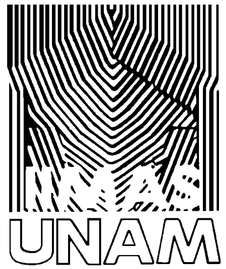
# Autómata Celular Neuronal

## Sistemas Complejos

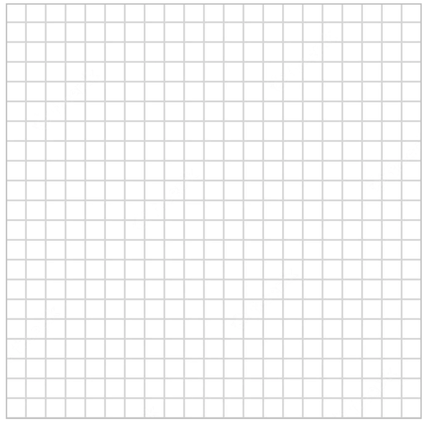

Sergio Adrian Martínez Tena



# Autómatas Celulares

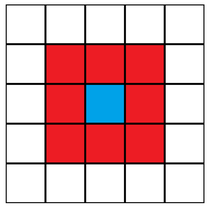
- Un espacio regular (retícula cuadrada) donde cada división es una célula.
- Conjunto de estados para la célula.
- Definir vecindades.
- Reglas de transición (función) que toman en cuenta la vecindad y el estado de la célula.
- Configuración inicial de espacio regular.

Espacio



Estados posibles


Vecindad

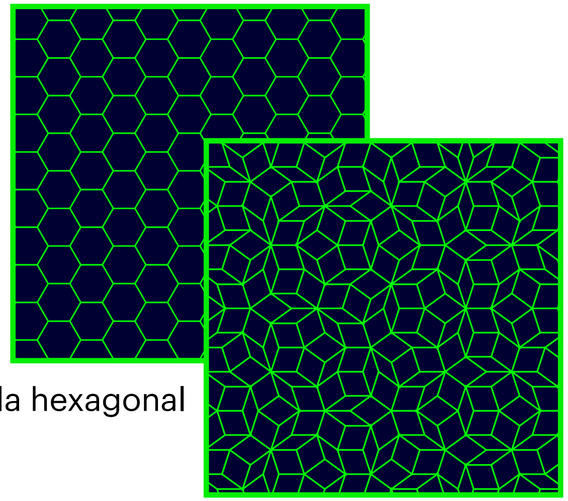


Reglas

```
def rules(cell, cellNeighbours):
    if cell == 1:
        return 1
    if sum(cellNeighbours) == 1 or sum(cellNeighbours) == 2:
        return 1
    else:
        return 0
```

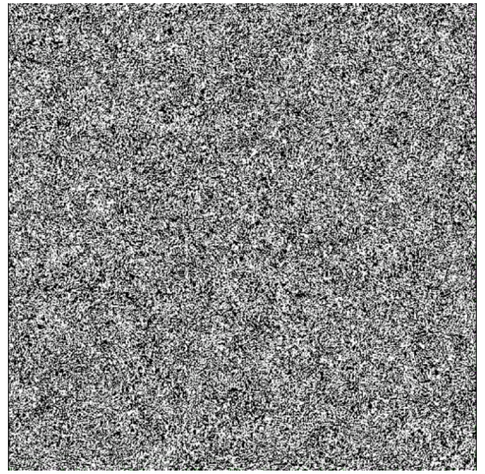
# Variaciones de autómatas celulares

- Utilizando retículas diferentes
- Reglas probabilísticas.
- Reglas y/o vecindades que cambian en función del tiempo y el espacio.
- Autómatas continuos, en estos el valor de los estados es continuo (usualmente en el rango de [0,1]).
- Autómatas espaciales continuos, tanto la localización de la celda como sus estados son continuos.

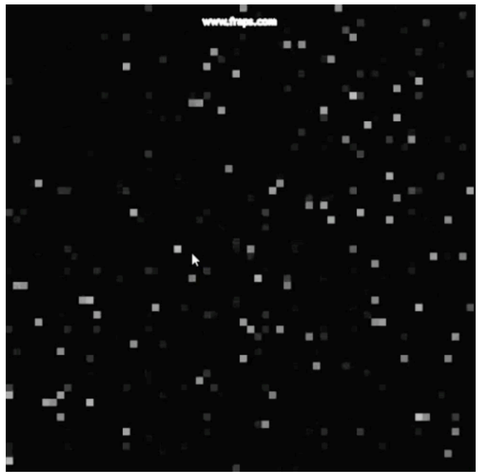


Retícula hexagonal

Retícula irregular



@Bencurlis (YouTube)



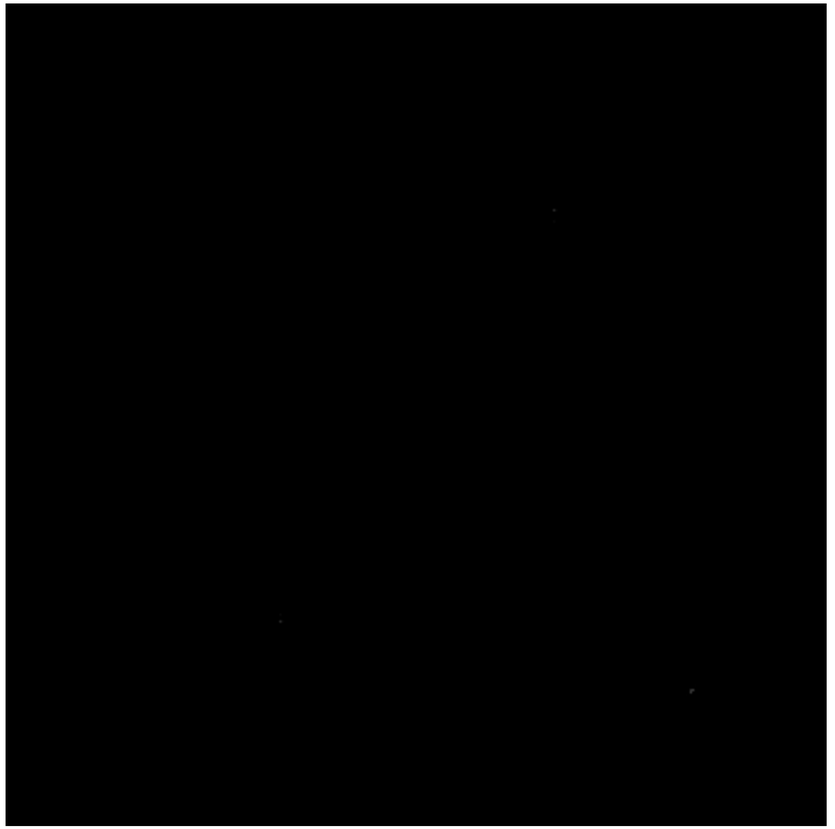
@zmeeshejka (YouTube)

SmoothLife (Stephan Rafler)

# Autómatas celulares neuronales

- El espacio es una retícula cuadrada.
- El conjunto de estados para la célula es continuo.
- Las reglas son aplicar una convolución seguida de una función de activación.
- La vecindad se define por el tamaño del kernel.
- Configuración inicial aleatoria o específica.

