Optimizing the Synchronization of Two Neural Networks

Chih-Chung Kuo
PID:A53075223
Email:c6kuo@eng.ucsd.edu

Jun Wang
PID:A53077269
Email:juw087@eng.ucsd.edu

Abstract

In this paper, we aim to provide a non-adaptive signal-preprocessing algorithm that enhances synchronization between two networks. The degree of synchronization of two networks can be a good indicator of the fidelity of signal delivered to the biological network, which is of increasing importance following the advent of sensory prosthetics. We propose that the channel-to-channel correlation matrix could serve as a way of gaining insights about a 'black box network', and that a pseudo network created based on the reverse correlation matrix can improve synchronization if the input signals are pre-processed through the pseudo network. We also investigate different intra- and inter-network connection topologies and their effects on efficiency of signal delivery in order to minimize the connections needed to completely drive a biological network.

1 Introduction

Recent advances in neural prosthetics have led to the advent of sensory prosthetic devices[1]. Such devices can create sensory emulating signals by electrically stimulating neurons. However, the interface between the biological neural network and the prosthetic devices are of particular interest in the context of sensory prosthetic devices due to the non-linear and stochastic nature of the biological network.

Simulations using Python-based Brian indicated that even very sparsely connected neurons result in considerable distortions in signals. In order to fully reconstruct the signals generated by the sensory prosthetics, signals must be pre-processed at the prosthetics to counteract distortions introduced by the biological network.

The degree of synchronization of two networks is a good indicator of the fidelity of signal delivered to the biological network, which is of increasing importance following the advent of sensory prosthetics [2]. Besides signal fidelity, the efficiency of signal delivery is also of importance. Since currently available neural stimulating techniques involve invasive contacts, reduction of interface area will undoubtedly attract more potential users. However, minimally invasive interface inevitably reduces the number of connections between the biological network and the neural prosthetic device. Different connection topologies must be incorporated to provide comprehensive stimulation of the biological neural network. (Figure 1).

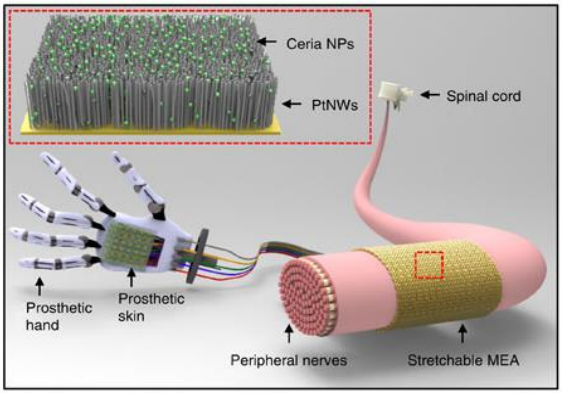


Figure 1. Interface between a prosthetic device and the biological neural network

Currently, several optimizing algorithms have been proposed, including the liquid state machine[3] and the echo state machine[4]. However, all these currently available methods rely greatly on machine learning algorithms, whose performance is often times mediocre given insufficient number of trainings.

In this paper, we tried a new way to enhance the synchronizing of two networks which is evaluated by compare the correlation between input and output. It is simplified and time-saving.

2 Method

2.1 Network construction

We tried to use two networks to mimic the process that signal transfers from a device to a biological neural network. As shown in Figure 2, A denotes the device and B denotes the biological network.

In order to obtain legitimate evaluation results, the simulated biological network has to be faithful representation of a real biological network. On top of that, the simulated network has to be scalable such that it is applicable to various scenarios. Below details the simulation setup we used throughout the project with an aim to satisfy the above requirements.

