UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

ESCUELA DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INDUSTRIAL Y MECÁNICA

OPTIMIZACIÓN DE LA ESTRUCTURA DE REDES NEURONALES PARA PRONÓSTICOS

TESIS QUE, PARA COMPLETAR LOS REQUISITOS DEL PROGRAMA DE HONORES PRESENTA EL ESTUDIANTE

MIGUEL ÁNGEL GOYTIA MORÚA

ID 163293

DR. JUAN ANTONIO DÍAZ GARCÍA

SAN ANDRÉS CHOLULA, PUEBLA. OTOÑO, 2023

TESIS QUE, PARA COMPLETAR LOS REQUISITOS DEL PROGRAMA DE HONORES PRESENTA EL ESTUDIANTE MIGUEL ÁNGEL GOYTIA MORÚA
ID 163293

DIRECTOR DE TESIS

Dr. Juan Antonio Díaz García

PRESIDENTE DE TESIS

Dra. Daniela Cortés Toto

SECRETARIO DE TESIS

Dra. Dolores Edwiges Luna Reyes

Agradecimientos

En primer lugar les agradezco a mis padres por su amor, esfuerzo y entrega durante toda mi vida. Ellos son los que han me han impulsado y han hecho posible el perseguir mis sueños. También, a mi hermana Majo, que ha sido una guía y apoyo incondicional en mi desarrollo.

Le agradezco muy profundamente a mi tutores por su tiempo, su paciencia y la confianza depositada en mi para realizar este trabajo. Gracias por todos los consejos y dedicación que fueron indispensables para éste trabajo y mi formación profesional.

A los docentes que tuve el placer de conocer en el transcurso del proceso universitario, quiero agradecerles por todas las enseñanzas y el conocimiento compartido con pasión y rigor.

Agradecerles a mi familia, amigas y amigos, los cuales han estado conmigo en los buenos momentos y en las etapas difíciles. Gracias por las historias y las memorias que hemos construido juntos. Y en especial a mi prima Ana Paula, que me ha acompañado en la vida y también en este proyecto de tesis.

Índice

Índice de tablas

Índice de figuras

Índice de algoritmos

Resumen

La predicción de la demanda es crucial para que la planeación en la cadena de suministro sea eficiente y rentable. Una red neuronal recurrente puede ser un modelo de pronósticos con alta precisión. Determinar la estructura óptima de una red neuronal recurrente (RNN) para un modelo de pronóstico preciso presenta un desafío significativo en la práctica. Esto se debe a la interacción compleja entre las capas ocultas y las neuronas en cada una de estas capas. Se desarrolló un algoritmo genético para determinar la estructura de una red neuronal recurrente con celdas de memoria de corto-largo plazo (LSTM), para realizar pronósticos con un menor error, en comparación con otros modelos existentes, tanto estadísticos como de aprendizaje automático y por conjuntos. El algoritmo genético propuesto demostró ser robusto para diseñar estas estructuras en comparación con una búsqueda puramente aleatoria, tomando el mismo tiempo de ejecución. En seis series de tiempo distintas, los modelos de RNN diseñados tuvieron menor error de pronóstico (MAPE) en los datos de prueba que los modelos estadísticos. Los modelos de aprendizaje automático fueron los mejores para pronosticar los datos de prueba. En cuanto a la comparación mediante *backtesting*, los modelos estadísticos mostraron tener menor error de forma consistente en los distintos horizontes de tiempo. Esto se debe principalmente a la naturaleza de los modelos. No obstante, esas diferencias no resultaron ser significativas en algunos casos.

Palabras clave: Redes Neuronales Artificiales, Algoritmo Genético, Pronósticos.

1. Introducción

La cadena de suministro involucra un conjunto de procesos y organizaciones que colaboran para proveer productos y servicios a los clientes finales (Ivanov, Tsipoulanidis, y Schönberger, 2019). La planeación de la cadena de suministro coordina estas operaciones, logrando el intercambio de recursos e información. Una planeación ejecutada correctamente genera eficiencia en la adquisición de materias primas, en la programación de la producción, en el manejo de inventarios y en las consecuentes actividades logísticas de distribución. Sin embargo, el éxito de la planeación en la cadena de suministro depende de la precisión con la que se estime la demanda futura de los clientes. Si el producto no está disponible para que los clientes lo compren porque está agotado, las empresas pierden ingresos y podrían llegar a perder clientes ante un competidor. Por el contrario, mantener una gran cantidad de inventario no utilizado incurre en costos innecesarios de producción, almacenamiento, o la posibilidad de pérdida del producto si es perecedero (Gil, Rodriguez-Marin, y Montoya, 2020). Además, esto afecta a todas las organizaciones que conforman la cadena. El efecto látigo (*bullwhip effect* en inglés) es el incremento en la fluctuación que experimenta la proyección de la demanda a medida que se aleja del cliente final (Mejía Villamizar, Palacio León, y Adarme Jaimes, 2013). Esto se debe a que los errores de predicción de la demanda entre las empresas se van acumulando, afectando negativamente los tiempos de respuesta, capacidad de almacenamiento, administración de los costos logísticos, entre otros aspectos (Mejía Villamizar et al., 2013). Si bien, la demanda es incierta y volátil, se pueden hacer estimaciones de ella. Entre mayor precisión se tenga de este dato, menor fluctuación habrá en la cadena y menor será el efecto látigo.

Existen diferentes formas de estimar la demanda. Una de ellas es el utilizar modelos de pronóstico de series de tiempo. Estos se sostienen del supuesto que el pasado se repite, o tan siquiera, que es un buen predictor del futuro. Por lo tanto, los modelos de pronóstico toman los datos históricos, ordenados secuencialmente y con un espacio temporal uniforme entre ellos para hacer pronósticos. A estos datos se les llama series de tiempo. Un modelo de pronóstico aprende a reconocer las características y patrones de la serie siendo analizada para poder determinar que sucederá en el futuro (Ivanov et al., 2019). Los modelos estadísticos, que emplean métodos como el promedio móvil o suavizamiento exponencial, han sido referentes para realizar predicciones. No obstante, la llegada de la inteligencia artificial ha revolucionado el campo de los pronósticos, generando nuevas propuestas de modelo prometedoras. Los métodos de aprendizaje automático y la subcategoría de aprendizaje profundo se utilizan cada vez más, ambos considerados subáreas de la inteligencia artificial (Moroff, Kurt, y Kamphues, 2021). La ventaja de éstos métodos frente a los estadísticos es el análisis automático de patrones y dependencias en los datos históricos de entrada, para posteriormente realizar pronósticos.

Particularmente, las redes neuronales artificiales han resultado ser modelos formidables en cuanto a rendimiento y flexibilidad para resolver distintos tipos de problema. La redes neuronales recurrentes son utilizadas específicamente para pronosticar datos secuenciales, como las series de tiempo. Sin embargo, existe un problema en cuanto a la construcción de estos modelos. Si bien, deducen los parámetros para pronosticar la serie de tiempo de forma automática durante el entrenamiento, los hiperparámetros del

modelo deben definirse antes de implementar el modelo. Los hiperparámetros de una red neuronal pueden separarse en dos categorías: de aprendizaje y de estructura del modelo. Para los que controlan el aprendizaje durante el proceso de entrenamiento, existen diversos métodos para obtener un rendimiento aceptable. Sin embargo, no hay un método contundente para establecer los hiperparámetros que diseñan la estructura de una red neuronal artificial. En la literatura, se han propuesto varias recomendaciones prácticas para establecer sus valores. Bengio (2012), que propuso varias de estas recomendaciones prácticas, concluyó que a pesar de décadas de trabajo experimental y teórico sobre redes neuronales artificiales, todavía quedaba mucho por hacer. Además, indicó que las recomendaciones debían tomarse como una guía, que debía ser probada y desafiada. En sí, las recomendaciones son un punto de partida, donde se procede a realizar varias prueba y error hasta encontrar un rendimiento aceptable. Un método establecido en el área de aprendizaje automático es el implementar validación cruzada. Del conjunto de entrenamiento, se toman conjuntos más pequeños de validación, donde se hacer pruebas con diversos valores de hiperparámetros escogidos por el usuario. Aquellos que resulten mejores para predecir el conjunto de validación, se utilizarán posteriormente en datos no vistos por el modelo para concluir la eficacia del mismo. En conclusión, esto conlleva a que la búsqueda de hiperparámetros se encuentra entre el arte y la ciencia, dependiendo de la habilidad del usuario junto con los recursos computacionales y tiempo dedicado para hacer múltiples pruebas.

Dado que determinar el valor de los hiperparámetros es un problema de búsqueda de soluciones, el objetivo de este trabajo es proponer e implementar una metaheurística que determine la estructura de una red neuronal recurrente que genera pronósticos a partir de una serie de tiempo. Específicamente, se propone un algoritmo genético que de forma automática, determina el mejor conjunto de hiperparámetros referentes a la estructura del modelo. El algoritmo propuesto se comparará con una búsqueda aleatoria para determinar su eficacia en encontrar soluciones al problema. Una vez determinada la estructura del modelo por el algoritmo, se pondrá a prueba para realizar pronósticos, y se comparará con otros modelos existentes, tanto estadísticos como de aprendizaje automático, para evaluar su desempeño. Se utilizarán diferentes series de tiempo con características distintas para establecer la robustez del método.

2. Justificación

La determinación de los hiperparámetros de estructura de una red neuronal es un problema cuya solución ha permanecido en el ámbito entre el arte y la ciencia. Se propone un método científico y heurístico para que diseñe la estructura del modelo. El propósito final es proporcionar a los responsables de la planificación en organizaciones que participan en cadenas de suministro, una herramienta que permita determinar la estructura de estos modelos y generar pronósticos con un menor margen de error, sin requerir de ajustes manuales ni una inversión significativa de tiempo a la prueba y error. Esto, a su vez, conlleva a una mejora sustancial en la planificación de la cadena de suministro y, teóricamente, a una optimización de las operaciones y una planificación estratégica a largo plazo más efectiva.

Este enfoque beneficia a todos los involucrados dentro de la cadena de suministro. Los clientes experimentarán una mayor satisfacción al ver cumplidas sus expectativas, asegurándose de que ningún producto o servicio quede fuera de su alcance. Los inversionistas y propietarios de negocio observarán un aumento en sus rendimientos económicos gracias a una gestión más eficaz. Por otro lado, los planificadores obtendrán resultados más precisos, lo que les permitirá asignar de manera más eficiente los recursos disponibles.

Además, este método se puede utilizar para definir estructuras de un modelo de red neuronal para otros tipos de problemas que involucren datos secuenciales, ampliando así su utilidad y relevancia en diversos contextos.

3. Objetivos

- **Objetivo General**

  Proponer e implementar una metaheurística que determine la estructura de una red neuronal recurrente que realice pronósticos de series de tiempo con un menor error, en comparación con otros modelos de pronóstico.

- **Objetivos Específicos**

  - Desarrollar un algoritmo genético que diseñe la arquitectura de una red neuronal recurrente.
    - Establecer una codificación de las soluciones.
    - Decretar una función de aptitud.
    - Implementar métodos de los operadores genéticos.
  - Comparar el rendimiento del algoritmo genético con una búsqueda aleatoria.
    - Desarrollar una búsqueda aleatoria de estructuras posibles.
    - Comparar su rendimiento contra el tiempo de ejecución.
  - Implementar predicciones con la estructura propuesta del mejor modelo encontrado con distintas series de tiempo.
  - Comparar el desempeño del modelo contra el de otros modelos de predicción con los datos de prueba y mediante *backtesting*.

4. Marco Teórico

1. La Cadena de Suministro

Una cadena de suministro es una red de procesos y organizaciones que colaboran a lo largo de todas las operaciones para adquirir materias primas, convertirlas en productos terminados, y entregarlos a los consumidores finales (Ivanov et al., 2019). La gestión de la cadena de suministro (*SCM* por sus siglas en inglés) se refiere a la integración y coordinación de materiales, información y flujos financieros entre departamentos y entre empresas, para transformar y utilizar los recursos de la cadena de suministro de la manera más efectiva (Ivanov et al., 2019). La SCM es un componente clave en cualquier organización y es responsable de balancear la demanda y la oferta a lo largo de toda la cadena de suministro. Por lo tanto, la tarea administrativa principal es la toma de decisiones para lograr este balance.

1.1 Pronóstico de la demanda

Pronosticar la demanda futura es una de las decisiones fundamentales en la SCM, dado que las ventas futuras representan el dato inicial para la planeación de la producción, transporte, compras e inventario. Incluso, muchas decisiones estratégicas, como la localización de instalaciones, gestión de ingresos, y diseño del proceso, interactuan con los pronósticos de demanda (Ivanov et al., 2019).

Las predicciones inexactas de la demanda provocan que la cadena de suministro tenga una mayor fluctuación, y esta volatilidad irá aumentando progresivamente (Liu, 2015). Por ende, mejorar la precisión de los pronósticos y minimizar la incertidumbre de la SCM es un tema de gran importancia.

2. Pronóstico de series de tiempo

Una serie de tiempo consiste en datos que se recopilan, registran u observan a lo largo de incrementos sucesivos de tiempo (Hanke y Wichern, 2005). La teoría de pronóstico está basada en la premisa de que la historia puede ser utilizada para hacer predicciones sobre el futuro. Para las series de tiempo, existe la idea de que es posible identificar patrones en los valores históricos e implementarlos con éxito en el proceso de pronosticar valores futuros (Petropoulos et al., 2022). El predecir un valor esperado en un punto futuro se conoce como predicción puntual.

