Reinforcement learning

Reinforcement learning

- **Model Based RL:** Have the transition probabilities P(s', s, a) and solve the Markov Decision Process (MDP).

- **Model-free RL:** Don't have transition probability & directly learn what action to do when.
  - *Gradient free:*

  - *Gradient based:*

Reinforcement learning

- **Model Based RL:** Have the transition probabilities P(s', s, a) and solve the Markov Decision Process (MDP).
    - Policy Iteration and Value Iteration algorithms
- **Model-free RL:** Don't have transition probability & directly learn what action to do when.

    - *Gradient based:*

Reinforcement learning

- **Model Based RL:** Have the transition probabilities P(s', s, a) and solve the Markov Decision Process (MDP).
    - Policy Iteration and Value Iteration algorithms
- **Model-free RL:** Don't have transition probability & directly learn what action to do when.
    - *Gradient free:*
        - Off Policy: different policy to select the next action and value evaluation, e.g., Q-Learning
        - On Policy: same policy to select the next action and value evaluation
            - SARSA (State–action–reward–state–action)
            - Monte Carlo (Model-free): play many games to estimate policy
    - *Gradient based:*

Reinforcement learning

- **Model Based RL:** Have the transition probabilities P(s', s, a) and solve the Markov Decision Process (MDP).
    - Policy Iteration and Value Iteration algorithms
- **Model-free RL:** Don't have transition probability & directly learn what action to do when.
    - *Gradient free:*
        - Off Policy: different policy to select the next action and value evaluation, e.g., Q-Learning
        - On Policy: same policy to select the next action and value evaluation
            - SARSA (State–action–reward–state–action)
            - Monte Carlo (Model-free): play many games to estimate policy
    - *Gradient based:*
        - Policy Gradient Optimization, e.g., Deep reinforcement learning for policy network

Monte Carlo methods

- To estimate the action values $q[s, a]$, a series of episodes are run.
- Each starts with a given state and action and thereafter follows the current policy, producing a series of actions, states, and returns (figure 19.11a).
- The action value for a given state-action pair under the current policy is estimated as the average of the empirical returns that follow after each time this pair is observed (figure 19.11b).
- Then the policy is updated by choosing the action with the maximum value at every state

$$\pi[a \mid s] \leftarrow \underset{a}{\text{argmax}}[q[s, a]].$$

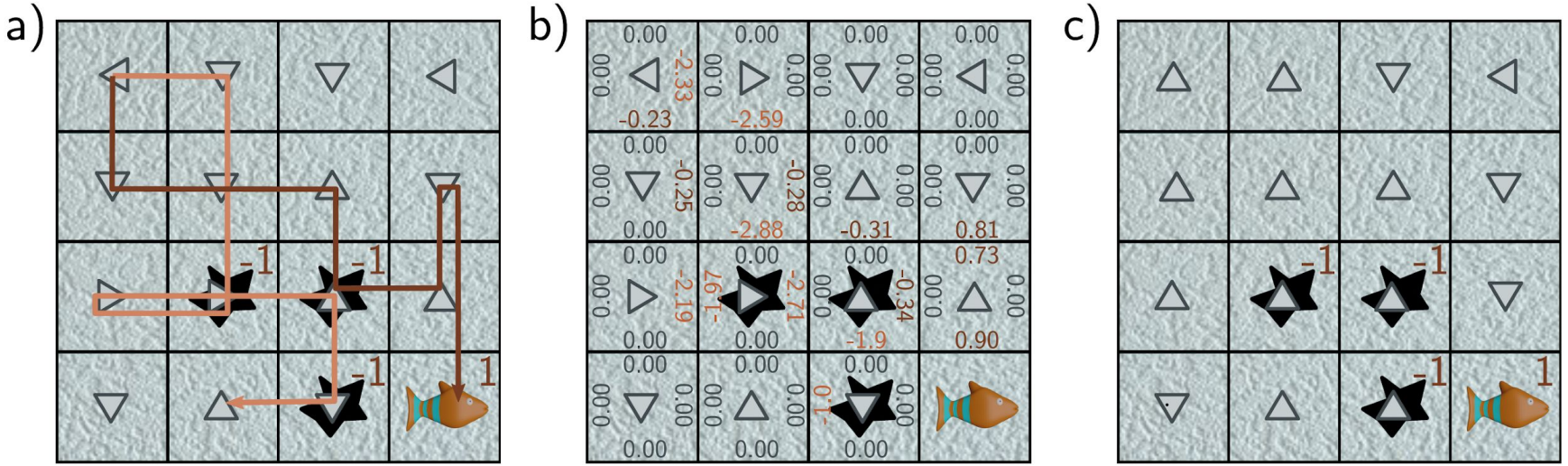


Figure 19.11 Monte Carlo methods. a) The policy (arrows) is initialized randomly. The MDP is repeatedly simulated, and the trajectories of these episodes are stored (orange and brown paths represent two trajectories). b) The action values are empirically estimated based on the observed returns averaged over these trajectories. In this case, the action values were all initially zero and have been updated where an action was observed. c) The policy can then be updated according to the action which received the best (or least bad) reward.