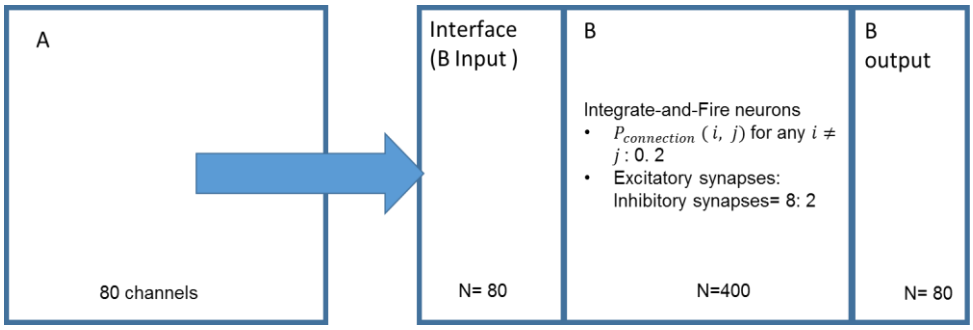


Figure 2. The basic frame of networks

2.1.1 Signal producing network

As figure1 shows, network A is the signal producing network. In real-life scenario, this is equivalent to the prosthetic device from which sensory signals are generated. This signal-generating network can be set up to produce any signals. Of particular significance here are the Possion input and sinusoidal input, which are used during the training process and evaluation

respectively. Network A is capable of performing distortion eliminating signal-preprocessing, which in this project is realized through the pseudo neural network connections from A to Binput.

2.1.2 Network B

Network B was constructed to mimic the biological neural network. It comprises three layers (input, hidden and output). A total of 400 integrate-and-fire neurons are used as the hidden layer, of which 80% of the neurons are excitatory and 20% are inhibitory ([5]). To evaluate the overall activity of network B, a total of 80 randomly selected channels are measured. The selected channels forward signals to B_{output}, where in real life can be an array of electrodes. Note that the channel numbers also reflect the location of a specific neuron, with $|n_i - n_j|$ being the distance between two neurons as the system is a 1-D approximation of a biological network.

2.1.3 Connection between networks

Neurons in a biological network can form sparse inter-connections within the same network. Synapses may as well span different networks, relaying information from one network to another. The ways to connect two networks can be random, full, small world and connections with custom matrix as weight. Actually, we can achieve different connection by adjusting the custom matrix.

2.2 Training and evaluation

We tried to use a matrix M to characterize the network B, then the output of network B can be described by equation (1). To achieve high fidelity of signal transfer, we use the inverse Matrix of M, namely M^{-1}, as the connection weight matrix between A and input of B. Then the input of network B, B_{input} can be denoted as equation (2). Therefore, B_{output} would equal to A, which shows in equation (3).

$$B_{output} = B_{input} * M \quad (1)$$

$$B_{input} = A * M^{-1} \quad (2)$$

$$\mathbf{B_{output}} = \mathbf{A} * M^{-1} * \boldsymbol{M^{-1}} * \mathbf{M} = \mathbf{A} \quad (3)$$

The key point of the network construction is how to obtain the Matrix M. we determined the value of elements of M via correlation. We supposed that a virtual direct link between every input element (I_1 to I_N) and every output element (O_1 to O_N) exists, as the Figure 3 shows. Herein, a corresponding correlation coefficient $Corr_{mn}$ could be calculated, as equation (4) shows. Further, we regarded the element M_{mn} of Matrix M is direct proportion to $Corr_{mn}$, as equation (5) shows. Finally, we found that the relative optimal results could be obtained when K is 2000. In this way, the matrix M was determined.

$$\mathrm{Correlation}(I_m, O_n) = Corr_{mn} \quad (4)$$

$$M_{mn} = K * Corr_{mn} \quad (5)$$

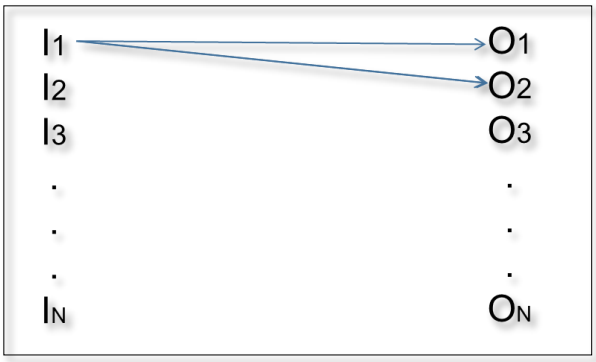


Figure 3. The principle used to calculate the correlation between input and output.

After every single time of running the code, we could obtain a M. Namely, N times later, we could get N Ms. Then the average value of Ms were calculated, and it was regarded as the final M. After the Matrix M was determined, the M^{-1} was calculated subsequently. Then we substituted the M^{-1} into weight matrix of connection between network A and network B. The following are the detailed steps.

