Final Deep Learning

Gary F. Martin

Profesor: Alberto Ezpondaburu

Martes 23 de noviembre de 2021

Universidad Complutense de Madrid

Master de Big Data y Business Analytics

Dado que el entrenamiento de redes neuronales es una tarea muy costosa, **se recomienda ejecutar el notebooks en Google Colab**, por supuesto también se puede ejecutar en local.

Al entrar en Google Colab bastará con hacer click en `upload` y subir este notebook. No olvide luego descargarlo en `File->Download .ipynb`

El examen deberá ser entregado con las celdas ejecutadas, si alguna celda no está ejecutadas no se contará.

El examen se divide en tres partes, con la puntuación que se indica a continuación. La puntuación máxima será 10.

- Actividad 1: Redes Densas: 4 pts
  - Correcta normalización: máximo de 0.25 pts
  - Cuestión 1: 1 pt
  - Cuestión 2: 1 pt
  - Cuestión 3: 0.5 pts
  - Cuestión 4: 0.25 pts
  - Cuestión 5: 0.25 pts
  - Cuestión 6: 0.25 pts
  - Cuestión 7: 0.25 pts
  - Cuestión 8: 0.25 pts
- Actividad 2: Redes Convolucionales: 4 pts
  - Cuestión 1: 1 pt
  - Cuestión 2: 1.5 pt
  - Cuestión 3: 0.5 pts
  - Cuestión 4: 0.25 pts
  - Cuestión 5: 0.25 pts

    - Cuestión 6: 0.25 pts
    - Cuestión 7: 0.25 pts
- Actividad 3: Redes Recurrentes: 2 pts
    - Cuestión 1: 0.5 pt
    - Cuestión 2: 0.5 pt
    - Cuestión 3: 0.5 pts
    - Cuestión 4: 0.25 pts
    - Cuestión 5: 0.25 pts

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Actividad 1: Redes Densas

Para esta primera actividad vamos a utilizar el boston housing dataset. Con el que trataremos de predecir el precio de una casa con 13 features.

Puntuación:

Normalizar las features correctamente (x_train, x_test): 0.1 pts , 0.25 si se normalizan con el Normalization layer de Keras. Ejemplo de uso: Introduction_to_RNN_Time_Series

```
tf.keras.layers.experimental.preprocessing.Normalization(
    axis=-1, dtype=None, mean=None, variance=None, **kwargs
)
```

- Correcta normalización: máximo de 0.25 pts
- Cuestión 1: 1 pt
- Cuestión 2: 1 pt
- Cuestión 3: 0.5 pts
- Cuestión 4: 0.25 pts
- Cuestión 5: 0.25 pts
- Cuestión 6: 0.25 pts
- Cuestión 7: 0.25 pts
- Cuestión 8: 0.25 pts

Luego de importar las librerías y funciones que usaremos, cargamos los datos de "boston_housing" donde el objetivo será predecir el precio de una vivienda en base a 13

variables sobre las diferentes viviendas que componen el dataset. Se divide a continuación este dataset para train y test.

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.boston_housing.load_data(
    path='boston_housing.npz',
    test_split=0.2,
)
print('x_train, y_train shapes:', x_train.shape, y_train.shape)
print('x_test, y_test shapes:', x_train.shape, y_train.shape)
print('Some prices: ', y_train[:5])
```

```
x_train, y_train shapes: (404, 13) (404,)
x_test, y_test shapes: (404, 13) (404,)
Some prices:  [15.2 42.3 50.  21.1 17.7]
```

```
## Si quiere, puede normalizar las features
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_norm = scaler.fit_transform(x_train)
X_test_norm = scaler.fit_transform(x_test)
print('X_train mu, sigma', X_train_norm.mean(0), X_train_norm.std(0))
print('X_test mu, sigma', X_test_norm.mean(0), X_test_norm.std(0))
```

```
X_train mu, sigma [-1.01541438e-16  1.09923072e-17  1.74337992e-15 -1.26686340e-16
 -5.25377321e-15  6.41414864e-15  2.98441140e-16  4.94653823e-16
  1.12671149e-17 -1.98136337e-16  2.36686358e-14  5.95679996e-15
  6.13920356e-16] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
X_test mu, sigma [-1.37417311e-16 -1.08845395e-17 -5.35519341e-16 -2.44902138e-17
  6.46541644e-16 -1.55812182e-15  1.82860263e-16  1.95921710e-17
 -1.74152631e-17 -4.08170230e-18 -3.91190348e-15 -3.20440842e-15
  8.71851611e-16] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Cuestión 1: Cree un modelo secuencial que contenga 4 capas ocultas(hidden layers), con más de 60 neuronas por capa, sin regularización y obtenga los resultados.

Puntuación:

- Obtener el modelo correcto: 0.8 pts
- Compilar el modelo: 0.1pts
- Acertar con la función de pérdida: 0.1 pts

He creado un modelo secuencial con 4 capas ocultas, que contienen 64 neuranas cada una y su función de activación es "relu". Para la capa de salida también utilizo relu ya que queremos una

salida única sobre un precio en formato numérico y positivo. En la compilación del modelo utilizo las métricas MAE y MSE para conocer el nivel de error del modelo y su bondad.

Entrenamos el modelo con 200 épocas, un batch size de 32 y usamos un validation_split de 0.2.

```
# Código aquí # primer parametro en el modelo es el numero de neuronas.
model = tf.keras.models.Sequential([
    keras.Input(shape=(13, )),
····layers.Dense(64,·activation='relu',·name='layer1'),
····layers.Dense(64,·activation='relu',·name='layer2'),
····layers.Dense(64,·activation='relu',·name='layer3'),
····layers.Dense(64,·activation='relu',·name='layer4'),
    layers.Dense(1, activation='relu')
]) # relu o linear ya que es regresión lineal.


# Compilación del modelo
# Código aquí
model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae', 'mse']
)


# No modifique el código # training
model.fit(x_train,
          y_train,
          epochs=200,
          batch_size=32,
          validation_split=0.2,
          verbose=1)
```

```
11/11 [==============================] - 0s 6ms/step - loss: 14.1802 - mae: 2.7018
Epoch 157/200
11/11 [==============================] - 0s 7ms/step - loss: 13.5603 - mae: 2.6589
Epoch 158/200
11/11 [==============================] - 0s 7ms/step - loss: 18.4885 - mae: 2.9529
Epoch 159/200
11/11 [==============================] - 0s 5ms/step - loss: 12.2033 - mae: 2.5985
Epoch 160/200
11/11 [==============================] - 0s 7ms/step - loss: 15.3498 - mae: 2.7766
Epoch 161/200
11/11 [==============================] - 0s 8ms/step - loss: 13.2022 - mae: 2.6471
Epoch 162/200
11/11 [==============================] - 0s 6ms/step - loss: 15.2515 - mae: 2.7865
Epoch 163/200
11/11 [==============================] - 0s 7ms/step - loss: 39.1442 - mae: 4.3112
Epoch 164/200
11/11 [==============================] - 0s 8ms/step - loss: 24.0716 - mae: 3.6717
Epoch 165/200
11/11 [==============================] - 0s 7ms/step - loss: 15.8539 - mae: 3.1108
Epoch 166/200
11/11 [==============================] - 0s 7ms/step - loss: 14.5023 - mae: 2.8588
Epoch 167/200
11/11 [==============================] - 0s 8ms/step - loss: 13.0480 - mae: 2.7203
Epoch 168/200
11/11 [==============================] - 0s 7ms/step - loss: 19.2575 - mae: 3.1195
```

```
Epoch 169/200
11/11 [==============================] - 0s 7ms/step - loss: 14.7734 - mae: 2.8960
Epoch 170/200
11/11 [==============================] - 0s 5ms/step - loss: 11.3175 - mae: 2.4211
Epoch 171/200
11/11 [==============================] - 0s 8ms/step - loss: 11.4114 - mae: 2.5045
Epoch 172/200
11/11 [==============================] - 0s 7ms/step - loss: 12.0502 - mae: 2.6260
Epoch 173/200
11/11 [==============================] - 0s 5ms/step - loss: 12.7507 - mae: 2.5124
Epoch 174/200
11/11 [==============================] - 0s 8ms/step - loss: 16.6052 - mae: 3.0306
Epoch 175/200
11/11 [==============================] - 0s 5ms/step - loss: 12.4196 - mae: 2.6487
Epoch 176/200
11/11 [==============================] - 0s 8ms/step - loss: 13.2947 - mae: 2.6685
Epoch 177/200
11/11 [==============================] - 0s 8ms/step - loss: 17.6699 - mae: 3.0262
Epoch 178/200
11/11 [==============================] - 0s 7ms/step - loss: 15.9941 - mae: 2.8157
Epoch 179/200
11/11 [==============================] - 0s 8ms/step - loss: 18.8155 - mae: 3.1557
Epoch 180/200
11/11 [==============================] - 0s 7ms/step - loss: 19.9229 - mae: 3.1206
Epoch 181/200
11/11 [==============================] - 0s 7ms/step - loss: 19.3361 - mae: 3.0617
Epoch 182/200
11/11 [==============================] - 0s 7ms/step - loss: 16.4253 - mae: 2.7751
Epoch 183/200
11/11 [==============================] - 0s 5ms/step - loss: 25.7584 - mae: 3.4105
Epoch 184/200
11/11 [==============================] - 0s 7ms/step - loss: 21.2543 - mae: 3.3709
Epoch 185/200
```

```
# No modifique el código
results = model.evaluate(x_test, y_test, verbose=1)
print('Test Loss: {}'.format(results))
```

```
4/4 [==============================] - 0s 3ms/step - loss: 32.2917 - mae: 3.7062 - ms
Test Loss: [32.29167938232422, 3.7062222957611084, 32.29167938232422]
```

