REINFORCEMENT LEARNING

@AI4EIC 2023

Haipeng Chen

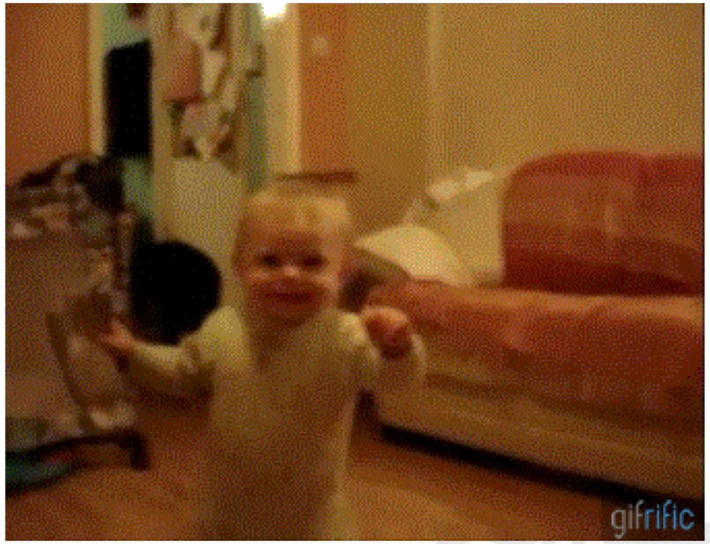
Assistant professor, Data Science

hchen23@wm.edu

WILLIAM & MARY | DATA SCIENCE

Reinforcement Learning in Humans

GIPHY



GIPHY

Humans appear to learn behaviors through "trial and error"

What is an example of Reinforcement Learning?

What is an example of Reinforcement Learning?

H

What is reinforcement learning?

Reinforcement Learning (RL) is a type of machine learning that focuses on training algorithms to make sequences of decisions to maximize a reward signal. It's inspired by behavioral psychology, where learning occurs through interaction with an environment to achieve specific goals. In RL, an agent interacts with an environment, learns from its actions, and adapts its behavior to achieve better outcomes over time.

What is an example of Reinforcement Learning?

H

What is reinforcement learning?

Reinforcement Learning (RL) is a type of machine learning that focuses on training algorithms to make sequences of decisions to maximize a reward signal. It's inspired by behavioral psychology, where learning occurs through interaction with an environment to achieve specific goals. In RL, an agent interacts with an environment, learns from its actions, and adapts its behavior to achieve better outcomes over time.

What is an example of Reinforcement Learning?

H

What is reinforcement learning?

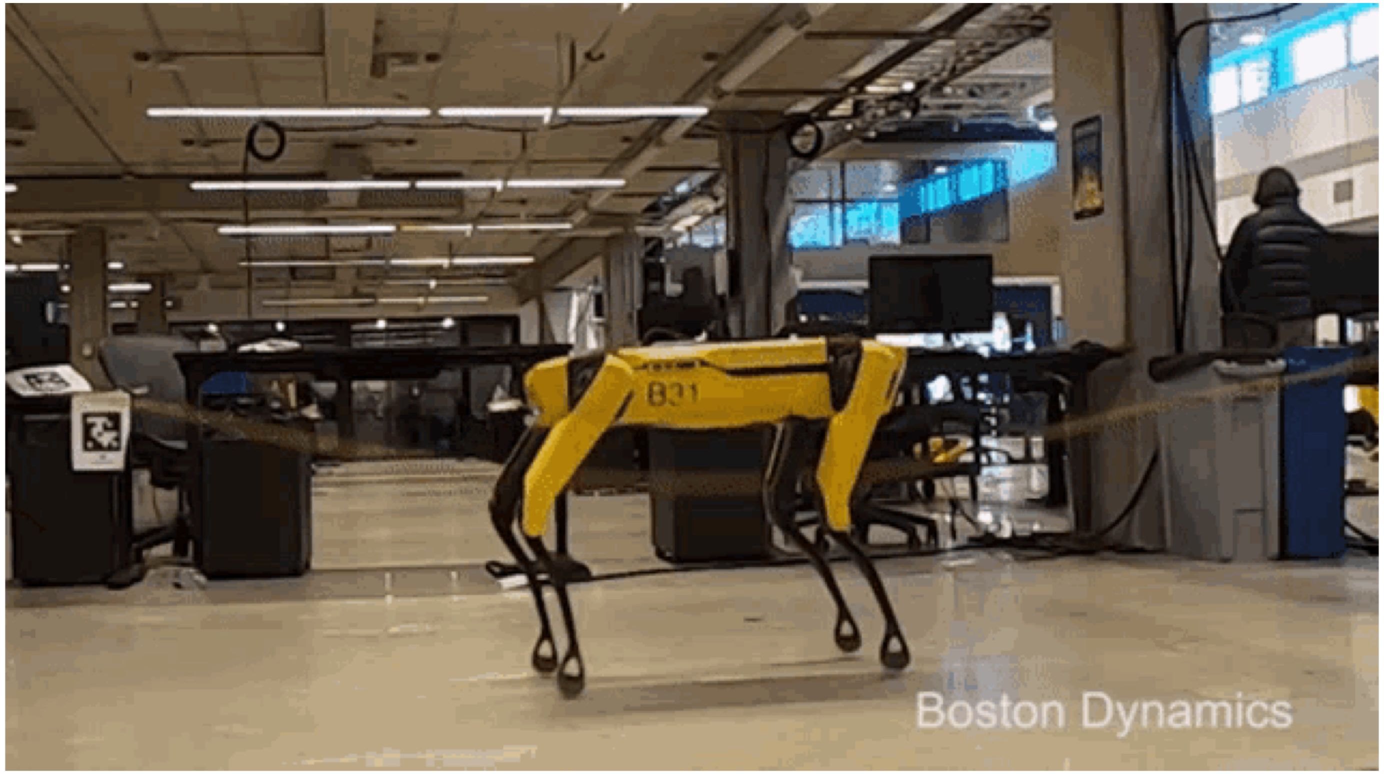
Reinforcement Learning (RL) is a type of ma[illegible]ocuses on training algorithms to make sequen[illegible]aximize a reward signal. It's inspired by behavioral [illegible]arning occurs through interaction with an environment to ac[illegible]goals. In RL, an agent interacts with an environment, learns from its actions, and adapts its behavior to achieve better outcomes over time.

Reinforcement Learning from Human Feedback (RLHF)

What are other examples?

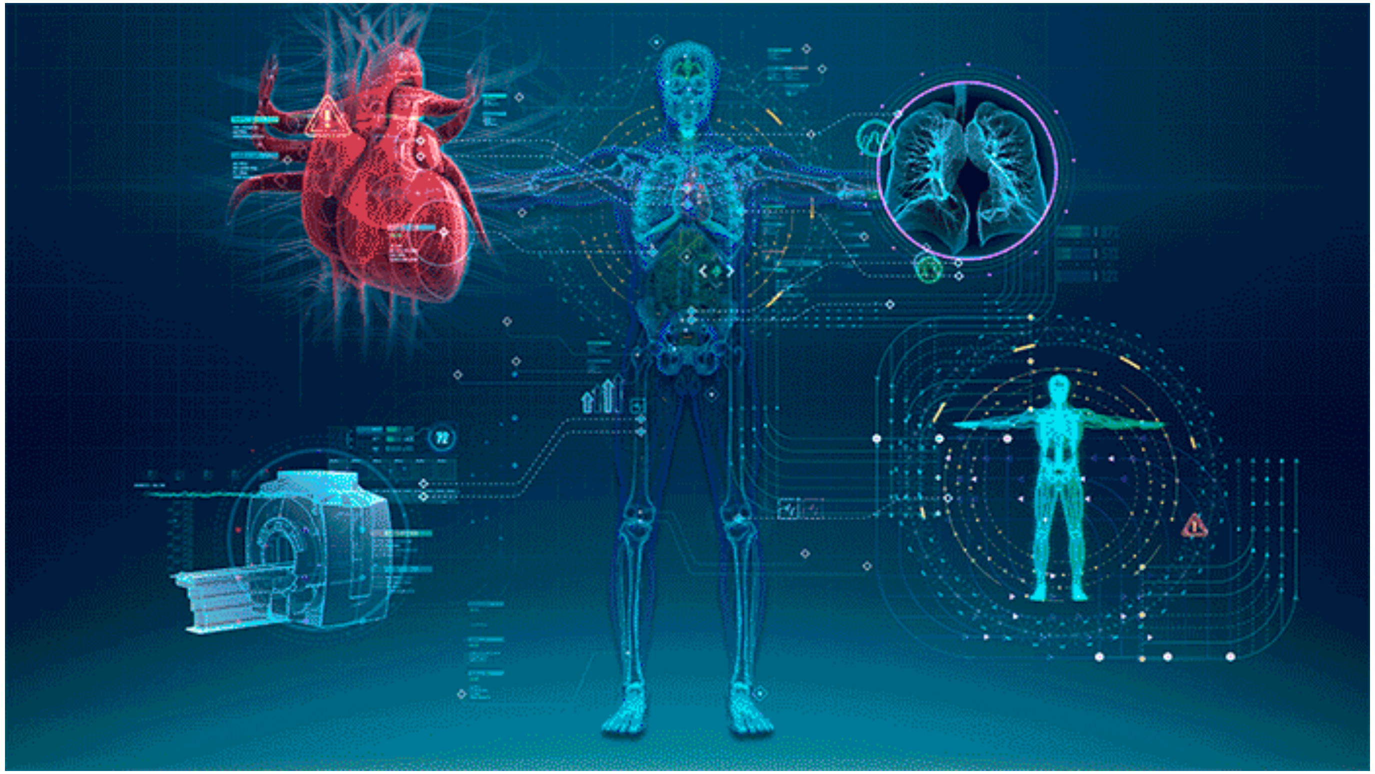
Games AI

What are other examples?



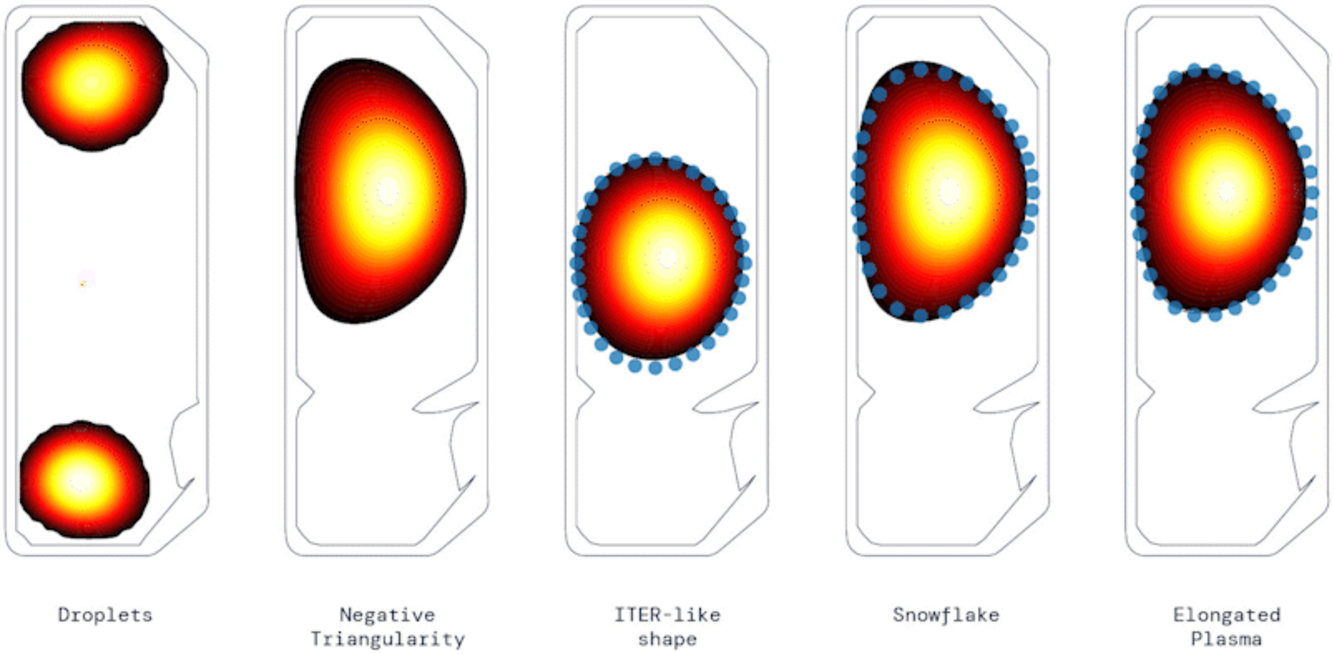
[Boston Dynamics Spot]

What are other examples?



