Universidad Autónoma de Querétaro

Facultad de Ingeniería
Maestría en Inteligencia Artificial
Sheila Leyva López
Cecilia Gabriela Rodríguez Flores
Javier Anguiano Almejo

Redes Neuronales Recurrentes: LSTM, GRU.

14 de junio del 2022

Resumen

El área de Aprendizaje Profundo ha demostrado ser de gran ayuda en diferentes ámbitos, como son en el reconocimiento de objetos, la clasificación de imágenes, reconocimiento de voz, traducción de idiomas, predicciones de acciones, etc. Los datos de serie cronológica desempeñan un papel importante, lo que demuestra la capacidad que tienen las redes neuronales de tratar este tipo de información. Debido a que por medio de ellas se puede formar una comprensión mucho más profunda de una secuencia y su contexto en comparación con otros algoritmos.

En el presente trabajo se propone la implementación de distintos modelos: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) en la base de datos sobre la predicción del precio de las acciones de la empresa IBM en el entorno de Google Colaboratory, en donde se presentan los resultados obtenidos con cada uno de los modelos mencionados anteriormente.

Objetivos

Analizar el funcionamiento y desempeño de las redes neuronales recurrentes, así como de sus variantes: LSTM y GRU, mediante su implementación con datos financieros y la librería de keras.

Justificación

En el campo del aprendizaje profundo las redes neuronales convolucionales (CNN, por sus siglas en inglés) son muy potentes. Sin embargo, cuando se busca resolver problemas que involucran datos secuenciales, no son la mejor opción, dado que, sólo les es posible procesar un solo dato a la vez, como un sonido o imagen. Es en este punto donde surgen las redes neuronales recurrentes con las cuales es posible trabajar con datos secuenciales, tales como, secuencia de sonidos o imágenes (vídeo o conversación), datos financieros en un periodo o procesamiento de datos (traductores de texto a texto, sonido a texto o texto a audio). Una de sus principales ventajas es que además tienen la capacidad de generar nuevas secuencias. Por tales motivos en esta práctica se busca aplicar los modelos, variantes de RNN, LSTM y GRU, a una base de datos de finanzas de la empresa IBM, con la finalidad de conocer el potencial de estos modelos cuando se trabaja con datos secuenciales.

Especificaciones

Herramientas utilizadas:

- Google Colab Pro
- Tensorflow versión 2.8.2
- Keras versión 2.8.0
- Conjunto de datos IBM

Conjunto de datos

En este trabajo, se utilizará la base de datos de la predicción del precio de las acciones de la empresa IBM de Kaggle [2], la cual está compuesta por 3020 filas x 6 columnas, en donde abarca del año 2006 hasta el año 2017.

Date	Open	High	Low	Close	Volume	Name
2006-01-03	82.45	82.55	80.81	82.06	11715200	IBM
2006-01-04	82.20	82.50	81.33	81.95	9840600	IBM
2006-01-05	81.40	82.90	81.00	82.50	7213500	IBM
2006-01-06	83.95	85.03	83.41	84.95	8197400	IBM
2006-01-09	84.10	84.25	83.38	83.73	6858200	IBM
...	...	...	...	...	...	...
2017-12-22	151.82	153.00	151.50	152.50	2990583	IBM
2017-12-26	152.51	153.86	152.50	152.83	2479017	IBM
2017-12-27	152.95	153.18	152.61	153.13	2149257	IBM
2017-12-28	153.20	154.12	153.20	154.04	2687624	IBM
2017-12-29	154.17	154.72	153.42	153.42	3327087	IBM

Figura 1. Base de datos IBM 2006-2017.

En este proyecto se utilizarán únicamente los valores de la columna “High”, los cuales se refieren a los valores de ventas más alto obtenidos en todo el periodo 2006-2017, sin embargo, solo se tomará en cuenta el periodo 2016-2017.

Marco Teórico

Recurrent Neural Network(RNN)

Las redes neuronales recurrentes (RNN) son relativamente antiguas, fueron creadas inicialmente en la década de 1980, pero hasta la actualidad se ha podido observar su verdadero potencial. La característica fundamental de una red neuronal recurrente (RNN) es que la red

contiene al menos una conexión de retroalimentación, por lo que las activaciones pueden fluir en un bucle, lo que les permite realizar procesamientos temporales y aprender secuencias, por ejemplo, predecir precios de acciones, generar texto, transcripciones y traducción automática.

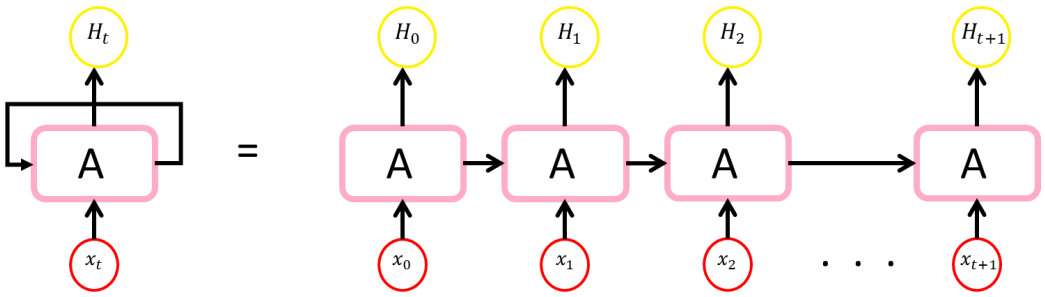

Figura 2.Red Neuronal Recurrente.

En la red neuronal tradicional, las entradas y las salidas son independientes entre sí, mientras que la salida en RNN depende de elementos previos dentro de la secuencia. Por tanto, una red neuronal recurrente (RNN) tiene dos entradas: el presente y el pasado reciente. Esto es importante porque la secuencia de datos contiene información crucial sobre lo que viene a continuación, razón por la cual una red neuronal recurrente (RNN) puede hacer cosas que otros algoritmos no pueden.