Test Loss: 28.860746383666992

MAE: 3.672987937927246

MSE: 28.860746383666992

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 64)                896

 layer2 (Dense)              (None, 64)                4160
```

```
 layer3 (Dense)              (None, 64)                4160

 layer4 (Dense)              (None, 64)                4160

 dense_13 (Dense)            (None, 1)                 65

=================================================================
Total params: 13,441
Trainable params: 13,441
Non-trainable params: 0
_________________________________________________________________
```

Cuestión 2: Utilice el mismo modelo de la cuestión anterior pero añadiendo al menos dos técnicas distinas de regularización.

Ejemplos de regularización: Prevent_Overfitting.ipynb

Puntuación:

- Obtener el modelo con la regularización: 0.8 pts
- Obtener un `test loss` inferior al anterior: 0.2 pts

Agrego ahora dos técnicas de regularización para intentar reducir el error en nuestros indicadores del modelo. Aplico el regularizador L1 y el regularizador L2. El error en este caso no mejora, obtenemos un resultado de MAE por 4.08 y un MSE de 33.8.

```
from tensorflow.keras import regularizers
kerner_regularizer_l1 = regularizers.l1_l2(l1=1e-5, l2=5e-4)
kerner_regularizer_l2 = regularizers.l2(5e-4)
kerner_regularizer_l3 = regularizers.l1(1e-5)


model = keras.models.Sequential([
    keras.layers.Dense(64, input_shape=(13,)),
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l1()),
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l2(0.0
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(
    keras.layers.Dense(1, activation='relu')
])

# https://towardsdatascience.com/regularization-techniques-and-their-implementation-in-ten


# Compilación del modelo
# Código aquí
model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae', 'mse']
)
```

```
batch_size=32



# No modifique el código
model.fit(x_train,
          y_train,
          epochs=200,
          batch_size=batch_size,
          validation_split=0.2,
          verbose=1)
```

```
11/11 [==============================] - 0s 6ms/step - loss: 22.9398 - mae: 2.8652
Epoch 172/200
11/11 [==============================] - 0s 7ms/step - loss: 18.9637 - mae: 2.6262
Epoch 173/200
11/11 [==============================] - 0s 7ms/step - loss: 20.2053 - mae: 2.6122
Epoch 174/200
11/11 [==============================] - 0s 5ms/step - loss: 22.0751 - mae: 2.8131
Epoch 175/200
11/11 [==============================] - 0s 7ms/step - loss: 25.9337 - mae: 3.1575
Epoch 176/200
11/11 [==============================] - 0s 6ms/step - loss: 21.7743 - mae: 2.8759
Epoch 177/200
11/11 [==============================] - 0s 8ms/step - loss: 20.7791 - mae: 2.6990
Epoch 178/200
11/11 [==============================] - 0s 8ms/step - loss: 21.2688 - mae: 2.7927
Epoch 179/200
11/11 [==============================] - 0s 8ms/step - loss: 20.0684 - mae: 2.6701
Epoch 180/200
11/11 [==============================] - 0s 8ms/step - loss: 22.0235 - mae: 2.9317
Epoch 181/200
11/11 [==============================] - 0s 6ms/step - loss: 19.4276 - mae: 2.6672
Epoch 182/200
11/11 [==============================] - 0s 6ms/step - loss: 25.5216 - mae: 2.9705
Epoch 183/200
11/11 [==============================] - 0s 9ms/step - loss: 23.8674 - mae: 3.0055
Epoch 184/200
11/11 [==============================] - 0s 8ms/step - loss: 22.3687 - mae: 3.0584
Epoch 185/200
11/11 [==============================] - 0s 7ms/step - loss: 21.2080 - mae: 2.8440
Epoch 186/200
11/11 [==============================] - 0s 8ms/step - loss: 22.4675 - mae: 2.8553
Epoch 187/200
11/11 [==============================] - 0s 5ms/step - loss: 24.4438 - mae: 3.1626
Epoch 188/200
11/11 [==============================] - 0s 7ms/step - loss: 21.7557 - mae: 2.8240
Epoch 189/200
11/11 [==============================] - 0s 7ms/step - loss: 19.9338 - mae: 2.6186
Epoch 190/200
11/11 [==============================] - 0s 7ms/step - loss: 19.8738 - mae: 2.8046
Epoch 191/200
11/11 [==============================] - 0s 8ms/step - loss: 18.0412 - mae: 2.4961
Epoch 192/200
11/11 [==============================] - 0s 8ms/step - loss: 16.8786 - mae: 2.4193
Epoch 193/200
11/11 [==============================] - 0s 8ms/step - loss: 19.3177 - mae: 2.6188
Epoch 194/200
```

```
11/11 [==============================] - 0s 7ms/step - loss: 26.5790 - mae: 3.1876
Epoch 195/200
11/11 [==============================] - 0s 8ms/step - loss: 20.2022 - mae: 2.8008
Epoch 196/200
11/11 [==============================] - 0s 7ms/step - loss: 18.4426 - mae: 2.5004
Epoch 197/200
11/11 [==============================] - 0s 7ms/step - loss: 18.6029 - mae: 2.5719
Epoch 198/200
11/11 [==============================] - 0s 7ms/step - loss: 20.1195 - mae: 2.6929
Epoch 199/200
11/11 [==============================] - 0s 6ms/step - loss: 19.2712 - mae: 2.5666
```

```
# No modifique el código
results = model.evaluate(x_test, y_test, verbose=1)
print('Test Loss: {}'.format(results))
```

