ALGORITMOS GENÉTICOS EN LA OPTIMIZACIÓN DE REDES NEURONALES LSTM PARA LA PREDICCIÓN DE SERIES TEMPORALES FINANCIERAS

Abstract
La predicción de series temporales financieras representa un desafío debido a su naturaleza no lineal y a la influencia de múltiples factores externos. En este trabajo se explora el uso de redes neuronales recurrentes del tipo LSTM, optimizadas mediante algoritmos genéticos para mejorar la precisión en tareas de pronóstico. La optimización de hiperparámetros se plantea como un aspecto crítico para el desempeño de estos modelos, dado que determina su capacidad de generalización y adaptación a datos complejos. Se implementó una solución en Python utilizando datos históricos de la acción de YPF como caso de estudio. Los resultados evidencian que la combinación de LSTM y algoritmos genéticos constituye una estrategia efectiva para abordar la predicción de series financieras, aportando un marco metodológico aplicable a distintos contextos.

Palabras clave
Algoritmos genéticos - Redes neuronales recurrentes - LSTM - Optimización de Hiperparámetros - Teoría del Caos - Machine Learning - Series temporales financieras

1. Introducción

La predicción de series temporales financieras es desafiante por su no estacionariedad, ruido y volatilidad, con rasgos compatibles con dinámicas no lineales e incluso caóticos[1] [1], lo que limita a modelos lineales como ARIMA[2] para dependencias de largo alcance. Los avances en aprendizaje profundo, en particular las redes recurrentes del tipo **Long Short-Term Memory (LSTM)**, permiten modelar relaciones no lineales en secuencias y han mostrado buen desempeño en finanzas.

El rendimiento de estas redes depende de la elección de hiperparámetros (profundidad, unidades, tasa de aprendizaje, regularización, tamaño de lote, etc.). En este trabajo abordamos su ajuste mediante **algoritmos genéticos (AG)**, un enfoque evolutivo que explora configuraciones candidatas y selecciona las de mejor desempeño. Evaluamos la combinación LSTM y AG en la serie de precios de YPF, comparando un modelo base con uno optimizado y cuantificando el impacto sobre el error de pronóstico.

2. Fundamentos Teóricos

2.1. Redes LSTM

Las Long Short-Term Memory (LSTM), propuestas por Hochreiter y Schmidhuber en 1997 [2], son una variante de las redes neuronales recurrentes (RNN) diseñada para superar el problema del gradiente desvaneciente [3]. Incorporan una **celda de memoria** que retiene información relevante durante largos intervalos y se regula mediante tres puertas: entrada, salida y olvido, que controlan qué información se incorpora, se utiliza y se descarta [4]. Gracias a esta arquitectura, las LSTM capturan patrones temporales complejos y se aplican ampliamente en predicción financiera y otros dominios secuenciales.

[1] Se entiende por caótico aquel sistema no lineal sensible a condiciones iniciales, lo que lo hace aparentemente aleatorio e impredecible a largo plazo, propio de la teoría del caos.
[2] Autoregressive Integrated Moving Average (Media Móvil Integrada Autorregresiva) y es un modelo estadístico utilizado para el análisis y la predicción de datos de series temporales.

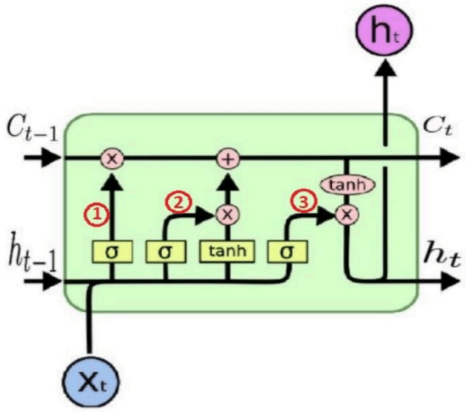

Figura 2. *Celda LSTM con sus tres puertas principales: olvido (1), entrada (2) y salida (3), encargadas de regular la información que se conserva, se actualiza y se utiliza en cada paso temporal.*

2.2. Hiperparámetros

En *machine learning*, se entiende por hiperparámetro a una variable que es externa al modelo y que debemos definir manualmente al momento de programar el algoritmo de entrenamiento. Son diferentes de los parámetros, que son elementos internos derivados de manera automática durante el proceso de aprendizaje, como por ejemplo los pesos de los nodos.

