Extracción de reglas de redes neuronales *feedforward* entrenadas con lógica de primer orden

Extracting rules from trained feedforward neural networks with first order logic

Pablo Negro[1] Claudia Pons[1, 2, 3]

1. CAETI – Centro de altos estudios en tecnología informática, facultad de tecnología. UAI.
2. LIFIA – Facultad de Informática. Universidad Nacional de La Plata.
3. CIC – Comisión de Investigaciones Científicas de Buenos Aires CIC-PBA.

Resumen. La necesidad de integración neural-simbólica se hace evidente a medida que se abordan problemas más complejos, y que van más allá de tareas de dominio limitadas como lo es la clasificación. Los métodos de búsqueda para la extracción de reglas de las redes neuronales funcionan enviando combinaciones datos de entrada que activan un conjunto de neuronas. Ordenando adecuadamente los pesos de entrada de una neurona, es posible acotar el espacio de búsqueda. Con base en esta observación, este trabajo tiene por objetivo presentar un método para extraer el patrón de reglas aprendido por una red neuronal entrenada *feedforward*, analizar sus propiedades y explicar estos patrones a través del uso de lógica de primer orden (FOL).

Palabras claves: Deep Learning, Extracción de reglas, Inteligencia Artificial, Lógica

Abstract. The need for neural-symbolic integration becomes evident as more complex problems are addressed, and they go beyond limited domain tasks such as classification. Search methods for extracting rules from neural networks work by sending input data combinations that activate a set of neurons. By properly ordering the input weights of a neuron, it is possible to narrow down the search space. Based on this observation, this work aims to present a method for extracting the pattern of rules learned by a trained feedforward neural network, analyzing its properties and explaining these patterns through the use of first-time logic (FOL).

Keywords: Deep Learning, Rules Extraction, Artificial Intelligence, Logic

1 Introducción

Con la mejora de la tecnología de almacenamiento de datos, ha habido un interés creciente en extraer conocimiento de los datos. Idealmente, el conocimiento descubierto debería ser preciso y comprensible para el usuario. Una de las dificultades para la extracción de conocimiento preciso es que los datos que se extraen pueden tener mucho ruido. En estos casos, las redes neuronales son una solución viable, debido a su relativamente buena tolerancia al ruido y capacidad de generalización (Santos et al., 2000).
Se ha demostrado empíricamente que las redes neuronales artificiales (ANN) funcionan bien en varios problemas de aprendizaje automático. Las redes multicapa *feedforward* entrenadas con el algoritmo *backpropagation* (Rumelhart et al., 1986) se consideran el método más eficiente en este respecto. Respuestas razonablemente satisfactorias a preguntas como cuántos ejemplos se necesitan para que una red neuronal *feedforward* profunda aprenda un concepto y cuál es la mejor arquitectura de red neuronal para un dominio de problema particular (dado un número fijo de ejemplos de entrenamiento) se encuentran disponibles, por lo que ahora es posible entrenar redes neuronales de un modo más eficiente. Esto hace que las redes neuronales sean una excelente herramienta para la minería de datos (Britos, 2005), donde el enfoque es aprender las relaciones entre los datos que se almacenan en grandes volúmenes.
Sin embargo, es bien sabido que las redes neuronales suelen representar su *conocimiento* en forma de pesos numéricos e interconexiones distribuidos, lo cual hace que este proceso no sea comprensible para el usuario. Esto representa un problema serio, dado que el usuario no es capaz de entender la información a la salida de la red o razonar a cerca del proceso cognitivo. Por lo tanto, el usuario tendría que confiar ciegamente en la respuesta dada por la red, lo cual es claramente indeseable en varios dominios de aplicación, (i.e., en el diagnóstico médico de enfermedades mortales, donde hay vidas en juego). Además, si el usuario no puede comprender ni validar conocimiento descubierto, podría decidir ignorarlo, lo cual podría conducir a toma de decisiones no deseadas.
Las redes neuronales son típicamente *cajas negras*. Los cálculos realizados por las capas sucesivas rara vez corresponden a pasos de razonamiento humanamente comprensibles, y los vectores intermedios de activaciones que se generan carecen de una semántica humanamente comprensible (Nielsen et al., 2022). Entonces, debido a su estructura anidada y no lineal las hace muy poco transparentes, es decir, no está claro qué información en los datos de entrada las hace llegar realmente a sus decisiones. Por lo tanto, estos modelos suelen considerarse *cajas negras.*
La explicabilidad aborda el problema crítico de que los humanos no pueden comprender directamente el comportamiento complejo de las ANNs, especialmente cuando tienen numerosas capas (i.e., redes neuronales profundas o DNN por sus siglas en inglés), o explicar su proceso de toma de decisiones subyacente. La explicabilidad en DNN es el requisito fundamental para *generar confianza* con los usuarios y es la *clave para su implementación segura, justa y exitosa en aplicaciones del mundo real*. En este sentido, existen aplicaciones como aprobación de crédito y diagnóstico médico donde es importante explicar el razonamiento de la red neuronal. La principal crítica contra las redes neuronales en tales dominios es que el proceso de toma de decisiones es difícil de entender. Esto se debe a que el conocimiento en la red neuronal se almacena como

parámetros de valor real (pesos y sesgos) de la red, el conocimiento se codifica de forma distribuida y el mapeo aprendido por la red puede ser no lineal y no monótono.
En este punto cabria cuestionar por qué se deben usar redes neuronales cuando la comprensibilidad es un tema importante. La razón es que la precisión predictiva también es muy importante y las redes neuronales tienen un sesgo inductivo apropiado para muchos dominios de aprendizaje automático (Krishnan et al., 1999a).
Las precisiones predictivas obtenidas con las redes neuronales suelen ser significativamente más altas que las obtenidas con otros paradigmas de aprendizaje. Investigaciones recientes sobre la comprensión del funcionamiento de una red neuronal entrenada se ha centrado en la extracción de reglas simbólicas ((AmirHosseini & Hosseini, 2019; Csiszár et al., 2020; Mahdavifar & Ghorbani, 2020)).
La tarea de extracción de reglas puede verse como una tarea de búsqueda o como una tarea de aprendizaje (Montavon et al., 2017), donde para el enfoque de búsqueda, las reglas se extraen a nivel de las neuronas individuales (ocultas y de salida) en la red, observando sus pesos y sesgos (Krishnan et al., 1999a).
En tal sentido, uno de los principales problemas con el enfoque de búsqueda es cómo *restringir el espacio de búsqueda* para las posibles combinaciones de reglas. En este artículo se presenta un nuevo método para restringir este espacio de búsqueda y se analizan algunas propiedades de este método. En la Sección 2, se explica el problema de la extracción de reglas. En la Sección 3 se discute el algoritmo AREBI para la extracción de reglas y se analizan sus propiedades en la Sección 4. La Sección 5 da un breve resumen del trabajo relacionado y concluye el documento.

