Faculty of SET / School of Computer and Mathematical Sciences

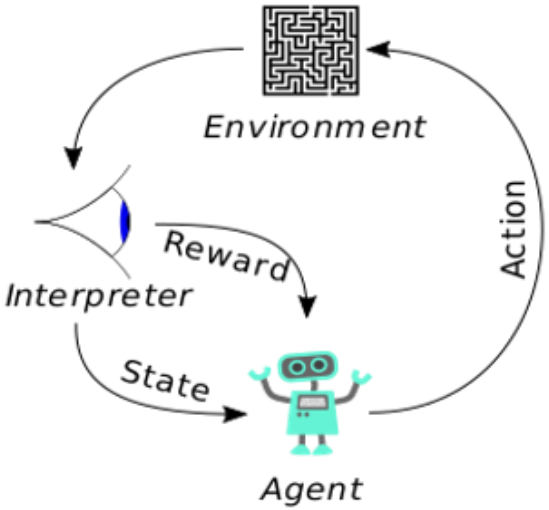
COMP SCI 3007&7059 Artificial Intelligence
Reinforcement Learning

Acknowledgement of Country

We acknowledge and pay our respects to the Kaurna people, the traditional custodians whose ancestral lands we gather on.

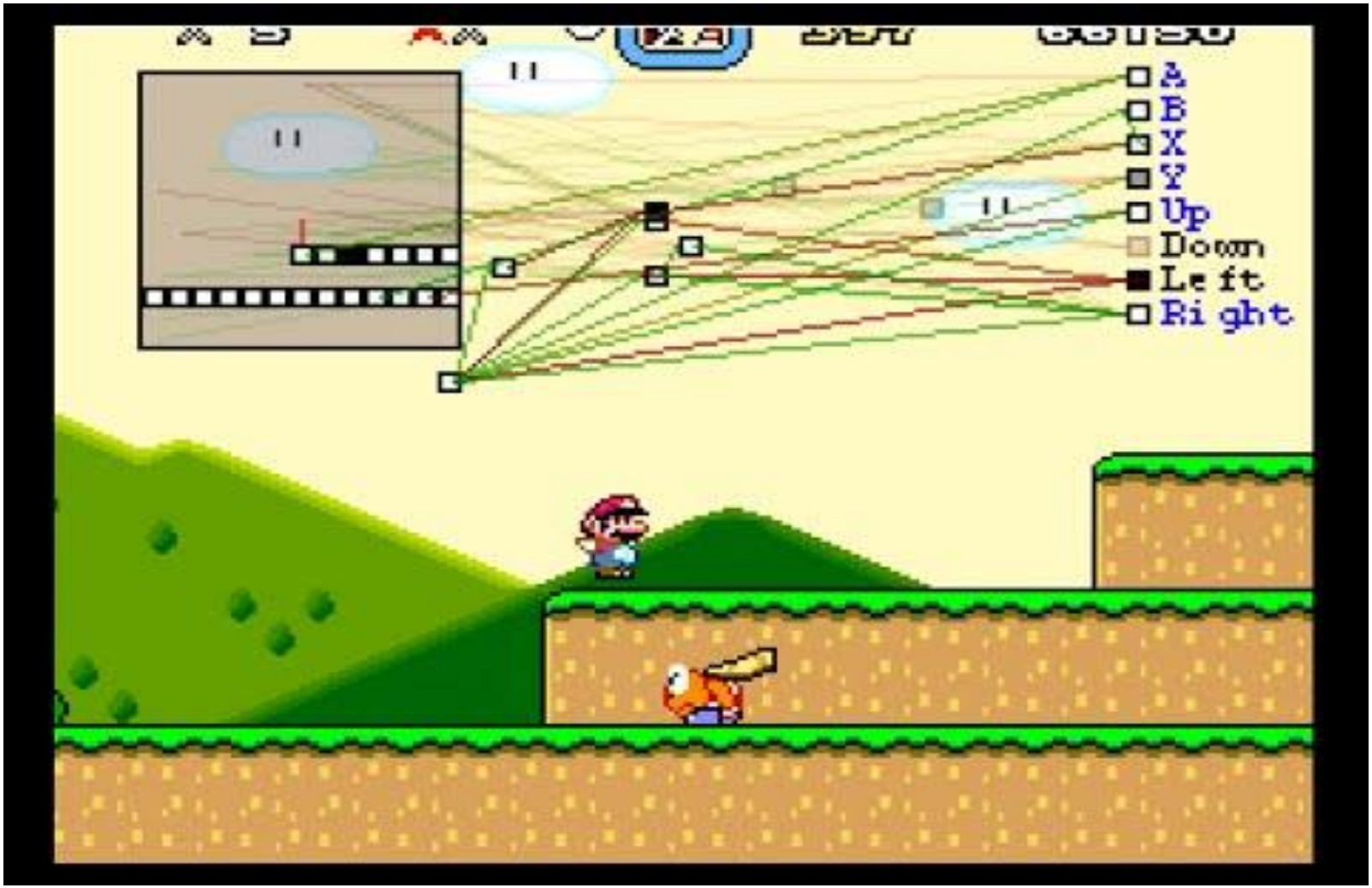
We acknowledge the deep feelings of attachment and relationship of the Kaurna people to the country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

Reinforcement Learning (RL) Basics



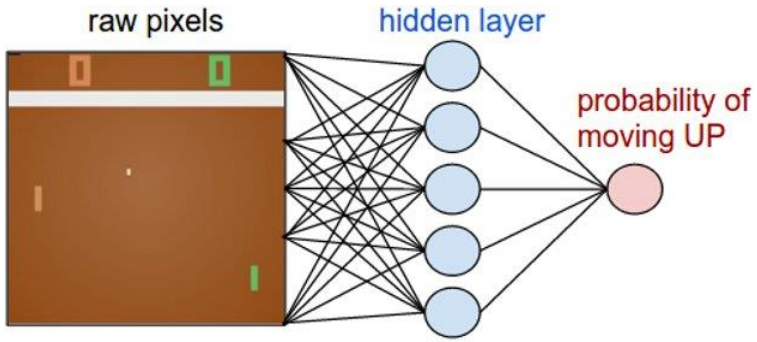
- State: observation of the current environment
- Action: reaction of an agent
- Reward: returns from the environment
- Objective: **maximize cumulative reward**
- Similar with human: observing by eyes ---> judging by brain ---> taking actions by hands