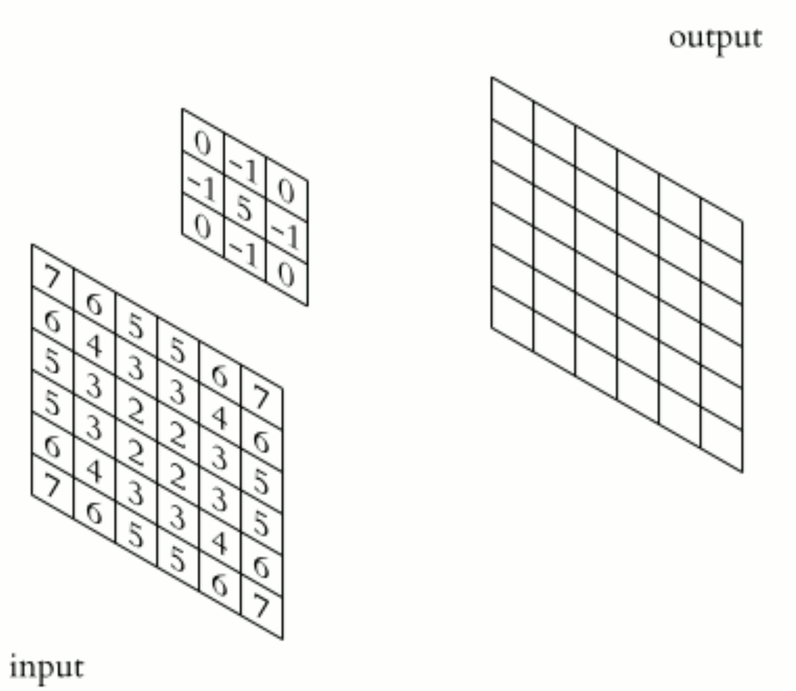
$\texttt{kernel}_{7x7}$



# 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



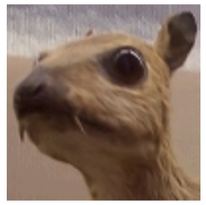

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right) = \begin{aligned}&(i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) \\ &+ (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).\end{aligned}$$

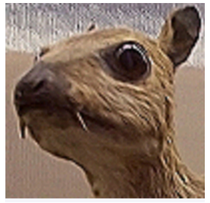
# Convolución

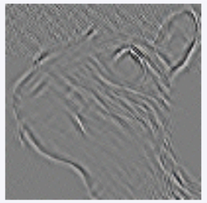
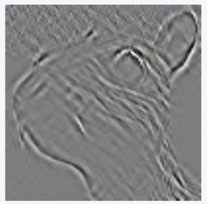
En una imagen esta operación puede desenfocar, enfocar, detectar bordes y más.

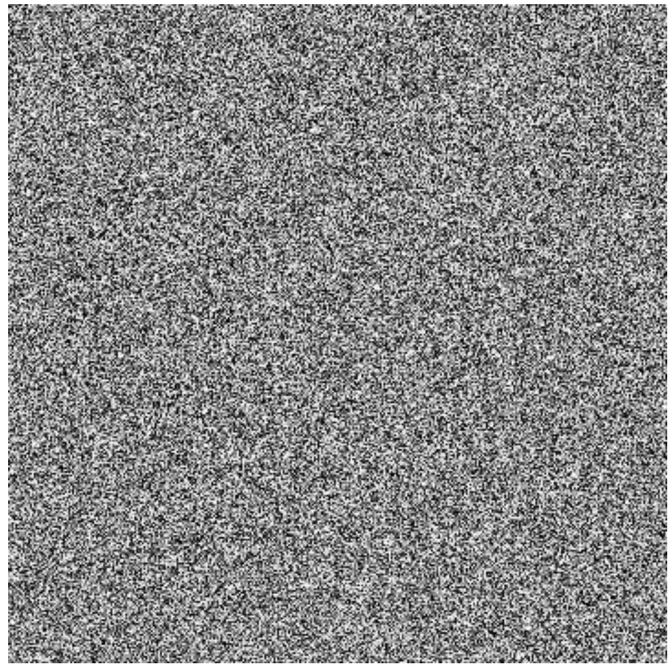
kernel =

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 4 | -1 |
| -1 | -1 | -1 |



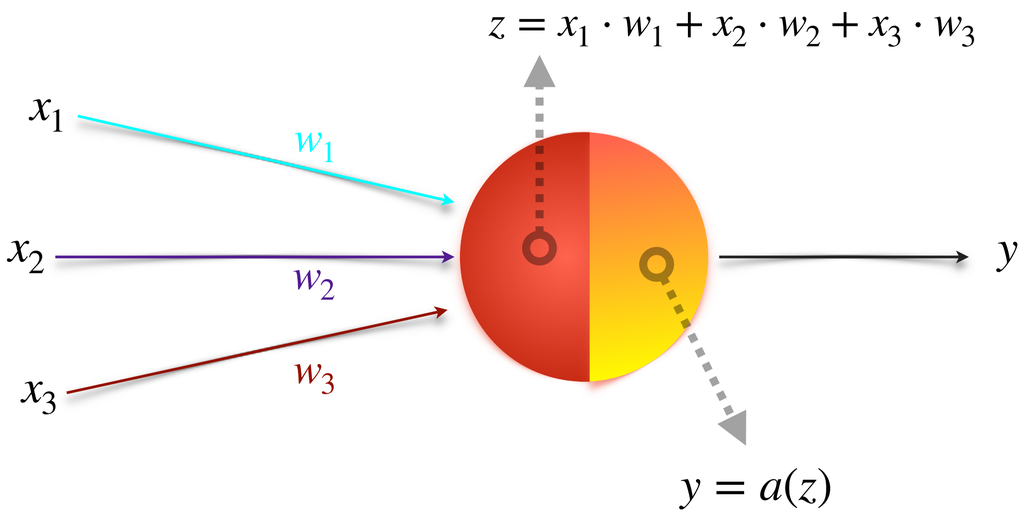


| | | |
|---|---|---|
| **Ridge** or **edge detection** | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 4 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |



$T = 300$

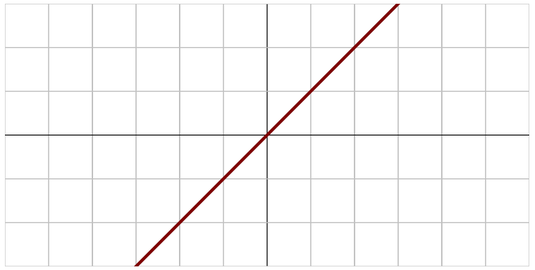
# Funciones de activación



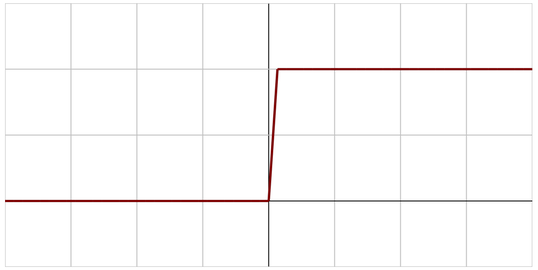

En redes neuronales, una función de activación define la salida de un nodo dada una entrada o conjunto de entradas. El rol principal de una función de activación es transformar las entradas ponderadas y sumada en un valor de salida.

