Universidad Nacional Autónoma de México

Facultad de Ciencias

Predicción de precios de acciones e índices usando redes neuronales LSTM

T E S I S

que para obtener el título de:

Actuario

P R E S E N T A :

Luis Vicente Ruiz Hernández

tutor

Dr. Ivan Vladimir Meza Ruiz

Ciudad Universitaria, Cd. Mx., 2021

Agradecimientos

A mis padres, por su infinito amor y cariño que recibo de ellos, por siempre apoyarme en mi vida, orientarme y guiarme para alcanzar mis metas, por estar siempre para mi, por su paciencia tan grande y la motivación que me han dado para buscar y perseguir mis sueños, son una inspiración para mi y estaré siempre tan agradecido por cada lección y enseñanza que me transmitieron y que hicieron que aprendiera todo lo necesario para llegar a estar donde estoy.

A mis abuelos Vicente y Luis y mis abuelas Isabel y Matilde, por darme una gran vida y por estar para mi en todo momento , gracias a ellos tuve y tengo todo lo necesario para desarrollarme como persona

A mi hermano y mi primo David, que me ayudaron en mi llegada a la universidad, siempre me brindaron su apoyo y me motivaron cada día a esforzarme para ser mejor persona.

A mi tía Elsi que siempre estuvo dispuesta a aconsejarme y ayudarme, me apoyó en mi camino a la ciencia brindándome su tiempo. A mis tías Nidia y Karina que me brindaron todo el apoyo a lo largo de mi carrera.

A Allison Odette, con quien compartí gran parte de la carrera y me ayudó en los momentos difíciles de la universidad y de mi vida, siempre estuvo para mi y desde que la conocí hizo que mi vida se llenara de amor.

A mi director de tesis, el doctor Ivan Vladimir Meza Ruiz, por aceptar trabajar conmigo y asesorarme al escribir el presenta trabajo; por sus conocimientos y su dedicación en enseñarme; por su paciencia y su forma de ser que facilitaron la realización del trabajo. Es una inspiración para mi y me siento muy feliz que la UNAM tenga profesores como el doctor Ivan Vladimir.

Índice general

Índice de figuras

Capítulo 1

Introducción

En los mercados financieros, particularmente en el mexicano, el mercado accionario ocupa un puesto importante ya que en este, las empresas obtienen inversión para financiarse y solventar las deudas mediante la venta de las acciones, que representan una parte del capital de la empresa; al mismo tiempo que la empresa obtiene financiamiento, los compradores y poseedores de acciones obtienen un beneficio por estas.

Al negociarse las acciones en los mercados, estas pueden generar una ganancia a los poseedores mediante estrategias de inversión, entre estas estrategias la más simple consiste en comprar una cantidad de acciones de una empresa cuando el precio de estas esté por incrementar, de manera que al alcanzar un precio mayor al precio de compra, se pueda vender y así de la diferencia entre los precios de compra y venta se tenga un saldo positivo que se traduce en ganancias.

Por lo anterior es de gran utilidad conocer los movimientos tanto pasados como futuros que tendrá el precio de las acciones pues se puede obtener un beneficio económico de esta información. Al igual que es importante conocer los precios de las acciones, lo es conocer los índices accionarios del mercado en el que se quiera invertir, ya que estos reflejan la situación económica y financiera de un conjunto de empresas e incluso del país.

Tomando en cuenta los beneficios que se pueden tener de conocer los precios y movimientos futuros de las acciones e índices, la tarea de predecirlos es de mucho

interés para los inversionistas y las empresas, sin embargo esta tarea es muy compleja ya que en el mercado accionario se presentan datos estocásticos que pueden o no presentar tendencias, además que estos datos se ven influenciados por muchos factores económicos, políticos, sociales tanto de las empresas como del mercado y del país en donde se realizan las operaciones. Para predecir tanto el valor futuro de los precios como su comportamiento, ya sea alcista o bajista, se han usado distintos métodos como el análisis técnico y el análisis fundamental, además de modelos matemáticos y herramientas como la estadística.

En la actualidad gracias a los avances tecnológicos y a una mayor interacción entre los campos de las matemáticas y el cómputo se han desarrollado algoritmos, llamados algoritmos de aprendizaje automático o de *machine learning*, que permiten resolver problemas de una mejor manera ya que el tiempo empleado por estos para resolver los problemas es menor en comparación al empleado usando otras técnicas y algoritmos, además que se obtienen resultados parecidos o incluso mejores usando el aprendizaje automático. Debido a la efectividad y buen desempeño de los algoritmos de *machine learning*, estos se comenzaron a aplicar a diferentes problemas de varias áreas no solo científicas sino también sociales, dando como resultado un mayor número de algoritmos que van desde las regresiones lineales hasta las redes neuronales, creando así modelos que permiten clasificar, agrupar, predecir, estimar funciones, entre otras.

Conforme la tecnología fue permitiendo, se aplicaron cada vez más algoritmos de aprendizaje automático a las tareas relacionadas al ambiente financiero, siendo una de las principales el predecir los precios de las acciones así como ajustar modelos que concentraran la información importante de los precios como su tendencia, si es que la hubiera, así como la aleatoriedad de los datos y su estacionalidad.

1.1. Planteamiento del problema

En la actualidad los algoritmos de *machine learning* se han vuelto más populares debido a la cercanía que se tiene con la tecnología y la facilidad con la que estos algoritmos se aplican a los diferentes problemas. Por otro lado debido a los diferentes

problemas económicos y financieros por los que las personas pasan se ha vuelto cada vez más necesario el buscar ingresos por diferentes fuentes, y gracias a la gran cantidad de información referente a los mercados accionarios que se encuentra al alcance de todos, el invertir se ha convertido en una manera de conseguir dinero. De igual manera para las compañías que buscan incrementar sus ingresos, el saber en donde invertir su capital es una gran prioridad para aumentar las ganancias.

Es por lo anterior que en el presente trabajo se propuso usar algoritmos de aprendizaje automático, específicamente redes neuronales *Long Short-Term Memory* o LSTM [Hochreiter and Schmidhuber, 1997] para crear modelos predictivos que sirvan como una herramienta para poder invertir y obtener ganancias en la bolsa mexicana. Para la predicción de valores futuros existen diversos algoritmos y modelos como la regresión lineal o logística, *Support Vector Regression* (SVR), modelos *ARIMA-GARCH*, árboles de decisión, entre otros, sin embargo para este caso se buscará construir modelos de redes neuronales que sean lo más simple posible en cuestión de tiempo empleado y de arquitectura de redes, y que el error obtenido de estos modelos sea semejante al de otros algoritmos e incluso buscar disminuir este error lo más posible.

1.2. Objetivos

El objetivo principal del presente trabajo es crear diversos modelos estadísticos para procesar los valores de índices accionarios y empresas que cotizan en la bolsa mexicana de valores con la finalidad de predecir los valores futuros. Para lograr este objetivo se plantean los siguientes objetivos específicos:

- Describir las bases teóricas financieras y estadísticas con las cuales se fundamentará el presente trabajo.
- Determinar los modelos estadísticos de aprendizaje automático a usar para procesar la información histórica y predecir los valores futuros.
- Diseñar los experimentos que servirán para comparar la capacidad de los modelos para realizar la tarea de predicción.

- Usando el error cuadrático medio como medida de desempeño evaluar los modelos.

- Analizar los resultados obtenidos y de acuerdo a estos identificar el modelo que mejor realice la tarea antes mencionada.

Para realizar los objetivos anteriores se plantea seguir el procedimiento a continuación mencionado, el cual constará de diferentes pasos para desarrollar los experimentos de predicción.

1.3. Procedimiento

Para resolver el problema planteado se seguirá una serie de pasos listados a continuación:

- Obtención de los datos: Para usar cualquier algoritmo de *machine learning* se necesita tener una cantidad suficiente de datos, en el caso del trabajo, dado que se planteó predecir los valores de acciones así como de un índice accionario, se toma el índice mexicano S&P/BMV IPC así como las 5 empresas más representativas que componen a dicho índice, de estas se descargan los precios históricos del 4 de enero del 2010 al 30 de diciembre del 2016.

- Creación de la red: Usando el lenguaje de programación *Python* y la biblioteca *Keras* se construirán las redes LSTM donde cada una tiene una configuración distinta. Estas redes procesarán los precios tanto del índice como de las empresas seleccionadas.

- Evaluación de los modelos: De las redes construidas se obtienen diferentes modelos predictivos y de acuerdo a las predicciones obtenidas se calculan los errores de cada modelo.

1.4. Estructura de la tesis

- Marco teórico: Se presentan los conceptos, tanto financieros y económicos como los correspondientes al aprendizaje automático, necesarios para poder desarrollar el trabajo y tener un entendimiento de lo que se desea realizar en el presente trabajo.

- Metodología: En esta sección se profundiza en los procedimientos necesarios para realizar los experimentos y obtener los posteriores resultados. Se expone la forma de llevar a la práctica los conceptos teóricos expuestos en el trabajo.

- Experimentación y resultados: De acuerdo a las diferentes arquitecturas de redes seleccionadas se explora la eficiencia de estas al realizar la tarea de predicción, asimismo se proponen diferentes escenarios de predicción seleccionando diferentes conjuntos de datos.

- Conclusiones: De acuerdo a los resultados obtenidos se presenta una interpretación general de estos además de una reflexión y opinión. Finalmente, se abordan ideas complementarias que se podrán llevar a cabo en futuras investigaciones de manera que complementen el presente trabajo.

Capítulo 2

Sistema Financiero Mexicano

En este capítulo se da una introducción al Sistema Financiero para así conocer su funcionamiento y cómo está conformado, enfocándose en el Mexicano ya que será la base sobre la cual se trabaja.

En el mundo actual las finanzas tienen un papel importante tanto para los países como para las empresas, personas morales y las personas físicas. Un buen manejo de las finanzas ayuda a un país a mejorar su economía, así como crear oportunidades de crecimiento para las personas físicas y morales, así como una fuente de financiamiento; por el contrario, un mal manejo de las finanzas repercute en perdidas para las empresas e instituciones, además de un desequilibrio en la economía del país e incluso mundial.

Para administrar las finanzas alrededor del mundo y distribuir el riesgo existe el Sistema Financiero. Este es un conjunto de empresas e instituciones, en donde se maneja información económica y financiera, se distribuye el riesgo existente entre los participantes y se recolecta el capital y dinero de los inversionistas, ya sean personas físicas, morales o instituciones; y se asigna capital financiero a las empresas y personas que necesiten de inversión [Diebolt and Haupert, 2019].

En el caso del Sistema Financiero Mexicano, éste está constituido por las organizaciones e instituciones públicas y privadas que se encargan de captar, organizar, controlar, canalizar y distribuir los recursos financieros de las personas a la inversión. Es decir, el sistema financiero se encarga de recibir los recursos de los lugares donde son más abundantes y poco necesarios, y distribuirlos a los lugares donde se tiene

escasez de recursos.

2.1. Estructura del Sistema Financiero Mexicano

El conjunto de instituciones públicas y privadas que conforman el Sistema Financiero Mexicano pueden clasificarse en dos grupos, donde el primero es aquel compuesto por las instituciones que se encargan de regular y dirigir las políticas financieras, así como el control del flujo del dinero y de crédito. El segundo grupo son aquellas que están dedicadas a la operatividad de las finanzas [Reséndiz Puente, 2014].

- Entre las instituciones del primer grupo, que llamaremos instituciones reguladoras se encuentran:
  - Secretaría de Hacienda y Crédito Público,
  - Banco de México,
  - Comisión Nacional Bancaria y de Valores,
  - Comisión Nacional de Seguros y Fianzas,
  - Comisión Nacional del Sistema de Ahorro para el Retiro.
- Las instituciones del segundo grupo, a las que llamaremos instituciones operativas, podemos encontrar:
  - Instituciones de crédito (sistema bancario),
  - Sistema bursátil,
  - Instituciones de seguros y fianzas,
  - Grupos Financieros.

Como se puede observar, las instituciones reguladoras son públicas, mientras que las operativas contemplan también instituciones privadas.

De acuerdo a la lista anterior se puede hacer un pequeño diagrama de la estructura del Sistema Financiero Mexicano.

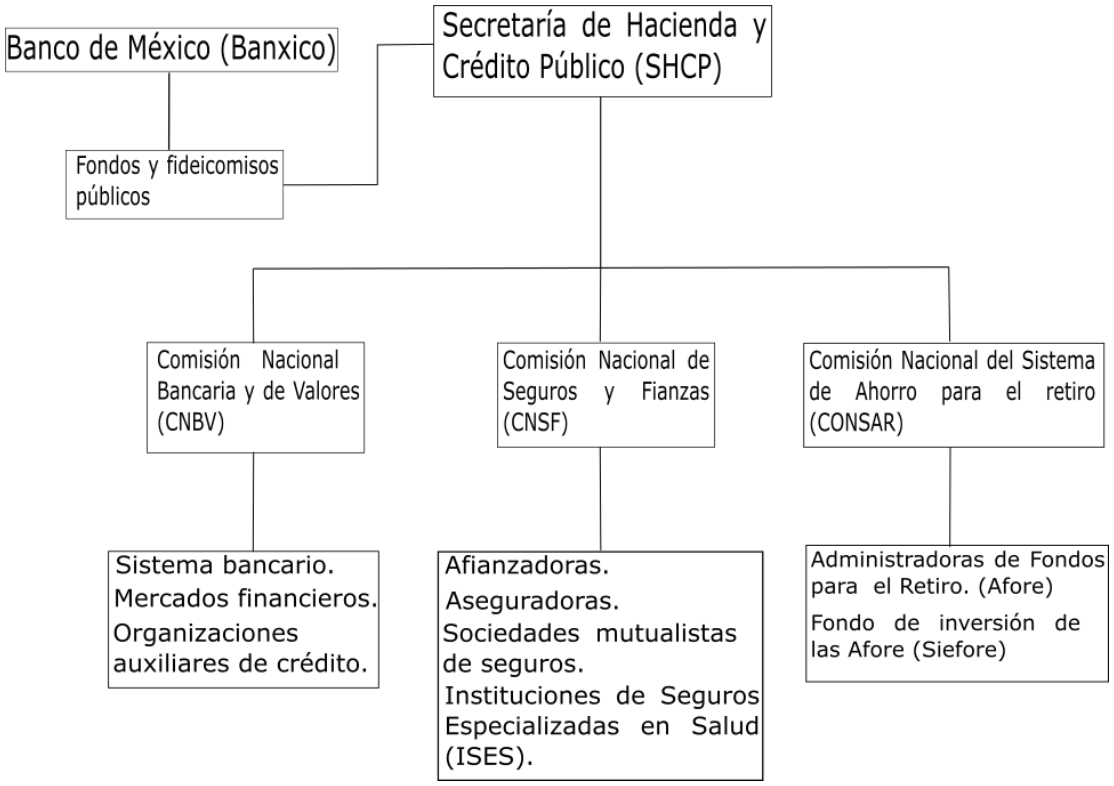


Figura 2.1: Composición del Sistema Financiero Mexicano.

En el Sistema Financiero Mexicano, la SHCP es la institución que se encarga de supervisar y controlar la política federal en el ámbito financiero, de ingresos, gastos, de deuda pública, entre otros [Secretaría de Hacienda y Crédito Público, 2017]. Estas funciones se realizan a través de tres comisiones que son la CNBV, la CNSF y la CONSAR, donde estas subdividen al sistema financiero en tres subsistemas:

- Sistema bancario y de valores
- Sistema asegurador y afianzador
- Sistema de ahorro para el retiro

Además de dividirlo en 3 subsistemas, el Sistema Financiero Mexicano se constituye por varios sectores, los cuales son: sector bancario, sector no bancario, sector de

pensiones, sector bursátil, sector de seguros y fianzas, sector de derivados y sector de ahorro y crédito popular.

2.2. Funciones de los Sistemas Financieros

La principal función de los sistemas financieros es asignar y distribuir los recursos económicos de quienes tienen y no los necesitan a corto plazo a aquellas organizaciones que lo necesitan para financiar sus necesidades y requerimientos [Bodie and Merton, 1998].

Quienes tienen recursos y están dispuestos a ofrecerlos, esperan recibir a cambio algún tipo de remuneración, usualmente un pago previamente pactado a una tasa de interés, puesto que estas personas e instituciones sacrifican el beneficio de usar sus recursos para cubrir deudas o efectuar pagos para consumo.

Aquellos que reciben los recursos, necesitan más de lo que tienen a corto plazo para realizar algún proyecto o pagar deudas, se comprometen a brindar el incentivo antes mencionado a cambio de obtener de inmediato los recursos necesarios.

Se puede pensar entonces que la tarea primordial de cualquier sistema financiero es relacionar las necesidades de los ahorradores o acreedores con las de los deudores de tal forma que circulen los recursos financieros y en donde estará de por medio una tasa de interés [Reséndiz Puente, 2014].

Un buen sistema financiero debe de proporcionar 6 funciones básicas a los usuarios [Bodie and Merton, 1998]:

- Compensación y liquidación de pagos: Esto facilita el intercambio de bienes, servicios y activos.

- Combinación de recursos y subdivisión de acciones: Permite la incorporación de las personas en el sistema financiero al dividir el precio de las acciones para que cada individuo adquiera una parte de la acción o poder combinar los recursos para adquirir los activos.

- Transferencia de recursos a través del tiempo y el espacio: El sistema financiero

permite transferir recursos a través del tiempo, entre regiones geográficas y entre las diferentes industrias del mercado.

- La gestión del riesgo: Permite la gestión y administración del riesgo para así controlarlo y evitar pérdidas.

- Proporcionar información: El sistema financiero debe brindar la información necesaria sobre los precios de manera que los agentes puedan tomar las mejores decisiones en varios sectores de la economía.

- Manejo de problemas de incentivos: Estos problemas ocurren cuando en un contrato financiero una de las partes posee información que la otra parte no. En este caso el sistema financiero debe proporcionar formas de lidiar e incluso mitigar estos problemas.

El Sistema Financiero Mexicano al concebirse como un buen sistema financiero debe por tanto cumplir al menos con estas 6 funciones.

Habiendo explicado brevemente sobre los Sistemas Financieros y particularmente el mexicano, es importante hablar sobre los mercados financieros, que son parte del sistema financiero.

2.3. Mercados financieros

Los mercados financieros son el espacio, ya sea físico o virtual, compuesto por participantes e instituciones, donde estos pueden ser compradores o vendedores y donde los participantes pueden ser instituciones financieras, inversionistas individuales y corporaciones.

En estos espacios se negocian instrumentos financieros, ya sean bonos, acciones, activos subyacentes, entre otros y los precios a los cuales se ofertan los productos dependen de la oferta y demanda en el mercado [Chan et al., 2019].

Entre las funciones principales de los mercados financieros encontramos [Banxico educa, 2015]

- Posibilitar el contacto entre los participantes del mercado en la negociación.
- Establecer los precios de los productos de acuerdo a su oferta y demanda.
- Disminuir los costos de intermediación con el fin de una mayor circulación de los productos.
- Administrar los flujos de liquidez ya sean de productos o de un mercado a otro mercado.

Podemos clasificar a los mercados financieros de acuerdo a las funciones que desempeñan, entre los principales mercados tenemos:

- Mercado de capitales: Proporciona financiación de las empresas mediante la emisión de acciones y deuda.
- Mercado de divisas: Donde se negocian las divisas.
- Mercado de productos básicos o *commodities*: Se comercializan productos agrícolas, energéticos, metales, entre otros.
- Mercado de derivados: Se encuentra la comercialización de aquellos productos financieros cuyos valores dependen directamente del precio de uno o más activos financieros, a estos productos se les conoce como productos derivados y entre los más conocidos se encuentran los futuros, *swaps* y opciones.

Asimismo podemos clasificar a los mercados financieros de acuerdo a otro criterio en dos grupos:

- Mercado primario: Donde se ofertan por primera vez las nuevas emisiones.
- Mercado secundario: Donde ocurren las operaciones posteriores a la salida de las emisiones en la bolsa entre los participantes del mercado.

Esta última forma de clasificarlos permite explicar de mejor manera cómo se ofertan los valores del mercado. Para esto la empresa acude a los mercados primarios para

la venta de sus valores, el precio se fija por la empresa y la casa de bolsa. Después, la empresa emisora recibe los recursos económicos obtenidos por la venta de los valores que adquieren los inversionistas. Una vez que estos valores o acciones son comprados, pueden volver a ser vendidos por los inversionistas en el mercado secundario donde habrán otros compradores de las acciones y ahora los precios se decidirán entre los compradores y vendedores.

Para fines del trabajo, se profundiza en el mercado de capitales, y específicamente en el mercado de acciones, pues es dentro de este mercado donde se emiten y negocian las acciones de las empresas, con las cuales se trabajan con el fin de predecirlas.

2.3.1. Mercado de capitales

Como se mencionó anteriormente, el mercado de capitales es aquel lugar en donde las empresas encuentran financiamiento y consiste en 2 mercados, el de deuda y el de acciones

- Mercado de deuda o mercado de renta fija: En este mercado participan los gobiernos estatales, locales y el gobierno federal, así como las empresas paraestatales y privadas. Aquí se negocian los productos gubernamentales y de renta fija como papel comercial o bonos; para financiamiento a corto (plazos menores a un año) o largo plazo (plazos mayores a un año).

- Mercado de acciones o mercado de valores: Aquí se negocian las acciones de las empresas que cotizan en la bolsa. En este mercado la fluctuación de los precios tiene una varianza más grande que los instrumentos del mercado de deuda y por lo tanto las inversiones en el mercado accionario son consideradas muy riesgosas. El valor de la acción refleja la situación de la empresa, así como las ganancias futuras y los pagos de dividendos esperados.

En México también encontramos los mercados de deuda y de acciones. Se participa en el mercado de deuda cuando se invierte en Cetes Directo o cuando se tienen

instrumentos de renta fija. Para participar en el mercado de valores se debe invertir en la bolsa de valores, en México existen dos bolsas de valores, la Bolsa Mexicana de Valores (BMV) y la Bolsa Institucional de Valores (BIVA) [García Padilla, 2018].

Después de hablar sobre el Sistema Financiero y los Mercados Financieros se mencionan algunos conceptos que servirán de base para la realización y un mejor entendimiento del trabajo.

2.3.2. Índices accionarios y acciones

Una de las maneras en la que una empresa obtiene dinero para financiar y pagar sus deudas es mediante la emisión de capital nuevo, también llamado acciones. Estas acciones son títulos representativos del capital invertido en la compañía y su valor representa una fracción del capital social de la compañía. Se pueden obtener en los mercados organizados, específicamente en los mercados de acciones y en México forma parte de la Bolsa Mexicana de Valores (BMV). Estos mercados están abiertos para todos los inversionistas; un inversionista es cualquier persona o entidad que cumpla con ciertos requisitos previamente establecidos en los mercados.

Por un lado, el poseer acciones representa un riesgo ya que la empresa solo paga a los inversionistas si la empresa genera utilidades. Por otro lado, el riesgo es menor para las empresas que emiten sus acciones, pues no pagarán si no tienen utilidades.

Asimismo tenemos los índices accionarios, que son una medición del comportamiento y desempeño de un conjunto de activos que determinan uno o varios sectores o industrias particulares de la economía de un país. Los índices accionarios tienen un papel importante en los mercados accionarios pues representan el desempeño, comportamiento y estabilidad de estos mercados, incluso sirven para analizar la economía de un país ya que en estos índices están plasmadas las épocas de crecimiento así como las crisis económicas.

