Máster en Inteligencia Artificial Aplicada

Unidad: Deep Learning - Caso Práctico 1 / Ejercicio1_California Housing

Nombre: Patricio Galván

Fecha: 20 de Septiembre de 2024

Contenido:

Comentarios

Se ha desarrollado el ejercicio completamente, sin errores.
Se ha realizado un breve analisis al dataset, pero sin modificar ouliers.
Ya que Keras no tiene incorporada la funcion de perdida RMSE, por simplicidad ocuparemos MSE que es basicamente el cuadrado de RMSE.

Analisis Final

- Hemos comenzado por una arquitectura simple, para tener un modelo base con el que poder comparar.
- Posteriormente hemos realizado varios entrenamiento comparando en forma aislada diferentes hiperparametros, como # capas, # neuronas, funciones de activacion, optimizacion, etc.
- Se pudo observar que el tipo de funciones de activacion presenta fuertes influencias, tanto en la presicion del resultado, como en el desempeño del entrenamiento, esto es rapidez y forma de converger.
- Finalmente nos hemos quedado con las mejores configuraciones que encontramos y generamos un modelo. A este modelo le hemos agregado el metodo dropout para ver su efecto(debiera evitar el sobre ajuste).
  Los resultados empeoraron en comparacion al modelo Base:

Modelo Base

- Métrica Entrenamiento Validacion Diferencia (%)
- MSE 0.3074 0.3224 -4.88
- MAE 0.3858 0.3956 -2.54
- R2 0.7540 NaN NaN

Modelo Final

- Métrica Entrenamiento Validacion Diferencia (%)
- MSE 0.3353 0.3233 3.58
- MAE 0.4052 0.3974 1.92
- R2 0.7533 NaN

Al incorporar el metodo dropout algunos parametros empeoraron, como Loss y MAE, sin embargo el R2 se mantuvo similar. Lo que es rescatable de observar es que los datos de validacion fueron mejores que los de entrenamiento. Esto se debe al efecto Dropout.

Conclusion

- Logramos encontrar una configuracion que funcionó similar a al modelo Base inicial, pero incorporando dropout, lo que es una mejoria en el modelo.
- Pudimos comparar diferentes arquitecturas e hiperparametros de redes neuronales.
- Encontramos un modelo que se ajusta bastante bien a los datos.
- Lo que queda sería probar incorporar metodos como Normalizacion L2 para evitar tener Sobreajuste.

Se propone el siguiente modelo para obtener mejores resultados, no obstante no lo haremos pues significaria mucho tiempo de computo y el objetivo de este trabajo se cumplio.

Arquitectura Propuesta

- Capa entrada: 8 n.
- Capa oculta1: 64 n. (Funcion de Activación : tanh) + Dropout
- Capa oculta2: 64 n. (Funcion de Activación : tanh)
- Capa Salida: 1 n. sin funcion de activación
- epochs: 70

- batch_size: 128
- optimizer: Nadam(learning_rate=0.001)
- loss: 'mean_squared_error'

Descripcion Data Set California Housing

El dataset California Housing proviene de un estudio realizado por el Departamento de Desarrollo de Vivienda y Comunidad de California, con datos tomados del censo de 1990.

Descripción del dataset:

- Propósito: Predecir el valor medio de las viviendas en distintos bloques de California.
- Tipo de problema: Regresión.
- Estructura del dataset: El dataset tiene 20,640 instancias (filas) y 8 características (columnas), además de la variable objetivo (median_house_value).

Variables/Características:

- MedInc: Ingresos medios de los hogares (en decenas de miles de dólares).
- HouseAge: Edad media de las viviendas en un bloque (en años).
- AveRooms: Promedio de habitaciones por vivienda en el bloque.
- AveBedrms: Promedio de dormitorios por vivienda en el bloque.
- Population: Población del bloque.
- AveOccup: Promedio de ocupantes por vivienda en el bloque.
- Latitude: Latitud geográfica del bloque.
- Longitude: Longitud geográfica del bloque.

Variable objetivo (Target):

- MedianHouseValue: Valor medio de las viviendas en el bloque (en cientos de miles de dólares).

Otras Consideraciones:

La variable objetivo median_house_value, está escaladas en cientos de miles de dólares.

Este dataset es ideal para proyectos de aprendizaje automático de nivel intermedio, permitiendo experimentar con modelos de regresión lineal, árboles de decisión, redes neuronales y más.

1.- Carga de Librerias

```
import numpy as np
import pandas as pd
import random

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt

# Fijar semillas para poder reproducir resultados
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

2.- Carga y Analisis del Dataset

Importamos el dataset, que está disponible en la librería de scikitlearn. Tras ello, separamos en train y test y escalamos los datos - recordad aquí que las redes neuronales funcionan mejor con valores centrados en 0.

Nota: los valores de las etiquetas no los normalizamos porque ya vienen normalizados en el propio dataset.

Capa de Entrada: El dataset de California Housing tiene 8 características (variables de entrada), por lo que la capa de entrada tendrá 8 neuronas

Capa Oculta: Tendrás una capa oculta con 32 neuronas, como mencionaste.

Capa de Salida: Este dataset normalmente se utiliza para predicción de precios de casas (un valor continuo). Por tanto, la capa de salida tendrá 1 neurona.

Función de Activación de la Última Capa: Para un problema de regresión, como es la predicción de precios de casas, la función de activación de la última capa debe ser una función lineal (sin activación, es decir, una identidad).

Veamos una descripcion de los Datos de Californiа Housing

```
# Cargamos el conjunto de datos de California Housing
california_housing = fetch_california_housing(as_frame=True)

# Cargamos la descripcion del dataset
print(california_housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surprisingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