$a(x) \rightarrow$ Función de activación

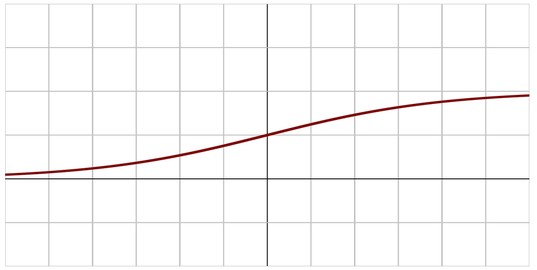
$a(x) = x$



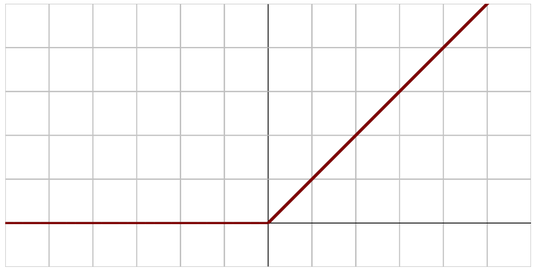
$$a(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$$



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



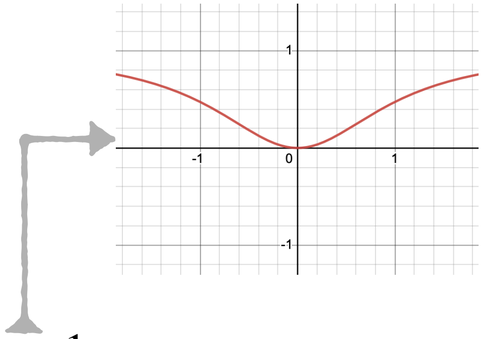
$a(x) = \max(0,x)$



# Funciones de activación

kernel =

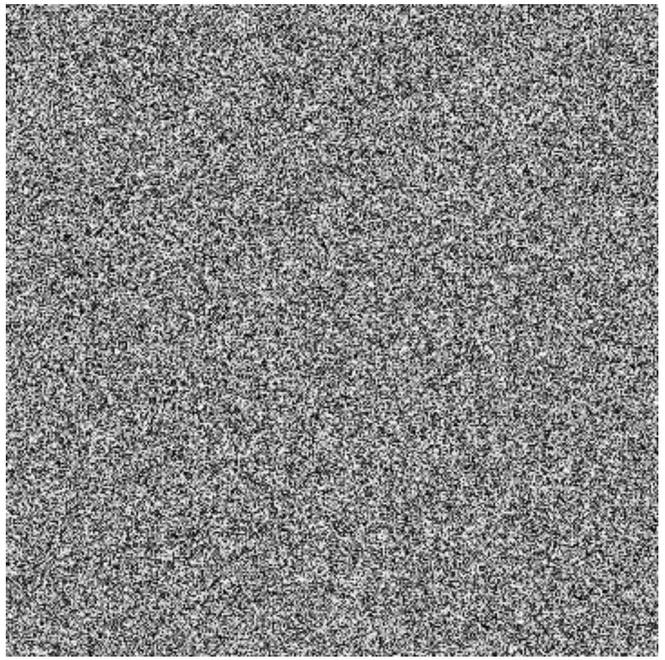
| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 4 | -1 |
| -1 | -1 | -1 |

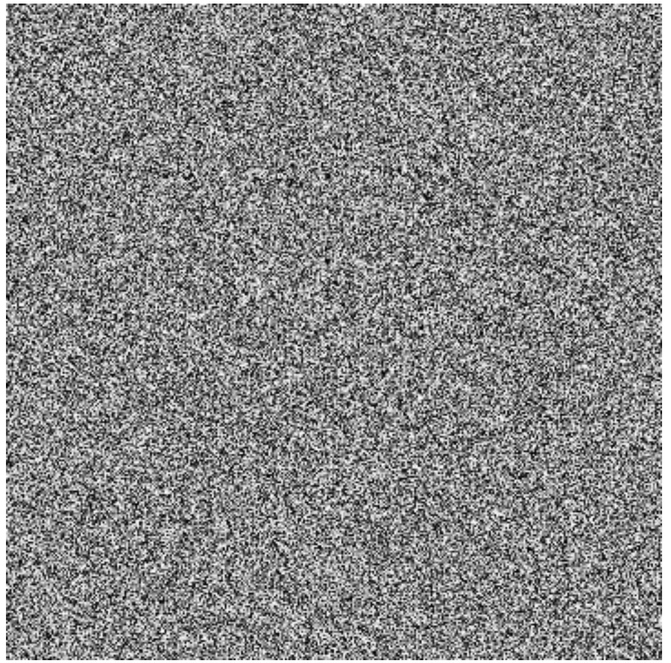


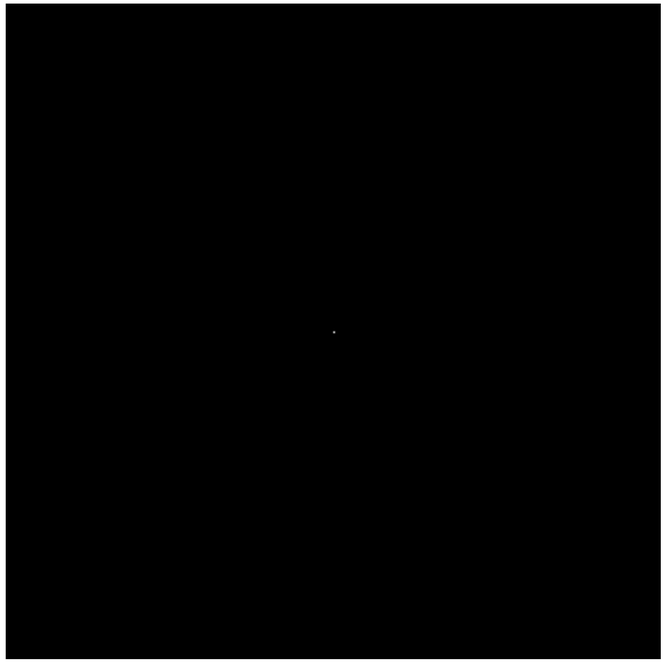

$$a(x) = x$$

$$a(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$$

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



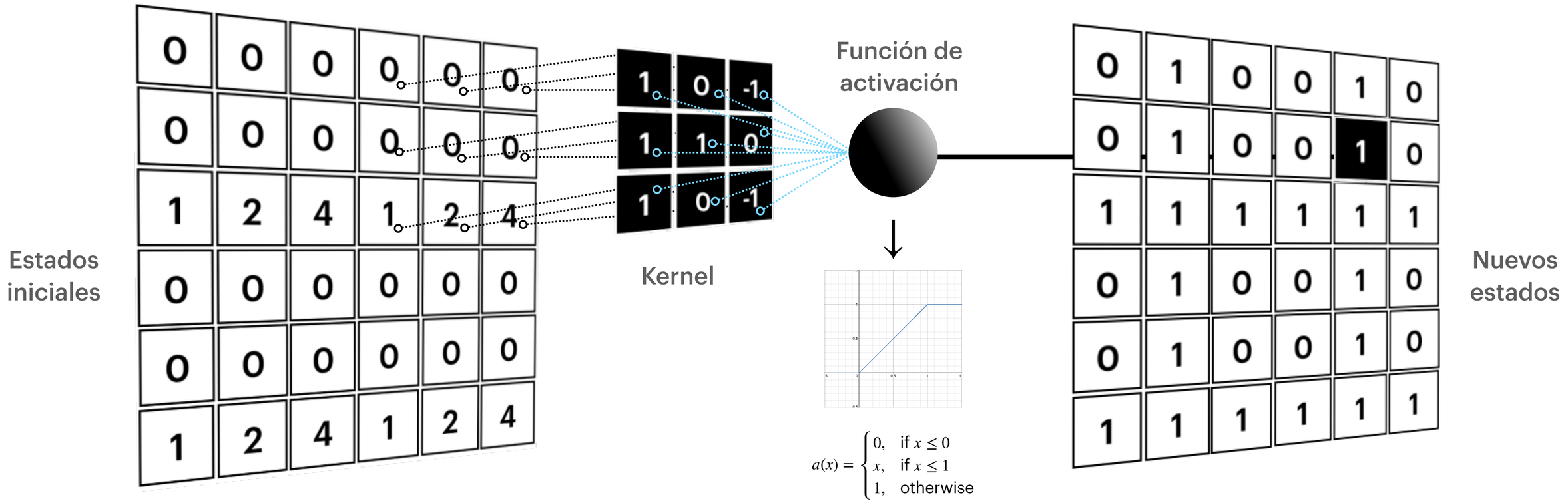




$$T = 300$$

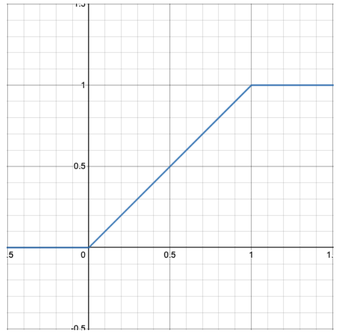
*La salida final se recorta entre 0 y 1.

