ActividadSesion11

May 21, 2020

1 Ejemplo Keras y Tensorflow

1.1 Ejemplo 1

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

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

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

- NAND(0,0) = 1
- NAND(0,1) = 1
- NAND(1,0) = 1
- NAND(1,1) = 0

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 NAND
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([[1],[1],[1],[0]], "float32")
```

Como se puede ver son las cuatro entradas posibles de la función NAND [0,0], [0,1], [1,0],[1,1] y sus cuatro salidas: 1, 1,1,0. 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 NAND) 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=100)
```

```
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 1/100
4/4 [==============================] - 0s 21ms/step - loss: 0.2455 -
binary_accuracy: 0.2500
Epoch 2/100
```

```
4/4 [==============================] - 0s 270us/step - loss: 0.2448 -
binary_accuracy: 0.5000
Epoch 3/100
4/4 [==============================] - 0s 226us/step - loss: 0.2442 -
binary_accuracy: 0.5000
Epoch 4/100
4/4 [==============================] - 0s 326us/step - loss: 0.2435 -
binary_accuracy: 0.5000
Epoch 5/100
4/4 [==============================] - 0s 232us/step - loss: 0.2429 -
binary_accuracy: 0.5000
Epoch 6/100
4/4 [==============================] - 0s 306us/step - loss: 0.2422 -
binary_accuracy: 0.5000
Epoch 7/100
4/4 [==============================] - 0s 263us/step - loss: 0.2416 -
binary_accuracy: 0.5000
Epoch 8/100
4/4 [==============================] - 0s 439us/step - loss: 0.2410 -
binary_accuracy: 0.5000
Epoch 9/100
4/4 [==============================] - 0s 351us/step - loss: 0.2404 -
binary_accuracy: 0.5000
Epoch 10/100
4/4 [==============================] - 0s 378us/step - loss: 0.2398 -
binary_accuracy: 0.5000
Epoch 11/100
4/4 [==============================] - 0s 427us/step - loss: 0.2392 -
binary_accuracy: 0.7500
Epoch 12/100
4/4 [==============================] - 0s 329us/step - loss: 0.2387 -
binary_accuracy: 0.7500
Epoch 13/100
4/4 [==============================] - 0s 300us/step - loss: 0.2381 -
binary_accuracy: 0.7500
Epoch 14/100
4/4 [==============================] - 0s 300us/step - loss: 0.2376 -
binary_accuracy: 0.7500
Epoch 15/100
4/4 [==============================] - 0s 299us/step - loss: 0.2370 -
binary_accuracy: 0.7500
Epoch 16/100
4/4 [==============================] - 0s 281us/step - loss: 0.2365 -
binary_accuracy: 0.7500
Epoch 17/100
4/4 [==============================] - 0s 367us/step - loss: 0.2359 -
binary_accuracy: 0.7500
Epoch 18/100
```

```
4/4 [==============================] - 0s 416us/step - loss: 0.2354 -
binary_accuracy: 0.7500
Epoch 19/100
4/4 [==============================] - 0s 299us/step - loss: 0.2349 -
binary_accuracy: 0.7500
Epoch 20/100
4/4 [==============================] - 0s 218us/step - loss: 0.2343 -
binary_accuracy: 0.7500
Epoch 21/100
4/4 [==============================] - 0s 344us/step - loss: 0.2338 -
binary_accuracy: 0.7500
Epoch 22/100
4/4 [==============================] - 0s 340us/step - loss: 0.2333 -
binary_accuracy: 0.7500
Epoch 23/100
4/4 [==============================] - 0s 261us/step - loss: 0.2328 -
binary_accuracy: 0.7500
Epoch 24/100
4/4 [==============================] - 0s 228us/step - loss: 0.2323 -
binary_accuracy: 0.7500
Epoch 25/100
4/4 [==============================] - 0s 209us/step - loss: 0.2318 -
binary_accuracy: 0.7500
Epoch 26/100
4/4 [==============================] - 0s 202us/step - loss: 0.2314 -
binary_accuracy: 0.7500
Epoch 27/100
4/4 [==============================] - 0s 315us/step - loss: 0.2309 -
binary_accuracy: 0.7500
Epoch 28/100
4/4 [==============================] - 0s 272us/step - loss: 0.2304 -
binary_accuracy: 0.7500
Epoch 29/100
4/4 [==============================] - 0s 292us/step - loss: 0.2299 -
binary_accuracy: 0.7500
Epoch 30/100
4/4 [==============================] - 0s 223us/step - loss: 0.2295 -
