TDDC17 LE7 HT2023

Machine Learning III

Fredrik Heintz

Dept. of Computer Science
Linköping University

fredrik.heintz@liu.se

@FredrikHeintz

Outline:

- Reinforcement learning
- Deep reinforcement learning
- Multi-objective reinforcement learning

LINKÖPING UNIVERSITY

Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$x \rightarrow y$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

Apple example:

Eat this thing because it will keep you alive.

From Supervised to Reinforcement Learning - Learning How to Act

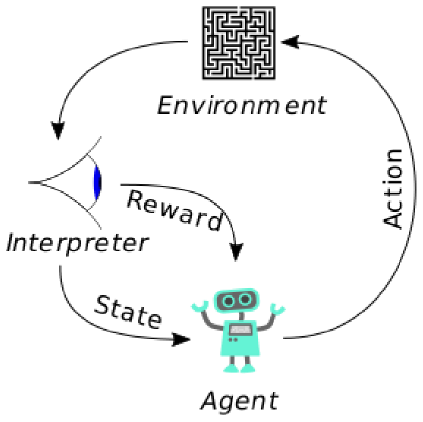
Humorous reminder from IEEE Spectrum: The DARPA 2015 Humanoid Challenge "Fail Compilation"
To be fair, this is the state of the art:
https://youtu.be/NR32ULxbjYc

- Can we use supervised learning to **learn** how to act?
- **E.g. engineering** robot behavior can be **fragile** and **time consuming**
  - Things humans do without thinking require **extremely detailed instructions** for a robot. Even robust locomotion is hard.

Learning How to Act

- Yes, one can learn a mapping from problem state (e.g. position) to action
  - As in all supervised learning, this requires a teacher
  - Sometimes called ”imitation learning”
- However, **supervised learning** with robots can get tedious as providing examples of correct behaviour is difficult to automate
- Can we remove the human from the loop?
  1. An **automated teacher** like a **planning or optimal control** algorithm can generate supervised examples **<u>if it has a model of the environment</u>**
     - Mordatch et al, https://www.youtube.com/watch?v=IxrnT0JOs4o
     - LiU’s research with real nano-quadcopters (deep ANN on-board the microcontroller)
  2. Reinforcement learning attempts to generalize this to learning from scratch in completely **unknown environments**

Reinforcement Learning Basic Concept



- *Reinforcement Learning is learning what to do – how to map situations to actions – so as to maximum a numerical reward.*

  Reinforcement Learning: An introduction
  Sutton & Barto
- Rather than learning from explicit training data, or discovering patterns in static data, reinforcement learning discovers the best option (highest reward) from trial and error.
- Inverse Reinforcement Learning
  - Learn reward function by observing an expert
  - “Apprenticeship learning“
  - E.g. Abbeel et al. *Autonomous Helicopter Aerobatics through Apprenticeship Learning*

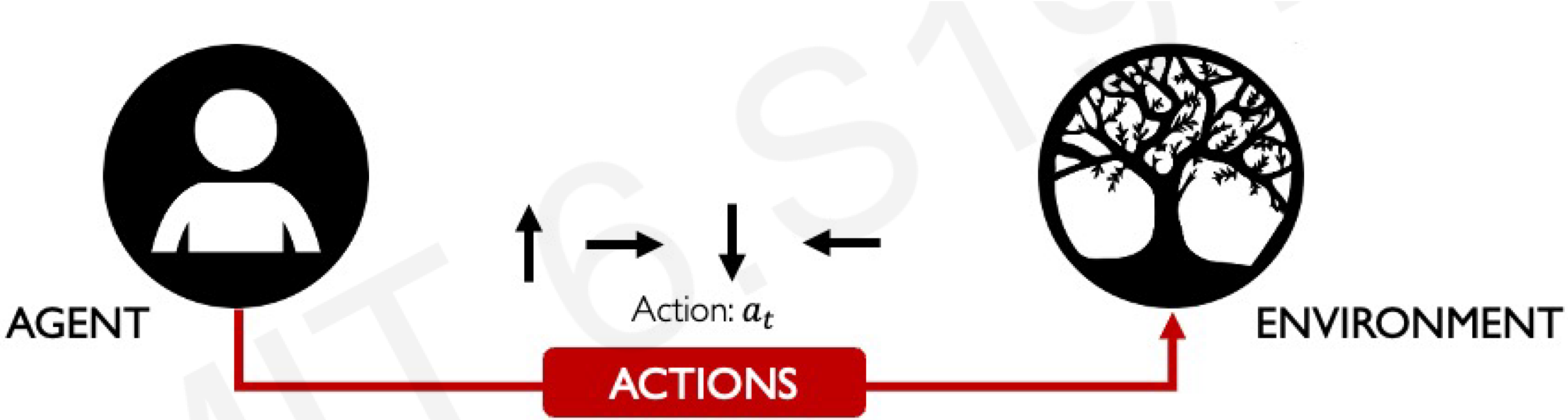
Reinforcement Learning: Key Concepts

Agent: takes actions.

Reinforcement Learning: Key Concepts

Environment: the world in which the agent exists and operates.

Reinforcement Learning: Key Concepts



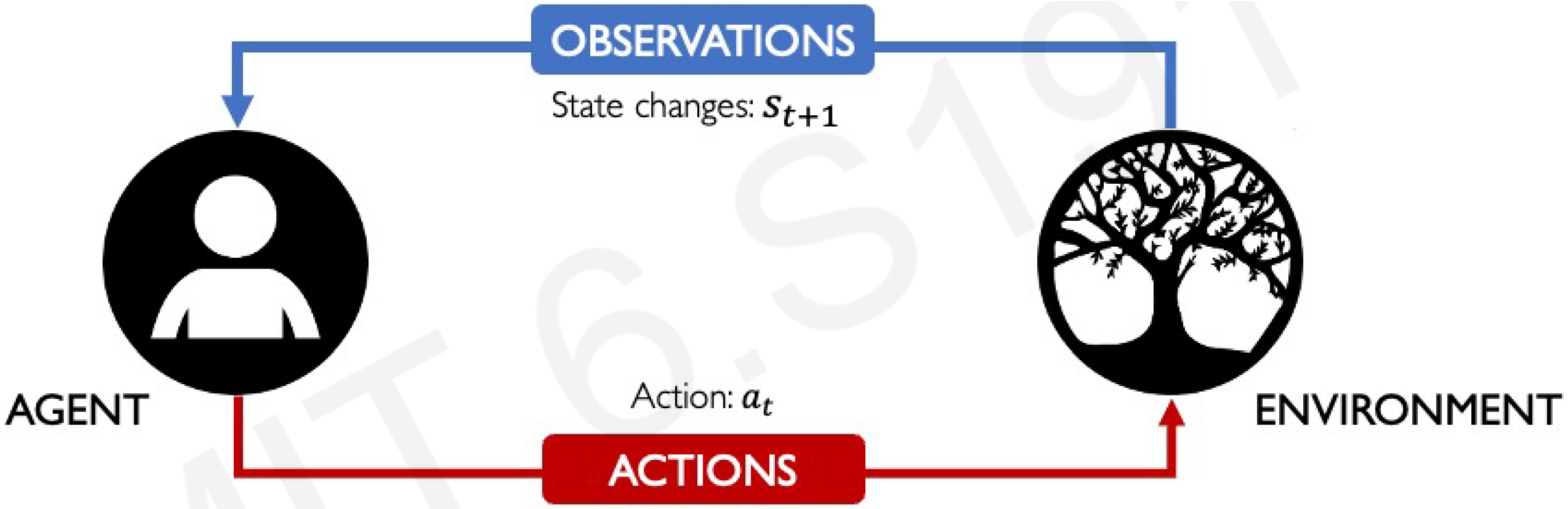
Action: a move the agent can make in the environment.

Action space A: the set of possible actions an agent can make in the environment

Reinforcement Learning: Key Concepts

Observations: of the environment after taking actions.

Reinforcement Learning: Key Concepts



State: a situation which the agent perceives.

Reinforcement Learning: Key Concepts

Reward: feedback that measures the success or failure of the agent's action.

Reinforcement Learning: Key Concepts

Total Reward (Return)

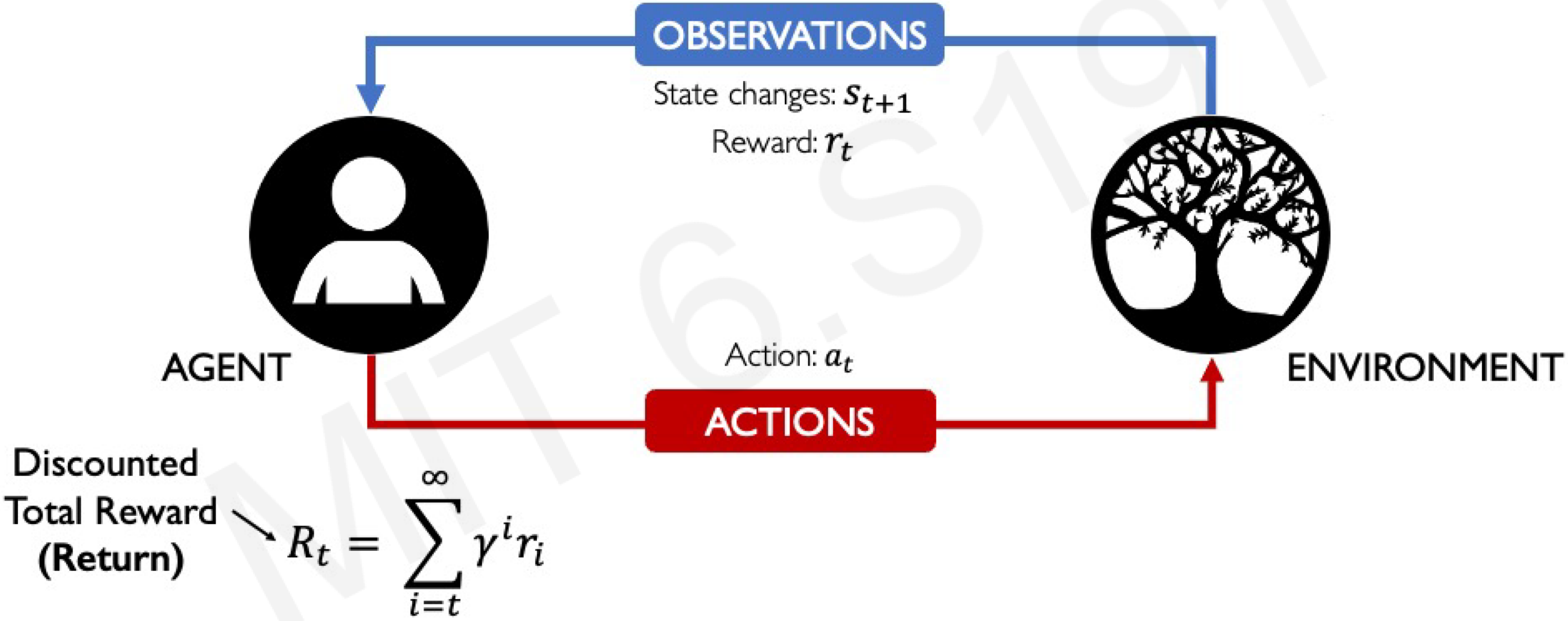
$$R_t = \sum_{i=t}^{\infty} r_i$$

Reinforcement Learning: Key Concepts

Total Reward (Return)

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} \ldots + r_{t+n} + \cdots$$

Reinforcement Learning: Key Concepts



Reinforcement Learning: Key Concepts

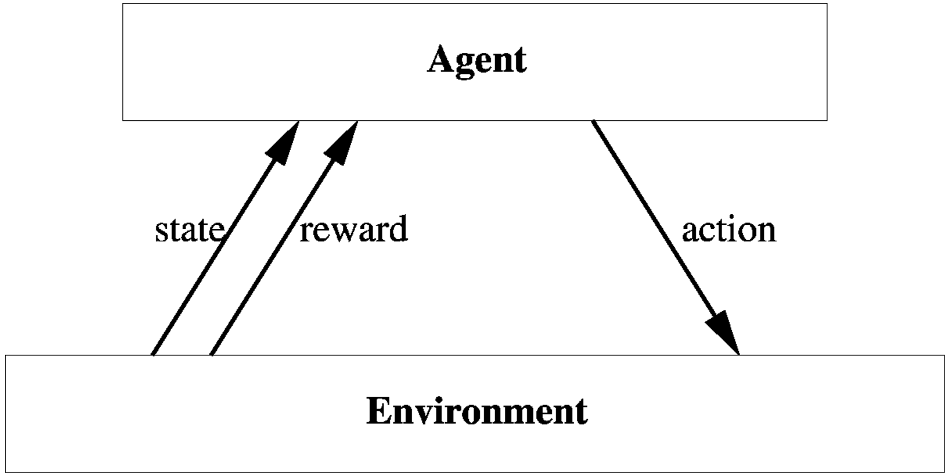
Discounted Total Reward (Return)

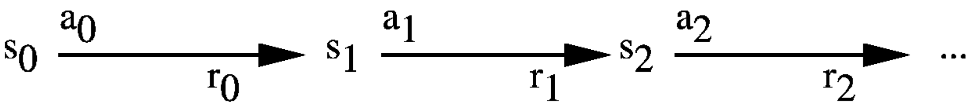
$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} \dots + \gamma^{t+n} r_{t+n} + \cdots$$

γ: discount factor; $0 < \gamma < 1$

A Reinforcement Learning Problem

- The environment
- The reinforcement function *r(s,a)*
  - Pure delay reward and avoidance problems
  - Minimum time to goal
  - Games
- The value function *V(s)*
  - Policy $\pi: S \rightarrow A$
  - Value $V^{\pi}(s) := \Sigma_i \gamma^i r_{t+i}$
- Find the optimal policy π^* that maximizes $V^{\pi^*}(s)$ for all states *s*.





