Redes neuronales y deep learning

Actividad Semana 3

NAtalia Camarano
Semana 3: Ajuste de modelos de Deep Learning
IEBS

```
import tensorflow as tf
import numpy as np
import pandas as pd

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

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


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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

En esta actividad vamos a abordar un problema desde 0 al igual que hemos visto en las clases de esta semana, para ello vamos a utilizar un nuevo dataset donde tendréis que realizar todos los pasos vistos en clase para ajustar un modelo vosotros mismos.

Destacar que en la actividad de esta semana nos centraremos en encontrar la mejor arquitectura de red para el dataset que tenemos y en el caso práctico nos centraremos en realizar más experimentos jugando con el tamaño del batch, el valor del learning rate y el uso de capas dropout.

Ejercicio

En este notebook vamos a usar un dataset nuevo, el dataset es muy parecido al dataset del precio de las casas de boston. Esta vez vamos a utilizar un conjunto de datos que contienen

información sobre el precio de las casas encontradas en un distrito de California. Las columnas son las siguientes:

- *longitude*: cuanto de al oeste está una casa; un valor más alto está más al oeste.
- *latitude*: cuanto de al norte está una casa; un valor más alto está más al norte.
- *housing_median_age*: edad media de una casa; un valor bajo es una casa más nueva.
- *total_rooms*: número total de habitaciones.
- *total_bedrooms*: número total de dormitorios.
- *population*: número total de personas que residen.
- *households*: número total de hogares, un grupo de personas que residen dentro de una unidad de vivienda.
- *median_income*: ingreso medio de los hogares dentro de un bloque de casas (medido en decenas de miles de dólares).
- *ocean_proximity*: ubicación de la casa cerca del océano o mar.
- *median_house_value* (***variables a predecir***): valor medio de la vivienda (medido en dólares).

Vamos a cargar los datos desde el fichero `housing.csv`:

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Ajuste de un modelo desde 0 con
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

Pasos siguientes: Generar código con `df` | Ver gráficos recomendados | New interactive sheet

```
df.shape
```

```
(20433, 10)
```

Vamos a separar la variable objetivo del resto de variables (accedemos al campo value para que los datos sean de tipo *numpy array* y se puedan usar como variable de entrada de nuestra red):

```
df.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity', 'median_house_value'],
      dtype='object')
```

```
x = df[['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity']].values
y = df[['median_house_value']].values
```

1. Establecer una función de coste adecuada a nuestro problema.

En este caso, como es un problema de regresión y los valores de nuestros datos son tan grandes, elegimos la función de coste `mean_absolute_percentage_error`, este error varía entre los valores 100 y 0 donde 100 es el pero error que podemos llegar a tener y 0 es el mejor error, por lo que en nuestros entrenamientos buscaremos un error más cercano a 0.