En las redes feedforward , hay diferentes pesos en cada nodo. Mientras que una red neuronal recurrente (RNN) comparte los mismos pesos dentro de cada capa de la red y durante el descenso de gradiente , los pesos y la base se ajustan individualmente para reducir la pérdida.

Limitaciones de RNN

Los módulos de repetición RNN simples tienen una estructura básica con una sola capa de tanh. La estructura simple de una red neuronal recurrente (RNN) sufre de memoria corta, donde se esfuerza por retener la información de pasos de tiempo anteriores en datos secuenciales más grandes.

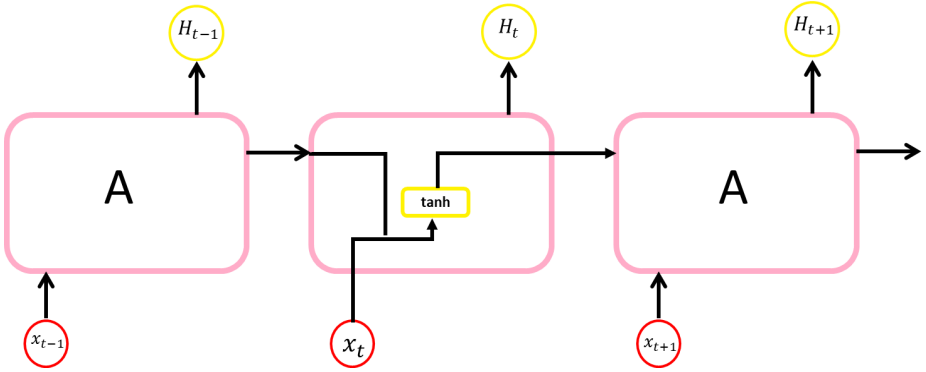

Figura 3.Celda de una Red Neuronal Recurrente Simple.

Por tanto, los modelos RNN simples generalmente se encuentran con dos problemas principales, los cuales están relacionados con el gradiente, que es la pendiente de la función de pérdida junto con la función de error, es decir, es el valor utilizado para ajustar los pesos internos de las redes, lo que permite que la red aprenda.

1. El problema del gradiente de fuga ocurre cuando el gradiente se vuelve tan pequeño que la actualización de los parámetros se vuelve insignificante; eventualmente el algoritmo deja de aprender.
2. El problema del gradiente explosivo ocurre cuando el gradiente se vuelve demasiado grande, lo que hace que el modelo sea inestable. En este caso, se acumulan gradientes de error más grandes y los pesos del modelo se vuelven demasiado grandes. Este problema puede causar tiempos de entrenamiento más prolongados y un rendimiento deficiente del modelo.

En pocas palabras, el problema proviene del hecho de que en cada paso de tiempo durante el entrenamiento estamos usando los mismos pesos para calcular su salida . Esa multiplicación también se realiza durante la retropropagación. Cuanto más nos movemos hacia atrás, más grande o más pequeña se vuelve nuestra señal de error. Esto significa que la red tiene dificultades para memorizar palabras que se encuentran muy lejos en la secuencia y hace predicciones basadas solo en las más recientes. Por eso vienen de la mano modelos más potentes como LSTM y GRU .

Long Short-Term Memory(LSTM)

Toda red neuronal utiliza un algoritmo llamado Backpropagation como método para la actualización de sus pesos, lo cual no representa inconvenientes en redes neuronales “simples”. Sin embargo, cuando se intenta utilizar en redes neuronales profundas presenta una notoria desventaja. El gradiente utilizado, a medida que avanza de regreso en la red puede que reduzca sus valores, lo que ocasiona un problema del gradiente de fuga, o bien, posiblemente sus valores crecen exponencialmente, a lo que se le denomina problema del gradiente explosivo.

Para solucionar lo anterior se propone utilizar la función ReLu como función de activación, RMSProp como optimizador y LSTM o GRU.

Específicamente, las redes LSTM (Long-Short Term Memory), son una versión mejorada de las RNN, dado que, agrega una celda de memoria para cada paso de tiempo. Se compone de los siguientes elementos:

- Compuerta de olvido **ft**: una red neuronal con una sigmoide. Permite eliminar elementos de la memoria.
- Capa candidata **c**: una red neuronal con Tanh.
- Compuerta de entrada **It**: una red neuronal con sigmoide.
- Compuerta de salida **Ot**: una red neuronal con sigmoide.
- Estado oculto **Ht**: un vector.
- Estado de memoria **Ct**: un vector.

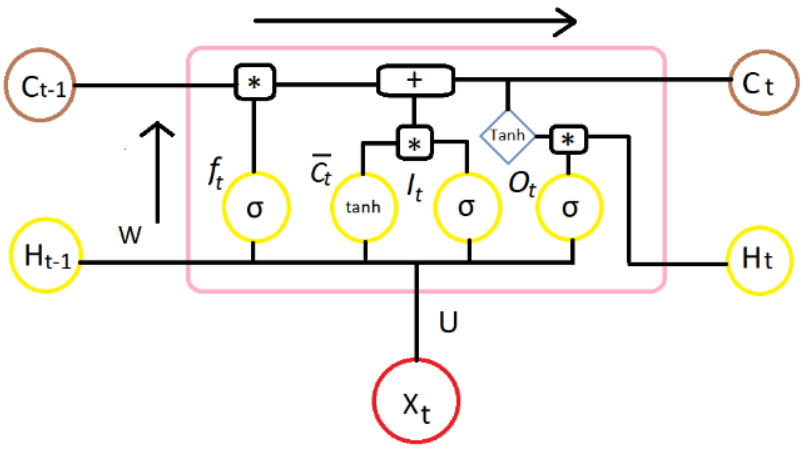

Figura 4. Diagrama de celda LSTM.

