Universidad de las Ciencias Informáticas

Facultad de Ciencias y Tecnologías Computacionales

Algoritmos meta-heurísticos de optimización aplicados al entrenamiento de redes neuronales profundas

Tesis presentada en opción al título de Máster en Informática Avanzada

Autor: Ing. Jairo Rojas Delgado

Tutor: Dr.C. Rafael Trujillo Rasúa

Co-tutor: Dr.C. Rafael Bello Pérez

La Habana, agosto de 2018

para quienes dedican sus esfuerzos a la

construcción de un mundo mejor...

DECLARACIÓN JURADA DE AUTORÍA

Declaro por este medio que yo Jairo Rojas-Delgado, con carné de identidad 91060735307, soy el autor principal del trabajo final de maestría "Algoritmos de optimización meta-heurísticos aplicados al entrenamiento de redes neuronales profundas", desarrollada como parte de la Maestría en Informática Avanzada y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Y para que así conste, firmo la presente declaración jurada de autoría en La Habana a los ____ días del mes de _____ del año 2018

_____

Firma del Maestrante

RESUMEN

La reciente introducción de las redes neuronales profundas ha provocado un resurgimiento en el interés por las redes neuronales artificiales. Las redes profundas son de importancia significativa para la resolución de un grupo importante de tareas, sin embargo, su entrenamiento presenta retos teóricos y prácticos al tratarse de un problema de optimización NP-duro. Específicamente, el algoritmo Gradiente Descendente Estocástico se enfrenta a cambios de orden cualitativo y cuantitavo a considerar. En este trabajo se propone el empleo de algoritmos de optimización meta-heurísticos para el entrenamiento de Redes Neuronales Profundas en paralelo. Se considera un algoritmo basado en métodos de continuación para disminuir el tiempo de ejecución del entrenamiento mediante meta-heurísticas y aumentar la precisión respecto al algoritmo Gradiente Descendente Estocástico. Los resultados obtenidos muestran que el entrenamiento mediante algoritmos meta-heurísticos obtiene mayor precisión que el entrenamiento mediante Gradiente Descendente Estocástico y que es posible disminuir el tiempo de ejecución de los algoritmos meta-heurísticos empleando un enfoque basado en continuación.

Palabras clave: red-neuronal, entrenamiento, meta-heurística, continuación

ABSTRACT

The recent introduction of deep neural networks has caused an increase in the interest by artificial neural networks. Deep neural networks are of paramount importance in the resolution of several important tasks, however, training a neural network is a challenging NP-hard optimization problem. Specifically, Stochastic Gradient Descent faces quantitative and qualitative changes to consider. In this work, meta-heuristic algorithms are proposed to perform neural network training in parallel, considering a continuation approach algorithm to efficiently reduce execution time and improve the accuracy of training respect Stochastic Gradient Descent. Results show that meta-heuristic training is capable to obtain higher accuracy than Stochastic Gradient Descent and when used with a continuation approach is possible to reduce execution time of training.

Keywords: *neural-network, training, meta-heuristic, continuation*

Índice general

Introducción

Las Redes Neuronales Artificiales (RNA) fueron introducidas por primera vez en el año 1943 por Warren McCulloch y Walter Pitts [1]. Posterior a la publicación del trabajo *Perceptrons* en 1969 por Marvin Minsky y Seymour Papert, la comunidad científica perdió el interés en el tema debido a la presentación de un grupo de limitaciones importantes. No fue hasta la década de 1980 a partir de una serie de avances teórico-prácticos y especialmente el descubrimiento del error de retro-propagación que los esfuerzos investigativos volvieron a enfocarse en el área [2].

Recientemente ha existido un re-surgimiento en el interés por las RNA con la introducción de las Redes Neuronales Profundas (RNP). Las RNP se conforman mediante capas de neuronas en cascada permitiendo transformar una señal de entrada en conceptos más abstractos y de alto nivel [3]. Esta transformación de la señal de entrada dentro de la estructura de una RNP permite realizar inferencias y generalizaciones con mayor precisión.

Las RNP son de gran importancia en el campo del reconocimiento de patrones y específicamente en el aprendizaje de representaciones de los datos. Una de las promesas fundamentales de las RNP y el denominado aprendizaje profundo no es solamente la obtención de mayor precisión, sino también proveer de procedimientos completos que realizen automáticamente lo que se ha dado a llamar ingeniería de atributos: extracción de rasgos, selección de rasgos, etc.

El campo de investigaciones referido a las RNP es uno de los más activos en la comunidad científica con múltiples aplicaciones recientes en actividades como el reconocimiento de imágenes [4, 5], el procesamiento de señales [6] y el reconocimiento de voz [7, 8]. De forma general, las RNA resultan atractivas para la resolución de una serie de problemas bien establecidos que surgen dentro de la disciplina de inteligencia artificial como el agrupamiento, la clasificación, la regresión, la aproximación de funciones y otros.

Coloquialmente, un problema de optimización se define sobre un conjunto de parámetros y una función objetivo donde es necesario encontrar la combinación de parámetros que minimize o maximize el valor de la función objetivo. El entrenamiento de una RNA es considerado un

problema de optimización NP-duro [9].

Los algoritmos de optimización de primer orden y segundo orden son aquellos que utilizan información de la primera y segunda derivada de la función objetivo respectivamente. Entre los algoritmos de entrenamiento de mayor adopción en la literatura para el entrenamiento de RNA se encuentran los métodos de primer orden, especialmente el Gradiente Descendente Estocástico (SGD por sus siglas en inglés) o variaciones de este. SGD ha sido caracterizado por ser un algoritmo secuencial [10] muy difícil de paralelizar. Igualmente se ha mostrado que los métodos de entrenamiento de primer orden tienen una alta probabilidad de convergencia local, lo que provoca una disminución significativa en la precisión. De manera general, los métodos de segundo orden no han mostrado obtener resultados significativamente mejores que los métodos de primer orden [3].

Adicionalmente, el entrenamiento de RNP presenta un grupo de cambios cuantitativos y cualitativos respecto al entrenamiento de RNA. En la literatura es común encontrar referencias a estos cambios y soluciones específicas a los problemas que surgen en esta área de investigación. Considerar estos cambios de manera sistemática se convierte en una tarea de vital importancia para encontrar soluciones teóricas y prácticas a la dificultad del entrenamiento profundo.

Uno de los factores que ha influenciado el desarrollo del aprendizaje profundo ha sido la consolidación de *hardware* paralelo a relativamente bajo costo. El aumento de las capacidades de cómputo ha posibilitado el aumento acelerado de la cantidad de parámetros y patrones de entrenamiento de las RNP como una metodología base para incrementar la precisión de estos modelos [11, 12, 13, 14].

Existen estudios relacionados con el empleo de algoritmos de optimización meta-heurísticos alternativos a los métodos de primer y segundo orden. En los últimos años se ha visto un incremento en las propuestas de estos algoritmos inspirados en la naturaleza y el comportamiento de los seres vivos. Un algoritmo de optimización meta-heurístico permite explorar un espacio de búsqueda mediante prueba y error implicando usualmente algún tipo de aleatoriedad. El principal criterio que define el éxito de estos algoritmos de optimización se basa en equilibrar la exploración del espacio de búsqueda y la explotación de la información local.

Los algoritmos meta-heurísticos exhiben usualmente un comportamiento paralelo al ser inspira-

dos en entidades biológicas independientes. Desde otra perspectiva, han sido aplicados con éxito a problemas no-convexos con una elevada cantidad de mínimos locales. En el campo de la optimización de RNA, existen reportes de algoritmos evolutivos [15, 16] y más recientemente otras meta-heurísticas bio-inspiradas como Cuckoo Search [17, 18, 19], Firefly [20, 21], Wolf Optimization [22], Ant Colony Optimization [15, 23], Particle Swarm Optimization [24], Multi-verse Optimization [25] e híbridos como [26].

A pesar de la proliferación de estudios teórico-prácticos pertenecientes a la rama de optimización basada en métodos de primer y segundo orden para el entrenamiento de RNP, la optimización meta-heurística aplicada al entrenamiento de RNP ha sido abordada en menor medida. Los estudios es esta área se refieren entrenamiento de RNA convencionales y problemas de regresión y/o clasificación, mientras que otros tipos de de RNA y problemas se estudian en menor medida. Dichas redes poseen poca cantidad de parámetros y no presentan superficies de optimización con las características de las RNP. Prolifera la configuración de hiper-parámetros de los algoritmos de entrenamiento de manera manual y trabajos que estudian problemas con poca cantidad de patrones de entrenamiento.

En Cuba, bajo el Programa Rector para la Informatización de la Sociedad, desde hace varios años se ha puesto en marcha un grupo de proyectos destinados a la investigación, desarrollo y asimilación tecnológica. En las bases teóricas del Modelo Económico y Social Cubano de Desarrollo Socialista del año 2016 se encuentra expresada la necesidad de acelerar avances en las comunicaciones y la informatización.

Siendo esto, en el estado del arte pueden identificarse al menos dos limitaciones relacionadas con el entrenamiento de RNP. La primera se relaciona con la necesidad de contar con procedimientos de entrenamiento que permitan tratar eficientemente la elevada complejidad computacional del entrenamiento de las RNP. La segunda se relaciona con la necesidad de contar con algoritmos de entrenamiento capaces de asimilar eficazmente los cambios del entrenamiento profundo.

Problema científico

De esta forma se plantea el siguiente problema científico: ¿Cómo entrenar eficientemente Redes Neuronales Profundas para obtener modelos con precisión superior a los entrenados mediante

Gradiente Descendente Estocástico?

El problema de investigación mencionado se encuentra contenido en el **objeto de estudio** del proceso de entrenamiento de Redes Neuronales Artificiales.

Objetivo general

El presente trabajo posee como objetivo: Desarrollar algoritmos paralelos basados en meta-heurísticas de optimización para entrenar Redes Neuronales Profundas con mayor precisión que las entrenadas mediante Gradiente Descendente Estocástico.

El **campo de acción** se enmarca en los algoritmos de optimización paralelos meta-heurísticos aplicados al entrenamiento de Redes Neuronales Profundas. Teniendo en cuenta lo anterior, se define la siguiente **hipótesis de investigación**: Si se desarrollan algoritmos paralelos para entrenar Redes Neuronales Profundas basados en meta-heurísticas de optimización, entonces es posible entrenar eficientemente Redes Neuronales Profundas con precisión superior a las entrenadas mediante el algoritmo Gradiente Descendente Estocástico.

A partir de la hipótesis declarada se identifica la variable independiente algoritmo de entrenamiento y las variables dependendientes precisión y eficiencia. En el Capítulo(3) se detalla la operacionalización de las variables abordadas en este trabajo.

Tareas de investigación

Las siguientes tareas de investigación definen la estructura y guían el trabajo a realizar:

1. Elaboración de un marco teórico referencial sobre el entrenamiento de RNA como un problema de optimización y los cambios del aprendizaje profundo para caracterizar los algoritmos meta-heurísticos de optimización aplicados al entrenamiento profundo.

2. Desarrollo de algoritmos paralelos basados en meta-heurísticas de optimización considerando los cambios del entrenamiento profundo para entrenar RNP con precisión superior a las entrenados mediante SGD.

3. Implementación de una biblioteca con variantes en paralelo de los algoritmos desarrollados para su análisis y aplicación a problemas de predicción estudiados comúnmente en la literatura especializada.

4. Análisis de los algoritmos propuestos en términos de eficiencia y precisión para estudiar sus propiedades y encontrar diferencias estadísticamente significativas respecto a SGD.

Métodos de investigación

Para dar cumplimiento al objetivo se han empleado un grupo de métodos teóricos y empíricos de la investigación científica, dentro de los cuales se encuentran:

Del nivel teórico

1. Analítico-sintético en el estudio de la literatura relacionada con el entrenamiento de RNP. Mediante este método fue posible obtener los elementos más relevantes, así como evaluar, analizar y sintetizar conceptos que ayudaron a la comprensión del problema.

2. Histórico-lógico en la revisión exhaustiva del desarrollo evolutivo del entrenamiento de RNA a través del tiempo con el objetivo de definir las limitaciones actuales de su conocimiento. El método permitió reconocer los avances teórico-prácticos y problemas que actualmente existen en el área del entrenamiento de RNA.

3. Hipotético-deductivo en la definición de la línea de investigación y las conjeturas primarias. El método permitió definir las variables involucradas en el estudio así como sus relaciones más importantes.

4. Sistémico-estructurado en la comprensión del entrenamiento de RNA y los cambios recientes del entrenamiento de RNP como sistema, así como las funciones y estructura de los diferentes componentes que lo caracterizan.

Del nivel empírico

1. Análisis documental en la revisión de la literatura especializada para consultar la información necesaria en el proceso de investigación.

2. Observación en la evaluación del comportamiento de los algoritmos para el entrenamiento de RNP.

3. Experimental en la comprobación de los resultados obtenidos en condiciones controladas y en el establecimiento de comparaciones con los resultados obtenidos por los distintos algoritmos de entrenamiento de RNP.

Aporte práctico

El aporte práctico de la presente investigación consiste en la implementación de una biblioteca con variantes en paralelo de algoritmos meta-heurísticos de optimización. Esto permitirá realizar el entrenamiento de RNP de forma eficiente y precisa contribuyendo al análisis y resolución de problemas de predicción que surgen en múltiples áreas de desarrollo de alcance económico y social.

Estructura del documento

El documento se encuentra estructurado en introducción, tres capítulos, conclusiones, recomendaciones y bibliografía. Los temas trabajados en cada capítulo se describen a continuación:

1. Capítulo 1: Fundamentación teórica. Se presenta un grupo de consideraciones referidas al entrenamiento neuronal, los cambios entre el entrenamiento de RNP y el entrenamiento de RNA convencionales y finalmente se presentan los resultados más importantes en la aplicación de algoritmos meta-heurísticos de optimización aplicados al entrenamiento de RNA.

2. Capítulo 2: Propuesta de solución. Se introducen cambios y modificaciones a los enfoques y algoritmos de optimización basados en meta-heurísticas aplicados al entrenamiento de RNA. Especialmente, se presenta un nuevo algoritmo basado en meta-heurísticas de optimización para el entrenamiento de RNA.

3. Capítulo 3: Pruebas y análisis de los resultados. Se describen las pruebas y configuraciones

experimentales realizadas para las mediciones del tiempo de ejecución y precisión de los algoritmos estudiados. Se presenta un análisis de los resultados obtenidos.

Capítulo 1

Fundamentos teóricos y estudio diagnóstico

Las RNA son modelos matemáticos computacionales inspirados en el funcionamiento del cerebro. Desde su surgimiento, desarrollo y consolidación su uso se ha expandido notablemente en la resolución de problemas teóricos y prácticos dentro de la disciplina de Inteligencia Artificial. Entre las tareas de mayor dificultad a considerar se encuentra el entrenamiento, definido como un problema de optimización NP-duro que define en buena parte la precisión de estos modelos. De ahí la importancia del estudio de la optimización meta-heurística y métodos de optimización en general en esta área.

En este capítulo se presenta un grupo de consideraciones respecto al entrenamiento neuronal y los conceptos asociados. Posteriormente, se aborda sobre el origen y complejización del entrenamiento al tratar RNP como la evolución de las RNA. Luego se realiza un análisis de los algoritmos de optimización meta-heurísticos aplicados al entrenamiento de RNP y los retos más importantes encontrados en la literatura. Finalmente se describen las conclusiones parciales del capítulo.

1.1. El procesamiento neuronal artificial

Las RNA, también conocidas como modelos conexionistas, surgieron en 1943 introducidas por McCulloch y Pitts [1]. De forma general, las RNA son modelos matemáticos computacionales inspirados en el funcionamiento del cerebro. El funcionamiento del cerebro todavía es un misterio para la ciencia moderna. A pesar que los seres humanos no comprenden aún algunos fenómenos naturales del universo, es posible construir modelos con utilidad práctica.

De acuerdo a [27] existen dos líneas de investigación referidas al descubrimiento de modelos artificiales del funcionamiento del cerebro. La primera de estas líneas se interesa en modelar y estudiar procesos de aprendizaje biológicos. La segunda línea se interesa por encontrar algoritmos

de aprendizaje por computadoras altamente eficientes, independientemente de si estos simulan con precisión procesos biológicos naturales.

Los elementos más básicos de un cerebro son un tipo específico de célula llamada neurona. El cerebro humano posee como promedio unos 100 billones de neuronas y son las responsables de nuestras habilidades para recordar, pensar y aplicar la experiencia adquirida en la resolución de problemas. Cada neurona se encuentra conectada a lo sumo con otras 200 mil, aunque es típico que una de ellas se conecte con un número que oscila entre las mil y 10 mil.

Las neuronas de forma individual son complicadas. Cada uno de estos pequeños procesadores poseen varias partes, sub-sistemas y mecanismos de control. Procesan información mediante conexiones electro-químicas. La ciencia moderna ha identificado unos 100 tipos diferentes de neuronas. En conjunto, las neuronas y la forma en la que se conectan, conforman un proceso que aún no es completamente entendido y no es similar al procesamiento de las computadoras modernas ni a la concepción artificial de los modelos neuronales con que hoy se cuenta.

Una perspectiva común para la caracterización de las RNA es la idea del Procesamiento Paralelo Distribuido [28]. Bajo esta perspectiva las RNA son variaciones de un modelo de procesamiento paralelo distribuido que se caracteriza por entre otros factores: unidades de procesamiento o neuronas, un patrón de conectividad y una regla de aprendizaje.

1.1.1. Los modelos de neuronas artificiales

El elemento fundamental de procesamiento de una red neuronal es la neurona. Una neurona biológica recibe una señal de entrada de fuentes externas y la combina de alguna forma para aplicar una operación no lineal sobre esa entrada y producir una salida. Luego de este simple procesamiento, la señal de salida se propaga hacia otras unidades de procesamiento.

Las investigaciones actuales corroboran y coinciden en que las neuronas biológicas son, estructural y funcionalmente, mucho más complejas que los modelos artificiales actuales. Esta brecha permite, a los investigadores en el área, mejorar continuamente dichos modelos en la medida que la biología permite comprender mejor el funcionamiento de estas estructuras. De hecho, nuevos aportes y potenciales aplicaciones son descubiertos cada día provenientes de múltiples campos de investigación asociados a las neurociencias [29].

Entre los modelos artificiales de neuronas más simples que son estudiadas en la bibliografía se encuentran las neuronas de tipo *sigma-unit*. Este tipo de neurona artificial, en lo siguiente simplemente unidad de procesamiento, está compuesta por una entrada neta y una función de activación. En la Figura(1.1) se describe de forma general el modelo artificial de neurona *sigma-unit*.

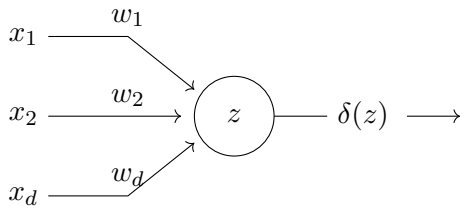


Figura 1.1: El modelo artificial de una neurona compuesto por una función de transferencia que realiza una combinación de la entrada y una función de activación.

En la Figura(1.1) cada entrada x_i tiene asociado un parámetro w_i que mide su contribución a la entrada neta de la neurona z. Posteriormente se aplica una función $\delta(z)$ sobre dicha entrada neta produciendo el valor de activación de la neurona. En la Ecuación(1.1) se muestra la forma de calcular la entrada neta de una unidad de procesamiento de tipo *sigma-unit*.

$$z = \sum_{j=1}^{d} w_j x_j + \theta \tag{1.1}$$

En la Ecuación(1.1), w_j es el peso o parámetro asociado a la entrada x_j, θ es el *bias* asociado a la unidad de procesamiento y d es la cantidad de entradas. En la literatura también se identifican en menor medida otras variantes de unidades de procesamiento que utilizan otras funciones para realizar la combinación de sus entradas y calcular la entrada neta. Un ejemplo de esto son las unidades de tipo *sigma-pi* que fueron propuestas por primera vez en [30].

Igualmente, en la literatura se identifica una variedad considerable en los tipos de funciones de activación. Frecuentemente se emplea una función derivable no decreciente definida sobre la entrada neta de la neurona. A continuación se mencionan algunas de las funciones de activación que comúnmente se emplean en el campo del procesamiento neuronal.

1. Función identidad:

$$\delta(z) = z \tag{1.2}$$

2. Función *sigmoid*:

$$\delta(z) = \frac{1}{1 + \exp(-z)} \tag{1.3}$$

3. Función tangente hiperbólica:

$$\delta(z) = \tanh(z) \tag{1.4}$$

En la literatura existe una gran variedad de criterios, análisis y recomendaciones alrededor de las funciones de activación. Por ejemplo, se ha estudiado que la función de activación *sigmoid* se satura rápidamente para elevados valores de la entrada neta. La recomendación por defecto en la última década, especialmente para las RNP parece ser un tipo de función de activación denominado ReLU[1] [31, 32]. ReLU es un término que suele usarse indistintamente para referirse a una unidad de procesamiento con una función de activación como en le Ecuación(1.5) o a la función de activación en sí.

4. Función ReLU:

$$\delta(z) = \text{máx}(0, z) \tag{1.5}$$

Al considerar más de una unidad de procesamiento en el marco de una RNA, la salida de cada unidad de procesamiento es propagada hacia otras unidades de procesamiento y es usada como entrada de estas o como la salida de la red. La forma en que cada unidad de procesamiento se conecta con el resto determina lo que se conoce como patrón de conectividad o arquitectura de la red. Muchas de las contribuciones y especificidades de las RNP provienen de la forma de interpretar y construir estas interconexiones entre las unidades de procesamiento.

1.1.2. Patrones de conectividad

La forma de diseñar la arquitectura, patrón de conectividad o modelo de una RNA es un problema de diseño y usualmente se trata como uno o varios hiper-parámetros del algoritmo. Estos términos se utilizan indistintamente para referirse a la forma en que las unidades de procesamiento se conectan unas a otras. En la literatura existen al menos dos modelos de importancia de RNA, las redes de propagación hacia adelante[2] y las redes recurrentes.

[1] ReLU: Siglas del inglés *Rectified Lineal Unit*.
[2] Red de propagación hacia adelante: Del término en inglés *feed-forward*.

1. **Redes de propagación hacia adelante:** La red se conforma mediante capas de unidades de procesamiento en cascada. Las unidades de procesamiento en una misma capa no comparten conexiones entre ellas. La entrada de la primera capa está constituida por los patrones de entrenamiento y su salida conforma la entrada de la siguiente capa. La señal de entrada de la red es procesada iterativamente por cada capa hasta que la salida de la última capa produce la salida de la red.