En México tenemos el S&P/BMV IPC (Índice de Precios y Cotizaciones) de la Bola Mexicana de Valores, este es un indicador que refleja el comportamiento del

mercado accionario mexicano, ya sea en su conjunto o como diferentes grupos de empresas que comparten características en común [Bolsa Mexicana de Valores, 2021].

En la gráfica 2.2 podemos observar el comportamiento del S&P/BMV IPC entre el primero de enero de 2007 y el 31 de diciembre de 2010. En este histórico del índice se observa la crisis económica de México en 2008 y 2009, esta crisis se puede ver al momento de la caída del IPC. Es por esto que los índices accionarios pueden mostrar el comportamiento de la economía, en este caso la mexicana.

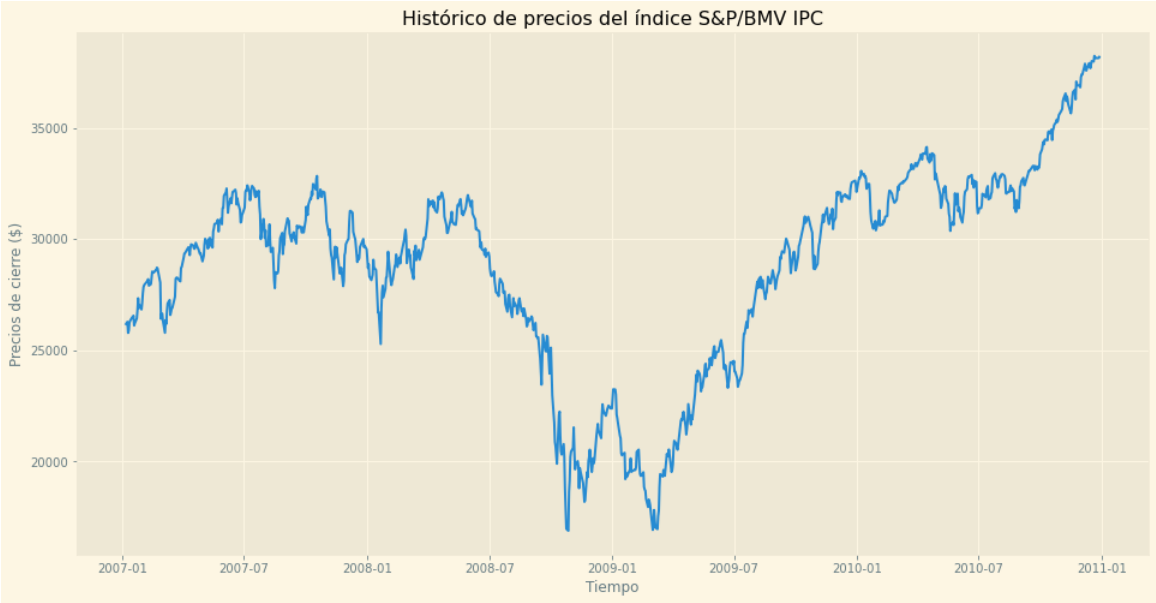


Figura 2.2: Serie de tiempo del precio de cierre histórico del S&P/BMV IPC entre los años 2007 y 2010.

Capítulo 3

Aprendizaje profundo y Redes Neuronales

El fin del presente trabajo es aplicar las Redes Neuronales Recurrentes o *Recurrent Neural Network* (RNN) a los precios de índices y acciones con el fin de predecirlos, analizar los incrementos y decrementos en estos precios y plantear una estrategia de compra-venta de las acciones.

Para hablar de las redes neuronales recurrentes debemos comenzar con los conceptos básicos de aprendizaje automático (*machine learning*), profundizar en el aprendizaje profundo (*deep learning*) y específicamente en las RNN, ya que el *deep learning* es un subconjunto de herramientas de *machine learning*.

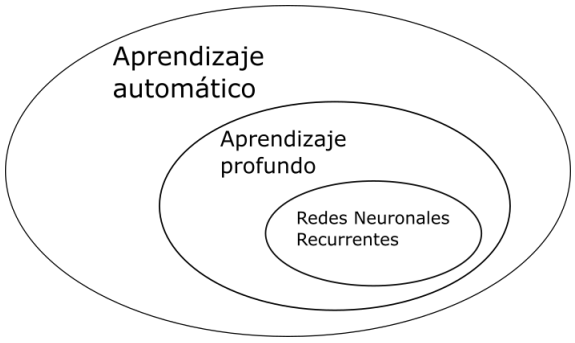


Figura 3.1: El aprendizaje automático es un gran conjunto de técnicas y algoritmos para la solución de problemas, un subconjunto de estos algoritmos es llamado aprendizaje profundo y dentro de este conjunto tenemos las RNN.

3.1. Una introducción al aprendizaje automático

En los últimos años, gracias a los avances en tecnología y computación, se han logrado implementar algoritmos y métodos estadísticos y computacionales para la solución de problemas que de otro modo serían difíciles de solucionar, además se accedió más fácilmente a estos algoritmos gracias a las capacidades de las computadoras actuales. Estos algoritmos comparten ciertas características y el conjunto de estos se llama aprendizaje automático.

El aprendizaje automático o *machine learning* surge como una forma de programar algoritmos en la que estos aprendan por sí mismos a solucionar problemas específicos, obteniendo así modelos de *machine learning*. Mientras que la forma clásica de programar consistía en realizar algoritmos donde se les pasaran datos y reglas para aplicarlas a los datos, dando como resultado las respuestas, en el aprendizaje automático se entrena al algoritmo ingresando los datos de base junto con los resultados esperados de ingresar dichos datos, obteniendo como respuesta del algoritmo un modelo que consiste en reglas y especificaciones para así aplicar estas reglas a nuevos datos [Chollet, 2017].

Por ejemplo, en la programación clásica podríamos tener un algoritmo que tome como datos 2 números y como regla el sumarlos, dando como respuesta la suma de estos números. Por otra parte, un algoritmo de *machine learning* tomaría los 2 números como datos y como respuesta el número obtenido de sumarlos, y el algoritmo devolvería un modelo que sepa que la regla es sumar.

En el siguiente esquema se puede observar de manera simplificada la diferencia entre el paradigma clásico de programación y el de *machine learning* usando el ejemplo mencionado anteriormente.

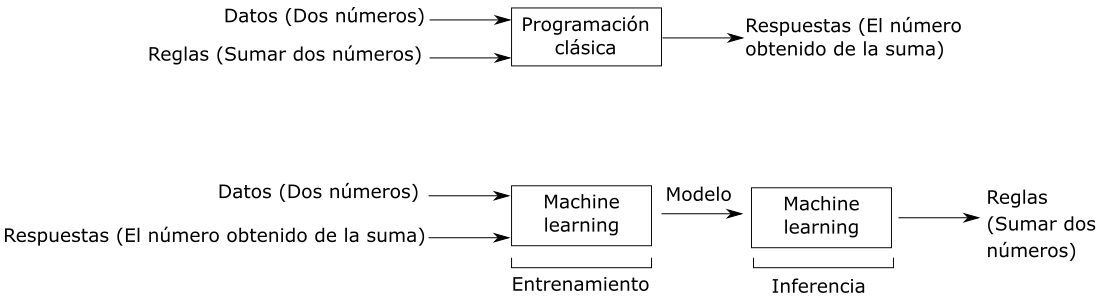


Figura 3.2: Diferencias entre los algoritmos de la programación clásica y los de aprendizaje automático.

Habiendo visto de manera general el funcionamiento de los algoritmos de aprendizaje automático podemos hablar más detalladamente sobre estos algoritmos. Como se ha mencionado antes, los algoritmos de *machine learning* en vez de ser programados estos aprenden por sí mismos a solucionar los problemas usando datos y es justo esta capacidad de aprender la que caracteriza a estos algoritmos.

Un algoritmo de *machine learning* aprende sobre una experiencia E con respecto a una tarea T y una medida de desempeño P cuando el desempeño al realizar la tarea T, medido por P mejora con la experiencia E [Mitchell, 1997].

Para comprender de mejor manera la definición anterior se hablará sobre los conceptos de T, P y E.

- Tarea (T): Las tareas de los algoritmos de *machine learning* son las acciones que deben realizar para resolver un problema, usualmente estas tareas se relacionan con el cómo procesa los datos el algoritmo. Entre las tareas más comunes de los algoritmos de *machine learning* se encuentran clasificar, predecir, traducir, transcribir y estimar funciones de densidad o distribución de probabilidad.

- Medida de desempeño (P): Después de definir la tarea o tareas que el algoritmo debe realizar, es necesario tener una medida para la eficiencia del algoritmo al realizar las tareas, esto permitirá analizar si el algoritmo realiza un buen trabajo o si es necesario modificarlo para obtener un mejor desempeño. Esta medida de desempeño está en función de la tarea a realizar, las medidas de desempeño más

usadas son la precisión del modelo, las tasas de error, el error cuadrático medio y error medio absoluto.

- Experiencia (E): La experiencia sobre la cual va a aprender el algoritmo es el conjunto de datos con los cuales se entrenará al algoritmo. De acuerdo a la tarea que se realizará, los datos con los que se entrenará el algoritmo pueden ser números, palabras, imágenes, entre otros conjuntos de datos.

Para comprender de mejor manera los términos antes mencionados se tiene el siguiente ejemplo que se relaciona con la finalidad del presente trabajo:

La tarea T es la predicción de los valores futuros de los precios de acciones e índices, como medida de desempeño P puede ser el error cuadrático medio para medir en promedio cuanto se aleja la predicción del valor real y en el caso de la experiencia E se tiene los precios históricos de los índices y acciones.

3.1.1. Tipos de algoritmos

Dependiendo de las tareas y experiencias, podemos clasificar a los algoritmos en tres grupos [Dasgupta and Nath, 2016]:

- Aprendizaje supervisado: Los algoritmos de esta clase son aquellos en donde se tiene al menos una variable explicativa (X_i), así como al menos una variable respuesta (Y). Las variables explicativas son aquellas usadas para explicar, describir o predecir otra variable, mientras que la variable respuesta es la variable que se desea predecir, explicar o describir. Con estos algoritmos se construye una función que relaciona las variables explicativas con la de respuesta, de manera que el algoritmo aprenda a mapear los datos de entrada contenidos en las X_i al objetivo Y dado un conjunto de ejemplos (comúnmente recolectados y anotados por un humano). La gran mayoría de tareas que se realizan en el aprendizaje supervisado son de predecir, clasificar o explicar una relación entre las X_i y Y.

- Aprendizaje no supervisado: En esta clase se encuentran los algoritmos que necesitan a las variables explicativas X_i pero no a la variable respuesta (Y)

para aprender, estos algoritmos resuelven tareas concernientes al análisis, visualización, agrupamiento o transformación (como por ejemplo la reducción de dimensión) de los datos de entrada. Su principal función es el entendimiento de los datos para analizar la relación y correlación de los mismos. Dado que estos algoritmos permiten un mejor entendimiento de los datos, usualmente se usan antes de aplicar algún algoritmo de aprendizaje supervisado.

- Aprendizaje por refuerzo: A diferencia de los algoritmos anteriores, en el aprendizaje por refuerzo se tiene un agente que interactúa con el ambiente y recibe información de éste. El fin de estos algoritmos es hacer que, mediante recompensas o penalizaciones el agente aprenda a tomar las mejores decisiones que lo conduzcan a la solución de la tarea. Dado el concepto de estos algoritmos, se aplican en mayor medida a los videojuegos pero no están limitados a esta área pues conforme avanzan las investigaciones se encuentran más aplicaciones en la toma de decisiones de forma inteligente y eficiente.

Se puede tener un cuarto conjunto de algoritmos de aprendizaje automático llamado aprendizaje auto-supervisado, consiste en aprender mediante etiquetas, al igual que en el aprendizaje supervisado, con la diferencia que estos datos no son ingresados por un humano, sino que se generan a partir de los datos de entrada [Chollet, 2017]. Por ejemplo, al querer predecir el siguiente fotograma en un videojuego dados los fotogramas anteriores o predecir la siguiente palabra en un texto dadas las palabras anteriores, se tiene que los datos con los que aprende el algoritmo (los fotogramas o palabras anteriores) no son ingresados por un humano, sino que están previamente generados.

Después de presentar brevemente el funcionamiento, la definición y la clasificación de los algoritmos, se procede a mencionar de manera general los problemas que pueden ocurrir al entrenar un algoritmo con los datos introducidos para su aprendizaje, asimismo se explica la forma de evitar estos problemas al particionar los datos con los que se cuentan.

3.1.2. Partición de los datos: Entrenamiento, validación y prueba

Se sabe que los algoritmos de *machine learning* aprenden mediante los datos introducidos, dependiendo de la cantidad de datos el algoritmo tendrá más información para aprender y hacer las tareas necesarias para resolver el problema. Se podría pensar en entrenar al algoritmo con todos los datos con los que se cuentan, pero surge un gran problema, pues como se ha mencionado, la finalidad de los algoritmos es aplicar lo que han aprendido a solucionar las tareas teniendo datos distintos a los que se le dieron para aprender. Y si ocurre que el algoritmo se entrena con el total de datos disponibles, aprende muy bien sobre esos datos, pero al aplicarlo a datos distintos el algoritmo falla ocasionando que las tareas realizadas tengan errores.

Para solucionar el problema antes mencionado se divide el conjunto de datos en tres subconjuntos llamados comúnmente *train*, *validation* y *test* o conjuntos de entrenamiento, validación y prueba, respectivamente [Mitchell, 1997]. La finalidad de dividir los datos es permitir que el algoritmo aprenda lo suficiente sin que aprenda exactamente todos los patrones de los datos, puesto que esto ocasionaría, como se mencionó, que el algoritmo solo funcione para los datos con que se entrenó.

Aunque no existe una regla que nos indique exactamente en que proporción dividir el total de los datos en los tres subconjuntos, se tienen algunas consideraciones para realizar de la mejor manera la división de los datos.

La consideración más importante, como se puede ver gráficamente en la Figura 3.3, es asignar un mayor número de datos al conjunto de entrenamiento, que será el conjunto en el que el algoritmo aprenda los patrones más importantes. Posteriormente se trata de asignar una cantidad parecida de datos para los conjuntos de validación y prueba. A pesar de que no es una regla, usualmente si se cuentan con la suficiente cantidad de datos, el conjunto de entrenamiento consiste en un 70 % u 80 % de los datos totales, mientras que cada uno de los conjuntos de validación y prueba en un 15 %, cuando el conjunto de entrenamiento es del 70 %, o un 10 % en otro caso [Goodfellow et al., 2016].

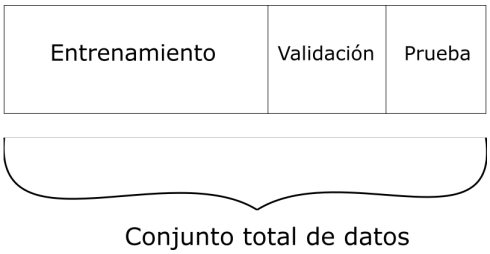


Figura 3.3: Representación de la división de los datos en los conjuntos de entrenamiento, prueba y validación.

Es importante mencionar ahora la función de los tres subconjuntos:

- Entrenamiento: Como se ha mencionado, con este subconjunto de datos se entrena al modelo para que aprenda solo los patrones más importantes.
- Validación: En este conjunto de datos se prueba el modelo obtenido con los datos de entrenamiento, si se tiene un mal ajuste o no se obtuvo el resultado esperado, se modifican los parámetros del algoritmo y se vuelve a probar el algoritmo hasta obtener los resultados deseados. Se debe tener especial cuidado con el ajuste de los parámetros, pues el modelo podría aprender todos los patrones y ajustarse muy bien a los datos de validación pero al momento de probarlo con otro conjunto de datos se podrían general errores.
- Prueba: Por último se tiene el conjunto con el cual, después de haber entrenado el modelo y ajustado los parámetros, se procede a probar el modelo para obtener los resultados finales. Con estos resultados se podrá analizar el modelo y su funcionalidad, para concluir si el modelo funciona para resolver las tareas y el problema.

Sin embargo, puede ocurrir que no se dispongan de los suficientes datos para dividirlos en los tres subconjuntos antes mencionados y que cada subconjunto tenga suficientes datos para entrenar, validar y probar el algoritmo. Ante este problema existen métodos de remuestreo, estos métodos usan distintas técnicas para generar, del mismo conjunto de datos, subconjuntos de datos con los cuales se logra entrenar el modelo así como validarlo.

Entre los métodos más usados se tienen los siguientes [James et al., 2017]:

- *Leave-One-Out Cross-Validation*: Este método consiste en tomar el dato que ocupa el lugar i, donde $i \in [1, n])$ con n el número total de datos, entrenar el modelo con los $n-1$ datos restantes y validar el modelo con el dato restante. Se repite esto para todos los valores de i, por cada iteración se obtiene el valor de la medida de desempeño P_i, y así la medida de desempeño del conjunto *test* será el promedio de los resultados de las iteraciones.

$$\frac{1}{n}\sum_{i=1}^{n} P_i$$

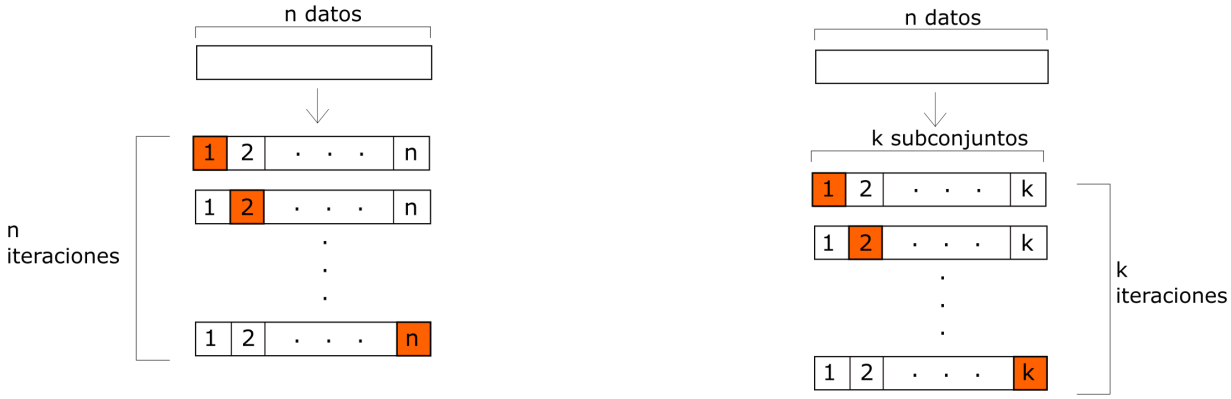
- *k-Fold Cross-Validation*: Parecido al método anterior, en este se dividen los datos en k subconjuntos de aproximadamente el mismo tamaño. Posteriormente se toma el conjunto en la posición i como conjunto de validación y los restantes $k-1$ conjuntos como datos de entrenamiento para el algoritmo. Se obtiene así una medida de desempeño P_i y repetimos el proceso iterando el valor de i de 1 a k. Volviendo a tener que el desempeño del conjunto *test* será

$$\frac{1}{n}\sum_{i=1}^{k} P_i$$

  En la Figura 3.4a se observa un esquema simple de cómo funciona el método *Leave-One-Out Cross Validation* con los n datos individuales, mientras que en la Figura 3.4b se observa el correspondiente a *k-Fold Cross Validation* agrupando los datos en k subconjuntos.

- *Bootstrap*: Finalmente en este método se elige un número m suficientemente grande que representará el número de muestras a usar, y se eligen al azar y con reemplazo n datos tomados de los n datos originales, de manera que cada una de las m muestras contenga n datos. Posteriormente el conjunto de entrenamiento estará comprendido por $m-l$ muestras, donde $l < m$, mientras que las l muestras restantes serán el conjunto de validación. A las muestras usadas para la validación se les llama *out-of-bag samples* o (OOB). Se proce-

de a calcular la medida de desempeño repitiendo el proceso $m - l$ de veces, uno por cada muestra del conjunto de entrenamiento, obteniendo así $m - l$ medidas de desempeño $(P_1, P_2, ..., P_{m-l})$ y cuya media será el desempeño en el conjunto de entrenamiento. Para el desempeño del conjunto de validación se procede de la misma manera, calculando la media de las l medidas de desempeño $(P_{m-l+1}, P_{m-l+2}, ..., P_m)$, una por cada muestra. En la Figura 3.5 podemos ver ejemplificado el método, tomando n datos y realizando m iteraciones.



(a) LOO CV (b) k-fold CV

Figura 3.4: Métodos de remuestreo: LOO CV (izquierda) y K-fold CV (derecha). En ambos métodos el algoritmo se valida con los datos en el recuadro naranja y se entrenan con los datos restantes. Tomado de *An Introduction to Statistical Learning: With Applications in R (pp. 179,181) por G James, D. Witten, T. Hastie, R Tibshirani, 2017. Springer.*

Existen otros métodos de remuestreo y básicamente el procedimiento de los más comunes es entrenar y validar en distintos subconjuntos de los datos originales obtenidos mediante particiones del conjunto total o elecciones al azar de estos datos.

Es importante mencionar que dependiendo de los datos que se tengan así como de la tarea a realizarse, será el método de remuestreo a emplearse, ya que por ejemplo, si se tienen datos ordenados por fechas y se quiere predecir un valor en el futuro, al seleccionar los datos aleatoriamente con reemplazo podría ocasionar que se entrenara con datos del futuro y se validara con algunos del pasado causando huecos en el tiempo. Asimismo si se tienen datos para clasificar (supóngase que se tienen 4 clases

distintas para clasificar los datos: A, B, C, D) y se dividen de cierta manera los datos, se puede entrenar el modelo con todos los datos de alguna clase (supóngase B), se valide con datos de otros conjuntos (A y D) y se pruebe el modelo con los datos de los conjuntos restantes (en este caso C) haciendo que el modelo falle al clasificar. Por último, se puede tener redundancia en los datos, es decir, que se tengan datos repetidos, lo que provocaría que al dividir los datos en los conjuntos *train*, *validation* y *test*, algunos datos caigan en al menos 2 de estos conjuntos, por lo que se podría estar entrenando el modelo y probando con datos que pertenecen al mismo conjunto, para esto es necesario que los conjuntos sean disjuntos.

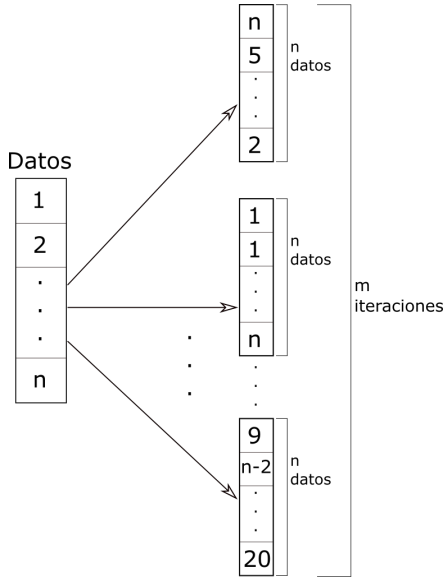


Figura 3.5: Método de remuestreo: Bootstrap, al seleccionar los datos de manera aleatoria y con reemplazo, puede que en un subconjunto de n datos, hayan datos repetidos.

3.1.3. Ajuste del modelo: sub-ajustar, ajustar o sobre-ajustar

Como se ha comentado anteriormente, el fin de los algoritmos de aprendizaje automático es aprender ciertos patrones de los datos para poder aplicarlo a nuevos datos y resolver las tareas y problemas. Así el modelo es entrenado para ajustar los parámetros y obtener los resultados esperados, y es justo en este procedimiento de ajuste donde se puede encontrar con un gran problema, ya que el modelo podría no ajustarse bien a los datos debido a diferentes factores, o incluso ajustarse extremadamente

bien a los datos de entrenamiento ocasionando que al aplicar el modelo a un conjunto distinto de datos este falle. En los dos casos mencionados el modelo tendrá un error muy grande al realizar las tareas.

