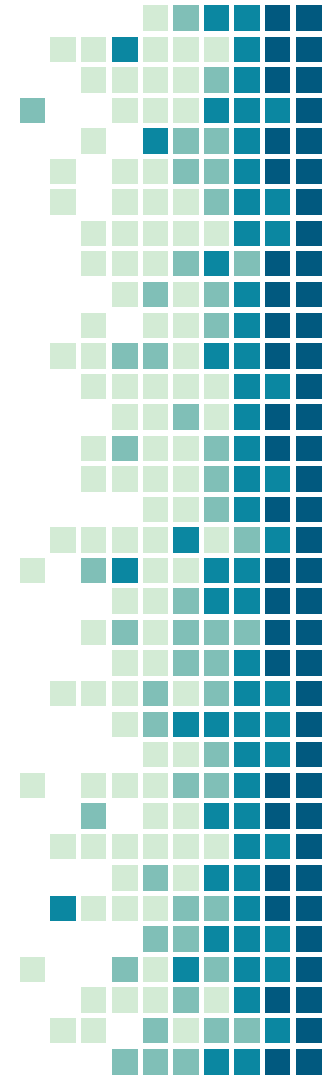


# Python e IA: Deep Learning & Keras

Ing. Luis Martínez

# Aplicaciones

- Faking videos
  - Video “fake” de Barack Obama
  - Convertir un caballo en una zebra en tiempo real
- Reconocimiento facial
- Reconocimiento del habla
- Clasificación de objetos
- Traducción de lenguajes



# Fake videos

AI-generated "real fake" video of Barack Obama



Source: <https://youtu.be/dkoi7sZvWlU>

Turning a horse video into a zebra video in real time using GANs

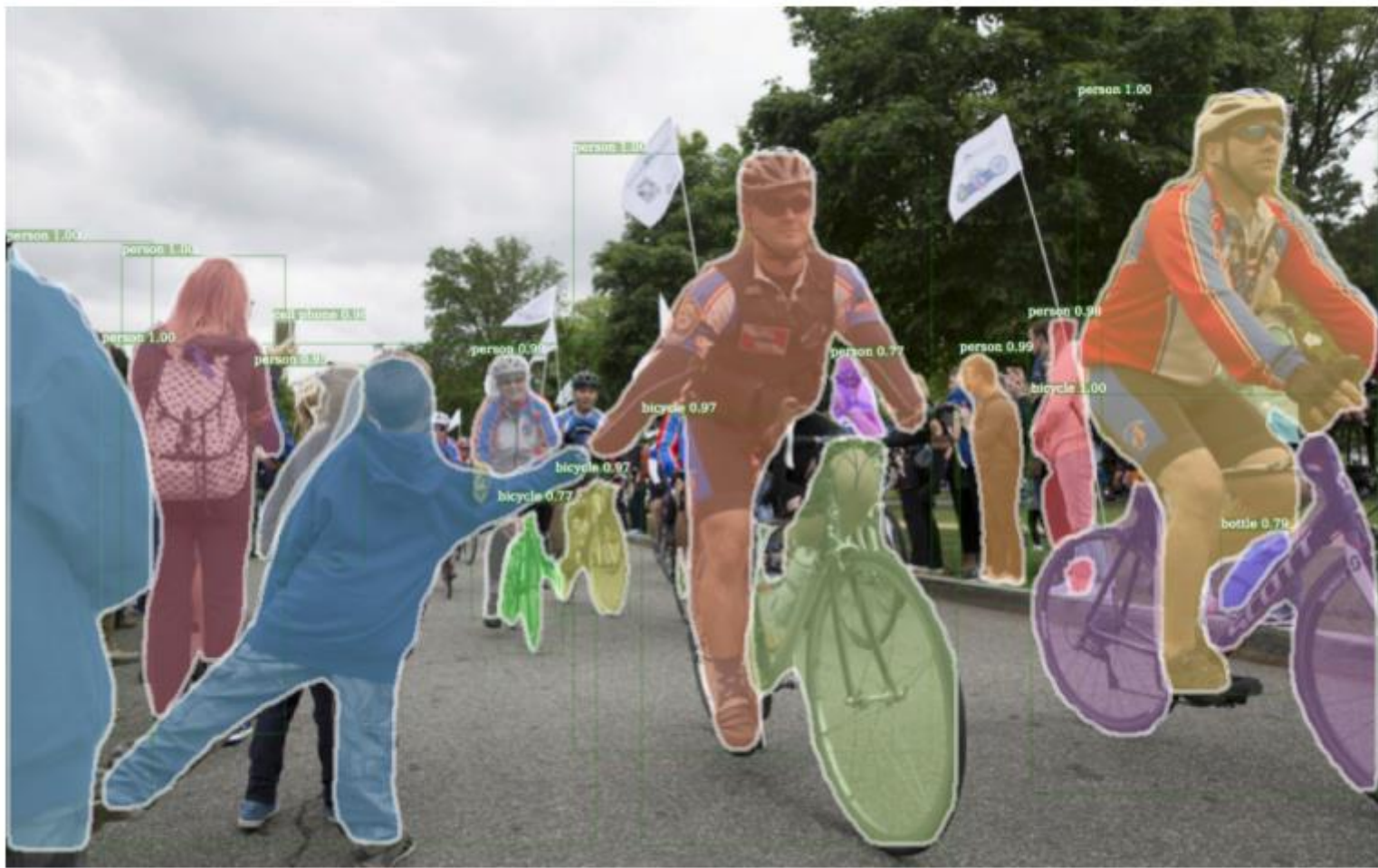


Source: <https://youtu.be/JzgOf1SLNjk>

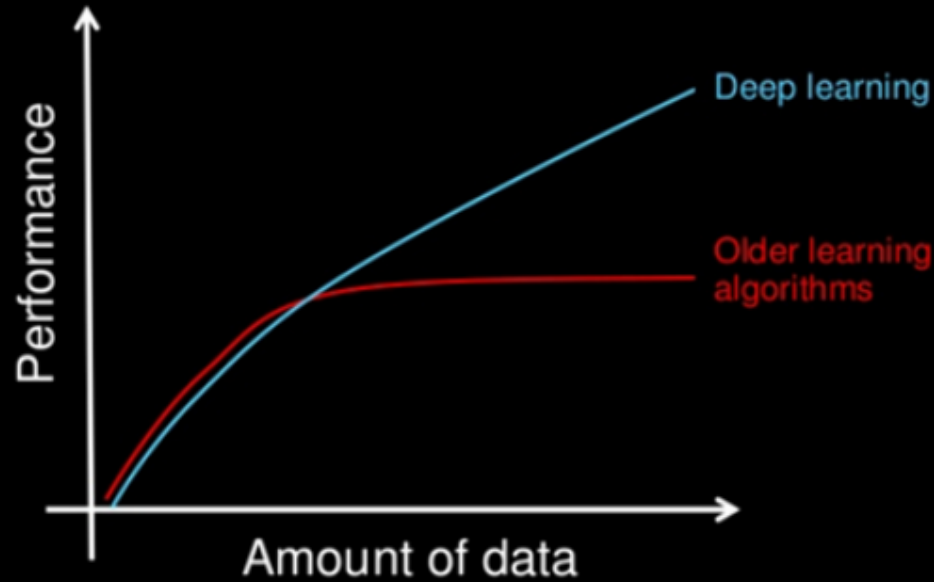
# Convertir día a noche



Source: <https://youtu.be/N7KbfWodXJE>



# Why deep learning



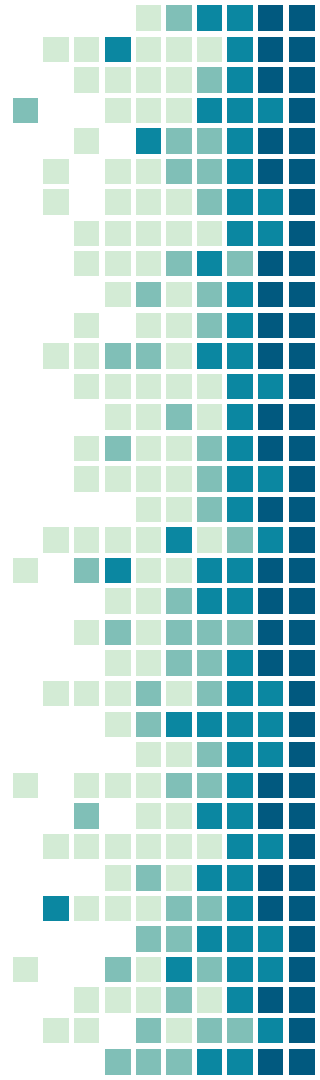
How do data science techniques scale with amount of data?

# ¿Redes Neuronales?



# ¿Qué son redes neuronales artificiales?

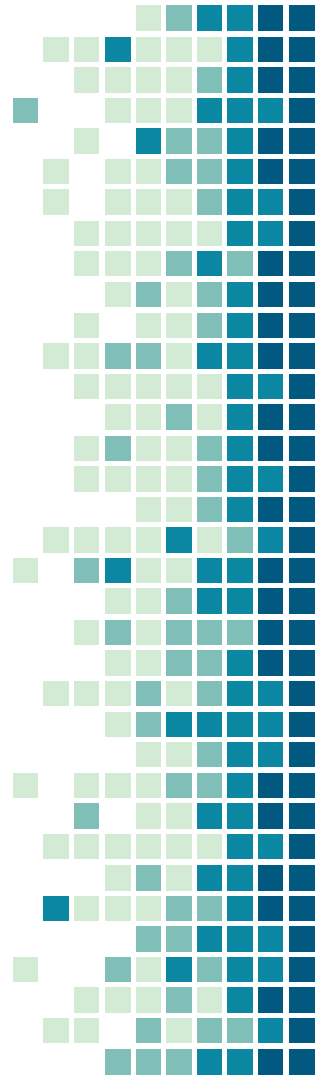
- Una neurona recibe una señal, la procesa y propaga la señal (o no)
- El cerebro está compuesto por alrededor de 100 billones de neuronas, cada una conectada a ~10 mil otras neuronas.
- ANN son una imitación simplista del cerebro compuesto de una red densa de simples estructuras



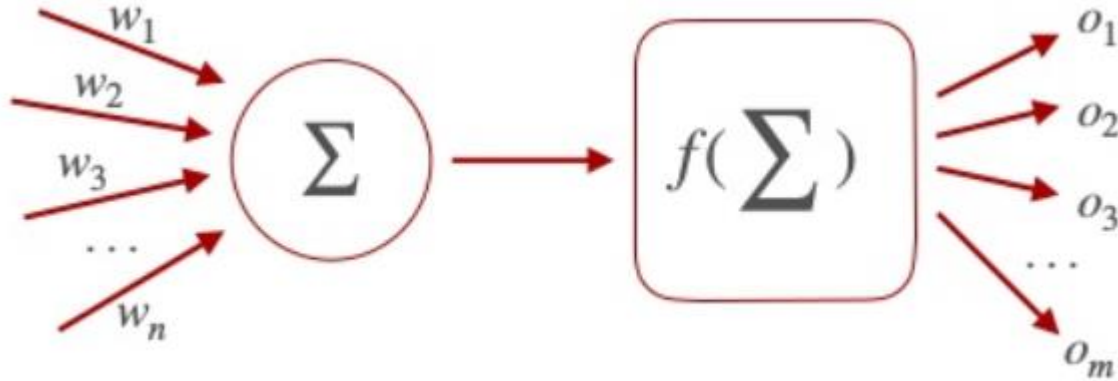


# Modelo matemático conceptual

- Recibe input de  $n$  fuentes
- Calcula la suma ponderada
  - $$h_1 = x_1w_1 + x_2w_2 + \cdots x_nw_n$$
- Pasa por un función de activación
- Envía la señal a ***m*** neuronas siguientes



# Modelo matemático conceptual



“ Una sola neurona en el cerebro es una máquina increíblemente compleja que aún hoy no entendemos. Una sola "neurona" en una red neuronal es una función matemática increíblemente simple que captura una fracción minúscula de la complejidad de una neurona biológica. Por lo tanto, decir que las redes neuronales imitan al cerebro, eso es cierto en el nivel de la inspiración floja, pero las redes neuronales realmente artificiales no son nada como lo que hace el cerebro biológico.

- Andrew Ng

“ Una sola neurona en el cerebro es una máquina increíblemente compleja que aún hoy no entendemos. Una sola "neurona" en una red neuronal es una función matemática increíblemente simple que captura una fracción minúscula de la complejidad de una neurona biológica. Por lo tanto, decir que las redes neuronales imitan al cerebro, eso es cierto en el nivel de la inspiración floja, pero las redes neuronales realmente artificiales no son nada como lo que hace el cerebro biológico.

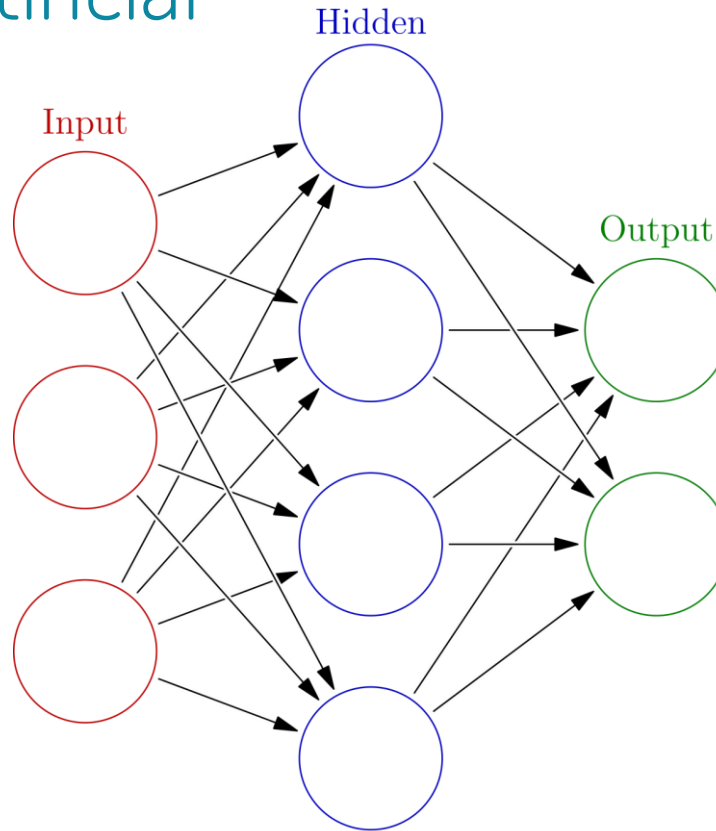
- Andrew Ng

# Red Neuronal Artificial

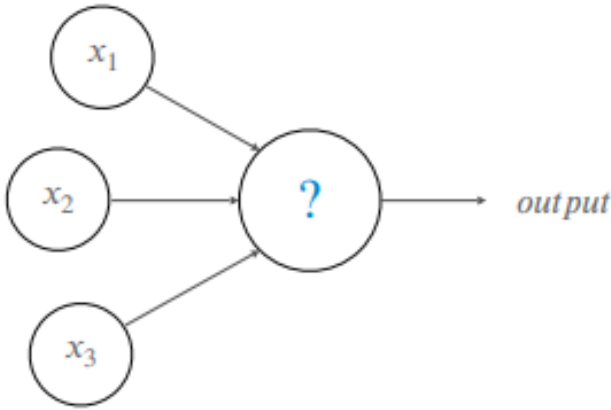
- Organizada en capas (layers) de neuronas como un modelo de caja negra
- Normalmente 3 o más: input, hidden y output



# Red Neuronal Artificial



# Perceptron: Neurona Artificial Simple



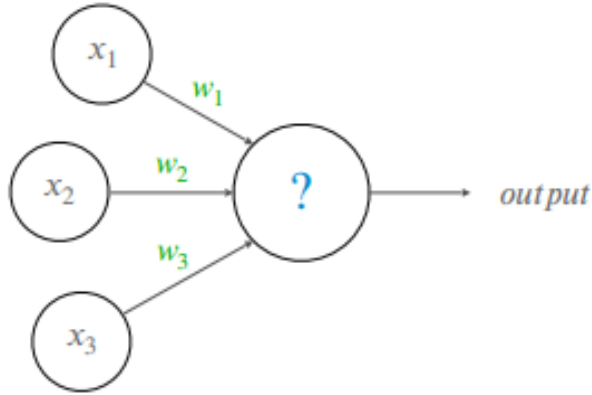
¿Debería ir a montar este fin de semana?

X1 -> ¿está bueno el clima?

X2 -> ¿hay mucho polvo?

X3 -> ¿estoy en el humor de manejar?

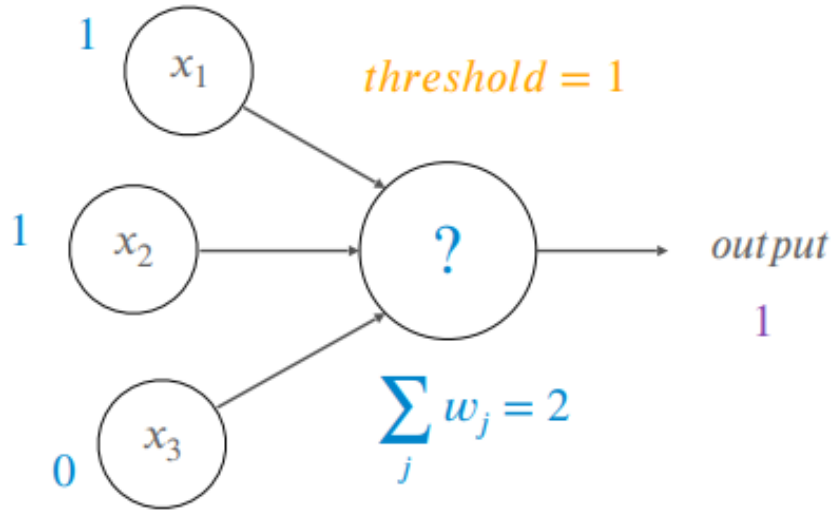
# Perceptron: Neurona Artificial Simple



$$output = \begin{cases} 0, & \sum_{j=0}^n w_j x_j \leq \text{threshold} \\ 1, & \sum_{j=0}^n w_j x_j > \text{threshold} \end{cases}$$



# Perceptron: Neurona Artificial Simple



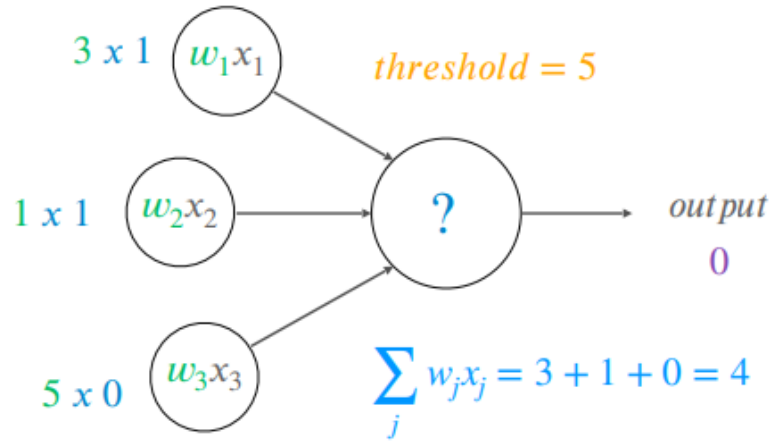
¿Debería ir a montar este fin de semana?