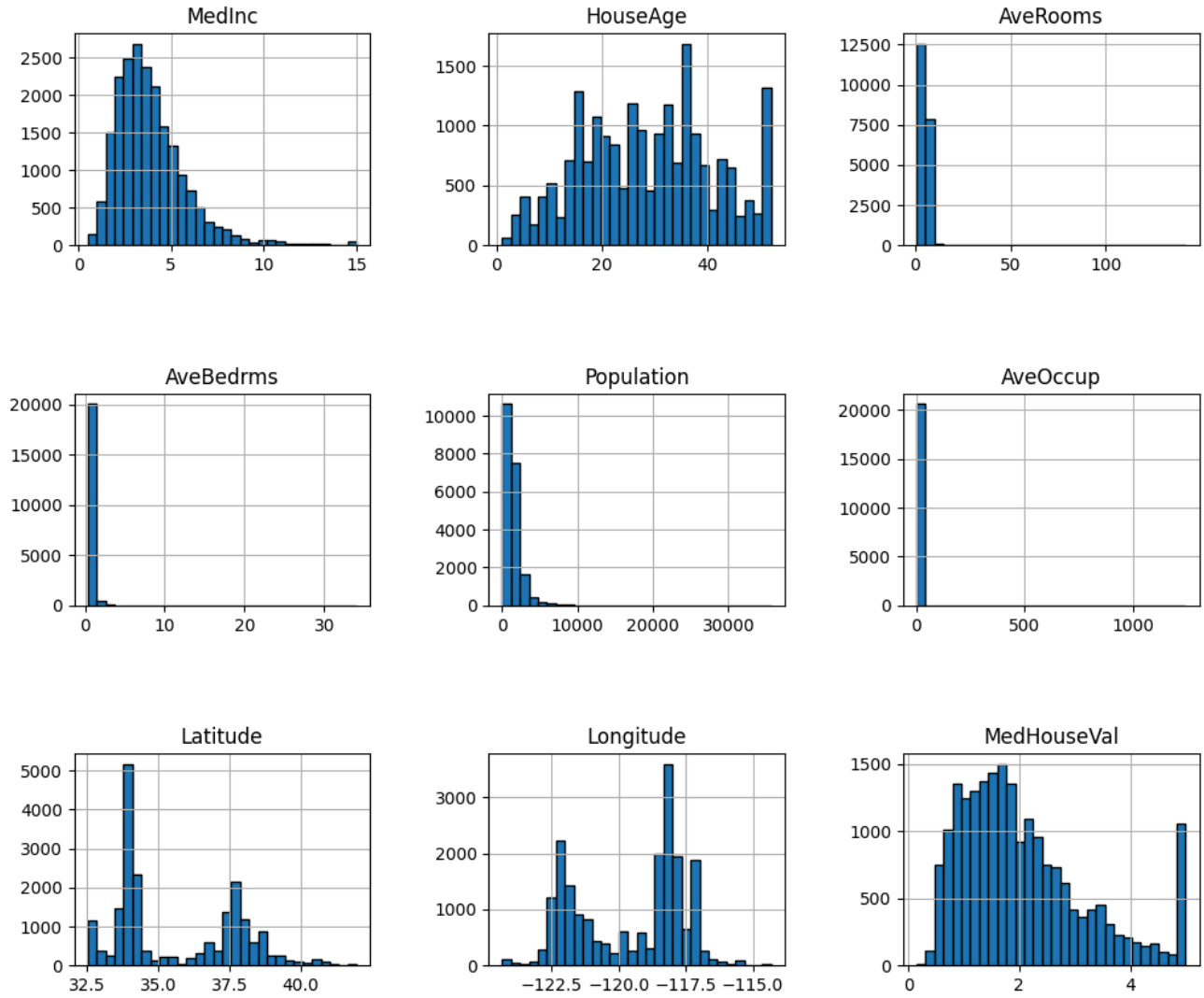
```
california_housing.frame.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
print("Numero de datos del dataset :",len(california_housing.frame))
```

```
Numero de datos del dataset : 20640
```

```
california_housing.frame.hist(figsize=(12, 10), bins=30, edgecolor="black")
plt.subplots_adjust(hspace=0.7, wspace=0.4)
```




```
# Analicemos los features mas especiales
features_of_interest = ["AveRooms", "AveBedrms", "AveOccup", "Population"]
california_housing.frame[features_of_interest].describe()
```

	AveRooms	AveBedrms	AveOccup	Population
count	20640.000000	20640.000000	20640.000000	20640.000000
mean	5.429000	1.096675	3.070655	1425.476744
std	2.474173	0.473911	10.386050	1132.462122
min	0.846154	0.333333	0.692308	3.000000
25%	4.440716	1.006079	2.429741	787.000000
50%	5.229129	1.048780	2.818116	1166.000000
75%	6.052381	1.099526	3.282261	1725.000000
max	141.909091	34.066667	1243.333333	35682.000000

Comentario PG:

Como se comentaba el data set tiene 8 features de analisis y 20640 registros.

De los histograma se puede observar que se distribuyen segun lo esperado.

Ejemplo:

MedInc, HouseAge y MedHouseVal , como son variables de edad y sueldos, tienen distribuciones similares a la distribucion normal.

Longitud y latitud corresponden a la ubicacion, por lo tanto no entran en este analisis.

Resto de Variables:

AveRooms: Promedio de 5.43. Es esperable. Tiene valores outliers fuertes.

AveBedrms: Promedio de 1.09. Es esperable. Tiene valores outliers fuertes.

Population: Promedio de 1425. No es comparable.

AveOccup: Promedio de 3.07. Es esperable. Tiene valores outliers fuertes.