2 El Problema de la extracción de reglas

En una red neuronal entrenada, el conocimiento adquirido en la fase de entrenamiento está codificado en la arquitectura de la red, las funciones de activación utilizadas y los pesos y sesgos de las neuronas. En tal sentido, el rendimiento de una red neuronal está directamente relacionado con su arquitectura y parámetros. Por tanto, la elección de una arquitectura para una red neuronal influye en el tiempo de aprendizaje, la precisión predictiva, la tolerancia al ruido y la capacidad de generalización de la red (Santos et al., 2000). La tarea de extracción de reglas consiste en utilizar una o más de las piezas de información anteriores y extraer un conjunto de reglas de las neuronas. Consideramos el caso donde las entradas a la red *feedforward* son tabulares y las salidas son booleanas (es decir, un problema de clasificación).
Las neuronas en las redes neuronales *feedforward* tienen activaciones definidas por:

$$Z_i = (\sum_j W_{i,j} . A_j) + \beta_i \quad \text{(eq1)}$$

$$A_i = Act(Z_i) \quad \text{(eq2)}$$

$$Act(x) = \frac{1}{1 + e^{-\alpha x}} \quad \text{(eq3)}$$

Donde A_i es la activación de la neurona *i*, $W_{i,j}$ es el peso en el enlace de la neurona *j* a la neurona *i*, A_j es la activación de la neurona *j*, β_i es el sesgo en la neurona i, Act() es una función de activación no lineal (la función sigmoidal), y α es una parámetro que controla la pendiente de la función sigmoidal.

En general, los métodos de búsqueda para extraer reglas intentan encontrar combinaciones de los valores de entrada a una neurona que dan como resultado que tenga una activación cercana a 1 (para una regla de confirmación) o una activación cercana a 0 (para una regla de no confirmación). Para que una neurona tenga una activación cercana a 1, se busca encontrar tales combinaciones de pesos tal que la cantidad en la (eq.2) $A_i \approx 1$. De manera similar, para una activación neuronal cercana a 0, deberíamos tener $A_i \approx 0$. El valor de A_i en el que $Act(Z_i) \approx 1$ se denomina el *umbral* de la neurona.
El proceso de encontrar reglas para las neuronas de la capa oculta se simplifica porque las entradas son booleanas. En este caso, para evaluar A_i en la (eq.1), debemos considerar solo los pesos que alimentan la neurona. El proceso de encontrar reglas para las neuronas de la capa de salida es más complejo dado que la neurona de la capa oculta podría tener cualquier activación en el intervalo [0, 1]. Sin embargo, se puede hacer que la neurona de la capa oculta se aproxime a una neurona booleana controlando la inclinación de la función de activación. Al aumentar el valor del parámetro a en la (eq.2), podemos hacer que la neurona de la capa oculta se aproxime a una activación booleana. Establecer el valor del parámetro α en un valor alto ($\approx$10) asegura que todas las neuronas de la capa oculta en la red tendrán una activación cercana a 0 o cercana a 1.
Esto nos permite entonces tratar las salidas de las neuronas de la capa oculta como cantidades booleanas y centrarnos solo en sus pesos para las neuronas de la capa de salida en el proceso de extracción de reglas. Las reglas extraídas deben ser *válidas* (las reglas deben cumplirse independientemente de los valores de las variables no mencionadas en las reglas), deben ser lo *más generales* posible (si se elimina alguno de los antecedentes, la regla ya no debería ser válida) y *completa* (se deben extraer todas las reglas válidas y máximamente generales posibles).

3 Alcance

Este trabajo se centra en redes neuronales profundas *feedforward* entrenadas clásicas, dado que el foco está puesto en la extracción de reglas a partir de las matrices de pesos de estas. Quedan fuera de este estudio variaciones de este tipo de redes tales como las redes neuronales recurrentes, y los Transformers que, si bien hacen uso de las redes neuronales *feedforward* presentan arquitecturas diferentes dado que utilizan capas de atención, *embeedings* y transformaciones creando arquitecturas hibridas más complejas. Entonces, al referir a tecnologías de Deep Learning, referimos a los conceptos vertidos por Goodfellow et al., (2016) y Russell & Norvig, (2010), o a una definición equivalente presentadas en Domingos, (2018) donde las redes neuronales profundas son presentadas como capas de neuronas densas, las cuales se combinan con capas de entrada y salida, y alguna función de activación en sus neuronas.

4 El método propuesto

En esta sección, explicamos el concepto de un árbol de combinación y cómo se usa en la generación de reglas. Para tratar uniformemente los pesos positivos y negativos de la neurona, se ha adoptado una transformación admisible de pesos utilizada por primera vez por (Sethi & Yoo, 1996) y adaptada en (Krishnan et al., 1999), para convertir todos

los pesos negativos de la neurona en cantidades positivas. Esta transformación nos permite trabajar solo con pesos positivos. Para tal propósito hemos utilizado una red *feed-forward* de tres capas entrenada usando la regla de *backpropagation* (Goodfellow et al., 2016).
Una regla de *confirmación* es aquella que explica cuándo se activa una neurona; una regla de *des confirmación* explica cuándo una neurona se apaga. Explicaremos en detalle el procedimiento para extraer una regla de confirmación; Se aplica un procedimiento similar para las reglas contrarias.
Como se mencionó en la Sección 1, una de las cuestiones más cruciales en el desarrollo de un algoritmo de extracción de reglas es cómo restringir el tamaño del espacio de solución buscado. Supongamos que una neurona en la red tiene cuatro pesos positivos etiquetados como 1, 2, 3 y 4, respectivamente. Con este vector de cuatro pesos podemos formar $\sum_{i=0}^{4} {}^{4}C_i$ combinaciones. Si ignoramos las combinaciones nulas (i.e. ${}^{4}C_0$), el resto de las combinaciones se pueden considerar como nodos de un árbol con una combinación de tamaño *i* en el *i-ésimo* nivel del árbol. La figura 1 muestra el árbol de combinaciones para una neurona que tiene cuatro pesos positivos. En lugar de intentar combinaciones aleatorias de pesos, primero ordenamos los pesos y luego generamos *combinaciones de todos los tamaños posibles*. Las combinaciones para cualquier tamaño en particular se ordenan en orden descendente de la suma de los pesos en la combinación. Debido al orden de los pesos y a la restricción por la regla de máxima generalidad, es posible excluir algunas combinaciones de la búsqueda. A continuación, se explica cómo el ordenamiento de los pesos de la neurona y considerar las combinaciones en el orden anterior, ayuda a reducir el espacio de búsqueda.

4.1 Poda de la búsqueda en un árbol de combinación

Existen dos tipos de poda que pueden ocurrir en un árbol de combinación.

