ActividadSesion10

May 16, 2020

1 Ejemplo Keras y Tensorflow

1.1 Ejemplo 1

Para el ejemplo, utilizaremos las compuertas AND. Recordemos que funcionan de la siguiente manera:

Tenemos dos entradas binarias (1 ó 0) y la salida será 1 sólo si las dos entradas son verdadera (1).

Es decir que de cuatro combinaciones posibles, sólo una tiene salida 1 y las otras tres serán 0, como vemos aquí:

- XOR(0,0) = 0
- XOR(0,1) = 0
- XOR(1,0) = 0
- XOR(1,1) = 1

Primero importamos las clases que utilizaremos:

[1]:
```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

```
Using TensorFlow backend.
/home/alberto/.local/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:516: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:517: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:518: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
```

```
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/alberto/.local/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

Utilizaremos numpy para el manejo de arrays. De Keras importamos el tipo de modelo Sequential y el tipo de capa Dense que es la «normal».

Creamos los arrays de entrada y salida.

[2]:
```
# cargamos las 4 combinaciones de las compuertas XOR
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")

# y estos son los resultados que se obtienen, en el mismo orden
target_data = np.array([[0],[0],[0],[1]], "float32")
```

Como se puede ver son las cuatro entradas posibles de la función XOR [0,0], [0,1], [1,0],[1,1] y sus cuatro salidas: 0, 0,0,1. Ahora crearemos la arquitectura de nuestra red neuronal:

[3]:
```
model = Sequential()
model.add(Dense(16, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Primero creamos un modelo vacío de tipo Sequential. Este modelo se refiere a que crearemos una serie de capas de neuronas secuenciales, «una delante de otra».

Agregamos dos capas Dense con «model.add()». Realmente serán 3 capas, pues al poner input_dim=2 estamos definiendo la capa de entrada con 2 neuronas (para nuestras entradas de la función AND) y la primer capa oculta (hidden) de 16 neuronas. Como función de activación utilizaremos «relu» que sabemos que da buenos resultados. Podría ser otra función, esto es un mero ejemplo, y según la implementación de la red que haremos, deberemos variar la cantidad de neuronas, capas y sus funciones de activación.

Y agregamos una capa con 1 neurona de salida y función de activación sigmoid.

Antes de de entrenar la red haremos unos ajustes de nuestro modelo:

[4]:
```
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['binary_accuracy'])
```

Con esto indicamos el tipo de pérdida (loss) que utilizaremos, el «optimizador» de los pesos de las conexiones de las neuronas y las métricas que queremos obtener.

Ahora sí que entrenaremos la red:

[5]:
```
model.fit(training_data, target_data, epochs=200)
```

```
WARNING:tensorflow:From /home/alberto/.local/lib/python3.6/site-
packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Epoch 1/200
4/4 [==============================] - 0s 23ms/step - loss: 0.2582 -
binary_accuracy: 0.5000
Epoch 2/200
```

```
4/4 [==============================] - 0s 267us/step - loss: 0.2573 -
binary_accuracy: 0.5000
Epoch 3/200
4/4 [==============================] - 0s 254us/step - loss: 0.2563 -
binary_accuracy: 0.5000
Epoch 4/200
4/4 [==============================] - 0s 448us/step - loss: 0.2554 -
binary_accuracy: 0.5000
Epoch 5/200
4/4 [==============================] - 0s 324us/step - loss: 0.2545 -
binary_accuracy: 0.5000
Epoch 6/200
4/4 [==============================] - 0s 367us/step - loss: 0.2536 -
binary_accuracy: 0.5000
Epoch 7/200
4/4 [==============================] - 0s 355us/step - loss: 0.2527 -
binary_accuracy: 0.5000
Epoch 8/200
4/4 [==============================] - 0s 304us/step - loss: 0.2518 -
binary_accuracy: 0.5000
Epoch 9/200
4/4 [==============================] - 0s 403us/step - loss: 0.2509 -
binary_accuracy: 0.5000
Epoch 10/200
4/4 [==============================] - 0s 362us/step - loss: 0.2500 -
binary_accuracy: 0.5000
Epoch 11/200
4/4 [==============================] - 0s 314us/step - loss: 0.2491 -
binary_accuracy: 0.5000
Epoch 12/200
4/4 [==============================] - 0s 347us/step - loss: 0.2483 -
binary_accuracy: 0.5000
Epoch 13/200
4/4 [==============================] - 0s 400us/step - loss: 0.2474 -
binary_accuracy: 0.5000
Epoch 14/200
4/4 [==============================] - 0s 370us/step - loss: 0.2465 -
binary_accuracy: 0.5000
Epoch 15/200
4/4 [==============================] - 0s 347us/step - loss: 0.2457 -
binary_accuracy: 0.5000
Epoch 16/200
4/4 [==============================] - 0s 296us/step - loss: 0.2448 -
binary_accuracy: 0.5000
Epoch 17/200
4/4 [==============================] - 0s 359us/step - loss: 0.2439 -
binary_accuracy: 0.5000
Epoch 18/200
```

```
4/4 [==============================] - 0s 351us/step - loss: 0.2431 -
binary_accuracy: 0.5000
Epoch 19/200
4/4 [==============================] - 0s 359us/step - loss: 0.2423 -
binary_accuracy: 0.5000
Epoch 20/200
4/4 [==============================] - 0s 290us/step - loss: 0.2414 -
binary_accuracy: 0.5000
Epoch 21/200
4/4 [==============================] - 0s 434us/step - loss: 0.2406 -
binary_accuracy: 0.5000
Epoch 22/200
4/4 [==============================] - 0s 255us/step - loss: 0.2398 -
binary_accuracy: 0.5000
Epoch 23/200
4/4 [==============================] - 0s 330us/step - loss: 0.2389 -
binary_accuracy: 0.5000
Epoch 24/200
4/4 [==============================] - 0s 574us/step - loss: 0.2381 -
binary_accuracy: 0.5000
Epoch 25/200
4/4 [==============================] - 0s 232us/step - loss: 0.2374 -
binary_accuracy: 0.5000
Epoch 26/200
4/4 [==============================] - 0s 269us/step - loss: 0.2366 -
binary_accuracy: 0.5000
Epoch 27/200
4/4 [==============================] - 0s 286us/step - loss: 0.2358 -
binary_accuracy: 0.5000
Epoch 28/200
4/4 [==============================] - 0s 312us/step - loss: 0.2351 -
binary_accuracy: 0.5000
Epoch 29/200
4/4 [==============================] - 0s 305us/step - loss: 0.2343 -
binary_accuracy: 0.5000
Epoch 30/200
4/4 [==============================] - 0s 333us/step - loss: 0.2335 -
binary_accuracy: 0.5000
Epoch 31/200
4/4 [==============================] - 0s 239us/step - loss: 0.2328 -
binary_accuracy: 0.5000
Epoch 32/200
4/4 [==============================] - 0s 319us/step - loss: 0.2321 -
binary_accuracy: 0.5000
Epoch 33/200
4/4 [==============================] - 0s 362us/step - loss: 0.2313 -
binary_accuracy: 0.5000
Epoch 34/200
```

