Reinforcement Learning

CS576 MACHINE LEARNING

Dr. Jin S. Yoo, Professor
Department of Computer Science
Purdue University Fort Wayne

Reference

- Kelleher, Fundamentals of Machine Learning, Ch 11

Outline

- Reinforcement Learning – Big Idea
- Fundamentals
  - Intelligent agent approach
  - Fundamental components of Reinforcement Learning
  - Markov decision processes (MDPs)
  - Temporal-difference learning
- Standard Approach: Q-Learning, Off-Policy Temporal-Difference Learning

Reinforcement learning

- **Reinforcement learning (RL)** is a type of machine learning where an agent learns to make decisions by taking actions in an environment to maximize cumulative rewards.
- Unlike supervised learning, RL does not require labeled data (input/output pairs) and does not need to correct sub-optimal actions explicitly (in the way that incorrect predictions are corrected in supervised learning).
- Unsupervised learning uncovers hidden patterns from unlabeled data, whereas RL is concerned with finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

Reinforcement Learning - Big Idea

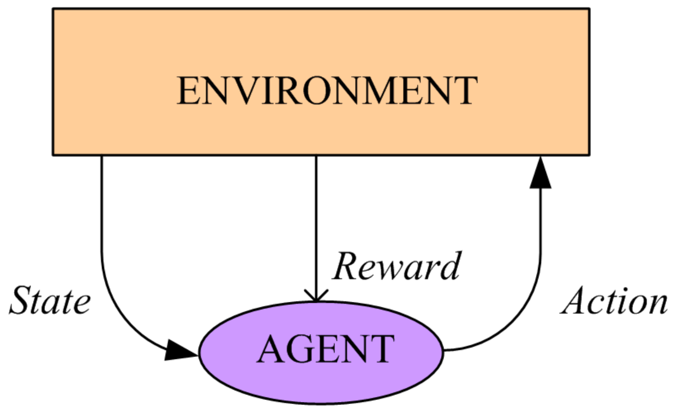
- Sarah, an aspiring venture scout, is working towards for her pioneering badge.
- A distinctive task in this pursuit involves mastering the skill of crossing a stream via stepping-stones, all while her vision is obscured by an electronic blindfold.
- The objective here is to get across the river with the minimum number of steps without getting wet.
- To aid in this endeavor, Sarah's blindfold briefly becomes clear for half a second before each move, granting her a momentary glimpse of the surroundings to strategize her next step's direction and distance.

Big Idea (cont.)

- Every decision that Sarah took resulted in instance feedback:
  - falling into the river brought negative feedback,
  - reaching the next stepping-stone yielded positive feedback,
  - retreating to a previous stone resulted in highly negative feedback, and
  - reaching the opposite riverbank generated highly positive feedback.
- Through refining her choices to enhance these immediate outcomes, Sarah mastered the task at hand.
- This process mirrors the goals of reinforcement learning

Introduction of Reinforcement Learning

- **Reinforcement learning** explores how an autonomous entity (, known as an agent) can learn to select the best actions within its environment to fulfill its objectives.
- In the context of reinforcement learning
  - The decision maker, or ***agent***, exists with a certain ***state*** in its environment, where it makes decisions or takes ***actions***, for which it may receive ***rewards***, leading to changes in its state
  - The **agent's goal** is to discover the most effective *sequence* of actions - a successful strategy or ***control policy*** – that resolves a challenge by maximizing the total rewards received over time (***maximum cumulative reward***).



Introduction of Reinforcement Learning

- Unlike supervised and unsupervised machine learning that depends on datasets for learning, **reinforcement learning** focuses on an agent's direct interaction with its environment by trying tasks multiple times and learning from the outcomes.
- Reinforcement learning is sometimes described as "learning with a critic" rather than "learning with a teacher", as seen in supervised learning.
  - In reinforcement learning, the "critic" provides feedback on the actions taken, not by dictating the actions but by indicating their effectiveness based on past performance.
  - This leads to the *credit assignment* problem, where determining which actions to attribute success or failure becomes a challenge.

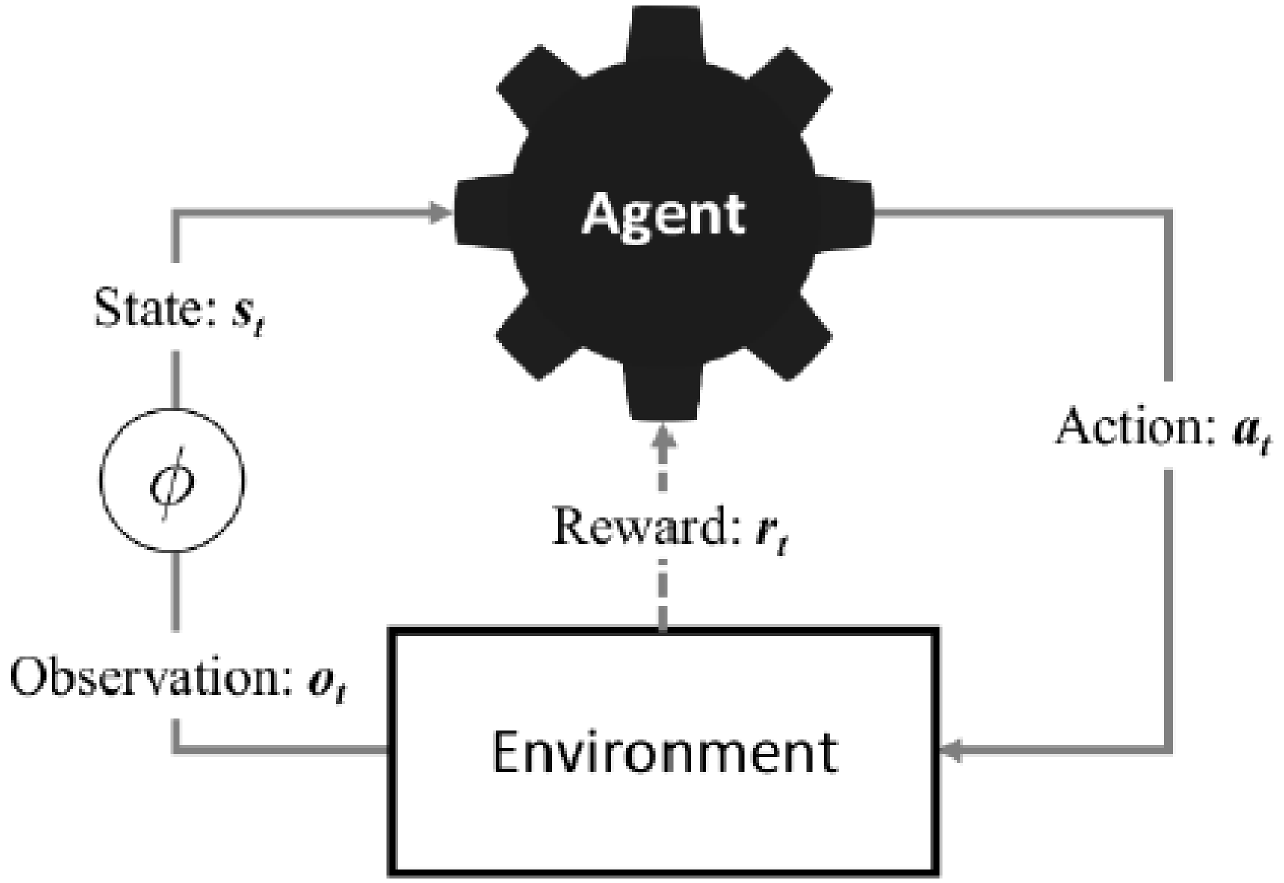
Applications

- Reinforcement learning is versatile, commonly used to guide the behavior of **autonomous systems** such as task-performing robots, automated game-playing agents, or systems managing factory operations.
- **Example**: Learning to play Backgammon [Tesauro, 1995]
  - The TD-Gammon program, designed to master Backgammon, improves through a sequence of strategic moves aimed at winning.
  - It rewards itself with +100 for a win, -100 for a loss, and 0 for any other outcomes.
  - By training through 1.5 million self-played games,
  - TD-Gammon has achieved a level of play that rivals the best human players.

Outline

- Reinforcement Learning – Big Idea
- ☞ **Fundamentals**
  - Intelligent Agent Approach
  - Fundamentals of Reinforcement Learning
  - Markov Decision Process (MDP)
  - Temporal-Difference Learning
- Standard Approach: Q-Learning, Off-Policy Temporal-Difference Learning

Intelligent Agent Approach



Intelligent Agent Approach

- In reinforcement learning, the learner or decision maker is known as the ***agent*** (or ***intelligent agent***) who endeavors to accomplish a task within a given setting.
- The agent engages with its ***environment*** which encompasses all external factors.
- **At any given moment, time *t*,** the agent
  - observes the current ***state*** of its environment;
  - based on these ***observations*** (denoted as o_t), decides on an ***action*** a_t;
  - following this, receives ***immediate feedback*** (often termed as ***rewards*** r_t), that indicates the quality of the action, especially informing the agent if the action taken was good or bad.

