Project 3 Report
Team: N. Auer, A. Matory, S. Salehi
Supervisor: Greg Knoll

1. Motivation of Model

For decades, computational neuroscientists attempted to create a model capable of producing asynchronous and irregular neural firing, resembling neural dynamics in many sensory systems. In 2011, Vogels et al. had a crowning achievement in modelling this type of firing. Implementing a spike-timing dependent plasticity (STDP) rule on a network of 10,000 leaky integrate-and-fire neurons, the research group created a model that produced neural firing with very similar asynchrony and irregularity to experimental data from mammalian cortical pyramidal neuron assemblies (Xue, Atallah, & Scansziani, Nature 2014). In these neuron assemblies, plastic inhibitory neurons learn to regulate the behavior of excitatory neurons. This has clear biological advantages. Given a network of excitatory neurons with the same preferred stimulus, coupled inhibitory neurons can ensure the excitatory neurons' potentials vary in potential from the resting potential at any given point in time. This type of network allows for consistently faster network response to the preferred stimulus, as opposed to a network who neurons that fire in synchrony and regularly.

Per Vogels et al., 2011, this network model is capable of "mimic[king] activity patterns observed in cortical neurons. Such asynchronous network states facilitate rapid responses to small changes in the input, providing an ideal substrate for cortical signal processing."

2. Implementation of Model

The basis of the model involves two equations: a leaky integrate-and-fire neuron model and STDP. Our network is comprised of 10,000 integrate-and-fire neurons (eq. 1), each complete with an excitatory, inhibitory, leak, and injected current. Our model has plastic inhibitory synapses and fixed excitatory synapses. The excitatory synapses are given a preferred signal. Our STDP learning rule (eq. 2) dictates that the weights of our inhibitory interneuron synapses increase in the event of near-coincident pre- and post-synaptic spikes, with the weight increase decaying exponentially as the time between the pre- and post-synaptic spikes increase.

$$\tau \frac{dV}{dt} = (V_{rest} - V) + (g_E(V_E - V) + g_I(V_I - V) + I_e) \cdot \frac{1}{g_{leak}} \qquad (\text{eq. } 1)$$

$$\Delta w = \eta(pre \times post - \rho_0 \times pre) \qquad (eq. 2)$$

Due to the weights we set between our two different types neuron populations, upon initialization, the network's excitatory neuron population experienced much more synchronous, regular, and higher firing rates among our excitatory neuron population vs. our inhibitory neuron population *(Fig. 1)*. Here, the weights from inhibitory to inhibitory, excitatory to excitatory, and excitatory to inhibitory are equal. The weight from inhibitory to excitatory neurons, however, is a magnitude of 1e10 lower than the connection the other weights. So, after initialization, we saw the inhibitory neurons inhibit each other much more than inhibiting the excitatory neurons, allowing for the excitatory neurons to fire at a much higher rate.

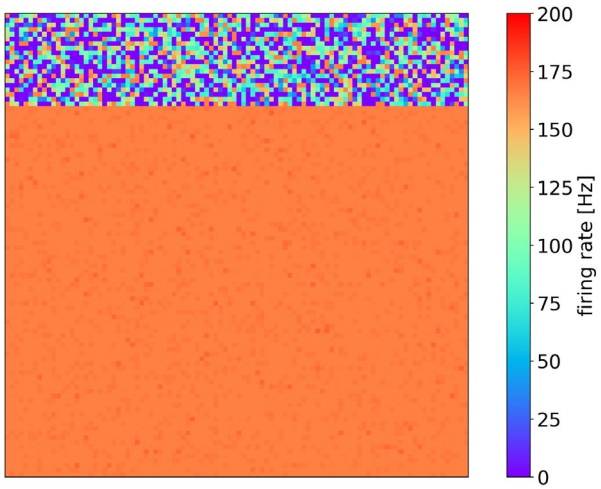


Figure 1. (Top) Population of 2000 inhibitory neurons and (bottom) 8000 excitatory neuron upon initialization of network.

A single neuron experienced an action potential when the net current it received exceeded a threshold voltage of -50 mV *(Fig 2)*.

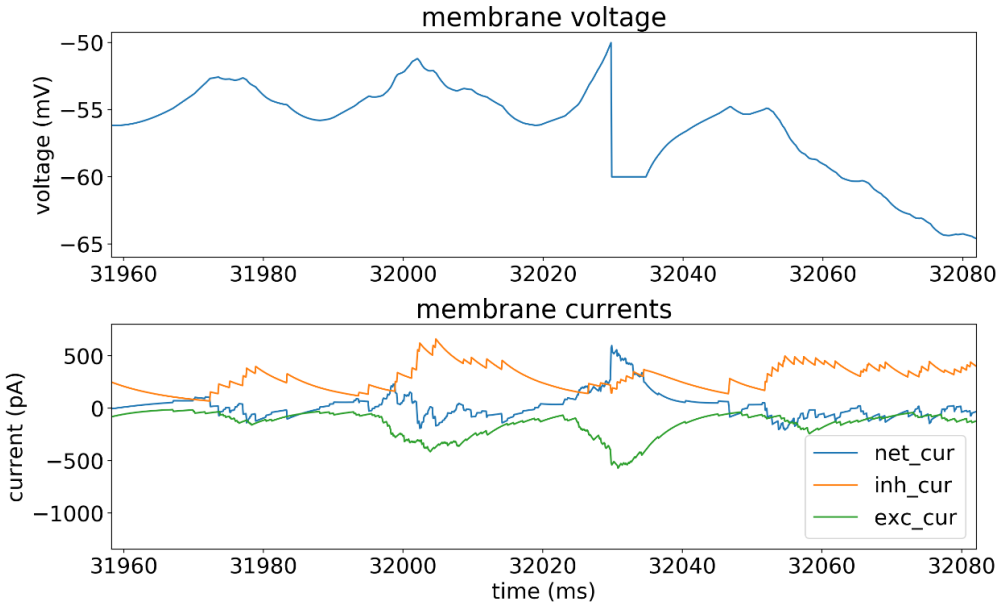


Figure 2. (Top) Voltage trace of a single neuron. (Bottom) Interplay of the excitatory and inhibitory currents, showing an action potential at ~32030ms when the net current exceeds a threshold value.

3. Explanation of STDP

To implement STDP, we took the product of the presynaptic and postsynaptic activity, and from it, subtracted the product of the presynaptic activity and a target rate for the postsynaptic activity (maintaining a global average firing rate, without which, neurons fire at much higher rates). Multiplying this by a learning rate, we had our weight change. With this weight change rule, our inhibitory interneurons learned respond to increased postsynaptic activity, seemingly indifferent to the tuning curve of the excitatory synapses, resembling results seen in experimental data from a number of sensory systems. This tuning curve indifference is a direct consequence of the correlation between excitatory and inhibitory synapses. The learning rule also accounts for a refractory period of synaptic depression after a postsynaptic neuron fires.

$$STDP\ rule\ in\ depth$$

In our model we used a learning rate that is two orders of magnitude larger than that used in the original paper. This allows us to see the effects of STDP much more quickly that Vogels et al. 2011, who ran the model over 180 virtual minutes.

Before STDP was implemented, there was relatively unregulated, synchronous and regular firing from out excitatory synapses. As time passed, our inhibitory synapse weights increased slowly, driving the minimum 'resting' potential of each neuron in our network lower *(Fig. 3)*.

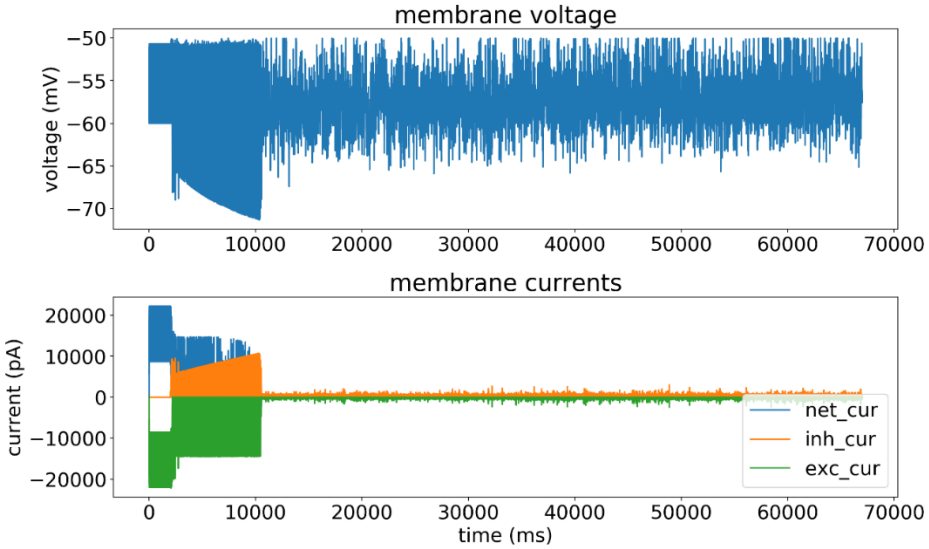


Figure 3. (Top) Single-cell membrane voltage and (bottom) inhibitory (orange), excitatory (green), and net (blue) currents recording from time of initialization to implementation of STDP (10,200ms), onward.

After our learning rule was implemented, near-coincident pre- and post-synaptic spikes motivated an increase in weight strength of all inhibitory synapses, until the network reaches our AI state *(Fig 4.)*, where inhibitory and excitatory synapses were roughly balanced. We observed the ramping up of the inhibitory current, as the synapses 'learned' (their weights increased) through near-coincident pre-/post-synaptic spiking.