```
4/4 [==============================] - 0s 342us/step - loss: 0.2306 -
binary_accuracy: 0.5000
Epoch 35/200
4/4 [==============================] - 0s 365us/step - loss: 0.2299 -
binary_accuracy: 0.5000
Epoch 36/200
4/4 [==============================] - 0s 298us/step - loss: 0.2291 -
binary_accuracy: 0.5000
Epoch 37/200
4/4 [==============================] - 0s 359us/step - loss: 0.2284 -
binary_accuracy: 0.5000
Epoch 38/200
4/4 [==============================] - 0s 257us/step - loss: 0.2277 -
binary_accuracy: 0.5000
Epoch 39/200
4/4 [==============================] - 0s 291us/step - loss: 0.2270 -
binary_accuracy: 0.5000
Epoch 40/200
4/4 [==============================] - 0s 261us/step - loss: 0.2263 -
binary_accuracy: 0.5000
Epoch 41/200
4/4 [==============================] - 0s 275us/step - loss: 0.2256 -
binary_accuracy: 0.5000
Epoch 42/200
4/4 [==============================] - 0s 344us/step - loss: 0.2249 -
binary_accuracy: 0.5000
Epoch 43/200
4/4 [==============================] - 0s 202us/step - loss: 0.2242 -
binary_accuracy: 0.5000
Epoch 44/200
4/4 [==============================] - 0s 205us/step - loss: 0.2235 -
binary_accuracy: 0.5000
Epoch 45/200
4/4 [==============================] - 0s 251us/step - loss: 0.2229 -
binary_accuracy: 0.5000
Epoch 46/200
4/4 [==============================] - 0s 257us/step - loss: 0.2222 -
binary_accuracy: 0.5000
Epoch 47/200
4/4 [==============================] - 0s 250us/step - loss: 0.2215 -
binary_accuracy: 0.5000
Epoch 48/200
4/4 [==============================] - 0s 403us/step - loss: 0.2209 -
binary_accuracy: 0.5000
Epoch 49/200
4/4 [==============================] - 0s 331us/step - loss: 0.2203 -
binary_accuracy: 0.5000
Epoch 50/200
```

```
4/4 [==============================] - 0s 268us/step - loss: 0.2196 -
binary_accuracy: 0.5000
Epoch 51/200
4/4 [==============================] - 0s 370us/step - loss: 0.2190 -
binary_accuracy: 0.5000
Epoch 52/200
4/4 [==============================] - 0s 308us/step - loss: 0.2184 -
binary_accuracy: 0.5000
Epoch 53/200
4/4 [==============================] - 0s 315us/step - loss: 0.2178 -
binary_accuracy: 0.5000
Epoch 54/200
4/4 [==============================] - 0s 412us/step - loss: 0.2172 -
binary_accuracy: 0.5000
Epoch 55/200
4/4 [==============================] - 0s 297us/step - loss: 0.2166 -
binary_accuracy: 0.5000
Epoch 56/200
4/4 [==============================] - 0s 381us/step - loss: 0.2160 -
binary_accuracy: 0.5000
Epoch 57/200
4/4 [==============================] - 0s 307us/step - loss: 0.2154 -
binary_accuracy: 0.5000
Epoch 58/200
4/4 [==============================] - 0s 231us/step - loss: 0.2148 -
binary_accuracy: 0.5000
Epoch 59/200
4/4 [==============================] - 0s 288us/step - loss: 0.2142 -
binary_accuracy: 0.7500
Epoch 60/200
4/4 [==============================] - 0s 320us/step - loss: 0.2136 -
binary_accuracy: 0.7500
Epoch 61/200
4/4 [==============================] - 0s 295us/step - loss: 0.2130 -
binary_accuracy: 0.7500
Epoch 62/200
4/4 [==============================] - 0s 220us/step - loss: 0.2124 -
binary_accuracy: 0.7500
Epoch 63/200
4/4 [==============================] - 0s 286us/step - loss: 0.2119 -
binary_accuracy: 0.7500
Epoch 64/200
4/4 [==============================] - 0s 268us/step - loss: 0.2113 -
binary_accuracy: 0.7500
Epoch 65/200
4/4 [==============================] - 0s 302us/step - loss: 0.2107 -
binary_accuracy: 0.7500
Epoch 66/200
```

```
4/4 [==============================] - 0s 317us/step - loss: 0.2102 -
binary_accuracy: 0.7500
Epoch 67/200
4/4 [==============================] - 0s 309us/step - loss: 0.2096 -
binary_accuracy: 0.7500
Epoch 68/200
4/4 [==============================] - 0s 257us/step - loss: 0.2091 -
binary_accuracy: 0.7500
Epoch 69/200
4/4 [==============================] - 0s 240us/step - loss: 0.2085 -
binary_accuracy: 0.7500
Epoch 70/200
4/4 [==============================] - 0s 224us/step - loss: 0.2080 -
binary_accuracy: 0.7500
Epoch 71/200
4/4 [==============================] - 0s 212us/step - loss: 0.2074 -
binary_accuracy: 0.7500
Epoch 72/200
4/4 [==============================] - 0s 231us/step - loss: 0.2069 -
binary_accuracy: 0.7500
Epoch 73/200
4/4 [==============================] - 0s 288us/step - loss: 0.2064 -
binary_accuracy: 0.7500
Epoch 74/200
4/4 [==============================] - 0s 359us/step - loss: 0.2058 -
binary_accuracy: 0.7500
Epoch 75/200
4/4 [==============================] - 0s 298us/step - loss: 0.2053 -
binary_accuracy: 0.7500
Epoch 76/200
4/4 [==============================] - 0s 463us/step - loss: 0.2048 -
binary_accuracy: 0.7500
Epoch 77/200
4/4 [==============================] - 0s 255us/step - loss: 0.2043 -
binary_accuracy: 0.7500
Epoch 78/200
4/4 [==============================] - 0s 228us/step - loss: 0.2037 -
binary_accuracy: 0.7500
Epoch 79/200
4/4 [==============================] - 0s 275us/step - loss: 0.2032 -
binary_accuracy: 0.7500
Epoch 80/200
4/4 [==============================] - 0s 257us/step - loss: 0.2027 -
binary_accuracy: 1.0000
Epoch 81/200
4/4 [==============================] - 0s 303us/step - loss: 0.2022 -
binary_accuracy: 1.0000
Epoch 82/200
```