binary_accuracy: 0.7500
Epoch 31/100
4/4 [==============================] - 0s 214us/step - loss: 0.2290 -
binary_accuracy: 0.7500
Epoch 32/100
4/4 [==============================] - 0s 202us/step - loss: 0.2286 -
binary_accuracy: 0.7500
Epoch 33/100
4/4 [==============================] - 0s 199us/step - loss: 0.2281 -
binary_accuracy: 0.7500
Epoch 34/100
```

```
4/4 [==============================] - 0s 226us/step - loss: 0.2277 -
binary_accuracy: 0.7500
Epoch 35/100
4/4 [==============================] - 0s 229us/step - loss: 0.2273 -
binary_accuracy: 0.7500
Epoch 36/100
4/4 [==============================] - 0s 287us/step - loss: 0.2269 -
binary_accuracy: 0.7500
Epoch 37/100
4/4 [==============================] - 0s 284us/step - loss: 0.2265 -
binary_accuracy: 0.7500
Epoch 38/100
4/4 [==============================] - 0s 274us/step - loss: 0.2261 -
binary_accuracy: 0.7500
Epoch 39/100
4/4 [==============================] - 0s 321us/step - loss: 0.2257 -
binary_accuracy: 0.7500
Epoch 40/100
4/4 [==============================] - 0s 293us/step - loss: 0.2254 -
binary_accuracy: 0.7500
Epoch 41/100
4/4 [==============================] - 0s 279us/step - loss: 0.2250 -
binary_accuracy: 0.7500
Epoch 42/100
4/4 [==============================] - 0s 232us/step - loss: 0.2246 -
binary_accuracy: 0.7500
Epoch 43/100
4/4 [==============================] - 0s 277us/step - loss: 0.2243 -
binary_accuracy: 0.7500
Epoch 44/100
4/4 [==============================] - 0s 308us/step - loss: 0.2239 -
binary_accuracy: 0.7500
Epoch 45/100
4/4 [==============================] - 0s 214us/step - loss: 0.2235 -
binary_accuracy: 0.7500
Epoch 46/100
4/4 [==============================] - 0s 273us/step - loss: 0.2232 -
binary_accuracy: 1.0000
Epoch 47/100
4/4 [==============================] - 0s 276us/step - loss: 0.2228 -
binary_accuracy: 1.0000
Epoch 48/100
4/4 [==============================] - 0s 244us/step - loss: 0.2224 -
binary_accuracy: 1.0000
Epoch 49/100
4/4 [==============================] - 0s 833us/step - loss: 0.2221 -
binary_accuracy: 1.0000
Epoch 50/100
```

```
4/4 [==============================] - 0s 263us/step - loss: 0.2217 -
binary_accuracy: 1.0000
Epoch 51/100
4/4 [==============================] - 0s 209us/step - loss: 0.2214 -
binary_accuracy: 1.0000
Epoch 52/100
4/4 [==============================] - 0s 197us/step - loss: 0.2210 -
binary_accuracy: 1.0000
Epoch 53/100
4/4 [==============================] - 0s 235us/step - loss: 0.2206 -
binary_accuracy: 1.0000
Epoch 54/100
4/4 [==============================] - 0s 293us/step - loss: 0.2203 -
binary_accuracy: 1.0000
Epoch 55/100
4/4 [==============================] - 0s 438us/step - loss: 0.2199 -
binary_accuracy: 1.0000
Epoch 56/100
4/4 [==============================] - 0s 298us/step - loss: 0.2196 -
binary_accuracy: 1.0000
Epoch 57/100
4/4 [==============================] - 0s 335us/step - loss: 0.2192 -
binary_accuracy: 1.0000
Epoch 58/100
4/4 [==============================] - 0s 251us/step - loss: 0.2188 -
binary_accuracy: 1.0000
Epoch 59/100
4/4 [==============================] - 0s 339us/step - loss: 0.2185 -
binary_accuracy: 1.0000
Epoch 60/100
4/4 [==============================] - 0s 360us/step - loss: 0.2181 -
binary_accuracy: 1.0000
Epoch 61/100
4/4 [==============================] - 0s 293us/step - loss: 0.2178 -
binary_accuracy: 1.0000
Epoch 62/100
4/4 [==============================] - 0s 257us/step - loss: 0.2174 -
binary_accuracy: 1.0000
Epoch 63/100
4/4 [==============================] - 0s 244us/step - loss: 0.2170 -
binary_accuracy: 1.0000
Epoch 64/100
4/4 [==============================] - 0s 311us/step - loss: 0.2167 -
binary_accuracy: 1.0000
Epoch 65/100
4/4 [==============================] - 0s 209us/step - loss: 0.2163 -
binary_accuracy: 1.0000
Epoch 66/100
```

```
4/4 [==============================] - 0s 202us/step - loss: 0.2160 -
binary_accuracy: 1.0000
Epoch 67/100
4/4 [==============================] - 0s 288us/step - loss: 0.2156 -