2. **Redes recurrentes:** A diferencia de las redes de propagación hacia adelante, la salida de una capa en una red recurrente puede constituir la entrada de una capa anterior en la jerarquía. Esto introduce un grupo de propiedades dinámicas interesantes en el comportamiento de la red. Las redes recurrentes son mayormente empleadas en el procesamiento de secuencias donde la relación entre elementos sucesivos en la secuencia tiene importancia. En la práctica, las redes recurrentes pueden desplegarse en un grafo computacional acíclico similar al de una red de propagación hacia adelante.

El concepto de aprendizaje profundo es una idea que trasciende un patrón de conectividad en concreto y a la misma vez guarda una estrecha relación con este. El término profundo proviene del hecho de acumular múltiples capas de unidades de procesamiento en cascada, de ahí que, a mayor cantidad de capas mayor profundidad.

Para que un algoritmo de aprendizaje automático pueda realizar generalizaciones e inferencias correctamente, debe ser guiado por suposiciones sobre la estructura de la función que deben aprender. Por ejemplo, entre estas se encuentra la suposición de suavidad o constancia local donde se asume que, conociendo un punto específico de la función, entonces es posible conocer su vecindad.

El aprendizaje profundo parte de la suposición de que la función a aprender se encuentra definida jerárquicamente de forma tal que es posible construir conceptos sobre conceptos arbitrariamente. De acuerdo a [33] esta aparentemente simple suposición introduce un grupo de beneficios que contrarrestan los retos presupuestos por el fenómeno de la maldición de la dimensionalidad[3].

En la literatura es posible identificar patrones de conectividad entre capas de neuronas especializados en tareas específicas. De esta forma pueden mencionarse, las capas totalmente conexas,

[3]Maldición de la dimensionalidad: Del término en inglés *curse of dimensionality*.

parcialmente conexas, convolucionales, capas de agrupación[4] , entre otras. Referirse a [3] para una ampliación en el tema. En la Figura(1.2) se ilustra la estructura de una capa totalmente conexa donde cada unidad de procesamiento recibe entradas provenientes de la capa anterior.

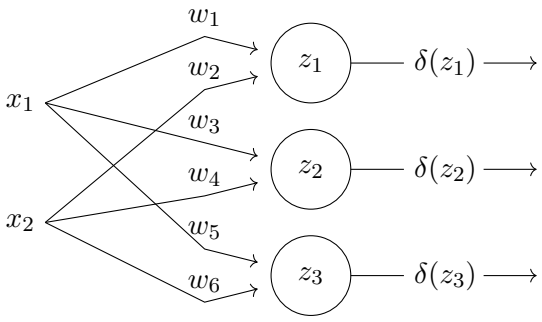


Figura 1.2: Capa totalmente conexa con tres unidades de procesamiento que reciben dos entradas provenientes de la capa anterior.

1.1.3. Regla de aprendizaje o algoritmo de entrenamiento

Coloquialmente, se dice que un algoritmo es de aprendizaje automático cuando dado una medida de su desempeño f y algún tipo de experiencia E sobre una tarea T, el desempeño del algoritmo mejora con la experiencia E medida según f [27]. En la literatura no existe consenso en cuanto a la naturaleza de las tareas T de las cuales se encarga el aprendizaje automático. Sin embargo, en algunos trabajos como [3], se presenta un grupo de tareas comunes identificadas en la práctica.

La representación de la experiencia en forma vectorial, es un problema abierto en el reconocimiento de patrones sobre el cual existe debate [34]. No obstante, la experiencia suele codificarse en forma de un vector $x \in \mathbb{R}^d$ también denominado patrón de entrenamiento o ejemplo.

Las medidas de precisión f por lo general dependen de la tarea T a resolver y permiten evaluar el comportamiento del algoritmo de aprendizaje automático. En dependencia de T, la medida de precisión suele llamarse en la literatura como precisión, ratio de error, etc. En lo siguiente, estos términos se usarán indistintamente. En el caso de las RNA como algoritmos de aprendizaje automático, el entrenamiento o aprendizaje, se concibe como el proceso de encontrar el mejor conjunto de parámetros $w \in \mathbb{R}^n$ de la red que mejoran f.

Específicamente, dado un conjunto de parámetros $w \in \mathbb{R}^n$, un conjunto con patrones de entrena-

[4]Capas de agrupación: Del término en inglés *pooling-layer*.

miento $x = \{x^1, ..., x^p\}$ donde por lo general $x^i \in \mathbb{R}^d$ y una medida de precisión $f(w, x)$ que mide el error de la red, el entrenamiento neuronal puede ser formulado de acuerdo a la Ecuación(1.6).

$$w = \underset{w \in \mathbb{R}^n}{\operatorname{argmin}} f(w, x) \tag{1.6}$$

Se ha demostrado que el entrenamiento neuronal es un problema de optimización NP-duro [9]. A diferencia de un problema de optimización convencional, en aprendizaje automático resulta más importante conocer la precisión del algoritmo en ejemplos o patrones de entrenamientos no empleados en el entrenamiento. De ahí que, para el cálculo de la precisión, se emplee un conjunto de patrones de entrenamiento llamado **conjunto de prueba**, diferente del conjunto de patrones de entrenamiento consecuentemente llamado **conjunto de entrenamiento**.

Al error del algoritmo obtenido mediante el conjunto de prueba se le denomina error de prueba y al error obtenido mediante el conjunto de entrenamiento se le denomina error de entrenamiento. Es posible minimizar el error de prueba mediante la minimización del error de entrenamiento debido a la suposición de que los ejemplos de los conjuntos de prueba y entrenamiento son independientes e idénticamente distribuidos [33].

1.2. Cambios en el entrenamiento profundo

En la naturaleza nada se encuentra fijo o estático, sino que se encuentra en constante evolución, cambio y transformación. Los objetos en la naturaleza poseen propiedades medibles que les definen, de ahí que su esencia cualitativa sea inseparable de dichas propiedades medibles.

Del materialismo dialéctico, la ley de la acumulación de cambios cuantitativos y su transformación en cambios cualitativos, predice que el incremento en la dimensionalidad del entrenamiento neuronal ha de provocar necesariamente un cambio en la esencia del propio fenómeno. Un cambio profundo en el área cualitativa implica, en ocasiones, una conceptualización distinta del fenómeno en cuestión y de ahí la importancia de su estudio.

Cada cambio cuantitativo observado en la naturaleza es el resultado de una lucha entre efectos contrarios por encontrar un punto de equilibrio. Lo anterior, conceptualizado también en el materialismo dialéctico como la ley de lucha entre contrarios, explica en cierta medida por qué

y cómo el entrenamiento de RNP ha sido posible solo recientemente.

En los últimos años se ha visto que el incremento en la dimensionalidad del entrenamiento neuronal, visto como un caso general del entrenamiento de RNP, ha estado restringido por varios factores. Se cree que, en la reciente resolución o resolución parcial de dichos factores, las RNP han provocado cambios importantes en el estado del arte en áreas como la ingeniería, la industria y la ciencia.

En este epígrafe se profundiza en cómo la contradicción de factores ha posibilitado que el aprendizaje profundo sea posible en la práctica y los cambios en el orden cuantitativo y cualitativo que esto ha provocado. La comprensión sistemática de estos cambios es de vital importancia para encontrar soluciones teóricas y prácticas a la dificultad del entrenamiento de RNP.

1.2.1. Cambios cuantitativos

Al realizar un análisis de los últimos años en el área de investigación de las RNA, es posible constatar que la cantidad de unidades de procesamiento empleadas se ha ido duplicando cada dos años y medio. De acuerdo a varios autores, esta tendencia parece sostenerse al menos por los próximos años.

En la Figura(1.3) adaptada de [3] a la izquierda se muestra la evolución histórica de la cantidad de unidades de procesamiento en el entrenamiento de RNA graficada en escala logarítmica. Consecuentemente la cantidad de parámetros depende del patrón de conectividad de la red, y se encuentra acotada por la cantidad de unidades de procesamiento multiplicada por la cantidad de conexiones entre ellas.

De manera análoga, es posible realizar un análisis similar respecto a la cantidad de patrones de entrenamiento en bases de datos de referencia internacional. El incremento en el tamaño de las bases de datos de aprendizaje es una consecuencia directa del incremento en la cantidad de parámetros de los algoritmos de aprendizaje automático y especialmente de las RNP.

En la Figura(1.3) a la derecha se muestra la evolución histórica del tamaño de las bases de datos de aprendizaje. Puede verse, que a pesar del notable incremento, el tamaño de las bases de datos de aprendizaje aún es pequeño en cuanto a cantidad de patrones de entrenamiento [35, 13, 4, 12] debido a que la construcción de estas bases de datos es por lo general un proceso costoso.

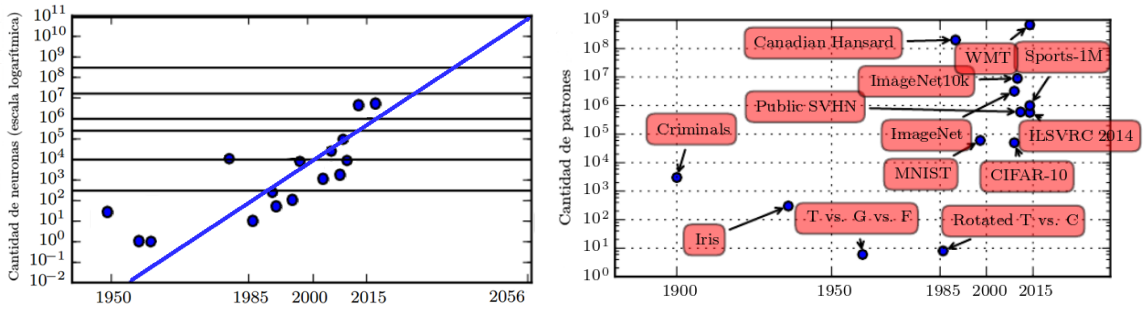


Figura 1.3: Aumento de la cantidad de unidades y patrones de entrenamiento en el procesamiento neuronal.

El aumento de la cantidad de parámetros y la cantidad de patrones de entrenamiento en el marco de las RNP, ha posibilitado obtener tasas de precisión sin precedentes en un número importante de tareas. En la Figura(1.4) tomada de [12] se ilustra el efecto de incrementar la cantidad de patrones de entrenamiento y la cantidad de parámetros en la precisión de las RNP. Los datos fueron recolectados a partir de RNP con mayor precisión en varios problemas de aprendizaje.

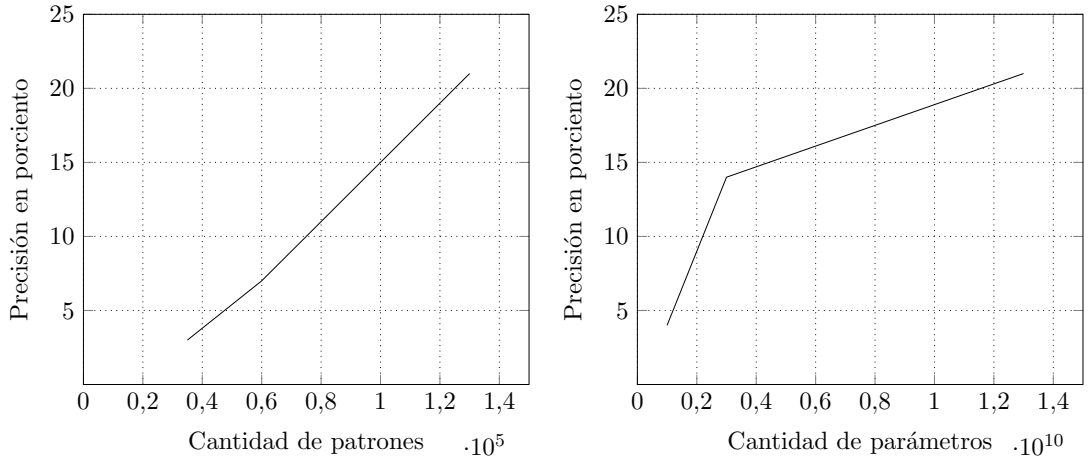


Figura 1.4: Aumento de la precisión de las RNP al aumentar la cantidad de patrones y la cantidad de parámetros.

En el aprendizaje profundo, muchos problemas involucran optimización. De acuerdo a [3] el más complejo de todos los problemas de optimización relacionados con el aprendizaje profundo es el entrenamiento. Por ejemplo, en las RNP especializadas en visión por computadora, cualquier incremento uniforme en la cantidad de filtros de dos capas convolucionales consecutivas representa un aumento cuadrático en la complejidad de los cálculos [36].

El incremento en la cantidad de patrones de entrenamiento y la cantidad parámetros de las RNP ha sido posible en gran medida al aumento del poder de cómputo del hardware moderno. Específicamente, el aumento de la cantidad de núcleos de ejecución en hardware paralelo, a relativamente bajo costo, ha sido uno de los elementos de mayor peso en el desarrollo de las RNP [3]. De esta forma, en la literatura es posible encontrar reportes de implementaciones de RNP en clústeres de Unidades Centrales de Procesamiento [11, 12], Unidades Gráficas de Procesamiento [13, 14] y hardware especializado [37, 38].

La estrecha relación entre la cantidad de parámetros, el tamaño de las bases de datos y la precisión se explica en un grupo de cambios de orden cualitativo. Por ejemplo, el fenómeno de la maldición de la dimensionalidad provoca que los patrones de entrenamiento se conviertan rápidamente en dispersos cuando la cantidad de parámetros se incrementa. En el siguiente sub-epígrafe se abordan sobre los cambios en el orden cualitativo que condicionan y son condicionados por estos cambios cuantitativos.

1.2.2. Cambios cualitativos

Además de los cambios cuantitativos observados en el entrenamiento de RNP, en la literatura es posible encontrar un grupo de cambios en al área cualitativa. Estos cambios a su vez, producen modificaciones y alteraciones cuantitativas condicionando el proceso de desarrollo natural de las cosas. En el presente sub-epígrafe se discute sobre estos cambios cualitativos que han sido estudiados a menudo de forma separada e independientemente y afectan el entrenamiento de RNP.

El incremento en la cantidad de parámetros de entrenamiento, que caracteriza el entrenamiento de RNP, provoca que en el proceso de exploración del espacio de búsqueda la información sobre este se convierta en dispersa. Entiéndase por información los patrones de entrenamiento. En función de obtener mediciones estadísticamente significativas y resultados confiables, la cantidad de patrones de entrenamiento necesaria crece de manera exponencial respecto a la dimensión de optimización. Esto se corresponde con las observaciones realizadas en el marco de la paradoja de Hughes [39] y en cómo el problema del sobre-entrenamiento adquiere nuevas características al considerar las RNP.

En otra dirección, los enfoques de optimización aplicados al entrenamiento neuronal se basan en el cálculo de una función objetivo que es básicamente una función definida sobre la señal de entrada o los patrones de entrenamiento. Una restricción fundamental es que dicha función objetivo debe ser trivial para de esta forma evitar exceder los límites computacionales prácticos. Sin embargo, el aumento en la cantidad de parámetros de las RNP agravado por el crecimiento exponencial en la cantidad de patrones de entrenamiento, provocan que la función objetivo en el entrenamiento neuronal sea costosa en términos de complejidad temporal y espacial.

De manera similar, existe un grupo de algoritmos de optimización basados en población que emplean algún tipo de métrica para calcular la diferencia entre individuos. Ha sido mostrado, que en configuraciones de elevadas dimensiones los objetos aparentan ser similares y diferentes en múltiples formas. En sentido general, la pérdida de significado semántico de las métricas de distancia en elevadas dimensiones previene el uso de estrategias de organización de los datos de forma eficiente. Esto se traduce en efectos negativos en la velocidad de convergencia y precisión del entrenamiento.

Las superficies de optimización de las RNP presentan una elevada no-convexidad y se ha visto una disparidad entre su estructura local y su estructura global [3]. Este es un efecto crítico que provoca una disminución considerable de la velocidad de convergencia y precisión de la mayoría de los algoritmos de optimización. Consecuentemente, se ha descubierto un incremento exponencial de la cantidad de mínimos locales y otras anomalías respecto a la cantidad de parámetros. Este es el origen del ampliamente conocido problema de convergencia local del entrenamiento de RNP.

El problema de la dimensionalidad

El principio de parsimonia define el empleo de modelos y procedimientos que contienen todo lo necesario para construir soluciones de la forma más sencilla posible. De acuerdo a [40], el sobre-entrenamiento es usar modelos y procedimientos que violan el principio de parsimonia, dicho de otra forma, hacen uso de más términos y parámetros de lo necesario o usan enfoques más complicados de lo necesario.

Según el mismo autor [40] el sobre-entrenamiento no es deseable por un grupo de razones. Por

ejemplo, al considerar un problema de selección de rasgos, los modelos que incluyen predictores innecesarios provocan malas decisiones. El uso de predictores irrelevantes provocan una pérdida en la precisión porque los coeficientes asociados añaden una variación aleatoria en las subsecuentes predicciones.

Con el surgimiento de las RNP el problema del sobre-entrenamiento parece convertirse en una consideración seria. En [13] se emplea una RNP de tipo convolucional con un total de 60 millones de parámetros. En función de clasificar 1.2 millones de imágenes de alta resolución en 1000 categorías diferentes, los autores debieron realizar aumento de los datos y aplicar otras metodologías reguladoras en función de evitar el sobre-entrenamiento. Más aún, existe consenso en cuanto a que incluso con bases de datos de entrenamiento como ImageNet [41], con 15 millones de imágenes de alta resolución clasificadas en 22 mil categorías, las RNP necesitarían poseer una gran cantidad de información a priori para evitar el sobre-entrenamiento y la falta de información.

Las limitaciones de las pequeñas bases de datos de imágenes han sido ampliamente identificadas en investigaciones como [13, 35, 12, 4]. La relación entre la cantidad de parámetros de las RNP, la cantidad de patrones de entrenamiento y la necesidad de emplear mayor cantidad de datos para el uso de modelos cada vez más complejos se explica en el problema de la dimensionalidad.

En [39] se plantea que en problemas de aprendizaje por computadora que involucran el aprendizaje de un conjunto finito de patrones de entrenamiento, la capacidad predictiva de los modelos se reduce en la medida que la cantidad de parámetros aumenta. Este descubrimiento es ampliamente conocido en la comunidad como la paradoja de Hughes. A pesar de las anotaciones posteriores a [39], la paradoja de Hughes ha sido observada con regularidad en los problemas de aprendizaje automático.

En espacios de elevadas dimensiones la cantidad de combinaciones de la señal de entrenamiento necesaria para describir el espacio de rasgos predictores es una explosión combinatoria. Esto puede ser corroborado al considerar variables binarias de dimensión d. La cantidad de posibles combinaciones en este caso se encuentra acotada por $O(2^d)$, lo cual es exponencial respecto a la dimensión.

Los algoritmos de aprendizaje automático necesitan ser guiados en base a suposiciones sobre el

tipo de funciones que deben aprender. Muchos algoritmos de aprendizaje automático se basan en la suposición de que dicha función es suave y que un patrón de entrenamiento describe consistentemente su localidad. La familia de algoritmos basados en los k-vecinos más cercanos es un caso extremo [3].

De acuerdo a [3], para que estos algoritmos sean capaces de distinguir r regiones, es necesario contar con al menos r patrones de entrenamiento. Informalmente, la cuestión ha sido formulada desde el campo del aprendizaje automático y radica en si es posible representar una función compleja que posea más regiones a distinguir que patrones de entrenamiento disponibles.

Adicionalmente, según [42] las bases de datos de una cantidad significativa de dominios de aplicación exhiben una propiedad conocida como *long-tail*. El problema de *long-tail* describe un fenómeno obervado en el que para bases de datos aparentemente masivas, pocas cosas son observadas con relativa frecuencia (por ejemplo: palabras) mientras la mayoría de las cosas son raras. Consecuentemente, la habilidad para generalizar a partir de una pequeña cantidad de patrones de entrenamiento continúa siendo de relevancia en la era de los datos masivos.

La complejidad temporal y espacial del cálculo de la función objetivo

La función objetivo $f(w, x)$ es un estimador de la precisión de la RNA denotado por $\hat{f}(w, x)$. A dicho estimador se le asocian un grupo de propiedades como el sesgo[5], la varianza y la consistencia [33]. En la práctica, suele usarse el Error Cuadrático Medio (MSE, *Mean Squared Error*), en la Ecuación(1.7), que proporciona un balance adecuado entre el sesgo y la varianza:

$$f(w, x) = \frac{\sum_{i=1}^{p} (\hat{y}^i - y^i)^2}{p} \tag{1.7}$$

donde y^i es la salida real y $\hat{y}^i$ es la salida esperada de la RNA para el patrón de entrenamiento x^i. La complejidad computacional de la función objetivo es elevada debido al cálculo de la salida real de la red y^i obtenida a partir de la propagación de los patrones de entrenamiento. Para una RNA con una neurona de tipo *sigma-unit* se tiene que la salida real está dada de acuerdo a la Ecuación(1.8), siendo $n = d$.

[5]Del término en inglés *bias*.

$$y^i = \delta(\sum_{j=1}^{n} w_j x_j^i + \theta) \tag{1.8}$$

En la Ecuación(1.8) w_j es el componente j del vector que representa los parámetros de la RNA y x_j^i es el componente j del patrón de entrenamiento x^i. Al considerar una RNA con más de una unidad de procesamiento del tipo propagación hacia adelante y totalmente conexa, el cálculo de la función objetivo se realiza de manera recursiva de acuerdo a la Ecuación(1.9).

$$y^i(c) = \begin{cases} x^i & \text{si } c = 1 \\ \delta(y^i(c-1)W(c)) & \text{en otro caso} \end{cases} \tag{1.9}$$

En la Ecuación(1.9) el vector $y^i(c)$ es la salida de la capa c para el patrón de entrenamiento x^i aumentado con un valor unitario:

$$y^i \leftarrow \{y_1^i, y_2^i, \cdots, y_d^i, 1\} \tag{1.10}$$

siendo d la dimensión original de y^i y $W(c)$ la matriz conformada por los parámetros de las neuronas de la capa c organizados en forma de columna de la siguiente forma:

$$W = \begin{bmatrix} w_1^1 & w_1^2 & \cdots & w_1^n \\ w_2^1 & w_2^2 & \cdots & w_2^n \\ \vdots & \vdots & \cdots & \vdots \\ w_d^1 & w_d^2 & \cdots & w_d^n \\ \theta^1 & \theta^2 & \cdots & \theta^n \end{bmatrix} \tag{1.11}$$

donde $w_{i,j}$ es el componente denotado por j perteneciente a la neurona i y n es la cantidad de neuronas de la capa c. Los valores de θ_i representan el valor del *bias* correspondiente a la neurona i.

