Proyecto Final - Modelo Neuronal Estocático

Integrantes

Adrián Bedón
adrian.bedon@udla.edu.ec

José Miguel Merlo
jose.merlo.santacruz@udla.edu.ec

Andrés Moreta
angel.moreta@udla.edu.ec

Dennis Ocaña
dennis.ocana@udla.edu.ec

Xavier Ramos
xavier.ramos@udla.edu.ec

¿Qué es el modelo estocástico?

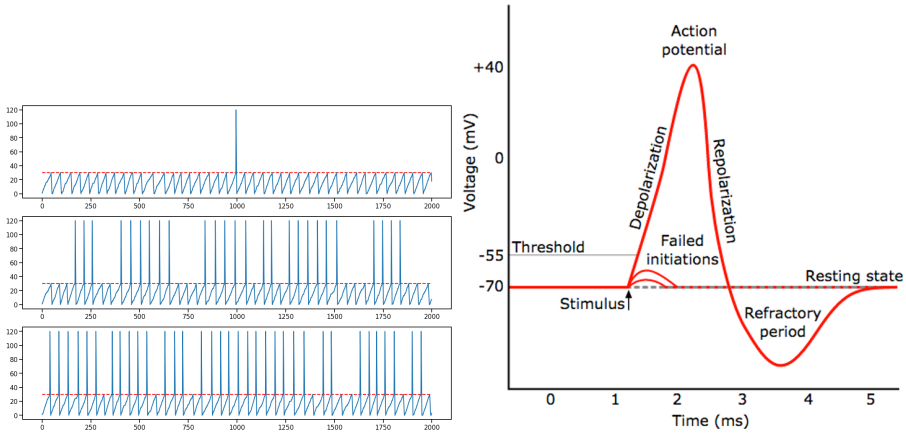
Un modelo estocástico es un sistema dinámico y cambiante a lo largo del tiempo. Este sistema esta compuesto por variables aleatorias y se analiza en términos de probabilidad.

Cada una de las variables en el proceso puede tener su propia función de distribución y pueden o no estar correlacionadas entre sí.

Un ejemplo de esto es el indice de la bolsa de valores.

¿Qué es una neurona con actividad de picos? (Neuron Model)

Al igual que una neurona real, una vez que la actividad alcanza cierto umbral existe una posibilidad en la que la actividad de la neurona se dispara. Este incremento repentino en la actividad produce un pico, lo cual le da el nombre a este tipo de neuronas.



Simulaciones

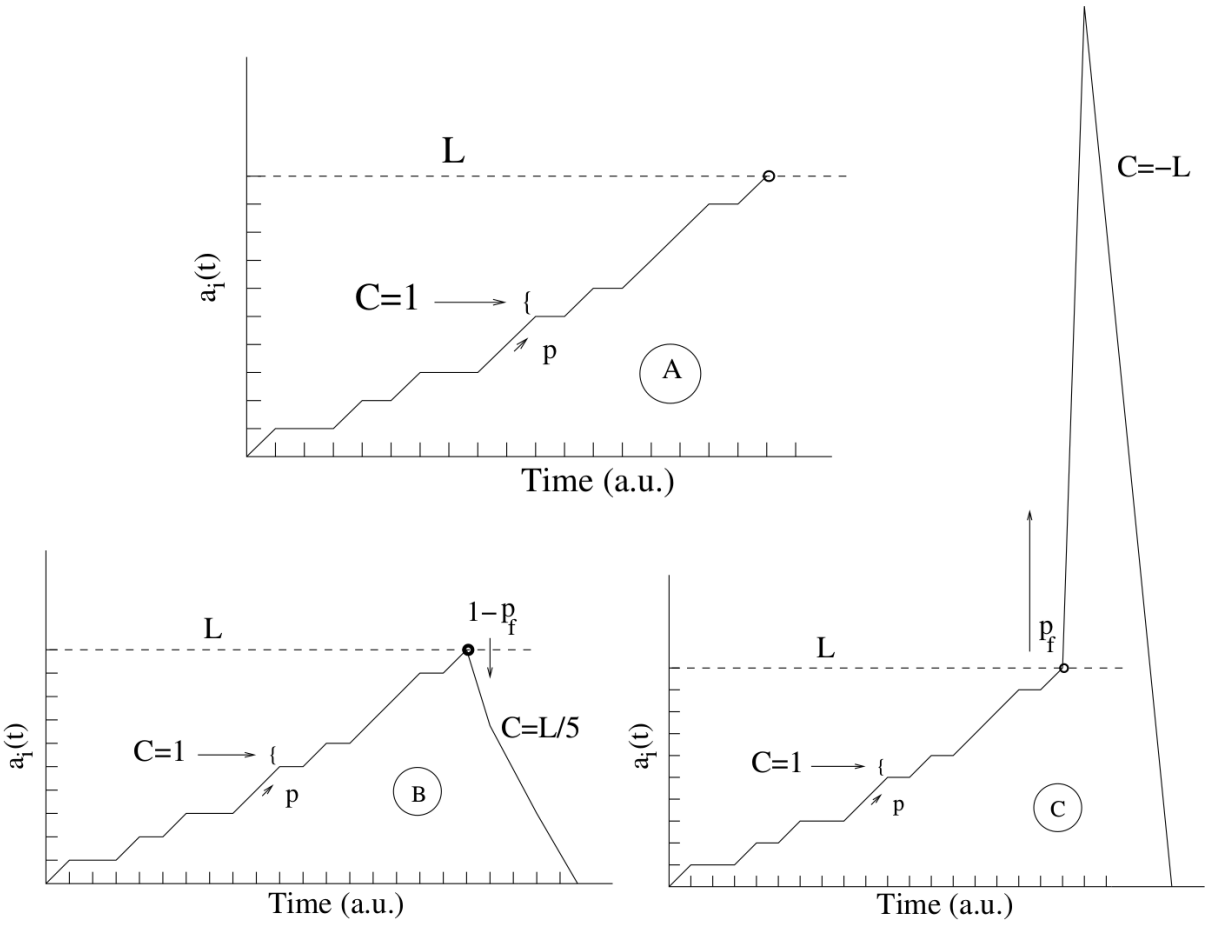
Simulación de actividad para una sola neurona

Dentro del siguiente apartado se presenta la simulación de actividad de una neurona que se encuentra condicionada por el siguiente comportamiento:

$$a(t+1) = \begin{cases} ai(t) + C, & \text{with probability } p \\ ai(t), & \text{otherwise,} \end{cases}$$

Donde p es la probabilidad de cambiar su estado interno en cada paso. Siendo así también que 1-p será la probabilidad de mantenerse en su estado actual. C es un parámetro que depende de la evolución a lo largo del tiempo, este parámetro tiene 3 estados posibles, de "escalada" (siendo que la neurona esta incrementando su actividad por debajo del umbral siendo C = 1, de "disparo" siendo que está en el umbral. Y de reset que será cuando la neurona haya disparado o haya alcanzado el umbral y deberá regresar a su punto inicial siguiento C = L / 5 con probablidad p = 1 de ocurrir.

