Introduction to Reinforcement Learning

Foundations for Deep Q-Networks

Vanni Leonardo

April 15, 2025

Bocconi
AI&Neuroscience
Student
Association

Outline

1.1 What is Reinforcement Learning? (Part 1)

Core Idea

Reinforcement Learning (RL) is about learning how to map situations to actions so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

Key Components & Interaction:

- **Agent**: The learner and decision-maker.
- **Environment**: Everything outside the agent; it responds to actions.
- **Interaction Loop**: Happens over discrete **time steps** $t = 0, 1, 2, \ldots$
  - Agent observes **state** $s_t \in \mathcal{S}$.
  - Agent takes **action** $a_t \in \mathcal{A}(s_t)$.
  - Environment transitions to new state s_{t+1}.
  - Environment provides **reward** $r_{t+1} \in \mathcal{R}(a_t, s_t) \subset \mathbb{R}$.

- **Policy** $\pi(a|s)$: Agent's strategy or behavior. Probability of taking action a given state s.

$$\pi : \mathcal{S} \to \Delta(\mathcal{A})$$
$$s \to P(\mathcal{A})$$

(Similar to Simple Acts in SEU framework of Decision Theory)

- **Trajectory/Episode**: A sequence resulting from interaction:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots$$

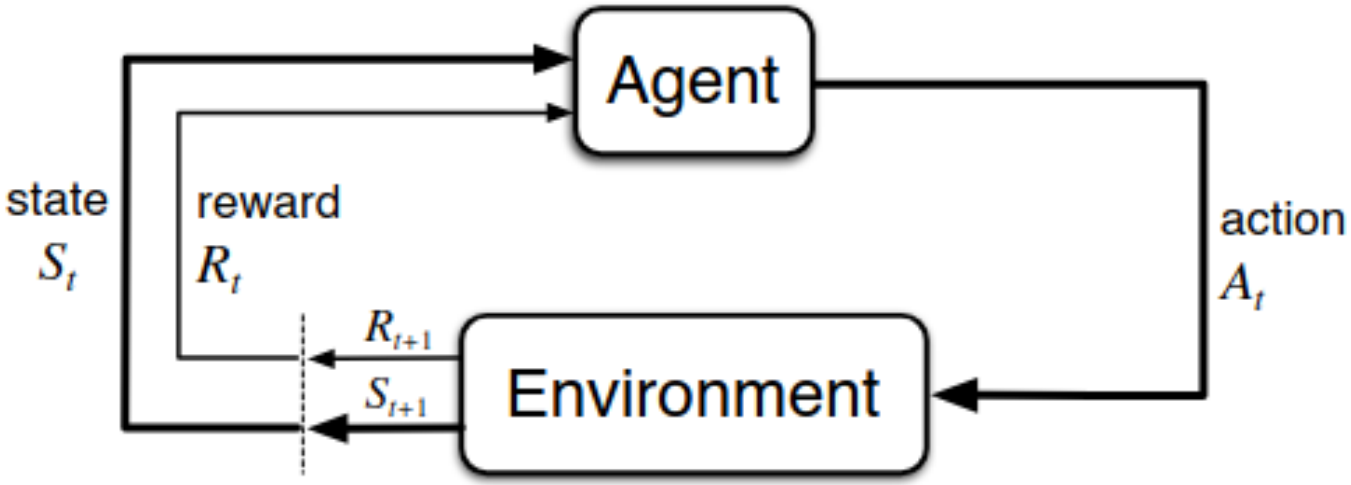
- **Return** G_t: The objective the agent aims to maximize. Total discounted future reward starting from time step t (discounted expected utility).

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$$

where $\gamma \in [0, 1]$ is the **discount factor**.

  - γ determines the present value of future rewards.
  - $\gamma \approx 0$: agent focuses only on immediate reward ("myopic").
  - $\gamma \approx 1$: agent values future rewards strongly ("farsighted").

Agent-Environment interaction Loop



Key Takeaways (Section 1.1 Recap)

- RL = Learning from interaction (trial-and-error + delayed reward).
- Goal: Maximize cumulative future reward (Return), not just immediate reward.
- Learning happens without explicit supervision (no "correct" action labels given).

1.2 Markov Decision Processes (MDPs)

Formalizing the Problem

We use MDPs $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ to formally describe the RL environment:

- $\mathcal{S}$: Set of states (State Space).
- $\mathcal{A}$: Set of actions (Action Space). May depend on state: $\mathcal{A}(s)$.
- P: Transition dynamics function.
  $P(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$.
- R: Reward function. Often simplified to expected reward
  $R(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$.
- γ: Discount factor.

