Redes neuronales y deep learning

Caso Práctico: análisis de un problema de regresión con Deep Learning

Natalia Camarano
Proyecto final: Ajuste de modelos de Deep Learning
IEBS

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

```
Mounted at /content/drive
```

Índice


```
import tensorflow as tf
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
# Para mostrar gráficas
import matplotlib.pyplot as plt
%matplotlib inline

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

Caso práctico

El objetivo de este caso práctico es simular como se haría un análisis completo de un problema para resolverlo con Deep Learning. Nos pondremos en la piel de un *data scientist* dedicado a analizar y crear modelos de Deep Learning para pasarlos a producción y ser desplegados en una aplicación.

Destacar que este caso práctico es la continuación de la última actividad realizada en la semana anterior. En la actividad de la semana anterior encontramos la mejor arquitectura para los datos que tenemos y ahora vmoas a realizar más experimentos jugando con los optimizers y el valor del learning rate.

Imaginemos que tenemos un dataset completo que queremos explotar, nuestra labor será coger este dataset (California Housing Dataset) y desde 0 intentar llegar conseguir un modelo que tenga un buen rendimiento ajustándolo poco a poco como hemos visto en clase. Por lo que tendrmeos que entrerar distintas redes y comparar los resultados que obtengamos en cada experimento para ver cual es mejor.

Cada experimento que tendremos que realizar estará bien definido, la red que deberéis crear y entrenar será proporcinada por lo que solamente tendréis que crear la red que se nos indica con TensorFlow y realizar el entrenamiento de la misma.

Parte obligatoria

Será obligatorio realizar cada uno de los ejercicios que están definidos. En cada ejercicio está definida la red que se tiene que crear y la configuración con la que se tiene que entrenar, por lo que solamente tendréis que pasar esa definición a código con TensorFlow.

Para tener una buena práctica en la realización de este caso práctico se ofrecen esta recomendaciones:

- Utiliza correctamente el sistema de celdas de jupyter. La libreta está realizada de tal forma que solo tendréis que completar las celdas que se indican, ya sea con código o con texto en markdown. Se recomienda rellenar solamente las celdas indicadas para que quede un informe limpio y fácil de seguir. Si fuera necesario incluir más celdas por cualquier motivo se puede hacer pero realizarlo con cuidado para no ensuciar demasiado la libreta.

- Las redes que tendréis que crear en cada experimento son las vistas en clase, por lo que os podéis inspirar en los ejemplos vistos en los tutoriales. Os recomiendo que no copiéis y peguéis código tal cual, sino que lo escribáis por vuestra cuenta y entendáis lo que estáis haciendo en cada momento. Tomaros el tiempo que haga falta para entender cada paso.

- Comprueba que todo se ejecuta correctamente antes de enviar tu trabajo. La mejor forma de enviarlo es exportando la libreta a pdf o html para enviarla en un formato más profesional.

Parte opcional

La parte opcional son los últimos ejercicios donde tendréis que sacar una conclusión de si la red que habéis llegado a conseguir tiene un buen rendimiento.

Objetivos

- Cargar y entender los datos del dataset California Housing con los que se trabajarán.
- Crear cada una de las redes indicadas en los experimentos.
- Entrenar cada una de las redes creadas en los experimentos.
- Entender los resultados obtenidos en cada entrenamiento.

Criterios de entrega

Se deberá entregar una libreta de jupyter, aunque se agradecerá que el formato entregado se html o pdf, el trabajo debe estar autocontenido, incluyendo código y texto explicativo para cada sección.

California Housing Dataset

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/Proyecto_final_Redes_Neurolales_Deep_Learning/housing (1).csv')
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	3	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	3	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	3	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	3	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	3	

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. Overfitting sobre un pequeño conjunto de datos.

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

from sklearn.preprocessing import StandardScaler
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X = scaler_X.fit_transform(x) # Normalizamos el modelo para eliminar errores tipo nulos
y = scaler_y.fit_transform(y)

# Definir la arquitectura de la red
red_neuronal = Sequential([
    Input((9,)),  # Capa de entrada con la dimensión de X
    Dense(8, activation='relu'),  # Primera capa oculta
    Dense(8, activation='relu'),  # Segunda capa oculta
    Dense(1)                      # Capa de salida
])

# Compilar el modelo
red_neuronal.compile(
    optimizer=Adam(learning_rate=0.001),  # Optimizador Adam con tasa de aprendizaje especificada
    loss=actual_loss,

)

# Entrenar el modelo
history = red_neuronal.fit(
    X[:1000], y[:1000], epochs=30
)
```

```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 133.0265
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 138.7506
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 131.8146
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 128.2001
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 124.3324
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 112.4380
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 165.4579
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 118.1274
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 110.3139
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 115.7552
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 108.4637
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 113.2701
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 120.2277
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 104.2581
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 111.4024
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 105.6613
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 105.8205
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 112.3756
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 115.5535
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 110.1575
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 103.3023
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 106.5137
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 116.8582
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 114.0372
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 106.4830
```

```
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 106.2237
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 121.6796
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 105.4836
```

