



INTRODUCCIÓN A REDES NEURONALES CON TENSORFLOW Y KERAS

Emilio Barragán Rodríguez

ÍNDICE

Redes neuronales

- Overview
- Perceptrón
- Red neuronal
- Aprendizaje

TensorFlow

- ¿Qué es?
- Tensores
- Primeros pasos

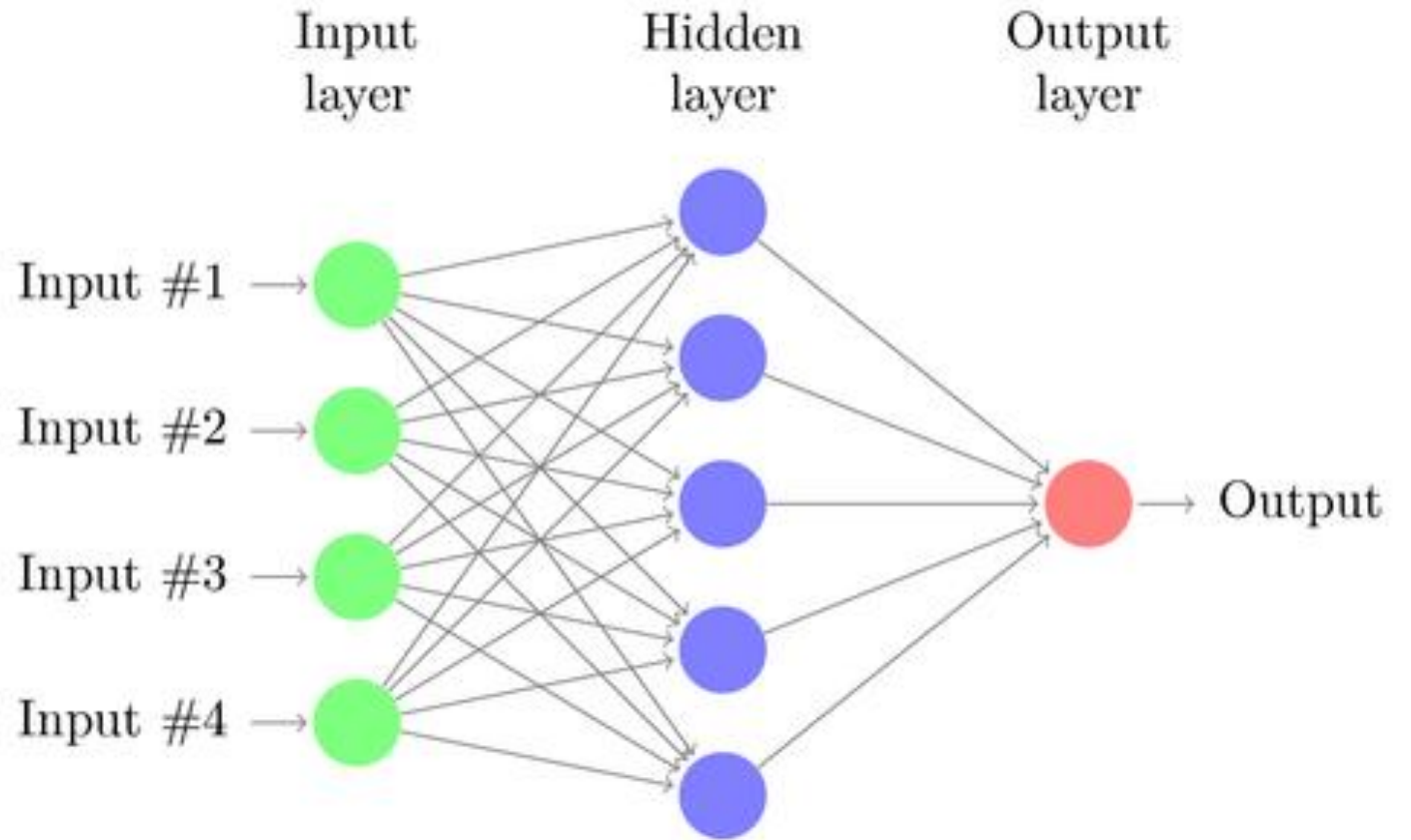
Keras

- ¿Qué es?
- Construyendo una red neuronal simple
 - `tf.keras.Sequential`
 - API funcional
- Compilando y entrenando la red
- Evaluando y generando predicciones

