PSY 5939-U06: Introduction to Computational Cognitive Neuroscience
Homework #5: Reinforcement Learning

In this exercise, you will expand the decision-making network that you built in Homework #4 and change its synaptic weights (for the connections from RSNs in visual cortex, representing "red" and "green" stimuli, to MSNs in the striatum, representing motor choices) using a reinforcement learning scheme. 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 #7; the slides from that tutorial can be found here (recently updated to correct some mistakes!). The setup for your simulations will be exactly the same used in the programming tutorial.

1. We will start by recording a baseline of how the network responds before learning. To do this, set all the connection weights W to 0.3 and **do not change them using reinforcement learning** in this first set of simulations. That is, this simulation will be pretty much the same as those that you already ran for Homework #4. Run 100 simulations in which you randomly present the red or green stimulus to the network, and each time store both the network's choice and RT as you did in Homework #4. You will have to create two empty numpy arrays called `choice_pre` and `RT_pre` before running the simulations, and then fill them with the values that you obtain in each simulation (`choice_pre[0]` and `RT_pre[0]` in the first simulation,`choice_pre[1]` and `RT_pre[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: This simulation (and all 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.
   Once your simulation is finished, print the proportion of correct responses and make sure that you get the results that you would expect, given the fact that all connection weights in the network are set to the same value (0.3).

2. Now you should train the network to respond "red" when presented with a red stimulus, and "green" when presented with a green stimulus, using the reinforcement learning scheme introduced in Programming Tutorial #7. Rather than running the learning algorithm for a single trial, as you did in the tutorial, you will have to run it for 160 trials, each time updating both the values of the connections weights W in the actor, and the reward values V in the critic. Use the function `print_trial_info` to print the results of each trial to screen.
   You should use numpy arrays to keep track of the RTs, the response accuracy (i.e., whether or not the choice in a trial was rewarded, represented by the variable`rw`), the RPE, the reward values V and the synaptic weights W. Once the simulation is finished, use the stored values to answer the following questions:

(a) Plot the mean RT and accuracy across training, by averaging these values in blocks of 20 trials. That is, you should create a line plot in which the first point is the average accuracy for trials 1-20, the second point is the average accuracy for trials 21-40, and so on. The x-axis should be labeled "Blocks of 20 trials" and numbered from 1 to 8. Do the same in a separate plot for RTs. What happened with accuracy and RTs across training? Do they reach a learning asymptote (i.e., the point where learning stops) at the same time? Regardless of whether your answer is "yes" or "no", explain the reason behind their equivalent or different asymptotes.

(b) Plot the reward values V for both stimuli ("green" and "red) across the 160 trials. These values are updated using an error-driven learning rule for the critic (the Rescorla-Wagner learning rule), just as in Programming Tutorial #6. In that tutorial, we created a plot of how the reward value of a stimulus changes when it is paired with reward. How are the shapes of learning curves obtained in this homework different from those obtained during Programming Tutorial #6? Explain the reason behind these differences.

(c) Plot the values of the synaptic weights W across trials in a line plot, making sure that you label them correctly (i.e., four different lines of different color, each color representing a specific W that should be labeled somewhere in the figure). In a different plot, display the RPE across trials. Using the information in these plots and in the plot that you created for question 2-(a), explain how RPEs drive learning of both V and W in this actor-critic reinforcement learning scheme. Make sure to explain how these different values are related to one another, and when and why learning seems to slow down (or stop) during the simulation.

3. Finally, you should record the network's behavior after learning. Repeat the simulation from point #1 above, but this time using the final values for the weights W obtained in the learning simulation from point #2. Remember to turn off learning in this simulation, you only want to record what the network does once it has learned the task. Keep the results of this new simulation in numpy arrays called `choice_post` and `RT_post`. When you are done, use the values that you stored in `choice_pre`, `RT_pre`, `choice_post` and `RT_post` to answer the following questions:

(a) Create a bar plot comparing the average accuracy before and after learning. Explain how learning affected accuracy.

(b) Create a plot comparing the distribution of RTs (correct and incorrect) before and after learning. Use the "histogram" function from matplotlib, as you did in Homework #4 (but note that now you will plot both correct and incorrect RTs, whereas before you plotted only correct RTs). Regardless of how you decide to plot the two histograms (one for RTs before learning, and one for RTs after learning), make sure that they use the same scale. You can do this by telling matplotlib exactly how many "bins" to use when plotting the histogram:

```
bins = np.linspace(300, 1000, 20)
plt.hist(RT, bins, normed=True, facecolor='gray', edgecolor='black')
plt.ylabel("Proportion")
plt.xlabel("Response␣time␣(ms)")
```

Here, we use the function `np.linspace` to create 20 evenly-spaced bins, going from $300ms$ to $1,000ms$ (you might have to change this minimum and maximum, if you observe RTs below or above this range). We then use those bins to create the histogram. The plot will display the relative frequency of RTs in each bin.
Explain how learning affected the distribution of RTs. Does learning affect the average RT, the dispersion of RTs, or both?