Se adjunta una representación esquemática.



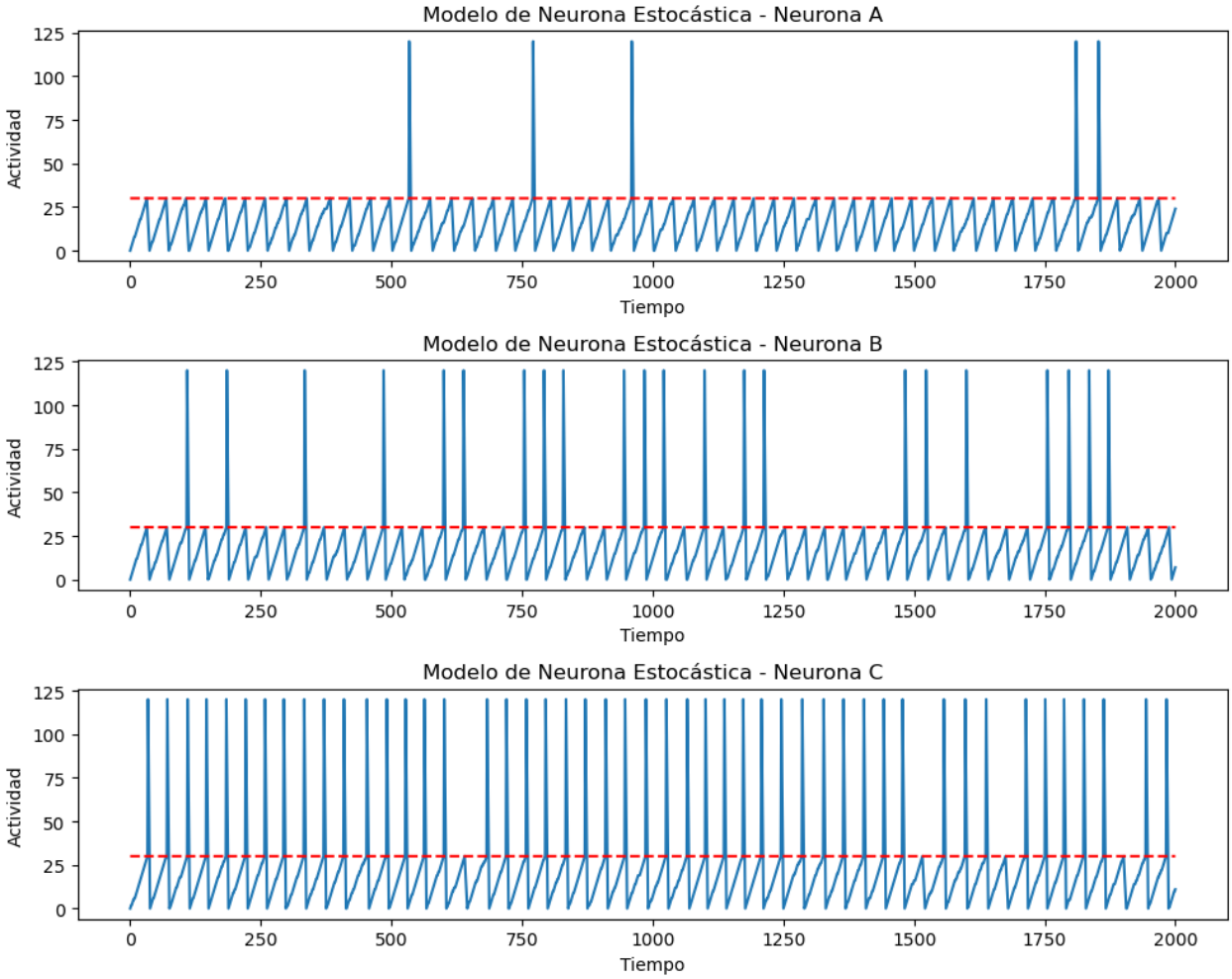
In [1]:
```
#Importación de librerías
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
from scipy.signal import argrelextrema
```

In [2]:
```
def stochastic_neuron_model(num_steps, p, pf, L):
    a = 0  # Actividad Inicial
    activity = [a]  # Lista para almacenar la actividad a lo largo del tiempo
    state = 0  # Estado inicial de la neurona (0: subumbral, 1: umbral, 2: refractario)
    count = 0  # Contador para llevar el registro del número total de cambios de estado
    count2 = 0  # Contador para llevar el registro del número de veces que se alcanzó el estado umbral

    for _ in range(num_steps):
        if a < L and state == 0:  # Si la actividad es menor que el umbral (L) y está en el estado subumbral (0)
            if np.random.random() < p:  # Si un número aleatorio es menor que p
                a += 1  # Incrementar la actividad en 1
                state = 0  # Permanecer en el estado subumbral
            else:
                a = a  # No se produce un cambio en la actividad
```

```
        else:  # Si la actividad alcanza o supera el umbral (L) o está en el estado umbral (1)
            if np.random.random() < pf and state == 0:  # Si un número aleatorio es menor que pf y está en el estado
                a += 3 * L  # Incrementar la actividad en 3 veces el umbral (L)
                state = 1  # Cambiar al estado umbral
                count += 1  # Incrementar el contador de cambios de estado
                count2 += 1  # Incrementar el contador de veces que se alcanzó el estado umbral
            else:
                if state == 1:  # Si ya está en el estado umbral (1)
                    a -= L  # Reducir la actividad en el umbral (L)
                    state = 1  # Permanecer en el estado umbral
                    if a <= 0:  # Si la actividad es menor o igual a cero
                        state = 0  # Volver al estado subumbral
                else:
                    a -= (L / 5)  # Reducir la actividad en una quinta parte del umbral (L)
                    state = 2  # Cambiar al estado refractario
                    if a <= 0:  # Si la actividad es menor o igual a cero
                        state = 0  # Volver al estado subumbral
        activity.append(a)  # Agregar la actividad actual a la lista de actividad a lo largo del tiempo
    return activity  # Devolver la lista de actividad
```