La correcta elección de los hiperparámetros es determinante para el desempeño de una red. Existen diversas técnicas de búsqueda y optimización que buscan encontrar configuraciones adecuadas para maximizar la capacidad predictiva del modelo.

Entre los hiperparámetros más comunes en redes neuronales se encuentran:

- **Número de capas ocultas (*layers*)**:

Determina su profundidad, lo que afecta a su complejidad y capacidad de aprendizaje. Menos capas hacen que el modelo sea más simple y rápido, pero más capas conducen a una mejor clasificación de los datos de entrada. Identificar aquí el valor óptimo es un compromiso entre velocidad y precisión. [5]

- **Número neuronas por capa (*units*)**:

Establece la amplitud de cada capa. Un mayor número de neuronas incrementa la capacidad de la red para aprender relaciones complejas, aunque también puede aumentar el riesgo de sobreajuste[3] y la demanda computacional. [5]

- **Tasa de aprendizaje (*learning rate*)**:

Regula el grado en el que un modelo ajusta sus parámetros en cada paso de su algoritmo de optimización, cuyo objetivo es minimizar la función de pérdida[4]. Cada vez que el modelo ejecuta su algoritmo de optimización, actualiza sus parámetros en función del resultado. [6]

Una tasa de aprendizaje alta permite un avance rápido pero puede volver inestable el entrenamiento. En cambio, una tasa de aprendizaje baja ajusta los parámetros de manera más gradual, lo que genera un proceso más lento pero estable, con mayor probabilidad de converger hacia un resultado adecuado.

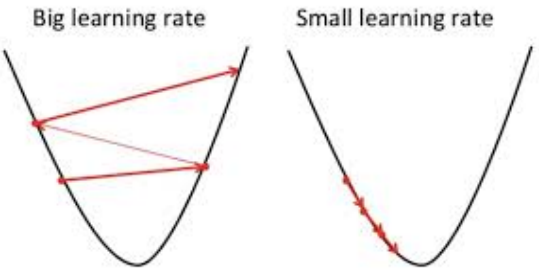

Figura 3. *Comparación del efecto de una tasa de aprendizaje grande y una tasa de*

[3] El sobreajuste (*overfitting*) ocurre cuando un modelo aprende en exceso los detalles y el ruido del conjunto de entrenamiento, perdiendo capacidad de generalizar correctamente a datos nuevos.

[4] La función de pérdida (*loss function*) es una medida matemática que cuantifica la diferencia entre las predicciones del modelo y los valores reales. Un valor más bajo indica que el modelo está realizando mejores predicciones.

aprendizaje pequeña con respecto a la función de pérdida y a su mínimo global.

- ***Tamaño de Lote:***

Siguiendo la línea de la tasa de aprendizaje, ya que ambos influyen directamente en la dinámica de actualización de los parámetros, el tamaño de lote (*batch size*) define cuántos ejemplos se procesan antes de actualizar los parámetros.

- ***Épocas* (iteraciones)*:***

Este es el número de veces que se van a pasar cada ejemplo de entrenamiento por la red. Es fácil caer en la tentación de pensar que un alto número de Epoch puede hacer que la red aprenda más fácil, pero lotes pequeños generan actualizaciones más ruidosas, y grandes, más estables pero costosas. [7]

- ***Dropout:***

Es una técnica de regularización, es decir, un conjunto de métodos destinados a mejorar la capacidad de generalización[5] del modelo. En la práctica, consiste en desactivar aleatoriamente un porcentaje de neuronas durante el entrenamiento. De este modo se evita que la red dependa en exceso de conexiones específicas y se fuerza a que aprenda representaciones más robustas. [8]

2.3. Algoritmos Genéticos

Los algoritmos genéticos (AG) son algoritmos de búsqueda heurística[6] adaptativa que pertenecen a la mayor parte de los algoritmos evolutivos. Estos algoritmos simulan el proceso de selección natural, que se basa en que los individuos más aptos de una población son los que sobreviven y pasan a la siguiente generación, al adaptarse más fácilmente a los cambios que se producen en su entorno. [9]

