Solución de Sudokus mediante el uso de Redes Neuronales Artificiales

Luis Barón, Sergio Rojas, Robert Urbina
Pontificia Universidad Javeriana
Departamento de ingeniería electrónica
Inteligencia Artificial
Bogotá, Colombia
baron_luis, s_rojas, r_urbina @javeriana.edu.co

***Resumen*—La solución tradicional de Sudokus, se realiza basada en algoritmos lógicos, probabilísticos y algunos casos a fuerza bruta. En este documento se busca explorar la solución de Sudokus por medio de redes neuronales artificiales, para lo cual se plantean dos modelos de perceptrón multicapa que reciben el planteamiento de un Sudoku tradicional y devuelven su solución. Para cada modelo se muestra su proceso de desarrollo, construcción y los resultados en el rendimiento de los dos modelos planteados.**

***Index Terms*—Redes neuronales Artificiales,Inteligencia artificial, Perceptrón, Sudokus,**

I. Introducción

I-A. Objetivo general

- A partir de un dataset de un millón de datos de partidas de sudokus, con y sin solución, construir una red neuronal que solucione sudokus.

I-B. Objetivos específicos

- Identificar el número de capas, número de neuronas y las funciones de activación que debe tener la red neuronal a entrenarse.
- Entrenar la red neuronal que resuelva sudokus.
- Evaluación del rendimiento y exactitud de las predicciones realizadas por la red.

I-C. Marco teórico

Los Sudokus son un juego matemático de origen Japones, inventado a finales de la década de 1970 y popularizado internacionalmente alrededor del 2005 [1]. El Sudoku está conformado por una cuadricula de 9x9 celdas, lo que da un total de 81 casillas. El objetivo del juego es completar las 81 casillas con los números naturales (1 al 9), de modo que no se repitan los números por cada fila, por cada columna y por cada subcuadro de 3x3. Un juego de Sudoku tradicional inicia con alguna cantidad números ya ubicados en posiciones especificas y el jugador debe completar las casillas de modo que se cumplan las reglas de juego.

La solución tradicional de un Sudoku, se realiza por medio análisis de sucesiva eliminación, descartando los posibles números candidatos a cada celda hasta dejar solo una elección que permite asignar ese numero a la respectiva casilla. Este método puede no ser suficiente para solucionar todas las posibles combinaciones de Sudokus que existen, por lo cual para algunos casos de juego se hace inevitable encontrar una solución probando todas las opciones posibles hasta que converja la solución correcta.

Al completar la solución de un Sudoku se obtiene un cuadro latino [2], que consiste en una matriz de nxn donde no se repiten los números por cada una de sus filas ni columnas, pero no todos los cuadros latinos de 9x9 corresponden a un Sudoku, debido a la restricción de no repetir los números en los subcuadros de 3x3. Usando este análisis y otros cálculos de probabilidad en [3], afirman que existen 6670903752021072936960 $\approx 6.671\times10^{21}$ Sudokus distintos.

De acuerdo con la cantidad de números ubicados al inicio de un juego y la posición de dichos números iniciales, la dificultad del Sudoku puede ser mayor o menor. Además, se dice que un Sudoku está bien planteado si tiene solución única, en [4] después de un análisis computacional por búsqueda de simetría y fuerza bruta, determinaron que un Sudoku debe tener mínimo 17 casillas iniciales para que su solución sea única.

En este documento no se busca solucionar un Sudoku por un modelo determinista, sino mediante el uso de perceptrones multicapa de redes neuronales artificiales (ANN por sus siglas en ingles). Las ANN se codifican en Python, usando la librería de tensorflow Keras, que es una biblioteca de redes neuronales de código abierto, con la cual se construyen dos modelos.

Para el primer modelo se usa como función de perdida *Binary srossentropy* y para el segundo modelo se usa como función de perdidas *Sparse categorical crossentropy*. En la sección de desarrollo se explica a detalle la composición de cada modelo. Los modelos son entrenados solo con muestras de hasta 1 Millón de datos, debido al alto costo computacional que se requiere para el entrenamiento de las redes. Los Sudokus usados en el entrenamiento tienen entre 30 y 37 posiciones

iniciales, que se pueden asociar a un juego tradicional de un periódico o revista. En la sección de resultados se muestra la exactitud y rendimiento de cada modelo, se analiza la viabilidad de aplicar ANN para solucionar Sudokus, se dan conclusiones e ideas para mejorar en trabajos futuros.