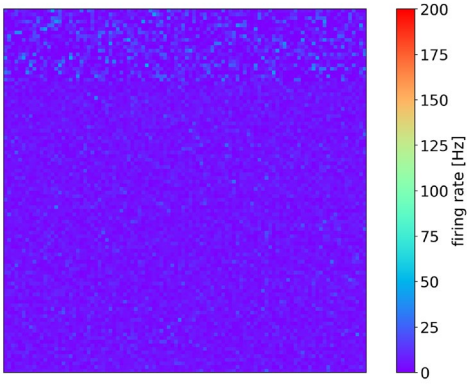


Figure 4. Network firing rates, after implementation of STDP learning rule

4. Synchrony and regularity measures

We determined the global state of our system by quantifying the synchrony and the regularity of the firing activity.

In order to quantify the regularity of the spiking pattern, we calculated the coefficient of variation (CV) of the interspike intervals (ISI). The coefficient of variation is the ratio of the standard deviation of a distribution and its mean:

$$\boldsymbol{CV_i = \frac{\sigma_i}{\mu_i}}\,.$$

For each neuron, we calculated the CV of the interspike intervals during a sufficiently large time period and plot the distribution in a histogram (*Fig. 4.a, 4.b*). The ISI CV of the whole network was given by the mean of all neurons' ISI CVs. CVs close to 0 describe a very regular system, while CVs near 1 indicate a more irregular firing pattern. ISI CVs larger than 1 can be interpreted as burstiness in the firing pattern.

The synchrony of the network activity was measured by the spiking correlations between neurons. The binary spike train $\boldsymbol{S_i(t)}$ of a neuron **i** was filtered by a symmetric, bi-exponential kernel $\boldsymbol{K(t)}$. We then computed the Pearson correlation coefficient $\boldsymbol{X_{ij}}$ between the neurons **i** and **j**, where $\boldsymbol{V_{ij}}$ is the unnormalized covariance over all times t.

$$\boldsymbol{S_i(t) = \sum_f \delta\left(t - t_i^f\right)} \text{ (binary spike train)}$$

$$\boldsymbol{K(t) = \frac{1}{50} exp\left(-\frac{|t|}{50}\right) - \frac{1}{200}\, exp\left(-\frac{|t|}{200}\right)} \text{ (symmetric, bi-exponential kernel)}$$

$$\boldsymbol{F_i(t) = S_i(t) * K(t)} \text{ (filtered spike train)}$$

$$\boldsymbol{V_{ij} = \sum_t F_i(t) F_j(t)} \text{ (unnormalized covariance)}$$

$$X_{ij} = \frac{V_{ij}}{\sqrt{V_{ii}V_{jj}}}$$ (Pearson correlation coefficient)

We calculated the spiking correlation between 300 randomly chosen neurons which leads to ~45,000 different correlation coefficients and we visualized the distribution of the spiking correlation coefficients in a histogram (*Fig. 4.c, 4.d*).

Together, ISI CV and spiking correlation provide a sound description of the network state in terms of regularity and synchrony.

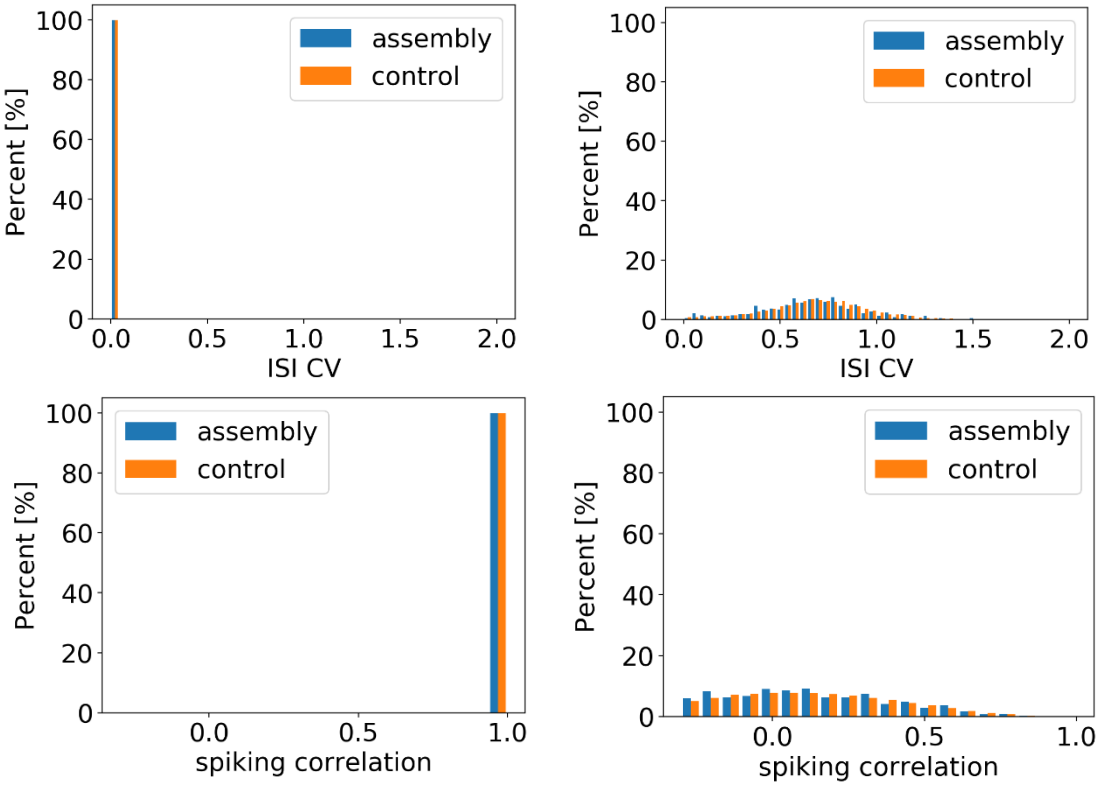


*Figure 5. Distributions of coefficients of variation of interspike intervals (ISI CVs) (upper row) and of spiking correlations between spike trains (bottom row) or pairs of neurons. The histograms in the left column correspond to a highly regular **(A)** (ISI CV near 0), synchronous **(C)** (spiking correlation near 1) firing pattern. Our network is in this state before the STDP is turned on. After the inhibitory synapses have been adapted using STDP, the network settles in an irregular **(B)** (ISI CV near 1), asynchronous **(D)** (spiking correlation near 0) state. The right column shows the histograms for this state. The distinction between assembly and control is the same as in Section 6 and,*

5. Network States

Upon initialization of our network, without any learning rule implemented for the inhibitory neurons, we observed a network state in which the excitatory neurons fire with high correlated and at very regular intervals, about every 5ms (the time constant for our excitatory neuron conductance). Once we implemented our STDP learning rule, we

observe a network that has many more irregular and asynchronously firing neurons, oscillating around a baseline amount of a neural activity (*Fig. 5*).

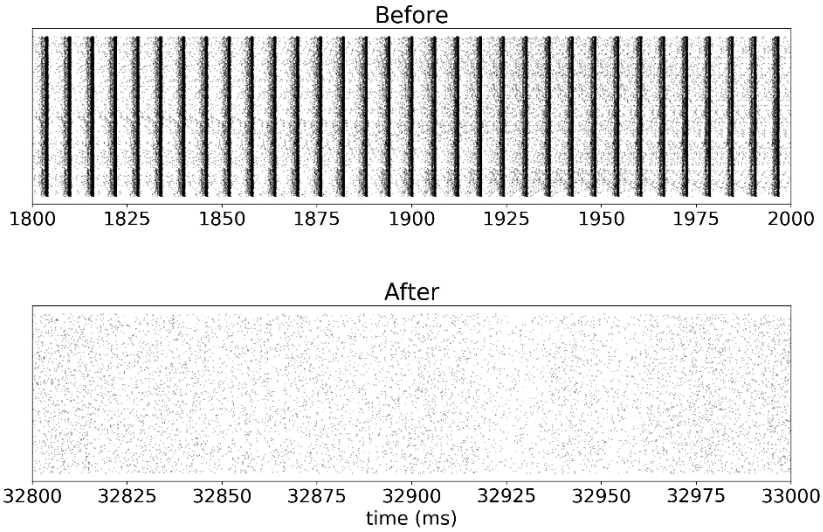


Figure 6. (Top) Raster plot of action potentials for network before implementation of STDP and (bottom) after implementation of STDP.

6. Activation of a Memory

This model even established a means to model the formation of memories, their activation, and subsequently deactivation. An assembly of excitatory neurons are connected with a firing probability, symbolizing the formation of a memory. Increased rates, synchrony, and regularity of firing of the assembly of neurons constituted the activation of the memory *(Fig 7.)* and a decrease in these three parameters, which caused them to approach the same values of other network neurons not included in the memory assembly, was the deactivation of the memory *(Fig 9.)*.

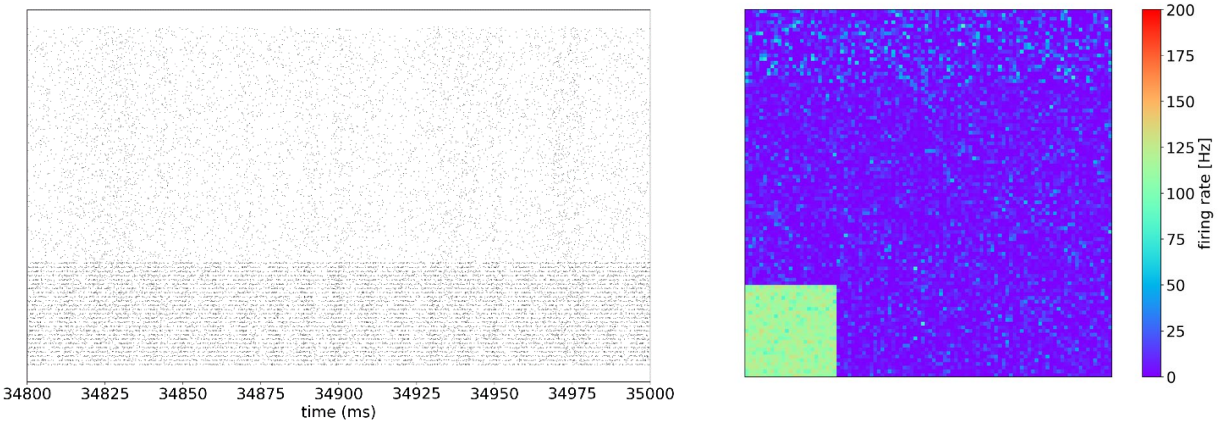


Figure 7. (Left) Network raster plot of action potentials and (right) firing rates after increased connected probability of assembly of neurons to .2.