```
4/4 [==============================] - 0s 320us/step - loss: 0.2017 -
binary_accuracy: 1.0000
Epoch 83/200
4/4 [==============================] - 0s 252us/step - loss: 0.2011 -
binary_accuracy: 1.0000
Epoch 84/200
4/4 [==============================] - 0s 316us/step - loss: 0.2006 -
binary_accuracy: 1.0000
Epoch 85/200
4/4 [==============================] - 0s 217us/step - loss: 0.2001 -
binary_accuracy: 1.0000
Epoch 86/200
4/4 [==============================] - 0s 240us/step - loss: 0.1996 -
binary_accuracy: 1.0000
Epoch 87/200
4/4 [==============================] - 0s 234us/step - loss: 0.1991 -
binary_accuracy: 1.0000
Epoch 88/200
4/4 [==============================] - 0s 378us/step - loss: 0.1986 -
binary_accuracy: 1.0000
Epoch 89/200
4/4 [==============================] - 0s 384us/step - loss: 0.1981 -
binary_accuracy: 1.0000
Epoch 90/200
4/4 [==============================] - 0s 246us/step - loss: 0.1976 -
binary_accuracy: 1.0000
Epoch 91/200
4/4 [==============================] - 0s 221us/step - loss: 0.1971 -
binary_accuracy: 1.0000
Epoch 92/200
4/4 [==============================] - 0s 241us/step - loss: 0.1966 -
binary_accuracy: 1.0000
Epoch 93/200
4/4 [==============================] - 0s 334us/step - loss: 0.1961 -
binary_accuracy: 1.0000
Epoch 94/200
4/4 [==============================] - 0s 319us/step - loss: 0.1956 -
binary_accuracy: 1.0000
Epoch 95/200
4/4 [==============================] - 0s 307us/step - loss: 0.1951 -
binary_accuracy: 1.0000
Epoch 96/200
4/4 [==============================] - 0s 445us/step - loss: 0.1946 -
binary_accuracy: 1.0000
Epoch 97/200
4/4 [==============================] - 0s 273us/step - loss: 0.1941 -
binary_accuracy: 1.0000
Epoch 98/200
```

```
4/4 [==============================] - 0s 329us/step - loss: 0.1937 -
binary_accuracy: 1.0000
Epoch 99/200
4/4 [==============================] - 0s 232us/step - loss: 0.1932 -
binary_accuracy: 1.0000
Epoch 100/200
4/4 [==============================] - 0s 370us/step - loss: 0.1927 -
binary_accuracy: 1.0000
Epoch 101/200
4/4 [==============================] - 0s 289us/step - loss: 0.1922 -
binary_accuracy: 1.0000
Epoch 102/200
4/4 [==============================] - 0s 226us/step - loss: 0.1917 -
binary_accuracy: 1.0000
Epoch 103/200
4/4 [==============================] - 0s 273us/step - loss: 0.1912 -
binary_accuracy: 1.0000
Epoch 104/200
4/4 [==============================] - 0s 408us/step - loss: 0.1908 -
binary_accuracy: 1.0000
Epoch 105/200
4/4 [==============================] - 0s 265us/step - loss: 0.1903 -
binary_accuracy: 1.0000
Epoch 106/200
4/4 [==============================] - 0s 267us/step - loss: 0.1898 -
binary_accuracy: 1.0000
Epoch 107/200
4/4 [==============================] - 0s 240us/step - loss: 0.1893 -
binary_accuracy: 1.0000
Epoch 108/200
4/4 [==============================] - 0s 218us/step - loss: 0.1889 -
binary_accuracy: 1.0000
Epoch 109/200
4/4 [==============================] - 0s 217us/step - loss: 0.1884 -
binary_accuracy: 1.0000
Epoch 110/200
4/4 [==============================] - 0s 292us/step - loss: 0.1879 -
binary_accuracy: 1.0000
Epoch 111/200
4/4 [==============================] - 0s 214us/step - loss: 0.1874 -
binary_accuracy: 1.0000
Epoch 112/200
4/4 [==============================] - 0s 291us/step - loss: 0.1870 -
binary_accuracy: 1.0000
Epoch 113/200
4/4 [==============================] - 0s 452us/step - loss: 0.1865 -
binary_accuracy: 1.0000
Epoch 114/200
```

```
4/4 [==============================] - 0s 295us/step - loss: 0.1860 -
binary_accuracy: 1.0000
Epoch 115/200
4/4 [==============================] - 0s 266us/step - loss: 0.1855 -
binary_accuracy: 1.0000
Epoch 116/200
4/4 [==============================] - 0s 306us/step - loss: 0.1851 -
binary_accuracy: 1.0000
Epoch 117/200
4/4 [==============================] - 0s 305us/step - loss: 0.1846 -
binary_accuracy: 1.0000
Epoch 118/200
4/4 [==============================] - 0s 252us/step - loss: 0.1841 -
binary_accuracy: 1.0000
Epoch 119/200
4/4 [==============================] - 0s 395us/step - loss: 0.1837 -
binary_accuracy: 1.0000
Epoch 120/200
4/4 [==============================] - 0s 236us/step - loss: 0.1832 -
binary_accuracy: 1.0000
Epoch 121/200
4/4 [==============================] - 0s 323us/step - loss: 0.1828 -
binary_accuracy: 1.0000
Epoch 122/200
4/4 [==============================] - 0s 219us/step - loss: 0.1823 -
binary_accuracy: 1.0000
Epoch 123/200
4/4 [==============================] - 0s 287us/step - loss: 0.1818 -
binary_accuracy: 1.0000
Epoch 124/200
4/4 [==============================] - 0s 260us/step - loss: 0.1814 -
binary_accuracy: 1.0000
Epoch 125/200
4/4 [==============================] - 0s 270us/step - loss: 0.1809 -
binary_accuracy: 1.0000
Epoch 126/200
4/4 [==============================] - 0s 257us/step - loss: 0.1805 -
binary_accuracy: 1.0000
Epoch 127/200
4/4 [==============================] - 0s 202us/step - loss: 0.1800 -
binary_accuracy: 1.0000
Epoch 128/200
4/4 [==============================] - 0s 219us/step - loss: 0.1795 -
binary_accuracy: 1.0000
Epoch 129/200
4/4 [==============================] - 0s 320us/step - loss: 0.1791 -
binary_accuracy: 1.0000
Epoch 130/200
```