Este problema se puede reescribir diciendo que el algoritmo debe encontrar un equilibrio entre la optimización y la generalización, donde [Chollet, 2017] :

- La optimización hace referencia al ajuste del modelo con los datos de entrenamiento. Entre mejor sea el ajuste, menor será la tasa de error y mayor el rendimiento del modelo.

- La generalización se refiere a la capacidad que tenga el modelo entrenado de resolver la tarea para la que se programó, ahora usando datos distintos a los de entrenamiento. Si el modelo se desempeña bien con datos que no había visto antes se tiene una buena generalización.

Lo deseado para los algoritmos es que generalicen bien, pero esto no se puede controlar, solo se tiene control del entrenamiento del algoritmo y el ajuste del modelo en los datos de entrenamiento. Como los parámetros del modelo no se ajustan a lo largo del tiempo sino que se ajustan de acuerdo a los datos de entrenamiento y validación, el error del algoritmo al realizar las tareas en los datos de prueba son en general mayores a los de entrenamiento y validación.

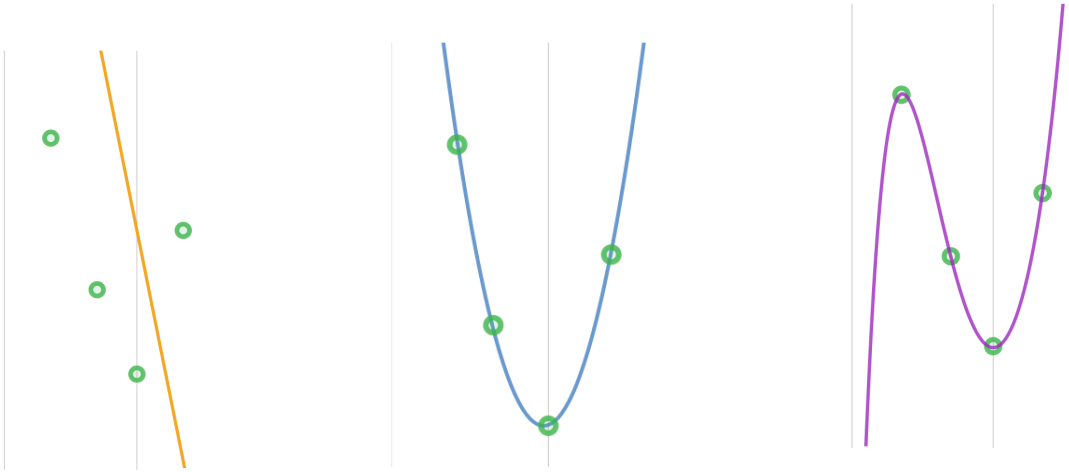
Por lo tanto para ajustar un modelo que dé buenos resultados se debe de minimizar en lo posible el error de entrenamiento y validación, al mismo tiempo que se disminuye la distancia entre el error de entrenamiento y el error de prueba.

Cuando se ajusta un modelo, y no se es capaz de obtener un error de entrenamiento suficientemente pequeño se tiene un problema de sub-ajuste, en otras palabras, se presenta este error cuando el modelo obtenido no tiene buena optimización. En este caso no se han ajustado bien los parámetros del modelo y por esto el modelo tiene una deficiencia en el ajuste.

Por otro lado si el error de prueba se aleja mucho del de entrenamiento tenemos sobre-ajuste, en este caso el modelo no generaliza bien debido a que el modelo aprende

los patrones y se ajusta muy bien a los datos de entrenamiento, ocasionando que falle al realizar las tareas necesarias con los datos de prueba.

En la Figura 3.6a se puede ver un ejemplo de sub-ajuste donde el modelo es incapaz de ajustarse a los puntos. Por otro lado en 3.6c el modelo ajusta demasiado bien a los datos, aprendiendo sus patrones y tendencias ocasionando un sobre-ajuste.



(a) Sub-ajuste (b) Buen ajuste (c) Sobre-ajuste

Figura 3.6: Ajustes de un modelo. En el lado izquierdo el modelo intenta ajustar a los datos con una recta ocasionando que tenga un mal ajuste, en el lado derecho ajusta muy bien a los datos usando un polinomio de grado 7, esto provocará que al intentar usar este modelo con otros datos ajustará mal. En el centro se tiene un buen modelo ya que ajusta bien a los datos sin sufrir de sub-ajuste ni sobre-ajuste, representando bien la curva, algo que la función lineal no logró, al mismo tiempo que no necesita ajustar polinomios de un grado alto.

Con esta pequeña introducción a los puntos principales del aprendizaje automático se ha cubierto una parte del esquema de la Figura 3.1, se procede ahora a hablar del aprendizaje profundo, que permite introducir los conceptos de redes neuronales recurrentes.

3.2. Aprendizaje profundo

El aprendizaje profundo o *deep learning* es una clase de algoritmos de *machine learning* basados en redes neuronales que facilitan la solución de problemas. Para

ahondar en esta clase de algoritmos se necesita primero hablar de las redes neuronales, sus funciones y aplicaciones a los problemas.

3.2.1. Redes neuronales

Las redes neuronales toman inspiración en las neuronas del cerebro del ser humano y las conexiones que tienen entre sí. Estas redes consisten en neuronas artificiales interconectadas entre sí que se usan para crear modelos de aprendizaje [Rebala et al., 2019].

Las redes neuronales se pueden representar gráficamente de diversas maneras, una de las más básicas es la siguiente:

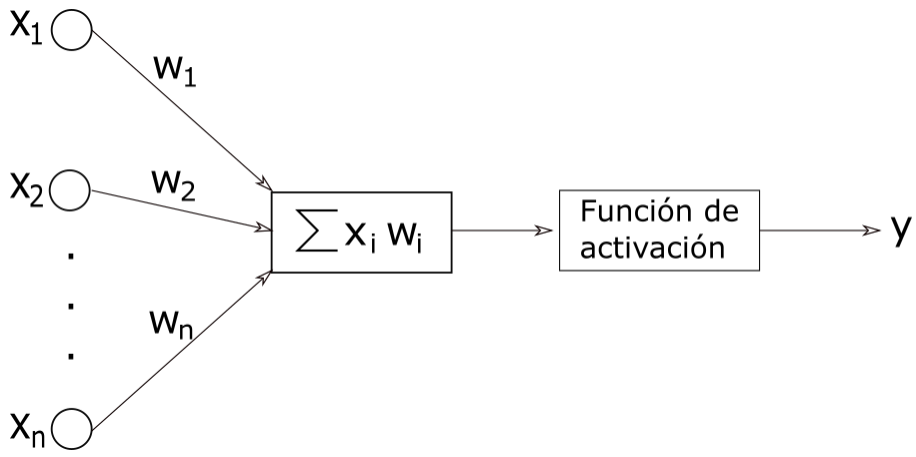


Figura 3.7: Modelo McCulloch–Pitts: Red neuronal compuesta por los datos de entrada x_i , los pesos w_i , la función de activación y la salida del modelo $\mathbf{y}$.

Las neuronas, representadas por nodos en las redes neuronales, procesan la información del modelo, ya sea información que esta entrando por primera vez o la información de salida de otra neurona. Una vez que las neuronas procesan la información, se multiplica por los pesos asignados a cada conexión neuronal y se suman los resultados de estas multiplicaciones, estos pesos son números y permiten mostrarle a la red la importancia o intensidad de la información suministrada, al finalizar el proceso completo de la red neuronal, se obtiene una función de pérdida que está relacionada con los pesos, para disminuir la pérdida de la red, es necesario modificar los pesos. Después de ingresar a las neuronas los datos, se obtiene una salida corres-

pondiente a una clase de función conocida como función de activación, esta función toma los datos de entrada y los mapea de forma lineal o no lineal a los de salida.

En el esquema de la Figura 3.7 se muestra a grandes rasgos el proceso que realiza una red, comenzando con los datos de entrada, las neuronas o nodos de la red, los pesos, un paso en donde se calcula la relevancia de los datos (multiplicando los datos por sus pesos y sumando estos productos), este resultado es la entrada de una función de activación para finalmente obtener el dato de salida de la red.

Función de activación.

La función de activación como su nombre lo indica, de acuerdo al valor que produzca, activará o no la neurona para el procesamiento de los datos. Existen muchas funciones de activación que se usan de acuerdo a los datos que se tengan y el problema a resolver, entre las funciones más comunes encontramos [Siddhart Sharma and Anidhya, 2020]:

- *Binary Step Function*: Esta es la función más básica y se usa cuando se va a clasificar los datos en solo 2 clases. Un ejemplo de esta función puede ser.

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

- Función de activación lineal: En este caso la función es directamente proporcional al dato de entrada. Se usa principalmente para realizar tareas sencillas y para interpretar los datos. Una función lineal puede escribirse como $f(x) = ax \quad con \quad a \in \mathbb{R}$.

- Función de activación sigmoide: Es una función no lineal y es la más utilizada, transforma los datos para que estén entre el 0 y el 1. Se puede definir como

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Función tanh o tangente hiperbólica: Es similar a la función anterior, con la

diferencia que ahora los datos estarán en el intervalo del -1 al 1. Para definir a esta función se puede hacer en base a la función sigmoide pues se puede escribir

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 = 2 * sigmoide(2x) - 1$$

- Función $ReLU$: Se tiene otra función no lineal que también se usa frecuentemente, la ventaja más grande de esta función es el hecho de que no todas las neuronas están activadas al mismo tiempo. La neurona se desactivará cuando la salida de la función sea cero. Se puede escribir de la siguiente manera

$$f(x) = max\{0, x\}$$

- Función $SoftMax$: Es una combinación de funciones sigmoideas y se puede interpretar como una probabilidad. Esta función se usa cuando se tienen diferentes clases y un vector con K entradas, $\mathbf{x} = (x_1, x_2, ..., x_K)$. Con el fin de predecir la clase a la que pertenece la muestra se realiza la siguiente operación para cada entrada del vector:

$$f(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

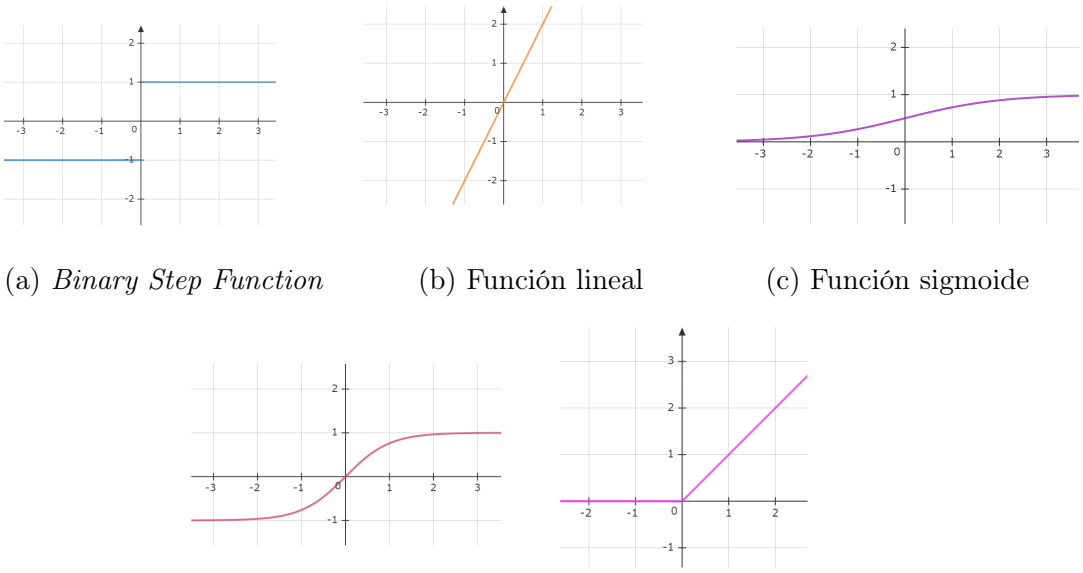
  Obteniendo un nuevo vector con K entradas en donde la entrada j representa la probabilidad de que el dato x_j pertenezca a la clase a la cual se desea clasificar.

Es importante mencionar que la elección de una buena función de activación es muy importante para que la red produzca buenos resultados. Para la elección de una función es importante considerar la tarea que se realizará, así como ciertas ventajas que tiene cada función, como por ejemplo

- La función sigmoide es una buena opción cuando se quiere clasificar.
- La función ReLU en la mayoría de los casos es una buena opción para empezar a desarrollar la red, es una de las funciones más usadas y en general se comporta mejor que otras.

A pesar de esto no existe una regla exacta para elegir la mejor función de activación para cada problema y usualmente se prueba con distintas funciones para elegir la que dé el menor error.

En las siguientes gráficas de la Figura 3.8 se observan las distintas funciones de activación mencionadas con anterioridad, con estas se puede tener una mejor idea de cómo mapean los datos de salida de las neuronas. Usualmente la salida de estas funciones están entre el 0 y el 1 o entre el -1 y el 1.



(a) *Binary Step Function* (b) Función lineal (c) Función sigmoide

(d) Función tanh (e) Función *ReLU*

Figura 3.8: Gráficas de las diferentes funciones de activación generadas en *Mathcha*.

Características de las redes neuronales

Algunas características importantes de estos modelos son su arquitectura, capacidades, propiedades y aplicaciones [Du and Swamy, 2014]. Conocerlas permitirá un mayor entendimiento de las redes, el alcance que estos modelos tienen y la diversidad que existen de acuerdo a las tareas que se requieran realizar.

- Arquitectura: Concentrándose por ahora en las neuronas, los pesos y las conexiones entre ellas; para definir la arquitectura de las redes, se considera a la matriz W donde en la entrada $i - j$, representada por w_{ij}, se encuentra el peso

de la conexión entre la neurona o nodo i y la neurona j. Así la arquitectura de una red neuronal es representada por la matriz W, y al hacer $w_{ij} = 0$ para alguna i y j, se tienen diferentes arquitecturas. Las arquitecturas más populares se enlistan a continuación y una representación gráfica se observa en la Figura 3.9:

  - Redes neuronales prealimentadas con capas completamente conectadas (*Fully connected layered feedforward networks*).
  - *Lattice network* (Red neuronal con cuadrículas)
  - Redes neuronales recurrentes (*Recurrent neural network*).
  - Redes celulares.

- Capacidades: Las redes neuronales, como algoritmos del aprendizaje automático, son concebidas para resolver tareas y problemas que en principio serían difíciles o casi imposibles de resolver con la programación clásica, por esto entre sus principales capacidades tenemos:

  - Aproximación de funciones: Las redes neuronales son capaces de aproximar funciones, ya sean lineales o no lineales para modelar los problemas y así dar una solución.
  - Clasificación: Como el nombre lo sugiere, consiste en la capacidad de clasificar la información en diferentes grupos de acuerdo a ciertas propiedades de la información.
  - Memoria asociativa y heteroasociativa: El primer tipo de memoria permite a las neuronas tomar información, reconocer patrones y asociar la información a estos patrones. El segundo tipo de memoria consiste en un mapeo entre dos conjuntos, uno de patrones de entrada y el otro de patrones de salida, de manera que al ingresar los patrones de entrada con alguna alteración, la red neuronal sea capaz de asociarlos a los patrones de salida [Hassoun, 1989]. Ciertas redes neuronales como las redes recurrentes tienen la capacidad de almacenar información mediante conexiones entre las

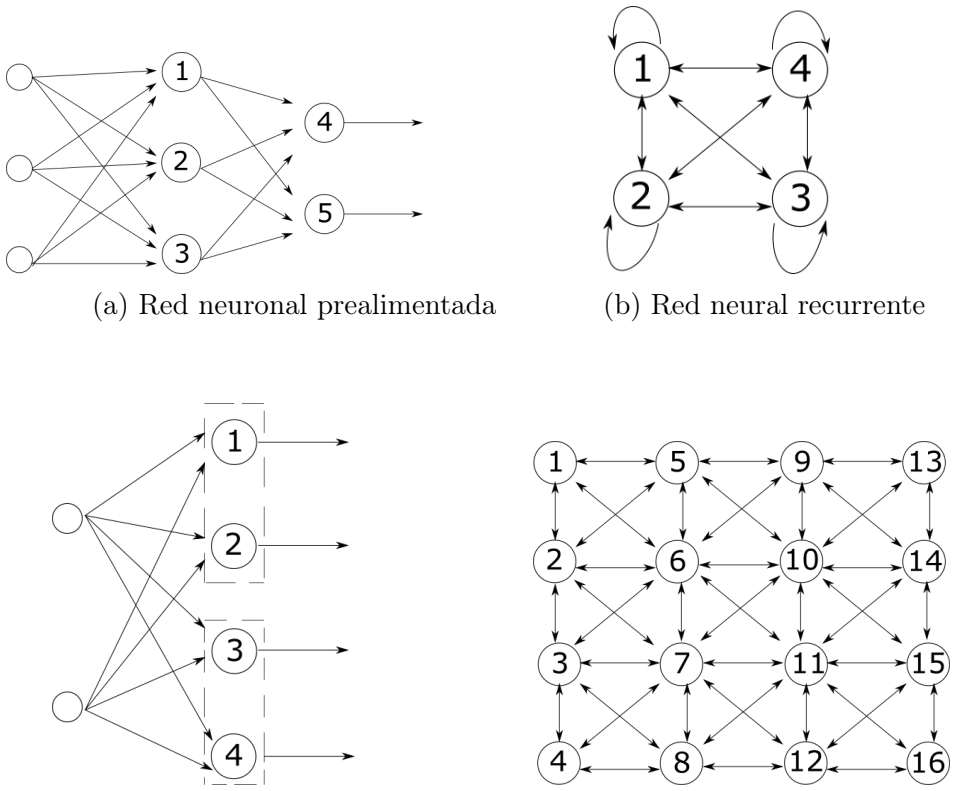
neuronas consigo mismas, de manera que los datos de entrada son procesados por la misma neurona un determinado número de iteraciones. Estas conexiones de retroalimentación permiten que los redes propaguen datos de eventos anteriores a los pasos de procesamiento actuales haciendo que las redes construyan una memoria de eventos de series de tiempo.

- Propiedades: Las redes neuronales tienen muchas propiedades, algunas dependen de su arquitectura, pero en general las propiedades más comunes son:

  - Aprendizaje adaptativo: Modificando los parámetros de la red, esta puede adaptarse a la información y al ambiente.
  - Generalización: Como se ha mencionado, al entrenar un algoritmo se espera que este aprenda ciertos patrones y que además pueda resolver las tareas usando información diferente a la que se usó para entrenarlo, es justo esta propiedad la que se le llama generalización.
  - Robustez y tolerancia al fallo: La primera propiedad se refiere a que la red neuronal puede manejar información imprecisa, difusa, con ruido y probabilística sin necesidad de darle un tratamiento especial a la información.

    Como la red distribuye la información entre sus neuronas, si ocurre que alguna neurona o conexión falla, la neurona puede continuar su proceso sin que su eficiencia sea degradada significativamente, a esto se refiere la propiedad de tolerancia al fallo.

- Aplicaciones: Como casi cualquier algoritmo de *machine learning*, las redes neuronales tienen muchas aplicaciones en diferentes áreas para resolver problemas muy variados como pueden ser modelación, clasificación, reconocimiento de patrones, análisis de datos, procesamiento de información, entre otros.



(a) Red neuronal prealimentada (b) Red neural recurrente

(c) *Lattice network* (d) Red celular

Figura 3.9: Arquitecturas de las Redes Neuronales: Los círculos grandes y con números representan neuronas y los círculos pequeños son los nodos de entrada de información. Como se puede ver en (a) todas las neuronas están conectadas con los nodos anteriores; en el caso (c) tenemos una red neuronal con 2 cuadrículas representada por los 2 rectángulos. En la red neuronal recurrente cada neurona está conectada con las demás y consigo misma, y la diferencia con la red celular es que las neuronas no se conectan consigo mismas.

Aprendizaje de la red: Ajuste de los pesos

Como se ha visto anteriormente, los algoritmos de aprendizaje automático se diferencian de la programación clásica en que estos aprenden de la información suministrada, para esto todo algoritmo es entrenado para que aprenda a solucionar las tareas. En el caso de las redes neuronales, estas son entrenadas y aprenden mediante los pesos, por lo que estos tienen un papel relevante no solo para las arquitecturas correspondientes, sino también para el procesamiento de la información pues de acuerdo a los pesos la información tendrá mayor o menor importancia. Como se puede ver en el esquema de la Figura 3.2 una parte fundamental de los algoritmos de *machine*

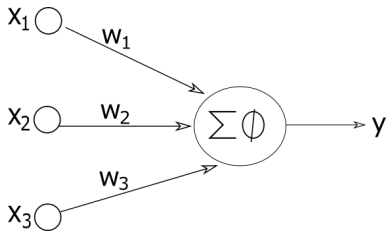
learning es la información que se le suministra, debido a esto es importante asignar de la mejor manera posible los pesos de las conexiones entre neuronas para que la red pueda aprender de manera óptima y se logre disminuir el error que se tenga al realizar las tareas.

Existen muchas formas de seleccionar los pesos y obtener buenos resultados, para comprender mejor la manera de proceder es necesario primero introducir los conceptos de perceptrón y capas en las redes neuronales.

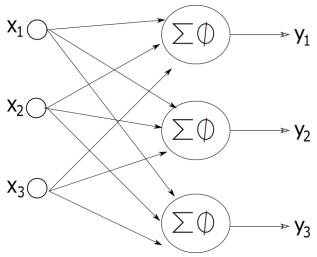
Un perceptrón es una neurona que recibe los datos y los pesos y de acuerdo a una función de activación procesa los datos devolviendo una variable *booleana* [Mitchell, 1997]. En el esquema de la Figura 3.7 se tiene una red neuronal con un solo perceptrón, que es el proceso realizado por la función activadora al procesar la operación $\sum_{j=1}^{n} x_j w_j$. Una red neuronal puede estar compuesta por más de un perceptrón.

Las capas en las redes se pueden dividir en capas de entrada, de salida o las capas ocultas. Una capa de entrada es el conjunto de datos que entran en el algoritmo, y de manera similar la capa de salida es el conjunto de salidas del algoritmo, ya que las redes neuronales pueden tener más de una salida; por último están las capas ocultas, que consisten en las neuronas que procesan la información de entrada para devolver datos procesados.

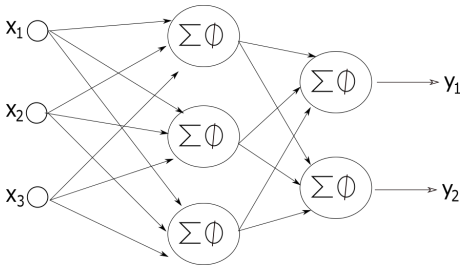
Si la red neuronal consta de un solo perceptrón se le llama *One-Neuron Perceptron*, si tiene más de un perceptrón en una sola capa oculta se le denomina *Single-Layer Perceptron*, por último si la configuración de la red consiste en más de un percetrón y más de una capa oculta es llamada *Multilayer Perceptrons* [Du and Swamy, 2014]. En el siguiente esquema se pueden ver gráficamente algunas diferencias entre las redes neuronales de acuerdo al número de perceptrones y las capas ocultas. Denotamos a cada perceptrón con el símbolo $\sum \phi$ donde $\sum$ es la suma de los productos de los datos ingresados por sus pesos $(x_j w_j)$ y ϕ es la función de activación.