X1 = 1 (buen clima)

X2 = 1 (mucho polvo)

X3 = 0 (manejar es horrible)

# Perceptron: Neurona Artificial Simple



¿Debería ir a montar este fin de semana?

$X_1 = 1$  (buen clima)

$W_1 = 3$

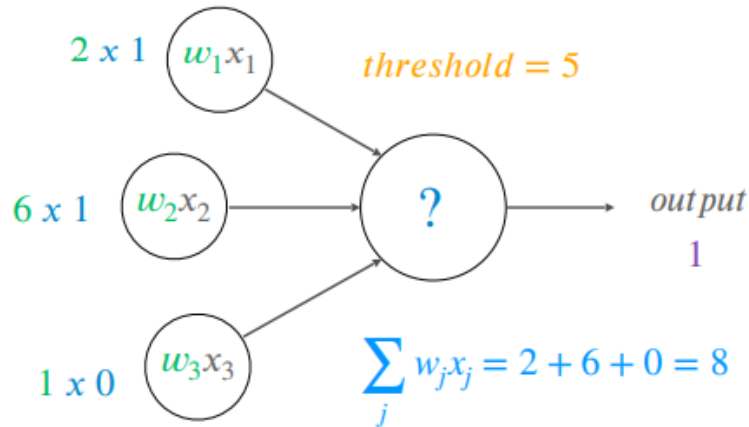
$X_2 = 1$  (mucho polvo)

$W_2 = 1$

$X_3 = 0$  (manejar es horrible)

$W_3 = 5$

# Perceptron: Neurona Artificial Simple



¿Debería ir a montar este fin de semana?

$X_1 = 1$  (buen clima)

$W_1 = 2$

$X_2 = 1$  (mucho polvo)

$W_2 = 6$

$X_3 = 0$  (manejar es horrible)

$W_3 = 1$

# Introduciendo Sesgo

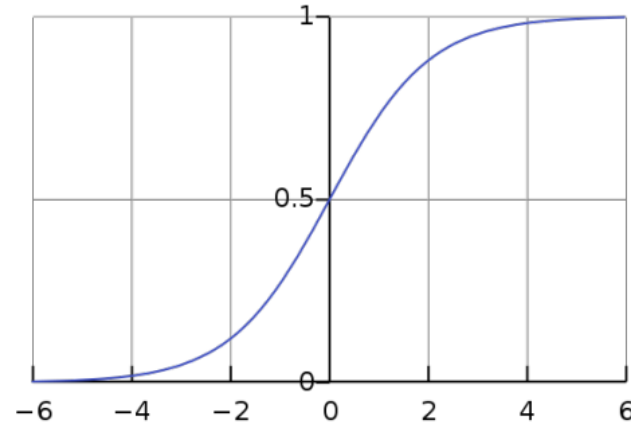
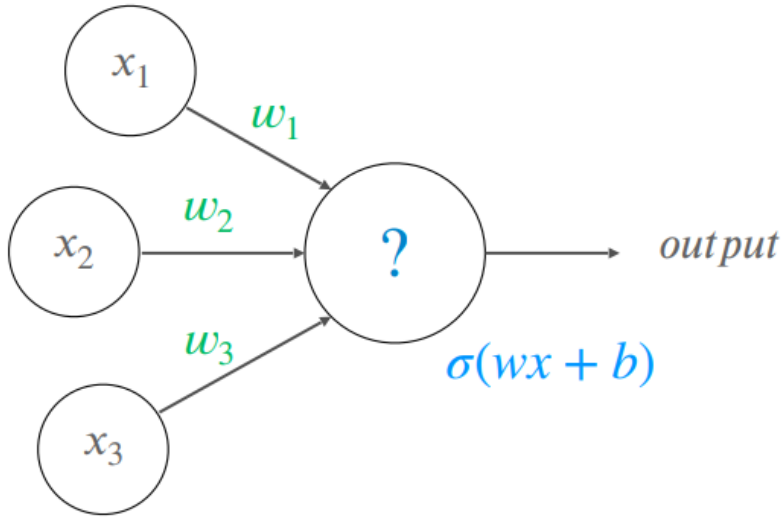
- Perceptron necesita tomar en cuenta el sesgo

$$output = \begin{cases} 0, & wx + b \leq 0 \\ 1, & wx + b > 0 \end{cases}$$

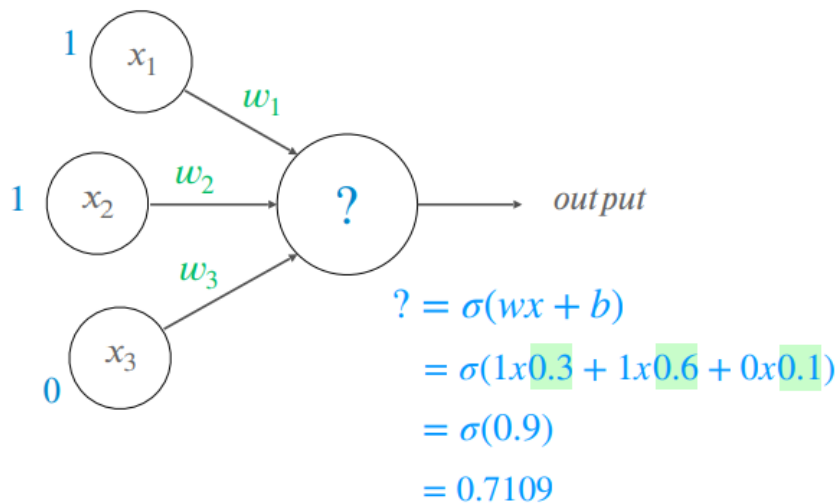
Donde  $b$  es que tan sencillo es que disparar el perceptrón

# Neuronal Sigmoid

- Es la Neurona Artificial más común



# Neurona Sigmoid



¿Debería ir a montar este fin de semana?

X1 = 1 (buen clima)

W1 = 0.3

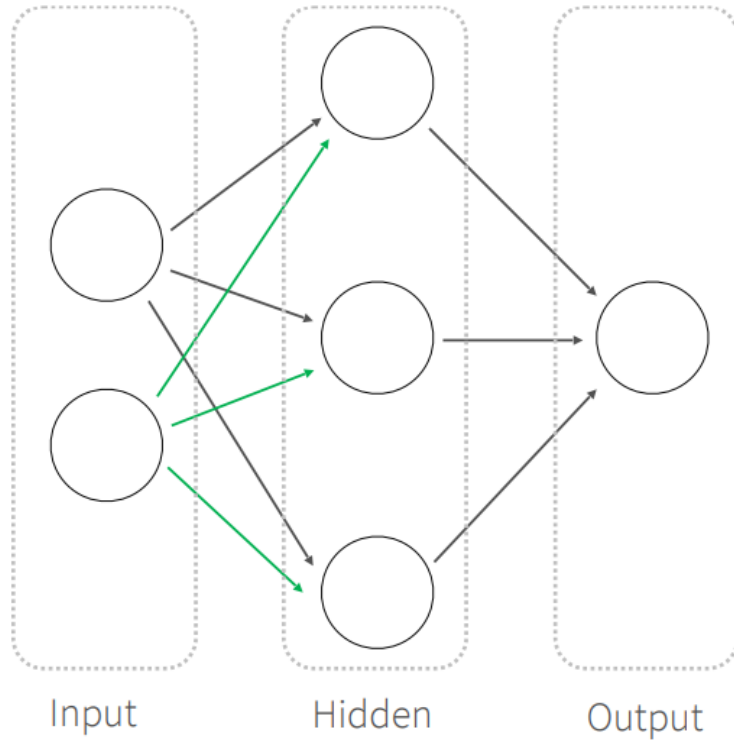
X2 = 1 (mucho polvo)

W2 = 0.6

X3 = 0 (manejar es horrible)

W3 = 0.1

# ANN Simple de 2 Capas



**¿Debería ir a montar este fin de semana?**

X1 -> Entrada 1

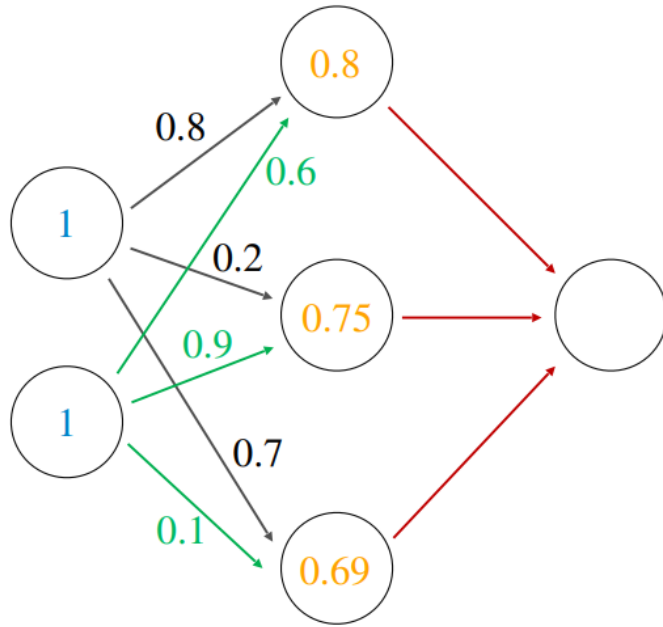
X2 -> Entrada 2

H1 -> Clima

H2 -> Polvo

H3 -> Manejar

# ANN Simple de 2 Capas



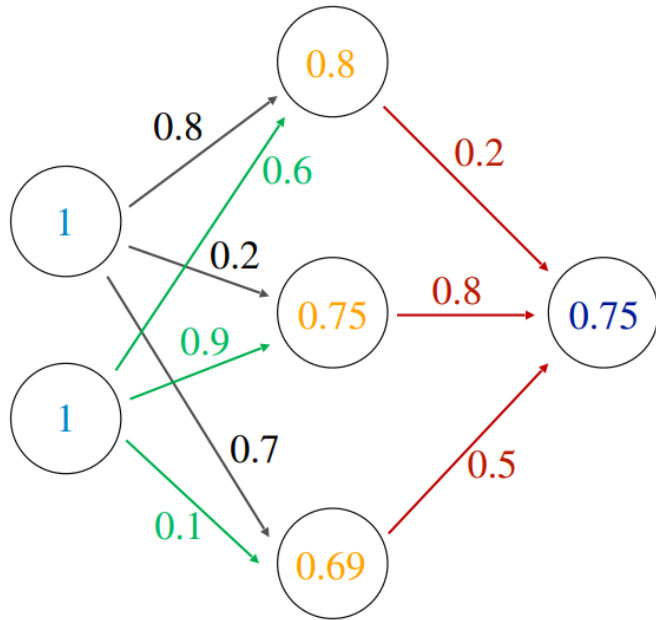
$$h_1 = \sigma(1 \times 0.8 + 1 \times 0.6) = 0.80$$

$$h_2 = \sigma(1 \times 0.2 + 1 \times 0.9) = 0.75$$

$$h_3 = \sigma(1 \times 0.7 + 1 \times 0.1) = 0.69$$



# ANN Simple de 2 Capas

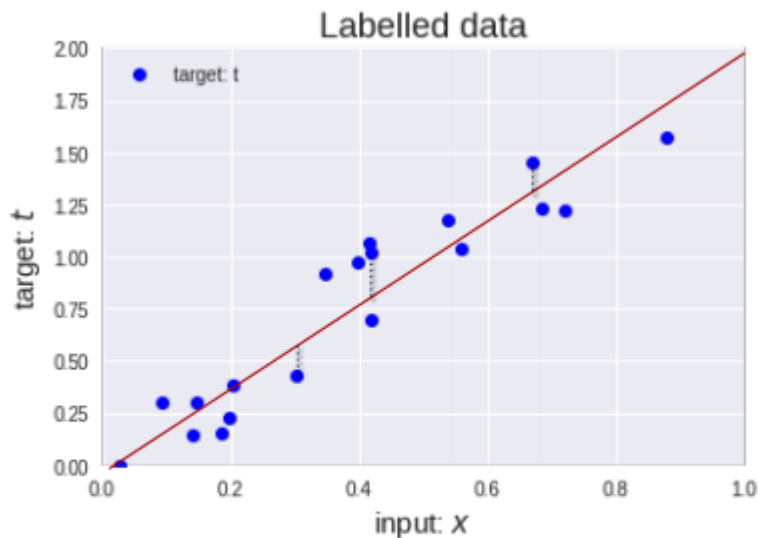


$$out = \sigma(0.2 \times 0.8 + 0.8 \times 0.75 + 0.5 \times 0.69)$$

$$= \sigma(1.105)$$

$$= 0.75$$

# Función de costo



Source: <https://bit.ly/2loAGzL>

Para este ejemplo de regresión lineal, para determinar la mejor  $p$ .

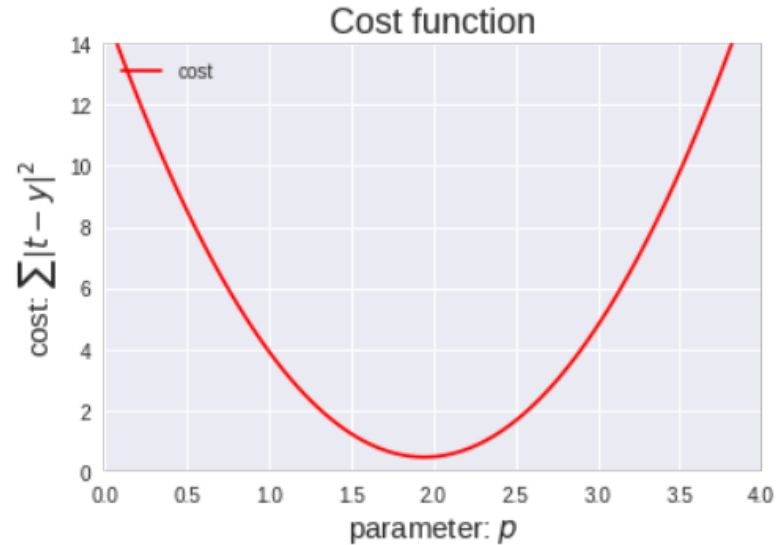
$$y = x * p$$

Podemos calcular la **función costo** como MSE, MAE, SVM Loss, etc...

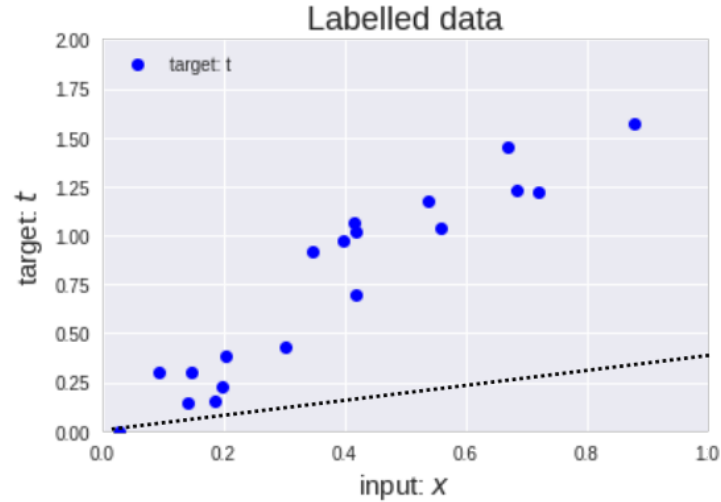
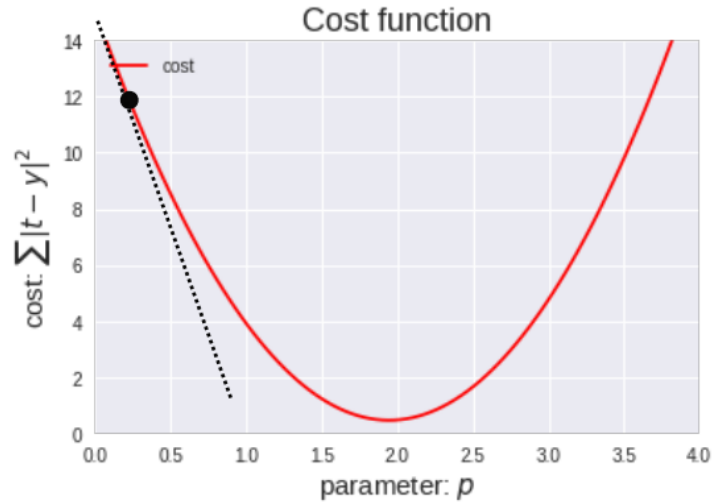
Para el ejemplo, usaremos **Suma Cuadrada Absoluta**.