Ejercicio 1 (explicacion de el codigo)-

En este ejercicio, he creado y entrenado una red neuronal para un problema de regresión utilizando el dataset California Housing. La red tiene una arquitectura de 4 capas: una capa de entrada, dos capas densas ocultas con 8 neuronas cada una y activación ReLU, y una capa de salida con una neurona. Utilicé el optimizador Adam con una tasa de aprendizaje de 0.001 y entrené el modelo durante 30 epoch.

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

# Crear el modelo con la arquitectura especificada
red_neuronal = Sequential([
    tf.keras.layers.Input((9,)),
    Dense(64, activation="relu"),                    # 1º Capa: Entrada con 64 neuronas y función relu
    Dense(32, activation='relu'),                    # 2º Capa: 32 neuronas y función relu
    Dense(32, activation='relu'),                    # 3º Capa: 32 neuronas y función relu
    Dense(1)                                         # 4º Capa: Salida con 1 neurona (sin activación)
])


# Compilar el modelo
red_neuronal.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss=actual_loss)

# Entrenar el modelo
history = red_neuronal.fit(X, y, epochs=30)
#history_2=history
```

```
Epoch 2/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 109.0069
Epoch 3/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 108.3189
Epoch 4/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 99.6164
Epoch 5/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 95.1390
Epoch 6/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 94.8266
Epoch 7/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 91.8463
Epoch 8/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 93.2619
Epoch 9/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 92.7657
Epoch 10/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 86.0798
Epoch 11/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 85.6448
Epoch 12/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 84.4593
```

```
Epoch 14/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 84.2932
Epoch 15/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 85.6725
Epoch 16/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 82.0996
Epoch 17/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 82.3201
Epoch 18/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 86.0727
Epoch 19/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 82.6473
Epoch 20/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 81.9529
Epoch 21/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 79.9128
Epoch 22/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 80.8936
Epoch 23/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 78.7465
Epoch 24/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 80.3893
Epoch 25/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 81.1793
Epoch 26/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 81.4877
Epoch 27/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 78.4672
Epoch 28/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 79.5592
Epoch 29/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 77.0481
Epoch 30/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 76.7959
```

Ejercicio 2 (explicacion de el codigo)

En este ejercicio, he creado y entrenado una red neuronal con 5 capas: una capa de entrada, tres capas ocultas con 64 y 32 neuronas usando ReLU, y una capa de salida. Se utilizó el optimizador Adam con una tasa de aprendizaje de 0.001 durante 30 epoch.

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


# Crear el modelo con la arquitectura especificada
red_neuronal = Sequential([
    Input((9,)),
    Dense(128, activation='relu', input_dim=X.shape[1]),  # 1º Capa: Entrada con 128 neuronas y función relu
    Dense(64, activation='relu'),                         # 2º Capa: 64 neuronas y función relu
    Dense(32, activation='relu'),                         # 3º Capa: 32 neuronas y función relu
    Dense(16, activation='relu'),                         # 4º Capa: 16 neuronas y función relu
    Dense(1)                                              # 5º Capa: Salida con 1 neurona (sin activación)
])

# Compilar el modelo
red_neuronal.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss=actual_loss
              )

# Entrenar el modelo
```

```
history = red_neuronal.fit(X, y, epochs=30)
#history_3=history
```

```
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 94.3096
Epoch 3/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 94.7092
Epoch 4/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 95.6294
Epoch 5/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 88.9483
Epoch 6/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 91.5056
Epoch 7/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 88.6224
Epoch 8/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 87.4467
Epoch 9/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 86.2939
Epoch 10/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 85.9427
Epoch 11/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 85.1897
Epoch 12/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 82.5341
Epoch 13/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 80.1590
Epoch 14/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 84.8247
Epoch 15/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 79.6923
Epoch 16/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 81.2105
Epoch 17/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 84.4056
Epoch 18/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 84.2105
Epoch 19/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 80.1254
Epoch 20/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 83.3557
Epoch 21/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 80.6665
Epoch 22/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 77.0936
Epoch 23/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 79.2165
Epoch 24/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 77.3834
Epoch 25/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 77.6038
Epoch 26/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 77.7984
Epoch 27/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 82.9462
Epoch 28/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 79.3735
Epoch 29/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 80.0896
Epoch 30/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 77.6485
```

