MOD202 Final Exam

Ciprian Bangu

May 27 2024

Short Answer Questions

1. When modelling a behavioral task as a reinforcement learning problem, the following are the four most imporant aspects to define and indentify:

   a) Firstly, one must define the state-space, s, of the task. Namely, the set of all possible states that the behavioral agent can be in in the task. This is parameter is defined by the experimenter when setting up the task. For example, in a maze task, each junction in the maze would be a state, as well as the start and end of the maze.

   b) Secondly, one must define the rewards the agent recieves at any given state, $r(s)$. Again, this is defined by the experimenter, and genearlly takes the form of pleasurable (or at least behavioraly reinforcing) stimuli for the agent. In the maze example, the reward could be food at the end of the maze. This is generally represented as a scalar value indicating the relative magnitude of the reward.

   c) Thirdly, one must define the action-space a, which consists of the possible acitons the agent can make in any given state. In the maze task, this amounts to leftward and rightward motion at the junctions. Again, this is defined by the experimental setup of the task (one could imagine a maze that requires the agent to move in 3 dimensions, for example).

   d) Finally, there is the agent's policy, $p(a|s)$, which is the probabiltiy that the agent takes a given action in a given state. This policy is internal to the agent, and is learned through experience in the task. It can be approximated, however, by fitting a model to the behavioral data generated by the agent.

2. The key difference between model-free and model-based RL is that in model-free RL agents learn the optimal policy and value functions directly from their interactions with the environment, while in model-based RL, agents learn a model of the environment and use this to guide their behavior.

   Model-based RL aims to solve the explosion of possible states in real-world settings. Since real-life scenarios contain a large number of states, it would take a model-free learner too long to explore all of them to be practical. A model-based learner, on the other hand, learns a model of the world and can thus make predictions about outcomes without having had to see the states before.

3. One exploitation/exploration strategy we see in RL is the ϵ-greedy strategy. In the ϵ-greedy strategy, the agent is mostly exploitative (ϵ is small), choosing the action that it currently believes maximizes its reward with probability $1 - \epsilon$. However, with probability ϵ, the agent

chooses randomly an action that it believes has a lower payoff. This strategy results in an agent that generally exploits the knowledge it has, but sometimes explores other options in case the environment has changed, or it has missed a possible higher payoff.

4. We would want to set a large learning rate for a given task in the case where we want convergence to be fast. The learning rate determines the magnitude of the update the agent makes to it's policy/value function. Thus, a large learning rate allows the agent to make large updates, leading to faster convergence. This could be useful in cases where there is no noise in the agent's assesment of the environment. If the agent is ceratin of their predicition and the true value of a certain state, for example, a learning rate of 1 would allow them to converge to the optimal policy in one step.

5. When the subject makes more mistakes, the ROC curve shifts down towards the identity line. This is because the ROC curve represents the percentage of correct responses; the identity line represents the 50%, or chance level. Thus, if the subject is making more mistakes, the ROC curve will be closer to the identity line (or below it), as the percentage of correct responses will be lower. Figure 1 below shows the ROC curve for an accurate subject, as well as the identity line (50% accuracy).

