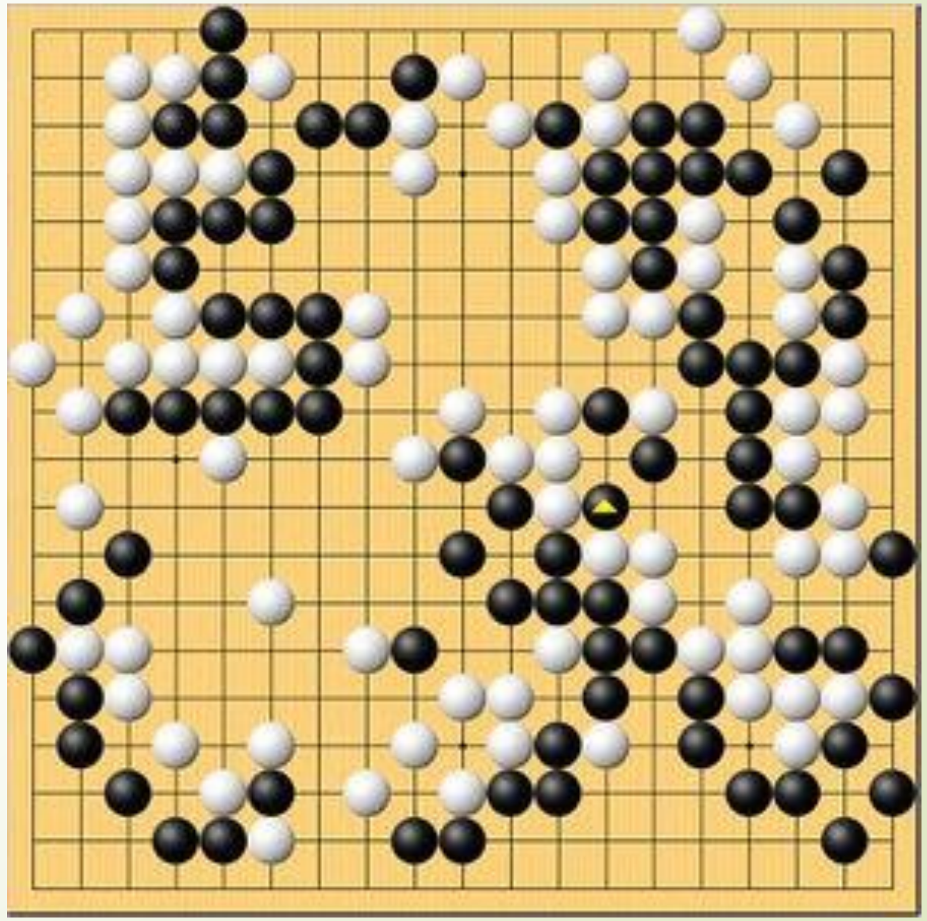
UNRAVELING THE CONCEPTS OF **REINFORCEMENT LEARNING**

Problem 1

- Teaching an AI agent to play Go
- The players take turns placing the stones on the intersections of a board with a 19×19 grid of lines
- Challenges
- The lower bound on the number of legal moves in Go has been estimated to be 2×10^{170} .
- After first two moves, number of possible next moves are close to 130,000



Problem 2

A mobile robot has to decide whether it should enter a new room in search of dirt to collect or start trying to find its way back to its battery recharging station. It has to make its decision based on how quickly and easily it has been able to find the recharger in the past.

Challenges

- Agent needs to interact with its environment and adjust its behavior
- The next action it takes has to be considered by taking into account indirect and delayed consequences for the agent (reaching out the charger before batteries run down)
- Agent requires planning considering all uncertainties in the environment
- Agent needs to use knowledge gained from previous experience

Reinforcement Learning

- Reinforcement Learning (RL) is a subfield of Machine Learning where an agent learns by interacting with its environment, observing the results of these interactions and receiving a reward (positive or negative) accordingly. This way of learning mimics the fundamental way in which humans and human-alike animals learn.
- Three aspects of formulation – sensation, action and goal.

RL vs Supervised and Unsupervised Learning

RL	Supervised Learning
Learning from interaction and experience	Learning from training examples
Dynamic environment	Static environment
Optimal solution from number of possible solutions that maximizes the reward	Prior knowledge of input and true solution(output)

RL	Unsupervised Learning
Finds optimal actions or most accurate labels for each particular situation to maximize long-term benefits	Assigns labels to similar data samples as clusters
For classification, reward if classified correctly else penalized	For classification, finds patterns for similar data

Features of Reinforcement Learning

- Trial-and-error search
- Delayed reward
- Exploration vs exploitation problem

Elements of Reinforcement Learning

Along with agent and environment, there are four sub-elements of RL

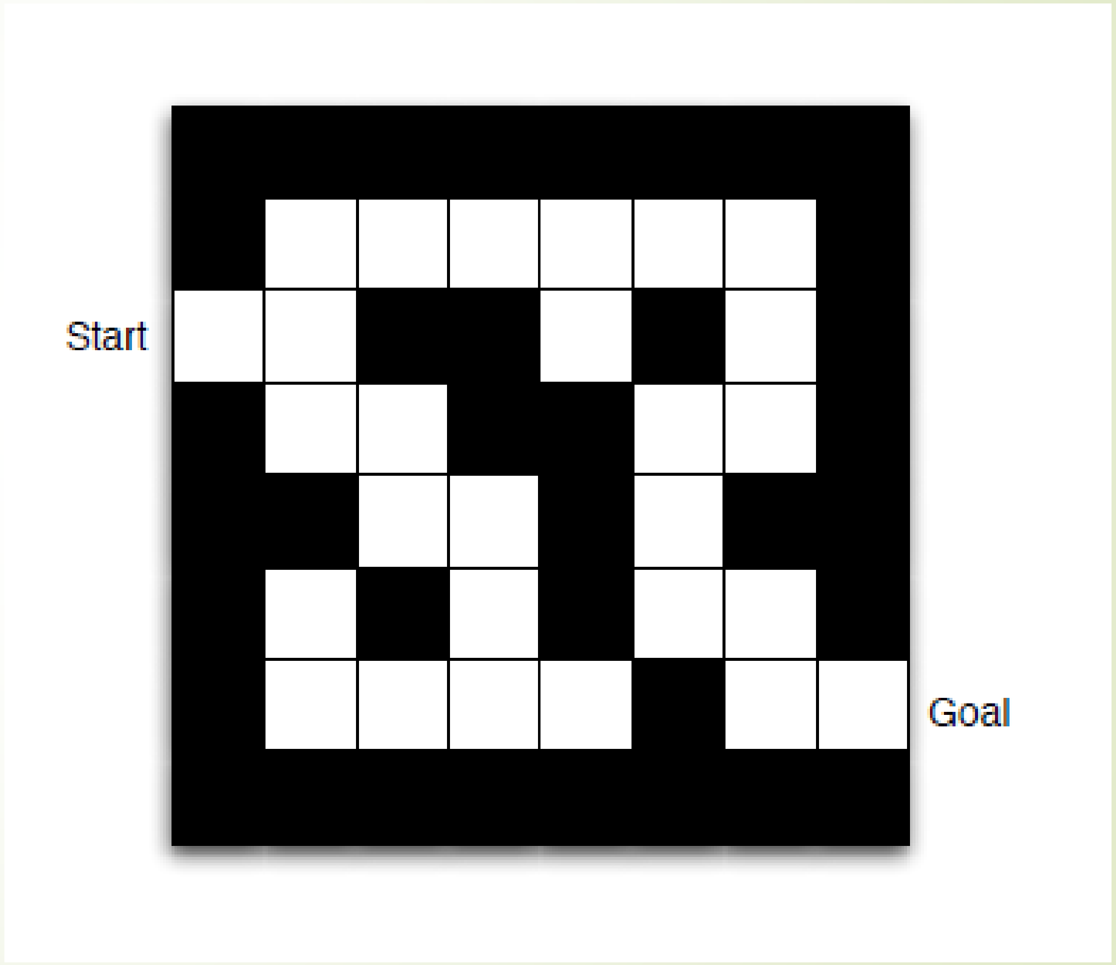
- A policy - agent's behaviour function
- A reward function - defines the goal in a reinforcement learning problem
- A value function - how good is each state and/or action in long run
- A model of the environment - agent's representation of the environment

Steps for Reinforcement Learning

- The agent observes an input state
- An action is determined by a decision making function (policy)
- The action is performed
- The agent receives a scalar reward or reinforcement from the environment
- Information about the reward given for that state / action pair is recorded

