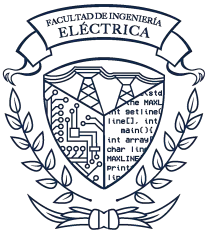


Universidad Michoacana de San Nicolás de Hidalgo

División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica

REDES NEURONALES BOROLO: UN ENFOQUE BASADO EN LA OPTIMIZACIÓN DE ESTRUCTURAS DINÁMICAS A NIVEL DE NEURONA

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

presenta

Rodrigo Israel Hernández Mazariegos

Dr. José Ortiz Bejar
Director de Tesis

Dr. Jaime Cerda Jacobo
Co-Director de Tesis

Morelia, Michoacán, noviembre 2025

Agradecimientos

Agradezco a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI) por el impulso que brinda a los avances científicos en México; sin su apoyo este trabajo no hubiera sido posible. En particular, reconozco el apoyo proporcionado a través del Proyecto CF-2023-I-1174 dentro del programa "Ciencia de Frontera 2023".

Lista de Publicaciones

Relacionados con esta investigación, se publicaron cuatro artículos de conferencia. A continuación se presenta la lista de publicaciones.

1. Huerta-Venegas, L. M., **Hernandez-Mazariegos, R. I.**, Cerda-Jacobo, J., & Ortiz-Bejar, J. (2025). *Genetic Algorithm–Based Optimization of RoBERTa for Sexism Detection*. In 2025 IEEE Central America and Panama Convention (CONCAPAN XLIII).

2. Cerda-Flores, J., Castro-Pineda, D., Juarez, M. G., **Hernandez-Mazariegos, R. I.**, Cerda-Jacobo, J., Graff, M., & Ortiz-Bejar, J. (2024). *Text Classification Based on Ensembles of Pre-Trained Models for Sarcasm Identification in Dravidian Languages*. In Forum of Information Retrieval and Evaluation FIRE-2024.

3. **Hernandez-Mazariegos, R.**, Ortiz-Bejar, J., & Ortiz-Bejar, J. (2023). *Evaluation of Heuristics for Taken's Theorem Hyper-Parameters Optimization in Time Series Forecasting Tasks*. Engineering Proceedings, 39(1), 71.

4. Cerda-Flores, J., **Hernández-Mazariegos, R.**, Ortiz-Bejar, J., Calderón-Solorio, F., & Ortiz-Bejar, J. (2023). *UMSNH at RestMex 2023: An XGBoost Stacking with Pre-Trained Word-Embeddings over Data Batches*. In IberLEF@ SEPLN.

Redes neuronales Borolo: un enfoque basado en la optimización de estructuras dinámicas a nivel de neurona

Los Miembros del Jurado de Examen de Grado aprueban la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica** de Rodrigo Israel Hernández Mazariego.

Dr. Luis Eduardo Gamboa Guzmán
Presidente del Jurado

Dr. José Ortiz Béjar
Director de Tesis

Dr. Jaime Cerda Jacobo
Co-director

Dr. Alejandro Zamora Méndez
Vocal

Dr. Eric Said Téllez Ávila
Revisor Externo

Dr. J. Aurelio Medina Rios
Jefe de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas)

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Noviembre 2025

Resumen

Las arquitecturas de redes neuronales actuales se construyen predominantemente mediante la concatenación de módulos predefinidos, como capas convolucionales o mecanismos de atención, lo cual restringe significativamente el espacio de soluciones posibles. El presente trabajo introduce las Redes Neuronales Borolo, un nuevo paradigma para el diseño y optimización de arquitecturas neuronales que opera directamente a nivel de neurona individual, trascendiendo las limitaciones impuestas por los enfoques tradicionales.

Se establece un marco teórico basado en teoría de grafos para representar y manipular redes neuronales como grafos dirigidos acíclicos, donde la topología evoluciona dinámicamente mediante la adición y eliminación selectiva de neuronas y conexiones sinápticas. El algoritmo propuesto integra mecanismos de propagación hacia adelante y retropropagación adaptados para estructuras dinámicas, junto con una función de clasificación de conexiones que combina la varianza de activaciones y la magnitud de pesos para evaluar la importancia de cada conexión.

La metodología implementa un proceso iterativo que alterna entre el crecimiento estructural, mediante la adición de neuronas completamente conectadas, y la poda de conexiones menos relevantes basada en la métrica $\sigma_i w_i$. Adicionalmente, se introduce el momento generacional para preservar la información aprendida en iteraciones anteriores mientras se permite el ajuste fino de nuevas conexiones.

La evaluación experimental se realizo sobre dos conjuntos de datos sintéticos con fronteras de decisión no lineales complejas: círculos concéntricos y espirales de tres brazos. Los resultados demuestran que Borolo alcanza clasificación perfecta (Macro F1-Score = 1.0) en ambos problemas, superando significativamente a redes neuronales densas y profundas tradicionales con cantidad similar de parámetros, las cuales obtuvieron F1-Scores inferiores a 0.64.

En conclusión, las Redes Neuronales Borolo representan un enfoque prometedor para la búsqueda automática de arquitecturas que permite explorar un espacio de soluciones mas amplio, adaptando automáticamente la complejidad estructural a los requerimientos específicos del problema con mayor eficiencia computacional.

Palabras clave: **Redes neuronales, Borolo, Optimización, Redes neuronales dinámicas, Estructura.**

Abstract

The predominant approach in neural network architecture design relies on the concatenation of pre-designed modules such as convolutional layers, recurrent units, or transformer attention mechanisms. While these modules have demonstrated strong performance, they significantly constrain the solution space by imposing structural biases based on human-conceived concepts. This thesis introduces *Borolo Neural Networks*, a novel paradigm for neural architecture optimization that operates directly at the individual neuron level, enabling greater flexibility in the search for optimal structures.

The proposed methodology represents neural networks as directed acyclic graphs, establishing a theoretical framework grounded in graph theory for the representation and manipulation of dynamic neural architectures. Forward and backpropagation algorithms are reformulated to accommodate arbitrary graph topologies, utilizing topological ordering to maintain computational efficiency through matrix operations. The Borolo algorithm implements an iterative structural evolution process that alternates between adding neurons with full connectivity and pruning connections based on a fitness function that combines synaptic weight magnitude with activation variance, effectively measuring both connection strength and information flow.

Key technical contributions include: (1) a connection classification function based on the product of weight magnitude and activation variance ($\sigma_i w_i$), which serves as a proxy for information-theoretic importance; (2) generational momentum, a technique that scales weight updates based on connection age to preserve learned representations while allowing fine-tuning; and (3) adaptive gradient clipping (AGC) combined with L2 regularization to prevent gradient explosion in potentially deep evolved architectures.

Experimental evaluation on synthetic classification datasets demonstrates the effectiveness of the approach. On a four-class concentric circles dataset, Borolo achieved perfect classification (Macro F1-Score = 1.0) with 96 trainable parameters after 3 structural iterations, while comparable dense (DNN) and deep (PNN) neural networks achieved only 0.2643 and 0.1801 respectively. On a more challenging three-arm spiral dataset, Borolo again achieved perfect classification with 247 parameters after 7 structural iterations, compared to 0.6366 for DNN and 0.1845 for PNN architectures with similar parameter counts.

The results validate the viability of neuron-level architecture optimization and demonstrate that dynamically evolved structures can significantly outperform manually designed architectures. The Borolo paradigm enables exploration of a broader solution space unconstrained by pre-conceived modular designs, offering potential for discovering more efficient and problem-specific neural network topologies.

SÍMBOLOS Y NOTACIÓN MATEMÁTICA

x_i	Entrada i-ésima de una red neuronal.
w_{ij}	Peso sináptico: peso que conecta la neurona i con la neurona j.
h	Notación general para una función de activación.
$\mathcal{L}$	Función de pérdida (general).
$\nabla\mathcal{L}$	Gradiente de la función de pérdida respecto a parámetros.
b	Sesgo (bias) de una neurona.
a	Activación (valor de salida de una neurona después de aplicar la función de activación).
z	Valor de pre-activación (suma ponderada antes de aplicar h).
δ	Error de capa (gradiente de la pre-activación durante retropropagación).
$\|\cdot\|$	Norma (magnitud de un vector o matriz).
$\odot$	Producto de Hadamard (multiplicación elemento a elemento).
h'	Derivada de la función de activación respecto a su argumento.
$\sigma_i w_i$	Notación usada como función de clasificación de conexiones (métrica combinada).
$\mathfrak{g} = (V, E)$	Grafo neuronal: vértices V (neuronas) y aristas dirigidas E (conexiones).
V	Conjunto de vértices (neuronas).
E	Conjunto de aristas (conexiones dirigidas).
G	Matriz de pesos (donde w_{ij} aparece como elemento).
G^i	Submatriz de predicción usada en propagación hacia adelante para la capa i.
G_i	Submatriz de retropropagación usada en la retropropagación para la capa i.
W^i	Matriz de pesos usada en la propagación hacia adelante para la capa i. en el algoritmo estándar (no Borolo).
X	Matriz de características (datos de entrada).
Y	Valores objetivo (etiquetas).
A	Arreglo de activaciones (vector con activaciones de todas las neuronas).
B	Arreglo de sesgos (vector con sesgos de todas las neuronas).
Z	Arreglo de pre-activaciones (valores z de todas las neuronas).
L	Arreglo de pérdidas (historial de valores de la función de pérdida).
λ	Factor de regularización (regularización L2).
τ	Razón de recorte AGC (hiperparámetro de AGC).
n_m	Número de neuronas ocultas agregadas en una iteración estructural.

SÍMBOLOS Y NOTACIÓN MATEMÁTICA (continuación)

n_e	Número de neuronas de entrada.
n_s	Número de neuronas de salida.
m_{d_s}	Distancia máxima a la salida.
m_{d_e}	Distancia máxima a la entrada.
d_e	Distancia a la entrada.
d_s	Distancia a la salida.
K	Lista K: orden topológico de neuronas.
M_i^j	Máscaras que mapean valores entre matrices de pesos.
p_b	Porcentaje de poda (fracción de conexiones a eliminar).
n_{cf}	Conexiones hacia adelante (cantidad).
n_{cb}	Conexiones hacia atrás (cantidad).
C_{g_n}	Constante de momento generacional.
t_c	Iteraciones de calentamiento (warm-up).
t_s	Número de sumandos de parada (ventana para early stopping).
n_g	Número de generaciones (edad de una conexión).
σ	Varianza de activación (usada para clasificar conexiones).
MaxIters	Máximo de iteraciones (entrenamiento en estructura estática).
MaxIterEstruct	Máximo de iteraciones estructurales.
n_{epocas}	Número de épocas.

ACRÓNIMOS

AGC	Recorte Adaptativo de Gradiente (Adaptive Gradient Clipping).
L2	Regularización L2.
DNN	Red Neuronal Densa.
PNN	Red Neuronal Profunda.
F1	*F1-Score.*
SGD	*Stochastic Gradient Descent.*
NAS	*Neural Architecture Search.*

Índice general

Capítulo 1

Introducción

1.1. Planteamiento del Problema

Dado el ímpetu con el que los investigadores se han concentrado en el aprendizaje de máquina [1], las diferentes arquitecturas se han concentrado en hacer analogías de lo que el ser humano puede hacer, por ejemplo, se concentraron en las Redes Neuronales Convolucionales intentando encontrar «bordes» (que es un concepto humano), o se concentraron en los transformadores intentando hacer que la arquitectura pudiese fijar la «atención» (lo cual también es otro concepto humano), observese que las arquitecturas actuales buscan emular los conceptos humanos sin tomar en cuenta que tal vez pueda llegar a existir una arquitectura que sin emular conceptos humanos pueda solucionar un problema dado de mucho mejor forma, es decir, no se está planteando que este mal hacerlo de la forma que se ha venido haciendo, es lo mas natural en la ciencia tratar de entender como la naturaleza hace ciertas cosas y de ahí emularlo, como por ejemplo, con la forma de volar de los pájaros emulada en los aviones, simplemente se está diciendo que no estamos completamente seguros que esa

sea la «mejor» aproximación para el fin de una red neuronal, no está demostrado que es la forma mas eficiente de hacer el trabajo que hace una red neuronal, o de otra forma, falta exploración.

Relativo a los modelos basados en redes neuronales, el enfoque predominante es llevado a cabo mediante la concatenación de diferentes módulos prediseñados [2, 3], como un módulo convolucional, un transformador o un módulo recurrente. Si bien es cierto que estos módulos han demostrado un buen rendimiento [4] y ayudan al rápido avance de la ciencia, no se puede asegurar que representen la solución óptima, ya que todo lo que se sabe es de forma empírica. Es por ello que en este trabajo se intenta optimizar la estructura de una red neuronal desvinculándose de estos módulos predefinidos y arquitecturas basadas en conceptos humanos para enfocarse en la relaciones de interacción (conexión y desconexión) entre cada par de neuronas. Esto con el fin de encontrar soluciones más eficientes y adaptadas a diferentes problemáticas.

1.2. Objetivo general

El objetivo del presente trabajo es proponer un esquema de optimización de arquitecturas de redes neuronales a nivel de neurona habilitando así una mayor flexibilidad en la búsqueda de estructuras óptimas.

- Implementación de un modelo de red neuronal basado en grafos, que permita la generación dinámica de una mayor variedad de estructuras posibles.
- Adaptar todos los métodos involucrados de entrenamiento y predicción para el modelo propuesto (propagación, retropropagación, etc), logrando así correr el modelo.
- Definir la equivalencia entre los modelos tradicionales y el modelo propuestos, que

muestre la contención de las arquitecturas tradicionales en estas nuevas arquitecturas.

- Perfilar el desempeño del modelo propuesto, comparando con estructuras neuronales tradicionales, con el fin de observar el rendimiento de este nuevo modelo de búsqueda de arquitectura.

1.3. Hipótesis

La optimización evolutiva de la topología neuronal a nivel de conexión individual permite descubrir arquitecturas dinámicas que superan en capacidad de generalización y eficiencia de parámetros a los modelos densos tradicionales, al liberar el espacio de búsqueda de las restricciones geométricas impuestas por los módulos predefinidos convencionales.