An Example for State/Action/Reward

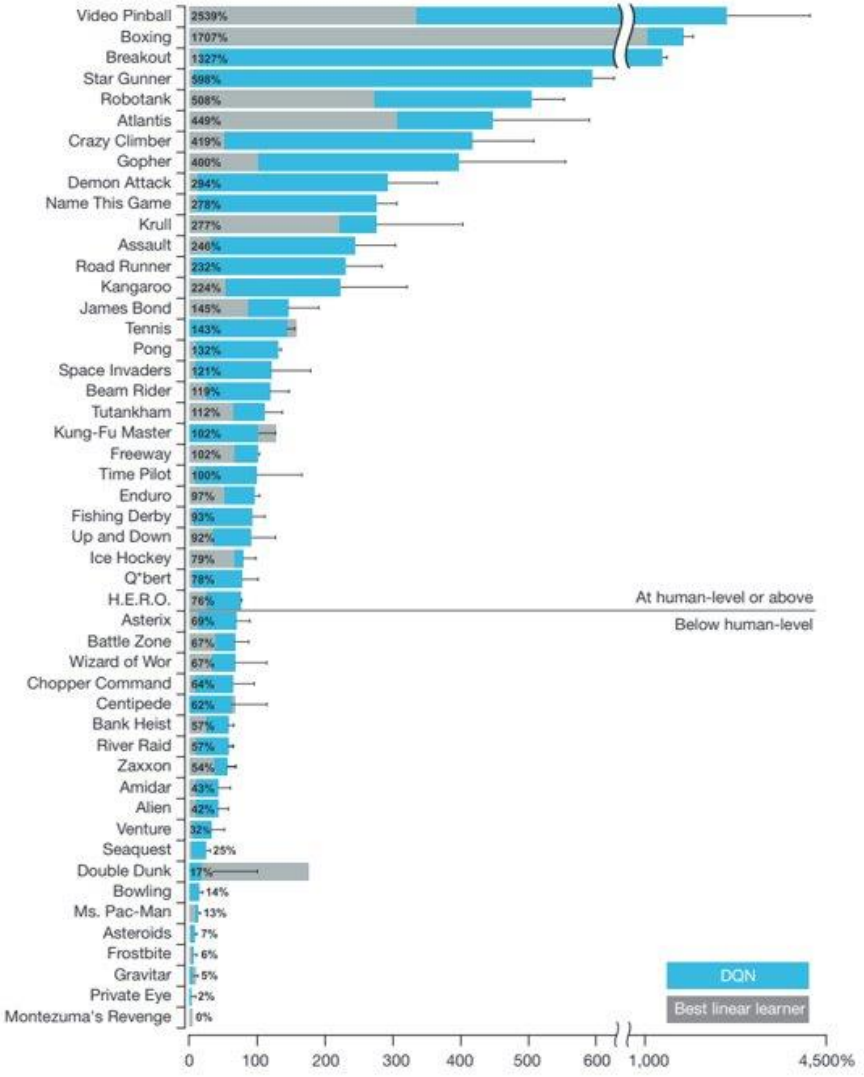


https://www.youtube.com/watch?v=qv6UVOQ0F44

Well-performed on Atari 2600 Games

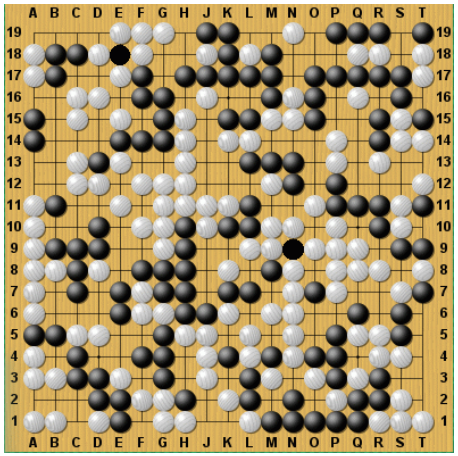


State: Raw Pixels
Actions: Valid Moves
Reward: Game Score

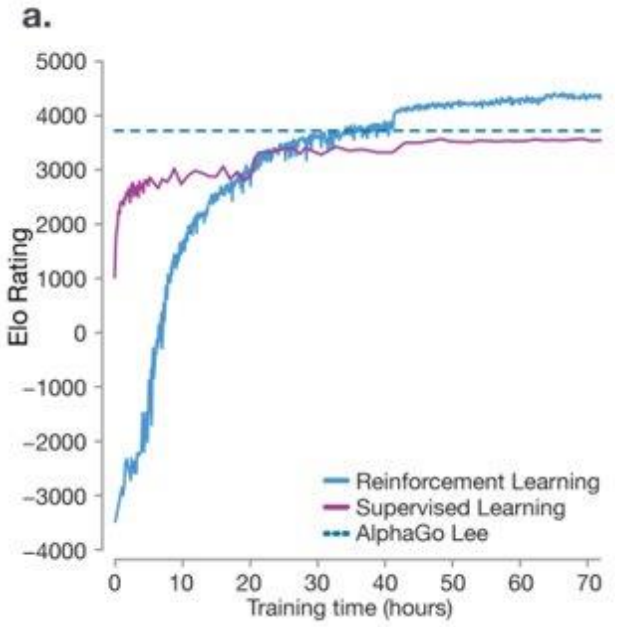


Well-performed on ‘Hard’ Games --- Go

- Learning how to beat humans at ‘hard’ games (super large search space)
- Received better results than Supervised learning
- Algorithm learned to outplay humans at chess in 24 hours



State: Board State
Actions: Valid Moves
Reward: Win or Lose



Well-performed on Robotics and Locomotion

State:
- Joint States/Velocities
- Accelerometer/Gyroscope
- Terrain

Actions: Apply Torque to Joints
Reward: Velocity, whether falls down, etc.