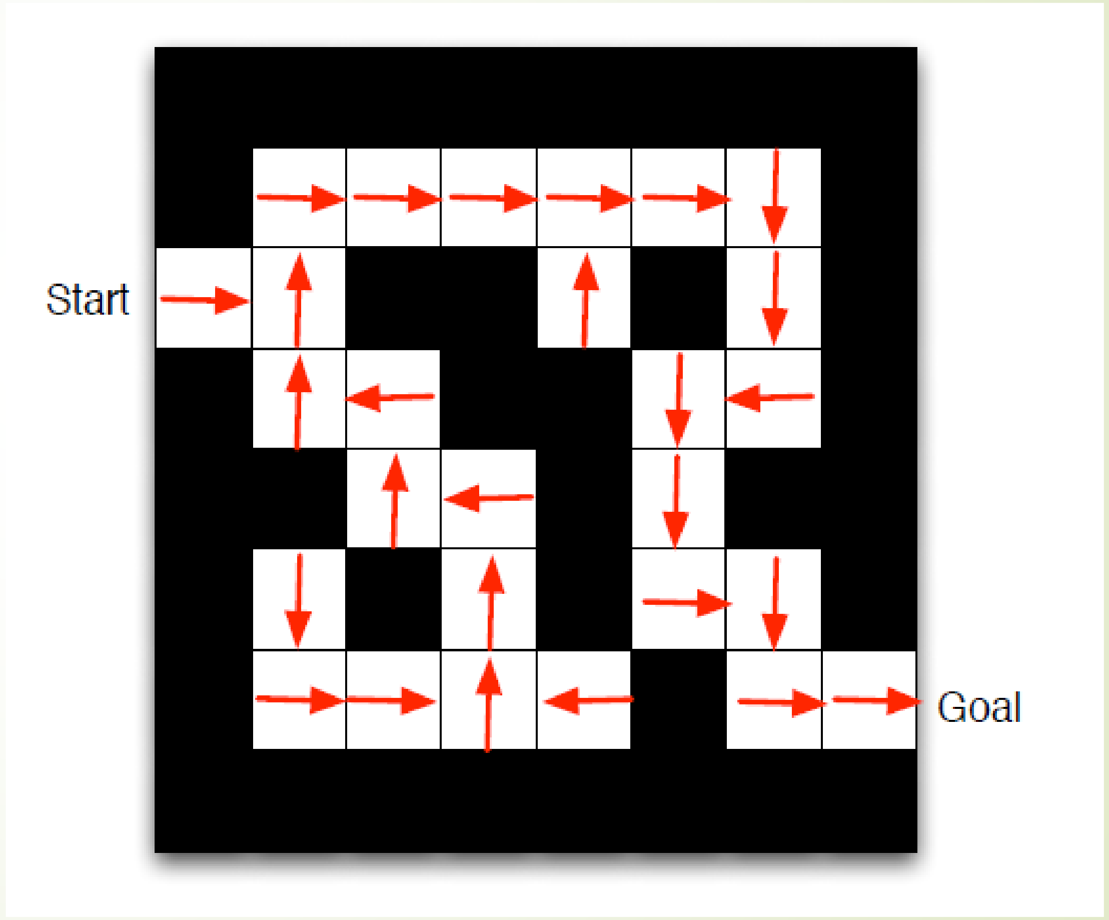
Maze example

- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location



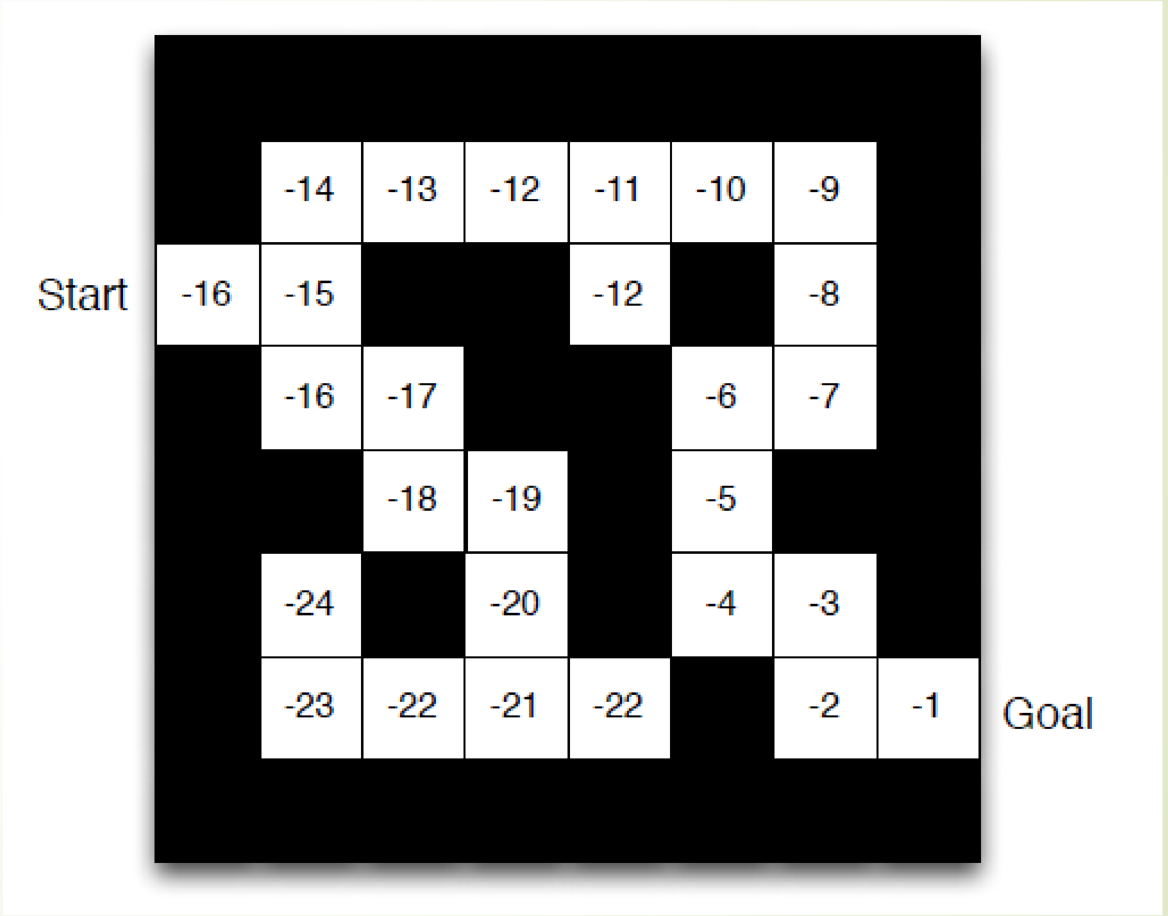
Maze example : Policy

- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location



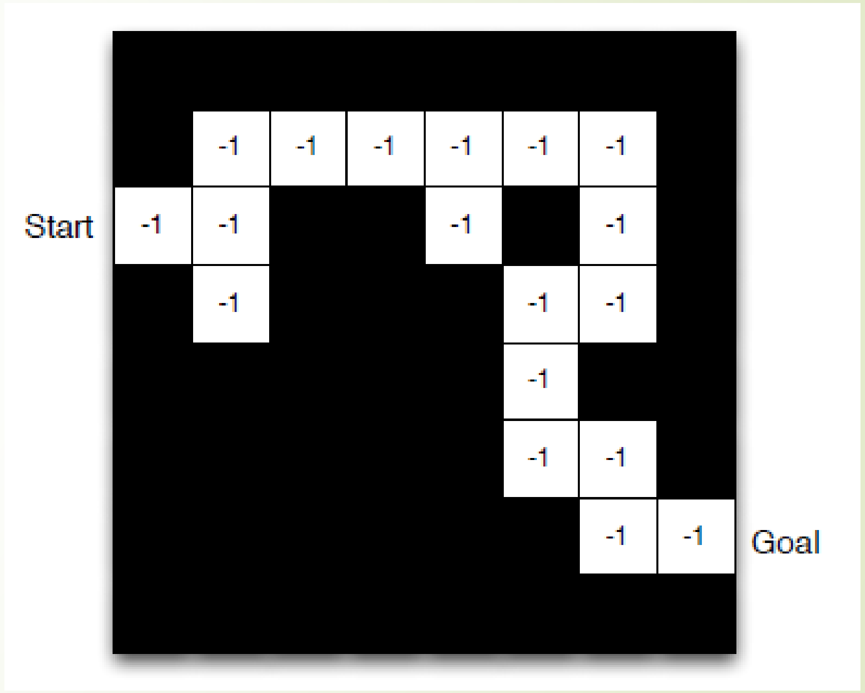
Maze example : Value Function

- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location



Maze example : Model

- Agent may have an internal model of the environment
- It has seen the trajectory through the maze and reached the goal
- Builds the model
- Dynamics: how actions change the state



