Reinforcement Learning (70028)

Markov Processes (Let's Go Markov)

In reinforcement learning, we need a real, tangible method for managing complexity. It's important to distinguish between **Markov Processes** and **Markov Decision Processes**.

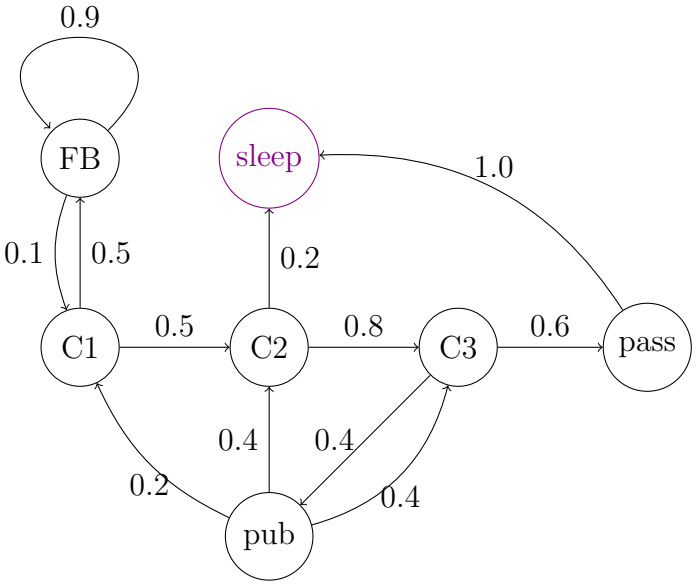
A Markov process is a tuple $(\mathcal{S}, \mathcal{P})$, where $\mathcal{S}$ is a set of states, and $\mathcal{P}_{ss'}$ is a matrix giving us the probability of transitioning from one state to another; note that the probabilities are based only on the current state (one at time t), and not looking beyond that - short memory is important;

$$\mathcal{P}_{ss'} = P[S_{t+1} = s' \mid S_t = s]$$

A Markov process generates a chain of states governed by probabilistic transitions.

A state s_t is Markov iff $P[s_{t+1} \mid s_t] = P[s_{t+1} \mid s_1, \dots, s_t]$; the conditional probability of transitioning to a particular state depends only on the particular state (previous states don't really matter). The equation states that the probabilities are equal, whether it be the current state, or all states preceding - the future is independent of the past given the present. Another way this can be thought of is that the present state, s_t, captures all information in the history of the agent's events, any data of the history is no longer needed once the state is known, or the current state is a sufficient statistic of the future.

An example is as follows, note that the black states are transient states (where it can lead to another state) and the violet states are terminal states. The following is Markovian as the probabilities don't change based on the history (how we got to a state).



The entries must be probabilities (hence between 0 and 1). The matrix defines transition probabilities from all states s to all successor states s'. Since all probabilities have to be accounted for (all rows of the matrix sum to 1) - after leaving s, we need to end up somewhere, which could also mean returning to s;

$$\sum_{s'} \mathcal{P}_{ss'} = 1$$

In the example above, the probability of going to sleep after C2 (class 2) in the morning could be different depending on the time of day (i.e. constantly changing). If $P[s_{t+1} \mid s_t]$ doesn't depend on t, but rather just the origin and destination states, then the Markov chain is stationary or homogenous.

Markov Reward Process

A Markov Reward Process is a Markov chain which emits rewards (the reward hypothesis states that all of what we think of as goals and purposes can be thought of as the maximisation of the expected value of the cumulative sum of a scalar signal known as reward); hence a tuple $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$. This has the following components;

- $\mathcal{S}$ — a set of states
- $\mathcal{P}_{ss'}$ — a state transition probability matrix
- $\mathcal{R}_s = \mathbb{E}[r_{t+1} \mid S_t = s]$

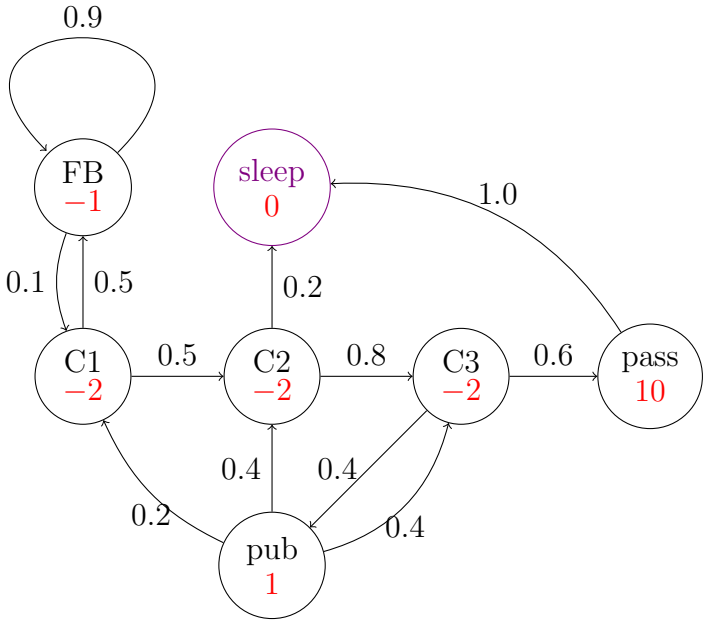
  an expected immediate reward, collected upon departing state s (collection occurs at time $t+1$, we are at state s at time t)
- $\gamma \in [0, 1]$ — discount factor

