Implementación de una Red Neuronal Recurrente de tipo Long-short Term Memory para predecir el precio de una acción en la Bolsa Mexicana de Valores de enero 2020 a abril 2021

Reynosa, Tamaulipas. 26 de mayo del 2021

1. **Introducción:**

En el presente proyecto se implementa una Red Neuronal Recurrente de tipo Long-short Term Memory para predecir el precio de una acción de la bolsa mexicana de Valores, Cemex, usando datos del 1 de enero del 2020 al 20 de abril del 2021 y se compara con una Red Neuronal Recurrente simple.

**Long short-term memory neural networks (LSTM)**

Se emplea la arquitectura de las redes neuronales recurrentes (RNN) propuesta por Sepp Hochreiter y Jurgen Schmidhuber que usa retroalimentación de iteraciones pasadas, lo cual guarda tanto en su memoria de largo plazo como en su memoria de corto plazo. (Sepp Hochreiter, 1997)

Consiste en células llamadas LSTM que reemplazan a las neuronas en las capas ocultas de la RNN, las cuales tienen un estado dado por la información de iteraciones anteriores.

Cada iteración cambia el estado de la célula, la cual tiene puertas de entrada, de olvido y de salida. Cada celda de memoria tiene tres capas sigmoides y una capa tanh. La figura 1 contiene su estructura. (Jiayu Qiu, 2020)

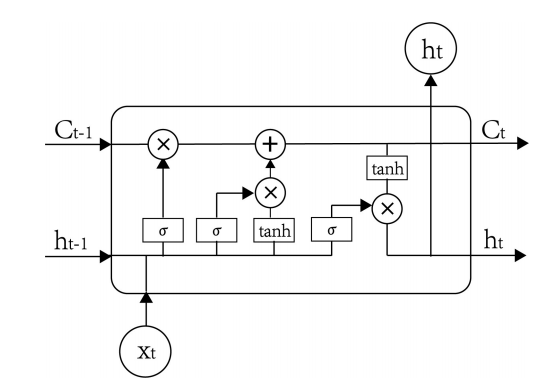


Figura 1

No existe un método para seleccionar el numero de capas escondidas o celdas que tendrá la red neuronal (Wei Bao, 2017) pero en el presente trabajo se usa una capa escondida con cuatro celdas.

Se va a comparar con una red neuronal recurrente simple con función de activación tangente y optimizador rmsprop, que funciona tomando como datos de entrada los datos de salida de la iteración anterior, como se muestra en la figura 2.