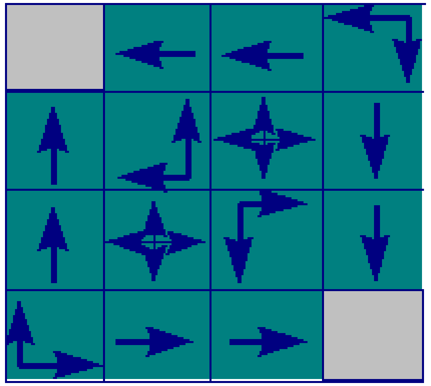
Goal: Learn to choose actions that maximize
$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0<\gamma<1$

RL Value Function - Example

A minimum time to goal world

0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

Value function for random movement



Optimal policy

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

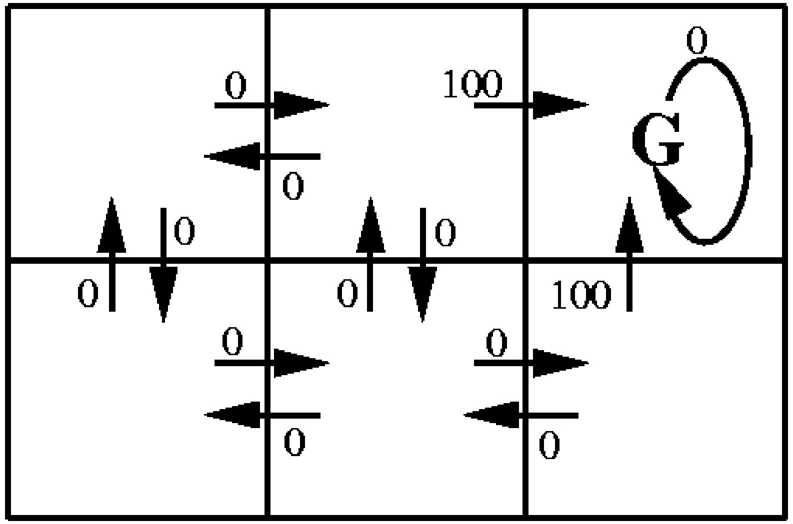
Optimal value function

Markov Decision Processes

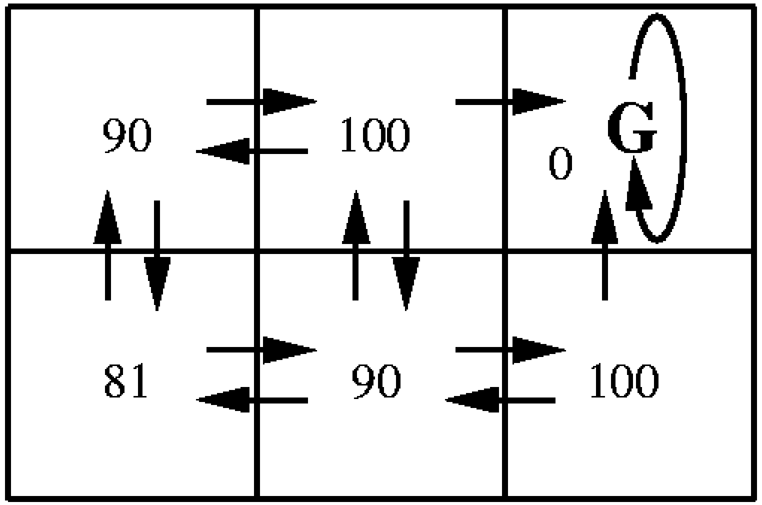
Assume:

- finite set of states *S*, finite set of actions *A*
- at each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$
- then receives immediate reward r_t
- and state changes to s_{t+1}
- Markov assumption: $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
  - i.e. r_t and s_{t+1} depend only on current state and action
  - functions δ and *r* may be non-deterministic
  - functions δ and *r* not necessarily known to the agent

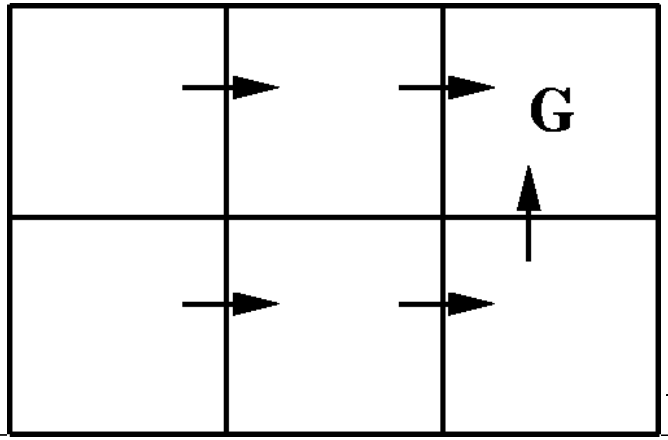
MDP Example



r(s,a)



V(s)*



An optimal policy

Defining the Q-Function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total reward, R_t, is the discounted sum of all rewards obtained from time t

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

The Q-function captures the **expected total future reward** an agent in state, s, can receive by executing a certain action, a

How to Take Actions Given a Q-Function

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

(state, action)

Ultimately, the agent needs a **policy** $\boldsymbol{\pi(s)}$, to infer the **best action to take** at its state, s

Strategy: the policy should choose an action that maximizes future reward

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

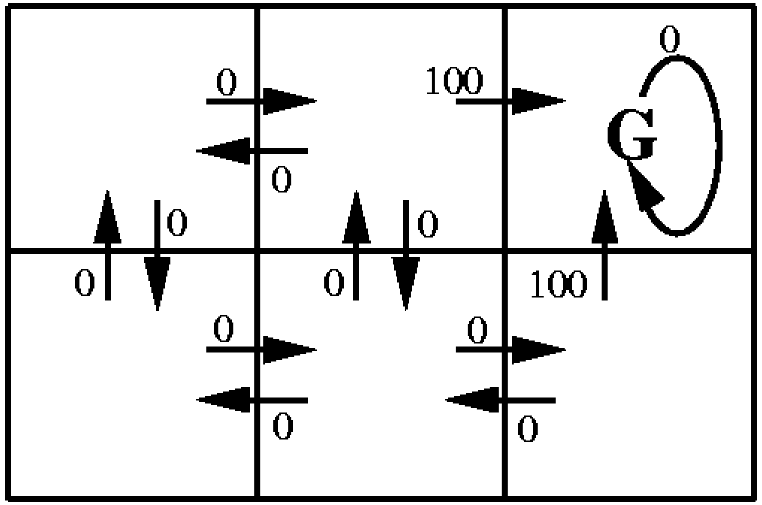
The Q-Function

Optimal policy:

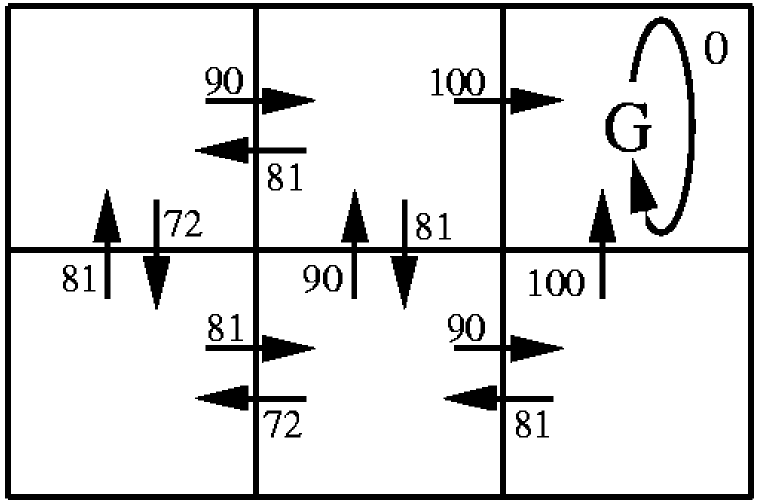
- $\pi^*(s) = \text{argmax}_a[r(s,a) + \gamma V^*(\delta(s,a))]$
- Doesn't work if we don't know *r* and δ.

The Q-function:

- $Q(s,a) := r(s,a) + \gamma V^*(\delta(s,a))$
- $\pi^*(s) = \text{argmax}_a Q(s,a)$



r(s,a)



Q(s,a)

The Q-Function

- Note Q and V* closely related:
  $V^*(s) = \max_{a'} Q(s,a')$
- Therefore Q can be written as:

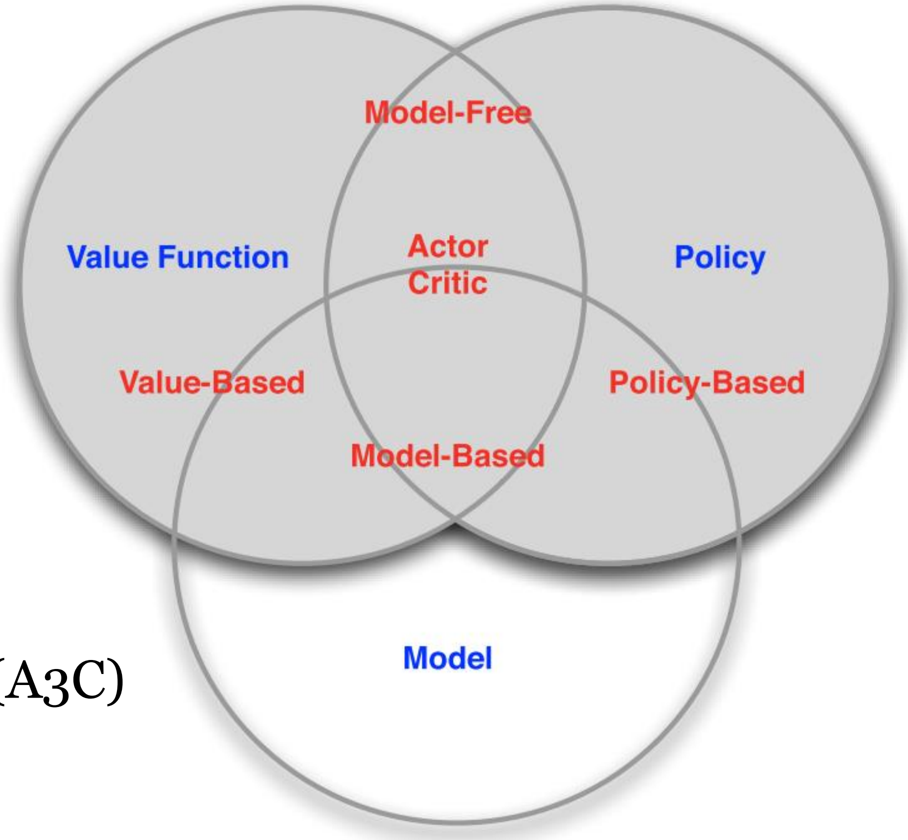
$$Q(s_t, a_t) := r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) =$$
$$r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$

- If $Q^\wedge$ denote the current approximation of Q then it can be updated by:

$$Q^\wedge(s,a) := r + \gamma \max_{a'} Q^\wedge(s',a')$$

Reinforcement Learning Concepts

- Value-Based:
  - Learn value function
  - Implicit policy (e.g. greedy selection)
  - Example: Deep Q Networks (DQN)
- Policy-Based:
  - No value function
  - Learn explicit (stochastic) policy
  - Example: Stochastic Policy Gradients
- Actor-Critic:
  - Learn value function
  - Learn policy using value function
  - Example: Asynchronous Advantage Actor Critic (A3C)



Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Reinforcement Learning Algorithms

Value Learning

Find $Q(s,a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Q-Learning for Deterministic Worlds

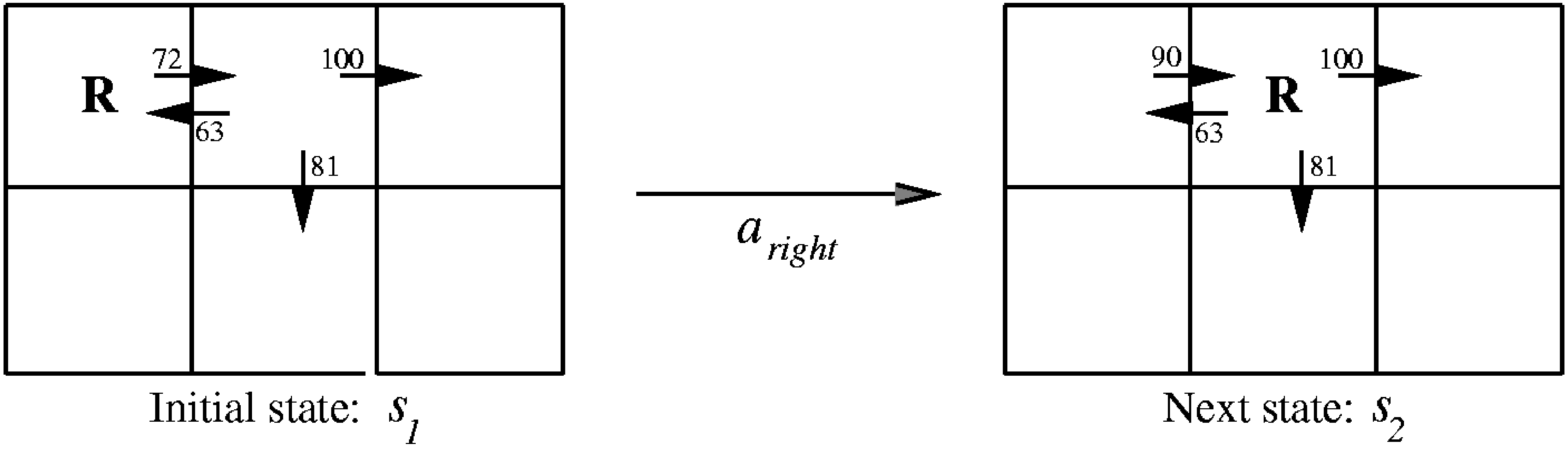
For each s, a initialize table entry $\hat{Q}(s,a) := 0$.

Observe current state s.

