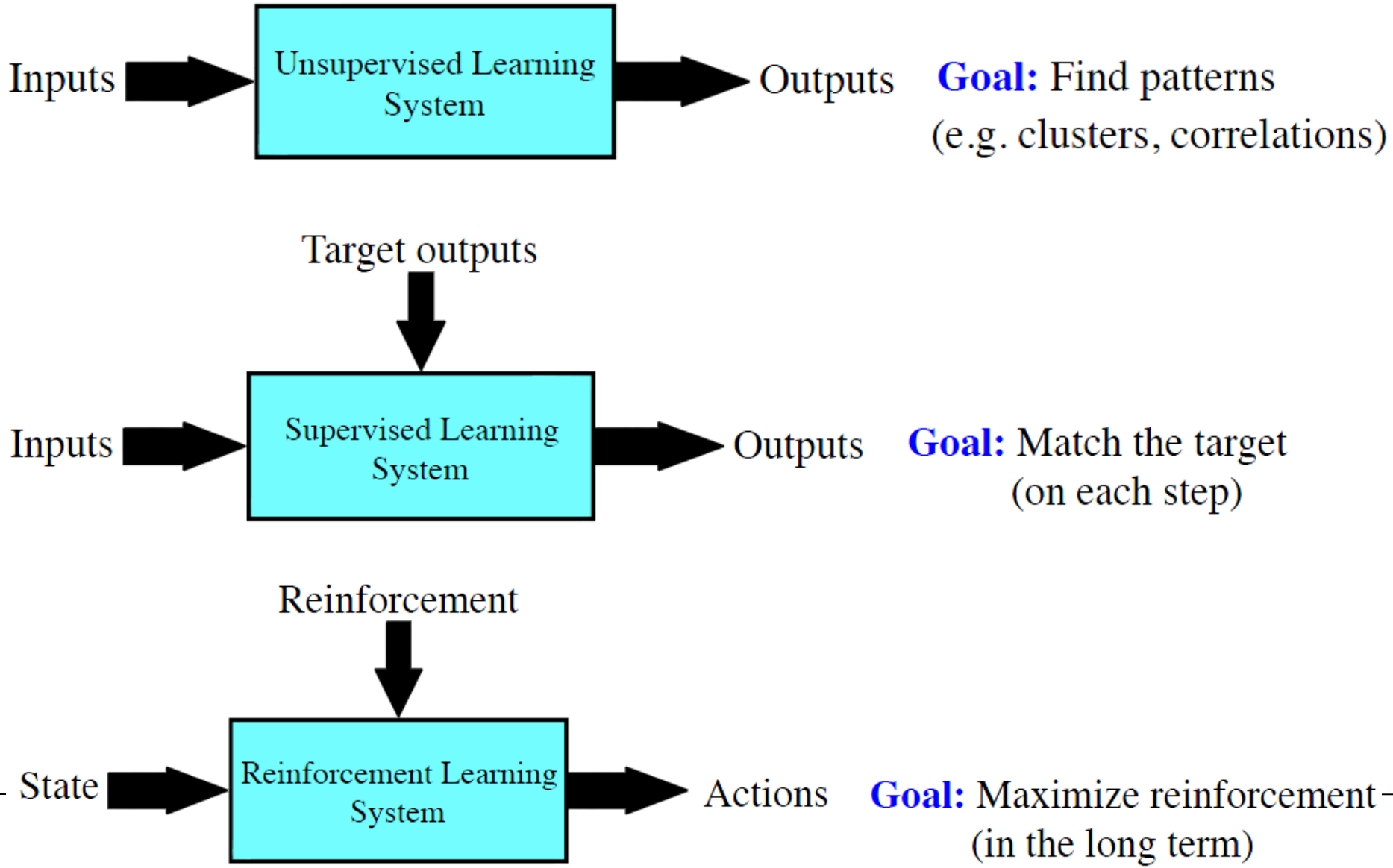
CS229: Machine Learning

Reinforcement Learning

Xiangliang Zhang

King Abdullah University of Science and Technology

3 Types of Learning



Outline of RL

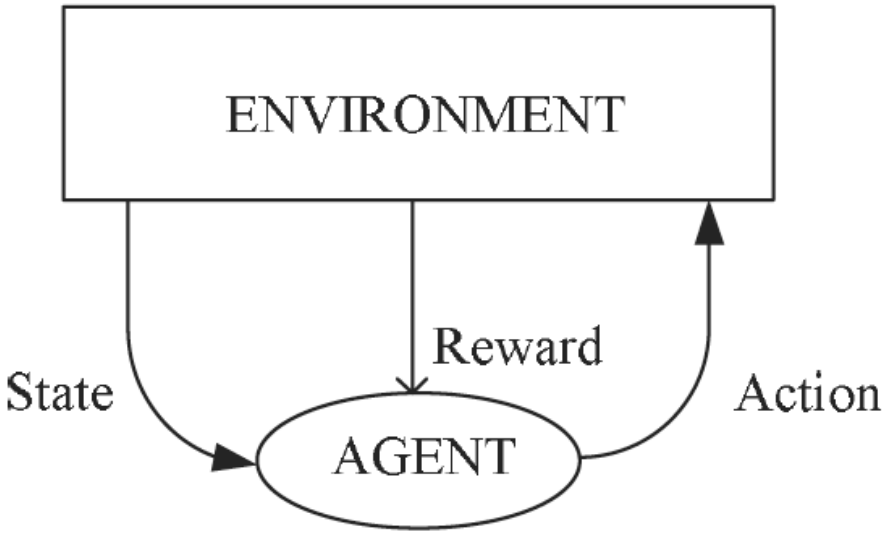
- Introduction of RL
- (Optimal) Value Functions
- Fundamental solutions
  - Dynamic programming
  - Monte Carlos (MC) methods
  - Temporal difference (TD) learning
    - On-policy, Off-policy
- RL with Deep Learning

Definition of RL

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.

Chapter 18, Introduction to Machine Learning.
Ethem Alpaydın 2009

RL diagram



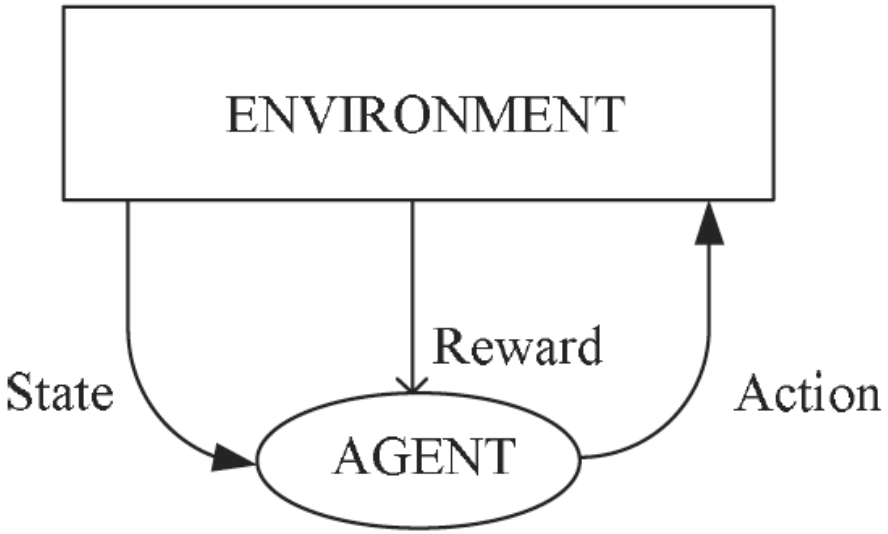
Agent: decision maker

State: one of the possible states in environment

Action: agent output to affect the environment

Reward: how good the action is

Objective of RL



Learning what to do —how to map states to actions— so as to maximize a numerical reward signal

Solution to a task: the best sequence of actions (with the maximum cumulative reward)

A Simple Example: k-armed bandit

Target: earn money AMAP

Task: decide which lever to pull

Agent: you

State: single state (one slot machine)

Action: choose one lever to pull

Reward: money given by the machine after one action (immediate reword after a single action)

Supervised learning? Need a teacher to tell you which one?

Another Example: a machine to play chess

Target: win the game

Task: decide a sequence of moves

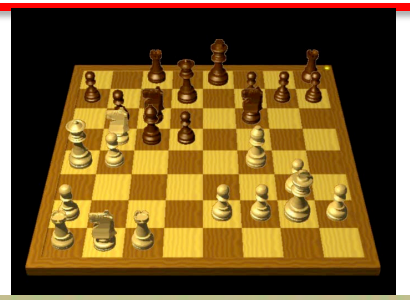
Agent: player

State: state of the board(environment)

Action: decide a legal move

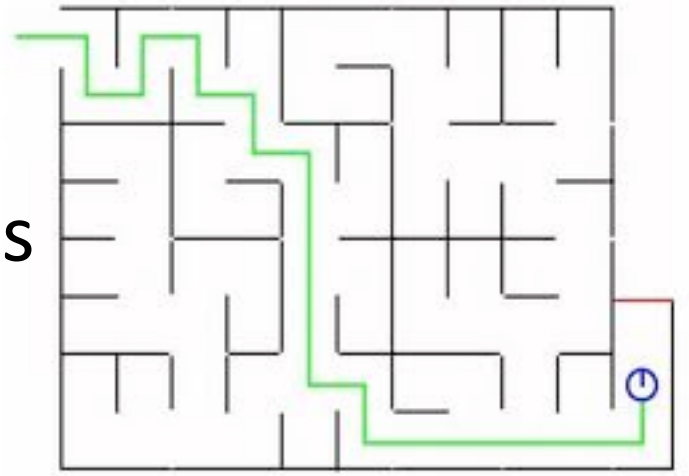
Reward: win or lose

(when game is over)



Is there a supervised learner who can tell you how to move?

