

Actividad 3: RNN y sus aplicaciones en las series temporales

En esta actividad se va a aplicar el conocimiento adquirido sobre las RNN para entrenar modelos que sean capaces de predecir el comportamiento de las series temporales. Para ello, se usará un dataset de temperaturas para mediante la aplicación de RNN, predecir los valores futuros que tendrá la serie temporal que se tiene. Este trabajo se suele hacer mediante modelos ARIMA, pero en esta práctica se verá cómo el modelado mediante RNN es una opción muy buena en estos casos de series temporales.

1. Descargar el dataset y almacenarlo

En primer lugar hay que importar tensorflow.

```
In [ ]: import tensorflow as tf
        print(tf.__version__)
```

2.16.1

El siguiente paso es importar las bibliotecas numpy y matplotlib. Además, se define el método **plot_series** que se utilizará para hacer las gráficas de las series temporales.

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        def plot_series(time, series, format="-", start=0, end=None):
            plt.plot(time[start:end], series[start:end], format)
            plt.xlabel("Time")
            plt.ylabel("Value")
            plt.grid(True)
```

A continuación se descarga el dataset de las temperaturas mínimas diarias.

```
In [ ]: import requests
        import os

        # URL del dataset
        url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-tem

        # Obtenemos el directorio de trabajo actual para guardar el archivo ahí
        current_dir = os.getcwd()
        file_path = os.path.join(current_dir, "daily-min-temperatures.csv")

        # Hacemos la solicitud HTTP para obtener el contenido del archivo
        response = requests.get(url)

        # Guardamos el contenido en un archivo en el directorio de trabajo actual
        with open(file_path, 'wb') as file:
            file.write(response.content)

        print(f"Archivo descargado correctamente en {file_path}")
```

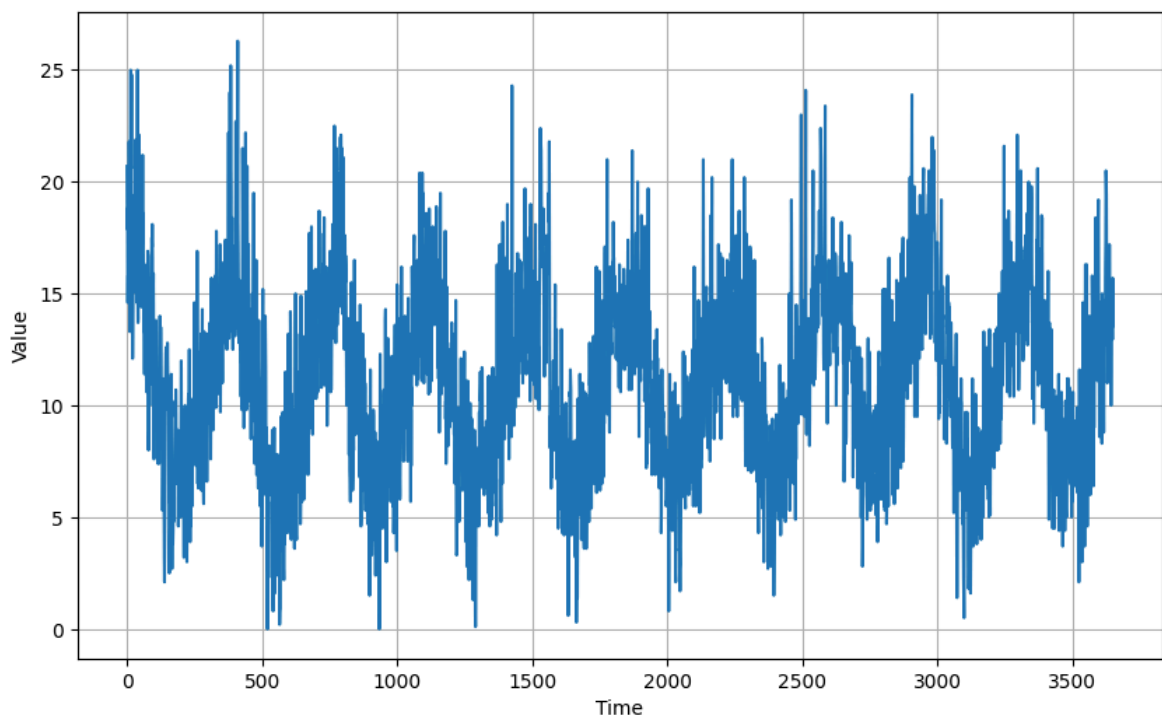
Archivo descargado correctamente en c:\Users\miguel\OneDrive\Documentos\Master_UNIR\Especialista Universitario en Big Data e IA\Sistemas Cognitivos Artificiales\Actividades\Act3\daily-min-temperatures.csv

En este paso, se utilizará la biblioteca csv de Python para guardar y poder leer el dataset de temperaturas mínimas diarias que ha sido descargado en el paso anterior. Además, se construye la variable **series** que será donde se guarde la serie temporal. Por último, siempre que se trate con una serie temporal, es una buena práctica hacer un gráfico para poder verla y tener una idea de cómo es.

```
In [ ]: import csv
time_step = []
temps = []

with open('daily-min-temperatures.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    step=0
    for row in reader:
        temps.append(float(row[1]))
        time_step.append(step)
        step = step + 1

series = np.array(temps)
time = np.array(time_step)
plt.figure(figsize=(10, 6))
plot_series(time, series)
```



2. Creación de las variables necesarias para el diseño de la red neuronal

Una técnica muy común cuando se trata con series temporales es utilizar una ventana temporal que se vaya desplazando sobre la serie temporal para reducir su análisis a lo

que ocurre en esa ventana de forma local, para a continuación realizar el modelado global.

Ejercicio 1 (0.4 puntos): Crear las variables de entrenamiento y validación y hacer la partición de las mismas. Las variables que hay que crear son:

- time_train
- x_train
- time_valid
- x_valid

En primer lugar, dividimos la serie temporal en conjuntos de entrenamiento y validación utilizando el índice 'split_time'.

Después, definimos las funciones que generan lotes de datos con una ventana deslizante que avanza sobre la serie temporal para el entrenamiento.

```
In [ ]: ## variables para la técnica de la ventana temporal
split_time = 2500 # Índice para dividir datos en entrenamiento y test
window_size = 30 # Cantidad de pasos temporales en cada ventana
batch_size = 32 # Tamaño de cada lote de datos
shuffle_buffer_size = 1000 # Tamaño del buffer para mezclar datos

## Split del dataset en entrenamiento y validación
time_train = time[:split_time] # Tiempos de entrenamiento
x_train = series[:split_time] # Datos de entrenamiento
time_valid = time[split_time:] # Tiempos de validación
x_valid = series[split_time:] # Datos de validación
```

2. Creación del método **windowed_dataset** para poder utilizarlo en el modelo. Las entradas por parámetros del método son:

- series
- window_size
- batch_size
- shuffle_buffer

El resto de elementos que se usan para construir la función ventana temporal para explorar el dataset, son métodos de Python para tratar con series temporales.

```
In [ ]: def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    """
    Prepara un dataset TensorFlow usando una ventana deslizante que agrupa los d
    formatos consecutivos para el entrenamiento de modelos de series temporales.

    Args:
    series (array-like): Datos de la serie temporal.
    window_size (int): Cantidad de pasos temporales por ventana.
    batch_size (int): Tamaño de lote para el entrenamiento.
    shuffle_buffer (int): Tamaño del buffer de mezcla para aleatorizar los eleme

    Returns:
    tf.data.Dataset: Dataset configurado para el entrenamiento del modelo.
```

```

"""
dataset = tf.data.Dataset.from_tensor_slices(series)
dataset = dataset.window(window_size+1, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(window_size+1))
dataset = dataset.shuffle(shuffle_buffer).map(lambda window: (window[:-1], w
dataset = dataset.batch(batch_size).prefetch(1)
return dataset

# Creamos los datasets de ventana temporal para entrenamiento y validación
train_dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffe
valid_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_si

# Función para visualizar una parte de los datos
def plot_series(time, series, format="-", start=0, end=None):
    """
    Grafica los datos de la serie temporal.