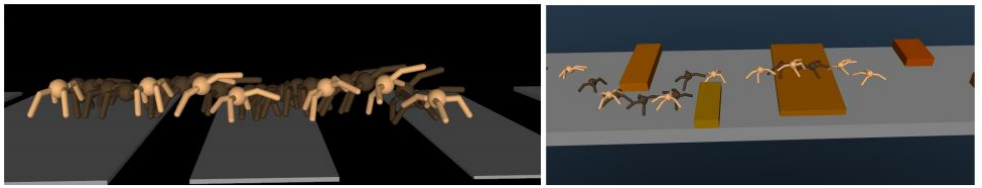
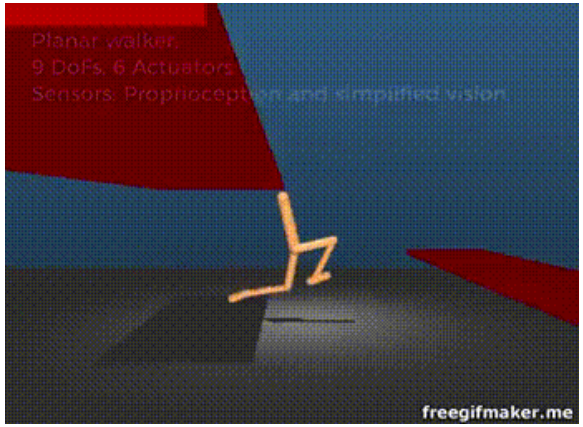
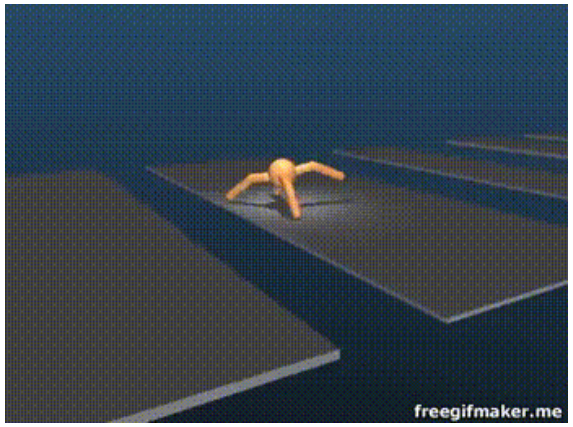
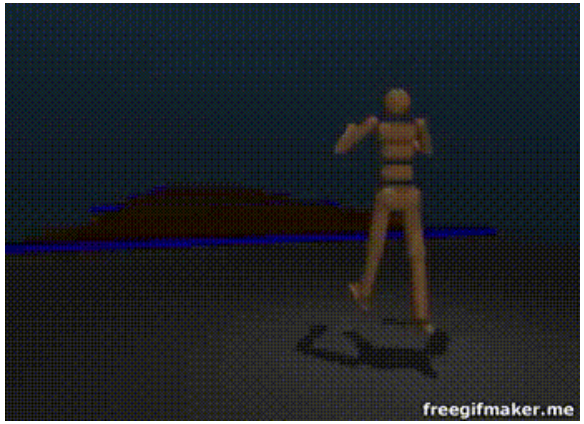


Figure 5: Time-lapse images of a representative *Quadruped* policy traversing gaps (left); and navigating obstacles (right)





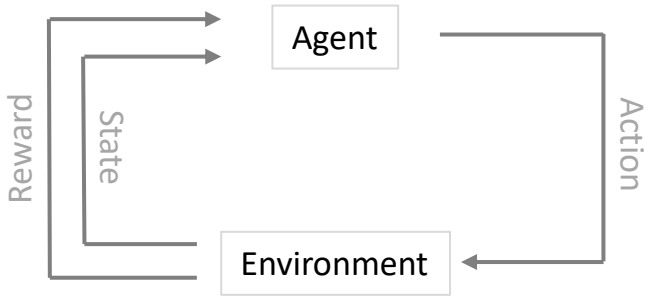


2017 paper https://arxiv.org/pdf/1707.02286.pdf

https://youtu.be/hx_bgoTF7bs

Differences of RL with Supervised Learning

- Learning to interact with an environment
  - Robots, games, process control
  - Without much human labels
  - Where the 'right thing' isn't obvious



- Supervised Learning:
  - Goal: $f(x) = y$
  - Data: $[< x_1, y_1 >, \ldots, < x_n, y_n >]$

- Reinforcement Learning:
  - Goal:

  Maximize $\sum_{i=1}^{\infty} \text{Reward}(\text{State}_i, \text{Action}_i)$

  - Data:

  $\text{Reward}_i, \text{State}_{i+1} = \text{Interact}(\text{State}_i, \text{Action}_i)$

Markov Decision Process (MDP)

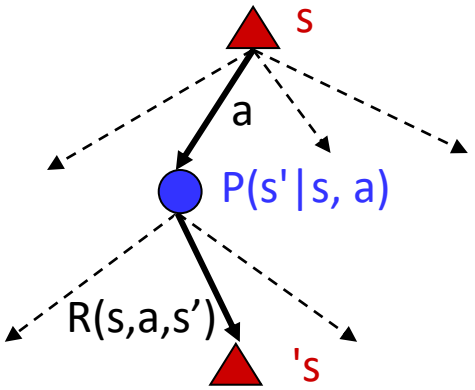
- Andrey Markov (1856-1922)
- "Markov" generally means that given the present state, the future and the past are independent
- For Markov decision processes, "Markov" means:

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0) =$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Terms in Markov Decision Process (MDP)

