Machine Learning

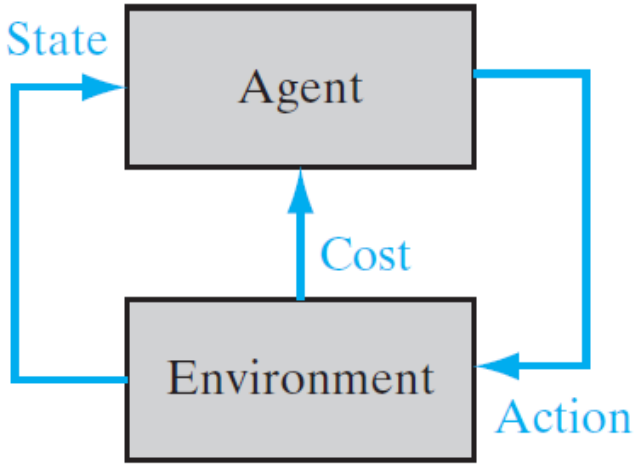
Reinforcement Learning

Dr. Pratyay Kuila

Dept. of Computer Science & Engineering
NIT Sikkim, Ravangla-737139

Reinforcement Learning

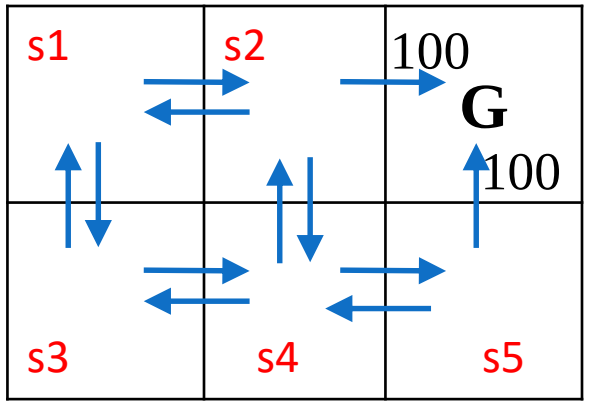
- Learning through rewards.



- *In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives reward (or penalty) for its actions in trying to solve a problem.*
- *After a set of trial-and error runs, it should learn the best policy, which is the sequence of actions that maximize the total reward.*

Reinforcement Learning

- Let us consider the environment. It consists of six different rooms or states. *G* is the goal state.
- The arrows indicate the possible movements or actions from one state to other state.
- Consider the bottom right state (s5). The up arrow towards the goal state *G* with the value 100 indicates a reward for immediately achieving the goal.



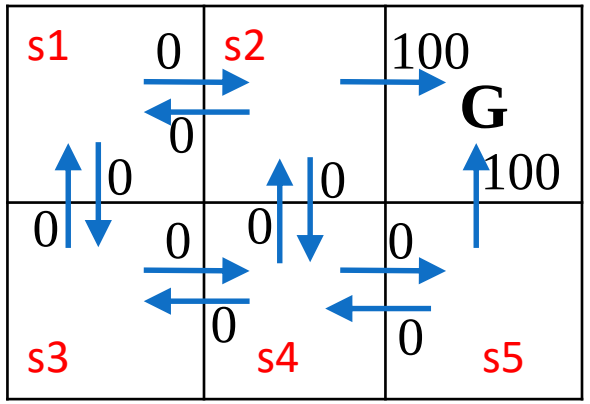
An environment

- Consider a robot or an agent at any state (say s3).
  - The task of the agent is to reach to the goal state (G) as immediate as possible to achieve maximum reward.
  - *Delayed reward:* Late rewards are panelized.

Reinforcement Learning

- *Delayed reward:* the *discounted cumulative reward value* V(*s*) achieved by following an arbitrary sequence of actions from an arbitrary initial state *s* as follows:

$$V(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots$$
$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$



r (*s*, *a*) (immediate reward) values

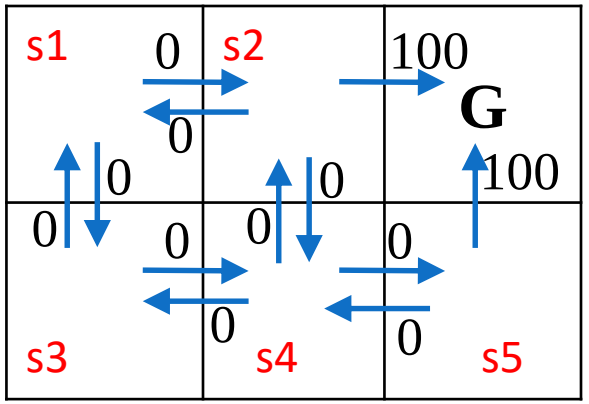
- Here $0 \leq \gamma < 1$ is a constant that determines the relative value of delayed versus immediate rewards.
- The sequence of rewards r_{t+i} is generated by beginning at s_t, and repeatedly applying the sequence of actions.
- The V(*s*4) can be calculated by applying the actions *s*4 → *s*5 → G:

$$V(s_4) = 0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \ldots = 90 \text{ (consider } \gamma = 0.9)$$

Reinforcement Learning

- ***Delayed reward:*** the ***discounted cumulative reward value*** $V(s)$ achieved by following an arbitrary sequence of actions from an arbitrary initial state s as follows:

$$V(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots$$
$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$



$r(s, a)$ (immediate reward) values

- The V($s4$) can be calculated by applying the actions $s4 \rightarrow s5 \rightarrow$ G:

$$V(s_4) = 0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \ldots = 90 \text{ (consider } \gamma = 0.9)$$

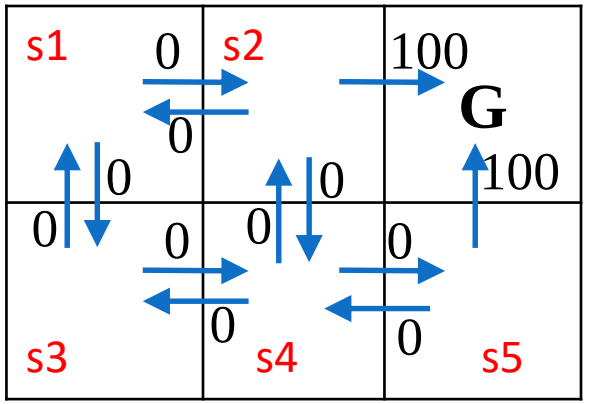
- The V($s4$) can be calculated by applying the actions $s4 \rightarrow s3 \rightarrow s1 \rightarrow s2 \rightarrow$ G:

$$V(s_4) = 0 + \gamma .0 + \gamma^2 .0 + \gamma^3 .100 + \ldots = 72.9 \text{ (consider } \gamma = 0.9)$$

- What if it applies the actions $s4 \rightarrow s2 \rightarrow$ G?

Reinforcement Learning

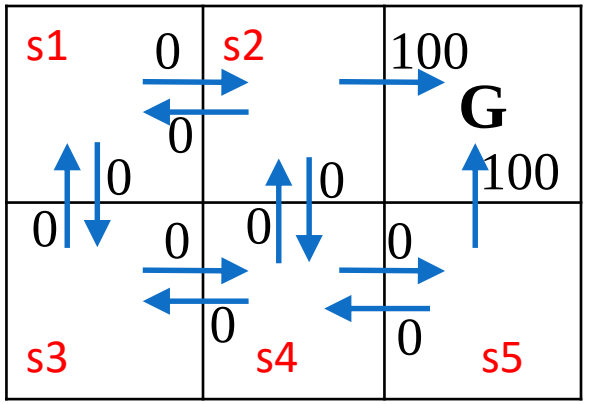
- How to find out the maximum discounted reward value V*(*s*) for each state *s*?
- What is the optimal sequence of actions from a state *s* such that the reward on that state, i.e., V(*s*) is maximized due to the taken sequence of actions?



r (*s*, *a*) (immediate reward) values

Reinforcement Learning

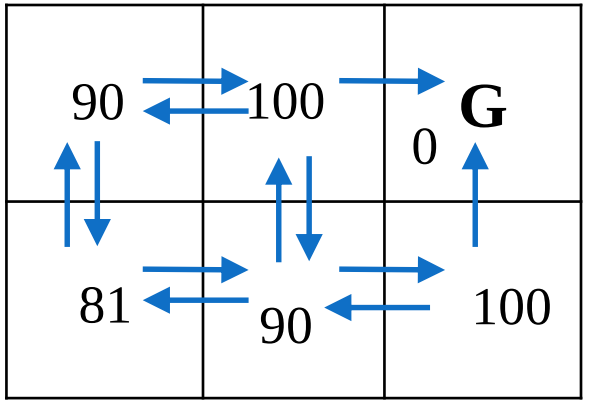
- How to find out the maximum discounted reward value V*(*s*) for each state *s*?
- What is the optimal sequence of actions from a state *s* such that the reward on that state, i.e., V(*s*) is maximized due to the taken sequence of actions?



r (*s*, *a*) (immediate reward) values

- The maximum value, V* for the state (*s*3) is 81.
- This is because the optimal policy will move the agent

$$s3 \rightarrow s4 \rightarrow s5 \rightarrow \text{G}.$$



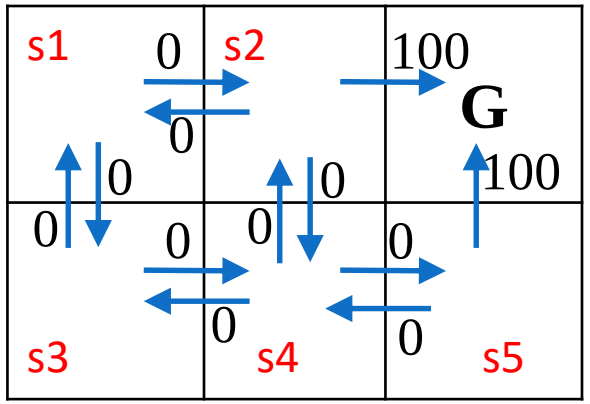
Optimal V*(*s*) values

- Thus, the maximum discounted future reward V*(s3) is:

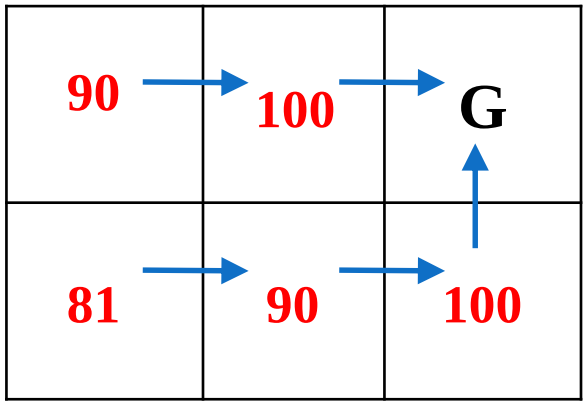
$$V^*(s_3) = 0 + \gamma 0 + \gamma^2 100 = 81 \ \ (\text{consider } \gamma = 0.9)$$

Reinforcement Learning

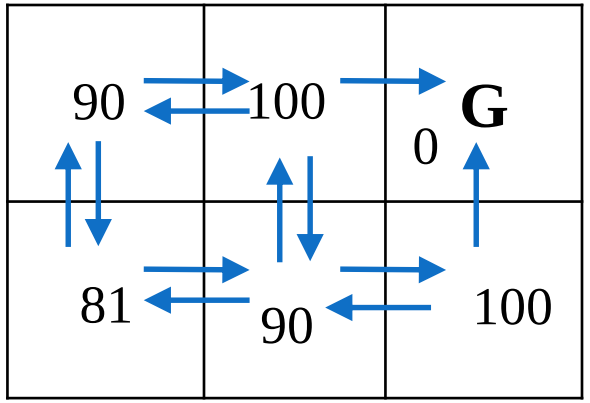
- How to find out the maximum discounted reward value V*(*s*) for each state *s*?
- What is the optimal sequence of actions from a state *s* such that the reward on that state, i.e., V(*s*) is maximized due to the taken sequence of actions?



r (*s*, *a*) (immediate reward) values



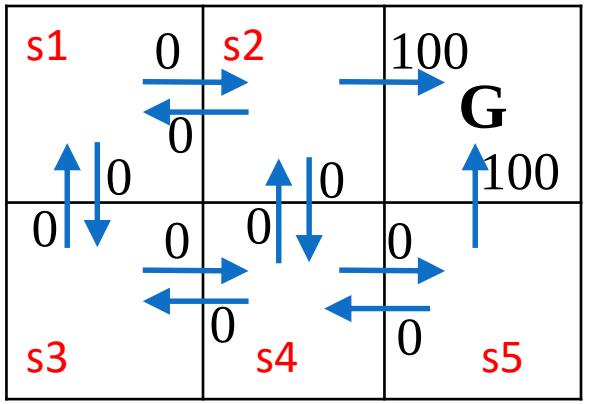
Optimal actions



Optimal V*(*s*) values

Reinforcement Learning

- How to find out the maximum discounted reward value V*(*s*) for each state *s*?
- What is the optimal sequence of actions from a state *s* such that the reward on that state, i.e., V(*s*) is maximized due to the taken sequence of actions?



r (*s*, *a*) (immediate reward) values

- Possible Challenges:
  - We may not have always complete knowledge about the environment. Only information about the current state and possible immediate reward related to the action just taken. We have to explore the environment and answer these questions.
  - Sometimes the actions and the corresponding rewards may be stochastic or probabilistic.

