

SEP
INSTITUTO TECNOLÓGICO DE CULIACÁN

TNM



ALGORITMO GENÉTICO COMPACTO DISTRIBUIDO PARA LA
OPTIMIZACIÓN DE REDES DE APRENDIZAJE PROFUNDO

TESIS

PRESENTADA ANTE EL DEPARTAMENTO ACADÉMICO DE ESTUDIOS DE POSGRADO
DEL INSTITUTO TECNOLÓGICO DE CULIACÁN EN CUMPLIMIENTO PARCIAL DE LOS
REQUISITOS PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

POR:

EMMANUEL FRANCISCO RAMÍREZ HERNÁNDEZ
INGENIERO EN SISTEMAS COMPUTACIONALES

DIRECTOR DE TESIS:
HÉCTOR RODRÍGUEZ RANGEL

CULIACÁN, SINALOA

6 de Septiembre del 2019

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Culiacán, Sin., 7 de Agosto del 2019

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

OFICIO: DEPI:336/VIII/2019

ASUNTO: **Autorización Impresión**

ING. EMMANUEL FRANCISCO RAMÍREZ HERNÁNDEZ
ESTUDIANTE DE LA MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN
PRESENTE.

Por medio de la presente y en virtud de que ha completado los requisitos para el examen de grado de la **Maestría en Ciencias de la Computación**, se concede autorización para la impresión de la tesis titulada: **ALGORITMO GENÉTICO COMPACTO DISTRIBUIDO PARA LA OPTIMIZACIÓN DE REDES DE APRENDIZAJE PROFUNDO** bajo la dirección del(a) **Dr. Héctor Rodríguez Rangel**

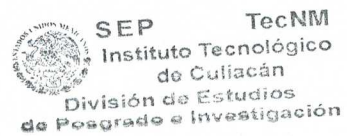
Sin otro particular reciba un cordial saludo.

ATENTAMENTE

Excelencia en Educación Tecnológica®



M.C. MARÍA ARACELY MARTÍNEZ AMAYA
JEFE(A) DE LA DIVISIÓN DE ESTUDIOS DE
POSGRADO E INVESTIGACIÓN



C.c.p. archivo

MAMA/lucy *

ALGORITMO GENÉTICO COMPACTO DISTRIBUIDO PARA LA OPTIMIZACIÓN DE REDES DE APRENDIZAJE PROFUNDO”

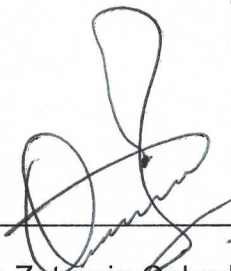
Tesis presentada por:

ING. EMMANUEL FRANCISCO RAMÍREZ HERNÁNDEZ

Aprobada en contenido y estilo por:



Dr. Héctor Rodríguez Rangel
Director de Tesis



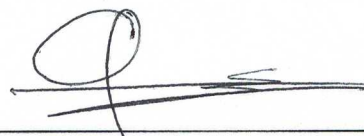
Dr. Ramón Zatarain Cabada
Secretario



Dra. María Lucía Barrón Estrada
Vocal -1



Dr. Víctor Alejandro González Huitrón
Vocal -2



M.C. María Aracely Martínez Amaya
Jefe(a) de la División de Estudios de
Posgrado e Investigación

Dedicatoria

Dedico este trabajo a mi madre Cleotilde Hernández Hernández por el apoyo incondicional que me ha brindado su apoyo en todos los proyectos que he emprendido. Además por enseñarme que para superarse debemos de ser persistentes y trabajar arduamente para lograr nuestros objetivos. Recuerda de donde venimos y visualiza a donde nos dirigimos.

Agradecimientos

Quiero agradecer al Instituto Tecnológico Nacional de México Campus Culiacán por haberme permitido realizar mis estudios en esta gran institución. También quisiera reconocer y agradecer al Consejo Nacional de Ciencia y Tecnología (CONACyT) por haber financiado mis estudios de maestría, además de darme la oportunidad de realizar una estancia de investigación a nivel internacional.

Agradezco a mi pareja que siempre me ha estado apoyando y motivado a seguir adelante.

A la Dra. Lucia Barron ya que he adquirido tanto conocimientos técnicos como sociales.

Al Dr. Héctor Rodríguez por el conocimiento compartido y apoyo brindado durante los dos años de maestría.

Al Dr. Ramón Zatarain que con su trayectoria nos ha enseñado que el trabajo arduo y constante es la clave del éxito.

Al Dr. Ricardo Quintero por su profesionalismo y su gusto por la docencia.

Declaración de Autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Emmanuel Francisco Ramírez Hernández. Culiacán, Sinaloa, México, 2019.

Resumen

Las tareas complejas y manuales para la optimización de hiperparámetros en redes de aprendizaje profundo han hecho que la comunidad de investigadores busque nuevas alternativas para encontrar mejores configuraciones de hiperparámetros, que funcione automáticamente y reduzca el proceso de tiempo de ejecución. El Algoritmo Genético compacto (AGc) es efectivo en la búsqueda de mejores soluciones a un problema y acelera el tiempo de búsqueda en comparación con un Algoritmo Genético (AG) común.

Esta tesis presenta una propuesta novedosa donde se combinan diversas estrategias para la reducción de tiempos en la búsqueda de hiperparámetros en redes de aprendizaje profundo las cuales son la distribución del AGc, la arquitectura propuesta para la distribución de los procesos del AGc y la paralelización de la ejecución de la función de aptitud con tarjetas GPU.

Además, se espera encontrar la forma de mejorar los tiempos de ejecución, al distribuir el proceso del AGc y acelerar la evaluación del individuo que utiliza GPU. Se utilizó el modelo maestro-esclavo, este modelo a lo largo de los años ha mostrado buenos resultados para los AG distribuidos, de hecho, permitirá usar Unidades de Procesamiento Gráfico (GPU) en cada esclavo y con esto reducir aún más el tiempo de ejecución de AGc para la optimización de hiperparámetros en redes de aprendizaje profundo. Cada esclavo está utilizando una GPU para minimizar el tiempo de consumo en la evaluación de la función de aptitud de los AGc.

Finalmente, los resultados muestran que la implementación de la arquitectura brinda escalabilidad, tolerancia a fallas y una configuración más sencilla del nodo esclavo (si necesita usar más nodos esclavos). Además, el AGc puede obtener resultados más rápido sin perder precisión.

Palabras Clave

- Optimización
- Redes de Aprendizaje Profundo
- Hiperparámetros
- Algoritmo Genético
- Sistemas Distribuidos
- Pase de mensajes
- Algoritmo Genético Distribuido

Índice general

Índice de figuras	X
Índice de tablas	XI
1. Introducción	1
1.1. Definición del Problema	3
1.2. Hipótesis	4
1.3. Objetivo	4
1.4. Objetivos Específicos	4
1.5. Justificación	5
1.6. Estructura de la Tesis	5
2. Marco teórico	7
2.1. Inteligencia Artificial	7
2.1.1. Aprendizaje Máquina	8
2.1.2. Redes Neuronales Artificiales	9
2.1.3. Aprendizaje Profundo	10
2.1.4. Redes Neuronales Convolucionales	11
2.1.4.1. Capas de entrada	12
2.1.4.2. Capas de extracción de características	13
2.1.4.3. Capa de clasificación	14
2.1.5. Hiperparámetros	14
2.1.6. Clasificación	15
2.1.7. Regresión	16
2.2. Algoritmos evolutivos	17
2.2.1. Algoritmo Genético	18
2.2.2. Algoritmo Genético Compacto	18
2.3. Computación Distribuida	20
2.3.1. Concurrencia	21
2.3.2. Semáforos	22
2.3.3. Algoritmos Genéticos Distribuidos	22
2.3.4. Modelo Maestro-Eslavo	23
2.3.5. AMQP: Advanced Messaging Queuing Protocol	23

3. Estado del Arte	25
3.1. Metodos Tradicionales de Búsqueda	25
3.2. Métodos evolutivos	26
3.3. Distribución y Paralelismo	26
3.4. Tabla Comparativa	29
4. Diseño del Experimento	30
4.1. Pre-procesamiento de datos.	31
4.1.1. Datos atípicos	31
4.1.2. Normalización	32
4.2. Definición de modelo.	33
4.3. Optimización.	34
4.3.1. Arquitectura propuesta.	35
4.3.2. Inicialización del Componente Maestro	37
4.3.3. Inicialización del Componente Esclavo	39
4.3.4. Generación de los individuos	40
4.3.5. Evaluación de los Individuos	42
4.3.6. Competencia de los Individuos	43
4.4. Tecnologías Utilizadas	45
4.4.1. Python	46
4.4.2. Keras	46
4.4.3. Tensorflow	47
4.4.4. RabbitMQ	47
5. Pruebas y Análisis de Resultados	49
5.1. GPU vs CPU	49
5.2. Algoritmo Genético Compacto Distribuido	51
5.2.1. Un Componente Maestro, un Componente Esclavo con una GPU por esclavo	52
5.2.2. Un Componente Maestro, dos Componentes Esclavos con una GPU por esclavo	54
5.2.3. Un Componente Maestro, un Componente Esclavo con 2 GPU por esclavo	56
5.2.4. Un Componente Maestro, dos Componentes Esclavos con 2 GPU por Esclavo	58
6. Conclusiones	62
6.1. Conclusiones	62
6.2. Contribuciones	63
6.3. Consideraciones	63
6.4. Trabajo Futuro	63

Índice de figuras

2.1. Neuronas artificiales básicas (Anderson and McNeill, 1992).	9
2.2. Arquitectura de una RNA con 4 neuronas en la capa de entrada, 5 neuronas en la capas ocultas y una sola neurona de salida.	10
2.3. Arquitectura general de una RNC.	12
2.4. Dimensiones de la capa de entrada.	12
2.5. Ejemplo del proceso de una capa de convolución.	13
2.6. Ejemplo de las diferentes capas para la obtención de una clasificación.	14
2.7. Representación de la base de datos MNIST (Deng, 2012).	16
2.8. Representación de un diagrama de dispersión (Patterson and Gibson, 2017) .	17
2.9. Representación del cromosoma (Sivanandam and Deepa, 2008).	18
2.10. Sistema distribuido conectando procesadores a través de una red de comunicación(Kshemkalyani and Singhal, 2008).	20
2.11. Ejemplo de arquitectura de un sistema distribuido (Kshemkalyani and Singhal, 2008).	21
2.12. Ejemplo de solución de una operación aritmética usando procesos concurrentes.	22
2.13. Representación del Modelo Maestro-Esclavo.	24
4.1. Diagrama de flujo del desarrollo del experimento.	30
4.2. Dato atípico en un conjunto de observaciones.	32
4.3. Serie de tiempo normalizada en un rango de valores entre [0, 1].	33
4.4. Topología seleccionada de la CNN.	34
4.5. Diagrama para la generación de un modelo óptimo.	34
4.6. Arquitectura general.	35
4.7. Diagrama de la arquitectura general.	36
4.8. Componente Maestro se suscribe al intermediario de mensajes de Rabbit MQ.	37
4.9. Codificación de los diferentes hiperparámetros en el cromosoma del AGc. . .	39
4.10. Componente Esclavo se suscribe al intermediario de mensajes de Rabbit MQ.	40
4.11. Representación del trabajo en conjunto entre los hilos Componente Maestro. .	44
4.12. Representación de flujo de trabajo de RabbitMQ.	48
5.1. Gráfica comparativa entre el uso las diferentes tarjetas GPU y el uso de CPU.	50
5.2. Gráfica comparativa del rendimiento de las diferentes tarjetas GPU.	51
5.3. Representación de la configuración de los experimentos realizados.	52

Índice de tablas

3.1.	Hiperparámetros utilizados en el algoritmo evolutivo en el trabajo de (Castillo et al., 2011).	27
3.2.	Comparativa de los diferentes trabajos que realizan la búsqueda del conjunto óptimo de hiperparámetros y acelerar el tiempo de búsqueda.	29
4.1.	Hiperparámetros generales y sus posibles valores.	39
4.2.	Hiperparámetros convolucionales y sus posibles valores.	39
5.1.	Resultados de los experimentos con configuración de un Componente Maestro y un Componente Esclavo.	53
5.2.	Relación del cromosoma y los valores de cada hiperparámetro general para los experimentos con un Componente Maestro y un Componente Esclavo.	53
5.3.	Relación del cromosoma y los valores de cada hiperparámetro convolucional para los experimentos con un Componente Maestro y dos Componentes Esclavos.	54
5.4.	Resultados de los experimentos con configuración de un Componente Maestro y dos Componentes Esclavos.	55
5.5.	Relación del cromosoma y los valores de cada hiperparámetros generales para los experimentos con un Componente Maestro y dos Componentes Esclavos.	55
5.6.	Relación del cromosoma y los valores de cada hiperparámetros convolucionales para los experimentos con un Componente Maestro y dos Componentes Esclavos.	56
5.7.	Resultados obtenidos en experimentos con un Componente Maestro y un Componente Esclavo.	57
5.8.	Relación del cromosoma y los valores de cada hiperparámetros generales para los experimentos con un Componente Maestro y dos Componentes Esclavos.	58
5.9.	Relación del cromosoma y los valores de cada hiperparámetros convolucionales para los experimentos con un Componente Maestro y dos Componentes Esclavos.	58
5.10.	Resultados obtenidos en experimentos con un Componente Maestro y dos Componentes Esclavos con dos GPU's por esclavo.	59
5.11.	Relación del cromosoma y los valores de cada hiperparámetros generales para los experimentos con un Componente Maestro y dos Componentes Esclavos.	60

5.12. Relación del cromosoma y los valores de cada hiperparámetros convolucionales para los experimentos con un Componente Maestro y dos Componentes Esclavos.	60
5.13. Comparativa de promedios de los experimentos realizados.	61

Capítulo 1

Introducción

El campo de la Inteligencia Artificial (IA) es demasiado extenso y ha existido durante bastante tiempo. El aprendizaje profundo es un campo de estudio dentro de aprendizaje máquina, a su vez el aprendizaje máquina es un campo de la inteligencia artificial (Patterson and Gibson, 2017).

En la actualidad las redes de aprendizaje profundo han revolucionando el ámbito de inteligencia Artificial (IA), debido a los buenos resultados generados y al gran número de áreas en las que se han implementado. Las redes de aprendizaje profundo han resurgido debido a que la capacidad de cómputo se ha incrementado, en consecuencia ahora es posible implementar estos modelos que son costosos en tiempo y procesamiento.

Las redes de aprendizaje profundo pueden tener millones de parámetros que pueden ser ajustados para obtener una solución eficaz. Además, es necesario utilizar bases de datos que contengan la mayor cantidad de instancias posibles para generar mayor conocimiento del modelo al momento de realizar el entrenamiento. Esto conduce a una problemática preponderante, ¿Qué sucede si la topología de la red de aprendizaje profundo cambia?, ¿Qué sucede si la base de datos con la que se realiza el entrenamiento es de mayor tamaño?. Si se incrementa el volumen de la red de aprendizaje profundo y la base de datos de entrenamiento aumenta podemos obtener mejores resultados pero a costa de un tiempo computacional increíblemente alto (Hegde and Usmani. 2016).

En la literatura (Patterson and Gibson, 2017) se define a las redes de aprendizaje profundo como redes neuronales con un número de parámetros y capas demasiado extensas en cualquiera de sus cuatro posibles arquitecturas de la red:

1. Redes pre-entrenadas no supervisadas.
2. Redes neuronales convolucionales (RNC).
3. Redes neuronales recurrentes.
4. Redes neuronales recursivas.

Las redes de aprendizaje profundo cuentan con parámetros del modelo y parámetros ajustables, estos últimos ocasionan que las redes se entrenen mejor y más rápido. Estos parámetros de ajuste se denominan *hiperparámetros* y se ocupan del control de las funciones de optimización y la selección de modelos durante el entrenamiento del algoritmo de aprendizaje máquina (Patterson and Gibson, 2017).

En la fase del entrenamiento para que el modelo pueda presentar buenos resultados, es necesario contar con amplios conjuntos de datos. Por lo tanto, para el proceso de entrenamiento de una red de aprendizaje profundo con volúmenes de datos inmensos se requieren equipos especializados de cómputo para poder reducir los tiempos de procesamiento del entrenamiento. Los equipos especializados necesarios para realizar los entrenamientos de redes de aprendizaje profundo son costosos, y no son accesibles para la mayoría de instituciones del país. Tomando demasiado tiempo al momento de realizar pruebas o experimentos de su modelos, aumentando el tiempo de investigación.

El uso de redes de aprendizaje profundo involucra un proceso cuidadoso el cual es conocido como ajuste hiperparámetros. Dichos ajustes son considerados como “un arte oscuro”(Snoek et al., 2012) que requiere experiencia de un especialista, para seguir un conjunto de recomendaciones generales o en ocasiones de fuerza bruta.

Una de las grandes soluciones para procesos extensos y demandantes es el *cómputo distribuido*. El cómputo distribuido ha sido utilizado ampliamente para aplicaciones tanto científicas y comerciales a gran escala. Estas aplicaciones realizan procesos que conllevan una gran cantidad de operaciones con una gran cantidad de datos. Las operaciones son realizadas en diferentes dispositivos que pueden compartir un espacio físico o no, aprovechando la capacidad de cómputo de los dispositivos.

En este trabajo se hace énfasis en la búsqueda de mejores resultados, al realizar el entrenamiento de las RNC y en reducir los tiempos de ejecución de dicha búsqueda. Para esto, se

propone una estrategia evolutiva distribuida para la optimización de la red neuronal convolucional. Existen varias estrategias evolutivas pero en este trabajo utilizaremos el algoritmo genético compacto (AGc) para la búsqueda estocástica de los mejores hiper-párametros de la red convolucional. El AGc generará individuos, los cuales son representaciones binarias de las configuraciones de la red convolucional. Cada configuración se ejecutará en un equipo diferente, dichos equipos contarán con n unidades de procesamiento (GPU) buscando reducir tiempos del entrenamiento de la red convolucional.

Existen diferentes problemas a resolver con las RNC pero los problemas que se centrará este trabajo son los siguientes:

1. Clasificación.
2. Regresión.

En el caso de los problemas de clasificación tomaremos como función de aptitud del algoritmo genético compacto distribuido (AGcD) la precisión obtenida en la ejecución del modelo y para los problemas de regresión tomaremos como función de aptitud el error.

1.1. Definición del Problema

El aumento del uso de las redes de aprendizaje profundo ha generado una alta demanda de modelos de redes de aprendizaje profundo, por lo tanto la necesidad de mejores y más rápidas soluciones han aumentado.

Actualmente la búsqueda de un modelo óptimo se lleva a cabo de una forma manual y “artesanal” ajustando los hiperparámetros hasta encontrar un mejor modelo. El tiempo invertido en el ajuste de hiperparámetros, entrenamiento y validación del modelo llegan a ser considerablemente altos sin conocer si dicho modelo otorgará el resultado esperado. Es necesario encontrar una forma de automatizar la búsqueda de dichos modelos. Además de una automatización es necesario siempre buscar el modelo con mejor resultado si es posible que la solución haya convergido.

La red neuronal convolucional cuenta con una gran cantidad de hiperparámetros, por lo que, debido al ajuste de estos, la ejecución del entrenamiento de cada uno de los posibles

modelos encontrados y la búsqueda automática requieren una gran capacidad de cómputo.

En esta sección se puede concluir que la forma manual en la que se realiza el ajuste de hiperparámetros, el tiempo de búsqueda para un modelo óptimo y la gran cantidad de cómputo son las problemáticas a resolver en este trabajo.

1.2. Hipótesis

En la búsqueda de hiperparámetros en modelos de aprendizaje profundo se obtendrán una red óptima en un menor tiempo mediante el uso del AGc Distribuido (AGcD) en comparación con el AGc sin afectar la calidad del modelo.

1.3. Objetivo

Modificar el AGc para que funcione en un ambiente paralelo/distribuido capaz de entrenar una red convolucional óptima reduciendo los tiempos de entrenamiento comparado con el AGc tradicional.

