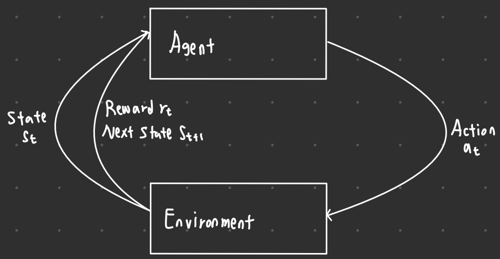
Pictorial depiction of Reinforcement learning.



Markov Decision Process (MDP)

- Mathematical formulation of the RL problem.
- Markov property: Current state completely characterizes the state of the world

Defined by: (S, A, R, P, γ)

S: Set of all possible states

A: Set of all possible actions

R: Distribution of reward given (state, action) pair

P: Transition probability over the next state given (state, action) pair

γ: discount factor

- At time step $t=0$, environment samples initial state $s_0 \sim p(s_0)$
- Then, for $t=0$ until done:
  1) Agent selects action a_t
  2) Environment samples reward $r_t \sim R(\cdot \mid s_t, a_t)$
  3) Environment samples next states $s_{t+1} \sim P(\cdot \mid s_t, a_t)$
  4) Agent receives reward r_t and next state s_{t+1}

- A policy π is a function from S to A that specifies what action to take in each state
- Objective: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

The optimal policy π^*

: We need optimal policy that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability)?

: Maximize the expected sum of rewards. Formally:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid \pi\right] \text{ with } s_0 \sim p(s_0),\ a_t \sim \pi(\cdot \mid s_t),\ s_{t+1} \sim p(\cdot \mid s_t, a_t)$$

How good is a state?

Value function: The value function at state s, is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

How good is a state-action pair?

: The Q-value function at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Bellman equation : The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Q^* satisfies the following Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \varepsilon}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$

Intuition: If the optimal state-action values for the next time-step $Q^*(s',a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$

The optimal policy π^* corresponds to taking the best action in any state as specified by Q^*

☆ Solving for the optimal policy

1. Value iteration algorithm
2. Q-learning, deep q-learning

Value iteration algorithm

: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') \mid s,a\right]$$

Q_i will converge to Q^* as $i \to$ infinity. However, it's not scalable. Must compute $Q(s,a)$ for every state-action pair.

Solution: use a function approximator to estimate $Q(s,a)$. E.g. a neural network.

Q-learning

- We want to find a Q function that satisfies the Bellman Equation

Forward Pass

: Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim p(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$

where $y_i = \mathbb{E}_{s' \sim \varepsilon}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \mid s,a\right]$

Backward Pass : Gradient update

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim p(\cdot); s' \sim \varepsilon}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)) \nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Q-network Architecture

FC-4 (Q-values)
FC-256
32 4x4 conv, stride 2
16 8x8 conv, stride 4
Current state s_t

Last FC layer has 4-d output (if 4-actions), corresponding to $Q(s_t,a_1)$, $Q(s_t,a_2)$, $Q(s_t,a_3)$, $Q(s_t,a_4)$

$Q(s,a;\theta)$: Neural network with weights θ

☆ Training the Q-network : Experience Replay

Learning from batches of consecutive samples is problematic :

① Samples are correlated → inefficient learning

② Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) → can lead to bad feedback loops

☆ Address these problems using experience replay.

- Continually update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples.

Pseudocode for Deep Q-Learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
] Initialize replay memory, Q-network

for episode $=1, M$ do : Play M episodes (full games)

  Initialize sequence $s_1=\{x_1\}$ and preprocessed sequence $\phi_1=\phi(s_1)$: Initialize state (starting game screen pixels) at the beginning of each episode

  for $t=1,T$ do : For each timestep t of the game

    With probability ε select a random action a_t : With small probability, select a random action, otherwise select greedy action from current policy

    Otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1} Take the action and observe the reward and next state

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D : Store transition in replay memory

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D : Experience Replay

    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$

  end

end