Universidad Nacional
SAN LUIS GONZAGA

UNIVERSIDAD NACIONAL "SAN LUIS GONZAGA"
VICERRECTORADO DE INVESTIGACIÓN
Facultad de Ciencias

"Método adaptativo para mejorar la velocidad del entrenamiento de una Red Neuronal Artificiales"

Línea de Investigación
Ciencias Naturales, Ingeniería y Tecnologías Sostenibles

PROYECTO DE INVESTIGACIÓN

Investigador Principal: Dr. Javier Eduardo Magallanes Yui, Docente Principal a Dedicación Exclusiva de la Universidad Nacional "San Luis Gonzaga" Adscrito a la Facultad de Ciencias

0000-0003-0943-9186

Investigador Asociado: Dr. Morales Almora, José Luis, Docente Principal a Dedicación Exclusiva de la Universidad Nacional "San Luis Gonzaga" Adscrito a la Facultad de Ciencias

0000-0003-4089-4320

Investigador Asociado: Dr. Huamaní Licas, Máximo, Docente Principal Dedicación Exclusiva de la Universidad Nacional "San Luis Gonzaga" Adscrito a la Facultad de Ciencias

0000-0003-3466-7615

Investigador Colaborador: Mg. Landeo Alfaro, Elmer Leonidas, Docente Auxiliar a Tiempo Parcial de la Universidad Nacional "San Luis Gonzaga" Adscrito a la Facultad de Ciencias.

0000-0003-1569-5629

.

Ica – Perú – 2025

1. RESUMEN

El proyecto desarrollado tuvo como propósito mejorar la velocidad de entrenamiento de una red neuronal artificial mediante el análisis, implementación y comparación de técnicas avanzadas de optimización, así como la formulación de un método adaptativo capaz de acelerar la convergencia del modelo sin comprometer la precisión. A lo largo del estudio se revisaron exhaustivamente los fundamentos matemáticos de la optimización, así como los principales algoritmos utilizados en redes neuronales, con el fin de identificar limitaciones y oportunidades para la construcción de un esquema eficiente de aprendizaje automático.

El punto de partida del proyecto fue el análisis de la estructura matemática del proceso de entrenamiento, entendido como la resolución de un problema de optimización donde se busca minimizar una función de pérdida mediante la actualización iterativa de los pesos y sesgos del modelo. Se estudió el comportamiento de la función objetivo en tres escenarios: clasificación binaria, clasificación multiclase y regresión, concluyéndose que para problemas con estructura temporal, como las series cronológicas o la previsión continua, la métrica más adecuada es el Error Cuadrático Medio (MSE). Este enfoque permitió formalizar el entrenamiento como una tarea de regresión supervisada, proporcionando el marco teórico necesario para evaluar la eficiencia de diversos algoritmos optimizadores.

Durante el desarrollo del proyecto se evidenció que el rendimiento del proceso de entrenamiento depende críticamente de factores como la tasa de aprendizaje, el tipo de optimizador, la arquitectura de la red y la calidad de la inicialización. En este sentido, se analizaron metodologías de optimización local—basadas en gradientes—como SGD, Momentum, Adagrad, RMSProp y Adam, así como técnicas de búsqueda global, entre ellas optimización bayesiana, algoritmos evolutivos, optimización por enjambres de partículas y evolución diferencial. Se constató que los métodos basados en gradientes continúan siendo los más efectivos para entrenar redes neuronales profundas por su eficiencia computacional, su escalabilidad y su capacidad para manejar grandes conjuntos de datos, aunque presentan desventajas como la sensibilidad a la inicialización y el riesgo de quedar atrapados en óptimos locales.

En contraste, los algoritmos de búsqueda global demostraron ser útiles para explorar hiperparámetros complejos y regiones amplias del espacio de soluciones. Sin embargo, debido a su elevado costo computacional, se concluyó que su uso directo para entrenar una red neuronal completa es poco práctico y puede conducir al sobreajuste si no se controla adecuadamente. Por ello, el enfoque más eficiente identificado en el proyecto consiste en utilizar algoritmos globales para determinar configuraciones iniciales prometedoras y luego emplear optimizadores basados en gradientes para el refinamiento final del modelo. Este hallazgo resultó fundamental para el diseño del método adaptativo propuesto.

Otro resultado importante del proyecto fue la identificación precisa de la influencia de diversos hiperparámetros sobre la estabilidad del entrenamiento. Se evaluó el impacto del número de capas, la cantidad de neuronas por capa, la selección de la función de activación, el tamaño del lote, los mecanismos de regularización y la tasa de aprendizaje. Se comprobó que pequeñas variaciones en estos valores pueden generar diferencias significativas en la rapidez de convergencia de la red. Este análisis permitió determinar que los algoritmos de optimización adaptativos, especialmente Adam y RMSProp, logran un aprendizaje más estable al ajustar automáticamente los pasos de actualización de acuerdo con el comportamiento reciente de los gradientes.

Asimismo, se realizó un análisis detallado de las funciones de pérdida para cada tipo de problema. Para clasificación binaria y multiclase se profundizó en la entropía cruzada y

su relación con las funciones de activación sigmoide y softmax. Para regresión, se justificó el uso del MSE por su suavidad, diferenciabilidad y adecuación al entrenamiento mediante retropropagación del error. Este examen permitió establecer las bases matemáticas que sustentan la formulación del método adaptativo y orientaron la selección de las métricas de evaluación utilizadas.

El proyecto también incluyó un estudio comparativo entre algoritmos de optimización locales y globales, demostrando que la mejor estrategia consiste en utilizar métodos híbridos, donde la búsqueda global permite aproximarse rápidamente a regiones de interés del espacio de soluciones y los métodos basados en gradiente refinan la solución final de forma eficiente. Los experimentos desarrollados revelaron que los optimizadores adaptativos basados en gradientes presentan la mayor velocidad de convergencia, mientras que los algoritmos globales permiten mejorar la exploración cuando el modelo presenta múltiples mínimos locales.

Entre los hallazgos más relevantes se encuentra que la formulación del problema como una tarea de regresión—particularmente para series temporales—facilita la implementación de un esquema adaptativo de optimización más robusto. Este enfoque posibilita la predicción de valores continuos con mayor estabilidad, permitiendo construir redes neuronales capaces de capturar dependencias temporales complejas sin requerir arquitecturas recurrentes sofisticadas. El comportamiento del MSE como función objetivo fue fundamental para evaluar la convergencia del modelo y comparar el rendimiento de los distintos optimizadores.

Finalmente, el avance acumulado en los informes mensuales permitió comprobar que la hipótesis principal del proyecto es válida**:** sí es posible mejorar significativamente la velocidad del entrenamiento de una red neuronal artificial mediante técnicas adaptativas de optimización**,** especialmente cuando se combinan métodos basados en gradientes con estrategias eficientes de ajuste de hiperparámetros. El estudio también evidenció que la regularización adecuada evita el sobreajuste y que la optimización de la arquitectura contribuye a obtener modelos más ligeros, estables y eficientes.

