ALGORITMO DE APRENDIZAJE PARA UNA RED NEURONAL PROFUNDA BASADO EN UNA META-HEURÍSTICA DE OPTIMIZACIÓN GLOBAL DE GRAN ESCALA

Universidad
del Cauca

JULIÁN FERNANDO MUÑOZ ORDÓÑEZ

Tesis de Maestría en Computación

Director: PhD. Carlos Alberto Cobos Lozada

Codirector: PhD. Martha Eliana Mendoza Becerra

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Maestría en Computación

Grupo de I+D en Tecnologías de la Información (GTI)

Área de Investigación en Sistemas Inteligentes

Popayán, 22 de Noviembre de 2019

JULIÁN FERNANDO MUÑOZ ORDÓÑEZ

ALGORITMO DE APRENDIZAJE PARA UNA RED NEURONAL PROFUNDA BASADO EN UNA META-HEURÍSTICA DE OPTIMIZACIÓN GLOBAL DE GRAN ESCALA

Tesis presentada a la Facultad de Ingeniería
Electrónica y Telecomunicaciones de la
Universidad del Cauca para la obtención del
Título de

Magíster en
Computación

Directores
Director: PhD. Carlos Alberto Cobos Lozada
Codirectora: PhD. Martha Eliana Mendoza Becerra

Popayán
2019

Dedicatoria

Quiero dedicar este trabajo a:

Mis padres Alba Enelia y Gerardo Alirio, cada logro que alcanzo es completamente suyo, gracias por inculcarme a través de su ejemplo valores claves en esta etapa investigativa: responsabilidad, honestidad y esfuerzo constante por superarse, son mis principales maestros y ejemplos a seguir. Los amo.

Mi hermano Cristian Camilo, siente también como tuyo este logro, me enseñas día a día como la disciplina es fundamental para superarse y como el esfuerzo constante produce excelentes frutos.

Mi amada Ana María, has sido desde hace mucho tiempo el apoyo más grande que he podido tener, progresamos siempre juntos y cada día me ayudas a ser la mejor versión de mí. Soy muy afortunado de tenerte a mi lado. Te amo.

Agradecimientos

Mis agradecimientos a:

La Universidad del Cauca, por brindar el espacio, los recursos tecnológicos y humanos para la realización exitosa del proyecto de investigación.

La Vicerrectoría de Investigaciones de la Universidad del Cauca, por la ayuda brindada a través de la convocatoria de investigación de apoyo a maestrías y doctorados.

Profesores de la Maestría en Computación, la culminación de esta investigación tiene en cierto modo un aporte de todos ellos debido a las enseñanzas que me han compartido a lo largo de la maestría.

Un profundo agradecimiento a:

Mis padres Gerardo y Alba, mi hermano Cristian y mi novia Ana María, por estar siempre y en cada momento apoyándome durante toda la maestría.

Mis directores el Profesor Carlos y la Profesora Martha, personas íntegras, responsables, honestas, serviciales, trabajadoras y ejemplo de verdaderos maestros. La pasión que demuestran por la investigación y su trabajo constante por superarse produce que sus pupilos siempre queramos seguir sus pasos y enseñanzas, su acompañamiento constante durante toda la maestría me proporcionó confianza en mi formación investigativa y siempre deseos de superación. Mi mayor respeto a Ustedes, Maestros.

Resumen Estructurado

El aprendizaje profundo se basa en el incremento del número de capas ocultas en las arquitecturas de redes neuronales, este incremento proporciona a la red una mayor capacidad de abstracción sobre el conjunto de datos que se entrena, permitiendo mejorar considerablemente las tasas de precisión y error con respecto a los algoritmos tradicionales de aprendizaje máquina. Sin embargo, incrementar el número de capas ocultas conlleva a que los procesos de entrenamiento sufran estancamientos debido a diferentes problemas en la alta dimensionalidad: puntos de silla, múltiples óptimos locales y puntos de platea. El algoritmo por excelencia para el entrenamiento de redes neuronales profundas es conocido como RMSProp, este algoritmo funciona con información brindada por el gradiente descendente estocástico, sin embargo, la Neuroevolución: técnicas relacionadas con algoritmos evolutivos para entrenamiento de redes neuronales, han brindado nuevas posibilidades para resolver estos tipos de problemas. El presente trabajo de investigación se enfoca en definir cuál de tres (3) de los mejores algoritmos metaheurísticos mono-objetivo especializados en resolver problemas de alta dimensionalidad: Coevolución cooperativa basada en evolución diferencial (DECC-G), Muestreo múltiple de descendientes (MOS) e Hibridación iterativa de evolución diferencial (DE) con búsqueda local con reinicio (IHDELS) consiguen mejorar los resultados del error cuadrático medio logrados por el RMSProp y un simple algoritmo evolutivo de evaluaciones limitadas (LEEA) sobre conjuntos de datos de regresión y clasificación. Los resultados muestran que las metaheurísticas logran ser competitivas frente al RMSProp y logran vencer al algoritmo LEEA en conjuntos de datos de regresión. Por otra parte, en esta investigación se plantea un nuevo reto en la optimización de redes neuronales convolucionales profundas sobre conjuntos de datos de imágenes con un grado mayor de complejidad, durante esta etapa los resultados que se obtienen consisten en la construcción de una nueva propuesta memética inmersa en un nuevo framework de Neuroevolución (DNF) que logra vencer al RMSProp optimizando arquitecturas de redes neuronales convolucionales. Generando así un nuevo algoritmo y conocimiento en el área de la Neuroevolución aplicada sobre el aprendizaje profundo.

Palabras claves: Aprendizaje profundo, algoritmos evolutivos, redes neuronales artificiales, redes neuronales convolucionales, Metaheurísticas, Algoritmos de Optimización Global de Gran Escala, Algoritmos Meméticos, RMSProp, Gradiente Descendente Estocástico.

Structured Abstract

Deep learning is based on increasing the number of hidden layers in neural network architectures. This increase gives the network a greater capacity for abstraction on the dataset being trained, thereby enabling a considerable improvement in accuracy and error rates compared to traditional machine learning algorithms. Increasing the number of hidden layers, however, means that the training processes suffer stagnation due to different problems in high dimensionality: saddle points, multiple local optimal points and plateau. The algorithm *par excellence* for the training of deep neural networks is known as RMSProp, although this algorithm depends on information provided by stochastic gradient descent, new possibilities for solving problems of high dimensionality have arrived in the form of Neuroevolution techniques, related to evolutionary algorithms for training neural networks. This research focuses on defining which of three (3) single-objective metaheuristic algorithms best equipped to solve problems of high dimensionality are able to improve on mean square error results achieved by RMSProp and a simple evolutionary algorithm of limited evaluations (LEEA) on regression and classification data sets. The three under study are Differential Evolution-based cooperative co-evolution (DECC-G); Multiple offspring sampling (MOS) and Iterative hybridization of DE with local search, with restart (IHDELS). The results show that the metaheuristics manage to be competitive against RMSProp and manage to beat the LEEA algorithm in regression data sets. This study meanwhile poses a new challenge in the optimization of deep convolutional neural networks on image data sets with a greater degree of complexity. During this stage the results obtained consist of the construction of a new memetic proposal immersed in a new Neuroevolution framework (DNF). This managed to outperform RMSProp by optimizing convolutional neural network architectures, thus generating a new algorithm and knowledge in the area of Neuroevolution applied to deep learning.

Keywords: Deep Learning, Evolutionary Algorithms, Artificial Neural Networks, Convolutional Neural Networks, Metaheuristics, Large Scale Global Optimization, Memetic Algorithms, RMSProp, Stochastic Gradient Descent.

Tabla de Contenido

Lista de figuras

Lista de tablas

Capítulo 1

1 Introducción

1.1 Planteamiento del problema

Las redes neuronales artificiales (RNA) han adquirido nuevamente protagonismo en el área del aprendizaje de máquina tanto en procesos de clasificación como en regresión debido al surgimiento del aprendizaje profundo o Deep Learning [1][2][3][4]. La cualidad de “profundo” en las RNA se adquiere por el incremento del número de capas ocultas en su arquitectura, lo que les permite codificar características cada vez más complejas [5]. Las arquitecturas del aprendizaje profundo están basadas en una serie de enfoques [6][2] que permiten solucionar problemas relacionados con clasificación de imágenes (Deep Convolutional Neural Networks), análisis y reconocimiento de texto y voz (Deep Recurrent Neural Networks), entre otras.

Históricamente las redes neuronales artificiales han sido entrenadas con el algoritmo backpropagation (BP), el cual hace uso del gradiente descendente estocástico (Stochastic Gradient Descent, SGD) para ajustar los pesos de las neuronas en el proceso de entrenamiento [5], logrando disminuir la tasa total de error en el proceso de aprendizaje.

En una red neuronal profunda, durante el entrenamiento con BP, el SGD queda atrapado en óptimos locales, pero Dauphin et al. [7] demostraron que en la alta dimensionalidad existen múltiples rutas de escape de dichos óptimos locales, y que la disminución en la caída del SGD no se debe por si mismo a la alta dimensionalidad, sino más bien a la dificultad de salir de los puntos de silla (Saddle point), zonas donde el gradiente se hace cero. De esta forma identificaron el principal obstáculo en la optimización del aprendizaje de una red neuronal profunda. Es así como a la fecha se han propuesto diferentes algoritmos de aprendizaje que presentan buenos resultados en el entrenamiento de una red neuronal profunda, entre ellos RMSProp (Root Mean Squared PROPagation) [8] que se encarga de evitar que el SGD quede atrapado en los puntos de silla.

A partir de las ventajas que tiene RMSProp para escapar de los puntos de silla y óptimos locales, Morse et al. [5] plantearon la posibilidad de que la computación evolutiva, específicamente los algoritmos evolutivos (Evolutionary Algorithm, EA) cuentan con las mismas posibilidades de éxito e incluso una ventaja sobre los algoritmos basados en SGD, debido a que en los EAs se tiene un conjunto de múltiples individuos que pueden evolucionar para encontrar mejores caminos hacia la optimización de la red neuronal. A través de la implementación de una arquitectura conformada por 2 capas ocultas con 50 y 20 nodos, respectivamente, Morse et al. [5] logran ser competitivos frente al algoritmo RMSProp en el análisis de 3 problemas, a saber: realizar la mejor aproximación de una función [5], predecir valores en una serie temporal [9] y estimar valores del conjunto de datos California housing [9][10]. Sus estudios revelan que mediante una modificación de un EA sencillo y teniendo un poder computacional óptimo, el algoritmo logra equiparar los resultados e incluso ganar en el análisis de series temporales. Además, sus estudios

dejan abierta la posibilidad de encontrar mejores algoritmos evolutivos, que pueden acelerar el proceso de entrenamiento y alcanzar menores tasas de error en las redes neuronales profundas, puesto que debido al tipo de arquitectura que fue implementada y teniendo en cuenta la diferencia en el número de entradas a la red que representa cada conjunto de datos, las conexiones que se crearon para dichos conjuntos de prueba fueron de 1170 (función de aproximación), 1270 (serie de tiempo) y 1470 (California housing), lo que lo convierte en un problema de optimización global a gran escala [11].

Teniendo en cuenta que el tipo de problema que se está abordando (optimización del aprendizaje de redes neuronales profundas), es multimodal [12], de alta dimensionalidad (o de gran escala) y que los algoritmos DECC-G, CC-CMA-ES, IHDELS y MOS [13], han demostrado en la Competencia de Optimización Global a Gran Escala (Competition on Large Scale Global Optimization) del Congreso de la IEEE en Computación Evolutiva (IEEE Congress on Evolutionary Computation, IEEE CEC) del año 2015, ser las meta-heurísticas del estado del arte en la solución de estos tipos de problemas, en el presente proyecto se planteó la siguiente pregunta de investigación:

¿Cuáles de las meta-heurísticas DECC-G, CC-CMA-ES, IHDELS y MOS pueden obtener resultados similares o mejores en el proceso de aprendizaje (medido en menor tasa de error o menor tiempo de entrenamiento) de una red neuronal profunda implementada en [5] en comparación con los algoritmos del estado del arte como RMSProp?

1.2 Aportes del proyecto

Los aportes de este proyecto se consolidan en tres aspectos fundamentales: investigación, innovación e impacto.

Desde el punto de vista de la investigación, previamente Morse et al.[5] obtuvieron resultados promisorios y competitivos, usando un algoritmo evolutivo sencillo contra el método tradicional del SGD para la optimización del aprendizaje de una arquitectura de red neuronal profunda clásica. En el presente trabajo se obtuvo un algoritmo que soluciona de una mejor manera los problemas generados por la alta dimensionalidad del problema en el entrenamiento de dicha red [52], entre ellos el manejo de muchos óptimos locales, la no separabilidad de las variables y los puntos de silla en el espacio de soluciones. Con este trabajo se aporta nuevo conocimiento en el proceso de entrenamiento de redes neuronales profundas clásicas y se muestra que las meta-heurísticas pueden ser competitivas frente a los algoritmos tradicionales basados en la optimización del SGD. La propuesta supera el trabajo previo de Morse et al., obteniendo menores tasas de error [5] en la evaluación en tres tipos de problemas (serie temporal, aproximación de una función y dataset de California housing) con una red neuronal densa de 2 capas ocultas.

Además, con este trabajo se muestra que el trabajo previo de Morse et al. y el propio (descrito en el párrafo anterior) no permite obtener mejores resultados que RMSPROP en la optimización de redes convolucionales profundas, lo que también es nuevo conocimiento. Con el objetivo de realizar un aporte en este tipo de redes, se planteó una nueva propuesta que incluye en RMSPROP un esquema de exploración que se activa con base en el rendimiento histórico del algoritmo. Los resultados de esta nueva propuesta son competitivos y prometedores, siendo un camino que se debe seguir explorando en el campo de neuro evolución sobre redes Deep Learning.

Basado en los resultados positivos del entrenamiento de redes Deep Learning en dos datasets reconocidos por la comunicada académica y científica, los algoritmos desarrollados se integraron en una herramienta denominada Deep Neuroevolution

Framework (DNF), desarrollada en la presente tesis, que permite construir redes neuronales profundas aplicables en problemas de clasificación y/o regresión en el área del aprendizaje de máquina utilizando Tensorflow con Python, lo que se convierte en un aporte de innovación, ya que basado en el conocimiento del autor, es la primera en su clase.

En relación con el impacto que trae consigo el nuevo método de optimización del aprendizaje de redes neuronales profundas basado en metaheurísticas, se generó nuevas posibilidades de éxito en el incremento de la velocidad de las tasas de aprendizaje y la disminución del error en el entrenamiento de redes neuronales profundas, lo anterior debido a que los algoritmos evolutivos implementados conforman el estado del arte para problemas de optimización global a gran escala. Esto permitió entrenar redes más complejas para resolver problemas más complejos o que tienen volúmenes más grandes de datos de entrenamiento.

1.3 Objetivos

A continuación, se presentan los objetivos tal y como fueron aprobados por el Consejo de Facultad de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca en el anteproyecto.

1.3.1 Objetivo General

Definir un algoritmo de aprendizaje para una red neuronal profunda basado en una metaheurística de optimización global de gran escala que obtenga resultados similares o mejores frente a las propuestas del estado del arte.

1.3.2 Objetivos Específicos

- Definir la línea base del trabajo de investigación, compuesta por la arquitectura de la red neuronal profunda para resolver tres problemas de regresión (serie temporal, aproximación de una función y dataset California housing) y los algoritmos del estado del arte RMSProp y el evolutivo LEEA (Limited Evaluation Evolutionary Algorithm) propuesto por Morse.
- Adaptar las metaheurísticas seleccionadas (DECC-G, CC-CMA-ES, IHDELS y MOS) al proceso de entrenamiento de la red neuronal profunda en los tres problemas de regresión seleccionados y definir los cambios requeridos (si los hay) para resolver problemas de clasificación.
- Evaluar el desempeño del aprendizaje de la red neuronal profunda basado en los cuatro algoritmos metaheurísticos y los algoritmos del estado del arte (RMSProp y el evolutivo propuesto por Morse) mediante el análisis comparativo de las tasas de error obtenidas durante el proceso de entrenamiento de la red y el tiempo de duración del mismo.

1.4 Resultados obtenidos

A continuación, se resumen los principales resultados de la tesis:

a) **Monografía de la tesis**: se refiere al presente documento el cual presenta el estado del arte relacionado con el proceso de entrenamiento de redes neuronales profundas, la creación de un marco de trabajo para construir metaheurísticas que puedan optimizar el proceso de aprendizaje de arquitecturas de redes neuronales profundas, implementaciones de nuevos algoritmos meméticos, finalmente, se exponen los resultados obtenidos de las metaheurísticas de optimización global de gran escala

junto con nuevas propuestas meméticas que son comparadas frente a la línea base y estado del arte en el aprendizaje de redes neuronales profundas, RMSProp.

b) **Framework de pruebas**: se refiere al código fuente del framework de neuro evolución denominado Deep Neuroevolution Framework (DNF), construido para realizar las pruebas e implementar los algoritmos meméticos, arquitecturas y diferentes problemas relacionados con redes neuronales profundas. DNF se encuentra disponible en **https://github.com/julianfer87/Deep-Neuroevolution-Framework-DNF** y en un anexo digital (**Anexo A**) de esta tesis. La documentación del código se presenta en el **Anexo B**.

c) **Artículo** titulado **"Vegetation Index Based on Genetic Programming for Bare Ground Detection in the Amazon"** que fue presentado en el evento MICAI'17 (Mexican International Conference on Artificial Intelligence) y publicado en la revista **Lecture Notes in Computer Science** (ISSN 0302-9743), la cual para el año 2017 fue reconocida como **A2** por el PUBLINDEX de Colciencias y clasificada como **Q2** en **SJR**. Ver **Anexo C**. En este artículo se describe el uso de programación genética para definir un índice de vegetación que permita la detección de suelo desnudo en la selva amazónica a través de la evaluación de imágenes satelitales Planet. Con este trabajo se buscó observar el comportamiento de algoritmos evolutivos frente a procesos de clasificación en conjuntos de imágenes que tienen una alta complejidad y que son resueltos actualmente con aprendizaje profundo. Los resultados de la investigación mostraron que fue posible encontrar dicho índice y con ello clasificar las zonas deforestadas de manera precisa, aportando bases sólidas para comenzar la investigación.

d) **Artículo** titulado: **"Framework for the Training of Deep Neural Networks in TensorFlow Using Metaheuristics"** presentado en el evento IDEAL'18 (19th International Conference Intelligent Data Engineering and Automated Learning) y publicado en la revista **Lecture Notes in Computer Science** (ISSN 0302-9743), la cual para el año 2018 fue clasificada como **Q2** en **SJR**. Ver **Anexo D**. Este articulo presenta resultados preliminares y promisorios en la optimización de redes neuronales profundas convencionales utilizando metaheurísticas como Global Best Harmony Search, Differential Evolution, Particle Swarm Optimization sobre el conjunto de datos MNIST, se presenta la primera versión del framework DNF.

e) **Artículo** titulado: **"DNF: Deep Neuroevolution Framework over Tensorflow"** contiene la descripción detallada del framework, que se encuentra en proceso de evaluación en la revista **SoftwareX** (ISSN 2352-7110), la cual es reconocida como **A1** por el PUBLINDEX de Colciencias y clasificada como **Q1** en **SJR**. Ver **Anexo E**. Este artículo describe en detalle el framework desarrollado durante la tesis.

1.5 Estructura de la monografía

A continuación, se presenta la organización general del documento y un resumen de la información presentada en cada capítulo:

- **CAPÍTULO 1: INTRODUCCIÓN**: Hace referencia al presente capítulo que introduce el tema de investigación, presenta la pregunta de investigación que originó el trabajo, los aportes al problema, el objetivo general y específicos definidos inicialmente en el anteproyecto, una descripción general de los resultados obtenidos y finaliza con la organización del documento.

- **CAPÍTULO 2: CONTEXTO TEÓRICO Y ESTADO DEL ARTE**: En este capítulo se presentan conceptos teóricos relacionados con las redes neuronales profundas y sus algoritmos clásicos de entrenamiento, así como una explicación de las arquitecturas de redes y conjuntos de datos utilizados en aprendizaje profundo. Por otra parte, se presentan conceptos teóricos relacionados con la neuro evolución y metaheurísticas que son consideradas top en la resolución de problemas de optimización global de gran escala. Finalmente, se presentan los avances en el campo de la neuroevolución logrados en diversas investigaciones.

- **CAPÍTULO 3: METAHEURÍSTICAS DE OPTIMIZACIÓN GLOBAL DE GRAN ESCALA PARA EL ENTRENAMIENTO DE REDES NEURONALES PROFUNDAS**: En este capítulo se presenta los resultados obtenidos de comparar el algoritmo de entrenamiento de redes neuronales profundas RMSProp contra las propuestas de metaheurísticas tradicionales y de optimización global de gran escala. Los resultados presentados están basados sobre la arquitectura de red neuronal profunda propuesta por Morse et al., así como, sus tres conjuntos de datos, a saber: Función de aproximación, series de tiempo y California Housing.

- **CAPÍTULO 4: PROPUESTAS MEMÉTICAS INICIALES**: En este capítulo se muestran los algoritmos meméticos implementados en el proyecto, estos algoritmos están basados en la modificación de las metaheurísticas de la Mejor Búsqueda Armónica Global (Global Best Harmony Search, GBHS), Optimización por enjambre de partículas (Particle Swarm Optimization, PSO) y Evolución Diferencial (Differential Evolution, DE), a través de la adición de un proceso de explotación basado en optimizadores del gradiente descendente estocástico: RMSProp. Por otra parte, se muestran los resultados obtenidos con estos algoritmos meméticos optimizando una red neuronal profunda convencional sobre un conjunto de imágenes relacionado con los números dibujados a mano alzada MNIST. Se comparan con los resultados alcanzados por el RMSProp. Finalmente, se realiza la modificación de los algoritmos de Muestreo Múltiple de Descendientes (Multiple Offspring Sampling, MOS) y Ascenso de Colina por la Máxima Pendiente (Steeptest Ascent Hill Climbing, SAHC) utilizando el enfoque planteado en este capítulo, adicionando, una etapa de explotación basada en optimizadores del gradiente logrando vencer al RMSProp.

- **CAPÍTULO 5: FRAMEWORK PARA SOPORTAR LA EXPERIMENTACIÓN**: En este capítulo se presenta DNF, un marco de optimización del entrenamiento de redes neuronales profundas basado en metaheurísticas. Además, muestra el diagrama de clases que indica la organización e interrelaciones entre los diferentes módulos implementados, logrando mostrar que DNF tiene una arquitectura débilmente acoplada y una baja cohesión facilitando la inclusión de nuevas metaheurísticas, arquitecturas de redes neuronales y conjuntos de datos.

- **CAPÍTULO 6: PROPUESTAS MEMÉTICAS FINALES**: En este capítulo se presentan los resultados obtenidos por dos nuevos enfoques denominados SAHC_BestSlope y SAHC_E2 los cuales logran vencer al estado del arte, RMSProp, en conjuntos de datos relacionados con imágenes: MNIST (Dígitos dibujados a mano alzada) y FASHION-MNIST (Prendas de vestir). El entrenamiento se realizo utilizando una arquitectura de red neuronal profunda convolucional llamada LeNet. Adicionalmente, se explican los algoritmos de los dos nuevos enfoques demostrando que el lanzamiento inteligente de procesos de exploración en RMSProp basado en

información acumulada de la pendiente permite hacer frente a los problemas típicos de la alta dimensionalidad en el entrenamiento de redes neuronales profundas convolucionales.

- **CAPÍTULO 7: CONCLUSIONES Y TRABAJOS A FUTURO**: En este capítulo se presentan las conclusiones obtenidas al finalizar la tesis de maestría e ideas que el grupo de investigación espera realizar en un trabajo a futuro.

- **CAPÍTULO 8: REFERENCIAS**: Este capítulo final contiene las referencias bibliográficas de los artículos, libros, entre otros, consultados para la realización de la investigación.

2. Contexto teórico y estado del arte

2.1. Contexto teórico

2.1.1 Redes Neuronales Artificiales (RNA)

Estos modelos computacionales están inspirados en el funcionamiento del cerebro humano, el cual interconecta unidades pequeñas de procesamiento (neuronas) que reciben señales de neuronas previamente conectadas procesando la información y produciendo una señal de salida que puede ser entrada de otra neurona [14][15].

Estas neuronas se apilan y forman capas. Todo modelo de una red neuronal consta de tres capas o grupos principales: la capa de entrada, encargada de recibir los vectores con los datos de entrada de los diferentes conjuntos (x). UN conjunto de capas ocultas, encargadas de procesar la información a través de la aplicación de pesos (W) y sesgos/bias (b) usando además una función de activación (S) y una capa de salida la cual produce un vector con la respuesta de la red (h), ver **Ecuación (1)** [14]

$$h(x) = S(Wx + b) \qquad \textbf{(1)}$$

En la **Fig. 1** se observa la arquitectura de una red neuronal artificial. Aunque existen muchas arquitecturas y diferentes tipos de capas, la figura muestra la forma convencional de este tipo de modelos. Una particularidad de esta red es que puede ser considerada “profunda” debido a que tiene cuatro capas ocultas. Aunque no está establecido un rango para la consideración de profundo, en la práctica tener un número mayor o igual a cuatro capas ocultas incrementa exponencialmente el proceso computacional.

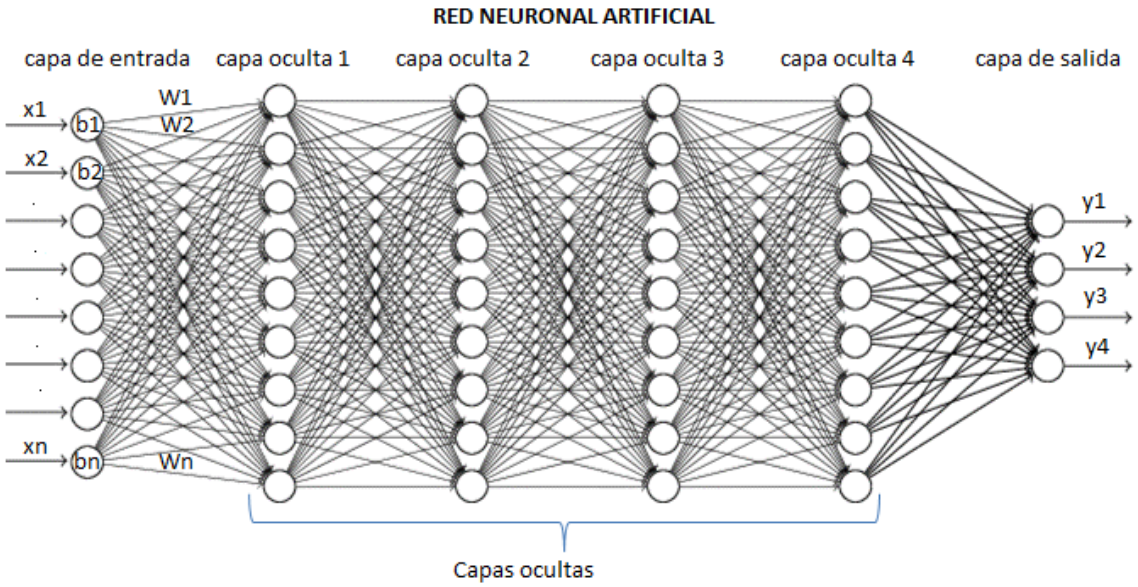

Fig. 1 Arquitectura de Red Neuronal Artificial profunda (Fuente propia)

2.1.2 Arquitecturas de redes neuronales convolucionales profundas

El aprendizaje profundo ha logrado solucionar tareas complejas que hasta hace poco eran exclusivas del ser humano, a través de diferentes arquitecturas de redes, problemas como: reconocimiento de rostro y lenguaje, procesamiento de lenguaje natural, traducción automática, entre otros [16]. A continuación, se nombran 4 tipos de arquitectura que por sus aplicaciones logran obtener una especial atención en el aprendizaje profundo: Redes preentrenadas no supervisadas (Pretrained neural networks), Redes neuronales convolucionales, Redes neuronales recurrentes y Redes neuronales recursivas [17].