II. Desarrollo

Teniendo en cuenta que las redes neuronales suelen tener diferentes comportamientos a partir del número de entradas, número de capas ocultas y cantidad de neuronas, se generan dos propuestas distintas con el objetivo de evaluar no solo la posible solución de sudokus, sino también cómo es la respuesta de la red ante la variación de estas variables. A continuación presentamos dos propuestas distintas para la evaluación de la red.

II-A. Primera propuesta para la red neuronal de perceptrón multicapa

Para comenzar, la primera propuesta de solución para la red neuronal se basa en hacer una normalización personalizada de los datos. Posteriormente, se define el número de capas, número de neuronas, funciones de activación y el número de posiciones para la salida esperada.

Entrando en detalle, la consideración para el tratamiento de los datos tanto de características como de solución esperada, se hace para que el dato de cada casilla (de las 81) se divida en arreglos de 9 posiciones con valores de 0 o 1 dependiendo de número de entrada. Para mayor especificidad, el 1 se colocará en la posición del valor correspondiente al número de entrada para el arreglo de 9 posiciones y a su vez, cada arreglo conformará al vector de entrada que tendrá un tamaño de $9 * 81 = 729$ posiciones.

Dadas las entradas, para las salidas también se consideró tener un arreglo de 729 posiciones, en las cuales se espera que cada salida indique la probabilidad de salida para cada índice. Lo anterior permitiría seleccionar entre grupos de 9 posiciones el índice de mayor argumento como solución del sudoku para cada casilla.

Respecto al número de capas ocultas, se consideró la realización de 2 a 3 capas ocultas con 2000 neuronas cada una sin contar las 729 salidas. Asimismo, para dar una respuesta rápida y no lineal a la red se propone utilizar la función de activación *relu*[5] en las capas ocultas, y para la salida utilizar la función de activación *softmax*[6] con el objetivo de normalizar las propiedades de salida y poder elegir correctamente entre las distintas clases.

Teniendo en cuenta lo anterior, para la elección del número de capas ocultas se ejecutaron entrenamientos con 2 y 3 capas ocultas con midiendo el rendimiento de la red para la predicción de sudokus y el *loss* midiendo la entropía cruzada binaria ($binary_crossentropy$ [7]), en la cual se calcula la pérdida por probabilidades, tal como se muestra a continuación:

>>> y_{true} = [[0., 1.], [0., 0.]]
>>> y_{pred} = [[0.6, 0.4], [0.4, 0.6]]
>>> bce = tf.keras.losses.BinaryCrossentropy()
>>> $bce(y_{true}, y_{pred})$.numpy()
0.815

Considerando la Fig. 1, en la que se muestra un sudoku sin solucionar y otro solucionado, en la Fig. 2 se ilustra cómo se dividen las entradas y es la estructura de la red.