Do forever:

1. Select an action a and execute it
2. Receive immediate reward r
3. Observe the new state s'
4. Update the table entry for $\hat{Q}(s,a)$:
   $\hat{Q}(s,a) := r + \gamma \max_{a'} \hat{Q}(s',a')$
5. $s := s'$

Q-Learning Example



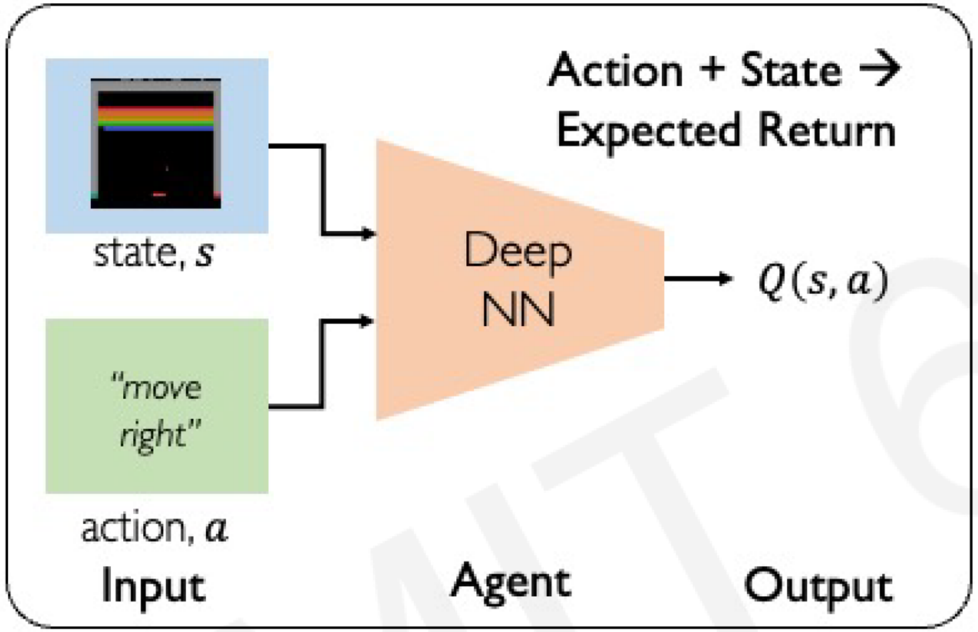
$$\hat{Q}(s_1, a_{right}) := r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
$$:= 0 + 0.9 \max\{63, 81, 100\}$$
$$:= 90$$

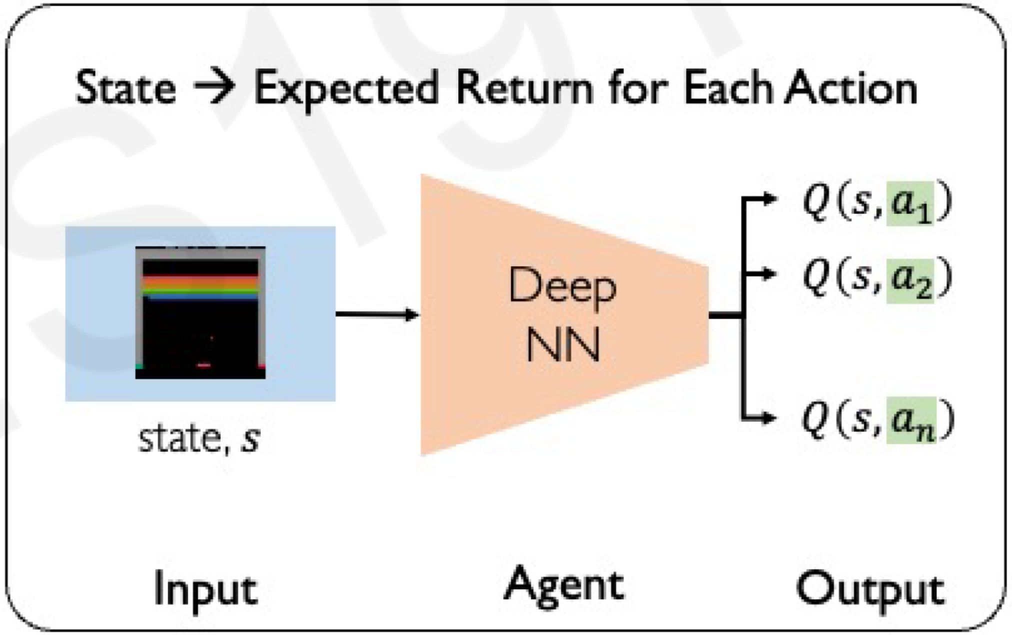
Q-Learning Continued

- Exploration
  - Selecting the best action
  - Probabilistic choice
- Improving convergence
  - Update sequences
  - Remember old state-action transitions and their immediate reward
- Non-deterministic MDPs
- Temporal Difference Learning

Deep Q-Learning (DQN)

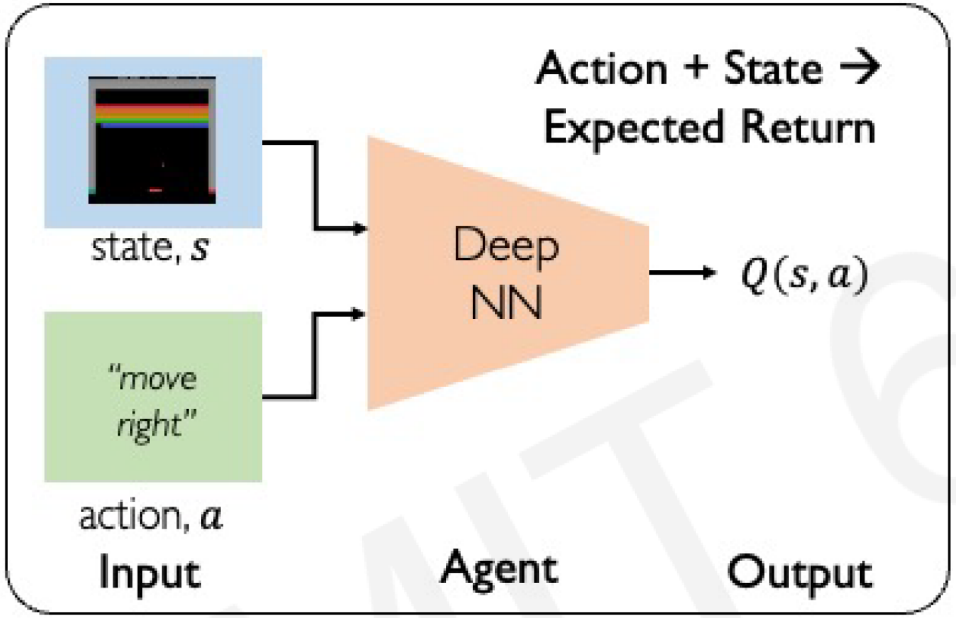
How can we use deep neural networks to model Q-functions?

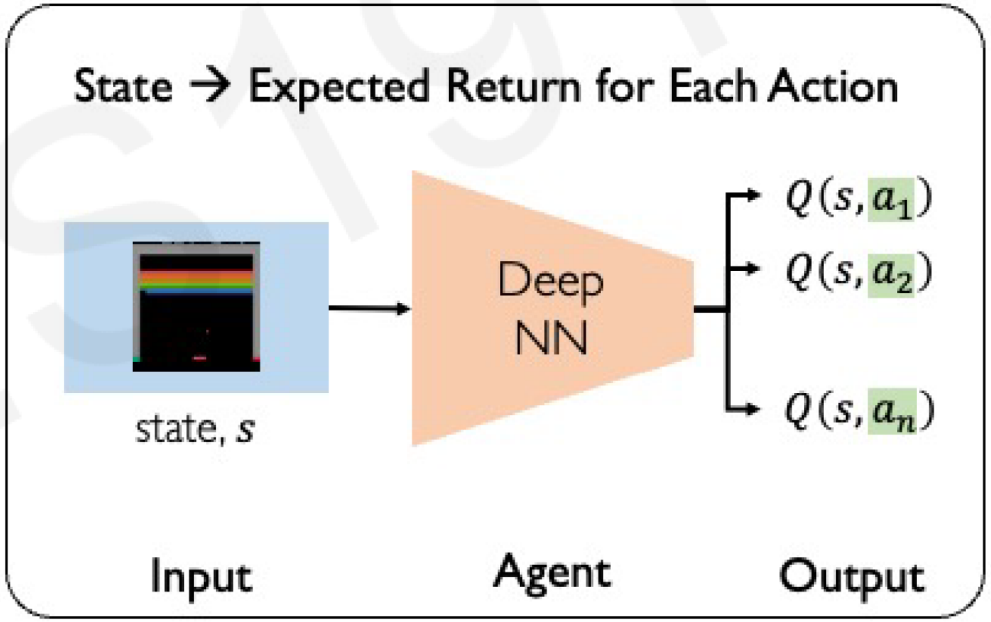




Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?

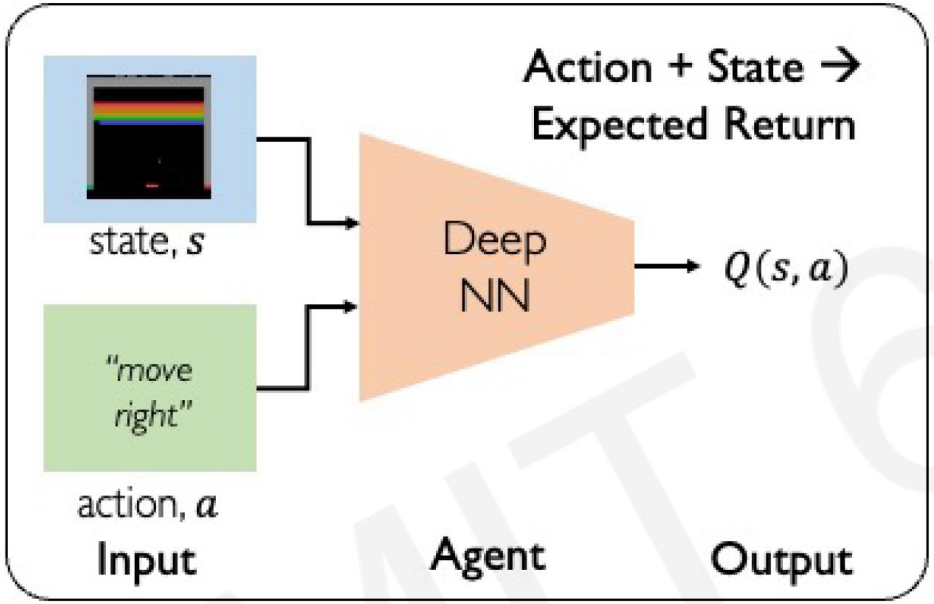


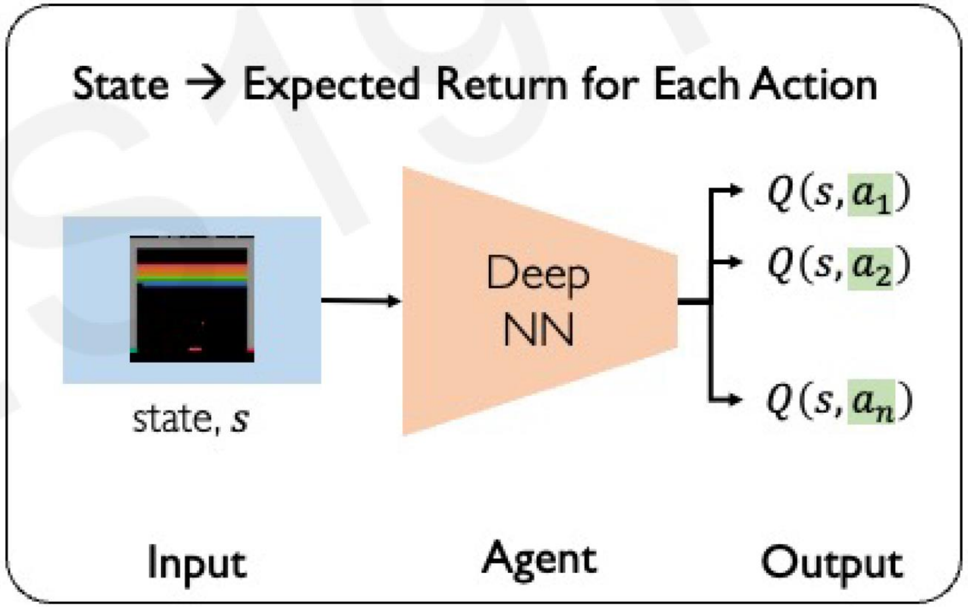


What happens if we take all the best actions?
Maximize target return → train the agent

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?



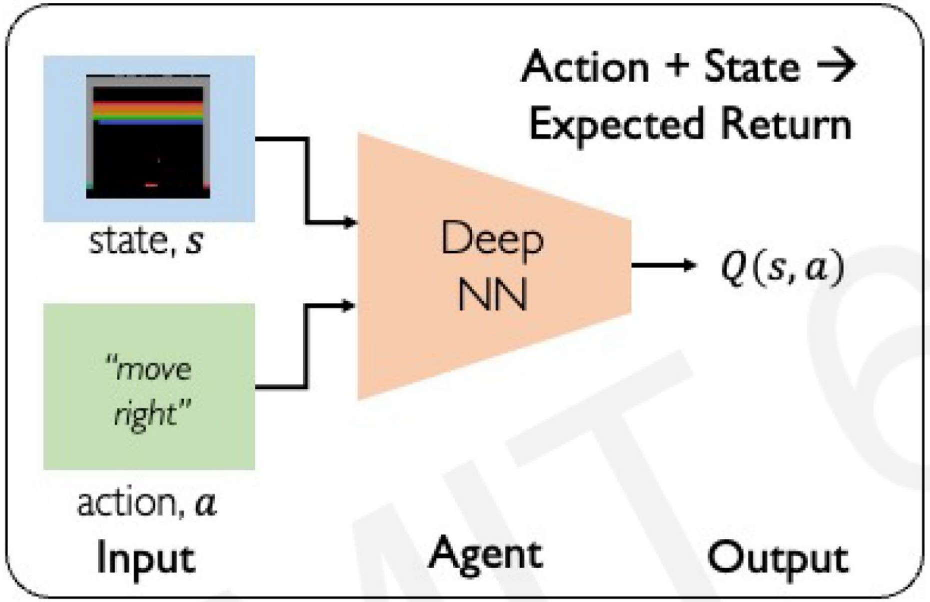


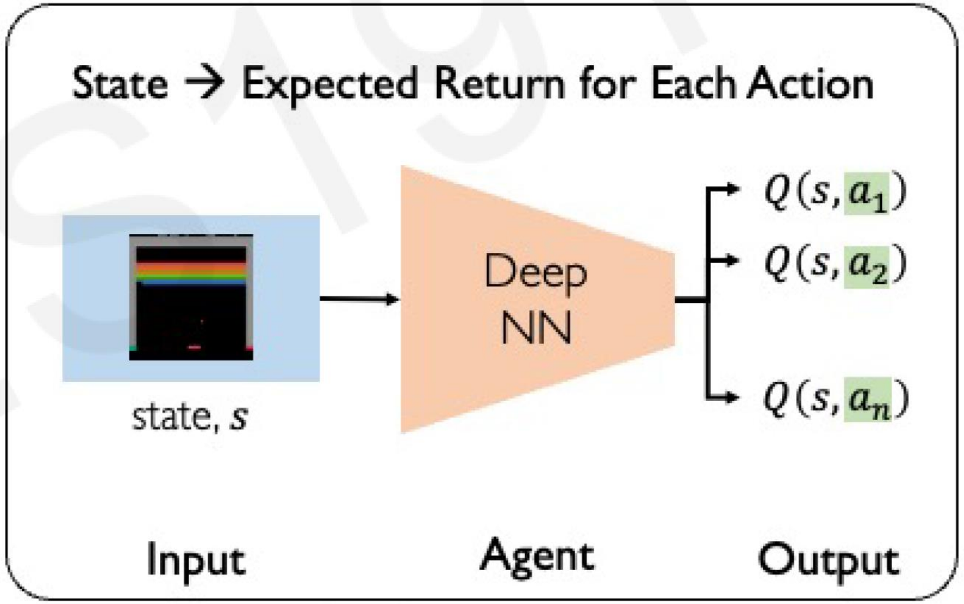
$$\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}}$$

Take all the best actions → target return

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?