Claramente, los valores de la matriz de parámetros W, n y d varían de capa en capa, pero se ha omitido ese detalle para simplificar la notación. De igual forma, estrictamente la función de propagación expresada en la Ecuación(1.9) varía en dependencia de cómo se organicen las

conexiones entre las capas de la RNA en consideración de la Sección 1.1.2.

Para RNP, donde la cantidad de parámetros es elevada y por tanto, la cantidad de patrones de entrenamiento también es elevada, resulta evidente la complejización del cálculo de la función objetivo de acuerdo a la Ecuación(1.9). En tal contexto, se hace imprescindible la implementación de técnicas de computación de alto rendimiento para evitar exceder los límites computacionales prácticos.

Cambios en la estructura de la superficie de optimización

Se ha mostrado que la superficie de error generada en el espacio de parámetro de las RNP es no convexa [43] y el número de puntos críticos se incrementa exponencialmente respecto a la dimensión del problema de optimización. De hecho, estas irregularidades aparentan comportarse como puntos de montura en lugar de mínimos locales verdaderos [44, 45, 9] lo que causa una disminución de la convergencia de los algoritmos de entrenamiento.

De acuerdo a [45] el mayor problema cuando se realiza optimización en espacios de elevadas dimensiones no es la presencia de mínimos locales, sino la proliferación de puntos de montura. En particular, los algoritmos de optimización basados en gradiente son especialmente susceptibles a la presencia de estos puntos de montura. La explicación para esto proviene del hecho que los puntos de montura son atractivos para estos algoritmos.

A pesar de la proliferación exponencial de irregularidades locales, algunos autores como [46] consideran que las superficies de optimización involucradas en el entrenamiento de RNP presentan una incoherencia entre su estructura local y su estructura global. Esto provoca el surgimiento de largas trayectorias de desplazamiento en el espacio de parámetros y convergencia prematura durante el entrenamiento.

En la Figura(1.5) se ilustra el principio de la dificultad de la incoherencia entre la estructura local y la estructura global de la superficie de error en el entrenamiento profundo. A la izquierda, se presenta la estructura local de la superficie de error indicando un mínimo en el punto $(320, 40)$. Sin embargo, al considerar la estructura global de dicha superficie en la figura a la derecha, el mínimo global se encuentra realmente en el punto $(700, 35)$ solo que a una distancia mayor en una trayectoria con menor pendiente. Nótese el cambio en la escala de la figura.

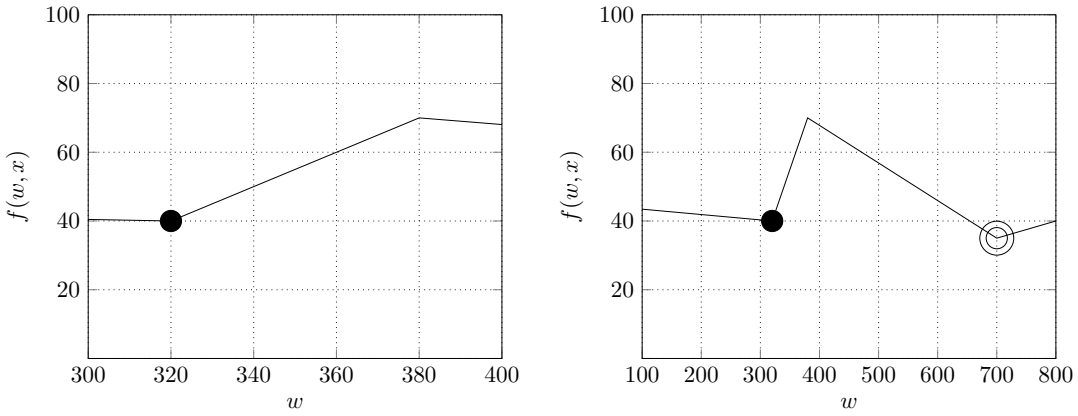


Figura 1.5: Incoherencia entre la estructura local y la estructura global de la superficie de error en el entrenamiento profundo.

1.3. Optimización meta-heurística aplicada al entrenamiento neuronal

La investigación en el área de la optimización tiene una larga historia. En este epígrafe se realiza un análisis de los algoritmos meta-heurísticos de optimización aplicados al entrenamiento de RNP. Para esto, se realizó un estudio de la literatura consistente en 35 trabajos de los últimos 15 años referidos al entrenamiento de RNA mediante algoritmos meta-heurísticos de optimización.

La optimización o de manera más general la búsqueda es un concepto clave para la Inteligencia Artificial pues es transversal a todas las disciplinas. Como se ha visto, los espacios de búsqueda de las aplicaciones prácticas y especialmente los espacios de búsqueda de las aplicaciones de las RNP son típicamente grandes, ver Figura(1.4), excluyendo la posibilidad de ser enumerados [47]. Siendo esto, los métodos de optimización tradicionales basados en el cálculo y en enumeración presentan serias dificultades al enfrentar este tipo de problemas.

Un algoritmo de optimización meta-heurístico permite explorar un espacio de búsqueda mediante prueba y error implicando usualmente algún tipo de aleatoriedad. El principal criterio que define el éxito de estos algoritmos de optimización se basa en equilibrar la exploración del espacio de búsqueda y la explotación de la información local. Para una introducción formal véase [48, 47].

Teniendo en cuenta los retos de optimización para los algoritmos de primer y segundo orden en el entrenamiento de RNP, los algoritmos meta-heurísticos de optimización representan una

alternativa. Los algoritmos meta-heurísticos de optimización han sido aplicados con éxito a problemas no convexos y con una elevada cantidad de mínimos locales. En la literatura se identifica un grupo de algoritmos meta-heurísticos de optimización que han sido aplicados al entrenamiento, lo que demuestra el interés de la comunidad científica por estos algoritmos:

- **Particle Swarm Optimization**(PSO). El algoritmo PSO fue introducido en [49] a partir de un grupo de estudios y observaciones de bandadas de pájaros, bancos de peces y otros seres vivos que muestran un comportamiento coordinado en su desplazamiento.
- **Firefly Algorithm**(FA). El algoritmo FA fue introducido en [50] inspirado en el apareamiento de las luciérnagas mediante destellos de luz. Cada luciérnaga representa una solución y son atraídas unas a otras en dependencia de la intensidad de la luz que emiten.
- **Cuckoo Search**(CS). El algoritmo CS fue introducido en [51] inspirado en el comportamiento del ave *cuckoo*. Cada ave representa una solución en el espacio de búsqueda y se mueve considerando un paseo aleatorio de *Lévy*. En cada generación, una proporción de estas aves es re-inicializada aleatoriamente.
- **Artificial Bee Colony Optimization**(ABC). El algoritmo ABC fue introducido en [52] inspirado en el comportamiento de las colmenas de abejas. Cada fuente de alimento representa una posible solución a la cual se le hace corresponder una abeja. Las abejas realizan búsqueda local y global teniendo en cuenta las fuentes de alimento descubiertas.

En la Tabla(1.1) se muestra un resumen de la revisión bibliográfica realizada relacionada con el entrenamiento de RNA mediante algoritmos meta-heurísticos. En la revisión bibliográfica se tuvo en cuenta los siguientes criterios para su análisis:

- Implementación (Impl.): Tecnología, lenguaje de programación o biblioteca empleado en la experimentación del trabajo.
- Cantidad de bases de datos (BD): Cantidad de bases de datos empleadas en el trabajo.
- Tamaño de la población (Pob.): Cantidad de individuos empleados por el algoritmo meta-heurístico en caso que este sea basado en población.

- Cantidad de patrones (Cant.): Cantidad de patrones de entrenamiento en la base de datos de mayor tamaño de las empleadas en el trabajo.
- Cantidad de parámetros (Params.): Cantidad de parámetros o pesos de la RNA de mayor tamaño de las empleadas en el trabajo.
- Empleo de regularización (Reg.): Especifica si se emplea alguna metodología regularizadora en el trabajo.
- Uso del algoritmo meta-heurístico (Uso): Especifica si el algoritmo meta-heurístico se emplea para el entrenamiento de la RNA o para el pre-entrenamiento. El pre-entrenamiento es un proceso de optimización global, por ejemplo mediante una meta-heurística, aplicada antes de un algoritmo de optimización local como SGD.
- Tarea a resolver (Tarea): Tipo de tarea o problema a resolver en el trabajo (clasificación, regresión, agrupamiento, etc.)
- Modelo de la RNA (Modelo): Tipo de modelo o arquitectura entrenado por el algoritmo meta-heurístico.
- Función de activación (FA): Función o funciones de activación empleadas en la RNA.
- Cantidad de capas (Capas): Cantidad de capas que posee el modelo de RNA.
- Paralelización (Paral.): Especifica si el algoritmo meta-heurístico es paralelizado de alguna forma.
- Comparación: Indica con qué otros algoritmos meta-heurísticos o métodos se realiza una comparación de los resultados obtenidos en el entrenamiento.
- Función objetivo: Indica la función objetivo empleada en el entrenamiento.
- Pruebas estadísticas (Pruebas estad.): Especifica las pruebas estadísticas realizadas para encontrar diferencias significativas en la comparación del algoritmo meta-heurístico propuesto respecto a otros métodos.
- Configuración de hiper-parámetros (Hiper-parámetros): Metodología y algoritmos para la optimización de hiper-parámetros relacionados con el algoritmo meta-heurístico.

- Métodología para la estimación del error de generalización (Metod.): Especifica cómo se calculó el error de generalización de la RNA.

TABLA 1.1: Análisis de los principales trabajos relacionados con el entrenamiento neuronal meta-heurístico.

Impl.	BD	Pob.	Cant.	Params.	Reg.	Uso[1]	Tarea[2]	Modelo[3]	FA[4]	Capas	Paral.	Comparación	Función objetivo[5]	Pruebas estad.[6]	Hiper-parámetros	Metod.	Comentarios	Ref.
Algoritmo PSO																		
MATLAB	1	40	1105	593	No	E	R	MLP	S	3	No	LM	MSE	No	Modelo (análisis de sensibilidad), PSO (arbitrario)	ND	Predicción del resultado de disputas legales relacionadas con la construcción	[53]
MATLAB	1	300	44	197	No	E	R	MLP	S	3	No	Métodos del dominio	MSE	No	Análisis de sensibilidad y búsqueda aleatoria	H-Method (80 % - 20 %)	Predicción de la vibración y distancia de la lluvia de rocas producida por explosiones en la construcción y minería	[54]
ND	1	30	9568	61	No	E	R	MLP	S	3	No	No	MSE	No	Arbitrario	H-Method (70 % - 30 %)	Predicción de la cantidad de MW producidos por una planta eléctrica	[55]
ND	1	ND	190	77	No	PE	R	MLP	S	3	No	Métodos del dominio	MSE	No	Arbitrario, trabajos previos	H-Method (70 % - 30 %)	Predicción de la precipitación de asfaltenos	[56]
ND	1	20	ND	801	No	E	R	MLP	S	3	No	No	MSE, R, E	No	Búsqueda rejilla	ND	Predicción de los períodos de sequía y lluvia basado en los antecedentes climáticos	[57]
ND	1	10	43	32	No	E	R	MLP	S	4	No	SGD	MSE	No	Búsqueda rejilla empleando el conjunto de validación	H-Method (80 % - 20 %)	Predicción del tiempo de duración y desgaste de performación	[58]
ND	1	20	2800	50	No	E	R	MLP	S	3	No	SGD	PINAW, PIACE	IC	Abritrario	H-Method (60 % - 40 %)	Predicción de la producción eléctrica de un parque eólico	[59]
MATLAB	1	3	32	85	No	E	R	MLP	TANH	3	No	LM	OE	No	Búsqueda rejilla (modelo), ND (PSO)	ND	Predicción de cuatro factores de relevancia en la soldadura	[60]
ND	1	350	699	43	No	E	R	MLP	S	3	No	LM	R, RMSE, VAF	No	Búsqueda aleatoria, análisis de sensibilidad, trabajos previos	ND	Predicción del riesgo de inestabilidad de pendientes durante temblores de tierras	[61]

Continuación de la página anterior.

Impl.	BD	Pob.	Cant.	Params.	Reg.	Uso[1]	Tarea[2]	Modelo[3]	FA[4]	Capas	Paral.	Comparación	Función objetivo[5]	Pruebas estad.[6]	Hiper-parámetros	Metod.	Comentarios	Ref.
ND	0	25	ND	17	No	E	R	MLP	S	3	No	SGD	MSE	No	Búsqueda aleatoria	ND	Ajuste de una RNA a una curva cuadrática arbitraria	[24]
Map-Reduce / Hadoop	5	10	15000	621	No	PE	C	MLP	S	3	Dist.	SGD	SE	Precision and recall	Búsqueda aleatoria	H-Method	Clasificación de imágenes de escenas. Clúster de cinco nodos (DUAL CORE 4GB RAM)	[62]
ND	3	30	300	ND	No	E	R, C	MLP	S	3	No	GA, TS, SA, EP	NMSE	T-Student	Trabajos previos	H-Method (50 % - 25 %- 25 %)	Predicción de varios problemas estándar relacionados con la bio-medicina	[63]
MATLAB	1	ND	98	ND	No	PE	C	MLP	S	3	No	No	OE	No	Arbitrario	ND	Clasificación de plantas IRIS	[64]
Algoritmo FA																		
MATLAB	0	50	6	2	No	E	R	MLP	S	3	No	No	SSE	No	Arbitrario	No aplica	Ajuste de una función arbitraria. Las neuronas no poseen un parámetro de sesgo (bias)	[65]
JAVA	3	50	9	22	No	E	C	MLP	S, SI	3	No	GA, ABC	MSE	No	Trabajos previos	No aplica	Ajuste de la RNA para resolver los problemas XOR, 4-Bit Party, 3-Bit Encoder	[66]
MATLAB	6	50	8000	ND	No	E	C	MLP	ND	ND	No	KNN	OE	No	Arbitrario	H-Method	Predicción sobre varias bases de datos relacionadas con problema de time-series	[67]
ND	3	ND	5184	105	No	PE	R	MLP	S	3	No	SGD	MAPE, MAE, MSE	No	Búsqueda aleatoria	H-Method	Predicción de la velocidad del viento	[68]
MATLAB	1	ND	1224	ND	No	PE	C	MLP	ND	ND	No	No	OE	No	Arbitrario	ND	Clasificación de imágenes de números. Los parámetros de varias épocas de SGD conforman la población para FA	[69]

Continuación de la página anterior.

Impl.	BD	Pob.	Cant.	Params.	Reg.	Uso[1]	Tarea[2]	Modelo[3]	FA[4]	Capas	Paral.	Comparación	Función objetivo[5]	Pruebas estad.[6]	Hiper-parámetros	Metod.	Comentarios	Ref.
MATLAB	11	ND	800	ND	No	E	C	SPH	SPH	ND	No	SGD, PSO, GA, Hybrid PSO-GA	RMSE	ANOVA, Friedman, Tukey, Dunnett, Post Hoc Test	Arbitrario	K-fold cross-validation	Clasificación de varios problemas estándar. Híbrido entre FA, SA y SGD. La FO involucra un paso de optimización con SGD y SA ajusta el momentum de SGD.	[70]
MATLAB	1	50	120	344	No	E	C	RBF	RBF	3	No	SGD	MSE	ROC	Arbitrario	H-Method (75 %-25 %)	Predicción del desorden asociado con las imágenes radiográficas relacionadas con el dolor del hombro	[71]
Algoritmo CS																		
MATLAB	3	ND	12	ND	No	PE	C	MLP	S	3	No	SGD, ABC-LM, ABC-SGD	MSE	No	Arbitrario	No aplica	Ajuste de la RNA para resolver los problemas 4-bit OR, 2-bit and 3-bit XOR. Refinación mediante SGD	[72]
MATLAB	3	ND	12	ND	No	PE	C	MLP	S	3	No	SGD, ABC-LM, ABC-SGD	MSE	No	Arbitrario	No aplica	Ajuste de la RNA para resolver los problemas 4-bit OR, 2-bit and 3-bit XOR. Refinación mediante LM	[72]
MATLAB	1	25	32	76	No	E	R	MLP	S	3	No	PSO, ABC	MSE	No	Búsqueda de rejilla (modelo), trabajos previos para CS	H-method	Predicción de emisiones CO2 de los países perteneciente a OPEC. Variación de CS que incorpora un factor de velocidad.	[73]
ND	8	ND	1000	300	No	PE	C	MLP	S, soft-max	5	No	SGD, LM	OE	No	Arbitrario, trabajos previos	K-fold cross-validation	Predicción de varios problemas estándar. Variación de CS con cambios en el tamaño del paso aleatorio y el factor de remplazo.	[74]
MATLAB	1	5	320	13	No	E	C	MLP	S	3	No	SGD, regresión lineal	MSE	POD, POFD, FAR, CSI, etc.	Arbitrario	H-method	Predicción de la ocurrencia de descargas eléctricas	[75]
ND	6	40	300	ND	No	E	C	NI	ND	ND	No	DE	OE	No	Arbitrario	H-Method (80 %-20 %)	Clasificación de varios problemas estándares	[76]

Continuación de la página anterior.

Impl.	BD	Pob.	Cant.	Params.	Reg.	Uso[1]	Tarea[2]	Modelo[3]	FA[4]	Capas	Paral.	Comparación	Función objetivo[5]	Pruebas estad.[6]	Hiper-parámetros	Metod.	Comentarios	Ref.
ND	2	10	699	89	No	E	C	MLP	ND	3	No	CS	SSE	No	Arbitrario	ND	Clasificación de varios problemas estándares. Se trata de una versión modificada de CS	[17]
Algoritmo ABC																		
ND	6	30	6200	3600	Si	E	C	MLP	S	ND	No	SGD, ABC	OE	No	Trabajos previos, arbitrario	H-Method	Clasificación empleando problemas estándar.	[77]
MATLAB	1	20	1653	381	No	E	C	MLP	S	3	No	SGD, PSO, GA, SA, ABC	MSE	Matriz de confusión	Búsqueda aleatoria	H-Method, K-fold cross-validation	Clasificación de frutas. Variación de ABC (ABC-Caótico)	[78]
MATLAB	1	20	66	211	No	E	C	MLP	S	3	No	SGD, GA, SA, ABC, Elite-GA	MSE	Matriz de confusión	Búsqueda aleatoria	K-fold cross-validation	Clasificación de resonancias magnéticas del cerebro. Versión mejorada de ABC.	[79]
MATLAB	1	ND	1000	9	No	E	R	MLP	S	3	No	SGD, ABC	MSE	No	Arbitrario	H-Method (70 %-30 %)	Predicción de la intensidad de los temblores de tierra Híbrido ABC-ACO.	[80]
ND	1	100	232	1594	No	E	C	MLP	TANH	3	ND	SGD	OE	No	Arbitrario	H-Method (80 %-20 %)	Clasificación de defectos en la madera	[81]
Python	4	10	214	4	No	E	C	MLP	S	3	No	GA	SSE	No	Arbitrario	ND	Clasificación de varias bases de datos estándar del repositorio UCI	[82]
Matlab	1	50	ND	16	No	E	R	MLP	S	3	No	SGD	MSE, NMSE	No	Arbitrario	H-Method (70 %-30 %)	Predicción de eventos sísmicos	[83]

[1] Uso del algoritmo meta-heurístico, posibles valores: Entrenamiento (E), Pre-entrenamiento (PE)

[2] Tarea relacionada con la RNA, posibles valores: Regresión (R), Clasificación (C)

[3] Modelo, posibles valores: Perceptrón Multi-capa (MLP), Radial Basis Function (RBF), Neurona Izhikevich (NI), SIGMA-PI-HONN (SPH)

[4] Función de activación de la neurona, posibles valores: Sigmoidal (S), Tangente Hiperbólica (TANH), Seno (SI), SIGMA-PI-HONN (SPH)

[5] Función objetivo empleada en el entrenamiento, posibles valores: Normalized Mean Squared Error (NMSE), Overall Error (OE), Sum of Squared Error (SSE), Correlation coefficient (R), Nash-Sutcliffe coefficient (E), Predicted Interval Normalized Average (PINAW), Predicted Interval Average Centering Error (PIACE), Value Account For (VAF), Mean Absolute Error (MAE), Mean Absolute Percent Error (MAPE)

[6] Pruebas estadísticas para comparar los algoritmos de entrenamiento, posibles valores: Probability Of Detection (POD), Probability Of False Detection (POFD), False Alarm Rate (FAR), Critical Success Index (CSI), Intervalos de confianza (IC)

En la Figura(1.6) se muestra el incremento del empleo de los algoritmos meta-heurísticos de optimización en el entrenamiento de RNA. Puede apreciarse que el uso de estos algoritmos se ha incrementado en los últimos años como una alternativa viable al entrenamiento de RNA. La causa para el incremento en la búsqueda de algoritmos de optimización alternativos a los algoritmos de primer y segundo orden se fundamenta en los cambios cuantitativos y cualitativos del entrenamiento de RNP. Sin embargo, es poco común en estos trabajos presentar un análisis teórico y práctico que consideren estos cambios de una manera sistemática.

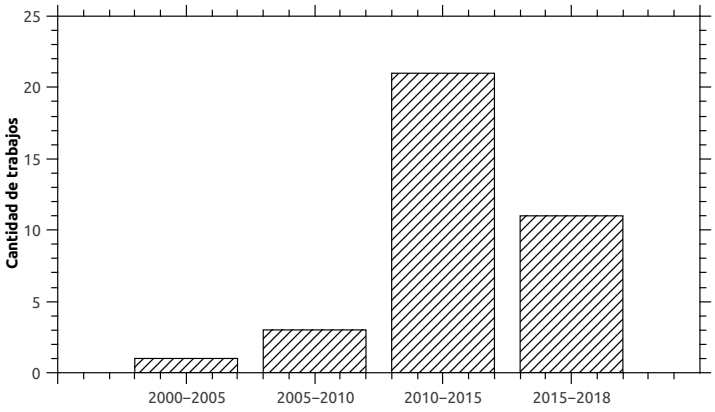


Figura 1.6: Cantidad de trabajos relacionados con el entrenamiento de RNA mediante algoritmos meta-heurísticos.