(a) Red neuronal con un perceptrón de una capa oculta.



(b) Red neural multiperceptrón de una capa oculta.



(c) Red neuronal multiperceptrón con varias capas ocultas.

Figura 3.10: Redes neuronales con diferente configuración de capas y perceptrones. En (a) se puede ver el caso más simple que se mencionó con anterioridad, se tiene una capa de entrada, una de salida y un solo perceptrón que procesa la información. Para (b) la configuración de la red cambia agregando 3 perceptrones en una sola capa oculta, dando como resultado 3 valores de salida, uno por cada perceptrón. Finalmente en (c) se tiene dos capas ocultas, una con 3 perceptrones y la otra con 2, la información que entra en la capa de entrada es enviada a la primera capa oculta para ser procesada, después es procesada por la siguiente capa que finalmente devuelve 2 resultados en la capa de salida.

Una vez definidos estos conceptos se procede a introducir los métodos de *back-propagation* y descenso por gradiente (o *gradient descent*) que permitirán obtener los pesos de las conexiones. Empecemos viendo el algoritmo de descenso por gradiente, que posteriormente usa el método de *backpropagation*.

Descenso por gradiente

Este es un algoritmo de optimización, llamado en inglés *Gradient descent*, es uno de los más populares y usado en el caso de las redes neuronales. De manera general el descenso por gradiente consiste en minimizar una función objetivo $F(\theta)$ parame-

trizada por sus parámetros, denotando por θ al conjunto de todos los parámetros, para realizar esto se actualizan los parámetros en dirección contraria al gradiente de la función, es decir, $\nabla_\theta F(\theta)$ [Goodfellow et al., 2016]. Usualmente la función $F(\theta)$ se considera como la función de costo, es por esto que se requiere minimizar y así obtener una solución eficiente.

En la gráfica 3.11 se ve un ejemplo de descenso por gradiente, en donde al calcular el gradiente, se obtiene una dirección y de acuerdo al algoritmo, se procede a modificar o actualizar los parámetros de la función en dirección contraria al gradiente para obtener el mínimo valor posible que en este caso es cuando la función F(x) vale cero.

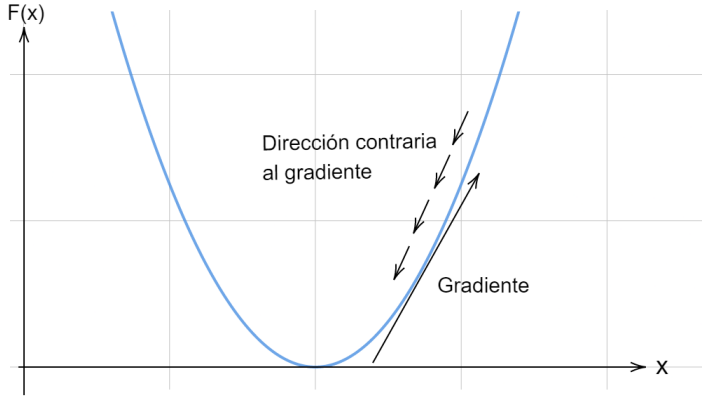


Figura 3.11: Descenso por gradiente.

Este algoritmo general tiene variantes que se usan de acuerdo al problema a resolver y que tiene sus ventajas y desventajas, entre los más conocidos están: Descenso por gradiente empleando *batch* (*Batch gradient descent*), Descenso por gradiente estocástico (*Stochastic gradient descent*) , Descenso por gradiente empleando *mini-batch* (*Mini-batch gradient descent*)[Ruder, 2017].

- Descenso por gradiente empleando *batch* : Esta es la variante más básica de las mencionadas anteriormente, consiste en calcular el gradiente de la función de costos de toda la base de datos con respecto a los parámetros y mediante una tasa de aprendizaje η actualiza los parámetros de la siguiente manera:

$$\theta = \theta - \eta \cdot \nabla_\theta F(\theta)$$

Como se debe calcular el gradiente para toda la base de datos para realizar una sola actualización, este algoritmo no es recomendable para conjunto de datos grandes.

- Descenso por gradiente estocástico: Para solucionar el problema del cálculo de gradiente para todos los datos, surge este algoritmo que consiste en calcular el gradiente para cada muestra de entrenamiento, es decir para $x^{(i)}, y^{(i)}$ obteniendo así

$$\theta = \theta - \eta \cdot \nabla_\theta F(\theta; x^{(i)}, y^{(i)})$$

  Este algoritmo es más rápido para actualizar los parámetros, pero sufre de una gran varianza al actualizarlos haciendo que la función de costos fluctúe en sus valores.

- Descenso por gradiente empleando *mini-batch*: Juntando las ideas de los dos algoritmos anteriores surge esta nueva variante, que como su nombre lo indica, emplea *mini-batch* de los datos, es decir toma subconjuntos del conjunto total de datos, y calculando el gradiente de la función con respecto a los parámetros y a los datos de cada subconjunto se obtiene la siguiente forma de actualizar θ

$$\theta = \theta - \eta \cdot \nabla_\theta F(\theta; x^{(i:i+n)}, y^{(i:i+n)})$$

  De esta manera se reduce la varianza al realizar la actualización; usualmente la longitud de cada *mini-batch* es entre 50 y 256 pero esto puede variar dependiendo del problema a resolver. En el caso del entrenamiento de las redes neuronales usualmente se elige este algoritmo.

Estos algoritmos a pesar de ser muy usados tienen algunas complicaciones al momento de seleccionar la tasa de aprendizaje ya que si se selecciona una tasa muy pequeña puede ocurrir que la convergencia a un punto factible sea muy lenta, por otro lado si se escoge una tasa grande la función de costos podría fluctuar y acercarse sin llegar al mínimo o incluso no converger.

Existen métodos para optimizar estos algoritmos, que consisten en solucionar algunos de los inconvenientes, como el problema de elegir la tasa de aprendizaje antes mencionada. Entre los más comunes se encuentran [Ruder, 2017]: *Momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSprop, Adam, AdaMax, Nadam.*

Prosiguiendo con la explicación del ajuste de los pesos, una vez comprendido el algoritmos de descenso por gradiente y sus variantes se puede explicar el algoritmo de *Backpropagation* que permitirá ajustar los pesos.

Backpropagation

Este algoritmo de aprendizaje es en general el más usado para realizar tareas de aprendizaje supervisado, es aplicado a muchos tipos de redes neuronales como las redes recurrentes y con distintas configuraciones como *Multilayer Perceptron*
[Du and Swamy, 2014]. El fin de este algoritmo es minimizar la función de costos, que orientado a las redes neuronales, esta función puede ser la tasa de error, el error cuadrático medio o cualquier medida de desempeño. Dado que este algoritmo se usa para entrenar la red, y que se conoce el valor real que debe resultar, *backpropagation* compara la salida de la red neuronal con el valor real obteniendo un error que se propagará hacía atrás de la red en las capas ocultas, y usando algún algoritmo de descenso por gradiente se ajustarán los pesos para obtener el menor error posible. Un esquema del algoritmo se observa en la Figura 3.12.

Para ejemplificar y comprender mejor este algoritmo, se presenta un ejemplo básico que se observa en la Figura 3.13. Se tiene una red neuronal con 3 nodos de entrada de información, 3 capas ocultas con 4, 3 y 4 perceptrones respectivamente, 3 conjuntos C_1, C_2, C_3 que contienen respectivamente los pesos de las conexiones de la primera, segunda y tercera capa oculta y finaliza con 4 nodos de salida de información.

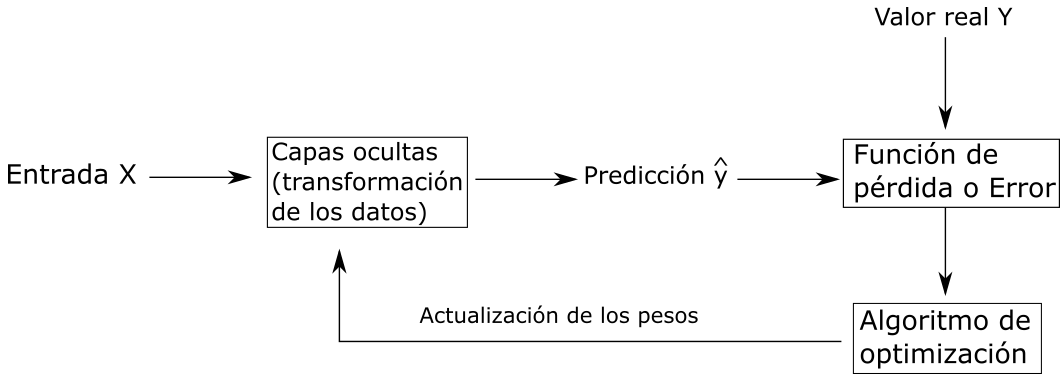


Figura 3.12: Esquema del funcionamiento de *Backpropagation*.

El primer paso del algoritmo es asignar pesos al azar a las conexiones para que las neuronas procesen la información y así obtener la salida de la red, en este caso las salidas son $\{\hat{y_1}, \hat{y_2}, \hat{y_3}, \hat{y_4}\}$. Posteriormente se calcula el error al comparar cada $\hat{y_i}$ con el valor real que se conoce, es decir $E_i =\mid \hat{y_i} - y_i \mid$, esta forma de calcular el error no es la única pero es simple. Este paso se puede ver en la Figura 3.13a

Como siguiente paso se propaga el error hacia atrás para modificar los pesos de las conexiones de manera que al calcular de nuevo el error, este sea menor al obtenido antes de la propagación del error. Para hacer esta propagación empezamos modificando los pesos del conjunto C_3 que corresponden a las conexiones que inciden en la capa oculta número 3, habiendo modificado los pesos se procede a cambiar los pesos del conjunto C_2 y finalmente los del C_1. Esta acción de ir retrocediendo entre las capas ocultas esta representado en la Figura 3.13b.

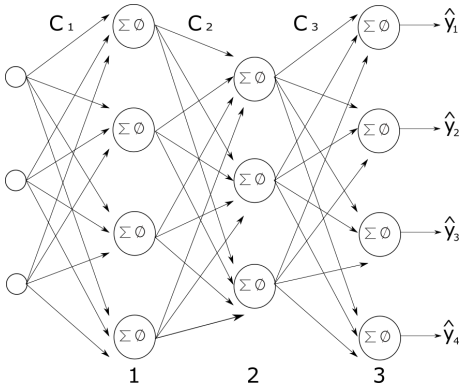
Como se mencionó anteriormente, para actualizar los pesos se usa el algoritmo de descenso por gradiente, en donde se calcula el gradiente del error para actualizar los pesos. De forma matemática la ecuación para actualizar los pesos puede tener la siguiente forma [Du and Swamy, 2014]:

$$W_i = W_i - \eta \frac{\partial E_i}{\partial W}$$

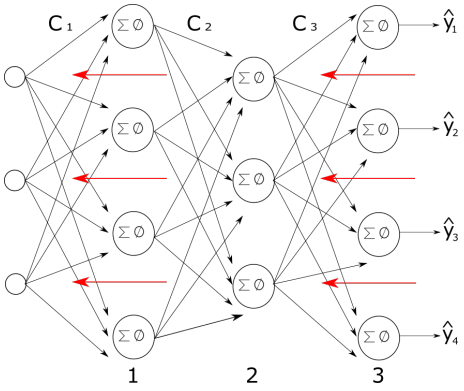
En donde W_i es el peso de la conexión i. Se puede agregar algún método de optimización como el de momentos o momentum, para esto se toma un término γ y

se modifica la ecuación anterior de forma que ahora se tiene

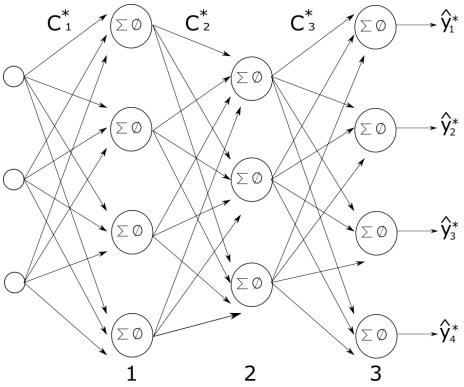
$$W_i = \gamma\, W_i - \eta\, \frac{\partial E_i}{\partial W}$$



(a) Primer paso del algoritmo, se asignan los pesos de las conexiones al azar.



(b) Segundo paso, se propaga el error hacia atrás.



(c) Finalmente se obtiene una red neuronal con pesos actualizados.

Figura 3.13: Proceso del algoritmo *Backpropagation*.

Así, los pesos modificados están contenidos en los conjuntos C_1^*, C_2^*, C_3^* y las salidas de la red son $\{y_1^*, y_2^*, y_3^*, y_4^*\}$. Este proceso de propagar el error se repite las veces necesarias hasta alcanzar un error lo suficientemente pequeño. La figura 3.13c muestra la red con los pesos actualizados y con nuevas salidas del algoritmo que minimizan el error.

Una vez entendido el concepto de las redes neuronales, su funcionamiento y algunos

algoritmos necesarios para mejorar los resultados, se puede continuar introduciendo el aprendizaje profundo.

3.2.2. Acercamiento al aprendizaje profundo

Para comprender de mejor manera el concepto de aprendizaje profundo es importante mencionar que la palabra profundo no hace referencia a algún tipo de profundidad ni nada parecido, este adjetivo se relaciona con el número de capas contempladas en las redes neuronales, entre más capas se agreguen mayor será la profundidad del modelo. Gracias a que el modelo puede tener un gran número de capas ocultas se puede procesar un mayor número de datos; este procesamiento de la información requiere un gran costo computacional, es por esto que, a pesar de que el concepto de aprendizaje profundo no es nuevo, el auge de estos algoritmos se dio cuando se tuvo el suficiente poder computacional y la capacidad de obtener un gran número de información, a pesar de lo mencionado anteriormente, la característica más importante del *deep learning* es que permite aprender automáticamente patrones y representaciones de información sin emplear mucho tiempo.

Se debe tener en cuenta que no cualquier red neuronal es considerada como un modelo de aprendizaje profundo, ya que cuando una red neuronal tiene 4 capas o más, incluyendo la capa de entrada y salida, es cuando se tiene el aprendizaje profundo.

Entre los algoritmos más comunes y usados del aprendizaje profundo se encuentran: Red Neuronal Convolucional o *Convolutional Neural Network* (CNN), Red Neuronal Recurrente o *Recurrent Neural Network*, Codificador automático o *Autoencoder*, *Boltzman Machine* (BM) y Red Generativa Antagónica o *Generative Adversarial Network* (GAN) [Coşkun et al., 2017].

Además de los algoritmos antes mencionados, existen muchos algoritmos más que pueden ser clasificados en 3 grupos dependiendo de su funcionalidad y los problemas que resuelven, estos grupos son generativo, discriminativo e híbrido [Kim et al., 2018].

- Algoritmos generativos: Son los algoritmos de aprendizaje no supervisado que utilizan datos sin etiquetar y se encargan entre otras cosas de etiquetarlos. Algu-

nos algoritmos de esta clase son el Codificador automático, las Redes Neuronales Recurrentes y el *Boltzman Machine*.

- Algoritmos discriminativos: Estos algoritmos son de aprendizaje supervisado donde se cuenta con información y sus etiquetas, uno de los fines de estos algoritmos es encontrar patrones y predecir los siguientes valores. La Red Neuronal Convolucional es un ejemplo de estos algoritmos, otro ejemplo son las Redes Neuronales Recurrentes cuando se usan con datos y sus etiquetas.

- Híbridos: Estos algoritmos combinan la estructura de los algoritmos generativos y discriminativos de forma que pueden realizar tareas de aprendizaje supervisado y no supervisado. Un ejemplo es la Red Neuronal Profunda o *Deep Neural Network*.

Dado que el aprendizaje profundo utiliza las redes neuronales, el proceso de ajustar los pesos es muy importante para estos algoritmos por lo que es necesario tener claro el algoritmo de *backpropagation*, en la Figura 3.13 se observa el algoritmo aplicado a una red neuronal de un modelo de aprendizaje profundo.

Una vez introducido el concepto de *deep learning* y redes neuronales se cuenta con conocimientos suficientes para continuar con las redes neuronales recurrentes, que sirve para entender las redes de tipo *Long-Short-Term-Memory* que se emplean para la predicción.

3.3. Redes neuronales recurrentes

Las redes neuronales recurrentes son una clase de redes pertenecientes a los algoritmos de *deep learning* cuya cualidad más representativa es la capacidad de almacenar la información. Esto se logra mediante un ciclo en el que la red procesa la información en secuencias de elementos, a diferencia de otro tipo de redes que procesan la información en un solo paso [Coşkun et al., 2017]. El siguiente esquema muestra una representación del ciclo de una red neuronal recurrente.

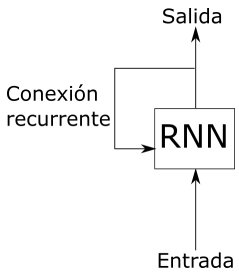


Figura 3.14: Red neuronal recurrente (RNN), la información entra a la red donde es procesada y la información además de salir del modelo entra en un ciclo en donde se vuelve a procesar.

En la figura 3.9b se puede observar el ciclo antes mencionado en donde cada neurona procesa la información y la vuelve a utilizar. A pesar de que la idea de este tipo de redes es simple, es necesario detallar más finamente el funcionamiento de estas redes para entenderlas mejor, ya que el esquema de la Figura 3.14 es una manera simple y compacta que sirve para formarse una idea pero que no es explicativa.

Al proceso de mostrar la red de forma más extensa se le conoce como desenrollar la red a través del tiempo [Chollet, 2017]. En el esquema 3.15 se puede ver este proceso, que además de mostrar cómo desenrollar la red recurrente también servirá para comprenderlas.

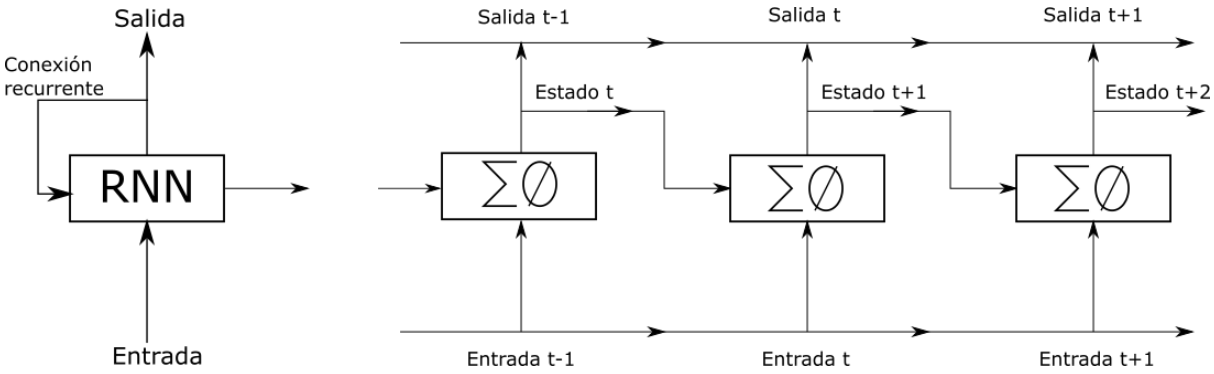


Figura 3.15: Red neuronal recurrente desenrollada, se aprecia con mayor detalle el proceso por el cual la información es procesada por la red, donde la conexión recurrente representa cómo los datos de salida de la neurona al tiempo t-1 se usarán como datos de entrada en el tiempo t.

En el esquema anterior se tiene un rectángulo con los símbolos $\sum \phi$, y es justo ahí donde la información se procesa para obtener una salida, en este tipo de red tenemos datos que provienen del procesamiento de la red en un paso anterior y la

información que se ingresa directamente al modelo. Para ejemplificar mejor, suponga que la red se encuentra en el tiempo t, la información que entra a la neurona es la proveniente de haber procesado los datos en el tiempo $t-1$ y la que es ingresada en el tiempo t, llamadas respectivamente Estado t y Entrada t, estos 2 conjuntos de datos son procesados en la neurona al tiempo t para obtener una salida, misma que se usará en el tiempo $t+1$ para volver a ser procesada. La manera en que se procesa la información es de la siguiente manera:

Se aplica una función de activación a la suma de la Entrada t por su peso más el Estado t por su peso, es decir

$$Salida\ \ t = \phi\left(W \cdot Entrada\ \ t + U \cdot Estado\ \ t\right)$$

Donde ϕ es la función de activación mientras que W es el peso asociado a los datos de Entrada en el paso t y U es el peso de los datos de Estado en t.

Dado que la información de salida en el tiempo $t-1$ es usada en el tiempo t, y la salida en t es a su vez usada en el tiempo de $t+1$ se concluye que no es necesario tener la salida del modelo en cada tiempo, pues con tener la salida del modelo en el último tiempo se tendrá la información procesada en los pasos anteriores.

Como se ha ido mencionando a lo largo del capítulo, para los algoritmos de aprendizaje automático es importante la información pues con esta se entrena al modelo, previamente se habló del algoritmo *Backpropagation* para ajustar los pesos en una red y dado que en la explicación del algoritmo no se hizo ninguna suposición de la red, se podría pensar en aplicar este algoritmo en las redes neuronales recurrentes, sin embargo este tipo de red procesa los datos a través del tiempo, por lo que el algoritmo para ajustar los pesos tiene que ser modificado de manera que se contemplen estos tiempos. Es por esto que surge el algoritmo de *Backpropagation Through Time* (BPTT) traducido como *Backpropagation* a través del tiempo.

3.3.1. Backpropagation Through Time y Desvanecimiento del gradiente

El concepto de este algoritmo es básicamente el mismo que el de *Backpropagation*, usando el descenso por gradiente se encuentra el valor mínimo del error para actualizar los pesos, la diferencia consiste en que, al tener un procesamiento recursivo de información, en cada iteración se debe propagar el error de manera que se minimice, además se tienen distintas entradas y salidas de información que se deben contemplar en el algoritmo. Asimismo es necesario aclarar que una vez obtenidos los pesos de la red estos no cambiarán en las distintas iteraciones, por lo que al actualizar uno de los pesos utilizados en los rectángulos de la Figura 3.15 se actualiza en todos los rectángulos [Bianchini et al., 2013]. Se puede pensar que la propagación del error se hace con la red desenrollada, mientras que el peso actualizado corresponderá a la red de la Figura 3.14 sin importar las iteraciones que se realicen.

Para actualizar los pesos se tiene a grandes rasgos la misma ecuación que la expuesta al explicar de manera general el algoritmo, la diferencia radica en el cálculo del gradiente del error con respecto a los pesos, ya que como se ve en la Figura 3.15 por cada tiempo hay una salida, por lo que en cada tiempo hay un error. En el esquema de la Figura 3.16 se muestra el recorrido que se debe que hacer para calcular las derivadas parciales correspondientes al tiempo.