On-Policy vs Off-Policy Learning

- On Policy – Improving/Evaluating on the original policy
- Off Policy – Improving/Evaluating a new policy
- If a good starting policy is available: on-policy may be interesting, but may not explore other policies well
- If more exploration is necessary, then perhaps off-policy is advisable, but maybe slow
- May lead to the same result (e.g. after greedification)
- Examples: On Policy – TD Learning, SARSA

  Off Policy – Q Learning, R Learning

Passive vs Active learning

Passive learning

- The agent has a fixed policy and tries to learn the utilities of states by observing the world go by
- Often serves as a component of active learning algorithms
- Often inspires active learning algorithms

Active learning

- The agent attempts to find an optimal (or at least good) policy by acting in the world

Model-Based vs Model-Free RL

Model based approach to RL:

- learn the MDP model, or an approximation of it use it for policy evaluation or to find the optimal policy

Model free approach to RL:

- derive the optimal policy without explicitly learning the model
- useful when model is difficult to represent and/or learn

Exploration vs Exploitation Dilemma

- Online decision making involves a fundamental choice:
- Exploitation: make the best decision given current information
- Exploration: gather more information

Exploration vs Exploitation Dilemma

- Restaurant Selection

  - Exploitation: Go to your favorite restaurant

  - Exploration: Try a new restaurant

- Game Playing

  - Exploitation: Play the move you believe is best

  - Exploration: Play an experimental move

N-armed Bandit Problem

- Choose repeatedly from one of n actions; each choice is called a play
- After each play , you get a reward , where,

$$E<r_t \mid a_t> = Q^*(a_t)$$

These are unknown action values

Distribution of r_t depends only on a_t

- Objective is to maximize the reward in the long term, e.g., over 1000 plays
- To solve the n-armed bandit problem, you must explore a variety of actions and then exploit the best of them.

Exploration vs Exploitation Dilemma

- Suppose you form estimates

  $Q_t(a) \sim Q^*(a)$ Action value estimates
- The greedy action at t is
- $A^*_t = \text{argmax}\ Q_t(a)$
- $A_t = A^*_t$ => Exploitation
- $A_t \neq A^*_t$ => Exploration
- You can't exploit all the time; you can't explore all the time
- You can never stop exploring; but you should always reduce exploring

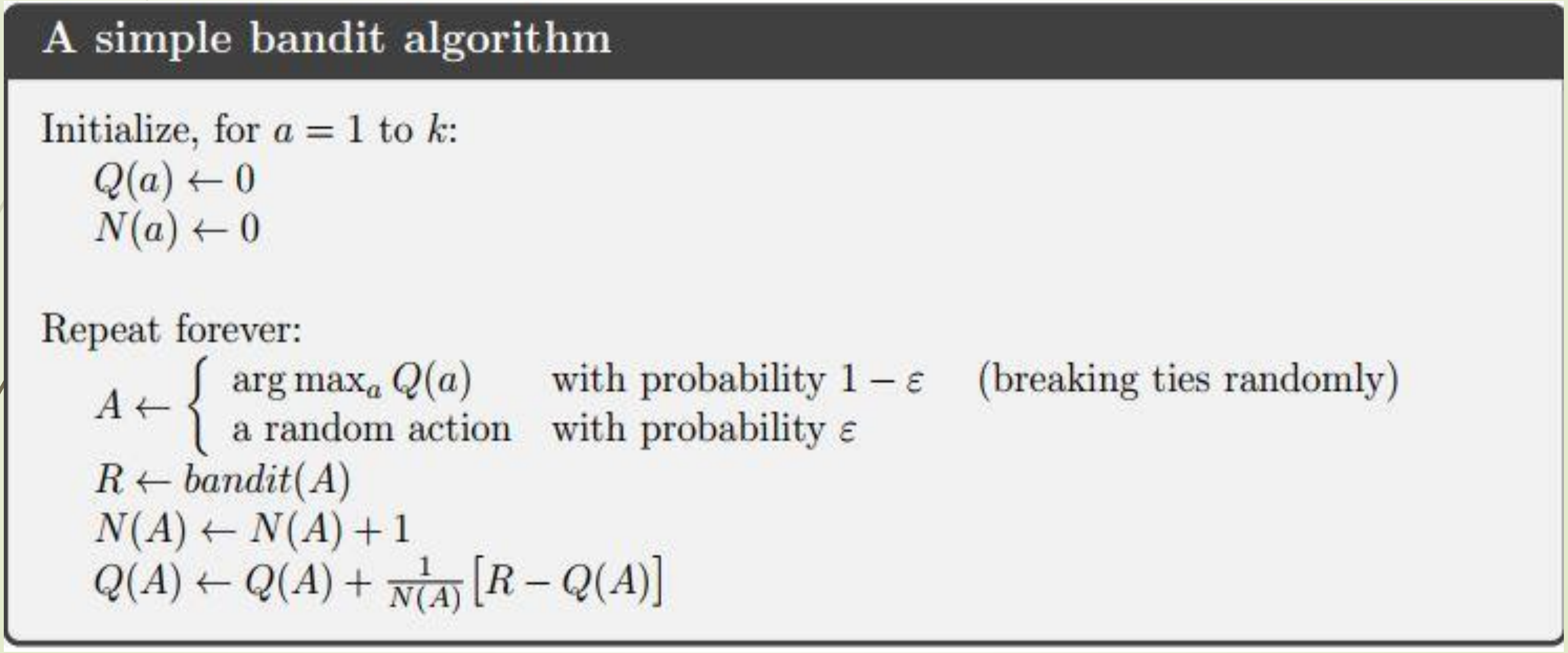
Balancing between Exploration and Exploitation

- Action-Value Methods
- Softmax Action Selection
- Incremental Implementation
- Tracking a Non Stationary Problem
- Optimistic Initial Values
- Associative Search (Contextual Bandits)

Action Value Method

- we denote the true (actual) value of action a as $q_*(a)$, and the estimated value on the tth time step as $Q_t(a)$.
- True Value: True value of an action is the mean reward received when that action is selected.
- How to estimate the reward?
- **$Q_t(a) = R1 + R2 + R3 + \ldots .. + R_{kn}\ / K_a$**
- The simplest action selection rule is to select the action (or one of the actions) with highest estimated action value, that is, to select at step t one of the greedy actions, At for which **$Q_t(A) = \max_a Q_t(a)$**.
- This method always exploits the current knowledge to maximize the immediate reward, it spends no time sampling the rewards.