Un método o modelo de pronóstico se define como una predeterminada serie de pasos que produce predicciones de ciertos periodos en el futuro (Petropoulos et al., 2022). El diseño de un modelo de pronóstico se enfoca en la aplicación de algoritmos que capturan los patrones y relaciones dinámicas existentes en los datos, y unifican estos elementos en una estructura matemática robusta y potencialmente flexible (Petropoulos et al., 2022). Los métodos de pronóstico univariados generan pronósticos de una serie de tiempo utilizando los valores históricos de la única variable disponible, mientras que los métodos multivariados o causales integran valores históricos de varias series de tiempo (Petropoulos et al., 2022).

Existe una amplia gama de modelos de pronósticos disponibles. Los modelos estadísticos tradicionales siguen siendo eficaces y altamente utilizados en las organizaciones. No obstante, nuevas metodologías se han desarrollado debido al

surgimiento de la inteligencia artificial (IA) y el incremento en el poder computacional.

En términos generales, la IA comprende cualquier técnica que permita a una computadora imitar el comportamiento humano y reproducir o sobresalir sobre la toma de decisiones para resolver tareas complejas de forma independiente o con mínima intervención humana (Janiesch, Zschech, y Heinrich, 2021).

El aprendizaje automático (*machine learning* en inglés), es visto como un subconjunto de la IA. Los algoritmos de aprendizaje automático iterativamente construyen un modelo matemático basado en datos de muestra, conocidos como datos de entrenamiento, para poder hacer predicciones o decisiones sin ser explícitamente programados para realizar la tarea (Khanzode y Sarode, 2020). Los algoritmos aprenden iterativamente, permitiendo a las computadoras encontrar información oculta y patrones complejos de forma autónoma (Janiesch et al., 2021).

Además, el aprendizaje profundo (*deep learning* en inglés) es un subconjunto del aprendizaje automático. Estos algoritmos de aprendizaje automático tienen grandes estructuras que imitan el funcionamiento del cerebro humano, llamadas redes neuronales artificiales (Janiesch et al., 2021).

2.1 Modelos Estadísticos

La aplicación de métodos estadísticos basados en autorregresión y promedios móviles se consideran el estado del arte para el modelado y predicción de series temporales desde hace más de medio siglo (Petropoulos et al., 2022). Los algoritmos que implementan estos métodos suponen que los datos siguen una distribución conocida. Con base en esa información, se definen los parámetros para ajustar el modelo a los datos y optimizar las predicciones. Sin embargo, el uso de estos métodos paramétricos requiere conceptos matemáticos sofisticados, así como una amplia experiencia técnica para el establecimiento de los parámetros del modelo (Petropoulos et al., 2022).

El uso de fundamentos de estadística descriptiva puede guiar la definición de parámetros. En varios casos, las funciones basadas en autocorrelación automatizan esta tarea (Petropoulos et al., 2022). Los resultados pueden interpretarse mediante correlogramas y mediante la aplicación de técnicas para obtener argumentos de entrada a partir de criterios de minimización de información, lo que penaliza al modelo por el número de parámetros requeridos para su ajuste (Petropoulos et al., 2022)

2.2 Modelos de aprendizaje automático

En las últimas dos décadas, con el auge del proceso de minería de datos, existe un creciente interés en la adaptación de métodos de aprendizaje automático (Petropoulos et al., 2022). Los métodos de predicción de aprendizaje automático, a diferencia de los modelos estadísticos, describen las propiedades de los datos sin el previo conocimiento de su distribución. Debido a que no dependen explícitamente de parámetros para modelar el comportamiento del fenómeno, estos métodos son más sencillos de ajustar y muestran un rendimiento confiable incluso cuando se aplican a procesos complejos y series altamente no lineales (Petropoulos et al., 2022).

Varios estudios han empleado modelos no paramétricos basados en algo-

ritmos de aprendizaje automático para pronosticar series de tiempo. Una de las principales ventajas de este enfoque es que no presupone la distribución de los datos (Petropoulos et al., 2022).

Algunas investigaciones empíricas han demostrado que los algoritmos de aprendizaje automático para la predicción de series de tiempo proporcionan resultados muy competitivos, incluso superando con frecuencia a los modelos estadísticos (Petropoulos et al., 2022).

2.3 Modelos de aprendizaje por conjuntos

El aprendizaje por conjuntos es un enfoque en el que se ajustan dos o más modelos a los mismos datos y se combinan las predicciones de cada modelo. El aprendizaje por conjuntos tiene como objetivo lograr un mejor rendimiento con el conjunto de modelos que de forma individual con cualquier modelo (Schneider y Xhafa, 2022). Esto requiere decidir cómo crear los modelos utilizados en el conjunto y cuál es la mejor manera de combinar las predicciones de los miembros del conjunto (Schneider y Xhafa, 2022).

El aprendizaje por conjuntos es un enfoque útil para mejorar la predictabilidad y reducir la varianza de los pronósticos (Petropoulos et al., 2022). Los conjuntos pueden estar compuestos de varios modelos del mismo tipo con diferentes parámetros, o bien, puede ser un conjunto de modelos distintos. Es por eso, que el aprendizaje por conjuntos puede integrar modelos tanto estadísticos como de aprendizaje automático.

3. Redes Neuronales Artificiales

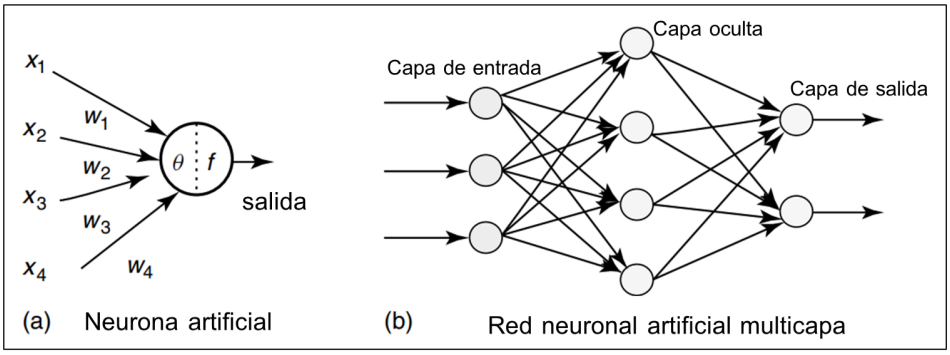
El campo del aprendizaje automático ha generado una variedad de avances notables en algoritmos avanzados de aprendizaje. Uno de estos avances fue la evolución de las redes neuronales artificiales (*ANNs*, por sus siglas en inglés) hacia arquitecturas de redes neuronales cada vez más profundas con capacidades de aprendizaje mejoradas resumidas como aprendizaje profundo (*Deep Learning* en inglés) (Janiesch et al., 2021).

Las redes neuronales artificiales son modelos computacionales inspirados en el funcionamiento del cerebro humano. Están compuestas por capas de nodos interconectados, donde cada nodo (o unidad de procesamiento) simula una neurona (Cicek y Ozturk, 2021). Cada conexión entre nodos tiene un peso que modula la influencia de una neurona con otra. La información fluye a través de la red desde la capa de entrada, donde se introducen los datos, pasa por cada capa oculta hasta llegar a la capa de salida, donde se obtiene el resultado deseado (propagación hacia adelante) (Cicek y Ozturk, 2021). Durante el proceso de entrenamiento, la red ajusta los pesos de las conexiones mediante algoritmos de optimización, con el objetivo de minimizar la diferencia entre las predicciones y los valores reales del conjunto de entrenamiento (propagación hacia atrás) (Aurélien, 2017).

La Figura 1 muestra la arquitectura básica de una ANN, constituida por los tres tipos de capa: entrada, oculta y de salida (Abraham, 2005). En las estructuras más sencillas, el flujo de información va estrictamente en una dirección, desde las unidades de entrada a las de salida (Abraham, 2005). El procesamiento de datos puede extenderse a múltiples capas de unidades, pero no hay conexiones de retroalimentación presentes (Abraham, 2005). Sin embargo, existen otros tipos de

arquitecturas.

Figura 1: Arquitectura de una Red Neuronal



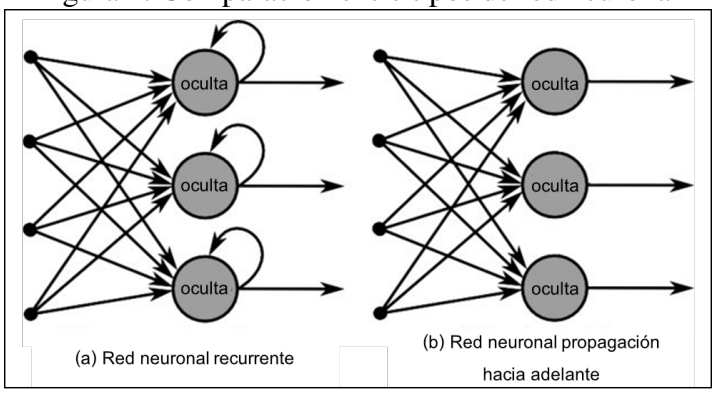
Nota. Adaptado de Abraham (2005).

3.1 Redes Neuronales Recurrentes

Las redes neuronales recurrentes (*RNN*, por sus siglas en inglés), son una clase de redes que pueden analizar datos de series de tiempo. En términos generales, pueden trabajar con secuencias de longitudes arbitrarias, en lugar de con entradas de tamaño fijo (Aurélien, 2017).

Una red neuronal recurrente se parece mucho a una red neuronal de propagación hacia adelante, excepto que también tiene conexiones que apuntan hacia atrás (Aurélien, 2017). Es decir, las neuronas contienen conexiones de retroalimentación (Abraham, 2005). La Figura 2 muestra las diferencias entre una ANN y una RNN.

Figura 2: Comparación entre tipos de red neuronal



Nota. Adaptado de De Mulder, Bethard, y Moens (2014).

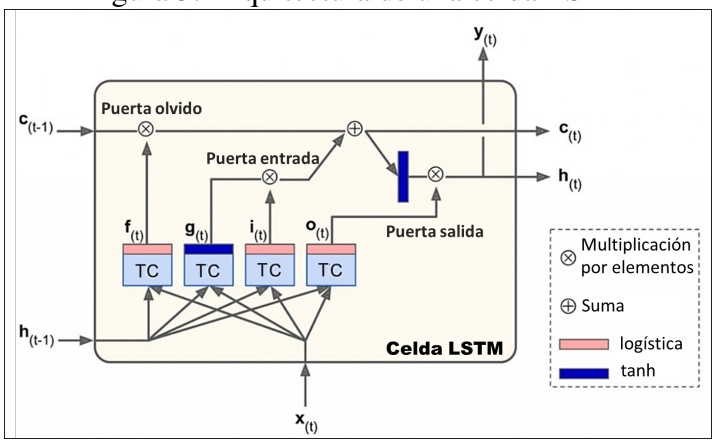
Una RNN puede simultaneamente tomar una secuencia de entradas y producir una secuencia de salida. Por ejemplo, puede tomar de una serie de tiempo los últimos N datos, y producir las predicciones desplazadas un periodo hacia el futuro (Aurélien, 2017).

La característica típica de la arquitectura de una RNN es una conexión cíclica, que permite que la red posea la capacidad de actualizar el estado actual en función de estados pasados y datos de entrada actuales (Yu, Si, Hu, y Zhang, 2019). Estas redes, que consisten en células recurrentes estándar, han tenido un éxito formidable en algunos problemas. A pesar de ese éxito, cuando la brecha temporal entre los datos de entrada y los datos históricos relevantes es grande, las RNN anteriores no son capaces de conectar los datos entre sí (Yu et al., 2019) Para manejar las ”dependencias a largo plazo”, Hochreiter y Schmidhuber (1997) propusieron la memoria de corto-largo plazo (*Long Short-Term Memory* (LSTM), en inglés.

3.1.1 Celdas LSTM

Las celdas LSTM son la razón detrás de los resultados notables de las RNN y, en consecuencia, se han convertido en el foco del aprendizaje profundo. Debido a su poderosa capacidad de aprendizaje, las LSTM funcionan eficientemente y se han utilizado ampliamente en diversos tipos de tareas (Yu et al., 2019). La arquitectura de una celda LSTM básica se muestra en la Figura 3.

Figura 3: Arquitectura de una celda LSTM



Nota. Adaptado de Aurélien (2017).

La idea clave es que la red pueda aprender qué almacenar a largo plazo, qué desechar y qué leer de los datos de entrada. A medida que el estado a largo plazo $c_{(t-1)}$ atraviesa la red de izquierda a derecha, primero pasa por una puerta del olvido, eliminando algunos recuerdos. Posteriormente, agrega algunos recuerdos nuevos mediante la operación de suma (que agrega las memorias que fueron seleccionadas por la puerta de entrada). El resultado $c_{(t)}$ se envía directamente, sin ninguna transformación adicional. Entonces, en cada paso del tiempo, algunos recuerdos se eliminan y otros se agregan. Además, después de la operación de suma, el estado a largo plazo se copia y pasa a través de la función tanh. Luego, la puerta de salida filtra el resultado. Esto produce el estado de corto plazo $h_{(t)}$ (que es igual a la salida de la celda para este paso de tiempo $y_{(t)}$).

En resumen, una celda LSTM puede aprender a reconocer una entrada importante (esa es la función de la puerta de entrada), almacenarla en un estado a largo plazo, aprender a preservarla durante el tiempo que sea necesario (esa es la función de la puerta de olvido), y aprender a extraerla cuando sea necesario (Aurélien, 2017). Esto explica por qué han tenido un gran éxito al momento de capturar patrones a largo plazo en series de tiempo y otros tipos de datos (Aurélien, 2017).