```
4/4 [==============================] - 0s 3ms/step - loss: 39.9958 - mae: 4.6306 - ms
Test Loss: [39.99578094482422, 4.630557060241699, 34.3633918762207]
```

Test Loss: 37.17987060546875

MAE: 4.078502178192139

MSE: 33.804229736328125

Cuestión 3: Utilice el mismo modelo de la cuestión anterior pero añadiendo un callback de early stopping. Obtenga un test loss inferior al del modelo anterior

```
model = tf.keras.models.Sequential()
# Código aquí
model = keras.models.Sequential([
    keras.layers.Dense(64, input_shape=(13,)),
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l1()),
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l2(0.0
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(
    keras.layers.Dense(1, activation='relu')
])
```

```
# Compilación del modelo
# Código aquí
model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae', 'mse']
)
```

```
## definir el early stopping callback
```

```
# Código aquí
es_callback = keras.callbacks.EarlyStopping(
    monitor='val_loss',  # can be 'val_accuracy'
    patience=5,  # if during 5 epochs there is no improvement in `val_loss`, the execution
    verbose=1)
model.fit(x_train,
          y_train,
          epochs=200,
          batch_size=32,
          validation_split=0.2,
          verbose=1,
          callbacks=[es_callback]) # Código aquí
```

```
Epoch 5/200
11/11 [==============================] - 0s 7ms/step - loss: 577.4386 - mae: 22.08
Epoch 6/200
11/11 [==============================] - 0s 6ms/step - loss: 576.7184 - mae: 22.08
Epoch 7/200
11/11 [==============================] - 0s 7ms/step - loss: 576.0520 - mae: 22.08
Epoch 8/200
11/11 [==============================] - 0s 8ms/step - loss: 575.4384 - mae: 22.08
Epoch 9/200
11/11 [==============================] - 0s 6ms/step - loss: 574.8743 - mae: 22.08
Epoch 10/200
11/11 [==============================] - 0s 6ms/step - loss: 574.3614 - mae: 22.08
Epoch 11/200
11/11 [==============================] - 0s 6ms/step - loss: 573.8976 - mae: 22.08
Epoch 12/200
11/11 [==============================] - 0s 7ms/step - loss: 573.4810 - mae: 22.08
Epoch 13/200
11/11 [==============================] - 0s 6ms/step - loss: 573.1163 - mae: 22.08
Epoch 14/200
11/11 [==============================] - 0s 5ms/step - loss: 572.7968 - mae: 22.08
Epoch 15/200
11/11 [==============================] - 0s 7ms/step - loss: 572.5222 - mae: 22.08
Epoch 16/200
11/11 [==============================] - 0s 7ms/step - loss: 572.2911 - mae: 22.08
Epoch 17/200
11/11 [==============================] - 0s 7ms/step - loss: 572.1037 - mae: 22.08
Epoch 18/200
11/11 [==============================] - 0s 6ms/step - loss: 571.9600 - mae: 22.08
Epoch 19/200
11/11 [==============================] - 0s 6ms/step - loss: 366.8341 - mae: 16.19
Epoch 20/200
11/11 [==============================] - 0s 7ms/step - loss: 145.0256 - mae: 9.143
Epoch 21/200
11/11 [==============================] - 0s 6ms/step - loss: 91.9374 - mae: 6.9937
Epoch 22/200
11/11 [==============================] - 0s 7ms/step - loss: 71.5689 - mae: 6.1563
Epoch 23/200

11/11 [==============================] - 0s 6ms/step - loss: 65.8925 - mae: 5.8353
Epoch 24/200
11/11 [==============================] - 0s 7ms/step - loss: 64.3731 - mae: 5.6995
Epoch 25/200
11/11 [==============================] - 0s 6ms/step - loss: 63.7799 - mae: 5.7322
Epoch 26/200
11/11 [==============================] - 0s 6ms/step - loss: 66.5768 - mae: 5.6584
Epoch 27/200
11/11 [==============================] - 0s 7ms/step - loss: 58.7866 - mae: 5.2255
```

```
11/11 [==============================] - 0s 7ms/step - loss: 58.7866 - mae: 5.3355
Epoch 28/200
11/11 [==============================] - 0s 8ms/step - loss: 63.4147 - mae: 5.5937
Epoch 29/200
11/11 [==============================] - 0s 6ms/step - loss: 58.9272 - mae: 5.3525
Epoch 30/200
11/11 [==============================] - 0s 7ms/step - loss: 58.7871 - mae: 5.1779
Epoch 31/200
11/11 [==============================] - 0s 5ms/step - loss: 60.0384 - mae: 5.3597
Epoch 32/200
11/11 [==============================] - 0s 9ms/step - loss: 58.3702 - mae: 5.1670
Epoch 00032: early stopping
```

```
# No modifique el código
results = model.evaluate(x_test, y_test, verbose=1)
print('Test Loss: {}'.format(results))
```

```
4/4 [==============================] - 0s 4ms/step - loss: 64.2640 - mae: 6.0613 - ms
Test Loss: [64.2640380859375, 6.06132173538208, 63.543975830078125]
```

Test Loss: 60.30381774902344

MAE: 5.070380210876465

MSE: 54.397457122802734

Cuestión 4: ¿Podría haberse usado otra función de activación de la neurona de salida? En caso afirmativo especifíquela.

Sí, podría haberse usado la función de activación linear por la regresión lineal, ya que es un precio lo que buscamos predecir, o también se puede utilizar la función de activación "relu" ya que será positivo el monto a predecir por nuestro modelo.

Cuestión 5: ¿Qué es lo que una neurona calcula?

a) Una función de activación seguida de una suma ponderada de las entradas.

b) Una suma ponderada de las entradas seguida de una función de activación.

c) Una función de pérdida, definida sobre el target.

d) Ninguna de las anteriores es correcta

- b) Una suma ponderada de las entradas seguida de una función de activación.*

Cuestión 6: ¿Cuál de estas funciones de activación no debería usarse en una capa oculta (hidden layer)?

a) `sigmoid`

b) `tanh`

c) `relu`

d) `linear`

La única de estas funciones de activación que no debe utilizarse en una capa oculta es la "linear".

"Las funciones de activación no pueden ser lineales porque las redes neuronales con una función de activación lineal son efectivas solo en una capa de profundidad, independientemente de cuán compleja sea su arquitectura. La entrada a las redes suele ser una transformación lineal (entrada * peso), pero el mundo real y los problemas no son lineales."

"La introducción de la no linealidad amplía los tipos de funciones que podemos representar con nuestra red neuronal."

Cuestión 7: ¿Cuál de estas técnicas es efectiva para combatir el overfitting en una red con varias capas ocultas? Ponga todas las que lo sean.

a) Dropout

b) Regularización L2.

c) Aumentar el tamaño del test set.

d) Aumentar el tamaño del validation set.

e) Reducir el número de capas de la red.

f) Data augmentation.

Respuesta:

Dropout, Regularización L2, Reducir el numero de capas de la red, Data Augmentation.

Cuestión 8: Supongamos que queremos entrenar una red para un problema de clasificación de imágenes con las siguientes clases:

{'perro','gato','persona'}. ¿Cuántas neuronas y que función de activación debería tener la capa de salida? ¿Qué función de pérdida (loss function) debería usarse?

La capa de salida debe tener cuantas clases existan en nuestro conjunto, en este caso serían 3, por perro, gato y persona. La función de activación en este caso será una "softmax" debido a que estamos entrenando una red para un problema de clasficación multiclase. Por parte de la función de pérdida, la adecuada para esta ocasión sería la "sparse_categorical_crossentropy" porque tenemos más de 2 clases con labels o etiquetas que buscamos clasificar.

Actividad 2: Redes Convolucionales

Vamos a usar el dataset cifar-10, que son 60000 imágenes de 32x32 a color con 10 clases diferentes. Para realizar mejor la práctica puede consultar Introduction_to_CNN.ipynb.

Puntuación:

- Cuestión 1: 1 pt
- Cuestión 2: 1.5 pt
- Cuestión 3: 0.5 pts
- Cuestión 4: 0.25 pts
- Cuestión 5: 0.25 pts
- Cuestión 6: 0.25 pts
- Cuestión 7: 0.25 pts

Puede normalizar las imágenes al principio o usar la capa Rescaling:

```
tf.keras.layers.experimental.preprocessing.Rescaling(
    scale, offset=0.0, name=None, **kwargs
)
```

Empezamos por crear una variable del tamaño de nuestras imágenes, siendo 32 por 32 pixeles. Seguimos creando los conjuntos de train y test al cargar las 60.000 imágenes en el dataset de cifar10.