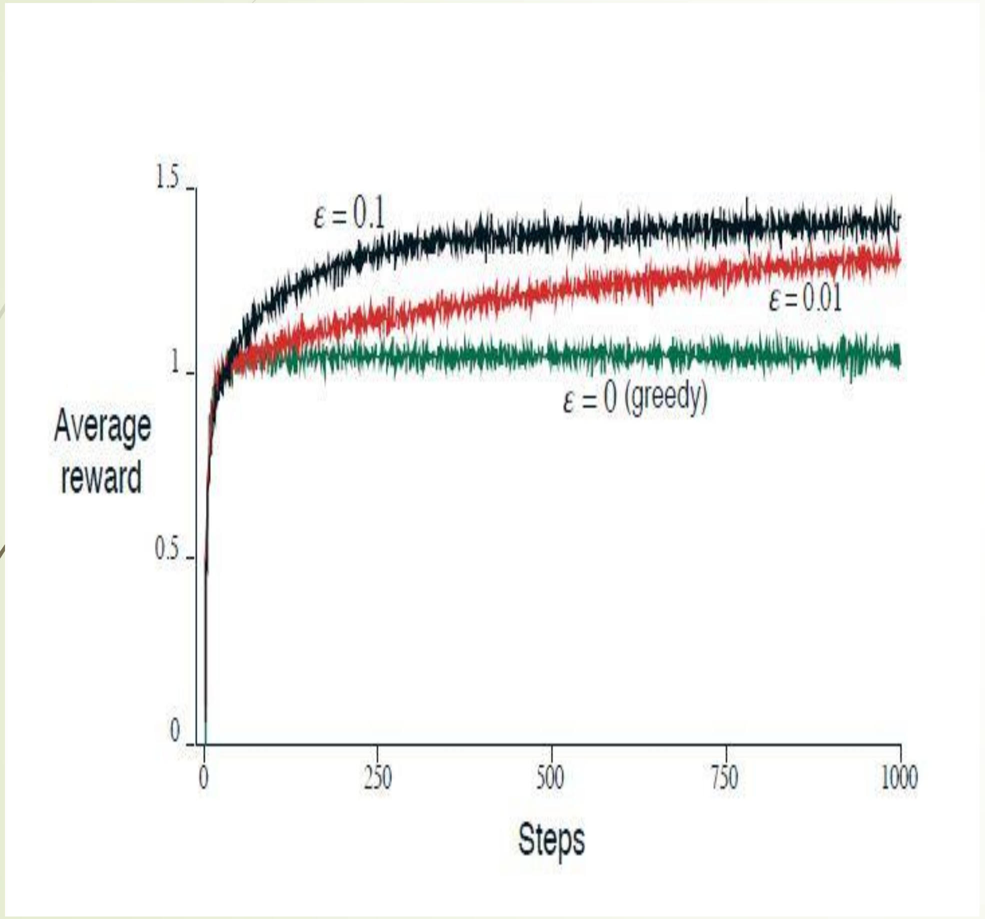
ε-Greedy Algorithm

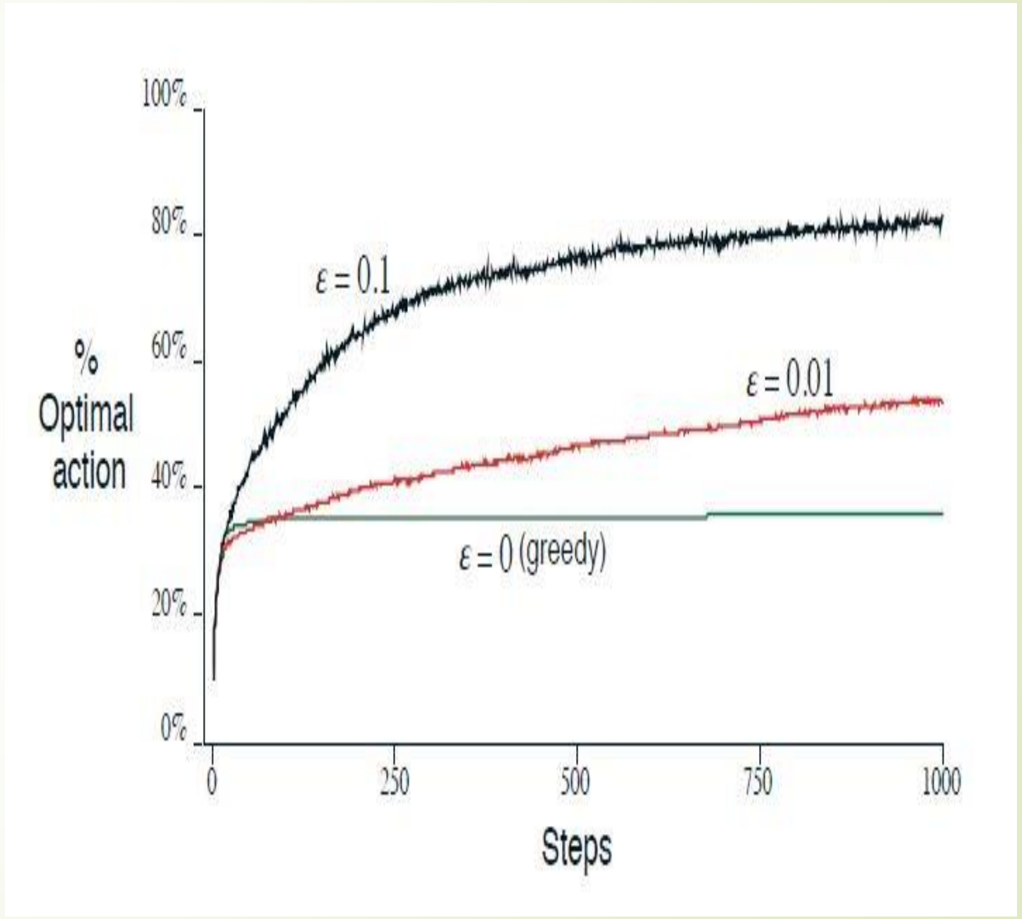


A simple bandit algorithm

```
Initialize, for $a = 1$ to $k$:
    $Q(a) \leftarrow 0$
    $N(a) \leftarrow 0$

Repeat forever:
    $A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
    $R \leftarrow bandit(A)$
    $N(A) \leftarrow N(A) + 1$
    $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$
```


Average reward
1.5
1
0.5
0
ε = 0.1
ε = 0.01
ε = 0 (greedy)
0
250
500
750
1000
Steps


% Optimal action
100%
80%
60%
40%
20%
0%
ε = 0.1
ε = 0.01
ε = 0 (greedy)
0
250
500
750
1000
Steps

Softmax Action Selection

- Epsilon greedy action selection is an effective and popular means of balancing exploration and exploitation in reinforcement learning.

- There is one drawback

1. When it explores it chooses equally among all actions. This means that it is as likely to choose the worst-appearing action as it is to choose the next-to-best action

- What could be the solution?

1. To vary the action probabilities as a graded function of estimated value.

- The greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their value estimates.

- The most common softmax method uses a Gibbs, or Boltzmann, distribution.
- It chooses action a on the tth time step with probability.

The formula is given as:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{i=1}^{n} e^{Q_t(i)/\tau}},$$

e = Exponent of Natural logarithm
T = Positive parameter called Temperature

Incremental Implementation

- In this also we start as Greedy algorithm and count the

$$\mathbf{Q_t(a) = R1 + R2 + R3 + \ldots\ldots + R_{kn} \;/\; K_a}$$

- A problem with this straightforward implementation is that its memory and computational requirements grow over time without bound. That is, each additional reward following a selection of action a requires more memory to store it and results in more computation being required to determine Qt(a).

Now after getting this value we perform the incrementation

$$NewEstimate \leftarrow OldEstimate + StepSize \left[Target - OldEstimate \right].$$

- Average of all k rewards can be computed by

$$\begin{aligned}
Q_{k+1} &= \frac{1}{k}\sum_{i=1}^{k} R_i \\
&= \frac{1}{k}\left(R_k + \sum_{i=1}^{k-1} R_i\right) \\
&= \frac{1}{k}\Big(R_k + (k-1)Q_k + Q_k - Q_k\Big) \\
&= \frac{1}{k}\Big(R_k + kQ_k - Q_k\Big) \\
&= Q_k + \frac{1}{k}\Big[R_k - Q_k\Big],
\end{aligned}$$

Tracking a Non-Stationary problem

- The averaging methods discussed so far are appropriate in a stationary environment, but not if the bandit is changing over time.
- In such cases it makes sense to weight recent rewards more heavily than long-past ones.
- One of the most popular ways of doing this is to use a constant step-size parameter.
- For example, the incremental update rule for updating an average Q_k of the k-1 past rewards is modified to be

$$Q_{k+1} = Q_k + \alpha \Big[R_k - Q_k \Big],$$

Incremental rule for non-stationary problem

where the step-size parameter, alpha , 0 <   1, is constant

$$
\begin{aligned}
Q_{k+1} &= Q_k + \alpha\Big[R_k - Q_k\Big] \\
&= \alpha R_k + (1-\alpha)Q_k \\
&= \alpha R_k + (1-\alpha)\left[\alpha R_{k-1} + (1-\alpha)Q_{k-1}\right] \\
&= \alpha R_k + (1-\alpha)\alpha R_{k-1} + (1-\alpha)^2 Q_{k-1} \\
&= \alpha R_k + (1-\alpha)\alpha R_{k-1} + (1-\alpha)^2 \alpha R_{k-2} + \\
&\qquad\qquad \cdots + (1-\alpha)^{k-1}\alpha R_1 + (1-\alpha)^k Q_1 \\
&= (1-\alpha)^k Q_1 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} R_i.
\end{aligned}
$$

The quantity 1 - alpha is less than 1, and thus the weight given to Ri decreases as the number of intervening rewards increases.

