TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores
(Computer Engineering School)

Programa de Licenciatura en Ingeniería en Computadores
(Licentiate Degree Program in Computer Engineering)

TEC | Tecnológico de Costa Rica

Aplicación de Técnicas de Aprendizaje Automático para Generación de Circuitos Aproximados

(Application of Machine Learning Techniques for the Generation of Approximate Circuits)

Informe de Trabajo de Graduación para optar por el título de Ingeniero en Computadores con grado académico de Licenciatura
(Report of Graduation Work in fulfillment of the requirements for the degree of Licentiate in Computer Engineering)

Ignacio Elías Vargas Campos
Cartago, <MES>, 2025

Hoja de Aprobación

Dedicatoria

Agradecimientos

Resumen

Abstract

Índice general

Índice de tablas

Índice de figuras

Siglas y acrónimos

- ALS: Síntesis de lógica aproximada (*Approximate Logic Synthesis*)
- ML: Aprendizaje automático (*Machine Learning*)
- DT: Árbol de decisión (*Decision Tree*)
- RF: Bosque aleatorio (*Random Forest*)
- LUT: Tabla de búsqueda (*Lookup Table*)
- DL: Aprendizaje profundo (*Deep Learning*)
- PGNN: Red neuronal de grafos con pesos en los caminos (*Path-weighted Graph Neural Network*)
- MCTS: Búsqueda en árbol de Monte Carlo (*Monte Carlo Tree Search*)
- CGP: Programación genética cartesiana (*Cartesian Genetic Programming*)
- CAD: Diseño asistido por computador (*Computer-aided design*)
- EDA: Diseño electrónico automático (*Electronic Design Automation*)
- SOP: Suma de productos (*Sum of products*)
- RL: Aprendizaje reforzado (*Reinforcement Learning*)
- BDD: Diagrama de decisión binario (*Binary Decision Diagram*)
- ER: Tasa de error (*Error Rate*)
- MHD: Distancia de Hamming promedio (*Mean Hamming Distance*)
- WHD: Peor distancia de Hamming (*Worst Hamming Distance*)
- WCE: Error en el peor caso (*Worst Case Error*)
- MAE: Error absoluto medio (*Mean Absolute Error*)
- MRE: Error relativo medio (*Mean Relative Error*)
- MSE: Error cuadrático medio (*Mean Squared Error*)

1 Introducción

Este proyecto consiste en la aplicación de técnicas de aprendizaje automático (Machine Learning, ML) al campo de la Síntesis Lógica Aproximada (Approximate Logic Synthesis, ALS). Este informe detalla el proceso para la integración de una técnica del estado del arte de ML en la herramienta de fuente abierta AxLS.

En este capítulo se presenta el contexto y la motivación detrás del desarrollo del proyecto. Se describen los antecedentes relevantes, el problema identificado, así como los objetivos y alcances planteados para dar respuesta a dicho problema.

1.1. Antecedentes del proyecto

En esta sección se recopila información sobre la organización involucrada, el área temática del proyecto y trabajos relacionados que han abordado problemáticas similares. Esto permite ubicar el proyecto dentro de un marco de referencia académico.

1.1.1. Descripción de la organización

Se realizará en colaboración con la escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica, específicamente con el laboratorio de "Efficient Computing Across the Stack" (ECASLab).

El Tecnológico de Costa Rica es una de las universidades públicas de Costa Rica, con su sede original en Cartago creada en 1971 [1].

1.1.2. Descripción del área de conocimiento del proyecto

Este proyecto trata temas en la intersección de las áreas de computación aproximada y ML. Específicamente en el campo generación de circuitos aproximados, a la que se le llama Síntesis de Lógica Aproximada.

1.1.3. Trabajos similares encontrados

En [2], Castro-Godínez et al. presentan una herramienta de fuente abierta para ALS llamada AxLS. Este framework implementa múltiples técnicas de ALS y permite utilizar los mismos métodos para calcular el umbral de error y métricas de calidad en ellas, lo que permite una mejor comparación de los diferentes enfoques de ALS. Este trabajo es fundamental para este proyecto, ya que se realizó sobre herramienta AxLS, aportando un enfoque de ML a la herramienta.

La intersección de aplicar técnicas de ML para ALS es un nicho que ha surgido principalmente en los últimos 5 años. Dentro de esta área se han identificado 3 categorías principales de aplicación de ML a ALS.

La primera categoría identificada es sobre métodos que buscan entrenar modelo que funcione como "asistente" en otros métodos de ALS, evaluando más rápidamente los efectos que causaría generar cambios en un circuito, usualmente ayudando a evaluar error rápidamente. El beneficio es que en muchos métodos se generan cambios a prueba y error o con heurísticas, pero se debe evaluar el error introducido por cada cambio lo cual puede conllevar simulaciones costosas computacionalmente, pero un modelo puede estimar el error con una exactitud aceptable mucho más rápido. Dentro de esta categoría se encuentran los trabajos de Pasandi et al. [3] [4], en donde aplica técnicas de aprendizaje reforzado y aprendizaje profundo (Deep Learning, DL), respectivamente, para estimar el error generado en un circuito al hacer cambios locales. En [5], Ye et al. no estiman directamente el error con ML, pero entrenan un agente de aprendizaje reforzado para realizar cambios locales en un circuito y los estados de este agente son representados con una red neuronal de grafos con pesos en los caminos (Path-weighted Graph Neural Network, PGNN).

La segunda categoría identificada es sobre métodos que se enfocan en técnicas de ML para realizar exploraciones del espacio de diseño de un circuito. Específicamente suelen emplear una técnica llamada Búsqueda en Árbol de Monte Carlo (Monte Carlo Tree Search, MCTS). El trabajo de Rajput et al. [6] plantea la aplicación de MCTS para ALS, modificando el algoritmo levemente para que pueda explorar nodos más profundos en el árbol y utilizando la reducción de área como recompensa para evaluar nodos en el árbol. También se cuenta con [7], donde Awais et al. también aplican MCTS al problema de ALS utilizando ML para la estimación de error, lo cual lo hace también un ejemplo de la primera categoría de aplicaciones de ML a ALS mencionada.