Podas al mismo nivel del árbol. Si una combinación en cualquier nivel no se cumple, todas las demás combinaciones en este nivel pueden eliminarse. Esto se debe a que todas las combinaciones en el mismo nivel tienen la misma longitud, y debido al orden de los pesos, si falla una combinación en un nivel, todas las demás combinaciones en ese nivel también fallarán, ya que su suma ponderada sumará menos de la combinación que falló. Por lo tanto, no es necesario considerarlos en la búsqueda de reglas.
Podas a niveles más profundos del árbol. Si una combinación en un nivel logra formar una regla, entonces no es necesario considerar las combinaciones en los siguientes niveles de las cuales la presente combinación forma un subconjunto. Aunque estas combinaciones lograrán formar reglas, estas reglas serán subsumidas por las reglas formadas a partir de la combinación actual y pueden ser excluidas debido a la condición de máxima generalidad.
Si Consideramos el árbol de combinación que se muestra en la Fig. 1., vemos que las combinaciones de nivel 2 son 12, 13, 14, 23, 24, 34. Si 12 no genera el antecedente positivo de una regla, no necesitamos probar las combinaciones 13, 14, 23, 24, 34 y estas pueden eliminarse. Por otro lado, si 12 logra formar un antecedente positivo de la regla, entonces todas las combinaciones en su subárbol, es decir, 123, 124 y 1234,

pueden eliminarse ya que formarán reglas más específicas que la combinación 12 y pueden eliminarse debido a la restricción de máxima generalidad.

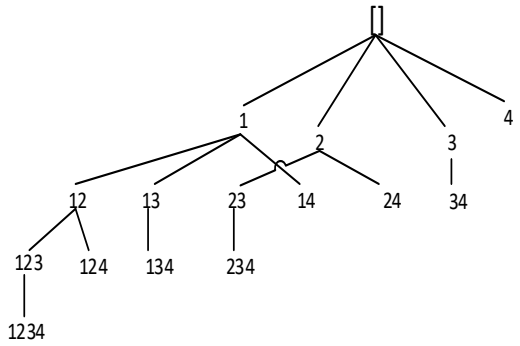

Fig 1. Árbol de combinaciones

4.2 El Algoritmo AREBI para generación de reglas

El algoritmo de extracción de reglas AREBI se basa en la estrategia de poda discutida anteriormente y funciona en tres etapas. En la primera etapa extraemos reglas para las neuronas de la capa de salida. Se considera que las neuronas de la capa oculta representan conceptos booleanos haciendo que su función de activación tenga una alta ganancia como se explica en la Sección 2. En la segunda etapa extraemos reglas para las neuronas de la capa oculta.
En la etapa final, las reglas obtenidas en la primera etapa se reescriben en términos de las reglas obtenidas en la segunda etapa para obtener reglas que expliquen la relación entrada-salida. El algoritmo para extraer una regla de confirmación para una neurona individual es el siguiente.

1. En el primer paso, todos los pesos negativos de la red neuronal se convierten en cantidades positivas. Esto se hace de la siguiente manera. *Para entradas binarias*: Se convierten todos los pesos negativos en pesos positivos usando la siguiente transformación admisible (Krishnan et al., 1999a).
Reemplazamos cada literal de entrada x, que tiene un peso negativo con su literal negado, ¬x. Reemplazamos su peso negativo, ¬w con w y calculamos un valor de umbral nuevo tal que T = $(\sum_{i=1}^{n} w_i)$ / 2, donde n es el tamaño del vector de pesos.

2. Los pesos de la neurona, para la que se requiere una regla, se ordenan en orden descendente.

3. Generar combinaciones de los pesos ordenados en orden ascendente por sus tamaños. Primero, se generan todas las combinaciones de tamaño uno, luego las combinaciones de tamaño dos, y así sucesivamente. Dentro de una combinación de tamaño M, las combinaciones se ordenan de la siguiente manera. Dadas dos combinaciones C_1 y C_2, C_1 sucede antes que C_2 si:

$$\sum_{w_1 \in C_1} W_i > \sum_{w_2 \in C_2} W_i$$

Luego guardamos todas las combinaciones en la *lista no testeada*.

4. Comenzar con combinaciones de tamaño 1. Para la siguiente combinación en la *lista no testeada*, verificar si:

$$\sum W_c + bias\ de\ la\ neurona > umbral\ de\ la\ neorona \qquad \text{(eq 4)}$$

Donde W_c son los pesos para cada combinación. Notar que Act(umbral) ≈ 1, por lo que, si la neurona tiene su valor de activación por encima de este valor, el concepto correspondiente a esta es verdadero. El valor real de este umbral depende del valor de α en la eq2.

a. Si la desigualdad anterior no se satisface, elimine todas las demás combinaciones del mismo tamaño de la lista no probada. Aquí el tamaño de la combinación es el número de pesos que representa. Por ejemplo, la combinación 12 tiene tamaño 2.
b. Si la desigualdad anterior se cumple para la combinación actual, insertar la combinación actual en la *lista de éxito*. Eliminar todas las combinaciones de tamaños mayores de las cuales la combinación actual es un subconjunto. Por ejemplo, en la figura 1, si la combinación 12 satisface la desigualdad anterior, elimine las combinaciones 123 y 1234 de la lista sin probar.

Repetir este paso hasta que no queden combinaciones en la lista de no probados.

5. Con cada combinación en la lista de éxito, formar la regla correspondiente.

Las reglas de conformidad para toda la red son generadas de la siguiente manera.

I. Realice los pasos 1-5 anteriores para todas las neuronas en las capas de salida y ocultas. Las reglas de las neuronas de la capa de salida tendrán las neuronas de la capa oculta como antecedentes, mientras que las neuronas de la capa oculta tendrán las neuronas de la capa de entrada como antecedentes.
II. Volver a escribir el conjunto de reglas anterior de modo que los antecedentes sean las neuronas de la capa de entrada y los consecuentes sean las neuronas de la capa de salida.

Las reglas contrarias son generadas por un método similar. Para ilustrar con ejemplos prácticos cómo funciona el algoritmo, presentamos tres ejemplos. El primero se basa en un ejemplo presentado en (Krishnan et al., 1999a)

Ejemplo 1: Consideremos una neurona cuyo vector de pesos es w = [2, 2, -1, -2] y el umbral T = 0.5. Después de aplicar la transformación admisible, el vector de pesos se convierte en w = [2, 2, 1, 2] y el nuevo valor de umbral pasa a ser T=3.5. Después de

ordenarlo, el vector de pesos se convierte en w= [2, 2, 2, 1]. Los índices de los pesos ordenados en los pesos originales vienen dados por el vector [1, 2, 4, 3] y estos representan los literales x_1, x_2, $\neg x_4$, $\neg x_3$. Esto da como resultado el árbol de combinación que se muestra en la Fig. 2.