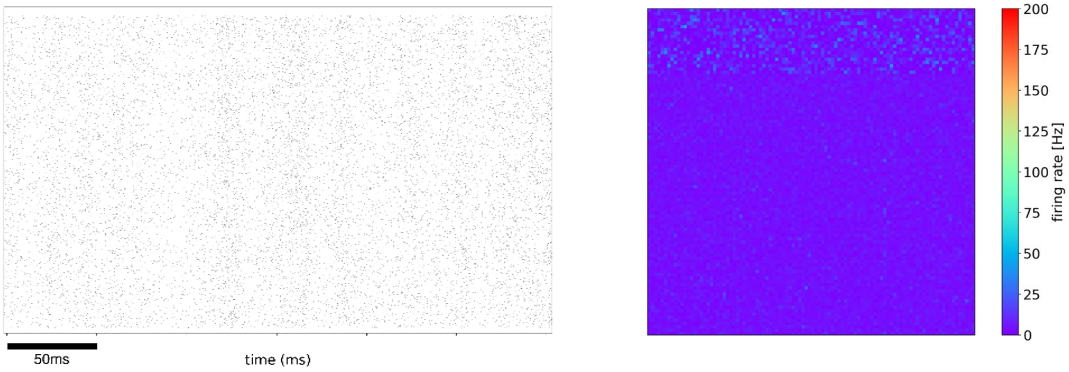


Figure 8. Return of the network to the AI state, after a memory was activated and subsequently deactivated, ~10 seconds after assembly connection.

In our model, after the STDP-implemented network settled, we connected an assembly of 625 excitatory neurons to each other with 2% probability. The probability-connected neuron assembly drove the assembly to more synchronous and regular firing rates *(Fig. 9a, 9b),* (vs. a group of control neurons whose connection probability wasn't changed) until the inhibitory neurons reestablished the global asynchronous irregular state of the network *(Fig. 8)*, where our ISI CV distribution approached 1 *(Fig. 9.b)* and our spiking correlation approached 0 *(Fig 9.d).* We saw an increase in not only the ISI CV, correlation, and firing rates between the assembly after time of connection, but also an increase in the background firing rates of the network.

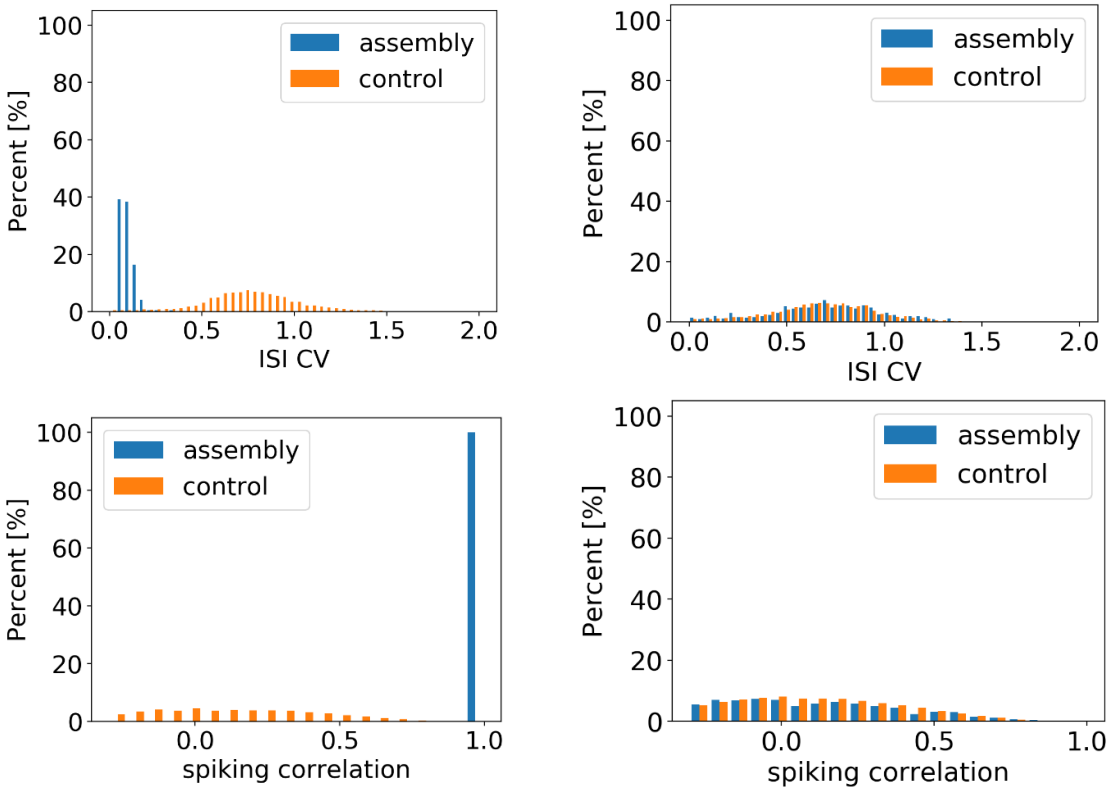


Figure 9. Measures of regularity and synchrony (left column) immediately after the assembly has been connected and (right column) after the network rebalances are returns to a network state.

7. Discussion

This model was capable of producing variability of firing in neural assemblies using a deterministic network without added noise. The model's main achievement is its use of the STDP leaning rule to achieve an AI state.

The model showed further value by providing a theoretical explanation for memory formation, activation, and deactivation. It was shown that an assembly of neurons whose connection increases can represent a memory and that implementing an STDP rule on inhibitory neurons can provide a means by which this memory becomes deactivated after activation.

AI_Network_Project3_V4_2.11.19

February 11, 2019

In [1]:
```
from brian2 import *
import numpy as np

rcParams.update({'font.size': 20})

def ISI_CV(trains, t_min, t_max):
    # number of neurons
    num_neu = len(trains)
    # ISI mean values for each neuron
    isi_mu1 = () * second
    isi_mu2 = () * second
    # ISI standard deviation for each neuron
    isi_std1 = () * second
    isi_std2 = () * second
    for idx in range(num_neu):
        # calc. ISIs for times between t_min and t_max
        train = diff(trains[idx][argmax(trains[idx] > t_min * second):argmax(trains[idx
        if len(train) > 1:
            if idx % 100 < 25 and idx < 2500:
                # calc. mean
                isi_mu1 = append(isi_mu1, mean(train))
                # calc. standard deviation
                isi_std1 = append(isi_std1, std(train))
            else:
                # calc. mean
                isi_mu2 = append(isi_mu2, mean(train))
                # calc. standard deviation
                isi_std2 = append(isi_std2, std(train))

    # calc. ISI CV for every neuron
    CV1 = isi_std1 / isi_mu1
    CV2 = isi_std2 / isi_mu2
    # calc. mean of all ISI CVs
    cv1 = mean(CV1)
    cv2 = mean(CV2)

    # plot histograms
```

```
    gca().set_ylim([0, 105])
    hist([CV1, CV2], bins=10, range=[0, 2],
             weights=[np.ones(len(CV1)) / len(CV1) * 100., np.ones(len(CV2)) / len(CV2)
             label=['assembly', 'control'])  # arguments are passed to np.histogram
    xlabel('ISI CV')
    ylabel('Percent [%]')
    # title('ISI CV distribution ')
    legend()
    show()
    return cv1, cv2


def spiking_corr(trains, t_min, t_max, chosen_neu):
    print('please wait...')
    # symmetric bi-exponential kernel
    TK = np.arange(-1, 1, 0.001)
    K = ((1 / (50 * ms)) * exp(-np.abs(TK * 1000) / (50))) - ((1 / (200 * ms)) * exp(-

    # array of spiking correlations in assembly and control group
    X1 = ()
    X2 = ()

    for i in chosen_neu:
        for j in chosen_neu:
            if i != j:
                # calc. only for times between t_min and t_max
                trains_short_i = trains[i][argmax(trains[i] >= t_min * second):argmax(t
                trains_short_j = trains[j][argmax(trains[j] >= t_min * second):argmax(t
                T = np.arange(t_min, t_max, 0.001)
                spikes_i = np.zeros(len(T))
                for k in range(len(trains_short_i)):
                    spikes_i[int((trains_short_i[k] / second - t_min) * 1000)] = 1
                spikes_j = np.zeros(len(T))
                for k in range(len(trains_short_j)):
                    spikes_j[int((trains_short_j[k] / second - t_min) * 1000)] = 1
                # convolutions with kernel
                F_i = convolve(spikes_i, K)
                F_j = convolve(spikes_j, K)
                # check if in assembly
                if i % 100 < 25 and j % 100 < 25 and i < 2500 and j < 2500:
                    X1 = append(X1, corrcoef(F_i, F_j)[0, 1])
                elif (i % 100 >= 25 or i > 2500) and (j % 100 >= 25 or j > 2500):
                    X2 = append(X2, corrcoef(F_i, F_j)[0, 1])

    # plot histograms
    figure()
    gca().set_ylim([0, 105])
    hist([X1, X2], bins=10, range=[-0.3, 1],
```

```
                    weights=[np.ones(len(X1)) / len(X1) * 100., np.ones(len(X2)) / len(X2) * 1
                    label=['assembly', 'control'])
        xlabel('spiking correlation')
        ylabel('Percent [%]')
        # title('spiking correlation distribution')
        legend()
        show()

        X1 = X1[where(logical_not(isnan(X1)))]
        X2 = X2[where(logical_not(isnan(X2)))]
        return mean(X1), mean(X2)
```