En conjunto, los resultados alcanzados representan una contribución sólida al campo del aprendizaje automático en la región, ya que establecen un marco teórico y metodológico para la implementación de redes neuronales optimizadas en diferentes aplicaciones. Asimismo, sientan las bases para desarrollar futuras líneas de investigación orientadas a la creación de algoritmos híbridos, la automatización del ajuste de hiperparámetros y la implementación de técnicas de aprendizaje profundo en problemas reales que requieren soluciones rápidas y eficientes.

2. SUMMARY

The project was developed with the purpose of improving the training speed of an artificial neural network through the analysis, implementation, and comparison of advanced optimization techniques, as well as the formulation of an adaptive method capable of accelerating model convergence without compromising accuracy. Throughout the study, an exhaustive review was conducted on the mathematical foundations of optimization and the main algorithms used in neural networks, with the aim of identifying limitations and opportunities for constructing an efficient machine learning framework.

The starting point of the project was the analysis of the mathematical structure of the training process, understood as the solution of an optimization problem in which the goal is to minimize a loss function by iteratively updating the model's weights and biases. The behavior of the objective function was examined in three scenarios: binary classification, multiclass classification, and regression, concluding that for problems with temporal structure—such as time series or continuous forecasting—the most appropriate metric is the Mean Squared Error (MSE). This approach allowed the training process to be

formalized as a supervised regression task, providing the theoretical foundation necessary to evaluate the efficiency of various optimization algorithms.

During the development of the project, it became evident that training performance depends critically on factors such as the learning rate, the choice of optimizer, the network architecture, and the quality of initialization. In this regard, local optimization methods—gradient-based techniques—such as SGD, Momentum, Adagrad, RMSProp, and Adam were analyzed, as well as global search techniques including Bayesian optimization, evolutionary algorithms, particle swarm optimization, and differential evolution. It was confirmed that gradient-based methods remain the most effective for training deep neural networks due to their computational efficiency, scalability, and ability to handle large datasets, although they present disadvantages such as sensitivity to initialization and the risk of becoming trapped in local minima.

In contrast, global search algorithms proved useful for exploring complex hyperparameters and broad regions of the solution space. However, due to their high computational cost, it was concluded that using them directly to train a full neural network is impractical and may lead to overfitting if not properly controlled. Therefore, the most efficient approach identified in the project consists of using global algorithms to determine promising initial configurations and then employing gradient-based optimizers for the final refinement of the model. This finding was fundamental for the design of the proposed adaptive method.

Another important result of the project was the precise identification of the influence of various hyperparameters on training stability. The impact of the number of layers, the number of neurons per layer, the choice of activation function, batch size, regularization mechanisms, and learning rate was evaluated. It was found that small variations in these values can generate significant differences in the network's convergence speed. This analysis revealed that adaptive optimization algorithms, especially Adam and RMSProp, achieve more stable learning by automatically adjusting update steps according to the recent behavior of the gradients.

A detailed analysis of loss functions for each type of problem was also carried out. For binary and multiclass classification, cross-entropy and its relationship with the sigmoid and softmax activation functions were examined. For regression, the use of MSE was justified due to its smoothness, differentiability, and suitability for training via backpropagation. This examination established the mathematical foundations underpinning the formulation of the adaptive method and guided the selection of the evaluation metrics used.

The project also included a comparative study between local and global optimization algorithms, demonstrating that the best strategy is to use hybrid methods, where global search allows rapid approximation to promising regions of the solution space and gradient-based methods efficiently refine the final solution. The experiments carried out revealed that adaptive gradient-based optimizers show the fastest convergence, while global algorithms improve exploration when the model presents multiple local minima.

Among the most relevant findings is that formulating the problem as a regression task—particularly for time series—facilitates the implementation of a more robust adaptive optimization scheme. This approach enables the prediction of continuous values with greater stability, allowing the construction of neural networks capable of capturing complex temporal dependencies without requiring sophisticated recurrent architectures. The behavior of MSE as an objective function was essential for evaluating model convergence and comparing the performance of different optimizers.

Finally, the accumulated progress in the monthly reports demonstrated that the project's main hypothesis is valid: it is indeed possible to significantly improve the training speed

of an artificial neural network through adaptive optimization techniques, especially when gradient-based methods are combined with efficient hyperparameter-tuning strategies. The study also showed that proper regularization prevents overfitting and that optimizing the architecture contributes to obtaining lighter, more stable, and more efficient models.
Overall, the results achieved represent a solid contribution to the field of machine learning in the region, as they establish a theoretical and methodological framework for the implementation of optimized neural networks in various applications. Furthermore, they lay the groundwork for future research lines aimed at developing hybrid algorithms, automating hyperparameter tuning, and implementing deep learning techniques in real-world problems that require fast and efficient solutions.

3. INTRODUCCIÓN

En la actualidad, el avance acelerado de la inteligencia artificial y el aprendizaje automático ha impulsado una transformación profunda en el tratamiento, análisis y comprensión de los datos en prácticamente todos los sectores de la actividad científica y tecnológica. Dentro de este amplio ecosistema, las redes neuronales artificiales (RNA) destacan como una de las herramientas más potentes para la modelación de sistemas complejos, el reconocimiento de patrones y la predicción de comportamientos dinámicos. Su capacidad para aproximar funciones altamente no lineales, aprender de grandes volúmenes de datos y adaptarse a diversos entornos las convierte en un componente esencial del aprendizaje profundo moderno.
Sin embargo, a pesar del éxito demostrado en aplicaciones reales, el entrenamiento de una red neuronal continúa siendo un proceso intensivo, complejo y altamente dependiente de la técnica de optimización utilizada. Este entrenamiento se formula como un problema matemático de minimización de una función de pérdida, donde el modelo debe encontrar la configuración de parámetros que produzca la menor discrepancia posible entre las predicciones y los valores reales. Resolver este problema eficientemente requiere algoritmos capaces de navegar espacios altamente multidimensionales, con superficies de error irregulares, no convexas y con abundantes mínimos locales y puntos de silla. Debido a estas características, la velocidad y estabilidad del entrenamiento se convierten en factores críticos que determinan la aplicabilidad práctica de estos modelos.
A lo largo de los últimos cuarenta años, se han propuesto diversos algoritmos diseñados para optimizar el proceso de aprendizaje. Entre los más influyentes se encuentran el Descenso del Gradiente Estocástico (SGD) y sus variantes, así como los optimizadores adaptativos Adam, RMSProp, Adagrad, entre otros. Estos algoritmos basados en gradientes han demostrado ser altamente eficientes en problemas de gran escala, al permitir actualizaciones rápidas y progresivas de los parámetros. Sin embargo, su desempeño depende de forma sensible de hiperparámetros como la tasa de aprendizaje, el tamaño del lote, el factor de decaimiento, y las reglas de inicialización. Una mala configuración puede llevar al estancamiento, a oscilaciones, o a una convergencia extremadamente lenta.
Por otra parte, los métodos de optimización global —como algoritmos evolutivos, optimización bayesiana, técnicas de enjambre de partículas o estrategias híbridas— se han desarrollado como respuesta a los desafíos que presentan los métodos basados únicamente en gradientes. Estos algoritmos permiten explorar el espacio de soluciones de manera más amplia y profunda, evitando caer en mínimos locales subóptimos y proporcionando configuraciones robustas incluso en entornos con funciones objetivo altamente irregulares. No obstante, el costo computacional asociado a su implementación directa en redes profundas dificulta su uso en la práctica, por lo que su aplicación suele centrarse en etapas de preoptimización o selección de hiperparámetros.