$$cost = \sum |t - y|^2$$

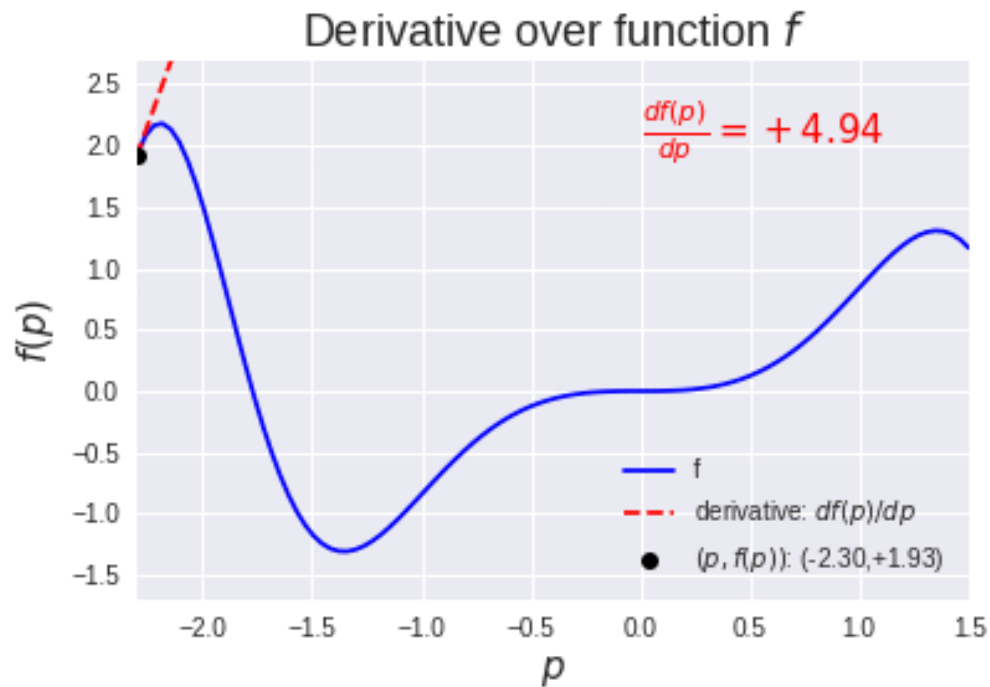
# Visualiza esta función



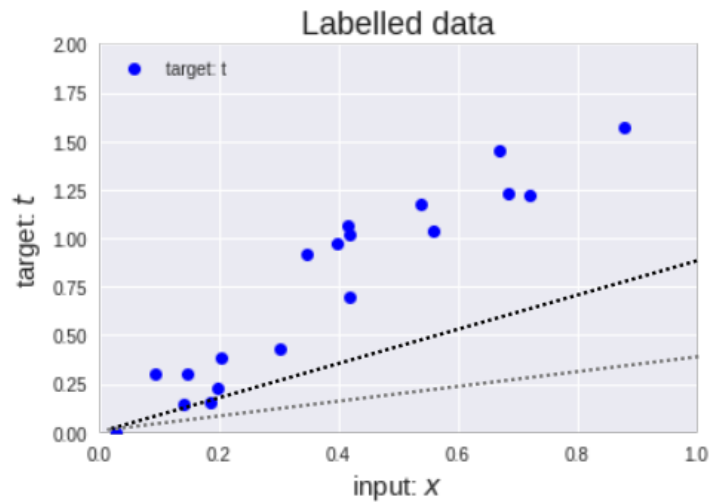
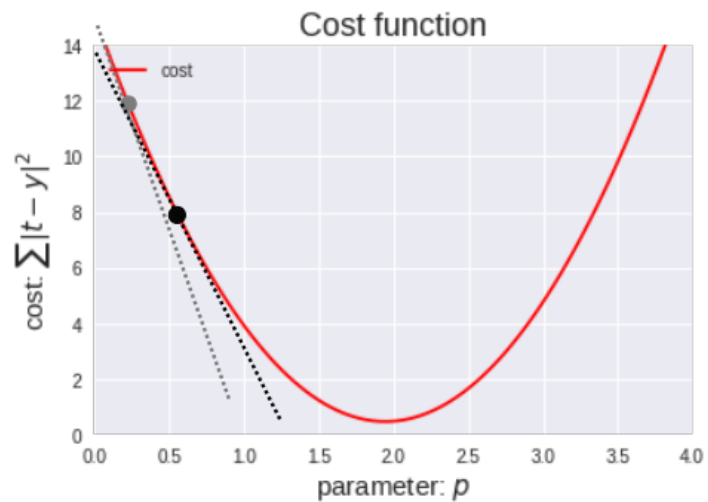
# Calcula su derivada



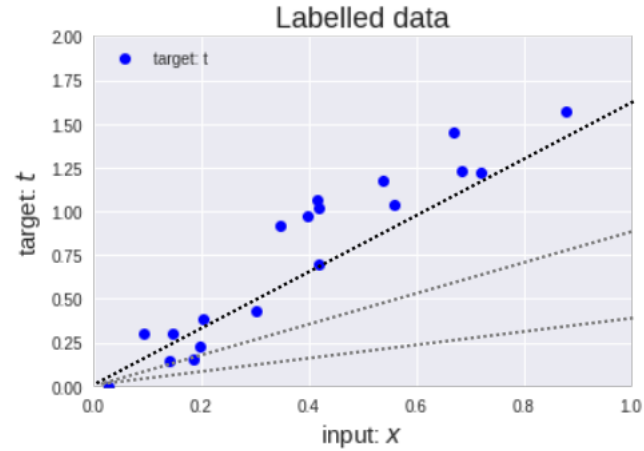
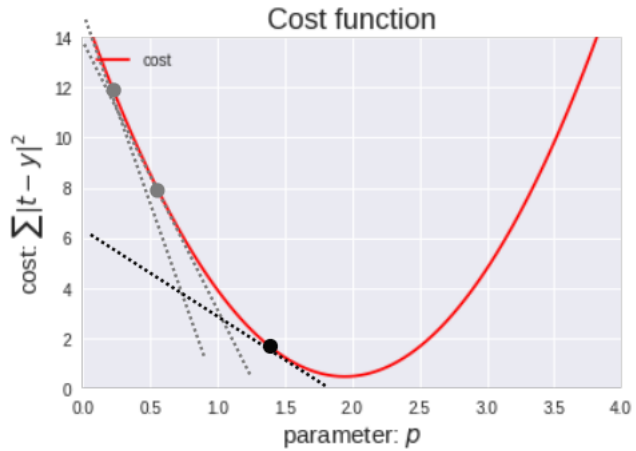
# ¿Por qué?



# Gradient Descent



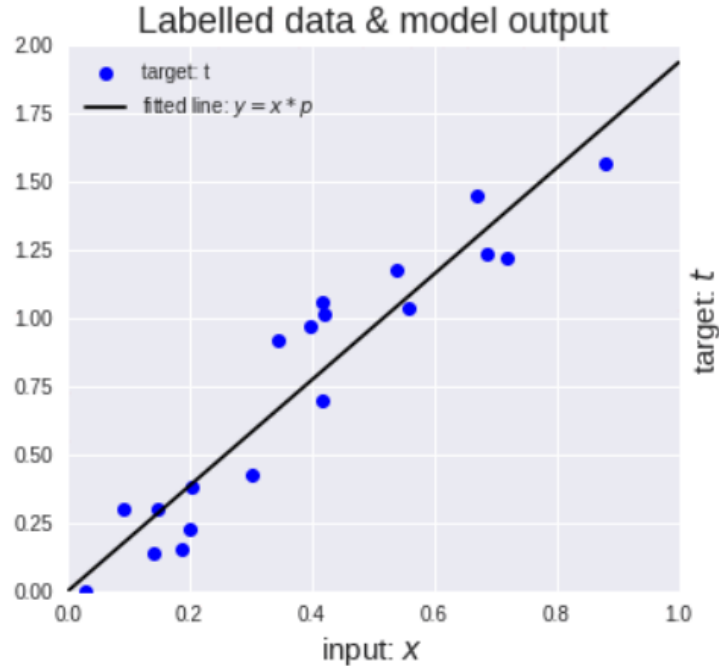
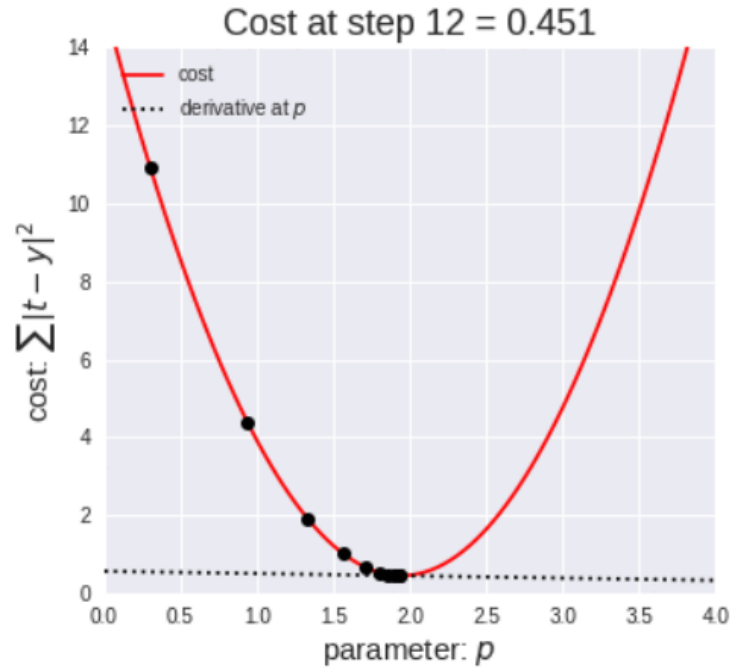
# Gradient Descent



El objetivo es encontrar el punto más bajo de la función costo (ejemplo, menor cantidad de costo o el mínimo o mínima)

Gradient Descent iterativamente (es decir, paso a paso) desciende la curva de la función de costo para encontrar los mínimos.

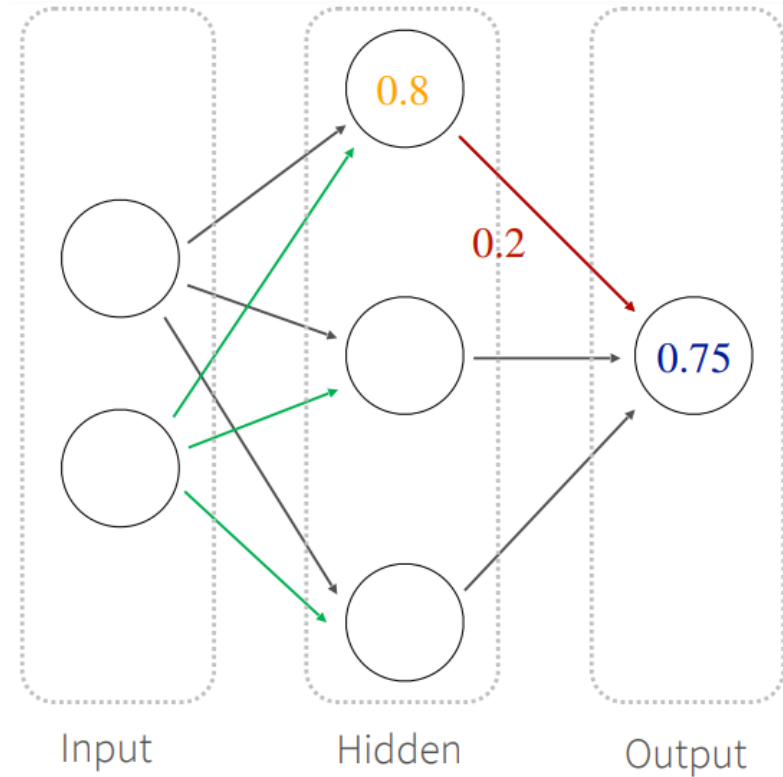
# Gradient Descent Optimización



Source: <https://bit.ly/2IoAGzL>



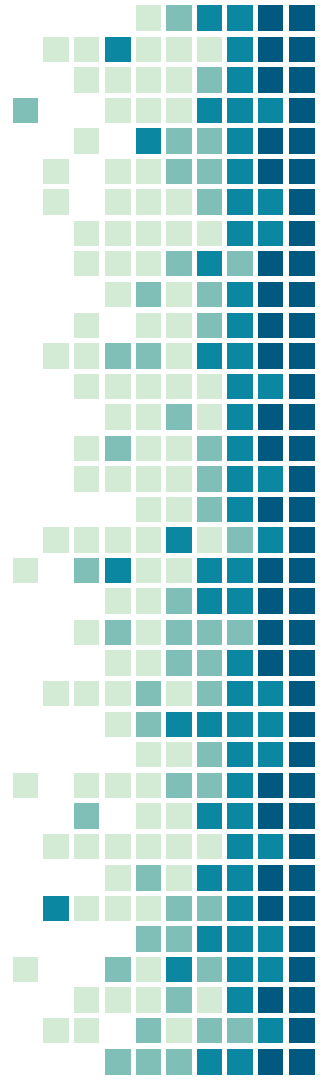
# Backpropagation



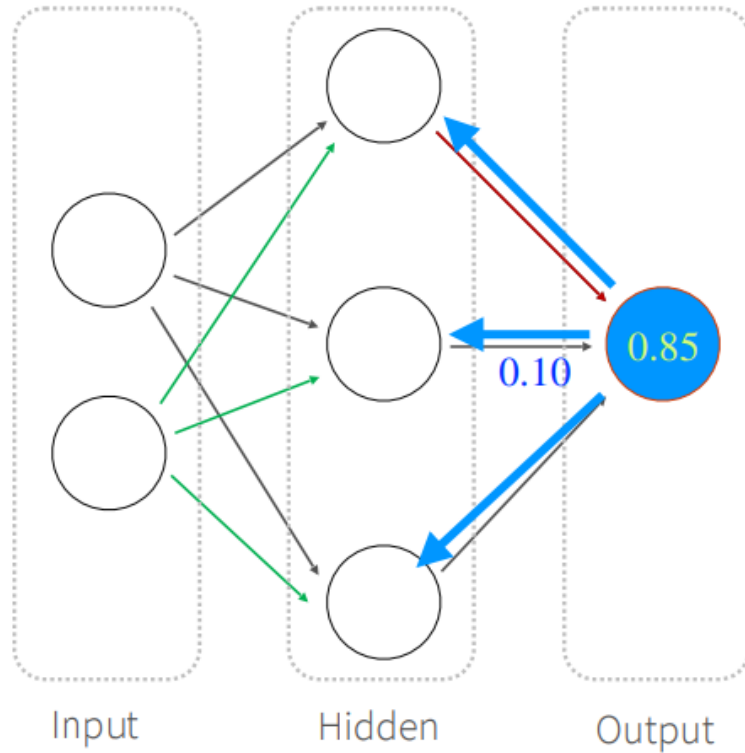
# Backpropagation

Calcula el gradiente de la función de costo en un red neuronal

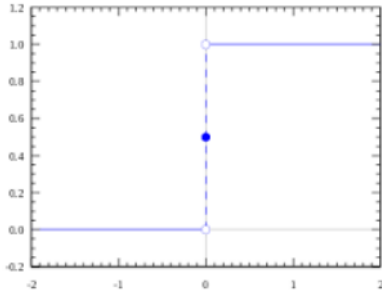
- Utilizado por el algoritmo de optimización de **descenso de gradiente** para ajustar el *peso* de las neuronas
- También conocido como **propagación hacia atrás de errores**, ya que el error se calcula y distribuye a través de la red de capas.



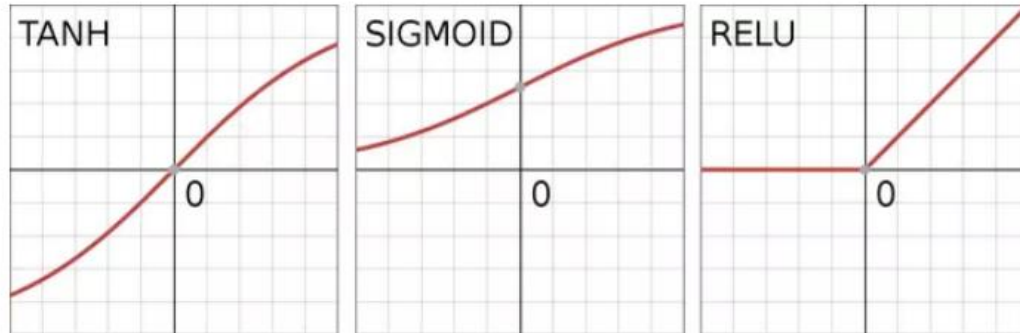
# Backpropagation



# Funciones de Activación



Source: <https://bit.ly/2ww3kcd>



Source: <http://bit.ly/2tKhhbMS>

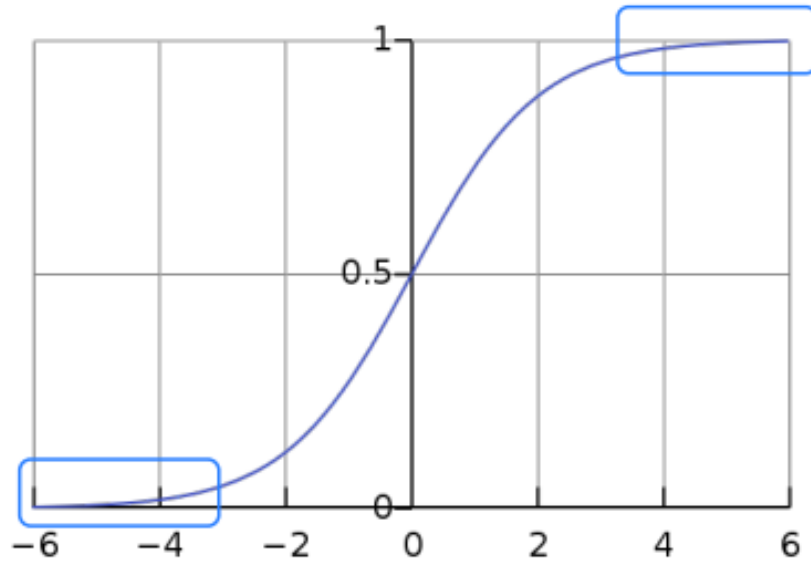
# Funciones de Activación

Sigmoide y Tanh introducen la no linealidad con diferentes dominios

**ReLU** es uno de los más populares porque es simple de calcular y muy robusto para entradas ruidosas.

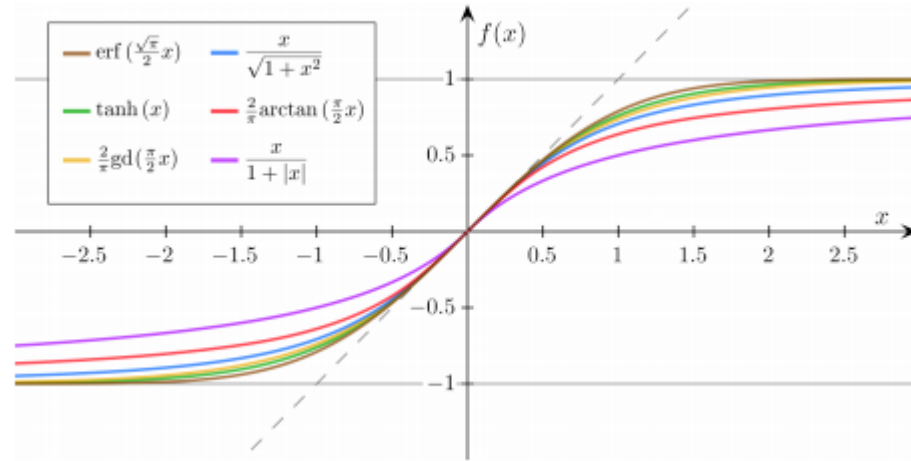


# Sigmoid



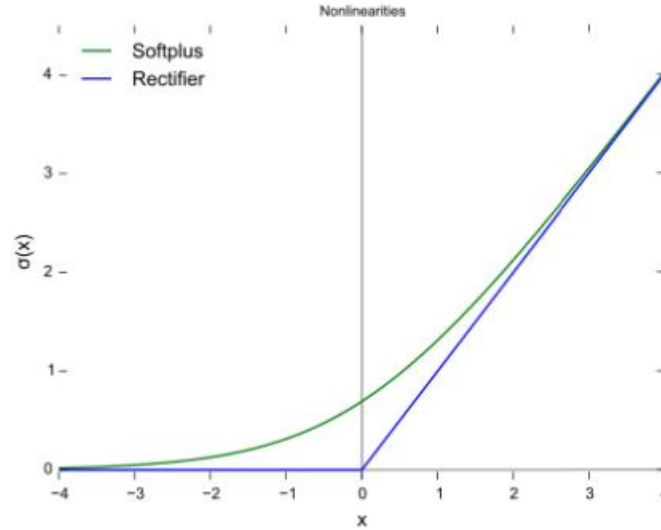
Actualmente, no se usa tanto porque valores realmente grandes demasiado cercanos a 0 o 1 dan como resultado gradientes demasiado cercanos a 0 que detienen la propagación hacia atrás

# Tanh








El mismo problema que sigmoide es que sus activaciones se saturan matando gradientes.