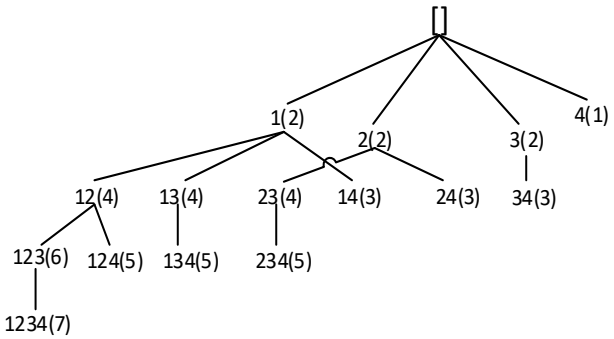

Fig. 2 Árbol de combinación para neurona con vector de pesos w= [2, 2, -1, 2] y T = 0.5

La suma de los pesos en cada contribución se muestra entre paréntesis. Para que se forme una regla, la suma de los pesos en la combinación debe ser mayor o igual que el umbral modificado, a saber, 3.5. En el primer nivel del árbol, la primera combinación tiene una suma ponderada de 2. Como esto es insuficiente para formar una regla, no es necesario considerar el resto de las combinaciones en el mismo nivel. La primera combinación en el nivel 2, 12, tiene una suma ponderada de 4 que supera el umbral de 3,5 y, por lo tanto, puede formar una regla. Las combinaciones del subárbol del cual 12 está en la raíz, es decir, 123, 124 y 1234 se pueden eliminar. Aunque estas combinaciones formarán reglas, estas reglas estarán subsumidas por la regla formada por la combinación 12. Al mismo nivel, las combinaciones 13 y 23 también formarán reglas. Por lo tanto, las combinaciones 134 y 234 también pueden ser podadas. También al mismo nivel, la combinación 14 no logra formar una regla, por lo tanto, todas las demás combinaciones al mismo nivel, es decir, 24 y 34, pueden eliminarse. La expresión booleana para las combinaciones que lograron formar reglas viene dada por:
$f = x_1x_2 \vee x_1\neg x_4 \vee x_2\neg x_4$.

Ejemplo 2: En este ejemplo, extraemos reglas usando AREBI para el problema de un codificador. En este problema, se mapea un conjunto de patrones de entrada ortogonales a un conjunto de patrones de salida ortogonales utilizando un pequeño conjunto de unidades ocultas. La topología de la red neuronal presenta ocho entradas y ocho salidas. La capa oculta tiene tres neuronas. Tanto en la capa oculta como en la capa de salida se utilizó la función sigmoide. La misión es hacer que la red neuronal aprenda un mapeo de identidad. Por ejemplo, si el vector de entrada es [10000000] la salida es también [10000000]. La red básicamente aprende a codificar los ocho patrones en tres bits utilizando las neuronas de la capa oculta. Los pesos de las capas oculta y de salida de la red entrenada se muestran en las Tablas 1 y 2, respectivamente. Las siguientes reglas fueron extraídas para las neuronas de la capa oculta.

- $L^{[1]}_0 = (\neg X6 \wedge X1) \vee (\neg X6 \wedge \neg X3) \vee (X1 \wedge \neg X3)$

- $L^{[1]}_1 = (\neg X0 \wedge X1) \vee (\neg X0 \wedge \neg X3) \vee (X1 \wedge \neg X3)$
- $L^{[1]}_2 = (\neg X6 \vee \neg X3 \vee \neg X0 \vee \neg X1)$

Luego de aplicar las transformaciones admitidas, los umbrales para cada neurona de la capa oculta quedan de la siguiente manera: **Th_0 = 11,5204; Th_1= 12,065; Th_2= 7,4491**

Las siguientes reglas fueron extraídas por AREBI de la capa de salida:

$L^{[2]}_0 = (\neg X1 \vee \neg X2)$ $L^{[2]}_1 = (X1 \wedge X0)$ $L^{[2]}_3 = (\neg X2 \vee \neg X0 \vee \neg X1)$
$L^{[2]}_5 = (\neg X0 \vee \neg X1 \vee X2)$ $L^{[2]}_6 = (\neg X2 \vee \neg X0 \vee X1)$ $L^{[2]}_7 = (\neg X0 \vee X2 \vee \neg X1)$

Los umbrales para cada neurona de la capa de salida quedan de la siguiente manera:

$TL^{[2]}_0 = 18,4005$; $TL^{[2]}_1 = 19,6766$; $TL^{[2]}_3 = 9,5887$; $TL^{[2]}_5 = 0,0495$; $TL^{[2]}_6 = 17,1678$; $TL^{[2]}_{07} = 0,0354$.

Como se puede observar, las neuronas 2 y 4 de la capa de salida no forman formulas bien formadas, con lo cual se las excluyen del análisis. La nomenclatura utilizada para identificar cada componente de la red es que el superíndice entre corchetes representa la capa dentro de la red y el subíndice la ubicación de la neurona dentro de la red.

	$L^{[1]}_0$	$L^{[1]}_1$	$L^{[1]}_2$
$L^{[0]}_0$	5,5636	-6,2065	-3,7184
$L^{[0]}_1$	5,8380	6,0896	-3,3470
$L^{[0]}_2$	0,0017	-0,0006	0,0078
$L^{[0]}_3$	-5,7647	-6,0446	-3,8870
$L^{[0]}_4$	0,0048	0,0042	0,0069
$L^{[0]}_5$	-0,0062	-0,0018	0,0002
$L^{[0]}_6$	-5,8596	5,7814	-3,9270
$L^{[0]}_7$	-0,0020	-0,0010	0,0036
Bias	0,0755	0,1208	7,3547

Tabla 1. Pesos de la capa oculta para el problema del codificador.

	$L^{[2]}_0$	$L^{[2]}_1$	$L^{[2]}_2$	$L^{[2]}_3$	$L^{[2]}_4$	$L^{[2]}_5$	$L^{[2]}_6$	$L^{[2]}_7$
$L^{[1]}_0$	11,6801	13,8896	0,0015	-6,2849	0,0356	-0,0506	-11,4906	-0,0269
$L^{[1]}_1$	-12,8318	14,0866	-0,0231	-6,1982	0,0291	-0,0131	9,8659	-0,017
$L^{[1]}_2$	-12,2891	-11,377	0,0614	-6,6944	0,0343	0,0058	-12,979	0,0269
Bias	6,3483	-8,1153	-0,0268	9,3658	-0,0560	0,0435	7,0913	0,0320

Tabla 2. Pesos de la capa de salida para el problema del codificador.

	$L^{[1]}_0$	$L^{[1]}_1$	$L^{[1]}_2$	$L^{[1]}_3$	$L^{[1]}_4$	$L^{[1]}_5$	$L^{[1]}_6$	$L^{[1]}_7$	$L^{[1]}_8$	$L^{[1]}_9$
$L^{[0]}_0$	0,4949	0,3751	0,3181	-0,2837	-0,4633	-0,3130	0,0462	-0,3694	-0,5308	0,2586
$L^{[0]}_1$	0,1949	-0,4851	0,8025	0,3736	-0,4757	0,2525	-0,2237	-0,1543	-0,1858	0,2754
$L^{[0]}_2$	-0,1240	-0,0694	0,0800	-0,2585	0,0249	0,2644	0,4629	-0,3231	-0,3739	0,8155
$L^{[0]}_3$	-0,4877	0,2059	-0,3034	-0,1130	0,3097	0,2768	0,4408	0,6138	-0,5277	-0,0089
Bias	0,0164	-0,0104	0,1322	0	0	-0,1037	0,0029	0	0	-0,1566

Tabla 3. Pesos de la capa oculta 1 para el problema del clasificador.