1.4. Objetivos Específicos

1. Explorar las diferentes formas optimización de los hiperparámetros de una RNC.
2. Distribuir el AGc para explorar un mayor número de configuraciones de forma automática una RNC.
3. Paralelizar el proceso de entrenamiento en diferentes tarjetas GPU's .
4. Optimizar de los hiper-parámetros de la red convolucional utilizando un AGc.
5. Diseñar una arquitectura que nos permitá realizar la distribución de los procesos del AGc.

1.5. Justificación

En la actualidad los tiempos en el entrenamiento de una red convolucional dependen del tamaño del conjunto de datos de entrenamiento y la configuración de hiperparámetros asignados. Si el conjunto de datos para el entrenamiento es demasiado extenso (para tener un modelo de calidad se requiere que el conjunto de datos sea extenso y balanceado) costará un mayor tiempo en finalizar el entrenamiento. Una vez finalizado el entrenamiento no se cuenta con la certeza que el modelo entrenado sea el resultado que estamos esperando. En ese caso, se tiene que realizar un ajuste de hiperparámetros, volver a entrenar y esperar para obtener un resultado diferente y que dicho resultado sea el que estamos esperando. Todas estas tareas hacen muy extenso el trabajo del investigador o científico de datos para encontrar un modelo que le otorgue buenos resultados y sin saber si el ajuste de hiperparámetros le otorgará buenos resultados.

La propuesta mejorará los tiempos de ejecución de las búsquedas de mejores modelos de RNC, permitiendo explorar un mayor número de configuraciones evaluadas, por lo tanto una posible mejor solución para el modelo.

El AGcD podrá ser utilizado en la creación de cualquier clasificador o pronosticador y permitirá explorar un mayor número de posibles soluciones, seleccionando la mejor en un tiempo más corto. Además, los costos de los equipos que utilizan GPU son más accesibles para las instituciones públicas que servidores dedicados por lo tanto se otorgará la oportunidad a estas instituciones públicas el realizar su experimentación a bajo costo.

Por último con la aplicación del AGcD automatizará la búsqueda de los mejores modelos, se reducirán los tiempos de búsqueda sin requerir un poder de cómputo descomunal para encontrar una solución óptima.

1.6. Estructura de la Tesis

Este trabajo está dividido en 6 capítulos y se encuentra organizado de la siguiente manera: En el Capítulo 2 se describe la teoría y conceptos fundamentales que son necesarios comprender para el entendimiento de este trabajo. En el Capítulo 3 se presenta el estado

del arte de la literatura actual relacionada con este trabajo. En el Capítulo 4 se muestra la descripción detallada de cada uno de los pasos del AGcD. En el Capítulo 5 se presentan las pruebas realizadas y los resultados obtenidos. Para finalizar las conclusiones presentadas en el Capítulo 6.

Capítulo 2

Marco teórico

En este capítulo se abordarán conceptos utilizados en el desarrollo de este trabajo. Esto con la intención de sumergir al lector en los conceptos utilizados a lo largo de este trabajo.

Se partirá de conceptos básicos relacionados a este documento tales como: regresión, clasificación, inteligencia artificial y algoritmos de aprendizaje máquina para otorgar una visión general sobre estos temas. También se profundizará en los algoritmos de aprendizaje profundo haciendo énfasis en las RNC. Además, se hablará sobre los algoritmos de optimización que fueron implementados para la selección de los hiperparámetros. Para finalizar se introducirá en los conceptos utilizados en la implementación tecnológica utilizada para la distribución de la evaluación de cada uno de los individuos.

2.1. Inteligencia Artificial

A lo largo de los años se ha discutido el significado de *inteligencia artificial*, existiendo múltiples definiciones de diferentes investigadores en diversas áreas. Con Base en la obra “*Artificial intelligence: a modern approach*” escrita por Russell and Norvig, donde se define inteligencia artificial como el campo de estudio de como el humano piensa, con este entendimiento ser capaces de que las máquinas cuenten con la capacidad de pensar, razonar y resolver problemas de forma autónoma.

2.1.1. Aprendizaje Máquina

El aprendizaje máquina actualmente se encuentra en muchos aspectos de la sociedad, desde buscadores web para filtrar contenido así como recomendaciones en páginas de comercio electrónico (e-commerce), también está presente en productos tales como cámaras y teléfonos inteligentes. Los sistemas con Aprendizaje Máquina (Machine Learning) son usados para identificar objetos en imágenes, pasar el habla a texto y seleccionar resultados relevantes en búsquedas (LeCun et al., 2015). Además, el aprendizaje máquina permite dotar a un equipo de cómputo de la habilidad de aprender y mejorarse conforme a ejemplos que se le proporcione sin la necesidad de realizar una programación explícita de este conocimiento. El proceso de aprendizaje consta de dar a la máquina una serie de ejemplos o datos históricos para que esta pueda detectar patrones en los datos que le permita tomar una mejor decisión con los nuevos datos. Existen diferentes tipos de aprendizaje máquina como lo son:

1. **Aprendizaje Supervisado:** En el aprendizaje supervisado se tiene una serie de observaciones asociadas a un valor esperado, en otras palabras se le dice a la máquina que es lo que debe aprender.
2. **Aprendizaje No supervisado:** Al contrario del aprendizaje supervisado, en el aprendizaje no supervisado no hay valor esperado asociado. Se realiza una búsqueda de cuáles son los grupos que mejor se apeguen a las características deseadas, en otras palabras se realiza una agrupación y no una clasificación (Harrington, 2012).

En términos generales, un sistema automático debe ser capaz de proporcionar una respuesta apropiada cuando se introduce información al mismo. En el aprendizaje supervisado, el núcleo del sistema es un modelo predictivo que asigna uno o varios valores de salida a cada elemento de entrada, con base en el conocimiento adquirido a partir de un conjunto de datos cuyas salidas son conocidas. Sin embargo, la información disponible no siempre se puede o se debe utilizar directamente para alimentar el modelo. Por ello, suele ser necesaria una fase denominada de pre-procesamiento para adecuar las entradas en aras de maximizar el rendimiento del modelo. Asimismo, la salida que ofrece el modelo puede no ser satisfactoria o quizás resulte difícil de interpretar. Por lo tanto, en una última fase de post-procesamiento, se

evalúa el rendimiento del modelo y, en la medida de lo posible, se adaptan las salidas a las necesidades del usuario (Sampedro and García, 2012).

2.1.2. Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNA) son metodologías computacionales inspiradas en una red de neuronas biológica. Estas pueden contener capas de nodos que realizan operaciones simples; los nodos están altamente interconectados y cada conexión cuenta con un peso, el cual es ajustado cuando los datos son presentados a la red mediante un proceso de entrenamiento. Un entrenamiento correcto puede resultar en un modelo de RNA que pueda realizar tareas como predecir un valor de salida, clasificar un objeto, aproximar una función y completar un patrón conocido, entre otros (LeCun et al., 2015).

La unidad de procesamiento fundamental en una RNA es la neurona. Básicamente una neurona biológica recibe entradas de otras fuentes, las combina y realiza generalmente operaciones no lineales y emite un resultado final (Anderson and McNeill, 1992). La Figura 2.1 muestra la representación de una neurona artificial. En la cual cada entrada a la red es representada por x_i . Cada una de las entradas es multiplicada por un peso, representados por w_{ij} . En el caso más simple, este producto es simplemente sumado. El resultado de la sumatoria es pasado a una función de activación o transferencia para generar un resultado y/o una salida.

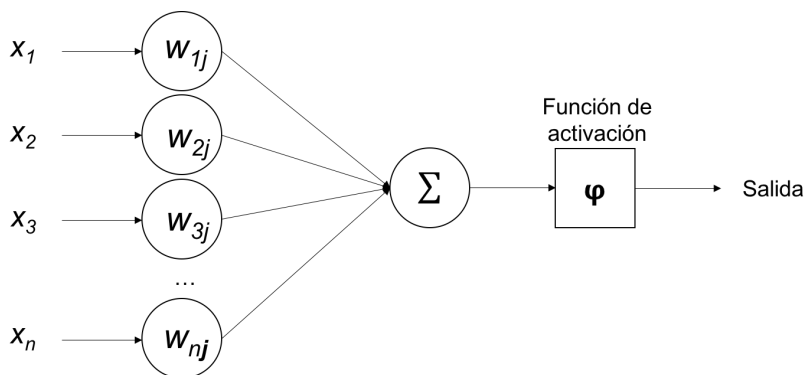


Figura 2.1: Neurona artificial básica (Anderson and McNeill, 1992).

En la Figura 2.2 se muestra la arquitectura clásica de una red neuronal, la cual consta de 3 capas (capa de entrada, capa oculta y capa de salida). En la capa de entrada no se realiza cómputo alguno, esta solo sirve como entrada a la red neuronal. En la capa oculta es donde

se realiza el mayor cálculo y la capa de salida provee la salida de la red. Una RNA como un aproximador universal puede aprender cualquier función dadas suficientes neuronas (Haykin, 1994).

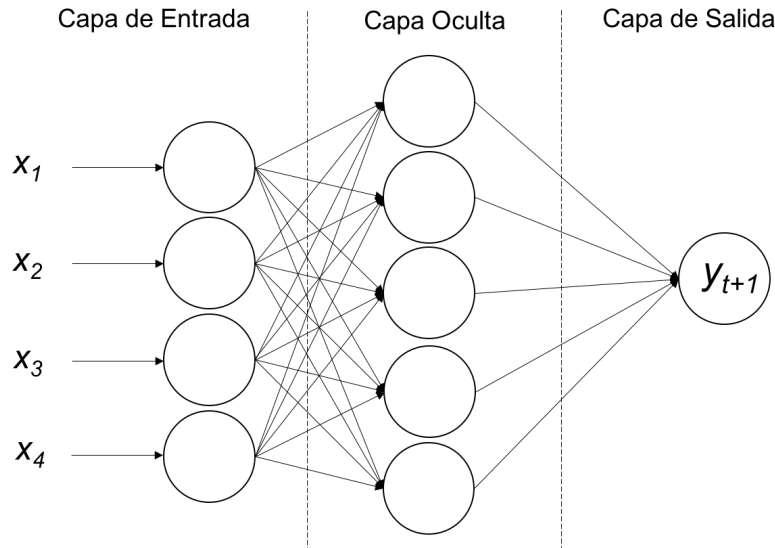


Figura 2.2: Arquitectura de una RNA con 4 neuronas en la capa de entrada, 5 neuronas en la capas ocultas y una sola neurona de salida.

2.1.3. Aprendizaje Profundo

Según la literatura, las redes de aprendizaje profundo son RNA con una gran cantidad de capas e hiper-parámetros (Patterson and Gibson, 2017). Entre las principales diferencias entre redes de aprendizaje profundo y una red neuronal multi-capas son las siguientes:

1. Un mayor número de neuronas.
2. Formas más complejas de conectar capas.
3. El aumento del poder computacional para el entrenamiento.
4. Extracción de características de manera automática.

Una de las grandes ventajas de las redes de aprendizaje profundo es la extracción de características de manera automatizada. Esto significa que la red decide que características de un conjunto de datos, se pueden utilizar como indicadores para etiquetar esos datos de manera confiable.

En este trabajo se enfoca en las RNC, las cuales son utilizadas para diversas tareas como:

1. Vision artificial.
2. Realizar pronósticos.
3. Clasificar objetos.
4. Extracción de características

2.1.4. Redes Neuronales Convolucionales

El objetivo principal de las RNC es aprender características de orden superior que se encuentran en los datos a través de convoluciones. La eficacia de las RNC en el reconocimiento de imágenes es una de las principales razones por las cuales el mundo reconoce el poder del aprendizaje profundo (Patterson and Gibson, 2017).

Las RNC extraen la información mediante convoluciones y estas características sirven de entrada a la capa de clasificación. En términos generales, se puede decir que las RNC básicamente están conformadas por tres grupos de capas:

1. Capa de entrada.
2. Capa de extracción de características (Convolución).
3. Capa de clasificación (aprendizaje).

A continuación se puede observar en la Figura 2.3 se muestra la arquitectura a nivel general de una RNC, donde podemos encontrar los grupos de capas mencionados anteriormente.

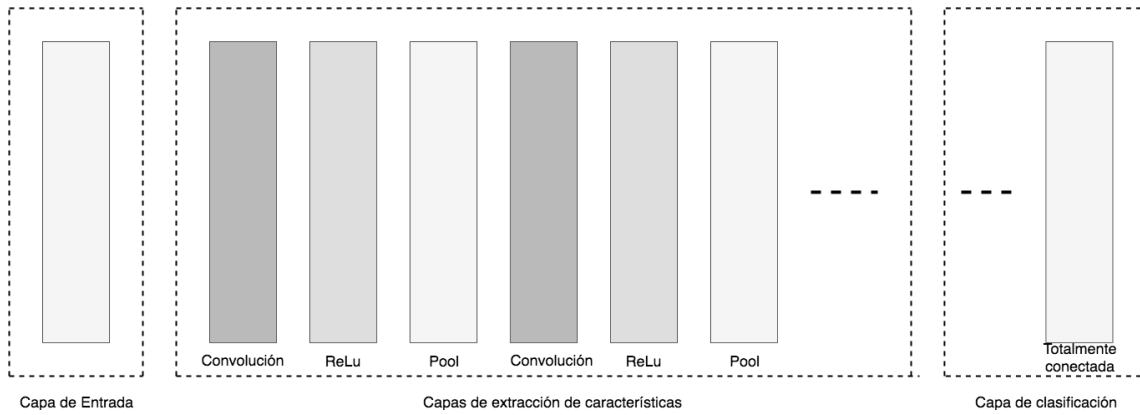


Figura 2.3: Arquitectura general de una RNC.

2.1.4.1. Capas de entrada

La capa de entrada es donde ingresan los datos crudos de una imagen para ser procesados en la red. Normalmente los datos de entrada especifican sus dimensiones, como el ancho y alto. Cuando la imagen se encuentra a color se maneja una dimensión adicional denominada canal. Típicamente el número de canales son tres, para el valor de RGB de cada píxel. En la Figura 2.4 se puede observar las dimensiones de la capa de entrada de una RNC.

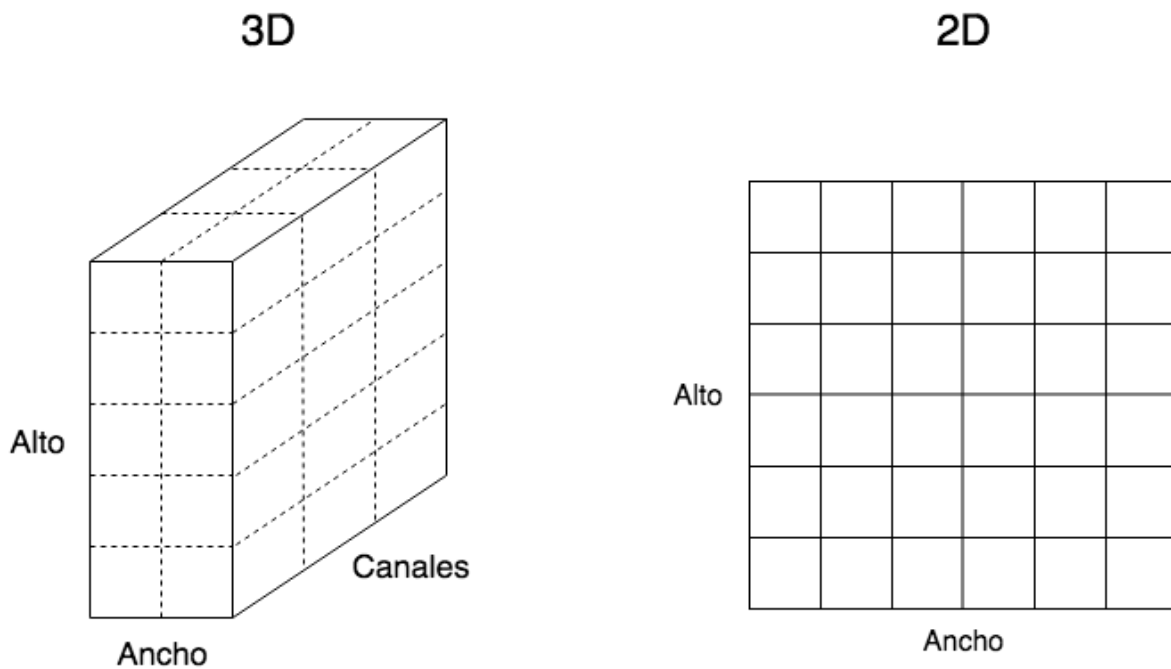


Figura 2.4: Dimensiones de la capa de entrada.

2.1.4.2. Capas de extracción de características

La capa de extracción de características tiene un patrón general repetitivo de la siguiente secuencia:

1. Capa de convolución.
2. Capa de pooling (Agrupamiento).

Las **capas de convolución** son consideradas el núcleo más importante en la construcción de la arquitectura para una RNC también conocidas como detector de características (Patterson and Gibson, 2017). Estas capas transforman los datos de entrada (datos crudos o mapa de características resultado de una convolución) usando un parche de neuronas conectado localmente desde una capa anterior. Esta capa calculará el producto punto entre la región de las neuronas de la capa de entrada y los pesos a los que se encuentran conectadas localmente en la capa de salida.

La salida resultante generalmente contará con las mismas dimensiones espaciales (o dimensiones espaciales más pequeñas) pero a veces aumenta el número de elementos en la tercera dimensión de la salida (dimensión de profundidad). La Figura 2.5 muestra como el filtro se va deslizando a través de los datos de entrada y como va calculando el producto punto entre el filtro y los datos de entrada en los que se superpone el filtro.

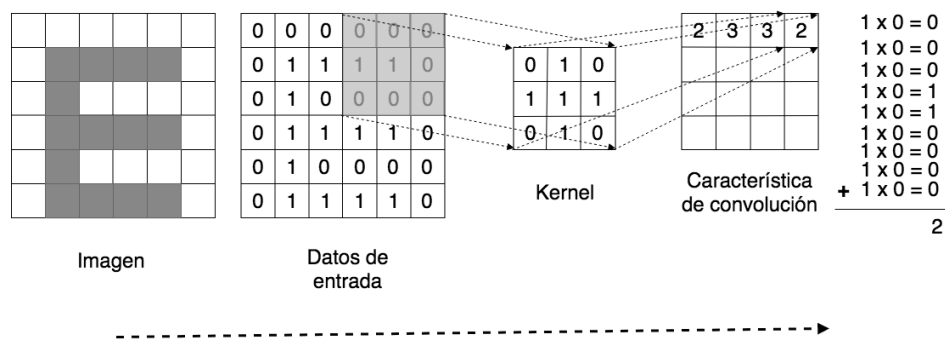


Figura 2.5: Ejemplo del proceso de una capa de convolución.

Las **capas de pooling (Agrupamiento)** encuentran una serie de características en las imágenes y construyen progresivamente características de un orden superior.

Estas capas se insertan comúnmente entre capas convolucionales sucesivas. Se busca seguir capas agrupadas para reducir progresivamente el tamaño espacial (ancho y alto) de la representación de datos. La agrupación de capas reduce la representación de datos progresivamente en la red y ayuda a controlar el sobre ajuste.

2.1.4.3. Capa de clasificación

Estas capas son en las que se encuentran una o mas capas completamente conectadas para tomar las características de orden superior y producir probabilidades o puntajes de clase. Estas capas están completamente conectadas a todas la neuronas de la capa anterior. La salida de estas capas produce normalmente una salida bidimensional de las dimensiones $[b \times N]$, donde b es el número de ejemplos en el mini lote y N es el número en calificar.

La Figura 2.6 muestra las diferentes capas trabajando en conjunto para realizar una clasificación desde la capa de entrada, pasando por un conjunto de capas de extracción de características (convolución y agrupamiento) y por último pasando por la capa de clasificación y otorgando un resultado.

