RL01. Markov Decision Processes

Introduction:

- In Supervised Learning, you're given (x, y) pairs, and the goal is to find a function $f(x)$ that maps a new x to a proper y (function approximation).
- In Unsupervised Learning, you're given a dataset of x points, and the goal is to find a function $f(x)$ the provides a description of this dataset (Clustering).
- In Reinforcement Learning, we're doing a similar task as in Supervised Learning. But instead of having (x, y) pairs as an input, we'll be given (x, z) pairs, and our job is to find $f(x)$ that generates y.

Markov Decision Processes:

- A Markov Decision Process captures the world in terms of:
  - States S: This is literally the state of the object of interest represented in any way.
  - Model: This is the Transition Model (function), which is the probability of transitioning to a new state s' given that the object starts at state s and performs action a.

    $$T(s, a, s') \sim P_r(s' | s, a)$$

    1. This model follows the Markovian Property, which states that only the current state s affects the result of the model (only present matters).
    2. The model assumes that the rules don't change over time.
  - Actions: $A(s)$ are the things the object is allowed to do in a particular state s.
  - Reward: $R(s)$ (or $R(s, a)$ or $R(s, a, s')$) is the reward the object gets when transitioning to a state s, which tells it the "usefulness" of transitioning to that state.
  - Policy: The previous four points define a Markov Decision Problem; the Policy defines the solution. The policy is a function $\pi(s)$ that takes a state s as an input and returns the proper action a. A problem might have different policies.
    The optimum policy π^* is the policy that maximizes the long term expected reward.
- We will be given (s, a, r) and our task would be to learn the optimum policy π^* that produces the optimum action (highest reward).

More About Rewards:

- In the Reinforcement Learning context, we don't get a reward for each action. Instead, we get a "delayed reward" in terms of a positive or negative label for the state we end up in.
- Because we don't have instant rewards for each action, we have the problem of figuring out which specific action(s) lead us to a positive/negative outcome. This problem is called Temporal Credit Assignment.

- A small negative reward will encourage the agent to take the shortest path to the positive outcome. However, a big negative reward will encourage the agent to take the shortest path no matter what the result is, so it might end up with a negative outcome.
- This means that determining the rewards is some sort of a domain knowledge.

Sequences of Rewards:

- Another point to take into consideration is how much time you have to reach the result. If you have an infinite amount of time (Infinite Horizon) you will be able to take longer paths to avoid possible risks. On the other hand, if you don't have that much time, you'll have to take the shortest path even if it underlies some risk of falling into a negative outcome. This means that the time you have might change the optimum policy.

  $$\pi(s, t) \rightarrow a$$

  That means that without assuming an Infinite Horizon, we will lose the notion of stationarity in our policies.
- Utility of Sequences:
  - The notion of stationarity of preferences states that if I prefer a sequence of states today over another sequence of states, then I'd prefer that sequence of states over the same other sequence of states tomorrow.
  - This notion forces you to do some sort of reward addition, because nothing else will guarantee the stationary of preferences.

    $$U(s_0\ s_1\ s_2\ \dots) = \sum_{t=0}^{\infty} R(s_t)$$

  - A problem with this equation is that it doesn't really differentiate between different sequences of rewards, since we add up to infinity either way.
  - To solve this issue, we add another term to the equation:

    $$U(s_0\ s_1\ s_2\ \dots) = \sum_{t=0}^{\infty} \Upsilon^t R(s_t) \qquad 0 \leq \Upsilon < 1$$

    By adding this term, we start off with substantial rewards, but then they trail off quickly.
    We can write down the same equation after bounding it off with the maximum possible reward:

    $$U(s_0\ s_1\ s_2\ \dots) \leq \sum_{t=0}^{\infty} \Upsilon^t R_{max}$$

    $$x = (\Upsilon^0 + \Upsilon^1 + \Upsilon^2 + \cdots)$$
    $$x = \Upsilon^0 + \Upsilon x$$
    $$x - \Upsilon x = \Upsilon^0$$
    $$x(1 - \Upsilon) = 1$$
    $$x = \frac{1}{1 - \Upsilon}$$
    $$U(s_0\ s_1\ s_2\ \dots) = \frac{R_{max}}{1 - \Upsilon}$$

1. If $\gamma \to 0$, we get the first reward and then everything else will fall off to nothing.
2. If $\gamma \to 1$, we get a maximized reward.

This is called Discounted Rewards, and it allows us to go infinite distance in finite time.

More About Policies:

- The optimum policy is the one that maximizes the long term expected reward:

$$\pi^* = argmax_\pi E[\sum_{t=0}^{\infty} \Upsilon^t R(s_t) \Big| \pi]$$

- We'll then define the utility of a state in such a way that will help us solve for the optimum policy. The utility of a particular state will be the expected set of states that we're going to see if we started from the given state and followed to given policy:

$$U^\pi(s) = E[\sum_{t=0}^{\infty} \Upsilon^t R(s_t) \Big| \pi, s_0 = s]$$

  Note that the utility of a state represents the long-term feedback (delayed reward) we get is we started from this state, which is different from the reward of the state. In other words, the utility is the reward of the state plus all the rewards that we get if we started from that state.
- Taking the utility of the state into consideration, we can rewrite the function of the optimum policy as follows:

$$\pi^* = argmax_a \sum_{t=0}^{\infty} T(s,a,s')U^{\pi^*}(s')$$

  This means that the optimum policy is the one that for every state returns the action that maximizes the expected utility, which in turn maximizes the overall reward.
- Now, the true utility of a state is defined by Bellman Equation:

$$U(s) = R(s) + \Upsilon max_a \sum_{t=0}^{\infty} T(s,a,s')U(s')$$

Solving Bellman Equation:

- If we have n utilities, then we have n equations in n unknows. If the equations are linear, they would have been solvable, but the max operator makes the equations non-linear.
- Value Iteration:
  - So, to solve this equation we'll follow this algorithm:
    1. Start with arbitrary utilities.
    2. Update utilities based on neighbors.
    3. Repeat until convergence.
  - We'll use this function to update the utilities:

$$\widehat{U}(s)_{t+1} = R(s) + \Upsilon max_a \sum_{s'} T(s,a,s')\widehat{U}(s')_t$$

We basically update the estimate of utility of state s by calculating the actual reward for this state plus the discounted utility expected from the original estimate of utility of s.

- This is guaranteed to converge because with each step we're adding $R(s)$, which is a true value. So, even if we started with a very wrong estimate of utility, we'll keep adding the true value $R(s)$ over and over that it will dominate the original wrong estimate.

- Policy Iteration:
  - Since we care about policies, and not the actual values, we can follow a simpler faster algorithm:
    1. Start with an arbitrary policy π_0
    2. Evaluate how good that policy is by calculating the utility of using the policy:
    $$U_t = U^{\pi_t}$$
    3. Improve:
    $$\pi_{t+1} = argmax_a \sum_{s'} T(s, a, s')U(s')_t$$
  - To calculate the utility at the evaluation step, we use Bellman Equation:
    $$U(s) = R(s) + \Upsilon \sum_{s'} T(s, \pi_t(s), s')U(s')_t$$
    Note that rather than having the max over actions as in the normal Bellman equation, we already know what action to take according to the policy we're evaluating. This trick removes the max operator, making this a set of n solvable linear equations in n unknowns.