La tercera categoría identificada conlleva realizar un entrenamiento supervisado sobre las entradas y salidas de un circuito, el modelo de ML entrenado es seguidamente mapeado a un circuito. Ya que los modelos de aprendizaje supervisado aprenden a generalizar una función, al ser mapeados a un circuito se obtiene un circuito que aproxima al original.

Uno de los primeros ejemplos en esta categoría es [8], donde Boroumand et al. desarrollan un método para aprender funciones lógicas de ejemplos y sintetizar circuitos basado en el modelo aprendido, a lo que le llaman síntesis lógica a partir de ejemplos. En [9], De Abreu et al. exploran el uso de árboles de decisiones (Decision Tree, DT) como alternativa tanto para optimizar circuitos exactos como para generar versiones aproximadas de los circuitos. En ambos casos la técnica es exitosa, reduciendo el tiempo de ejecución para optimizar los circuitos exactos al compararse las alternativas comunes ABC y Espresso, así como siendo capaz de generar circuitos aproximados de profundidad y área sumamente reducidos e igual logrando buenos niveles de exactitud. Miayasaka et al. prueban y comparan varios métodos populares de aprendizaje supervisado en [10], incluyendo redes neuronales, árboles de decisiones y redes de tablas de búsqueda (Lookup Table, LUT). Evalúan estos métodos en su capacidad de aproximar circuitos lógicos y aritméticos, notando que estos modelos populares no pueden aprender de manera efectiva ciertos tipos de circuitos aritméticos.

Los resultados del concurso del International Workshop on Logic & Synthesis (IWLS) del 2020 [11] han sido sumamente claves en el avance de esta área. El concurso consistía en implementar 100 funciones booleanas dados ejemplos incompletos de sus tablas de verdad, luego las implementaciones fueron validadas con el set de ejemplos de validación que no fue proporcionado a los concursantes. Entre las técnicas evaluadas se encuentran DT, bosques aleatorios (Random Forest, RF), redes de LUT, Espresso, redes neuronales y programación genética cartesiana (Cartesian Genetic Programming, CGP).

Las principales conclusiones del análisis de resultados son que ninguna técnica dominó todas las pruebas; la mayoría de los equipos, incluyendo el ganador, emplearon un conjunto de técnicas diferentes. Los bosques aleatorios y los árboles de decisión fueron muy populares y constituyen un punto de referencia sólido para ALS. Y muy importantemente, sacrificar un poco de exactitud permite una reducción significativa en el tamaño del circuito.

Zeng et al. [12] explora con mayor profundidad el uso de DT para ALS, utilizando una alteración de la técnica llamada árbol de decisión adaptable, particularmente aplicando variaciones guiadas por una métrica llamada "Shapley Additive Explanations" (SHAP), que busca explicar la importancia de las características de entrada a un modelo. En la misma línea de explorar variaciones a la técnica de DT, Huang y Jiang [13] exploran el uso de grafos de decisión para relajar las limitaciones estructurales de un árbol, como el crecimiento exponencial a medida que aumenta la complejidad. Otra variación novedosa de los DT es la propuesta de Hu y Cai [14], donde aplican una técnica a la que llaman árboles de decisión óptimos, proporcionando una garantía de optimalidad, lo que les permite un mejor control en el balance entre precisión y complejidad del circuito.

También se ha profundizado en la técnica de CGP. Berndt et al. [15] proponen recibir una tabla de verdad que representa el comportamiento deseado y evolucionar circuitos para ajustarse a ese comportamiento, partiendo de circuitos aleatorios o de un circuito previamente especificado. Una ventaja de esta técnica es que puede combinarse con otras, utilizando circuitos generados por otras técnicas como punto de partida en el proceso evolutivo.

Prats Ramos et al. [16] presentan un análisis comparativo entre las técnicas de DT, CGP y un enfoque mixto de ML que fue originalmente presentado por el equipo ganador del concurso de IWLS 2020 [11] y combina el uso de redes neuronales y LUT.

1.2. Planteamiento del problema

En esta sección se describe el problema central que motiva este proyecto. Se presenta el contexto en el que surge, su justificación desde una perspectiva investigativa, y se formula claramente el problema que se busca resolver.

1.2.1. Contexto del problema

Los investigadores del Instituto Tecnológico de Costa Rica enfrentan limitaciones en el uso de técnicas de ALS basadas en ML. Actualmente, la herramienta disponible para

ellos, AxLS, no cuenta con las capacidades técnicas necesarias para implementar estas técnicas, lo que restringe sus posibilidades de experimentación y análisis.

1.2.2. Justificación del problema

La capacidad de utilizar técnicas de ALS basadas en ML permitiría a los investigadores verificar y comparar resultados obtenidos por otros grupos de investigación. Además, facilitaría el diseño y desarrollo de técnicas novedosas de ALS basadas en ML, permitiéndoles contribuir de manera más significativa en la investigación en ALS.

1.2.3. Enunciado del problema

Los investigadores del Instituto Tecnológico de Costa Rica carecen de una herramienta que les permita aplicar técnicas de ALS basadas en aprendizaje automático. La herramienta disponible, AxLS, no posee las capacidades técnicas necesarias para ello, lo que impide la validación, comparación y desarrollo de nuevas técnicas en este campo.

1.3. Objetivos del proyecto

1.3.1. Objetivo General

Formular una técnica del estado del arte de aprendizaje automático para ALS en la herramienta AxLS, de manera que permita ser comparada con otras técnicas implementadas dentro de la herramienta así como con otras implementaciones diferentes de las técnicas seleccionadas.

1.3.2. Objetivos Específicos

1. Evaluar al menos 3 técnicas de ALS basadas en aprendizaje automático determinando cuál es la más apropiada para ser integrada en la herramienta AxLS.

2. Adaptar la herramienta AxLS para que sea posible la implementación de la técnica escogida, con la consideración de que permita más fácilmente la implementación de futuras técnicas de aprendizaje automático.

3. Evaluar los resultados de la técnica de aprendizaje automático implementada en AxLS con respecto a resultados obtenidos mediante las técnicas ya existentes en la herramienta.

1.4. Alcances, entregables y limitaciones del proyecto

El alcance de este proyecto incluye investigar y evaluar las técnicas del estado del arte, escoger e implementar una en la herramienta AxLS y documentar el proceso. Los entregables asociados a este alcance, las técnicas y herramientas a emplear así como sus estrategias de verificación se presentan en la Tabla 1.1.

