



University: Isfahan University - Faculty of Computer Engineering

Course: Fundamentals and Applications of Artificial Intelligence

Reinforcement Learning – Q Learning Algorithm

Course Instructor: Dr. Hossein Karshenas

Student Name: **Sheida Abedpour**

Student ID: 4003623025

Team\_id = 17

2023 Dec

## Reinforcement Learning:

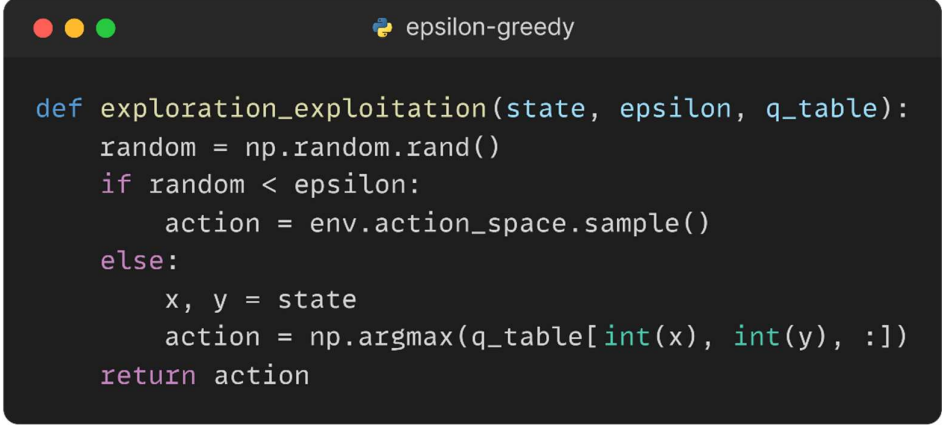
Reinforcement Learning (RL) is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or punishments based on the actions it takes. The goal of RL is for the agent to learn a policy, a strategy that maps states to actions, in order to maximize the cumulative reward over time.

When the environment dynamics are already known, the problem becomes fairly easy. The agent can come up with the optimal policy based on some simple dynamic programming concepts, because no exploration is needed. This is known as **model-based**, where the agent bases its actions on the model of the environment. However, when the environment dynamics are unknown, the problem becomes much harder. This is called **model-free**, where there needs to be a balance between exploration and exploitation. This is where temporal difference learning comes in. **Temporal difference learning (TD)** is a class of model-free RL methods which learn by bootstrapping the current estimate of the value function. There are two different TD algorithms, namely **on-policy** and **off-policy**. On-policy uses the same strategy for both the behaviour and target policy. Off-policy algorithms use a different strategy for the behaviour and target policy.

## Exploration & exploitation:

In order for the agent to develop an optimal policy, it needs to make a trade-off between exploitation and exploration. The goal of an agent is to take actions that maximise the long-term reward. However, the agent doesn't know which actions maximise the reward because environment dynamics are unknown. To discover these actions, it must try actions that it has never selected before. So there is a consideration between exploring new actions or exploiting rewards from familiar actions. This is where action selection strategies come into play. **Action selection** is the strategy where the agent bases its selection of actions on. The most basic strategy is the greedy strategy, which always goes for the highest reward. Another strategy is  **$\epsilon$ -greedy** which is a strategy that balances exploration and exploitation based on the epsilon parameter. Instead of always selecting the action with the best estimated value from the **Q-table**, it gives a small probability (the epsilon) to select possible actions uniformly and

randomly. This makes sure the agent doesn't overlook actions that potentially result in higher rewards.



```
def exploration_exploitation(state, epsilon, q_table):
    random = np.random.rand()
    if random < epsilon:
        action = env.action_space.sample()
    else:
        x, y = state
        action = np.argmax(q_table[int(x), int(y), :])
    return action
```

[codesnap.dev](https://codesnap.dev)

### Q-table:

The agent must base its actions on previously experienced rewards. It does this by keeping a table with Q-values. It stores each Q-value for a state-action pair.

### Q\_Learning:

Q-Learning is an off-policy and model-free learning method. It updates the Q-value for a certain action based on the obtained reward from the next state and the maximum reward from the possible states after that.

The objective of the model is to find the best course of action given its current state. To do this, it may come up with rules of its own or it may operate outside the policy given to it to follow. This means that there is no actual need for a policy, hence we call it off-policy.