The 80 neurons in network A produced Poisson signal at 40Hz, the signal of 80 channel was fully delivered to input layer of B, which also contains 80 neurons as mentioned above. Then the corresponding output of network B could be produced. Both the data of input and output of network B were stored for calculating the correlation coefficient between each channel of input and output. According to the method mentioned above, correlation matrix M $(80 * 80)$ was determined. Then we used inv(M), the inverse matrix of M, as the connection matrix for $A\text{-}B_{input}$. Until now, the training of the networks were finished.

Last, we calculated the Pearson's correlation coefficient between input and output to evaluate the degree of two neural networks' synchronizing using the following equation (6).

$$\rho_{Input,Output} = \mathrm{corr}(Input, Output) = \frac{\mathrm{cov}(Input, Output)}{\sigma_{Input}\sigma_{Output}} = \frac{E\left[\left(Input - \mu_{Input}\right)\left(Output - \mu_{Output}\right)\right]}{\sigma_{Input}\sigma_{Output}} \quad (6)$$

3 Result and discussion

3.1 The Signal input and corresponding output

Before we can evaluate the training performance, a proper model of a biological neural network must first be constructed. Brian Simulator allows semi scalable systems in which the number of neurons in a network can be specified arbitrarily. However, simulations showed that the built in connection topologies in Brian Simulator induces excess spiking activities when a network is sufficiently large.

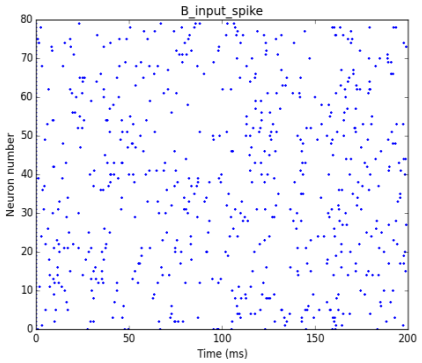
3.1.1 Random connections in Biological Network

When the neurons within a network are connected randomly, spontaneous spiking occurs when the network is large. Since increased neural network size also increases the number of synapses connected to a neuron will also increase. The accumulation of signals from these synapses may easily exceed the spiking threshold for the post-synaptic neuron. This eventually results in a positive feedback loop, which is not biologically tolerable.

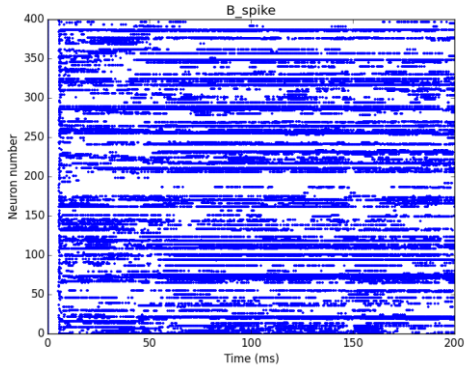
3.1.2 Localized connections in Biological Network

In an attempt to solve for the scalability of the system, a new routing algorithm that takes into account the distance between two neurons is proposed, with probabilities that a connection exists between neurons in close proximity being higher than when two neurons are placed distantly apart. This eliminates the spontaneous spiking activities of a simulated biological neural network.

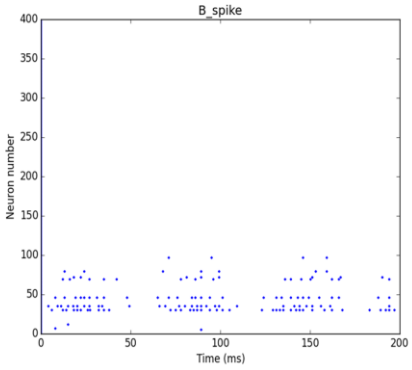
As of now, the input signal from B_{input} to B is delivered on a 1-1 basis, in which the 80 input (Figure 4(a)) channels map to the first 80 out of the 400 channels in network B. When hidden layer of B is connected by a random way, the neurons of B are easy to be bursting, like Figure 4 (b) shows. It is still problematic, however, if localized connections are implemented in network B. If only the first 80 channels of network B are provided with input, the signals normally would not reach faraway neurons because localized connection dictate that signals are not likely to be delivered to distant neurons in a single hop (Figure 4(c)). In the next section we aim to provide a solution to this problem.



