Starkes Studium.
Prima Zukunft.

Hochschule Heilbronn

Technik Wirtschaft Informatik

Autonome Systeme: Achitecture and Planning

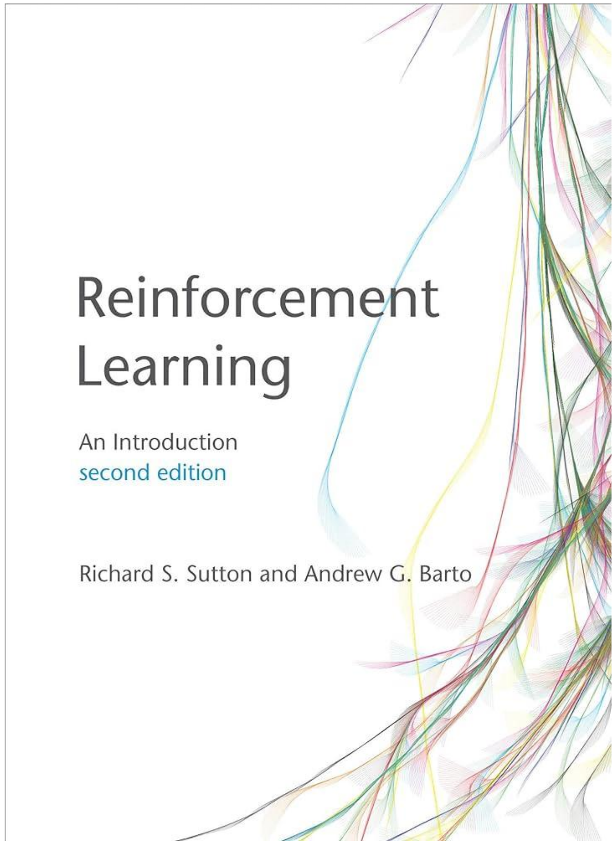
Prof. Dr.-Ing. Raoul D. Zöllner

Campus Heilbronn

Planning Based on Reinforcement Learning (Chapter 10)

Book: Reinforcement Learning An Introduction: second edition

- R. S. Sutton & G. Barto
- MIT press



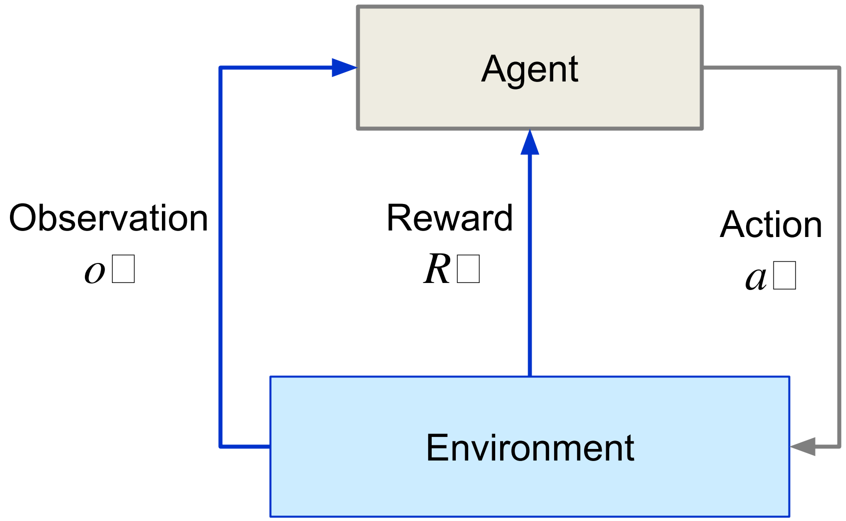
Course: Deep RL course

- Hugging Face – The AI community building the future.