REDES NEURONALES

OVERVIEW

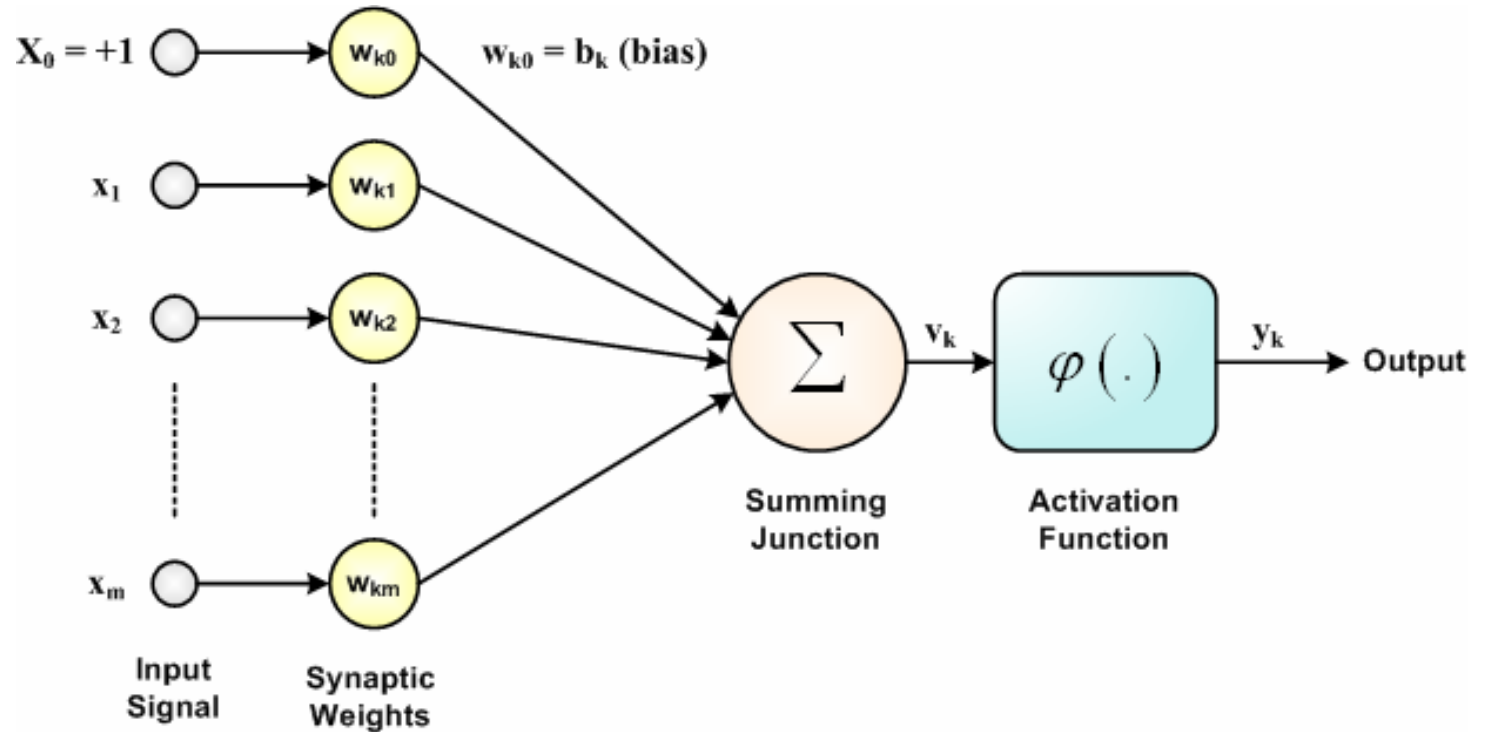
- Es un "cerebro", al que le damos datos y nos devuelve un resultado.
- Formado por perceptrones (neuronas).
- Perceptrones forman capas.
- Algoritmo de aprendizaje.