(a)



(b)



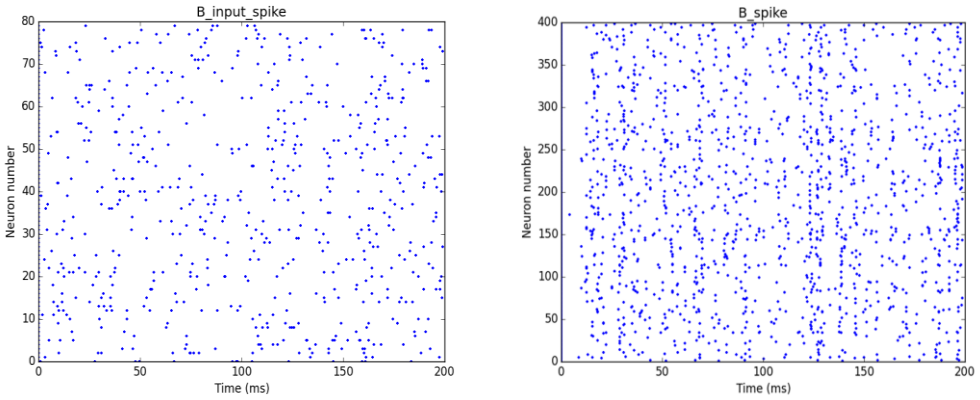
(c)

Figure 4. The plot of spiking of input and output layer when the connection is conducted via small world. (a) Random spiking of input layer of network B; (b) the spiking of network B when the connection of B itself is random; the connection between input layer and hidden layer of B is also random; (c) the spiking of network B when the connection of B itself is localized; the connection between input layer and hidden layer of B is one to one to one.

3.1.3 Small world connections at the interface

To provide comprehensive stimulation of a network that adopts localized intra-connections, optimally inputs need to be spread evenly on all channels. However, this is infeasible *in vivo* as neurons in living organisms are not evenly spaced, making it hard for electrodes to evenly stimulate the whole network. To better reflect reality, a simplified small world connection model

is used. In our model, 40 synapses form random connections with neurons in network B. The rest of the 40 connections are location-based. As Figure 5 shows, results from this combination of small world and location-based connections are consistent with biological experiments. Therefor we will use such connecting topologies throughout this project.



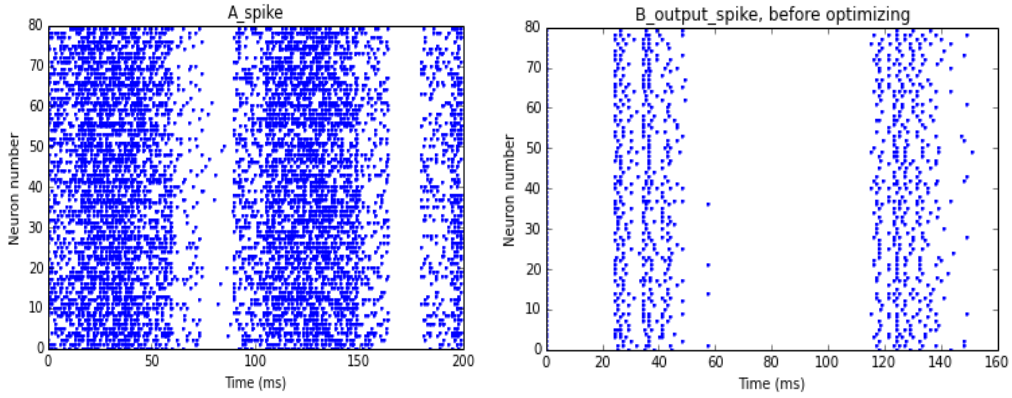
(a) (b)

Figure 5. The plot of spiking of input and output layer when the connection is conducted via small world. (a) The spiking of input layer of B; (b) the spiking of output layer of B.

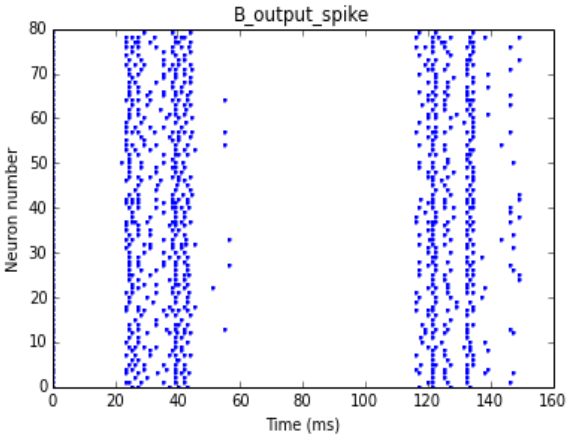
3.2 Performance analysis

There are two ways to quantify the synchronization between two networks. To analyze the channel-to-channel synchronization, correlation between any two channels can be evaluated. In Python, this is simply done by taking the Pearson's correlation coefficients between two channels. To determine the activity of a whole network, a small-time-window averaging of all the spikes in a certain time period can be calculated as a function of time[6]. Correlation coefficient can then be derived based on the averaging results from the input and the output networks.

