PSY 5939-U06: Introduction to Computational Cognitive Neuroscience
Homework #4: Decision Making

In this homework assignment, you will use Brian2 to simulate a simple decision-making neural circuit. You should submit your results as **a single jupyter notebook** via email. Separate the different questions with titles by inserting a markdown cell and then typing "# Question 1", "# Question 2", etc, and running the cell. Your explanations of results and answers to questions should be included in **markdown cells** (not as comments) within the notebook.

The model that you should use in this assignment is exactly the same that we used for programming tutorial #5; the slides from that tutorial can be found here. Your simulations should use $\Delta t = 1ms$ and run for $2,000ms$. Thus, you should change the size of the manual inputs to each neuron from $1,000$ to $2,000$.

1. We will start by expanding the model to obtain behavioral choices and response times from activity patterns in the MSNs. That is, we will assume that activity in MSNs is correlated with behavioral choices. A stronger activity in MSN_0 leads to "left" responses, whereas a stronger activity in MSN_1 leads to "right" responses. There are many possible decision rules linking neural activity to overt behavior, but here we will use one of the simplest possibilities: a threshold on the number of spikes generated by each of the neurons. That is, when one of the MSNs reaches a total of S action potentials within a trial, the response represented by that MSN is recorded as the behavioral choice ("left" for MSN_0; "right" for MSN_1) and the response time will be the time of the Sth spike of that neuron plus a baseline response time of $200ms$, representing the time that it takes for the MSN to influence downstream motor neurons and for the behavioral response to be executed.
   One reason to use this decision rule is that it is relatively easy to implement in Brian2. You only have to follow these steps:

   - Rather than continuously recording neuronal variables with the function `StateMonitor`, record the spikes generated by each MSN using the following line of code:

     ```
     spikes_MSN = SpikeMonitor(MSN)
     ```

     After you run a simulation, `spikes_MSN.t` will contain the spike times for all MSNs, and `spikes_MSN.i` will contain the index of the MSN producing each spike (i.e., 0 or 1). That is, `spikes_MSN.t[spikes_MSN.i==0]` should return an array with the spike times of MSN_0, whereas `spikes_MSN.t[spikes_MSN.i==1]` should return an array with the spike times of MSN_1.

- In our simulations, we will use the spike threshold $S = 5$. The time at which MSN_0 reaches its $5th$ spike is recorded in `spikes_MSN.t[spikes_MSN.i==0][4]`, which is the $5th$ element (index $= 4$) of the array with that neuron's spike times. Note that this is a record of the time when the network makes a decision, with the timer starting at the beginning of the simulation. If you want to record the time when the network makes a decision, taking as a reference the stimulus onset, then you must subtract the stimulus onset time to this value. We can record the behavioral choice and response time with the following two lines of code:

  ```
  RT = 100 + 1000*np.amin([spikes_MSN.t[spikes_MSN.i==0][4],
                           spikes_MSN.t[spikes_MSN.i==1][4]])
  choice = np.argmin([spikes_MSN.t[spikes_MSN.i==0][4],
                           spikes_MSN.t[spikes_MSN.i==1][4]])
  ```

  The first line compares the time of the $5th$ spike in MSN_0 and MSN_1, keeps the value that is smaller (i.e., faster to reach threshold) using the function `np.amin`, converts the value from seconds to ms, and adds the baseline RT of $200ms$ ($-100ms$, which is the stimulus onset time in our simulations). The result is the response time of the network.
  The second line also compares the time of the $5th$ spike in MSN_0 and MSN_1, but returns the index of the neuron with a shorter time. The result is the actual behavioral choice, with a value of 0 when the choice is "left" and a value of 1 when the choice is "right"

We will start by running a single simulation, to make sure that everything is working the way it should. Set $\bar{g}_{GABA} = 80nS$, $\bar{g}_{AMPA_{LARGE}} = 100nS$ and $\bar{g}_{AMPA_{SMALL}} = 80nS$. Set the noise in the MSNs to $\sigma = 0mV$ for now. Record the choice and RT as indicated in the previous box and print them to screen using `print choice` and `print RT`. Make absolutely sure that the results of this simulation are correct before continuing with the rest of this assignment.

2. In the second simulation, you should increase the noise level to $\sigma = 5mV$. Then, run the simulation 100 times, and each time store both the network's choice and RT. You will have to create two empty numpy arrays called `choice` and `RT` before running the simulations, and then fill them with the values that you obtain in each simulation (`choice[0]` and `RT[0]` in the first simulation,`choice[1]` and `RT[1]` in the second simulation, and so on). If you don't remember how to work with numpy arrays, go back to the slides from programming tutorial #2, which can be found here.
WARNING #1: This simulation (and the following) will take a long time to run. Make sure to run it first with only a few repetitions (e.g., 5 instead than 100), so that you can check that it is working. My recommendation is that you use a `for` loop to run this simulation, because that way you can leave it running in the background and do something else. However, this is not mandatory. You can finish the simulation without a `for` loop, but you should know that: (1) this increases the likelihood of making a mistake, and (2) this means spending an afternoon doing mindless work.
WARNING #2: Because there is a random component to this simulation, you might get slighly different results from those of other students.

Once your simulation is finished, answer the following questions:

(a) Print the proportion of correct responses to the screen using `np.mean(choice==0)` (remember: the correct choice here is "left", indexed by 0, because we are presenting our network with a "green" stimulus). What is the proportion of correct responses and errors? Why is the network making errors, when we hardwired it to respond "left" whenever we presented it with a "green" stimulus?

(b) Plot the distribution of correct response times using the "histogram" function from matplotlib:

```
plt.hist(RT[choice==0],normed=True,facecolor='gray',edgecolor='black')
plt.ylabel("Proportion")
plt.xlabel("Response␣time␣(ms)")
```

You can change the color of this plot by changing the value of `facecolor` and `edgecolor`. What is the range of RTs? What is the mode?

(c) What single parameter can you change in this simulation to obtain both more variable RTs (i.e., a wider range of values) and choices (i.e., more errors)?

3. Re-run the previous simulation, but set $\bar{g}_{GABA} = 160nS$. How does this affect the results of your simulation? Do you see any changes in the proportion of correct responses? If the answer is "yes," then explain why. Do you see any changes in the distribution of RTs? If the answer is "yes," then explain why.

4. For the final simulation, we will imagine a case in which we present a stimulus to the network that is not a pure "green" or "red" color, but a more ambiguous color between those two extremes. Re-run the simulation in point #2, but with an input to RSN$_0$ should of $360pA$ and an input to RSN$_1$ of $200pA$.

   (a) Compare the results to those obtained in the simulation from point #2. How does the new input pattern affect the distribution of RTs (compared to the simulation in point #2)? How does it affect the proportion of correct responses? Explain why the change in input patterns produced the observed change in RTs and proportion of correct responses.

   (b) Given the discussion about the speed-accuracy tradeoff in Wang (2008). What would be a way to increase proportion of correct responses in this simulation? How would that affect the response time distribution?