In [3]:

```
# Importar las bibliotecas necesarias
import matplotlib.pyplot as plt

# Crear una figura y tres ejes (paneles) dispuestos en una columna (3 filas, 1 columna)
fig, axs = plt.subplots(3, 1, figsize=(10, 8))

# Panel A - Caso 1
# Definir los parámetros para la primera neurona estocástica (Neurona A)
num_steps = 2000
p = 0.9
pf = 0.1
L = 30

# Calcular la actividad de la neurona utilizando el modelo estocástico con los parámetros dados
activity = stochastic_neuron_model(num_steps, p, pf, L)

# Crear una lista con los valores de tiempo correspondientes a cada punto de actividad
time = list(range(num_steps + 1))

# Graficar la actividad de la neurona en el primer panel
axs[0].plot(time, activity)

# Dibujar una línea punteada roja que representa el valor del umbral (L) a lo largo de todo el tiempo
axs[0].plot(time, [L] * len(time), 'r--', label='L')
axs[0].set_xlabel('Tiempo')
axs[0].set_ylabel('Actividad')
axs[0].set_title('Modelo de Neurona Estocástica - Neurona A')

# Panel A - Caso 2
# Definir los parámetros para la segunda neurona estocástica (Neurona B)
pf = 0.5

# Calcular la actividad de la neurona utilizando el modelo estocástico con los nuevos parámetros
activity = stochastic_neuron_model(num_steps, p, pf, L)

# Graficar la actividad de la neurona en el segundo panel
axs[1].plot(time, activity)

# Dibujar una línea punteada roja que representa el valor del umbral (L)
axs[1].plot(time, [L] * len(time), 'r--', label='L')
axs[1].set_xlabel('Tiempo')
axs[1].set_ylabel('Actividad')
axs[1].set_title('Modelo de Neurona Estocástica - Neurona B')

# Panel A - Caso 3
# Definir los parámetros para la tercera neurona estocástica (Neurona C)
pf = 0.9

# Calcular la actividad de la neurona utilizando el modelo estocástico con los nuevos parámetros
activity = stochastic_neuron_model(num_steps, p, pf, L)

# Graficar la actividad de la neurona en el tercer panel
axs[2].plot(time, activity)

# Dibujar una línea punteada roja que representa el valor del umbral (L)
axs[2].plot(time, [L] * len(time), 'r--', label='L')
axs[2].set_xlabel('Tiempo')
axs[2].set_ylabel('Actividad')
axs[2].set_title('Modelo de Neurona Estocástica - Neurona C')

# Ajustar el espacio entre los subplots para evitar superposiciones
plt.tight_layout()
```



Simulación de un modelo conectado de 3 neuronas

Las siguientes neuronas simularán una interconexión como fueran un acomplamiento eléctrico y su conductividad será dada por:

$$a_i(t) = a_i(t) + g \sum_{j=neighbors} [a_j(t-1) - a_i(t-1)]$$

Donde g es la conductividad del acoplamiento y $aj(t)$ es la actividad del vecino j. Este valor es constante entre todas las conecciones e induce una sincronización en la red de neuronas segun g

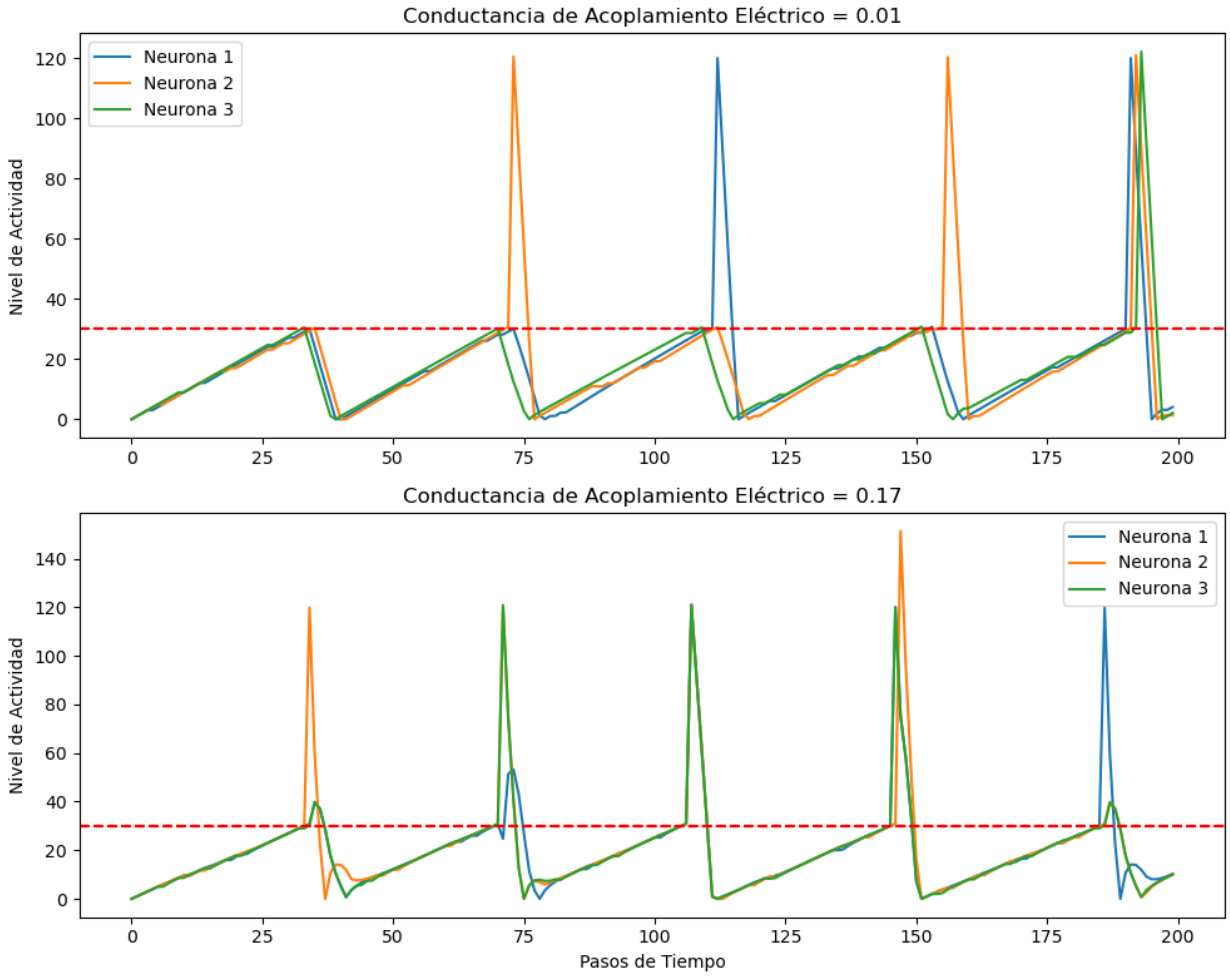
En la siguiente implementación se demostrará una red de 3 neuronas con valores g diferentes. Estos siendo g = 0.01 y g = 0.17

In [4]:
```
# Definición de la función para el cálculo de la actividad de la red neuronal de 3 neuronas
def stochastic_neuron_model_network(p, pf, L, g, num_steps, n_neurons):
    # Crear un grafo completo donde cada neurona es un nodo.
    G = nx.complete_graph(n_neurons)

    # Inicializar la matriz de actividad y estados de las neuronas.
    a = np.zeros((n_neurons, num_steps))
    states = np.zeros(n_neurons, dtype=int)

    # Simulación de la actividad de la red neuronal.
    for step in range(1, num_steps):
        for node in range(n_neurons):
            if a[node, step-1] < L and states[node] == 0:  # Estado subumbral
                if np.random.random() < p:  # Se incrementa la actividad en uno
                    a[node, step] = a[node, step-1] + 1
                    states[node] = 0
                else:
                    a[node, step] = a[node, step-1]
            else:  # Estado de umbral
                if np.random.random() < pf and states[node] == 0:  # Generación de disparo
                    a[node, step] = a[node, step-1] + (3 * L)
                    states[node] = 1
                else:
                    if states[node] == 1: # Descenso si se produce el disparo
                        a[node, step] = a[node, step-1] - L
                        if a[node, step] <= 0:
```

```
                        states[node] = 0
                    else:  # Descenso si no se produce el disparo
                        a[node, step] = a[node, step-1] - (L / 5)
                        states[node] = 2
                        if a[node, step] <= 0:
                            states[node] = 0

        for node in range(n_neurons):
            neighbors = list(G.neighbors(node))  # Obtención de los nodos vecinos
            c = g * np.sum([a[n, step-1] - a[node, step-1] for n in neighbors])  # Cálculo de acoplamiento eléctri
            a[node, step] = max(0, a[node, step] + c)

    return a
```

