

# Brian2GeNN introductory tutorial

# Content

- Introduction
- Brian2, GeNN and Brian2GeNN
- Practical tutorial Brian2GeNN
  - Basics: Neurons, connectivity, synapses, monitors, etc
  - Simulations: Inputs, weights updates, etc
- Hands on
  - Implement LIF neuron with adaptation
  - Implement a type of STDP and balance inputs of a neuron
- Complex models
  - Example complex synapses
  - Not supported features Brian2GeNN
  - Multicompartment neurons

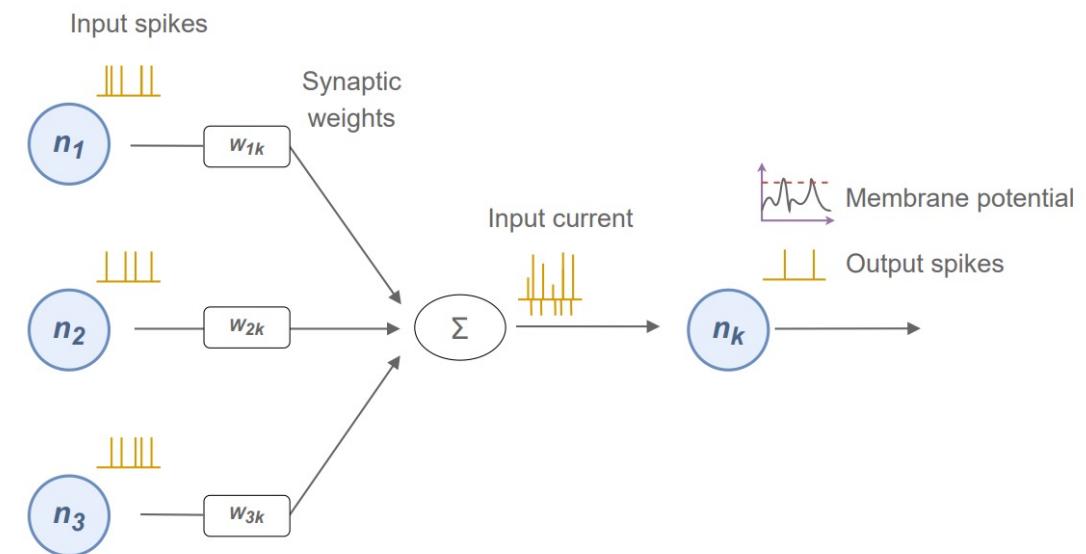
# Introduction

# Introduction

- Spiking Neural Networks (SNNs)
  - Neurons communicate through spikes
  - Membrane potential as an internal state
    - Incoming spikes modifies its value
    - If threshold is reached a spike is generated
  - Membrane's dynamics depend on the used model
    - E.g. Leaky Integrate and Fire (LIF)

$$\tau \frac{\partial V}{\partial t} = V_r - V + I, \quad \text{if } V > V_{th} \rightarrow \text{spike}$$

- Decoding spikes
  - Spiking rate (count # of spikes per second)
  - Readout weights



# Introduction

- Current ML frameworks are not entirely suitable for SNNs
  - Spiking neurons as a dynamical systems
  - SNNs are event-based systems
  - Optimized for applying backpropagation
    - SNNs sometimes need other learning rules
- SNNs simulators
  - Focus on comp. neuroscience
    - More flexibility and control
  - Focus on ML
    - More efficiency



GeNN

BRIAN

BindsNET

Nengo

# Brian2, GeNN and Brian2Genn

# Brian2

- Python-based SNN simulator
- High level and natural descriptions
  - C++ code generation from strings
  - Easy implementation of custom models
- High flexibility
  - Automatic integration of diff. equations
  - Physical units support
    - But not mandatory
  - Events
- Only supports CPU execution

```
1  from brian2 import *
2  start_scope()
3
4  # Parameters
5  vr = -70*mV
6  vth = -50*mV
7  tau = 20*ms
8
9  # Neurons population
10 eqs='''  
11 dv/dt = (vr-v+I)/tau: volt (unless refractory)  
12 I : volt  
13 '''  
14 G = NeuronGroup(10, eqs, threshold='v>vth', reset='v=vr',
15                      refractory=1*ms, method='euler')  
16 G.I = 'rand()*volt'  
17 G.v = 'vth + rand()*(vr-vth)'
18
19 # Connectivity and synapses
20 S = Synapses(G, G, on_pre='v_post += 1*mV')
21 S.connect(j='k for k in range(i-3, i+4) if i!=k',
22           skip_if_invalid=True)
23
24 # Run simulation
25 run(100*ms)
```

$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

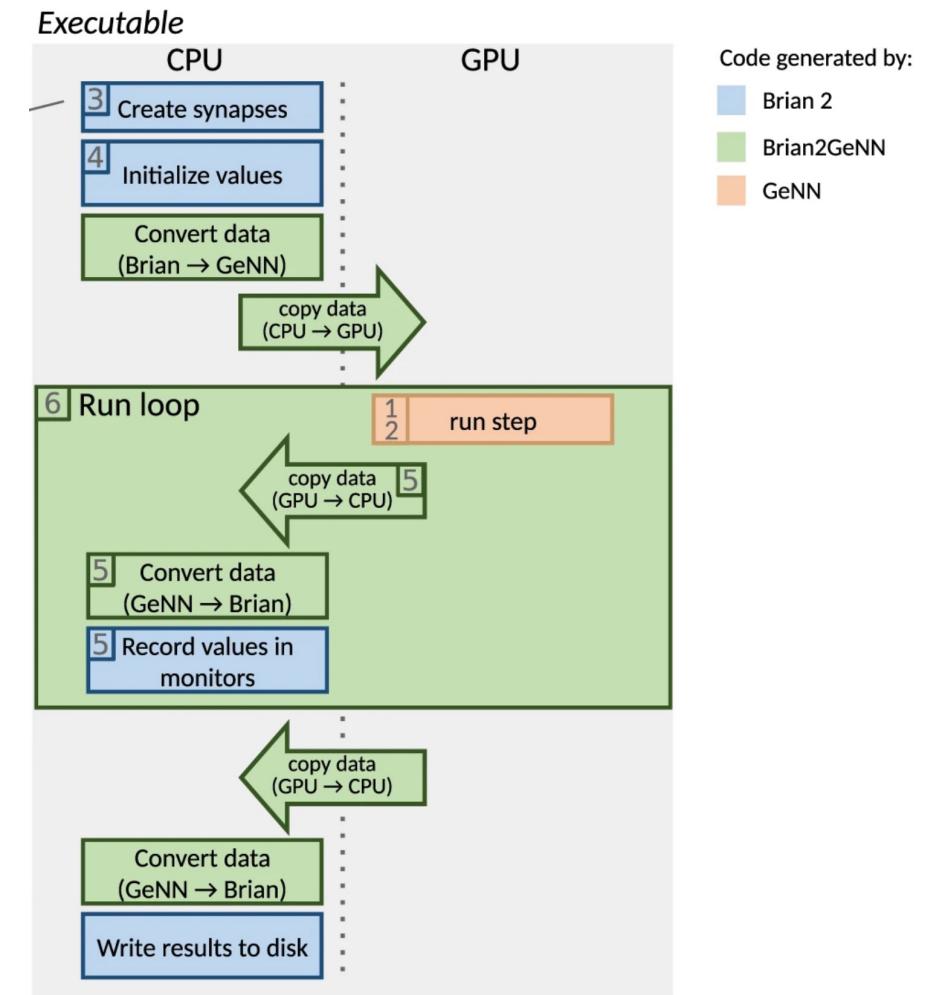
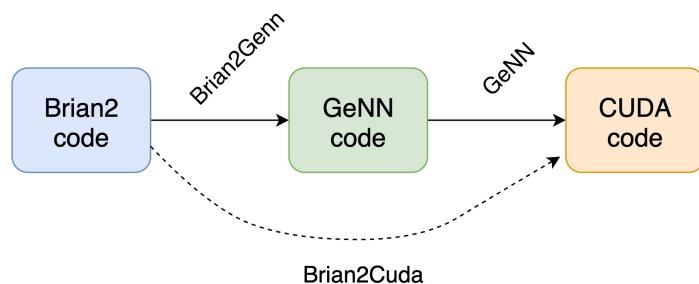
# GeNN

