Redes neuronales y deep learning

Actividad Semana 2

Daniel González
Semana 2: Frameworks de Deep Learning: TensorFlow
IEBS

```
import tensorflow as tf
import numpy as np

# Para mostrar gráficas
import matplotlib.pyplot as plt
%matplotlib inline

# Anaconda fixing problem
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

En esta actividad vamos a seguir familiarizándonos con la herramienta *TensorFlow*, para ello vamos a utilizar un dataset donde tendréis que relizar todos los pasos (cargar datos, crear arquitectura de la red, etc.) vosotros mismos para prácticar con la sintásis de *TensorFlow*. Tendéis ejercicios obligatorio y un ejercicio opcional.

Ejercicio

Usando los datos visto en los ejercicios sobre casa y sus precios realiza los ejercicios que se indican.

La información de los datos podéis verla en este enlace:

https://www.tensorflow.org/api_docs/python/tf/keras/datasets/boston_housing/load_data

Ejercicio 1

Carga los datos como hemos visto en el notebook de los ejercicios donde tengas los conjuntos de train y test:

```
# Complet
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

Explora los tamaños de cada conjunto:

```
# Completar
# Importa TensorFlow y carga los datos de viviendas de Boston
import tensorflow as tf

# Cargar los datos de entrenamiento y prueba
(train_data, train_targets), (test_data, test_targets) = tf.keras.datasets.boston_housing.load_data()

# Explora el tamaño de cada conjunto
print("Tamaño del conjunto de entrenamiento:", train_data.shape)
print("Tamaño de los targets de entrenamiento:", train_targets.shape)
print("Tamaño del conjunto de prueba:", test_data.shape)
print("Tamaño de los targets de prueba:", test_targets.shape)
```

```
Tamaño del conjunto de entrenamiento: (404, 13)
Tamaño de los targets de entrenamiento: (404,)
```

```
Tamaño del conjunto de prueba: (102, 13)
Tamaño de los targets de prueba: (102,)
```

Ejercicio 2

Crear una red con la siguiente configuración y entrénala:

- **Configuración de la red**:
  - Arquitectura de la red:
    - *1º Capa*: capa de entrada donde indiques la dimensión de los datos.
    - *2º Capa*: capa densa con 8 neuronas y función de activación *relu*.
    - *3º Capa*: capa densa con 8 neuronas y función de activación *relu*.
    - *4º Capa*: capa de salida con una neurona sin función de activación.
  - Tipo de entrenamiento:
    - *Epochs*: 100
    - *Optimizador*: *adam*
    - *Learning Rate*: 0.001
    - *Función de error*: error cuadrático medio (*mean_squared_error*)

```
# Completar
import tensorflow as tf

# Crear el modelo secuencial
model = tf.keras.models.Sequential()

# 1º Capa: capa de entrada con la dimensión de los datos
model.add(tf.keras.layers.InputLayer(input_shape=(13,)))  # Cambia el número según la dimensión de tus datos

# 2º Capa: capa densa con 8 neuronas y función de activación relu
model.add(tf.keras.layers.Dense(8, activation='relu'))

# 3º Capa: capa densa con 8 neuronas y función de activación relu
model.add(tf.keras.layers.Dense(8, activation='relu'))

# 4º Capa: capa de salida con una neurona sin función de activación
model.add(tf.keras.layers.Dense(1))

# Compilar el modelo
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='mean_squared_error')

# Entrenar el modelo
model.fit(train_data, train_targets, epochs=100, batch_size=32, validation_split=0.2)
```

```
Epoch 87/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 54.0674 - val_loss: 72.4235
Epoch 88/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 58.4588 - val_loss: 70.0853
Epoch 89/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 44.2819 - val_loss: 67.2453
Epoch 90/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 54.1142 - val_loss: 72.7962
Epoch 91/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 49.6454 - val_loss: 65.8947
Epoch 92/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 56.5390 - val_loss: 71.1666
Epoch 93/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 49.5341 - val_loss: 65.7808
Epoch 94/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 55.6712 - val_loss: 73.9731
Epoch 95/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 56.4246 - val_loss: 66.5697
Epoch 96/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 60.5972 - val_loss: 66.3113
Epoch 97/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 50.0614 - val_loss: 69.7965
Epoch 98/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 55.2237 - val_loss: 63.5900
Epoch 99/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 55.1557 - val_loss: 67.3451
Epoch 100/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 39.4107 - val_loss: 62.6973
<keras.src.callbacks.history.History at 0x7f8f542bcd90>
```

```
#Cambiar nombre a los modelos
# Crear y entrenar el primer modelo
modelo_1 = model  # asigna el primer modelo a modelo_1

# Crear y entrenar el segundo modelo
modelo_2 = model  # asigna el segundo modelo a modelo_2

# Crear y entrenar el tercer modelo
modelo_3 = model  # asigna el tercer modelo a modelo_3
```