binary_accuracy: 1.0000
Epoch 68/100
4/4 [==============================] - 0s 351us/step - loss: 0.2152 -
binary_accuracy: 1.0000
Epoch 69/100
4/4 [==============================] - 0s 247us/step - loss: 0.2149 -
binary_accuracy: 1.0000
Epoch 70/100
4/4 [==============================] - 0s 342us/step - loss: 0.2145 -
binary_accuracy: 1.0000
Epoch 71/100
4/4 [==============================] - 0s 403us/step - loss: 0.2141 -
binary_accuracy: 1.0000
Epoch 72/100
4/4 [==============================] - 0s 261us/step - loss: 0.2138 -
binary_accuracy: 1.0000
Epoch 73/100
4/4 [==============================] - 0s 225us/step - loss: 0.2134 -
binary_accuracy: 1.0000
Epoch 74/100
4/4 [==============================] - 0s 367us/step - loss: 0.2130 -
binary_accuracy: 1.0000
Epoch 75/100
4/4 [==============================] - 0s 333us/step - loss: 0.2127 -
binary_accuracy: 1.0000
Epoch 76/100
4/4 [==============================] - 0s 271us/step - loss: 0.2123 -
binary_accuracy: 1.0000
Epoch 77/100
4/4 [==============================] - 0s 282us/step - loss: 0.2119 -
binary_accuracy: 1.0000
Epoch 78/100
4/4 [==============================] - 0s 237us/step - loss: 0.2115 -
binary_accuracy: 1.0000
Epoch 79/100
4/4 [==============================] - 0s 211us/step - loss: 0.2112 -
binary_accuracy: 1.0000
Epoch 80/100
4/4 [==============================] - 0s 288us/step - loss: 0.2108 -
binary_accuracy: 1.0000
Epoch 81/100
4/4 [==============================] - 0s 243us/step - loss: 0.2104 -
binary_accuracy: 1.0000
Epoch 82/100
```

```
4/4 [==============================] - 0s 222us/step - loss: 0.2100 -
binary_accuracy: 1.0000
Epoch 83/100
4/4 [==============================] - 0s 236us/step - loss: 0.2096 -
binary_accuracy: 1.0000
Epoch 84/100
4/4 [==============================] - 0s 227us/step - loss: 0.2093 -
binary_accuracy: 1.0000
Epoch 85/100
4/4 [==============================] - 0s 217us/step - loss: 0.2089 -
binary_accuracy: 1.0000
Epoch 86/100
4/4 [==============================] - 0s 312us/step - loss: 0.2085 -
binary_accuracy: 1.0000
Epoch 87/100
4/4 [==============================] - 0s 350us/step - loss: 0.2081 -
binary_accuracy: 1.0000
Epoch 88/100
4/4 [==============================] - 0s 330us/step - loss: 0.2077 -
binary_accuracy: 1.0000
Epoch 89/100
4/4 [==============================] - 0s 221us/step - loss: 0.2073 -
binary_accuracy: 1.0000
Epoch 90/100
4/4 [==============================] - 0s 249us/step - loss: 0.2069 -
binary_accuracy: 1.0000
Epoch 91/100
4/4 [==============================] - 0s 231us/step - loss: 0.2066 -
binary_accuracy: 1.0000
Epoch 92/100
4/4 [==============================] - 0s 248us/step - loss: 0.2062 -
binary_accuracy: 1.0000
Epoch 93/100
4/4 [==============================] - 0s 243us/step - loss: 0.2058 -
binary_accuracy: 1.0000
Epoch 94/100
4/4 [==============================] - 0s 205us/step - loss: 0.2054 -
binary_accuracy: 1.0000
Epoch 95/100
4/4 [==============================] - 0s 255us/step - loss: 0.2050 -
binary_accuracy: 1.0000
Epoch 96/100
4/4 [==============================] - 0s 274us/step - loss: 0.2046 -
binary_accuracy: 1.0000
Epoch 97/100
4/4 [==============================] - 0s 289us/step - loss: 0.2042 -
binary_accuracy: 1.0000
Epoch 98/100
```

```
4/4 [==============================] - 0s 239us/step - loss: 0.2038 -
binary_accuracy: 1.0000
Epoch 99/100
4/4 [==============================] - 0s 220us/step - loss: 0.2034 -
binary_accuracy: 1.0000
Epoch 100/100
4/4 [==============================] - 0s 336us/step - loss: 0.2030 -
binary_accuracy: 1.0000
```

[5]:
```
<keras.callbacks.callbacks.History at 0x7f14b1f66ef0>
```

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 NAND, pasando nuestras entradas:

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

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

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

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