- C++-based SNN simulator
- Object oriented descriptions
  - CUDA code generated from them
  - Many implemented models
- Custom models as extensions of existing templates
  - Plug them in the simulation
  - Hard to debug
- Supports GPU execution

```
1 // Model definition file
2 #include "modelSpec.h"
3
4 // Connectivity
5 class Ring : public InitSparseConnectivitySnippet::Base
6 {
7 public:
8     DECLARE_SNIPPET(Ring, 0);
9     SET_ROW_BUILD_CODE(
10         $(addSynapse, $(id_pre) + 1) % $(num_post))\n"
11         "$(endRow);\n");
12     SET_MAX_ROW_LENGTH();
13 };
14 IMPLEMENT_SNIPPET(Ring);
15
16
17 // LIF neuron
18 class LeakyIntegrator : public NeuronModels::Base
19 {
20 public:
21     DECLARE_MODEL(LeakyIntegrator, 4, 1);
22
23     SET_SIM_CODE("$(V) += $(Vr)-$(V)+$(I))*(DT/$(tau));");
24     SET_THRESHOLD_CONDITION_CODE("$(V) >= $(Vth)");
25     SET_RESET_CODE("$(V) = $(Vth);");
26
27     SET_PARAM_NAMES({"tau"}, {"Vr"}, {"Vth"}, {"I"});
28
29     SET_VARS({{"V", "scalar"}});
30 };
31 IMPLEMENT_MODEL(LeakyIntegrator);
32
33
34
35 // Neurons with connectivity and synapses
36 void modelDefinition(ModelSpec *model)
37 {
38     // definition of tenHHRing
39     model->setT(0.1);
40     model->setName("MyModel");
41
42     // Parameters
43     double tau = 20.0;
44     double Vr = -70.0;
45     double Vth = -50.0;
46
47     // Random numbers generator for V and I (min, max)
48     InitVarSnippet::Uniform::ParamValues randV(0.0, 1.0);
49     InitVarSnippet::Uniform::ParamValues randI(-70.0, -50.0);
50
51     // Input layer
52     model->addNeuronPopulation<LeakyIntegrator>("Pop1", 10,
53                                                 LeakyIntegrator::ParamValues(tau), // tau
54                                                 Vr, // Rest value
55                                                 Vth, // Threshold
56                                                 LeakyIntegrator::VarValues(initVar<InitVarSnippet::Uniform>(randI), // I
57                                                 LeakyIntegrator::VarValues(initVar<InitVarSnippet::Uniform>(randV))); // V
58
59     WeightUpdateModels::StaticPulse::VarValues s_ini(
60         -0.2); // 0 - tau_S: decay time constant for S [ms]
61
62     PostsynapticModels::ExpCond::ParamValues ps_p(
63         1.0, // 0 - tau_S: decay time constant for S [ms]
64         -80.0); // 1 - Erev: Reversal potential
65
66     model->addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::ExpCond>(
67         "Pop1self", SynapseMatrixType::SPARSE_GLOBALG, 100,
68         "Pop1", "Pop1",
69         {}, s_ini,
70         ps_p, {},
71         initConnectivity<Ring>());
72
73 }
```

# Brian2GeNN

- Brian interface with GPU support
- Translates Brian code into GeNN code



# Practical tutorial

# Brian2GeNN

# Basics - Neurons

- Equations
- Threshold
- Reset
- Refractoriness
- Integration
- Subgroups

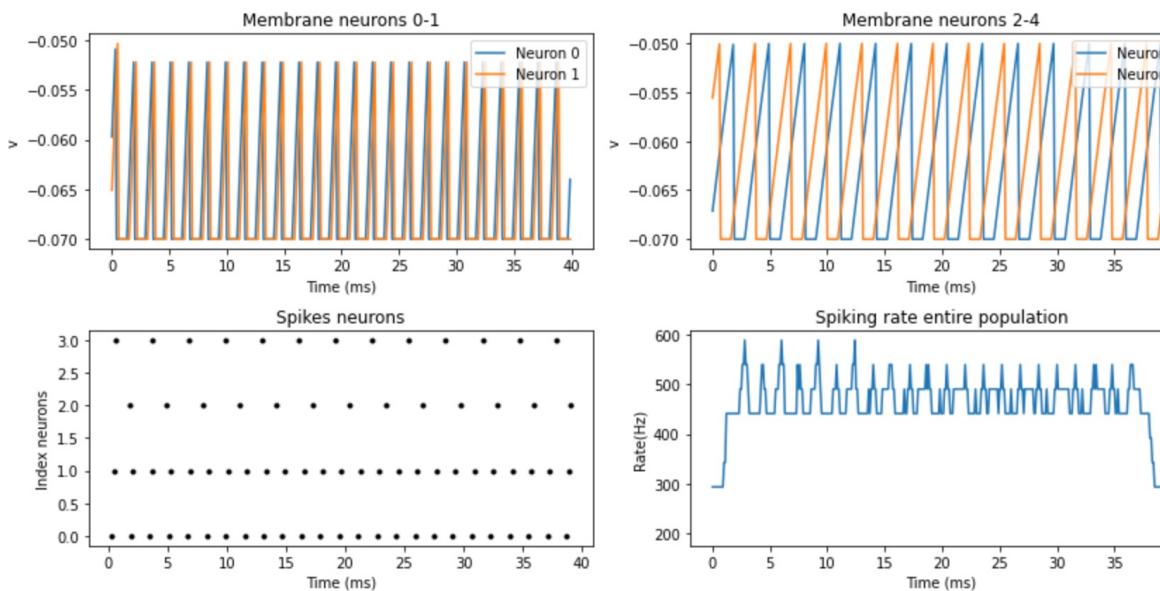
$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

```
3 # Restart Brian objects
4 start_scope()
5
6 # Parameters
7 vr = -70*mV
8 vth = -50*mV
9 tau = 20*ms
10 n_neurons = 4
11 refractory = 1*ms
12
13 # Neurons population
14 eqs='''
15 dv/dt = (vr - v + I)/tau : volt (unless refractory)
16 I : volt
17 '''
18 reset=''
19 v=vr
20
21 G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,
22                   refractory=refractory, method='euler')
23
24 # Subgroups
25 G1 = G[:int(n_neurons/2)]
26 G2 = G[int(n_neurons/2):]
27
28 # Initializations
29 G.v = 'vth + rand()*(vr-vth)' # All neurons
30 G1.I = '0.6*volt' # First group
31 G2.I = '0.2*volt' # Second group
32
33 # Run simulation
34 simulation_time=40
35 run(simulation_time*ms)
```

# Basics - Monitors

- Record variables
- Record spikes
- Record spiking rate

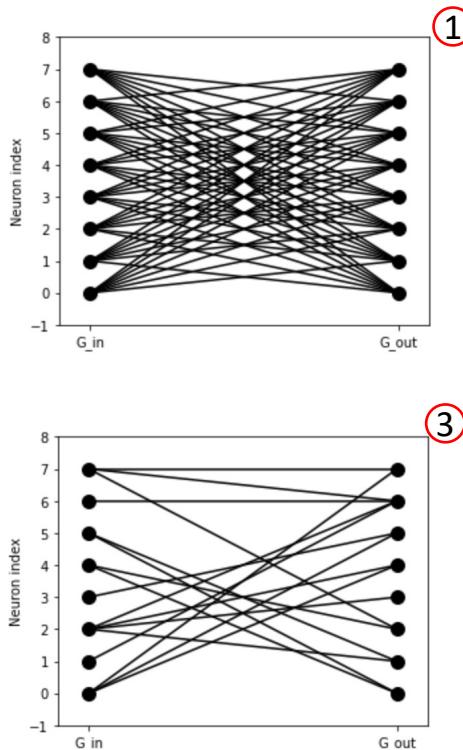
$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$



```
20 # Neurons population
21 eqs=''
22 dv/dt = (vr - v + I)/tau : volt (unless refractory)
23 I : volt
24 ''
25 reset=''
26 v=vr
27 ''
28 G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,
29                   refractory=refractory, method='euler')
30
31 # Subgroups
32 G1 = G[:int(n_neurons/2)]
33 G2 = G[int(n_neurons/2):]
34
35 # Initializations
36 G.v = 'vth + rand()*(vr-vth)' # All neurons
37 G1.I = '0.6*volt' # First group
38 G2.I = '0.2*volt' # Second group
39
40 # Monitors
41 monitor_v = StateMonitor(G, 'v', record=True)
42 monitor_spk = SpikeMonitor(G)
43 monitor_rate = PopulationRateMonitor(G)
44
45 # Run simulation
46 run(40*ms)
47
48 # Visualizations
49 plt.figure(figsize=(12,6))
50 plt.subplot(221)
51 for n in range(int(len(monitor_v.v)/2)):
52     plot(monitor_v.t/ms, monitor_v.v[n], label='Neuron '+str(n))
53 xlabel('Time (ms)')
54 ylabel('v')
55 plt.title('Membrane neurons 0-'+str(int(len(monitor_v.v)/2)-1))
56 legend(loc='upper right')
57
58 plt.subplot(222)
59 for n in range(int(len(monitor_v.v)/2)):
60     plot(monitor_v.t/ms, monitor_v.v[int(len(monitor_v.v)/2)+n],
61          label='Neuron '+str(int(len(monitor_v.v)/2)+n))
62 xlabel('Time (ms)')
63 ylabel('v')
64 plt.title('Membrane neurons '+str(int(len(monitor_v.v)/2))+ '-' +str(len(monitor_v.v)))
65 legend(loc='upper right')
66
67 plt.subplot(223)
68 plot(monitor_spk.t/ms, monitor_spk.i, '.k')
69 xlabel('Time (ms)')
70 ylabel('Index neurons')
71 plt.title('Spikes neurons')
```