```
image_size = (32, 32)
```

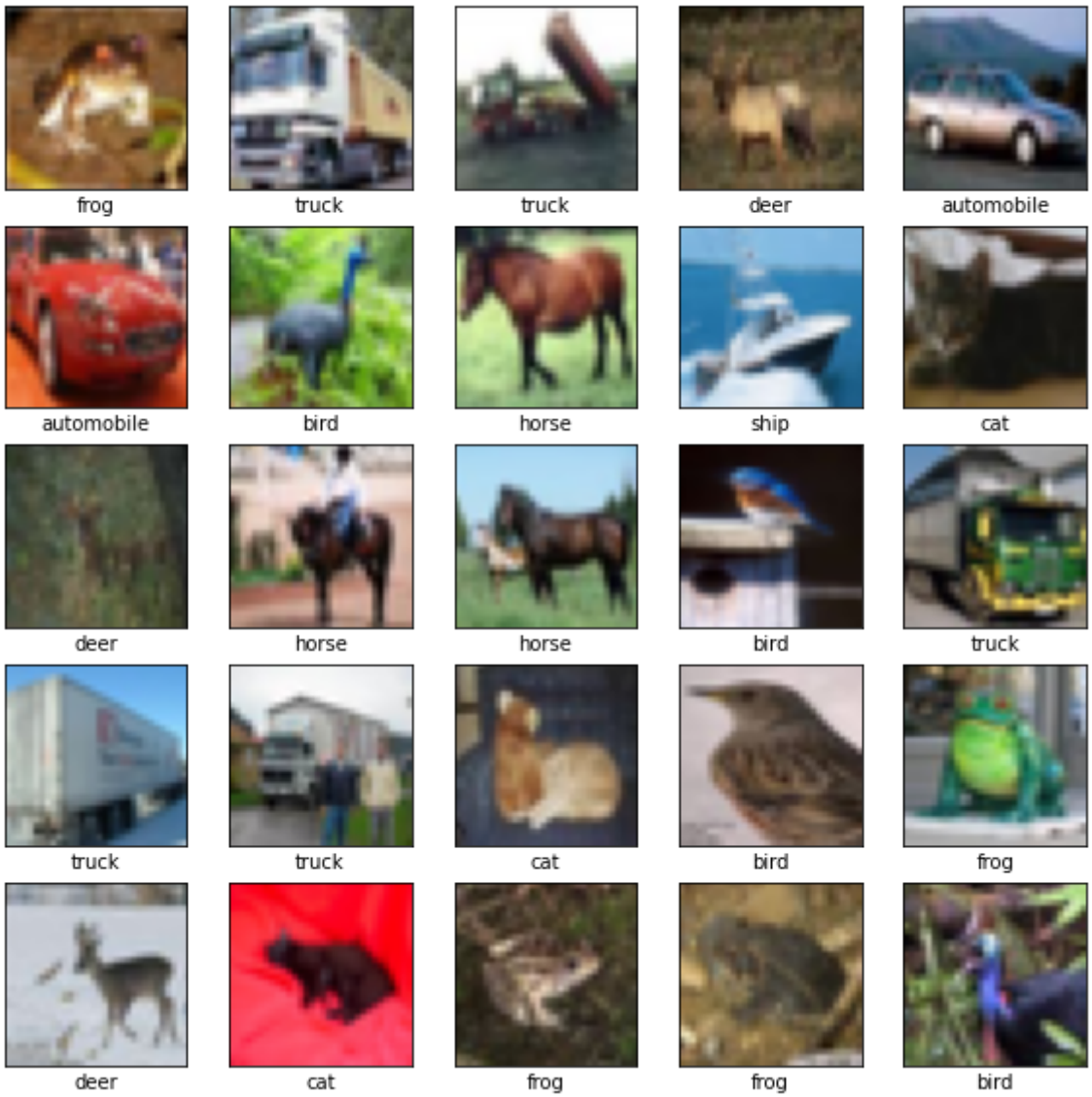
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
y_train = y_train.flatten()
y_test = y_test.flatten()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
```

```
     170500096/170498071 [==============================] - 4s 0us/step
     170508288/170498071 [==============================] - 4s 0us/step
```

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i])
    plt.xlabel(class_names[y_train[i]])
plt.show()
```




```
print('x_train, y_train shapes:', x_train.shape, y_train.shape)
print('x_test, y_test shapes:', x_test.shape, y_test.shape)
```

```
     x_train, y_train shapes: (50000, 32, 32, 3) (50000,)
     x_test, y_test shapes: (10000, 32, 32, 3) (10000,)
```

Cuestión 1: Cree una red convolucional con la API funcional con al menos dos capas convolucionales y al menos dos capas de pooling. Utilize sólo Average Pooling y no añada ninguna regularización.

Empezamos por crear la variable inputs con las dimensiones de las imágenes de nuestro conjunto e indicamos que son a color con el número 3, para luego hacer un reescaling a estos inputs. Creo dos capas convolucionales utilizando la función de activación en ambas capas pero en una agrego un padding como "valid", además, tengo las dos capas de Average Pooling incluidas en la red convolucional. Esto nos da un resultado para el accuracy del modelo de 0.5686.

```
inputs = tf.keras.Input(shape=image_size + (3, ), name='input')
reescaling = layers.experimental.preprocessing.Rescaling(1. / 255)(inputs)

# Convolution + pooling layers
conv_1 = layers.Conv2D(filters=8, kernel_size=3, activation='relu', name='conv_1')(reescal
pool_1 = layers.MaxPooling2D(pool_size=(2, 2), name='pool_1')(conv_1)

conv_2 = layers.Conv2D(4, 3, padding='valid', activation='relu', name='conv_2')(pool_1)
pool_2 = layers.MaxPooling2D(pool_size=(2, 2), name='pool_2')(conv_2)

# Flattening
flat = layers.Flatten(name='flatten')(pool_2)
dense = layers.Dense(64, activation='relu', name='dense')(flat)