Reward

- Again, the primary objective of the agent is to complete the task as *successfully* as possible.
- We assume that the agent goals are encapsulated by a ***reward* function,** which assigns a numerical reward value $r_t \in \Re$ for immediate feedback corresponding to each specific action a_t taken in each unique state s_t at any given time t.
  - For example, when training an agent to play a game, the trainer might offer a positive reward (e.g., +100) for a win, a negative reward (e.g., -100) for a loss, and zero for any other outcomes .
- When the rewards are probabilistic rather than fixed – reflecting a level of unpredictability (uncertainty) in outcomes – , the expected reward is determined by the probability distribution $p(r_t|a_t, s_t)$, which calculates the likelihood of receiving a reward r_t given the action a_t in state s_t

Episode, History

- An attempt to carry out the mission is known as an ***episode*** or a ***trial***, which is a sequence of tuples at different time steps:

$$(o_1, a_1, r_1), (o_2, a_2, r_2), \ldots, (o_e, a_e, r_e).$$

  , in which the episode proceeds through time-steps $t = 1, \ldots, e$, where e is a time-step at which the agent reaches the desired goal state.

  - An episode refers to a complete sequence of the agent's interactions with the environment from the start to an ending point.

- ***History***, H_t, is a record of all observations, actions and rewards up to a particular time-step t, within an episode.
  - The history does not necessarily includes a full episode. It is the sequence of steps taken so far.

State

- At any given time-step *t*, the state s_t encapsulates:
  - all important information regarding the environment at that specific moment,
  - Vital details of the environment's previous states (at preceding time-steps, and
  - any essential information about the agent's internal state
- **Example**
  - In the case of a robot operating in a hospital to transport equipment to surgical rooms, the *state* would capture the robot's current location, the proximity of people, its objective at the moment (either collecting or delivering items), and the charge level of its batteries.

State-Action-Reward

- The data collected about the environment at any time-point t is transformed into a state, s_t, using a ***state generation function*** (also referred to as ***transition function***)
- When the environment can be **fully observable**, the transition function often acts as direct mapping, meaning the observation itself directly constitutes the state.
- However, in situations where the environment is **partially observable or more complex**, this function may integrate observations from multiple time-steps to construct the state.
- Therefore, rather than using raw observations o, we represent an **episode** using states s, such as a sequence of **state-action-reward** triples:

$$(s_1, a_1, r_1), (s_2, a_2, r_2), \dots, (s_e, a_e, r_e)$$

Assumptions for Reinforcement Learning

- **We assume that**
  - A finite set *S* of distinct states of agent's environment
  - A finite set *A* of actions that it can perform.
  - At each discrete time step *t*, the agent
    - detects the current state $s_t \in S$,
    - chooses an applicable action $a_t \in A$, and
    - performs it.
  - The environment then reacts by:
    - giving the agent a reward based on the current state and action: $r_t = r(s_t, a_t)$, and
    - generating the subsequent state s_{t+1} using the function $\emptyset(s_t, a_t)$: $s_{t+1} = \emptyset(s_t, a_t)$, where $\emptyset$ is a state generation function.

Assumptions (cont.)

- **We further assume that**
  - The reward r_t and the secured state s_{t+1} are solely influenced by the *current* state and the chosen action, independent of past states or actions.
  - The state transition function ∅ and the reward function r could be nondeterministic (- probabilistic and unpredictable).
  - The agent may not have prior knowledge of the state generation function ∅ and the reward function r.

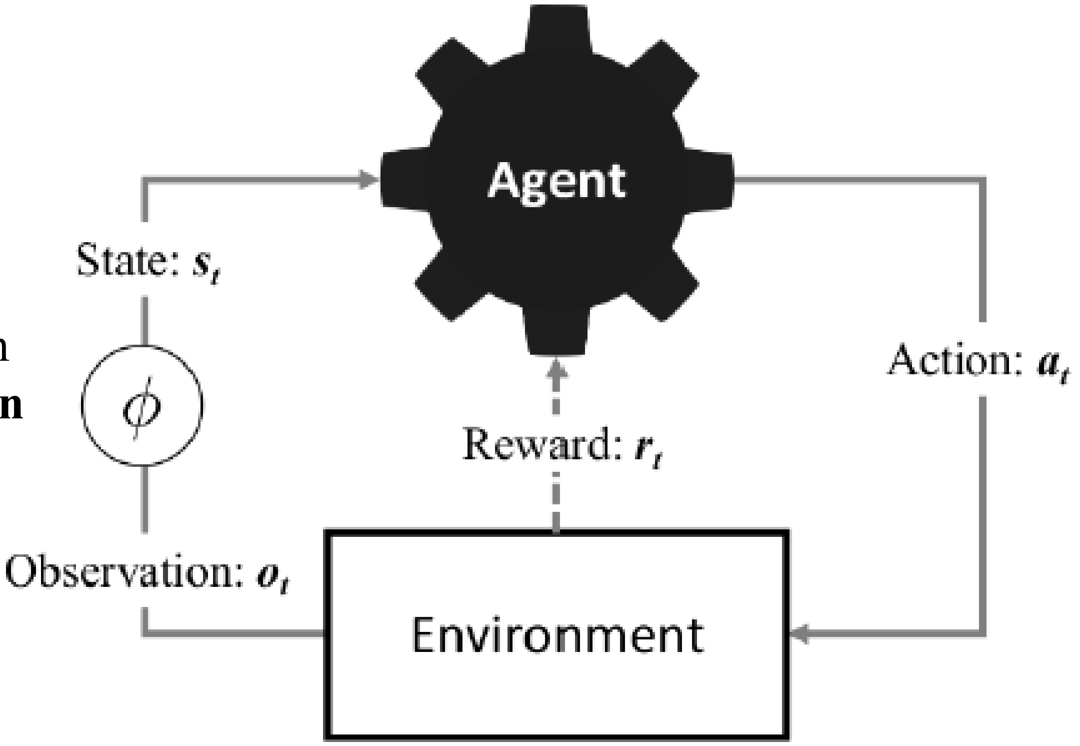
Summary: Intelligent Agent Approach

Episode = $(\boldsymbol{o_1}, \boldsymbol{a_1}, \boldsymbol{r_1}), (\boldsymbol{o_2}, \boldsymbol{a_2}, \boldsymbol{r_2}), \ldots, (\boldsymbol{o_e}, \boldsymbol{a_e}, \boldsymbol{r_e})$

If the environment is **fully observable**, $\boldsymbol{o_t} = \boldsymbol{s_t}$,

Episode = $(\boldsymbol{s_1}, \boldsymbol{a_1}, \boldsymbol{r_1}), (\boldsymbol{s_2}, \boldsymbol{a_2}, \boldsymbol{r_2}), \ldots, (\boldsymbol{s_e}, \boldsymbol{a_e}, \boldsymbol{r_e})$

ϕ is a state generation function (or **transition function**) which generates a state



Outline

- Reinforcement Learning – Big Idea
- Fundamentals
  - Intelligent Agent Approach
  - ☞ **Fundamentals of Reinforcement Learning**
  - Markov Decision Process (MDP)
  - Temporal-Difference Learning
- Standard Approach: Q-Learning, Off-Policy Temporal-Difference Learning

Cumulative Reward

- The goal of an agent is to complete a task as *successfully* as possible.

  What does it mean to successfully complete a task?

- In RL, the degree to which an agent has achieved a goal is measured solely by the *cumulative reward* through its actions aimed at that goal.
- The **cumulative reward** during an episode is also known as the **return** and is calculated as

$$G = r_t + r_{t+1} + r_{t+2} + \cdots + r_e$$

  where e is a time-step at which the agent reaches the goal.

- The primary objective of an intelligent agent is **to maximize the cumulative reward**.

Policy

- A ***control policy*** (or simply ***policy***) informs the agent which action a_t, should be selected in any particular state s_t.
  - It is essentially a function that maps states (S) to actions (A), $\pi: S \rightarrow A$.
- The policy can be described by the equation:

$$\boldsymbol{a_t = \pi(s_t)}$$

- Policies can also be described in terms of probabilities:

$$\boldsymbol{P(A_t = a|S_t = s) = \pi(S_t = s)}$$

, where A_t and S_t are random variables representing the action and state at time t, respectively.