Un análisis de los trabajos relacionados con el empleo de algoritmos meta-heurísticos en el entrenamiento de RNA, revela que la cantidad de patrones en las bases de datos de aprendizaje es relativamente pequeña. Es significativa la diferencia respecto a la cantidad de patrones de entrenamiento en los problemas de aprendizaje trabajados mediante las RNP entrenadas con algoritmos basados en gradiente. De manera similar, es significativa la diferencia respecto a la cantidad de neuronas y parámetros.

Consecuentemente, solamente el 2,8 % de la literatura consultada se refiere a la paralelización de los algoritmos meta-heurísticos aplicados al entrenamiento de RNA. Nótese que al menos el 48,5 % de estos trabajos fue implementado en alguna versión de MATLAB de manera secuencial.

La aplicación de algoritmos meta-heurísticos al entrenamiento de RNP se estudia en menor medida en la literatura. Estos trabajos se refieren a RNA de tipo perceptrón multi-capa con función de activación de tipo *sigmoid* en su mayoría sin considerar modelos profundos que

incluyan capas de tipo convolucionales, parcialmente conexas, de agrupamiento, etc. De hecho, la falta de investigación en el área de la optimización meta-heurística aplicada al entrenamiento de RNP ha sido constatado recientemente en trabajos como [84].

Desde otra perspectiva, los trabajos revisados refieren problemas de clasificación o regresión, ignorando otras tareas de importancia como la clasificación con valores ausentes [85], la traducción asistida por computadora [86, 87], etc. Igualmente, el estudio y adaptación de los métodos de regularización y su efecto en el entrenamiento de RNA mediante algoritmos meta-heurísticos ha sido abordado en poca medida.

Se observa la presencia de anomalías experimentales en una parte importante de los trabajos estudiados. En la literatura consultada se hace uso de un particionado fijo de la base de datos en subconjuntos disjuntos de entrenamiento y prueba. Esto sucede incluso para bases de datos con una cantidad pequeña de patrones de entrenamiento para las cuales la recomendación es el empleo del algoritmo *k-fold cross-validation* [42].

Prolifera la configuración de hiper-parámetros de los algoritmos de entrenamiento de manera manual, en base a la experiencia del investigador o en base a trabajos previos. Esto resulta crítico pues es conocida la sensibilidad de los algoritmos de aprendizaje automático y optimización a la configuración de estos hiper-parámetros [88, 89]. Incluso, en los trabajos en los que se realiza algún tipo de análisis basado en prueba y error [61, 78], análisis de sensibilidad [54, 61] o una simple estrategia de búsqueda de rejilla [60, 58] no se realiza una estimación del sesgo inducido producto de la optimización de hiper-parámetros. Referirse a [90, 91] para una sugerencia metodológica en la optimización de hiper-parámetros.

Finalmente, se constata un vacío en la metodología de comparación entre los distintos algoritmos de optimización aplicados al entrenamiento de RNA. Los estudios comparativos son escasos, en una importante cantidad de los trabajos consultados no se llega a comparar los algoritmos meta-heurísticos propuestos con trabajos previos o algoritmos similares. En los trabajos que se realiza una comparación con otros métodos, muchas veces no se realizan análisis estadísticos para encontrar desviaciones significativas. La variedad de métricas de precisión contribuye de manera especial a la complejización de esta tarea.

1.4. Conclusiones parciales

A partir del análisis de los principales conceptos y literatura realizado en el presente capítulo es posible llegar a las siguientes conclusiones parciales sobre el entrenamiento de RNP:

1. El entrenamiento de RNA es un problema de optimización NP-duro que presenta un grupo de cambios cuantitativos y cualitativos al considerar las RNP. Estos cambios producen dificultades y retos teórico-prácticos afectando la precisión de los modelos de RNP construidos.

2. Los algoritmos de primer y segundo orden basados en derivadas presentan limitaciones teóricas y prácticas de relevancia al considerar el entrenamiento de RNP. Entre estas limitaciones se encuentra la convergencia en mínimos locales y su paralelización. La investigación de algoritmos meta-heurísticos de optimización capaces de superar estas limitaciones es de interés en el área demostrado en la aparición de trabajos relacionados.

3. Los algoritmos meta-heurísticos de optimización aplicados al entrenamiento de RNA se han estudiado tradicionalmente en los últimos 20 años de forma ascendente. No obstante, en la literatura se constatan vacíos y limitaciones relacionadas con su empleo. Fundamentalmente, al tratar RNP los estudios son más limitados, dejando de considerar los cambios evidenciados en el entrenamiento de estos modelos.

Capítulo 2

Algoritmos meta-heurísticos aplicados al entrenamiento neuronal

---

Teniendo en cuenta los elementos abordados en la fundamentación teórica, la investigación se centra en la adopción de un grupo de algoritmos meta-heurísticos de optimización para su aplicación al entrenamiento de RNP. En el presente capítulo se describe el empleo de tres algoritmos meta-heurísticos para el entrenamiento de RNP (PSO, FA, CS) en paralelo. Posteriormente se introduce un nuevo algoritmo basado en métodos de continuación para la simplificación de la función objetivo empleada en el entrenamiento de RNP mediante algoritmos meta-heurísticos de optimización. Se estudian variantes del algoritmo para la simplificación de la función objetivo considerando la probabilidad de convergencia local, la complejidad computacional del entrenamiento y su implementación en paralelo. Finalmente, se presentan las conclusiones parciales del capítulo.

2.1. Paralelización de algoritmos meta-heurísticos

En esta sección se abordan algunos aspectos generales sobre la paralelización de los algoritmos meta-heurísticos, fundamentalmente en GPU. De acuerdo a [92] los algoritmos meta-heurísticos se conforman de cuatro fases: inicialización de la población, evaluación de la función objetivo, comunicación y actualización de la población. Debido a la diversidad de algoritmos meta-heurísticos, estas cuatro fases no siempre se ejecutan en el mismo orden y no siempre pueden ser identificadas por separado. Estas cuatro fases son ejecutadas de manera iterativa hasta que cierto criterio de convergencia se cumple.

El criterio de convergencia de un algoritmo meta-heurístico determina en qué punto el algoritmo debe terminar las iteraciones. En este trabajo se consideran tres criterios de convergencia adoptados para su estudio:

- Cantidad fija de iteraciones. El algoritmo realizará un número fijo η de iteraciones antes de terminar.
- Umbral de mejora mínima. El algoritmo continuará realizando iteraciones mientras que la diferencia en el valor de la función objetivo en las últimas η iteraciones sobrepase cierto umbral ψ.
- Diferencia mínima global. El algoritmo continuará realizando iteraciones mientras la varianza en el valor de la función objetivo de todos los individuos de la población sobrepase cierto umbral ψ.

Teniendo esto en cuenta se proponen dos modelos de paralelización para los algoritmos meta-heurísticos considerados en el presente trabajo: modelo simplificado[1] y modelo multi-fase. En la Figura(2.1) se muestra el modelo secuencial, el modelo simplificado y el modelo multi-fase de implementación de algoritmos meta-heurísticos en paralelo de izquierda a derecha respectivamente.

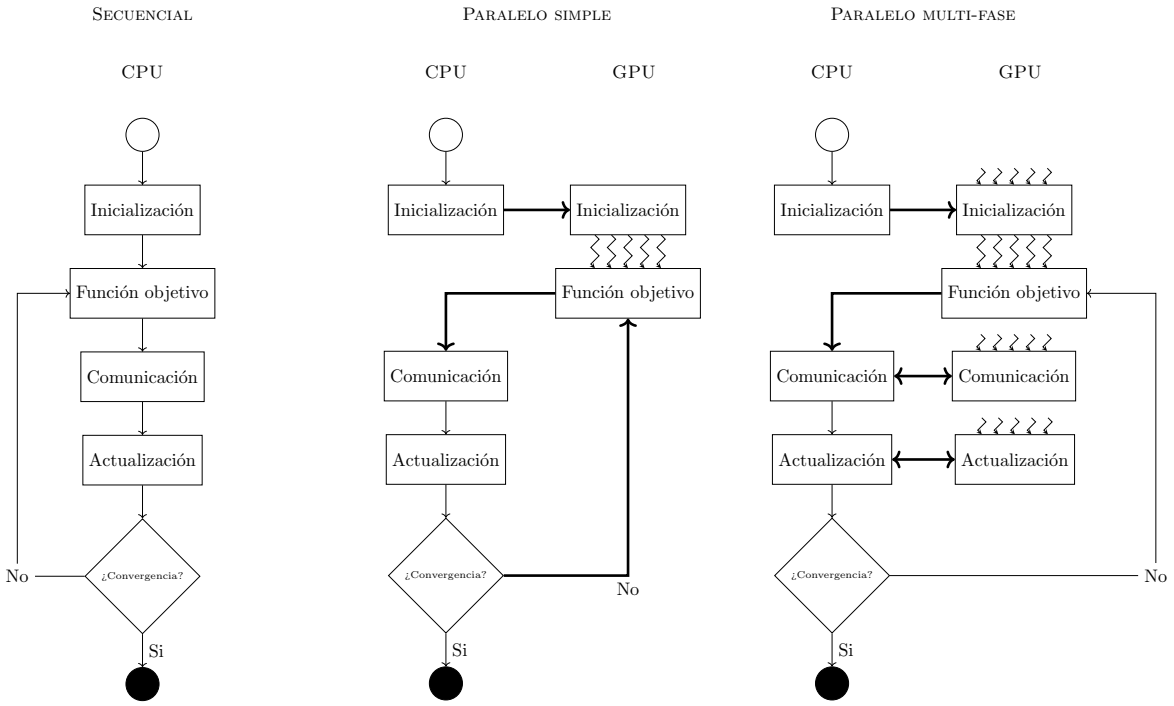


Figura 2.1: Modelo secuencial, simple y multi-fase para la implementación en paralelo de los algoritmos meta-heurísticos.

[1]El término usado en el artículo original es *naive-parallel-model*.

El modelo simplificado puede emplearse cuando la función objetivo es paralelizable y es un punto de partida para disminuir el tiempo de ejecución del algoritmo meta-heurístico cuando dicha función objetivo posee una elevada complejidad computacional. El atractivo de este modelo de paralelización proviene del hecho de que con paralelizar solamente el cálculo de la función objetivo es posible realizar el entrenamiento en menor tiempo.

En la Figura(2.1), las líneas más oscuras representan transferencia de datos entre los espacios de memoria de la CPU y la GPU. Nótese que en la fase de inicialización los patrones de entrenamiento son cargados primero en el espacio de memoria del CPU, posteriormente se crea y genera la población inicial en la CPU y todo esto se copia al espacio de memoria de la GPU. De manera similar, una vez que se ha realizado el cálculo de la función objetivo para cada individuo de la población en la GPU, los vectores de parámetros de las RNA se copian hacia el espacio de memoria de la CPU para que las siguientes fases de comunicación y actualización se ejecuten.

El modelo multi-fase propone considerar la paralelización en GPU del resto de las fases del algoritmo meta-heurístico. El objetivo es realizar todo el cálculo posible en GPU para evitar la transferencia de datos entre los espacios de memoria.

De esta forma en la fase de inicialización, la generación de la población se ejecuta en la GPU. Una vez calculado el valor de la función objetivo solamente se copia hacia el espacio de memoria del CPU dicho valor en lugar del vector de parámetros de la RNA. Esto es posible siempre que la computación correspondiente a las fases de comunicación y actualización se realice en la GPU. Este modelo es una evolución del modelo simple, pero involucra enfrentarse al reto de paralelizar todas las fases del algoritmo meta-heurístico.

2.1.1. Función objetivo

El cálculo de la función objetivo se encuentra entre los procedimientos más costos de efectuar en el entrenamiento neuronal. Para realizar la evaluación de la función objetivo es necesario propagar cada patrón de entrenamiento en virtud de la Ecuación(1.9) y luego estimar el error cometido, por ejemplo, mediante el MSE en la Ecuación(1.7).

Para la paralelización del cálculo de la función objetivo se estudiaron varios esquemas de particionado. En [93] se proponen tres esquemas de particionado para la paralelización del cálculo de

la función objetivo en una RNA, el particionado completo, particionado vertical y particionado horizontal. En la Figura(2.2) se muestra a la izquierda la paralelización vertical y a la derecha la paralelización horizontal. Los círculos representan una neurona y las flechas representan las conexiones entre estas.

El particionado completo, propone asignar a cada neurona un hilo de ejecución, sin embargo, este enfoque es ineficiente debido a que en cada momento solamente dos capas de la RNA se encuentran activas por lo que se desperdiciaría una parte significativa de estos hilos. El particionado vertical propone asignar a cada hilo de ejecución, representado por p_i en la Figura(2.2), una neurona de cada capa de la RNA. De esta forma cada hilo posee trabajo que realizar en cada etapa de la propagación.

El particionado horizontal propone asignar un hilo de ejecución a cada capa de la RNA. Debido a que la propagación de cada patrón de entrenamiento es paralelizable únicamente dentro de una misma capa no se recomienda este enfoque. Adicionalmente, el particionado horizontal se encuentra limitado por la cantidad de capas de la RNA.

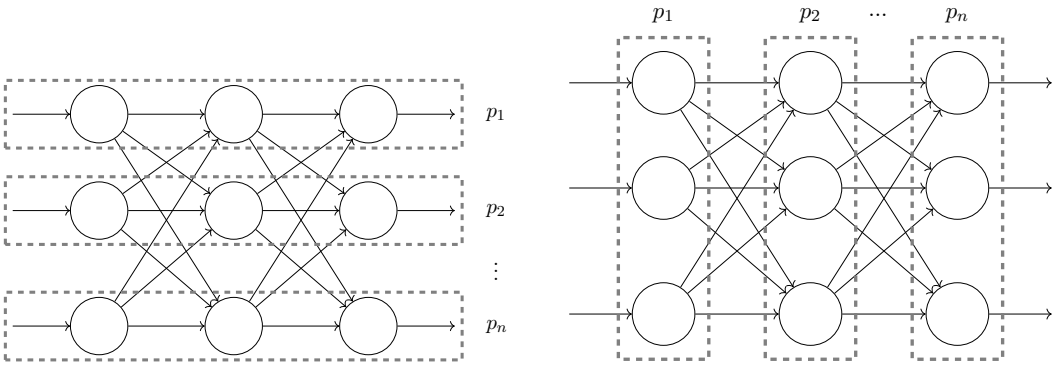


Figura 2.2: Particionado vertical a la izquierda y particionado horizontal a la derecha para la paralelización de la propagación hacia adelante de una RNA.

Una cuarta fuente de paralelización proviene de realizar la propagación de cada patrón de entrenamiento de manera independiente. En [11] se proponen dos algoritmos distribuidos para el entrenamiento mediante variaciones del algoritmo SGD. De manera general los algoritmos consisten en propagar los patrones de entrenamiento de manera paralela en distintos nodos de ejecución que poseen una réplica de los parámetros de la RNA. Una vez propagados, los nuevos parámetros calculados de manera independiente en virtud del paso de retro-propagación de SGD se sincronizan en un servidor de parámetros global.

En este trabajo se emplea una versión de la propagación paralela de cada patrón de entrenamiento para la ejecución en GPU. Para esto se considera la transformación del paso de propagación de los patrones de una RNA en una multiplicación de matrices como se ha sugerido en trabajos como [14]. Siendo esto, se conforma la matriz con los patrones de entrenamiento de la siguiente forma:

$$Y(1) = \begin{bmatrix} y_1^1 & y_2^1 & \cdots & y_d^1 & 1 \\ y_2^2 & y_2^2 & \cdots & y_d^2 & 1 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ y_1^n & y_2^n & \cdots & y_d^n & 1 \end{bmatrix} \tag{2.1}$$

donde $Y(c)$ representa la entrada de la capa c. Para la primera capa esta matriz es conformada con los patrones de entrenamiento. Considerar la Ecuación(1.10) como referencia a los valores y_j^i.

Reformulando la Ecuación(1.9), la propagación de los patrones de entrenamiento es:

$$Y(c) = \begin{cases} Y(1) & \text{si } c = 1 \\ \delta(Y(c-1)W(c)) & \text{en otro caso} \end{cases} \tag{2.2}$$

donde $\delta(.)$ representa la evaluación de la función de activación en cada componente de la matriz resultante. Considerando la transformación en producto de matrices del entrenamiento neuronal, la paralelización se delega en las rutinas paralelas optimizadas GEMM[2] ampliamente tratadas en la literatura [94, 95, 96].

2.2. Meta-heurísticas de optimización aplicadas al entrenamiento neuronal profundo

En optimización, meta-heurística significa encontrar o descubrir mediante prueba y error [97]. Soluciones de calidad a un problema de optimización pueden encontrarse en una cantidad de tiempo razonable pero sin garantías de encontrar soluciones óptimas. La mayoría de los algo-

[2] Acrónimo de *General Matrix Multiplication*.

ritmos meta-heurísticos de optimización poseen una combinación de dos componentes: intensificación y diversificación. La solución óptima a encontrar depende de un adecuado balance entre estos dos elementos.

Entre los algoritmos meta-heurísticos de optimización con mayor grado de tratamiento y maduración en la literatura se identifican el PSO, FA y CS. En este epígrafe se propone su empleo con el objetivo de obtener una precisión similar al algoritmo SGD en el entrenamiento de RNP.

2.2.1. Algoritmo PSO

El algoritmo PSO fue introducido en [49] a partir de un grupo de estudios y observaciones de bandadas de pájaros, bancos de peces y otros seres vivos que muestran un comportamiento coordinado en su desplazamiento. En algunas especies de animales, el movimiento coordinado expone algún tipo de experiencia o inteligencia de grupo en la consecución de determinado objetivo. A partir de esta observación, el algoritmo PSO es idealizado de acuerdo a los siguientes principios:

1. Cada individuo en la población es conceptualizado como una partícula que representa una solución candidata en el espacio de búsqueda.
2. Cada partícula posee una velocidad y dirección de movimiento en el espacio de búsqueda que es actualizada en cada generación.
3. La velocidad y dirección de la partícula se calcula teniendo en cuenta la mejor partícula en la población, la mejor posición visitada por la partícula y el movimiento inercial.

La velocidad de una partícula se calcula según la Ecuación(2.3) donde ω^1 y ω^2 son dos vectores aleatorios de longitud n generados a partir de una distribución uniforme. El factor w^* es la mejor partícula de la población de acuerdo al valor de la función objetivo y w^{i*} es la mejor posición visitada por la partícula i.

$$v^i \leftarrow \alpha v^i + \beta \omega^1 w^* + \gamma \omega^2 w^{i*} \tag{2.3}$$

.

La Ecuación(2.3) define tres hiper-parámetros del algoritmo: α se conoce en la literatura como el factor de inercia, β mide la contribución del mejor individuo en la dirección de la partícula y γ mide la contribución de la mejor posición visitada en la dirección de la partícula. Una vez calculada la velocidad de la partícula, su movimiento viene dado por la Ecuación(2.4)

$$w^i \leftarrow w^i + v^i \tag{2.4}$$

El Algoritmo(1) describe en pseudo-código el algoritmo PSO para minimizar el valor de una función objetivo. En el paso 1 la población de partículas es inicializada aleatoriamente en el espacio de búsqueda. Esto significa generar una cantidad $n \times p$ de números aleatorios, siendo n la dimensión del vector de parámetros de cada individuo y p la cantidad de individuos. En el paso 2 se comprueba que el algoritmo no se encuentre en un estado de convergencia. Luego, en cada iteración se re-calcula la velocidad de cada individuo, su nueva posición y se actualiza el mejor individuo de la población y la mejor posición visitada por cada individuo correspondiente a los pasos 5, 6 y 9 respectivamente. Finalmente se devuelve el individuo con menor valor de la función objetivo en el paso 12.

Algoritmo 1 Algoritmo PSO para minimizar una función objetivo.

1: Inicializar aleatoriamente la población w^i siendo $i = 1, ..., p$
2: **while** Convergencia == False **do**
3: **for** j = 1 : p **do**
4: Actualizar w^{j*} y w^*
5: **end for**
6: **for** j = 1 : p **do**
7: Calcular velocidad de la partícula j de acuerdo a la Ecuación(2.3)
8: Calcular nueva posición de la partícula j de acuerdo a la Ecuación(2.4)
9: **end for**
10: **end while**
11: **return** Partícula con el menor valor de la función objetivo w^*

A partir de los pasos 7 y 8 del Algoritmo(1) puede verse que el cálculo de la velocidad de cada partícula y su nueva posición depende de su mejor posición visitada y el mejor individuo global. Esto permite realizar estas dos operaciones de manera paralela para cada elemento del vector de parámetros sin afectar la convergencia del algoritmo.

Al considerar la paralelización del algoritmo PSO, en la fase de inicialización el proceso maestro

reserva de manera secuencial la memoria necesaria para los parámetros de cada individuo de la población y los patrones de entrenamiento. Posteriormente se cargan los patrones de entrenamiento en la memoria del CPU y seguidamente hacia el espacio de memoria de la GPU. En la propia fase de inicialización se lleva a cabo la generación de la población en la GPU. Cada hilo de ejecución paralelo se encarga de generar cierta cantidad de números aleatorios correspondientes a los componentes w_j^i del vector de parámetros de los individuos siendo $i < p$ y $j < n$.

Una vez terminada la generación de la población, el flujo de programa vuelve al proceso maestro en la fase de comunicación. De manera secuencial, se comprueba que el algoritmo no se encuentre en un estado de convergencia. Si el algoritmo no se encuentra en estado de convergencia, se calcula el valor de la función objetivo para cada individuo y se actualiza el mejor local y global en la GPU. En esta fase los vectores de parámetros de los mejores individuos locales y el mejor individuo global se mantienen en el espacio de memoria de la GPU.

Posteriormente se realiza el movimiento de cada individuo w^k en la fase de actualización. Los individuos de la población actualizan su posición de manera paralela siendo $0 \leq k < p$, considerando también que cada componente w_i^k es actualizado de manera paralela en la GPU. Como puede verse, la paralelización del algoritmo PSO se realizó teniendo en cuenta el modelo de paralelización multi-fase con el objetivo de disminuir la transferencia de datos entre la CPU y la GPU.

