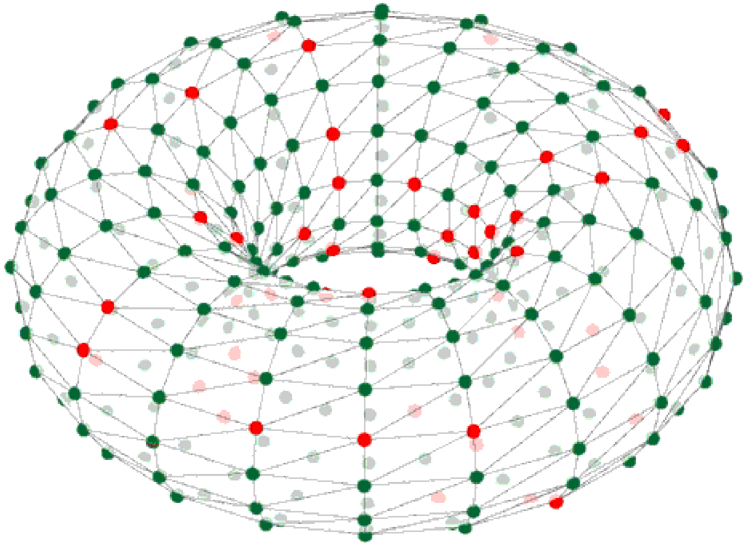
Synaptic plasticity on SpiNNaker with PyNN



Sergio Davies

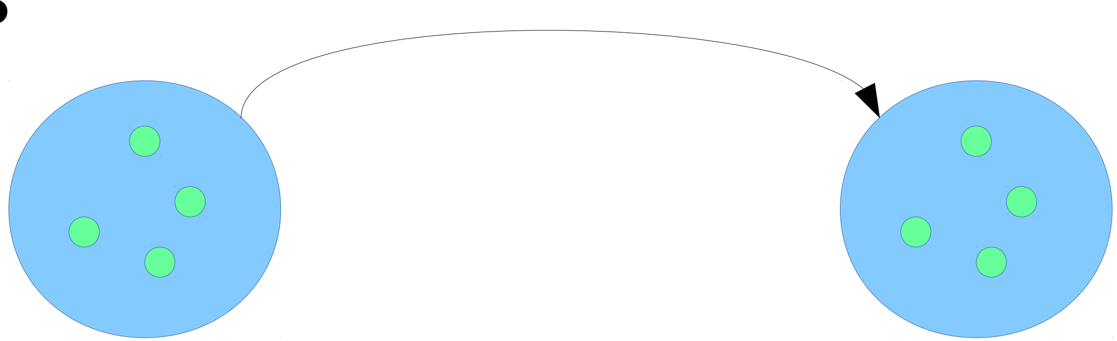
SpiNNaker Workshop
September 2015

Neural network description

A neural network is usually described in terms of:

- Populations of neurons
- Projections between populations

Each population and projection has its own properties



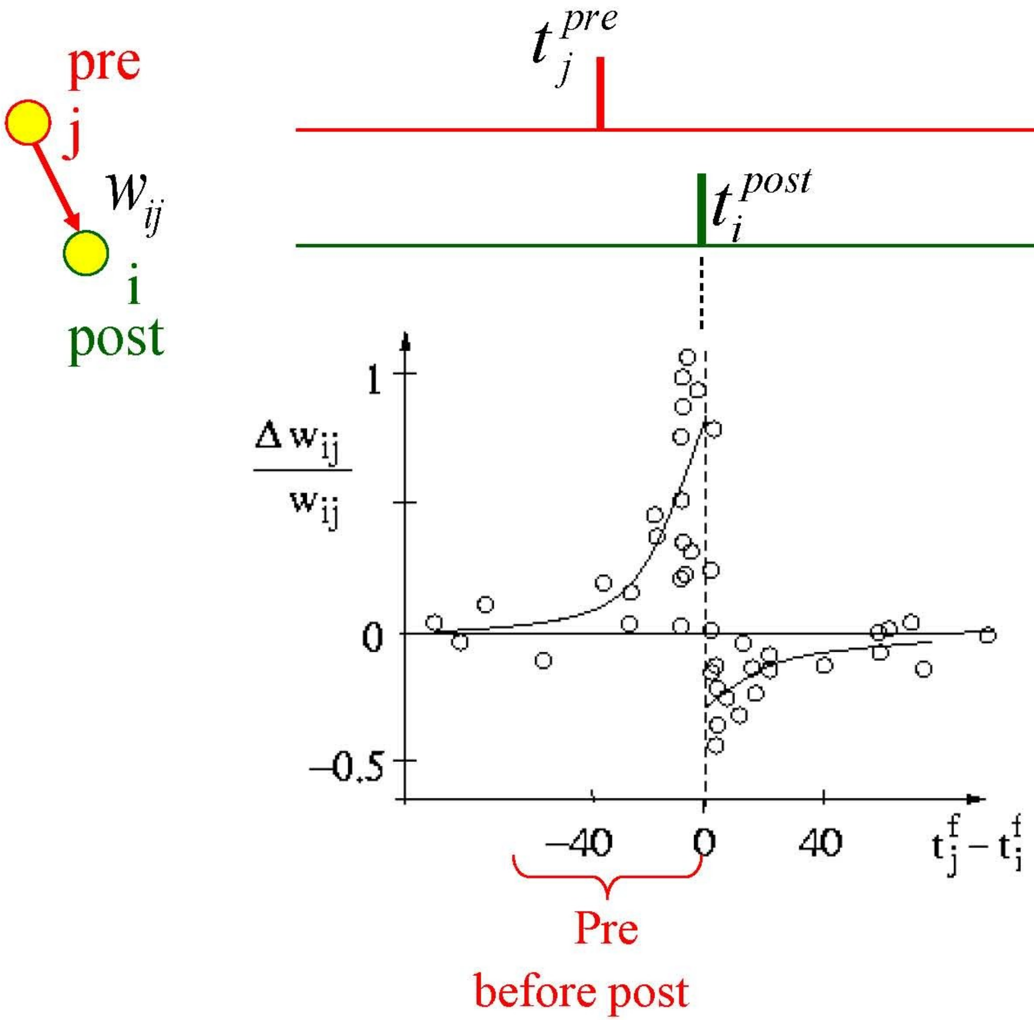
Projections

Attributes of the projections include (but are not limited to):

- Pre-synaptic population
- Post-synaptic population
- Connector type (All-To-All, One-To-One, etc.)
- Target – Synapse type (Excitatory, Inhibitory)
- Static or Dynamic attributes

Plasticity rules (1/2)

STDP spike pair rule



[Sjöström and Gerstner, 2010]

Plasticity rules (2/2)

Other rules available on SpiNNaker in "sPyNNakerExtraModelsPlugin" module:

- Vogels *[Vogels et al. (2011)]*
- Triplet-based rule *[J.-P. Pfister et al. (2006)]*
- *More...*

Behaviour dependence

Behaviour of plasticity rules may depend on one or more parameters:

- Weight dependence
- Time dependence

Weight dependence

- Additive weight dependence
- Multiplicative weight dependence

Weight Dependence example:

```
PyNN.AdditiveWeightDependence
(w_min, w_max, A_plus, A_minus)
```

Timing dependence

- SpikePairRule
- Vogels2011Rule
- Etc.

Timing Dependence example:

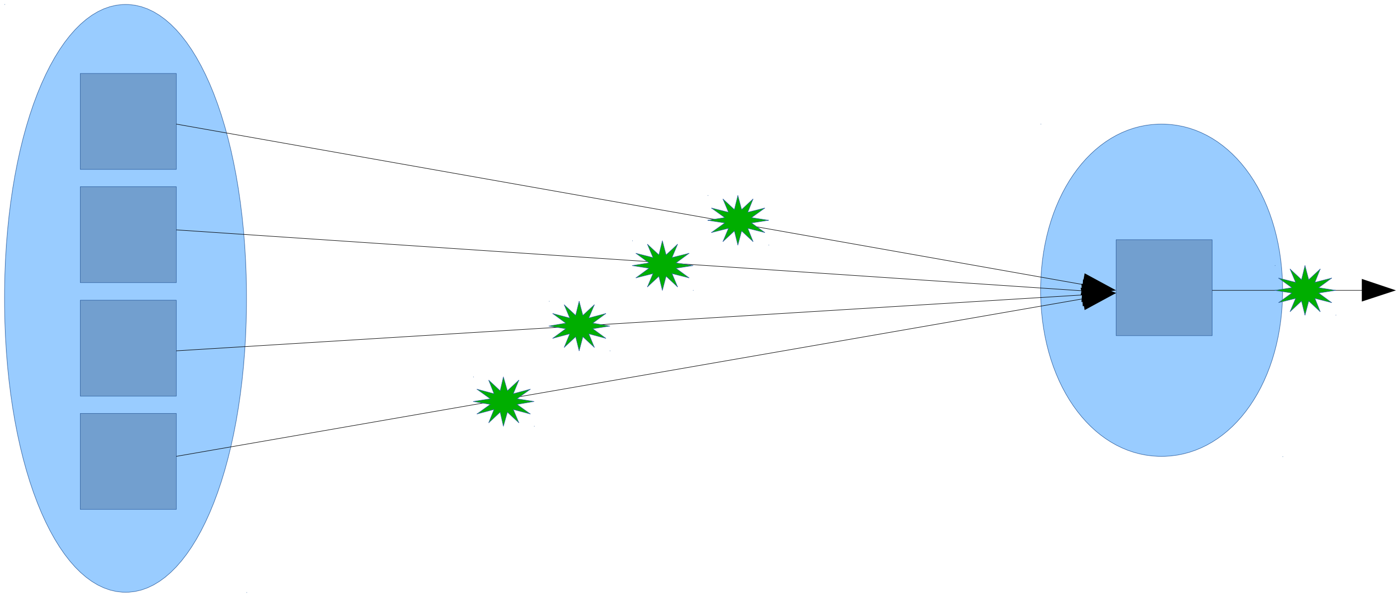
```
PyNN.SpikePairRule
(tau_plus, tau_minus, nearest)
```