The policy π, returns a probability distribution over all possible actions available to the agent in state s

Policy Type

There are many action-selection policies:

- **Greedy policy** (Greedy action selection policy)
  - By the only goal of maximizing return, the agent should always take the action that will give it the highest immediate reward.
- **Random policy**
- ϵ**-greedy policy**
- **Off-policy**
- **On-policy**
- etc.

Value Function

- The **value function** corresponding to a policy π, $\boldsymbol{V_\pi(s_t)}$, provides the expected return (or expected cumulative reward) anticipated if the agent consistently follows a policy π, starting from state s_t at any given time-step t until the end of an episode.

$$V_\pi(s_t) = \boldsymbol{E}_\pi[r_t + r_{t+1} + \ \dots + r_e | s_t]$$

, where the sequence of rewards r_{t+i} is generated by beginning at state s_i and repeatedly selecting actions according to policy π such as $a_t = \pi(s_t)$, $a_{t+1} = \pi(s_{t+1})$, $a_{t+2} = \pi(s_{t+2})$, etc.

- A value function estimates how good it is for an agent to be in a given state.

Action-Value Function (Q-function)

- The action-value function represents the expected return of taking a action in a state and then following a certain policy.
- The **action-value function** (or **Q-function**) of a policy π, $Q_\pi(s_t, a_t)$, outputs the expected cumulative reward anticipated if the agent takes action a_t in state s_t at time-step *t* and then follows policy π through to the end of an episode.

$$Q_\pi(s_t, a_t) = \boldsymbol{E}_\pi[r_t + r_{t+1} + \ \ldots + r_e | s_t, a_t]$$

, where r_{t+i} are the rewards received at future time steps, continuing from state s_i and action a_t, in adherence to policy π

- It is used to evaluate the quality of an action in a particular.

Value-Function vs Q-Function

- **Value Function:**
  - A value function $\boldsymbol{V}(s)$ estimates the expected return (cumulative future rewards) of being in a state *s* and following a particular policy π thereafter. It does not consider what the initial action taken from that state is; rather, it assumes the agent will follow the policy π.
  - The value function answers the question: "What is the expected reward if I am in a given state and follow the optimal policy?"
- **Q-Function** (or **Value-Action Function**):
  - A Q-function $\boldsymbol{Q}(s, a)$ estimates the expected return of taking an action *a* in a state *s*, and then following a policy π until the end of an episode. It provides a value for each action-state pair, thereby directly informing the choice of action.
  - The Q-function answers the question: "What is the expected reward if I take a certain action in a given state, and then follow the optimal policy?"

Discounted Return

- [Reminder] The **cumulative reward** (or **return**) during an episode is G=$r_t + r_{t+1} + r_{t+2} + \cdots + r_e$
- **To prioritize immediate rewards**, we can apply a ***discount rate*** ($0 \leq \gamma < 1$)
- The total future rewards adjusted by this discount rate are known as the **discounted return**:

$$G_\gamma = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{e-1} r_e$$

  - A reward received *i* time-steps into the future is exponentially discounted by γ^i , **reducing the impact of future rewards on the current value estimation for an action.**
  - If γ=0, only the immediate reward are considered.
  - As γ approaches 1, future rewards become increasingly significant.

Q-function with Discount Rate

- The original Q-function is $Q_\pi(s_t, a_t) = \boldsymbol{E}_\pi[r_t + r_{t+1} + \ldots + r_e | s_t, a_t]$
- The **Q-function** (or action-value function) **with discount rate** outputs the expected discounted return

$$Q_\pi(s_t, a_t) = E_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{e-1} r_e | s_t, a_t]$$

Outline

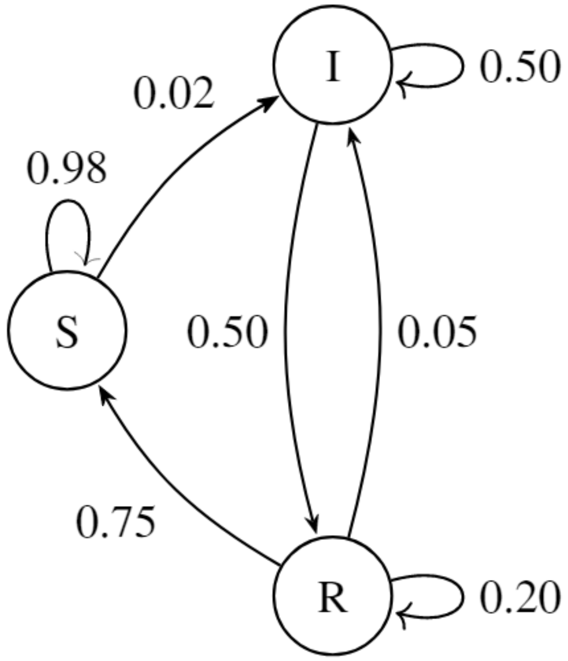
- Reinforcement Learning – Big Idea
- Fundamentals
  - Intelligent agent approach
  - Fundamental components of Reinforcement Learning
  - ☞ **Markov decision processes (MDPs)**
  - Temporal-difference learning
- Standard Approach: Q-Learning, Off-Policy Temporal-Difference Learning

Markov Decision Process and Markov Process

- A **Markov Decision Process** (MDP) is a mathematical framework used to describe an environment in reinforcement learning.
- The MDP builds upon the foundation of **Markov Processes.**

Markov Process

- A **Markov process** provides a foundational framework for modeling discrete random processes without decision-making elements, transitioning through a defined set of states, *S*.
- **Example**: Consider the application of Markov process in epidemiological modeling during an outbreak:
  - The set of states, *S* ={**S**usceptible, **I**nfected, **R**ecovered)
  - An individual is categorized into one of these states at any given time and transitions between states according to the probabilities defined by the Markov process.



S-I-R Markov process

Markov Property

- Markov processes are built on the **Markov property** (or **Markov assumption**) that the likelihood of transitioning to a particular state at the next time-step relies solely on the present state, independently of the state sequence preceding it.

$$P(S_{t+1}|S_t, S_{t-1}, S_{t-2}, \dots) = \boldsymbol{P(S_{t+1}|S_t)}$$

- **Under the Markov property**, the state **transition probability** between two successive states is determined as

$$P(s_1 \rightarrow s_2) = P(S_{t+1} = s_2 | S_t = s_1)$$

, where S_t and S_{t+1} represent the stochastic state variables s at time t and $t+1$, respectively.

Modeling Markov Processes with Transition Matrix

- The entirety of a Markov process's behavior can be captured within a **transition matrix**:

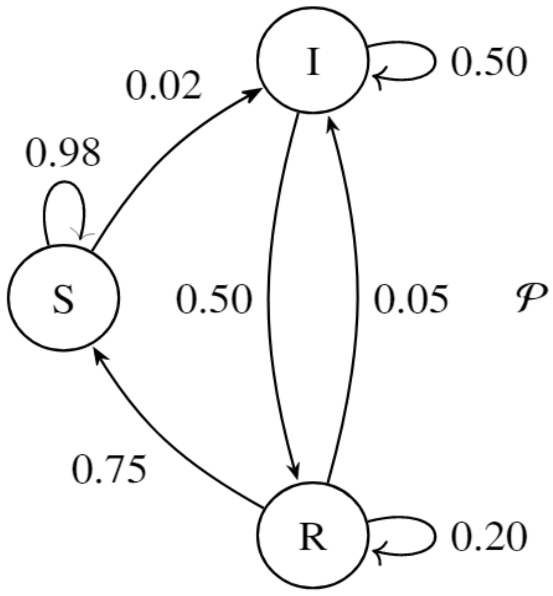
$$\mathcal{P} = \begin{bmatrix} P(s_1 \to s_1) & P(s_1 \to s_2) & \dots & P(s_1 \to s_n) \\ P(s_2 \to s_1) & P(s_2 \to s_2) & \dots & P(s_2 \to s_n) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_n \to s_1) & P(s_n \to s_2) & \dots & P(s_n \to s_n) \end{bmatrix}$$

, where states s_1 through s_n represent the distinct possible states within the process.

- A Markov process is fully described by its state space S and its transition matrix $\mathcal{P}$, which together describes the probabilities of transitioning from one state to any other state.