# Fully-connected
outputs = layers.Dense(10, activation='softmax', name='output')(dense)
model = keras.Model(inputs=inputs, outputs=outputs)


model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

Atención: El siguiente código podría tardar unos 10 a 15 minutos, pero funciona bien, solo toma tiempo.

```
history = model.fit(x_train, y_train, epochs=25, batch_size=64,
                    validation_split=0.15)
```

```
Epoch 1/25
665/665 [==============================] - 23s 34ms/step - loss: 1.7741 - accuracy: 0
Epoch 2/25
665/665 [==============================] - 23s 34ms/step - loss: 1.5338 - accuracy: 0
```

```
Epoch 3/25
665/665 [==============================] - 23s 34ms/step - loss: 1.4567 - accuracy: 0
Epoch 4/25
665/665 [==============================] - 22s 34ms/step - loss: 1.4032 - accuracy: 0
Epoch 5/25
665/665 [==============================] - 22s 34ms/step - loss: 1.3622 - accuracy: 0
Epoch 6/25
665/665 [==============================] - 23s 34ms/step - loss: 1.3350 - accuracy: 0
Epoch 7/25
665/665 [==============================] - 23s 34ms/step - loss: 1.3054 - accuracy: 0
Epoch 8/25
665/665 [==============================] - 22s 34ms/step - loss: 1.2878 - accuracy: 0
Epoch 9/25
665/665 [==============================] - 22s 34ms/step - loss: 1.2675 - accuracy: 0
Epoch 10/25
665/665 [==============================] - 24s 36ms/step - loss: 1.2497 - accuracy: 0
Epoch 11/25
665/665 [==============================] - 23s 34ms/step - loss: 1.2332 - accuracy: 0
Epoch 12/25
665/665 [==============================] - 23s 34ms/step - loss: 1.2224 - accuracy: 0
Epoch 13/25
665/665 [==============================] - 23s 34ms/step - loss: 1.2083 - accuracy: 0
Epoch 14/25
665/665 [==============================] - 23s 34ms/step - loss: 1.1976 - accuracy: 0
Epoch 15/25
665/665 [==============================] - 23s 34ms/step - loss: 1.1862 - accuracy: 0
Epoch 16/25
665/665 [==============================] - 24s 35ms/step - loss: 1.1766 - accuracy: 0
Epoch 17/25
665/665 [==============================] - 22s 34ms/step - loss: 1.1684 - accuracy: 0
Epoch 18/25
665/665 [==============================] - 22s 34ms/step - loss: 1.1558 - accuracy: 0
Epoch 19/25
665/665 [==============================] - 23s 34ms/step - loss: 1.1516 - accuracy: 0
Epoch 20/25
665/665 [==============================] - 23s 34ms/step - loss: 1.1415 - accuracy: 0
Epoch 21/25
665/665 [==============================] - 23s 34ms/step - loss: 1.1359 - accuracy: 0
Epoch 22/25
665/665 [==============================] - 23s 34ms/step - loss: 1.1258 - accuracy: 0
Epoch 23/25
665/665 [==============================] - 24s 36ms/step - loss: 1.1215 - accuracy: 0
Epoch 24/25
665/665 [==============================] - 23s 34ms/step - loss: 1.1158 - accuracy: 0
Epoch 25/25
665/665 [==============================] - 23s 34ms/step - loss: 1.1040 - accuracy: 0
```

```
results = model.evaluate(x_test, y_test, verbose=0, batch_size=1000)
print('Test Loss: {}'.format(results[0]))
print('Test Accuracy: {}'.format(results[1]))
```

```
Test Loss: 1.2447603940963745
Test Accuracy: 0.5655999779701233
```

Test Loss: 1.236914038658142

Test Accuracy: 0.5685999989509583

Cuestión 2: Cree un modelo con la API funcional con un máximo de 2 capas convolucionales y un máximo de 2 capas de pooling. Utilize Max Pooling o Average Pooling y añada la regularización que quiera. Debe obtener un `Test accuracy > 0.68`

En este modelo aplicamos un Max Pooling en ambas capas convolucionales, aplicamos relu como función de activación en ambas y padding en la segunda. Aplicamos además una regularización L2 en la capa de output del modelo para buscar mejores resultados predictivos. El Accuracy de este modelo con los nuevos parámetros es de 0.7025, lo que indica que ha mejorado significativamente su bondad en relación al modelo anterior.

```
inputs = tf.keras.Input(shape=image_size + (3, ), name='input')
reescaling = layers.experimental.preprocessing.Rescaling(1. / 255)(inputs)

# Convolution + pooling layers
conv_1 = layers.Conv2D(filters=64, kernel_size=3, activation='relu', name='conv_1')(reesca
pool_1 = layers.MaxPooling2D(pool_size=(2, 2), name='pool_1')(conv_1)

conv_2 = layers.Conv2D(64, 3, padding='valid', activation='relu', name='conv_2')(pool_1)
pool_2 = layers.MaxPooling2D(pool_size=(2, 2), name='pool_2')(conv_2)

# Flattening
flat = layers.Flatten(name='flatten')(pool_2)
dense = layers.Dense(64, activation='relu', name='dense')(flat)

# Fully-connected
outputs = layers.Dense(10, kernel_regularizer='l2', activation='softmax', name='output')(d
model = keras.Model(inputs=inputs, outputs=outputs)


model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

Atención: El siguiente código también podría tardar unos 10 a 15 minutos, pero funciona bien.

```
es_callback = keras.callbacks.EarlyStopping(
    monitor='val_accuracy',  # can be 'val_accuracy'
    patience=4,  # if during 5 epochs there is no improvement in `val_accuracy`, the execu
    verbose=1)

history = model.fit(x_train, y_train, epochs=40, batch_size=80,
                    validation_split=0.15, callbacks=[es_callback])
```

```
Epoch 1/40
```

```
532/532 [==============================] - 91s 170ms/step - loss: 1.6837 - accuracy:
Epoch 2/40
532/532 [==============================] - 92s 172ms/step - loss: 1.3346 - accuracy:
Epoch 3/40
532/532 [==============================] - 92s 173ms/step - loss: 1.1750 - accuracy:
Epoch 4/40
532/532 [==============================] - 92s 173ms/step - loss: 1.0807 - accuracy:
Epoch 5/40
532/532 [==============================] - 94s 176ms/step - loss: 1.0105 - accuracy:
Epoch 6/40
532/532 [==============================] - 92s 172ms/step - loss: 0.9575 - accuracy:
Epoch 7/40
532/532 [==============================] - 94s 176ms/step - loss: 0.9124 - accuracy:
Epoch 8/40
532/532 [==============================] - 94s 176ms/step - loss: 0.8730 - accuracy:
Epoch 9/40
532/532 [==============================] - 93s 176ms/step - loss: 0.8373 - accuracy:
Epoch 10/40
532/532 [==============================] - 93s 176ms/step - loss: 0.8121 - accuracy:
Epoch 11/40
532/532 [==============================] - 96s 181ms/step - loss: 0.7803 - accuracy:
Epoch 12/40
532/532 [==============================] - 93s 175ms/step - loss: 0.7510 - accuracy:
Epoch 13/40
532/532 [==============================] - 93s 175ms/step - loss: 0.7218 - accuracy:
Epoch 14/40
532/532 [==============================] - 94s 177ms/step - loss: 0.6968 - accuracy:
Epoch 15/40
532/532 [==============================] - 93s 175ms/step - loss: 0.6744 - accuracy:
Epoch 16/40
532/532 [==============================] - 92s 174ms/step - loss: 0.6543 - accuracy:
Epoch 17/40
532/532 [==============================] - 93s 174ms/step - loss: 0.6287 - accuracy:
Epoch 18/40
532/532 [==============================] - 92s 174ms/step - loss: 0.6092 - accuracy:
Epoch 00018: early stopping
```

Test Loss: 0.9825606346130371

Test Accuracy: 0.7024999856948853

```
results = model.evaluate(x_test, y_test, verbose=0, batch_size=1000)
print('Test Loss: {}'.format(results[0]))
print('Test Accuracy: {}'.format(results[1]))
```

```
Test Loss: 0.9608255624771118
Test Accuracy: 0.6952999830245972
```

Cuestión 3: Añada data augmentation al principio del modelo

```
data_augmentation= keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip(),
```

```
    layers.experimental.preprocessing.RandomRotation(0.25),
    layers.experimental.preprocessing.RandomZoom(0.25),
  ]
)


inputs = tf.keras.Input(shape=image_size + (3, ), name='input')
data_aug= data_augmentation(inputs)

reescaling = layers.experimental.preprocessing.Rescaling(1. / 255)(data_aug)