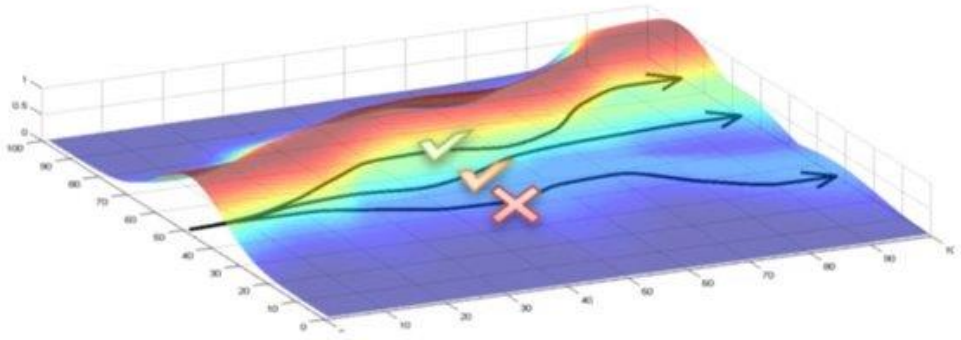
- Markov decision processes:
  - Current state s
  - The next state s'
  - Current actions a
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s')



- MDP quantities so far:
  - Policy = Choice of action for each state (If we do not consider policies, MDP can be generally considered as the environment)
  - Return (or utility) = sum of discounted rewards

Policy

- Policy determines the strategy that the agent adopted to interact with the world
- A policy is a mapping from the perceived states of the environment to actions to be taken when in those states
- An agent uses a policy to select actions given the current environment state
- For example, if the following 3 trajectories represent 3 policies, the yellow one > green one > blue one



Reward Discounting

- Infinite horizon
  - If we did not discount the reward accumulated from previous steps, Policy & value iteration fail to converge
  - Rewards in the future are worth less than an immediate reward
  - Because of uncertainty: who knows if/when you're going to get to that reward state

- Solution: discounting

$$R_t = \sum_{k=t}^{T} \gamma^{(k-t)} r_k(s_k, a_k)$$

Reward Discounting --- An Example

- Discount factor $\gamma \leq 1$ (often $\gamma = 0.9$)
- Assume reward n years in the future is only worth $(\gamma)^n$ of the value of immediate reward

  - $(0.9\text{^}6) * 10{,}000 = 0.531 * 10{,}000 = 5310$

- For each state, calculate a *utility* value equal to the *Sum of Future Discounted Rewards*

Value Iteration --- To Learn the Value Function

- Idea:
  - Start with $V_0^*(s) = 0$
  - Given V_i^*, calculate the values for all states for depth i+1:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right]$$

  - This is called a value update or Bellman update
  - Repeat until convergence
  - Inspired by the idea of Dynamic Programming

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Q-Learning

- Q-Learning: Q-value iteration
- Learn Q*(s,a) values --- expected accumulated rewards
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

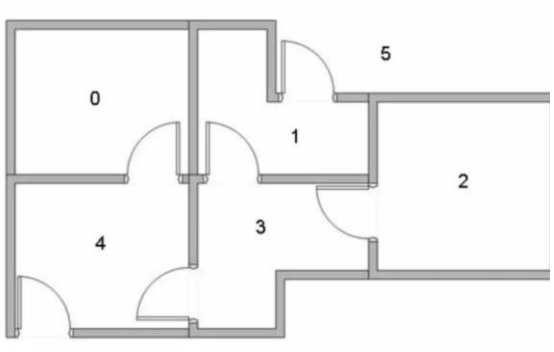
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]$$

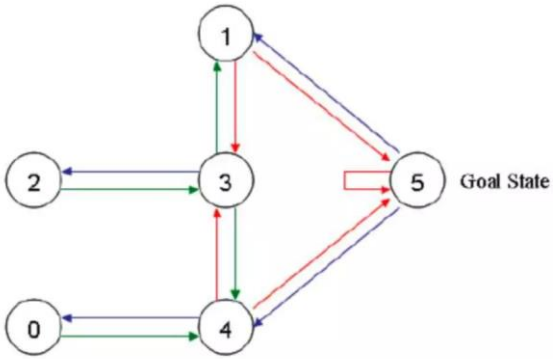
$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

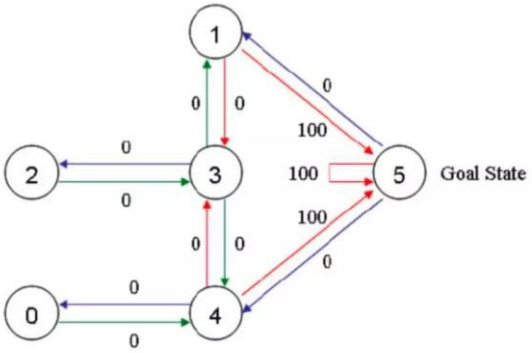
  - Usually incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) \left[sample \right]$$

Q-Learning by Hand

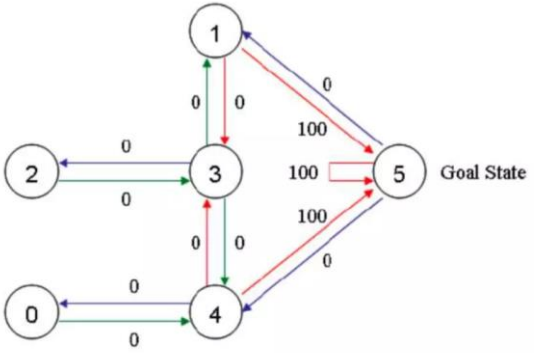






- Q learning is based on the idea of Dynamic Programming (DP)
- If the robots can get out of the house then succeed, and the reward (shown on the edges) to location 5 is set to 100; otherwise, the rewards are set to 0
- State: the agent currently in which room
- Action: from which room to another room

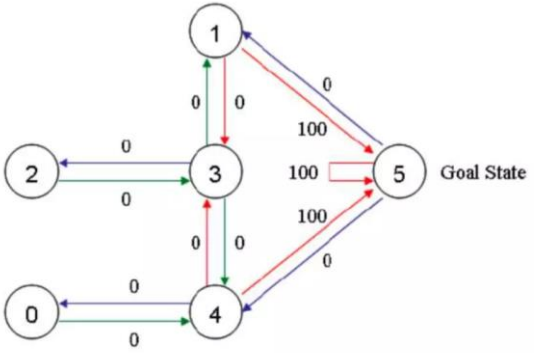
Q-Learning by Hand



$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array} \qquad Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- Instant reward matrix: 0 --- can be passed through; -1 --- cannot be passed through; 100 --- success
- Initial a Q table with the same rank of R matrix, but with all zeros
- Reward discount factor $\gamma = 0.8$