	$L^{[2]}_0$	$L^{[2]}_1$	$L^{[2]}_2$	$L^{[2]}_3$	$L^{[2]}_4$	$L^{[2]}_5$	$L^{[2]}_6$	$L^{[2]}_7$	$L^{[2]}_8$	$L^{[2]}_9$
$L^{[1]}_0$	-0,017	0,175	-0,084	-0,237	-0,436	-0,120	-0,450	-0,508	-0,271	-0,412
$L^{[1]}_1$	-0,296	-0,083	0,417	-0,109	0,653	0,398	0,405	-0,222	-0,658	0,213
$L^{[1]}_2$	0,028	0,303	-0,052	-0,382	-0,181	-0,281	0,082	0,276	0,539	0,035
$L^{[1]}_3$	0,313	0,076	0,392	0,515	-0,019	-0,131	0,298	-0,233	0,286	0,207
$L^{[1]}_4$	0,313	-0,157	0,420	-0,242	0,291	-0,480	0,191	-0,218	0,389	0,220
$L^{[1]}_5$	0,788	-0,410	0,249	-0,133	0,013	0,460	0,132	0,184	0,110	-0,032
$L^{[1]}_6$	-0,203	-0,001	-0,403	0,468	0,298	-0,157	-0,431	0,717	0,234	0,536
$L^{[1]}_7$	0,397	-0,232	-0,323	-0,541	0,308	0,190	-0,485	-0,170	-0,173	0,513
$L^{[1]}_8$	0,218	0,431	-0,305	0,478	0,017	0,199	0,149	-0,246	0,502	0,516
$L^{[1]}_9$	0,223	0,351	-0,318	0,718	0,607	0,166	-0,155	0,215	0,147	0,249
Bias	-0,079	0,093	0	-0,074	-0,050	-0,055	-0,134	0	0,136	-0,023

Tabla 4. Pesos de la capa oculta 2 para el problema del clasificador.

	$L^{[3]}_0$	$L^{[3]}_1$	$L^{[3]}_2$
$L^{[2]}_0$	-0,4558	-0,7504	0,3269
$L^{[2]}_1$	-0,1039	-0,1854	-0,6559
$L^{[2]}_2$	0,2857	0,1208	-0,5869
$L^{[2]}_3$	-0,6514	0,0361	0,1253
$L^{[2]}_4$	-0,8796	-0,3285	-0,3988
$L^{[2]}_5$	0,1496	-0,0413	-0,1389
$L^{[2]}_6$	-0,3261	0,1937	0,3480
$L^{[2]}_7$	-0,2504	0,5358	0,3766
$L^{[2]}_8$	0,6003	-0,6083	-0,4603
$L^{[2]}_9$	-0,0371	0,4301	0,8630
Bias	0,1504	0,0048	-0,0914

Tabla 5. Pesos de la capa salida para el problema del clasificador.

En este punto, también se observa que las reglas extraídas pueden expresarse como formulas bien formadas (fbf) de la lógica de primer orden (Pons et al., 2017), entonces si calculamos las tablas de verdad para cada una de las reglas obtendremos valores de verdad en cada neurona, respecto de los valores de entrada. Si se toma como ejemplo $L^{[2]}_0$ = *(-X1 v -X2),* su tabla de verdad será como se ilustra en la tabla de verdad 1. Por favor notar que en lugar de usar valores Verdadero o Falso para representar verdad o falsedad utilizamos 1 para identificar valor de verdad y 0 para identificar valor de falsedad. De este modo se siguen utilizando los mismos valores binarios con los que fue entrenada la red.

Aplicando el mismo procedimiento para el resto de las reglas, se obtienen las tablas de verdad que se muestran en la tabla de verdad 2.

Finalmente, al calcular las conjunciones entre todas las componentes del sistema, se obtienen tautologías a lo largo de todo el sistema de conjunciones, lo cual demuestra que las reglas son válidas y explican la función aprendida por la red neuronal. Las tablas de verdad del sistema final se pueden ver en la tabla de verdad 3.

Calculando la relación de implicancia entre las reglas formadas en cada capa, se observa que los resultados son tautologías también. A modo de ejemplo, si se toman las fbf entre $L^{[1]}_0$ y $L^{[2]}_0$ y se calcula $L^{[1]}_0$ -> $L^{[2]}_0$ se obtiene como resultado una tautología. Por último, si se calculan las implicancias entre los valores de verdad para las reglas obtenidas en cada capa, se obtiene que las siguientes fbf están implícitamente relacionadas:

Capa oculta:

$f^{[1]}_0 = (\neg X6 \wedge X1) \vee (\neg X6 \wedge \neg X3) \vee (X1 \wedge \neg X3)$
$f^{[1]}_1 = (\neg X0 \wedge X1) \vee (\neg X0 \wedge \neg X3) \vee (X1 \wedge \neg X3)$
$f^{[1]}_{out} = (X1 \wedge \neg X3) \vee ((\neg X6 \wedge X1) \vee (\neg X6 \wedge \neg X3) \wedge (\neg X0 \wedge X1) \vee (\neg X0 \wedge \neg X3))$

Capa de Salida:
$f^{[2]}_0 = (\neg X1 \vee \neg X2)\, f^{[2]}_3 = (\neg X2 \vee \neg X0 \vee \neg X1)$
$f^{[2]}_5 = (\neg X0 \vee \neg X1 \vee X2)\; f^{[2]}_6 = (\neg X2 \vee \neg X0 \vee X1)$

Al reducir el sistema, la fbf final quedaría del siguiente modo: Net = $f^{[1]}_{out}$-> ($f^{[2]}_0$ v $f^{[2]}_3$ v $f^{[2]}_5$ v $f^{[2]}_6$)

X1	X2	-X1	-X2	-X1 v -X2
1	1	0	0	0
1	1	0	0	0
1	0	0	1	1
1	0	0	1	1
0	1	1	0	1
0	1	1	0	1
0	0	1	1	1
0	0	1	1	1

Tabla de verdad 1. Valores de verdad Para $L^{[2]}_0$ = (-X1 ^ -X2) v (-X1 ^ X0) v (-X2 ^ X0)

A	*B*	*C*	*D*	*E*	*F*
$L^{[2]}_0$	$L^{[2]}_1$	$L^{[2]}_3$	$L^{[2]}_5$	$L^{[2]}_6$	$L^{[2]}_7$
0	1	0	1	1	0
0	1	1	0	0	1
1	0	1	1	1	0
1	0	1	1	1	1
1	0	1	1	1	1
1	0	1	1	1	1
1	0	1	1	1	0
1	0	1	1	1	1

Tabla de verdad 2. Valores de verdad para todas las reglas obtenidas en las neuronas en la capa de salida

V	*W*	*X*	*Y*	*Z*
$A \vee B$	$V \vee C$	$W \vee D$	$X \vee E$	$Y \vee F$
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Tabla de verdad 3. Conjunciones para las salidas de las reglas de cada neurona de la capa de salida.