[Precision health]

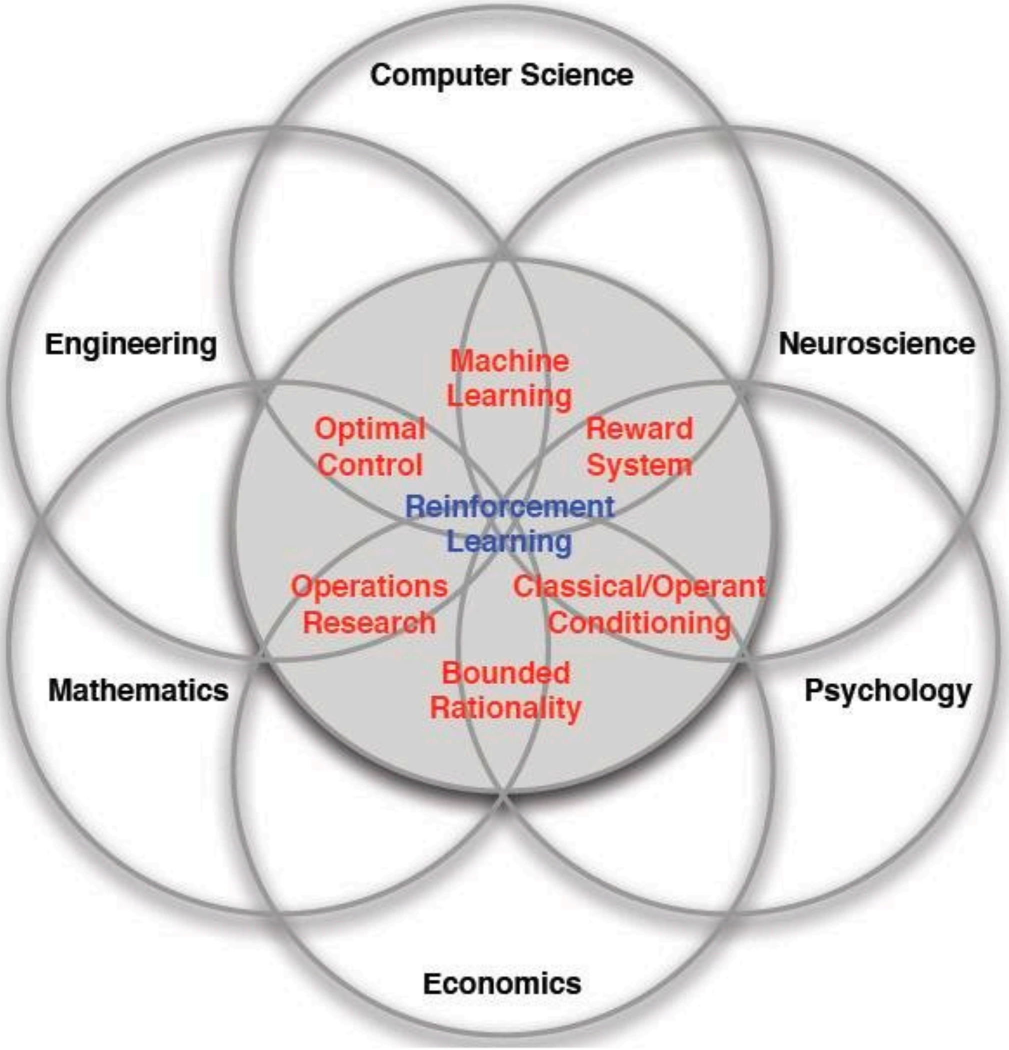
What are other examples?



[Nuclear fusion plasma control]

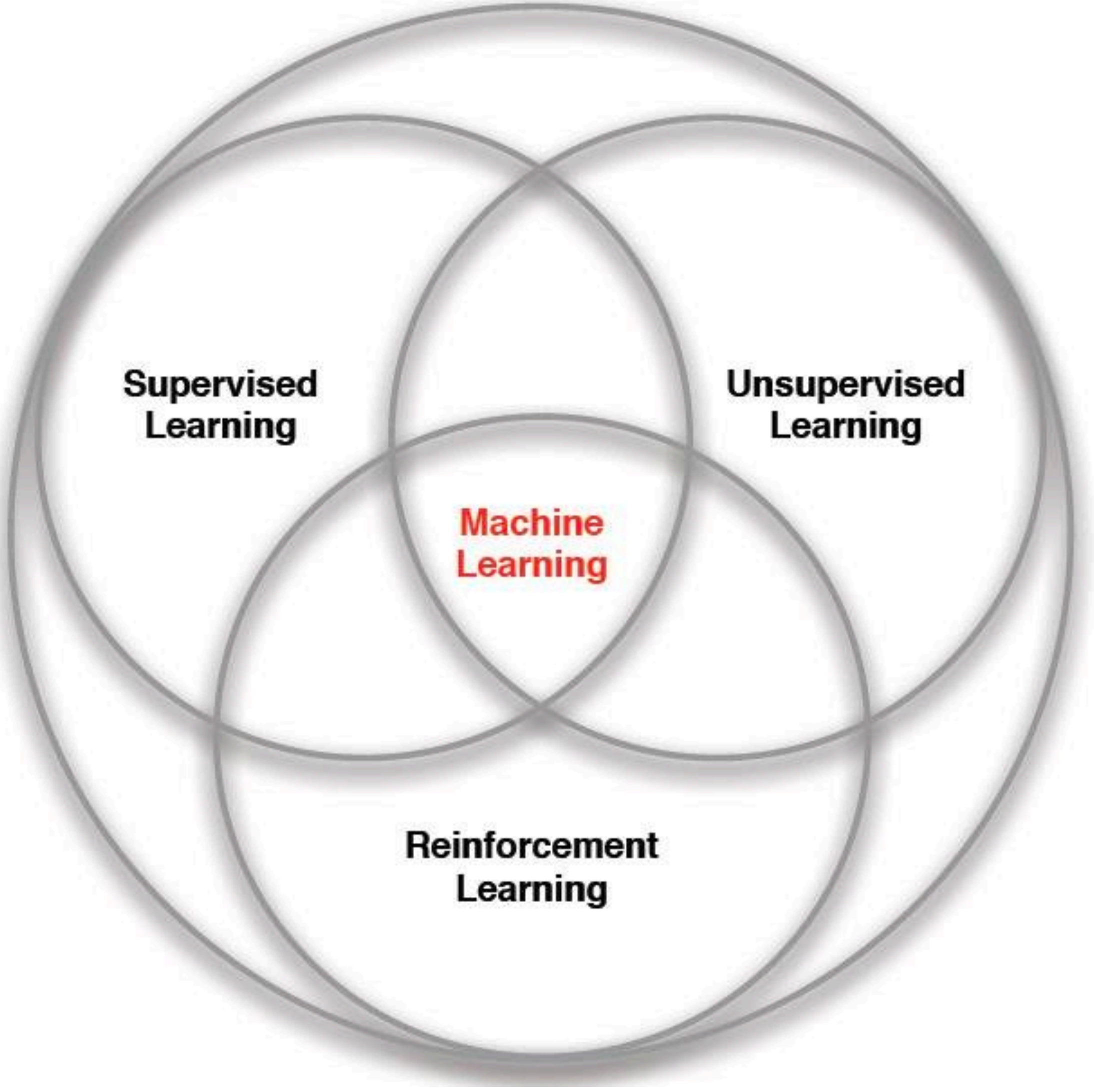
Degrave et al., Nature, 2022

Many Faces of Reinforcement Learning



Source: [David Silver RL lecture]

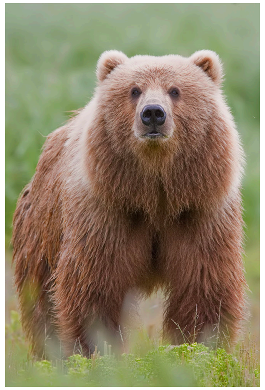
Branches of Machine Learning

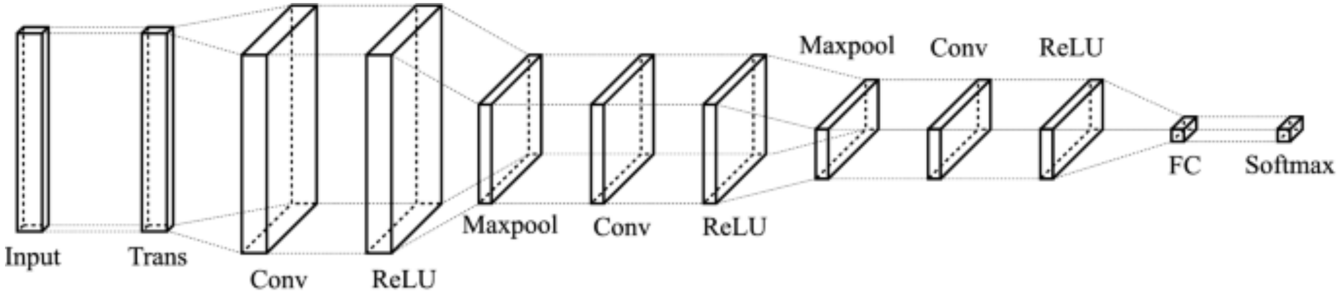


Source: [David Silver RL lecture]

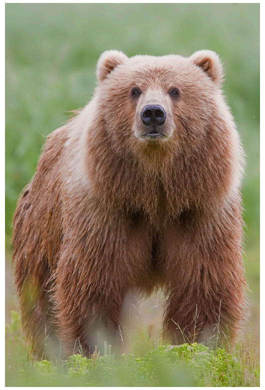
RL vs Supervised Learning?

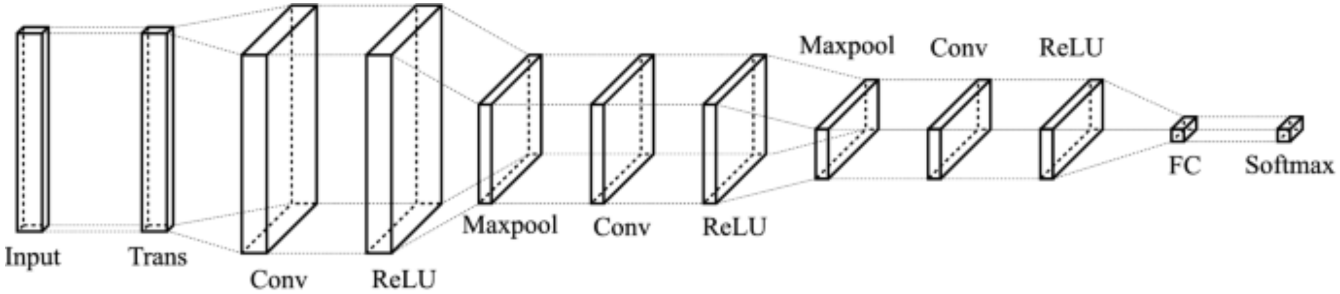
RL vs Supervised Learning?





RL vs Supervised Learning?

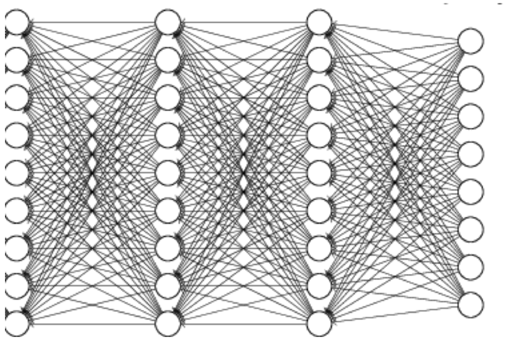




Cat

Dog

Bear



Alive

Run away
(Take action)

"Policy"

RL vs Supervised Learning?