```
4/4 [==============================] - 0s 283us/step - loss: 0.1786 -
binary_accuracy: 1.0000
Epoch 131/200
4/4 [==============================] - 0s 306us/step - loss: 0.1782 -
binary_accuracy: 1.0000
Epoch 132/200
4/4 [==============================] - 0s 307us/step - loss: 0.1777 -
binary_accuracy: 1.0000
Epoch 133/200
4/4 [==============================] - 0s 225us/step - loss: 0.1773 -
binary_accuracy: 1.0000
Epoch 134/200
4/4 [==============================] - 0s 343us/step - loss: 0.1768 -
binary_accuracy: 1.0000
Epoch 135/200
4/4 [==============================] - 0s 279us/step - loss: 0.1763 -
binary_accuracy: 1.0000
Epoch 136/200
4/4 [==============================] - 0s 209us/step - loss: 0.1759 -
binary_accuracy: 1.0000
Epoch 137/200
4/4 [==============================] - 0s 204us/step - loss: 0.1754 -
binary_accuracy: 1.0000
Epoch 138/200
4/4 [==============================] - 0s 282us/step - loss: 0.1750 -
binary_accuracy: 1.0000
Epoch 139/200
4/4 [==============================] - 0s 249us/step - loss: 0.1745 -
binary_accuracy: 1.0000
Epoch 140/200
4/4 [==============================] - 0s 441us/step - loss: 0.1741 -
binary_accuracy: 1.0000
Epoch 141/200
4/4 [==============================] - 0s 212us/step - loss: 0.1736 -
binary_accuracy: 1.0000
Epoch 142/200
4/4 [==============================] - 0s 194us/step - loss: 0.1732 -
binary_accuracy: 1.0000
Epoch 143/200
4/4 [==============================] - 0s 184us/step - loss: 0.1727 -
binary_accuracy: 1.0000
Epoch 144/200
4/4 [==============================] - 0s 226us/step - loss: 0.1723 -
binary_accuracy: 1.0000
Epoch 145/200
4/4 [==============================] - 0s 224us/step - loss: 0.1718 -
binary_accuracy: 1.0000
Epoch 146/200
```

```
4/4 [==============================] - 0s 288us/step - loss: 0.1714 -
binary_accuracy: 1.0000
Epoch 147/200
4/4 [==============================] - 0s 298us/step - loss: 0.1709 -
binary_accuracy: 1.0000
Epoch 148/200
4/4 [==============================] - 0s 256us/step - loss: 0.1705 -
binary_accuracy: 1.0000
Epoch 149/200
4/4 [==============================] - 0s 268us/step - loss: 0.1700 -
binary_accuracy: 1.0000
Epoch 150/200
4/4 [==============================] - 0s 343us/step - loss: 0.1696 -
binary_accuracy: 1.0000
Epoch 151/200
4/4 [==============================] - 0s 246us/step - loss: 0.1691 -
binary_accuracy: 1.0000
Epoch 152/200
4/4 [==============================] - 0s 259us/step - loss: 0.1687 -
binary_accuracy: 1.0000
Epoch 153/200
4/4 [==============================] - 0s 444us/step - loss: 0.1683 -
binary_accuracy: 1.0000
Epoch 154/200
4/4 [==============================] - 0s 219us/step - loss: 0.1678 -
binary_accuracy: 1.0000
Epoch 155/200
4/4 [==============================] - 0s 298us/step - loss: 0.1674 -
binary_accuracy: 1.0000
Epoch 156/200
4/4 [==============================] - 0s 284us/step - loss: 0.1669 -
binary_accuracy: 1.0000
Epoch 157/200
4/4 [==============================] - 0s 307us/step - loss: 0.1665 -
binary_accuracy: 1.0000
Epoch 158/200
4/4 [==============================] - 0s 211us/step - loss: 0.1660 -
binary_accuracy: 1.0000
Epoch 159/200
4/4 [==============================] - 0s 534us/step - loss: 0.1656 -
binary_accuracy: 1.0000
Epoch 160/200
4/4 [==============================] - 0s 367us/step - loss: 0.1651 -
binary_accuracy: 1.0000
Epoch 161/200
4/4 [==============================] - 0s 299us/step - loss: 0.1647 -
binary_accuracy: 1.0000
Epoch 162/200
```

```
4/4 [==============================] - 0s 241us/step - loss: 0.1643 -
binary_accuracy: 1.0000
Epoch 163/200
4/4 [==============================] - 0s 211us/step - loss: 0.1638 -
binary_accuracy: 1.0000
Epoch 164/200
4/4 [==============================] - 0s 207us/step - loss: 0.1634 -
binary_accuracy: 1.0000
Epoch 165/200
4/4 [==============================] - 0s 259us/step - loss: 0.1629 -
binary_accuracy: 1.0000
Epoch 166/200
4/4 [==============================] - 0s 243us/step - loss: 0.1625 -
binary_accuracy: 1.0000
Epoch 167/200
4/4 [==============================] - 0s 257us/step - loss: 0.1620 -
binary_accuracy: 1.0000
Epoch 168/200
4/4 [==============================] - 0s 413us/step - loss: 0.1616 -
binary_accuracy: 1.0000
Epoch 169/200
4/4 [==============================] - 0s 266us/step - loss: 0.1612 -
binary_accuracy: 1.0000
Epoch 170/200
4/4 [==============================] - 0s 319us/step - loss: 0.1607 -
binary_accuracy: 1.0000
Epoch 171/200
4/4 [==============================] - 0s 503us/step - loss: 0.1603 -
binary_accuracy: 1.0000
Epoch 172/200
4/4 [==============================] - 0s 273us/step - loss: 0.1598 -
binary_accuracy: 1.0000
Epoch 173/200
4/4 [==============================] - 0s 216us/step - loss: 0.1594 -
binary_accuracy: 1.0000
Epoch 174/200
4/4 [==============================] - 0s 358us/step - loss: 0.1589 -
binary_accuracy: 1.0000
Epoch 175/200
4/4 [==============================] - 0s 270us/step - loss: 0.1585 -
binary_accuracy: 1.0000
Epoch 176/200
4/4 [==============================] - 0s 252us/step - loss: 0.1581 -
binary_accuracy: 1.0000
Epoch 177/200
4/4 [==============================] - 0s 351us/step - loss: 0.1577 -
binary_accuracy: 1.0000
Epoch 178/200
```

