

Autómatas Celulares Neuronales

Sergio Adrian Martínez Tena

Sistemas Complejos

Maestría en Ciencias e Ingeniería de la Computación

Universidad Nacional Autónoma de México

Resumen—Los autómatas celulares (AC) permiten modelar una gran variedad de sistemas que pueden ser descritos por un conjunto de objetos simples que interactúan localmente entre ellos. Estos objetos se representan por su posición y por una serie de estados discretos. Sus interacciones se modelan por un conjunto de reglas bien definidas, donde la aplicación iterativa de estas reglas da como resultado comportamientos complejos.

En este documento se propone dar una descripción y algunos ejemplos de una variación de los AC, los autómatas celulares neuronales (ACN). En donde, a diferencia de los AC "regulares" los estados de cada objeto se representan por valores continuos, y el conjunto de reglas se sustituyen por la aplicación de una convolución discreta, seguida por alguna función de activación. Estas variaciones permiten que los ACN tengan comportamientos muy variados, desde logrando reproducir resultados de los AC "regulares", hasta obtener comportamientos muy parecidos a un sistema de reacción-difusión.

Index Terms—Convolución, kernel, función de activación.

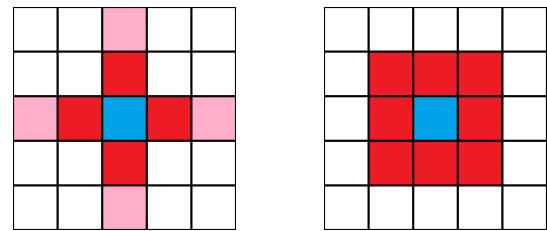
I. INTRODUCCIÓN

I-A. Autómata celular

Un autómata celular es un modelo matemático discreto de un sistema dinámico, el cual está compuesto por una cuadrícula de celdas (células) en un determinado número de dimensiones. Cada celda puede estar en un número finito de estados, como por ejemplo 0 y 1. Para cada celda se define un conjunto de celdas vecinas, donde esta vecindad puede tomar diferentes tamaños y formas. Dos de los tipos de vecindades más comunes (en dos dimensiones) son la *vecindad de von Neumann* (Fig. 1a) y la *vecindad de Moore* (Fig. 1b). El valor de los estados de cada celda evoluciona en unidades de tiempo discreto siguiendo un conjunto de reglas, las cuales se basan en el estado de la misma celda y en los estados de las celdas vecinas. Usualmente, la regla de evolución es la misma para todas las celdas, no cambia con el tiempo y se aplica simultáneamente a toda la cuadrícula. Algo muy importante son las condiciones iniciales de la retícula, ya que a partir de estas se da la evolución del sistema. Una de las propiedades fundamentales de los autómatas celulares es el tipo de cuadrícula con la que se construye, donde la más sencilla solo es una línea unidimensional [1] [2].

El tipo de autómata celular más sencillo es un automata binario unidimensional que solo toma en cuenta a sus celdas vecinas más cercanas. Un ejemplo de este tipo de autómatas celulares es la regla 30, en esta la vecindad se define con las primeras dos celdas adyacentes (Fig. 2a). El valor de la celda entonces cambia (entre "encendida" y "apagada") según un conjunto de reglas que dependen del valor de la misma y

el valor de sus dos vecinas (Fig. 2b). Si como condiciones iniciales solo está activa la celda del centro y aplicamos iterativamente la regla a los estados resultantes, después de 20 pasos obtenemos un patrón complejo (Fig. 2c). Existen 256 posibles automatas celulares elementales. Stephen Wolfram planteó un esquema, conocido como el código Wolfram, para asignar a cada regla un número de 0 a 255 que se ha convertido en estándar [3].



(a) Vecindad de von Neumann

(b) Vecindad de Moore

Figura 1: Tipos de vecindades comunes. La celda de azul representa la celda actual, las celdas rojas forman la vecindad de esta [1].