A	*B*	*C*	*D*	*E*	*F*
$L^{[2]}_0$	$L^{[2]}_1$	$L^{[2]}_3$	$L^{[2]}_4$	$L^{[2]}_8$	$L^{[2]}_9$
1	0	0	1	1	1
1	0	0	1	0	1
1	0	0	1	1	0
1	0	0	1	0	1
1	0	0	1	0	1
1	0	0	1	0	1
1	1	0	1	0	0
1	1	0	1	0	0

Tabla de verdad 4. fbfs de la capa 2 para el problema del clasificador.

G	*H*	*U*	*J*	*K*
$A \vee B$	$G \vee C$	$H \vee D$	$I \vee E$	$J \vee F$
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Tabla de verdad 5. Conjunciones para la fbf de la capa 2 para el problema del clasificador.

Ejemplo 3: En este ejemplo, se extraen reglas usando AREBI para el problema de un clasificador para el conjunto de datos Iris. En este problema, se clasifica un conjunto de *features* de entrada a un conjunto variables categóricas en la salida. La topología de la red neuronal presenta cuatro neuronas en la entrada y 3 en la salida. Además, presenta dos capas ocultas con 10 neuronas en cada capa. En las capas ocultas se utilizó la función de activación *Relu* y para la capa de salida la función *Softmax*. La misión es hacer que la red neuronal aprenda a clasificar el tipo el tipo de flor adecuado. Los pesos de las capas oculta y de salida de la red entrenada se muestran en las Tablas 3, 4 y 5, respectivamente. La siguiente regla fue extraída para las neuronas de la capa de salida:

$L^{[2]}_1 = (-X0\text{-}\hat{}X8\hat{}X7) \vee (-X0\hat{}\text{-}X8\hat{}X9) \vee (-X0\hat{}\text{-}X8\text{-}\hat{}X4)$

Si bien las neuronas 0 y 2 de la capa de salida forman formulas bien formadas, estas no están implicadas por ninguna fbf en las capas ocultas, con lo cual se las excluyen del análisis. Luego de aplicar las transformaciones admitidas, los umbrales para cada neurona de las capas ocultas y de salida, quedan de la siguiente manera:

$TL^{[1]}_0$ =0,6508; $TL^{[1]}_1$= 0,5678; $TL^{[1]}_2$= 0,7521 $TL^{[1]}_3$=0,5144; $TL^{[1]}_4$= 0,6369;
$TL^{[1]}_5$= 0,5534 $TL^{[1]}_6$=0,5869; $TL^{[1]}_7$= 0,7304; $TL^{[1]}_8$= 0,8092 $TL^{[1]}_{9=}$0,6793

$TL^{[2]}_0$ =1,4002; $TL^{[2]}_1$= 1,1118; $TL^{[2]}_2$= 1,4838 $TL^{[2]}_3$= 1,9146; $TL^{[2]}_4$= 1,4147;
$TL^{[2]}_5$= 1,2928
$TL^{[2]}_6$= 1,3904; $TL^{[2]}_7$= 1,4964; $TL^{[2]}_8$= 1,6582 $TL^{[2]}_{9=}$ 1,4692
$TL^{[3]}_{0=}$ 1,8701; $TL^{[3]}_{1=}$ 1,6154; $TL^{[3]}_{2=}$ 2,1405

Las neuronas 5, 6 y 7 de la capa oculta 2 forman fbf que son demasiado específicas y violan el principio de máxima generalidad, con lo cual se las omitió del análisis. Las tablas de verdad para esta capa respecto de las fbfs junto con sus conjunciones, se muestra en las tablas 4, y 5.
Para esta DNN, luego de construir las tablas de verdad, se verifican las tautologías entre las reglas extraídas para cada capa. Además, calculando la relación de implicancia entre las reglas formadas en cada capa, se observa que los resultados son tautologías también. En tal sentido, las reglas extraídas para cada capa junto con las relaciones de implicancia entre capas devuelven la siguiente regla:

$\mathbf{R_1}$ = ((X2^X1) ᵛ (X2^X0)) -> (((X5^X7) ^ (X3 ᵛ X4 ᵛ -X1)) ᵛ ((X9^X3) ^ (-X7 ᵛ -X4 ᵛ X0)) ᵛ (X3^-X7^-X4)) -> ((-X0^-X8) ^ (X7 ᵛ X9 ᵛ -X4))

El patrón para cada regla sigue el siguiente formato:

R_x= *(regla capa 1) -> (regla capa 2) -> (regla capa de salida)*.

	X_0	X_1	X_2	X_3
	sepal length	sepal width	petal length	petal width
Clase 0				
Media	5.0	3.4	1.5	0.2
Max	5.8	4.4	1.9	0.6

Cotas	[5..5,8]	[3,4..4,4]	[1,5..1,9]	[0,2..0,6]
Clase 1				
Min	4,9	2	3	1
Media	5,9	2,8	4,2	1,3
Cotas	[4,9..5,9]	[2..2,8]	[3..4,2]	[1..1,3]
Clase 2				
Perc 50	6,5	3	5,6	2
Max	7,9	3,8	6,9	2,5
Cotas	[6,5..7,9]	[3..3,8]	[5,6..6,9]	[2..2,5]

Tabla 6. Estadísticas por clase y *feature* para el problema del clasificador.

En la tabla 6, se presentan los valores de las estadísticas calculadas sobre todo el conjunto de datos, para las cuales la precisión predictiva de la red se maximiza. Dado que AREBI busca aquellos valores que maximicen la función Act(Zi) ≈ 1, se probará la regla extraída con dichos valores estadísticos para verificar que estas expliquen a los datos.

Para validar la regla se partirá de la premisa de que los valores de verdad para cada variable de entrada y para cada clase, estará en relación con si la variable se encuentra dentro del rango de valores expresado en la tabla 6, para cada clase. Entonces, por ejemplo, para X_2 la variable será verdadera si toma valores entre 1,5 y 1,9 para determinar la clase 0, de lo contrario será falsa.

Si se toma el primer término de la regla ((X2^X1) ^{v} (X2^X0)) y se le asignan a cada variable de entrada los rangos de la tabla 6, se observa que para la clase 0 la regla quedaría: ([1,5..1,9] ^[3,4..4,4]) ^{v} ([1,5..1,9] ^[5..5,8]). Aquí se observa que alcanza con evaluar la primera expresión del término dado que resuelve completamente el solapamiento que se evidencia en X_1 para las clases 0 y 3. El segundo termino determina la fidelidad de la expresión. Entonces la regla satisface la clasificación para la clase 1 cuando ([1,5..1,9] ^[3,4..4,4]) ^{v} ([1,5..1,9] ^[5..5,8]), y para la clase 2 cuando ([6,5..7,9]^[3..3,8]) ^{v} ([5,6..6,9] ^[2..2,5]). Desde el punto de vista de la regla, cualquier combinación de valores en la entrada que no siga esta distribución, hará que las conjunciones y disyunciones en la entrada no formen tautologías, y este patrón fluirá hacia el resto de la regla, haciendo que las relaciones de implicancia no se cumplan y en consecuencia que los datos no expliquen la clasificación a la salida.