Si se considera la celda LSTM como una caja negra, se verá y se usará de la misma forma que una celda o neurona básica, excepto que funcionará mucho mejor (Aurélien, 2017). El entrenamiento de una red con estas celulas convergerá más rápido y podrá detectar dependencias a largo plazo en los datos (Aurélien, 2017).

3.2 Hiperparámetros de una red neuronal

La flexibilidad de las redes neuronales es también uno de sus principales inconvenientes: hay muchos hiperparámetros que ajustar (Aurélien, 2017). Los hiperparámetros controlan el comportamiento y la estructura de los modelos de redes neuronales. No deben confundirse los hiperparámetros con los parámetros del modelo. Ambas son variables que existen en el aprendizaje automático y los algoritmos de aprendizaje profundo.

Los parámetros son variables cuyos valores se aprenden de los datos y se actualizan durante el entrenamiento (Pramoditha, 2022). En las redes neuronales, los pesos entre neuronas y los sesgos son parámetros. Se ajustan durante el entrenamiento para minimizar el error.

Por otro lado, los hiperparámetros son variables cuyos valores se establecen antes de entrenar el modelo, no se aprenden automáticamente de los datos (Pramoditha, 2022). Por lo tanto, es necesario fijarlos arbitrariamente para construir mejores modelos. No obstante, encontrar los valores óptimos para los hiperparámetros es una tarea compleja (Pramoditha, 2022).

Los hiperparámetros en una red neuronal se pueden clasificar considerando los siguientes criterios: de aprendizaje u optimización y de estructura de red. (Pramoditha, 2022).

Los hiperparámetros de aprendizaje u optimización controlan el proceso de entrenamiento de la red. Algunos de ellos son el tipo de optimizador (algoritmo de optimización), la tasa de aprendizaje, la función de pérdida, el tamaño del lote y el número de épocas de entrenamiento (Pramoditha, 2022). Sin embargo, estos hiperparámetros son más fáciles de ajustar que los de estructura. Los optimizadores como *Adam* adaptan iterativamente la tasa de aprendizaje (Aurélien, 2017). Incluso, no es necesario modificar el número de épocas si se utiliza un criterio de parada anticipada (Aurélien, 2017).

Los hiperparámetros de la estructura de red diseñan la arquitectura del modelo. Hay tres hiperparámetros principales de este tipo (Aurélien, 2017):

- Número de capas ocultas
- Número de neuronas por capa oculta
- Tipo de función de activación

El número de neuronas en las capas de entrada y salida no es un hiper-

parámetro porque está determinado por el tipo de entrada y salida requerida por la tarea (Aurélien, 2017). Por otro lado, establecer los hiperparámetros listados es una tarea compleja, porque con respecto a la estructura, se puede utilizar cualquier arquitectura de red imaginable (Aurélien, 2017).

3.2.1 Puesta a punto de hiperparámetros de estructura de red

Aunque una red neuronal es ampliamente utilizada para resolver problemas de mapeo no lineal, no existe un criterio general para seleccionar la estructura óptima para un problema específico (Leung, Lam, Ling, y Tam, 2003). Se ha comprobado que una red neuronal con solo una capa oculta es un aproximador universal, es decir, puede aproximarse arbitrariamente bien a cualquier mapeo continuo, siempre que haya suficientes neuronas en la capa oculta (Leung et al., 2003). Una ANN que tiene más de una capa oculta, se considera una red neuronal profunda (*DNN*, por sus siglas en inglés) (Aurélien, 2017)

Sin embargo, la capacidad para generar resultados precisos para las entradas de entrenamiento compite con la capacidad de predecir resultados apropiados para entradas desconocidas. Por ejemplo, en el caso de una red neuronal profunda, cuando agregamos algunos nodos en las capas ocultas, la red puede producir resultados más precisos para los datos de entrenamiento, pero también puede dar resultados imprecisos para los datos aún no vistos (Leung et al., 2003). Además, las redes con un número excesivo de parámetros o pesos tienen una mayor probabilidad de alcanzar mínimos locales durante el procedimiento de entrenamiento (Leung et al., 2003). Por lo tanto, es importante encontrar la estructura de red más simple posible, es decir, utilizar el número mínimo de pesos y nodos, sin degradar el rendimiento del modelo (Leung et al., 2003).

Esta búsqueda de la mejor combinación de hiperparámetros puede realizarse manualmente o mediante un algoritmo de búsqueda (Bengio, 2012). La primera opción es simple. Consiste en probar muchas combinaciones de hiperparámetros y ver con cuál el modelo funciona mejor con algún conjunto de prueba o validación. Si se automatiza este proceso, la técnica se denomina *grid search* en inglés (Aurélien, 2017). No obstante, dado que existen muchos hiperparámetros, es preferible utilizar una búsqueda aleatoria en lugar de una *grid search*.

Usar la búsqueda aleatoria no es difícil y funciona bien para muchos problemas simples. Sin embargo, cuando los problemas tienen grandes conjuntos de datos, este enfoque solo explorará una pequeña porción del espacio de hiperparámetros (Aurélien, 2017). Se puede aliviar parcialmente este problema ayudando al proceso de búsqueda manualmente: se ejecuta una búsqueda aleatoria rápida usando amplios rangos de valores de hiperparámetros, luego se ejecuta otra búsqueda usando rangos más pequeños de valores centrados en los mejores encontrados durante la primera ejecución, y así sucesivamente (Aurélien, 2017). Es de esperar que esto encuentre un buen conjunto de hiperparámetros. Sin embargo, esto lleva mucho tiempo y probablemente no haga un uso eficiente del tiempo

(Aurélien, 2017).

Ahora bien, existen muchas técnicas para explorar un espacio de búsqueda de manera mucho más eficiente que de forma manual y/o aleatoria. La idea central es simple: cuando una región del espacio resulta ser buena, se debe explorar más (Aurélien, 2017). Esto se encarga del proceso de "ampliación" de forma autónoma y conduce a mejores soluciones en menos tiempo (Aurélien, 2017). Por esta razón, el ajuste de hiperparámetros de una red neuronal sigue siendo un área activa de investigación y los algoritmos evolutivos (considerados metaheurísticos) están resurgiendo últimamente para este problema (Aurélien, 2017).

4. Metaheurísticas

Las metaheurísticas constituyen un campo de investigación activo que también interactúa exitosamente con muchos otros campos de investigación (Johnson et al., 2020).

Las heurísticas son procedimientos que muestran empíricamente buenos resultados en algunas familias de problemas o en instancias de ellas. Para tener éxito, una heurística tiene que encontrar un balance entre dos estrategias diferentes: exploración y explotación. Exploración es la capacidad de evaluar soluciones factibles en diversas partes del espacio de búsqueda. En cambio, explotación significa buscar mejores soluciones en las proximidades de una solución ya conocida (Johnson et al., 2020). Estas estrategias son difíciles de equilibrar en general. En algunas heurísticas simples, se puede encontrar un equilibrio adecuado para una instancia del problema, aunque para una instancia diferente, el enfoque puede resultar muy desequilibrado. (Johnson et al., 2020). Estos desequilibrios restringirán la heurística a una pequeña región del espacio de búsqueda, obligándola a converger a óptimos locales pobres, o en los otros casos, obligarán a la heurística a buscar aleatoriamente grandes secciones del espacio de búsqueda sin mejorar significativamente los resultados (Johnson et al., 2020).

Las metaheurísticas son procedimientos de alto nivel que se utilizan para determinar o diseñar una heurística capaz de lograr una solución suficientemente buena a un problema de optimización (Johnson et al., 2020). Las metaheurísticas son eficientes en problemas complejos que requieren grandes cantidades de datos y grandes cantidades de procesamiento (Johnson et al., 2020). Ésto debido a que no garantizan un óptimo global, pero son capaces de encontrar una solución suficientemente buena para resolver el problema consumiendo menos tiempo y recursos de procesamiento (Johnson et al., 2020). En consecuencia, son particularmente útiles cuando no existe un método exacto de resolución o se requieren demasiados recursos para un método exacto, y también cuando no se necesita la solución óptima y una buena es suficiente (Johnson et al., 2020). En particular, las metaheurísticas de inspiración biológica, como los algoritmos genéticos, extraen sus estrategias de fenómenos que ocurren naturalmente (Johnson et al., 2020).

4.1 Algoritmo Genético

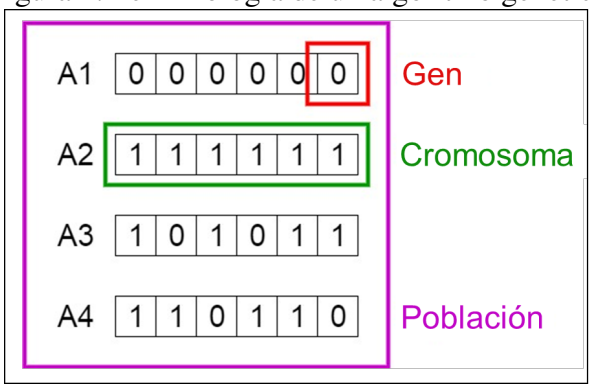
La evolución biológica es una fuente de inspiración atractiva para abordar problemas computacionales difíciles. La evolución es, en efecto, un método de

búsqueda entre un enorme número de posibles “soluciones” (Mitchell, 1998). En biología, ese conjunto de posibilidades es el conglomerado de secuencias genéticas posibles, y las “soluciones” deseadas son organismos altamente aptos: organismos capaces de sobrevivir y reproducirse en sus entornos (Mitchell, 1998). De forma general, las “reglas” de la evolución son simples: las especies evolucionan mediante variaciones aleatorias (ya sean mediante mutación, recombinación y/u otros operadores), seguida de la selección natural en donde los más aptos tienden a sobrevivir y reproducirse, logrando así propagar su material genético a las generaciones futuras (Mitchell, 1998).

Los algoritmos genéticos (*GA*, por sus siglas en inglés) son algoritmos de búsqueda basados en la mecánica de la selección natural y la genética biológica (Goldberg, 1989). En cada generación, se crea un nuevo conjunto de individuos utilizando fragmentos de los individuos anteriores más aptos (Goldberg, 1989). De vez en cuando, se desarrollan nuevos fragmentos “mutados”. Si bien los algoritmos utilizan la aleatorización, los algoritmos genéticos no son una simple búsqueda aleatoria (Goldberg, 1989). Éstos explotan eficientemente la información histórica para especular sobre nuevos puntos de búsqueda con un mejor rendimiento esperado (Goldberg, 1989).

En los algoritmos genéticos, el término cromosoma (o individuo) normalmente se refiere a una solución candidata a un problema, a menudo codificada como una cadena de bits llamados “genes” (Mitchell, 1998). Cada cromosoma puede considerarse como un punto en el espacio de búsqueda de soluciones candidatas (Mitchell, 1998). La Figura 4 muestra gráficamente los términos mencionados.

Figura 4: Terminología de un algoritmo genético



Nota. Tomado de Mallawaarachchi (2017).

Como ocurre con cualquier método de búsqueda y aprendizaje, la forma en la que se codifican las soluciones candidatas es un factor central (o bien el más importante) en el éxito de un algoritmo genético (Mitchell, 1998). La mayoría de las aplicaciones de un GA utilizan cadenas de bits de orden y longitud fija para codificar soluciones candidatas (Mitchell, 1998). Sin embargo, en los últimos años se han realizado numerosos experimentos con otros tipos de codificación (Mitchell, 1998).

Para muchos problemas, se pueden utilizar números reales para formar cromosomas en lugar de bits (Mitchell, 1998). Además, elegir una codificación fija de antemano presenta una paradoja para el usuario potencial de un GA: para cualquier problema que sea lo suficientemente difícil como para usar el algoritmo, no se tiene de antemano el conocimiento suficiente sobre el problema como para determinar la mejor codificación (Mitchell, 1998) De hecho, encontrar la mejor codificación es casi equivalente a resolver el problema en sí (Mitchell, 1998). Una segunda razón para adaptar la codificación es que una representación de longitud fija limita la complejidad de las soluciones candidatas (Mitchell, 1998). Es por eso que usar cromosomas de longitud variable permite una evolución más "abierta" (Mitchell, 1998)

Resulta que no existe una definición rigurosa de "algoritmo genético" aceptada por todos en la comunidad de computación evolutiva (Mitchell, 1998). Sin embargo, se puede decir que la mayoría de los métodos llamados "GA" tienen al menos los siguientes elementos en común: poblaciones de cromosomas, selección según la aptitud, cruce para producir nuevos descendientes y mutación aleatoria de los nuevos descendientes (Mitchell, 1998).

4.1.1 Población de cromosomas

El funcionamiento de los algoritmos genéticos es notablemente sencillo (Goldberg, 1989). Todo comienza con una población inicial aleatoria de n cromosomas o individuos (Mitchell, 1998). Se debe definir n de antemano(Goldberg, 1989).

4.1.2 Función de aptitud

El valor de la función objetivo de cada solución candidata se llama función de aptitud (f). Esta función puede verse como la medida del beneficio, utilidad o bondad de la solución para resolver el problema en cuestión (Goldberg, 1989). Biológicamente, la aptitud de un organismo se define típicamente como la probabilidad de que el organismo viva para reproducirse (viabilidad) o como una función del número de descendientes que tiene el organismo (fertilidad) (Mitchell, 1998). Una vez generada la población, el algoritmo calcula la aptitud de cada individuo. Posteriormente, el algoritmo aplica tres tipos de operadores a la población actual para producir una nueva población: selección, cruce y mutación.

4.1.3 Selección

Este operador selecciona cromosomas de la población para su reproducción. Cuanto más apto sea el cromosoma, más probabilidades habrá de que sea seleccionado para reproducirse. La selección se realiza con reemplazo, lo que significa que el mismo cromosoma puede seleccionarse más de una vez para convertirse en padre (Mitchell, 1998).