The Markov Property

$$P(S_{t+1}, R_{t+1}|S_t, A_t) = P(S_{t+1}, R_{t+1}|S_t, A_t, R_t, S_{t-1}, \ldots, R_1, S_0, A_0)$$

The current state S_t captures all relevant information from the history.

Value functions estimate "how good" something is for the Agent in terms of future rewards (state-contingent utility functions).

State-Value Function $V^\pi(s)$:

- Expected return starting in state s and following policy π.

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

$$= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]$$

- "How good is it to be in state s under policy π?"

Action-Value Function $Q^\pi(s, a)$:

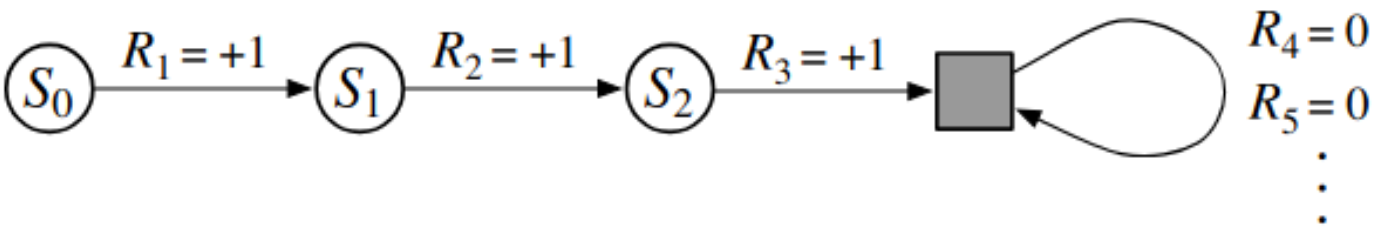
- Expected return starting in state s, taking action a, then following policy π.

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

$$= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = \right.$$

- "How good is it to take action a in state s under policy π?"

State Transition Diagram



Thus we can also write

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

where S_T is the terminal state.

Key Takeaways

- MDPs provide the mathematical language for RL problems.
- The agent's goal is implicitly defined: find a policy π that maximizes the expected return (value functions) $\neq$ immeadiate return.

Finding the Optimal Policy

The ultimate goal is to find an **optimal policy**, denoted π_*, which maximizes the expected return from every state.

$$\pi_* = \arg\max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}$$

(Note that value functions define a partial ordering over policies thus an optimal policy always exists)
An optimal policy achieves the optimal value functions:

- Optimal state-value function: $V_*(s) = \max_{\pi} V^{\pi}(s)$
- Optimal action-value function: $Q_*(s, a) = \max_{\pi} Q^{\pi}(s, a)$

If we know $Q_*(s, a)$, the optimal policy is simple: always choose the action with the highest Q-value:

$$\pi_*(s) = \arg\max_{a \in \mathcal{A}(s)} Q_*(s, a)$$

Bellman Optimality Equation

The optimal Q-function Q_* satisfies a special recursive relationship:

$$Q_*(s,a) = \mathbb{E}_{S',R}[R + \gamma \max_{a'} Q_*(S', a') | S = s, A = a]$$

Breaking down the expectation:

$$Q_*(s,a) = \sum_{s',r} P(s', r|s, a) \left[r + \gamma \max_{a'} Q_*(s', a') \right]$$

This equation provides a way to find Q_*. If we can solve it, we solve the RL problem.

Proof.

We start with the definition of the action-value function $Q^\pi(s,a)$ and the relationship between the return G_t and G_{t+1}:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \end{aligned}$$

The expectation depends on the next state S_{t+1} and reward R_{t+1}, which occur according to the dynamics $P(s', r|s, a)$, and on subsequent actions chosen by π. Let's condition on the specific next state s' and reward r:

$$\begin{aligned} Q^\pi(s,a) = \sum_{s',r} P(s', r|s, a)\, \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a, S_{t+1} = s', \\ R_{t+1} = r] \\ = \sum_{s',r} P(s', r|s, a) \left[r + \gamma\, \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \right] \end{aligned} \tag{1}$$

Proof continued.

Recognizing the definition of the state-value function $V^\pi(s') = \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']$, we have the Bellman equation for Q^π:

$$Q^\pi(s,a) = \sum_{s',r} P(s',r|s,a)\left[r + \gamma V^\pi(s')\right] \tag{2}$$

Now consider the optimal action-value function $Q_*(s,a) = \max_\pi Q^\pi(s,a)$. There exists at least one optimal policy π_* such that $Q_*(s,a) = Q^{\pi_*}(s,a)$ and $V_*(s') = V^{\pi_*}(s')$. Applying (2) to the optimal policy π_*:

$$\begin{aligned} Q_*(s,a) = Q^{\pi_*}(s,a) &= \sum_{s',r} P(s',r|s,a)\left[r + \gamma V^{\pi_*}(s')\right] \\ &= \sum_{s',r} P(s',r|s,a)\left[r + \gamma V_*(s')\right] \end{aligned} \tag{3}$$

Proof continued.