Optimistic Initialization

- Simple and practical idea: initialize Q(a) to high value
- Update action value by incremental Monte-Carlo evaluation
- Starting with N(a) > 0

$$\hat{Q}_t(a_t) = \hat{Q}_{t-1} + \frac{1}{N_t(a_t)}(r_t - \hat{Q}_{t-1})$$

- Encourages systematic exploration early on
- But can still lock onto suboptimal action

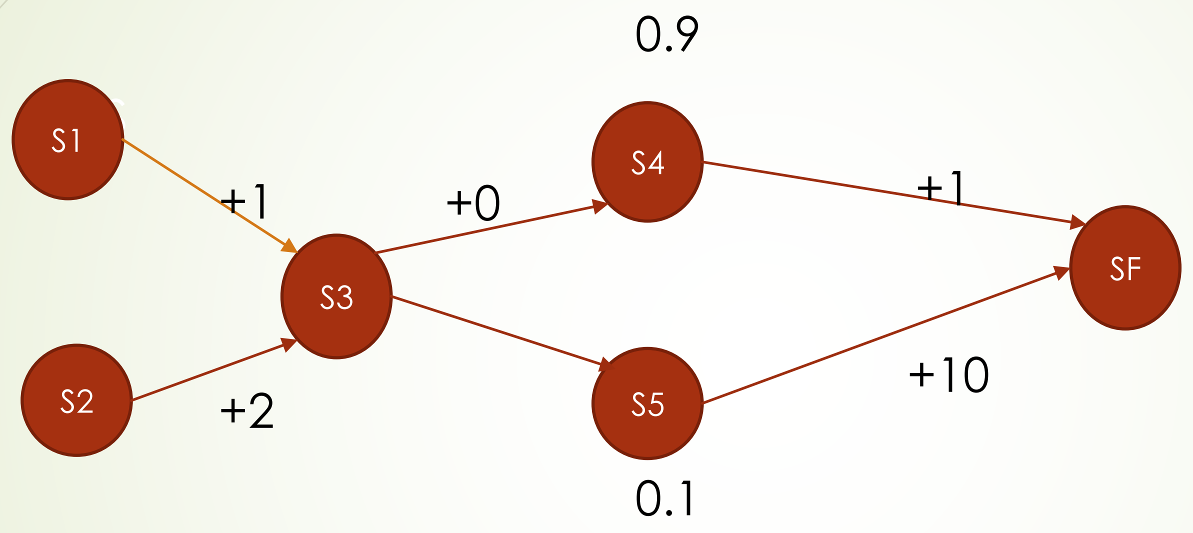
Associative Search

- So far we have considered only non associative tasks, in which there is no need to associate different actions with different situations.
- In these tasks the learner either tries to find a single best action when the task is stationary, or tries to track the best action as it changes over time when the task is nonstationary.

Temporal-Difference Learning

- The central and novel idea of Reinforcement Learning
- Learns a prediction of future rewards from another later learned prediction
- Ultimately learns to make long-term predictions without a model of the environment's dynamics
- Update estimates based in part on other learned estimates, without waiting for a final outcome
- TD error is the difference between two temporally successive predictions, called Temporal Difference

Temporal-Difference Learning



Learn V(s) = $\begin{cases} 0, \text{if } s = s_F \\ E[r + \gamma V(S`)] \text{ otherwise} \end{cases}$

Temporal-Difference Learning (cntd.)

- Target

Given some experience following a policy π, it updates estimate v of vπ for the nonterminal states St occurring in that experience.

A simple Monte Carlo method is

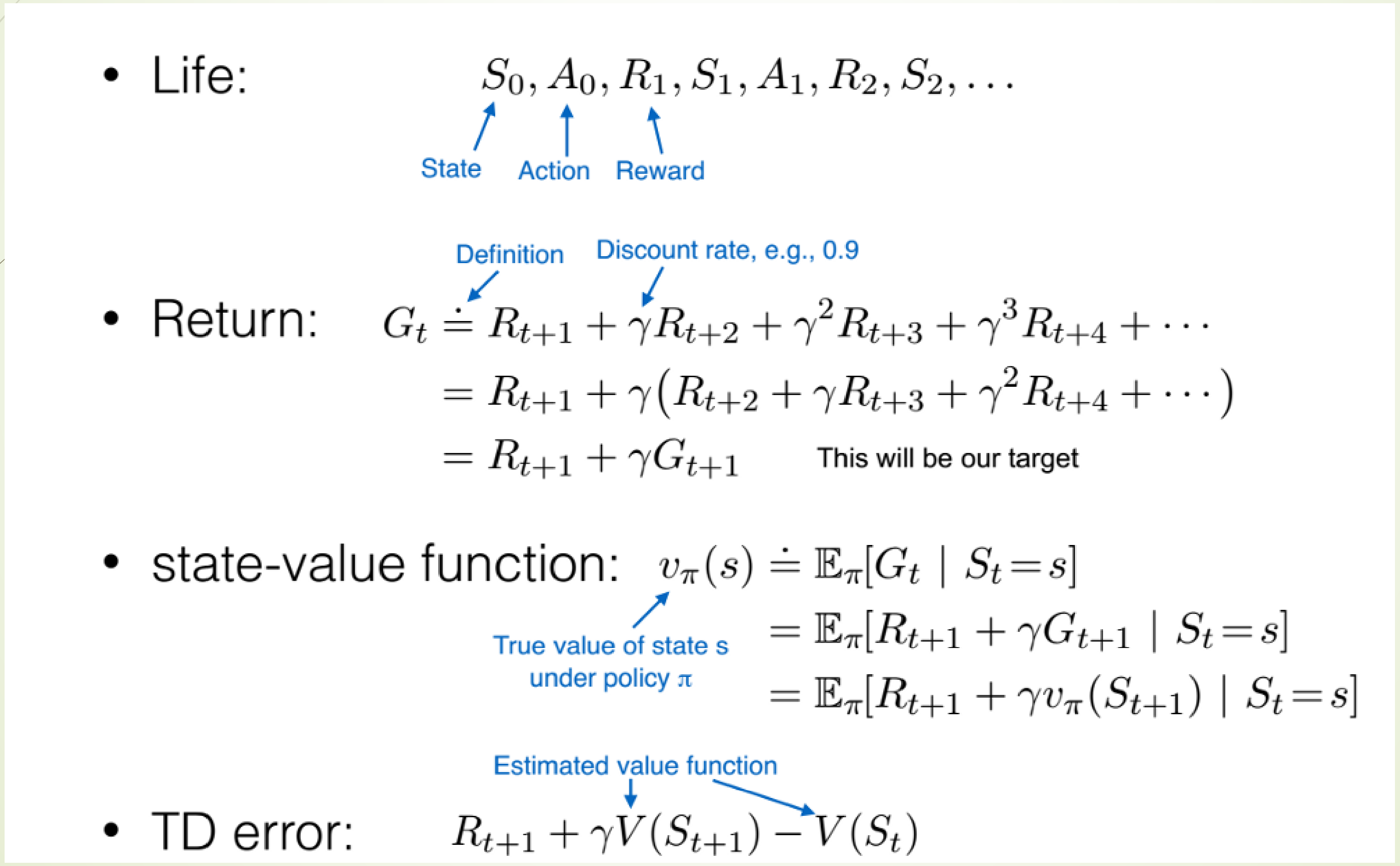
V(St) → V(St) + $\alpha[G(t) - V(St)]$, G(t) is the target

A simple TD-learning method is

V(St) → V(St) + $\alpha[R(t+1) + \gamma V(St+1) - V(St)]$,

$R(t+1) + \gamma V(St+1)$ is the target

Temporal-Difference Learning (cntd.)



- Life: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$

  State Action Reward

- Return:

  Definition Discount rate, e.g., 0.9

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\big(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\big) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

  This will be our target

- state-value function:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned}$$