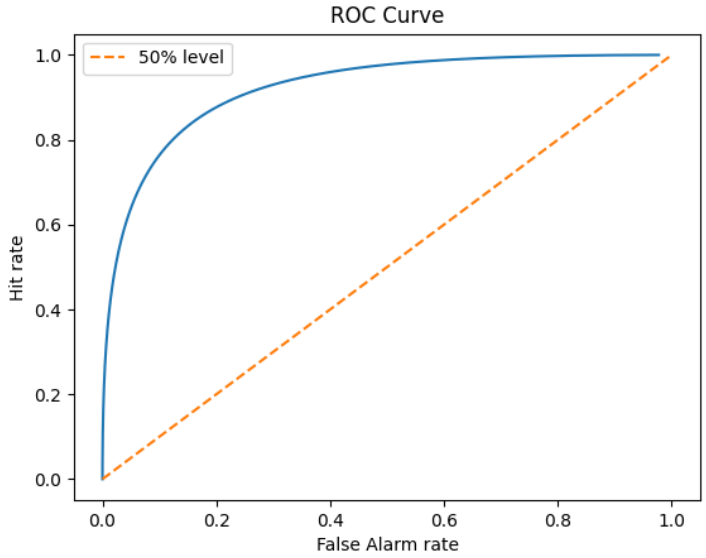

Figure 1: ROC Curve for an accurate subject

6. A neural decision threshold is the value of neural activity at which a decision in favor of one option over another is made. In a task with two options, we can imagine there being two neurons, each of which tuned to one of the options. In this case, we can take a linear combination of the responses of the two neurons, $x = \mathbf{a} \cdot \mathbf{r}$, where $\mathbf{a}$ is a vector representing the weights of the neurons and $\mathbf{r}$ is a vector representing the responses of the neurons. $\mathbf{a}$ is defined in such a way as to ensure that the line $x = \mathbf{a} \cdot \mathbf{r}$ cleanly seperates the responses from the two neurons. The decision threshold is then the value of x at which the decision is made in favor of one option over the other.

7. One piece of experimental evidence in favor of the 'drift-diffusion' model of decision making is the recording of Middle Temporal (MT) and Lateral Intraparietal (LIP) areas in the brains of monkeys perfroming a motion discrimination task. Different portions of MT neurons are tuned to be receptive to motion in different directions, and fire when presented with a dot-motion-stimulus. These are connected to LIP neurons, which seem to integrate the activity produced by the stimulated MT neurons. That is, the firing rate of the LIP neurons increases over time, reaching a threshold and then abruptly dropping. This activity is predicitive of the monkey's decision to saccade in a particular direction in the task, corresponding with the neural activity. Moreover, it is consistent with the drift diffusion model, which predicts that the decision is made when the integration of neural activity reaches a certain threshold.

8. Neural encoding amounts to representing the external world in terms of neural responses. In terms of probabilities, encoding can be thought of as the probability of a response, given a stimulus, i.e., $p(r|s)$. Conversly, neural decoding can be thought of as extracting the stimulus that caused a neural response, from the response itself. In terms of probabilities, decoding is the probability of a stimulus, given a response, i.e., $p(s|r)$. They can be related to each other with Bayes' theorem:

$$p(s|r) = \frac{p(r|s)p(s)}{p(r)}$$

9. The vector form of neuronal response can be written as:

$$\mathbf{r} = \mathbf{v} \cdot \mathbf{w}$$

where $\mathbf{r}$ is the vector of neural responses, $\mathbf{v}$ is the vector of stimulus values, and $\mathbf{w}$ is the vector of weights.

The neuron's receptive field correpsonds to the set of stimuli for which the neuron has a non-zero response to, i.e., which elicit a response from the neuron. The receptive field can be found via the tuning curve of a neuron, namely a function $f(s) = r_{max}[\mathbf{v} \cdot \mathbf{w}]_+$, which returns the (positive) response of a neuron to a given stimulus, scaled by the maximum response. The optimal stimulus is the one which elicits the largest response from the neuron, i.e., where $\mathbf{v}$ and $\mathbf{w}$ are aligned; where $\mathbf{v} = \mathbf{w}$

Considering each as unit vectors, and $\mathbf{v} = \mathbf{w}$:

$$\begin{aligned} &v \cdot w = w \cdot w = ||w||^2 = 1 \\ &f(s) = r_{max}[w \cdot w]_+ = r_{max}[1] = r_{max} \qquad \text{substituting into tuning curve} \end{aligned}$$

Thus, the neuron's response is maximised, i.e., the optimal stimulus is found, when the stimulus and receptive field are aligned.

10. The (single layer) perceptron can solve problems insofar as they are linearly seperable. That is, it can classify data into two categories as long as there exits a linear hyperplane that seperates the two. Notably, it cannot solve problems that require an XOR operation to seperate the classes.

1 Integrate-and-Fire Neuron

(a) The differential equation for the integrate-and-fire neuron is the following:

$$\tau \frac{dV(t)}{dt} = E_l - V(t) + RI(t) \tag{1}$$

Where $I(t)$ is a constant function, $I(t) = I_0$, and there is some threshold voltage V_{th}.
Finding how large I_0 must be such that the neuron starts spiking amounts to solving the differential equation for $V(t)$ to give an expression for V_{th} in terms of I_0.
i.e.,