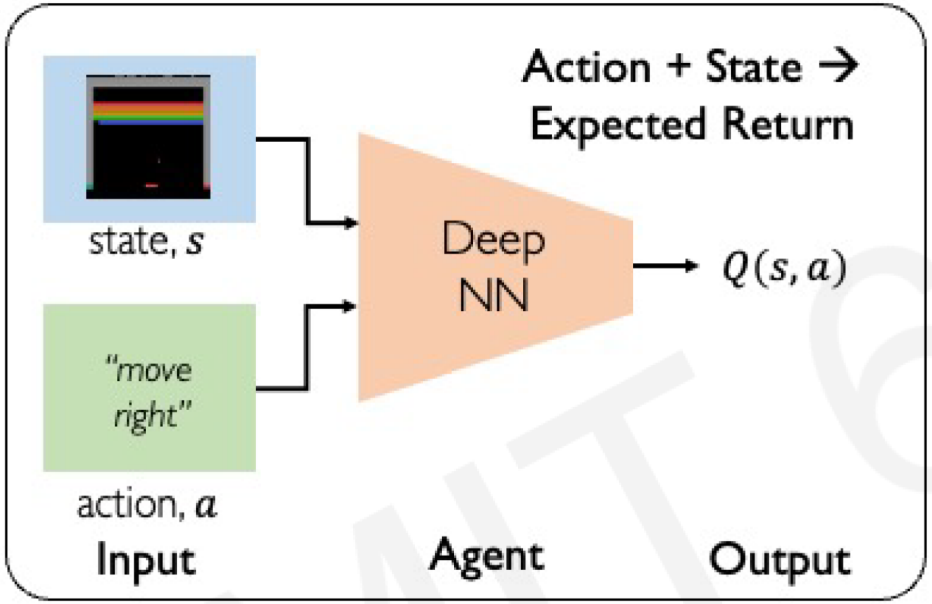


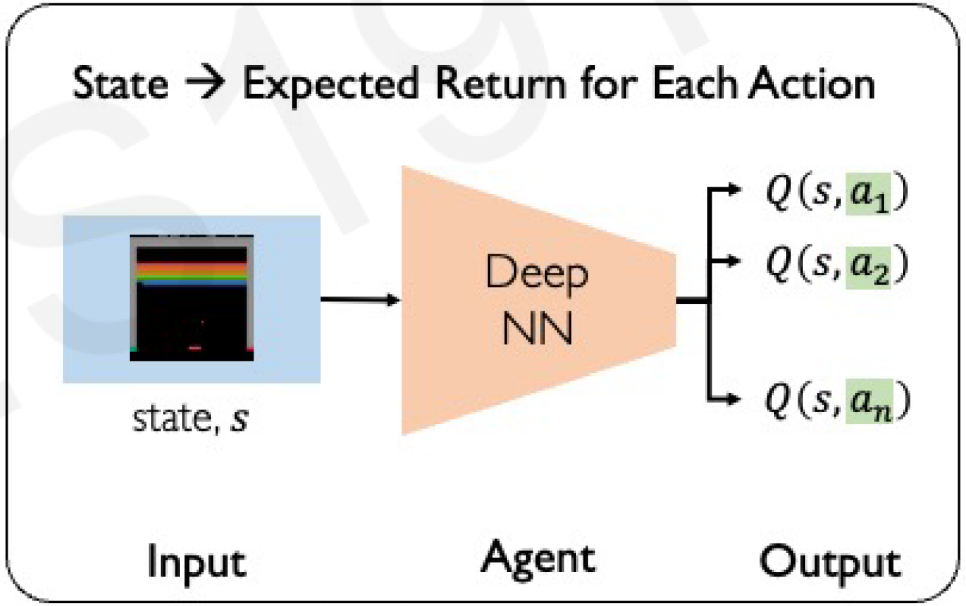
$$\underbrace{\left(r + \gamma \max_{a'} Q(s',a')\right)}_{\text{target}} \qquad \underbrace{Q(s,a)}_{\text{predicted}}$$

Network prediction

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?



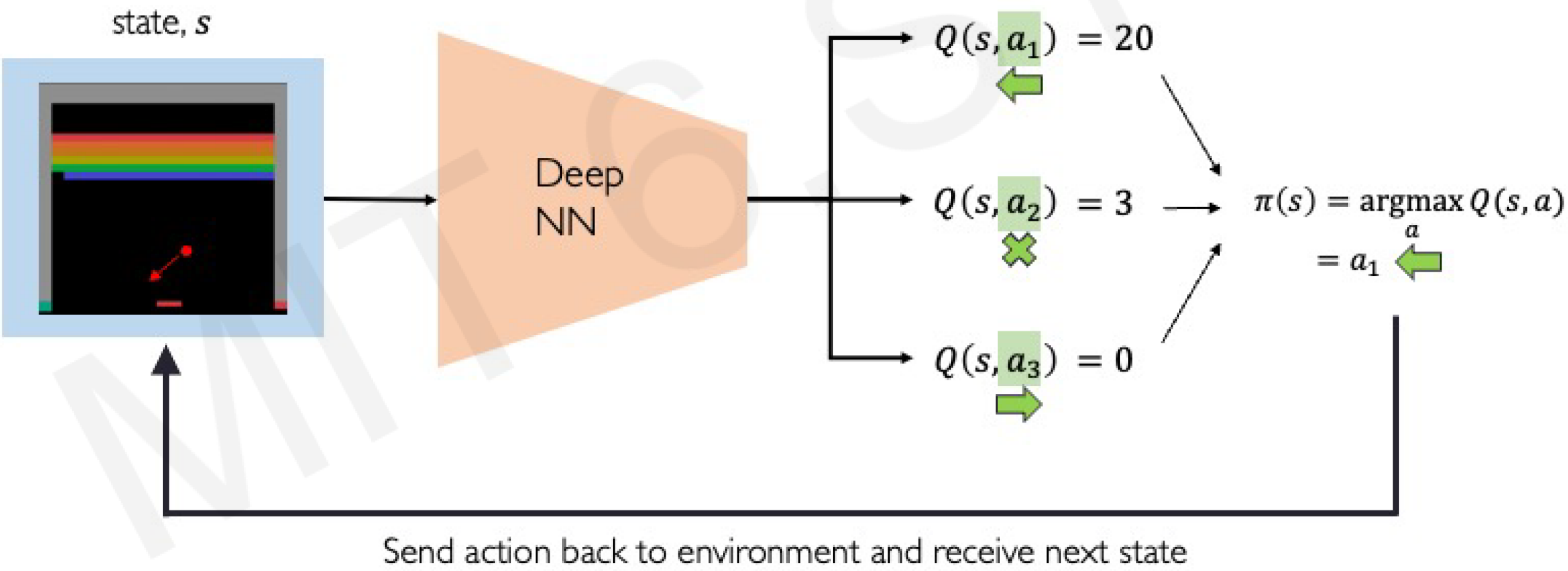


$$\mathcal{L} = \mathbb{E}\left[\left\| \overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} - \overbrace{Q(s, a)}^{\text{predicted}} \right\|^2\right]$$

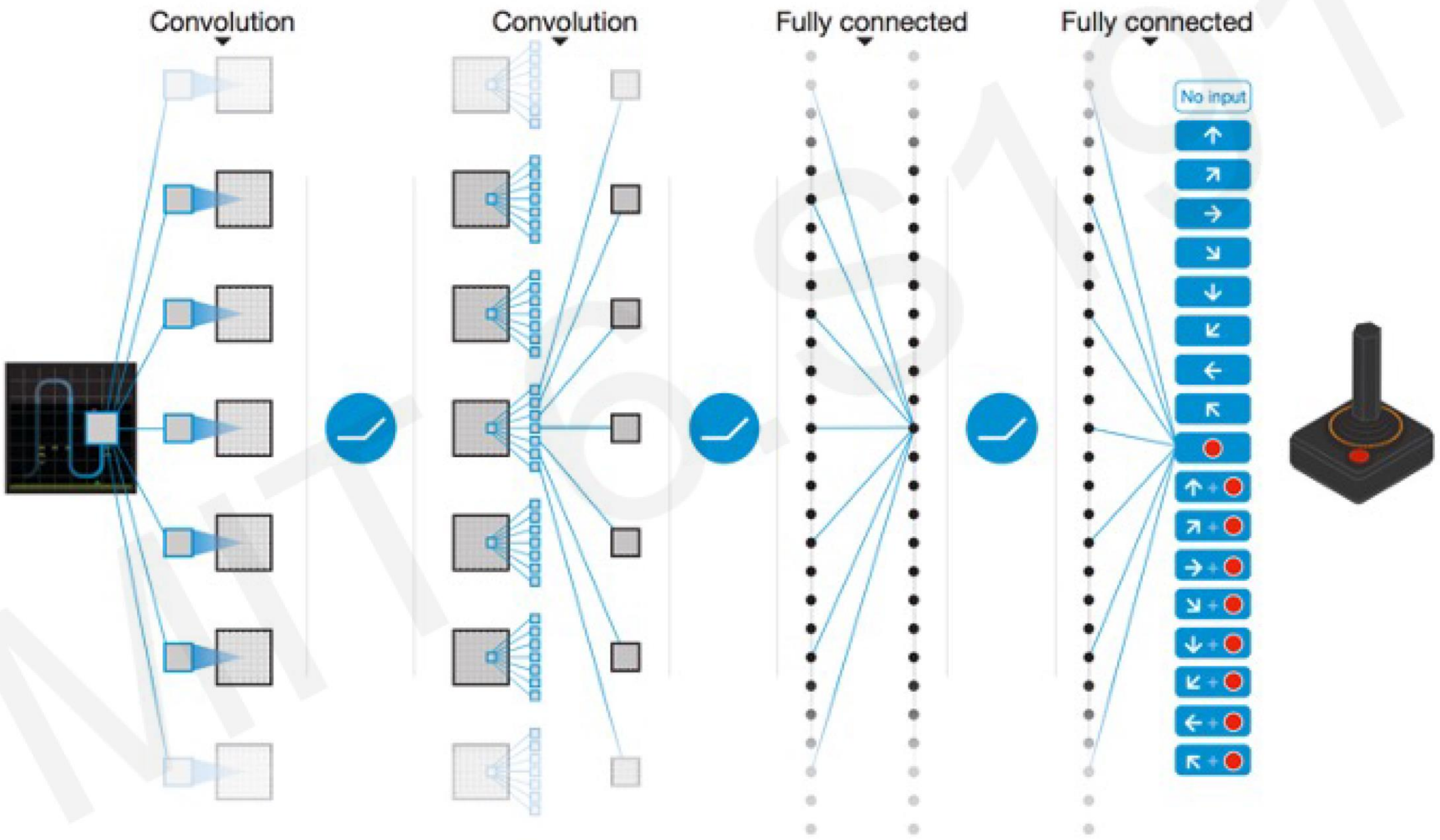
Q-Loss

Deep Q Network Summary

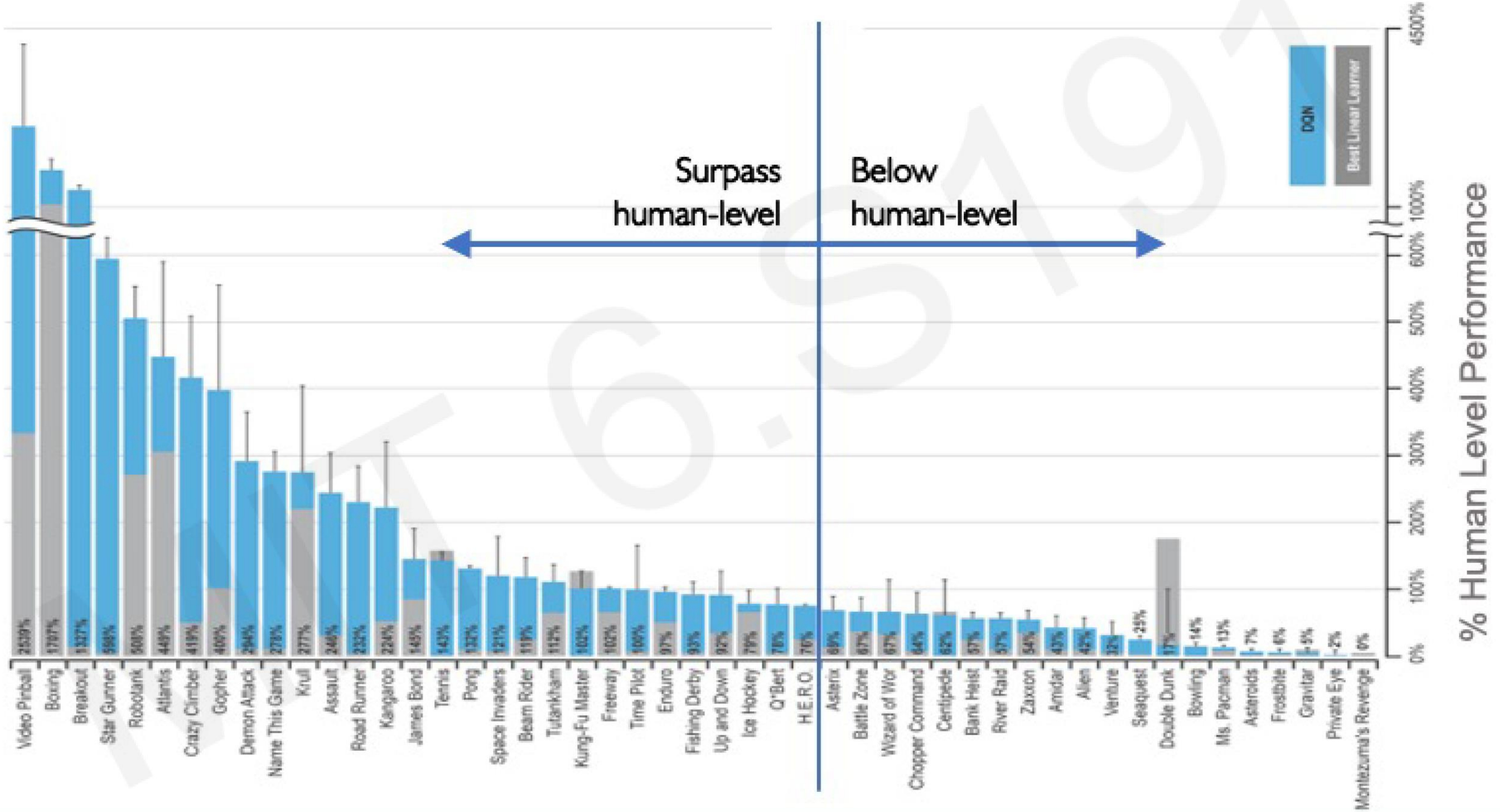
Use NN to learn Q-function and then use to infer the optimal policy, $\pi(s)$



DQN Atari Results



DQN Atari Results



Downsides of Q-Learning

Complexity:
- Can model scenarios where the action space is discrete and small
- Cannot handle continuous action spaces

Flexibility:
- Policy is deterministically computed from the Q function by maximizing the reward → cannot learn stochastic policies

To address these, consider a new class of RL training algorithms: Policy gradient methods

Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

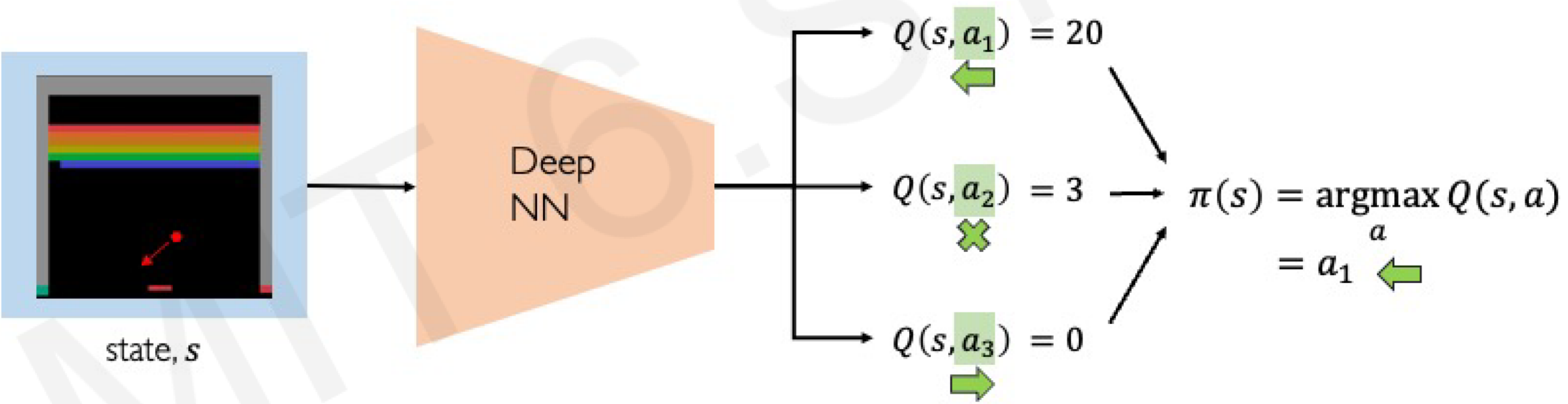
Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Deep Q Networks

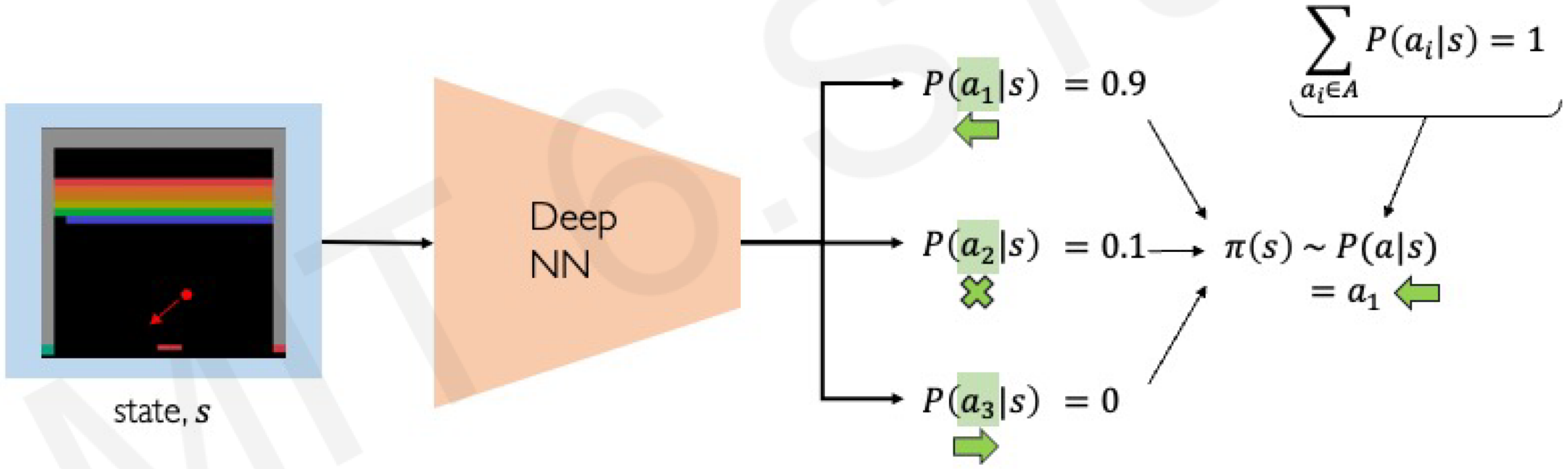
DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$



Policy Gradient (PG): Key Idea

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

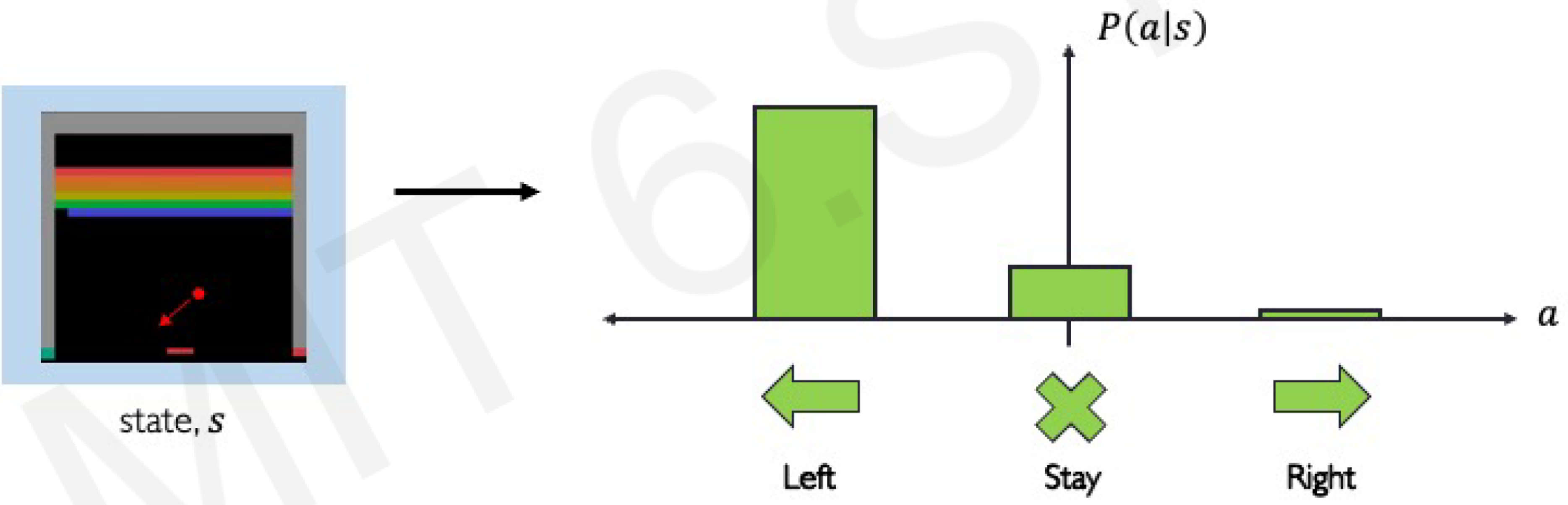
Policy Gradient: Directly optimize the policy $\pi(s)$



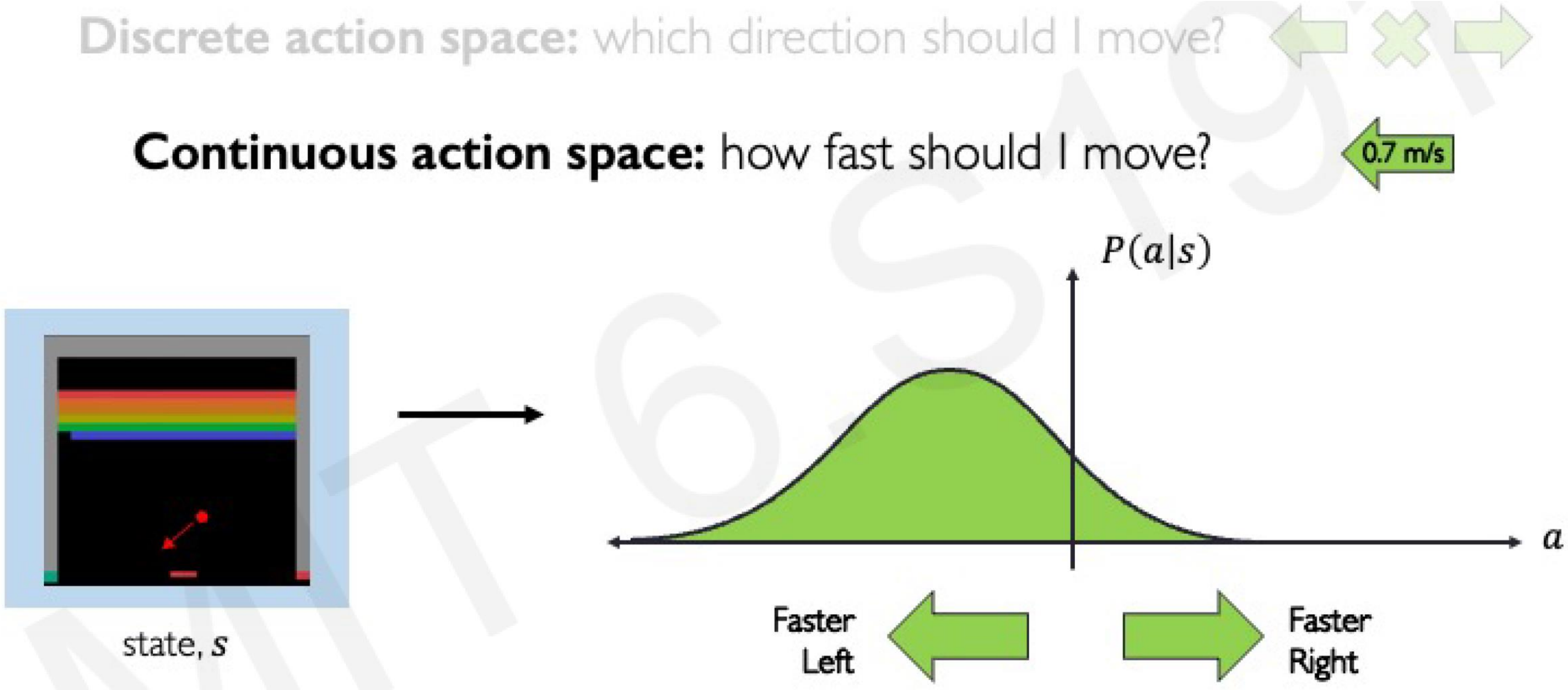
What are some advantages of this formulation?

Discrete vs Continuous Action Spaces

Discrete action space: which direction should I move?

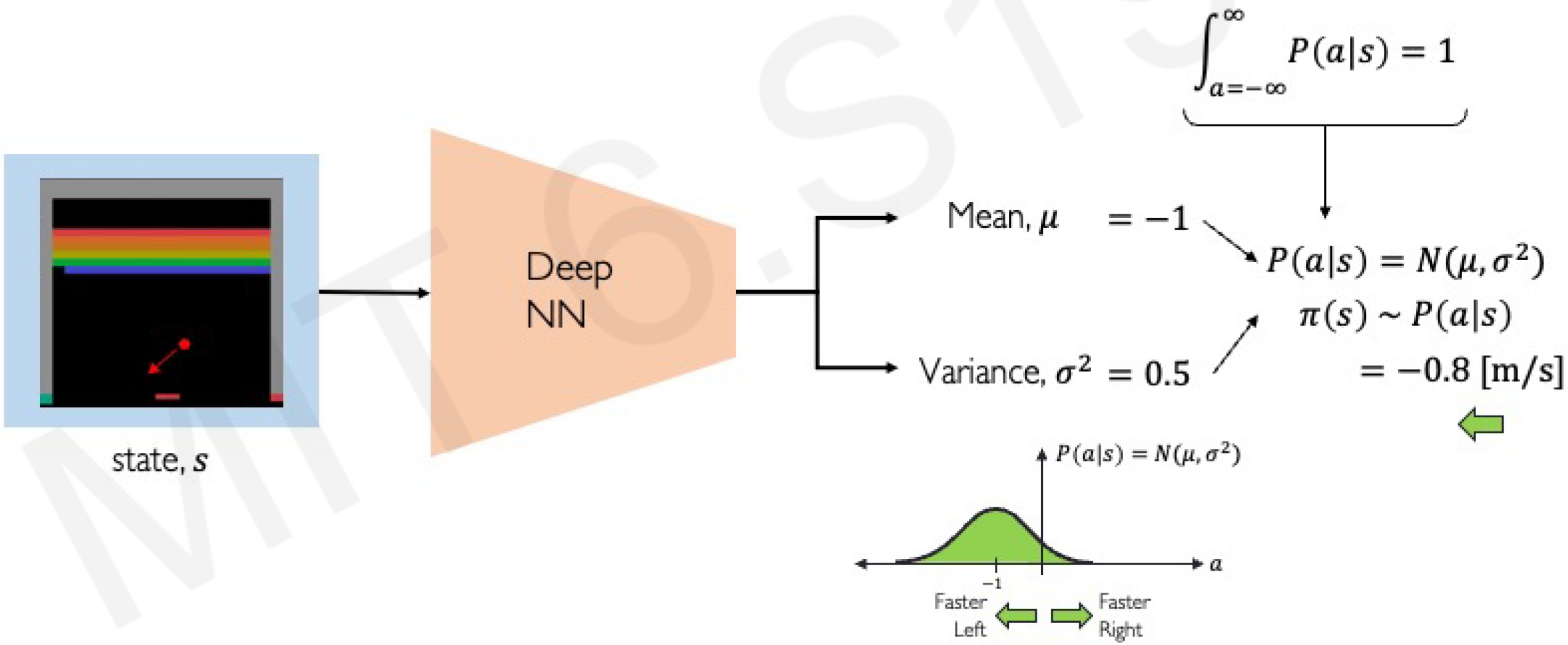


Discrete vs Continuous Action Spaces



Policy Gradient (PG): Key Idea

Policy Gradient: Enables modeling of continuous action space



Training Policy Gradients: Case Study

Reinforcement Learning Loop:

Case Study – Self-Driving Cars

Agent: vehicle

State: camera, lidar, etc

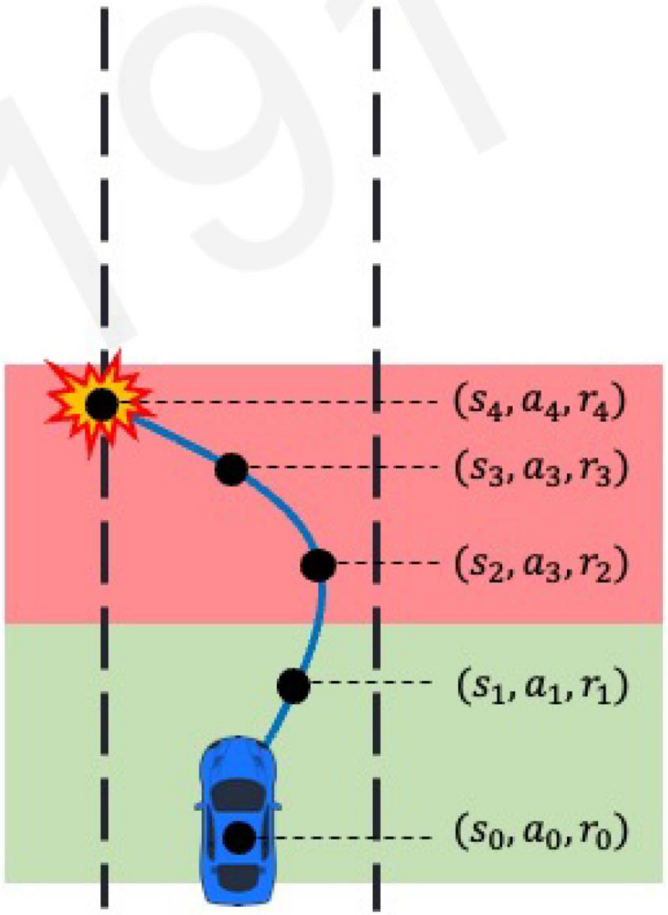
Action: steering wheel angle

Reward: distance traveled

Training Policy Gradients

Training Algorithm

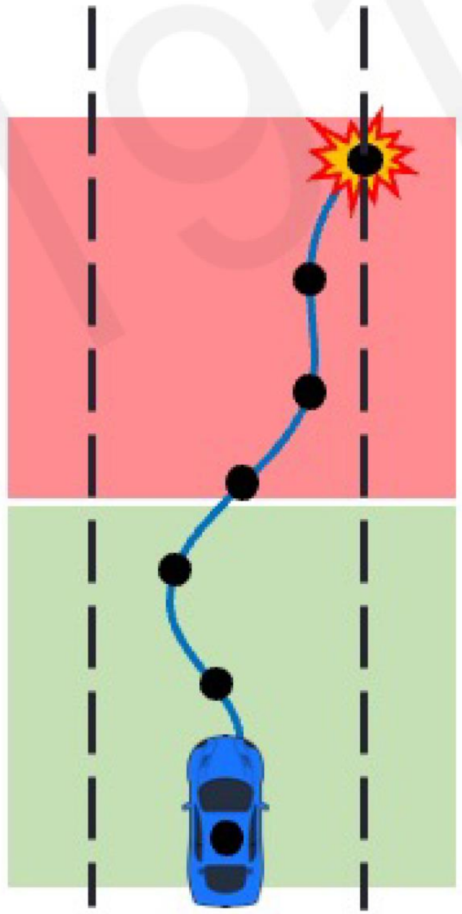
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

Training Algorithm

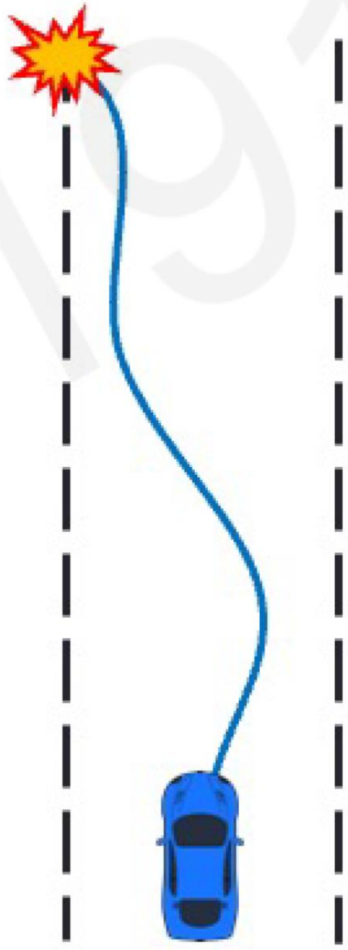
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

Training Algorithm

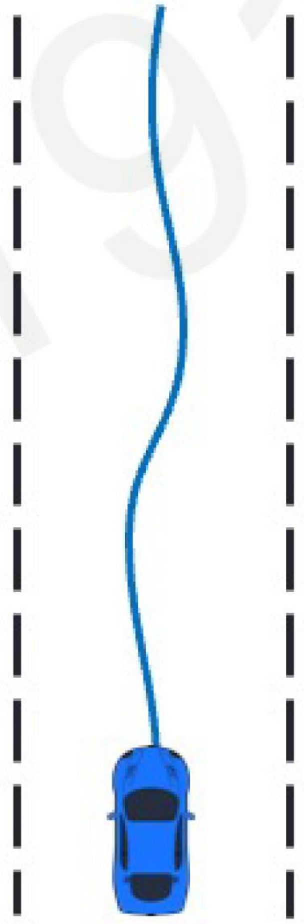
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

log-likelihood of action

$$\textbf{loss} = -\log \mathrm{P}(a_t|s_t)\, R_t$$

reward

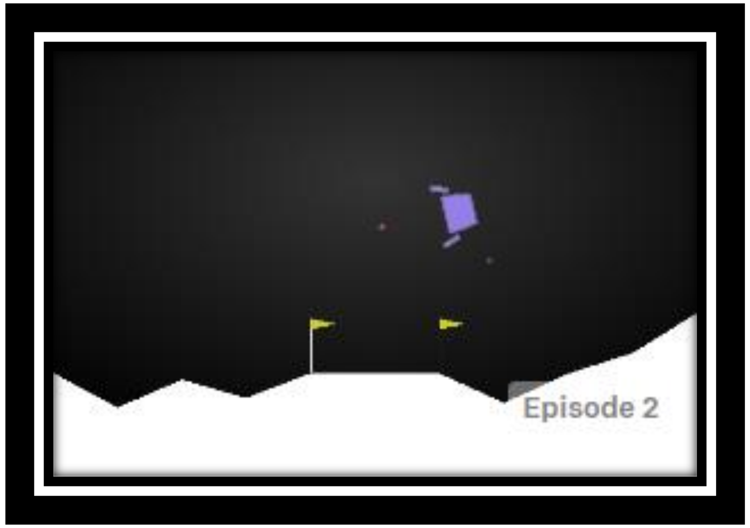
Gradient descent update:

$$w' = w - \nabla\, \textbf{loss}$$

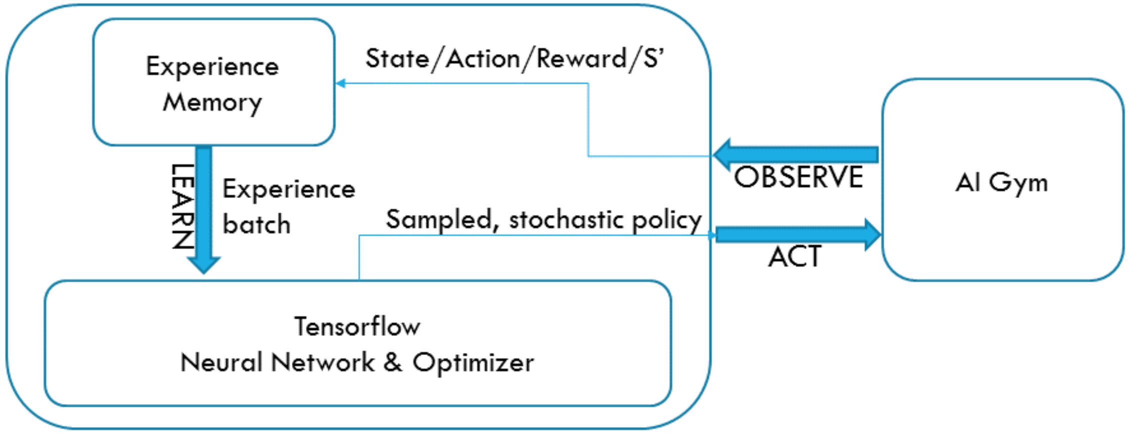
$$w' = w + \nabla \log \mathrm{P}(a_t|s_t)\, R_t$$

Policy gradient!

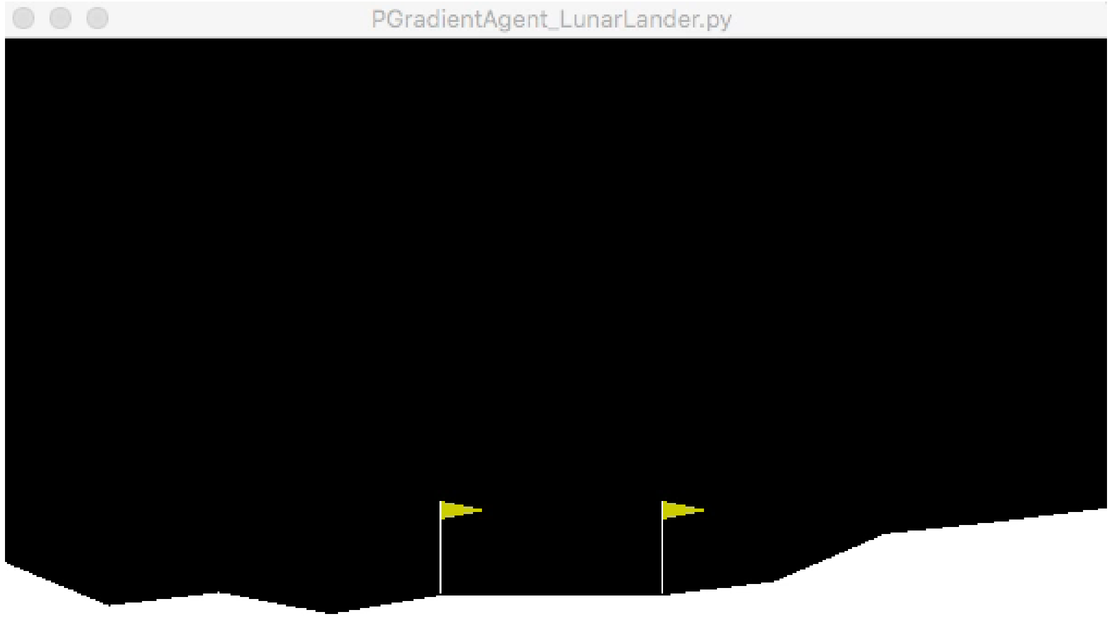
Reinforcement Learning – Neural Networks as Function Approximators



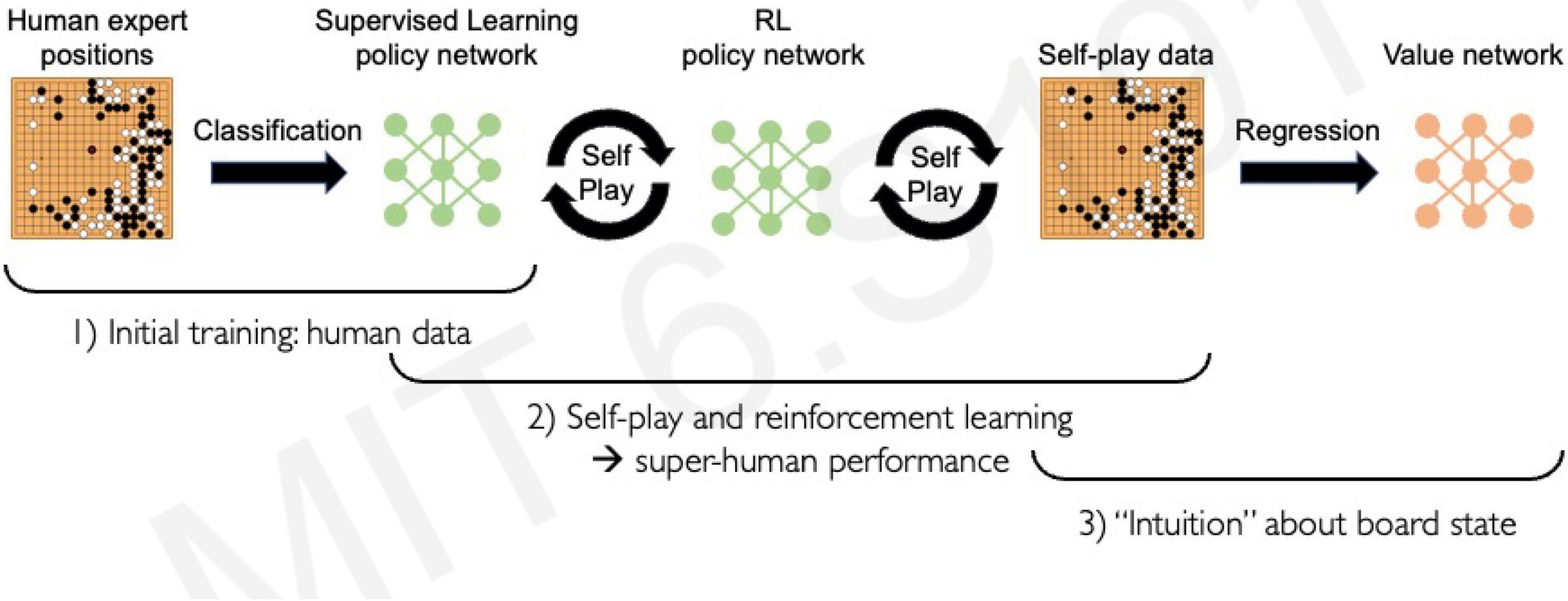
- To tackle a high-dimensional state space or continous states we can use a neural network as function approximator
- Lunar Lander experiment
    - 8 continous/discrete states
        - XY-Pos, XY-Vel, Rot, Rot-rate, Leg1/Leg2 ground contact
    - 4 discrete actions
        - Left thrust
        - Right thrust
        - Main engine thrust
        - NOP
    - Rewards
        - Move from top to bottom of the screen (+ ~100-140)
        - Land between the posts (+100)
        - Put legs on ground (+10 per leg)
    - Penalties
        - Using main engine thrust (-0.3 per frame)
        - Crashing (-100)
- Solved using Stochastic Policy Gradients



Reinforcement Learning Neural Networks as Function Approximators



AlphaGo Beats Top Human Player (2016)



MuZero: Learning Dynamics for Planning (2020)

Deep Reinforcement Learning Summary

Foundations

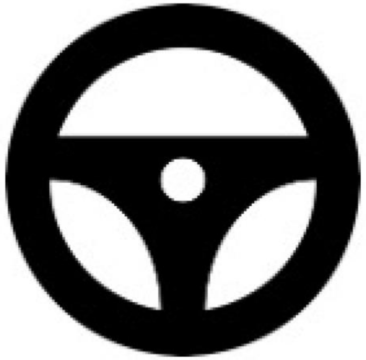
- Agents acting in environment
- State-action pairs → maximize future rewards
- Discounting

Q-Learning

- Q function: expected total reward given **s, a**
- Policy determined by selecting action that maximizes Q function