3.- Preparacion de Datos

```
# Preparamos los datos para comenzar el analisis con Redes Neuronales

# Separar características y etiquetas
X = california_housing.data
y = california_housing.target

# Dividir el conjunto de 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 características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

4.- Implementación y entrenamiento de la red neuronal

Pasamos aquí a diseñar y entrenar nuestra red neuronal. Alguna de las cosas que podemos probar es:

- Número de capas y neuronas, así como capas de drop out o algún otro tipo de regularización al crear la red neuronal con el modelo *Sequential*. ¡OJO! Tenemos que pensar, especialmente, en qué función de activación y tamaño queremos en la última capa.
- Optimizador al compilar el modelo. Además aquí la función de perdida será el MSE, ya que es la métrica a optimizar, como está explicado en el documento de texto de la tarea.
- Número de epochs o tamaño del batch, que le podemos indicar a la red en el entrenamiento - método *fit*.

Estas directrices valen para el otro dataset que usaremos en la siguiente sección.

Nota: se recomienda empezar con una red sencilla, de una capa y unas pocas neuronas, e ir añadiendo complejidad poco a poco. No empezemos con muchas capas y técnicas de regularización de primeras, sobre todo si no es necesario.

Arquitectura de la Red Neuronal (Modelo Base):

- Capa entrada: 8 n
- Capa oculta: 32 n
- Capa Salida: 1 n, sin funcion activación.
- Funcion de Activación : Relu
- epochs: 50
- batch_size: 64
- optimizer: 'adam'
- loss: 'mean_squared_error'

```
# Arquitectura de la Red Neuronal con Keras

# Definimos el modelo
model = models.Sequential()

# Capa de entrada 8 (features) y capa oculta con 32 neuronas
model.add(Dense(32, input_shape=(8,), activation='relu'))

# Capa de salida sin funcion de activacion
model.add(Dense(1, activation='linear'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Compilamos el modelo
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Mostramos el resumen del modelo
model.summary()
```

```
Model: "sequential"
```

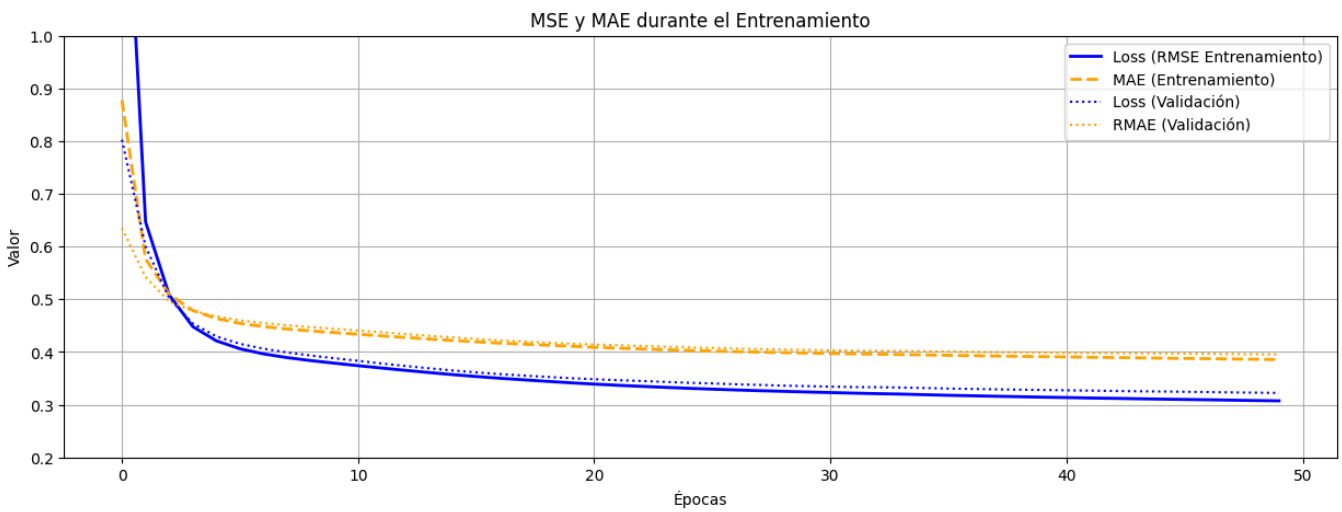
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	288
dense_1 (Dense)	(None, 1)	33

```
Total params: 321 (1.25 KB)
Trainable params: 321 (1.25 KB)
Non-trainable params: 0 (0.00 B)
```

```
# Entrenamos el modelo definiendo el numero de Epochs y el batch_size. Tambien obtenemos los resultados de los datos de validacion
history = model.fit(X_train_scaled, y_train, epochs=50, batch_size=64,validation_data=(X_test_scaled, y_test), verbose=0)
```

5.- Evaluacion de la Red Neuronal

```
# Graficamos los valores de las funciones de error durante el proceso de entrenamiento de la red
fig = plt.figure(figsize=(15,5))

# Graficamos las métricas de entrenamiento
plt.plot(history.history['loss'], label='Loss (RMSE Entrenamiento)', color='blue', linestyle='-', linewidth=2)
plt.plot(history.history['mae'], label='MAE (Entrenamiento)', color='orange', linestyle='--', linewidth=2)

# Graficamos las métricas de validacion
plt.plot(history.history['val_loss'], label='Loss (Validación)', color='blue', linestyle=':')
plt.plot(history.history['val_mae'], label='RMAE (Validación)', color='orange', linestyle=':')


# Configurar el título y etiquetas de los ejes
plt.title('MSE y MAE durante el Entrenamiento')
plt.ylabel('Valor')
plt.xlabel('Épocas')

# Agregar la leyenda
plt.legend(loc='upper right')

plt.ylim(0.2, 1)
plt.grid(True)

# Mostrar el gráfico
plt.show()
```



Comentario PG:

Se aprecia una convergencia rapida en los primeros epochs, luego el se hace mas gradual. Cercano a 50 epochs se tiene a estabilizar.
Los valores de entrenamiento y validacion se parecen mucho, siendo levemente mejores los valores de entrenamiento.

