Computational neurodynamics

Exercise Sheet 4 (**Assessed**)
Dynamical complexity

A good experiment can be replicated. Your task is to replicate one of the experiments described in Topic 9 (Dynamical Complexity). The experiment involves the generation of modular small-world networks of spiking neurons, and an assessment of their dynamical complexity.

Question 1.

Write Python code that will generate small-world modular networks of Izhikevich neurons. The basic method is the rewiring procedure described in Topic 8 (Modular Networks). But for details of the network construction, follow the description of the experiment in Topic 9 (Dynamical Complexity). Note that the networks need a separate inhibitory population. Use the neuron parameters from the lecture notes for excitatory and inhibitory Izhikevich neurons. You can use the `IzNetwork` class from the course repository at https://www.github.com/pmediano/ComputationalNeurodynamics .

Generate a network for p = {0, 0.1, 0.2, 0.3, 0.4, 0.5}. For each of these networks:

a) Generate a plot of the matrix connectivity.

b) Generate a raster plot of the neuron firing in a 1000ms run.

c) Generate a plot of the mean firing rate in each of the eight modules for a 1000ms run. Downsample the firing rates to obtain the mean by computing the average number of firings in 50ms windows shifted every 20ms. (This should yield 50 data points for each module.)

Your results should be consistent with those in the Topic 9 slides.

Question 2.

Following the description in the slides, write Python code that generates a network as in Question 1, for a random value of p between 0 and 1, then runs it for 60s while recording the (downsampled) mean firing rates in each of the eight modules. Discard the first second's worth of data to avoid effects from transient states. (This should yield eight time series of 2950 data points each.)

Now write Python code to calculate the *multi-information* of such a set of time series. In the lectures we saw how to estimate I(S) for a system of discrete variables, but the module firing rates are real-valued – and unfortunately integration of real-valued data is hard to estimate. To ignore the details of this estimation, you are allowed (and encouraged, but not restricted) to use the Kraskov2 multi-information estimator in the

Java Information Dynamics Toolkit (JIDT). You can find examples and documentation at the project's website,

http://jlizier.github.io/jidt/

Perform at least 20 trials as described above, calculate the integration of the network and plot the results. This should resemble the plot in the Topic 9 slides.

What to Submit

You should submit all your Python code, plus your plots in any vector graphics format. Also submit a README.txt file which specifies the structure and dependencies of the functions for each question, as well as the file names of the plots associated with each question. The top-level function for each question should be in files Q1.py and Q2.py.

You should have at least the following files:
- README.txt
- Q1.py
- Q2.py
- The plots requested in question 1 (preferably in a vector graphics format)
- Whatever additional Python scripts/functions you find appropriate

Your code should have no dependencies outside the submitted zip, with the exception of the following packages/libraries: numpy, scipy, matplotlib, bct, jpype and JIDT. Include all other files necessary, i.e. even the ones we provide like `IzNetwork`. You can write any additional information or comments you think we should consider in your README.

Working in Groups

You are encouraged to work in groups of up to three students. But you may work in pairs or alone if you prefer.

Marking Scheme

70% of the marks will be allocated to Question 1, and 30% to Question 2. Note that this is not a direct reflection of the relative difficulty of the two questions. Rather, it allows students to gain extra marks on this coursework for extra effort (on Question 2), while allowing those who want to balance their workload with other courses to put in less effort (on Question 2) without significantly compromising their grade.

In addition, a fraction of 5% of the marks can be deducted on the basis of unclear code, poor coding practice or excessively intricate solutions. An extra 5% may be awarded on the basis of creative solutions, meaningful discussions and the extension of the analysis to other complexity measures.

Frequently Asked Questions

QUESTION: Isn't the description of how you build a modular small-world network in the Topic 8 slides slightly different from the description in the relevant Topic 9 slides?

ANSWER: Yes! According to the notes from Topic 8, to build a modular small-world network with n nodes, m edges, and C communities, you first construct C disconnected communities with n/C nodes each, where each community has m/C random symmetric edges. According to the exercise specification, you should try to reproduce the results in Topic 9, where "each module has 1000 randomly assigned one-way excitatory-to-excitatory connections (before the rewiring phase)" and "connections are not symmetrical, so we have a directed network". For the exercise, you should follow the second description, from Topic 9.

QUESTION: What scaling factors / weights / conduction delays should we use?

ANSWER: These are all specified on a slide in Topic 9 (Dynamical Complexity).

QUESTION: What are layers for? How should we use them?

ANSWER: The layers (used in `IzNetwork`) are only there as a software engineering aid to help organise large networks. You could put all the neurons in a single layer. But in the case of the exercise, a natural organisation is to put all the excitatory neurons in one layer and all the inhibitory neurons in another. The modules (which comprise excitatory neurons) are then all within a single layer.

QUESTION: My code takes a long time to run! Is there a way to make it faster?

ANSWER: Yes, it probably does take a while to generate all the data for Question 2. You can modify IzNetwork if you want to, and structure your simulation appropriately to make it faster. The second best solution is to start the coursework early. Also, since all of the provided Python code is single-threaded, feel free to use any multi-threading library of your choice to parallelise it (you can use multi-threading libraries although they're not explicitly listed as allowed in the *What to Submit* section).

QUESTION: Can we use any other libraries not mentioned in the list above?

ANSWER: You can use any multi-threading library to speed up your simulations, but the code you submit must be able to run without them. You may also use any library other than JIDT to calculate information-theoretic measures, but in that case either A) the library must be pip-installable or B) you must provide the source in the submitted zip.