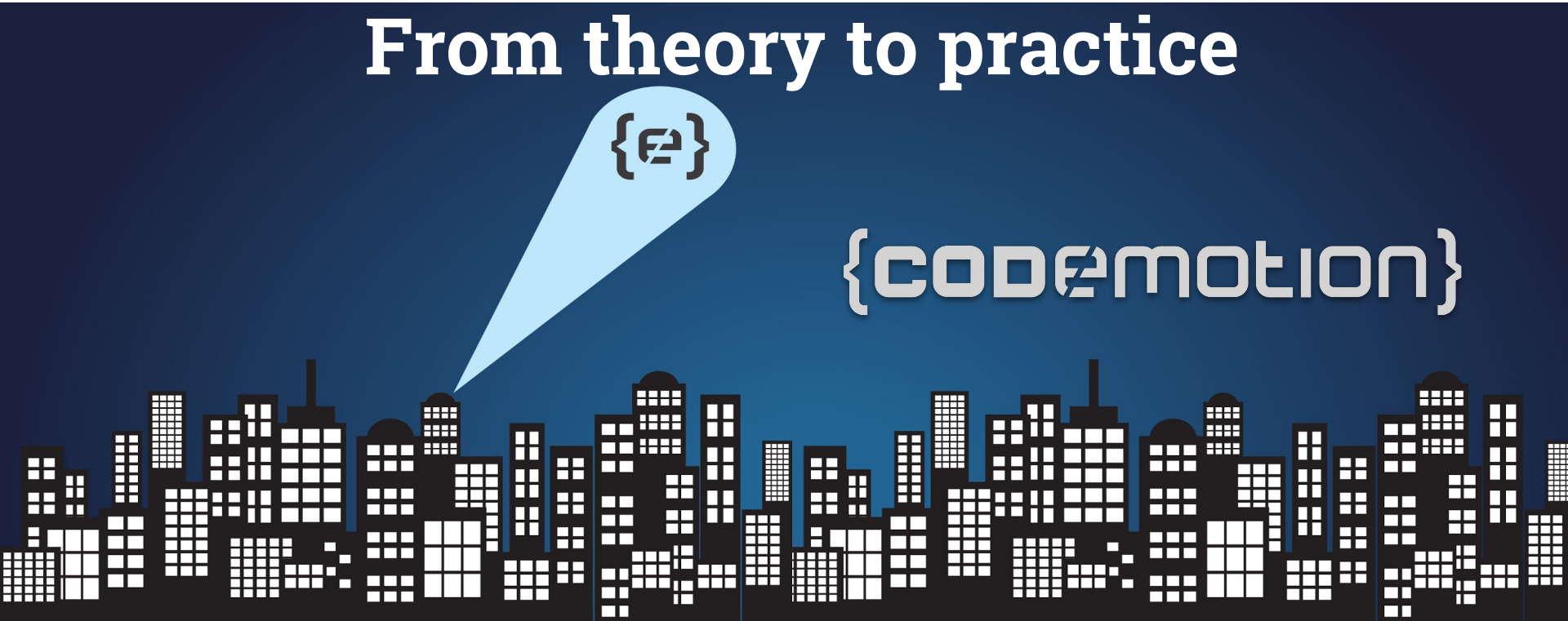
TensorFlow 101

From theory to practice



¿Quiénes somos?

Doctor en Ciencia y Tecnología Informática

Inteligencia artificial, Robótica, Espacio, Videojuegos y Aprendizaje Automático

@moisipm

Moisés Martínez

Estudiante de Doctorado en Ciencia y Tecnología Informática

Inteligencia artificial, Robótica y Planificación Automática Multi Agente

@sailormercury91

Nerea Luis

1.

Deep Learning

¿Qué significa? ¿Qué es? ¿Por qué todos hablan de *deep learning*?

¿Qué significa Machine Learning?

Conjunto de modelos y algoritmos basados en funciones matemáticas que nos permiten identificar **patrones** con el fin de **predecir**

Tipos de aprendizaje

Aprendizaje supervisado
con intervención humana

» ¿Cómo clasifico fotos de gatos y perros?

Aprendizaje no supervisado
sin intervención humana

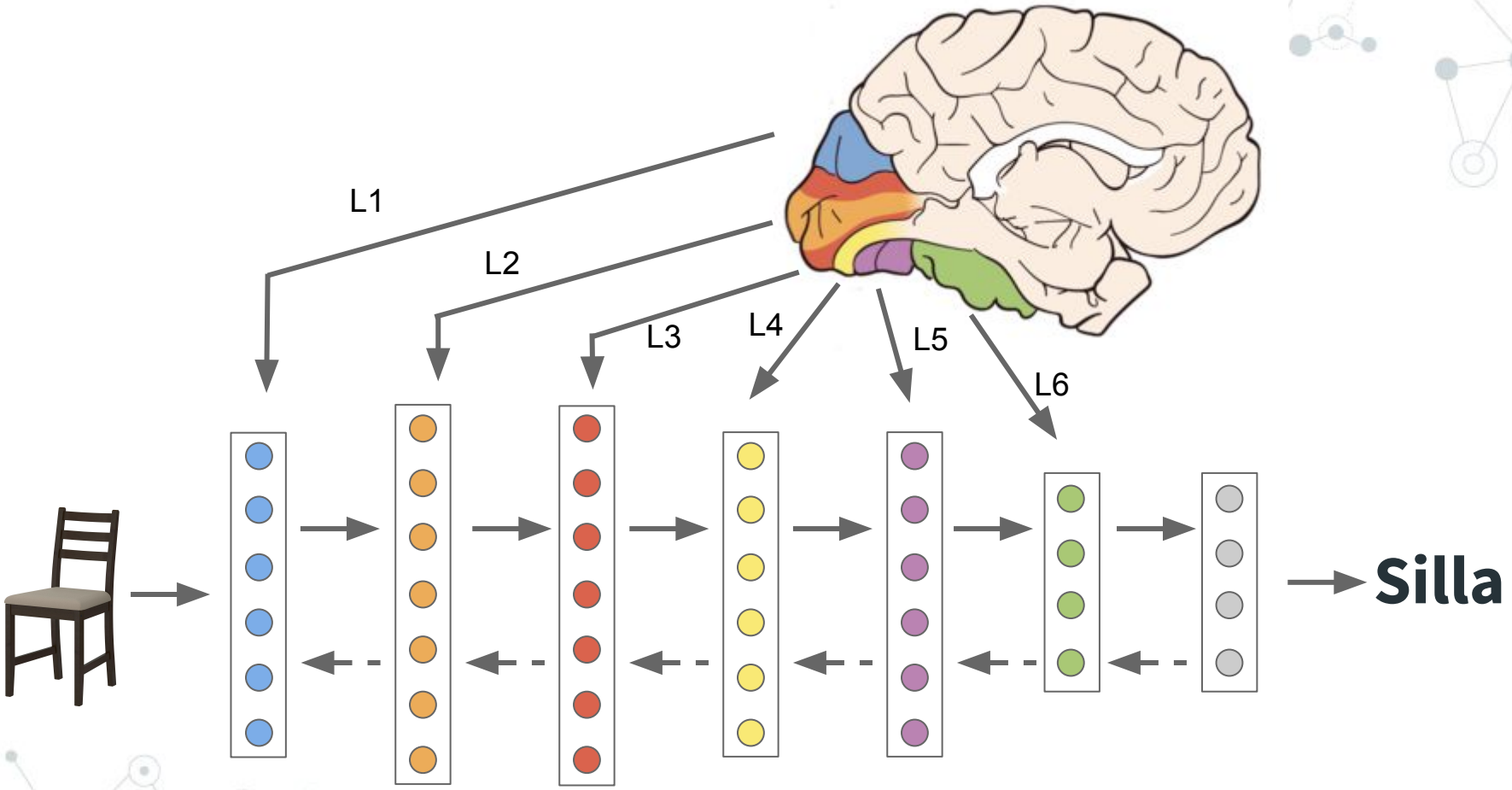
» ¿Cómo clasifico tipos de perros si no sé qué los distinguen?

Aprendizaje por refuerzo
sin intervención humana

» ¿Cómo entreno a mi perro? ¿Robótica?