Cada compuerta funciona como una válvula, totalmente abiertas permiten el paso de información y cerradas lo bloquean por completo. Cada compuerta está conformada por una red neuronal, una función sigmoidal (le da a la compuerta el comportamiento de válvula) y un elemento multiplicador

Básicamente, la celda LSTM tiene una celda de estado (memoria), a la cual se puede añadir o remover datos. Primero, la compuerta de olvido permite decidir qué información se descarta y no pasará a la celda de estado, para ello toma el estado oculto anterior y la entrada actual, transformándolos mediante la función de activación sigmoidal, donde Wf y bf se aprenden durante el entrenamiento y como salida se obtiene el vector ft, si uno de los valores del vector es cero o muy cercano a él, la LSTM eliminará esa información. Mientras que, si alcanza valores de uno o cercanos a él, la información se mantendrá y llegará a la celda de estado:

$$f_t = sigmoidal(W_f[H_{t-1}, x_t] + b_f$$

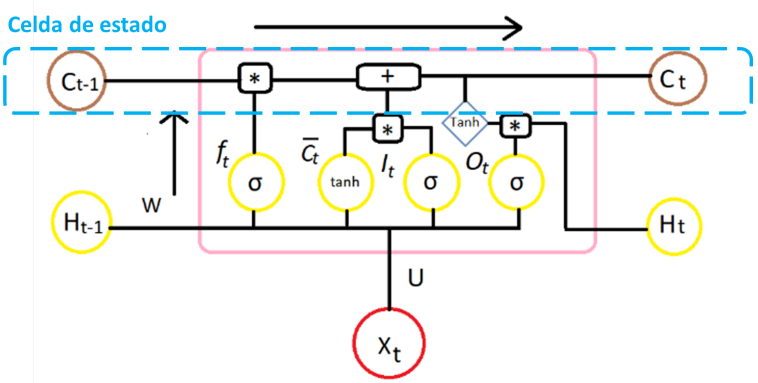

Figura 5. Diagrama de celda LSTM: celda de estado (memoria).

Ahora bien, para agregar nueva información se utiliza la compuerta de actualización, por lo cual se toma nuevamente el estado oculto anterior y la entrada actual y se aplica la función sigmoidal, obteniendo el vector It:

$$I_t = sigmoidal(W_f[H_{t-1}, x_t] + b_f$$

Una vez generados por las compuertas de olvido y actualización, se procede a actualizar la celda de estado, es decir, la memoria de la LSTM. Para ello se elimina la información irrelevante:

$$f_t * C_{t-1}$$

A continuación, se genera un vector de valores candidatos a formar parte de la nueva memoria:

$$\bar{C}_t = tanh(W_c[H_{t-1}, x_t] + b_c$$

Se filtran los valores obtenidos del vector candidato multiplicandolo por el vector de entrada

$$I_t * \bar{C}_t$$

para posteriormente sumar el resultado a los valores anteriores de la celda de estado, generando así la memoria actualizada

$$f_t * C_{t-1} + I_t * \bar{C}_t$$

Por último, se debe calcular el nuevo estado oculto, el cual es simplemente una versión filtrada del estado de celda. Primero se escala el estado de celda para garantizar que está dentro del rango [-1,1], para determinar qué información del estado de celda formarán parte del nuevo estado oculto se utiliza la compuerta de salida

$$O_t = sigmoidal(W_o[H_{t-1}, x_t] + b_o$$

Finalmente, se filtran los valores del estado de celda:

$$O_t * tanh[C_t]$$

Gated Recurrent Unit (GRU)

La red de unidad recurrente cerrada (GRU) es otra red neuronal recurrente basada en el control de puerta. Dicha red mejora la memoria de largo tiempo a un costo computacional menor que las LSTM a costa de un menor desempeño. La estructura de red de GRU [7] es más simple que la de LSTM. GRU fusiona la puerta de entrada y la puerta de olvido en LSTM en una puerta, llamada puerta de actualización. En la red GRU, no hay división entre el estado interno y el estado externo en la red LSTM. En cambio, el gradiente se resuelve agregando directamente una dependencia lineal entre el estado actual de la red ht y el estado anterior de la red ht-1 [8].

En estas redes, la salida y la memoria van por el mismo conducto, se puede ver a ht-1 como la estimación actual de la red dada la información al momento disponible.

Existen dos switches (gates) denominados:

1.- Switch de “reset”, que hace a la unidad olvidar lo acumulado en memoria para calcular su nueva salida:

$$rt = \sigma(Wrxxt + Wroot-1 + bz)$$

2.- Switch de “actualización”, que hace que la solución acumulada sea actualizada o se usa la solución anterior como solución de la unidad actual. Es útil para los casos en que la información actual no aporta información relevante.

$$zt = \sigma(Wzxxt + Wzoot-1 + bz)$$

La salida de la red se selecciona con z mediante:

$$ot = ztot-1 + (1-zt)\tilde{O}t$$

Donde ot es la respuesta de la red dada la información disponible: xt y $ot-1$. Se calcula mediante:

$$\tilde{O}t=\phi(Wxt+rt\odot\tilde{O}t-1)$$

Donde $\odot$ representa el producto punto a punto, aquí es donde se olvida o no la información en la memoria.

