Neural Modelling exercise 5: Learning to act

Hand-in by **Friday, 20.12.24 (midnight)** to neuralmodelling24@gmail.com

If you are handing in as a two person team, make sure to put both of your names on your solution (please hand in only one report per team). Besides your responses to the questions in text form, your submission should contain your code. You could e.g. link to a Github repo or submit your report as a Jupyter notebook.

Learning to act

We will program an actor-critic to learn a policy on the maze world we used for exercise 3 (see section 13.5 in Sutton & Barto for a full treatment of actor-critic). Our actor will be a direct actor, learning a table M of action propensities, for each state and each action. These get turned into an action through a softmax. Our critic, instantiating the value function V, will be the product of a state representation X and learned weights w, such that $V(s) = X(s) \times w$, for any state s. We will put a reward at location (1,1) into our maze environment.

- Fill in the actor-critic function in the provided template. We will learn through a number of episodes of interacting with the maze environment, an episode ends once the reward located at (1,1) is reached. Besides the framework for this, you will have to derive the update equation for the action propensities M (does require some steps, similar to what we once did in the tutorial) and the update equation for the ws parameterizing the value function (very straightforward). For our state representation X, we will simply us a one-hot encoding of the current state (though you should write your actor-critic in such a way as to accommodate different representations, as we will use a different one in the next exercise). Let the actor-critic run for 1000 episodes, keeping track of how much summed, discounted reward it received during each episode. Plot this evolution of reward acquisition.

- Now, we will use the successor representation (SR) to improve learning speed. Imagine the agent had ample exposure to the environment under a random policy, and acquired an accurate SR for each state. We thus create a value function as such: $V(s) = SR(s) \times w$. Use the provided function to compute the SR for all states, and use this as your state representation. This might require you to make some parts of your code more general, but the same code can run both versions of actor-critic in principle. Compare the evolution of reward acquisition for this new actor-critic. What do you see and how do you explain this? Plot the learned value function overlayed onto the maze.

- Next, we study the effects of re-learning the SR while learning the policy. Make two extensions to your code: Rather than always starting an episode

at the start location, use a function which returns a random (valid) location of the maze to start at. Additionally, also update the SR while interacting with the environment. You can do this with the provided function "learn_from_traj" after each episode, or during each episode if you programmed a temporal difference SR algorithm during exercise 3 (or want to do so now). If you use "learn_from_traj", note that it only updates the SR of the first state of the trajectory, so you have to write some extra code to also update the SR of the states along the trajectory. Plot the SR for a couple of states of your choice which showcase what it learned while specialising to the learned policy (and, as always, elaborate a bit on what you observe).

- How does a re-learned SR affect future policy changes? We will now move the reward location to (5, 5). Perform 1000 episodes of actor-critic (always starting from the original starting position again), with (a) the original SR from the random walk policy, and (b) the re-learned SR, which is tuned towards finding the reward at (1, 1). Let each learner run from scratch for 20 times and record how much reward was received in each episode. To get a smoother estimate of the reward trajectory, plot the averages over the 20 episodes for each type of learner. What do you see and how do you explain this (the difference may be somewhat subtle)?

- Lastly, we will study how value intialization can aid in the learning of a policy. The reward location is back at (1, 1), we always start at the original starting position, and use the SR from a random-walk policy as our representation. So far, we have initialized our weights w with 0. Experiment with different initializations along with both the 1-hot representation and the SR. Try a couple of representative points (like 4-5 different values) from 0 to 90 as your intialization. What do you observe, why do you think some values help while others hurt?