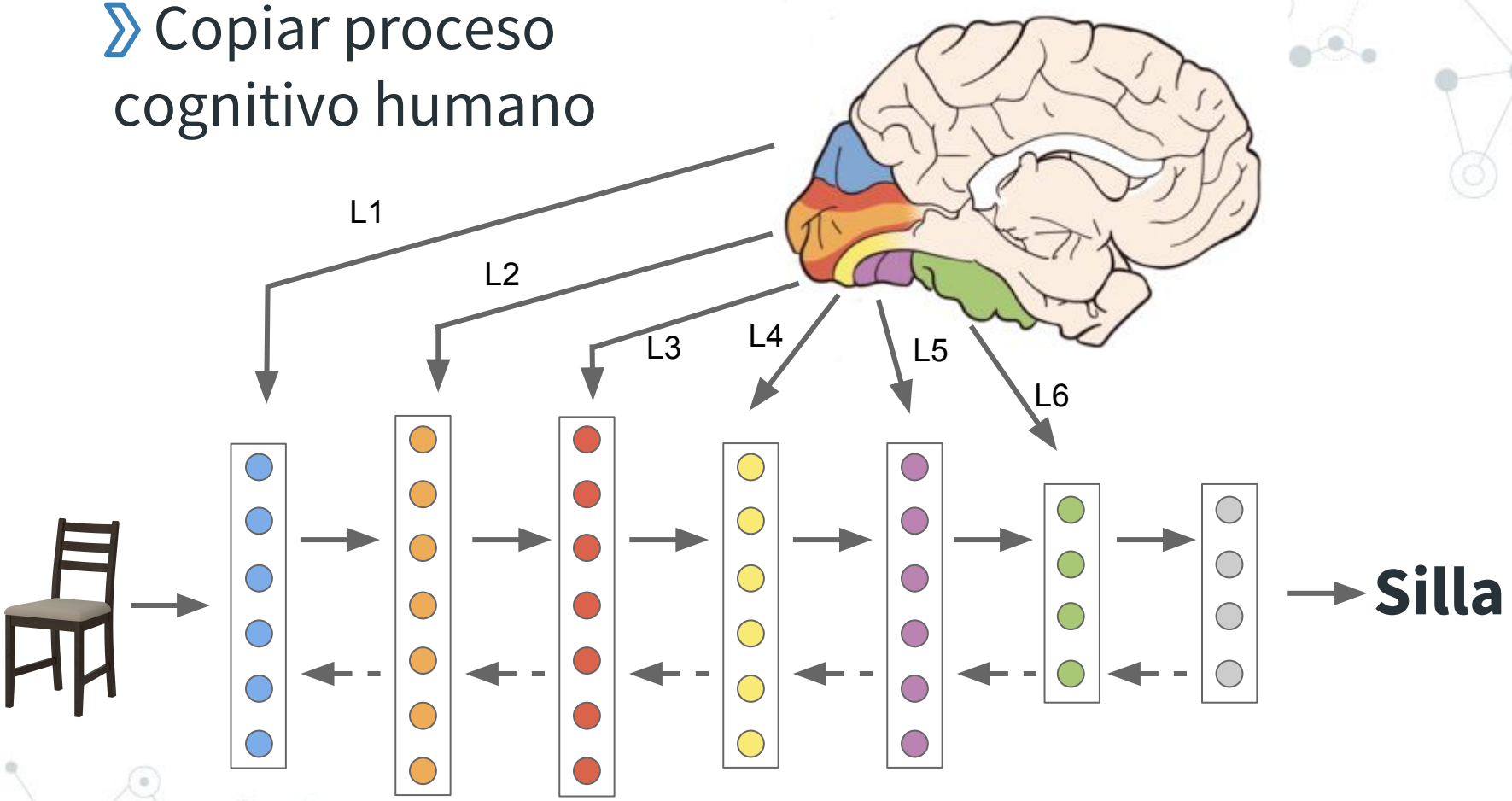

Deep Learning
Aprendizaje no supervisado
Aprendizaje por refuerzo

Deep Learning



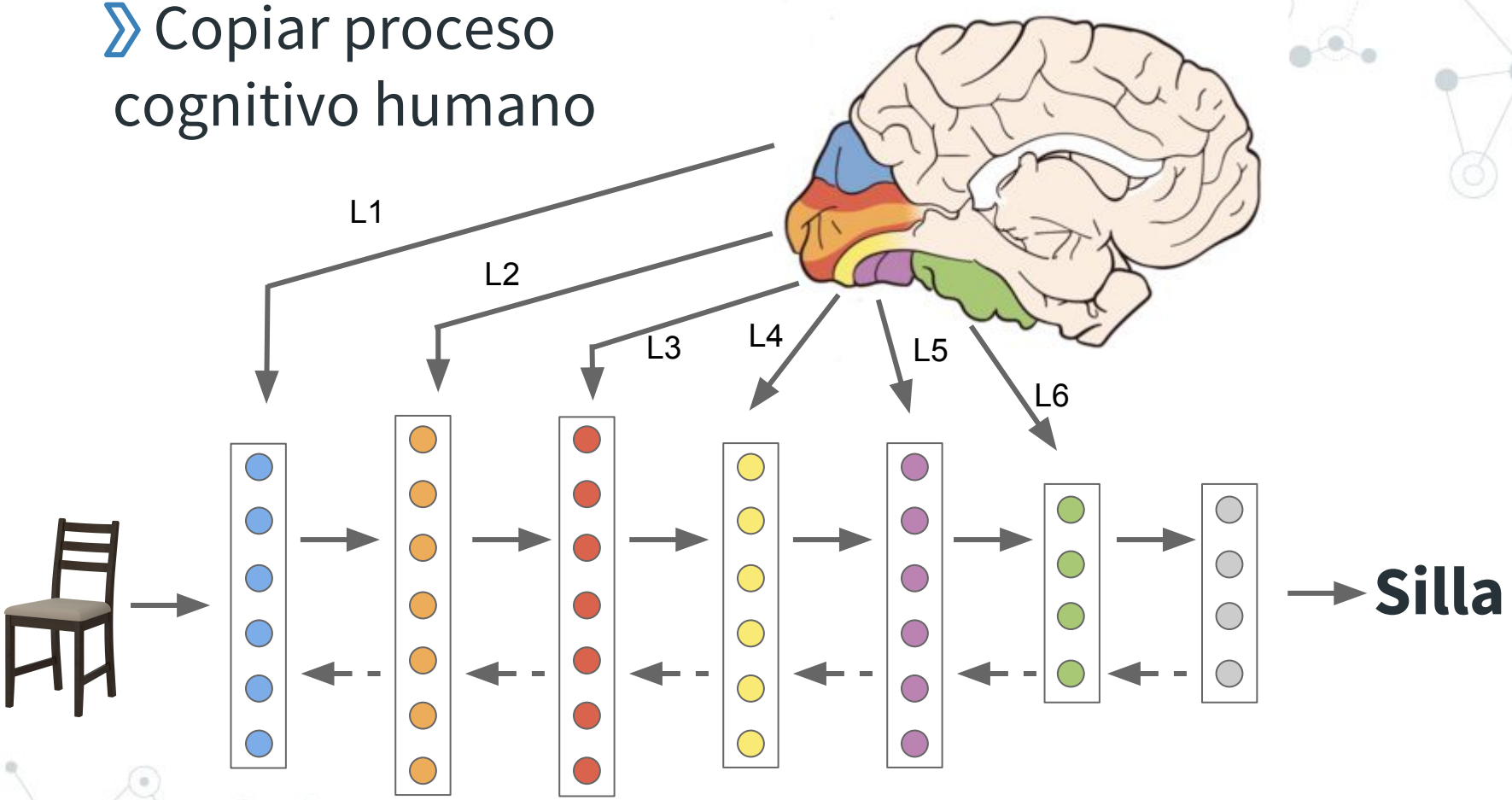
Deep Learning

- Copiar proceso cognitivo humano



Deep Learning

- Copiar proceso cognitivo humano



Colección de funciones matemáticas **sencillas**

Entendiendo una red de neuronas

> Utilizamos elementos llamados **‘neuronas’**

> Conectadas de forma múltiple entre sí formando una **‘red’**

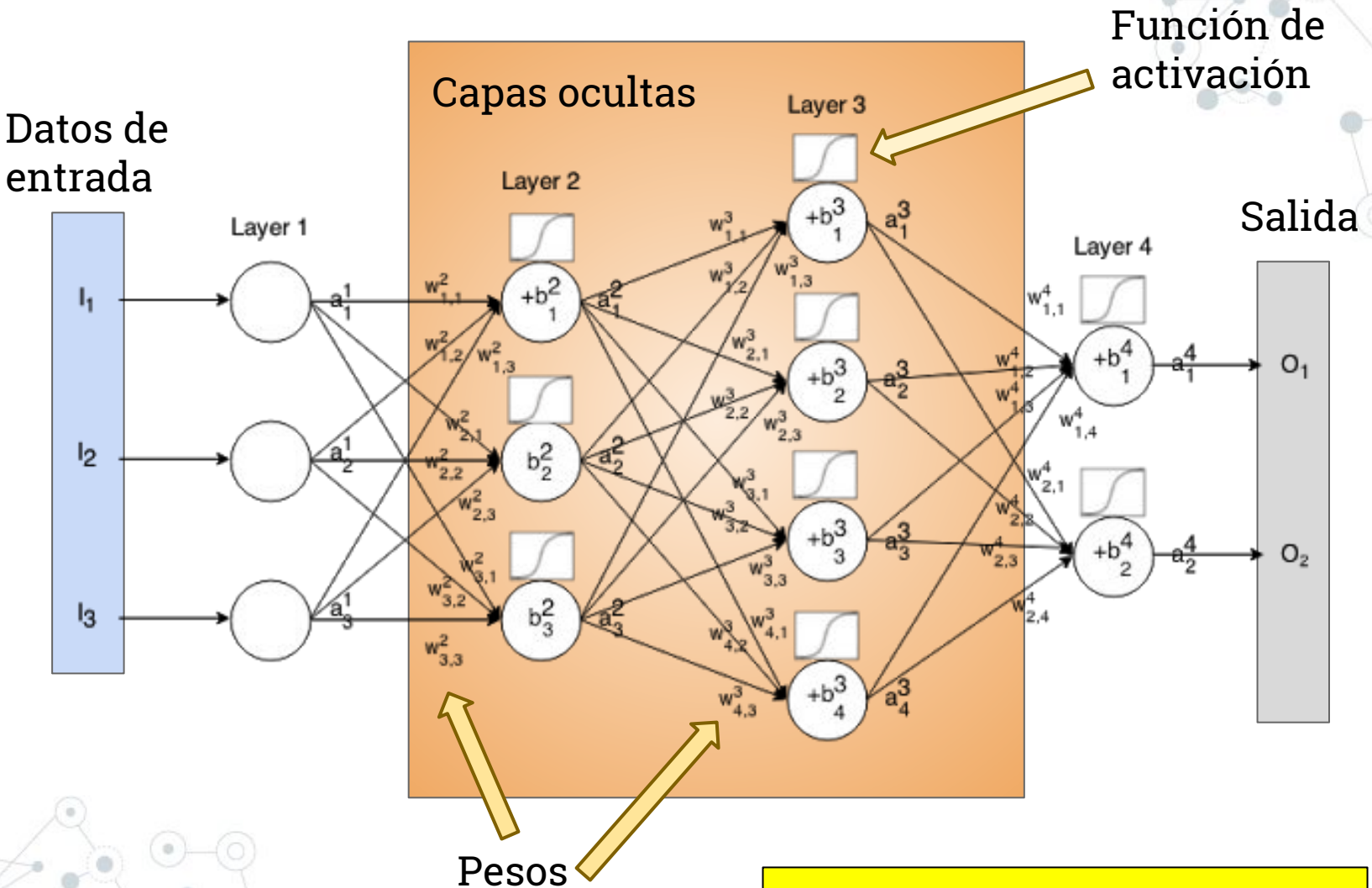
> Damos importancia a los **‘datos de entrada’**

> Escogemos **pesos, función de activación**

> Interpretamos la **salida:**

- Semántica de los datos
- Características Comunes

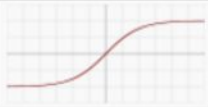
¡Y ahora todo junto que empezamos!



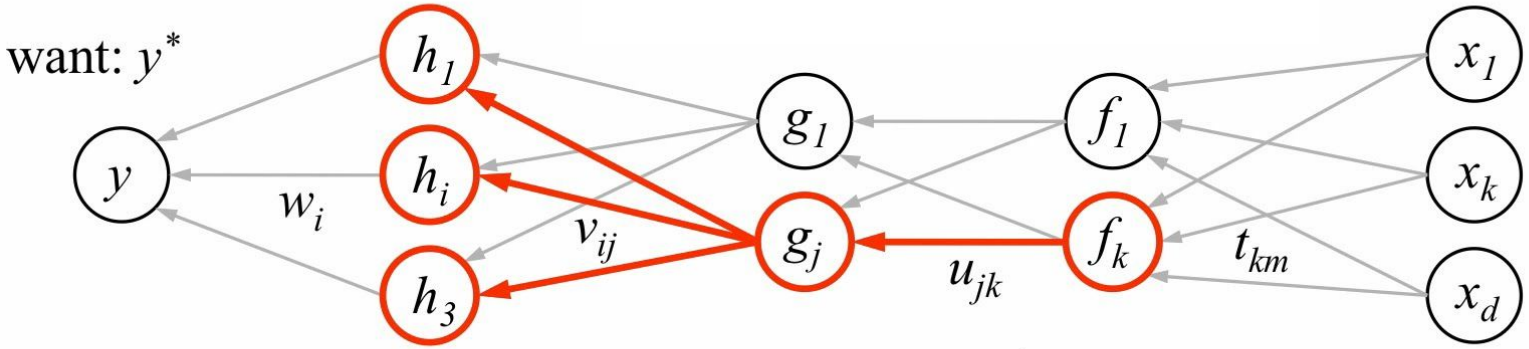
Gradiente, función de coste, umbral….

Ejemplos de funciones de activación

ReLU

Name	Plot	Equation
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$

El algoritmo de back-propagation



1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer 1…L
   compute g_j based on units f_k from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction y and error $(y\text{-}y^*)$
4. **back-propagate error:** for each unit g_j in each layer L…1

(a) compute error on g_j

$$\underbrace{\frac{\partial E}{\partial g_j}}_{\text{should } g_j \text{ be higher or lower?}} = \sum_i \underbrace{\sigma'(h_i) v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \underbrace{\frac{\partial E}{\partial g_j}}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{\sigma'(g_j) f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

(ii) update the weight

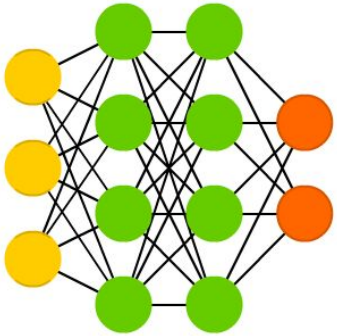
$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

WTF IS THIS SH*T PLEASE STOP NOW

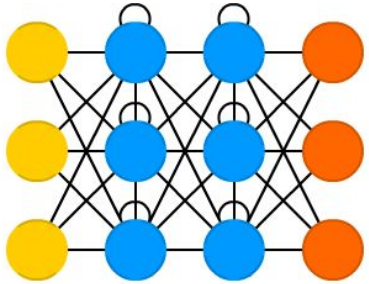
KEEP CALM

Tipos de redes de neuronas...
...para trabajar en **Deep** Learning

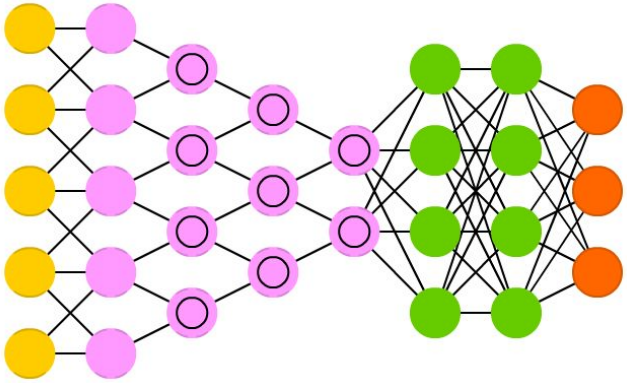
Feed Forward (FFN)



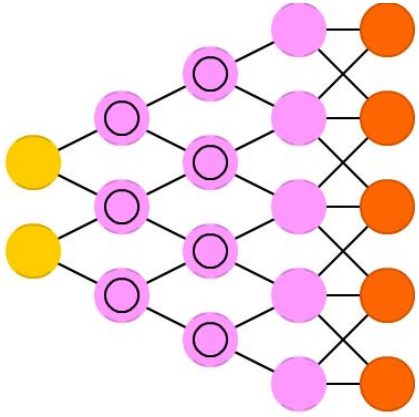
Recurrentes (RNN)



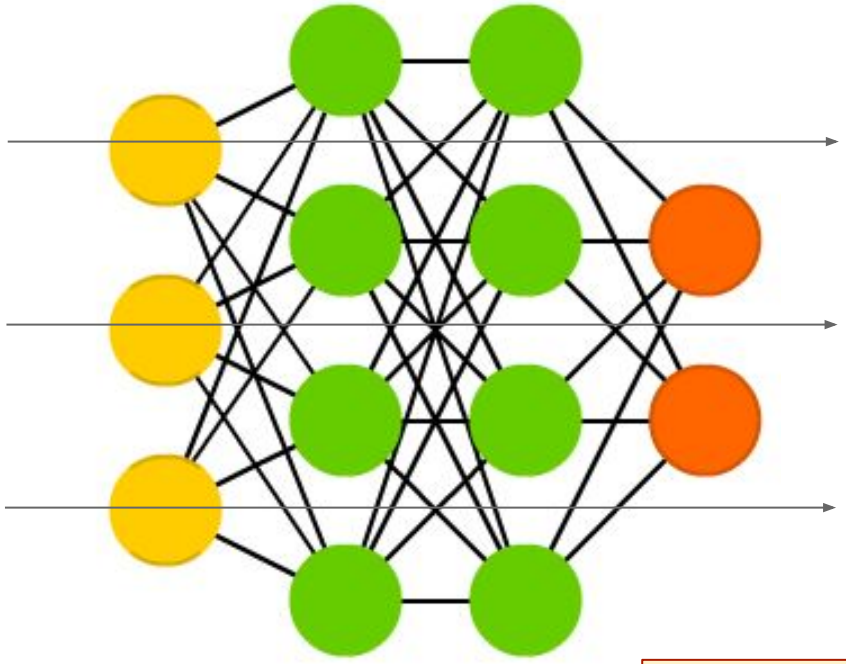
Convolucionales (CNN)



Deconvolucionales (DNN)