Como limitación, la implementación escogida está restringida a las capacidades actuales de AxLS, incluyendo las métricas y conjunto de pruebas con las que cuenta actualmente. Este proyecto no plantea agregar más pruebas a la herramienta. Otra limitación del proyecto es la falta de acceso a máquinas con capacidades computacionales de alto rendimiento, por lo que la técnica seleccionada no puede requerir un entrenamiento demasiado pesado. Debe ser factible su entrenamiento en poco tiempo en un computador personal.

Tabla 1.1: Entregables del proyecto.

Entregable	Técnicas / Herramientas	Estrategias de verificación	Objetivo Específico
Documento de diseño que justifica la escogencia de la técnica de aprendizaje automático a implementar en AxLS. Compara todas las técnicas que estén bajo consideración inicialmente de aquellas encontradas en trabajos relacionados del estado del arte.	Zotero para manejo de referencias.	Documento detalla y escoge una técnica en específico, da razones de por qué se escogió tomando en cuenta la factibilidad y relevancia en el estado del arte.	1
Documento de diseño que explica el plan de implementación de la técnica escogida dentro de la herramienta AxLS.	AxLS, Python, Yosys.	Documento detalla todos los cambios de API necesarios en AxLS y la API para utilizar la técnica nueva, con ejemplos de uso y un plan de pruebas. Detalla el proceso de crear los datos de entrenamiento, entrenamiento del modelo y sintetización del circuito.	2
Código adaptado de herramienta AxLS con técnica de ML integrada.	AxLS, Python, Yosys, git, Github, Pytorch u otra biblioteca de ML.	La herramienta AxLS sigue funcionando con todas sus técnicas pasadas y además se puede escoger utilizar la técnica de ML y la cual se ejecuta correctamente. La ejecución correcta se validará con un plan de pruebas.	2
Informe final sobre el proyecto realizado y los resultados encontrados al comparar la técnica de ML con las ya existentes en la herramienta AxLS	LaTeX, IEEE, AxLS	El informe describe cómo se llevó a cabo el proyecto, la técnica de ML empleada y presenta los resultados de esta técnica comparados con los de las otras técnicas ya existentes en la máquina con gráficos.	3

2 Marco de referencia teórico

Este capítulo introduce los conceptos teóricos necesarios para respaldar este proyecto, el cual se sitúa en la intersección entre las áreas de ALS y ML. La integración de estas disciplinas permite explorar nuevos métodos para la generación de circuitos digitales que intercambien exactitud por eficiencia y complejidad.

Se revisan los conceptos básicos de la computación aproximada y las técnicas tradicionales de ALS, con el objetivo de contrastarlas con los enfoques basados en ML. Luego se introducen conceptos fundamentales de ML, necesarios para entender los mecanismos mediante los cuales un modelo puede ser útil en la generación de circuitos aproximados, siendo especialmente relevante su capacidad para aprender y generalizar funciones booleanas. Finalmente, se abordan las técnicas de ML más valiosos en la aplicación de ALS, junto con los métodos para transformar los modelos de ML a circuitos lógicos, paso comúnmente necesario para su aplicación práctica dentro del marco de ALS.

La Figura 2.1 mapea los conceptos tratados en este capítulo en un mapa conceptual.

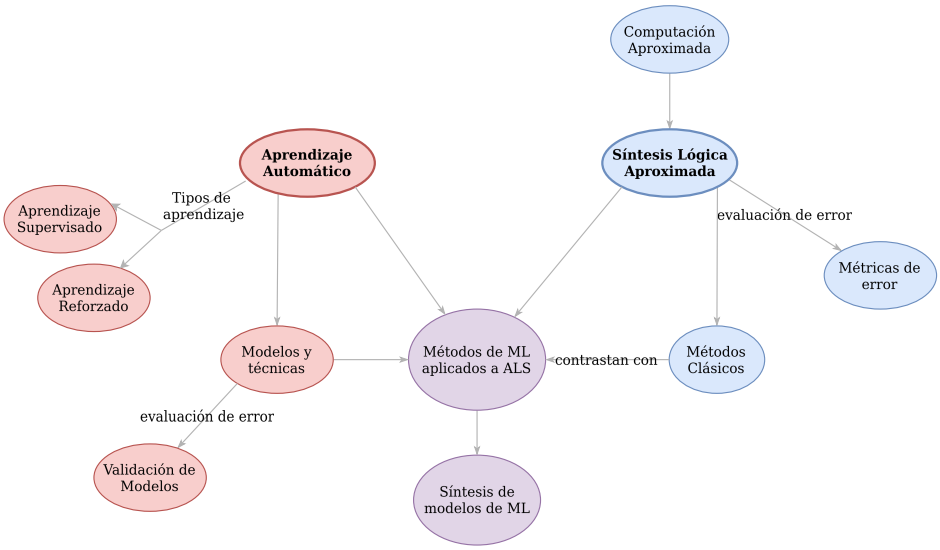

Figura 2.1: Mapa conceptual de los temas abordados en el marco teórico. Las burbujas rojas corresponden a temas asociados a ML, las burbujas azules corresponden a temas asociados a ALS y las burbujas moradas corresponden a temas que integran las 2 áreas.

2.1. Computación aproximada

En esta sección se explican los conceptos de computación aproximada más fundamentales para este trabajo.

La computación aproximada surgió como un paradigma para diseñar sistemas digitales más eficientes en consumo de energía y rendimiento. Muchos sistemas y aplicaciones son tolerantes a errores en los resultados computados; por ejemplo, el procesamiento multimedia (audio, video, gráficos e imágenes), el reconocimiento de patrones y la minería de datos. Al relajar la necesidad de realizar operaciones exactas y determinísticas, las técnicas de computación aproximada permiten mejorar significativamente la eficiencia de los circuitos. [17]

Se denomina circuito aproximado a este tipo de circuito digital que sacrifica precisión en sus resultados a cambio de una mayor eficiencia en su rendimiento, área, consumo de potencia o una combinación de estas métricas.