Example

- Definition of a learning rule:

```
time_rule = SpikePairRule(tau_plus=1, tau_minus=1)

weight_rule = AdditiveWeightDependence(
        w_min=0.0, w_max=2, A_plus=0.5, A_minus=0.5)

stdp_model = STDPMechanism(
    timing_dependence = time_rule,
    weight_dependence = weight_rule)

syn_dyn = SynapseDynamics(slow = stdp_model)

Projection(pop_src, pop_dst, p.AllToAllConnector(weights,
delays), syn_dyn)
```

Example



Synapses with long-term plasticity

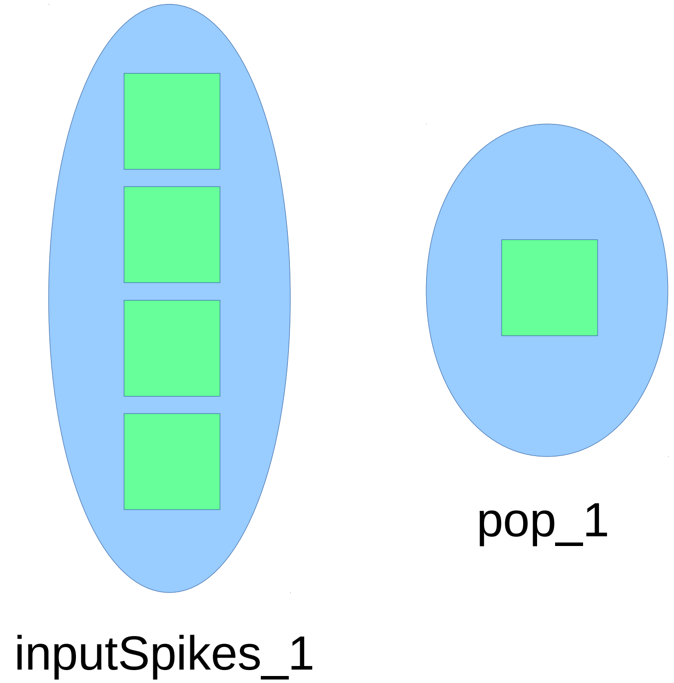
MANCHESTER 1824

Building the network – 1

```
import pyNN.spiNNaker as p
import numpy

p.setup(timestep=1.0, min_delay =
1.0, max_delay = 16.0)

end_time = 1100

cell_params_lif = {
    'cm'         : 0.25, # nF
    'i_offset'   : 0.0,
    'tau_m'      : 20.0,
    'tau_refrac': 2.0,
    'tau_syn_E' : 5.0,
    'tau_syn_I' : 5.0,
    'v_reset'    : -70.0,
    'v_rest'     : -65.0,
    'v_thresh'   : -50.0
    }

SpikeArray = {
    'spike_times':
    [range(0,end_time,50),
     range(3,end_time,50),
     range(6,end_time,50),
     range(9,end_time,50)]}

populations = list()
projections = list()
```