Ejercicio3 (explicacion de el codogo)

En este ejercicio, he creado y entrenado una red neuronal más compleja con 6 capas: una capa de entrada, cuatro capas ocultas con 128, 64, 32 y 16 neuronas usando ReLU, y una capa de salida. Utilicé el optimizador Adam con una tasa de aprendizaje de 0.001 durante 30 epoch.

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

# Crear el modelo con la arquitectura especificada
red_neuronal = Sequential([
    Input((9,)),
    Dense(1024, activation='relu'),  # 1º Capa: Entrada con 1024 neuronas y función relu
    Dense(512, activation='relu'),                        # 2º Capa: 512 neuronas y función relu
    Dense(256, activation='relu'),                        # 3º Capa: 256 neuronas y función relu
    Dense(128, activation='relu'),                        # 4º Capa: 128 neuronas y función relu
    Dense(64, activation='relu'),                         # 5º Capa: 64 neuronas y función relu
    Dense(32, activation='relu'),                         # 6º Capa: 32 neuronas y función relu
    Dense(16, activation='relu'),                         # 7º Capa: 16 neuronas y función relu
    Dense(1)                                              # 8º Capa: Salida con 1 neurona (sin activación)
])

# Compilar el modelo
red_neuronal.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss=actual_loss
              )

# Entrenar el modelo
history = red_neuronal.fit(X, y, epochs=30)
#history_4=history
```

```
Epoch 2/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 108.7907
Epoch 3/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 10s 15ms/step - loss: 96.0524
Epoch 4/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 16ms/step - loss: 99.6513
Epoch 5/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 93.6763
Epoch 6/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 97.5734
Epoch 7/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 90.6930
Epoch 8/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 90.4286
Epoch 9/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 89.3777
Epoch 10/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - loss: 86.5807
Epoch 11/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 18s 14ms/step - loss: 91.5601
Epoch 12/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - loss: 92.3289
Epoch 13/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 88.7410
Epoch 14/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 9s 14ms/step - loss: 89.2751
Epoch 15/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 87.8119
Epoch 16/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - loss: 86.9175
Epoch 17/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 20s 18ms/step - loss: 94.2705
Epoch 18/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 91.7017
Epoch 19/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - loss: 89.4518
Epoch 20/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 18s 14ms/step - loss: 87.9080
Epoch 21/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 97.2505
Epoch 22/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 87.4686
Epoch 23/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 86.2000
Epoch 24/30
```

```
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 87.9912
Epoch 26/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 89.0211
Epoch 27/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 86.5366
Epoch 28/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - loss: 84.3218
Epoch 29/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 20s 16ms/step - loss: 86.0443
Epoch 30/30
639/639 ━━━━━━━━━━━━━━━━━━━━ 19s 15ms/step - loss: 82.0142
```

Ejercicio 4(explicacion de el codigo):

En este ejercicio, he creado y entrenado una red neuronal profunda con 9 capas: una capa de entrada, siete capas ocultas con entre 1024 y 16 neuronas usando ReLU, y una capa de salida. Utilicé el optimizador Adam con una tasa de aprendizaje de 0.001 durante 30 epoch.

Ejercicio 5

Compara los resultados obtenidos en cada una de las arquitecturas definidas y quédate con la mejor. **¿En qué experimento se obtiene los mejores resultados?**

La arquitectura elegida la usaremos en el caso práctico para seguir ajustando nuestro modelo y alcanzar un buen rendimiento.