2.1.1. Síntesis lógica aproximada

La construcción automática de circuitos digitales aproximados se realiza mediante herramientas de diseño asistido por computador (Computer-Aided Design, CAD), disponibles en entornos de diseño electrónico automatizado (Electronic Design Automation, EDA). ALS se enfoca en mejorar el comportamiento de un circuito (ya sea en consumo de potencia, área o rendimiento) sin necesidad de conocer sus detalles de implementación. Este proceso se lleva a cabo de forma sistemática, modificando la funcionalidad especificada sin exceder un umbral de error predefinido.

Existen métodos para crear circuitos aproximados manualmente, que no caen dentro de la categoría de ALS, ya que no son automáticos. Este tipo de diseño requiere un profundo conocimiento del comportamiento del circuito para poder hacer aproximaciones que no degraden demasiado su precisión. Este nivel alto de pericia requerida sobre la topología del circuito a aproximar es la principal desventaja de estos métodos y el porqué en los últimos años la investigación se ha enfocado hacia métodos de ALS. [18]

Los métodos de optimización de circuitos digitales se suelen dividir en métodos para dos o para múltiples niveles de profundidad lógica. En este caso la profundidad lógica hace referencia a la cantidad máxima de compuertas lógicas AND u OR por las que debe pasar una señal de la entrada a la salida del circuito [19].

Ya que los circuitos de dos niveles tienen una profundidad lógica fija, su optimización se enfoca en reducir la cantidad de operaciones en la representación de suma de productos (Sum Of Products, SOP) del circuito. Para circuitos de múltiples niveles, el objetivo de optimización depende de la estructura booleana utilizada para representarlos y optimizarlos, pero los métodos suelen buscar reducir el número de compuertas lógicas y la profundidad del circuito. Los métodos de ML se destacan en este contexto, ya que pueden manejar circuitos de profundidad arbitraria y son particularmente útiles para optimizar circuitos complejos, lo que los convierte en una herramienta valiosa para la optimización multi-nivel. [18]

En este contexto, en los últimos años se ha explorado con gran éxito la aplicación

de técnicas de ML para ALS [3], [11], [20], [16]. Gracias a su capacidad de generalizar funciones, el aprendizaje de funciones booleanas ha sido una de las técnicas de ML más exploradas y útiles para la generación de circuitos aproximados. Esta efectividad se debe a que, al aprender una función, el modelo busca cubrir todas las posibles soluciones de manera eficiente. Así, logra un buen equilibrio entre precisión y rendimiento, lo que permite crear circuitos eficientes sin perder demasiada exactitud.

2.1.2. Métricas de error

Independientemente de la metodología utilizada para ALS, es necesario definir una manera de calcular el error insertado en la síntesis.

Existen dos categorías principales de métricas: de propósito general, que están relacionadas con la ocurrencia del error, y métricas aritméticas, que están relacionadas con la magnitud del error.

Algunos métodos de ALS calculan el error introducido de manera exacta, mientras que otros lo estiman de manera aproximada. Los métodos para calcularlo de manera exacta suelen ser más complejos y se pueden basar en el problema de satisfaciabilidad booleana, diagramas de decisión binarios (Binary Decision Diagram, BDD), álgebra simbólica computacional o incluso revisar todas las posibles combinaciones de entradas, con tablas de verdad o simulaciones exhaustivas. La estimación aproximada del error se suele implementar aplicando simulaciones Monte Carlo, pero existen otras técnicas [18]. De particular interés para este trabajo, se han aplicado técnicas de ML para la estimación de error de manera aproximada; por ejemplo, en [4] entrenan un modelo de DL para predecir el error en las salidas primarias de un circuito al quitar o reemplazar compuertas lógicas en la red del circuito.

La decisión de cuál metodología utilizar para el cálculo de error debe tomar en cuenta el balance entre la calidad de los resultados y el esfuerzo computacional. Un cálculo de error preciso suele resultar en mejores circuitos aproximados, porque lleva a mejores decisiones de cuáles aproximaciones aplicar durante el proceso de ALS. Sin embargo, un cálculo preciso puede presentar costos computacionales altos, especialmente al ejecutarlos sobre cada posible modificación a la topología del circuito [18]. Además, el costo computacional suele crecer exponencialmente con la complejidad y la cantidad de entradas del circuito.

El resto de esta sección introducirá algunas de las métricas de error más comúnmente utilizadas para ALS. Para las expresiones presentadas, n corresponde con la cantidad de bits de entrada del circuito, N con la cantidad de combinaciones de entradas siendo tomadas en cuenta, $f(x)$ y $f'(x)$ son las salidas del circuito original y el aproximado, respectivamente, con $f_i(x)$ y $f'_i(x)$ representando el bit de salida en la posición i. Nótese que si se toman en cuenta todas las posibles combinaciones de entradas del circuito el valor de N sería 2^n.

Tasa de error

La tasa de error (Error Rate, ER) corresponde con la probabilidad de observar uno o más bits erróneos en la salida. ER es la métrica más comúnmente utilizada en la

aproximación de circuitos, siendo aplicada en circuitos tanto generales como aritméticos [18]. El cálculo del ER se presenta en la ecuación 2.1.

$$ER = \frac{1}{N} \sum_{\forall x \in N} f(x) \neq f'(x) \tag{2.1}$$

Distancia de Hamming promedio

La distancia de Hamming promedio (Mean Hamming Distance, MHD), también llamada tasa de error de bits, representa la cantidad de bits de salida erróneos en promedio. Esta tasa fue propuesta como una variación de ER para evaluar aproximaciones con múltiples salidas. El cálculo del MHD se presenta en la ecuación 2.2.

$$MHD = \frac{1}{N} \sum_{\forall x \in N} \sum_{i=0}^{n-1} f_i(x) \oplus f'_i(x) \tag{2.2}$$

Peor distancia de Hamming

La peor distancia de Hamming (Worst Hamming Distance, WHD) representa la mayor distancia de Hamming entre las salidas originales y las aproximadas. Se utiliza para verificar cuál es el peor caso en cantidad de bits de salida erróneos. El cálculo del WHD se presenta en la ecuación 2.3.

$$WHD = \max_{\forall x \in N} \sum_{i=0}^{n-1} f_i(x) \oplus f'_i(x) \tag{2.3}$$

Error en el peor caso

El error en el peor caso (Worst Case Error, WCE) representa la mayor diferencia absoluta en magnitud entre las salidas del circuito original y el aproximado. Es la métrica más comúnmente utilizada en el diseño aritmético [18]. El cálculo del WCE se presenta en la ecuación 2.3.

$$WHD = \max_{\forall x \in N} \left| f(x) - f'(x) \right| \tag{2.4}$$

Error absoluto medio

El error absoluto medio (Mean Absolute Error, MAE) representa la diferencia en magnitud media entre las salidas del circuito original y el aproximado. El cálculo del MAE se presenta en la ecuación 2.5.

$$MAE = \frac{1}{N} \sum_{\forall x \in N} \left| f(x) - f'(x) \right| \tag{2.5}$$

Error relativo medio

El error relativo medio (Mean Relative Error, MRE) es similar al MAE, pero el resultado se presenta como un porcentaje y es relativo a las salidas del circuito original. Formalmente, representa la diferencia en magnitud media relativa entre las salidas del circuito original y el aproximado. La ventaja de utilizar un método de error relativo es que, como el valor de error es un porcentaje, es posible analizar circuitos con cantidades de entradas diferentes.