```
actual_loss = 'mean_absolute_percentage_error'
```

2. Elegir una estructura de red base

Ahora, como ya hemos visto en clase vamos a encontrar una estructura de red que encaje con los datos que vamos a utilizar. Vamos a crear varias redes a ver que talfuncionan.

Para hacer entrenamientos rápidos y ver si la red se adapta a los datos vamos a usar solo un subconjunto de los datos, es decir usaremos 1000 datos y no usaremos conjunto de validación.

Ejercicio 1

Crear una red con la siguiente configuración y entrénala:

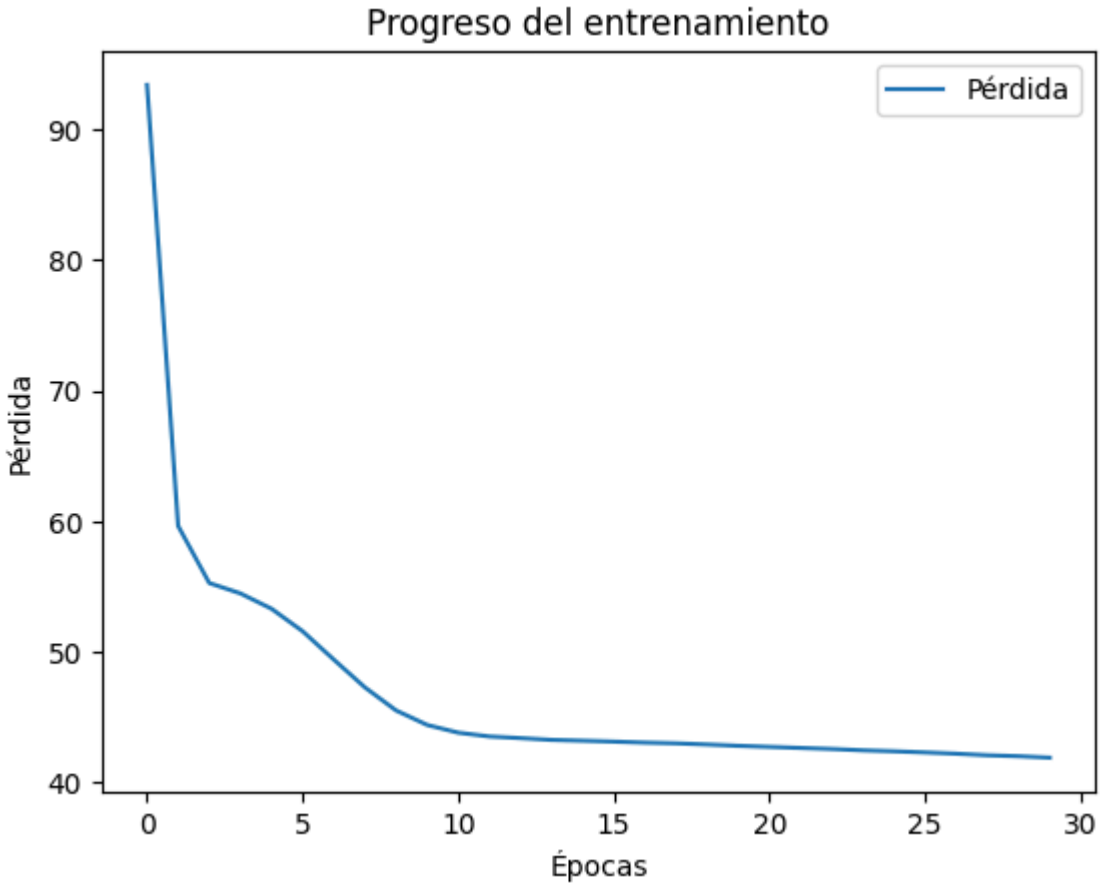
- **Configuración de la red**:
  - Arquitectura de la red:
    - *1º Capa*: capa de entrada donde indiques la dimensión de los datos.
    - *2º Capa*: capa densa con 8 neuronas y función de activación *relu*.
    - *3º Capa*: capa densa con 8 neuronas y función de activación *relu*.
    - *4º Capa*: capa de salida con una neurona sin función de activación.
  - Tipo de entrenamiento:
    - *Epochs*: 30

- *Optimizador*: *adam*
- *Learning Rate*: 0.001

```
# Completar
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Dimensiones de entrada (debes ajustar esto al tamaño real de tus datos)
input_dim = x.shape[1]  # Número de características en el dataset

# Crear el modelo
model = Sequential([
    Dense(8, activation='relu', input_dim=input_dim),  # Primera capa densa
    Dense(8, activation='relu'),                       # Segunda capa densa
    Dense(1)                                           # Capa de salida
])

# Compilar el modelo
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='mean_absolute_percentage_error')

# Entrenar el modelo
history = model.fit(x, y, epochs=30, batch_size=32)

# Mostrar resultados del entrenamiento
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Pérdida')
plt.title('Progreso del entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.show()
```