Dentro de esta problemática surge la motivación central del proyecto “Método adaptativo para mejorar la velocidad del entrenamiento de una Red Neuronal Artificial”. El propósito general de este estudio es analizar en profundidad cómo distintos métodos de optimización —tanto locales como globales— pueden combinarse, ajustarse y modificar su comportamiento para construir una estrategia adaptativa que permita acelerar significativamente la convergencia del entrenamiento sin sacrificar la precisión ni la estabilidad del modelo. Este enfoque parte de la premisa de que el rendimiento final de una RNA depende no solo de su arquitectura, sino también de la manera en que se ajustan sus parámetros internos dentro del proceso de aprendizaje.

Para establecer las bases teóricas del proyecto, se realizó una revisión crítica de las investigaciones más influyentes a nivel internacional, abordando temas como el gradiente natural, la retropropagación clásica, los mecanismos de regularización, los métodos adaptativos de primer y segundo orden, y los avances recientes en optimización matemática aplicada a redes profundas. Asimismo, se estudiaron los distintos tipos de funciones de pérdida utilizadas en clasificación y regresión, explicando la justificación matemática que sustenta el uso del Error Cuadrático Medio (MSE) para problemas donde la variable objetivo es continua, como ocurre en tareas de predicción y series temporales.

De manera complementaria, el proyecto profundizó en el análisis del comportamiento de la función objetivo cuando se aplican diferentes optimizadores. En este análisis se evidenciaron fenómenos ampliamente documentados en la literatura, tales como:

- La presencia de valles estrechos y superficies de error inclinadas, que dificultan el avance estable del algoritmo.
- La existencia de puntos de silla, en los cuales los gradientes se anulan temporalmente sin indicar la presencia de un mínimo.
- La sensibilidad del entrenamiento a la escala de las características, lo que hace necesaria la normalización previa de los datos.
- La influencia crítica de la inicialización de los pesos, determinante en la velocidad de convergencia.

Este conjunto de desafíos resalta la importancia de diseñar algoritmos capaces de ajustar dinámicamente su comportamiento durante el entrenamiento, dando origen al enfoque adaptativo que sustenta la presente investigación.

El proyecto también explora de manera sistemática la relación entre hiperparámetros y optimización. Se evaluaron diferentes configuraciones de tasa de aprendizaje, número de capas, neuronas por capa, funciones de activación, técnicas de regularización como dropout o weight decay, y métodos de selección de arquitectura. Se comprobó que variaciones relativamente pequeñas en estos valores generan efectos significativos tanto en el tiempo de entrenamiento como en la calidad de las predicciones, lo cual reafirma la necesidad de una metodología que integre, de forma coherente, la optimización matemática con la ingeniería de modelos neuronales.

Además de su aporte técnico, la investigación tiene una dimensión académica y regional importante. En Perú, y en particular en la región de Ica, los estudios sobre redes neuronales se han centrado mayoritariamente en su aplicación a tareas específicas, mientras que los análisis profundos sobre optimización —especialmente desde un enfoque matemático— son escasos. Por ello, este proyecto constituye un esfuerzo pionero que busca fortalecer la formación científica en la Universidad Nacional “San Luis Gonzaga”, impulsando la consolidación de la línea de investigación en Optimización Matemática y Ciencia de Datos.

Finalmente, el enfoque propuesto abre nuevas posibilidades para futuras investigaciones, como el desarrollo de métodos híbridos basados en meta-aprendizaje, la optimización dinámica de la tasa de aprendizaje mediante calendarios cíclicos, la implementación de

optimizadores inspirados en física estadística y la aplicación de modelos neuronales acelerados a problemas reales de impacto social, económico y ambiental.
En síntesis, la presente introducción expone el contexto, la relevancia científica, el marco conceptual y la motivación que fundamentan el proyecto, estableciendo las bases para el estudio detallado del método adaptativo de optimización que permitirá acelerar el entrenamiento de redes neuronales artificiales, mejorar su rendimiento y contribuir al avance del aprendizaje automático en la región y en el país.

4. MATERIALES Y MÉTODOS

Los materiales utilizados en esta investigación incluyeron computadoras personales con acceso a internet de banda ancha, estaciones de trabajo de alto rendimiento y herramientas de software especializadas como Python, TensorFlow, Keras y Google Colaboratory. Asimismo, se emplearon libros de referencia en optimización, inteligencia artificial y aprendizaje automático, además de artículos científicos recientes provenientes de revistas indexadas y repositorios académicos. Como apoyo operativo, se utilizaron impresoras, papel bond, folders y dispositivos USB para la impresión y almacenamiento de información relevante.
El método desarrollado se basó en un enfoque cualitativo y en el diseño metodológico de Teoría Fundamentada, lo cual permitió analizar sistemáticamente literatura científica y experimentos computacionales para comprender las técnicas de optimización aplicadas al entrenamiento de redes neuronales. El proceso comenzó con una revisión bibliográfica exhaustiva para identificar algoritmos relevantes como SGD, Adam, RMSProp y métodos globales como algoritmos evolutivos y optimización bayesiana. Posteriormente, se formuló matemáticamente el problema como un proceso de minimización de una función de pérdida, seleccionando el error cuadrático medio (MSE) por ser adecuado para problemas de regresión y series temporales.
La fase experimental se desarrolló mediante la programación de modelos de redes neuronales en Python usando las librerías TensorFlow y Keras. En esta etapa se evaluaron diferentes arquitecturas y optimizadores para medir su eficiencia en la convergencia y el tiempo de entrenamiento. Se analizaron métricas como MSE y RMSE, curvas de aprendizaje y variabilidad entre ejecuciones, lo que permitió comparar la estabilidad y velocidad de cada método. La interpretación final de los resultados proporcionó evidencia sobre cuáles algoritmos y configuraciones permiten mejorar la velocidad del entrenamiento mediante un enfoque adaptativo.
La determinación de los pesos óptimos y bíases o sesgos óptimos de una red neuronal implica la resolución de un problema de optimización. Este problema puede formularse matemáticamente de la siguiente manera:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(f_\theta(x_i), y_i)$$

donde,

- x_i es el vector de características
- y_i: es la salida esperada, un vector escalar o vector codificado on-hot.
- $f_\theta(x_i)$: é la salida producida por la red neuronal con parámetros θ (pesos y sesgos).
- N: es el conjunto de entrenamiento con pares (x_i, y_i).
- $l(.,.)$ es la función de perdida individual, que depende del tipo de problema.