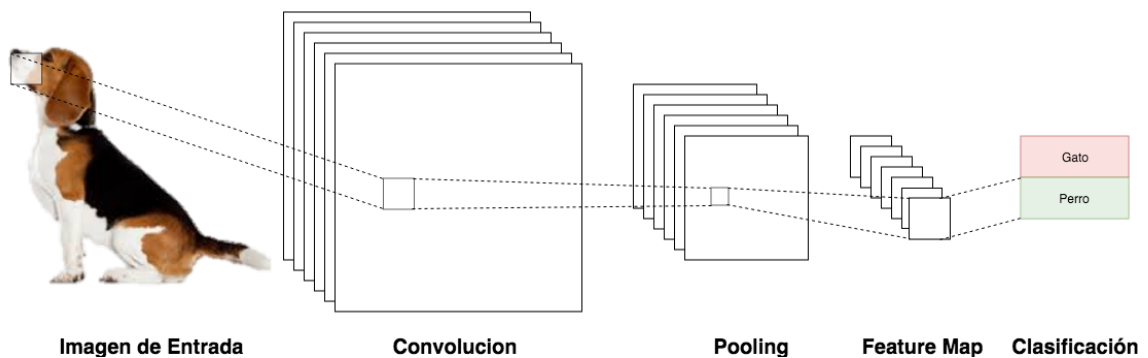


Figura 2.6: Ejemplo de las diferentes capas para la obtención de una clasificación.

2.1.5. Hiperparámetros

En el aprendizaje automático, existen parámetros de modelo y parámetros ajustables; estos últimos pueden hacer que las redes se entrenen y generen mejores resultados en un menor tiempo. Estos parámetros de ajuste se denominan hiperparámetros, y se encargan del control de las funciones de optimización y la selección de modelos durante el entrenamiento con nuestro algoritmo de aprendizaje. En otras palabras se puede definir hiperparámetro como

cualquier configuración asignada que el usuario pueda seleccionar libremente que afectará el rendimiento del modelo (Patterson and Gibson, 2017).

Los hiperparámetros se dividen en varias categorías como:

1. Tamaño de las capas.
2. Magnitud.
3. Regularización.
4. Activaciones.
5. Estrategia de inicialización de pesos.
6. Asignación de épocas durante el entrenamiento.
7. Esquema de normalización para datos de entrada.

2.1.6. Clasificación

La clasificación es un modelado basado en delinear clases de salida basadas en un conjunto de características de entrada. La clasificación nos otorga una salida, esa salida responde a la pregunta “¿Que es?”. En otras palabras La variable dependiente y es categórica más que numérica (Patterson and Gibson, 2017).

La forma mas básica de clasificación es la clasificación binaria que solo tiene una simple salida de dos valores. Los valores posibles de salida valores también son denominados como clases y son los posibles resultados de nuestra clasificación. En el caso de la clasificación binaria solo existen dos posibles clases la cual serían los números 1 y 0.

Dejando de lado la clasificación binaria, que solo cuenta con dos posibles clases, también se pueden realizar clasificación de más de dos clases posibles, por ejemplo el conjunto de datos “MNIST Database of Handwritten Digit” que contiene 10 clases posibles (Deng, 2012), que se utilizará para probar la metodología propuesta. En la Figura 2.7 se pueden observar las imágenes de los números que contiene la base de datos MNIST.

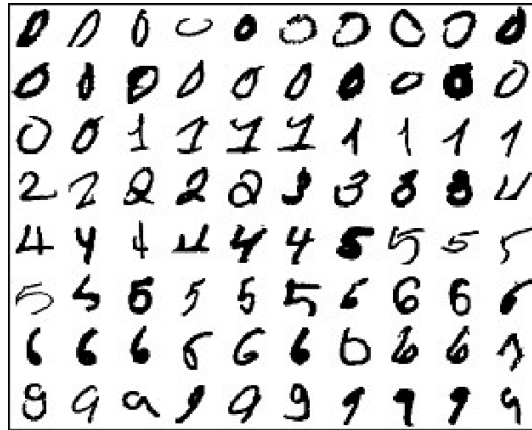


Figura 2.7: Representación de la base de datos MNIST (Deng, 2012).

2.1.7. Regresión

La regresión se refiere a las funciones que intentan predecir una salida con respecto a un valor. Este tipo de funciones realiza una predicción de la variable dependiente conociendo la variable independiente. La clase más común de regresión es la regresión lineal.

La regresión lineal intenta encontrar una función que describa la relación entre x e y , y para valores conocidos de x , predice valores de y que resultan ser precisos. Un ejemplo de un problema de regresión lineal será el intentar conocer el gasto mensual de gasolina que genera un automóvil, donde se necesita encontrar la función para encontrar el costo con base en la distancia recorrida mensualmente. En este caso la gasolina sería la variable dependiente y la variable independiente sería la distancia recorrida, de esta manera se puede definir como en la Formula 2.1.

$$\text{costo} = f(\text{distancia}) \tag{2.1}$$

Visualmente se puede representar un problema de regresión, como encontrar una línea que se acerca a diferentes puntos en un diagrama de dispersión de datos como por ejemplo la Figura 2.8.

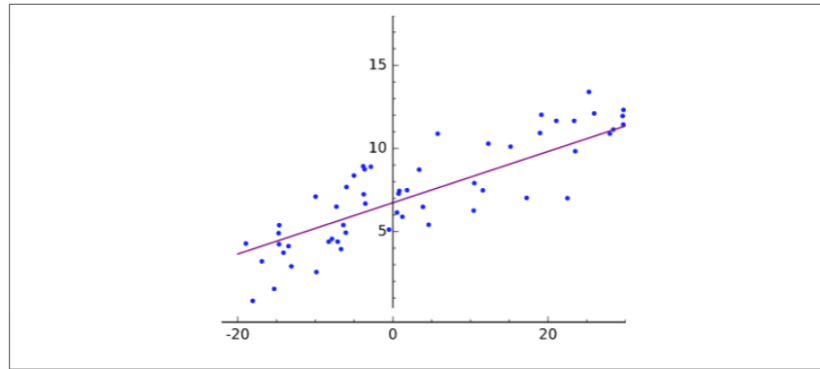


Figura 2.8: Representación de un diagrama de dispersión (Patterson and Gibson, 2017)

2.2. Algoritmos evolutivos

Los algoritmos evolutivos son métodos de optimización y búsqueda de soluciones basados en los postulados de la evolución biológica. En ellos se mantiene un conjunto de entidades que representan posibles soluciones, las cuales se mezclan, y compiten entre sí, de tal manera que las más aptas son capaces de prevalecer a lo largo del tiempo, evolucionando hacia mejores soluciones cada vez.

Los algoritmos evolutivos, y la computación evolutiva, son una rama de la inteligencia artificial. Son utilizados principalmente en problemas con espacios de búsqueda extensos y no lineales, en donde otros métodos no son capaces de encontrar soluciones en un tiempo razonable.

Siguiendo la terminología de la teoría de la evolución, las entidades que representan las soluciones al problema se denominan individuos o cromosomas, y el conjunto de éstos, población. Los individuos son modificados por operadores genéticos, principalmente el cruce, que consiste en la mezcla de la información de dos o más individuos; la mutación, que es un cambio aleatorio en los individuos; y la selección, consistente en la elección de los individuos que sobrevivirán y conformarán la siguiente generación. Dado que los individuos que representan las soluciones más adecuadas al problema tienen más posibilidades de sobrevivir, la población va mejorando gradualmente.

2.2.1. Algoritmo Genético

La computación evolutiva simula la teoría de la evolución en una computadora, esta simulación puede ser realizada utilizando una serie de algoritmos de optimización. Esta optimización iterativamente mejora la calidad de las soluciones hasta que se encuentra una solución óptima. El enfoque evolutivo está basado en modelos de la selección natural y la genética. Estos modelos son basados en computación evolutiva y está formada por una amplia gama de términos combinando algoritmos genéticos, estrategias evolutivas y programación genética. Todas estas técnicas simulan la evolución usando los procesos de selección, mutación y reproducción de acuerdo con (Negnevitsky, 2005).

Los algoritmos genéticos son una clase algoritmos de búsqueda estocástica basados en la evolución biológica. Esta es la técnica más popular en la computación evolutiva. la representación del problema utilizada (Cromosoma) es una cadena de bits de longitud fija, la cual es mostrada en la Figura 2.9. Cada posición en la cadena representa una característica de un individuo (Gen). Por lo que, el cromosoma representa una propuesta de solución al problema dado.

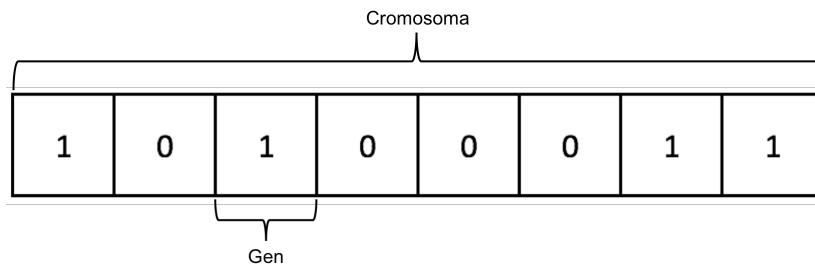


Figura 2.9: Representación del cromosoma (Sivanandam and Deepa, 2008).

2.2.2. Algoritmo Genético Compacto

El AGc es una técnica de búsqueda/optimización probabilística. Esta relacionado con el AG y otros algoritmos evolutivos que están inspirados en la teoría de la evolución por medio de la selección natural. El objetivo AGc es simular el comportamiento del AG con mucho menos requerimiento de memoria (Sin requerir que una población sea mantenida en memoria). Esto se logra manteniendo un vector de probabilidades en lugar de la población entera. Las soluciones candidatas son probabilísticamente generadas desde la cadena de característi-

cas. Las características que proveen una mejor solución se utilizan para realizar pequeños cambios en un vector de probabilidades (Brownlee, 2011). El Algoritmo 2.1 muestra el AGc (Harik et al., 1999).

Algoritmo 2.1 AGc.

```

1: procedure AGC( $n, l$ )      ▷  $n$  como tamaño de población,  $l$  longitud del cromosoma
2:   Inicializar el vector de probabilidad
3:   for  $i := 1$  to  $l$  do  $p[i] := 0.5$ ; do
4:     Generar 2 individuos desde el vector
5:      $a := \text{generate}(p)$ ;
6:      $b := \text{generate}(p)$ ;
7:      $\text{Evaluate}(a)$ ;
8:      $\text{Evaluate}(b)$ ;
9:      $\text{winner}, \text{loser} := \text{compete}(a, b)$ ;
10:    //Actualizar el vector utilizando el mejor
11:    for  $i := 1$  to  $l$  do
12:      if  $\text{winner}[i] \neq \text{loser}[i]$  then
13:        if  $\text{winner}[i] = 1$  then
14:           $p[i] := p[i] + 1/n$ 
15:        else
16:           $p[i] := p[i] - 1/n$ ;
17:        end if
18:      end if
19:    end for
20:    // Revisar si el vector a convergido
21:    for  $i := 1$  to  $l$  do
22:      if  $p[i] > 0$  &  $p[i] < 1$  then
23:        return to step 4;
24:      end if
25:    end for
26:  end for
27:  return  $p$ 
28: end procedure

```

El AGc inicia definiendo el vector de probabilidad en la línea 2. Enseguida, desde ese vector dos individuos se generan y evalúan. El proceso de evaluación de los individuos prueba los hiperparámetros propuestos por el AGc. Los individuos generados compiten y de acuerdo al ganador el vector de probabilidad es actualizado. Este proceso se repite hasta que el criterio de convergencia se cumple.

2.3. Computación Distribuida

La computación distribuida cubre todos los aspectos de computación y acceso a la información a través de múltiples elementos de procesamiento. Actualmente la computación distribuida es muy usada en los nuevos sistemas de información, por ejemplo cualquier sistema móvil hoy en día está interconectado con otros sistemas para resolver una necesidad.

La literatura define un sistema distribuido como un conjunto de entidades independientes que de manera coordinada cooperan para resolver un problema que individualmente no puede ser solucionado (Kshemkalyani and Singhal, 2008). Un sistema distribuido normalmente cuenta con las siguientes características:

1. No existe un reloj físico en común.
2. No existe memoria compartida.
3. Separación geográfica.
4. Autonomía y heterogeneidad .

Cada equipo de cómputo tiene una unidad de procesamiento de memoria y las computadoras están conectadas por una red de comunicación como se observa en la Figura 2.10.

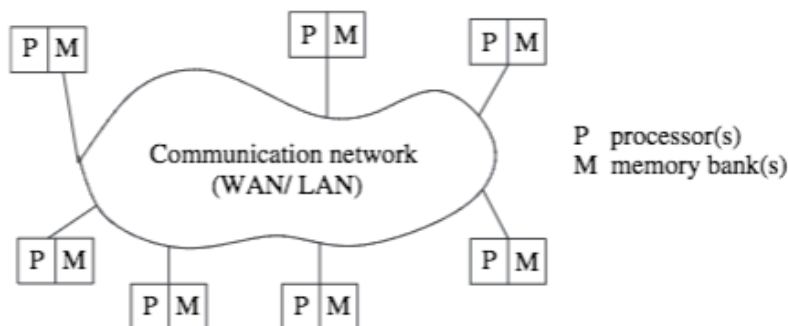


Figura 2.10: Sistema distribuido conectando procesadores a través de una red de comunicación (Kshemkalyani and Singhal, 2008).

La Figura 2.11 muestra las relaciones de los componentes de software que se ejecutan en cada una de las computadoras y utilizan el sistema operativo local y la pila de protocolos de red para su funcionamiento. El software distribuido también se denomina *Middleware*.

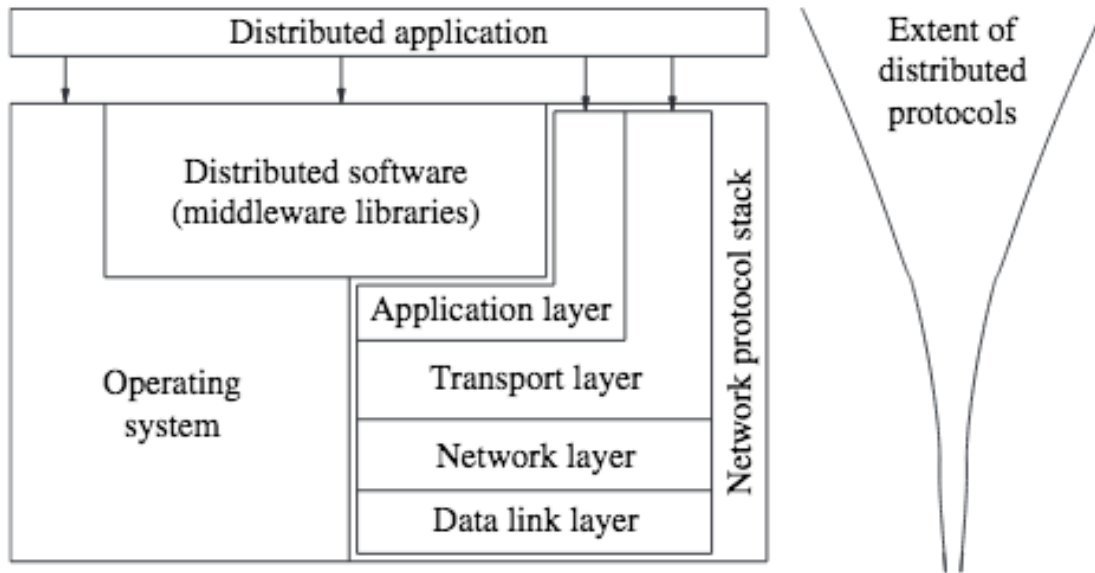


Figura 2.11: Ejemplo de arquitectura de un sistema distribuido (Kshemkalyani and Singhal, 2008).

La ejecución distribuida es la ejecución de procesos en todo un sistema distribuido para lograr un objetivo común en colaboración. El sistema distribuido utiliza una arquitectura en capas para separar la complejidad del diseño del sistema. El *Middleware* es el software distribuido que en la Figura 2.11 realiza una interacción con los componentes de software de cada procesador. Procesador se dice que es aquel que realiza un proceso para dejar claro ese concepto ya que puede ser confundido con el procesador de un equipo de cómputo.

2.3.1. Concurrencia

Concurrencia es la capacidad de de un algoritmo programa o problema de ser ejecutado parcialmente sin afectar el resultado esperado. Los procesos pueden ser ejecutados en la misma computadora en múltiples procesadores en distintos hilos de ejecución o ejecutados de forma distribuida separados físicamente. En la Figura 2.12 se muestra un ejemplo de diferentes procesos ejecutándose concurrentemente.

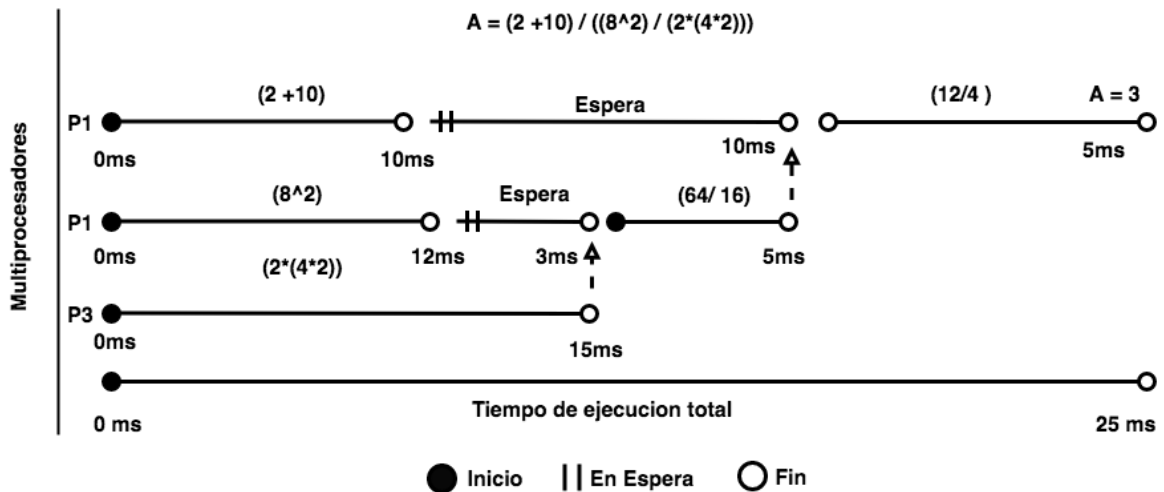


Figura 2.12: Ejemplo de solución de una operación aritmética usando procesos concurrentes.

2.3.2. Semáforos

El semáforo es una técnica para controlar recursos que son compartidos o en un entorno mutiproseso, el cual fue inventado por Edsger Dijkstra en el año 1965 y se uso por primera vez en el sistema operativo llamado Theos.

Este mecanismo de control nos permite decidir en que momento un recurso compartido es accedido. Comúnmente este mecanismo es utilizado cuando se realiza una sincronización entre diferentes procesos y se necesita garantizar la consistencia del recurso compartido.

En la Figura 2.12 se muestra un ejemplo de un semáforo, donde se puede observar como el P1 espera hasta que los procesos P2 y P3 hayan finalizado, ya que si el P3 realiza la operación sin contar con los resultados de los demás procesos ocasionaría un problema de inconsistencia y el resultado final será afectado pudiendo generar un valor erróneo.

2.3.3. Algoritmos Genéticos Distribuidos

La idea básica detrás de la mayoría de los programas paralelos o distribuidos es separar una tarea en diferentes fragmentos; estos fragmentos se realizarán simultáneamente usando múltiples procesadores. Este enfoque conocido como “divide y vencerás” puede aplicarse a los AG de muchas formas diferentes, y la literatura contiene muchos ejemplos de implementaciones paralelas/distribuidas exitosas. Algunos métodos de paralelización utilizan solo una

población, mientras que otros la dividen en varias sub-poblaciones relativamente aisladas. Algunos métodos pueden explotar arquitecturas informáticas paralelas masivas, mientras que otros son más adecuados para multicomputadoras con menos pero más potentes elementos de procesamiento (Cantú-Paz, 1998).