$$\begin{aligned}
\tau \frac{dV(t)}{dt} &= E_l - V(t) + RI(t) \\
\frac{dV(t)}{dt} + \frac{V(t)}{\tau} &= \frac{E_l}{\tau} + \frac{RI_0}{\tau} && \text{collect Vs and substitute } I(t) = I_0 \\
e^{\frac{t}{\tau}} \frac{dV(t)}{dt} + \frac{1}{\tau} e^{\frac{t}{\tau}} V(t) &= e^{\frac{t}{\tau}} \left(\frac{E_l}{\tau} + \frac{RI_0}{\tau} \right) && \text{multiply by integrating factor } e^{\frac{1}{\tau}} \\
\frac{d}{dt} \left(e^{V(t)\frac{t}{\tau}} \right) &= e^{\frac{t}{\tau}} \left(\frac{E_l}{\tau} + \frac{RI_0}{\tau} \right) && \text{Rewrite LHS} \\
V(t) e^{\frac{t}{\tau}} &= \int e^{\frac{t}{\tau}} \left(\frac{E_l}{\tau} + \frac{RI_0}{\tau} \right) dt && \text{Integrate both sides} \\
V(t) e^{\frac{t}{\tau}} &= \int \frac{E_l + RI_0}{\tau} e^{\frac{t}{\tau}} dt && \text{rewriting since } I_0 \text{ is a constant} \\
\int \frac{E_l + RI_0}{\tau} e^{\frac{t}{\tau}} dt &= (E_l + RI_0) e^{\frac{t}{\tau}} + C && \text{Integrating} \\
\text{Thus, } V(t) &= E_l + RI_0 + Ce^{-\frac{t}{\tau}} && \text{Divide by } e^{\frac{t}{\tau}}
\end{aligned}$$

Then, getting the constant of integration, using $V(0) = V_0$:

$$\begin{aligned}
V(0) &= E_l + RI_0 + Ce^0 \\
V_0 &= E_l + RI_0 + C \\
C &= V_0 - E_l - RI_0
\end{aligned}$$

Substitution back into the equation for $V(t)$ gives:

$$V(t) = E_l + RI_0 + (V_0 - E_l - RI_0)e^{-\frac{t}{\tau}} \tag{2}$$

Therefore, I_0 must be large enough s.t: $\exists t$, $E_l + RI_0 + (V_0 - E_l - RI_0)e^{-\frac{t}{\tau}} \geq V_{th}$
Solving for any t, we have that:

$$\begin{aligned}
V_{th} &\leq E_l + RI_0 + (V_0 - E_l - RI_0)e^{-\frac{t}{\tau}} \\
V_{th} &\leq E_l + RI_0 + V_0 e^{-\frac{t}{\tau}} - E_l e^{-\frac{t}{\tau}} - RI_0 e^{-\frac{t}{\tau}} && \text{distributing } e^{-\frac{t}{\tau}} \\
V_{th} &\leq E_l + I_0(R - Re^{-\frac{t}{\tau}}) + V_0 e^{-\frac{t}{\tau}} - E_l e^{-\frac{t}{\tau}} && \text{collect } I_0 \\
V_{th} - E_l - V_0 e^{-\frac{t}{\tau}} + E_l e^{-\frac{t}{\tau}} &\leq I_0(R - Re^{-\frac{t}{\tau}})
\end{aligned}$$

Therefore, I_0 must be such that:

$$I_0 \geq \frac{V_{th} - E_l - V_0 e^{-\frac{t}{\tau}} + E_l e^{-\frac{t}{\tau}}}{R(1 - e^{-\frac{t}{\tau}})} \tag{3}$$

The actual value is given by the amount of time available to the neuron.

(b) Suppose that the current provided to the neuron is:

$$I(t) = \begin{cases} 0 & \text{for } t < t_0 \\ I_0 & \text{for } t_0 \leq t < t_0 + T \\ 0 & \text{for } t \geq t_0 + T \end{cases} \tag{4}$$

where T is the duration of the current pulse, i.e., the constant current I_0 is applied for T seconds.

By definition, when $T = 0$, the current is never turned on, so there is no value for I_0 that will make the neuron spike. The same applies when $T < 0$.

