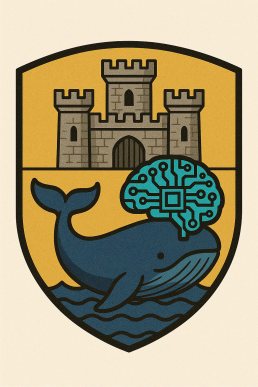
Documentación eBallena



Autor Andres Romo

1 Presentación

eBallena (exact Ballena) es un simulador de spiking neural networks (SNNs) desarrollado en el marco del proyecto MANIAS, con el objetivo de probar de manera rápida modelos de redes neuronales spiking. El simulador está inspirado en Brian2 [3], tomando este como referencia e intentando mejorar principalmente en aspectos de velocidad. A continuación, se presentan algunas características del simulador:

- El simulador implementa el modelo de neurona LIF (leaky integrate and fire) [2].
- El simulador está escrito en **C**, pero incorpora un binding de Python mediante la biblioteca `ctypes` [1]. Todo el flujo de trabajo se realiza desde Python, aunque la ejecución del simulador se lleva a cabo en C.
- El simulador está orientado a eventos: solo se realizan cálculos cuando ocurre un evento (es decir, un spike). Cuanto más escasos (sparse) sean los spikes, más rápido será el simulador.

2 Modelo neuronal LIF

El modelo LIF (leaky integrate and fire) es un modelo simplificado del funcionamiento de una neurona. En él, cada neurona mantiene un estado interno ($V_m(t)$, voltaje de su membrana). En ausencia de estímulos, el voltaje de la membrana tiende asintóticamente a un voltaje de equilibrio V_{rest}, con una velocidad determinada por la constante τ. Los estímulos ($I(t)$, spikes presinápticos representados como corriente que ingresa a la neurona) se modelan como una sumatoria de voltajes que se agregan al voltaje de la membrana. La ecuación 1 describe este comportamiento, que puede estudiarse con mayor profundidad en [2].

$$\tau \frac{d}{dt} V_m(t) = -\big(V_m(t) - V_{\text{rest}}\big) + R\, I(t), \tag{1}$$

$$s(t) = \begin{cases} 1, & \text{if } V_m(t^-) \geq V_{\text{thres}}, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{and when } s(t) = 1, \text{ then } V_m(t) \leftarrow V_{\text{reset}}, \tag{2}$$

En la práctica, $RI(t)$ se implementa como una sumatoria de pesos sinápticos $\sum_{i=1}^{N} \omega_i \cdot s_i(t)$, donde ω_i representa el peso sináptico de la neurona presináptica i y $s_i(t)$ indica la presencia o ausencia de spikes (disparos) de la neurona i. Cuando el voltaje de una neurona supera el umbral v_{thres}, esta genera un spike y su voltaje se reinicia a v_{reset}. Es común también que, durante un tiempo después de haber disparado, la neurona se encuentre inhibida y no pueda recibir spikes presinápticos ni volver a disparar. A este periodo se le conoce como periodo refractario o refractory time. Cuando una neurona hace spike, la señal puede tardar un tiempo (delay de propagación) en sumarse al voltaje de la neurona post sináptica.

Conceptos del modelo		
Concepto	**Nombre**	**Descripcion**
Spike	Spike, disparo	Evento discreto que utilizan las neuronas para comunicarse
Sinapsis	Sinpasis, conexion	Conexiones entre 2 neuronas. Define una neurona presinaptica, otra post sinaptica y un peso sinaptico ω.
Resumen de variables del modelo LIF		
Variable	**Nombre**	**Descripcion**
$V_m(t)$	Voltaje de membrana	Estado interno de la neurona.
V_{rest}	Voltaje de descanso	Volaje al cual tiende la neurona en reposo.
V_{reset}	Voltaje de reset	Volaje de la neurona despues de hacer spike.
ω_i	Peso sinaptico	Voltaje que se suma a la neurona cuando la neurona presinaptica i hace spike.
τ	Constante de tiempo	Define que tan rapido se mueve la neurona en direccion a V_{rest}
Refractory time	Refractory time	Cuanto tiempo la neurona se encuentra inactiva despues de hacer spike.
Delay	Delay sináptico	Tiempo que tarda una señal en afectar a la neurona post sináptica.

3 Structs y elementos del simulador

Acontinuacion se da una breve descripcion de los structs que intervienen en el simulador.

