Machine Learning
Agosto – Diciembre 2022
Dr. Luis Carlos Padierna García

Long Short Term Memory (LSTM) para predicción de la Criptomoneda Bitcoin.

Proyecto Final

Ramírez Andrade Darío, 436963, d.ramirezandrade@ugto.mx

Introducción

¿La historia de los datos cuenta? Absolutamente sí, podemos extraer bastante información y relevancia del orden secuencial de los datos.

Un ejemplo del que podemos partir es:

- "El caballo blanco corre por la pradera libre y rápido".

"Libre" nos da mucho contexto y significados que podemos interpretar.

- "Las nuevas elecciones fueron necesarias, aunque pensé que iban a ser un desastre"

¿Cómo hacemos para analizar el sentimiento del enunciado?

Podemos tener una red neuronal para cada palabra, este conjunto de redes se le denomina time-step.

Objetivo

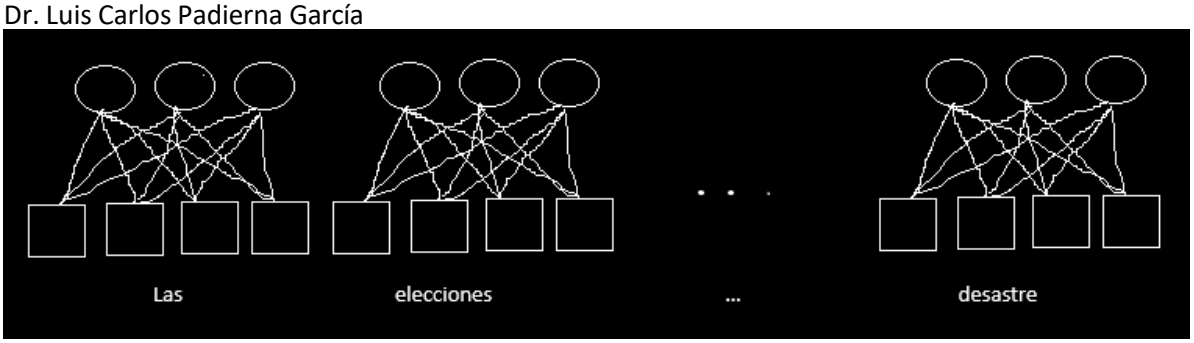
El objetivo será poder hacer una predicción del valor Close (USD) de la criptomoneda Bitcoin usando su serie temporal con los atributos Date, High, Low, Open, Close, Volume, Marketcap usando una LSTM, modelo del Deep Learning.

Justificación

Este proyecto es importante ya que ayuda a introducirnos a un campo complejo donde desde hace tiempo, en la creación del mercado bursátil se ha querido predecir valores futuros tanto como de acciones y monedas. Este proyecto ayudará a tener una mejor visión sobre este campo, de como las herramientas del Deep Learning nos pueden ayudar a predecir valores; el comprender y analizar este ámbito obtendremos un beneficio personal, científico, empresarial y profesional.

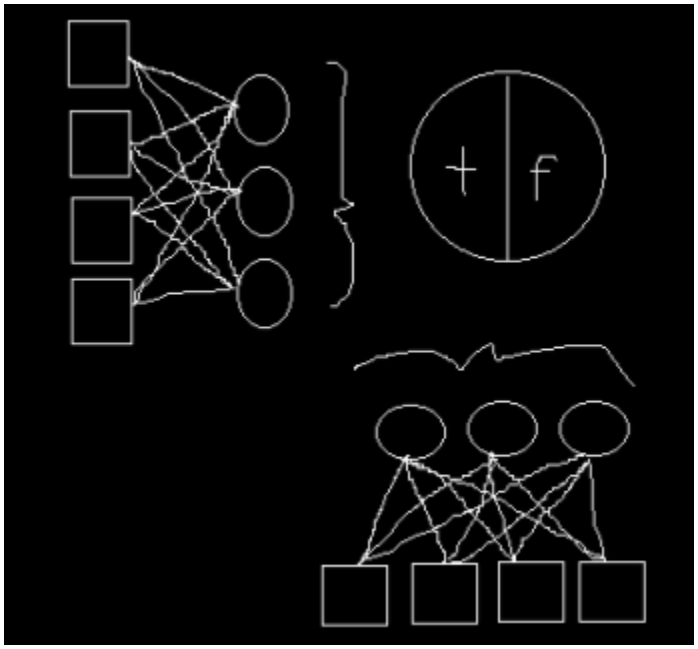
Marco Teórico

Retomando el problema de la introducción, si estas redes de alguna manera ya funcionasen y tienen toda su implementación, la palabra desastre sin todo el contexto la clasificaría como "mala", sentimiento negativo; es ahí donde entra las Redes Neuronales Recurrentes.



¿Qué podemos hacer para no tener esta pérdida de contexto? Hacemos "un vector con contexto" que incluye toda la información previa, es por eso que recuerda información del pasado.

Agregamos una capa densa el cual tendrá toda la información histórica. La suma de esta capa histórica y de la capa secuencial es una RNN cell.



En cada time-step vamos a tener una estructura similar a esta.

De manera descriptiva este sería el funcionamiento de una RNN donde en su estructura tiene varias celdas RNN y al final, dependiendo del problema puede hacer una regresión o una clasificación.

Puntos clave:

- Los pesos iniciales de la capa densa oculta es ceros.

- Normalmente se usa tanh como función de activación en las RNN cell

Se actualizan los pesos como normalmente se suele hacer para capas densas y/o CNN, usando un tipo de retro propagación (Backpropagation throug time).

El problema y solución:

- Vanishing Gradient: si las derivadas en cada paso son pequeñas (por ejemplo menor que 1), al multiplicarlas muchas veces (una por cada paso en el tiempo), el gradiente se hace cada vez más pequeño, hasta que es casi cero. Eso impide que la red aprenda dependencias lejanas.