```
4/4 [==============================] - 0s 273us/step - loss: 0.1572 -
binary_accuracy: 1.0000
Epoch 179/200
4/4 [==============================] - 0s 212us/step - loss: 0.1568 -
binary_accuracy: 1.0000
Epoch 180/200
4/4 [==============================] - 0s 203us/step - loss: 0.1564 -
binary_accuracy: 1.0000
Epoch 181/200
4/4 [==============================] - 0s 259us/step - loss: 0.1559 -
binary_accuracy: 1.0000
Epoch 182/200
4/4 [==============================] - 0s 296us/step - loss: 0.1555 -
binary_accuracy: 1.0000
Epoch 183/200
4/4 [==============================] - 0s 239us/step - loss: 0.1551 -
binary_accuracy: 1.0000
Epoch 184/200
4/4 [==============================] - 0s 302us/step - loss: 0.1546 -
binary_accuracy: 1.0000
Epoch 185/200
4/4 [==============================] - 0s 259us/step - loss: 0.1542 -
binary_accuracy: 1.0000
Epoch 186/200
4/4 [==============================] - 0s 266us/step - loss: 0.1538 -
binary_accuracy: 1.0000
Epoch 187/200
4/4 [==============================] - 0s 315us/step - loss: 0.1533 -
binary_accuracy: 1.0000
Epoch 188/200
4/4 [==============================] - 0s 189us/step - loss: 0.1529 -
binary_accuracy: 1.0000
Epoch 189/200
4/4 [==============================] - 0s 192us/step - loss: 0.1525 -
binary_accuracy: 1.0000
Epoch 190/200
4/4 [==============================] - 0s 255us/step - loss: 0.1521 -
binary_accuracy: 1.0000
Epoch 191/200
4/4 [==============================] - 0s 218us/step - loss: 0.1516 -
binary_accuracy: 1.0000
Epoch 192/200
4/4 [==============================] - 0s 235us/step - loss: 0.1512 -
binary_accuracy: 1.0000
Epoch 193/200
4/4 [==============================] - 0s 286us/step - loss: 0.1508 -
binary_accuracy: 1.0000
Epoch 194/200
```

```
4/4 [==============================] - 0s 277us/step - loss: 0.1504 -
binary_accuracy: 1.0000
Epoch 195/200
4/4 [==============================] - 0s 217us/step - loss: 0.1500 -
binary_accuracy: 1.0000
Epoch 196/200
4/4 [==============================] - 0s 222us/step - loss: 0.1496 -
binary_accuracy: 1.0000
Epoch 197/200
4/4 [==============================] - 0s 290us/step - loss: 0.1492 -
binary_accuracy: 1.0000
Epoch 198/200
4/4 [==============================] - 0s 260us/step - loss: 0.1488 -
binary_accuracy: 1.0000
Epoch 199/200
4/4 [==============================] - 0s 271us/step - loss: 0.1484 -
binary_accuracy: 1.0000
Epoch 200/200
4/4 [==============================] - 0s 422us/step - loss: 0.1480 -
binary_accuracy: 1.0000
```

[5]:
```
<keras.callbacks.callbacks.History at 0x7fc638b81898>
```

Indicamos con model.fit() las entradas y sus salidas y la cantidad de iteraciones de aprendizaje (epochs) de entrenamiento. Toca evaluar el modelo:

[6]:
```python
scores = model.evaluate(training_data, target_data)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
4/4 [==============================] - 0s 4ms/step

binary_accuracy: 100.00%
```

Y vemos que tuvimos un 100% de precisión (recordemos lo trivial de este ejemplo).

Y hacemos las 4 predicciones posibles de AND, pasando nuestras entradas:

[7]:
```python
print (model.predict(training_data).round())
```

```
[[0.]
 [0.]
 [0.]
 [1.]]
```

y vemos las salidas 0,0,0,1 que son las correctas.

Ejemplo extraído de: https://www.aprendemachinelearning.com/una-sencilla-red-neuronal-en-python-con-keras-y-tensorflow/ y modificado.

1.2 EJEMPLO 2

En este tutorial vamos a utilizar el conjunto de datos sobre el inicio de la diabetes en los indios Pima. Este es una conjunto de datos de machine learning estándar disponible para su descarga gratuita desde el sitio web de Machine Learning de UCI repositorio (https://unipython.com/wp-content/uploads/2018/04/pima-indians-diabetes.csv).

[8]:
```
from keras.models import Sequential
from keras.layers import Dense
import numpy
# Fija las semillas aleatorias para la reproducibilidad
numpy.random.seed(7)
```

Ahora podemos cargar nuestros datos. En este tutorial, vamos a utilizar el conjunto de datos sobre el inicio de la diabetes en los indígenas Pima. Ahora puede cargar el archivo directamente usando la función loadtxt() de NumPy. Hay ocho variables de entrada y una variable de salida (la última columna). Una vez cargado podemos dividir el conjunto de datos en variables de entrada (X) y la variable de clase de salida (Y).

[9]:
```
# carga los datos
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# dividido en variables de entrada (X) y salida (Y)
X = dataset[:,0:8]
Y = dataset[:,8]
```

Hemos inicializado nuestro generador de números aleatorios para asegurarnos de que nuestros resultados sean reproducibles y cargamos nuestros datos. Ahora estamos listos para definir nuestro modelo de red neuronal.

Tenga en cuenta que el conjunto de datos tiene 9 columnas y el rango 0:8 seleccionará las columnas de 0 a 7, deteniéndose antes del índice 8 (ya que éste es el resultado final, el cual es conocido).

[10]:
```
# crea el modelo
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Ahora que el modelo está definido, podemos compilarlo. Debido a que es un problema de clasificación, recopilaremos y reportaremos la exactitud de la clasificación como la métrica. También debemos especificar la función de pérdida a utilizar para evaluar un conjunto de pesos, el optimizador utilizado para buscar a través de diferentes pesos para la red y cualquier métrica opcional que nos gustaría recopilar y reportar durante el entrenamiento.

En este caso, utilizaremos la pérdida logarítmica, que para un problema de clasificación binaria se define en Keras como “binary_crossentropy”. También utilizaremos el algoritmo de descenso de gradiente eficiente “adam” por su alta eficiencia en estos problemas.

[11]:
```
# Compila el modelo
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

```
WARNING:tensorflow:From /home/alberto/.local/lib/python3.6/site-
packages/tensorflow/python/ops/nn_impl.py:180:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

El proceso de entrenamiento se ejecutará para un número fijo de iteraciones denominado epochs o épocas. También podemos establecer el número de instancias que se evalúan antes de que se realice una actualización de peso en la red llamada batch_size y establecerlo mediante el argumento batch_size. Para este problema utilizaremos un pequeño número de epochs (150) y un batch_size relativamente pequeño (10). Una vez más, estos pueden ser elegidos experimentalmente por ensayo y error.