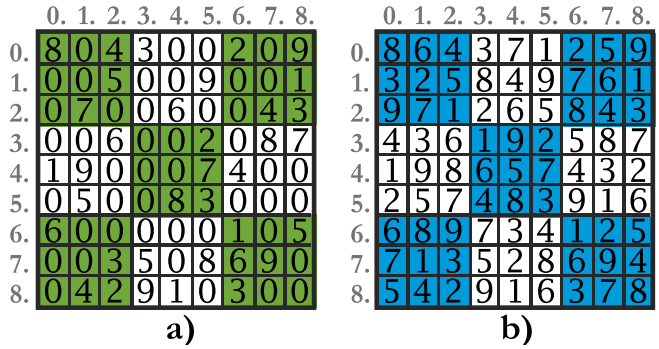

Figura 1. Sudoku: a)Juego inicial b)Juego solucionado

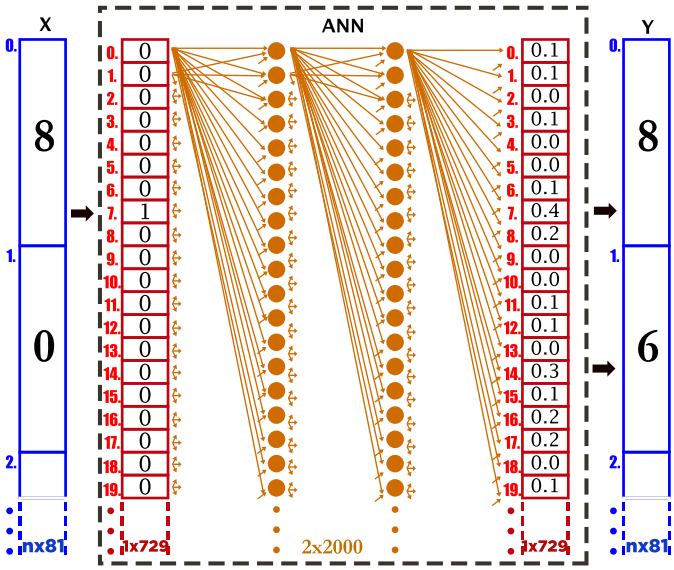

Figura 2. Modelo ANN 1

II-B. Segunda propuesta para la red neuronal de perceptrón multicapa

En esta sección se presenta una solución que inicia con un pre-procesamiento de datos un poco particular. Primero, se toman los valores que se encuentran en el dataset y se reforman en la forma matricial que suele

verse convencionalmente en los sudokus (equivalente a reshape((9,9,1))) sin embargo, en adición a este cambio, se realiza una normalización en un rango de entre -0.5 a 0.5, y esto con el fin de obtener un mejor comportamiento al ejecutarse la función de entrenamiento.

Esta propuesta surge tras analizar la solución de sudokus que se realiza en [8], donde se implementa una red neuronal de tipo convolucional. Dicho tratamiento de los datos mencionado nos hace cambiar la idea inicial, donde reducimos drásticamente el numero de capas y neuronas dentro de estas de forma considerable, yendonos al otro extremo, donde en ver de tener un posible caso de *overfitting* debido al gran numero de capas y neuronas, podríamos obtener un caso de *underfitting*, lo cual también podría ser un inconveniente, sin embargo, no fue exactamente así.

Ahora, con un nuevo formato en los datos, se propone un modelo de red neuronal como el que se observa en la Fig. 3, donde contamos con 3 capas ocultas, siendo las dos primeras de 64 neuronas y la tercera del doble, es decir, 128, para finalmente, tener a la salida de la red de nuevo un arreglo de 729 posiciones, como se mostró en la primera red neuronal propuesta. A esto, se agrega el detalle de haberse usado una normalización por lotes con la función *BatchNormalization()* para las dos primeras capas ocultas del modelo, y debido a que las redes neuronales se entrenan rápidamente si la distribución de los datos de entrada permanece similar a lo largo del tiempo. La normalización por lotes le ayuda a hacer esto al hacer dos cosas: normalizar el valor de entrada, escalarlo y cambiarlo.

Entrando en detalle a la parte de las funciones de activación, se implementaron las funciones de *relu* en las tres capas ocultas, y de igual manera que en el primer modelo propuesto, se uso la función *softmax* para la capa de salida, esto por motivo de la necesidad de normalizar las propiedades de salida y poder elegir correctamente los valores. En temas de rendimiento, se utiliza la función de *sparse categorical crossentropy* en el *loss* de la compilación, la cual, según su documentación, calcula la pérdida de entropía cruzada categórica para mantener enteros como etiquetas de clasificación y alimentar los objetivos de entrenamiento como secuencias de números enteros por lo tanto, la salida del modelo estará en forma de *softmax* mientras que las etiquetas son números enteros.

Finalmente, para la etapa de entrenamiento, se hicieron otros cambios con respecto al primero modelo, implementando un tamaño de lote mucho mas pequeño, de tamaño 32 y solo dos épocas,y para el dataset utilizado de un millón de datos, se hacen 25000 iteraciones por cada época.

