SYDE 556/750
Simulating Neurobiological Systems
Assignment 4

Due date: March 24, 2020

Important Notes – Please Read Carefully

- This assignment is worth 10 marks (10% of the final grade). The number of marks for each question is indicated in brackets to the left of each question.
- Please use Python 3 along with the `numpy` and `matplotlib` libraries to solve these assignments. In particular, fill out this Jupyter Notebook template:

  `https://github.com/astoeckel/syde556-w20/blob/master/assignments/assignment_04/syde556_assignment_04_template.ipynb`

- Clearly label any plot you produce, including the axes. Provide a legend if there are multiple graphs in the same plot.
- You won't be judged on the quality of your code, but writing reusable functions will greatly simplify this and future assignments.
- Make sure to execute the Jupyter command "Restart Kernel and Run All Cells" before submitting your solutions. You will lose marks if your code fails to run or produces results that differ significantly from what you've submitted.
- Rename the completed notebook to `syde556_assignment_04_<STUDENT ID>.ipynb` and email it to `astoecke@uwaterloo.ca` using your University of Waterloo email account, and make sure to include "[SYDE 556/750]", "Assignment 4" and your student ID in the subject line. The deadline is at `23:59 EST` on March 24, 2020.
- There is a late penalty of one mark per day late. You may be at most seven days late.
- **For this assignment, you must use Nengo**. Feel free to look through the examples folder and/or the tutorials on the Nengo website before doing this assignment.

1 Building an ensemble of neurons

Make a new model with a single ensemble of neurons. It should have 100 neurons, and represent a 1-dimensional space. The intercepts should be between -1 and 1, and the maximum firing rates should be between 100 Hz and 200 Hz. τ_{RC} should be 20 ms and τ_{ref} should be 2 ms.

- You don't need to run the model over time for this question.

[0.5] a) *Tuning curves.* Plot the population tuning curves. Plot the representation accuracy plot ($x - \hat{x}$). Compute and report the RMSE.

- Feel free to have a look at this Nengo tutorial to solve this question.
- Create a "dummy" identity connection from the ensemble to itself to have Nengo compute the population identity decoders. Then, you can use `nengo.utils.connection.eval_point_decoding` to compute $\hat{x}$. Nengo will take care of computing the decoders under noise. You don't have to use your code from previous assignments, but, of course, feel free to do so. In this case, compute the decoders under the assumption of noise with a magnitude of $\sigma = 0.1 \cdot 200\,\text{Hz}$.

[0.5] b) *RMSE and radius.* Compute the RMSE for (at least) the four different radii 0.5, 1, 2, and 4.

- Nengo will automatically rescale the intercepts as the radius increases, so you don't have to worry about changing the intercepts as you had to in Assignment 3.

[0.5] c) *Discussion.* What relationship between the radius and the RMSE do you observe? Explain why this is the case.

[0.5] d) *RMSE and refractory period.* What happens to the RMSE and the tuning curves as τ_{ref} changes between 1 ms to 5 ms? Plot the tuning curves for at least four different τ_{ref} and produce a plot showing the RMSE over τ_{ref}.

[0.5] e) *RMSE and membrane time constant.* What happens to the RMSE and the tuning curves as τ_{RC} changes between 10 ms to 100 ms? Plot the tuning curves for at least four different τ_{RC} and produce a plot showing the RMSE over τ_{RC}.

[0.5] f) *Discussion.* Discuss the last two results. Describe what happens to the tuning curves as τ_{ref} and τ_{RC} change. Explain why the change in tuning curve shape influences the RMSE in the way you observe.

2 Connecting neurons

Make a second ensemble of spiking neurons. It should have the same parameters as the first ensemble of neurons (from the first question), but have only 50 neurons in it. Connect the first ensemble to the second such that it computes the identity function, using a post-synaptic time constant of 10 ms. Create a step-function input that is a value of 1 for $0.1 < t < 0.4$ seconds, and otherwise is zero.

- You must use a Nengo `Node` object to feed an arbitrary Python function into a Nengo network. You can use a Python `lambda` to write this in a very concise way. For example

```
nd_input = nengo.Node(lambda t: 0.0 if t < 0.5 else (1.0 if t < 1.0 else 0.0))
```

will create a step function that is 1 for $0.5 < t < 1.0$ and 0 otherwise. Instead of using a lambda, you can also use the `Piecewise` process built into Nengo:

```
nd_input = nengo.Node(nengo.processes.Piecewise({
    0.5: 1.0,
    1.0: 0.0
}))
```

[1] a) *Computing the identity function.* Show the input value and the decoded values from the two ensembles in three separate plots. Run the simulation for 0.5 s.

[0.5] b) *Computing an affine transformation.* Make a new version of the model where instead of computing the identity function, it computes $y(t) = 1 - 2x(t)$. Show the same graphs as in part (a).