1.4. Estado del Arte

La búsqueda automática de arquitecturas (NAS) [5] se refiere a la optimización de la topología de redes neuronales (tipos de capas, conexiones y sus hiperparámetros) sin intervención humana [6, 7]. En la práctica se emplean enfoques basados en aprendizaje por refuerzo, algoritmos evolutivos y métodos diferenciables como DARTS [8] (Differentiable Architecture Search), que relaja la selección de conexiones a variables continuas. Dado el enorme espacio de búsqueda, se han desarrollado técnicas como esquemas de fidelidad adaptativa [9].

La evolución neuronal genera arquitecturas mediante algoritmos genéticos o evolutivos [10]. demostraron que es posible evolucionar modelos competitivos de gran escala a partir de redes triviales [10, 11]. Estas técnicas iteran operaciones de mutación (añadiendo o quitando capas, alterando hiperparámetros) y selección por aptitud, descubriendo modelos de alta

precisión sin diseño manual.

Las técnicas de podado eliminan pesos, neuronas o filtros suficientemente irrelevantes. Existen dos tipos principales de podado [12]: el estructurado, que elimina canales completos o capas enteras; y el no estructurado, que corta neuronas aisladas [13]. Por otro lado, en la poda dinámica se generan subredes adaptativas en tiempo de ejecución según la entrada [14]. Experimentos demuestran que es posible eliminar más del 90 % de filtros en arquitecturas convolucionales sin degradar significativamente la precisión [12, 15].

Otras técnicas como la cuantización reduce la precisión numérica de pesos y activaciones [16, 17] (por ejemplo de 32 a 8 bits, o incluso redes binarias), acelerando el cómputo y reduciendo la memoria requerida. Por otro lado, la destilación de modelos entrena modelos pequeños guiados por la salida de un modelo más grande, comprimiendo el modelo mientras retiene el desempeño [18].

A diferencia de los modelos estáticos, las redes dinámicas adaptan su estructura en función de la muestra de entrada [14]. El cómputo en estas redes a las cuales se le agregan agentes que deciden el flujo de la información, permite un equilibrio flexible entre precisión y velocidad. Por ejemplo, técnicas de early exiting o gates binarias en bloques residuales deciden dinámicamente si saltan bloques completos o no; así la red aplica esfuerzo “a demanda” según la dificultad de cada muestra.Sin embargo, siguen usando bloques predefinidos para hacer todas las operaciones.

Algunos enfoques se inspiran en la plasticidad neuronal para optimizar la arquitectura en el podado. La regla de aprendizaje «hebbiano» se ha usado para guiar dinámicamente la adición o poda de neuronas [19, 20]. Estudios recientes muestran que medir localmente la importancia de una sinapsis basada en la actividad conjunta de pre y post sinápsis permite podar más eficientemente [13, 21].

Nuestro enfoque propone alternar iterativamente el aumento de neuronas y la poda de

conexiones inútiles, observese como aquí en lugar de podar neuronas se podan conexiones, con el fin de encontrar arquitecturas óptimas compactas y eficientes.

1.5. Metodología

Se diseñó e implementó un proceso de optimización redefiniendo los algoritmos de propagación y retropropagación a nivel neurona. Adicionalmente, se muestra que las arquitecturas tradicionales están contenidas en el espacio de soluciones del modelo propuesto.
Se evaluará empíricamente, el modelo propuesto en problemas de clasificación, esto para evaluar y probar la validez y el desempeño del esquema implementado. Los resultados se contrastarán con diferentes estructuras tradicionales. Para este propósito se propondrá un prueba de validación cruzada que permita inferir resultados con validez estadística.

1.6. Justificación

En el estado del arte actual las técnicas de optimización de arquitectura son mayormente artesanales [22, 23], y las técnicas que no lo son exploran un subconjunto del espacio de soluciones posibles, con Borolo (Nuestro enfoque de optimización de arquitecturas neuronales a nivel de neurona, el termino Borolo hace referencia a la falta de orden con la cual se optimiza la estructura) se pretende proporcionar un método de optimización de arquitectura que permita explorar un conjunto mayor, haciendo posible obtener estructuras más eficientes en desempeño, complejidad y memoria.

1.7. Descripción de capítulos

En este capítulo se describen las razones que motivan este trabajo, se describe la metodología y se presenta el trabajo en general relacionado a la propuesta.

En el capítulo 2 se definen los conceptos y se desarrollan las bases teóricas sobre las que se construye el modelo de arquitecturas propuesto (Borolo).

En el capítulo 3 se presentan los métodos de propagación y retropropagación adaptados al esquema propuesto, se describen diferentes métodos necesarios para la correcta implementación de Borolo, también se muestra que el espacio de soluciones generado es más amplio que el subconjunto de soluciones explorado por lo métodos tradicionales. Adicionalmente, se esquematiza la convergencia de nuestra propuesta, y al final del capítulo se muestra un ejemplo práctico del uso de los métodos introducidos.

En el capítulo 4, se definen las criterios y estrategias de conexión/desconexión, así como la descripción del algoritmo completo para la evolución de la estructura de la red propuesta, posteriormente se muestra un ejemplo del uso de Borolo con datos.

En el capítulo 5 se presenta un análisis experimental del desempeño de Borolo, sobre ejemplos adicionales de clasificación. Los resultados son contrastados utilizando diferentes métricas y comparados con los de modelos tradicionales.

Finalmente, en el capítulo 6 se dan las conclusiones más importantes de este trabajo, y se plantean posibles líneas para investigaciones futuras.

Capítulo 2

Conceptos Generales

En el estado del arte de diseño de arquitecturas neuronales las técnicas son principalmente combinatorias [22, 24], es decir, se prueban diferentes arquitecturas heurísticamente. Las diferentes arquitecturas son construidas mediante el uso de diferentes módulos; cada arquitectura es evaluada para determinar la mejor para cada problema. Por otro lado, hay intentos que buscan optimizar arquitecturas neuronales de forma más inteligente [5, 8], como el podado, que busca quitar las neuronas poco eficientes para agilizar el algoritmo, sin embargo, en el podado el espacio de soluciones ya está limitado por el modelo inicial. La finalidad de la propuesta en este trabajo es desarrollar un algoritmo para evolucionar la estructura de una red neuronal de forma automática y a nivel de neurona/conexión. Lo anterior para proveer a la optimización de la estructura neuronal de una flexibilidad no presente en los métodos del estado del arte. Esta flexibilidad permitirá ampliar el espacio de soluciones original. Para la construcción de la propuesta es esencial que describamos las redes neuronales en terminología de teoría grafos.

2.1. Definiciones:

2.1.1. Grafo dirigido

Un grafo dirigido es un par ordenado $\mathfrak{g} = (V, E)$ de dos conjuntos, uno de nodos V y otro de aristas E, el conjunto de aristas a su vez es un conjunto de pares ordenados de nodos (a, b) donde se indica que la dirección de la arista es de a a b ($a \to b$, $a \neq b$).

Es importante notar que una red neuronal puede ser vista como un grafo. Por ejemplo, supongamos la red neuronal de la figura 2.1, se observa la estructura caracterizada por el conjunto de nodos $V = \{a, b, c, d, e, f\}$ y el conjunto de aristas $E = \{(a, c), (a, d), (a, e), (b, c), (b, d), (b, e), (c, f), (d, f), (e, f)\}$, lo cual lo vuelve un grafo, obsérvese que las aristas de este grafo tienen dirección, en este caso es la dirección en la que fluye la información durante l predicción lo que lo hace un grafo dirigido. Nótese también que este grafo no tiene ciclos, y es justo el tipo de grafos de interés puesto que no se quiere que los valores fluyan infinitamente en la estructura. Además, esto permite asignarle un orden topológico al grafo [25], por ejemplo, el nodo c va antes que el nodo f en el flujo de información.

Los nodos de los cuales no sale ninguna arista dirigida son nodos finales, en este trabajo se les llama neuronas de salida, mientras que a los nodos a los cuales no llega ninguna arista dirigida, se les denomina como neuronas de entrada, adicionalmente, a las neuronas que inciden sobre la neurona N_i las denominaremos neuronas padres de N_i, y el conjunto de las neuronas padre de N_i se referirán como P_i, similarmente, el conjunto de las neuronas sobre las cuales incide N_i se llamarán neuronas hijas de N_i y pertenecerán al conjunto H_i (conjunto de neuronas hijas de N_i). Finalmente, P es el conjunto de todas las neuronas que tienen al menos un hijo y H es el conjunto de todas las neuronas que tienen al menos un padre.

En teoría de grafos a los pesos de redes neuronales se les llama ponderación de la arista, en

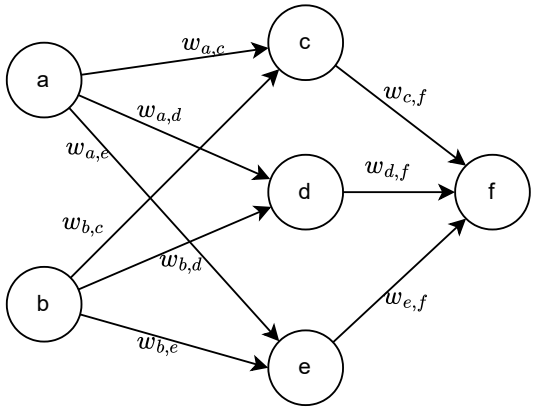


Figura 2.1: Red neuronal básica descrita como grafo mediante los conjuntos $V = \{a, b, c, d, e, f\}$ de nodos y $E = \{(a, c), (a, d), (a, e), (b, c), (b, d), (b, e), (c, f), (d, f), (e, f)\}$ de aristas.

este trabajo se les llamará simplemente peso. Algo importante respecto de los pesos en los grafos es que se pueden representar mediante la matriz de pesos G de (2.1).

$$G = \begin{pmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n} \\ w_{21} & w_{11} & w_{23} & \dots & w_{2n} \\ w_{31} & w_{32} & w_{11} & \dots & w_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nn} \end{pmatrix} \tag{2.1}$$

Donde w_{ij} es el peso que conecta la neurona i-ésima a la j-ésima, por lo que los pesos $w_{i,j}$ son 0 cuando $i = j$ o no existe una arista que vaya de la neurona i y la neurona j. Además, como el grafo es dirigido y acicliclo $w_{ij} \neq 0 \implies w_{ji} = 0$, lo cual resulta a lo más en una matriz triangular superior estricta.

También cabe mencionar que en las redes neuronales, las neuronas aplican una función

que transforma (función de activación) el valor de entrada, esta función f_n va de $\mathbb{R} \to \mathbb{R}$ y siempre recibe la suma $z_n = \sum_i f_i(x) w_{i,n}$ donde f_i es la función de activación para $N_i \in P_n$, si N_n no es neurona de entrada y x es el valor que recibe cada una de las neuronas padres de n.

Ejemplificando la matriz de pesos, la red neuronal de la figura 2.1 que es una red neuronal con dos capas densas se puede representar con la matriz G de (2.2).

$$G = \begin{pmatrix} 0 & 0 & w_{a,c} & w_{a,d} & w_{a,e} & 0 \\ 0 & 0 & w_{b,c} & w_{b,d} & w_{b,e} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{c,f} \\ 0 & 0 & 0 & 0 & 0 & w_{d,f} \\ 0 & 0 & 0 & 0 & 0 & w_{e,f} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.2}$$

Para completar la representación de la red neuronal de la figura 2.1 falta definir una lista de funciones de activación, lo cual por el momento se obviará debido a que independientemente de las funciones de activación que se elijan todas cumplen la misma función ver que tanto o que tan poco se activa una neurona, si acaso, la selección de dichas funciones de activación tienen impacto en el tiempo de optimización, en el desvanecimiento de gradiente, o en algunas otras complicaciones de la optimización de las redes neuronales, pero estructuralmente todas funcionan igual. Si se gusta ahondar mas en el tema se pueden consultar las diferentes funciones de activación en Shiv et al [26].

Es interesante notar que cualquier arquitectura tradicional puede ser representada como un grafo. Por ejemplo, una red neuronal con una capa convolucional en una dimensión con 3 entradas y un núcleo simétrico (h_1, h_2, h_1) descrita por la figura **??**, puede ser representada

mediante la matriz de pesos de la figura 2.3 y por su correspondiente lista de funciones de activación.

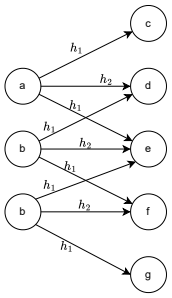


Figura 2.2: Representación mediante grafo de una capa convolucional.

$$G = \begin{pmatrix} 0 & 0 & 0 & h_1 & h_2 & h_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_1 & h_2 & h_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & h_1 & h_2 & h_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 2.3: Matriz de pesos para la capa convolucional de la figura **??**.

Dicho lo anterior surgen las preguntas ¿Qué pasaría si se agregan todas las conexiones posibles, es decir, que el grafo quede representado por una matriz triangular superior? ¿Cuál será el desempeño de los diferentes grafos en función del número de conexiones? o ¿En función de la distribución de estas?¿Hay conexiones que se puedan agregar (o quitar) que hagan más rápida la red neuronal? ¿Más precisa? Estas son las preguntas que este trabajo busca abordar.

2.1.2. Lista de neuronas

Para garantizar que nunca se generen bucles durante la construcción de la estructura del grafo, se almacenan las neuronas en una lista ordenada que representa un ordenamiento topológico. Las más cercanas a la entrada estarán a la izquierda y las más lejanas a la derecha, así, una lista ordenada para la red de la figura 2.1 podría ser $K = (a, b, c, d, e, f)$. Esto permite, por ejemplo, suponiendo que se agrega una nueva neurona n a la lista K

entre las posiciones de la neurona c y la d, conectarla de tal forma que no haya bucles; mientras se generan las conexiones aleatoriamente hacia adelante (siguiendo el flujo de información el propagación) solo con neuronas que estén a la derecha de ella, y hacia atrás (siguiendo el flujo de información durante la retropropagación) solo con neuronas que están a su izquierda. En el ejemplo de la figura 2.1 al agregar una neurona n, se agregaría al conjunto A de pares ordenados las aristas $(a, n), (c, n), (n, f)$ y al conjunto de neuronas V se agregaría n. Por su parte, K quedaría como $K = (a, b, c, n, d, e, f)$, lo cual generaría el nuevo grafo de la figura 2.4. Obsérverse que con esta estrategia, al contrario del podado, crece el grafo, es decir, se amplia el espacio de soluciones de diseño de redes neuronales, que es la finalidad de este trabajo.