In [5]:

```
# Crear un grafo completo con 3 nodos (neuronas)
G = nx.complete_graph(3)

# Definir los parámetros del modelo de la neurona estocástica
p = 0.9  # Probabilidad de transición del estado interno
pf = 0.7  # Probabilidad de disparo (firing)
L = 30  # Umbral de activación
num_steps = 200  # Número de pasos de tiempo
g = [0.01, 0.17]  # Valores de conductancia de acoplamiento eléctrico

# Crear una figura con subplots para diferentes valores de conductancia
fig, axes = plt.subplots(len(g), 1, figsize=(10, 8))

# Iterar a través de los diferentes valores de conductancia
for i, _ in enumerate(g):
    # Obtener la actividad de las neuronas estocásticas utilizando el modelo definido
    a = stochastic_neuron_model_network(p, pf, L, _, num_steps, len(G))

    # Graficar la actividad de cada neurona a través del tiempo
    for element in range(len(G)):
        axes[i].plot(a[element, :], label=f'Neurona {element+1}')
        axes[i].axhline(y=L, color='r', linestyle='--')
        axes[i].set_title(f'Conductancia de Acoplamiento Eléctrico = {_}', fontsize=12)
        axes[i].set_ylabel('Nivel de Actividad', fontsize=10)
        axes[i].legend()

    # Etiquetas y título para el último subplot (el eje x)
    axes[-1].set_xlabel('Pasos de Tiempo', fontsize=10)

# Ajustar el espaciado entre subplots para que los títulos no se superpongan
plt.tight_layout()
```



Simulación de un modelo de red de 30x30 neuronas

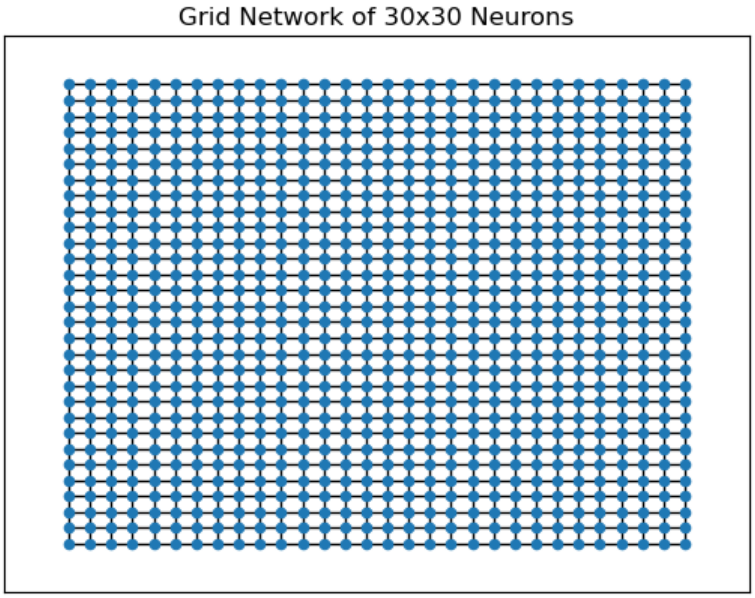
Para esta simulación se hace uso de las fórmulas definidas para el modelo de 3 neuronas, con la excepción de que ahora se evalua cada uno de los vecinos de cada neurona dentro de la red conectada en forma de grilla (malla o rejilla)

In [6]:
```
# Definición de la función para el cálculo de la actividad de la red neuronal de 30X30 neuronas
def stochastic_neuron_model_network(p, pf, L, g, num_steps, n_neurons):
    # Crear una red bidimensional (grilla) de neuronas utilizando el grafo grid_2d_graph de NetworkX.
    G = nx.grid_2d_graph(n_neurons, n_neurons)

    # Inicializar una matriz tridimensional para almacenar la actividad de cada neurona en cada paso de tiempo.
    a = np.zeros((n_neurons, n_neurons, num_steps))

    # Inicializar una matriz bidimensional para almacenar el estado de cada neurona (subumbral, umbral o refractari
    states = np.zeros((n_neurons, n_neurons), dtype=int)

    # Simulación de la actividad de la red neuronal bidimensional.
    for step in range(1, num_steps):
        for x in range(n_neurons):
            for y in range(n_neurons):
                node = (x, y)
                if a[x, y, step - 1] < L and states[x, y] == 0:
                    if np.random.random() < p:
                        a[x, y, step] = a[x, y, step - 1] + 1
                        states[x, y] = 0
                    else:
                        a[x, y, step] = a[x, y, step - 1]
                else:  # Umbral alcanzado
                    if np.random.random() < pf and states[x, y] == 0:  # Generación de disparo
                        a[x, y, step] = a[x, y, step - 1] + (3 * L)
                        states[x, y] = 1
                    else:
                        if states[x, y] == 1:  # Descenso si se produce el disparo
                            a[x, y, step] = a[x, y, step - 1] - L
                            states[x, y] = 1
                            if a[x, y, step] <= 0:
                                states[x, y] = 0
                        else:  # Descenso si no se produce el disparo
                            a[x, y, step] = a[x, y, step - 1] - (L / 5)
```

```
                            states[x, y] = 2
                            if a[x, y, step] <= 0:
                                states[x, y] = 0

        for x in range(n_neurons):
            for y in range(n_neurons):
                node = (x, y)
                # Obtener los vecinos de cada neurona en la red bidimensional.
                neighbors = list(G.neighbors(node))
                # Calcular el acoplamiento eléctrico entre la neurona y sus vecinos.
                c = g * np.sum([a[n[0], n[1], step - 1] - a[x, y, step - 1] for n in neighbors])
                # Actualizar la actividad de la neurona con el acoplamiento eléctrico calculado.
                a[x, y, step] = a[x, y, step] + c

    return a
```

In [7]:
```
# Crea una grilla de 30x30 neuronas
num_rows = 30
num_columns = 30
G = nx.grid_2d_graph(30, 30)

# Gráfico de la grilla obtenida
pos = {(x,y): (x, y) for x,y in G.nodes()}
nx.draw_networkx(G, pos=pos, with_labels=False, node_size=20)
plt.title("Grid Network of 30x30 Neurons")
```