```
Epoch 1/30
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 98.0343
Epoch 2/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 64.8556
Epoch 3/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 55.3257
Epoch 4/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 55.2986
Epoch 5/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 54.1797
Epoch 6/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 52.3874
Epoch 7/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 50.6459
Epoch 8/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 48.1021
Epoch 9/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 45.7763
Epoch 10/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 44.3606
Epoch 11/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 43.8880
Epoch 12/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - loss: 43.2766
Epoch 13/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 43.3494
Epoch 14/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 43.4589
Epoch 15/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42.9790
Epoch 16/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 43.2034
Epoch 17/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 43.1934
Epoch 18/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 43.0656
Epoch 19/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 42.7236
Epoch 20/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 42.8005
Epoch 21/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 42.5777
Epoch 22/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - loss: 43.1985
Epoch 23/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 42.8874
Epoch 24/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42.6127
Epoch 25/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42.4617
Epoch 26/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 41.8558
Epoch 27/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 42.3135
Epoch 28/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 42.4543
Epoch 29/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 42.1817
```

```
Epoch 30/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 41.8703
```



Ejercicio 2

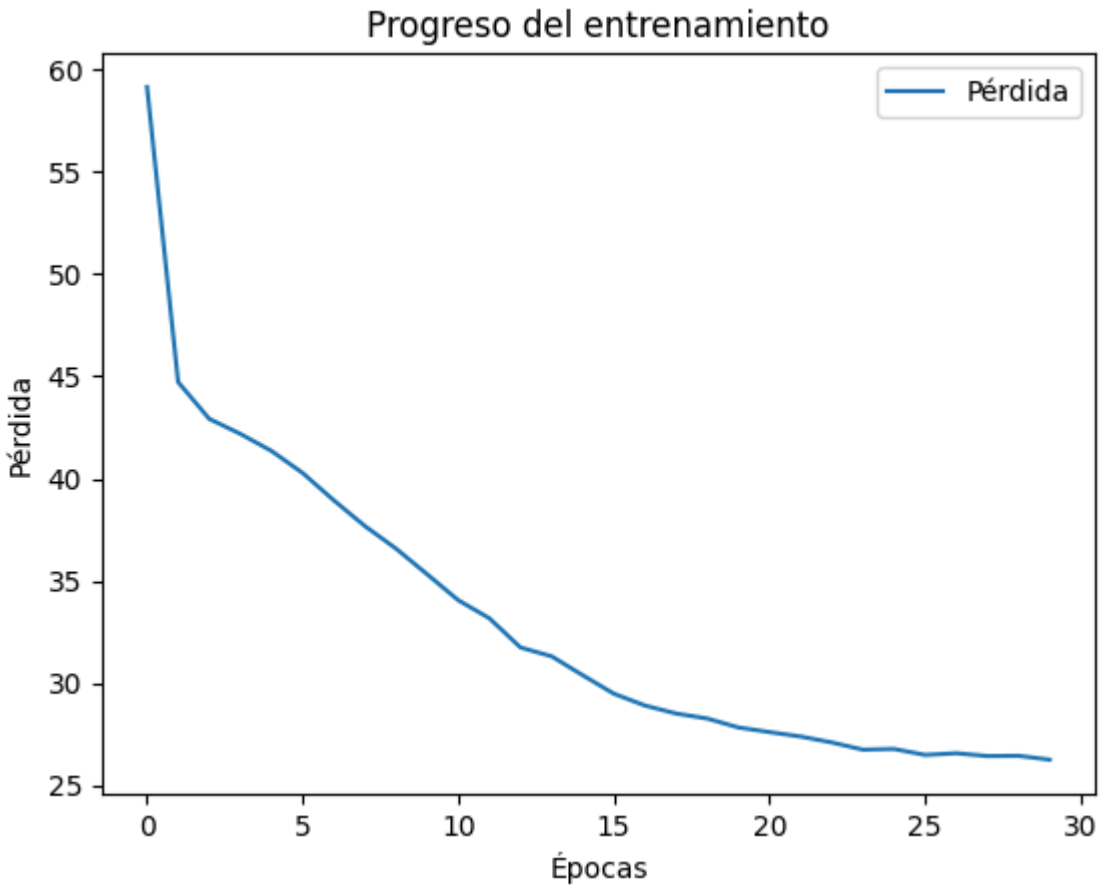
Vamos a complicar un poco más la arquitectura de la red:

- **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 64 neuronas y función de activación *relu*.
    - *3º Capa*: capa densa con 32 neuronas y función de activación *relu*.
    - *4º Capa*: capa densa con 32 neuronas y función de activación *relu*.
    - *5º Capa*: capa de salida con una neurona sin función de activación.
  - Tipo de entrenamiento:
    - *Epochs*: 30
    - *Optimizador*: *adam*
    - *Learning Rate*: 0.001