Example: Epidemic Modeling

- A Markov process to model how infection process in an individual when a disease epidemic breaks out.
  - The set of states, S ={**S**usceptible, **I**nfected, **R**ecovered)



(a) S-I-R Markov process

$$\mathcal{P} = \begin{array}{c} \\ \text{S} \\ \text{I} \\ \text{R} \end{array} \begin{array}{c} \begin{array}{ccc} \text{S} & \text{I} & \text{R} \end{array} \\ \begin{bmatrix} 0.98 & 0.02 & 0.00 \\ 0.00 & 0.50 & 0.50 \\ 0.75 & 0.05 & 0.20 \end{bmatrix} \end{array}$$

(b) S-I-R transition matrix

- S-I-R transition matrix
  - Most people remain in the Susceptible state indefinitely, $P(S \rightarrow S) = 0.98$
  - But , with a small probability, $P(S \rightarrow I) = 0.02$, can transition to the Infected state.
  - Individual will most likely remain in the Infected state for some time, $P(I \rightarrow I) = 0.50$, but will transition eventually to the Recovered state, $P(I \rightarrow R) = 0.50$.
  - …

Markov Decision Process (MDP)

- **Markov Decision Processes** (MDPs) build upon the foundation of **Markov processes** by integrating decision-making capabilities and the concepts of rewards.
  - The **Markov property** implies that the future state depends only on the current state and action, not on the sequence of events that preceded it.
- MDP consists of a set of states, a set of actions, a transition function that predicts the next state given a current state and action, and a reward function.

Markov Decision Process

- [Reminder] In Markov process, the state transition probability between two successive states is as

$$P(s_1 \rightarrow s_2) = P(S_{t+1} = s_2 | S_t = s_1)$$

- MDPs expand the Markov process model to include a set of actions, incorporating these actions into the calculation of state transition probabilities
- The **transition probability** in MDPs is determined as

$$P(s_1 \xrightarrow{a} s_2) = P(S_{t+1} = s_2 | S_t = s_1, A_t = a)$$

  - Adhering to the **Markov property**, the likelihood to moving to a specific state is determined solely by the present state and the action executed at that moment.
- Moreover, each state transition prompted by an action is associated with the corresponding **reward**:

$$R(s_1 \xrightarrow{a} s_2) = E(r_t | S_t = s_1, S_{t+1} = s_2, A_t = a)$$

Q Function with MDP

- Original Q function (Action-value function)

$$Q_\pi(s_t, a_t) = \boldsymbol{E_\pi\left[\sum_{k=0}^{\infty} \gamma^k \, r_{t+k} \,|s_t, a_t\right]} = E_\pi[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \,|s_t, a_t]$$

- **Q-function with *MDP components***
  - When incorporating MDP components, the action-value function is adjusted to include the transition probabilities and rewards for each action, followed by the expected return from the next state onwards, again according to policy $\boldsymbol{\pi}$**.**

$$Q_\pi(s_t, a_t)$$

$$= \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1}) \left[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma E_\pi[\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \,|s_{t+1}]\right]$$

$$= \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1})[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \sum_{a_{t+1}} \boldsymbol{E_\pi[\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \,|s_{t+1}, a_{t+1}]}]$$

The recursive relations form:

$$Q_\pi(s_t, a_t) = \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1})[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \underbrace{\boldsymbol{Q_\pi(s_{t+1}, a_{t+1})}}_{\text{Nice recursive definition}}]$$

Action-Value Function with MDP

$$Q_\pi(s_t, a_t) = \sum_{s_{t+1}} P\left(s_t \xrightarrow{a_t} s_{t+1}\right)\left[R\left(s_t \xrightarrow{a_t} s_{t+1}\right) + \gamma \boldsymbol{Q_\pi}(\boldsymbol{s_{t+1}}, \boldsymbol{a_{t+1}})\right]$$

- This recursive relationship forms the basis for many dynamic programming and reinforcement learning algorithms, which seek to find the optimal policy that maximizes the action-value function.

Bellman Equation

$$Q_\pi(s_t, a_t) = \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1})[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \sum_{a_{t+1}} E_\pi[\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \, | s_{t+1}, a_{t+1}]]$$
$$= \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1})[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \boldsymbol{Q_\pi(s_{t+1}, a_{t+1})}]$$

- Further, we define the policy π as a function that returns a probability distribution over potential actions from a state.
- The Q-function of the next state, $Q_\pi(s_{t+1}, a_{t+1})$ is weighted by the policy π. It provides the probability of taking each action a_{t+1} in the next state, considering the likelihood of taking each action as specified by the policy.

$$\boldsymbol{Q_\pi(s_t, a_t)}$$
$$\boldsymbol{= \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1})[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \sum_{a_{t+1}} \pi(s_{t+1}, a_{t+1}) Q_\pi(s_{t+1}, a_{t+1})]}$$

, where $\boldsymbol{\pi(s_{t+1}, a_{t+1})}$ represents the policy's probability of choosing action a_{t+1} in state s_{t+1}, and $Q_\pi(s_{t+1}, a_{t+1})$ recursively applies the action-value function for the subsequent state and action. .

- The final formulation of the action-value function is known a **Bellman equation,** which is the fundamental in reinforcement learning.

Bellman Optimality Equation

- For any Markov Decision Process (MDP), there exists an optimal policy, π_*, which guarantees the highest possible returns from any state, equal to or better than those obtainable by any other policy.
- The **Bellman optimality equation** for the action-value function, which finds the optimal value of taking an action a_t in state a_t, is given by:

$$\boldsymbol{Q_*(s_t, a_t)}$$
$$= \sum_{s_{t+1}} P\left(s_t \xrightarrow{a_t} s_{t+1}\right) \left[R\left(s_t \xrightarrow{a_t} s_{t+1}\right) + \gamma \max_{a_{t+1}} Q_*(s_{t+1}, a_{t+1}) \right]$$

  - The sum includes the immediate reward for the transition $R\left(s_t \xrightarrow{a_t} s_{t+1}\right)$ plus the discounted maximum return of the subsequent state-action pair, optimized over all possible actions a_{t+1}
  - This equations states the maximum cumulative return of taking an action a_t in state s_t, and thereafter following the optimal policy π_*, is achieved by summing over all possible next states s_{t+1}.

Bellman Equation (Bellman Optimality Equation)

- The Bellman equation is a recursive relationship that provides a way to calculate the value of a state under a certain policy.
- The **Bellman Equation** is a theoretical foundation, an equation that explains how to decompose the value function into immediate rewards and future values.
- It states that the value of a state is equal to the immediate reward from an action taken in that state plus the discounted value of the subsequent state.
- The **Bellman Optimality Equation** is a special case where the policy being evaluated is the optimal policy.
- For Q-values, it expresses the relationship that the Q-value for a state-action pair is equal to the immediate reward plus the discounted maximum Q-value of the next state.

Outline

- Reinforcement Learning – Big Idea
- Fundamentals
  - Intelligent agent approach
  - Fundamental components of Reinforcement Learning
  - Markov decision processes (MDPs)
  - ☞ **Temporal-difference learning**
- Standard Approach: Q-Learning, Off-Policy Temporal-Difference Learning

Temporal-Difference Learning

- **Temporal-difference learning** is one of the most important of iterative approaches for developing approximate solutions
- It employs **an action-value table** (or **Q-table**) to store estimates of the anticipated returns (i.e., cumulative rewards) from executing each possible action a_t within every possible state s_t, thereby calculating the value of $Q_\pi(s_t, a_t)$
- Initially, all entries in the action-value table are assigned random values or set of zeros.
- Temporal-difference learning **aims** to find the true values for the table entries.
- This is achieved by engaging the agent within its the environment and **iteratively refining the table's values based on the agent's observed performance.**

Example: Action-Value Table (Q-Table)

State	Action	Value
PL-DL	*Twist*	0.039
PL-DL	*Stick*	−0.623
PM-DL	*Twist*	−0.597
PM-DL	*Stick*	−0.574
BUST	*Twist*	0.000
BUST	*Stick*	0.000
LOSE	*Twist*	0.000
LOSE	*Stick*	0.000
PH-DL	*Twist*	−0.666
PH-DL	*Stick*	0.940
PL-DH	*Twist*	−0.159
PL-DH	*Stick*	−0.379
TIE	*Twist*	0.000
TIE	*Stick*	0.000
PM-DH	*Twist*	−0.668
PM-DH	*Stick*	−0.852
PH-DH	*Twist*	−0.883
PH-DH	*Stick*	0.391
WIN	*Twist*	0.000
WIN	*Stick*	0.000
TWENTYTWO	*Twist*	0.000
TWENTYTWO	*Stick*	0.000

An entry for each action-state combination

The terminal states always have a value of 0.0000 for every action

Update Rule in Temporal-Difference Learning

1. First, each entry in the action-value table is initially assigned a random value or 0.
2. When an agent engages in action a_t from a given state s_t, it computes the **difference** or **'temporal difference'** between the estimated return $Q_\pi(s_t, a_t)$ from the table for the state-action pair and the actual return $G(s_t, a_t)$ obtained post-action.
3. If the current estimate is higher than the actual return, then the estimated value is marginally decreased, and vice versa.
   - The adjustment is made according to the formula:

     $$\boldsymbol{Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(\underbrace{G(s_t, a_t) - Q_\pi(s_t, a_t)}_{\text{Difference between actual and expected returns}})}$$

   - Here, a **learning rate α**, is used to control the magnitude of modification applied to the action-value estimate after each iteration.
4. Through continuous application of this update mechanism, the value within the action-value table gradually approximate the true values, reflecting more accurate estimates of the expected returns.