When $T \approx 0$ (when it is very small), the neuron's integration window is small, so the spiking threshold will only be reached when the current pulse is large - i.e., I_0 must be large such that $\tau \frac{dV(t)}{dt}$ is large in order for V_{th} to be reached. Specifically, when $T \to 0$, $\frac{dV(t)}{dt} \to \Delta V$ where ΔV is defined as $V_{th} - E_l$, since the neuron's voltage is just the resting potential, E_l, in the absence of a current. Therefore, I_0 must be such that:

$$\Delta V = \frac{E_l - V(t) + RI_0}{\tau}$$

$$\Delta V = \frac{RI_0}{\tau} \qquad \text{Simplifying since } E_l = V(t) \text{ in this scenario}$$

so:

$$I_0 = \frac{\Delta V \tau}{R} \tag{5}$$

When $T \to \infty$, the situation is the same as in (a), since the current is applied for an arbitrary amount of time. We know already that $V_{th} \leq E_l + RI_0 + (V_0 - E_l - RI_0)e^{-\frac{t}{\tau}}$. Therefore, we can solve for I_0 given T:

$$\begin{aligned} V_{th} &\leq E_l + RI_0 + (V_0 - E_l - RI_0)e^{-\frac{T}{\tau}} \\ V_{th} - E_l &\leq RI_0(1 - e^{-\frac{T}{\tau}}) + (V_0 - E_l)e^{-\frac{T}{\tau}} \\ V_{th} - E_l - (V_0 - E_l)e^{-\frac{T}{\tau}} &\leq RI_0(1 - e^{-\frac{T}{\tau}}) \end{aligned}$$

So,

$$I_0 \geq \frac{V_{th} - E_l - (V_0 - E_l)e^{-\frac{T}{\tau}}}{R(1 - e^{-\frac{T}{\tau}})} \tag{6}$$

Graphing the relationship, we get:

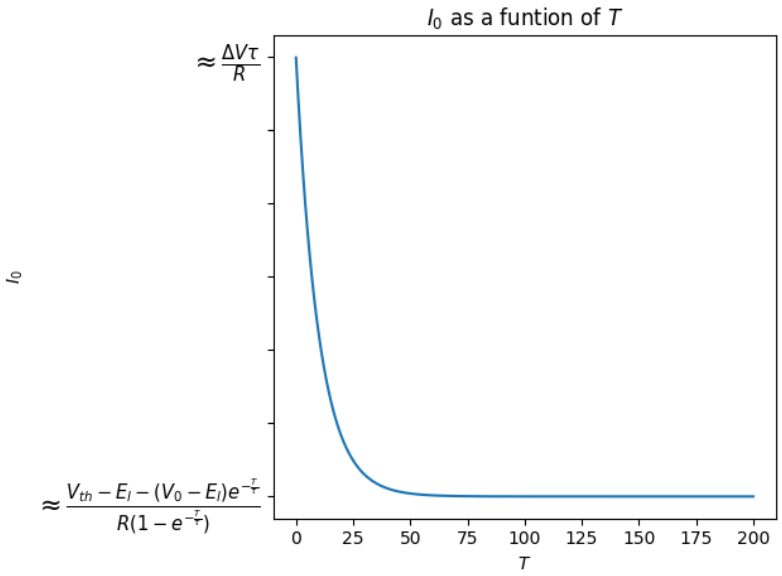

Figure 2: Graph of I_0 vs T

2 Reinforcement Learning

(a) **Model-Based**

Assuming there are no barriers in the grid, we can code the car in the following way such that it finds the shortest possible path between the start S and the finish F, with a model-based reinforcement learning approach. Specifically, we can use a Value Iteration algorithm to find the optimal policy for the car to use.

First, we will defind the state-space S as:

$$S = \{(x, y) | 1 \leq x < n, 1 \leq y < n\}$$

Which denotes the possible states the car can be in (starting from 1 because of the definition of the grid).

Then, we define the action-space A as:

$$A = \{(-1, 0), (1, 0), (0, -1), (-1. - 1), (1, 1), (-1, 1), (1, -1)\}$$

Which denotes the possible actions the car can take in any given state.

Since it is a Model-Based system, we can provide it with botha Transition Model and a Reward Model.

We will denote the Transition Model as a probabiltity function as $P(s'|s, a)$, which gives the probability of transitioning to state s' given that the car is in state s and takes action a. This is defined by the grid, and is 1 if the car can move to the state, and 0 otherwise.