PERCEPTRÓN

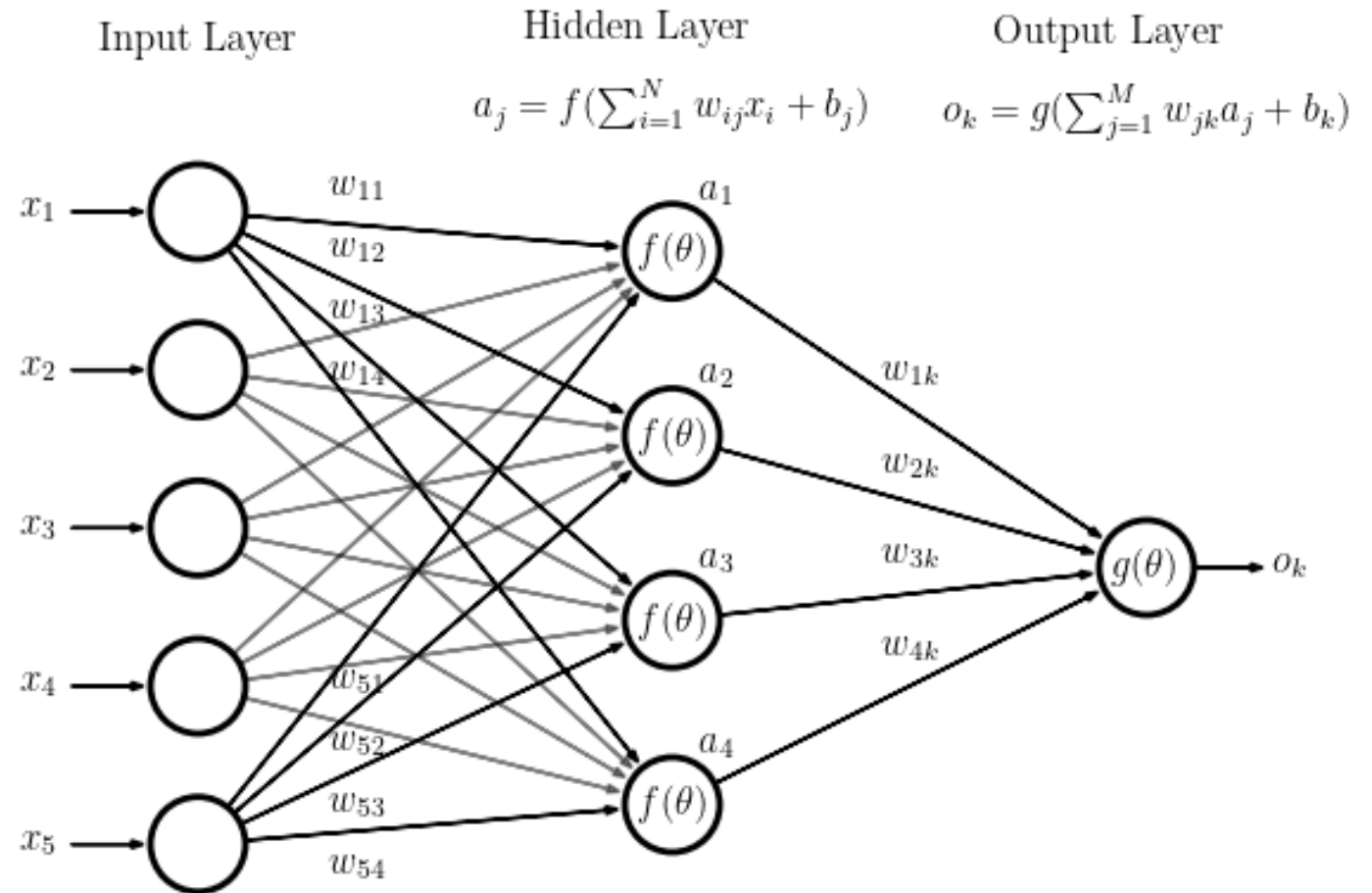
- Son las neuronas.
- Tiene unos datos de entrada.
- Formados por unos pesos sinápticos (synaptic weights).
- Tiene un sesgo (bias).
- Suma y función de activación.
- Nos devuelve una salida:

$$y = \varphi(x_0 w_0 + \sum_{i=1}^n x_i w_i)$$



RED NEURONAL

- Es el "cerebro" que forman los perceptrones.
- Formada por dos o más capas.
- Capa inicial se llama capa de entrada
- Capas intermedias se llaman capas ocultas.
- Capa final se llama capa de salida.



APRENDIZAJE

- Se itera sobre los datos (puede tener criterios de parada).
- Se basa en actualizar los pesos de los perceptrones:

$$w_i \leftarrow w_i + \Delta w_i$$

where

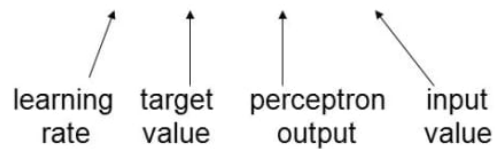
$$\Delta w_i = \eta(t - o)x_i$$

learning
rate

target
value

perceptron
output

input
value



- Para redes neuronales se suele usar el algoritmo de "backpropagation".

TENSORFLOW

¿QUÉ ES?

- Plataforma open source para aprendizaje automático.
- Nos permite manejar tensores de manera sencilla.
- Lo usaremos con Python.



