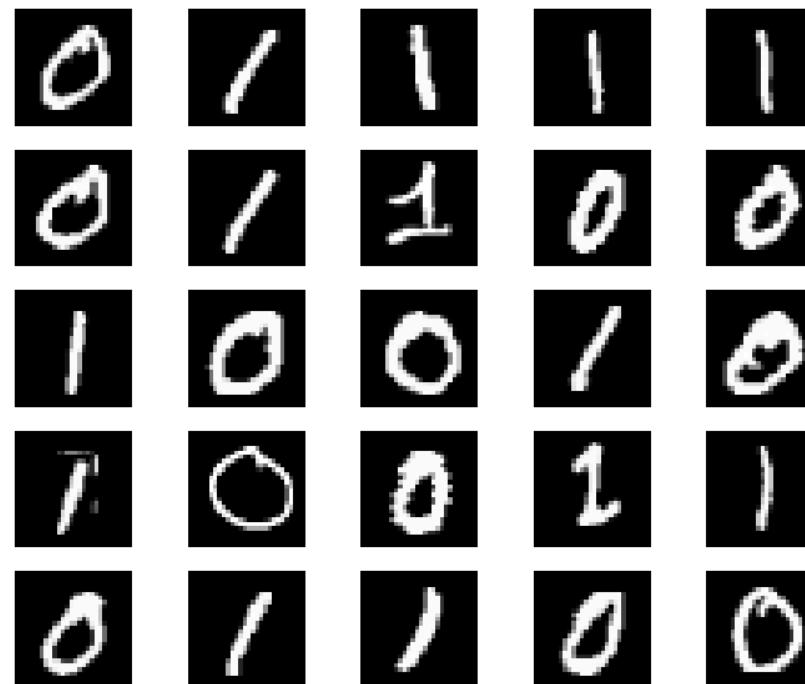
“Jugar” con los problemas en <https://playground.tensorflow.org/>



Retroalimentación.

## Actividad 8

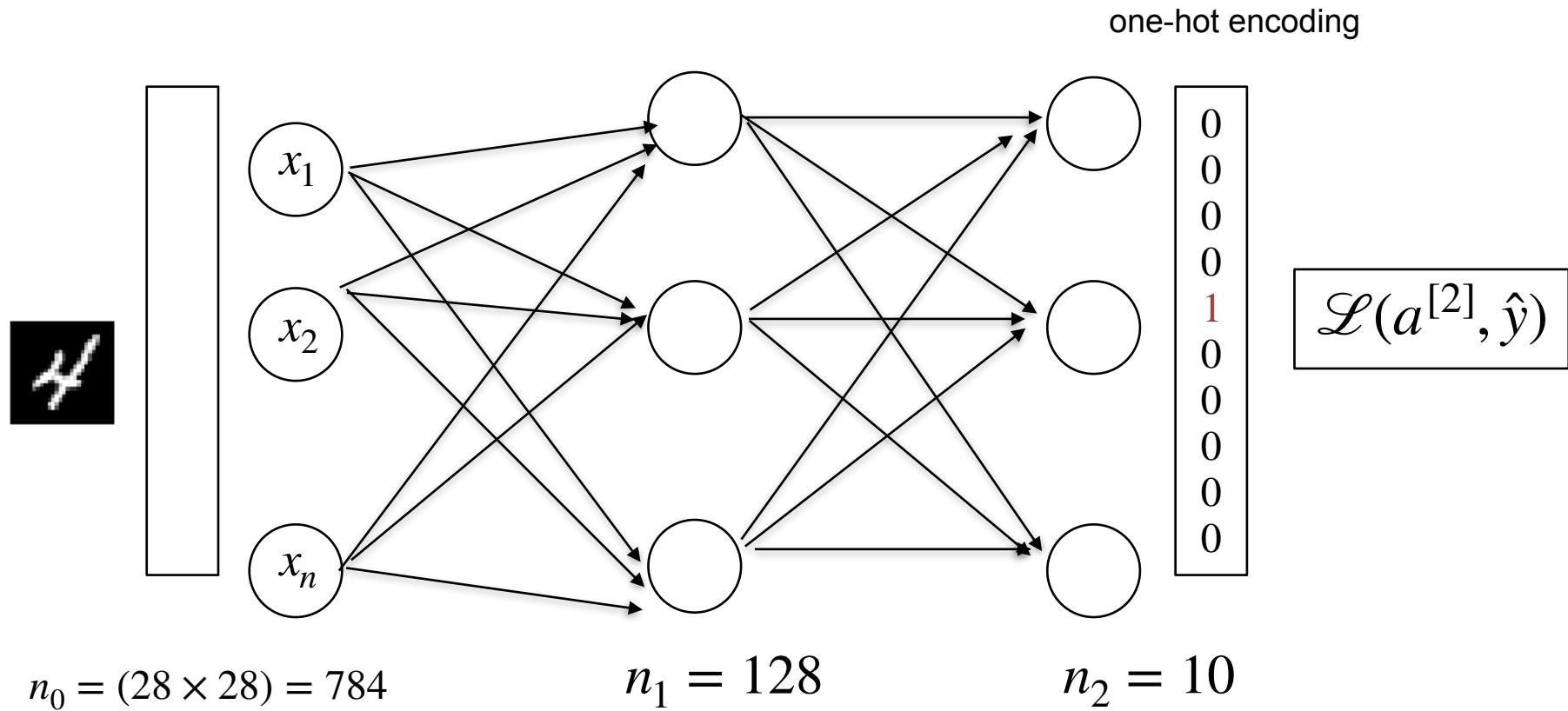
Terminar el notebook [1.7\\_shallowNN.ipynb](#)