Q-Learning by Hand



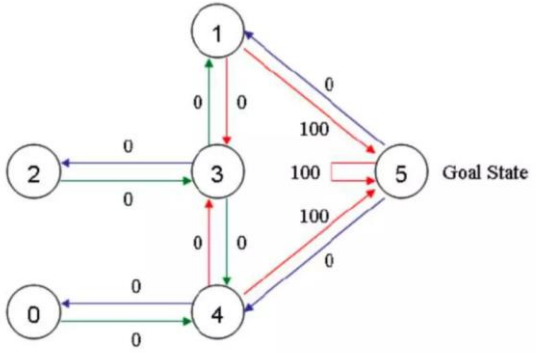
$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array} \qquad Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- Randomly choose a state as initialization, for example room 1. The agent can reach room 3 or room 5. Randomly, if we choose 5, then we update the Q table as:

$$Q(s,a) = R(s,a) + \gamma \cdot \max_{\tilde{a}}\{Q(\tilde{s},\tilde{a})\}$$

$$\begin{aligned} Q(1,5) &= R(1,5) + 0.8 * \max\{Q(5,1), Q(5,4), Q(5,5)\} \\ &= 100 + 0.8 * \max\{0,0,0\} \\ &= 100. \end{aligned}$$

Q-Learning by Hand



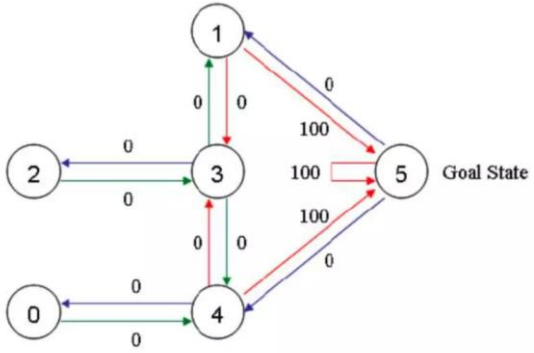
Action

$$R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array} \qquad Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- Then we choose a state randomly again, like room 3. If in room 3, the agent can go to room [1, 2, 4], and we random choose room 1:

$$\begin{aligned} Q(3,1) &= R(3,1) + 0.8 * \max\{Q(1,3), Q(1,5)\} \\ &= 0 + 0.8 * \max\{0, 100\} \\ &= 80. \end{aligned}$$

Q-Learning by Hand



Action

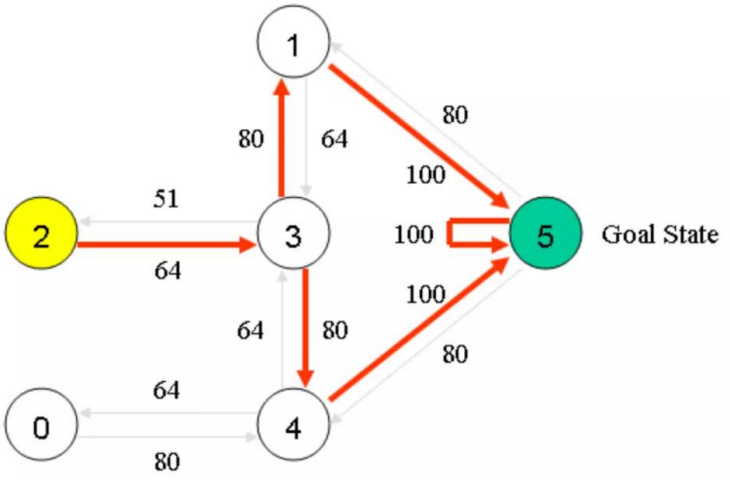
$$R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

- After many iterations, our final Q table is shown as right one
- So we can choose our action based on our Q table

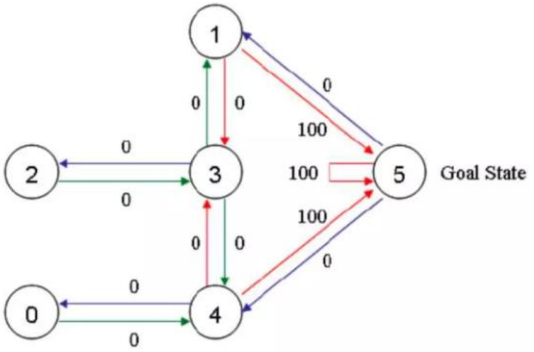
Q-Learning by Hand

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

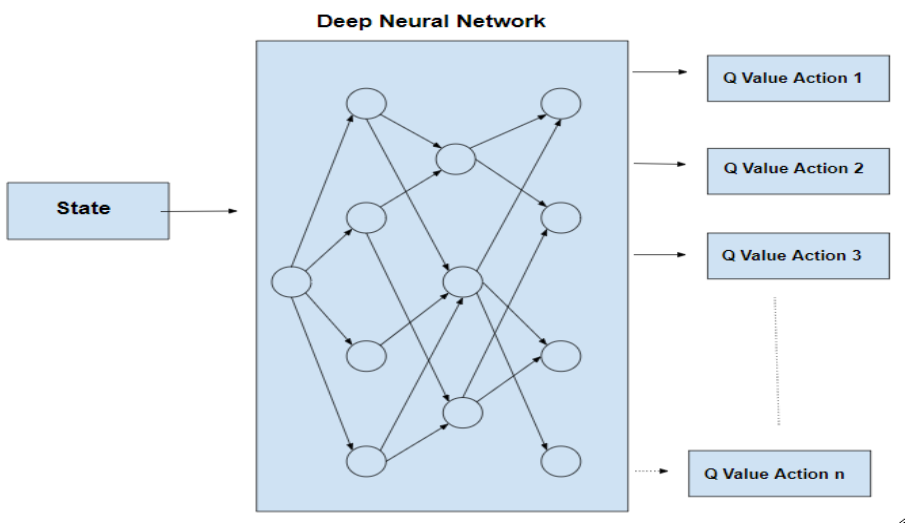


- For example, if we initialize the agent in room 2
- The correct path is: 2 --> 3 --> 1 or 4 --> 5
- Q-value represents the expected accumulative reward of an action towards to the end