El cálculo del MRE se presenta en la ecuación 2.6. Se nota que si $f(x)$ tiene un valor de 0, la ecuación se vuelve indefinida; por lo tanto, para aplicaciones prácticas se recomienda reemplazar el divisor con la función máx $(1, |f(x)|)$.

$$MRE = \frac{1}{N} \sum_{\forall x \in N} \frac{|f(x) - f'(x)|}{|f(x)|} \tag{2.6}$$

Error cuadrático medio

El error cuadrático medio (Mean Squared Error, MSE) representa la diferencia en magnitud cuadrática media entre las salidas del circuito original y el aproximado. Se presenta su cálculo en la ecuación 2.7. Aunque el MAE suele ser más intuitivo de interpretar, el MSE ofrece una visión más completa, ya que corresponde con el concepto de varianza de los errores. Además, el MSE es computacionalmente menos costoso de calcular, lo que lo hace especialmente útil cuando se están manejando grandes cantidades de datos.

$$MSE = \frac{1}{N} \sum_{\forall x \in N} \big(f(x) - f'(x)\big)^2 \tag{2.7}$$

2.2. Aprendizaje automático

Según Russel y Norvig [21], en el aprendizaje automático una computadora analiza datos, construye un modelo basado en esos datos y luego usa ese modelo tanto como una hipótesis sobre el mundo como una herramienta de software para resolver problemas.

En el contexto de ALS, una de las formas que más nos interesa aplicar ML es que la computadora analice los datos de la tabla de verdad de un circuito, de modo que se construya una representación generalizada del comportamiento de dicho circuito. Este modelo actúa como una hipótesis sobre lo que haría el circuito original, y aunque normalmente en otros enfoques de ML se mantendría como una solución en software, en nuestro caso buscamos transformarlo en hardware. De esta forma, el modelo sigue funcionando como una hipótesis de lo que haría el circuito original, pero implementada directamente como un circuito aproximado.

2.2.1. Aprendizaje supervisado

En el aprendizaje supervisado, el agente observa parejas de entradas y salidas y aprende una función que mapea las entradas a sus correspondientes salidas [21].

Es decir, en el aprendizaje supervisado, se dispone de los datos del comportamiento que se desea aprender antes del entrenamiento.

Un ejemplo de esto, en el contexto de ALS, es cuando se entrena un modelo utilizando la tabla de verdad de un circuito, que es un conjunto de datos que muestra las salidas correspondientes a sus entradas.

2.2.2. Aprendizaje reforzado

El aprendizaje reforzado (Reinforcement Learning, RL) es aprender qué hacer o cómo asociar situaciones con acciones, para maximizar una señal de recompensa. No se le dice al agente qué acciones tomar, sino que debe descubrir cuáles acciones dan la mayor recompensa mediante prueba y error. [22]

En el contexto de ALS, un agente RL podría ser entrenado para aprender a modificar un circuito, realizando modificaciones y recibiendo una recompensa basada en parámetros como la reducción en área, en tiempo de ejecución, o el aumento en el error del circuito. Así el agente, después de mucho entrenamiento se convertirá en un experto en realizar modificaciones a circuitos para generar versiones aproximadas de ellos.

2.2.3. Generalización

Según Bishop [23], en el contexto de ML, la generalización es la capacidad de un modelo para producir respuestas correctas ante entradas no vistas durante el entrenamiento. En la práctica, el espacio de posibles entradas suele ser tan amplio que el conjunto de entrenamiento solo representa una pequeña fracción del total.

Por ejemplo, un modelo entrenado para distinguir entre fotos de gatos y perros no puede haber visto todas las imágenes de estos animales en la existencia; debe aprender patrones generales (es decir, debe generalizar), para poder clasificar imágenes nuevas.

En el contexto de la síntesis lógica aproximada (ALS), si bien es posible generar todas las combinaciones de entrada/salida de la tabla de verdad para circuitos pequeños, esto se vuelve inviable a medida que el número de entradas crece. Un circuito con 64 bits de entrada tiene 2^{64} combinaciones posibles, aproximadamente $1{,}8 \times 10^{19}$, lo que hace imposible utilizar su tabla de verdad completa. Por tanto, el modelo debe aprender a generalizar a partir de un subconjunto representativo. Incluso en circuitos pequeños, donde es posible entrenar modelos con una tabla de verdad exhaustiva, el modelo no memoriza los datos, sino que generaliza al construir una representación abstracta del circuito basada en patrones presentes en los ejemplos.

2.2.4. Sobreajuste

Se dice que una función aprendida tiene sobreajuste cuando presta demasiada atención al conjunto de datos con el que fue entrenada, lo que provoca un mal desempeño con datos no vistos [21].

En el contexto de aprendizaje de circuitos, el sobreajuste puede ocurrir si un modelo memoriza las entradas y salidas de un subconjunto limitado de la tabla de verdad, sin capturar el comportamiento general del circuito. Como resultado, puede fallar al predecir

correctamente salidas para combinaciones de entrada no vistas, aunque estas sigan la misma lógica.

2.2.5. Validación de modelos

La validación de modelos es el proceso de confirmar que un modelo produce resultados suficientemente correctos para el propósito con el que fue diseñado, dentro del dominio en el que se aplica [24].