2.3.4. Modelo Maestro-Esclavo

El enfoque Maestro-Esclavo de los algoritmos genéticos es un enfoque básico para ejecutar en paralelo. En este modelo, las principales operaciones del AG se realizan mediante un proceso llamado maestro. El maestro mantiene a toda la población y realiza operaciones de cruce, mutación y selección. El resto de las unidades de procesamiento, se denominan trabajadores o esclavos; realizan solo la evaluación de soluciones (Abdelhafez et al., 2019). El modelo Maestro-Esclavo es similar al comportamiento básico del AG, excepto la evaluación paralela para los individuos. El objetivo principal de este diseño es reducir la capacidad de cómputo requerida y, por lo tanto, reducir el tiempo de ejecución. En la Figura 2.13 se muestra la representación del modelo Maestro-Esclavo.

2.3.5. AMQP: Advanced Messaging Queuing Protocol

El Protocolo “Advanced Messaging Queuing Protocol (AMQP)” es un estándar de código abierto para realizar el paso de mensajes entre aplicaciones u organizaciones. Conecta los sistemas, alimenta los procesos de negocios con la información que necesitan y transmite de manera confiable las instrucciones que logran sus objetivos.

AMQP fue diseñado con las siguientes características principales:

1. Seguridad.
2. Confiabilidad.
3. Interoperabilidad.
4. Estandar.
5. Abierto.

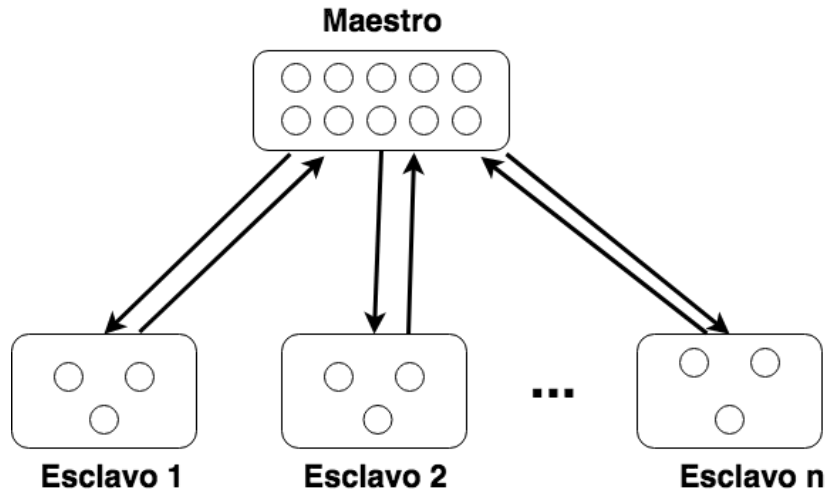


Figura 2.13: Representación del Modelo Maestro-Esclavo.

La comunicación en el enfoque Maestro-Esclavo son hechas por el proceso maestro, y por lo tanto no hay comunicación directa entre esclavos por diseño. El maestro siempre se detiene, espera enviar y recibir información de todos los esclavos antes de pasar a la próxima generación. El uso de este tipo de comunicación centralizada hace que el modelo Maestro-Esclavo sea más eficiente en los sistemas informáticos de memoria compartida cuando la función de condición física es compleja y requiere mucho tiempo (Luque and Alba, 2011).

Capítulo 3

Estado del Arte

La búsqueda de mejores soluciones con menores tiempos de procesamiento ha sido un gran potenciador para un gran número de trabajos enfocados en la búsqueda de hiperparámetros y optimización de los procesos de entrenamiento de redes de aprendizaje profundo. Se realizó una investigación de los distintos métodos de optimización que han sido utilizados para la optimización de las redes de aprendizaje profundo. También se analizaron ciertas arquitecturas para sistemas distribuidos con el objetivo de construir un diseño arquitectónico que soporte la distribución.

3.1. Metodos Tradicionales de Búsqueda

En el trabajo Bergstra and Bengio, 2012 en el cual se realizó una comparativa entre la búsqueda manual, búsqueda de cuadrícula y búsqueda aleatoria para el ajuste de hiperparámetros en redes neuronales y redes de aprendizaje profundo. Para la búsqueda manual se utilizaron las estrategias más utilizadas para la optimización de hiperparámetros. Encontrando que los resultados realizando un ajuste manual de hiperparámetros superaron a los resultados utilizando búsqueda aleatoria para cuando los modelos contaban 32 dimensiones. Por último plantean que se debe trabajar con métodos de búsqueda/optimización que sean secuenciales y adaptativos para lograr una alta efectividad.

En el trabajo de Snoek et al., 2012 se utilizó optimización bayesiana aplicada para mejorar los modelos los algoritmos aprendizaje. En este trabajo, se considero el problema de ajuste manual de hiperparámetros (donde se requiere mucha experiencia por el gran número de

hiperparámetros existentes) para utilizar optimización bayesiana, en el que el rendimiento del algoritmo de aprendizaje se modela como una muestra de un proceso gaussiano. Fue utilizada la base de datos llamada CIFAR-10, donde la optimización Bayesiana superó a la de un experto humano en la selección de hiperparámetros superando el estado de la técnica en más del 3 %.

3.2. Métodos evolutivos

En la literatura se muestran diferentes formas de implementar un AG para la búsqueda de mejores soluciones. En el trabajo presentado por Stanley and Miikkulainen, 2002 se busca obtener alguna ventaja de las técnicas evolutivas; dichas técnicas son utilizadas con la finalidad optimizar la topología de la red y los pesos iniciales. Con base en lo anterior se presenta un método llamado Neuro-evolución de topologías aumentadas, que a través de la implementación del AG es posible optimizar los modelos modificando la topología y los pesos mediante la evolución.

En el trabajo presentado por Miikkulainen et al., 2019 se define una nueva alternativa para la optimización de la topología, los componentes e hiperparámetros para una red de aprendizaje profundo utilizando Neuro-evolución. Donde el cromosoma definido esta generando soluciones con diferentes configuraciones y se evalúa el resultado de la red.

El trabajo presentado por Desell, 2017 expone un nuevo algoritmo que es llamado *Evolutionary exploration of augmenting convolutional topologies (EXACT)*, el cual es capaz de evolucionar la estructura de una RNC. *EXACT* está modelado con base en el algoritmo *Neuro-evolution of augmenting topologies (NEAT)* que permite escalar en ambientes computacionales de manera distribuida y evolucionar las RNC. Los experimentos en este trabajo utilizaron la base de datos MNIST en el que obtuvieron un 98.32 % de precisión.

3.3. Distribución y Paralelismo

Se han encontrado una serie de trabajos sobre como aprovechando el paralelismo y la distribución se han mejorado los tiempo de entrenamiento, como el trabajo presentado por Dean

et al., 2012 en este trabajo, se considera el problema de la construcción de una red profunda con miles de millones de parámetros utilizando decenas de miles de núcleos de CPU. Para esta problemática se propone un marco de trabajo llamado *DistBelief* que básicamente aprovecha el poder de cómputo de grupos de equipos para entrenar modelos grandes. También se propone un algoritmo de entrenamiento distribuido llamado *Downpour SGD*, que consiste en un descenso de gradiente estocástico que soporta un gran número de modelos. Por último un algoritmo que soporta una gran variedad de procedimientos de optimización por lotes de manera distribuida.

En el trabajo realizado por Hegde and Usmani, 2016 se realiza una exploración a las diferentes formas de paralelizar o distribuir el aprendizaje profundo. Se analizaron los tiempos de ejecución de una RNC utilizando un CPU y una GPU. El uso de una GPU superó ampliamente los tiempos de entrenamiento de una RNC en comparación con los tiempos del uso del CPU, por lo cual recomiendan el uso del GPU sobre el CPU para el entrenamiento una RNC.

En el trabajo presentado por Castillo et al., 2011 se utiliza el protocolo denominado transferencia de estado representacional (REST) basado en HTTP como protocolo de comunicación para realizar la distribución de las tareas del AG. Donde definen el cromosoma con los siguientes hiperparámetros mostrados en la Tabla 3.1.

Tabla 3.1: Hiperparámetros utilizados en el algoritmo evolutivo en el trabajo de (Castillo et al., 2011).

Hiperparámetros	Valores
Número de generaciones	100
Individuos por población	100
Porcentaje de reemplazo de la población	30 %
Número unidades en la capa oculta	2-90
Número de épocas	300

En el trabajo mas reciente encontrado sobre distribución de procesos de los algoritmos genéticos se encuentra el de Abdelhafez et al., 2019 donde realizan la paralelización de los procesos del AG comparando dicha ejecución de forma síncrona y asíncrona, donde la forma asíncrona superó la forma sincrónica en términos de tiempo de ejecución.

En dicho trabajo, el modelo maestro y esclavo ha muestran los siguientes resultados:

1. El modelo maestro/esclavo es capaz de escalar bien sobre multiprocesadores para dife-

rentes números de problemas.

2. Los experimentos confirman una proporcionalidad entre la velocidad y la complejidad de la función de aptitud.
3. Los resultados de los experimentos proveen que el AG síncrono y asíncrono tienen diferente comportamiento en cuanto a velocidad cuando corren en sistemas multiprocesadores.
4. Se encontró que la velocidad podría ser alta e incluso súper lineal en multiprocesadores para diferentes algoritmos.

3.4. Tabla Comparativa

En la Tabla 3.2 se muestra una comparación de los diferentes trabajos que realizan la búsqueda del conjunto óptimo de hiperparámetros y acelerar el tiempo de búsqueda, donde se mencionan sus características, que caso se estudio y que conjunto de datos de entrenamiento utilizaron.

Tabla 3.2: Comparativa de los diferentes trabajos que realizan la búsqueda del conjunto óptimo de hiperparámetros y acelerar el tiempo de búsqueda.

Autor	Trabajo	Características	Caso de Estudio	Conjunto de Datos
Bergstra, J. and Bengio, Y	Random Search for Hyper-Parameter Optimization.	Este trabajo presenta una comparativa de los métodos de Búsqueda en Cuadrícula, Aleatoria y Manual para la optimización de hiperparámetros.	RNA.	MNIST. Rectangulos. Convex.
Snoek, J. et al.	Practical Bayesian Optimization of Machine Learning Algorithms.	Este trabajo presenta métodos para realizar la optimización Bayesiana para la selección de hiperparámetros de algoritmos generales de aprendizaje automático.	RNC.	CIFAR-10.
Stanley, K. O. and Miikkulainen, R.	Evolving Neural Networks Through Augmenting Topologies.	Este trabajo presenta un método llamado Neuro-Evolución de topologías aumentadas (NEAT). Se utilizan Algoritmos genéticos para encontrar topologías que aumenten la calidad de los modelos de las RNA.	RNA.	No especificado.
Miikkulainen, et al.	Evolving Deep Neural Networks.	Este trabajo presenta la Neuro-Evolución de topologías aumentadas, pero también incluye la modificación de hiperparámetros basado en el principio de la Neuro-Evolución.	RNC. LSTM.	CIFAR-10.
Desell, T.	Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing.	Este trabajo presenta Evolutionary exploration of augmenting convolutional topologies (EXACT) basado en NEAT. Exact permite escalar en ambientes computacionales de manera distribuida. Este trabajo presenta un 98.32 % de precisión.	RNC.	MNIST.
Dean et al.	Large Scale Distributed Deep Networks	Este trabajo presenta DisBelief el cual propone es utilizar conjuntos de equipos de cómputo para aprovechar el poder computacional en modelos muy extensos.	RNC.	ImagenNET.
Hegde, V. and Usmani, S	Parallel and Distributed Deep Learning.	Este trabajo presenta un esquema paralelo distribuido donde utilizan tarjetas GPU para acelerar el entrenamiento de las CNN.	RNC	No especificado.
Castillo et al.	Distributed Evolutionary Computation Using REST.	Este trabajo presenta la distribución de los procesos del AG utilizando el protocolo denominado transferencia de estado representacional (REST), el cual realiza la comunicación entre los componentes esclavo y maestro.	RNA.	Glass
Abdelhafez, A. et al.	Performance Analysis of Synchronous and Asynchronous Distributed Genetic Algorithms on Multiprocessors	Este trabajo presenta realizar la paralelización de los procesos del AG comparando la ejecución de forma síncrona y asíncrona, donde la forma asíncrona superó a la forma síncrona en términos de tiempo de ejecución. En este trabajo se evaluó el modelo Maestro-Esclavo en contra de un modelo de isla para la distribución de los procesos.	No especificado.	No especificado.

Capítulo 4

Diseño del Experimento

En esta sección se presenta a detalle el diseño del AGcD inspirado en el AGc. A su vez se describen los distintos experimentos que se realizaron para la evaluación del AGcD.

La Figura 4.1 muestra el proceso en su totalidad para la optimización de RNC. El proceso completo cuenta con 3 sub-procesos, los cuales son *pre-procesamiento de datos*, *Definición del modelo*, *Optimización* y como resultado la obtención del *Modelo óptimo*.

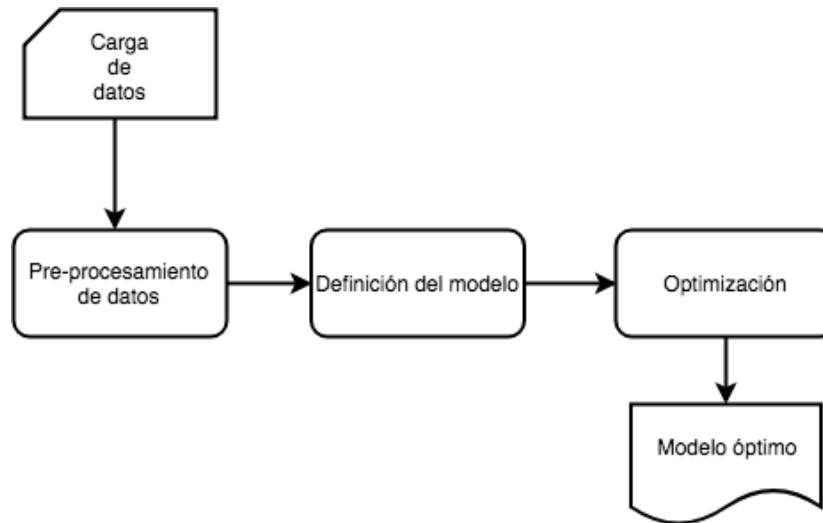


Figura 4.1: Diagrama de flujo del desarrollo del experimento.

En el subproceso de *Pre-Procesamiento de los datos*, se toman como entrada los datos proporcionados en el cargado de los datos. Normalmente en los conjuntos de datos de entrenamiento existen anomalías o casos atípicos también llamados *outliers*. Estas anomalías tienen que ser trabajadas para evitar que el modelo sufra algún sesgo. Además, después de la sanitización de los datos se realiza un proceso de normalización de los mismos.

En el subproceso de *Definición del modelo* se realiza la definición de la arquitectura y topología con la cual el modelo será creado y entrenado. Esta arquitectura será implementada al evaluar el modelo en los nodos trabajadores.

En el subproceso de *Optimización* es el que realiza la mayor cantidad de trabajo, se utiliza un AGcD. Este proceso consiste en realizar la creación de las generaciones especificadas en el nodo jefe, a su vez se realiza la creación de los individuos (posibles configuraciones de hiper-parámetros). Posteriormente estos individuos serán enviados por el Componente Maestro hacia el Componente Esclavo para ser procesados, en donde se le asignó la arquitectura según la definición especificada en el subproceso anterior. También se realiza el entrenamiento según el individuo que se le haya asignado al nodo trabajador, al finalizar el entrenamiento se envían los resultados obtenidos al nodo jefe en donde se realizara la evaluación de los individuos, se seleccionará el mejor y continuará con la siguiente generación de individuos hasta concluir con el proceso y entregar un modelo óptimo.

4.1. Pre-procesamiento de datos.

En la etapa de Pre-Procesamiento de datos se realizan distintas tareas que preparan los datos para que puedan ser procesados correctamente. Entre estas tareas está el remover los datos atípicos (Outliers) en la serie de tiempo y la normalización de los datos.

4.1.1. Datos atípicos

Un dato atípico es una observación que cae fuera del patrón general de la distribución de los datos. Usualmente, la presencia de un dato atípico indica algún tipo de problema. Esto puede ser un caso en el cual el dato no sea conveniente para el problema que se estudia, o un error en la medición. Los datos atípicos en los datos pueden sesgar significativamente los resultados de los datos que se están procesando. En la Figura 4.2 se presenta un ejemplo de una serie de datos en la cual se encuentran dos datos atípicos.

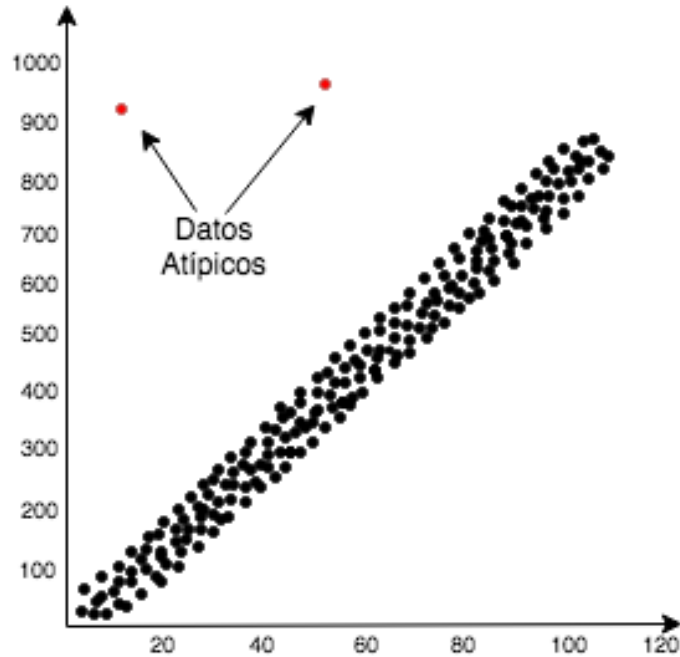


Figura 4.2: Dato atípico en un conjunto de observaciones.

4.1.2. Normalización

Además del análisis de datos atípicos en la serie de tiempo se realizó una normalización de los datos en la serie de tiempo, lo cual significa que las observaciones fueron pasadas a un rango más pequeño de valores entre $0 \leq x_i \leq 1$ donde x_i es una observación en la serie de tiempo X en la posición i .

La Equación 4.1 muestra la manera en la que se llevó a cabo la normalización de los datos. Las funciones X_{max} y X_{min} obtienen el valor máximo y mínimo respectivamente de X .

$$X_{normalizado} = \frac{x_i - X_{min}}{X_{max} - X_{min}} \quad (4.1)$$

La Figura 4.3 muestra un ejemplo de la normalización de una serie de tiempo, a la izquierda se muestra X que son los datos como fueron medidos originalmente por los sensores sin pasar por un proceso de normalización. Sus valores se encuentran en un rango no estandarizado. A la derecha se encuentra X_n la cual muestra la serie de tiempo normalizada entre un

rango de valores de [0, 1]. También se muestran los valores Máximo(X_{Max}) y Mínimo(X_{Min}) respectivamente de la serie de tiempo que serán utilizados en la sustitución para el ejemplo que muestra la Figura 4.3.

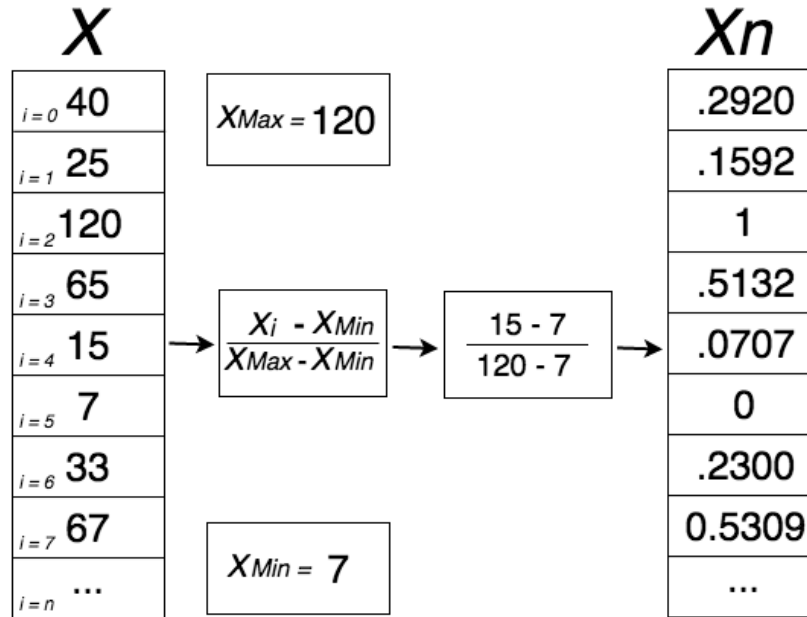


Figura 4.3: Serie de tiempo normalizada en un rango de valores entre [0, 1].

