Brian2GeNN tutorial

Progress – Week 03/12/2021

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

Brian2, GeNN and Brian2Genn

Brian2

- Python-based SNN simulator
- High level and natural desciptions
  - 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

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

```
from brian2 import *
start_scope()

# Parameters
vr = -70*mV
vth = -50*mV
tau = 20*ms

# Neurons population
eqs='''
dv/dt = (vr-v+I)/tau: volt (unless refractory)
I : volt
'''
G = NeuronGroup(10, eqs, threshold='v>vth', reset='v=vr',
                refractory=1*ms, method='euler')
G.I = 'rand()*volt'
G.v = 'vth + rand()*(vr-vth)'

# Connectivity and synapses
S = Synapses(G, G, on_pre='v_post += 1*mV')
S.connect(j='k for k in range(i-3, i+4) if i!=k',
          skip_if_invalid=True)

# Run simulation
run(100*ms)
```

Stimberg, M., Brette, R., & Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. Elife, 8, e47314.

GeNN

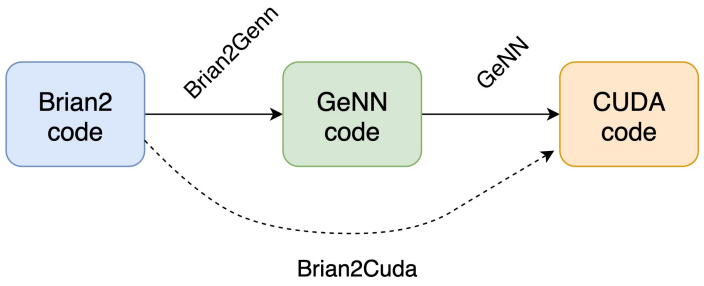
- C++-based SNN simulator
- Object oriented descriptions
  - CUDA code generated from them
  - Many implemented models
- Custom models as extensions of existing templates
  - Plug them in the simulation
  - Hard to debug
- Supports GPU execution

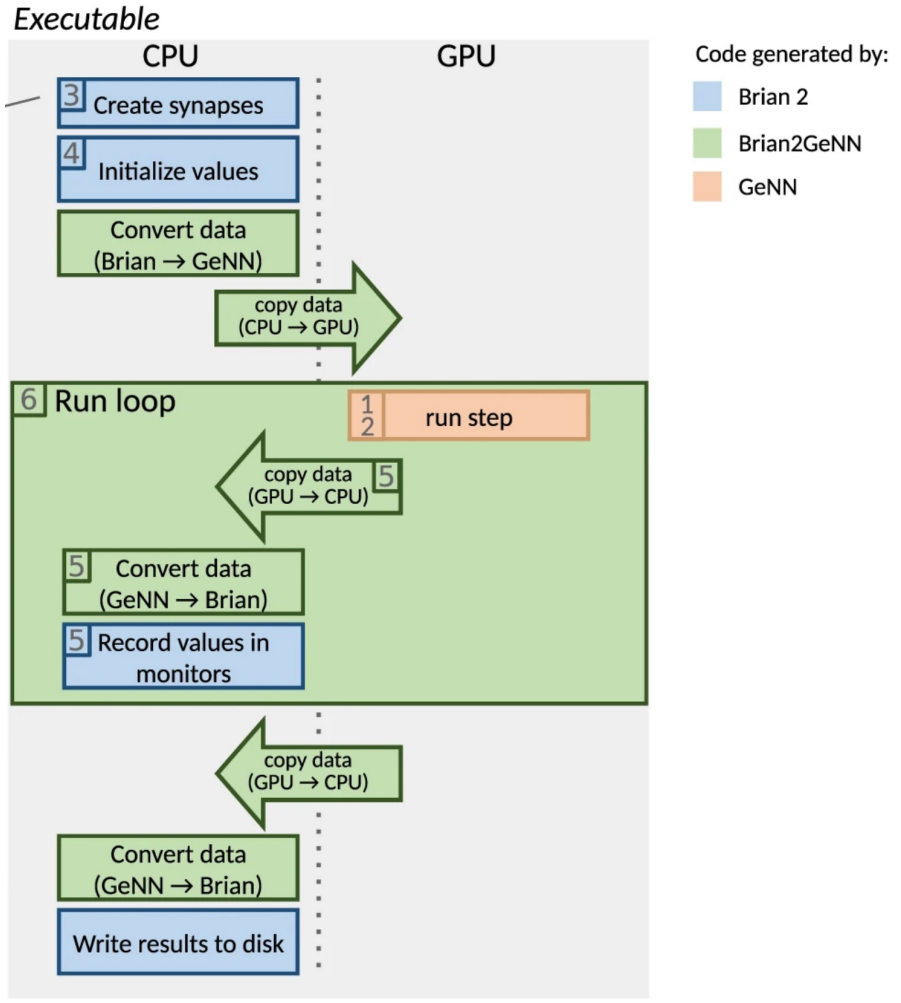
```
// Model definintion file
#include "modelSpec.h"

// Connectivity
class Ring : public InitSparseConnectivitySnippet::Base
{
public:
    DECLARE_SNIPPET(Ring, 0);
    SET_ROW_BUILD_CODE(
        "$(addSynapse, ($(id_pre) + 1) % $(num_post));\n"
        "$(endRow);\n");
    SET_MAX_ROW_LENGTH(1);
};
IMPLEMENT_SNIPPET(Ring);


// LIF neuron
class LeakyIntegrator : public NeuronModels::Base
{
public:
    DECLARE_MODEL(LeakyIntegrator, 4, 1);

    SET_SIM_CODE("$(V) += ($(Vr)-$(V)+$(I))*(DT/$(tau));");

    SET_THRESHOLD_CONDITION_CODE("$(V) >= $(Vth)");
    SET_RESET_CODE("$(V) = $(Vth);");

    SET_PARAM_NAMES({"tau"}, {"Vr"}, {"Vth"}, {"I"});

    SET_VARS({{"V", "scalar"}});
};
IMPLEMENT_MODEL(LeakyIntegrator);


// Neurons with connectivity and synapses
void modelDefinition(ModelSpec &model)
{
    // definition of tenHHRing
    model.setDT(0.1);
    model.setName("MyModel");

    // Parameters
    double tau = 20.0;
    double Vr = -70.0;
    double Vth = -50.0;

    // Random numbers generator for V and I (min, max)
    InitVarSnippet::Uniform::ParamValues randV(0.0, 1.0);
    InitVarSnippet::Uniform::ParamValues randI(-70.0, -50.0);

    // Input layer
    model.addNeuronPopulation<LeakyIntegrator>("Pop1", 10,
                                               LeakyIntegrator::ParamValues(tau), // tau
                                               Vr, // Rest value
                                               Vth, // Threshold
                                               LeakyIntegrator::VarValues(initVar<InitVarSnippet::Uniform>(randI), // I
                                               LeakyIntegrator::VarValues(initVar<InitVarSnippet::Uniform>(randV)); //V


    WeightUpdateModels::StaticPulse::VarValues s_ini(
         -0.2); //

    PostsynapticModels::ExpCond::ParamValues ps_p(
        1.0,    // 0 - tau_S: decay time constant for S [ms]
        -80.0); // 1 - Erev: Reversal potential

    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::ExpCond>(
        "Poplself", SynapseMatrixType::SPARSE_GLOBALG, 100,
        "Pop1", "Pop1",
        {}, s_ini,
        ps_p, {},
        initConnectivity<Ring>());
}
```

Yavuz, E., Turner, J., & Nowotny, T. (2016). GeNN: a code generation framework for accelerated brain simulations. Scientific reports, 6(1), 1-14.