# ReLU

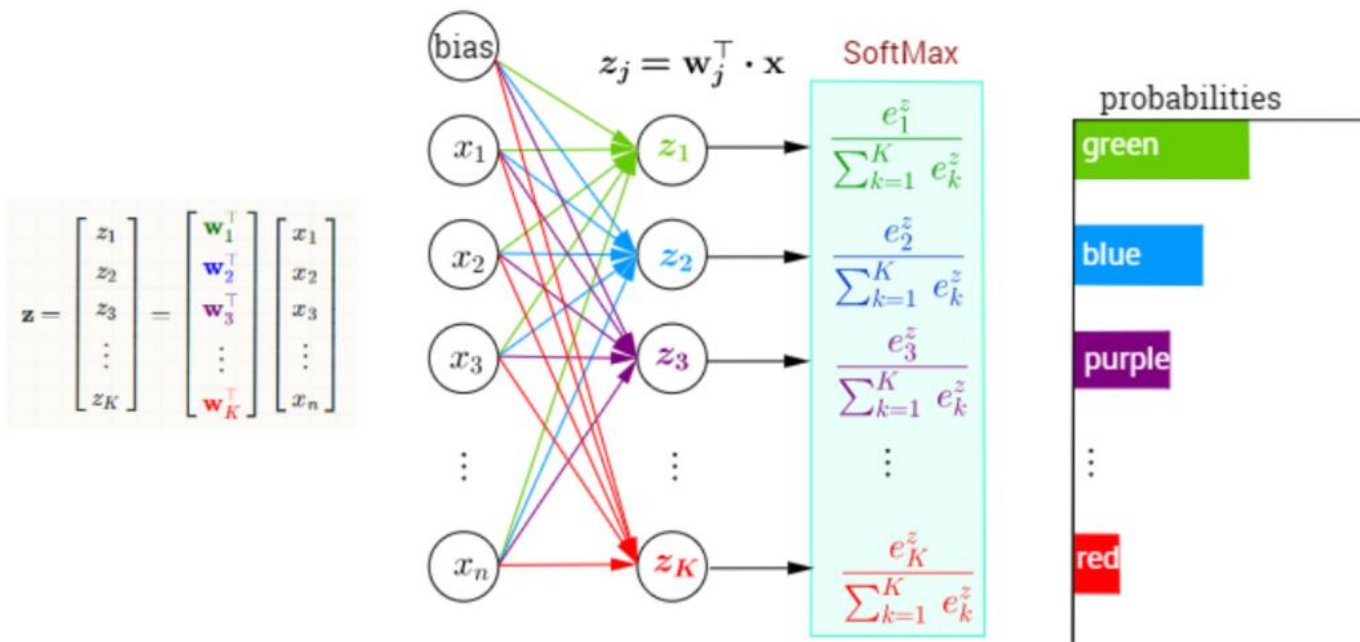


Muy popular en los últimos años. Es más fácil de implementar debido a funciones matemáticas más simples.



Name	Visualization	$f(x) =$	Notes
Linear (= Identity)		$x$	Not useful for hidden layers
Heaviside Step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	Not differentiable
Rectified Linear (ReLU)		$\begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	Surprisingly useful in practice
Tanh		$\frac{2}{1+e^{-2x}} - 1$	A soft step function; ranges from -1 to 1
Logistic ('sigmoid')		$\frac{1}{1+e^{-x}}$	Another soft step function; ranges from 0 to 1

# Softmax Function



# TensorFlow

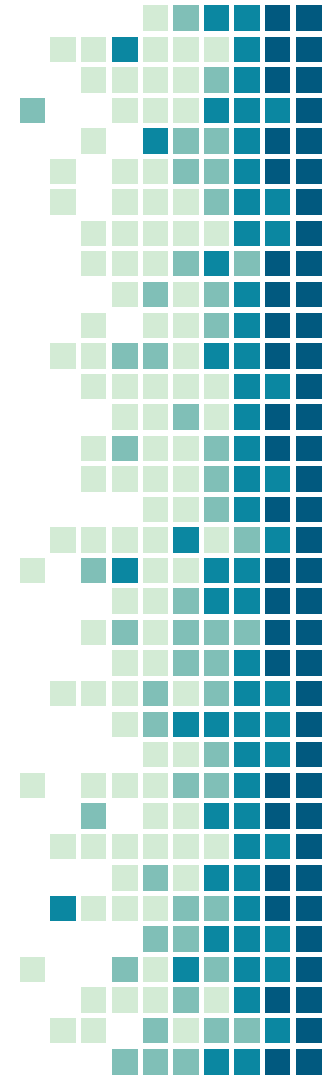


Biblioteca de software de código abierto para  
computación numérica utilizando datos  
gráficos de flujo  
Construido en C ++ con una interfaz Python  
Admite cálculos de GPU

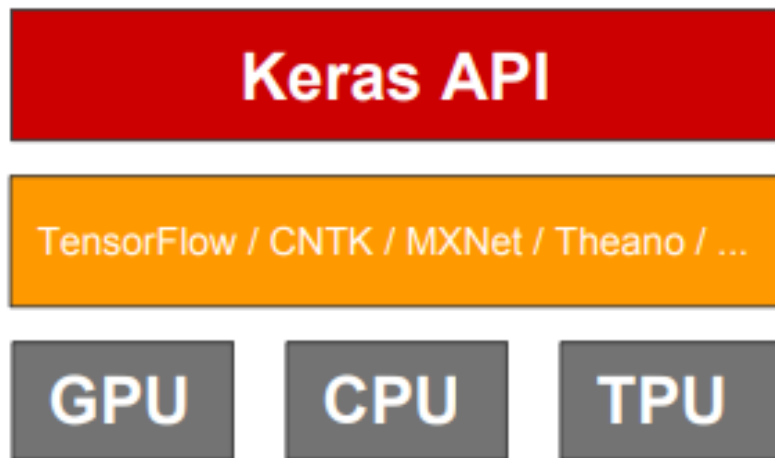


# Keras

- Biblioteca de aprendizaje profundo de Python
- API de redes neuronales de alto nivel
- Se ejecuta sobre TensorFlow, CNTK o Theano
- Más simple y fácil de usar, lo que facilita la creación rápida de prototipos

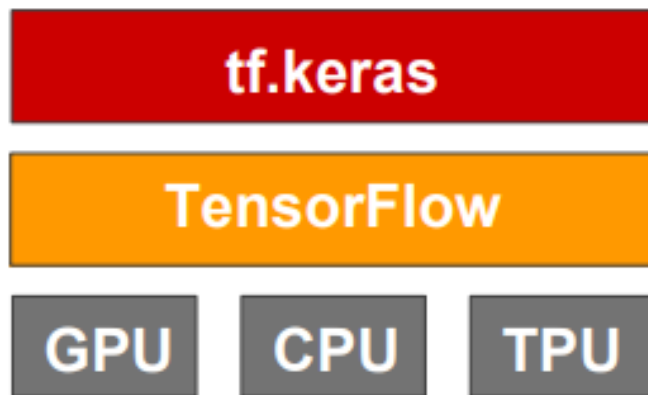


# Keras



# Keras: API oficial de alto nivel de TensorFlow

- Parte del núcleo TensorFlow desde v1.4
- Mejor optimizado para TF



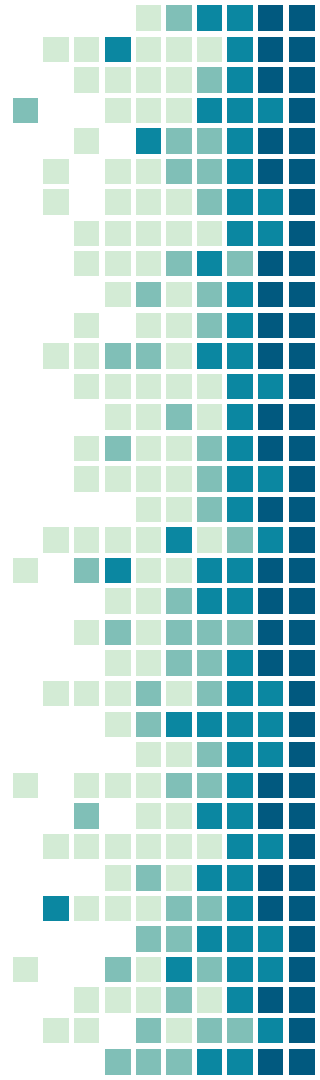
# Contribuidores y patrocinadores

 **633** contributors



# ¿Qué tiene especial Keras?

- Enfocada en la experiencia de usuarios
- Gran adopción en la industria y la comunidad de investigación.
- Multi-backend, multi-platform.
- Fácil productización de modelos.





# Tracción industrial

**NETFLIX**

**UBER**

**Google**

 **instacart**

 **HUAWEI**

 **nVIDIA.**

 **Square**

 **Expedia®**

 **Zocdoc**

**yelp.** 

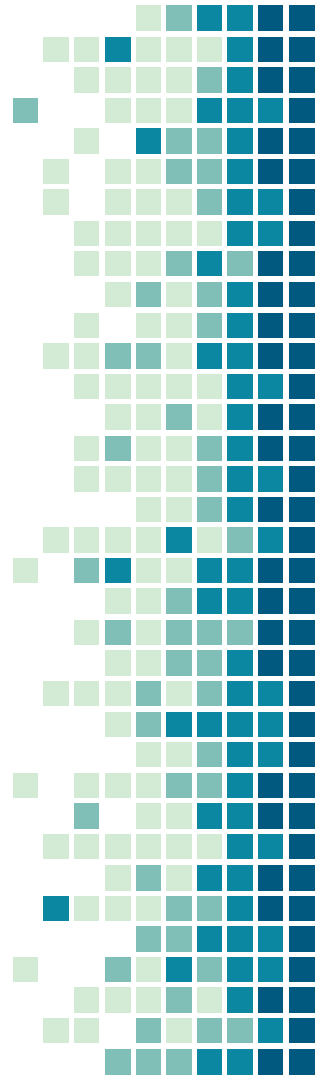
# Multi-backend, multi-platform

- Desarrollada en Python, R
  - En Unix, Windows, OSX
- Ejecute el mismo código con:
  - TensorFlow
  - CNTK
  - Theano
- CPU, Nvidia GPU, AMD GPU, TPU...



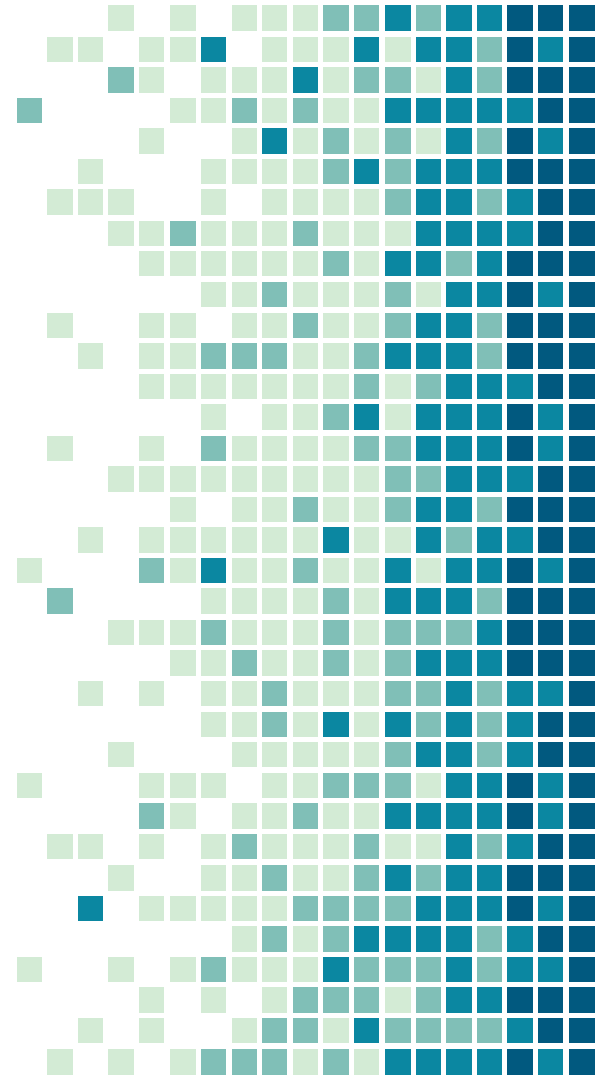
# Productización de modelos

- In-browser, con aceleración GPU (WebKeras, Keras.js, WebDNN...)
- Android (TF, TF Lite), iPhone (native CoreML support)
- Raspberry Pi
- JVM
- TF-Serving



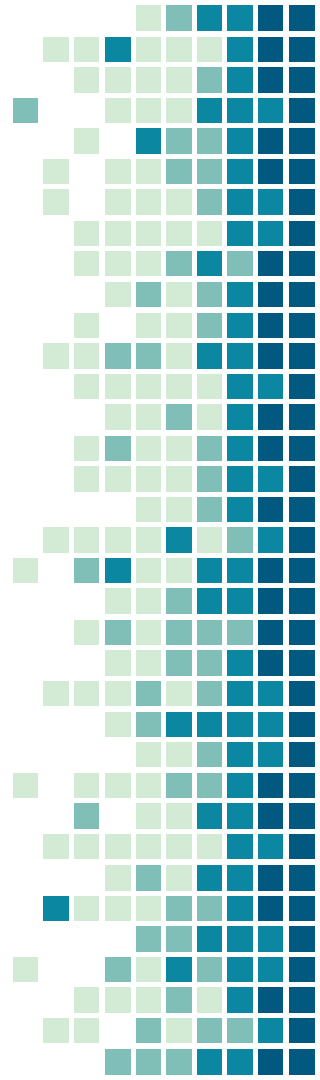
# ¿Cómo usar Keras?

Una introducción



# Tres estilos de API

- Sequential
  - Muy simple
  - Solo para capas secuenciales de entrada única, salida única
  - Bueno para más del 70% de los casos de uso
- Functional
  - Como jugar con ladrillos de Lego
  - Topologías de entrada múltiple y salida múltiple
  - Bueno para el 95% de los casos de uso
- Subclassing
  - Máxima flexibilidad
  - Mayor superficie de error potencial



# Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

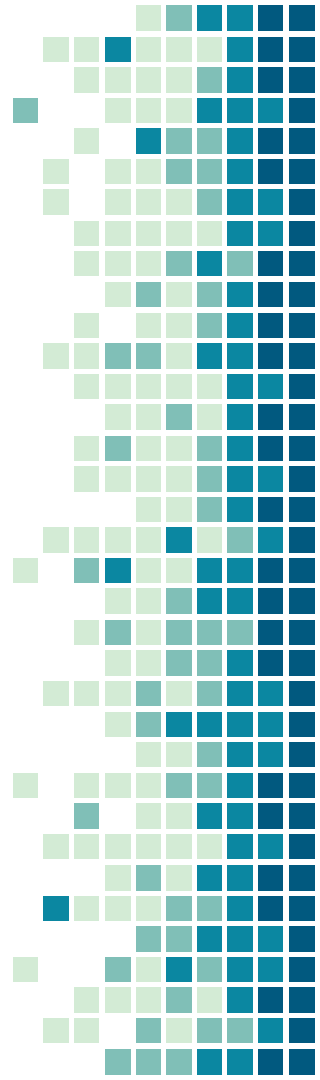
model.fit(x, y, epochs=10, batch_size=32)
```

# Functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```



# Model subclassing

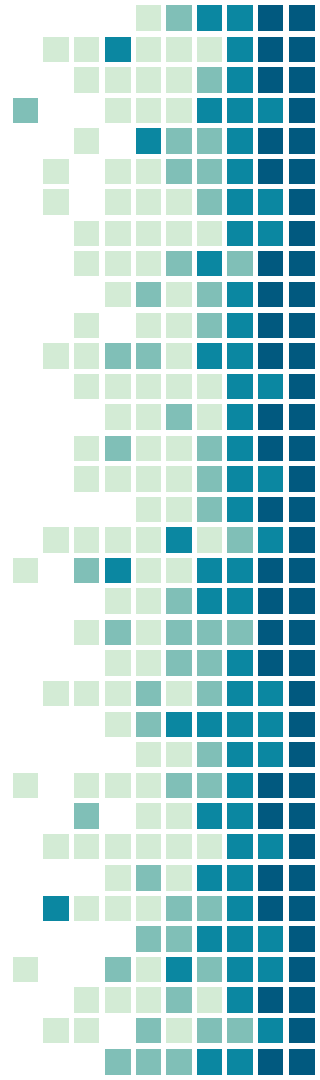
```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