Structs que entran al simulador (desde python):	
struct	**descripcion**
Config	Contiene la configuración de la simulación. Incluye parametros del modelo LIF (v_threshold, v_rest, tau, refractory_time, syn_delay) asi como parámetros de la simulacion (n_neu, n_inputs, max_sim_time y coda). Una consecuencia de definir el modelo LIF en la configuración es que **todas las neuronas comparten los mismos parametros**.
InputRaw 5	Señales de inputs. Cada input contiene un tiempo exacto de spike y el indice del input que dispara.
SynapsesRaw	Sinapsis: mediante las sinapsis se define la estructura de la red. Con el mismo struct se definen las sinapsis entre inputs-reservoir y las sinapsis reservoir-reservoir.

Structs internos del simulador	
struct	**descripción**
Node	El simulador internamente utiliza listas enlazadas para gestionar los spikes y las sinapsis.
Synapses	Conteneder para almacenar y gestionar las synapses. A diferencia de Synapses raw, aqui solo se contiene una sinapsis.
Spike	Conteneder para almacenar y gestionar los spikes. A diferencia de SpikesRaw, aqui solo se contiene 1 spike.
Voltage	Conteneder para almacenar y el voltaje de las neuronas. Es el voltage de membrana en la ecuacion LIF. Tambien se indica una marca de tiempo de cuando fue la ultima vez que se actualizo el voltaje.

Structs que salen del simulador (hacia python):	
struct	**descripción**
VoltageMarker	Marcador de voltaje: cada vez que ocurre un evento (spike) y se actualiza el voltaje de una neurona, su valor queda almacenado en el marcador de voltajes.
SpikeMarker	Marcador de spikes. Cada vez que una neurona fire (genera un spike), el evento queda registrado en un marcador de spikes.

4 Logica del simulador

El simulador recibe un **struct Config**, un listado de inputs, un listado de sinapsis input-reservoir y un listado de sinapsis reservoir-reservoir. También recibe como parámetros un puntero a una lista de voltajes y un puntero a una lista de spikes, que se utilizan para almacenar los resultados de la simulación.
Lo primero que realiza el simulador es ordenar los datos para que sean accesibles de forma rápida y estructurada. Los datos llegan inicialmente como arrays, por lo que no sería eficiente recorrerlos sin antes organizarlos de manera conveniente.
Los **inputs** se ordenan como una lista de **struct Input** (ver figura 1). Cada Input contiene: **pre**, que indica la neurona o input que emite el spike (pre-sináptico); **time**, que indica el tiempo exacto del disparo; y **type**, que permite diferenciar entre sinapsis que provienen de inputs y sinapsis que provienen de otras neuronas del reservoir. Los inputs deben llegar temporalmente ordenados desde Python, ya que el simulador asume que así están. El binding de Python se encarga de ordenarlos ante de enviarlos al simulador.
Las **synapses** se organizan como un array de listas enlazadas (ver figura 2). Cada índice idx del array contiene una lista con todas las sinapsis que parten de la neurona número idx. Cada sinapsis incluye la neurona post-sináptica y un peso sináptico ω.

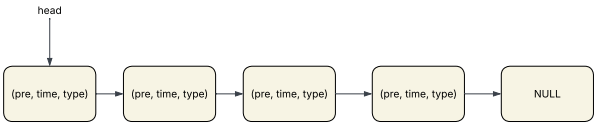


Figure 1: Lista de inputs

$$V_m(t) = v_{rest} + (V_0 - v_{rest})exp(-\frac{(t - t_{init})}{\tau}) \tag{3}$$

Una vez ordenados los inputs y las sinapsis, se genera un array de **struct Voltage** para almacenar los voltajes internos de las neuronas. Cada vez que ocurre un spike, se actualizará el voltaje de las neuronas post-sinápticas basado en la ecuación 3. Esta ecuación es la solución a la ecuacion diferencial 1 y requiere conocer el voltaje inicial de la neurona, así como el tiempo transcurrido. Por esta razón, **struct Voltage** almacena el momento en que el voltaje fue actualizado por última vez. Lógica de la ecuación 3: "Si necesito saber cuanto mide $V_m(t)$ lo calculo basandome en el ultimo voltaje conocido y cuanto tiempo ha transcurrido". También se mantiene un array de