```
# Completar
import tensorflow as tf
from tensorflow.keras import Sequential
```

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Dimensiones de entrada (debes ajustar esto al tamaño real de tus datos)
input_dim = x.shape[1]  # Número de características en el dataset

# Crear el modelo
model = Sequential([
    Dense(64, activation='relu', input_dim=input_dim),  # Primera capa densa
    Dense(32, activation='relu'),                       # Segunda capa densa
    Dense(32, activation='relu'),                       # Tercera capa densa
    Dense(1)                                            # Capa de salida
])

# Compilar el modelo
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='mean_absolute_percentage_error')

# Entrenar el modelo
history = model.fit(x, y, epochs=30, batch_size=32)

# Mostrar resultados del entrenamiento
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Pérdida')
plt.title('Progreso del entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.show()
```

```
Epoch 1/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 71.9170
Epoch 2/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 45.7355
Epoch 3/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42.8489
Epoch 4/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 42.5594
Epoch 5/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 41.6067
Epoch 6/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 40.1998
Epoch 7/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 39.5649
Epoch 8/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 38.1421
Epoch 9/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 36.8974
Epoch 10/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 35.5005
Epoch 11/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 34.3799
Epoch 12/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 33.4564
Epoch 13/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 32.1475
Epoch 14/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 31.7491
Epoch 15/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 30.9301
Epoch 16/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 29.5960
Epoch 17/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 28.9120
Epoch 18/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 28.9022
Epoch 19/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 28.1124
Epoch 20/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 28.0163
Epoch 21/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 27.6690
Epoch 22/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 27.4763
Epoch 23/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 27.0234
Epoch 24/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 27.0351
Epoch 25/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 26.6481
Epoch 26/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 26.6680
Epoch 27/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 27.0122
Epoch 28/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 26.4274
Epoch 29/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 26.6397
Epoch 30/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 26.3121
```



Ejercicio 3

Vamos a complicar aun más la arquitectura de la red:

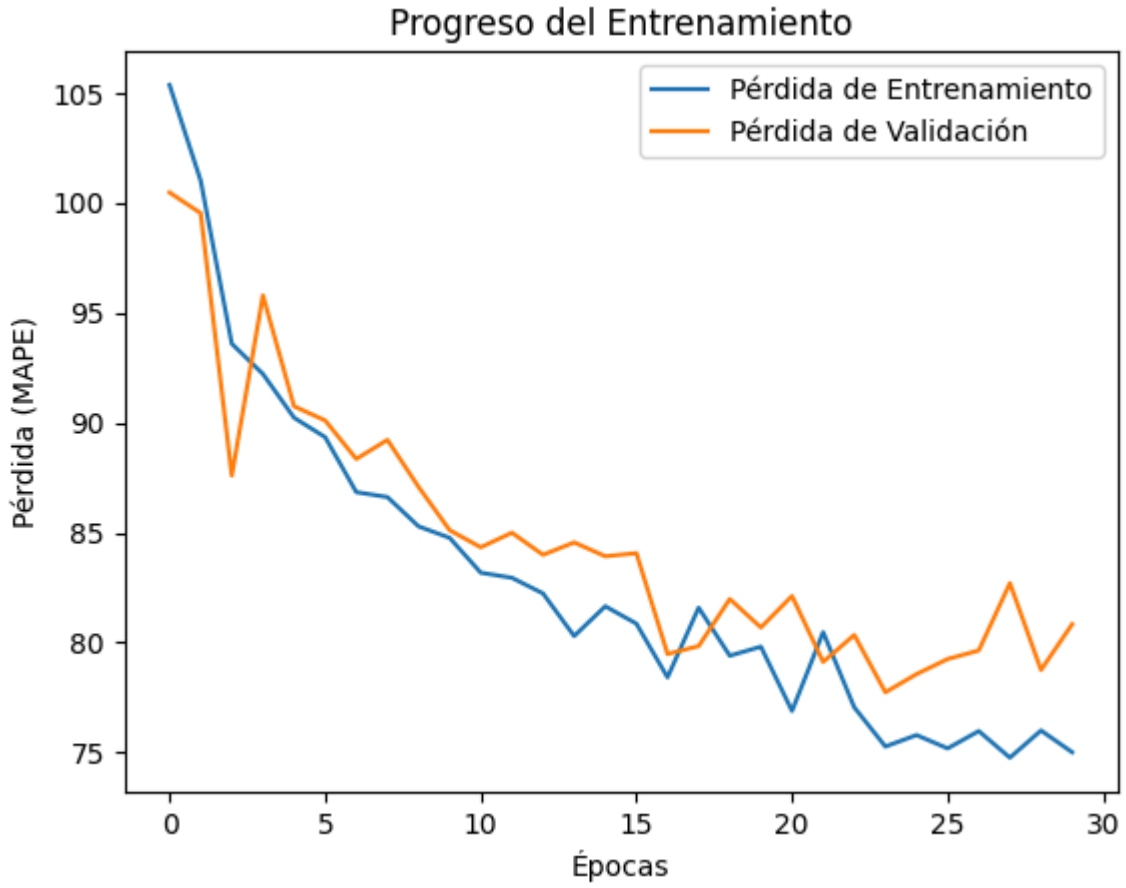
- **Configuración de la red**:
  - Arquitectura de la red:
    - *1º Capa*: capa de entrada donde indiques la dimensión de los datos.
    - *2º Capa*: capa densa con 128 neuronas y función de activación *relu*.
    - *3º Capa*: capa densa con 64 neuronas y función de activación *relu*.
    - *4º Capa*: capa densa con 32 neuronas y función de activación *relu*.
    - *5º Capa*: capa densa con 16 neuronas y función de activación *relu*.
    - *6º Capa*: capa de salida con una neurona sin función de activación.
  - Tipo de entrenamiento:
    - *Epochs*: 30
    - *Optimizador*: *adam*
    - *Learning Rate*: 0.001

```
# Completar
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Preprocesamiento de los datos
# Dividir datos en entrenamiento y prueba
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Escalar los datos para que el modelo entrene de manera óptima
scaler_x = StandardScaler()
scaler_y = StandardScaler()

x_train = scaler_x.fit_transform(x_train)
x_test = scaler_x.transform(x_test)

y_train = scaler_y.fit_transform(y_train)
y_test = scaler_y.transform(y_test)

# Dimensiones de entrada
input_dim = x_train.shape[1]  # Número de características en el dataset

# Crear el modelo
model = Sequential([
    Dense(128, activation='relu', input_dim=input_dim),  # Primera capa densa
    Dense(64, activation='relu'),                        # Segunda capa densa
    Dense(32, activation='relu'),                        # Tercera capa densa
    Dense(16, activation='relu'),                        # Cuarta capa densa
    Dense(1)                                             # Capa de salida
])

# Compilar el modelo
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='mean_absolute_percentage_error',
              metrics=['mae'])

# Entrenar el modelo
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=30, batch_

# Graficar el progreso del entrenamiento
plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.title('Progreso del Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida (MAPE)')
plt.legend()
plt.show()

# Evaluar el modelo

test_loss, test_mae = model.evaluate(x_test, y_test, verbose=0)
print(f"Pérdida en el conjunto de prueba: {test_loss:.4f}")
print(f"MAE en el conjunto de prueba: {test_mae:.4f}")
```

```
Epoch 1/30
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
511/511 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 105.6660 - mae: 0.7543 - val_loss: 1
Epoch 2/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 100.2302 - mae: 0.6970 - val_loss: 9
Epoch 3/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 94.1579 - mae: 0.6723 - val_loss: 87
Epoch 4/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 90.6172 - mae: 0.6433 - val_loss: 95
