FYS388_V15_Exercise08

March 20, 2015

1 FYS388 Exercise 8: Spiking Neuronal Network Models

1.1 Introduction

In this exercise, you will explore spiking networks of simple neurons. In particular, we will consider a widely studied model proposed by Nicolas Brunel about 15 years ago (N Brunel: *Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons*. J Comput Neurosci **8**:183-208 (2000). Reprint).

For details on the model structure, please see Section 2, *The Model*, of Brunel's paper. Section 3 and 4 of the paper provide the theoretical analysis and are not immediately relevant for this exercise. Section 6, in particular Figures 7 and 8 and Table 1, provide the data against which you will compare your results.

The details of the model implementation and use in NEST are described in M.-O. Gewaltig, A. Morrison, and H. E. Plesser (2014) *NEST by Example: An Introduction to the Neural Simulation Tool NEST Version 2.6.0*, which is included with NEST and also in the material for this exercise.

In Task 5, you will also investigate a network incorporating spike-timing dependent plasticity. The practical side of this is described in *NEST by Example*. For the theoretical background, see Morrison et al: *Spike-Timing-Dependent Plasticity in Balanced Random Networks*. Neural Comput **19**:1437-1467 (2007) Reprint, and Morrison et al: *Phenemenological models of synaptic plasticity based on spike timing*. Biol Cybern **98**:459-478 (2008) Reprint. Roughly speaking, if Δt is the time difference between the most recent spike of the postsynaptic neuron and the time of an incoming spike, then that spike changes the membrane potential according to (Morrison et al (2007), Eq. (2.3), with $\mu = 1$):

$$\begin{cases} w \leftarrow w + \lambda w e^{-|\Delta t|/\tau_+} & \Delta t > 0 \\ w \quad \text{unchanged} & \Delta t = 0 \\ w \leftarrow w - \alpha\lambda w e^{-|\Delta t|/\tau_-} & \Delta t < 0 \end{cases} \tag{1}$$

- λ is the learning rate
- α describes the balance between facilitation and depression
- In addition, weights are limited to $|w| \leq W_{\max}$

1.2 Task 1: Running the Brunel Model

Work in this exercise will be based on the Brunel model implementation `brunel-delta-nest.py`, which you can find among the NEST PyNEST examples. For convenience, it is also included with this exercise.

In the first task, you will get the model to run in two different ways: as a script and as a function.

As a preparatory step, open file `brunel-delta-nest.py`, find the following lines

```
nest.SetStatus(espikes,[{"label": "brunel-py-ex",
                         "withtime": True,
                         "withgid": True,
                         "to_file": True}])

nest.SetStatus(ispikes,[{"label": "brunel-py-in",
                         "withtime": True,
```

```
                  "withgid": True,
                  "to_file": True}])
```

and set the value of the `"to_file"` entries to `False`, so that NEST won't write spike data to file.

1.2.1 Simulation 1: The original model

Run the original model once, without any changes. You should obtain output that reports to the following:

```
Number of neurons : 12500
Number of synapses: 15637600
       Exitatory  : 12512500
       Inhibitory : 3125000
Excitatory rate   : 31.86 Hz
Inhibitory rate   : 31.98 Hz
Building time     : 8.52 s
Simulation time   : 91.71 s
```

Building and simulation time will be different on your computer, but the number of neurons and synapses and the firing rates reported should be identical.

1.2.2 Simulation 2: Converting the script to a function

Before you proceed, activate automatic reloading of modules in IPython with the following commands:

```
%load_ext autoreload
%autoreload 2
```

For further exploration of the model, it would be cumbersome if you had to create a new script file each time you change a parameter. As the next step, you shall therefore turn the `brunel-delta-nest.py` script into a Python module `brunel_delta.py` defining a `sim_brunel_delta()` function that builds and simulates the model network. Note the underscore in the module name: Files that are to be loaded as Python modules cannot have hyphens in their name.

To allow for greatest flexibilty, the function should take the following parameters, all with default values from the orignial script (delete the corresponding lines in the function body):

```
def sim_brunel_delta(dt=0.1,
                     simtime=1000.0,
                     delay=1.5,
                     g=5.0,
                     eta=2.0,
                     epsilon=1.0,
                     order=2500,
                     J=0.1,
                     N_rec=50,
                     num_threads=1,
                     print_report=True):
```

The `num_threads` parameter is new. To make use of it, find the following line

```
nest.SetKernelStatus({"resolution": dt, "print_time": True})
```

and extend it to

```
nest.SetKernelStatus({"resolution": dt, "print_time": True,
                      "local_num_threads": num_threads})
```

If `print_report` is true, the function should print out information about its progress and the report shown above, otherwise, it should be quiet. To suppress all output from NEST, add

```
nest.set_verbosity('M_WARNING')
```

to your code.