# Basics - Synapses

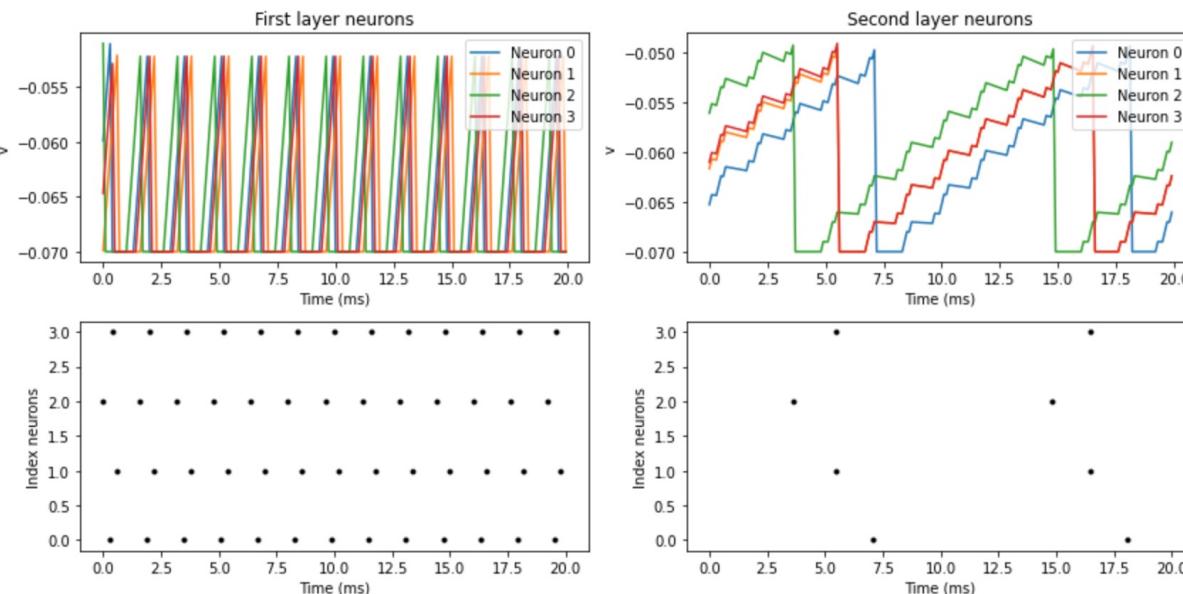
- Connectivity


$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

```
14 # Neurons population
15 eqs=''
16 dv/dt = (vr - v + I)/tau : volt (unless refractory)
17 I : volt
18 ...
19 reset=''''
20 v=vr
21 ...
22 G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,
23 refractory=refractory, method='euler')
24
25 # Subgroups
26 G_in = G[:int(n_neurons/2)]
27 G_out = G[int(n_neurons/2):]
28
29 # Connectivity (FULL except same idx - 100% Chance)
30 S = Synapses(G_in, G_out)
31 S.connect(condition='j!=i', p=1.0) 1
32 visualise_connectivity(S)
33
34 # Connectivity (FULL - 100% Chance)
35 S = Synapses(G_in, G_out)
36 S.connect(p=1.0) 2
37 visualise_connectivity(S)
38
39 # Connectivity (FULL - 30% chance)
40 S = Synapses(G_in, G_out)
41 S.connect(p=0.3) 3
42 visualise_connectivity(S)
43
44 # Connectivity (Closest neurons - convolution)
45 S = Synapses(G_in, G_out)
46 S.connect(j='k for k in range(i-3, i+4)', skip_if_invalid=True) 4
47 visualise_connectivity(S)
```

# Basics - Synapses

- Post spike effect
  - Current injection (Increase  $v$  or  $I$ )

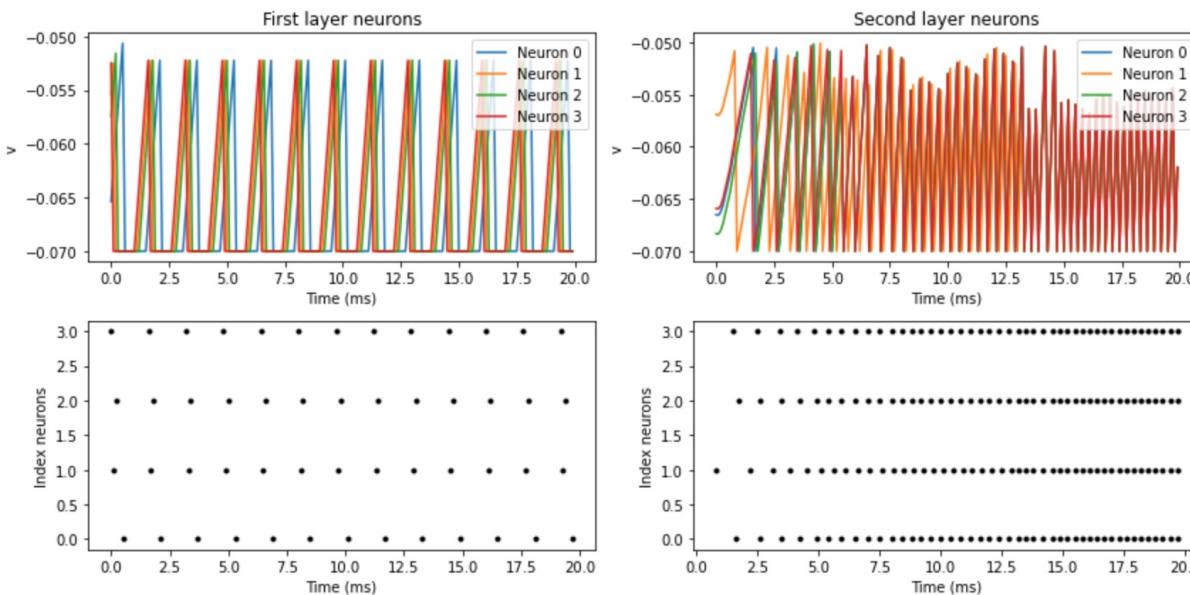

$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

```
13 # Neurons population
14 eqs='''  
15 dv/dt = (vr - v + I)/tau : volt (unless refractory)  
16 I : volt  
17 '''  
18 reset='''  
19 v=vr  
20 '''  
21 G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,  
22 refractory=refractory, method='euler')  
23  
24 # Subgroups  
25 G_in = G[:int(n_neurons/2)]  
26 G_out = G[int(n_neurons/2):]  
27  
28  
29 # Initializations  
30 G.v = 'vth + rand()*(vr-vth)' # All neurons  
31 G_in.I = '0.6*volt' # First group  
32 G_out.I = '0*volt' # Second group  
33  
34  
35 # Connectivity and synapses  
36 S = Synapses(G_in, G_out, on_pre='v post += 1*mV')  
S.connect(p=1.0)  
37  
38  
39  
40 # Monitors  
41 monitor_v = StateMonitor(G, 'v', record=True)  
42 monitor_spk_in = SpikeMonitor(G_in)  
43 monitor_spk_out = SpikeMonitor(G_out)  
44  
45  
46 # Run simulation  
run(20*ms)
```