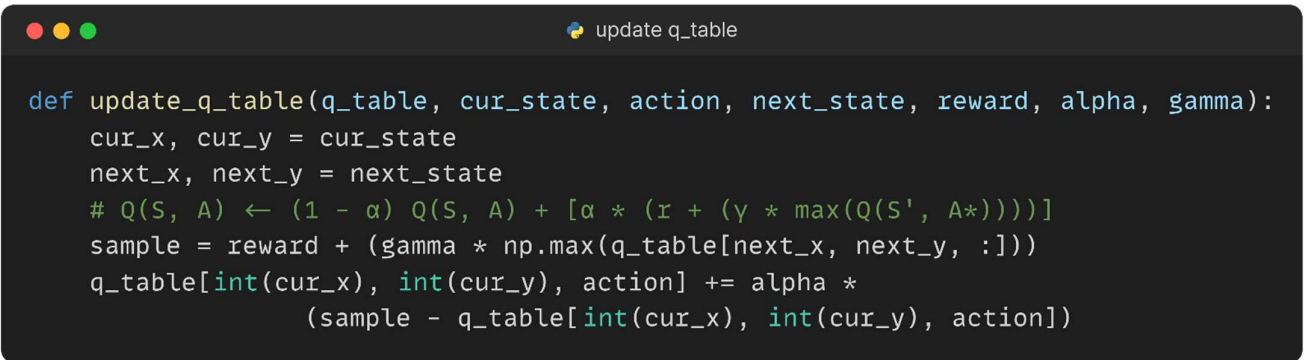
Model-free means that the agent uses predictions of the environment's expected response to move forward. It does not use the reward system to learn, but rather, trial and error.

The Q-learning algorithm is based on the **Bellman equation**, which expresses the relationship between the value of a state and the values of its successor states. Q-learning uses the Bellman equation to iteratively update the Q-values.

$$Q(S, A) = (1 - \alpha) * Q(S, A) + \alpha [R(S, A) + \gamma * \max_{\hat{A}} \hat{Q}(\hat{S}, \hat{A})]$$

or:

$$Q(S, A) = Q(S, A) + \alpha [R(S, A) + \gamma * \max_{\hat{A}} \hat{Q}(\hat{S}, \hat{A}) - Q(S, A)]$$



```
def update_q_table(q_table, cur_state, action, next_state, reward, alpha, gamma):
    cur_x, cur_y = cur_state
    next_x, next_y = next_state
    # Q(S, A) ← (1 - α) Q(S, A) + [α * (r + (γ * max(Q(S', A*))))]
    sample = reward + (gamma * np.max(q_table[next_x, next_y, :]))
    q_table[int(cur_x), int(cur_y), action] += alpha *
        (sample - q_table[int(cur_x), int(cur_y), action])
```

[codesnap.dev](https://codesnap.dev)

```
def q_learning(num_episodes, q_table, epsilon, alpha, gamma):  
  
    for i in range(num_episodes):  
  
        # start from the initial state  
        cur_state = env.reset()  
        game_over = False  
  
        while not game_over:  
  
            # choose an action  
            action = exploration_exploitation(cur_state, epsilon, q_table)  
            if epsilon > 0 and num_actions != 0:  
                epsilon -= ((0.00001 * i) / (num_actions))  
  
            # Perform the action and receive feedback from the environment  
            next_state, reward, done, truncated = env.step(action)  
  
            if truncated:  
                break  
  
            if done:  
                game_over = True  
  
            else:  
                total_reward += reward  
  
            # update Q_table  
            update_q_table(q_table, cur_state, action, next_state, reward, alpha, gamma)  
  
            # update state  
            cur_state = next_state  
  
    return q_table, actions, episode_steps, episode_rewards
```

## References:

- Chat-GPT
- [medium.com/codex/temporal-difference-learning-sarsa-vs-q-learning](https://medium.com/codex/temporal-difference-learning-sarsa-vs-q-learning)
- [medium.com/@kimrodrikwa/temporal-difference-rl-sarsa-vs-q-learning](https://medium.com/@kimrodrikwa/temporal-difference-rl-sarsa-vs-q-learning)
- [github.com/mirqwa/reinforcement-learning](https://github.com/mirqwa/reinforcement-learning)

## Python Libraries:

- Gym
- NumPy
- Time
- Matplotlib