After the training is complete, the system is ready to be tested. A perturbed sinusoidal signal (Figure 6(a)) is used to evaluate the performance of our anti-distortion training algorithm. Our input signal differs from normal sinusoidal spike trains in that every channel is still independent of each other. Thus channel-to-channel correlation analysis still gives meaningful information as to how an input channel correlates to an output channel. Figure 6(b), (c) show the output before and after optimizing respectively.



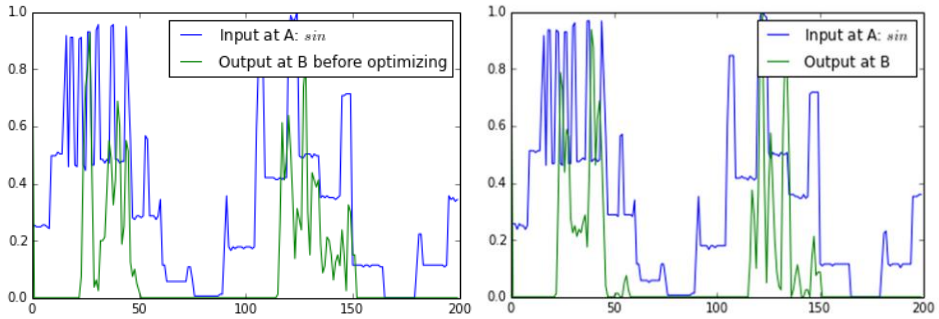
(a) (b)



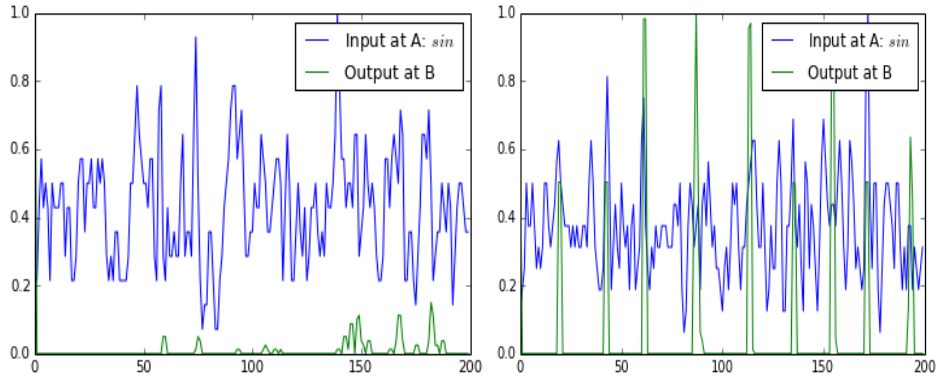
(c)

Figure 6. Results of raster_plot when the input is sine. (a) The spike of A; (b) the spike of output layer of B before optimizing; (c) the spike of output layer of B after optimizing.

The raster plots above show that even after optimization, signals are still rather distorted. However, small-time-window averaging indicates improvement in overall synchronization of neural activities between the two networks, with an increase in correlation from 0.4610 to 0.7255. Figure 7(a), (b) show the results of using perturbed sinusoidal signals as inputs before and after optimizing respectively. When using indecently random signals as inputs, the improvement is also evident, with an increase in correlation from -0.0644 to 0.2915, as the Figure 7(c), (d) show.



(a) (b)



(c) (d)

Figure 7. Results of raster_plot. The width of time window is 1ms. (a) The spike of A, when the input is sine; (b) the spike of output layer of B, when the input is sine; (c) the spike of A, when the input is Poisson; (d) the spike of output layer of B, when the input is Poisson.

We further analyzed the exact reasons that lead to the increase of the synchronization. Above results show that our optimizing is effective. After using the M^{-1} to replace the original matrix of full connection, what parameters are changed? 'The effective number of synapse and the strength of every synapse'.