1. **Redes preentrenadas no supervisadas**: en estas redes se incluyen los autoencoders, lasredes de creencia profunda y las redes adversarias generativas. Un diagrama que ilustra la arquitectura de estas redes se muestra en la **Fig. 2.**

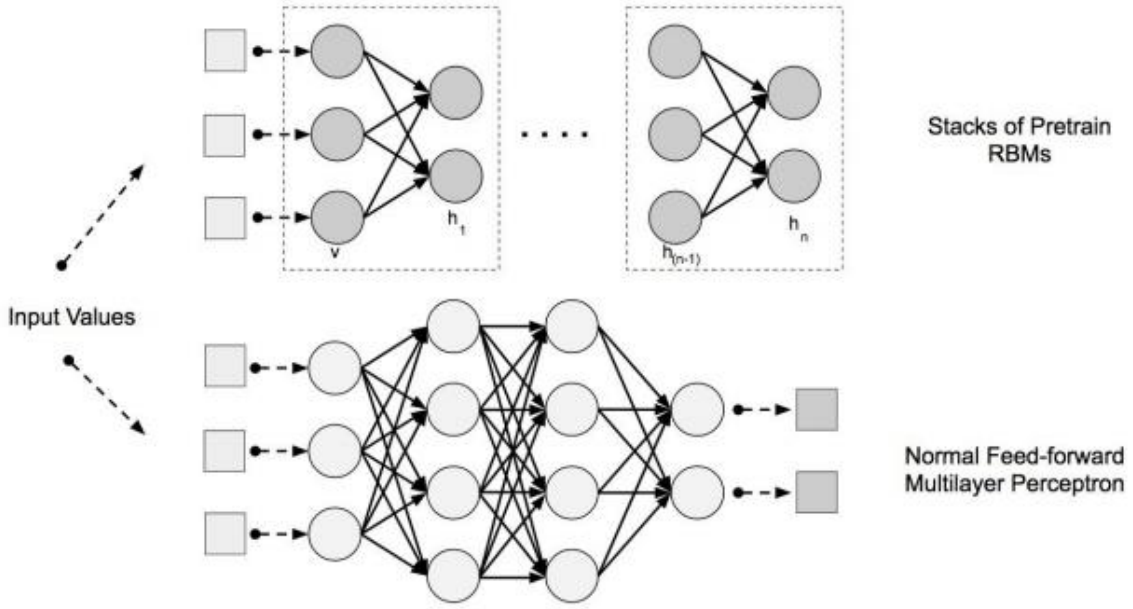

Fig. 2 Arquitectura de una red de creencia profunda (tomada de [18])

Este tipo de arquitecturas se componen en su parte de preentrenamiento de capas con máquinas restringidas de Boltzman, su principal función es la extracción de características de alto nivel del conjunto de datos de entrada, es decir que, cuando se solicite a las máquinas por la reconstrucción de los datos la respuesta sean valores muy cercanos al vector de entrada. Lo anterior es útil en el aprendizaje profundo porque logra generar un aprendizaje progresivo incrementando el nivel de las características y logrando que el aprendizaje sea mejor conforme dichas características aprendidas (mayor abstracción) de capas anteriores comiencen su propagación hacia adelante [18]. Una de las principales ventajas de estos tipos de redes neuronales radica en que utilizar los vectores de pesos proporcionados por las redes de creencia profunda logran que se inicie el proceso de entrenamiento en zonas del espacio de búsqueda más favorables logrando minimizar la función de costo.

2. **Redes neuronales recurrentes**: la principal particularidad de este tipo de redes se basa en que a diferencia de las demás arquitecturas pueden trabajar sobre conjuntos de vectores de características a través del tiempo, es decir que su aprendizaje no se basa en un único vector estático, por ejemplo, una imagen o un frame de un video o un vector con valores numéricos de una clase específica, por el contrario estas redes

pueden ser consideradas como múltiples copias de la misma red [19] lo que les permite funcionar más parecido a cómo trabaja el cerebro humano La **Fig. 3** muestra el diagrama de una red recurrente.

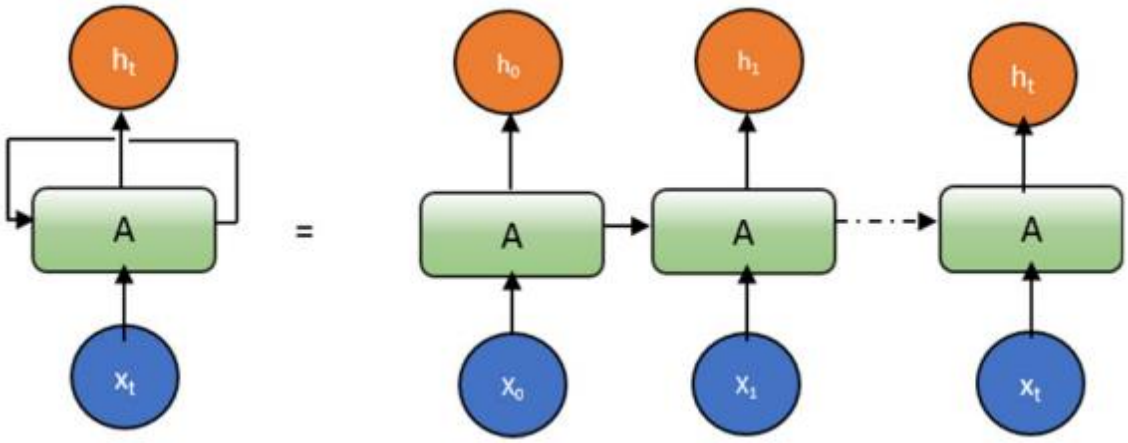

Fig. 3 Red neuronal recurrente (tomada de [19])

Las redes neuronales recurrentes son capaces de procesar una secuencia de vectores de entrada, esta operación la hacen vector por vector lo que les permite conservar el estado mientras modela el siguiente vector que ingresa, esta característica permite que la red pueda modelar teniendo en cuenta una línea temporal por lo que aplicaciones como traducción del lenguaje natural, análisis de video, predicción de series temporales, generación de música, entre otras, sean abordadas con estas arquitecturas [18].

3. **Redes neuronales recursivas**: son capaces de manejar entradas de longitud variable al igual que las redes recurrentes, sin embargo se diferencian con dichas redes en que no son capaces de modelar la estructura jerárquica del conjunto de datos de entrada [18]. Son modelos adaptativos no lineales, con un costo computacional elevado que permite crear modelos de procesamiento de información extremadamente complejos. Estas redes pueden resolver problemas de clasificación y regresión en tareas supervisadas como no supervisadas. El principal objetivo que persigue este tipo de arquitectura de red es estrechar el vacío entre los modelos de procesamiento simbólico y subsimbólico con el fin de combinar información numérica y simbólica en un mismo modelo a través del desarrollo de esquemas computacionales [20]. Sus principales aplicaciones hasta el momento están enfocadas en la bioinformática [21], análisis de imágenes [22], entre otras.

Esta investigación se centra en la optimización de redes neuronales profundas, por alcance del proyecto no fue posible abarcar todas las arquitecturas existentes, sin embargo, el presente trabajo logró centrarse en el entrenamiento de las redes neuronales convolucionales, arquitectura ampliamente usada en el campo industrial e investigativo, que ha permitido solucionar tareas hasta hace poco exclusivas del ser humano.

4. **Redes neuronales convolucionales**: inicialmente propuestas por Fukushima en 1988 [23] pero solo utilizadas con éxito por LeCun en 1990 [24], solucionando el problema de reconocimiento de dígitos hechos a mano. Después fue utilizada por varios investigadores en tareas relacionadas con reconocimiento de objetos. Tiene muchas ventajas frente a las redes neuronales profundas puesto que funcionan más parecido al sistema de visión humana, están más optimizadas para estructuras de procesamiento de imágenes 2D y 3D, además, logran una mayor capacidad de

abstracción y mejor aprendizaje de características 2D [19]. En la **Fig. 4** se observa la estructura de una red neuronal convolucional:

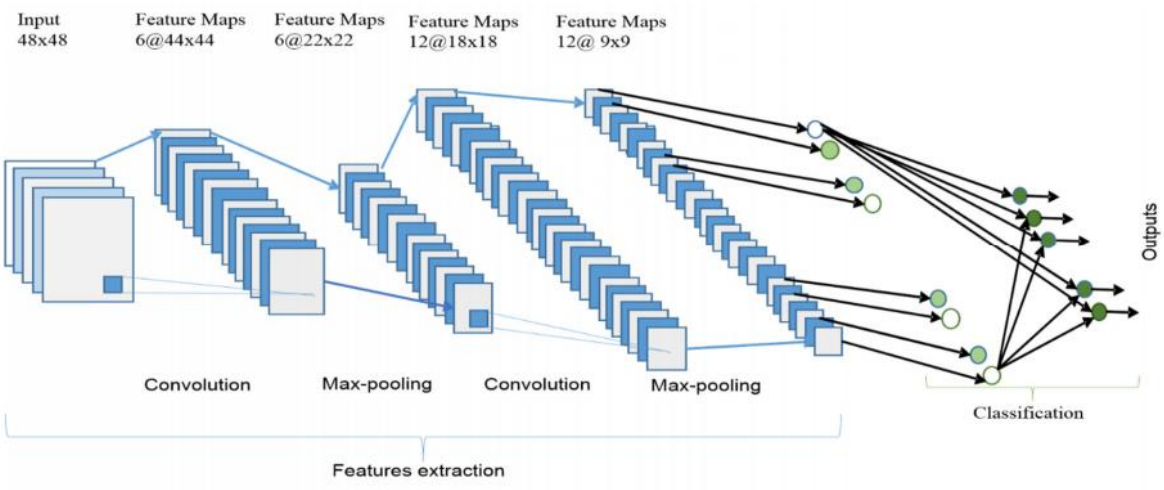

Fig. 4 Red neuronal convolucional (tomada de [19])

La **Fig. 4** muestra la arquitectura completa de una red neuronal convolucional, se puede observar que se compone de dos partes: extracción de características (capas azules oscuras y claras) y una capa final de clasificación, normalmente densa junto con la de salida. La extracción de características se realiza principalmente con filtros o capas convolucionales los cuales permiten aplicar sobre la imagen de entrada operaciones sobre los pixeles a través de kernels (convolución) encontrando mejores características conforme se realiza la propagación hacia adelante, es decir, mejorando la abstracción [18]. También se pueden detallar capas de “pooling” las cuales reducen el espacio de búsqueda ayudando a evitar el sobreajuste. Finalmente aparece en la etapa de clasificación una capa densa y una capa de salida que le permite a la red proporcionar las diferentes clases requeridas en la tarea que esté resolviendo. La combinación de capas convolucionales y capas de pooling han dado origen a diversos tipos de familias de redes neuronales convolucionales profundas especializadas en el reconocimiento de patrones visuales, dos ejemplos puntuales de dichas redes se observan en: LeNe t[25] y AlexNet [26]. A continuación, en las **Fig. 5** y en la **Fig. 6** se muestran estas arquitecturas.

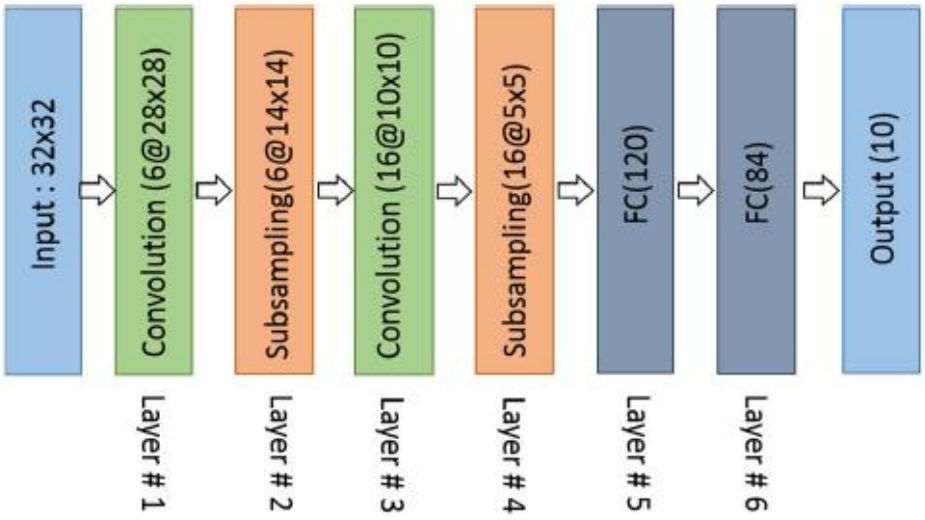

Fig. 5 Arquitectura de la red neuronal convolucional LeNet (Tomada de [19])

LeNet se ha caracterizado por resolver con éxito el conjunto de datos de dígitos escritos a mano MNIST y otras aplicaciones de visión por computadora y aprendizaje de máquina. Fue propuesta en 1990, sin embargo, solo pudo ser utilizada hasta el año 2010 puesto que anteriormente no existía la capacidad computacional y de memoria para implementarla.

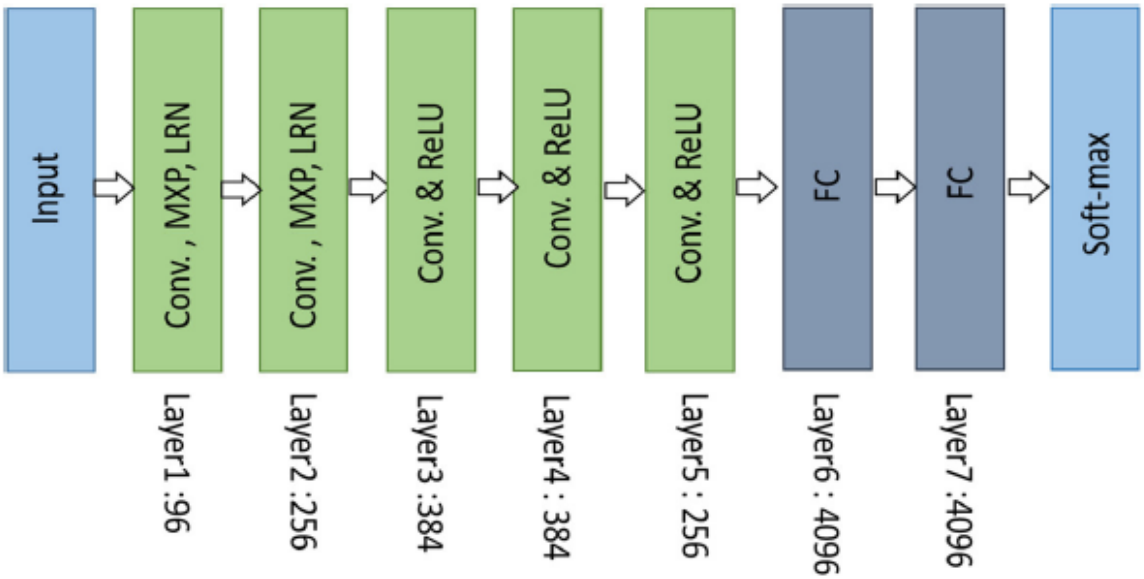

Fig. 6 Arquitectura de la red neuronal convolucional AlexNet (Tomada de [19])

AlexNet fue creada en 2012 por Alex Krizhevesky y otros, ganó el más difícil reto de reconocimiento de objetos llamado Reto de reconocimiento visual de gran escala ImageNet. AlexNet logró vencer a los algoritmos tradicionales de aprendizaje máquina y enfoques de visión por computadora. Es considerada una arquitectura amplia y profunda debido a la gran cantidad de filtros convolucionales.

2.1.3 Conjuntos de datos tradicionales y de imágenes en redes neuronales profundas

Para esta investigación fueron considerados 5 conjuntos de datos relacionados con el aprendizaje profundo, tres de ellos están basados en tareas de regresión y los restantes en tareas de clasificación. Con respecto a la regresión se tiene los conjuntos de datos utilizados por Morse en [27]: Función de aproximación, serie de tiempo caótica Mackey-Glass y tarea de predicción del valor de una vivienda en USA (California Housing). Por otra parte, se utilizaron conjuntos de datos de imágenes para realizar tareas de reconocimiento de patrones: MNIST y FASHION-MNIST. A continuación, se realiza una breve descripción de los conjuntos de datos.

2.1.3.1 Función de aproximación

Este primer dominio tiene 800 muestras de entrenamiento y es considerado una tarea de aproximación, la función se muestra en la **Ecuación (2).**

$$f(x,y) = \frac{\sin\big(5x(3y+1)\big)+1}{2} \qquad \textbf{(2)}$$

Morse en [27] plantea que la superficie que describe esta función es lo suficientemente difícil para encontrar un mínimo.

2.1.3.2 Serie de tiempo caótica de Mackey-Glass

Está considerada como una tarea de predicción, contiene 1200 muestras de entrenamiento y en el pasado según menciona Morse en [27] ha sido utilizada debido a su alto grado de dificultad. En la **Ecuación (3)** se observa la serie de tiempo.

$$\frac{dx}{dt} = \beta \frac{x_r}{1+x_r^n} - \gamma x, \qquad \gamma, \beta, n > 0 \qquad \textbf{(3)}$$

Donde, $\gamma = 1,\ \beta = 2,\ n = 9.65,\ x_0 = 1.1,\ x_1 = 1.2.$

2.1.3.3 California Housing

El último dominio correspondiente a las tareas de predicción está basado en la estimación del precio de las viviendas en Estados Unidos llamado comúnmente conjunto de datos California Housing. Posee 10000 muestras de entrenamiento y cada muestra tiene asociado 8 atributos.

2.1.3.4 MNIST

Corresponde a una tarea de clasificación y reconocimiento de patrones visuales debido a que este conjunto de datos se compone de imágenes de dígitos del 0 al 9 dibujados a mano de tamaño 28 filas por 28 columnas [28], el total de datos está distribuido en dos conjuntos: 60000 muestras para entrenamiento y 10000 muestras para prueba. En esta investigación, MNIST es resuelto con la arquitectura LeNet debido a que es el estado del arte para este conjunto de datos.

2.1.3.5 FASHION-MNIST

Comparte las mismas características que MNIST tanto en tamaño de conjuntos de entrenamiento, prueba, número de clases y dimensiones de las imágenes, sin embargo, el

contenido de las imágenes está relacionado con artículos de ropa [29] lo que conlleva a que la complejidad de la información aumente considerablemente con respecto a MNIST.

2.2 Estado del arte

A continuación, se describen los principales temas abordados en esta propuesta de investigación. Primero se realiza las definiciones relacionadas con Deep Learning y sus principales algoritmos de entrenamiento (Backpropagation, SGD y RMSProp). Luego se hace un recuento histórico de cómo la neuro-evolución ha abarcado el problema de entrenamiento de las redes neuronales hasta la actualidad, con lo anterior se pretende contextualizar el problema planteado en esta investigación. Finalmente, se realiza una descripción general de las principales metaheurísticas, que son el estado del arte, en problemas de optimización continua de gran escala.

2.2.1 Deep Learning

Constituye un conjunto de algoritmos de aprendizaje de máquina que usan redes neuronales de múltiples capas con unidades de procesamiento no lineal que se conectan en cascada dándole la cualidad de "profundo". Esta cascada de capas permite la extracción y transformación de características, brindando niveles múltiples de aprendizaje de representaciones lo que es equivalente a tener diferentes niveles de abstracción del conocimiento [30]. Deep Learning puede ser utilizado para análisis de patrones y algoritmos de clasificación, entre otros [31].

Deep Learning es parte de una familia de métodos de aprendizaje de máquina basados en el aprendizaje de representaciones de datos, por ejemplo, si se tiene como entrada una imagen, este conjunto de datos puede ser representado en diferentes niveles de jerarquía, las capas iniciales de la red tendrán un nivel de abstracción menor por lo que podrán representar la imagen de entrada como vectores de intensidades de pixeles, mientras que si se profundiza en la red, el nivel de abstracción será mayor logrando representar el conjunto de datos, por ejemplo, en conjunto de bordes o regiones de formas particulares, etc.

Al incrementar el número de capas ocultas en procesos Deep Learning el entrenamiento a través del Backpropagation y el SGD, sufre un estancamiento en la velocidad del aprendizaje debido al problema conocido como el Desvanecimiento del gradiente (Vanishing gradient) [32]. Esto ocurre porque matemáticamente, el SGD utiliza el proceso de la derivada (específicamente, la regla de la cadena) y debido a la cantidad de capas ocultas en la red, los valores de gradiente cercanos a cero son multiplicados constantemente en la etapa del Backpropagation, haciendo que el aprendizaje sufra un desaceleramiento y por ende el entrenamiento no logre llegar al óptimo global. Fue así como en el año 2006, Geoffrey Hinton y Ruslan Salakhutdinov [33][34] lograron entrenar una red neuronal profunda mediante un proceso de "pre-entrenamiento", el cual consiste en tratar cada capa de la red como una máquina restringida de Boltzmann (RBM) obteniendo como resultado una aceleración en el aprendizaje de la misma.

A partir de la solución planteada en [33] para el problema del desvanecimiento del gradiente, se desarrollaron muchas arquitecturas de aprendizaje profundo, a continuación se nombran algunas:

- Redes Neuronales Convolucionales (CNN), que se convirtieron en los algoritmos adecuados para procesamiento de imágenes y datos bi-dimensionales [35][36];

- Redes Neuronales Recursivas (RNN) [37], cuya principal función se basa en aprender representaciones distribuidas de estructuras de datos, una aplicación de dichas redes se observa en tema del procesamiento del lenguaje natural [38];
- Redes de Creencia Profunda (DBN) [39], que son eficientes para un entrenamiento no supervisado debido a que utilizan las máquinas restringidas de Boltzmann (RBM) permitiendo reconstruir el conjunto de datos cada vez que se profundiza en la red;
- Stacked Denoising Auto-encoders [40], en el que el resultado final del entrenamiento se usa como entrada de un algoritmo de aprendizaje supervisado, tales como máquinas de soporte vectorial o regresión lógica mutilclase [40], lo anterior es posible debido a que el autoencoder se comporta como un reductor de características, similar a lo que se hace con el Análisis de Componentes Principales (Principal Component Analysis, PCA).

En los siguientes ítems se explica de forma general en qué consisten los algoritmos de entrenamiento y optimización del aprendizaje de una red neuronal profunda.

2.2.2 Backpropagation y el Gradiente Descendente Estocástico (SGD)

El algoritmo de Backpropagation es un método común para el entrenamiento de redes neuronales artificiales (y también de redes neuronales profundas, DNN) basado en el método de optimización denominado el gradiente descendente. El backpropagation le permite aprender a la red neuronal multicapa representaciones internas apropiadas teniendo en cuenta muestras de conjuntos de datos que ingresan en su capa de entrada [41]. El proceso de aprendizaje de la red se realiza en dos etapas que se repiten una cantidad de veces (épocas). En la primera etapa, un vector de representaciones del problema ingresa a la red, mediante la multiplicación individual de pesos sinápticos y la sumatoria de dichos factores multiplicativos más un sesgo, los resultados se propagan hacia adelante capa por capa hasta obtener un resultado final. En la segunda etapa, la salida obtenida se compara con la salida deseada mediante una función de pérdida obteniendo un valor de error en cada una de las neuronas de la capa de salida, ese error calculado se propaga hacia atrás usando la regla de la cadena, con el objetivo de repartir sobre cada capa y sus neuronas el porcentaje de error con el que contribuyeron durante la primera etapa. En resumen, el objetivo final del Backpropagation es realizar cálculos de derivadas parciales (o Gradiente Descendente Estocástico) de una función de pérdida con respecto a los pesos de la red permitiéndole actualizar los pesos sinápticos y así lograr el aprendizaje en las RNA [42].

2.2.3 Algoritmos de Optimización basados en SGD

Estudios han demostrado que en el espacio de soluciones relacionado con la alta dimensionalidad, existen multiplicidad de óptimos locales y especialmente puntos de silla [7]. El SGD es vulnerable a estas características, por lo que en múltiples investigaciones se han desarrollado algoritmos de optimización del SGD, permitiendo que el SGD escape de estas zonas de conflicto. El SGD tiene problemas en su descenso sobre zonas que son mucho más empinadas que otras, generando fluctuaciones y por ende un desacelaramiento para encontrar el óptimo global, el Momentum (2013) [43] soluciona este problema adicionando una fracción del vector de posición de un estado anterior del SGD al estado actual, logrando impulsar el descenso por la pendiente del espacio de soluciones y amortiguando las oscilaciones. Sin embargo, hacer que el SGD descienda sin ningún tipo de conocimiento o guía es ineficiente, por ello, el algoritmo Nesterov Accelerated Gradiente (NAG, 1983) [44] define una estimación previa del valor de la pendiente cuando se hace el siguiente movimiento, introduciendo conocimiento al

descenso del SGD, haciendo que pueda detener su impulso en zonas donde el espacio de soluciones no sea una mejor solución que la actual, permitiendo hacer una corrección al vector de movimiento.

Los esfuerzos para continuar mejorando el SGD conllevaron al desarrollo del algoritmo Adagrad (2010) [45] que realiza cambios a NAG, en términos de hacer pequeñas o grandes actualizaciones del vector de posición dependiendo de la frecuencia en que aparecen los parámetros de entrenamiento. Parámetros frecuentes, actualizaciones pequeñas y viceversa. Con este algoritmo se supera el problema de la escasez de los datos (sparse data) [46] y se elimina el problema de sintonización manual de la tasa de aprendizaje. Una aplicación de este algoritmo fue realizada por Pennington et al. [47] quien usó Adagrad para entrenar GloVe, un algoritmo de aprendizaje no supervisado para obtener representaciones vectoriales de palabras. Matemáticamente, Adagrad va acumulando los gradientes (magnitudes positivas) en el denominador de su función, lo que conlleva a que las tasas de aprendizaje se vuelvan pequeñas en magnitud (debido a la división por un factor grande) generando la pérdida de la capacidad de aprendizaje, por lo que el algoritmo Adadelta (2012) [48] nace como extensión de Adagrad y busca reducir la tasa de aprendizaje monótonamente decreciente a través de la acumulación de los gradientes pasados pero restringidos a una ventana de tamaño fijo, logrando eliminar por completo la sintonización por defecto del valor de la tasa de aprendizaje. Adam (Adaptive Moment Estimation, 2014) [49] propone otro algoritmo basado en SGD, que calcula las tasas de aprendizaje adaptativo para cada parámetro, algunas características comunes con otros métodos explicados anteriormente, están relacionadas con la acumulación de promedios de decaimiento exponencial de gradientes cuadrados pasados (como en Adadelta) y además, conserva promedios de decaimiento exponenciales de gradientes pasados como en el algoritmo de Momentum.

Rprop (2012) es un algoritmo de optimización del SGD que nace en paralelo con Adadelta por lo que busca de igual manera evitar el estancamiento en el aprendizaje que sufre el algoritmo Adagrad, este enfoque fue desarrollado por Hinton [50] y usa específicamente el signo del gradiente para poder adaptar el tamaño del paso separadamente para cada peso de las neuronas, lo anterior, se basa en que la magnitud del gradiente puede ser muy diferente para diferentes pesos y puede variar durante el aprendizaje, por lo que no es viable la escogencia de una única tasa global de aprendizaje.

El algoritmo Rprop funciona incrementando el tamaño del paso para un peso de forma multiplicativa si los signos de los dos últimos gradientes coinciden y decrece la magnitud del paso si los signos no coinciden. Para mejorar dicho algoritmo, se plantea RMSprop (2012) como la unión de rprop y el concepto de mini-batch, que define una ventana de promedio móvil (moving average) de los gradientes cuadrados para poder actualizar los pesos debido a cada mini-batch, haciendo que el aprendizaje trabaje mejor [50].

2.2.4 Neuro evolución

La neuro-evolución se originó en la década de los 80’s y consiste en el proceso de entrenar las redes neuronales mediante algoritmos evolutivos, definiendo el problema de entrenamiento como un problema de optimización continua [51][52]. Un primer estudio fue realizado por Montana y Davis [53] en 1989, quienes entrenaron una red neuronal artificial para realizar un proceso de clasificación de imágenes de Sonar mediante la implementación de un algoritmo genético, sus resultados obtienen mejoras de precisión frente al entrenamiento con Backpropagation. Estos trabajos surgieron en la época en que el Backpropagation estaba en auge [54] lo que conllevó a que los desarrollos relacionados en el campo de la neuro evolución sufrieran un estancamiento. Pero en los años noventa

se introduce una gran variedad de enfoques [55], haciendo que la neuro evolución sea protagonista nuevamente en la optimización del aprendizaje de las RNAs. En trabajos posteriores de Yao (1999) [55] y Floreano (2008) [51] se plantea cómo la combinación entre el aprendizaje y la evolución con las redes neuronales se consideran fundamentales para la adaptación, sus revisiones incluyen las diferentes combinaciones entre las RNAs y los algoritmos evolutivos (EA's) e involucran los EA con los pesos de conexiones de la red neuronal, arquitecturas, reglas de aprendizaje y características de entrada.