El operador es una versión artificial de la selección natural (Goldberg, 1989). Hay muchas formas diferentes de implementar el operador de selección. Quizás, la forma más sencilla de implementarlo es crear una ruleta sesgada, donde cada individuo actual de la población ocupa un segmento en la ruleta con tamaño proporcional a su aptitud (Goldberg, 1989). La ruleta se gira y por lo tanto, a mayor aptitud, mayor probabilidad de salir sorteado.

La selección debe equilibrarse con la variación proveniente de los operadores de cruce y la mutación (encontrar el equilibrio entre explotación y exploración) (Mitchell, 1998). Una selección fuerte conlleva a que individuos altamente aptos pero no-óptimos se apoderen de la población, reduciendo la diversidad necesaria para obtener mayores cambios y progresos(Mitchell, 1998). En cambio, una selección débil dará como resultado una evolución demasiado lenta (Mitchell, 1998)

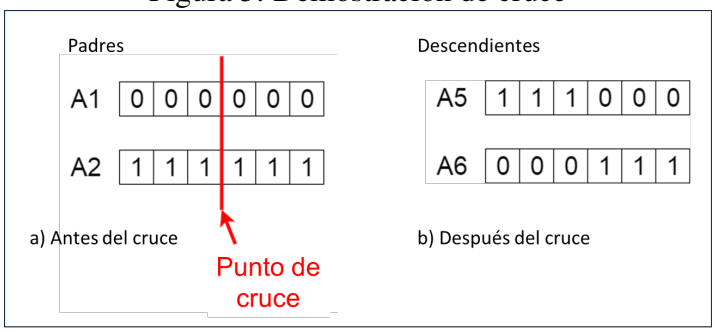
El elitismo es un complemento a muchos métodos de selección que obliga al GA a retener una cierta cantidad de los mejores individuos presentes en la población (Mitchell, 1998). Estos individuos pueden perderse si no se seleccionan para reproducirse o si se modifican por cruce o mutación (Mitchell, 1998). Muchos investigadores han descubierto que el elitismo mejora significativamente el rendimiento del GA (Mitchell, 1998).

4.1.4 Cruce

El operador de cruce imita la reproducción biológica entre dos organismos. Con una probabilidad definida ("probabilidad de cruce" o "tasa de cruce"), el operador cruza un par de cromosomas en un posición elegida al azar (llamada "punto de cruce") (Mitchell, 1998). Las partes de los dos padres posteriores a la posición de cruce se intercambian, formando así dos descendientes (Mitchell, 1998). Si no se produce el cruce, los dos descendientes serán copias exactas de sus respectivos padres (Mitchell, 1998).

El "cruce de punto único" es la forma más simple para reproducir cromosomas (Mitchell, 1998). No obstante, también existen versiones del GA de "cruce multipunto" en las que la tasa de cruce para un par de padres es el número de puntos en los que se produce un cruce (Mitchell, 1998). La Figura 5 ejemplifica el proceso de cruce.

Figura 5: Demostración de cruce



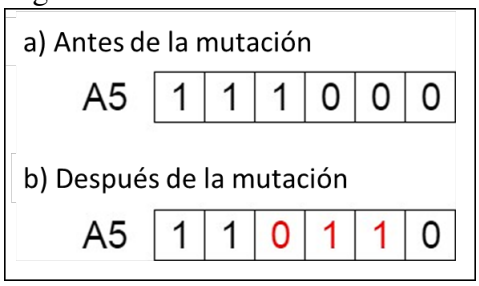
Nota. Adaptado de Mallawaarachchi (2017).

4.1.5 Mutación

El operador de mutación altera aleatoriamente algunos de los genes de un cromosoma descendiente. La mutación puede ocurrir en cada gen del cromosoma con cierta probabilidad ("probabilidad de mutación" o "tasa de mutación"), generalmente muy pequeña (Mitchell, 1998). Por sí misma, la mutación es un paseo aleatorio a través del espacio de cromosomas (Goldberg,

1989). La Figura 6 muestra un ejemplo de mutación.

Figura 6: Demostración de mutación



Nota. Adaptado de Mallawaarachchi (2017).

Cada iteración de este proceso que genera una nueva población se llama generación, y el conjunto completo de generaciones se llama corrida (Mitchell, 1998). Al final de una corrida, suele haber uno o más cromosomas altamente aptos en la población. Dado que la aleatoriedad juega un papel importante en cada ejecución, dos ejecuciones con diferentes semillas de números aleatorios generalmente producirán diferentes comportamientos (Mitchell, 1998). Los investigadores de GA suelen reportar estadísticas (como la mejor aptitud encontrada en una corrida y la generación en la que se descubrió al individuo con esa aptitud) promediadas sobre muchas corridas diferentes de GA sobre el mismo problema (Mitchell, 1998).

5. Evolucionando arquitecturas de redes neuronales

En las últimas décadas, se han propuesto varios enfoques metaheurísticos para optimizar la arquitectura de las ANN. En particular, varios trabajos han implementado algoritmos genéticos en cooperación con redes neuronales. Inclusive, un campo llamado redes neuronales evolutivas se formó en el campo de investigación (Ding, Su, y Yu, 2011). Se presenta un resumen de algunos trabajos en orden cronológico:

- Miller, Todd, y Hegde (1989) presentan un método de algoritmo genético que desarrolla arquitecturas de ANNs para tareas específicas. Sin embargo, el enfoque sólo mejora las conexiones de una arquitectura de entrada, requiriendo predefinir el número de unidades ocultas.
- Mezard y Nadal (1989) implementan un nuevo algoritmo que construye una ANN, donde el número de capas y de unidades ocultas en cada capa fueron resultados del algoritmo. Es un algoritmo de crecimiento de la red, que agrega capas y unidades dentro de una capa a voluntad hasta la convergencia. Sin embargo, el enfoque es aditivo, sin posibilidad de disminuir el número de capas y/o unidades ocultas.
- Kitano (1990) proponen un método mejorado para diseñar redes neuronales utilizando el algoritmo genético para evitar la degradación de la convergencia a medida que aumenta el tamaño de la red. Se propuso una codificación gramatical gráfica que genera patrones de conectividad más regulares con cromosomas de longitud más corta.

- Oliker (1992) ofrecieron un nuevo enfoque para diseñar y entrenar redes neuronales utilizando un algoritmo genético distribuido. Éste modifica libremente el número de neuronas ocultas con operadores genéticos. Aun así, el número de capas ocultas no se modifica, utilizando solo una.
- Angeline, Saunders, y Pollack (1994) desarrollan un programa evolutivo, llamado GNARL, que adquiere simultáneamente la estructura y los pesos de las RNN. Las técnicas evolutivas se comienzan a a aplicar a RNN, para el diseño y entrenamiento de sus estructuras.
- Muchos grupos, incluidos OpenAI, Uber Labs, Sentient Labs, DeepMind y Google Brain, trabajan en el desarrollo de un método automatizado para generar la arquitectura adecuada de las ANN, para cualquier tarea determinada, mediante el uso de algoritmos evolutivos. Estos experimentos requieren muchos cálculos, usando cientos de GPU/TPU durante días (Real, 2018).
- Aliefa y Suyanto (2020) proponen un algoritmo genético de cromosomas de longitud variable (VLC-GA) para encontrar la estructura óptima para las RNN, capaz de hacer evolucionar la estructura de manera constructiva y destructiva.
- Johnson et al. (2020) proponen un nuevo algoritmo genético para la optimización de la arquitectura de una red neuronal convolucional para clasificación de imágenes. El algoritmo permite una exploración en profundidad, introduciendo un nuevo operador de cruce secuencial, con un programa de presión selectiva incremental sobre la evolución (que favorece una mayor diversidad en las primeras generaciones) y evaluando el desempeño individual sobre el conjunto de validación con parada temprana. Aunque el artículo se centra en otro tipo de ANN para una tarea no relacionada con el pronóstico de series de tiempo, propone la implementación de una parada temprana y también calcular la aptitud con un conjunto de validación.
- Cicek y Ozturk (2021) desarollan un nuevo algoritmo híbrido basado en algoritmos genéticos de clave aleatoria sesgados para la optimización de ANN. El denominado BRKGA–NN determina el número de neuronas ocultas, los valores de sesgo de las neuronas ocultas y los pesos de conexión entre nodos. El algoritmo se comparó con otros modelos de pronóstico en algunos conjuntos de datos de series de tiempo. BRKGA-NN generó mejores pronósticos que los demás métodos. A pesar de ser una estrategia innovadora para predecir series de tiempo, sólo se utiliza una capa oculta.
- Nikbakht, Anitescu, y Rabczuk (2021) aplican un algoritmo genético a una ANN para optimizar el número de capas ocultas, las conexiones y las neuronas en cada capa para alcanzar la mayor precisión de predicción. Las capas ocultas comienzan a ser modificadas.

6. Evaluación de un modelo de pronóstico

6.1 Medición del error de pronóstico

Para medir la calidad de un modelo predictivo, es necesario comparar los pronósticos y los valores reales (Szablowski, 2023). Un residual es la diferencia entre el valor real y su valor de pronóstico (Hanke y Wichern, 2005).

También se le llama error de pronóstico. El error de pronóstico de cada periodo pronosticado se calcula con la ecuación 1.

$$e_t = Y_t - \hat{Y}_t \tag{1}$$

donde

$$\begin{aligned} e_t &= \text{error de pronóstico en el periodo } t \\ Y_t &= \text{valor real en el periodo } t \\ \hat{Y}_t &= \text{valor de pronóstico para el periodo } t \end{aligned}$$

El error cuadrático medio (*MSE*, siglas del inglés *mean squared error*) es un método para evaluar un método de pronóstico (Hanke y Wichern, 2005). El MSE se determina con la Ecuación 2.

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(Y_t - \hat{Y}_t)^2 \tag{2}$$

Se pueden utilizar muchas medidas de error en el pronóstico de series de tiempo. El error porcentual absoluto medio (*MAPE* en inglés) se utiliza ampliamente para monitorear el rendimineto de un modelo y es de interpretación más sencilla por su naturaleza porcentual. Se calcula con la Ecuación 3.

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|Y_t - \hat{Y}_t|}{Y_t} \tag{3}$$

El MAPE es especialmene útil para comparar la precisión de las mismas o diferentes técnicas de pronóstico en series totalmente distintas (Hanke y Wichern, 2005). Por lo tanto, ayuda a encontrar el método óptimo entre las opciones. El problema con el MAPE es que tiene que tratar con valores infinitos si la serie tiene valores iguales a cero o indefinidos. No obstante, si la serie no contiene estos valores, el MAPE es útil.

6.2 Método de evaluación

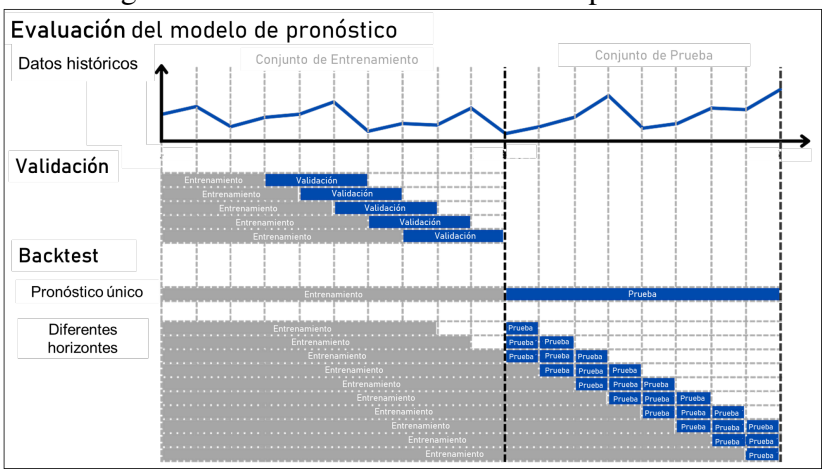
La evaluación de un modelo predictivo de series de tiempo se enfoca en el rendimiento del modelo con datos que no se utilizaron para entrenarlo, llamados datos no vistos o fuera de la muestra (Brownlee, 2016). Primero, se deben dividir los datos disponibles. A menudo se dividen dos conjuntos: el conjunto de entrenamiento utilizado para preparar el modelo, y el conjunto de prueba utilizado para evaluarlo (Brownlee, 2016). La evaluación de las predicciones del conjunto de prueba proporciona un buen indicador de cómo funcionaría el modelo operativemente (Brownlee, 2016).

En las series temporales, los datos están ordenados en relación con el tiempo y las observaciones se relacionan entre sí. Por ende, no es posible usar aleatorización al dividir los conjuntos de entrenamiento y prueba y usar una validación cruzada, porque esta situación generaría fuga de información entre conjuntos (Szablowski, 2023). En vez de esto, los datos se deben dividir respetando el orden temporal en el que se observaron los valores (Brownlee,

2016). En el pronóstico de series de tiempo, esta evaluación de modelos sobre datos históricos se llama *validación cruzada de series de tiempo* o *backtesting* (en inglés) (Brownlee, 2016).

Del conjunto de entrenamiento, se pueden separar subconjuntos de validación, que se utilizan para probar y seleccionar los hiperparámetros del modelo de pronóstico (Szablowski, 2023). Existen dos enfoques para evaluar el modelo con el conjunto de pruebas mediante *backtesting*. La primera es ajustar el modelo al conjunto de entrenamiento y posteriormente generar un pronóstico que cubra todo el conjunto de prueba (Szablowski, 2023). Este método es rápido y no requiere alto poder de cómputo. El segundo enfoque consta de probar diferentes horizontes de pronóstico, donde el modelo puede actualizarse cada vez que se reciben nuevos datos (Brownlee, 2016). En la práctica, es muy probable que se vuelva a entrenar un modelo a medida que haya nuevos datos disponibles (Brownlee, 2016), por lo que este enfoque imita el rendimiento operativo de un modelo una vez implementado. La Figura 7 compara visualmente ambos enfoques de evaluación.