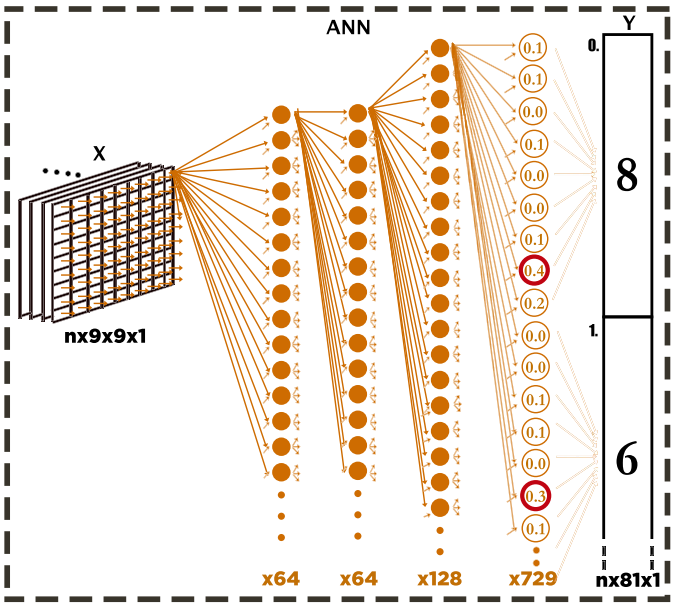

Figura 3. Modelo ANN 2

II-C. Evaluación de los modelos

La evaluación de los modelos se realiza a partir de de dos métricas distintas denominadas *Exactitud* y *Rendimiento*.

II-C1. Exactitud: Se calcula a partir de los valores de Y reales y los valores de Y_p predichos, se comparan los dos grupos de datos y se cuenta la cantidad de números que se encuentran en la misma posición y son iguales entre ambos grupos de datos. El valor total resultante se divide en el numero total de datos comparados, en este caso 81, que corresponde al numero total de casillas que tiene un Sudoku. La expresión que representa la métrica de exactitud se muestra en (1)

$$E = \frac{\sum_{k=1}^{81} y_k == y_{kp}}{81} \tag{1}$$

El objetivo ideal de esta métrica es lograr una resultado de 1 que indicaría la solución completamente correcta del Sudoku. El principal problema de esta métrica consiste que al no tener en cuenta las características iniciales de X, se desprecia la dificultad inicial del Sudoku. Debido a que no es correcto comparar la exactitud de una muestra cuyo Sudoku tiene 30 casillas que faltan a un Sudoku que tiene 50 casillas que faltan, puede que la mejor exactitud del primero se deba a que ya tiene 51 casillas correctas en comparación con el segundo que solo tiene 31.

II-C2. Rendimiento: Es una métrica que se basa en la exactitud, pero se tiene en cuenta las condiciones iniciales del Sudoku, para así obtener una figura de mérito un poco más acertada. Para calcular el rendimiento se usan las condiciones

iniciales de X, las etiquetas reales Y y las etiquetas predichas de Y_p, la expresión que representa la metrica de exactitud se muestra en (2)

$$\lambda = \left(\sum_{k=1}^{81} x_k \; ! = 0 \right)$$
$$\alpha = \left(\sum_{k=1}^{81} y_k == y_{kp} \right) - \lambda$$
$$R = \begin{cases} \frac{\alpha}{81-\lambda} & \text{si } \alpha > 0 \\ \frac{\alpha}{\lambda} & \text{si } \alpha < 0 \end{cases} \quad (2)$$

Con esta métrica de evaluación entre más cercano a 1, significa que ha resuelto el Sudoku correctamente en su totalidad y entre más cercano a -1 significa que el Sudoku está complemente mal. Si arroja un numero mayor a 0, significa que aumento el numero de posiciones correctas en comparación con el numero de casillas correctas iniciales que traía el Sudoku antes de entrar a la ANN, si por el contrario da un numero inferior a 0, significa que tiene una menor cantidad de soluciones correctas a las que ya tenia antes de entrar a la red. En ese orden de ideas 0 indica que el Sudoku tiene la misma cantidad de posiciones correctas a la entrada que a la salida, además el objetivo ideal es llegar a tener un rendimiento de 1.

Supongamos que se quiere calcular el rendimiento de un arreglo de 10 posiciones, para los cuales se tiene las características X, los valores Y_p predichos por el modelo ANN y los valores reales Y como se muestra a continuación.