```
# Evaluamos el modelo con el conjunto de prueba
mse = model.evaluate(X_test_scaled, y_test)

# Generamos las Predicciones del Modelo
y_pred = model.predict(X_test_scaled)
```

```
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3181 - mae: 0.3936
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

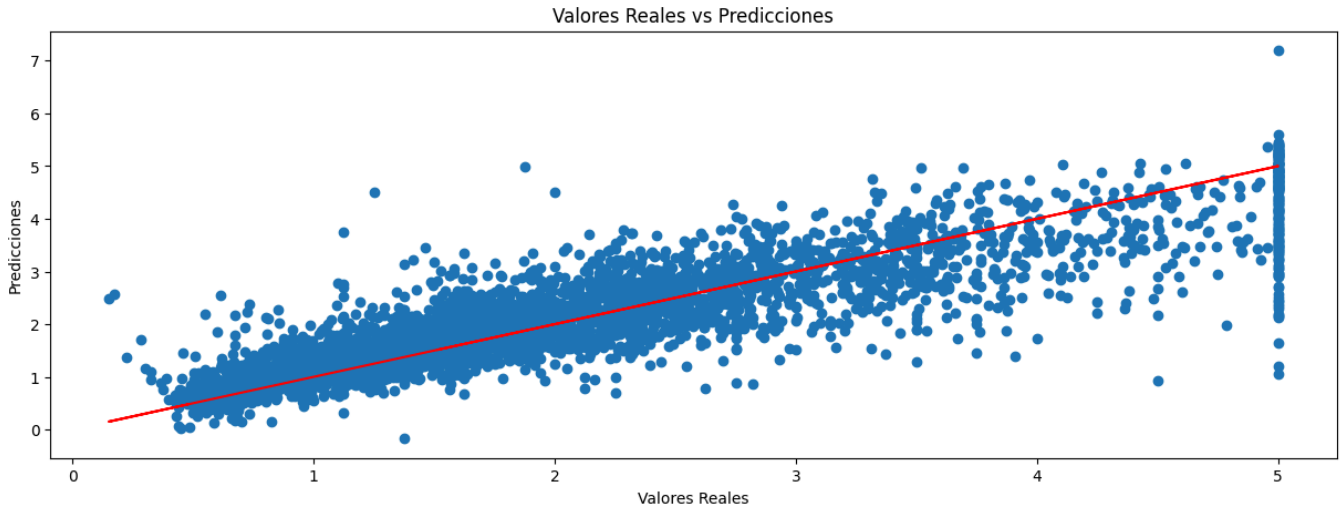
```
# Imprimimos las metricas del modelo

# Obtenemos las métricas del historial de entrenamiento
mse_train = round(history.history['loss'][-1], 4)
mae_train = round(history.history['mae'][-1], 4)

# Obtenemos las métricas de validación (test) del historial de entrenamiento
mse_val = round(history.history['val_loss'][-1], 4)
mae_val = round(history.history['val_mae'][-1], 4)

# Calculamos R2 para el conjunto de prueba
r2 = round(r2_score(y_test, y_pred), 4)

# Creamos un diccionario con los resultados
resultados = {
    'Métrica': ['MSE', 'MAE', 'R2'],
    'Entrenamiento': [mse_train, mae_train, r2],
    'Validacion': [mse_val, mae_val, None]
}

# Creamos el DataFrame
df_resultados = pd.DataFrame(resultados)

# Calculamos la diferencia porcentual
df_resultados['Diferencia (%)'] = (
    (df_resultados['Entrenamiento'] - df_resultados['Validacion']) / df_resultados['Entrenamiento'] * 100
).round(2)

# Mostramos el DataFrame
df_resultados
```

	Métrica	Entrenamiento	Validacion	Diferencia (%)
0	MSE	0.3074	0.3224	-4.88
1	MAE	0.3858	0.3956	-2.54
2	R2	0.7540	NaN	NaN

Comentario PG:

- Tanto el valor de la funcion de perdia , MSE como el MAE son relativamente bajos, pero queda espacion para mejorarlos. El RMSE es del orden de 0,5.
- El valor de R2 es satisfactorio pero puede mejorar, debe acercarse lo maximo posible a 1.
- Las curvas perdida para los datos de validacion son un poco peores que la de los datos de entenamiento. Se puede ver que una diferencia de 4.88% para el MSE y 2.54% para el MAE.
- Probablemente los resultados mejorarian bastantes si eliminaramos los ouliers de los datos.

```
# Realizamos un grafico para comparar los valores reales versus la prediccion del modelo.
fig=plt.figure(figsize=(15,5))

plt.scatter(y_test, y_pred)
plt.plot(y_test,y_test,'r')


# plt.scatter(y_test, y_pred)
#plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')  # Línea ideal
plt.xlabel('Valores Reales')
plt.ylabel('Predicciones')
plt.title('Valores Reales vs Predicciones')
plt.show()
```



Comentario PG:
Como se comentaba anteriormente, la curva de prediccion del precio, pasa bastante central a la media de los valores, lo que se ve reflejado en R2 = 0,754.

6.- Implementacion y Analisis de Diferentes Configuraciones de Hiperparametros

A continuacion generamos un codigo que nos permitirá realizar comparaciones del problema estudiado `cambiando hiperparametros como numero de capas, neuronas, tipo de optimizacion, tipo de funcion de perdida`, etc. Dejaremos los resultados del Modelo Base como la solucion estandar para comparar. Es importante mencionar que aca no es necesario tener un numero de Epoch y Batch_size grande, porque lo que queremos comparar son otros parametros. Mantendremos un numero de Epoch y Batch_size bajo para no hacer la prueba mas costosa.

```
# Recordemos que tenemos cargados los datos del dataset

#X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.2, random_state=42)
#X_train_scaled = scaler.fit_transform(X_train)
#X_test_scaled = scaler.transform(X_test)

# Definimos una función para crear modelos con diferentes configuraciones de hiperparametros
def create_model(layers, activation, optimizer):
    model = Sequential()
    model.add(Dense(layers[0], input_shape=(X_train_scaled.shape[1],), activation=activation))
    for layer in layers[1:]:
        model.add(Dense(layer, activation=activation))
    model.add(Dense(1))  # Última capa para regresión
    model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])
    return model
```

