Lecture 2: Fundamentals of Reinforcement Learning

Dr. Yaodong Yang

Institute for AI, Peking University

08/2023

Example 1: Reinforcement Learning in Atari Games

Success of DQN [MKS+13] in learning to play a wide range of Atari games directly from raw pixel inputs.

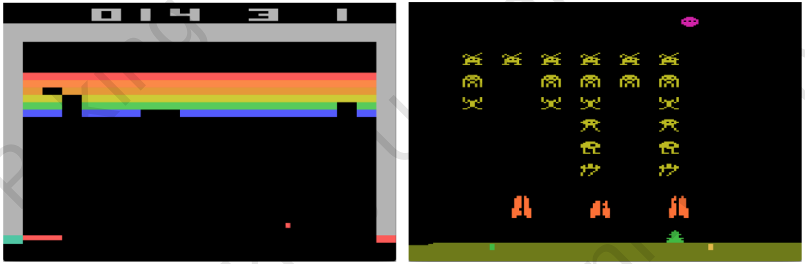

Fig. 1. Deep Q-Network (DQN) in Atari games uses a deep neural network to approximate the Q-value function and learn to play games from raw pixel inputs.

Example 2: Reinforcement Learning in AlphaZero

AlphaZero [SHS+17] combines RL, deep neural networks, and Monte Carlo Tree Search (MCTS) for game playing.

Fig. 2. AlphaZero combines RL, deep neural networks, and Monte Carlo Tree Search (MCTS) to achieremarkable success in chess, shogi, and Go.

Example 3: Reinforcement Learning in Language Models

Reinforcement Learning is used to fine-tune language models based on human-generated evaluations.

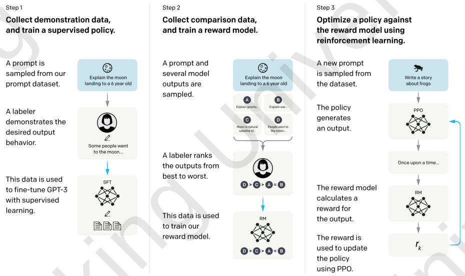
Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy using PPO.
r_k

Fig. 3. Language models utilize RL to fine-tune their performance and generate coherent and contextually relevant text based on human evaluations. [OWJ+22]

What is reinforcement learning?

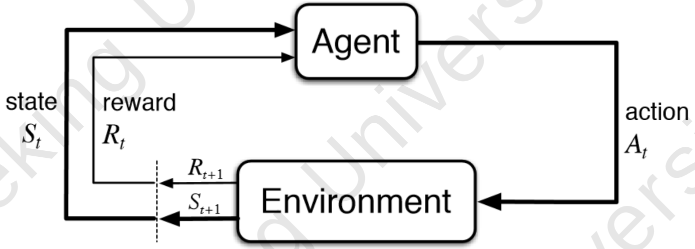

Fig. 4. A computational approach to learning whereby an agent tries to **maximize** the total amount of reward it receives while interacting with a complex and uncertain environment. Reinforcement Learning (RL) enables agents to learn from interactions with the environment to maximize cumulatirewards.

Characteristics

- **Trial-and-Error Learning**: Reinforcement learning agents learn by interacting with the environment and receiving feedback in the form of rewards or penalties. Through repeated trials, the agent refines its policy to achieve better performance.
- **Delayed Rewards**: Reinforcement learning often involves delayed rewards, where the consequences of an action may not be immediately apparent. The agent needs to learn to associate its actions with future rewards.
- **Exploration vs. Exploitation**: The agent faces the exploration-exploitation dilemma. It must explore the environment to discover potentially better actions but also exploit its current knowledge to maximize immediate rewards.

Characteristics

- **Markov Decision Process (MDP)**: Reinforcement learning problems are often formulated as MDPs, which model sequential decision-making in stochastic environments.
- **Policy and Value Functions**: Reinforcement learning employs policy and value functions to represent the agent's strategy and estimate the expected rewards from different states and actions.
- **Sample Efficiency**: Reinforcement learning algorithms should aim to achieve good performance with as few samples (interactions with the environment) as possible, especially in real-world scenarios with high costs or risks.
- **Generalization**: Effective reinforcement learning agents should be able to generalize their learning to new, unseen states or similar tasks, rather than memorizing specific experiences.

Model-free Algorithms

Model-free algorithms are a class of reinforcement learning methods that do not rely on explicitly building a model of the environment. Instead, they learn from direct interactions with the environment.

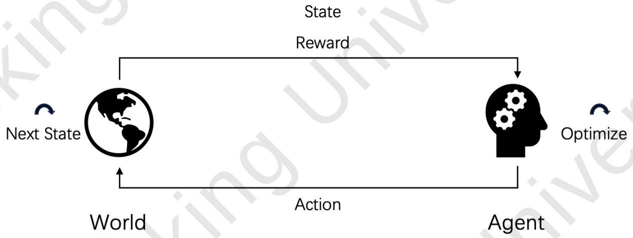

Fig. 5. Illustration of a model-free algorithm in action.

Model-based Algorithms

Model-based algorithms are a category of reinforcement learning methods that involve building an explicit model of the environment. This model allows the agent to simulate possible outcomes and plan its actions accordingly.

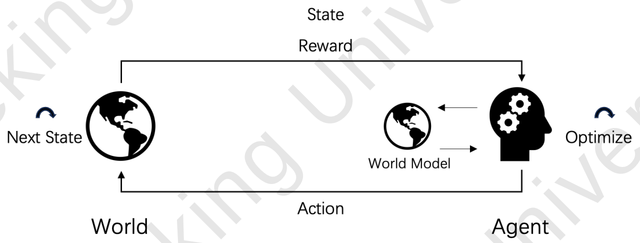

Fig. 6. Illustration of a model-based algorithm using environment simulation.

Offline Algorithms

Offline algorithms, also known as batch reinforcement learning methods, learn from pre-existing datasets without interacting directly with the environment. They are useful when gathering new data is costly or infeasible.

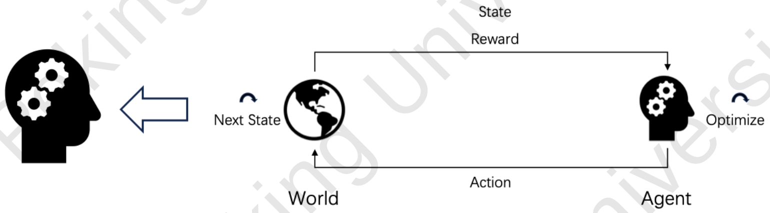

Fig. 7. Illustration of an offline reinforcement learning algorithm using historical data.

Rewards

The reward in reinforcement learning is a numerical value that measures the immediate feedback an agent receives from the environment after taking a particular action in a specific state.

- The reward function $R(s, a, s')$ measures the benefit of taking action a in state s and transitioning to state s'.
- Positive rewards encourage desirable actions and reinforce the agent's behavior in similar situations.
- Negative rewards discourage unwanted actions and steer the agent away from undesirable states or actions.
- Rewards play a crucial role in shaping the agent's learning process and guiding it towards an optimal policy that achieves the best possible cumulatireward over time.

Rewards

Here are some examples:

- **Defeat the world champion at Backgammon**
  - +reward for winning a game
  - -reward for losing a game
- **Control a power station**
  - +reward for producing power
  - -reward for exceeding safety thresholds
- **Make a humanoid robot walk**
  - +reward for forward motion
  - -reward for falling over
- **Play many different Atari games better than humans**
  - +reward for increasing the score
  - -reward for decreasing the score

Observation and State

- Observation: Partial and noisy information received by the agent from the environment.
- State: The true representation of the environment's current condition.
- Observation function $O(s) \rightarrow o$: Maps states to observations.
- State estimation function $B(o) \rightarrow b$: Estimates the belief state based on observations.

In partially observable environments, an agent receives observations o_t instead of the true state s_t.

Observation and State

Here are some examples of observation and state.

- *Autonomous Driving*: The car's sensors (camera, lidar, radar) provide observations, and the observation function maps these observations to the current understanding of the car's environment.
- *Robot Navigation*: A robot equipped with range sensors and cameras receives observations of its surroundings, and the state estimation function estimates the probability distribution of the robot's location and map.
- *Healthcare Monitoring*: A wearable health device collects various physiological data, and the observation function converts these data into useful information for healthcare professionals.

State Transition Matrix

Definition

The State transition matrix, denoted as $\mathbf{A}$, is a fundamental concept in the theory of linear time-invariant systems. It describes how the state of a system evolves over time.

- For a discrete-time system, the state transition from time instant k to $k+1$ is given by:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$$

where $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ are the state vectors at time k and $k+1$, respectively.

State Transition Matrix

- For a continuous-time system, the state transition equation becomes:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t)$$

  where $\mathbf{x}(t)$ is the state vector at time t, and $\dot{\mathbf{x}}(t)$ is its derivative.
- The State transition matrix is a critical component in analyzing the stability and response characteristics of linear systems.
- Markov property: The future state depends only on the current state and action, not the history.
- MDP assumption: The environment follows the stationary and stochastic process.

MDP Process

Definition

The Markov Decision Process (MDP) is a tuple $\langle S, A, P, R \rangle$, where:The Markov Decision Process (MDP) is a tuple $\langle S, A, P, R \rangle$, where:

- S: Set of states
- A: Set of actions
- $P(s'|s,a)$: State transition matrix
- $R(s,a,s')$: Reward function
- The agent interacts with the environment through a sequence of actions, observations, and rewards.

Policy and Value Function

- Policy $\pi(a|s)$: A mapping from states to probabilities of selecting actions.
- Value function $V(s)$: The expected cumulative reward starting from state s under policy π.
- Action-value function $Q(s, a)$: The expected cumulative reward starting from state s, taking action a, and following policy π.
- The policy determines the agent's behavior in an environment by specifying the probabilities of selecting different actions in different states.
- The value functions provide estimates of the total expected reward that an agent can achieve from a given state, and they are essential in guiding the agent's decision-making process in reinforcement learning tasks.

Example

In this example, let's consider a simple grid world environment where an agent moves in a 3x3 grid to reach a goal state. The agent can take actions UP, DOWN, LEFT, or RIGHT.

State	Action: UP	Action: DOWN	Policy
S_1	0.3	0.2	0.5 UP, 0.5 DOWN
S_2	0.6	0.1	0.7 UP, 0.3 DOWN
S_3	0.1	0.9	0.1 UP, 0.9 DOWN

Table. 1. Example policy probabilities for different states. The policy determines the agent's behavior in each state by specifying the probabilities of selecting different actions. In this example, the agent's policy is 0.5 UP, 0.5 DOWN in state S_1, 0.7 UP, 0.3 DOWN in state S_2, and 0.1 UP, 0.9 DOWN in state S_3. These probabilities indicate that the agent is equally likely to choose UP or DOWN in S_1, more likely to choose UP in S_2, and more likely to choose DOWN in S_3.

Example

The **value function** $V(s)$ represents the expected cumulative reward starting from each state under a given policy. In this example, the value function for different states is $V(S_1) = 6$, $V(S_2) = 8$, and $V(S_3) = 3$. These values indicate the expected rewards the agent can achieve starting from each state following the specified policy.

State	**Value Function**	**Policy**
S_1	6	0.5 UP, 0.5 DOWN
S_2	8	0.7 UP, 0.3 DOWN
S_3	3	0.1 UP, 0.9 DOWN

Table. 2. Example value function and policy for different states.

Bellman Equation

- The Bellman equation expresses the relationship between value functions.
- Bellman Expectation Equation for $V(s)$:

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

- Bellman Expectation Equation for $Q(s,a)$:

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

Policy Improvement

- Policy π' is better than policy π if $V^{\pi'}(s) \geq V^{\pi}(s)$ for all states s.
- Policy Improvement Theorem: If π' is better than π, then π' is also a greedy policy with respect to V^{π}.
- The process of iteratively improving the policy is called policy iteration.

Value Iteration

- Value Iteration is an iteratialgorithm to find the optimal value function $V^*(s)$.
- It performs Bellman backup updates until the value function converges to the optimal solution.
- Value Iteration often converges faster than policy iteration.

Advantage

Definition

In reinforcement learning, the **advantage** is a measure of how much better an action is compared to the average action selection under a specific policy in a given state.

- The advantage function $A(s, a)$ is defined as the difference between the action-value function $Q(s, a)$ and the state-value function $V(s)$:

$$A(s, a) = Q(s, a) - V(s)$$

- A positive advantage indicates that the action is better than the average, while a negative advantage suggests that the action is worse than the average.

Example of Advantage Function

In this example, let's consider a simple grid world environment where an agent moves in a 3x3 grid to reach a goal state. The agent can take actions UP, DOWN, LEFT, or RIGHT. Each step incurs a reward of -1, and reaching the goal state gives a reward of +10. We will calculate the advantage function for a specific state-action pair.

Example of Advantage Function

Consider the state-action pair (State: S_1, Action: UP) in the grid world. The value function $V(S_1)$ is 6, and the action-value function $Q(S_1, \text{UP})$ is 7.
The advantage function $A(S_1, \text{UP})$ is given by:

$$A(S_1, \text{UP}) = Q(S_1, \text{UP}) - V(S_1) = 7 - 6 = 1$$

State	Action	Advantage
S_1	UP	1
S_2	DOWN	2
S_3	LEFT	0
S_4	RIGHT	-1

Table. 3. Advantage function values for different state-action pairs.

Monte Carlo (MC) Method in Reinforcement Learning

Introduction

The Monte Carlo (MC) method is a model-free reinforcement learning algorithm that estimates the value of states based on sample returns obtained through simulation.

Key Idea

Unlike other value estimation methods, MC does not require a model of the environment (transition probabilities) and directly learns from sample experiences.

Monte Carlo (MC) Method in Reinforcement Learning

MC Prediction (Value Estimation)

Given a policy π, MC estimates the state-value function $V_\pi(s)$ for a state s as the average return observed from simulations:

$$V_\pi(s) \approx \frac{1}{N}\sum_{i=1}^{N} G_i(s),$$

where N is the number of episodes, and $G_i(s)$ is the return observed for state s in the i-th episode.

Monte Carlo (MC) Method in Reinforcement Learning

MC Control (Policy Improvement)

MC can also be used to improve the policy by making it *greedy* with respect to the current value estimates:

$$\pi'(s) = \text{argmax}_a Q(s, a),$$

where $Q(s, a)$ is the action-value function, and $\pi'(s)$ is the improved policy at state s.

Temporal Difference (TD) Method in Reinforcement Learning

Introduction

The Temporal Difference (TD) method is another model-free reinforcement learning algorithm that combines elements of both Monte Carlo and Dynamic Programming approaches.

Key Idea

TD learning updates its value estimates based on the observed reward and the estimated value of the next state, without requiring complete episodes as in Monte Carlo.

Temporal Difference (TD) Method in Reinforcement Learning

TD(0) (One-Step TD)

Given a policy π, TD(0) estimates the state-value function $V_\pi(s)$ by bootstrapping from the current state:

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha \left[r_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t) \right],$$

where:

- s_t is the current state,
- r_{t+1} is the reward observed after taking an action from s_t,
- s_{t+1} is the next state after the action,
- α is the learning rate, and
- γ is the discount factor.

n-step Temporal Difference (TD) Method

Introduction

The n-step Temporal Difference (TD) method is an extension of TD learning that considers multi-step returns, allowing for a trade-off between the bootstrap updates of TD(0) and the full episode returns of Monte Carlo.

Key Idea

In n-step TD learning, value estimates are updated using rewards and state values observed over multiple time steps (n), bridging the gap between one-step TD and Monte Carlo.

n-step Temporal Difference (TD) Method

n-step Return

The n-step return, denoted as $G_t^{(n)}$, is the sum of rewards obtained over n time steps, plus the discounted value of the state reached after n steps:

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}),$$

where:

- s_t is the current state,
- r_{t+i} is the reward observed after taking an action from s_{t+i} for $i = 1$ to n,
- s_{t+n} is the state reached after n steps, and
- γ is the discount factor.

Overview

Aspect	Exploration	Exploitation
Goal	Discover new strategies and possibilities	Exploit the best-known strategies
Strategy	Random or uncertain actions	Actions based on current knowledge
Risk	Higher risk of low immediate rewards	Lower risk, usually higher immediate rewards
Time	More time-consuming	Less time-consuming
Trade-off	Short-term sacrifice for long-term gain	Focused on short-term rewards
Learning Phase	Early stages of learning	Later stages of learning

Possible plan

Strategy	Description
Epsilon-Greedy	Balances exploration and exploitation using a parameter ε to control the exploration rate.
Softmax	Chooses actions probabilistically, giving lower probabilities to low-value actions, promoting exploration.
Upper Confidence Bound (UCB)	Uses confidence bounds to estimate action values and encourages exploration of uncertain actions.
Thompson Sampling	Bayesian approach that randomly samples action values from posterior distributions to explore and exploit.

Epsilon-Greedy Policy

Definition

The Epsilon-Greedy policy is a popular exploration-exploitation strategy used in reinforcement learning.

Policy

At each time step, the agent selects an action according to the following probability distribution:

$$\text{With probability } \varepsilon : \begin{cases}\text{Select a random action with uniform probability.}\end{cases}$$

$$\text{With probability } (1-\varepsilon) : \begin{cases}\text{Select the action with the highest estimated reward based on the c}\end{cases}$$

where ε is a small positive value between 0 and 1.

Epsilon-Greedy Policy in Toy Game

Scenario

Consider a simple toy game where a player is presented with three slot machines (arms) to choose from. Each slot machine provides a reward when pulled, but the player doesn't know the true reward probabilities associated with each arm.

Epsilon-Greedy Strategy

Let's apply the Epsilon-Greedy policy to this game with $\varepsilon = 0.3$, meaning there is a 30% chance of exploration in each time step.

- With 30% probability, the player randomly selects one of the three slot machines to explore.
- With 70% probability, the player selects the slot machine that has provided the highest average reward so far.

Epsilon-Greedy Policy in Toy Game

Slot Machine	Estimated Reward Value
1	0
2	0
3	0

Table. 4. Estimated reward values for each slot machine (initialized to zero).

- At each time step, generate a random number between 0 and 1.
- If the random number $\leq \varepsilon$, choose exploration:
  - Randomly select one of the three slot machines (with uniform probability) and pull its arm.
- If the random number $> \varepsilon$, choose exploitation:
  - Select the slot machine with the highest estimated reward value and pull its arm.

Epsilon-Greedy Policy in Toy Game

Update Estimated Rewards

After pulling the arm of a slot machine, observe the obtained reward.

Slot Machine	Estimated Reward Value
1	0
2	5 (Example: Obtained reward)
3	0

Table. 5. Updated estimated reward values after pulling Slot Machine 2.

Repeat

Continue the process of action selection, reward observation, and updating estimated rewards for a certain number of time steps or until a stopping criterion is met.

SARSA Algorithm

Introduction

SARSA is an on-policy model-free reinforcement learning algorithm that learns the action-value function by updating the Q-values during each step of interaction with the environment.

Key Idea

SARSA stands for "State-Action-Reward-State-Action," indicating that the updates are based on transitions from one state-action pair to another state-action pair.

SARSA Algorithm

Given a policy π, the SARSA update for the action-value function $Q(s, a)$ at each time step is defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right],$$

where:

- s_t is the current state,
- a_t is the current action,
- r_{t+1} is the reward observed after taking action a_t from state s_t,
- s_{t+1} is the next state after taking action a_t,
- a_{t+1} is the next action chosen by the policy, and
- α is the learning rate, and γ is the discount factor.

SARSA Algorithm

On-Policy Learning

SARSA is an on-policy algorithm because it follows the same policy π for action selection and updates the action-value function using the values from the next state-action pair.

Policy Improvement

To ensure convergence, SARSA typically employs an ε-greedy policy, which balances exploration and exploitation during learning.

Q-Learning Algorithm

Introduction

Q-Learning is an off-policy model-free reinforcement learning algorithm that learns the action-value function by estimating the optimal Q-values and following a different policy for action selection.

Key Idea

Q-Learning updates the action-value function by considering the maximum Q-value of the next state, allowing for policy improvement without being restricted to a specific policy.

Q-Learning Algorithm

The Q-Learning update for the action-value function $Q(s, a)$ at each time step is defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right],$$

where:

- s_t is the current state,
- a_t is the current action,
- r_{t+1} is the reward observed after taking action a_t from state s_t,
- s_{t+1} is the next state after taking action a_t, and
- α is the learning rate, and γ is the discount factor.

Q-Learning Algorithm

Off-Policy Learning

Q-Learning is an off-policy algorithm as it learns the optimal action-value function independently of the policy being followed during action selection.

Greedy Policy

After learning the optimal action-value function, Q-Learning follows a greedy policy to choose actions with the highest Q-values at each state.

Summary and Outlook

In this lecture, we covered the fundamentals of reinforcement learning:

- Introduction of reinforcement learning.
- Markov decision process and Bellman equation.
- MC and TD method.
- SARSA and Q-learning.

In the next lecture, we will introduce:

- On-policy algorithms
- Off-policy algorithms

Thanks!

References I

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller.
Playing atari with deep reinforcement learning.
arXiv preprint arXiv:1312.5602, 2013.

[OWJ+22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.
Training language models to follow instructions with human feedback.
Advances in Neural Information Processing Systems, 35:27730–27744, 2022.

[SHS+17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al.
Mastering chess and shogi by self-play with a general reinforcement learning algorithm.
arXiv preprint arXiv:1712.01815, 2017.