4.2. Definición de modelo.

En el proceso de Definición del modelo es donde se realiza la selección del tipo de problema a resolver, ya sea un problema de regresión o un problema de clasificación. Después se realiza la definición de la arquitectura con la que nuestro modelo sera construido y por último se presentarán las métricas con las que se evaluara el modelo.

Para la construcción del modelo es necesario seleccionar una arquitectura de una red de aprendizaje profundo(fueron especificadas en el capítulo 1), el caso de estudio seleccionado se utilizará una RNC. La topología de la RNC con la cual fue construida el modelo se muestra en la Figura 4.4. La topología fue seleccionada de forma artesanal, probando diferentes configuraciones de las diferentes capas respetando el orden descrito en el capítulo 2. La topología especificará el número de capas de la red de aprendizaje profundo.

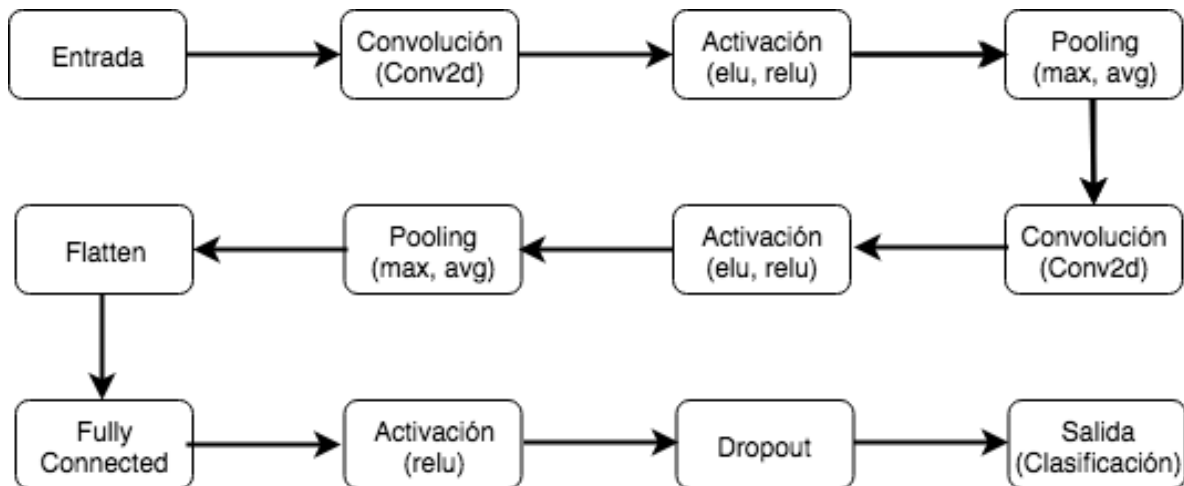


Figura 4.4: Topología seleccionada de la CNN.

4.3. Optimización.

En el proceso de la optimización se explica detalladamente la forma en que se lleva a cabo la optimización de los modelos de aprendizaje profundo. Este proceso cuenta con 6 sub-procesos principales los cuales son *Inicialización del Componente Maestro*, *Inicialización de los Componentes Esclavos*, *Creación de los individuos*, *Asignación de los individuos*, *Entrenamiento*, *Competencia* y como resultado la obtención del *Modelo óptimo*. En la Figura 4.5 se observa un diagrama de los pasos a seguir para la generación de un modelo óptimo.

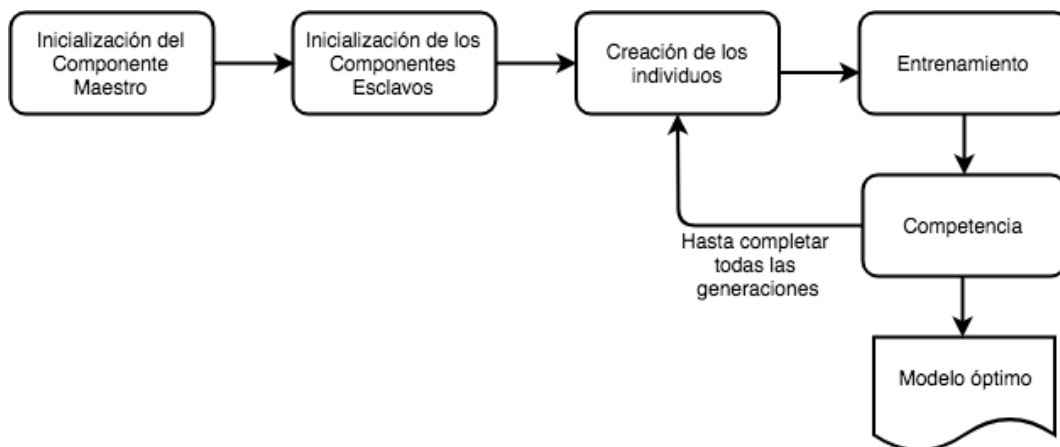


Figura 4.5: Diagrama para la generación de un modelo óptimo.

4.3.1. Arquitectura propuesta.

Antes de presentar la etapa de optimización, se explicará un punto de gran importancia que no es un proceso dentro del AGcD pero si es parte integral de este. En esta sección describiremos la arquitectura propuesta para llevar a cabo los procesos del AGcD. En la Figura 4.6 se muestra de forma general la arquitectura utilizada.

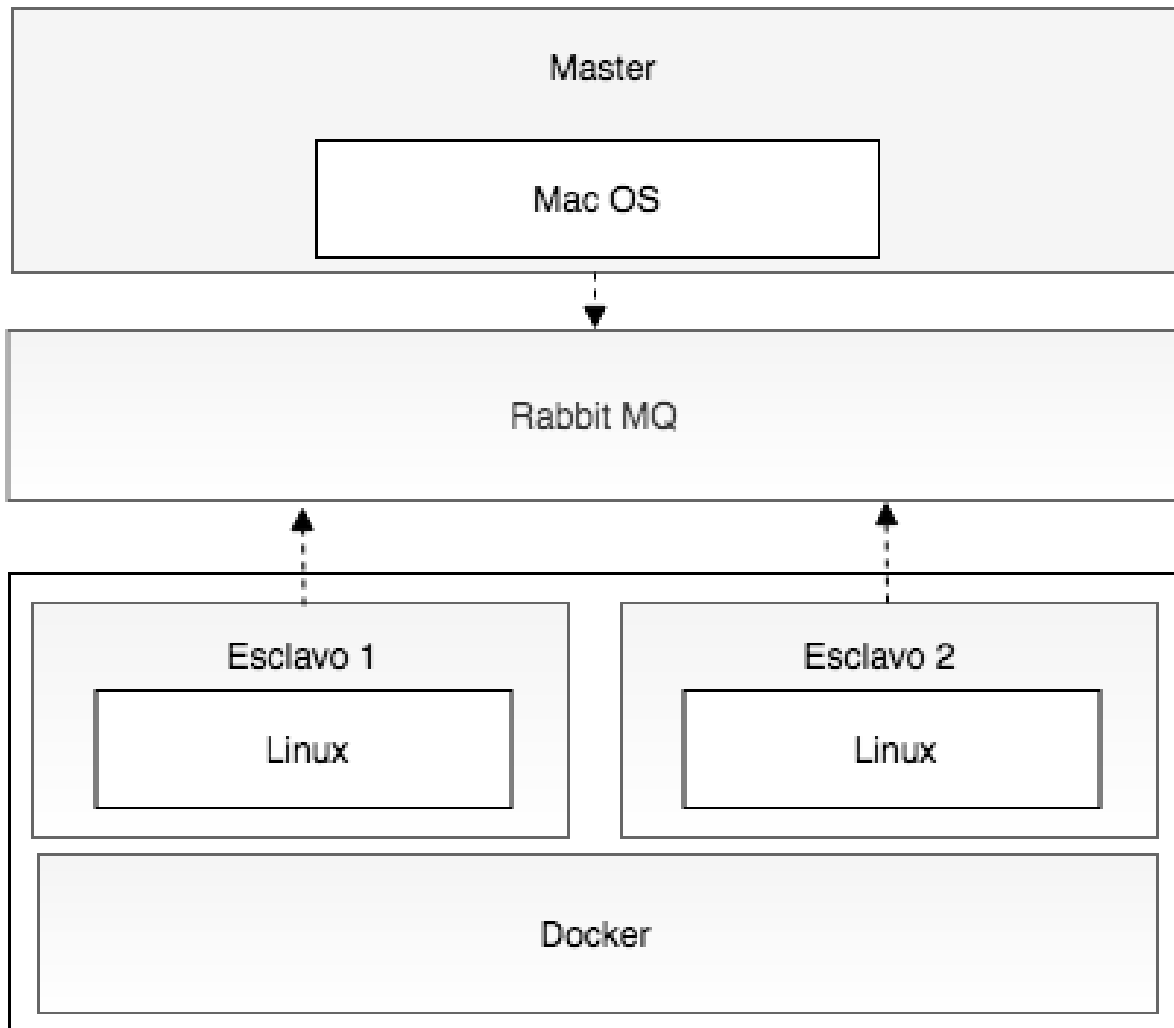


Figura 4.6: Arquitectura general.

El componente *Esclavo* es el encargado de realizar la ejecución de la función de aptitud del individuo, esto quiere decir que en este caso el componente esclavo realizará el entrenamiento de la RNC según la configuración especificada por el individuo. Al finalizar la ejecución del entrenamiento, este componente retorna el valor de aptitud especificado.

El componente *Maestro* es el encargado de coordinar los procesos que se realizan en el AGc. Este componente realiza la generación de los individuos, la recepción de individuos entrenados y la competencia de los individuos. Además es el encargado de realizar la asignación del individuo que será procesado por el esclavo.

El componente llamado Docker es el encargado de la virtualización de los ambientes requeridos para realizar la ejecución de los procesos necesarios para el funcionamiento correcto del sistema. Docker nos permite utilizar entornos de aplicación aislados llamados contenedores. Esto es diseñado para el apoyo de los desarrolladores con el fin de poder replicar el ambiente utilizado en los esclavos sin mayor complejidad.

El último componente *RabbitMQ*, un “intermediario de mensajes” de código abierto que implementa el protocolo Avanzado de Message Queue Server (AMQP), tiene como función el garantizar que los datos (los mensajes) vayan de un productor a los consumidores, para que estos mensajes sean procesados. El principal destinatario de los mensajes es la “cola”, un acumulador de datos potencialmente ilimitado, que reside dentro de RabbitMQ. Si el productor y los consumidores están conectados a la misma cola, pueden comunicarse sin que realmente se conozcan entre sí. Esto convierte a RabbitMQ en una poderosa herramienta para la distribución escalable y confiable para realizar la segmentación de los procesos. En la Figura 4.7 se puede observar el diagrama general de la arquitectura propuesta para la distribución de los diferentes procesos del AGc.

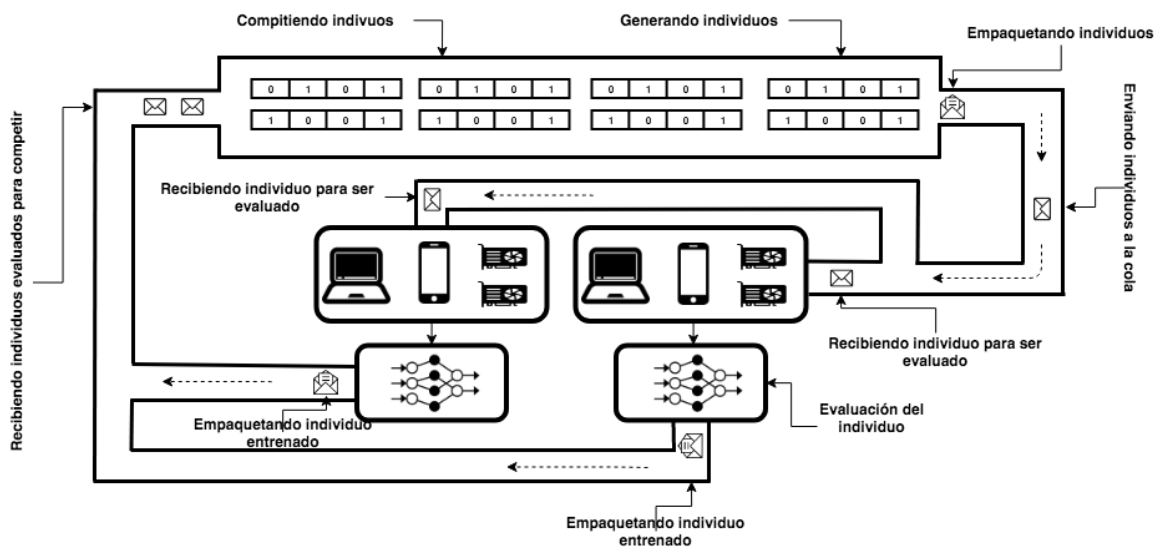


Figura 4.7: Diagrama de la arquitectura general.

4.3.2. Inicialización del Componente Maestro

En este subproceso el Componente Maestro es inicializado, suscribiéndose al intermediario de mensajes (Rabbit MQ), asociando una función que se ejecutará de manera asincrónica al momento de recibir un individuo. También se crean las colas de individuos e individuos entrenados. La cola de individuos registrará aquellos que están siendo generados en cada generación del AGcD en la cual el Componente Maestro será el productor. La cola de individuos entrenados registrará los individuos que ya hayan finalizado función de aptitud en donde el Componente Maestro será consumidor. El Componente Maestro necesita conocer el dominio y las credenciales del servidor de Rabbit MQ para realizar la conexión. En la Figura 4.8 se muestra una representación del Componente Maestro comunicándose a través del protocolo AMQP para el registro de los Componentes Esclavos.

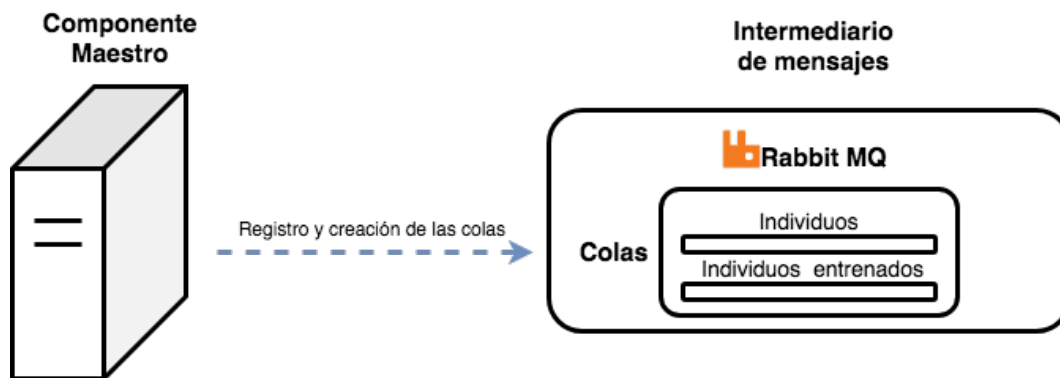


Figura 4.8: Componente Maestro se suscribe al intermediario de mensajes de Rabbit MQ.

Cuando se haya configurado la conectividad entre el Componente Maestro y el servidor de Rabbit MQ, el Componente Maestro continuará con la definición del cromosoma que se utilizará para definir los hiperpámetros a optimizar en la ejecución del AGcD. Dichos hiperpámetros son los siguientes:

1. Generales

- a) **Épocas:** Es un número entero positivo que limita el número de pasos que el conjunto de validación es evaluado.
- b) **Aprendizaje:** Es un número de punto flotante que representa la magnitud de la actualización por cada época de entrenamiento.

- c) **Entrenamiento:** Es un número que representa la proporción de datos utilizados en el conjunto de entrenamiento.
- d) **Optimizador:** Son funciones que calculan gradientes para una medida de pérdida y aplican gradiente a variables. Los tres posibles valores de este parámetro son Stochastic Gradient Descent(SGD), AdamOptimizer(ADAM) y RMSPropOptimizer (RMSProp).
- e) **Activación:** Funciones que proveen la no linealidad. Este parámetro puede tomar dos valores: una función rectificadora (relu) y una función para suavizar la no linealidad (elu).

2. Convolucionales

- a) **Tamaño del Filtro:** Es un número que representa el tamaño del filtro que recorrerá la matriz que representa la imagen.
- b) **Strides:** Una lista de números enteros que representa el número de características que serán recorridas de izquierda a derecha en la matriz, moviendo el filtro por cada dimensión en el vector de entrada.
- c) **Padding:** Indica si el filtro puede ir más allá de los límites de la matriz. Los valores que puede tomar este parámetro son dos *same* y *valid*.
- d) **Pooling:** Reduce la dimensionalidad de la entrada permitiendo hacer suposiciones de características contenidas en una región de la entrada. Los valores que puede tomar este parámetro son *maxpooling* y *avgpooling*.
- e) **Dropout:** Es una técnica de regularización para reducir el sobre entrenamiento en RNA. Los valores que puede tomar son: 0.3, 0.4, 0.5, y 0.6.

La Figura 4.9 muestra un ejemplo de qué posiciones del cromosoma proporcionado por el AGc corresponden a los diferentes hiperparámetros. El cromosoma es un vector de valores binarios, el cual es decodificado (e.g. para obtener el hiperparámetro correspondiente a las épocas, se extraen las 3 primeras posiciones del cromosoma, se codifica a decimal y así se determinará que valor corresponde a dicha variable).

Épocas	Aprendizaje	Entrenamiento	Optimizador	Activación	Tamaño del Filtro	Strides	Padding	Polling	Dropout
1	0	1	0	0	0	1	1	0	1
0	0	0	1	1	0	0	0	1	1

Figura 4.9: Codificación de los diferentes hiperparámetros en el cromosoma del AGc.

Una vez que los hiperparámetros han sido delimitados, también debemos delimitar los posibles valores de los hiperparámetros especificados anteriormente en el cromosoma. Las siguientes Tablas 4.1 y 4.2 muestran los hiperparámetros a optimizar, clasificados en Generales y Convolucionales, respectivamente.

Tabla 4.1: Hiperparámetros generales y sus posibles valores.

Parámetro	Rango
Épocas	[20, 40, 60, 80, 100, 120, 160]
Aprendizaje	[0.0001, 0.0006, 0.0011, 0.0016, 0.0021, 0.0026, 0.0031]
Entrenamiento	[0.70, 0.80, 0.90, 1.00]
Optimizador	[SGD, ADAM, RMSprop]
Activación	[relu, elu]

Tabla 4.2: Hiperparámetros convolucionales y sus posibles valores.

Parámetro	Rango
Tamaño del Filtro	[3, 4, 5, 6]
Strides	[2, 3, 4, 5]
Padding	[valid, same]
Polling	[MaxPooling2D, AveragePooling2D]
Dropout	[0.3, 0.4, 0.5, 0.6]

4.3.3. Inicialización del Componente Esclavo

En este subproceso los Componentes Esclavos son inicializados de igual forma que el Componente Maestro. Los Componentes Esclavos tienen que realizar la carga de la base de datos con la cual se realizará el entrenamiento. La base de datos utilizada para los entrenamientos es conocida como “MNIST Database of Handwritten Digits (Deng, 2012)” que fue especificada en el capítulo 2.

Los Componentes Esclavos continuarán con la construcción de la RNC utilizando topología seleccionada (que fue presentada en el punto de la Definición del modelo). Los hiperparámetros no serán asignados en este momento, debido a que serán asignados por el AGC hasta que el individuo haya sido generado.