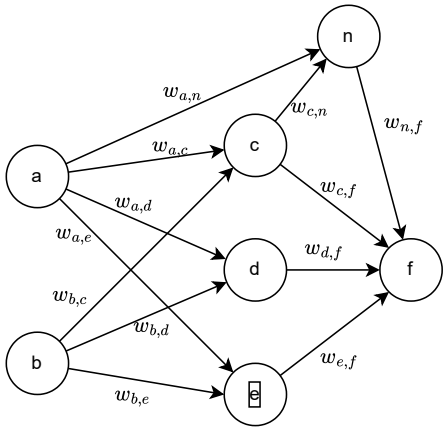


Figura 2.4: Red neuronal resultante de agregar la neurona n y las conexiones wa, n, wc, n y wn, f.

2.1.3. Distancia a la salida

Esta función nos permite calcular la distancia a la salida d_s, que nos indica que tan lejos esta cada neurona de la salida y se calcula recursivamente con (2.4).

$$d_s(N_K) = \begin{cases} 0 & \text{si } H_k = \emptyset \\ \max\{d_s(N_j)|N_j \in H_k\} + 1 & \text{si } H_k \neq \emptyset \end{cases} \tag{2.3}$$

Con esta definición de distancia a la salida, siempre es posible encontrar una neurona en P a la cual calcularle su d_s hasta agotar el conjunto de neuronas.

Sea $K = (N_1, N_2, N_3, ..., N_n)$ el ordenamiento de neuronas en la lista definida anteriormente, de un grafo dirigido acíclico con n_s neuronas de salida. Entonces, las últimas n_s neuronas de la lista ordenada K por definición su distancia es 0, por lo que ya se puede calcular la distancia de N_{n-n_s-1} puesto que todos los hijos de esta neurona están a su derecha y ya les hemos calculado la distancia, y una vez calculada la distancia N_{n-n_s-1} es posible determinar $d_{N_{n-n_s-2}}$ recursivamente.

2.1.4. Distancia a la entrada

Similar a la distancia a la salida, se define la distancia a la entrada d_e mediante la ecuación 2.4.

$$d_e(N_k) = \begin{cases} 0 & \text{si } P_k = \emptyset \\ \max\{d_s(N_k)|N_j \in P_k\} + 1 & \text{si } P_k \neq \emptyset \end{cases} \tag{2.4}$$

Por supuesto al igual que la distancia a la salida, la distancia a la entrada siempre se puede calcular para todas las neuronas, pero esta vez empezando el cálculo de izquierda a derecha

en la lista K.

2.2. Espacio de soluciones

Normalmente resulta difícil comparar la estructura de dos redes neuronales salvo mediante métricas aplicadas a sus salidas. Representar una red como un grafo facilita dicha comparación entre arquitecturas distintas. Sea el espacio de soluciones del grafo inicial

$$S_0 = \{(w_1, \ldots, w_m) \in \mathbb{R}^m\},$$

y supongamos que la solución óptima en S_0 es $\mathbf{G}$. Si añadimos una nueva conexión con peso w_{m+1}, el espacio de soluciones se amplía a

$$S_1 = \{(w_1, \ldots, w_m, w_{m+1}) \in \mathbb{R}^{m+1}\}.$$

Obsérvese que $S_0 \subset S_1$. Si extendemos $\mathbf{G}$ a S_1 fijando la nueva coordenada en cero, es decir $\mathbf{G}' = (\mathbf{G}, 0)$, entonces $\mathbf{G}'$ y la solución óptima en S_1 son directamente comparables en el mismo espacio de parámetros; la diferencia entre ambas cuantifica el efecto de introducir la conexión. Por tanto, rellenar con ceros permite comparar redes con arquitecturas distintas en un espacio paramétrico común.

En términos generales, el espacio de soluciones explorado por una ejecución de Borolo puede expresarse como

$$\mathcal{S}_{\text{Borolo}} = \bigcup_{m \in \mathbb{N}} \mathbb{R}^m = \{(w_1, \ldots, w_m) \mid w_i \in \mathbb{R},\ m \in \mathbb{N}\}.$$

En contraste, el espacio de soluciones de un algoritmo como TPOT [6], que busca la mejor

pipeline ensamblando bloques preconcebidos, puede describirse simbólicamente como

$$\mathcal{S}_{\text{TPOT}} = \{(B_1, B_2, \ldots, B_m) \mid B_i \in \mathcal{B}\},$$

donde cada B_i es un bloque predefinido perteneciente a un conjunto $\mathcal{B}$. Obsérvese que dos soluciones en $\mathcal{S}_{\text{TPOT}}$ no son directamente comparables en el espacio de configuración salvo mediante métricas de salida, a menos que se defina explícitamente una representación común que permita su comparación.

Así, mientras Borolo opera sobre un espacio paramétrico ampliable que facilita comparaciones directas entre arquitecturas, enfoques basados en bloques predefinidos requieren pasos adicionales (mapeos o embeddings) para establecer un terreno de comparación homogéneo.

2.3. Sumario

Este capítulo formalizó la representación de las redes neuronales como grafos dirigidos acíclicos y estableció la notación y herramientas necesarias para evolucionar su estructura a nivel de neurona y conexión. Se definieron los conceptos fundamentales de teoría de grafos aplicados a redes neuronales (nodos, aristas, pesos y matrices de pesos), así como la correspondencia entre arquitecturas tradicionales.

Asimismo, se introdujo la lista ordenada K, que actúa como un orden topológico para garantizar la ausencia de ciclos al modificar la arquitectura, junto con las distancias a la salida d_s y a la entrada d_e, calculables de manera recursiva y útiles para razonar sobre la posición funcional de cada neurona.

Capítulo 3

Estructura estática

Definidos los conceptos anteriores podemos construir los algoritmos de propagación y retro-propagación para la optimización de la red neuronal, así como especificar todo lo necesario para el correcto funcionamiento de esta en su forma estática.

3.1. Algoritmo de propagación

El algoritmo tradicional de propagación está definido por las ecuaciones en 3.1.

$$z^l = \begin{cases} (W^l)^T a^{l-1} + b^l & \text{si } l \neq 0 \\ (W^l)^T x + b^l & \text{si } l = 0 \end{cases} \qquad (3.1)$$
$$a^l = h^l(z^l)$$

Donde W^l es la matriz de pesos de la capa l, b^l es el vector de sesgos de la capa l, h^l es el vector de funciones de activación asociada a la capa l, z^l es el valor en la capa l previo a la aplicación de las funciones de activación, a^l es la salida del vector funciones de activación

aplicado sobre z^l y x es el vector de características.

Algo importante de esta forma de calcular la propagación hacia adelante, es la eficiencia computacional de multiplicar matrices, dado que en nuestro algoritmo nos interesa mantener esta propiedad se define el siguiente procedimiento:

1. Se ordenan las neuronas por su distancia a la salida en K de mayor a menor.
2. Se guardan las activaciones de cada neurona en un arreglo A con el orden heredado de K.
3. Los sesgos se guardan en un arreglo B también con el orden de K.

El procedimiento descrito permite emular capas, ya que se agrupan las neuronas que tengan la misma distancia a la salida. Esto garantiza que cuando el algoritmo vaya a generar la salida en la capa i todos los valores necesarios para computarla ya hayan sido calculados, es decir, las salidas de todas las capas $j < i$ ya estarán disponibles como ocurre en el algoritmo estándar de propagación hacia adelante. En la figura 3.1 es fácil ver a qué capa pertenece cada neurona. Como nota adicional, cabe mencionar que un mismo elemento de A puede servir de entrada a diferentes capas.

Si se toma el ordenamiento de capas propuesto en esta sección las ecuaciones de (3.1) se reescriben como en las ecuaciones (3.2).

$$z^i = \begin{cases} (G^i)^T A^k + B^i & \text{si } i < m_{d_s} - 1 \\ (G^i)^T x + B^i & \text{si } i = m_{d_s} - 1 \end{cases} \qquad (3.2)$$
$$A^i = h^i(z^i)$$

Tomando la matriz G de (2.1) y permutando sus renglones y columnas de tal forma que tengan el mismo orden de la lista K, es decir, en la columna k-ésima están los pesos de

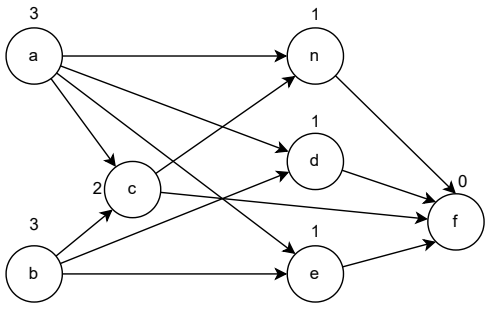


Figura 3.1: Se calculan las distancias a la salida al grafo de la figura 2.4, las neuronas a y b están a distancia 3; la c a distancia 2; las neuronas n, d y e a distancia 1 y f a distancia 0.

.

las conexiones que salen de la neurona k-ésima en K, y en la fila j-ésima están los pesos de las conexiones que llegan a la neurona j-ésima (obsérvese la matriz de (3.3) tomando en cuenta que para ese ejemplo $K = [a, b, c, d, e, n]$); una vez reordenada G de esta forma, G^i es la submatriz de G tal que solo tomamos los valores w_{kj} para los cuales N_j (neurona destino) tiene distancia i a la salida, A^k es el subarreglo de A tal que w_{kj} está en G^i, B^i es el subarreglo de B donde sólo se encuentran los sesgos de las neuronas con distancia a la salida i, y m_{d_s} es la distancia máxima a la salida. Adicionalmente, se guardan los z^i en un arreglo Z con el orden de K para ser utilizados en el algoritmo de retropropagación.

$$G = \begin{pmatrix} 0 & 0 & w_{a,c} & w_{a,d} & w_{a,e} & w_{a,n} & 0 \\ 0 & 0 & w_{b,c} & w_{b,d} & w_{b,e} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{c,n} & w_{c,f} \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{d,f} \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{e,f} \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{n,f} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.3}$$

3.2. Algoritmo de Retropropagación

El algoritmo de retropropagación busca, dada una función de perdida $\mathcal{L}$, minimizarla respecto a los pesos de las capas y los sesgos. Tradicionalmente, este problema de optimización se soluciona mediante las ecuaciones en 3.4.

$$\delta^l = \begin{cases} (W^{l+1})^T \delta^{l+1} \odot h^{l'}(z^l) & \text{si } l \neq L \\ (a^l - Y) \odot h^{l'}(z^l) & \text{si } l = L \end{cases} \tag{3.4}$$

Donde Y es el vector de los valores objetivo de nuestro conjunto de datos, L es la cantidad de capas de la red neuronal, δ^l el error en la capa l y $\odot$ es el producto Hadamard $[A \odot B]_{ij} = A_{ij} \cdot B_{ij}$. Una vez calculado el error se actualizan los pesos y sesgos mediante las ecuaciones 3.5.

$$\begin{aligned} W^l &\leftarrow W^l - \eta \delta^l (a^{l-1})^T \\ b^l &\leftarrow b^l - \eta \delta^l \end{aligned} \tag{3.5}$$

Donde η es la taza de aprendizaje.

Para mantener una metodología consistente ahora se calcula la distancia a la entrada. Lo anterior para agrupar las neuronas de tal forma que cuando se calcula cada multiplicación matricial los valores necesarios ya estén listos. Obsérvese como las capas son diferentes en la figura 3.2 respecto de la figura 3.1; es decir, para un mismo grafo hay dos redes con diferentes ordenamientos de capas, uno permite hacer la predicción (propagación), y el otro nos permite hacer la actualización (retropropagación).

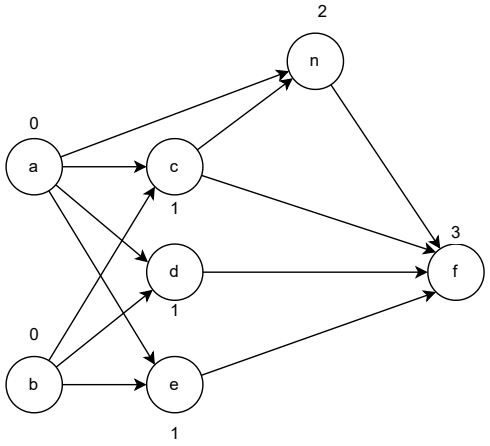


Figura 3.2: Estructura de la red neuronal resultante de calcular las distancias a la entrada para el grafo de la figura 2.4.

Con lo anterior las ecuaciones de retropropagación quedan según (3.6).

$$\begin{aligned} \delta_i &= \begin{cases} (G_i)^T \delta_{i+1} \odot h^{i'}(Z_i) & \text{si } i \neq m_{d_e} \\ (A_i - Y) \odot h^{i'}(Z_i) & \text{si } i = m_{d_e} \end{cases} \\ G_i &\leftarrow G_i - \eta \delta_i (A_{i-1})^t \\ B_i &\leftarrow B_i - \eta \delta_i \end{aligned} \tag{3.6}$$

Donde G_i es la submatriz de G tal que solo tiene los w_{jk} para los cuales N_j esta a distancia i de la entrada, Z_i es el subarreglo de Z tal que solo se toman los z_j cuyo N_j esta a distancia i de la entrada, Y son las etiquetas del conjunto de entrenamiento, y B_i es el subarreglo de B que contiene solo los sesgos de las neuronas a distancia i de la entrada. Siempre respetando el orden heredado de K. Para mantener coherencia respecto al algoritmo tradicional, se señala que δ_i es el error de la capa a distancia i de la entrada.

Nótese que las submatrices G de pesos que se requieren para la retropropagación son diferentes a las que se necesitan para la propagación hacia adelante.

3.3. Número de conexiones

Un aspecto a considerar con esta forma de ver las redes neuronales es el número de conexiones. Supóngase que existen todas las conexiones posibles en un grafo dirigido acíclico con 5 neuronas, entonces el grafo queda como en la figura 3.3 y en la matriz de la figura 3.4.

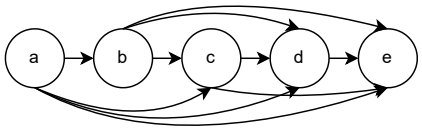


Figura 3.3: Estructura neuronal de cinco neuronas. Se incluyen todas las posibles conexiones hacia adelante (número máximo).

$$G = \begin{pmatrix} 0 & w_{a,b} & w_{a,c} & w_{a,d} & w_{a,e} \\ 0 & 0 & w_{b,c} & w_{b,d} & w_{b,e} \\ 0 & 0 & 0 & w_{c,d} & w_{c,e} \\ 0 & 0 & 0 & 0 & w_{d,e} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 3.4: Matriz de conectividad del grafo de la figura 3.3.