En el ejemplo de la Figura 3.16 se presenta la trayectoria por la cual se tiene que propagar el error en el tiempo t. Lo que se busca calcular es $\frac{\partial E_t}{\partial W}$ y para esto se usa la regla de la cadena, ya que se quiere propagar el error hasta el tiempo cero, el camino es comenzar con el error e ir pasando por los perceptrones en cada tiempo hasta llegar a la capa de entrada deteniéndose en cada conexión entre perceptrones ya que cada conexión tiene el peso que queremos actualizar, el procedimiento es comenzar obteniendo $\frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial W}$ aquí se propaga el error hasta el peso, que corresponde a la conexión entre los tiempos $t-1$ y t, se continúa ahora obteniendo $\frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W}$ ahora el error se propagó hasta el peso de la conexión entre los tiempos $t-2$ y $t-1$, y sucesivamente se realiza este procedimiento hasta llegar a la conexión entre la capa

de entrada y el primer perceptrón, en el tiempo 1, obteniendo

$$\frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\cdots\frac{\partial h_1}{\partial W}$$

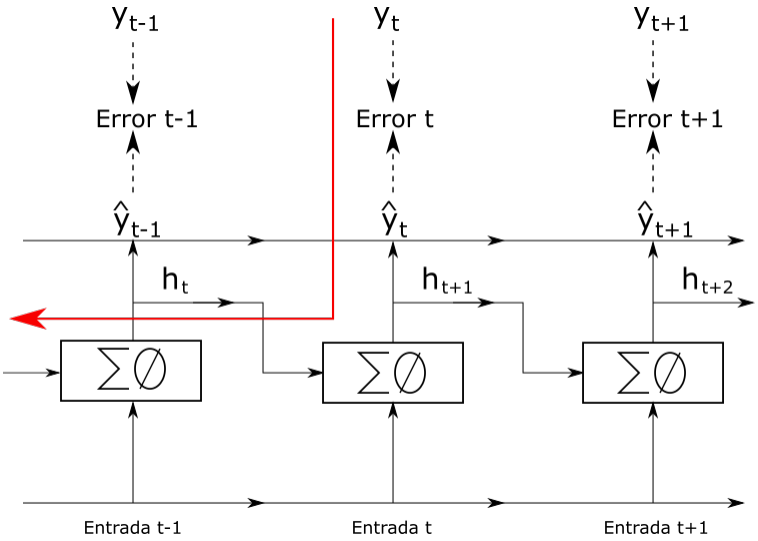


Figura 3.16: Esquema del algoritmo *Backpropagation Through Time*, la flecha roja muestra la dirección de propagación del error. Para la propagación se utilizan los datos de salida $\hat{y}_k$ así como los datos que se comparten de un perceptrón a otro, se denota por h_k a la información procesada en el tiempo $k-1$ y que se usa como datos de entrada en el tiempo k.

Todo lo anterior fue para obtener $\frac{\partial E_t}{\partial W}$ en todos los tiempos, por lo que la ecuación queda de la siguiente manera

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial W} + \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W} + \ldots + \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\cdots\frac{\partial h_1}{\partial W}$$

Lo anterior se puede reescribir de la siguiente manera:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t}\frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

Esta manera de simplificar $\frac{\partial E_t}{\partial W}$ podría generar confusión ya que cuando $k=1$ se tendrá

$$\frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_1}\frac{\partial h_1}{\partial W}$$

que difiere de la manera extensa de escribirlo, pero recordando la regla de la cadena se tiene

$$\frac{\partial h_t}{\partial h_1} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_2}{\partial h_1}$$

por lo que termina siendo lo mismo y se puede usar esta manera compacta de escribir $\frac{\partial E_t}{\partial W}$.

Con todo lo antes mencionado en esta sección se puede escribir la ecuación de *Backpropagation Throught Time* que a pesar de escribirse igual que en el caso de *Backpropagation*, tiene detrás el cálculo de la parcial del error a través del tiempo.

$$W_t = W_t - \eta \, \frac{\partial E_t}{\partial W}$$

A pesar de todas las consideraciones que este algoritmo toma, tiene un gran problema relacionado con el número de capas y los tiempos de las redes recurrentes, este problema conocido como Desvanecimiento del gradiente o *Vanishing gradient* ocurre al momento de entrenar una red con muchas capas ocultas. Cuando se aplica *Backpropagation* o *Backpropagation Throught Time* para actualizar los pesos, propagando el error, el gradiente utilizado para actualizarlos se va desvaneciendo a tal punto que no permite que los pesos se actualicen, esto ocasiona que el error del modelo al realizar las tareas no disminuya y la red no pueda ser entrenada [Rebala et al., 2019].

Tomándose por ejemplo la función de activación tanh $f(x) = \frac{2}{1+e^{-2x}} - 1$ cuyo gradiente es $\frac{4e^{-2x}}{(1+e^{-2x})^2}$ y su rango es $\{y \in \mathbb{R} : 0 < y \leq 1\}$, se aprecia que al aplicar n veces la regla de la cadena, como parte del algoritmo *Backpropagation*, se tienen n números entre el cero y el uno que se multiplican, y al hacer crecer n, el producto tiende a cero, ocasionando que no se actualicen los pesos de las capas ocultas más cercanas a la capa de entrada.

En el problema del desvanecimiento del gradiente, las últimas capas del modelo son las que pueden actualizar los pesos, y al retroceder sobre el modelo, se dificultaría actualizar los pesos de las primeras capas, por lo que se imposibilitaría el aprender a largo plazo, solo se contaría con el aprendizaje a corto plazo.

Existen varias soluciones a este problema, como usar métodos que no requieran

calcular el gradiente, métodos que usen redes con arquitecturas especiales, usar ciertas funciones de activación que no sufren tanto de este problema como la función $ReLU$. [Hochreiter, 1998].

Capítulo 4

Redes Long Short-Term Memory (LSTM)

Como se mencionó al final del capítulo anterior una de las formas de evitar el desvanecimiento del gradiente es usando ciertas arquitecturas especiales de las redes. Una de estas es la red *Long Short-Term Memory*, que se obtiene al modificar las redes neuronales recurrentes agregando un mecanismo para conservar la información.

Las redes LSTM fueron introducidas por primera vez por Hochreiter y Schmidhuber en 1997 para solucionar el problema de conservar la información introducida en los primeros tiempos de la red. Para esto se agregan a la arquitectura usual de las RNN celdas de memoria, compuertas por donde la información pasa y un control que permita olvidar la información poco relevante y continuar transmitiendo la información importante en los siguientes tiempos de la red [Hochreiter and Schmidhuber, 1997].

Al igual que en las redes recurrentes, estas nuevas redes siguen el proceso de obtener información tanto del tiempo anterior como de la información ingresada en el tiempo actual, agregando para la red LSTM una nueva entrada que transporta solo la información relevante de los tiempos anteriores.

Estas redes no solo son capaces de procesar datos individuales sino que pueden procesar secuencias completas de datos lo que facilita la realización de las tareas [Taroon et al., 2020]. Un ejemplo de esta mejora se puede observar en el procesamiento de palabras con el fin de predecir la siguiente o siguientes palabras, pues al procesar

no solo una palabra sino una frase completa, el algoritmo aprende patrones y reglas gramaticales que facilitan la realización de la predicción, asimismo puede analizar una serie de tiempo tomando los datos como una secuencia.

A continuación se observa la arquitectura de la red LSTM de manera desenrollada en el tiempo t, posteriormente se analiza cada uno de sus componentes para tener un mejor entendimiento del proceso por el cual la información se procesa.

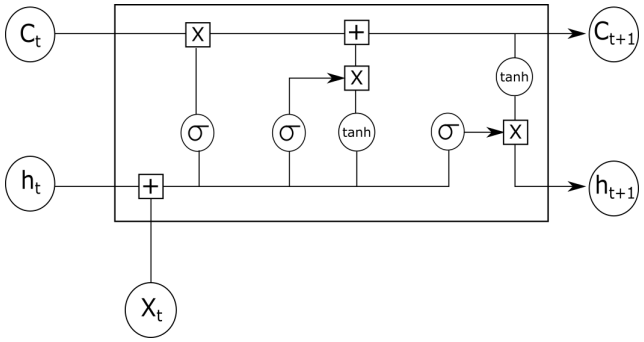


Figura 4.1: Red LSTM en el tiempo t.

Como se observa en el tiempo t la red recibe como entradas X_t, h_t y C_t, las primeras dos entradas conservan la función que tenían en las redes recurrentes, la nueva entrada, C_t, es el mecanismo del que se ha hablado previamente por el cual se conserva la información anterior.

Para comprender las características y funcionamiento de la red, se debe analizar cada flujo y cada componente de la red en 4.1, teniendo en cuenta que aun cuando la explicación sea para el tiempo t, el análisis se podrá replicar para cualquier tiempo de la red.

Para comenzar se explica cada componente de la red para tener una idea general del funcionamiento:

- X_t: Información externa al modelo que es ingresada en el tiempo t.
- h_t: Información procesada en el tiempo $t-1$ de la red y que se usará en el tiempo t.

- C_t: Información proveniente del tiempo $t-1$ con la cual se evita el desvanecimiento del gradiente ya que contiene la información relevante hasta ese tiempo, permitiendo que al aplicar *Backpropagation* se tenga información de los primeros tiempos del modelo y así se represente toda la información sin importar en que tiempo t se aplique el algoritmo.
- σ: Representa la función de activación sigmoide
- tanh: Función de activación tangente hiperbólica.
- $\boxed{\times}$: Operación en la cual se multiplican entrada por entrada dos vectores.
- $\boxed{+}$: Operación en la cual dos vectores se suman entrada por entrada.

Habiendo mencionado los componentes, se procede a analizar las primeras operaciones que realiza la red que se representan en la siguiente figura. En ella se tiene el ingreso de la información del modelo y las operaciones realizadas. Es importante aclarar que en todos los procesos de la red, se usan las dos entradas del modelo X_t y h_t, así como los vectores que vayan resultando de realizar las operaciones.

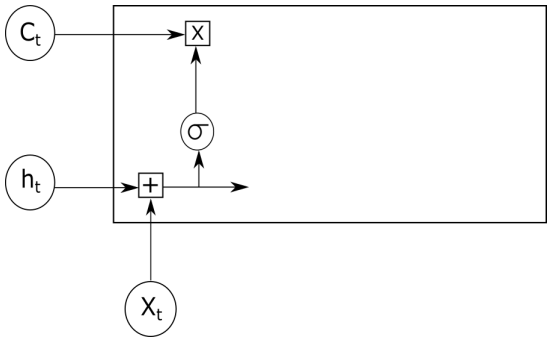


Figura 4.2: Primeros procedimientos de la red LSTM en el tiempo t, el algoritmo recibe la información en X_t,h_t y C_t, usando operaciones matemáticas la red procesa la información obteniendo una parte de la salida del modelo.

El proceso descrito en la Figura 4.2 es la entrada de los datos a la red, los vectores X_t y h_t son sumados entrada por entrada para obtener un vector nuevo, a este vector se le aplica la función sigmoide, que se sabe convierte el vector ingresado en un vector donde cada entrada es un número entre 0 y 1. Con esta operación se tiene una manera

de olvidar los datos irrelevantes, los datos que fueron asignados a números cercanos a cero al aplicar la función de activación son los datos que se olvidarán mientras que los cercanos a 1 se mantienen en la red.

El vector resultante de la función sigmoide es multiplicado entrada por entrada con el vector C_t que como se ha mencionado, contiene la información relevante hasta el tiempo t-1. Con este producto se obtiene un vector que indica la información relevante de C_t ya que las entradas multiplicadas por números cercanos a cero son pequeñas, se interpretan como información poco importante y se olvidan, quedando solo la información relevante de C_t, denotado por C'_t. El vector obtenido del producto es una parte del vector C_{t+1}.

Continuando con las operaciones se tiene la siguiente etapa, mostrada en el esquema 4.3, en esta etapa se realizan operaciones con los vectores X_t y h_t así como el vector C'_t. Se toma el vector que resulta de la suma de cada entrada de los vectores X_t y h_t $(X_t + h_t)$ y se le aplica la función sigmoide, al mismo tiempo se toma una vez más el vector $X_t + h_t$ y ahora se le aplica la función tanh. Después se multiplican entrada por entrada los vectores resultantes de aplicar las funciones sigmoide y tanh, llamado para este caso F_t. Finalmente se suman los vectores F_t y C'_t dando como resultado el vector C_{t+1} que contiene la información relevante de todos los tiempos hasta el t.

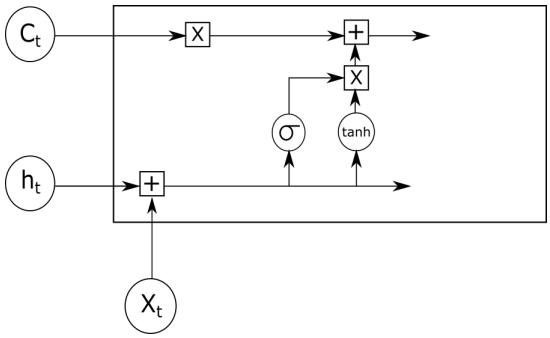


Figura 4.3: Segunda etapa del procesamiento de información de la red LSTM.

El hecho de usar la función tanh, como se mencionó en el capítulo anterior, permite transformar los datos en números que se encuentran en el intervalo $[-1, 1]$. Además como se vio previamente, al usar la función sigmoide se obtiene un vector de olvido para considerar solo la información relevante, por lo que al multiplicar el vector ob-

tenido de la función sigmoide con el de la función tanh se obtiene un vector de olvido pero con datos en el intervalo antes mencionado.

Para terminar el proceso, la red realiza unas últimas operaciones para obtener el vector h_{t+1} que se usa en el siguiente tiempo. En esta última etapa se toma el vector $X_t + h_t$ y se le aplica la función sigmoide, posteriormente se aplica la función tanh al vector C_{t+1} que se obtuvo en la etapa anterior y finalmente se multiplicaran los vectores obtenidos de las funciones tanh y sigmoide, dando como resultado el vector h_{t+1} usado en el siguiente tiempo de la red. El esquema de esta etapa se encuentra en la siguiente imagen.

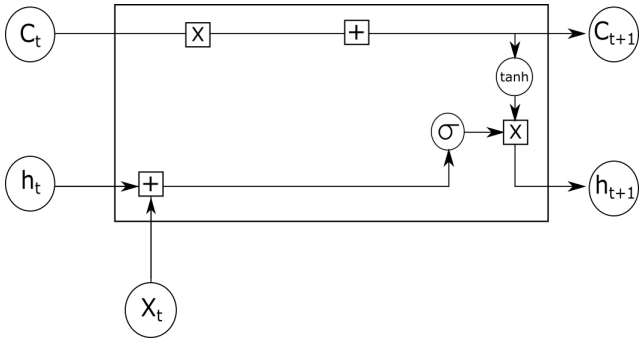


Figura 4.4: Última etapa del proceso de la red. Usando el vector de olvido y multiplicándolo por el vector C_{t+1} con entradas entre -1 y 1 se obtiene la información importante que se usa en el tiempo $t + 1$.

Habiendo comprendido el funcionamiento de la red y las operaciones que se realizan en cada tiempo se puede concluir de forma breve que las redes LSTM funcionan con compuertas (*gates*) por las cuales pasa la información. Se pueden agrupar las operaciones antes mencionadas en las siguientes compuertas [Siami-Namini and Namin, 2018]:

- Compuerta de olvido (*Forget Gate*): En esta primera compuerta se usa el vector $h_t + X_t$ y la función sigmoide para obtener el vector de olvido, que como se ha mencionado, permite mantener en la red la información importante asignando números cercanos a 1.

- Compuerta de entrada (*Input Gate*): Esta compuerta comprende las operaciones de aplicar las funciones sigmoide y tanh al vector $h_t + X_t$ y multiplicar los dos vectores resultando en el vector F_t.

- *Cell state*: Usando los resultados de las compuertas anteriores se obtiene el vector C_{t+1} para esto se multiplica el vector C_t por el vector de olvido obtenido en la compuerta de olvido dando como resultado el vector C'_t antes mencionado, posteriormente se suman los vectores C'_t y F_t dando como resultado el vector C_{t+1}.

- Puerta de salida (*Output Gate*): Finalmente usando el vector C_{t+1} y el vector resultante de aplicar la función sigmoide al vector $h_t + X_t$ se obtiene el vector h_{t+1}.

Debido a la arquitectura de las redes LSTM se tiene un mejor análisis y procesamiento de la información ya que, además de agregar un vector para evitar el desvanecimiento del gradiente, incorpora diferentes operaciones en una sola capa. Debido a estas mejoras, se usan para realizar diferentes tareas obteniendo mejores resultados que si se usaran otros algoritmos de redes neuronales. Entre las aplicaciones más comunes se tiene la clasificación, procesamiento de lenguaje, reconocimiento de voz y procesamiento de datos de series de tiempo [Jia et al., 2019].

Existen otras configuraciones de las redes, entre las que destacan las redes LSTM bidireccionales y las GRU (*Gated recurrent unit*). Las redes GRU comparten la idea principal de las LSTM pero son redes más sencillas.

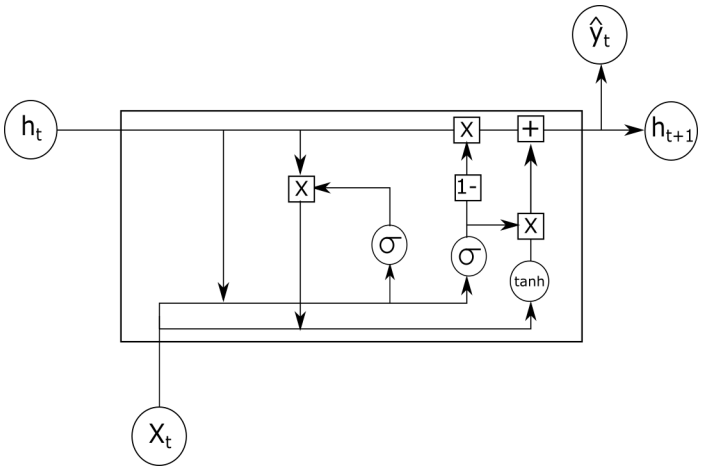


Figura 4.5: Representación de la capa en el tiempo t de la red GRU.

Como se puede ver en la Figura 4.5 la red GRU similar a la red LSTM, consiste

en una serie de operaciones aplicadas a los datos de entrada, al igual que en el caso de la red LSTM, se cuenta con un mecanismo para olvidar la información irrelevante y agregar la información importante en cada tiempo t. La única operación que realiza la red GRU distinta a la red LSTM es la que se representa por $\boxed{1-}$ y consiste en realizar la operación $1 - sigmoide(X_t + h_t)$. Esta red cuenta con dos compuertas:

- Compuerta de reinicio (*Reset Gate*): Esta compuerta es usada para decidir que información del pasado olvidar. Para esto se aplica la función sigmoide al vector $X_t + h_t$ y el resultado se multiplica por el vector h_t obteniendo la información relevante.

- Compuerta de actualización (*Update Gate*): Se aplica la función sigmoide al vector $X_t + h_t$ para posteriormente realizar la operación $1 - sigmoide(X_t + h_t)$ y multiplicar el resultado al vector h_t. En esta compuerta también se decide que información mantener y cual desechar. Funciona de manera similar a la combinación de las compuertas de olvido y de entrada.

Por otro lado las redes LSTM bidireccionales son a grandes rasgos iguales que las redes LSTM con la diferencia que en las bidireccionales los datos son recorridos en ambos sentidos, es decir, del pasado al presente y del presente al pasado. Mientras que en las redes LSTM los datos usados consisten en un vector de información de tamaño n que contiene información del pasado; para el caso de la bidireccional se usa un vector de tamaño $2n$ en donde las primeras n entradas son iguales a las del vector usado para la LSTM, y para las otras n entradas se usan los n datos pero cambiando el orden del más actual al mas antiguo. La Figura 4.6 muestra la diferencia entre los datos procesados por las redes LSTM y LSTM bidireccionales, respectivamente.

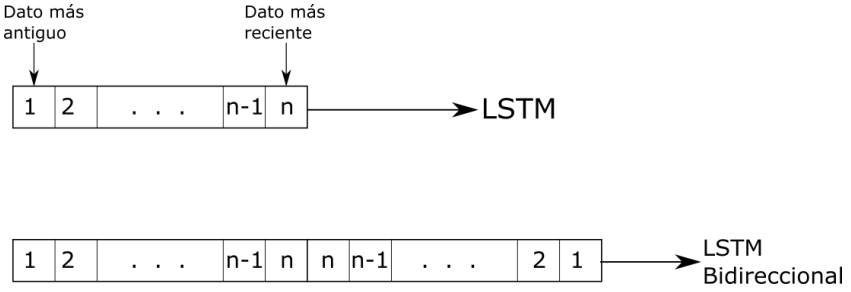


Figura 4.6: Diferencia entre los datos de entrada de la red LSTM y la bidireccional.

Capítulo 5

Metodología

Como se ha mencionado a lo largo del trabajo, el fin de este es usar algoritmos de *Machine Learning*, específicamente las redes LSTM, para predecir los valores en el futuro de índices y acciones. En el presente capítulo se explica el proceso por el cual se pasó para predecir el precio de las acciones, desde la obtención de la información de las empresas hasta el entrenamiento del modelo. Posteriormente en el siguiente capítulo se presentan los experimentos realizados así como los resultados obtenidos. Es importante mencionar que se trabaja con el lenguaje de programación *Python*.

5.1. Obtención de los datos

Lo primero que se debe hacer es elegir los datos que sirven para entrenar el modelo y de los cuales se realizan las predicciones. Para el índice accionario se eligió el S&P/BMV IPC ya que permite tener una idea del comportamiento de la economía mexicana. En general este índice esta constituido por las acciones de 35 empresas que son las más representativas de los sectores mostrados en la Figura 5.1, donde se puede observar que el sector que más presencia tiene es el de productos básicos de consumo, que son los productos como alimentos, bebidas o artículos de higiene.

Figura 5.1: Sectores a los que pertenecen las distintas empresas que conforman el S&P/BMV IPC. Se incluye el porcentaje que cada sector tiene en el índice.

Lo primero que se realizó fue elegir cuáles serían los datos con los que se trabajaría así como la cantidad de datos que se usarían. Para esto se eligieron las empresas América Móvil, Walmart de México y Centroamérica, FEMSA, Grupo Televisa y Grupo Financiero Banorte para usar sus acciones.

Cantidad de datos

De los capítulos anteriores se sabe que un algoritmo de aprendizaje automático necesita una gran cantidad de datos para aprender contemplando el no sobre-ajustar el modelo. Y dado que se trabaja con datos financieros y económicos se necesita tener información que no presente una gran variación repentina, es por esto que se eligieron los datos comprendidos entre el 4 de enero del 2010 al 30 de diciembre del 2016, ya que en este periodo hubo una estabilidad en el sentido que no se encontraba en una crisis como la del 2008-2009 ni en una situación como la del SARS-CoV-2, y a pesar de que en el 2009-2010 se tuvo una pandemia por el influenzavirus A subtipo H_1N_1, esta pandemia no detuvo la economía ni causo los problemas financieros que la pandemia que comenzó en el 2019.

Empresas

De las 35 empresas que constituyen el S&P/BMV IPC se eligieron las cinco que tengan mayor representación en el índice, es decir, las de mayor ponderación. Dado que se usaron 7 años de datos, se buscó la mayor información sobre la ponderación de las empresas en esos años y de acuerdo a la búsqueda se optó por las siguientes empresas:

- América Móvil (AMXL.MX): Es una empresa multinacional mexicana de telecomunicaciones.
- Walmart de México y Centroamérica (WALMEX.MX): Es la cadena de comercialización minorista que Walmart Stores posee en México y Centroamérica.
- Fomento Económico Mexicano S.A.B. de C.V. (FEMSAUB.MX): Conocida comúnmente como FEMSA, es una empresa multinacional mexicana que participa en la industria de las bebidas, y en el sector comercial y de restaurantes.
- Grupo Televisa (TLEVISACPO.MX): Es una empresa mexicana de medios de comunicación.
- Grupo Financiero Banorte, S.A.B de C.V. (GFNORTEO.MX): Conocido simplemente como Banorte y como Ixe, es una institución financiera y bancaria mexicana, es uno de los cuatro bancos más grandes de México, en términos de activos y préstamos, y el administrador más largo de afores.