Out[7]:
```
Text(0.5, 1.0, 'Grid Network of 30x30 Neurons')
```

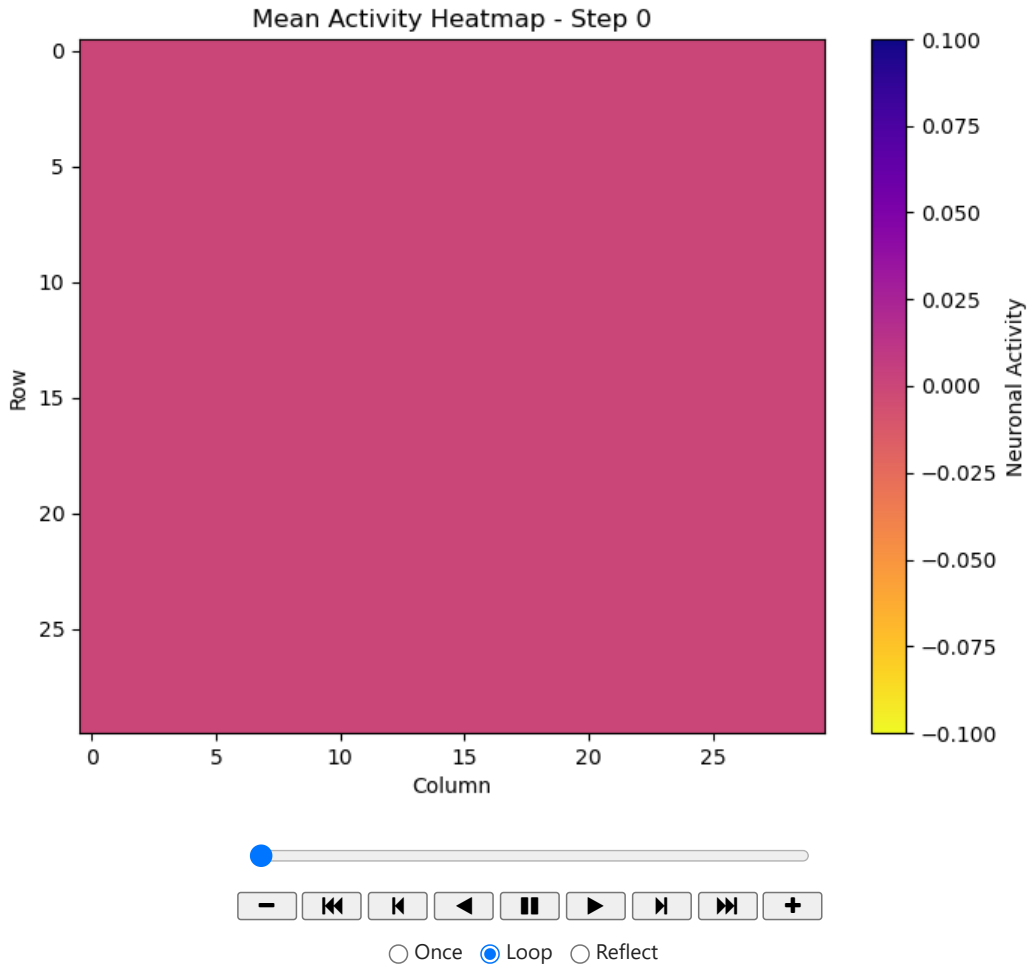


In [8]:
```
# Parámetros de la red
num_rows = 30
num_columns = 30

# Parámetros de la red neuronal estocástica
p = 0.9  # Probabilidad de incrementar la actividad
pf = 0.7  # Probabilidad de generar un pico (spike)
L = 30  # Valor de decremento de actividad
g = 0.17  # Fuerza de acoplamiento
num_steps = 500  # Número de pasos de la simulación

# Reorganizar la actividad en una matriz 2D para visualización
activity = stochastic_neuron_model_network(p, pf, L, g, num_steps, num_rows)
activity_grid = activity.T.reshape((num_steps, num_rows, num_columns))

# Crear el heatmap para el primer frame
fig, ax = plt.subplots(figsize=(8, 6))
im = ax.imshow(activity_grid[0], cmap='plasma_r', aspect='auto')
plt.colorbar(im, ax=ax, label='Neuronal Activity')
ax.set_title('Mean Activity Heatmap - Step 0')
ax.set_xlabel('Column')
ax.set_ylabel('Row')

# Función de actualización de la animación
def update(frame):
    current_activity = activity_grid[frame]
    im.set_array(current_activity)
    ax.set_title(f'Mean Activity Heatmap - Step {frame}')
    return [im]

# Generar la animación
```

```
num_frames = num_steps
anim = FuncAnimation(fig, update, frames=num_frames, interval=50, blit=True)

# Guardar la animación como un archivo GIF
anim.save('neuronal_activity_heatmap.gif', writer='Pillow')
plt.close()

# Mostrar la animación (esto solo es efectivo en Jupyter Notebook u otros entornos compatibles con animaciones en t
HTML(anim.to_jshtml())
```

```
MovieWriter Pillow unavailable; using Pillow instead.
```

Out[8]:



Nota: La presente simulación corresponde al valor g = 0.17

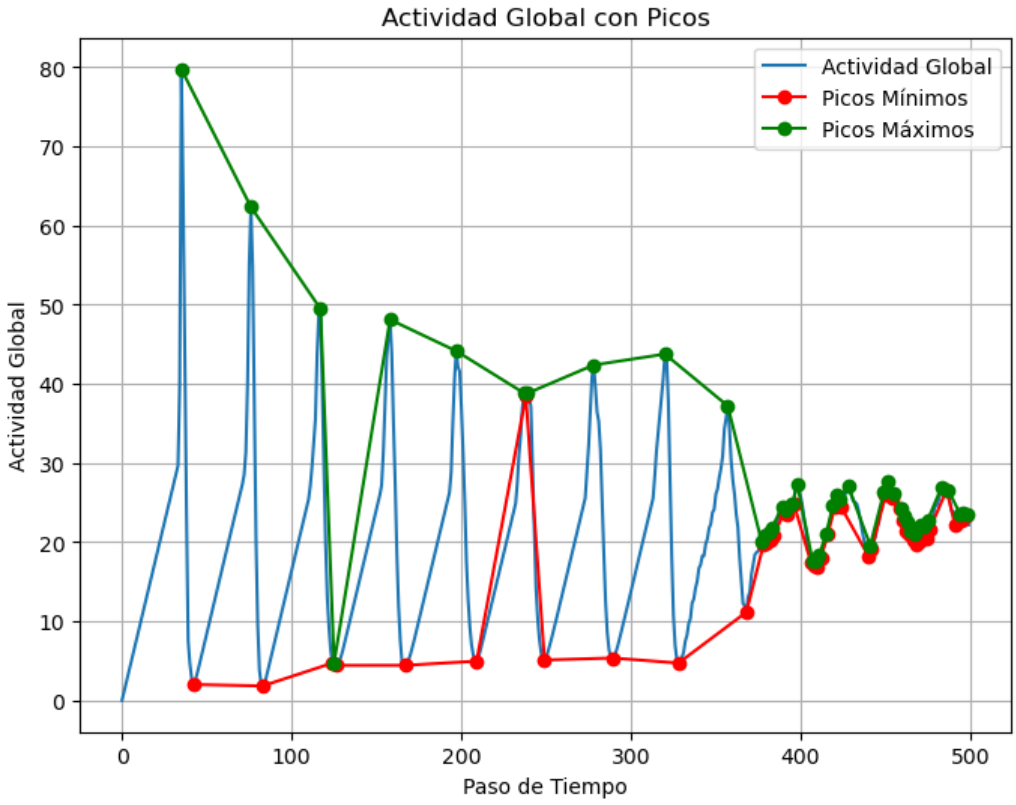
Actividad Global de la red en el tiempo

Se calculó la actividad global de la red como la media de la actividad en cada instante de tiempo, y posterior a esto se identificaron los picos mínimmos y máximos