Monte Carlo methods: on-policy

- This is an on-policy method; the current best policy is used to guide the agent through the environment.
- This policy is based on the observed action values in every state, but it's not possible to estimate the value of actions that haven't been used.
- A different approach is to use an **epsilon greedy policy**, in which a random action is taken with probability ϵ, and the optimal action is allotted the remaining probability.
- The choice of ϵ trades off exploitation and exploration.

Monte Carlo methods: off-policy

- In off-policy methods, the optimal policy π (the target policy) is learned based on episodes generated by a different behavior policy π'.
- Some off-policy methods explicitly use importance sampling to estimate the action value under policy π using samples from π'.
- Others, such as Q-learning (described in the next section), estimate the values based on the greedy action

Temporal-difference (TD) learning

Temporal-difference methods are similar to

Dynammic programming update estimates based in part on other learned estimates, without waiting for the final outcome (they bootstrap)

Monte Carlo methods learn directly from raw experience without a model of the environment's dynamics

Temporal difference methods

- TD methods only wait until the next time step to update the value estimates.
- At time $t+1$ they immediately form a target and make an update using the observed reward r_{t+1} and the current estimate $V(s_{t+1})$.

$$V(s_t) \leftarrow V(s_t) + \alpha \left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right),$$

where $\alpha > 0$ is a step-size parameter.

Q- Learning - Off Policy TD Learning

By contrast, Q-Learning is an off-policy algorithm with update

$$Q\left[s_t, a_t\right] \leftarrow Q\left[s_t, a_t\right] + \alpha \overbrace{\left(r\left[s_t, a_t\right] + \gamma \cdot \max_a \left[Q\left[s_{t+1}, a\right]\right] - Q\left[s_t, a_t\right]\right)}^{\text{TD Error}}$$

where now the choice of action at each step is derived from a different behavior policy π'.

- In both cases, the policy is updated by taking the maximum of the action values at each state
- It can be shown the action values will eventually converge, assuming that every state-action pair is visited an infinite number of times.

Q Learning - Off Policy TD Learning

1: Initialize $Q(s,a), \forall s \in S, a \in A \;\; t = 0$, initial state $s_t = s_0$
2: Set π_b to be ϵ-greedy w.r.t. Q
3: loop
4: Take $a_t \sim \pi_b(s_t)$ // Sample action from policy
5: Observe (r_t, s_{t+1})
6: Update $Q[s_t, a_t] \leftarrow Q[s_t, a_t] + \alpha(r[s_t, a_t] + \gamma \cdot \max_a [Q[s_{t+1}, a]] - Q[s_t, a_t])$
7: Policy improvement $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
8: $t = t + 1$ (until terminal state not reached)
9: end loop