Q-Learning by Hand

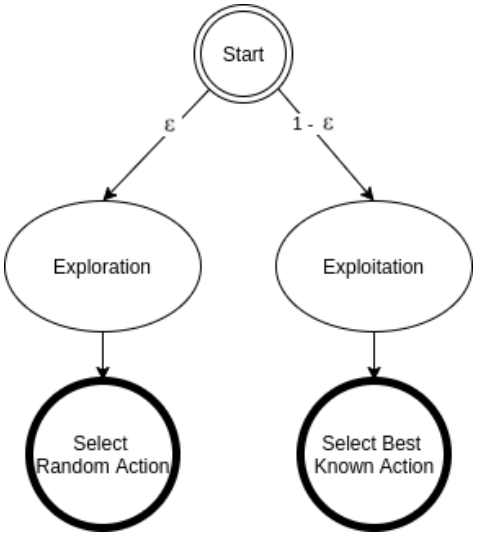


$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$



- The Q table works well for our 'work out of the house' toy example
- But in our real-world, the states are usually continuous and can be infinite many number, e.g. in Super-Mario game, each frame is a state
- In the case, now we usually use a deep neural network to present the Q table

Action Exploration --- Epsilon-Greedy



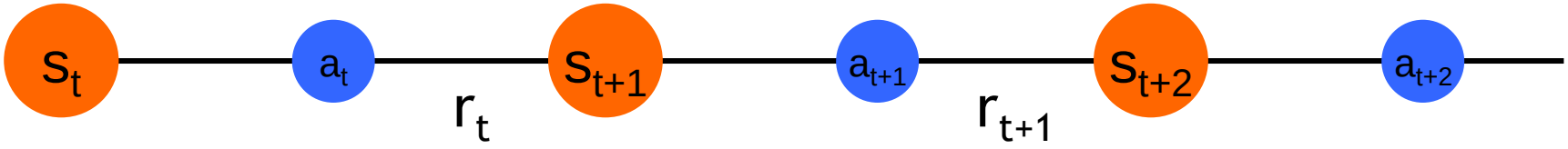
- Initially, select a small positive number between (0, 1), as epsilon.
- Each time randomly generate a real number, if it smaller than the epsilon, select an action randomly; otherwise, the action with the largest Q value is selected.
- The closer epsilon to 0, the more conservative; vice versa, the more radical.

On-policy V.S. Off-policy

- An off-policy agent learns about a policy or policies different from the one that it is executing
- An on-policy agent learns only about the policy that it is executing
- Q-Learning is off-policy
- Sarsa in the next slide is on-policy

Sarsa Algorithm

- Very similar to Q-Learning, we need Q(s,a), not just V(s)



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- Learning process
  - start with a random policy
  - update Q and π after each step
  - again, need ε-soft policies

Q-Learning V.S. Sarsa

- Q-learning: off-policy
  - use any policy to estimate Q

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

  - Q directly approximates Q* (Bellman optimality eqn)
  - independent of the policy being followed
  - only requirement: keep updating each (s,a) pair
  - Learn from history (s,a) pairs

- Sarsa: on-policy algorithms
  - start with a random policy, iteratively improve
  - converge to optimal
  - Cannot use history (s,a) pairs

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Policy Iteration --- Optimise the Value/Policy Functions

- policy evaluation: π -> V^{π}
  - Bellman equation defines a system equations
  - Could be solved, but we adopt iterative version

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{k'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

  - Start with an arbitrary value function V_0, iterate until V_k converges

- policy improvement: V^{π} -> π'

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$
$$= \arg\max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^{\pi}(s') \right]$$

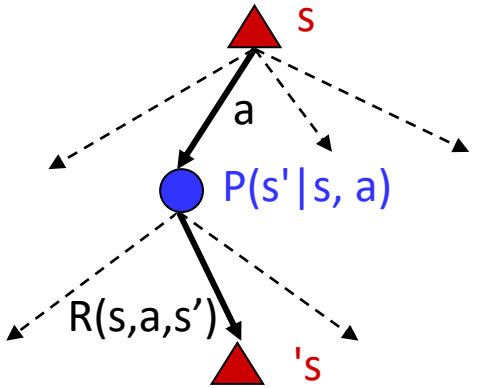
  - π' either strictly better than π, or π is optimal (if $\pi = \pi'$)

Policy Iteration V.S. Value Iteration

- Policy iteration

$$\pi_0 \rightarrow^E V^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E V^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E V^*$$

  - π represents a policy
  - two nested iterations; relatively slow
    - Evaluation (represented by E)
    - Improvement (represented by I)