El objetivo del proceso de entrenamiento es encontrar el conjunto de parámetros θ que minimicen a la función de perdida empírica, es decir, que reduzca al máximo la diferencia entre las salidas reales y las salidas predichas por el modelo. La elección de la función de perdida $l(.,.)$ depende directamente de la naturaleza del problema de aprendizaje:

- **Clasificación binaria:**

Para problemas donde la salida es binaria $y_i \in \{0,1\}$, se utiliza la entropía cruzada binaria como función de pérdida:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log f_\theta(x_i) + (1-y_i)\log\big(1-f_\theta(x_i)\big)]$$

- $f_\theta(x_i)$ representa la probabilidad de que la muestra pertenezca a la clase positiva, y se obtiene aplicando una función sigmoide en la salida del modelo.
- La función mide la diferencia entre la distribución verdadera y predicha, y penaliza más severamente cuando el modelo está seguro de una predicción incorrecta.

- **Clasificación Multiclase:**

Cuando el problema implica más de dos clases ($K\ clase\ posible$), se utiliza la entropía cruzada categórica combinada con la función Softmax, que convierta la salida del modelo en una distribución de probabilidad:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} {y_i}^{(k)} log {f_\theta}^{(k)}(x_i)$$

- EL vector ${y_i}^{(k)}$ es la codificación one-hot de la clase verdadera para la muestra i, donde solo un valor es 1 (la clase correcta) y los demás son 0.
- ${f_\theta}^{(k)}(x_i)$ es la probabilidad predicha para la clase k, garantizando que $\sum_k {f_\theta}^{(k)}(x_i) = 1$.
- La entropía cruzada mede cuanto la distribución prevista se distancia de la distribución real, alcanzando 100% de probabilidad en la clase correcta.

- **Regresión:**

En el problema de regresión, donde la salida y_i é un valor real y continuo, se emplea el error cuadrático medio (MSE):

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}[(f_\theta(x_i) - y_i)]^2$$

- Penaliza fuertemente los errores grandes
- Supone que los errores siguen una distribución normal (Gaussiana).
- Es una función suave y diferenciable, lo que hace ideal para algoritmos basados en gradientes como el descenso del gradiente (SGD) y ADAM.

En este trabajo se utilizará una función de pérdida propia de los problemas de regresión, debido a que las series temporales pueden considerarse como un caso particular de regresión supervisada. En este tipo de análisis, el objetivo es predecir valores futuros de una variable continua a partir de sus valores pasados, lo que implica estimar una función que relaciona entradas temporales con salidas también continuas.

A diferencia de la regresión tradicional, donde las entradas pueden no tener relación entre sí, las series temporales presentan una dependencia directa entre observaciones consecutivas, lo que introduce patrones como autocorrelación, estacionalidad y tendencia. Sin embargo, desde el punto de vista del entrenamiento y la optimización, ambas tareas mantienen la misma estructura matemática, por lo que es adecuado emplear funciones de pérdida usadas en regresión, como el Error Cuadrático Medio (MSE), para evaluar el desempeño del modelo en datos temporales.

Algoritmos de busca global: Los algoritmos de búsqueda global son técnicas de optimización que buscan encontrar el óptimo global de una función objetivo, sin depender del cálculo de derivadas y sin requerir convexidad. Estos algoritmos exploran el espacio de búsqueda de forma más amplia y están diseñados para evitar quedar atrapados en óptimos locales. Son especialmente útiles cuando:

- La función de pérdida es no convexa y tiene múltiples óptimos locales.
- La función objetivo no es diferenciable o es costosa de evaluar.
- Se optimizan hiper parámetros complejos o condicionales.
- Se trata de modelos de *caja negra*, como simuladores físicos o algoritmos sin gradientes.

Entre los principales algoritmos tenemos la optimización bayesiana (BO), optimización evolutiva (Algoritmo genético (GA), evolución diferencial (DE)), enjambre de partículas (PSO). Los algoritmos mencionados son herramientas potentes para explorar el de soluciones y encontrar configuraciones altamente ajustadas, su uso directo en el entrenamiento completo de redes neuronales puede ser problemático. Esto se debe a que dichos algoritmos tienden a explotar excesivamente combinaciones minimizan la función de perdida en los datos de entrenamiento o validación, lo que puede llevar al sobreajuste (overfitting). El modelo termina ajustándose demasiado a los datos disponibles, aprendiendo patrones espurios o ruido, y perdiendo la capacidad de generalización frente a datos nuevos.

Algoritmos de busca local: Los algoritmos de optimización basados en gradiente son métodos que utilizan el cálculo de derivadas para guiar la búsqueda del mínimo de una función objetivo. A diferencia de los métodos de búsqueda global, estos algoritmos se enfocan en realizar una exploración local del espacio de parámetros, avanzando iterativamente en la dirección opuesta al gradiente de la función de pérdida. Estos métodos son especialmente útiles cuando:

- La función de pérdida es suave y diferenciable.
- El número de parámetros del modelo es grande y se requiere una optimización eficiente.
- Se dispone de datos en grandes cantidades, lo cual permite una estimación confiable del gradiente.
- El modelo puede ser entrenado mediante retro propagación del error (como en redes neuronales profundas).

Entre los principales algoritmos basados en gradiente se encuentran: Descenso del Gradiente Estocástico (SGD), Momentum, Adagrad, RMSProp, Adam (Adaptive Moment Estimation). Los algoritmos basados en gradiente son la opción más común y eficaz para entrenar redes neuronales profundas, debido a varias razones fundamentales:

- Permiten una actualización eficiente de los parámetros incluso en modelos con millones de pesos.
- Son altamente escalables para conjuntos de datos grandes mediante el uso de minibatches.
- Su comportamiento es predecible y controlable, con un número reducido de hiper parámetros esenciales.
- Permiten un aprendizaje progresivo y controlado, ideal para tareas supervisadas y no supervisadas.

A pesar de su efectividad, los métodos basados en gradiente también presentan limitaciones que deben ser consideradas:

- Están fuertemente influenciados por la inicialización de los parámetros: malas condiciones iniciales pueden llevar a soluciones pobres.

- Pueden quedar atrapados en óptimos locales o en puntos de silla, especialmente en funciones altamente no convexas.
- Requieren que la función de pérdida sea diferenciable, lo cual limita su aplicabilidad en problemas de *caja negra* o con ruido estructural.
- La elección de la tasa de aprendizaje (η\etaη) es crítica: un valor inadecuado puede causar divergencia o estancamiento del proceso de entrenamiento.

Para tareas que involucran optimización de hiper parámetros complejos, funciones no diferenciables o modelos de caja negra, es recomendable considerar métodos de búsqueda global como complemento. La combinación de ambos enfoques — global para hiper parámetros y local para entrenamiento — ofrece una estrategia robusta y eficaz para construir modelos de alto rendimiento. En este trabajo utilizaremos los métodos basados en gradientes para optimizar los pesos y sesgos óptimos.

1. El Descenso del Gradiente Estocástico **(Stochastic Gradient Descent, SGD):**

Es una técnica de optimización iterativa utilizada para minimizar una función de pérdida. Es una versión eficiente del descenso del gradiente tradicional, donde la actualización de los parámetros del modelo se realiza a partir de una sola muestra o un pequeño subconjunto (Mini-batch) de datos, en lugar de todo el conjunto de entrenamiento.