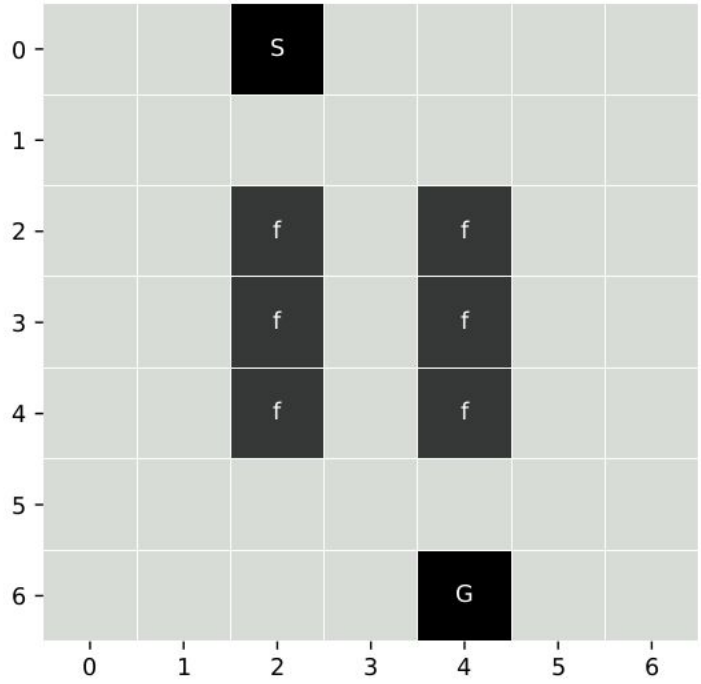


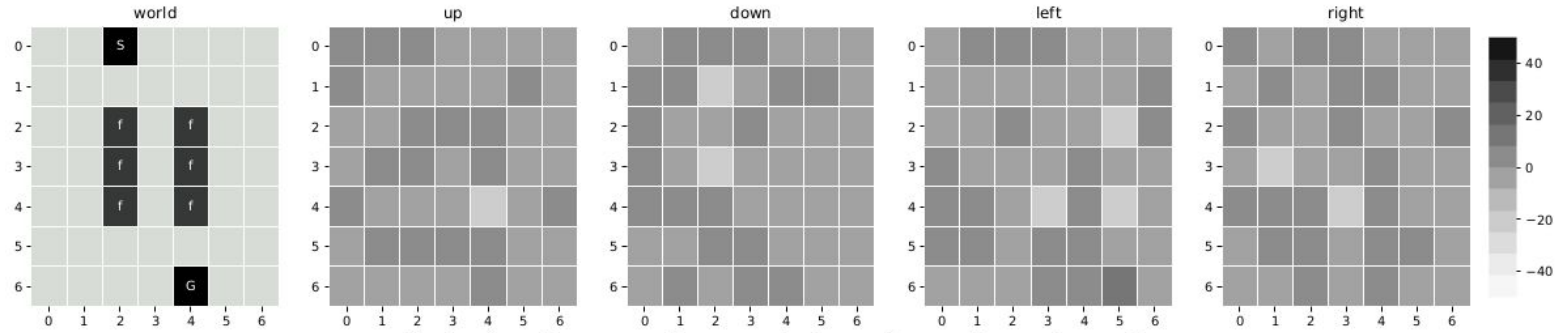
Figure 4: A simple grid world. The start position is annotated with an *S* and the goal with a *G*. The squares marked *f* denote fire, which is very damaging to an agent.

Table 3: A portion of the action-value table for the grid world example at its first initialization.

State	Action	Value	State	Action	Value	State	Action	Value
0-0	*up*	0.933		...			...	
0-0	*down*	−0.119	2-0	*left*	−0.691	6-2	*right*	0.201
0-0	*left*	−0.985	2-0	*right*	0.668	6-3	*up*	−0.588
0-0	*right*	0.822	2-1	*up*	−0.918	6-3	*down*	0.038
0-1	*up*	0.879	2-1	*down*	−0.228	6-3	*left*	0.859
0-1	*down*	0.164	2-1	*left*	−0.301	6-3	*right*	−0.085
0-1	*left*	0.343	2-1	*right*	−0.317	6-4	*up*	0.000
0-1	*right*	−0.832	2-2	*up*	0.633	6-4	*down*	0.000
0-2	*up*	0.223	2-2	*down*	−0.048	6-4	*left*	0.000
0-2	*down*	0.582	2-2	*left*	0.566	6-4	*right*	0.000
0-2	*left*	0.672	2-2	*right*	−0.058	6-5	*up*	0.321
0-2	*right*	0.084	2-3	*up*	0.635	6-5	*down*	−0.793
0-3	*up*	−0.308	2-3	*down*	0.763	6-5	*left*	−0.267
0-3	*down*	0.247	2-3	*left*	−0.121	6-5	*right*	0.588
0-3	*left*	0.963	2-3	*right*	0.562	6-6	*up*	−0.870
0-3	*right*	0.455	2-4	*up*	0.629	6-6	*down*	−0.720
0-4	*up*	−0.634	2-4	*down*	−0.409	6-6	*left*	0.811
	...			...		6-6	*right*	0.176