# Autómatas celulares neuronales

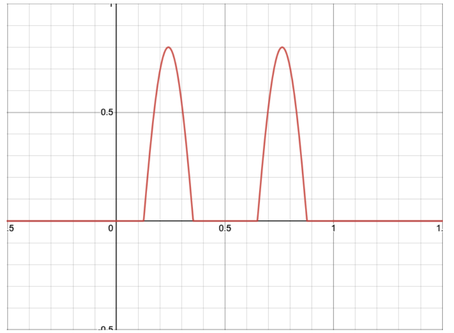


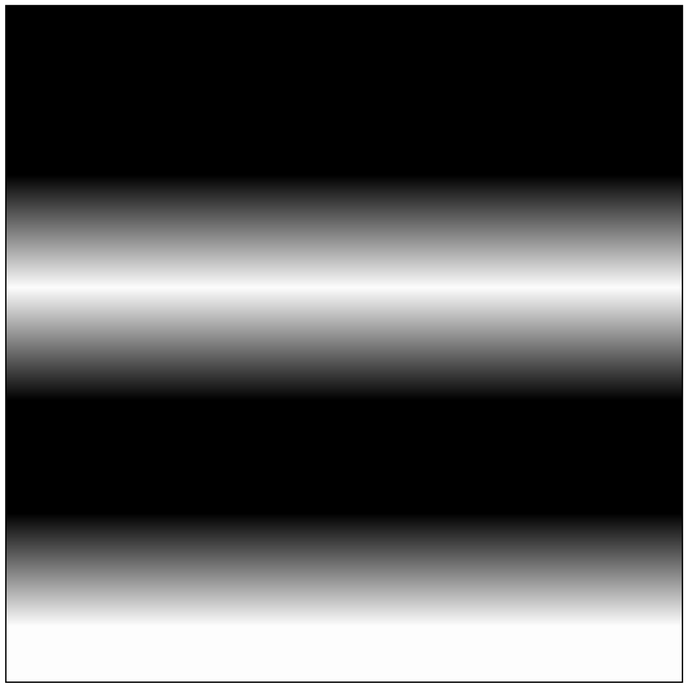
# Autómatas celulares neuronales

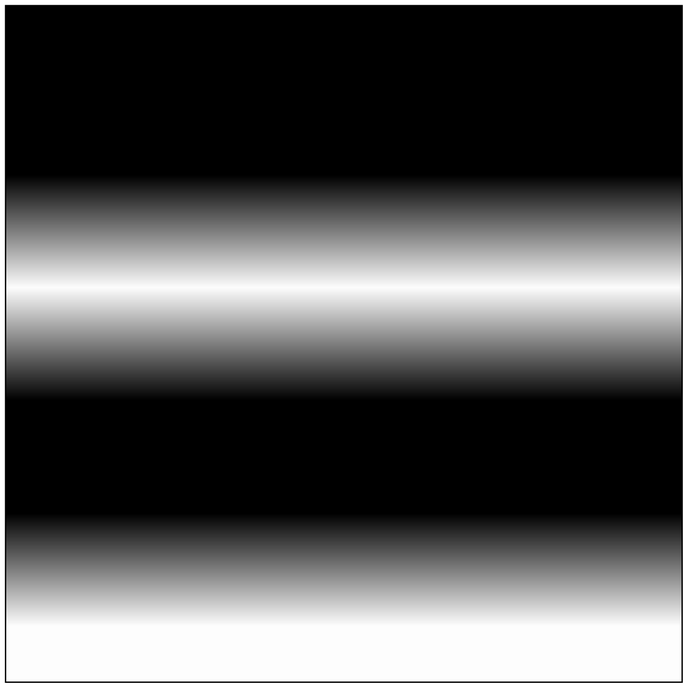
$$a(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x \leq 1 \\ 1, & \text{otherwise} \end{cases}$$



$$a(x) = \begin{cases} sin(12x - 1.3) - 0.2, & \text{if } 0 \leq x \leq 1 \\ 0, & \text{otherwise} \end{cases}$$