[12]:
```
# Ajusta el modelo
model.fit(X, Y, epochs=150, batch_size=10)
```

```
Epoch 1/150
768/768 [==============================] - 0s 227us/step - loss: 3.1753 -
accuracy: 0.5820
Epoch 2/150
768/768 [==============================] - 0s 81us/step - loss: 0.9556 -
accuracy: 0.5716
Epoch 3/150
768/768 [==============================] - 0s 91us/step - loss: 0.7623 -
accuracy: 0.6315
Epoch 4/150
768/768 [==============================] - 0s 96us/step - loss: 0.7212 -
accuracy: 0.6471
Epoch 5/150
768/768 [==============================] - 0s 93us/step - loss: 0.6940 -
accuracy: 0.6745
Epoch 6/150
768/768 [==============================] - 0s 92us/step - loss: 0.6683 -
accuracy: 0.6862
Epoch 7/150
768/768 [==============================] - 0s 92us/step - loss: 0.6626 -
accuracy: 0.6758
Epoch 8/150
768/768 [==============================] - 0s 92us/step - loss: 0.6480 -
accuracy: 0.6862
Epoch 9/150
768/768 [==============================] - 0s 90us/step - loss: 0.6347 -
accuracy: 0.6992
```

```
Epoch 10/150
768/768 [==============================] - 0s 91us/step - loss: 0.6432 -
accuracy: 0.6810
Epoch 11/150
768/768 [==============================] - 0s 92us/step - loss: 0.6572 -
accuracy: 0.6706
Epoch 12/150
768/768 [==============================] - 0s 90us/step - loss: 0.6491 -
accuracy: 0.6706
Epoch 13/150
768/768 [==============================] - 0s 91us/step - loss: 0.6331 -
accuracy: 0.6797
Epoch 14/150
768/768 [==============================] - 0s 95us/step - loss: 0.6241 -
accuracy: 0.6979
Epoch 15/150
768/768 [==============================] - 0s 93us/step - loss: 0.6051 -
accuracy: 0.7031
Epoch 16/150
768/768 [==============================] - 0s 91us/step - loss: 0.5895 -
accuracy: 0.7044
Epoch 17/150
768/768 [==============================] - 0s 92us/step - loss: 0.5850 -
accuracy: 0.7005
Epoch 18/150
768/768 [==============================] - 0s 91us/step - loss: 0.5933 -
accuracy: 0.6888
Epoch 19/150
768/768 [==============================] - 0s 90us/step - loss: 0.5774 -
accuracy: 0.7135
Epoch 20/150
768/768 [==============================] - 0s 91us/step - loss: 0.5763 -
accuracy: 0.7292
Epoch 21/150
768/768 [==============================] - 0s 92us/step - loss: 0.5668 -
accuracy: 0.7161
Epoch 22/150
768/768 [==============================] - 0s 93us/step - loss: 0.5788 -
accuracy: 0.7018
Epoch 23/150
768/768 [==============================] - 0s 91us/step - loss: 0.5730 -
accuracy: 0.7083
Epoch 24/150
768/768 [==============================] - 0s 91us/step - loss: 0.5681 -
accuracy: 0.7292
Epoch 25/150
768/768 [==============================] - 0s 96us/step - loss: 0.5559 -
accuracy: 0.7435
```

```
Epoch 26/150
768/768 [==============================] - 0s 104us/step - loss: 0.5730 -
accuracy: 0.7174
Epoch 27/150
768/768 [==============================] - 0s 97us/step - loss: 0.5566 -
accuracy: 0.7240
Epoch 28/150
768/768 [==============================] - 0s 98us/step - loss: 0.5572 -
accuracy: 0.7240
Epoch 29/150
768/768 [==============================] - 0s 96us/step - loss: 0.5786 -
accuracy: 0.7135
Epoch 30/150
768/768 [==============================] - 0s 93us/step - loss: 0.5623 -
accuracy: 0.7174
Epoch 31/150
768/768 [==============================] - 0s 93us/step - loss: 0.5709 -
accuracy: 0.7135
Epoch 32/150
768/768 [==============================] - 0s 93us/step - loss: 0.5679 -
accuracy: 0.7174
Epoch 33/150
768/768 [==============================] - 0s 97us/step - loss: 0.5555 -
accuracy: 0.7148
Epoch 34/150
768/768 [==============================] - 0s 94us/step - loss: 0.5534 -
accuracy: 0.7305
Epoch 35/150
768/768 [==============================] - 0s 99us/step - loss: 0.5550 -
accuracy: 0.7161
Epoch 36/150
768/768 [==============================] - 0s 96us/step - loss: 0.5644 -
accuracy: 0.7174
Epoch 37/150
768/768 [==============================] - 0s 96us/step - loss: 0.5351 -
accuracy: 0.7318
Epoch 38/150
768/768 [==============================] - 0s 94us/step - loss: 0.5464 -
accuracy: 0.7161
Epoch 39/150
768/768 [==============================] - 0s 97us/step - loss: 0.5521 -
accuracy: 0.7227
Epoch 40/150
768/768 [==============================] - 0s 96us/step - loss: 0.5499 -
accuracy: 0.7148
Epoch 41/150
768/768 [==============================] - 0s 97us/step - loss: 0.5493 -
accuracy: 0.7279
```

```
Epoch 42/150
768/768 [==============================] - 0s 98us/step - loss: 0.5413 -
accuracy: 0.7279
Epoch 43/150
768/768 [==============================] - 0s 98us/step - loss: 0.5359 -
accuracy: 0.7383
Epoch 44/150
768/768 [==============================] - 0s 100us/step - loss: 0.5397 -
accuracy: 0.7383
Epoch 45/150
768/768 [==============================] - 0s 97us/step - loss: 0.5376 -
accuracy: 0.7591
Epoch 46/150
768/768 [==============================] - 0s 92us/step - loss: 0.5314 -
accuracy: 0.7448
Epoch 47/150
768/768 [==============================] - 0s 95us/step - loss: 0.5397 -
accuracy: 0.7331
Epoch 48/150
768/768 [==============================] - 0s 93us/step - loss: 0.5409 -
accuracy: 0.7344
Epoch 49/150
768/768 [==============================] - 0s 95us/step - loss: 0.5405 -
accuracy: 0.7409
Epoch 50/150
768/768 [==============================] - 0s 96us/step - loss: 0.5326 -
accuracy: 0.7331
Epoch 51/150
768/768 [==============================] - 0s 99us/step - loss: 0.5344 -
accuracy: 0.7396
Epoch 52/150
768/768 [==============================] - 0s 96us/step - loss: 0.5380 -
accuracy: 0.7331
Epoch 53/150
768/768 [==============================] - 0s 95us/step - loss: 0.5436 -
accuracy: 0.7370
Epoch 54/150
768/768 [==============================] - 0s 94us/step - loss: 0.5463 -
accuracy: 0.7292
Epoch 55/150
768/768 [==============================] - 0s 96us/step - loss: 0.5272 -
accuracy: 0.7409
Epoch 56/150
768/768 [==============================] - 0s 89us/step - loss: 0.5360 -
accuracy: 0.7422
Epoch 57/150
768/768 [==============================] - 0s 89us/step - loss: 0.5391 -