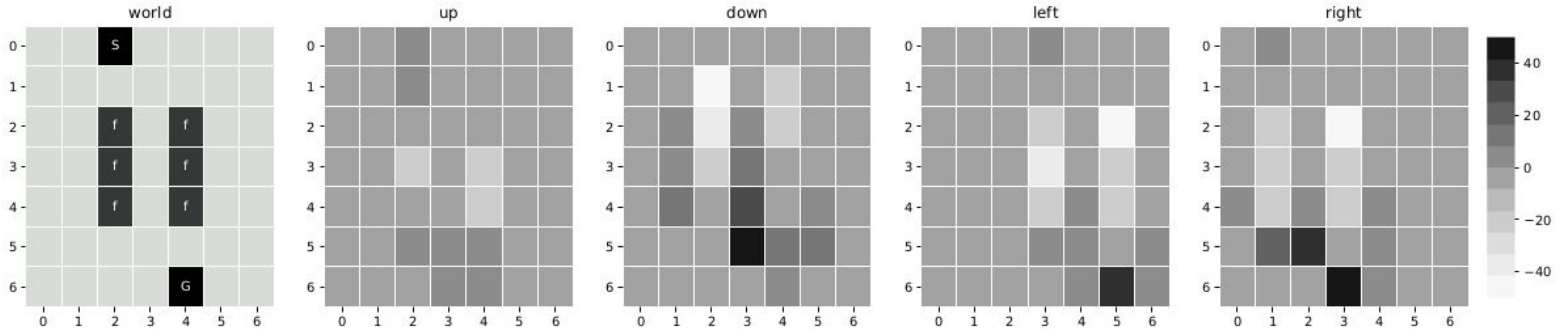
$$Q(\text{0-3}, \textit{left}) \leftarrow Q(\text{0-3}, \textit{left}) + \alpha \times (R(\text{0-3}, \textit{left}) + \gamma \times Q(\qquad\qquad) - Q(\text{0-3}, \textit{left}))$$

$$
\begin{aligned}
Q(\text{0-3}, \textit{left}) \leftarrow & Q(\text{0-3}, \textit{left}) + \alpha \times (R(\text{0-3}, \textit{left}) + \gamma \times Q(\text{0-2}, \textit{left}) - Q(\text{0-3}, \textit{left})) \\
& 0.963 + 0.2 \times (-1 + 0.9 \times 0.672 - 0.963) \\
& 0.691
\end{aligned}
$$



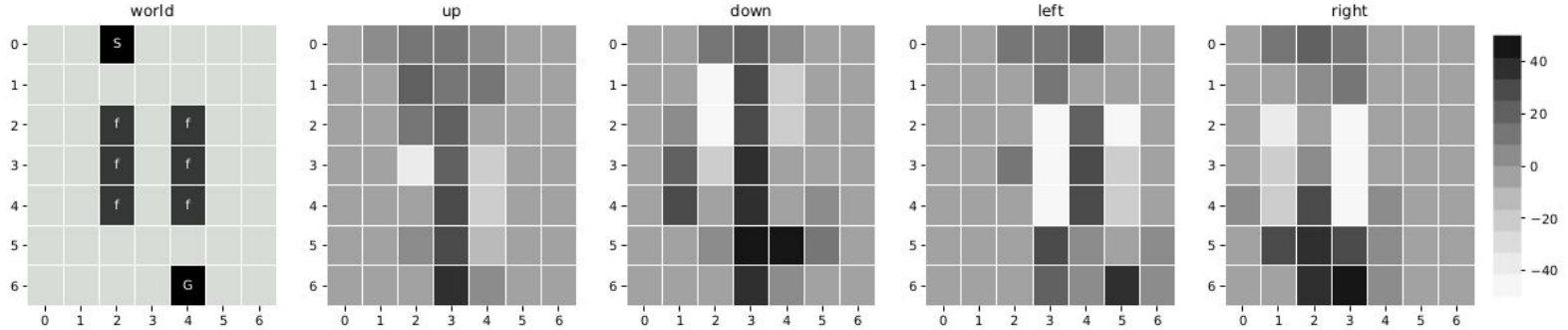
(a) Action-value table after 1 episode

Figure 5: (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.



(b) Action-value table after 35 episodes

Figure 6: (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.



(c) Action-value table after 350 episodes

Figure 7: (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.

Table 4: A portion of the action-value table for the grid world example after 350 episodes of Q-learning have elapsed.

State	Action	Value
0-0	*up*	−1.627
0-0	*down*	−1.255
0-0	*left*	−1.655
0-0	*right*	−1.000
0-1	*up*	1.302
0-1	*down*	−1.900
0-1	*left*	−1.900
0-1	*right*	15.173
0-2	*up*	13.299
0-2	*down*	12.009
0-2	*left*	8.858
0-2	*right*	18.698
0-3	*up*	13.921
0-3	*down*	21.886
0-3	*left*	15.900
0-3	*right*	13.846
0-4	*up*	1.637
...		