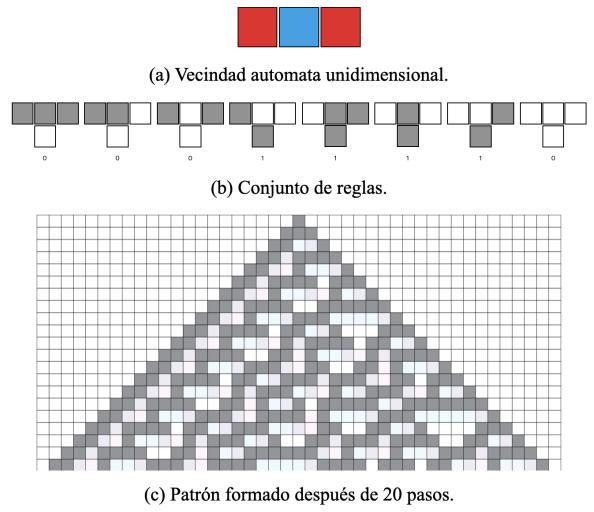


Figura 2: Definición regla 30. (a) Vecindad utilizada para la regla. (b) Actualización de estado según el estado de la misma y de sus vecinas. (c) Resultado de aplicar la regla durante 20 iteraciones, donde en un inicio solo la celda de centro está encendida.

Uno de los autómatas celulares mejor conocido es el juego de la vida, inventado por John H. Conway. Este consiste en una cuadrícula en dos dimensiones, en la que en cada generación las celdas cambian su valor ("encendida" o "apagada") dependiendo de sus ocho vecinos (*vecindad de Moore*) y de las siguientes reglas:

- Cualquier celda encendida con menos de dos vecinas encendidas se apaga.
- Cualquier celda encendida con dos o tres vecinas encendidas permanece encendida.
- Cualquier celda encendida con más de tres vecinas encendidas se apaga.
- Cualquier celda apagada con tres vecinos encendidos se enciende.

Esta combinación de reglas junto con las condiciones iniciales adecuadas produce una gran variedad de [patrones](#) inesperados [4].

I-B. Variaciones de los autómatas celulares

Existen muchas generalizaciones diferentes de los AC que han aparecido desde la introducción de los mismos.

Algunas variaciones son utilizando un tipo de retícula diferente. Por ejemplo, en lugar de tener celdas cuadradas, se pueden definir celdas hexagonales (Fig. 3a) o triangulares. De igual forma es factible emplear una retícula de geometría irregular (Fig. 3b). Realizar este tipo de variaciones permite conseguir otro tipo de vecindades y reglas.

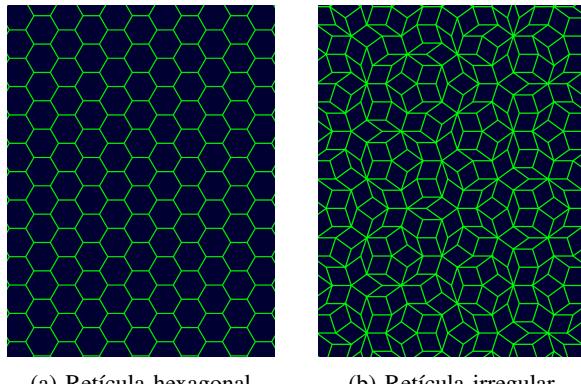


Figura 3: (a) Retícula de geometría hexagonal. (b) Retícula de geometría irregular.

Es posible hacer las reglas probabilísticas en lugar deterministas. Por ejemplo, se pueden definir las mismas reglas que en un AC "regular", pero a cada tiempo de evolución existe un porcentaje de probabilidad de que la celda evolucione al estado contrario definido por la regla. A esta variación de los AC se les llama autómatas celulares estocásticos.

La vecindad y las reglas del modelo también pueden cambiar en función del tiempo y/o del espacio.