Figura 7: Evaluación de un modelo de pronóstico



Nota. Adaptado de Szablowski (2023).

Debido a que esta metodología implica moverse a lo largo de la serie temporal paso a paso, a menudo se la denomina *walk-forward validation* o *walk-forward testing* (en inglés). Además, debido a que se utiliza una ventana deslizante o expandible para entrenar un modelo, este método también se conoce como *rolling window analysis* o *rolling forecast* (en inglés) (Brownlee, 2016). La metodología tiene la ventaja de proporcionar una estimación mucho más robusta del rendimiento del modelo y los parámetros elegidos en la práctica. Esta estimación mejorada conlleva un mayor costo computacional al crear o entrenar tantos modelos, pero se ha convertido en el estándar de evaluación (Brownlee, 2016).

5. Metodología

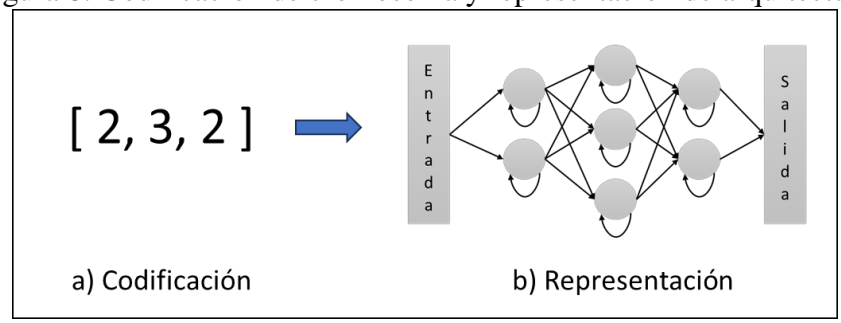
La determinación de la estructura de una RNN para pronosticar series de tiempo no es trivial. Para resolver el problema, se llevaron a cabo dos etapas. En la primera, se diseñó un algoritmo genético para establecer los siguientes hiperparámetros de la red: número de capas ocultas y número de neuronas LSTM por capa oculta. En la segunda etapa, el modelo se puso a prueba con diferentes series de tiempo y su desempeño se comparó con el de otros modelos de pronóstico. Se utilizó una libreta de Google Colab con Python 3 para la programación.

- **Etapa 1: Diseño del algoritmo genético**

  El algoritmo genético propuesto tuvo que ser capaz de generar codificaciones de múltiples arquitecturas posibles de una RNN (para construir la población inicial de cromosomas), calcular su función de aptitud, y aplicar los operadores genéticos en cada generación.

  Los cromosomas se codificaron con números reales en lugar de bits, debido a que su interpretación es más comprensible y su decodificación para construir el modelo RNN fue más rápida. Cada cromosoma se definió como un arreglo dinámico de números reales. Fueron dinámicos porque se admitieron cromosomas de longitud variable. Cada número real representa el número de neuronas de tipo LSTM en la capa oculta respectiva a la posición del elemento en el arreglo. Un ejemplo de cromosoma se muestra en la Figura 8

Figura 8: Codificación de cromosoma y representación de arquitectura



El cromosoma en la Figura 8 se interpreta de la siguiente forma: la estructura propuesta de la RNN es de 3 capas ocultas, con 2 neuronas en la primera capa, 3 en la segunda, y 2 en la tercera. Por lo tanto, el algoritmo tuvo libertad en proponer arquitecturas con número de capas diferentes y que cada capa no necesariamente tuviera el mismo número de neuronas.

El Algoritmo 1 muestra el pseudocódigo del algoritmo genético propuesto. En resumen, el algoritmo genera una población inicial de tamaño N aleatoria de cromosomas codificados. Para construir la nueva población, calcula la aptitud de cada individuo y realiza un reporte de estadísticas de la población. Un porcentaje e de los individuos más aptos (la élite) se transfieren a la nueva población. Hasta que la nueva población no tenga el tamaño suficiente, selecciona de la población dos padres con el método de ruleta sesgada según la aptitud. Con probabilidad

r, se aplicará el cruce a los padres para generar dos descendientes. Cada gen de los descendientes sufrirá una mutación con probabilidad *m*. El algoritmo repite el proceso hasta completar todas las generaciones o hasta que el criterio de parada temprana sea activado.

Algoritmo 1 Pseudocódigo algoritmo genético para estructura de una RNN

function ALGORITMOGENÉTICO($N, maxcapas, maxneuronas, generaciones, maxiter, limite, e, r, m$)
  $iter \leftarrow 0$
  $poblacion \leftarrow \emptyset$
  $poblacion \leftarrow$ GENERARPOBLACION($N, maxcapas, maxneuronas$)
  for ($generacion \in generaciones$) **do**
    for ($individuo \in poblacion$)) **do**
      $aptitud(individuo) \leftarrow$ CALCULARAPTITUD($individuo$)
    end for
    REPORTEGENERACIONAL($aptitud$)
    PARADAANTICIPADA($maxiter, limite$)
    $aptitud \leftarrow 1/aptitud$
    $nueva\_poblacion \leftarrow \emptyset$
    $elite \leftarrow$ ELITISMO($poblacion, aptitud, e$)
    $nueva\_poblacion \leftarrow nueva\_poblacion \cup \{elite\}$
    while $|nueva\_poblacion| < N$ **do**
      $padre1 \leftarrow$ SELECCIONRULETA($N, poblacion, aptitud$)
      $padre2 \leftarrow$ SELECCIONRULETA($N, poblacion, aptitud$)
      $hijo1, hijo2 \leftarrow$ CRUCE($padre1, padre2, r$)
      $hijo1 \leftarrow$ MUTACION($hijo1, m$)
      $hijo2 \leftarrow$ MUTACION($hijo2, m$)
      $nueva\_poblacion \leftarrow nueva\_poblacion \cup \{hijo1\} \cup \{hijo2\}$
    end while
    $poblacion \leftarrow nueva\_poblacion$
    $aptitud(individuo) \leftarrow \emptyset$, $\forall$ $inidividuo \in poblacion$
  end for
end function

La función de población *GenerarPoblacion(N, maxcapas, maxneuronas)* se define en el Algoritmo 2. La función genera *N* individuos. Aleatoriamente, escoge el número de capas ocultas entre 1 y *maxcapas*. Por cada capa oculta, escoge aleatoriamente el número de neuronas entre 1 y *maxneuronas*.

La función *CalcularAptitud(individuo)* toma la codificación del cromosoma y la decodifica para construir un modelo de RNN con la libreria *Keras* en Python. *Keras* construye modelos de manera secuencial, en donde cada capa y sus respectivas neuronas se agregan en orden, una a la vez. La primera capa oculta toma *n_historicos* pasos de tiempo hacia atrás como entrada. Después de añadir las capas ocultas restante, se agrega la capa de salida que transforma la salida en un pronóstico de *n_pronostico* pasos hacia adelante. Una vez construido, el modelo se compila con la función de pérdida MSE y con el optimizador *Adam*. La aptitud de la solución es el promedio del MAPE obtenido realizando el proceso de evaluación *backtesting*. Por lo tanto, al ser un problema de minimización, un cromosoma con menor aptitud es preferible. Se usó una ventana expandible, que se expande

Algoritmo 2 Pseudocódigo algoritmo generar población

function GENERARPOBLACION($N, maxcapas, maxneuronas$)
  for $i \in 1...N$ **do**
    $capas \leftarrow aleatorio[1, maxcapas]$
    for $j \in 1...capas$ **do**
      $neuronas \leftarrow aleatorio[1, maxneuronas]$
      $individuo \leftarrow individuo \cup \{neuronas\}$
    end for
  end for
end function

hacia adelante de acuerdo a la longitud de validación, posterior a que el modelo se entrenó y realizó una predicción. Cada modelo se entrena con 100 épocas y lotes de 32 datos. El proceso se repite hasta que la ventana contiene a todos los datos de entrenamiento.

Algoritmo 3 Pseudocódigo algoritmo función de aptitud

function CALCULARAPTITUD($individuo$)
  $errores \leftarrow \emptyset$
  $entrenamiento \leftarrow conjunto\ de\ entrenamiento$
  $n\_historico \leftarrow longitud\ de\ validacion$
  $n\_pronostico \leftarrow longitud\ de\ validacion$
  $paso \leftarrow longitud\ de\ expansion\ de\ la\ ventana$
  Construir el modelo RNN(LSTM) conforme estructura de $individuo$
  $ventana \leftarrow entrenamiento(1, n\_historico + n\_pronostico)$
  while $entrenamiento$ no recorrido por completo **do**
    Entrenar el modelo con la ventana de datos disponible
    Predecir los siguientes $n\_pronostico$ datos
    $error \leftarrow error\ de\ pronostico\ MAPE$
    $errores \leftarrow errores \cup error$
    Expandir $ventana$ con $paso$ datos hacia adelante
  end while
  $aptitud(individuo) \leftarrow \frac{errores}{|errores|}$
end function

Ya que el algoritmo calculó la aptitud de cada individuo en la población, la función *ReporteGeneracional(aptitud)* toma el arreglo donde se guardó la aptitud de todos los individuos y calcula las estadísticas de la generación. Se obtiene la mejor, la peor y la aptitud promedio, al igual que la desviación estándar y el tiempo de ejecución.

Antes de proceder a la siguiente generación, la función *ParadaAnticipada(maxiter, limite)* verifica si han pasado *maxiter* generaciones sin encontrar una mejor solución o si ésta ya está dentro de un nivel deseado *limite*. Si alguna de éstas condi-

Algoritmo 4 Pseudocódigo algoritmo reporte generacional

function REPORTEGENERACIONAL($aptitud$)
$minima(generacion) \leftarrow min\{aptitud(i) : i \in N\}$
$maxima(generacion) \leftarrow max\{aptitud(i) : i \in N\}$
$promedio(generacion) \leftarrow \frac{1}{N}\sum_{i=1}^{N} aptitud(i)$
$desvest(generacion) \leftarrow \sqrt{\frac{\sum_{i=1}^{N}(aptitud(i)-promedio)^2}{N}}$
Recabar tiempo de ejecución hasta el momento
end function

ciones se cumple, el algoritmo genético se detendrá por completo. Es importante, ya que puede ahorrar tiempo de ejecución innecesario.

Algoritmo 5 Pseudocódigo algoritmo parada anticipada

function PARADAANTICIPADA($maxiter, limite$)
if $generacion > 1$ **then**
if $minima(generacion) > minima(generacion - 1)$ **then**
$iter \leftarrow iter + 1$
end if
if $iter > maxiter$ **then**
break
end if
if $minima(generacion) < limite$ **then**
break
end if
end if
end function

En cambio, mientras la búsqueda siga siendo atractiva, el algoritmo irá produciendo generaciones con los operadores genéticos. El elitismo, que es un componente de la selección, se lleva a cabo con la función *Elitismo(poblacion, aptitud, e)*. Ésta escoge al porcentaje *e* de los individuos más aptos en la población actual *poblacion* de acuerdo a *aptitud*.

Algoritmo 6 Pseudocódigo algoritmo elitismo

function ELITISMO($poblacion, aptitud, e$)
$num\_elite \leftarrow redondear(|poblacion| \cdot e)$
Ordenar los individuos en la población por $aptitud$ de forma descendiente
Seleccionar los primeros num_elite del listado ordenado
end function

La función *SeleccionRuleta(N,poblacion, aptitud)* escoge a un individuo de los *N* existentes en la población por ruleta sesgada. Primero, calcula la aptitud

acumulada de todos los individuos. Luego, genera un número aleatorio entre 0 y el total de aptitud de la población. Para cada individuo, si su aptitud acumulada es mayor o igual al número aleatorio, se escoge a ese individuo. A mayor aptitud, mayor probabilidad de ser seleccionado.

Algoritmo 7 Pseudocódigo algoritmo de selección por ruleta sesgada

```
function SELECCIONRULETA(N, poblacion, aptitud)
    aptitud_acumulada(0) ← 0
    for i ∈ 1...N do
        aptitud_acumulada(i) ← aptitud_acumulada(i − 1) + aptitud(i)
    end for
    aptitud_total ← Σ_{i=1}^{N} aptitud(i)
    valor ← aleatorio[0, aptitud_total]
    indice ← 0
    for i ∈ 1...N do
        if aptitud_acumulada(i) >= valor then
            indice ← i
            break
        end if
    end for
end function
```

La función de cruce *Cruce(padre1, padre2, r)* cruza a dos padres con probabilidad *r*. Si el cruce se da, primero determina la longitud mínima entre ambos padres. Aleatoriamente, se escoge el punto de cruce entre 1 y la longitud mínima. Los descendientes se forman con el primer segmento de cada padre antes del punto de cruce y se intercambia el siguiente segmento de los padres entre los descendientes.

Por último, la función *Mutacion(hijo, m)* produce mutaciones a cada gen del descendiente con probabilidad *m*. Si se da la mutación, el gen es sustituido por otro generado aleatoriamente entre 1 y *maxneuronas*. No obstante, también se puede mutar la longitud del descendiente. Con la misma probabilidad *m*, se escoge una gen del descendiente. Con un 0.5 de probabilidad, se removerá ese gen. Con el resto de la probabilidad, antes del gen en cuestión se añadirá un gen con valor entre 1 y *maxneuronas*.