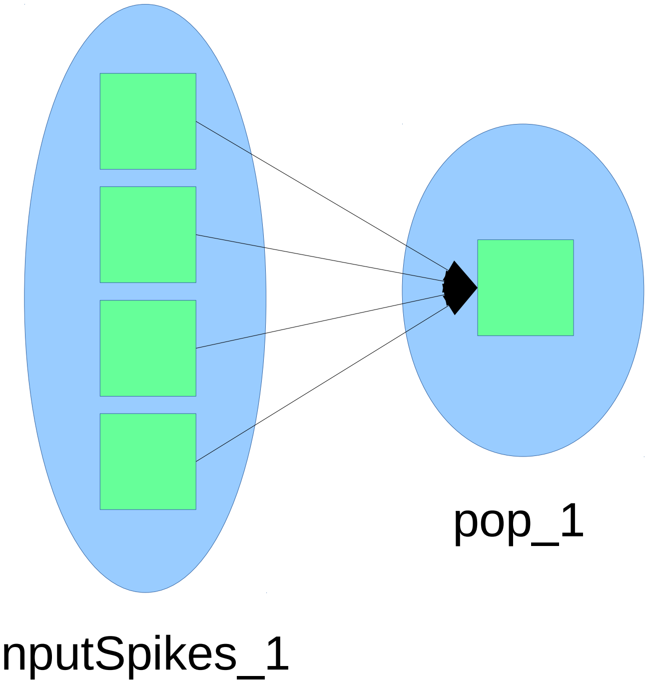
Building the network – 2

```
p.Population(
    4,
    p.SpikeSourceArray,
    spikeArray,
    label='inputSpikes_1')

p.Population(
    1,
    p.IF_curr_exp,
    cell_params_lif,
    label='pop_1')
```



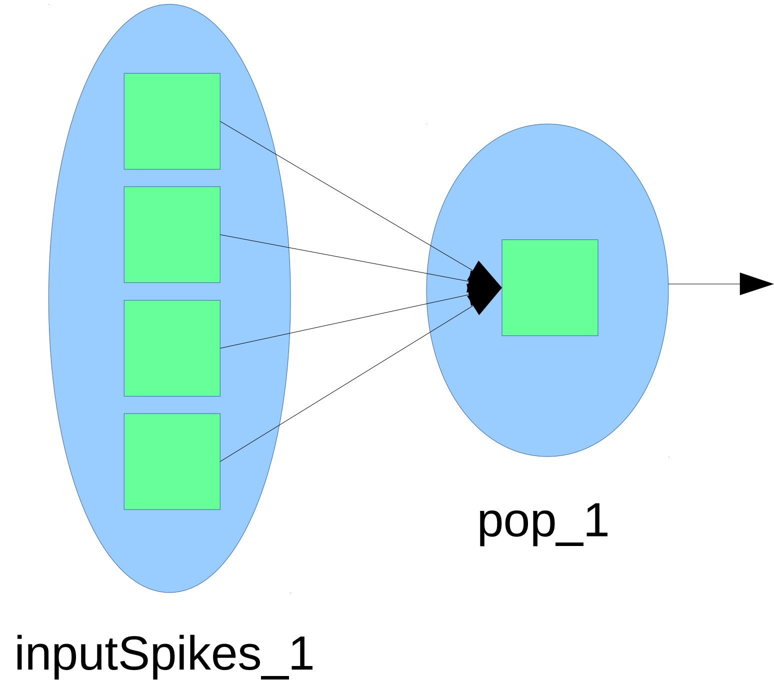
Building the network - 3

```
t_rule = p.SpikePairRule(
    tau_plus=1, tau_minus=1,
    nearest=True)
w_rule = p.AdditiveWeightDependence(
    w_min=0.0,  w_max=2,
    A_plus=0.5, A_minus=0.5)

stdp_model = p.STDPMechanism(
    timing_dependence = t_rule,
    weight_dependence = w_rule,
)

p.Projection(
    populations[0],
    populations[1],
    p.AllToAllConnector(
        weights = weight_to_spike,
        delays = delay),
    synapse_dynamics =
        p.SynapseDynamics(
            slow = stdp_model)))
```