- Action/decision is involved
- Reward signal (vs class label)
- Non i.i.d data: agent's actions affect the subsequent data it receives

Outline

- What is reinforcement learning?
- The reinforcement learning problem
- Value-based
- Policy-based

The RL Problem

Environment

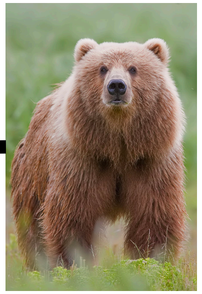
The RL Problem

Environment

Agent

W&M DATA SCIENCE

The RL Problem



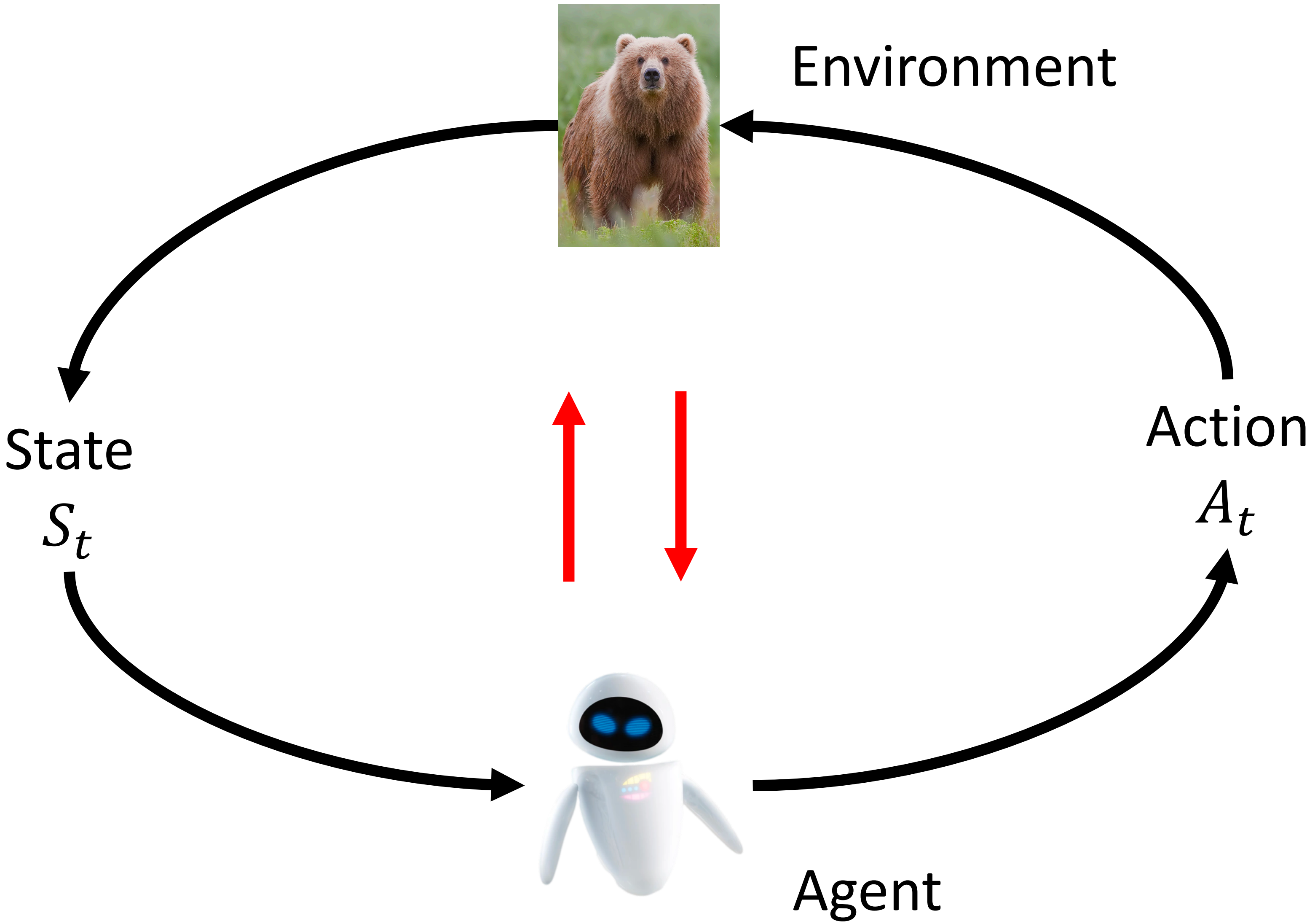
Environment

State

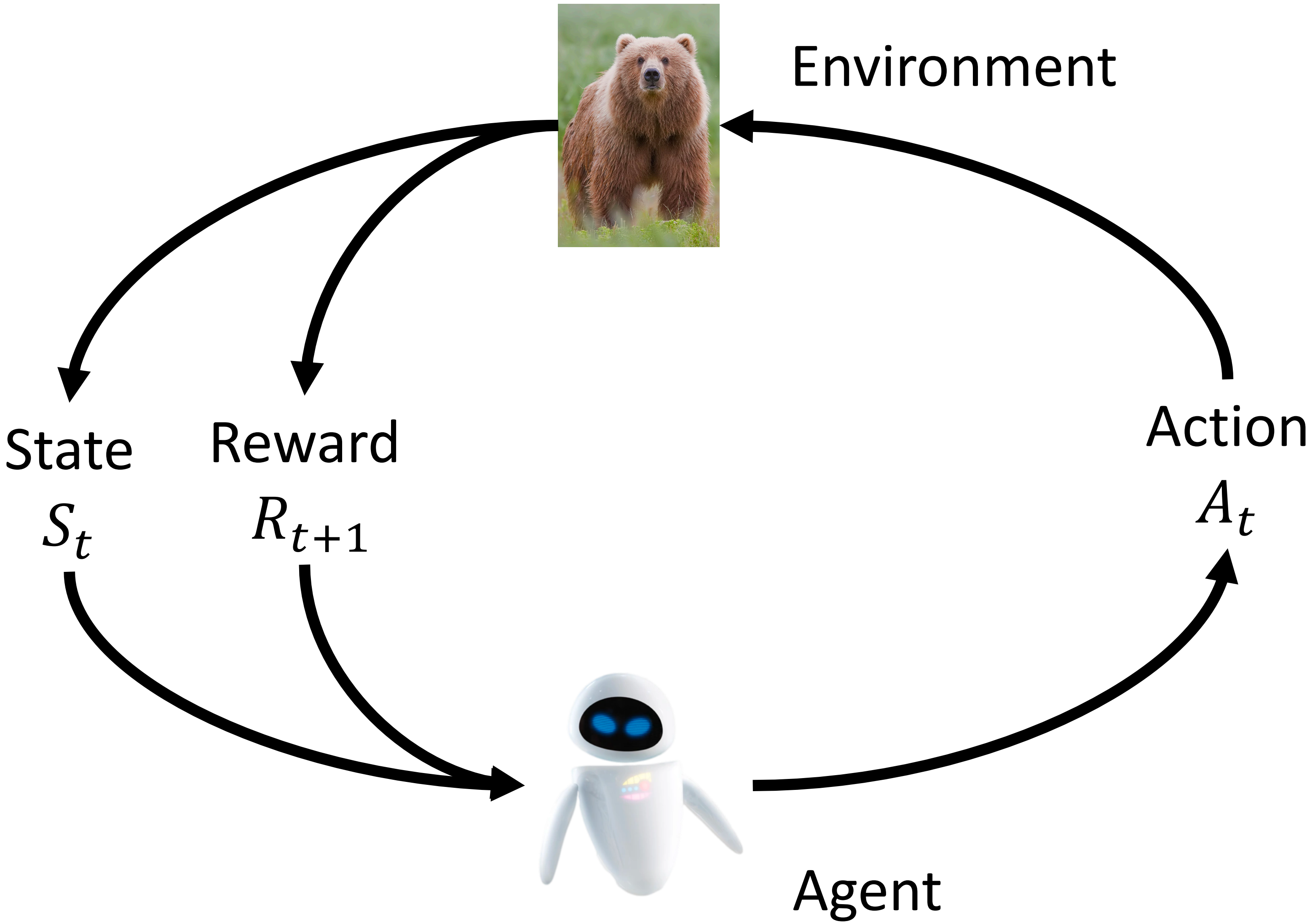
S_t

Agent

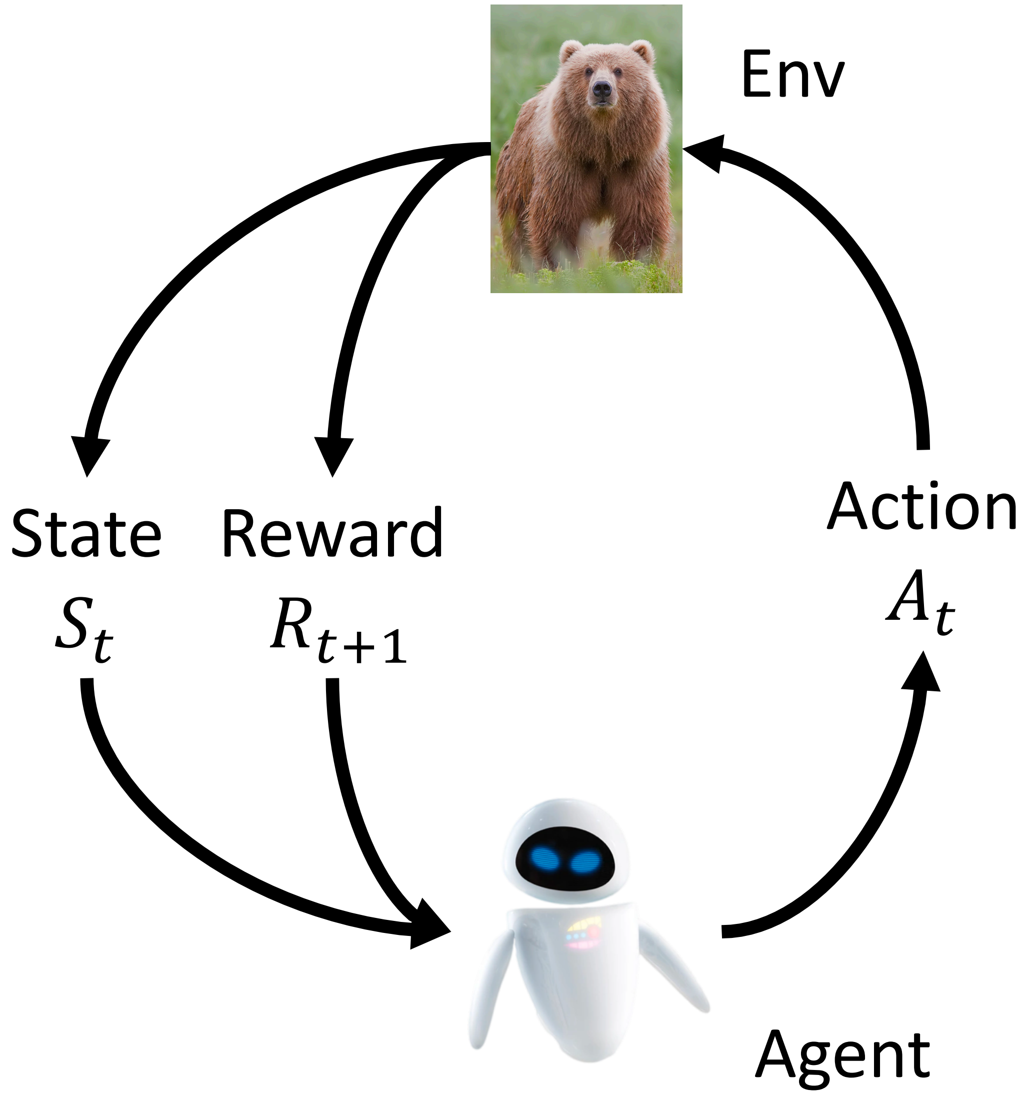
The RL Problem



The RL Problem



The RL Problem

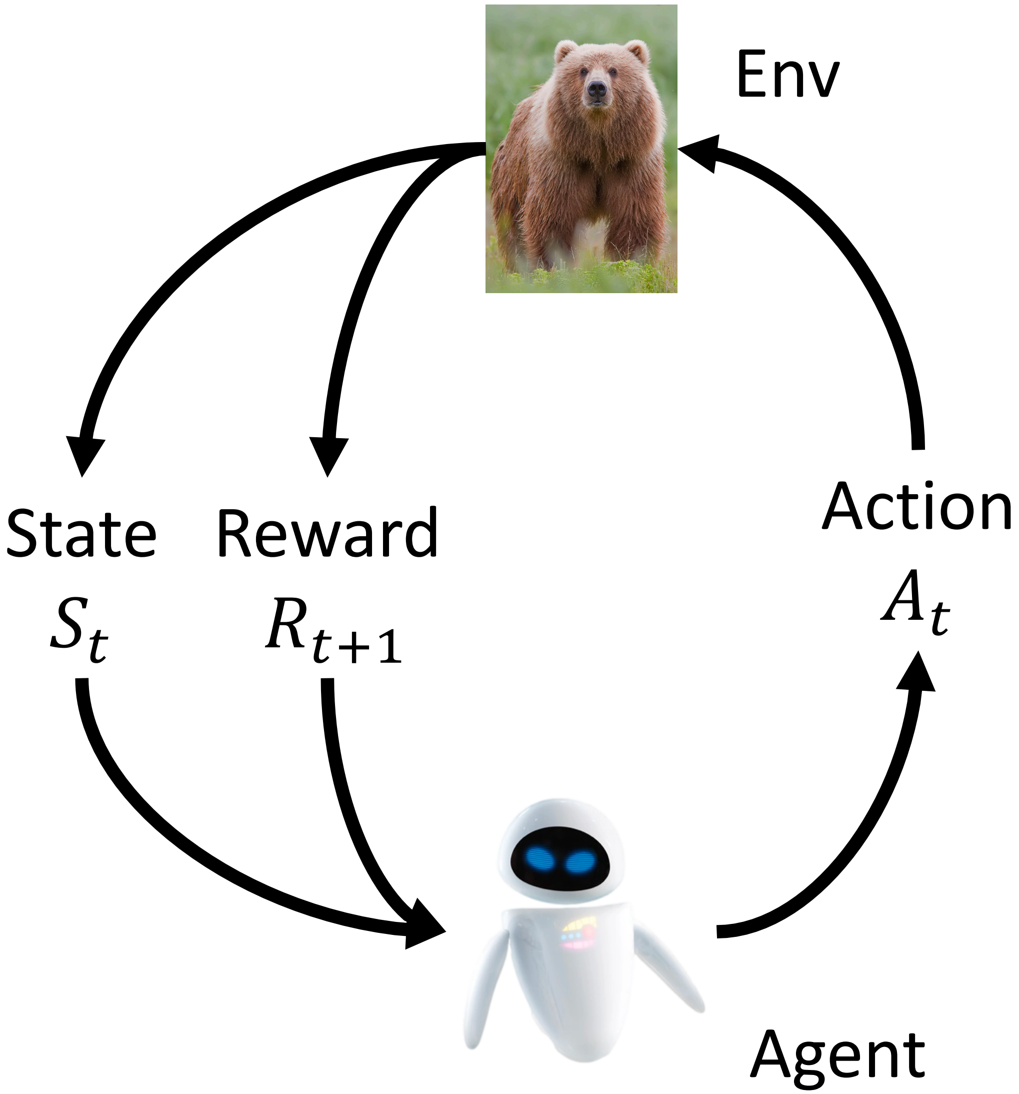


- The agent:
  - Observes state S_t