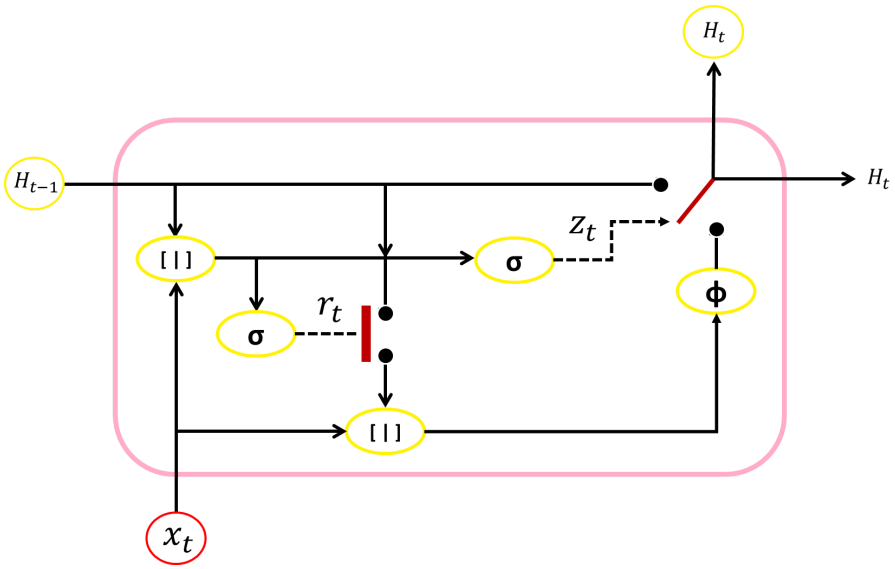

Figura 6. Diagrama de Unidad Recurrente Cerrada.

- **[|]:** Operación de Concatenación.
- **H_t-1:** Salida de la celda anterior.
- **r_t**: Decide si procesa la salida de la celda anterior.
- **z_t**: Decide si la salida es tomada de la salida anterior o la procesada por la celda.
- **σ**: Función sigmoide, aproximan a una respuesta [0,1].
- **ϕ**: Función tangente hiperbólica, aproximan a una respuesta [-1,1].

Desarrollo

Para la elaboración de este trabajo, se utilizaron los modelos Long Short-Term Memory(LSTM) y Gated Recurrent Unit(GRU). La metodología utilizada para la implementación de los modelos mencionados anteriormente fue muy similar entre sí.

Por lo cual, a continuación se visualiza por medio de la siguiente gráfica, en donde el conjunto de prueba consta únicamente de un año, del 2017-2018, mientras que el resto del conjunto de datos sobre la predicción del precio de las acciones de la empresa IBM se utiliza para el conjunto de entrenamiento.

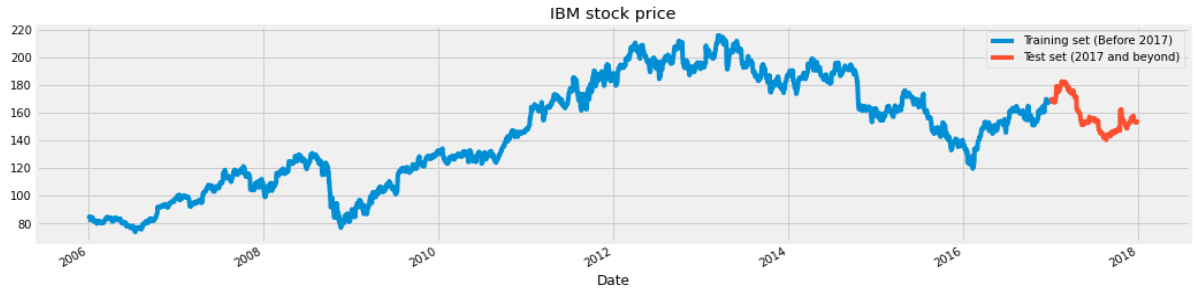

Figura 7. Visualización del conjunto de datos de entrenamiento y prueba de la base de datos propuesta.

Preprocesamiento de datos

Se utilizará la función MinMaxScaler para estandarizar el conjunto de entrenamiento, lo cual ayuda a evitar los valores atípicos o las anomalías.

```
# Escalamiento
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

Dado que los LSTM almacenan el estado de la memoria a largo plazo, creamos una estructura de datos con 60 pasos de tiempo y 1 salida. Entonces, para cada elemento del conjunto de entrenamiento, tenemos 60 elementos del conjunto de entrenamiento previo.

```
# Ajuste de conjunto de datos de entrenamiento
X_train = []
y_train = []
for i in range(60,2769):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
# Re ajustando el conjunto de entrenamiento para tener los bloques de 60x1.
```

```
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

Antes de proseguir con los resultados obtenidos de cada uno de los modelos propuestos, se procede a visualizar la arquitectura de los modelos propuestos.

En la figura 8 y 9 se muestran las arquitecturas de los modelos LSTM y GRU, los cuales constan únicamente de 4 capas ocultas, en donde la primera capa contiene 50 unidades, seguida sucesivamente de otras 3 capas con 50 unidades. Así bien, se aplica la técnica de dropout después de cada capa para controlar el sobreajuste. Por último, se conecta con una capa final densa con unidad simple.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 50)            10400

 dropout (Dropout)           (None, 60, 50)            0

 lstm_1 (LSTM)               (None, 60, 50)            20200

 dropout_1 (Dropout)         (None, 60, 50)            0

 lstm_2 (LSTM)               (None, 60, 50)            20200

 dropout_2 (Dropout)         (None, 60, 50)            0

 lstm_3 (LSTM)               (None, 50)                20200

 dropout_3 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
_________________________________________________________________
```

Figura 8. Arquitectura del modelo LSTM.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_4 (GRU)                 (None, 60, 50)            7950

 dropout_12 (Dropout)        (None, 60, 50)            0

 gru_5 (GRU)                 (None, 60, 50)            15300

 dropout_13 (Dropout)        (None, 60, 50)            0

 gru_6 (GRU)                 (None, 60, 50)            15300

 dropout_14 (Dropout)        (None, 60, 50)            0

 gru_7 (GRU)                 (None, 50)                15300

 dropout_15 (Dropout)        (None, 50)                0

 dense_3 (Dense)             (None, 1)                 51

=================================================================
Total params: 53,901
Trainable params: 53,901
Non-trainable params: 0
_________________________________________________________________
```

Figura 9. Arquitectura del modelo GRU.