Epoch 5/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 89.8005 - mae: 0.6379 - val_loss: 90
Epoch 6/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 90.7950 - mae: 0.6507 - val_loss: 90
Epoch 7/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 86.3051 - mae: 0.5976 - val_loss: 88
Epoch 8/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 87.0998 - mae: 0.6062 - val_loss: 89
Epoch 9/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 85.9680 - mae: 0.5898 - val_loss: 87
Epoch 10/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 83.7443 - mae: 0.5703 - val_loss: 85
Epoch 11/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 82.5853 - mae: 0.5568 - val_loss: 84
Epoch 12/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 81.9649 - mae: 0.5384 - val_loss: 84
Epoch 13/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 83.7960 - mae: 0.5300 - val_loss: 83
Epoch 14/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 79.8709 - mae: 0.5043 - val_loss: 84
Epoch 15/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 82.8562 - mae: 0.5464 - val_loss: 83
Epoch 16/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 79.3382 - mae: 0.5127 - val_loss: 84
Epoch 17/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 79.2566 - mae: 0.5130 - val_loss: 79
Epoch 18/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 81.6219 - mae: 0.5239 - val_loss: 79
Epoch 19/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 79.4175 - mae: 0.4829 - val_loss: 81
Epoch 20/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 80.4794 - mae: 0.4978 - val_loss: 80
Epoch 21/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 77.6448 - mae: 0.5004 - val_loss: 82
Epoch 22/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 80.2556 - mae: 0.4974 - val_loss: 79
Epoch 23/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 76.2251 - mae: 0.5013 - val_loss: 80
Epoch 24/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 74.9300 - mae: 0.4773 - val_loss: 77
Epoch 25/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 75.7313 - mae: 0.4783 - val_loss: 78
Epoch 26/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 74.7509 - mae: 0.4704 - val_loss: 79
Epoch 27/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 74.3107 - mae: 0.4623 - val_loss: 79
Epoch 28/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 74.7342 - mae: 0.4622 - val_loss: 82
Epoch 29/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 74.6228 - mae: 0.4764 - val_loss: 78
```

```
Epoch 30/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 75.5304 - mae: 0.4805 - val_loss: 80
```




```
Pérdida en el conjunto de prueba: 80.8365
MAE en el conjunto de prueba: 0.4674
```