Feed Forward Neural Networks (FFNN)

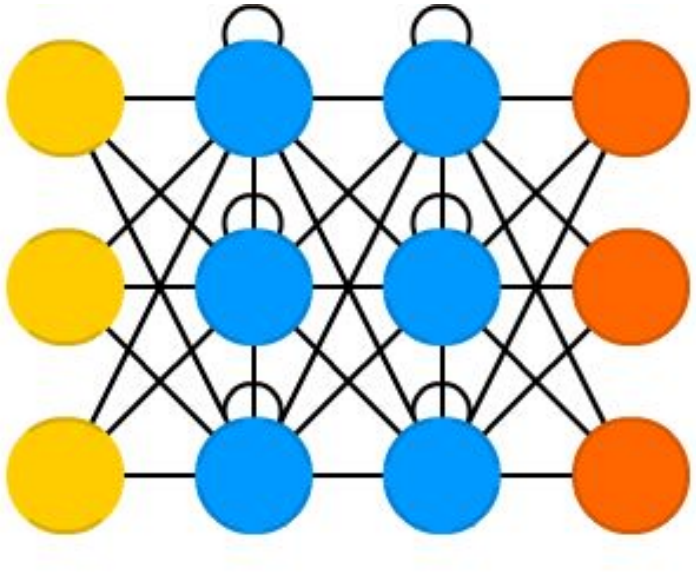


Simbología:

- Neurona de entrada
- Neurona oculta
- Neurona de salida

Todo el flujo de operaciones se realiza desde los datos de entrada hacia la salida en la misma dirección. De izquierda a derecha, 'avanzando hacia adelante'

Recurrent Neural Networks (RNN)



Simbología:

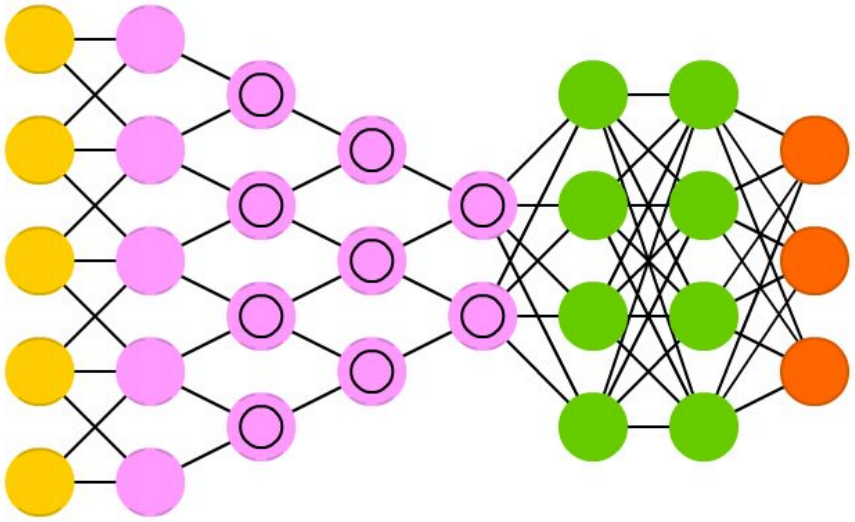
- Neurona de entrada
- Neurona oculta
- Neurona de salida

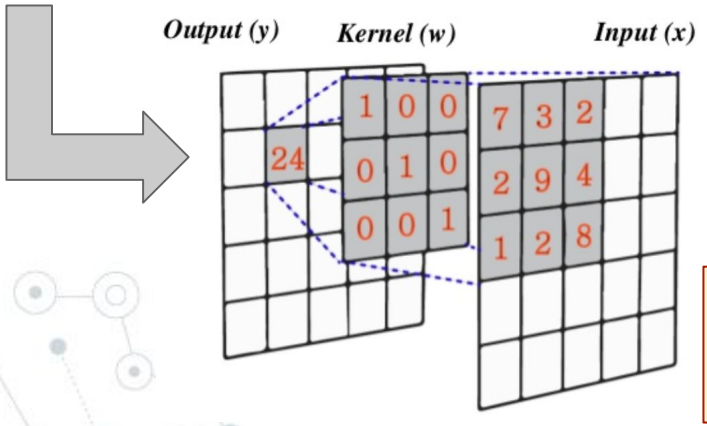
Las neuronas de las capas ocultas retienen más tiempo el entrenamiento si el peso es elevado. Esto es porque el propio output se recibe de nuevo como input.

Convolutional Neural Networks (CNN)

Matriz de mxn pesos. Equivalentes cada uno a un trozo pequeño de la imagen

IMAGEN





Simbología:

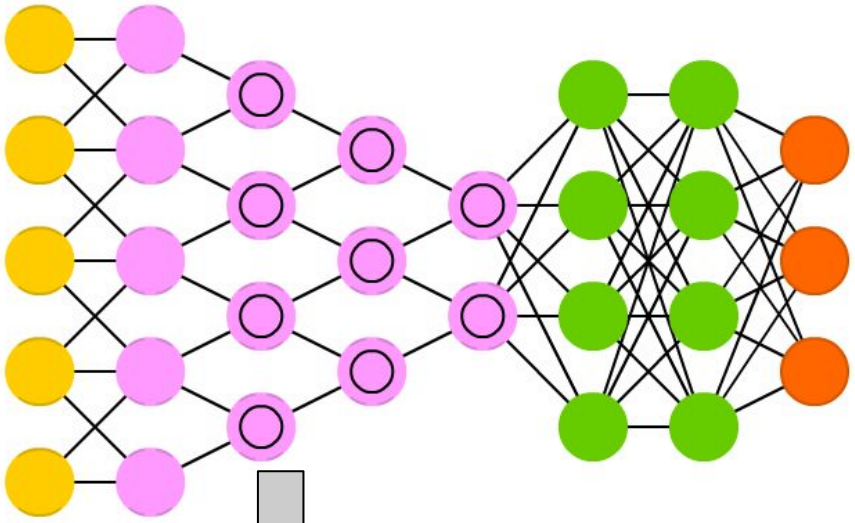
- Neurona de entrada
- Neurona Convolucional
- Neurona Pool
- Neurona Forward
- Neurona de salida

Permite submuestreo de datos de entrada.

Convolutional Neural Networks (CNN)

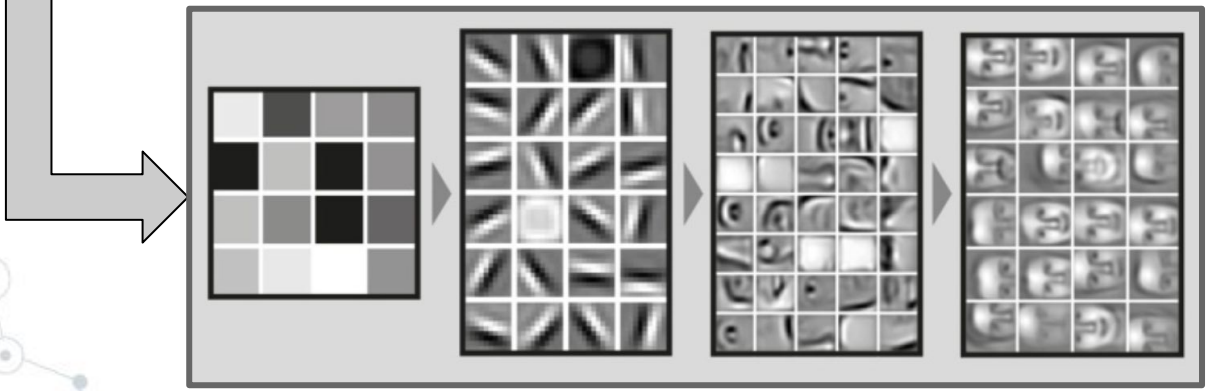
Matriz de mxn pesos. Equivalentes cada uno a un trozo pequeño de la imagen

IMAGEN

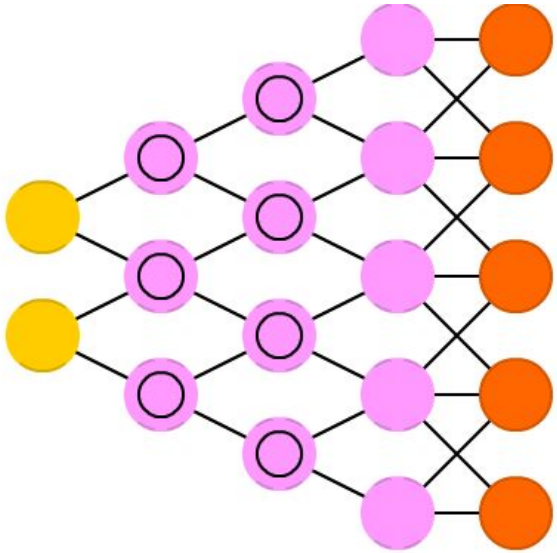


Simbología:

- Neurona de entrada
- Neurona Convolucional
- Neurona Pool
- Neurona Forward
- Neurona de salida

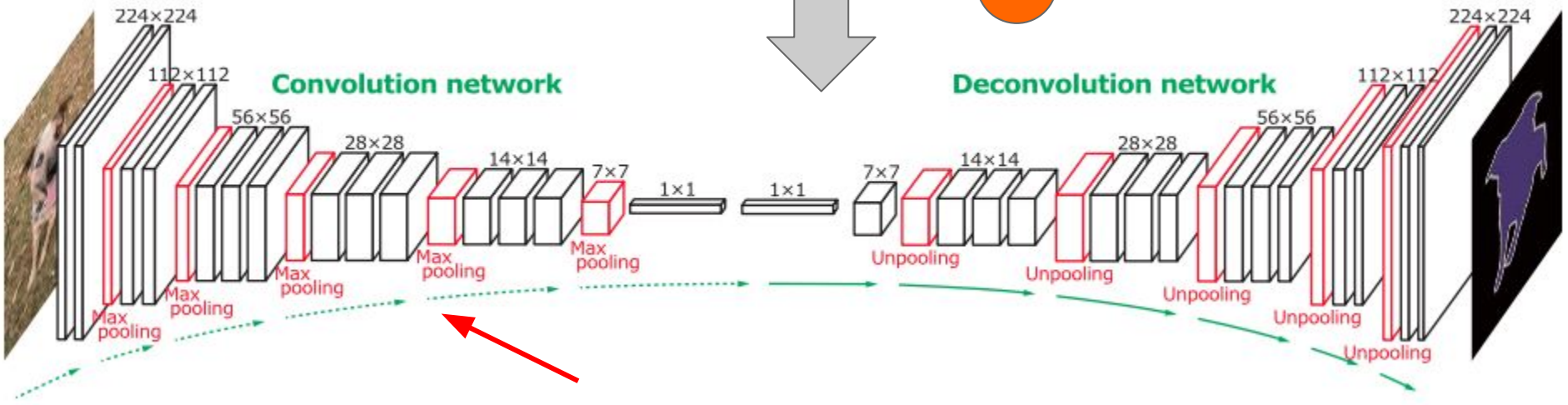


Deconvolutional Neural Networks (DNN)



Simbología:

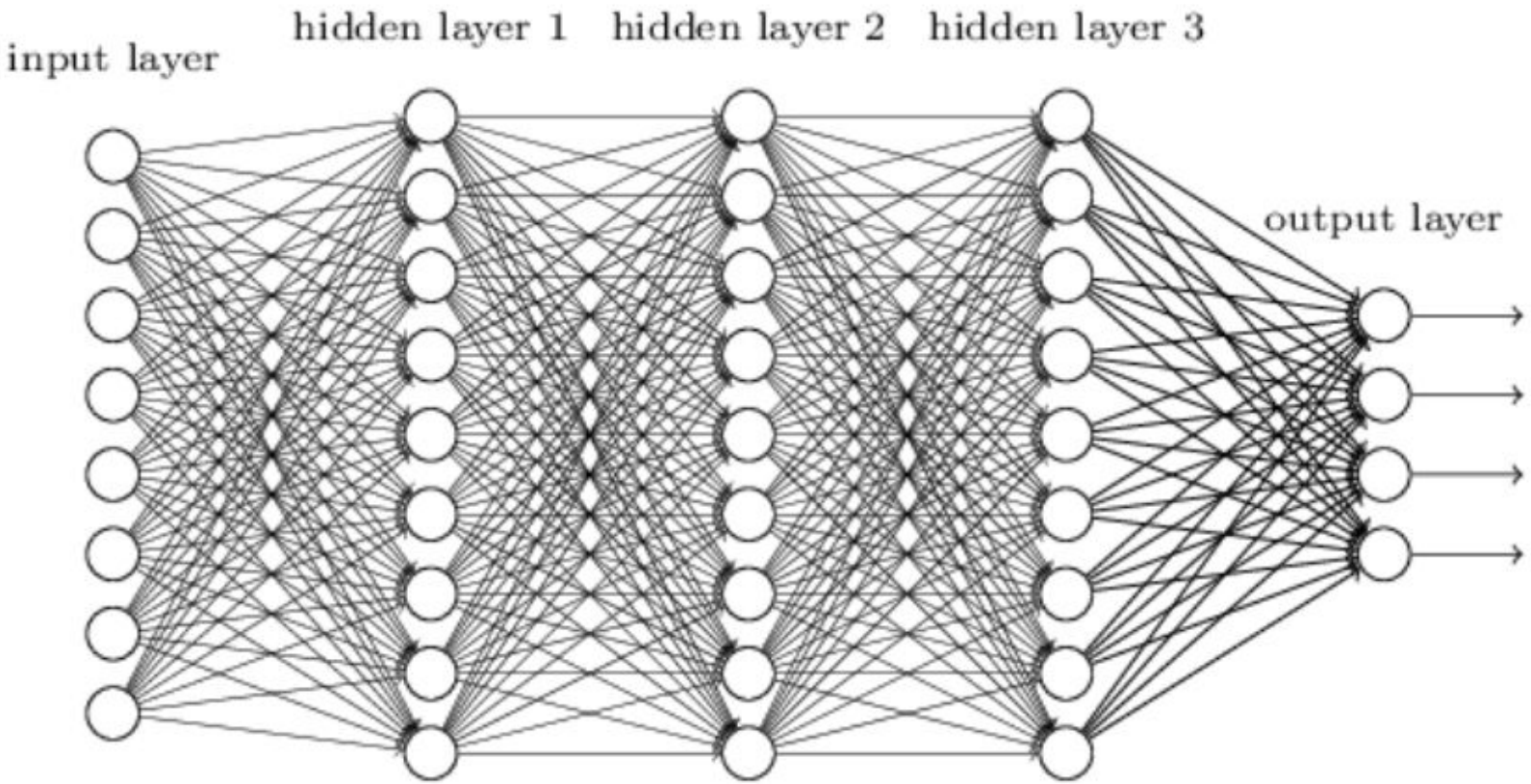
- Neurona de entrada
- Neurona Convolucional
- Neurona Pool
- Neurona de salida