Tomando las neuronas de la figura 3.3, se observa que tienen las distancias a la salida $(4, 3, 2, 1, 0)$ por lo que por ejemplo la submatriz de G^1 de la matriz de la figura 3.4 quedaría como en (3.7). Es decir, se está calculando una activación a la vez, lo cual lo vuelve ineficiente, si por otro lado se usan solo las conexiones mínimas posibles el grafo queda como en la

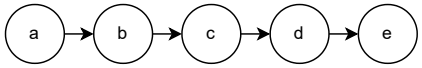


Figura 3.5: Grafo con el número mínimo de conexiones para una estructura con cinco neuronas.

$$G = \begin{pmatrix} 0 & w_{a,b} & 0 & 0 & 0 \\ 0 & 0 & w_{b,c} & 0 & 0 \\ 0 & 0 & 0 & w_{c,d} & 0 \\ 0 & 0 & 0 & 0 & w_{d,e} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 3.6: Matriz de conectividad del grafo de la figura 3.5.

figura 3.5 y la matriz de la figura 3.6, lo cual genera el mismo problema, es por ello que seleccionar un número de conexiones adecuado es crucial para un desempeño rápido de este enfoque en redes neuronales.

$$G^1 = \begin{pmatrix} w_{1,4} \\ w_{2,4} \\ w_{3,4} \end{pmatrix} \tag{3.7}$$

3.4. Máscaras

Nótese que una vez que se tiene una actualización de los pesos, esta solo se tiene en las matrices G_i, por lo que para pasarlas a las matrices G^i se usan máscaras.

La máscara M_i^j mapea los valores comunes de la matriz G_i a la matriz G^j, esta tiene 4 dimensiones y la multiplicación $M_i^j G_i$ es una contracción tensorial cuyo resultado tiene las mismas dimensiones que G^j. Si por ejemplo G_i es de 3×2 y G^j es de 4×5 entonces M_i^j será de tamaño $4 \times 5 \times 3 \times 2$ y las únicas posiciones diferentes de cero son las $M_i^j[i', j', k', l'] = 1$ tal que i', j', es la posición de w_{kl} en G^j y k', l' es la posición de w_{kl} en G_i de forma que w_{kl} es común a ambas matrices.

Finalmente se usa la convención de suma de Einstein [27] en la multiplicación $M_i^j G_i$ como

en (3.8) para obtener una matriz de dimensión 4×5.

$$(M_i^j G_i)_{i'j'} = M_{i'j'k'l'}\, G_{k'l'} \tag{3.8}$$

Aplicando el procedimiento descrito para todas las capas de predicción y de retropropagación se obtiene las máscaras necesarias para el proceso de optimización de pesos en Borolo.

3.5. Desvanecimiento y explosión de gradiente

Dado que la estructura neuronal óptima puede ser profunda se usan funciones de activación no acotadas, como la ReLU, LReLU, GELU, etc [28, 29, 30]. Esto para evitar el desvanecimiento de gradiente [31].

Por su parte, para evitar la explosión del gradiente, tomando en cuenta que las funciones de activación son no acotadas, se aplica regularización L2 y recorte adaptativo de gradiente (AGC) [32, 33].

3.5.1. Regularización L2

Esta regularización se hace en la función de pérdida. En lugar de optimizar $\mathcal{L}_{original}$ se optimiza la función descrita en (3.9) [34].

$$\mathcal{L}_{reg} = \mathcal{L}_{original} + \lambda \sum_{i,j} w_{i,j}^2 \tag{3.9}$$

Donde λ es la razón de regularización. Con este proceso la segunda expresión en (3.4) se reescribe como sigue:

$$G_i \leftarrow G_i - \eta \left(\delta^i (A_{i-1})^T + 2\lambda G_i \right) \tag{3.10}$$

Es decir, el nuevo gradiente de la función de pérdida respecto de la capa a distancia i de la entrada está dada por $\delta^i (A_{i-1})^t + 2\lambda G_i$. Por supuesto el 2 en el nuevo gradiente puede ser perfectamente absorbido por λ. Observe que L2 tambien trata de mantener los pesos los más pequeños posibles.

3.5.2. Recorte adaptativo de gradiente (AGC)

Si por alguna razón los pesos se inicializan lejos de su punto óptimo, L2 no evitará la explosión de gradiente, en esos casos es cuando AGC [33] tiene utilidad.

AGC se ejecuta por capas, lo que busca es que si la norma del gradiente de la capa es muy grande respecto a los pesos de ésta, entonces se mantiene la dirección del gradiente, pero se le da la norma de los pesos.

Suponga que $\nabla_i \mathcal{L}$ es el gradiente de la función de pérdida respecto a la capa que tiene distancia i a la entrada se calcula como en (3.11).

$$\nabla_i \mathcal{L}_{recortado} \leftarrow \min \left(1, \frac{\tau \|G_i\|}{\|\nabla_i \mathcal{L}\|} \right) \cdot \nabla_i \mathcal{L} \tag{3.11}$$

Donde τ es la razón de corte.

3.6. Algoritmo de estructura estática.

Habiendo definido todo lo anterior, se puede definir el algoritmo para la estructura estática. Para ello se supone un conjunto de datos entrenamiento y se quiere construir una red neuronal con n_e neuronas de entrada y n_s neuronas de salida.

Se definen los valores para los hiperparámetros de la tabla 3.1 para luego iniciar el proceso descrito en el algoritmo 1

Tabla 3.1: Parámetros del algoritmo estático

Parámetro	**Descripción**
τ	Parámetro τ de AGC
η	Tasa de aprendizaje
λ	Parámetro λ de L2
n_m	Número de neuronas ocultas
n_{cf}	Número de conexiones hacia adelante de cada neurona oculta
n_{cb}	Número de conexiones hacia atrás de cada neurona oculta
t_c	Número de iteraciones de calentamiento
t_s	Número de sumandos de la condición para terminar el proceso de optimización
$MaxIters$	Máximo número de iteraciones
$BatchSize$	Tamaño de lote

Algoritmo 1 Algoritmo de estructura estática

1: Se conectan todas las neuronas de entrada con todas las neuronas de salida.
2: Se inserta una a una las n_m neuronas ocultas como se explicó en la sección 2.1.2, con número de conexiones n_{cf} y n_{cb} hacia adelante y hacia atrás respectivamente.
3: Se crean las capas G^i (3.2) para predecir, y G_i (3.6) para la retropropagación.
4: Se crean las máscaras M_i^j para pasar los valores de las capas de retropropagación a las capas de predicción.
5: Se dividen los datos en lotes con el tamaño de lote deseado.
6: Se declara un arreglo L para las pérdidas.
7: **for** $iter$ in $0 : MaxIters$ **do**
8: $X, Y \leftarrow$ siguiente lote
9: Se usa X en las ecuaciones 3.2 para obtener Y_p
10: Se calculan los gradientes de los pesos y los sesgos utilizando 3.6
11: Se actualizan los pesos de predicción usando las máscaras M_i^j
12: Se calcula $\mathcal{L}_i$
13: $L \leftarrow \text{append}(L, \mathcal{L}_i)$
14: **if** $iter > t_c$ AND $\sum_{i=\#L-2t_s}^{\#L-t_s} L_i < \sum_{i=\#L-t_s}^{\#L} L_i$ **then**
15: **break**
16: **end if**
17: **end for**

3.7. Ejemplos

Para ilustrar el funcionamiento del algoritmo estático, se utiliza un conjunto de datos sintético bidimensional con dos clases. Se generan 64 datos (32 por clase) utilizando una distribución normal (la función *make_blobs* disponible en *sklearn* [35]). Los hiperparámetros de la estructura se fijan como sigue: $n_e = 2$, $n_s = 2$, $\tau = 0.01$, $\lambda = 0.001$, $\eta = 0.1$, $n_m = 10$, $n_{cf} = 2$, $n_{cb} = 2$. Para el ajuste del modelo se pasan los 64 ejemplos al mismo tiempo, es decir, cada lote es de tamaño 64 y por lo tanto cada iteración de lote se corresponde con una época, se ejecutan 800 épocas.

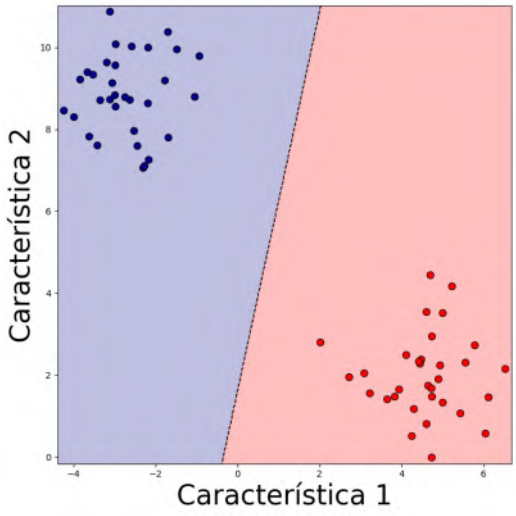
Los resultados obtenidos se observan en la figura 3.7 los cuales muestra una clasificación correcta de los datos. Ademas, se tienen curvas de pérdida y precisión consistentes con las curvas observadas en redes neuronales tradicionales.

Como segundo ejemplo, se repite el experimento con un conjunto de datos que no es linealmente separable. Se generar dos clases usando la función *make_moons* [35]. Utilizando los mismos hiperparámetros de la red, a excepción del número de épocas que ahora se fija en 2500 para asegurar convergencia. Los resultados obtenidos se muestran en la figura 3.8 donde se ve que el enfoque también funciona para conjuntos que no son linealmente separables, lo cual es el comportamiento esperado del modelo.

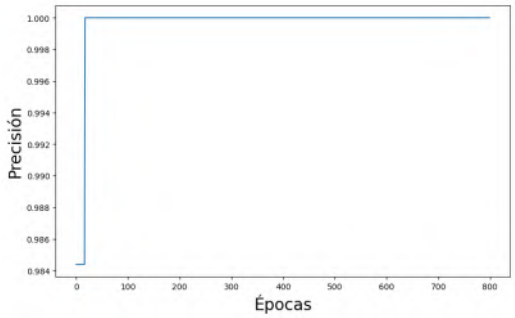
Como se puede observar el modelo ajusta de forma correcta los datos. Es importante mencionar que en este punto la intención es solo mostrar que el enfoque descrito si optimiza los parámetros de modelo.

3.7.1. Perfilado del número de neuronas vs conexiones

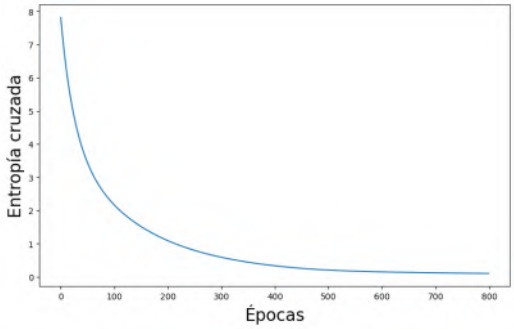
Para perfilar el comportamiento del número de conexiones y neuronas; se considera $n_c = n_{cf} = n_{cb}$ y se define una malla. La búsqueda exhaustiva se realiza sobre la malla definida



(a) Resultado

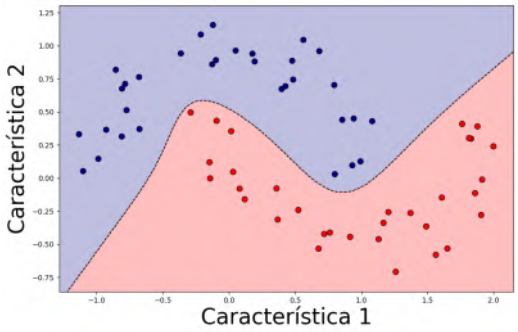


(b) Precisión

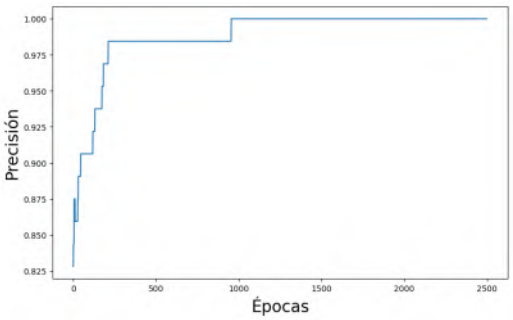


(c) Entropía cruzada

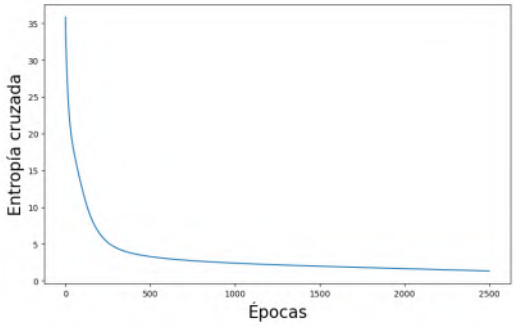
Figura 3.7: Resultados obtenidos utilizando una estructura estática con hiperparámetros $n_e = 2$, $n_s = 2$, $\tau = 0.01$, $\lambda = 0.001$, $\eta = 0.1$, $n_m = 10$, $n_{cf} = 2$, $n_{cb} = 2$ sobre un conjunto sintético bidimensional con dos clases.



(a) Resultado



(b) Precisión



(c) Entropía cruzada regularizada

Figura 3.8: Resultados obtenidos utilizando una estructura estática sobre un conjunto de datos con dos clases no linealmente separables.

por $n_c = 2, 4, 6, 8, 10, 12, 14, 16, 18$; $n_m = 1, 150, 300, 450, 600, 750, 900, 1050, 1200$. Para cada configuración de la malla, se ejecuta el algoritmo propuesto durante 100 épocas, cada una sobre una muestra de 64. Lo anterior permite observar el comportamiento ilustrado en la figura 3.10. Observese que conforme aumenta el número de conexiones no aumenta demasiado el tiempo de ejecución (se aprecia una tendencia lineal) como si lo hace el aumento de neuronas (posiblemente cuadrático). Se aprovecha este comportamiento para la definición del proceso de evolución descrito en la sección 4.1.1.