El Componente Esclavo una vez cargado e inicializado se suscribirá a la cola de Rabbit MQ llamada *individuos* y a partir de ese momento los Componentes Esclavos estarán listos para procesar los individuos que el Componente Maestro vaya generando y enviando a la cola de *individuos*. En la Figura 4.10 se muestra una representación de los Componentes Esclavos comunicándose mediante el protocolo AMQP para el registro de los mismos.

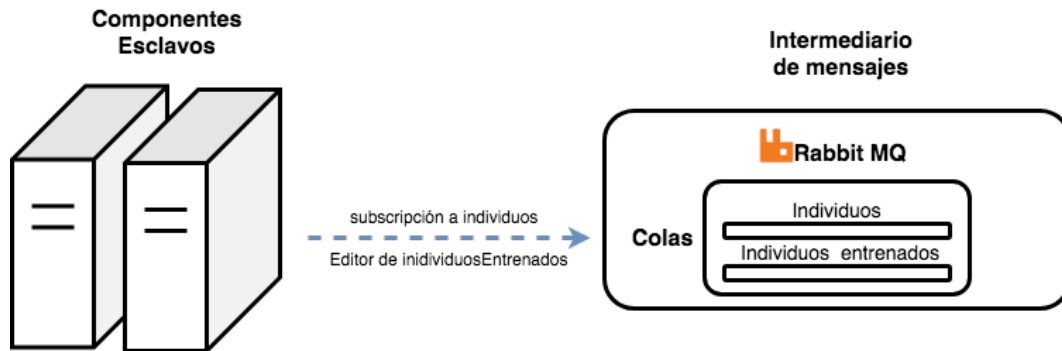


Figura 4.10: Componente Esclavo se suscribe al intermediario de mensajes de Rabbit MQ.

4.3.4. Generación de los individuos

El Componente Maestro procederá a iniciar el proceso del AGC como se indica en la línea 8 en donde se inicializa el vector de probabilidades con tamaño 19 que corresponde al número de casillas que fueron especificados en la Figura 2.9. Cada casilla del vector de probabilidades será inicializado con un valor medio el cual será .5.

Una vez inicializado el vector de probabilidades procederá a crear los individuos de la primera generación como se muestra en las línea 13 y 14 del Algoritmo 4.1.

Algoritmo 4.1 AGcD.

```
1: procedure AGCD( $n, l, g$ )  $\triangleright n$  como tamaño de población,  $l$  longitud del cromosoma
2:   Global individualTrainedOne
3:   Global individualTrainedTwo
4:   Global individualsTraining
5:   //Se subscribe a la cola de Rabbit MQ
6:   subscribeQueueIndividualTrained(recieveIndividualTrained);
7:   Inicializar el vector de probabilidad
8:   for  $i := 1$  to  $l$  do  $p[i] := 0.5$ ; do
9:     individualsTraining = True;
10:    Generar 2 individuos desde el vector
11:     $a := generate(p)$ ;
12:     $b := generate(p)$ ;
13:    sendIndividualToQueueIndividualGenerated(a);
14:    sendIndividualToQueueIndividualGenerated(b);
15:    while individualsTrained do
16:      wait();
17:    end while
18:    //Finalizado el entrenamiento se compiten los individuos
19:     $winner, loser := compete(a, b)$ ;
20:    //Actualizar el vector utilizando el mejor
21:    for  $i := 1$  to  $l$  do
22:      if  $winner[i] \neq loser[i]$  then
23:        if  $winner[i] = 1$  then
24:           $p[i] := p[i] + 1/n$ 
25:        else
26:           $p[i] := p[i] - 1/n$ ;
27:        end if
28:      end if
29:    end for
30:    // Revisar si el vector ha convergido
31:    for  $i := 1$  to  $l$  do
32:      if  $p[i] > 0$  &  $p[i] < 1$  then
33:        return to step 9;
34:      end if
35:    end for
36:  end for
37:  return  $p$ 
38: end procedure
```

La generación de los individuos se realiza con base en el vector de probabilidades como se muestra en el Algoritmo 4.2. Cada una de las casillas del individuo nuevo es asignada en función de un número aleatorio, si este número aleatorio es menor que el valor correspon-

diente a la misma casilla del vector de probabilidades se le asigna un 1, caso contrario se le asigna un 0. Por último cuando se haya realizado la asignación de cada una de las casillas se retorna el individuo nuevo.

Algoritmo 4.2 Generación del individuo.

```
1: procedure GENERATE( $v, l$ )  $\triangleright v$  es el vector de probabilidades y  $l$  la longitud del vector
2:    $individuoNuevo = [l]$ 
3:   for  $i := 1$  to  $l$  do
4:     if  $random() < v[i]$  then
5:        $individuoNuevo[i] = 1$ 
6:     else
7:        $individuoNuevo[i] = 0$ 
8:     end if
9:   end for
10:  return  $individuoNuevo$ 
11: end procedure
```

Una vez generados los individuos, serán enviados a la cola de *individuos* como se muestra en la línea 13 y 14 del Algoritmo 2.1.

4.3.5. Evaluación de los Individuos

Los Componentes Esclavos previamente suscritos a la cola *individuos*, seleccionarán un individuo y empezará la ejecución del entrenamiento de la RNC. El entrenamiento se realizará con el 80 % de los datos que contiene la base de datos. La métrica para la ejecución del entrenamiento es la *precisión*. El proceso de entrenamiento se realiza con base en el individuo generado, realizando una decodificación del individuo gen por gen y asignando cada uno de los hiperparámetros como se muestra en la Figura 4.9 previamente decrita. En caso de que el individuo genere algún valor fuera de los rangos especificados en las Tablas 4.1 y 4.2 será penalizado con un valor muy bajo para que en la siguiente generación sea excluido y no vuelva a generar ningún individuo con esas características.

Una vez que el Componente Esclavo ha terminado el entrenamiento, se realizarán las pruebas del modelo con el 20 % de los datos totales. El resultado de las pruebas retorna la precisión, que se codificará en formato JSON para ser enviadas a la cola de *individuosEntrenados* de la cual el Componente Maestro está escuchando para recibir los resultados de cada

uno de los individuos que se evaluaron.

4.3.6. Competencia de los Individuos

La competencia de los individuos se realizará una vez que los Componentes Esclavos hayan enviado los resultados de la ejecución del entrenamiento de la RNC. El Componente Maestro que se encuentra suscrito a la cola de *individuosEntrenados* se encuentra esperando los resultados, en el momento que exista un individuo en dicha cola el Componente Maestro procederá a la recepción de los individuos, la cual se realizará con el Algoritmo 4.3.

Algoritmo 4.3 Algoritmo de control de la siguiente generación.

```
1: procedure RECIEVEINDIVIDUALS(individualRecieved) ▷ individualRecieved es el
   individuo ya entrenado
2:   Global individualTrainedOne
3:   Global individualTrainedTwo
4:   Global individualsTraining
5:   Global individualsTrainingCounter
6:   Se recibe el individuo desde la cola
7:   individual = json.load(individualRecieved);
8:   contadorIndivudosEntrenados = contadorIndivudosEntrenados + 1
9:   if individualsTrainingCounter == 2 then
10:    individualTrainedTwo = individualRecieved;
11:    individualsTrainingCounter = 0
12:    individualsTrained = False
13:   else
14:    individualTrainedOne = individualRecieved;
15:   end if
16: end procedure
```

El Algoritmo 4.3 fue asociado en el momento que el Componente Maestro reciba un individuo al instante de la inicialización. El Algoritmo 4.3 se ejecutará cuando se le otorgue un individuo entrenado y este proceso se ejecute de manera asincrónica en hilo diferente al hilo principal de ejecución del programa. La Figura 4.11 representa la coordinación entre los dos hilos que se encuentran trabajando en conjunto.

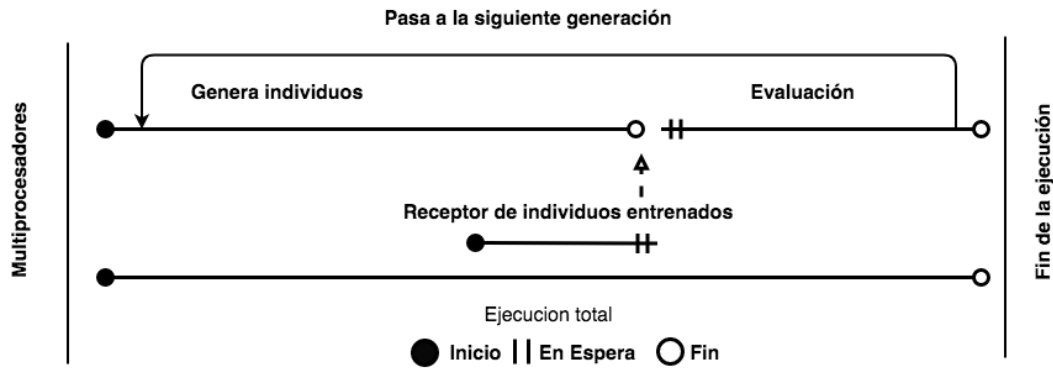


Figura 4.11: Representación del trabajo en conjunto entre los hilos Componente Maestro.

Al recibir un individuo es aceptado como texto plano, por lo tanto se debe realizar una transformación del texto plano a formato JSON como lo indica la línea 7 del algoritmo 4.3. Cuando el proceso inicie existirá una variable global, esta es la encargada de controlar si una generación ha terminado. Cuando se recibe el primer individuo de una generación se le suma un 1 como se muestra en la línea 8 del Algoritmo 4.3. Continuando con el proceso se verifica si la variable de control ya ha recibido a los dos individuo, como es el caso del primero individuo, solo se asigna el individuo a una variable de seguimiento como lo muestra la línea 14 del Algoritmo 4.3. Cuando se recibe el segundo individuo además de asignar el individuo se reinicia la variable de control del numero de individuo y se cambia a falso la variable de control de flujo como se muestra entre las líneas 9 y 12 del algoritmo 4.3. la variable de control de flujo afecta directamente el flujo del AGc, dicha variable en caso de ser modificada a *false* avanzará con el flujo como se muestra en la línea 15 del algoritmo 4.1.

La competencia de los individuos se realiza conforme a los dos individuos recibidos donde se compara su valor de aptitud como se muestra en la línea 2 del Algoritmo 4.4, en caso de que el individuo a sea mayor que el individuo b se regresa una tupla como se muestra en la línea 3 del Algoritmo 4.4 en caso contrario se regresa en orden inverso como se muestra en la línea 5 del algoritmo 4.4.

Algoritmo 4.4 Algoritmo de competencia

```
1: procedure COMPETE( $a, b$ )  $\triangleright a$  corresponde al individuo numero 1 y  $b$  corresponde al
   individuo numero 2.
2:   if  $a.fitness > b.fitness$  then
3:     return  $a, b$ ;
4:   else
5:     return  $b, a$ ;
6:   end if
7: end procedure
```

Una vez finalizada la competencia se actualiza el vector de probabilidades, recorriendo las casillas del vector de probabilidades hasta su longitud como se muestra en la línea 21 del Algoritmo 4.1. Se compara todas las casillas del individuo ganador, en caso de que sea un 1, se aumenta a la casilla correspondiente del vector de probabilidades el siguiente valor $1/n$ como se muestra en la línea 24 del Algoritmo 4.1. En caso contrario se decrementa el mismo valor como se muestra en la línea 26 del Algoritmo 4.1.

Una vez finalizada la actualización completa del vector de probabilidades se verifica si el problema ha convergido. En caso contrario vuelve a generar los individuos de la siguiente generación como se muestra en la línea 9 del Algoritmo 4.1. El proceso termina hasta que el problema haya convergido o se finalice las generaciones especificadas, entregando el modelo con mejor resultado.

4.4. Tecnologías Utilizadas

A lo largo del desarrollo del AGcD se utilizaron distintos lenguajes de programación, librerías y equipo de hardware. El desarrollo se realizó utilizando el lenguaje de programación Python por su practicidad, debido a que la curva de aprendizaje es muy suave y a la gran cantidad de librerías que existen para el área de inteligencia artificial. Dichas librerías te permiten realizar la implementación de una red neuronal convolucional muy sencilla. También se utilizaron tarjetas GPU con la finalidad de reducir aun más los tiempos de ejecución. A continuación se especificará cada lenguaje, librería y hardware utilizado en este trabajo.

4.4.1. Python

Python es un lenguaje de programación interpretado con una sintaxis muy expresiva tanto que ha sido comparado con un pseudocódigo ejecutable (Oliphant, 2007). Python es multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional, implementa tipado dinámico y es multiplataforma. A continuación enumeraremos otras cualidades del lenguaje:

1. Código abierto.
2. Sintaxis clara.
3. Interprete interactivo.
4. Gran número de módulos existentes.
5. Gran comunidad.
6. Manejador de módulos.

4.4.2. Keras

Para la implementación de la RNC en este trabajo se utilizó la librería Keras la cual es una interfaz de programación de aplicaciones (API por sus siglas en inglés) de alto nivel para RNA. Esta escrita en el lenguaje de programación Python y es capaz de correr sobre diferentes Backends como lo son TensorFlow, CNTK o Theano. Fue desarrollada enfocándose en permitir experimentación rápida, permitiendo poder ir de una idea a resultados rápidamente con el menor retraso posible lo cual es una de las claves para realizar una buena investigación (Chollet et al., 2015).

Además, minimiza el número de acciones que un usuario debe seguir para una tarea común. Está diseñada pensando en la modularidad. Un Modelo es entendido como una secuencia y son completamente configurables de manera que pueden ser unidos unos con otros (Chollet et al., 2015).

4.4.3. Tensorflow

TensorFlow es un sistema de aprendizaje automático que opera a gran escala y en ambientes heterogéneos. TensorFlow utiliza gráficos de flujo de datos para representar el cálculo, estado compartido, y las operaciones que mutan ese estado. Eso mapea los nodos de un gráfico de flujo de datos a través de muchas máquinas en un clúster, y dentro de una máquina a través de múltiples dispositivos computacionales, incluidas las CPU multinúcleo, las GPU de propósito general y los ASIC de diseño personalizado conocidos como Unidades Tensores de Procesamiento (TPUs). TensorFlow soporta una variedad de aplicaciones, con un enfoque en el entrenamiento e inferencia en redes neuronales profundas. Varios servicios de Google utilizan TensorFlow en producción, lanzado como un proyecto de código abierto, utilizado ampliamente para la investigación del aprendizaje automático (Abadi et al., 2016).

4.4.4. RabbitMQ

RabbitMQ es un intermediario de mensaje y servidor de colas de código abierto que puede ser usado para desacoplar aplicaciones compartiendo datos a través de un protocolo en común o simplificar el trabajo con colas para el procesamiento distribuido con múltiples nodos. RabbitMQ soporta múltiples protocolos de mensajería, los más importantes son *STOMP: Streaming Text Oriented Messaging Protocol* y **AMQP: Advanced Messaging Queuing Protocol** explicado anteriormente. En la Figura 4.12 se muestra un flujo de trabajo de como interactúa Rabbit MQ con los productores y consumidores de mensajería.

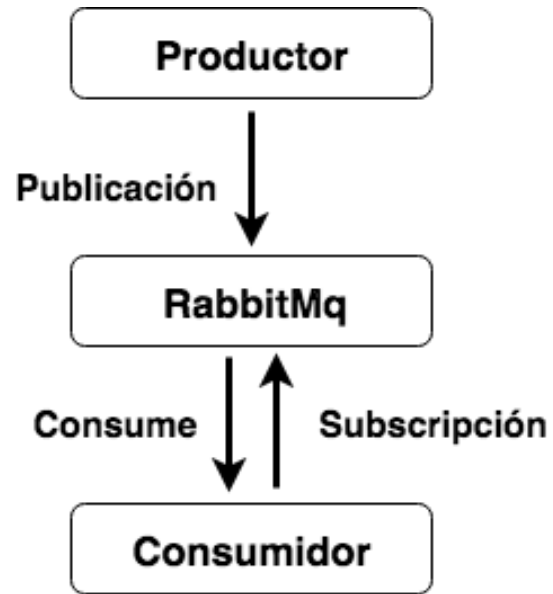


Figura 4.12: Representación de flujo de trabajo de RabbitMQ.

Capítulo 5

Pruebas y Análisis de Resultados

En esta sección se presentan los experimentos realizados con los cuales se evaluará el AGcD, también se describen los diferentes esquemas de los experimentos y configuraciones de cada uno de dichos experimentos.

Los experimentos se realizaron utilizando la base de datos “MNIST Database of Handwritten Digit” descrita en el capítulo 2. Dicha base de datos ha sido utilizada para realizar evaluaciones comparativas de diferentes métodos.

5.1. GPU vs CPU

Primeramente se efectuaron una serie de análisis comparativos entre el entrenamiento usando solo CPU y utilizando múltiples tarjetas GPU. En la Figura 5.1 se observa la gráfica comparativa entre el tiempo que toma entrenar un modelo tomando en cuenta un rango de 10 a 100 épocas usando GPU y CPU. En dicha figura es muy complicado observar los tiempos de ejecución de una RNC ejecutándose con alguna tarjeta GPU debido a los tiempos de ejecución tan altos sin el uso de una tarjeta GPU.

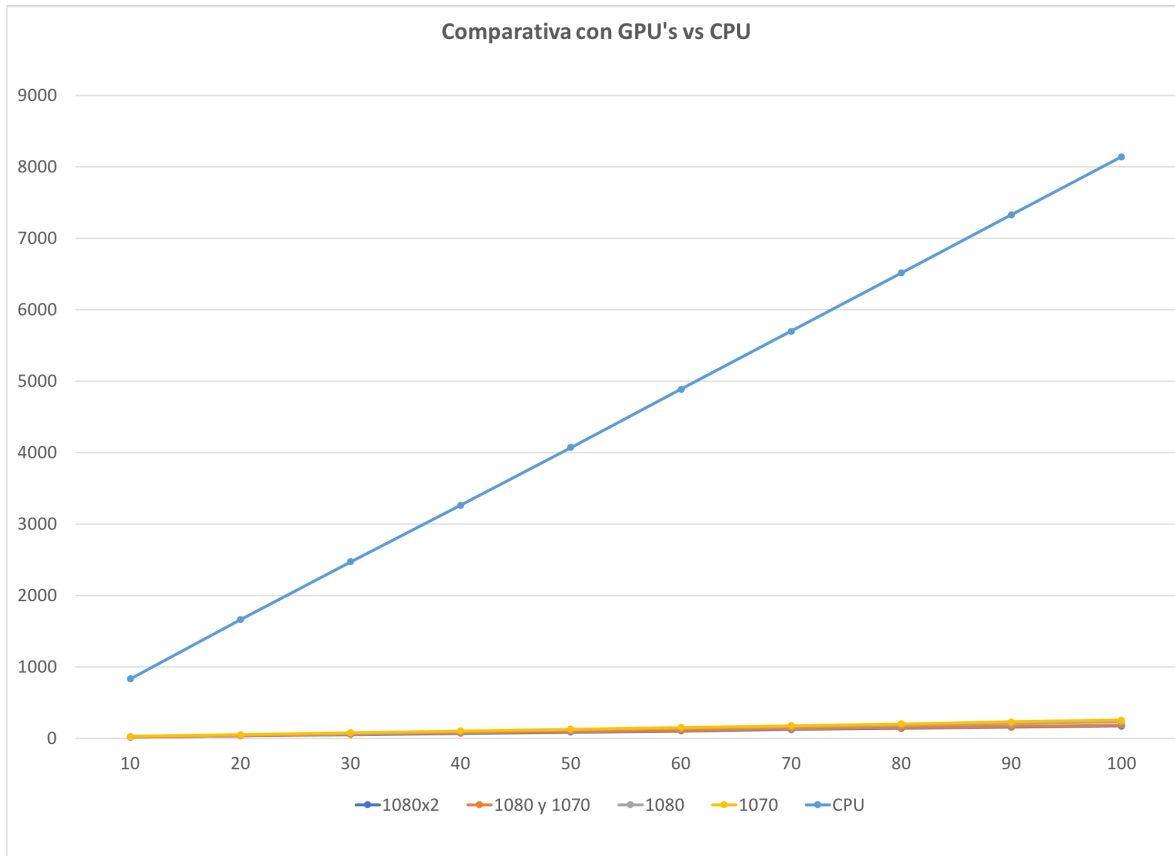


Figura 5.1: Gráfica comparativa entre el uso las diferentes tarjetas GPU y el uso de CPU.