TABLA 1
Parámetros asociados a los modelos propuestos LSTM y GRU para dropout 0.2 y 0.5.

Modelos propuestos	Épocas	Tamaño del Batch	Optimizador	Pérdida
Modelo 1 LSTM	50	32	Rmsprop	MSE
		64		
		150		
	100	32		
		64		
		150		
Modelo 2 LSTM	50	32	Adam	MSE
		64		
		150		
	100	32		
		64		
		150		
Modelo 3 GRU	50	32	SGD	MSE
		64		
		150		
	100	32		
		64		
		150		

TABLA 2

Parámetros asociados a los modelos propuestos GRU.

Modelos propuestos	Épocas	Tamaño del Batch	Opt	Tasa de Aprendizaje	Decay	Momentum	Nesterov	Pérdida
Modelo 3 GRU	50	32	SGD	0.01	1e-7	0.9	False	MSE
		64						
		150						
	100	32						
		64						
		150						

Resultados obtenidos

Al realizar la ejecución de cada uno de los modelos fue posible obtener diversos resultados para cada uno de los modelos propuestos, por lo cual a continuación se mostrarán los resultados obtenidos por medio de tablas y figuras.

En las Figuras 10 - 27 , se muestra el resultado del entrenamiento del conjunto de datos de IBM en los modelos LSTM y GRU con un optimizador de Rmsprop, Adam y GSM, realizando diferentes combinaciones en conjunto a los parámetros batch size [32, 64, 150], épocas [50 100] y Dropout (DO) [0.2, 0.5] respectivamente. Asimismo, en la Tabla 3 y 4 se muestra el valor obtenido de Media de Error Cuadrático para cada uno de los 36 experimentos realizados.

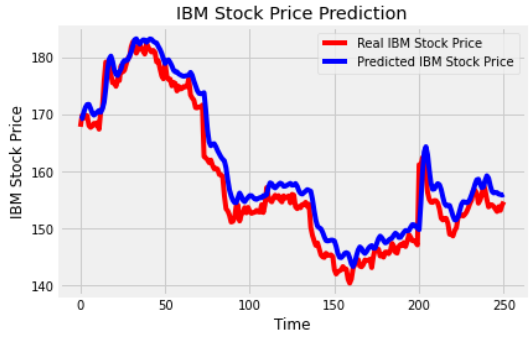

Figura 10. LSTM con batch_size = 32, 50 épocas, DO=0.2, RMSprop

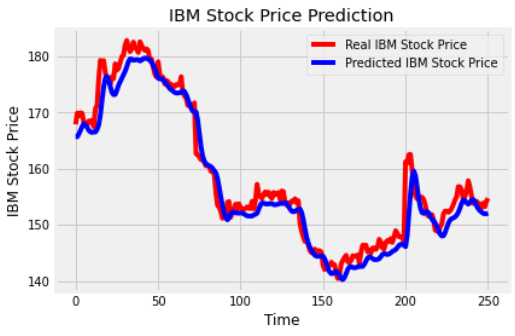

Figura 11. LSTM con batch_size = 32, 50 épocas, DO=0.2, Adam

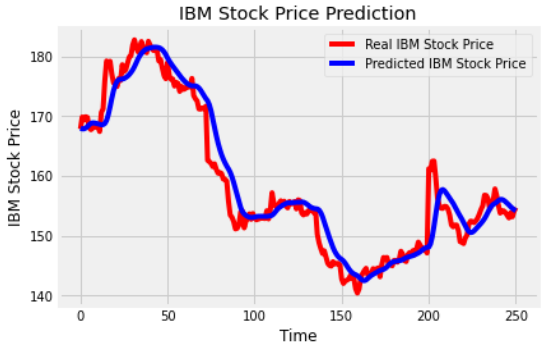

Figura 12. GRU con batch_size = 32, 50 épocas, DO=0.2, GSD.

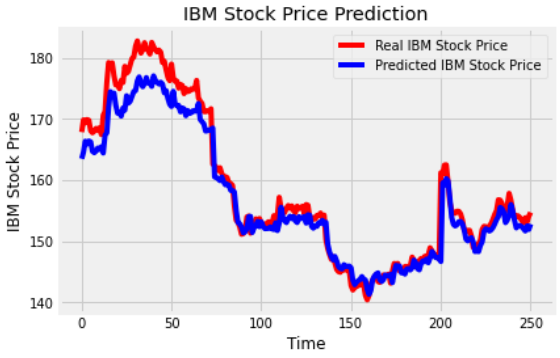

Figura 13. LSTM con batch_size = 64, 50 épocas, DO=0.2, RMSprop

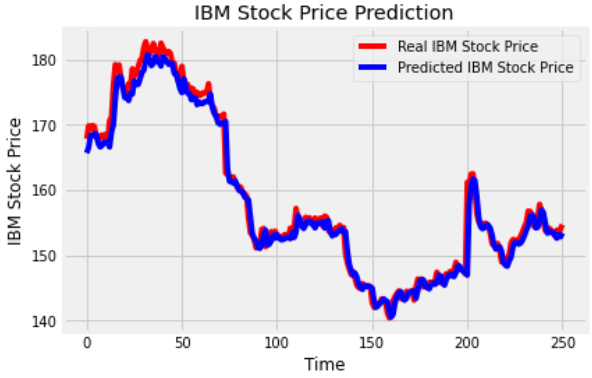

Figura 14. LSTM con batch_size = 64, 50 épocas, DO=0.2, Adam

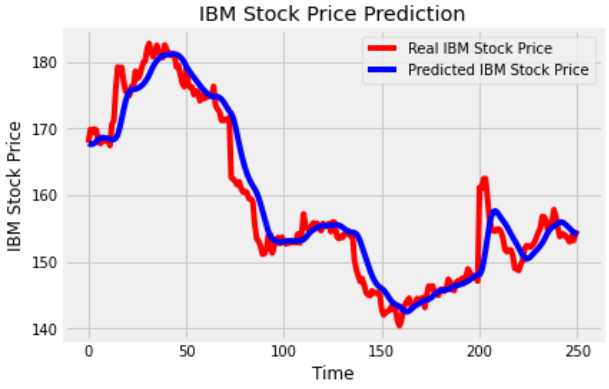

Figura 15. GRU con batch_size = 64, 50 épocas, DO=0.2, SGD.