El reto de superar las capacidades del backpropagation usando algoritmos evolutivos (ó neuro evolución) hizo que aparecieran estudios que igualaban (1993) [56] o incluso superaban a backpropagation en problemas de clasificación [57][53][58] durante los años de 1993-1995. Los avances logrados durante esa época hicieron suponer que las redes neuronales evolucionarían a redes neuronales genéticas (1990) [59]. Además, aparecen trabajos que permiten determinar, mediante Algoritmos Genéticos Estructurados (1992) [60], la topología de una red neuronal y sus pesos de conexión. Otros trabajos relacionados, por ejemplo en 1993 [61], enfrentan el problema de diseñar de forma óptima estructuras para redes neuronales mediante el uso de algoritmos genéticos para solución de problemas específicos. Durante ese mismo año otras investigaciones fueron realizadas [62][12] en donde se demostró que las características de muestreo global del algoritmo genético complementaban las búsquedas locales realizadas por el gradiente descendente, en cuyos casos se mostraba el potencial de combinar la evolución de la topología con el backpropagation. Sin embargo, con el incremento de la capacidad computacional, los problemas de clasificación y regresión migraron hacia la alta dimensionalidad, llevando a reconocer por parte de la comunidad científica, las limitaciones que tenían los AE's en la optimización de redes neuronales más complejas y de mayor tamaño en sus conexiones, así como, las limitaciones a la hora de procesar un gran volumen de variables [5].

En el año 2002, Mandischer [63] concluye que las estrategias evolutivas (ES's) sólo pueden competir contra el gradiente descendente en escenarios de problemas pequeños y que las ES's pueden entrenar redes neuronales con una función de activación no diferenciable. Además, con resultados posteriores el SGD y el backpropagation mostraron un dominio en los procesos de clasificación y tuvieron éxito en el aprendizaje profundo (Deep Learning) [6][2]. Lo anterior hizo que la neuro evolución migrara a territorios donde la retropropagación era más difícil de aplicar: el aprendizaje por refuerzo (reinforcement learning), que requiere de conexiones recurrentes y arquitecturas especializadas [11][64], dichas arquitecturas permiten enfocarse y solucionar problemas muy complejos de control y toma de decisiones, para lo cual el SGD no tiene una respuesta adecuada. En ésta área aparecieron algoritmos como NEAT [65], HyperNEAT [66] y un algoritmo evolutivo derivado de CMA-ES [67] en 2003.

El espacio de soluciones, en problemas continuos de alta dimensionalidad, como el entrenamiento de RNAs, implica el manejo de múltiples óptimos locales y zonas de puntos de silla [7]. El gradiente descendente estocástico no es ajeno a estas características por lo que, diversas investigaciones se han enfocado en la optimización del algoritmo, entre los más importantes y previamente mencionados, se tiene a Momentum [43], Nesterov accelerated gradient [44], Adagrad [47], Adadelta [48], Adam [49] y la optimización más reciente propuesta por Hinton, RMSProp [50]. A pesar del dominio del SGD en el aprendizaje de redes neuronales profundas, los investigadores se siguen preguntando ¿por qué el gradiente descendente tiene ventajas en la alta dimensionalidad, a pesar de que está sujeto a caer en los óptimos locales? Esta pregunta generó una hipótesis planteada por Morse et al. [5] en 2015, en la que propone que la alta dimensionalidad

tiene múltiples rutas de escape, por lo que esas mismas ventajas que recibe el SGD pueden ser también favorables para los algoritmos evolutivos. Morse, propuso un algoritmo genético sencillo para la optimización de una arquitectura de red neuronal profunda, con tres conjuntos de datos, que por su cantidad de variables están inmersos en la alta dimensionalidad, además, plantea que para ser competitivos contra el SGD el número de instancias evaluadas por generación deben ser pequeñas (o realizadas por lotes), rompiendo el esquema de evaluación total de las instancias en cada generación de los algoritmos evolutivos. Su trabajo constituyó en ese momento el estado del arte en neuro evolución aplicada a procesos deep learning, logrando igualar el rendimiento de algoritmos de optimización tradicionales basados en SGD, como RMSProp, y superar las tasas de aprendizaje en la evaluación de una serie temporal. En el año 2017, los trabajos en neuro evolución continuaron y cada vez se observa con más éxito que las redes neuronales profundas pueden ser entrenadas utilizando algoritmos genéticos simples [68], por ejemplo en [69] una red neuronal convolucional profunda fue entrenada utilizando un algoritmo genético simple para solucionar problemas de aprendizaje por refuerzo profundo, solucionando con éxito problemas como jugar en Atari y humanoid locomotion. Un trabajo más reciente fue publicado en el año 2018 [70], en esta investigación crean un algoritmo llamado EvoDeep, el cual se enfoca en optimizar los parámetros y arquitectura de las redes neuronales profundas logrando maximizar la precisión de clasificación sobre el conjunto de datos MNIST, la investigación refuerza que los algoritmos evolutivos son otra solución al proceso de entrenamiento de redes neuronales profundas.

2.2.5 Metaheurísticas de optimización global de gran escala

Un proceso de optimización global es considerado de gran escala cuando se trabaja con espacios de soluciones con más de 1000 variables [71][5]. En Deep Learning las conexiones entre las neuronas de las capas ocultas representan un conjunto extenso, superior a este límite, por lo que se considera como un tema de optimización global de gran escala (Large Scale Global Optimization, LSGO). Con respecto a lo anterior y teniendo en cuenta el alcance del proyecto, se seleccionaron cuatro meta-heurísticas de optimización continua especializadas en problemas de alta complejidad, dicha selección se hace con base en los resultados de la competencia de la IEEE CEC realizada en el año 2015 [72], en la cual, las meta-heurísticas demostraron resolver diversos tipos de problemas relacionados con la alta dimensionalidad.

2.2.5.1 Evolución Diferencial basado en Coevolución Cooperativa

Publicado por Yang et al. [73][74] es un método de Evolución Diferencial (Differential Evolution, DE) basado en Coevolución Cooperativa (Cooperative Coevolution , CC), que utiliza agrupamientos aleatorios para resolver problemas (large scale global optimization, LSGO) con dimensiones de 500 y 1000, llamado Coevolución cooperativa basada en evolución diferencial (Cooperative Coevolution based on differential evolution, DECC-G). Diseñado para problemas separables pero que también funciona bien para problemas no separables. Este algoritmo particiona aleatoriamente (con un tamaño de conjunto predeterminado) un vector objetivo n-dimensional logrando obtener múltiples subconjuntos de baja dimensionalidad, los cuales evolucionan con un algoritmo de búsqueda de vecindario, SaNSDE [75]. A cada subconjunto o subcomponente se le asigna un peso después de cada ciclo y a través de un algoritmo de optimización se afinan los pesos, pero esta vez el proceso se realiza sobre un conjunto dimensionalmente más pequeño que en el problema original. Los resultados muestran un rendimiento eficiente en funciones multimodales no separables hasta 1000 dimensiones, pero su

rendimiento decae entre mayor sea el número de variables que interactúen con respecto al límite expresado anteriormente [76].

2.2.5.2 Muestreo de Múltiples Descendientes

El muestreo de múltiples descendientes (Multiple Offspring Sampling, MOS) es un framework que permite desarrollar algoritmos evolutivos híbridos dinámicos [77][78]. Ha sido aplicado con éxito a diversos problemas de optimización continua [79][80]. El framework combina varios algoritmos basados en población y búsquedas locales de manera dinámica, permitiendo crear nuevas soluciones candidatas dependiendo de una medida de calidad, por ejemplo, mejor valor promedio del fitness en los nuevos individuos creados. Las etapas que componen el algoritmo están divididas por bloques de números fijos de evaluaciones de la función objetivo y la ejecución de los algoritmos involucrados en MOS, donde su escogencia está basada en el rendimiento que tiene en la iteración pasada.

2.2.5.3 Hibridización Iterativa de Evolución de la adaptación de la matriz de covarianza usando coevolución cooperativa

El algoritmo de Hibridación Iterativa de Evolución Diferencial con Búsqueda Local (Iterative Hybridization of Differential Evolution with Local Search, IHDELS) realiza la combinación entre el proceso de exploración (Evolución diferencial) y la explotación (Búsqueda local), que ayuda a acelerar la convergencia en problemas LSGO [81]. El algoritmo incluye un método de reinicio e incorpora una alternancia entre los métodos de búsqueda local (BL). El algoritmo exploratorio que utiliza IHDELS es SaDE [82], un algoritmo basado en evolución diferencial clásico que auto adapta sus parámetros, mientras que, durante cada iteración el algoritmo IHDELS escoge entre dos tipos de BL: MTS LS-1 [83], diseñado especialmente para problemas de alta dimensionalidad; y L-BFGS-B [84], que utiliza una aproximación del gradiente para optimizar la búsqueda. Los dos métodos de BL brindan características especiales, mientras que el MTS es rápido y adecuado para problemas separables, no es sensible a rotaciones. Por otra parte, L-BFGS-B es sensible a rotaciones, pero menos potente para problemas separables.

2.2.5.4 Estrategia de Evolución de la Adaptación de la Matriz de Covarianza usando Coevolución Cooperativa

La adaptación de la matriz de covarianza en estrategias evolutivas (stands for covariance matrix adaptation evolution strategy, CMA-ES) ha demostrado resolver problemas multimodales no separables, sin embargo, cuando se aumentan las dimensiones, el algoritmo tiene problemas. Por consiguiente aprovechando los beneficios que ofrece la coevolución cooperativa se propone el algoritmo CC-CMA-ES, que tiene una efectividad mayor frente a problemas LSGO [85]. Este algoritmo mejora el esquema de muestreo extrayendo una distribución Gaussiana subespacial de la distribución Gaussiana global, a diferencia de su antecesor que dividía la población en pequeñas subpoblaciones a las cuales les aplicaba operaciones de mutación y cruce para generar descendencia. Además, en CC-CMA-ES se implementan dos estrategias de descomposición basadas en la diagonal de la matriz de covarianza: Min-Variance (MiVD) y Max-Variance (MaVD), con el objetivo de balancear la exploración y explotación en el algoritmo. CC-CMA-ES ha sido probado con funciones de referencia provistas por CEC2013 Special Session on large scale global optimization [86] demostrando excelentes resultados frente a algoritmos del estado del arte como DECC-G.

En el framework desarrollado para esta investigación se agregan algunas de estas metaheurísticas de optimización global de gran escala, así como, algoritmos metaheurísticos tradicionales como Global Best Harmony Search y Particle Swarm Optimization, entre otros.

Capítulo 3

3 Metaheurísticas de optimización global de gran escala para el entrenamiento de redes neuronales profundas.

Los algoritmos evolutivos, han demostrado a lo largo de diversas investigaciones, su potencial para resolver problemas en el dominio de la alta dimensionalidad. La neuro evolución aparece con el objetivo de brindar una opción a la optimización del entrenamiento de redes neuronales artificiales. Morse et al. en [5] demostraron como un algoritmo evolutivo sencillo denominado LEEA logra ser competitivo frente a los algoritmos de entrenamiento de redes neuronales: SGD y RMSProp. Sin embargo, un algoritmo evolutivo sencillo no tiene todo el potencial para tratar con problemas de alta dimensionalidad, muy a pesar de que sus resultados pueden ser considerados buenos, no logró vencer al RMSProp en la mayoría de los dominios propuestos. En esta investigación se planteó la hipótesis de qué metaheurísticas especializadas en la alta dimensionalidad como lo son: MOS, IHDELS y DECC-G lograrían tener un mayor éxito que LEEA en el entrenamiento de una arquitectura de red neuronal profunda planteada en [5]. Este capítulo muestra los resultados obtenidos.

La red neuronal inicial propuesta por Morse et al. (MorseNet, en adelante) y en la que se enfoca esta investigación, está clasificada como una red neuronal profunda convencional. Se compone de una capa de entrada que varía el número de sus neuronas según el conjunto de datos que se trabaje: 1170 entradas para la función de aproximación, 1270 entradas para serie de tiempo Mackey-Glass y 1470 entradas para California Housing. La red tiene dos capas ocultas con 50 y 20 nodos respectivamente. En su capa de salida tiene un único nodo debido a que se trata de problemas de regresión (Minimizar el error cuadrático medio, RMSE). Con el objetivo de optimizar el aprendizaje de la red neuronal con los tres dominios se usaron tres metaheurísticas de optimización global de gran escala, como se nombró anteriormente, cada metaheurística fue ejecutada 30 veces para cada dominio, además, se compararon con los algoritmos del estado del arte: RMSProp y LEEA. A continuación, se muestran los resultados de las 30 repeticiones del experimento (nube de puntos) para el conjunto de datos California Housing utilizando el algoritmo MOS.

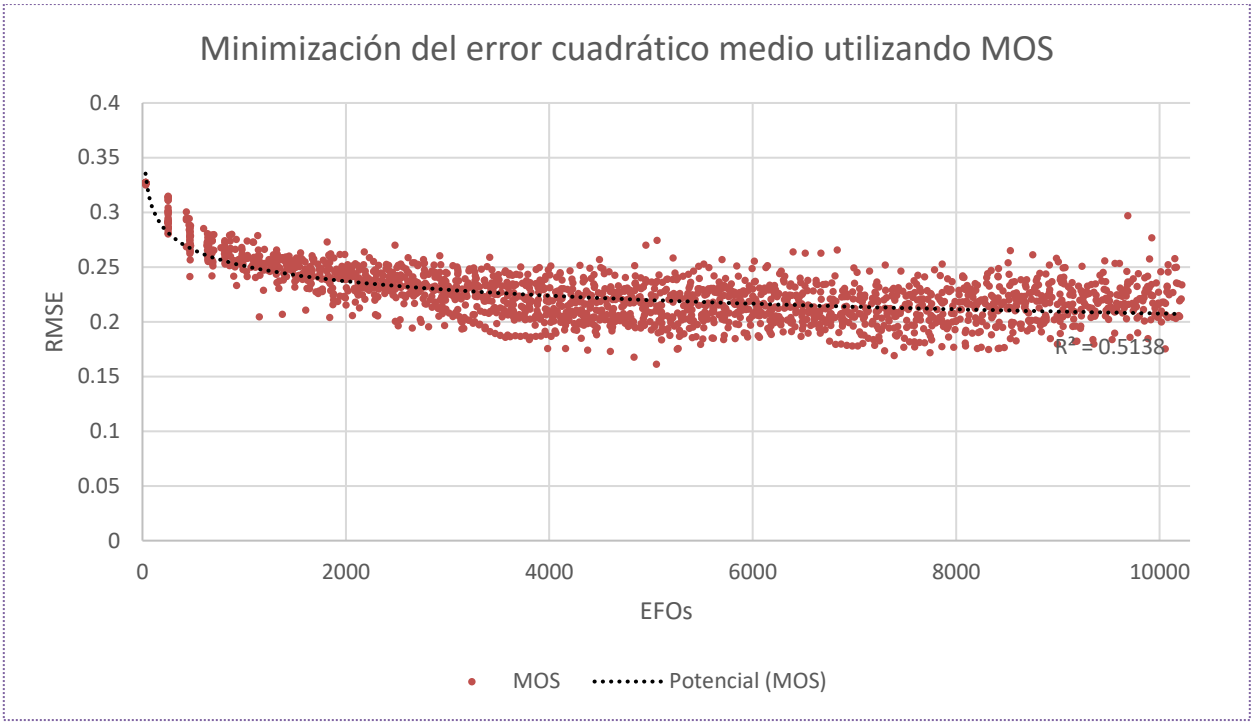

Fig. 7 Minimización del error cuadrático medio en arquitectura: MorseNet, Dataset: California Housing, Metaheurística: MOS – 30 repeticiones

El total de las gráficas que se obtendrían con los resultados de los experimentos sería de 15 (3 datasets x 5 algoritmos en nubes de puntos), por cuestiones de espacio en el documento y con el fin de mejorar la presentación y comprensión de los resultados se opta por generar gráficas que muestren líneas de tendencia o ajuste sobre toda la nube de puntos generada de las 30 repeticiones por ejecución de cada algoritmo en cada dataset. Estas líneas de tendencia permiten analizar de mejor manera el comportamiento de las metaheurísticas de optimización global con respecto al RMSProp y LEEA. En las siguientes figuras se grafican los resultados.

La **Fig. 8** muestra las líneas de tendencia de los algoritmos de optimización global de gran escala: MOS, IHDELS y DECC-G, así como, los algoritmos del estado del arte RMSProp y LEEA. En esta figura se puede apreciar que RMSProp obtiene mejores resultados durante el entrenamiento de la red para el problema de aproximar la función, sin embargo, MOS demuestra que puede ser competitivo frente a este algoritmo en el aprendizaje profundo. Por otra parte, parcialmente se corrobora la hipótesis de esta investigación puesto que MOS siendo una metaheurística de optimización global de gran escala tiene mejores resultados que el simple algoritmo evolutivo LEEA.

En la **Fig. 9** se muestran las líneas de tendencia obtenidas en el problema de la serie de tiempo, lo primero que se puede observar es que todas las metaheurísticas de optimización global vencen al algoritmo evolutivo LEEA, lo que corrobora la hipótesis planteada en esta investigación. Por otra parte, MOS vence al RMSProp en la tarea de minimizar el error cuadrático medio (RMSE), demostrando que esta metaheurística brinda nuevas posibilidades en los procesos de entrenamiento. En la **Fig. 10** se muestran los resultados obtenidos en el problema California Housing, se puede observar que todas las metaheurísticas de optimización global son competitivas frente al RMSProp y que LEEA

no es capaz de minimizar el RMSE en las 10000 evaluaciones de la función objetivo (EFOs) que duró el entrenamiento.

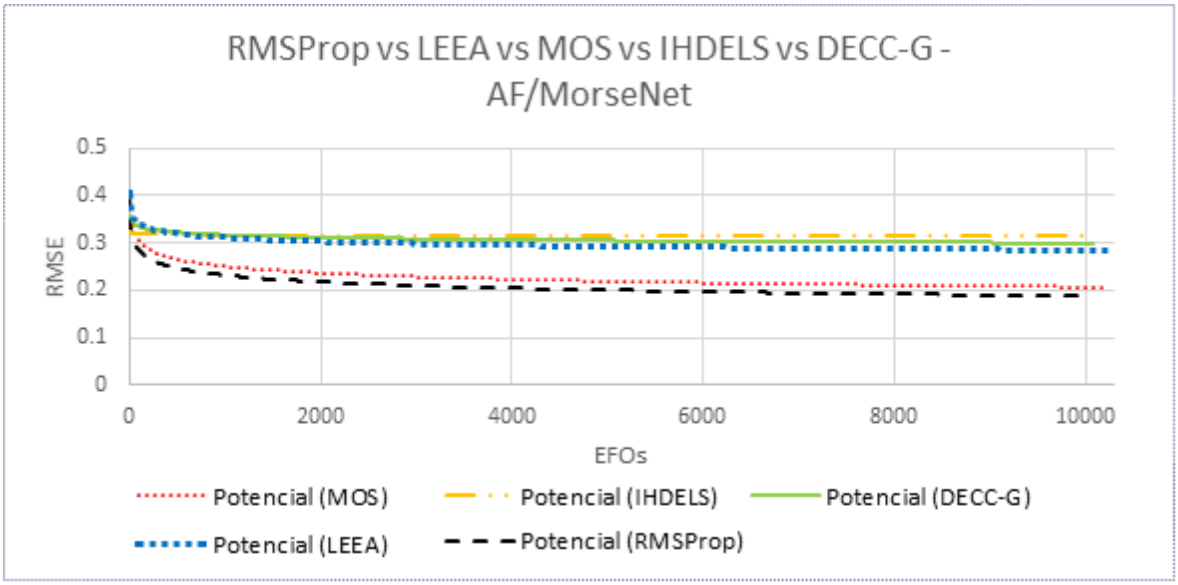

Fig. 8 Líneas de tendencia en la minimización del RMSE en Función de Aproximación utilizando MorseNet y los algoritmos: RMSProp, LEEA, MOS, IHDELS y DECC-G

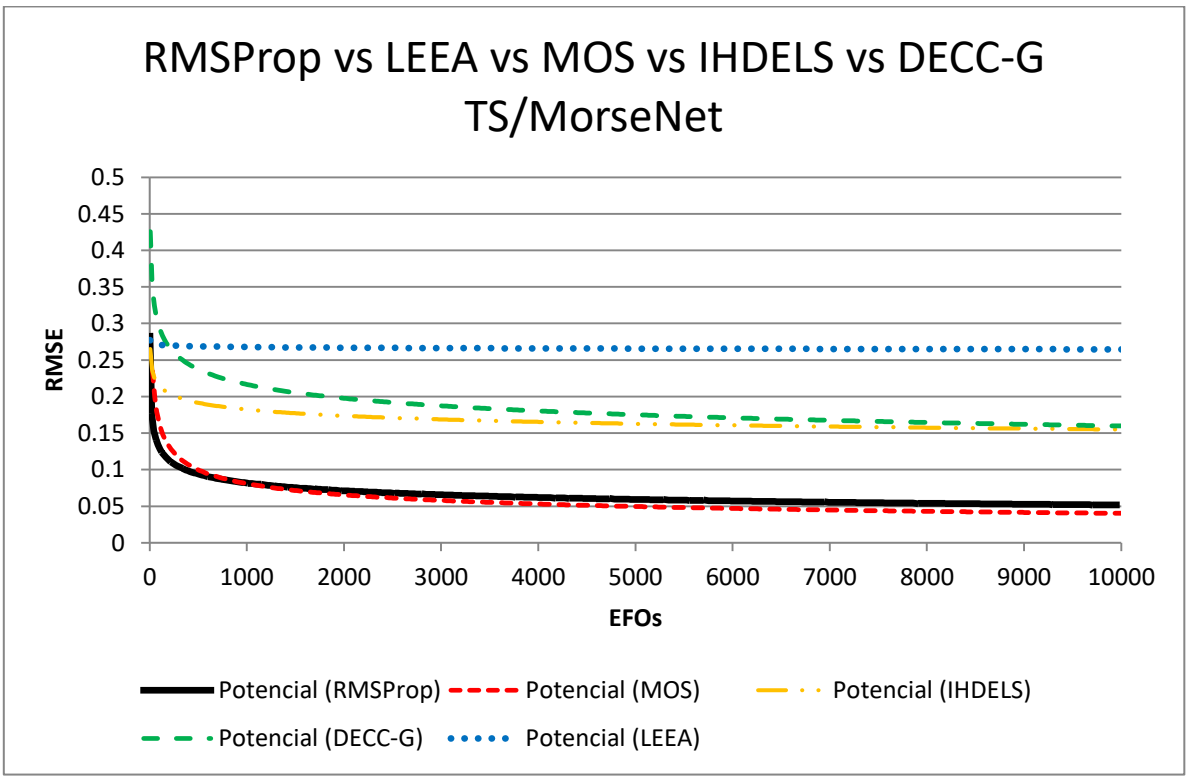

Fig. 9 Líneas de tendencia en la minimización del RMSE en Serie de tiempo Mackey-Glass utilizando MorseNet y los algoritmos: RMSProp, LEEA, MOS, IHDELS y DECC-G

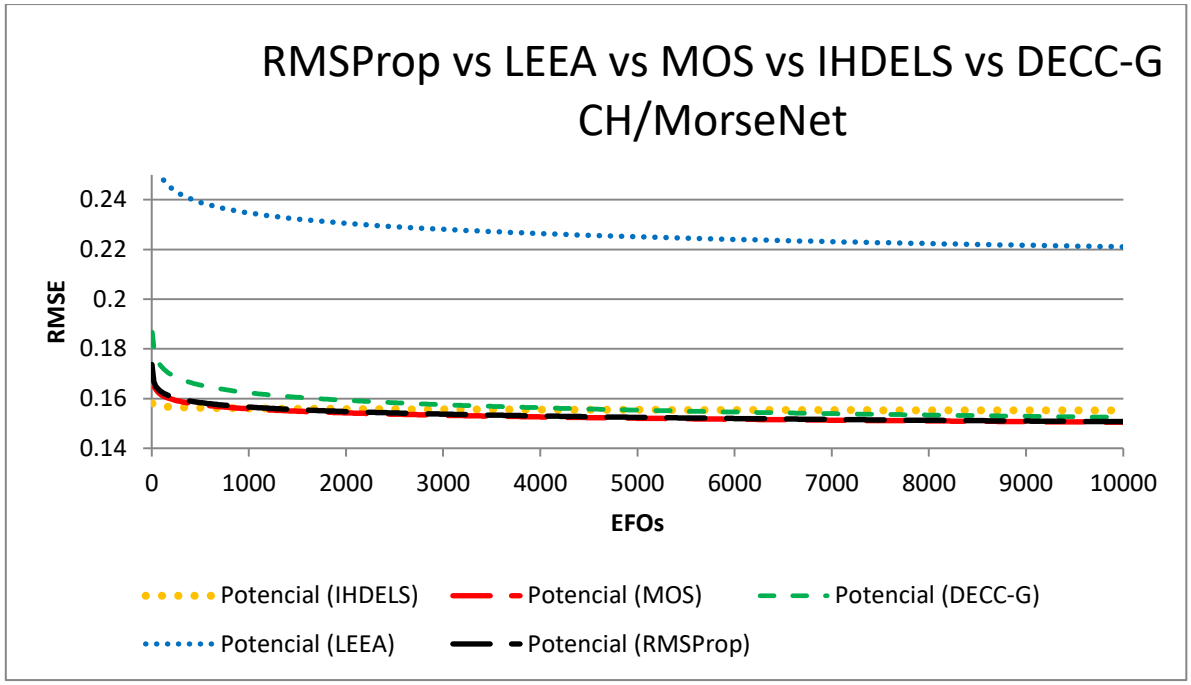

Fig. 10 Líneas de tendencia en la minimización del RMSE en California Housing utilizando MorseNet y los algoritmos: RMSProp, LEEA, MOS, IHDELS y DECC-G

En la práctica, cuando se entrena una red neuronal para que resuelva algún problema en específico, interesa obtener el mejor modelo, por ejemplo, el modelo que menor error cuadrático medio haya obtenido (Algoritmos de regresión) o el modelo que menor valor de entropía cruzada haya logrado alcanzar (Algoritmos de clasificación). Por consiguiente, en la **Tabla 1** se presenta el valor promedio del RMSE obtenido en la etapa de entrenamiento y en la etapa de prueba, así como, los valores de RMSE mínimos y máximos alcanzados en la época final del entrenamiento en cada experimento. Los resultados consolidados pertenecen todos a los algoritmos analizados en este capítulo: algoritmos del estado del arte: RMSProp y LEEA, y metaheurísticas de optimización global: MOS, IHDELS y DECC-G.

Tabla 1 Valores mínimos, máximos y promedios de RMSE en la última época para los algoritmos RMSProp, LEEA, MOS, IHDELS y DECC-G en los tres conjuntos de datos

Estadístico	Algoritmo	Función de aproximación	Serie de tiempo Mackey – Glass	California Housing
Mín(RMSE) en entrenamiento	RMSProp	**0.175576477**	**0.027080782**	**0.149481433**
	LEEA	0.25013737	0.243632113	0.218634911
	MOS	**0.175230948**	**0.031108956**	**0.149269165**
	IHDELS	0.273333299	0.085298779	0.151914822
	DECC-G	0.282451955	0.129022646	0.153161591
Máx(RMSE) en	RMSProp	**0.244856263**	**0.08820981**	**0.154537378**

entrenamiento	LEEA	0.289595602	0.271491546	0.223418819
	MOS	**0.257731173**	**0.081191934**	0.155271401
	IHDELS	0.328519402	0.294227796	0.208297473
	DECC-G	0.305356106	0.157285326	**0.154659447**
Mín(RMSE) en prueba	RMSProp	**0.098909155**	**0.003824568**	**0.036103602**
	LEEA	0.246375082	0.266418308	0.218199382
	MOS	**0.23405935**	**0.000662912**	0.09561087
	IHDELS	0.26189715	0.006178421	0.10093911
	DECC-G	0.24037607	0.012883222	**0.02366048**
Máx(RMSE) en prueba	RMSProp	0.7712916	**0.06940537**	0.22362854
	LEEA	**0.295992508**	0.280635442	0.225060087
	MOS	0.7821958	**0.04405988**	**0.20528789**
	IHDELS	0.46081537	0.2830842	0.4275972
	DECC-G	**0.44221678**	0.096665785	**0.034756616**
Prom(RMSE) en entrenamiento	RMSProp	**0.20547618**	**0.06184644**	**0.151392588**
	LEEA	0.270595937	0.25605232	0.221134518
	MOS	**0.223005818**	**0.046317043**	**0.15091076**
	IHDELS	0.31932467	0.176416751	0.156696823
	DECC-G	0.293094716	0.15150993	0.153942562
Prom(RMSE) en prueba	RMSProp	0.498754029	**0.017726586**	0.149831408
	LEEA	**0.269373722**	0.27400126	0.221299511
	MOS	0.497072045	**0.014486454**	**0.14700794**
	IHDELS	0.351124199	0.119236259	0.155749747
	DECC-G	**0.280320456**	0.046441449	**0.028105733**

En la **Tabla 1**, se observa en negrilla para cada algoritmo, el mejor valor de RMSE alcanzado, en los tres tipos de conjuntos de datos. En color rojo se detalla el segundo mejor valor de RMSE, esto con el fin de comparar cuáles de los algoritmos son los triunfadores en cada experimento. Se observa que en la mayoría de los casos el algoritmo MOS vence al RMSProp, por ejemplo, MOS alcanza el mínimo valor de RMSE en la función de aproximación y en California Housing, además logra el segundo lugar en la serie de tiempo durante la etapa de entrenamiento. El algoritmo LEEA no logra ser competitivo frente al RMSProp y MOS, esto es fácilmente explicable debido a que MOS es especialista en optimizar procesos en la alta dimensionalidad.