  - Takes action A_t
  - Receives reward R_t

- The environment:
  - Receives action A_t
  - Emits next state S_{t+1}
  - Emits reward $R_{\boldsymbol{t+1}}$

- Iteration: $t \rightarrow t+1$

The RL Problem: MDP

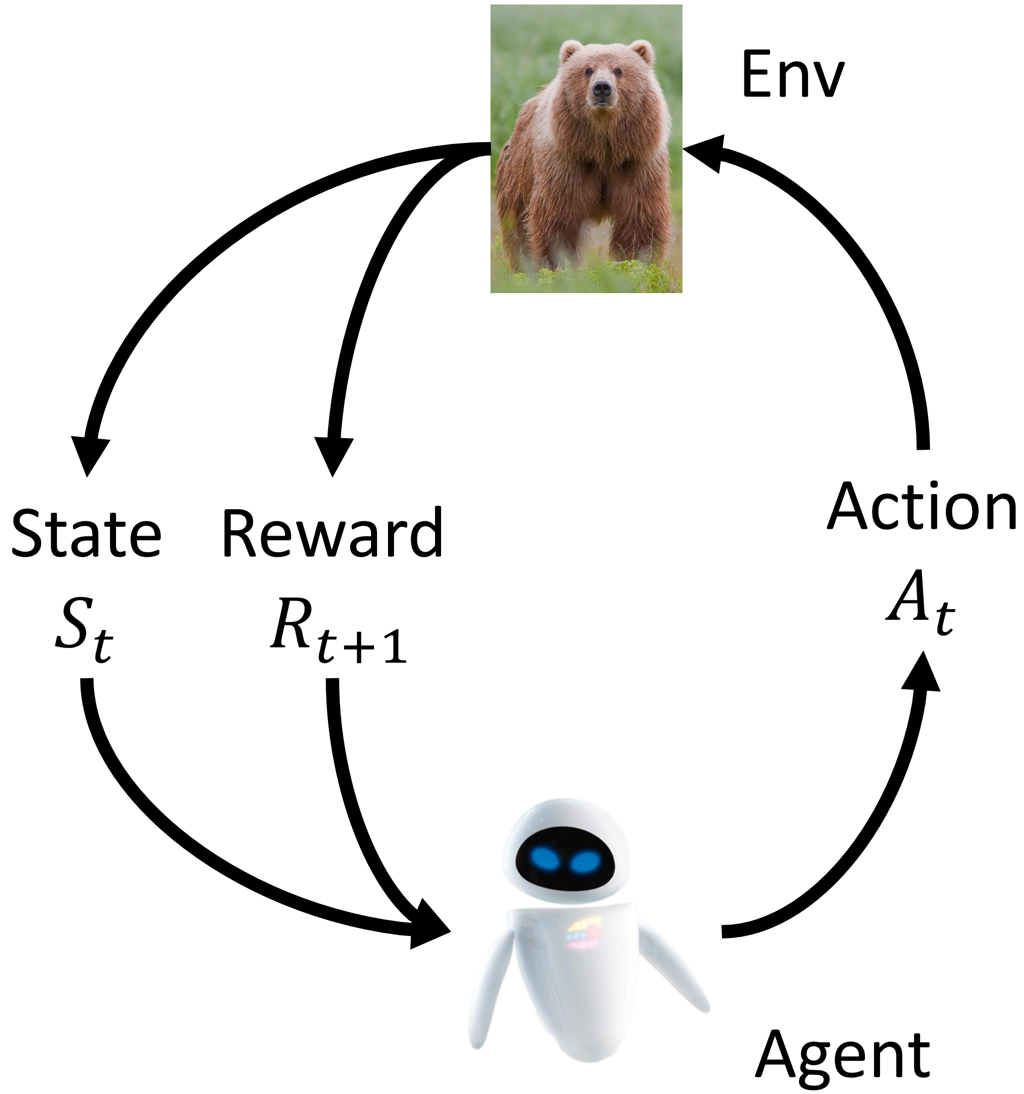


Definition (MDP)

A **Markov Decision Process (MDP)** is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix
  $$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
- $\mathcal{R}$ is a reward function
  $$\mathcal{R}_s^a = \mathbb{E}[\mathcal{R}_{t+1} \mid S_t = s, A_t = a]$$
- $\gamma \in [0,1]$ is a discount factor

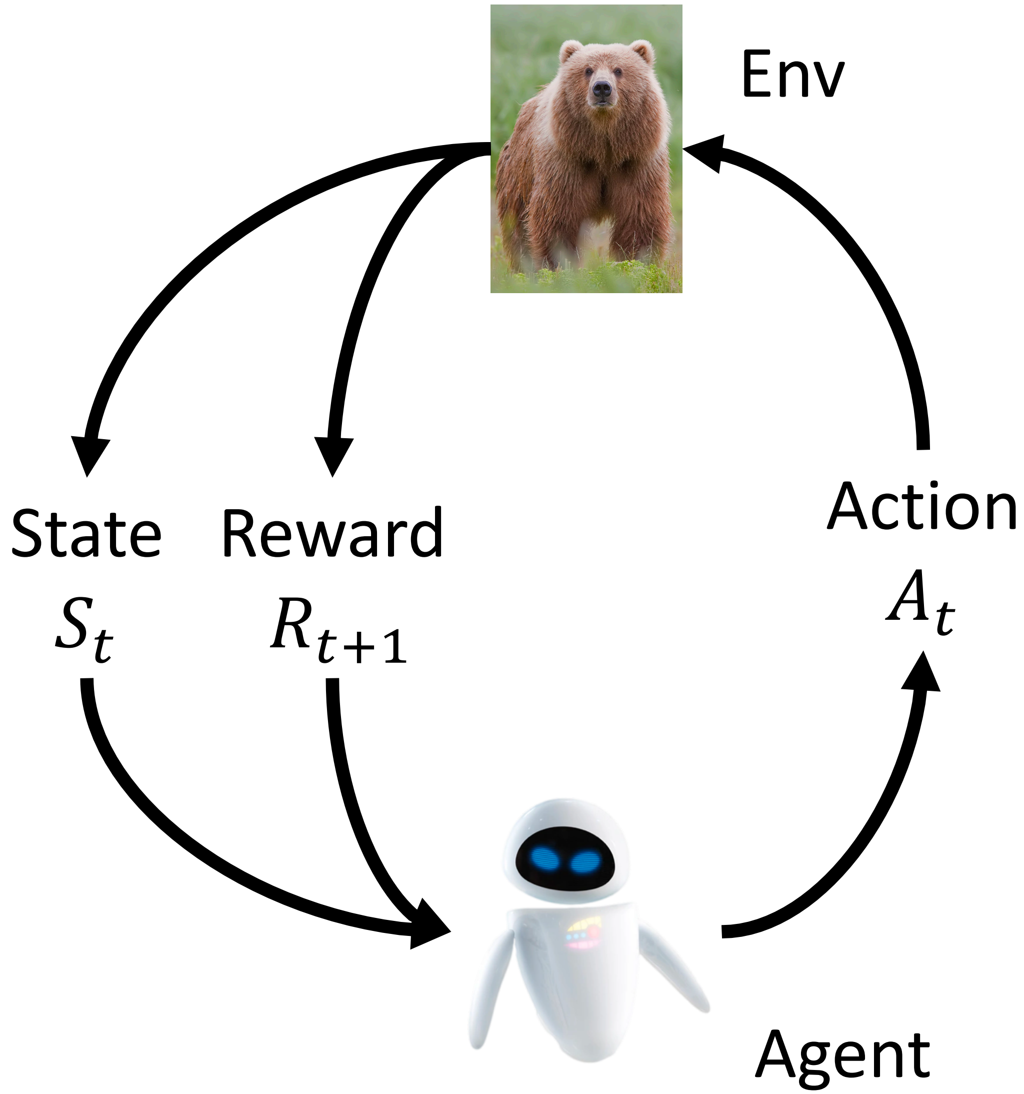
The RL Problem: Return



Return (total discounted reward)