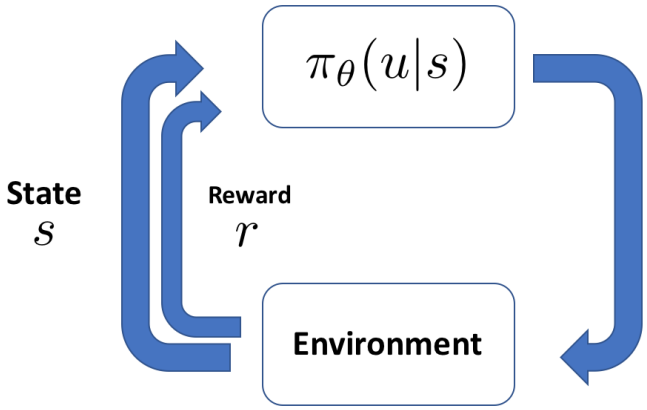


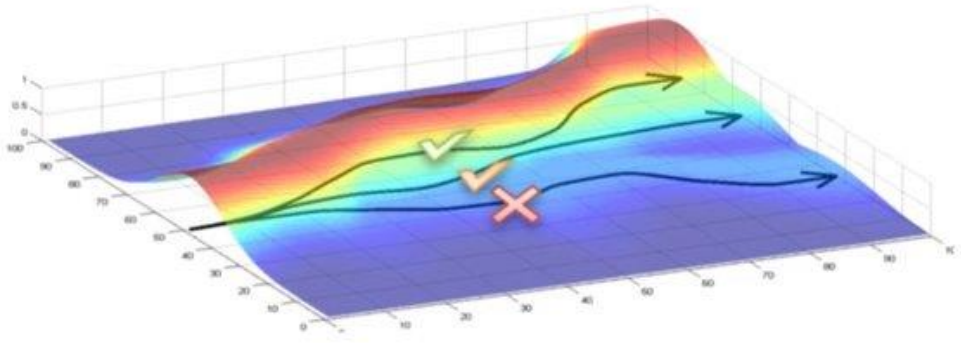
- Value iteration

$$V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

  - use Bellman optimality equation as an update
  - converges to V*

Policy Gradient --- Intuition and Formulation





- Optimization intuition
  - If an action leads to larger rewards ---> increase the probability
  - Otherwise ---> decrease the prob

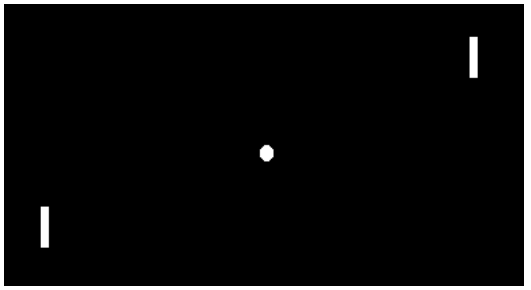
$$Policy\ gradient: E_\pi[\nabla_\theta(log\pi(s,a,\theta))R(\tau)]$$

Policy function — Score function

$$Update\ rule: \Delta\theta = \alpha * \nabla_\theta(log\pi(s,a,\theta))R(\tau)$$

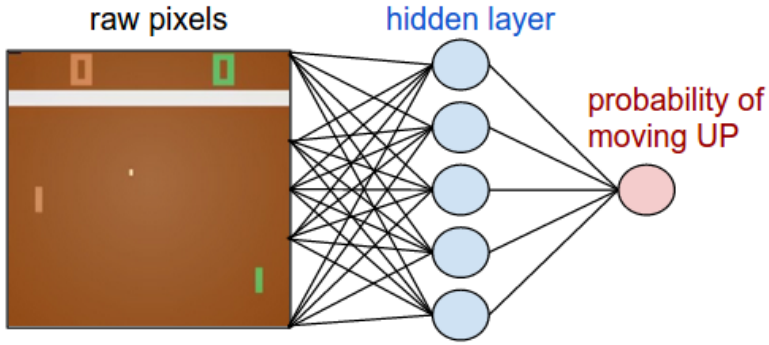
Change in parameters — Learning rate

Q-Learning V.S. Policy Gradient



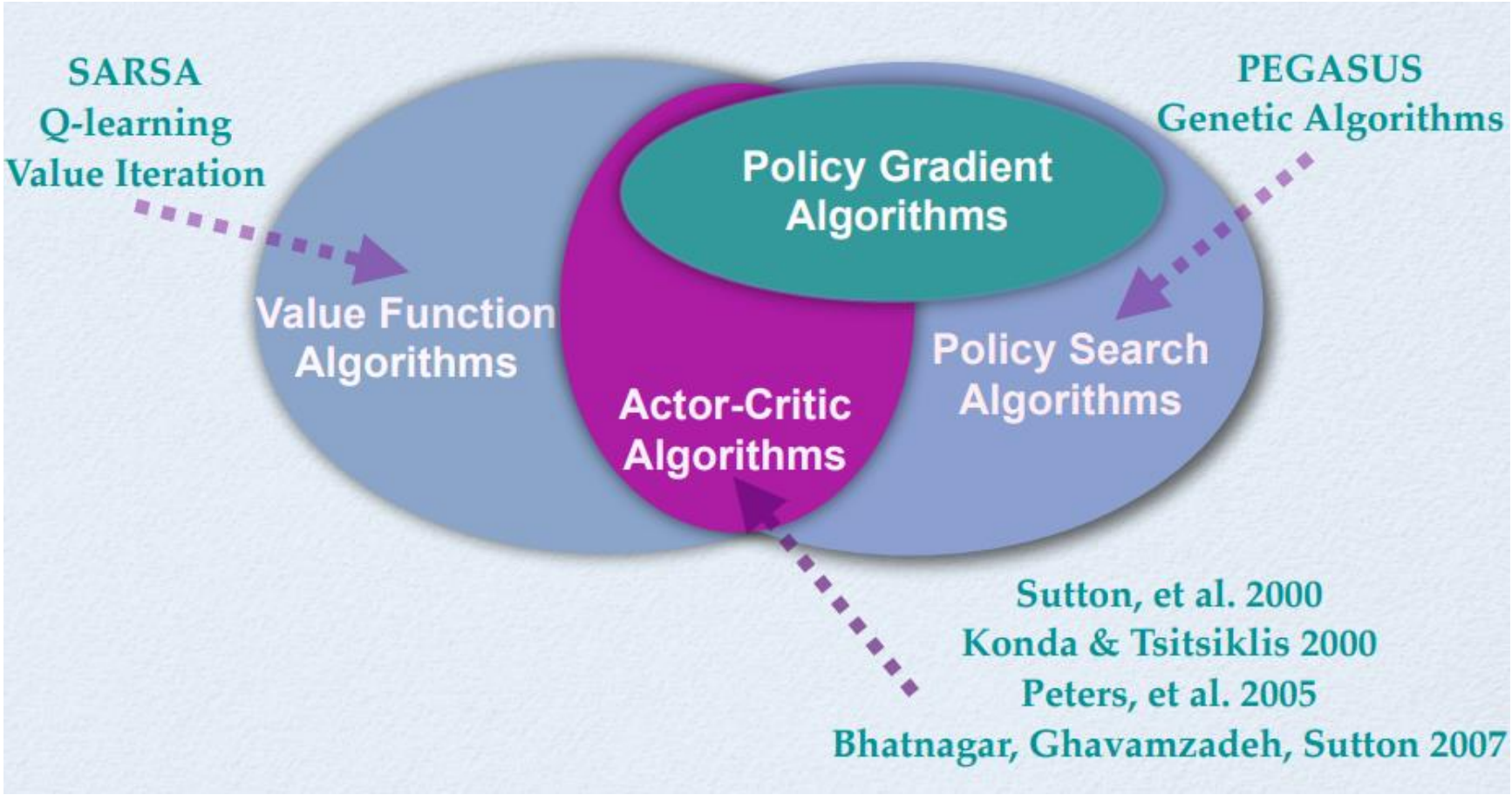
- Q-learning -> learn a value function
  - $\hat{Q}(s, a)$ = an estimate of the expected discounted reward of taking a from s
  - Performance time: take the action that has the highest estimated value