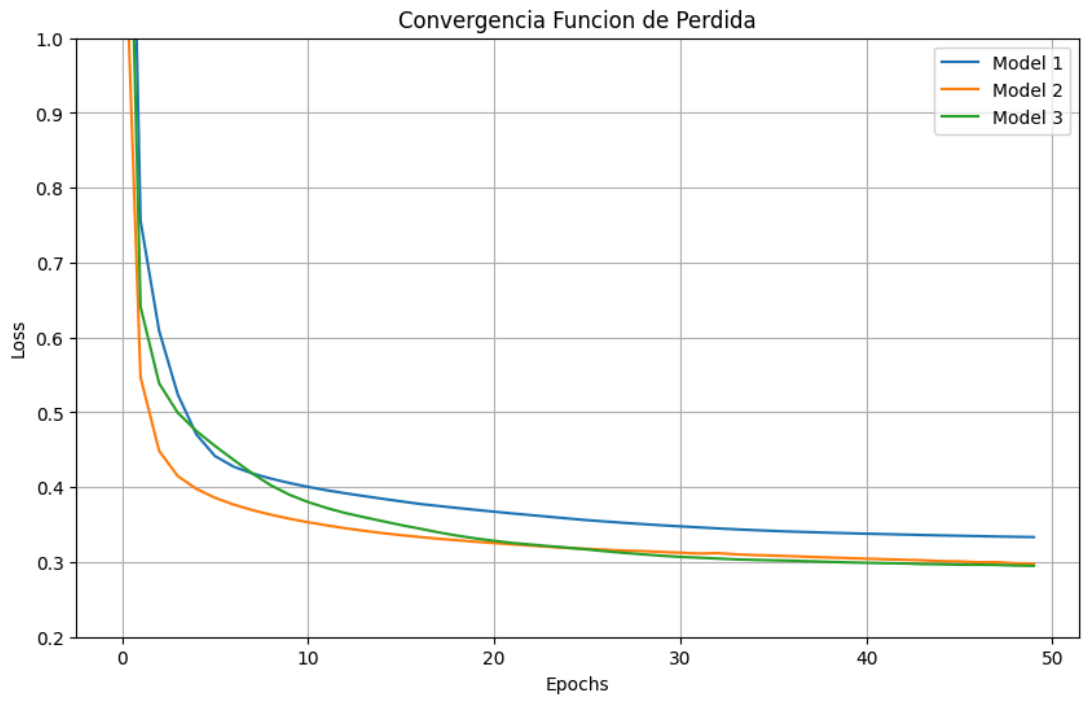
Prueba 1: cambiando capas y numero de neuronas

```
# Definimos las configuraciones de los modelos (capas, activación, optimizadores)
models_config = [
    {"layers": [16], "activation": "relu", "optimizer": 'adam'},
    {"layers": [16, 16], "activation": "relu", "optimizer": 'adam'},
    {"layers": [8, 16, 8], "activation": "relu", "optimizer": 'adam'}
]

histories = []
for config in models_config:
    model = create_model(config["layers"], config["activation"], config["optimizer"])
    history = model.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), epochs=50, batch_size=64, verbose=0)
    histories.append(history)

# Evaluamos los modelos usando Loss, MAE y R^2 (pseudo-accuracy)
results = []
for i, config in enumerate(models_config):
    loss, mae = histories[i].model.evaluate(X_test_scaled, y_test, verbose=0)
    predictions = histories[i].model.predict(X_test_scaled)
    r2 = r2_score(y_test, predictions)
    rounded_config = {key: round(value, 4) if isinstance(value, (int, float)) else value for key, value in config.items()}
    results.append({"config": rounded_config, "loss": round(loss,4), "mae": round(mae,4), "r2": round(r2,4)})

# Comparamos resultados
```

```
# Comparamos resultados
for result in results:
    print(f"Config: {result['config']}, Loss: {result['loss']}, MAE: {result['mae']}, R2 (Accuracy): {result['r2']}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Config: {'layers': [16], 'activation': 'relu', 'optimizer': 'adam'}, Loss: 0.3513, MAE: 0.4129, R2 (Accuracy): 0.7319
Config: {'layers': [16, 16], 'activation': 'relu', 'optimizer': 'adam'}, Loss: 0.3091, MAE: 0.387, R2 (Accuracy): 0.7641
Config: {'layers': [8, 16, 8], 'activation': 'relu', 'optimizer': 'adam'}, Loss: 0.3111, MAE: 0.3858, R2 (Accuracy): 0.7626
```

```
# Graficamos la convergencia de las pérdidas (loss) para cada modelo
plt.figure(figsize=(10,6))
for i, history in enumerate(histories):
    plt.plot(history.history['loss'], label=f'Model {i+1}')
plt.title('Convergencia Funcion de Perdida')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.ylim(0.2, 1)
plt.grid(True)
plt.show()
```



Comentario PG:

- EL mejor accuracy lo tuvo elmodelo 2(2 2 capas), no obstante el modelo 3 (3 capas) obtuvo valores similares.
- Config: {'layers': [16, 16], 'activation': 'relu', 'optimizer': 'adam'}, Loss: 0.3091, MAE: 0.3871, R2 (Accuracy): 0.7642

