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.

Haz doble clic (o pulsa Intro) para editar

1. Descargar el dataset y almacenarlo

En primer lugar hay que importar tensorflow.

```
import tensorflow as tf
print(tf.__version__)
```

```
2.8.2
```

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.

```
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.

```
!wget --no-check-certificate \
    https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv
    -O /tmp/daily-min-temperatures.csv
```

```
--2022-09-16 19:08:36--  https://raw.githubusercontent.com/jbrownlee/Datasets/master/
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 1
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|
HTTP request sent, awaiting response... 200 OK
Length: 67921 (66K) [text/plain]
Saving to: ‘/tmp/daily-min-temperatures.csv’
```

```
/tmp/daily-min-temp 100%[===================>]  66.33K  19.6KB/s    in 3.4s

2022-09-16 19:08:41 (19.6 KB/s) - ‘/tmp/daily-min-temperatures.csv’ saved [67921/679
```

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.

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

with open('/tmp/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 ese 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

```
## variables para la técnica de la ventana temporal

#x es el eje, y time seria / valid es Test

split_time = 2500
window_size = 30
batch_size = 32
shuffle_buffer_size = 1000

## Split del dataset en entrenamiento y validación

## tu código para la creación de las 4 variables del ejercicio 1 aquí
#listo#
##Train set
time_train = time[:split_time]
x_train = series[:split_time]

##Test set
time_valid = time[split_time:]
x_valid = series[split_time:]
```

2. Creación del método **windowed_datset** 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.

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

```
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 linea 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.

```
def model_forecast(model, series, window_size):
## tu código para el método model_forecast del ejercicio 2 aquí
   ds = tf.data.Dataset.from_tensor_slices(series)
   ds = ds.window(window_size, shift=1,drop_remainder=True)
   ds = ds.flat_map(lambda w: w.batch(window_size))
   ds = ds.batch(32).prefetch(1)
   forecast = model.predict(ds)
   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.

```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
window_size = 64
batch_size = 256
```

4. Diseño de la red neuronal

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

```
## tu código aquí para el ejercicio 3
train_set = windowed_dataset(x_train,window_size,batch_size,shuffle_buffer_size)
```

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)

```
## tu código para la red neuronal del ejercicio 4 aquí

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(32,5,1,'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)])
```

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** (epoch / 20) **texto en negrita**

```
## tu código para crear la variable lr_schedule aquí
#Se modifico la formula de acuerdo a la clase de repaso.
#lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8*10*(epoch/20))
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)

```
## tu código para compilar la red neuronal para el ejercicio 5 aquí
model.compile(
    loss='huber',
    optimizer=tf.keras.optimizers.SGD(learning_rate=1e-8,momentum=0.9),
    metrics=["mae"]
)
```

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

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

```
10/10 [==============================] - 0s 30ms/step - loss: 14.9020 - mae: 15.39
Epoch 72/100
10/10 [==============================] - 0s 30ms/step - loss: 10.2480 - mae: 10.74
Epoch 73/100
10/10 [==============================] - 0s 30ms/step - loss: 9.1635 - mae: 9.6592
Epoch 74/100
10/10 [==============================] - 0s 31ms/step - loss: 7.3594 - mae: 7.8523
Epoch 75/100
10/10 [==============================] - 0s 30ms/step - loss: 4.5741 - mae: 5.0546
Epoch 76/100
10/10 [==============================] - 0s 30ms/step - loss: 3.0283 - mae: 3.4984
Epoch 77/100
10/10 [==============================] - 0s 32ms/step - loss: 2.7366 - mae: 3.2035
Epoch 78/100
10/10 [==============================] - 0s 31ms/step - loss: 2.5222 - mae: 2.9867
Epoch 79/100
10/10 [==============================] - 0s 33ms/step - loss: 2.1306 - mae: 2.5883
Epoch 80/100
```