It seems obvious that when the strength of every synapse increases, the synchronization will increase[7]. But that makes sense only when inhibitive synapse does not exist. Actually, the synchronization decreased when we enlarge the value of M^{-1}. That identifies that the increase of synchronization of two networks constructed in our project is not directly related to the increase of strength of matrix weight. Therefore the remaining possible reason is that the effective number of synapse which enhances synchronization increase.

3.3 Advantages and disadvantages

Basically, we only need one run to obtain the connection matrix. It is more efficient compared with other recursive training methods given limited train time. However, the matrix we obtained is not exact the transfer matrix. As a result, the corresponding correlation between input and output might not be the highest.

3.4 Future work

The results above hinted that conversion from the inverse matrix to the pre-processing connection matrix may not be straightforward. Several optimizations of the inverse correlation matrix can be carried out in order to obtain greatest improvement over synchronization between two networks. Such modifications include normalization and shifting the weights such that the mean becomes zero; however, none of these measures can completely counteract the distortions introduced in the biological network. We suspect this is due to the intrinsic difference between a connection matrix and an inverse correlation matrix.

A more deterministic way of solving such problems involves deriving the transfer function of the nonlinear biological neural network[8]. More accurate distortion elimination mechanisms can be developed accordingly.

4 Conclusion

In this project, we tried to use the synchronizing degree of two networks to measure the fidelity of signal reconstructions at the brain. Based on Python and Brian, two networks were constructed.

Compared ways to connect two network, we found that the efficiency of signal delivery could be enhanced when using small world connection. Besides, we obtained the characteristic matrix M of network B basically via calculating the correlation between input elements and output elements of B. At last, we consider the M^{-1} as the weight matrix of A and input of B. In this way, we successfully increased the synchronizing between two networks.

Acknowledgments

We wish to express our sincere gratitude to our supervisor of this project, Professor Cauwenberghs for the valuable guidance and advice. He inspired us greatly to continue working on this project. We also would like to thank Dr. Fred and Bruno who offered a great assistance and support. This project would not have been possible without the support of bioengineering department of University of California, San Diego.

References

[1] J. Kim, M. Lee, H. J. Shim, R. Ghaffari, H. R. Cho, D. Son, *et al.*, "Stretchable silicon nanoribbon electronics for skin prosthesis," *Nat Commun*, vol. 5, 12/09/online 2014.

[2] D. Eytan and S. Marom, "Dynamics and effective topology underlying synchronization in networks of cortical neurons," *J Neurosci*, vol. 26, pp. 8465-76, Aug 16 2006.

[3] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of Computer and System Sciences*, vol. 69, pp. 593-616, 12// 2004.

[4] F. Xue, Z. Hou, and X. Li, "Computational capability of liquid state machines with spike-timing-dependent plasticity," *Neurocomputing*, vol. 122, pp. 324-329, 12/25/ 2013.

[5] N. Brunel, "Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons," *Journal of Computational Neuroscience*, vol. 8, pp. 183-208, 2000/05/01 2000.

[6] A. Sornborger, T. Yokoo, A. Delorme, C. Sailstad, and L. Sirovich, "Extraction of the average and differential dynamical response in stimulus-locked experimental data," *Journal of Neuroscience Methods*, vol. 141, pp. 223-229, 2/15/ 2005.

[7] J. Cao and J. Lu, "Adaptive synchronization of neural networks with or without time-varying delay," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 16, pp. -, 2006.

[8] D. Lekkas, C. Imrie, and M. Lees, "Improved non-linear transfer function and neural network methods of flow routing for real-time forecasting," *Journal of Hydroinformatics*, vol. 3, pp. 153-164, 2001.