Another Example: a robot in a maze



Target: reach the exit

Task: decide a sequence of movements

Agent: robot

State: position of the robot in the maze

Action: choose one of the 4 directions without hitting the walls

Reward: length of trajectories (playing time)

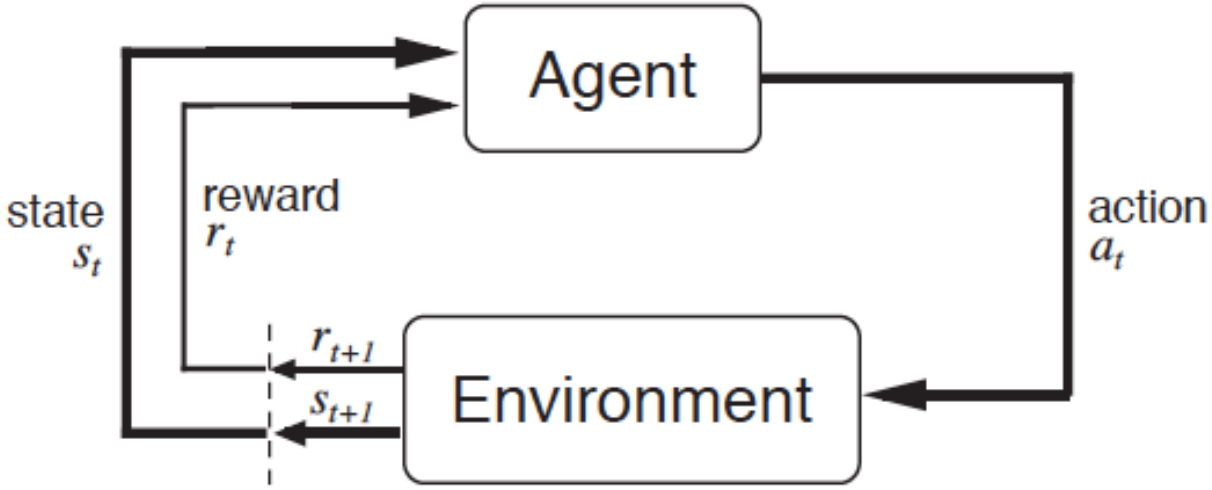
(when the robot reaches the exit)

More

A person's growing up is a selecting process.

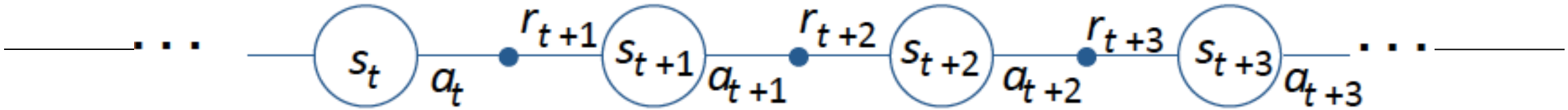
A person became mature through making decisions and taking actions.

The Agent-Environment Interface

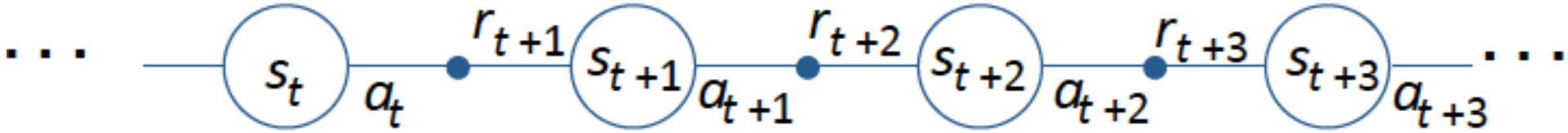


Agent and environment interact at discrete time steps: $t = 0,1,2,\ldots$

- Agent observes state at step t: $s_t \in S$
- produces action at step t: $a_t \in A(s_t)$
- gets resulting reward: $r_{t+1} \in \Re$
- and resulting next state: s_{t+1}



State, Action and Reward



Modeled using a *Markov Decision Process* (MDP)

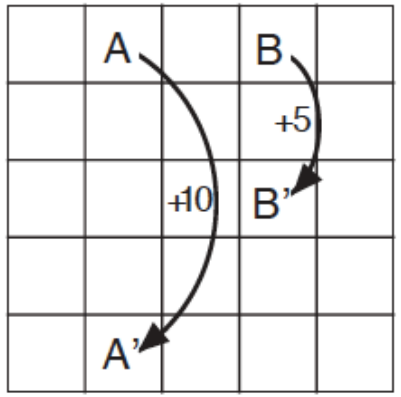
- Reward and next state are sampled from their respective probability distributions

$$p(r_{t+1} \mid s_t, a_t) \quad \text{and} \quad p(s_{t+1} \mid s_t, a_t)$$

- r_{t+1} and s_{t+1} can be
  - deterministic (only one possible value and state)
  - stochastic (different reward value and next state each time when choosing the same action) – reward value is defined by a probability distribution $p(r|a)$ or $p(r|s,a)$

Example: Gridworld

- deterministic actions: north, south, east, and west
- reward of **-1**: actions that would take the agent off the grid (location unchanged)
- reward of **+10**: all actions taking A ->A'
- reward of **+5**: all actions taking B->B';
- reward of **0**: others



(a)

Actions

Returns

Suppose the sequence of rewards after step t is:

$$r_{t+1}, \ r_{t+2}, \ r_{t+3}, \ldots\ldots$$

What do we want to maximize?

In general, we want to maximize the expected reword, $\mathrm{E}(R_t)$, for each step t

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

where T is a final time step at which a terminal state is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

Discounted Return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where $\gamma,\ \mathbf{0} \le \gamma \le 1$ is the **discount rate** (e.g., 0.9), decides the ***present value*** of ***future rewards***.

$$\textbf{shortsighted} \quad \mathbf{0} \leftarrow \gamma \rightarrow 1 \quad \textbf{farsighted}$$

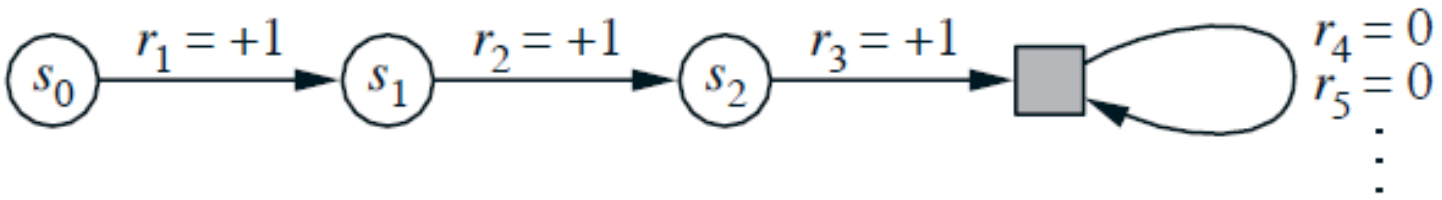
A Unified Notation of Return

- **An episodic task** and a **continuing task** can have the return:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Thinking of each episode as ending in an absorbing state that always produces reward of zero

and



$\gamma = 1$

Outline of RL

- Introduction of RL
- (Optimal) Value Functions
- Fundamental solutions
  - Dynamic programming
  - Monte Carlos (MC) methods
  - Temporal difference (TD) learning
    - On-policy, Off-policy
- RL with Deep Learning

Value Functions

- The **value of a state** ***s*** under a policy **π**, denoted $V^{\pi}(\boldsymbol{s})$, is the expected return when starting in ***s*** and following **π** thereafter

State-value function for policy π :

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- The **value of taking action** ***a*** in state ***s*** under a policy **π**, denoted $Q^{\pi}(\boldsymbol{s},\boldsymbol{a})$, as the expected return starting from ***s***, taking the action ***a***, and thereafter following policy **π**:

Action-value function for policy π :

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Bellman Equation for a Policy π

A fundamental property of value functions in RL: they satisfy **recursive** relationships!

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma\left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

Therefore:

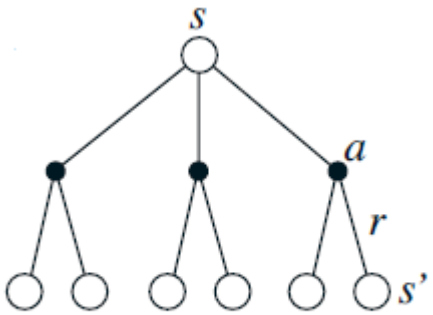
$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t \mid s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma R_{t+1} \mid s_t = s\} \end{aligned}$$

Bellman Equation for a Policy π (2)

Without the expectation operator:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V^{\pi}(s')]$$

Backup diagrams



for V^{π}

where $P^{a}_{ss'}$ is **transition probabilities**:

$$P^{a}_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad \textbf{for all } s, s' \in S, a \in A(s)$$

and $R^{a}_{ss'}$ is **reward probabilities:**

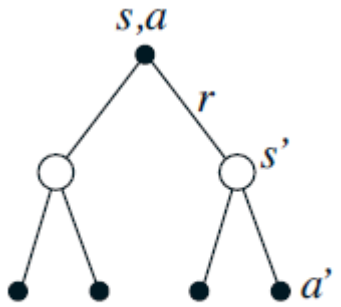
$$R^{a}_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \quad \textbf{for all } s, s' \in S, a \in A(s)$$

This is a set of equations (in fact, linear), one for each state.
The value function for π is its unique solution.