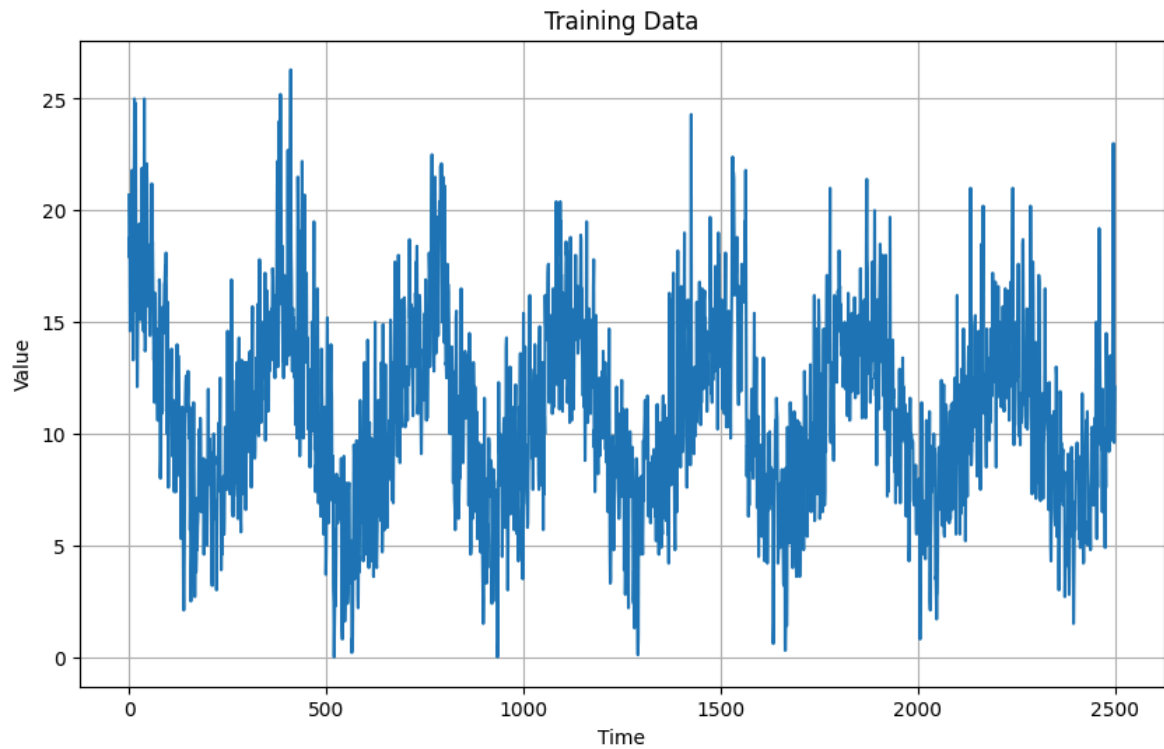
    Args:
    time (array-like): Vector de tiempos asociados a los datos de la serie.
    series (array-like): Datos de la serie temporal a graficar.
    format (str): Estilo del trazo de la línea en la gráfica.
    start (int): Índice inicial de los datos a graficar.
    end (int): Índice final de los datos a graficar.
    """
    plt.figure(figsize=(10, 6))
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.grid(True)

plt.figure(figsize=(10, 6))
plot_series(time_train, x_train)
plt.title('Training Data')
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plt.title('Validation Data')

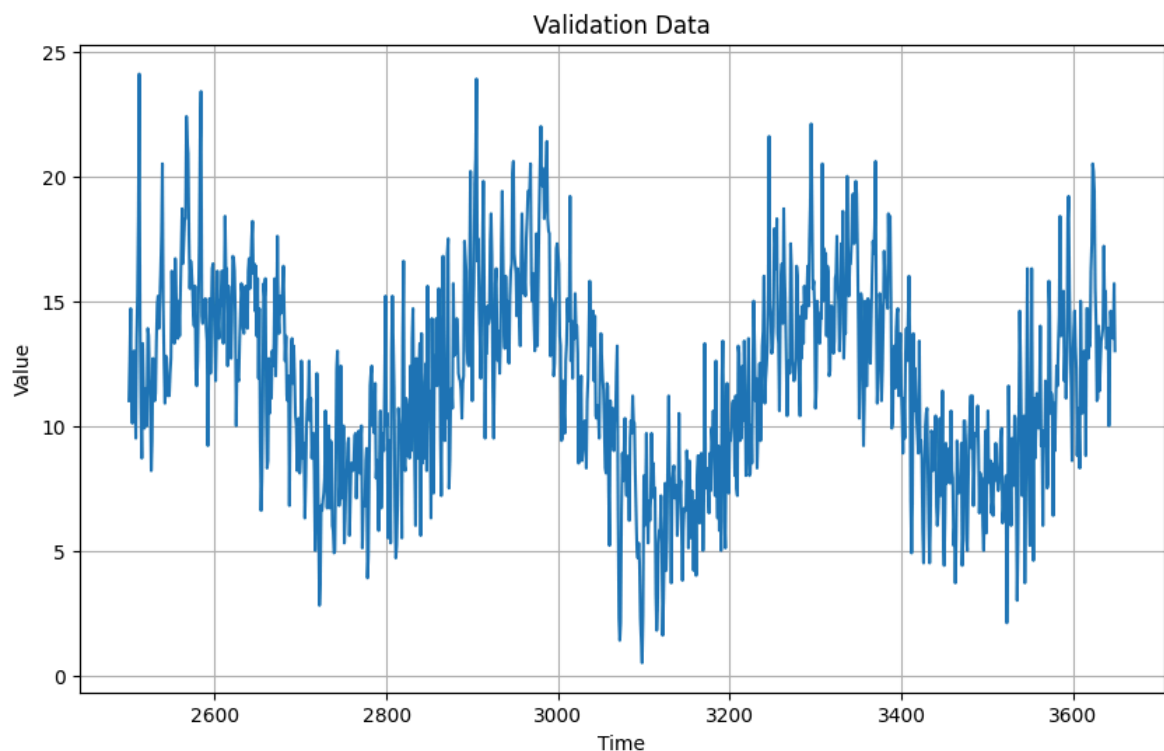
plt.show()

```

<Figure size 1000x600 with 0 Axes>



<Figure size 1000x600 with 0 Axes>



3. Diseño de la función para predecir los siguientes valores de la serie temporal usando la técnica de la ventana temporal

```
In [ ]: def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    series = tf.expand_dims(series, axis=-1)
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
```

```
ds = ds.shuffle(shuffle_buffer)
ds = ds.map(lambda w: (w[:-1], w[1:]))
return ds.batch(batch_size).prefetch(1)
```

A continuación, y usando como modelo el método **windowed_dataset** se procederá a adaptar el método **model_forecast** que se usará para predecir los siguientes valores de la serie temporal utilizando la técnica de la ventana temporal.

Ejercicio 2 (1.6 puntos): completar el método `model_forecast` creando los elementos necesarios dentro del método:

1. Crear la variable **ds** y darle el valor resultante del método **from_tensor_slices** pasando por parametro **series (0.4 puntos)**
2. Actualizar la ventana (**window**) de la variable **ds** (nota: en este caso el tamaño es el mismo de la ventana, no es necesario que sea `window_size+1`) **(0.4 puntos)**
3. Crear el **flat_map** de la variable, teniendo en cuenta que el tamaño es **window_size (0.4 puntos)**
4. Añadir la siguiente línea de código: `ds = ds.batch(32).prefetch(1)`
5. Crear la variable **forecast** en la que se usará el método **predict (0.4 puntos)**
6. Por último, se devolverá la variable `forecast`.

```
In [ ]: def model_forecast(model, series, window_size):
        """
        Esta función prepara un conjunto de datos en una forma que un modelo de pred
        para realizar predicciones basadas en la técnica de ventana temporal.

        Args:
        model (tf.keras.Model): Modelo entrenado para hacer predicciones.
        series (array-like): Datos de la serie temporal.
        window_size (int): Tamaño de la ventana de datos utilizada para generar cada

        Returns:
        array-like: Predicciones del modelo para la serie temporal proporcionada.
        """

        # 1. Creamos la variable 'ds' y la asignamos el valor de salida del método '
        ds = tf.data.Dataset.from_tensor_slices(series)

        # 2. Actualizamos la ventana de la variable 'ds'
        ds = ds.window(window_size, shift=1, drop_remainder=True)

        # 3. Creamos el flat_map de la variable, teniendo en cuenta que el tamaño es
        ds = ds.flat_map(lambda window: window.batch(window_size))

        # 4. Añadimos la línea de código para preparar el dataset por el batch y la
        ds = ds.batch(32).prefetch(1)

        # 5. Crear la variable forecast en la que se usará el método predict
        forecast = model.predict(ds)

        # 6. Devolvemos la variable
        return forecast
```

A continuación, se limpia la sesión de keras, y se inicializan las variables necesarias para poder diseñar el modelo de series temporales a entrenar usando RNN.

Añadimos también la definición del valor de 'shuffle_buffer_size' en la configuración de los parámetros para aumentar la efectividad del entrenamiento y la generalización del modelo.