$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

# Basics - Synapses

- Post spike effect
  - Conductance (more complex neuron models)



```

13 ## --- FIRST LAYER -- ##
14
15 # Parameters
16 vr = -70*mV
17 vth = -50*mV
18 tau = 20*ms
19 n_neurons = 8
20 refractory = 1*ms
21
22
23 # Neurons population
24 eqs=''''
25 dv/dt = (vr - v + I)/tau : volt (unless refractory)
26 I : volt
27 '''
28 reset=''''
29 v=vr
30 '''
31 G_in = NeuronGroup(int(n_neurons/2), eqs, threshold='v>vth', reset=reset,
32 refractory=refractory, method='euler')
33
34
35 ## --- SECOND LAYER -- ##
36
37 # New parameters
38 Ee = 0*mV # Excitatory input
39 El = -74*mV # Inhibitory input
40 tau_e = 10*ms
41
42 # Neurons population
43 eqs=''''
44 dv/dt = (ge * (Ee-vr) + El - v)/tau : volt
45 dge/dt = -ge/tau_e : 1
46 '''
47 G_out = NeuronGroup(int(n_neurons/2), eqs, threshold='v>vth', reset=reset, method='euler')
48
49
50 # Initializations
51 G_in.v = 'vth + rand()*(vr-vth)'
52 G_out.v = 'vth + rand()*(vr-vth)'
53 G_in.I = '0.6*volt'
54
55
56 # Connectivity and synapses
57 S = Synapses(G_in, G_out, on_pre='ge_post += 1')
58 S.connect(p=1.0)

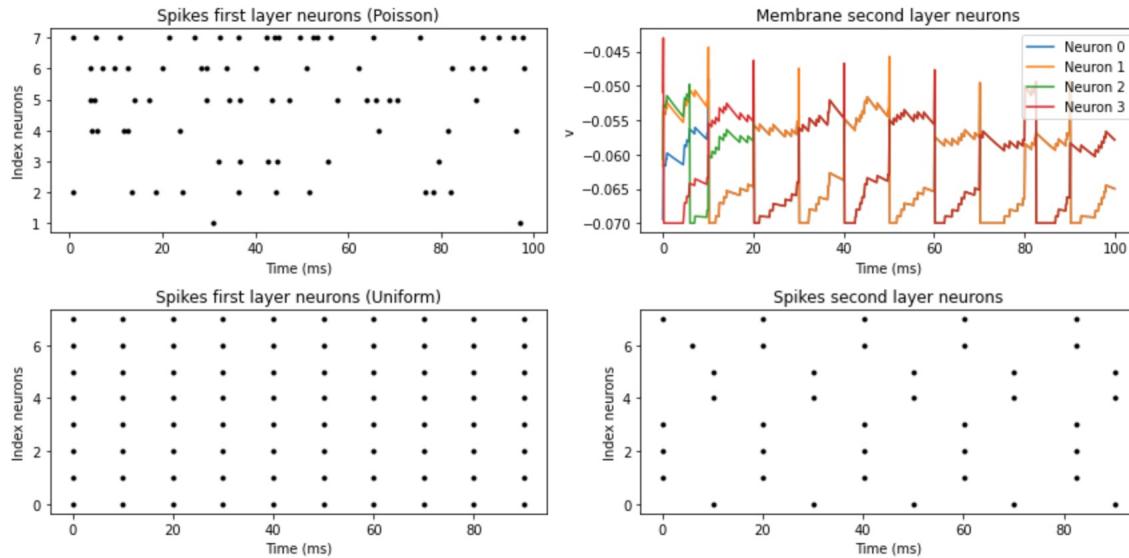
```

$\tau \frac{\partial V}{\partial t} = g_e * (E_e - V_r) + E_l - V$

$\tau_e \frac{\partial g_e}{\partial t} = -g_e$

# Simulations - Neurons input

- Input current
- Input spikes
  - PoissonGroup
  - SpikesGenerator



```

21 # Spiking rate different for each input neuron in Poisson group (between 10 and 220 Hz)
22 spk_rates_input = np.arange(n_neurons)*30*Hz + 10*Hz
23
24 # Fixed spiking rate for every neuron in the uniform spike generator
25 spk_period_input = (n_neurons+2)*ms
26
27
28 ## --- FIRST LAYER -- ##
29
30 # Input poisson spikes
31 G_in1 = PoissonGroup(n_neurons, spk_rates_input)
32
33 # Input uniform spikes
34 G_in2 = SpikeGeneratorGroup(N=n_neurons, indices=range(n_neurons), times=[0]*n_neurons*ms,
35 period=spk period input)
36
37
38 ## --- SECOND LAYER -- ##
39
40 # Neurons population
41 eqs='''  

42 dv/dt = (vr - v + I)/tau : volt (unless refractory) ←
43 I : volt
44 '''
45 reset='''  

46 v=vr
47 '''
48 G_out = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset, refractory=refractory,
49 method='euler')
50
51
52 # Initializations
53 G_out.v = 'vth + rand()*(vr-vth)'
54 G_out.I = '0*volt'
55
56
57 # Connectivity and synapses
58 S1 = Synapses(G_in1, G_out, on_pre='v_post += 1*mV')
59 S1.connect(p=1.0)
60 S2 = Synapses(G_in2, G_out, on_pre='v_post += 1*mV')
61 S2.connect(p=1.0)

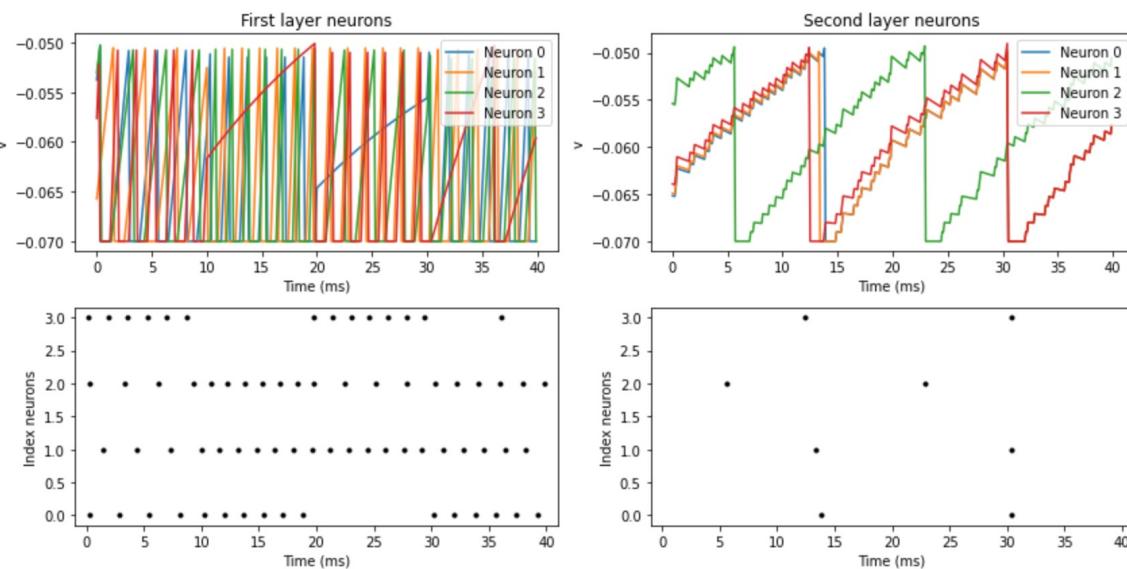
```

$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

# Simulations - Neurons input

- Non-static input
  - Time array
  - Run regularly



```
G_in.I = ta_inputs
```