accuracy: 0.7383
```

```
Epoch 58/150
768/768 [==============================] - 0s 88us/step - loss: 0.5302 -
accuracy: 0.7487
Epoch 59/150
768/768 [==============================] - 0s 89us/step - loss: 0.5189 -
accuracy: 0.7500
Epoch 60/150
768/768 [==============================] - 0s 89us/step - loss: 0.5426 -
accuracy: 0.7331
Epoch 61/150
768/768 [==============================] - 0s 86us/step - loss: 0.5320 -
accuracy: 0.7253
Epoch 62/150
768/768 [==============================] - 0s 89us/step - loss: 0.5175 -
accuracy: 0.7435
Epoch 63/150
768/768 [==============================] - 0s 89us/step - loss: 0.5467 -
accuracy: 0.7318
Epoch 64/150
768/768 [==============================] - 0s 88us/step - loss: 0.5380 -
accuracy: 0.7305
Epoch 65/150
768/768 [==============================] - 0s 80us/step - loss: 0.5229 -
accuracy: 0.7474
Epoch 66/150
768/768 [==============================] - 0s 87us/step - loss: 0.5085 -
accuracy: 0.7474
Epoch 67/150
768/768 [==============================] - 0s 98us/step - loss: 0.5187 -
accuracy: 0.7331
Epoch 68/150
768/768 [==============================] - 0s 95us/step - loss: 0.5166 -
accuracy: 0.7500
Epoch 69/150
768/768 [==============================] - 0s 92us/step - loss: 0.5185 -
accuracy: 0.7552
Epoch 70/150
768/768 [==============================] - 0s 91us/step - loss: 0.5365 -
accuracy: 0.7214
Epoch 71/150
768/768 [==============================] - 0s 90us/step - loss: 0.5222 -
accuracy: 0.7409
Epoch 72/150
768/768 [==============================] - 0s 89us/step - loss: 0.5190 -
accuracy: 0.7500
Epoch 73/150
768/768 [==============================] - 0s 90us/step - loss: 0.5186 -
accuracy: 0.7526
```

```
Epoch 74/150
768/768 [==============================] - 0s 89us/step - loss: 0.5133 -
accuracy: 0.7617
Epoch 75/150
768/768 [==============================] - 0s 89us/step - loss: 0.5142 -
accuracy: 0.7539
Epoch 76/150
768/768 [==============================] - 0s 90us/step - loss: 0.5153 -
accuracy: 0.7448
Epoch 77/150
768/768 [==============================] - 0s 88us/step - loss: 0.5225 -
accuracy: 0.7591
Epoch 78/150
768/768 [==============================] - 0s 93us/step - loss: 0.5190 -
accuracy: 0.7526
Epoch 79/150
768/768 [==============================] - 0s 93us/step - loss: 0.5174 -
accuracy: 0.7461
Epoch 80/150
768/768 [==============================] - 0s 93us/step - loss: 0.5104 -
accuracy: 0.7591
Epoch 81/150
768/768 [==============================] - 0s 95us/step - loss: 0.5112 -
accuracy: 0.7617
Epoch 82/150
768/768 [==============================] - 0s 93us/step - loss: 0.5047 -
accuracy: 0.7500
Epoch 83/150
768/768 [==============================] - 0s 92us/step - loss: 0.5027 -
accuracy: 0.7461
Epoch 84/150
768/768 [==============================] - 0s 90us/step - loss: 0.5000 -
accuracy: 0.7617
Epoch 85/150
768/768 [==============================] - 0s 93us/step - loss: 0.5079 -
accuracy: 0.7487
Epoch 86/150
768/768 [==============================] - 0s 91us/step - loss: 0.5104 -
accuracy: 0.7461
Epoch 87/150
768/768 [==============================] - 0s 94us/step - loss: 0.5019 -
accuracy: 0.7591
Epoch 88/150
768/768 [==============================] - 0s 97us/step - loss: 0.5039 -
accuracy: 0.7604
Epoch 89/150
768/768 [==============================] - 0s 97us/step - loss: 0.5104 -
accuracy: 0.7552
```

```
Epoch 90/150
768/768 [==============================] - 0s 90us/step - loss: 0.5107 -
accuracy: 0.7526
Epoch 91/150
768/768 [==============================] - 0s 88us/step - loss: 0.5026 -
accuracy: 0.7448
Epoch 92/150
768/768 [==============================] - 0s 91us/step - loss: 0.5084 -
accuracy: 0.7422
Epoch 93/150
768/768 [==============================] - 0s 89us/step - loss: 0.5003 -
accuracy: 0.7578
Epoch 94/150
768/768 [==============================] - 0s 89us/step - loss: 0.5002 -
accuracy: 0.7656
Epoch 95/150
768/768 [==============================] - 0s 91us/step - loss: 0.5096 -
accuracy: 0.7474
Epoch 96/150
768/768 [==============================] - 0s 98us/step - loss: 0.4965 -
accuracy: 0.7565
Epoch 97/150
768/768 [==============================] - 0s 94us/step - loss: 0.4982 -
accuracy: 0.7773
Epoch 98/150
768/768 [==============================] - 0s 94us/step - loss: 0.4917 -
accuracy: 0.7591
Epoch 99/150
768/768 [==============================] - 0s 93us/step - loss: 0.4921 -
accuracy: 0.7643
Epoch 100/150
768/768 [==============================] - 0s 93us/step - loss: 0.4887 -
accuracy: 0.7734
Epoch 101/150
768/768 [==============================] - 0s 87us/step - loss: 0.4918 -
accuracy: 0.7682
Epoch 102/150
768/768 [==============================] - 0s 94us/step - loss: 0.5017 -
accuracy: 0.7500
Epoch 103/150
768/768 [==============================] - 0s 91us/step - loss: 0.5020 -
accuracy: 0.7500
Epoch 104/150
768/768 [==============================] - 0s 91us/step - loss: 0.4948 -
accuracy: 0.7721
Epoch 105/150
768/768 [==============================] - 0s 90us/step - loss: 0.5301 -
accuracy: 0.7422
```

```
Epoch 106/150
768/768 [==============================] - 0s 91us/step - loss: 0.4952 -
accuracy: 0.7695
Epoch 107/150
768/768 [==============================] - 0s 94us/step - loss: 0.4929 -
accuracy: 0.7812
Epoch 108/150
768/768 [==============================] - 0s 94us/step - loss: 0.5057 -
accuracy: 0.7539
Epoch 109/150
768/768 [==============================] - 0s 91us/step - loss: 0.4892 -
accuracy: 0.7591
Epoch 110/150
768/768 [==============================] - 0s 91us/step - loss: 0.4903 -
accuracy: 0.7591
Epoch 111/150
768/768 [==============================] - 0s 90us/step - loss: 0.4855 -
accuracy: 0.7721
Epoch 112/150
768/768 [==============================] - 0s 91us/step - loss: 0.4939 -
accuracy: 0.7773
Epoch 113/150
768/768 [==============================] - 0s 89us/step - loss: 0.5050 -
accuracy: 0.7604
Epoch 114/150
768/768 [==============================] - 0s 93us/step - loss: 0.4922 -
accuracy: 0.7513
Epoch 115/150