```
In [ ]: tf.keras.backend.clear_session()
        tf.random.set_seed(51)
        np.random.seed(51)
        window_size = 64
        batch_size = 256
        shuffle_buffer_size = 1000
```

4. Diseño de la red neuronal

Ejercicio 3.1 (0.5 puntos): Hay que crear la variable **train_set** dándole el valor que se reciba del método **windowed_dataset**, los parametros que debe recibir este método son: **x_train, window_size, batch_size, shuffle_buffer_size**

```
In [ ]: train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_si
```

Ejercicio 3.2 (4 puntos): Se debe construir la red neuronal de aprendizaje profunda basada para modelar la serie temporal de las temperaturas minimas diarias. Esta red neuronal debera contar con las siguientes capas ocultas:

1. Una capa de convolución en una dimensión que tenga 32 filtros, una tamaño del kernel de 5, un stride de 1, padding "causal", la función de activación debe ser relu y el input shape debe ser [None, 1]
2. Una capa LSTM con 64 neuronas y retorno de secuencias
3. Una capa LSTM con 64 neuronas y retorno de secuencias
4. Una capa densa con 30 neuronas
5. Una capa densa con 10 neuronas
6. Una capa densa con 1 neuronas
7. Por último, se añade la siguiente capa: `tf.keras.layers.Lambda(lambda x: x * 400)`

```
In [ ]: """Esta red combina convolución en una dimensión y capas LSTM, seguidas de varia
model = tf.keras.models.Sequential([
    # Capa de convolución en 1D
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu",
                           input_shape=[None, 1]),

    # Primera capa LSTM
    tf.keras.layers.LSTM(64, return_sequences=True),
    # Segunda capa LSTM
    tf.keras.layers.LSTM(64, return_sequences=True),
    # Capa densa con 30 neuronas
    tf.keras.layers.Dense(30, activation="relu"),
    # Capa densa con 10 neuronas
    tf.keras.layers.Dense(10, activation="relu"),
    # Capa densa con 1 neurona
    tf.keras.layers.Dense(1),
    # Capa Lambda para escalar la salida
```

```

tf.keras.layers.Lambda(lambda x: x * 400)
])

# Compilación del modelo
model.compile(loss="mse", optimizer=tf.keras.optimizers.Adam(), metrics=["mae"])

# Resumen del modelo
model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, None, 32)	192
lstm (LSTM)	(None, None, 64)	24,832
lstm_1 (LSTM)	(None, None, 64)	33,024
dense (Dense)	(None, None, 30)	1,950
dense_1 (Dense)	(None, None, 10)	310
dense_2 (Dense)	(None, None, 1)	11
lambda (Lambda)	(None, None, 1)	0



Total params: 60,319 (235.62 KB)

Trainable params: 60,319 (235.62 KB)

Non-trainable params: 0 (0.00 B)

Explicación de la configuración del modelo:

Capa Conv1D

Esta capa usa 32 filtros con un tamaño de kernel de 5.

Capas LSTM

Se utilizan dos capas LSTM con 64 unidades cada una. 'return_sequences=True' en ambas capas permite que la salida de cada tiempo sea pasada a la siguiente capa, algo importante cuando se apilan capas LSTM.

Capas Densas

Las capas densas son parte de la red "feed-forward" que sigue a las LSTM. Aquí, usamos dos con 30 y 10 neuronas respectivamente, ambas con activación ReLU, seguido de una capa densa con una sola neurona sin función de activación para la predicción final.

Capa Lambda

Esta capa final multiplica la salida por 400.

5. Entrenamiento de la red neuronal

Ejercicio 4 (0.5 puntos): Se va a diseñar un método callbacks para el learning rate que será guardado en la variable **lr_schedule**, este método deberá usar el método **LearningRateScheduler** de Python y será una función **lambda** que le de el valor a epoch de $1e-8 * 10(\text{epoch} / 20)$ **texto en negrita**

```
In [ ]: lr_schedule = tf.keras.callbacks.LearningRateScheduler(  
        lambda epoch: 1e-8 * 10**(epoch / 20)  
    )
```

Ejercicio 5 (1.5 puntos): Compilar la red neuronal con los siguientes parametros:

- loss: método Huber de keras
- El optimizador debe ser el SGD con learning rate $1e-8$ y momentum 0.9
- La métrica a visualizar es el error absoluto medio (medium absolute error en ingles)

```
In [ ]: # Configuramos el modelo  
model = tf.keras.models.Sequential([  
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal",  
                           activation="relu", input_shape=[None, 1]),  
    tf.keras.layers.LSTM(64, return_sequences=True),  
    tf.keras.layers.LSTM(64, return_sequences=True),  
    tf.keras.layers.Dense(30, activation="relu"), # Corregido aquí  
    tf.keras.layers.Dense(10, activation="relu"),  
    tf.keras.layers.Dense(1),  
    tf.keras.layers.Lambda(lambda x: x * 400)  
)  
  
# Compilamos  
model.compile(  
    loss=tf.keras.losses.Huber(), # Función de pérdida de Huber  
    optimizer=tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9), # Opti  
    metrics=["mae"] # Métrica de error absoluto medio  
)  
  
# Resumen del modelo para confirmar la configuración  
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, None, 32)	192
lstm_2 (LSTM)	(None, None, 64)	24,832
lstm_3 (LSTM)	(None, None, 64)	33,024
dense_3 (Dense)	(None, None, 30)	1,950
dense_4 (Dense)	(None, None, 10)	310
dense_5 (Dense)	(None, None, 1)	11
lambda_1 (Lambda)	(None, None, 1)	0























Total params: 60,319 (235.62 KB)





















Trainable params: 60,319 (235.62 KB)


Non-trainable params: 0 (0.00 B)


Para terminar se entrena el modelo previamente diseñado y compilado en los pasos anteriores.


```
In [ ]: history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```


Epoch 1/100
10/10  4s 89ms/step - loss: 39.9294 - mae: 40.4285 - learning_rate: 1.0000e-08
Epoch 2/100
10/10  1s 88ms/step - loss: 37.7611 - mae: 38.2601 - learning_rate: 1.1220e-08
Epoch 3/100
10/10  1s 82ms/step - loss: 34.2635 - mae: 34.7625 - learning_rate: 1.2589e-08
Epoch 4/100
10/10  1s 90ms/step - loss: 29.9043 - mae: 30.4033 - learning_rate: 1.4125e-08
Epoch 5/100
10/10  1s 87ms/step - loss: 24.8697 - mae: 25.3687 - learning_rate: 1.5849e-08
Epoch 6/100
10/10  1s 83ms/step - loss: 19.5011 - mae: 19.9997 - learning_rate: 1.7783e-08
Epoch 7/100
10/10  1s 83ms/step - loss: 14.3069 - mae: 14.8008 - learning_rate: 1.9953e-08
Epoch 8/100
10/10  1s 81ms/step - loss: 11.1794 - mae: 11.6710 - learning_rate: 2.2387e-08
Epoch 9/100
10/10  1s 83ms/step - loss: 10.0975 - mae: 10.5911 - learning_rate: 2.5119e-08
Epoch 10/100
10/10  1s 83ms/step - loss: 9.5837 - mae: 10.0767 - learning_rate: 2.8184e-08
Epoch 11/100
10/10  1s 83ms/step - loss: 9.2045 - mae: 9.6968 - learning_rate: 3.1623e-08
Epoch 12/100
10/10  1s 81ms/step - loss: 8.8025 - mae: 9.2950 - learning_rate: 3.5481e-08
Epoch 13/100
10/10  1s 87ms/step - loss: 8.4456 - mae: 8.9376 - learning_rate: 3.9811e-08
Epoch 14/100
10/10  1s 89ms/step - loss: 8.1029 - mae: 8.5943 - learning_rate: 4.4668e-08
Epoch 15/100
10/10  1s 84ms/step - loss: 7.6820 - mae: 8.1726 - learning_rate: 5.0119e-08
Epoch 16/100
10/10  1s 78ms/step - loss: 7.2288 - mae: 7.7187 - learning_rate: 5.6234e-08
Epoch 17/100
10/10  1s 79ms/step - loss: 6.8206 - mae: 7.3099 - learning_rate: 6.3096e-08
Epoch 18/100
10/10  1s 82ms/step - loss: 6.3297 - mae: 6.8177 - learning_rate: 7.0795e-08
Epoch 19/100
10/10  1s 81ms/step - loss: 5.7399 - mae: 6.2262 - learning_rate: 7.9433e-08
Epoch 20/100
10/10  1s 76ms/step - loss: 4.9741 - mae: 5.4568 - learning_rate: 8.9125e-08


Epoch 21/100
10/10  1s 78ms/step - loss: 4.2253 - mae: 4.7038 - learning_rate: 1.0000e-07
Epoch 22/100
10/10  1s 84ms/step - loss: 3.7423 - mae: 4.2140 - learning_rate: 1.1220e-07
Epoch 23/100
10/10  1s 82ms/step - loss: 3.4789 - mae: 3.9478 - learning_rate: 1.2589e-07
Epoch 24/100
10/10  1s 81ms/step - loss: 3.3693 - mae: 3.8386 - learning_rate: 1.4125e-07
Epoch 25/100
10/10  1s 76ms/step - loss: 3.2852 - mae: 3.7539 - learning_rate: 1.5849e-07
Epoch 26/100
10/10  1s 81ms/step - loss: 3.2229 - mae: 3.6915 - learning_rate: 1.7783e-07
Epoch 27/100
10/10  1s 84ms/step - loss: 3.1560 - mae: 3.6245 - learning_rate: 1.9953e-07
Epoch 28/100
10/10  1s 82ms/step - loss: 3.0762 - mae: 3.5445 - learning_rate: 2.2387e-07
Epoch 29/100
10/10  1s 77ms/step - loss: 3.0137 - mae: 3.4811 - learning_rate: 2.5119e-07
Epoch 30/100
10/10  1s 84ms/step - loss: 2.9452 - mae: 3.4126 - learning_rate: 2.8184e-07
Epoch 31/100
10/10  1s 89ms/step - loss: 2.8888 - mae: 3.3558 - learning_rate: 3.1623e-07
Epoch 32/100
10/10  1s 79ms/step - loss: 2.8185 - mae: 3.2851 - learning_rate: 3.5481e-07
Epoch 33/100
10/10  1s 77ms/step - loss: 2.7479 - mae: 3.2145 - learning_rate: 3.9811e-07
Epoch 34/100
10/10  1s 78ms/step - loss: 2.6805 - mae: 3.1466 - learning_rate: 4.4668e-07
Epoch 35/100
10/10  1s 80ms/step - loss: 2.6174 - mae: 3.0833 - learning_rate: 5.0119e-07
Epoch 36/100
10/10  1s 81ms/step - loss: 2.5663 - mae: 3.0306 - learning_rate: 5.6234e-07
Epoch 37/100
10/10  1s 82ms/step - loss: 2.5062 - mae: 2.9699 - learning_rate: 6.3096e-07
Epoch 38/100
10/10  1s 79ms/step - loss: 2.4638 - mae: 2.9267 - learning_rate: 7.0795e-07
Epoch 39/100
10/10  1s 79ms/step - loss: 2.4025 - mae: 2.8645 - learning_rate: 7.9433e-07
Epoch 40/100
10/10  1s 83ms/step - loss: 2.3531 - mae: 2.8141 - learning_rate: 8.9125e-07