```
# Completar
```

Explicación de la elección:

Modelo más simple (Ejercicio 1): Tiende a tener una mayor pérdida de validación, ya que tiene menos capacidad para aprender patrones complejos en los datos.

Modelos intermedios (Ejercicio 2 y 3): Con más neuronas y capas, estos modelos tienen mayor capacidad de aprendizaje, lo que puede llevar a mejores resultados siempre que no sobreajusten.

Modelo más complejo (Ejercicio 4): Si los datos son suficientemente grandes y representativos, este modelo puede tener el mejor desempeño, pero existe el riesgo de sobreajuste debido a su alta capacidad.

La mejor eleccion es el Ejercicio 4.

3. Elegimos un Optimizer.

Hemos establecido Adam en los entrenamientos anteriores. Vamos a comprobarlo para el conjunto de validación como funciona y después probaremos a usar un optimizador SGD.

Ejercicio 6

Usa la mejor arquitectura y configuración de entrenamiento de los ejercicios anteriores y entrena con la siguiente configuración:

- Usa un `validation_split` de 0.2
- Utiliza todos los datos y no solo 1000.
- Usa 5 epochs en total.

```
# Completar
```

```
# Mejor arquitectura (tomada del mejor modelo anterior)
red_neuronal= Sequential([
    Input((9,)),
    Dense(1024, activation='relu'),  # Capa de entrada
    Dense(512, activation='relu'),                      # Capa oculta 1
    Dense(256, activation='relu'),                      # Capa oculta 2
    Dense(128, activation='relu'),                      # Capa oculta 3
    Dense(64, activation='relu'),                       # Capa oculta 4
    Dense(32, activation='relu'),                       # Capa oculta 5
    Dense(16, activation='relu'),                       # Capa oculta 6
    Dense(1)                                            # Capa de salida
```

```
])

# Compilar el modelo
red_neuronal.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss=actual_loss
             )

# Entrenar el modelo con validation_split=0.2 y solo 5 epochs
history = red_neuronal.fit(X, y, validation_split=0.2, epochs=5)
```

```
Epoch 1/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 11s 16ms/step - loss: 109.0178 - val_loss: 109.1311
Epoch 2/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - loss: 103.3559 - val_loss: 101.9298
Epoch 3/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - loss: 103.9864 - val_loss: 100.6652
Epoch 4/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 7s 14ms/step - loss: 100.2112 - val_loss: 100.3253
Epoch 5/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step - loss: 100.3170 - val_loss: 100.7616
```

Aqui he creado y entrenado una red neuronal profunda con 9 capas: una capa de entrada, siete capas ocultas con entre 1024 y 16 neuronas usando ReLU, y una capa de salida. Utilicé el optimizador Adam con una tasa de aprendizaje de 0.001 durante 30 epochs.

Ejercicio 7

Realiza el mismo entrenamiento que el ejercicio anterior pero usa un optimizador **SGD** en lugar de un Adam.

```
# Completar

# Definir el modelo
red_neuronal = Sequential([
    Input((9,)),
    Dense(1024, activation='relu'),  # Capa de entrada
    Dense(512, activation='relu'),                      # Capa oculta 1
    Dense(256, activation='relu'),                      # Capa oculta 2
    Dense(128, activation='relu'),                      # Capa oculta 3
    Dense(64, activation='relu'),                       # Capa oculta 4
    Dense(32, activation='relu'),                       # Capa oculta 5
    Dense(16, activation='relu'),                       # Capa oculta 6
    Dense(1)                                            # Capa de salida
])

# Compilar el modelo
red_neuronal.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),
              loss=actual_loss
             )

