EC332 Machine Learning

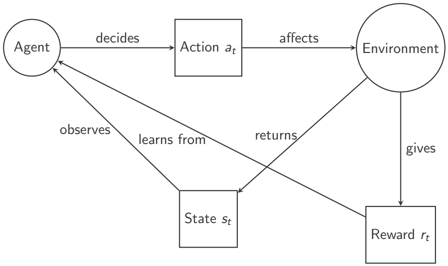
Reinforcement Learning: Balancing a CartPole with Q-Learning

Nazar Khan
Department of Computer Science
University of the Punjab

What is Reinforcement Learning?

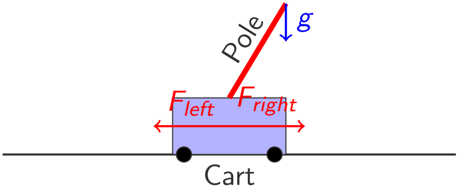
- RL is a learning paradigm where an agent learns by interacting with an environment.
- Goal: Maximize cumulative reward over time.
- Key components:
    - **Agent**: Learns and takes actions.
    - **Environment**: Provides feedback through states and rewards.
    - **Policy**: Maps states to actions.

Reinforcement Learning Loop



CartPole Environment

- Objective: Balance a pole on a moving cart by applying left or right forces.
- A training **episode** ends when:
  - Pole angle exceeds a threshold.
  - Cart moves out of bounds.
- Rewards:
  - $+1$ for each time step the pole is balanced.



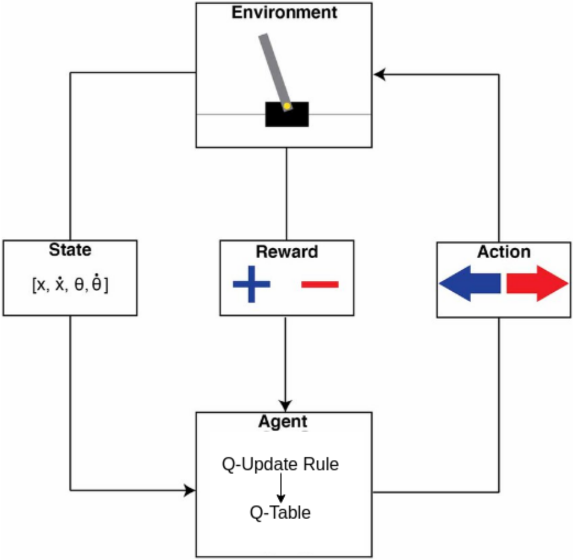
What is Q-Learning?

- Model-free RL algorithm.
- Uses a Q-table to estimate the value of state-action pairs (s, a).
- $Q(s, a)$ refers to the *quality* of taking action a in state s in terms of expected future rewards.
- A Q-table for a grid world with 3 states (S1, S2, S3) and 4 actions (up, down, left, right):

	Up	**Down**	**Left**	**Right**
S1	0.5	0.2	-0.1	0.0
S2	0.0	0.3	0.1	0.4
S3	-0.2	0.1	0.2	0.5

In state S1, best action is to move up.
In states S2 and S3, best action is to move right.

Q-Learning for CartPole Balancing



Modified from Maffettone et al. 2021.

The Q-Update Rule

Q-learning uses the following update rule iteratively:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$$

- $Q(s,a)$ represents the current estimate of the Q-value for taking action a in state s.
- s is the **current state** of the agent in the environment.
- a is the **action** chosen by the agent in the current state s.

The Q-Update Rule

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

- α is the **learning rate** (a scalar value between 0 and 1).
  - Controls how much the new information $r + \gamma \max_{a'} Q(s', a')$ influences the Q-value update.
  - Smaller α results in slower updates, preserving past knowledge.
- r is the immediate **reward** received after taking action a in state s.
- γ is the **discount factor** (a scalar value between 0 and 1).
  - Determines the importance of future rewards.
  - A value close to 0 makes the agent short-sighted (focuses only on immediate rewards), while a value closer to 1 considers long-term rewards.

The Q-Update Rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha \big[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \big]$$

- s' is the next state after taking action a in state s.
- $\max_{a'} Q(s', a')$ is the maximum Q-value of the next state s', considering all possible actions a'.
  - This represents the agent's estimate of the best possible future reward it can achieve from the next state s'.
- $r + \gamma \max_{a'} Q(s', a')$ represents the "target" Q-value, which combines the immediate reward r and the discounted estimate of future rewards.

The Q-Update Rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha \big[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \big]$$

is equivalent to

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \big[r + \gamma \max_{a'} Q(s',a') \big]$$

- So updated Q-value is a weighted average of the old Q-value $Q(s,a)$ and the target Q-value $r + \gamma \max_{a'} Q(s',a')$.
- This update process adjusts the Q-values iteratively, helping the agent improve its policy over time by reinforcing actions that lead to higher rewards.

Key Functions in Q-Learning

- **Discretize States**: Bucket continuous states into discrete bins.
- **Select Action**: ϵ-greedy policy.
  - ϵ chance of taking random action (**Exploration**)
  - $1 - \epsilon$ chance of taking action with highest Q-value (**Exploitation**)
- **Update Q-Table**: Apply Q-learning update formula.
- **Decay** ϵ: Gradually reduce exploration.

Training the Agent

- Initialize Q-table with zeros.
- For each episode:
    - Reset environment.
    - Take actions, observe rewards, and update Q-table.
- Decay ϵ after each episode.

```
state, _ = env.reset()
state = agent.discretize(state)
for t in range(max_steps):
    action = agent.select_action(state)
    next_state, reward, done, _, _ = env.step(action)
    next_state = agent.discretize(next_state)
    agent.update_q_table(state, action, reward, ...
                         next_state, done)
    state = next_state
    if done:
        break
```

Conclusion

- Q-learning enables agents to learn effective policies through trial and error.
- Key challenges:
  - Balancing exploration and exploitation.
  - Tuning hyperparameters.
  - Agent needs to be deployed in and learn from the *live environment* or its *digital twin*.
- Extendable to more complex environments.