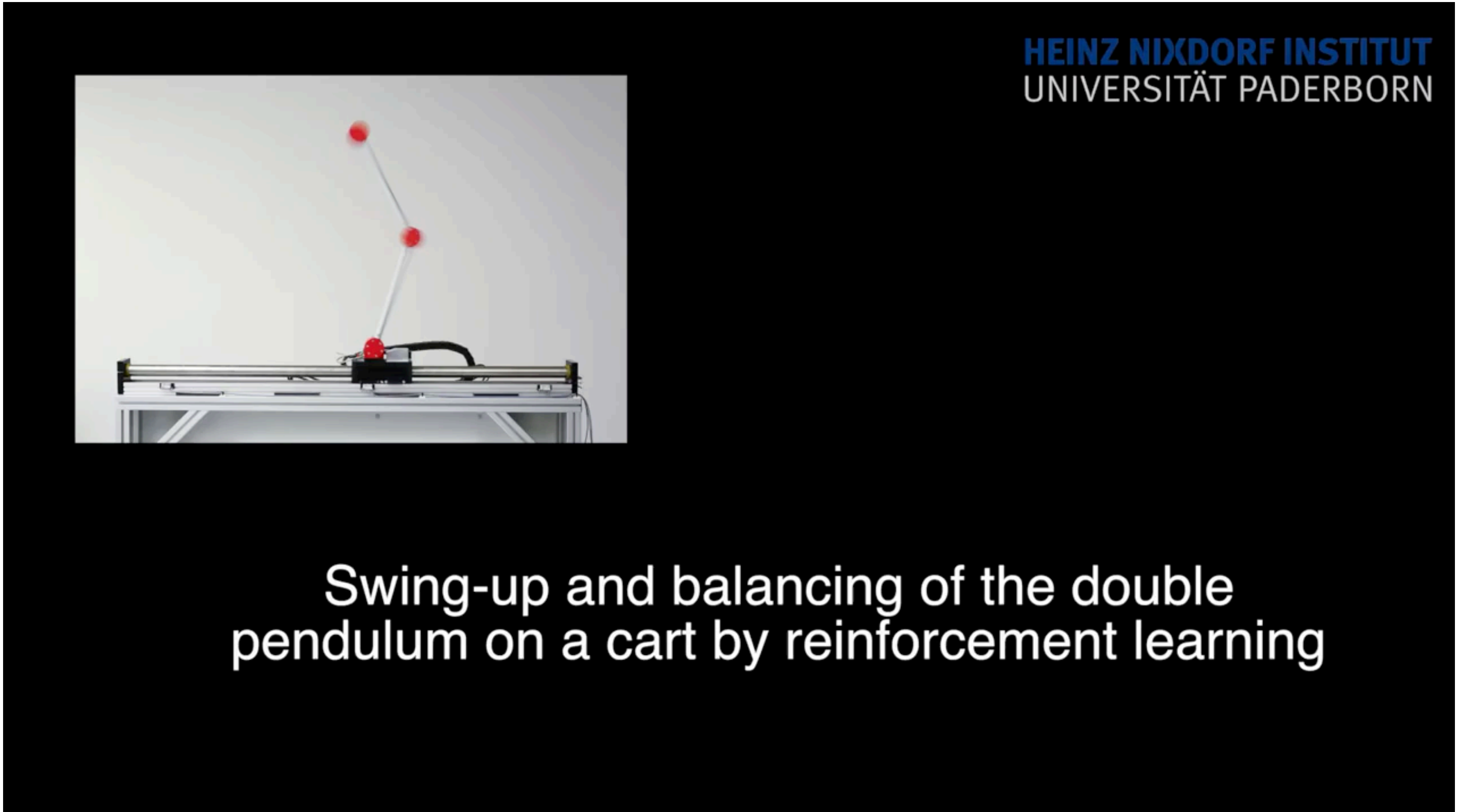
Master
Reinforcement Learning 2022
Lecture 2:
Tabular Value Based Methods

Aske Plaat

Motivation


HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN
Swing-up and balancing of the double pendulum on a cart by reinforcement learning

Overview

- Background: Biology, Psychology
- Agent/Environment
- MDP
- Bellman, Temporal Difference, Bandit/Exploration, On/Off-Policy
- Value Iteration, SARSA, Q-learning
- Gym