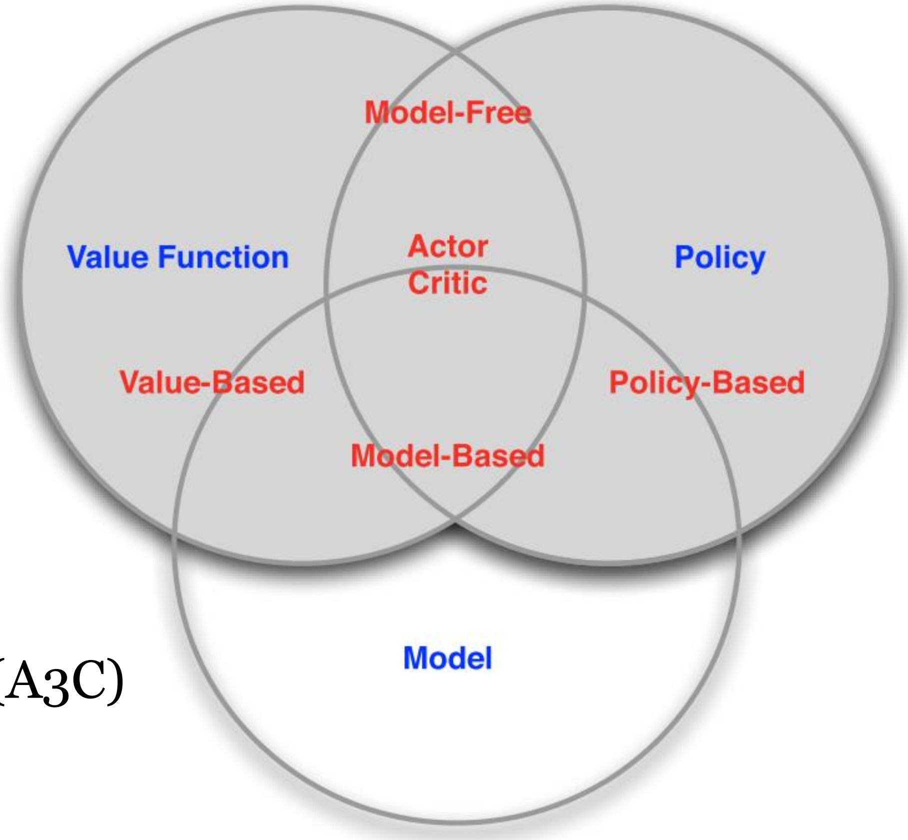
Policy Gradients

- Learn and optimize the policy directly
- Applicable to continuous action spaces



Reinforcement Learning Concepts

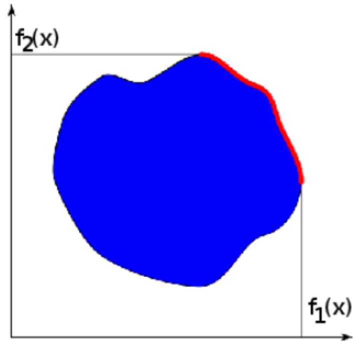
- Value-Based:
  - Learn value function
  - Implicit policy (e.g. greedy selection)
  - Example: Deep Q Networks (DQN)
- Policy-Based:
  - No value function
  - Learn explicit (stochastic) policy
  - Example: Stochastic Policy Gradients
- Actor-Critic:
  - Learn value function
  - Learn policy using value function
  - Example: Asynchronous Advantage Actor Critic (A3C)

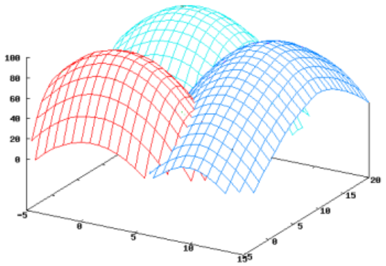


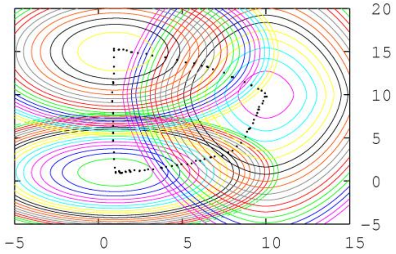
Multi-Objective Reinforcement Learning

Multi-Objective Reinforcement Learning (MORL)

- Many real-world tasks may present an agent with multiple, possibly conflicting objectives:
  - Time
  - Safety
  - Resource consumption
- Multi-Objective Reinforcement Learning allows an agent to learn how to prioritize among objectives at runtime
- Possible to create diverse populations of agents, or adapt agents to time-varying user needs, e.g. difficulty level or training session contents
- Training goals can also be considered by agents







J. Källström and F.Heintz, Tunable Dynamics in Agent-Based Simulation using Multi-Objective Reinforcement Learning, AAMAS Adaptive and Learning Agents Workshop 2019.

A Practical Guide to Multi-Objective Reinforcement Learning and Planning

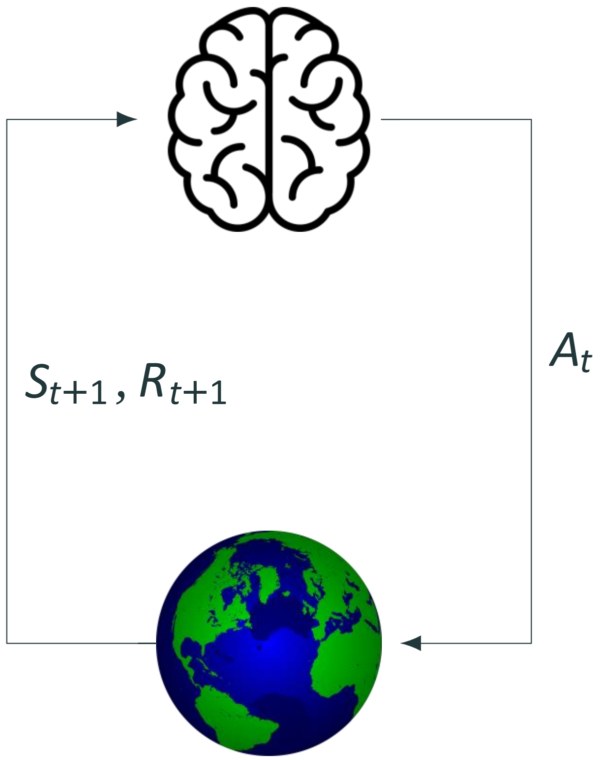
Conor F. Hayes*, Roxana Rădulescu*, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers

Autonomous Agents and Multi-Agent Systems. 2022

Slides from Connor Hayes

Reinforcement Learning

- Autonomous agent that learns via experience
- An environment which the agent can interact with
- The agent has a state in the environment
- At each step t the agent:
  - executes action A_t
  - receives state S_t
  - receives a scalar reward R_{t+1}



Reinforcement Learning [4]

- Reinforcement learning (RL) problems can be modelled as a Markov Decision Process (MDP)
- A MDP is a tuple: $(\mathrm{S}, \mathrm{A}, \mathrm{T}, \gamma, \mu, R)$,
  - S the state space
  - A the action space
  - $\mathrm{T} : \mathrm{S} \times \mathrm{A} \times \mathrm{S} \rightarrow [0, 1]$ is a probabilistic transition function
  - γ is a discount factor determining the relative importance of future rewards
  - $R : \mathrm{S} \times \mathrm{A} \times \mathrm{S} \rightarrow \mathrm{R}$ is a reward function, where r is the immediate reward.
  - $\mu : \mathrm{S} \rightarrow [0, 1]$ is a probability distribution over initial states
- In MDPs, the agent acts according to a policy π, where a policy is a mapping from states to actions
- The value function of a policy π is defined as follow $V^{\pi} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi, \mu \right]$

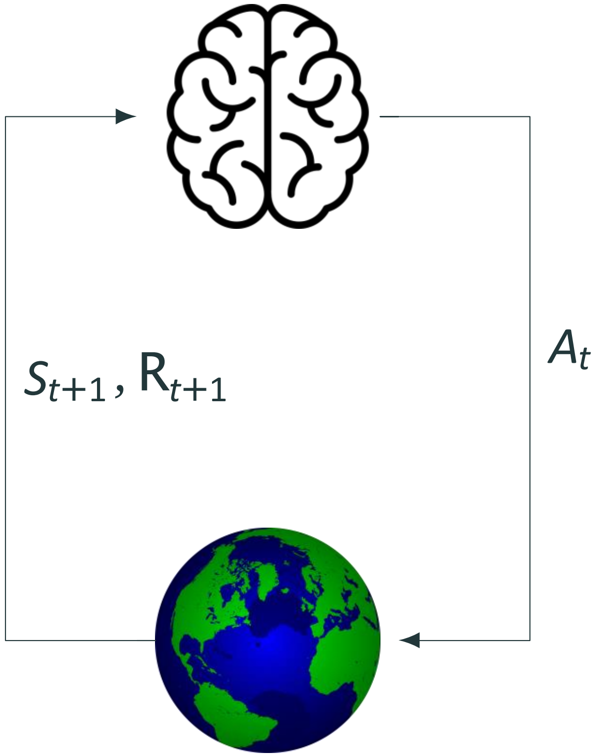
Slides from Connor Hayes

Reinforcement Learning

- Many real-world problems have multiple objectives
- Some RL approaches only consider a single objective
- Others combine the objectives linearly (GT Sophy)
- Such approaches oversimplify the problem and can produce sub-optimal results
    - For example: Power plant control
    - Objectives: maximise power output, minimise CO_2 emissions
        - Tuning the linear combination can be a difficult and iterative process
        - Should the behaviour be tuned by an AI engineer?
        - Why not just learn a set of optimal policies for all linear combinations?
- Solution: Take a multi–objective approach

Multi-Objective Reinforcement Learning

- Autonomous agent that learns via experience
- An environment which the agent can interact with
- The agent perceives a state in the environment
- At each step t the agent:
  - executes action A_t
  - receives state S_{t+1}
  - receives a vector reward
  - R_{t+1}



Multi-Objective Reinforcement Learning [1]

- Multi-Objective Reinforcement learning (MORL) problems can be modelled as a multi-objective Markov decision process (MOMDP)
- A MOMDP is a tuple: $(\mathrm{S}, \mathrm{A}, \mathrm{T}, \gamma, \mu, \mathrm{R})$,
    - S the state space
    - A the action space
    - $\mathrm{T} : \mathrm{S} \times \mathrm{A} \times \mathrm{S} \to [0, 1]$ is a probabilistic transition function
    - γ is a discount factor determining the relative importance of future rewards
    - $\mathrm{R}: \mathrm{S} \times \mathrm{A} \times \mathrm{S} \to \mathrm{R}^d$ is a vector valued reward function, where where $d >= 2$
    - $\mu : \mathrm{S} \to [0, 1]$ is a probability distribution over initial states
- In MOMDPs, the agent acts according to a policy π
- The value function of a policy π is defined as follows:

$$\mathbf{V}^{\pi} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \mid \pi, \mu \right]$$

Slides from Connor Hayes

Multi-Objective Reinforcement Learning

- A utility function, u, is used to represent a user's preferences over objectives
  - Utility function maps a vector reward to a scalar utility
    - $u :: \mathbb{R}^n \rightarrow \mathbb{R}$
  - For MORL, a utility function, u is assumed to be monotonically increasing:

$$(\forall i : \mathbf{V}_i^{\pi} \geq \mathbf{V}_i^{\pi'}) \wedge (\exists i : \mathbf{V}_i^{\pi} > \mathbf{V}_i^{\pi'}) \implies u(\mathbf{V}^{\pi}) \geq u(\mathbf{V}^{\pi'})$$

Multi-Objective Reinforcement Learning

- It is possible to compute different solutions sets like the Pareto front:

$$PF(\Pi) = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : \mathbf{V}^{\pi'} >_P \mathbf{V}^{\pi}\},$$

- where $>_P$ is the Pareto dominance relation,

$$\mathbf{V}^{\pi} >_P \mathbf{V}^{\pi'} \iff (\forall i : \mathbf{V}_i^{\pi} \geq \mathbf{V}_i^{\pi'}) \wedge (\exists i : \mathbf{V}_i^{\pi} > \mathbf{V}_i^{\pi'}).$$

- or the Convex Hull:

- $CH(\Pi) = \{\pi \in \Pi \mid \exists \mathbf{w}, \forall \pi' \in \Pi : \mathbf{w}^\top \mathbf{V}^{\pi} \geq \mathbf{w}^\top \mathbf{V}^{\pi'}\}$

- where $\mathbf{w}^\top \mathbf{V}^{\pi}$ computes the inner product of a weight vector $\mathbf{w}$
- and a value vector $\mathbf{V}^{\pi}$