Update Rule (cont.)

- In fact, it is not possible to know the actual return $G(s_t, a_t)$ that will be earned by the agent across the entire episode until the episode is complete.

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(\boldsymbol{G(s_t, a_t)} - Q_\pi(s_t, a_t))$$

- To address the challenge, temporal-difference learning employs a technique called **bootstrapping**
- This method uses the **existing estimates of expected returns in the action-value table** for updates during the episode.
- The action-value update equation incorporating bootstrapping is expressed as:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(\underbrace{\boldsymbol{r_t + \gamma Q_\pi(s_{t+1}, a_{t+1})}}_{\text{Estimated actual return}} - \underbrace{Q_\pi(s_t, a_t)}_{\text{Expected return}})$$

TD(0): Simple Temporal-Difference Learning

- This basic form of temporal-difference learning, known as **TD(0)**, updates the action-value table after every action is taken by the agent.

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(r_t + \gamma Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t))$$

- The **key characteristics of TD(0)** include:
  - Immediate learning from each action's outcome, facilitating a quick adjustment to the estimates of the action-value function.
  - Potential slow convergence to the optimal action values, as it relies solely on the next step and does not consider the entire return until the end of the episode.

Limitation of Greedy Policy

- The **greedy action selection policy** based on greediness tends to exploit what is currently known to yield the best rewards.
- **Example**: A person who likes chicken and what item has chicken (e.g., *sicín* in Irish) in the menu, the person might always choose it over other, less familiar options on the menu.
- However, strictly adhering to such a policy might lead to potentially suboptional outcomes, as it precludes the chance to discover other rewarding choices.

Exploration and Exploitation in Learning

- To facilitate learning, it is crucial to incorporate both **exploration** of new possibilities and **exploitation** of known rewards.
  - A policy that chooses actions randomly can maximize exploration, though it may result in inconsistent rewards, sometimes high, sometimes low.
- The **ϵ-greedy action selection policy** is one such approach, where there's a set probability ϵ of choosing an action at random, thus allowing for exploration, while the remaining probability $(1 - \epsilon)$ is used to exploit the best-known action.

Behavior Policy vs Target Policy

- It's important to differentiate between two types of policies: behavior policy and target policy
- A **behavior policy** which the agent uses to interact with the environment (often exploratory).
  - This policy usually allows for exploration, helping the agent to gather information about the environment and the possible rewards of different actions.
- A **target policy** which the algorithm is trying to learn (the optimal policy).
  - Ideally, this policy is optimized based on the knowledge acquired during the learning phase and is primarily focused on exploitation to maximize rewards.

Outline

- Reinforcement Learning – Big Idea
- Fundamentals
  - Intelligent agent approach
  - Fundamental components of Reinforcement Learning
  - Markov decision processes (MDPs)
  - Temporal-difference learning
- ☞ **Standard Approach: Q-Learning, Off-Policy Temporal-Difference Learning**

Q-Learning

- **Q-learning** is a model-free reinforcement learning algorithm that seeks to find the best action to take in each state.
- It uses the Q-function to store the expected returns for state-action pairs.
- Q-learning updates the Q-values using the Temporal Difference (TD) method.

Q-Learning Algorithm for Off-Policy TD Learning

```
Require: a behavior policy, $\pi$, that chooses actions
Require: an action-value function $Q$ that performs a lookup into an
    action-value table with entries for every possible action, $a$, and state, $s$
Require: a learning rate, $\alpha$, a discount-rate, $\gamma$, and a number of episodes to
    perform
 1: initialize all entries in the action-value table to random values (except for
    terminal states which receive a value of 0)
 2: for each episode do
 3:     reset $s_t$ to the initial agent state
 4:     repeat
 5:         select an action, $a_t$, based on policy, $\pi$, current state, $s_t$, and
    action-value function, $Q$
 6:         take action $a_t$ observing reward, $r_t$, and new state $s_{t+1}$
 7:         update the record in the action-value table for the action, $a_t$, just
    taken in the last state, $s_t$, using:
            $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left( r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$
 8:         let $s_t = s_{t+1}$
 9:     until agent reaches a terminal state
10: end for
```

Q-Learning Approach to Temporal-Difference Learning

- In Q-learning, the value of taking a specific action in a specific state is captured by the Q-value, which is updated as follows (Line 7) :
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \; - Q_\pi(s_t, a_t))$
  Which is, for examining the actions with the maximum expected return, a slightly modified version of $Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(r_t + \gamma Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t))$
  - The Q-value should increase when the action taken results in a positive immediate reward and leads to a state that offers a high potential for future rewards.
  - Conversely, the Q-value should decrease if the action results in a negative immediate reward or transitions to a state with lower future reward potential.
- The update rule reflects the core principle of Q-learning: to learn the optimal policy by maximizing the expected value of the total reward over all successive steps, starting from the current state.

Q-Learning of Off-Policy TD Learning

- Q-learning is categorized to as **off-policy** learning since the learning process evaluates a greedy policy - the policy that always selects the action with the highest estimated value – but does not necessarily follow it during learning.
- Instead, the agent often employs an ϵ**-greedy policy** for action selection to ensure a balance of exploration (trying new actions) and exploitation (using the known best actions).

Q-Learning of Off-Policy TD Learning

- Q-learning is **an off-policy algorithm** that learns the value of the optimal policy <u>independently of</u> the agent's actions.
  - It updates its Q-values using the maximum reward attainable from the next state—this is the "best future action" part—regardless of the policy the agent is currently following to select actions. So, the learning process is separate from the agent's action-selection policy.
    - In Q-learning, the Q-value update step considers the best possible action that could be taken in the next state, rather than the action that was actually taken according to the agent's exploration strategy.
  - This is done by using the max function over Q-values in the Bellman equation, which selects the highest Q-value for the next state across all possible actions.
- The Q-learning algorithm learns the optimal policy even if the agent does not always choose the optimal action while it is exploring the environment.

Example

- Imagine an agent in a maze where the goal is to find the shortest path to the treasure.
- The agent, guided by Q-learning, might take random turns to learn about the environment (exploration).
- However, the updates to the Q-values always assume that, from any given state, the agent would take the best known path towards the treasure, even if that's not the action the agent actually takes during exploration.
- Over time, as the agent explores various paths, the Q-values converge towards those representing the shortest paths to the treasure, thus learning the optimal policy.

A Worked Example: Grid Worlds

- **Grid worlds** are a classic representation for demonstrating reinforcement learning algorithms.
- These are simplified, discrete environments where an agent's task is to find an optimal path from a designated starting point to a target destination, negotiating any potential barriers in its path.

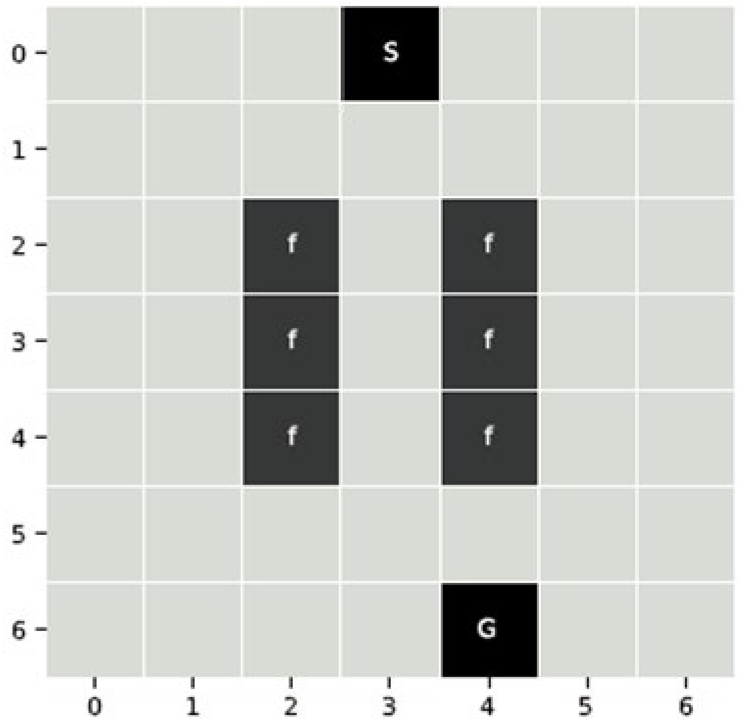
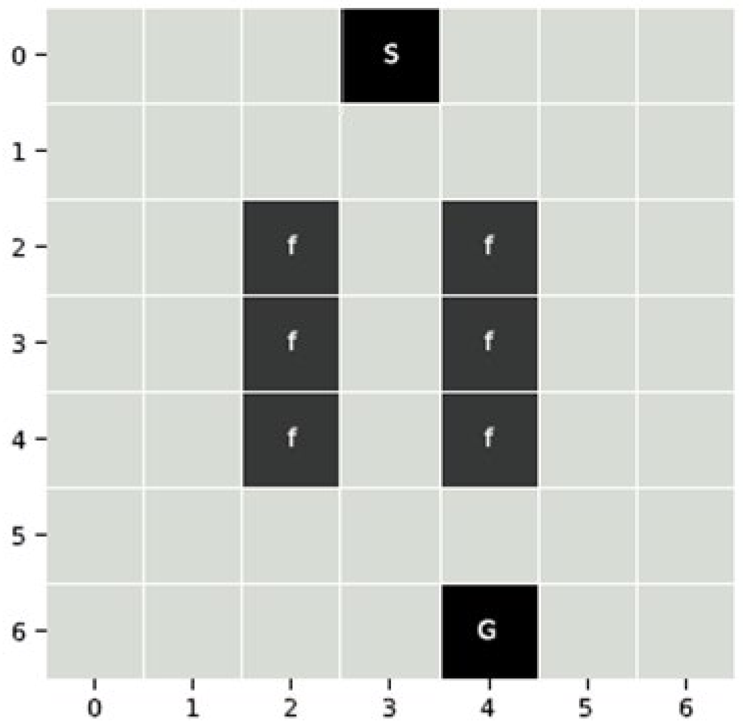