Brian2GeNN

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





Stimberg, M., Goodman, D. F., & Nowotny, T. (2020). Brian2GeNN: accelerating spiking neural network simulations with graphics hardware. Scientific reports, 10(1), 1-12.

Practical tutorial Brian2GeNN

Basics - Neurons

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

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

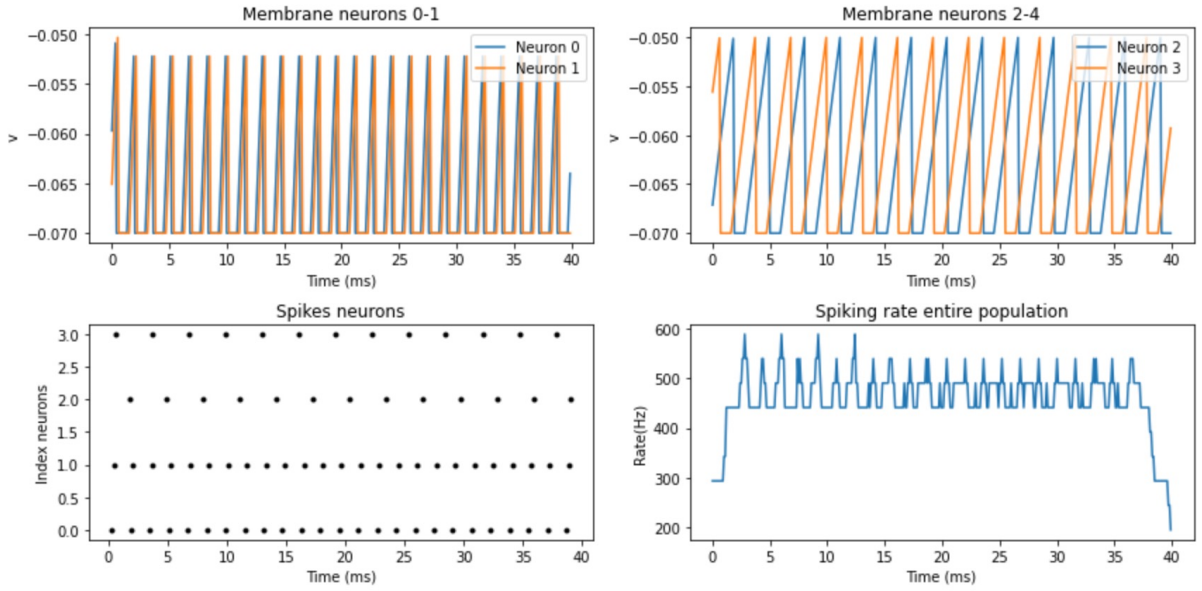
```
# Restart Brian objects
start_scope()

# Parameters
vr = -70*mV
vth = -50*mV
tau = 20*ms
n_neurons = 4
refractory = 1*ms

# Neurons population
eqs='''
dv/dt = (vr - v + I)/tau : volt (unless refractory)
I : volt
'''
reset='''
v=vr
'''
G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,
                refractory=refractory, method='euler')

# Subgroups
G1 = G[:int(n_neurons/2)]
G2 = G[int(n_neurons/2):]

# Initializations
G.v = 'vth + rand()*(vr-vth)' # All neurons
G1.I = '0.6*volt'             # First group
G2.I = '0.2*volt'             # Second group

# Run simulation
simulation_time=40
run(simulation_time*ms)
```

https://brian2.readthedocs.io/en/stable/user/models.html#id10

Basics - Monitors

- Record variables
- Record spikes
- Record spiking rate

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

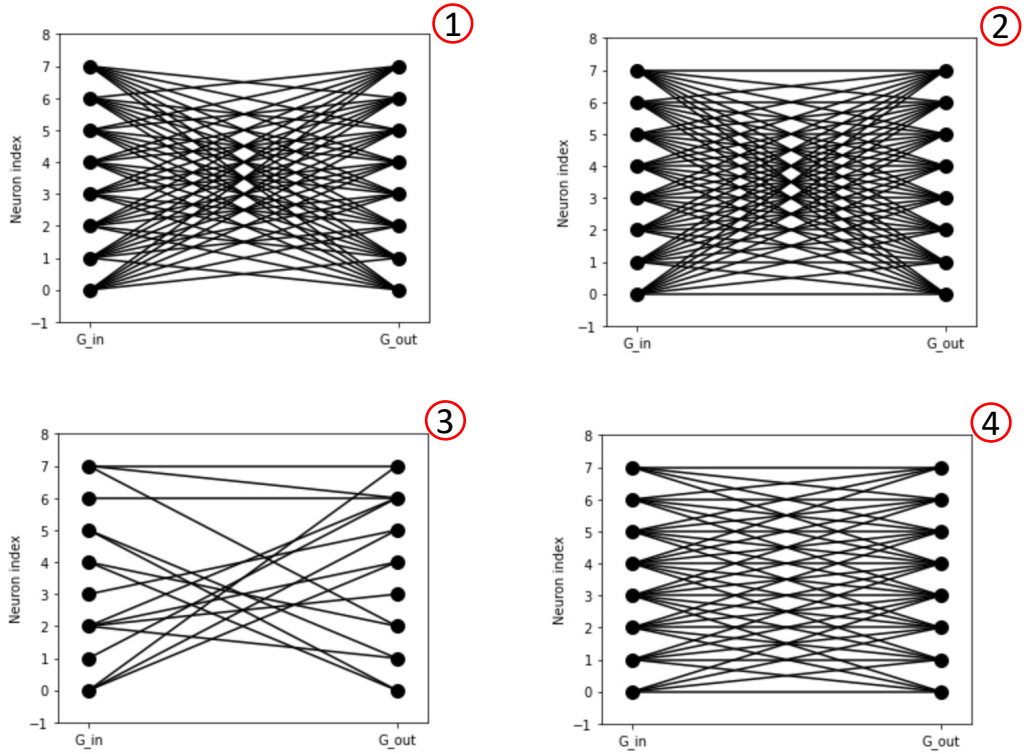



```python
# Neurons population
eqs='''
dv/dt = (vr - v + I)/tau : volt (unless refractory)
I : volt
'''
reset='''
v=vr
'''
G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,
                refractory=refractory, method='euler')

# Subgroups
G1 = G[:int(n_neurons/2)]
G2 = G[int(n_neurons/2):]

# Initializations
G.v = 'vth + rand()*(vr-vth)' # All neurons
G1.I = '0.6*volt'             # First group
G2.I = '0.2*volt'             # Second group

# Monitors
monitor_v = StateMonitor(G, 'v', record=True)
monitor_spk = SpikeMonitor(G)
monitor_rate = PopulationRateMonitor(G)

# Run simulation
run(40*ms)

# Visualizations
plt.figure(figsize=(12,6))
plt.subplot(221)
for n in range(int(len(monitor_v.v)/2)):
    plot(monitor_v.t/ms, monitor_v.v[n], label='Neuron '+str(n))
xlabel('Time (ms)')
ylabel('v')
plt.title('Membrane neurons 0-'+str(int(len(monitor_v.v)/2)-1))
plt.legend(loc='upper right')

plt.subplot(222)
for n in range(int(len(monitor_v.v)/2)):
    plot(monitor_v.t/ms, monitor_v.v[int(len(monitor_v.v)/2)+n],
         label='Neuron '+str(int(len(monitor_v.v)/2)+n))
xlabel('Time (ms)')
ylabel('v')
plt.title('Membrane neurons '+str(int(len(monitor_v.v)/2))+'-'+str(len(monitor_v.v)))
plt.legend(loc='upper right')

plt.subplot(223)
plot(monitor_spk.t/ms, monitor_spk.i, '.k')
xlabel('Time (ms)')
ylabel('Index neurons')
plt.title('Spikes neurons')
```