Toda la información de los precios de las acciones y del S&P/BMV IPC se obtuvieron de la página *Yahoo Finance* (https://finance.yahoo.com/) y se descargaron mediante la librería *yfinance* con la cual se crea una conexión entre la página de *Yahoo Finance* y la máquina donde se ejecuta el código.

Por cada una de las empresas y del índice se obtuvo un archivo .csv que contiene los datos de los precios de cada acción. Un ejemplo de los datos se puede ver en la Tabla 5.1 en donde para cada día se tiene una serie de precios y el volumen de la acción en ese día

Date	Open	High	Low	Close	Adj Close	Volume
04/01/2010	32,120.74	32,758.53	32,120.74	32,758.53	32,758.53	136,257,800
05/01/2010	32,729.30	33,073.71	32,628.24	32,732.76	32,732.76	165,541,900
06/01/2010	32,730.42	32,922.12	32,639.44	32,830.16	32,830.16	126,000,100
07/01/2010	32,830.22	33,069.78	32,670.98	33,064.57	33,064.57	123,563,200
08/01/2010	33,067.57	33,080.05	32,779.66	32,892.04	32,892.04	109,516,900
11/01/2010	32,892.11	33,047.23	32,711.04	32,935.38	32,935.38	150,332,200
12/01/2010	32,950.34	32,976.61	32,719.62	32,792.66	32,792.66	139,205,800
13/01/2010	32,792.87	32,929.32	32,607.94	32,836.08	32,836.08	143,683,200
14/01/2010	32,843.56	32,847.93	32,429.67	32,729.58	32,729.58	447,963,200
15/01/2010	32,725.41	32,728.41	32,262.10	32,262.30	32,262.30	318,270,300
18/01/2010	32,262.53	32,491.48	32,262.53	32,482.73	32,482.73	32,108,200
19/01/2010	32,493.36	32,618.01	32,367.86	32,473.05	32,473.05	213,255,300
20/01/2010	32,466.04	32,466.04	31,946.77	32,025.34	32,025.34	215,329,000
21/01/2010	32,033.85	32,151.80	31,181.72	31,205.30	31,205.30	243,330,800
22/01/2010	31,203.10	31,212.55	30,612.06	30,830.91	30,830.91	210,227,600
25/01/2010	30,831.40	31,080.98	30,465.06	30,465.06	30,465.06	196,961,600
26/01/2010	30,461.86	30,764.76	30,209.09	30,651.56	30,651.56	183,139,500
27/01/2010	30,629.79	30,677.04	30,082.77	30,610.83	30,610.83	306,309,200
28/01/2010	30,620.59	30,926.05	30,334.67	30,811.35	30,811.35	202,374,700
29/01/2010	30,811.47	31,002.79	30,343.05	30,391.61	30,391.61	241,668,900

Tabla 5.1: Visualización de un segmento de datos del índice S&P/BMV IPC. Se tienen los precios de apertura y cierre de mercado, así como el precio más alto y el más bajo que tuvo en el día, también se tiene el precio de cierre ajustado, que se obtiene de modificar el precio de cierre para reflejar el valor del acción después de los movimientos corporativos; y el volumen de acciones que se negociaron en el día.

Posteriormente se realizó un análisis de los datos con el fin de encontrar problemas como datos faltantes o *missing*, este problema es común al momento de trabajar con bases de datos, ya que estos datos pudieron no ser capturados por un error humano o porque simplemente no se encuentran disponibles, tal es el caso de los datos financieros ya que los mercados financieros no trabajan en fin de semana y días de descanso obligatorio, este problema aumenta al momento de trabajar con información de mercados financieros a lo largo del mundo, ya que cada país tiene distintos días de asueto. Este no es el caso ya que se trabajarán con empresas mexicanas y cuyas acciones se negocian en la Bolsa Mexicana de Valores. Sin embargo para los datos de FEMSAUB.MX se encontró un dato faltante en el día 29 de abril del 2010 en todas las columnas, dado que se usarán redes neuronales, es necesario que no haya ninguna entrada vacía, para solucionar este problema se agregó en esta fecha el valor obtenido

de promediar el valor de los dos días anteriores y el de los dos días siguientes al dato faltante, es decir

$$valor_t = \frac{valor_{t-2} + valor_{t-1} + valor_{t+1} + valor_{t+2}}{4}$$

Es importante mencionar que no es la única forma de actuar ante los datos faltantes, pues dependiendo de lo que se vaya a realizar puede incluso no importar que se tengan estos datos vacíos, entre las soluciones más comunes se encuentran el excluir los registros o las variables en donde se tengan estos datos *missing* o completar los valores con técnicas estadísticas.

Habiendo solucionado el problema de los datos faltantes se procede a transformar los datos para poder ser ingresados a la red neuronal de la que posteriormente se habla.

5.2. Selección y transformación de los datos

Como se mencionó antes y como se puede ver en la Tabla 5.1 para cada día se tienen distintas variables que contienen los precios para cada acción y para el índice, y aunque se pueden predecir los siguientes valores para cada variable, para el propósito del trabajo solo se predicen los valores para la variable *Close* que representa los precios de cierre.

Existen muchos artículos y trabajos que buscan predecir los valores futuros de la variable *Close* usando los datos pasados de todas las variables, es decir *Open, High, Low, Close, Adj Close y Volume*, sin embargo para el presente trabajo se planteó usar solo la variable *Close* del S&P/BMV IPC y de las 5 empresas antes mencionadas para predecir los precios de cierre del IPC, es por esto que se crea un *Dataframe* que contenga los precios de cierre de cada una de las empresas y el índice.

Acción/índice	Valor máximo	Valor mínimo	Media	Primer cuartil	Segundo Cuartil	Tercer cuartil
IPC	48694.89	30368.08	41074.65	37997.34	41803.13	44395.76
AMXL.MX	18.66	10.75	14.73	13.18	15.08	16.18
WALMEX.MX	47.22	28.02	36.70	33.75	36.05	40.08
FEMSAUB.MX	155.00	40.00	106.05	79.00	108.00	137.00
TLVISACPO.MX	122.93	45.25	76.74	57.57	74.82	95.39
GFNORTEO.MX	113.85	37.95	77.04	58.49	82.95	90.71

Tabla 5.2: Análisis exploratorio de los datos que se usan en el trabajo. Los datos usados para el análisis son los precios de cierre de las acciones de las empresas y del índice.

Como se puede ver en la Tabla 5.2, los datos que se tienen están en diferentes escalas, ya que en los datos del IPC los números están en decenas de miles mientras que los datos de las empresas son en decenas y centenas. Muchos algoritmos de aprendizaje automático necesitan que los datos de entrada estén normalizados para obtener mejores resultados, es decir, que se transformen los datos de manera que se encuentren en un rango definido. Por esta razón es necesario que todos los datos de entrada, sin importar si se tratan de datos del IPC, de Walmex o de cualquier otra empresa, estén en un intervalo con el cual la red neuronal pueda procesarlos. Dado lo anterior se procede a escalar los datos de manera que todos se encuentren en el intervalo $(0, 1)$, para esto se usa la función $sklearn.preprocessing.MinMaxScaler()$ para definir el intervalo en el que se normalizarán los datos, para este caso será: $sklearn.preprocessing.MinMaxScaler(feature\_range = (0, 1))$.

La función anterior realiza las siguientes operaciones

$$X_i^{escalado} = \frac{X_i - min(X)}{max(X) - min(X)}(max - min) + min$$

en donde

- X_i es el dato en la posición i de la variable X.
- $min(X)$ es el mínimo valor que toma la variable X.
- $max(X)$ de manera similar al punto anterior, este es el valor máximo que toma la variable X.

- *max* es el limite superior del intervalo que se especificó en la función, en este caso es 1.

- *min* análogamente al caso anterior este el mínimo del intervalo, en nuestro caso es 0.

Es importante mencionar que existen muchos métodos para cambiar la escala de los datos y el uso de cada método dependerá de los datos que se tengan, por eso es necesario asegurarse de que, a pesar de que los datos estén escalados, conserven las características que se necesitan para realizar el análisis y procesamiento. En el caso de las redes neuronales, estas necesitan tomar de la serie de tiempo su tendencia, su estacionalidad y aleatoriedad para poder obtener mejores resultados en las predicciones.

Por lo anterior es necesario que al escalar los datos de las empresas y el S&P/BMV IPC sigan conservando su forma para no perder ninguna característica

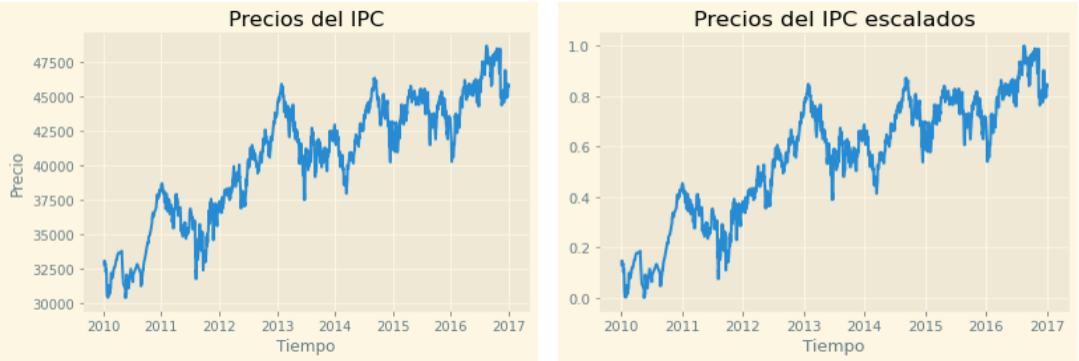


Figura 5.2: Del lado izquierdo se tiene los precios de cierre del IPC del año 2010 al 2016. Del lado derecho se tienen los precios de cierre escalados en el mismo intervalo de tiempo.

Como se puede observar en las gráficas de la figura 5.2 aun cuando los datos cambiaron de escala siguen preservando sus características de serie de tiempo lo que permite que la red pueda procesar la información de una mejor manera. Del mismo modo que se realiza el procedimiento con los datos del índice también se realiza con los precios de cierre de las 5 empresas seleccionadas.

Para finalizar con la sección de transformación de datos es necesario dividir los datos en los subconjuntos mencionados en el capítulo 3, es decir, los subconjuntos de

entrenamiento, validación y prueba. Se sabe que no existe una fórmula para dividir los datos pero si existen consideraciones a seguir para dividir los datos de la mejor manera. De lo escrito en el capítulo 3 se tiene que una manera de dividir los datos es en 70 % para el conjunto *train* y 15 % para cada uno de los conjuntos *validation* y *test*, sin embargo para el trabajo, se dividirán los datos de otra manera, ignorando los porcentajes y seleccionando los datos para cada conjunto de acuerdo a la fecha, de manera que los datos que compongan el conjunto de entrenamiento son los datos que se encuentren entre el 4 de enero del 2010 al 31 de diciembre del 2014, continuando con los datos de validación que se encuentran entre el primero de enero y el 3 de diciembre del 2015, finalmente los datos de prueba están comprendidos entre el primero de enero y el 30 de diciembre del 2016.

Figura 5.3: Representación gráfica de la división de los datos en los tres subconjuntos.

5.3. Creación de la red en Keras para predecir

Con anterioridad se ha hablado un poco de la teoría y funcionamiento que hay detrás de las redes neuronales y específicamente de las LSTM, una vez comprendidas las bases de estas, se encuentra en condiciones de implementarlas para predecir los valores de nuestras acciones. Antes de comenzar con la construcción de la red es importante mencionar que se trabaja con la biblioteca *Keras* para *Python*, esta permite definir y entrenar muchos modelos de aprendizaje profundo, por lo que sirve para las redes LSTM.

Para continuar se deben comentar algunas consideraciones para la creación de modelos de redes neuronales con *Keras*.

5.3.1. Datos de entrada y salida en una red

Para cada lenguaje de programación existen diferentes formas para almacenar los datos, y *Python* no es la excepción, este cuenta con varias formas, entre las formas de almacenar los datos tenemos los tensores o *tensors*, estos son contenedores de datos, mayormente datos numéricos y su importancia radica en que la gran mayoría de los modelos de aprendizaje automático reciben los datos en forma de tensores, y devuelven de igual manera, los datos en tensores.

Los tensores pueden variar de dimensión y dependiendo de esto se tienen arreglos de datos distintos, como se puede ver a continuación

- Un tensor de dimensión 0, denotado por $0D$ tensor, se le conoce como escalar o *scalar* y consiste en un tensor que solo contiene un número. Un escalar puede ser (1).

- Un tensor de dimensión 1 o también llamado vector, consiste en un arreglo que contiene al menos un escalar; a este tensor se le denota como $1D$ tensor y en *Python* se denota como ($[1, 2, 3, 4, 5, 6]$). Este vector esta compuesto de 6 escalares o 6 tensores de dimensión 0.

- Una matriz es un tensor de dimensión 2 y se compone por una serie de vectores, esta se denotan por $2D$ tensor y su característica más importante es que cuenta con filas y columnas, es por esto que tiene 2 dimensiones. Un $2D$ tensor en *Python* se puede ver como

$$
\begin{aligned}
&([\,[1, 2, 3, 4],\\
&\quad [5, 6, 7, 8],\\
&\quad [9, 1, 2, 3]\,]\,)
\end{aligned}
$$

  Aquí se tiene un $2D$ tensor, ya que cada entrada es un vector. Esta matriz esta compuesta de 3 filas y 4 columnas, es decir que son 3 vectores y cada vector tiene 4 entradas.

- Existen además tensores de dimensión 3, estos son arreglos que en cada entrada contienen una matriz, así además de tener las 2 dimensiones de la matriz (renglones y columnas) tienen otra dimensión que especifica la posición de cada matriz.

$$
\begin{array}{r}
([\,[\,[1,2,3,4], \\
[5,6,7,8], \\
[9,1,2,3]\,], \\
[\,[2,3,4,5], \\
[6,7,8,9], \\
[1,2,3,4]\,]\,])
\end{array}
$$

  En este ejemplo se tiene un $3D$ tensor con dos entradas, cada entrada es una matriz (o $2D$ tensor) con 3 filas y 4 columnas.

Existen tensores de mayor dimensión que se usan para almacenar mayor cantidad de arreglos de datos, usualmente estos tensores se utilizan como entradas de los modelos que analizan imágenes y vídeos.

Un atributo muy importante de los tensores es su forma o *shape* con esto se puede identificar la dimensión del tensor así como las dimensiones de cada entrada. Tomando los ejemplos vistos antes para cada dimensión del tensor, se tiene que la forma del $0D$ tensor es () es decir, es vació; la forma del $1D$ tensor es $(6,)$ ya que cuenta con 6 entradas, mientras que la del $2D$ tensor es $(3,4)$ esto porque tiene 3 filas y 4 columnas; finalmente la forma del $3D$ tensor del ejemplo es $(2,3,4)$ pues son 2 entradas (en cada entrada hay una matriz) y cada matriz tiene 3 filas y 4 columnas.

Para el caso de las series de tiempo financieras con las que se trabaja se necesitan usar tensores de dimensión 3, en cada entrada del tensor habrá una muestra de los datos que corresponderá a una matriz en donde cada columna representará las observaciones de la variable (o empresa en nuestro caso) a lo largo del tiempo y por ende cada fila representará todas las observaciones de las variables (o empresas) en un solo

día.

Para finalizar con los datos de entrada, salida y los tensores se debe comentar que a pesar de que las redes neuronales reciben tensores, al momento de seleccionar los datos y agruparlos, no siempre se obtiene el tensor con la dimensión requerida, puede ocurrir que se necesite un tensor de dimensión 3 pero los datos se hayan agrupado naturalmente en un tensor de dimensión 2, para solucionar esto se usa la función $reshape()$ en donde explícitamente se indican las nuevas dimensiones que el tensor debe de tener. Así un $2D$ tensor se puede transformar en un $3D$ tensor agregando el número de filas y columnas que cada matriz debe tener así como el número de matrices que se desean.

Más adelante se habla sobre la construcción de los tensores necesarios para realizar el trabajo, primero es necesario hablar sobre la construcción de las redes en *Keras*

5.3.2. Construcción de una red

Gracias a la practicidad de *Keras* se puede construir de manera fácil y rápida un modelo de redes neuronales, para esto se debe tener claro los siguientes puntos

- Capas.
- Optimizador y medida de desempeño.
- Entrenamiento del modelo.

Capas

Los modelos que se crean usando *Keras* son en base de capas, se inicia creando el modelo con la clase $Sequential()$, esta permite agrupar capas en un $tf.keras.Model$ con el comando $add.()$ y estas se irán apilando para formar el modelo.

Entre las capas más comunes y las que se utilizaran en este trabajo se tienen:

- LSTM: Agrega una capa de tipo LSTM.
- *Dropout*: La capa *Dropout* aleatoriamente establece las unidades de entrada en cero, lo que ayuda a evitar el sobreajuste.

- *Dense*: Esta capa aplica al tensor de entrada una función f retornando en el tensor modificando. La función f se puede escribir como $output = activation(dot(W, input)+ b)$. En donde *activation* es la función de activación elegida, W son los pesos de la red, *dot* es el producto punto entre los pesos W y el tensor de entrada *input* y finalmente *bias* es un vector de sesgo creado por la capa.

- *Bidirectional*: Agrega una capa de tipo LSTM bidireccional.

- *BatchNormalization*: Aplica una transformación para que los datos de salida de la capa previa tengan una media cercana a cero y una desviación estándar cercana a 1, es decir, se aplica la siguiente transformación

$$\overline{x_i} = \frac{x_i - \mu_B}{\sigma_B}$$

  En donde μ_B es la media empírica de cada *batch* y σ_B es la desviación estándar empírica.

De manera general una capa se crea de la siguiente manera:

$$model.add(tipo\_de\_capa(units, activation, input\_shape, return\_sequences))$$

donde *el tipo_de_capa* es cualquiera de la antes mencionadas aunque no son las únicas capas; *units* es un entero positivo y representa la dimensión de salida, es decir, si *units*$= x$, el tensor de salida en esa capa tendrá la forma $(numero\_de\_muestras, x)$ donde el $numero\_de\_muestras$ será el número de entradas (número de matrices) que tendrá el tensor que se ingresa a la capa; *activation* es la función de activación elegida; *input_shape* es la forma que deben tener los tensores para ser ingresados en la capa, en el caso de las series de tiempo se mencionó que se usarán $3D$ tensores, así que *input_shape* representará la forma que debe tener cada matriz, es decir el número de columnas y filas por lo que en este caso *input_shape* tiene la forma (x, y) donde x es el número de filas y y el de columnas; finalmente se tiene el parámetro $return\_sequences$ que es binario y por *default* esta definido como

$return\_sequences = False$, si $return\_sequences = True$ se imprimen por cada paso (*time steps*) todos los estados ocultos (*hidden states*), es decir, por cada fila que tenga la matriz imprimirá los datos procesados de cada columna.

Existen muchos otros parámetros como por ejemplo para la capa *Dropout* existe un parámetro llamado *rate* que es la tasa con la que convertirá aleatoriamente los datos en cero. A grandes rasgos los parámetros antes mencionados son los más usados y los que se usan para las tareas de este trabajo.

Optimizador y medida de desempeño

Como se habló en el capítulo 3, existen algoritmos para asignar los pesos a la red de una manera óptima y además de los algoritmos también se tienen métodos para optimizarlos. En el presente código se usa el optimizador *Adam*, este es un método de descenso por gradiente estocástico que se basa en la estimación del primer y segundo momento. Es computacionalmente eficiente, requiere de poca memoria y es buen optimizador para problemas con una gran cantidad de datos o parámetros.

De igual manera en el capítulo 3 se introdujo la idea de las medidas de desempeño que cuantifican que tan bueno es el algoritmo en base a la realización de las tareas, en este caso se trabaja con el error cuadrático medio (ECM) o *mean squared error* (MSE) cuya fórmula en el caso en donde se tienen n observaciones es:

$$\sum_{i=1}^{n} \frac{(Y_i - \hat{Y}_i)^2}{n}$$

donde Y_i es la observación i real que se quiere estimar y $\hat{Y}_i$ es la estimación de la observación Y_i. La operación anterior realiza una comparación entre el valor real y el estimado para obtener cuanto varían, se obtiene por cada observación un "*error*" que se eleva al cuadrado simplemente para obtener valores positivos y finalmente se calcula la media de estos "*errores*".

A pesar de que existen muchas medidas de desempeño, cada una es útil para ciertas tareas, por ejemplo para clasificar, la medida de desempeño puede ser las tasas de error y matriz de confusión; por otro lado cuando se busca predecir, la medida de

desempeño comúnmente usada es el ECM.

Para ingresar al código el optimizador a elegir y la medida de desempeño se usa la función $model.compile(loss, optimizer)$ con *loss* la medida de desempeño y *optimizer* el optimizador elegido.

Entrenamiento del modelo

Una vez se crea el modelo, se eligen las capas a usar, el optimizador y medida de desempeño, se procede a entrenar el modelo con la función $model.fit()$ y al igual que con la función $model.add()$ esta tiene muchos parámetros pero solo se explican los más comunes y que se usaron en el trabajo.

- x Datos de entrada en forma de tensor, contienen los datos con los que se entrena el modelo y corresponden a los datos de las variables explicativas.

- y Datos correspondientes a la variable respuesta, con estos se entrena al modelo y sirven para indicar los valores que se deben predecir, al igual que en el caso anterior, son ingresados en forma de tensores.

- *epochs* Número de veces que los datos serán ingresados al modelo, es decir, el número de iteraciones donde en cada iteración todos los datos son procesados por la red.

- *batch_size* Determina el número de muestras que contiene cada *mini batch*, el número de *mini batch* se obtiene de dividir el número total de muestras entre el número asignado a *batch_size*. Recordando que el número de muestras es la primera entrada de la forma del tensor y en cada *epoch* el modelo procesa todos los *mini batch*

- *validation_data* Son los datos que se usan como conjunto de validación en el modelo, en este caso se deben ingresar tanto los datos de las variables explicativas como de la variable respuesta. La forma de ingresarlos es $(X_{validation}, Y_{validation})$

- *steps_per_epoch* Este parámetro indica el número de *mini batch* que de información en el conjunto de entrenamiento que procesa en cada época o *epoch*,

notemos la diferencia con *batch_ size*, en el primero se eligen los *mini batch* mientras que en el segundo se elige el número de muestras en cada *mini batch*.

- *validation_ steps* Es prácticamente igual al parámetro *steps_ per_ epoch* con la diferencia que se aplica a los datos de validación

- *verbose* Toma los valores 0, 1 o 2 y su función es presentar el progreso del modelo al ser entrenado, es decir, si es cero el modelo se entrena y no imprime nada, si la opción es 1 (la opción por *default*) se imprime el proceso que va teniendo el modelo en cada época, el error tanto de entrenamiento como de validación y el número de *mini batches* que se procesarán, finalmente si la opción es 2 se imprime lo mismo que en el caso $verbose = 1$ con la diferencia que no se va mostrando el progreso, sino que al finalizar cada época se imprime la información.

Para finalizar con este punto es necesario mencionar que tanto *steps_ per_ epoch* como *validation_ steps* son usados cuando se tiene una gran cantidad de información o cuando se usan métodos de remuestreo para generar datos de manera indefinida.