Reinforcement Learning: Introduction

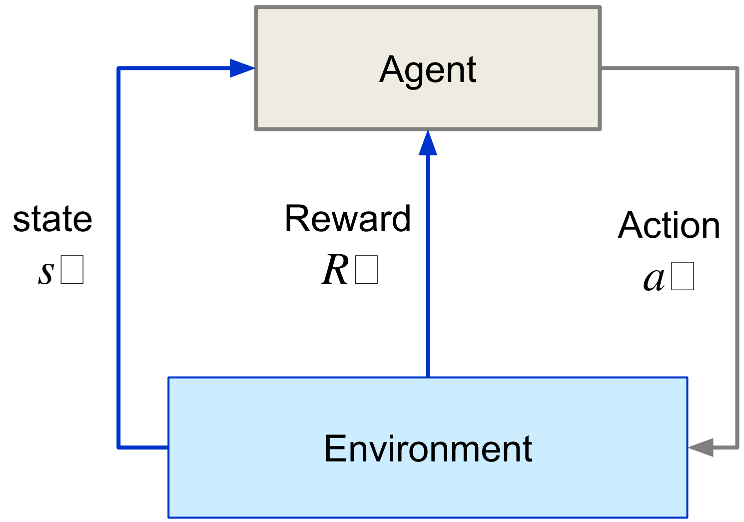
- Agent learns how to behave by interacting with the environment

- At each state ($s_{\square}$ ϵ S)
  - Agent takes an action ($a_{\square}$ ϵ A)
  - Transitions to state ($s_{(\square+1)}$ ϵ S)
  - Obtains a reward ($R_{\square}$)

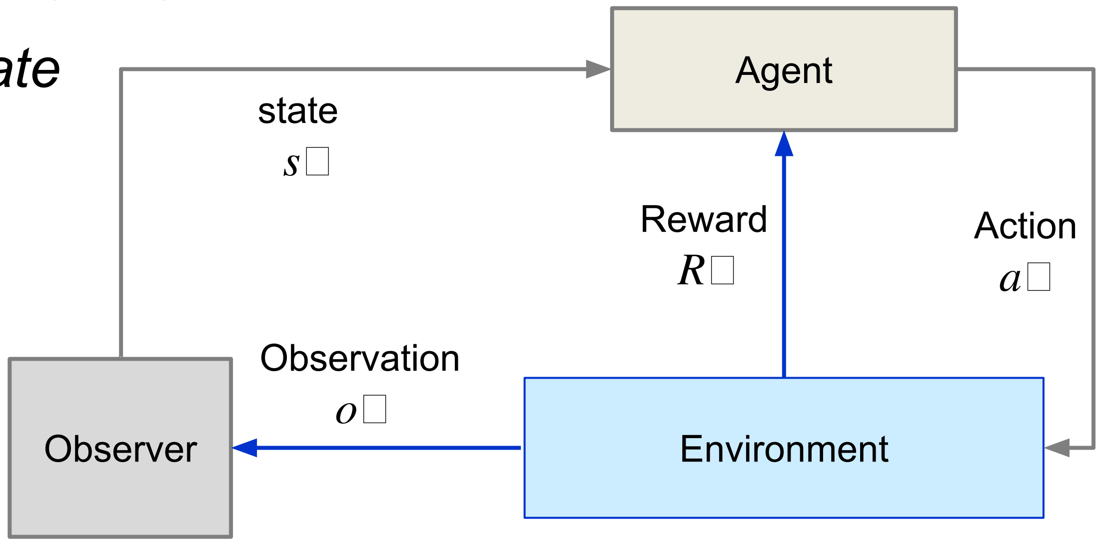


Observability

- *Full Observability:*
  - *Agent directly observes environment*
  - $O_t = S_t^a = S_t^e$
  - *Formally this is a MDP*



- *Partial Observability:*
  - Agent indirectly observes environment
  - *agent state ≠ environment state*
  - *Formally this is a POMDP*

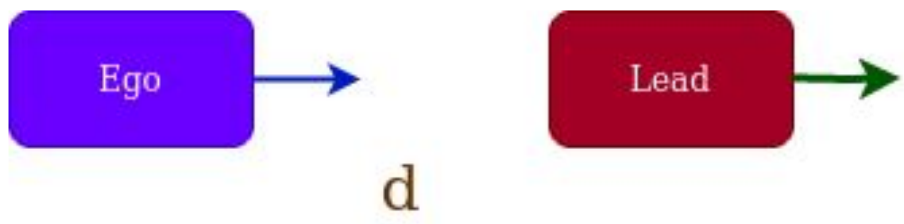


Elements of Reinforcement Learning

- Apart from the agent and the environment, there are four main sub-elements in RL system
  - *Policy*
  - *Reward Signal*
  - *Value function*
  - *Model of environment*