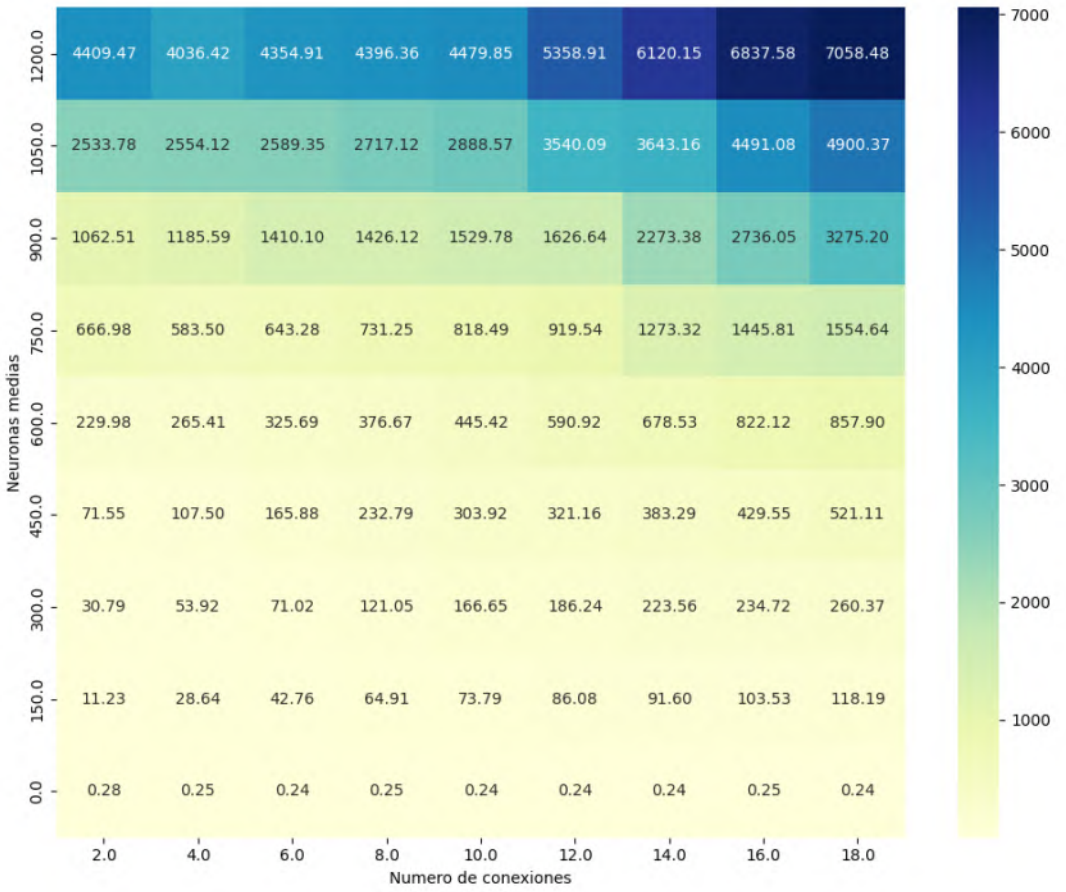
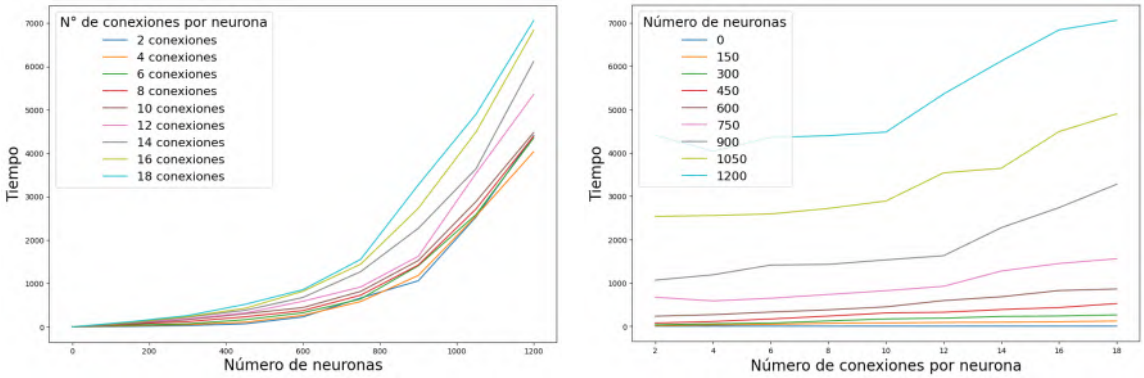


Figura 3.9: Mapa de calor que ilustra el tiempo de convergencia de una estructura estática como función del número de neuronas medias y el número de conexiones.



(a) Aumentando las conexiones (b) Aumentando las neuronas

Figura 3.10: Comparativa de como el tiempo de convergencia dependes más del aumentado en número de neuronas que del número de conexiones.

Una vez demostrada la funcionalidad de nuestro modelo, se puede abordar el punto central de este trabajo, que es la búsqueda de estructuras óptimas de redes neuronales para diferentes problemas.

3.8. Sumario

En este capítulo se vieron las adaptaciones necesarias a las funciones básicas de las redes neuronales para poder trabajarlas en términos de grafos. Se reescribieron las funciones de retropropagación y propagación, y se definen las máscaras que permiten transferir los pesos entre las do arquitectura neuronales. Además se describe la adaptación a Borolo de técnicas para enfrentar el desvanecimiento y explosión de gradiente. Finalmente concluimos con ejemplos ilustrativos del funcionamiento del modelo sobre datos sintéticos.

Capítulo 4

Borolo

Para conseguir la optimización de la estructura, se deben definir los mecanismos de evolución de la misma. Para ello se modifica el algoritmo propuesto para la estructura estática. En particular se definen procesos para agregar neuronas y clasificar las conexiones en la estructura.

4.1. Estrategia de modificación de la estructura

Dada la matriz G del grafo, las máscaras para esta matriz definidas en en la sección 3.4 y las capas a las cuales se aplican estas máscaras son la base de la estructura estática.
Una **iteración estructural** está acotada al tiempo en el cual se agregan y/o quitan neuronas a una estructura estática. Entre dos iteraciones estructurales sucesivas, se requiere la reconstrucción de las capas del grafo resultante. Una vez reconstruidas, se procede a la optimización de los pesos de esta nueva estructura estática.

4.1.1. Agregar neuronas

Para el proceso de agregar neuronas se definen 3 nuevos hiperparámetros: el primero indica el número de neuronas a agregar en cada iteración, y dos más que definen la cantidad de conexiones a agregar hacia adelante y hacia atrás en cada neurona nueva.

Aunque que en el algoritmo estático ya se tenía un hiperpárametro para las neuronas medias que se agregan. Sin embargo, como nos interesa no incrementar el número de hiperpárametros [36, 9] se puede prescindir de n_m en la estructura estática igualándolo a 0 y resignificando n_m como el número de neuronas a agregar entre cada iteración estructural. Por lo que al inicio la estructura solo consta de una capa densa y conforme va avanzando Borolo la complejidad de las operaciones avanza con este.

Por otro lado, los hiperparámetros de las conexiones de cada neurona nueva se pueden mantener justo como se definen para el algoritmo estático; otra opción es conectar de forma porcentual; o bien con todas las neuronas posibles hacia adelante y todas las neuronas posibles hacia atrás, permitiendo así explorar una mayor cantidad de rutas de la información. La última opción es la que se considera apropiada, esto para continuar con la política de mantener el número de parámetros pequeño. Además de que, como se puede apreciar en la figura 3.9, el tiempo de cómputo crece más al agregar neuronas que al agregar conexiones.

4.1.2. Función de clasificación de conexiones

Una vez agregadas las neuronas y habiendo entrenado el nuevo modelo por un breve periodo, es necesaria una forma de clasificar las conexiones agregadas, esto para seleccionar solo las mejores conexiones y así ir construyendo un modelo solo con las más relevantes.

Dada la importancia de esa clasificación es necesario definir una función que permita graduar las conexiones agregadas. Primero, se desea que las conexiones seleccionadas tengan un

peso relativamente grande, puesto que se puede suponer que un peso pequeño indica que la información que cruza esta conexión tiene relativamente poca importancia para la neurona de llegada. Con lo anterior un primer intento de función de clasificación es simplemente graduar las conexiones por la magnitud de sus pesos. Sin embargo, si se considera la situación de una neurona que envía un valor constante, es decir, que independientemente de la clase que este computando en ese momento siempre manda el mismo valor, esto hace que esta neurona actúe como el sesgo de sus neuronas hijas. Pues bien, estas conexiones también serán poco útiles independientemente de que su peso sea alto. Así pues, es importante, que aunque las magnitudes de los pesos sean grandes, se integre a la función una forma de distinguir la cantidad de información útil que cada conexión aporta. Es en este punto donde la entropía toma un papel relevante.

Considérense los siguientes puntos:

- Las funciones de activación no se pueden observar como una función de distribución de probabilidad, porque rara vez suman 1.
- Si se quisiera calcular directamente la entropía de las neuronas para cada función de activación tendríamos que normalizarla de formas diferentes, lo que haría una función de normalización para cada función de activación, lo cual es costoso.

Ahora bien, suponiendo una distribución gaussiana la entropía se calcula como $H = \frac{n}{2}\ln(2\pi e) + \frac{1}{2}\ln|\Sigma|$ [37] donde n es la cantidad de las variables aleatorias. Observe que $H(\Sigma_1) > H(\Sigma_2) \iff |\Sigma_1| > |\Sigma_2|$, por lo que si queremos usar H para clasificar, basta con usar $|\Sigma|$ además como las neuronas son univariadas entonces $|\Sigma| \to |\sigma| = \sigma$, es decir, podemos usar la varianza de la salida de la neurona para clasificar la cantidad de información que esta provee.

Lo que acabamos de ver es que para una variable aleatoria con función de densidad gaussiana univariada, la varianza está fuertemente relacionada con la entropía, ya que la entropía es una función monótona creciente respecto de la varianza. Esta propiedad, aunque solo la ejemplificamos con la distribución gaussiana, no es exclusiva de ella; la distribución uniforme o la distribución de Laplace también lo cumplen, y varias distribuciones más [38, 39]. Es decir, aunque no todas, una gran parte de las funciones de distribución de probabilidad cumplen que la entropía crece conforme crece la varianza, si bien no sabemos que distribución tendrá cada neurona, si sabemos que la varianza es una buena primera aproximación para clasificar la importancía de una neurona, con el algoritmo 2. Con el subsecuente ejemplo quedará mas claro.

Algoritmo 2 Cálculo de la varianza

1: Se guardan las activaciones de la neurona durante el entrenamiento, etiquetándolas con su clase.
2: Se calcula el promedio de activación en la neurona por clase, por lo que tendremos un valor para cada clase.
3: Se calcula la varianza de esta distribución de valores usando $\frac{\sum_i |a_i - \bar{a}|}{N}$.

En la figura 4.1 se muestra un ejemplo numérico del uso del algoritmo 2 para dos clases. Con esta varianza finalmente podemos ponderar las diferentes conexiones usando la ecuación 4.1.

$$\sigma_i w_i \tag{4.1}$$

donde σ_i es la varianza de la neurona de la que viene la conexión, y w_i es el peso de la conexión.

Siguiendo con el ejemplo de la figura 4.1 tenemos la tabla de 4.2 con la cual ya se pueden

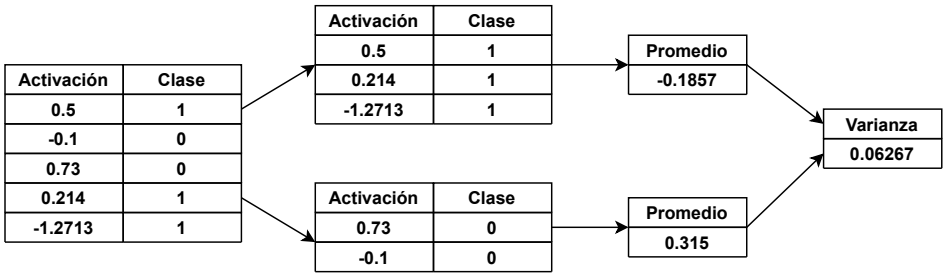
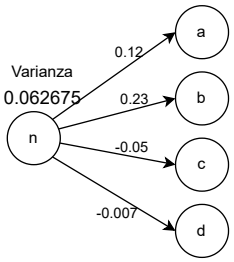


Figura 4.1: El procedimiento para el cálculo de la varianza en una conexión neuronal consiste en promediar los valores de activación de la conexión, agrupados por clase, y luego calcular la varianza de estos valores medios.

clasificar las conexiones, si por ejemplo se quieren desechar el 75 % de las conexiones (a este hiperparámetro le llamaremos p_b), nos quedaríamos solamente con las conexión (n, b).



(a) Varianza y pesos

Conexión	Ponderación
(n, a)	$0.062 * 0.12 = 0.0075$
(n, b)	$0.062 * 0.23 = 0.014$
(n, c)	$0.062 * 0.05 = 0.0031$
(n, d)	$0.062 * 0.007 = 0.00043$

(b) Ponderación de conexiones

Figura 4.2: Cada conexión se pondera como el producto de la varianza y el peso, en base esto se define un umbral para la poda de conexiones del grafo.

Una vez filtradas las conexiones, se verifica que todas las neuronas ingresadas tengan al menos una conexión de entrada y al menos una conexión de salida, las que no cumplan este requisito también se eliminan.

4.1.3. Momento generacional

Con el objetivo de mantener la información aprendida en iteraciones estructurales pasadas, se define el **momento generacional**, se busca que los pesos ya entrenados se ajusten de forma limitada. Esto puede entenderse como si en la primera iteración estructural se entrena la parte más gruesa de la clasificación, y una vez entrenada se fijase, se le agregase el siguiente grupo de neuronas, el cual tiene por objeto aprender los ajustes más finos de la clasificación, y así siguiésemos sucesivamente manteniendo la estructura aprendida mientras se controlan detalles cada vez más finos.

Sin embargo, también se quiere dar cierta libertad de entrenamiento, por si cuando se le agrega el siguiente grado de complejidad, este requiere que se mueva un poco el grado anterior.