We can define the return R_t as the total discounted reward from time-step t (note that we use $t+1$ as the first element, since it's collected at $t+1$);

$$R_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

The factor γ is how we discount the present value of future rewards; the value of receiving a reward r after $k+1$ time steps is $\gamma^k r$, valuing immediate reward higher than a delayed reward - hence γ closer to 0 leads to short-sighted evaluation, whereas γ closer to 1 leads to far-sighted evaluation (taking future rewards more strongly).

We can add a reward to the previous example as follows (in red);



For example, consider a certain run, where the starting state $S_1 = \text{C1}$ and $\gamma = \frac{1}{2}$, T is the time to reach the terminal state;

$$R_1 = r_2 + \gamma r_3 + \cdots + \gamma^{T-2} r_T$$

Consider the run where the student attends all classes in order and passes; hence C1, C2, C3, pass, sleep;

$$R_1 = -2 + \frac{1}{2} \cdot -2 + \frac{1}{2}^2 \cdot -2 + \frac{1}{2}^3 \cdot 10$$

Most MRPs are discounted with $\gamma < 1$, as it's mathematically convenient by avoiding infinite returns in cyclic / infinite processes (by causing convergence). It also aids in expressing uncertainty in future

rewards. A more tangible example is a financial reward, where immediate rewards can be put into a bank and earn interest, similarly, animal decision making shows preference for immediate rewards rather than future rewards.

We can define the state value function $v(s)$ of a MRP as the expected return R starting from state s at time t, thinking of the state as a function parameter;

$$v(s) = \mathbb{E}[R_t \mid S_t = s]$$

The lecture then goes over an example using golf, which is actually quite intuitive.

The Bellman Equation for MRPs is as follows. We can express it in a recurrence relation, as the immediate reward and the discounted return of the successor state.

$$\begin{aligned} v(s) &= \mathbb{E}[R_t \mid S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \mid S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots) \mid S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma R_{t+1} \mid S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma v(S_{t+1}) \mid S_t = s] \end{aligned}$$

The equation can also be written as the sum notation (the previous one was the expectation notation, this has the expectation written out) - there are a total of n of these equations, as there's one for each state;

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} v(s')$$

As such, this can be written in vector notation as follows, with $\boldsymbol{v}$ being n-dimensional;

$$\boldsymbol{v} = \mathcal{R} + \gamma \mathcal{P} \boldsymbol{v}$$

This can be directly solved as follows, as it's linear and self-consistent;

$$\begin{aligned} \boldsymbol{v} &= \mathcal{R} + \gamma \mathcal{P} \boldsymbol{v} \\ (\mathbb{1} - \gamma \mathcal{P}) \boldsymbol{v} &= \mathcal{R} \\ \boldsymbol{v} &= (\mathbb{1} - \gamma \mathcal{P})^{-1} \mathcal{R} \end{aligned}$$

Since matrix inversion is computationally expensive, being in the order of n^3 for n states, a direct solution is only feasibly for small MRPs. Iterative methods for solving large MRPs include (and all three will be covered);

- dynamic programming
- Monte-Carlo evaluation
- Temporal-Difference learning

Policies

A policy π is a function of the state, formalising the actions to take at a given state. A rigid, deterministic policy can be disadvantageous (e.g. rock, paper, scissors) - exposing the agent to being systematically exploited. A policy can be formally described as the conditional probability distribution to execute an action $a \in \mathcal{A}$ given that one is in state $s \in \mathcal{S}$ at time t;

$$\pi_t(a, s) = P[A_t = a \mid S_t = s]$$

The general form of the policy is probability, or stochastic, hence π is a probability. However, if the policy is deterministic (only a single a is possible for state s), then $\pi(a, s) = 1$, $\pi(a', s) = 0$, $\forall a \neq a'$.

Consider the following example, where there are two actions, a_1, a_2 where we either play the lottery (costing 1), or save (not costing anything). The two states, s_1 and s_2 correspond to winning or losing the lottery.

$$a^* = \underset{a_i}{\operatorname{argmax}} \sum_{j=1}^{2} \mathcal{R}_{s_j}^{a_j} P[s_j \mid a_i]$$