Con respecto a la etapa de prueba, MOS logra vencer al RMSProp en el conjunto de datos: Serie de tiempo y logra el segundo lugar en la función de aproximación. Sin embargo, en California Housing DECC-G vence claramente al RMSProp quien obtiene el segundo lugar en este conjunto de datos. Lo anterior demuestra que metaheurísticas de optimización global son capaces de vencer al RMSProp en los tres problemas de regresión.

Finalmente, cuando se observa los valores promedios del RMSE en el entrenamiento, el algoritmo MOS vence al RMSProp en la serie de tiempo y california Housing. Además, observando los valores promedios del RMSE en la etapa de prueba, RMSProp no alcanza a estar en la segunda posición en el conjunto de datos California Housing y función de aproximación, demostrando que LEEA y MOS logran vencerlo en dichos dominios, respectivamente.

A continuación, se realiza un análisis basado en la cuantificación de los valores RMSE obtenidos durante el entrenamiento y la prueba en rangos específicos. Esto logra mostrar cómo se comportan los algoritmos (RMSProp, LEEA, IHDELS, MOS y DECC-G) en las 30 repeticiones, permitiendo observar y comparar la distribución de los valores RMSE logrados al final del entrenamiento de la red neuronal.

La **Fig. 11** muestra para el problema de aproximación a una fusión, que el RMSProp tiene la mayor cantidad de respuestas de RMSE en rangos desde 0.16 hasta 0.24 incluidos, es decir, que el RMSProp durante sus 30 ejecuciones logra posicionar 29 resultados de RMSE en dicho rango. MOS es la metaheurística que logra tener la mayor cantidad de soluciones con RMSE en este umbral, exactamente 24. Además, se puede observar que la mayoría de las soluciones están cercanas a valores de RMSE de 0.32. IHDELS por otra parte es el que muestra peores (valores más altos) de RMSE.

La **Fig. 12**, contiene información similar a la explicada anteriormente, pero está relacionada con los valores RMSE obtenidos en la etapa de prueba. RMSProp logra posicionar el mejor valor de RMSE cercano a 0.16 seguido por el algoritmo MOS el cual coloca más soluciones con valores cercanos a 0.24, sin embargo, MOS logra tener más soluciones que RMSProp en umbrales con valores más bajos, por ejemplo, 0.32 y 0.4, respectivamente.

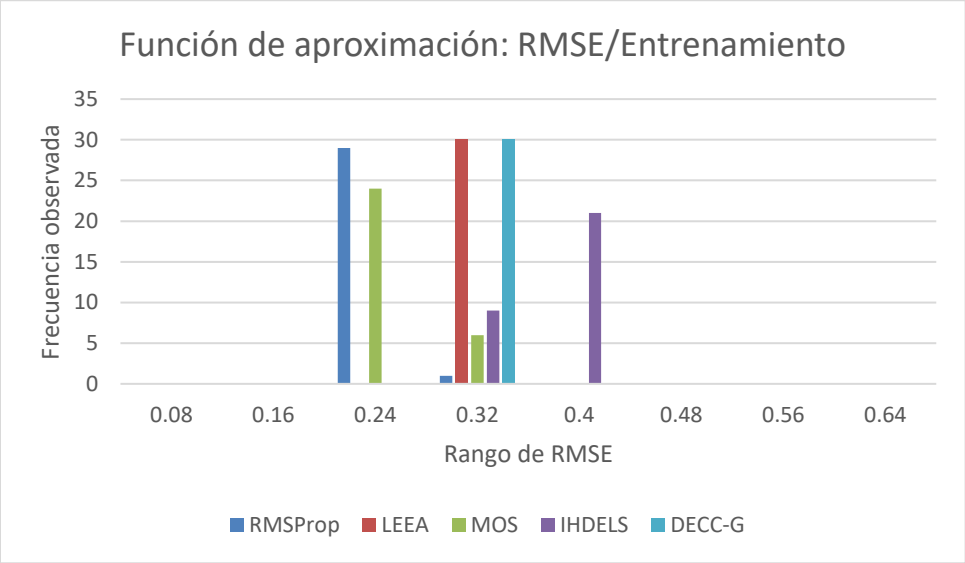

Fig. 11 Histograma acumulado de frecuencias para RMSE en la etapa de entrenamiento: Función de aproximación

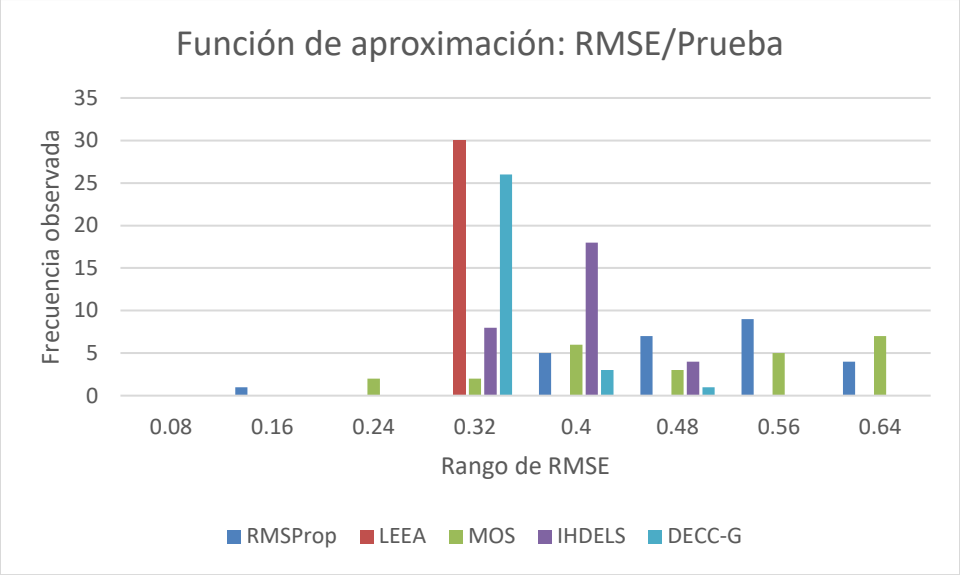

Fig. 12 Histograma acumulado de frecuencias para RMSE en la etapa de prueba: Función de aproximación

La **Fig. 13** y la **Fig. 14**, están relacionadas con el conjunto de datos de series de tiempo, en este dominio se observa que MOS tiene mayor cantidad de soluciones que el RMSProp en valores cercanos a 0.08, demostrando ser la mejor opción para afrontar este dominio. Nuevamente se observa en la **Fig. 14**, que las metaheurísticas de optimización global, logran soluciones en el mismo rango que el RMSProp en la etapa de prueba. Demostrando que son competitivas frente a este algoritmo del estado del arte.

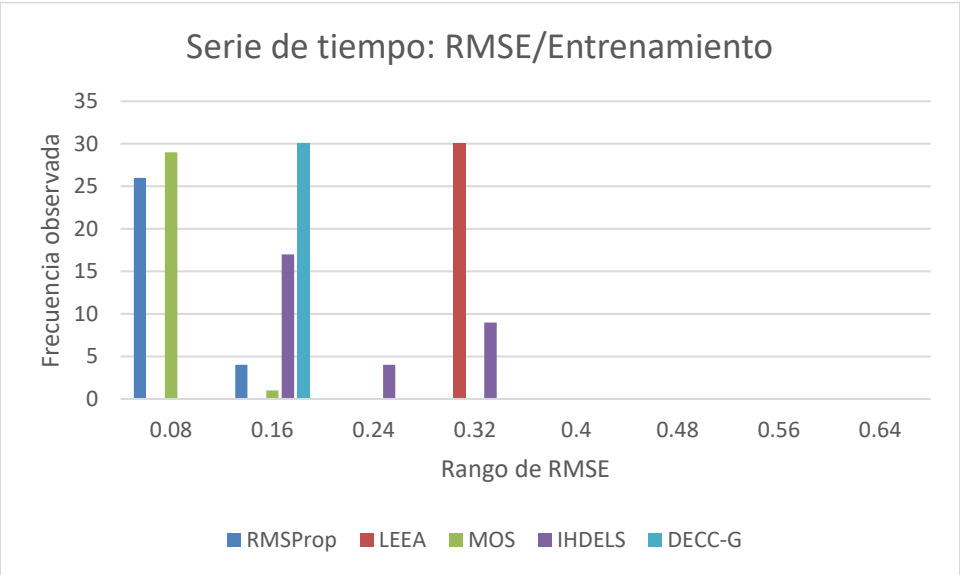

Fig. 13 Histograma acumulado de frecuencias para RMSE en la etapa de entrenamiento: ST Mackey-Glass

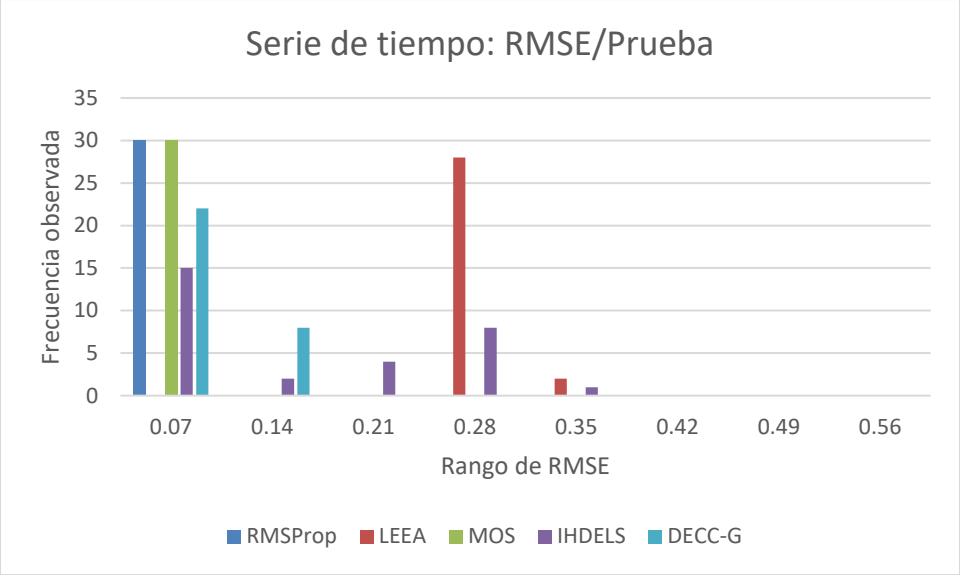

Fig. 14 Histograma acumulado de frecuencias para RMSE en la etapa de prueba: ST Mackey-Glass

Finalmente, se calculan los resultados de histogramas por rango para el dominio de California Housing. Este dominio tiene una mayor complejidad debido al aumento de características en el conjunto de datos. Las **Fig. 15** y **Fig. 16**, siguen demostrando que las metaheurísticas de optimización global logran alcanzar valores bajos de RMSE tal como lo hace el RMSProp, pero cabe aclarar que la cantidad de soluciones que aportan las metaheurísticas son mayores a lo logrado por el RMSProp, por ejemplo, en la **Fig. 15**, DECC-G e IHDELS logran tener mayor número de soluciones cercanas a 0.2 de RMSE. En la **Fig. 16**, se observa como DECC-G logra minimizar en todos los experimentos el RMSE mucho mejor que el RMSProp, aportando un total de 30 soluciones con valores cercanos al 0.08 a diferencia del RMSProp que solo obtiene en dos experimentos valores cercanos a este rango.

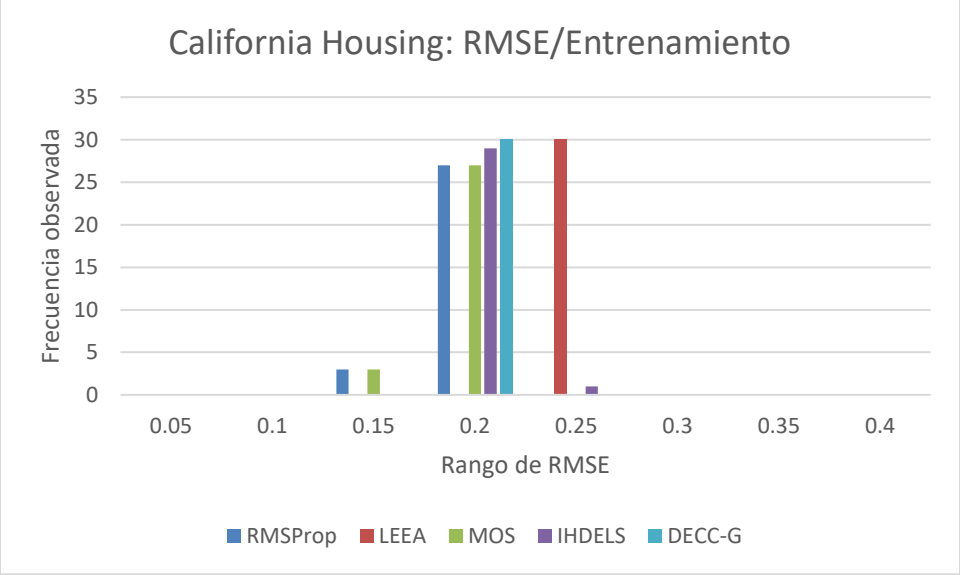

Fig. 15 Histograma acumulado de frecuencias para RMSE en la etapa de entrenamiento: California Housing

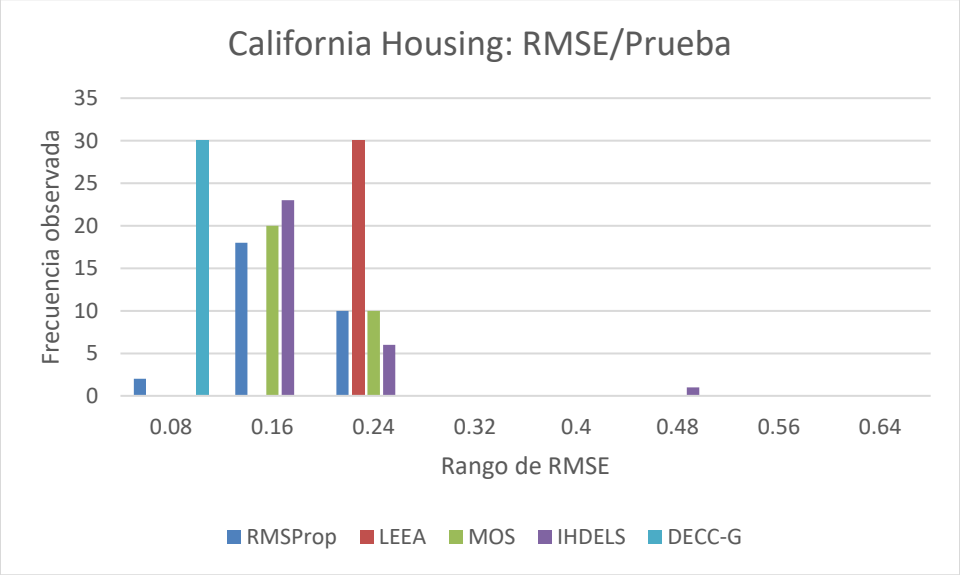

Fig. 16 Histograma acumulado de frecuencias para RMSE en la etapa de prueba: California Housing

Los resultados previos permiten afirmar que las metaheurísticas de optimización global de gran escala son una opción viable para la optimización del aprendizaje de redes neuronales profundas, que obtienen mejores resultados que LEEA, sin embargo, estos experimentos se hacen usando únicamente conjuntos de datos de regresión (estimación) y arquitecturas de redes, que, aunque son consideradas profundas, son convencionales.

Las redes neuronales convolucionales son arquitecturas que han demostrado solucionar problemas cada vez más cercanos y exclusivos a tareas humanas, su complejidad en el proceso de aprendizaje aumenta debido a la cantidad de filtros espaciales utilizados sobre las imágenes (principal domino para estas redes) por tal motivo y en aras de profundizar en esta investigación, se realizaron experimentos que consistieron en probar el mejor algoritmo de optimización global MOS sobre la arquitectura LeNet resolviendo el conjunto de datos de dígitos dibujados a mano MNIST, además, se implementaron nuevas propuestas meméticas que logran resolver con éxito este problema y vencer al RMSProp. Los resultados obtenidos se amplían en los siguientes capítulos.

Capítulo 4

4 Propuestas meméticas iniciales

Inicialmente para esta investigación se propuso implementar metaheurísticas de optimización global de gran escala para resolver tres dominios enfocados en tareas de predicción y aproximación como se mostró en el capítulo 3 de esta tesis, el objeto de estas implementaciones fue demostrar que algoritmos heurísticos enfocados en la alta dimensionalidad tendrían mejores resultados frente al algoritmo clásico de entrenamiento de redes neuronales profundas, en específico, RMSProp.

Los resultados permitieron observar que en dichos dominios el RMSProp tenía un buen rendimiento debido a que siempre optimiza una única solución, basado en información de la pendiente, mientras que, en las metaheurísticas al crear una población y evaluar cada solución un número determinado de épocas le permite a RMSProp tomar ventaja en la búsqueda del menor error cuadrático medio, es decir, que en 10000 épocas que por ejemplo dura el entrenamiento, RMSProp aprovecha mejor estas evaluaciones de la función objetivo en su única solución.

Por otra parte, aunque los conjuntos de datos tratados inicialmente en esta investigación tienen cierto grado de dificultad, no son considerados en el estado del arte, como conjuntos de datos apropiados para evaluar algoritmos de aprendizaje profundo, además, arquitecturas de redes neuronales convencionales como la presentada por Morse en [5] son fácilmente resueltas por los algoritmos clásicos, por consiguiente, se propone una nueva arquitectura de red neuronal profunda convencional descrita en [87] con más números de capas ocultas y de neuronas (por facilidad, en adelante se nombrará como una red **convencional profunda**) para resolver un conjunto de datos de imágenes relacionados con los dígitos escritos a mano (MNIST) los cuales hacen parte del estado del arte del aprendizaje profundo.

En este capítulo, se muestran los resultados obtenidos utilizando MNIST junto con la arquitectura de red neuronal profunda Convencional y un nuevo enfoque relacionado con una propuesta memética, logrando mejores resultados que el RMSProp.

El algoritmo del gradiente descendente estocástico (SGD), Adam, Adagrad, Momentum, RMSProp, entre otros, disponibles en TensorFlow son un esquema de búsqueda local que permiten llegar a mejores soluciones partiendo de una solución inicial generada aleatoriamente. Estos algoritmos de explotación trabajan sobre un único individuo y los resultados a la fecha en el entrenamiento de redes neuronales profundas son buenos, sin embargo, se han encontrado algunos problemas de estos algoritmos cuando el espacio de soluciones es muy grande, lo que generalmente ocurre en este contexto debido al incremento en el número de capas ocultas en las redes neuronales. Uno de los problemas de estos algoritmos, es que sólo realizan explotación y la exploración se hace de forma ingenua, ejecutando múltiples veces el algoritmo y escogiendo la mejor solución encontrada. Por otra parte, las metaheurísticas realizan al mismo tiempo las tareas de exploración y explotación sobre una solución o un conjunto de soluciones, lo que les

permite encontrar mejores soluciones en el mismo o menor tiempo. En este sentido, combinar lo mejor de los dos mundos realizando procesos de exploración a través de las metaheurísticas y optimizando los individuos de la población a través del uso de los optimizadores disponibles en Tensorflow permite que las metaheurísticas se conviertan en algoritmos memético, los cuales son el estado del arte hoy en día en un sinnúmero de problemas de optimización, en especial aquellos que son muy complejos y con alta dimensionalidad.

En este trabajo se realizó la modificación de 3 metaheurísticas poblacionales, a saber: La mejor búsqueda armónica global (Global-best Harmony Search, GBHS), Optimización por enjambre de partículas (Particle Swarm Optimization, PSO) y Evolución diferencial (Differential Evolution, DE). La modificación realizada sobre estas metaheurísticas consiste en adicionar un proceso de explotación sobre el mejor individuo de la población basado en optimizadores del gradiente descendente estocástico: RMSProp o Adam, disponibles en la librería de entrenamiento de redes neuronales profundas Tensorflow. Los seudocódigos de GBSH memético (**Seudocódigo 1**), PSO memético (Seudocódigo 2) y DE memético (Seudocódigo 3) se muestran a continuación.

En el **Búsqueda de la mejor armonía global meméticaSeudocódigo 1**, se observa la estructura del algoritmo GBHS memético, desde las líneas 1 a la 8 se ajustan parámetros como límites mínimos y máximos de las soluciones, tamaño de la población, creación de la población y su evaluación, índice que indica la posición de la mejor y peor solución. La ejecución del algoritmo con respecto a las evaluaciones de la función objetivo inicia en la línea 9, en la línea 10 comienza el recorrido por la población y desde las líneas 11 a la 16 se realiza la etapa de creación de una nueva solución por iteración teniendo en cuenta los parámetros específicos del algoritmo GBHS tales como HMCR y PAR. Desde las líneas 17 a la 24 se realiza el proceso de ingreso de la nueva solución siempre y cuando su valor de aptitud sea mejor con respecto a la peor solución de la población. En la línea 25, se aplica el proceso de explotación sobre la mejor solución. El algoritmo finaliza con la inclusión o no de la solución optimizada con la ayuda del RMSProp (Líneas 27 a la 34) y retorna el mejor individuo de la solución (Línea 35).

Seudocódigo 1 Búsqueda de la mejor armonía global memética (adaptado de [88]**)**

Entrada: Tamaño de la memoria armónica **HMS = 5**,
Número de iteraciones $\boldsymbol{N = 50}$,
Tasa de ajuste del tono $\boldsymbol{PAR = 0.35}$,
Tasa de consideración de la memoria armónica $\boldsymbol{HMCR > 0.9}$,
max_evals > 0
Problema a resolver $\boldsymbol{problema}$

Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
1:    efos = 0
2:    N = problema.size_pop
3:    LB = problema.LimInferior
4:    UB = problema.LimSuperior
5:    poblacion = crearPoblacionAleatoria(HMS)
6:    evaluar(poblacion) // y ordenar por fitness
7:    pos_best = 0
8:    pos_worst = HMS – 1
9:    mientras efos < max_evals hacer
```

```
10:       para j=1 hasta N
11:         si generarAleatorio() < HMCR entonces
12:           sol = escogerSolucionAleatoria()
13:           si generarAleatorio() < PAR entonces
14:             sol = escogerSolucion(pos_best)
15:         sino
16:           sol = crearSolucion(LB,UB)
17:         new_solution = sol
18:         evaluar(new_solution)
19:         efos = retornarEfos()
20:         si   new_solution.fitness   <   poblacion(pos_worst).fitness
              entonces
21:           borrar(poblacion(pos_worst))
22:           adicionar(poblacion,new_solution)
22:           ordenarPoblacion(poblacion)
23:         sino
24:           borrar(new_solution)
25:         mejor = AplicarRMSProp(poblacion(pos_best), n_epocas)
26:         efos = retornarEfos()
27:         si mejor.fitness < poblacion(pos_best).fitness entonces
28:           borrar(poblacion(pos_best))
29:           adicionar(poblacion,mejor)
30:           ordenarPoblacion(poblacion)
31:         sino
32:           borrar(mejor)
33        fin para
34:     fin mientras
35:     mejor = encontrarMejor(poblacion)
36:     retornar mejor
```

La optimización por enjambre de partículas (ver **Seudocódigo 2**) recibe inicialmente los parámetros relacionados con las diferentes soluciones que involucra el algoritmo: mejor local, mejor de los informantes y el mejor global. Crea una población de individuos (Línea 3), evalúa la población a través del entrenamiento de la red neuronal un número de épocas (Línea 7), cada partícula solución es optimizada utilizando el algoritmo RMSProp (Línea 10). Con la información proporcionada por el mejor de la población (Línea 12), el mejor de los informantes (Línea 13) y una partícula cualquiera (Línea 14), PSO crea un vector de velocidad basado en la ubicación de los tres individuos y unas constantes aleatorias (Línea 19). Junto con el vector posición y el vector velocidad crea un nuevo individuo (Línea 21), finalmente PSO retorna el mejor individuo de la población (Línea 22).

Seudocódigo 2. Optimización por enjambre de partículas memético (adaptado de [88]**)**

Entrada: tamaño del enjambre $\boldsymbol{swarmsize} > \mathbf{0}$,
Proporción de velocidad para ser retenida $\boldsymbol{\alpha}$,
Proporción del mejor personal para ser retenida $\boldsymbol{\beta}$,
Proporción del mejor de los informantes para ser retenida $\boldsymbol{\gamma}$,
Proporción del mejor global para ser retenida $\boldsymbol{\delta}$,
Tamaño del salto de una partícula $\boldsymbol{\varepsilon}$,

Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
1:    P <- {}
2:    Para i=0 hasta swarmsize
3:      P <- P U {nueva partícula aleatoria}
4:    Best <- None
5:    Para i = 0 Hasta Max_EFOS
6:       Para cada partícula x de P con velocidad v Hacer
7:          Evaluar(x)
8:          Si Best = None o x.fitness < Best.fitness Entonces
9:             Best <- x
10:            Best = aplicarRMSProp(Best,n_epocas)
11:      Para cada particula x ϵ P con velocidad v Hacer
12:         x* ← Ubicación previa mas adecauda de x
13:         x+ ← ubicación previa más adecuada de informantes de x
14:         x! ← Ubicación previa alguna particula
15:         Para cada dimensión i Hacer
16:            b ← numero aleatorio desde 0.0 a β inclusivo
17:            c ← numero aleatorio desde 0.0 a γ inclusivo
18:            d ← numero aleratorio desde 0.0 a δ inclusivo
19:            v_i ← αv_i + b(x_i* − x_i) + c(x_i+ − x_i) + d(x_i! − x_i)
20:      Para cada partícula x ϵ P con velocidad v Hacer
21:         x ← x + ev
22:   Retornar Best
```

El **Seudocódigo 3**, muestra el algoritmo para la evolución diferencial (DE). De manera similar a las propuestas meméticas de GBHS y PSO, la evolución diferencial se convierte en una versión memética en la línea 8 cuando el optimizador RMSProp es aplicado sobre la mejor solución de la población. DE memético inicia con la creación de la población (Línea 1), posteriormente, hasta que el número de evaluaciones de la función objetivo: épocas en la red neuronal, sea menor al máximo valor de las evaluaciones (Línea 3) realiza un proceso de reproducción (Línea 4) y reemplazo (Línea 5) sobre la población inicial, luego se encuentra el mejor individuo de la población (Línea 7) y como se explicó anteriormente se optimiza a través del algoritmo RMSProp (Línea 8). El algoritmo termina retornando la mejor solución de la población (Línea 11).