Reinforcement Learning: Policy

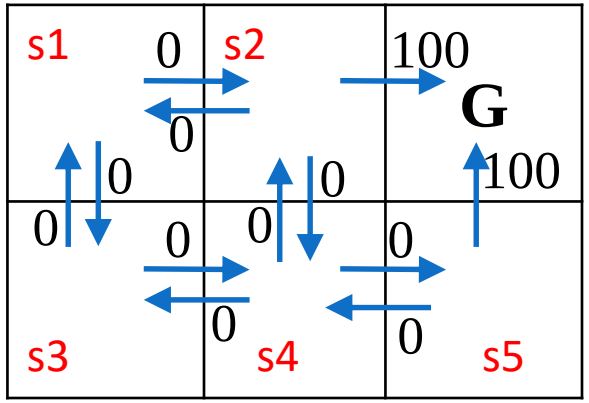
- A ***policy*** is defined as a *mapping of states into actions*.
- Policy is a rule used by the agent to decide what to do, given knowledge of the current state of the environment.

Definition 2 (Policy): A policy *is a mapping* $\pi : S \rightarrow A$. The policy defines the action to be taken in any state s_j, i.e., $a_j = \pi(s_j)$.

- More precisely, this is the definition of a *stationary policy* since the choice of the action does not depend on the time.
- We could define a *non-stationary policy* as a sequence of mappings $\pi_t : S \rightarrow A$ indexed by t.

Reinforcement Learning: Policy

Definition 3 (Policy Value): The value $V^{\pi}(s)$ of a policy π at state $s \in S$ is defined as the expected reward returned when starting at s and following policy π.



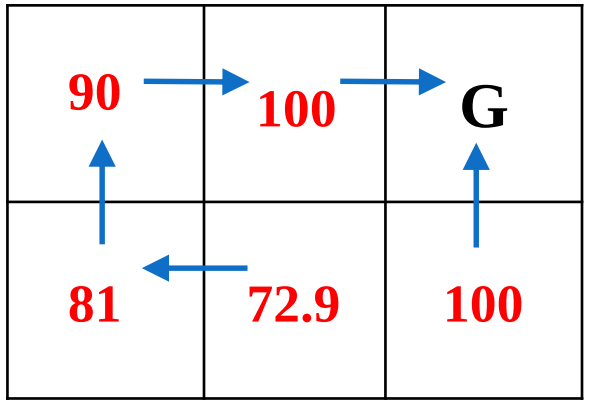
r (s, a) (immediate reward) values

- As per the given policy π, the agent will move from $s4$:

$$s4 \rightarrow s3 \rightarrow s1 \rightarrow s2 \rightarrow \text{G}.$$

- Thus, the policy value, $V^{\pi}(s4)$ is:

$$V^{\pi}(s_4) = 0 + \gamma 0 + \gamma^2 0 + \gamma^3 100 = 72.9 \text{ (consider } \gamma = 0.9)$$



A Policy (π) and its value

Reinforcement Learning: Policy

- We are now in a position to state precisely the agent's learning task. We require that *the agent learn a policy π that maximizes $V^{\pi}(s)$ for all states s.*
- We will call such a policy an *optimal policy* and denote it by π*.

$$\pi^* = \text{argmax}_{\pi} V^{\pi}(s), (\forall s)$$

- We will refer to the value function $V^{\pi*}(s)$ of such an optimal policy as $V^*(s)$.

$$V^{\pi*}(s) \geq V^{\pi}(s), \forall s \in S$$

- $V^{\pi*}(s) \approx V^*(s)$ gives the maximum discounted cumulative reward that the agent can obtain starting from state *s*.

Reinforcement Learning: Reward Value

- *Discounted cumulative reward value $V^{\pi}(s_t)$:*

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- The quantity $V^{\pi}(s)$ is often denoted as the *discounted cumulative reward* achieved by policy π from initial state s.
- It is reasonable to discount future rewards relative to immediate rewards because, in many cases, we prefer to obtain the reward sooner rather than later.
- The *discount factor*, γ describes the preference of an agent for current rewards over future rewards.
  - When γ is close to 0, the rewards in distant future are viewed as insignificant (only the immediate reward is considered).
  - When γ is close to 1, an agent is more willing to wait for long term reward (future rewards are given more preference over the immediate reward).
  - When γ is exactly 1, discounted rewards reduce to the special case of additive rewards.

Reinforcement Learning: Reward Value

- Other definitions of total reward have also been explored.

- *finite horizon reward*: $V^{\pi}(s_t) = \sum_{i=0}^{h} r_{t+i}$

  considers the undiscounted sum of rewards over a finite number *h* of steps.

- *average reward*: $V^{\pi}(s_t) = \lim_{h \to \infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$

  considers the average reward per time step over the entire lifetime of the agent.

Reinforcement Learning

- The objective of the agent is to maximize his reward and thus to determine the best course of actions, or *policy*, to achieve that objective.
- However, the information he receives from the environment is only the immediate reward related to the action just taken. No future or long-term reward feedback is provided by the environment.
- An important aspect of reinforcement learning is to take into consideration *delayed rewards* or *penalties*.
- *The agent is faced with the dilemma between exploring unknown states and actions to gain more information about the environment and the rewards, and exploiting the information already collected to optimize his reward.*
- This is known as the *exploration versus exploitation tradeoff* inherent in reinforcement learning.

Reinforcement Learning

- Note that within this scenario, training and testing phases are intermixed.

- Two main settings can be distinguished here:
  - the case where the *environment model is known* to the agent, in which case his objective of maximizing the reward received is reduced to a *planning problem*, and
  - the case where the *environment model is unknown*, in which case he faces a *learning problem*.

- In the latter case, the agent must learn from the state and reward information gathered to both gain information about the environment and determine the best action *policy*.

- The problem is modeled using a *Markov decision process* (MDP).

Markov Decision Process

Markov Property

- "The future is independent of the past given the present."
- A state s_t is Markov if and only if

$$P[s_{t+1}|s_1, s_2, ..., s_t] = P[s_{t+1}|s_t]$$

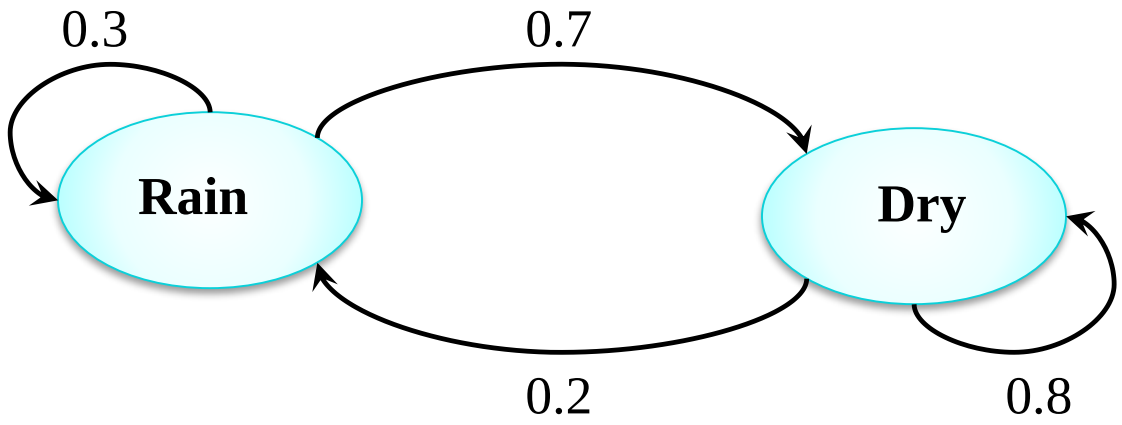
- The state captures all relevant information from the history.
- Once the state is known, the history may be thrown away.
- The current state is a sufficient statistic of the future.

A *Markov process* (or *Markov chain*) is a memoryless random process, i.e., a sequence of random states s_1, s_2, … with the Markov property.

- S is a finite set of states with initial probability.
- M is state transition probability matrix.

$$M_{ss'} = P[s_{t+1} = s'|s_t = s]$$

Markov Model: Example



- Two states : ‘Rain’ and ‘Dry’.
- Initial probabilities: say $P(\text{‘}Rain\text{’}) = 0.4$, $P(\text{‘}Dry\text{’}) = 0.6$.

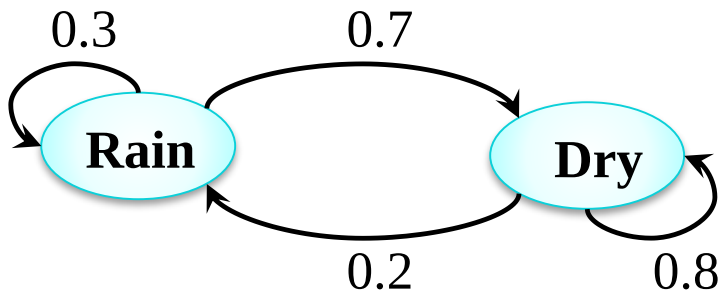
- Transition probabilities:
  - $P(\text{‘}Rain\text{’}|\text{‘}Rain\text{’})=0.3$ and $P(\text{‘}Dry\text{’}|\text{‘}Rain\text{’})=0.7$
  - $P(\text{‘}Rain\text{’}|\text{‘}Dry\text{’})=0.2$ and $P(\text{‘}Dry\text{’}|\text{‘}Dry\text{’})=0.8$

Probability of Sequence of States

- By Markov chain property, probability of state sequence can be found by the formula:

$$\begin{aligned} P(s_{i1}, s_{i2}, \ldots, s_{ik}) &= P(s_{ik}|s_{i1}, s_{i2}, \ldots, s_{ik-1})P(s_{i1}, s_{i2}, \ldots, s_{ik-1}) \\ &= P(s_{ik}|s_{ik-1})P(s_{i1}, s_{i2}, \ldots, s_{ik-1}) \\ &= P(s_{ik}|s_{ik-1})P(s_{ik-1}|s_{ik-2})\ldots P(s_{i2}|s_{i1})P(s_{i1}) \end{aligned}$$

- Suppose we want to calculate a probability of a sequence of states in our example,
  {'*Dry*', '*Dry*', '*Rain*', '*Rain*'}.



$$\begin{aligned} &P(\{Dry, Dry, Rain, Rain\}) \\ &= P(Rain|Rain)P(Rain|Dry)P(Dry|Dry)P(Dry) \\ &= 0.3 \times 0.2 \times 0.8 \times 0.6 \end{aligned}$$

Markov Decision Process

A Markov Decision Process (MDP) model contains:

- A set of possible states *S*, may be finite or infinite.
- A starting state s_0, $s_0 \in S$.
- A set of possible actions *A*, may be finite or infinite.
- A real valued reward function *r*(*s*,*a*) with some probability.
- A transition function δ of each action's effects in each state with some probability.

Representing Actions:

▷ Deterministic Actions: $\delta : S \times A \rightarrow S$
For each state and action we specify a new state.

▷ Stochastic Actions: $\delta : S \times A \rightarrow Prob(S)$
For each state and action we specify a probability distribution over next states.

❖ The model is *Markovian* because the transition and reward probabilities depend only on the current state *s* and not the entire history of states and actions taken.

Solving MDPs: Value Iteration

- In an MDP, the functions $\delta(s_t, a_t)$ and $r(s_t, a_t)$ depend only on the current state and action, and not on earlier states or actions.
- Solution approach is based on *Dynamic Programming (DP)* by *Richard Bellman* (1957).
- *Principle of Optimality for Policy:* Optimal policy has the property that whatever the initial state and initial decisions are, the remaining decision must constitute an optimal policy with regard to the state resulting from previous decision.
- *Principle of Optimality for Value*: The optimal value of a state is the immediate cost for the state plus the expected discounted optimal value for the next state, assuming that the agent chooses the optimal action.
- For every MDP, there exist an optimal deterministic policy which maximizes the expected, total discounted return from any initial state.