$T = 10$

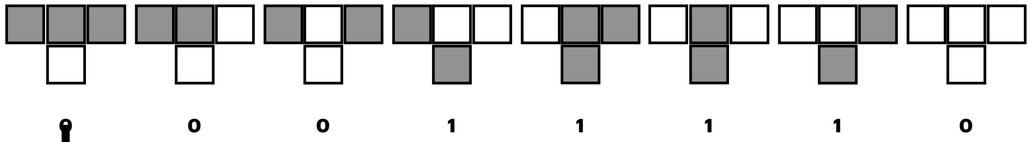
# Resultados

# Regla 30

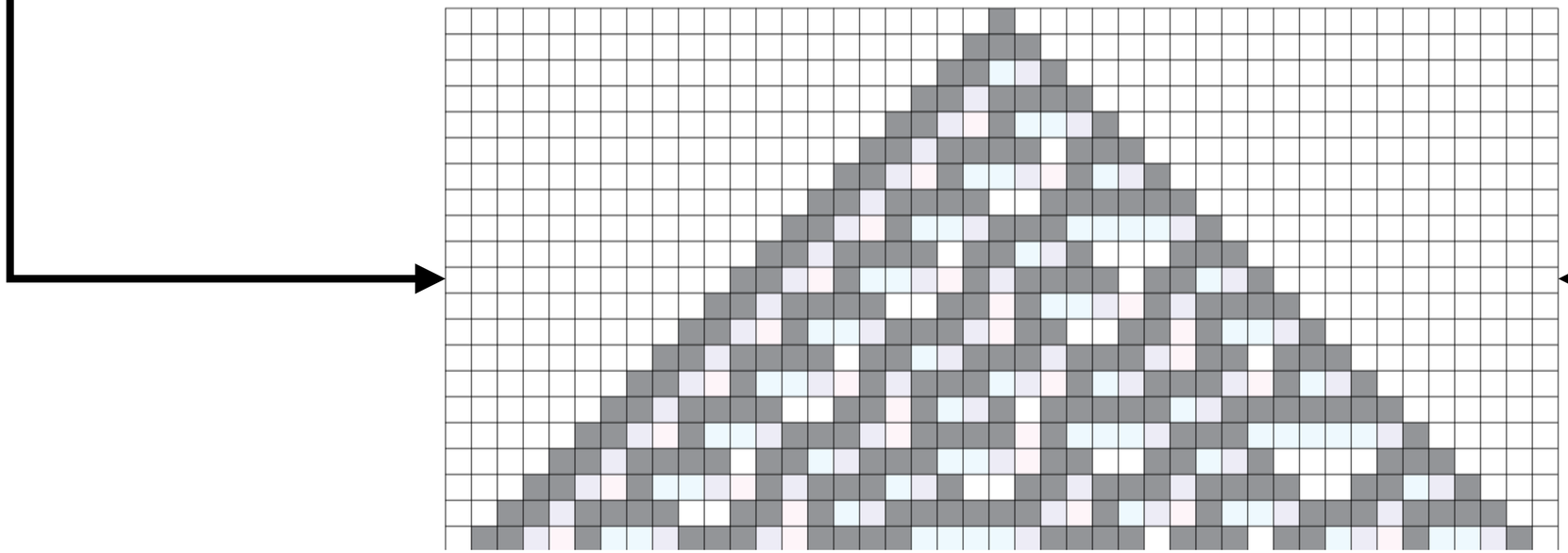
Vecindad

kernel = | 1 | 2 | 4 |

Reglas

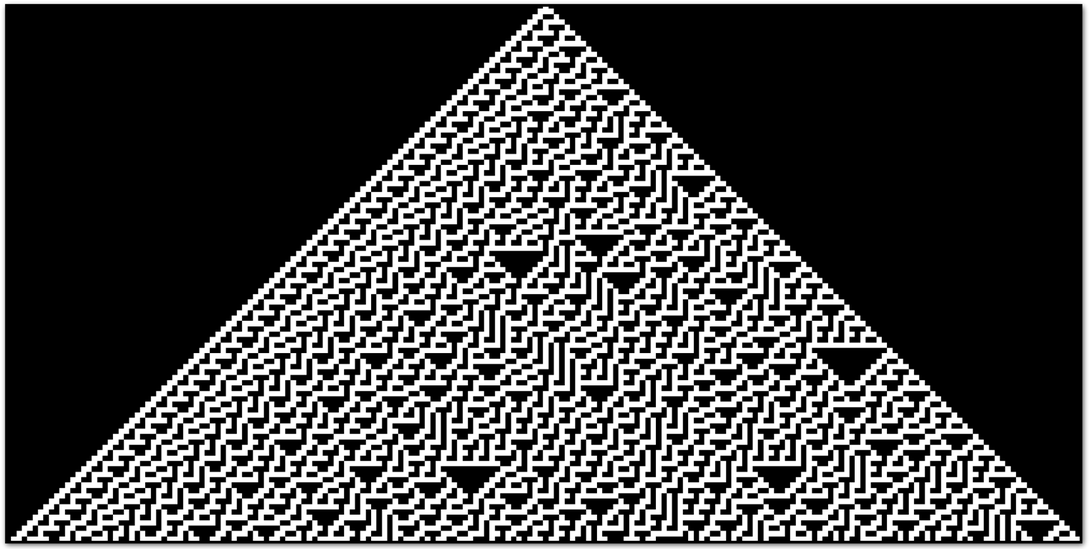


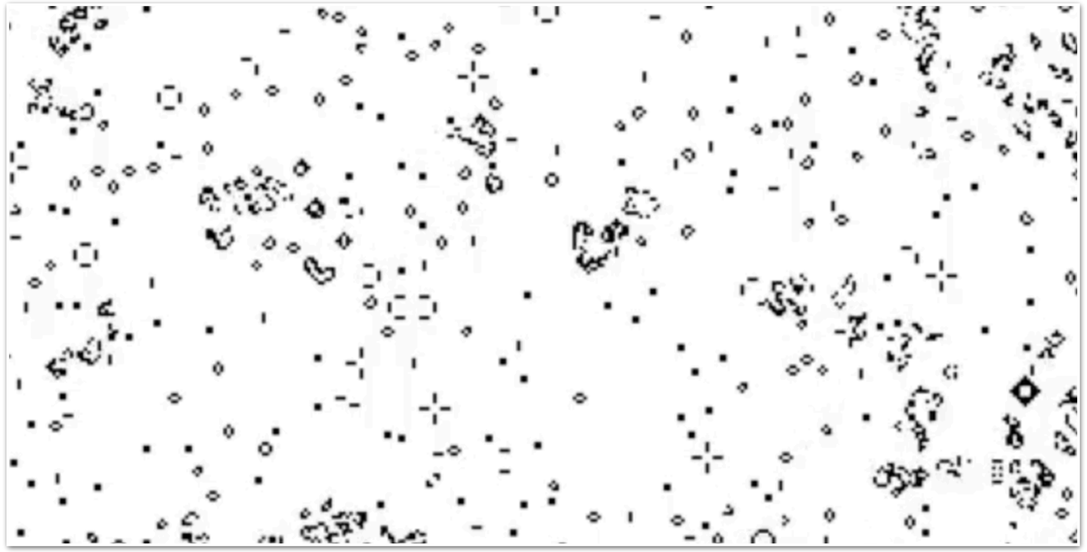

```
def activation(x):
   if (x == 1 | x == 2 | x == 3 | x == 4):
      return 1
   else:
      return 0
```