Lo anterior se soluciona introduciendo el momento generacional. Hay varias formas de definir el momento, en este trabajo se hace escalando la actualización por el factor definido en la ecuación 4.2.

$$\frac{1}{C_{g_n}^{n_g-1}} \tag{4.2}$$

Donde C_{g_n} es una constante que va entre $[1, \infty)$ y n_g es es el número de generaciones que ha existido cada neurona, es decir las conexiones que estamos entrenando en la iteración estructural actual i-ésima su momento generacional será de 1, las conexiones que se filtraron en la iteración estructural $(i-1)$-ésima pertenecen a la generación 2, las de la iteración $(i-2)$-ésima pertenecen a la generación 3, etc.

4.2. Algoritmo Borolo

Dado un conjunto de datos, con una matriz de características X y un vector de salidas y para entrenar un clasificador Borolo. Se inicia con la selección de los hiperparámetros de la tabla 4.1.

Tabla 4.1: Hiperparámetros de Borolo

Parámetro	**Descripción**
τ	Parámetro τ de AGC
η	Tasa de aprendizaje
λ	Parámetro λ de L2
n_m	Número de neuronas agregadas por iteración estructural
t_c	Iteraciones de calentamiento para el algorimto estático
t_s	Sumandos para detener algritmo estático
$n_{\text{épocas}}$	Cantidad de épocas por iteración estructural máxima
$MaxIters$	Máximo número de iteraciones por estructura
MaxIterEstruct	Número máximo de iteraciones estructurales
p_b	Porcentaje de conexiones a eliminar por estructura
Batch_Size	Tamaño de lote
C_{g_n}	Base del momento generacional

Una vez definidos los hiperparámetros, se divide el conjunto de datos en $\mathcal{E}$ (Entrenamiento) y $\mathcal{V}$ (Validación). Finalmente, se aplica el proceso de optimización estructural descrito en el algoritmo 3.

Algoritmo 3 Algoritmo Borolo

1: Se conectan todas las neuronas de entrada con todas las neuronas de salida.
2: **for** $iter$ in $0 : MaxIters$ **do**
3: Se divide $\mathcal{E}$ en $\mathcal{E}_1$, $\mathcal{E}_2$ y $\mathcal{E}_3$ aleatoriamente, en la figura 4.3 se muestra visualmente la utilidad de esta división de datos.
4: Se agregan n_m neuronas.
5: Se optimiza los pesos de la estructura estática actual usando $\mathcal{E}_1$ y $\mathcal{E}_2$
6: Se podan las conexiones con 4.1 sobre $\mathcal{E}_3$
7: Se eliminan las conexiona más deficientes por debajo de $p_b\,\%$.
8: Se verifica que todas las neuronas tengan al menos una entrada y una salida, las que no se eliminan.
9: **end for**

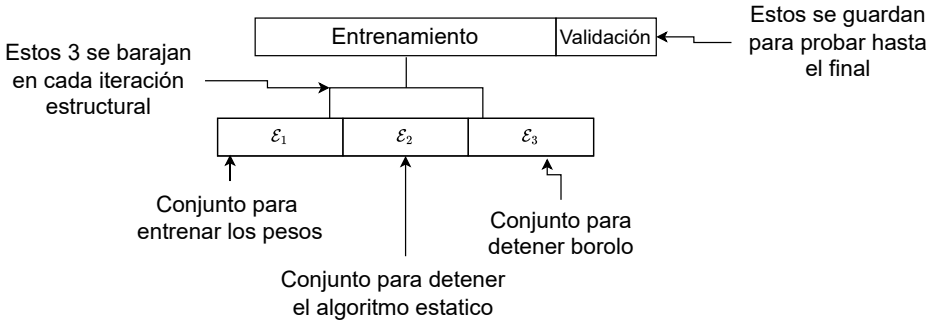


Figura 4.3: Los conjuntos $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$ funciona como conjuntos de entrenamiento, validación y prueba para cada estructura estática.

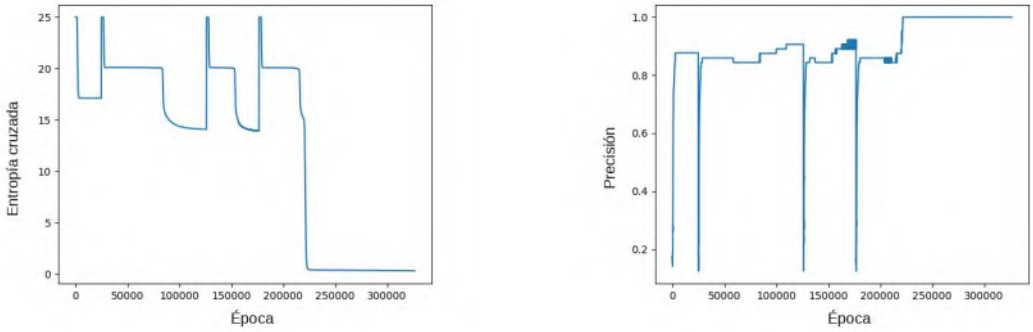
Para ilustrar el uso del algoritmo se aplica al conjunto de datos sintéticos *make_moons* de *scikit-sklearn* [35] con un 0.1 % de ruido. Los hiperparámetros son los de la tabla 4.2.

Tabla 4.2: Parámetros del modelo para ejemplificar Borolo

Parámetro	**Valor**	**Parámetro**	**Valor**
τ	0.01	$n_{\text{épocas}}$	150,000
η	0.1	MaxIterEstruct	3
λ	0.001	p_b	0.2
n_m	2	Batch_Size	64
t_s	50	C_{g_n}	1
		batches_calentamiento	25,000

Nótese como el hecho de que $C_{g_n} = 1$ implica que nuestro momento generacional es desactivado. Adicionalmente los conjuntos $\mathcal{E}_1$, $\mathcal{E}_2$, y $\mathcal{E}_3$ son el mismo conjunto, esto para agilizar la convergencia.

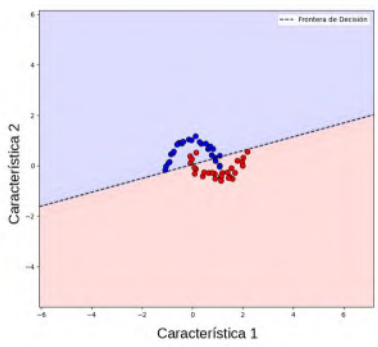
Los resultados obtenidos se muestran en la figura 4.4. Observese como en la entropía cruzada de la figura 4.4a se notan las diferentes iteraciones estructurales, cada vez que hay una iteración estructural la entropía cruzada se dispara aunque en este ejemplo la acotamos a 25, en esta gráfica se pueden ver claramente las 4 estructuras que se trabajaron durante el entrenamiento, por su parte en la figura 4.4b se puede observar como mientras sea aumenta la complejidad de la red, la precisión del algoritmo también aumenta. Finalmente, la figura 4.5 muestra las fronteras de decisión de cada una de las estructuras estáticas.



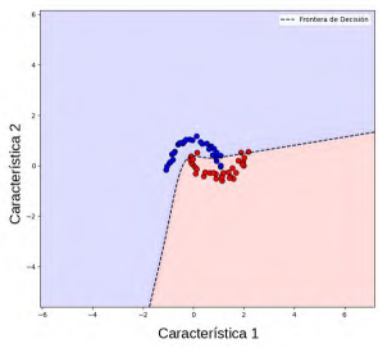
(a) Entropía cruzada

(b) Precisión

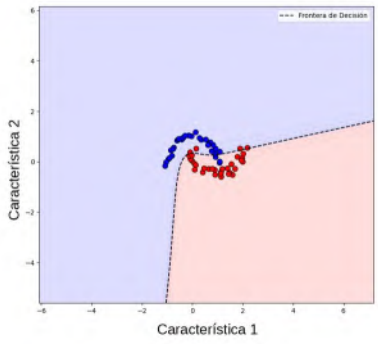
Figura 4.4: Evolución de la entropía cruzada y la precisión para las cuatro estructuras estáticas necesarias para la convergencia Borolo usando el conjunto de datos no linealmente separable (lunas).



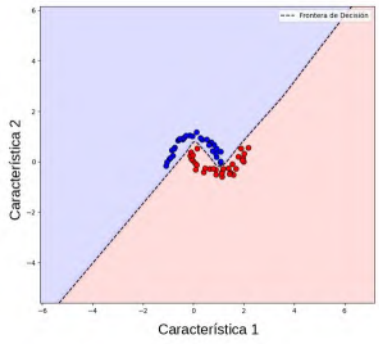
(a) Estructura 1



(b) Estructura 2



(c) Estructura 3



(d) Estructura 4

Figura 4.5: Fronteras de decisión para las cuatro estructuras estáticas necesarias para la convergencia Borolo usando el conjunto de datos no linealmente separable (lunas).

Capítulo 5

Experimentos

Este capítulo presenta la evaluación experimental del algoritmo Borolo mediante dos conjuntos de datos sintéticos con características geométricas complejas. Adicionalmente, se realiza una comparación con redes neuronales tradicionales. Esto permite validar la capacidad de la metodología propuesta para aprender fronteras de decisión no lineales complejas, así como comparar su rendimiento con redes neuronales convencionales.

5.1. Metodología experimental

Para cada experimento se siguieron los siguientes pasos:

1. **Partición de datos:** Los conjuntos de datos se dividieron en una proporción 80/20, donde el 80 % se destinó al entrenamiento y el 20 % restante se reservó para la evaluación final.

2. **Entrenamiento del algoritmo Borolo:** Se aplicó el algoritmo propuesto con hiperparámetros específicos para cada conjunto de datos.

3. **Construcción de redes de comparación:** Se diseñaron dos arquitecturas de redes neuronales con un número similar de parámetros entrenables al de nuestra propuesta:
    - **DNN:** Red neuronal densa con una única capa oculta.
    - **PNN:** Red neuronal profunda con múltiples capas ocultas densas.
4. **Evaluación:** Se utilizó Macro F1-Score como métrica de evaluación [40] en el conjunto de prueba, esto para garantizar una comparación objetiva del rendimiento.

Todas las redes neuronales utilizan como algoritmo de optimización SGD [41] (Stochastic Gradient Descent), esto con el fin de que las diferentes aproximaciones sean comparables, así como para mantener consistencia con el método de optimización empleado por Borolo.

5.2. Experimento 1: Círculos concéntricos

5.2.1. Descripción del conjunto de datos

Para el primer experimento se utiliza un conjunto de datos sintético compuesto por 1000 puntos distribuidos en 4 círculos concéntricos[42]. Este conjunto presenta un desafío debido a la naturaleza no lineal de las fronteras de decisión.

5.2.2. Configuración experimental

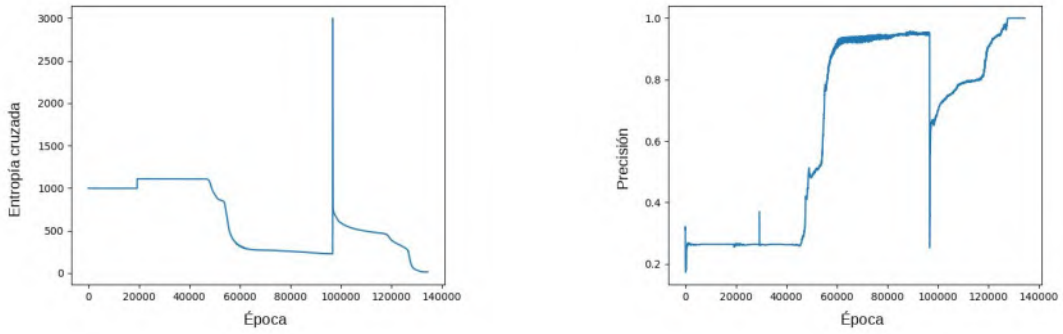
Los hiperparámetros utilizados para el algoritmo Borolo son los siguientes:
Es importante destacar que se estableció $C_{g_n} = 1$ y se utilizaron los conjuntos $\mathcal{E}_1 = \mathcal{E}_2 = \mathcal{E}_3$ para la evaluación del crecimiento estructural.

Tabla 5.1: Parámetros del modelo

Parámetro	Valor	Parámetro	Valor
τ	0.01	$n_{\text{épocas}}$	1,000,000
η	0.1	MaxIterEstruct	15
λ	0.001	p_b	0.2
n_m	3	Batch_Size	800
t_s	100	C_{g_n}	1
		batches_calentamiento	10,000

5.2.3. Resultados del entrenamiento

en que conjuntos estamos poniento los resultados

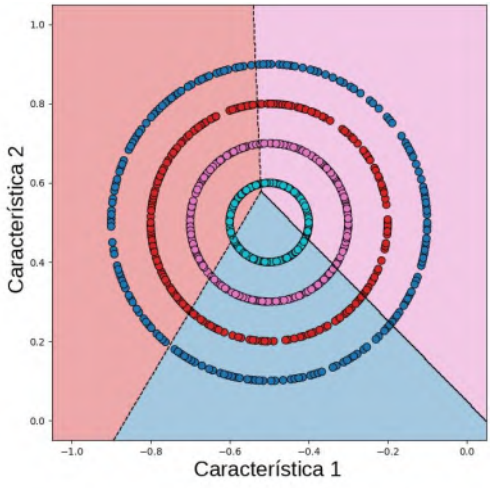


(a) Entropía cruzada (b) Precisión

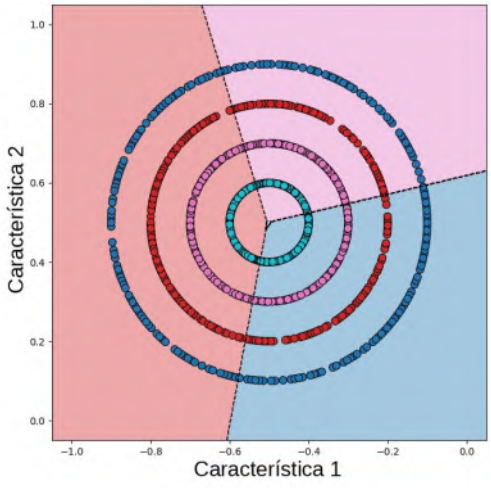
Figura 5.1: Evolución de las métricas de entrenamiento de Borolo en el conjunto de círculos concéntricos

La Figura 5.1 presenta la evolución de las métricas durante el entrenamiento. La Figura 5.1a muestra la disminución progresiva de la entropía cruzada, mientras que la Figura 5.1b ilustra el incremento correspondiente en la precisión, confirmando que el algoritmo converge satisfactoriamente.