When  
using only  
Brian

```

21 # Neurons population
22 eqs='''  

23 dv/dt = (vr - v + I)/tau : volt (unless refractory)  

24 I : volt  

25 '''  

26 reset='''
27 v=vr
28 '''

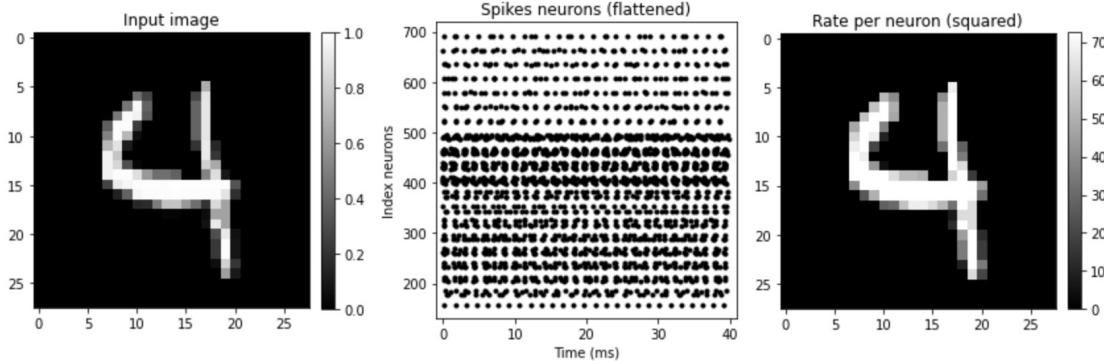
29 G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,
30                         refractory=refractory, method='euler')
31
32 # Subgroups
33 G_in = G[:int(n_neurons/2)]
34 G_out = G[int(n_neurons/2):]
35
36
37 # Initializations
38 G.v = 'vth + rand()*(vr-vth)' # All neurons
39 G_out.I = '0*volt' # Second group
40
41
42 # Create random input changing every 10 seconds
43 inputs = np.random.random((4, n_neurons))*volt
44 ta_inputs = TimedArray(inputs, dt=10*ms)
45
46
47 # Parallel run modifying input
48 G_in.run_regularly('I = ta_inputs(t, i)', dt=10*ms)

50
51 # Connectivity and synapses
52 S = Synapses(G_in, G_out, on_pre='v_post += 1*mV')
53 S.connect(p=1.0)
54
55
56 # Monitors
57 monitor_v = StateMonitor(G, 'v', record=True)
58 monitor_spk_in = SpikeMonitor(G_in)
59 monitor_spk_out = SpikeMonitor(G_out)
60
61
62 # Run simulation
63 run(40*ms)

```

# Simulations - Neurons input

- Using datasets
  - Example: MNIST
- One neuron per input value
- Input values as input currents or as spikes trains (value= spiking rate)

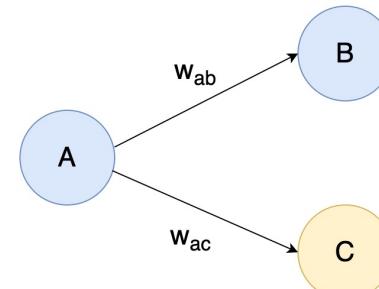
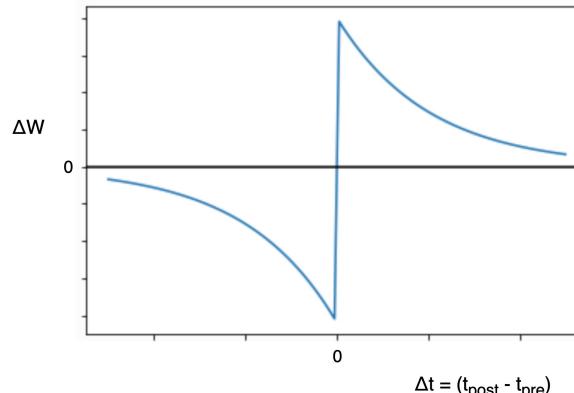


```
3 # Get MNIST dataset (only training subset)
4 from keras.datasets import mnist
5 (train_X, train_Y), (_, _) = mnist.load_data()
6 train_X = (train_X - 0) / (255 - 0) # Normalize
7 train_X = train_X.reshape(train_X.shape[0], train_X.shape[1]*train_X.shape[2])
8 sample = train_X[np.random.choice(len(train_X))] # Pick up random sample
9
10 # Restart Brian objects
11 start_scope()
12
13 # Parameters
14 vr = -70*mV
15 vth = -50*mV
16 tau = 20*ms
17 n_neurons = len(sample)
18 refractory = 1*ms
19
20 # Neurons population
21 eqs=''
22 dv/dt = (vr - v + I)/tau : volt (unless refractory)
23 I : volt
24 ...
25 reset=''
26 v=vr
27 ...
28 G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,
29                   refractory=refractory, method='euler')
30
31 # Initializations
32 G.v = 'vth + rand()*(vr-vth)' # All neurons
33 G.I = sample*volt
34
35 # Monitors
36 monitor_spk = SpikeMonitor(G)
37
38 # Run simulation
39 simulation_time=40
40 run(simulation_time*ms)
41
42 # Calculate spiking rate of each neuron
43 rates = rate_per_neuron(monitor_spk, n_neurons, simulation_time)
```

$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

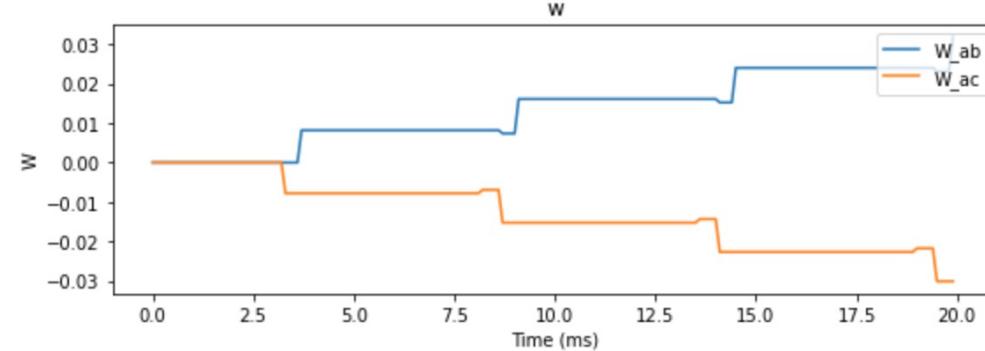
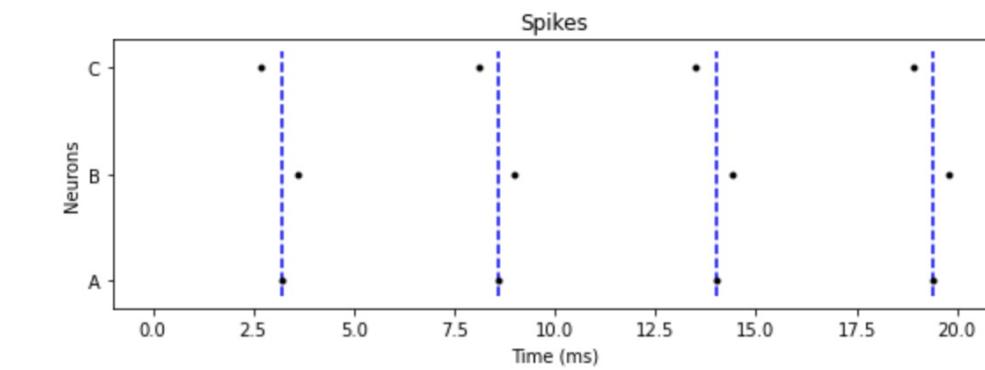
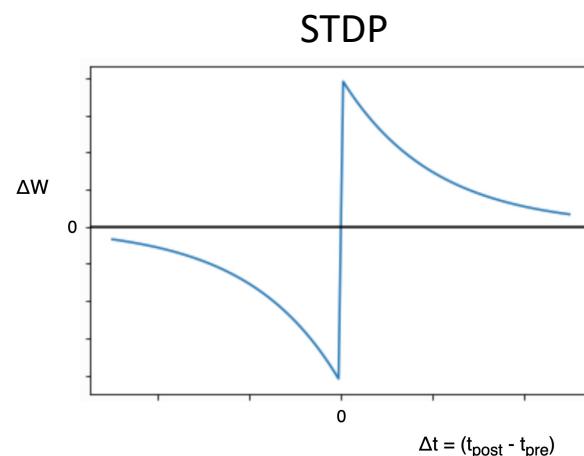
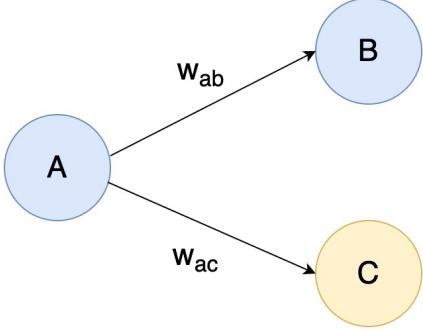
# Simulations - Weights updates