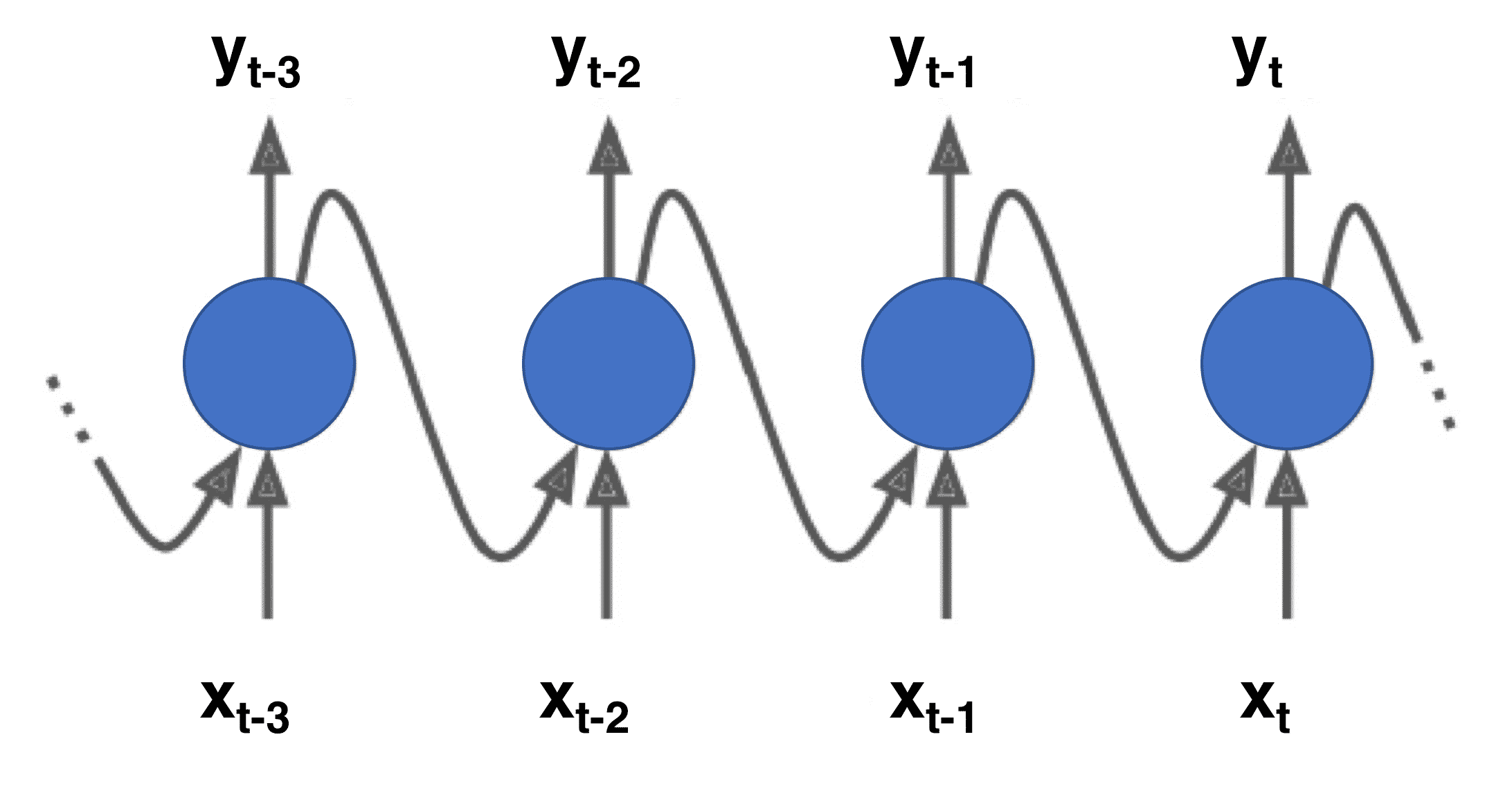


Figura 2

1. **Metodología:**

El presente trabajo se basa en una guía para predecir el número de pasajeros en vuelos (Brownlee, 2016) adaptada a la bolsa mexicana de valores. Se probaron distintos números de neuronas, capas escondidas, epochs y batch, y se decidió por 64 neuronas, 1 capa, 500 epochs y batch size de 10 para ambos modelos, pues se encontró que daban los mejores resultados.

1. **Obtención de Datos**

Los datos se obtienen de la página web de Yahoo Finance (Yahoo Finance, 2021) en formato csv. Se emplean los datos del 1 de enero del 2020 al 20 de abril. Se dividen en 80% para entrenamiento y 20% para prueba de efectividad.

1. **Importar librerías necesarias**

import numpy

import matplotlib.pyplot as plt

from pandas import read\_csv

import math

from keras.models import Sequential

from keras.layers import Dense, SimpleRNN

from keras.layers import LSTM

from keras.utils import plot\_model

from sklearn.preprocessing import MinMaxScaler

from sklearn import metrics

1. **Función para convertir el array de valores en una matriz**

def create\_dataset(dataset, look\_back=1):

    dataX, dataY = [], []

    for i in range(len(dataset)-look\_back-1):

        a = dataset[i:(i+look\_back), 0]

        dataX.append(a)

        dataY.append(dataset[i + look\_back, 0])

    return numpy.array(dataX), numpy.array(dataY)