También se realiza una comparación entre el entrenamiento de las diferentes tarjetas GPU, donde se efectúan experimentos con tarjetas GPU NVIDIA GeForce GTX 1080 y GPU NVIDIA GeForce GTX 1070. En la Figura 5.2 se presenta una gráfica comparativa entre el uso de GPU y CPU en relación al tiempo de entrenamiento, tomando en cuenta un rango de 10 a 100 épocas. En esta Figura se observa la mejora dependiendo el modelo de la tarjeta GPU, donde se puede señalar que los modelos 1080 es superior al modelo 1070.

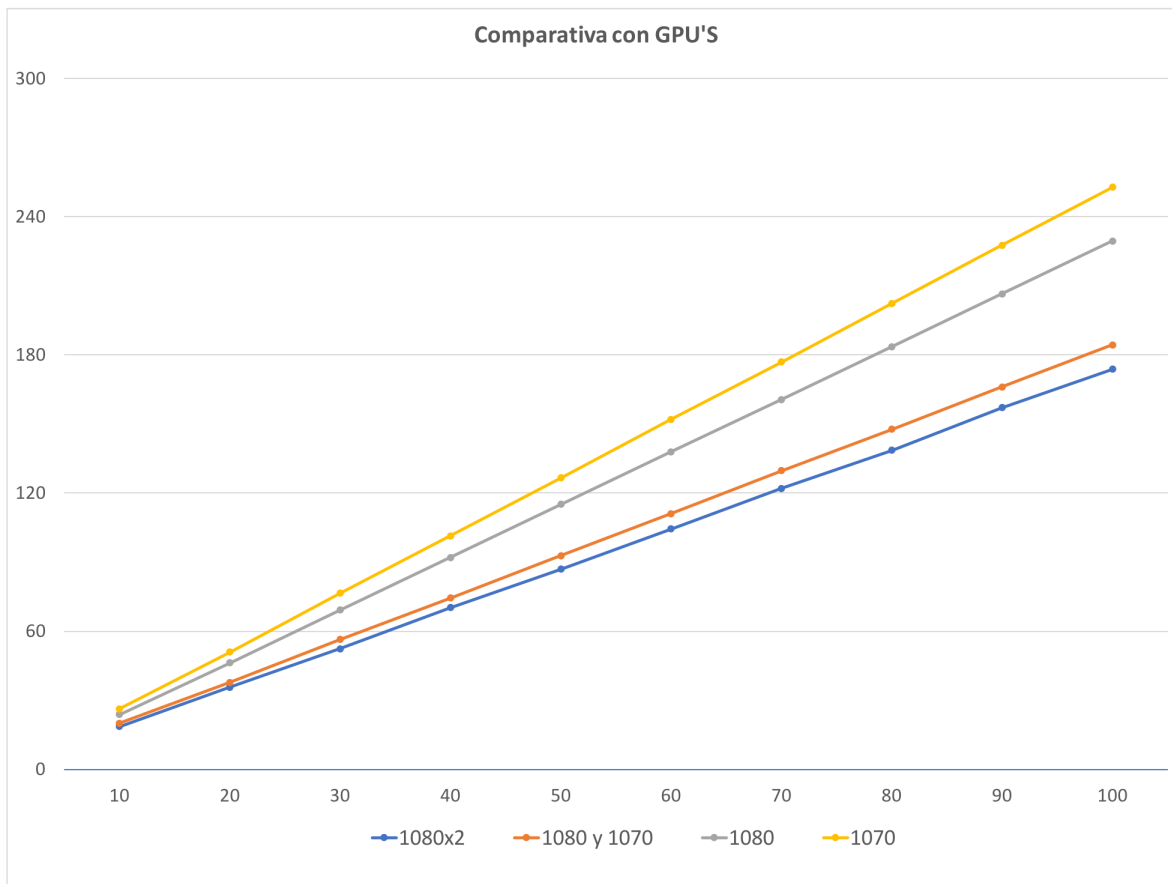


Figura 5.2: Gráfica comparativa del rendimiento de las diferentes tarjetas GPU.

5.2. Algoritmo Genético Compacto Distribuido

La configuración de los componentes asignados a cada uno de los experimentos llevados a cabo fue la siguiente:

1. Monólitico.
2. Distribuido en un Componente Maestro, un Componente Esclavo.
3. Distribuido en un Componente Maestro, dos Componentes Esclavos.

La Figura 5.3 especifica las dos configuraciones con las cuales se realizaron los experimentos para comprobar que el AGcD logró reducir los tiempos de procesamiento obteniendo la misma precisión. El Componente Maestro es un equipo con un procesador Intel Core i5 de 2.9 GHz, con 8 GB de memoria RAM DDR3 con una frecuencia de 1600 MHz y una tarjeta

GPU NVIDIA GeForce GTX 660M de 512 MB. El esclavo número uno es un equipo con un procesador Intel core i5-7500 de 3.40 GHz, con 8 GB de memoria RAM DDR3 con una frecuencia 1600 MHz y una tarjeta de video GPU NVIDIA GeForce GTX 1080 con 8 Gb en RAM. El esclavo número dos es un equipo con un procesador Intel Xenon v4 de 1.70 GHz, y 32 GB de memoria RAM DDR3 a una frecuencia 1600 MHz y una tarjeta GPU NVIDIA GeForce GTX 1080 con 8 Gb en RAM.

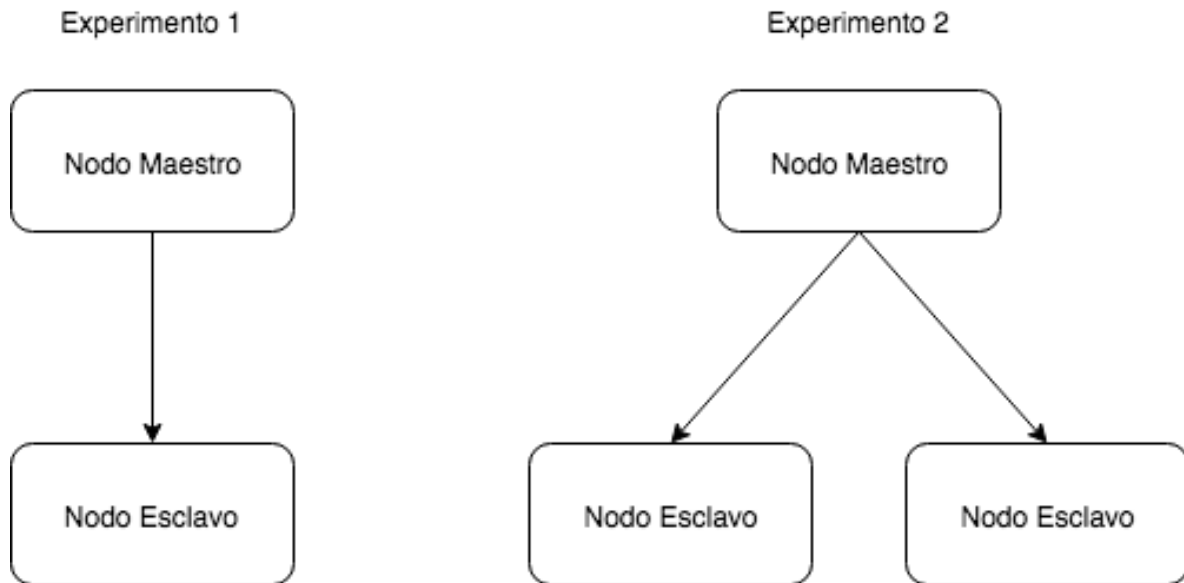


Figura 5.3: Representación de la configuración de los experimentos realizados.

5.2.1. Un Componente Maestro, un Componente Esclavo con una GPU por esclavo

A continuación se presentan los resultados obtenidos con la configuración de un esclavo y un Componente Maestro. Todos los experimentos fueron ejecutados con 100 generaciones del AGcD. En la Tabla 5.1 se observa la representación del cromosoma el cual es el mejor individuo en dicho experimento, con los tiempos de ejecución y la precisión obtenida. El tiempo promedio de ejecución es de 9 horas con 42 minutos y 59 segundos con una precisión promedio de 99.445 %.

Tabla 5.1: Resultados de los experimentos con configuración de un Componente Maestro y un Componente Esclavo.

Representación del cromosomas	Tiempo	Precisión
011 001 00 10 0 10 11 0 0 11	9:11:32	99.500 %
110 010 10 11 0 11 00 1 0 10	8:57:34	99.320 %
110 110 11 01 0 11 10 0 0 10	9:26:05	99.480 %
010 001 11 11 0 10 10 0 0 11	9:06:12	99.420 %
001 011 00 10 1 10 11 0 1 11	9:03:15	99.120 %
101 110 01 01 0 01 01 1 0 11	10:44:54	99.490 %
011 101 11 10 0 10 11 0 0 10	9:07:23	99.490 %
111 110 10 01 0 10 11 0 0 11	10:45:35	99.480 %
100 111 10 10 0 11 10 0 0 00	10:37:09	99.500 %
111 000 10 10 0 11 10 0 0 00	10:24:25	99.520 %
100 110 11 11 0 01 10 0 0 00	9:18:17	99.450 %
111 100 00 10 0 11 11 0 0 11	9:29:49	99.480 %
010 100 10 11 0 11 11 0 0 01	10:36:02	99.460 %
100 101 11 01 0 11 10 0 0 11	09:46:34	99.460 %
011 010 10 10 0 10 00 1 0 11	09:08:16	99.510 %
Promedio	09:48:52	99.445 %

También se presenta la relación del cromosoma y su decodificación con los respectivos valores de los hiperparámetros generales e hiperparámetros convolucionales para los experimentos con configuración de un Componente Maestro y un Componente Esclavo. En la Tabla 5.2 se observa la representación del cromosoma y la respectiva decodificación de los hiperparámetros generales y En la Tabla 5.3 se observan los hiperparámetros convolucionales.

Tabla 5.2: Relación del cromosoma y los valores de cada hiperparámetro general para los experimentos con un Componente Maestro y un Componente Esclavo.

Representación del cromosomas	Épocas	Aprendizaje	Entrenamiento	Optimizador	Activación
011 001 00 10 0 10 11 0 0 11	80	0.0006	0.70	ADAM	RELU
110 010 10 11 0 11 00 1 0 10	140	0.0011	0.90	RMSPROP	RELU
110 110 11 01 0 11 10 0 0 10	140	0.0031	1.00	SGD	RELU
010 001 11 11 0 10 10 0 0 11	60	0.0006	1.00	RMSPROP	RELU
001 011 00 10 1 10 11 0 1 11	40	0.0016	0.70	ADAM	ELU
101 110 01 01 0 01 01 1 0 11	120	0.0031	0.80	SGD	RELU
011 101 11 10 0 10 11 0 0 10	80	0.0026	1.00	ADAM	RELU
111 110 10 01 0 10 11 0 0 11	160	0.0031	0.90	SGD	RELU
100 111 10 10 0 11 10 0 0 00	100	0.0036	0.90	ADAM	RELU
111 000 10 10 0 11 10 0 0 00	160	0.0001	0.90	ADAM	RELU
100 110 11 11 0 01 10 0 0 00	100	0.0031	1.00	RMSPROP	RELU
111 100 00 10 0 11 11 0 0 11	160	0.0021	0.70	ADAM	RELU
010 100 10 11 0 11 11 0 0 01	60	0.0021	0.90	RMSPROP	RELU
100 101 11 01 0 11 10 0 0 11	100	0.0026	1.00	SGD	RELU
011 010 10 10 0 10 00 1 0 11	80	0.0011	0.90	ADAM	RELU

Tabla 5.3: Relación del cromosoma y los valores de cada hiperparámetro convolucional para los experimentos con un Componente Maestro y dos Componentes Esclavos.

Representación del cromosomas	Tamaño del Filtro	Strides	Padding	Pooling	Dropout
011 001 00 10 0 10 11 0 0 11	5 x 5	4	VALID	MAXPOOLING2D	0.6
110 010 10 11 0 11 00 1 0 10	6 x 6	1	SAME	MAXPOOLING2D	0.5
110 110 11 01 0 11 10 0 0 10	6 x 6	3	VALID	MAXPOOLING2D	0.5
010 001 11 11 0 10 10 0 0 11	5 x 5	3	VALID	MAXPOOLING2D	0.6
001 011 00 10 1 10 11 0 1 11	5 x 5	4	VALID	AVERAGEPOOLING2D	0.6
101 110 01 01 0 01 01 1 0 11	4 x 4	2	SAME	MAXPOOLING2D	0.6
011 101 11 10 0 10 11 0 0 10	5 x 5	4	VALID	MAXPOOLING2D	0.5
111 110 10 01 0 10 11 0 0 11	5 x 5	4	VALID	MAXPOOLING2D	0.6
100 111 10 10 0 11 10 0 0 00	6 x 6	3	VALID	MAXPOOLING2D	0.3
111 000 10 10 0 11 10 0 0 00	6 x 6	3	VALID	MAXPOOLING2D	0.3
100 110 11 11 0 01 10 0 0 00	4 x 4	3	VALID	MAXPOOLING2D	0.3
111 100 00 10 0 11 11 0 0 11	6 x 6	4	VALID	MAXPOOLING2D	0.6
010 100 10 11 0 11 11 0 0 01	6 x 6	4	VALID	MAXPOOLING2D	0.4
100 101 11 01 0 11 10 0 0 11	6 x 6	3	VALID	MAXPOOLING2D	0.6
011 010 10 10 0 10 00 1 0 11	5 x 5	1	SAME	MAXPOOLING2D	0.6

5.2.2. Un Componente Maestro, dos Componentes Esclavos con una GPU por esclavo

Los resultados obtenidos con la configuración de dos esclavos y un Componente Maestro. En la Tabla 5.4 se observa una mejoría en cuanto a tiempo de ejecución como en precisión. El promedio de tiempo de ejecución es de 6 horas 24 minutos con 11 segundos consiguiendo una mejora significativa en comparación a los métodos simples. El tiempo promedio de ejecución es de 6 horas con 17 minutos y 42 segundos con una precisión promedio de 99.443 %.

Tabla 5.4: Resultados de los experimentos con configuración de un Componente Maestro y dos Componentes Esclavos.

Representación del cromosomas	Tiempo	Precisión
100 010 11 11 0 10 01 0 0 00	06:48:52	99.510 %
100 001 01 01 0 11 01 0 0 00	06:28:07	99.490 %
110 101 00 11 0 01 10 0 0 00	06:23:53	99.480 %
011 110 10 11 0 11 00 0 0 11	06:33:21	99.470 %
111 010 11 11 0 01 11 0 0 11	05:48:26	99.470 %
100 011 01 01 0 11 01 0 0 00	06:48:38	99.450 %
101 011 00 01 0 10 01 1 0 01	06:20:52	99.460 %
110 100 10 11 0 10 10 0 0 01	06:02:53	99.510 %
110 000 01 10 0 10 01 1 0 10	06:22:39	99.490 %
111 010 01 11 0 10 01 1 0 10	06:17:21	99.485 %
100 001 01 11 0 10 01 0 0 00	06:24:51	99.500 %
011 110 10 11 0 11 00 0 0 11	06:12:45	99.490 %
100 011 01 11 1 11 01 0 0 00	05:54:13	99.200 %
111 111 10 01 0 11 01 1 0 01	06:11:31	99.485 %
011 110 10 11 0 11 11 1 1 11	05:47:09	99.150 %
Promedio	06:17:42	99.443 %

A continuación se presentan la relación del cromosoma decodificado con los respectivos valores de los hiperparámetros generales e hiperparámetros convolucionales para los experimentos con configuración un Componente Maestro y dos Componentes Esclavos. En la Tabla 5.5 se observa la representación del cromosoma y la respectiva decodificación de los hiperparámetros generales y en la Tabla 5.6 se observan los hiperparámetros convolucionales.

Tabla 5.5: Relación del cromosoma y los valores de cada hiperparámetros generales para los experimentos con un Componente Maestro y dos Componentes Esclavos.

Representación del cromosomas	Épocas	Aprendizaje	Entrenamiento	Optimizador	Activación
100 010 11 11 0 10 01 0 0 00	100	0.0011	1.00	RMSPROP	RELU
100 001 01 01 0 11 01 0 0 00	100	0.0006	0.80	SGD	RELU
110 101 00 11 0 01 10 0 0 00	140	0.0026	0.70	RMSPROP	RELU
011 110 10 11 0 11 00 0 0 11	80	0.0031	0.90	RMSPROP	RELU
111 010 11 11 0 01 11 0 0 11	160	0.0011	1.00	RMSPROP	RELU
100 011 01 01 0 11 01 0 0 00	100	0.0016	0.80	SGD	RELU
101 011 00 01 0 10 01 1 0 01	120	0.0016	0.70	SGD	RELU
110 100 10 11 0 10 10 0 0 01	140	0.0021	0.90	RMSPROP	RELU
110 000 01 10 0 10 01 1 0 10	140	0.0001	0.80	ADAM	RELU
111 010 01 11 0 10 01 1 0 10	160	0.0011	0.80	RMSPROP	RELU
100 001 01 11 0 10 01 0 0 00	100	0.0006	0.80	RMSPROP	RELU
011 110 10 11 0 11 00 0 0 11	80	0.0031	0.90	RMSPROP	RELU
100 011 01 11 1 11 01 0 0 00	100	0.0016	0.80	RMSPROP	ELU
111 111 10 01 0 11 01 1 0 01	160	0.0036	0.90	SGD	RELU
011 110 10 11 0 11 11 1 1 11	80	0.0031	0.90	RMSPROP	RELU

Tabla 5.6: Relación del cromosoma y los valores de cada hiperparámetros convolucionales para los experimentos con un Componente Maestro y dos Componentes Esclavos.

Representación del cromosomas	Tamaño del Filtro	Strides	Padding	Polling	Dropout
100 010 11 11 0 10 01 0 0 00	5 x 5	2	VALID	MAXPOOLING2D	0.3
100 001 01 01 0 11 01 0 0 00	6 x 6	2	VALID	MAXPOOLING2D	0.3
110 101 00 11 0 01 10 0 0 00	4 x 4	3	VALID	MAXPOOLING2D	0.3
011 110 10 11 0 11 00 0 0 11	6 x 6	1	VALID	MAXPOOLING2D	0.6
111 010 11 11 0 01 11 0 0 11	4 x 4	4	VALID	MAXPOOLING2D	0.6
100 011 01 01 0 11 01 0 0 00	6 x 6	2	VALID	MAXPOOLING2D	0.3
101 011 00 01 0 10 01 1 0 01	5 x 5	2	SAME	MAXPOOLING2D	0.4
110 100 10 11 0 10 10 0 0 01	5 x 5	3	VALID	MAXPOOLING2D	0.4
110 000 01 10 0 10 01 1 0 10	5 x 5	2	SAME	MAXPOOLING2D	0.5
111 010 01 11 0 10 01 1 0 10	5 x 5	2	SAME	MAXPOOLING2D	0.5
100 001 01 11 0 10 01 0 0 00	5 x 5	2	VALID	MAXPOOLING2D	0.3
011 110 10 11 0 11 00 0 0 11	6 x 6	1	VALID	MAXPOOLING2D	0.6
100 011 01 11 1 11 01 0 0 00	6 x 6	2	VALID	MAXPOOLING2D	0.3
111 111 10 01 0 11 01 1 0 01	6 x 6	2	SAME	MAXPOOLING2D	0.4
011 110 10 11 0 11 11 1 1 11	6 x 6	4	SAME	AVERAGEPOOLING2D	0.6

Al analizar los experimentos realizados en la sección GPU vs CPU se incorporaron tarjetas GPU en la ejecución de la función de los individuos del AGcD.