Otra variación son los autómatas continuos, en estos el valor de los estados posibles es continuo (usualmente en el rango de $[0,1]$) en lugar de discreto. Sin embargo, las celdas permanecen discretamente separadas entre sí.

Asimismo, existen los automatas espaciales continuos, donde las celdas forman un continuo (en lugar de estar discretamente separadas) y el valor de los estados posibles también sigue siendo continuo. Un ejemplo es "SmoothLife" (Fig. 4) creado por Stephan Rafler. Este es una generalización de del "Juego de la vida" utilizando automatas espaciales continuos.

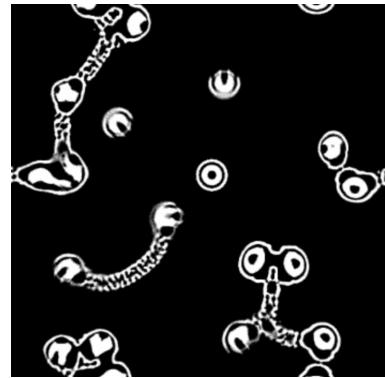


Figura 4: "SmoothLife" creado por Stephan Rafler. Generalización de del "Juego de la vida" utilizando automatas espaciales continuos [5].

I-C. Autómata celular neuronal

En [6] se menciona el uso de autómatas celulares neuronales con el fin de simular la morfogénesis (el proceso de desarrollo de la forma de un organismo). Lo que se pretende simular en este trabajo es como las células se comunican con su vecindad para decidir la forma de los órganos, donde hacer que se desarrollen, como interconectarlos y cuando detenerse. En [7] los ACN se emplean para simular como grupos de células deciden si un órgano o tejido es correcto, o si es necesario remodelar la anatomía actual. Por ejemplo, este fenómeno se puede apreciar cuando al trasplantar quirúrgicamente una cola de salamandra a su flanco, este tejido se transforma lentamente hasta lograr ser una extremidad.

Otra aplicación de los ACN es en la síntesis de texturas [8]. En esta se intenta reproducir la apariencia general de una plantilla de textura, en lugar de hacer una copia perfecta.

Aunque en los trabajos mencionados el uso de ACN en conjunto con redes neuronales profundas, es lo que a hecho posible modelar el comportamiento de estos fenómenos tan complejos. Los ACN por sí mismos también muestran la capacidad de producir cierta complejidad de aspecto orgánico.

Los elementos básicos de los ACN se definen, tanto en [6], [7] y [8], de la siguiente manera:

- El espacio es una retícula cuadrada.
- El conjunto de estados para la célula es continuo.
- La regla de actualización es la aplicación de una convolución entre un kernel y el espacio, seguido de una función de activación.
- La vecindad de la célula se define por el tamaño del kernel (usualmente 3x3).
- Se considera una frontera periódica.

La figura 5 muestra los elementos básicos de un ACN.