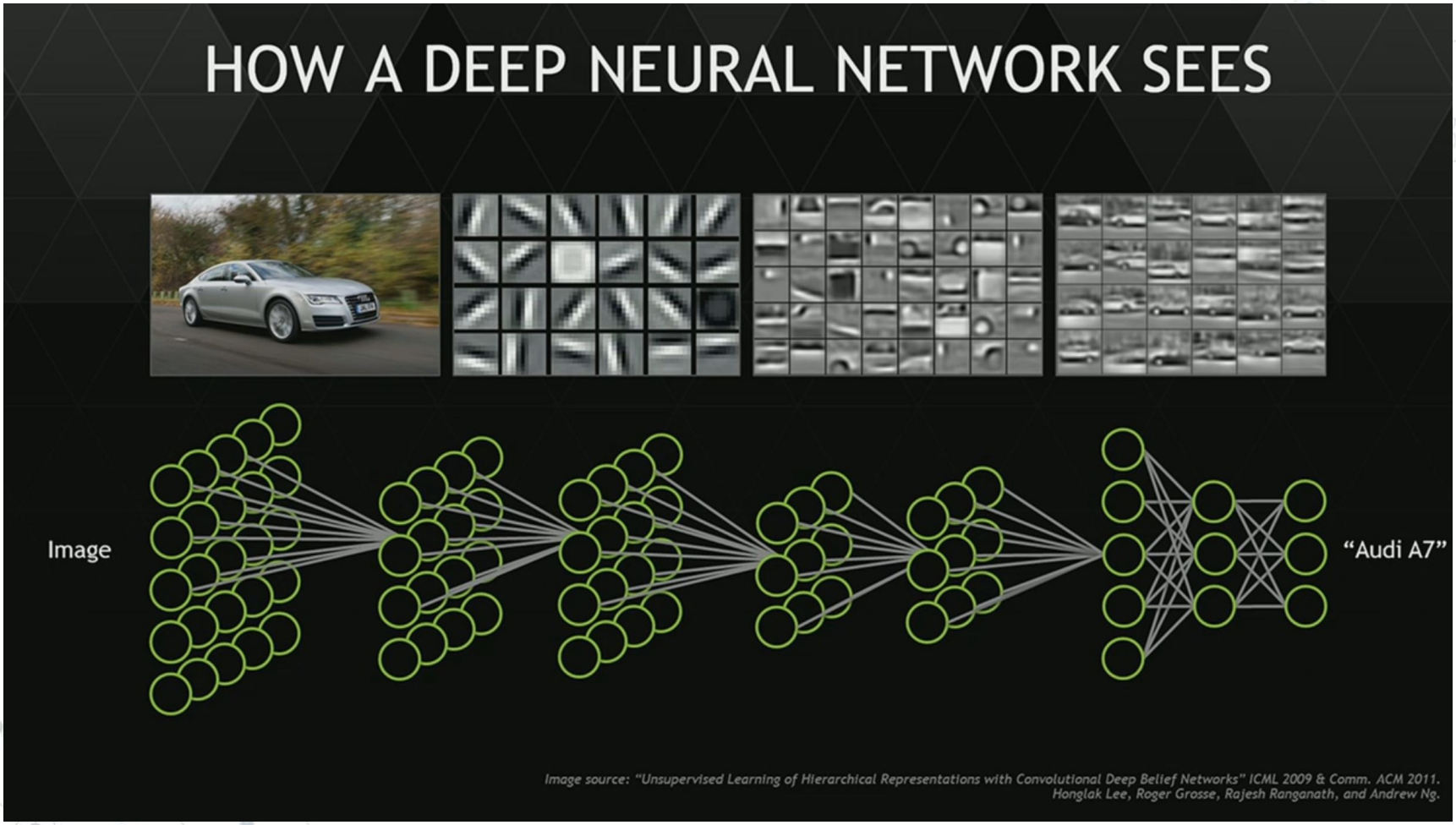
WTF IS THIS SH*T AGAIN PLEASE STOP

KEEP CALM

¿Pero entonces qué es Deep Learning?



¿Pero entonces qué es Deep Learning?



Un par de ‘detallitos’ o cinco….

- Las redes de neuronas se encuentran dentro de las técnicas de Inteligencia Artificial de ‘caja negra’
- Hay que probar y probar y probar y probar y probar y probar y probar y probar y probar y probar y probar arquitecturas de redes
- No es evidente por qué unas funcionan mejor y otras no
- No toda tu solución tiene que estar centrada en DL o NN hay más técnicas
- DL no es aprendizaje no supervisado y viceversa

2.
TensorFlow

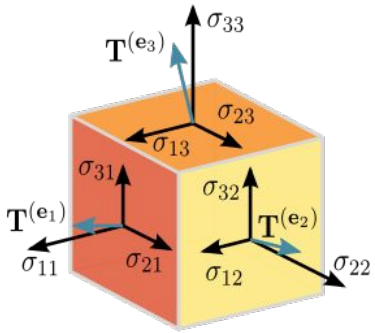
Abramos la caja negra de la que todo el mundo habla

Tensor Flow

Tensor ⟹ **N-dimensional array**

Vector ⟹ 1-D tensor

Matriz ⟹ 2-D tensor

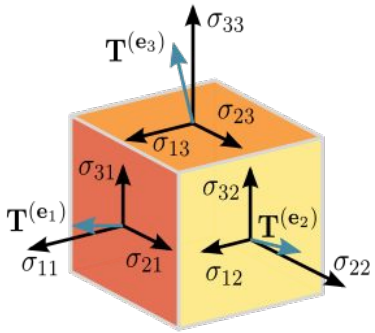


Tensor Flow

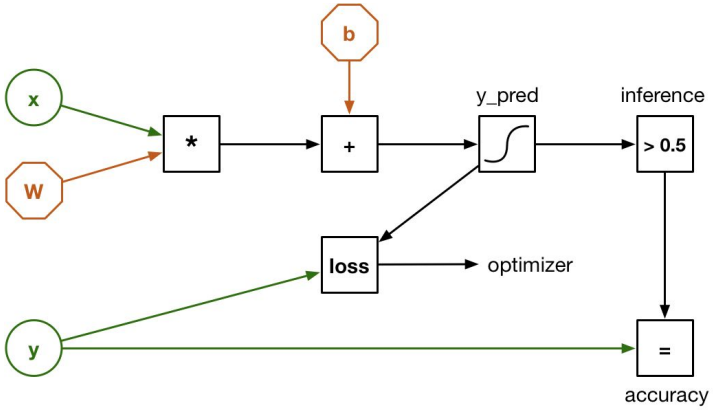
Tensor ⟹ **N-dimensional array**

Vector ⟹ 1-D tensor

Matriz ⟹ 2-D tensor



Flow ⟹ **Secuencia de operaciones**



Tensor Flow

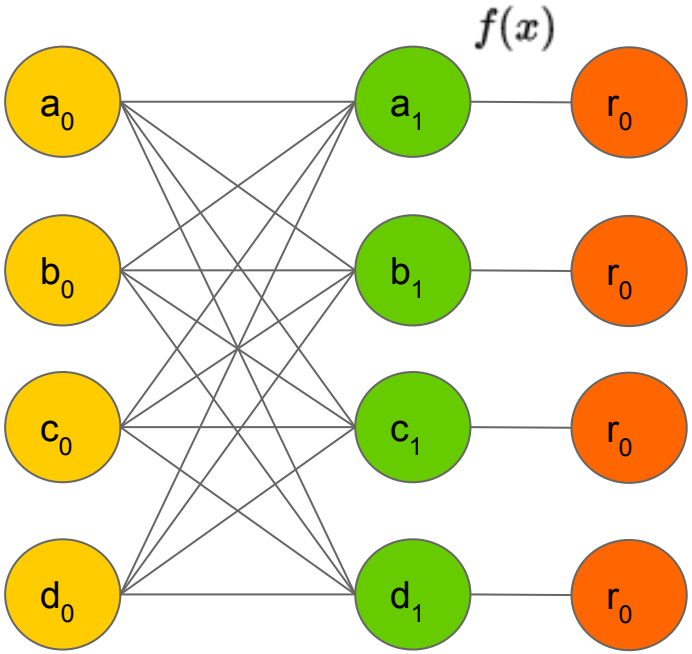
Rectified Linear Unit (ReLU) ⇒ $f(x) = \max(0, x)$

Funcion de activacion

Tensor Flow

Rectified Linear Unit (ReLU) $\Rightarrow$ $f(x) = \max(0, x)$

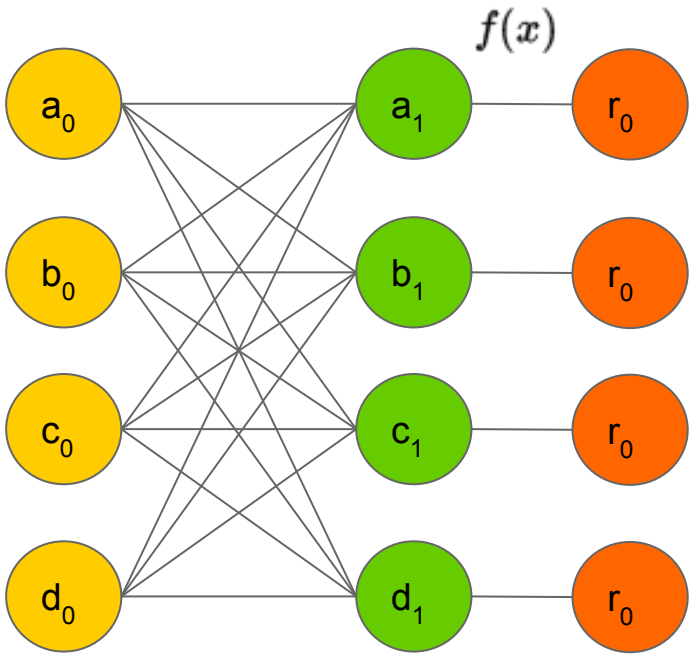
Funcion de activacion

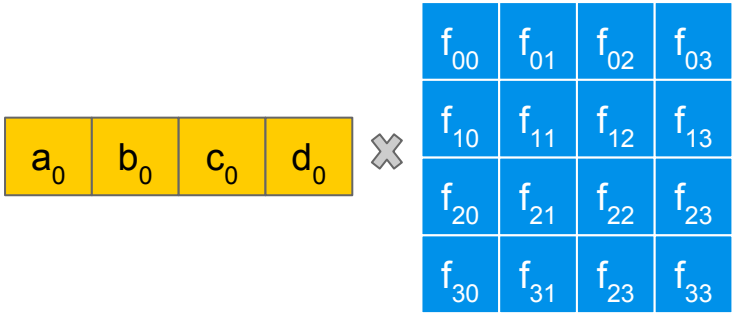


Tensor Flow

Rectified Linear Unit (ReLU) $\Rightarrow$ $f(x) = \max(0, x)$

Funcion de activacion





$$a_1 = a_0f_{00} + b_0f_{10} + c_0f_{20} + d_0f_{30}$$

.

.

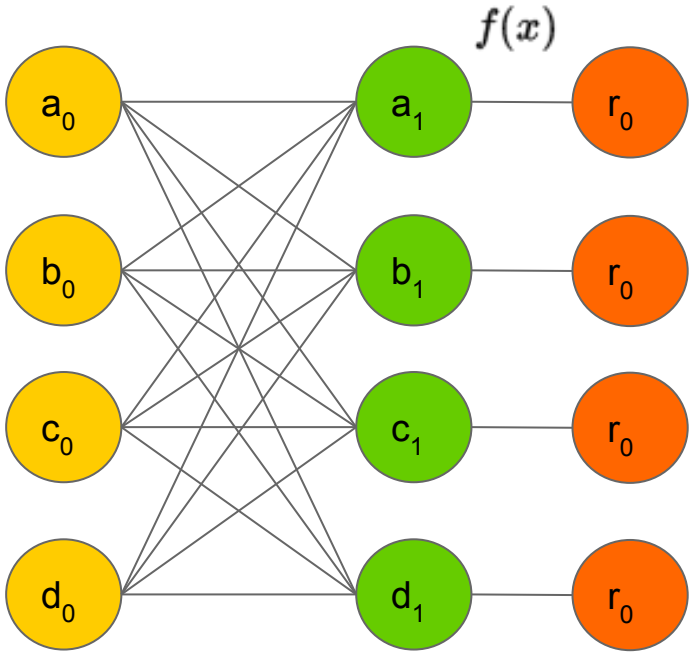
.

$$d_1 = a_0f_{03} + b_0f_{13} + c_0f_{23} + d_0f_{33}$$

Tensor Flow

Rectified Linear Unit (ReLU) $\Rightarrow$ $f(x) = \max(0, x)$

Funcion de activacion



ReLU $\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \end{bmatrix}$

$r_1 = \max(0, a_1)$

.