Etiqueta	Valor
X	[1,2,3,4,5,0,0,0,0,0]
Y	[1,2,3,4,5,6,7,8,9,10]
Y_p	[1,2,3,4,5,6,7,8,9,0]

Nótese que la cantidad de posiciones correctas iniciales en X es 5 y al mirar los valores predichos en Y_p solo uno es incorrecto respecto al Y real, por lo tanto el rendimiento es de 0.8. Ahora se tiene el siguiente ejemplo

Etiqueta	Valor
X	[1,2,3,4,0,0,0,0,0,0]
Y	[1,2,3,4,5,6,7,8,9,10]
Y_p	[1,2,3,4,5,6,7,8,9,0]

En este caso la cantidad de posiciones correctas iniciales en X es 4 y al mirar los valores predichos en Y_p solo uno es incorrecto, por lo tanto el rendimiento es de 0.833. Es decir que aunque en cambos casos el numero de valores incorrectos en la predicción fue el mismo al partir de distintas condiciones iniciales arroja un mejor rendimiento al segundo caso cuando solo se tenia 4 valores correctos inicialmente, puesto que a pesar que se cuenta con menos información se logran los mismo resultados. De forma inversa sucede lo opuesto cuando la cantidad de posiciones correctas a la salida es menor a la cantidad de posiciones correctas a la entrada, para un mismo caso de posiciones incorrectas a la salida se penaliza más si el numero de posiciones correctas a la entrada es mayor en un caso u otro.
Con la métrica de rendimiento ya no es tan notorio el problema del numero variado de casillas correctas iniciales. Lo que permite compara de una forma más clara el rendimiento de la red para Sudokus cuyo numero de posiciones correctas iniciales cambia.

III. RESULTADOS

En esta sección se presentan los resultados para las propuestas vistas anteriormente. Para el resultado de la predicción del juego de sudoku se tomó el argumento mayor por cada 9 datos de salida de la red neuronal, lo que indica a qué clase pertenece un número dado el entorno de predicción para el sudoku.

Respecto a la primera propuesta de solución, se ejecutaron distintos entrenamientos con el objetivo de identificar el comportamiento de la red. El protocolo que se siguió fue establecer la mayor cantidad de neuronas acordes al tamaño de la entrada de los datos hacia la red y el mayor número de capas propuesto con el objetivo de evaluar empíricamente el comportamiento del loss, accuracy, el rendimiento y algunas predicciones a sudokus de distinta dificultad entrenando con 100 mil datos de sudokus con y sin solución.

Esta primera aproximación permitió visualizar sobreentrenamiento, dejando claro que a mayor número de capas la red tiende a empeorar el cálculo de los pesos arrojando resultados sin sentido. Asimismo, a mayor número de capas la propagación hacia atrás para el cálculo del loss volvió más demorado el cálculo de los pesos correctos de la red. A partir de lo anterior se decidió entrenar dos redes distintas variando los datos de entrenamiento y dejando fijo el número de capas en dos.

Los casos preparados para la primera propuesta consisten en la evaluación de la red con 500 epochs, batch size de 10000 datos y cantidad de datos para entrenamiento y validación variable. Respecto al segundo caso de propuesto se entrenó la red con 2 epochs, batch size de 32 y división entre conjunto de entrenamiento y de validación en proporción de 80 %-20 % para una totalidad de un millón de datos. La tabla I muestra los resultados del cálculo realizado internamente en la red para el loss encontrado en cada entrenamiento realizado. Asimismo, la tabla II muestra los resultados para la exactitud y rendimiento de las redes para las predicciones hechas.

Cuadro I
VALOR FINAL DE LA FUNCIÓN DE PERDIDA

Modelo	N° Datos entrenamiento	loss
1	10k	0.62
1	100k	1.53
2	800k	0.39

Cuadro II
MÉTRICAS DE EVALUACIÓN DE LOS MODELOS ANN

Modelo	N° Datos		Promedio	Máximo	Mínimo
1	2.5k	Exactitud	0.65	0.81	0.48
		Rendimiento	0.41	0.68	0.14
1	25k	Exactitud	0.17	0.32	0.04
		Rendimiento	-0.6	-0.19	-0.91
2	100	Exactitud	0.99	1	0.75
		Rendimiento	0.98	1	0.59

IV. ANÁLISIS Y CONCLUSIONES

A partir de las propuestas y resultados vistos anteriormente se presentan las siguientes ideas que se consideran importantes al momento de construir una red neuronal con el modelo de perceptrón.