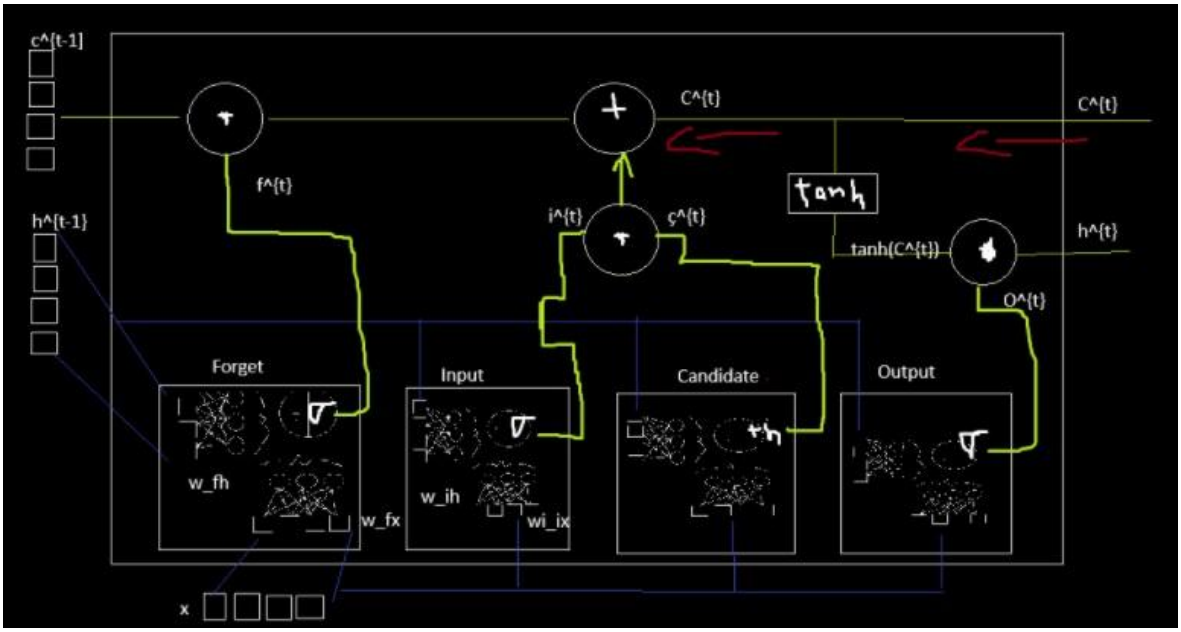
- Entra LSTM (variante de RNN), usa una conexión directa para el estado de memoria (el cell-state), que permite que el gradiente fluya sin apagarse.

Long Short Term Memory (LSTM):

Agrego un "camino", un camino fácil para el backpropagation donde además tenemos el control de Gates (que tanto conservamos o dejamos pasar del estado pasado de la celda).

El paper original de LSTM se presentó en 1997, pasaron 20 años para ser relevante, definitivamente adelantado a su época.

Para entender las LSTM, podemos decir que es una estructura con varias sub-celdas RNN dentro de una sola celda LSTM.



Puntos clave y ecuaciones:

Machine Learning
Agosto – Diciembre 2022
Dr. Luis Carlos Padierna García

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \quad (1)$$
$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \quad (2)$$
$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \quad (3)$$
$$\tilde{C}_t = \tanh(W_{Cx}x_t + W_{Ch}h_{t-1} + b_C) \quad (4)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (5)$$
$$h_t = o_t \odot \tanh(C_t) \quad (6)$$

1. Input Gate: Un valor cercano a 1 significa "dejar pasar" o "agregar mucha" de la nueva información potencial. Un valor cercano a 0 significa "casi no agregar" nada de la nueva información. Este valor i_t es una señal de cuánta atención se le debe prestar a la nueva información entrante.

2. Forget Gate: Un valor cercano a 1 para un elemento en particular significa "recordar" o "mantener" esa parte del estado de memoria anterior. Un valor cercano a 0 significa "olvidar" o "descartar" esa parte. Es la señal que controla el flujo de información hacia adelante desde el estado de celda anterior.

3. Output Gate: Un valor cercano a 1 para un elemento significa que esa parte del estado de celda actual es muy importante para la salida. Un valor cercano a 0 significa que no es tan relevante. Este valor O_t es una señal que controla cuánta información del estado de la celda se expone para convertirse en el nuevo estado oculto.

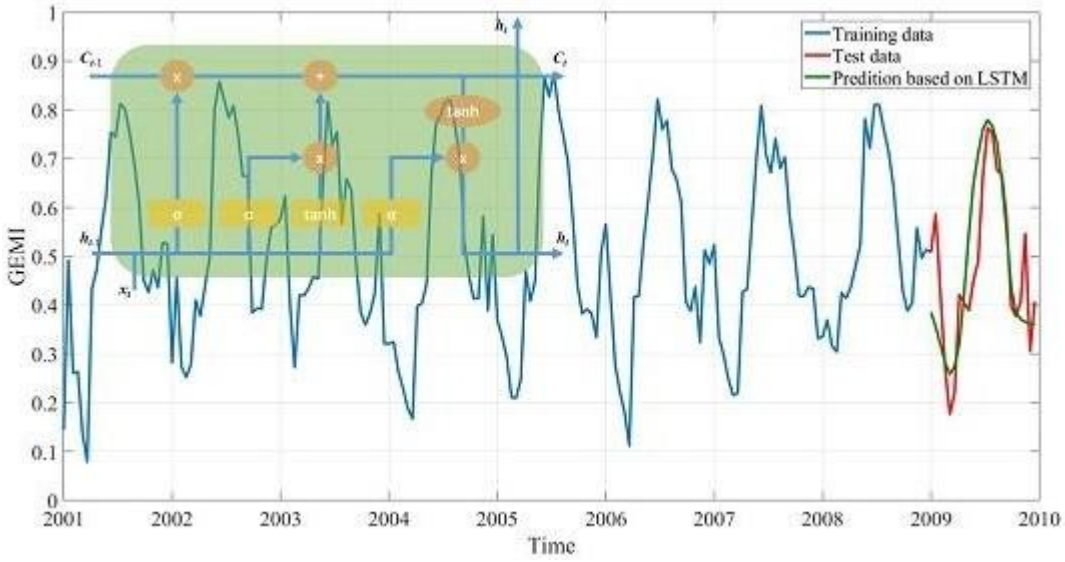
4. Los valores de Ç representan la información nueva y fresca que la red ha decidido que es potencialmente importante para el estado de memoria, ya sea para agregarla (valores positivos) o para restarla (valores negativos).