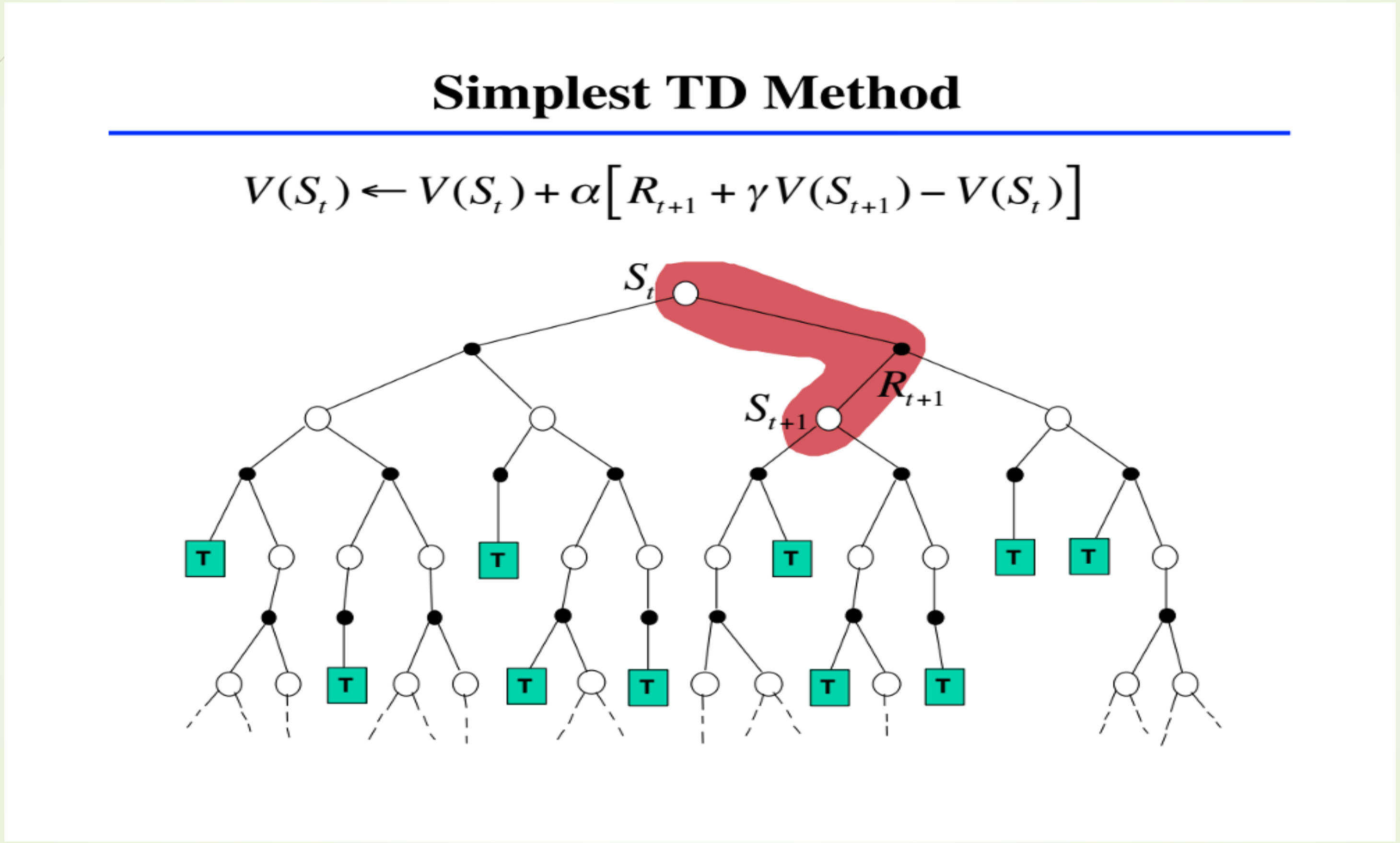
  True value of state s under policy π

- TD error:

  Estimated value function

$$R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

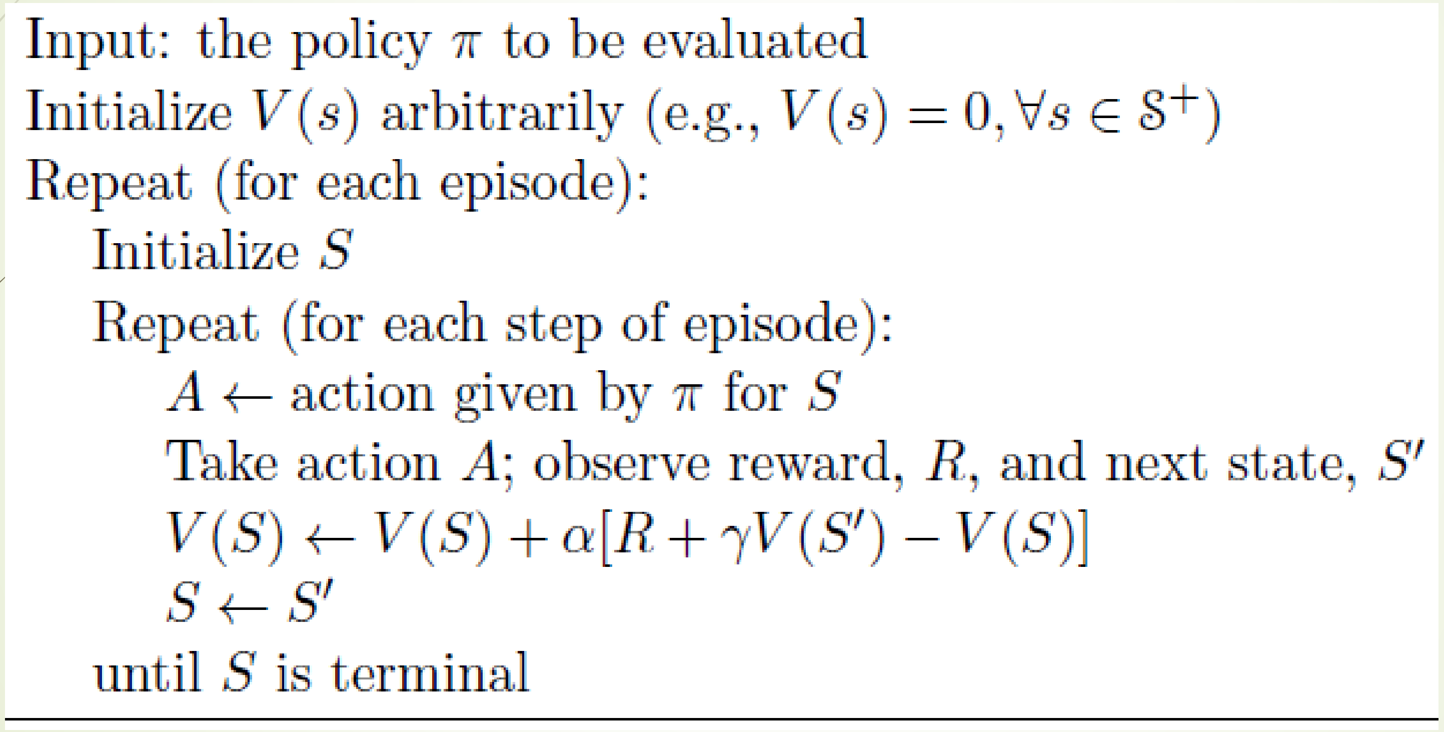
Simplest TD method - TD (0)


Simplest TD Method

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

TD method: state value V(St) is updated (backed up) with immediate reward R(t+1) and discounted value of the next state—V(St+1)

TD Prediction using TD (0)