Desde el punto de vista de la red, las distribuciones de datos en la entrada por fuera de estos rangos de valores podrían arriban a clasificaciones erróneas, con una baja probabilidad o con probabilidades solapadas entre clases.

5 Análisis del algoritmo AREBI

Las reglas generadas por AREBI son válidas, genéricas y completas.

Proposición 1. Las reglas generadas por AREBI son válidas.

Prueba. Una regla es válida si se cumple independientemente de las activaciones de las neuronas no especificadas en la regla. Supongamos que WC son los pesos de la combinación. Una regla se forma solo si se cumple la siguiente condición:

$$\sum_{W_C \in W} W_C \; + \; Bias > T_{new}$$

Aquí T_{new} es el valor recalculado del umbral luego de aplicar las transformaciones admitidas. Es fácil ver que en presencia de los pesos que no están en la combinación anterior, la relación aún se mantendrá. Esto se debe a que todos los pesos que no están en la combinación también son positivos. La presencia de tales pesos solo aumentará la suma ponderada. Por lo tanto, todas las reglas generadas por AREBI son válidas.

Proposición 2. Las reglas generadas por AREBI son genéricas.

Prueba. En la búsqueda de reglas de confirmación en el árbol de combinación, cuando una combinación tiene éxito, todas las combinaciones del subárbol del que es raíz se eliminan. Por lo tanto, todas las combinaciones que podrían formar reglas subsumidas se eliminan, por lo tanto, las reglas generadas cumplen con la máxima generalidad.

Proposición 3. El conjunto de reglas generado por AREBI es completo.

Prueba. El algoritmo elimina las combinaciones de dos maneras: a lo ancho y en profundidad a lo largo del árbol. En el primer caso, las combinaciones podadas no pueden formar una regla. Para el último caso, las combinaciones podadas pueden formar reglas, pero estas están subsumidas por otras reglas más generales. Dado que se generarán todas las demás reglas posibles, las reglas extraídas forman un conjunto completo.

5.1 Complejidad del modelo

Cuando se verifica la formación de reglas, es decir, existen algunas reglas para la neurona, el éxito podría ocurrir en cualquier nivel del árbol de combinaciones. Supongamos que hay N pesos para una neurona dada, entonces el número máximo de combinaciones que podrían tener éxito en el árbol de combinación para los N pesos es ${}^{N}C_{N/2}$. Para llegar al nivel del árbol donde se intentan combinaciones de tamaño N/2, una combinación en cada uno de los niveles anteriores debería haber fallado. Por lo tanto, el número máximo de combinaciones que deben intentarse es: ${}^{N}C_{N/2} + \frac{1}{2}(N-1)$. Para valores grandes de N el termino $\frac{1}{2}(N-1)$ puede ser desestimado, y en consecuencia el número de combinaciones máximo que podrían formar una regla se resume a ${}^{N}C_{N/2}$. Por lo tanto ${}^{N}C_{N/2}$ será el número máximo de reglas que pueden ser extraídas dado el vector de pesos de una neurona dada. Utilizando la aproximación de Stirling $m! = \sqrt{2\pi m}(\frac{m}{e})^m$ obtenemos ${}^{N}C_{N/2} \approx 2^n \sqrt{\frac{2}{\pi N}}$

Por lo tanto, la complejidad del método en el peor de los casos es $O(2^N)$.

5.2 Límites en el número mínimo de podas

Como se señaló anteriormente, las podas pueden ocurrir de dos maneras, a lo ancho y en profundidad. Si en una neurona dada una combinación logra formar una regla, se realiza una poda en profundidad, de lo contrario, se realiza una poda en anchura. La poda se realizará en todos los nodos internos del árbol. En los nodos de las hojas del árbol, la poda puede o no realizarse. En general, todas aquellas combinaciones que contengan el dígito 'N' (donde N es el número total de pesos) podrían resultar en ninguna poda. En los casos en que N es grande y el valor de Bias ≈ Umbral, las reglas tienden a formarse en el primer nivel del árbol, con lo cual la poda será al menos N/2, y obtendremos reglas formadas por disyunciones. Para los casos donde la distancia entre Umbral y Bias es grande, la poda será al menos N - log2(N). Aquí las reglas se darán en niveles medios del árbol y estas serán una combinación entre conjunciones y disyunciones. Para los nodos donde una regla se forma con éxito se observa que al menos una combinación fallará cuando *Bias* < *Umbral/2*, con lo cual la poda será como mínimo N-1. Como se señaló anteriormente, cuando el valor de *Bias* ≈ *Umbral*, las reglas tienden a formarse en el primer nivel del árbol, obtendremos disyunciones y la probabilidad de poda tendera a 0.

6 Trabajo relacionado y conclusiones

Varios investigadores han propuesto algoritmos para extraer reglas de una red neuronal alimentada hacia adelante entrenada. Para ello se aborda la explicabilidad desde diferentes enfoques. Para el caso los métodos donde se explica la *estructura de decisión global* (Krishnan et al., 1999b) extraen arboles de decisión desde las redes neuronales entrenadas y utilizan algoritmos genéticos para consultar la red y selección de prototipos. (Markowska-Kaczmar & Wnuk-Lipiński, 2004) presentan un método de extracción de reglas a partir de redes neuronales basado en algoritmos genéticos con optimización de Pareto.

Para el caso donde se analiza el *ruido y dimensionalidad relevante* (Montavon et al., 2011) Analizan la evolución por capas de la representación en una red profunda mediante la construcción de una secuencia de núcleos. (Braun & Buhmann, 2008) Explican como la información relevante de un problema de aprendizaje supervisado está contenida hasta un error insignificante en un número finito de componentes PCA principales del kernel. (Özbakir et al., 2009) proponen un método que utiliza el algoritmo Touring Ant Colony Optimization (TACO) para extraer reglas precisas y comprensibles de bases de datos a través de redes neuronales artificiales entrenadas.

Para el caso donde la explicabilidad se aborda dese el análisis del *papel que juegan neuronas particulares* (Erhan et al., 2010), buscan mejorar las herramientas para encontrar buenas interpretaciones cualitativas de características de alto nivel aprendidas por los modelos. También se busca comprender mejor las invariancias aprendidas por redes neuronales profundas. (Augasta & Kathirvalavakumar, 2012) proponen un algoritmo RxREN que extrae reglas de redes neuronales entrenadas para conjuntos de datos

con atributos de modo mixto. El algoritmo se basa en la técnica de ingeniería inversa para podar las neuronas de entrada insignificantes y descubrir los principios tecnológicos de cada neurona de entrada significativa de la red neuronal en la clasificación.
Desde la óptica de la *invariancia a ciertas transformaciones de los datos* (Goodfellow et al., 2009) proponen una serie de pruebas empíricas que miden directamente el grado en que las características aprendidas son invariantes a diferentes transformaciones de entrada. (Zarlenga et al., 2021) presentan ECLAIRE, un novedoso algoritmo de extracción de reglas de tiempo polinomial capaz de escalar tanto a grandes arquitecturas de redes neuronales, como a grandes conjuntos de datos de entrenamiento.