Figure. A simple grid world. The start position is annotated with an **S** and the goal with a **G**. The cells marked with **f** denote fire, which is very damaging to an agent.

Grid Worlds: Environment, Action, State

- The grid world **environment** is characterized by a rectangular grid, where the agent's possible **states** corresponds to the grid's cells.
- The agent's **actions** are limited to moving in the four cardinal directions: left, right, up, or down..



Grid Worlds: Task, Reward, Policy, Hyper-Parameters

- The agent is required to find the most efficient path to the goal while minimizing penalties from hazardous cells and the cost of movement.
- The **reward** structure has been designed as following:
  - the reward for moving between any two normal cells is -1,
  - the reward for arriving at the goal is 50, and
  - the reward for entering a fiery cell is -20.
- The **ϵ-greedy policy** with ϵ=0.1 is used
  - Utilizing Q-learning, combined with an ϵ-greedy policy, allows the agent to balance the need for exploration (to discover potentially better paths) with exploitation (to make use of the current best-known path).
- The chosen hyper-parameters: **learning rate** α and **discount rate** γ are set to α =0.2 and $\gamma = 0.9$.

Initialized Action-Value Table

State	Action	Q value Value
0-0	*up*	0.933
0-0	*down*	−0.119
0-0	*left*	−0.985
0-0	*right*	0.822
0-1	*up*	0.879
0-1	*down*	0.164
0-1	*left*	0.343
0-1	*right*	−0.832
0-2	*up*	0.223
0-2	*down*	0.582
0-2	*left*	0.672
0-2	*right*	0.084
0-3	*up*	−0.308
0-3	*down*	0.247
0-3	*left*	0.963
0-3	*right*	0.455
0-4	*up*	−0.634
	. . .	

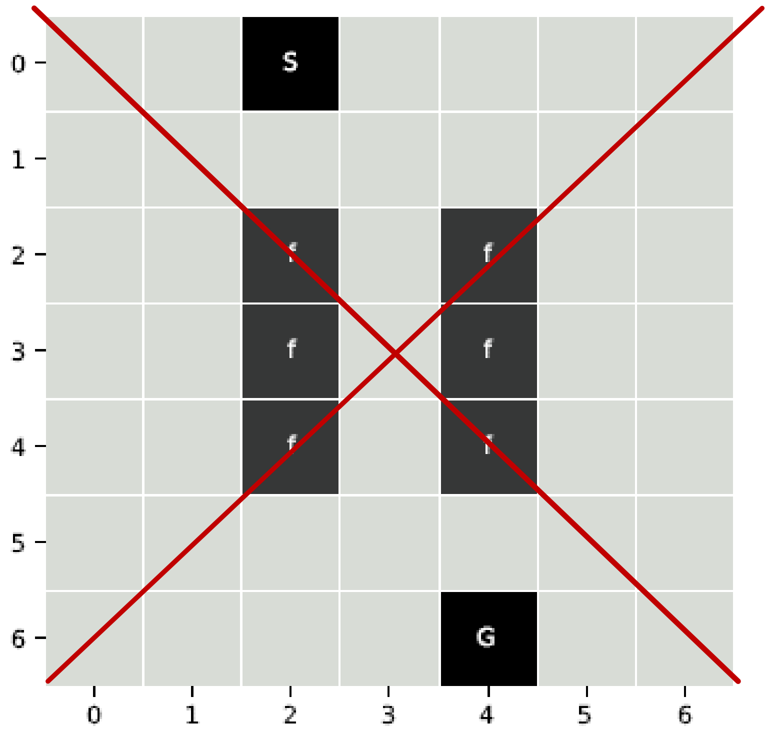
State	Action	Q value Value
	. . .	
2-0	*left*	−0.691
2-0	*right*	0.668
2-1	*up*	−0.918
2-1	*down*	−0.228
2-1	*left*	−0.301
2-1	*right*	−0.317
2-2	*up*	0.633
2-2	*down*	−0.048
2-2	*left*	0.566
2-2	*right*	−0.058
2-3	*up*	0.635
2-3	*down*	0.763
2-3	*left*	−0.121
2-3	*right*	0.562
2-4	*up*	0.629
2-4	*down*	−0.409
	. . .	

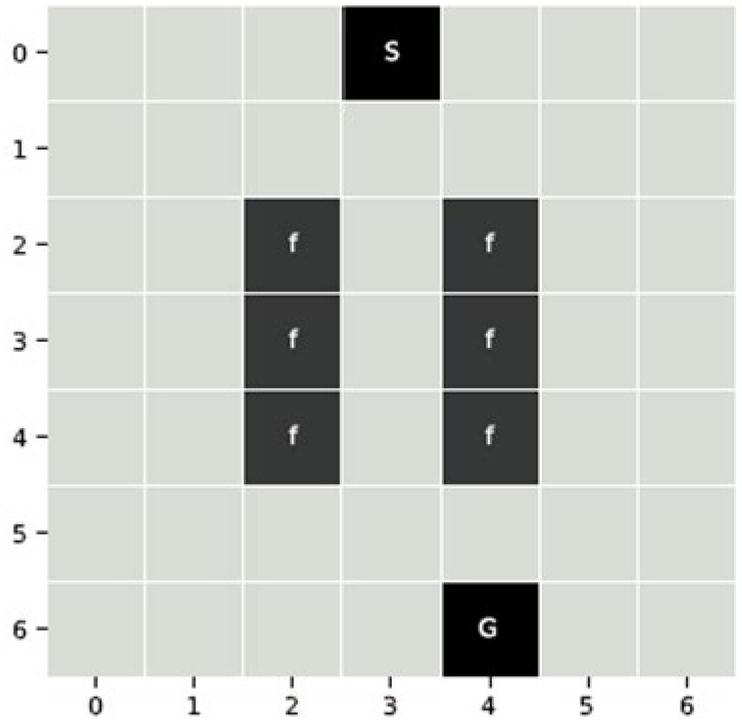
State	Action	Q value Value
	. . .	
6-2	*right*	0.201
6-3	*up*	−0.588
6-3	*down*	0.038
6-3	*left*	0.859
6-3	*right*	−0.085
6-4	*up*	0.000
6-4	*down*	0.000
6-4	*left*	0.000
6-4	*right*	0.000
6-5	*up*	0.321
6-5	*down*	−0.793
6-5	*left*	−0.267
6-5	*right*	0.588
6-6	*up*	−0.870
6-6	*down*	−0.720
6-6	*left*	0.811
6-6	*right*	0.176

Table. A portion of the action-value table for the grid world at its first initialization.
There are 196 entries in the full action-value table—one for each of the four actions that can be taken in each of the 49 (7 x7) states that make up the grid world.
In this example, all entries have been initialized with random numbers in [-1, 1]

NOTE

- The Keller book, pp 660 mentions the start state s_0 is the cell$(0 - 3)$ but the book figures show cell (0-2) for the start.
- This slide shows the figures changed with the start cell (0-3)





First Episode of Q-Learning

- At the beginning of each episode of the Q-learning process, the agent is placed in the starting cell in the grid world ($s_0 = 0-3$)
- The first episode
  - The Q values of actions available to the agent from the staring cell ($s_0 = 0-3$)

    $Q(0-3, up) = -0.308, \qquad Q(0-3, down) = 0.247$

    $\boldsymbol{Q(0-3, left) = 0.963}, \qquad Q(0-3, right) = 0.455$
  - The action with highest Q value is $a_0 = left$.
  - Using the ϵ- greedy policy, the agent generates a random number (from [0,1]) of 0.634, which is greater than ϵ=0.1. So the action uses the **greedy policy (off-policy)** and chooses $a_0 = left$,
  - The agent moves to the *left*, records the next state, $s_1 = 0-2$, and the reward received, r_0= -1 (the reward for moving between any two normal cells)

First Episode of Q-Learning (cont.)