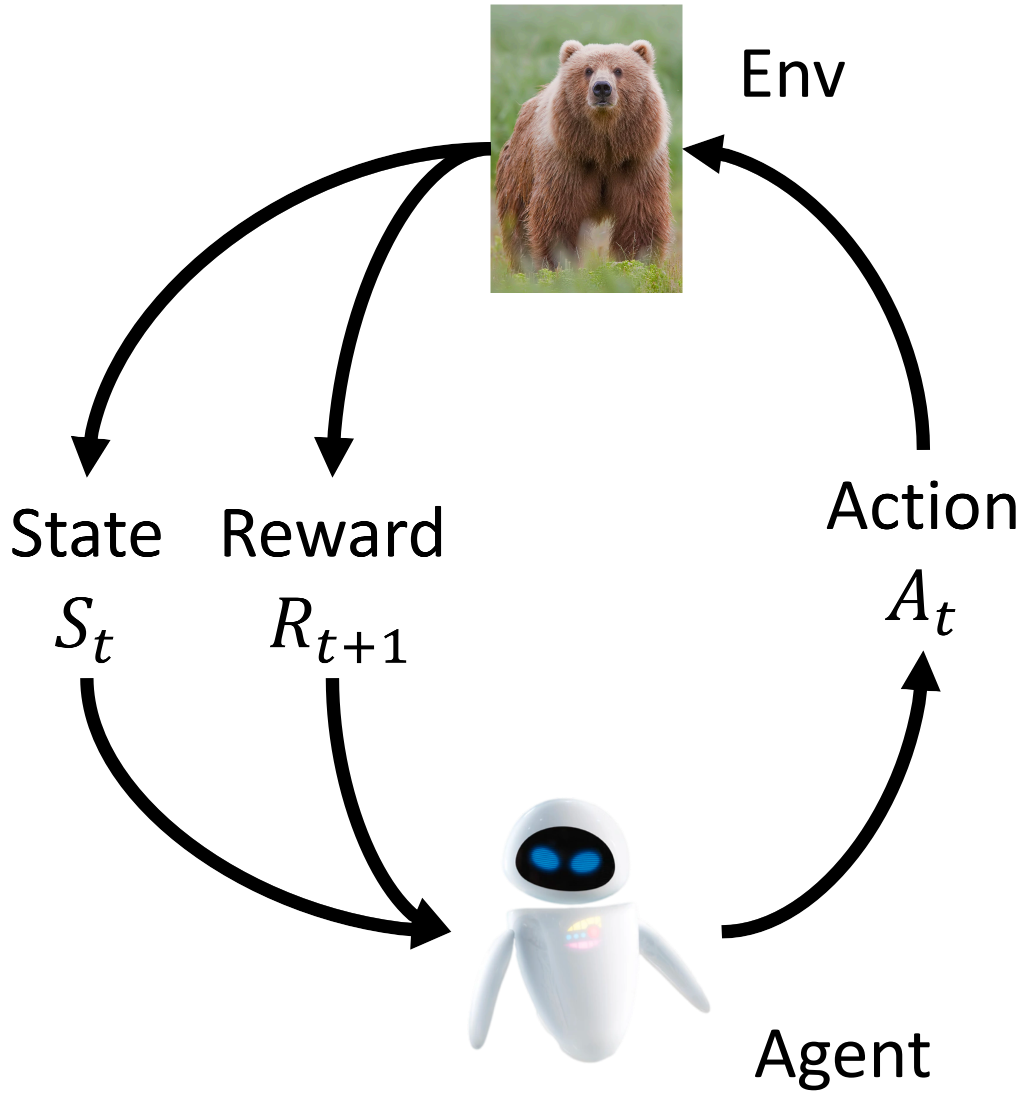
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots$$
$$= \sum_{\tau=0}^{\infty} \gamma^{\tau} R_{t+\tau+1}$$

The RL Problem: RL Agent



- An RL agent includes one or more of the following:
  - Policy: agent's behavior function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

The RL Problem: RL Agent



- An RL agent includes one or more of the following:
  - Policy: agent's behavior function
  - Value function: how good is each state and/or action
  - ~~Model: agent's representation of the environment~~

Policy

- A policy π is a distribution over actions given states,

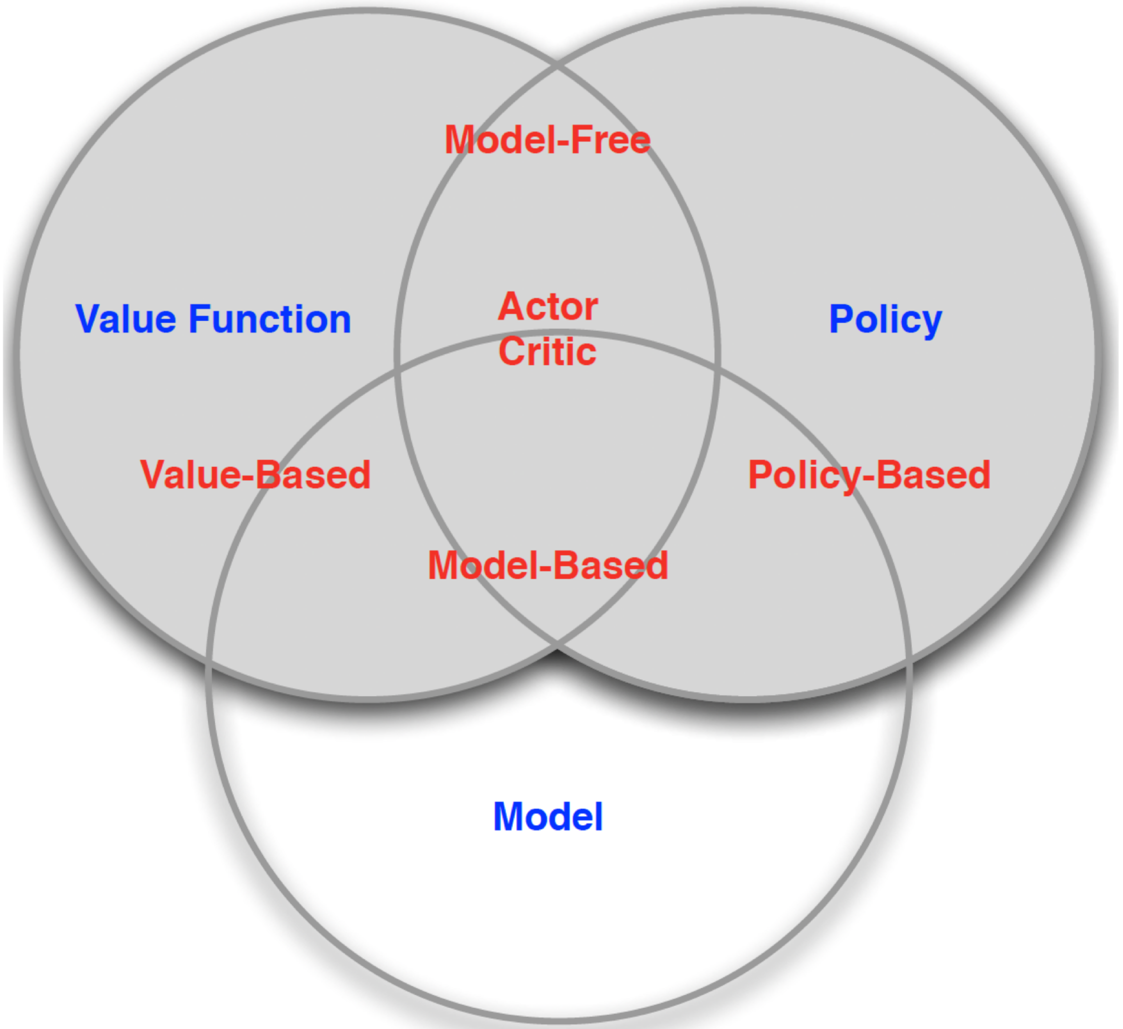
$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

Value Function

- The action-value function $q_\pi(s, a)$ of an MDP is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

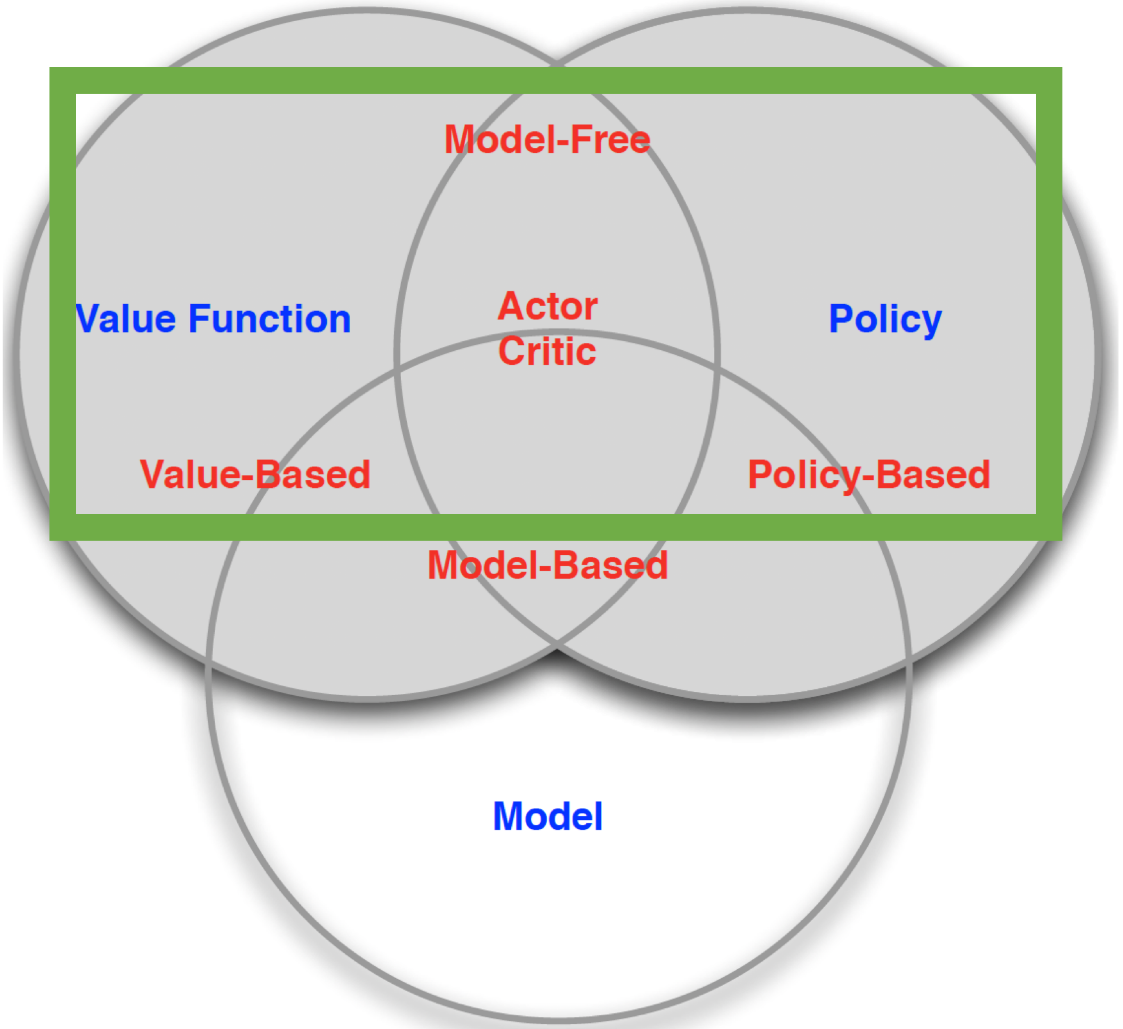
RL Taxonomy



Source: [David Silver RL lecture]

W&M DATA SCIENCE

This Tutorial



Source: [David Silver RL lecture]()

This Tutorial

Value-based:

Find **optimal** $\boldsymbol{q^*(s,a)}$ as proxy of policy:

$$q^*(s,a) = \max_{\pi} q_{\pi}(s,a)$$

Policy-based:

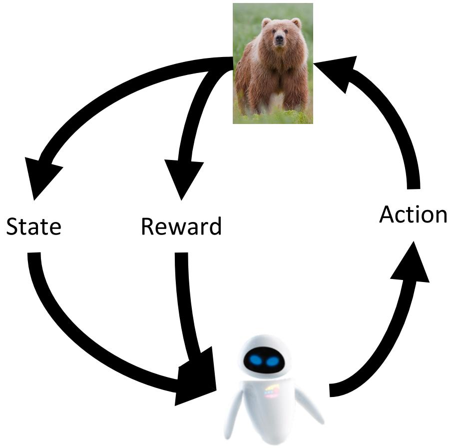
Directly find **optimal** $\boldsymbol{\pi^*(a|s)}$:

$$\pi^*(a|s) = \operatorname*{argmax}_{\pi} v^{\pi}(s)$$

Outline

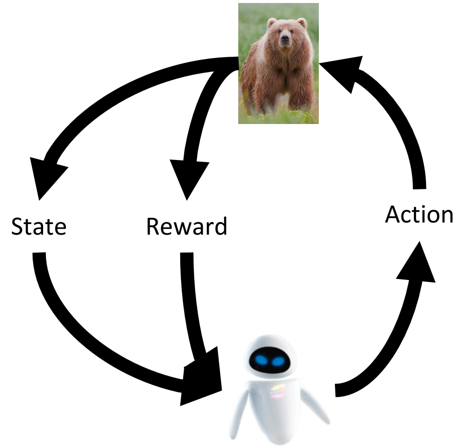
- What is reinforcement learning?
- The reinforcement learning problem
- Value-based
- Policy-based

Q-Learning: Training Procedure



- For each training episode:
  - Start in the initial state:
  - Loop until a terminal state is reached:
    - Choose an action based on policy π derived from $Q(s, a)$
    - Take the action and observe next state and reward
    - Update $Q(s, a)$ for the current state-action pair
    - Move to the next state

Q-Learning: Training Procedure



- For each training episode:
  - Start in the initial state:
  - Loop until a terminal state is reached:
    - Choose an action based on policy π derived from $Q(s, a)$
    - Take the action and observe next state and reward
    - Update $Q(s, a)$ for the current state-action pair
    - Move to the next state