5.4. Aplicación de lo anterior al problema

Una vez divididos los datos en los 3 conjuntos antes mencionados y haberlos escalados se procede a construir los tensores que se usan para entrenar, validar y probar el modelo. Para construirlos se debe observar que la tarea de predecir los siguientes precios corresponde a una tarea realizada por algoritmos de aprendizaje supervisado, ya que se tienen los precios históricos que sirven como variables explicativas (X_i) para obtener la variable respuesta (Y) que es el valor predicho.

Teniendo en cuenta lo anterior, tanto el conjunto de entrenamiento como el de validación se deben dividir de manera que por cada conjunto se tengan dos subconjuntos, uno que contenga las variables explicativas y otro que contenga las variables respuestas; de esta manera el modelo puede ser entrenado al observar los patrones y

relaciones entre las variables explicativas y respuestas; asimismo sirve para validar el modelo y obtener los errores.

Para obtener el conjunto de variables explicativas y respuestas se decidió tomar intervalos de 30 días que representan las variables X_i mientras que el siguiente día es el valor a predecir y es la variable Y_i. Lo anterior se escribe de la siguiente manera $X_i = \{Dia_i, ..., Dia_{i+30}\}$, $Y_i = \{Dia_{i+31}\}\ \forall\ i \in \{1, 2, ..., (longitud\ del\ conjunto - 31)\}$ [Chollet, 2017].

Una representación gráfica de este proceso se puede ver en el esquema de la Figura 5.4.

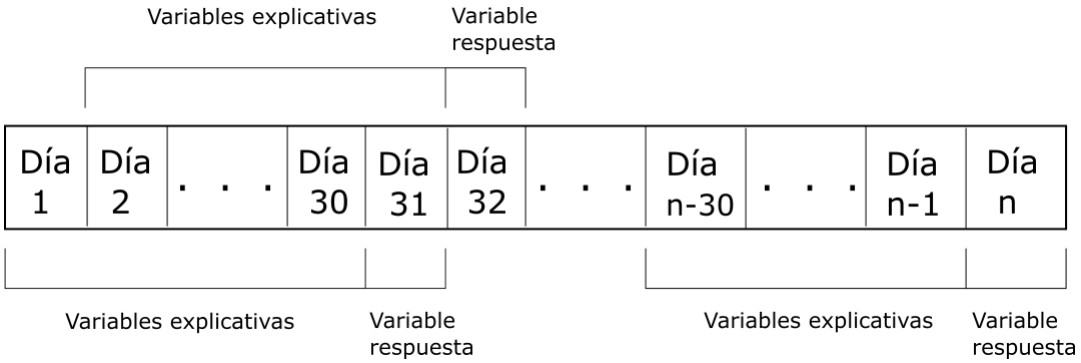


Figura 5.4: Para seleccionar las variables se recorre el conjunto (ya sea de entrenamiento, validación o prueba) en intervalos de 30 días, de manera que este intervalo represente el conjunto de variables explicativas mientras que el día siguiente es la variable respuesta.

Una vez comprendida la forma de dividir los datos en las variables antes mencionadas, se procede a convertir los datos en $3D$ tensores en el caso de los tensores que contienen las variables X_i, la forma o *shape* de estos tensores depende del conjunto con el que se este trabajando, ya sea el de entrenamiento, validación o prueba. En general la forma de los tensores es

$$(longitud\ del\ conjunto - 29, 30, Numero\ de\ variables)$$

Es decir, se tienen "$longitud\ del\ conjunto - 29$" entradas donde cada entrada es una matriz de 30 filas y "$Numero\ de\ variables$" columnas. Para el caso de las variables Y_i como solo contienen el valor a predecir, son $1D$ tensor y tienen la forma

$(longitud\ del\ conjunto - 29,)$.

Una vez se cuentan con los tensores se procede a crear los modelos que se usan más adelante en los experimentos. Para los experimentos se usan las capas LSTM *Dense, Bidirectional* y *BatchNormalization*.

Para los experimentos no se usa la capa *Dropout* ya que su función es evitar el sobreajuste pero en este caso ningún experimento sufre de este problema, por lo que en lugar de ayudar al ajuste del modelo, esta capa alentaría el aprendizaje, dando como resultado un mayor error.

A continuación se listan los demás parámetros usados para crear y entrenar el modelo.

- Función de activación: Tanh o Tangente hiperbólica.
- *units*: 50 para las capas *LSTM y Bidirectional*, 1 para la capa *Dense*.
- *input_shape*: $(30, Numero\ de\ variables)$
- Optimizador o *optimizer*: *adam*.
- Épocas o *epochs* : 60
- *step_per_epoch*: 95
- *validation_steps*: 45

Capítulo 6

Experimentación y resultados

Para el presente trabajo se realizan cuatro experimentos que sirven para comprobar la eficiencia del modelo al momento de predecir los precios futuros del índice S&P/BMV IPC y de las cinco empresas. Estos experimentos son:

- Predecir el siguiente valor del índice usando solamente los precios históricos de dicho índice.
- Predecir el siguiente valor del índice usando los precios históricos de las 5 empresas seleccionadas, así como los del índice.
- Predecir el valor del siguiente día de la acción AMXL.MX, que corresponde a la empresa América Móvil, usando solo los precio históricos de la acción.
- Predecir el valor del siguiente día de la acción AMXL.MX usando sus precios históricos y los del índice.

Para cada experimento se escogen 5 semillas con el fin de reproducir en un futuro los resultados, por cada semilla se reporta el Error Cuadrático Medio (ECM) y para cada semilla se prueban 4 configuraciones distintas de la red y se obtiene el ECM de cada configuración en cada semilla, posteriormente se calcula el error promedio de cada una de las 4 configuraciones para así elegir el mejor modelo para realizar las tareas. Las configuraciones son:

- Capa *LSTM Bidireccional* con capa *BatchNormalization*.

- Capa *LSTM Bidireccional* sin capa *BatchNormalization*.
- Capa *LSTM* con capa *BatchNormalization*.
- Capa *LSTM* sin capa *BatchNormalization*.

6.1. Experimento 1

Usando la información de los valores históricos del S&P/BMV IPC se entrena el modelo para predecir el siguiente valor. En la siguiente tabla se muestran los errores obtenidos para cada semilla y cada configuración de la red, así como el promedio y desviación estándar de los errores por cada configuración. Es importante mencionar que los valores fueron escalados por lo que el error es calculado en base a estos valores modificados.

Semilla	LSTM Bidireccional con BN	LSTM Bidireccional	LSTM con BN	LSTM
9	0.000934	**0.000572**	0.0027	0.000581
2	0.00145	**0.000577**	0.000859	0.000581
4	0.00142	0.000595	0.00125	**0.000571**
5	0.000857	0.000611	0.001672	**0.000591**
7	0.009159	0.00068	0.001815	**0.000648**
Media	0.002764	0.000607	0.001659	0.0005944
Desviación estándar	0.003207	0.000039	0.000619	0.000027

Tabla 6.1: Error cuadrático medio obtenido al usar el modelo en el conjunto *test* con diferentes configuraciones de la red y distintas semillas. Así como la media y desviación estándar.

Como se mencionó en el capítulo anterior, los modelos fueron entrenados y validados con 60 épocas; obteniendo en cada época el error cuadrático medio se pudo realizar la gráfica del lado izquierdo de la Figura 6.1, del lado derecho de la misma figura se tienen las predicciones del modelo en el conjunto de prueba contrastado con los valores reales.

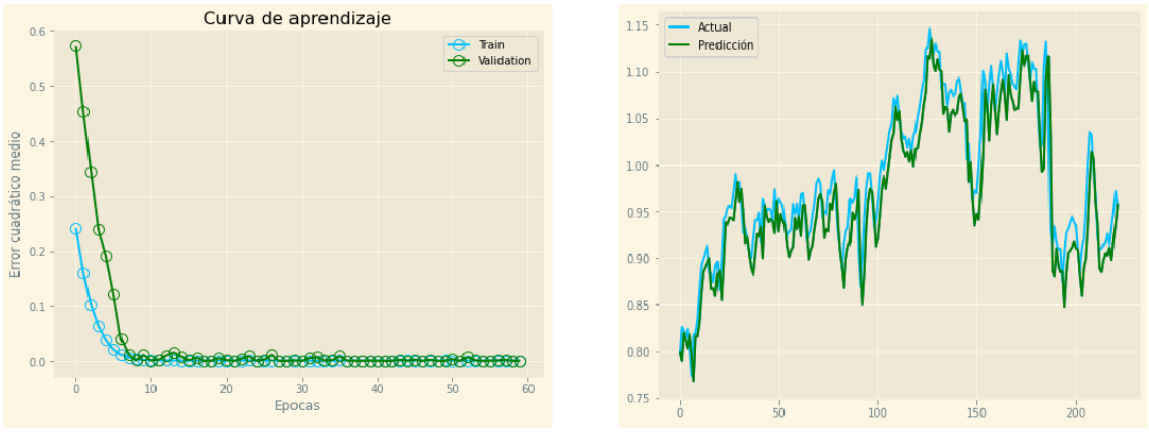


Figura 6.1: Resultados gráficos del experimento 1 usando la semilla = 9 y la configuración *LSTM Bidireccional con BachNormalization.* Del lado izquierdo se tiene la curva de aprendizaje del modelo en los conjuntos *train* y *validation* mientras que del lado derecho se encuentra la comparación entre los datos reales del conjunto *test* las predicciones del modelo en el conjunto.

Además de mostrar los errores en el conjunto de prueba, se muestran los valores predichos para el siguiente día, este valor no se encuentra en los datos usados anteriormente por lo que se realizó una búsqueda en *Yahoo-Finance* de este valor, obteniendo que el nuevo valor es 45,695.10.

Semilla	LSTM Bidireccional con BN	LSTM Bidireccional	LSTM con BN	LSTM
9	45,345.48	45,673.17	46,327.68	45,708.73
2	45,157.36	45589.18	45,865.25	45,543.60
4	45,171.62	45,739.14	45,210.30	45,575.28
5	45,877.82	45,525.15	45,123.33	45,587.24
7	44,231.49	45,471.21	45,126.68	45,479.37

Tabla 6.2: Valores predichos del S&P/BMV IPC para el día 2 de enero del 2017.

Para mostrar gráficamente estas predicciones se tiene la Figura 6.2, solo se muestran los valores predichos en el caso de la semilla 9, esto para observar la mejor configuración de la red neuronal.

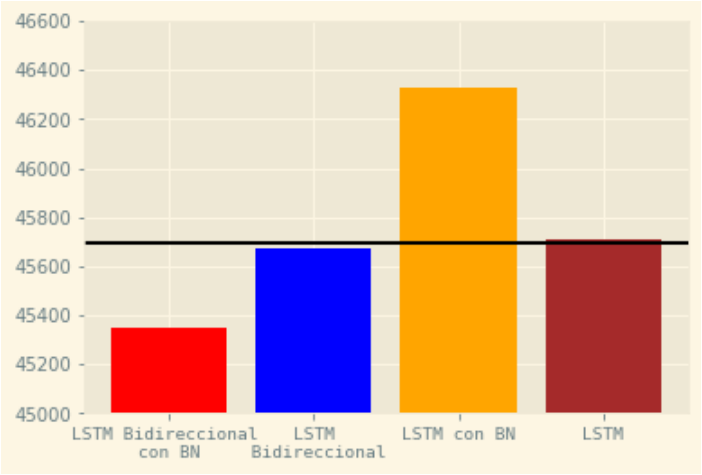


Figura 6.2: Valores predichos del S&P/BMV IPC para el 2 de enero del 2017 para la semilla 9, el valor real se representa por la línea negra.

Se consideró una extensión a este experimento en donde se usa el precio predicho del 2 de enero del 2017 para predecir el precio del índice del 3 de enero del 2017, es decir, se toman los datos del 22 de noviembre del 2016 al 2 de enero del 2017 como variables explicativas para predecir el valor el 3 de enero. Siguiendo este mismo proceso se predicen los valores del 4 y 5 de enero del 2017.

Para simplificar el experimento solo se utiliza la configuración LSTM de la semilla 9, obteniendo así los resultados siguientes.

Día	Valor real	Valor predicho
3 de enero	46,123.36	45,732.95
4 de enero	46,587.74	45,803.87
5 de enero	46,719.99	45,877.21

Tabla 6.3: Valores predichos del S&P/BMV IPC para los días 3, 4 y 5 de enero del 2017 usando los datos hasta el 30 de diciembre del 2016 y los valores predichos de los días anteriores a la predicción.

6.2. Experimento 2

Para este experimento se usa tanto el histórico del índice como el de las 5 empresas seleccionadas. De igual forma que en el experimento anterior, se muestra el error cuadrático medio, la media y desviación estándar para cada caso.

Semilla	LSTM Bidireccional con BN	LSTM Bidireccional	LSTM con BN	LSTM
9	**0.000871**	0.002538	0.001438	0.001267
2	0.008523	**0.00152**	0.013384	0.002381
4	0.00167	**0.001606**	0.002424	0.002027
5	0.00114	0.00254	0.001689	**0.000793**
7	0.000943	0.001627	0.003613	**0.000812**
Media	0.002629	0.001966	0.00451	0.001456
Desviación estándar	0.00296	0.000469	0.004501	0.000644

Tabla 6.4: Error cuadrático medio obtenido con el modelo que considera la información del S&P/BMV IPC y las 5 empresas, usado en el conjunto *test*, además de la media y desviación estándar para cada configuración de la red.

Para evitar acumular una gran cantidad de gráficas (una por cada semilla) se presentarán los resultados obtenidos para la semilla 9 y la configuración *LSTM Bidireccional con BN*, es importante mencionar que, a pesar de que en cada semilla se obtienen resultados distintos, en general el comportamiento de la curva de aprendizaje es similar tanto en las 5 semillas como en las 4 configuraciones. Es claro que las predicciones obtenidas del modelo varían mucho de acuerdo al error reportado en la Tabla 6.4

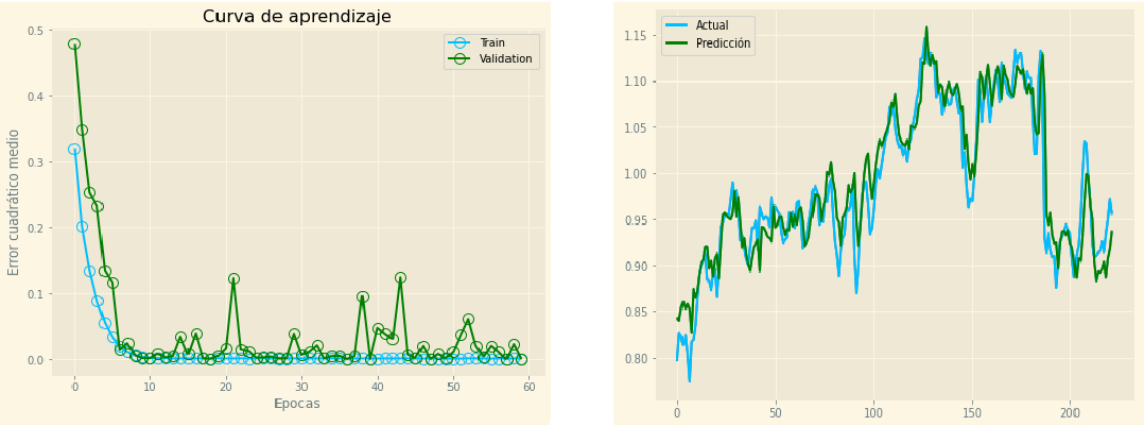


Figura 6.3: Resultados gráficos del experimento 2 usando la semilla = 9 y la configuración *LSTM Bidireccional con BachNormalization*. Del lado izquierdo se tiene la curva de aprendizaje del modelo en los conjuntos *train* y *validation* mientras que del lado derecho se presentan las predicciones del modelo en el conjunto *test*.

Finalmente para este experimento se muestran los valores predichos para el si-

guiente día de acuerdo a las semillas y a las diferentes redes consideradas. Recordando que el valor que se busca predecir es 45,695.10.

Semilla	LSTM Bidireccional con BN	LSTM Bidireccional	LSTM con BN	LSTM
9	45,058.06	45,447.12	45,975.86	45,457.45
2	46,269.28	44,783.65	43,845.81	45,184.30
4	44,645.08	45,435.55	44,812.12	45,151.00
5	45,016.30	44,939.81	45,201.49	45,185.60
7	44,989.55	44,953.95	46,386.33	45,440.55

Tabla 6.5: Valores predichos del S&P/BMV IPC para el día 2 de enero del 2017.

De la tabla anterior se usaron los valores predichos de la semilla 9 para graficar los valores predichos junto al valor real. Al igual que en el experimento 1, se obtiene una gráfica que muestra lo cercano que están los valores predichos por las configuraciones LSTM y LSTM Bidireccional y el valor real, sin embargo es importante notar que el modelo LSTM Bidireccional con BN predijo un valor más cercano inlcuso que el de la red LSTM a pesar de tener un error más grande en el conjunto de prueba.

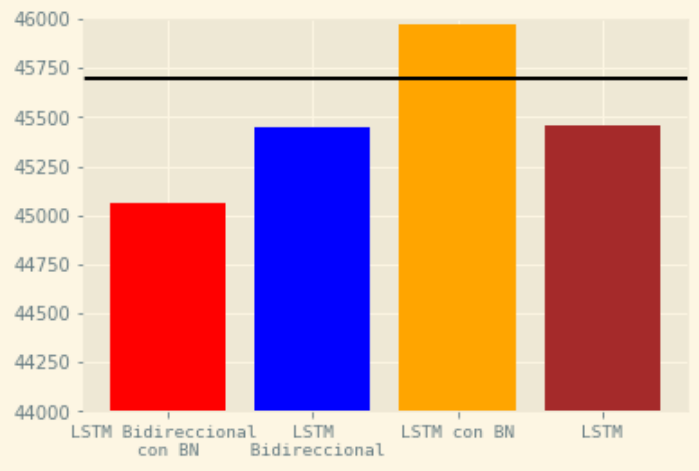


Figura 6.4: Valor real y valores predichos del S&P/BMV IPC para el 2 de enero del 2017 para la semilla 9, en donde la recta negra representa el valor actual.

6.3. Experimento 3

Para este experimento se usan los precios de la acción AMXL.MX con la finalidad de predecir el valor del siguiente día de la acción. Como se mencionó con anterioridad, se realizó una búsqueda de las empresas más representativas del IPC a lo largo de los 7 años de información, obteniendo que América Móvil fue la empresa con mayor ponderación en el IPC teniendo por ejemplo, un peso de 16.63 % en Mayo del 2015. Es por esto que se eligió a esta empresa para predecir sus precios.

A continuación se muestran los errores obtenidos de usar el modelo en el conjunto de prueba para obtener las predicciones. Se muestran además la media y desviación estándar de los errores por cada configuración.

Semilla	LSTM Bidireccional con BN	LSTM Bidireccional	LSTM con BN	LSTM
9	0.001262	**0.001005**	0.001858	0.001446
2	0.001122	0.001007	**0.000976**	0.001026
4	**0.000967**	0.001191	0.001002	0.000985
5	0.00105	0.001167	0.001081	**0.001004**
7	0.001294	0.001132	**0.001056**	0.001137
Media	0.001134	0.0011	0.001195	0.00112
Desviación estándar	0.000124	0.000079	0.000334	0.000171

Tabla 6.6: Error cuadrático medio obtenido del modelo que considera solo la información de la acción AMXL.MX usado en el conjunto *test*, se incluye además la media y desviación estándar para cada tipo de red.

Al igual que en los experimentos anteriores, se muestran las gráficas de la curva de aprendizaje y la comparación entre los valores actuales y los predichos, estas gráficas se encuentran en la figura 6.5. Como se mencionó no se muestran las gráficas de todas las configuraciones y todas las semillas, esto debido a la semejanza entre las gráficas de un mismo experimento.

Es importante notar que, aunque se traten de experimentos distintos, las curvas de aprendizaje muestran como el modelo logra aprender independientemente de los datos y el contexto en el que se aplique el modelo.

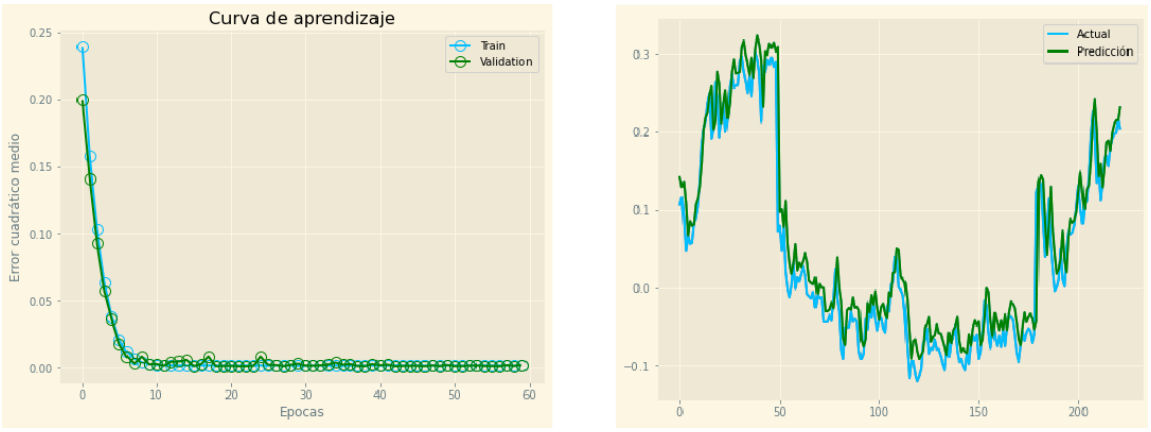


Figura 6.5: Resultados gráficos del experimento 3 usando la semilla = 9 y la configuración LSTM Bidireccional con *BachNormalization*. Del lado izquierdo se muestra la curva de aprendizaje del modelo en los conjuntos *train* y *validation* mientras que del lado derecho se tienen las predicciones del modelo en el conjunto *test*.

En la siguiente tabla se muestran los valores predichos para el siguiente día, en este caso el valor real de la acción de América Móvil es de 13.03, que corresponde al 2 de enero del 2017.

Semilla	LSTM Bidireccional con BN	LSTM Bidireccional	LSTM con BN	LSTM
9	13.16	13.06	12.93	13.08
2	13.08	13.08	13.03	13.07
4	13.03	13.04	12.96	13.06
5	13.09	13.07	13.12	13.04
7	13.17	13.09	13.02	13.10

Tabla 6.7: Valores predichos de la acción AMXL.MX para el día 2 de enero del 2017.

Se observa en la gráfica 6.6 los valores predichos de los distintos modelos de redes usando la semilla 9. A diferencia de las 2 gráficas anteriores en donde los valores parecían estar más cerca del valor real pues las barras parecen acercarse a la recta negra, en este caso podría parecer que están más alejados y por tanto que el modelo tiene un gran error al predecir, pero lo que ocurre es que mientras en los dos experimentos anteriores, los valores predichos más cercanos distaban del real por 3.48 y 4.8 respectivamente, en este caso el valor predicho más cercano dista del real por 0.04 pesos.