In [2]:
```
# ##########################################
# defining the simulation episodes
# ##########################################
sim_episodes = {'init': 1,
                'no_learning': 5,
                'set_learning': 14,
                'asyn_irg': 5,
                'pre_assembly': 1,
                'assembly': 5,
                'wait_rebalance': 14,
                'post_rebalance': 6}

# just adding times up
sim_times = {'init': 1,
             'no_learning': 6,
             'set_learning': 20,
             'asyn_irg': 25,
             'pre_assembly': 26,
             'assembly': 31,
             'wait_rebalance': 45,
             'post_rebalance': 50}

# ##########################################
# Defining network model parameters
# ##########################################

NE = 8000           # Number of excitatory cells
NI = int(NE/4)          # Number of inhibitory cells

tau_ampa = 5.0*ms    # Glutamatergic synaptic time constant
tau_gaba = 10.0*ms   # GABAergic synaptic time constant
epsilon = 0.02       # Sparseness of synaptic connections

tau_stdp = 20*ms     # STDP time constant

# defaultclock.dt = 0.2*ms
```

```
# ##########################################
# Neuron model
# ##########################################

gl = 10.0*nsiemens   # Leak conductance
el = -60*mV          # Resting potential
er = -80*mV          # Inhibitory reversal potential
vt = -50.*mV         # Spiking threshold
memc = 200.0*pfarad  # Membrane capacitance
bgcurrent = 200*pA   # External current

eqs_neurons='''
dv/dt = I_n/memc : volt (unless refractory)
I_n = (-gl*(v-el)-(I_e+I_i)+bgcurrent) : amp
I_i = g_gaba*(v-er) : amp
I_e = g_ampa*v : amp
dg_ampa/dt = -g_ampa/tau_ampa : siemens
dg_gaba/dt = -g_gaba/tau_gaba : siemens
'''

# ##########################################
# Initialize neuron group
# ##########################################

neurons = NeuronGroup(NE+NI, model=eqs_neurons, threshold='v > vt',
                      reset='v=el', refractory=5*ms, method='euler')
neurons.v = np.random.randint(-60,-50,10000)*mV

Pe = neurons[:NE]
Pi = neurons[NE:]

# ##########################################
# Connecting the network
# ##########################################

con_e = Synapses(Pe, neurons, on_pre='g_ampa += 0.3*nS')
con_e.connect(p=epsilon)
con_ii = Synapses(Pi, Pi, on_pre='g_gaba += 3*nS')
con_ii.connect(p=epsilon)

# ##########################################
# Inhibitory Plasticity
# ##########################################

eqs_stdp_inhib = '''
w : 1
dApre/dt=-Apre/tau_stdp : 1 (event-driven)
```

```
        dApost/dt=-Apost/tau_stdp : 1 (event-driven)
        '''
        alpha = 3*Hz*tau_stdp*2  # Target rate parameter
        gmax = 100               # Maximum inhibitory weight

        con_ie = Synapses(Pi, Pe, model=eqs_stdp_inhib,
                          on_pre='''Apre += 1.
                                 w = clip(w+(Apost-alpha)*eta, 0, gmax)
                                 g_gaba += w*nS''',
                          on_post='''Apost += 1.
                                  w = clip(w+Apre*eta, 0, gmax)
                                ''')
        con_ie.connect(p=epsilon)
        con_ie.w = 1e-10

        # ###########################################
        # Setting up monitors
        # ###########################################

        I_rec_e = StateMonitor(Pe[0:1], ('v', 'I_i', 'I_e', 'I_n'), record=[0])
        S_rec_e = SpikeMonitor(Pe[0:1])
        I_rec_i = StateMonitor(Pi[0:1], ('v', 'I_i', 'I_e', 'I_n'), record=[0])
        S_rec_i = SpikeMonitor(Pi[0:1])
        SM_e = SpikeMonitor(Pe)
        SM_i = SpikeMonitor(Pi)
```

In [3]:
```
        # ###########################################
        # Run without plasticity
        # ###########################################
        eta = 0          # Learning rate

        # free run for the system to settle down (initialize)
        run(sim_episodes['init']*second,report='text')

        SM_n_time = SpikeMonitor(neurons, record=False)
        run(sim_episodes['no_learning']*second,report='text')
        count_0_1 = np.array(SM_n_time.count).reshape((100,100))
        heat_no_learning = count_0_1 / sim_episodes['no_learning']

        # ###########################################
        # Run with plasticity
        # ###########################################
        eta = 1e-2           # Learning rate
        run(sim_episodes['set_learning']*second, report='text')
        count_0_2 = np.array(SM_n_time.count).reshape((100,100))

        # ###########################################
        # Run after plasticity
```

```
# ##########################################
run(sim_episodes['asyn_irg']*second, report='text')
count_0_3 = np.array(SM_n_time.count).reshape((100,100))
heat_asyn_irg = (count_0_3 - count_0_2) / sim_episodes['asyn_irg']

# ##########################################
# turn off plasticity
# ##########################################
eta = 0          # Learning rate
run(sim_episodes['pre_assembly']*second, report='text')
count_0_4 = np.array(SM_n_time.count).reshape((100,100))
heat_reset_learning = (count_0_4 - count_0_3) / sim_episodes['pre_assembly']

# ##########################################
# change the connection probability
# ##########################################
p_rc = 0.1

# adding new connections between 625 randomly chosen neurons
con_e.connect(condition='i%100<25 and j%100<25 and i<2500 and j<2500 and i!=j',
              p=p_rc)

run(sim_episodes['assembly']*second,report='text')
count_0_5 = np.array(SM_n_time.count).reshape((100,100))
heat_change_connect = (count_0_5 - count_0_4) / sim_episodes['assembly']

# ##########################################
# run after change in connection probability
# ##########################################
eta = 1e-2

run(sim_episodes['wait_rebalance']*second,report='text')
count_0_6 = np.array(SM_n_time.count).reshape((100,100))

# ##########################################
# asymmetry returned
# ##########################################
run(sim_episodes['post_rebalance']*second,report='text')
count_0_7 = np.array(SM_n_time.count).reshape((100,100))
heat_post_rebalance = (count_0_7 - count_0_6) / sim_episodes['post_rebalance']

print(defaultclock.dt)
```

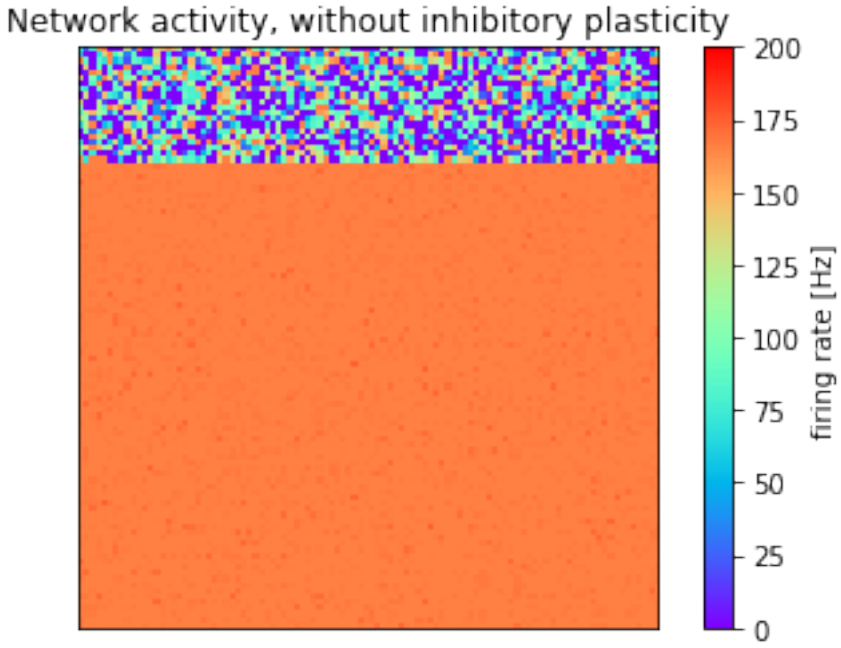
```
Starting simulation at t=0. s for a duration of 1. s
1.0 (100%) simulated in 7s
Starting simulation at t=1. s for a duration of 5. s
0.7544166666666667 (15%) simulated in 10s, estimated 56s remaining.
1.6532500000000003 (33%) simulated in 20s, estimated 40s remaining.
```