Policy

- Policy (π) $\rightarrow$ defines agents way of behaving at a given time
- Mapping from perceived states of the environment to actions to be taken in those states
- It is core of RL as it determines the behavior
- Policy may be stochastic specifying probabilities for each action



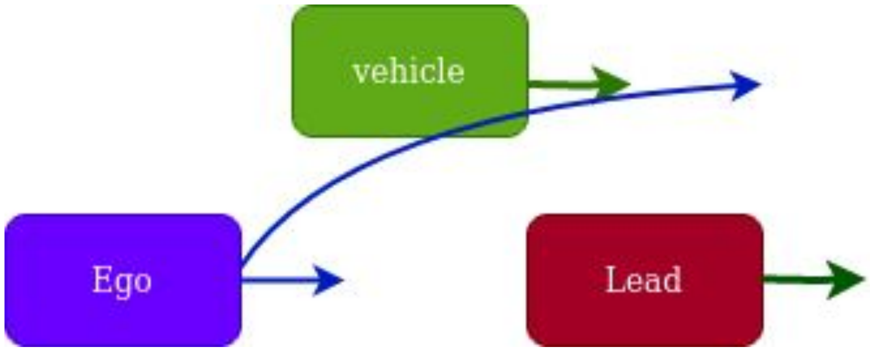
Policy:

If d <= brake safe distance:
  decelerate

else:
  maintain

Reward Signal

- Reward signal defines the goal of a RL problem
  - maximization problem
  - minimization problem
- It determines the effectiveness of an action(good or bad)
- Reward can be sparse or continuous
- Policy can be altered based on the basis of reward



Action: lane_change (bad)
reward: penalty because of collision

Action: follow (good)
reward: positive reward

Value function

- *Value function is a prediction of reward in long run*
- *Total amount of reward an agent can accumulate over the future starting from a state*
- $v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right]$
- $\gamma \rightarrow$ discount factor; $\gamma \in [0,1]$
- γ close to 0: Myopic (short-sighted) evaluation
- γ close to 1: Far-sighted evaluation
- *We seek actions that result in states of highest value*
- *Value is more important than reward as they determine long run policy*

Model of environment

- It mimics the behavior of environment
- Given a state and action, Model predicts
  - Next state
  - Next reward
- Model based RL algorithms:
  - Model of environment is used to predict the course of action in future states
  - Dynamics can be modeled or learned from experience
- Model free RL algorithms:
  - Operate in absence of complete knowledge of environment dynamics
  - Learn directly by experience or trial and error

Model based reinforcement-learning

- Used when dynamics of environment is not complex
- Advantages:
  - Dynamics is accurate and efficient
- Disadvantages:
  - Computationally intensive
- Algorithms:
  - Dynamic Programming: Solves Bellman equation iteratively
  - Monte Carlo Tree search (MCTS):
    - Search for action over the action sequences in the environment
    - Predict the outcome of action using the learned model

Model free reinforcement-learning

- Used when dynamics of environment are unknown or complex

- Advantages:
  - Less computationally intensive
  - Can learn directly from raw sensor data

- Disadvantages:
  - High variance
  - Slower convergence rate

- Algorithms:
  - Q-learning: uses a Q-Table that stores rewards for all state-action pairs
  - Deep-Q-Learning: uses neural networks to approximate Q-Table

Solution Methods for reinforcement-learning

- Policy-based methods: Learn a policy function directly
  - Deterministic Policy:
    - $a = \pi(s)$; will always return the same action given a state
  - Probabilistic Policy:
    - $\pi(a|s) = P[A|s]$; probability distribution over a set of actions given a state

- Value-based methods: Learns the value function mapping state to its value
  - $$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$$
  - The policy then will select states with highest value

Two types of value-based methods

State Value function

$$\underbrace{V_\pi(s)}_{\text{Value of state s}} = \underbrace{\mathbf{E}_{\boxed{\pi}}}_{\text{Expected return}}[G_t \,|\, \underbrace{S_t = s}_{\text{If the agent starts at state s}}]$$

$\boxed{\pi}$: And uses the policy to choose its actions for all time steps

For each state,
the state-value function outputs
the expected return
if the agent starts in that state
and then follows the policy forever after.

Action value function

$$\underbrace{Q_\pi(s,a)}_{\text{Value of state-action pair s,a}} = \underbrace{\mathbf{E}_{\boxed{\pi}}}_{\text{Expected return}}[G_t \,|\, \underbrace{S_t = s}_{\text{If the agent starts at state s}}, \underbrace{A_t = a}_{\text{and chooses action a}}]$$

$\boxed{\pi}$: And then uses the policy to choose its actions for all time steps

For each state and action,
the action-value function outputs
the expected return
if the agent starts in that state
and takes the action
and then follows the
policy forever after.

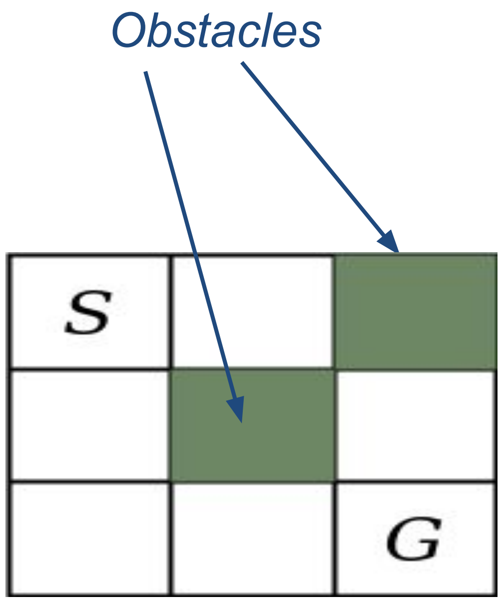
Link between value-function and policy:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Choose action with maximum value

Path Planning problem

- Robot starts at S
- Destination → G
- Objective:
  - Planning
  - Obstacle Avoidance
- Actions:
  - Up ↑
  - Down ↓
  - Left ←
  - Right →



0	1	2
3	4	5
6	7	8

State Indexing

Q-Learning

Step 1:

- Initialize Q arbitrarily, e.g Q = 0 for all states,actions
- discount factor = 0.99

Step 2:

- Choose action using epsilon greedy:
  - If epsilon is big, choose a random action
  - Otherwise, choose action with maximum value

Step 3:

- Perform action:
  - Observe Reward
  - Transition to next state

Q-Learning

Step 4:

- Update Q:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation
Former Q-value estimation
Learning Rate
Immediate Reward
Discounted Estimate optimal Q-value of next state
Former Q-value estimation
TD Target
TD Error

- Termination condition:
  - reached goal or reached 5 steps

Solution: Step 1

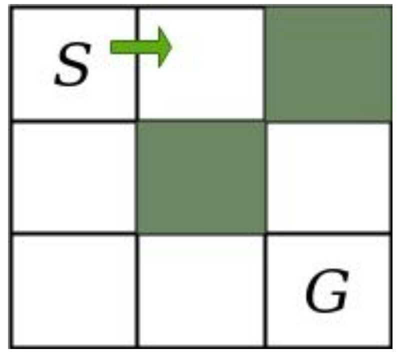
- Initialize Q-Table with 0
- Set rewards
  - -5 ; if obstacle
  - 10 ; if goal
  - 1 ; otherwise
- $\gamma = 1$
- $\alpha = 0.5$

Q-Table

	↑	↓	←	→
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

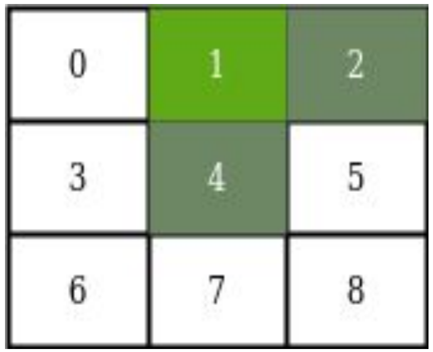
Solution: Step 2 & Step 3

- Choose action using ϵ-greedy
  - Initialize ϵ with 1
  - generate a random number, r in (0,1)
  - action:
    - random ; if r < ϵ
    - with maximum Q-value ; otherwise



random action: go right

- Perform action
  - reward = 1
  - next state → state 1



Solution: Step 4

- Update Q-Table using

$$\underbrace{Q(S_t, A_t)}_{\text{New Q-value estimation}} \leftarrow \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}} + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{\underbrace{\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}}}_{\text{TD Target}} - \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}}}_{\text{TD Error}}]$$

- Q(0,right) = 0 + 0.5(1 + 1 * 0 - 0) = 0.5

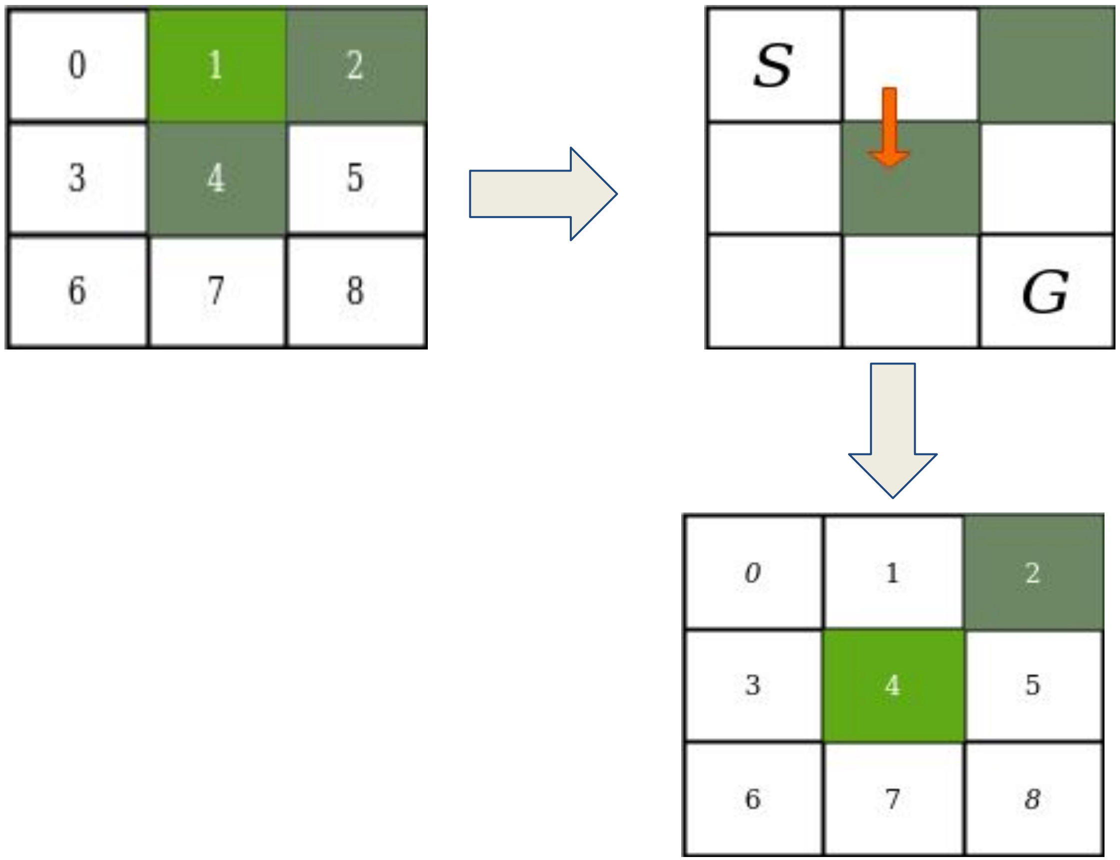
- Update epsilon using epsilon decay

- Check termination condition:
  - end episode ; if true
  - repeat the steps → otherwise

Q-Table

	↑	↓	←	→
0	0	0	0	0.5
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

Repeat in next state

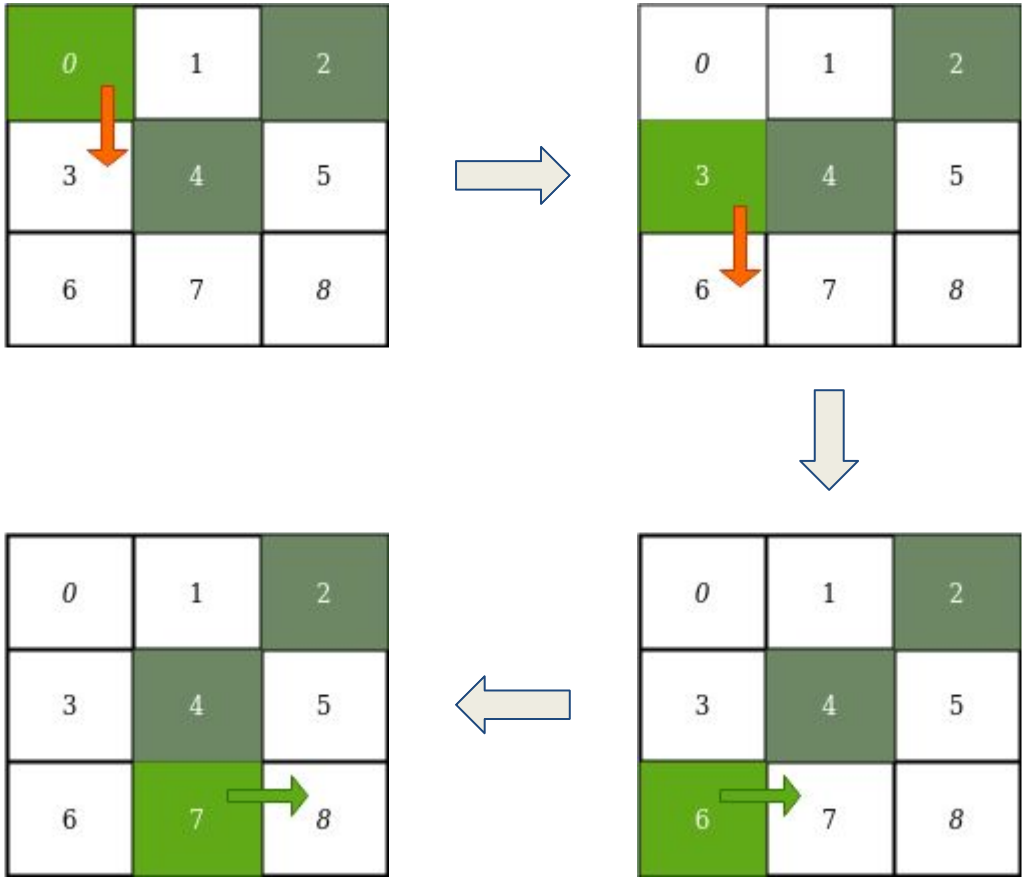


Q-Table

	↑	↓	←	→
0	0	0	0	0.1
1	0	-2.5	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

- $Q(1,down) = 0 + 0.5(-5 + 1 * 0 - 0) = -0.5$

Q-Table after 10 episodes



Optimal Policy

Q-Table

	↑	↓	←	→
0	0	7.3	0	3.6
1	0	-3.6	6.2	-4
2	0	0.94	4.4	0.9
3	4.9	8.3	-4	-0.9
4	3.2	5.4	5.7	0.75
5	-3	0	-3.8	0
6	0.9	0	0	8.2
7	0.22	0	2	9.4
8	0	0	0	0