Epoch 41/100
10/10  1s 86ms/step - loss: 2.3230 - mae: 2.7839 - learning_rate: 1.0000e-06


Epoch 42/100
10/10  1s 81ms/step - loss: 2.2806 - mae: 2.7411 - learning_rate: 1.1220e-06


Epoch 43/100
10/10  1s 80ms/step - loss: 2.2572 - mae: 2.7171 - learning_rate: 1.2589e-06


Epoch 44/100
10/10  1s 79ms/step - loss: 2.2261 - mae: 2.6858 - learning_rate: 1.4125e-06


Epoch 45/100
10/10  1s 83ms/step - loss: 2.2069 - mae: 2.6654 - learning_rate: 1.5849e-06


Epoch 46/100
10/10  1s 81ms/step - loss: 2.1651 - mae: 2.6229 - learning_rate: 1.7783e-06


Epoch 47/100
10/10  1s 83ms/step - loss: 2.1422 - mae: 2.6001 - learning_rate: 1.9953e-06


Epoch 48/100
10/10  1s 85ms/step - loss: 2.1269 - mae: 2.5846 - learning_rate: 2.2387e-06


Epoch 49/100
10/10  1s 82ms/step - loss: 2.1287 - mae: 2.5865 - learning_rate: 2.5119e-06


Epoch 50/100
10/10  1s 79ms/step - loss: 2.1045 - mae: 2.5620 - learning_rate: 2.8184e-06


Epoch 51/100
10/10  1s 80ms/step - loss: 2.0613 - mae: 2.5186 - learning_rate: 3.1623e-06


Epoch 52/100
10/10  1s 77ms/step - loss: 2.0376 - mae: 2.4941 - learning_rate: 3.5481e-06


Epoch 53/100
10/10  1s 80ms/step - loss: 2.0580 - mae: 2.5152 - learning_rate: 3.9811e-06


Epoch 54/100
10/10  1s 81ms/step - loss: 2.0157 - mae: 2.4719 - learning_rate: 4.4668e-06


Epoch 55/100
10/10  1s 81ms/step - loss: 2.0333 - mae: 2.4904 - learning_rate: 5.0119e-06


Epoch 56/100
10/10  1s 83ms/step - loss: 2.0090 - mae: 2.4649 - learning_rate: 5.6234e-06


Epoch 57/100
10/10  1s 81ms/step - loss: 1.9967 - mae: 2.4523 - learning_rate: 6.3096e-06


Epoch 58/100
10/10  1s 78ms/step - loss: 1.9771 - mae: 2.4321 - learning_rate: 7.0795e-06


Epoch 59/100
10/10  1s 84ms/step - loss: 1.9844 - mae: 2.4395 - learning_rate: 7.9433e-06


Epoch 60/100
10/10  1s 81ms/step - loss: 2.3245 - mae: 2.7877 - learning_rate: 8.9125e-06


Epoch 61/100
10/10  1s 81ms/step - loss: 2.1920 - mae: 2.6521 - learning_rate: 1.0000e-05


Epoch 62/100
10/10  1s 76ms/step - loss: 2.4967 - mae: 2.9620 - learning_rate: 1.1220e-05


Epoch 63/100
10/10  1s 82ms/step - loss: 2.3114 - mae: 2.7731 - learning_rate: 1.2589e-05


Epoch 64/100
10/10  1s 82ms/step - loss: 2.8535 - mae: 3.3217 - learning_rate: 1.4125e-05


Epoch 65/100
10/10  1s 83ms/step - loss: 2.6423 - mae: 3.1099 - learning_rate: 1.5849e-05


Epoch 66/100
10/10  1s 75ms/step - loss: 2.4926 - mae: 2.9562 - learning_rate: 1.7783e-05


Epoch 67/100
10/10  1s 77ms/step - loss: 3.1036 - mae: 3.5783 - learning_rate: 1.9953e-05


Epoch 68/100
10/10  1s 79ms/step - loss: 3.1282 - mae: 3.6025 - learning_rate: 2.2387e-05


Epoch 69/100
10/10  1s 81ms/step - loss: 3.4396 - mae: 3.9183 - learning_rate: 2.5119e-05


Epoch 70/100
10/10  1s 80ms/step - loss: 3.4881 - mae: 3.9676 - learning_rate: 2.8184e-05


Epoch 71/100
10/10  1s 78ms/step - loss: 3.2570 - mae: 3.7310 - learning_rate: 3.1623e-05


Epoch 72/100
10/10  1s 77ms/step - loss: 3.8629 - mae: 4.3460 - learning_rate: 3.5481e-05


Epoch 73/100
10/10  1s 80ms/step - loss: 3.7167 - mae: 4.1975 - learning_rate: 3.9811e-05


Epoch 74/100
10/10  1s 82ms/step - loss: 3.9282 - mae: 4.4084 - learning_rate: 4.4668e-05


Epoch 75/100
10/10  1s 82ms/step - loss: 5.0462 - mae: 5.5238 - learning_rate: 5.0119e-05





















Epoch 76/100
10/10  1s 83ms/step - loss: 5.6113 - mae: 6.0957 - learning_rate: 5.6234e-05

Epoch 77/100
10/10  1s 86ms/step - loss: 3.2996 - mae: 3.7728 - learning_rate: 6.3096e-05

Epoch 78/100
10/10  1s 82ms/step - loss: 2.6073 - mae: 3.0727 - learning_rate: 7.0795e-05

Epoch 79/100
10/10  1s 79ms/step - loss: 2.3327 - mae: 2.7939 - learning_rate: 7.9433e-05

Epoch 80/100
10/10  1s 81ms/step - loss: 2.2303 - mae: 2.6889 - learning_rate: 8.9125e-05

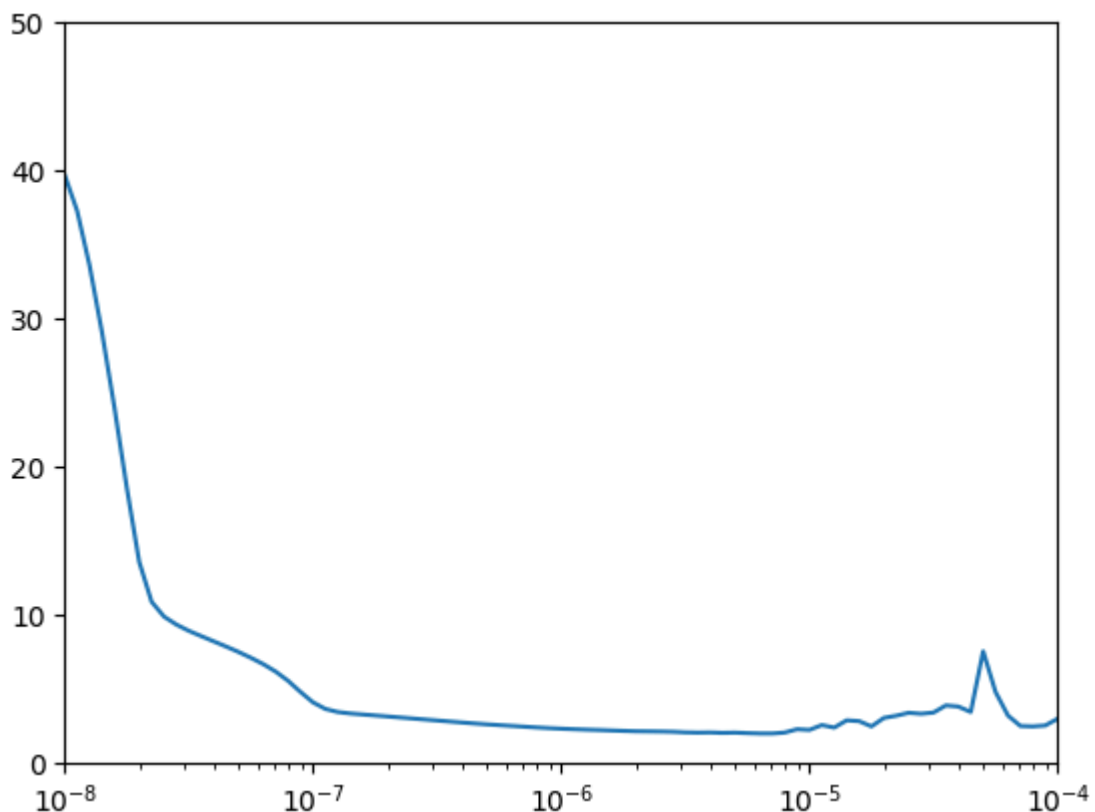
Epoch 81/100
10/10  1s 86ms/step - loss: 2.7356 - mae: 3.2020 - learning_rate: 1.0000e-04
Epoch 82/100
10/10  1s 86ms/step - loss: 3.7281 - mae: 4.2081 - learning_rate: 1.1220e-04
Epoch 83/100
10/10  1s 80ms/step - loss: 4.3906 - mae: 4.8751 - learning_rate: 1.2589e-04
Epoch 84/100
10/10  1s 78ms/step - loss: 4.8378 - mae: 5.3251 - learning_rate: 1.4125e-04
Epoch 85/100
10/10  1s 83ms/step - loss: 5.7163 - mae: 6.2063 - learning_rate: 1.5849e-04
Epoch 86/100
10/10  1s 79ms/step - loss: 7.1655 - mae: 7.6580 - learning_rate: 1.7783e-04
Epoch 87/100
10/10  1s 81ms/step - loss: 13.2385 - mae: 13.7257 - learning_rate: 1.9953e-04
Epoch 88/100
10/10  1s 79ms/step - loss: 37.3979 - mae: 37.8905 - learning_rate: 2.2387e-04
Epoch 89/100
10/10  1s 77ms/step - loss: 44.1793 - mae: 44.6793 - learning_rate: 2.5119e-04
Epoch 90/100
10/10  1s 78ms/step - loss: 39.3084 - mae: 39.8083 - learning_rate: 2.8184e-04
Epoch 91/100
10/10  1s 88ms/step - loss: 64.0156 - mae: 64.5148 - learning_rate: 3.1623e-04
Epoch 92/100
10/10  1s 79ms/step - loss: 52.3734 - mae: 52.8734 - learning_rate: 3.5481e-04
Epoch 93/100
10/10  1s 79ms/step - loss: 30.3545 - mae: 30.8470 - learning_rate: 3.9811e-04
Epoch 94/100
10/10  1s 83ms/step - loss: 36.0649 - mae: 36.5649 - learning_rate: 4.4668e-04
Epoch 95/100
10/10  1s 82ms/step - loss: 42.3138 - mae: 42.8138 - learning_rate: 5.0119e-04
Epoch 96/100
10/10  1s 80ms/step - loss: 44.1372 - mae: 44.6372 - learning_rate: 5.6234e-04
Epoch 97/100
10/10  1s 74ms/step - loss: 53.5830 - mae: 54.0830 - learning_rate: 6.3096e-04
Epoch 98/100
10/10  1s 81ms/step - loss: 56.0262 - mae: 56.5262 - learning_rate: 7.0795e-04
Epoch 99/100
10/10  1s 85ms/step - loss: 67.1610 - mae: 67.6610 - learning_rate: 7.9433e-04
Epoch 100/100
10/10  1s 79ms/step - loss: 71.0739 - mae: 71.5739 - learning_rate: 8.9125e-04