- Learning at spike times
  - “on\_pre” and “on\_post”
  - Equations can be simulated in synapses (e.g. traces spiking rate)
- Example:
  - Simple STDP with 3 neurons
  - Neuron B spikes right after A and neuron C spikes right before A



```
20 # Neurons population
21 eqs=''
22 dv/dt = (vr - v + I)/tau : volt (unless refractory)
23 I : volt
24 ''
25 reset=''''
26 v=vr
27 ''
28 G = NeuronGroup(3, eqs, threshold='v>vth', reset=reset,
29                   refractory=refractory, method='euler')
30
31 # Subgroups
32 G_in = G[0]
33 G_out = G[1:]
34
35 # Initializations
36 G.v = [-64, -66, -62]*mV
37 G.I = '0.1*volt'
38
39 # Increase trace
40 inc = 0.01
41 tau_trace = tau/10 # Much faster than neuron's membrane
42
43 # Connectivity and synapses
44 # *Neuron's spikes don't affect other neurons
45 # *xpre and xposts are traces used to track the neuron's spiking rate
46 # *xpre is positive and xpost negative
47 S = Synapses(G_in, G_out,
48               '''
49               w : 1
50               dxpre/dt = -xpre/tau_trace : 1 (event-driven)
51               dxpost/dt = -xpost/tau_trace : 1 (event-driven)
52               ''',
53               on_pre=''''
54               xpre += inc
55               w += xpost
56               ''',
57               on_post=''''
58               xpost += -inc
59               w += xpre
60               ''')
61 S.connect(p=1.0)
62
63 # Monitors
64 monitor_vars = StateMonitor(S, ['w', 'xpre', 'xpost'], record=True)
65 monitor_spk = SpikeMonitor(G)
66
67 # Run simulation
68 run(20*ms)
```

# Simulations - Weights updates



# Basics - GeNN backend

- Install GeNN and Brian2GeNN beforehand
  - Configure paths
- Add just a few lines
- Program compiled on the run
  - It adds compilation time
  - Worth it only in large simulations

```
1 from brian2 import *
2
3 import brian2genn
4 set_device('genn')
5
6
7 # Restart GeNN backend
8 if GeNN_backend:
9     device.reinit()
10    device.activate()
11
12
13 # Restart Brian objects
14 start_scope()
15
```

# Hands on

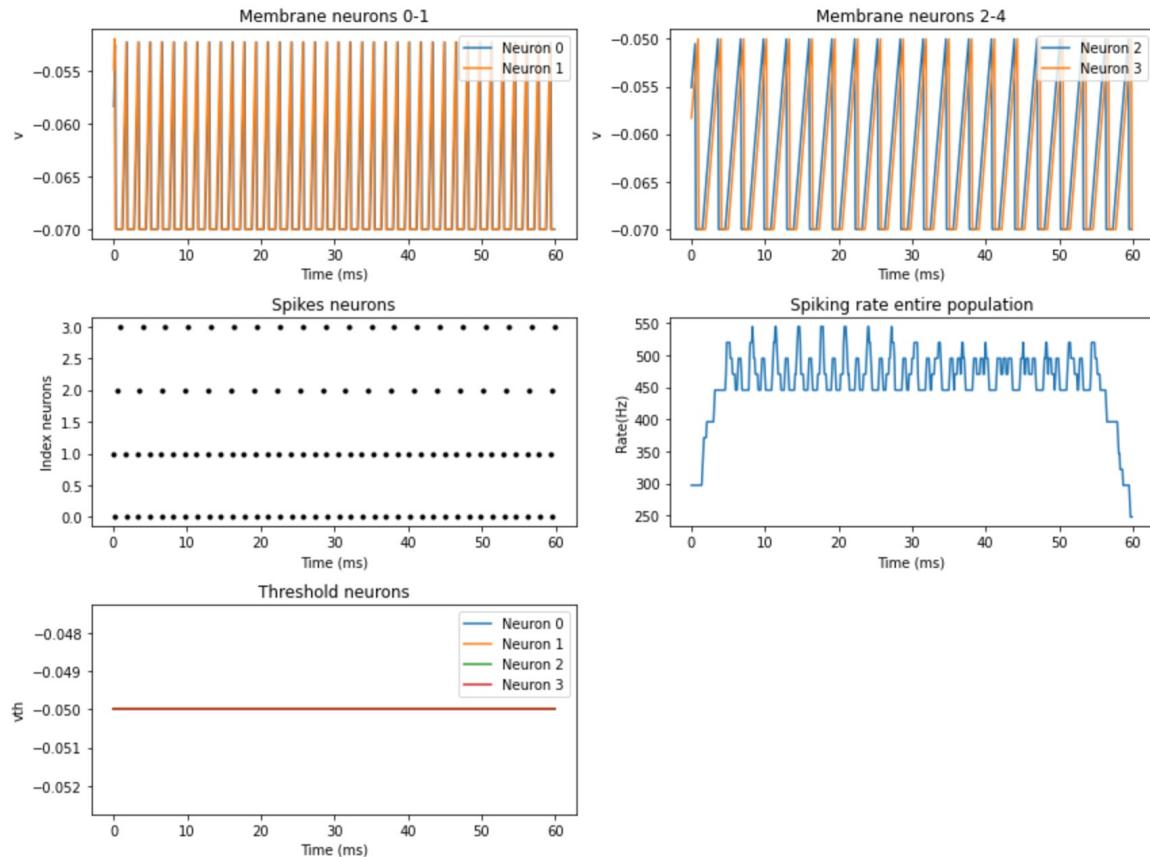
# Leaky Integrate and Fire with adaptation

- Execute previous notebooks
    - Explore different parameters and see how they affect the simulation
  - Implement a Leaky Integrate and Fire neuron with adaptive threshold
- 
- Dynamics LIF                       $\rightarrow$        $\tau \frac{\partial V}{\partial t} = V_r - V + I, \quad if \ V > V_{th} \rightarrow spike$
  - Dynamics threshold  $\rightarrow$ 
    - $V_{th}$  tends to a  $V_{th0}$
    - $\tau_{th}$  is much faster than  $\tau$
    - $V_{th}$  increases in every spike $\tau_{th} \frac{\partial V_{th}}{\partial t} = (V_{th0} - V_{th})$

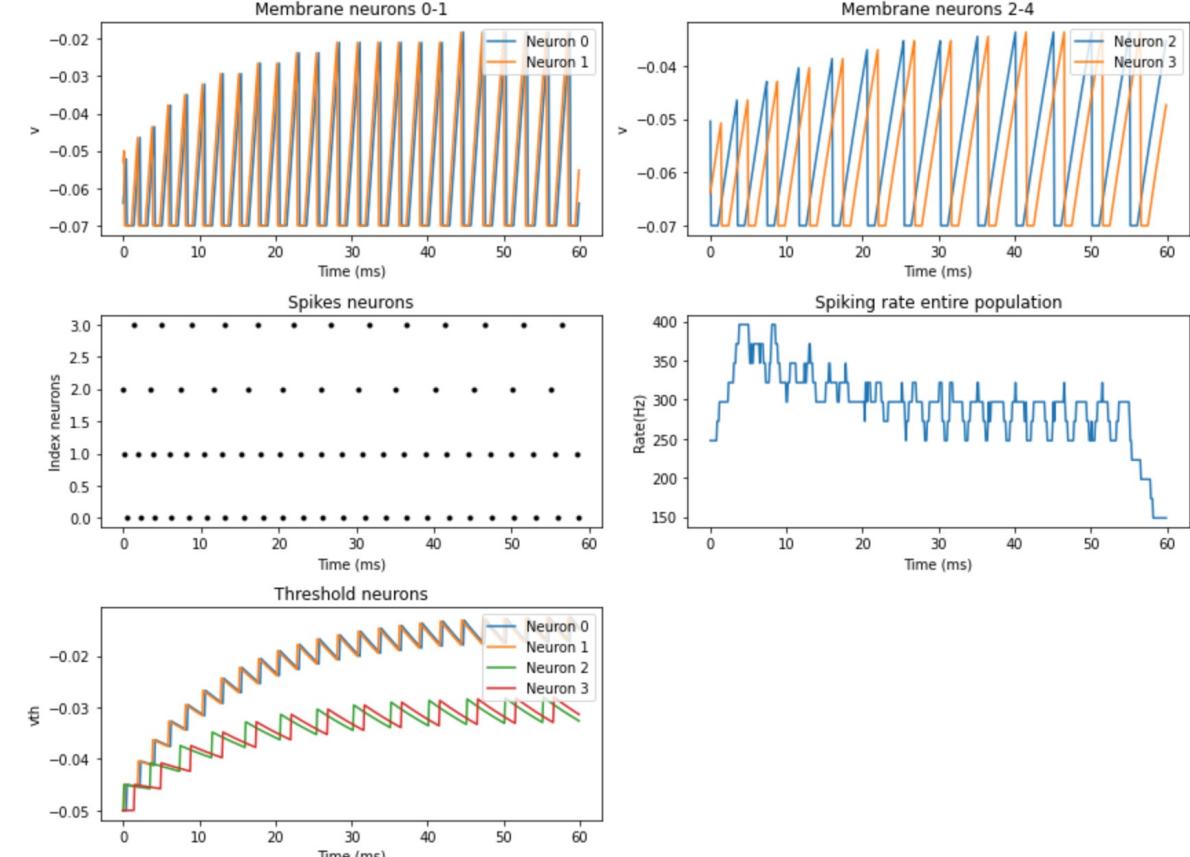
Link to notebook: [https://colab.research.google.com/drive/11BDgUn\\_zkbtLn7uw\\_isz4i1Ro\\_5nbf3O?usp=sharing](https://colab.research.google.com/drive/11BDgUn_zkbtLn7uw_isz4i1Ro_5nbf3O?usp=sharing)  
[https://github.com/jesusgf96/Brian2GeNN\\_introduction/blob/main/Brian2GeNN\\_tutorial.ipynb](https://github.com/jesusgf96/Brian2GeNN_introduction/blob/main/Brian2GeNN_tutorial.ipynb)