https://brian2.readthedocs.io/en/stable/user/recording.html

Basics - Synapses

- Connectivity



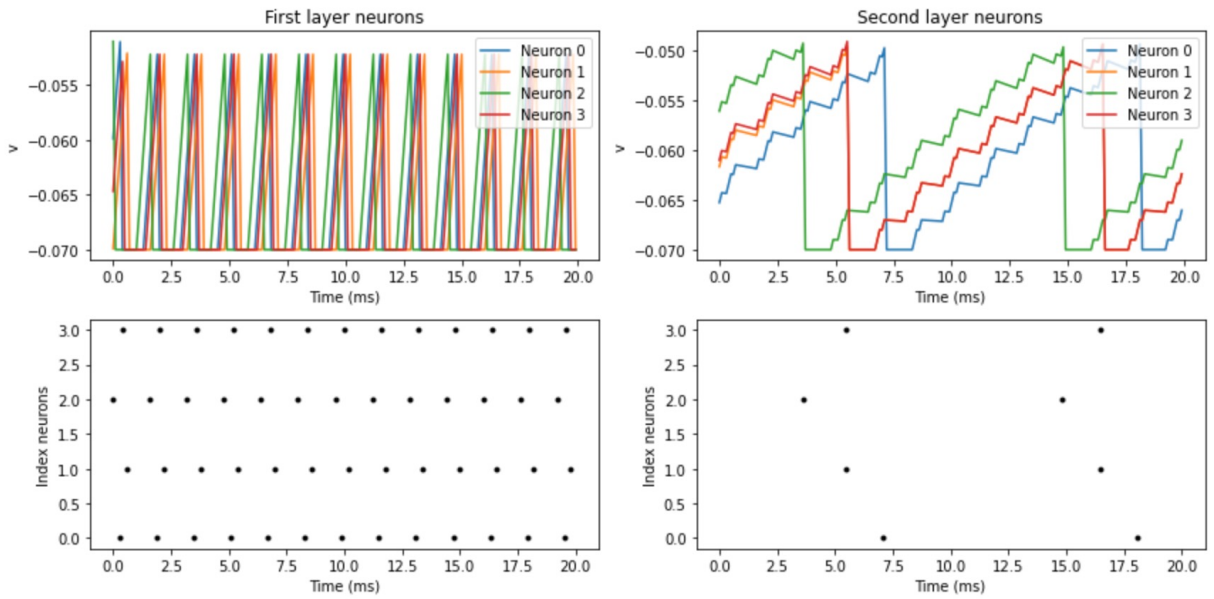
$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

```
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)
```

https://brian2.readthedocs.io/en/stable/user/synapses.html#synaptic-connection-weight-matrices

Basics - Synapses

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



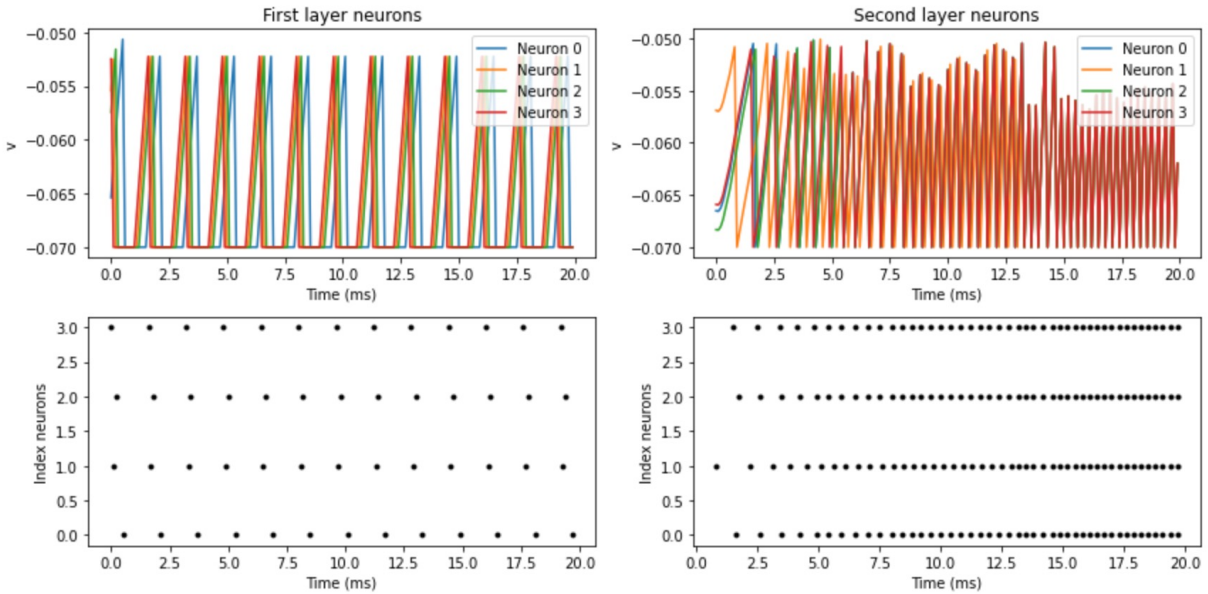
$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

```
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')
37 S.connect(p=1.0)
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
47 run(20*ms)
```

https://brian2.readthedocs.io/en/stable/user/synapses.html#

Basics - Synapses

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



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

```python
## --- FIRST LAYER -- ##

# Parameters
vr = -70*mV
vth = -50*mV
tau = 20*ms
n_neurons = 8
refractory = 1*ms


# Neurons population
eqs='''
dv/dt = (vr - v + I)/tau : volt (unless refractory)
I : volt
'''
reset='''
v=vr
'''
G_in = NeuronGroup(int(n_neurons/2), eqs, threshold='v>vth', reset=reset,
                   refractory=refractory, method='euler')


## --- SECOND LAYER -- ##

# New parameters
Ee = 0*mV     # Excitatory input
El = -74*mV   # Inhibitory input
taue = 10*ms

# Neurons population
eqs='''
dv/dt = (ge * (Ee-vr) + El - v)/tau : volt
dge/dt = -ge/taue : 1
'''
G_out = NeuronGroup(int(n_neurons/2), eqs, threshold='v>vth', reset=reset, method='euler')


# Initializations
G_in.v = 'vth + rand()*(vr-vth)'
G_out.v = 'vth + rand()*(vr-vth)'
G_in.I = '0.6*volt'


# Connectivity and synapses
S = Synapses(G_in, G_out, on_pre='ge_post += 1')
S.connect(p=1.0)
```