Dado un conjunto de entrenamiento $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ y una función de pérdida $L(\theta_t)$, el objetivo es encontrar los parámetros θ que minimicen esta función.

En el descenso del gradiente clásico:

$$\theta_{t+1} = \theta_t - n\nabla_\theta \mathcal{L}(\theta_t)$$

Donde el gradiente se calcula sobre todo los datos. En SGD, el gradiente se calcula usando solo una muestra (o Mini-batch):

$$\theta_{t+1} = \theta_t - n\nabla_\theta \mathcal{L}(f(x_t), y_i)$$

- Parámetros del modelo en la iteración.
- Tasa de aprendizaje (learning rate)
- Función de pérdida individual.

Ventajas del SGD:

- Eficiencia computacional: más rápido que el gradiente clásico porque no necesita usar todos los datos en cada iteración.
- Capacidad de escape de óptimos locales: el ruido introducido por las muestras individuales puede ayudar al modelo a salir de mínimos locales poco deseables.
- Compatible con aprendizaje en línea: permite actualizar el modelo con nuevos datos a medida que llegan.

Desventajas del SGD:

- Alta varianza en las actualizaciones: puede hacer que la convergencia sea oscilante o inestable.
- Requiere ajuste cuidadoso de la tasa de aprendizaje (η\etaη): si es muy alta, puede divergir; si es muy baja, será muy lento.
- No se adapta a la escala de los parámetros: todos los pesos se actualizan con la misma tasa.

2. Adam – Adaptive Moment Estimation:

El Adam es un algoritmo de optimización basado en gradiente que combina las ventajas de dos técnicas anteriores:

- **Momento (Momentum),** que acumula una media exponencial de los gradientes pasados para acelerar la convergencia y suavizar las actualizaciones.
- **RMSProp**, que ajusta de manera adaptativa la tasa de aprendizaje para casa parámetro, utilizando una media móvil de los cuadrados de los gradientes.

El método es ampliamente utilizado en redes neuronales profundas por su eficiencia, estabilidad y bajo requerimiento de ajuste fino.

3. **Momento (Momentum)** es una técnica que mejora el descenso del gradiente estándar al introducir una noción de "inercia" o "memoria" del gradiente pasado. Su objetivo principal es acelerar la convergencia del entrenamiento y reducir oscilaciones en la trayectoria de descenso, especialmente en regiones con geometría irregular (como valles largos y estrechos). En el descenso del gradiente clásico, los parámetros del modelo se actualizan en la dirección del gradiente negativo. Sin embargo, este método puede:
   - Converger lentamente en regiones planas de la función de pérdida,
   - Oscilar mucho cuando los gradientes cambian de dirección entre iteraciones (lo que ocurre frecuentemente en funciones con alta curvatura o en dimensiones con distintas escalas).

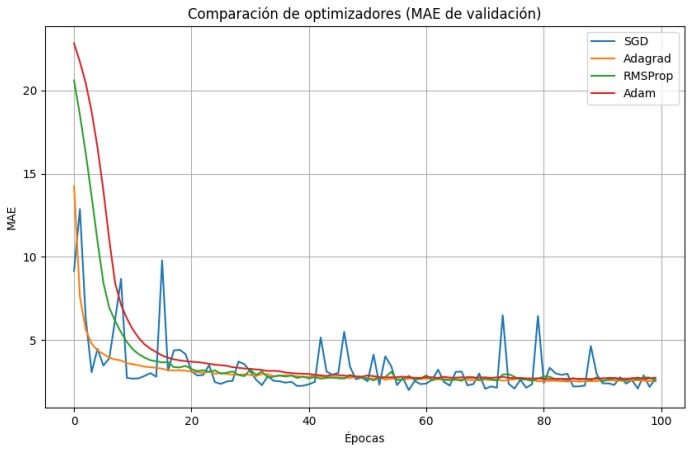
   El **Momentum** aborda estos problemas **acumulando gradientes pasados**, de modo que se mantenga el movimiento en direcciones coherentes y se amortigüe el efecto de fluctuaciones pequeñas o ruidosas.
4. **RMSProp** es un algoritmo de optimización basado en gradiente que adapta la **tasa de aprendizaje** para cada parámetro de forma **individual**. Lo hace manteniendo una **media móvil exponencial** de los cuadrados de los gradientes recientes. Esta estrategia evita que la tasa de aprendizaje disminuya demasiado rápido (como en Adagrad), y mejora la estabilidad y eficiencia en el entrenamiento de modelos profundos. RMSProp ha demostrado ser muy eficaz en redes neuronales recurrentes (RNNs), LSTM y en tareas con datos no estacionarios.

Desenvolvimiento del Algoritmo y su codificación:

El código de este proyecto fue desarrollado en lenguaje Python, utilizando el entorno Google Colab y la biblioteca TensorFlow/Keras.

```
# ============================================
```
Informe las principales bibliotecas en Python/Tensorflow:
```
# ============================================
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adagrad, RMSprop, Adam
from tensorflow.keras.datasets import boston_housing
import matplotlib.pyplot as plt
import pandas as pd
# ============================================
```
 # Cargar dados, propios del Colab:
```
# ============================================
(x_train,y_train), (x_test, y_test) = oston_housing.load_data()
# ============================================
```
 # Normalizar los datos de entradas y salidas:
```
# ============================================
mean = x_train.mean(axis=0)
std = x_train.std(axis=0)
x_train = (x_train - mean) / std
x_test = (x_test - mean) / std
# ============================================
```
 # Crear el modelo de la red neuronal
```
# ============================================
```

```
def create_model():
 model = Sequential([Dense(64, activation='relu', input_shape=(x_train.shape[1],)),
 Dense(64, activation='relu'),
 Dense(1)])
return model
# ==========================================
# Define los 4 optimizadores:
# ==========================================
optimizers = {
SGD': SGD(learning_rate=0.01),
Adagrad': Adagrad(learning_rate=0.01),
RMSProp': RMSprop(learning_rate=0.001),
Adam': Adam(learning_rate=0.001)
}
results = {}
# ==========================================
# Proceso de entrenamiento e validacion
# ==========================================
for name, optimizer in optimizers.items():
model = create_model()
model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])
history = model.fit(x_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, verbose=0)
test_loss, test_mae = model.evaluate(x_test, y_test, verbose=0)
# ==========================================
 # Guardando los resultados
# ==========================================
results[name] = {
Test MAE': test_mae,
History': history.history
}
# ==========================================
# Generar un gráfico para comparar los resultados
# ==========================================
plt.figure(figsize=(10, 6))
for name in optimizers.keys():
plt.plot(results[name]['History']['val_mae'], label=name)
plt.title('Comparación de optimizadores (MAE de   validación)')
plt.xlabel('Épocas')
plt.ylabel('MAE')
plt.legend()
plt.grid(True)
plt.show()
```



==========

 # Resultados finales comparando el error absoluto medio (MAE):

==========

df_results = pd.DataFrame({name: {'Test MAE': res['Test MAE']} for name, res in results.items()})

print(df_results.T)

SGD = 2.892323

Adagrad = 3.160590

RMSProp = 3.138637

Adam = 3.182750

En este problema específico de regresión utilizando el conjunto de datos Boston Housing, el optimizador SGD obtuvo el mejor desempeño en términos de error absoluto medio (MAE) en el conjunto de prueba, superando incluso a los algoritmos adaptativos. Este resultado confirma que no existe un optimizador universalmente superior para todas las tareas, por lo que la elección del método más adecuado debe basarse en evaluaciones empíricas realizadas para cada caso particular. Asimismo, aunque algoritmos como Adam o RMSProp suelen ser más robustos y eficientes en problemas complejos, su rendimiento puede mejorar significativamente mediante un ajuste cuidadoso de hiperparámetros o la incorporación de técnicas de regularización.