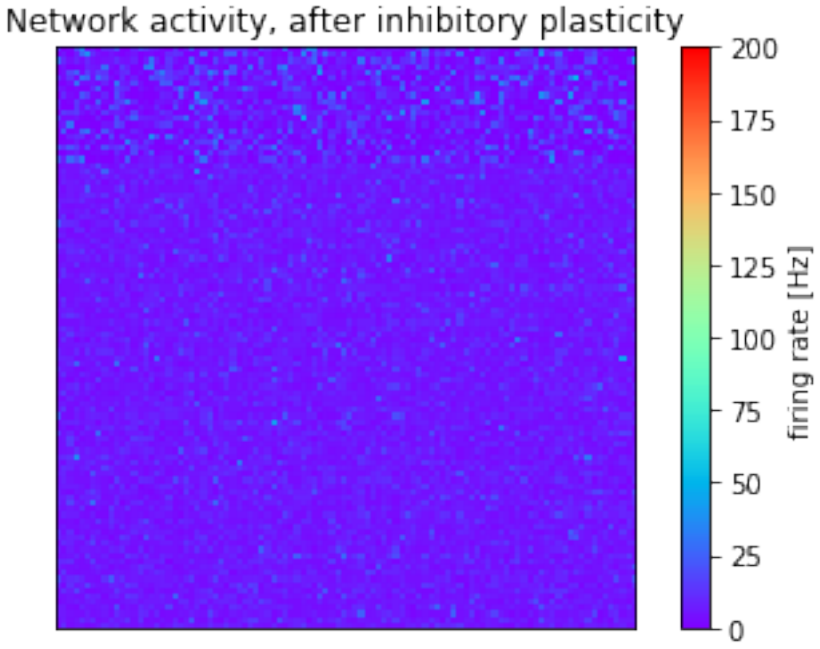
```
2.4937500000000004 (49%) simulated in 30s, estimated 30s remaining.
3.387583333333333 (67%) simulated in 40s, estimated 19s remaining.
5.0 (100%) simulated in 48s
Starting simulation at t=6. s for a duration of 14. s
1.0291400000000002 (7%) simulated in 10s, estimated 2m 6s remaining.
2.55451 (18%) simulated in 20s, estimated 1m 30s remaining.
3.7592799999999995 (26%) simulated in 30s, estimated 1m 22s remaining.
5.5824300000000004 (39%) simulated in 40s, estimated 1m 0s remaining.
7.53998 (53%) simulated in 50s, estimated 43s remaining.
9.25505 (66%) simulated in 1m 0s, estimated 31s remaining.
14.0 (100%) simulated in 1m 2s
Starting simulation at t=20. s for a duration of 5. s
0.5781800000000005 (11%) simulated in 10s, estimated 1m 16s remaining.
5.0 (100%) simulated in 18s
Starting simulation at t=25. s for a duration of 1. s
1.0 (100%) simulated in 3s
Starting simulation at t=26. s for a duration of 5. s
0.38472580645161303 (7%) simulated in 10s, estimated 2m 0s remaining.
0.7823387096774194 (15%) simulated in 20s, estimated 1m 48s remaining.
5.0 (100%) simulated in 20s
Starting simulation at t=31. s for a duration of 14. s
0.8253155555555554 (5%) simulated in 10s, estimated 2m 40s remaining.
1.8646444444444459 (13%) simulated in 20s, estimated 2m 10s remaining.
2.7237155555555566 (19%) simulated in 30s, estimated 2m 4s remaining.
3.592991111111112 (25%) simulated in 40s, estimated 1m 56s remaining.
14.0 (100%) simulated in 48s
Starting simulation at t=45. s for a duration of 6. s
0.30914117647058875 (5%) simulated in 10s, estimated 3m 4s remaining.
0.6632470588235293 (11%) simulated in 20s, estimated 2m 41s remaining.
6.0 (100%) simulated in 21s
100. us
```

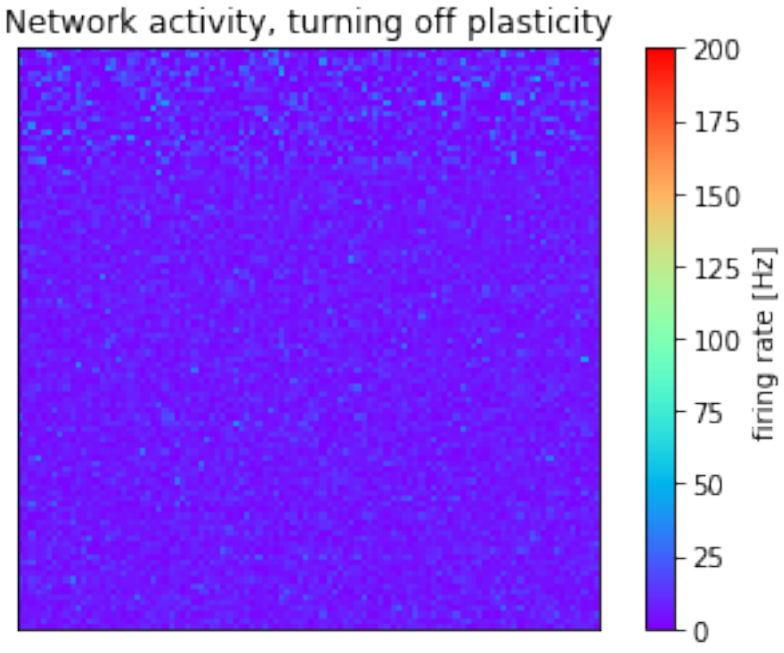
In [4]:
```
###############################################
#Make plots
###############################################
imshow(heat_no_learning, origin='lower', cmap='rainbow')
title('Network activity, without inhibitory plasticity')
yticks([])
xticks([])
clim(0, 200)
cbar = colorbar()
cbar.set_label('firing rate [Hz]')
show()

imshow(heat_asyn_irg, origin='lower', cmap='rainbow')
title('Network activity, after inhibitory plasticity')
clim(0, 200)
```

```
cbar = colorbar()
yticks([])
xticks([])
cbar.set_label('firing rate [Hz]')
show()

imshow(heat_reset_learning, origin='lower', cmap='rainbow')
title('Network activity, turning off plasticity')
clim(0, 200)
cbar = colorbar()
yticks([])
xticks([])
cbar.set_label('firing rate [Hz]')
show()

imshow(heat_change_connect, origin='lower', cmap='rainbow')
title('Network activity, immediately after increasing connection probability')
clim(0, 200)
cbar = colorbar()
yticks([])
xticks([])
cbar.set_label('firing rate [Hz]')
show()

imshow(heat_post_rebalance, origin='lower', cmap='rainbow')
title('Network activity, asymmetry returned after increasing connection probability')
clim(0, 200)
cbar = colorbar()
yticks([])
xticks([])
cbar.set_label('firing rate [Hz]')
show()

i, t = SM_e.it
# plotting before and after STDP
subplot(211)
plot(t/ms, i, 'k.', ms=0.25)
title('synchronous regular')
xlabel('')
yticks([])
xlim((sim_times['no_learning']-0.2)*1e3, sim_times['no_learning']*1e3)
subplot(212)
plot(t/ms, i, 'k.', ms=0.25)
xlabel('time (ms)')
yticks([])
title('asynchronous irregular')
xlim((sim_times['asyn_irg']-0.2)*1e3, sim_times['asyn_irg']*1e3)
show()
```

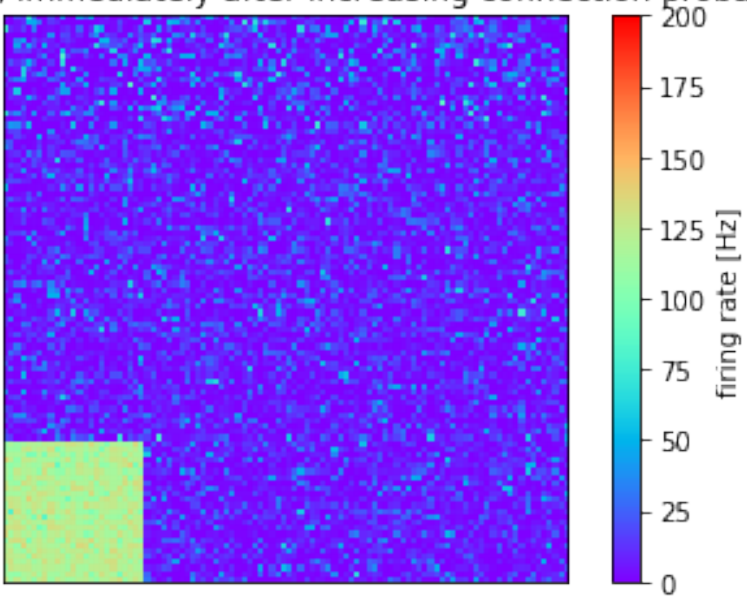
```
plot(t/ms, i, 'k.', ms=0.25)
xlabel('time (ms)')
yticks([])
xlim((sim_times['assembly']-0.2)*1e3, sim_times['assembly']*1e3)
# title('change in the connection probability')
show()

subplot(211)
# plot(S_rec_e.t/ms, S_rec_e.i, 'o')
plot(I_rec_e.t/ms, I_rec_e.v[0]/mV)
title('membrane voltage')
ylabel('voltage (mV)')
# yticks([])
subplot(212)
plot(I_rec_e.t/ms, I_rec_e.I_n[0]/pA, label='net_cur')
plot(I_rec_e.t/ms, I_rec_e.I_i[0]/pA, label='inh_cur')
plot(I_rec_e.t/ms, I_rec_e.I_e[0]/pA, label='exc_cur')
ylabel('current (pA)')
xlabel('time (ms)')
title('membrane currents')
legend(loc='lower right')
show()

plot(t/ms, i, 'k.', ms=0.25)
xlabel('time (ms)')
yticks([])
show()
```



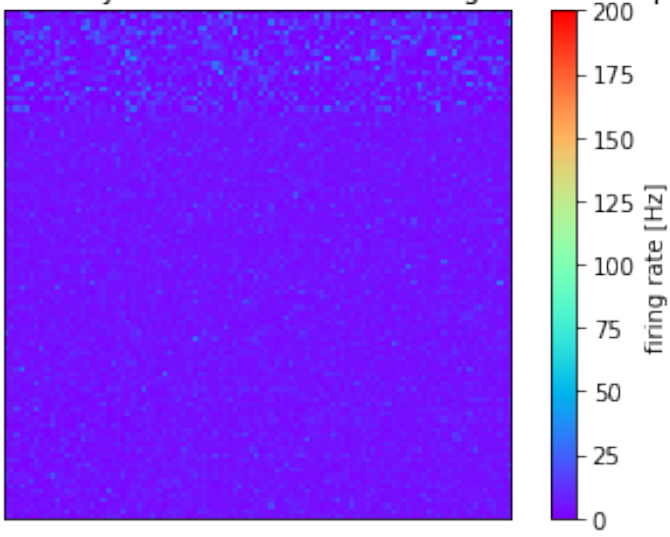

Network activity, after inhibitory plasticity
200
175
150
125
100
75
50
25
0
firing rate [Hz]