# Leaky Integrate and Fire with adaptation

Fixed threshold

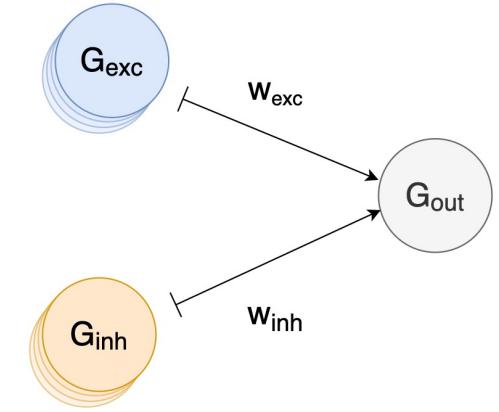


Adaptive threshold



# Balance excitation/inhibition

- LIF neuron receives excitatory and inhibitory inputs
- Excitation received is larger than inhibition
- Implement learning rule to balance excitation/inhibition by updating  $W_{inh}$



## Dynamics LIF

("I" increases/decreases with spikes)

("I" decays over time)

( $\tau_I$  is much faster than  $\tau$ )

$$\tau \frac{\partial V}{\partial t} = V_r - V + (I_{exc} + I_{inh})$$

$$\tau_I \frac{\partial I_{exc}}{\partial t} = -I_{exc}$$

$$\tau_I \frac{\partial I_{inh}}{\partial t} = -I_{inh}$$

## Learning rule similar to Vogels et al

(Updates at spikes time)

(Traces to track spiking rate)

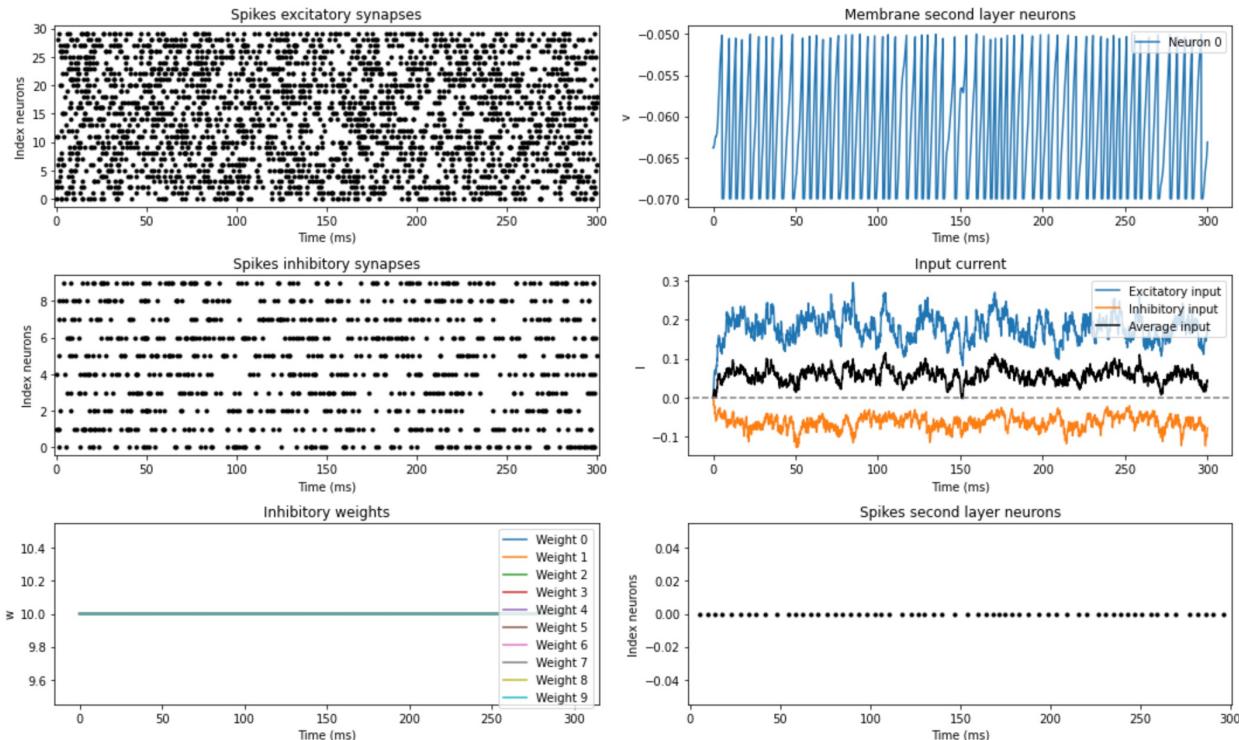
$$\Delta W_{inh} = \begin{cases} \eta(X_{post}), & \text{if presynaptic spike} \\ \eta(X_{pre}), & \text{if postsynaptic spike.} \end{cases}$$

$$\tau \frac{\partial X_i}{\partial t} = -X_i$$

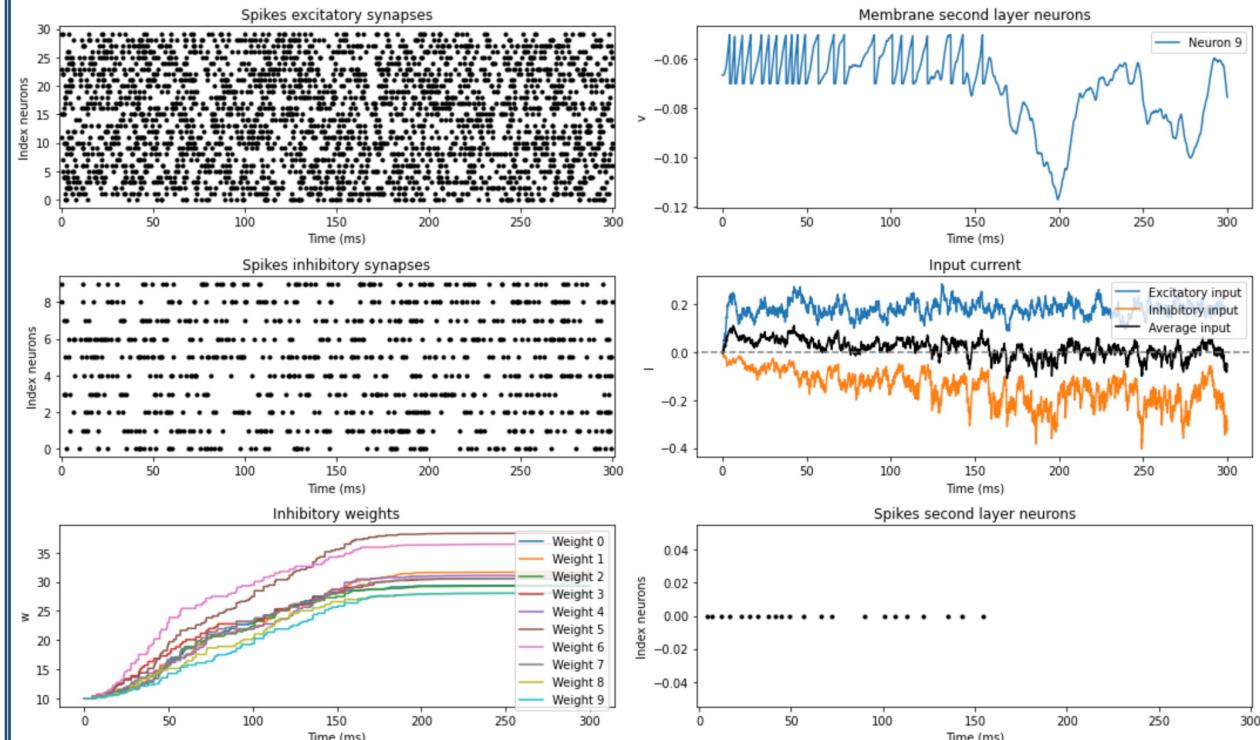
# Balance excitation/inhibition

(Poisson input spikes)