The crucial step is relating the optimal state value $V_*(s')$ back to the optimal action values Q_*. An agent in state s' following an optimal policy will choose the action a' that maximizes the optimal action-value function $Q_*(s', a')$. Therefore:

$$V_*(s') = \max_{a'} Q_*(s', a')$$

Substituting this into (3) gives the Bellman Optimality Equation for Q_*:

$$Q_*(s, a) = \sum_{s', r} P(s', r|s, a) \left[r + \gamma \max_{a'} Q_*(s', a') \right]$$

This holds for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. □

Backup Diagrams

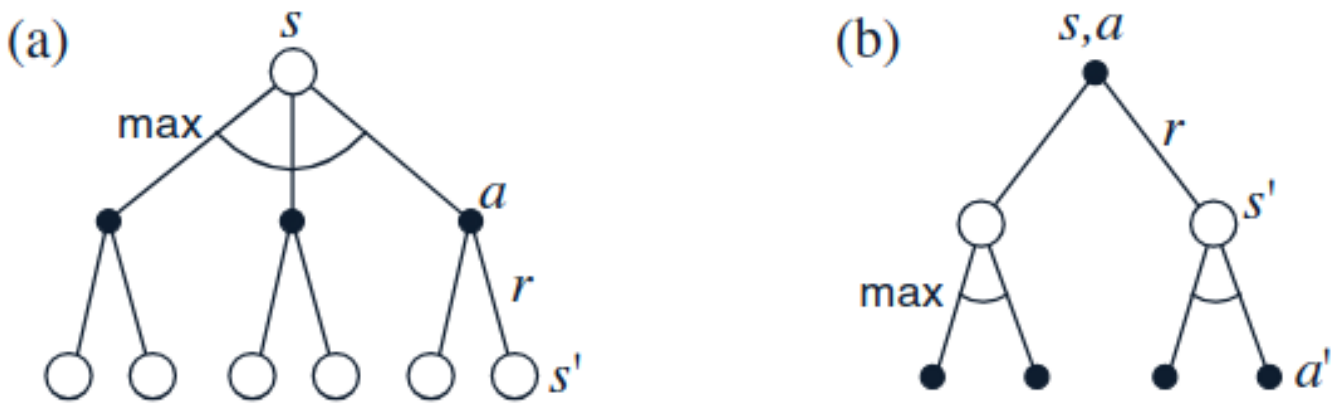


Figure 3.7: Backup diagrams for (a) v_* and (b) q_*

The backup diagrams show graphically the spans of future states and actions considered in the Bellman optimality equations for V_* and Q_*.

Key Takeaways

- RL is fundamentally about solving the Bellman Optimality Equation to find Q_*.
- The equation expresses the value of a state-action pair recursively in terms of the values of successor states.

1.4 Tabular Q-Learning

Solving without a Model

Q-learning is a foundational RL algorithm that finds the optimal action-value function Q_* directly from experience, without needing the transition probabilities $P(s', r|s, a)$ (model free).

- It's **off-policy**: Learns about the optimal (greedy) policy while possibly behaving using a different (e.g., exploratory) policy.
- It's based on **Temporal Difference (TD)** learning: Updates estimates based on other learned estimates without waiting for a final outcome (bootstrapping).

Breakthrough novel idea that combines Monte Carlo ideas and dynamic programming!
Nowadays there is a way to seamlessly integrate TD and Monte Carlo methods (through Eligibility Traces).

Advantages of TD

vs. Dynamic Programming (DP)

- **Model-Free:** TD learns directly from raw experience (samples).
- **No Need for Dynamics:** It does *not* require knowledge of the environment's transition probabilities $P(s', r|s, a)$ or reward function $R(s, a)$.
- (DP requires a full model).

vs. Monte Carlo (MC)

- **Incremental:** TD updates value estimates after *each time step*.
- **Online Learning:** It does *not* need to wait until the end of an episode to learn. Can learn from incomplete sequences.
- (MC requires complete episodes to calculate the return G_t).

Core Mechanism

TD methods **bootstrap**: they update estimates based in part on other learned estimates ($V(S_{t+1})$), allowing step-by-step updates without a model or full returns.