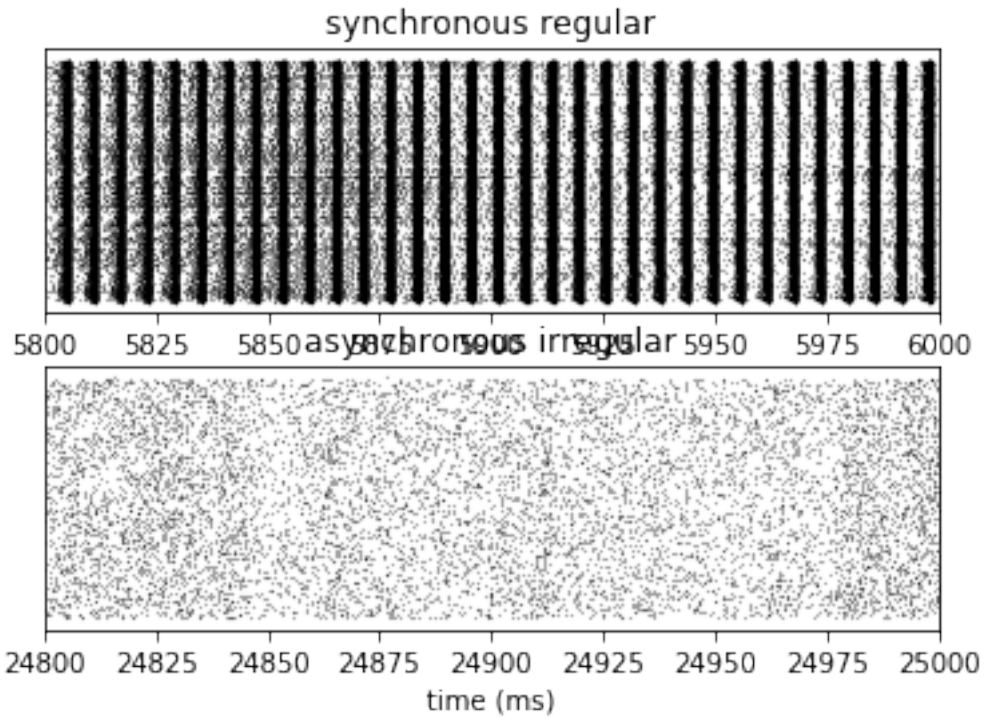

Network activity, turning off plasticity
200
175
150
125
100
75
50
25
0
firing rate [Hz]

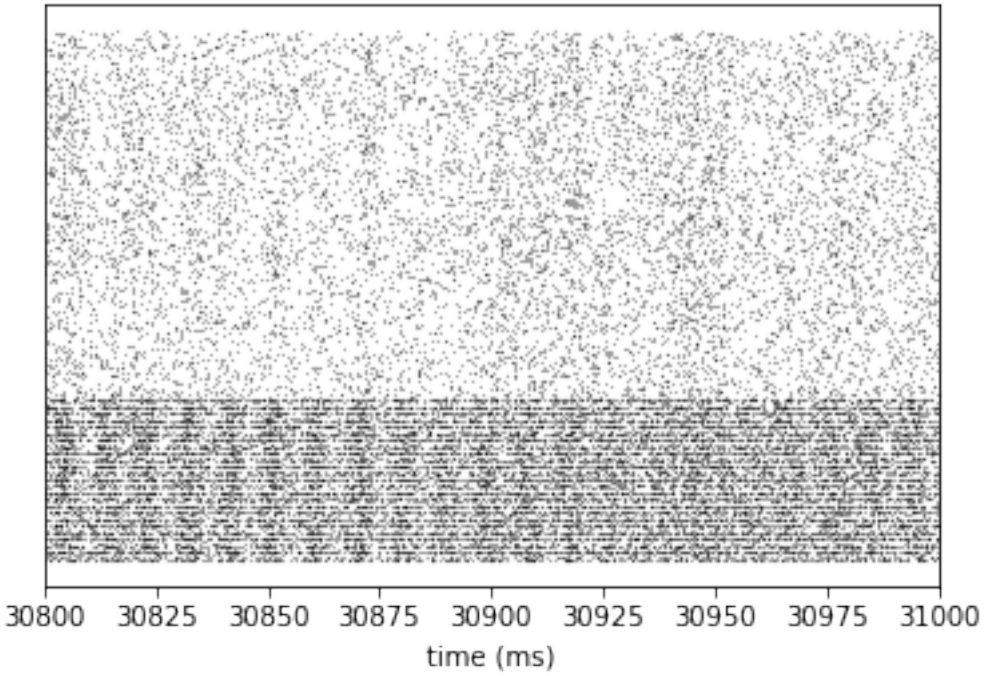
Network activity, immediately after increasing connection probability

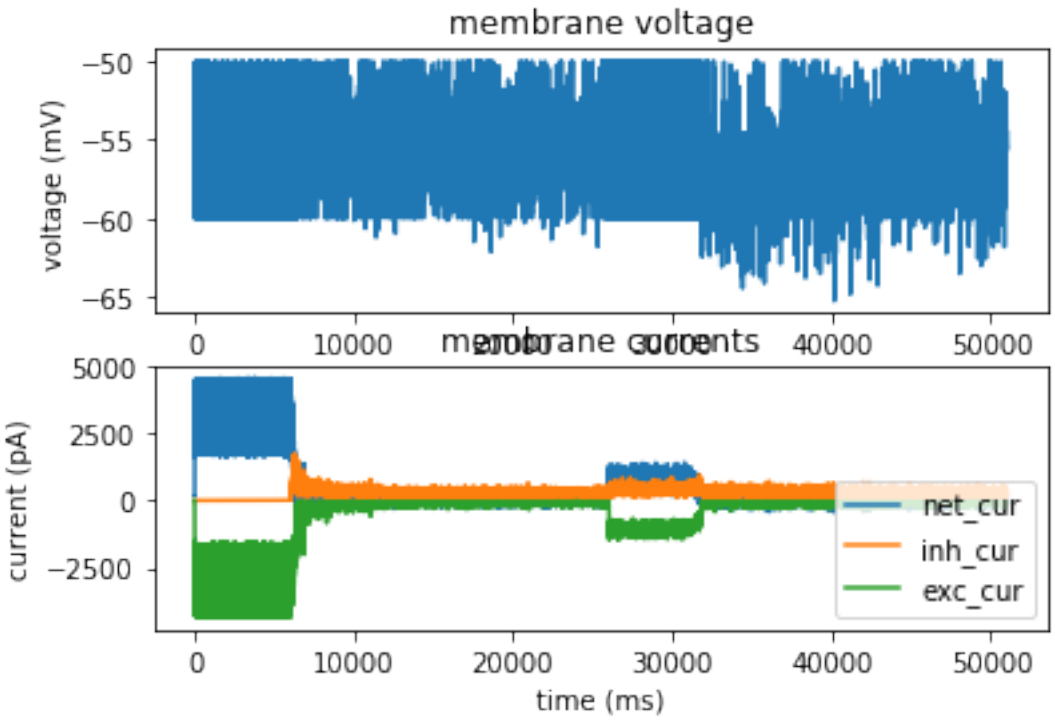


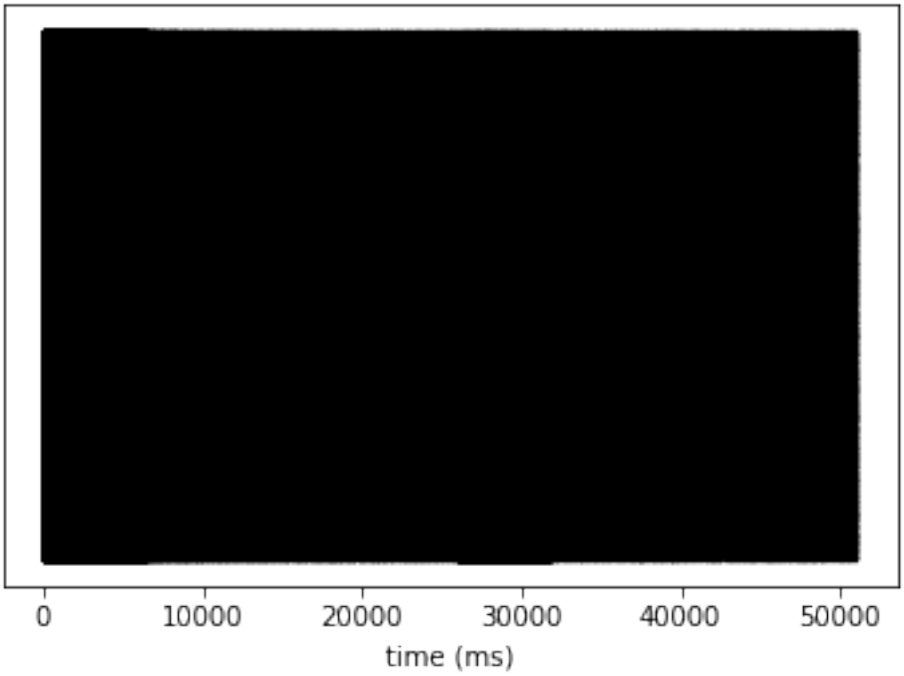
Network activity, asymmetry returned after increasing connection probability