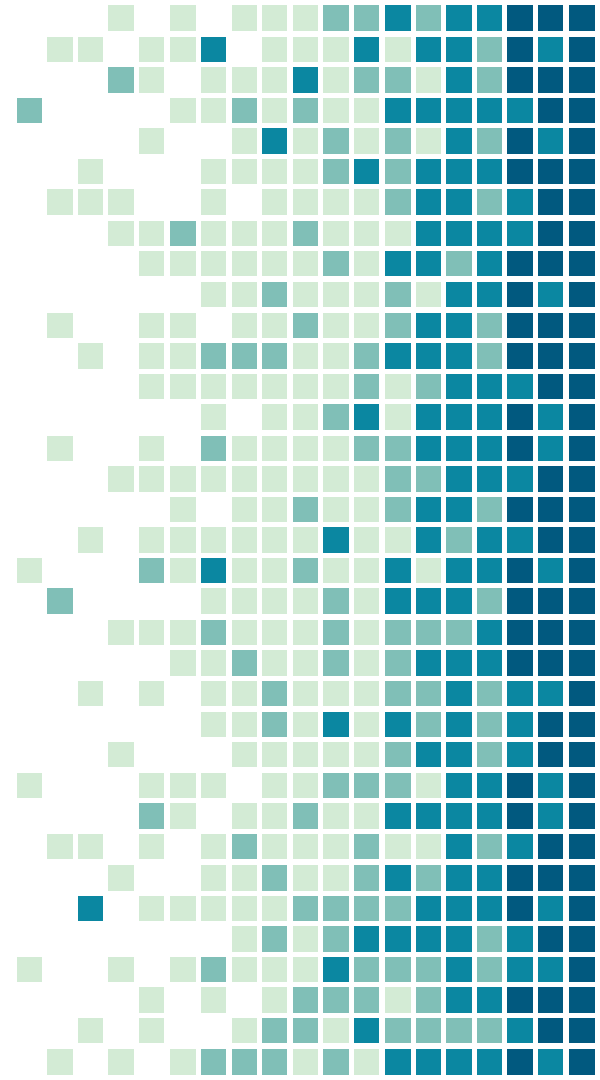
    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```





# ¿Cómo instalar Keras?



# ¿Cómo instalar Keras?

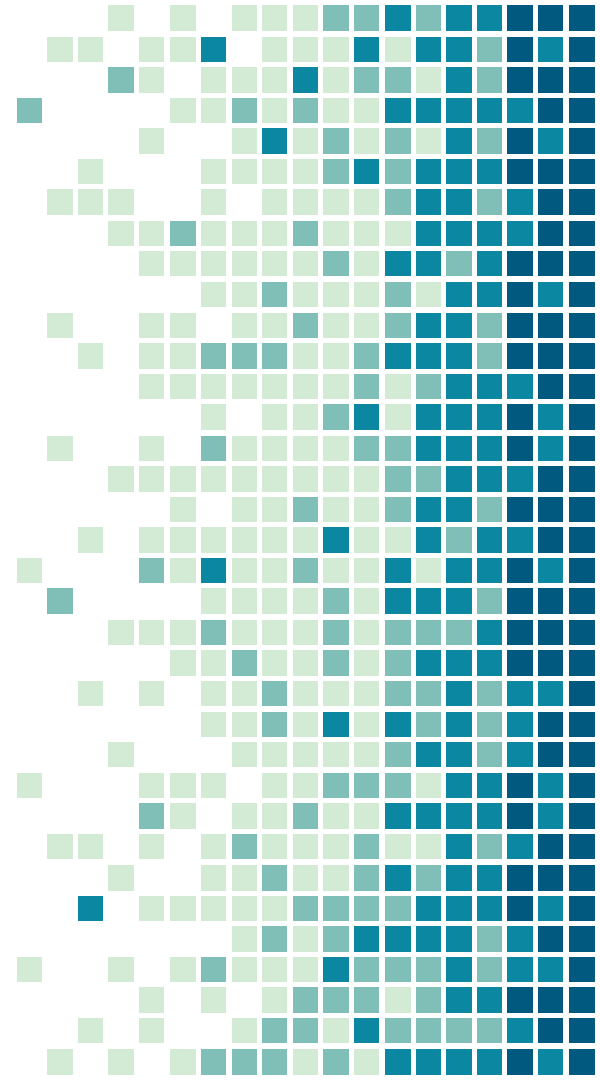
```
pip install keras
```

```
0
```

```
import tensorflow.keras
```

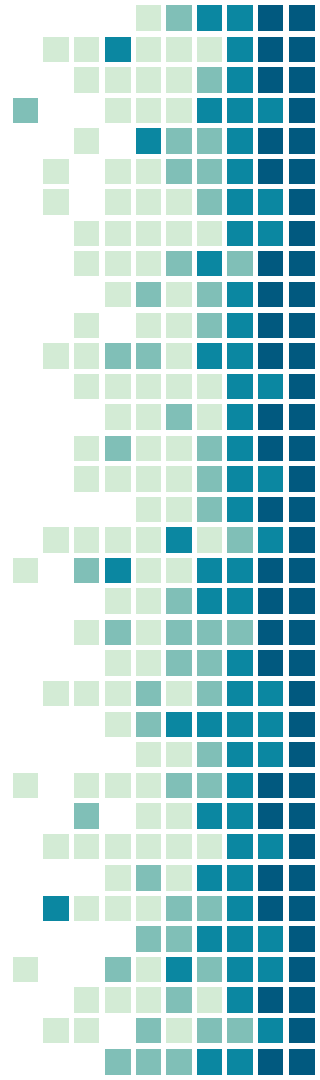


# ¿Cómo usar Keras?



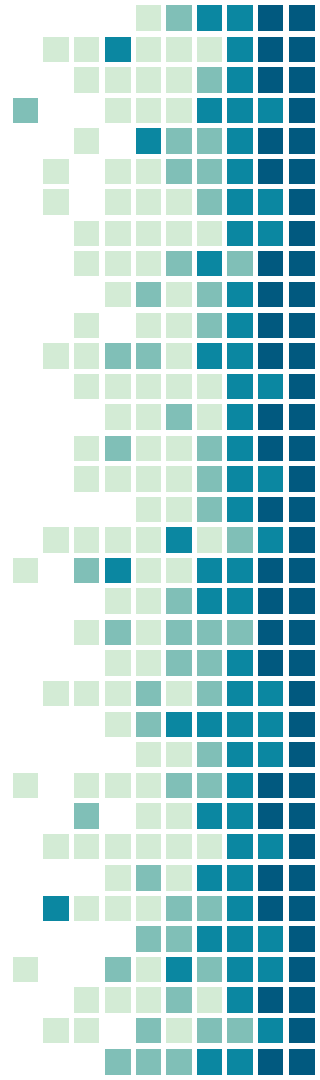
# Sequential API

1. from **keras.models** import **Sequential**
2. model = **Sequential()**



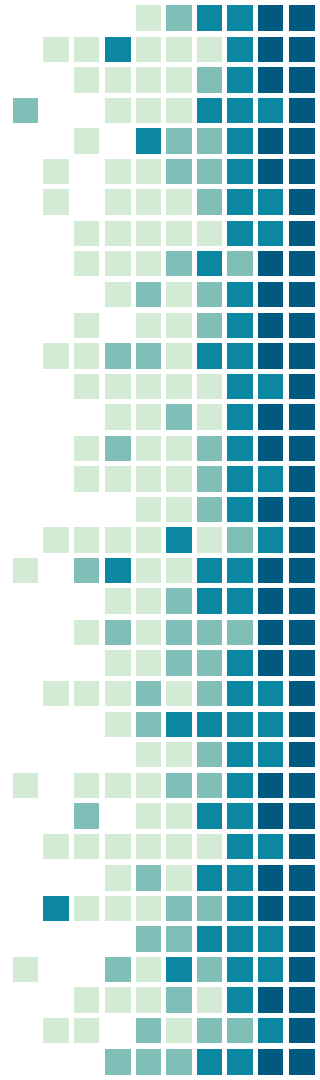
# Layers

1. from **keras.layers** import **Dense**
2. model.add(**Dense**(units=64, activation='relu',  
input\_dim=100))
3. model.add(**Dense**(units=10,  
activation='softmax'))



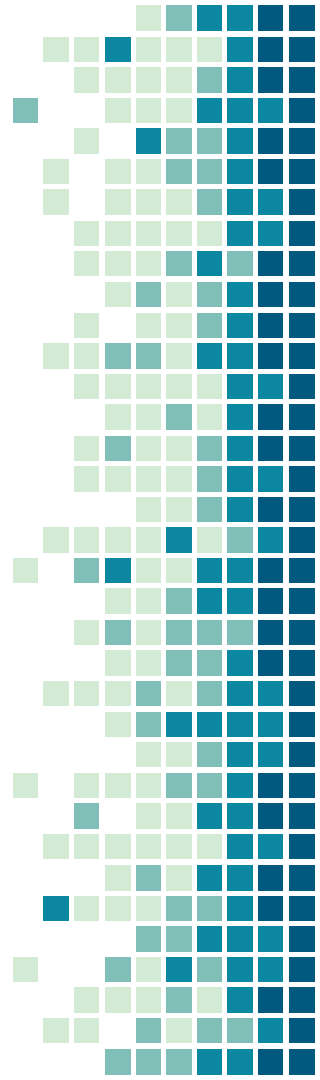
# Compile

1. `model.compile(loss='categorical_crossentropy',  
optimizer='sgd',  
metrics=['accuracy'])`

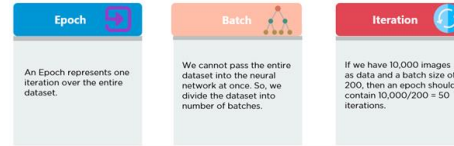


# Train

1. `model.fit(x=train, y=train, epochs=5, batch_size=32)`



# ¿Qué es batch y epoch?

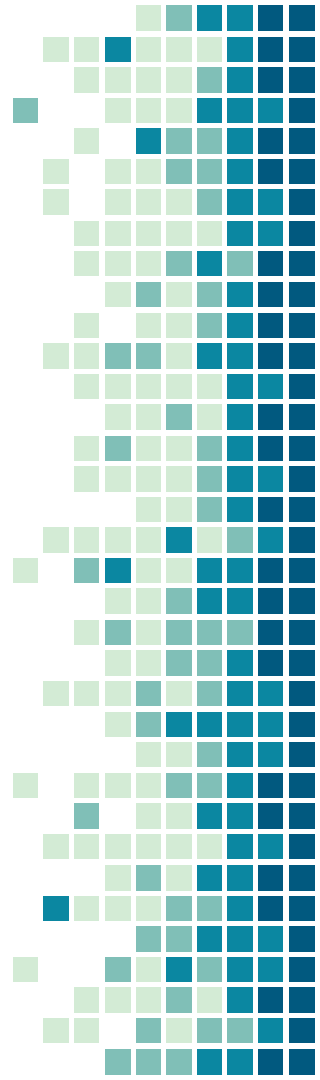


- El entrenamiento ocurre durante epochs y cada epoch se divide en batches.
  - **Epoch:** una pasada a través de todas las filas del conjunto de datos de entrenamiento.
  - **Batch:** una o más muestras consideradas por el modelo dentro de una época antes de actualizar los pesos.
- Un epoch se compone de uno o más batches, según el tamaño de batch elegido y el modelo es apto para muchos epochs.



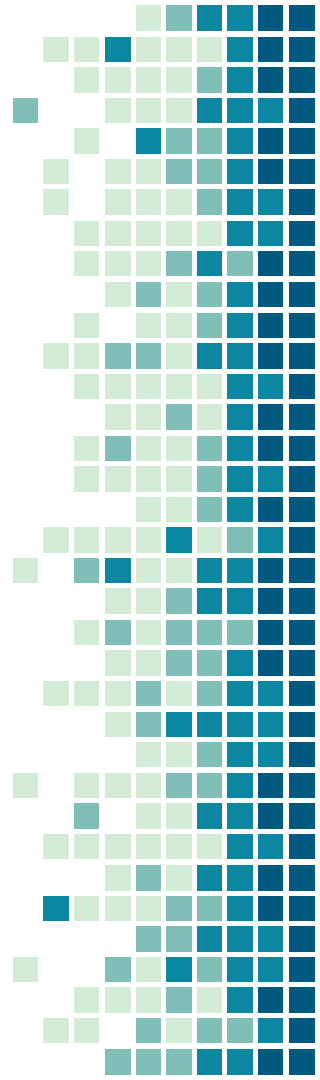
# Evaluate

1. `loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)`



# Predict

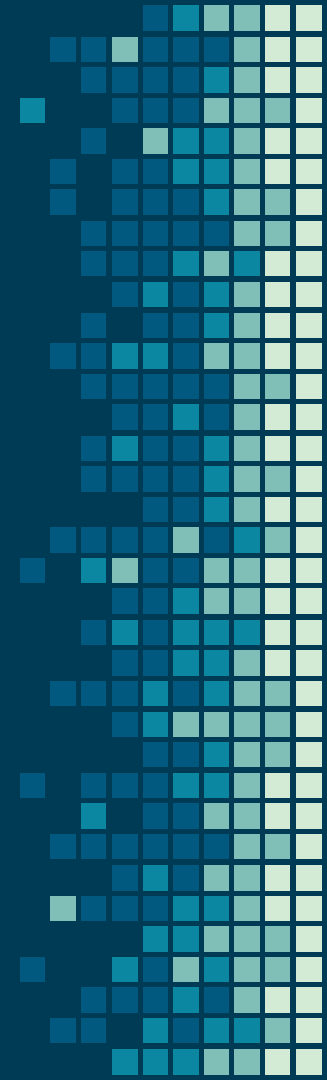
1. `classes = model.predict(x_test, batch_size=128)`



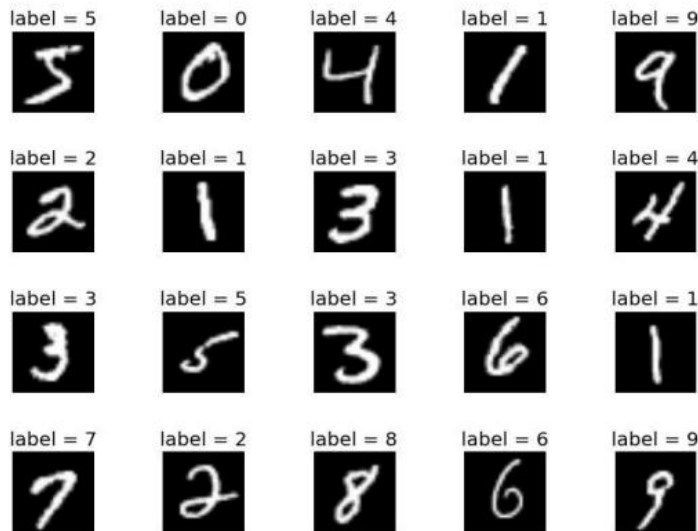
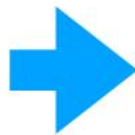


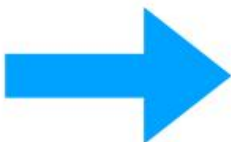
# Notebook

Visitar Google Colab





# MNIST: Hello World de Deep Learning





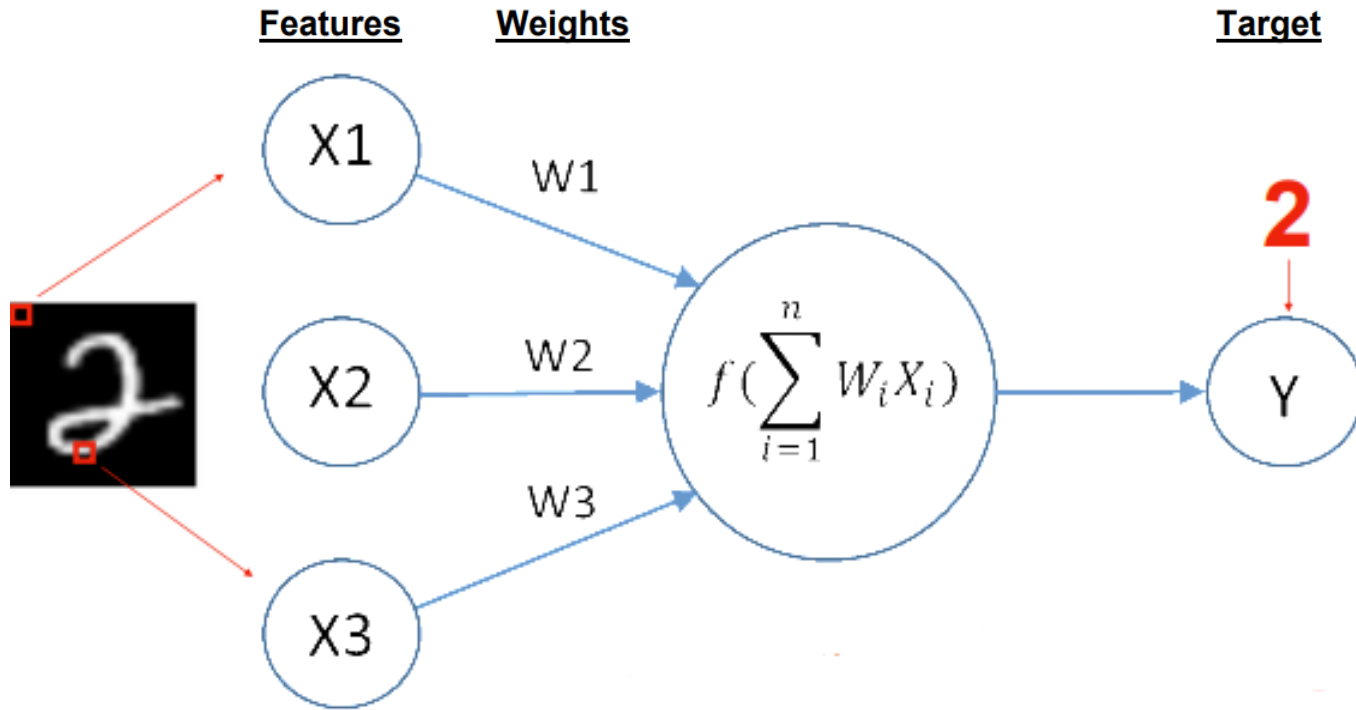
# MNIST: Training

(, 2), (, 8)