Bellman Equation for a Policy π (3)

$$
\begin{aligned}
Q^{\pi}(s,a) &= E_{\pi}\{R_t \mid s_t = s, a_t = a\} \\
&= E_{\pi}\{r_{t+1} + \gamma R_{t+1} \mid s_t = s, a_t = a\} \\
&= \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')]
\end{aligned}
$$

Backup diagrams

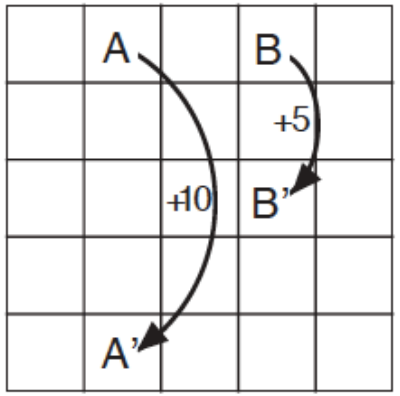


for Q^{π}

Example: Gridworld

- deterministic actions: north, south, east, and west
- reward of -1: actions that would take the agent off the grid
- reward of +10: all actions taking A ->A'
- reward of +5: all actions taking B->B’;
- reward of 0: others

A is the best state
But expected return is less than 10 (its immediate reward)



(a)

Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

$V^{\pi}(s)$

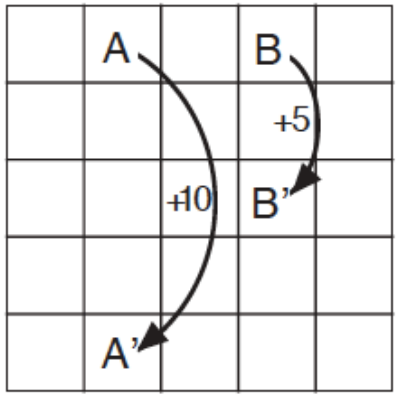
State-value function for equiprobable random policy;
$\gamma = 0.9$

high probability of hitting the edge of the grid

Example: Gridworld

- deterministic actions: north, south, east, and west
- reward of -1: actions that would take the agent off the grid
- reward of +10: all actions taking A ->A'
- reward of +5: all actions taking B->B’;
- reward of 0: others

State B,
Valued more than 5 (its immediate reward)



(a)

Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

$V^{\pi}(s)$

State-value function for equiprobable random policy;
$\gamma = 0.9$

high probability of hitting the edge of the grid

Optimal Value Functions

Solving a reinforcement learning task = finding the **optimal policy** π^* that achieves **max reward** over the long run

- The Optimal policy results in the **optimal state-value function**:

$$V^*(s) = \max_\pi V^\pi(s) \quad \text{for all } s \in S$$

- The Optimal policy results in the **optimal action-value function**:

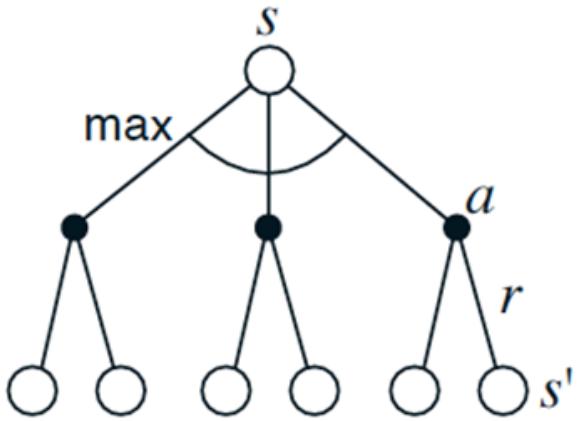
$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \quad \text{for all } s \in S \text{ and } a \in A(s)$$

Bellman Optimality Equation for V*

The **value of a state** under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^*(s,a) \\
&= \max_{a \in A(s)} E_{\pi^*}\{r_{t+1} + \gamma R_{t+1} \mid s_t = s, a_t = a\} \\
&= \max_{a \in A(s)} E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \\
&= \max_{a \in A(s)} \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]
\end{aligned}
$$

Backup diagram



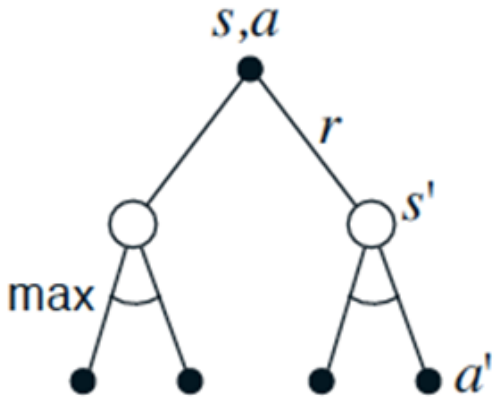
V* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for Q*

The Bellman optimality equation for Q* is

$$
\begin{aligned}
Q^*(s,a) &= E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \\
&= E\{r_{t+1} + \gamma \max_{a' \in A(s_{t+1})} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\} \\
&= \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma \max_{a' \in A(s')} Q^*(s', a')]
\end{aligned}
$$

Backup diagram



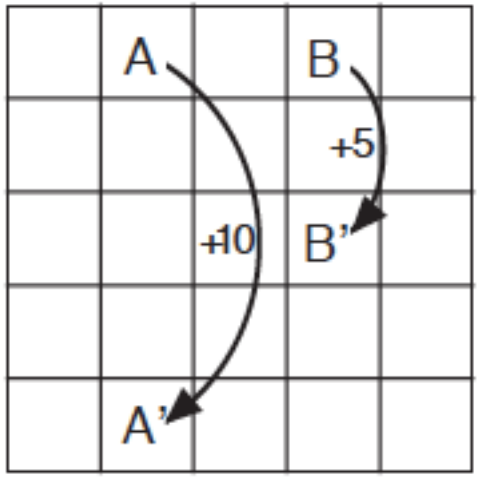
backup diagram: the root node is updated based on leaf nodes whose rewards/values are estimated

Q* is the unique solution of this system of nonlinear equations.

Optimal Value Function of the grid example

Given V*, one-step-ahead search produces the long-term optimal actions.

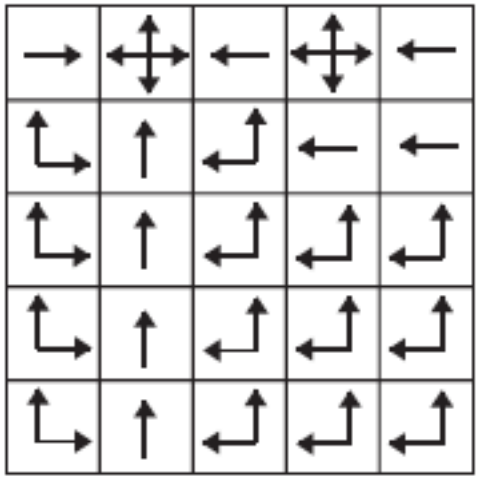
- E.g., back to the example of gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) π^*

Unique optimal value, but non-unique policy

Find Policy from Value

Given V*, the agent does a greedy one-step search:

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]$$

Given Q*, the agent does not even to do a one-step search:

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} Q^*(s,a)$$

→ Q is more interesting to have

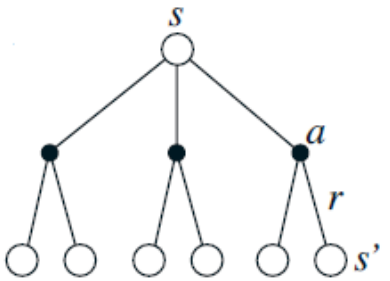
Outline of RL

- Introduction of RL
- (Optimal) Value Functions
- Fundamental solutions
  - Dynamic programming
  - Monte Carlos (MC) methods
  - Temporal difference (TD) learning
    - On-policy, Off-policy
- RL with Deep Learning

When the environment model parameters, $P^a_{ss'}$ and $R^a_{ss'}$ are known.

Model-based Learning: **Policy** Iteration algorithm

1. Initialize a policy π' and V(S) arbitrarily
2. Repeat
3. π ← π'
4. **Policy evaluation**: compute the values using π by solving the linear equations



for V^π

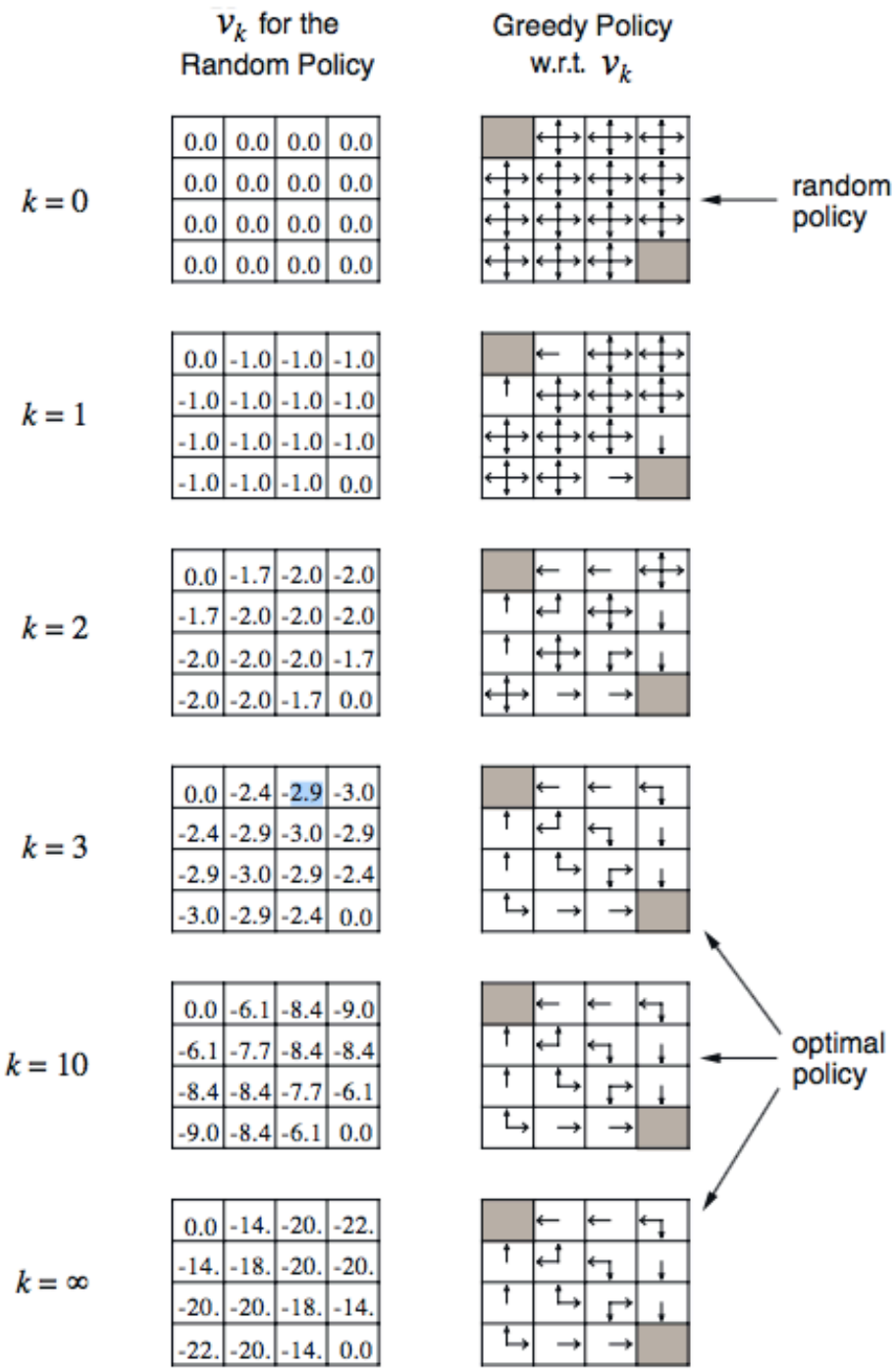
$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]$$