- The first episode (cont.)
  - On the basis of the action $\boldsymbol{a_0} = left$, r_0= -1,
    and the update rule,
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{\boldsymbol{a_{t+1}}} \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \boldsymbol{a_{t+1}}) - Q_\pi(s_t, a_t))$,
    the agent makes an update to $Q(0-3, left)$.
  - For $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$, consider $\max_{a_1} Q(s_1, a_1)$.
    - The Q values of actions available to the agent from state $s_1 = 0-2$
      $Q(0-2, up) = 0.223$, $Q(0-2, down) = 0.582$
      $\boldsymbol{Q(0-2, left) = 0.672}$, $Q(0-2, right) = 0.084$
    - The action with highest Q value is $left$ and uses $Q(0-2, left) = 0.672$ which is an off-policy
  - $Q(0-3, left)$
    $= Q(0-3, left) + \alpha(R(0-3, left) + \gamma \times Q(0-2, left) - Q(0-3, left))$
    = 0.963 + 0.2 × (-1 + 0.9 × 0.672 – 0.963)
    = 0.691

First Episode (cont.)

- The first episode (cont.)
  - After transition, the current state $s_1 = 0 - 2$ of the agent is changed to s_0 (i.e., s_0=s_1)
  - And returns to the beginning of the algorithm to choose the next action according to the policy.
  - The agent already knows the Q values of actions available from the current cell (s_0= $0 - 2$)

    $Q(0-2, up) = 0.223,\ \ Q(0-2, down) = 0.582$

    $\boldsymbol{Q(0-2, left) = 0.672},\ \ Q(0-2, right) = 0.084$
  - Again, following the **ϵ -greedy policy** a random number is generated, 0.073 which is less than $\epsilon = 0.1$, so a **random action** from those available from state $0 - 2$ is selected. Suppose *down*.
  - The agent moves to the *down*, records the next state, s_1= $1 - 2$, and the reward received, r_0= -1 (the reward for moving between any two normal cells)

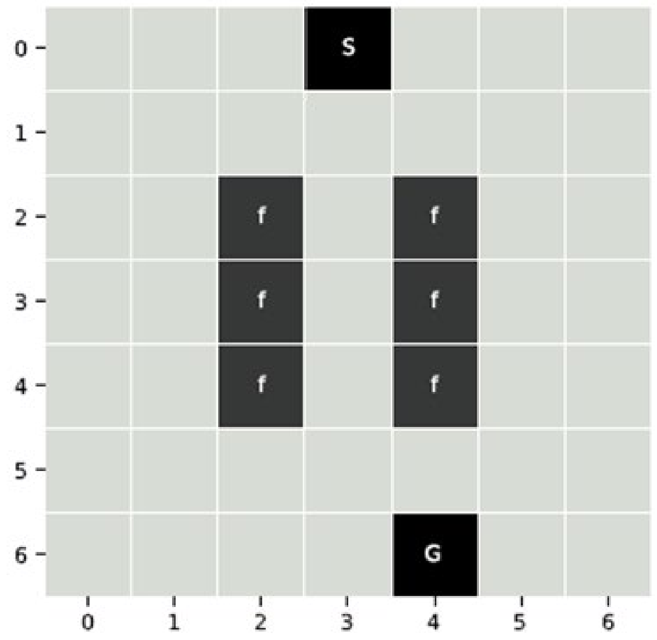
First Episode (cont.)

- The first episode (cont.)
  - …
  - At the end of the first episode the agent reached the goal cell, state $6-4$, by moving $left$ form state $6-5$

$$Q(6-5, left) \leftarrow Q(6-5, left) + \alpha(R(6-5, left) + \gamma \times Q(6-4, up) - Q(6-5, left))$$

$$= -0.267 + 0.2 \times (50 + 0.9 \times 0 - -0.267)$$

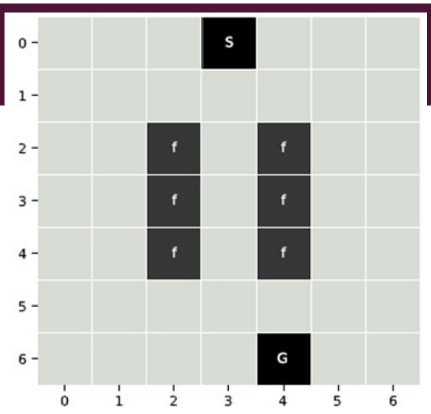
$$= 9.786$$

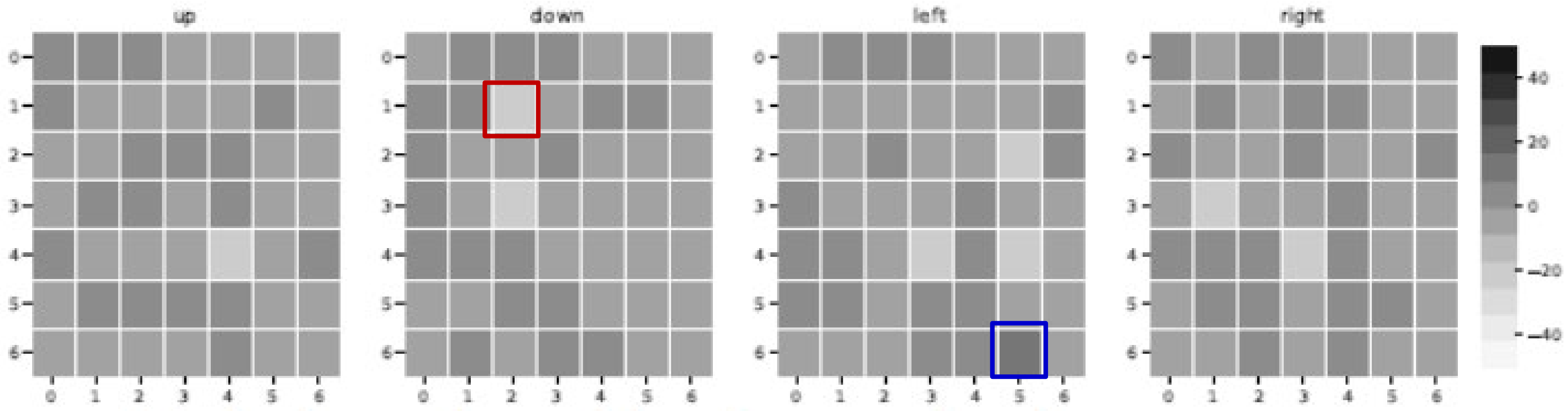


Second Episode, Third Episode, …

- At the beginning of each episode of the Q-learning process, the agent is placed in the starting cell in the grid world ($s_0 = 0 - 3$)
- The Second episode
  - …
  - At the end of the first episode the agent reached the goal cell, state $6 - 4$
- The Third episode
  - …
  - At the end of the first episode the agent reached the goal cell, state $6 - 4$
- …