# MNIST: Test



# Logistic Regression

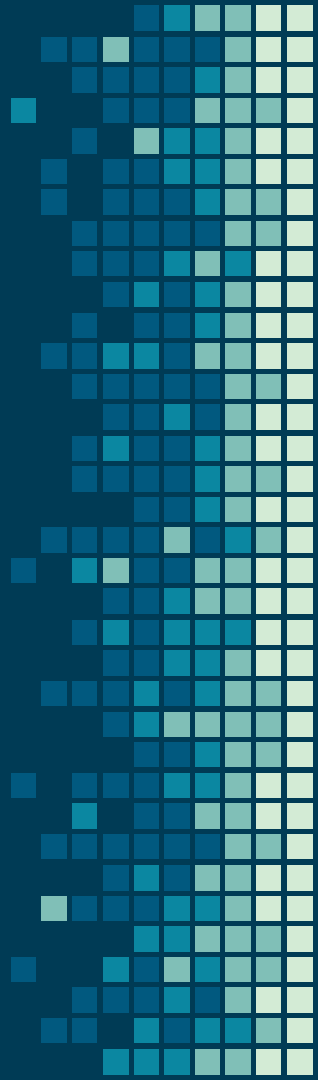




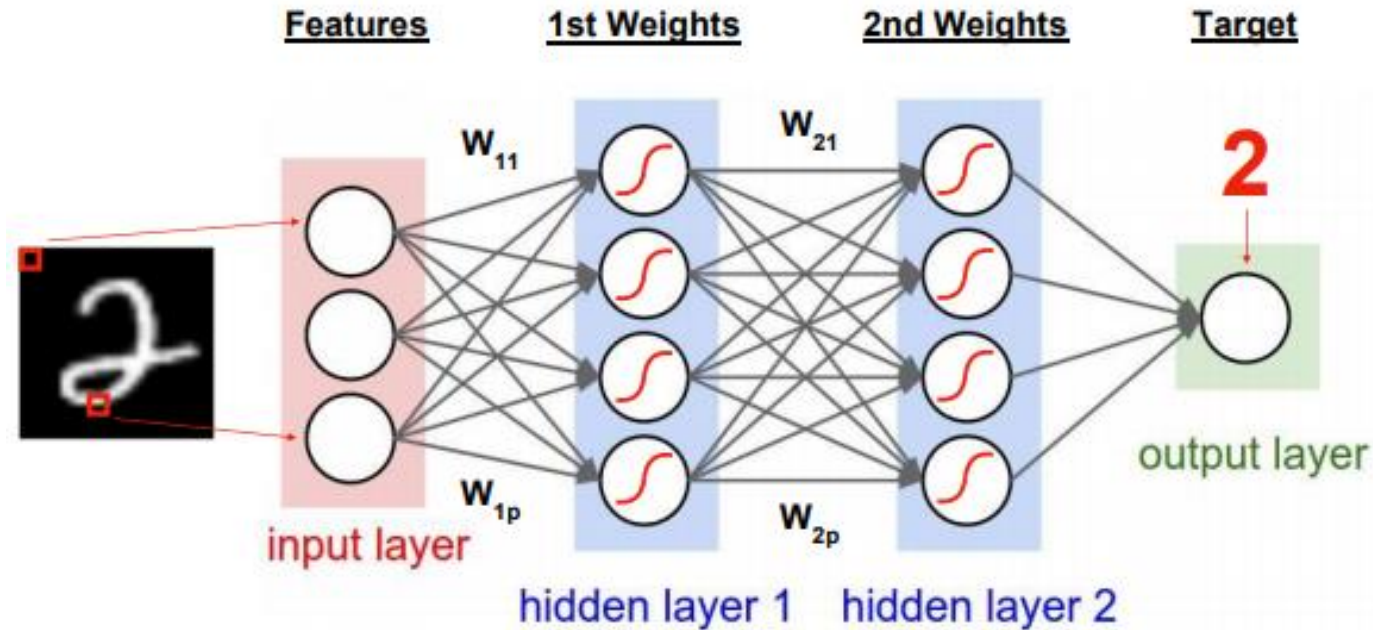


# Notebook

Visitar Google Colab



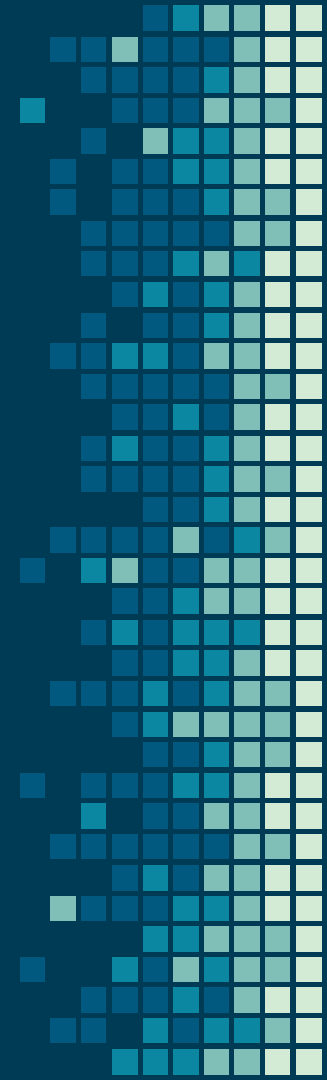
# Multilayer Perceptron





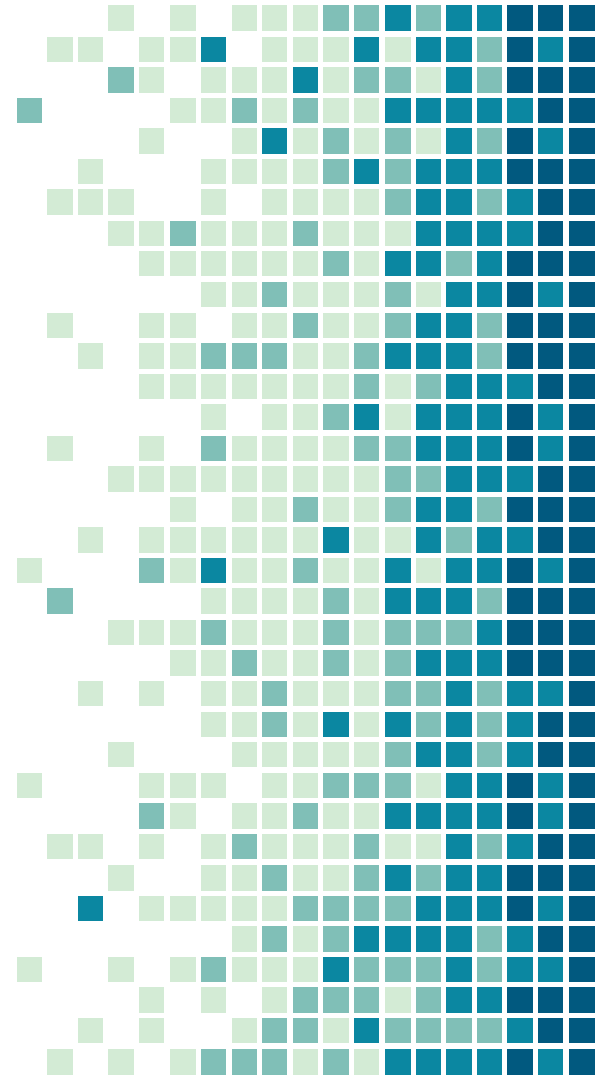
# Notebook

Visitar Google Colab

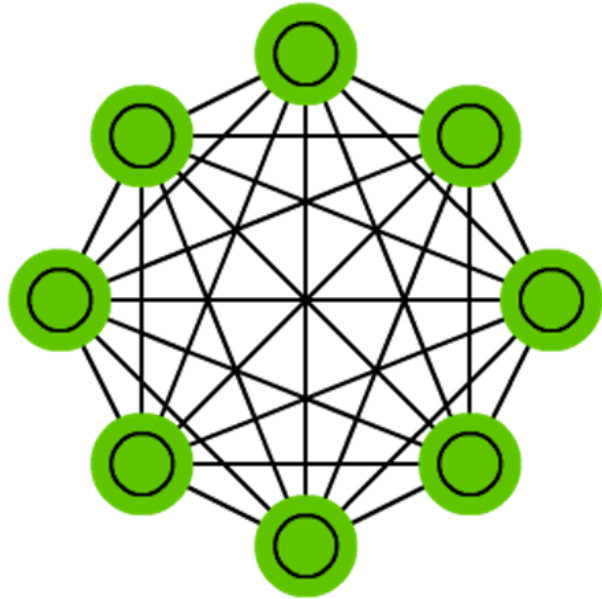


# Tipos de Redes Neuronales

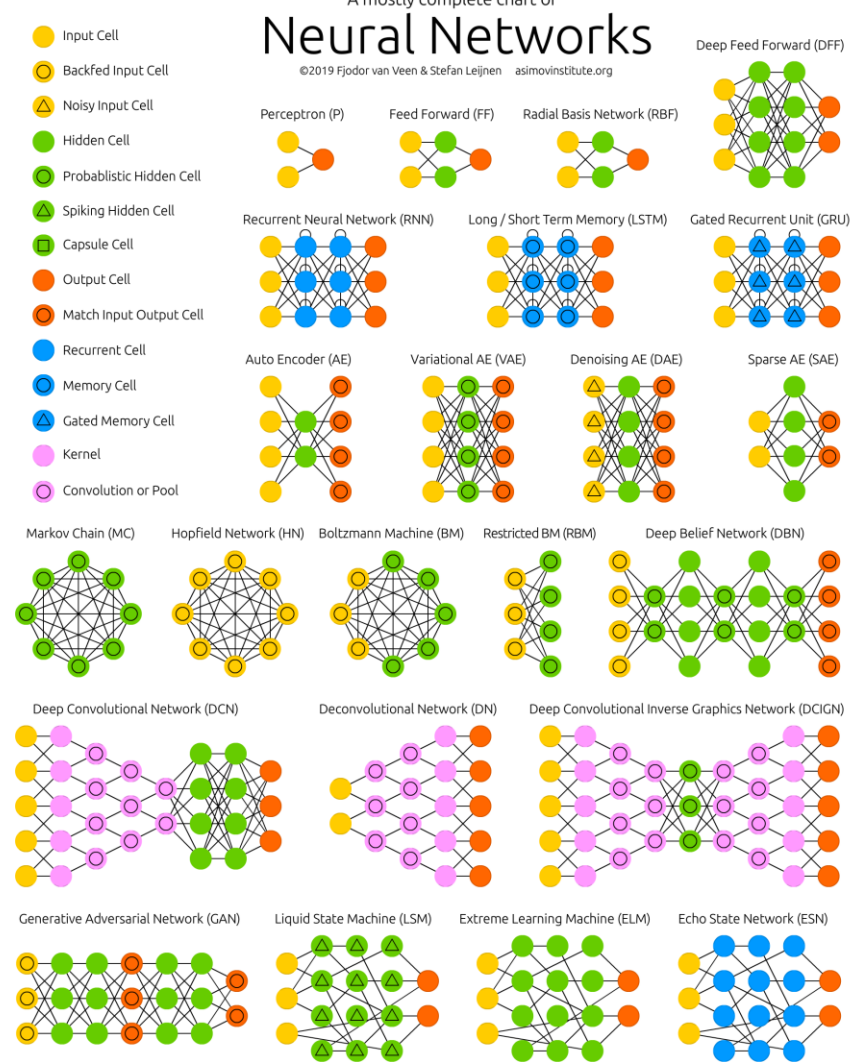
Realmente, algunos de ellos...



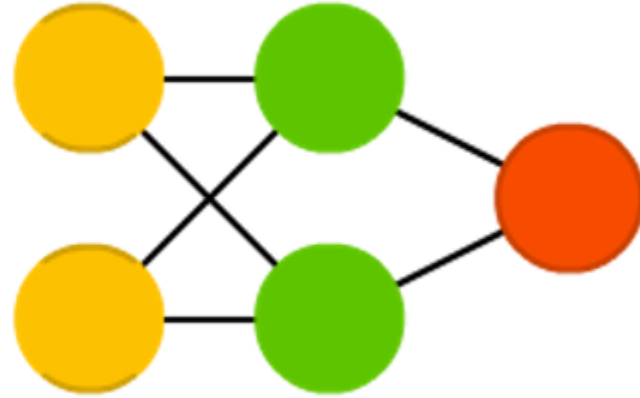
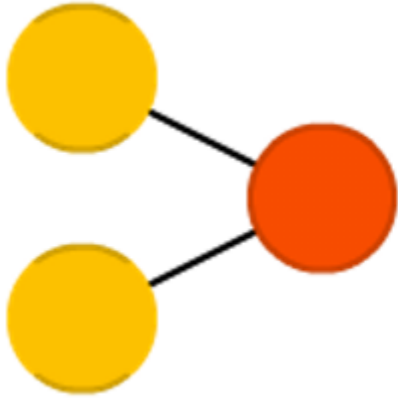
Existen muchas...



Neural Network Zoo

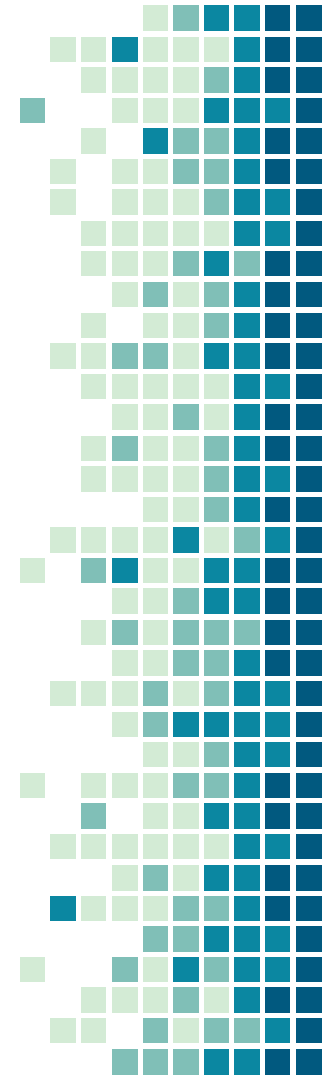


# Fully Connected Neural Networks



```
model.add(Dense(32, input_shape=(784,)))
```





# Convolution Filter

I(0,0)	I(1,0)	I(2,0)	I(3,0)	I(4,0)	I(5,0)	I(6,0)
I(0,1)	I(1,1)	I(2,1)	I(3,1)	I(4,1)	I(5,1)	I(6,1)
I(0,2)	I(1,2)	I(2,2)	I(3,2)	I(4,2)	I(5,2)	I(6,2)
I(0,3)	I(1,3)	I(2,3)	I(3,3)	I(4,3)	I(5,3)	I(6,3)
I(0,4)	I(1,4)	I(2,4)	I(3,4)	I(4,4)	I(5,4)	I(6,4)
I(0,5)	I(1,5)	I(2,5)	I(3,5)	I(4,5)	I(5,5)	I(6,5)
I(0,6)	I(1,6)	I(2,6)	I(3,6)	I(4,6)	I(5,6)	I(6,6)

Input image

×

H(0,0)	H(1,0)	H(2,0)
H(0,1)	H(1,1)	H(2,1)
H(0,2)	H(1,2)	H(2,2)

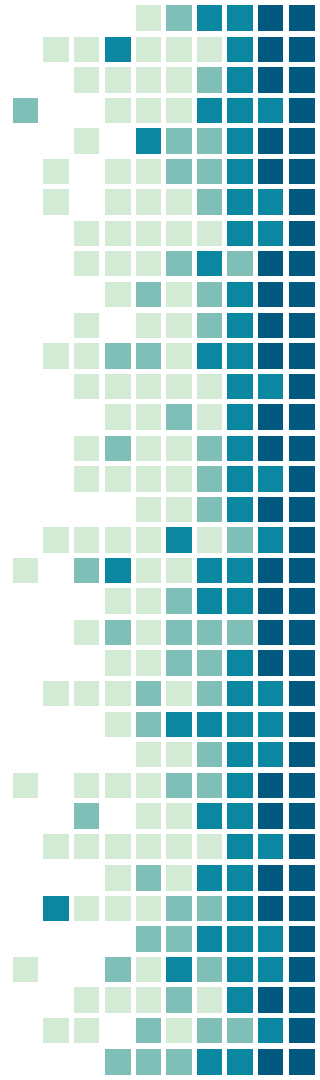
Filter

=

O(0,0)				

Output image

Stride = 1  
Pad = 0





# Ejemplo

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1																									
2		<u>Input</u>								<u>Kernel</u>						<u>Intermediate Output</u>									
3																									
4		1	0	1	0	2																			
5		1	1	3	2	1				0	1	0				7	5	3							
6		1	1	0	1	1				0	0	2				4	7	5							
7		2	3	2	1	3				0	1	0				7	2	8							
8		0	2	0	1	0																			
9																									
10		1	0	0	1	0																			
11		2	0	1	2	0				2	1	0				5	3	10							
12		3	1	1	3	0				0	0	0				13	1	13							
13		0	3	0	3	2				0	3	0				7	12	11							
14		1	0	3	2	1																			
15																									
16		2	0	1	2	1																			
17		3	3	1	3	2				1	0	0				7	5	2							
18		2	1	1	1	0				1	0	0				11	8	2							
19		3	1	3	2	0				0	0	2				9	4	6							
20		1	1	2	1	1																			
21																									

Output

19 13 15  
28 16 20  
23 18 25

# Filtros

Operation

Filter

Convolved  
Image

Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

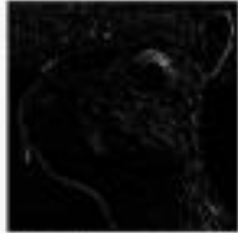


$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$



Edge detection

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Box blur  
(normalized)

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

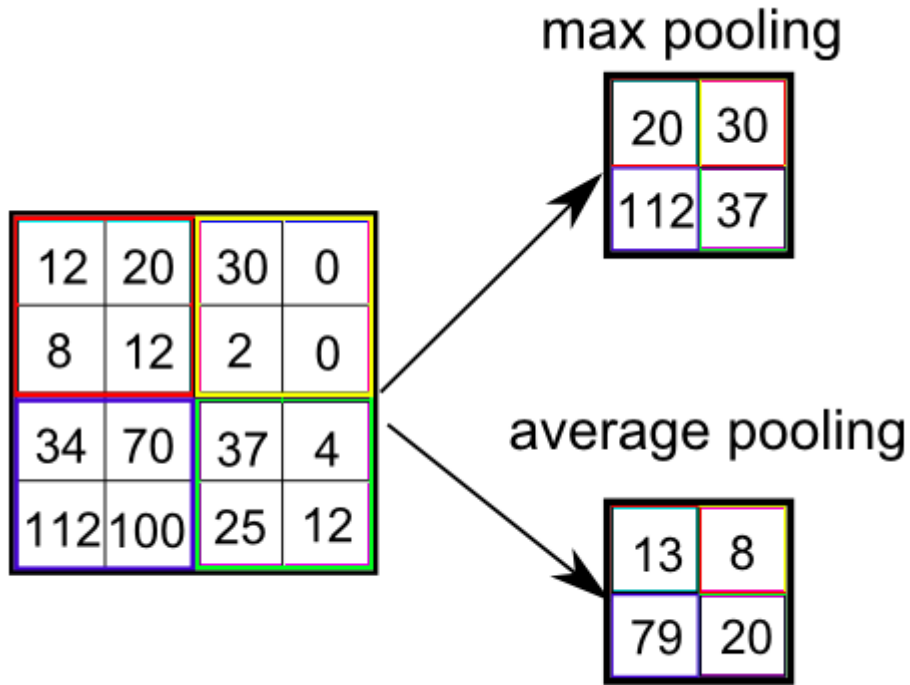


Gaussian blur  
(approximation)