Solving MDPs: Value Iteration

- In an MDP, the functions $\delta(s_t, a_t)$ and $r(s_t, a_t)$ depend only on the current state and action, and not on earlier states or actions.
- Solution approach is based on *Dynamic Programming (DP)* by *Richard Bellman* (1957).

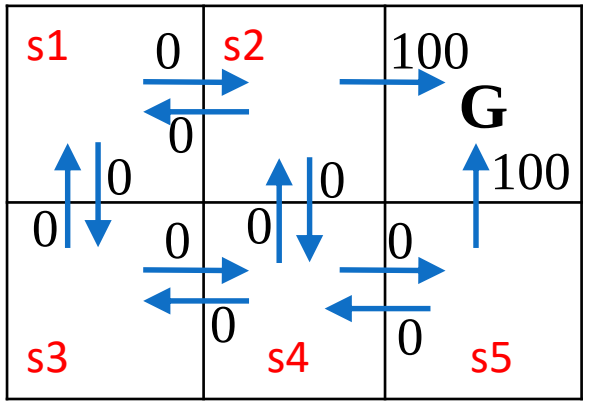
Bellman equation: The values $V^{\pi}(s)$ of policy π at states $s \in S$ for an infinite horizon MDP obey the following system of linear equations:

$$V^{\pi}(s) = \mathbb{E}[r(s, \pi(s))] + \gamma \sum_{s' \in S} P(s'|s, \pi(s)).V^{\pi}(s')$$

- *The value of a state is the immediate reward for that state plus the expected discounted value of the next state, assuming that the agent chooses the optimal action.*
- For every MDP, there exist an optimal deterministic policy which maximizes the expected, total discounted return from any initial state.

Solving MDPs: Value Iteration

Example 1: The grid world with immediate rewards for all transition (γ = 0.9). *G* (s6) is the absorbing state. To apply value iteration, assume all initial $V_0(s) = 0$.



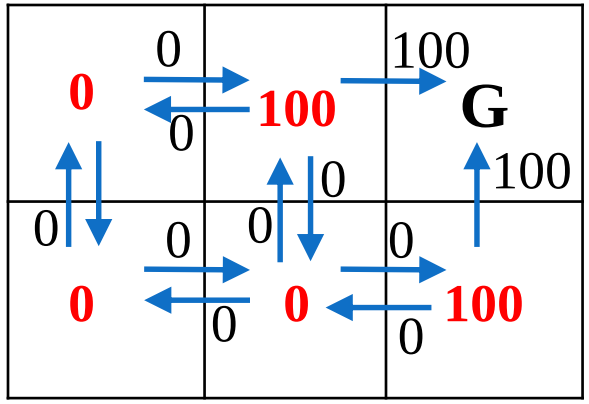
r (*s*, *a*) (immediate reward) values

The iterated value of s2 can be calculated by

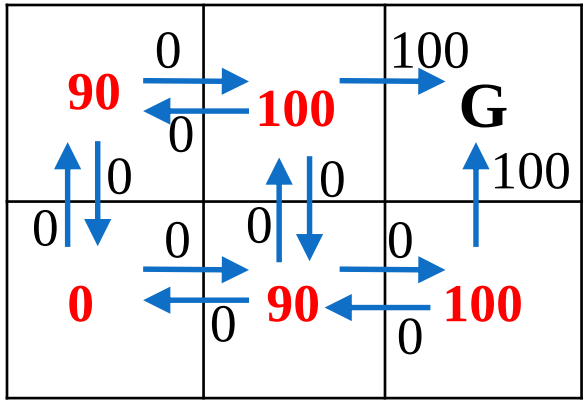
$$V_{n+1}(s2) = \text{Max}\left\{100 + \gamma V_n(G), 0 + \gamma V_n(s1), 0 + \gamma V_n(s4)\right\}$$

$$= \text{Max}\left\{100, 0, 0\right\} = 100$$

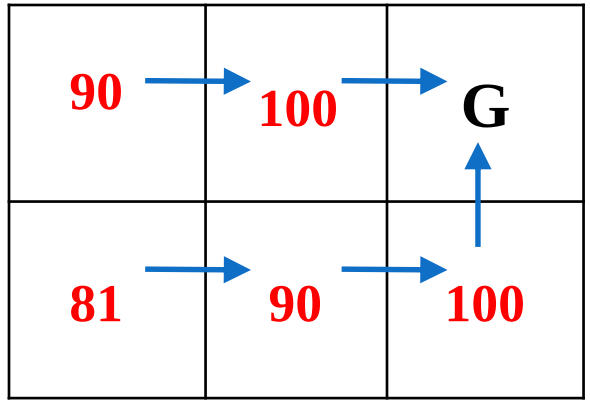
Thereby, following are the corresponding value table.



V_1



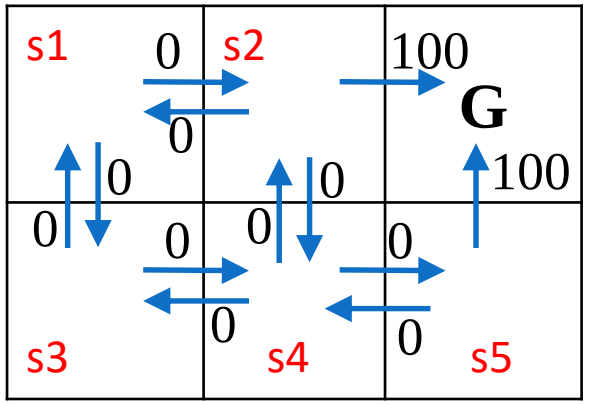
V_2



$V^* = V_3$ with *Policy*

Solving MDPs: Value Iteration

Example 1: The grid world with immediate rewards for all transition (γ = 0.9). G (s6) is the absorbing state. To apply value iteration, assume all initial $V_0(s) = 0$.

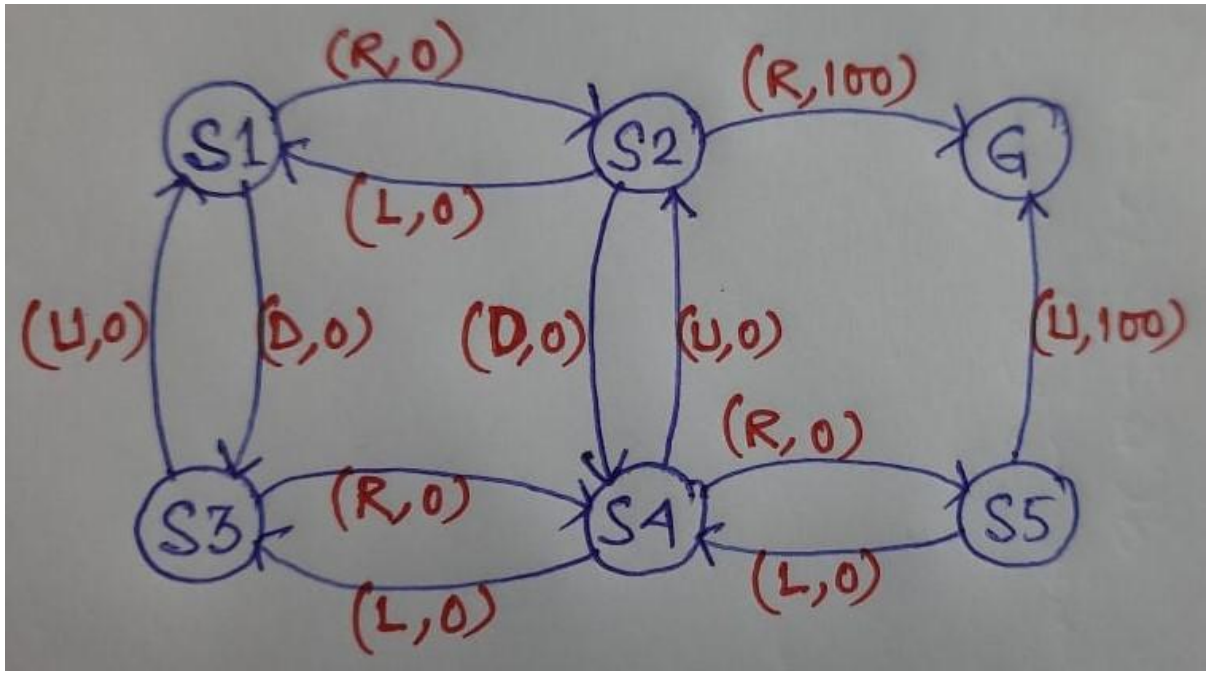


r (s, a) (immediate reward) values

- Here each transition is labeled by a pair [a, r], where a is the action and r is the reward.
- In this MDP, all transition and reward probabilities are either 0 or 1.

MDP for the same can be drawn as follow:

- The set of states, $S = \{s1, s2, s3, s4, s5, G\}$.
- The action set is $A = \{L, R, U, D\}$.



Solving MDPs: Value Iteration

- The optimal action in state *s* is the action *a* that maximizes the sum of the immediate reward *r*(*s*, *a*) plus the value V* of the immediate successor state, discounted by γ.

$$V^*(s_t) = \text{Max}_{\forall a_t}[r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + ...]$$
$$= \text{Max}_{\forall a_t}[r(s_t, a_t) + \gamma V^*(s_{t+1})]$$
$$V^*(s) = \text{Max}_{\forall a}[r(s, a) + \gamma V^*(\delta(s, a))]$$

- The optimal policy $\pi^*(s)$ carries out an action in state *s* which results in the maximum value V*. Thus,

$$\pi^*(s) = \text{argmax}_{\forall a}[r(s, a) + \gamma V^*(\delta(s, a))]$$

- Thus, a state-forward recursive rule of the problem is:

$$V(s) = \text{Max}_{\forall a}[r(s, a) + \gamma V(\delta(s, a))]$$

Solving MDPs: Value Iteration

- To begin, the V(s) for all states are initialized with zero or any arbitrary value.
- Now V(s) is repeatedly updated for each state by recursively falling back on the value V($\delta(s, a)$) of the best successor state.

Algorithm Value Iteration

1: **for** each $s \in S$
2: $V(s) = 0$ or any arbitary value.
3: **end for**
4: **repeat**
5: **for** each $s \in S$
6: $V(s) = \text{Max}_{\forall a}[r(s,a) + \gamma V(\delta(s,a))]$ // Update the value
7: **end for**
8: **until** none of the $V(s)$ changes
9: $\pi^*(s) = \text{argmax}_{\forall a}[r(s,a) + \gamma V(\delta(s,a))], \forall s \in S$ // The optimal policy for all states

Solving MDPs: Value Iteration

- Till now, we have considered the scenarios with deterministic policies and transition, i.e., all transition probabilities are either 0 or 1.

- Accordingly, following equation is applied:

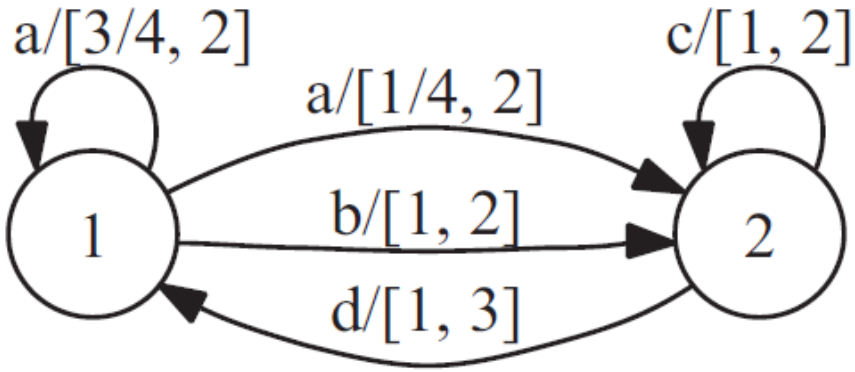
$$V^*(s) = \text{Max}_{\forall a}[r(s,a) + \gamma V^*(\delta(s,a))]$$

- If probabilities are involved then the Bellman equation can be stated as:

$$V^*(s) = \text{Max}_{\forall a}\left\{\mathbb{E}[r(s,a)] + \gamma \sum_{s' \in S} P(s'|s,a)V^*(s')\right\}$$

Let us take one example!!

Solving MDPs: Value Iteration



Consider the pair [*p*, *r*]:

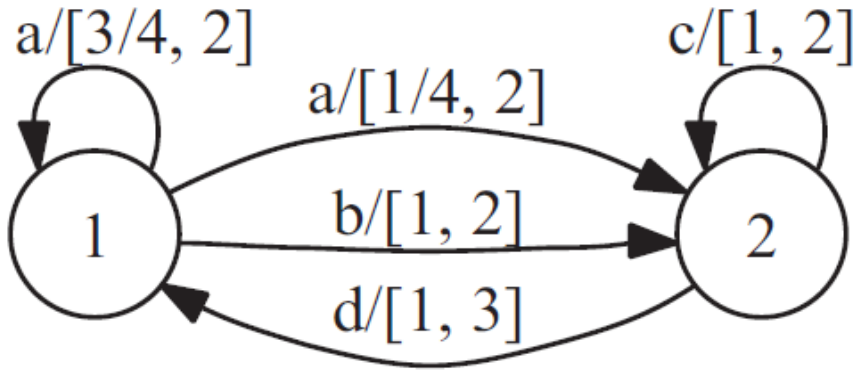
- *p* is the probability of the transition and
- *r* the expected reward for taking that transition.