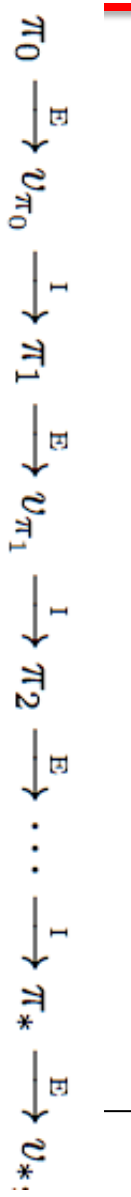
5. **Policy improvement**: for each state

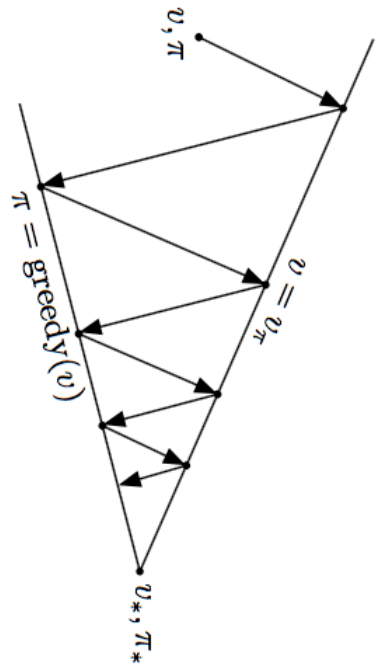
$$\pi'(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]$$

6. Until π = π'

Learning: **Policy** Iteration







Update estimates on the basis of other estimates (e.g., update estimates of the values of states based on estimates of the values of successor states) -- bootstrapping

Outline of RL

- Introduction of RL
- (Optimal) Value Functions
- Fundamental solutions
  - Dynamic programming
  - Monte Carlos (MC) methods
  - Temporal difference (TD) learning
    - On-policy, Off-policy
- RL with Deep Learning

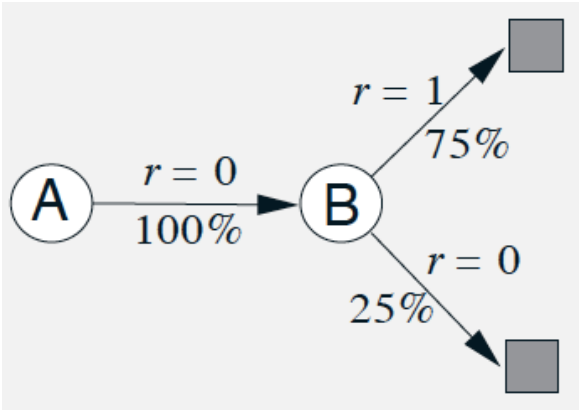
Monte Carlo Method

- **No** complete **knowledge** of the **environment**
- Use **experience** --- sample sequences of states, actions and rewards, from actual or simulated interaction with the environment
- Policy evaluation in *policy iterat*ion, $V^{\pi}(s)$ is estimated by using a set of episodes obtained by following π and passing through state s
  - **First-visit MC**, estimate $V^{\pi}(s)$ as the **average of returns** following **first visit to s**
- V(s) is updated independently w.r.t. s (no Bootstrap)

First-visit MC Method

Observed 8 episodes:

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0



By batch MC, V(B) = 6/8= 0.75 and V(A) = 0*

- With a model $P^a_{ss'}$ state values V(s) alone are sufficient to determine a policy;
- Without a model $P^a_{ss'}$ however, state values alone are not sufficient. Need Q(s,a)!

DP diagram: all possible (one-step) transitions
MC diagram: only those sampled on the one episode (to the end of the episode.)

*In fact, V(A) = 0 + V(B) = 0.75

On-policy first-visit MC with **ε**-greedy

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list
$\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
(a) Generate an episode using π
(b) For each pair s, a appearing in the episode:
$G \leftarrow$ the return that follows the first occurrence of s, a
Append G to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each s in the episode:
$A^* \leftarrow \arg\max_a Q(s,a)$ (with ties broken arbitrarily)
For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

ε =0.01, 0.1 …

- **most of the time** they choose an **action** that has **maximal estimated action value,**
- but **with probability ε** they instead select **an action at random**.

Exploration vs Exploitation

Explore in the environment: with Q, how can we choose an action?

Exploration	Exploitation
Discover new possibilities	Refining current procedure
e.g., find another lever that probably gives a higher reward	e.g., keep choosing a lever once it gives immediate reward
Long-term process	Short-term process
A tradeoff between Exploration and Exploitation	

Outline of RL

- Introduction of RL
- (Optimal) Value Functions
- Fundamental solutions
  - Dynamic programming
  - Monte Carlos (MC) methods
  - Temporal difference (TD) learning
    - On-policy, Off-policy
- RL with Deep Learning

Temporal Difference (TD) Learning

- A combination of MC and DP
  - Learn with **experience** without model knowledge, like MC
  - **Bootstrap**, like DP
- MC, wait until the return G_t following s is known to
  - Update $V(s_t)$ by $$V(s_t) + \alpha[G_t + V(s_t)] = (1-\alpha)V(s_t) + \alpha G_t$$

i.e., **Average** when α=0.5

- TD, wait until only the next time step t+1
  - Update $V(s_t)$ by

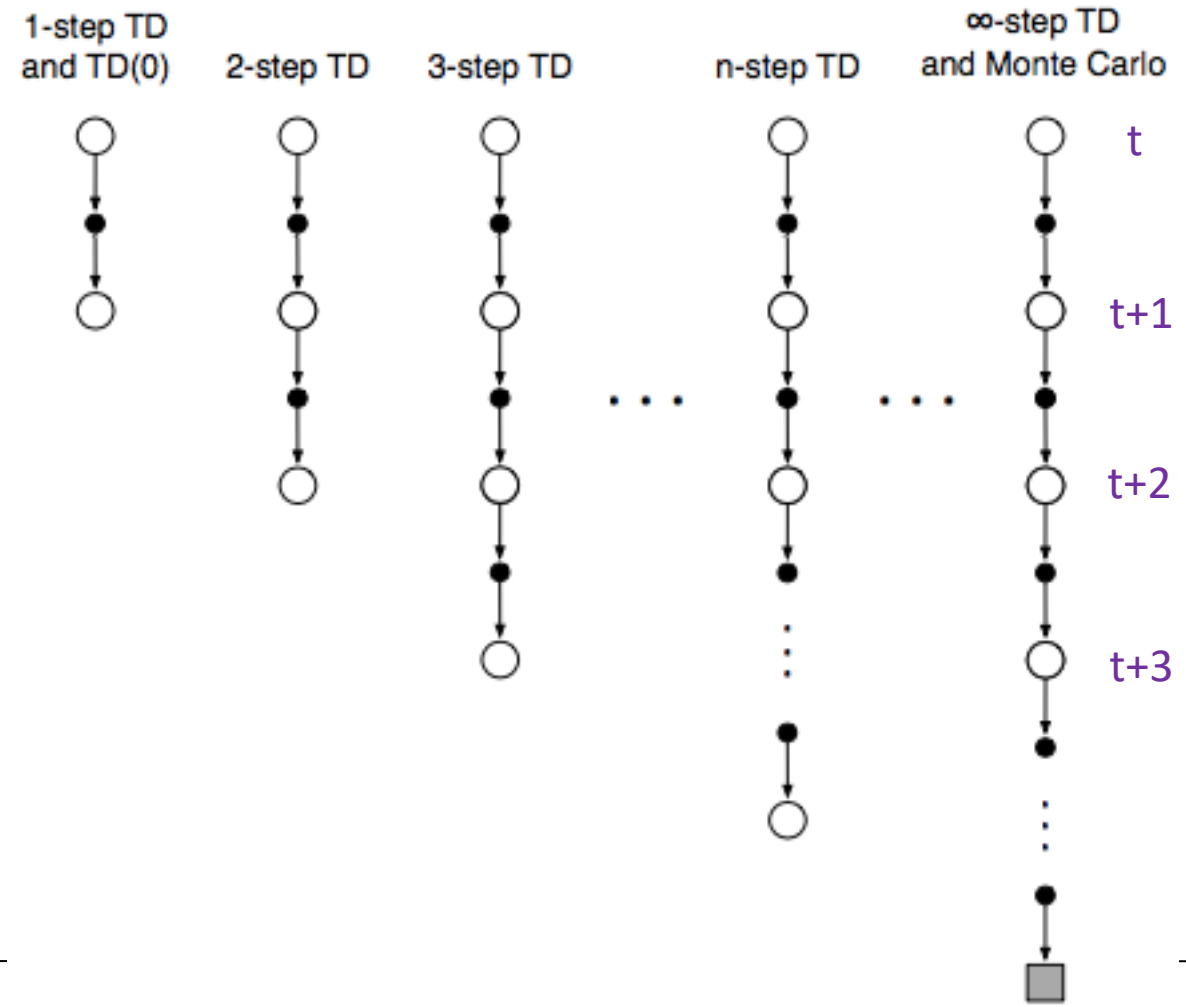
Time difference

$$V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$
$$= (1-\alpha)V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1})]$$