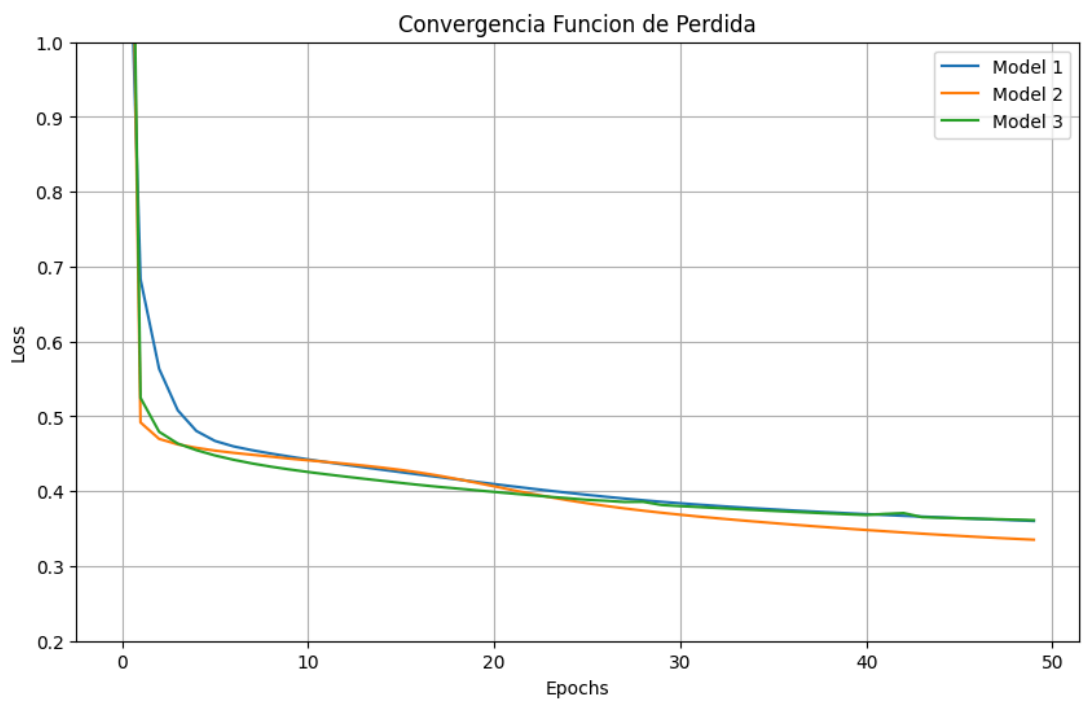
Prueba 2: Probamos diferentes funciones de activacion

```
# Definimos las configuraciones de los modelos (capas, activación, optimizadores)
models_config = [
    {"layers": [32], "activation": "sigmoid", "optimizer": 'adam'},
    {"layers": [32], "activation": "tanh", "optimizer": 'adam'},
    {"layers": [32], "activation": "elu", "optimizer": 'adam'}
]

histories = []
for config in models_config:
    model = create_model(config["layers"], config["activation"], config["optimizer"])
    history = model.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), epochs=50, batch_size=64, verbose=0)
    histories.append(history)

# Evaluamos los modelos usando Loss, MAE y R^2 (pseudo-accuracy)
results = []
for i, config in enumerate(models_config):
```

```
    loss, mae = histories[i].model.evaluate(X_test_scaled, y_test, verbose=0)
    predictions = histories[i].model.predict(X_test_scaled)
    r2 = r2_score(y_test, predictions)
    rounded_config = {key: round(value, 4) if isinstance(value, (int, float)) else value for key, value in config.items()}
    results.append({"config": rounded_config, "loss": round(loss,4), "mae": round(mae,4), "r2": round(r2,4)})

# Comparamos resultados
for result in results:
    print(f"Config: {result['config']}, Loss: {result['loss']}, MAE: {result['mae']}, R2 (Accuracy): {result['r2']}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Config: {'layers': [32], 'activation': 'sigmoid', 'optimizer': 'adam'}, Loss: 0.3756, MAE: 0.435, R2 (Accuracy): 0.7134
Config: {'layers': [32], 'activation': 'tanh', 'optimizer': 'adam'}, Loss: 0.3468, MAE: 0.4141, R2 (Accuracy): 0.7353
Config: {'layers': [32], 'activation': 'elu', 'optimizer': 'adam'}, Loss: 0.3765, MAE: 0.4325, R2 (Accuracy): 0.7127
```

```
# Graficamos la convergencia de las pérdidas (loss) para cada modelo
plt.figure(figsize=(10,6))
for i, history in enumerate(histories):
    plt.plot(history.history['loss'], label=f'Model {i+1}')
plt.title('Convergencia Funcion de Perdida')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.ylim(0.2, 0.7)
plt.grid(True)
plt.show()
```



Comentario PG:

- El modelo 2 tuvo el mejor accuracy.
- Config: {'layers': [32], 'activation': 'tanh', 'optimizer': 'adam'}, Loss: 0.3468, MAE: 0.4141, R2 (Accuracy): 0.7353
- Se uso funcion de activacion `Tanh` .

