Lecture Notes on Reinforcement Learning

1. Introduction to Reinforcement Learning (RL)

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize some notion of cumulative reward. Unlike supervised learning, where the model is trained on a given dataset, RL involves learning through interaction with the environment.

2. Key Concepts in Reinforcement Learning

2.1 Agent and Environment

- **Agent**: The learner or decision-maker.
- **Environment**: Everything the agent interacts with.

2.2 State, Action, and Reward

- **State (s)**: A representation of the current situation.
- **Action (a)**: A set of all possible moves the agent can make.
- **Reward (r)**: Immediate return received after performing an action.

2.3 Policy, Value Function, and Model

- **Policy (π)**: A strategy used by the agent to decide actions based on the current state. Can be deterministic or stochastic.

$$\pi(a|s) = P(A_t = a|S_t = s)$$

- **Value Function**: Estimates the expected return (cumulative future reward) from a state or state-action pair.
  - **State-Value Function ($V(s)$)**: Expected return starting from state s.

$$V^{\pi}(s) = \mathrm{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s\right]$$

  - **Action-Value Function ($Q(s, a)$)**: Expected return starting from state s and taking action a.

$$Q^{\pi}(s, a) = \mathrm{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a\right]$$

- **Model**: Describes the environment in terms of state transitions and rewards. Used in model-based RL.

3. Markov Decision Process (MDP)

MDPs provide a mathematical framework for modeling decision-making in RL. An MDP is defined by:

- **States** (S): Set of all possible states.
- **Actions** (A): Set of all possible actions.
- **Transition Probability** (P): Probability of moving from one state to another given an action.

$$P(s', r|s, a) = P(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

- **Reward Function** (R): Expected reward received after transitioning from state s to s' due to action a.
- **Discount Factor** (γ): Determines the importance of future rewards.

$$0 \leq \gamma \leq 1$$

4. Bellman Equations

The Bellman equations provide recursive relationships for value functions.

4.1 Bellman Expectation Equation for $V(s)$

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^{\pi}(s')]$$

4.2 Bellman Expectation Equation for $Q(s, a)$

$$Q^{\pi}(s, a) = \sum_{s', r} P(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') Q^{\pi}(s', a')]$$

5. RL Algorithms

5.1 Model-Free Methods

5.1.1 Monte Carlo Methods

Monte Carlo methods learn from complete episodes of experience. They require the episode to end to update the value function. This method estimates the value of a state as the average return following visits to that state.

5.1.2 Temporal Difference (TD) Learning

TD Learning methods learn directly from raw experience without waiting for the episode to end. They update the value function based on a combination of the actual reward and the estimated value of the next state.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

5.2 Q-Learning

Q-Learning is a popular off-policy TD control algorithm that aims to find the optimal policy. It updates the action-value function based on the maximum estimated future rewards.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

5.3 SARSA (State-Action-Reward-State-Action)

SARSA is an on-policy TD control algorithm that updates the action-value function based on the policy being followed.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

6. Policy Gradient Methods

Policy Gradient methods directly parameterize the policy and optimize the parameters using gradient ascent. Instead of learning the value function, these methods learn the policy function directly.

$$\nabla J(\theta) = \mathrm{E}_\pi [\nabla \log \pi_\theta(a|s) Q^\pi(s, a)]$$

7. Actor-Critic Methods

Actor-Critic methods combine the benefits of value-based and policy-based methods. The actor updates the policy (π) while the critic updates the value function (V or Q). The actor chooses actions according to the policy, and the critic provides feedback on the value of those actions.

8. Advanced Topics in RL

8.1 Deep Reinforcement Learning

Combines deep learning with RL, where neural networks are used to approximate value functions and policies. Examples include Deep Q-Networks (DQN) and Deep Deterministic Policy Gradient (DDPG).

8.2 Exploration vs. Exploitation

Balancing exploration (trying new actions to discover their effects) and exploitation (choosing actions that are known to yield high rewards) is crucial in RL. Techniques such as ϵ-greedy, softmax, and Upper Confidence Bound (UCB) are used.

8.3 Multi-Agent Reinforcement Learning (MARL)

Extends RL to environments with multiple agents that can interact and cooperate or compete with each other. Examples include cooperative game theory and competitive game theory applications.

8.4 Inverse Reinforcement Learning (IRL)

Aims to infer the reward function given observed behavior from an expert. Useful in scenarios where it is easier to demonstrate the desired behavior than to specify the reward function.

9. Applications of Reinforcement Learning

- **Robotics**: Autonomous control and navigation.
- **Game Playing**: AlphaGo, OpenAI Five.
- **Finance**: Algorithmic trading.
- **Healthcare**: Personalized treatment plans.
- **Recommendation Systems**: Dynamic content recommendation.

10. Case Studies and Examples

10.1 AlphaGo

AlphaGo, developed by DeepMind, used a combination of supervised learning and RL to master the game of Go. It employed a policy network to select moves and a value network to predict the winner of the game.

10.2 OpenAI Five

OpenAI Five is an RL-based AI that plays the game Dota 2. It uses a combination of LSTM networks and proximal policy optimization (PPO) to learn complex strategies in the game.