Las principales contribuciones hechos en este trabajo son:

- Proponer un nuevo algoritmo para extraer el patrón de reglas aprendido de una red neuronal *feedforward* entrenada y analizar sus propiedades.
- Uso de lógica de primer orden (FOL) para explicar los patrones aprendidos por la red neuronal.
- Presentar un método de caja blanca que analiza las matrices de pesos de las DNN para extraer reglas,
- El método no requiere ninguna regla de aprendizaje especial durante la extracción de reglas, lo que facilita su uso.
- El método funciona de manera eficiente tanto para atributos continuos como enumerados.
- El algoritmo es muy eficiente en la poda de neuronas no relevantes de la DNN y no requiere reentrenamiento después de la poda.
- Las reglas extraídas se pueden usar para explicar el proceso de toma de decisiones de la red neuronal, a través de fórmulas de la lógica de primer orden, lo que puede ser útil en diversas aplicaciones, como el diagnóstico médico, la detección de fraudes y la calificación crediticia.

7 Conclusiones

En este artículo, se ha propuesto un nuevo algoritmo de búsqueda para la extracción de reglas que se basa en ordenar los pesos y considerar combinaciones de pesos en el orden de su suma ponderada. Este algoritmo genera reglas válidas, máximamente generales y completas. Aunque el algoritmo tiene una complejidad exponencial, todavía es viable para su uso en redes neuronales de tamaño pequeño o mediano. Se investigaron las propiedades de poda del algoritmo y se calculó un límite inferior para el número de posibles podas.
Aunque posiblemente se podría obtener un límite más estricto, se conjetura que este método da como resultado el número máximo de podas. El trabajo futuro implicará analizar las reglas extraídas para verificar que el algoritmo no solo sirve para explicar la función aprendida por la red, sino también para regularización.

Referencias

1. AmirHosseini, B., & Hosseini, R. (2019). An improved fuzzy-differential evolution approach applied to classification of tumors in liver CT scan images. *Medical & Biological Engineering & Computing*, *57*(10), Article 10. https://doi.org/10.1007/s11517-019-02009-7
2. Augasta, M. G., & Kathirvalavakumar, T. (2012). Reverse Engineering the Neural Networks for Rule Extraction in Classification Problems. *Neural Processing Letters*, *35*(2), 131-150. https://doi.org/10.1007/s11063-011-9207-8
3. Braun, M. L., & Buhmann, J. M. (2008). *On Relevant Dimensions in Kernel Feature Spaces*.
4. Britos, P. V. (2005). *Minería de datos basada en sistemas inteligentes* (1a ed.). Nueva Libreria.
5. Csiszár, O., Csiszár, G., & Dombi, J. (2020). How to implement MCDM tools and continuous logic into neural computation?: Towards better interpretability of neural networks. *Knowledge-Based Systems*, *210*, 106530. https://doi.org/10.1016/j.knosys.2020.106530
6. Domingos, P. (2018). How the Quest for the Ultimate Learning Machine will remake our World. En *The Master Algorithm How*.
7. Erhan, D., Courville, A., Bengio, Y., & Box, P. O. (2010). *Understanding Representations Learned in Deep Architectures*.
8. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.
9. Goodfellow, I., Lee, H., Le, Q. V., Saxe, A., & Ng, A. Y. (2009). *Measuring Invariances in Deep Networks*.
10. Krishnan, R., Sivakumar, G., & Bhattacharya, P. (1999a). A search technique for rule extraction from trained neural networks. *Pattern Recognition Letters*, *20*(3), 273-280. https://doi.org/10.1016/S0167-8655(98)00145-7
11. Krishnan, R., Sivakumar, G., & Bhattacharya, P. (1999b). *Extracting decision trees from trained neural networks*. 12.
12. Mahdavifar, S., & Ghorbani, A. A. (2020). DeNNeS: Deep embedded neural network expert system for detecting cyber attacks. *Neural Computing and Applications*, *32*(18), Article 18. https://doi.org/10.1007/s00521-020-04830-w
13. Markowska-Kaczmar, U., & Wnuk-Lipiński, P. (2004). Rule Extraction from Neural Network by Genetic Algorithm with Pareto Optimization. En L. Rutkowski, J. H. Siekmann, R. Tadeusiewicz, & L. A. Zadeh (Eds.), *Artificial Intelligence and Soft Computing—ICAISC 2004* (pp. 450-455). Springer. https://doi.org/10.1007/978-3-540-24844-6_66
14. Montavon, G., Braun, M. L., & Mueller, K.-R. (2011). *Kernel Analysis of Deep Networks*.
15. Montavon, G., Lapuschkin, S., Binder, A., Samek, W., & Müller, K.-R. (2017). Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, *65*, 211-222. https://doi.org/10.1016/j.patcog.2016.11.008
16. Nielsen, I. E., Dera, D., Rasool, G., Bouaynaya, N., & Ramachandran, R. P. (2022). Robust Explainability: A Tutorial on Gradient-Based Attribution Methods for Deep Neural Networks. *IEEE Signal Processing Magazine*, *39*(4), 73-84. https://doi.org/10.1109/MSP.2022.3142719
17. Özbakir, L., Baykasoğlu, A., Kulluk, S., & Yapıcı, H. (2009). TACO-miner: An ant colony based algorithm for rule extraction from trained neural networks. *Expert Systems with Applications*, *36*(10), 12295-12305. https://doi.org/10.1016/j.eswa.2009.04.058
18. Pons, C., Rosenfeld, R., & Smith, C. P. (2017). *Lógica para Informática*. Editorial de la Universidad Nacional de La Plata (EDULP). https://doi.org/10.35537/10915/61426
19. Russell, S., & Norvig, P. (2010). Artificial Intelligence A Modern Approach Third Edition. En *Pearson*. https://doi.org/10.1017/S0269888900007724

20. Santos, R. T., Nievola, J. C., & Freitas, A. A. (2000). Extracting comprehensible rules from neural networks via genetic algorithms. *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (Cat. No.00EX448)*, 130-139. https://doi.org/10.1109/ECNN.2000.886228
21. Sethi, I. K., & Yoo, J. H. (1996). Symbolic mapping of neurons in feedforward networks. *Pattern Recognition Letters*, *17*(10), 1035-1046.
22. Zarlenga, M. E., Shams, Z., & Jamnik, M. (2021). *Efficient Decompositional Rule Extraction for Deep Neural Networks* (arXiv:2111.12628). arXiv. https://doi.org/10.48550/arXiv.2111.12628