# Convolution + pooling layers
conv_1 = layers.Conv2D(filters=10, kernel_size=3, activation='relu', name='conv_1')(reesca
pool_1 = layers.MaxPooling2D(pool_size=(2, 2), name='pool_1')(conv_1)

conv_2 = layers.Conv2D(64, 3, padding='valid', activation='relu', name='conv_2')(pool_1)
pool_2 = layers.MaxPooling2D(pool_size=(2, 2), name='pool_2')(conv_2)

conv_3 = layers.Conv2D(64, 3, padding='valid', activation='relu', name='conv_3')(pool_2)
pool_3 = layers.MaxPooling2D(pool_size=(2, 2), name='pool_3')(conv_3)

# Flattening
flat = layers.Flatten(name='flatten')(pool_3)
dense = layers.Dense(64, activation='relu', name='dense')(flat)

# Fully-connected
outputs = layers.Dense(10, activation='softmax', name='output')(dense)

model = keras.Model(inputs=inputs, outputs=outputs)


model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

A primera vista nos damos cuenta que el resultado es peor pero al aplicarlo en la vida real, debería ser más capaz de predecir que los modelos anteriores.

```
es_callback = keras.callbacks.EarlyStopping(
    monitor='val_accuracy',  # can be 'val_accuracy'
    patience=7,  # if during 6 epochs there is no improvement in `val_accuracy`, the execu
    verbose=1)

history = model.fit(x_train, y_train, epochs=25, batch_size=100,
                    validation_split=0.15, callbacks=[es_callback])
```

```
Epoch 1/25
425/425 [==============================] - 46s 106ms/step - loss: 1.9732 - accuracy:
Epoch 2/25
425/425 [==============================] - 46s 108ms/step - loss: 1.7939 - accuracy:
Epoch 3/25
425/425 [==============================] - 44s 104ms/step - loss: 1.7151 - accuracy:
Epoch 4/25
425/425 [==============================] - 44s 104ms/step - loss: 1.6702 - accuracy:
```

```
Epoch 5/25
425/425 [==============================] - 44s 104ms/step - loss: 1.6237 - accuracy:
Epoch 6/25
425/425 [==============================] - 45s 106ms/step - loss: 1.5966 - accuracy:
Epoch 7/25
425/425 [==============================] - 45s 106ms/step - loss: 1.5715 - accuracy:
Epoch 8/25
425/425 [==============================] - 45s 106ms/step - loss: 1.5384 - accuracy:
Epoch 9/25
425/425 [==============================] - 45s 105ms/step - loss: 1.5174 - accuracy:
Epoch 10/25
425/425 [==============================] - 45s 106ms/step - loss: 1.5029 - accuracy:
Epoch 11/25
425/425 [==============================] - 44s 104ms/step - loss: 1.4883 - accuracy:
Epoch 12/25
425/425 [==============================] - 44s 104ms/step - loss: 1.4639 - accuracy:
Epoch 13/25
425/425 [==============================] - 45s 105ms/step - loss: 1.4481 - accuracy:
Epoch 14/25
425/425 [==============================] - 45s 105ms/step - loss: 1.4295 - accuracy:
Epoch 15/25
425/425 [==============================] - 45s 106ms/step - loss: 1.4230 - accuracy:
Epoch 16/25
425/425 [==============================] - 45s 105ms/step - loss: 1.4013 - accuracy:
Epoch 17/25
425/425 [==============================] - 45s 107ms/step - loss: 1.4014 - accuracy:
Epoch 18/25
425/425 [==============================] - 45s 106ms/step - loss: 1.3885 - accuracy:
Epoch 19/25
425/425 [==============================] - 45s 107ms/step - loss: 1.3753 - accuracy:
Epoch 20/25
425/425 [==============================] - 46s 107ms/step - loss: 1.3615 - accuracy:
Epoch 21/25
425/425 [==============================] - 46s 108ms/step - loss: 1.3511 - accuracy:
Epoch 22/25
425/425 [==============================] - 46s 107ms/step - loss: 1.3548 - accuracy:
Epoch 23/25
425/425 [==============================] - 45s 107ms/step - loss: 1.3391 - accuracy:
Epoch 24/25
425/425 [==============================] - 46s 107ms/step - loss: 1.3332 - accuracy:
Epoch 25/25
425/425 [==============================] - 45s 107ms/step - loss: 1.3291 - accuracy:
```

```
results = model.evaluate(x_test, y_test, verbose=0, batch_size=1000)
print('Test Loss: {}'.format(results[0]))
print('Test Accuracy: {}'.format(results[1]))
```

```
Test Loss: 1.303092122077942
Test Accuracy: 0.5346999764442444
```

Test Loss: 1.3284704685211182

Test Accuracy: 0.5268999934196472

Cuestión 4: Cree el mismo modelo de manera secuencial. No es necesario compilar ni entrenar el modelo

```
# Código aquí
inputs = tf.keras.Input(shape=image_size + (3, ), name='input')
sec = layers.experimental.preprocessing.Rescaling(1. / 255)(inputs)

model_seq = tf.keras.models.Sequential(
    [
        layers.Conv2D(filters=64, kernel_size=3, activation='relu', name='conv_1'),
        layers.MaxPooling2D(pool_size=(2, 2), name='pool_1'),
        layers.Conv2D(64, 3, padding='valid', activation='relu', name='conv_2'),
        layers.MaxPooling2D(pool_size=(2, 2), name='pool_2'),
        layers.Conv2D(64, 3, padding='valid', activation='relu', name='conv_3'),
        layers.MaxPooling2D(pool_size=(2, 2), name='pool_3'),
        layers.Flatten(name='flatten'),
        layers.Dense(64, activation='relu', name='dense'),
        layers.Dense(10, kernel_regularizer='l2', activation='softmax', name='output'),
    ]
)
```

Cuestión 5: Si tenenemos una una imagen de entrada de 300 x 300 a color (RGB) y queremos usar una red densa. Si la primera capa oculta tiene 100 neuronas, ¿Cuántos parámetros tendrá esa capa (sin incluir el bias) ?

Respuesta

La capa tendría 30720 parámetros sin incluir el bias. Como nuestras imágenes son de 32 x 32 pixeles, vamos a multiplicarlo entre ellos y luego por 3. Multiplico por 3 debido que las imágenes son a color (Red, Green, Blue). Por el momento tenemos, 3072 neuronas (32x32x3) y al multiplicarlo por las 100 neuronas de la primera capa oculta, obtenemos 30720 neuronas en total.

Cuestión 6 Ponga las verdaderas ventajas de las redes convolucionales respecto a las densas

a) Reducen el número total de parámetros, reduciendo así el overfitting.

b) Permiten utilizar una misma 'función' en varias localizaciones de la imagen de entrada, en lugar de aprender una función diferente para cada pixel.

c) Permiten el uso del transfer learning.

d) Generalmente son menos profundas, lo que facilita su entrenamiento.

Respuesta

a). Reducen el número total de parámetros, reduciendo así el overfitting.

b) Permiten utilizar una misma 'función' en varias localizaciones de la imagen de entrada, en lugar de aprender una función diferente para cada pixel.

c) Permiten el uso del transfer learning.

Cuestión 7: Para el procesamiento de series temporales las redes convolucionales no son efectivas, habrá que usar redes recurrentes.