State	Action	Value
...		
2-0	*left*	−1.583
2-0	*right*	−1.217
2-1	*up*	−1.493
2-1	*down*	4.132
2-1	*left*	−1.643
2-1	*right*	−36.301
2-2	*up*	13.247
2-2	*down*	−46.862
2-2	*left*	−0.858
2-2	*right*	−1.157
2-3	*up*	16.973
2-3	*down*	29.366
2-3	*left*	−88.492
2-3	*right*	−77.447
2-4	*up*	−1.016
2-4	*down*	−20.255
...		

State	Action	Value
...		
6-2	*right*	40.190
6-3	*up*	34.375
6-3	*down*	40.206
6-3	*left*	24.784
6-3	*right*	50.000
6-4	*up*	0.000
6-4	*down*	0.000
6-4	*left*	0.000
6-4	*right*	0.000
6-5	*up*	−0.353
6-5	*down*	−0.793
6-5	*left*	36.823
6-5	*right*	−0.342
6-6	*up*	−0.870
6-6	*down*	−0.720
6-6	*left*	1.008
6-6	*right*	−0.802

SARSA - On Policy TD Learning

SARSA (State-Action-Reward-State-Action) is an on-policy algorithm with update:

$$Q\left[s_t, a_t\right] \leftarrow Q\left[s_t, a_t\right] + \alpha \underbrace{\left(r\left[s_t, a_t\right] + \gamma \cdot Q\left[s_{t+1}, a_{t+1}\right] - Q\left[s_t, a_t\right]\right)}_{\text{TD Error}}$$

where $\alpha \in \mathbb{R}^+$is the learning rate.

- The bracketed term is called the TD error and measures the consistency between the estimated action value $Q\left[s_t, a_t\right]$ and the estimate $r\left[s_t, a_t\right] + \gamma \cdot Q\left[s_{t+1}, a_{t+1}\right]$ after taking a single step.

SARSA - On Policy TD Learning

Initialize all $Q(,)$ arbitrarily
For all episodes

- Initalize s_t
- Choose a_t using policy derived from Q, e.g., ϵ-greedy
- Repeat
  - Take action a_t, observe r_t and s_{t+1}
  - Choose a_{t+1} using policy derived from Q, e.g., ϵ-greedy
  - Update $Q\left[s_t, a_t\right] \leftarrow Q\left[s_t, a_t\right] + \alpha\left(r\left[s_t, a_t\right] + \gamma \cdot Q\left[s_{t+1}, a_{t+1}\right] - Q\left[s_t, a_t\right]\right)$
  - $s_t \leftarrow s_{t+1}, a_t \leftarrow a_{t+1}$
- Until s_t is terminal state

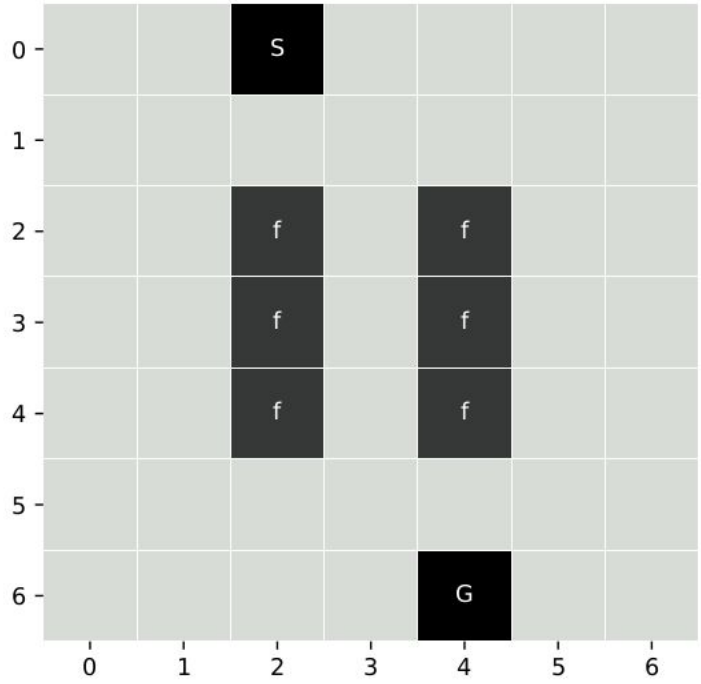
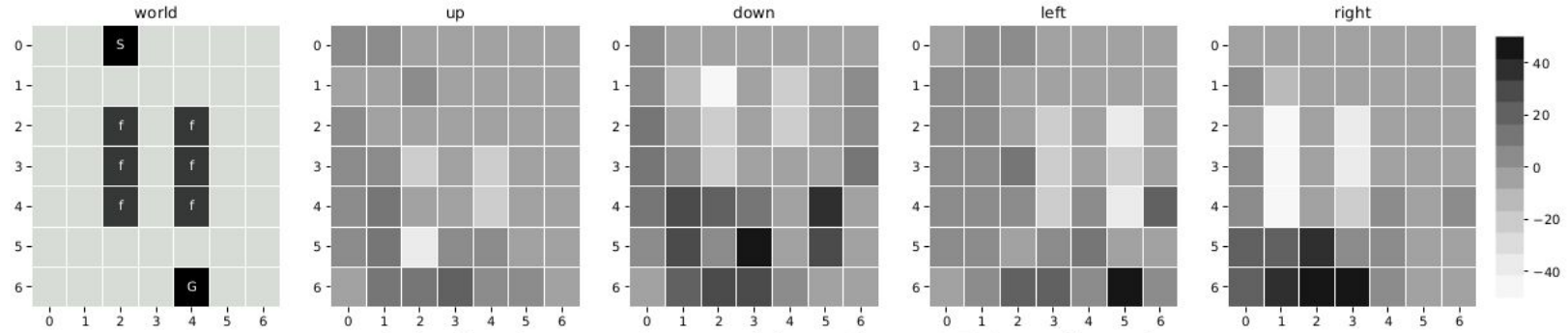