TD variants

- Update based on an **intermediate** number of rewards

Backup diagram



Q Learning algorithm by TD

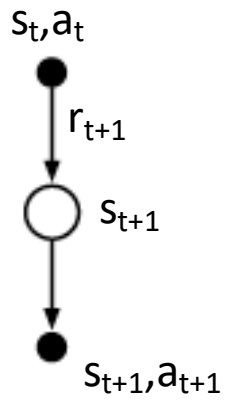
- TD for learning Q(s,a)

$$\hat{Q}(s_t, a_t) := \hat{Q}(s_t, a_t) + \alpha(r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t))$$

Estimate at t+1 Estimate at t

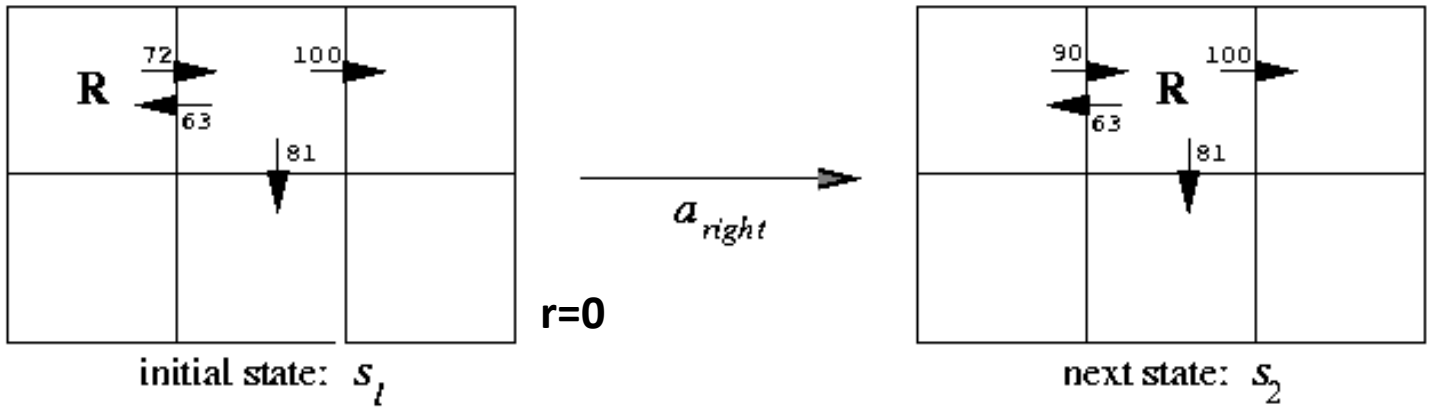
- Update Q based on the difference between estimates at two different times
- Like keep a running average

$$\hat{Q}(s_t, a_t) := (1-\alpha)\hat{Q}(s_t, a_t) + \alpha(r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}))$$



Example of updating Q

- Given the Q values from a previous iteration on the arrows
- Update:



$$\hat{Q}(s_1, right) := (1-\alpha)\hat{Q}(s_1, right) + \alpha(r + \gamma\ \hat{Q}(s_2, a_{t+1}))$$

Right?
Left?
Down?

Sarsa algorithm (on-policy TD)

For each (s,a), initialize Q(s,a) arbitrarily

Repeat (for each episode):

Initialize *s*

Select *a* using policy derived from *Q*, (e.g., e-greedy)

Repeat (for each step of episode):

- Take action a
- Observe immediate reward r_{t+1} and new state s_{t+1}
- Select a_{t+1} using policy derived from *Q*, (e.g., e-greedy)
- Update:

$$\hat{Q}(s_t, a_t) := (1-\alpha)\hat{Q}(s_t, a_t) + \alpha(r_{t+1} + \gamma\hat{Q}(s_{t+1}, a_{t+1}))$$

- $s := s_{t+1}$, $a := a_{t+1}$

Until *s* is terminal

Q Learning algorithm by TD, off-policy

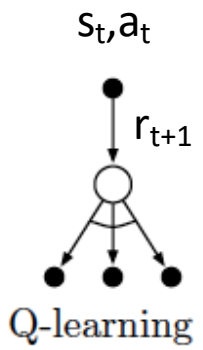
- TD for learning Q(s,a)

$$\hat{Q}(s_t, a_t) := \hat{Q}(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t))$$

Estimate at t+1 Estimate at t

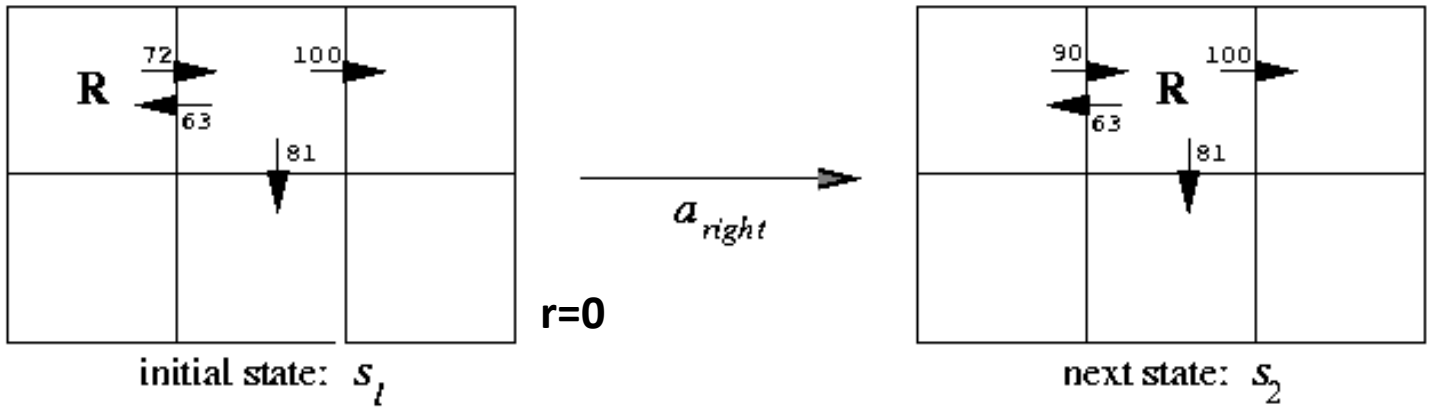
- Update Q based on the difference between estimates at two different times
- Like keep a running average

$$\hat{Q}(s_t, a_t) := (1-\alpha)\hat{Q}(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}))$$



Example of updating Q

- Given the Q values from a previous iteration on the arrows
- Update:



$$\hat{Q}(s_1, right) := (1-\alpha)\hat{Q}(s_1, right) + \alpha(r + \gamma \max_{a_{t+1}} \hat{Q}(s_2, a_{t+1}))$$

$$= 0.9 * 72 + 0.1 * 0.9 * 100 = 73.8$$

Q Learning algorithm (off-policy TD)

For each (s,a), initialize Q(s,a) arbitrarily

Repeat (for each episode):

Initialize *s*

Repeat (for each step of episode):

- Select an action a_{t+1} using policy derived from *Q*, e.g., e-greedy, and execute it
- Observe immediate reward r_{t+1} and new state s_{t+1}
- Update table entry as follows:

$$\hat{Q}(s_t, a_t) := (1-\alpha)\hat{Q}(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}))$$

- $s := s_{t+1}$

Until *s* is terminal

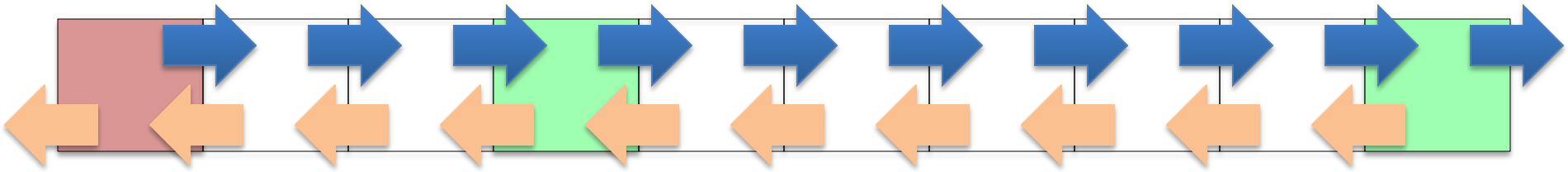
α gradually decreased in time for convergence

Off-policy vs On-policy TD algorithm

Off-policy (Q-learning)	On-policy (Sarsa)
$\hat{Q}(s_t,a_t) := (1-\alpha)\hat{Q}(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1},a_{t+1}))$	$\hat{Q}(s_t,a_t) := (1-\alpha)\hat{Q}(s_t,a_t) + \alpha(r_{t+1} + \gamma \hat{Q}(s_{t+1},a_{t+1}))$
look for all possible next actions **a** and choosing the best	use the policy derived from **Q** values to choose one next action **a** and use its **Q** value to calculate the temporal difference
Behavior policy (generating behavior) differs from *Estimation policy* (evaluated and improved)	estimating the value of a policy while using it to take actions

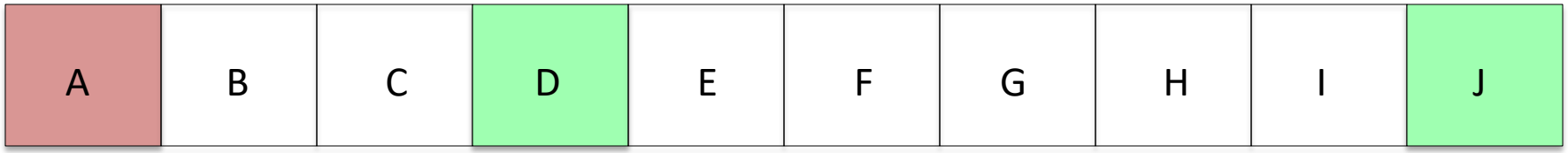
Example of Q Learning

- 10 states
- Reward +1 entering green squares, -1 entering red, 0 otherwise
- 2 actions: Left and Right (trying to move off the end goes to the other end)
- e=0.25 (e-greedy), 25% probability of moving opposite of the chosen action
- Discount factor $\gamma = 0.9$
- Learning rate $\alpha = 0.5$ (fixed) or $1/(t+1)^{0.5}$ (decreasing with t)