No learning



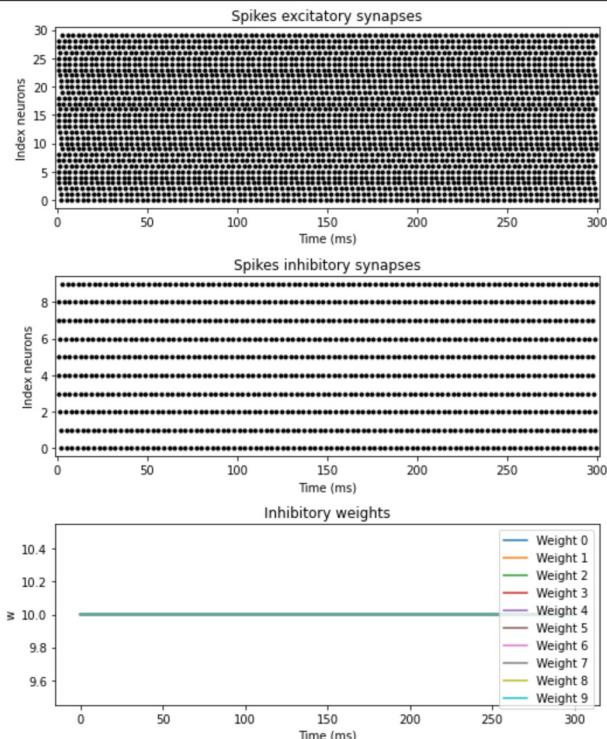
Learned weights



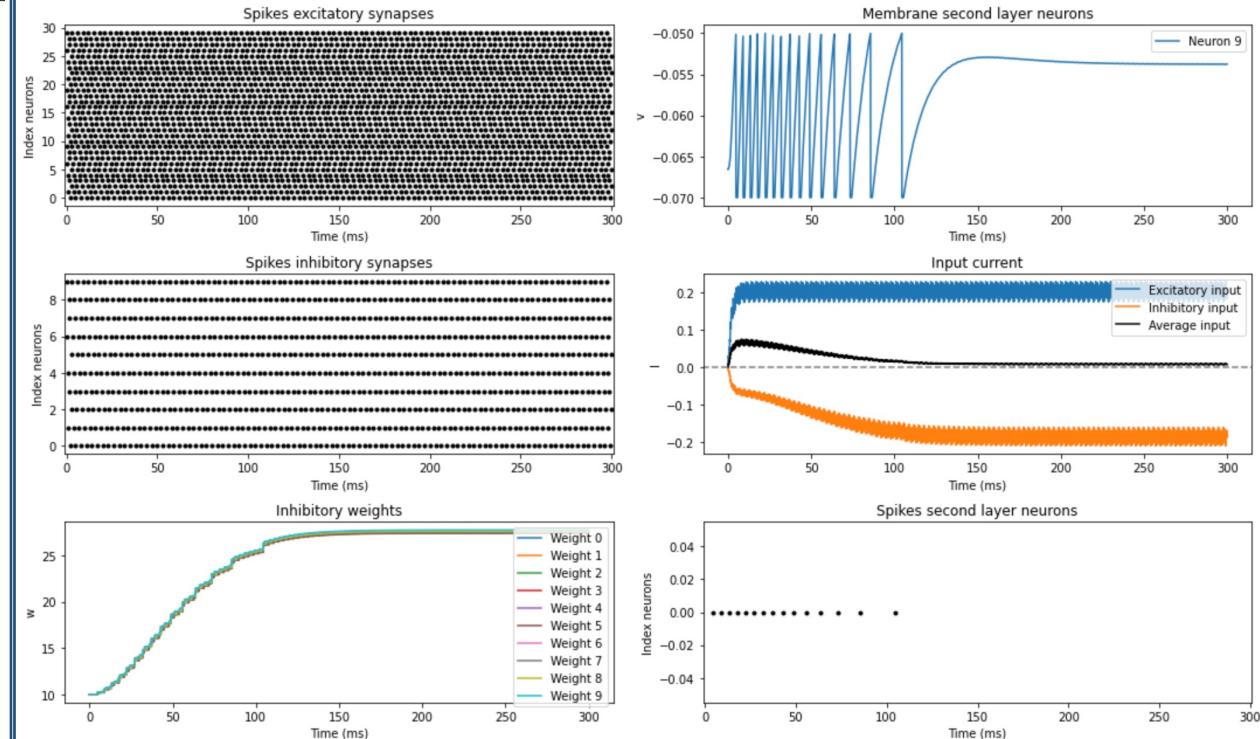
# Balance excitation/inhibition

(Uniform input spikes)

No learning



Learned weights



# Solved tasks

- Link to notebook:

<https://colab.research.google.com/drive/1e4vvgqbMEXZmjnvtnkfuQmr4zFZsjzyB?usp=sharing>

[https://github.com/jesusgf96/Brian2GeNN\\_introduction/blob/main/Brian2GeNN\\_solved\\_tasks.ipynb](https://github.com/jesusgf96/Brian2GeNN_introduction/blob/main/Brian2GeNN_solved_tasks.ipynb)

# Complex models

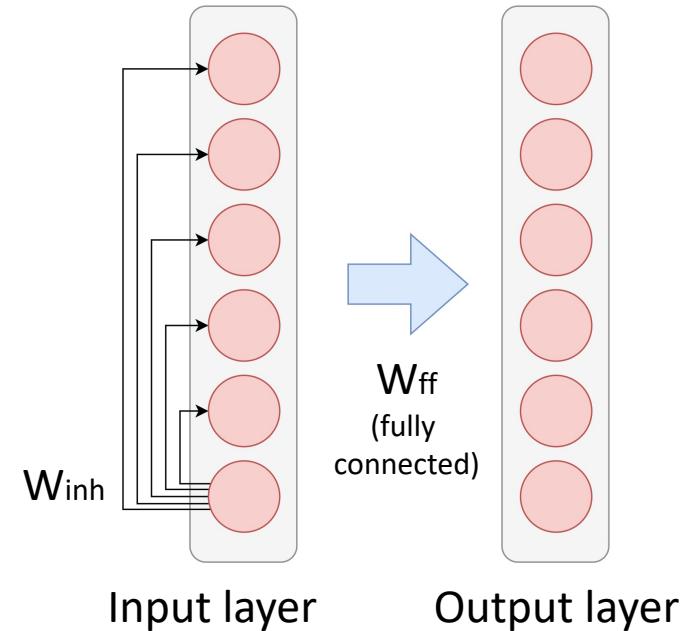
# Disadvantages Brian2GeNN

- Not supported features
  - Time Arrays on GPU
    - Use Run Regularly to call them from CPU
  - Multiple clocks (Only one run regularly)
  - Multiple networks
  - Multiple runs
    - Restart GeNN backend before every run
- Monitoring values is slow (Data is copied to CPU constantly)

There are others  
not supported features but these  
are the ones everyone will probably  
face at some point

# Disadvantages Brian2GeNN

- My current work
  - Biologically plausible learning rules
  - Learning at spike times
  - Inhibitory synapses to decorrelate inputs
  - Feedforward synapses to learn transformations
  - Testing on MNIST dataset
    - 60000 training & 10000 test samples
- 100 epochs training over entire dataset takes ~4 days
  - Each sample is presented during 200ms
  - GPU: Quadro RTX 6000



# Multicompartment neurons