En este trabajo se implementó el AGcD utilizado para optimizar el entrenamiento de una RNC empleado para tareas de clasificación. Se probaron diferentes configuraciones de equipo, aumentando gradualmente el número de esclavos, donde cada esclavo tiene una tarjeta GPU.

Los resultados muestran una mejora en los tiempos de ejecución cuando se utiliza AGcD, no así en la aptitud que de acuerdo a la literatura se llegó a límite máximo reportado.

5.2.3. Un Componente Maestro, un Componente Esclavo con 2 GPU por esclavo

A continuación se presentan los resultados obtenidos con la configuración de un esclavo con 2 GPU's y un Componente Maestro. Todos los experimentos fueron ejecutados con 100 generaciones del AGcd. En la Tabla 5.7 se observa la representación del cromosoma el cual es el mejor individuo en dicho experimento, con los tiempos de ejecución y la precisión obtenida. El tiempo promedio de ejecución es de 5 horas con 55 minutos y 37 segundos con una precisión promedio de 99.490 %.

Tabla 5.7: Resultados obtenidos en experimentos con un Componente Maestro y un Componente Esclavo.

Representación del cromosomas	Tiempo	Precisión
111 010 10 11 0 10 10 0 0 11	05:56:25	99.46 %
111 000 01 11 0 01 10 0 0 00	06:14:20	99.51 %
011 111 10 11 0 11 11 0 0 11	05:37:43	99.47 %
110 011 01 11 0 11 01 0 0 00	05:45:20	99.51 %
111 110 00 01 0 10 10 1 0 10	06:01:12	99.45 %
110 011 10 11 0 10 01 0 0 00	05:09:29	99.45 %
111 111 01 01 0 01 01 0 0 00	05:32:14	99.50 %
110 101 11 11 0 01 01 0 0 00	05:26:10	99.52 %
111 010 01 11 0 11 10 0 0 10	06:21:17	99.58 %
111 101 01 11 0 11 10 0 0 10	05:58:55	99.50 %
110 100 00 10 0 11 01 0 0 11	05:34:38	99.48 %
100 111 11 11 0 10 00 0 0 00	06:11:59	99.48 %
101 000 00 10 0 11 11 1 0 11	06:30:39	99.47 %
101 001 00 01 0 11 01 0 0 10	06:32:12	99.50 %
110 000 10 10 0 11 00 0 0 10	06:01:40	99.49 %
Promedio	05:55:37	99.49 %

Ahora se presentan la relación del cromosoma decodificado con los respectivos valores de los hiperparámetros generales e hiperparámetros convolucionales para los experimentos con configuración un Componente Maestro y un Componente Esclavo con 2 GPU's por esclavo. En la Tabla 5.8 se observa la representación del cromosoma y la respectiva decodificación de los hiperparámetros generales y en la Tabla 5.9 se observan los hiperparámetros convolucionales.

Tabla 5.8: Relación del cromosoma y los valores de cada hiperparámetros generales para los experimentos con un Componente Maestro y dos Componentes Esclavos.

Representación del cromosomas	Épocas	Aprendizaje	Entrenamiento	Optimizador	Activación
111 010 10 11 0 10 10 0 0 11	160	0.0011	0.90	RMSPROP	RELU
111 000 01 11 0 01 10 0 0 00	160	0.0001	0.80	RMSPROP	RELU
011 111 10 11 0 11 11 0 0 11	80	0.0036	0.90	RMSPROP	RELU
110 011 01 11 0 11 01 0 0 00	140	0.0016	0.80	RMSPROP	RELU
111 110 00 01 0 10 10 1 0 10	160	0.0031	0.70	SGD	RELU
110 011 10 11 0 10 01 0 0 00	140	0.0016	0.90	RMSPROP	RELU
111 111 01 01 0 01 01 0 0 00	160	0.0036	0.80	SGD	RELU
110 101 11 11 0 01 01 0 0 00	140	0.0026	1.00	RMSPROP	RELU
111 010 01 11 0 11 10 0 0 10	160	0.0011	0.8	RMSPROP	RELU
111 101 01 11 0 11 10 0 0 10	160	0.0026	0.8	RMSPROP	RELU
110 100 00 10 0 11 01 0 0 11	140	0.0021	0.7	ADAM	RELU
100 111 11 11 0 10 00 0 0 00	100	0.0036	1	RMSPROP	RELU
101 000 00 10 0 11 11 1 0 11	120	0.0001	0.7	ADAM	RELU
101 001 00 01 0 11 01 0 0 10	120	0.0006	0.7	SGD	RELU
110 000 10 10 0 11 00 0 0 10	140	0.0001	0.9	ADAM	RELU

Tabla 5.9: Relación del cromosoma y los valores de cada hiperparámetros convolucionales para los experimentos con un Componente Maestro y dos Componentes Esclavos.

Representación del cromosomas	Tamaño del Filtro	Strides	Padding	Polling	Dropout
111 010 10 11 0 10 10 0 0 11	5 x 5	3	VALID	MAXPOOLING2D	0.6
111 000 01 11 0 01 10 0 0 00	4 x 4	3	VALID	MAXPOOLING2D	0.3
011 111 10 11 0 11 11 0 0 11	6 x 6	4	VALID	MAXPOOLING2D	0.6
110 011 01 11 0 11 01 0 0 00	6 x 6	2	VALID	MAXPOOLING2D	0.3
111 110 00 01 0 10 10 1 0 10	5 x 5	3	SAME	MAXPOOLING2D	0.5
110 011 10 11 0 10 01 0 0 00	5 x 5	2	VALID	MAXPOOLING2D	0.3
111 111 01 01 0 01 01 0 0 00	4 x 4	2	VALID	MAXPOOLING2D	0.3
110 101 11 11 0 01 01 0 0 00	4 x 4	2	VALID	MAXPOOLING2D	0.3
111 010 01 11 0 11 10 0 0 10	6 x 6	3	VALID	MAXPOOLING2D	0.5
111 101 01 11 0 11 10 0 0 10	6 x 6	3	VALID	MAXPOOLING2D	0.5
110 100 00 10 0 11 01 0 0 11	6 x 6	2	VALID	MAXPOOLING2D	0.6
100 111 11 11 0 10 00 0 0 00	5 x 5	1	VALID	MAXPOOLING2D	0.3
101 000 00 10 0 11 11 1 0 11	6 x 6	4	SAME	MAXPOOLING2D	0.6
101 001 00 01 0 11 01 0 0 10	6 x 6	2	VALID	MAXPOOLING2D	0.5
110 000 10 10 0 11 00 0 0 10	6 x 6	1	VALID	MAXPOOLING2D	0.5

5.2.4. Un Componente Maestro, dos Componentes Esclavos con 2 GPU por Esclavo

A continuación se presentan los resultados obtenidos con la configuración de dos esclavo con 2 GPU's por esclavo y un Componente Maestro. Todos los experimentos fueron ejecutados con 100 generaciones del AGcd. En la Tabla 5.10 se observa la representación del cromosoma el cual es el mejor individuo en dicho experimento, con los tiempos de ejecución

y la precisión obtenida. El tiempo promedio de ejecución es de 4 horas con 51 minutos y 29 segundos con una precisión promedio de 99.480 %.

Tabla 5.10: Resultados obtenidos en experimentos con un Componente Maestro y dos Componentes Esclavos con dos GPU's por esclavo.

Representación del cromosomas	Tiempo	Precisión
111 110 01 10 0 01 00 0 0 00	04:28:38	99.49 %
100 110 11 01 0 11 10 0 0 00	05:00:27	99.52 %
110 110 01 10 1 10 00 0 0 00	04:48:56	99.46 %
111 011 00 11 0 11 11 0 0 00	04:53:30	99.47 %
101 110 10 11 1 10 11 0 0 00	05:11:27	99.50 %
111 010 11 10 0 10 01 0 0 00	05:06:24	99.46 %
100 010 01 01 0 10 01 0 0 00	04:43:18	99.47 %
101 110 11 10 0 10 00 0 0 00	05:12:50	99.50 %
111 011 10 11 0 11 01 0 0 00	04:10:12	99.50 %
111 000 00 01 0 10 10 0 0 00	04:42:19	99.50 %
111 101 01 01 1 10 11 0 0 00	05:00:42	99.52 %
111 100 10 11 0 11 00 0 0 00	04:51:56	99.45 %
101 111 10 11 0 10 10 0 0 00	04:59:49	99.45 %
101 111 00 10 0 11 11 0 0 00	04:40:37	99.48 %
111 110 00 01 0 10 11 0 0 00	05:01:04	99.47 %
Promedio	04:51:29	99.48 %

A continuación se presentan la relación del cromosoma decodificado con los respectivos valores de los hiperparámetros generales e hiperparámetros convolucionales para los experimentos con configuración un Componente Maestro y dos Componentes Esclavos con 2 GPU's por esclavo. En la Tabla 5.11 se observa la representación del cromosoma y la respectiva decodificación de los hiperparámetros generales y en la Tabla 5.12 se observan los hiperparámetros convolucionales.

Tabla 5.11: Relación del cromosoma y los valores de cada hiperparámetros generales para los experimentos con un Componente Maestro y dos Componentes Esclavos.

Representación del cromosomas	Épocas	Aprendizaje	Entrenamiento	Optimizador	Activación
111 110 01 10 0 01 00 0 0 00	160	0.0031	0.80	ADAM	RELU
100 110 11 01 0 11 10 0 0 00	100	0.0031	1.00	SGD	RELU
110 110 01 10 1 10 00 0 0 00	140	0.0031	0.80	ADAM	ELU
111 011 00 11 0 11 11 0 0 00	160	0.0016	0.70	RMSPROP	RELU
101 110 10 11 1 10 11 0 0 00	120	0.0031	0.90	RMSPROP	ELU
111 010 11 10 0 10 01 0 0 00	160	0.0011	1.00	ADAM	RELU
100 001 01 01 0 01 00 0 0 00	100	0.0006	0.80	SGD	RELU
101 110 11 10 0 10 00 0 0 00	120	0.0031	1.00	ADAM	RELU
111 011 10 11 0 11 01 0 0 00	160	0.0016	0.90	RMSPROP	RELU
111 000 00 01 0 10 10 0 0 00	160	0.0001	0.7	SGD	RELU
111 101 01 01 1 10 11 0 0 00	160	0.0026	0.8	SGD	ELU
111 100 10 11 0 11 00 0 0 00	160	0.0021	0.9	RMSPROP	RELU
101 111 10 11 0 10 10 0 0 00	120	0.0036	0.9	RMSPROP	RELU
101 111 00 10 0 11 11 0 0 00	120	0.0036	0.7	ADAM	RELU
111 110 00 01 0 10 11 0 0 00	160	0.0031	0.7	SGD	RELU

Tabla 5.12: Relación del cromosoma y los valores de cada hiperparámetros convolucionales para los experimentos con un Componente Maestro y dos Componentes Esclavos.

Representación del cromosomas	Tamaño del Filtro	Strides	Padding	Polling	Dropout
111 110 01 10 0 01 00 0 0 00	4 x 4	1	VALID	MAXPOOLING2D	0.3
100 110 11 01 0 11 10 0 0 00	6 x 6	3	VALID	MAXPOOLING2D	0.3
110 110 01 10 1 10 00 0 0 00	5 x 5	1	VALID	MAXPOOLING2D	0.3
111 011 00 11 0 11 11 0 0 00	6 x 6	4	VALID	MAXPOOLING2D	0.3
101 110 10 11 1 10 11 0 0 00	5 x 5	4	VALID	MAXPOOLING2D	0.3
111 010 11 10 0 10 01 0 0 00	5 x 5	2	VALID	MAXPOOLING2D	0.3
100 001 01 01 0 01 00 0 0 00	4 x 4	1	VALID	MAXPOOLING2D	0.3
101 110 11 10 0 10 00 0 0 00	5 x 5	1	VALID	MAXPOOLING2D	0.3
111 011 10 11 0 11 01 0 0 00	6 x 6	2	VALID	MAXPOOLING2D	0.3
111 000 00 01 0 10 10 0 0 00	5 x 5	3	VALID	MAXPOOLING2D	0.3
111 101 01 01 1 10 11 0 0 00	5 x 5	4	VALID	MAXPOOLING2D	0.3
111 100 10 11 0 11 00 0 0 00	6 x 6	1	VALID	MAXPOOLING2D	0.3
101 111 10 11 0 10 10 0 0 00	5 x 5	3	VALID	MAXPOOLING2D	0.3
101 111 00 10 0 11 11 0 0 00	6 x 6	4	VALID	MAXPOOLING2D	0.3
111 110 00 01 0 10 11 0 0 00	5 x 5	4	VALID	MAXPOOLING2D	0.3

Se analizaron los diferentes resultados en el los cuales se observan en la tabla 5.13 que el uso del AGcD mejora notablemente los tiempos de ejecución del AGc y acelera la búsqueda de los modelos de la RNC hasta un 33 % utilizando dos esclavos con una tarjeta GPU. A utilizar las dos esclavos con dos tarjetas GPU cada esclavo el tiempo mejora hasta un 51 % en comparación al AGc simple ejecutado en una sola computadora con una tarjeta GPU.

Tabla 5.13: Comparativa de promedios de los experimentos realizados.

Configuración	Tiempo	Precisión
AGc simple.	09:25:37	99.458 %
AGcD con un Componente Esclavo con una GPU.	9:48:52	99.445 %
AGcD con 2 Componentes Esclavos con una GPU cada Componente Esclavo.	6:17:42	99.443 %
AGcD con un Componente Esclavo con 2 GPU.	5:55:37	99.490 %
AGcD con 2 Componentes Esclavos con 2 GPU cada Esclavo.	4:51:29	99.480 %

Capítulo 6

Conclusiones

6.1. Conclusiones

El constante uso y evolución de las redes de aprendizaje profundo han hecho que sus usuarios requieran mayor velocidad en la obtención de modelos certeros y de alta calidad, por lo cual se han explorado algunas de técnicas de optimización. Las técnicas de optimización analizadas en la búsqueda de mejores hiperparámetros son costosas tanto en poder de computo así como en el tiempo invertido para encontrar buenos resultados.

Se analizaron múltiples estrategias para la optimización tanto en tiempo como en calidad de los modelos de aprendizaje profundo. Por lo cual podemos concluir que al realizar una fusión de las técnicas analizadas es posible mejorar los tiempos de ejecución del AGc y reducir el tiempo de ejecución de una red de aprendizaje profundo utilizando GPU. Por lo tanto se utilizó una combinación de estas estrategias para tener una mejora significativa en los tiempos de ejecución del AGc la cual fue denominada como AGcD.

Además de que se propuso una arquitectura para soportar la distribución en los diferentes equipos que intervendrían en el proceso del AGcD. Esta arquitectura mostró capacidades deseables en sistemas distribuidos como: Tolerancia a fallos, Escalabilidad y poca configuración.

Para finalizar podemos concluir que con la implementación del AGcD podemos obtener una mejora significativa en los tiempos de ejecución comparado con el AGc.

6.2. Contribuciones

Las principales contribuciones de este trabajo son el planteamiento del AGcD así como la arquitectura necesaria para su implementación, con lo cual se disminuyó significativamente el tiempo de búsqueda de una configuración de hiperparámetros óptimos.

Los resultados promedio obtenidos (ver la Tabla 5.13) se han comparado con otros trabajos tales como el de Tabik et al. en el cual observamos que la precisión obtenida en este trabajo utilizando la base de datos MNIST original es muy similar e incluso superada, entonces se puede afirmar que el AGcD es comparable con el estado del arte en términos de precisión pero mejor en términos de tiempos de entrenamiento.

6.3. Consideraciones

Es importante resaltar la asignación de la configuración (individuo), ya que esta puede ser mas costosa en tiempo de ejecución, y al ser entregada al equipo con menor capacidad de procesamiento producirá que el equipo con mayor capacidad culminará la evaluación del individuo y esperará la finalización del equipo con menor capacidad y mas carga de trabajo. También es necesario que los Componentes Esclavos y el Componente Maestro se encuentren interconectados a través de una red, esta se recomienda que sea una conexión alámbrica.

Para obtener un resultado en tiempos óptimo los Componentes Esclavos necesitan tener un hardware muy similar (capacidad de procesamiento), debido a que el AGcd esperará la finalización de los dos individuos, esto quiere decir que el tiempo de ejecución de una generación dependerá del equipo con menor capacidad de cómputo.

6.4. Trabajo Futuro

Esta investigación tiene múltiples direcciones, por lo cual el trabajo a futuro puede ser muy extenso ya que se aplicaron múltiples estrategias y prácticamente cada una de dichas estrategias podrían ser mejoradas para obtener mejores resultados.

Primeramente por parte del AGcD se pueden realizar un mayor número de experimentos

aumentando el número de individuos por población y aumentando el número de tarjetas GPU en los esclavos. También se puede experimentar aumentando el número de esclavos en caso de aumentar el número de individuos.

Por otra parte se pueden realizar experimentos con diferentes conjuntos de datos como: MNIST Fashion, CIFAR-10 u otros conocidos conjuntos de datos, con la finalidad de reafirmar la efectividad de la propuesta planteada.

Para finalizar se podrían realizar pruebas de los diferentes métodos de optimización existentes y distribuir su función objetivo, utilizando la arquitectura propuesta en este trabajo.

Bibliografía

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283. 47
- Abdelhafez, A., Alba, E., and Luque, G. (2019). Performance analysis of synchronous and asynchronous distributed genetic algorithms on multiprocessors. *Swarm and Evolutionary Computation*. 23, 27
- Anderson, D. and McNeill, G. (1992). Artificial neural networks technology. *Kaman Sciences Corporation*, 258(6):1–83. x, x, 9
- Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305. 25
- Brownlee, J. (2011). *Clever Algorithms: Nature-Inspired Programming Recipes*. Lulu.com, 1st edition. 19
- Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171. 23
- Castillo, P. A., Arenas, M. G., Mora, A. M., Laredo, J. L. J., Romero, G., Rivas, V. M., and Merelo, J. (2011). Distributed evolutionary computation using rest. *arXiv preprint arXiv:1105.4971*. XI, XI, 27
- Chollet, F. et al. (2015). Keras. <https://keras.io>. 46

- Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231. 26, 27
- Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142. x, x, 15, 16, 39
- Desell, T. (2017). Large scale evolution of convolutional neural networks using volunteer computing. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 127–128. ACM. 26
- Harik, G. R., Lobo, F. G., and Goldberg, D. E. (1999). The compact genetic algorithm. *IEEE transactions on evolutionary computation*, 3(4):287–297. 19
- Harrington, P. (2012). *Machine learning in action*, volume 5. Manning Greenwich, CT. 8
- Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR. 10
- Hegde, V. and Usmani, S. (2016). Parallel and distributed deep learning. In *Tech. report, Stanford University*. 1, 27
- Kshemkalyani, A. D. and Singhal, M. (2008). *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, New York, NY, USA, 1 edition. x, x, x, x, 20, 21
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. 8, 9
- Luque, G. and Alba, E. (2011). *Parallel genetic algorithms: theory and real world applications*, volume 367. Springer. 24
- Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. (2019). Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier. 26

- Negnevitsky, M. (2005). *Artificial intelligence: A guide to intelligent systems*. 18
- Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20. 46
- Patterson, J. and Gibson, A. (2017). *Deep Learning: A Practitioner’s Approach*. O’Reilly Media, Inc., 1st edition. x, x, 1, 2, 10, 11, 13, 15, 17
- Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,. 7
- Sampedro, J. and García, J. D. (2012). Estudio y aplicación de técnicas de aprendizaje automático orientadas al ámbito médico: estimación y explicación de predicciones individuales. *EPS-UAM*. 9
- Sivanandam, S. and Deepa, S. (2008). Genetic algorithms. In *Introduction to genetic algorithms*, pages 15–37. Springer. x, x, 18
- Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959. 2, 25
- Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127. 26
- Tabik, S., Peralta, D., Herrera-Poyatos, A., and Herrera, F. (2017). A snapshot of image pre-processing for convolutional neural networks: case study of mnist. *International Journal of Computational Intelligence Systems*, 10(1):555–568. 63