TensorFlow

TENSORES

- Son objetos n-dimensionales:

Escalar \Rightarrow Tensor 0-dimensional

Vector \Rightarrow Tensor 1-dimensional

Matriz \Rightarrow Tensor 2-dimensional

Scalar

1

Vector

$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Matrix

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Tensor

$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$

PRIMEROS PASOS

- Importar TensorFlow:

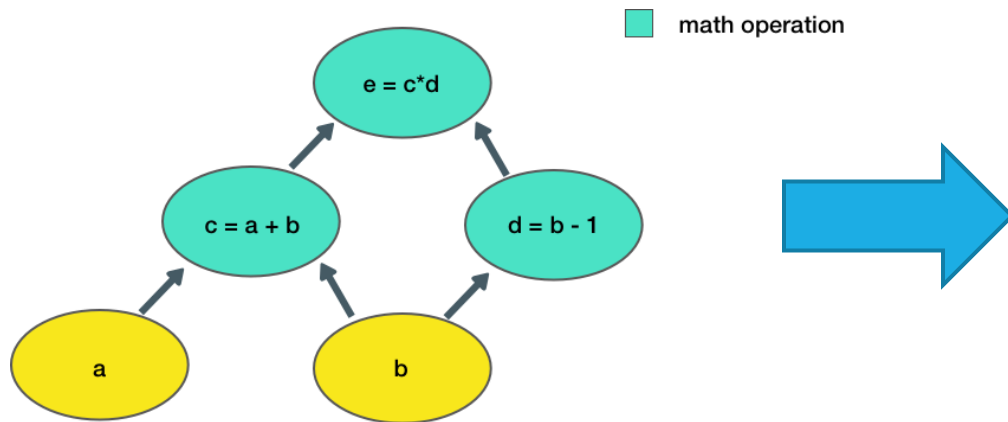
```
%tensorflow_version 2.x  
import tensorflow as tf
```

- Creación de tensores:

```
constant_number = tf.constant(100, dtype=tf.int32)  
tensor_1d = tf.constant([1,2,3,4]) # vector  
tensor_2d = tf.constant([[1,2,3],[4,5,6]]) # matrix  
tensor_3d = tf.constant([[[1,2,3],[4,5,6]], [[1,2,3],[4,5,6]]]) # cube
```

PRIMEROS PASOS

- Operando con tensores:



```
## APPROACH 1 ##
# We simply perform the operations

# Initial nodes in the graph:
a2 = tf.constant(1)
b2 = tf.constant(2)

# Intermediate nodes:
c2 = tf.add(a2,b2)
d2 = tf.subtract(b2,1)

# Final node:
e2 = tf.multiply(c2,d2) # element-wise matrices multiplication

print("Result from approach 1: {}".format(e2))

## APPROACH 2 ##
# We define a function to compute the operation

def operation(a,b): # We suppose that 'a' and 'b' are tensors
    c,d = tf.add(a,b), tf.subtract(b,1)
    return tf.multiply(c,d)

print("Result from approach 2: {}".format(operation(a2,b2)))
```

Result from approach 1: 3

Result from approach 2: 3

KERAS

¿QUÉ ES?

- API de alto nivel para construir redes neuronales.
- Escrita en Python.
- Corre sobre TensorFlow.
- Centrada en permitir una experimentación rápida.



Keras

CONSTRUYENDO UNA RED NEURONAL SIMPLE

Vamos a ver dos formas de crear una red neuronal:

- Usando `tf.keras.Sequential`
- Usando la API funcional.

CONSTRUYENDO UNA RED NEURONAL SIMPLE

- Usando `tf.keras.Sequential`:

```
from tensorflow.keras import layers

model = tf.keras.Sequential()
# Adds a densely-connected layer with 64 units to the model,
# with the input data shape -> (32,)
model.add(layers.Dense(64, activation='relu', input_shape=(32,)))
# Add another:
model.add(layers.Dense(64, activation='relu'))
# Add an output layer with 10 output units:
model.add(layers.Dense(10))
```


CONSTRUYENDO UNA RED NEURONAL SIMPLE

- Usando la API funcional:

```
# We create the input data , with shape -> (32,)
inputs = keras.Input(shape=(32,))
# We create a densely-connected layer with 64 units to the model
dense = layers.Dense(64, activation='relu')
# We combine the input shape and the dense layer to build our input layer
# and store it in x
x = dense(inputs)
# We add another dense layer with 64 units to x
x = layers.Dense(64, activation='relu')(x)
# We create the output layer with 10 units by adding one more layer to x
outputs = layers.Dense(10)(x)
model = keras.Model(inputs=inputs, outputs=outputs, name='mnist_model')
```

COMPILANDO Y ENTRENANDO LA RED

- Compilando la red:

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),  
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),  
              metrics=['accuracy'])
```

- Entrenando la red:

```
import numpy as np  
  
# We load our data and our targets(labels)  
data = np.random.random((1000, 32))  
labels = np.random.random((1000, 10))  
  
# We train our model  
model.fit(data, labels, epochs=10, batch_size=32)
```

EVALUANDO Y GENERANDO PREDICCIONES

- Evaluando el modelo:

```
# With Numpy arrays  
data = np.random.random((1000, 32))  
labels = np.random.random((1000, 10))  
  
model.evaluate(data, labels)
```

- Generando predicciones:

```
result = model.predict(data)  
print(result)
```

DEMO