In [9]:

```
# Calcular la actividad global promedio en cada paso de tiempo
global_activity = activity_grid.mean(axis=(1, 2))
time_steps = range(num_steps)

# Encontrar los picos locales mínimos y máximos
min_peaks = argrelextrema(global_activity, np.less)[0]
max_peaks = argrelextrema(global_activity, np.greater)[0]

# Graficar la actividad global con los picos y líneas de conexión
plt.figure(figsize=(8, 6))
plt.plot(time_steps, global_activity, label='Actividad Global')
plt.plot(min_peaks, global_activity[min_peaks], 'ro-', label='Picos Mínimos')
plt.plot(max_peaks, global_activity[max_peaks], 'go-', label='Picos Máximos')
plt.xlabel('Paso de Tiempo')
plt.ylabel('Actividad Global')
plt.title('Actividad Global con Picos')
plt.legend()
plt.grid(True)
plt.show()
```

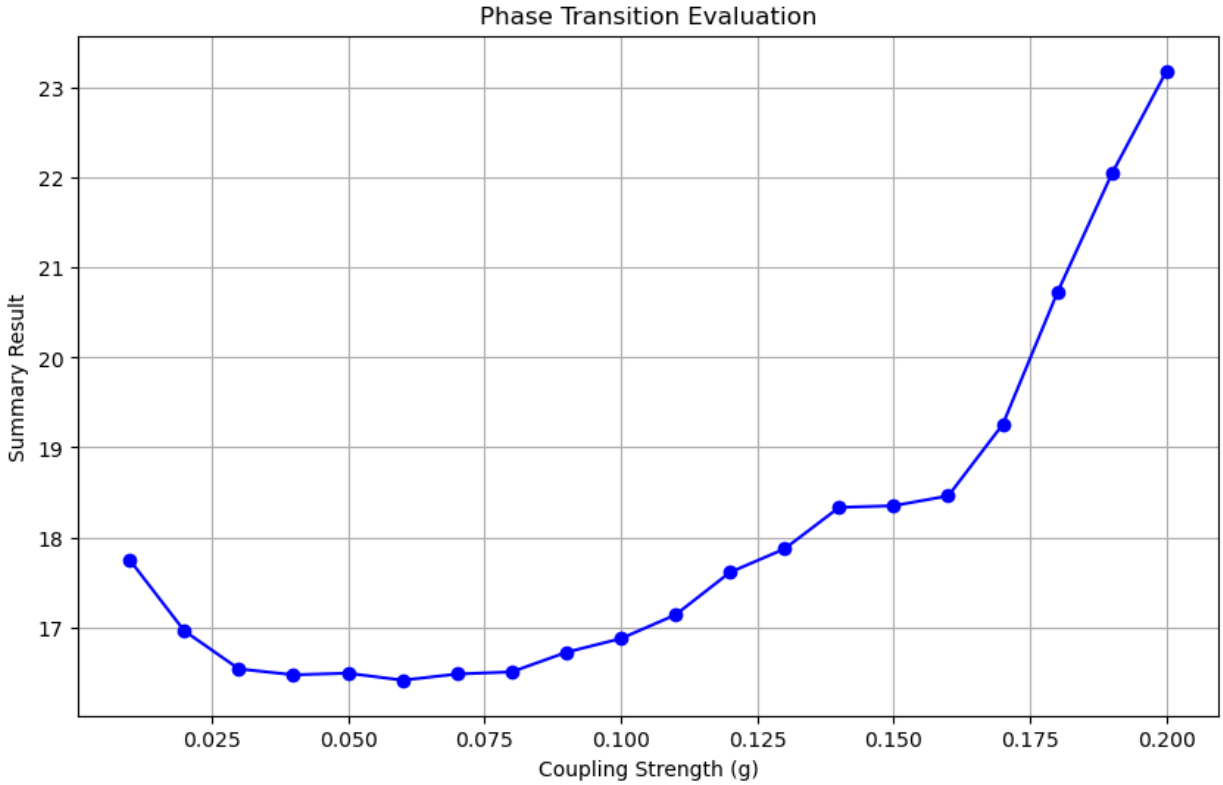
```
# Calcular las diferencias de tiempo entre picos consecutivos
time_diff_between_peaks = np.diff(max_peaks)

# Calcular la diferencia promedio entre picos
average_peak_diff = np.mean(time_diff_between_peaks)

print("Diferencia Promedio Entre Picos:", average_peak_diff)
```



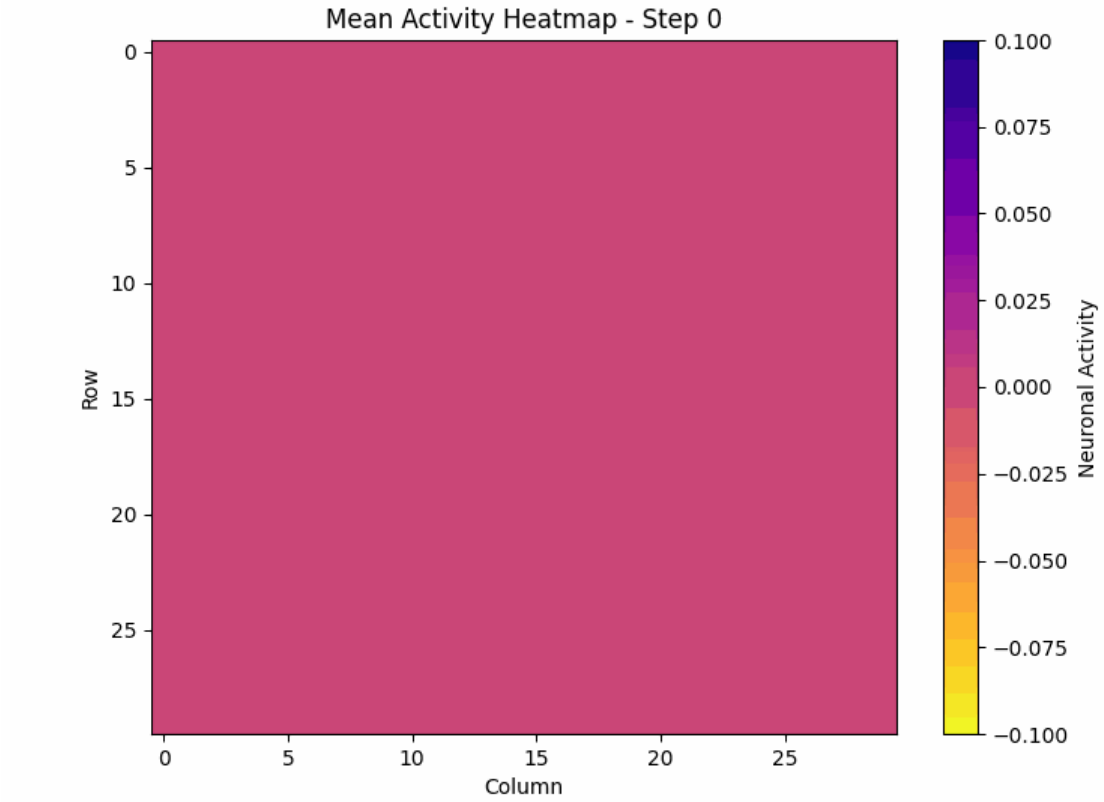

```
Diferencia Promedio Entre Picos: 10.522727272727273
```