Action-value table after 1 episode

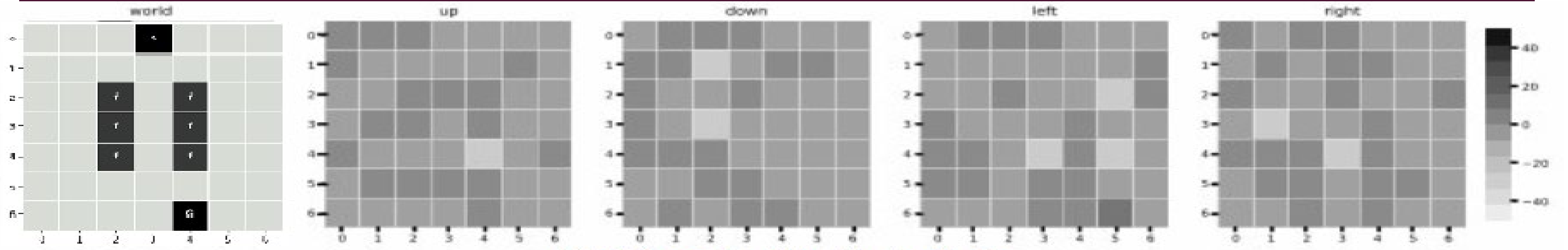




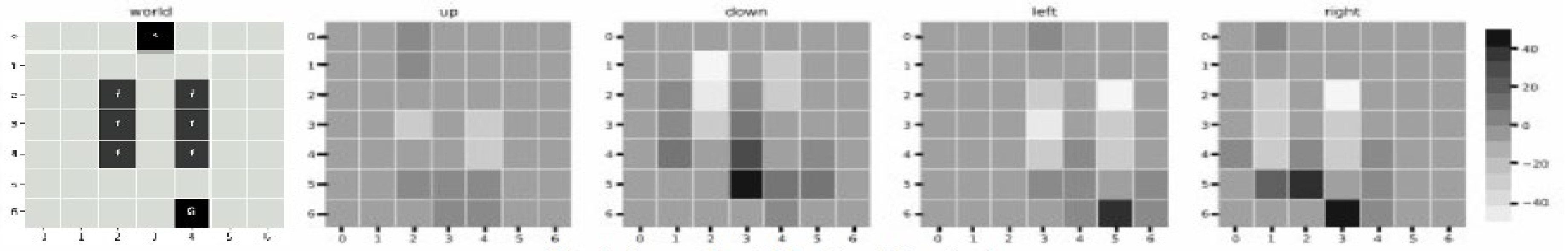
(a) Action-value table after 1 episode

- The value can be seen in the dark shading of Cell 6-5 (State 6-5) in the "left" panel has a large positive reward because the agent reached the goal cell, Cell 6-4.
- Cell 1-2 in the *down* panel has large negative values resulting from trips into the fiery cells and subsequent negative reward.

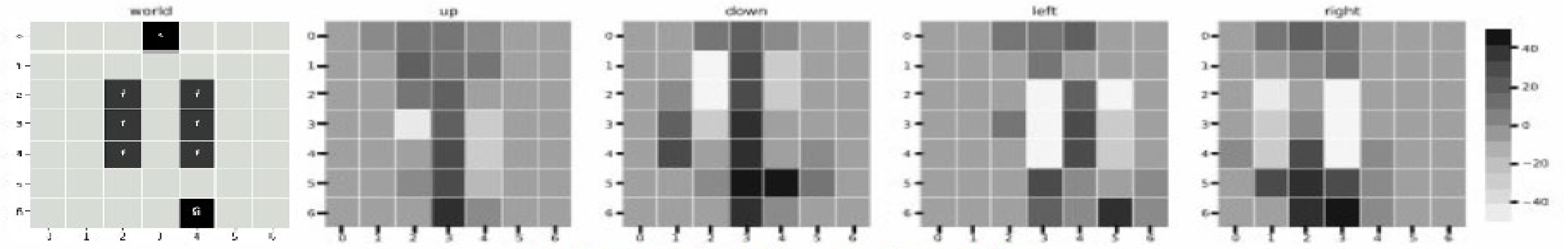
Action-value tables after 1, 35 and 350 episodes



(a) Action-value table after 1 episode



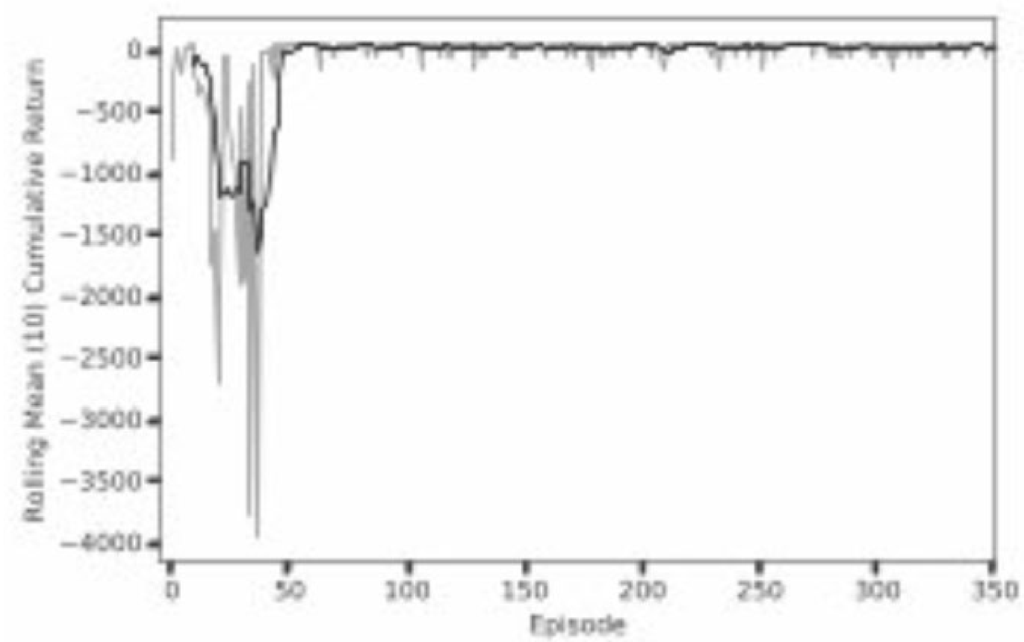
(b) Action-value table after 35 episodes



(c) Action-value table after 350 episodes

Figure (a)–(c) The evolution of the entries in the action-value table over episodes of Q-learning off-policy temporal-difference learning across the grid world. In **(b),** The large positive Q values from actions taken in states near the goal state have started to propagate, although there are not yet large positive values near the start state

Result: Cumulative Reward

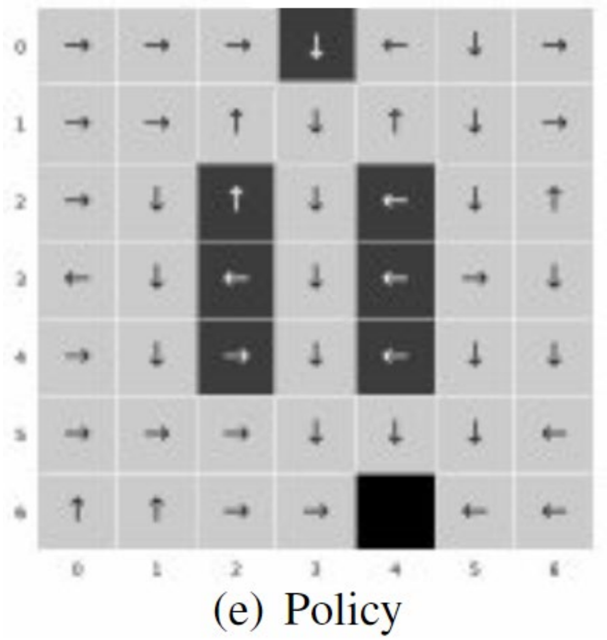


(d) Cumulative Reward

Figure (d) The cumulative reward earned from each episode (Y axis presents a rolling mean across 10 episodes)

- This graph shows that the agent's performance initially declined, and it started to perform quite badly, until after about 40 episodes its performance began to improve. Performance then became stable with a return of about 40 after the agent learned a useful path through the environment.

Result: Policy and Path


(e) Policy

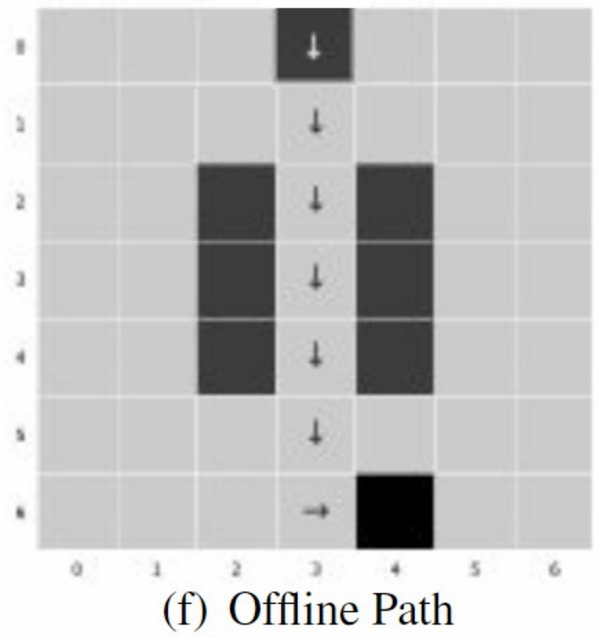

(f) Offline Path

Figure (e) An illustration of the **target policy** learned by the agent after 350 episodes. The arrows show the direction with the highest entry in the action-value table for each state. **(f)** The **path** the agent will take from the start state to the goal state when greedily following the target policy.