APPENDIX

```
# -*- coding: utf-8 -*-

from brian import *
import scipy as sp
import random
from scipy.stats.stats import pearsonr
from numpy import matrix

matplotlib.pyplot.close("all")



tau = 20 * msecond # membrane time constant
Vt = -50 * mV # spike threshold
Vr = -60 * mV # reset value
El = -61 * mV # resting potential
psp = 0.5 * mvolt # postsynaptic potential size
neuron_number=400
runtime=200 * msecond
for runrep in range (1):
reinit(0*second)
def Corr(array1, array2,len_array):
M_forward=sp.zeros((len_array,len_array))
for i in range(len_array):
for j in range(len_array):
temp=pearsonr(array1[i],array2[j])
M_forward[i,j]=temp[0]
return M_forward
def SPKvsT(neu_spikes, timewindow, timescale):
neuronnum, spktime= zip(*neu_spikes)
spikeaccum=zeros(timewindow*timescale)
for tidx in range(timewindow*timescale):
for spkidx in range(len(spktime)):
if abs(tidx/timescale *ms-spktime[spkidx]) < 1/timescale *ms:
spikeaccum[tidx]=spikeaccum[tidx ]+1
return spikeaccum
#def GetConnectionMatrix():
# B_input=PoissonGroup(50, rates=40*Hz)
# V_B_input = StateMonitor(B_input, 'V', record=True)
# V_B_output = StateMonitor(B_output, 'V', record=True)
# corr=Corr(V_B_input,V_B_output,50)
# return corr
#def Matrix_distance(neuron_number):
# weightmatrix=sp.zeros((neuron_number,neuron_number))
# for x in range(neuron_number):
# for y in range(x+1,neuron_number):
```

```
366 # if random.random()>0.3: #excitatory connection
367 # weightmatrix[x,y]=50*exp(-abs(x-y)/100)*mV
368 # elif random.random()<0.1: #inhibitory connection
369 # weightmatrix[x,y]=-50*exp(-abs(x-y)/100)*mV
370 # else:
371 # weightmatrix[x,y]=0
372 # return weightmatrix
373 #
374 #def reconstructMatrix(Mat):
375 # x_len, y_len =Mat.shape
376 # weightmatrix=sp.zeros((x_len,y_len))
377 # for x in range(x_len):
378 # for y in range(y_len):
379 #eqs = Equations('''
380 #dV/dt = (-V+ge-gi)/taum : volt
381 #dge/dt = -ge/taue : volt
382 #dgi/dt = -gi/taui : volt
383 #''')
384 ### definition of connection
385 #B = NeuronGroup(N=400, model='dV/dt = -(V-El)/tau : volt',
386 # threshold=Vt, reset=Vr)
387 #B_Connection = Connection(B,B,weight=Matrix(len(B),len(B)))
388 #
389 #
390 #B_input=NeuronGroup(N=50, model='dV/dt = -(V-El)/tau : volt',
391 # threshold=Vt, reset=Vr)
392 #B_input_Connection =
393 Connection(B_input,B_input,structure='dense',weight=Matrix(len(B_input),len(B
394 _input)))
395 #
396 #
397 #B_output=NeuronGroup(N=50, model='dV/dt = -(V-El)/tau : volt',
398 # threshold=Vt, reset=Vr)
399 #B_output_Connection =
400 Connection(B_output,B_output,weight=Matrix(len(B_output),len(B_output)))
401 #
402 #B_input_B_Connection =
403 Connection(B_input,B,weight=Matrix(len(B_input),len(B)))
404 #B_B_output_Connection =
405 Connection(B,B_output,weight=Matrix(len(B),len(B_output)))
406 ## definition of connection
407 def sinchoose():
408 spikelist=[]
409 for time in range(200):
410 numberofspikes=int((sin(time/15 )/2 + 0.5)*80)
411 for idx in range(numberofspikes):
```

```
412 spikelist.append([random.randint(0,80),time*ms])
413 return spikelist
414 A = SpikeGeneratorGroup(80,sinchoose())
415 # A=PoissonGroup(80, rates=40*Hz)
416 B = NeuronGroup(N=400, model='dV/dt = -(V-El)/tau : volt',
417 threshold=Vt, reset=Vr)
418 B_Connection = Connection(B,B,weight=load('W_BB.npy'))
419 B_input=NeuronGroup(N=80, model='dV/dt = -(V-El)/tau : volt',
420 threshold=Vt, reset=Vr)
421 #B_input_Connection =
422 Connection(B_input,B_input,weight=load('W_BiBi.npy'))
423 B_output=NeuronGroup(N=80, model='dV/dt = -(V-El)/tau : volt',
424 threshold=Vt, reset=Vr)
425 #B_output_Connection =
426 Connection(B_output,B_output,weight=load('W_BoBo.npy'))