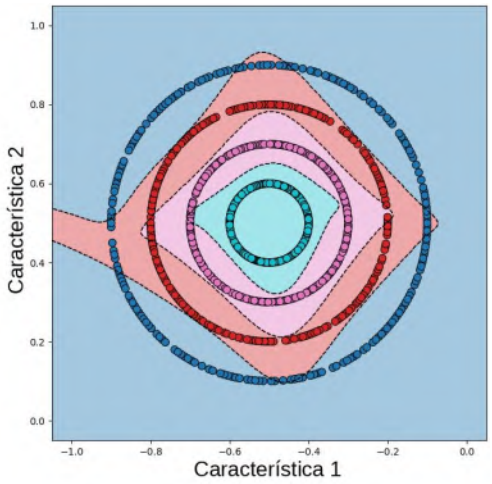
El proceso de crecimiento estructural se visualiza en la Figura 5.2, donde se observa la evolución de las fronteras de decisión a través de las diferentes iteraciones estructurales.



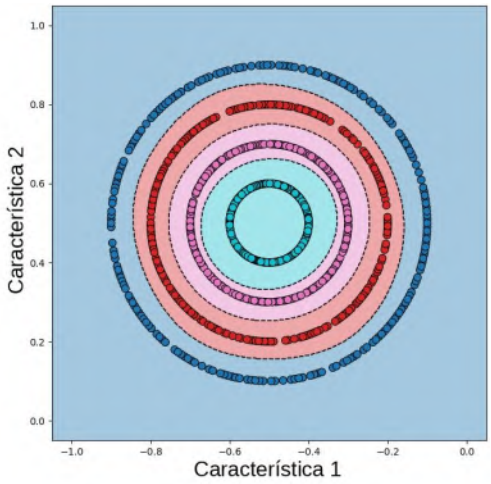
(a) Iteración estructural 1



(b) Iteración estructural 2



(c) Iteración estructural 3



(d) Estructura final

Figura 5.2: Evolución de las fronteras de decisión durante el crecimiento estructural para el conjunto de círculos concéntricos

5.2.4. Análisis de la convergencia estructural

El algoritmo Borolo requirió únicamente 3 iteraciones estructurales para converger, generando una arquitectura final con 96 parámetros entrenables. Esto desmuestra la capacidad del algoritmo para adaptar automáticamente su complejidad a los requisitos del problema.

5.2.5. Comparación con arquitecturas convencionales

Para establecer una comparación objetiva, se diseñaron dos redes neuronales con complejidades similares:

- **DNN:** Red neuronal con una capa densa de 30 neuronas, resultando en 214 parámetros entrenables.
- **PNN:** Red neuronal profunda con 7 capas ocultas densas de 5 neuronas cada una, totalizando 219 parámetros entrenables.

5.2.6. Resultados de evaluación

Los resultados de la evaluación en el conjunto de prueba se presentan en la tabla 5.2. La métrica Macro F1-Score proporciona una evaluación equilibrada del rendimiento de clasificación considerando todas las clases por igual.

Tabla 5.2: Resultados de Macro F1-Score para el conjunto de círculos concéntricos

Algoritmo	**Macro F1-Score**
Borolo	1.0000
DNN	0.2643
PNN	0.1801

Los resultados demuestran que algoritmo Borolo supera a las arquitecturas tradicioneales

evaluadas. La propuesta alcanza una clasificación perfecta (F1-Score = 1.0), mientras que las redes neuronales convencionales evaluadas quedan por debajo de 30 %.

5.3. Experimento 2: Espirales

5.3.1. Descripción del conjunto de datos

El segundo experimento empleó un conjunto de datos más desafiante, compuesto por 312 puntos distribuidos en una espiral de 3 brazos[42]. Este conjunto representa un problema de clasificación particularmente complejo debido a la naturaleza entrelazada de las clases.

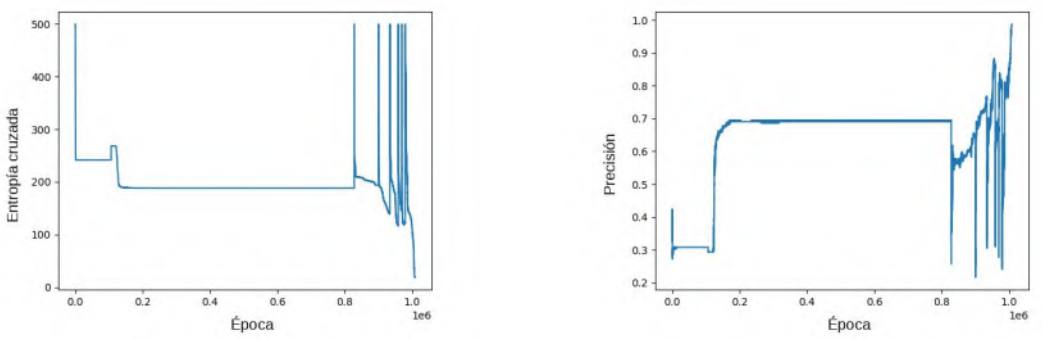
5.3.2. Configuración experimental

Los hiperparámetros utilizados fueron similares al experimento anterior, con las siguientes adaptaciones:

Tabla 5.3: Parámetros del modelo

Parámetro	**Valor**	**Parámetro**	**Valor**
τ	0.01	$n_{\text{épocas}}$	1,000,000
η	0.1	MaxIterEstruct	15
λ	0.001	p_b	0.2
n_m	3	Batch_Size	249
t_s	100	C_{g_n}	1
		batches_calentamiento	10,000

El tamaño del lote se ajustó a 249 para adaptarse al del conjunto de entrenamiento.



(a) Entropía cruzada (b) Precisión

Figura 5.3: Evolución de las métricas de entrenamiento de Borolo en el conjunto de espirales

5.3.3. Resultados del entrenamiento

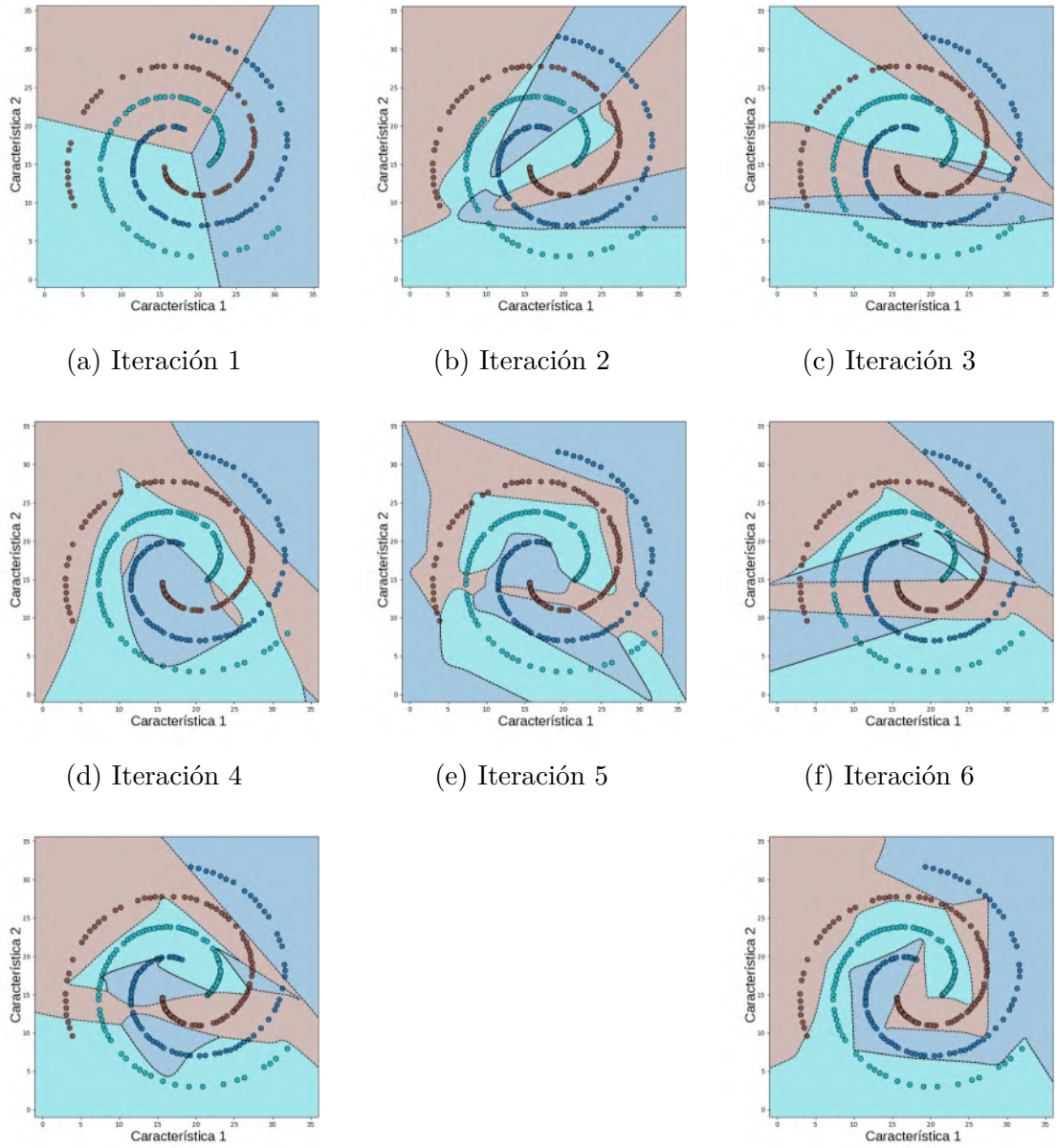
La Figura 5.3 ilustra el proceso de convergencia del algoritmo. A pesar de la mayor complejidad del problema, se observa una convergencia apropiada, tanto en la entropía cruzada (Figura 5.3a) como en la precisión (Figura 5.3b).

5.3.4. Análisis de la convergencia estructural

Para este problema más complejo, el algoritmo requirió 7 iteraciones estructurales, generando una arquitectura final con 247 parámetros entrenables. Este incremento en la complejidad estructural refleja la adaptabilidad del algoritmo a problemas de mayor dificultad.
La Figura 5.4 muestra la progresión completa del crecimiento estructural, mostrando cómo el algoritmo genera gradualmente fronteras de decisión cada vez más sofisticadas.

5.3.5. Comparación con arquitecturas convencionales

Las redes neuronales de comparación se ajustaron para mantener una complejidad similar:



(a) Iteración 1 (b) Iteración 2 (c) Iteración 3

(d) Iteración 4 (e) Iteración 5 (f) Iteración 6

(g) Iteración 7 (h) Estructura final

Figura 5.4: Evolución de las fronteras de decisión durante el crecimiento estructural para el conjunto de espirales

- **DNN:** Red neuronal con una capa densa de 41 neuronas, resultando en 249 parámetros entrenables.
- **PNN:** Red neuronal profunda con 9 capas ocultas densas de 5 neuronas cada una, totalizando 273 parámetros entrenables.

5.3.6. Resultados de evaluación

Tabla 5.4: Resultados de Macro F1-Score para el conjunto de espirales

Algoritmo	**Macro F1-Score**
Borolo	1.0000
DNN	0.6366
PNN	0.1845

Los resultados presentados en la tabla 5.4 confirman nuevamente la efectividad del algoritmo Borolo.

5.4. Discusión

Los experimentos realizados demuestran que el algoritmo propuesto tiene un rendimiento superior a las arquitecturas de redes neuronales convencionales evaluadas. Los principales hallazgos incluyen:

- **Rendimiento superior:** Borolo alcanza una clasificación perfecta en ambos conjuntos de datos, el cual es significativamente superior a las redes neuronales tradicionales.
- **Adaptabilidad estructural:** El enfoque propuesto demuestra la capacidad para ajustar automáticamente su complejidad arquitectónica según los requisitos del problema, requiriendo 3 iteraciones para círculos concéntricos y 7 para espirales.

- **Eficiencia paramétrica:** Las arquitecturas generadas por Borolo utilizan un número de parámetros comparable a las redes neuronales convencionales, pero con un rendimiento sustancialmente superior.
- **Convergencia estable:** En ambos experimentos se observa una convergencia satisfactoria de las métricas de entrenamiento, indicando estabilidad del algoritmo.

5.5. Reproducibilidad

Para garantizar la reproducibilidad de los experimentos, se proporciona acceso completo al código implementado:

- Experimento 1 (Círculos concéntricos): Colab Experimento 1
- Experimento 2 (Espirales): Colab Experimento 2

Los notebooks incluyen la implementación completa de los experimentos, la configuración de hiperparámetros y los procedimientos de evaluación utilizados.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

El presente trabajo introduce un nuevo paradigma para el diseño de redes neuronales artificiales denominado *Redes Neuronales Borolo*, este propone una aproximación fundamentalmente diferente a la optimización de arquitecturas neuronales predominantes en el estado del arte. El enfoque ayuda a trascender las limitaciones impuestas por la optimización tradicional basada en módulos pre-diseñados, como las capas convolucionales o los mecanismos de atención tipo transformer, esto mediante la implementación de un enfoque que opera directamente a nivel de neurona individual.

En el presente trabajo se establece un marco teórico basado en la teoría de grafos para la representación y manipulación de arquitecturas neuronales dinámicas. Esta formalización matemática conceptualiza las redes neuronales como un grafo, del cual se evoluciona su topología mediante la adición o eliminación selectiva de neuronas y conexiones sinápticas. Se propone un algoritmo de evolución estructural que integra mecanismos de propagación

hacia adelante y hacia atrás adaptados específicamente para estructuras dinámicas. Se implementan funciones de selección neuronal que permiten optimizar simultáneamente los parámetros de la red y su topología, representando un avance sustancial respecto a los métodos tradicionales de búsqueda de arquitecturas neuronales (NAS).

Los resultados experimentales revelan características prometedoras del enfoque Borolo. En primer lugar, al haber menos conexiones se tiene una menor cantidad de operaciones computacionales, abonando con ello a una mayor eficiencia computacional comparada con arquitecturas estáticas de tamaño fijo. Esta eficiencia está relacionada con la capacidad del sistema para adaptar su complejidad a los requerimientos específicos del problema.

Adicionalmente, la naturaleza adaptativa de las Redes Neuronales Borolo permite una exploración más completa del espacio de soluciones. Esto dado que la topología de la red coevoluciona con los parámetros sinápticos.