Remove the `nest.raster_plot.from_device()` call from the function.

The function shall return a tuple consisting of the excitatory and inhibitory spikes recorded, as Pandas data frames:

```
import pandas as pd

exc_spikes = nest.GetStatus(espikes, 'events')[0]
inh_spikes = nest.GetStatus(ispikes, 'events')[0]

return pd.DataFrame(exc_spikes), pd.DataFrame(inh_spikes)
```

Using the function Use the function like this:

```
from brunel_delta import sim_brunel_delta

esp, isp = sim_brunel_delta()
```

This should print the same report (except some variation in the building and simulation times) as Simulation 1.

For a quick check, of the results, use the `describe()` method, and look at the first few spikes:

```
esp.describe()
```

senders	times	
count	1593.000000	1593.000000
mean	25.497175	501.450345
std	14.433182	287.833542
min	1.000000	13.200000
25%	13.000000	249.200000
50%	25.000000	501.600000
75%	38.000000	751.900000
max	50.000000	998.100000

```
esp[:5]
```

senders	times	
0	18	13.2
1	11	13.5
2	41	13.6
3	44	13.8
4	26	14.0

You should see exactly the same spike times.

Plotting the results Now create a raster plot showing the spike times of the excitatory neurons in blue and the inhibitory neurons in red, by using `times` as x-axis values and `senders` as y-axis values.

Hint: Check the minimum and maximum values of the `senders` gids and offset the plotted values suitably to get a tidy graph.

Create spike time histograms, using the `plt.hist()` function with a suitable number of bins. Use `histtype=step` to combine PSTHs for excitatory and inhibitory cells.

1.2.3 Simulation 3: Exploring parallel simulation

Run the same simulation as above with `t_sim=250.` (to save time) with different values of `n_threads` (start with 1, 2, 4) to see if you can save time using several cores on your computer. In some cases, it actually may help to use more threads than your computer has cores. You should not do much else on your computer when running on as many or more threads than your computer has cores.

1.3 Task 2: Reproducing Brunel

In this task, you shall try to reproduce the results presented by Brunel in his paper. In particular, you shall simulate the network for the four parameter combinations marked in Fig. 7 of the paper and see if you can obtain responses similar to those displayed in Fig. 8 of the paper.

For each of the four simulations below, you shall

- record from `N_rec=1000` neurons (each excitatory and inhibitory)
- simulate for 1 second biological time
- plot raster plots showing the responses of 40 excitatory and 10 inhibitory neurons for $800ms < t < 1000ms$
- plot spike-time histograms with bin width $dt = 0.1$ ms for both excitatory and inhibitory neurons, for the same time interval; the histograms shall use data from all recorded neurons

Note that $\eta = \nu_{\text{ext}}/\nu_{\text{thr}}$.
Carefully compare your results with Fig 8 in Brunel (2000):

- Are your results sufficiently close to Brunel's to conclude that you have reproduced Brunel's results?
- If not, describe the differences!
- Are differences larger for some case(s) than for others?
- *Carefully* read section 2 "The Model" in the paper and compare with the simulation code.
  - Is the code a faithful implementation of the model?
  - If not, can you make the implementation faithful?

1.4 Task 3: Turning off the input, and cranking up the weights

In this task, you will turn off the Poisson input driving all neurons in the beginning after 400 ms and continue simulation until 1 second is simulated.

To this end, modify your code so that you can set the `stop` parameter of the Poisson generators. The stopping time should be passed to `sim_brunel_delta()` as keyword argument `input_stop`. You should also incorporate any corrections made as part of Task 2.

Plot spike rasters and histograms for the full second; use 2ms bin widths for the histograms.

1. Simulate with the parameters for case C.
2. Simulate once more, with parameters for case C, but with $J = 1.0$, i.e., with tenfold synaptic weights.
3. Describe your observations and attempt to analyse their causes!
4. Why is their a highly synchronous volley of spikes at the beginning of the simulation?
5. How could you avoid that volley?