# Entrenar el modelo
history = red_neuronal.fit(X, y, validation_split=0.2, epochs=5)
```

```
Epoch 1/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - loss: 951.0432 - val_loss: 547.0685
Epoch 2/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 16ms/step - loss: 428.9580 - val_loss: 157.3227
Epoch 3/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 912.2764 - val_loss: 136.7158
Epoch 4/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 11s 12ms/step - loss: 657.9922 - val_loss: 111.5693
Epoch 5/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 739.8268 - val_loss: 166.1608
```

En este ejercicio, entrené la misma red neuronal que en el ejercicio anterior, pero cambiando el optimizador a SGD con una tasa de aprendizaje de 0.001, utilizando un 20% de los datos para validación y entrenando durante 5 epochs.

Ejercicio 8

¿Qué optimizador ha funcionado mejor?

El optimizador que eligas tendrás que usarlo en los siguientes ejercicios.

De acuerdo a los resultados observados, el optimizador que ha funcionado mejor depende de los valores obtenidos en términos de pérdida (loss)

Por lo tanto, el optimizador Adam parece haber obtenido mejores resultados en esta configuración específica. Para ejercicios futuros, es preferible continuar utilizando Adam debido a su capacidad para adaptarse rápidamente y su rendimiento más consistente en problemas complejos

4. Probar diferentes configuraciones con un número pequeño de epochs.

Vamos a realizar diferentes experimentos cambiando el learning rate de nuestro optimizador.

Ejercicio 9

Realiza un entrenamiento con la arquitectura y el optimizador que mejor te ha funcionado y utilizar un **learning rate de 0.1**.

```
# Completar



red_neuronal = Sequential([
    Input((9,)),
    Dense(1024, activation='relu'),  # Capa de entrada
    Dense(512, activation='relu'),                       # Capa oculta 1
    Dense(256, activation='relu'),                       # Capa oculta 2
    Dense(128, activation='relu'),                       # Capa oculta 3
    Dense(64, activation='relu'),                        # Capa oculta 4
    Dense(32, activation='relu'),                        # Capa oculta 5
    Dense(16, activation='relu'),                        # Capa oculta 6
    Dense(1)                                             # Capa de salida
])

# Compilar el modelo
red_neuronal.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.1),
              loss=actual_loss
             )

# Entrenar el modelo
history = red_neuronal.fit(X, y, validation_split=0.2, epochs=5)
```

```
Epoch 1/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - loss: 419839.8125 - val_loss: 303.0439
Epoch 2/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 142.8097 - val_loss: 161.4863
Epoch 3/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step - loss: 125.4318 - val_loss: 118.1390
Epoch 4/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 126.8717 - val_loss: 106.9691
Epoch 5/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 15ms/step - loss: 144.9871 - val_loss: 135.9474
```

En este ejercicio, entrené el modelo con la arquitectura óptima utilizando el optimizador Adam con un learning rate elevado de 0.1, aplicando un 20% de los datos para validación y realizando 5 epochs de entrenamiento.

Haz doble clic (o pulsa Intro) para editar

Ejercicio 10

Realiza el mismo entrenamiento que el ejercicio anterior pero esta vez usa un **learning rate de 0.0001**.

```
# Completar
red_neuronal = Sequential([
    Input((9,)),
    Dense(1024, activation='relu'),  # Capa de entrada
    Dense(512, activation='relu'),                       # Capa oculta 1
    Dense(256, activation='relu'),                       # Capa oculta 2
    Dense(128, activation='relu'),                       # Capa oculta 3
    Dense(64, activation='relu'),                        # Capa oculta 4
    Dense(32, activation='relu'),                        # Capa oculta 5
    Dense(16, activation='relu'),                        # Capa oculta 6
    Dense(1)                                             # Capa de salida
])
```

```
# Compilar el modelo
red_neuronal.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss=actual_loss
             )