Por otro lado en dominios donde no hay una arquitectura neuronal conocida apropiada, Borolo puede ofrecer una aproximación optimizada ad hoc a problemas de dichos dominios.

En conclusión, los resultados preliminares validan la viabilidad del enfoque y justifican investigación futura para explorar su potencial completo.

6.2. Trabajo Futuro

Una mejora prioritaria consiste en desarrollar una implementación eficiente en CUDA [43, 44, 45] que aproveche las capacidades de computación paralela de las unidades de procesamiento gráfico (GPU). Esto para escalar los experimentos a conjuntos de datos de mayor complejidad y evaluar el comportamiento del sistema en problemas del mundo real.

El sistema actual utiliza descenso de gradiente básico, lo cual limita significativamente su capacidad de exploración y convergencia. Las mejoras prioritarias incluyen: implementación

de optimizadores adaptativos como Adam, AdamW, etc [46, 34] que aceleren la convergencia; esquemas que ajusten dinámicamente la tasa de aprendizaje basándose en la profundidad efectiva de la red; estrategias de entrenamiento que aprovechen las características dinámicas de la arquitectura.

Por otro lado, la función de selección neuronal representa un componente crítico del sistema Borolo. En este sentido, las investigaciones futuras pueden abordar temas como: el análisis comparativo de diferentes métricas de importancia neuronal, gradientes, activaciones, conectividad; funciones de selección basadas en teoría de información que maximicen la ganancia de información mutua; criterios de poda, etc.

Otro aspecto a explorar es el uso de estrategia para la exploración de hiperparámetros evolutivos. El espacio de hiperparámetros específicos de Borolo requiere investigación sistemática de los valores de: el momento generacional, las tasas de crecimiento neuronal y de conexiones.

Una de las direcciones más interesantes consiste en extender el paradigma Borolo hacia, arquitecturas con múltiples salidas distribuidas a lo largo de la red. Esta aproximación, particularmente relevante para modelos de lenguaje, ya que involucra aspectos como:

El uso de neuronas que representen conceptos específicos (entidades, relaciones, atributos) distribuidas en diferentes niveles de la red; funciones de pérdida adaptativas y la exploración de topologías que permitan flujo de información en múltiples direcciones manteniendo la aciclicidad del grafo.

Bibliografía

[1] Fortune Business Insights. «Machine learning market trends and growth statistics 2020-2024». En: *Market Research Report* (2024). Global ML market valued at USD 35.32 billion in 2024, projected to reach USD 309.68 billion by 2032 with CAGR of 30.5 %.

[2] Lars Kotthoff et al. «Neural module networks: A review». En: *Neurocomputing* 545 (2023), pág. 126410.

[3] Jacob Andreas et al. «Neural module networks». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 39-48.

[4] Louis Meunier et al. «Modular networks: Learning to decompose neural computation». En: *Advances in Neural Information Processing Systems* 32 (2019).

[5] Barret Zoph y Quoc V. Le. «Neural Architecture Search with Reinforcement Learning». En: *5th International Conference on Learning Representations, ICLR 2017*. 2017. URL: https://arxiv.org/abs/1611.01578.

[6] Randal S Olson y Jason H Moore. «TPOT: A tree-based pipeline optimization tool for automating machine learning». En: *Workshop on automatic machine learning*. PMLR. 2016, págs. 66-74.

[7] Takuya Akiba et al. «Optuna: A next-generation hyperparameter optimization framework». En: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, págs. 2623-2631.

[8] Hanxiao Liu, Karen Simonyan y Yiming Yang. «DARTS: Differentiable Architecture Search». En: *7th International Conference on Learning Representations, ICLR 2019*. 2019. URL: https://arxiv.org/abs/1806.09055.

[9] Stefan Falkner, Aaron Klein y Frank Hutter. «BOHB: Robust and Efficient Hyperparameter Optimization at Scale». En: *International Conference on Machine Learning*. 2018, págs. 1437-1446.

[10] Esteban Real et al. «Large-Scale Evolution of Image Classifiers». En: *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*. Proceedings of Machine Learning Research, 2017, págs. 2902-2911.

[11] Kenneth O. Stanley y Risto Miikkulainen. «Evolving Neural Networks Through Augmenting Topologies». En: *Evolutionary Computation* 10.2 (2002), págs. 99-127.

[12] Hongrong Ren et al. «A Survey on Deep Neural Network Pruning: Taxonomy, Comparison, Analysis, and Recommendations». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024). DOI: 10.1109/TPAMI.2024.3447085.

[13] Carolin Scholl, Michael E. Rule y Matthias H. Hennig. «The information theory of developmental pruning: Optimizing global network architectures using local synaptic rules». En: *PLOS Computational Biology* 17.10 (2021), e1009458.

[14] Yizeng Han et al. «Dynamic neural networks: A survey». En: *IEEE transactions on pattern analysis and machine intelligence* 44.11 (2021), págs. 7436-7456.

[15] Jonathan Frankle y Michael Carbin. «The lottery ticket hypothesis: Finding sparse, trainable neural networks». En: *arXiv preprint arXiv:1803.03635* (2018).

[16] Amir Gholami et al. «A Survey of Quantization Methods for Efficient Neural Network Inference». En: *arXiv preprint arXiv:2103.13630* (2021).

[17] Jiwei Yang et al. «Quantization networks». En: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, págs. 7308-7316.

[18] Geoffrey Hinton, Oriol Vinyals y Jeff Dean. «Distilling the knowledge in a neural network». En: *arXiv preprint arXiv:1503.02531* (2015).

[19] Dmitry Krotov y John J. Hopfield. «Unsupervised learning by competing hidden units». En: *Proceedings of the National Academy of Sciences* 116.16 (2019), págs. 7723-7731.

[20] Thomas Miconi, Kenneth Stanley y Jeff Clune. «Differentiable plasticity: training plastic neural networks with backpropagation». En: *International Conference on Machine Learning* (2018), págs. 3559-3568.

[21] Julia V. Gallinaro, Benjamin Scholl y Claudia Clopath. «Synaptic weights that correlate with presynaptic selectivity increase decoding performance». En: *PLOS Computational Biology* 19.8 (2023), e1011362.

[22] Thomas Elsken, Jan Hendrik Metzen y Frank Hutter. «Neural architecture search: A survey». En: *The Journal of Machine Learning Research* 20.1 (2019), págs. 1997-2017.

[23] Colin White et al. «Neural architecture search: Insights from 1000 papers». En: *arXiv preprint arXiv:2301.08727* (2023).

[24] Gabriel Bender et al. «Understanding and simplifying one-shot architecture search». En: *International conference on machine learning*. PMLR. 2018, págs. 550-559.

[25] Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd. Chapter 22.4: Topological sort, pages 612–615. Cambridge, MA: MIT Press, 2009.

[26] Shiv Ram Dubey, Satish Kumar Singh y Bidyut Baran Chaudhuri. «Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark». En: *Neurocomputing* 503 (2022), págs. 92-108.

[27] Ivan S. Sokolnikoff. «Tensor Analysis: Theory and Applications to Geometry and Mechanics of Continua». En: *Applied Mathematics Series* (1964). Chapter 1: The Summation Convention, pages 1–15.

[28] Xavier Glorot, Antoine Bordes y Yoshua Bengio. «Deep Sparse Rectifier Neural Networks». En: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, págs. 315-323.

[29] Andrew L. Maas, Awni Y. Hannun y Andrew Y. Ng. «Rectifier Nonlinearities Improve Neural Network Acoustic Models». En: *Proc. icml*. Vol. 30. 1. 2013, pág. 3.

[30] Dan Hendrycks y Kevin Gimpel. «Gaussian Error Linear Units (GELUs)». En: *arXiv preprint arXiv:1606.08415* (2016).

[31] Sepp Hochreiter. «Untersuchungen zu dynamischen neuronalen Netzen». Tesis doct. Technische Universität München, 1991.

[32] Andrey Nikolayevich Tikhonov. «On the stability of inverse problems». En: *Doklady Akademii Nauk SSSR* 39.5 (1943), págs. 195-198.

[33] Andy Brock et al. «High-Performance Large-Scale Image Recognition Without Normalization». En: *Proceedings of the 38th International Conference on Machine Learning*. Ed. por Marina Meila y Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul de 2021, págs. 1059-1071. URL: https://proceedings.mlr.press/v139/brock21a.html.

[34] Ilya Loshchilov y Frank Hutter. «Decoupled Weight Decay Regularization». En: *7th International Conference on Learning Representations, ICLR 2019*. 2019. URL: https://arxiv.org/abs/1711.05101.

[35] Fabian Pedregosa et al. «Scikit-learn: Machine learning in Python». En: *Journal of machine learning research* 12 (2011), págs. 2825-2853.

[36] Lisha Li et al. «Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization». En: vol. 18. 2017, págs. 1-52.

[37] Claude E. Shannon. «A Mathematical Theory of Communication». En: *Bell System Technical Journal* 27.3 (jul. de 1948). Foundational paper introducing information entropy, págs. 379-423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[38] Jagat Narain Kapur. «Measures of Information and their Applications». En: *Wiley-Interscience* (1992). Comprehensive treatment of entropy-variance relationships for uniform and Laplace distributions.

[39] Peter J. Bickel y Kjell A. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*. 2nd. Vol. 1. Advanced treatment of entropy properties for probability distributions. Boca Raton, FL: Chapman y Hall/CRC, 2015. ISBN: 978-1-498-72380-8.

[40] David Martin Ward Powers. «Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation». En: *arXiv preprint arXiv:2010.16061* (2020).

[41] Herbert Robbins y Sutton Monro. «A stochastic approximation method». En: *The annals of mathematical statistics* (1951), págs. 400-407.

[42] Milaan Parmar / milaan9. *Clustering-Datasets: collection of UCI and synthetic datasets for clustering algorithms*. `https://github.com/milaan9/Clustering-Datasets`. acceso: 2025-11-11. Nov. de 2025.

[43] Jaegeun Han y Bharatkumar Sharma. *Learn CUDA Programming: A beginner's guide to GPU programming and parallel computing with CUDA 10.x and C/C++*. Chapter 9: Deep Learning Acceleration with CUDA, pages 295-342. Packt Publishing, 2020.

[44] Jamie Flux. *Machine Learning with CUDA: Enhancing Neural Network Performance*. Comprehensive guide to CUDA programming for neural network acceleration. GPU Mastery Series, 2024.

[45] NVIDIA. *CUDA Deep Neural Network library (cuDNN)*. `https://developer.nvidia.com/cudnn`. GPU-accelerated library for deep neural network primitives. 2024.

[46] Diederik P. Kingma y Jimmy Ba. «Adam: A Method for Stochastic Optimization». En: *3rd International Conference on Learning Representations, ICLR 2015*. 2015. URL: https://arxiv.org/abs/1412.6980.

Rodrigo Israel Hernández Mazariegos

Redes neuronales Borolo un enfoque basado en la optimización de estructuras dinámicas a nivel de neurona.pdf

Universidad Michoacana de San Nicolás de Hidalgo

Detalles del documento

Identificador de la entrega
trn:oid:::3117:532553531

Fecha de entrega
24 nov 2025, 12:30 p.m. GMT-6

Fecha de descarga
24 nov 2025, 12:35 p.m. GMT-6

Nombre del archivo
Redes neuronales Borolo un enfoque basado en la optimización de estructuras dinámicas a nivel....pdf

Tamaño del archivo
3.1 MB

70 páginas

12.980 palabras

68.787 caracteres

6% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Texto citado
- Texto mencionado
- Coincidencias menores (menos de 10 palabras)

Fuentes principales

6% Fuentes de Internet
4% Publicaciones
0% Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.º de alerta de integridad para revisión

Caracteres reemplazados
25 caracteres sospechosos en N.º de páginas
Las letras son intercambiadas por caracteres similares de otro alfabeto.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

Formato de Declaración de Originalidad y Uso de Inteligencia Artificial

Coordinación General de Estudios de Posgrado
Universidad Michoacana de San Nicolás de Hidalgo

A quien corresponda,

Por este medio, quien abajo firma, bajo protesta de decir verdad, declara lo siguiente:

● Que presenta para revisión de originalidad el manuscrito cuyos detalles se especifican abajo.
● Que todas las fuentes consultadas para la elaboración del manuscrito están debidamente identificadas dentro del cuerpo del texto, e incluidas en la lista de referencias.
● Que, en caso de haber usado un sistema de inteligencia artificial, en cualquier etapa del desarrollo de su trabajo, lo ha especificado en la tabla que se encuentra en este documento.
● Que conoce la normativa de la Universidad Michoacana de San Nicolás de Hidalgo, en particular los Incisos IX y XII del artículo 85, y los artículos 88 y 101 del Estatuto Universitario de la UMSNH, además del transitorio tercero del Reglamento General para los Estudios de Posgrado de la UMSNH.

Datos del manuscrito que se presenta a revisión		
Programa educativo	**Maestría en Ciencias de Ingeniería Eléctrica**	
Título del trabajo	**Redes neuronales Borolo: un enfoque basado en la optimización de estructuras dinámicas a nivel de neurona.**	
	Nombre	**Correo electrónico**
Autor/es	**Rodrigo Israel Hernández Mazariegos**	**1301441a@umich.mx**
Director	**José Ortiz Bejar**	**jose.ortiz@umich.mx**
Codirector	**Jaime Cerda Jacobo**	**jaime.cerda@umich.mx**
Coordinador del programa	**J.Aurelio Medina Ríos**	**aurelio.medina@umich.mx**

Uso de Inteligencia Artificial		
Rubro	**Uso (sí/no)**	**Descripción**

Asistencia en la redacción	SI	Solo en algunas partes(muy pocas), por ejemplo en el resumen y el abstract.

Formato de Declaración de Originalidad y Uso de Inteligencia Artificial

Coordinación General de Estudios de Posgrado
Universidad Michoacana de San Nicolás de Hidalgo

Uso de Inteligencia Artificial		
Rubro	**Uso (sí/no)**	**Descripción**
Traducción al español	no	
Traducción a otra lengua	no	
Revisión y corrección de estilo	no	
Análisis de datos	no	
Búsqueda y organización de información	no	
Formateo de las referencias bibliográficas	si	Generación de citas y ordenamiento.
Generación de contenido multimedia	no	
Otro	no	

Datos del solicitante	
Nombre y firma	Rodrigo Israel Hernández Mazariegos
Lugar y fecha	Morelia Michoacán, 21 Noviembre 2025