Los algoritmos genéticos son capaces realizar búsquedas rápidamente en espacios grandes e irregulares de soluciones, y solo requiere información del valor de la función objetiva para guiarse (Han y May, 1996). Además, adoptan una visión más global del espacio de soluciones en comparación con otros métodos de búsqueda.

Sin embargo, el algoritmo genético debe compararse con un marco de referencia para comprobar que es más eficiente que una búsqueda puramente aleatoria. El algoritmo 10 muestra el pseudocódigo de dicha búsqueda. La búsqueda comienza igual que el genético, generando un conjunto de *N* soluciones aleatorias. Mientras el tiempo de ejecución establecido no termine, el algoritmo calcula la aptitud de cada solución y genera un reporte de estadísticas del conjunto.

Algoritmo 8 Pseudocódigo algoritmo cruce

```
function CRUCE(padre1, padre2, r)
    valor ← aleatorio[0, 1]
    if valor < r then
        longitud_minima ← min{|padre1|, |padre2|}
        if longitud_minima = 1 then
            punto_de_cruce ← 1
        else
            posicion ← aleatorio[1, longitud_minima − 1]
            punto_de_cruce ← posicion
        end if
        hijo1 ← {padre1(i) : i ≤ posicion} ∪ {padre2(i) : i > posicion}
        hijo2 ← {padre2(i) : i ≤ posicion} ∪ {padre1(i) : i > posicion}

    else
        hijo1 ← padre1
        hijo2 ← padre2
    end if
end function
```

Algoritmo 9 Pseudocódigo algoritmo mutación

```
function MUTACION(hijo, m)
    for i ∈ 1...|hijo| do
        valor ← aleatorio[0, 1]
        if valor < m then
            nuevo_gen ← aleatorio[1, maxneuronas]
            hijo(i) ← nuevo_gen
        end if
    end for
    valor ← aleatorio[0, 1]
    posicion ← aleatorio[1, |hijo|]
    if valor < m then
        valor2 ← aleatorio[0, 1]
        if valor2 <= 0.5 then
            if |hijo| > 1 then
                Remover del hijo el gen ubicado en posicion
            end if
        else
            gen_adicional ← aleatorio[1, maxneuronas]
            Insertar gen_adicional en ubicación posicion en el hijo
        end if
    end if
end function
```

Algoritmo 10 Pseudocódigo algoritmo de búsqueda aleatoria para estructura de una RNN

function BUSQUEDAALEATORIA($N, maxcapas, maxneuronas, maxtiempo, maxiter, limite$)
  $iter \leftarrow 0$
  $tiempo \leftarrow$ Tomar tiempo transcurrido desde el inicio
  while $tiempo < maxtiempo$ **do**
    $poblacion \leftarrow \emptyset$
    $aptitud(individuo) \leftarrow \emptyset$, $\forall$ $inidividuo \in poblacion$
    $poblacion \leftarrow GenerarPoblacion(N)$
    for $(individuo \in poblacion)$) **do**
      $aptitud(individuo) \leftarrow CalcularAptitud(individuo)$
    end for
    $ReporteGeneracional(aptitud)$
    $ParadaAnticipada(maxiter, limite)$
    $tiempo \leftarrow$ Tomar tiempo transcurrido desde el inicio
  end while
end function

- **Etapa 2: Evaluación del modelo propuesto**

  Al terminar, el algoritmo genético devolvió la mejor estructura encontrada del modelo RNN. Para determinar si el modelo es robusto para construir modelos de pronóstico, se implementó con diferentes series de tiempo y se comparó cada modelo propuesto con otros modelos de pronóstico.

  Se utilizaron seis series de tiempo. La Tabla 1 describe cada una, sus características y el tamaño de cada instancia. Es notable que las series provienen de diferentes ámbitos, como el financiero, retail, turismo, económico y fenómenos naturales. La diversidad en tamaños, características y ámbito de procedencia propició una evaluación integral. En la Tabla 1 también se establece cómo se dividieron los datos en los conjuntos de entrenamiento, validación y prueba. Se asignó alrededor del 80 por ciento de los datos a entrenamiento y el resto a prueba.

Tabla 1: Detalles de las series de tiempo

Serie de tiempo	Descripción	Características	Tamaño	Entrenamiento	Validación	Prueba	Fuente
Air Passengers	Pasajeros mensuales de aerolínea internacional en miles entre 1949 y 1960	Estacionalidad mensual, con tendencia	144	120	12	24	Darts (2023)
Real GNP	Producto Nacional Bruto real EE.UU. en miles de millones de dólares entre 1890 y 1974.	Sin estacionalidad, con tendencia	85	70	5	15	Federal Reserve Bank St. Louis (2023)
Lynx	Linces anuales atrapados en el Río MacKenzie entre 1821 y 1934	Estacionaria, estacionalidad irregular	114	94	20	10	Encyclopedia of mathematics (2020)
Exchange Rate	Tipo de cambio USD/GBP diario entre octubre 2021 y septiembre 2023	No estacionaria, con tendencia irregular	730	608	122	61	Google Finance (2023)
Walmart	Ventas del producto 2 en la tienda 28 diarias entre 2015 y 2017	Estacionalidad anual, con tendencia	1,096	912	184	92	Kaggle (2023)
Sunspots	Número de manchas solares mensuales de Wolf entre 1700 y 1988	Estacionalidad de 11 años, estacionaria	2,820	2,220	600	300	Darts (2023)

Para construir los demás modelos de pronóstico, se utilizó la librería *Darts*. Esta

librería permite desplegar los modelos fácilmente y realizar el *backtesting* mediante un método integrado en cada modelo. Se utilizaron modelos de las diferentes categorías mencionadas en el marco teórico. Los modelos estadísticos fueron: Ingenuo Estacional, AutoETS, AutoARIMA, y Prophet. Los modelos de aprendizaje automático: N-BEATS, N-HiTS, LSTM RNN con estructura propuesta del algoritmo genético. Y por último, modelos de aprendizaje de conjuntos (tanto estadísticos como de aprendizaje automático): Ensamble Estadístico, LightGBM, XGBoost. Los parámetros (para modelos estadísticos) y los hiperparámetros (para modelos de aprendizaje automático) fueron optimizados mediante un análisis exploratorio de datos junto con *grid search* de forma arbitraria.

Primero, con cada modelo se pronosticaron los datos de prueba de cada serie de tiempo y se compararon los errores de pronóstico entre ellos. Se les asignó un puntaje conforme a su desempeño con cada serie. El modelo con el menor error obtuvo 11 puntos, y el modelo con mayor error obtuvo 1 punto. Así, al juntar la puntuación de cada modelo en todas las series, se clasifica al mejor aquel con más puntos. Posteriormente, todos los modelos realizaron pronósticos a distintos horizontes de tiempo con cada serie. Algunos modelos, especialmente los de aprendizaje automático, pueden tomar covariables estáticas o futuras para mejorar sus predicciones. Para que las comparaciones fueran justas, ningún modelo fue alimentado con covariables en el entrenamiento, únicamente con la serie de tiempo. La serie se escaló entre 0 y 1 para los modelos de aprendizaje automático. Los errores de pronóstico se recabaron mediante *backtesting* en cada horizonte, y se verificó que fueran independientes y normalmente distribuidos. Finalmente, se realizó una ANOVA de Welch para determinar si los modelos generaron medias de errores iguales, tomando en cuenta que las varianzas no eran iguales. Las muestras por horizonte y serie de tiempo fueron del mismo tamaño. Además, se utilizó el método de Games-Howell para determinar cuáles medias fueron significativamente diferentes haciendo comparación por pares. Las pruebas se realizaron con el software estadístico Minitab.

6. Resultados y Discusión

- **Etapa 1: Desempeño del algoritmo genético**

  El algoritmo genético se utilizó con cada serie de tiempo para proponer una estructura del modelo RNN de pronóstico. Se estableció en general una población inicial de tamaño $N = 50$, con $maxcapas = 5$ y $maxneuronas = 256$. En cuanto a los operadores genéticos, se utilizó un elitismo $e = 0.12$, tasa de cruce $r = 0.7$, y tasa de mutación $m = 0.1$. Aunque es recomendable utilizar una tasa de mutación muy pequeña (cercana a 0.001), dado que el espacio de soluciones es extremadamente amplio, el valor de *m* tenía que ser más grande. En lo particular, el número de *generaciones* y los criterios de parada *maxiter* y *límite* dependieron del tamaño de la serie. A mayor tamaño, menor número de generaciones debido a que se contaba con recursos computacionales limitados. La búsqueda aleatoria comenzó con el mismo tamaño de población inicial y parámetros de estructura de las soluciones. También, se implementó tomando el mismo tiempo de ejecución del algoritmo genético.

  La Figura 9 muestra el desempeño del algoritmo genético para encontrar arquitecturas con menor error contra la búsqueda aleatoria. Es notable que, con el paso de las generaciones, el algoritmo genético encuentra arquitecturas con menor error, culminando en una mejor solución encontrada que alguna de las mínimas en cada iteración de la búsqueda aleatoria. El único caso donde la búsqueda aleatoria encontró una mejor solución, se observa en la Figura 9 e) con la serie *Walmart*. La búsqueda aleatoria por cuestiones de azar puede encontrar una arquitectura formidable, pero no de forma consistente como el algoritmo genético. Por lo tanto, el algoritmo genético demostró tener un mejor desempeño.

  La estructura con aptitud mínima encontrada por el algoritmo genético por serie de tiempo se muestra en la Tabla 2.

Tabla 2: Estructuras de aptitud mínima encontradas

Serie de tiempo:	Air Pasengers	Real GNP	Lynx	Exchange Rate	Walmart	Sunspots
Estructura:	[112, 93]	[60]	[52]	[122]	[156, 153]	[151, 1, 30]

  De igual manera, la Figura 10 muestra el desempeño promedio del algoritmo genético contra la búsqueda aleatoria. Al finalizar la ejecución, el algoritmo genético no solo encuentra una arquitectura con aptitud menor, también ofrece diversas soluciones con una aptitud menor en comparación con la búsqueda aleatoria.

- **Etapa 2: Comparación de modelos de pronóstico**

  La comparación comienza con el rendimiento de cada modelo con los datos de prueba. Este conjunto de datos jamás fue utilizado por los modelos para su entrenamiento. La Tabla 3 muestra el MAPE de cada modelo para todas las series de tiempo. Conforme al puntaje obtenido, la última columna muestra la clasificación de los modelos. Es evidente que los modelos con un menor error en los datos de prueba son los modelos de aprendizaje automático, siendo los primeros los modelos más recientes, basados en redes neuronales: N-BEATS y N-HiTS. No obstante,