Cada generación consta de una **población** de individuos , y cada individuo representa un punto en el **espacio de búsqueda** y una posible solución. Cada individuo se codifica como un vector de longitud finita (similar a un cromosoma) de componentes. Estos componentes variables son análogos a los genes. **Por lo tanto, un cromosoma (individuo) se compone de varios genes (componentes variables)**. [10]

Figura 4. *Representación de una población, cromosoma y gen.*

2.3.1. Función de aptitud (*fitness function*)

La función de aptitud **tiene como objetivo medir qué tan adecuada es una solución candidata con respecto al problema planteado.** Cada individuo de la población se evalúa mediante la función de aptitud, siendo aquéllos con el mejor puntaje los que tienen más probabilidades de ser seleccionados como “padres”, transmitiendo sus características a la siguiente generación.

La definición concreta de la aptitud depende del problema en cuestión. Puede

[5] En *machine learning*, la generalización es la capacidad de un modelo para aplicar correctamente su aprendizaje a datos nuevos y nunca antes vistos.

[6] Procedimiento que utiliza reglas prácticas o aproximadas para explorar soluciones en un problema complejo, sin garantizar la mejor solución absoluta pero encontrando resultados satisfactorios en un tiempo razonable.

tratarse, por ejemplo, del error cuadrático medio en un problema de predicción, o de la distancia recorrida en un problema de rutas. En todos los casos, traduce el objetivo del problema en una medida cuantificable que permite comparar las soluciones entre sí.

2.3.2. Operadores Genéticos

Una vez creada la generación inicial, el algoritmo la desarrolla utilizando los siguientes operadores.

2.3.2.1. Selección.

El operador de selección elige a los individuos que serán padres, favoreciendo a los de mayor aptitud para que transmitan sus genes a generaciones sucesivas.

2.3.2.2. Operador de cruce (*crossover*).

El *crossover* representa el apareamiento entre individuos. Se seleccionan dos individuos mediante el operador de selección y se eligen aleatoriamente los sitios de cruce. Posteriormente, se intercambian los genes en estos sitios de cruce, creando así un individuo completamente nuevo (descendencia). [10]

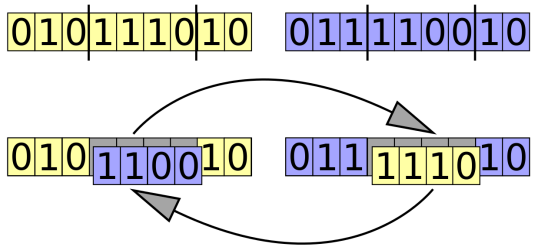

Figura 5. *Ejemplo de crossover, donde dos cromosomas intercambian segmentos de genes para generar descendencia.*

2.3.2.3. Mutación

Introduce variaciones aleatorias en los hijos para mantener la diversidad de la población y evitar la convergencia prematura. Se aplica con una probabilidad por gen o por individuo.

La tasa de mutación suele ser baja en comparación con la tasa de crossover.

3. Desarrollo del Trabajo

3.1. Conjunto de datos

Se utilizaron cotizaciones históricas de YPF descargadas mediante la librería *yfinance.* Nuestra variable objetivo es el precio de cierre diario. Del total, se trabajó con los últimos 2000 cierres diarios (8 años) , que resulta suficiente para capturar patrones representativos de la dinámica reciente del activo y, al mismo tiempo, garantiza que el entrenamiento del modelo sea factible en términos de tiempo y recursos. Trabajar con series excesivamente largas no siempre aporta mejoras sustanciales en el desempeño.

3.2. Preprocesamiento de los datos.

El procesamiento incluyó la normalización de la serie mediante un escalado *Min-Max*[7] en el rango [0,1], tal como sugiere la bibliografía especializada. Este paso es fundamental porque las redes LSTM entrenan de manera más estable cuando los datos se encuentran en un rango reducido. [11]

Posteriormente, la serie normalizada se transformó en ejemplos supervisados mediante el uso de ventanas deslizantes. Cada ventana de observación estuvo compuesta por los últimos 20 días y el

[7] El escalado Min–Máx funciona restando el valor mínimo de la serie y dividiendo por el rango (máximo–mínimo).

modelo se entrenó para predecir los 5 días siguientes. En la práctica, esto significa que el modelo "ve" una secuencia de 20 precios pasados y aprende a generar una predicción para los 5 precios posteriores.