Slides from Connor Hayes

Multi-Objective Reinforcement Learning

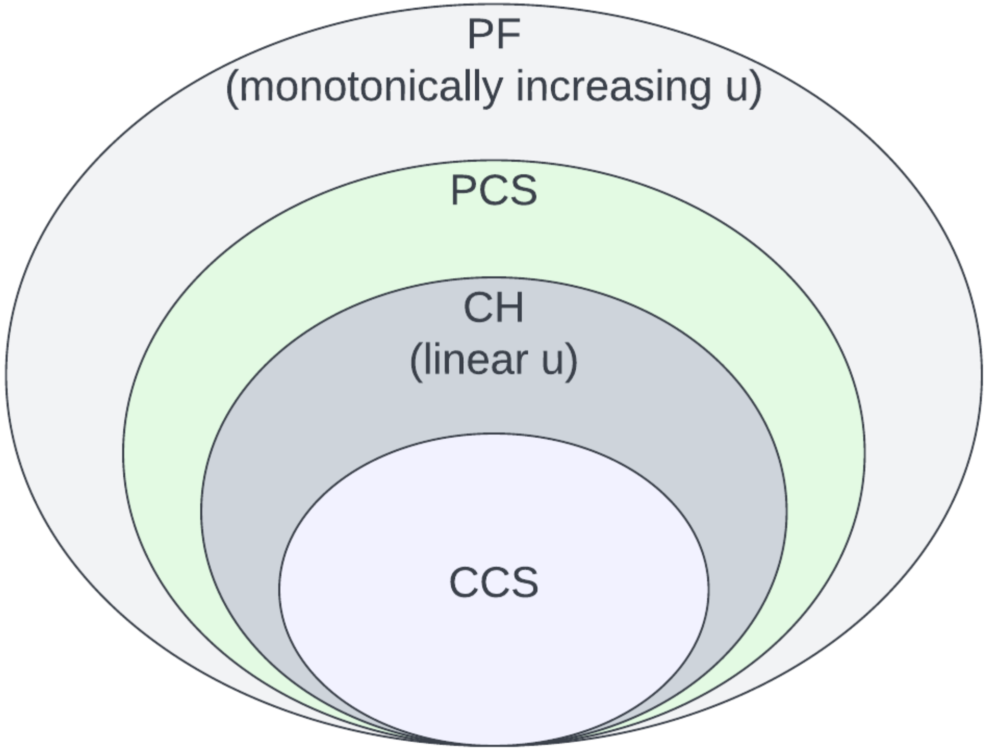


Figure 1: The Convex Hull is a subset of the Pareto front

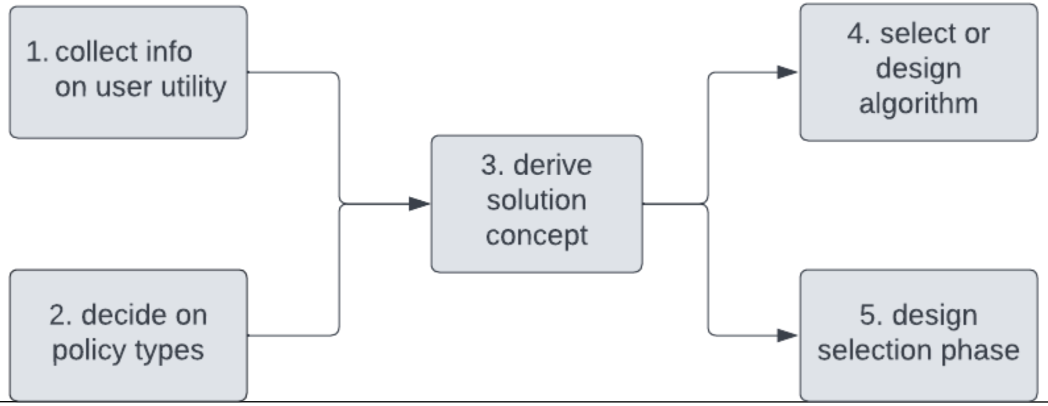
Slides from Connor Hayes

Axiomatic Approach

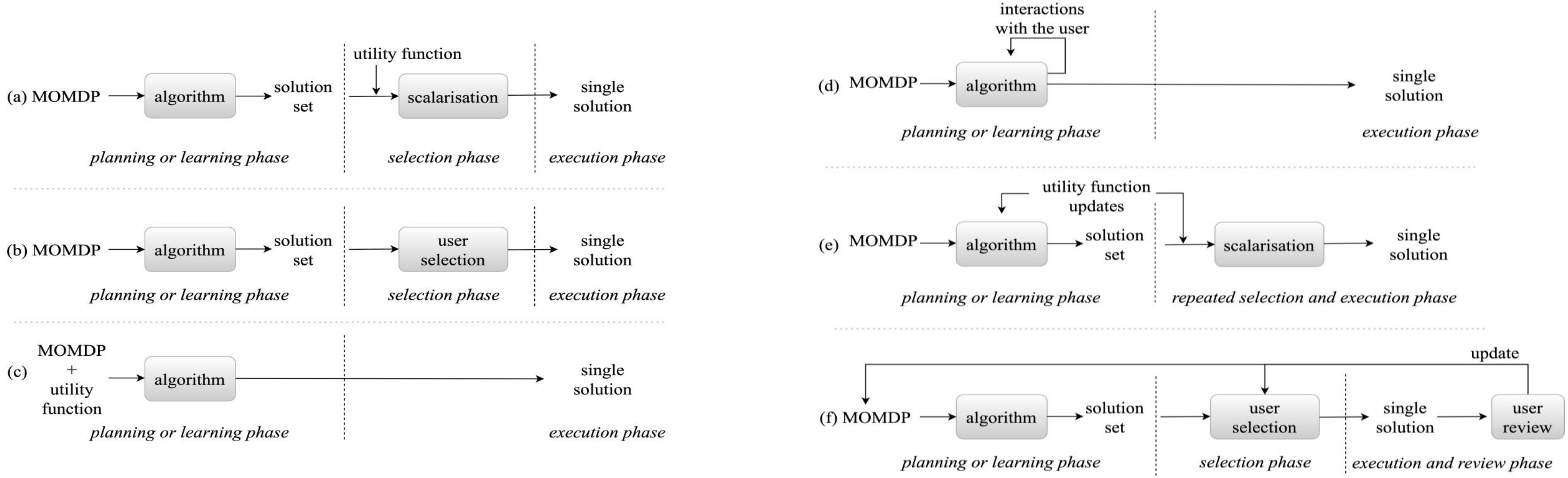
- The Pareto front is assumed to be the optimal solution set
- Solutions sets are derived without considering the utility function
- In practical settings the axiomatic approach may not be sufficient
  - In practical settings more information about the utility function of a user might be known (domain knowledge)
  - In practical settings the computing the Pareto front might be prohibitively expensive
  - It is not possible to encode domain knowledge when taking an axiomatic approach
  - Computation/time is wasted if the Pareto front is not actually the optimal set
    - Some policies on the Pareto front might be undesirable a priori (considering the knowledge of system expert)
- Solution: Take a utility-based approach

Utility-Based Approach

- Considering a utility function first is key to the successful application of AI in practical settings
- The properties of a user's utility may:
  - drastically alter the desired solution
  - change what methods are available (single-policy or multi-policy)
- The utility- based approach aims to derive the optimal solution set from the available knowledge about the user's utility function



MORL Scenarios



Slides from Connor Hayes

MORL Scenarios

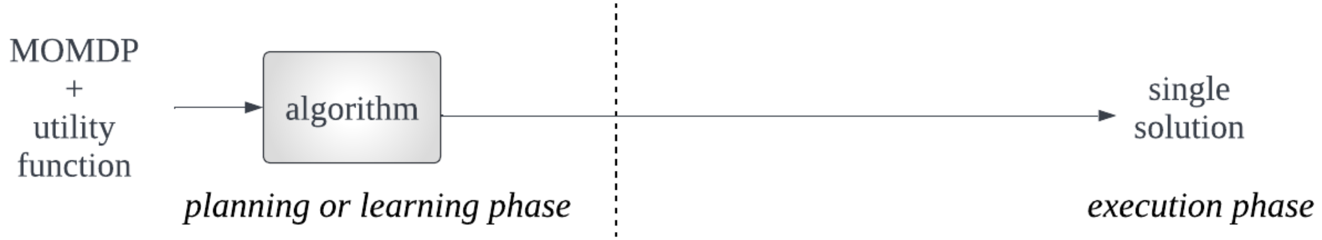


Figure 2: The known utility function scenario

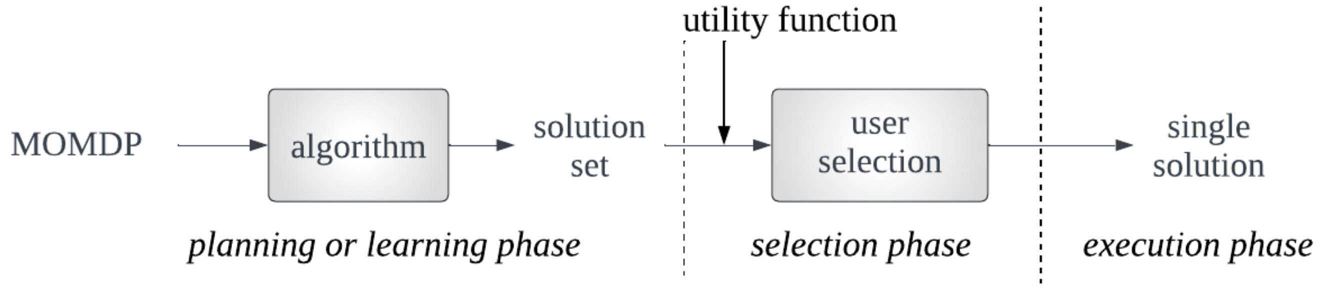


Figure 3: The unknown utility function scenario

Slides from Connor Hayes

Multi-Objective Reinforcement Learning Example

- For example, Deep Sea Treasure [5]
- Objectives: maximise treasure, minimise fuel
- reward = [treasure, -fuel]

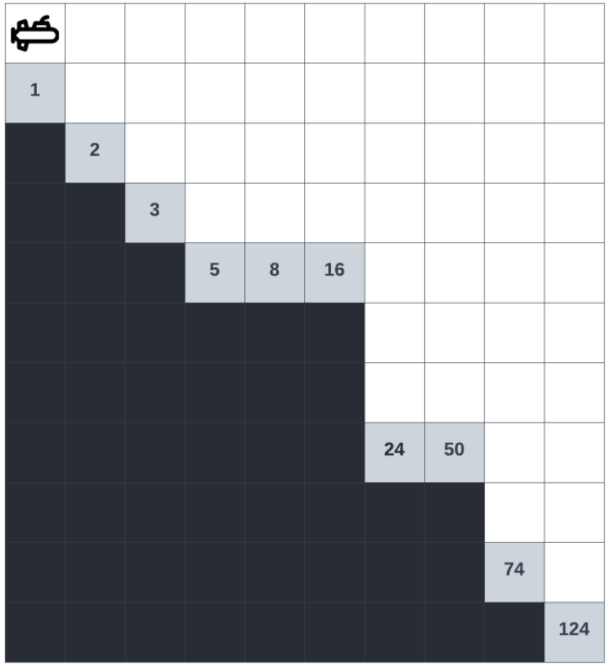


Figure 4: The Deep Sea Treasure environment

Multi-Objective Reinforcement Learning Example

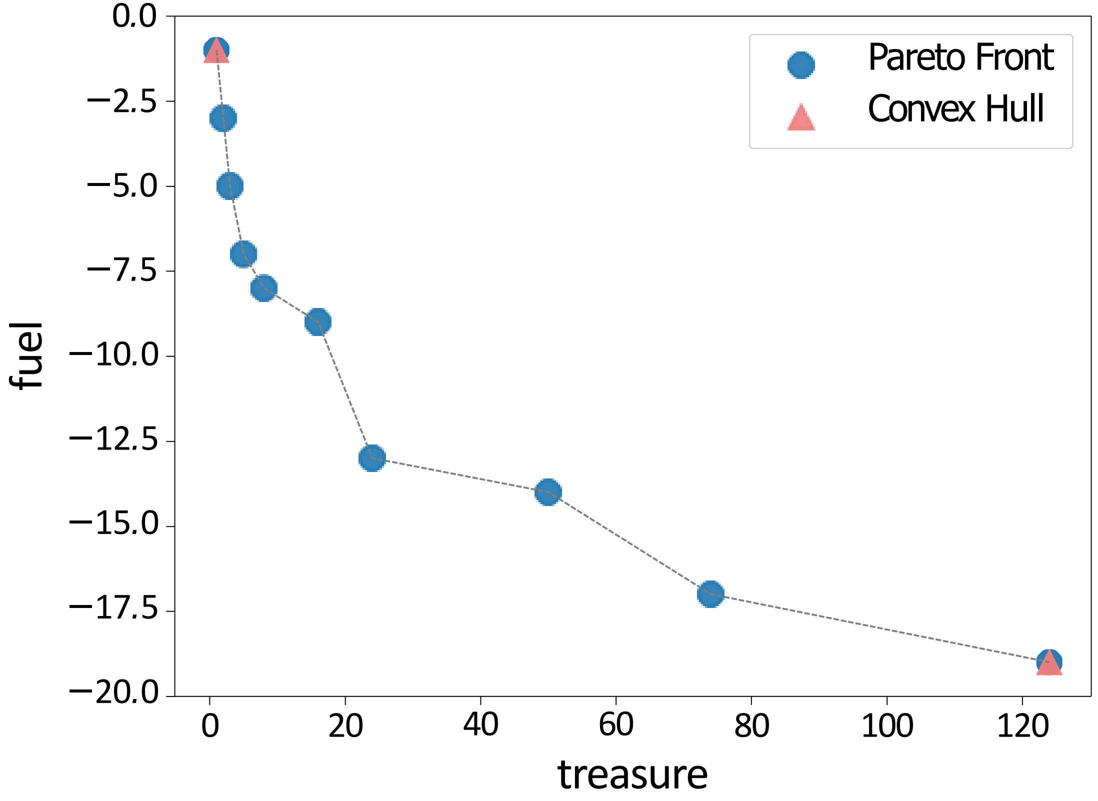
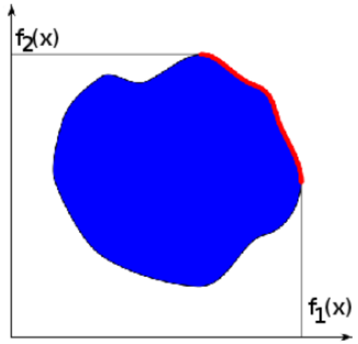


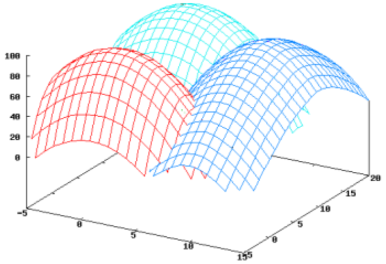
Figure 5: Pareto front (blue) and Convex Hull (red) of the Deep Sea Treasure benchmark problem

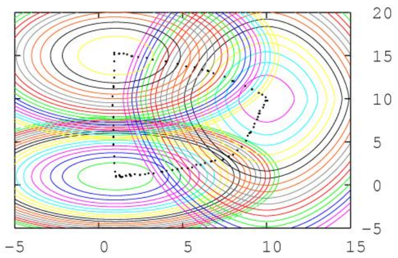
Slides from Connor Hayes

Multi-Objective Reinforcement Learning (MORL)

- Many real-world tasks may present an agent with multiple, possibly conflicting objectives:
  - Time
  - Safety
  - Resource consumption
- Multi-Objective Reinforcement Learning allows an agent to learn how to prioritize among objectives at runtime
- Possible to create diverse populations of agents, or adapt agents to time-varying user needs, e.g. difficulty level or training session contents
- Training goals can also be considered by agents







J. Källström and F.Heintz, Tunable Dynamics in Agent-Based Simulation using Multi-Objective Reinforcement Learning, AAMAS Adaptive and Learning Agents Workshop 2019.

References I

- Hayes, C.F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L.M., Dazeley, R., Heintz, F., Howley, E., Irissappane, A.A., Mannion, P., Nowé, A., Ramos, G., Restelli, M., Vamplew, P., Roijers, D.M.: A practical guide to multi-objective reinforcement learning and planning. Autonomous Agents and Multi-Agent Systems 36(1), 26 (2022). URL https://doi.org/10.1007/s10458-022-09552-y
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. nature 518(7540), 529–533 (2015)

References II

- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science 362(6419), 1140–1144 (2018). URL https://www.science.org/doi/abs/10.1126/science.aar6404
- Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Proceedings of the 12th International Conference on Neural Information Processing Systems, NIPS'99, p. 1057–1063. MIT Press, Cambridge, MA, USA (1999)

References III

- Vamplew, P., Yearwood, J., Dazeley, R., Berry, A.: On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In: W. Wobcke, M. Zhang (eds.) AI 2008: Advances in Artificial Intelligence, pp. 372–378. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
- Wurman, P.R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T.J., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F., et al.: Outracing champion gran turismo drivers with deep reinforcement learning. Nature 602(7896), 223–228 (2022)

Slides from Connor Hayes

TDDC17 AI LE7 HT2023:
Reinforcement learning
Deep reinforcement learning
Multi-objective reinforcement learning

www.ida.liu.se/~TDDC17

LINKÖPING UNIVERSITY