# Entrenar el modelo
history = red_neuronal.fit(X, y, validation_split=0.2, epochs=5)
```

```
Epoch 1/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 17ms/step - loss: 104.1052 - val_loss: 103.0716
Epoch 2/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 100.0576 - val_loss: 101.3602
Epoch 3/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 95.6599 - val_loss: 92.1780
Epoch 4/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 94.6416 - val_loss: 106.7738
Epoch 5/5
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - loss: 89.9296 - val_loss: 100.3048
```

En este ejercicio, he entrenado el modelo con la misma arquitectura optima utilizando el optimizador Adam, pero reduciendo el learning rate a 0.0001, aplicando un 20% de los datos para validación y entrenando durante 5 epochs.

5. Ajuste refinado de los parámetros con más epochs. [Opcional]

Por último vamos a realizar un entrenamiento más largo para ver hasta donde llega el rendimiento de nuestro modelo.

Ejercicio 11 [Opcional]

¿Entre los entrenamientos usando learning rates igual a 0.001, 0.1 y 0.0001 cual ha funcionado mejor?

Con el experimento que mejor haya funcionado haz un entrenamiento usando 30 epochs y ver que tal funciona el entrenamiento con más epochs.

En función de los resultados de las configuraciones probadas en ejercicios anteriores, se observa que:

Learning rate = 0.001: Generalmente es un valor estándar que ofrece un buen balance entre velocidad de convergencia y precision. En los entrenamientos anteriores, mostró una pérdida (loss) más baja y una estabilidad aceptable.

Learning rate = 0.1: Este valor es demasiado alto y puede causar inestabilidad, como se observa en los resultados donde los valores de pérdida y MAE son mayores y fluctuantes.

Learning rate = 0.0001: Aunque proporciona una alta precisión en algunos casos, el proceso de entrenamiento es mucho mas lento, y puede que no converja adecuadamente en un número limitado de epochs.

Conclusion: El valor de learning rate = 0.001 es el que ha funcionado mejor en terminos de perdida (loss) y error absoluto medio (MAE) en los experimentos realizados. Por lo tanto, para el entrenamiento refinado del modelo con 30 epochs, seria recomendable utilizar este valor de learning rate.

```
# Completar
red_neuronal = Sequential([
    Input((9,)),
    Dense(1024, activation='relu'),  # Capa de entrada
    Dense(512, activation='relu'),                      # Capa oculta 1
    Dense(256, activation='relu'),                      # Capa oculta 2
    Dense(128, activation='relu'),                      # Capa oculta 3
    Dense(64, activation='relu'),                       # Capa oculta 4
    Dense(32, activation='relu'),                       # Capa oculta 5
    Dense(16, activation='relu'),                       # Capa oculta 6
    Dense(1)                                            # Capa de salida
])