Este enfoque ubica nuestro trabajo en el marco de los modelos de predicción multi-paso, que generan más de un valor futuro por cada secuencia de entrada. En contraste, los modelos de un solo paso producen únicamente el valor inmediato siguiente a partir de la ventana de entrada. [11]

En este esquema, el 70 % de los datos más antiguos se destinó al entrenamiento, el 15 % siguiente a la validación y el 15 % restante, correspondiente a los datos más recientes, a la prueba final. La lógica es: el modelo se entrena con información pasada, ajusta su desempeño en un conjunto intermedio que simula datos aún no vistos, y finalmente se evalúa en los datos más recientes, reproduciendo de manera más realista la situación de pronosticar hacia adelante.

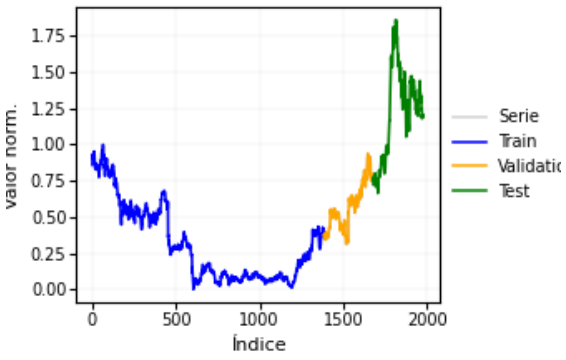

Figura 7. *División de la serie temporal en subconjuntos de entrenamiento (70 %), validación (15 %) y prueba (15 %). Los valores fueron previamente normalizados.*

De esta manera, el conjunto de datos queda preparado para ser utilizado en la construcción y entrenamiento del modelo LSTM.

3.3. Configuración del modelo base

En predicción financiera con series diarias, las configuraciones poco profundas (1–2 capas) y con unidades por capa moderadas (≈32–128 unidades) son prácticas comunes. [11]

El modelo base se implementó como una red LSTM de **2 capas** con **50 unidades por capa** y para la salida, una capa densa[8] dimensión 5 para producir, en un único paso, las predicciones correspondientes a los cinco días futuros del horizonte definido.

El entrenamiento se realizó con Adam como algoritmo optimizador y **tasa de aprendizaje** $\mathbf{1\times10^{-3}}$, pérdida MSE y métrica RMSE. Se empleó un máximo de **100 épocas**, controladas por parada temprana[9].

Se optó por utilizar la **Raíz del Error Cuadrático Medio (RMSE)** como métrica principal de evaluación, dado que proporciona una medida interpretable en las mismas unidades que la variable objetivo (dólares estadounidenses) y penaliza con mayor fuerza los errores grandes.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Figura 8. *Cálculo del error: y representa el valor real de la serie (observado en los*

[8] Una capa densa (fully connected) es una capa en la que cada neurona está conectada a todas las neuronas de la capa anterior, lo que permite combinar la información extraída en las etapas previas.

[9] La parada temprana (*early stopping*) interrumpe el entrenamiento cuando la mejora en el conjunto de validación deja de ser significativa, evitando sobreajuste y ahorro de recursos.

datos), $\widehat{y}$ *corresponde al valor predicho por el modelo.*

Estas decisiones proporcionan un punto de partida razonable sobre el cual evaluar el aporte de la optimización de hiperparámetros mediante algoritmos genéticos.

3.4. Optimización de hiperparámetros con AG

Para optimizar la configuración de la red LSTM se implementó un Algoritmo Genético (AG) que explora combinaciones de hiperparámetros y selecciona aquellas que minimizan el error de validación.

3.4.1. Espacio de búsqueda

Cada individuo es un conjunto de hiperparámetros. Se consideraron como genes los hiperparámetros más influyentes del modelo:

- Capas de la red (entero, 1–3)
- Número de neuronas por capa (entero, 32–96)
- Épocas (entero, 40–90)
- Tasa de aprendizaje (float, 0,0001 - 0,003)
- Tasa de dropout (float, 0.05–0.35)
- Tamaño de lote (categórico: {16, 32, 64})

3.4.2. Configuración del AG

Se usó una población de 8 individuos, elitismo[10] de 2 mejores por generación, torneo[11] de tamaño 3 como operador de selección, cruce con probabilidad 0.9 y mutación con probabilidad 0.2 por gen.