2.2.2. Algoritmo FA

El algoritmo FA fue introducido en [50] inspirado en el apareamiento de las luciérnagas mediante destellos de luz. En su entorno natural, el parpadeo de la luz de las luciérnagas es usado para atraer a otros de su misma especie. A partir de este comportamiento natural, el algoritmo FA es idealizado de acuerdo a los siguientes principios:

1. Todas las luciérnagas son asexuadas de forma tal que cualquiera en la población es atraída por otras. Los conceptos de individuo o luciérnaga representan una solución candidata en el problema de optimización.

2. La atracción entre dos luciérnagas es una proporción entre su brillo y la distancia entre

ellas. El brillo de la luciérnaga es determinado por el valor de la función objetivo de la solución que representa.

3. Si no existe diferencias entre el brillo de las luciérnagas, se moverán aleatoriamente.

La distancia entre dos luciérnagas usualmente se calcula mediante la distancia euclidiana, Ecuación(2.5). De ahí que, el movimiento de una luciérnaga w^i hacia una luciérnaga w^j venga dado por la Ecuación(2.6).

$$r(w^i, w^j) = \sqrt{\sum_{k=1}^{n} (w_k^i - w_k^j)^2} \tag{2.5}$$

$$w^i \leftarrow w^i + \beta e^{-\gamma r(w^i, w^j)^2}(w^j - w^i) + \eta\omega \tag{2.6}$$

En la Ecuación(2.6) se definen tres hyper-parámetros: β el brillo inicial cuando la distancia entre dos luciérnagas es cero, γ el coeficiente de atenuación de la luz y η la influencia de la aleatoriedad. En la ecuación, ω es un vector de longitud n generado aleatoriamente a partir de una distribución normal.

El Algoritmo(2) describe en pseudo-código el algoritmo FA para minimizar el valor de una función objetivo. En el paso 1 la población de luciérnagas es inicializada aleatoriamente en el espacio de búsqueda. En el paso 2 se comprueba que el algoritmo no se encuentra en un estado de convergencia.

En caso de que el algoritmo FA no se encuentre en un estado de convergencia, en el paso 3 la población es ordenada teniendo en cuenta el valor de la función objetivo para cambiar el orden en el que las luciérnagas son comparadas en los pasos 4 y 5. En cada iteración posterior cada luciérnaga se mueve en cierta medida hacia cada una de las luciérnagas que posean mayor atracción, pasos 4, 5, 6 y 7. Finalmente en el paso 12 se devuelve la luciérnaga con el menor valor de la función objetivo una vez que la cantidad máxima de iteraciones fue alcanzada.

A partir del paso 6 del Algoritmo(2) puede verse que una vez que una luciérnaga realiza un movimiento hacia otra el valor de su función objetivo es re-calculado. Este detalle posee una gran influencia en la convergencia del algoritmo, sin embargo, dificulta su paralelización. El

Algoritmo 2 Algoritmo FA para minimizar una función objetivo.

```
1: Inicializar aleatoriamente la población w^i siendo i = 1, ..., p
2: while Convergencia == False do
3:     Ordenar los individuos teniendo en cuenta la función objetivo
4:     for j = 1 : p do
5:         for k = 1 : p do
6:             if f(x^k) < f(x^j) then
7:                 Mover individuo w^j hacia w^k de acuerdo a la Ecuación(2.6)
8:             end if
9:         end for
10:    end for
11: end while
12: return Individuo con el menor valor de la función objetivo w*
```

orden en que las luciérnagas de la población se mueven en el espacio de búsqueda es significativo y por tanto los ciclos en los pasos 4 y 5 no pueden ser paralelizados.

Al considerar la paralelización del algoritmo FA, en la fase de inicialización el proceso maestro reserva de manera secuencial la memoria necesaria para los parámetros de cada individuo de la población y los patrones de entrenamiento. Posteriormente se cargan los patrones de entrenamiento en la memoria del CPU y seguidamente hacia el espacio de memoria de la GPU. Siendo esto, en la propia fase de inicialización se lleva a cabo la generación de la población en la GPU. Cada hilo de ejecución paralelo se encarga de generar cierta cantidad de números aleatorios correspondientes a los componentes w_j^i del vector de parámetros de los individuos siendo $i < p$ y $j < n$.

Una vez terminada la generación de la población, el flujo de programa vuelve al proceso maestro en la fase de comunicación. De manera secuencial, se comprueba que el algoritmo no se encuentre en un estado de convergencia. Si el algoritmo no se encuentra en estado de convergencia, se calcula el valor de la función objetivo en la GPU y se copia hacia el espacio de memoria del CPU dicho valor.

Posteriormente se realiza el movimiento de cada individuo w^j hacia w^k si $f(x^k) < f(x^j)$ en la fase de actualización. Los individuos de la población actualizan su posición de manera secuencial para no afectar las propiedades de convergencia del algoritmo, sin embargo, cada componente w_i^j es actualizado de manera paralela en la GPU. Como puede verse, la paralelización del algoritmo FA se realizó teniendo en cuenta el modelo de paralelización multi-fase con el objetivo de disminuir

la transferencia de datos entre la CPU y la GPU.

2.2.3. Algoritmo CS

El algoritmo CS fue introducido en [51] inspirado en el comportamiento del ave *cuckoo*. En su entorno natural, algunas especies del ave *cuckoo* son parásitas y depositan sus huevos en los nidos de otras aves. Con determinada frecuencia, los huevos extraños son detectados y arrojados fuera del nido. Este simple comportamiento natural es idealizado de acuerdo a los siguientes principios:

1. Los conceptos de *cuckoo*, huevo y nido representan una solución candidata en el problema de optimización indistintamente.
2. Solamente los mejores *cuckoos*, huevos o nidos de mayor calidad, de acuerdo al valor de la función objetivo, persistirán en cada generación. Las peores soluciones serán remplazadas con una probabilidad $p_a \in [0, 1]$.

Una característica de este algoritmo es que los *cuckoos*, huevos o nidos, serán generados en el espacio de búsqueda usando paseos aleatorios de *Lévy*. En la práctica los paseos aleatorios de *Lévy* muestran mayor eficiencia en la exploración de espacios de búsquedas que los paseos aleatorios convencionales. De esta forma, cuando un nuevo individuo $w^{\dagger}$ se genera a partir de un individuo w la Ecuación(2.7) es empleada.

$$w^{\dagger} \leftarrow w + \alpha \oplus \text{Lévy}(\lambda) \tag{2.7}$$

En la Ecuación(2.7) α es el tamaño del paso relacionado con la escala del problema de optimización, $\oplus$ significa *entry-wise-multiplication*. El vuelo de *Lévy* proporciona un paseo aleatorio mientras que el paso aleatorio es obtenido a partir de una distribución de *Lévy*: Lévy $\sim u = t^{-\lambda}$. El algoritmo CS posee dos hiper-parámetros: α relacionado con la escala del problema y p_a que representa la proporción de la población que es reemplazada en cada iteración.

El Algoritmo(3) describe en pseudo-código los pasos del algoritmo FA para minimizar el valor de una función objetivo. En el paso 1 del algoritmo cada individuo de la población es inicializado

aleatoriamente en el espacio de búsqueda. En el paso 2 se comprueba que el algoritmo no se encuentre en un estado de convergencia. En caso de que el algoritmo no se encuentre en un estado de convergencia, se elige un individuo de la población de manera aleatoria en el paso 3 y se genera uno nuevo realizando un paseo aleatorio a partir de este en el paso 4.

En el paso 5 del Algoritmo(3) si el valor de la función objetivo del individuo generado $w^{\dagger}$ es menor que w^{j} entonces este es remplazado en el paso 6. Posteriormente en el paso 9 una fracción de los peores individuos son remplazadas por soluciones generadas aleatoriamente en el espacio de búsqueda. Finalmente se devuelve el individuo o solución con menor valor de la función objetivo.

Algoritmo 3 Algoritmo CS para minimizar una función objetivo.

1: Inicializar aleatoriamente la población w^{i} siendo $i = 1, ..., p$
2: **while** Convergencia == False **do**
3: Encontrar un individuo w^{i} aleatoriamente
4: Generar un nuevo individuo $w^{\dagger}$ mediante la Ecuación(2.7)
5: Encontrar un individuo w^{j} aleatoriamente
6: **if** $f(w^{\dagger}, x) < f(w^{j}, x)$ **then**
7: Remplazar w^{j} con $w^{\dagger}$
8: **end if**
9: Remplazar una porción p_a de los peores individuos
10: **end while**
11: **return** Individuo con el menor valor de la función objetivo w^{*}

Al considerar la paralelización del algoritmo CS, en la fase de inicialización el proceso maestro reserva de manera secuencial la memoria necesaria para los parámetros de cada individuo de la población y los patrones de entrenamiento. Posteriormente se cargan los patrones de entrenamiento en la memoria del CPU y seguidamente hacia el espacio de memoria de la GPU. Siendo esto, en la propia fase de inicialización se lleva a cabo la generación de la población en la GPU. Cada hilo de ejecución paralelo se encarga de generar cierta cantidad de números aleatorios correspondientes a los componentes w_j^i del vector de parámetros de los individuos siendo $i < p$ y $j < n$.

Una vez terminada la generación de la población, el flujo de programa vuelve al proceso maestro en la fase de comunicación. De manera secuencial, se comprueba que el algoritmo no se encuentre en un estado de convergencia. Si el algoritmo no se encuentra en estado de convergencia, se calcula el valor de la función objetivo en la GPU y se copia hacia el espacio de memoria del CPU dicho

valor.

Posteriormente se selecciona un individuo de manera aleatoria y secuencial correspondiente al paso 3 del Algoritmo(3). La generación del nuevo individuo correspondiente al paso 4 se realiza en la GPU. Posteriormente se remplaza w^j con $w^\dagger$ realizando una copia dentro del mismo espacio de parámetros de la GPU. En el paso 9 los peores individuos son reemplazados mediante la generación de nuevos individuos de manera aleatoria en la GPU. Como puede verse, la paralelización del algoritmo CS se realizó teniendo en cuenta el modelo de paralelización multi-fase con el objetivo de disminuir la transferencia de datos entre la CPU y la GPU.

2.3. Algoritmo basado en métodos de continuación para el entrenamiento

La función objetivo de una RNP se define sobre el conjunto de parámetros de la red w y el conjunto de patrones de entrenamiento x. Las RNP se caracterizan por un incremento en la cantidad de parámetros a entrenar n y un incremento de la cantidad de patrones de entrenamiento p.

En el campo de optimización, es aceptado que el costo computacional del cálculo de la función objetivo debe ser bajo en función de evitar exceder los límites computacionales prácticos. Con el incremento de la cantidad de patrones de entrenamiento, la simplificación de la función objetivo para el entrenamiento de RNP se convierte en una tarea de importancia.

En adición a las dificultades computacionales del cálculo de la función objetivo, se ha visto un grupo de cambios y modificaciones cualitativas que afectan el entrenamiento de RNP. Entre estos cambios pueden mencionarse el aumento de la no-convexidad de las superficies de optimización, el incremento exponencial respecto a n de la cantidad de mínimos locales y regiones críticas en forma de puntos de monturas, precipicios, etc.

El empleo de algoritmos meta-heurísticos de optimización aplicados al problema de entrenamiento de RNP se encuentra actualmente limitado por el costo computacional de evaluar la función objetivo. En este epígrafe se estudia el efecto de simplificar la función objetivo para disminuir la complejidad computacional del cálculo de la función objetivo de los algoritmos meta-heurísticos

de optimización.

2.3.1. Función objetivo basada en métodos de continuación

Una función objetivo evalúa la calidad de una solución candidata en un problema de optimización. A groso modo, la función objetivo realiza una discriminación entre distintas soluciones candidatas. Si el espacio de parámetros de una RNA estuviera constituido por un parámetro, la superficie de error definida por los patrones de entrenamiento pudiera representarse como un paisaje en dos dimensiones conformado por valles y colinas.

Un individuo en un punto cualquiera de una superficie de optimización en busca de un máximo tendría que decidir en qué dirección ir. Esto se corresponde al hecho de generar un grupo de soluciones candidatas y sobre la base de cierta experiencia elegir cuál podría ser la mejor. En la práctica esa experiencia se encuentra modelada en la función objetivo. La función objetivo representa una medida de la altura o calidad de cada solución candidata.

Intuitivamente puede verse que no es necesario obtener una medición altamente precisa de la función objetivo para comparar cualitativamente soluciones candidatas. Por ejemplo, no siempre es un requisito conocer la cantidad exacta de metros de altura de dos elevaciones para estimar que una es más alta que la otra. De este modo, es posible reducir la precisión de la función objetivo sin que esto afecte la capacidad de exploración y explotación del algoritmo de optimización.

En la Figura(2.3) se ilustra una superficie de optimización sobre la cual se desea determinar cuál de dos individuos A o B es máximo. A la derecha se muestra una superficie arbitraria y a la izquierda la misma superficie de manera simplificada. A pesar que la superficie simplificada difiere de manera significativa con la superficie arbitraria original, puede verse que esta última puede emplearse para determinar con cierto grado de error que A se encuentra a mayor elevación que B.

El objetivo es obtener una representación simplificada aproximada de la superficie de optimización generada a partir de los patrones de entrenamiento. Luego emplear la versión simplificada de la función objetivo para dirigir el proceso de optimización. Idealmente la superficie resultante, tendrá las irregularidades más prominentes en su topología, al menos desde una perspectiva global.

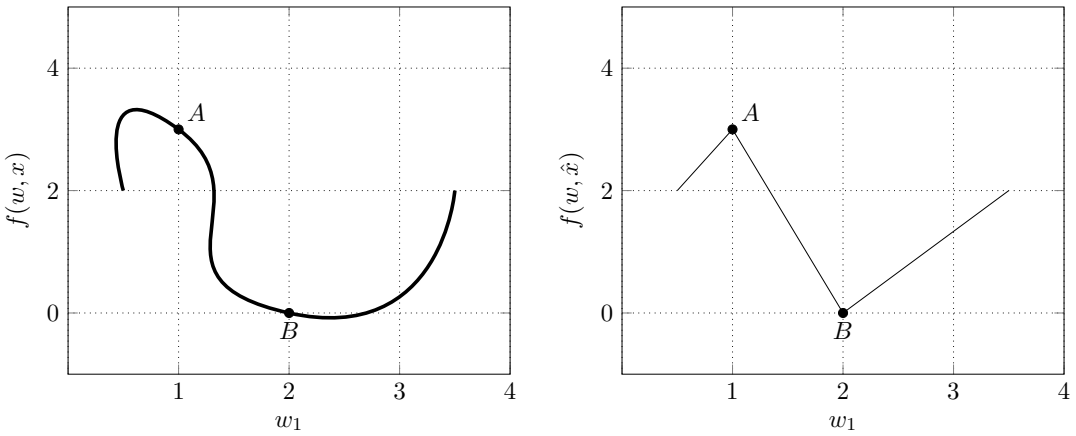


Figura 2.3: Superficie de optimización de una RNA con un único parámetro. A la izquierda una superficie arbitraria y a la derecha la superficie simplificada.

El principio detrás del empleo de versiones simplificadas de la función objetivo para dirigir el proceso de optimización ha sido abordado extensamente en la literatura y es conocido como método de continuación [98]. La idea general que sustenta los métodos de continuación es comenzar resolviendo un problema de optimización sencillo y complejizar progresivamente el problema hasta llegar a la superficie de optimización original. Para mayor claridad, el problema a tratar consiste en cómo obtener una representación aproximada simplificada de la superficie de error generada a partir de los patrones de entrenamiento presentados a una RNA.

Formalmente dado $\hat{x} \subset x$ y $w \in \mathbb{R}$ se tiene que: $f(w,x) - \epsilon \leq f(w,\hat{x}) \leq f(w,x) + \epsilon$. Aquí ϵ define una propiedad asociada a $\hat{x}$ que describe su capacidad para aproximar la superficie de optimización original.

Definición 2.3.1 *Se denomina intolerancia, al valor de ϵ asociado a un subconjunto de patrones de entrenamiento $\hat{x} \subset x$ que mide el error de $\hat{x}$ al aproximar una superficie de optimización definida por el conjunto de patrones de entrenamiento x y un conjunto de parámetros w asociados a una RNA siendo $f(w,x) - \epsilon \leq f(w,\hat{x}) \leq f(w,x) + \epsilon$ para cualquier $w \in \mathbb{R}$.*

Dado un conjunto de patrones de entrenamiento x del cual se obtiene un subconjunto $\hat{x}$ con intolerancia ϵ y los conjuntos de parámetros de dos RNA w^i y w^j. Es posible deducir el siguiente teorema:

Teorema 2.3.1 *Si $f(w^i,\hat{x}) - f(w^j,\hat{x}) > 2\epsilon$ entonces $f(w^i,x) > f(w^j,x)$.*

Demostración.

1. $f(w^i, \hat{x}) - f(w^j, \hat{x}) > 2\epsilon$ *(Axioma)*
2. $f(w^i, x) + \epsilon - (f(w^j, x) - \epsilon) > 2\epsilon$ *(Por definición de intolerancia)*
3. $f(w^i, x) > f(w^j, x)$

El Teorema(2.3.1) posibilita que un subconjunto $\hat{x}$ pueda emplearse en lugar de x para discriminar entre dos conjuntos de parámetros arbitrarios w^i y w^j con una intolerancia ϵ. Es posible obtener una cantidad arbitraria $k < |\mathcal{P}(x)|$ de subconjuntos $\hat{x}^i$ asociados a x, siendo $\mathcal{P}(x)$ el conjunto potencia de x. Cada uno de los subconjuntos $\hat{x}^i$ describe una superficie de optimización definida por los parámetros de la red y el conjunto de patrones de entrenamiento con una intolerancia ϵ^i.

Definición 2.3.2 *Se denomina sucesión representativa de conjuntos de parámetros a la sucesión conformada por* $\hat{x}^1, \hat{x}^2, ..., \hat{x}^k$ *de forma tal que* $\epsilon^i > \epsilon^{i+1}$.

Definición 2.3.3 *Se denomina sucesión restringida de conjuntos de parámetros a la sucesión conformada por* $\hat{x}^1, \hat{x}^2, ..., \hat{x}^k$ *de forma tal que* $\hat{x}^i \subset \hat{x}^{i+1}$.

A partir de la Definición(2.3.2) y la Definición(2.3.3) es posible afirmar que la segunda es un caso particular de la primera. El Teorema(2.3.2) establece la relación entre las sucesiones restringidas y representativas de conjuntos de parámetros.

Teorema 2.3.2 *Si una sucesión de conjuntos de parámetros es restringida entonces también es representativa.*

Demostración. *A partir de las definiciones anteriores, el valor de* ϵ *puede verse como un error que mide la diferencia de* $f(w, \hat{x})$ *respecto a* $f(w, x)$. *A partir de esto se tiene que:*

1. $f(w, \hat{x}^i) \leq f(w, x) + \epsilon^i$ *(Axioma)*
2. $f(w, \hat{x}^{i+1}) \leq f(w, x) + \epsilon^{i+1}$ *(Axioma)*

por definición también se tiene que:

3. $|\hat{x}^i| < |\hat{x}^{i+1}|$ *(Por definición de sucesión restringida)*

   Al considerar un análisis de la dimensión de Vapnik-Chervonenkis [99], se puede asegurar con un nivel de confianza de $1-\delta$ *que la cantidad de patrones de entrenamiento de cada subconjunto de patrones de entrenamiento obedece las siguientes inecuaciones:*

4. $|\hat{x}^i| \geq \text{máx}\,(\frac{4}{f(w,x)+\epsilon^i}\log\frac{2}{\delta}, \frac{8d^\dagger}{f(w,x)+\epsilon^i}\log\frac{13}{f(w,x)+\epsilon^i})$

5. $|\hat{x}^{i+1}| \geq \text{máx}\,(\frac{4}{f(w,x)+\epsilon^{i+1}}\log\frac{2}{\delta}, \frac{8d^\dagger}{f(w,x)+\epsilon^{i+1}}\log\frac{13}{f(w,x)+\epsilon^{i+1}})$

   donde $d^\dagger$ *es la dimensión de Vapnik-Chervonenkis de la RNA. En este caso,* $d^\dagger$ *se considera constante, debido a que el modelo de RNA es el mismo para cada subconjunto de patrones de entrenamiento* $\hat{x}^i$*.*

   A partir de 4 y 5 puede verse que ϵ^i *y* ϵ^{i+1} *son inversamente proporcionales a* $|\hat{x}^i|$ *y* $|\hat{x}^{i+1}|$ *respectivamente. De ahí que al considerar 3 pueda asegurarse que:*

6. $\epsilon^i > \epsilon^{i+1}$ *concluyendo la prueba del teorema.*

En la práctica, el cálculo de ϵ implica resolver un problema de optimización descrito por la Ecuación(2.8). Tal problema de optimización tiene características computacionales similares al problema de optimización referido en la Ecuación(1.6) en el Capítulo(1) que se intenta simplificar. De ahí que, el cálculo de ϵ resulte impráctico a los efectos de esta solución.

$$\epsilon = \max_{w\in\mathbb{R}^n} |f(w,x) - f(w,\hat{x})| \tag{2.8}$$

Teniendo en cuenta el Teorema(2.3.2) es relativamente sencillo obtener una sucesión de conjuntos representativos de parámetros. La dificultad radica en la aplicabilidad del Teorema(2.3.1) para lo cual es necesario conocer ϵ. Sin embargo, a partir del Teorema(2.3.1) se tiene que:

COROLARIO 2.3.1 *Si* $f(w^i,\hat{x}) > f(w^j,\hat{x})$ *entonces* $f(w^i,x) > f(w^j,x)$ *con una probabilidad de* $\xi = 1/(1+\epsilon)$.

Partiendo del Corolario(2.3.1) es posible encontrar una sucesión restringida de conjuntos de parámetros cualquiera que termine con un subconjunto cercano a todo el conjunto de patrones de entrenamiento $\hat{x}^k \approx x$ de forma tal que la probabilidad $\xi^k \approx 1$. En el Algoritmo(4) se detallan en pseudo-código los pasos para el uso de un enfoque basado en métodos de continuación para el cálculo de la función objetivo.

Algoritmo 4 Optimización de una RNA empleando una función objetivo basada en métodos de continuación.

```
1: Encontrar una sucesión representativa x̂^1, ..., x̂^k
2: for s = 1 : k do
3:     while (Condición de complejización == False) do
4:         Paso de optimización usando f(w, x̂^s)
5:     end while
6: end for
```

En el paso 1, se encuentra una sucesión representativa de subconjuntos de patrones de entrenamiento. La construcción de la sucesión restringida de subconjuntos de patrones de entrenamiento se realiza mediante la selección de una proporción $\beta \in [0, 1]$ patrones de entrenamiento en $\hat{x}^i$ respecto a $\hat{x}^{i+1}$.