Example 2: An MDP with two states.

- The state set is $S = \{1, 2\}$
- The action set is $A = \{a, b, c, d\}$.
- Only transitions with non-zero probabilities are represented.
- Each transition is labeled as: ***a*** / [*p*, *r*], where ***a*** is the action.
- Now, we have to use the Bellman equation:

$$V^*(s) = \text{Max}_{\forall a}\left\{\mathbb{E}[r(s,a)] + \gamma \sum_{s' \in S} P(s'|s,a)V^*(s')\right\}$$

Solving MDPs: Value Iteration



The iterated values of these states calculated by the algorithm for that MDP are:

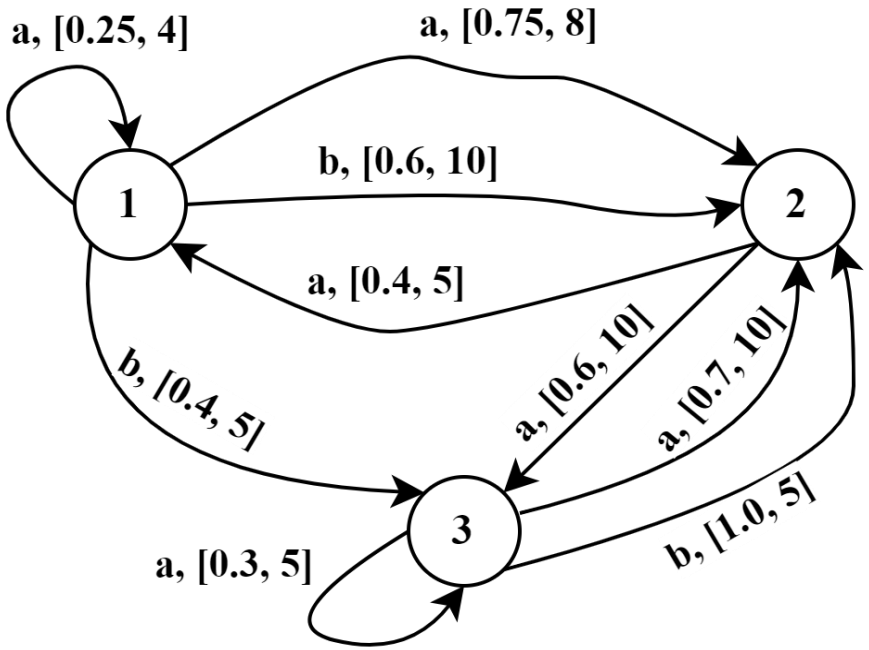
$$V_{n+1}(1) = \text{Max}\left\{2 + \gamma\left(\tfrac{1}{4}V_n(2) + \tfrac{3}{4}V_n(1)\right), 2 + \gamma V_n(2)\right\}$$

$$V_{n+1}(2) = \text{Max}\left\{2 + \gamma V_n(2), 3 + \gamma V_n(1)\right\}$$

- If we initially start with $V_0(1) = -1$, $V_0(2) = 1$, and $\gamma = 1/2$, we obtain $V_1(1) = V_1(2) = 5/2$. Thus, both states seem to have the same policy value initially.
- However, by the fifth iteration, $V_5(1) = 4.53125$, $V_5(2) = 5.15625$
- The algorithm quickly converges to the optimal values $V^*(1) = 14/3$ and $V^*(2) = 16/3$ showing that state 2 has a higher optimal value.

Solving MDPs: Value Iteration

***Example* 3**: Consider the following environment with three states. The state set is $S = \{1, 2, 3\}$ and the action set is A = {a, b}. Only transitions with non-zero probabilities are represented. Each transition is labelled with the action taken followed by a pair [p, r], where p is the probability of the transition and r the reward for taking that transition. To apply value iteration algorithm:



a) Build the formulas to update the value of all the three states. Given $\gamma = 0.9$.

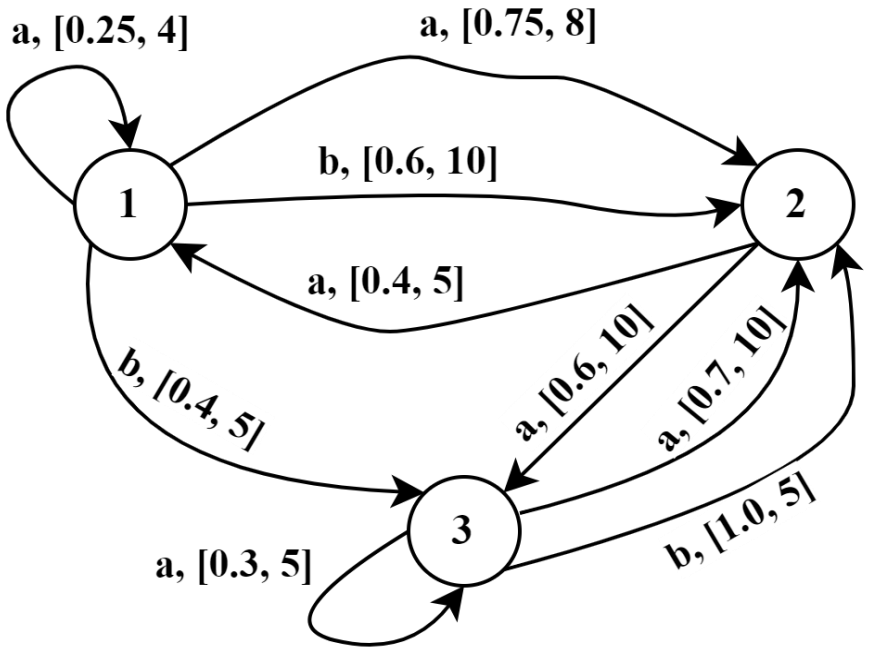
b) Use the above formulas to update the value of all the three states for a single episode only.

Solving MDPs: Value Iteration

***Example* 3**:

a) Build the formulas to update the value of all the three states. Given $\gamma = 0.9$.

b) Use the above formulas to update the value of all the three states for a single episode only.



Solution: (a)

$$V_{n+1}(1) = \text{Max}\Big\{7 + \gamma\Big(0.25V_n(1) + 0.75V_n(2)\Big), \; 8 + \gamma\Big(0.6V_n(2) + 0.4V_n(3)\Big)\Big\}$$

$$V_{n+1}(2) = \text{Max}\Big\{8 + \gamma\Big(0.6V_n(3) + 0.4V_n(1)\Big)\Big\}$$

$$V_{n+1}(3) = \text{Max}\Big\{8.5 + \gamma\Big(0.3V_n(3) + 0.7V_n(2)\Big), 5 + \gamma\Big(V_n(2)\Big)\Big\}$$

(b) Assume, $V_0(1) = 0$, $V_0(2) = 0$, and $V_0(3) = 0$.

Therefore, $V_1(1) = \text{Max}\{7, 8\} = 8$, $V_1(2) = \text{Max}\{8\} = 8$, and $V_1(3) = \text{Max}\{8.5, 5\} = 8.5$

Solving MDPs: Policy Iteration

- The **policy iteration** algorithm alternates the following two steps, beginning from some initial policy π_0:
  1) **Policy evaluation**: given a policy π, calculate V(s), the value of each state by following the policy π.

$$V_{n+1}(s) = r(s, \pi_n(s)) + \gamma V_n(\delta(s, \pi_n(s))), \forall s \in S$$

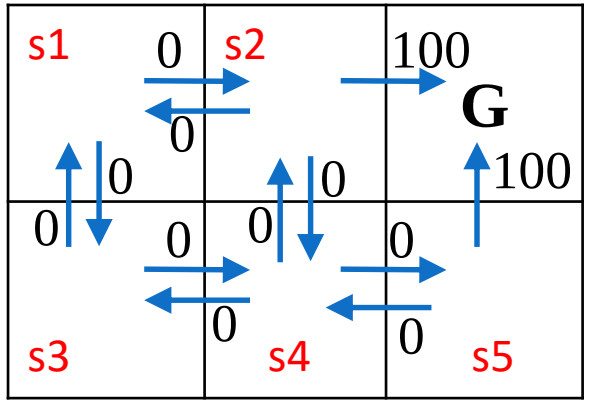
  2) **Policy improvement**: Calculate a new policy π_{i+1}, using one-step look-ahead based on V(s).

$$\pi_n(s) = \text{argmax}_{\forall a}[r(s, a) + \gamma V_n(\delta(s, a))], \forall s \in S$$

- The algorithm terminates when the policy improvement step yields no change in the values.

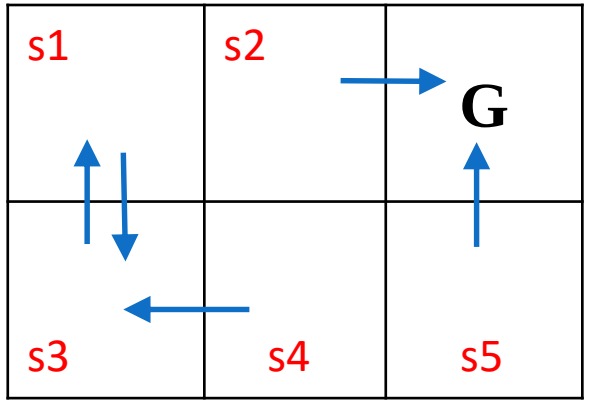
Solving MDPs: Policy Iteration

Example 4: The grid world with immediate rewards for all transition (γ = 0.9). G (s6) is the absorbing state. To apply policy iteration.



r (s, a) (immediate reward) values

➢ Arbitrarily assign all $V_0(s) = 0$ and take the initial policy π_0 as:



An arbitrary initial *Policy* π_0

➢ Policy evaluation: by following π_0 using the equation.

$$V_{n+1}(s) = r(s, \pi_n(s)) + \gamma V_n(\delta(s, \pi_n(s)))$$

➢ Therefore, $V_1(s_1) = r(s_1, \pi_0(s_1)) + \gamma V_0(\delta(s_1, \pi_0(s_1)))$

$$= r(s_1, D) + \gamma V_0(s_3) = 0 + \gamma 0 = 0$$

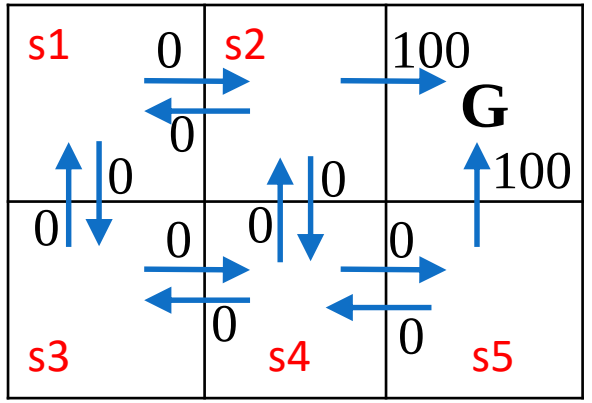
$$V_1(s_2) = r(s_2, \pi_0(s_2)) + \gamma V_0(\delta(s_2, \pi_0(s_2))) = r(s_2, R) + \gamma V_0(s_6) = 100 + \gamma 0 = 100$$

$$V_1(s_3) = r(s_3, U) + \gamma V_0(s_1) = 0 + \gamma 0 = 0$$

$$V_1(s_4) = r(s_4, L) + \gamma V_0(s_3) = 0 \qquad V_1(s_5) = r(s_5, U) + \gamma V_0(s_6) = 100$$

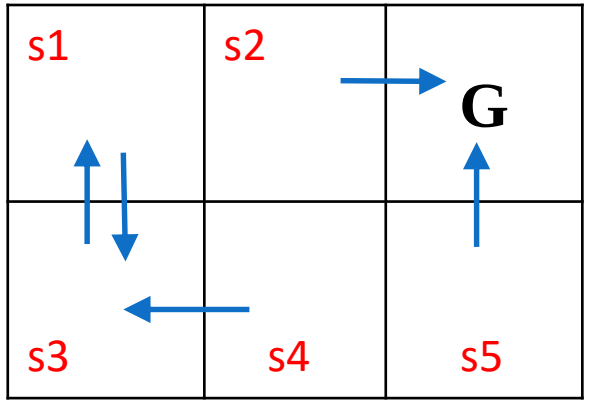
Solving MDPs: Policy Iteration

Example 4: The grid world with immediate rewards for all transition (γ = 0.9). *G* (s6) is the absorbing state. To apply policy iteration.