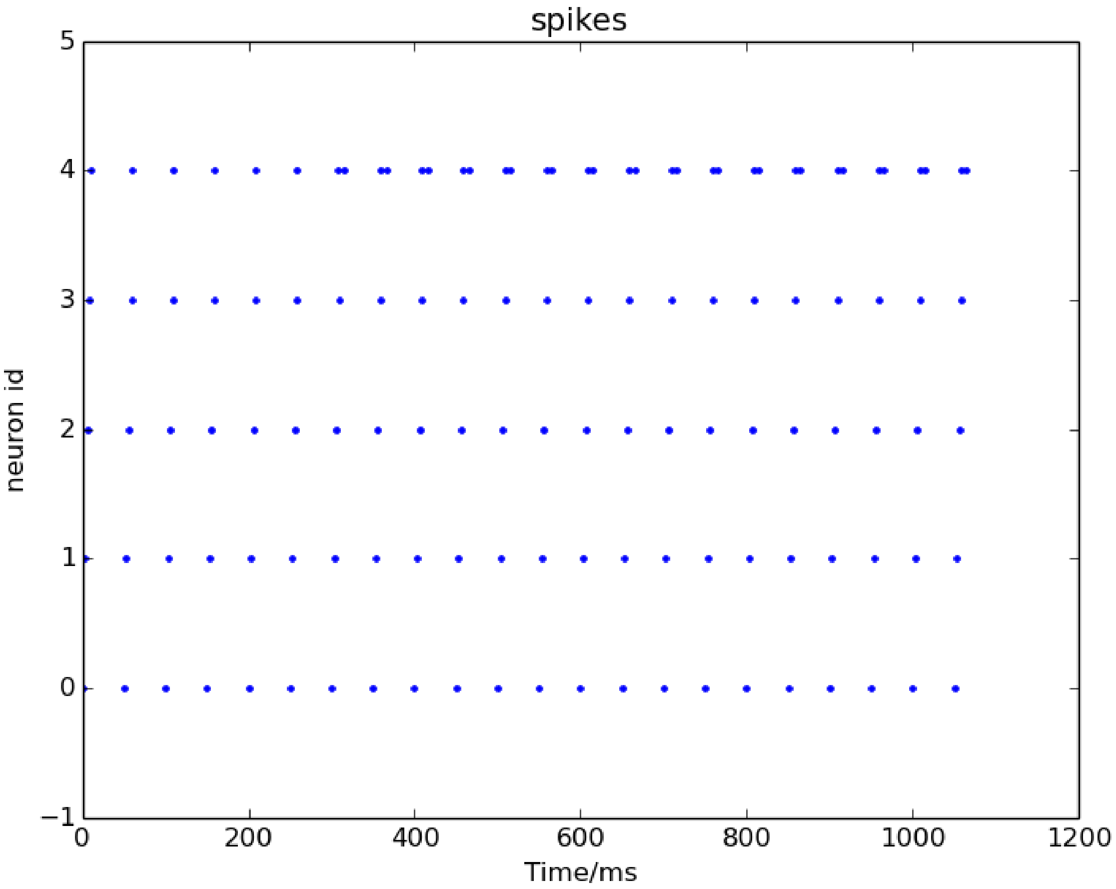


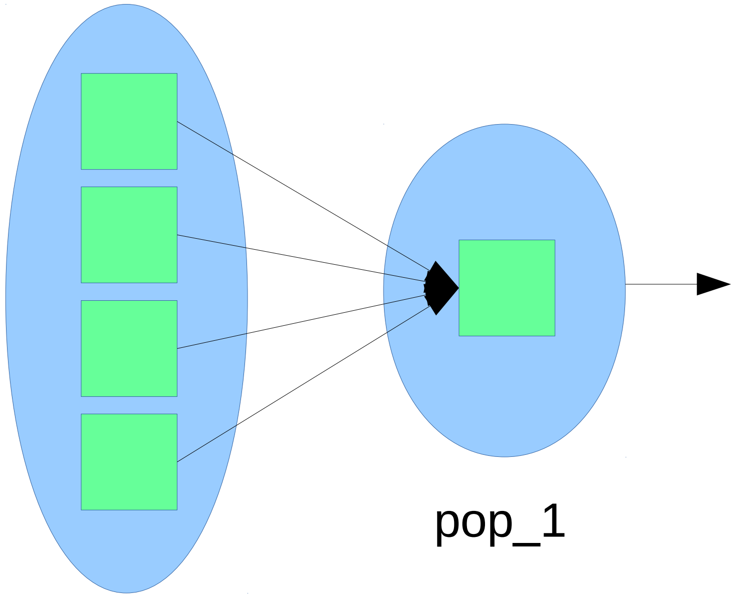
Running the network

```
populations[1].record()

p.run(end_time)

spikes_2 = populations[1].getSpikes()
```



Results – 1


spikes
neuron id
Time/ms
5
4
3
2
1
0
−1
0
200
400
600
800
1000
1200


pop_1
inputSpikes_1

Results – 2

Evolution of the synaptic weights: