CS-566 Deep Reinforcement Learning

Model-Free Learning - III
Learning from Rewards

Nazar Khan
Department of Computer Science
University of the Punjab

Three Principles of Model-Free Learning

- There are three principles of model-free learning.
  1. *Reward Sampling*: How should rewards be sampled from the environment?
  2. *Action Selection*: How does the agent decide which action to take?
  3. *Learning from Rewards*: How to use reward to make the agent better?
- Different answers to these questions lead to different methods of reinforcement learning.
- In this lecture: Learning from Rewards.

Learning Methods in Reinforcement Learning

- Beyond action selection, a key design question is:
  - Which **learning method** to use?
- RL is about learning an **action-policy** from rewards
- Two main approaches:
  1. **On-policy learning**
  2. **Off-policy learning**

On-policy Learning

- Agent selects an action using the current policy
- The **value of that chosen action** is used to update the policy
- Learning is tied directly to the behavior of the policy

Key Idea

Update policy values using the **action actually taken**.

Off-policy Learning

- Learning uses values of **another action**, not necessarily the chosen one
- Makes sense during exploration:
  - Behavior policy may select a *non-optimal* action
  - On-policy learning would back up its inferior value
  - Off-policy learning instead backs up the **best action's value**
- Advantage:
  - Avoids "polluting" the policy with bad exploratory actions

On-Policy SARSA

Idea

- **On-policy** algorithm: learns from the action actually taken.
- Uses the same policy for both:
  - **Action selection (behavior policy)**
  - **Target updates (learning policy)**
- Typical choice: ϵ-greedy exploration.

SARSA Update Rule

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

- Uses the next action a_{t+1} *chosen by the current policy*.
- Predictive: learns directly from behavior values.

SARSA Intuition

- Agent follows its policy π (possibly ϵ-greedy).
- Updates Q-values using *the same action it just took*.
- Policy gradually improves while respecting its own exploration.

Off-Policy Q-Learning

Idea

- **Off-policy** algorithm: learns as if it always followed a greedy policy.
- Behavior policy may explore, but updates are from the *best possible action*.

Q-Learning Update Rule

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

- Uses $\max_a Q(s_{t+1}, a)$ instead of $Q(s_{t+1}, a_{t+1})$.
- Learns from **greedy action**, not the exploratory one.

Q-Learning Intuition

- Behavior policy may try exploratory actions.
- But updates pretend the agent acted greedily.
- Leads to convergence to the **optimal policy**.

SARSA vs. Q-Learning

SARSA (On-policy)

- Learns values of *current behavior*.
- More stable (lower variance).
- May converge to sub-optimal policy if ϵ is fixed.

Q-Learning (Off-policy)

- Learns values of *greedy policy*.
- Converges to optimal $Q^\star$ (low bias).
- Can be unstable with function approximation (max operator).

On-policy vs. Off-policy (Summary)

On-policy	**Off-policy**
Updates from the action actually taken	Updates from the *best* action
Tied to behavior policy	Separate behavior + target policy
Exploration actions may lower value estimates	More efficient during exploration
SARSA[1]	**Q-learning**

[1] Name from the update tuple (s, a, r, s', a')

Convergence Behavior

- Proven convergence in tabular RL when policy is:
  - Greedy in the limit with infinite exploration (GLIE)
- Off-policy methods:
  - Learn from greedy rewards
  - $\Rightarrow$ Converge to optimal policy after enough samples
- On-policy methods:
  - With fixed ϵ, never fully converge (keep exploring)
  - With decaying $\epsilon \to 0$, do converge to greedy policy

Sparse vs Dense Rewards

- **Dense reward:** Every state has a reward.
  - Example: supermarket (cost per step $\rightarrow$ negative reward).
- **Sparse reward:** Rewards only at special states.
  - Example: chess (only win/draw/loss at terminal positions).

Challenges of Sparse Rewards

- Harder to find good policies.
- Reward landscape: flat with rare sharp peaks.
- Gradient often zero $\Rightarrow$ optimization difficult.

Reward Shaping

- Modify reward function $\rightarrow$ easier optimization.
- Encodes heuristic knowledge into MDP.
- Common in board games (heuristics in chess, checkers).
- Classic reference: Ng et al. (1999)[2].

[2] Andrew Y Ng, Daishi Harada, and Stuart Russell. 'Policy invariance under reward transformations: Theory and application to reward shaping'. In: *International Conference on Machine Learning*. Vol. 99. 1999, pp. 278–287.