Seudocódigo 3. **Evolución diferencial memético (adaptado de** [89]**)**

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$,
Tasa de cruce $\boldsymbol{CR} > \mathbf{0}$,
Factor de escala $\boldsymbol{F} > \mathbf{0}$
Tamaño población $\boldsymbol{tamanoPoblacion} > \mathbf{0}$

Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
1:    población = crearPoblacionAleatoria(tamañoPoblacion)
2:    evaluaciones = tamañoPoblacion
3:    mientras evaluaciones < maxEvaluaciones hacer
4:      poblacionDescendencia = reproduccion(poblacion)
5:      poblacion = reemplazo(poblacionDesdendencia)
6:      evaluaciones = evaluaciones + tamañoPoblacion
```

```
7:        mejor = encontrarMejor(poblacion)
8:        mejor = AplicarRMSProp(mejor, n_epocas)
9:      fin mientras
10:     mejor = encontrarMejor(poblacion)
11:     retornar mejor
```

Con el fin de evaluar el comportamiento de las propuestas meméticas en la optimización de redes neuronales profundas, se realizaron dos experimentos, a saber: 1) Una comparación entre los optimizadores Adam y RMSProp disponibles en TensorFlow y las metaheurísticas implementadas usando solo una (1) época de optimización local, y 2) Una comparación de los resultados de la metaheurística con los mejores resultados frente a su versión memética, variando el número de épocas (intensificación) de la búsqueda local.

4.1 Problema

La experimentación se realizó buscando entrenar una red neuronal profunda con tres capas ocultas en la clasificación de dígitos escritos a mano alzada conocido como MNIST y cuyos datos están disponibles a través de TensorFlow. La arquitectura de la red se muestra en la **Tabla 2**. En los dos experimentos se busca minimizar la función de costo conocida como Entropía Cruzada (Cross Entropy). Se escoge esta arquitectura porque esta reportada en varios documentos con buenos resultados de exactitud y el proceso de entrenamiento no es costoso en tiempo, requiriendo aproximadamente 1000 iteraciones con el optimizador RSMProp.

Tabla 2. **Arquitectura de la Red Neuronal Profunda para el problema de clasificación MNIST**

Layer	Number of neurons	Activation function
First	200	Sigmoid
Second	100	Sigmoid
Third	60	Sigmoid
Fourth	30	Sigmoid
Fifth	10	Softmax

4.2 Resultados del Experimento 1

En GBHS se crea una población de 5 soluciones (HMS, Harmony Memory Size), se realizan 50 iteraciones (Number of Improvisations, N), Tasa de ajuste del tono (PAR, Pitch Adjusting Rate) se fija en 0.35 y tasa de consideración de la memoria armónica (HMCR, Harmony Memory Consideration Rate) en 0.9. En PSO, el tamaño de la población también es de 5, constante ponderada de inercia (constant inertia weight) igual a 0.5 y constantes cognitivas y sociales (cognative and social constants) equivalentes a 1 y 2 respectivamente. Para DE, la población inicial es de 5, tasa de mutación (mutate rate) igual a 0.5 y tasa de recombinación (recombination rate) se iguala a 0.7. Para la experimentación se tomaron los valores de los parámetros recomendados en la literatura, no se ha realizado por el momento un proceso de afinamiento de parámetros.

Al ejecutar los algoritmos, se almacenan los valores de la entropía cruzada cada cierto número de evaluaciones de la función objetivo (EFOs) y con ello luego se procede a hacer una curva de convergencia de la mejor solución encontrada hasta cierto número de EFOs. Para este experimento se ejecutaron los optimizadores Adam y RMSProp con un total de

1000 épocas, valor que permite obtener muy buenos resultados de entropía. Como se mencionó, las metaheurísticas realizan una búsqueda local sobre las soluciones utilizando optimizadores de Tensorflow, para este experimento se utiliza el optimizador Adam y se le permite realizar sólo una (1) época (intensificación de la búsqueda local) de optimización sobre las soluciones.

Con el objetivo de ser justos al momento de comparar las metaheurísticas con los optimizadores propios de Tensorflow, se define con la **Ecuación (4)** el número máximo de iteraciones que se realizan con las metaheurísticas poblacionales, para realizar las mismas 1000 evaluaciones de la función objetivo que se hacen con RMSProp y Adam.

$$NI = \frac{1000 - P_0 * NELS}{NEFOs} \quad (4)$$

Donde NI es el número de iteraciones que realiza la metaheurística poblacional (GBHS, PSO o DE), P_0 es la cantidad de soluciones en la población, NELS es el número de épocas en la búsqueda local y NEFOs es el número de EFOs realizadas dentro del bloque de iteraciones de cada metaheurística, así: GBHS usa 2 EFOs (una para la optimización local del nuevo improviso y otra para el mejor individuo de la población), PSO y DE realizan 5 EFOs (una por cada solución de la nueva población). Por lo tanto, 1000 épocas en Adam o RMSProp equivalen a 498 iteraciones (NI) para GBHS y a 199 para PSO y DE. En la **Fig. 17** se muestran la curva de convergencia de los algoritmos (Adam, RMSProp, PSO con Adam, DE con Adam, GBHS con Adam y GBHS con RMSProp) usando el promedio de 30 repeticiones del experimento.

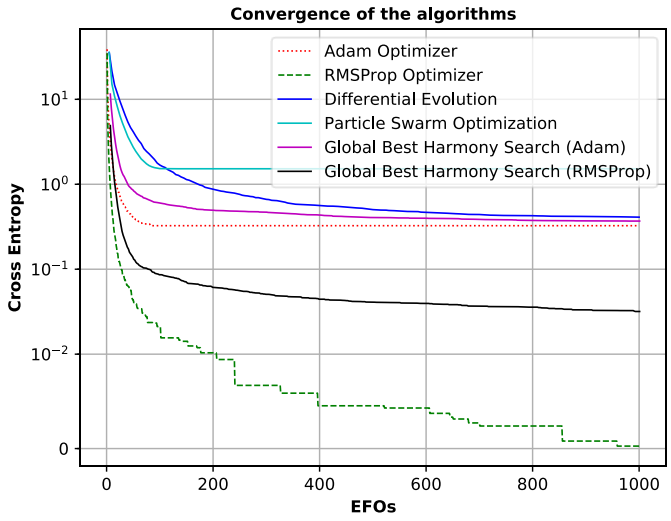

Fig. 17 Curvas de convergencia para 1000 Evaluaciones de la función objetivo entre optimizadores Adam, RMSProp y las metaheurísticas GBHS, PSO y DE

En la **Fig. 17** se observa como las metaheurísticas son un tanto competitivas frente al optimizador RMSProp, además cuando GBHS usa como optimizador local a RMSProp tiene mejores rendimientos que cuando usa el optimizador Adam. En la **Tabla 3**, se muestra el valor de la entropía promedio con diferentes valores de EFOs en este experimento.

Tabla 3. **Promedio de Cross Entropy (Minimizar) para distintas EFOs durante el proceso de entrenamiento de la red neuronal (los mejores resultados están en negrita)**

Algoritmo	EFOs			
	100	200	500	1000
RMSProp	**0.021136**	**0.010380**	**0.004532**	**0.000261**
GBHS/RMSProp	**0.086010**	**0.061008**	**0.040913**	**0.031804**
Adam	0.325254	0.325254	0.325254	0.325254
GBHS/Adam	0.599207	0.494281	0.406116	0.368744
DE/Adam	1.683368	0.873293	0.503006	0.410012
PSO/Adam	1.528210	1.522163	1.522163	1.522163

4.3 Resultados del Experimento 2

Este experimento busca analizar el impacto de variar la cantidad de épocas de entrenamiento (intensificación) en la búsqueda local dentro de GBHS (1, 2, 4, 8 y 20 épocas de entrenamiento). En este sentido, se busca encontrar el valor del parámetro de épocas de entrenamiento que mejor rendimiento le aporta a GBHS en su versión memética. Para lograr esto se presenta en la **Fig. 18** las curvas de convergencia de GBHS con sus 5 series de valores para las épocas de entrenamiento y su comparación frente a RMSProp.

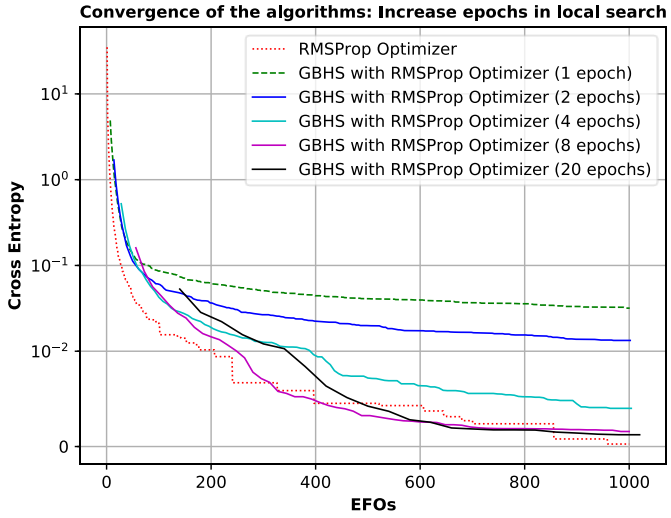

Fig. 18 Curvas de convergencia de GBHS con diferente número de épocas de entrenamiento (intensidad) en la búsqueda local.

En la **Fig. 18** se observa que conforme aumenta el número de épocas de entrenamiento (intensificación) en la búsqueda locales, la curva de convergencia de GBHS obtiene mejores resultados. En la **Tabla 4** se puede observar que el GBHS con 8 épocas en su búsqueda local supera a RMSProp cuando usa 420 EFOs, luego GBHS con 20 épocas se convierte hasta las 856 EFOs en el que obtiene menor valor, momento en el que RMSProp logra superar a las metaheurísticas hasta la EFO 1000. Cabe resaltar que el GBHS con 20 épocas sigue arrojando resultados competitivos frente a RMSProp y se

comporta como la configuración que mejor promedio de entropía conserva (color rojo) después de la EFO 620.

Tabla 4 Convergencia de la Entropía Cruzada en el optimizador RMSProp vs. GBHS/RMSProp con diferentes niveles de intensificación en la búsqueda local (1, 2, 4, 8 and 20 épocas)

Algoritmo	EFOs				
	100	420	620	856	1000
RMSProp	**0.021136**	0.004532	0.003730	**0.000794**	**0.000261**
GBHS/RMSprop (20 epochs)	0.053043	0.006345	0.002545	0.001514	0.001237
GBHS/RMSprop (8 epochs)	0.044487	0.004311	0.002552	0.001825	0.001578
GBHS/RMSprop (4 epochs)	0.042615	0.008954	0.006255	0.004889	0.004006
GBHS/RMSprop (2 epochs)	0.059845	0.021795	0.017199	0.014530	0.013361
GBHS/RMSprop (1 epoch)	0.086010	0.043396	0.038824	0.034029	0.031803

Finalmente, los resultados del entrenamiento se evalúan sobre el conjunto de datos de prueba disponible para el problema MNIST, los resultados basados en el experimento 2 se muestran en la **Tabla 5**.

En la **Tabla 5**, se puede observar cómo los valores de Precisión de GBHS son muy cercanas a los de RMSProp, con una disminución que varía entre el 0.1% y el 0.25%. Además, se observa que a medida que aumentan el número de épocas de entrenamiento (intensificación) en la búsqueda local la desviación estándar de los resultados tiende a cero. Lo anterior es prometedor y permite pensar que es posible seguir siendo competitivo con las metaheurísticas frente a los algoritmos clásicos del estado del arte en el entrenamiento de redes neuronales. Además, que se deben evaluar otras metaheurísticas que estén especializadas en problemas de optimización continuo de alta dimensionalidad con funciones multimodales que se parezcan al paisaje de las funciones de entrenamiento de las RNA profundas.

Tabla 5 Precisión del conjunto de prueba del problema MNIST utilizando los resultados de RMSProp y GBHS con 1, 2, 4, 8 y 20 épocas de búsqueda local

Algoritmo	Average Accuracy	Standard deviation
RMSProp	0.97637	0,00099
GBHS (20 epochs)	0.97538	0.00138
GBHS (8 epochs)	0,97476	0,00117
GBHS (2 epochs)	0,97463	0,00167
GBHS (4 epochs)	0,97449	0,00155
GBHS (1 epoch)	0,97389	0,00212

4.4 Multiple Offspring Sampling

Los resultados anteriores muestran lo prometedor que puede llegar a ser el método de crear propuestas meméticas que involucren a las metaheurísticas con la etapa de exploración y a los optimizadores del gradiente descendente en la etapa de intensificación o explotación. Por tales motivos se decidió modificar la metaheurística de optimización global de gran escala MOS (Multiple Offspring Sampling) repitiendo la metodología adoptada en el experimento 2 de este capítulo. MOS es una metaheurística que funciona

como juez entre dos o más algoritmos de optimización, dependiendo de los resultados buenos o malos en el momento de la optimización por parte de los algoritmos, MOS reparte un número mayor de EFOs siempre en función de premiar al mejor algoritmo. El seudocódigo de MOS se muestra a continuación en el **Seudocódigo 4**.

Los parámetros que recibe MOS están relacionados con la cantidad de evaluaciones de la función objetivo que se va a realizar (Línea 6), la máxima cantidad de EFOs que se tendrán en total para las técnicas que MOS esté arbitrando, este valor se reparte según el rendimiento de cada algoritmo subordinado y se controla por el radio de participación que maneja MOS (Línea 8), finalmente aplica los algoritmos sobre el individuo o población y retorna el mejor de todo el conjunto de soluciones.

Un factor clave de éxito de MOS para el problema en cuestión, consiste en definir las metaheurísticas que mejor combinen y mejores resultados obtengan en el proceso de optimización de la arquitectura de la red neuronal convencional profunda que se planteó al inicio de este capítulo. Si se estudia el algoritmo RMSProp, se puede observar que su idea original de acumular los errores a través de la etapa de entrenamiento se constituye en una idea sencilla, por lo que se plantea en la investigación encontrar metaheurísticas que tengan complejidad de funcionamiento baja pero que sean tan efectivas y rápidas como puede ser el RMSProp, así que se opta por utilizar las diferentes variaciones del ascenso de colina: básico (Hill Climbing, HC) y su versión de máxima pendiente (Steepest Ascent Hill Climbing, SAHC). HC se presenta en el **Seudocódigo 5** y SAHC se presenta en el **Seudocódigo 6**.

Seudocódigo 4. Multiple Offspring Sampling (MOS) (Tomado de [89]**)**

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$,
Evaluaciones por iteración $\boldsymbol{FE} > \mathbf{0}$,
Técnicas $\boldsymbol{tecnicas} = \{\boldsymbol{t_0}, \boldsymbol{t_1}, \dots, \boldsymbol{t_n}\}$
Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

1: `evaluaciones = 0`
2: `n = |tecnicas|`
3: $\forall\, j\ radio\_participacion_j = \frac{1}{n}$
4: `mejor = ejecutarTecnicas(radio_participacion, FE, mejor)`
5: `evaluaciones = evaluaciones + FE`
6: **`mientras`** `evaluaciones < maxEvaluaciones` **`hacer`**
7: $\forall\, j\ calidad_j = \frac{1}{|individuos_j|} \sum_{i=1}^{|individuos_j|} individuos_j[i].objetivo$
8: $\forall\, j\ radio\_participacion_j = PF(calidad_j)$
9: `mejor = ejecutarTecnicas(radio_participacion, FE, mejor)`
10: `evaluaciones = evaluaciones + FE`
11: **`fin mientras`**
12: **`retornar`** `mejor`

Seudocódigo 5. Algoritmo Ascenso a la colina. (adaptado de [88]**)**

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$
Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

1: `mejor <- solución candidata inicial`

```
2:    para i = 0 hasta maxEvaluaciones
3:       R <- AplicarRMSProp(Copiar(mejor),n_epocas)
4:       si R.fitness < mejor.fitness entonces
5:          mejor <- R
6:    fin para
7:    retornar mejor
```

En el ascenso de colina memético básico, la solución es optimizada utilizando el RMSProp un número determinado de épocas (Línea 3) para finalmente retornar la solución con el menor valor de entropía cruzada (Líneas 4 a la 7).

Seudocódigo 6 Algoritmo Ascenso a la colina por la máxima pendiente. (Adaptado de [88]**).**

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$

Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
1:    n <- cantidad de ajustes deseados para muestrear el gradiente
2:    mejor <- solución candidata inicial
3:    para i = 0 hasta maxEvaluaciones
4:       R <- AplicarRMSProp(Copiar(mejor),n_epocas)
5:       para i = 0 hasta n-1
6:          W <- AplicarRMSProp(Copiar(mejor),n_epocas)
7:          si W.fitness < R.fitness entonces
8:             R <- W
9:          si R.fitness < mejor.fitness entonces
10:            mejor <- R
11:   retornar mejor
```

En el algoritmo de ascenso de colina memético por la máxima pendiente, las optimizaciones basadas en el RMSProp se realizan tanto en la solución inicial (Línea 2), así como, en sus vecinos (Línea 6). Retornando la solución con un menor valor de entropía cruzada (Líneas 7 a la 11).

Una vez seleccionados los algoritmos se procede a ejecutar el MOS con los dos algoritmos subordinados, el RMSProp y HC o SAHC, el objetivo de mantener al RMSProp radica en que al ser un método basado en gradiente es muy adecuado para la explotación. Los resultados de las diferentes comparaciones con las propuestas meméticas se observan en la **Fig. 19**.

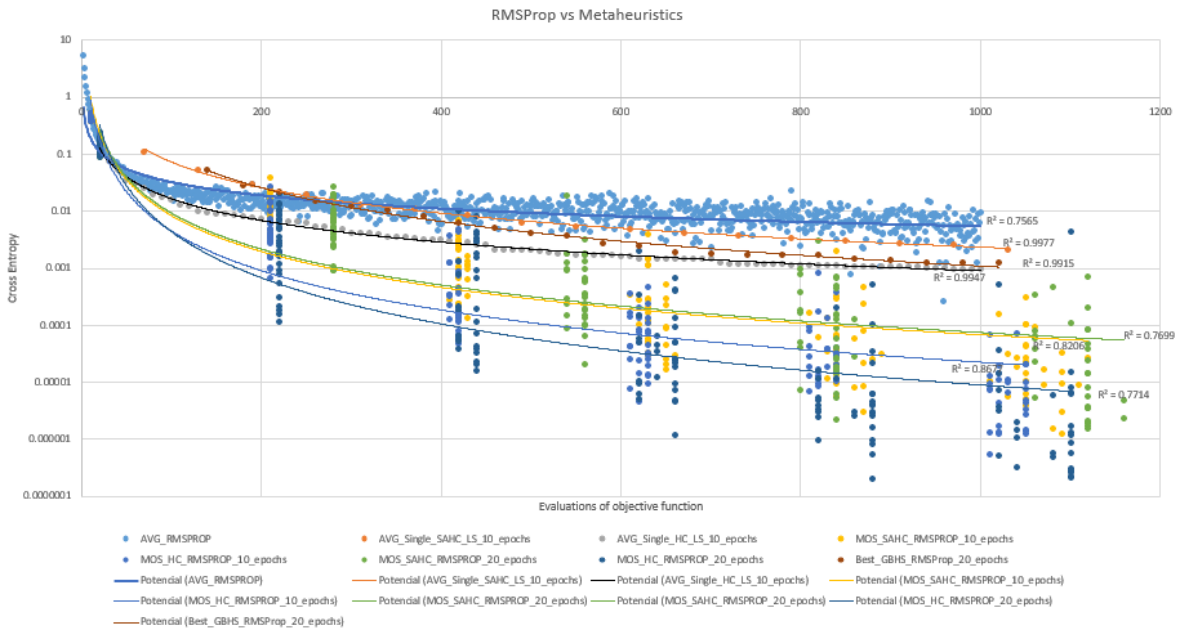

Fig. 19 Curvas de convergencia de la entropía cruzada: RMSProp vs MOS(HC & RMSProp), MOS(SAHC & RMSProp), HC, SAHC, Best GBHS

Los resultados de la **Fig. 19** muestran como todas las propuestas meméticas logran superar por amplio margen al RMSProp en la optimización de la arquitectura de red neuronal convencional profunda y el conjunto de datos MNIST. Cabe destacar que la propuesta memética basada en MOS con subordinados HC y RMSProp logran obtener los mejores valores de entropía cruzada durante las 1000 épocas que duró el entrenamiento.

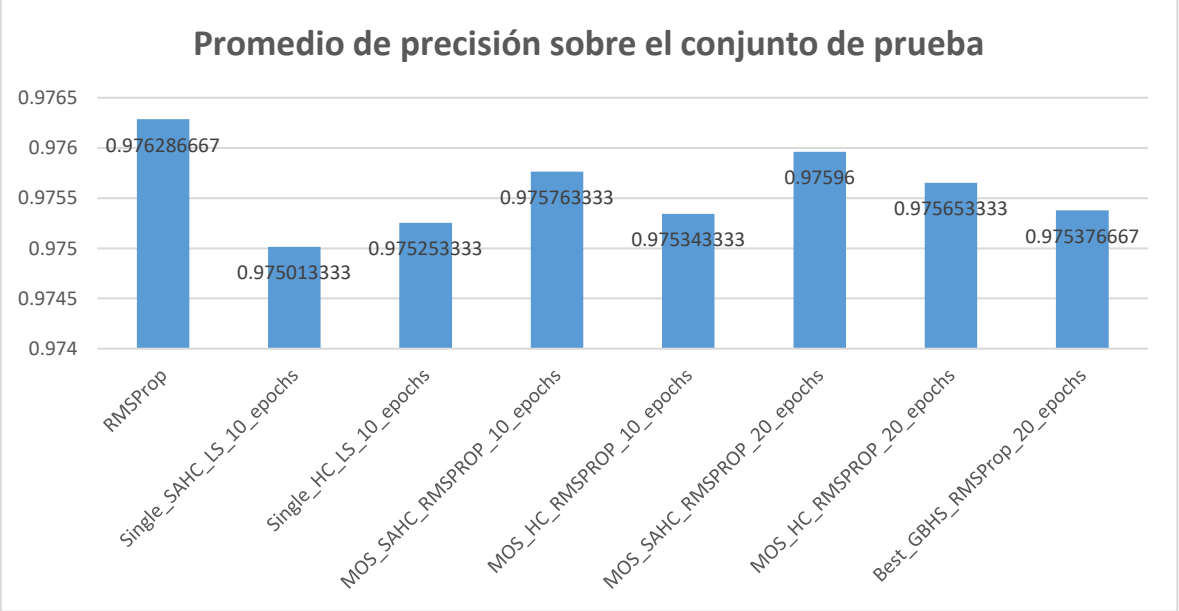

Fig. 20 Valor promedio de la precisión sobre 30 experimentos para cada propuesta memética vs el RMSProp en el conjunto de datos MNIST

En la **Fig. 20** se observa que todas las propuestas meméticas tiene un promedio de precisión sobre el conjunto de prueba comparables al obtenido por el RMSProp, hasta el momento con estos enfoques meméticos se ha logrado vencer claramente al RMSProp en la etapa de entrenamiento y los resultados en la etapa de prueba difieren en el orden de milésimas, la ventaja es que aún existen muchas más opciones de mejora de las metaheurísticas, una mejora se muestra en el **Capítulo 6** de esta tesis.

Debido a que durante el transcurso de la investigación se observó que diversos enfoques fueron implementados de forma aislada y además, haciendo una revisión del estado del arte sobre frameworks en neuroevolución no se encontró herramientas que facilitaran el trabajo de implementación entonces se decidió crear un marco de trabajo que facilite a los investigadores y entusiastas en el área de la neuroevolución a generar nuevas propuestas que logren ser competitivas frente a los algoritmos tradicionales de optimización del aprendizaje de redes neuronales profundas. A continuación, el **Capítulo 5** define y explica el Deep Neuroevolution Framework (DNF) desarrollado en esta tesis.

Capítulo 5

5 Framework para el diseño y experimentación de metaheurísticas que soportan el entrenamiento de Deep Neural Networks

5.1 Descripción de Deep Neuroevolution Framework (DNF)

La neuro evolución se constituye en un enfoque prometedor en la optimización del entrenamiento de redes neuronales profundas. Esta investigación presenta a Deep Neuroevolution Framework, el primer framework orientado a objetos, desarrollado en Python y con una estructura orientada a objetos altamente cohesiva y débilmente acoplada, que combina el potencial de exploración/explotación probabilística de los algoritmos evolutivos (EA) y los algoritmos de explotación basados en SGD disponibles en TensorFlow, aprovechando con esto además la capacidad de realizar procesamiento paralelo basado en GPUs en TensorFlow.

La fácil inclusión de metaheurísticas diseñadas para la optimización del aprendizaje en Deep Learning (DL) pretende generar un aporte en el campo de la neuro evolución impulsando nuevos desarrollos y soluciones al proceso de entrenamiento, puesto que, como lo afirman Chiroma et al. [90] los algoritmos inspirados en la naturaleza no están siendo aprovechados en DL. Por otra parte, se espera que DNF permita una mejor sinergia entre las comunidades de los EA y DL para seguir mejorando la calidad de los resultados obtenidos por DL en las diversas tareas donde se está usando y en otras donde puede ser usado.

DNF está compuesto por cinco clases principales: metaheurística, problema, dataset, diseñador de red (neuronal) y solución, y tiene la posibilidad de crecer en metaheurísticas y problemas (dataset y arquitecturas de red neuronal que se usan para resolver los problemas). Actualmente, DNF tiene implementados tres problemas cada uno con diferentes arquitecturas de red que se pueden usar para resolverlo. Los datasets son MNIST, FASHION-MNIST y CIFAR-10. Incluye seis metaheurísticas clásicas de optimización continua: PSO, GBHS, DE, Hill Climbing (HC), Steepest Ascent Hill Climbing (SAHC) y Steepest Ascent Hill Climbing with Replacement (SAHCwR), y tres metaheurísticas de optimización global de gran escala: Iterative Hybridization of Differential Evolution with Local Search (IHDELS)[81], Self-adaptive differential evolution (SaDE)[91] y Multiple Offspring Sampling (MOS)[92]. Estas metaheurísticas se hibridan incluyendo explotación basado en SGD con los algoritmos disponibles en TensorFlow, entre ellos: RMSProp, Adam, Adagrad, Adadelta y Momentum. Es preciso mencionar que DNF no cuenta con la implementación de todos los EAs clásicos (los cuales se pueden incluir fácilmente), la razón de esto se centra en dar prioridad a nuevos desarrollos en EAs de optimización global a gran escala (alta dimensionalidad) [72] que sean apropiados para optimizar el entrenamiento de DL por sí solos o hibridados con algoritmos de optimización local basado en SGD.

Por otra parte, DNF incluye tres datasets que son de baja a mediana complejidad en el campo del DL. Esto no limita la inclusión de otros datasets de mayor complejidad y tamaño, como ImageNet (1000 clases, 14197122 imágenes, más de 2 TB de almacenamiento, disponible en http://www.image-net.org), tarea que es fácil de llevar a cabo pero que no se realizó debido a limitaciones de espacio de almacenamiento en el repositorio de GitHub. A continuación, en la Tabla 6, se presenta una comparación de los frameworks mencionados teniendo en cuenta 9 características de interés: disponibilidad de algoritmos evolutivos tradicionales (Traditional Evolutionary Algorithms, TEA), soporte para diseño de redes neuronales artificiales (DoANN), estilo de programación, disponibilidad de algoritmos de optimización global a gran escala (AOGS), disponibilidad de optimizadores del SGD (SGD Optimizers, SGDop), lenguaje de programación, Soporte GPU/CPU, Nivel de API and Arquitectura.

Tabla 6 Comparación de las principales características de los frameworks

Framework	Heurísticas		Deep Learning			Neuroevolución
Características	MOEA	jMetal	TensorFlow	Pytorch	Keras	DNF
TEA	20 o más	20 o más	Ninguno	Ninguno	Ninguno	9
DoANN	Ninguno	Ninguno	Si	Si	Si	Si
Estilo de programación	Object-Oriented Programming (OOP)	OOP	Scripting	Scripting	Scripting/OOP	OOP
LSGO	Ninguno	Ninguno	Ninguno	Ninguno	Ninguno	4
SGDop	Ninguno	Ninguno	8	11	8	8
Lenguaje de programación	Java	Java	Python	LUA, Python	Python	Python
Soporte GPU/CPU	CPU	CPU	CPU/GPU	CPU/GPU	CPU/GPU	CPU/GPU
Nivel de API	Alto y bajo nivel	Alto y bajo nivel	Bajo y alto nivel	Bajo nivel	Alto nivel	Alto y bajo nivel
Arquitectura	Simple y fácil de usar	Simple y fácil de usar	Complejo y no fácil de usar	Complejo perola legibilidad es buena	Simple y fácil de usar	Simple y fácil de usar

5.2 Descripción de Software

5.2.1 Arquitectura Software

En la **Fig. 21** Se presenta el diagrama conceptual de clases de DNF, donde se destacan cinco clases principales (Fondo café claro en **Fig. 21**):

- **Metaheuristic class:** permite la implementación de diferentes metaheurísticas (Fondo amarillo claro en **Fig. 21**) para optimizar las arquitecturas de redes neuronales profundas con distintos conjuntos de datos.
- **Solution class:** esta clase se relaciona con la clase Metaheuristic. Permite crear individuos o soluciones (Fondo café claro en **Fig. 21**) con una representación vectorial unidimensional (clásica en Metaheurísticas) que gestiona los pesos y bias de la red neuronal profunda como por ejemplo la partícula de PSO que incluye velocidad, mejor solución encontrada por la partícula y calidad de esa mejor solución encontrada (Fondo gris claro en **Fig. 21**).
- **NetworkDesigner class:** Clase abstracta que permite diseñar y crear a través de la instanciación de clases heredadas cualquier tipo de red neuronal profunda (Fondo azul claro en **Fig. 21**).
- **Dataset class:** clase abstracta que permite la carga de los conjuntos de datos (Fondo rojo claro en **Fig. 21**).
- **ProblemBuilder class:** es una clase abstracta que permite unir una red neuronal profunda heredada de la clase **NetworkDesigner** con un conjunto de datos heredados de la clase **Dataset** para definir un problema concreto de optimización a resolver (Fondo verde claro en **Fig. 21**). La clase ProblemBuilder permite utilizar los distintos optimizadores del gradiente descendente disponibles en TensorFlow.