Ejercicio 4

Vamos a hacer una última red con más capas y neuronas:

- **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 1024 neuronas y función de activación *relu*.
    - *3º Capa*: capa densa con 512 neuronas y función de activación *relu*.
    - *4º Capa*: capa densa con 256 neuronas y función de activación *relu*.
    - *5º Capa*: capa densa con 128 neuronas y función de activación *relu*.
    - *6º Capa*: capa densa con 64 neuronas y función de activación *relu*.
    - *7º Capa*: capa densa con 32 neuronas y función de activación *relu*.
    - *8º Capa*: capa densa con 16 neuronas y función de activación *relu*.
    - *9º Capa*: capa de salida con una neurona sin función de activación.
  - Tipo de entrenamiento:
    - *Epochs*: 30
    - *Optimizador*: *adam*
    - *Learning Rate*: 0.001

```
# Completar
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Preprocesamiento de los datos
# Dividir datos en entrenamiento y prueba
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Escalar los datos para que el modelo entrene de manera óptima
scaler_x = StandardScaler()
scaler_y = StandardScaler()

x_train = scaler_x.fit_transform(x_train)
x_test = scaler_x.transform(x_test)

y_train = scaler_y.fit_transform(y_train)
y_test = scaler_y.transform(y_test)

# Dimensiones de entrada
input_dim = x_train.shape[1]  # Número de características en el dataset
```

```
# Crear el modelo
model = Sequential([
    Dense(1024, activation='relu', input_dim=input_dim),  # Primera capa densa
    Dense(512, activation='relu'),                        # Segunda capa densa
    Dense(256, activation='relu'),                        # Tercera capa densa
    Dense(128, activation='relu'),                        # Cuarta capa densa
    Dense(64, activation='relu'),                         # Quinta capa densa
    Dense(32, activation='relu'),                         # Sexta capa densa
    Dense(16, activation='relu'),                         # Séptima capa densa
    Dense(1)                                              # Capa de salida
])

# Compilar el modelo
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='mean_absolute_percentage_error',
              metrics=['mae'])

# Entrenar el modelo
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=30, batch_

# Graficar el progreso del entrenamiento
plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.title('Progreso del Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida (MAPE)')
plt.legend()
plt.show()

# Evaluar el modelo
test_loss, test_mae = model.evaluate(x_test, y_test, verbose=0)
print(f"Pérdida en el conjunto de prueba: {test_loss:.4f}")
print(f"MAE en el conjunto de prueba: {test_mae:.4f}")
```

```
Epoch 1/30
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 19ms/step - loss: 106.0785 - mae: 0.7392 - val_loss:
Epoch 2/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - loss: 102.9521 - mae: 0.7485 - val_loss:
Epoch 3/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 93.8309 - mae: 0.6757 - val_loss: 9
Epoch 4/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 19ms/step - loss: 97.8508 - mae: 0.7218 - val_loss:
Epoch 5/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - loss: 90.4019 - mae: 0.6505 - val_loss:
Epoch 6/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 16ms/step - loss: 95.7788 - mae: 0.7089 - val_loss: 9
Epoch 7/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 19ms/step - loss: 95.2882 - mae: 0.6996 - val_loss:
Epoch 8/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 89.8943 - mae: 0.6562 - val_loss: 9
Epoch 9/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - loss: 91.2323 - mae: 0.6537 - val_loss:
Epoch 10/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 87.1985 - mae: 0.6297 - val_loss:
Epoch 11/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 92.0641 - mae: 0.6934 - val_loss: 9
Epoch 12/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 88.1870 - mae: 0.6413 - val_loss:
Epoch 13/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 91.5315 - mae: 0.6463 - val_loss: 9
Epoch 14/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 17s 33ms/step - loss: 87.0160 - mae: 0.6403 - val_loss:
Epoch 15/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 15s 29ms/step - loss: 86.6818 - mae: 0.6284 - val_loss:
Epoch 16/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 86.2075 - mae: 0.6183 - val_loss:
Epoch 17/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 16ms/step - loss: 84.5305 - mae: 0.6154 - val_loss: 8
Epoch 18/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 86.6880 - mae: 0.6296 - val_loss:
Epoch 19/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 89.5147 - mae: 0.6563 - val_loss: 8
Epoch 20/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 16ms/step - loss: 86.2053 - mae: 0.6196 - val_loss: 8
Epoch 21/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - loss: 86.2468 - mae: 0.6211 - val_loss:
Epoch 22/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 11s 20ms/step - loss: 85.7208 - mae: 0.6020 - val_loss:
Epoch 23/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 16ms/step - loss: 87.2949 - mae: 0.6255 - val_loss: 8
Epoch 24/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 86.6421 - mae: 0.6249 - val_loss:
Epoch 25/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 17ms/step - loss: 85.6856 - mae: 0.6124 - val_loss: 8
Epoch 26/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - loss: 83.7986 - mae: 0.6041 - val_loss:
Epoch 27/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - loss: 88.6017 - mae: 0.6377 - val_loss:
Epoch 28/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 87.0644 - mae: 0.6258 - val_loss: 9
Epoch 29/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 86.6081 - mae: 0.6132 - val_loss:
```