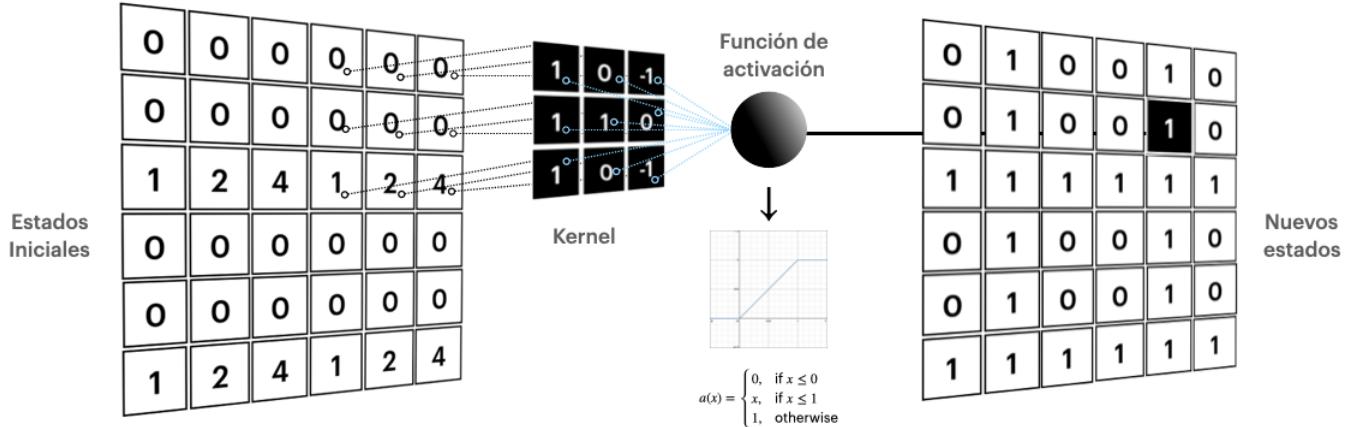


Figura 5: Elementos básicos de un ACN. La función de actualización es aplicar una convolución, seguido de una función de activación. Los estados para la célula son valores continuos y su vecindad se define por el tamaño del kernel.

I-D. Convolución

Una convolución es una operación matemática que transforma dos funciones (f y g) en una tercera función ($f * g$).

Desde el punto de vista discreto en dos dimensiones, una convolución es el proceso de añadir cada elemento de una matriz, ponderados por otra matriz (kernel). Este kernel se desliza sobre cada elemento de la primera matriz hasta obtener otra matriz de salida (Fig. 6) [9].

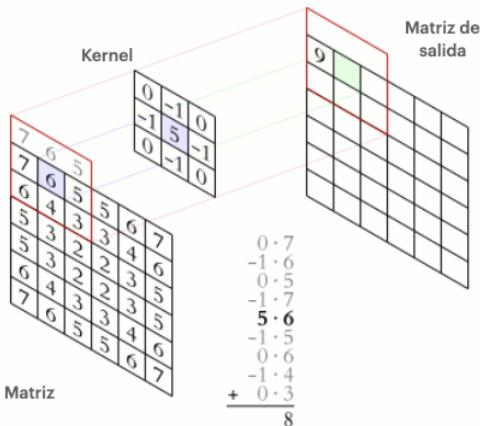


Figura 6: Ejemplo de convolución discreta para un solo elemento [10].

En procesamiento de imágenes las convoluciones se utilizan para la extracción de características, con el fin de determinar las partes más importantes de una imagen. En una imagen esta operación puede desenfocar, enfocar, detectar bordes y más.

I-E. Funciones de activación

En redes neuronales, una función de activación define la salida de un nodo dada una entrada o conjunto de entradas. El

rol principal de una función de activación es transformar las entradas ponderadas y sumadas en un valor de salida (Fig. 7), donde su propósito es añadir no linealidad. Existen muchas funciones de activación con diferentes propiedades. Una de las más sencillas es la función de paso binario, en la cual dada una entrada se compara con un umbral, si la entrada es mayor al umbral, la neurona se activa, de lo contrario se desactiva [9]. La figura 8 muestra algunas funciones de activación comúnmente utilizadas.

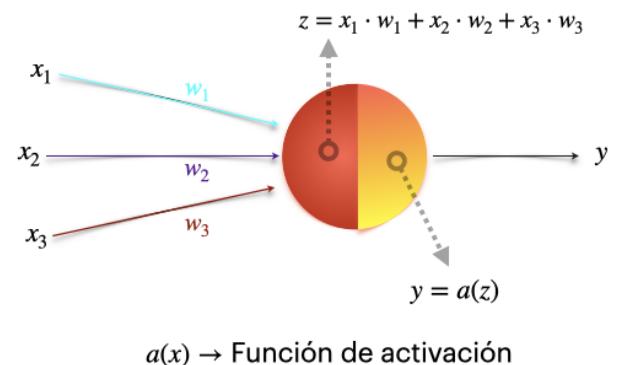


Figura 7: Una función de activación transformar las entradas ponderadas y sumadas en un valor de salida.