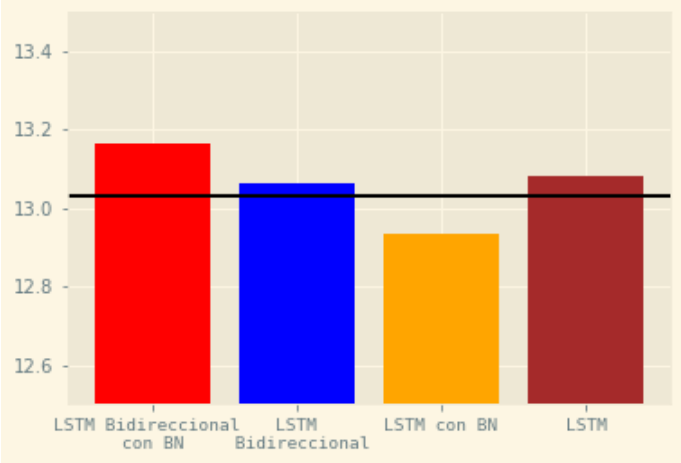


Figura 6.6: Valor actual y valores predichos de la acción AMXL.MX para el 2 de enero del 2017 para la semilla 9, el valor real esta representado por la recta negra.

6.4. Experimento 4

Finalmente en este experimento se usan los precios de América Móvil y del S&P/BMV IPC para predecir los valores de la acción. Como se mencionó, la acción AMXL.MX es la que mayor representación tiene en el índice en el periodo de tiempo considerado, es por esto que se planteó usar el histórico de precios del índice puesto que tiene mayor información de la acción lo que ayuda al modelo a aprender las tendencias de los precios.

En la siguiente tabla se tiene los errores de las predicciones en el conjunto *test*. Recordando que tanto los precios del índice como de la acción están escalados, lo que produce un error cuadrático medio escalado; sin embargo como se vio antes, esto no afecta ni en los resultados predictivos del modelo ni en la tendencia de los datos. Es importante mencionar que independientemente del experimento que se realice, los datos escalados coinciden en todos los experimentos, es decir, aún cuando en el experimento 1 solo se usen los valores del índice y en el experimento 4 se usen los valores del índice y de la acción AMXL.MX, los valores escalados del índice son los mismos, esto ya que la transformación es aplicada a cada variable de manera independiente, es por esto que se pueden comparar los resultados entre experimentos para observar en que situación el modelo obtuvo mejores resultados.

Semilla	LSTM Bidireccional con BN	LSTM Bidireccional	LSTM con BN	LSTM
9	0.003009	0.001352	0.002391	**0.000971**
2	0.001976	**0.00101**	0.001035	0.001029
4	0.001236	0.001213	0.003654	**0.001139**
5	0.00483	0.001008	**0.000944**	0.00114
7	0.013318	0.000997	0.003874	**0.000985**
Media	0.004874	0.001116	0.00238	0.001053
Desviación estándar	0.004391	0.000143	0.001243	0.000073

Tabla 6.8: Error cuadrático medio obtenido del modelo que considera tanto la información de la acción AMXL.MX como la del índice, el modelo es usado en el conjunto *test*. Asimismo se muestra para cada tipo de red, la media y la desviación estándar del error.

Posteriormente se muestra tanto la curva de aprendizaje como las predicciones y el ajuste que estas tienen a los valores reales del conjunto de prueba. Recordando que el conjunto de prueba tiene más datos de los que se predicen, ya que las predicciones empiezan a partir del dato que ocupa la posición 31 en los datos, teniendo que los primeros 30 datos (o días) solo se usan como variables explicativas, es decir, no se predicen los primeros 30 datos del conjunto.

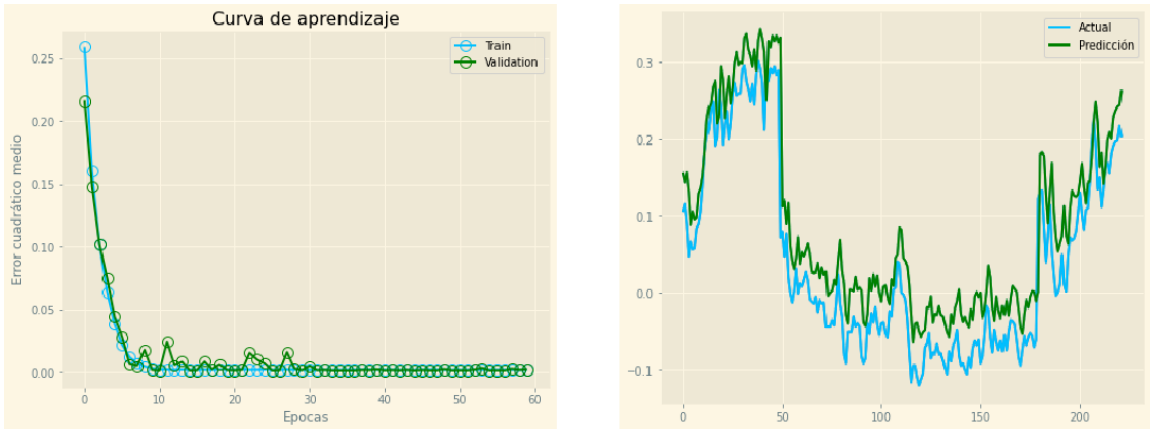


Figura 6.7: Resultados gráficos del experimento 4 usando la semilla = 9 y la configuración LSTM Bidireccional con *BachNormalization*. Del lado izquierdo se tiene la curva de aprendizaje del modelo en los conjuntos *train* y *validation* mientras que del lado derecho se tienen las predicciones del modelo en el conjunto *test*.

De las gráficas anteriores se observa que tanto la curva de aprendizaje del experi-

mento 3 como del 4 se comportan muy parecidos, por lo que, como se ha mencionado, permite no agregar las gráficas correspondientes a las demás semillas y configuraciones de la red, pues a pesar de que se tiene el factor aleatorio, las curvas se comportan parecidas debido al buen aprendizaje del modelo.

A continuación se muestra una tabla que muestra las predicciones de los modelos utilizados en los experimentos, posteriormente se muestra una gráfica de los valores predichos contrastándolos con el valor real.

Semilla	LSTM Bidireccional con BN	LSTM Bidireccional	LSTM con BN	LSTM
9	13.33	13.16	12.68	13.06
2	12.87	13.06	13.07	13.07
4	12.90	13.08	13.34	13.09
5	13.47	13.07	12.94	13.10
7	12.35	13.05	13.40	13.00

Tabla 6.9: Valores predichos de la acción AMXL.MX para el día 2 de enero del 2017.

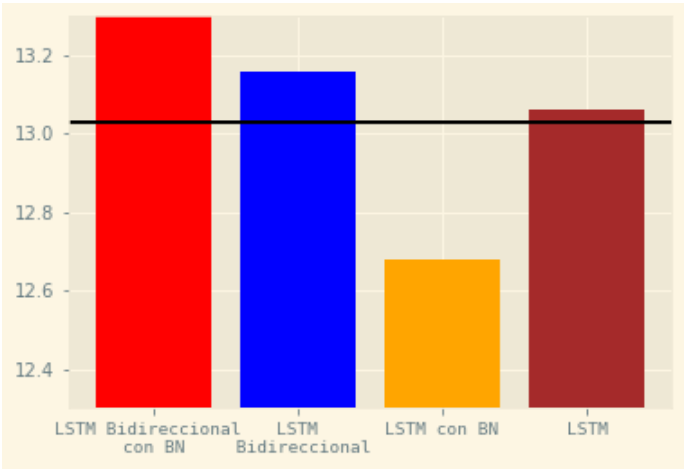


Figura 6.8: Valor actual y valores predichos de la acción AMXL.MX para el 2 de enero del 2017 para la semilla 9, la recta negra representa el valor real de la acción

Al igual que en la gráfica 6.6 podría parecer que las predicciones están mas alejadas que las obtenidas en los primeros dos experimentos, lo que ocurre es que debido a los valores más grandes del índice en contraste con los valores de la acción, ocasionan que al graficarse los estimados se tenga que cambiar la escala de las gráficas.

Capítulo 7

Conclusiones

En esta tesis se exploró la predicción de los valores futuros del índice S&P/BMV IPC así como los precios futuros de la empresa America Movil que es la más representativas del IPC, para esto se usaron los datos comprendidos entre los años 2010 y 2016 como datos de entrenamiento, validación y prueba. Para poder predecir los valores futuros se usaron 4 diferentes configuraciones de redes *Long Short-Term Memory*, se plantearon 4 experimentos donde la finalidad era predecir el siguiente valor del S&P/BMV IPC o de la acción AMXL.MX usando distintos conjuntos de datos para entrenar y validar los modelos. Si bien los resultados obtenidos varían dependiendo del experimento y la configuración de la red, estos muestran que los modelos tienen una capacidad predictiva similar ya que las medias de los errores reportadas son menores a 0.005 lo que indica que en promedio nuestras estimaciones no difieren de los valores reales escalados en más de 0.005. Y aunque los modelos que se tienen no logran predecir con exactitud los precios, logran predecir las tendencias.

Lo primero que debemos mencionar con respecto a los resultados obtenidos de realizar los experimentos es el hecho de que las redes que mejor realizaron la tarea de predecir el precio fueron en primer lugar la red *Long-Short Term Memory* y en segundo lugar la red *Long-Short Term Memory Bidireccional*, esto lo podemos notar en la media obtenida en cada experimento, donde en 3 de los 4 experimentos (experimentos 1,2 y 4) la red LSTM tuvo un en error promedio menor a las otras 3 redes, mientras que en el experimento restante la mejor red en promedio fue la LSTM Bidireccional.

Los resultados muestran que agregar la capa *BatchNormalization* a las redes no tiene un efecto tan favorable al entrenar y probar el modelo; esta capa es en realidad un gran avance en el aprendizaje profundo ya que permite un mejor entrenamiento, por lo que puede parecer extraño que al aplicarse a nuestro problema los resultados no mejoren a comparación de las redes en donde no se usa esta capa. Esto se debe principalmente a dos razones, la primera esta relacionada con el tamaño de los *batch*, ya que como se vio antes, la transformación que aplica esta capa usa la media y desviación empírica de los *batch*, y sabemos que mientras más grande sea el tamaño del *batch*, contendrá un mayor número de datos y será mas precisa la estimación de la media y desviación empírica a la real; en nuestro caso el tamaño del *batch* es de 95 que si bien no es un número pequeño, influye en el calculo empírico de las medidas. La segunda razón tiene que ver con la arquitectura de la red, ya que se esta investigando el uso de la capa *BatchNormalization* en redes recurrentes, y a pesar de que no se cuenta con suficiente información experimental, algunas investigaciones sugieren que aplicar esta capa a las RNN perjudican el aprendizaje debido a la dificultad de aplicar la normalización entre *timesteps* o pasos de tiempo.

Recordando lo mencionado, la red LSTM tuvo en promedio los mejores resultados en la mayoría de los experimentos, superando a la red Bidireccional, que como se mencionó es una configuración que procesa a mayor detalle los datos. La red Bidireccional obtuvo resultados casi tan buenos sin embargo solo en un experimento resultó ser mejor que la red LSTM, esto nos indica que para las series de tiempo usadas no hace mucha diferencia procesarlos como lo hace la red Bidireccional. Debemos mencionar que para el caso de predicciones de valores de series de tiempo, la eficiencia de la red Bidireccional en comparación de la LSTM variará dependiendo de los datos que se tengan.

Como se ha mencionado antes, por cada experimento y por cada una de las 4 configuraciones de redes se obtuvieron diferentes errores que se desearían comparar con los obtenidos en otros trabajos de investigación con el fin de observar qué tan bien se ajustaron los modelos y cuál es su capacidad predictiva en relación a otros modelos, sin embargo no se encontraron investigaciones que realicen los mismos experimentos

que los del presente trabajo y por lo tanto las comparaciones que se hagan entre estos resultados y los de otros trabajos solo servirán para tener una idea del rango en el que pueden estar los errores.

En la tabla 7.1 se observa el contraste entre el menor error obtenido del experimento 3 usando la red LSTM que corresponde al resultado de la semilla 4 y el error reportado de predecir el valor futuro de la acción de Nike en el artículo titulado *Stock Market Prediction Using LSTM Recurrent Neural Network* que plantea usar una red LSTM [Moghar and Hamiche, 2020].

	Predicción AMXL.MX	Predicción NKE
Error	0.000985	0.001

Tabla 7.1: Comparación entre el menor error obtenido del modelo LSTM del presente trabajo al predecir el siguiente valor de la acción AMXL.MX y el error reportado en el artículo que predice el siguiente valor de la acción NKE.

Aún cuando se tomó el error más pequeño del experimento 3 usando la red LSTM, se observa que todos los errores presentados en la Tabla 6.6 están cercanos al 0.001 obtenido en el artículo antes mencionado, lo que nos indica que el modelo construido es capaz de predecir de la misma manera que otros modelos como el del artículo.

De la misma manera en la Tabla 7.2 se observa la comparación entre el error más pequeño obtenido en el experimento 1 que corresponde a la red LSTM en la semilla 4 y el valor reportado en el artículo *Particle swarm optimization of ensemble neural networkswith fuzzy aggregation for time series prediction of the MexicanStock Exchange* donde se busca predecir el valor futuro del índice S&P/BMV IPC usando un método híbrido basado en la optimización del enjambre de partículas para diseñar una red neuronal que realice la tarea [Pulido et al., 2014].

	Predicción IPC con redes LSTM	Predicción IPC con método híbrido
Error	0.000571	0.0079875

Tabla 7.2: Comparación entre el error más pequeño presentado en el experimento 1 al predecir el índice S&P/BMV IPC y el error mostrado al predecir el mismo índice usando una red neuronal diseñada con la optimización del enjambre de partículas.

Se puede observar de los resultados mostrados en la Tabla 6.1 que la gran mayoría de los errores son menores al presentado en el artículo mencionado antes y además la media de los errores para cada una de las configuraciones de la red es menor al valor 0.0079875.

De acuerdo a las comparaciones realizadas antes se puede ver que los modelos planteados en este trabajo cuentan con capacidad para predecir los valores futuros de la acción AMXL.MX y el índice S&P/BMV IPC permitiendo tener buenos resultados.

De igual forma se desearía comparar los resultados de las redes LSTM con algunos modelos de series de tiempo como los son los modelos ARIMA, y si bien no se puede realizar una comparación directa para este trabajo, diversas investigaciones muestran que las redes LSTM pueden mejorar las predicciones realizadas por los modelos de series de tiempo. De acuerdo a [Siami-Namini and Namin, 2018] la red LSTM obtiene resultados que en promedio mejoran a los resultados del modelo ARIMA en un 85 %.

Contrastando ahora los resultados de los experimentos 1 y 2 podemos concluir que se obtuvieron mejores resultados al predecir el valor del índice usando solo los valores pasados de este, esto podría ir en contra del concepto de aprendizaje automático ya que a pesar de que estamos agregando mucha más información al modelo, este no puede realizar de mejor manera la tarea. Lo que ocurre en estos casos es que los datos introducidos al modelo aportan muy poca información, lo que resulta en un peor ajuste y pérdida de tiempo pues el modelo tarda más en procesar una mayor cantidad de datos. Es por esto que para el experimento 4 se decidió tomar solo los valores de la acción de América Móvil y los valores del S&P/BMV IPC, disminuyendo la cantidad de información permitiendo que el modelo aprenda y se ajuste mejor a los datos. Sin embargo al comparar los experimentos 3 y 4 podemos observar que los errores obtenidos del experimento 3 en donde se contemplan los datos de la acción AMXL.MX son en promedio menores a los del experimento 4 donde se agregan los precios históricos del índice salvo en el caso de la red LSTM en donde el promedio de los errores es menor en el experimento 4, con una diferencia de 0.000067 con respecto al error de la red LSTM del experimento 3, pero esta mejora en el error no justifica el mayor trabajo de la red al procesar una mayor cantidad de datos. Se puede

concluir entonces que las diferentes configuraciones de redes realizan mejor el trabajo de predecir usando solo la información pasada de lo que se busca predecir, ya sea la acción o el índice.

Finalmente se debe tener en cuenta que la tarea de predecir valores usando datos históricos es muy compleja, como se pudo observar en los experimentos es extremadamente difícil (o incluso imposible) predecir con exactitud valores futuros, sin embargo esto no nos debe desanimar puesto que obtuvimos resultados que se acercan mucho a los valores reales.

Es importante mencionar que, aun cuando los modelos no puedan predecir bien los siguientes valores, en general todos los modelos pueden predecir las tendencias de los precios, y si bien no son capaces de predecir las magnitudes de estas tendencias, es un gran logro el poder predecir tanto los movimientos alcistas como los bajistas. El poder conocer los movimientos de la acción es suficiente información para poder comprar o vender acciones ya que si se cuenta con una acción X y el siguiente día se predice una caída en el precio de la acción, sin importar cual sea la cantidad que disminuye el precio, podremos tomar una decisión más fundamentada sobre vender la acción. De igual manera prediciendo la tendencia de los índices se puede tener una idea sobre la economía del país e incluso sobre las empresas que conforman ese índice.

La implementación de la metodología que nos permitió realizar los experimentos y de donde se obtuvieron los resultados mostrados en el capitulo anterior se encuentra en un repositorio público[1] que consiste en una serie de archivos de *Python* (uno por cada experimento) donde principalmente se usan librerías de *machine learning* como son *Tensorflow* y *Keras*.

7.1. Trabajo futuro

Aún cuando los resultados obtenidos no fueron malos, no encontramos algún resultado que nos impida pensar que estos resultados pueden mejorar. Lo primero que se deberá hacer en el futuro es obtener información importante, ya no de los precios

[1] https://github.com/LuisVicente0901/Prediccion-ML

históricos sino información económica y financiera sobre la empresa o empresas e incluso información del país. Para realizar esto se debe profundizar en el procesamiento de lenguaje natural para crear modelos que logren extraer información importante de las compañías, sus acciones, el mercado financiero y accionario, la estabilidad del país y la inversión tanto local como extranjera del país.

Una vez se cuente con la información importante, se puede crear un modelo que junte tanto los precios históricos como la información extraída de manera que el modelo tenga una visión más general no solo de la tendencia de los precios sino del comportamiento del mercado. Un primer paso puede consistir en asignar el valor de 1 a la información que refleje un aspecto positivo de la acción (ya sea un incremento en la inversión a la compañía, un mejor posicionamiento en la industria, una mejora en las finanzas del país, etc) y por otro lado asignar un 0 a los aspectos negativos que pueden hacer bajar de precio a la acción. Después de haber convertido la información a una variable binaria, se crearía un modelo que prediga el valor futuro usando los precios y la variable binaria.

Para mejorar aún más el modelo se podría considerar convertir la información en una variable numérica no binaria que refleje la mayor o menor importancia de la información al asignarle valores más grandes o más pequeños. Se podría comenzar entrenando el modelo con noticias e información que uno mismo descargue de Internet y después automatizar el modelo de manera que busque de manera autónoma la información en Internet y la extraiga.

Llevando el trabajo un paso más adelante, se puede pensar en aplicar el aprendizaje por refuerzo a la tarea de predecir los valores futuros de las acciones, de manera que se cree un agente que en cada tiempo t analice la fluctuación del precio y tome la decisión de comprar o vender las acciones de la empresa en base al valor que se tuvo en el tiempo $t-1$ y en los resultados obtenidas del modelo predictivo para el tiempo $t+1$.

Para concluir con el trabajo se debe recordar el buen trabajo que realizaron los modelos predictivos, ya que sin importar si era predecir el valor del índice o de la acción, los diferentes modelos obtuvieron resultados favorables para la predicción de

al menos la tendencia. Si bien los resultados obtenidos son suficientes para tomar elecciones de manera fundamentada, podemos ir mejorando los modelos empleados e incluso emplear muchos otros algoritmos de aprendizaje automático para realizar la tarea de predicción y obtener mejores resultados. A pesar de la dificultad que el mercado accionario presenta para predecir los valores futuros, los modelos de redes LSTM cumplieron con el objetivo de predecir.

Referencias

[Bianchini et al., 2013] Bianchini, M., Maggini, M., and Jain, L. C. (2013). *Handbook on Neural Information Processing*. Springer.

[Bodie and Merton, 1998] Bodie, Z. and Merton, R. C. (1998). A conceptual framework for analyzing the financial environment. *The Global Financial System: A Functional Perspective*.

[Bolsa Mexicana de Valores, 2021] Bolsa Mexicana de Valores (2021). Tipos de índices. https://www.bmv.com.mx/es/Grupo_BMV/Tipos_de_indices.

[Chan et al., 2019] Chan, R. H., Guo, Y. Z., Lee, S. T., and Li, X. (2019). *Financial Mathematics, Derivatives and Structured Products*. Springer.

[Chollet, 2017] Chollet, F. (2017). *Deep Learning with Python*. Manning Publications, 1 edition.

[Coşkun et al., 2017] Coşkun, M., Yıldırım, O., Uçar, A., and Demır, Y. (2017). An overview of popular deep learning methods. *European Journal of Technique (EJT)*, 7:165 – 176.

[Dasgupta and Nath, 2016] Dasgupta, A. and Nath, A. (2016). Classification of machine learning algorithms. *International Journalof Innovative Research in Advanced Engineering*, 3(3):6–11.

[Diebolt and Haupert, 2019] Diebolt, C. and Haupert, M. (2019). *Handbook of Cliometrics*. Springer Reference, 2 edition.

[Du and Swamy, 2014] Du, K.-L. and Swamy, M. (2014). *Neural Networks and Statistical Learning*. Springer.

[García Padilla, 2018] García Padilla, J. R. (2018). Los mercados financieros en méxico. http://educa.banxico.org.mx/banxico_educa_educacion_financiera/blog-24-mercados-financieros.html.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Hassoun, 1989] Hassoun, M. H. (1989). Dynamic heteroassociative neural memories. *Neural Networks*, 2(4):275–287.

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty Fuzziness and Knowledge-Based Systems*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[James et al., 2017] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2017). *An Introduction to Statistical Learning: With Applications in R*. Springer.

[Jia et al., 2019] Jia, M., Huang, J., Pang, L., and Zhao, Q. (2019). Analysis and research on stock price of lstm and bidirectional lstm neural network. *International Conference on Computer Engineering, Information Science & Application Technology*.

[Kim et al., 2018] Kim, K., Aminanto, M. E., and Tanuwidjaja, H. C. (2018). *Network Intrusion Detection Using Deep Learning: A Feature Learning Approach*. Springer.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Education.

[Moghar and Hamiche, 2020] Moghar, A. and Hamiche, M. (2020). Stock market prediction using lstm recurrent neural network. *Procedia Computer Science*, 170:1168–1173. The 11th International Conference on Ambient Systems, Networks and Tech-

nologies (ANT) / The 3rd International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.

[Pulido et al., 2014] Pulido, M., Melin, P., and Castillo, O. (2014). Particle swarm optimization of ensemble neural networks with fuzzy aggregation for time series prediction of the mexican stock exchange. *Information Sciences*, 280:188–204.

[Rebala et al., 2019] Rebala, G., Ravi, A., and Churiwala, S. (2019). *An Introduction to Machine Learning*. Springer.

[Reséndiz Puente, 2014] Reséndiz Puente, F. (2014). El análisis financiero de los seguros privados y el impacto en el sistema financiero mexicano (2008-2013). http://132.248.9.195/ptd2014/junio/0714390/Index.html.

[Ruder, 2017] Ruder, S. (2017). An overview of gradient descent optimization algorithms.

[Secretaría de Hacienda y Crédito Público, 2017] Secretaría de Hacienda y Crédito Público (2017). ¿qué hacemos? https://www.gob.mx/shcp/que-hacemos.

[Siami-Namini and Namin, 2018] Siami-Namini, S. and Namin, A. S. (2018). Forecasting economics and financial time series: Arima vs. lstm.

[Taroon et al., 2020] Taroon, G., Tomar, A., Manjunath, C., Balamurugan, M., Ghosh, B., and Krishna, A. V. N. (2020). Employing deep learning in intraday stock trading. *2020 Fifth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pages 209–214.