21. Reinforcement Learning

This chapter explores how an agent can learn optimal behaviors through trial and error using feedback from rewards and punishments.

21.1 Introduction

Introduces reinforcement learning as learning without labeled examples, where agents learn by receiving rewards and punishments from the environment.

21.2 Passive Reinforcement Learning

In passive learning, the agent follows a fixed policy and learns the utility of states based on rewards received while executing that policy.

21.2.1 Direct Utility Estimation

The agent estimates state utilities by averaging the total rewards observed during trials without modeling transitions between states.

21.2.2 Adaptive Dynamic Programming

This approach learns a model of state transitions and uses dynamic programming to solve the Bellman equations for utility values.

21.2.3 Temporal-Difference Learning

Uses observed state transitions to update utility values based on the difference between current estimates and the observed rewards.

21.3 Active Reinforcement Learning

Active learning involves not only evaluating states but also learning which actions to take by balancing exploration and exploitation.

21.3.1 Exploration

The agent must explore the environment to improve its model and avoid getting stuck in suboptimal policies by only exploiting known strategies.

21.3.2 Learning an Action-Utility Function

Q-learning is introduced, where the agent learns action-specific utilities (Q-values) without needing to model state transitions explicitly.

21.4 Generalization in Reinforcement Learning

Generalization involves using function approximation to represent utility or Q-values when dealing with large state spaces, enabling learning from limited experiences.

21.5 Policy Search

This method involves directly searching for the best policy by optimizing parameters in a parameterized policy, often using gradient-based methods.

21.6 Applications of Reinforcement Learning

This section highlights applications such as game playing, robotics, and autonomous control tasks where reinforcement learning is successfully applied.

21. Reinforcement Learning

Reinforcement learning (RL) is a method where agents learn to make decisions by interacting with an environment. Instead of being provided with explicit instructions (as in supervised learning), agents receive feedback in the form of rewards or punishments and must figure out which actions lead to better outcomes. This method is inspired by how humans and animals learn through trial and error.

21.1 Introduction

In supervised learning, an agent is typically given labeled examples to learn from. In contrast, reinforcement learning deals with learning through interaction without explicit labeled examples. The agent doesn't know which action is best but instead learns from the consequences of its actions, receiving positive or negative reinforcement (rewards or penalties).

Example: Imagine teaching a robot to play chess. Instead of telling the robot which move is optimal in every situation, you just let it play and tell it whether it wins or loses at the end. The robot must learn, through trial and error, which moves help it win more often.

21.2 Passive Reinforcement Learning

In passive reinforcement learning, the agent's policy (i.e., a fixed strategy that tells the agent which action to take in each state) is fixed, and the task is to evaluate the quality of the policy by learning the utility values (long-term expected rewards) of the states it visits.

21.2.1 Direct Utility Estimation

In direct utility estimation, the agent estimates the utility of a state by averaging the total rewards it receives over multiple trials when starting from that state. This is a simple, trial-based approach that doesn’t consider how the states are related to each other.

Example: Suppose an agent moves through a gridworld, and each cell (state) has a reward. The agent starts in different cells and records the total reward received until it reaches the end. The utility of a cell is updated based on the average total reward received from starting at that cell.

21.2.2 Adaptive Dynamic Programming

Adaptive dynamic programming (ADP) improves upon direct utility estimation by learning the model of the environment’s state transitions (i.e., the probabilities of moving between states). Once the transition probabilities are learned, ADP uses these to solve the Bellman equations (a set of equations describing the utilities of states) and determine the optimal utility values for all states.

Example: In the same gridworld example, ADP would keep track of how often each action (like "move left" or "move up") leads to specific states. Once enough data is collected, the agent can solve the Bellman equations to compute the utility of each state more accurately.

21.2.3 Temporal-Difference Learning

Temporal-Difference (TD) learning is an incremental method that updates the utility values of states based on the difference between the current utility estimate and the reward plus the utility of the next state (this difference is known as the temporal difference). Unlike ADP, TD does not require learning the full transition model of the environment.

Example: In the gridworld, if the agent moves from state A to state B, TD learning updates the utility of state A based on the reward received in state B and the current estimate of state B’s utility, rather than waiting for the final outcome at the end of the trial.