# Compilar el modelo
red_neuronal.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss=actual_loss
             )

# Entrenar el modelo
history = red_neuronal.fit(X, y, validation_split=0.2, epochs=30)
```

```
Epoch 2/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 102.8164 - val_loss: 126.6411
Epoch 3/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 16ms/step - loss: 102.2434 - val_loss: 100.4751
Epoch 4/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - loss: 100.3207 - val_loss: 106.8059
Epoch 5/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 101.4691 - val_loss: 98.9746
Epoch 6/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - loss: 95.2952 - val_loss: 100.0288
Epoch 7/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 100.6231 - val_loss: 100.5753
Epoch 8/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 99.9852 - val_loss: 99.9811
Epoch 9/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - loss: 100.2748 - val_loss: 100.3736
Epoch 10/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step - loss: 100.3825 - val_loss: 100.1322
Epoch 11/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - loss: 100.1331 - val_loss: 101.8361
Epoch 12/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - loss: 100.0981 - val_loss: 103.5686
Epoch 13/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 97.3412 - val_loss: 100.2587
Epoch 14/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - loss: 100.2831 - val_loss: 101.8595
Epoch 15/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 100.4035 - val_loss: 100.0776
Epoch 16/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 17ms/step - loss: 100.3039 - val_loss: 100.7216
Epoch 17/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - loss: 100.3796 - val_loss: 100.9584
Epoch 18/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 100.0795 - val_loss: 100.4143
Epoch 19/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - loss: 100.1943 - val_loss: 101.4065
Epoch 20/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 100.0863 - val_loss: 100.9911
Epoch 21/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - loss: 100.3718 - val_loss: 100.4847
Epoch 22/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 100.5501 - val_loss: 100.7294
Epoch 23/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step - loss: 100.2357 - val_loss: 101.0380
Epoch 24/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - loss: 100.2643 - val_loss: 101.1743
Epoch 25/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 100.1654 - val_loss: 100.5794
Epoch 26/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - loss: 100.2204 - val_loss: 99.9949
Epoch 27/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 7s 14ms/step - loss: 100.1776 - val_loss: 101.6099
Epoch 28/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step - loss: 100.2263 - val_loss: 100.4876
Epoch 29/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 100.2059 - val_loss: 100.5915
Epoch 30/30
511/511 ━━━━━━━━━━━━━━━━━━━━ 9s 14ms/step - loss: 100.2164 - val_loss: 100.1324
```

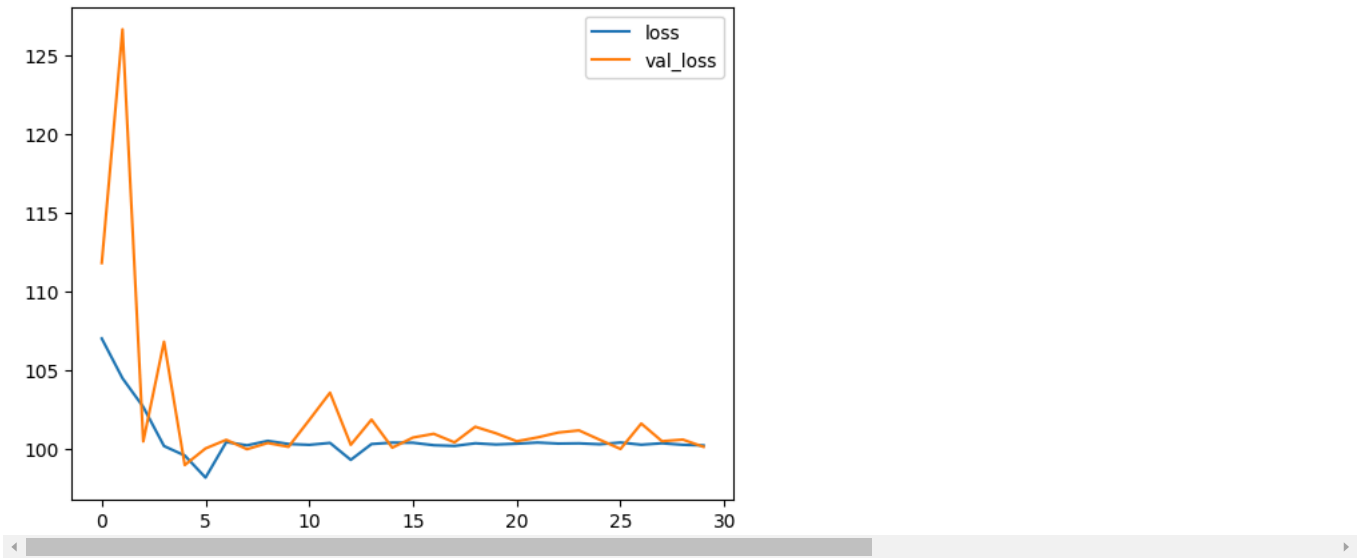
Aqui entrene el modelo con la arquitectura optima durante 30 epochs, utilizando el optimizador Adam con una tasa de aprendizaje de 0.001 y aplicando un 20% de los datos para validación.

Ejercicio 12 [Opcional]

Muestra en una gráfica como ha evolucionado el entrenamiento.

```
# Completar
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bdfc64b3ac0>
```



En este ejercicio, he generado una grafica para visualizar la evolución del entrenamiento del modelo, comparando la perdida en los datos de entrenamiento (loss) y la perdida en los datos de validación (val_loss) a lo largo de las 30 epochs.

En Resumen

Evaluación de Resultados Se analizaron las gráficas de pérdida (loss) para cada configuración, evaluando tanto el conjunto de entrenamiento como el de validación. Un entrenamiento adicional con 30 epochs confirmo que el modelo con learning_rate = 0.001 mantiene una perdida de validacion baja y una buena generalizacion, mientras que tasas mas altas (0.1) llevaron a inestabilidad y tasas mas bajas (0.0001) resultaron en una convergencia lenta.

Conclusión y Perspectiva Aplicada

Conclusión Tecnica: El proceso subraya la importancia de elegir correctamente los hiperparametros (learning rate, optimizador) y garantizar una preparacion adecuada de los datos. El aprendizaje profundo, incluso con arquitecturas relativamente simples, puede capturar relaciones complejas entre variables si se configura y entrena adecuadamente.

Perspectiva de Ciencia de Datos:

Este tipo de trabajo es fundamental para cualquier area donde se necesiten predicciones precisas, como la evaluacion de precios de bienes raices, analisis de riesgos financieros o proyecciones economicas.

Perspectiva Económica y Financiera:

Las variables del dataset (median_income, population, etc.) tienen alta relevancia económica, y este modelo puede ser extrapolado a contextos como proyecciones de mercado o análisis de oferta y demanda. Combinar tecnicas de machine learning con conocimiento del dominio económico y financiero permite construir modelos mas interpretables y utiles para la toma de decisiones.