Tabular Q-Learning (Part 2)

The Q-Table

- Store an estimate $Q(s, a)$ for *every* possible state-action pair (s, a).
- Initialize arbitrarily (e.g., to 0).
- Requires finite state and action spaces $(\mathcal{S}, \mathcal{A})$.
- Size: $|\mathcal{S}| \times |\mathcal{A}|$.

The Q-Learning Update Rule

- After observing a transition $(S_t, A_t, R_{t+1}, S_{t+1})$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[\underbrace{R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')}_{\text{TD Target}} - Q(S_t, A_t) \right]$$

- $\alpha \in (0, 1]$ is the **learning rate**.
- The term in brackets is the **TD Error**: the difference between the estimated return (TD Target) and the current Q-value.
- Update moves the current estimate towards the TD Target.

Key Takeaways

- Q-learning iteratively refines Q-value estimates using sampled transitions.
- Simple, elegant, and guaranteed to converge to Q_* under standard conditions (if all pairs are visited enough, α decreases appropriately).
- BUT: Only feasible for small, discrete state-action spaces.
- Forms the conceptual basis for DQN.

1.5 Exploration vs. Exploitation

The Dilemma

To find the best actions, an RL agent must explore actions it hasn't tried much before. But to maximize reward, it should exploit actions that it currently estimates to be the best.

- **Exploration**: Gathering new information about the environment and action values. May involve taking suboptimal actions in the short term.
- **Exploitation**: Using current knowledge to choose the action believed to yield the highest return.

How to balance these two? This is a fundamental challenge in RL.

ϵ-Greedy Policy

A common and simple strategy:

- With probability $1 - \epsilon$: Choose the greedy action $a = \arg\max_{a'} Q(s, a')$. (Exploit)
- With probability ϵ: Choose a random action from $\mathcal{A}(s)$ uniformly. (Explore)

Typically, ϵ is small (e.g., 0.1 or 0.05) and may decrease over time and is related to the agent's risk aversity.

- **Why is exploration crucial?** Without it, the agent might converge to a suboptimal policy because it never discovered a truly better action. Q-learning needs to update all relevant (s, a) pairs.
- **Contrast with Supervised Learning**: In SL, the data is fixed, and the goal is to fit it. In RL, the agent influences the data it receives through its actions (via exploration).
- Other strategies exist (Softmax/Boltzmann exploration, Upper Confidence Bound - UCB), but ϵ-greedy is widely used, especially with DQN.

Key Takeaways

- Exploration is essential for finding optimal policies in unknown environments.
- The agent must actively try different actions to ensure it learns accurate Q-values for all relevant options.
- ϵ-greedy is a simple mechanism to enforce exploration.

Example code: Multi-Armed bandits.

1.6 Challenges in Scaling Tabular Q-Learning

Limitations of the Tabular Approach

Tabular Q-learning is elegant, but breaks down for problems with large or continuous state/action spaces.

- **Curse of Dimensionality (Memory):** The Q-table size ($|\mathcal{S}| \times |\mathcal{A}|$) grows exponentially with the number of state/action variables. Impractical for complex problems (e.g., vision, robotics).
- **Curse of Dimensionality (Data):** Requires visiting and updating every (or most) state-action pairs many times to learn accurate values. Need huge amounts of experience.
- **Lack of Generalization:** Learning about state s provides *no* information about other states, even if they are very similar. Cannot handle novel, unseen states.
- **Raw Inputs:** Cannot directly use high-dimensional sensory inputs like images or audio. Requires pre-processed, low-dimensional state features.

Solution

The Solution: Function Approximation

Instead of a table, approximate the Q-function using a parameterized function:

$$Q(s, a; \boldsymbol{\theta}) \approx Q_*(s, a)$$

- $\boldsymbol{\theta}$ are the parameters (e.g., weights of a neural network).
- **Generalization**: Similar states/actions will produce similar Q-value estimates. Can handle unseen states.
- **Scalability**: Memory requirement depends on the number of parameters $|\boldsymbol{\theta}|$, not $|\mathcal{S}| \times |\mathcal{A}|$.
- **Raw Inputs**: Deep Neural Networks (DNNs) can learn relevant features directly from high-dimensional inputs like pixels.

Summary & Next Steps

- RL learns optimal behaviour through interaction via trial-and-error.
- MDPs provide the formal framework (States, Actions, Rewards, Transitions).
- Goal: Maximize cumulative discounted reward by finding optimal policy π_* / value function Q_*.
- Bellman equation is key to relating values recursively.
- Q-learning provides a model-free way to solve the Bellman equation (tabularly).
- Exploration vs. Exploitation is a critical trade-off.
- Tabular methods fail for large problems due to the curse of dimensionality and lack of generalization.
- **Need for function approximation $\implies$ Deep RL (DQN)!**

Thank you! Questions?