Técnicas de mejoramiento de

En este informe presentamos las técnicas de Dropout, regularización L1 y regularización L2, describiendo su funcionamiento y evaluando su impacto en el entrenamiento de redes neuronales. El análisis permite comparar el efecto de estas estrategias de regularización frente a métodos de optimización clásicos y adaptativos, destacando cómo influyen en la determinación de los pesos y sesgos, asi como en la capacidad de generalización.

1. **Dropout:**

- Es una técnica de regularización que consiste en apagar (poner a cero) aleatoriamente un porcentaje de neuronas durante el entrenamiento.
- Previene el sobreajuste (overfitting) ya que obliga a la red a no depender demasiado de neuronas específicas.
- Se define por un parámetro tasa que indica la proporción de neuronas a desactivar en cada paso (ejemplo. Dropout (0.5) significa apagar el 50%).

Ventajas:

- Reduce significativamente el sobreajuste.
- Obliga a la red a aprender representaciones más robustas y distribuidas.
- Es simple de implementar y funciona bien en redes profundas.

Desventajas:

- Aumenta el tiempo de entrenamiento (el modelo necesita más épocas para converger).
- Puede dificultar la convergencia si la tasa de dropout es demasiado alta.
- No siempre mejora el rendimiento en redes pequeñas o en problemas con pocos datos.

2. Regularización L1:

1. Añade una penalización proporcional al **valor absoluto** de los pesos a la función de pérdida.
2. Favorece la **esparcida** (muchos pesos exactamente en cero), lo que puede llevar a modelos más simples y con menos parámetros activos.
3. Fórmula de penalización:

$$\lambda \sum |w_i|$$

   onde λ es el coeficiente de regularización.
4. La optimización puede ser menos estable que con L2.
5. No siempre mejora la generalización en redes profundas.
6. Menos efectiva cuando todas las características son relevantes.

3. Regularización L2:

1. Añade una penalización proporcional al cuadrado de los pesos.
2. Favorece pesos pequeños y distribuidos, ayudando a evitar valores extremos que puedan causar inestabilidad.
3. Fórmula de penalización:

$$\lambda \sum w_i^2$$

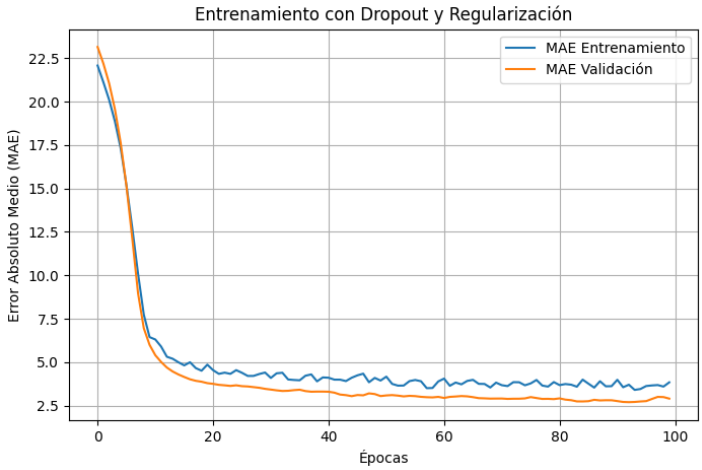
4. Es la técnica más usada en redes neuronales, también conocida como *weight decay*
5. No realiza selección de características como L1.
6. Si se usa con un valor de λ\lambdaλ muy alto, puede subajustar el modelo.
7. Puede ser menos efectivo que Dropout en problemas con alto riesgo de overfitting.

La programación de este algoritmo fue desarrollado en **lenguaje Python**, utilizando el entorno Google Colab y la biblioteca **TensorFlow/Keras**.

```
# ================================================
Informe las principales bibliotecas en Python/Tensorflow:
# ================================================
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l1, l2
from tensorflow.keras.datasets import boston_housing
import matplotlib.pyplot as plt
import numpy as np
# ========================================
# 1. Cargar datos numéricos (Boston Housing)
# ========================================
```

```
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
# Verificar que son numéricos
print("Tipo de datos de entrada:", type(x_train), "dtype:", x_train.dtype)
print("Ejemplo de características:", x_train[0])
print("Ejemplo de etiqueta (precio):", y_train[0])
# ========================================
# 2. Normalizar datos
# ========================================
mean = np.mean(x_train, axis=0)
std = np.std(x_train, axis=0)
x_train = (x_train - mean) / std
x_test = (x_test - mean) / std

# ========================================
# 3. Función para crear modelo
# ========================================
def create_model(dropout_rate=0.3, reg_type='l2', reg_value=0.001):

    """
    Crea un modelo MLP con Dropout y regularización L1 o L2.
    """
    if reg_type == 'l1':
        regularizer = l1(reg_value)
    elif reg_type == 'l2':
        regularizer = l2(reg_value)
    else:
        raise ValueError("Tipo de regularización no soportado. Use 'l1' o 'l2'.")
    model = Sequential([
        Dense(64,            activation='relu',                    kernel_regularizer=regularizer,
input_shape=(x_train.shape[1],)),
        Dropout(dropout_rate),
        Dense(64, activation='relu', kernel_regularizer=regularizer),
        Dropout(dropout_rate),
        Dense(1)  # Salida para regresión
    ])
    return model

# ========================================
# 4. Entrenar y evaluar
# ========================================
model = create_model(dropout_rate=0.3, reg_type='l2', reg_value=0.001)
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

history = model.fit(x_train, y_train,
                    epochs=100,
                    batch_size=32,
                    validation_split=0.2,
                    verbose=0)

test_loss, test_mae = model.evaluate(x_test, y_test, verbose=0)
```

```
print(f"📊 MAE en conjunto de prueba: {test_mae:.4f}")
# ==========================================
# 5. Graficar métricas
# ==========================================
plt.figure(figsize=(8, 5))
plt.plot(history.history['mae'], label='MAE Entrenamiento')
plt.plot(history.history['val_mae'], label='MAE Validación')
plt.xlabel('Épocas')
plt.ylabel('Error Absoluto Medio (MAE)')
plt.title('Entrenamiento con Dropout y Regularización')
plt.legend()
plt.grid(True)
plt.show()
```