Cruce: intercambio 50/50 en genes enteros/categóricos; para reales se promedia entre los padres y se añade ruido gaussiano[12] acotado.

Mutación: perturbaciones pequeñas y acotadas:

- ±1 capa.
- ±4/±8 unidades.
- ±5/±10 épocas.
- Vecino en tamaño de lote (por ej: si está en 16, pasa a 32)
- Ruido gaussiano en dropout
- Se multiplica por factor *log-normal*[13] en la tasa de aprendizaje.

En esta corrida se fijaron 2 generaciones

La **función de aptitud** es el ***RMSE* promedio** normalizado en validación, promediado sobre los cinco horizontes $t+1 \dots t+5$ (menor es mejor). Además, registramos esta métrica en USD por razones de interpretabilidad.

El criterio de parada es un número máximo de generaciones.

3.4.3 Resultados.

El modelo base, con dos capas LSTM de 50 unidades, **alcanzó un *RMSE promedio* de 2.62 USD** en el conjunto de prueba, que para tener una referencia, es un 9.15% del valor de la acción a día de hoy (10-09-2025), 28.64 USD. [12] Su

---

[10] El elitismo es una estrategia en algoritmos genéticos que consiste en conservar directamente los mejores individuos de una generación para garantizar que no se pierda la calidad de las soluciones en el proceso evolutivo.

[11] El torneo es un método de selección en algoritmos genéticos que consiste en elegir aleatoriamente un grupo de individuos y seleccionar como padre al que presente la mejor aptitud dentro de dicho grupo.

[12] El ruido gaussiano se refiere a una perturbación aleatoria extraída de una distribución normal, caracterizada por su media y varianza. En los algoritmos genéticos se utiliza para introducir variaciones pequeñas y controladas en los valores reales de los genes.

[13] El factor log-normal corresponde a un valor aleatorio generado de una distribución log-normal. Permite escalar hiperparámetros positivos en órdenes de magnitud pequeños, preservando siempre la positividad.

principal limitación fue la dificultad para reaccionar ante cambios bruscos, generando predicciones más suaves y con cierto desfase, lo que en la práctica se traduce en un menor poder predictivo.

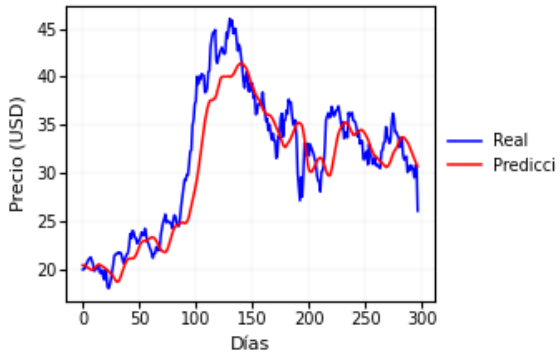

Figura 9. *Predicción del conjunto de prueba utilizando el modelo base.*

Tras la optimización, el modelo pasó a tener solo una capa, pero con 83 unidades. Esta modificación redujo la profundidad, evitando redundancias y sobreajuste, y **resultando en un modelo más simple**. Mientras que el aumento de unidades en la única capa disponible amplió su capacidad de memoria, logrando capturar dependencias temporales de mayor alcance.

Otro ajuste notorio, además de la arquitectura, fue la reducción del batch size de 32 a 16, lo que permitió actualizaciones más frecuentes y sensibles a las variaciones de los datos.

Los cambios en dropout y la tasa de aprendizaje *(LR)* también contribuyeron, aunque de forma secundaria, al modelo final.

Hiperparámetro	Base	Optim. AG
Capas LSTM	2	1
Unidades/capa	50	83
Dropout	0.20	0.105
LR	0.001	0.001418
Batch	32	16
Épocas	100	60

Tabla 1. *Comparación de los hiperparámetros empleados en el modelo base y en el modelo optimizado mediante algoritmos genéticos (AG).*

En conjunto, estos cambios se reflejan en un *RMSE* promedio que disminuye a 2.03 USD, aproximadamente un 22.5% de mejora. Más allá de la métrica, en la Figura 10 se observa que el modelo optimizado logra seguir con mayor precisión las oscilaciones de la serie, ajustándose mejor en los tramos de alta volatilidad y reduciendo el rezago respecto de la dinámica real.