Example of Q Learning

- 10 states
- Reward +1 entering green squares, -1 entering red, 0 otherwise



Example of Q Learning

- Update Q(s,a) by

$$\hat{Q}(s_t, a_t) := (1-\alpha)\hat{Q}(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}))$$

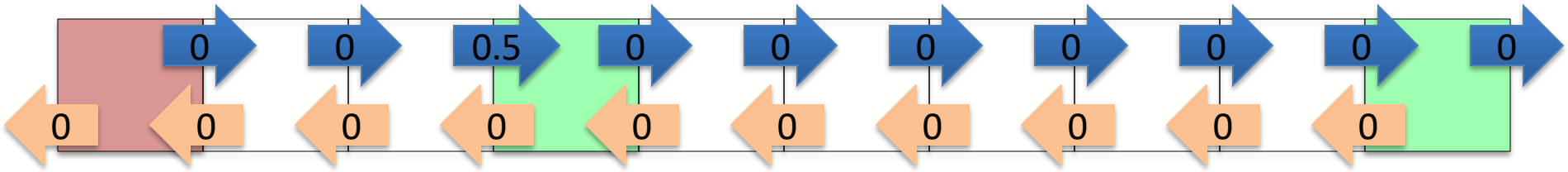
- After a while, what we have is (moving from C to D):

$$Q(C, right) = 0 + \alpha(1 + 0.9 * 0) = \alpha = 0.5$$

then if moving to D

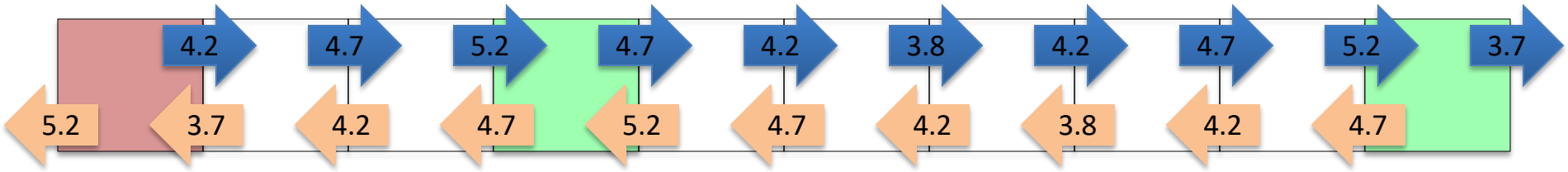
$$Q(D, right) = 0$$

$$Q(D, left) = 0 + \alpha(0 + 0.9 * max(0, 0.5)) = 0.225$$



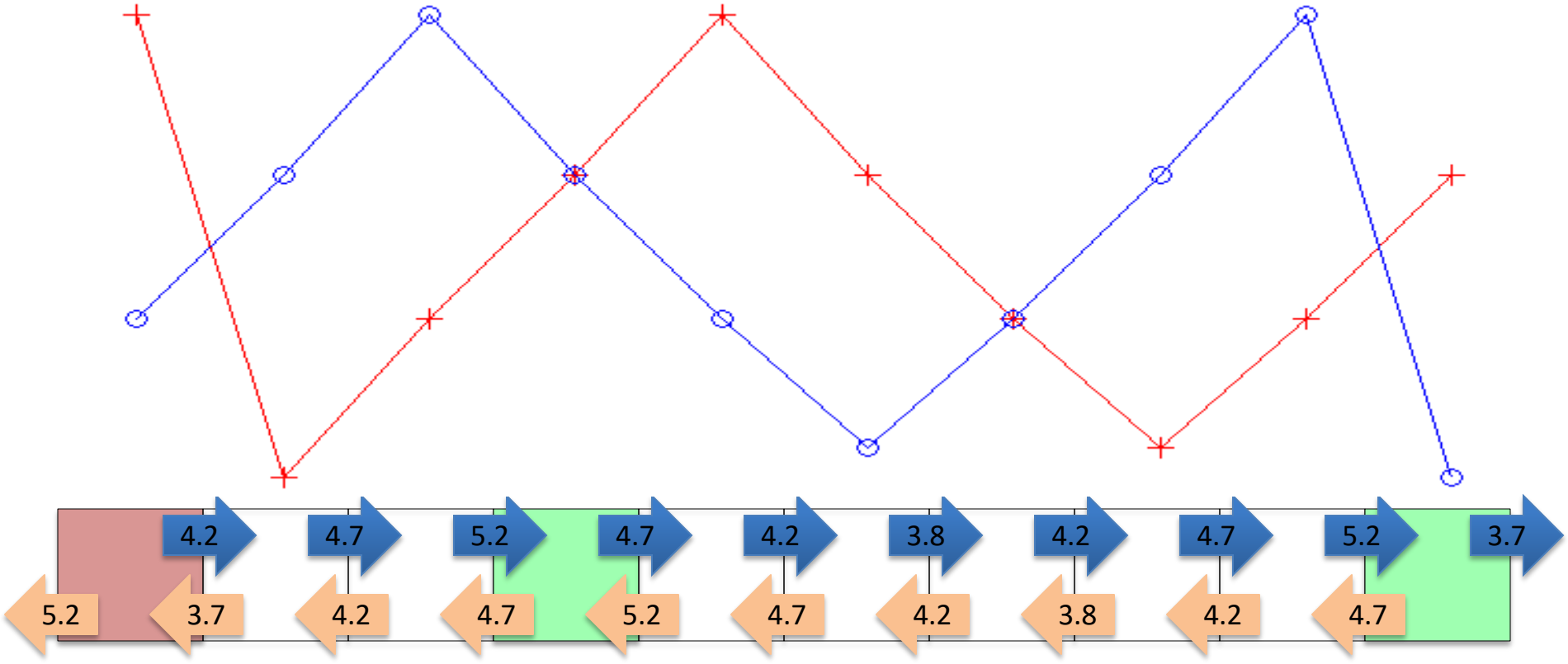
Example of Q Learning

- Finally, get the optimal Q values



Example of Q Learning

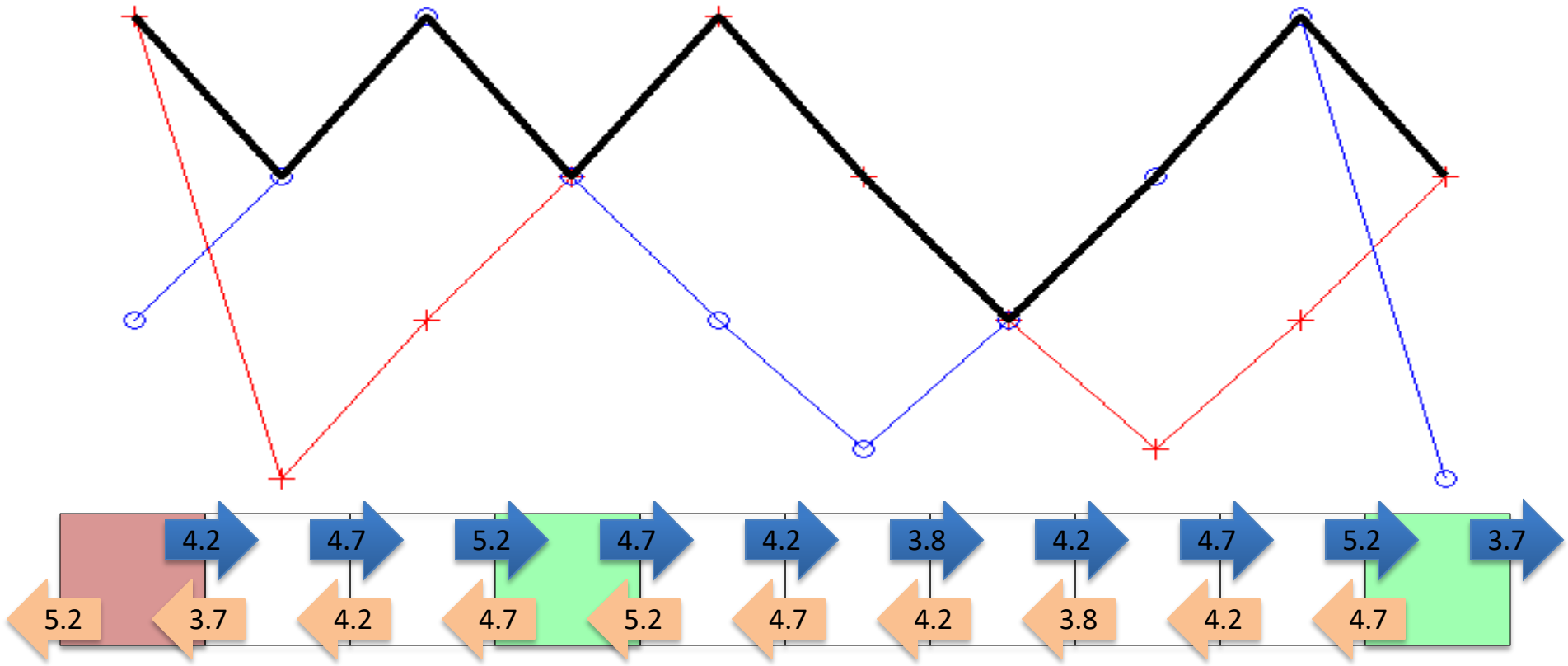
- Plot the Q values, with BLUE for right and RIGHT for left



Example of Q Learning

- What is **the optimal value** in each state? V*(s)=?