768/768 [==============================] - 0s 96us/step - loss: 0.4937 -
accuracy: 0.7656
Epoch 116/150
768/768 [==============================] - 0s 94us/step - loss: 0.4911 -
accuracy: 0.7747
Epoch 117/150
768/768 [==============================] - 0s 95us/step - loss: 0.4941 -
accuracy: 0.7630
Epoch 118/150
768/768 [==============================] - 0s 93us/step - loss: 0.4921 -
accuracy: 0.7695
Epoch 119/150
768/768 [==============================] - 0s 91us/step - loss: 0.4892 -
accuracy: 0.7630
Epoch 120/150
768/768 [==============================] - 0s 90us/step - loss: 0.4985 -
accuracy: 0.7682
Epoch 121/150
768/768 [==============================] - 0s 91us/step - loss: 0.4963 -
accuracy: 0.7773
```

```
Epoch 122/150
768/768 [==============================] - 0s 99us/step - loss: 0.4823 -
accuracy: 0.7747
Epoch 123/150
768/768 [==============================] - 0s 99us/step - loss: 0.4884 -
accuracy: 0.7617
Epoch 124/150
768/768 [==============================] - 0s 91us/step - loss: 0.4834 -
accuracy: 0.7786
Epoch 125/150
768/768 [==============================] - 0s 94us/step - loss: 0.4841 -
accuracy: 0.7878
Epoch 126/150
768/768 [==============================] - 0s 91us/step - loss: 0.4822 -
accuracy: 0.7617
Epoch 127/150
768/768 [==============================] - 0s 91us/step - loss: 0.4894 -
accuracy: 0.7643
Epoch 128/150
768/768 [==============================] - 0s 92us/step - loss: 0.4724 -
accuracy: 0.7760
Epoch 129/150
768/768 [==============================] - 0s 94us/step - loss: 0.4811 -
accuracy: 0.7643
Epoch 130/150
768/768 [==============================] - 0s 91us/step - loss: 0.4700 -
accuracy: 0.7891
Epoch 131/150
768/768 [==============================] - 0s 90us/step - loss: 0.4806 -
accuracy: 0.7630
Epoch 132/150
768/768 [==============================] - 0s 88us/step - loss: 0.4813 -
accuracy: 0.7747
Epoch 133/150
768/768 [==============================] - 0s 90us/step - loss: 0.4847 -
accuracy: 0.7669
Epoch 134/150
768/768 [==============================] - 0s 92us/step - loss: 0.4838 -
accuracy: 0.7708
Epoch 135/150
768/768 [==============================] - 0s 91us/step - loss: 0.4740 -
accuracy: 0.7708
Epoch 136/150
768/768 [==============================] - 0s 92us/step - loss: 0.4721 -
accuracy: 0.7721
Epoch 137/150
768/768 [==============================] - 0s 91us/step - loss: 0.4678 -
accuracy: 0.7839
```

```
Epoch 138/150
768/768 [==============================] - 0s 89us/step - loss: 0.4778 -
accuracy: 0.7839
Epoch 139/150
768/768 [==============================] - 0s 93us/step - loss: 0.4704 -
accuracy: 0.7669
Epoch 140/150
768/768 [==============================] - 0s 90us/step - loss: 0.4792 -
accuracy: 0.7826
Epoch 141/150
768/768 [==============================] - 0s 91us/step - loss: 0.4710 -
accuracy: 0.7917
Epoch 142/150
768/768 [==============================] - 0s 89us/step - loss: 0.4810 -
accuracy: 0.7656
Epoch 143/150
768/768 [==============================] - 0s 89us/step - loss: 0.4729 -
accuracy: 0.7747
Epoch 144/150
768/768 [==============================] - 0s 88us/step - loss: 0.4719 -
accuracy: 0.7760
Epoch 145/150
768/768 [==============================] - 0s 91us/step - loss: 0.4929 -
accuracy: 0.7565
Epoch 146/150
768/768 [==============================] - 0s 92us/step - loss: 0.4917 -
accuracy: 0.7656
Epoch 147/150
768/768 [==============================] - 0s 90us/step - loss: 0.4861 -
accuracy: 0.7760
Epoch 148/150
768/768 [==============================] - 0s 93us/step - loss: 0.4723 -
accuracy: 0.7812
Epoch 149/150
768/768 [==============================] - 0s 91us/step - loss: 0.4720 -
accuracy: 0.7578
Epoch 150/150
768/768 [==============================] - 0s 92us/step - loss: 0.4738 -
accuracy: 0.7839
```

```
[12]: <keras.callbacks.callbacks.History at 0x7fc630263cc0>
```

Hemos entrenado nuestra red neuronal en todo el conjunto de datos y podemos evaluar el rendimiento de la red en el mismo conjunto de datos. Esto sólo nos dará una idea de lo bien que hemos modelado el conjunto de datos, pero no nos dará una idea de lo bien que el algoritmo podría funcionar con los nuevos datos.

[15]:
```python
# evalua el modelo
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
768/768 [==============================] - 0s 17us/step

accuracy: 76.95%
```

[14]:
```python
# calcula las predicciones
predictions = model.predict(X)
# redondeamos las predicciones
rounded = [round(x[0]) for x in predictions]
print(rounded)
```

```
[1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0,
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0,
0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0,
1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0,
0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0,
1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0,
0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0,
0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0,
1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0,
1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0,
0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0,
0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0,
1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0,
0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0,
```

```
0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0,
1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0,
1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0,
1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0,
0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0,
0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0,
0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0]
```

Estos 1 o 0 que han salido por consola definen si el usuario tiene diabetes o no y lo podemos comparar con la última columna que tenemos del excel para ver si nuestra predicción es buena.

Ejemplo extraído de: https://unipython.com/desarrolla-primera-red-neural-python-keras-paso-paso/