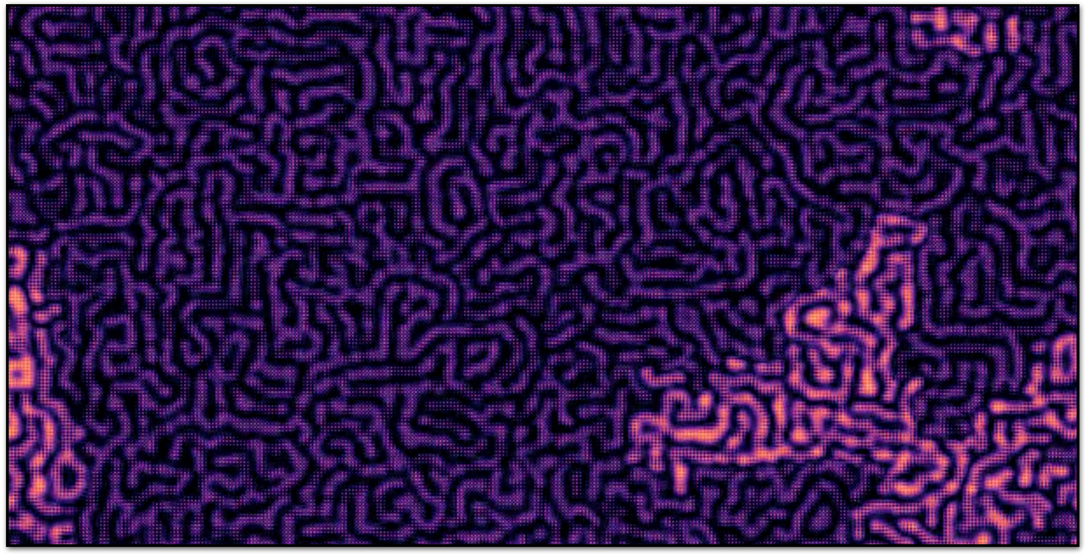


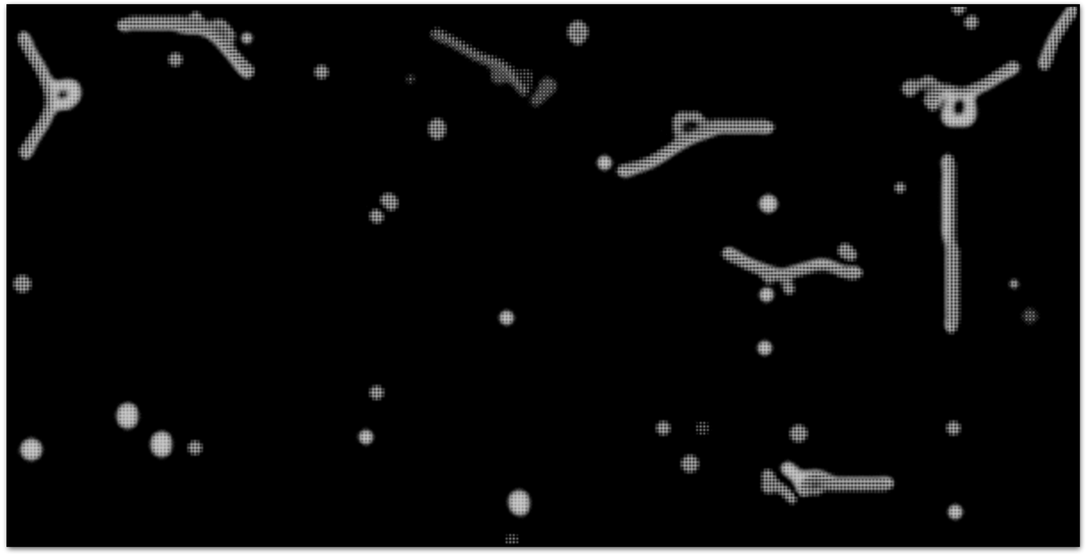
20 pasos.