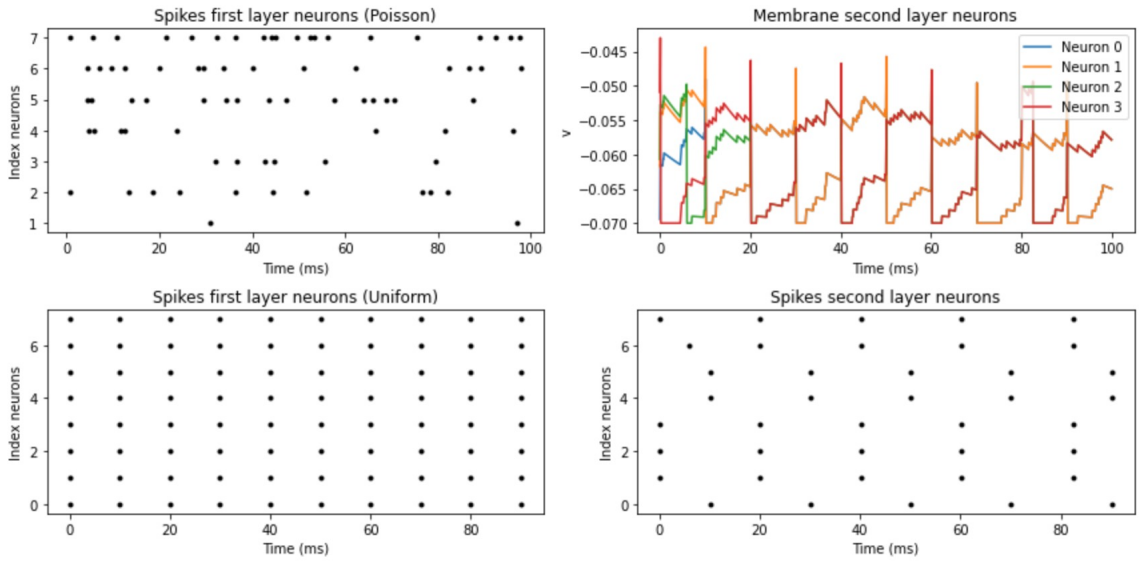
$$\tau \frac{\partial V}{\partial t} = g_e * (E_e - V_r) + E_l - V)$$

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

https://brian2.readthedocs.io/en/stable/user/synapses.html#

Simulations - Neurons input

- Input current
- Input spikes
  - PoissonGroup
  - SpikesGenerator




```python
# Spiking rate different for each input neuron in Poisson group (bewteen 10 and 220 Hz)
spk_rates_input = np.arange(n_neurons)*30*Hz + 10*Hz

# Fixed spiking rate for every neuron in the uniform spike generator
spk_period_input = (n_neurons+2)*ms


## --- FIRST LAYER -- ##

# Input poisson spikes
G_in1 = PoissonGroup(n_neurons, spk_rates_input)

# Input uniform spikes
G_in2 = SpikeGeneratorGroup(N=n_neurons, indices=range(n_neurons), times=[0]*n_neurons*ms,
                            period=spk_period_input)


## --- SCEOND LAYER -- ##

# Neurons population
eqs='''
dv/dt = (vr - v + I)/tau : volt (unless refractory)
I : volt
'''
reset='''
v=vr
'''
G_out = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset, refractory=refractory,
                    method='euler')


# Initializations
G_out.v = 'vth + rand()*(vr-vth)'
G_out.I = '0*volt'


# Connectivity and synapses
S1 = Synapses(G_in1, G_out, on_pre='v_post += 1*mV')
S1.connect(p=1.0)
S2 = Synapses(G_in2, G_out, on_pre='v_post += 1*mV')
S2.connect(p=1.0)
```

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

https://brian2.readthedocs.io/en/stable/user/input.html

Simulations - Neurons input

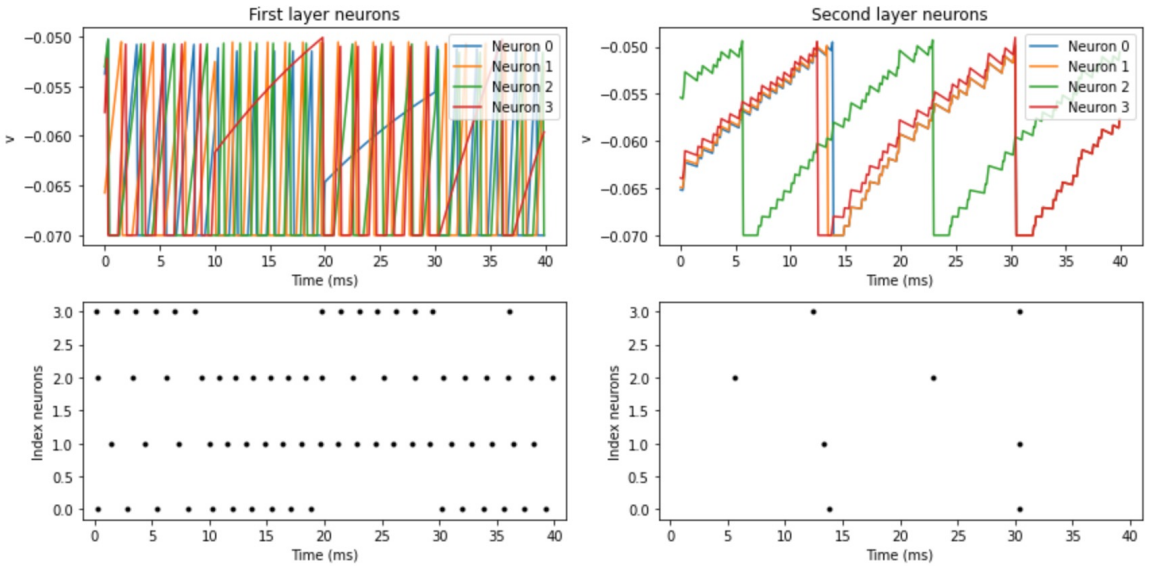
- Non-static input
  - Time array
  - Run regularly

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

```
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)
49
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)
```

```
G_in.I = ta_inputs
```