3 Dynamics

Build a neural integrator. This consists of one ensemble, one input, a connection from the input to the ensemble, and a connection from the ensemble back to itself. The ensemble should have 200 neurons and the same parameters as in question 1.

Important: Use two different time constants in your network. The post-synaptic time constant of the recurrent connection is $\tau = 50$ ms, and the post-synaptic time constant of the input is 5 ms. Having two different time constants violates what we assumed for deriving principle three; use the longer time constant τ (i.e., τ from the recurrent connection) when computing A' and B'. Part of this assignment is to explore how having two different time constants affects the quality of the result compared to what we saw in the lecture.

To be an integrator, the desired dynamical system is $\frac{\mathrm{d}x(t)}{\mathrm{d}t} = u$, where x is the state of the system and u is the input.

For all probes, use a synapse of 10 ms. Explicitly plot the ideal, which can help when answering the questions.

[0.5] a) *Transforming the dynamical system.* Rewrite the linear dynamical system describing the integrator in terms of $\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$, i.e., write down the matrices $\mathbf{A}$ and $\mathbf{B}$. What are the matrices $\mathbf{A}'$ and $\mathbf{B}'$ we have to use when implementing this system using the recurrent connection post-synaptic filter?

- You may assume that the input $\mathbf{u}$ and the state $\mathbf{x}$ are d-dimensional, i.e., you may write down the equations as general as possible. In the remainder of this question we assume $d = 1$.

[0.5] b) *Integrator using spiking neurons.* Show the input and the value represented by the ensemble when the input is a value of 0.9 from $t = 0.04$ to $t = 1.0$ (and 0 for other times). Run the simulation for 1.5 s.

- For this and the following questions, we consider the scalar case, i.e., the dimensionality n of x and u is one.
- We're going to run this simulation with various input functions and ensemble parameters in the following questions. It is highly recommended that you write a function that takes ensemble parameters and the input function as arguments and produces all plots you need.

[0.5] c) *Discussion.* What is the expected ideal result, i.e., if we just mathematically computed the integral of the input, what would we get? How does the simulated output compare to that ideal?

- When you write down the integral of the step function, you have to distinguish between three cases. Your function should be a *definite* integral of the form

$$\int_0^t u(t')\,\mathrm{d}t' = \begin{cases} \dots & \text{if } t < 0.04\,, \\ \dots & \text{if } 0.04 \le t < 1\,, \\ \dots & \text{if } 1 \le t\,. \end{cases}$$

- Feel free to plot the ideal on top of your simulation result in your answer for the previous question.
- You can use the `numpy` function `np.cumsum` to compute the ideal integral numerically (don't forget to scale by `dt`). However, still write down the integral mathematically to answer this question.

[0.5] d) *Simulation using rate neurons.* Change the neural simulation to rate mode. Re-run the simulation in rate mode. Show the resulting plots.

- Pass the extra parameter `neuron_type = nengo.LIFRate()` to the `Ensemble` constructor to switch an ensemble to rate mode.

[0.5] e) *Discussion.* How does this compare to the result in part (b)?

[0.5] f) *Integration of a shorter input pulse.* Returning to spiking mode, change the input to be a value of 0.9 from $t = 0.04$ to 0.16. Show the same plots as before (the input and the value represented by the ensemble over 1.5 s).

[0.5] g) *Discussion.* How does this compare to (b)? Does it work as intended? If not, why is it better or worse?

[0.5] h) *Input ramp.* Change the input to a ramp input from 0 to 0.9 from $t = 0$ to $t = 0.45$ (and 0 for $t > 0.45$). Show the same plots as in the previous parts of this question.

[0.5] i) *Discussion.* What does the ensemble end up representing, and why? What is the (ideal) equation for the curve traced out by the ensemble?

- As above, make sure to write down the *definite* integral $\int_0^t u(t')\,\mathrm{d}t'$. This means that there will be no integration constant in your result.

[0.5] j) *Sinusoidal input.* Change the input to $5\sin(5t)$. Show the same plots as before.

[0.5] k) *Discussion.* What should the value represented by the ensemble be? Write the equation. How well does it do? What are the differences between the model's behaviour and the expected ideal behaviour and why do these differences occur?

- Again, write down the *definite* integral $\int_0^t u(t')\,\mathrm{d}t'$. There is no integration constant.

[+1] l) ☆ *Bonus question.* Implement a nonlinear dynamical system we have not seen in class (and that is not in the book). Demonstrate that it's working as expected.

- In order to gain marks for this question you must
  (i) write down the differential equation of the nonlinear system,

(ii) write down the transformed equations that implement the system using the post-synaptic filter,

(iii) implement the system in Nengo using spiking neurons,

(iv) show that the simulation results match the expected system behaviour at least qualitatively.