Evalúa el modelo usando la función *evaluate* para el conjunto de test:

```
# Completar

test_loss = model.evaluate(test_data, test_targets)

print("Pérdida en el conjunto de prueba:", test_loss)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 51.2412
Pérdida en el conjunto de prueba: 57.315887451171875
```

Ejercicio 3

Crear una red con la siguiente configuración y entrénala:

- **Configuración de la red**:
  - Arquitectura de la red:
    - *1º Capa*: capa de entrada donde indiques la dimensión de los datos.
    - *2º Capa*: capa densa con 16 neuronas y función de activación *relu*.
    - *3º Capa*: capa densa con 16 neuronas y función de activación *relu*.
    - *4º Capa*: capa densa con 32 neuronas y función de activación *relu*.
    - *4º Capa*: capa de salida con una neurona sin función de activación.
  - Tipo de entrenamiento:
    - *Epochs*: 300
    - *Optimizador*: *adam*
    - *Learning Rate*: 0.0001
    - *Función de error*: error cuadrático medio (*mean_squared_error*)

```
# Completar
import tensorflow as tf

# Crear el modelo secuencial
model = tf.keras.models.Sequential()

# 1° Capa: capa de entrada con la dimensión de los datos
model.add(tf.keras.layers.InputLayer(input_shape=(13,)))  # Cambia el número según la dimensión de tus datos

# 2° Capa: capa densa con 16 neuronas y función de activación relu
model.add(tf.keras.layers.Dense(16, activation='relu'))

# 3° Capa: capa densa con 16 neuronas y función de activación relu
model.add(tf.keras.layers.Dense(16, activation='relu'))

# 4° Capa: capa densa con 32 neuronas y función de activación relu
model.add(tf.keras.layers.Dense(32, activation='relu'))

# 5° Capa: capa de salida con una neurona sin función de activación
model.add(tf.keras.layers.Dense(1))

# Compilar el modelo
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='mean_squared_error')

# Entrenar el modelo
model.fit(train_data, train_targets, epochs=300, batch_size=32, validation_split=0.2)
```

```
Epoch 299/300
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 51.5519 - val_loss: 67.1436
Epoch 300/300
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 46.1336 - val_loss: 65.3009
<keras.src.callbacks.history.History at 0x7f8f43f69540>
```

Evalúa el modelo usando la función *evaluate* para el conjunto de test:

```
# Completar

test_loss = model.evaluate(test_data, test_targets)

print("Pérdida en el conjunto de prueba:", test_loss)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 55.8198
Pérdida en el conjunto de prueba: 62.91109085083008
```

Ejercicio 4

Crear una red con la siguiente configuración y entrénala:

- **Configuración de la red**:
  - Arquitectura de la red:
    - *1º Capa*: capa de entrada donde indiques la dimensión de los datos.
    - *2º Capa*: capa densa con 32 neuronas y función de activación *relu*.
    - *3º Capa*: capa densa con 64 neuronas y función de activación *relu*.
    - *4º Capa*: capa densa con 128 neuronas y función de activación *relu*.
    - *4º Capa*: capa de salida con una neurona sin función de activación.
  - Tipo de entrenamiento:
    - *Epochs*: 200
    - *Optimizador*: *adam*
    - *Learning Rate*: 0.001
    - *Función de error*: error cuadrático medio (*mean_squared_error*)

```
# Completar
import tensorflow as tf

# Crear el modelo secuencial
model = tf.keras.models.Sequential()

# 1° Capa: capa de entrada con la dimensión de los datos
model.add(tf.keras.layers.InputLayer(input_shape=(13,)))  # Cambia el número según la dimensión de tus datos

# 2° Capa: capa densa con 32 neuronas y función de activación relu
model.add(tf.keras.layers.Dense(32, activation='relu'))

# 3° Capa: capa densa con 64 neuronas y función de activación relu
model.add(tf.keras.layers.Dense(64, activation='relu'))

# 4° Capa: capa densa con 128 neuronas y función de activación relu
model.add(tf.keras.layers.Dense(128, activation='relu'))

# 5° Capa: capa de salida con una neurona sin función de activación
model.add(tf.keras.layers.Dense(1))

# Compilar el modelo
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='mean_squared_error')

# Entrenar el modelo
model.fit(train_data, train_targets, epochs=200, batch_size=32, validation_split=0.2)
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 38.4798 - val_loss: 46.4641
Epoch 176/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 33.4428 - val_loss: 38.8764
Epoch 177/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 38.7024 - val_loss: 34.7594
Epoch 178/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 30.9310 - val_loss: 27.5034
Epoch 179/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 24.6684 - val_loss: 27.7814
Epoch 180/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 23.4735 - val_loss: 26.7801
Epoch 181/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 19.5379 - val_loss: 32.3971
Epoch 182/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 22.5361 - val_loss: 20.9792
Epoch 183/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 15.5022 - val_loss: 22.0425
Epoch 184/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 13.3497 - val_loss: 20.3963
Epoch 185/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 13.5002 - val_loss: 21.4448
Epoch 186/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 13.7261 - val_loss: 21.5052
Epoch 187/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 14.4277 - val_loss: 20.3914
Epoch 188/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 13.0434 - val_loss: 20.8471
Epoch 189/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 15.4983 - val_loss: 22.2064
Epoch 190/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 14.2205 - val_loss: 20.0697
Epoch 191/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 16.7135 - val_loss: 21.1949
Epoch 192/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 15.0748 - val_loss: 21.9256
Epoch 193/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 13.3670 - val_loss: 21.4883
Epoch 194/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 14.8650 - val_loss: 20.0123
Epoch 195/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 12.7678 - val_loss: 22.4803
Epoch 196/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 14.3467 - val_loss: 23.1180
Epoch 197/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 13.8046 - val_loss: 24.1446
Epoch 198/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 24.7854 - val_loss: 27.1249
Epoch 199/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 23.0430 - val_loss: 22.3545
Epoch 200/200
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 14.8039 - val_loss: 24.6781
<keras.src.callbacks.history.History at 0x7f8f51809360>
```