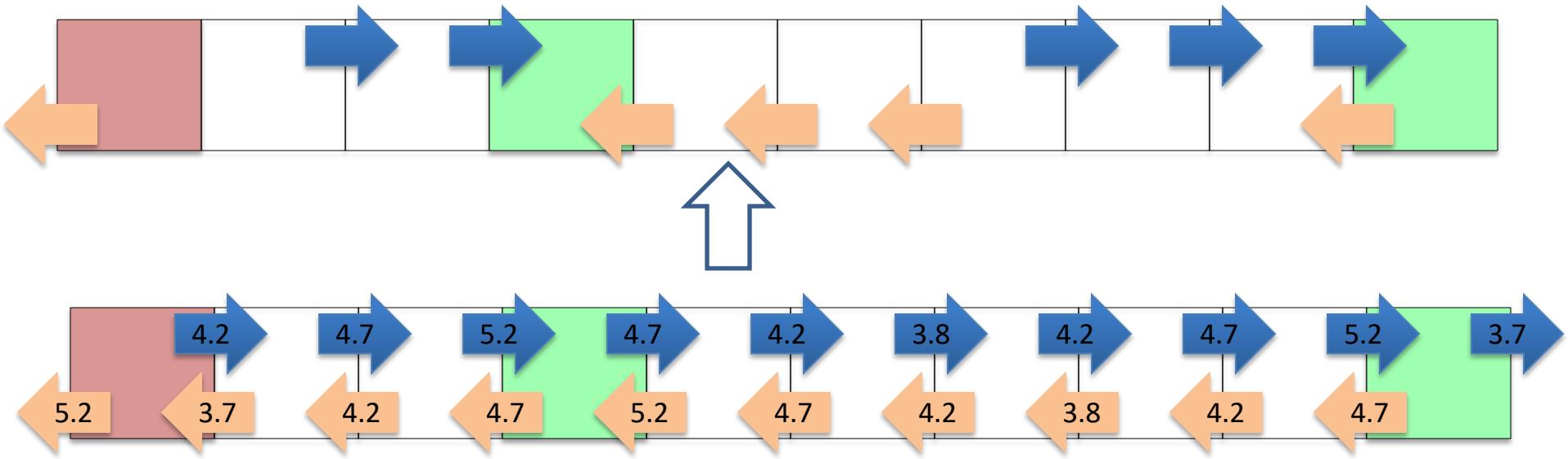
  The MAX shown in BLACK



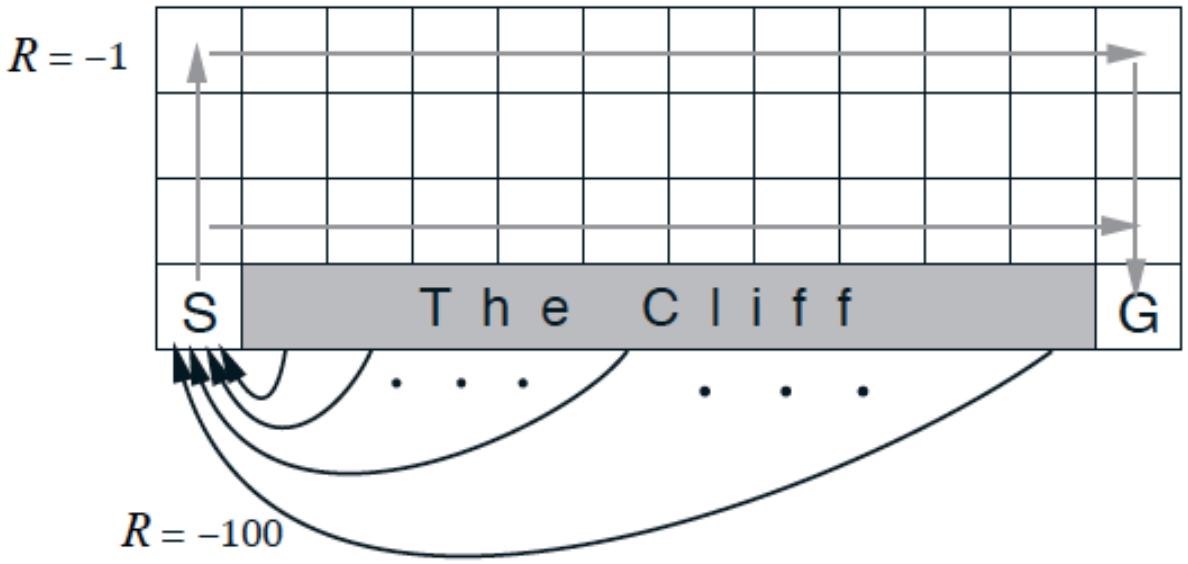
Example of Q Learning

- What is the **optimal policy** in each state? $\pi^*(s)=?$

  The action that gives MAX



Q-learning vs Sarsa



safe path

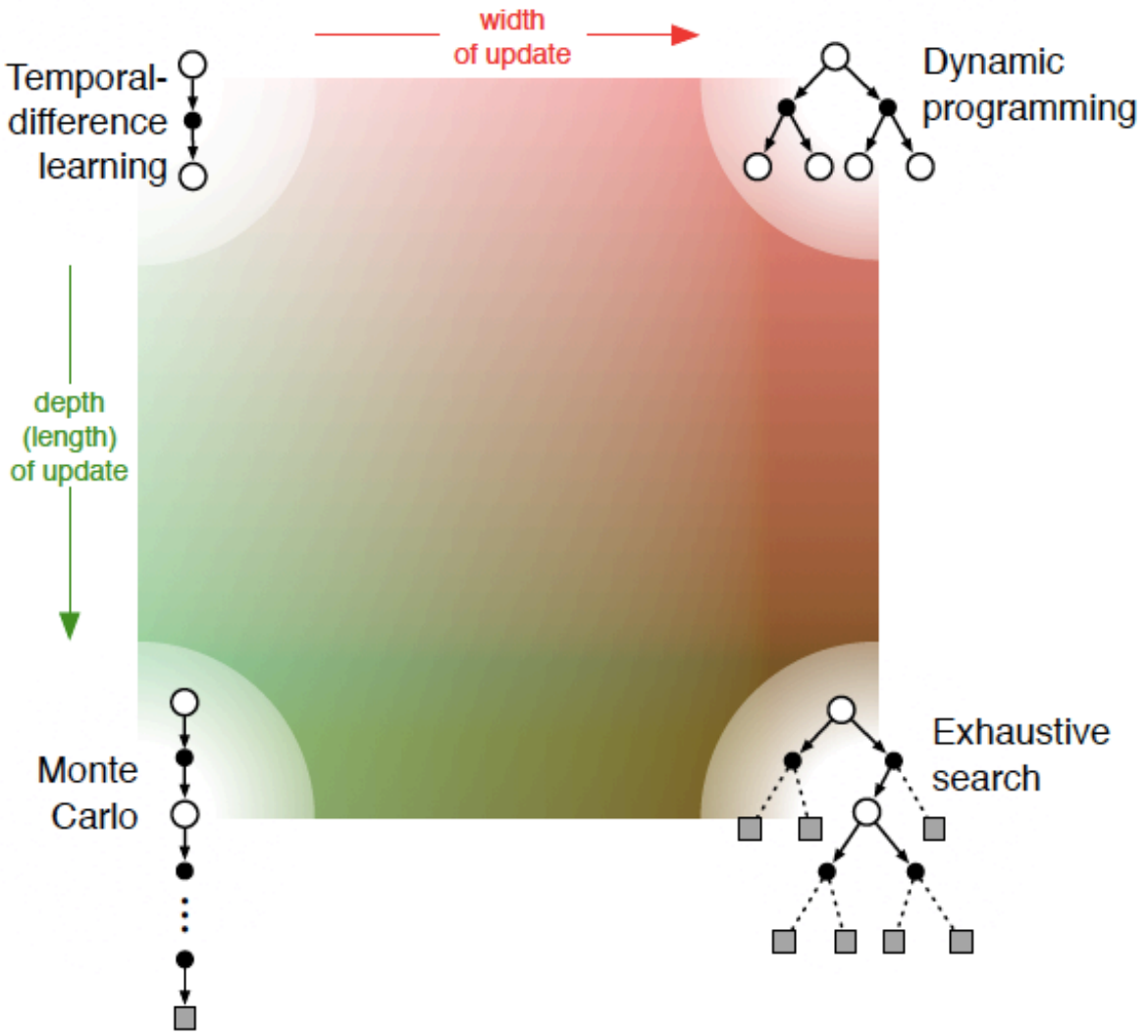
optimal path

Longer by Sarsa
(on policy)

Shorter by Q Learning
(off policy)

- Starting from S to G
- **Action**: up, down, right, and left.
- **Reward**: **– 1** on all transitions except those into the region marked "The Cliff" (reward of **– 100** and the agent is back to the start S.

Summary of different RL algorithms



Outline of RL

- Introduction of RL
- (Optimal) Value Functions
- Fundamental solutions
  - Dynamic programming
  - Monte Carlos (MC) methods
  - Temporal difference (TD) learning
    - On-policy, Off-policy
- RL with Deep Learning