r (*s*, *a*) (immediate reward) values

- Arbitrarily assign all $V_0(s) = 0$ and take the initial policy π_0 as:

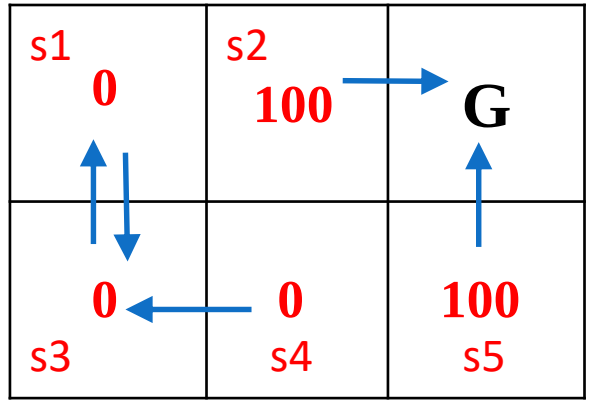


An arbitrary initial *Policy* π_0

- Policy evaluation: by following π_0 using the equation.

$$V_{n+1}(s) = r(s, \pi_n(s)) + \gamma V_n(\delta(s, \pi_n(s)))$$

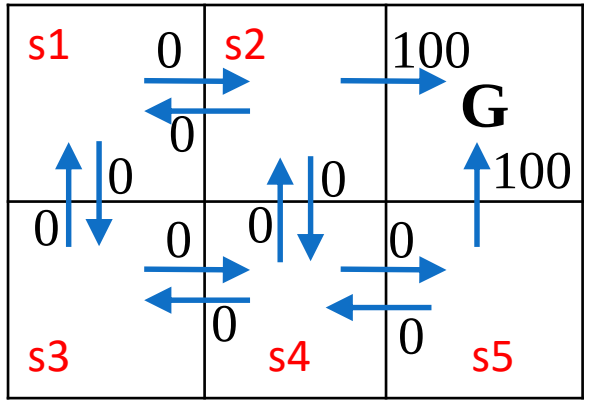
- Now, we have final V_1 after repeatedly evolution of all the states until it is converged.



V_1 by following policy π_0

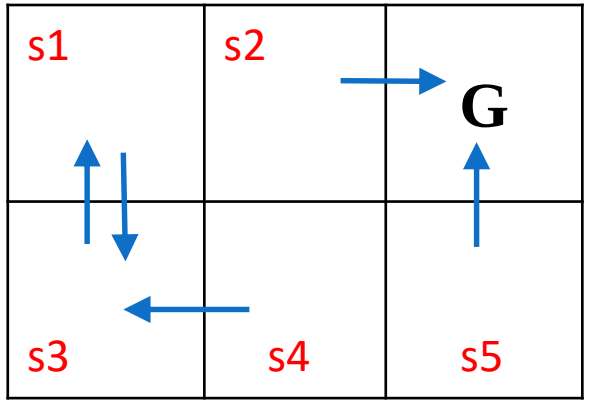
Solving MDPs: Policy Iteration

Example 4: The grid world with immediate rewards for all transition (γ = 0.9). G (s6) is the absorbing state. To apply policy iteration.



r (*s*, *a*) (immediate reward) values

➢ Arbitrarily assign all $V_0(s) = 0$ and take the initial policy π_0 as:

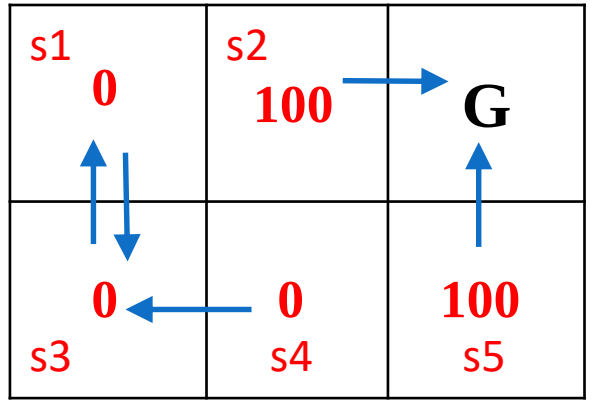


An arbitrary initial *Policy* π_0

➢ Policy evaluation: by following π_0 using the equation.

$$V_{n+1}(s) = r(s, \pi_n(s)) + \gamma V_n(\delta(s, \pi_n(s)))$$

➢ Now, we have final V_1 after repeatedly evolution of all the states until it is converged.



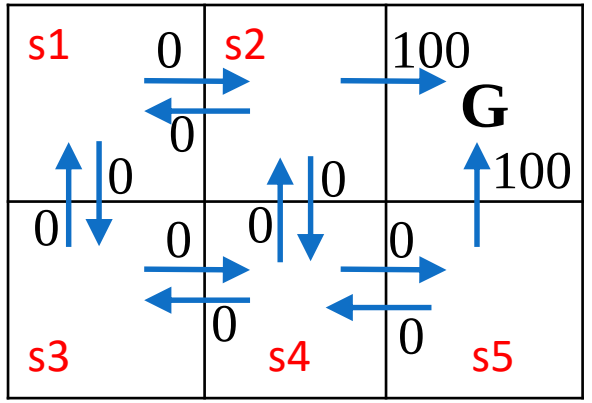
V_1 by following policy π_0

➢ Next, we have improve the policy (π_1) based on V_1 using:

$$\pi_n(s) = \text{argmax}_{\forall a}[r(s, a) + \gamma V_n(\delta(s, a))]$$

Solving MDPs: Policy Iteration

Example 4: The grid world with immediate rewards for all transition (γ = 0.9). G (s6) is the absorbing state. To apply policy iteration.



r (*s*, *a*) (immediate reward) values

- Policy improvement: using the following equation.

$$\pi_n(s) = \text{argmax}_{\forall a}[r(s,a) + \gamma V_n(\delta(s,a))]$$

- Therefore, $\pi_1(s_1) = \text{argmax}_{\{R,D\}}[r(s_1,R) + \gamma V_1(s_2), r(s_1,D) + \gamma V_1(s_3)]$

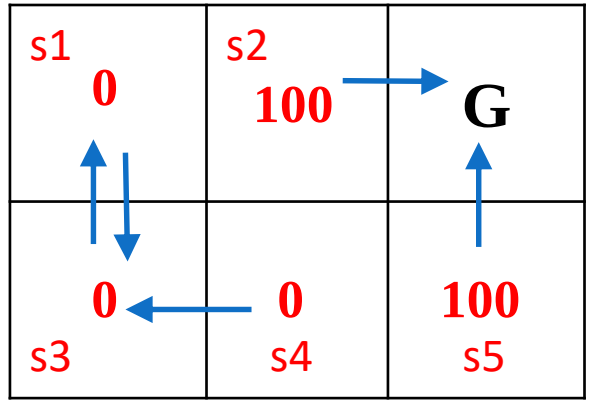
$$= \text{argmax}_{\{R,D\}}[0 + \gamma 100, 0 + \gamma 0] = R$$

$$\pi(s_2) = \text{argmax}_{\{L,R,D\}}[0 + \gamma 0, 0 + \gamma 100, 0 + \gamma 0] = R$$

$$\pi(s_3) = \text{argmax}_{\{R,U\}}[0 + \gamma 0, 0 + \gamma 0] = U \text{ or } R$$

$$\pi(s_4) = \text{argmax}_{\{L,R,U\}}[0 + \gamma 0, 0 + \gamma 100, 0 + \gamma 100] = R \text{ or } U$$

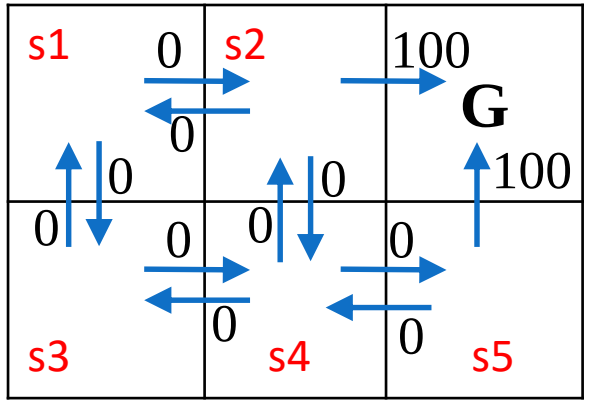
$$\pi(s_5) = \text{argmax}_{\{L,U\}}[0 + \gamma 0, 0 + \gamma 100] = U$$



V_1 by following policy π_0

Solving MDPs: Policy Iteration

Example 4: The grid world with immediate rewards for all transition (γ = 0.9). G (s6) is the absorbing state. To apply policy iteration.



r (*s*, *a*) (immediate reward) values

- Policy improvement: using the following equation.

$$\pi_n(s) = \text{argmax}_{\forall a}[r(s,a) + \gamma V_n(\delta(s,a))]$$

- Therefore, $\pi_1(s_1) = \text{argmax}_{\{R,D\}}[r(s_1,R) + \gamma V_1(s_2), r(s_1,D) + \gamma V_1(s_3)]$

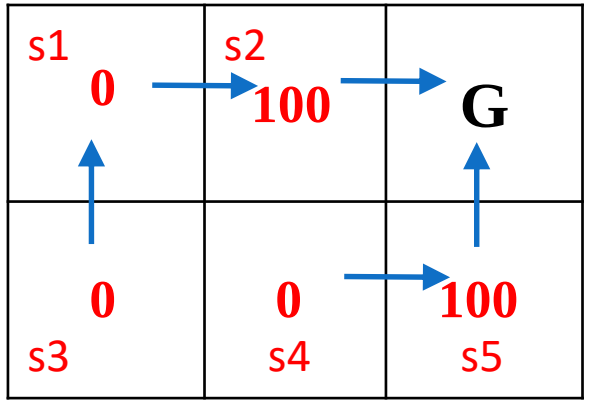
$$= \text{argmax}_{\{R,D\}}[0 + \gamma 100, 0 + \gamma 0] = R$$

$$\pi(s_2) = \text{argmax}_{\{L,R,D\}}[0 + \gamma 0, 0 + \gamma 100, 0 + \gamma 0] = R$$

$$\pi(s_3) = \text{argmax}_{\{R,U\}}[0 + \gamma 0, 0 + \gamma 0] = U \text{ or } R$$

$$\pi(s_4) = \text{argmax}_{\{L,R,U\}}[0 + \gamma 0, 0 + \gamma 100, 0 + \gamma 100] = R \text{ or } U$$

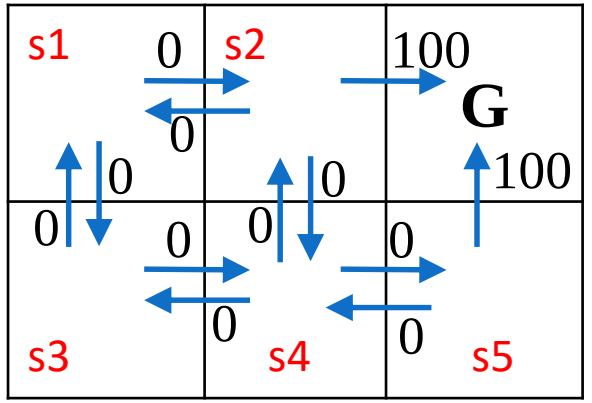
$$\pi(s_5) = \text{argmax}_{\{L,U\}}[0 + \gamma 0, 0 + \gamma 100] = U$$



V_1 and updated policy π_1

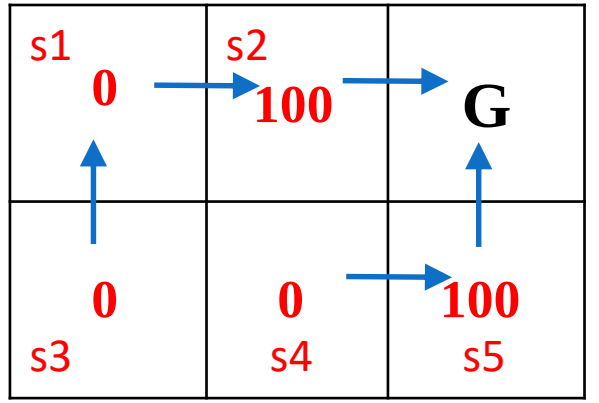
Solving MDPs: Policy Iteration

Example 4: The grid world with immediate rewards for all transition (γ = 0.9). *G* (s6) is the absorbing state. To apply policy iteration.



r (*s*, *a*) (immediate reward) values

- Next, we have to again apply the Policy evaluation to find the V_2 following the current policy π_1.
- Then apply the Policy improvement to find out the updated the policy π_2.
- Repeat these two phases until no update in the policy.



V_1 and updated policy π_1

Solving MDPs: Policy Iteration

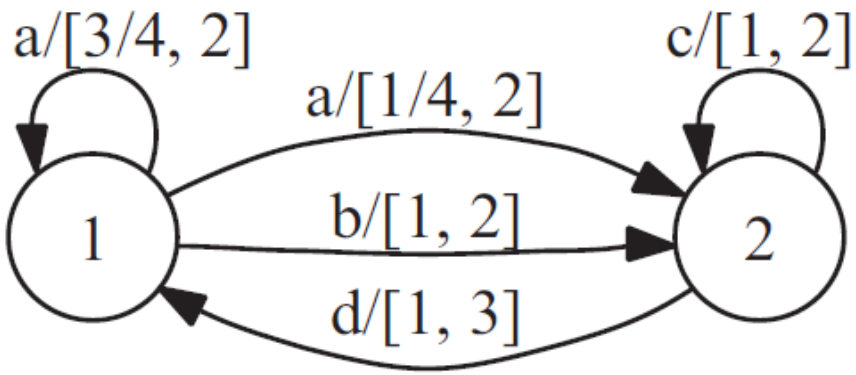
Algorithm Policy Iteration

1: Initialize $V(s) \in \mathbb{R}$ and $\pi(s) \in A$ arbitrarily for all $s \in S$.
2: $Not\_Stable = true$ // Initially, policy is not stable
3: **while** (Not_Stable) **do**
4: **repeat**
5: **for** each $s \in S$
6: $V(s) = r(s, \pi(s)) + \gamma V(\delta(s, \pi(s))$
7: **end for**
8: **until** none of the $V(s)$ changes
9: $Not\_Stable = false$
10: **for** each $s \in S$
11: $old\_action = \pi(s)$
12: $\pi(s) = \text{argmax}_{\forall a}[r(s, a) + \gamma V(\delta(s, a))]$ // Update policy
13: **if**$(old\_action \neq \pi(s))$ **then** $Not\_Stable = true$
14: **end for**
15: **end while**
16: Return $\pi(s)$, and $V(s)$ $\forall s \in S$

Policy Evaluation

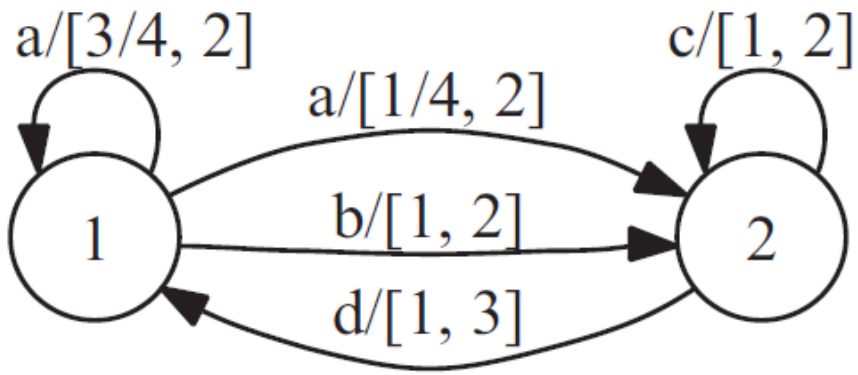
Policy Improvement

Solving MDPs: Policy Iteration



Example 5: Given MDP with two states, $S = \{1, 2\}$ and the action set, $A = \{a, b, c, d\}$. Apply policy iteration algorithm ($\gamma = 0.9$).

Solving MDPs: Policy Iteration



Example 5: Given MDP with two states, $S = \{1, 2\}$ and the action set, $A = \{a, b, c, d\}$. Apply policy iteration algorithm ($\gamma = 0.9$).

- It is a stochastic model. The equations for **Policy evaluation** and **Policy improvement** are:

❖ **Policy evaluation:**

$$V_{n+1}(s) = \sum_{s' \in S} P(s'|s, \pi_n(s)).\left[r(s, \pi_n(s)) + \gamma V_n(s')\right]$$

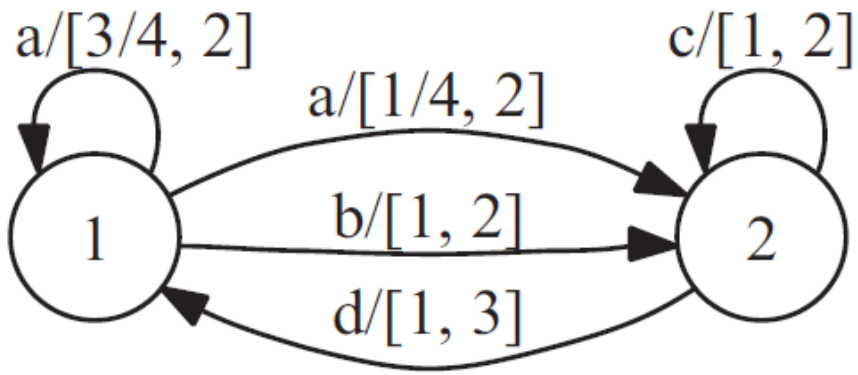
$$= \sum_{s' \in S} P(s'|s, \pi_n(s)).r(s, \pi_n(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_n(s)).V_n(s')$$

$$= \mathbb{E}[r(s, \pi_n(s))] + \gamma \sum_{s' \in S} P(s'|s, \pi_n(s))V_n(s')$$

❖ **Policy improvement:**

$$\pi_n(s) = \text{argmax}_{\forall a}\left\{\mathbb{E}[r(s, a)] + \gamma \sum_{s' \in S} P(s'|s, a)V_n(s')\right\}$$

Solving MDPs: Policy Iteration



Example 5: Given MDP with two states, $S = \{1, 2\}$ and the action set, $A = \{a, b, c, d\}$. Apply policy iteration algorithm ($\gamma = 0.9$).

- Let us arbitrarily assign all $V_0(s) = 0$ and the initial policy $\pi_0(1) = a$ and $\pi_0(2) = d$.

❖ **Policy evaluation:** the system of linear equations for evaluating this policy is

$$V_{n+1}(1) = 2 + \gamma\left(\tfrac{1}{4}V_n(2) + \tfrac{3}{4}V_n(1)\right) \qquad V_{n+1}(2) = 3 + \gamma V_n(1)$$

Therefore, $V_1(1) = 2$ and $V_1(2) = 3$.

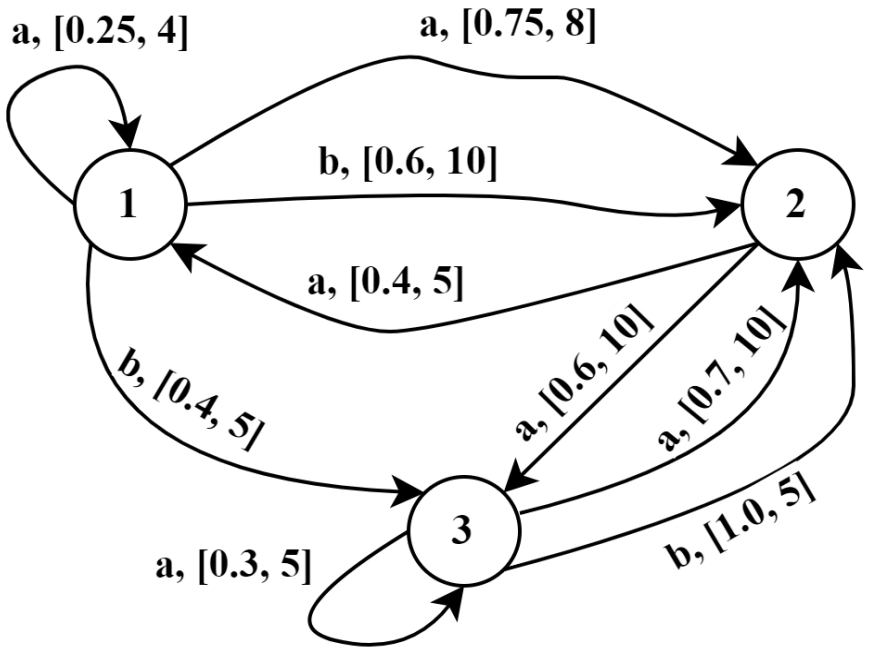
❖ **Policy improvement:**

$$\pi_1(1) = \text{argmax}_{\{a,b\}}\left\{2 + \gamma\left(\tfrac{1}{4}V_1(2) + \tfrac{3}{4}V_1(1)\right), 2 + \gamma V_1(2)\right\} = b$$

$$\pi_1(2) = \text{argmax}_{\{c,d\}}\left\{2 + \gamma V_1(2), 3 + \gamma V_1(1)\right\} = d$$

Solving MDPs: Policy Iteration

***Example* 6**: Consider the following environment with three states. The state set is $S = \{1, 2, 3\}$ and the action set is A = {a, b}. Only transitions with non-zero probabilities are represented. Each transition is labelled with the action taken followed by a pair [p, r], where p is the probability of the transition and r the reward for taking that transition. To apply **policy iteration** algorithm:



a) Build the formulas to update the value of all the three states. Given $\gamma = 0.9$.

b) Use the above formulas to update the value of all the three states for a single episode only.

Solving MDPs: Policy Iteration

- Note that two consecutive policy values can be equal only at the last iteration of the algorithm.
- The total number of possible policies is $|A|^{|S|}$, thus this constitutes a straightforward upper bound on the maximal number of iterations.
- The policy iteration algorithm converges in a smaller number of iterations than the value iteration algorithm due to the optimal policy.
- But, each iteration of the policy iteration algorithm requires computing a policy value, that is, solving a system of linear equations, which is more expensive to compute that an iteration of the value iteration algorithm.

Solving MDPs: Q Learning

Q Learning:

- The agent should prefer state s_1 over state s_2 whenever $V^*(s_1) > V^*(s_2)$, because the cumulative future reward will be greater from s_1.
- *Q*-function: The value of *Q* is the reward received immediately upon executing action *a* from state *s*, plus the value (discounted by γ) of following the optimal policy thereafter.

$$Q(s,a) = Q(s,a) + \alpha[r(s,a) + \gamma \mathrm{Max}_{\forall a'}(Q(s',a')) - Q(s,a)]$$

- Where, $\alpha \in (0,1]$ is an appropriate learning rate which controls the contribution of the new experience to the current estimate.
- The agent repeatedly observes its current state *s*, chooses some action *a*, executes this action, then observes the resulting reward *r*(*s*, *a*) and the new state *s*' = δ(*s*, *a*). It then updates the table entry for *Q*(*s*,*a*) following each such transition, according to the rule:

The optimal policy is determined as: $\pi^*(s) = \mathrm{argmax}_a Q(s,a)$

Solving MDPs: Q Learning

Q Learning:

- Learning the *Q*-function corresponds to learning the optimal policy. How can *Q* be learned?
- *Q*-learning starts with an initial estimate for each state-action pair.
- When an action a is taken in state *s*, resulting in the next state *s'*, the corresponding *Q*-value *Q*(*s*, *a*) is updated with a combination of its current value and the temporal difference (TD) error:

$$TD = [r(s,a) + \gamma \mathrm{Max}_{\forall a'}(Q(s',a')) - Q(s,a)]$$

- The TD error is the difference between the current estimate *Q*(*s*, *a*) and the expected discounted return based on the experienced sample.
- The Q value of each state-action pair is stored in a table for a discrete state-action space.
- It has been proved that this tabular Q-learning converges to the optimal *Q*(*s*, *a*) with probability 1 when all state-action pairs are visited infinitely often and an appropriate exploration strategy and learning rate are chosen.

Solving MDPs: Q Learning

Q Learning:

- Learning the *Q*-function corresponds to learning the optimal policy. How can *Q* be learned?