Retroalimentación.

# Proyecto

# Proyecto: shallow NN for MNIST



$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(a_{(i)}^{[2]}, y_i) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} \|a_{(i)}^{[2]} - y_{(i)}\|_2^2 \quad (\text{MSE})$$

# Proyecto: shallow NN for MNIST

## Prepare the data

```
[9] x_train, x_test = x_train/255.0, x_test / 255.0

[10] y_train_ohe = tf.one_hot(y_train, 10)
     y_test_ohe = tf.one_hot(y_test, 10)

[11] print('Train data: ', x_train.shape, y_train_ohe.shape)
     print('Test data: ', x_test.shape, y_test_ohe.shape)
```

Train data: (60000, 28, 28) (60000, 10)  
Test data: (10000, 28, 28) (10000, 10)

## Build the model

```
▶ model = tf.keras.models.Sequential()

model.add( tf.keras.layers.Flatten( input_shape=x_train[0].shape ) )
model.add( tf.keras.layers.Dense( 128, activation='tanh' ) )
model.add( tf.keras.layers.Dense(10, activation='sigmoid'))

model.compile(optimizer='adam', loss='mse')
```

## Train the model

```
[ ] result = model.fit(x_train, y_train_ohe, validation_data=(x_test, y_test_ohe), epochs=15, verbose=1)

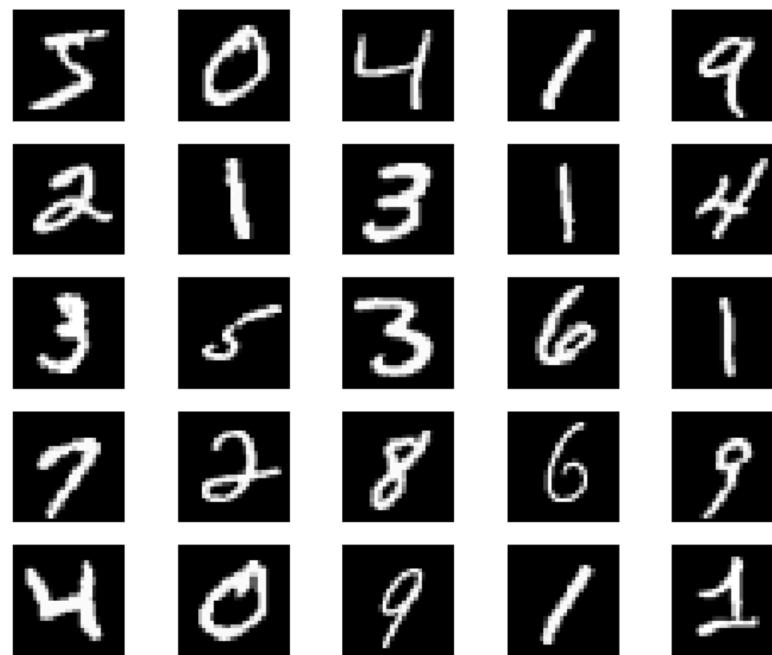
[17] np.sum(y_test_hat==y_test) / y_test.shape[0]
```

0.9758

# Limitaciones NN

## Actividad 9

"Jugar" con el notebook [1.11\\_FNN\\_MNIST\\_SOLUCION.ipynb](#)



Retroalimentación.

# Actividad 10

Terminar el notebook [2.0\\_FNN\\_FashionMNIST.ipynb](#)



Este ejercicio nos servirá para motivar la necesidad de las **redes convolucionales**.

## **Odin Eufracio**

Centro de Investigación en Matemáticas - CIMAT  
Jalisco SN, Mineral de Valenciana Gto. Gto.

Office: D307

Phone: (+52) 473 732 7155 ext. 4730

E-Mail: [odin.eufracio@cimat.mx](mailto:odin.eufracio@cimat.mx)