427 B_input_B_Connection = Connection(B_input,B,weight=load('W_BiB.npy'))
428 #B_input_B_Connection = Connection(B_input,B, sparseness=0.015625,
429 weight=5*mV)
430 B_B_output_Connection = Connection(B,B_output,weight=load('W_BBo.npy'))
431 #B_B_output_Connection = Connection(B,B_output, sparseness=0.015625,
432 weight=5*mV)
433 C1 = Connection(A, B_input) #assume one to one A->B connection
434 C1.connect_one_to_one(weight=1*mV)
435 # A_B_input_Connection = Connection(A,B_input,
436 weight=load('corr_I_avg.npy'))
437 #A_B_input_Connection = Connection(A,B_input,
438 weight=load('avg_corrI60.npy'))
439 ###A_B = Connection(A,B_input, weight=)
440 #func=10* mvolt
441 #C1 = Connection(A, B_input, weight=func)
442 #B_input=B.subgroup(50);
443 #B_output=B.subgroup(50);
444 # func is the function of distance between
445 # pre and post, the membrane potential, and the number of connection
446 A_spike = SpikeMonitor(A)
447 B_spike = SpikeMonitor(B)
448 B_input_spike = SpikeMonitor(B_input)
449 B_output_spike = SpikeMonitor(B_output)
450 #B.V = Vr + rand(400) * (Vt - Vr)
451 #M = StateMonitor(B, 'V', record=True)
452 #C1[0, 0] = 50 * mV
453 V_B_input = StateMonitor(B_input, 'V', record=True)
454 V_B_output = StateMonitor(B_output, 'V', record=True)
455 run(runtime)
456 corr=Corr(V_B_input,V_B_output,80)
457 # corr_temp=matrix(corr)
```

```
458 # corr_I=corr_temp.I
459 # save('corr_locoff1_'+str(runrep) ,corr )
460 # save('corr_I_locoff1_'+str(runrep),corr_I)
461 print B_spike.nspikes
462 print B_output_spike.nspikes
463 #
464 #figure(1)
465 #plot(M.times / ms, M[0] / mV)
466
467 xlabel('Time (in ms)')
468 ylabel('Membrane potential (in mV)')
469 title('Membrane potential for neuron 0')
470
471
472 figure(1)
473 raster_plot(A_spike)
474 title('A_spike')
475
476 figure(2)
477 raster_plot(B_spike)
478 title('B_spike')
479
480 figure(3)
481 raster_plot(B_input_spike)
482 title('B_input_spike')
483
484 figure(4)
485 raster_plot(B_output_spike)
486 title('B_output_spike, before optimizing')
487
488
489 inputspk =SPKvsT(A_spike.spikes,200,1)
490 outputspk=SPKvsT(B_output_spike.spikes,200,1)
491 figure()
492 plot(sp.arange(0,200,1),inputspk /max(inputspk ), label='Input at A: $sin$')
493 plot(sp.arange(0,200,1),outputspk/max(outputspk), label='Output at B before
494 optimizing')
495 pylab.legend(loc='upper right')
496 print pearsonr(inputspk /max(inputspk ),outputspk/max(outputspk))
497
498 ////////////////////////////////////////////////////////////////////////////////////////////////////
499 from brian import *
500 import scipy as sp
501 import random
502 from scipy.stats.stats import pearsonr
```

```
from numpy import matrix
import scipy as sp
from numpy import matrix

avg=sp.zeros((80,80))
for i in range (5):
avg=avg+load('corr_locoff1_'+str(i)+'.npy')
avg=avg/10

#avg=avg/(avg.max())/10

save('corr_locoff1_',avg)

avg_temp=matrix(avg)
avg_I=avg_temp.I
avg_I_transfer=avg_I+abs(avg_I.min())
avg_I_modify=avg_I_transfer/avg_I_transfer.max()
save('corr_I_avg',avg_I_modify/2000)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

import scipy as sp
import random
from brian import *

def Matrix(na,nb):
weightmatrix=sp.zeros((na,nb))
for x in range(na):
for y in range(nb):
if x==y:
weightmatrix[x,y]=0*mV
elif random()>0.8: #excitatory connection
weightmatrix[x,y]=1.00*mV
elif random()<0.2: #inhibitory connection
weightmatrix[x,y]=-1*mV
else:
weightmatrix[x,y]=0
return weightmatrix
save('W_BB1',Matrix(400,400))
save('W_BiBi1',Matrix(80,80))
save('W_BoBo1',Matrix(80,80))
save('W_BiB1',Matrix(80,400))
save('W_BBo1',Matrix(400,80))
```