Monte Carlo Approach

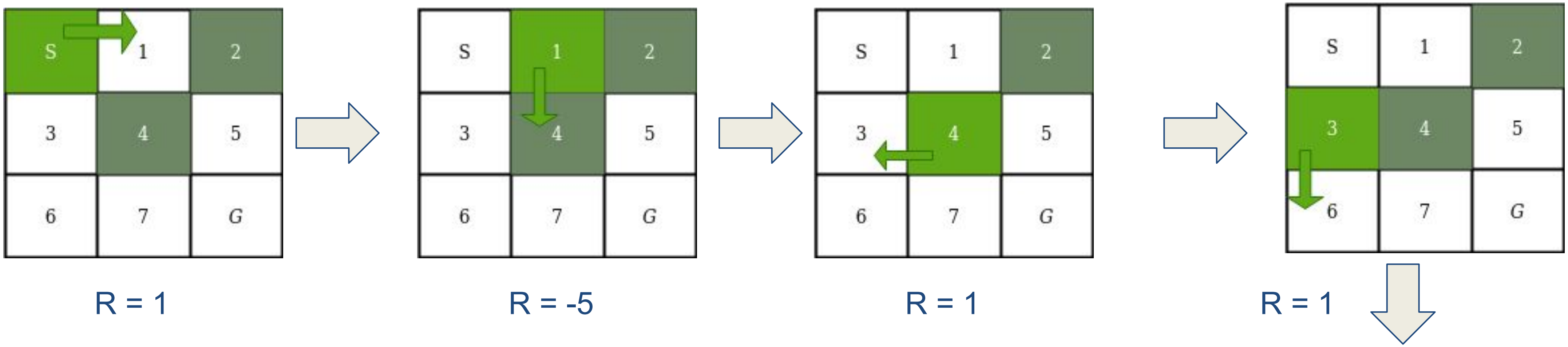
- Learning at the end of episode
- Take actions using epsilon-greedy until the episode ends
- Update the Value-function using:

$$\underbrace{V(S_t)}_{\text{New value of state t}} \leftarrow \underbrace{V(S_t)}_{\substack{\text{Former estimation}\\\text{of value of state t}\\\text{(= Expected return}\\\text{starting at that state)}}} + \underbrace{\alpha}_{\substack{\text{Learning}\\\text{Rate}}}[\underbrace{G_t}_{\substack{\text{Return at}\\\text{timestep}\\\text{t}}} - \underbrace{V(S_t)}_{\substack{\text{Former estimation}\\\text{of value of state t}\\\text{(= Expected return}\\\text{starting at that}\\\text{state)}}}]$$

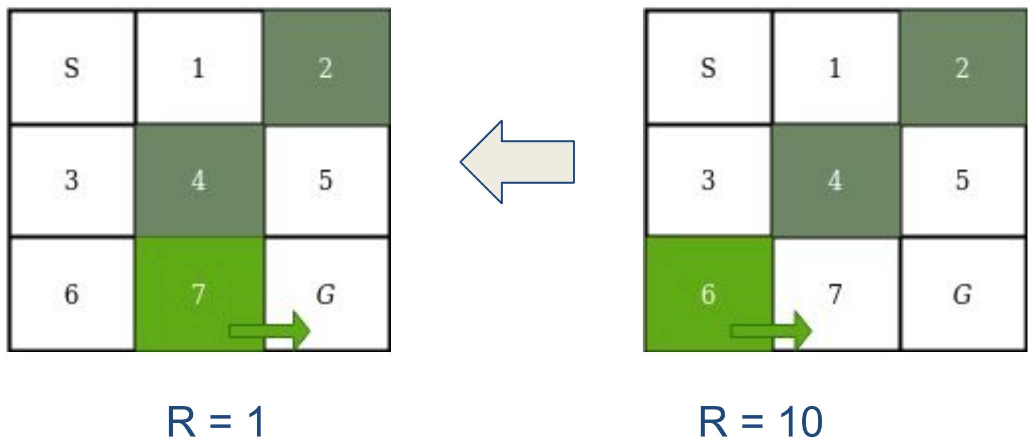
G_t is the cumulative reward

$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots$

Monte Carlo Approach Solution



- G_0 = 1 + -5 + 1 + 1 + 1 + 10 = 9
- $V(S_0) = V(S_0) + \alpha * [G_0 - V(S_0)]$
- $V(S_0) = 0 + 0.5 * [9 - 0] = 4.5$
- Similarly, V can be updated for all states
- Then, the state with maximum V will be selected for optimal policy



Deep-Q-Network (DQN)