II. OBJETIVOS

- Implementar el algoritmo de funcionamiento de los autómatas celulares neuronales.
- Utilizar diferentes kernel y funciones de activación, con el fin de obtener diferentes comportamientos.

Name	Plot	Function, $g(x)$
Identity		x
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1 + e^{-x}}$
Rectified linear unit (ReLU) ^[8]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x1_{x>0}$

Figura 8: Ejemplos de algunas funciones de activación [11].

III. DESARROLLO

III-A. Algoritmo del autómata celular neuronal

La implementación se realizó utilizando *Python*. Primariamente, se definió el espacio de las células, para ello se recurrió a la biblioteca *numpy*, con el fin de formar una matriz de valores de ancho y alto definidos. Luego esta matriz se inicializó con algunos valores (específicos o aleatorios). El kernel y la función de activación también fueron previamente seleccionados.

El primer paso del algoritmo es aplicar la convolución. Esta se computó empleando la biblioteca *scipy*, específicamente el método *convolve2d*. Este método recibe como argumentos las matrices a convolucionar (matriz con valores iniciales y kernel), un argumento indicando el tamaño de la matriz de salida ("mode") y otro indicando las condiciones de frontera ("boundary"). Para "mode" se seleccionó "same", este indica que el tamaño de la matriz de salida es el mismo que el de la matriz de entrada. Para "boundary" se seleccionó "wrap", indicando que se debe considerar una frontera periódica para el cálculo. El segundo paso es aplicar la función de activación (previamente definida) a cada elemento de la matriz resultante tras la aplicación de la convolución. Por último, se recortan los valores entre 0 y 1, donde valores mayores a 1 pasan a tener un valor de 1 y valores menores a 0 pasan a tener un valor de 0. Estos tres pasos generan la matriz con los nuevos estados de las células.

Posteriormente, podemos iterar los tres pasos anteriores un número N (computable) de veces, luego almacenar cada matriz generada en el tiempo n en otro arreglo de *numpy*. Este último arreglo contiene la evolución del autómata celular neuronal durante N pasos.

III-B. Generación de la animación

El arreglo que contiene la evolución del autómata pasa por un fragmento de código que produce el video de la simulación. Para lo cual se empleó la biblioteca *OpenCV*, concretamente primero se crea una instancia de la clase *VideoWriter*, donde los argumentos definen la configuración del video a producir (nombre del archivo, formato del video, fotogramas por segundo, resolución, entre otros). Luego, el método *write* de la instancia de *VideoWriter*, previamente generada, recibe una

matriz a la vez del arreglo con toda la evolución. Donde finalmente el método *release* genera el video final.

III-C. Algunos autómatas celulares neuronales

III-C1. Replicando comportamientos de AC "regulares": Primeramente, se replicó el comportamiento de la "regla 30". Como condiciones iniciales solo la celda del centro en el primer renglón tiene un valor de 1, todas las demás son 0. El kernel y función de activación requeridos son los siguientes:

$$\text{kernel} = [1, 2, 4] \quad a(x) = \begin{cases} 1, & \text{if } x = 1 \\ 1, & \text{if } x = 2 \\ 1, & \text{if } x = 3 \\ 1, & \text{if } x = 4 \\ 0, & \text{if otherwise} \end{cases}$$

Luego, se replicó el "juego de la vida". Las condiciones iniciales de cada célula son valores aleatorios discretos (0 o 1). El kernel y función de activación requeridos son los siguientes:

$$\text{kernel} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 9 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad a(x) = \begin{cases} 1, & \text{if } x = 3 \\ 1, & \text{if } x = 11 \\ 1, & \text{if } x = 12 \\ 0, & \text{if otherwise} \end{cases}$$