.

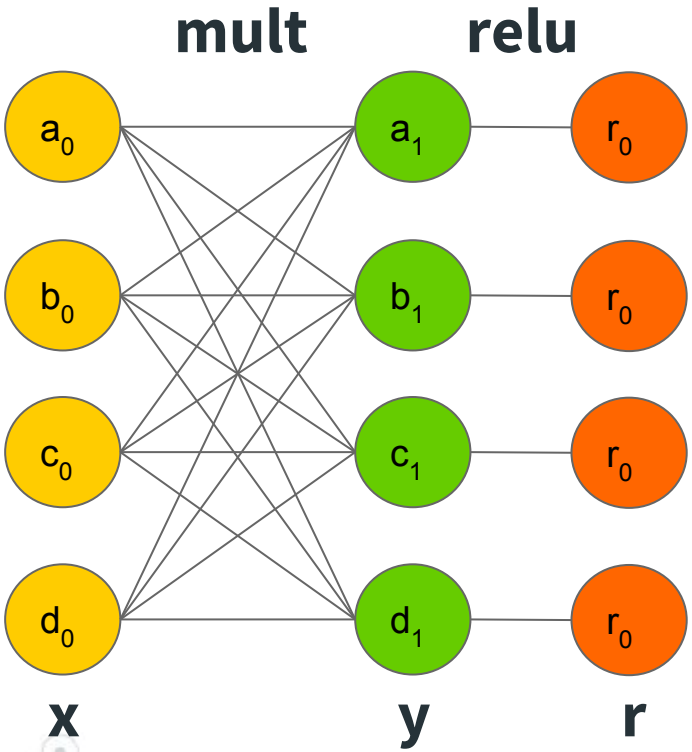
.

$r_1 = \max(0, a_1)$

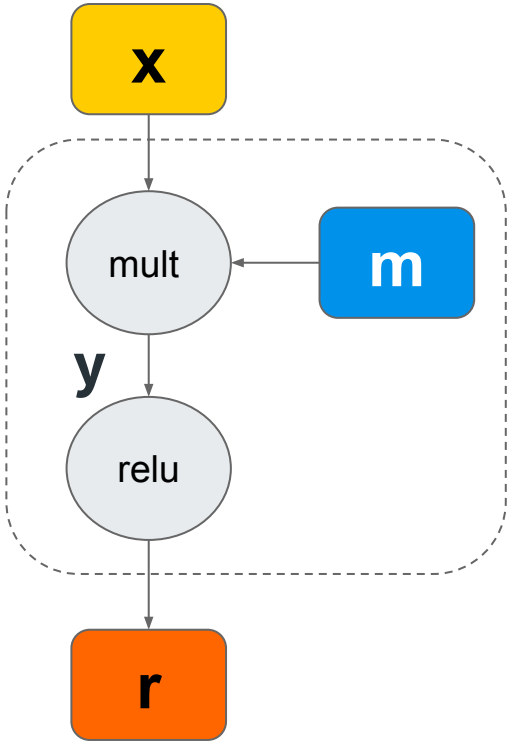
Tensor Flow

Rectified Linear Unit (ReLU) $f(x) = \max(0, x)$

Funcion de activacion



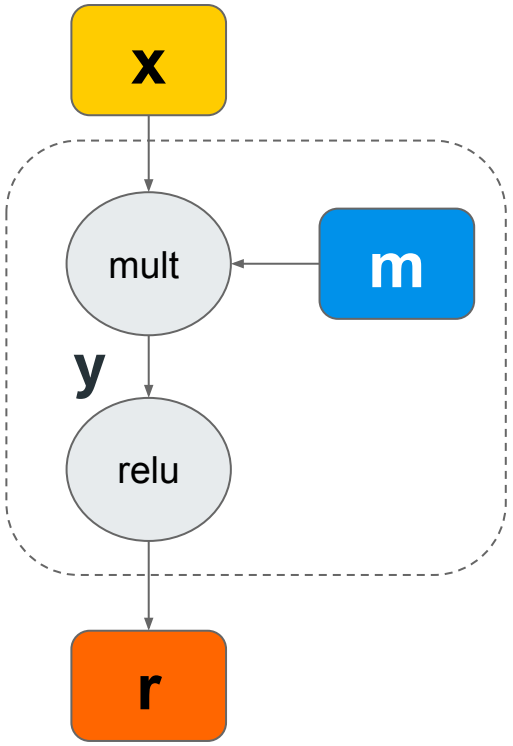
grafo



Tensor Flow

Rectified Linear Unit (ReLU) ⇒ $f(x) = \max(0, x)$

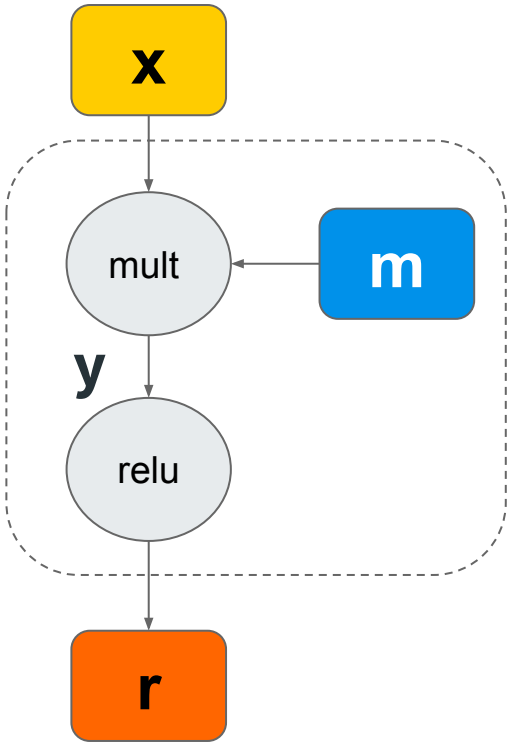
```
import tensorflow as tf
import numpy as np

session = tf.Session()

x = tf.placeholder("float", [1,4])
m = tf.Variable(tf.random_normal([4,4]), name='m')
y = tf.matmul(x, m)
r = tf.nn.relu(y)

session.run(tf.global_variables_initializer())

print(session.run(r, feed_dict {x:np.array([[1.0,2.0,3.0,4.0]])}))

session.close()
```



Tensor Flow

Rectified Linear Unit (ReLU) ⇒ $f(x) = \max(0, x)$

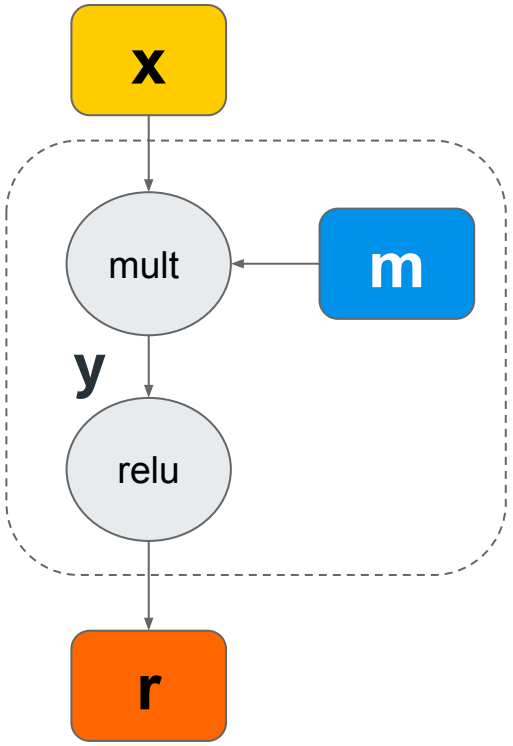
```
import tensorflow as tf
import numpy as np

session = tf.Session()

x = tf.placeholder("float", [1,4])
m = tf.Variable(tf.random_normal([4,4]), name='m')
y = tf.matmul(x, m)
r = tf.nn.relu(y)

session.run(tf.global_variables_initializer())

print(session.run(r, feed_dict {x:np.array([[1.0,2.0,3.0,4.0]])}))

session.close()
```



Tensor Flow

Rectified Linear Unit (ReLU) ⇒ $f(x) = \max(0, x)$

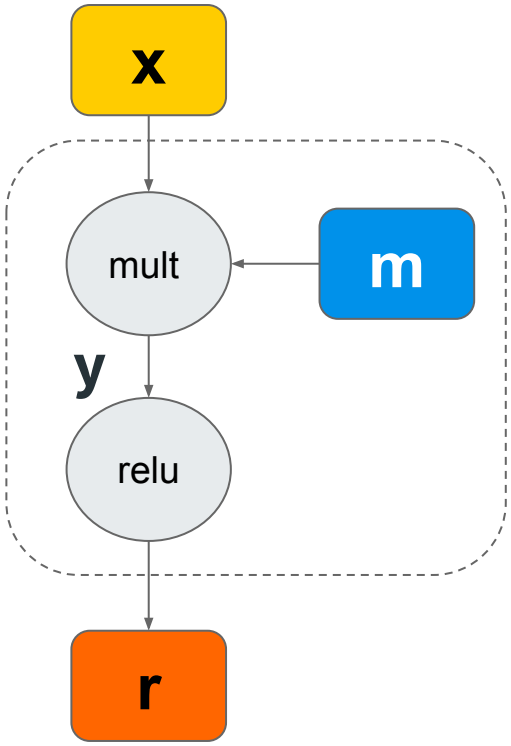
```
import tensorflow as tf
import numpy as np

session = tf.Session()

x = tf.placeholder("float", [1,4])
m = tf.Variable(tf.random_normal([4,4]), name='m')
y = tf.matmul(x, m)
r = tf.nn.relu(y)

session.run(tf.global_variables_initializer())

print(session.run(r, feed_dict {x:np.array([[1.0,2.0,3.0,4.0]])}))

session.close()
```



Tensor Flow

Rectified Linear Unit (ReLU) ⇒ $f(x) = \max(0, x)$

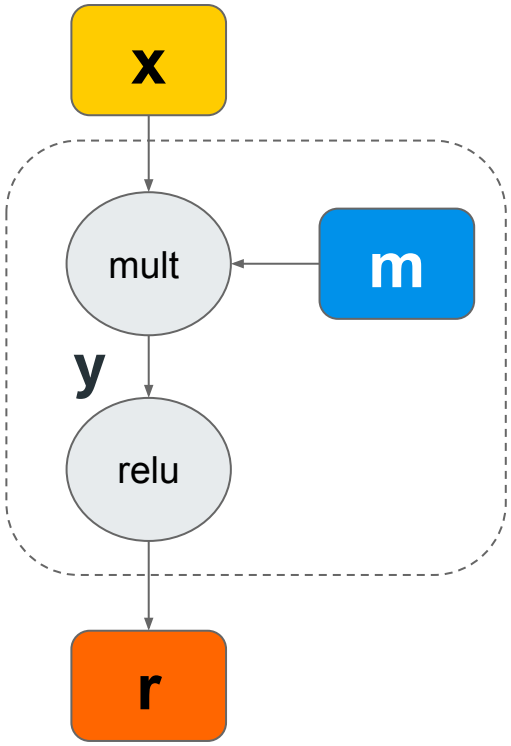
```
import tensorflow as tf
import numpy as np

session = tf.Session()

x = tf.placeholder("float", [1,4])
m = tf.Variable(tf.random_normal([4,4]), name='m')
y = tf.matmul(x, m)
r = tf.nn.relu(y)

session.run(tf.global_variables_initializer())

print(session.run(r, feed_dict {x:np.array([[1.0,2.0,3.0,4.0]])}))

session.close()
```



Tensor Flow

Rectified Linear Unit (ReLU) ⇒ $f(x) = \max(0, x)$

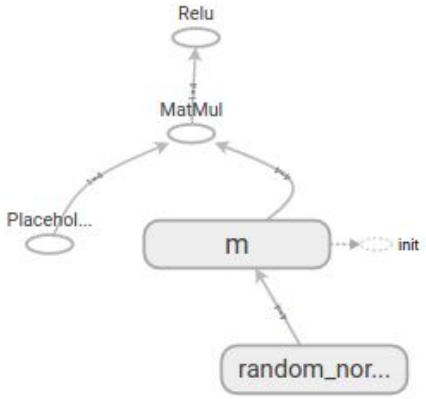
```
import tensorflow as tf
import numpy as np

session = tf.Session()

x = tf.placeholder("float", [1,4])
m = tf.Variable(tf.random_normal([4,4]), name='m')
y = tf.matmul(x, m)
r = tf.nn.relu(y)

session.run(tf.global_variables_initializer())

print(session.run(r, feed_dict {x:np.array([[1.0,2.0,3.0,4.0]])}))

session.close()
```



Tensor Flow

Rectified Linear Unit (ReLU) ⇒ $f(x) = \max(0, x)$

```
import tensorflow as tf
import numpy as np

session = tf.Session()

x = tf.placeholder("float", [1,4])
m = tf.Variable(tf.random_normal([4,4]), name='m')
y = tf.matmul(x, m)
r = tf.nn.relu(y)

session.run(tf.global_variables_initializer())

print(session.run(r, feed_dict {x:np.array([[1.0,2.0,3.0,4.0]])}))

session.close()
```

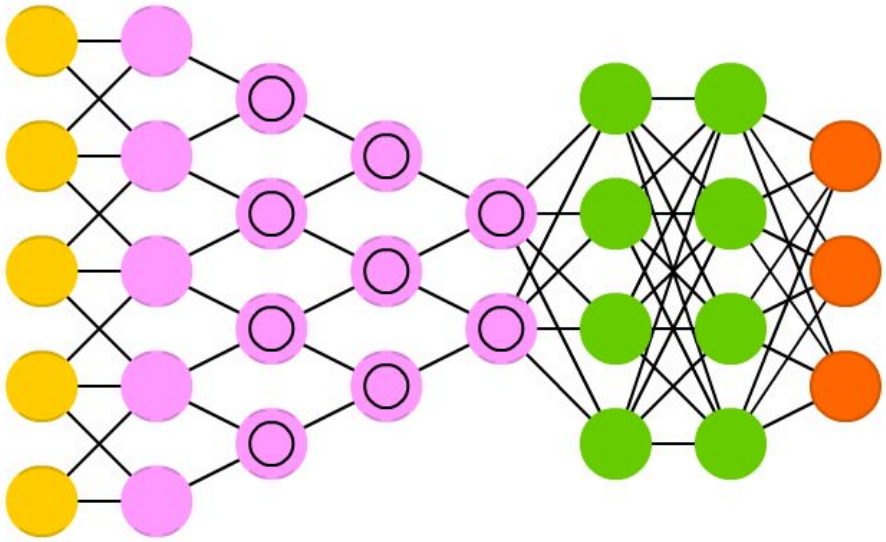


3.