21.3 Active Reinforcement Learning

Unlike passive learning, where the policy is fixed, active reinforcement learning involves the agent learning not just the utilities of states, but also which actions to take to maximize rewards. The agent must balance **exploration** (trying new actions to learn about the environment) and **exploitation** (choosing actions that maximize current knowledge of reward).

21.3.1 Exploration

The agent faces a challenge: it needs to explore actions it hasn't tried yet to learn about their outcomes, but it also wants to exploit the best-known actions to maximize rewards. A **greedy agent** always exploits (chooses the best action based on current knowledge), which can lead to suboptimal behavior because it doesn't explore enough.

Example: In the gridworld, the agent might learn that moving up leads to good rewards in certain areas. However, if it never explores moving right, it might miss an even better path.

Exploration Strategies:

1. **Epsilon-greedy strategy**: With probability ϵ, the agent picks a random action (exploration), and with probability $1-\epsilon$, it picks the best-known action (exploitation).
2. **Optimistic initialization**: The agent starts with overly optimistic utility estimates, encouraging it to explore more because all actions seem potentially rewarding initially.

21.3.2 Learning an Action-Utility Function (Q-Learning)

In **Q-learning**, instead of learning the utility of states, the agent learns the utility of **state-action pairs** (Q-values). This means the agent learns the expected reward for taking a particular action in a given state. Q-learning is a **model-free** method, meaning the agent does not need to know or learn the transition probabilities between states.

The **Q-value** of a state-action pair $Q(s, a)$ is updated using the formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

Where:

- $R(s)$ is the reward for the current state.
- γ is the discount factor (how much future rewards are valued).
- α is the learning rate.

Example: In the gridworld, the agent learns the value of taking specific actions (like moving left or right) in each state. By updating its Q-values after every action, it gradually learns the best action to take in every situation.

21.4 Generalization in Reinforcement Learning

In environments with large state spaces (like games or robotics), it's impractical to learn separate utility or Q-values for every possible state. Instead, **function approximation** is used to generalize from limited data, where the utility or Q-value is represented as a function of the state features, compressing the representation.

Example: In chess, instead of learning the exact utility of each possible board configuration (which is infeasible due to the vast number of possibilities), a function approximator (e.g., a

neural network) can learn to estimate the utility based on key features of the board, like material balance or control of the center.

Function Approximation

Function approximation uses models like linear functions or neural networks to approximate the utility or Q-value as a function of the state's features. This allows the agent to generalize from observed states to unseen ones.

21.5 Policy Search

In **policy search**, instead of learning the utility of states or Q-values, the agent directly searches for the best policy (a mapping from states to actions). This is often done by parameterizing the policy and adjusting the parameters to improve performance. Policy search is especially useful in continuous action spaces or complex environments where utility-based methods struggle.

Example: In robotic control (e.g., controlling a helicopter), the policy might be a set of parameters that control the helicopter's movement. Policy search algorithms adjust these parameters based on how well the helicopter performs specific maneuvers.

Stochastic Policies: Often, the policy is represented as a probability distribution over actions (called a **stochastic policy**), which allows for smoother optimization since small changes in parameters lead to small changes in the action probabilities.

21.6 Applications of Reinforcement Learning

Reinforcement learning has been applied in various fields, from game playing to robotics.

21.6.1 Applications to Game Playing

One of the most famous applications of RL is **TD-Gammon**, a backgammon program developed by Gerald Tesauro. TD-Gammon used **temporal-difference learning** to learn how to play the game by playing against itself, with no human intervention. Over time, it became one of the top backgammon players in the world, demonstrating the power of RL.

Example: TD-Gammon started with random play but gradually improved by updating its value function after every game. It learned to predict winning positions based on game outcomes and adjusted its strategy accordingly.

21.6.2 Application to Robot Control

In robotics, RL has been used for tasks like the **cart-pole balancing problem** (also known as the inverted pendulum). The task is to control a cart that moves left or right to balance a pole upright. Using RL techniques, the agent learns to apply the correct forces to keep the pole balanced without knowing the physics of the system beforehand.

Example: A robot learning to fly a helicopter can use **policy search** to optimize control strategies. By trying different control parameters in a simulator (or real world), the robot learns the best way to control the helicopter to perform complex maneuvers, like hovering or landing.