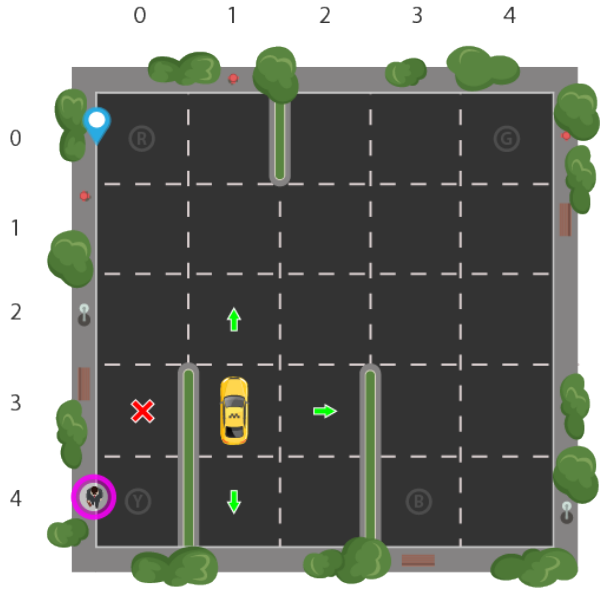
Hands-On: Q-Learning on Taxi

Hands-On: Q-learning on Taxi

- **Value Iteration:** works if transition model is known.
- **Q-learning:** model-free; learns by sampling.
- Stores rewards in a **Q-table**, approximating $Q(s, a)$.
- Once best actions are known for all states $\rightarrow$ optimal policy.

Taxi Environment Setup

- Grid world: $5 \times 5 = 25$ locations.
- State space size:
  25 (taxi positions) $\times$ 5 (passenger states) $\times$ 4 (destinations) $= 500$
- Actions: up, down, left, right, pick-up, drop-off.
- Rewards (Gym Taxi):
  - $+20$: successful drop-off.
  - -1: each time step.
  - -10: illegal drop-off.



Q-learning Intuition

- Goal: learn a policy $\pi(s)$ maximizing cumulative reward.
- Q-values = expected rewards for (s, a).
- Stored in array $Q(s, a)$, updated with experience.
- Use ϵ-greedy policy:
  - Best action most of the time.
  - Random action occasionally (exploration).

Q-learning Update Rule

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

- α: learning rate ($0 < \alpha \leq 1$).
- γ: discount factor ($0 \leq \gamma \leq 1$).
- Bootstrapping: update current Q using next state's Q.
- Q-table initialized randomly; values converge over time.

Q-learning Implementation

```
# Q learning for OpenAI Gym Taxi environment
import gymnasium as gym
import numpy as np
import random
#Environment Setup
env = gym.make("Taxi-v2")
env.reset()
env.render()
# Q[state,action] table implementation
Q = np.zeros([env.observation_space.n, env.action_space.n])
gamma = 0.7    # discount factor
alpha = 0.2    # learning rate
epsilon = 0.1 # epsilon greedy
for episode in range(1000):
    done = False
    total_reward = 0
    state = env.reset()
    while not done:
```

Q-learning Implementation

```
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample() # Explore state
                space
        else:
            action = np.argmax(Q[state]) # Exploit learned
                values
        next_state, reward, done, info = env.step(action) #
            invoke Gym
        next_max = np.max(Q[next_state])
        old_value = Q[state,action]

        new_value = old_value + alpha * (reward + gamma *
            next_max - old_value)

        Q[state,action] = new_value
        total_reward += reward
        state = next_state
    if episode % 100 == 0:
```

Q-learning Implementation

```
        print("Episode {} Total Reward: {}".format(episode,
            total_reward))
```

Algorithm Summary

1. Initialize Q-table randomly.
2. Choose initial state s.
3. Select action a from s:
   - Greedy or ϵ-random.
4. Execute a, observe r, s', update Q.
5. Repeat until terminal state.
6. Continue until Q-table converges.

Evaluating the Learned Policy

```
total_epochs, total_penalties = 0, 0
ep = 100
for _ in range(ep):
    state = env.reset()
    epochs, penalties, reward = 0, 0, 0
    done = False
    while not done:
        action = np.argmax(Q[state])
        state, reward, done, info = env.step(action)
        if reward == -10:
            penalties += 1
        epochs += 1
    total_penalties += penalties
    total_epochs += epochs
print(f"Results after {ep} episodes:")
print(f"Average timesteps per episode: {total_epochs / ep}")
print(f"Average penalties per episode: {total_penalties / ep}")
```

Tuning Hyperparameters

- Exploration ϵ: balance between exploration/exploitation.
- Discount γ: close to 1 for long-term reward.
- Learning rate α: small values stabilize learning.
- **Warning:** high α can cause divergence.

Tip: Start with $\gamma \approx 0.9$, $\alpha \approx 0.1$, $\epsilon \approx 0.1$.

Takeaways

- Q-learning is model-free and effective in discrete problems.
- Builds Q-table of expected rewards $\rightarrow$ optimal policy.
- Taxi world: small, fast, builds intuition.
- Key to mastery: experiment with hyperparameters!

Summary

- Value functions can be learned **without a transition model**, by sampling the environment.
- **Model-free methods:**
  - Use irreversible actions.
  - Sample states and rewards using exploration/exploitation trade-off.
  - Apply backup rules with bootstrapping.
- On-policy (SARSA): follows the chosen behavior policy, including explorative actions.
- Off-policy (Q-learning): always follows the value of the best action.
- Both use tabular representations of the value function.

Next: Function approximation with deep neural networks for high-dimensional state spaces.