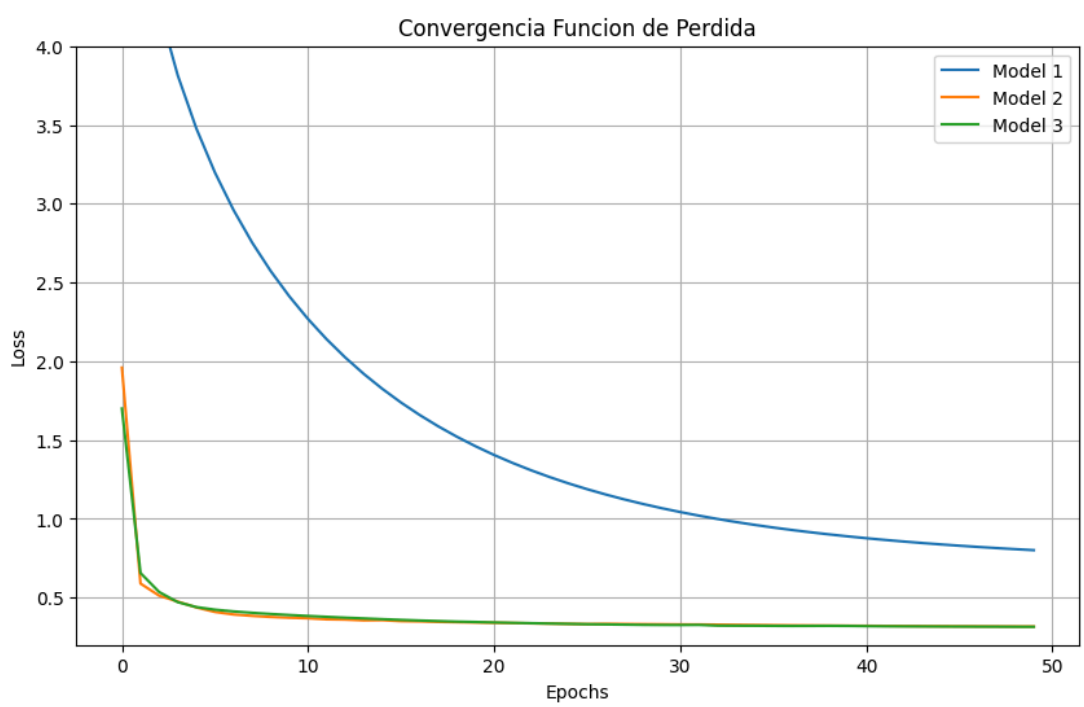
Prueba 3: Probamos diferentes funciones de optimizacion

```
from keras.optimizers import Adagrad, RMSprop, Nadam
# Definimos las configuraciones de los modelos (capas, activación, optimizadores)
models_config = [
    {"layers": [32], "activation": "relu", "optimizer": Adagrad(learning_rate=0.001)},
    {"layers": [32], "activation": "relu", "optimizer": RMSprop(learning_rate=0.001)},
    {"layers": [32], "activation": "relu", "optimizer": Nadam(learning_rate=0.001)}
]

histories = []
```

```
for config in models_config:
    model = create_model(config["layers"], config["activation"], config["optimizer"])
    history = model.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), epochs=50, batch_size=64, verbose=0)
    histories.append(history)

# Evaluamos los modelos usando Loss, MAE y R^2 (pseudo-accuracy)
results = []
for i, config in enumerate(models_config):
    loss, mae = histories[i].model.evaluate(X_test_scaled, y_test, verbose=0)
    predictions = histories[i].model.predict(X_test_scaled)
    r2 = r2_score(y_test, predictions)
    rounded_config = {key: round(value, 4) if isinstance(value, (int, float)) else value for key, value in config.items()}
    results.append({"config": rounded_config, "loss": round(loss,4), "mae": round(mae,4), "r2": round(r2,4)})

# Comparamos resultados
for result in results:
    print(f"Config: {result['config']}, Loss: {result['loss']}, MAE: {result['mae']}, R2 (Accuracy): {result['r2']}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Config: {'layers': [32], 'activation': 'relu', 'optimizer': <keras.src.optimizers.adagrad.Adagrad object at 0x7bc66cac4fd0>}, Loss: 0.79
Config: {'layers': [32], 'activation': 'relu', 'optimizer': <keras.src.optimizers.rmsprop.RMSprop object at 0x7bc66cac6bc0>}, Loss: 0.32
Config: {'layers': [32], 'activation': 'relu', 'optimizer': <keras.src.optimizers.nadam.Nadam object at 0x7bc66cac7490>}, Loss: 0.3299,
```

```
# Graficamos la convergencia de las pérdidas (loss) para cada modelo
plt.figure(figsize=(10,6))
for i, history in enumerate(histories):
    plt.plot(history.history['loss'], label=f'Model {i+1}')
plt.title('Convergencia Funcion de Perdida')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.ylim(0.2, 4)
plt.grid(True)
plt.show()
```



Comentario PG:

- El modelo que tuvo mejor accuracy fue el 2.
- Config: {'layers': [32], 'activation': 'relu', 'optimizer': <keras.src.optimizers.rmsprop.RMSprop object at 0x7bc66cac6bc0>}, Loss: 0.327, MAE: 0.3992, R2 (Accuracy): 0.7505
- Se uso funcion de optimizacion `RMSprop`.

Es interesante destacar que el modelo 1, con Adagrad(learning_rate=0.001), produce una convergencia suave, aunque no los mejores resultados.

El modelo 3,con Nadam(learning_rate=0.001) , tambien tiene buen desempeño.

7.- Entrenamiento con Mejores **Hiperparametros**

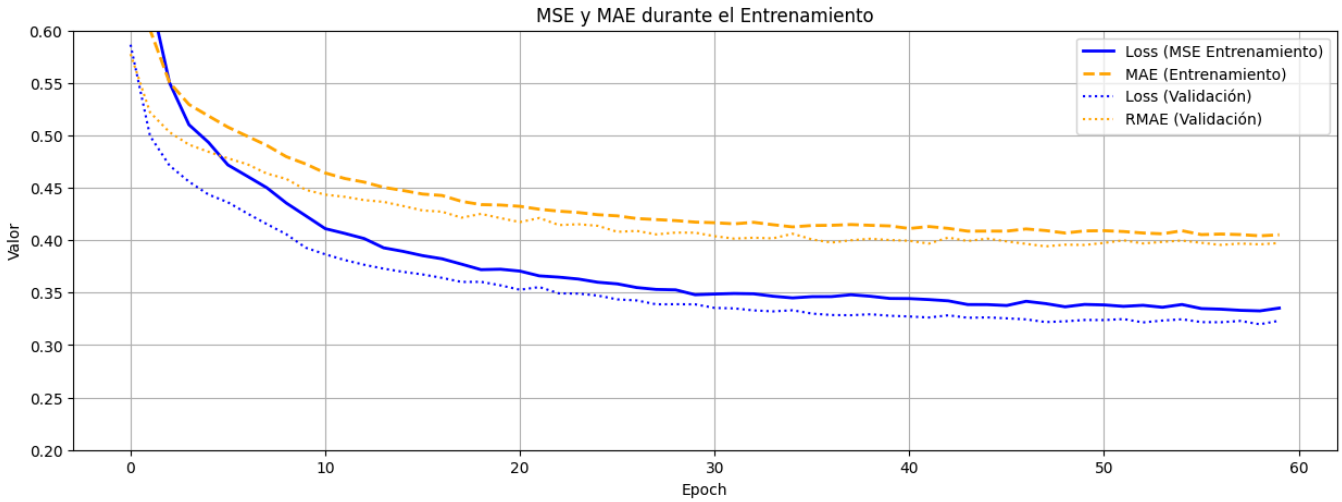
Lo que sigue es probar una arquitectura con los mejores resultados de los hiperparametros obtenidos y aplicando la tecnica de Drop out par evitar sorbreajuste:

- Capa entrada: 8 n
- Capa oculta1: 32 n, Funcion de Activación : tanh (se usaran 2 capaz pero se aumentará el numero de neuronas)
- Capa oculta2: 32 n, Funcion de Activación : tanh
- Capa Salida: 1 n, sin funcion activación.
- epochs: 60 (modelo se estabiliza en 50)
- batch_size: 64
- optimizer: RMSprop(learning_rate=0.001)
- loss: 'mean_squared_error'

```
# Arquitectura de la Red Neuronal con Keras

# Definimos el modelo secuencial
model = Sequential()

# Primera capa con 16 neuronas y función de activación 'tanh'
model.add(Dense(16, input_shape=(8,), activation='tanh'))

# Dropout para desactivar el 20% de las neuronas
model.add(Dropout(0.2))

# Segunda capa con 16 neuronas y función de activación 'tanh'
model.add(Dense(16, activation='tanh'))

# Capa de salida con 1 neurona y activación 'lineal' ( sin activacion)
model.add(Dense(1, activation='linear'))

# Compilamos el modelo
model.compile(optimizer=RMSprop(learning_rate=0.001), loss='mean_squared_error', metrics=['mae'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# definimos un early stoping para evitar entrenar de mas
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Entrenamos el modelo definiendo el numero de Epochs y el batch_size
history = model.fit(X_train_scaled, y_train, epochs=60, batch_size=64,validation_data=(X_test_scaled, y_test), verbose=0)


# Graficamos los valores de las funciones de error durante el proceso de entrenamiento de la red
fig = plt.figure(figsize=(15,5))

# Graficar las métricas de entrenamiento
plt.plot(history.history['loss'], label='Loss (MSE Entrenamiento)', color='blue', linestyle='-', linewidth=2)
plt.plot(history.history['mae'], label='MAE (Entrenamiento)', color='orange', linestyle='--', linewidth=2)


# Graficamos las métricas de validacion
plt.plot(history.history['val_loss'], label='Loss (Validación)', color='blue', linestyle=':')
plt.plot(history.history['val_mae'], label='RMAE (Validación)', color='orange', linestyle=':')

# Configurar el título y etiquetas de los ejes
plt.title('MSE y MAE durante el Entrenamiento')
plt.ylabel('Valor')
plt.xlabel('Epoch')

# Agregar la leyenda
plt.legend(loc='upper right')

plt.ylim(0.2, 0.6)
plt.grid(True)

# Mostrar el gráfico
plt.show()
```




```
# Evaluamos el modelo con el conjunto de prueba
mse = model.evaluate(X_test_scaled, y_test)

# Generamos las Predicciones del Modelo
y_pred = model.predict(X_test_scaled)
```

Mostrar el resultado oculto

```
# Obtenemos las métricas del historial de entrenamiento
mse_train = round(history.history['loss'][-1], 4)
mae_train = round(history.history['mae'][-1], 4)

# Obtenemos las métricas de validación (test) del historial de entrenamiento
mse_val = round(history.history['val_loss'][-1], 4)
mae_val = round(history.history['val_mae'][-1], 4)

# Calculamos R2 para el conjunto de prueba
r2 = round(r2_score(y_test, y_pred), 4)

# Creamos un diccionario con los resultados
resultados = {
    'Métrica': ['MSE', 'MAE', 'R2'],
    'Entrenamiento': [mse_train, mae_train, r2],
    'Validacion': [mse_val, mae_val, None]
}

# Creamos el DataFrame
df_resultados = pd.DataFrame(resultados)

# Calculamos la diferencia porcentual
df_resultados['Diferencia (%)'] = (
    (df_resultados['Entrenamiento'] - df_resultados['Validacion']) / df_resultados['Entrenamiento'] * 100
).round(2)

# Mostramos el DataFrame
df_resultados
```

	Métrica	Entrenamiento	Validacion	Diferencia (%)
0	MSE	0.3353	0.3233	3.58
1	MAE	0.4052	0.3974	1.92
2	R2	0.7533	NaN	NaN

Analisis Final

- Hemos comenzado por una arquitectura simple, para tener un modelo base con el que poder comparar.
- Posteriormente hemos realizado varios entrenamiento comparando en forma aislada diferentes hiperparametros, como # capas, # neuronas, funciones de activacion, optimizacion, etc.
- Se pudo observar que el tipo de funciones de activacion presenta fuertes influencias, tanto en la presicion del resultado, como en el desempeño del entrenamiento, esto es rapidez y forma de converger.
- Finalmente nos hemos quedado con las mejores configuraciones que encontramos y generamos un modelo. A este modelo le hemos agregado el metodo dropout para ver su efecto(debiera evitar el sobre ajuste).
  Los resultados empeoraron en comparacion al modelo Base:

Modelo Base

- Métrica Entrenamiento Validacion Diferencia (%)
- MSE 0.3074 0.3224 -4.88
- MAE 0.3858 0.3956 -2.54
- R2 0.7540 NaN NaN

Modelo Final

- Métrica Entrenamiento Validacion Diferencia (%)
- MSE 0.3353 0.3233 3.58
- MAE 0.4052 0.3974 1.92
- R2 0.7533 NaN

Al incorporar el metodo dropout algunos parametros empeoraron, como Loss y MAE, sin embargo el R2 se mantuvo similar. Lo que es rescatable de observar es que los datos de validacion fueron mejores que los de entrenamiento. Esto se debe al efecto Dropout.

Conclusion

- Logramos encontrar una configuracion que funcionó similar a al modelo Base inicial, pero incorporando dropout, lo que es una mejoria en el modelo.
- Pudimos comparar diferentes arquitecturas e hiperparametros de redes neuronales.
- Encontramos un modelo que se ajusta bastante bien a los datos.
- Lo que queda sería probar incorporar metodos como Normalizacion L2 para evitar tener Sobreajuste.

========== FIN ==========