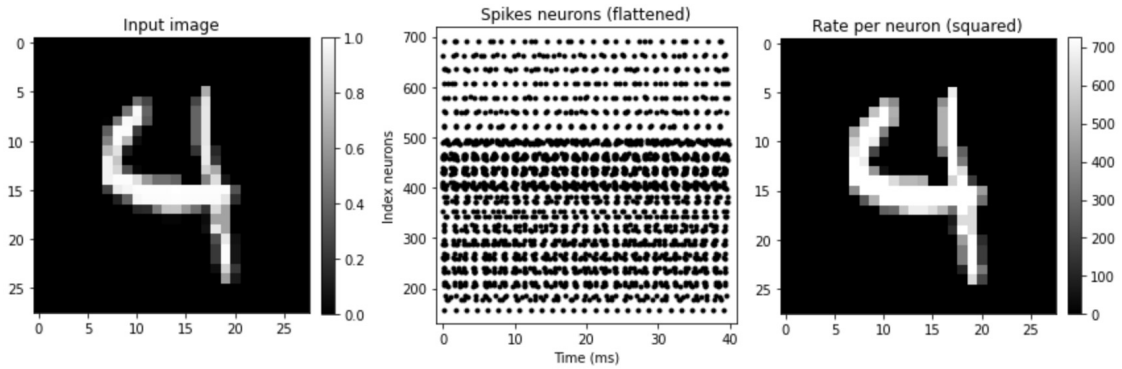
When using only Brian



https://brian2.readthedocs.io/en/stable/user/input.html

Simulations - Neurons input

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



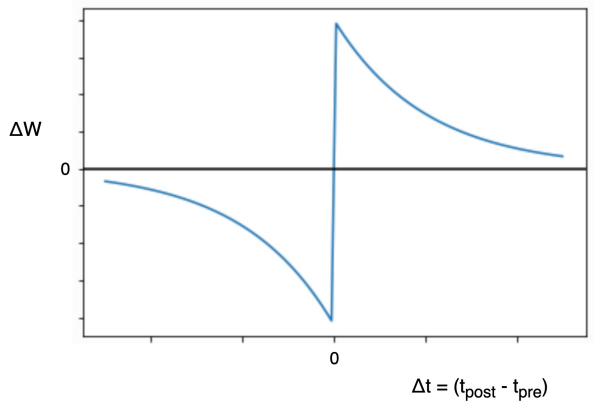

```python
# Get MNIST dataset (only training subset)
from keras.datasets import mnist
(train_X, train_Y), (_, _) = mnist.load_data()
train_X = (train_X - 0) / (255 - 0) # Normalize
train_X = train_X.reshape(train_X.shape[0], train_X.shape[1]*train_X.shape[2])
sample = train_X[np.random.choice(len(train_X))] # Pick up random sample

# Restart Brian objects
start_scope()

# Parameters
vr = -70*mV
vth = -50*mV
tau = 20*ms
n_neurons = len(sample)
refractory = 1*ms

# Neurons population
eqs='''
dv/dt = (vr - v + I)/tau : volt (unless refractory)
I : volt
'''
reset='''
v=vr
'''
G = NeuronGroup(n_neurons, eqs, threshold='v>vth', reset=reset,
                refractory=refractory, method='euler')

# Initializations
G.v = 'vth + rand()*(vr-vth)' # All neurons
G.I = sample*volt

# Monitors
monitor_spk = SpikeMonitor(G)

# Run simulation
simulation_time=40
run(simulation_time*ms)

# Calculate spiking rate of each neuron
rates = rate_per_neuron(monitor_spk, n_neurons, simulation_time)
```

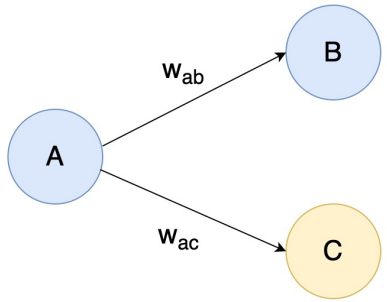
$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

https://brian2.readthedocs.io/en/stable/user/input.html

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



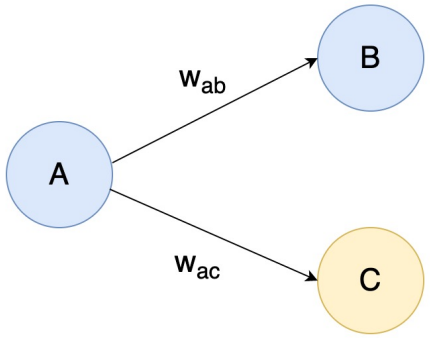


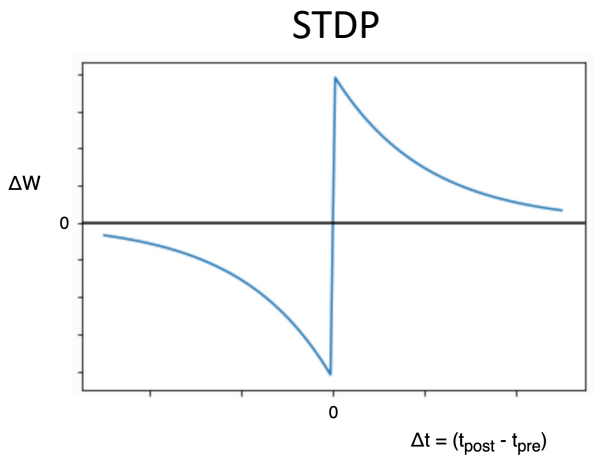
$$\tau \frac{\partial V}{\partial t} = V_r - V + I$$

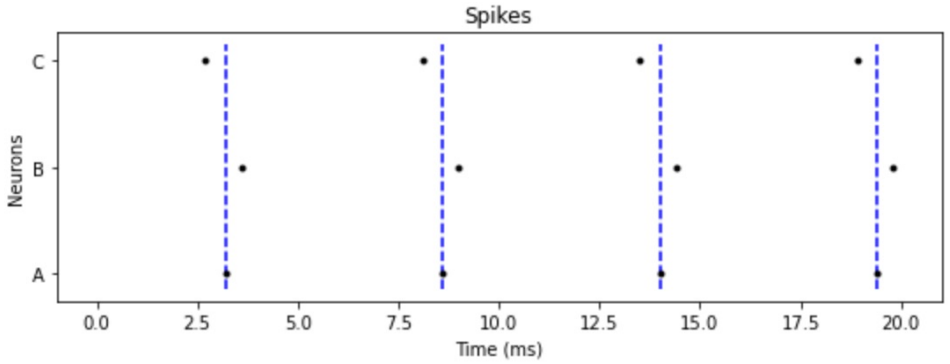
```
# Neurons population
eqs='''
dv/dt = (vr - v + I)/tau : volt (unless refractory)
I : volt
'''
reset='''
v=vr
'''
G = NeuronGroup(3, eqs, threshold='v>vth', reset=reset,
                refractory=refractory, method='euler')

# Subgroups
G_in = G[0]
G_out = G[1:]

# Initializations
G.v = [-64, -66, -62]*mV
G.I = '0.1*volt'

# Increase trace
inc = 0.01
tau_trace = tau/10  # Much faster than neuron's membrane

# Connectivity and synapses
#   *Neuron's spikes don't affect other neurons
#   *xpre and xpots are traces used to track the neuron's spiking rate
#   *xpre is positive and xpost negative
S = Synapses(G_in, G_out,
             '''
             w : 1
             dxpre/dt = -xpre/tau_trace : 1 (event-driven)
             dxpost/dt = -xpost/tau_trace : 1 (event-driven)
             ''',
             on_pre='''
             xpre += inc
             w += xpost
             ''',
             on_post='''
             xpost += -inc
             w += xpre
             ''')
S.connect(p=1.0)

# Monitors
monitor_vars = StateMonitor(S, ['w', 'xpre', 'xpost'], record=True)
monitor_spk = SpikeMonitor(G)

# Run simulation
run(20*ms)
```

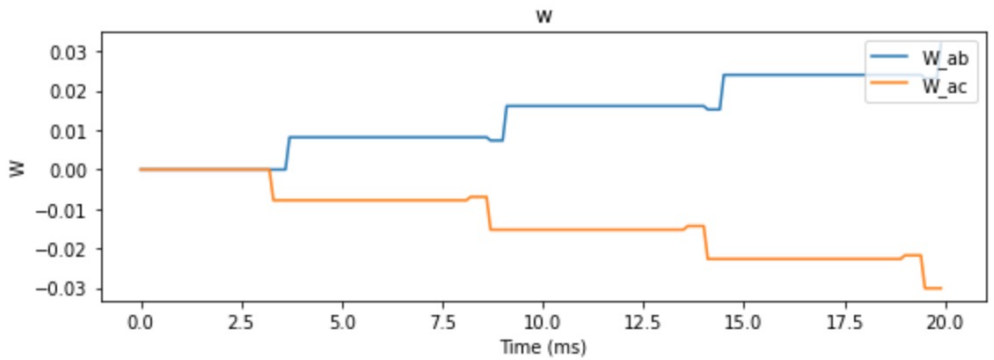
http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity

Simulations - Weights updates









http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity

Basics - GeNN backend

- Install GeNN and Brian2GeNN beforehand
  - Configure some 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
```

https://brian2genn.readthedocs.io/en/stable/introduction/index.html

Hands on

Leaky Integrate and Fire with adaptation

- Execute previous noteboks
  - Explore different parameters and see how they affect the simulation
- Implement a Leaky Integrate and Fire neuron with adaptive threshold

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

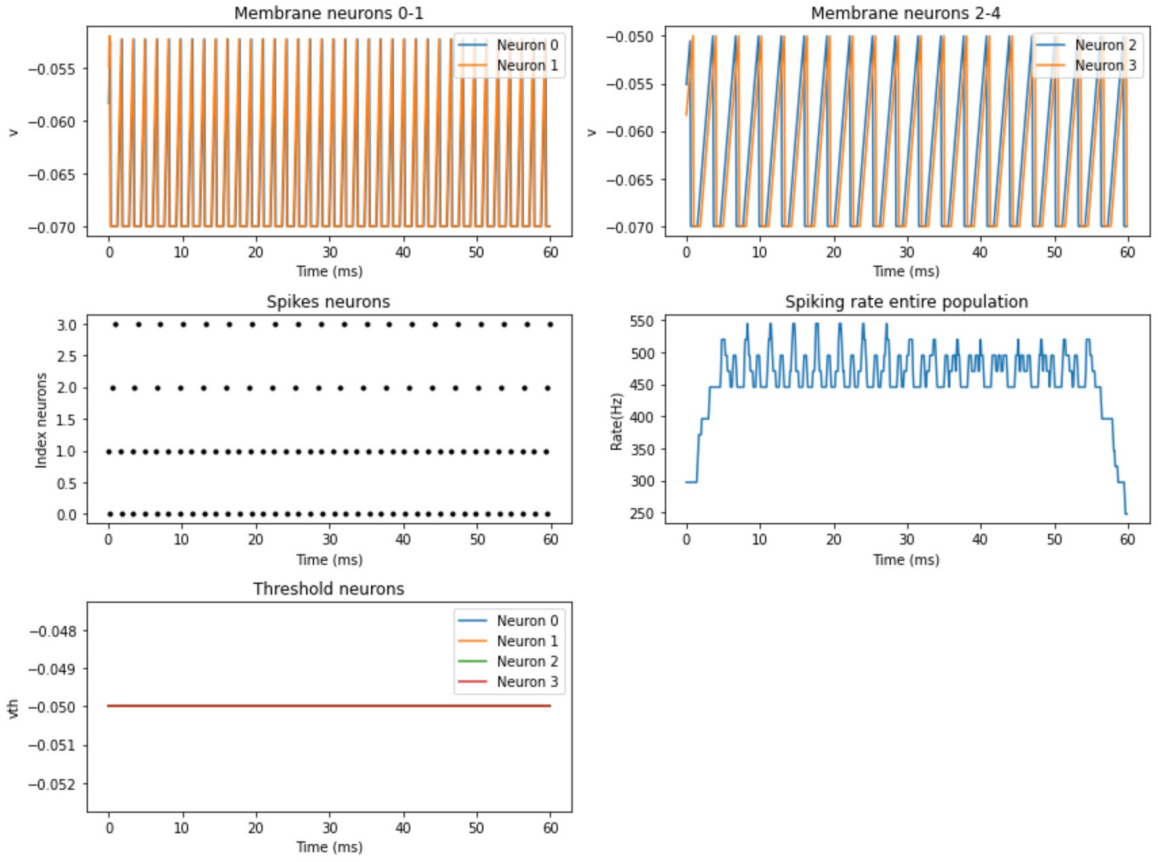
- Dynamics threshold →
  - V_{th} tends to a V_{th0}
  - τ_{th} is much faster than τ
  - V_{th} increases in every spike

$$\tau_{th} \frac{\partial V_{th}}{\partial t} = (V_{th0} - V_{th})$$

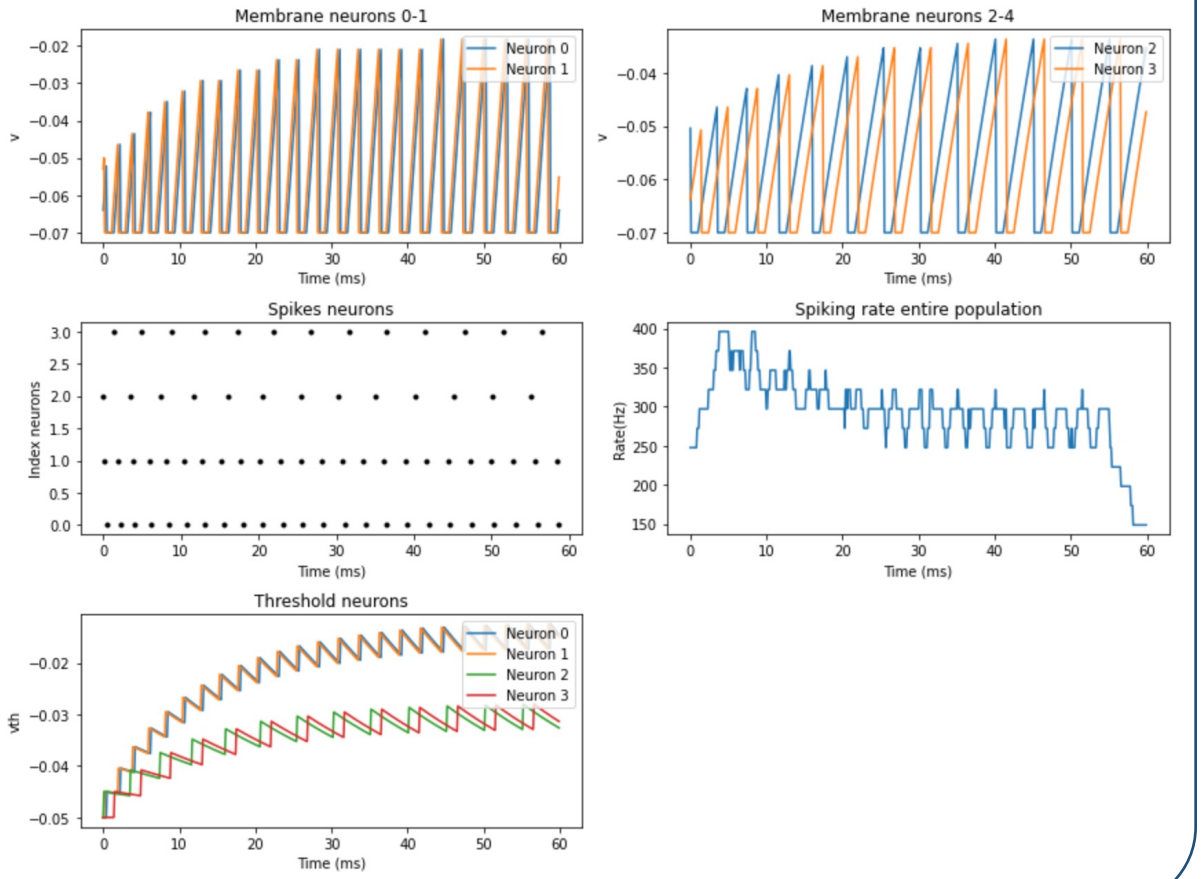