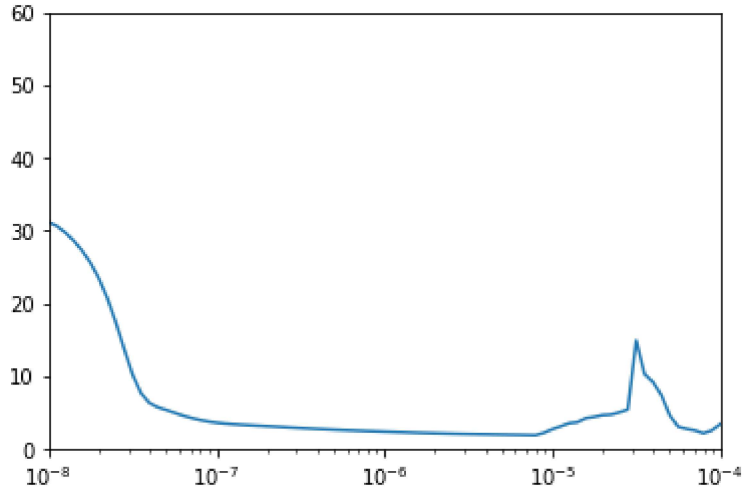
```
10/10 [==============================] - 0s 31ms/step - loss: 2.5095 - mae: 2.9739
Epoch 81/100
10/10 [==============================] - 0s 32ms/step - loss: 3.3784 - mae: 3.8569
Epoch 82/100
10/10 [==============================] - 0s 30ms/step - loss: 3.9892 - mae: 4.4741
Epoch 83/100
10/10 [==============================] - 0s 33ms/step - loss: 4.5495 - mae: 5.0379
Epoch 84/100
10/10 [==============================] - 0s 30ms/step - loss: 5.2434 - mae: 5.7347
Epoch 85/100
10/10 [==============================] - 0s 29ms/step - loss: 5.8185 - mae: 6.3037
Epoch 86/100
10/10 [==============================] - 0s 29ms/step - loss: 15.6072 - mae: 16.10
Epoch 87/100
10/10 [==============================] - 0s 32ms/step - loss: 16.2911 - mae: 16.79
Epoch 88/100
10/10 [==============================] - 0s 30ms/step - loss: 19.5820 - mae: 20.07
Epoch 89/100
10/10 [==============================] - 0s 30ms/step - loss: 35.0403 - mae: 35.53
Epoch 90/100
10/10 [==============================] - 0s 29ms/step - loss: 46.2379 - mae: 46.73
Epoch 91/100
10/10 [==============================] - 0s 30ms/step - loss: 48.9835 - mae: 49.48
Epoch 92/100
10/10 [==============================] - 0s 33ms/step - loss: 50.3362 - mae: 50.83
Epoch 93/100
10/10 [==============================] - 0s 31ms/step - loss: 59.8404 - mae: 60.34
Epoch 94/100
10/10 [==============================] - 0s 32ms/step - loss: 70.7893 - mae: 71.28
Epoch 95/100
10/10 [==============================] - 0s 30ms/step - loss: 76.7292 - mae: 77.22
Epoch 96/100
10/10 [==============================] - 0s 30ms/step - loss: 71.8077 - mae: 72.30
Epoch 97/100
10/10 [==============================] - 0s 32ms/step - loss: 38.0759 - mae: 38.56
Epoch 98/100
10/10 [==============================] - 0s 31ms/step - loss: 54.5507 - mae: 55.04
Epoch 99/100
10/10 [==============================] - 0s 33ms/step - loss: 89.6863 - mae: 90.18
```

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.

```
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-8, 1e-4, 0, 60])
```

```
(1e-08, 0.0001, 0.0, 60.0)
```



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

```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
train_set = windowed_dataset(x_train, window_size=60, batch_size=100, shuffle_buffer=shuff
```

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.

```
## tu código para la red neuronal del ejercicio 6 aquí

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(60,5,1,'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),
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x*400)])
```

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.

```
## tu código para compilar la red neuronal para el ejercicio 7 aquí
model.compile(
    loss='huber',
    optimizer=tf.keras.optimizers.SGD(learning_rate=1e-5,momentum=0.9),
    metrics=["mae"]
)


history = model.fit(train_set,epochs=150)
```

```
Epoch 121/150
```