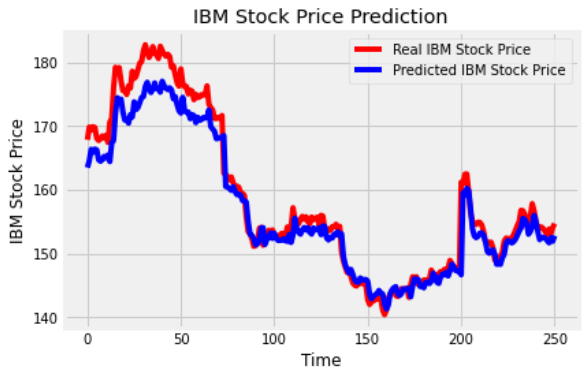

Figura 16. LSTM con batch_size = 150, 50 épocas, DO=0.2, RMSprop

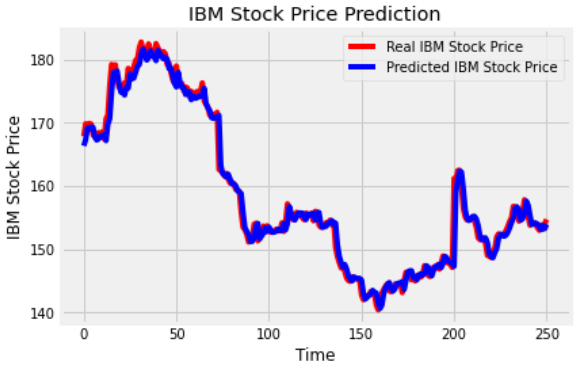

Figura 17. LSTM con batch_size = 150, 50 épocas, DO=0.2, Adam

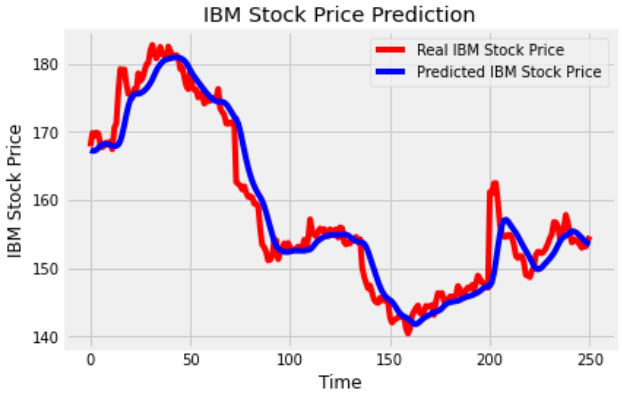

Figura 18. GRU con batch_size = 150 50 épocas, DO=0.2, SGD.

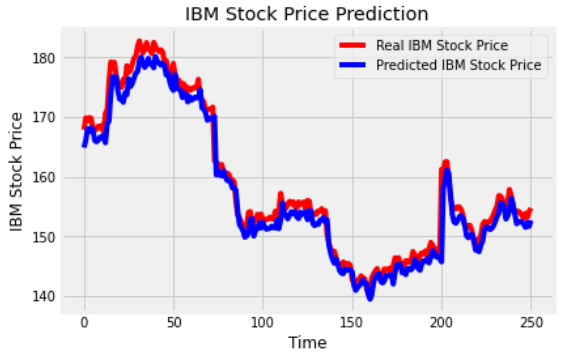

Figura 19. LSTM con batch_size = 32, 100 épocas, DO=0.2, RMSprop

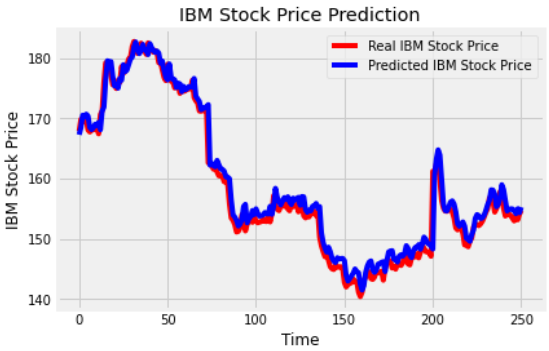

Figura 20. LSTM con batch_size = 32, 100 épocas, DO=0.2, Adam

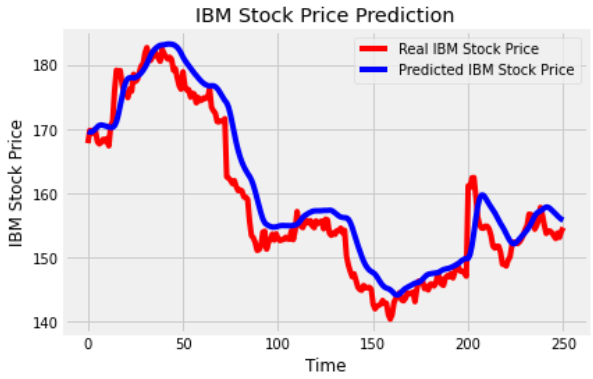

Figura 21. GRU con batch_size = 32, 50 épocas, DO=0.2, GSD.