En el paso 2 se establece el subconjunto de patrones de entrenamiento a emplear. Se comienza con el primer elemento de la sucesión representativa y se termina con la superficie de optimización definida por el conjunto de entrenamiento completo. Mientras no se cumpla la condición de complejización en el paso 3, se efectúa un paso de optimización en el paso 4 empleando un algoritmo meta-heurístico arbitrario denominado algoritmo base.

En este trabajo se estudia únicamente el criterio de complejización basado en cantidad fija de llamadas a la función objetivo con el objetivo de facilitar la comparación de los métodos de continuación. Para una cantidad total η de evaluaciones de la función objetivo cada subconjunto de patrones de entrenamiento se emplea por una cantidad η/k de evaluaciones de la función objetivo.

En principio, el algoritmo base puede ser cualquier algoritmo meta-heurístico. Los errores cometidos a partir de probabilidad involucrada en el Corolario(2.3.1), añaden un comportamiento aleatorio a la evaluación de la función objetivo. Para una meta-heurística que incorpora un componente estocástico dicha aleatoriedad se considera saludable. De esta forma se plantea que el

Algoritmo(4) es agnóstico respecto al algoritmo base.

2.3.2. Análisis de los subconjuntos de patrones de entrenamiento

Al reducir la cantidad de patrones de entrenamiento en los subconjuntos de patrones de entrenamiento, un análisis del peor caso en la dimensión Vapnik-Chervonenkis sugiere que esta reducción causa necesariamente un aumento en el error del modelo predictivo [99].

$$n \geq \text{máx}\,(\frac{4}{\epsilon}\log(\frac{2}{\delta}), \frac{8d}{\epsilon}\log(\frac{13}{\epsilon})) \tag{2.9}$$

En la Ecuación(2.9) n es la cantidad de patrones de entrenamiento, d es la dimensión de Vapnik-Chervonenkis que en este caso es constante debido a que el espacio de hipótesis para cada subconjunto de patrones de entrenamiento es el mismo (no se cambia la arquitectura de la RNA entrenada) y ϵ es una cota superior del error cometido con nivel de confianza de $1-\delta$.

Esta cota de error es el resultado de un análisis del peor caso, por lo que en la práctica es posible obtener una tasa de error inferior con una cantidad más pequeña de patrones de entrenamiento [100]. Nótese que el error ϵ en la Ecuación(2.9) es inversamente proporcional a la cantidad de patrones de entrenamiento n. De esta forma, en esta sección se realiza una prueba de concepto para determinar cómo se comporta el error de la RNA al reducir la cantidad de patrones de entrenamiento.

Para la realización de la prueba de concepto se realizó una medición del MSE al emplear el 20 %, 40 %, 60 %, 80 % y 100 % de los patrones de entrenamiento. El MSE se obtiene considerando tres bases de datos de prueba: *Concrete Compresive Dataset* (CCS), *Wine Quality Dataset* (WQ) y *Combined Cycle Power Plant Dataset* (CCPP) disponible públicamente en internet[3].

En la Tabla(2.1) se muestra los resultados de estas mediciones para el algoritmo SGD luego de 100 épocas usando un tamaño de *mini-batch* de 30 patrones de entrenamiento y el algoritmo FA luego de 4 mil evaluaciones de la función objetivo. Referirse a [101] para una descripción detallada de la configuración experimental de esta prueba.

[3]Disponible en: http://archive.ics.uci.edu/ml/

TABLA 2.1: Medición del MSE al emplear menor cantidad de patrones de entrenamiento.

Proporción	CCS		WQ		CCPP	
	MSE FA	MSE SGD	MSE FA	MSE SGD	MSE FA	MSE SGD
20 %	0.0156	0.0238	0.0147	0.0444	0.00331	0.0529
40 %	0.0163	0.0226	0.0146	0.0445	0.00332	0.0523
60 %	0.0160	0.0228	0.0145	0.0463	0.00331	0.0515
80 %	0.0166	0.0219	0.0144	0.0422	0.00327	0.0519
100 %	0.0166	0.0218	0.0145	0.0432	0.00331	0.0511

Los resultados muestran que el MSE de los subconjuntos de patrones de entrenamiento se comporta de manera similar al MSE cuando todos los patrones de entrenamiento son empleados en el entrenamiento mediante el algoritmo FA. Sin embargo, este comportamiento no se manifiesta en el caso del algoritmo SGD, donde la precisión suele decrecer al emplear una menor cantidad de patrones de entrenamiento.

Los resultados sugieren que a pesar de la pérdida de información al emplear una menor cantidad de patrones de entrenamiento, la información global de la superficie de optimización no se pierde aunque el error crezca ligeramente. Lo anterior permitiría acotar el espacio de búsqueda en las iteraciones iniciales e ir aumentando la explotación hacia las iteraciones finales de la optimización en la medida que se aumenta dinámicamente la cantidad de patrones de entrenamiento.

2.4. Conclusiones parciales

A partir del análisis de los algoritmos meta-heurísticos aplicados al entrenamiento neuronal realizado en el presente capítulo es posible arribar a las siguientes conclusiones parciales:

1. El modelo de paralelización multi-fase de los algoritmos meta-heurísticos provee un marco de trabajo eficiente para su implementación paralela disminuyendo la transferencia de datos entre el espacio de memoria de la GPU y la CPU.

2. La paralelización del cálculo de la función objetivo de una RNA puede ser modelada eficientemente como un problema de multiplicación de matrices para el cual existen soluciones probadas en la literatura.

3. Los algoritmos basados en métodos de continuación para el cálculo de la función objetivo de las RNA, permite disminuir el tamaño efectivo de las matrices que se emplean en el cálculo de la función objetivo disminuyendo el tiempo de ejecución sin afectar la precisión del entrenamiento.

Capítulo 3

Resultados y discusión

En el presente capítulo se exponen los resultados obtenidos a partir de un conjunto de pruebas realizadas a los algoritmos meta-heurísticos propuestos. Se describe la biblioteca para el entrenamiento de RNA desarrollada y se presentan los resultados de pruebas conceptuales de rendimiento realizadas a la implementación de los algoritmos paralelos.

Posteriormente, se describe la metodología seguida para el diseño de los experimentos para el análisis de la precisión del entrenamiento y el tiempo de ejecución de los algoritmos PSO, FA y CS empleando la función objetivo basada en métodos de continuación introducida en el Capítulo(2). Finalmente se realiza una valoración de los resultados obtenidos, tomando como referencia un grupo de bases de datos de aprendizaje públicas.

3.1. Biblioteca para el entrenamiento de RNA

Para la realización de los experimentos se desarrolló una biblioteca de clases en el lenguaje de programación C++11. La biblioteca presenta un núcleo de clases sin dependencias con bibliotecas de terceros y adicionalmente un grupo de capas que permiten extender las funcionalidades del núcleo. El núcleo presenta implementaciones secuenciales de los algoritmos de optimización (PSO, CS y FA) además de las implementaciones necesarias de los modelos computacionales de RNA.

Para extender las implementaciones secuenciales del núcleo de la biblioteca, se agregaron funcionalidades para la implementación en paralelo de los algoritmos propuestos. La biblioteca está diseñada teniendo en cuenta una arquitectura n-capas con interfaces desacopladas que permite abstraer al usuario de los detalles de implementación en paralelo.

La biblioteca cuenta con tres capas sobre un núcleo con implementaciones secuenciales: una capa con implementaciones secuenciales optimizadas de los algoritmos propuestos, una capa

con implementaciones en paralelo para CPU y una capa con implementaciones en paralelo para GPU. A continuación, se describen las bibliotecas de terceros incluidas en la implementación de cada capa:

- En la **capa secuencial optimizada** se empleó la biblioteca BLAS [102] que implementa un grupo de operaciones básicas del álgebra lineal de forma eficiente. La biblioteca BLAS se encuentra optimizada de forma tal que permite emplear los recursos de la CPU como la vectorización AVX y SSE [103] disminuyendo el tiempo de ejecución de los algoritmos.
- En la **capa con implementaciones en paralelo para CPU** se empleó la biblioteca OpenMP [104]. OpenMP es una biblioteca que permite abstraer los complejos detalles de plataforma de manera eficiente y escalable para la paralelización en CPU multi-núcleo.
- En la **capa con implementaciones en paralelo para GPU** se empleó las bibliotecas Thrust [105], cuBLAS [106] y cuRAND [107]. Thrust es una biblioteca de alto nivel que posee implementaciones optimizadas en GPU de un grupo de primitivas paralelas. La biblioteca cuBLAS implementa en GPU un grupo de operaciones básicas del álgebra lineal de forma similar a BLAS. Finalmente, cuRAND permite generar de forma eficiente secuencias de números aleatorios en paralelo.

3.2. Diseño de los experimentos

Para la validación de la hipótesis planteada se estudia las diferencias en el entrenamiento neuronal de los algoritmos basados en gradiente y los algoritmos meta-heurísticos respecto a la precisión. Posteriormente se estudia la diferencia de los algoritmos meta-heurísticos (PSO, FA, CS) y sus variantes basadas en continuación considerando la precisión y el tiempo de ejecución. Teniendo esto en cuenta, se definieron las siguientes variables de investigación:

Variable independiente:

1. Algoritmo de entrenamiento. La variable es operacionalizada teniendo en cuenta el algoritmo SGD, los algoritmos de entrenamiento basados en meta-heurísticas de optimización (PSO, FA, CS) y sus variantes basadas en continuación. Esta variable se define como

categórica y se establece un nivel de medición ordinal.

Variables dependientes:

1. Precisión. La variable es operacionalizada teniendo en cuenta el error de la RNA una vez finalizado el proceso de entrenamiento. La variable se encuentra definida por dos dimensiones: error de entrenamiento y error de generalización. El error de entrenamiento es una dimensión continua con nivel de medición de razón calculado mediante el MSE de la RNA empleando el conjunto de patrones de entrenamiento. El error de generalización es una dimensión continua con nivel de medición de razón calculado mediante el MSE de la RNA empleando el conjunto de patrones de validación.

2. Eficiencia. La variable es operacionalizada teniendo en cuenta el tiempo de ejecución en segundos que transcurre desde el inicio del entrenamiento hasta que el algoritmo de entrenamiento termina. La variable se encuentra definida por dos dimensiones: tiempo secuencial y tiempo paralelo. El tiempo secuencial es una dimensión continua con nivel de medición de razón calculado como el tiempo en segundos para una implementación secuencial de los algoritmos. El tiempo paralelo es una dimensión continua con nivel de medición de razón calculado como el tiempo en segundos para una implementación paralela de los algoritmos.

3.2.1. Configuración de hiper-parámetros

Los hiper-parámetros de un algoritmo son aquellos que no son optimizados directamente por el algoritmo de entrenamiento. La mayor distinción que se tuvo en cuenta en las pruebas realizadas fue respecto a los hiper-parámetros relacionados con la capacidad de los modelos neuronales (patrones de conectividad, cantidad de neuronas, etc.) y los hiper-parámetros relacionados con los algoritmos de entrenamiento. En el Epígrafe(1.3) se abordó sobre las limitaciones encontradas en la literatura. Referirse a [33] para una ampliación en el tema de la optimización de hiper-parámetros.

Para la configuración de los hiper-parámetros relacionados con la capacidad de los modelos neuronales se consideró los resultados obtenidos en trabajos previos. En el Epígrafe(3.2.2) se describen las bases de datos correspondientes a diferentes problemas de aprendizaje que se

tuvieron en cuenta con los respectivos modelos neuronales a considerar.

Para los hiper-parámetros relacionados con los algoritmos de entrenamiento se consideró el algoritmo SMAC [108] para su configuración automática. Cada algoritmo meta-heurístico posee una población de 40 individuos establecida arbitrariamente con propósitos de comparación y no exceder los límites computacionales prácticos. Adicionalmente, se considera el método *Tibshirani-Tibshirani* (TT) para calcular el sesgo introducido por la optimización de los hiper-parámetros [90] como una alternativa al método *nested-cross-validation*, ver [91] para una comparación.

Para la configuración de los hiper-parámetros se realizó 80 iteraciones del algoritmo SMAC en los dominios de hiper-parámetros especificados en la Tabla(3.1). Los hiper-parámetros empleados se detallan con el objetivo de proporcionar un punto de partida para futuras investigaciones y para reproducir los resultados obtenidos.

TABLA 3.1: Configuración de SMAC para cada algoritmo meta-heurístico.

Algoritmo	Hiper-parámetro	Dominio	Descripción
SGD	Longitud de paso(λ)	[0, 1,0]	Ratio de aprendizaje.
	Inercia(ν)	[0, 1,0]	Factor inercial
PSO	Global(α)	[0, 1,0]	Atracción hacia el mejor individuo global.
	Local(β)	[0, 1,0]	Atracción hacia el mejor individuo local.
	Velocidad Máxima(max)	[0, 1,0]	Factor inercial máximo.
	Velocidad Mínima(min)	[0, 1,0]	Factor inercial mínimo.
FA	Brillo (β)	[0, 1,0]	Brillo inicial cuando dos luciérnagas están a distancia cero.
	Atenuación (γ)	[0, 0,01]	Atenuación del brillo de la luciérnaga con la distancia.
	Aleatoriedad (η)	[0, 1,0]	Cantidad de aleatoriedad incorporado al movimiento de la luciérnaga.
CS	Escala (α)	[0, 1,0]	Escala del problema de optimización.
	Reemplazo(p_a)	[0, 1,0]	Proporción de nidos reemplazados en cada iteración.
Continuación	Proporción($\beta^{\dagger}$)	[0,4, 1,0]	Proporción de patrones de entrenamiento contenidos en x^i respecto a x^{i+1}.
	Subconjuntos(K)	[0, 10]	Cantidad de subconjuntos de patrones de entrenamiento a emplear en la sucesión representativa de subconjuntos de patrones de entrenamiento.

3.2.2. Selección de las bases de datos

Para la realización de las mediciones se empleó un conjunto de bases de datos de aprendizaje de referencia internacional. En función de evaluar los algoritmos de optimización se tuvo en cuenta la cantidad de patrones de entrenamiento en cada base de datos y los resultados previos en el estado del arte como punto de partida en el análisis realizado. De esta forma, cada uno de los problemas modelados en las bases de datos han sido trabajados mediante RNA anteriormente. Las bases de datos 1-3 fueron elegidas del repositorio de problemas de aprendizaje UCI [109] disponible públicamente en Internet[1].

1. *Concrete Compressive Strength Dataset* (CCS). La base de datos está relacionada con la fortaleza compresiva del concreto. La fortaleza compresiva del concreto involucra el tiempo y los ingredientes empleados en su confección. El objetivo consiste en predecir la fortaleza compresiva del concreto caracterizado por sus atributos o ingredientes. La base de datos contiene 1030 instancias, para las cuales se consideran 9 atributos modelando un problema de regresión. Las pruebas se realizaron empleando un perceptrón multi-capa con 8 unidades de entrada, 14 unidades intermedias y 1 unidad de salida para un total de 141 parámetros de acuerdo a los resultados obtenidos en [110].

2. *Wine Quality Dataset* (WQ). La base de datos está relacionada con muestras de vino tinto y vino blanco tomadas en el norte de Portugal. El objetivo es predecir la calidad del vino basado en un grupo de pruebas físico-químicas. La base de datos referente a la calidad del vino blanco contiene 4898 instancias y 12 atributos modelando un problema de regresión. Se considera un modelo de RNA del tipo perceptrón multi-capa con 11 unidades de entrada, 14 unidades intermedias y 1 unidad de salida para un total de 183 parámetros de acuerdo a los resultados obtenidos en [111].

3. *Combined Cycle Power Plant Dataset* (CCPP). La base de datos contiene información relacionada con el ciclo de producción de energía de una planta durante 6 años. Los atributos consisten en variables ambientales como la presión, la temperatura y la humedad relativa. El objetivo es predecir la cantidad de energía eléctrica que la planta produce cada hora. La base de datos posee 9568 instancias y 5 atributos modelando un problema de regresión.

[1] Disponible en https://archive.ics.uci.edu/ml

Las pruebas se llevaron a cabo con una RNA de tipo perceptrón muti-capa con 4 unidades de entrada, 17 unidades intermedias y 1 unidad de salida para un total de 103 parámetros de acuerdo a los resultados obtenidos en [55].

3.3. Resultados experimentales

En este epígrafe se describen los experimentos realizados para validar la hipótesis planteada de la investigación. En lo subsiguiente se realiza una comparación entre los algoritmos de entrenamiento basados en gradiente y los algoritmos meta-heurísticos (PSO, FA, CS) teniendo en cuenta la precisión del entrenamiento. Para este experimento no se realiza una evaluación del tiempo de ejecución pues se descarta la posibilidad de que un algoritmo meta-heurístico basado en población pueda superar a un algoritmo basado gradiente teniendo en cuenta estas variables en un esquema secuencial o en GPU como los estudiados en este trabajo.

Posteriormente, se describen los experimentos realizados en la comparación de los algoritmos meta-heurísticos y sus variantes basadas en métodos de continuación. Un análisis de la precisión y tiempo de ejecución secuencial y paralelo según corresponda se presenta en cada caso.

3.3.1. Comparación de SGD y los algoritmos meta-heurísticos

En esta sección se compara la precisión del algoritmo SGD respecto a los algoritmos meta-heurísticos en el entrenamiento de RNA. Se consideran los algoritmos meta-heurísticos PSO, FA y CS.

En la Tabla(3.2) se muestran los hiper-parámetros optimizados por el algoritmo SMAC para cada algoritmo de optimización considerando las bases de datos de aprendizaje empleadas. Para cada juego de hiper-parámetro se presenta el valor del sesgo TT que influye en el cálculo del error de geneneralización.

TABLA 3.2: Configuración de los hiper-parámetros de los algoritmos de optimización en la comparación de algoritmos basados en gradiente y meta-heurísticos.

	SGD			PSO					FA				CS		
	λ	ν	TT	α	β	max	min	TT	β	γ	η	TT	α	p_a	TT
CSS	0.130	0.166	1.28E-4	0.938	0.366	1.254	0.763	1.36E-3	0.936	2.56E-4	0.603	3.15E-4	0.181	0.517	2.15E-4

Continuación de la página anterior.

	SGD			PSO					FA				CS		
	λ	ν	TT	α	β	max	min	TT	β	γ	η	TT	α	p_a	TT
WQ	0.772	0.059	4.59E-4	0.938	0.366	1.254	0.763	2.39E-4	0.936	2.56E-4	0.603	2.52E-4	0.685	0.775	1.82E-4
CCPP	0.036	0.025	6.47E-5	0.938	0.366	1.254	0.763	1.85E-5	0.913	9.81E-4	0.256	5.00E-7	0.994	0.968	6.80E-7

Para la medición de la precisión se considera el MSE siguiendo la metodología descrita en el Epígrafe(3.2) teniendo en cuenta el sesgo introducido por la configuración de hiper-parámetros calculado mediante el método TT. Se considera el entrenamiento en estado de convergencia para los algoritmos meta-heurísticos, cuando la cantidad de evaluaciones de la función objetivo sobrepase una cantidad $\eta = 40000$. Para los algoritmos basados en gradiente se emplea la versión *mini-batch* con una cantidad de 30 patrones de entrenamiento realizando 100 épocas. Los criterios de convergencia para los algoritmos de optimización se determinaron mediante una búsqueda de rejilla [3].

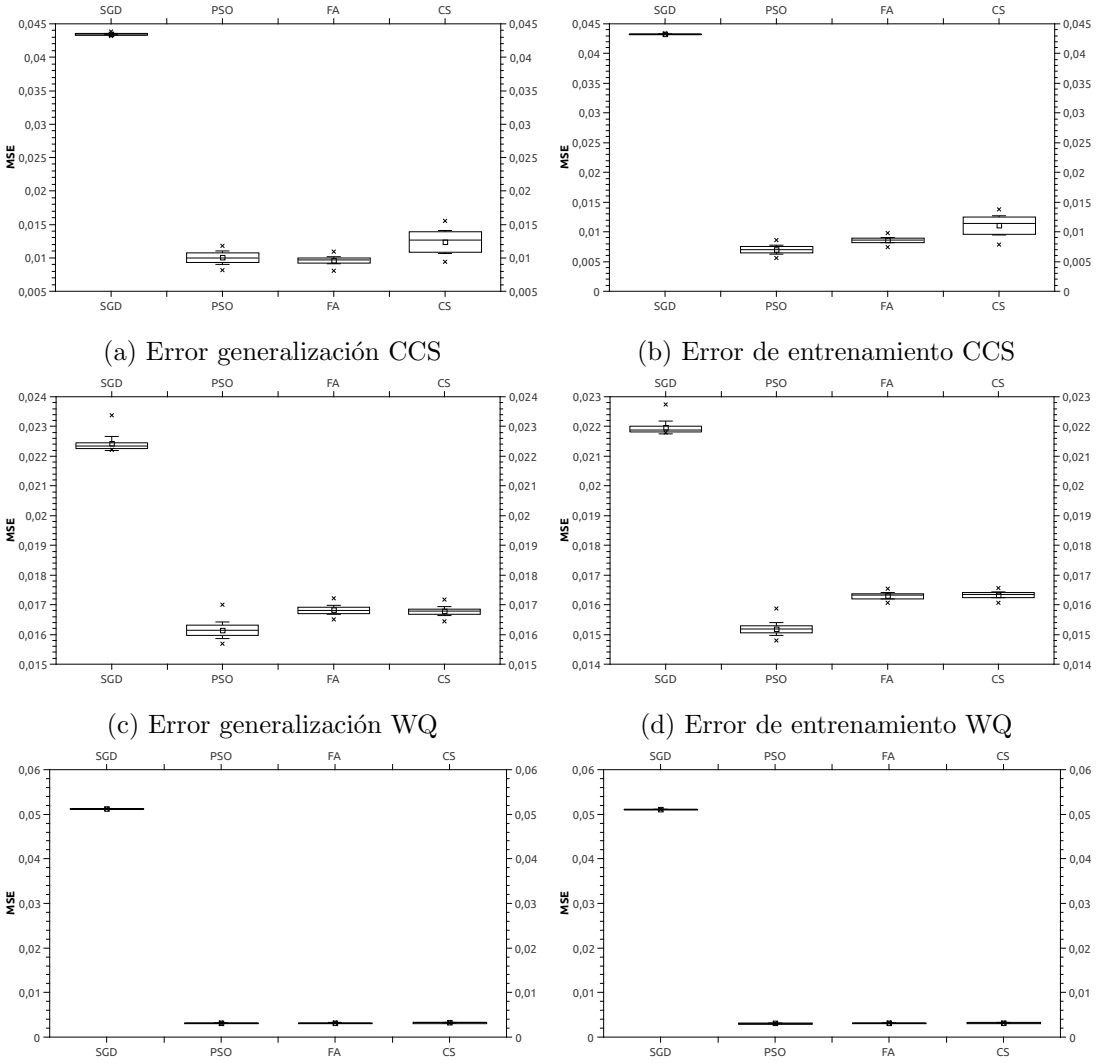
En la Figura(3.1) se muestran los resultados obtenidos considerando la precisión de los distintos algoritmos de entrenamiento aplicados a los problemas de aprendizaje definidos en 50 mediciones. En el eje horizontal se grafican los algoritmos utilizados en el entrenamiento y en el eje vertical la precisión obtenida. La parte inferior y superior de las cajas representan el primer y tercer cuartil respectivamente. La linea divisoria indica la mediana de las mediciones. Los brazos de las cajas representan la desviación estándar.