6. Actualización del learning rate según los resultados obtenidos del primer entrenamiento de la red neuronal

Después del entrenamiento de la red neuronal se ve que learning rate resultante es de $1e-5$. Se visualizará gráficamente para entender el motivo por el que se ha usado ese valor. En la gráfica se puede ver cómo el learning rate con el que menos loss hay es $1e-5$, y por ese motivo, se debe volver a entrenar la red neuronal con dicho learning rate.

```
In [ ]: plt.semilogx(history.history["learning_rate"], history.history["loss"])
plt.axis([1e-8, 1e-4, 0, 50])
```

```
Out[ ]: (1e-08, 0.0001, 0.0, 50.0)
```



Se vuelve a inicializar la sesión de entrenamiento y la variable train_set:

```
In [ ]: tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
```

Ejercicio 6 (0.5 puntos): Para crear el nuevo modelo, reutiliza la red neuronal diseñada en el ejercicio 4, pero esta vez utilizando 60 filtros en la capa de convolución.

```
In [ ]: window_size = 60
batch_size = 100
shuffle_buffer_size = 1000

train_set = windowed_dataset(x_train, window_size=60, batch_size=100, shuffle_bu
```



```
# Configuración del modelo con 60 filtros en la capa de convolución
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=60, kernel_size=5, strides=1, padding="causal",
                           activation="relu", input_shape=[None, 1]),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 400)
])
```

c:\Anaconda\envs\Sist_Cognitivos_Artificiales\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Ejercicio 7 (0.5 puntos): Se debe volver a compilar la red neuronal de manera análoga a la del ejercicio 5, pero esta vez utilizar un learning rate obtenido de la función callback.

```
In [ ]: # Suponiendo que el Learning rate óptimo encontrado es 1e-5
        optimal_learning_rate = 1e-5

        # Compilación de la red neuronal con el Learning rate obtenido
        model.compile(
            loss=tf.keras.losses.Huber(), # Uso de la función de pérdida de Huber
            optimizer=tf.keras.optimizers.SGD(learning_rate=optimal_learning_rate, momentum=0.9),
            metrics=["mae"] # Métrica de error absoluto medio
        )

        # Resumen del modelo para confirmar la configuración
        model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, None, 60)	360
lstm (LSTM)	(None, None, 64)	32,000
lstm_1 (LSTM)	(None, None, 64)	33,024
dense (Dense)	(None, None, 30)	1,950
dense_1 (Dense)	(None, None, 10)	310
dense_2 (Dense)	(None, None, 1)	11
lambda (Lambda)	(None, None, 1)	0

◀ ————— ▶


Total params: 67,655 (264.28 KB)

Trainable params: 67,655 (264.28 KB)


Non-trainable params: 0 (0.00 B)