En el dominio de ML, esta validación comúnmente se lleva a cabo separando los datos de entrenamiento en 3 conjuntos:

- Un conjunto de entrenamiento, utilizado para entrenar los modelos
- Un conjunto de validación, que se utiliza para evaluar los modelos entrenados y ver que generalicen bien y no tengan sobreajuste. Se puede utilizar para entrenar múltiples modelos similares y escoger el que tenga mejor rendimiento.
- Un conjunto de prueba para realizar una evaluación final e imparcial del modelo [21].

A pesar de los nombres utilizados para los conjuntos, la validación final de la efectividad del modelo tendrá que venir del conjunto de prueba.

En el caso de aprender el comportamiento de un circuito tenemos una situación única que no se da comúnmente en ML. Cuando el circuito es pequeño, se pueden conocer todos sus posibles pares de entrada/salida. Por lo que en estos casos se puede entrenar y validar el modelo con el conjunto completo de entradas/salidas posibles del circuito.

2.2.6. Modelos y técnicas

En esta sección se explican los modelos más populares en aplicaciones de ALS, así como los métodos utilizados para adaptarlos a este dominio.

Árboles de decisión (DT)

Un árbol de decisión es una representación de una función que mapean un vector de atributos a un solo valor de salida, una "decisión". Utilizando términos que generalmente se utilizan para estructuras de grafo en forma de árbol, un árbol de decisión llega a su decisión realizando una secuencia de pruebas, iniciando en la raíz del árbol y siguiendo las ramas apropiadas hasta llegar a una hoja. Cada nodo interno del árbol corresponde a una prueba del valor de uno de sus atributos de entrada, las ramas del nodo son etiquetadas con los posibles valores del atributo, y los nodos hoja corresponden con los valores que puede retornar la función.

En el caso general, los valores de entrada y salida pueden ser discretos o continuos. Cuando las entradas son valores discretos y las salidas solo pueden tener dos valores, falso o verdadero, se le llama clasificación Booleana [21].

En la Figura 2.2 se puede observar la estructura de un árbol de decisión booleano.

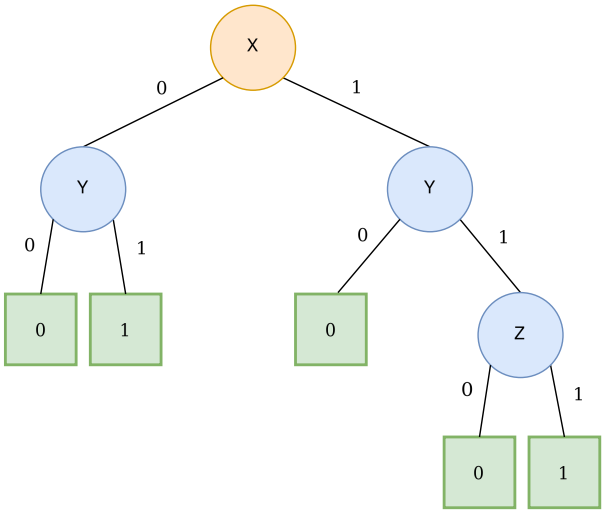

Figura 2.2: Árbol de decisión booleano con tres variables de entrada. El nodo naranja es la raíz del árbol, los nodos azules los nodos intermedios y los nodos verdes son las hojas del árbol, los cuales representan la decisión a la que llega.

En el caso de aprender el comportamiento de un circuito nos enfrentamos a un problema de clasificación booleana: dependiendo de las entradas del circuito, cuál debe ser la salida. Por esto es que, dentro del contexto de ALS, son de particular interés los árboles de decisión booleanos.

Para transformar un modelo DT booleano a una expresión booleana que puede ser expresada con componentes de un circuito se puede emplear el teorema de expansión de Boole [25]. Aplicando este teorema al árbol de la figura 2.2, este se puede representar con la expresión booleana $(\bar{X}Y) + (XY\bar{Z})$. Un árbol de decisiones no necesariamente se encontrará en la configuración más óptima para expresar la función booleana que implementa, en el caso de la Figura 2.2 la expresión que obtuvimos se puede simplificar más para obtener $Y(\bar{X} + \bar{Z})$. Por este motivo, se recomienda aplicar técnicas de simplificación a la expresión booleana o al circuito derivado de ella.

Un solo DT booleano tiene una sola salida, por lo que para circuitos con múltiples salidas se debe de aplicar un DT que aprenda la función individual de cada salida del circuito.

Bosque aleatorio (RF)

Es común crear modelos que son colecciones de hipótesis y combinar las predicciones de cada hipótesis, puede escogerse un promedio de las predicciones, tratarlas como un voto de mayoría o aplicar otras técnicas de ML para obtener el resultado final. A las hipótesis individuales se les llama modelo base y su combinación modelo de conjunto. Crear estos modelos de conjunto puede ser beneficioso para disminuir el sobreajuste y sesgo del modelo base. Los bosques aleatorios son de los modelos de conjunto más

comúnmente aplicados, son formados por DT como modelo base. [21]

En el caso de un RF conformado por DT booleanos, se determina la salida del bosque con un voto de mayoría por parte de los árboles. Para implementar este voto por mayoría en un circuito se pueden utilizar sumadores y un comparador.

Perceptrón multicapa / Red neuronal

Las redes neuronales tienen sus orígenes en [26], donde con inspiración biológica se intentó modelar la red de neuronas en el cerebro con circuitos computacionales. A pesar de sus orígenes, al utilizar redes neuronales en el contexto de ML el realismo biológico impondría restricciones innecesarias que limitarían su practicabilidad. Por esto en la actualidad las redes neuronales no se tratan como un modelo biológico, sino que son de interés como modelos puramente estadísticos para el reconocimiento de patrones.

Existen muchos modelos de redes neuronales, pero uno de particular interés es el perceptrón multicapa. Este uno de los tipos de red neuronal más conocida y utilizada [27].

La arquitectura de perceptrón multicapa se caracteriza por formar un grafo acíclico dirigido de nodos llamados neuronas artificiales. Los nodos se organizan en capas, como se puede observar en la Figura 2.3. Hay capas de nodos designadas de entrada y de salida, cada nodo le aplica un procesamiento a sus entradas y pasa los valores resultantes a sus sucesores en la red.