- **Verdadero**
- **Falso**

Respuesta:

En teoría Verdadero, porque dan mejores resultados con datos estructurados como lo son las series temporales. Pero, podría ser Falso. Investigando he encontrado argumentos por científicos de datos que sostienen que este enunciado es falso y que las redes convolucionales sí pueden ser efectivas para el procesamiento de series temporales. Uno de los argumentos siendo el siguiente: "las series temporales son señales que evolucionan en el tiempo y estas redes pueden representar adecuadamente su estructura temporal." Agrega que, "En caso de contar con varias variables, cada dato se puede disponer en una subcapa de entrada o bien agrupar todos los datos en una única capa, pero con los datos típicamente dispuestos en dos o más dimensiones, de tal forma que cada línea sea un tipo de dato."

Actividad 3: Redes Recurrentes

- Cuestión 1: 0.5 pt
- Cuestión 2: 0.5 pt
- Cuestión 3: 0.5 pts
- Cuestión 4: 0.25 pts
- Cuestión 5: 0.25 pts

Vamos a usar un dataset de las temperaturas mínimas diarias en Melbourne. La tarea será la de predecir la temperatura mínima en dos días. Puedes usar técnicas de series temporales vistas

en otras asignaturas, pero no es necesario.

```
dataset_url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-tempe
data_dir = tf.keras.utils.get_file('daily-min-temperatures.csv', origin=dataset_url)
```

```
Downloading data from https://raw.githubusercontent.com/jbrownlee/Datasets/master/dai
73728/67921 [================================] - 0s 0us/step
81920/67921 [==================================] - 0s 0us/step
```

```
df = pd.read_csv(data_dir, parse_dates=['Date'])
df.head()
```

	Date	Temp
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8
3	1981-01-04	14.6
4	1981-01-05	15.8

```
temperatures = df['Temp'].values
print('number of samples:', len(temperatures))
train_data = temperatures[:3000]
test_data = temperatures[3000:]
print('number of train samples:', len(train_data))
print('number of test samples:', len(test_data))
print('firsts trainn samples:', train_data[:10])
```

```
number of samples: 3650
number of train samples: 3000
number of test samples: 650
firsts trainn samples: [20.7 17.9 18.8 14.6 15.8 15.8 15.8 17.4 21.8 20. ]
```

Cuestión 1: Convierta `train_data` y `test_data` en ventanas de tamaño 5, para predecir el valor en 2 días

En la nomenclatura de Introduction_to_RNN_Time_Series.ipynb

```
past, future = (5, 2)
```

Para las primeras 10 muestras de train_data `[20.7, 17.9, 18.8, 14.6, 15.8, 15.8, 15.8, 17.4, 21.8, 20. ]` el resultado debería ser:

```
x[0] : [20.7, 17.9, 18.8, 14.6, 15.8] , y[0]: 15.8
x[1] : [17.9, 18.8, 14.6, 15.8, 15.8] , y[1]: 17.4
```

```
x[2] : [18.8, 14.6, 15.8, 15.8, 15.8] , y[2]: 21.8
x[3] : [14.6, 15.8, 15.8, 15.8, 17.4] , y[3]: 20.
```

```
# windowing function


import random

def convert2matrix(data_arr, past, future, shuffle=False):
    X, Y = [], []
    size = len(data_arr)
    for i in range(size - future - past + 1):
        d = i + past
        y_ind = i + past + future - 1
        X.append(data_arr[i:d])
        Y.append(data_arr[y_ind])
    if shuffle:
        c = list(zip(X, Y))
        random.shuffle(c)
        X, Y = zip(*c)
    return np.array(X), np.array(Y)


# codigo
past, future = (5, 2)
X_train, y_train = convert2matrix(train_data, past, future, shuffle=True)
X_test, y_test = convert2matrix(test_data, past, future)


X_test[0, :], y_test[0], test_data[:past + future]
```

```
(array([16.9, 16.5, 13.6, 13.2,  9.4]),
 11.8,
 array([16.9, 16.5, 13.6, 13.2,  9.4,  9.5, 11.8]))
```

Cuestión 2: Cree un modelo recurrente de dos capas GRU para predecir con las ventanas de la cuestión anterior.

```
inputs = keras.layers.Input(shape=(past,1))
#inputs_norm = norm(inputs)

gru_1 = keras.layers.GRU(128, return_sequences=True)(inputs)
gru_2 = keras.layers.GRU(128, return_sequences=False)(gru_1)
outputs = keras.layers.Dense(1)(gru_2)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer=keras.optimizers.Adam(), loss="mse")
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 5, 1)]            0

 gru (GRU)                   (None, 5, 128)            50304

 gru_1 (GRU)                 (None, 128)               99072

 dense_11 (Dense)            (None, 1)                 129

=================================================================
Total params: 149,505
Trainable params: 149,505
Non-trainable params: 0
_________________________________________________________________
```

```
es_callback = keras.callbacks.EarlyStopping(
    monitor="val_loss", min_delta=0, patience=10)

history = model.fit(
    X_train, y_train,
    epochs=200,
    validation_split=0.2, shuffle=True, batch_size = 64, callbacks=[es_callback]
)
```

```
38/38 [==============================] - 1s 25ms/step - loss: 8.2649 - val_loss: 9
Epoch 26/200
38/38 [==============================] - 1s 24ms/step - loss: 8.1757 - val_loss: 9
Epoch 27/200
38/38 [==============================] - 1s 25ms/step - loss: 8.1378 - val_loss: 9
Epoch 28/200
38/38 [==============================] - 1s 28ms/step - loss: 8.1314 - val_loss: 1(
Epoch 29/200
38/38 [==============================] - 1s 26ms/step - loss: 8.2849 - val_loss: 9
Epoch 30/200
38/38 [==============================] - 1s 26ms/step - loss: 8.3731 - val_loss: 9
Epoch 31/200
38/38 [==============================] - 1s 25ms/step - loss: 8.2469 - val_loss: 9
Epoch 32/200
38/38 [==============================] - 1s 27ms/step - loss: 8.2459 - val_loss: 9
Epoch 33/200
38/38 [==============================] - 1s 26ms/step - loss: 8.1851 - val_loss: 9
Epoch 34/200
38/38 [==============================] - 1s 26ms/step - loss: 8.0962 - val_loss: 9
Epoch 35/200
38/38 [==============================] - 1s 26ms/step - loss: 8.2335 - val_loss: 9
Epoch 36/200
38/38 [==============================] - 1s 26ms/step - loss: 8.0949 - val_loss: 9
Epoch 37/200
38/38 [==============================] - 1s 26ms/step - loss: 8.1917 - val_loss: 9
Epoch 38/200
38/38 [==============================] - 1s 27ms/step - loss: 8.0681 - val_loss: 9
Epoch 39/200
38/38 [==============================] - 1s 26ms/step - loss: 8.0823 - val_loss: 9
Epoch 40/200
38/38 [==============================] - 1s 24ms/step - loss: 8.0356 - val_loss: 9
Epoch 41/200
38/38 [==============================] - 1s 24ms/step - loss: 8.1340 - val_loss: 9
```

```
38/38 [==============================] - 1s 24ms/step - loss: 8.1349 - val_loss: 9
Epoch 42/200
38/38 [==============================] - 1s 25ms/step - loss: 8.0516 - val_loss: 9
Epoch 43/200
38/38 [==============================] - 1s 25ms/step - loss: 8.0401 - val_loss: 9
Epoch 44/200
38/38 [==============================] - 1s 25ms/step - loss: 8.0832 - val_loss: 9
Epoch 45/200
38/38 [==============================] - 1s 25ms/step - loss: 8.0612 - val_loss: 9
Epoch 46/200
38/38 [==============================] - 1s 25ms/step - loss: 8.0342 - val_loss: 9
Epoch 47/200
38/38 [==============================] - 1s 25ms/step - loss: 8.0863 - val_loss: 9
Epoch 48/200
38/38 [==============================] - 1s 25ms/step - loss: 8.0195 - val_loss: 9
Epoch 49/200
38/38 [==============================] - 1s 25ms/step - loss: 7.9474 - val_loss: 9
Epoch 50/200
38/38 [==============================] - 1s 25ms/step - loss: 8.0056 - val_loss: 9
Epoch 51/200
38/38 [==============================] - 1s 26ms/step - loss: 8.0131 - val_loss: 9
Epoch 52/200
38/38 [==============================] - 1s 24ms/step - loss: 7.9792 - val_loss: 9
Epoch 53/200
38/38 [==============================] - 1s 24ms/step - loss: 8.0071 - val_loss: 1
Epoch 54/200
```

```
results = model.evaluate(X_test, y_test, verbose=1)
print('Test Loss: {}'.format(results))
```

```
21/21 [==============================] - 0s 5ms/step - loss: 7.1372
Test Loss: 7.137163162231445
```