```
In [ ]: history = model.fit(train_set, epochs=150)
```

Epoch 1/150































25/25  5s 49ms/step - loss: 11.2141 - mae: 11.7043

Epoch 2/150

1/25  3s 152ms/step - loss: 5.3948 - mae: 5.8846

c:\Anaconda\envs\Sist_Cognitivos_Artificiales\Lib\contextlib.py:158: UserWarning:
Your input ran out of data; interrupting training. Make sure that your dataset or
generator can generate at least `steps_per_epoch * epochs` batches. You may need
to use the `.repeat()` function when building your dataset.

self.gen.throw(typ, value, traceback)

25/25		1s	44ms/step	-	loss: 5.2472	-	mae: 5.7349
Epoch 3/150							
25/25		1s	45ms/step	-	loss: 4.0819	-	mae: 4.5621
Epoch 4/150							
25/25		1s	46ms/step	-	loss: 3.8240	-	mae: 4.3030
Epoch 5/150							
25/25		1s	42ms/step	-	loss: 3.3312	-	mae: 3.8070
Epoch 6/150							
25/25		1s	48ms/step	-	loss: 3.0216	-	mae: 3.4950
Epoch 7/150							
25/25		1s	49ms/step	-	loss: 2.6062	-	mae: 3.0741
Epoch 8/150							
25/25		1s	46ms/step	-	loss: 2.4253	-	mae: 2.8897
Epoch 9/150							
25/25		1s	48ms/step	-	loss: 1.8539	-	mae: 2.3060
Epoch 10/150							
25/25		1s	52ms/step	-	loss: 1.7693	-	mae: 2.2186
Epoch 11/150							
25/25		1s	53ms/step	-	loss: 1.7224	-	mae: 2.1708
Epoch 12/150							
25/25		1s	49ms/step	-	loss: 1.7113	-	mae: 2.1588
Epoch 13/150							
25/25		1s	55ms/step	-	loss: 1.6914	-	mae: 2.1382
Epoch 14/150							
25/25		1s	52ms/step	-	loss: 1.7094	-	mae: 2.1572
Epoch 15/150							
25/25		2s	55ms/step	-	loss: 1.6869	-	mae: 2.1336
Epoch 16/150							
25/25		2s	60ms/step	-	loss: 1.6575	-	mae: 2.1040
Epoch 17/150							
25/25		2s	66ms/step	-	loss: 1.6591	-	mae: 2.1060
Epoch 18/150							
25/25		2s	67ms/step	-	loss: 1.6497	-	mae: 2.0961
Epoch 19/150							
25/25		2s	66ms/step	-	loss: 1.6285	-	mae: 2.0737
Epoch 20/150							
25/25		2s	54ms/step	-	loss: 1.6508	-	mae: 2.0979
Epoch 21/150							
25/25		1s	55ms/step	-	loss: 1.6186	-	mae: 2.0642
Epoch 22/150							
25/25		1s	49ms/step	-	loss: 1.6095	-	mae: 2.0544
Epoch 23/150							
25/25		1s	52ms/step	-	loss: 1.6046	-	mae: 2.0493
Epoch 24/150							
25/25		1s	50ms/step	-	loss: 1.5996	-	mae: 2.0445
Epoch 25/150							
25/25		1s	45ms/step	-	loss: 1.5856	-	mae: 2.0296
Epoch 26/150							
25/25		1s	46ms/step	-	loss: 1.5930	-	mae: 2.0382
Epoch 27/150							
25/25		1s	44ms/step	-	loss: 1.6054	-	mae: 2.0507
Epoch 28/150							
25/25		1s	47ms/step	-	loss: 1.5748	-	mae: 2.0191
Epoch 29/150							
25/25		2s	60ms/step	-	loss: 1.5707	-	mae: 2.0146
Epoch 30/150							
25/25		1s	52ms/step	-	loss: 1.5789	-	mae: 2.0227
Epoch 31/150							
25/25		1s	53ms/step	-	loss: 1.5679	-	mae: 2.0119
Epoch 32/150							

25/25	—————	2s	56ms/step	-	loss: 1.5804	-	mae: 2.0248
Epoch 33/150							
25/25	—————	2s	58ms/step	-	loss: 1.5838	-	mae: 2.0280
Epoch 34/150							
25/25	—————	1s	51ms/step	-	loss: 1.5614	-	mae: 2.0056
Epoch 35/150							
25/25	—————	1s	50ms/step	-	loss: 1.5644	-	mae: 2.0080
Epoch 36/150							
25/25	—————	1s	51ms/step	-	loss: 1.5624	-	mae: 2.0069
Epoch 37/150							
25/25	—————	1s	44ms/step	-	loss: 1.5546	-	mae: 1.9987
Epoch 38/150							
25/25	—————	1s	45ms/step	-	loss: 1.5473	-	mae: 1.9908
Epoch 39/150							
25/25	—————	1s	51ms/step	-	loss: 1.5636	-	mae: 2.0068
Epoch 40/150							
25/25	—————	1s	47ms/step	-	loss: 1.5452	-	mae: 1.9890
Epoch 41/150							
25/25	—————	1s	43ms/step	-	loss: 1.5505	-	mae: 1.9936
Epoch 42/150							
25/25	—————	1s	45ms/step	-	loss: 1.5461	-	mae: 1.9883
Epoch 43/150							
25/25	—————	1s	50ms/step	-	loss: 1.5353	-	mae: 1.9778
Epoch 44/150							
25/25	—————	1s	53ms/step	-	loss: 1.5429	-	mae: 1.9863
Epoch 45/150							
25/25	—————	1s	45ms/step	-	loss: 1.5748	-	mae: 2.0192
Epoch 46/150							
25/25	—————	1s	43ms/step	-	loss: 1.5351	-	mae: 1.9782
Epoch 47/150							
25/25	—————	1s	43ms/step	-	loss: 1.5327	-	mae: 1.9756
Epoch 48/150							
25/25	—————	1s	42ms/step	-	loss: 1.5318	-	mae: 1.9743
Epoch 49/150							
25/25	—————	1s	43ms/step	-	loss: 1.5357	-	mae: 1.9785
Epoch 50/150							
25/25	—————	1s	46ms/step	-	loss: 1.5378	-	mae: 1.9805
Epoch 51/150							
25/25	—————	2s	57ms/step	-	loss: 1.5412	-	mae: 1.9846
Epoch 52/150							
25/25	—————	2s	58ms/step	-	loss: 1.5289	-	mae: 1.9717
Epoch 53/150							
25/25	—————	2s	67ms/step	-	loss: 1.5572	-	mae: 2.0021
Epoch 54/150							
25/25	—————	2s	56ms/step	-	loss: 1.5734	-	mae: 2.0191
Epoch 55/150							
25/25	—————	2s	61ms/step	-	loss: 1.5175	-	mae: 1.9595
Epoch 56/150							
25/25	—————	1s	54ms/step	-	loss: 1.5163	-	mae: 1.9583
Epoch 57/150							
25/25	—————	1s	49ms/step	-	loss: 1.5340	-	mae: 1.9766
Epoch 58/150							
25/25	—————	1s	43ms/step	-	loss: 1.5343	-	mae: 1.9761
Epoch 59/150							
25/25	—————	1s	46ms/step	-	loss: 1.5175	-	mae: 1.9593
Epoch 60/150							
25/25	—————	1s	45ms/step	-	loss: 1.5208	-	mae: 1.9629
Epoch 61/150							
25/25	—————	1s	49ms/step	-	loss: 1.5141	-	mae: 1.9553
Epoch 62/150							

25/25	1s 45ms/step	loss: 1.5214	mae: 1.9634
Epoch 63/150			
25/25	1s 44ms/step	loss: 1.5391	mae: 1.9814
Epoch 64/150			
25/25	1s 47ms/step	loss: 1.5168	mae: 1.9586
Epoch 65/150			
25/25	2s 56ms/step	loss: 1.5072	mae: 1.9485
Epoch 66/150			
25/25	1s 48ms/step	loss: 1.5391	mae: 1.9833
Epoch 67/150			
25/25	2s 62ms/step	loss: 1.5302	mae: 1.9726
Epoch 68/150			
25/25	2s 71ms/step	loss: 1.5275	mae: 1.9700
Epoch 69/150			
25/25	2s 71ms/step	loss: 1.5108	mae: 1.9522
Epoch 70/150			
25/25	1s 51ms/step	loss: 1.5065	mae: 1.9472
Epoch 71/150			
25/25	2s 56ms/step	loss: 1.5520	mae: 1.9960
Epoch 72/150			
25/25	2s 56ms/step	loss: 1.5086	mae: 1.9504
Epoch 73/150			
25/25	1s 47ms/step	loss: 1.5295	mae: 1.9725
Epoch 74/150			
25/25	1s 45ms/step	loss: 1.5063	mae: 1.9474
Epoch 75/150			
25/25	1s 48ms/step	loss: 1.5113	mae: 1.9532
Epoch 76/150			
25/25	1s 42ms/step	loss: 1.5128	mae: 1.9542
Epoch 77/150			
25/25	1s 53ms/step	loss: 1.5227	mae: 1.9651
Epoch 78/150			
25/25	1s 47ms/step	loss: 1.5065	mae: 1.9475
Epoch 79/150			
25/25	1s 55ms/step	loss: 1.5102	mae: 1.9512
Epoch 80/150			
25/25	1s 48ms/step	loss: 1.5129	mae: 1.9542
Epoch 81/150			
25/25	1s 43ms/step	loss: 1.5234	mae: 1.9651
Epoch 82/150			
25/25	1s 42ms/step	loss: 1.5209	mae: 1.9628
Epoch 83/150			
25/25	1s 55ms/step	loss: 1.5163	mae: 1.9573
Epoch 84/150			
25/25	2s 55ms/step	loss: 1.5176	mae: 1.9593
Epoch 85/150			
25/25	1s 51ms/step	loss: 1.5233	mae: 1.9661
Epoch 86/150			
25/25	1s 47ms/step	loss: 1.5096	mae: 1.9506
Epoch 87/150			
25/25	1s 47ms/step	loss: 1.5010	mae: 1.9419
Epoch 88/150			
25/25	1s 48ms/step	loss: 1.5019	mae: 1.9422
Epoch 89/150			
25/25	1s 47ms/step	loss: 1.5263	mae: 1.9690
Epoch 90/150			
25/25	1s 50ms/step	loss: 1.4961	mae: 1.9364
Epoch 91/150			
25/25	1s 50ms/step	loss: 1.5184	mae: 1.9602
Epoch 92/150			

25/25	1s 47ms/step	loss: 1.4943	mae: 1.9346
Epoch 93/150			
25/25	1s 47ms/step	loss: 1.5145	mae: 1.9569
Epoch 94/150			
25/25	1s 54ms/step	loss: 1.5100	mae: 1.9515
Epoch 95/150			
25/25	1s 49ms/step	loss: 1.5153	mae: 1.9576
Epoch 96/150			
25/25	1s 54ms/step	loss: 1.4954	mae: 1.9358
Epoch 97/150			
25/25	2s 58ms/step	loss: 1.5139	mae: 1.9562
Epoch 98/150			
25/25	1s 55ms/step	loss: 1.5109	mae: 1.9521
Epoch 99/150			
25/25	1s 46ms/step	loss: 1.5316	mae: 1.9747
Epoch 100/150			
25/25	1s 44ms/step	loss: 1.5066	mae: 1.9471
Epoch 101/150			
25/25	1s 45ms/step	loss: 1.5255	mae: 1.9685
Epoch 102/150			
25/25	1s 47ms/step	loss: 1.5219	mae: 1.9641
Epoch 103/150			
25/25	1s 44ms/step	loss: 1.4998	mae: 1.9403
Epoch 104/150			
25/25	1s 45ms/step	loss: 1.4997	mae: 1.9397
Epoch 105/150			
25/25	1s 47ms/step	loss: 1.4948	mae: 1.9346
Epoch 106/150			
25/25	1s 43ms/step	loss: 1.5039	mae: 1.9443
Epoch 107/150			
25/25	1s 45ms/step	loss: 1.4978	mae: 1.9394
Epoch 108/150			
25/25	1s 45ms/step	loss: 1.5017	mae: 1.9416
Epoch 109/150			
25/25	1s 45ms/step	loss: 1.5104	mae: 1.9526
Epoch 110/150			
25/25	1s 51ms/step	loss: 1.4947	mae: 1.9354
Epoch 111/150			
25/25	1s 46ms/step	loss: 1.4993	mae: 1.9393
Epoch 112/150			
25/25	2s 56ms/step	loss: 1.4966	mae: 1.9374
Epoch 113/150			
25/25	1s 45ms/step	loss: 1.4871	mae: 1.9268
Epoch 114/150			
25/25	1s 53ms/step	loss: 1.4895	mae: 1.9301
Epoch 115/150			
25/25	1s 48ms/step	loss: 1.4967	mae: 1.9378
Epoch 116/150			
25/25	1s 50ms/step	loss: 1.4829	mae: 1.9226
Epoch 117/150			
25/25	1s 46ms/step	loss: 1.4995	mae: 1.9401
Epoch 118/150			
25/25	1s 43ms/step	loss: 1.4956	mae: 1.9360
Epoch 119/150			
25/25	1s 44ms/step	loss: 1.5005	mae: 1.9405
Epoch 120/150			
25/25	1s 43ms/step	loss: 1.4985	mae: 1.9389
Epoch 121/150			
25/25	1s 52ms/step	loss: 1.4966	mae: 1.9366
Epoch 122/150			

25/25	<div></div>	1s 46ms/step	- loss: 1.4984	- mae: 1.9391
Epoch 123/150				
25/25	<div></div>	1s 43ms/step	- loss: 1.4899	- mae: 1.9297
Epoch 124/150				
25/25	<div></div>	1s 50ms/step	- loss: 1.4969	- mae: 1.9373
Epoch 125/150				
25/25	<div></div>	1s 42ms/step	- loss: 1.5057	- mae: 1.9467
Epoch 126/150				
25/25	<div></div>	1s 49ms/step	- loss: 1.5460	- mae: 1.9889
Epoch 127/150				
25/25	<div></div>	1s 46ms/step	- loss: 1.4983	- mae: 1.9398
Epoch 128/150				
25/25	<div></div>	1s 43ms/step	- loss: 1.4909	- mae: 1.9307
Epoch 129/150				
25/25	<div></div>	1s 43ms/step	- loss: 1.4940	- mae: 1.9337
Epoch 130/150				
25/25	<div></div>	1s 45ms/step	- loss: 1.5200	- mae: 1.9620
Epoch 131/150				
25/25	<div></div>	1s 42ms/step	- loss: 1.4943	- mae: 1.9341
Epoch 132/150				
25/25	<div></div>	2s 57ms/step	- loss: 1.4947	- mae: 1.9348
Epoch 133/150				
25/25	<div></div>	1s 53ms/step	- loss: 1.4923	- mae: 1.9319
Epoch 134/150				
25/25	<div></div>	1s 54ms/step	- loss: 1.4848	- mae: 1.9246
Epoch 135/150				
25/25	<div></div>	1s 46ms/step	- loss: 1.5172	- mae: 1.9585
Epoch 136/150				
25/25	<div></div>	1s 45ms/step	- loss: 1.4887	- mae: 1.9279
Epoch 137/150				
25/25	<div></div>	2s 57ms/step	- loss: 1.4961	- mae: 1.9378
Epoch 138/150				
25/25	<div></div>	2s 63ms/step	- loss: 1.4781	- mae: 1.9177
Epoch 139/150				
25/25	<div></div>	2s 76ms/step	- loss: 1.4984	- mae: 1.9383
Epoch 140/150				
25/25	<div></div>	2s 54ms/step	- loss: 1.4995	- mae: 1.9405
Epoch 141/150				
25/25	<div></div>	2s 57ms/step	- loss: 1.4930	- mae: 1.9323
Epoch 142/150				
25/25	<div></div>	1s 48ms/step	- loss: 1.4882	- mae: 1.9281
Epoch 143/150				
25/25	<div></div>	1s 50ms/step	- loss: 1.4809	- mae: 1.9202
Epoch 144/150				
25/25	<div></div>	1s 45ms/step	- loss: 1.4797	- mae: 1.9194
Epoch 145/150				
25/25	<div></div>	1s 43ms/step	- loss: 1.5025	- mae: 1.9426
Epoch 146/150				
25/25	<div></div>	1s 45ms/step	- loss: 1.5033	- mae: 1.9452
Epoch 147/150				
25/25	<div></div>	1s 43ms/step	- loss: 1.4883	- mae: 1.9285
Epoch 148/150				
25/25	<div></div>	1s 46ms/step	- loss: 1.5002	- mae: 1.9404
Epoch 149/150				
25/25	<div></div>	1s 42ms/step	- loss: 1.4898	- mae: 1.9303
Epoch 150/150				
25/25	<div></div>	1s 47ms/step	- loss: 1.4881	- mae: 1.9284

7. Predicción de los siguientes valores de la serie temporal

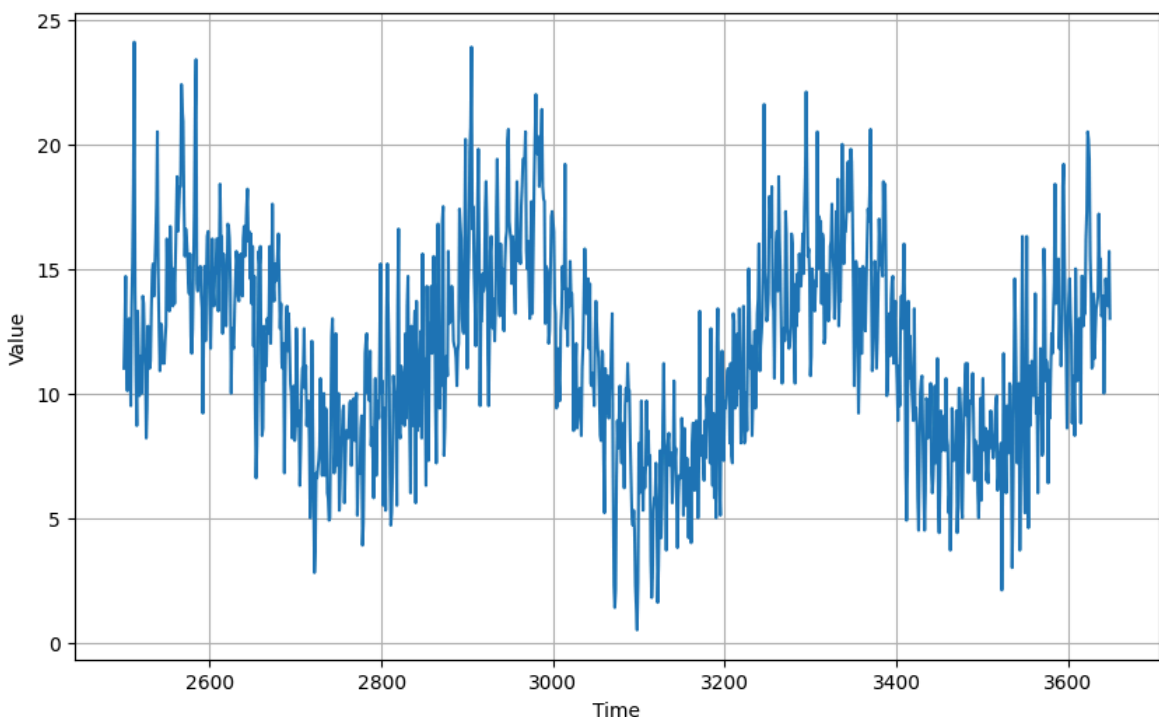
Para concluir la actividad, se usa el método `model_forecast` que se ha diseñado utilizando el método de la ventana temporal para hacer el nuevo método `rnn_forecast` con el cual se calcularán los nuevos valores de la serie temporal. Posteriormente, se pinta una gráfica para ver esos resultados y comprobar de forma visual que son correctos. Además, se dan los resultados de esas predicciones en forma numérica, de esta forma, este modelo diseñado en esta actividad podría ser el input de un nuevo algoritmo si fuera necesario.

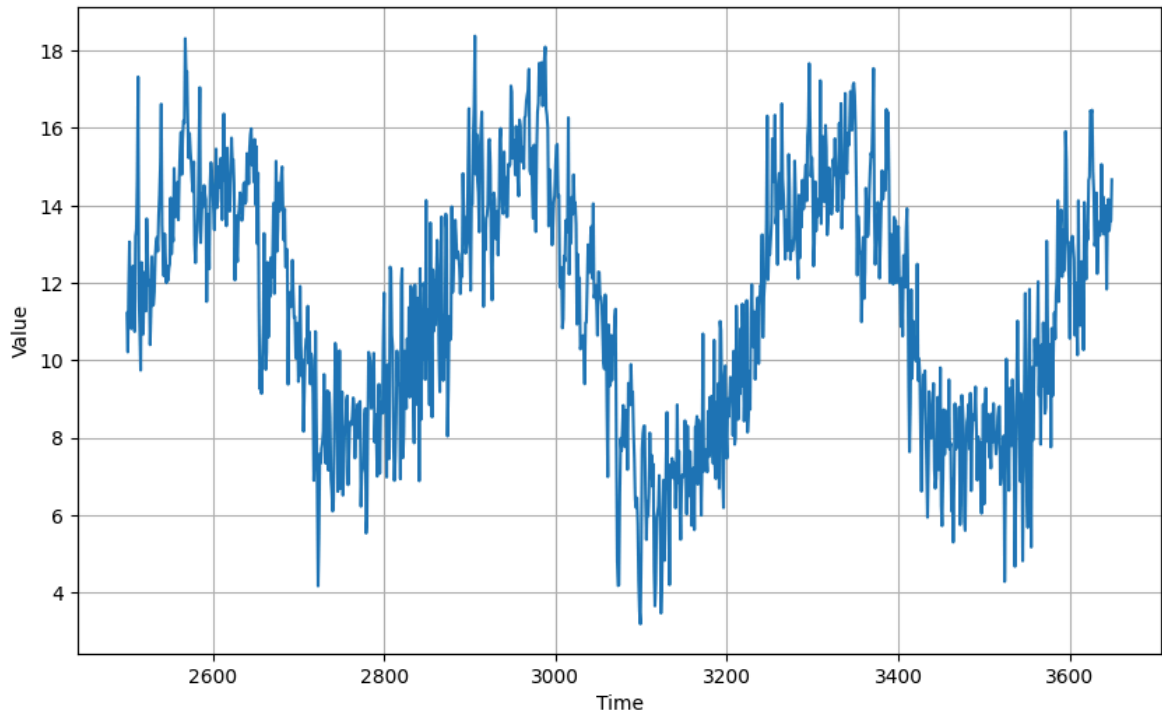
```
In [ ]: rnn_forecast = model_forecast(model, series[..., np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_time - window_size:-1, -1, 0]
```

113/113 ————— 1s 9ms/step

```
In [ ]: plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, rnn_forecast)
```

<Figure size 1000x600 with 0 Axes>





```
In [ ]: # Inicializamos la métrica
mae_metric = tf.keras.metrics.MeanAbsoluteError()

# Calculamos el MAE
mae_metric.update_state(x_valid, rnn_forecast)
mae_value = mae_metric.result().numpy() # Obtenemos el valor

print("Mean Absolute Error:", mae_value)
```

Mean Absolute Error: 1.8105675

```
In [ ]: print(rnn_forecast)
```

[11.218924 10.214475 11.901641 ... 13.611639 13.608533 14.664597]

8. Mostrar gráficamente los resultados.

Una vez obtenido el resultado de la actividad, se procede a revisar de forma gráfica el training y validation loss a lo largo de los epochs en este nuevo entrenamiento con el learning rate optimizado.

```
In [ ]: import matplotlib.image as mpimg
import matplotlib.pyplot as plt

#-----
# Recuperar una lista de resultados de la lista de datos de entrenamiento y prueba
#-----
loss=history.history['loss']

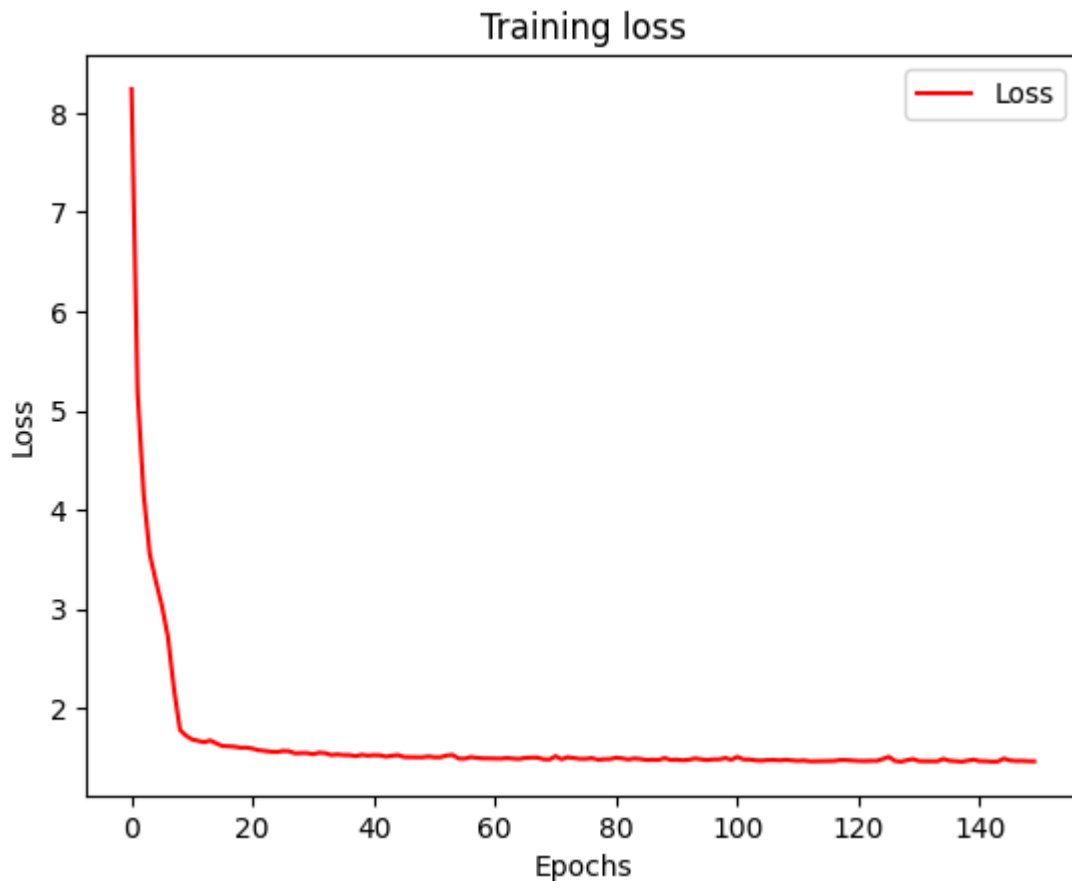
epochs=range(len(loss)) # Get number of epochs
```

A continuación se realiza el plot de la pérdida frente a los epochs

```
In [ ]: #-----
# Pérdida de entrenamiento y validación por epoch
#-----
plt.plot(epochs, loss, 'r')
plt.title('Training loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Loss"])

plt.figure()
```

Out[]: <Figure size 640x480 with 0 Axes>



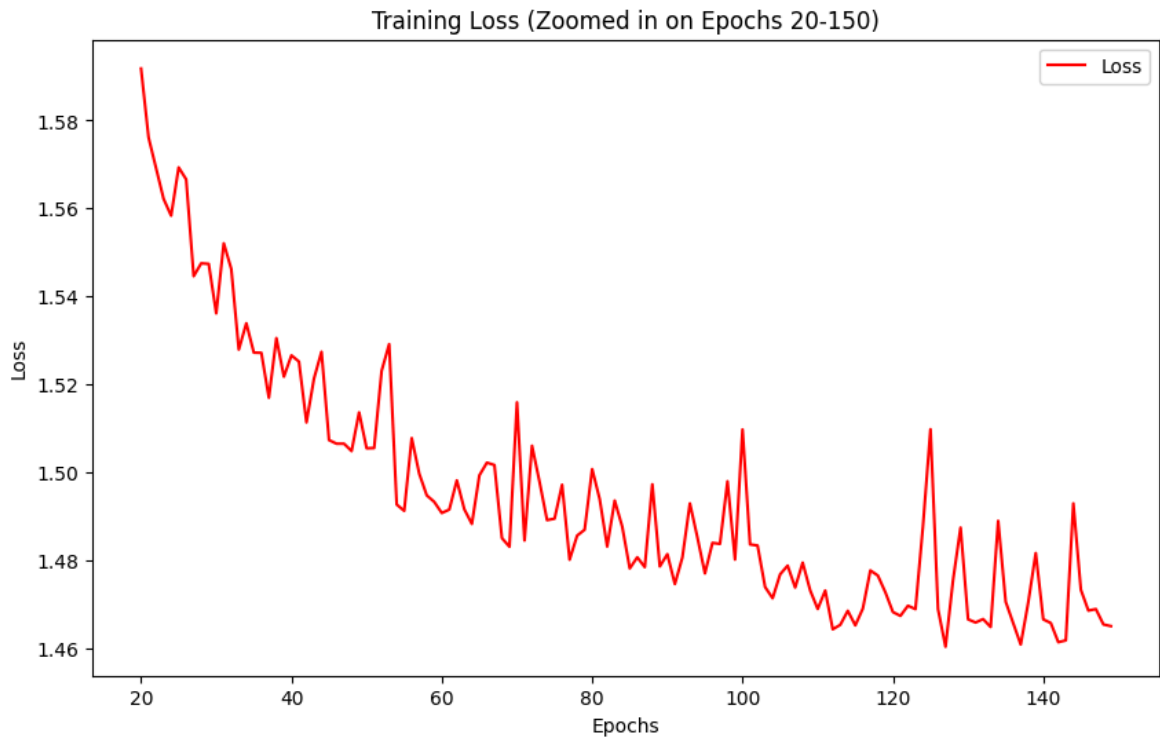
<Figure size 640x480 with 0 Axes>

Ejercicio 8 (0.5 punto): Utilizando las 2 nuevas variables `zoomed_loss` y `zoomed_epochs` y con base en el código anterior, hacer el plot del loss frente a los epochs entre los epoch 20 y 150 para ver como va oscilando y no es un proceso lineal como podría parecer según el anterior plot.

```
In [ ]: #-----
# Pérdida de entrenamiento y validación por epoch con zoom
#-----
zoomed_loss = loss[20:]
zoomed_epochs = range(20,150)

# Graficamos la pérdida de entrenamiento con zoom
plt.figure(figsize=(10, 6))
plt.plot(zoomed_epochs, zoomed_loss, 'r')
plt.title('Training Loss (Zoomed in on Epochs 20-150)')
plt.xlabel("Epochs")
plt.ylabel("Loss")
```

```
plt.legend(["Loss"])  
plt.show()
```



El gráfico obtenido representa el comportamiento de la función de pérdida en un segmento específico del entrenamiento. Es de utilidad para identificar sobreajustes, posibles ajustes de hiperparámetros o para comprender la dinámica del aprendizaje del modelo por epochs.