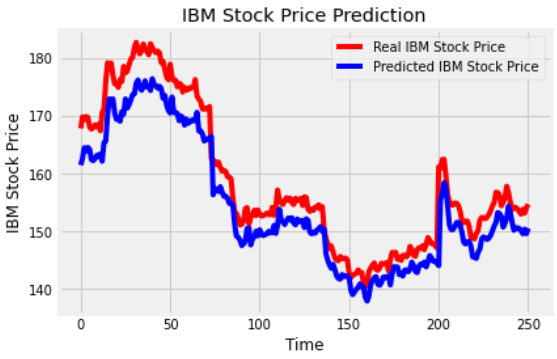

Figura 22. LSTM con batch_size = 64, 100 épocas, DO=0.2, RMSprop

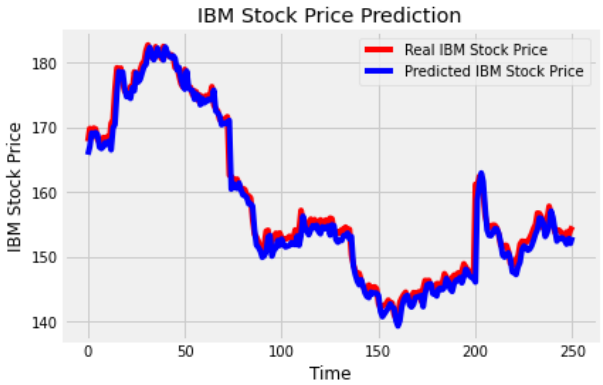

Figura 23. LSTM con batch_size = 150, 100 épocas, DO=0.2, Adam

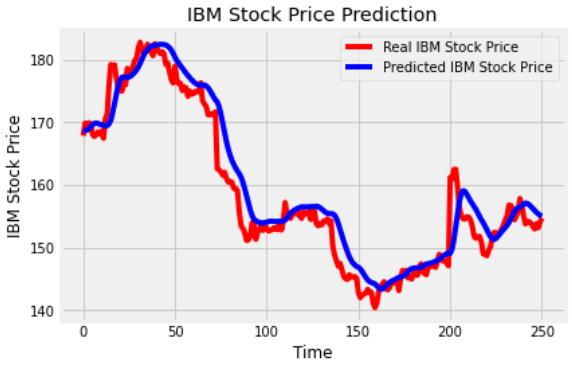

Figura 24. GRU con batch_size = 150, 100 épocas, DO=0.2, SGD.

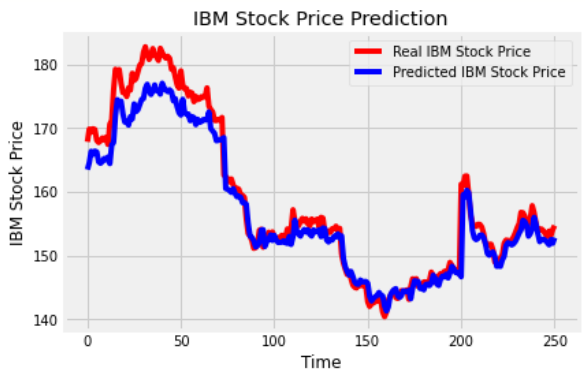

Figura 25. LSTM con batch_size = 150, 100 épocas, DO=0.2, RMSprop

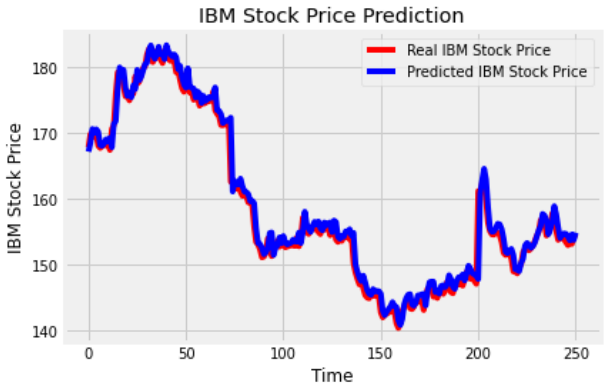

Figura 26. LSTM con batch_size = 150, 100 épocas, DO=0.2, Adam

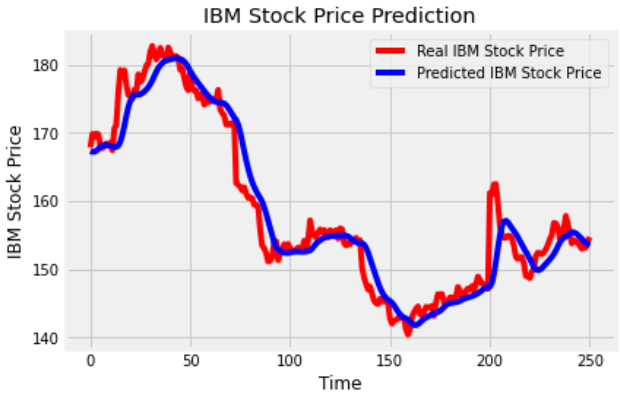

Figura 27. GRU con batch_size = 150 100 épocas, DO=0.2, SGD.

Análisis de los resultados

En esta sección se realizará una comparación de los resultados obtenidos al realizar la implementación de cada uno de los modelos propuestos, por lo cual a continuación se podrán visualizar las siguientes tablas comparativas.

TABLA 3

Comparación de resultados obtenidos de los modelos propuestos para Dropout 0.2.

Modelos propuestos	Optimizador	Épocas	Tamaño del Batch	MSE
Modelo 1 LSTM	Rmsprop	50	32	3.077208835414
			64	2.404169076492
			150	2.896142749793
		100	32	2.222818247007
			64	2.784897638018
			150	4.496653558203
Modelo 2 LSTM	Adam	50	32	2.772550229470
			64	1.744395071936
			150	**1.580665404610**
		100	32	1.847839921916
			64	1.776294549151
			150	1.675534385474
Modelo 3 GRU	SGD	50	32	3.202733213327
			64	3.152270941329
			150	3.131083635684
		100	32	3.788674971274
			64	3.321515193038
			150	2.983989701985

TABLA 4

Comparación de resultados obtenidos de los modelos propuestos para Dropout 0.5.