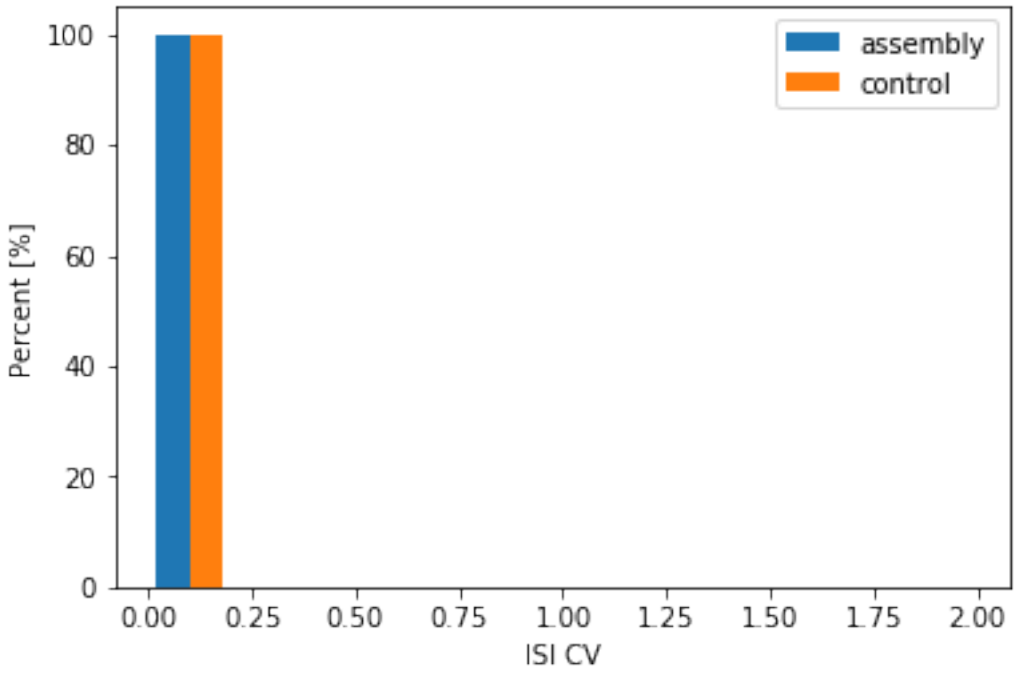

synchronous regular
5800
5825
5850
5950
5975
6000
asynchronous irregular
24800
24825
24850
24875
24900
24925
24950
24975
25000
time (ms)

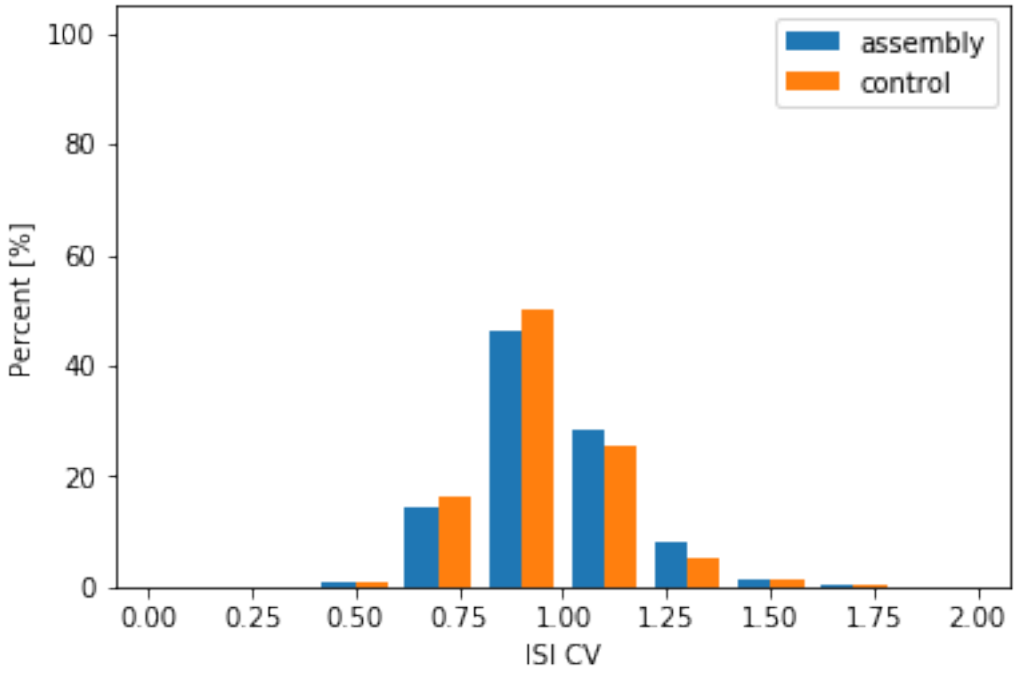

30800
30825
30850
30875
30900
30925
30950
30975
31000
time (ms)

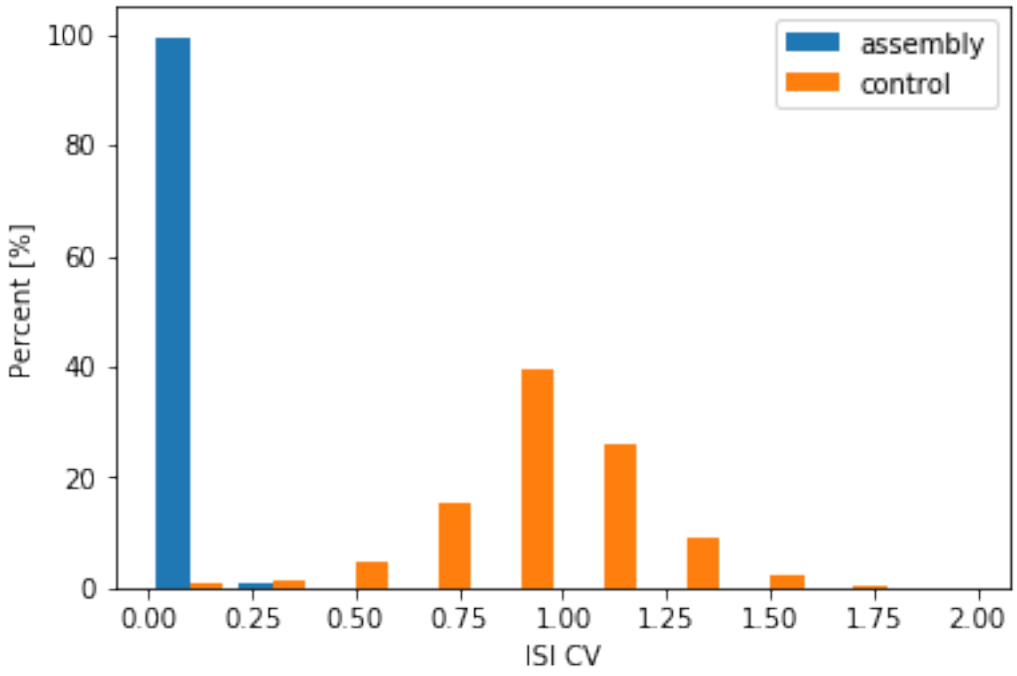

membrane voltage
voltage (mV)
−50
−55
−60
−65
0
10000
20000
30000
40000
50000
membrane currents
current (pA)
5000
2500
0
−2500
net_cur
inh_cur
exc_cur
time (ms)

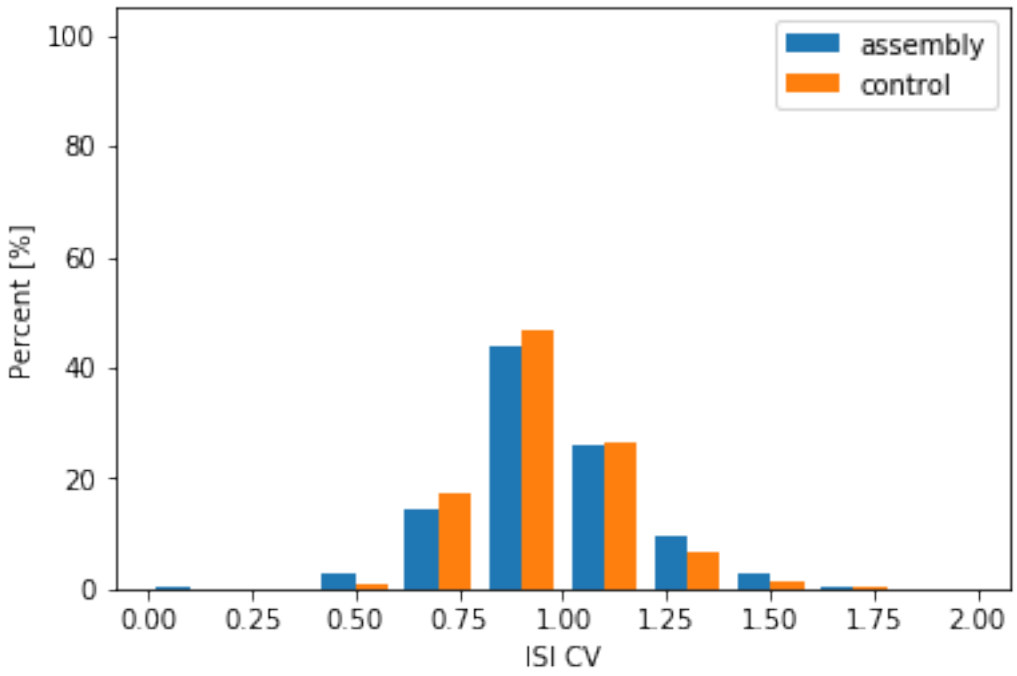

0
10000
20000
30000
40000
50000
time (ms)

In [5]:
```
trains = SM_e.spike_trains()

#ISI CV before STDP
cv1_A, cv2_A = ISI_CV(trains, sim_times['init'], sim_times['no_learning'])
#ISI CV when balanced state is reached
cv1_B, cv2_B = ISI_CV(trains, sim_times['set_learning'], sim_times['asyn_irg'])
#assembly
cv1_C, cv2_C = ISI_CV(trains, sim_times['pre_assembly'], sim_times['assembly'])
#after rebalancing
cv1_D, cv2_D = ISI_CV(trains, sim_times['wait_rebalance'], sim_times['post_rebalance'])
```