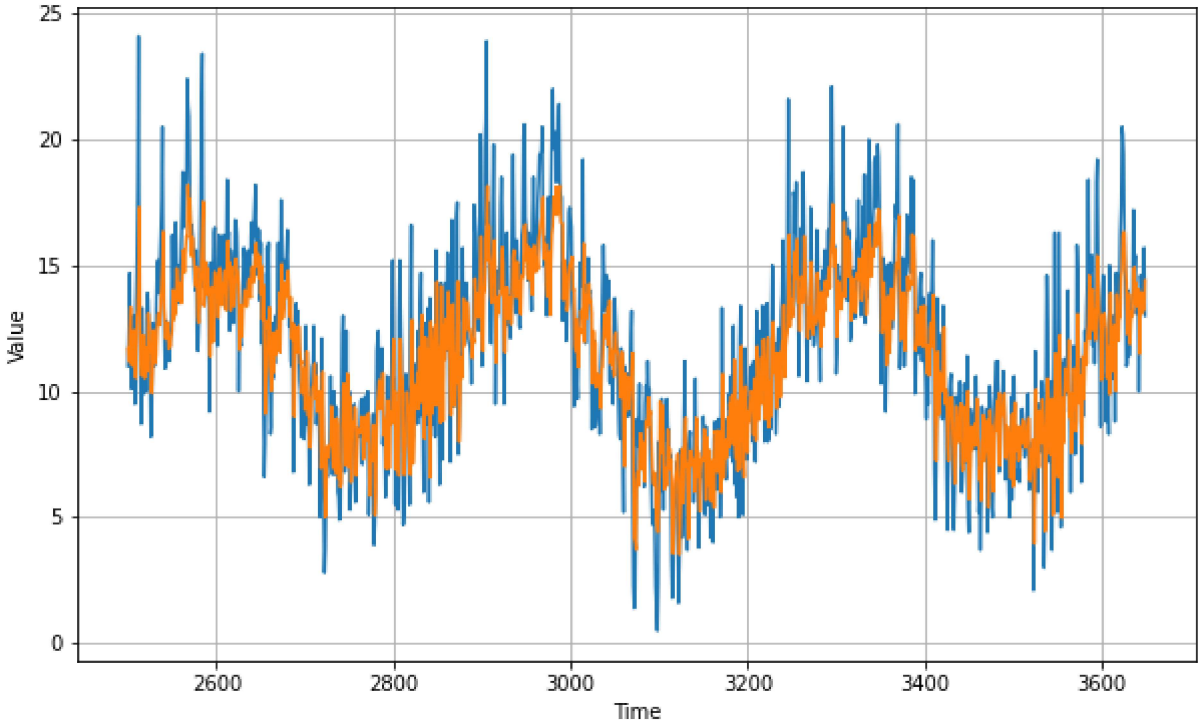
```
25/25 [==============================] - 1s 17ms/step - loss: 1.4682 - mae: 1.9068
Epoch 122/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4670 - mae: 1.9051
Epoch 123/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4770 - mae: 1.9162
Epoch 124/150
25/25 [==============================] - 1s 19ms/step - loss: 1.4669 - mae: 1.9050
Epoch 125/150
25/25 [==============================] - 1s 19ms/step - loss: 1.4708 - mae: 1.9095
Epoch 126/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4668 - mae: 1.9050
Epoch 127/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4651 - mae: 1.9029
Epoch 128/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4661 - mae: 1.9041
Epoch 129/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4694 - mae: 1.9077
Epoch 130/150
25/25 [==============================] - 1s 19ms/step - loss: 1.4756 - mae: 1.9144
Epoch 131/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4653 - mae: 1.9032
Epoch 132/150
25/25 [==============================] - 1s 19ms/step - loss: 1.4660 - mae: 1.9039
Epoch 133/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4655 - mae: 1.9035
Epoch 134/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4648 - mae: 1.9028
Epoch 135/150
25/25 [==============================] - 1s 19ms/step - loss: 1.4714 - mae: 1.9094
Epoch 136/150
25/25 [==============================] - 1s 19ms/step - loss: 1.4685 - mae: 1.9066
Epoch 137/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4673 - mae: 1.9053
Epoch 138/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4645 - mae: 1.9024
Epoch 139/150
25/25 [==============================] - 1s 19ms/step - loss: 1.4715 - mae: 1.9100
Epoch 140/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4703 - mae: 1.9085
Epoch 141/150
25/25 [==============================] - 1s 17ms/step - loss: 1.4741 - mae: 1.9128
Epoch 142/150
25/25 [==============================] - 1s 17ms/step - loss: 1.4648 - mae: 1.9025
Epoch 143/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4704 - mae: 1.9085
Epoch 144/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4703 - mae: 1.9082
Epoch 145/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4638 - mae: 1.9017
Epoch 146/150
25/25 [==============================] - 1s 31ms/step - loss: 1.4640 - mae: 1.9016
Epoch 147/150
25/25 [==============================] - 1s 27ms/step - loss: 1.4654 - mae: 1.9030
Epoch 148/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4670 - mae: 1.9050
Epoch 149/150
25/25 [==============================] - 1s 18ms/step - loss: 1.4719 - mae: 1.9104
Epoch 150/150
```

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 metodo 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 númerica, de esta forma, este modelo diseñado en esta actividad podría ser el input de un nuevo algoritmo si fuera necesario.