Modelos propuestos	Optimizador	Épocas	Tamaño del Batch	MSE
Modelo 1 LSTM	Rmsprop	50	32	3.502538250896650
			64	2.524334632006038
			150	3.101617866518339
		100	32	2.023684618783070
			64	1.833243404136719
			150	5.579822424094887
Modelo 2 LSTM	Adam	50	32	3.322587878021638
			64	1.872113359477385
			150	**1.673752986849156**
		100	32	1.911484885855271
			64	2.528009041864300
			150	1.758206474071890
Modelo 3 GRU	SGD	50	32	4.741710670942616
			64	3.435217109928659
			150	3.619718213344062
		100	32	3.453744892482385
			64	3.548597060666450
			150	3.572726638978344

Conclusiones

La realización de esta tarea se enfocó en conocer el comportamiento de los modelos propuestos basados en redes neuronales recurrentes: LSTM y GRU; en donde para ello fue necesario la selección de una base de datos, que nos permitiera realizar una predicción de datos a partir de un historial del comportamiento de los mismos. De igual forma se intentó modificar los hiperparametros (como son el tipo de optimizador, el tamaño del batch, el número de épocas y el porcentaje de dropout) de los modelos durante su entrenamiento y analizar si existían cambios en las predicciones, cambios en los resultados al variar ciertos parámetros

Por lo cual, en la sección de resultados fue posible observar que la mayoría de los modelos implementados dieron valores muy cercanos entre sí, aunque al analizar la tabla 3 y la tabla 4 se puede observar que el modelo con la menor pérdida fue el modelo LSTM con el optimizador Adam, el cual dio una predicción más precisa ya que su comportamiento era muy similar con el precio de las acciones reales de la empresa IBM.

Por el contrario, para el caso de los modelos GRU se pudo observar que su resultados no eran tan detallados como los modelos LSTM, pero aun así mantienen cierta similitud con el precio de las acciones reales de la empresa IBM.

Analizando los resultados de los experimentos dentro de las tablas se pudo observar que al incrementar el Batch en el modelo LSTM con optimizador Adam y entrenamiento de 50 épocas, DO de 0.2; Generó un efecto de reducción del MSE, a diferencia del modelo con optimizador Rmsprop, el cual tuvo su mejor resultado con un Batch de 64. Así mismo, en el caso de las 100 épocas se observó el caso contrario al obtener un incremento en las pérdidas. No obstante este efecto expresó una mayor relación en los casos donde se entrenó con el optimizador Adam.

En contrario para el modelo GRU, el incremento de Batch, en ambos experimentos con entrenamiento de 50 y 100 épocas, este representó siempre una reducción del MSE.

Para los experimentos con DO de 0.5, el comportamiento de la selección de Batch cambió, ya que los mejores resultados para el modelo LSTM con optimizador Rmsprop, se dieron para los

casos con Batch de 64. Y para el modelo LSTM entrenado con optimizador Adam, se dieron con Batch de 150 para ambos casos de entrenamiento.

Para el modelo GRU, esté expresó como mejor resultado en los entrenamientos de 50 épocas con una configuración de Batch de 64, difiriendo en el entrenamiento de 100 épocas, donde el resultado con menor MSE se encontraba en la configuración de Batch de 32.

Este análisis permite denotar que el MSE tiene una respuesta al cambio de Batch con respecto al número de épocas, sirviendo así como factor de interés para próximos experimentos. Cabe destacar que la configuración del modelo LSTM con optimizador Adam, Batch de 150 en periodos de entrenamiento de 50 épocas, expresó siempre los mejores resultados.

Referencias

[1] DeePam Gautam (2019) Stock Price Prediction,LSTM, GRU, RNN. Recuperado el día 11 de Junio de 2021 de https://www.kaggle.com/code/dpamgautam/stock-price-prediction-lstm-gru-rnn/notebook

[2] Yadav, S. (2018) Intro to Recurrent Neural Network LSTM | GRU. Recuperado el día 12 de Junio de 2022 de https://www.kaggle.com/code/thebrownviking20/intro-to-recurrent-neural-networks-lstm-gru/data?select=IBM_2006-01-01_to_2018-01-01.csv

[3] Codificando Bits (2018) Predicción de acciones en la bolsa con PYTHON (tutorial redes LSTM). Recuperado el día 12 de Junio de 2022 de https://www.youtube.com/watch?v=3kXj6VgxbP8

[4] Donges, N. (2021) Una guía para RNN: comprensión de las redes neuronales recurrentes y las redes LSTM. Recuperado el día 13 de Junio de 2022 de https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[5] Miguel Phi (2018) Guía ilustrada de redes neuronales recurrentes. Recuperado el día 13 de Junio de 2022 de https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

[6] All Awan, A. (2022) Tutorial de redes neuronales recurrentes (RNN). Recuperado el día 13 de Junio de 2022 de https://www.datacamp.com/tutorial/tutorial-for-recurrent-neural-network

[7] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches,” Sep. 2014, [Online]. Available: http://arxiv.org/abs/1409.1259

[8] Mariano Rivera (2018) Introducción a Redes Neuronales Recurrentes (RNN). Recuperado el dia 13 de Junio de 2022 de http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/RNN_LTSM/introduccion_rnn.html#:~:text=El%20estado%20oculto%20s%20t,para%20procesar%20el%20nuevo%20dato

[9] Codificando Bits (2018) Introducción a redes neuronales recurrentes. Recuperado el día 12 de Junio de 2022 de https://www.codificandobits.com/blog/introduccion-redes-neuronales-recurrentes/

[10] Michael Phil.Codificando (2018). Modelos de secuencia. Todo lo que necesitas es atención. Recuperado el día 12 de Junio de 2022 de https://www.themachinelearners.com/modelos-secuencia/