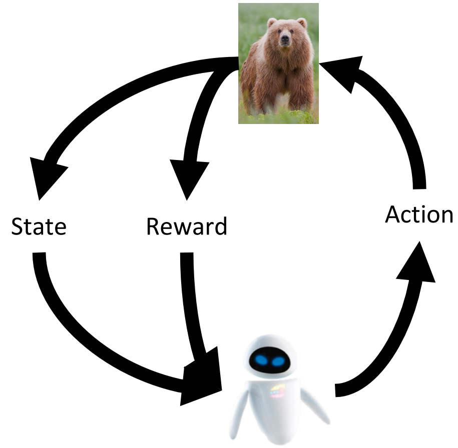
Q-Learning: Policy

- Policy π is derived from actioin value function $Q(s, a)$, e.g., ϵ-greedy for exploration

$$A_t = \begin{cases} \text{argmax}_{a'} Q(S_t, a'), & \text{with prob. } 1-\epsilon \\ \text{any } a, & \text{with prob. } \epsilon \end{cases}$$

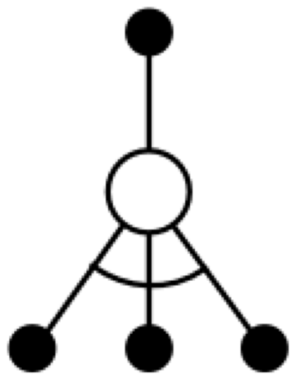
W&M DATA SCIENCE

Q-Learning: Training Procedure



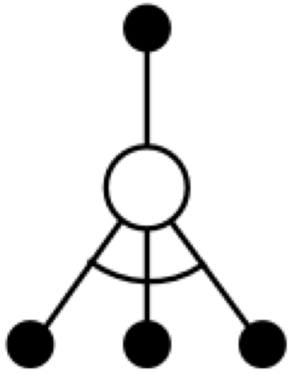
- For each training episode:
  - Start in the initial state:
  - Loop until a terminal state is reached:
    - Choose an action based on policy π derived from $Q(s, a)$
    - Take the action and observe next state and reward
    - Update $Q(s, a)$ for the current state-action pair
    - Move to the next state

Q-Learning: Bellman Optimality Equation



$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

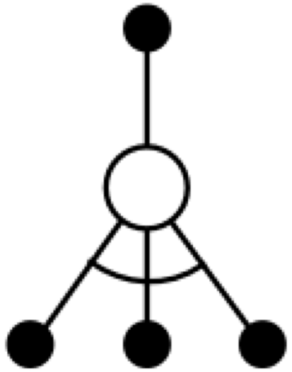
Q-Learning: Bellman Optimality Equation



$$Q(S,A) \leftarrow Q(S,A) + \alpha\Big(\underbrace{R + \gamma \max_{a'} Q(S',a')}_{\text{Target}} - Q(S,A)\Big)$$

W&M DATA SCIENCE

Q-Learning: Bellman Optimality Equation



$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(\underbrace{R + \gamma \max_{a'} Q(S',a')}_{\text{Target}} - \underbrace{Q(S,A)}_{\text{Predicted}}\right)$$

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve *large* problems, e.g.

- Backgammon: 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

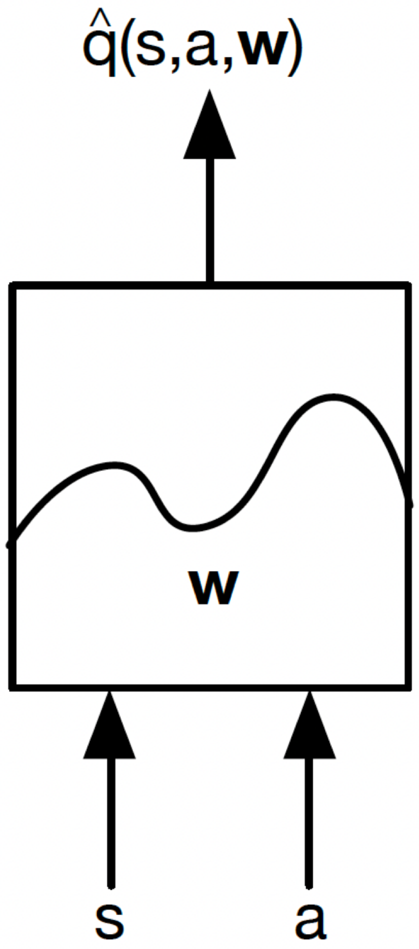
- Tabular Q-Learning for large MDPs:
  - *Memory*: too many states and/or actions to store
  - *Computation*: too slow to learn the value of each state individually

Value Function Approximation

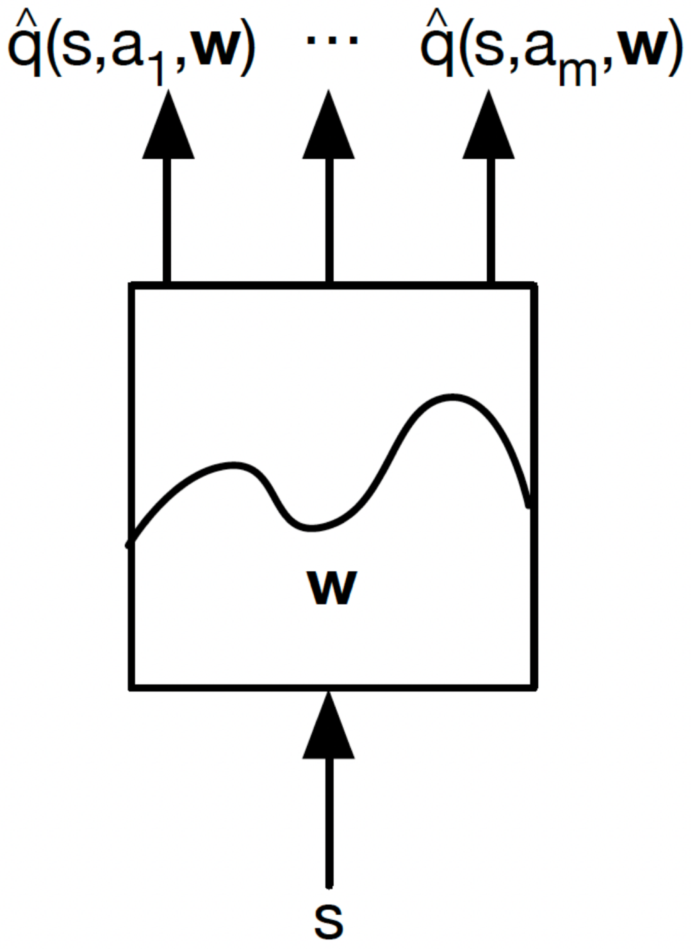
- Solution for large MDPs:
  - Estimate value function with function approximation
  - e.g., deep neural networks

$$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

Types of Value Function Approximation



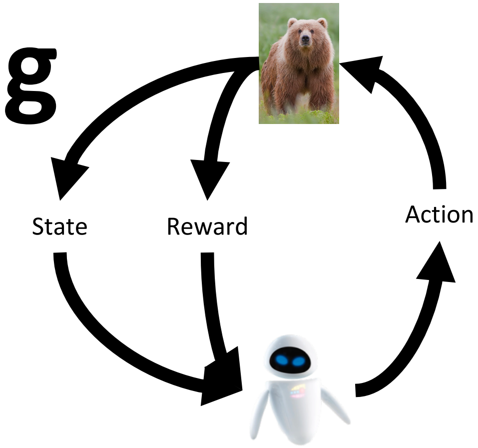
Action-in



Action-out

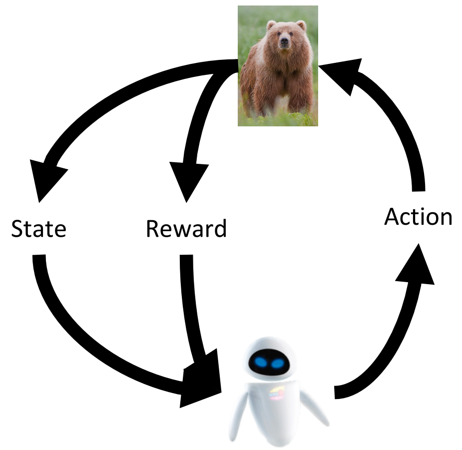
W&M DATA SCIENCE

Recall: Training of Tabular Q-Learning



- For each training episode:
  - Start in the initial state:
  - Loop until a terminal state is reached:
    - Choose an action based on policy π derived from $Q(s, a)$
    - Take the action and observe next state and reward
    - Update $Q(s, a)$ for the current state-action pair
    - Move to the next state

Training Procedure of DQN



- For each training episode:
  - Start in the initial state:
  - Loop until a terminal state is reached:
    - Choose an action based on policy π derived from $Q(s, a, \mathbf{w})$
    - Take the action and observe next state and reward
    - Update $Q(s, a, \mathbf{w})$ for the current state-action pair
    - Move to the next state

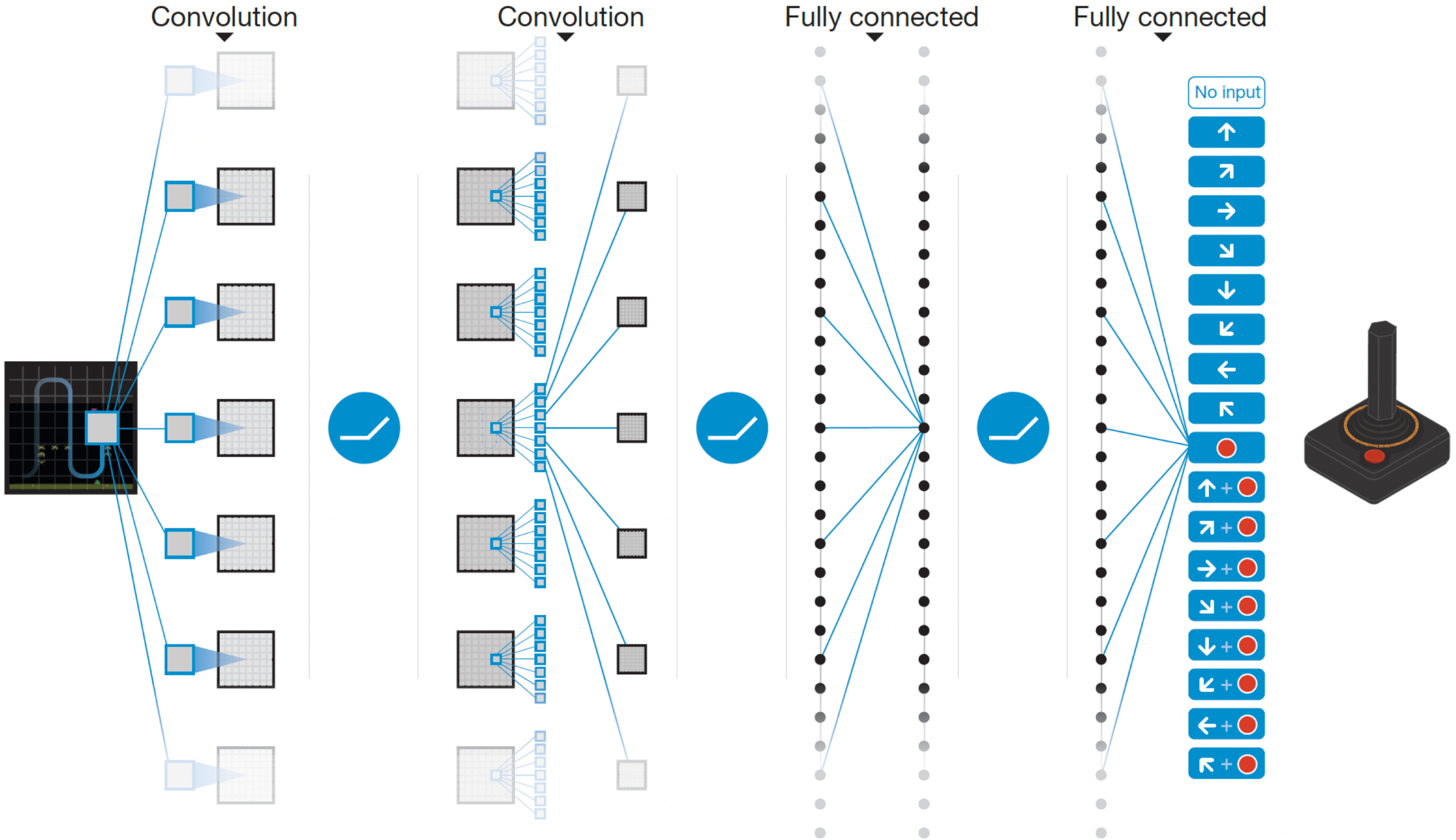
DQN vs Tabular Q-Learning

1. Q-function approximation w DNNs
   - Optimize MSE between prediction and target using SGD

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s',a';w_i) - Q(s,a;w_i)\right)^2\right]$$