Típicamente, el procesamiento de cada nodo es tomar una suma ponderada de sus nodos predecesores y aplicarle una función de activación, la cual en perceptrones multicapa modernos es continua y no lineal. Sea a_j la salida del nodo j y sea $w_{i,j}$ el factor de ponderación para la conexión entre el nodo i y el nodo j; entonces se tiene

$$a_j = g_j\left(\sigma_i w_{i,j} a_i\right),$$

donde g_j es la función de activación no lineal del nodo j. [21]

Para profundizar en el tema del funcionamiento y entrenamiento de perceptrones multicapa, se recomienda consultar el Capítulo 5 de [23].

No es fácil traducir MLP booleanos a circuitos. Un método sencillo es sintetizar módulos aritméticos para realizar las computaciones necesarias, como sumadores y multiplicadores. Sin embargo, esto resulta en circuitos muy grandes [10]. En [11], el equipo 3 utiliza un método donde podan conexiones de baja importancia para reducir el tamaño de la red y cada nodo del MLP es transformado en una LUT a través de cuantización de su salida, evitando utilizar circuitos aritméticos complejos. Se recomienda leer el apéndice de la referencia para más detalles.

Programación genética cartesiana (CGP)

El término CGP aparece originalmente en [28] como un método de programación genética para aprender funciones booleanas. El cromosoma utilizado por el algoritmo representa una matriz rectangular de funciones primitivas con sus respectivas conexiones y funcionalidades, lo cual constituye una representación de un circuito. Se le denomina

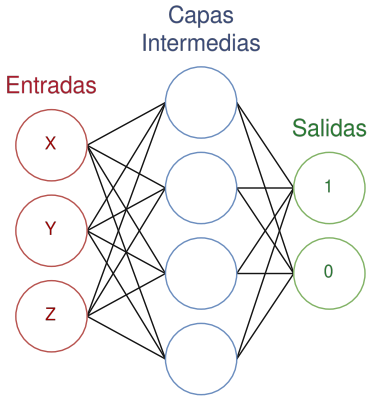

Figura 2.3: Estructura general de un perceptrón multicapa con salidas booleanas. Puede llegar a tener más capas intermedias. La salida se decide típicamente interpretando el valor de cada nodo de salida como un valor de confianza de que esa sea la salida y se escoge la salida con un mayor valor.

"cartesiana" porque organiza los nodos en una rejilla bidimensional identificada mediante coordenadas cartesianas.

Un gran beneficio de CGP es que el proceso evolutivo puede empezar desde un circuito aleatorio o uno predefinido, lo que permite trabajar a partir de circuitos encontrados por otros métodos [15]. Esto lo hace fácil de combinar con técnicas existentes. Otra ventaja es que su cromosoma ya representa directamente un circuito, por lo que convertirlo en uno es un proceso simple.

3 Marco metodológico

4 Descripción del trabajo realizado

4.1. Descripción del proceso de solución

4.2. Análisis de los resultados obtenidos

5 Conclusiones y recomendaciones

Bibliografía

[1] Tecnológico de Costa Rica. «Reseña del TEC.» (), dirección: https://www.tec.ac.cr/resena-tec (visitado 04-04-2025).

[2] J. Castro-Godinez, H. Barrantes-Garcia, M. Shafique y J. Henkel, «AxLS: A framework for approximate logic synthesis based on netlist transformations,» *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, n.º 8, págs. 2845-2849, ago. de 2021, ISSN: 1549-7747, 1558-3791. DOI: 10.1109/TCSII.2021.3068757. dirección: https://ieeexplore.ieee.org/document/9386244/ (visitado 07-02-2025).

[3] G. Pasandi, S. Nazarian y M. Pedram, «Approximate logic synthesis: A reinforcement learning-based technology mapping approach,» en *20th International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, USA: IEEE, mar. de 2019, págs. 26-32, ISBN: 978-1-7281-0392-1. DOI: 10.1109/ISQED.2019.8697679. dirección: https://ieeexplore.ieee.org/document/8697679/ (visitado 07-02-2025).

[4] G. Pasandi, M. Peterson, M. Herrera, S. Nazarian y M. Pedram, «Deep-PowerX: A deep learning-based framework for low-power approximate logic synthesis,» en *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, Boston Massachusetts: ACM, 10 de ago. de 2020, págs. 73-78, ISBN: 978-1-4503-7053-0. DOI: 10.1145/3370748.3406555. dirección: https://dl.acm.org/doi/10.1145/3370748.3406555 (visitado 16-12-2024).

[5] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu y L. Shi, «Timing-Driven Technology Mapping Approximation Based on Reinforcement Learning,» *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, n.º 9, págs. 2755-2768, sep. de 2024, ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2024.3379016. dirección: https://ieeexplore.ieee.org/document/10477245/ (visitado 21-02-2025).

[6] M. A. Rajput, S. Alyami, Q. A. Ahmed, H. Alshahrani, Y. Asiri y A. Shaikh, «Improved Learning-Based Design Space Exploration for Approximate Instance Generation,» *IEEE Access*, vol. 11, págs. 18 291-18 299, 2023, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3247303. dirección: https://ieeexplore.ieee.org/document/10049435/ (visitado 21-02-2025).

[7] M. Awais, H. G. Mohammadi y M. Platzner, «DeepApprox: Rapid Deep Learning based Design Space Exploration of Approximate Circuits via Check-pointing,» en *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Knoxville, TN, USA: IEEE, 1 de jul. de 2024, págs. 88-93, ISBN: 979-8-3503-5411-9. DOI: 10.1109/ISVLSI61997.2024.00027. dirección: https://ieeexplore.ieee.org/document/10682776/ (visitado 21-02-2025).

[8] S. Boroumand, C.-S. Bouganis y G. A. Constantinides, «Learning boolean circuits from examples for approximate logic synthesis,» en *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, Tokyo Japan: ACM, 18 de ene. de 2021, págs. 524-529, ISBN: 978-1-4503-7999-1. DOI: 10.1145/3394885.3431559. dirección: https://dl.acm.org/doi/10.1145/3394885.3431559 (visitado 07-02-2025).

[9] B. A. De Abreu, A. Berndt, I. S. Campos et al., «Fast Logic Optimization Using Decision Trees,» en *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, Daegu, Korea: IEEE, mayo de 2021, págs. 1-5, ISBN: 978-1-7281-9201-7. DOI: 10.1109/ISCAS51556.2021.9401664. dirección: https://ieeexplore.ieee.org/document/9401664/ (visitado 14-03-2025).