Figure 4: A simple grid world. The start position is annotated with an *S* and the goal with a *G*. The squares marked *f* denote fire, which is very damaging to an agent.

Table 3: A portion of the action-value table for the grid world example at its first initialization.

State	Action	Value	State	Action	Value	State	Action	Value
0-0	*up*	0.933		...			...	
0-0	*down*	−0.119	2-0	*left*	−0.691	6-2	*right*	0.201
0-0	*left*	−0.985	2-0	*right*	0.668	6-3	*up*	−0.588
0-0	*right*	0.822	2-1	*up*	−0.918	6-3	*down*	0.038
0-1	*up*	0.879	2-1	*down*	−0.228	6-3	*left*	0.859
0-1	*down*	0.164	2-1	*left*	−0.301	6-3	*right*	−0.085
0-1	*left*	0.343	2-1	*right*	−0.317	6-4	*up*	0.000
0-1	*right*	−0.832	2-2	*up*	0.633	6-4	*down*	0.000
0-2	*up*	0.223	2-2	*down*	−0.048	6-4	*left*	0.000
0-2	*down*	0.582	2-2	*left*	0.566	6-4	*right*	0.000
0-2	*left*	0.672	2-2	*right*	−0.058	6-5	*up*	0.321
0-2	*right*	0.084	2-3	*up*	0.635	6-5	*down*	−0.793
0-3	*up*	−0.308	2-3	*down*	0.763	6-5	*left*	−0.267
0-3	*down*	0.247	2-3	*left*	−0.121	6-5	*right*	0.588
0-3	*left*	0.963	2-3	*right*	0.562	6-6	*up*	−0.870
0-3	*right*	0.455	2-4	*up*	0.629	6-6	*down*	−0.720
0-4	*up*	−0.634	2-4	*down*	−0.409	6-6	*left*	0.811
	...			...		6-6	*right*	0.176

$$
\begin{aligned}
Q(\text{0-3}, \textit{left}) \leftarrow & Q(\text{0-3}, \textit{left}) + \alpha \times (R(\text{0-3}, \textit{left}) + \gamma \times Q(\text{0-2}, \textit{down}) - Q(\text{0-3}, \textit{left})) \\
& 0.963 + 0.2 \times (-1 + 0.9 \times 0.582 - 0.963) \\
& 0.675
\end{aligned}
$$



(a) Action-value table after 350 episodes

Figure 9: (a) A visualization of the final action-value table for an agent trained using SARSA on-policy temporal-difference learning across the grid world after 350 episodes.

Summary

Algorithm	Description	Policy	Action Space	State Space	Operator
Monte Carlo (Model-free)	Estimate $Q_\pi^{avg}(s, a)$ average of all the trajectories	Either	Discrete	Discrete	Sample-means
Q-learning	Aim to learns $Q_\pi(s, a)$	Off-policy	Discrete	Discrete	Q-value
SARSA	Aim to learns $Q_\pi(s, a)$	On-policy	Discrete	Discrete	Q-value