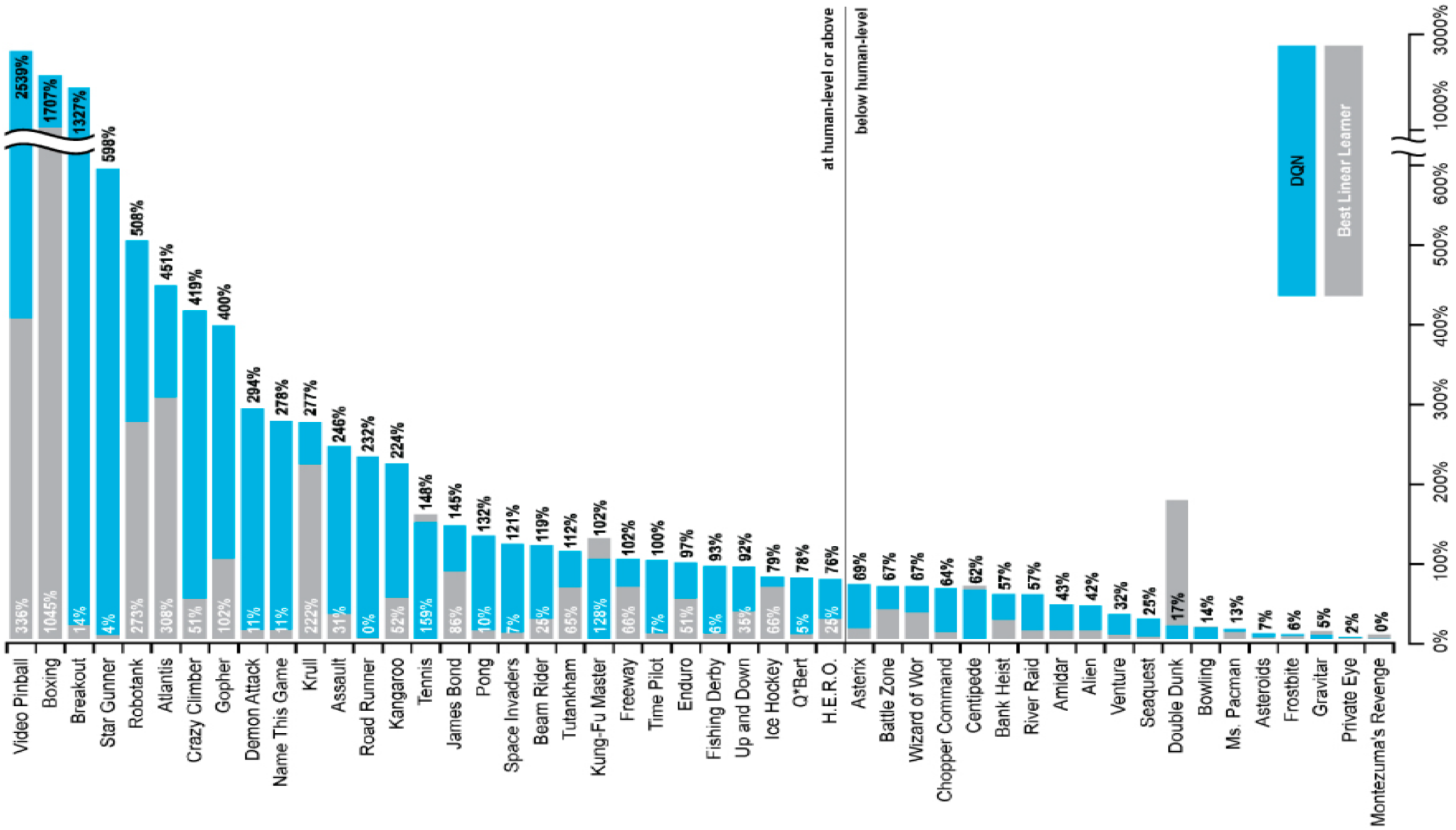
2. Experience replay
   - Store transition (s,a,r,s') in replay memory $\mathcal{D}$
   - Repeatedly sample random mini-batch of transitions from $\mathcal{D}$
3. Fixed Q-targets
   - Compute Q-learning targets w.r.t. old, fixed parameters $\boldsymbol{w}^{-}$

DQN for Atari Games



Mnih et al., Human-level control through deep reinforcement learning, Nature, 2015

DQN Results in Atari Games



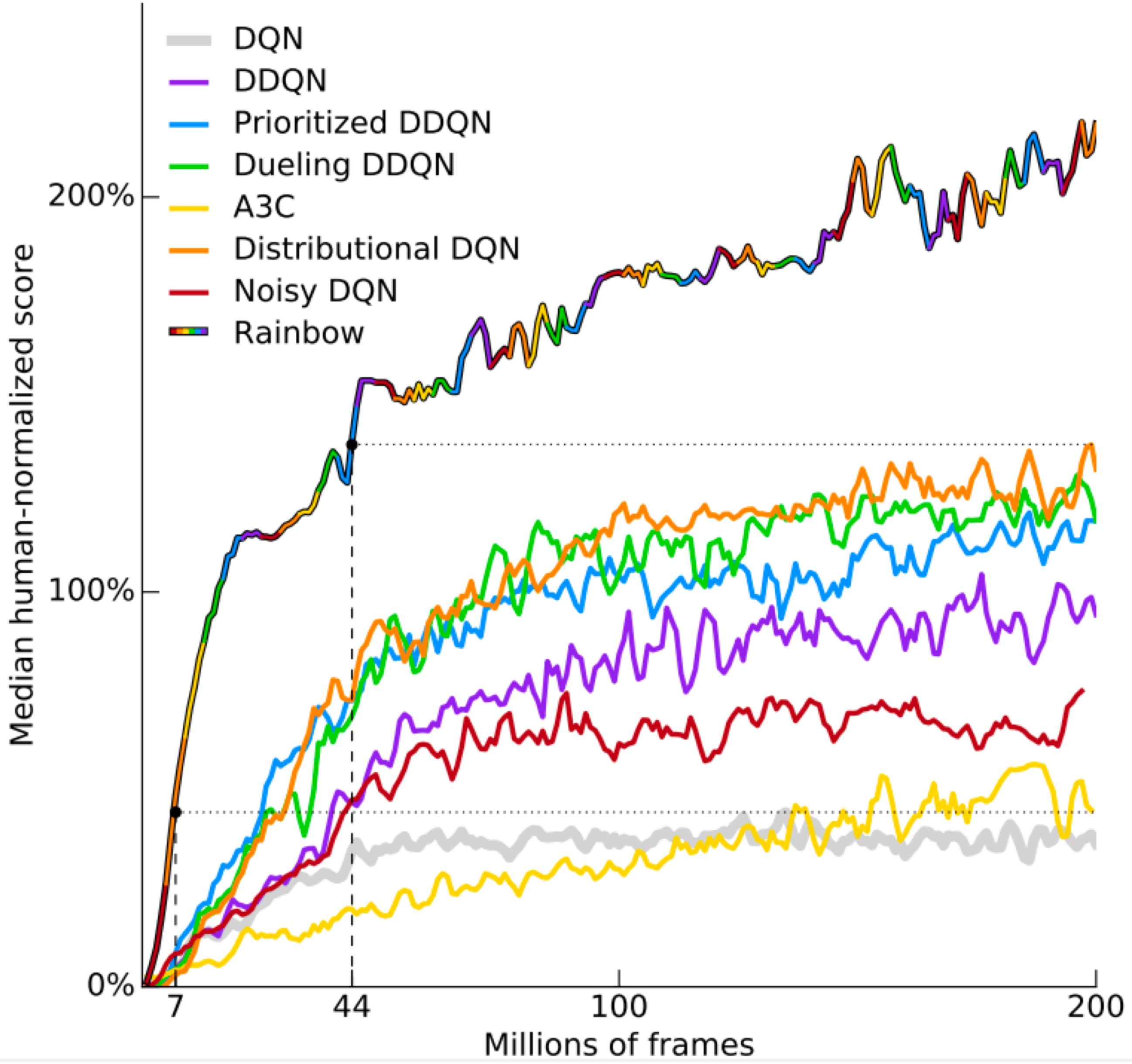
W&M DATA SCIENCE

Variants of DQN

- Success in Atari has led to huge excitement in DQN
- Some immediate and most successful improvements
  - **Double DQN** (Van Hasselt et al, AAAI 2016)
  - **Prioritized Experience Replay** (Schaul et al, ICLR 2016)
  - **Dueling DQN** (Wang et al, ICML 2016)
  - **Raibow** (Hessel, Matteo, et al. AAAI 2018)

Rainbow

DDQN + PER + Dueling DQN
+ Other improvements:
- n-step learning
- Distributional RL
- Noisy Nets



Hessel, Matteo, et al. 2018

Example DQN Code

DQN and Variants

Example

```python
from rlzoo.common.env_wrappers import build_env
from rlzoo.common.utils import call_default_params
from rlzoo.algorithms import DQN

AlgName = 'DQN'
EnvName = 'PongNoFrameskip-v4'
EnvType = 'atari'

# EnvName = 'CartPole-v1'
# EnvType = 'classic_control'  # the name of env needs to match the type of env

env = build_env(EnvName, EnvType)
alg_params, learn_params = call_default_params(env, EnvType, AlgName)
alg = eval(AlgName+'(**alg_params)')
alg.learn(env=env, mode='train', **learn_params)
alg.learn(env=env, mode='test', render=True, **learn_params)
```

Deep Q-Networks

class **rlzoo.algorithms.dqn.dqn.DQN**(*net_list, optimizers_list, double_q, dueling, buffer_size, prioritized_replay, prioritized_alpha, prioritized_beta0*) [source]

[Source: RLZoo – DQN and variants]

DQNs Summary

- Q-learning is a value-based method
  - selects action based on policy derived from Q-function
  - updates/learns Q with Bellman equation
- DQN improves Q-learning with DNNs for function approximation, experience replay, and fixed Q-targets
- DQN is further improved with ideas such as Double DQN, PER, Dueling DQN

Outline

- What is reinforcement learning?
- The reinforcement learning problem
- Value-based
- Policy-based

This Tutorial

Value-based:

Find **optimal** $\boldsymbol{q^*(s, a)}$ as proxy of policy:

$$q^*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

Policy-based:

Directly find **optimal** $\boldsymbol{\pi^*(a|s)}$:

$$\pi^*(a|s) = \operatorname*{argmax}_{\pi} v^{\pi}(s)$$

Downsides of Q-Learning vs PG

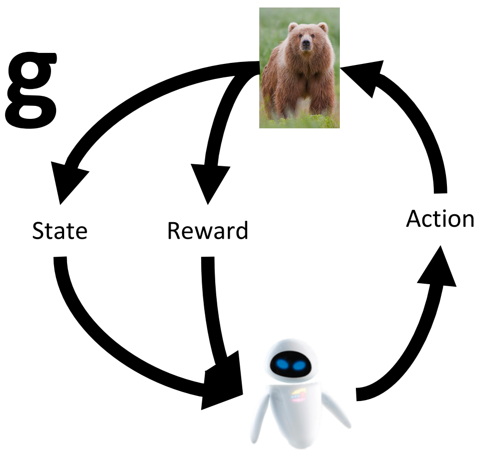
- Cannot handle high-dimensional or continuous action spaces
- Cannot learn stochastic policies
  - E.g., rock-paper-scissor
- Not directly optimizing the objective, but MSE

RL as Policy Optimization

- Policy based reinforcement learning is an optimization problem
- Find π_θ that maximizes $J(\theta) \equiv v^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[v(s)]$:

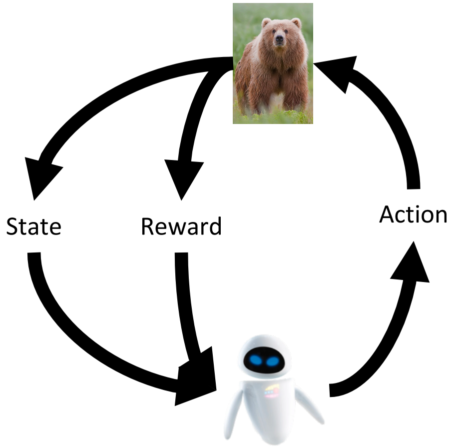
$$\pi_\theta{}^*(a|s) = \underset{\pi}{\operatorname{argmax}}\, J(\theta)$$

Recall: Training of Tabular Q-Learning



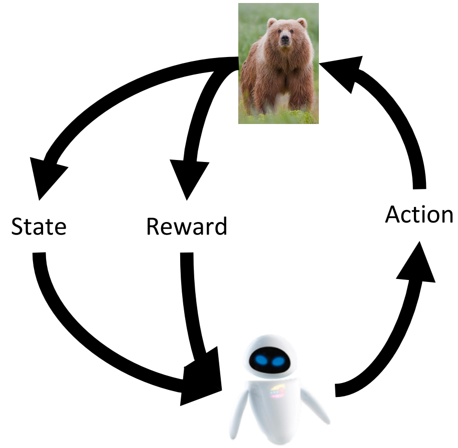
- For each training episode:
  - Start in the initial state:
  - Loop until a terminal state is reached:
    - Choose an action based on policy π derived from $Q(s, a)$
    - Take the action and observe next state and reward
    - Update $Q(s, a)$ for the current state-action pair
    - Move to the next state

Training of Policy Gradient Method



- For each training episode:
  - Start in the initial state:
  - Loop until a terminal state is reached:
    - Choose an action directly based on policy π_θ
    - Take the action and observe next state and reward
    - Update the policy π_θ in the direction that increases expected cumulative reward
    - Move to the next state

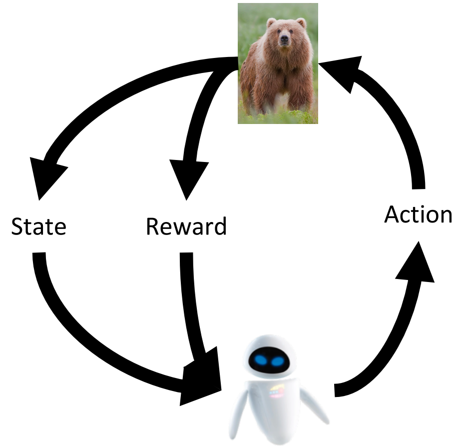
Training of Policy Gradient Method



- For each training episode:
  - Start in the initial state:
  - Loop until a terminal state is reached:
    - Choose an action directly based on policy π_θ
    - Take the action and observe next state and reward
    - Update the policy π_θ in the direction that increases expected cumulative reward
    - Move to the next state

W&M DATA SCIENCE

Training of Policy Gradient Method



- For each training episode:
  - Start in the initial state:

Policy Gradient Theorem (roughly)

For any differentiable policy $\pi_\theta(s,a)$, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\; Q^{\pi_\theta}(s,a)\;]$$

    - Move to the next state

W&M DATA SCIENCE

Practical Policy Gradient Algorithms

The policy gradient has many forms – based on how $Q^{\pi_\theta}(s,a)$ is estimated

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ Q^{\pi_\theta}(s,a)\ ]$$ Explicit PG

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ G_t]$$ REINFORCE

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\ ]$$ Q Actor-Critic

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\ ]$$ Advantage Actor-Critic (A2C)

$$A^w(s,a) = Q^w(s,a) - V^w(s)$$

Downsides of PG Methods

- Hard to choose step sizes
  - Input data is nonstationary due to changing policy
  - Bad step is more damaging than in supervised learning
- Sample efficiency
  - Only one gradient step per environment sample

Trust Region Policy Optimization (TRPO)

- Policy optimization as a constrained optimization

$$\max_{\theta} \quad \widehat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right]$$

$$\text{s.t.} \quad \widehat{\mathbb{E}}_t\left[KL\left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right] \le \delta$$

- Alternatively, constraint as a penalty

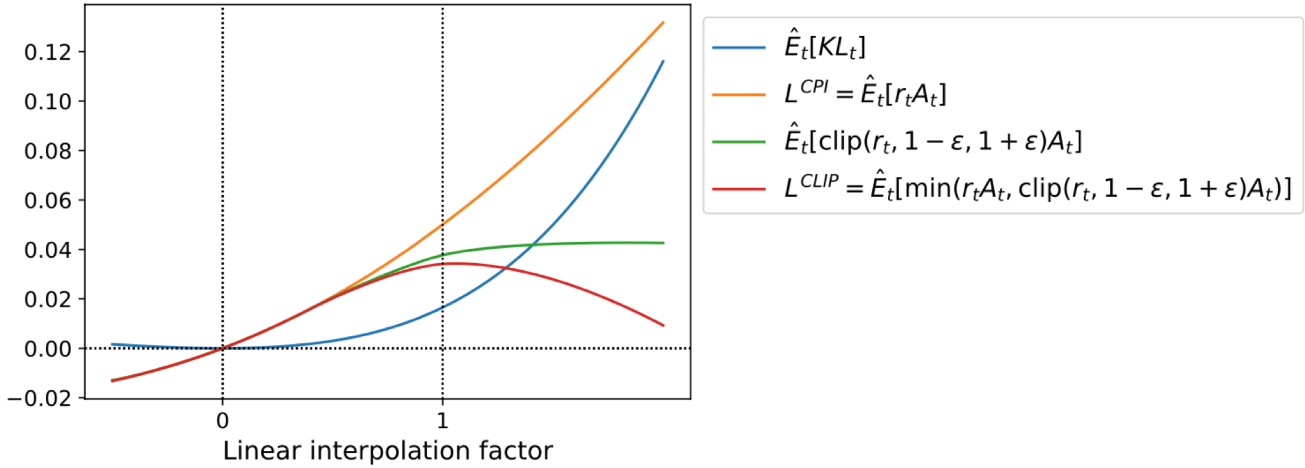
$$\max_{\theta} \quad \widehat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] - \beta\widehat{\mathbb{E}}_t\left[KL\left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right]$$

Schulman et al., 2015

Proximal Policy Optimization (PPO)

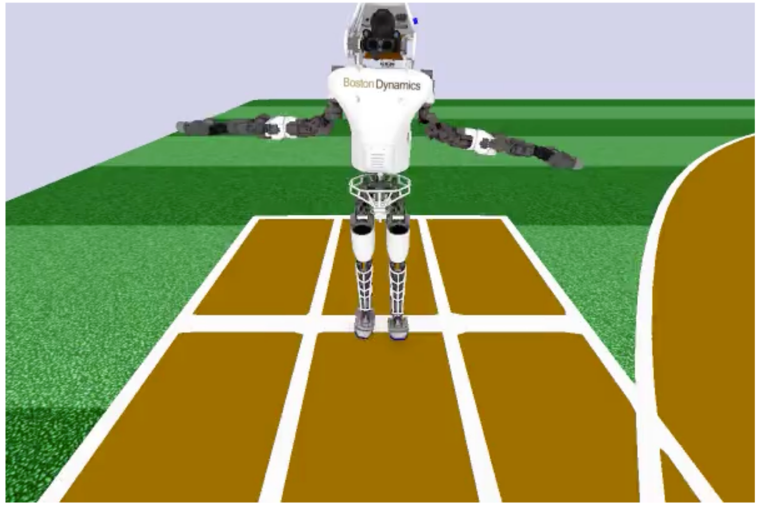
- PPO forms a lower bound objective by clipped importance scores:
- Let $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$

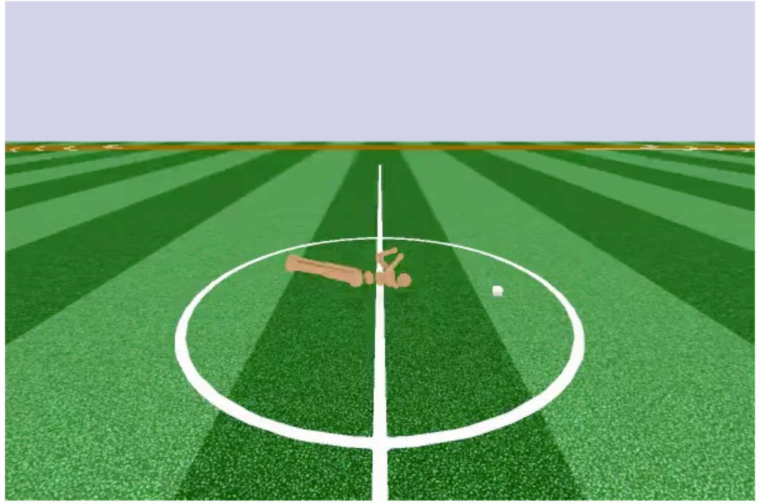
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$



Schulman et al. 2017

PPO on Roboschool





Source: https://openai.com/research/openai-baselines-ppo

Example PPO code

```
from rlzoo.common.env_wrappers import build_env
from rlzoo.common.utils import call_default_params
from rlzoo.algorithms import PPO

EnvName = 'PongNoFrameskip-v4'
EnvType = 'atari'

# EnvName = 'Pendulum-v0'
# EnvType = 'classic_control'

# EnvName = 'BipedalWalker-v2'
# EnvType = 'box2d'

# EnvName = 'Ant-v2'
# EnvType = 'mujoco'

# EnvName = 'FetchPush-v1'
# EnvType = 'robotics'

# EnvName = 'FishSwim-v0'
# EnvType = 'dm_control'

# EnvName = 'ReachTarget'
# EnvType = 'rlbench'

env = build_env(EnvName, EnvType)
alg_params, learn_params = call_default_params(env, EnvType, 'PPO')
alg = PPO(method='clip', **alg_params) # specify 'clip' or 'penalty' method for PPO
alg.learn(env=env,  mode='train', render=False, **learn_params)
alg.learn(env=env,  mode='test', render=False, **learn_params)
```

Source: [RLZoo - PPO]

Practical Tips of Implementing PPO

RL Library	GitHub Stars	Benchmark Source	Breakout	Pong	BeamRider	Hopper	Walker2d	HalfCheetah
Baselines `pposgd` / `ppo1` (da99706)	stars 15k	paper ($)	274.8	20.7	1590	~2250	~3000	~1750
Baselines `ppo2` (7bfbcf1 and ea68f3b)		docs (*)	114.26	13.68	1299.25	2316.16	3424.95	1668.58
Baselines `ppo2` (ea25b9e)		this blog post (*)	409.265 ± 30.98	20.59 ± 0.40	2627.96 ± 625.751	2448.73 ± 596.13	3142.24 ± 982.25	2148.77 ± 1166.023
Stable-Baselines3	stars 6.8k	docs (0) (^)	398.03 ± 33.28	20.98 ± 0.10	3397.00 ± 1662.36	2410.43 ± 10.02	3478.79 ± 821.70	5819.09 ± 663.53
CleanRL	stars 3.6k	docs (1) (*)	~402	~20.39	~2131	~2685	~3753	~1683
Tianshou	stars 6.9k	paper, docs (5) (^)	~400	~20	-	7337.4 ± 1508.2	3127.7 ± 413.0	4895.6 ± 704.3
Ray/RLlib	stars 29k	repo (2) (*)	201	-	4480	-	-	9664
SpinningUp	stars 9k	docs (3) (^)	-	-	-	~2500	~2500	~3000

[Source: The 37 Implementation Details of Proximal Policy Optimization]

Policy Gradient Summary

- Policy gradient methods get actions directly from learned policies
  - Updates policy based on policy gradient (REINFORCE)
- A2C/A3C improves vanilla PG by the advantage function
- TRPO/PPO improves vanilla PG by constraining/penalizing large policy updates

Reinforcement Learning Summary

- The reinforcement learning problem
  - MDP
  - Elements of RL agent: policy, value, ~~model~~
- Value-based
  - Q-Learning
  - DQN, DDQN, PER, Dueling DQN, Rainbow
- Policy-based
  - REINFORCE
  - A2C/A3C
  - TRPO, PPO

Useful coding resources

- RLZoo (Tensorflow)
- Stable-baselines3
- OpenAI Spinning Up
- OpenAI Gymnasium

Thank you!
Email request for slides to hchen23@wm.edu

Haipeng Chen

Assistant professor, Data Science

hchen23@wm.edu

WILLIAM & MARY | DATA SCIENCE