III-C2. ACNs con comportamientos de aspecto orgánico: Comportamiento parecido al Moho. Para esto las condiciones iniciales fueron 5 celdas elegidas aleatorias con valor de 1. El kernel y función de activación requeridos son los siguientes:

$$\text{kernel} = \begin{bmatrix} -0.85 & 0.917 & -0.85 \\ 0.917 & 0. & 0.917 \\ -0.85 & 0.917 & -0.85 \end{bmatrix}$$

$$a(x) = \sin(\text{abs}(\frac{x}{2}))$$

Comportamiento parecido a gusanos. En este caso, las condiciones iniciales de todas las células son aleatorias (producidas por una función uniforme) en el rango de [0,1]. El kernel y función de activación requeridos son los siguientes:

$$\text{kernel} = \begin{bmatrix} 0.8 & -0.85 & 0.8 \\ -0.85 & -0.2 & -0.85 \\ 0.8 & -0.85 & 0.8 \end{bmatrix}$$

$$a(x) = \frac{-1}{0.89x^2 + 1} + 1$$

Comportamiento parecido a la mitosis celular. Las condiciones iniciales de todas las células son aleatorias en el rango de [0,1]. El kernel y función de activación requeridos son los siguientes:

$$\text{kernel} = \begin{bmatrix} -0.94 & 0.88 & -0.94 \\ 0.88 & 0.4 & 0.88 \\ -0.94 & 0.88 & -0.94 \end{bmatrix}$$

$$a(x) = \frac{-1}{0.9x^2 + 1} + 1$$

III-C3. ACNs con comportamientos misceláneos: Generación de caminos. Las condiciones iniciales de todas las células son aleatorias en el rango de [0,1]. El kernel y función de activación requeridos son los siguientes:

$$\text{kernel} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$a(x) = \frac{-1}{2(x-b)^2}$$

Olas. Las condiciones iniciales de todas las células son aleatorias en el rango de [0,1]. El kernel y función de activación requeridos son los siguientes:

$$\text{kernel} = \begin{bmatrix} 0.5646 & -0.7159 & 0.5646 \\ -0.7159 & 0.627 & -0.7159 \\ 0.5646 & -0.7159 & 0.5646 \end{bmatrix}$$

$$a(x) = \text{abs}(1.2x)$$

IV. RESULTADOS

La figura 9 muestra el resultado de replicar el comportamiento de la "regla 30". El ACN que implementa el "juego de la vida" se muestra en la figura 10.

En la figura 11 se observa un comportamiento parecido al moho. El comportamiento parecido al movimiento de gusanos y a la mitosis celular, se muestra en la figura 12 y 13 respectivamente.

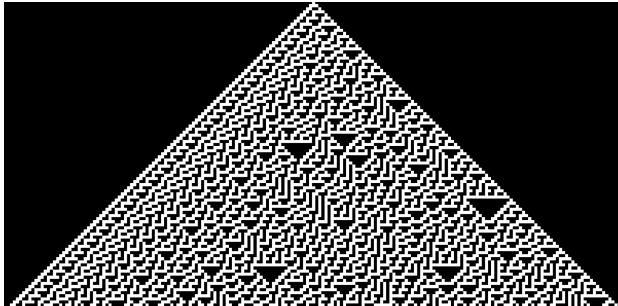


Figura 9: "Regla 30" replicada con un ACN.

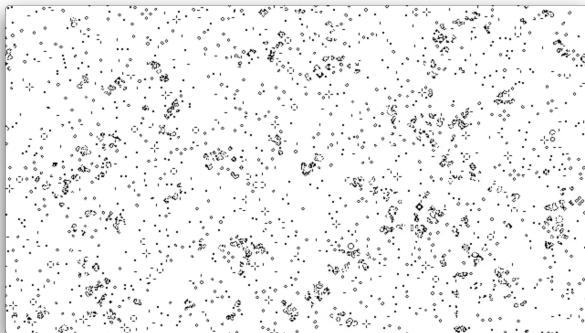


Figura 10: "Juego de la vida" replicado con un ACN.