Algorithm Q-Learning

1: For each (s, a) initialize the table entry $Q(s, a)$ arbitrarily or with zero.
2: Observe the current state s.
3: **for** each $t = 0$ to T
4: Select a state s.
5: **while** (until reaches to goal state) **do**
6: Choose an action a and execute it.
7: Receive immediate reward r
8: Observe the new state s'
9: $Q(s, a) = Q(s, a) + \alpha[r(s, a) + \gamma \text{Max}_{\forall a'}(Q(s', a')) - Q(s, a)]$
10: $s = s'$
11: **end while**
12: **end for**

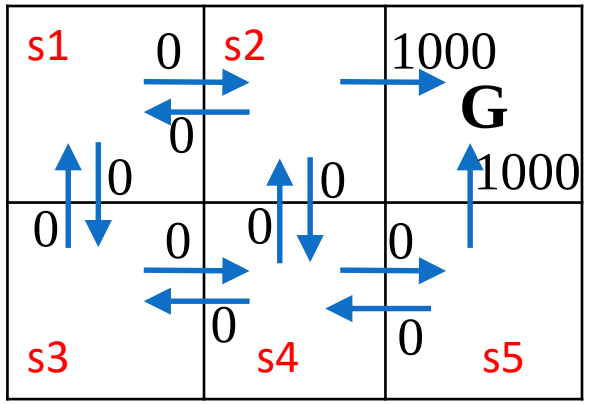
Solving MDPs: Q Learning

Example 6: The initial Q-table. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	0	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	0	X

Episode 1

1. Consider an arbitrary state (say *s*2).
2. Apply an action and update the Q value.
3. Continue with the next state until reaches to the goal state.



r (*s*, *a*) (immediate reward) values

- Take the action *Right*. Updated Q-value for *Q*(s2, *Right*).

$$Q(s_2, R) = Q(s_2, R) + \alpha[r(s_2, R) + \gamma \text{Max}_{\forall a'}(Q(s', a')) - Q(s_2, R)]$$

$$= 0 + 0.1[1000 + 0.9 \times \text{Max}(0, 0) - 0] = 100$$

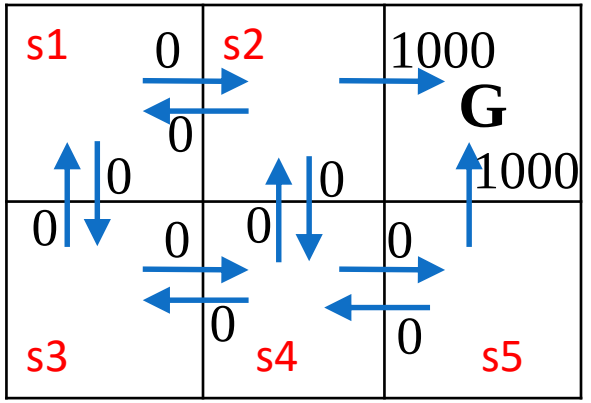
- Update Q-table accordingly. Next state is the goal state (after right move from s2).

Solving MDPs: Q Learning

Example 6: The initial Q-table. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	**100**	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	0	X

Episode 2



r (*s*, *a*) (immediate reward) values

Consider state *s*5:

- Take the action *Up*. Updated Q-value for *Q*(s5, *Up*).

$$Q(s_5, U) = Q(s_5, U) + \alpha[r(s_5, U) + \gamma \text{Max}_{\forall a'}(Q(s', a')) - Q(s_5, U)]$$

$$= 0 + 0.1[1000 + 0.9 \times \text{Max}(0, 0) - 0] = 100$$

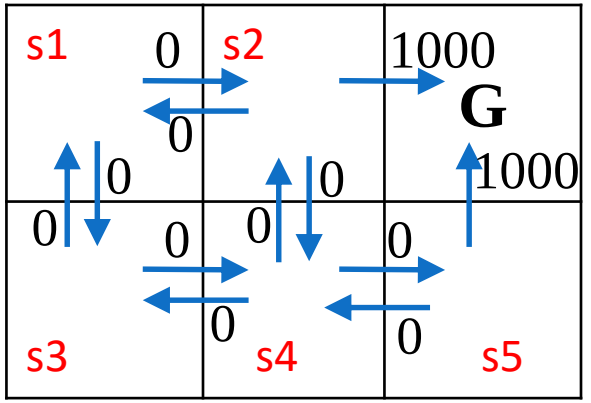
- Update Q-table accordingly. Next state is the goal state (after up move from s5).

Solving MDPs: Q Learning

Example 6: The initial Q-table. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	**100**	X

Episode 3



r (*s*, *a*) (immediate reward) values

Consider state *s*1:

- Take the action *Right*. Updated Q-value for *Q*(s1, *Right*).

$$Q(s_1, R) = Q(s_1, R) + \alpha[r(s_1, R) + \gamma \text{Max}_{\forall a'}(Q(s', a')) - Q(s_1, R)]$$

$$= 0 + 0.1[0 + 0.9 \times \text{Max}(0, 100, 0) - 0] = 9$$

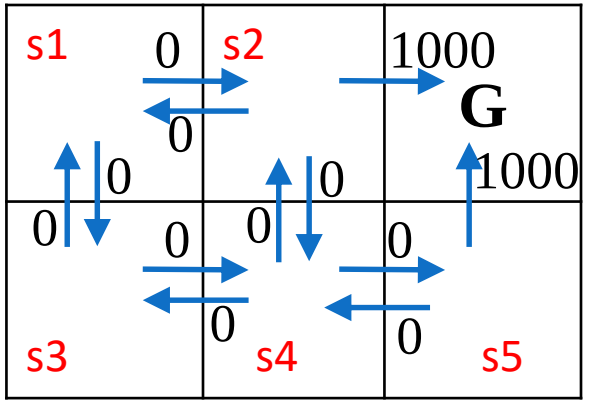
- Update Q-table. Next state is s2 (after right move from s1).

Solving MDPs: Q Learning

Example 6: The initial Q-table. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	**9**	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	100	X

Episode 3



r (*s*, *a*) (immediate reward) values

Next state is *s*2:

- Take the action *Down*. Let us now update *Q*(s2, *Down*).

$$Q(s_2, D) = Q(s_2, D) + \alpha[r(s_2, D) + \gamma \text{Max}_{\forall a'}(Q(s', a')) - Q(s_2, D)]$$

$$= 0 + 0.1[0 + 0.9 \times \text{Max}(0, 0, 0) - 0] = 0$$

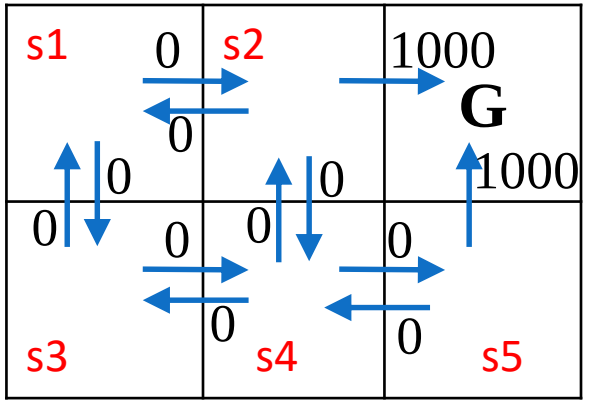
- Update Q-table. Next state is s4 (after down move from s2).

Solving MDPs: Q Learning

Example 6: The initial Q-table. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	9	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	100	X

Episode 3



r (*s*, *a*) (immediate reward) values

Next state is *s*4:

- Take the action *Right*. Let us now update *Q*(s4, *Right*).

$$Q(s_4, R) = Q(s_4, R) + \alpha[r(s_4, R) + \gamma \text{Max}_{\forall a'}(Q(s', a')) - Q(s_4, R)]$$
$$= 0 + 0.1[0 + 0.9 \times \text{Max}(0, 100) - 0] = 9$$

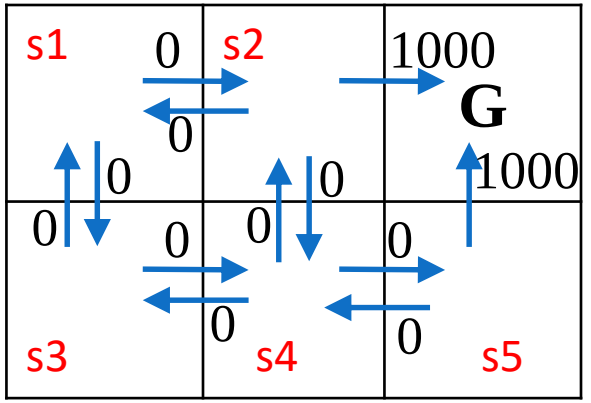
- Update Q-table. Next state is s5 (after right move from s4).

Solving MDPs: Q Learning

Example 6: The initial Q-table. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	9	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	**9**	0	X
S5	0	X	100	X

Episode 3



r (*s*, *a*) (immediate reward) values

Next state is *s*5:

- Take the action *Up*. Let us now update *Q*(s5, *Up*).

$$Q(s_5, U) = Q(s_5, U) + \alpha[r(s_5, U) + \gamma \text{Max}_{\forall a'}(Q(s', a')) - Q(s_5, U)]$$

$$= 100 + 0.1[1000 + 0.9 \times \text{Max}(0, 0) - 100] = 190$$

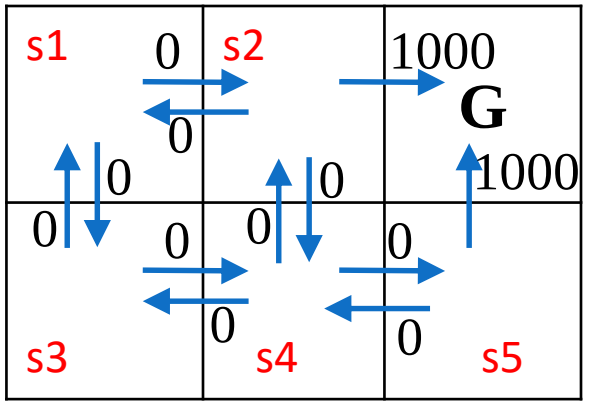
- Update Q-table. Next state is Goal state (after Up move from s5).

Solving MDPs: Q Learning

Example 6: The initial Q-table. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	9	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	9	0	X
S5	0	X	**190**	X

Episode 4



r (*s*, *a*) (immediate reward) values

- Now, consider state *s*3.
- Continue taking action until reaches to Goal state.

➢ Take the action *Up*. Let us now update *Q*(s3, *Up*).

➢ Continue the process till it converges.

➢ The optimal policy is $\pi^*(s) = \text{argmax}_a Q(s, a)$

➢ *This is an example of deterministic model. What about the stochastic model?*

Solving MDPs: Q Learning

- Assume that training consists of a series of ***episodes.***
- During each episode, the agent begins at some randomly chosen state and is allowed to execute actions until it reaches the absorbing goal state.
- When it does, the episode ends and the agent is transported to a new, randomly chosen, initial state for the next episode.
- Given a sufficient number of training episodes, the information will propagate from the transitions with nonzero reward back through the entire state-action space available to the agent.
- The above approach is same for all the Reinforcement learning algorithms.

Solving MDPs: Q Learning

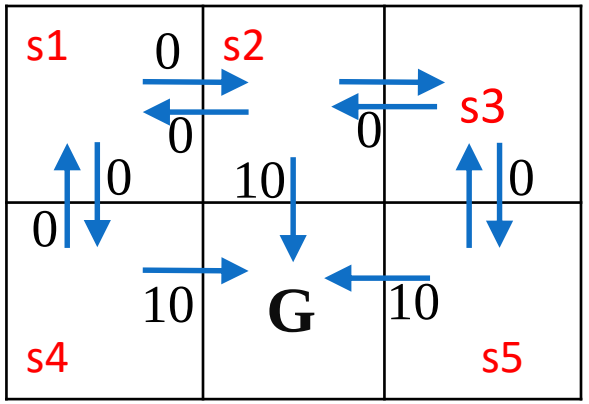
- The action selection of the Q-learning algorithm (line 6, slide 47) use a random selection.
- Therefore, it gives emphasize on *exploration*. However, we have make a balance or trade-off between *exploration* and *exploitation*.
- The ε-Greedy Algorithm makes use of the *exploration-exploitation tradeoff* by instructing the agent to
  - choose a random option with probability epsilon (ε) (i.e., explore) and
  - choose the option which seems to be the best (i.e., exploit).

- In the ε-Greedy algorithm, the action selection is determined as follows:

$$a_t = \begin{cases} \arg\max_a Q(s_t, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Solving MDPs: Q Learning

Example 7: Consider applying Q-learning algorithm to this grid world.



- Assume the table of Q values is initialized to zero.
- Assume the agent begins in the bottom left grid square and then travels clockwise around the perimeter of the grid until it reaches the absorbing goal state, completing the first training episode. Describe which Q values are modified as a result of this episode, and give their revised values.
- Answer the question again assuming the agent now performs a second identical episode. Answer it again for a third episode.

Example 8: Apply Value iteration and Policy iteration algorithm on the same grid world of the above example.

Solving MDPs: SARSA

- The Q-learning does not use policy to update the Q-value, $Q(s, a)$. This is why it is known as ***off-policy*** learning algorithm.
- Like Q-learning, SARSA is a TD algorithm but uses policy to update $Q(s,a)$. Therefore, SARSA is an ***on-policy*** learning algorithm.

- **SARSA** stands for **State-Action-Reward-State-Action**, an on-policy reinforcement learning algorithm.