1. **Se inicia la semilla y se importan los datos**

numpy.random.seed(7)

dataframe = read\_csv('Cemex.csv', usecols=[1], engine='python')

dataset = dataframe.values

dataset = dataset.astype('float32')

1. **Se normaliza el dataset**

scaler = MinMaxScaler(feature\_range=(0, 1))

dataset = scaler.fit\_transform(dataset)

1. **Se divide el dataset entre datos de entrenamiento y de prueba**

train\_size = int(len(dataset) \* 0.8)

test\_size = len(dataset) - train\_size

train, test = dataset[0:train\_size,:], dataset[train\_size:len(dataset),:]

1. **Se le da la forma necesaria para el modelo**

look\_back = 1

trainX, trainY = create\_dataset(train, look\_back)

testX, testY = create\_dataset(test, look\_back)

trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

1. **Se crea y se ajusta la red LSTM, se crea la imagen del modelo.**

model = Sequential()

model.add(LSTM(64, input\_shape=(1, look\_back)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(trainX, trainY, epochs=500, batch\_size=10, verbose=0)

plot\_model(model,to\_file="model.png",show\_shapes=True,show\_layer\_names=True,dpi=96)

1. **Se hacen las predicciones**

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

1. **Se invierten las predicciones**

trainPredict = scaler.inverse\_transform(trainPredict)

trainY = scaler.inverse\_transform([trainY])

testPredict = scaler.inverse\_transform(testPredict)

testY = scaler.inverse\_transform([testY])

1. **Se calcula MSE y R2**

trainScore = math.sqrt(metrics.mean\_squared\_error(trainY[0], trainPredict[:,0]))

print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(metrics.mean\_squared\_error(testY[0], testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))

r2\_train = metrics.r2\_score(trainY[0], trainPredict[:,0])

print("Train Score: {} R2".format(r2\_train))

r2\_test = metrics.r2\_score(testY[0], testPredict[:,0])

print("Test Score: {} R2".format(r2\_test))

1. **Se acomodan las predicciones de entrenamiento para graficar**

trainPredictPlot = numpy.empty\_like(dataset)

trainPredictPlot[:, :] = numpy.nan

trainPredictPlot[look\_back:len(trainPredict)+look\_back, :] = trainPredict

1. **Se acomodan las predicciones de prueba para graficar**

testPredictPlot = numpy.empty\_like(dataset)

testPredictPlot[:, :] = numpy.nan

testPredictPlot[len(trainPredict)+(look\_back\*2)+1:len(dataset)-1, :] = testPredict

1. **Se grafican los datos originales con las predicciones**

plt.plot(scaler.inverse\_transform(dataset))

plt.plot(trainPredictPlot)

plt.plot(testPredictPlot)

plt.show()

1. **Resultados**

Se muestra el diagrama del modelo LSTM en la figura 3 y la del modelo simple RNN en la figura 4.