Link to notebook: https://colab.research.google.com/drive/11BDgUn_zkbtLn7uw_isz4i1Ro_5nbf3O?usp=sharing
https://github.com/jesusgf96/Brian2GeNN_introduction/blob/main/Brian2GeNN_tutorial.ipynb

Leaky Integrate and Fire with adaptation

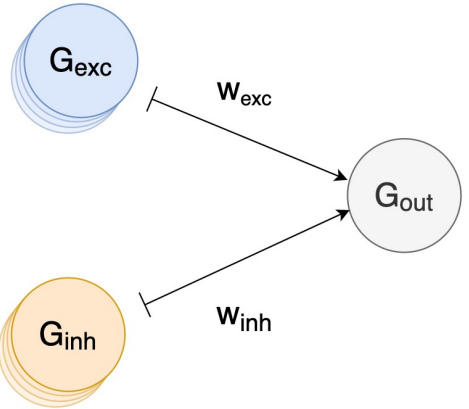
Fixed threshold



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)

(τ_I is much faster than τ)

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

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

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

Learning rule similar to *Vogels et al*

(Updates at spikes time)

(Traces to track spiking rate)

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

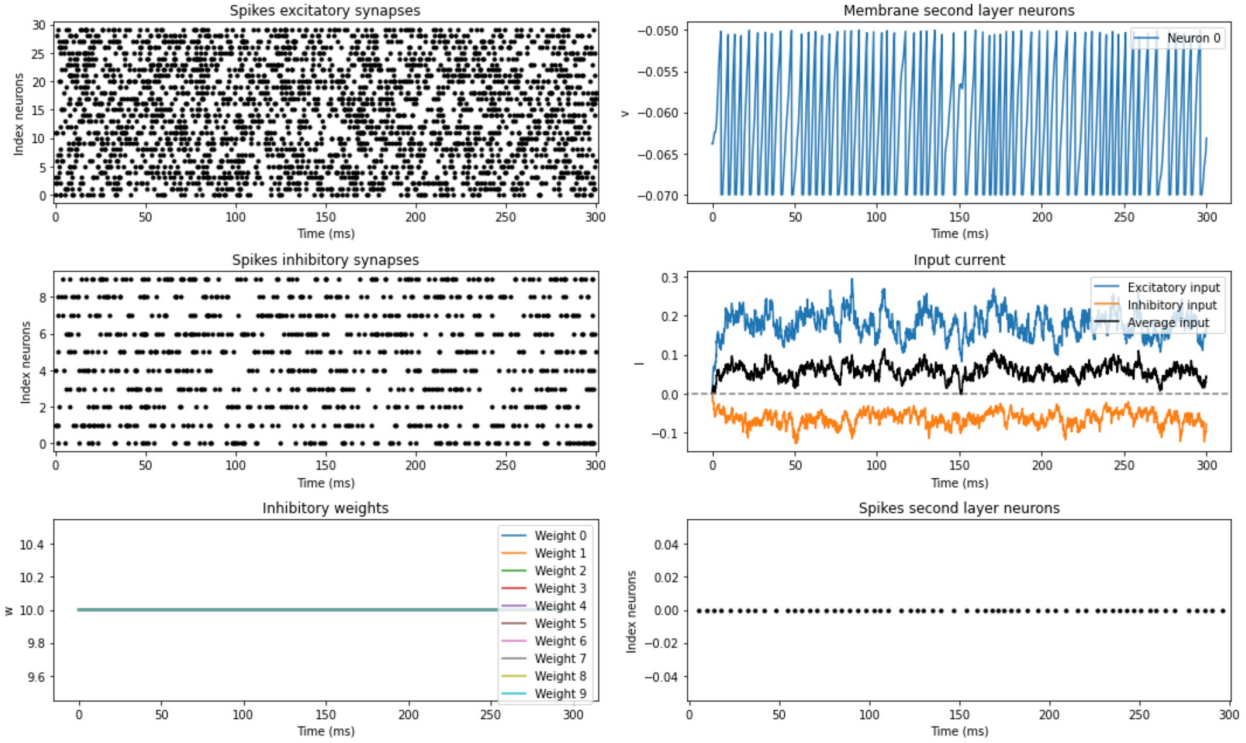
$$\tau \frac{\partial X_i}{\partial t} = -X_i$$

Vogels, Tim P., et al. "Inhibitory plasticity balances excitation and inhibition in sensory pathways and memory networks." *Science* 334.6062 (2011): 1569-1573.