1. La selección del numero de capas y la cantidad de neuronas es dependiente de la aplicación en la cual se esté trabajando. Como se experimentó a lo largo del diseño del perceptrón multicapa, si se aumentaba el numero de capas llegaba un punto donde se sobre entrenaba la red y siempre arrojaba el mismo numero como valor de predicción sin importar cual fuera el cambio a la salida. Inevitablemente la selección del numero de capas y neuronas es un proceso de ensayo y error, donde siempre se debe recordar sobre entrenar la red puede ser tan contraproducente como infra entrenar la red.
2. La selección de la función de activación y la función de perdidas son esenciales para un adecuado rendimiento. Para la selección de la función de activación se usó *relu* basados en la experiencia previa y para la selección de la función de perdidas de leyó la documentación y base a ellas se plantearon las redes, escogiendo la *Binary crossentropy* para el primer modelo y para el segundo modelo *Sparse categorical crossentropy*. De acuerdo con los resultados se puede evidenciar la notable ventaja de un modelo ANN con la segunda función de perdida, demostrando la importancia de realizar la selección de una función de perdidas adecuada e implementarla de forma correcta.
3. La selección de los datos para entrenar la red es importante, aunque en nuestra aplicación el numero de muestras de Sudokus (A pesar de ser casi 1 Millón de muestras) no alcanza ni el 0.1 % de la población total de Sudokus, por lo cual no se conoce el verdadero rendimiento sobre una muestra significativa, debido al alto costo computacional que esto tendría. Por lo cual los buenos resultados presentados puede que se encuentren acotados a valores locales y no a los valores globales.
4. Es importante normalizar los datos pues se encontró que el modelo ANN tiene un mejor rendimiento cuando se realiza este proceso. Además se debe evaluar el rendimiento del modelo con una métrica adecuada, como sucedió en esta aplicación la sola exactitud tenia un error acumulado basado en las condiciones iniciales del Sudoku y se reemplazó por una métrica que penaliza la exactitud en función de si el Sudoku inicial tiene más o menos posiciones correctas.
5. El tiempo de entrenamiento el modelo 1, para 10k datos fue de 0.66 h, con 100k demoro 10 h y para el segundo modelo con 800k datos demoro 1.18 h. De los cuales el segundo modelo obtuvo un mejor rendimiento, de lo cual se puede inferir que el tiempo de entrenamiento no es necesariamente proporcional a tener una mejor modelo.

ACKNOWLEDGMENT

A Pachito

REFERENCIAS

[1] *Sudoku - Wikipedia, la enciclopedia libre*. dirección: https : / / es . wikipedia . org / wiki / Sudoku % 7B % 5C # %7Dcite%7B%5C_%7Dnote-2 (visitado 01-12-2020).

[2] B. D. McKay e I. M. Wanless, "On the number of Latin squares," *Annals of Combinatorics*, vol. 9, n.º 3, págs. 335-344, 2005, ISSN: 02180006. DOI: 10.1007/s00026-005-0261-7. arXiv: 0909.2101.

[3] B. Felgenhauer y F. Jarvis, "Enumerating possible Sudoku grids," *Mathematical Spectrum.*, págs. 3-9, 2005.

[4] G. McGuire, B. Tugemann y G. Civario, *There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem*, 2013. arXiv: 1201.0749.

[5] D. Liu, *A Practical Guide to ReLU*, nov. de 2017. dirección: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7#:~:text=ReLU%20stands%20for % 20rectified % 20linear , neural % 20networks , %20especially%20in%20CNNs..

[6] J. Brownlee, *Softmax Activation Function with Python*, jun. de 2020. dirección: https://machinelearningmastery.com/softmax-activation-function-with-python/.

[7] *tf.keras.losses.BinaryCrossentropy : TensorFlow Core v2.3.0*. dirección: https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy.

[8] *Solving Sudoku with Convolution Neural Network — Keras — by Shiva Verma — Towards Data Science*. dirección: https://towardsdatascience.com/solving-sudoku-with-convolution-neural-network-keras-655ba4be3b11 (visitado 03-12-2020).