[10] Y. Miyasaka, X. Zhang, M. Yu, Q. Yi y M. Fujita, «Logic Synthesis for Generalization and Learning Addition,» en *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France: IEEE, 1 de feb. de 2021, págs. 1032-1037, ISBN: 978-3-9819263-5-4. DOI: 10.23919/DATE51398.2021.9474169. dirección: https://ieeexplore.ieee.org/document/9474169/ (visitado 25-03-2025).

[11] S. Rai, W. L. Neto, Y. Miyasaka et al., «Logic synthesis meets machine learning: Trading exactness for generalization,» en *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France: IEEE, 1 de feb. de 2021, págs. 1026-1031, ISBN: 978-3-9819263-5-4. DOI: 10.23919/DATE51398.2021.9473972. dirección: https://ieeexplore.ieee.org/document/9473972/ (visitado 07-02-2025).

[12] W. Zeng, A. Davoodi y R. O. Topaloglu, «Sampling-Based Approximate Logic Synthesis: An Explainable Machine Learning Approach,» en *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Munich, Germany: IEEE, 1 de nov. de 2021, págs. 1-9, ISBN: 978-1-6654-4507-8. DOI: 10.1109/ICCAD51958.2021.9643484. dirección: https://ieeexplore.ieee.org/document/9643484/ (visitado 20-02-2025).

[13] Y.-S. Huang y J.-H. R. Jiang, «Circuit Learning: From Decision Trees to Decision Graphs,» *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, n.º 11, págs. 3985-3996, nov. de 2023, ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2023.3258747. dirección: https://ieeexplore.ieee.org/document/10075520/ (visitado 14-03-2025).

[14] H. Hu y S. Cai, *OPTDTALS: Approximate Logic Synthesis via Optimal Decision Trees Approach*, Version Number: 1, 2024. DOI: 10.48550/ARXIV.2408.12304. dirección: https://arxiv.org/abs/2408.12304 (visitado 14-03-2025).

[15] A. A. S. Berndt, B. Abreu, I. S. Campos et al., «CGP-based Logic Flow: Optimizing Accuracy and Size of Approximate Circuits,» *Journal of Integrated Circuits and Systems*, vol. 17, n.º 1, págs. 1-12, 30 de abr. de 2022, ISSN: 1872-0234, 1807-1953. DOI: 10.29292/jics.v17i1.546. dirección: https://jics.org.br/ojs/index.php/JICS/article/view/546 (visitado 14-03-2025).

[16] J. C. Prats Ramos, N. Sachetti, A. Berndt, J. T. Carvalho y C. Meinhardt, «Impact on Delay, Power and Area of Machine Learning-based Approximate Logic Synthesis,» en *2024 37th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, Joao Pessoa, Brazil: IEEE, 2 de sep. de 2024, págs. 1-5, ISBN: 979-8-3503-9169-5. DOI: 10.1109/SBCCI62366.2024.10703989. dirección: https://ieeexplore.ieee.org/document/10703989/ (visitado 14-03-2025).

[17] J. Han y M. Orshansky, «Approximate computing: An emerging paradigm for energy-efficient design,» en *2013 18TH IEEE EUROPEAN TEST SYMPOSIUM (ETS)*, Avignon, France: IEEE, mayo de 2013, págs. 1-6, ISBN: 978-1-4673-6377-8 978-1-4673-6376-1. DOI: 10.1109/ETS.2013.6569370. dirección: http://ieeexplore.ieee.org/document/6569370/ (visitado 20-04-2025).

[18] G. Ammes, P. F. Butzen, A. I. Reis y R. Ribas, «Two-Level and Multilevel Approximate Logic Synthesis,» *Journal of Integrated Circuits and Systems*, vol. 17, n.º 3, págs. 1-14, 31 de dic. de 2022, ISSN: 1872-0234, 1807-1953. DOI: 10.29292/jics.v17i3.661. dirección: https://jics.org.br/ojs/index.php/JICS/article/view/661 (visitado 07-02-2025).

[19] R. M. Barr, «An investigation of the symmetric properties of logical functions,» Tesis doct., Monterey, California: US Naval Postgraduate School, 1960.

[20] A. A. S. Berndt, M. Fogaça y C. Meinhardt, «Review of Machine Learning in Logic Synthesis,» *Journal of Integrated Circuits and Systems*, vol. 17, n.º 3, págs. 1-12, 31 de dic. de 2022, ISSN: 1872-0234, 1807-1953. DOI: 10.29292/jics.v17i3.649. dirección: https://jics.org.br/ojs/index.php/JICS/article/view/649 (visitado 21-02-2025).

[21] S. J. Russell y P. Norvig, *Artificial intelligence: a modern approach*. pearson, 2016, págs. 651, 653, 655, 657, 666, 696-697, 751-752.

[22] R. S. Sutton y A. Barto, «Reinforcement learning: an introduction,» 2018, Publisher: The MIT Press.

[23] C. M. Bishop, *Pattern recognition and machine learning* (Information science and statistics). New York: Springer, 2006, pág. 2, 738 págs., ISBN: 978-0-387-31073-2.

[24] S. Schlesinger, «Terminology for model credibility,» *SIMULATION*, vol. 32, n.º 3, págs. 103-104, mar. de 1979, Publisher: SAGE Publications, ISSN: 0037-5497, 1741-3133. DOI: 10.1177/003754977903200304. dirección: https://journals.sagepub.com/doi/10.1177/003754977903200304 (visitado 10-04-2025).

[25] G. Boole, *An Investigation of the Laws of Thought on which are Founded the Mathematical Theories of Logic and Probabilities*. Walton y Maberly, 1854, pág. 72, 451 págs.

[26] W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity,» *The bulletin of mathematical biophysics*, vol. 5, págs. 115-133, 1943, Publisher: Springer.

[27] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu y N. Mastorakis, «Multilayer perceptron and neural networks,» *WSEAS Transactions on Circuits and Systems*, vol. 8, n.º 7, págs. 579-588, 2009.

[28] J. F. Miller et al., «An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach,» en *Proceedings of the genetic and evolutionary computation conference*, vol. 2, 1999, págs. 1135-1142.

6 Apéndices y anexos