Figura 9: Comparación aptitud mínima: Genético vs. Aleatorio

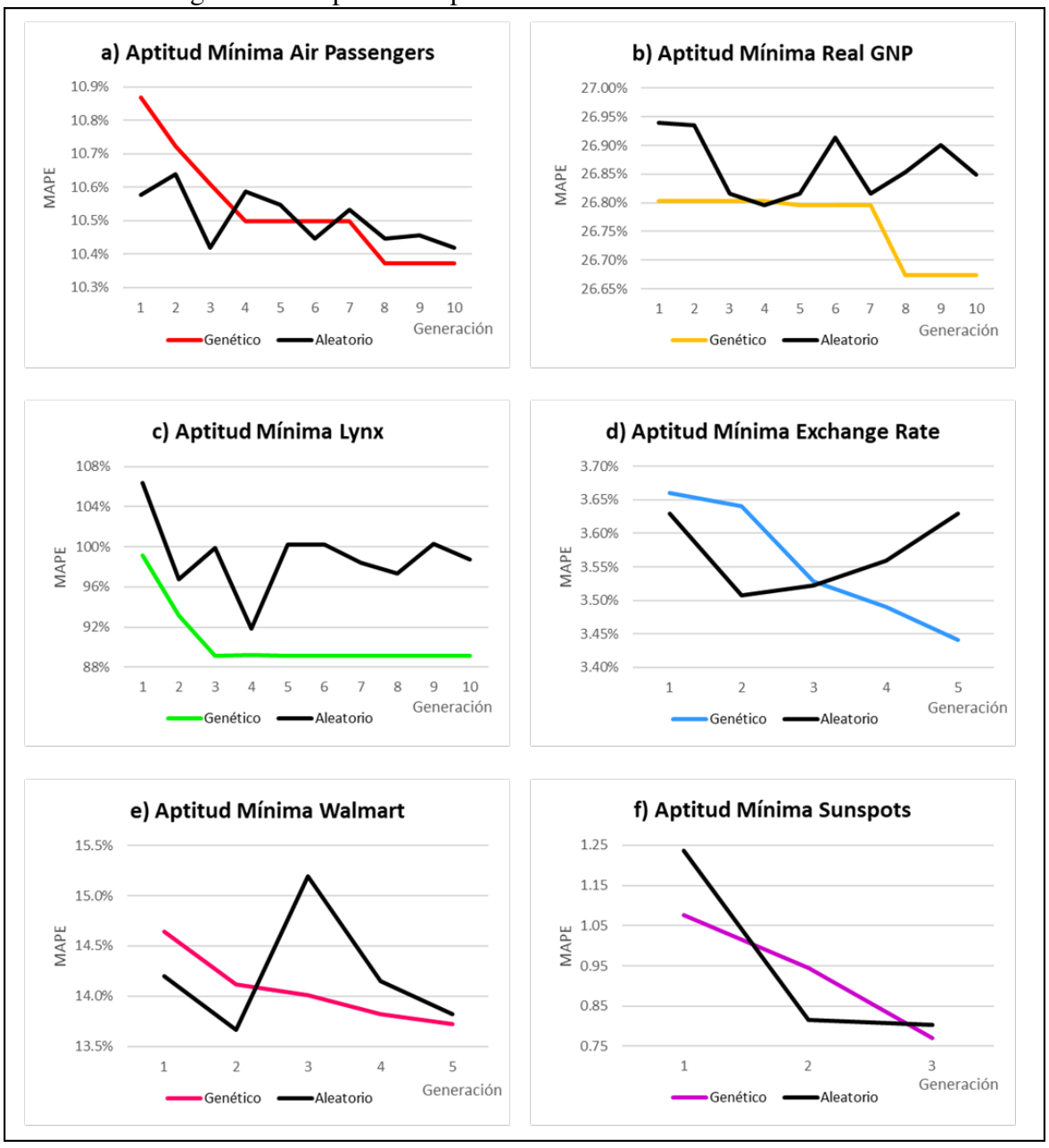
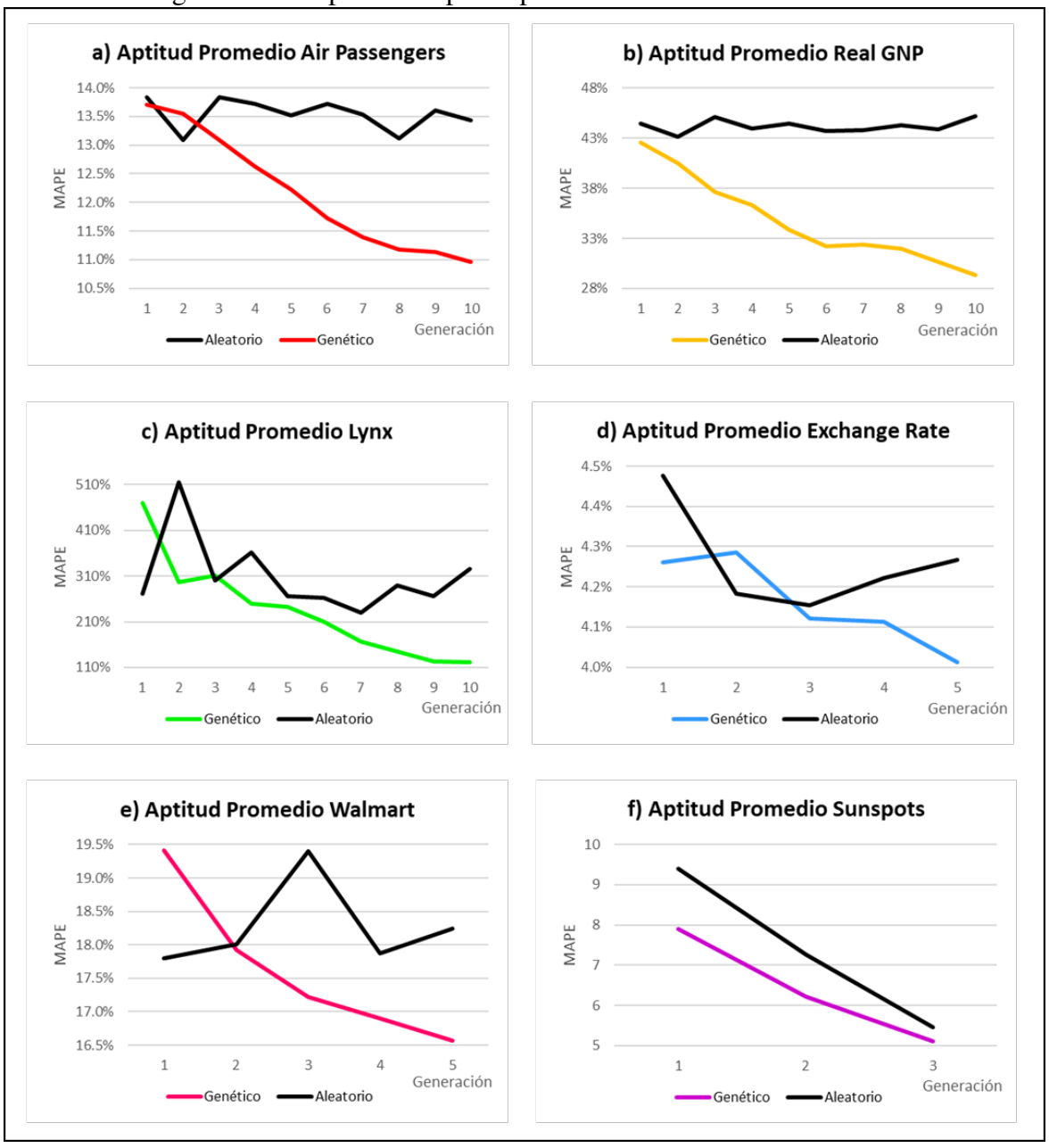


Figura 10: Comparación aptitud promedio: Genético vs. Aleatorio



el Ensamble Estadístico logró destacarse entre los mejores modelos. En cuanto al modelo RNN (con celdas LSTM) con la arquitectura propuesta por el algoritmo genético, logró posicionarse por encima de los modelos estadísticos, justo a la mitad de la clasificación.

Tabla 3: Clasificación de modelos por error de pronóstico de prueba

Modelo	Air Passengers	Real GNP	Lynx	Exchange Rate	Walmart	Sunspots	**Puntaje**	**Clasificación**
Ingenuo Estacional	15.52 %	38.40 %	249.21 %	2.04 %	10.70 %	100.87 %	25	N-BEATS
Ensamble Ingenuo	12.77 %	20.26 %	348.26 %	2.06 %	10.09 %	132.12 %	33	N-HiTS
AutoETS	5.32 %	26.46 %	210.07 %	2.04 %	19.60 %	184.57 %	32	Ensamble Estadístico
AutoARIMA	14.93 %	23.27 %	116.08 %	2.07 %	10.21 %	195.37 %	35	LightGBM
Prophet	4.75 %	31.11 %	512.70 %	1.73 %	10.81 %	92.76 %	35	XGBoost
Ensamble Estadístico	10.65 %	28.55 %	266.81 %	1.78 %	9.69 %	87.04 %	40	LSTM + Genético
LightGBM	15.83 %	60.92 %	152.44 %	1.81 %	8.58 %	74.48 %	38	AutoARIMA
XGBoost	13.17 %	39.09 %	128.55 %	4.37 %	8.85 %	60.37 %	37	Prophet
N-BEATS	11.96 %	24.25 %	101.71 %	2.73 %	8.99 %	74.88 %	44	Ensamble Ingenuo
N-HiTS	11.59 %	26.85 %	162.57 %	2.23 %	9.04 %	62.59 %	41	AutoETS
LSTM + Genético	6.56 %	24.27 %	132.13 %	2.67 %	11.56 %	91.06 %	36	Ingenuo Estacional

La segunda parte de la comparación consiste en el desempeño de los modelos mediante *backtesting*. El proceso se realizó utilizando diferentes horizontes de pronóstico. Para cada serie, se definió un corto plazo (*CP*), mediano plazo (*MP*) y largo plazo (*LP*). El *backtesting* no pudo realizarse para la serie *Sunspots* debido a su tamaño y los recursos computacionales disponibles. Las Figuras 11 y 12 muestran gráficamente el desempeño de cada modelo con las demás series en los diversos horizontes. A pesar de las variaciones y excepciones presentes, se puede visualizar que en general, los modelos estadísticos muestran un MAPE menor que los modelos de aprendizaje automático a través de los horizontes. No obstante, se necesita un análisis contundente.

La aplicación de la ANOVA de Welch determinó si las medias de errores entre los modelos en cada horizonte por serie son significativamente diferentes. La prueba post hoc de Games-Howell se utilizó para realizar comparaciones por parejas entre los modelos. Con un nivel de significancia de 0.05, la Tabla 4 muestra, por horizonte y por serie de tiempo, que los modelos que no comparten alguna letra entre ellos son significativamente diferentes. También, van en orden descendente, por lo que las medias con la letra A son mayores a las de la letra B, sucesivamente. En el caso de la serie *Lynx*, se concluye que no hubo diferencias significativas entre los modelos en el CP y LP. Aunque las diferencias se han atenuado, sigue siendo notable que los modelos estadísticos son más consistentes en tener menores errores mediante el *backtesting*. Este comportamiento se debe a la diferente naturaleza de los modelos. Como los modelos de aprendizaje automático van ajustando sus parámetros a lo largo del tiempo, al comienzo del *backtesting* generan grandes errores de pronóstico. Al entrenarse con más datos durante el aprendizaje, los errores van disminuyendo y el pronóstico incrementa su precisión. Los modelos estadísticos son más estables en los errores debido a que sus parámetros deben ser definidos antes de realizar el *backtesting*.

Figura 11: Backtesting por horizonte de series a, b, y c

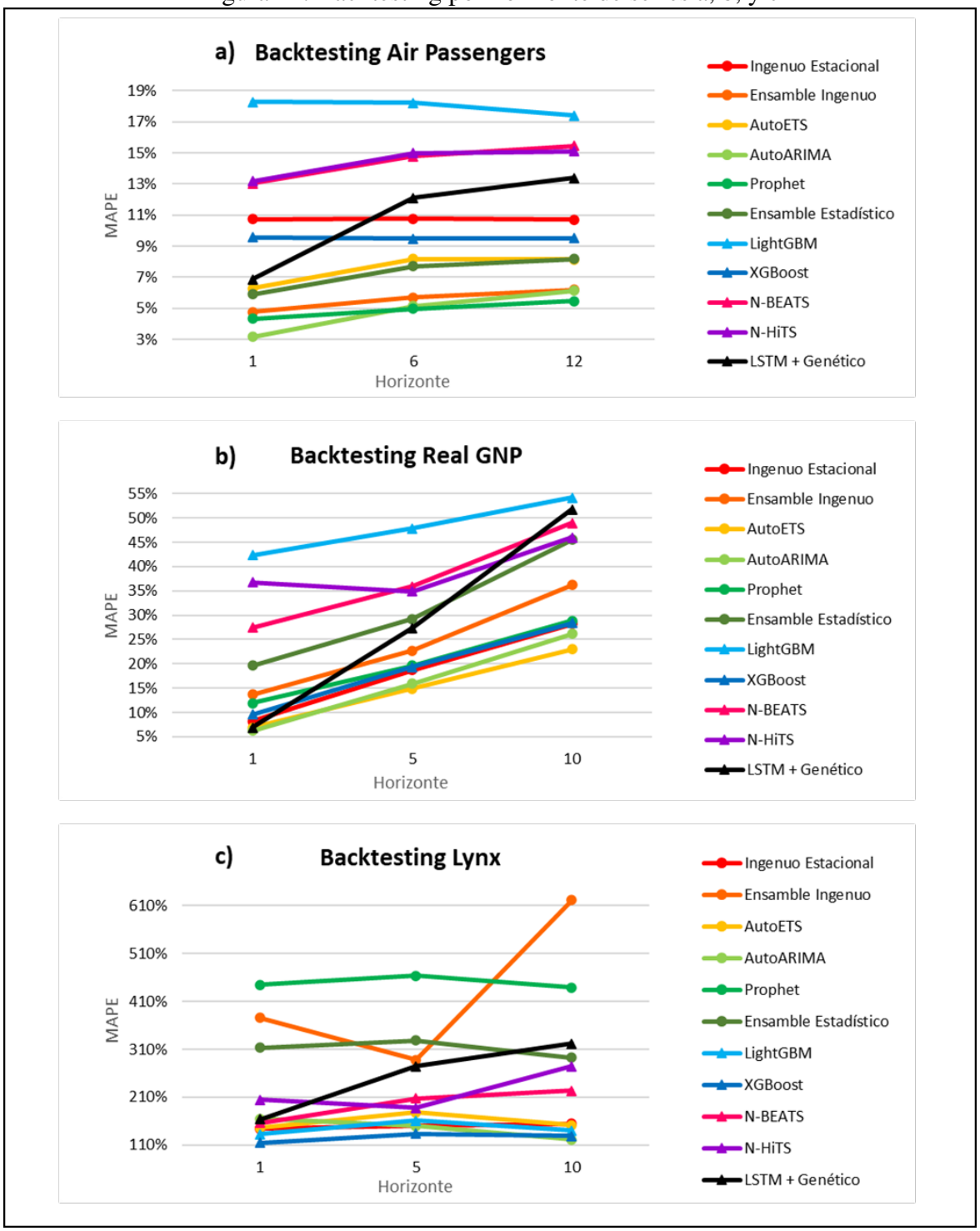


Figura 12: Backtesting por horizonte de series d y c

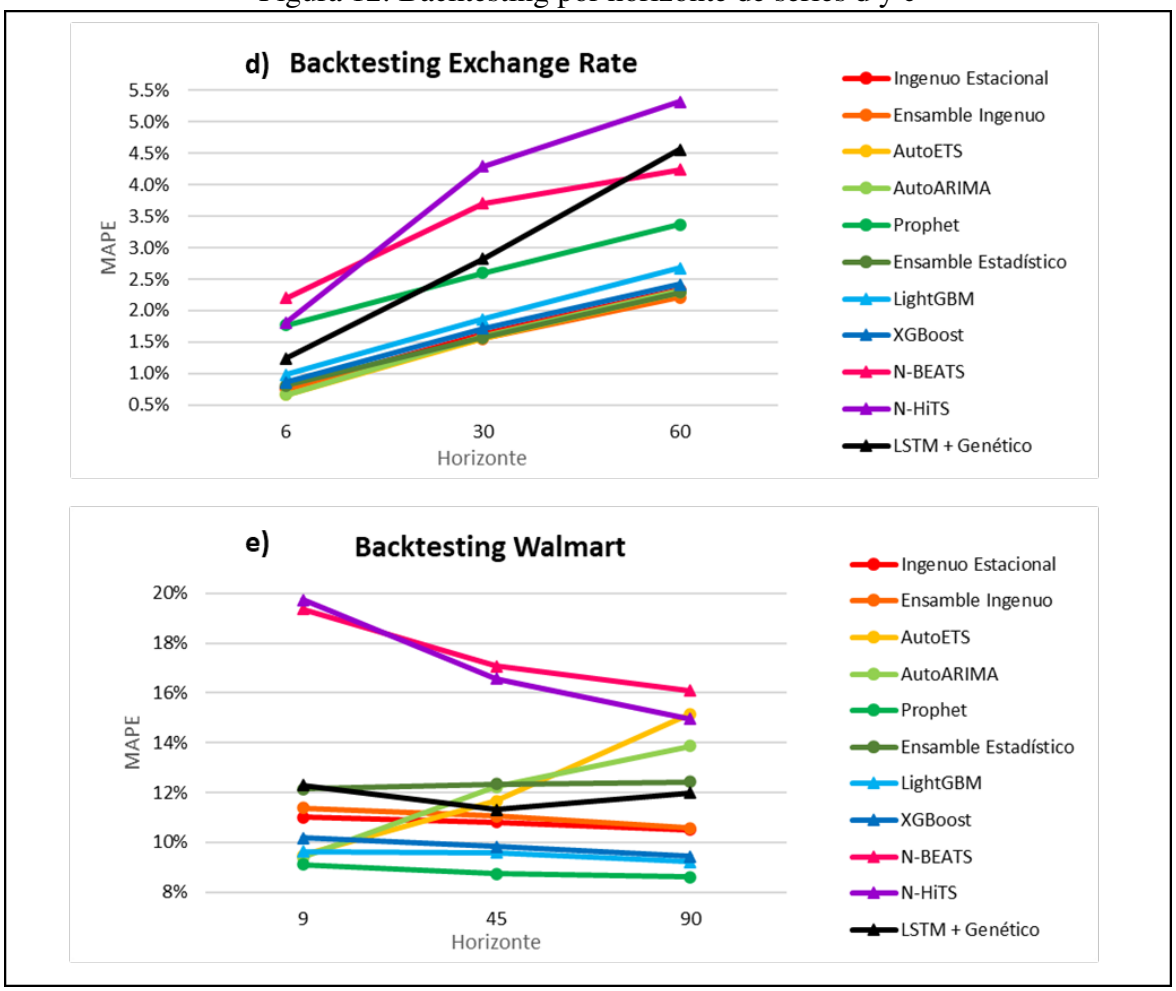


Tabla 4: Comparación de medias Games-Howell

Serie de Tiempo:	Air Passengers						Real GNP						Lynx			Exchange Rate			Walmart					
Horizonte:	CP			MP			LP			CP			MP			LP			CP			MP			LP		
Ingenuo Estacional	B			C	D		C			C			C			C	D	E	A		B	A			B			C			C	
Ensamble Ingenuo		D	E			G			E	F	B	C		B	C		B	C	D	E	A	A	B	A			D	E		B			C			C	
AutoETS		D			E			D	E		C			C				E			A		B	A			E			B			C		A	B	
AutoARIMA			F			G			E	F	C			C			D	E			A		B	A			E			B			C		D	A	B	
Prophet		E	F			G			F		B	C		C			C	D	E		A	A		A	A	B		A		B			D				D
Ensamble Estadístico		D	E			E	F		D	E	B	C		B	C		A	B	C	D	A	A	B	A			D	E		B			C		B	C		B	C
LightGBM	A			A			A			A			A			A					A		B	A			C	D		B			B	C		B	C	D
XGBoost		B	C			D	E			C	D		C			C		D	E		A		B	A			C	D	E	B			B	C		B	C	D
N-BEATS	A	B		A	B		A	B			B		A	B		A	B				A	A	B	A	A			A			A			A			A	
N-HiTS		B		A	B		A	B		A	B		A	B		A	B				A	A	B	A	A	B		A			A			A	B		A	B	
LSTM + Genético		C	D		B	C		B			C			B	C		A	B			A	A	B	A		B	C		A			B			C		B	C	

7. Conclusiones y Recomendaciones

Establecer la estructura de una red recurrente neuronal para realizar pronósticos no es trivial. El algoritmo genético propuesto demostró ser robusto para diseñar estas estructuras en comparación con una búsqueda puramente aleatoria, tomando el mismo tiempo de ejecución. Para las diversas series de tiempo, los modelos de RNN diseñados tuvieron menor error de pronóstico (MAPE) en los datos de prueba que los modelos estadísticos. Los modelos de aprendizaje automático fueron los mejores para pronosticar los datos de prueba. En cuanto a la comparación mediante *backtesting*, los modelos estadísticos mostraron tener menor error de forma consistente en los distintos horizontes de tiempo. Esto se debe principalmente a la naturaleza de los modelos. Los modelos de aprendizaje automático obtienen errores grandes al principio de *backtesting*, ya que se encuentran en una fase inicial de aprendizaje. No obstante, esas diferencias no resultaron ser significativas en algunos casos.

Los modelos estadísticos resultaron más efectivos para series lineales con estacionalidad fija. Por otro lado, los modelos de aprendizaje automático son más efectivos para series no lineales y de patrones irregulares. A pesar de que los modelos de aprendizaje automático tuvieron un peor desempeño mediante *backtesting*, este método sigue siendo fundamental para determinar los mejores hiperparámetros para estos modelos. El *backtesting* disminuye la posibilidad de sobreajustar el modelo a los datos de entrenamiento y por ende, tener mejor desempeño de generalización, para futuros pronósticos. Sin embargo, los modelos de aprendizaje automático, incluso los que entran en la categoría de aprendizaje por conjuntos, requieren de suficientes datos y de más recursos computacionales para entrenarse y lograr pronósticos de error bajo.

En conclusión, para pronosticar la demanda futura, la implementación de nuevos modelos propuestos por el aprendizaje automático puede aumentar la precisión de los pronósticos y minimizar la incertidumbre de la SCM. Se recomienda hacer un análisis de datos exploratorio de la demanda histórica y probar distintos modelos, para utilizar el que obtenga mejor desempeño. El algoritmo genético propuesto soluciona de manera adecuada el diseño de la estructura del modelo RNN. Además, el algoritmo puede usarse para diseñar estructuras para otros tipos de problemas que involucren datos secuenciales.

Es importante destacar que existieron limitaciones en el desarrollo de la propuesta. Una de ellas fue que los recursos computacionales disponibles no permitieron ejecutar con más generaciones el algoritmo genético con las series de gran tamaño. En consecuencia, se indican las siguientes oportunidades y recomendaciones:

- Incluir en la codificación de los cromosomas los hiperparámetros del modelo RNN que no son de estructura, como los *n_historicos* pasos de tiempo hacia atrás, ya que estos pueden ser decisivos para el desempeño del modelo.
- Intentar diferentes métodos del operador selección, como lo son la selección estocástica universal o por torneo. Cada uno contiene diferentes ventajas en cuanto a eficiencia de selección o a garantizar diversidad en las poblaciones.
- Modificar las tasas de elitismo, cruce y mutación y realizar replicaciones para obtener una combinación de parámetros que produzca mejores soluciones. Incluso, se puede realizar un método adaptativo de los parámetros del algoritmo si éste no encuentra una mejor solución en un límite de generaciones.

- Realizar más experimentación con series de tiempo con características y tamaños diferentes, si se tienen más recursos computacionales, en especial las unidades de procesamiento gráfico (GPU).
- Permitir a los modelos utilizar covariables estáticas y futuras, transformaciones de los datos y métodos de regularización para evitar el sobreajuste. Así se puede analizar el máximo potencial de los modelos aunque la comparación pueda ser inequitativa.

Referencias

Abraham, A. (2005). Artificial neural networks. *Handbook of measuring system design*.

Aliefa, M. H., y Suyanto, S. (2020). Variable-length chromosome for optimizing the structure of recurrent neural network. En *2020 international conference on data science and its applications (icodsa)* (pp. 1–5).

Angeline, P. J., Saunders, G. M., y Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, *5*(1), 54–65.

Aurélien, G. (2017). Hands-on machine learning with scikit-learn & tensorflow. , *134*.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. En *Neural networks: Tricks of the trade: Second edition* (pp. 437–478). Springer.

Brownlee, J. (2016). *How to backtest machine learning models for time series forecasting. machine learning mastery.* Descargado de `https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/`

Canadian lynx data. (2020). Encyclopedia of Mathematics. Descargado de `http://encyclopediaofmath.org/index.php?title=Canadian_lynx_data&oldid=46191`

Cicek, Z. I. E., y Ozturk, Z. K. (2021). Optimizing the artificial neural network parameters using a biased random key genetic algorithm for time series forecasting. *Applied Soft Computing*, *102*, 107091.

Datasets. (2023). Darts. Descargado de `https://unit8co.github.io/darts/generated_api/darts.datasets.html`

De Mulder, W., Bethard, S., y Moens, M.-F. (2014, 01). A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech Language*, *30*. doi: 10.1016/j.csl.2014.09.005

Ding, S., Su, C., y Yu, J. (2011). An optimizing bp neural network algorithm based on genetic algorithm. *Artificial intelligence review*, *36*, 153–162.

Gil, M., Rodriguez-Marin, M., y Montoya, M. A. (2020). Demand planning on small and medium-sized enterprises in mexico: A case study of a confectionery firm. *Adv. Bus.-Relat. Sci. Res. J*, *11*, 56–73.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

Han, S.-S., y May, G. S. (1996). Optimization of neural network structure and learning parameters using genetic algorithms. En *Proceedings eighth ieee international conference on tools with artificial intelligence* (pp. 200–206).

Hanke, J. E., y Wichern, D. W. (2005). *Business forecasting*. Pearson Educación.

Hochreiter, S., y Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Ivanov, D., Tsipoulanidis, A., y Schönberger, J. (2019). *Global supply chain and operations management: A decision-oriented introduction to the creation of value* (Vol. 2). Springer.

Janiesch, C., Zschech, P., y Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685–695.

Johnson, F., Valderrama, A., Valle, C., Crawford, B., Soto, R., y Nanculef, R. (2020). Automating configuration of convolutional neural network hyperparameters using genetic algorithm. *IEEE Access*, *8*, 156139–156152.

Khanzode, K. C. A., y Sarode, R. D. (2020). Advantages and disadvantages of artificial intelligence and machine learning: A literature review. *International Journal of Library & Information Science (IJLIS)*, *9*(1), 3.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex System*, *4*(4), 461–476.

Leung, F. H.-F., Lam, H.-K., Ling, S.-H., y Tam, P. K.-S. (2003). Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, *14*(1), 79–88.

Liu, H. (2015). Forecasting model of supply chain management based on neural network. En *2015 international conference on automation, mechanical control and computational engineering* (pp. 179–183).

Mallawaarachchi, V. (2017). *Introduction to genetic algorithms. towards data science.* Descargado de `https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf33`

Mejía Villamizar, J. C., Palacio León, Ó., y Adarme Jaimes, W. (2013). Efecto látigo en la planeación de la cadena de abastecimiento, medición y control. *Ciencia e Ingeniería Neogranadina*, *23*(2), 37–54.

Mezard, M., y Nadal, J.-P. (1989). Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A: Mathematical and General*, *22*(12), 2191.

Miller, G. F., Todd, P. M., y Hegde, S. U. (1989). Designing neural networks using genetic algorithms. En *Icga* (Vol. 89, pp. 379–384).

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Moroff, N. U., Kurt, E., y Kamphues, J. (2021). Machine learning and statistics: A study for assessing innovative demand forecasting models. *Procedia Computer Science*, *180*, 40–49.

Nikbakht, S., Anitescu, C., y Rabczuk, T. (2021). Optimizing the neural network hyperparameters utilizing genetic algorithm. *Journal of Zhejiang University-Science A*, *22*(6), 407–426.

Oliker, S. (1992). A distributed genetic algorithm for neural network design and training. *Complex systems*.

Petropoulos, F., Apiletti, D., Assimakopoulos, V., Babai, M. Z., Barrow, D. K.,

Taieb, S. B., ... others (2022). Forecasting: theory and practice. *International Journal of Forecasting*, *38*(3), 705–871.

Pramoditha, R. (2022). *Classification of neural network hyperparameters. towards data science.* Descargado de https://towardsdatascience.com/classification-of-neural-network-hyperparameters-c7991b6937c3

Real, E. (2018). *Using evolutionary automl to discover neural network architectures. google ai blog.*

Real gross national product [gnpc96]. (2023). Federal Reserve Bank of St. Louis. Descargado de https://fred.stlouisfed.org/series/GNPC96

Schneider, P., y Xhafa, F. (2022). Chapter 8 - machine learning: Ml for ehealth systems. En P. Schneider y F. Xhafa (Eds.), *Anomaly detection and complex event processing over iot data streams* (p. 149-191). Academic Press.

Store item demand forecasting challenge. (2023). Kaggle. Descargado de https://www.kaggle.com/competitions/demand-forecasting-kernels-only

Szablowski, B. (2023). *Forecast multiple time series like a master. towards data science.* Descargado de https://towardsdatascience.com/forecast-multiple-time-series-like-a-master-1579a2b6f18d

United states dollar to pound sterling. (2023). Google Finance. Descargado de https://www.google.com/finance/quote/USD-GBP

Yu, Y., Si, X., Hu, C., y Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, *31*(7), 1235–1270.