- The Q-Table is approximated as a neural network

- Good for large state spaces

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation

Former Q-value estimation

Learning Rate

Immediate Reward

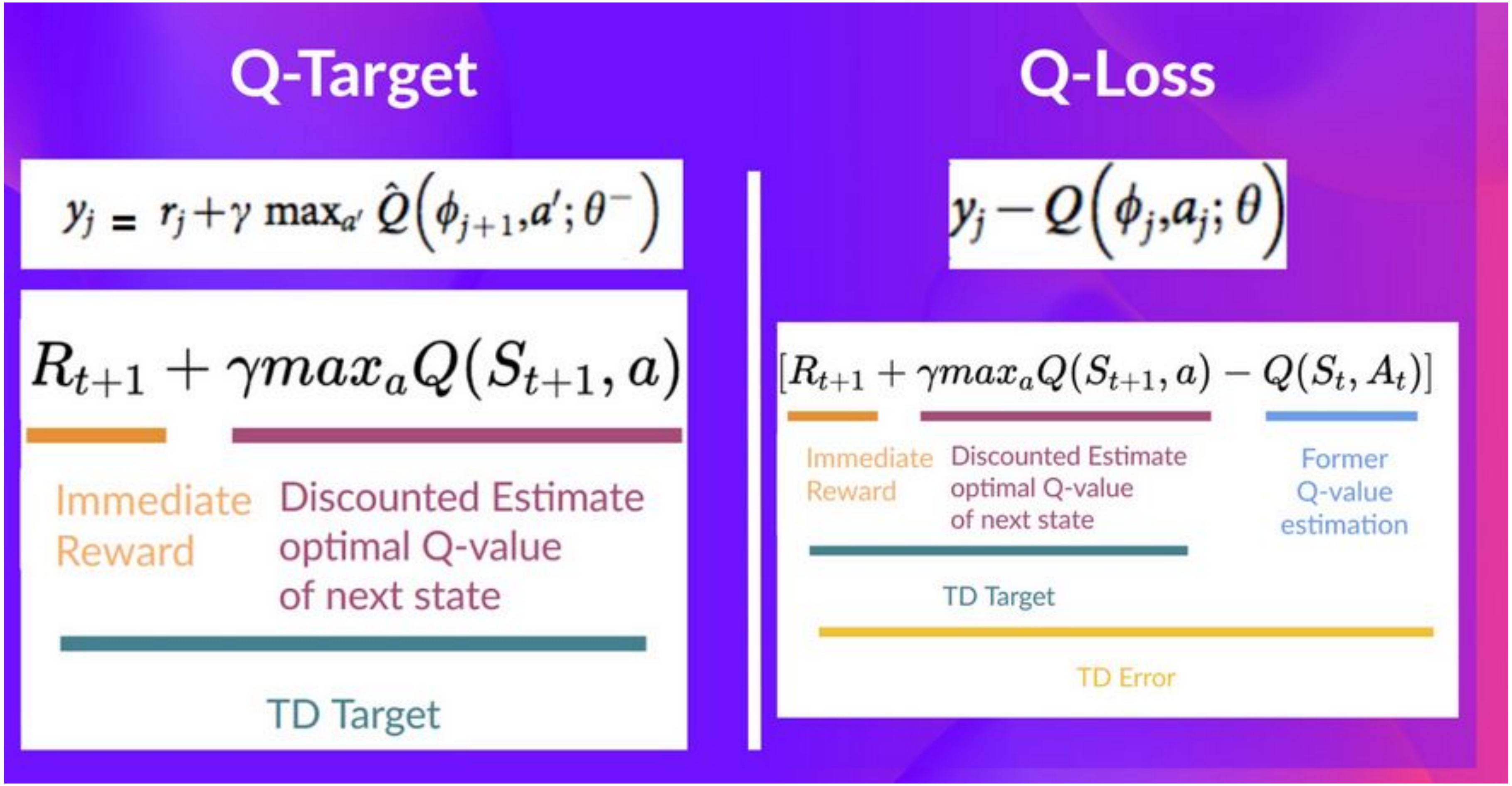
Discounted Estimate optimal Q-value of next state

Former Q-value estimation

TD Target

TD Error

Deep-Q-Network (DQN)



DQN Algorithm

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sampling

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$

Training

  End For
End For

Assignment

- Solve the path planning problem using DQN