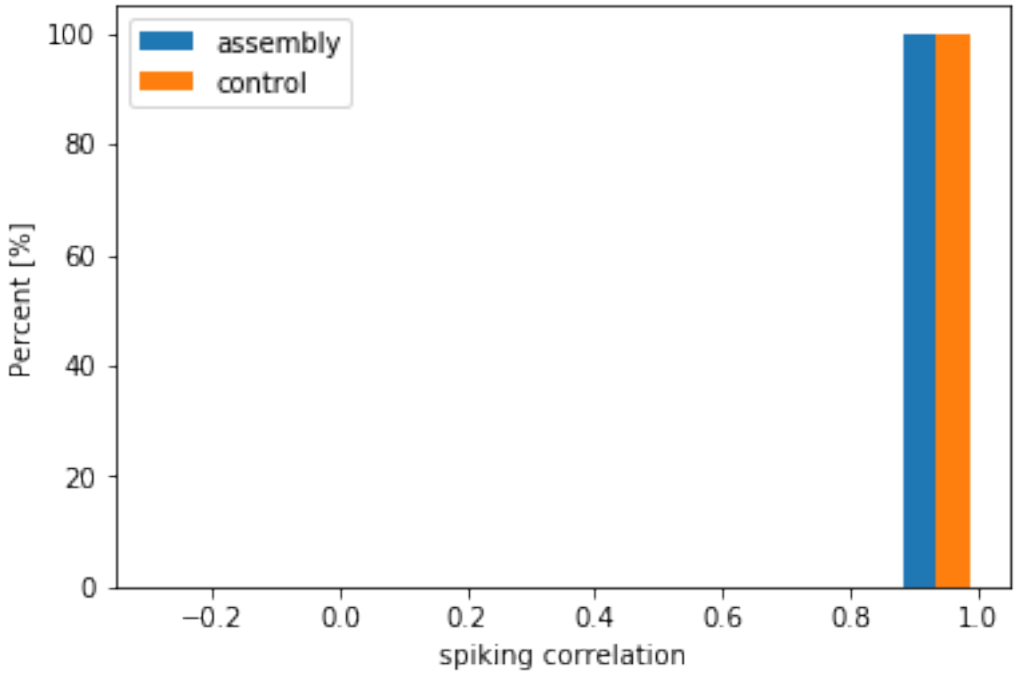

assembly
control
Percent [%]
100
80
60
40
20
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
ISI CV


assembly
control
Percent [%]
100
80
60
40
20
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
ISI CV

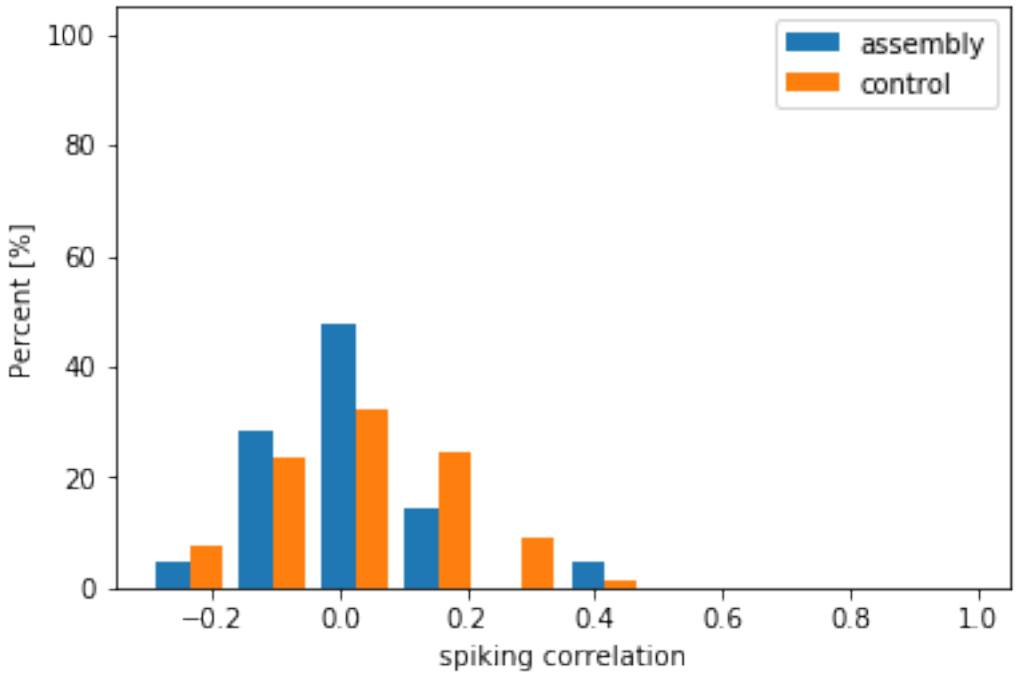


In [6]:
```
rand_set_exc = np.random.randint(0, 8000, size=100)

        #before STDP
        X1_A, X2_A = spiking_corr(trains, sim_times['init'], sim_times['no_learning'], rand_se
        #balanced
        X1_B, X2_B = spiking_corr(trains, sim_times['set_learning'], sim_times['asyn_irg'], ra
        #assembly
        X1_C, X2_C = spiking_corr(trains, sim_times['pre_assembly'], sim_times['assembly'], ra
        #after rebalancing
        X1_D, X2_D = spiking_corr(trains, sim_times['wait_rebalance'], sim_times['post_rebalan
```

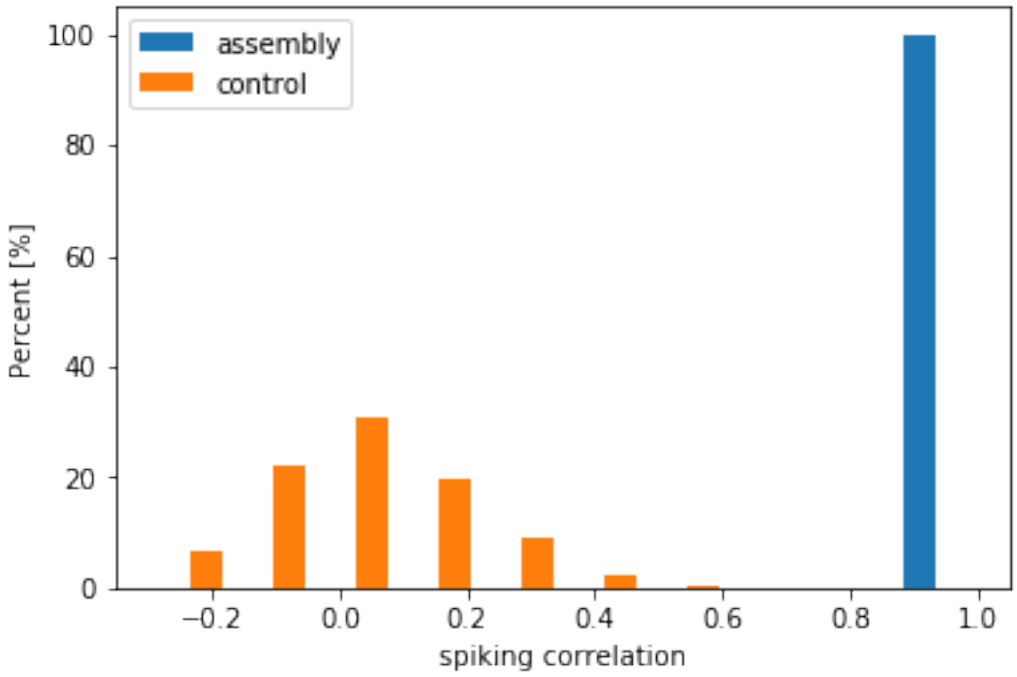
```
please wait...
```


assembly
control
Percent [%]
spiking correlation
100
80
60
40
20
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

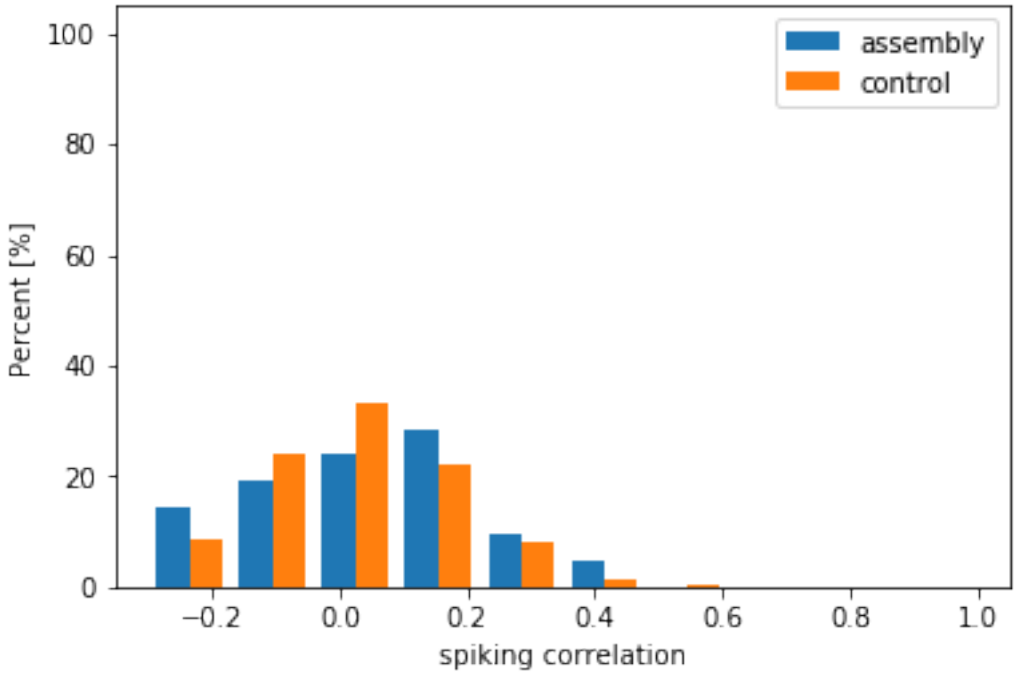
```
please wait...
```


assembly
control
Percent [%]
spiking correlation
100
80
60
40
20
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

```
please wait...


WARNING    C:\Anaconda3\lib\site-packages\numpy\lib\function_base.py:3183: RuntimeWarning: inva
  c /= stddev[:, None]
 [py.warnings]
WARNING    C:\Anaconda3\lib\site-packages\numpy\lib\function_base.py:3184: RuntimeWarning: inva
  c /= stddev[None, :]
 [py.warnings]
```




```
please wait...
```


assembly
control
100
80
60
40
20
0
Percent [%]
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
spiking correlation