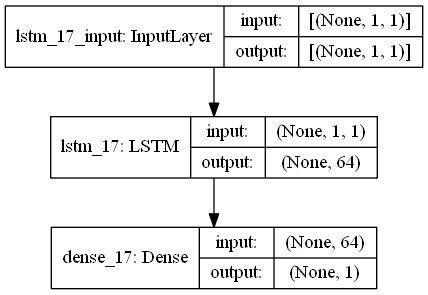


Figura 3

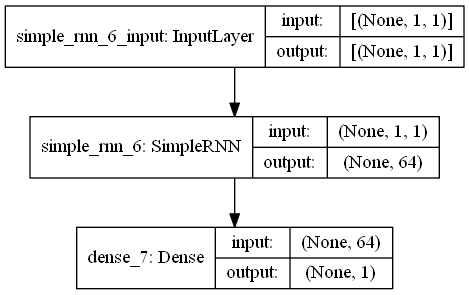


Figura 4

Los resultados del modelo LSTM se muestran en la figura 5

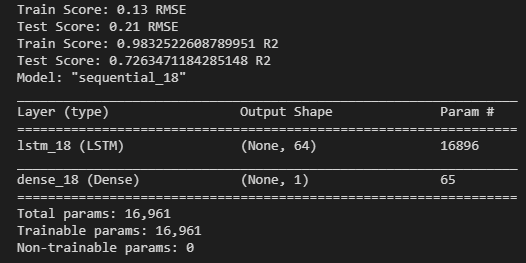


Figura 5

El grafico del modelo LSTM se muestra en la figura 6. La línea azul son los precios de la acción de Cemex, en naranja se muestran las predicciones de entrenamiento y en verde las predicciones de prueba.

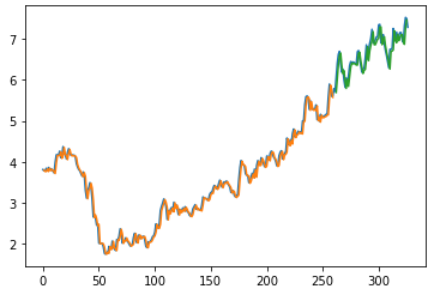


Figura 6

Los resultados del modelo RNN simple se muestran en la figura 7

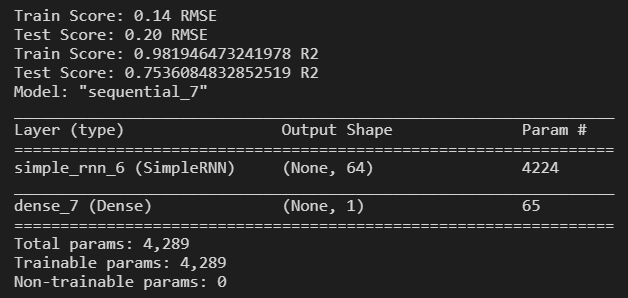


Figura 7

El grafico del modelo RNN simple se muestra en la figura 8. La línea azul son los precios de la acción de Cemex, en naranja se muestran las predicciones de entrenamiento y en verde las predicciones de prueba.

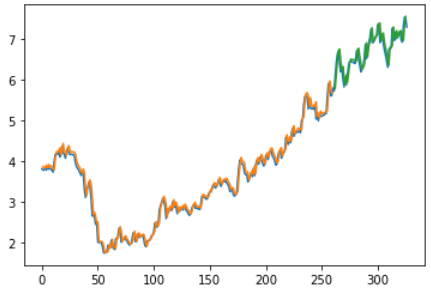


Figura 8

1. **Conclusiones:**

Ambos modelos se ajustan razonablemente, el LSTM teniendo una R2 de 0.72 en los datos de prueba y el RNN simple teniendo una R2 de 0.75 en los datos de prueba (El R2 indica que tanto se ajustan las predicciones del modelo a la realidad), así como una RMSE de 0.21 y 0.20 respectivamente. Pero el modelo de RNN simple tiene una R2 ligeramente mayor y un error cuadrático medio ligeramente menor, lo cual indica que es el mejor modelo para predecir los datos con los que se están trabajando.

Se recomienda a futuros investigadores probar ambos modelos y distintas combinaciones de neuronas, capas, epochs y batch para encontrar el mas eficiente en cada caso.

# **Bibliografía**

Brownlee, J. (2016, July 21). *Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Jiayu Qiu, ,. B. (2020). Forecasting stock prices with long-short term memory neural network based on attention mechanism. *Public Library of Science*.

Sepp Hochreiter, J. S. (1997). Long Short-Term Memory. *Neural Computation*, 1735-1780.

Wei Bao, J. Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *Plos One*, 11.

Yahoo Finance. (2021, May 19). *Grupo Elektra, S.A.B. de C.V. (ELEKTRA.MX)*. Retrieved from Yahoo Finance: https://finance.yahoo.com/quote/ELEKTRA.MX/