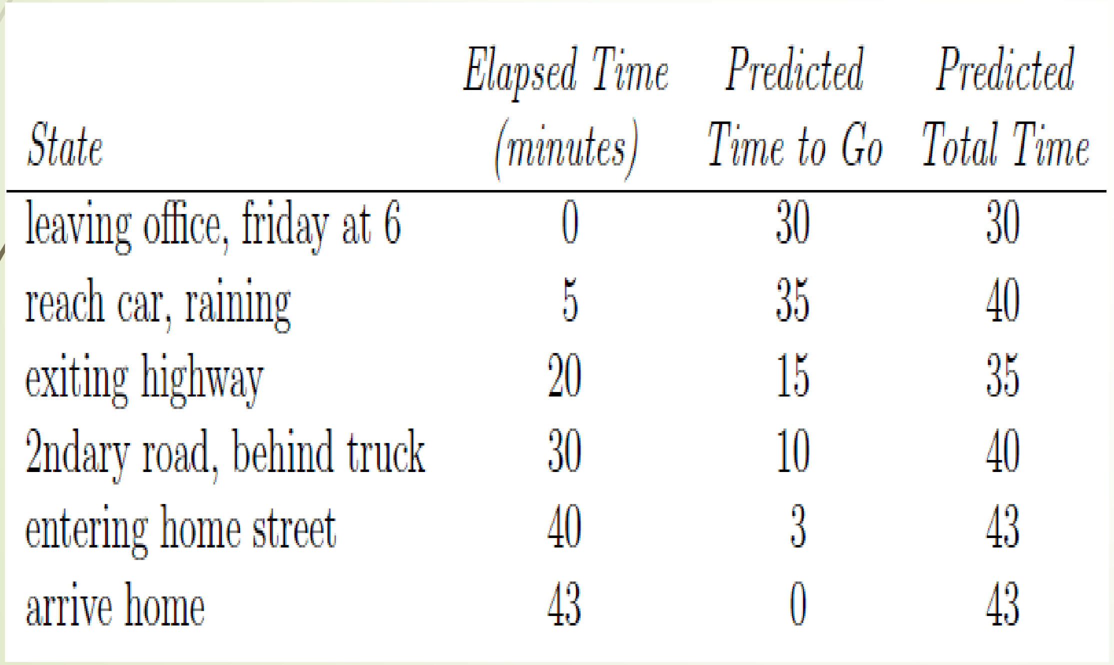


```
Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$; observe reward, $R$, and next state, $S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

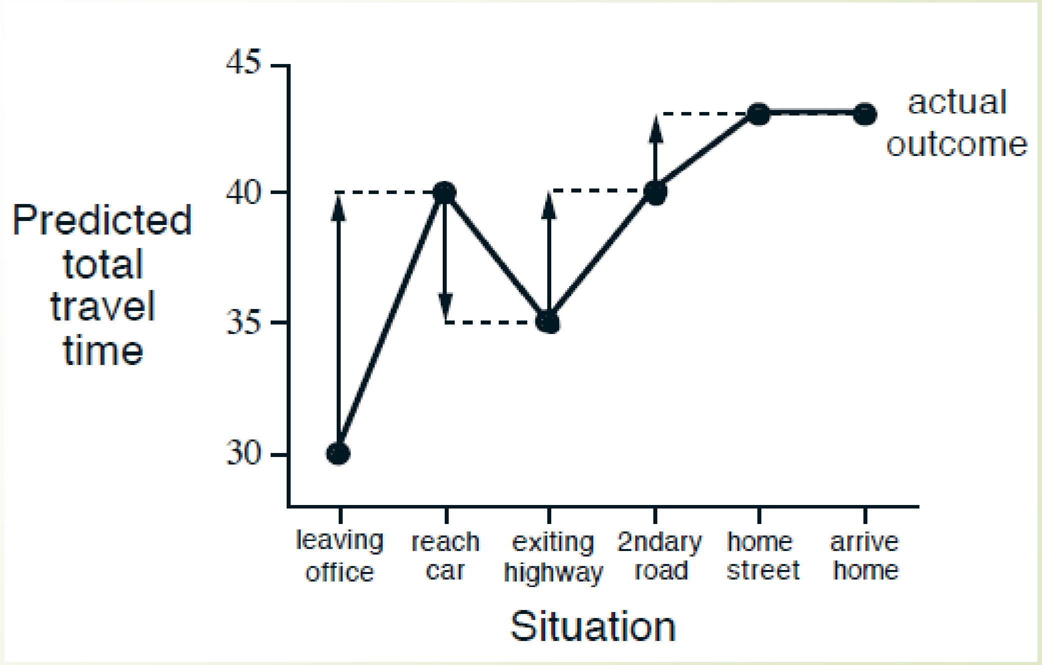
Tabular TD(0) for estimating policy v_π

TD Prediction using TD (0)

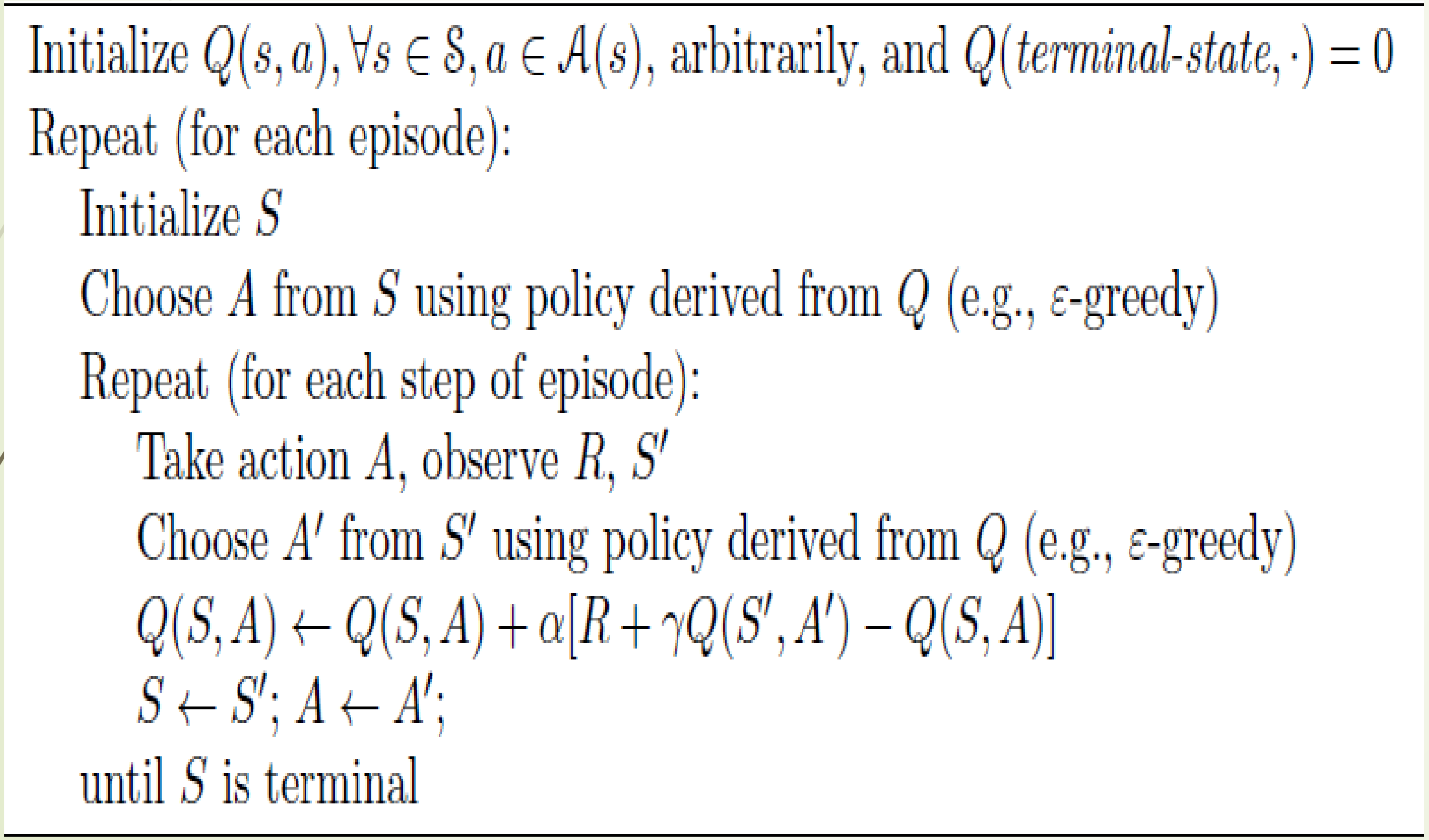
The Driving Home Problem



State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43



TD Control using SARSA

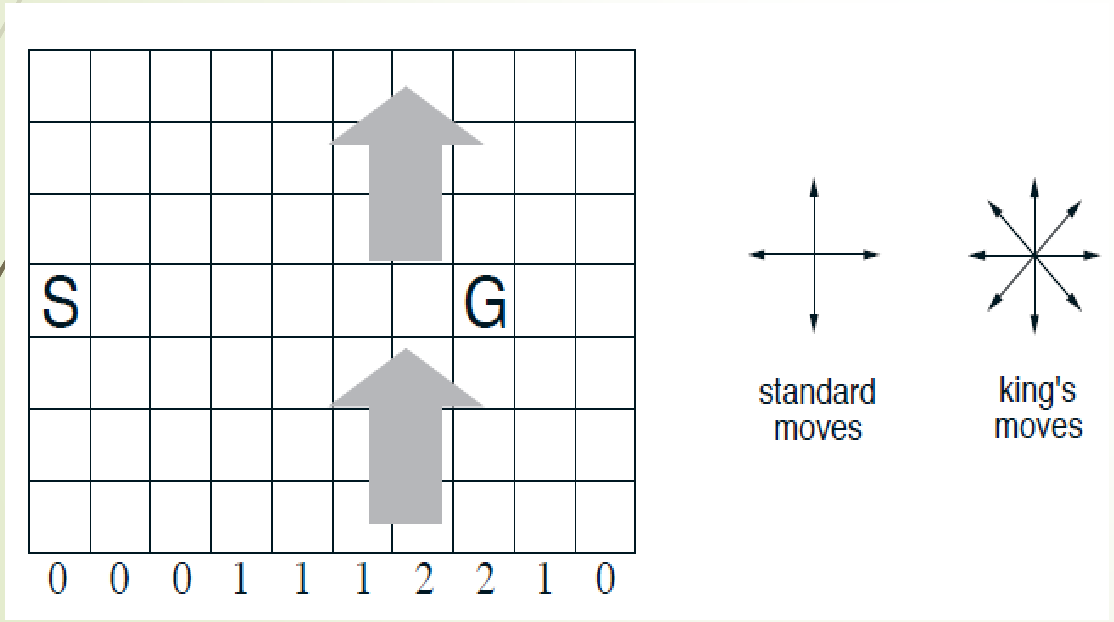


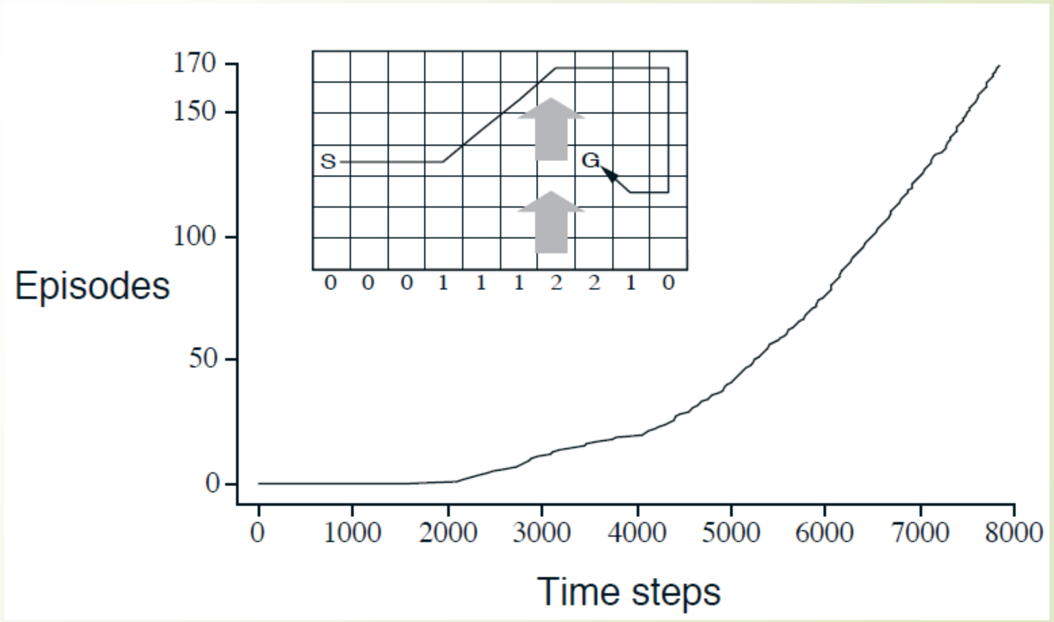
```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Quintuple of events,
$(S_{t+1}, A_{t+1}, R_{t+1}, S_t, A_t)$
S A R S A