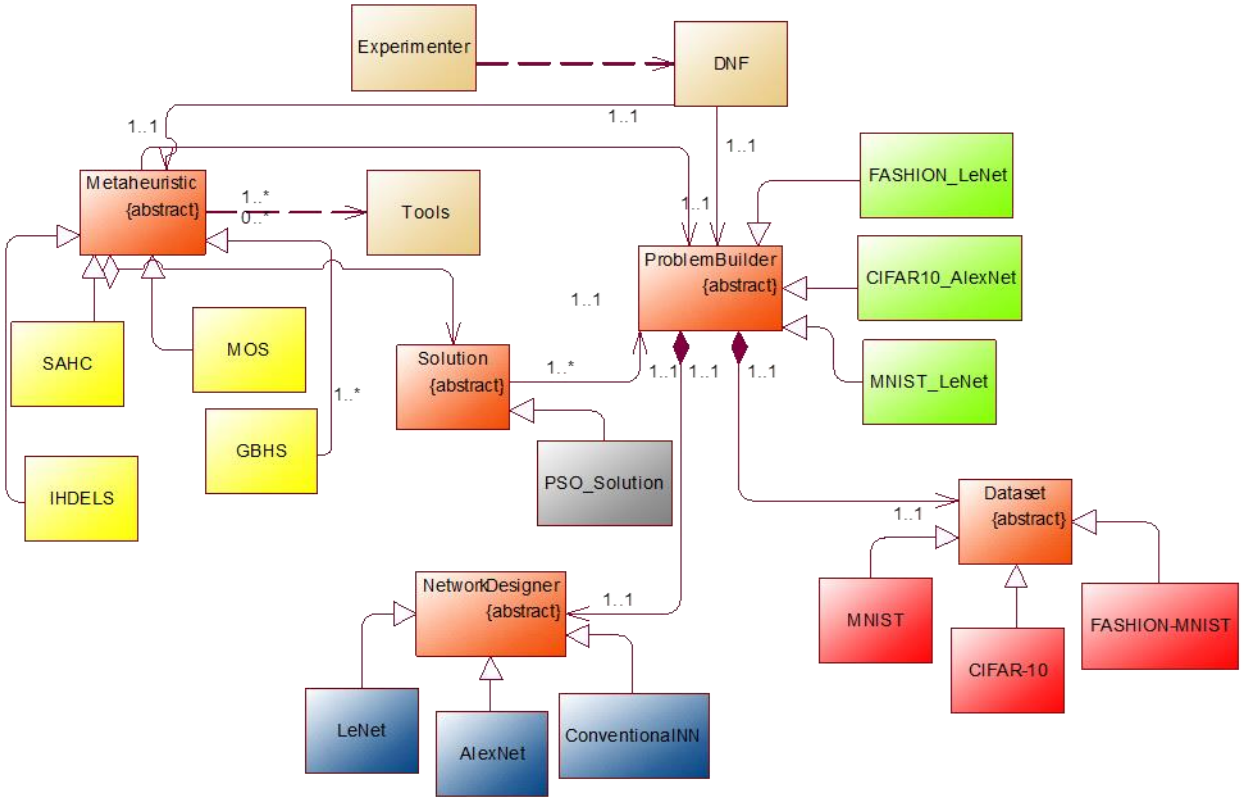

Fig. 21 Arquitectura de DNF

5.2.2 Funcionalidades Software

Step 1 - Cargue del conjunto de datos: DNF pone a disposición una clase abstracta llamada **Dataset**. Esta clase usa funciones de TensorFlow (e.g. **tf.data**) que le permiten

alimentar el proceso de entrenamiento de redes neuronales con el conjunto de datos que se quiera trabajar. La clase retorna a través de la función **loadData** las muestras de entrenamiento y testing del dataset.

Step 2 - Diseño de la red neuronal: a través de la clase abstracta **NetworkDesigner**, DNF permite diseñar cualquier tipo de arquitectura de red neuronal profunda utilizando funciones de TensorFlow, la función **model** de esta clase, retorna el modelo creado mediante una variable que contiene el diseño de los diferentes tipos de capas: ReLU, convolutions y pooling. Además, si es requerido, es posible apagar ciertas neuronas durante el entrenamiento utilizando el Dropout con el objetivo de evitar el sobre ajuste. El resultado retornado por la función **model** es un tensor con el diseño detallado de la red. En DNF se proporcionan redes neuronales convolucionales trabajadas en el estado del arte (LeNet y AlexNet), así como redes neuronales convencionales con un mínimo de 4 capas ocultas.

Step 3 - Definir el problema: A través de la clase abstracta ProblemBuilder, DNF permite crear clases concretas con las que se pueda usar cualquier dataset (ya disponible o nuevo) cargado a través de la clase Dataset y cualquier arquitectura de red neuronal profunda (ya disponible o nueva) diseñada a partir de la clase NetworkDesigner. ProblemBuilder consta de tres métodos: train_eval_test encargado de realizar el entrenamiento de la red utilizando los pesos y bias de las soluciones que componen la población, dicho entrenamiento se realiza utilizando una sesión de Tensorflow y una función de perdida, e.g. la entropía cruzada. Por otra parte, el método recibe un entero que representa el número de épocas que se requiere entrenar, este parámetro le permite a DNF ejecutar las siguientes opciones según el valor de este parámetro: 1) entrenamiento cuando training_epochs > 0, 2) evaluación de una solución de la población obteniendo el valor promedio de la entropía cruzada cuando training_epochs = 0 y 3) evaluación del entrenamiento final con los pesos y bias obtenidos sobre el conjunto de testing cuando training_epochs = -1. Finalmente, train_eval_test recibe un tipo de dato flotante que representa la tasa de aprendizaje que se manejará durante el entrenamiento, este parámetro permite brindarle al framework la posibilidad de que cada solución que compone la población pueda ser inicializada y modificada de manera diferente, mejorando los procesos de exploración durante la búsqueda del óptimo global. El método **merge_wb** permite unir los diferentes pesos y bias de cada capa de la red neuronal en un vector unidimensional y el método **split_wb** divide el vector unidimensional en los peos y bias de cada capa de la red neuronal para poder asignarlos en el grafo de TensorFlow.

Step 4 - Definir la solución: esta clase recibe como parámetro un objeto de la clase **ProblemBuilder** en su constructor, esto le permite conocer previamente el tamaño y rango de datos para su inicialización y trabajar la solución durante el proceso de entrenamiento de la red neuronal profunda. DNF ha adoptado una representación clásica y simple de las soluciones, consistente en un vector unidimensional el cual está compuesto por los pesos y los bias de las diferentes neuronas y capas de la red. La clase **Solution** se compone de 6 métodos, algunos tienen un rol principal mientras que otros están diseñados para suplir requerimientos simples que logran que DNF tenga una estructura débilmente acoplada. Entre los métodos principales se encuentran: **random_initialization** encargado de inicializar las soluciones de la población teniendo en cuenta los rangos mínimos y máximos en los cuales se moverán los peos y bias de las soluciones, además, permite propagar hacia adelante en la red neuronal los valores de dicha solución con el objetivo de obtener el valor de aptitud de cada solución (e.g. Cross Entropy). Los métodos **eval_fitness** y **eval_testing_data** se relacionan con métodos de la clase **ProblemBuilder** que permiten entrenar la red con un número de épocas, pesos y

bias específicos y evaluar el entrenamiento final sobre un conjunto de entrenamiento o testing, respectivamente. En los dos casos, los métodos retornan el valor de aptitud de la solución, el número de evaluaciones de la función objetivo (EFOs) utilizadas, el valor de la pendiente durante una ventana o buffer de entropías y el vector de pesos y bias de las capas de la red. Algunos métodos adicionales incluyen: **assign_values** que permite asignar a una instancia de la clase **Solution**, los valores de pesos y bias específicos expresados en un vector unidimensional, **copy_from** que copia atributos de la clase **Solution** de un objeto solución a otro y **copy_some_variables** que copia únicamente los pesos y bias de una solución. Existen metaheurísticas como PSO que requieren adicionar atributos a esta clase, como la velocidad y la posición de sus soluciones, DNF está diseñado para que la clase **Solution** pueda ser heredada y así lograr una independencia del tipo de solución que se requiera en cada metaheurística.

Step 5 - Definir la Metaheurística: la clase **Metaheuristic** es la responsable de permitir implementar cualquier metaheurística sobre DNF, esta clase abstracta está compuesta por un único método: **run**, este método solo recibe el nombre donde se guardarán los resultados del entrenamiento. La clase **run** conecta los procesos evolutivos con el entrenamiento de la red neuronal profunda en Tensorflow, esta característica se logra teniendo en cuenta dos aspectos a la hora del diseño de una nueva metaheurística: 1) que el individuo sea una instancia de la clase **Solution** o de una clase que herede de **Solution** y que su representación corresponda a un vector unidimensional y 2) Que permite a través de archivos de configuración ajustar diferentes parámetros como: cantidad de soluciones en la población, tasa de aprendizaje, número de épocas y cantidad de evaluaciones de la función objetivo en los procesos de exploración y explotación.

Step 6 - Definir el experimento: DNF pone a disposición un archivo llamado **Experimenter** el cual facilita al usuario diseñar un experimento que relaciona la metaheurística que se usará para resolver un problema ya disponible en el framework. La ejecución del experimento se realiza por consola y debido a que DNF es un framework de neuro evolución, **Experimenter** permite la ejecución de un experimento durante un número determinado de veces (repeticiones), inicialmente DNF se encuentra configurado para que ejecute 30 veces con diferentes semillas de números aleatorios con el objetivo de que los resultados promedios de estas repeticiones cumplan con el teorema del límite central.

5.2.3 Ejemplo ilustrativo

A continuación, se ilustra cómo se implementó la metaheurística Steepest Ascent Hill Climbing [93] modificando el proceso de explotación utilizando el RMSProp (RMSPropSAHC) para la optimización del aprendizaje de la red neuronal convolucional profunda LeNet [94] utilizando el dataset MNIST [95].

Step 1 - Cargue del conjunto de datos: La **Fig. 22** muestra la implementación de la clase MNIST que hereda de la clase Dataset. Dentro de esta clase se debe implementar el método loadData que retorna los tensores con los datos de entrenamiento y testing (línea 12).

Step 2 - Diseño de la red neuronal: La **Fig. 23** muestra el método **model** implementado para la clase LeNet (ya implementada en DNF) que en su definición hereda de la clase NetworkDesigner. Esta red está compuesta por tres capas convolucionales (líneas 137, 142 y 147), sobre cada una de estas capas se define la función de activación ReLU (líneas 138, 143 y 148) además realiza un proceso de pooling sobre las dos primeras capas convolucionales con el fin de reducir la resolución de las imágenes (líneas 139 y 144), dos capas fully-connected que se ubican en la etapa final de la arquitectura LeNet

(líneas 150 y 157) encargadas del proceso de clasificación, finalmente, con el objetivo de evitar el sobreajuste en la líneas 140, 145, 152 y 155 se aplica un proceso de dropout sobre cada capa que compone la red.

```
1   from Dataset import *
2   from tensorflow.examples.tutorials.mnist import input_data
3   class MNIST(Dataset):
4       def __init__(self):
5           Dataset.__init__(self)
6       def loadData(self):
7           self.data = input_data.read_data_sets("MNIST_data/mnist", one_hot=True)
8           trX, trY, teX, teY = self.data.train.images, self.data.train.labels, self.data.test.images, \
9                                self.data.test.labels
10          trX = trX.reshape(-1, self.img_size, self.img_size, 1)
11          teX = teX.reshape(-1, self.img_size, self.img_size, 1)
12          return trX, trY, teX, teY
```

Fig. 22 Desarrollo de la clase MNIST para la carga de datos

```
136     def model(self, X, w, w2, w3, w4, w_o, p_keep_conv, p_keep_hidden):
137         convolutional_1 = tf.nn.conv2d(X, w, strides=[1, 1, 1, 1], padding='SAME')
138         convolutional_1_a = tf.nn.relu(convolutional_1)
139         convolutional_1 = tf.nn.max_pool(convolutional_1_a, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
140         convolutional_1 = tf.nn.dropout(convolutional_1, p_keep_conv)
141
142         convolutional_2 = tf.nn.conv2d(convolutional_1, w2, strides=[1, 1, 1, 1], padding='SAME')
143         convolutional_2_a = tf.nn.relu(convolutional_2)
144         convolutional_2 = tf.nn.max_pool(convolutional_2_a, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
145         convolutional_2 = tf.nn.dropout(convolutional_2, p_keep_conv)
146
147         convolutional_3 = tf.nn.conv2d(convolutional_2, w3, strides=[1, 1, 1, 1], padding='SAME')
148         convolutional_3 = tf.nn.relu(convolutional_3)
149
150         FC_layer = tf.nn.max_pool(convolutional_3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
151         FC_layer = tf.reshape(FC_layer, [-1, w4.get_shape().as_list()[0]])
152         FC_layer = tf.nn.dropout(FC_layer, p_keep_conv)
153
154         output_layer = tf.nn.relu(tf.matmul(FC_layer, w4))
155         output_layer = tf.nn.dropout(output_layer, p_keep_hidden)
156
157         result = tf.matmul(output_layer, w_o)
158         return result
```

Fig. 23 Implementación de LeNet en DNF

Step 3 - Definir el problema: para definir un nuevo problema, en este caso LeNet con MNIST, es necesario crear una clase que herede de **ProblemBuilder**, la nueva clase debe instanciar objetos de las clases **Dataset** (tener disponibles los datos de MNIST) y **NetworkDesigner** (tener el modelo relacionado con la arquitectura de la red, LeNet). Lo anterior permite conocer las características del problema a resolver: rango mínimo y máximo de la solución, tamaño de filas y columnas de las imágenes, número de clases del problema (para este dataset es 10, dígitos de 0 a 9) y finalmente el conjunto total de muestras (training y testing) que componen a MNIST. Adicionalmente, el método **train_eval_test** debe ser implementado debido a que es el encargado de entrenar la arquitectura LeNet (red neuronal convolucional) a través del uso de los optimizadores SGD disponibles en Tensorflow. En este método se definen aspectos básicos como el número de épocas y el tamaño del batch durante el entrenamiento, además, el método debe retornar los pesos y bias (requeridos para las soluciones utilizados en las metaheurísticas), la entropía promedio (valor de aptitud de la solución) y la cantidad de evaluaciones de la función objetivo. Con respecto a este último parámetro, el método

train_eval_test debe incluir tres estructuras condicionales que le indiquen a DNF cómo debe evaluar a una solución según el valor del número de épocas: inicialización de los pesos y bias de la solución (**si** training_epochs = 0), entrenamiento de la red con los pesos y bias evolucionados a través de la metaheurística (**si** training_epochs > 0) y evaluación del conjunto de testing con los pesos y bias entrenados (**si** training_epochs = -1).

En la **Fig. 24** se puede observar la implementación del método **train_eval_test** correspondiente a la clase MNIST_LeNet, inicialmente se observan los tres parámetros requeridos por el método, el primero, relacionados con el vector de pesos y bias de la red que maneja la solución, el segundo parámetro corresponde a las épocas de entrenamiento y el último, la tasa de aprendizaje (línea 45). En las líneas 64, 75 y 95 aparecen las implementaciones relacionadas con el criterio de evaluación de la solución. Por limitaciones en el tamaño del artículo sólo se expande el código que se encarga de entrenar la red con los pesos y bias ya que el parámetro training_epochs es igual a cero (líneas 64 a la 74).

```
45      def train_eval_test(self, Wb, training_epochs, lr):
46          with tf.Graph().as_default() as graph:
47              with tf.Session(graph=graph) as sess:
48                  self.WB = self.no_concatenate(Wb, self.Vector_shapes)
49                  w = self.init_weights(self.WB[0])
50                  w2 = self.init_weights(self.WB[1])
51                  w3 = self.init_weights(self.WB[2])
52                  w4 = self.init_weights(self.WB[3])
53                  w_o = self.init_weights(self.WB[4])
54                  trX, trY, teX, teY = self.data.train.images, self.data.train.labels, self.data.test.images, self.data.test.labels
55                  trX = trX.reshape(-1, self.img_size, self.img_size, 1)
56                  teX = teX.reshape(-1, self.img_size, self.img_size, 1)
57                  X = tf.placeholder("float", [None, self.img_size, self.img_size, 1])
58                  Y = tf.placeholder("float", [None, self.num_classes])
59                  p_keep_conv = tf.placeholder("float")
60                  p_keep_hidden = tf.placeholder("float")
61                  py_x = self.PB.model(X, w, w2, w3, w4, w_o, p_keep_conv, p_keep_hidden)
62                  Y_ = tf.nn.softmax_cross_entropy_with_logits(logits=py_x, labels=Y)
63                  cost = tf.reduce_mean(Y_)
64                  if training_epochs == 0:
65                      efos = 1
66                      sess.run(tf.global_variables_initializer())
67                      average_entropy = 0.0
68                      training_batch = zip(range(0, len(trX), self.batch_size),range(self.batch_size, len(trX) + 1, self.batch_size))
69                      for start, end in training_batch:
70                          entropy = sess.run(cost, feed_dict={X: trX[start:end], Y: trY[start:end], p_keep_conv: 0.8, p_keep_hidden: 0.5})
71                          average_entropy = average_entropy + entropy
72                      average_entropy = average_entropy / len(training_batch)
73                      Wb_np = Wb
74                      SLOPE = None
75                  elif training_epochs == -1:···
95                  else:···
123         gc.collect()
124         return Wb_np, average_entropy, efos
```

Fig. 24 Implementación de la clase MNIST_LeNet

Step 4 - Definir la solución: DNF proporciona la clase **Solution**, con esta clase se tiene a disposición el diseño completo del individuo que contendrá la información necesaria para realizar el proceso de neuro evolución con la metaheurística seleccionada en este ejemplo, por tal motivo, no es necesario modificar esta clase ni crear una nueva que herede de esta. Esta clase adapta el vector a los pesos y bias con base en el problema específico a solucionar. La **Fig. 25** muestra los tres métodos principales de esta clase: inicialización, entrenamiento y prueba sobre un conjunto de testing, según el parámetro **training_epochs** (funciones ubicadas en las líneas 10, 13 y 20). Se puede observar una clara asociación entre la solución y el problema a resolver puesto que la clase **Solution** en su constructor (línea 4) recibe un objeto de la clase **ProblemBuilder**.

En las líneas 22, 24 y 28 de la **Fig. 25** se pueden observar métodos con tareas muy específicas, por ejemplo, **assign_values** que inicializa el vector solución con un vector específico provisto mediante un parámetro de la función, **copy_from** que permite copiar de otro objeto solución, el vector de los pesos y bias, su valor de aptitud y la precisión sobre el conjunto de testing, finalmente **copy_some_variables** que se encarga de copiar únicamente el vector solución de otra solución enviada como parámetro de la función.

```
3   class Solution(object):
4       def __init__(self, problem):
5           self.problem = problem
6           self.accuracy_final = None
7           self.Wb = None
8           self.fitness = None
9           self.valslope = None
10      def eval_fitness(self, training_epochs, lr):
11          self.Wb, self.fitness, efos, self.valslope = self.problem.train_eval_test(self.Wb, training_epochs, lr)
12          return efos
13      def random_initialization(self, training_epochs, lr):
14          LB = self.problem.LB
15          UB = self.problem.UB
16          size_of_solution = self.problem.size_of_solution
17          self.Wb = LB + np.random.rand(size_of_solution) * (UB - LB)
18          efos = self.eval_fitness(training_epochs, lr)
19          return efos
20      def eval_testing_data(self, lr):
21          w, self.accuracy_final, efos, self.valslope = self.problem.train_eval_test(self.Wb, -1, lr)
22      def assign_values(self, wb):
23          self.Wb = wb
24      def copy_from(self, other):
25          self.Wb = np.copy(other.Wb)
26          self.fitness = other.fitness
27          self.accuracy_final = other.accuracy_final
28      def copy_some_variables(self, other, variables):
29          for i in variables:
30              self.Wb[i] = other.Wb[i]
```

Fig. 25 Implementación de la clase Solution

Step 5 - Definir la Metaheurística: para implementar una metaheurística se debe generar una clase que herede de la clase Metaheuristic, en este caso, la clase RMSPropSAHC y a continuación se debe implementar el método run. La **Fig. 26** muestra cómo se debe inicializar el vector de pesos y bias de una red neuronal a través de la creación de una instancia u objeto de la clase Solution (línea 30). En la línea 31 se puede observar una característica importante de DNF, el framework permite inicializar una solución "optimizada" basada en optimizadores del SGD (asociación con la clase ProblemBuilder), por otra parte, esta optimización que se realiza sobre la solución cuenta el número de evaluaciones de la función objetivo (EFOs) usadas, con el objetivo de hacer comparaciones justas frente a otras metaheurísticas o algoritmos clásicos de entrenamiento de redes neuronales. Este contador de EFOs se inicializa en la línea 25 y se actualiza durante el proceso de inicialización y entrenamiento (líneas 31, 38 y 48) para retornar al final la cantidad de EFOs usadas por la metaheurística, valor que es normalmente usado como criterio de terminación del proceso evolutivo (línea 35).

En las líneas 35 a 54 de la **Fig. 26**, se observa cómo se realiza el proceso evolutivo de la población durante un número determinado de EFOs. En el proceso se almacena el valor de aptitud de cada solución en una lista de fitness con el objetivo de tener respaldo en el proceso de selección de las mejores soluciones. Desde las líneas 45 a la 52 y específicamente en la línea 48, se muestra el proceso de explotación contemplado en

DNF, ya que si el valor de training_epochs es mayor que cero se permite el uso de un optimizador clásico de DL sobre la solución, usando el problema que fue previamente definido para dicha solución.

```
3   from Metaheuristics.Solution import *
4   from Utils.Log import Log
5   class RMSPropSAHC(Metaheuristic):
6       def __init__(self, problem, session, experiment):···
24      def run(self, name_of_result):
25          self.efos = 0
26          name_of_result = name_of_result + str(self.experiment) + ".txt"
27          self.list_fitness = []
28
29          if self.best is None:
30              self.best = Solution(self.problem)
31              evaluated_efos = self.best.random_initialization(self.training_epochs, self.learning_rate)
32              self.efos = self.efos + evaluated_efos
33              self.list_fitness.append(self.best.fitness)
34
35          while self.efos < self.max_efos:
36              R = Solution(self.problem)
37              R.copy_from(self.best)
38              evaluated_efos = R.eval_fitness(self.training_epochs, self.learning_rate)
39              self.efos = self.efos + evaluated_efos
40              self.list_fitness.append(R.fitness)
41
42              sigma = self.learning_rate / 10
43              lr_vector = self.learning_rate + abs(np.random.normal(0, sigma, self.population))
44
45              for pop in range(self.population):
46                  W = Solution(self.problem)
47                  W.copy_from(self.best)
48                  evaluated_efos = W.eval_fitness(self.training_epochs, lr_vector[pop])
49                  self.efos = self.efos + evaluated_efos
50                  self.list_fitness.append(W.fitness)
51                  if W.fitness < R.fitness:
52                      R.copy_from(W)
53              if R.fitness < self.best.fitness:
54                  self.best.copy_from(R)
55          if self.testing:
56              self.best.eval_testing_data(self.learning_rate)
```

Fig. 26 Implementación de la clase Metaheurística: SAHC

Step 6 - Definir el experimento: toda metaheurística y problema implementado debe ser importado en el archivo **DNF.py** (línea 18 de la **Fig. 27**), luego este se modifica para ejecutar el problema con su metaheurística incluyendo algunos condicionales que permitan la selección de los parámetros requeridos: Problema y Metaheurística (líneas 76 a la 79). Después, se utiliza el archivo **Experimenter.py** para la ejecución del experimento. Este archivo tiene dos variables denominadas **Type_of_metaheuristic** y **Type_of_problem** (líneas 28 y 29 en la **Fig. 28**) las cuales una vez ajustadas, ejecutan a través de una ventana de comandos el experimento, este proceso será ejecutado 30 veces (por defecto para que el valor promedio cumpla con el teorema de límite central, a partir de la línea 31), pero este número puede ser ajustado. Los valores de las entropías en el proceso de entrenamiento y precisión sobre el conjunto de testing se almacenan en un archivo de texto que se crea en la misma carpeta de DNF.

```
from Metaheuristics.RMSProp_HillClimbing.RMSProp_SAHC_E2 import *

if __name__ == "__main__":
    tf.set_random_seed(0)
    sess = tf.Session()
    exp = int(argv[1])
    ProblemId = int(argv[2])
    Type_of_Metaheuristic = int(argv[3])
    name_of_result = ''
    if ProblemId == 1:···
    elif ProblemId == 2:···
    elif ProblemId == 3:
        TheProblem = [Problem_MNIST_cnn(), sess, exp]
        name_of_result = 'MNIST_cnn'
    else:···
    if Type_of_Metaheuristic == 0:···
    elif Type_of_Metaheuristic == 9:
        name_of_result = name_of_result + '_RMSProp_SAHC_'
        RMSPropSAHC_alg = RMSPropSAHC(*TheProblem)
        RMSPropSAHC_alg.run(name_of_result)
    elif Type_of_Metaheuristic == 10:···
    elif Type_of_Metaheuristic == 11:···
    elif Type_of_Metaheuristic == 12:···
    else:···
    gc.collect()
    sess.close()
```

Fig. 27 Código principal de DeepNeuroevolution Framework

```
'''
Problem                          id
Problem_MNIST_one_hidden_layer   1
Problem_MNIST_five_hidden_layers 2
Problem_MNIST_cnn                3

Type of Metaheuristic
GBHS (Global Best Harmony Search)  0
PSO (Particle Swarm Optimization)  1
DE (Differential Evolution)        2
RMSProp                            3
MTS                                4
SADEBD                             5
IHDELS                             6
MOS                                7
RMSProp_HC                         8
RMSProp_SAHC                       9
RMSProp_SAHCwR                     10
RMSProp_CoefVaria                  11
RMSProp_E2_CriterioSlope           12
'''

Type_of_Problem = 3
Type_of_Metaheuristic = 12

for exp in range(30):
    cmd = 'python DNF.py '+ str(exp) + ' ' + str(Type_of_Problem) + ' ' + str(Type_of_Metaheuristic)
    print (cmd)
    os.system(cmd)
```

Fig. 28 Programa Experimenter perteneciente a DNF

En la **Fig. 29** se presenta el histograma de las frecuencias observadas para rangos de precisión y entropía cruzada en los 30 experimentos. Estos resultados son obtenidos con DNF para la optimización del entrenamiento de la red neuronal denominada LeNet (minimización del valor de la entropía cruzada) para los conjuntos de datos MNIST y FASHION-MNIST vs los mismos resultados obtenidos por RMSProp. Las metaheurísticas utilizadas estuvieron basadas en ascenso de colina por la máxima pendiente (SAHC) con una modificación en el proceso de explotación utilizando RMSProp (RMSProp SAHC). Se puede observar que el algoritmo RMSPropSAHC obtiene mejores resultados con respecto al RMSProp, ya que RMSPropSAHC obtiene mayor cantidad de soluciones con valores más bajos de entropía en los datasets (Columna 2 de la **Fig. 29**), de igual manera se observa como a medida que el conjunto de datos aumenta en complejidad (FASHION-MNIST es un dataset más complejo que MINIST) el algoritmo RMSPropSAHC mejora sus valores de accuracy, llegando a ser el único algoritmo que obtiene valores de accuracy de 50% (Columna 1 de la **Fig. 29**).

Por otra parte, en la Tabla 7 se presentan los resultados de los valores promedios de las 30 ejecuciones de los dos algoritmos, RMSProp y RMSPropSAHC tanto para accuracy como para entropía cruzada. Los resultados muestran como la metaheurística implementada con explotación basada en SGD (RMSPropSAHC) logra vencer en entrenamiento (entropía cruzada) y prueba (accuracy) al RMSProp (valores en negrita y color rojo en la **Tabla 7**). Lo anterior muestra que la Neuro evolución expresada en el desarrollo del DNF es una alternativa para futuras investigaciones en el área de la optimización del aprendizaje de redes neuronales profundas.