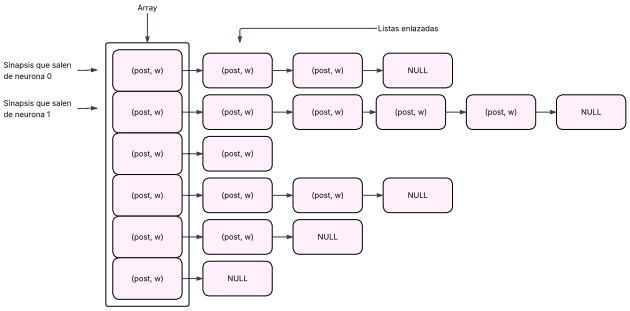


Figure 2: Array de listas de sinpases

floats llamado **refractory** para almacenar el tiempo de la última vez que la neurona generó un spike. Con esto se calcula si la neurona se encuentra en estado refractario o no.

```
// Reordenar datos de entrada
list_spikes <= lista con todos los spikes
syn_in <= construir sinpases input-reservoir
syn_re <= construir sinapses reservoir-reservoir

// incializacion
neu_state <= array de struct Voltages // el estado de la neurona es su voltaje
refractory <= array de floats

// Inicializar elementos a retornar
voltages = []
spikes   = []

// bucle principal
while list_spikes is not empy do
    spike <= get item from list_spikes

    // Se busca en el array de sinapsis la lista de neuronas post synapticas
    if spike.type = INPUT_SPIKE do
        synapses <= syn_in[ spike.pre ]
    else if spike.type = NEURON_SPIKE do
        synapses <= syn_re[ spike.pre ]
    end if

    // leak
    for synapse in synapses do
        if synapses.post is not refractory do
            neu_state[ synapses.post ] <= actualize voltage and time with
                lif_equation and spike.time
        end if
    end for

    // integrate
    to_fire_list <= []  // lista de neuronas a disparar
    for synapse in synapses do
        if synapses.post is not refractory do
            neu_state[ synases.post ] += synapse.w
            if neu_state[ synapses.post ] >= v_thres do
                add synapses.post to to_fire_list()
                add (new voltaje, spike.time, neuron_idx) to voltages
            end if
        end if
    end for

    // REC
    add (new voltaje, spike.time, neuron_idx) to voltages

    // fire
    for spike_neuron in to_fire list do
        neu_state[spike_neuron] <= set voltage to zero()
        refactory[spike_neuron] <= set last spike to spike.time
        add (spike_neuron, spike.time) to spikes      // este es el retorno de la
            funcion
        add (spike_neuron, spike.time) to list_spikes // estos son los eventos a
            simular
    end if
end while

// coda
medir_voltaje_final()

// Exit
free_memory()
save_resturns()
```

El bucle principal procesa eventos (los spikes). Cuando ocurre un spike, todas las neuronas implicadas (neuronas post-sinápticas) actualizan su voltaje siguiendo la ecuación 3, suman a su voltaje el valor del peso sináptico correspondiente y generan un spike si corresponde. Es importante notar que, si una neurona no recibe ningún input, no actualizará su voltaje. Al final de la simulación, hay una **coda** que mide el voltaje final de las neuronas. Si una neurona genera un spike, el evento debe agregarse a la lista enlazada de spikes. La lista de spikes se encuentra ordenada por tiempo de spike, y el nuevo evento debe respetar ese orden (debe insertarse de manera ordenada). De todos modos, la mayoría de las veces un spike nuevo será el primer elemento de la lista. Los spikes se implementaron con delay sináptico, el cual se recomienda que sea pequeño (menos de $1ms$).

Otros aspectos a considerar:

- La mayoria de estructuras de datos utilizadas son listas enlazadas. Esta estructura es flexible y permite agregar insertar datos sin tener que realocar. Ademas es facil de usar.
- Todas las unidades de voltaje se expresan en mV (v_rest, v_threshold, pesos sinapticos, etc...), mientras que todas las unidades de tiempo estan en ms (constante τ, refractory_time, max_sim_sim).

5 Binding de python

El binding de python permite llamar al simulador desde python. Esta escrito en ctypes [1]. ctypes permite llamar funciones escritas que hayan sido compilidas en bibliotecas compartidas. Para compilar el archivo ceBallena.c hay usar el comando

```
gcc -shared -o ceBallena.so -fPIC ceBallena.c
```

En archivo eBallena.py se encuentra el codigo de python. Ahi se replican las estructuras de datos de ceBallena.c y se ofrecen funciones utiles para recuerar y presentar los resultados de la simulacion. Por favor conservar el programa ceBallena.so en la carpeta eBallena, ya que ahí es donde la libreria va a buscarlo.