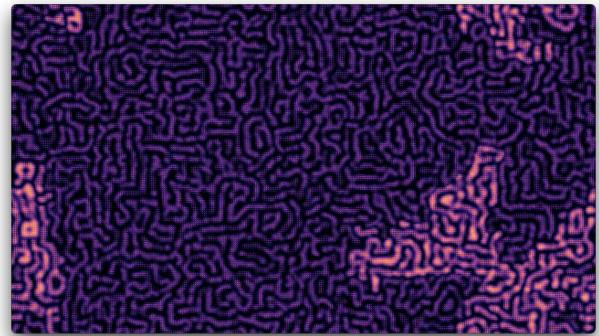


Figura 11: ACN con comportamiento parecido al moho.



Figura 12: ACN con comportamiento parecido gusanos.



Figura 13: ACN con comportamiento parecido a la mitosis celular.

Los ACN con comportamientos misceláneos, como la generación de caminos y la formación de olas, se pueden observar en las figuras 14 y 15.

Todos los resultados muestran solo un fotograma de la simulación. Donde el fotograma que se seleccionó, se consideró como el más representativo.

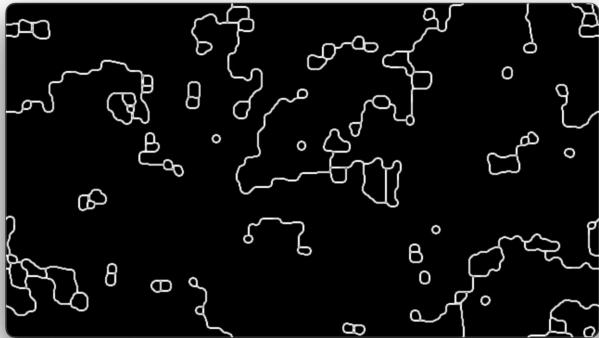


Figura 14: Generación de caminos utilizando un ACN.

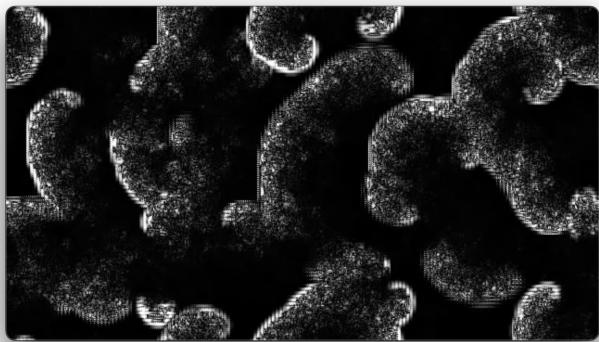


Figura 15: ACN con comportamiento parecido a las olas.

La mayoría de las combinaciones de kernel y función de activación se obtuvieron del canal de YouTube *@Emergent-Garden* [12] y de su comunidad en Discord [13].

V. DISCUSIÓN

Algo importante que mencionar es el recorte realizado después de la aplicación de la función de activación, este fue necesario debido a que los estados del autómata se representan como una imagen. Al no recortarlos, la mayoría de las veces, provoca que algunas funciones de activación (las que tienen un rango diferente a $[0,1]$) generen imágenes saturadas (sin ningún comportamiento notable).

Es interesante ver como los ACNs son capaces de replicar los comportamientos de reglas discretas, como es el caso de las figuras 9 y 10.

Otra cosa muy reléate es como las figuras 11, 12 y 13, muestran una evolución parecida a la de los sistemas de reacción-difusión, generando un comportamiento con aspecto orgánico.

VI. CONCLUSIONES

Aunque la implementación de los ACNs a grandes rasgos es muy sencilla, es impresionante la gran variedad de com-

portamientos complejos que emergen (incluso variando muy poco los valores en el kernel).