- Policy Gradient -> learn policy directly
  - $\pi(s)$ = Probability distribution over A_s
  - Performance time: choose action according to distribution



Example from: https://www.youtube.com/watch?v=tqrcjHuNdmQ

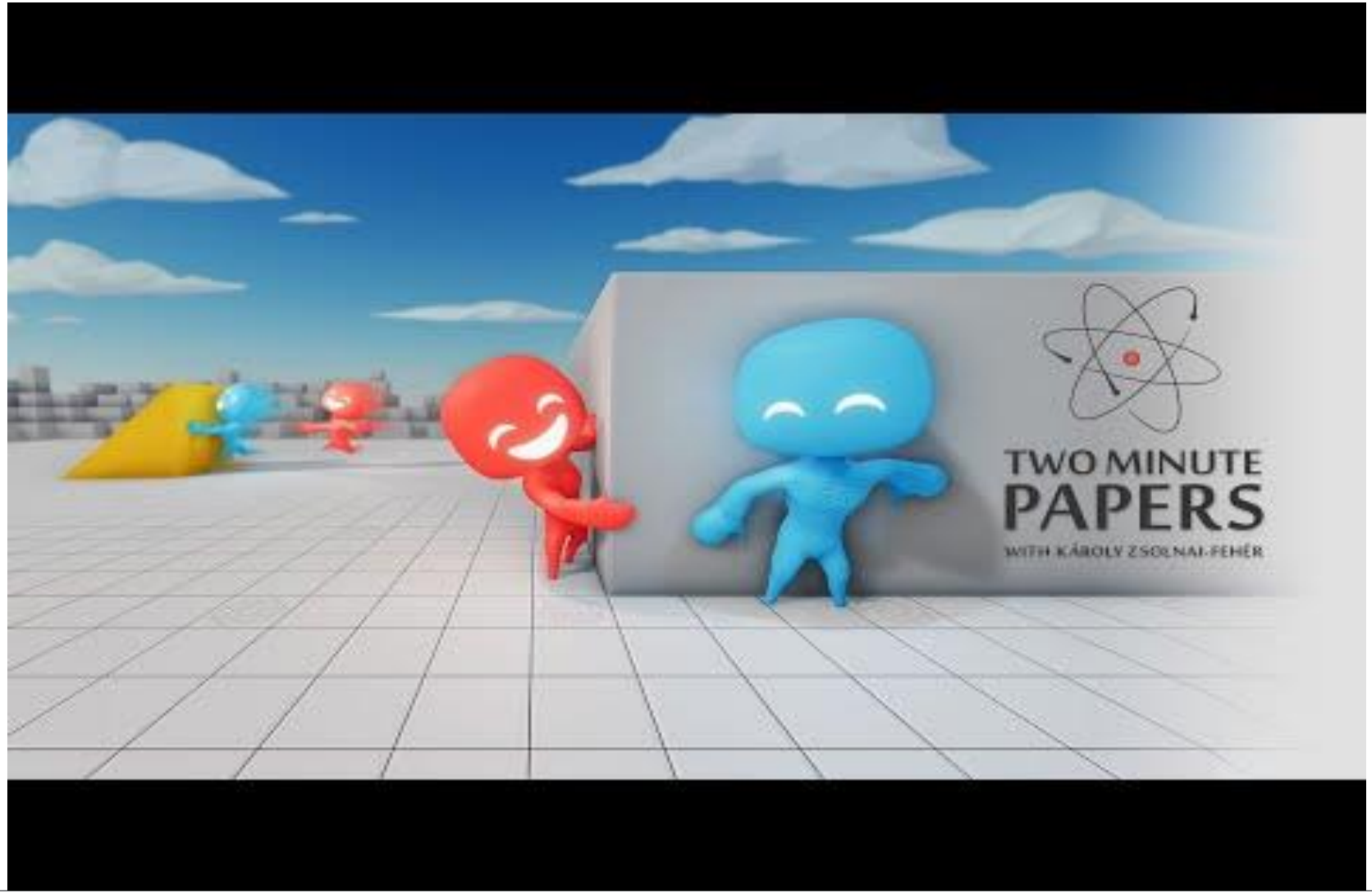
Different RL Algorithms



Open Questions for Current RL Algorithms

- Reward Design Problem
  - the challenge of designing a reward function that incentivizes an agent to learn behaviours that align with the desired objectives, while avoiding unintended consequences or undesirable behaviour patterns.
- Delayed Reward Problem
  - the challenge of determining the optimal actions to take in an environment where the reward associated with a particular action may not be immediately apparent, but rather delayed until a later time step.
- Credit Assignment Problem
  - the challenge of properly assigning credit or blame to the actions taken by an agent in a given state, given that the ultimate outcome may depend on multiple previous actions and environmental factors.

Multi-agent RL Cooperation Is Also An Active Research Topic Recently



https://www.youtube.com/watch?v=Lu56xVlZ40M

CRICOS PROVIDER 00123M

Thank you!

adelaide.edu.au

seek LIGHT