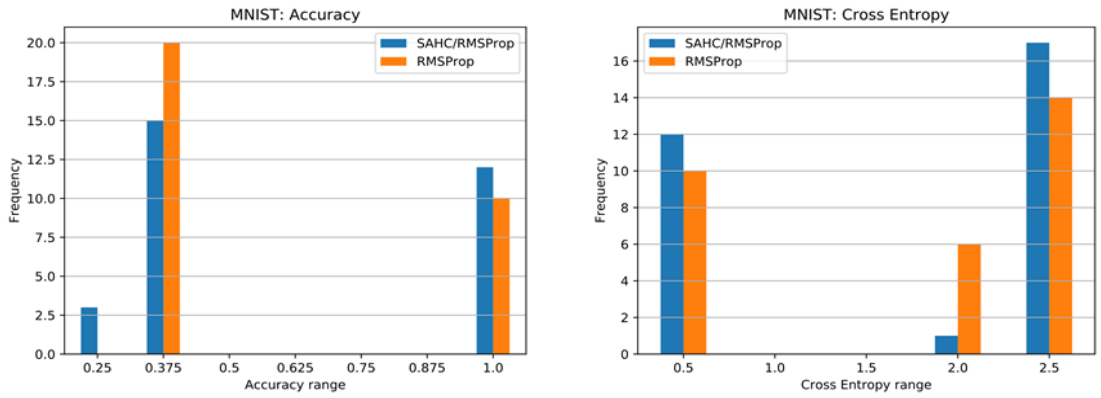

Fig. 29 Histograma de frecuencias observadas en rangos para los datasets MNIST entrenado con los algoritmos RMSProp y RMSPropSAHC usando 30 repeticiones.

5.2.4 Impacto

La neuro evolución es una alternativa para la optimización del aprendizaje de redes neuronales profundas, sin embargo, recientes investigaciones han demostrado que las nuevas aplicaciones en el área están creciendo lentamente [90], a pesar del potencial que tiene debido a que son algoritmos especializados para escapar de los principales problemas de la alta dimensionalidad comunes en el campo del DL. Con DNF se busca impulsar las investigaciones que interceptan a los EA con el DL, aprovechando lo mejor de los dos mundos a través de la implementación de metaheurísticas que utilizan algoritmos inspirados en la naturaleza para realizar el proceso de exploración/explotación probabilística junto con la optimización local basada en SGD.

Tabla 7 Valores promedios de Entropía Cruzada (menor valor expresa mayor calidad) y Accuracy (mayor valor expresa mayor calidad) sobre 30 experimentos en los datasets MNIST utilizando los algoritmos RMSProp y RMSPropSAHC

Algorithm	MNIST	
	Avg. Accuracy	**Avg. Cross Entropy**
RMSProp	0.51271875	1.4076886
RMSPropSAHC	**0.54953646**	**1.2870173**

DNF se basa en clases altamente cohesivas y débilmente acopladas que permiten enfocarse en el desarrollo de nuevas metaheurísticas sin preocuparse de los procesos comunes en la creación de redes neuronales profundas. La comunicación entre los cinco componentes principales de DNF: Dataset, Problema, Arquitectura de Red, Solución, y Metaheurística permite que el diseño de nuevos experimentos y metaheurísticas sean muy sencillas, con resultados repetibles, extensibles y fácilmente modificables. DNF además busca que la comunidad abandone el uso del scripting (código mezclado con datos difícil de entender y extender) como estrategia de codificación y empiece a incorporar mejores prácticas de ingeniería de software que faciliten entender y extender los algoritmos desarrollados.

Teniendo en cuenta que DNF se soporta en TensorFlow como principal herramienta de implementación de redes neuronales profundas, garantiza a los investigadores que su proceso de entrenamiento estará altamente optimizado en tiempo y recursos computacionales, debido a que, por las propiedades de dicha librería estará habilitado el trabajo con los núcleos CPU/GPU disponibles, por lo que el usuario estará utilizando el primer framework de neuro evolución que trabaja todas las ventajas de esta librería en continua actualización y mejoramiento.

6 Propuestas meméticas finales

En el **¡Error! No se encuentra el origen de la referencia.**, se obtuvieron conclusiones claves acerca de la optimización del entrenamiento de redes neuronales profundas convencionales basado en la aplicación de algoritmos meméticos. Uno de esos descubrimientos está relacionado con tener en cuenta que, para poder competir contra el RMSProp, un algoritmo de optimización local, se debería tener especial atención en lanzar procesos de exploración cuando sean requeridos, debido a que el algoritmo memético puede "gastar" EFOs de forma innecesaria en zonas del espacio de búsqueda (altamente dimensional) que se pueden resolver con el descenso del gradiente.

Por otra parte, metaheurísticas con un grado de complejidad bajo en su implementación lograron resolver de mejor manera los problemas de la alta dimensionalidad en el entrenamiento de una red convencional profunda. Ahora, como se ha explicado a lo largo del documento, conjuntos de datos como los tratados por Morse (Función de aproximación, serie de tiempo de Mackey-Glass y California Housing) no representan un mayor reto en el aprendizaje profundo, además, arquitecturas de redes neuronales convencionales como la tratada en el capítulo 4 ya hoy no son de interés para la comunidad académica e investigativa, ya que su estudio y análisis ha sido abordado por completo.

Por lo anterior, en la investigación se diseñó un experimento que propone retos más desafiantes en el proceso de la neuro evolución. El experimento consistió en optimizar el proceso de aprendizaje de una red neuronal profunda convolucional llamada LeNet, utilizando el conjunto de datos MNIST y FASHION-MNIST el cual se caracteriza por tener un espacio de búsqueda más complejo.

Debido a que los mejores resultados obtenido en el capítulo 4 están relacionados con las diversas variaciones del ascenso de colina, en este capítulo se crearon dos enfoques basados en la metaheurística de ascenso de colina por la máxima pendiente (SAHC, Steepest ascent Hill climbing), los cuales se explican a continuación.

6.1 SAHC_BestSlope

El **Seudocódigo 1Seudocódigo 7** muestra el primer enfoque para optimizar el aprendizaje de la red neuronal convolucional LeNet con el conjunto de datos MNIST. El enfoque se denominó SAHC_BestSlope debido a que el criterio que tiene para seleccionar el mejor individuo del vecindario es el valor de la pendiente, este valor se calcula teniendo como base un número determinado de valores de aptitud (entropía cruzada) obtenidos durante las épocas de entrenamiento de cada solución. Es decir, inicialmente el algoritmo crea una solución o recibe una solución inicial (Línea 2). Para poder generar la vecindad, el algoritmo en la línea 3 crea un vector de tamaño número de vecinos con valores aleatorios de tasas de aprendizaje, posteriormente aplica el optimizador RMSProp un número de épocas para lograr ubicar a la solución inicial en una mejor posición del espacio de búsqueda (Línea 5), cada época de entrenamiento obtiene un valor para la entropía cruzada, ese valor se almacena en un vector de tamaño n_epocas y finalmente, cuando termina el entrenamiento, se calcula la pendiente para todos los valores de entropía almacenos en el vector.

Desde la línea 6 hasta la línea 9 se crean los vecinos relacionados con la solución inicial (Línea 7), se verifica y se reemplaza el mejor de la vecindad basado en el criterio de la menor pendiente (Línea 8 y 9). Finalmente, el mejor de la vecindad se compara con respecto a su pendiente con la mejor solución que se tiene hasta el momento, una pendiente menor indica que la entropía durante el entrenamiento está decayendo, lo que permite inferir que el algoritmo no se está estancando en óptimos locales, puntos de silla o platea, por lo que el mejor individuo será quien tenga la menor pendiente.

Seudocódigo 7 Algoritmo Ascenso a la colina por la máxima pendiente enfoque 1: Mejor pendiente (SAHC_BestSlope). (Adaptado de [88]).

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$
Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
1:  neighborhood <- número de vecinos
2:  mejor <- solución candidata inicial
3:  lr <- vector con tasas de aprendizaje
4:  para i = 0 hasta maxEvaluaciones
5:     R <- AplicarRMSProp(Copiar(mejor),n_epocas,lr[0])
6:     para i = 0 hasta neighborhood -1
7:        W <- AplicarRMSProp(Copiar(mejor),n_epocas,lr[i])
8:        si W.valSlope < R.valSlope entonces
9:           R <- W
10:       si R.valSlope < mejor.valSlope entonces
             mejor <- R
11: retornar mejor
```

Los resultados del entrenamiento con el enfoque 1 (SAHC_BestSlope) y su comparación contra el optimizador RMSProp para la arquitectura LeNet y MNIST se muestra en la **Fig. 30**. En esta figura se pueden observar las nubes de puntos que se obtuvieron de las 30 repeticiones del experimento, para los dos algoritmos de optimización de la arquitectura LeNet utilizando el conjunto de datos MNIST.

La nube de puntos correspondiente a SAHC_BestSlope presenta resultados con valores más bajos para la entropía cruzada en comparación con el RMSProp, además, al observar las líneas de tendencia se puede verificar que SAHC_BestSlope alcanza mejores resultados. En aras de dar un mejor panorama de la comparación y observar claramente que SAHC_BestSlope vence al optimizador RMSProp se decide obtener de las 30 repeticiones del experimento el valor final de entropía cruzada, posteriormente se ordena de mayor a menor y se calcula un histograma que contabiliza la cantidad de valores de entropía que se obtuvieron en cinco rangos específicos, la **Fig. 31** muestra los resultados calculados.

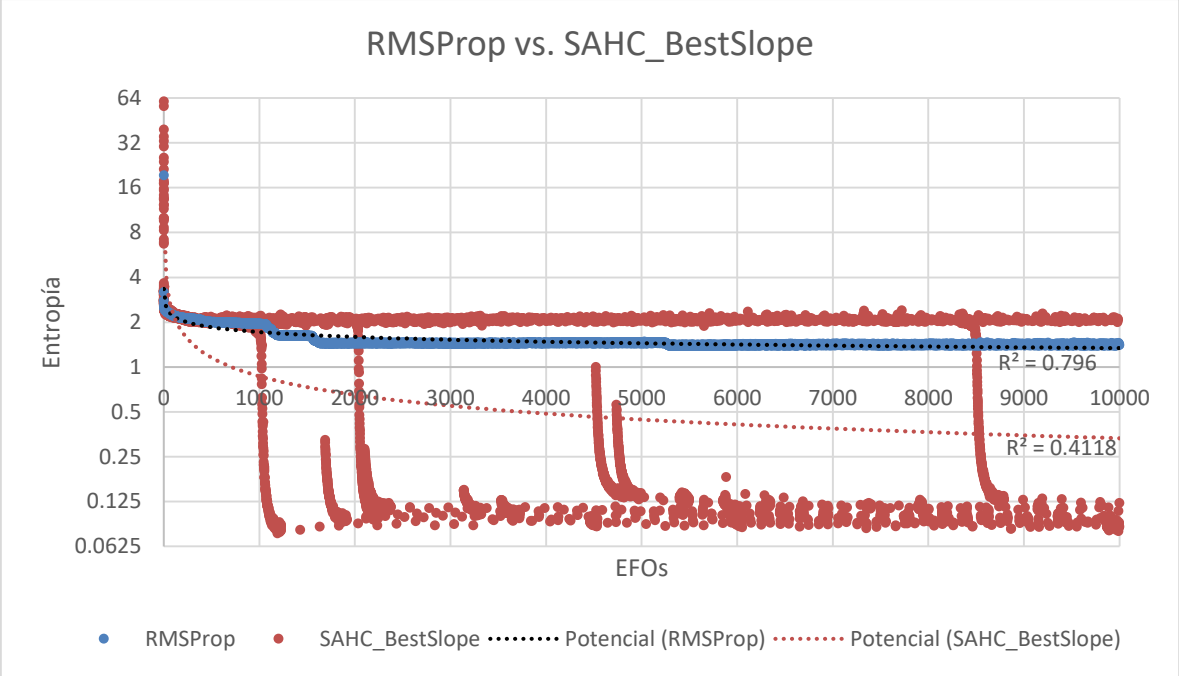

Fig. 30 Comparación entre SAHC_BestSlope vs. RMSProp en la optimización de la entropía cruzada para la red neuronal convolucional profunda LeNet sobre MNIST

En la **Fig. 31** se observa que SAHC_BestSlope obtiene más soluciones con menor valor de entropía que RMSProp, por lo que al final del entrenamiento SAHC_BestSlope brindó un modelo mejor entrenado en la optimización del problema. Similar a la entropía cruzada esta misma gráfica se construye para los valores de precisión en el conjunto de prueba del MNIST, la **Fig. 32** muestra el histograma con las frecuencias acumuladas de la precisión. En esta figura se observa como SAHC_BestSlope logra obtener mejores resultados finales de precisión al término de las 30 repeticiones del experimento, logrando vencer al RMSProp en la etapa de prueba sobre el conjunto de datos destinados para tal fin en el MNIST.

La **Tabla 8** que se muestra a continuación, presenta de forma resumida los promedios de las últimas entropías obtenidas del entrenamiento, así como, los promedios sobre los 30 resultados de precisión. En esta tabla, en color rojo, se resaltan los valores ganadores en los dos algoritmos, el SAHC_BestSlope logra vencer tanto en la entropía cruzada como en la precisión sobre el dataset de prueba en promedio con las 30 repeticiones del experimento.

Con respecto a los tiempos de ejecución, no se puede llegar a ninguna conclusión basado en los resultados experimentales, ya que el tiempo promedio para realizar 10000 épocas o evaluaciones de la función objetivo para el RMSProp fue de 22.48 horas mientras que para el SAHC_BestSlope se tiene un tiempo promedio de 15.89 horas, pero la experimentación se realizó con varios equipos con diferentes tarjetas gráficas NVIDIA, por ello, la diferencia en tiempos se debe al poder de cómputo de cada tarjeta gráfica así como los núcleos CUDA disponibles en cada dispositivo. Teóricamente, RMSProp debe ser ´levemente más rápido que SAHC_BestSlope, ya que no debe almacenar y calcular la pendiente, pero esta tarea es insignificante en relación con el proceso de ajuste de pesos en la red, es decir, la evaluación de una función objetivo (EFO).

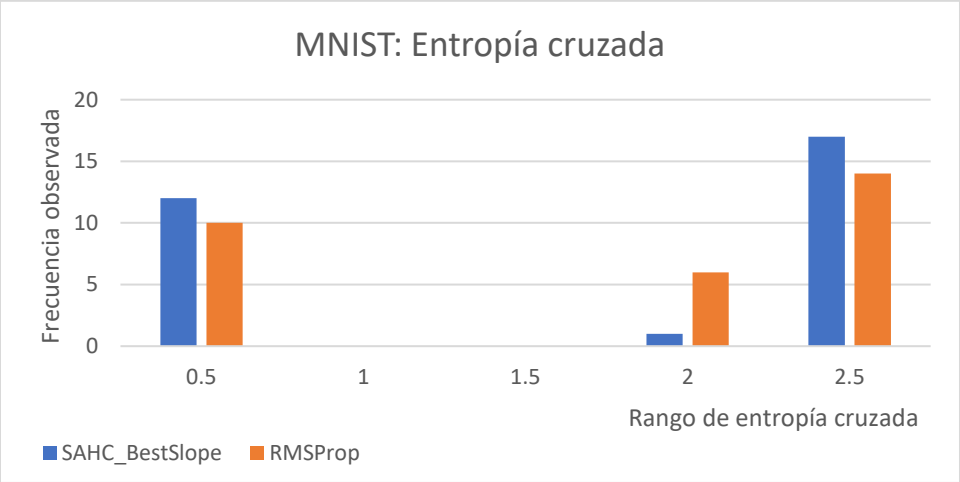

Fig. 31 Histograma de entropías finales del entrenamiento de la red neuronal LeNet con MNIST para los dos algoritmos: SAHC_BestSlope y RMSProp

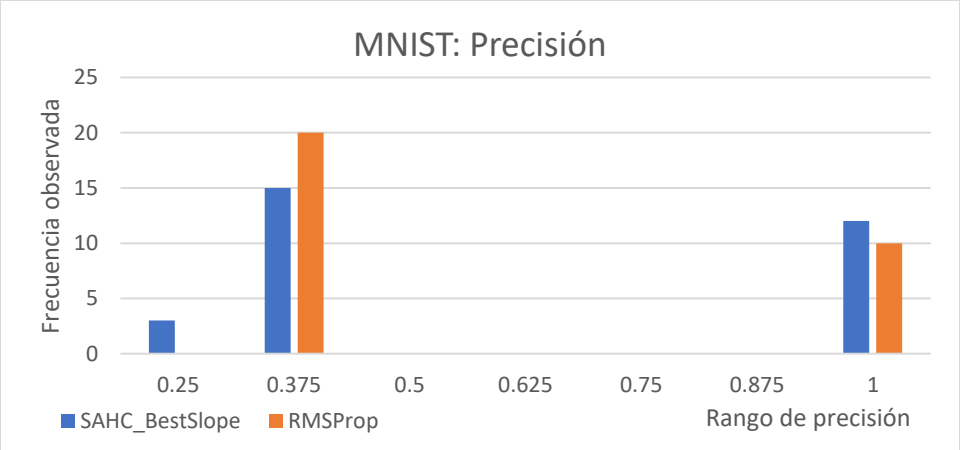

Fig. 32 Histograma de precisiones finales del entrenamiento de la red neuronal LeNet con MNIST para los dos algoritmos: SAHC_BestSlope y RMSProp

Tabla 8. Resumen de los promedios de entropía y precisión sobre la arquitectura LeNet y MNIST para los algoritmos SAHC_BestSlope y RMSProp.

Algoritmo	Promedio de Entropía cruzada	Promedio de Precisión (%)
SAHC_BestSlope	**1.287**	**54.953**
RMSProp	1.407	51.271

Continuando con la experimentación, se realizó el proceso de entrenamiento de la red LeNet pero está vez se cambia el conjunto de datos a FASHION-MNIST, que tiene un nivel de complejidad mayor a MNIST por lo que se quiere probar el algoritmo

SAHC_BestSlope con este nuevo dataset. Los resultados de comparación entre SAHC_BestSlope y RMSPRop sobre FASHION-MNIST se muestran en la **Tabla 9**.

Tabla 9. Resumen de los promedios de entropía y precisión sobre la arquitectura LeNet y FASHION-MNIST para los algoritmos SAHC_BestSlope y RMSProp.

Algoritmo	Promedio de Entropía cruzada	Promedio de Precisión (%)
SAHC_BestSlope	2.213	21.405
RMSProp	**2.190**	**22.722**

En esta oportunidad se observa en la **Tabla 9**, en color rojo, que el enfoque SAHC_BestSlope no vence al RMSProp, sin embargo, los valores logrados no están lejos de los obtenidos por RMSProp. Los tiempos promedios para realizar un experimento basado en 10000 épocas para RMSProp y SAHC_BestSlope fueron 17.79h y 16.94h, respectivamente.

6.2 SAHC_E2

El espacio de soluciones del conjunto de datos FASHION-MNIST es más complejo que el de MNIST. Usando el enfoque 1, basado en la escogencia del vecino con la menor pendiente o como se conoce actualmente SAHC_BestSlope realiza muy seguido, procesos de exploración sobre el espacio de búsqueda, haciendo que RMSPRop tome ventaja sobre el algoritmo memético debido a que en este dataset la explotación tiene mayor importancia. Por lo tanto, se propone un segundo enfoque que maneja de manera más eficiente la exploración, iniciando esta tarea sólo en ciertos momentos. Debido a que el concepto de pendiente tuvo éxito en el experimento anterior, en este enfoque 2 se mantiene como criterio para iniciar la exploración. El **Seudocódigo 8** muestra el algoritmo llamado SAHC_E2.

SAHC_E2 crea una solución inicial (Línea 2) la cual se optimiza a través de RMSProp (Línea 5), nuevamente se construye un vector de tasas de aprendizaje basado en el tamaño del vecindario (Línea 6) y se almacena los valores de aptitud o entropía cruzada en un vector (Línea 7). Se verifica si el vector de fitness está lleno con el total de valores permitidos (Línea 8) y se procede a calcular la pendiente de estos valores (Línea 9). SAHC_E2 utiliza este valor de pendiente para tomar la decisión de lanzar o no el proceso de exploración. Si la pendiente es mayor que cero, significa que la entropía cruzada no está disminuyendo, en este punto SAHC_E2 utiliza a SAHC_BestSlope (Línea 11) para generar un vecindario y ejecutar el proceso de exploración. Finalmente, el mejor valor de aptitud retornado por SAHC_BestSlope se toma como el mejor del proceso de exploración para continuar el proceso de entrenamiento con este.

La **Fig. 33** muestra la nube de puntos con los valores de entropía cruzada para la red neuronal convolucional LeNet en el conjunto de datos FASHION-MNIST y los algoritmos RMSProp y SAHC_E2. También se presentan las líneas de tendencia para los dos algoritmos, lograndose observar que SAHC_E2 tiene mejores valores de entropía, la nube de puntos del RMSProp no alcanza regiones más bajas de niveles de entropía que el SAHC_E2.

Seudocódigo 8 Algoritmo Ascenso a la colina por la máxima pendiente enfoque 2: Exploración basado en la pendiente (SAHC_E2). (Adaptado de [88]).

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$
Tamaño del vector de pendientes $\boldsymbol{size\_of\_buffer} > \boldsymbol{0}$
Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
neighborhood <- número de vecinos
mejor <- solución candidata inicial
lr <- vector con tasas de aprendizaje
para i = 0 hasta maxEvaluaciones
    R <- AplicarRMSProp(Copiar(mejor),n_epocas,lr[0])
    Lr <- crearVectorTasasAprendizaje(neighborhood)
    fitnessVector <- AlmacenarFitness(R.fitness)
    si len(fitnessVector) == size_of_buffer entonces
        pendiente <- CalcularPendiente(fitnessVector)
        si pendiente > 0 entonces
            mejor <- SAHC_BestSlope(mejor)
retornar mejor
```

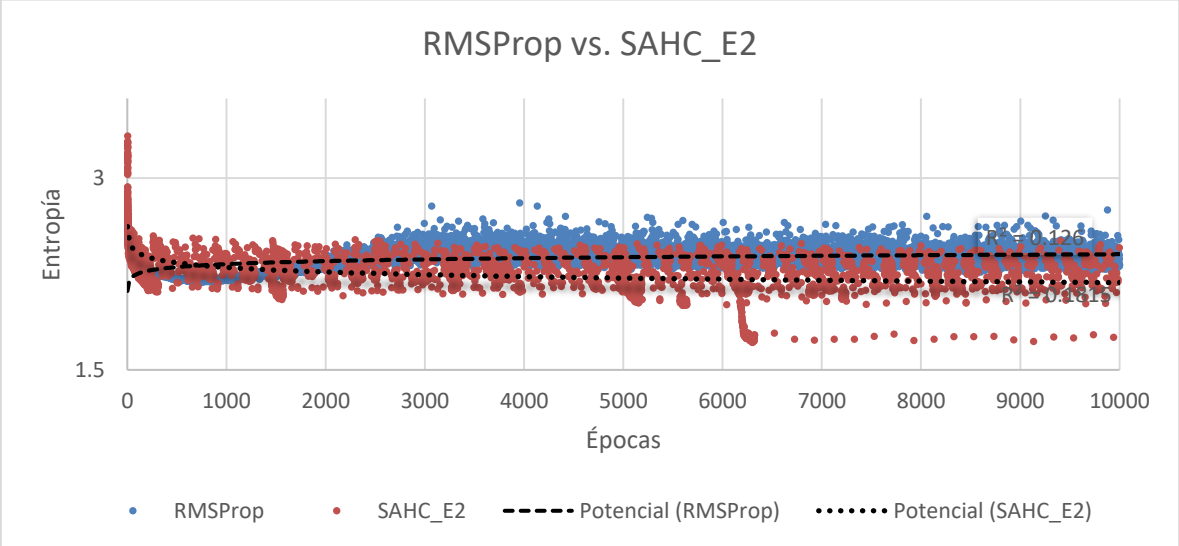

Fig. 33 Comparación entre SAHC_E2 vs. RMSProp en la optimización de la entropía cruzada para la red neuronal convolucional profunda LeNet sobre FASHION-MNIST

Al igual que en el experimento anterior, con el objetivo de mejorar la visualización y representación de los resultados se realizan los histogramas de los valores de entropía cruzada con la precisión obtenida durante la última época de entrenamiento. Los resultados se muestran en las **Fig. 34** y **Fig. 35.**

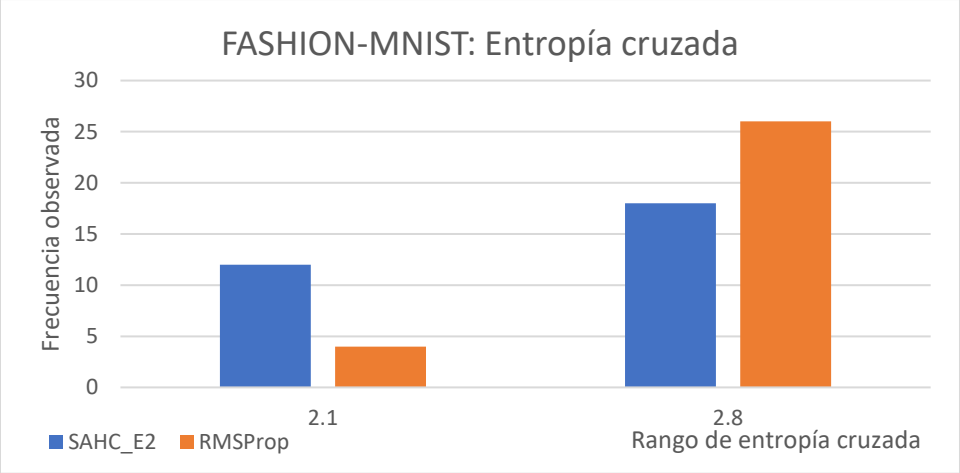

Fig. 34. Histograma de entropías finales del entrenamiento de la red neuronal LeNet con FASHION-MNIST para los dos algoritmos: SAHC_E2 y RMSProp.

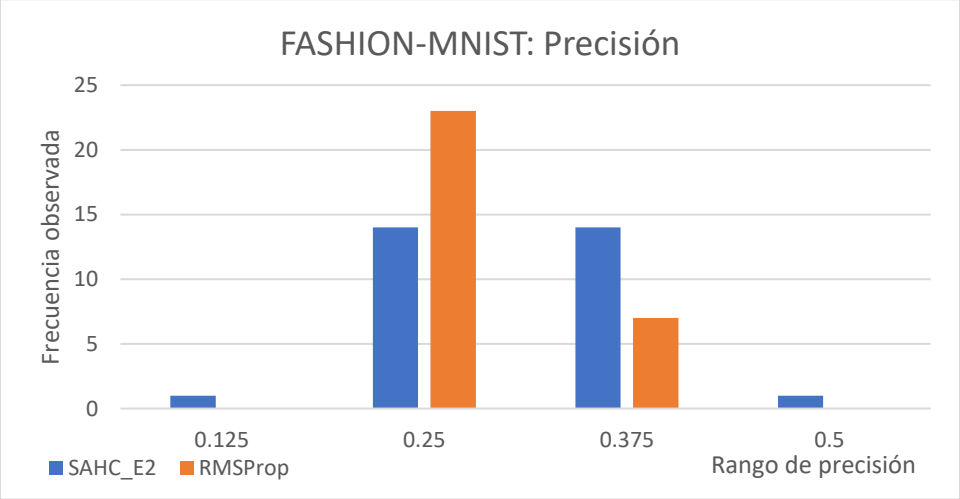

Fig. 35. Histograma de precisiones finales del entrenamiento de la red neuronal LeNet con FASHION-MNIST para los dos algoritmos: SAHC_E2 y RMSProp.

En la **Fig. 34** se observa como SAHC_E2 supera considerablemente a RMSProp en obtener buenas soluciones en la entropía cruzada, de igual manera en la **Fig. 35** se puede verificar que el SAHC_E2 es el único algoritmo capaz de obtener precisiones superiores al 50%. Lo anterior permite concluir que el segundo enfoque logra mejorar el proceso de entrenamiento de la red neuronal convolucional LeNet y el conjunto de datos FASHION-MNIST.

