Computational Neuroscience Problem Set 3

Consider the simplest kind of reinforcement learning, where on every trial an agent has to choose one of two possible actions, and each action is rewarded probabilistically. The environment carries no information about which action is more likely to be rewarded; the only information the agent has to optimize its behavior is the sequence of choices it has already made and the outcome of each of those choices (i.e., rewarded or not). You will construct two models of an agent performing this task. For each model, the agent will execute 361 trials, where the probabilities of reward associated with each action are periodically changed. The reward probabilities assigned to each action are given in two CSV files (prob_rwd_1.csv and prob_rwd_2.csv), i.e., prob_rwd_1.csv contains a list of 361 numbers giving the probability that choosing action 1 yields a reward on each trial. If you're using MATLAB, you can import these values with the csvread function.

(1) Q-Learning. In this model, the agent stores the value (Q) of each choice it might make. For this task, the agent stores separate values for actions 1 and 2, Q_1 and Q_2. We use a softmax function to convert these values into the probability of choosing an action (the "policy"):

$$\text{Prob}[\text{action1}] = \frac{1}{1 + e^{-\beta(Q_1 - Q_2)}}$$

After an action is chosen, the outcome of that choice is used to update the action values Q_1, Q_2 as follows:

$$RPE = reward - Q_{chosen} \qquad Q_{chosen} = Q_{chosen} + \alpha RPE$$

Here, *reward* is either 0 (not rewarded) or 1 (rewarded) and Q_{chosen} is the value of the action chosen on this trial; the value of the unselected action is not updated. The model parameters are the learning rate α and the "inverse temperature" β that controls how strongly the agent prefers the higher-value action.

(a) Simulate 361 trials of choices made by an agent using Q-learning, with the reward probabilities for each choice on each trial given in prob_rwd_1.csv and prob_rwd_2.csv. You will need to select initial values for the two actions (e.g., $Q_1 = Q_2 = 0.5$) and parameter values (e.g., $\alpha = 0.2$ and $\beta = 3$).

(b) Compute the probability that the model produces this sequence of choices given the outcomes of each choice. *Hint*: the reward probabilities you were given to simulate this behavior are now irrelevant; just take the trial outcomes (sequence of rewards) that emerged from your simulation as given. This probability is called the *likelihood* of the data (sequence of choices) given the model and trial outcomes. Give the answer in the form of a formula (or code, or pseudocode), not just a number.

(c) Write a function that computes the logarithm of the likelihood for a set of choices given the Q-leaning model and sequence of outcomes. This function should have the following arguments: list of choices, list of outcomes, initial action values, learning rate α, and inverse temperature β. It should output a single number, the log likelihood. The log likelihood is more numerically convenient than the likelihood itself (why?).

(d) Suppose you are given a sequence of choices and outcomes generated by an unknown agent. You can try to fit this behavior to a Q-learning model by finding model parameter values that maximize the likelihood (or equivalently, the log likelihood) of generating that data. Using your function from 1c, write another function that finds Q-learning parameters that maximizes the log likelihood of the data (sequence of choices and outcomes) you give it. Test this function on your own model-generated data. Did it find the correct parameters? *Hint*: don't write your own gradient ascent function if you don't have to. In MATLAB, you can use fmincon. Since this finds the minimum of the function you give it, multiply your log likelihood function by -1.
Here's the format for using fmincon:

x = fmincon(@(x) –*myLikeFunc*(choices, outcomes, Q_0, x(1), x(2)), [α_0, β_0])

Here, x is an array of parameters you wish to fit (the way this is written, x(1) and x(2) should take the place of α and β, respectively, in your likelihood function) and [α_0, β_0] are initial guesses for those parameter values (please don't use the known parameter values here!). If you wanted to, you could also maximize the likelihood over the initial values Q_0 in addition to the model parameters. Sadly, if you're using Python, you will probably have to write your own gradient ascent algorithm.

(e) Plot the log likelihood as a function of α and β, keeping the choices, outcomes, and initial action values fixed. This will be a 3D plot; in MATLAB you can use the surf function to make this plot. You can adjust the viewing angle for this plot with the view function. Does this plot say anything about why it might be challenging to identify values for α and β that maximize the likelihood? Try α values ranging from 0.01 to 0.6 and β values of 0.1 to 6.

(2) Actor-Critic. In this *very* simple version of an actor-critic model, the "critic" tracks the "state value" V, the global value (expected future reward) of being in the current state given the current policy (probabilities of choosing various actions). Unlike the Q-Learning model, the critic does not explicitly track the values of each action; only the overall value given the policy is stored. The critic uses the difference between the outcome of a trial and its stored state value, in conjunction with the actor's choice, to update the policy of the actor. Here, the actor's policy is defined by action-specific weights, w_1 and w_2, which are converted into action probabilities via the softmax function:

$$\text{Prob}[\text{action1}] = \frac{1}{1+e^{-\beta(w_1-w_2)}}$$

$$RPE = reward - V \qquad w_{chosen} = w_{chosen} + \alpha_w RPE$$

The outcome is also used to update the critic's state value:

$$V = V + \alpha_V RPE$$

This model has three parameters: two distinct learning rates, one for policy updates (α_w) and one for state value updates (α_V), plus the inverse temperature β.

(a) As you did with Q-learning, simulate 361 trials of choices made by an actor-critic agent, with the reward probabilities found in prob_rwd_1.csv and prob_rwd_2.csv. You will need to select parameter values and initial action weights.

(b) Write a function that computes the log likelihood for a set of choices given actor-critic parameter values and sequence of outcomes.

(c) Write a function that fits actor-critic parameter values to a sequence of choices and outcomes by maximizing the log likelihood. *Hint*: to fit the actor critic model, you should constrain the parameter values to a certain range—otherwise gradient ascent might actually fit negative learning rates! Constrain the learning rates to stay between 0 and 1, and β to be between 0 and 10. If you're using fmincon, that can be done as follows:

fmincon(@(x) –*myLikeF*(choices, etc.), [α_{V0}, α_{w0}, β_0], [], [], [], [], [0, 0, 0], [1, 1, 10])

(d) Using your actor-critic simulator, generate three sets of behavior using the same parameter values, then use your parameter-fitting function to fit parameters to each behavioral set. How closely does each fit resemble the true parameter values? How closely do the fits from each set resemble each other?

(e) Plot the log likelihood function given a particular set of choices and outcomes. Because there are 3 parameters, plot the likelihood surface as a function of just two parameters at a time, with the third parameter held fixed (perhaps at its true value, or at the value you obtained from fitting the behavior); you will generate 3 plots. Does this help explain the results you obtained in problem 2d?