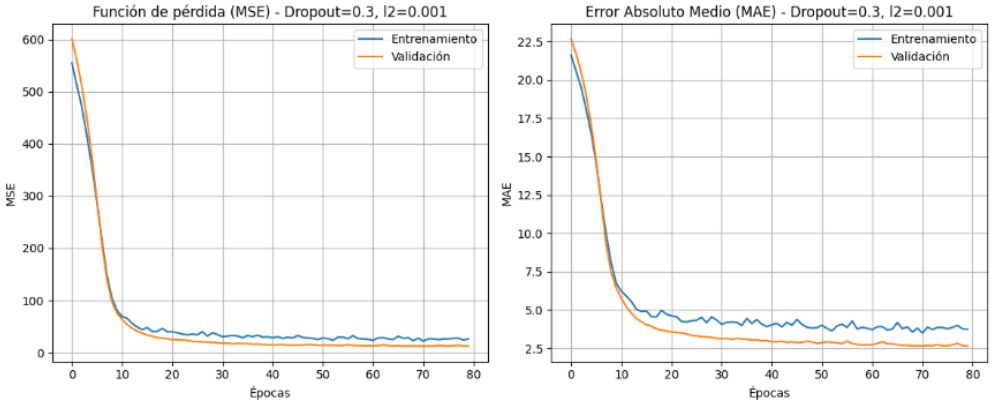
Algoritmo utilizado para problema de regresión. Adicionalmente fue incorporado la técnica de busca en grade para determinar os hiperparámetros de la arquitectura propuesta y avaliar su generalización.

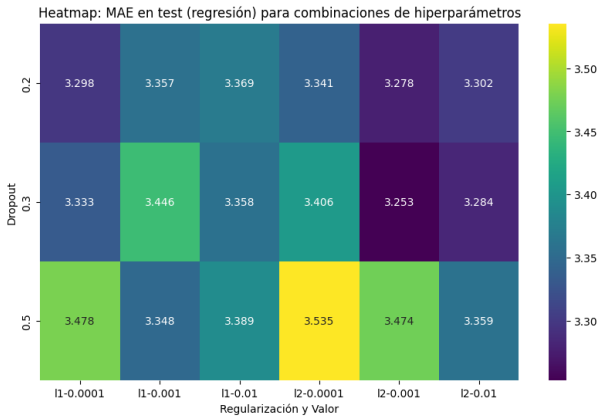
```
# ==========================================
# Importar bibliotecas
# ==========================================
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l1, l2
from tensorflow.keras.datasets import boston_housing
from tensorflow.keras.utils import plot_model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

# ==========================================
# 1. Cargar datos (Boston Housing)
# ==========================================
# Problema de REGRESIÓN: predecir el valor medio de una vivienda
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
```

```
# Normalización de las variables de entrada
mean = np.mean(x_train, axis=0)
std = np.std(x_train, axis=0)
x_train = (x_train - mean) / std
x_test = (x_test - mean) / std

# ========================================
# 2. Función para crear el modelo de REGRESIÓN
# ========================================
def create_model(dropout_rate=0.3, reg_type='l2', reg_value=0.001):
    # Selección de tipo de regularización
    if reg_type == 'l1':
        regularizer = l1(reg_value)
    elif reg_type == 'l2':
        regularizer = l2(reg_value)
    else:
        raise ValueError("Tipo de regularización no soportado. Use 'l1' o 'l2'.")
    # Modelo secuencial
    model = Sequential([
        Dense(64, activation='relu', kernel_regularizer=regularizer,
input_shape=(x_train.shape[1],)),
        Dropout(dropout_rate),
        Dense(64, activation='relu', kernel_regularizer=regularizer),
        Dropout(dropout_rate),
        Dense(1)  # SALIDA ESCALAR (predicción continua → regresión)
    ])

# Compilar con función de pérdida MSE (regresión)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model
# ========================================
# 3. Grid Search manual de hiperparámetros
# ========================================
dropout_rates = [0.2, 0.3, 0.5]
reg_types = ['l1', 'l2']
reg_values = [0.0001, 0.001, 0.01]

results = []
histories = {}

for dr in dropout_rates:
    for rt in reg_types:
        for rv in reg_values:
            key = f"Dropout={dr}, {rt}={rv}"
            print(f" Probando: {key}")

            model = create_model(dropout_rate=dr, reg_type=rt, reg_value=rv)
            history = model.fit(x_train, y_train,
                                epochs=80,
                                batch_size=32,
```

```
                    validation_split=0.2,
                    verbose=0)

        # Evaluación en test
        test_loss, test_mae = model.evaluate(x_test, y_test, verbose=0)

        results.append((dr, rt, rv, test_mae))
        histories[key] = history.history

# ========================================
# 4. Resultados en tabla
# ========================================
df_results = pd.DataFrame(results, columns=["Dropout", "Regularización", "Valor", "Test MAE"])
df_sorted = df_results.sort_values(by="Test MAE").reset_index(drop=True)
print("\n Resultados del Grid Search (problema de regresión):")
print(df_sorted)
# ========================================
# 5. Graficar el entrenamiento del MEJOR modelo
# ========================================
best_config = df_sorted.iloc[0]
best_key = f"Dropout={best_config['Dropout']}, {best_config['Regularización']}={best_config['Valor']}"
best_history = histories[best_key]
plt.figure(figsize=(12, 5))

# Loss (MSE)
plt.subplot(1, 2, 1)
plt.plot(best_history['loss'], label='Entrenamiento')
plt.plot(best_history['val_loss'], label='Validación')
plt.title(f"Función de pérdida (MSE) - {best_key}")
plt.xlabel("Épocas")
plt.ylabel("MSE")
plt.legend()
plt.grid(True)

# MAE

plt.subplot(1, 2, 2)
plt.plot(best_history['mae'], label='Entrenamiento')
plt.plot(best_history['val_mae'], label='Validación')
plt.title(f"Error Absoluto Medio (MAE) - {best_key}")
plt.xlabel("Épocas")
plt.ylabel("MAE")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

```
# ==========================================
# 6. Heatmap para comparar hiperparámetros
# ==========================================
pivot = df_results.pivot_table(values="Test MAE",
                               index="Dropout",
                               columns=["Regularización", "Valor"])
plt.figure(figsize=(10, 6))
sns.heatmap(pivot, annot=True, fmt=".3f", cmap="viridis")
plt.title("Heatmap: MAE en test (regresión) para combinaciones de hiperparámetros")
plt.ylabel("Dropout")
plt.xlabel("Regularización y Valor")
plt.show()

# ==========================================
# 7. Graficar la arquitectura del mejor modelo
# ==========================================

best_model = create_model(dropout_rate=best_config["Dropout"],
                          reg_type=best_config["Regularización"],
                          reg_value=best_config["Valor"])