Con respecto al tiempo promedio que utiliza SAHC_E2 para realizar 10000 épocas de entrenamiento, se puede decir que es de 15.44h lo que indica que la propuesta memética sigue siendo computacionalmente viable con respecto al RMSProp, pero al igual que en el caso anterior, no se puede dar una conclusión en relación con los tiempos debido al uso de diversos equipos. Finalmente, la **Tabla 10** consigan los valores promedios de entropía

y precisión de la última época realizada en los 30 experimentos comparándolos con los resultados obtenidos por el RMSProp.

Tabla 10. Resumen de los promedios de entropía y precisión sobre la arquitectura LeNet y FASHION-MNIST para los algoritmos SAHC_E2 y RMSProp.

Algoritmo	Promedio de Entropía cruzada	Promedio de Precisión (%)
SAHC_E2	**2.127**	**23.774**
RMSProp	2.19	22.722

En la **Tabla 10**, en color rojo, se observa como los promedios de entropía cruzada al final del entrenamiento y la precisión en la etapa de prueba del conjunto de datos FASHION-MNIST favorece al algoritmo SAHC_E2. Esto permite inferir que manejar inteligentemente los procesos exploratorios en el aprendizaje de la red con conjuntos de datos de alta complejidad es una característica clave para lograr superar al RMSProp que únicamente realiza procesos de explotación.

En la práctica, al entrenar la red neuronal cambiando las semillas (repeticiones) se escogería el modelo con la mayor precisión en el conjunto de prueba y también se observaría cuál fue el valor más bajo de entropía cruzada, la **Tabla 11** y la **Tabla 12** muestran los valores mínimos y máximos obtenidos de todos las repeticiones realizadas.

Tabla 11. Valores mínimo y máximo de la entropía cruzada para la red neuronal convolucional LeNet con el conjunto de datos MNIST y FASHION con los tres algoritmos: RMSProp, SAHC_BestSlope y SAHC_E2

Algoritmos (Dataset)	**Entropía mínima**	**Entropía máxima**
RMSProp (MNIST)	0.085	2.245
SAHC_BestSlope (MNIST)	**0.083**	**2.168**
RMSProp (FASHION)	1.956	2.720
SAHC_BestSlope (FASHION)	1.802	2.677
SAHC_E2 (FASHION)	**1.706**	**2.357**

Tabla 12. Valores mínimo y máximo de precisión para la red neuronal convolucional LeNet con el conjunto de datos MNIST y FASHION con los tres algoritmos: RMSProp, SAHC_BestSlope y SAHC_E2

Algoritmos (Dataset)	**Mínima Precisión**	**Máxima Precisión**
RMSProp (MNIST)	**25.13**	98.40
SAHC_BestSlope (MNIST)	17.69	**98.73**
RMSProp (FASHION)	**17.95**	29.22
SAHC_BestSlope (FASHION)	15.76	46.22
SAHC_E2 (FASHION)	12.32	**46.67**

En la **Tabla 11**, se puede observar que para los dataset MNIST y FASHION los valores de entropía cruzada obtenidos por los diferentes enfoques propuestos y el RMSProp. En esta tabla se muestra en negrilla los mejores valores de entropía mínima y máxima para el enfoque vencedor (SAHC_BestSlope), cabe aclarar que el objetivo del entrenamiento consiste en minimizar los valores de entropía, por lo que entropías bajas implican un mejor entrenamiento. Con respecto a FASHION, el mejor rango de entropía se obtiene con el enfoque SAHC_E2, señalados en color rojo en la misma tabla. Lo anterior permite concluir que en un caso práctico SAHC_BestSlope y SAHC_E2 tendrían los mejores modelos de entrenamiento para MNIST y FASHION respectivamente.

En la **Tabla 12**, se muestran los valores de precisión mínimo y máximo, lo cuales se han resaltado en negrilla y en color rojo en cada columna. En este caso, RMSProp logra obtener precisiones mínimas más altas que los enfoques SAHC_BestSlope y SAHC_E2. Sin embargo, los enfoques propuestos en esta investigación son los únicos que alcanzan precisiones superiores al 46%, el RMSProp se encuentra 17 puntos porcentuales por debajo de los enfoques meméticos.

Los anteriores resultados permiten analizar las bondades que poseen los nuevos enfoques frente al algoritmo tradicional RMSProp. Lanzar procesos de exploración de manera inteligente y controlada en SAHC_BestSlope y SAHC_E2 basado en el decaimiento de la pendiente en el proceso de minimización de la entropía cruzada facilitan el escape de óptimos locales, puntos de silla y platea en el espacio de búsqueda, cabe aclarar que el RMSProp carece de esta cualidad y siempre, hasta el final del entrenamiento, tiene una única solución. Por otra parte, utilizar el RMSProp como proceso de explotación favorece en que la optimización del aprendizaje cae siempre con información del gradiente evitando las desventajas que presentaban las metaheurísticas debido a su naturaleza probabilística y que no logran ser competitivas cuando el RMSProp encuentra un camino hacia valores óptimos de entropía. Los resultados logrados para los conjuntos de datos MNIST y FASHION-MNIST mostraron como dos nuevos enfoques de metaheurísticas logran vencer al RMSProp en campos donde el aprendizaje profundo domina, lo anterior, abre muchas posibilidades puesto que las metaheurísticas tienen un gran número de enfoques que pueden ser probados sobre varios conjuntos de datos y arquitecturas del aprendizaje profundo.

7 Conclusiones y trabajos a futuro

Las conclusiones de este trabajo son:

1. En relación con el primer objetivo específico, se realizó la comparación durante 30 repeticiones sobre los tres conjuntos de regresión: Aproximación de una función, serie de tiempo Mackey-Glass y California Housing para todos los algoritmos: RMSProp, LEEA, MOS, DECC-G e IHDELS.

   Adicional a lo originalmente planteado y con el fin de aportar al campo de la neuro evolución y profundizar más en la investigación se diseñó, implementó y probó un marco de trabajo para el entrenamiento de redes neuronales profundas utilizando metaheurísticas llamado: Deep Neuroevolution Framework (DNF). DNF tiene un diseño débilmente acoplado y altamente cohesivo lo que permite crear fácilmente nuevas propuestas metaheurísticas y meméticas que logren minimizar métricas de evaluación como el error cuadrático medio en algoritmos de regresión y entropía cruzada en algoritmos de clasificación. DNF facilita la inclusión de nuevas propuestas en el campo de redes neuronales profundas optimizadas a través de algoritmos inspirados en la naturaleza (Neuro evolución), facilitando así que se amplié el conocimiento en este campo de investigación. Las clases implementadas en DNF permiten independizar los procesos, por una parte, relacionados con las redes neuronales profundas: carga de datos, diseño de la arquitectura de la red neuronal y su entrenamiento y por el otro, el diseño de diversas metaheurísticas pertenecientes a los procesos evolutivos. En DNF, utilizando la clase ProblemBuilder es posible definir los datos del problema e implementar cualquier red neuronal profunda utilizando TensorFlow, por otro lado, con la clase Metaheuristic, se puede implementar, independientemente de la red neuronal artificial, cualquier algoritmo inspirado en la naturaleza y por último, con la clase Solution se le brinda al investigador la posibilidad de optimizar cada solución de la población con algoritmos optimizadores del gradiente descendente estocástico (SGD, por sus siglas en inglés) reconocidos en el estado del arte e implementados en TensorFlow. La sinergia implementada en DNF entre algoritmos evolutivos (Evolutionary Algorithms, EA) y aprendizaje profundo (Deep Learning, DL), espera generar una motivación especial en el área de la neuro evolución logrando que nuevos enfoques proliferen basados en la mejora de las métricas de evaluación en el entrenamiento de redes neuronales profundas para diferentes conjuntos de datos que diariamente son publicados. DNF está disponible como software de código abierto y se espera que se utilize por la comunidad científica que intercepta los EA con DL, interesada en la optimización del entrenamiento de redes neuronales profundas, además, se espera que el framework sea enriquecido a través de la adición de nuevas metaheurísticas, arquitecturas de redes neuronales profundas y conjuntos de datos del estado del arte y futuros de DL, conservando y motivando el uso de buenas prácticas de ingeniería de software en detrimento del scripting.

2. En relación con el segundo objetivo específico, en esta investigación se adaptaron las metaheurísticas seleccionadas (DECC-G, IHDELS y MOS) al proceso de entrenamiento de la red neuronal profunda en los tres problemas de regresión seleccionados conforme lo previamente planteado. La primera repetición del experimento con CC-CMA-ES en la aproximación a la función demoró demasiado

tiempo, haciendo que esta metaheurística fuera descartada del resto de la investigación, ya que ni con una tarjeta NVIDIA TITAN XP se lograban tiempos aceptables para el desarrollo del proyecto, tema que además la hace impráctica para la solución de problemas más complejos. Las otras tres metaheurísticas (DECC-G, IHDELS y MOS) también se incluyeron en DNF con el objetivo de realizar la experimentación y su adaptación al framework fue sencilla de hacer.

Adicionalmente, se definieron e implementaron nuevas propuestas de algoritmos meméticos (optimización global + optimización local + conocimiento del problema), los cuales unen exitosamente las cualidades de explotación de los algoritmos optimizadores del gradiente descendente estocástico con las cualidades de exploración de las metaheurísticas. Las nuevas propuestas lograron vencer al RMSProp en los diferentes conjuntos de datos planteados originalmente y, además, lograron vencerlo en conjuntos de datos de imágenes tales como los dígitos dibujados a mano alzada (MNIST) y el conjunto de datos de prendas de vestir (FASHION-MNIST). Estas propuestas también fueron adaptadas e implementadas en DNF, facilitando su uso con conjuntos de datos para algoritmos de regresión, así como clasificación. Por otra parte, la optimización del aprendizaje de redes neuronales profundas no sólo fue realizado sobre redes neuronales convencionales sino sobre una arquitectura de red neuronal convolucional LeNet. En DNF queda disponible la arquitectura de red neuronal convolucional AlexNet para futuros entrenamientos.

3. En relación con el tercer objetivo específico, se puede concluir que las metaheurísticas de optimización global logran los mejores resultados en el proceso de minimización del error cuadrático medio sobre los tres conjuntos de datos: Aproximación de una función, Serie de tiempo y California Housing. Los resultados permiten concluir que algoritmos evolutivos simples como LEEA no tienen posibilidad frente algoritmos determinísticos como RMSProp. LEEA fue competitivo con el RMSProp después de evaluar muchas veces la función objetivo, tal como lo demuestran los resultados de Morse et al., sin embargo, en la práctica, un entrenamiento extenso es muy poco común por su costo computacional y tiempo invertido, sin embargo, las propuestas basadas en metaheurísticas y meméticas implementadas en la investigación lograron responder de una mejor manera en los mismos tiempos e iteraciones que RMSPRop, por lo que, después de la investigación, se puede concluir que la hipótesis planteada en la investigación es cierta, puesto que, no solo se logra vencer en los tres dominios de regresión, sino que se compite/vence al RMSProp.

   Las propuestas meméticas finales, diseñadas e implementadas en esta investigación logran vencer en conjuntos de datos relacionados con imágenes y utilizados en arquitecturas de redes más complejas como lo son las redes convolucionales. La clave de estas propuestas meméticas está en lanzar un proceso de exploración inteligente que se basa en analizar cómo evoluciona la pendiente durante un número determinado de épocas, lo anterior logra que mezclando el RMSProp, destacado en procesos de explotación, con conceptos de la metaheurística de ascenso de colina por la máxima pendiente (proceso de exploración) se logre minimizar la entropía cruzada en las redes neuronales profundas convolucionales, indicando así que la sinergia de estas dos técnicas ayuda a escapar de problemas típicos presentes en la alta dimensionalidad.

   La investigación permite concluir que un sencillo algoritmo evolutivo (LEEA) no puede vencer al RMSProp, lo anterior, debido a que RMSProp es un algoritmo determinístico que siempre está optimizando el proceso con información de la pendiente o descenso del gradiente, mientras que los algoritmos evolutivos se basan en procesos

probabilísticos, estos procesos probabilísticos hacen que el evolutivo avance a “ciegas”, ahora, teniendo en cuenta que el entrenamiento tiene un número determinado de iteraciones, hace que el RMSProp aproveche de mejor manera el proceso de entrenamiento iteración tras iteración mientras que los evolutivos constantemente están creando nuevas soluciones que en muchos casos quedan sobre puntos de silla, platea u óptimos locales haciendo que el entrenamiento se vea afectado, razón por la cual la exploración inteligente de las propuestas meméticas tiene mayor éxito.

Los resultados permiten concluir que los tiempos de ejecución de las metaheurísticas y propuestas meméticas son similares al RMSProp, lo anterior es posible gracias a que DNF aprovecha las potencialidades de TensorFlow y su procesamiento paralelo a través de tarjetas gráficas o hilos del procesador.

Como trabajo futuro se considera importante realizar lo siguiente:

1. Realizar un proceso de afinamiento de los parámetros de las metaheurísticas de optimización global de gran escala, puesto que, los parámetros se asumieron como los mejores obtenidos en investigaciones previas, con este proceso de afinamiento se puede mejorar el entrenamiento de redes neuronales profundas utilizando neuro evolución.
2. Potencializar DNF, mediante la adición de nuevas metaheurísticas de optimización global de gran escala, así como, agregar más propuestas meméticas que tengan nuevos criterios de lanzamiento de procesos de exploracion, puesto que, esto demostró ser clave en la competencia contra el RMSProp.
3. Probar las propuestas meméticas que ganaron al RMSProp en la arquitectura de red neuronal convolucional LeNet, en otras arquitecturas convolucionales más complejas como AlexNet y sobre conjuntos de datos mucho más complejos como CIFAR 101, CalTech e ImageNet.

Capítulo 8

8 Referencias

[1] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy Layer-Wise Training of Deep Networks,” *Adv. Neural Inf. Process. Syst.*, vol. 19, no. 1, p. 153, 2007.

[2] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[3] Q. V. Le *et al.*, “Building high-level features using large scale unsupervised learning,” in *arXiv preprint arXiv:1112.6209*, 2011, pp. 8595–8598.

[4] M. A. Ranzato, Y.-L. Boureau, and Y. Lecun, “Sparse Feature Learning for Deep Belief Networks,” *Adv. neural Inf. Process. Syst.*, no. Mcmc, pp. 1185–1192, 2008.

[5] G. Morse and K. O. Stanley, “Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks,” in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference - GECCO ’16*, 2016, no. Gecco, pp. 477–484.

[6] Y. Bengio, Y. {LeCun}, and Y. Lecun, “Scaling Learning Algorithms towards AI,” *Large Scale Kernel Mach.*, no. 1, pp. 321–360, 2007.

[7] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,” *arXiv*, pp. 1–14, 2014.

[8] G. Hinton, N. Srivastava, and K. Swersky, “Neural Networks for Machine Learning, Lecture 6,” 2014. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. [Accessed: 29-Mar-2017].

[9] G.-B. G.-B. Huang, P. Saratchandran, and N. Sundararajan, “A Generalized Growing and Pruning RBF (GGAP-RBF) Neural Network for Function Approximation,” *IEEE Trans. Neural Networks*, vol. 16, no. 1, pp. 57–67, Jan. 2005.

[10] Nan-Ying Liang, Guang-Bin Huang, P. Saratchandran, and N. Sundararajan, “A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks,” *IEEE Trans. Neural Networks*, vol. 17, no. 6, pp. 1411–1423, Nov. 2006.

[11] P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks.,” *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, 1994.

[12] D. Whitley, T. Starkweather, and C. Bogart, “Genetic algorithms and neural networks: optimizing connections and connectivity,” *Parallel Comput.*, vol. 14, no. 3, pp. 347–361, 1990.

[13] Goanna.cs.rmit.edu.au, “Competition on Large Scale Global Optimization, IEEE CEC 2015,” 2015. [Online]. Available: http://goanna.cs.rmit.edu.au/~xiaodong/lsgo-competition-cec15/. [Accessed: 30-Mar-2017].

[14] S. C. Kothari and H. Oh, “Neural Networks for Pattern Recognition,” *Adv. Comput.*,

vol. 37, no. C, pp. 119–166, 1993.

[15] C. J. C. B. Y. Lecun, C. Cortes, “A theoretical framework for Back-Propagation,” *Proc. 1988 Connect. Model. Summer Sch.*, pp. 21–28, 1988.

[16] “Top 5 Deep Learning Architectures | Packt Hub.” [Online]. Available: https://hub.packtpub.com/top-5-deep-learning-architectures/. [Accessed: 02-Nov-2019].

[17] A. Gibson and J. Patterson, “Major Architectures of Deep Networks - Deep Learning [Book],” in *Deep Learning*, .

[18] J. Patterson and A. Gibson, *Deep learning : a practitioner’s approach*. .

[19] M. Z. Alom *et al.*, “A state-of-the-art survey on deep learning theory and architectures,” *Electronics (Switzerland)*, vol. 8, no. 3. p. 292, 05-Mar-2019.

[20] A. Chinea, “Understanding the principles of recursive neural networks: A generative approach to tackle model complexity,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5768 LNCS, no. PART 1, pp. 952–963.

[21] A. Ceroni, P. Frasconi, and G. Pollastri, “Learning protein secondary structure from sequential and relational data,” *Neural Networks*, vol. 18, no. 8, pp. 1029–1039, Oct. 2005.

[22] C. de Mauro, M. Diligenti, M. Gori, and M. Maggini, “Similarity learning for graph-based image representations,” *Pattern Recognit. Lett.*, vol. 24, no. 8, pp. 1115–1122, May 2003.

[23] K. Fukushima, “Neocognitron: A hierarchical neural network capable of visual pattern recognition,” *Neural Networks*, vol. 1, no. 2, pp. 119–130, 1988.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[25] Y. Xie, H. Jin, and E. C. C. Tsang, “Improving the lenet with batch normalization and online hard example mining for digits recognition,” in *International Conference on Wavelet Analysis and Pattern Recognition*, 2017, vol. 1, pp. 149–153.

[26] L. Xiao, Q. Yan, and S. Deng, “Scene classification with improved AlexNet model,” in *Proceedings of the 2017 12th International Conference on Intelligent Systems and Knowledge Engineering, ISKE 2017*, 2018, vol. 2018-Janua, pp. 1–6.

[27] G. Morse and K. O. Stanley, “Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks,” in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference - GECCO ’16*, 2016, pp. 477–484.

[28] Y. LeCun, C. Cortes, and C. J. C. Burges, “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.” 1998.

[29] Zalando, “Fashion MNIST | Kaggle,” 2017. [Online]. Available: https://www.kaggle.com/zalando-research/fashionmnist. [Accessed: 11-Nov-2019].

[30] L. Deng and D. Yu, “Deep Learning: Methods and Applications,” NOW Publishers, May 2014.

[31] P. O. Glauner, “Deep Convolutional Neural Networks for Smile Recognition,” Aug. 2015.

[32] S. Hochreiter, S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, “Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies,” 2001.

[33] J. Schmidhuber, “Learning Complex, Extended Sequences Using the Principle of History Compression,” *Neural Comput.*, vol. 4, no. 2, pp. 234–242, Mar. 1992.

[34] J. Schmidhuber, “My First Deep Learning System of 1991 + Deep Learning Timeline 1962-2013,” Dec. 2013.

[35] Y. LeCun *et al.*, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[36] Y. LeCun, “Deep Learning and the Future of AI.” [Online]. Available: https://indico.cern.ch/event/510372/. [Accessed: 30-Mar-2017].

[37] C. Szegedy *et al.*, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[38] C. Goller, C. Goller, and A. Küchler, “Learning Task-Dependent Distributed Representations by Backpropagation Through Structure,” *IN PROC. OF THE ICNN-96*, vol. 1, pp. 347--352, 1996.

[39] G. Hinton, “Deep belief networks,” *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009.

[40] R. Salakhutdinov and G. Hinton, “An Efficient Learning Procedure for Deep Boltzmann Machines,” *Neural Comput.*, vol. 24, no. 8, pp. 1967–2006, Aug. 2012.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[42] Michael Nielsen, *Neural Networks and Deep Learning*. 2015.

[43] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, “Advances in optimizing recurrent networks,” in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2013, pp. 8624–8628.

[44] Y. Nesterov, “A method of solving a convex programming problem with convergence rate O (1/k2),” in *Soviet Mathematics Doklady*, 1983, vol. 27, no. 2, pp. 372–376.

[45] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” in *COLT 2010 - The 23rd Conference on Learning Theory*, 2010, vol. 12, no. Jul, pp. 257–269.

[46] J. Dean *et al.*, “Large Scale Distributed Deep Networks,” 2012, pp. 1223–1231.

[47] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global vectors for word representation,” in *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 2014, pp. 1532–1543.

[48] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” *arXiv Prepr. arXiv1212.5701*, 2012.

[49] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Dec. 2014.

[50] G. Hinton, N. Srivastava, and K. Swersky, “Neural Networks for Machine Learning Lecture 6a Overview of mini---batch gradient descent.”

[51] D. Floreano, P. Dürr, and C. Mattiussi, “Neuroevolution: from architectures to learning,” *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, Mar. 2008.

[52] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[53] D. J. Montana and L. Davis, “Training Feedforward Neural Networks Using Genetic Algorithms,” *Proc. 11th Int. Jt. Conf. Artif. Intell. - Vol. 1*, vol. 89, pp. 762–767, 1989.

[54] E. Mingolla and D. Bullock, “Neurocomputing: Foundations of Research,” in *Neural Networks*, vol. 2, no. 5, J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1989, pp. 405–409.

[55] X. Yao, “Evolving artificial neural networks,” *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[56] W. Wienholt, “Minimizing the System Error in Feedforward Neural Networks with Evolution Strategy,” in *ICANN ’93*, London: Springer London, 1993, pp. 490–493.

[57] C. Goerick, C. Goerick, and T. Rodemann, “Evolution Strategies: An Alternative to Gradient Based Learning.”

[58] V. W. Porto, D. B. Fogel, and L. J. Fogel, “Alternative neural network training methods [active sonar processing],” *IEEE Expert*, vol. 10, no. 3, pp. 16–22, Jun. 1995.

[59] H. Mühlenbein, “Limitations of multi-layer perceptron networks - steps towards genetic neural networks,” *Parallel Comput.*, vol. 14, no. 3, pp. 249–260, Aug. 1990.

[60] D. Dasgupta and D. R. McGregor, “Designing application-specific neural networks using the structured genetic algorithm,” in *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pp. 87–96.

[61] H. Braun and J. Weisbrod, “Evolving Neural Feedforward Networks,” in *Artificial Neural Nets and Genetic Algorithms*, Vienna: Springer Vienna, 1993, pp. 25–32.

[62] R. K. Belew, R. K. Belew, J. Mcinerney, and N. N. Schraudolph, “Evolving Networks: Using the Genetic Algorithm with Connectionist Learning,” *IN*, pp. 511--547, 1990.

[63] M. Mandischer, “A comparison of evolution strategies and backpropagation for neural network training,” *Neurocomputing*, vol. 42, no. 1, pp. 87–117, 2002.

[64] D. B. Fogel and M. Kaufmann, “Blondie24, Playing at the Edge of AI,” *Publishers*, vol. 1, pp. 55860–783, 2002.

[65] K. O. Stanley and R. Miikkulainen, “Competitive coevolution through evolutionary complexification,” *J. Artif. Intell. Res.*, vol. 21, pp. 63–100, 2004.

[66] K. O. Stanley, D. D ’ambrosio, and J. Gauci, “A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks,” *Artif. Life J.*, vol. 15, no. 2, 2009.

[67] C. Igel, “Neuroevolution for reinforcement learning using evolution strategies,” in *The 2003 Congress on Evolutionary Computation, 2003. CEC ’03.*, vol. 4, pp. 2588–2595.

[68] V. K. Ojha, A. Abraham, and V. Snášel, “Metaheuristic design of feedforward neural networks: A review of two decades of research,” *Eng. Appl. Artif. Intell.*, vol. 60, pp. 97–116, 2017.

[69] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning,” Dec. 2017.

[70] A. Martín, R. Lara-Cabrera, F. Fuentes-Hurtado, V. Naranjo, and D. Camacho, "EvoDeep: A new evolutionary approach for automatic Deep Neural Networks parametrisation," *J. Parallel Distrib. Comput.*, vol. 117, pp. 180–191, Jul. 2018.

[71] A. Cano and C. Garcia-Martinez, "100 Million dimensions large-scale global optimization using distributed GPU computing," in *2016 IEEE Congress on Evolutionary Computation, CEC 2016*, 2016, pp. 3566–3573.

[72] Goanna.cs.rmit.edu.au, "Competition on Large Scale Global Optimization, IEEE CEC 2015," 2015. [Online]. Available: http://goanna.cs.rmit.edu.au/~xiaodong/lsgo-competition-cec15/. [Accessed: 23-Jul-2019].

[73] Z. Yang, K. Tang, and X. Yao, "Differential evolution for high-dimensional function optimization," in *2007 IEEE Congress on Evolutionary Computation, CEC 2007*, 2007, pp. 3523–3530.

[74] Z. Yang, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci. (Ny).*, vol. 178, no. 15, pp. 2985–2999, 2008.

[75] Zhenyu Yang, Ke Tang, and Xin Yao, "Self-adaptive differential evolution with neighborhood search," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1110–1116.

[76] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, 2010, pp. 1–8.

[77] A. Latorre, S. Muelas, and J. M. Pena, "Large scale global optimization: Experimental results with MOS-based hybrid algorithms," *2013 IEEE Congr. Evol. Comput. CEC 2013*, pp. 2742–2749, 2013.

[78] A. L. de la Fuente, "A framework for hybrid dynamic evolutionary algorithms : multiple offspring sampling (MOS)," 2009.

[79] A. LaTorre, S. Muelas, and J. M. Peña, "A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: A scalability test," *Soft Comput.*, vol. 15, no. 11, pp. 2187–2199, Nov. 2011.

[80] A. LaTorre, S. Muelas, and J. M. Peña, "Benchmarking a MOS-based algorithm on the BBOB-2010 noiseless function testbed," in *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10 - Companion Publication*, 2010, pp. 1649–1656.

[81] D. Molina and F. Herrera, "Hibridación Iterativa de DE con Búsqueda Local con reinicio para problemas de alta dimensionalidad," in *Actas de la XVI Conferencia de la Asociación Española para la Inteligencia Artificial*, 2015, pp. 251–260.

[82] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.

[83] L. Y. Tseng and C. Chen, "Multiple trajectory search for large scale global optimization," in *2008 IEEE Congress on Evolutionary Computation, CEC 2008*, 2008, pp. 3052–3058.

[84] J. L. Morales and J. Nocedal, "Remark on {\textquotedblleft}algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained

optimization{\textquotedblright},” *ACM Trans. Math. Softw*, vol. 38, no. 1, pp. 7:1--7:4, 2011.

[85] J. Liu and K. Tang, “Scaling up covariance matrix adaptation evolution strategy using cooperative coevolution,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8206 LNCS, Springer, Berlin, Heidelberg, 2013, pp. 350–357.

[86] X. Li, K. Tang, M. Omidvar, Z. Yang, K. Qin, and H. China, “Benchmark functions for the CEC’2013 special session and competition on large-scale global optimization,” *Gene*, vol. 7, no. 33, p. 8, 2013.

[87] G. Zaccone, M. R. Karim, and A. Menshawy, *Deep Learning with TensorFlow.* Packt Publishing, 2017.

[88] S. Luke, *Essentials of Metaheuristics*. 2012.

[89] D. F. Velásquez Garzón, Angie Daniela; Sotelo Montero, “Entrenamiento de una Máquina de Aprendizaje Extremo basado en un Algoritmo Meta-Heurístico Mono-Objetivo de Optimización Global Continua de Alta Dimensionalidad,” Universidad del Cauca, 2018.

[90] H. Chiroma *et al.*, “Nature Inspired Meta-heuristic Algorithms for Deep Learning: Recent Progress and Novel Perspective,” in *Advances in Intelligent Systems and Computing*, 2020, vol. 943, pp. 59–70.

[91] M. G. H. Omran, A. Salman, and A. P. Engelbrecht, “Self-adaptive differential evolution,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2005, vol. 3801 LNAI, pp. 192–199.

[92] A. LaTorre, S. Muelas, and J. M. Peña, “Multiple offspring sampling in large scale global optimization,” in *2012 IEEE Congress on Evolutionary Computation, CEC 2012*, 2012, pp. 1–8.

[93] Y. Sazaki, A. Primanita, H. Satria, and R. Apriliansyah, “A comparison application of the genetic and steepest ascent hill climbing algorithm in the preparation of the crossword puzzle board,” in *Proceeding of 2018 12th International Conference on Telecommunication Systems, Services, and Applications, TSSA 2018*, 2019, pp. 1–5.

[94] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[95] Y. LeCun and C. Cortes, “{MNIST} handwritten digit database,” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/.