In [10]:

```
# Parámetros de la red
num_rows = 30
num_columns = 30

# Parámetros de la red neuronal estocástica
p = 0.9  # Probabilidad de incrementar la actividad
pf = 0.7  # Probabilidad de generar un pico (spike)
L = 30  # Valor de decremento de actividad
num_steps = 500  # Número de pasos de la simulación

# Range of coupling strengths (g values) to evaluate
g_values = np.arange(0.01, 0.21, 0.01)

# List to store the results
phase_transition_results = []

for g in g_values:
    # Simulate the stochastic neuron model network for the current g value
    activity = stochastic_neuron_model_network(p, pf, L, g, num_steps, num_rows)
    activity_grid = activity.T.reshape((num_steps, num_rows, num_columns))

    # Calculate the summary result for this g value (e.g., the mean or any other relevant metric)
    # For example, you could calculate the mean activity over time and use that as the result.
    phase_transition_results.append(np.mean(activity_grid))

# Plot the results
plt.figure(figsize=(10, 6))
plt.errorbar(g_values, phase_transition_results, yerr=0.05, fmt='o-', color='b')
plt.xlabel('Coupling Strength (g)')
plt.ylabel('Summary Result')
plt.title('Phase Transition Evaluation')
plt.grid(True)
plt.show()
```



Simulación con g=0.10



Actividad inicial

Al inicio de la simulación se puede apreciar una falta de actividad en los primeros 40 pasos aproximadamente, a partir de estos comienza una actividad desde el centro de la simulación y que sirve como catalizador para una expansión de actividad hacia los bordes de la red, esta actividad dura aproximadamente 15 pasos antes que se vuelva a un estado de falta de actividad. Este ciclo se repite aproximadamente cada 40 pasos hasta la cuarta expansión de actividad.

Actividad media

Tras esta primera actividad a partir del paso 200 se registra un cambio, este se puede apreciar en la forma que la expansión inicia alrededor de la fila 10 desde el lado izquierdo de la matriz para posteriormente expandirse primero a la zona de arriba de la matriz y continuar hacia la parte de abajo de la matriz generando dos olas de actividad que convergen en el centro antes de desaparecer. Este ciclo se repite hasta los 400 pasos.

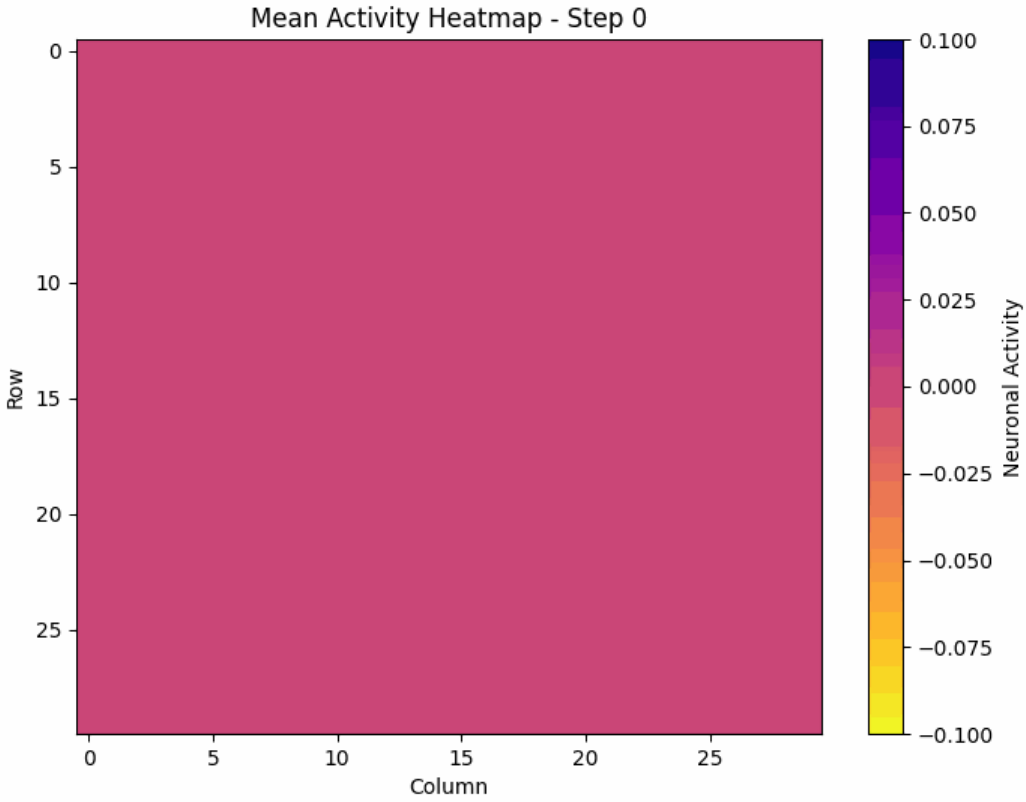
Actividad final

Finalmente se puede apreciar que el mismo comportamiento se mantiene desde la etapa anterior sin embargo, esta actividad tiene un menor nivel de intensidad, se puede apreciar que si bien el patrón de expansión continúa como en la etapa anterior es menor la actividad generada al igual que se demora un mayor número de pasos para desaparecer en aproximadamente 20, de igual manera se puede apreciar que no existe actividad constante dentro de todos los pasos realizados, esto refiriéndose a que entre los ciclos de expansión y las olas de actividad no existe una actividad menor si no que se mantiene una inactividad total de las neuronas.

Posible explicación

Principalmente se puede decir que con al ser g (sincronización de las neuronas) menor en 0.1 existe una menor posibilidad de que el disparo de una neurona afecte el comportamiento de sus vecinas y de esta manera exista una menor generación de actividad y a su vez que esta sea cíclica ya que solo una vez se haya dado una mínima cantidad de pasos permite que comiencen olas de actividad en toda la matriz.

Simulación con g=0.19



Actividad inicial

Se puede apreciar al igual que cuando g era igual a 0.1 que existe un periodo inicial de inactividad de 40 pasos, tras esto una expansión de actividad, aunque mucho menor y con olas que duran aproximadamente 10 pasos, este ciclo continuo aproximadamente hasta los 120 pasos.