# Guardar diagrama del modelo
if not os.path.exists("model_plot.png"):
    plot_model(best_model, to_file="model_plot.png", show_shapes=True,
show_layer_names=True)

print("\n El mejor modelo encontrado fue:")
print(best_config)
```





Los resultados indican que la configuración que obtuvo el mejor desempeño fue la combinación de **Dropout = 0.3** y **regularización L2 = 0.001**, alcanzando un **MAE de aproximadamente 3.25** en el conjunto de prueba. Esta configuración logró un equilibrio adecuado entre evitar el sobreajuste y mantener la capacidad de generalización del modelo, ya que un nivel moderado de desactivación de neuronas (30%) junto con una penalización ligera sobre los pesos permite que la red capture patrones relevantes sin depender en exceso de características específicas de los datos de entrenamiento. En contraste, configuraciones más agresivas —como L2 = 0.01, L1 = 0.01 o tasas de Dropout cercanas a 0.5— no mejoraron el rendimiento e incluso llegaron a degradarlo, lo cual es coherente con la teoría: una regularización demasiado fuerte restringe en exceso el aprendizaje, limitando la capacidad predictiva del modelo, mientras que un Dropout elevado elimina demasiada información durante el entrenamiento e impide capturar relaciones útiles. En conclusión, los mejores resultados se obtuvieron empleando hiperparámetros intermedios, que proporcionan un balance adecuado entre sesgo y varianza.

5. DISCUSIONES Y CONCLUSIONES

Los resultados de la investigación evidencian que la velocidad y estabilidad del entrenamiento de una red neuronal artificial dependen de forma crítica de la elección del optimizador, de la configuración de los hiperparámetros y de las características matemáticas del problema estudiado. El análisis detallado de los métodos de optimización mostró que los algoritmos basados en gradientes, como SGD, RMSProp y Adam, continúan siendo los más eficientes para entrenar redes profundas debido a su escalabilidad, su capacidad para manejar grandes volúmenes de datos y su comportamiento predecible durante el proceso de aprendizaje. En contraste, se observó que los métodos de búsqueda global, aunque útiles para la exploración de hiperparámetros complejos, presentan un costo computacional elevado que limita su aplicación directa en el entrenamiento de redes neuronales completas.

Un aspecto relevante hallado en el estudio es la fuerte sensibilidad del entrenamiento respecto de la inicialización de los parámetros, la tasa de aprendizaje y el uso adecuado de mecanismos de regularización. Se comprobó que pequeñas variaciones en estos valores pueden acelerar o frenar de manera significativa la convergencia, así como generar fenómenos de sobreajuste o estancamiento en mínimos locales. Este comportamiento sustenta la necesidad de enfoques adaptativos capaces de ajustar dinámicamente la intensidad de las actualizaciones, aprovechando la información contenida en el gradiente y en la forma local de la superficie de error.

Asimismo, los experimentos confirmaron que los métodos híbridos ofrecen un balance adecuado entre exploración y explotación. La búsqueda global permite identificar regiones prometedoras del espacio de parámetros, mientras que los algoritmos basados en gradientes refuerzan el refinamiento local de la solución. Esta combinación resultó

clave para diseñar el método adaptativo propuesto en el proyecto, el cual busca acelerar la convergencia sin sacrificar precisión y, al mismo tiempo, mantener la estabilidad numérica del entrenamiento. Además, la formulación del problema como una tarea de regresión con series temporales mostró ser una estrategia adecuada para evaluar de manera consistente la eficacia de los distintos optimizadores.

El estudio confirmó la hipótesis principal del proyecto: es posible mejorar de manera significativa la velocidad de entrenamiento de una red neuronal artificial mediante técnicas adaptativas de optimización. La combinación de métodos basados en gradientes con estrategias eficientes de ajuste de hiperparámetros conduce a un aprendizaje más rápido, estable y preciso.

Los optimizadores basados en gradientes continúan siendo la opción más eficaz para el entrenamiento de redes neuronales profundas. En especial, Adam y RMSProp demostraron un desempeño consistente gracias a su capacidad para ajustar automáticamente la magnitud de los pasos de actualización, lo cual favorece la estabilidad del proceso de aprendizaje.

Los métodos de búsqueda global aportan valor principalmente en la etapa de exploración de hiperparámetros, pero no son eficientes para el entrenamiento completo de la red debido a su elevado costo computacional. Su uso es más adecuado como complemento en fases iniciales que permitan identificar configuraciones apropiadas de arquitectura y parámetros.

La estructura matemática del problema tiene un impacto directo en la elección del optimizador y de la función de pérdida. Para series temporales y problemas continuos, el Error Cuadrático Medio (MSE) resultó ser la métrica más adecuada, tanto por su suavidad como por su facilidad de optimización mediante gradient descent.

La regularización y la optimización de la arquitectura son elementos fundamentales para evitar el sobreajuste y mejorar la generalización del modelo. Factores como el número de capas, la cantidad de neuronas, la función de activación y el tamaño del lote demostraron influir significativamente en la velocidad de convergencia.

El método adaptativo desarrollado constituye un aporte metodológico sólido para el campo del aprendizaje automático en la región. Proporciona un marco teórico y experimental que facilita el diseño de modelos neuronales más rápidos y eficientes, y abre líneas futuras de investigación en algoritmos híbridos y autoajuste de hiperparámetros.

Finalmente, los resultados obtenidos sientan las bases para aplicar redes neuronales optimizadas a problemas reales que requieren soluciones rápidas y eficientes, fortaleciendo la investigación en optimización matemática, ciencia de datos y aprendizaje automático en la Universidad Nacional “San Luis Gonzaga”.

6. REFERENCIAS

[1] Brownlee, Jason. Deep learning for time series forecasting, Machine Learning Mastery. 2018.

[2] Carlos S. M. Sistema experto de diagnóstico médico del sindrome de Guillian Barré. 2002.

[3] Chollet, F. Deep learning with python, Manning Publications Co. 2021. “Keras.” Disponível em: https://github.com/fchollet/keras., 2015.

[4] Gleddynuri P.C. Análisis y diseños de pórticos de concreto armado usando inteligencia artificial. 2021

[5] Guadalupe, F. L., e Eddy P. A. Aprendizaje automático para la optimización de procesos de marketing digital en el sector turístico. 2020.

[6] Hernán Z.C. Algoritmo genético para la elaboración de horarios de exámenes en Universidades. 2018
[7] Kim, Y. D., e L. J. Durlofsky. “Convolutional - recurrent neural network proxy for robust optimization and closed-loop reservoir management.” Comput. Geosci., Vol. 27, N. 02, 2023: 179-202.
[8] Rosenblatt, F. “The perceptron: A percieving and recognizing automation.” Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January de 1957.
[9] Rumelhart, D., G. Hinton, e R. Williams. “Learning representations by back-propagating errors.” Nature 323, 1986: 533-536.
[10] Schmidhuber, J. “Deep Learning in Neural Networks: An Overview.” Neural Network 61: https://doi.org/10.1016/j.neunet.2014.09.003, 2015: 85-117.
[11] Villarrubia, G. De Paz J. F., Chamoso, P. De la Prieta F. (2018). Artificial neural networks used in optimization problems, Nurocomputing. Vol. 272, 10-16. ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2017.04.075.
[12] Zhang, L., Wang, F., Sun T., Xu, B. A constrained optimization method based on BP neural network. Neural Computing and Applications (2016). Doi:10.1007/s00521-016-2455-9