Para usar eBallena desde python, lo primero es configurar la simulación. El siguiente código crea una configuración con dos inputs, dos neuronas, un tau de 10, ms y un refractory time de 1, ms. También se define la topología de la figura 3, donde las sinapsis se definen en listas de tuplas (neurona presináptica, neurona post-sináptica, peso sináptico). Los inputs se definen como una lista de tuplas (tiempo del spike, índice del input). En el ejemplo siguiente, el input 0 dispara en 10, ms, 15, ms, 20, ms, 25, ms, 30, ms y 80, ms, mientras que el input 1 dispara solo en 12, ms.

```
from eBallena import eBallena

config   = eBallena.createConfig(n_inputs=2,n_neu=2,tau=10,max_sim_time=100,
    refractory_time=1)
instance = [(10,0),(15,0),(20,0),(25,0),(30,0),(80,0),(12,1)]
syn_in   = [(0,0,60),(1,1,10)]
syn_re   = [(0,1,150)]
```

Luego solo hace falta invocar a la funcion simulate, la cual recible la configuracion, los inputs y las sinpasis.

```
res = eBallena.simulate(config, instance, syn_in, syn_re, mode='VOLTAGES')
time, voltages = simulacionBrian2Voltages(config, instance, syn_in, syn_re)
```

Los modos de simulacion implementados son 3:

- VOLTAGE: Devuelve la curva de voltajes a lo largo de la simulacion.
- SPIKES: Devuelve una lista de tuplas con los tiempos exactos de spikes.
- STATE: Devuelve la cantidad de spikes por cada neurona (state es el estado del reservorio en una liquid state machine).

En los tres casos, la simulación es la misma; solo cambia qué datos de respuesta se muestran y cuáles se descartan. **simulacionBrian2Voltages()** es una función auxiliar para simular una topología en el simulador Brian2, con el objetivo de comparar ambas implementaciones. Los resultados del código anterior se muestran en la figura 4. En las dos imágenes de la izquierda se presentan los resultados de eBallena y Brian2 superpuestos, mientras que en las imágenes de la derecha se encuentran desfazadas en el tiempo a propósito para evidenciar que ambas curvas son equivalentes.

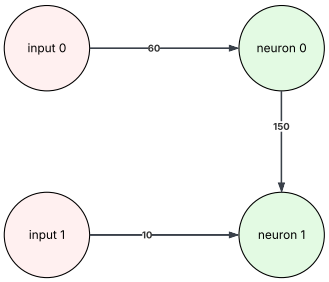


Figure 3: Red sencilla. Topologia de ejemplo.

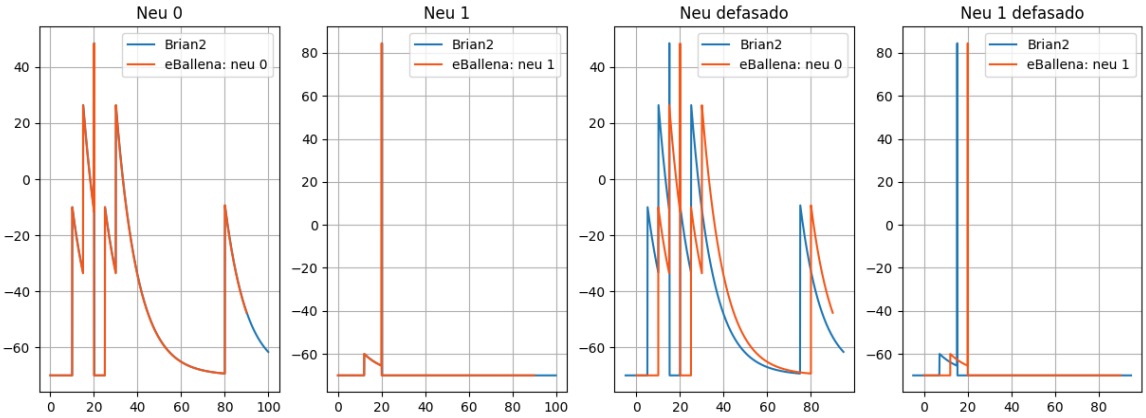


Figure 4: Voltaje de las neuronas en el codigo de ejemplo.

6 Contacto

Cualquier duda o comentario escribir a ai.romo.moena@upm.es

References

[1] Python Software Foundation. ctypes una biblioteca de funciones foráneas para python. URL: https://docs.python.org/es/3.10/library/ctypes.html.

[2] Wulfram Gerstner, Werner Kistler, Richard Naud, and Liam Paninski. *Introduction: Neurons and Mathematics*, chapter Introduction: Neurons and Mathematics. was published with Cambridge University Press, 2014. URL: https://neuronaldynamics.epfl.ch/online/Ch1.html.

[3] Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *eLife*, 8, Aug 2019. URL: https://brian2.readthedocs.io/en/stable/, doi:10.7554/elife.47314.