5. Cell State: La LSTM decide explícitamente que olvidar de la memoria pasada f_t ** C_t y que nueva información añadir i_t ** Ç_t para formar la memoria actual. El producto Hadamard (punto) es clave aquí, ya que actúa como un "filtro" elemento a elemento.

6. La compuerta de salida O_t decide qué partes del estado de memoria filtrado (tanh(C_t)) son relevantes para ser expuestas como la salida de esta celda en este paso de tiempo. Esto permite que la LSTM mantenga información compleja en su estado de celda C_t sin tener que exponerla toda en su salida inmediata h_t, lo que la hace muy potente para modelar secuencias.

El siguiente artículo profundiza un poco más sobre las ecuaciones y nos quedamos con la siguiente imagen proporcionada. https://www.mdpi.com/2072-4292/10/3/452

Machine Learning
Agosto – Diciembre 2022
Dr. Luis Carlos Padierna García



Ventanas y tipos de series temporales:

Dependiendo de nuestro problema y serie temporal, es que podemos hacer diferentes tipos de "ventanas", como ya visto, son los time-step.

Entrada	Salida
Univariado	Unistep
Univariado	Multistep
Multivariado	Unistep
Multivariado	Multistep

Diagrama de bloques

Como primer componente se hizo la Implementación del dataset. El objetivo será predecir los valores Close de la Criptomoneda Bitcoin. Donde usando diferentes atributos (Multivariado) haremos una predicción de una sola clase y un instante de tiempo, en este caso "Close" en dólares .

Machine Learning
Agosto – Diciembre 2022
Dr. Luis Carlos Padierna García

1. Cargar el dataset obtenido en Keggle https://www.kaggle.com/datasets/sudalairajkumar/cryptocurrencypricehistory?select=coin_Bitcoin.csv

```
[ ]
1 import pandas as pd # manejo de dataset
2 import matplotlib.pyplot as plt # graficar
3 import seaborn as sns #graficar
4
5 from google.colab import drive
6 drive.mount('/content/drive', force_remount=True) #permitimos acceso
7
8 #leemos el csv
9 df = pd.read_csv("/content/drive/MyDrive/Proyecto/coin_Bitcoin.csv")
10
11 df
```

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

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
0	1	Bitcoin	BTC	2013-04-29 23:59:59	147.488007	134.000000	134.444000	144.539993	0.000000e+00	1.603769e+09
1	2	Bitcoin	BTC	2013-04-30 23:59:59	146.929993	134.050003	144.000000	139.000000	0.000000e+00	1.542813e+09
2	3	Bitcoin	BTC	2013-05-01 23:59:59	139.889999	107.720001	139.000000	116.989998	0.000000e+00	1.298955e+09
3	4	Bitcoin	BTC	2013-05-02 23:59:59	125.599998	92.281898	116.379997	105.209999	0.000000e+00	1.168517e+09
4	5	Bitcoin	BTC	2013-05-03 23:59:59	108.127998	79.099998	106.250000	97.750000	0.000000e+00	1.085995e+09
...	...	...	...	...	...	...	...	...	...	...

Después se pasó a un preprocesamiento en el que se eliminó columnas innecesarias y se modificó el índice.

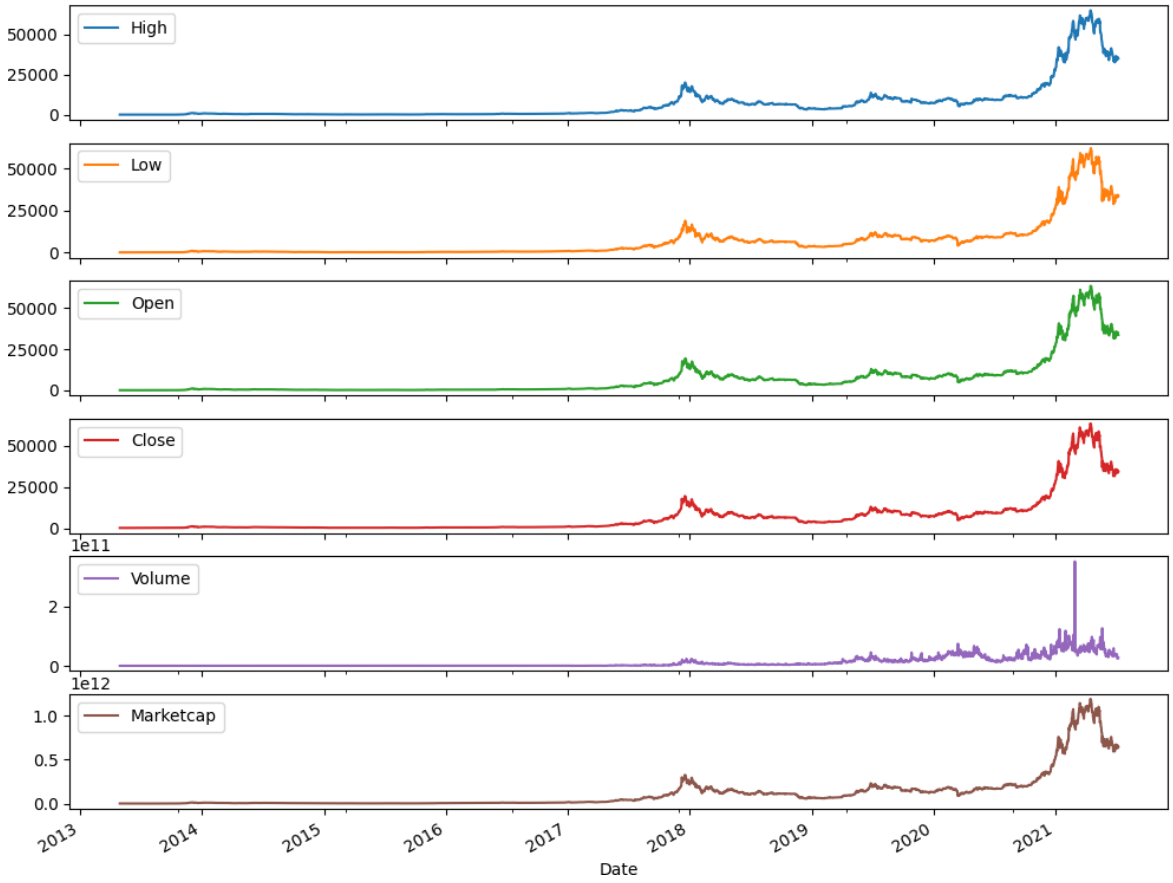
```
1 # eliminamos columnas no necesarias
2 df = df.drop(columns=["SNo", "Name", "Symbol"])
3
4 # convierto date a formato valido de pd datetime sino no funciona las fechas IMPORTANTE
5 df["Date"] = pd.to_datetime(df["Date"])
6
7 df
```

	Date	High	Low	Open	Close	Volume	Marketcap
0	2013-04-29 23:59:59	147.488007	134.000000	134.444000	144.539993	0.000000e+00	1.603769e+09
1	2013-04-30 23:59:59	146.929993	134.050003	144.000000	139.000000	0.000000e+00	1.542813e+09
2	2013-05-01 23:59:59	139.889999	107.720001	139.000000	116.989998	0.000000e+00	1.298955e+09
3	2013-05-02 23:59:59	125.599998	92.281898	116.379997	105.209999	0.000000e+00	1.168517e+09
4	2013-05-03 23:59:59	108.127998	79.099998	106.250000	97.750000	0.000000e+00	1.085995e+09
...	...	...	...	...	...	...	...

Se obtuvo las gráficas con:

```
1 date_time = df.pop("Date") #convierto date_time a los índices
2
3 #Evolución caracteristicas a lo largo del tiempo
4 plot_cols = ["High", "Low", "Open", "Close", "Volume", "Marketcap"]
5 plot_features = df[plot_cols]
6 plot_features.index = date_time
7 _ = plot_features.plot(subplots=True, figsize=(12, 10))
8 plt.show()
9
```

Podemos observar de mejor manera las series de tiempo que contamos.



Hacemos una partición 70/20/10 para entrenamiento, validación y prueba.
IMPORTANTE: Respetar las particiones cronológicamente.

```python
#genero particiones
n = len(df) #cantidad datos en df
train_data = df[0:int(n*0.7)] #70
val_data = df[int(n*0.7):int(n*0.9)] #20
test_data = df[int(n*0.9):] #10

```

Implemento sklearn para usar un normalizador, ya que como los modelos ML se requiere normalizar los datos y evitar sesgos y sensibilidades.

En este punto, varios programadores usan StandardScaler o MinMaxScaler; pro prueba y error MinMax obtenía un val_loss menor. (si tuve en consideración la función de activación tanh).

```python
from sklearn.preprocessing import MinMaxScaler #StandardScaler

# creo el scaler usando solo los datos de entrenamiento
scaler = MinMaxScaler(feature_range=(-1, 1)) #StandardScaler()

train_scaled = scaler.fit_transform(train_data)
# scaler para validacion y prueba
val_scaled = scaler.transform(val_data)
test_scaled = scaler.transform(test_data)

# convertimos a DataFrames
train_scaled = pd.DataFrame(train_scaled, columns=train_data.columns)
val_scaled = pd.DataFrame(val_scaled, columns=val_data.columns)
test_scaled = pd.DataFrame(test_scaled, columns=test_data.columns)


print(f"{df.shape[0]}\n") #No. datos y columna siempre 6 [1]
print(f"{train_data.shape[0]}") #No. datos entrenamiento y columna siempre 6 [1]
print(f"{val_data.shape[0]}") #No. datos validacion y columna siempre 6 [1]
print(f"{test_data.shape[0]}") #No. datos test y columna siempre 6 [1]

```

Procedemos a hacer la ventana temporal antes vista en la metodología:

```python
#VENTANA TEMPORAL
#past = 30     # Usamos 30 días anteriores (ultimos 30 dias para predecir)
#past = 15     # Usamos 15 días anteriores (ultimos 30 dias para predecir)
past = 7     # Usamos 7 días anteriores (ultimos 30 dias para predecir)
future = 1    # Queremos predecir 1 día en el futuro (target)

#HIPERPARAMETROS
learning_rate = 0.001 #para el optimizador
batch_size = 256 #cuantas muestras se usan en cada paso del entrenamiento
                 # normalmente potencias de 2
epochs = 20      #epocas que recorre el dataset
step = 1         #sampling rate: para construir las secuencias sin saltos

import keras

x_train = train_scaled.values  # convertimos a Array con valores escalados
y_train = train_scaled.iloc[past + future :][["Close"]]
#el target será Close, past + future son los indices para alinear la predicción
#ej. como aquí past=30 y future=1, para la secuencia usa datos [0:29]
#como input, la etiqueta será el valor en la posición 31
```

Usamos el método ya implementado por Keras.

```python
#https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/timeseries_dataset_from_array
dataset_train = keras.preprocessing.timeseries_dataset_from_array(
    x_train,
    y_train,
    sequence_length= past,
    sequence_stride= 1, #se desliza con superopocisión
    sampling_rate= step, #se salta (dependiendo) los datos de la ventana
    batch_size= batch_size,
)
```

Defino índices para que no haya desfaces y los arrays tengan correctos índices y steps.

```python
#defino indice maximo, que no pase del límite del conjunto de validación
x_end = len(val_scaled) - past - future

#extraigo las filas desde el inicio hasta x_end de val_scaled (ya escalado)
x_val = val_scaled.iloc[:x_end].values

#el primer índice de y_val es past + future para alinear con las secuencias de entrada.
y_val = val_scaled.iloc[past + future :][["Close"]]

# verificación
#dataset_val tiene bach size y las columnas
for batch in dataset_val.take(1):
    inputs, targets = batch

print("Input shape:", inputs.numpy().shape)
print("Target shape:", targets.numpy().shape)
print(f"{x_end}")
print(f"{dataset_val}")
```

Machine Learning
Agosto – Diciembre 2022
Dr. Luis Carlos Padierna García
Constructor:

¿Dónde quedan todas las ecuaciones antes vistas? Todas las ecuaciones de la LSTM, como la input gate, forget gate, output gate etc., están embebidas dentro de la capa keras.layers.LSTM; Keras las implementa internamente cuando definimos esa capa. Nosotros solo preparamos la entrada como una secuencia temporal (en mi caso la variable past=7), y luego esa capa se encarga de aplicar todos los pasos recurrentes. Finalmente, usamos una capa densa para proyectar el estado oculto final a una predicción del dia futuro.

A la salida final, me da el estado oculto final h_t, con dimensión (batch_size, 32). Ese 32 es cada célula LSTM con 32 unidades. Y cada unidad LSTM tiene sus propias puertas.

```python
#capa entrada, 30 muestras de 6 features(columnas)
inputs = keras.layers.Input(shape=(inputs.shape[1], inputs.shape[2]))
#añado una capa LSTM con 32 neuronas
#devuelve el último estado oculto por cada muestra de dimensión: (batch_size, 32)
#Keras retorna solo el output del último timestep
lstm_out = keras.layers.LSTM(32)(inputs)
#ultima capa densa full conected con una neurona de salida para la regresión
outputs = keras.layers.Dense(1)(lstm_out)

#creo el modelo
model = keras.Model(inputs=inputs, outputs=outputs)
#el optimizador con adam y la tasa de aprendizaje
#función de pérdida mse
model.compile(optimizer=keras.optimizers.Adam(learning_rate=learning_rate), loss="mse")
#resumen arq
model.summary()

print(f"{inputs.shape[1]}")
print(f"{inputs.shape[2]}")
```

Resumen del modelo:

Model: "functional_23"

Layer (type)	Output Shape	Param #
input_layer_23 (InputLayer)	(None, 30, 6)	0
lstm_23 (LSTM)	(None, 32)	4,992
dense_23 (Dense)	(None, 1)	33

Total params: 5,025 (19.63 KB)
Trainable params: 5,025 (19.63 KB)
Non-trainable params: 0 (0.00 B)

```
30
6
```

Finalmente entrenamos con la ayuda de las ventanas.

```python
#inicio entrenamiento con validacion para medirlo
#history para guardar el historial de pérdidas y métricas por época
history = model.fit(
    dataset_train,
    epochs=epochs,
    validation_data=dataset_val,
    #callbacks=[es_callback, modelckpt_callback],
)
```

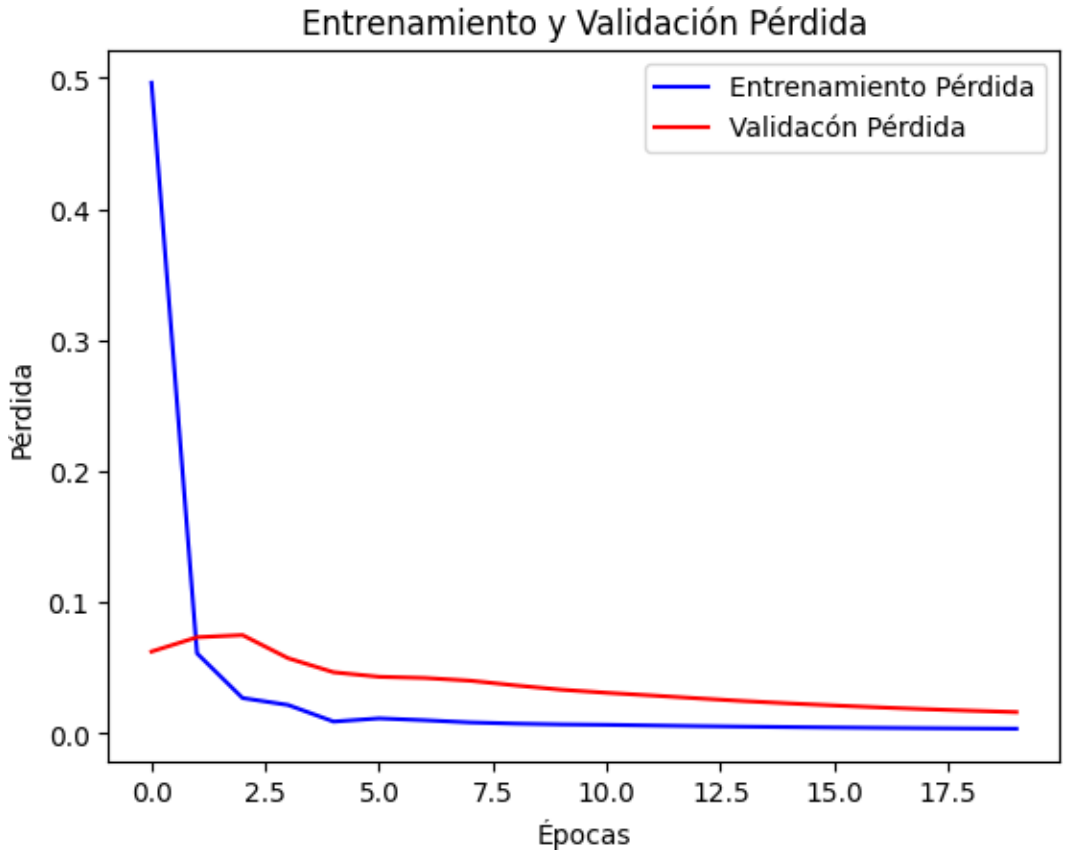
Observamos las últimas épocas (relevantes):

```
Epoch 15/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - loss: 0.0022 - val_loss: 0.0230
Epoch 16/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 104ms/step - loss: 0.0021 - val_loss: 0.0214
Epoch 17/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - loss: 0.0019 - val_loss: 0.0199
Epoch 18/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 100ms/step - loss: 0.0018 - val_loss: 0.0186
Epoch 19/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - loss: 0.0017 - val_loss: 0.0174
Epoch 20/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - loss: 0.0016 - val_loss: 0.0163
```

Finalmente hacemos evaluaciones.

```python
#https://www.ibm.com/docs/es/masv-and-l/maximo-vi/cd?topic=models-graph-training
def visualize_loss(history, title):
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, "b", label="Entrenamiento Pérdida")
    plt.plot(epochs, val_loss, "r", label="Validacón Pérdida")
    plt.title(title)
    plt.xlabel("Épocas")
    plt.ylabel("Pérdida")
    plt.legend()
    plt.show()


visualize_loss(history, "Entrenamiento y Validación Pérdida")
```



El objetivo es que el modelo converja al final del entrenamiento con un nivel de pérdida bajo.

Para hacer las gráficas debemos des escalar para que tengan los mismos tamaños.

```python
import numpy as np
import matplotlib.pyplot as plt

#invierto el escalado de valores escalados que predice
def invert_scaling(scaled_close):
    #dummy para lleno de ceros con el mismo número de columnas que
    #tenía el dataset original (train_scaled.shape[1])
    dummy = np.zeros((len(scaled_close), train_scaled.shape[1]))
    #coloco los valores predichos en la columna close
    dummy[:, df.columns.get_loc("Close")] = scaled_close[:, 0]
    #hago transformación inversa y devuelve Close ya descalada
    return scaler.inverse_transform(dummy)[:, df.columns.get_loc("Close")]

# Preparar dataset_test igual que dataset_val
x_test_full = test_scaled.values #convierto a array
#defino el punto hasta donde tomar datos para crear las secuencias
x_end_test = len(test_scaled) - past - future
#extraigo el rango válido
x_test = test_scaled.iloc[:x_end_test].values
#preparo las salidas alineando con las secuencias generadas
#(sáltandose los primeros past + future para que coincidan con x)
y_test = test_scaled.iloc[past + future :][["Close"]]
```

```python
y_test = test_scaled.iloc[past + future :][["Close"]]

#creo ventana similar a la pasada pero para test
dataset_test = keras.preprocessing.timeseries_dataset_from_array(
    x_test,
    y_test,
    sequence_length=past,
    sampling_rate=step,
    batch_size=batch_size,
)

#hago las predicciones del modelo sobre el dataset de validación
pred_val = model.predict(dataset_val)
#son los valores no escalados aún extraídos del dataset.
real_val = np.concatenate([y for x, y in dataset_val], axis=0)

#lo mismo pero con test
pred_test = model.predict(dataset_test)
real_test = np.concatenate([y for x, y in dataset_test], axis=0)

```

```
#invierto el escalado
pred_val_inv = invert_scaling(pred_val)
real_val_inv = invert_scaling(real_val)
pred_test_inv = invert_scaling(pred_test)
real_test_inv = invert_scaling(real_test)

#extraigo los índices de fechas correspondientes para poder graficar validación y test
#alineo para que tengan el mismo largo que los vectores real_val_inv y real_test_inv
val_indices = val_data.index[past + future : past + future + len(real_val_inv)]
test_indices = test_data.index[past + future : past + future + len(real_test_inv)]
```

Una vez hecho esto podemos hacer las gráficas y obtener métricas.

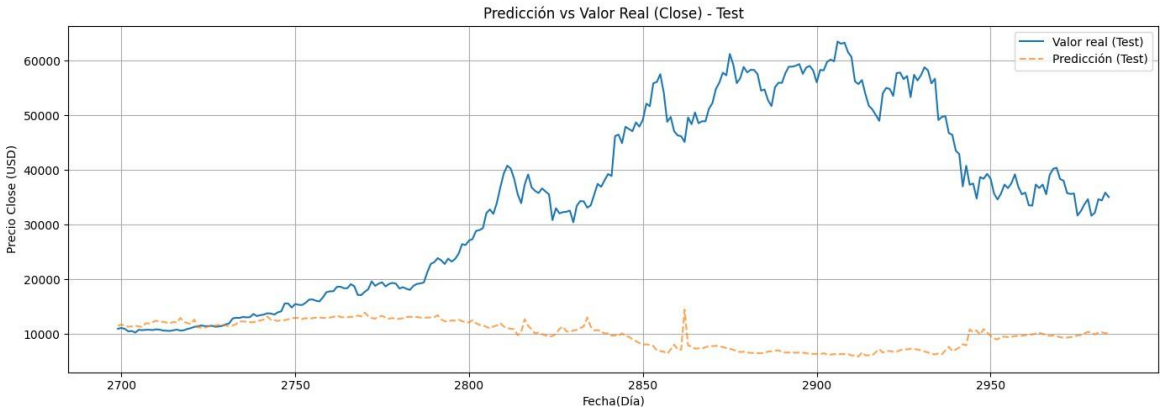
```
# grafico resultados val y test
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 10), sharex=False)

#gráfica 1 validación (predicción vs real)
ax1.plot(val_indices, real_val_inv, label="Valor real (Validación)")
ax1.plot(val_indices, pred_val_inv, label="Predicción (Validación)", alpha=0.7, linestyle='--')
ax1.set_title("Predicción vs Valor Real (Close) - Validación")
ax1.set_xlabel("Fecha(Día)")
ax1.set_ylabel("Precio Close (USD)")
ax1.legend()
ax1.grid(True)

#gráfica 2 test (predicción vs real)
ax2.plot(test_indices, real_test_inv, label="Valor real (Test)")
ax2.plot(test_indices, pred_test_inv, label="Predicción (Test)", alpha=0.7, linestyle='--')
ax2.set_title("Predicción vs Valor Real (Close) - Test")
ax2.set_xlabel("Fecha(Día)")
ax2.set_ylabel("Precio Close (USD)")
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()
```

Machine Learning
Agosto – Diciembre 2022
Dr. Luis Carlos Padierna García



La validación fue buena, pero el test falla. Pueden ocurrir varias cosas:

- El modelo aprendió muy bien el set de entrenamiento/validación, pero no generaliza bien al test. Esto lo podemos mantener ya que ajustando hiperparametros no cambia demasiado, de hecho, todas las actualizaciones dan bien la validación, pero no el testeo.

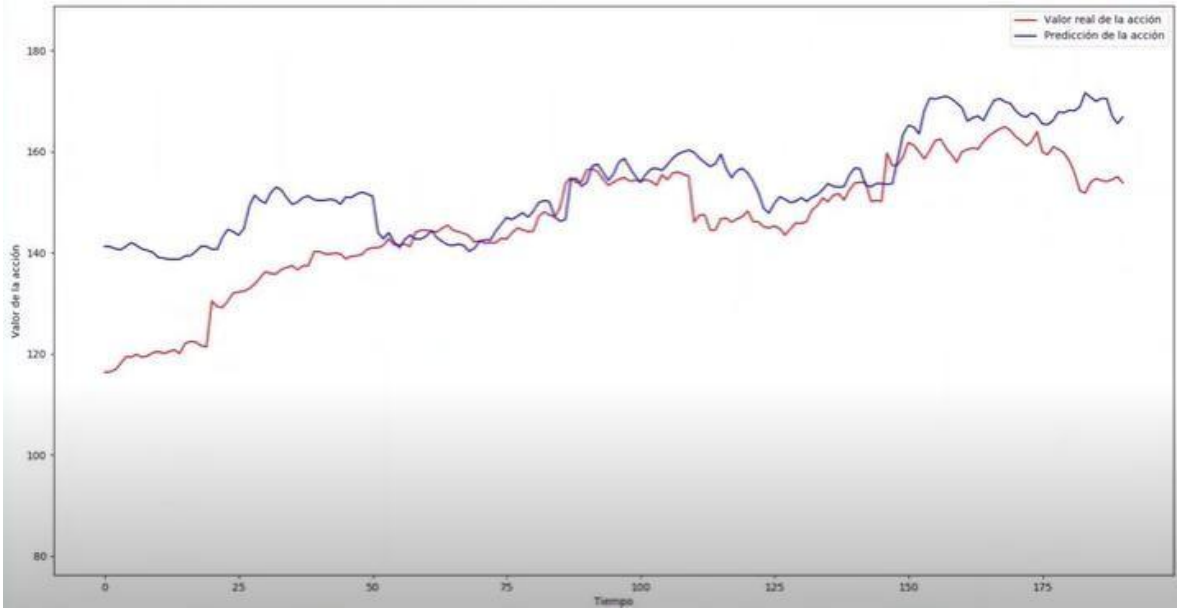
Igual podemos dar alguna explicación, pero será más a detalle adelante.

- Error en el escalado o des escalado. Esto también lo podemos descartar ya que viendo otras fuentes parecidas, donde analizan igual "acciones" mediante LSTM vemos gráficas de validación y valor real con una naturaleza similar.

- Predicciones lejanas (future). En este caso usamos el mínimo (1), justo para evitar complicaciones en cuestión de predecir más instantes de tiempo.

Predicción de acciones en la bolsa con PYTHON (tutorial redes LSTM) por Codificando Bits: https://www.youtube.com/watch?v=3kXj6VgxbP8 Min. 4:00:

Machine Learning
Agosto – Diciembre 2022
Dr. Luis Carlos Padierna García



Mean Absolute Error (MAE)

Usamos métricas cuantitativas, una común para regresión es MAE que mide el error promedio en valor absoluto.

```
1 from sklearn.metrics import mean_absolute_error
2
3 mae_val = mean_absolute_error(real_val_inv, pred_val_inv)
4 mae_test = mean_absolute_error(real_test_inv, pred_test_inv)
5
6 print(f"MAE Validación: {mae_val:.2f}")
7 print(f"MAE Test: {mae_test:.2f}")
```

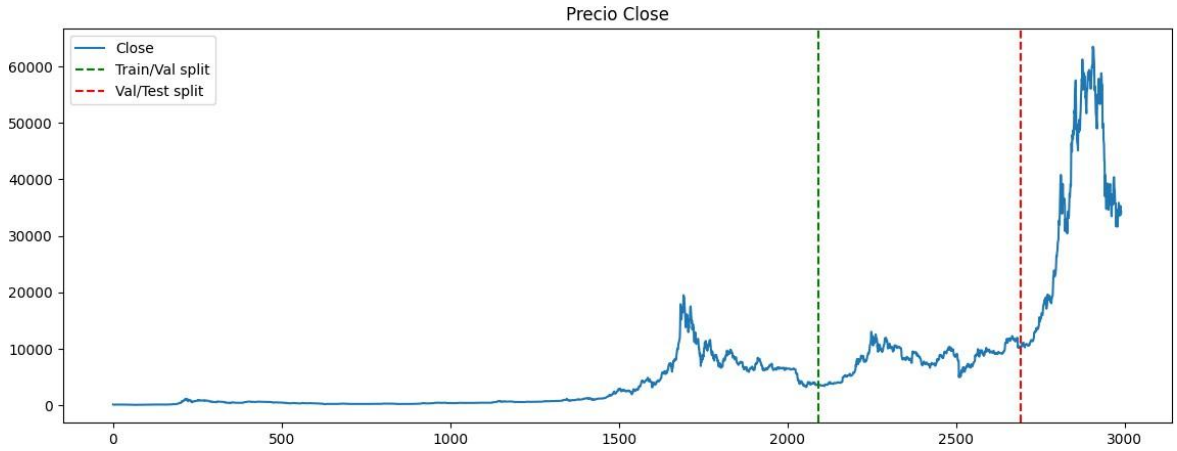
```
MAE Validación: 911.84
MAE Test: 25187.40
```

En mejor de los casos tuvimos un MAE de Validación de 694.72 y MAE en Test de 22188.24

Esto indica que entre el error de validación y de test dice que el modelo aprendió bien en entrenamiento/validación, pero cuando ve datos nuevos (test) falla mucho.

Esto puede deberse a que por ejemplo y de hecho los precios en testeo son mucho más altos que en validación (por eso el MAE es tan alto). Por lo que asumimos que tal vez hubo una subida fuerte en el mercado y el modelo nunca vio o "vivió" algo parecido. Que de hecho fue cuando fue más el "BOOM" de la cripto.

Machine Learning
Agosto – Diciembre 2022
Dr. Luis Carlos Padierna García



Para ver un poco mejor: El Error Porcentual Absoluto Medio (MAPE)

Es una métrica para evaluar la precisión de los modelos de regresión. Se calcula como el promedio de los porcentajes absolutos de diferencia entre los valores previstos y los valores reales (que ya lo hicimos arriba), expresados como un porcentaje. El MAPE es útil para comprender la desviación promedio de las predicciones en relación con los valores reales.

```
[ ]  1 from sklearn.metrics import mean_absolute_percentage_error
     2
     3 mape_val = mean_absolute_percentage_error(real_val_inv, pred_val_inv) * 100
     4 mape_test = mean_absolute_percentage_error(real_test_inv, pred_test_inv) * 100
     5
     6 print(f"MAPE Validación: {mape_val:.2f}%")
     7 print(f"MAPE Test: {mape_test:.2f}%")
     8

MAPE Validación: 12.25%
MAPE Test: 58.30%
```

Como el anterior, el mejor MAPE fueron de 7.81% para validación y para MAPE Test de 51.80%

Conclusiones:

La LSTM aprendió bien las secuencias de entrenamiento/validación.

Los errores en test son casi del 50%, similar a lo que se nos comentó en su día en la clase, ya que los modelos donde se predicen cripto, acciones, el mercado bursátil en general es algo muy completo; ya que tiene tantos factores y métricas que se pueden agregar, los cuales no tenemos o se encuentra alguna relevancia o significado, como por ejemplo lo que diga algún presidente, guerras, etc. cosas que claramente no están presentes en el modelo y su implementación seria por no decir imposible.

Sin embargo, el propósito en general del modelo fue implementado correctamente, dando a explicar las métricas, sus conceptos clave como las ventanas, etc.

En consideración, mi principal idea del proyecto era incluir un API el cual día con día se actualicen, carguen y entrene el modelo ya que así tendrá siempre en constante actualización la capa histórica, claro, seguirá siendo un modelo falible ante las cuestiones ya mencionadas(política), pero así evitaríamos el error de que los datos de entrada son cosas que no ha "vivido" ya que al final esa es la ventaja de las LSTM.

Se puede obtener todos los códigos, datasets, libretas en el GitHub:

https://github.com/darioIF/Machine-Learning-Enero-Junio/blob/main/Proyecto_LSTM_Dar%C3%ADo_RA.ipynb

Referencias

- Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., & Gao, R. X. (2018). Deep learning and its applications to machine health monitoring. *Remote Sensing*, *10*(3), 452. https://doi.org/10.3390/rs10030452
- Pepe Cantoral. (2021, junio 3). *Qué es una LSTM | Redes neuronales recurrentes* [Video]. YouTube. https://www.youtube.com/watch?v=f6PaCo-NfJA
- Pepe Cantoral. (2021, junio 3). *Cómo funcionan las LSTM paso a paso | Parte práctica* [Video]. YouTube. https://www.youtube.com/watch?v=x6E44DDWg5Q
- Codificando Bits. (2017, diciembre 1). *Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM)* [Video]. YouTube. https://www.youtube.com/watch?v=3kXj6VgxbP8
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.
- TensorFlow. (n.d.). *tf.keras.preprocessing.timeseries_dataset_from_array*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/timeseries_dataset_from_array
- Keras. (n.d.). *Time series forecasting using weather data*. Keras. https://keras.io/examples/timeseries/timeseries_weather_forecasting/
- IBM. (n.d.). *Graph training models*. IBM Documentation. https://www.ibm.com/docs/es/masv-and-l/maximo-vi/cd?topic=models-graph-training
- OpenAI. (2025). *ChatGPT (June 2025 version) [Large language model]*. https://chat.openai.com/ Utilizado para generar imágenes, ecuaciones y consultas.
- Kumar, S. (2018). *Cryptocurrency Historical Market Price Data* [Dataset]. Kaggle. https://www.kaggle.com/datasets/sudalairajkumar/cryptocurrencypricehistory