Algo sumamente interesante es el uso de estos en conjunto con redes neuronales, ya que permiten modelar organismos aún más complejos. Aun así, es importante explorar este tipo de autómatas celulares por sí mismos, dado que son fundamentales para el desarrollo de modelos muy interesantes, como los mostrados en [6], [7] y [8].

En cuanto a la generación de la simulación, aunque la mayoría de los ACNs utiliza el programa mencionado, algunos (como la "regla 13") requirieron de algunas modificaciones para no obtener un resultado trivial. Las modificaciones se muestran en el código fuente.

No se tuvo ningún problema con la implementación. Sin embargo, cabe mencionar que hay espacio para mejora en cuanto a tiempos de ejecución.

VII. CÓDIGO FUENTE

```

# Se crea el video
for i in tqdm(range(duration),
              desc='Creating video'):
    if skip_frames:
        if i%2 != k:
            data = (frames[i]*255)
            data = data.astype('uint8')
            out.write(data)
        else:
            data = (frames[i]*255)
            data = data.astype('uint8')
            out.write(data)
    out.release()

if __name__ == '__main__':
    # Tamano del espacio
    weight = 640
    height = 360
    init_frame = np.zeros((height, weight))
    # Numero de iteraciones
    N = 2000
    # Condiciones iniciales
    init_frame[3*height//4,weight//3] = 1
    init_frame[height//4,2*weight//3] = 1
    init_frame[5*height//6,5*weight//6] = 1
    init_frame[5*height//6+1,5*weight//6] = 1
    init_frame[5*height//6,5*weight//6+1] = 1
    # Se selecciona el kernel
    kernel = np.array(
        [[-0.84899998, 0.912, -0.84899998],
         [ 0.912, 0, 0.912],
         [-0.84899998, 0.912, -0.84899998]])
    # Se define la funcion de activacion
    activation = lambda x: np.sin(np.abs(x/2))
    # Se computan las N iteraciones
    frames = create_animation(init_frame, kernel,
                               activation,
                               N, persistent=False)
    # Se crea y exporta el video
    export_video_cv2(frames, name='moho.mp4',
                      fps=60, skip_frames=True,
                      skip='odd')

```

REFERENCIAS

- [1] Wikipedia, “Cellular automaton.” [Online]. Available: https://en.wikipedia.org/wiki/Cellular_automaton
- [2] W. MathWorld, “Cellular automaton.” [Online]. Available: <https://mathworld.wolfram.com/CellularAutomaton.html>
- [3] Wikipedia, “Elementary cellular automaton.” [Online]. Available: https://en.wikipedia.org/wiki/Elementary_cellular_automaton
- [4] W. MathWorld, “Game of life.” [Online]. Available: <https://mathworld.wolfram.com/GameofLife.html>
- [5] S. Murphy, “Smoothlife.” [Online]. Available: <https://github.com/duckythescientist/SmoothLife>
- [6] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, “Growing neural cellular automata,” *Distill*, 2020, <https://distill.pub/2020/growing-ca>.
- [7] E. Randazzo, A. Mordvintsev, E. Niklasson, M. Levin, and S. Greydanus, “Self-classifying mnist digits,” *Distill*, 2020, <https://distill.pub/2020/selfforg/mnist>.
- [8] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin, “Self-organising textures,” *Distill*, 2021, <https://distill.pub/2021/textures>.
- [9] R. Szeliski, *Computer vision. Algorithms and applications*. London; New York: Springer, 2011. [Online]. Available: <http://dx.doi.org/10.1007/978-1-84882-935-0>
- [10] Wikipedia, “Kernel (image processing).” [Online]. Available: [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))
- [11] ———, “Activation function.” [Online]. Available: https://en.wikipedia.org/wiki/Activation_function
- [12] Y. Garden, “Neural patterns.” [Online]. Available: <https://www.youtube.com/@EmergentGarden/playlists>

- [13] Discord, “Emergent garden.” [Online]. Available: <https://discord.gg/PEqFp8ad>