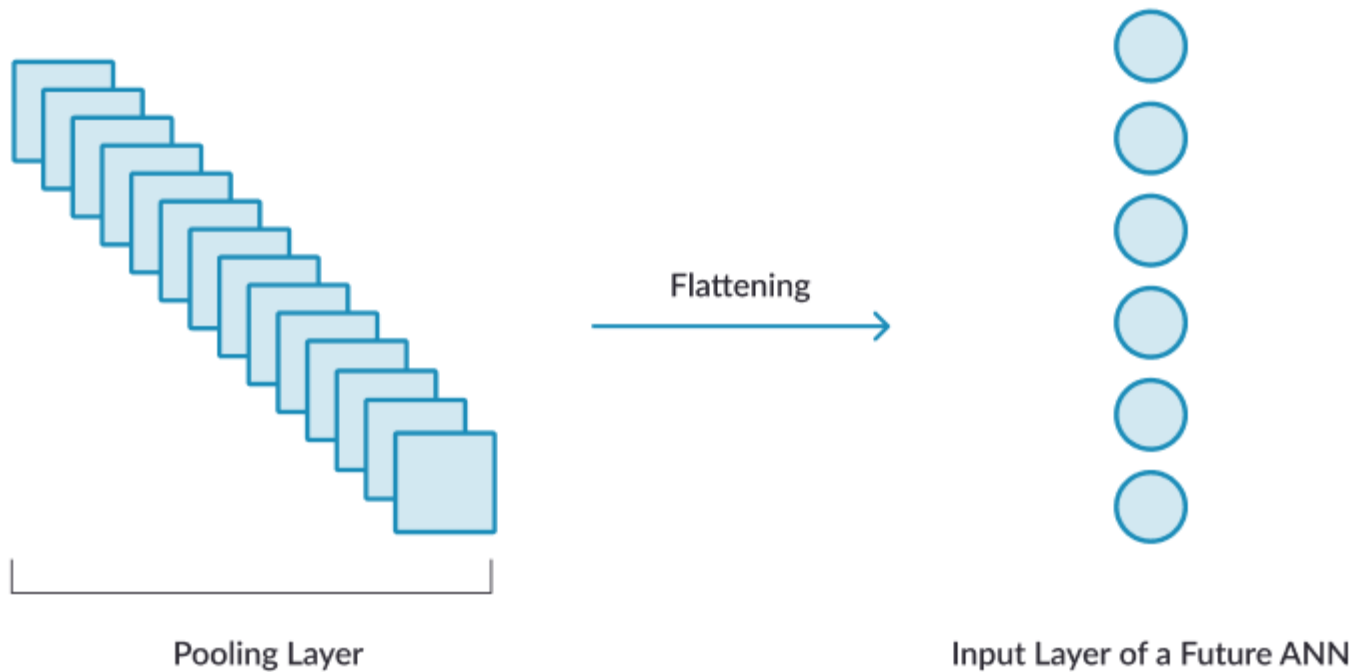
$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