# Algunos resultados

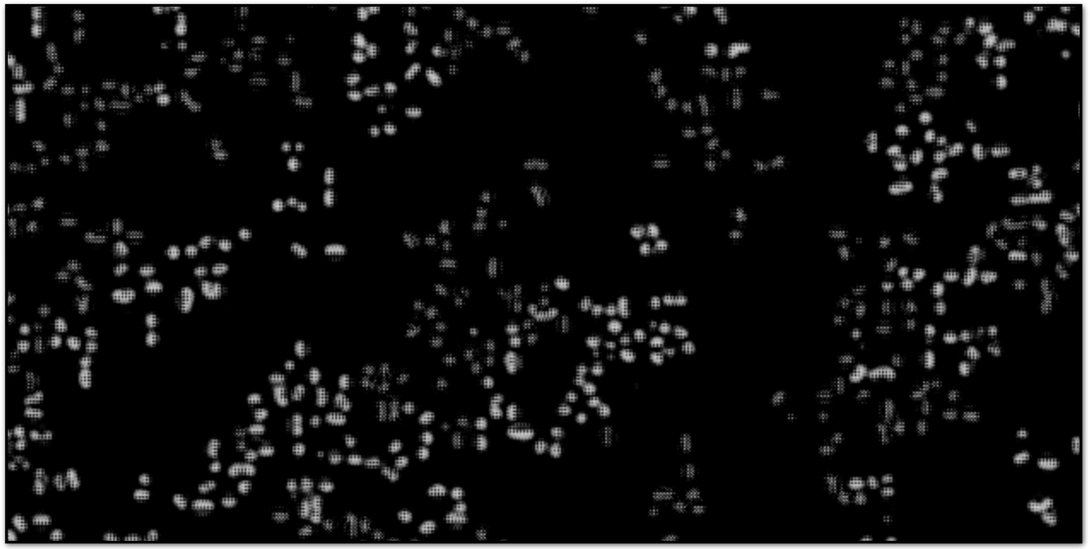


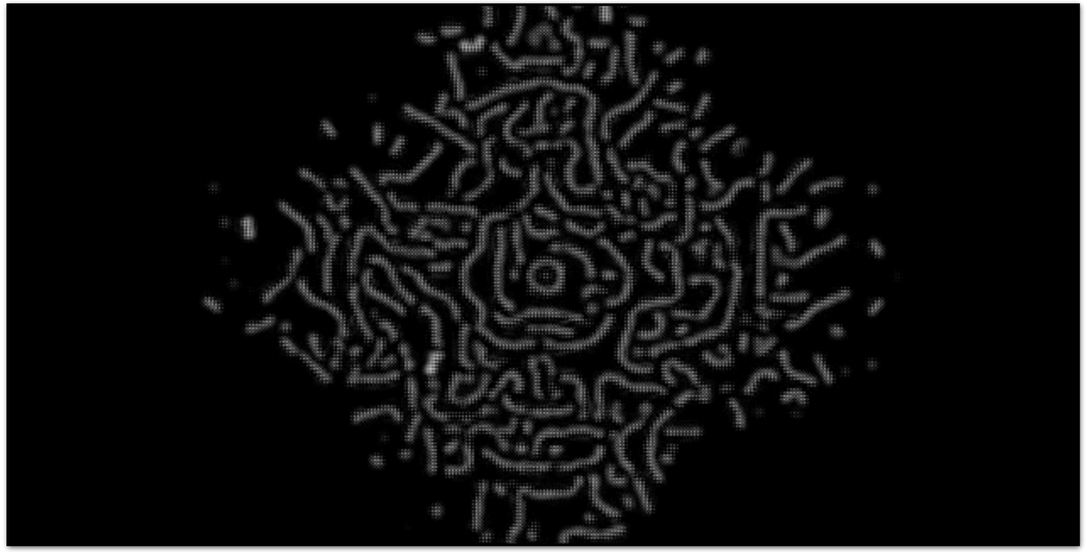


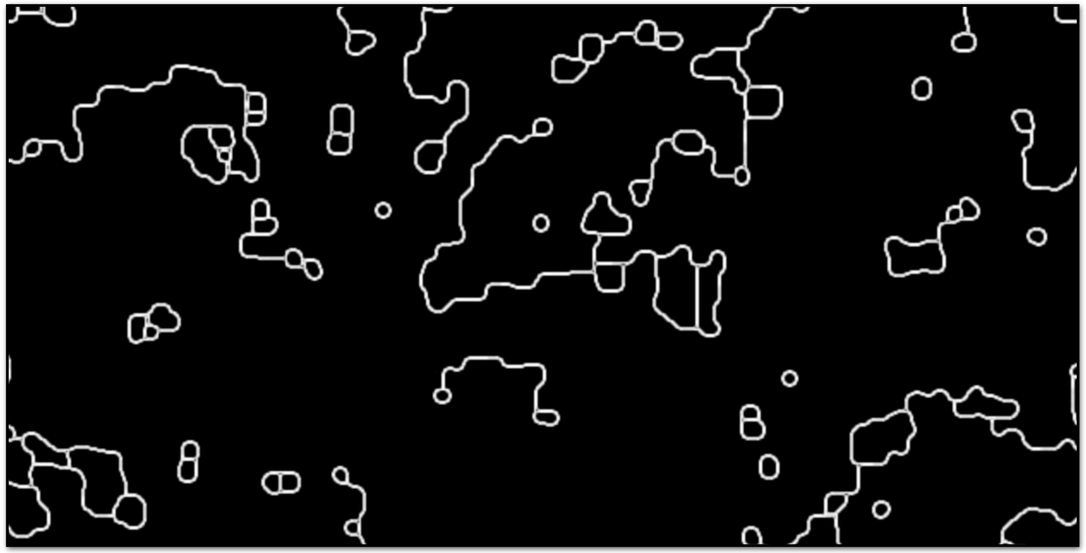


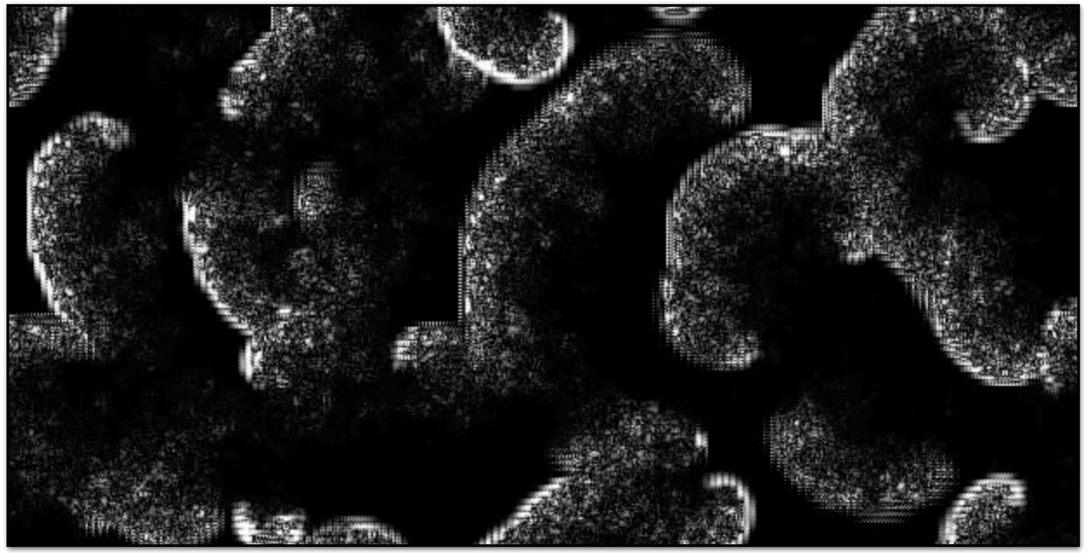


# Algunos resultados

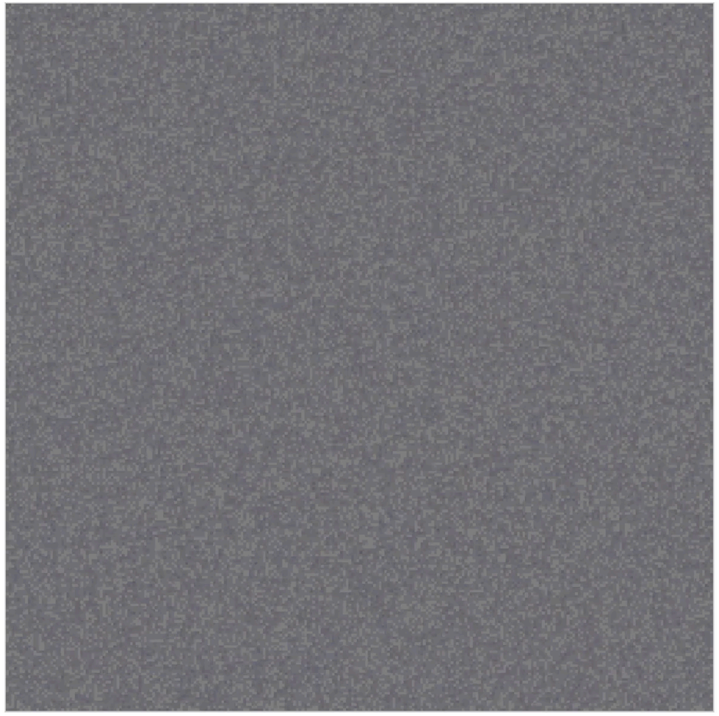




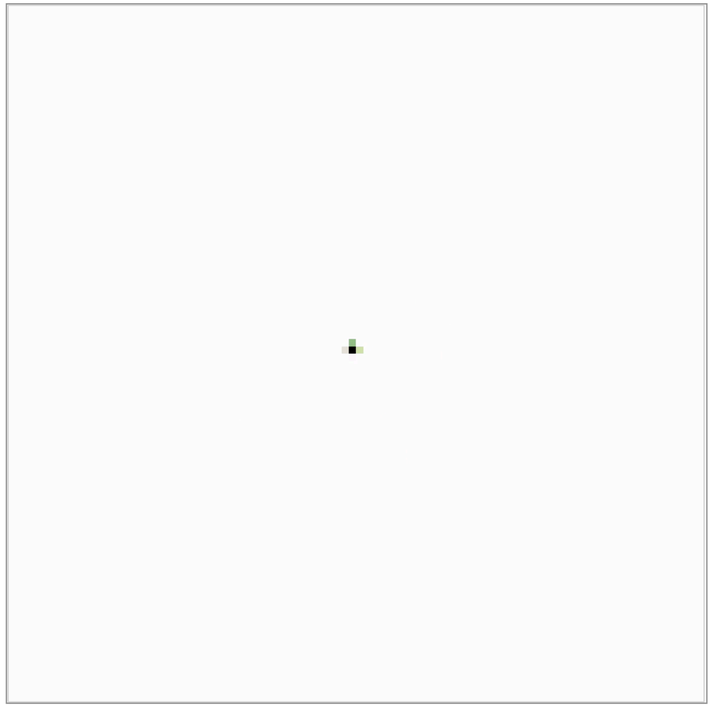




# Autómatas celulares neuronales como generadores de patrones específicos

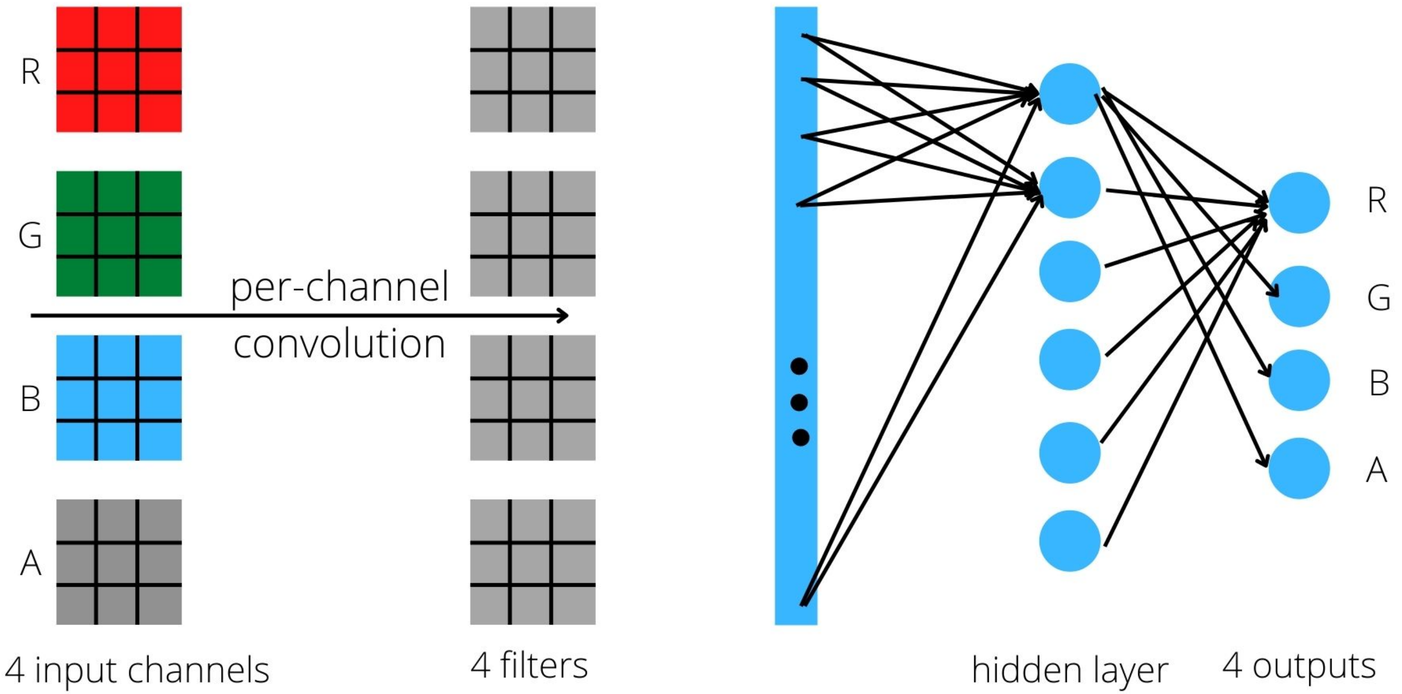


"Self-Organising Textures", Distill, 2021.