Creando mi Red de Neuronas

¿Cómo la creo? ¿Qué necesito?

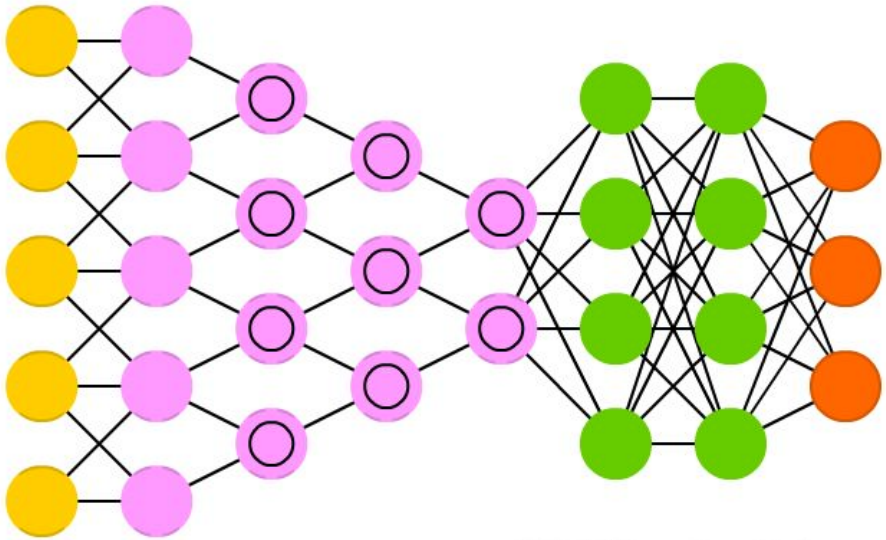
Redes Convolucionales

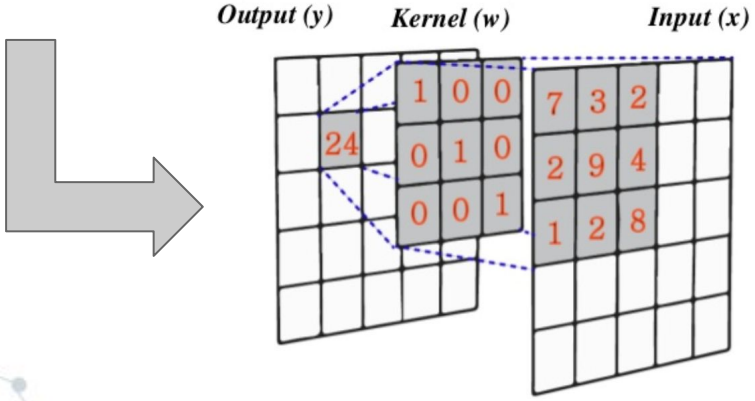


Operaciones

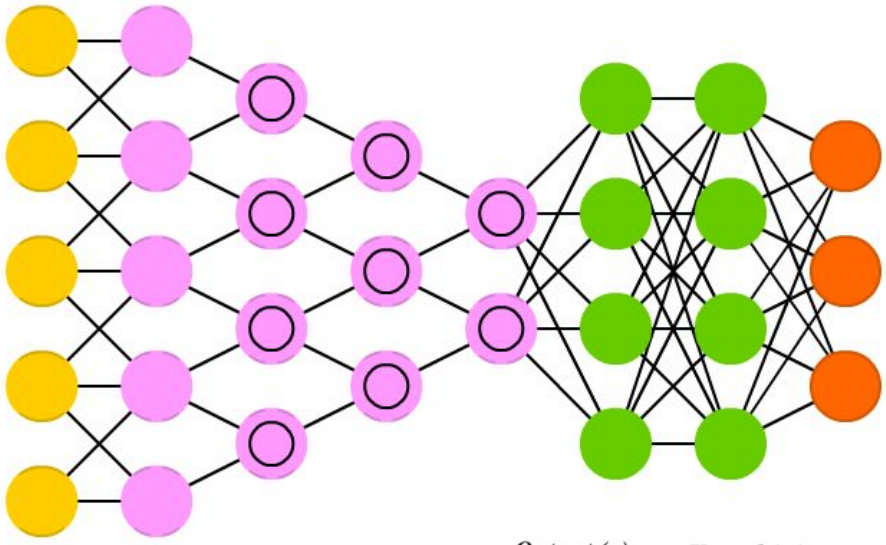
- Convoluciones
- Max-Pool
- Full connected (FC)

Redes Convolucionales: Convoluciones

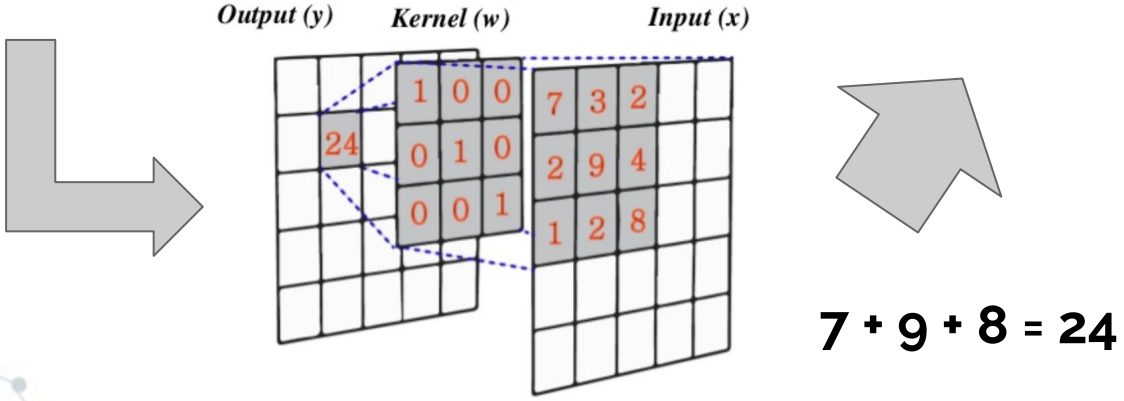




Redes Convolucionales: Convoluciones

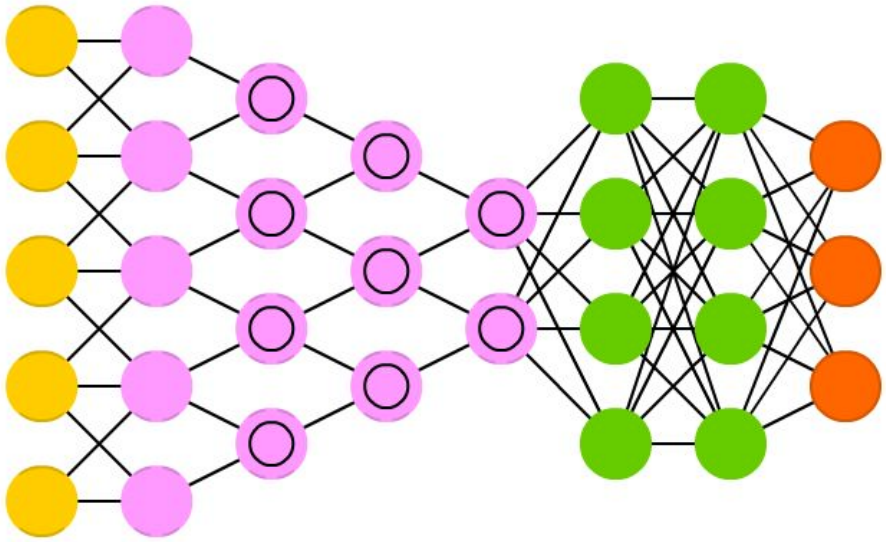


```
conv = tf.layers.conv2d(
    inputs=input_layer,
    filters=64,
    w=[6, 6],
    padding="same",
    activation=tf.nn.relu)
```