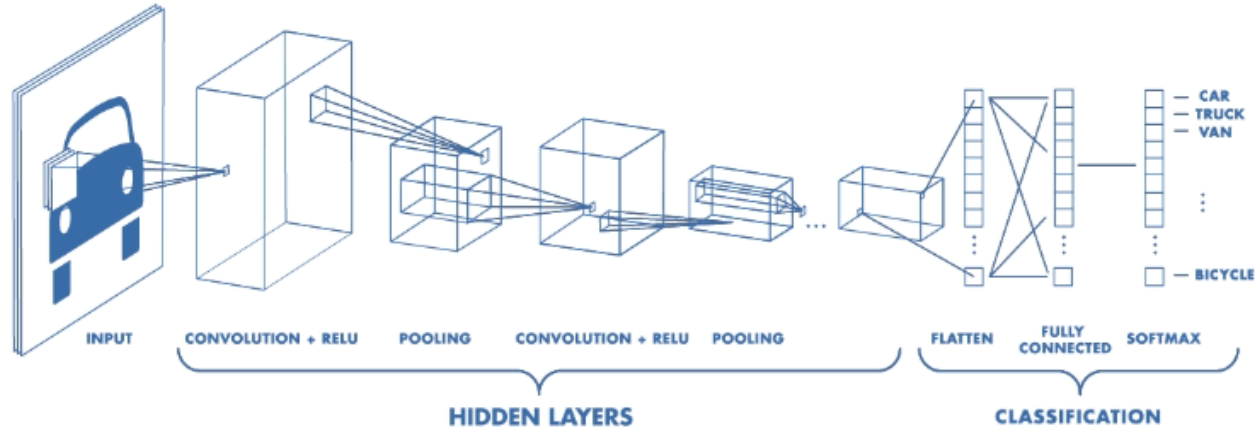
# Pooling



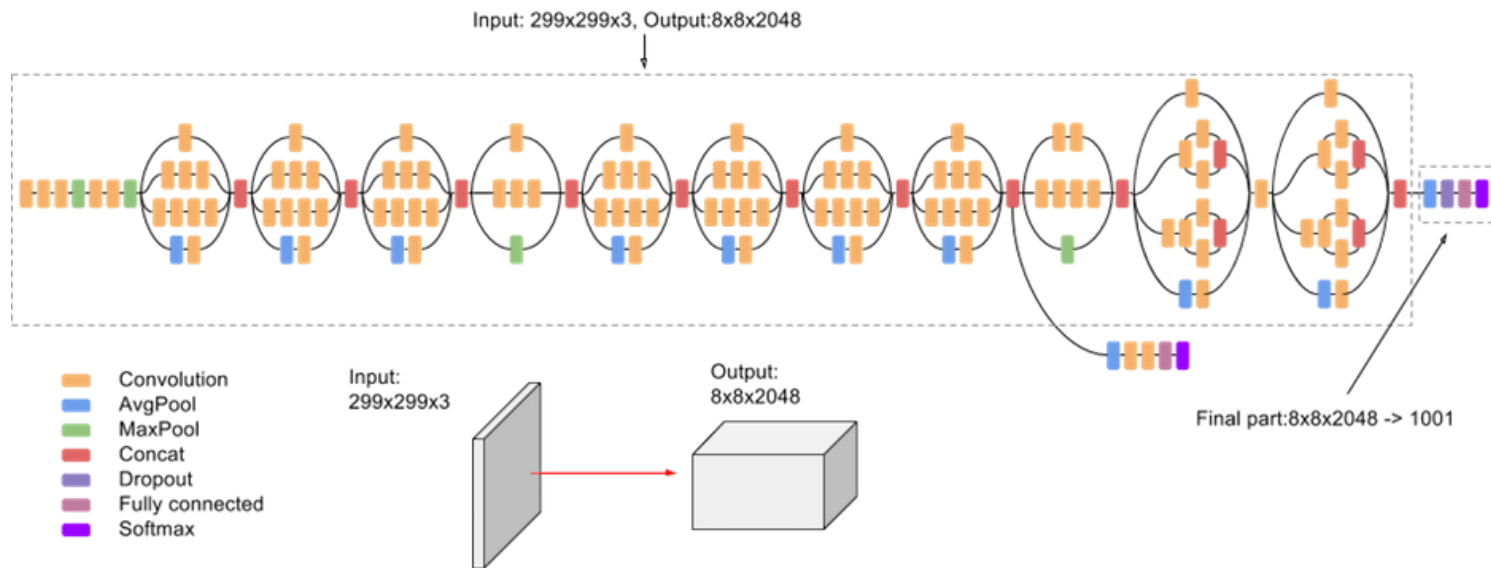
# Flatten



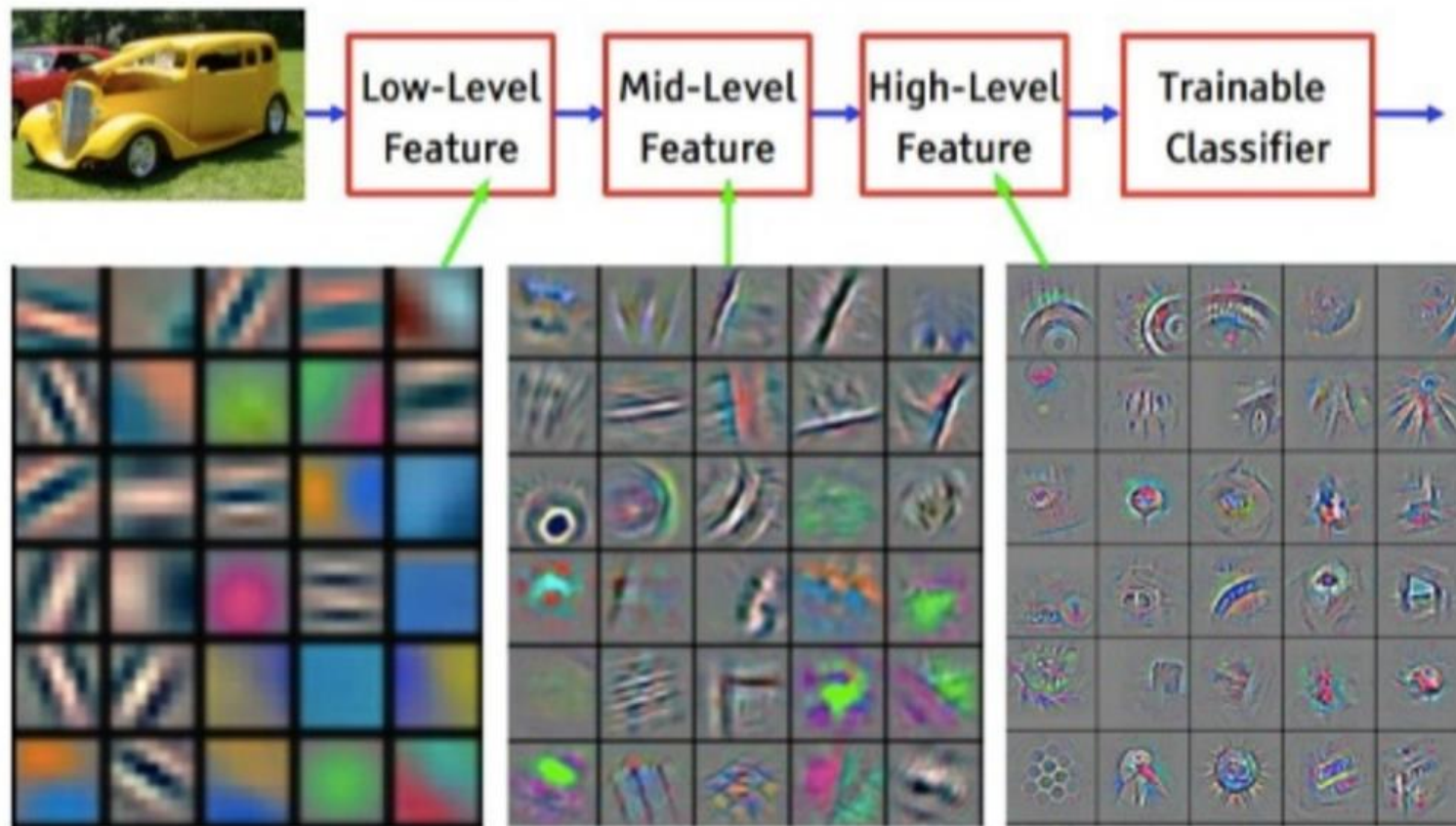
# Convolutional Neural Network



# Inception v3

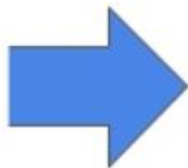


# Convolutional Neural Network

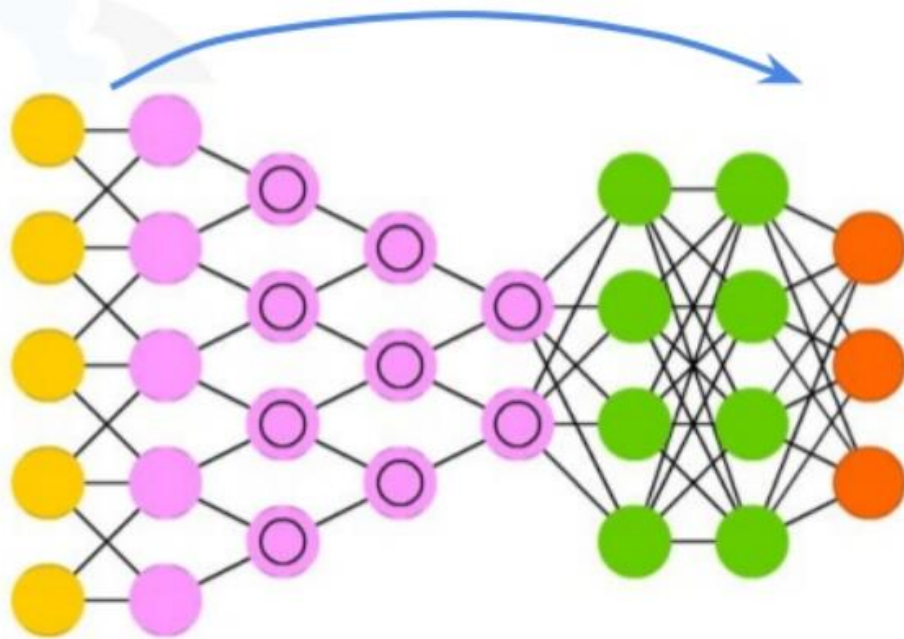


# Convolutional Neural Networks

training image  
label: pants



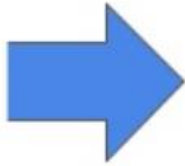
Forward pass



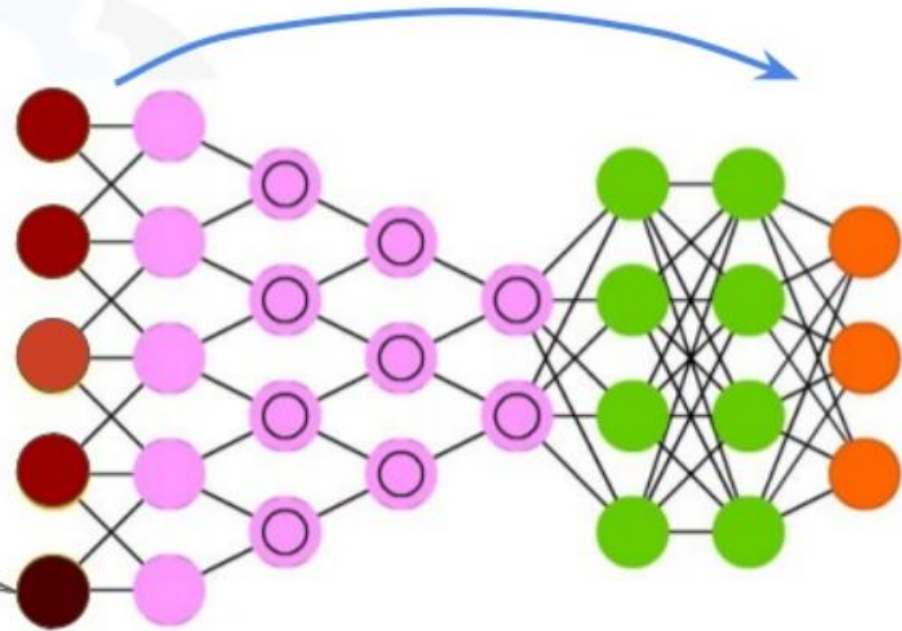


# Convolutional Neural Networks

training image  
label: pants



Forward pass

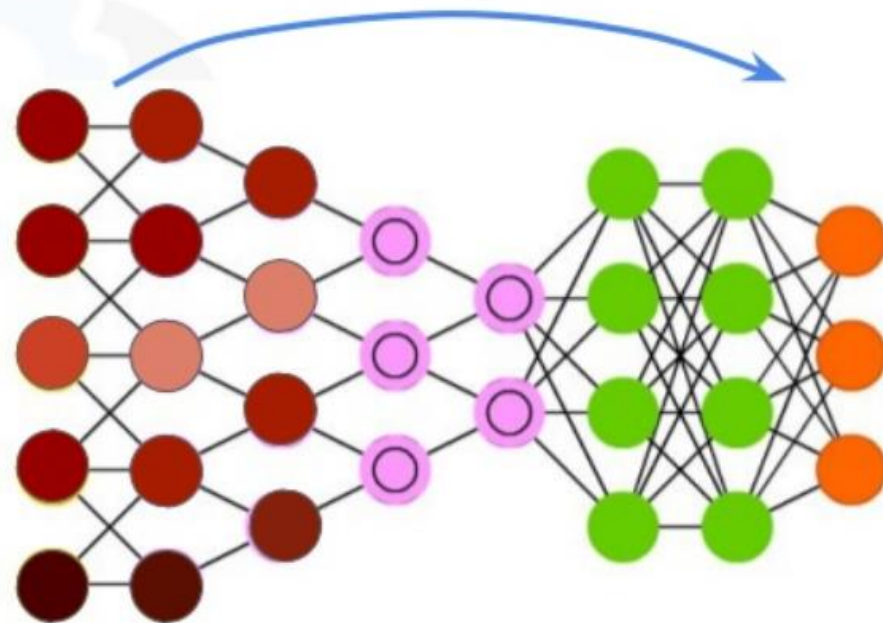


# Convolutional Neural Networks

training image  
label: pants

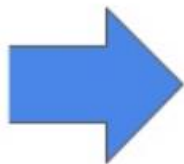


Forward pass

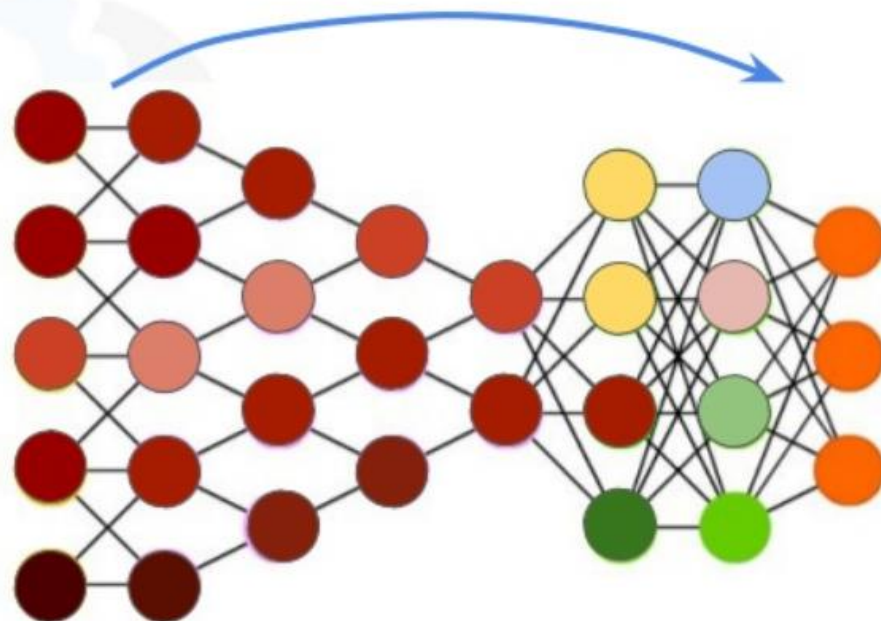


# Convolutional Neural Networks

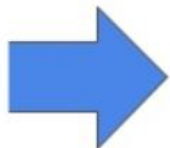
training image  
label: pants



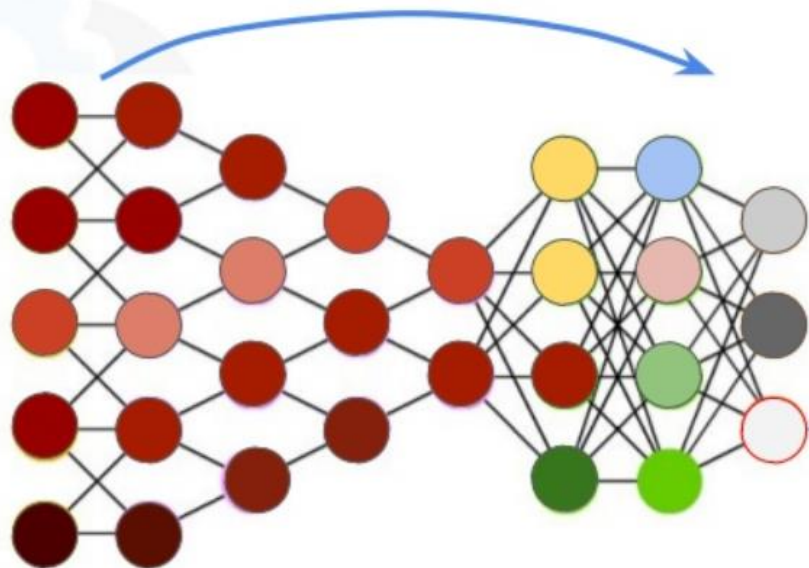
Forward pass



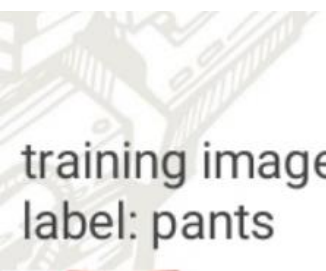
training image  
label: pants



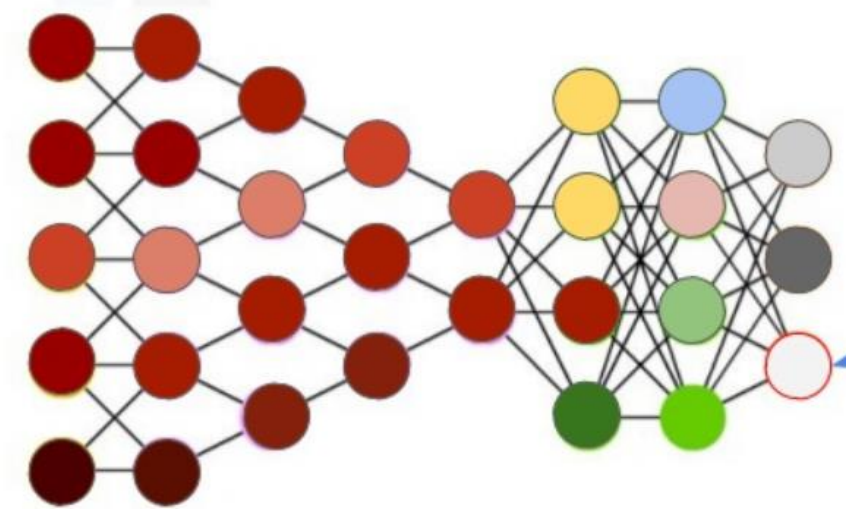
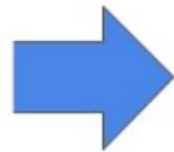
Forward pass



predicted  
label=  
*cap*



training image  
label: pants

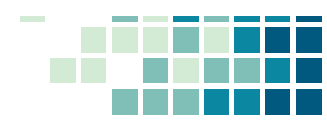


predicted  
label=  
*cap*

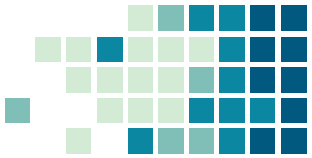
Backpropagation + Gradient descent



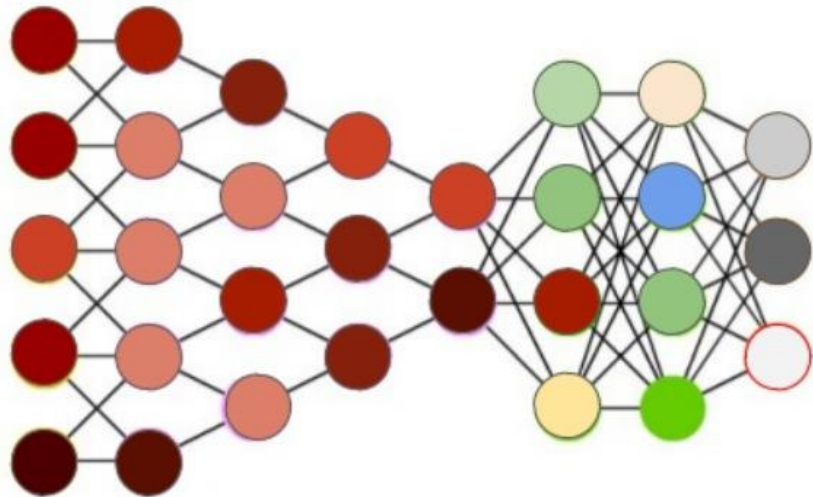
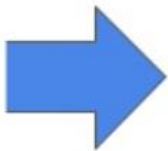
BRAINCREATORS







training image  
label: pants

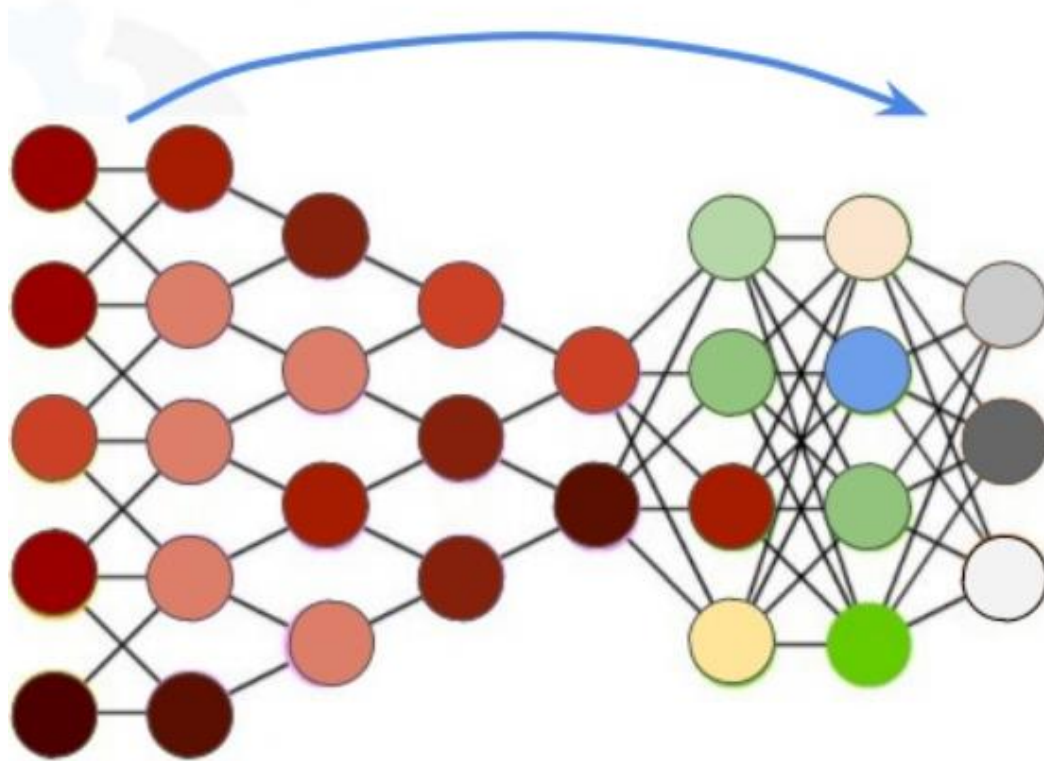


predicted  
label=  
*pants*

Backpropagation + Gradient descent  
= update weights "towards target" **BRAINCREATORS**



Forward pass



Backpropagation + Gradient descent



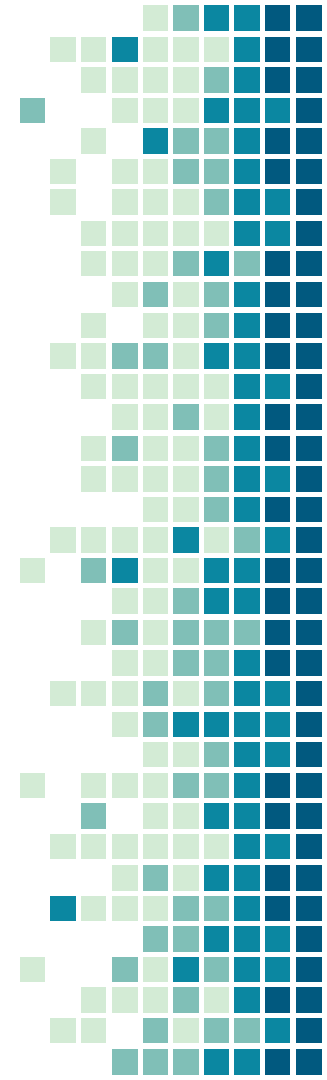
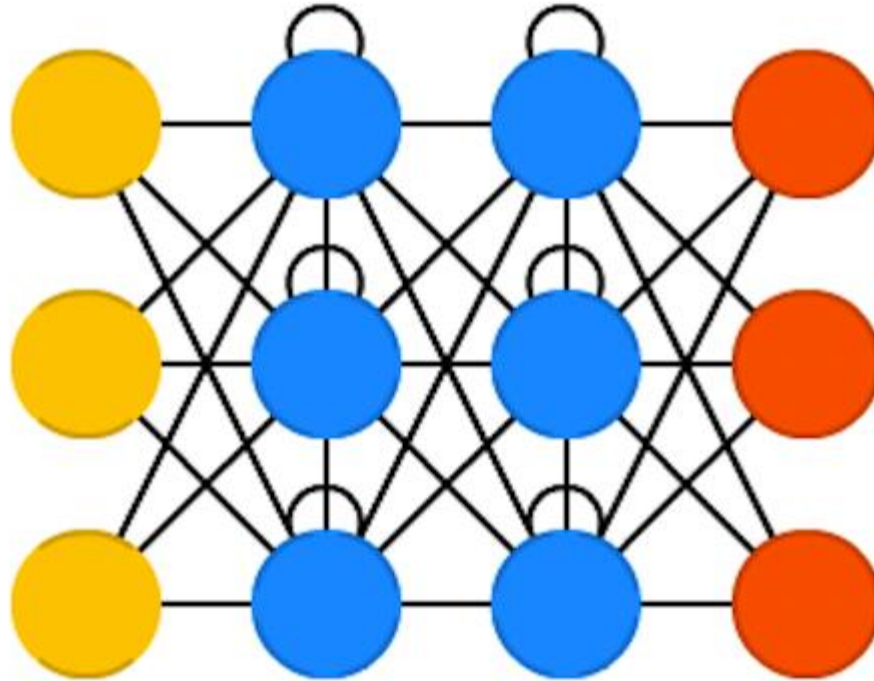
**BRAIN**

# Recurrent Neural Network

Para mapear  
predicciones en  
secuencias.

Aplicaciones:

subtítulos de imágenes,  
análisis de opinión, respuesta  
a preguntas, reconocimiento  
de voz, series de tiempo,  
generación de música,  
traducción automática,  
automóviles autónomos, etc.

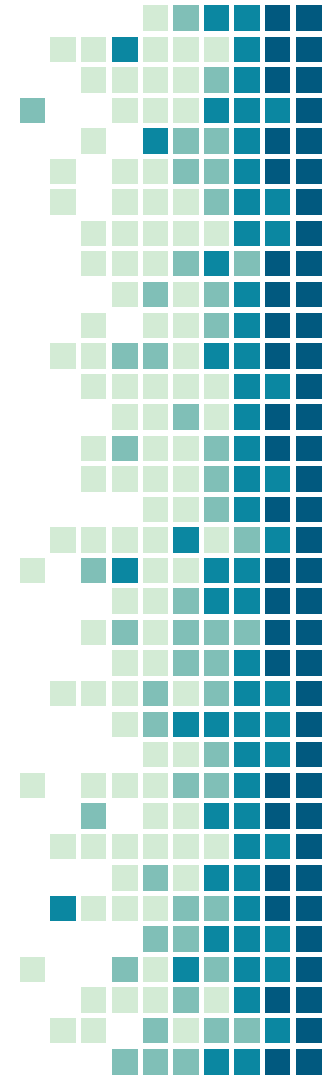
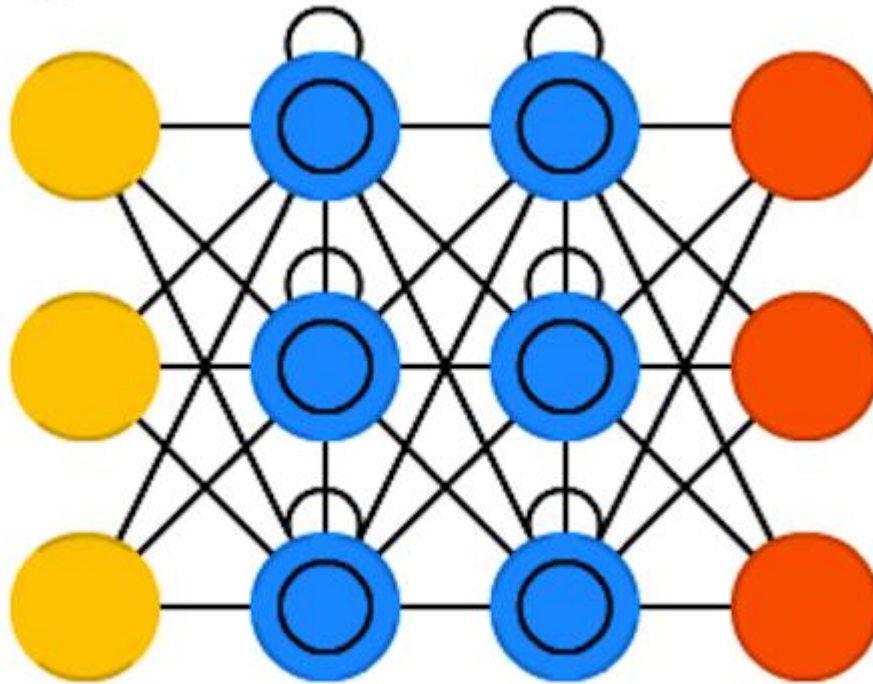




# LSTM

Para mapear  
predicciones en  
secuencias.

Intenta combatir el problema  
de gradiente de desaparición /  
explosión mediante la  
introducción de puertas y una  
celda de memoria  
explícitamente definida



# Proyecto en CNN:

- Clasificador de Imágenes



# Tutoriales en CNN:

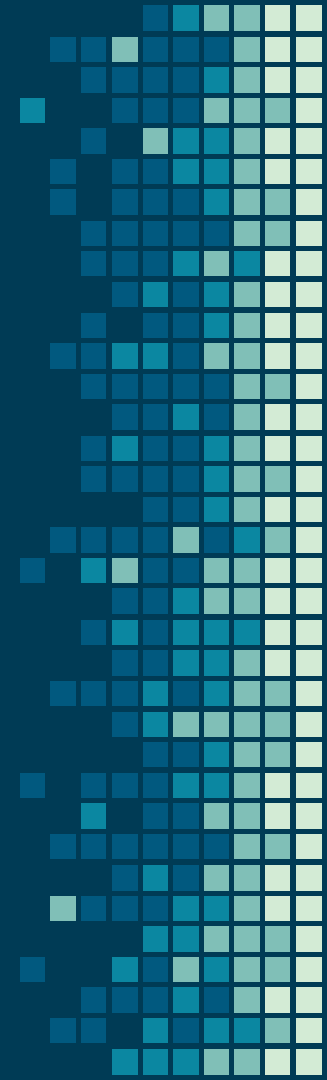
- [CNN in Keras, Towards Data Science](#)
- [Keras CNN Tutorial](#)





# Notebook

Visitar Google Colab



# Referencias

[https://www.slideshare.net/databricks/introduction-to-neural-networks-122033415?from\\_action=save](https://www.slideshare.net/databricks/introduction-to-neural-networks-122033415?from_action=save)

<https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf>

<https://www.slideshare.net/JohnRamey2/introduction-to-keras>

[http://sct.uab.cat/estadistica/sites/sct.uab.cat.estadistica/files/sessio\\_sea.pdf](http://sct.uab.cat/estadistica/sites/sct.uab.cat.estadistica/files/sessio_sea.pdf)

<https://www.slideshare.net/braincreators/introduction-to-deep-neural-networks>

<https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>

[https://medium.com/@zhang\\_yang/number-of-parameters-in-dense-and-convolutional-neural-networks-34b54c2ec349](https://medium.com/@zhang_yang/number-of-parameters-in-dense-and-convolutional-neural-networks-34b54c2ec349)