En la Tabla(3.3) se detallan los resultados graficados en la Figura(3.1). Para cada algoritmo y problema de aprendizaje se especifica la media y la desviación estándar de la precisión obtenida en formato $\mu(\sigma)$.

TABLA 3.3: Error de generalización y entrenamiento de los algoritmos de entrenamiento basados en gradiente y meta-heurísticos.

	SGD	PSO	FA	CS
	Error de Generalización			
CSS	4.34E-2(1.28E-4)	1.00E-2(9.99E-4)	9.62E-3(5.25E-4)	1.20E-2(1.69E-3)
WQ	2.24E-2(2.41E-4)	1.61E-2(2.76E-4)	1.68E-2(1.40E-4)	1.67E-2(1.43E-4)
CCPP	5.12E-2(1.64E-5)	3.13E-3(3.79E-5)	3.21E-3(2.30E-5)	3.23E-3(2.87E-5)
	Error de Entrenamiento			
CSS	4.33E-2(2.02E-5)	6.98E-3(7.36E-4)	8.61E-3(4.50E-4)	1.11E-2(1.52E-3)
WQ	2.19E-2(2.13E-4)	1.51E-2(2.05E-4)	1.63E-2(1.03E-4)	1.63E-2(1.05E-4)
CCPP	5.11E-2(5.07E-6)	3.09E-3(3.67E-5)	3.20E-3(2.21E-5)	3.22E-3(2.61E-5)

La estrategia de validación estadística consiste en comprobar la normalidad de las mediciones



(a) Error generalización CCS

(b) Error de entrenamiento CCS

(c) Error generalización WQ

(d) Error de entrenamiento WQ

(e) Error generalización CCPP

(f) Error de entrenamiento CCPP

Figura 3.1: Precisión del algoritmo SGD respecto a los algoritmos meta-heurísticos para el entrenamiento de RNA.

obtenidas mediante la prueba de Anderson-Darling. Posteriormente se comprueba si existe diferencia estadísticamente significativa entre las medias del error de generalización de los algoritmos basados en gradiente y los algoritmos meta-heurísticos mediante la prueba de T-Student. En la Tabla(3.4) se muestran los resultados de la prueba de T-Student en la comparación de las medias de los algoritmos basados en gradiente y los algoritmos meta-heurísticos. Los resultados muestran que existe una mejora estadísticamente significativa de la precisión para las tres bases de datos al emplear los algoritmos meta-heurísticos estudiados respecto a SGD.

TABLA 3.4: Resultados de la prueba de T-Student en la comparación de la media del error de generalización del algoritmo SGD respecto a los algoritmos meta-heurísticos.

	CCS			WQ			CCPP		
	PSO	FA	CS	PSO	FA	CS	PSO	FA	CS
P-VALOR	<1.0E-4	<1.0E-4	<1.0E-4	<1.0E-4	<1.0E-4	<1.0E-4	<1.0E-4	<1.0E-4	<1.0E-4
T-VALOR	232.44	438.34	136.12	120.21	141.22	141.98	8142.46	11899.85	10897.74

Debido a que los algoritmos PSO, FA y CS son basados en población, es necesario evaluar la función objetivo para cada uno de los individuos de la población. Esto provoca un aumento en el tiempo de ejecución debido al costo computacional asociado. En las siguientes secciones se estudia el efecto de los algoritmos meta-heurísticos basados en continuación en la precisión y el tiempo de ejecución del entrenamiento de RNA.

3.3.2. Comparación de los algoritmos meta-heurísticos y sus variantes basadas en continuación

En esta sección se comparan los algoritmos meta-heurísticos respecto a los algoritmos meta-heurísticos basadas en métodos de continuación en el entrenamiento de RNA. Se consideran los algoritmos meta-heurísticos PSO, FA y CS y sus respectivas extensiones basadas en métodos de continuación.

Análisis de la precisión del entrenamiento

Para la medición de la precisión se emplea el MSE siguiendo la metodología descrita en el Epígrafe(3.2). Se tiene en cuenta el sesgo introducido por la configuración de hiper-parámetros calculado mediante el método TT. Se considera el entrenamiento en estado de convergencia para los algoritmos meta-heurísticos cuando la cantidad de evaluaciones de la función objetivo realizadas sobrepase cierto umbral η. En las pruebas realizadas se empleó un umbral $\eta = 40000$ evaluaciones de la función objetivo. Para las variantes basadas en métodos de continuación, cada subconjunto de patrones de entrenamiento se emplea durante η/k evaluaciones de la función objetivo para un total de η iteraciones.

En la Tabla(3.5) se muestran los hiper-parámetros optimizados por el algoritmo SMAC para cada algoritmo de optimización considerando las bases de datos de aprendizaje empleadas. Para

cada juego de hiper-parámetro se presenta el valor del sesgo TT que influye en el cálculo del error de geneneralización.

TABLA 3.5: Configuración de los hiper-parámetros de los algoritmos de optimización en la comparación de algoritmos meta-heurísticos y sus variantes basadas en continuación.

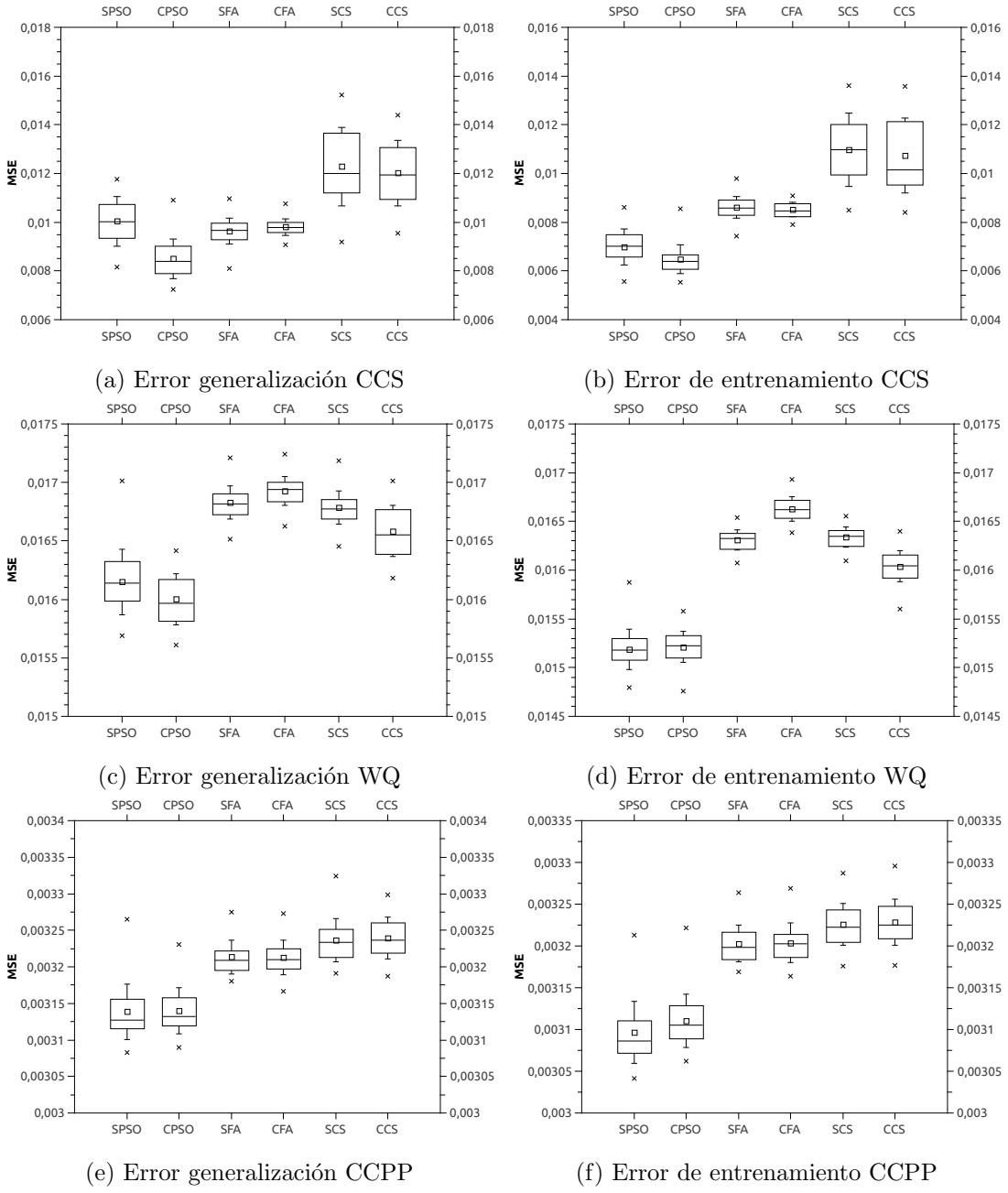
	PSO							FA						CS					
	α	β	max	min	$\beta^{\dagger}$	K	TT	β	γ	η	$\beta^{\dagger}$	K	TT	α	p_a	$\beta^{\dagger}$	K	TT
	Algoritmos meta-heurísticos estándar																	
CSS	0.938	0.366	1.254	0.763	-	-	1.36E-3	0.936	2.56E-4	0.603	-	-	3.15E-4	0.181	0.517	-	-	2.15E-4
WQ	0.938	0.366	1.254	0.763	-	-	2.39E-4	0.936	2.56E-4	0.603	-	-	2.52E-4	0.685	0.775	-	-	1.82E-4
CCPP	0.938	0.366	1.254	0.763	-	-	1.85E-5	0.913	9.81E-4	0.256	-	-	5.00E-7	0.994	0.968	-	-	6.80E-7
	Algoritmos meta-heurísticos basados en continuación																	
CSS	0.911	0.522	1.063	0.802	0.817	4	4.03E-4	0.890	8.33E-4	0.264	0.832	6	4.50E-5	0.063	0.727	0.635	5	8.31E-5
WQ	0.760	0.883	0.940	0.784	0.518	4	1.12E-4	0.752	6.91E-4	0.067	0.822	2	1.43E-4	0.221	0.974	0.829	2	6.70E-4
CCPP	0.648	0.718	1.073	0.827	0.464	2	1.05E-5	0.977	1.13E-2	0.081	0.667	8	3.30E-8	0.063	0.727	0.635	8	9.20E-8

En la Figura(3.2) se muestran los resultados obtenidos considerando la precisión de los distintos algoritmos de entrenamiento aplicados a los problemas de aprendizaje definidos en 50 mediciones. La parte inferior y superior de las cajas representan el primer y tercer cuartil respectivamente. La línea divisoria en el centro de la caja indica la mediana de las mediciones. Los brazos de las cajas representan la desviación estándar. En la figura, el prefijo S delante del nombre del algoritmo meta-heurístico significa estándar y el prefijo C significa continuación.

En la Tabla(3.6) se detallan los resultados de la Figura(3.2). Para cada algoritmo y problema de aprendizaje se especifica la media y la desviación estándar de la precisión obtenida en formato $\mu(\pm\sigma)$.

TABLA 3.6: Precisión de los algoritmos meta-heurísticos y sus variantes basadas en métodos de continuación.

	Función objetivo estándar			Continuación		
	SPSO	SFA	SCS	CPSO	CFA	CCS
	Error de generalización					
CCS	1.00E-2(9.99E-4)	9.62E-3(5.25E-4)	1.20E-2(1.69E-3)	8.50E-3(7.95E-4)	9.79E-3(3.40E-4)	1.16E-2(1.53E-3)
WQ	1.61E-2(2.76E-4)	1.68E-2(1.40E-4)	1.67E-2(1.43E-4)	1.60E-2(2.16E-4)	1.69E-2(1.23E-4)	1.65E-2(2.10E-4)
CCPP	3.13E-3(3.79E-5)	3.21E-3(2.30E-5)	3.23E-3(2.87E-5)	3.13E-3(3.12E-5)	3.21E-3(2.33E-5)	3.23E-3(2.66E-5)
	Error de entrenamiento					
CCS	6.98E-3(7.36E-4)	8.61E-3(4.50E-4)	1.11E-2(1.52E-3)	6.48E-3(5.71E-4)	8.51E-3(2.97E-4)	1.10E-2(1.51E-3)
WQ	1.51E-2(2.05E-4)	1.63E-2(1.03E-4)	1.63E-2(1.05E-4)	1.52E-2(1.54E-4)	1.66E-2(1.24E-4)	1.60E-2(1.55E-4)
CCPP	3.09E-3(3.67E-5)	3.20E-3(2.21E-5)	3.22E-3(2.61E-5)	3.11E-3(3.17E-5)	3.20E-3(2.36E-5)	3.23E-3(2.96E-5)



(a) Error generalización CCS

(b) Error de entrenamiento CCS

(c) Error generalización WQ

(d) Error de entrenamiento WQ

(e) Error generalización CCPP

(f) Error de entrenamiento CCPP

Figura 3.2: Precisión de los algoritmos meta-heurísticos y sus variantes basadas en métodos de continuación.

La estrategia de validación estadística consiste en comprobar la normalidad de las mediciones obtenidas mediante la prueba de Anderson-Darling. Posteriormente se comprueba si existe diferencia estadísticamente significativa entre las medias del error de generalización de los algoritmos meta-heurísticos y los algoritmos meta-heurísticos basados en continuación mediante la prueba

de T-Student.

En la Tabla(3.7) se muestran los resultados de la prueba de T-Student en la comparación de las medias de los algoritmos algoritmos meta-heurísticos y sus variantes basadas en continuación. En negrita se destacan las mediciones donde se detectaron diferencias estadísticamente significativas.

TABLA 3.7: Resultados de la prueba de T-Student en la comparación de la media del error de generalización de los algoritmos meta-heurísticos y sus variantes basadas en continuación.

	CCS			WQ			CCPP		
	PSO	FA	CS	PSO	FA	CS	PSO	FA	CS
P-VALOR	**<1.0E-4**	0.06	0.09	**4.7E-3**	**5.0E-3**	**<1.0E-4**	0.82	0.95	0.61
T-VALOR	**8.41**	1.89	1.68	**2.89**	**3.62**	**5.32**	0.21	0.06	0.50

Los resultados obtenidos muestran que existe una pequeña mejora estadísticamente significativa de la precisión del algoritmo SPSO respecto al algoritmo CPSO para la base de datos CCS. En el caso de la base de datos WQ se observa una mejora estadísticamente significativa entre el algoritmo SPSO respecto al algoritmo CPSO y el algoritmo SCS respecto a CCS. Para el resto de las bases de datos, no existen diferencias estadísticamente significativas entre los algoritmos meta-heurísticos y los algoritmos meta-heurísticos basados en continuación considerando el error de generalización. Solamente en el caso de la base de datos WQ se observa un pequeño aumento estadísticamente significativo del error de generalización al emplear el algoritmo FA.

Adicionalmente, se observa una disminución de la desviación estándar del error de generalización de los algoritmos meta-heurísticos basados en continuación respecto a los algoritmos meta-heurísticos estándar. Esto indica un aumento de la estabilidad de la optimización. De igual forma se observa una disminución de la brecha entre el error de generalización y el error de entrenamiento al emplear los algoritmos meta-heurísticos basados en continuación. Este fenómeno indica un posible efecto regularizador permitiendo emplear modelos de RNA con mayor cantidad de parámetros en problemas con una cantidad relativamente menor de patrones de entrenamiento.

Análisis del tiempo de ejecución

Para la medición del tiempo de ejecución se tiene en cuenta la cantidad de segundos que demora el proceso de entrenamiento. Se considera el tiempo de ejecución secuencial medido en CPU y el tiempo de ejecución paralelo en GPU. Se considera el entrenamiento en estado de convergencia para los algoritmos meta-heurísticos cuando la cantidad de evaluaciones de la función objetivo

realizadas sobrepase cierto umbral η. En las pruebas realizadas se empleó un umbral $\eta = 40000$ evaluaciones de la función objetivo. Para las variantes basadas en métodos de continuación, cada subconjunto de patrones de entrenamiento se emplea durante η/k evaluaciones de la función objetivo para un total de η iteraciones.

Los hiper-parámetros empleados para los algoritmos pueden encontrarse en la Tabla(3.5). En los experimentos se utilizaron dos configuraciones de hardware, una para medir el tiempo de ejecución en CPU y otra para medir el tiempo de ejecución en GPU. A continuación, se especifican las principales características de cada configuración:

1. Configuración CPU: procesador Intel Core i3, velocidad del procesador 2.3 GHz, cantidad de memoria RAM 4 Gb.

2. Configuración GPU: procesador NVidia GeForce 920M, cantidad de núcleos[2] 384, cantidad de memoria gráfica 4 Gb.

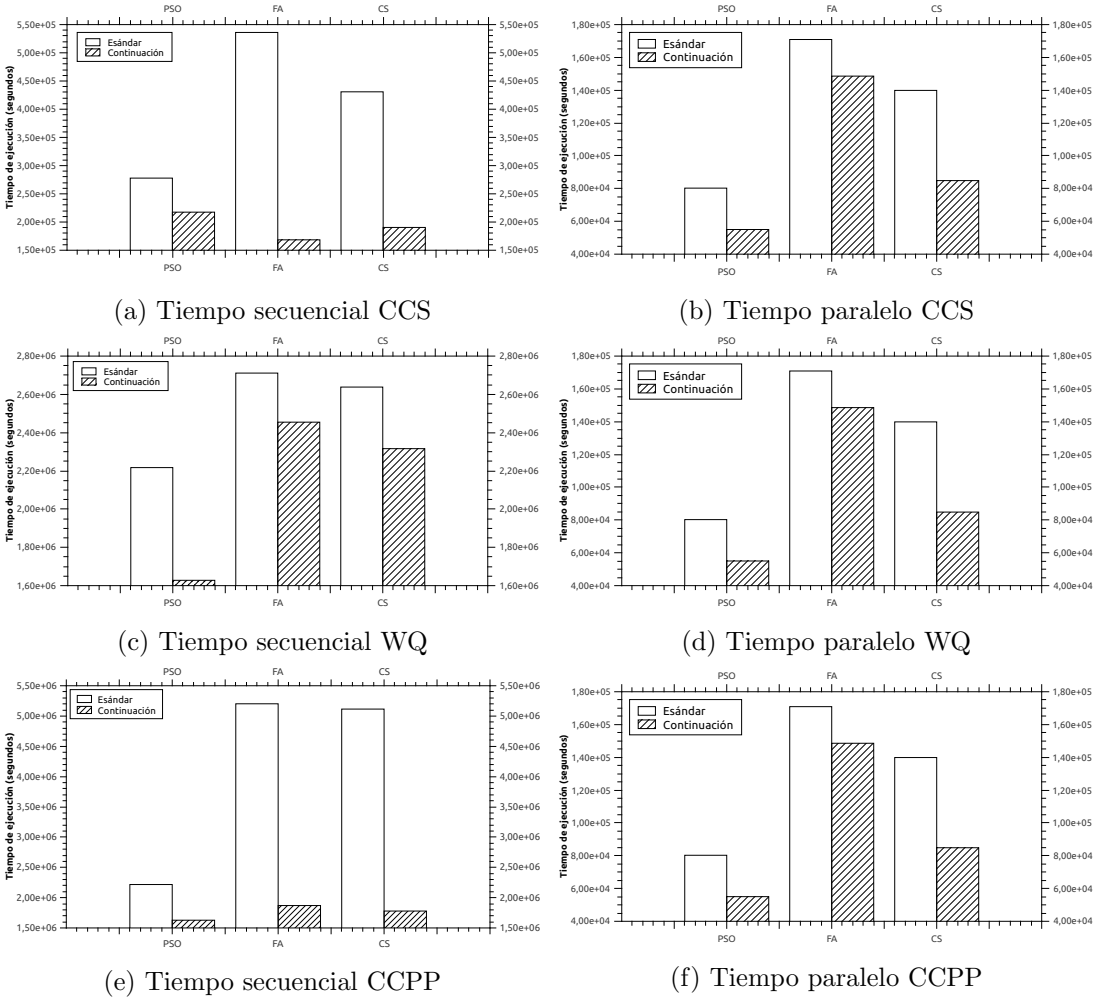
En la Figura(3.3) se muestra el consumo de tiempo secuencial y el consumo de tiempo paralelo para los algoritmos meta-heurísticos en barras transparentes y para los algoritmos meta-heurísticos basados en continuación en barras rayadas. Se tuvo en cuenta los algoritmos PSO, FA y CS así como las bases de datos CCS, WQ y CCPP para la obtención de los resultados. En cada caso se realizaron 50 mediciones de las cuales se reporta la media calculada.

En la Tabla(3.8) se se detallan los resultados de la Figura(3.3). Para cada algoritmo y problema de aprendizaje se especifica la media y la desviación estándar de la precisión obtenida en formato $\mu(\pm\sigma)$.

TABLA 3.8: Tiempo de ejecución en segundos de los algoritmos meta-heurísticos y sus variantes basadas en métodos de continuación.