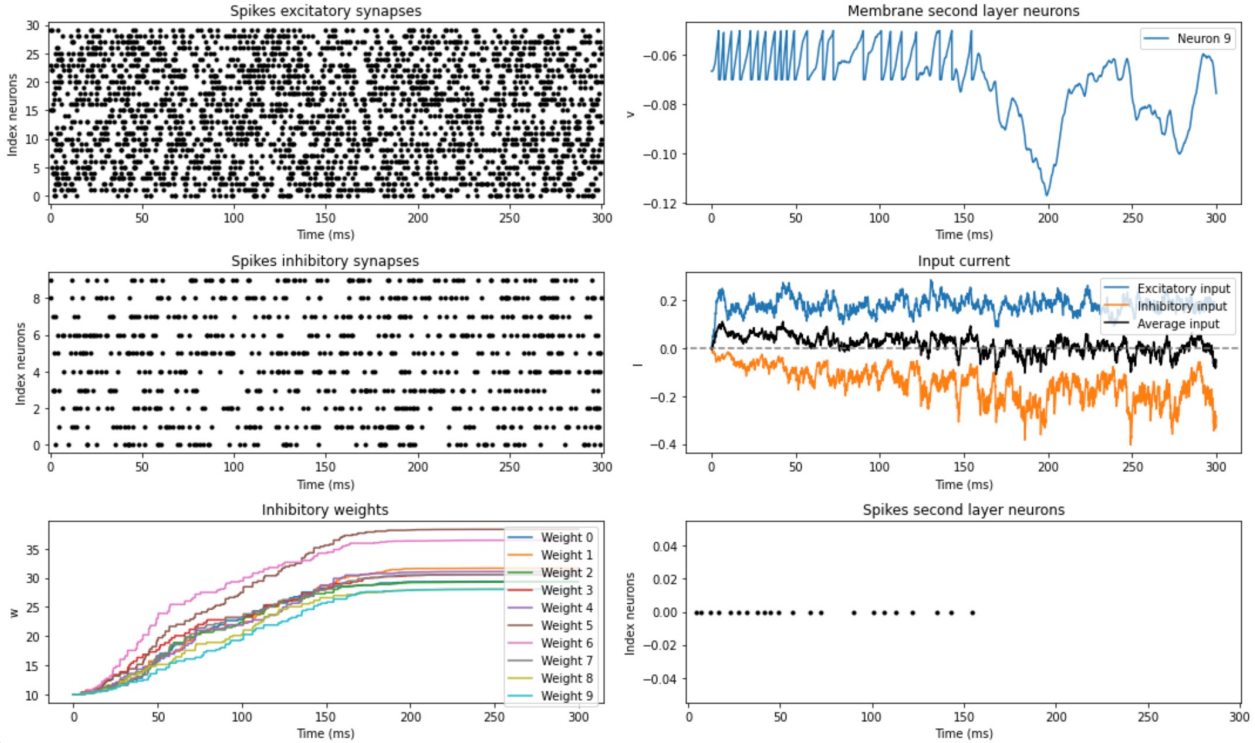
Balance excitation/inhibition

(Poisson input spikes)

No learning



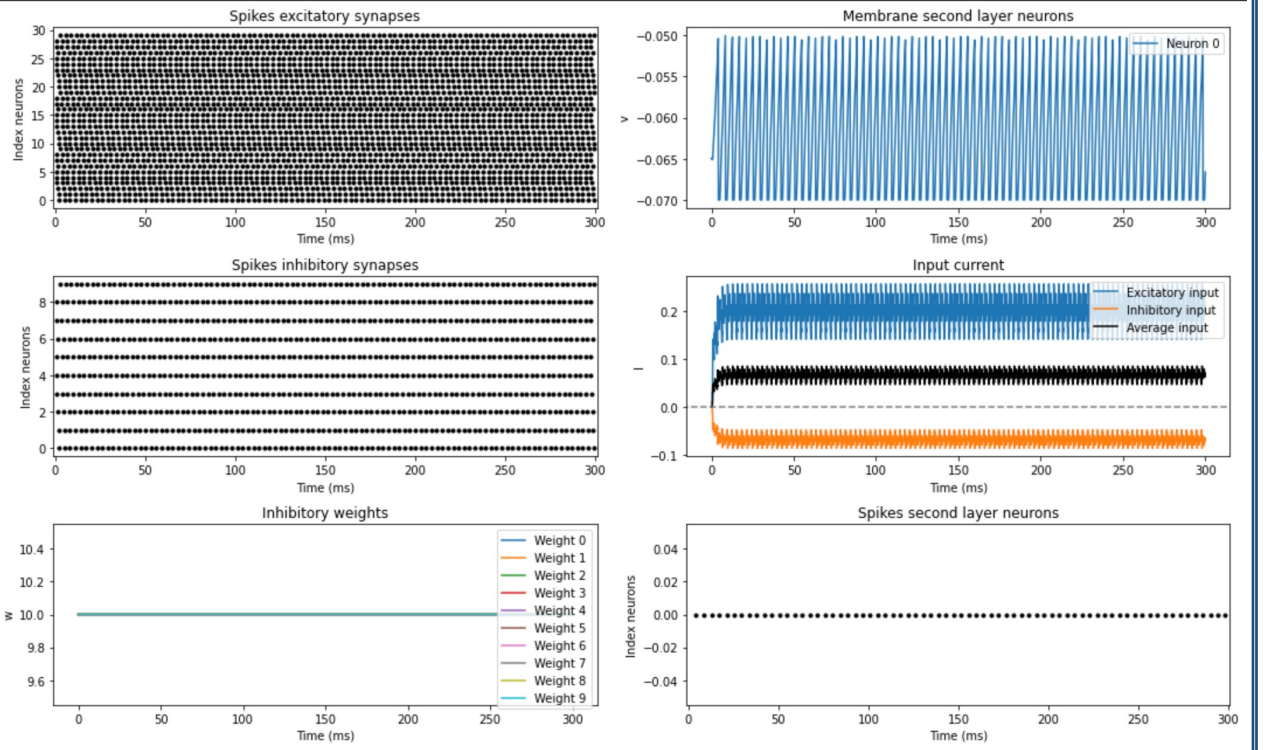
Learned weights



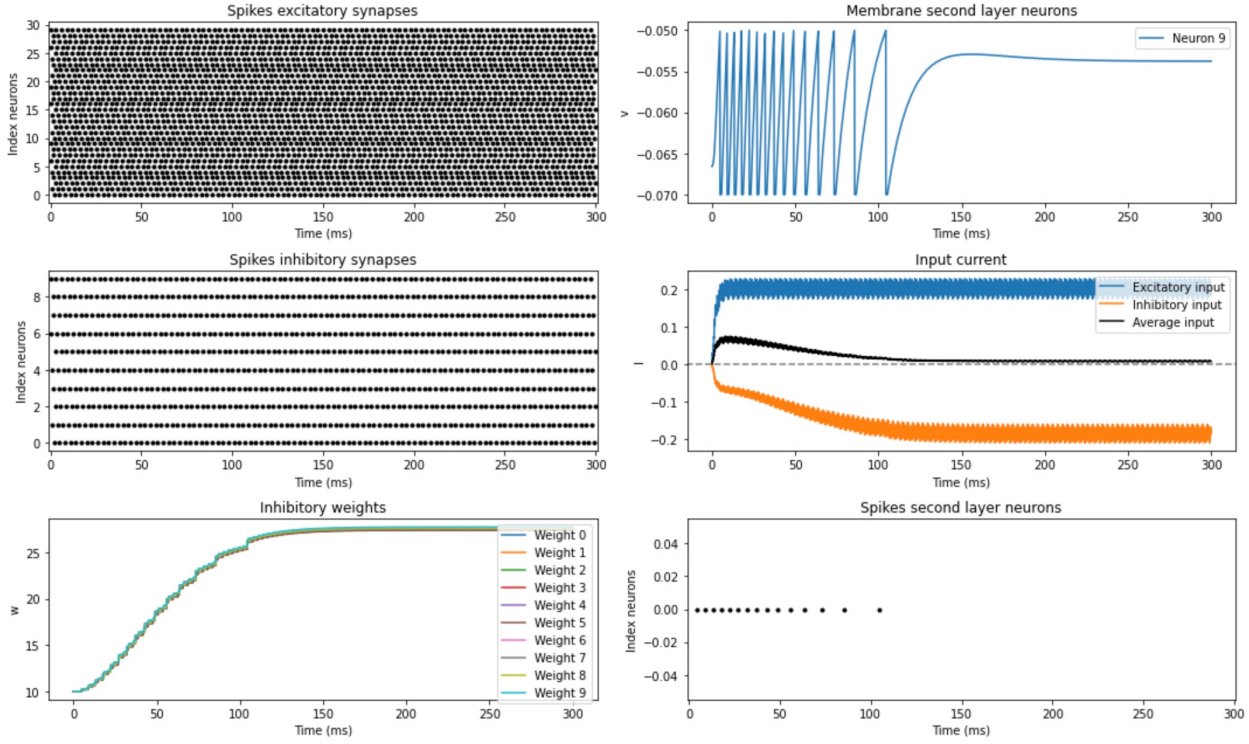
Balance excitation/inhibition

(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

Complex models

Example complex synapses

- *Here I'll quickly show the network I'm working on*
  - *Inhibitory synapses to decorrelate inputs*
  - *Feedforward synapses to learn transformations*
- *I'll give some personal insight and mention a few struggles I found with Brian2GeNN*

Not supported features Brian2GeNN

- 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 still some more but these are the ones everyone will probably face at some point

https://brian2genn.readthedocs.io/en/stable/introduction/exclusions.html

Multicompartment neurons

- *Sander's part*