1.5 Task 4: Recuding the network size

Simulations of the full Brunel model take more than a minute (on my machine). For fast exploration, it would therefore be attractive to simulate a smaller network. Network size is determined by the `order` parameter according to the following rule:

- $N\_E = 4\times$ `order`
- $N\_I =$ `order`

1. Reduce the network size to 1/10 of the original size (i.e., `order=250`) and simulate case C using the original $J = 0.1$; reduce the number of recorded neurons to 250.
2. Plot a raster plot and histogram as previously.
3. Describe the result and compare it to the full model.
4. Try to adjust parameters (J, g, η) to obtain (approximately) the same firing rate and firing pattern as for the full scale model.
    - *Note*: To avoid problems with initial transients, compute firing rates only over $500ms < t < 1000ms$.
5. Discuss your experiences!

1.6 Task 5: A network with plastic synapses

In the final task of this exercise, explore the network with plastic excitatory-excitatory synapses. In this case, we want to stop the simulation regularly, so that we can read out the synaptic weights. We therefore need a function building the network and a separate function to simulate it.

1.6.1 Preparations

Note: If you feel comfortable with object-oriented programming in Python, feel free to deviate from the detailed implementation suggested below and choose an approach closer to the one suggest in Sec. 8 of *NEST by Example*.

1. Add a function `build_brunel_delta_plastic()` to your `brunel_delta` module.
    1. Start from the `sim_brunel_delta()` function an remove all simulation-related parameters and code.
    2. Create a new synapse model `excitatory_plastic` and use it to connect the excitatory-excitatory connections.
    3. Excitatory-excitatory and excitatory-inhibitory synapses shall have randomized weights, uniformly drawn from $[0.5J_E, 1.5J_E)$. See *NEST by Example*, Sec. 6 and 7 for suggestions.
    4. The initial membrane potential of all neurons shall be randomized uniformly over $[-20mV, 20mV)$.
    5. The function shall also accept parameters `alpha` (default 2.02), `lambd` (default 0.01; `a` dropped to avoid conflict with keyword `lambda`), and `Wmax` (default 3.0).
    6. The function shall return the following four lists:
        1. excitatory spike detector (single element list)
        2. inhibitory spike detector (single element list)
        3. all excitatory nodes
        4. all inhibitory nodes
2. Add a function `sim_brunel_delta_plastic()` to your `brunel_delta` module.
    1. This function should be based on the simulation part of `sim_brunel_delta()`.
    2. It should take the following arguments:
        1. the simulation time
        2. the excitatory and inhibitory spike detector lists returned by the build-function

3. a list of all excitatory neurons for which you want to collect synaptic weights (outgoing synapses)

3. It should return the same spike-data dataframes as `sim_brunel_delta()` and in addition a NumPy array containing the weights of all outgoing synapses from the specified neurons at the end of the simulation.

1.6.2 Usage

You can now use these two functions like this:

```
esd, isd, enrn, inrn = build_brunel_delta_plastic(order=250, N_rec=250, g=5., eta=1.25, J=1., V_reset=1
                                                  lambda=0.01, alpha=2.02, Wmax=3.)
esp, isp, e_wgts = sim_brunel_delta_plastic(200., esd, isd, enrn[:250])
```

In that example, we build a small network and then simulate for 200 ms. `e_wgts` will contain the weights of all outgoing synapses formed by the first 250 excitatory neurons.

The following code will simulate `n_step` times for `t_step` and collect weights after each simulation, as columns of `e_wgts`, while `t_wgts` will contain the times at which the weights were measured. `ini_wgts` are the weights at the beginning of the simulation.

```
esd, isd, enrn, inrn = build_brunel_delta_plastic(order=250, N_rec=250, g=5., eta=1.25, J=1., V_reset=1
                                                  lambda=0.01, alpha=2., Wmax=3.)

t_step = 200.
n_step = 10

ini_wgts = nest.GetStatus(nest.GetConnections(source=enrn[:250], synapse_model='excitatory_plastic'), 'w
e_wgts = np.nan * np.ones((len(ini_wgts), n_step+1))
e_wgts[:, 0] = ini_wgts
t_wgts = t_step * np.arange(n_step+1)

for k in range(10):
    esp, isp, e_wgts[:, k+1] = sim_brunel_delta_plastic(200., esd, isd, enrn[:250], print_report=False)
```

1.6.3 Simulations

Perform simulations varying the learning rate λ and the depression-facilitation-ratio α. For each parameter combination

- simulate for 10 times 200 ms and measure the weight distribution after each 200 ms
- plot historgrams of all weight distributions in a single diagram
- plot a raster plot over the full duration of the experiment
- plot a spike-time histogram over the full duration of the experiment

Describe and discuss your observations!