Markov decision process

Program	Agent	environment	action	state

,

Markov decision process (MDP):

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, ...$$

Reward is a signal from the environment

Policy function $\pi(a \mid s) =$ probability

return $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$

discount factor: $0 \leq \gamma \leq 1$

Monte Carlo method

world models: $p(s', r \mid s, a)$

An episode describes a single run through of the process, with state and return reset after each epsidoe.

Value functions: keep track of average return (G_t) expected when following a certain policy (π) in a certain state (s) or state + actions (s, a)

State-value: $V_\pi(s)$, expected return when in state s while follow policy π, or the average expected return for $Q_\pi(s, a)$ accross all possible a.

Action-value: $Q_\pi(s, a)$, expected return when in state s, pick a certain action a, and then follow policy (π).

Optimal functions: $V_*(s)$, $Q_*(s, a)$.

Three methods of RL:

1. Policy gradient method
   - Issues with variance (different averages)
   - More dependent on policy function
2. Policy gradient method, with value function
   - Accounts for variance, which is good
   - Balance of using policy & value function
3. Just use action-value (Q) function
   - Policy becomes dependent on action-value function
   - More dependent on value function

Developing Q following method 3

Epsilon-greedy balance:

Epsilon (ε) describes how much the policy picks a random value as opposed to the optimal value.

Epsilon usually starts off high (because the agent is not much better than random), and becomes lower and lower ask training progresses. This ensures sufficient exploration while retaining effective exploitation.

Temporal Difference

Credit assignment problem: figuring out the impact of individual actions within a sequence of many actions.

Temporal difference breaks down evaluation to an action by action basis, rather than episode by episode.

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$$

$$G_t = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...)$$

$$G_t = r_t + G_{t+1}$$

Learning **action-value (Q)** function

SARSA:

$$Q(s_t, a_t) \rightarrow r_t + \gamma Q(s_{t+1}, a_{t+1})$$

Expected SARSA:

$$Q(s_t, a_t) \rightarrow r_t + \gamma \sum_{i=1}^{n} \pi(a, s_{t+1}) Q(s_{t+1}, a)$$

Q-Learning:

$$Q(s_t, a_t) \rightarrow r_t + \gamma \ \max(\mathrm{a}) Q(s_{t+1}, a)$$

Learning **state-value (V)** function:

$$V(s_t) \rightarrow r_t + \gamma V(s_{t+1})$$

Actual math for learning at episode e:

$$Q_{e+1}(s_t, a_t) = Q_e(s_t, a_t) + \alpha[r_t + \gamma Q_e(s_{t+1}, a_{t+1}) - Q_e(s_t, a_t)]$$

Bellman's Principle of Optimality: The overall best policy must involve choosing the best action at every point.

Deep Q Networks

Deep Q neural networks allows for continuous states but not continous actions. Off-policy Q-learning is used to approximate the action-value function. Epislon greedy balances exploitation and exploration.

Q-learning logic for tic tac toe

begin
make actual move at s0 using Qa0, creating s1

while not end:
opponent makes theoretical best move/random move at s1 using Qa1, creating s2
if board ends with draw, update Q with
[draw] at s0
else:
[loss] at s0
[win] at s1

inference to get Qa2 at s2
update Q at s0 using Qa2

s1 becomes s0, s2 becomes s1
Qa1 becomes Qa0, Qa2 becomes Qa1

components:
1. Board
2. Reward giving environment
3. Agent which uses Qa values to make moves and update Q
4. Model(s) which gives Qa values

Policy gradients

Unlike a value function, the policy has no ground truth target. Instead, we seek to maximised the objective function $J(\pi_\theta)$.

Examples of $J(\pi_\theta)$:

$E_{J\sim\pi_\theta}[R(J)]$	Expected total reward in every trajectory generated by the policy
$V_{\pi(s_0)}$	State value of the initial state in an episode with a start and end
$\sum_s d_\pi(s)V_\pi(s)$	Average state value across all states
$\sum_s d_\pi(s)\sum_a \pi_\theta(a\|s)\Gamma_{a,s}$	Average reward per time step

According to the **policy gradient theorem**:

$$\nabla_\theta J(\pi_\theta) = E\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)\Psi_t\right]$$

where:
- $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ represents how to increases the probability of action a. In softmax, this can be achieved by adjusting the probablity of a single action.
- Ψ_t describes how good an action a is, often represented by $Q_\pi(s_t, a_t) - V_\pi(s_t)$ in an actor-critic model
- E averages across all states and actions

Surrogate objective functions

TRPO:

$$E\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\text{ old}}(a_t|s_t)}\hat{A}_t\right] \text{ subject to } E[\mathrm{Kl}[\pi_{\theta\text{ old}}(-1s_t), \pi_\theta(-1s_t)]] < \delta$$

PPO:

$$\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\text{ old}}(a_t|s_t)}\hat{A}_t, \mathrm{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\text{ old}}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t\right)$$