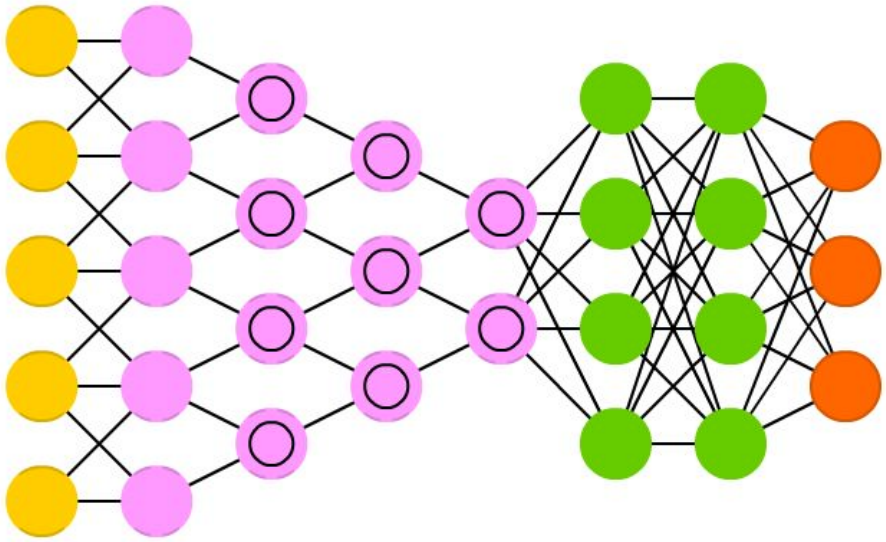


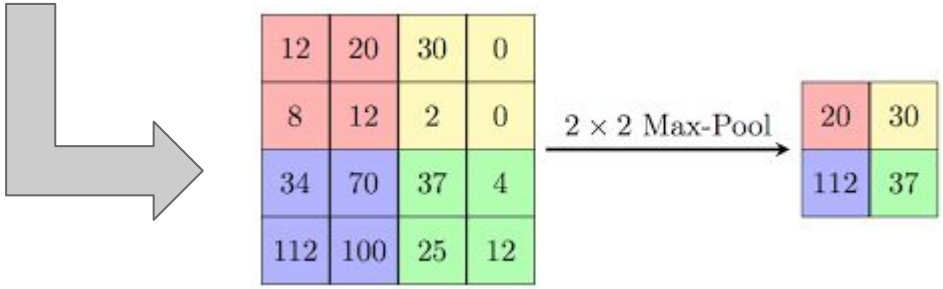
7 + 9 + 8 = 24

Redes Convolucionales: Max - Pooling

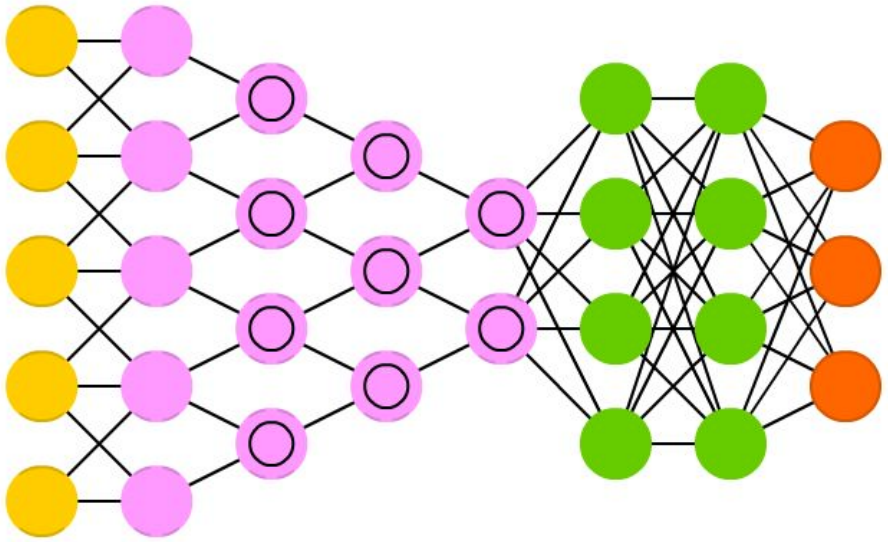


Redes Convolucionales: Max - Pool

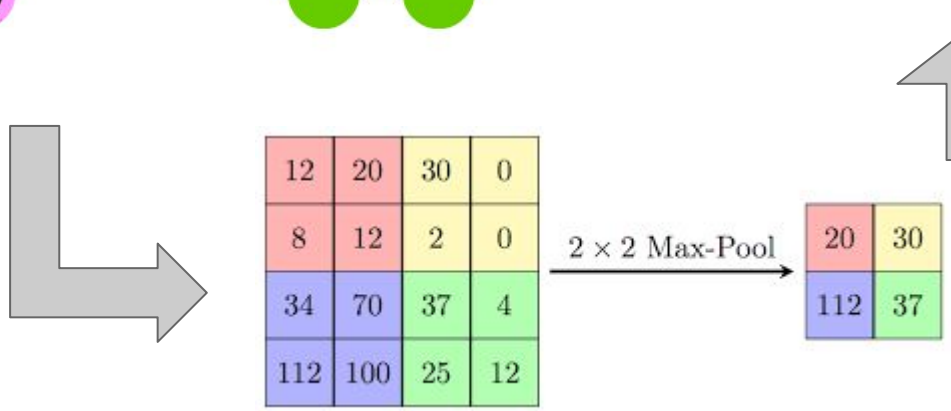




Redes Convolucionales: Max - Pooling

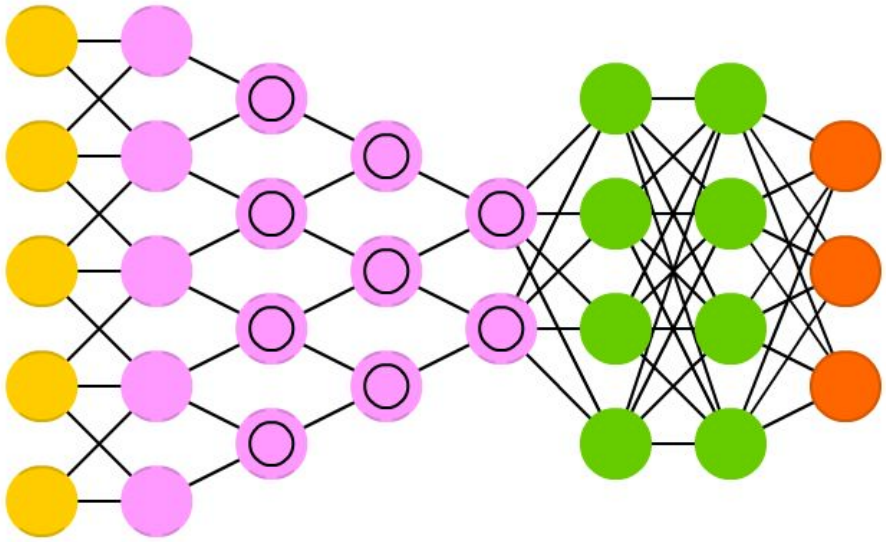


```
pool = tf.layers.max_pooling2d(
        inputs=conv,
        pool_size=[2, 2],
        strides=2)
```



Redes Convolucionales: Dense

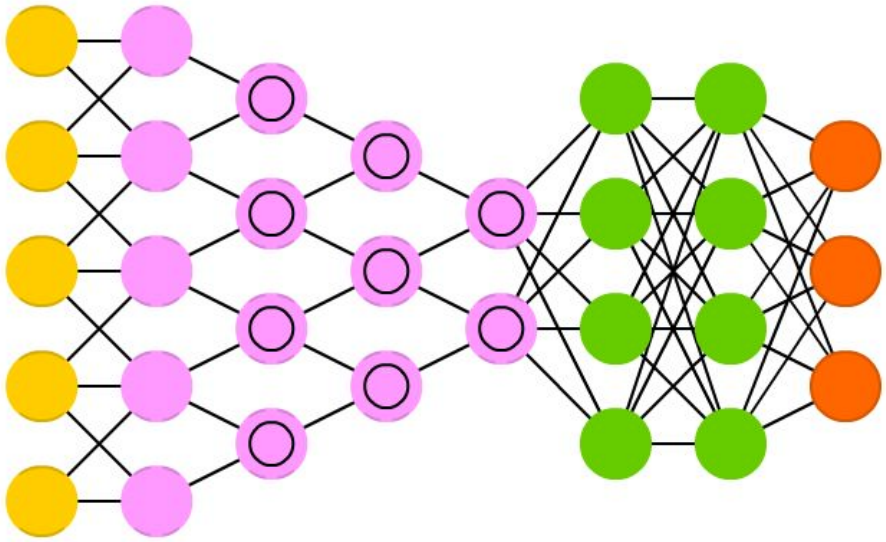
Extraer información **Clasificar información**



```
dense = tf.layers.dense(
        inputs=pool
        units=1024,
        activation=tf.nn.relu)
```

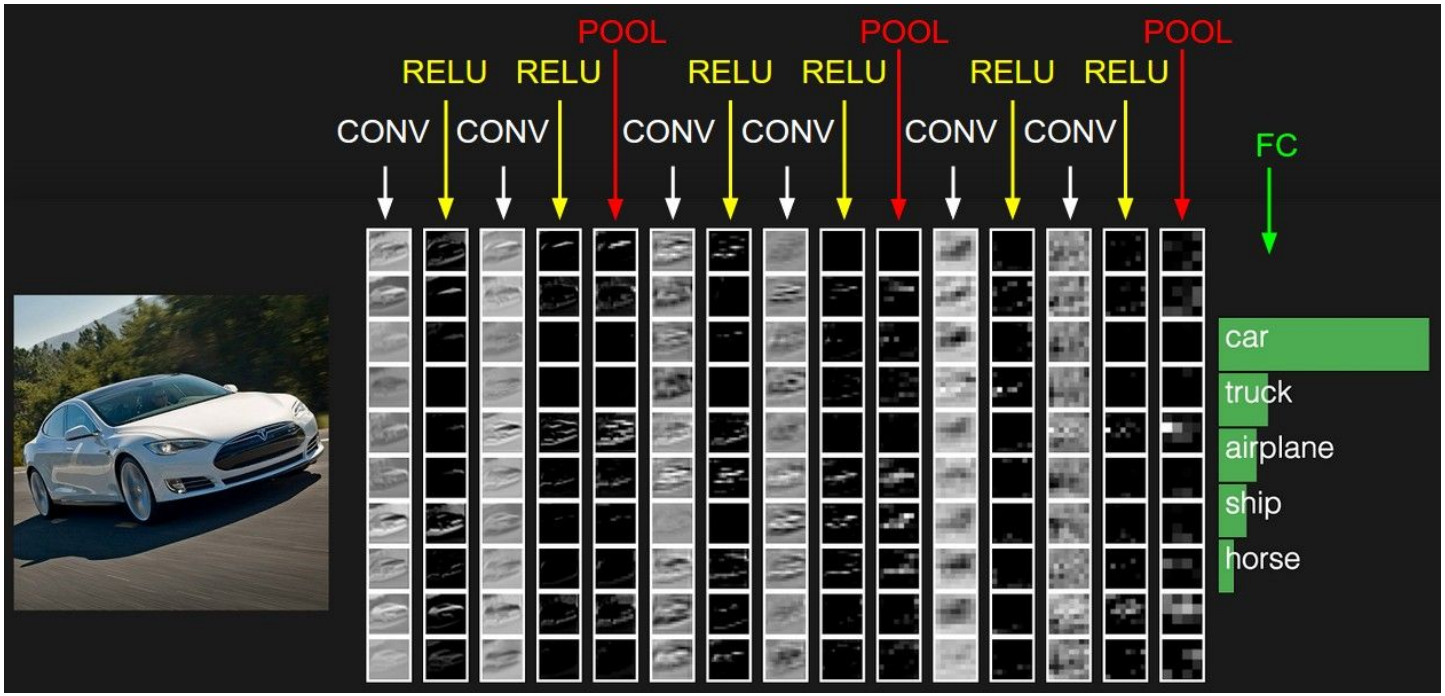
Redes Convolucionales: Dense

Extraer información **Clasificar información**



```
pool_reshape = tf.reshape(pool, [-1, 7 * 7 * 64])
dense = tf.layers.dense(
        inputs=pool_reshape
        units=1024,
        activation=tf.nn.relu)
```

Redes Convolucionales

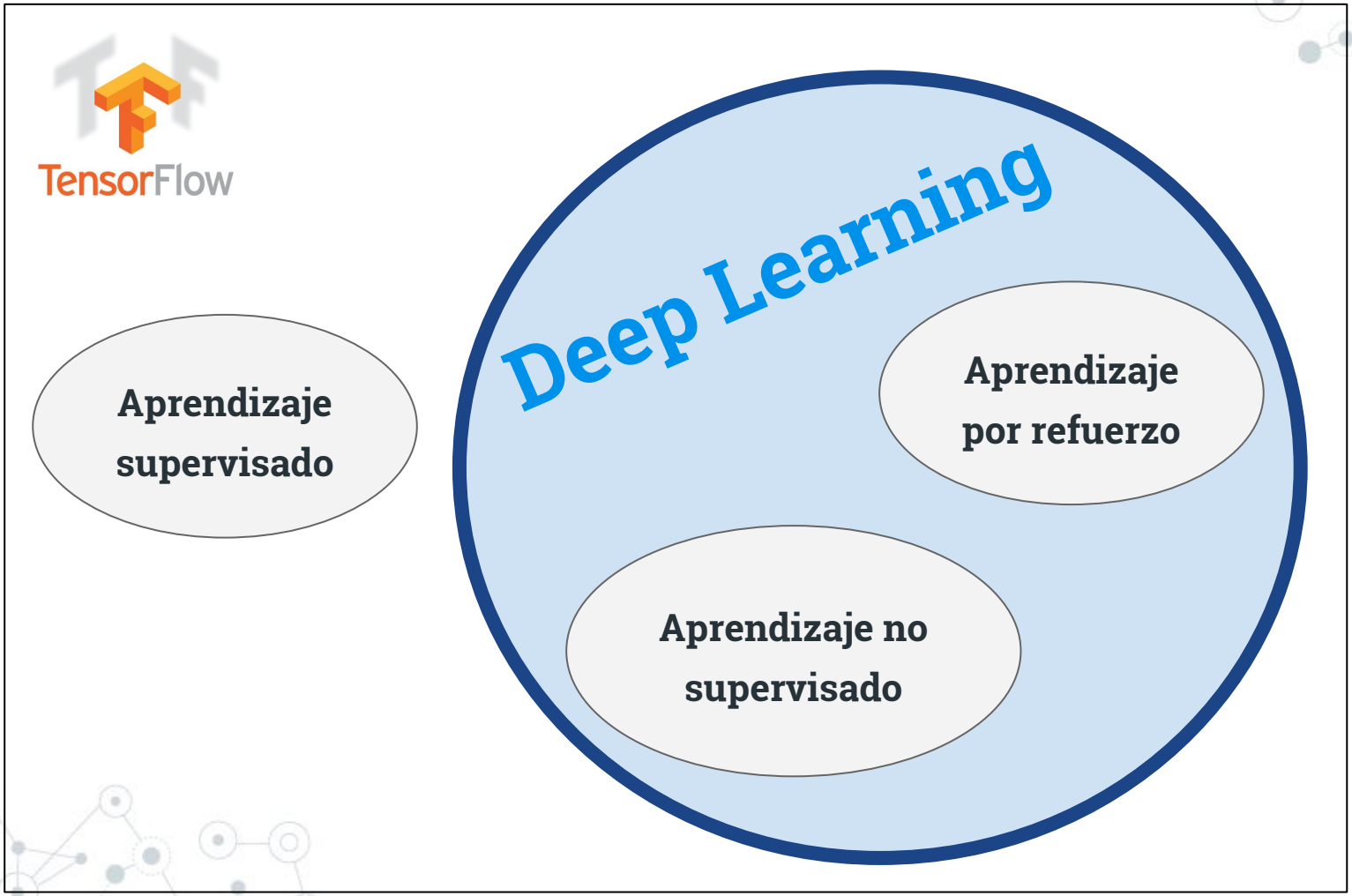


4.

Son sólo redes de neuronas

¿Podemos construir otros algoritmos?