Actividad media

A partir del paso 140 se puede notar un cambio total en la actividad, en primer lugar, se dé una gran ola de actividad sin embargo también se mantiene cierto nivel de actividad menor en la matriz con ciertas neuronas y sus vecinos generando

actividad, en segundo lugar, no se genera una nueva ola de actividad, en su lugar se incrementa la actividad de ciertas neuronas y sus vecinos sin generar olas de actividad, en su lugar ciertos vecindarios se activan y se pierden otros. De esta manera se rompe con el ciclo de la generación de una ola de actividad cada 40 pasos, en su lugar se ve actividad continua de los vecindarios sin que estos logren generar suficiente actividad para una ola.

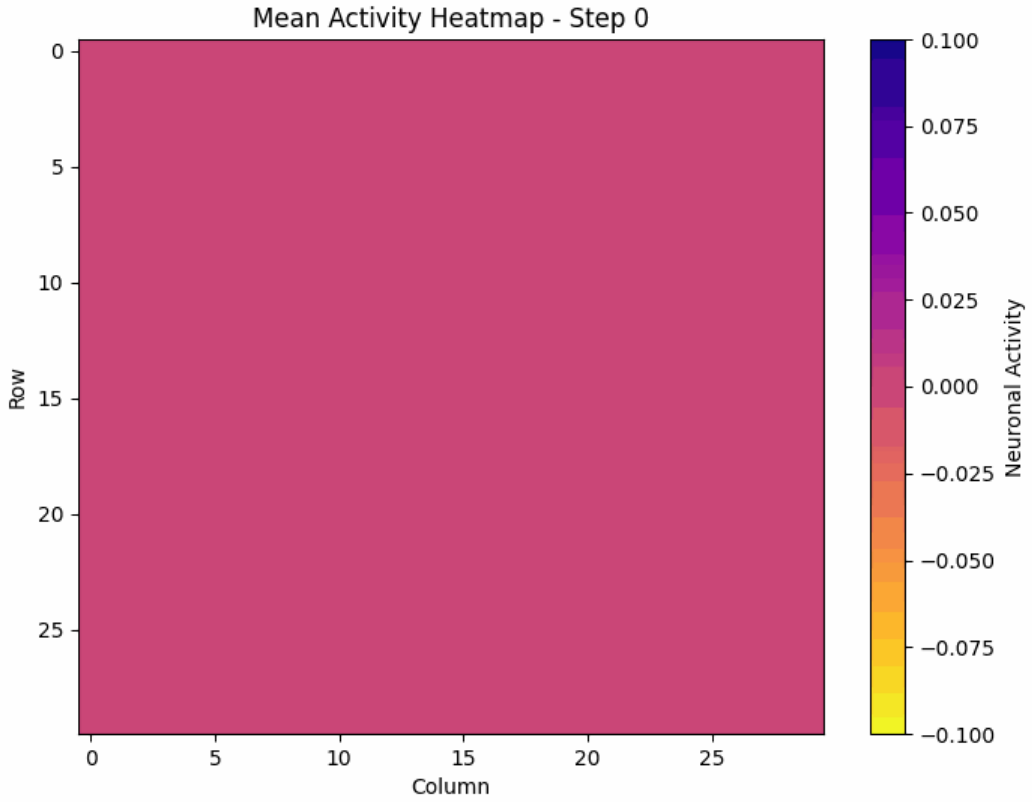
Actividad final

Se mantiene la actividad continua de ciertos vecindarios hasta el final de la simulación de nuestra matriz, de esta manera existe ciertas neuronas en sus vecindarios y se aprecia esta actividad a lo largo de la matriz en pequeña escala, sin embargo, se puede de igual manera apreciar que esta disminuye en comparación con la etapa media y aun se observa la falta de una generación de una gran ola de actividad.

Posible explicación

Se puede ver que al tener un valor de g (sincronización de las neuronas) mayor en 0.19 se genera una actividad inicial de menor escala ya que cada neurona se ve más afectada si uno de sus vecinos no se dispara, esto puede llegar a un punto en que estas ya no pueden generar nuevas olas debido a esto y por ello se localiza la actividad a ciertas zonas de la matriz que luego a su vez ganan y pierden actividad, pero no deja de existir alguna actividad por menor que sea.

Simulación con g=0.01



Actividad inicial

Durante los primeros 120 pasos se puede apreciar un ciclo de 40 pasos de inactividad y luego la aparición una ola de actividad en toda la matriz, esta actividad pasa rápidamente en 10 pasos aproximadamente, sin embargo, un cambio que se da es que queda una pequeña cantidad de actividad en un número de neuronas, 10 aproximadamente, que se mantiene entre estos ciclos y desaparece antes de la aparición de una ola.

Actividad media

A partir de esto existe un cambio notable con el resto de las simulaciones, igual existe un ciclo de aproximadamente 40 pasos, sin embargo, se da el cambio de que existe actividad de neuronas individuales y alguna con sus vecindarios en la matriz, de esta manera existe un nivel de actividad constante durante cada ciclo. Otro cambio se identifica es en las olas de actividad estas tienen una menor intensidad y se puede inferir que son más localizadas a solo zonas de la matriz en lugar de afectar a toda la red neuronal.

Actividad final

A partir de aproximadamente los 400 pasos se ve una disminución aún mayor de las olas de actividad y estas se vuelven aún más localizadas, de igual manera continua la actividad de neuronas individuales y también se ve un ligero aumento de esta actividad, sin embargo, lo más destacable es esta ligera localización de la actividad y las olas a zonas de la red

Posible explicación

Se puede inferir que este cambio se da al bajo valor de g (sincronización de las neuronas) en 0.01 este puede generar que al darse las olas de actividad ciertas neuronas sin ser casi afectadas por sus vecinos mantengan su actividad entre la realización de las olas, y conforme aumenta el número de pasos este comportamiento se ve reforzado provocando que las neuronas se vean menos afectadas por la generación de nuevas olas y en su lugar tengan una mayor independencia y por lo tanto actividad a parte del resto de la red.

Conclusiones

Una de las conclusiones llegadas es la importancia de la sincronización de las neuronas al momento de realizar las simulaciones, esto principalmente debido a que el valor de esta afecta directamente al comportamiento del vecindario de cada neurona, de esta manera una baja sincronización permite una mayor independencia y por lo tanto un cambio en la actividad de las neuronas a que tengan una mayor actividad de manera individual, mientras una mayor sincronización permite que exista una mayor coordinación de los vecindarios y por lo tanto la generación de actividad sea más determinada por la red neuronal en general antes que por comportamiento independiente. De igual manera el vecindario elegido en este caso el de Von Neumann permite que las neuronas tengan una mayor independencia de sus vecindarios ya que tienen menos neuronas que afecten su comportamiento, de esta manera aumenta aún más la importancia del valor de sincronización a que si el modelo usara un vecindario de Moore el cual tiene en cuenta una mayor cantidad de neuronas al momento de determinar un cambio en la actividad.

Adicional a esto se considera que existen mejoras que pueden ser implementadas dentro de este modelo con el fin de perfeccionar sus resultados