Evalúa el modelo usando la función *evaluate* para el conjunto de test:

```
# Completar

test_loss = model.evaluate(test_data, test_targets)

print("Pérdida en el conjunto de prueba:", test_loss)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 25.3730
Pérdida en el conjunto de prueba: 31.154115676879883
```

Ejercicio 5

Compara los resultados obtenidos en cada uno de los modelos entrenados y quédate con el mejor. **Justifica tu respuesta**.

El resultado no tiene porqué ser el mismo ya que los entrenamientos son aleatorios, pero un posible resultado podría ser este:

- Modelo 1: 83.3437 MSE
- Modelo 2: 51.6446 MSE
- Modelo 3: 27.6275 MSE

Además guarda el mejor modelo en formato *.h5* usando la función `save`:

Instrucciones para el proceso Entrenar los modelos: Ejecuta cada uno de los ejercicios previos (Ejercicio 2, Ejercicio 3, y Ejercicio 4), entrenando los modelos con las configuraciones específicas para cada ejercicio.

Evaluar cada modelo: Una vez que cada modelo esté entrenado, utiliza la función evaluate para calcular el MSE (Error Cuadrático Medio) en el conjunto de prueba.

```
# Completar
# Evaluar cada modelo en el conjunto de prueba
mse_modelo_1 = modelo_1.evaluate(test_data, test_targets)
mse_modelo_2 = modelo_2.evaluate(test_data, test_targets)
mse_modelo_3 = modelo_3.evaluate(test_data, test_targets)

print("MSE Modelo 1:", mse_modelo_1)
print("MSE Modelo 2:", mse_modelo_2)
print("MSE Modelo 3:", mse_modelo_3)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 51.2412
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 51.2412
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 51.2412
MSE Modelo 1: 57.315887451171875
MSE Modelo 2: 57.315887451171875
MSE Modelo 3: 57.315887451171875
```

```
# Supongamos que Modelo 3 es el mejor
modelo_3.save("mejor_modelo.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
```

Ejercicio 6 (Opcional)

Carga el mejor modelo que has guardado en el ejercicio anterior usando la función `load_model` y realizar predicciones usando la función `predict`.

Para realizar las predicciones usa 10 datos cualquier del conjunto de test:

```
# Completar
import tensorflow as tf

# Cargar el mejor modelo guardado
mejor_modelo = tf.keras.models.load_model("mejor_modelo.h5")

# Seleccionar 10 datos del conjunto de prueba para hacer las predicciones
datos_prueba = test_data[:10]

# Realizar predicciones
predicciones = mejor_modelo.predict(datos_prueba)

# Imprimir las predicciones
print("Predicciones para los 10 datos de prueba:", predicciones)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
Predicciones para los 10 datos de prueba: [[13.375984]
 [19.25963 ]
 [26.98676 ]
 [18.613022]
 [26.544611]
 [21.385908]
 [30.248775]
 [28.943789]
 [22.222166]
 [20.70221 ]]
```