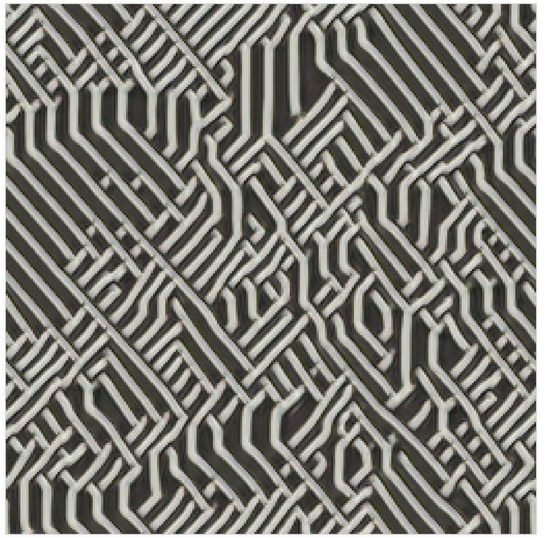


“Growing Neural Cellular Automata", Distill, 2020.

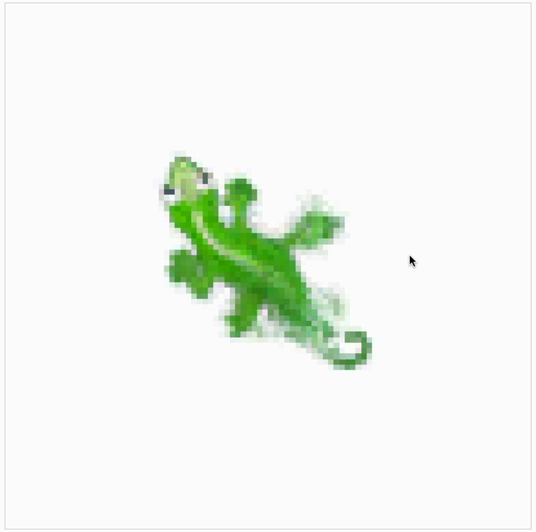
# Automatas celulares neuronales como generadores de patrones específicos



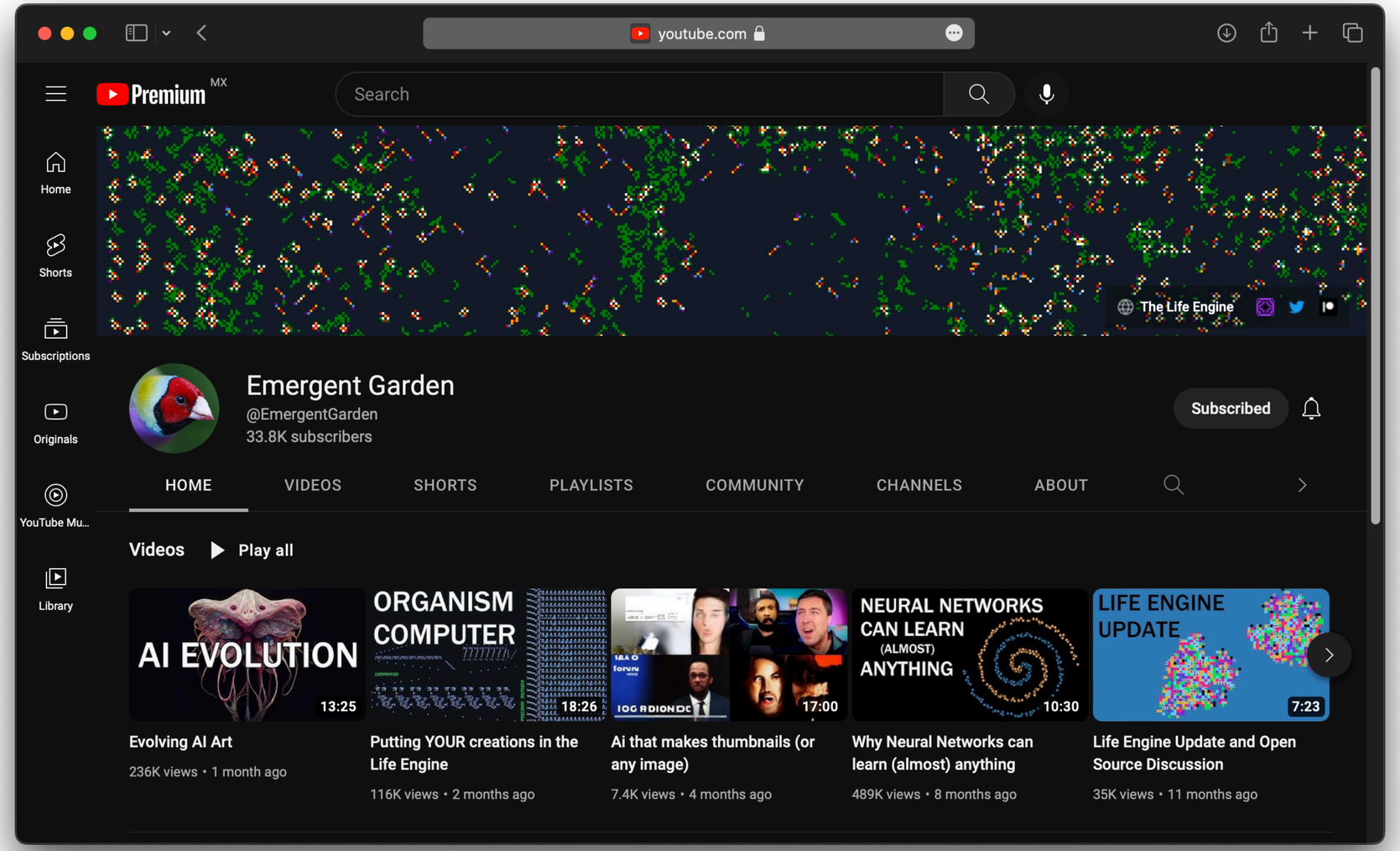

"Fun with Neural Cellular Automata", Weights & Biases, 2022



"Self-Organising Textures", Distill, 2021.



"Growing Neural Cellular Automata", Distill, 2020.




https://discord.gg/PEqFp8ad

# Referencias


- Mordvintsev, et al., "Growing Neural Cellular Automata", Distill, 2020.
- Niklasson, et al., "Self-Organising Textures", Distill, 2021.
- Randazzo, et al., "Self-classifying MNIST Digits", Distill, 2020.
- https://www.youtube.com/@EmergentGarden
- https://discord.gg/PEqFp8ad
- https://wandb.ai/johnowhitaker/nca/reports/Fun-with-Neural-Cellular-Automata--VmlldzoyMDQ5Mjg0