And the Reward Model as a function:

$$R(s,a,s') = \begin{cases} 100, & \text{if } s = F \\ -1, & \text{otherwise} \end{cases} \tag{7}$$

We will also initialize a Value function $V(s)$ for each state, which will be updated as the car learns the optimal policy. Its origial value is 0 $\forall s \in S$

To get the optimal policy, we will use the Value Iteration algorithm, using the Bellman Equation update at each step.

That is, we initialize the value funciton:

$$V(s) = 0 \forall s \in S \tag{8}$$

Then, we iterate over the value function, updating it at each step:

$$V_{new}(s) \leftarrow \max_{a \in A} \left(\sum_{s \in S} P(s'|s,a)[R(s,a,s') + \gamma V_{old}(s)] \right) \tag{9}$$

where gamma is a discount factor, which we can set anywhere between 0 and 1. This informs teh model how much to value future rewards.

After each update, we will check to see if the maximum change in the value funciton has converged to some small value, θ, like 10^{-6}. That is, we check whether:

$$\max_{s \in S} |V_{new}(s) - V_{old}(s)| < \theta \tag{10}$$

Once this becomes the case, we stop iterating, and extract the policy. Namely:

$$\pi(s) = \text{argmax}_{a \in A} \left(\sum_{s \in S} P(s'|s,a)[R(s,a,s) + \gamma V(s)] \right) \tag{11}$$

Once the policy is extracted, the car can follow the policy to ensure it reaches the finish line in the shortest possible time.

Model-Free

In the case of a Model-Free approach, we can use the Q-Learning technique to update the policy for the car.

The State Space and Action space are similarily defined as above. However, now we introduce the Q-Value, i.e., the expected cumulative reward for taking action a in state s as $Q(s,a)$

Moreover, for our policy, we can use the ϵ-greedy strategy, namely, for some small ϵ, the action the car will choose is:

$$a = \begin{cases} \text{argmax}_a Q(s,a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \tag{12}$$

Thus, for every state s and action a, we update the Q-value as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{13}$$

Where α is the learning rate, $R(s,a)$ is the reward recived by the car at that state, and γ is, again, the future discount factor.

The car will cycle through the possible states until it reaches the goal-state, F. Afterwards, it will restart the process, initiating a new *episode*, until the update to the Q-values converges to some small value, θ, like 10^{-6}.

(b) For the **Model-Based** approach, we can speed up the convergence to an optimal policy by using a 'Prioritized Sweeping' strategy. Prioritized Sweeping allows the agent to focus on updating only the states that are likely to have large updates, and skip those that are likely to have small updates, minimizing the number of iterations needed to converge to the optimal policy. Priority Sweeping works by using a priority que for the update of states. In order to be added to the que, a state's expected change to the value function, $\Delta(s)$, must exceed some threshold, Θ .

To do prioritize sweeping, we first initialize $V(s)$:

$$V(s) = 0 \forall s \in S \tag{14}$$

Then, we compute the expected change of each state, s as:

$$\Delta(s) = \left| V(s) - \max_{a \in A} \sum_{s} P(s'|s,a)[R(s,a,s') + \gamma V(s)] \right| \tag{15}$$

and add the state to the priority que if $\Delta(s) > \Theta$

Then, we pop the state s with the highest priority from the que, and update the value funciton with that state. Then, we update the expected change for all states that are affected by the update, and add them to the que if their expected change is greater than Θ.

Following this algorithm will allow the car to converge to the optimal policy faster than the standard Value Iteration algorithm.

For the **Model-Free** approach, we can speed up convergence by varying the value of ϵ in our policy. Specifically, we can start with an explorartory phase, where ϵ is large, and the car is more likely to take random actions, allowing the car to explore the state-space more thoroughly. Then, we decrease ϵ over time as the car learns the optimal policy, shifting to a more exploitative strategy. This is called the ϵ-Greedy with Decay strategy.

Simply put, we set ϵ as a function of the episode number t. Formally, on each episode we set:

$$\epsilon_t = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min})e^{-\lambda t} \tag{16}$$

where λ is the decay rate, and ϵ_{min} and ϵ_{max} are the minimum and maximum values of ϵ respectively. The value of the decay rate, λ, will determine how quickly the car shifts from exploration to exploitation. This is a hyperameter that can be tuned to the task. The same goes for ϵ_{min} and ϵ_{max}. But, in general, this strategy will work to speed up the convergence of the Q-Learning algorithm to the optimal policy.