TD Control using SARSA

The Windy Gridworld Problem





When to use TD-Learning

Multi-step prediction problems

- Only when the reward predicted is multiple steps in future
- And each step reveals some information about the final prediction

Examples

- Predicting outcome of a game, like chess or backgammon
- Predicting what a stock market index will be at the end of a year

Challenges of RL

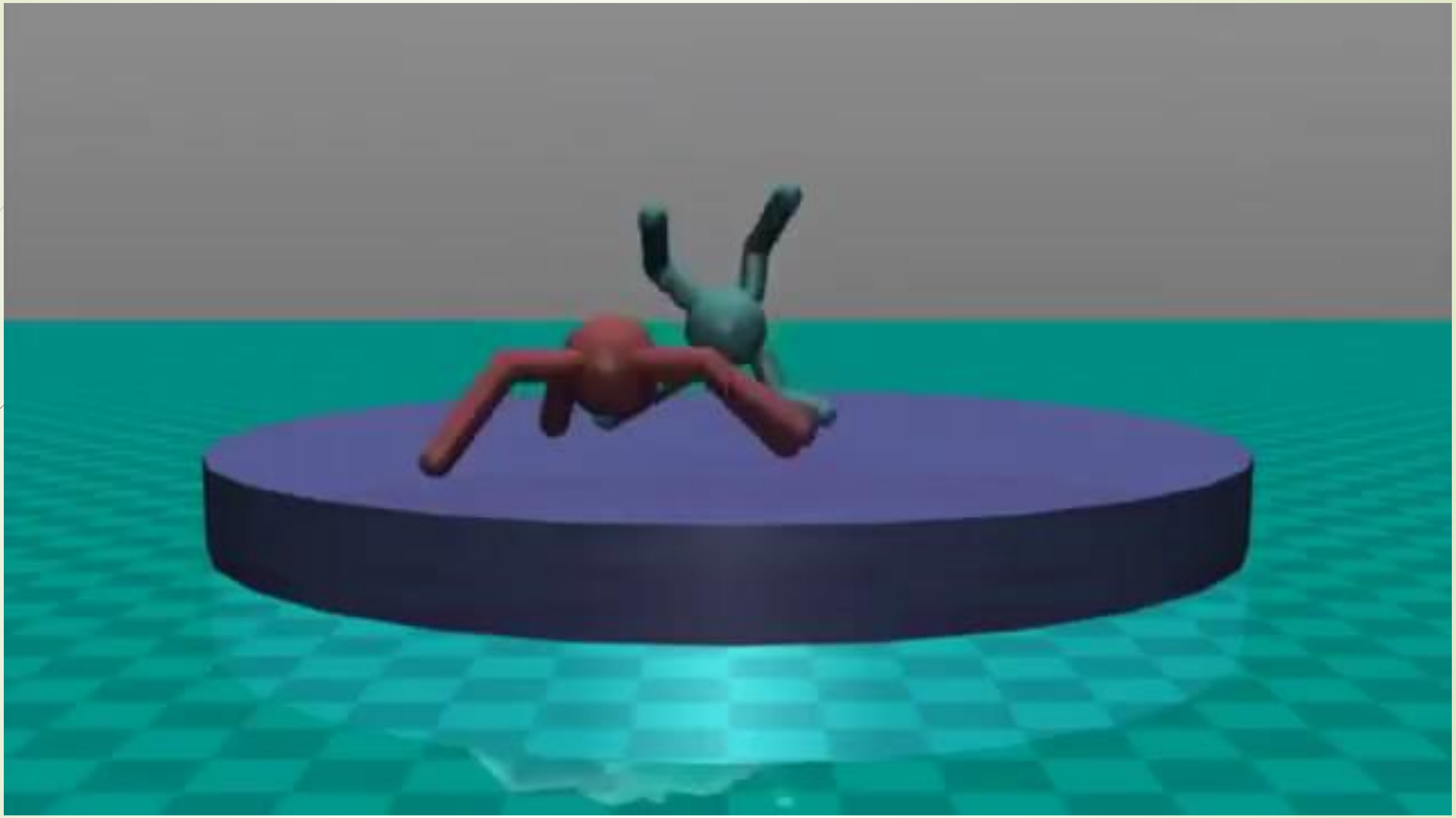
- Time-consuming training procedure
- The optimal policy must be inferred by trial-and-error interaction with the environment
- The only learning signal the agent receives is the reward
- Convergence and parameters optimization is not guaranteed in continuous reinforcement learning
- Credit assignment problem – a common problem where we are not sure how to decide which action an AI took contributed to the reward it received

Recent Success in RL

Google's DeepMind trained AI AlphaGO beats world champion Lee Sedol in GO

Ongoing researches

- Deep Reinforcement Learning
- Multi-agent Reinforcement Learning
- Transfer learning – transferring previously acquired knowledge from one task to another related task
- Imitation Learning and behavioral cloning – adaptation to new situations



Conclusion

Scientists are hoping that integrating Reinforcement Learning techniques with other traditional AI approaches will create the general-purpose AI systems that can interact with and learn from the world around them. Despite of having some limitations, RL endows agents with the ability to perform experiments to better understand their surroundings and learn even high-level causal relationships. Perhaps we are not too far away from AI systems that can learn and act in more human-like ways in increasingly complex environments.

Reference

[1] Sutton, R. S., & Barto, A. G. (2012). *Introduction to reinforcement learning*. Cambridge, Mass: MIT Press.

[2] Jaderberg, M., Mnih, V., Czarnecki, W. M., & M Schau, T. (2016). *Reinforcement learning with unsupervised auxiliary tasks*. ArXiv:1611.05397v1.

[3] Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep Reinforcement Learning: A Brief Survey. IEEE Signal Processing Magazine, 34(6), 26-38. doi:10.1109/msp.2017.2743240

[4] http://www.cs.cmu.edu/~rsalakhu/10703/Lecture_Exploration.pdf

[5] https://sites.google.com/view/multi-agent-competition