Test Loss: 6.858748912811279

Cuestión 3: Añada más features a la series temporal, por ejemplo `portion_year`. Cree un modelo que mejore al anterior.

```
# Volvemos a cargar los datos anteriores, ya que hemos editado los datos originales.

dataset_url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-tempe
data_dir = tf.keras.utils.get_file('daily-min-temperatures.csv', origin=dataset_url)

df = pd.read_csv(data_dir, parse_dates=['Date'])
df.head()
```

	Date	Temp
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8

```
## Puede añadir más features
df['portion_year'] = df['Date'].dt.dayofyear / 365.0
df_multi = df[['Temp', 'portion_year']].copy()

## train - test split
train_data = df_multi.iloc[:3000].copy()
test_data = df_multi.loc[3000:, :].copy()


def convert2matrix_multi(df, past, future, target, shuffle=False):
    X, Y = [], []
    size = len(df)
    for i in range(size - future - past + 1):
        d = i + past
        y_ind = i + past + future - 1
        X.append(df.iloc[i:d, :].values)
        Y.append(df.iloc[y_ind][target])
    if shuffle:
        c = list(zip(X, Y))
        random.shuffle(c)
        X, Y = zip(*c)
    return np.array(X), np.array(Y)


## Create windows
X_train, y_train = convert2matrix_multi(train_data, past, future, target="portion_year", s
X_test, y_test = convert2matrix_multi(test_data, past, future, target="portion_year")
```

He creado un modelo recurrente de dos capas GRU para predecir, con las ventanas de 5 días para pasado y 2 a futuro, y está compuesta por 128 neuronas en cada capa oculta. Añadimos más features agregando a la serie temporal en este modelo, utilizamos el EarlyStopping, aplicamos iteraciones por 200 épocas y un batch de tamaño 64.

Utilizo la métrica Test Loss para conocer el nivel de bondad del modelo, luego de realizar ajustes y mejoras al mismo, conseguí un valor de 0.003478.

```
#norm = tf.keras.layers.experimental.preprocessing.Normalization(
  #  axis=-1
#)
#norm.adapt(dataset_train.map(lambda x, y: x))

num_features =  2
inputs_shape = (past, num_features)

inputs = keras.layers.Input(shape=inputs_shape)
gru_1 = keras.layers.GRU(128, return_sequences=True)(inputs)
```

```
gru_2 = keras.layers.GRU(128, return_sequences=False)(gru_1)
outputs = keras.layers.Dense(1)(gru_2)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer=keras.optimizers.Adam(), loss="mse")
model.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 5, 2)]            0

 gru_2 (GRU)                 (None, 5, 128)            50688

 gru_3 (GRU)                 (None, 128)               99072

 dense_12 (Dense)            (None, 1)                 129

=================================================================
Total params: 149,889
Trainable params: 149,889
Non-trainable params: 0
_________________________________________________________________
```

```
es_callback = keras.callbacks.EarlyStopping(
    monitor="val_loss", min_delta=0, patience=10)

history = model.fit(
    X_train, y_train,
    epochs=200,
    validation_split=0.2, shuffle=True, batch_size = 64, callbacks=[es_callback]
)
```

```
Epoch 1/200
38/38 [==============================] - 6s 52ms/step - loss: 0.1844 - val_loss: 0
Epoch 2/200
38/38 [==============================] - 1s 25ms/step - loss: 0.0487 - val_loss: 0
Epoch 3/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0196 - val_loss: 0
Epoch 4/200
38/38 [==============================] - 1s 25ms/step - loss: 0.0102 - val_loss: 0
Epoch 5/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0081 - val_loss: 0
Epoch 6/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0072 - val_loss: 0
Epoch 7/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0076 - val_loss: 0
Epoch 8/200
38/38 [==============================] - 1s 25ms/step - loss: 0.0076 - val_loss: 0
Epoch 9/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0069 - val_loss: 0
Epoch 10/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0071 - val_loss: 0
Epoch 11/200
38/38 [==============================] - 1s 23ms/step - loss: 0.0071 - val_loss: 0
Epoch 12/200
```

```
38/38 [==============================] - 1s 24ms/step - loss: 0.0065 - val_loss: 0
Epoch 13/200
38/38 [==============================] - 1s 23ms/step - loss: 0.0064 - val_loss: 0
Epoch 14/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0064 - val_loss: 0
Epoch 15/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0065 - val_loss: 0
Epoch 16/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0072 - val_loss: 0
Epoch 17/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0061 - val_loss: 0
Epoch 18/200
38/38 [==============================] - 1s 23ms/step - loss: 0.0068 - val_loss: 0
Epoch 19/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0073 - val_loss: 0
Epoch 20/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0064 - val_loss: 0
Epoch 21/200
38/38 [==============================] - 1s 23ms/step - loss: 0.0062 - val_loss: 0
Epoch 22/200
38/38 [==============================] - 1s 23ms/step - loss: 0.0064 - val_loss: 0
Epoch 23/200
38/38 [==============================] - 1s 25ms/step - loss: 0.0058 - val_loss: 0
Epoch 24/200
38/38 [==============================] - 1s 23ms/step - loss: 0.0058 - val_loss: 0
Epoch 25/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0068 - val_loss: 0
Epoch 26/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0062 - val_loss: 0
Epoch 27/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0058 - val_loss: 0
Epoch 28/200
38/38 [==============================] - 1s 24ms/step - loss: 0.0063 - val_loss: 0
Epoch 29/200
```

```
results = model.evaluate(X_test, y_test, verbose=1)
print('Test Loss: {}'.format(results))
```

```
21/21 [==============================] - 0s 5ms/step - loss: 0.0037
Test Loss: 0.0036690961569547653
```

Test Loss: 0.0034783771261572838

Cuestión 4: ¿En cuáles de estas aplicaciones se usaría un arquitectura 'many-to-one'?

a) Clasificación de sentimiento en textos

b) Verificación de voz para iniciar el ordenador.

c) Generación de música.

d) Un clasificador que clasifique piezas de música según su autor.

Respuesta:

a) Clasificación de sentimiento en textos

Cuestión 5: ¿Qué ventajas aporta el uso de word embeddings?

a) Permiten reducir la dimensión de entrada respecto al one-hot encoding.

b) Permiten descubrir la similaridad entre palabras de manera más intuitiva que con one-hot encoding.

c) Son una manera de realizar transfer learning en nlp.

d) Permiten visualizar las relaciones entre palabras con métodos de reducción de dimensioones como el PCA.

Respuesta:

a) Permiten reducir la dimensión de entrada respecto al one-hot encoding.

b) Permiten descubrir la similaridad entre palabras de manera más intuitiva que con one-hot encoding.

c) Son una manera de realizar transfer learning en nlp.

d) Permiten visualizar las relaciones entre palabras con métodos de reducción de dimensioones como el PCA.