The History

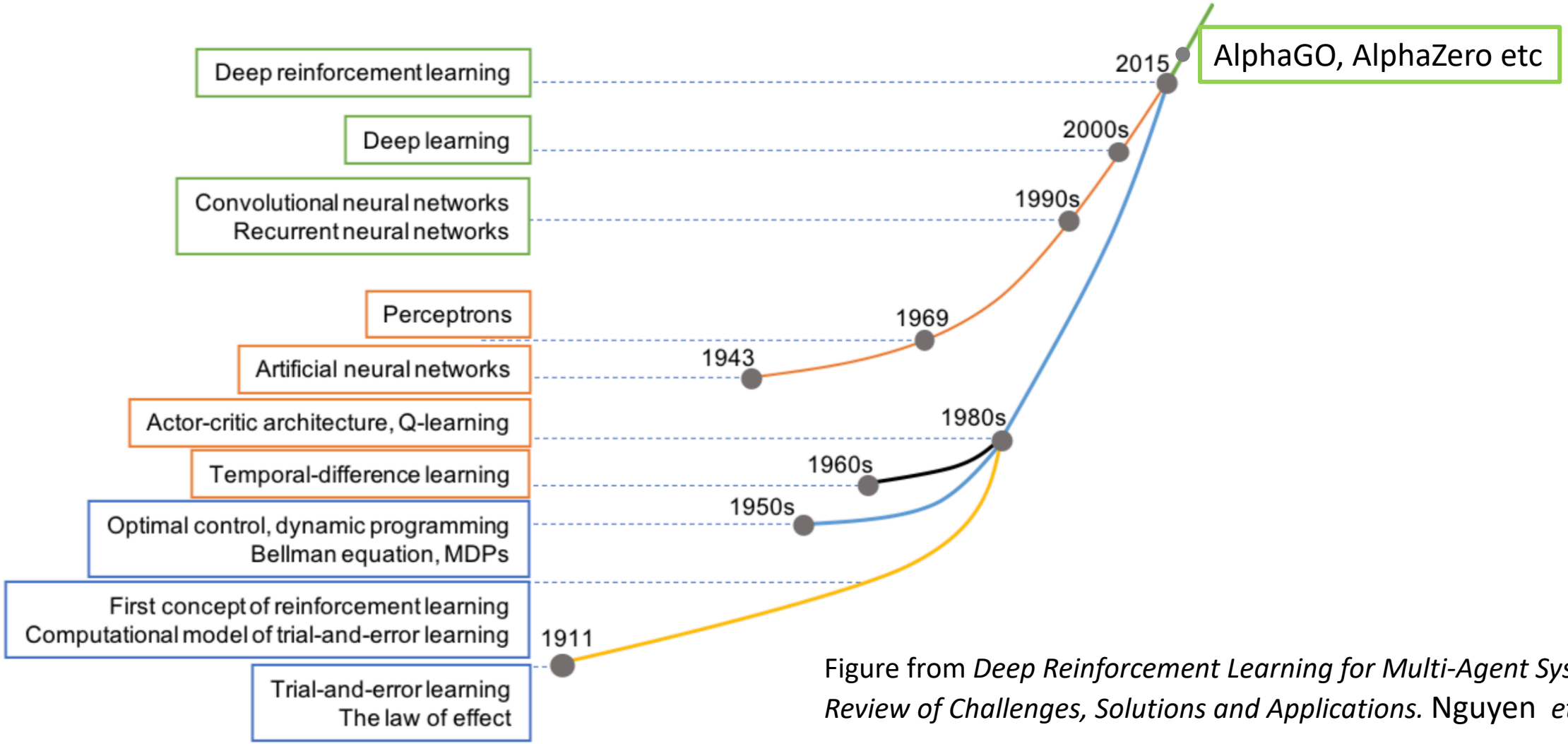


Figure from *Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications.* Nguyen *et al. 2020*

Fig. 1: Emergence of deep RL through different essential milestones.

Summary

Category	Dynamic programming	RL Methods: MC	Sarsa	Q-learning	AC
Model-free		✓	✓	✓	✓
On-policy		✓	✓		✓
Off-policy		✓		✓	✓
Bootstrapping	✓		✓	✓	✓

Memory expensive: use table memory structure (tabular method) to save the value function of each state or each state–action pair.

Actor–critic (AC) architecture: two separate memory structures for an agent:

1) **actor**: select a suitable action according to the observed state and transfer to the critic structure for evaluation
2) **critic**: uses a TD error function to decide the future tendency of the selected action

Table from *Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications.* Nguyen *et al. 2020*

Deep Q Networks

www.cs.toronto.edu › ~vmnih › docs › dqn PDF
Playing Atari with Deep Reinforcement Learning - University ...
by V Mnih - Cited by 3812 - Related articles
Playing Atari with Deep Reinforcement Learning. Volodymyr Mnih. Koray Kavukcuoglu. David Silver. Alex Graves Ioannis Antonoglou. Daan Wierstra.

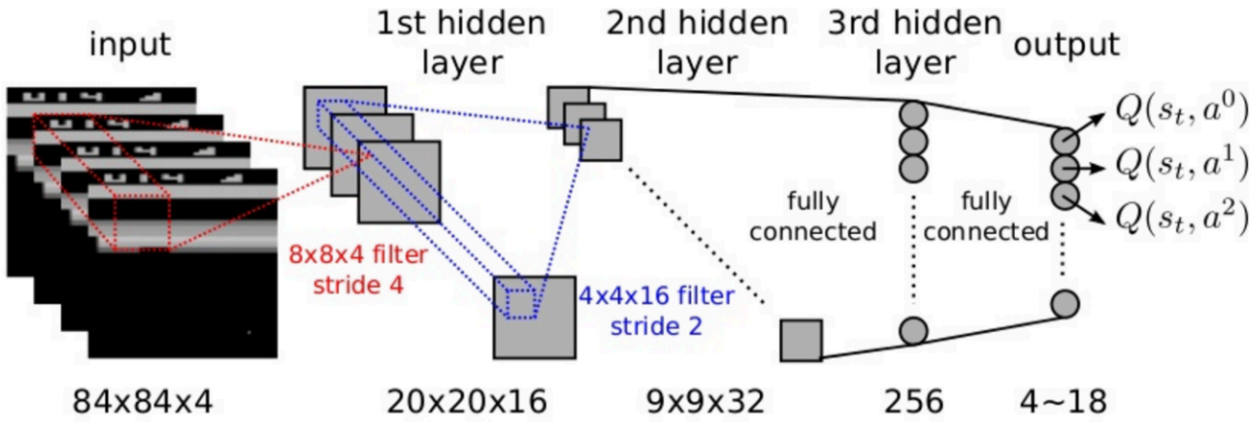
Approximating Q function Q(s,a) as $Q(s,a;\theta)$

Neural Network weights

with loss function

$$\left[Q(s,a;\theta) - \left(r(s,a) + \gamma \max_{a} Q(s',a;\theta)\right)\right]^2$$

Input: state s_t



Source: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/image_folder_7/DQNBreakoutBlocks.png

Output:
Q value of every action for state s_t

Playing Atari with Deep Reinforcement Learning. Minh et al. 2015
Successfully married deep learning and reinforcement learning.

Deep Q Networks

Approximating Q function Q(s,a) as Q(s,a; θ)

Neural Network weights

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Experience Replay

Playing Atari with Deep Reinforcement Learning. Minh et al. 2015
Successfully married deep learning and reinforcement learning.

Reference

Reading:

1. **Reinforcement Learning: An Introduction**

By Richard S. Sutton and Andrew G. Barto

2017

2. **Chapter 18 of Book**: Introduction to Machine Learning

By Ethem Alpaydın

2009

An interesting summary
https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

More if interested, not for exam, not for homework

Policy Gradient: works directly on policy $\pi(s,a)$, which is parameterized as $\pi(s,a|\theta)$

Find the **optimal** policy π^* with **parameter** θ^*, by maximizing what objective function?

$$J(\theta) = \sum_{s\in\mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a)$$

Stationary state distribution under policy π

$$d^\pi(s) = \sum_{s'} \sum_{t=0}^{T-1} p_0(s') p(s_{t+1}=s | s_t = s', a_t = \pi(s'))$$

$$d^\pi(s) = \lim_{t\to\infty} P(s_t = s | s_0, \pi_\theta)$$

What's the gradient?

$$\nabla_\theta J(\theta) \propto \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

Proof in Sutton et al. Reinforcement learning: An introduction.

An interesting summary
https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

More if interested, not for exam, not for homework

Policy Gradient method

REINFORCE:

1. Initialize the policy parameter θ at random.
2. Generate one trajectory on policy π_θ: $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$.
3. For t=1, 2, … , T:
    1. Estimate the the return G_t;
    2. Update policy parameters: $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$

- Policy Gradient Algorithms
  - REINFORCE
  - Actor-Critic
  - Off-Policy Policy Gradient
  - A3C
  - A2C
  - DPG
  - DDPG
  - D4PG
  - MADDPG
  - TRPO
  - PPO
  - ACER
  - ACTKR
  - SAC
  - SAC with Automatically Adjusted Temperature
  - TD3
  - SVPG
  - IMPALA

An interesting summary
https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

More if interested, not for exam, not for homework

(on policy) **Actor-Critic**, consists of two models

- **Critic** updates the value function parameters w and depending on the algorithm it could be action-value $Q_w(a|s)$ or state-value $V_w(s)$.
- **Actor** updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic.

1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \dots T$:
   1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
   2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
   3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
   4. Compute the correction (TD error) for action-value at time t:
      $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
      and use it to update the parameters of action-value function:
      $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
   5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

Update Actor

Update Critic

- Policy Gradient Algorithms
  - REINFORCE
  - Actor-Critic
  - Off-Policy Policy Gradient
  - A3C
  - A2C
  - DPG
  - DDPG
  - D4PG
  - MADDPG
  - TRPO
  - PPO
  - ACER
  - ACTKR
  - SAC
  - SAC with Automatically Adjusted Temperature
  - TD3
  - SVPG
  - IMPALA

An interesting summary
https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

More if interested, not for exam, not for homework

Deep Deterministic Policy Gradient, a model-free off-policy actor-critic algorithm, combining DPG with DQN.

- DQN: stabilize the learning of Q-function by experience replay and the frozen target network. But in discrete space
- DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy
  - An exploration policy by adding noise

$$\mu'(s) = \mu_\theta(s) + \mathcal{N}$$

  - soft updates ("conservative policy iteration") on the parameters of both actor and critic,

$$\text{with } \tau \ll 1 : \theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

- Policy Gradient Algorithms
  - REINFORCE
  - Actor-Critic
  - Off-Policy Policy Gradient
  - A3C
  - A2C
  - DPG
  - DDPG
  - D4PG
  - MADDPG
  - TRPO
  - PPO
  - ACER
  - ACTKR
  - SAC
  - SAC with Automatically Adjusted Temperature
  - TD3
  - SVPG
  - IMPALA

An interesting summary
https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R — Experience Replay
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$ — Update Critic
    Update the actor policy using the sampled policy gradient: — Update Actor

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks: — Update target network (off policy)

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Policy Gradient Algorithms

- REINFORCE
- Actor-Critic
- Off-Policy Policy Gradient
- A3C
- A2C
- DPG
- DDPG
- D4PG
- MADDPG
- TRPO
- PPO
- ACER
- ACTKR
- SAC
- SAC with Automatically Adjusted Temperature
- TD3
- SVPG
- IMPALA