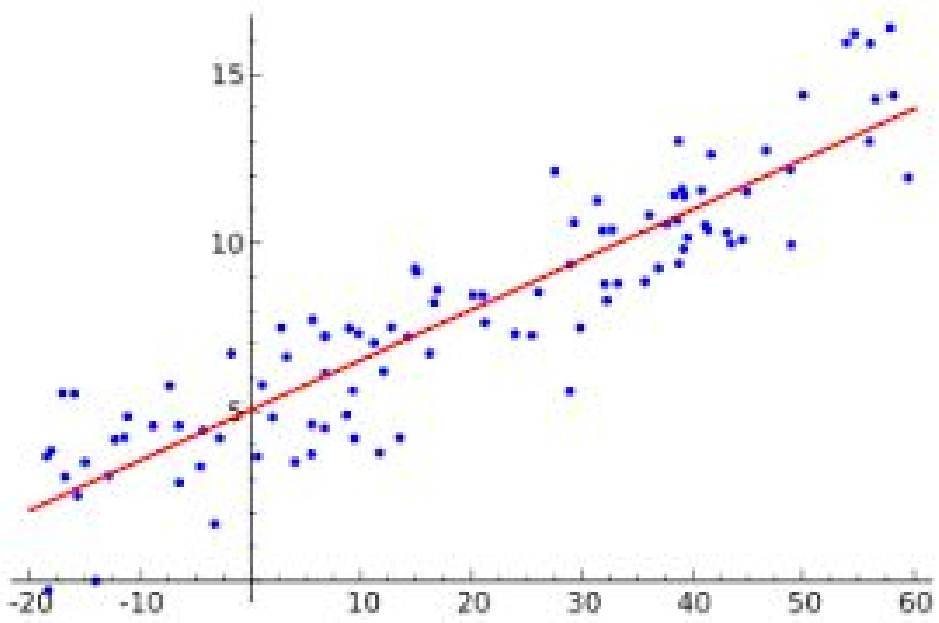
Más que redes de neuronas



Regresión Lineal

Construye un modelo para aproximar la relación de dependencia entre una variable dependiente Y y conjunto de variables independientes X.

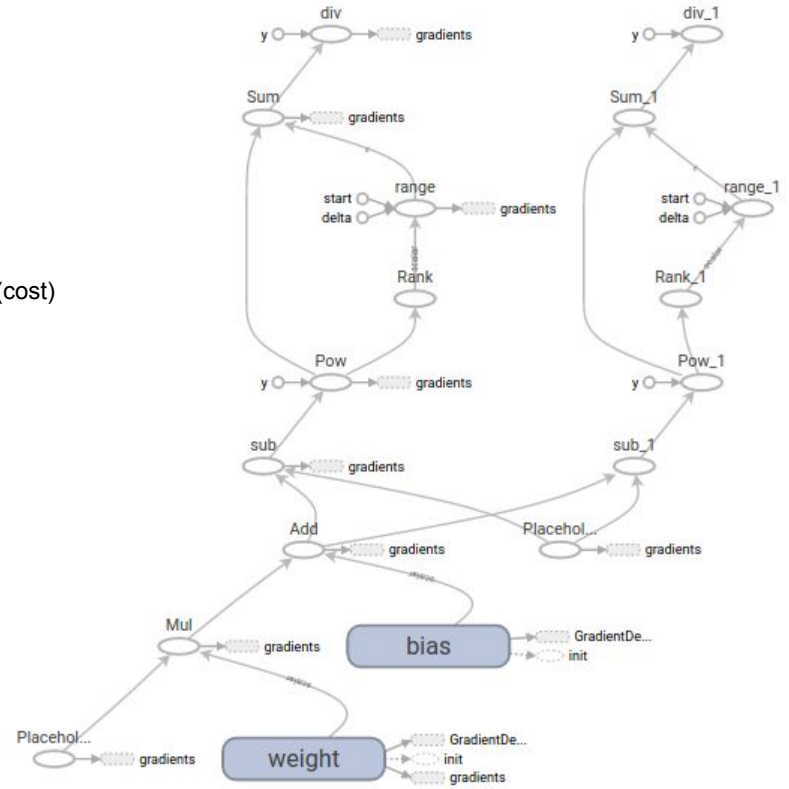
$$Y_t = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$



Regresión Lineal con TensorFlow

$$Y = w * X + b$$

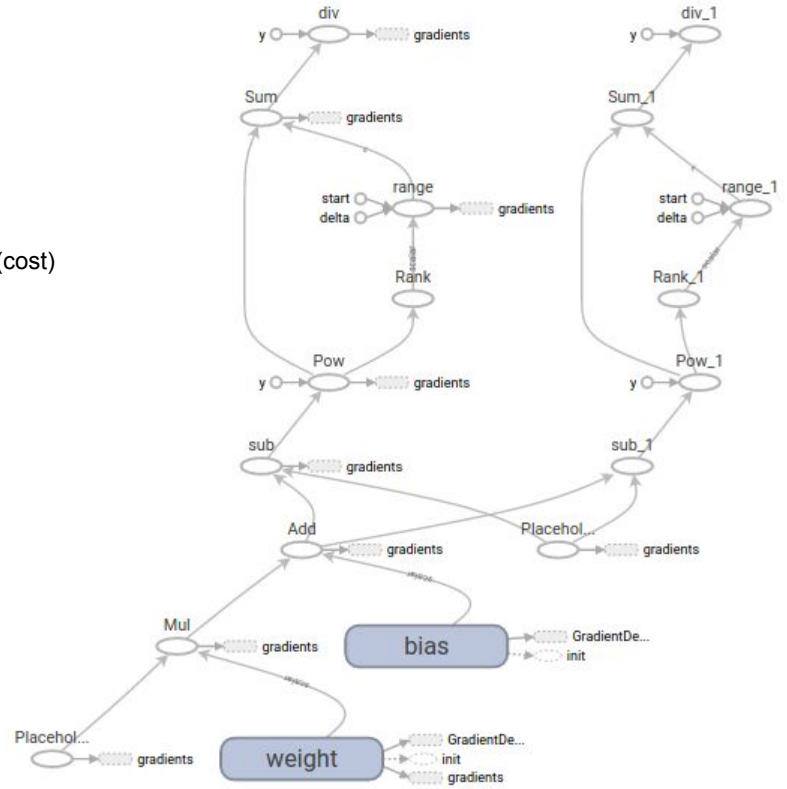
```
import tensorflow as tf
import numpy

rng = numpy.random

X = tf.placeholder("float")
Y = tf.placeholder("float")

w = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

operation = tf.add(tf.multiply(X, w), b)
cost = tf.reduce_sum(tf.pow(operation-Y, 2))/(2*n_samples)
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

with tf.Session() as session:

    session.run(tf.global_variables_initializer())
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})
        if (epoch+1) % display_step == 0:
            c = session.run(cost, feed_dict={X: train_X, Y:train_Y})
    training_cost = session.run(cost, feed_dict={X: train_X, Y: train_Y})

    testing_cost = session.run(
        tf.reduce_sum(tf.pow(operation-Y, 2)) / (2 * test_X.shape[0]),
        feed_dict={X: test_X, Y: test_Y})
```



Regresión Lineal con TensorFlow

```
import tensorflow as tf
import numpy

rng = numpy.random

X = tf.placeholder("float")
Y = tf.placeholder("float")

w = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

operation = tf.add(tf.multiply(X, w), b)
cost = tf.reduce_sum(tf.pow(operation-Y, 2))/(2*n_samples)
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

with tf.Session() as session:

    session.run(tf.global_variables_initializer())
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})
        if (epoch+1) % display_step == 0:
            c = session.run(cost, feed_dict={X: train_X, Y:train_Y})
    training_cost = session.run(cost, feed_dict={X: train_X, Y: train_Y})

    testing_cost = session.run(
        tf.reduce_sum(tf.pow(operation-Y, 2)) / (2 * test_X.shape[0]),
        feed_dict={X: test_X, Y: test_Y})
```

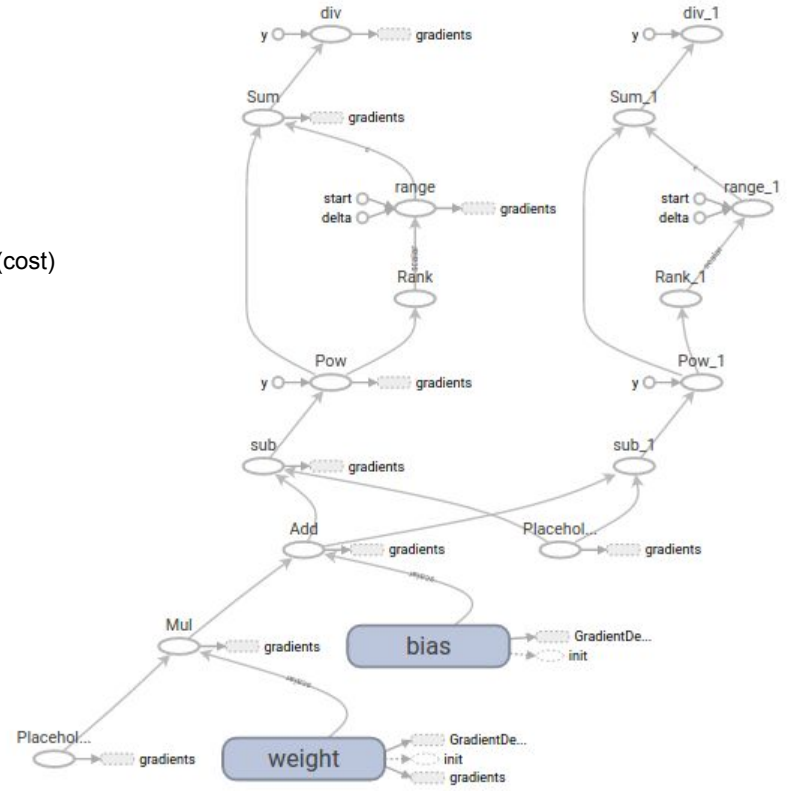
Variables de entrada



Regresión Lineal con TensorFlow

```
import tensorflow as tf
import numpy

rng = numpy.random

X = tf.placeholder("float")
Y = tf.placeholder("float")

w = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

operation = tf.add(tf.multiply(X, w), b)
cost = tf.reduce_sum(tf.pow(operation-Y, 2))/(2*n_samples)
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

with tf.Session() as session:

    session.run(tf.global_variables_initializer())
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})
        if (epoch+1) % display_step == 0:
            c = session.run(cost, feed_dict={X: train_X, Y:train_Y})
    training_cost = session.run(cost, feed_dict={X: train_X, Y: train_Y})

    testing_cost = session.run(
        tf.reduce_sum(tf.pow(operation-Y, 2)) / (2 * test_X.shape[0]),
        feed_dict={X: test_X, Y: test_Y})
```

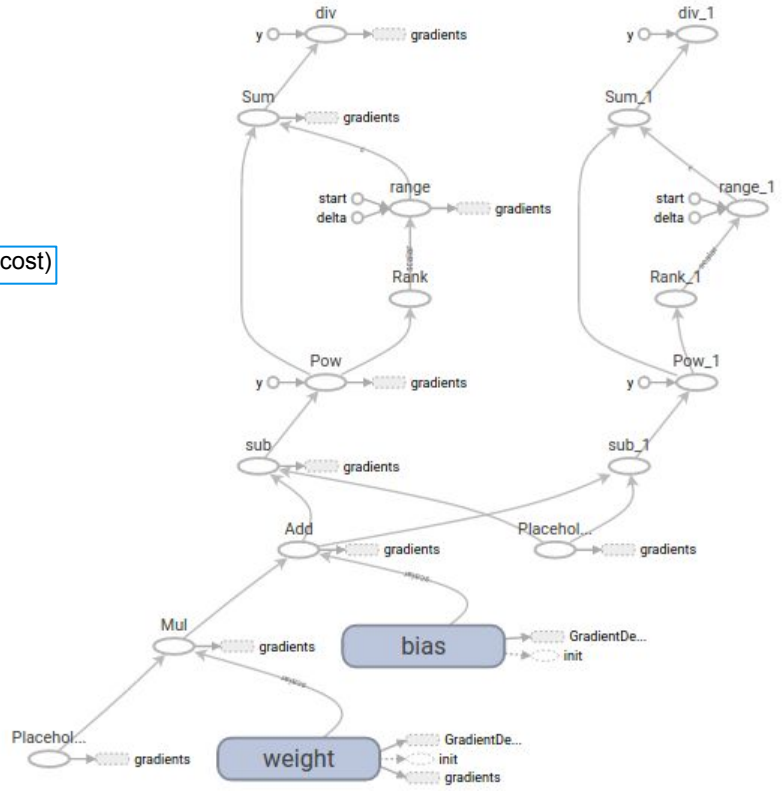
Recta de regresión



Regresión Lineal con TensorFlow

```
import tensorflow as tf
import numpy

rng = numpy.random

X = tf.placeholder("float")
Y = tf.placeholder("float")

w = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

operation = tf.add(tf.multiply(X, w), b)
cost = tf.reduce_sum(tf.pow(operation-Y, 2))/(2*n_samples)
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

with tf.Session() as session:

    session.run(tf.global_variables_initializer())
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})
        if (epoch+1) % display_step == 0:
            c = session.run(cost, feed_dict={X: train_X, Y:train_Y})
    training_cost = session.run(cost, feed_dict={X: train_X, Y: train_Y})

    testing_cost = session.run(
        tf.reduce_sum(tf.pow(operation-Y, 2)) / (2 * test_X.shape[0]),
        feed_dict={X: test_X, Y: test_Y})
```

Algoritmo



5.
Conclusiones

¿Qué has aprendido en 35 minutos?

Conclusiones

- TensorFlow nos permite construir diferentes tipos de algoritmos
- Las redes convolucionales son complicadas
- No es sólo sirve para crear redes de neuronas.
- El algoritmo se transforma en un grafo de operaciones dependiendo de los datos (tensores)
- Las operaciones sobre los tensores se paralelizan

http://t3chfest.uc3m.es

4 TRACKS | 90 CHARLAS/TALLERES | CONCURSO PROGRAMACIÓN | HACKATHON | GOOGLE HASH CODE | 1500 PERSONAS... ¡Y FALTAS TU!

¡Envíanos tu propuesta!
DEADLINE: DICIEMBRE

¡Muchas Gracias!

Preguntas?

@moisipm

@sailormercury91
@witsstories

https://github.com/momartinm/codemotion2017-TensorFlow