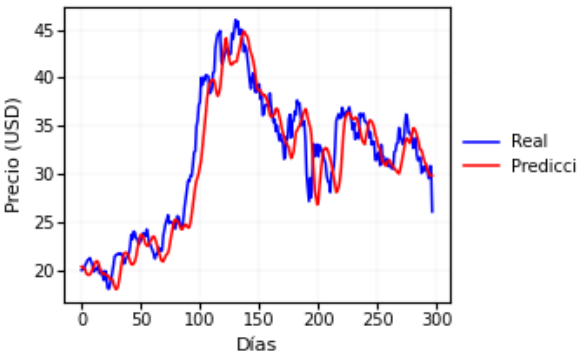

Figura 10. *Predicción del conjunto de prueba utilizando el modelo optimizado.*

4. Conclusión

La aplicación de algoritmos genéticos en la optimización de hiperparámetros de redes LSTM **demostró ser una estrategia eficaz para mejorar el desempeño en la predicción de series temporales financieras,** reduciendo el error en un 22,5

% respecto del modelo base. Los resultados muestran que esta técnica puede encontrar configuraciones más adecuadas que las seleccionadas manualmente. No obstante, esta mejora **conlleva un costo computacional elevado**, ya que los algoritmos genéticos requieren múltiples evaluaciones de modelos a lo largo de generaciones para explorar el espacio de búsqueda.

En síntesis, el estudio confirma la utilidad de los algoritmos evolutivos como herramienta para la optimización automática de hiperparámetros en problemas financieros, siempre que se cuente con el tiempo y los recursos necesarios.

5. Trabajos Futuros

Entre las posibles líneas a seguir se incluyen:

1. **Explorar otros métodos de optimización** de hiperparámetros con el objetivo de comparar su desempeño y eficiencia frente a los algoritmos genéticos.

2. **Ampliar el modelo hacia un enfoque multivariado**, incorporando variables adicionales que puedan influir en el precio de la acción.

6. Referencias

[1] Faggini, M., Parziale, A., & Viviani, A. (2019). Does chaos matter in financial time series analysis? *International Journal of Economics and Financial Issues.* https://www.econjournals.com/index.php/ijefi/article/download/8058/pdf/20029

[2] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*

[3] Wikipedia. (s.f.). *Red neuronal recurrente.* https://es.wikipedia.org/wiki/Red_neuronal_recurrente

[4] Wikipedia. (s.f.). *Memoria larga a corto plazo.* https://es.wikipedia.org/wiki/Memoria_larga_a_corto_plazo

[5] IBM. (s.f.). *Ajuste de hiperparámetros (Hyperparameter tuning).* https://www.ibm.com/es-es/think/topics/hyperparameter-tuning

[6] IBM. (s.f.). *Learning rate.* https://www.ibm.com/think/topics/learning-rate

[7] Casas González, J. (2021). *Tuneando los hiperparámetros de una red neuronal LSTM.* LinkedIn. https://www.linkedin.com/pulse/tuneando-los-hiperpar%C3%A1metros-de-una-red-neuronal-lstm-casas-gonzalez/

[8] Coursera. (s.f.). *Dropout in neural networks: A simple way to prevent overfitting.* https://www.coursera.org/articles/dropout-neural-network

[9] Díaz, D. E. (2013). *Introducción a los algoritmos genéticos.* Universidad Tecnológica Nacional, Facultad Regional Rosario. Apunte de cátedra.

[10] GeeksforGeeks. (s.f.). *Genetic algorithms.* https://www.geeksforgeeks.org/dsa/genetic-algorithms/

[11] TensorFlow. (s.f.). *Time series forecasting.* https://www.tensorflow.org/tutorials/structured_data/time_series?hl=es-419

[12] Yahoo Finance. (s.f.). *YPF S.A. (YPF).* https://es.finance.yahoo.com/quote/YPF/