	Función objetivo estándar			Continuación		
	PSO	FA	CS	PSO	FA	CS
	Tiempo de ejecución secuencial					
CCS	2.77E+5(2.14E+3)	5.36E+5(6.41E+3)	4.31E+5(5.31E+4)	2.17E+5(3.32E+3)	1.68E+5(1.73E+3)	1.90E+5(1.30E+4)
WQ	1.44E+6(2.85E+4)	2.71E+6(2.68E+4)	2.64E+6(4.80E+4)	7.09E+5(2.86E+3)	2.45E+6(2.01E+4)	2.32E+6(6.45E+4)
CCPP	2.21E+6(1.00E+4)	5.20E+6(5.88E+4)	5.11E+6(8.40E+4)	1.62E+6(9.33E+3)	1.87E+6(2.89E+4)	1.78E+6(5.77E+4)
	Tiempo de ejecución paralelo					
CCS	8.02E+4(1.42E+2)	1.70E+5(6.99E+3)	1.40E+5(1.93E+4)	5.50E+4(1.49E+3)	1.48E+5(2.74E+3)	8.49E+4(4.70E+3)
WQ	1.60E+5(4.26E+3)	3.77E+5(2.19E+4)	3.66E+5(4.90E+3)	8.50E+4(1.54E-4)	3.32E+5(1.91E+3)	3.13E+5(1.04E+4)
CCPP	1.55E+5(1.30E+3)	3.69E+5(3.43E+3)	3.74E+5(6.47E+3)	1.30E+5(8.00E+2)	2.17E+4(1.89E+3)	2.85E+5(1.04E+4)

[2]El término exacto en inglés sería *stream cores*.



(a) Tiempo secuencial CCS

(b) Tiempo paralelo CCS

(c) Tiempo secuencial WQ

(d) Tiempo paralelo WQ

(e) Tiempo secuencial CCPP

(f) Tiempo paralelo CCPP

Figura 3.3: Tiempo de ejecución de los algoritmos meta-heurísticos y sus variantes basada en métodos de continuación.

La estrategia de validación estadística consiste en comprobar la normalidad de las mediciones obtenidas mediante la prueba de Anderson-Darling. Posteriormente se comprueba si existe diferencia estadísticamente significativa entre las medias del tiempo de ejecución de los algoritmos meta-heurísticos y los algoritmos meta-heurísticos basados en continuación mediante la prueba de T-Student.

En la Tabla(3.9) se muestran los resultados de la prueba de T-Student en la comparación de las medias de los algoritmos algoritmos meta-heurísticos y sus variantes basadas en continuación. En negrita se destacan las mediciones donde se detectaron diferencias estadísticamente significativas.

TABLA 3.9: Resultados de la prueba de T-Student en la comparación de la media del tiempo de ejecución de los algoritmos meta-heurísticos y sus variantes basadas en continuación.

	CCS			WQ			CCPP		
	PSO	FA	CS	PSO	FA	CS	PSO	FA	CS
	Implementación secuencial								
p-valor	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**
t-valor	**15.24**	**55.29**	**4.40**	**25.37**	**7.65**	**4.19**	**42.93**	**50.82**	**32.67**
	Implementación paralela								
p-valor	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**	**<1.0E-4**
t-valor	**16.81**	**2.97**	**2.77**	**17.67**	**14.47**	**5.32**	**4.58**	**8.14**	**3.62**

A partir de los resultados obtenidos puede verse que los métodos basados en continuación presentan una disminución estadísticamente significativa del tiempo de ejecución tanto para la implementación secuencial como para la implementación paralela. La disminución del tiempo de ejecución guarda una estrecha relación con los hiper-parámetros relacionados con el método de continuación.

Adicionalmente se observa que la diferencia del tiempo de ejecución paralelo entre los algoritmos meta-heurísticos y sus variantes basadas en continuación es menor que en el caso de la implementación secuencial. Esto se debe al costo asociado a la copia de datos hacia el espacio de memoria de la GPU. Es de esperar que para problemas que involucren RNA de mayor cantidad de parámetros y problemas con mayor cantidad de patrones de entrenamiento esta diferencia sea mayor.

3.4. Conclusiones parciales

A partir del análisis realizado a los algoritmos estudiados en el presente capítulo es posible arribar a las siguientes conclusiones parciales:

1. Los algoritmos meta-heurísticos analizados permiten aumentar la precisión del entrenamiento de RNA respecto a los algoritmos basados en gradiente de manera estadísticamente significativa.

2. Los algoritmos meta-heurísticos con función objetivo estándar no presentan diferencias estadísticamente significativas respecto a sus variantes basados en métodos de continuación considerando la precisión del entrenamiento.

3. Los algoritmos meta-heurísticos basados en métodos de continuación disminuyen significativamente el tiempo de ejecución en CPU y GPU respecto a los algoritmos meta-heurísticos estándar.

Conclusiones generales

1. El entrenamiento de RNA es un problema de optimización NP-duro que presenta un grupo de cambios cuantitativos y cualitativos al considerar las RNP. Al considerar RNP los estudios relacionados con meta-heurísticas de optimización son limitados.

2. El modelo de paralelización multi-fase de los algoritmos meta-heurísticos provee un marco de trabajo eficiente para su implementación paralela disminuyendo el tiempo de ejecución del entrenamiento.

3. Los algoritmos basados en métodos de continuación para el cálculo de la función objetivo de las RNA, permiten disminuir el tiempo de ejecución del entrenamiento sin afectar la precisión.

4. Los algoritmos meta-heurísticos analizados permiten aumentar la precisión del entrenamiento de RNA respecto a SGD de manera estadísticamente significativa.

5. Los algoritmos meta-heurísticos estándar no presentan diferencias estadísticamente significativas respecto a sus variantes basadas en métodos de continuación al considerar la precisión del entrenamiento. Sin embargo, los algoritmos basados en continuación son más eficientes, constatándose una disminución significativa del tiempo de ejecución en CPU y en GPU.

Referencias bibliográficas

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[2] A. Griewank, "Who invented the reverse mode of differentiation?," *Documenta Mathematica, Extra Volume ISMP*, pp. 389–400, 2012.

[3] Y. Bengio, I. J. Goodfellow, and A. Courville, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[4] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3642–3649, IEEE, 2012.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] D. Yu and L. Deng, "Deep learning and its applications to signal and information processing [exploratory dsp]," *Signal Processing Magazine, IEEE*, vol. 28, no. 1, pp. 145–154, 2011.

[7] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

[8] A. Y. Hannun, A. L. Maas, D. Jurafsky, and A. Y. Ng, "First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns," *arXiv preprint arXiv:1408.2873*, 2014.

[9] Z. C. Lipton, "Stuck in a what? adventures in weight space," *arXiv preprint arXiv:1602.07320*, 2016.

[10] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 265–272, 2011.

[11] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

[12] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 571–582, 2014.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[14] L. Wang, W. Wu, J. Xiao, and Y. Yi, "Large scale artificial neural network training using multi-gpus," *arXiv preprint arXiv:1511.04348*, 2015.

[15] M. Mavrovouniotis and S. Yang, "Evolving neural networks using ant colony optimization with pheromone trail limits," in *Computational Intelligence (UKCI), 2013 13th UK Workshop on*, pp. 16–23, IEEE, 2013.

[16] S. S. D. Sahel and M. Boudour, "Application of particle swarm optimization based neural network to fault classification," in *2015 4th International Conference on Electrical Engineering (ICEE)*, pp. 1–4, IEEE, 2015.

[17] E. Valian, S. Mohanna, and S. Tavakoli, "Improved cuckoo search algorithm for feedforward neural network training," *International Journal of Artificial Intelligence & Applications*, vol. 2, no. 3, pp. 36–43, 2011.

[18] N. M. Nawi, A. Khan, and M. Rehman, "Data classification using metaheuristic cuckoo search technique for levenberg marquardt back propagation (cslm) algorithm," in *INTERNATIONAL CONFERENCE ON MATHEMATICS, ENGINEERING AND INDUSTRIAL APPLICATIONS 2014 (ICoMEIA 2014)*, vol. 1660, p. 050068, AIP Publishing, 2015.

[19] M. Sreeshakthy and J. Preethi, "Classification of human emotion from deap eeg signal using hybrid improved neural networks with cuckoo search," *BRAIN. Broad Research in Artificial Intelligence and Neuroscience*, vol. 6, no. 3-4, pp. 60–73, 2016.

[20] S. Nandy, P. P. Sarkar, and A. Das, "Analysis of a nature inspired firefly algorithm based back-propagation neural network training," *arXiv preprint arXiv:1206.5360*, 2012.

[21] J. Nayak, B. Naik, and H. Behera, "A novel nature inspired firefly algorithm with higher order neural network: Performance analysis," *Engineering Science and Technology, an International Journal*, 2015.

[22] N. M. Nawi, M. Rehman, and A. Khan, "Ws-bp: An efficient wolf search based back-propagation algorithm," in *INTERNATIONAL CONFERENCE ON MATHEMATICS, ENGINEERING AND INDUSTRIAL APPLICATIONS 2014 (ICoMEIA 2014)*, vol. 1660, p. 050027, AIP Publishing, 2015.

[23] A. Pandian, *Training neural networks with ant colony optimization.* PhD thesis, California State University, Sacramento, 2013.

[24] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks," in *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pp. 110–117, IEEE, 2003.

[25] H. Faris, I. Aljarah, and S. Mirjalili, "Training feedforward neural networks using multi-verse optimizer for binary classification problems," *Applied Intelligence*, pp. 1–11, 2016.

[26] J.-F. Chen, Q. H. Do, and H.-N. Hsieh, "Training artificial neural networks by a hybrid pso-cs algorithm," *Algorithms*, vol. 8, no. 2, pp. 292–308, 2015.

[27] T. M. Mitchell *et al.*, "Machine learning. wcb," 1997.

[28] J. L. McClelland, D. E. Rumelhart, P. R. Group, *et al.*, *Parallel distributed processing*, vol. 2. MIT press Cambridge, MA, 1987.

[29] D. T. Simon, K. C. Larsson, D. Nilsson, G. Burstrm, D. Galter, M. Berggren, and A. Richter-Dahlfors, "An organic electronic biomimetic neuron enables auto-regulated neuromodulation," *Biosensors and Bioelectronics*, vol. 71, pp. 359 – 364, 2015.

[30] J. A. Feldman and D. H. Ballard, "Connectionist models and their properties," *Cognitive science*, vol. 6, no. 3, pp. 205–254, 1982.

[31] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[32] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[33] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[34] R. P. Duin and E. Pekalska, "Open issues in pattern recognition," in *Computer Recognition Systems*, pp. 27–42, Springer, 2005.

[35] J. J. DiCarlo, N. Pinto, and D. D. Cox, "Why is real-world visual object recognition hard?," 2008.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[37] H. Chen, *FPGA Based Multi-core Architectures for Deep Learning Networks*. PhD thesis, University of Dayton, 2015.

[38] J. Park and W. Sung, "Fpga based implementation of deep neural networks using on-chip memory only," *arXiv preprint arXiv:1602.01616*, 2016.

[39] G. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE transactions on information theory*, vol. 14, no. 1, pp. 55–63, 1968.

[40] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, IEEE, 2009.

[42] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[43] M. Janzamin, H. Sedghi, and A. Anandkumar, "Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods," *CoRR abs / 1506.08473*, 2015.

[44] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points—online stochastic gradient for tensor decomposition," *arXiv preprint arXiv:1503.02101*, 2015.

[45] Y. N. 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," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2933–2941, Curran Associates, Inc., 2014.

[46] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, "Qualitatively characterizing neural network optimization problems," *arXiv preprint arXiv:1412.6544*, 2014.

[47] K.-L. Du and M. Swamy, *Search and optimization by metaheuristics: techniques and algorithms inspired by nature*. Birkhäuser, 2016.

[48] M. Gendreau and J.-Y. Potvin, *Handbook of metaheuristics*, vol. 2. Springer, 2010.

[49] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pp. 39–43, IEEE, 1995.

[50] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *International Symposium on Stochastic Algorithms*, pp. 169–178, Springer, 2009.

[51] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pp. 210–214, IEEE, 2009.

[52] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," tech. rep., Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[53] K. Chau, "Application of a pso-based neural network in analysis of outcomes of construction claims," *Automation in construction*, vol. 16, no. 5, pp. 642–646, 2007.

[54] D. J. Armaghani, M. Hajihassani, E. T. Mohamad, A. Marto, and S. Noorani, "Blasting-induced flyrock and ground vibration prediction through an expert artificial neural network based on particle swarm optimization," *Arabian Journal of Geosciences*, vol. 7, no. 12, pp. 5383–5396, 2014.

[55] M. Rashid, K. Kamal, T. Zafar, Z. Sheikh, A. Shah, and S. Mathavan, "Energy prediction of a combined cycle power plant using a particle swarm optimization trained feedforward neural network," in *Mechanical Engineering, Automation and Control Systems (MEACS), 2015 International Conference on*, pp. 1–5, IEEE, 2015.

[56] M. A. Ahmadi, "Neural network based unified particle swarm optimization for prediction of asphaltene precipitation," *Fluid Phase Equilibria*, vol. 314, pp. 46–51, 2012.

[57] K. Kuok, S. Harun, and S. Shamsuddin, "Particle swarm optimization feedforward neural network for modeling runoff," *International Journal of Environmental Science & Technology*, vol. 7, no. 1, pp. 67–78, 2010.

[58] S. Garg, K. Patra, and S. K. Pal, "Particle swarm optimization of a neural network model in a machining process," *Sadhana*, vol. 39, no. 3, pp. 533–548, 2014.

[59] J. Wang, K. Fang, W. Pang, and J. Sun, "Wind power interval prediction based on improved pso and bp neural network," *network*, vol. 1, p. 2, 2017.

[60] P. Sreeraj, T. Kannan, and S. Maji, "Pso-based neural network prediction and its utilization in gmaw process.," *Jordan Journal of Mechanical & Industrial Engineering*, vol. 8, no. 5, 2014.

[61] B. Gordan, D. J. Armaghani, M. Hajihassani, and M. Monjezi, "Prediction of seismic slope stability through combination of particle swarm optimization and neural network," *Engineering with Computers*, vol. 32, no. 1, pp. 85–97, 2016.

[62] J. Cao, H. Cui, H. Shi, and L. Jiao, "Big data: a parallel particle swarm optimization-back-propagation neural network algorithm based on mapreduce," *PloS one*, vol. 11, no. 6, p. e0157551, 2016.

[63] M. Carvalho and T. B. Ludermir, "Particle swarm optimization of neural network architectures andweights," in *Hybrid Intelligent Systems, 2007. HIS 2007. 7th International Conference on*, pp. 336–339, IEEE, 2007.

[64] A. Roy, D. Dutta, and K. Choudhury, "Training artificial neural network using particle swarm optimization algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 3, 2013.

[65] S. Mandal, G. Saha, and R. K. Pal, "Neural network training using firefly algorithm," *Global Journal on Advancement in Engineering and Science (GJAES)*, vol. 1, no. 1, 2015.

[66] I. Brajevic and M. Tuba, "Training feed-forward neural networks using firefly algorithm," in *Proceedings of the 12th International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED 13)*, pp. 156–161, 2013.

[67] M. Alweshah, "Firefly algorithm with artificial neural network for time series problems," *Research Journal of Applied Sciences, Engineering and Technology*, vol. 7, no. 19, pp. 3978–3982, 2014.

[68] Y. Gao, C. Qu, and K. Zhang, "A hybrid method based on singular spectrum analysis, firefly algorithm, and bp neural network for short-term wind speed forecasting," *Energies*, vol. 9, no. 10, p. 757, 2016.

[69] M. Sahoo, J. Nayak, S. Mohapatra, B. Nayak, and H. Behera, "Character recognition using firefly based back propagation neural network," in *Computational Intelligence in Data Mining-Volume 2*, pp. 151–164, Springer, 2015.

[70] M. Alweshah and S. Abdullah, "Hybridizing firefly algorithms with a probabilistic neural network for solving classification problems," *Applied Soft Computing*, vol. 35, pp. 513–524, 2015.

[71] C.-F. Chao and M.-H. Horng, "Firefly algorithm for training the radial basis function network in ultrasonic supraspinatus image classification," *Computer Modelling and New Techologies*, vol. 18, no. 3, pp. 77–83, 2014.

[72] N. M. Nawi, A. Khan, and M. Z. Rehman, "A new back-propagation neural network optimized with cuckoo search algorithm," in *International Conference on Computational Science and Its Applications*, pp. 413–426, Springer, 2013.

[73] H. Chiroma, S. Abdul-kareem, A. Khan, N. M. Nawi, A. Y. Gital, L. Shuib, A. I. Abubakar, M. Z. Rahman, and T. Herawan, "Global warming: predicting opec carbon dioxide emissions from petroleum consumption using neural network and hybrid cuckoo search algorithm," *PloS one*, vol. 10, no. 8, p. e0136140, 2015.

[74] H. Salimi, D. Giveki, M. A. Soltanshahi, and J. Hatami, "Extended mixture of mlp experts by hybrid of conjugate gradient method and modified cuckoo search," *International Journal of Artificial Intelligence & Applications*, vol. 3, no. 1, p. 1, 2012.

[75] J. Wang, Z. Sheng, B. Zhou, and S. Zhou, "Lightning potential forecast over nanjing with denoised sounding-derived indices based on ssa and cs-bp neural network," *Atmospheric Research*, vol. 137, pp. 245–256, 2014.

[76] R. A. Vazquez, "Training spiking neural models using cuckoo search algorithm," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pp. 679–686, IEEE, 2011.

[77] J. A. Bullinaria and K. AlYahya, "Artificial bee colony training of neural networks," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*, pp. 191–201, Springer, 2014.

[78] Y. Zhang, S. Wang, G. Ji, and P. Phillips, "Fruit classification using computer vision and feedforward neural network," *Journal of Food Engineering*, vol. 143, pp. 167–177, 2014.

[79] Y. Zhang, L. Wu, and S. Wang, "Magnetic resonance brain image classification by an improved artificial bee colony algorithm," *Progress In Electromagnetics Research*, vol. 116, p. 65?79, 2011.

[80] H. Shah, R. Ghazali, and N. M. Nawi, "Hybrid ant bee colony algorithm for volcano temperature prediction," in *International Multi Topic Conference*, p. 453?465, Springer, 2012.

[81] D. Pham, A. J. Soroka, A. Ghanbarzadeh, E. Koc, S. Otri, and M. Packianather, "Optimising neural networks for identification of wood defects using the bees algorithm," in *Industrial Informatics, 2006 IEEE International Conference on*, p. 1346?1351, IEEE, 2006.

[82] S. Nandy, P. P. Sarkar, and A. Das, "Training a feed-forward neural network with artificial bee colony based backpropagation method," *arXiv preprint arXiv:1209.2548*, 2012.

[83] H. Shah and R. Ghazali, "Prediction of earthquake magnitude by an improved abc-mlp," in *Developments in E-systems Engineering (DeSE), 2011*, pp. 312–317, IEEE, 2011.

[84] G. Rosa, J. Papa, K. Costa, L. Passos, C. Pereira, and X.-S. Yang, "Learning parameters in deep belief networks through firefly algorithm," in *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pp. 138–149, Springer, 2016.

[85] I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio, "Multi-prediction deep boltzmann machines," in *Advances in Neural Information Processing Systems*, pp. 548–556, 2013.

[86] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[87] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[88] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International Conference on Machine Learning*, pp. 115–123, 2013.

[89] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*, pp. 2113–2122, 2015.

[90] R. J. Tibshirani and R. Tibshirani, "A bias correction for the minimum error rate in cross-validation," *The Annals of Applied Statistics*, pp. 822–829, 2009.

[91] I. Tsamardinos, A. Rakhshani, and V. Lagani, "Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization," *International Journal on Artificial Intelligence Tools*, vol. 24, no. 05, p. 1540023, 2015.

[92] Y. Tan and K. Ding, "A survey on gpu-based implementation of swarm intelligence algorithms," *IEEE transactions on cybernetics*, vol. 46, no. 9, pp. 2028–2041, 2016.

[93] T. Brennich and S.-u. REGELUNGSTECHNIK, "Deep learning on a million core computing engine," 2015.

[94] B. Kågström, P. Ling, and C. Van Loan, "Gemm-based level 3 blas: High-performance model implementations and performance evaluation benchmark," *ACM Transactions on Mathematical Software (TOMS)*, vol. 24, no. 3, pp. 268–302, 1998.

[95] Y. Li, J. Dongarra, and S. Tomov, "A note on auto-tuning gemm for gpus," in *International Conference on Computational Science*, pp. 884–892, Springer, 2009.

[96] R. Nath, S. Tomov, and J. Dongarra, "An improved magma gemm for fermi graphics processing units," *The International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 511–515, 2010.

[97] X.-S. Yang and L. Press, "Nature-inspired metaheuristic algorithms second edition," 2010.

[98] H. Mobahi and J. W. Fisher III, "A theoretical analysis of optimization by gaussian continuation.," in *AAAI*, pp. 1205–1211, 2015.

[99] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the vapnik-chervonenkis dimension," *Journal of the ACM (JACM)*, vol. 36, no. 4, pp. 929–965, 1989.

[100] M. Kubat, *An Introduction to Machine Learning*. Springer, 2016.

[101] J. Rojas Delgado and R. Trujillo Rasúa, "Algoritmo meta-heurístico firefly aplicado al preentrenamiento de redes neuronales artificiales," *Revista Cubana de Ciencias Informáticas*, vol. 12, no. 1, pp. 14–27, 2018.

[102] I. S. Duff, M. A. Heroux, and R. Pozo, "An overview of the sparse basic linear algebra subprograms: The new standard from the blas technical forum," *ACM Transactions on Mathematical Software (TOMS)*, vol. 28, no. 2, pp. 239–267, 2002.

[103] N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo, "Intel avx: New frontiers in performance improvements and energy efficiency," *Intel white paper*, vol. 19, p. 20, 2008.

[104] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[105] N. Bell and J. Hoberock, "Thrust: A productivity-oriented library for cuda," in *GPU computing gems Jade edition*, pp. 359–371, Elsevier, 2011.

[106] C. Toolkit, "4.0 cublas library,?," *Nvidia Corporation*, vol. 2701, pp. 59–60, 2011.

[107] N. Nandapalan, R. P. Brent, L. M. Murray, and A. P. Rendell, "High-performance pseudo-random number generation on graphics processing units," in *International Conference on Parallel Processing and Applied Mathematics*, pp. 609–618, Springer, 2011.

[108] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration.," *LION*, vol. 5, pp. 507–523, 2011.

[109] M. Lichman, "UCI machine learning repository," 2013.

[110] D. Neeraja and G. Swaroop, "Prediction of compressive strength of concrete using artificial neural networks," *Research Journal of Pharmacy and Technology*, vol. 10, no. 1, pp. 35–40, 2017.

[111] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547–553, 2009.