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

```
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, rnn_forecast)
```




```
tf.keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy()
```

```
1.8271741
```

```
print(rnn_forecast)
```

```
[11.688922 11.176365 12.432274 ... 13.273293 13.252902 14.406997]
```

8. Mostrar gráficamente los resultados.

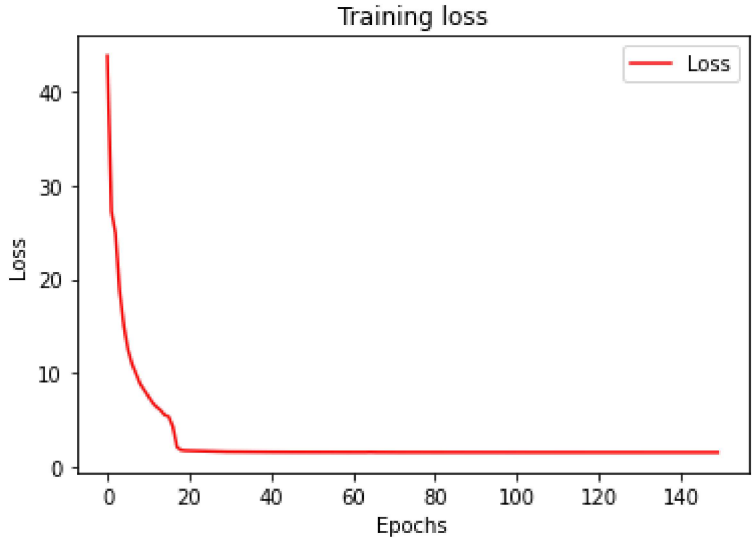
Una vez obtenido el resultado de la actividad, se procede a revisr de forma gráfica el training y validation loss a lo largo de los epochs en este nuevo entrenamiento con el learning rate optimizado

```
import matplotlib.image  as mpimg
import matplotlib.pyplot as plt

#-----------------------------------------------------------
# Recuperar una lista de resultados de la lista de datos de entrenamiento y pruebas para c
#-----------------------------------------------------------
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

```
#------------------------------------------------
# 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()
```

```
<Figure size 432x288 with 0 Axes>
```



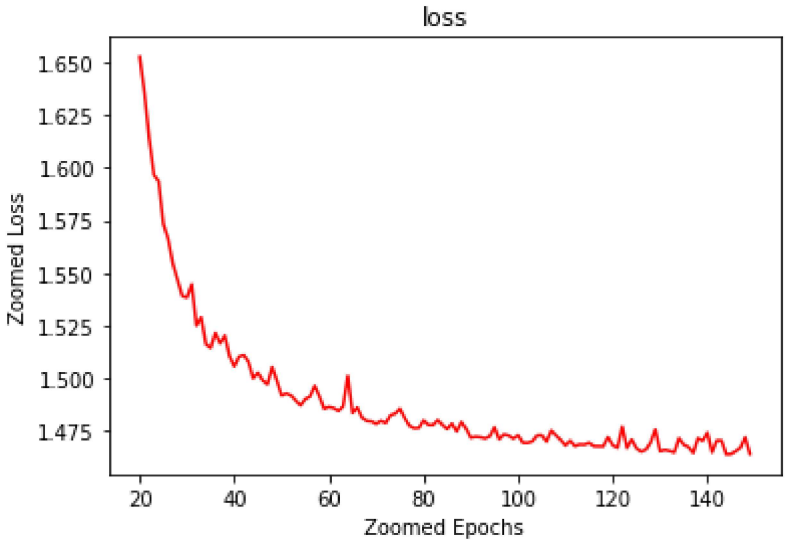

```
<Figure size 432x288 with 0 Axes>
```

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 podria parecer según el anterior plot.

```
#------------------------------------------------
```

```
# Pérdida de entrenamiento y validación por epoch con zoom
#-------------------------------------------------
zoomed_loss = loss[20:]
zoomed_epochs = range(20,150)
#
## tu código para el plot con zoom del ejercicio 8 aquí
plt.plot(zoomed_epochs, zoomed_loss, 'r')
plt.title('loss')
plt.xlabel("Zoomed Epochs")
plt.ylabel("Zoomed Loss")
#plt.legend(["Zoomed Loss"])

plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

Productos de pago de Colab - Cancelar contratos

✓ 0 s completado a las 14:11