- It reflects the sequence of events used in the algorithm to learn from an environment:
  - **State (S)**: The current state of the agent.
  - **Action (A)**: The action taken by the agent.
  - **Reward (R)**: The reward received after performing the action in the state.
  - **Next State (S')**: The new state the agent transitions to after taking the action.
  - **Next Action (A')**: The action chosen by the agent in the new state.

Solving MDPs: SARSA

Algorithm SARSA

1: For each (s, a) initialize the table entry $Q(s, a)$ arbitrarily or with zero.
2: **for** each $t = 0$ to T
S 3: Select a state s.
A 4: Choose an action a using policy based on Q-value and ϵ-greedy.
5: **while** (until reaches to goal state) **do**
6: Take action a.
R 7: Receive immediate reward r.
S 8: Observe the next state s'.
A 9: Choose an action a' using policy based on Q-value and ϵ-greedy.
10: $Q(s, a) = Q(s, a) + \alpha[r(s, a) + \gamma Q(s', a') - Q(s, a)]$
11: $s = s'$ and $a = a'$
12: **end while**
13: **end for**

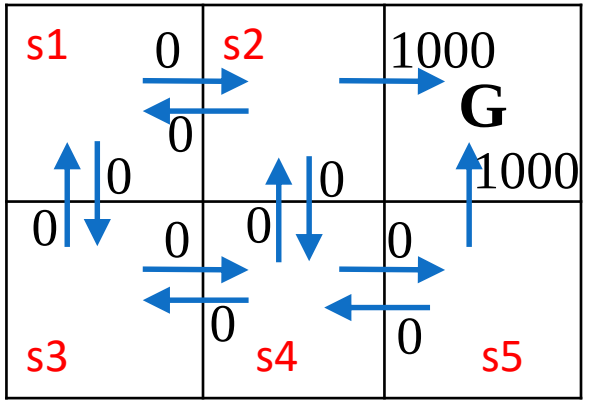
Solving MDPs: SARSA

Example 9: Apply SARSA. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	0	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	0	X

Episode 1

1. Select a state (say *s*2).
2. Choose an action (say *R*).
3. Reward, r(s2, *R*).
4. Next state, G.
5. Choose next action.



r (*s*, *a*) (immediate reward) values

- Take the action *Right*. Updated Q-value for *Q*(s2, *R*).

$$Q(s_2, R) = Q(s_2, R) + \alpha[r(s_2, R) + \gamma Q(G, .) - Q(s_2, R)]$$
$$= 0 + 0.1[1000 + 0.9 \times 0 - 0] = 100$$

- Stop the episode as the next state is terminal state.
- We have to initiate next episode.

- Update Q-table accordingly. Next state is the goal state (after right move from s2).

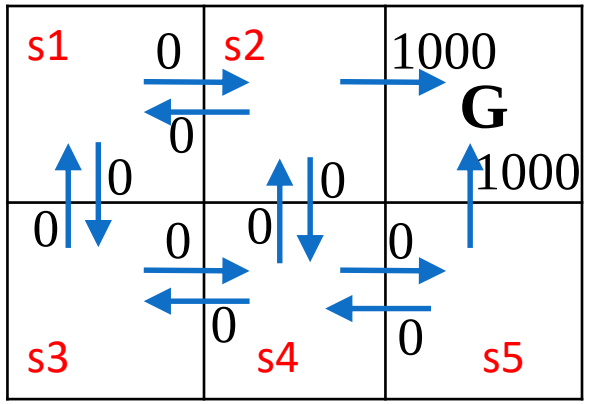
Solving MDPs: SARSA

Example 9: Apply SARSA. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	**100**	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	0	X

Episode 2

1. Select a state (say *s*1).
2. Choose an action (say *R*).
3. Reward, r(s1, *R*).
4. Next state, s2.
5. Choose next action (*D*).



r (*s*, *a*) (immediate reward) values

➢ Take the action *Right*. Updated Q-value for *Q*(s1, *R*).

$$Q(s_1, R) = Q(s_1, R) + \alpha[r(s_1, R) + \gamma Q(s_2, D) - Q(s_1, R)]$$

$$= 0 + 0.1[0 + 0.9 \times 0 - 0] = 0$$

For next iteration

- Next state is s2.
- Next action is D.
- Iterate it till the terminal state.

➢ Update Q-table accordingly. Next state is s2 (after right move from s1).

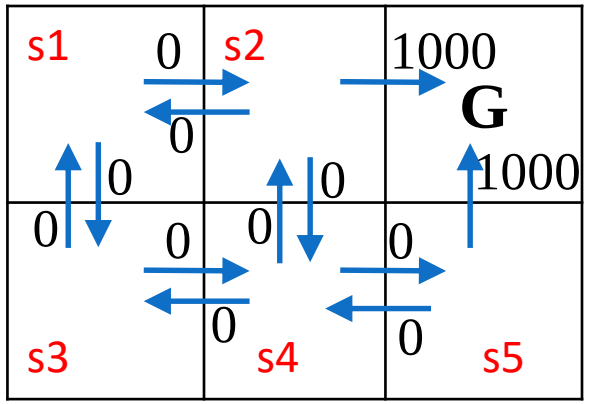
Solving MDPs: SARSA

Example 9: Apply SARSA. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	0	X

Episode 2

1. The state is *s*2.
2. The action is *D*.
3. Reward, r(s2, *D*).
4. Next state, s4.
5. Choose next action (*R*).



r (*s*, *a*) (immediate reward) values

➢ Take the action *Down*. Updated Q-value for *Q*(s2, *D*).

$$Q(s_2, D) = Q(s_2, D) + \alpha[r(s_2, D) + \gamma Q(s_4, R) - Q(s_2, D)]$$

$$= 0 + 0.1[0 + 0.9 \times 0 - 0] = 0$$

For next iteration

- Next state is s4.
- Next action is R.
- Iterate it till the terminal state.

➢ Update Q-table accordingly. Next state is s4 (after down move from s2).

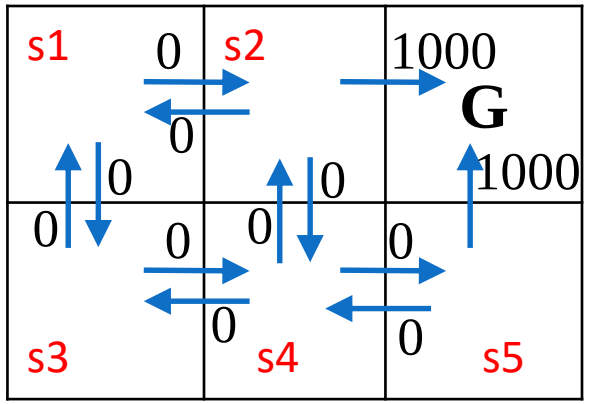
Solving MDPs: SARSA

Example 9: Apply SARSA. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	0	X

Episode 2

1. The state is *s*4.
2. The action is *R*.
3. Reward, r(s4, *R*).
4. Next state, s5.
5. Choose next action (*U*).



r (*s*, *a*) (immediate reward) values

➢ Take the action *Right*. Updated Q-value for *Q*(s4, *R*).

$$Q(s_4, R) = Q(s_4, R) + \alpha[r(s_4, R) + \gamma Q(s_5, U) - Q(s_4, R)]$$

$$= 0 + 0.1[0 + 0.9 \times 0 - 0] = 0$$

For next iteration

- Next state is s5.
- Next action is U.
- Iterate it till the terminal state.

➢ Update Q-table accordingly. Next state is s5 (after right move from s4).

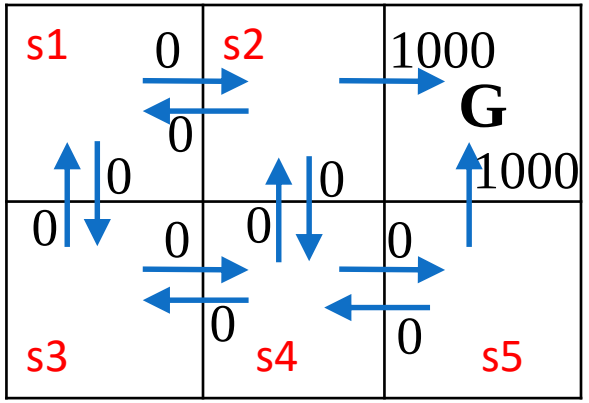
Solving MDPs: SARSA

Example 9: Apply SARSA. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	0	X

Episode 2

1. The state is s5.
2. The action is *U*.
3. Reward, r(s5, *U*).
4. Next state, G.
5. Choose next action.



r (*s*, *a*) (immediate reward) values

For next iteration

- Next state G.
- Next action: NIL.
- Terminal state.
- Initiate next episode.

➢ Take the action *Up*. Updated Q-value for *Q*(s5, *U*).

$$Q(s_5, U) = Q(s_5, U) + \alpha[r(s_5, U) + \gamma Q(G, .) - Q(s_5, U)]$$

$$= 0 + 0.1[1000 + 0.9 \times 0 - 0] = 100$$

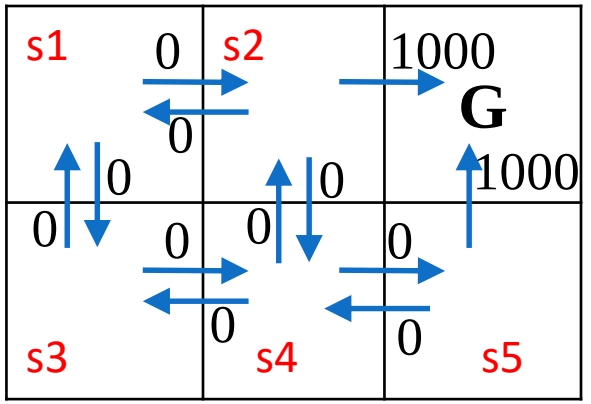
➢ Update Q-table accordingly. Next state is the goal state (after right move from s2).

Solving MDPs: SARSA

Example 9: Apply SARSA. Given $\gamma = 0.9$ and $\alpha = 0.1$.

	Left	Right	Up	Down
S1	X	0	X	0
S2	0	100	X	0
S3	X	0	0	X
S4	0	0	0	X
S5	0	X	**100**	X

Episode 2



r (*s*, *a*) (immediate reward) values

- Take the action *Up*. Updated Q-value for *Q*(s5, *U*).

$$Q(s_5, U) = Q(s_5, U) + \alpha[r(s_5, U) + \gamma Q(G, .) - Q(s_5, U)]$$
$$= 0 + 0.1[1000 + 0.9 \times 0 - 0] = 100$$

- Now, initiate next episode by following S-A-R-S-A sequence.

Solving MDPs: SARSA

- *TD does not need a transition model to perform its updates.* The environment supplies the connection between neighboring states in the form of observed transitions.
- *A TD agent that learns a Q-function does not need a model of the form P*(*s*'| *s*, *a*), *either for learning or for action selection.* For this reason, Q-learning is called a ***model-free*** method.
- Because Q-learning uses the best Q-value, it pays no attention to the actual policy being followed—it is an **off-policy** learning algorithm, whereas SARSA is an **on-policy** algorithm.
- Q-learning is more flexible than SARSA, in the sense that a Q-learning agent can learn how to behave well even when guided by a random or adversarial exploration policy. On the other hand, SARSA is more realistic: for example, if the overall policy is even partly controlled by other agents, it is better to learn a Q-function for what will actually happen rather than what the agent would like to happen.

Solving MDPs: SARSA

- **Exploration vs. Exploitation**: The agent balances exploration (choosing random actions) and exploitation (choosing the best-known action) through the epsilon-greedy policy.

- The key difference between SARSA and *Q*-learning is that SARSA uses the next action taken (according to the current policy) to update the *Q*-values, while *Q*-learning uses the best possible action for the next state.

Reinforcement Learning

- Framework: MDP (State, Action, Reward, Transition)

- Model-based Planning Methods
  - Value Iteration
  - Policy Iteration
  - Linear Programming

- Model-free Learning Methods
  - Monte Carlo (MC) method
  - Temporal difference (TD) algorithm
    - ✓ Q-learning algorithm
    - ✓ SARSA
    - ✓ TD(λ) algorithm
  - Gradient-based, and Gradient-free

➢ *Model-based methods, also referred to as planning methods, require a complete description of the model in terms of the transition and reward functions.*

➢ *Model-free methods, also referred to as learning methods, learn an optimal policy simply based on received observations and rewards.*

Books:

1. Sutton, Richard S., and Andrew G. Barto. ***Reinforcement learning: An introduction***. The MIT Press, 2018.
2. Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, ***Foundation of Machine Learning***. The MIT Press.
3. Tom Mitchell, ***Machine Learning***. McGraw Hill.