Deep Reinforcement Learning

=

Deep Learning

+

Reinforcement Learning

Deep Reinforcement Learning

- Modeling of Interaction, Behavior, Action
- Database-free learning
- Power of Deep Learning for High-dimensional inputs: vision and Generalization
- In Human terms: Eye - Hand Coordination

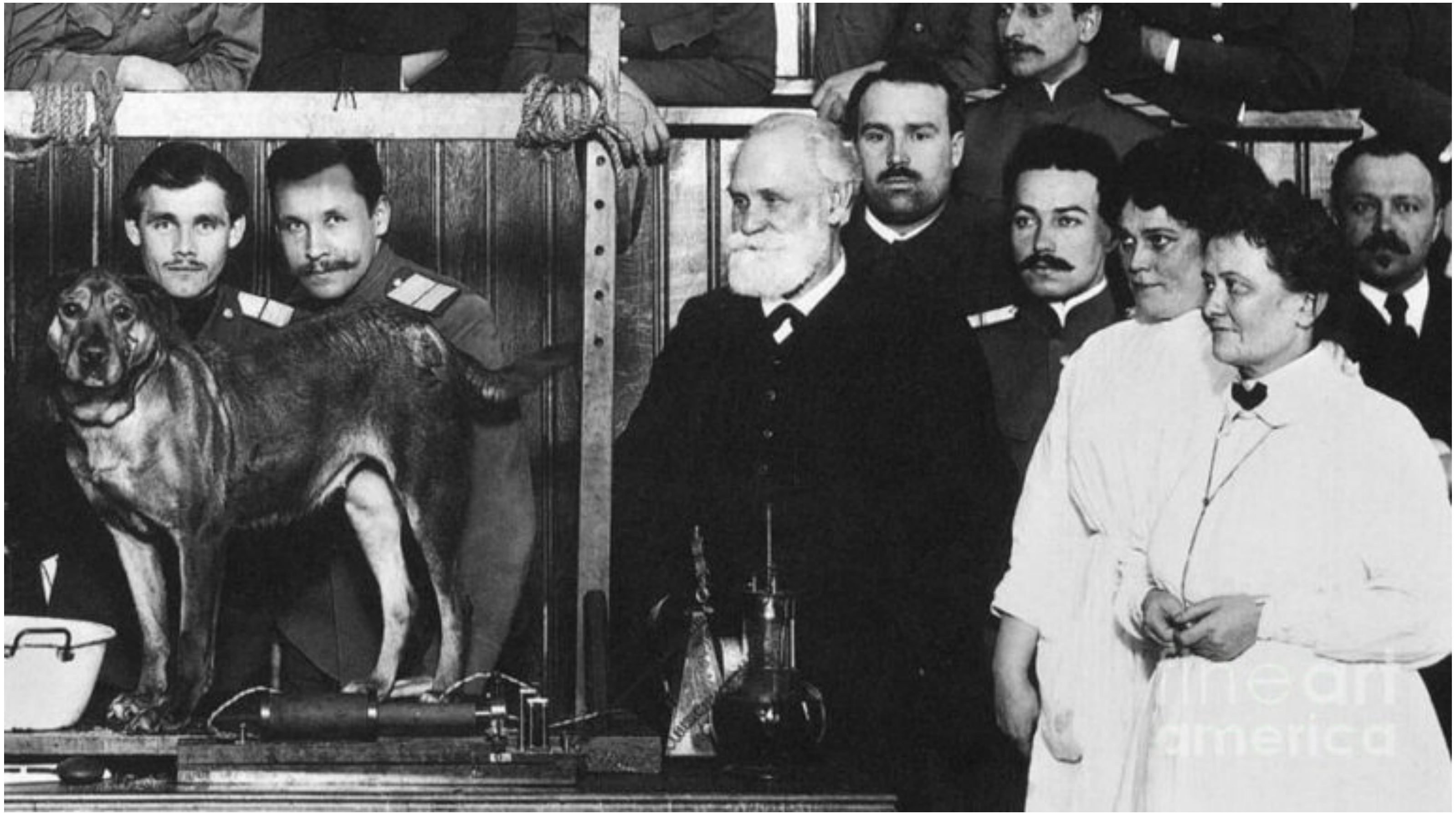
Deep **RL**

Biological Roots

RL Intuition

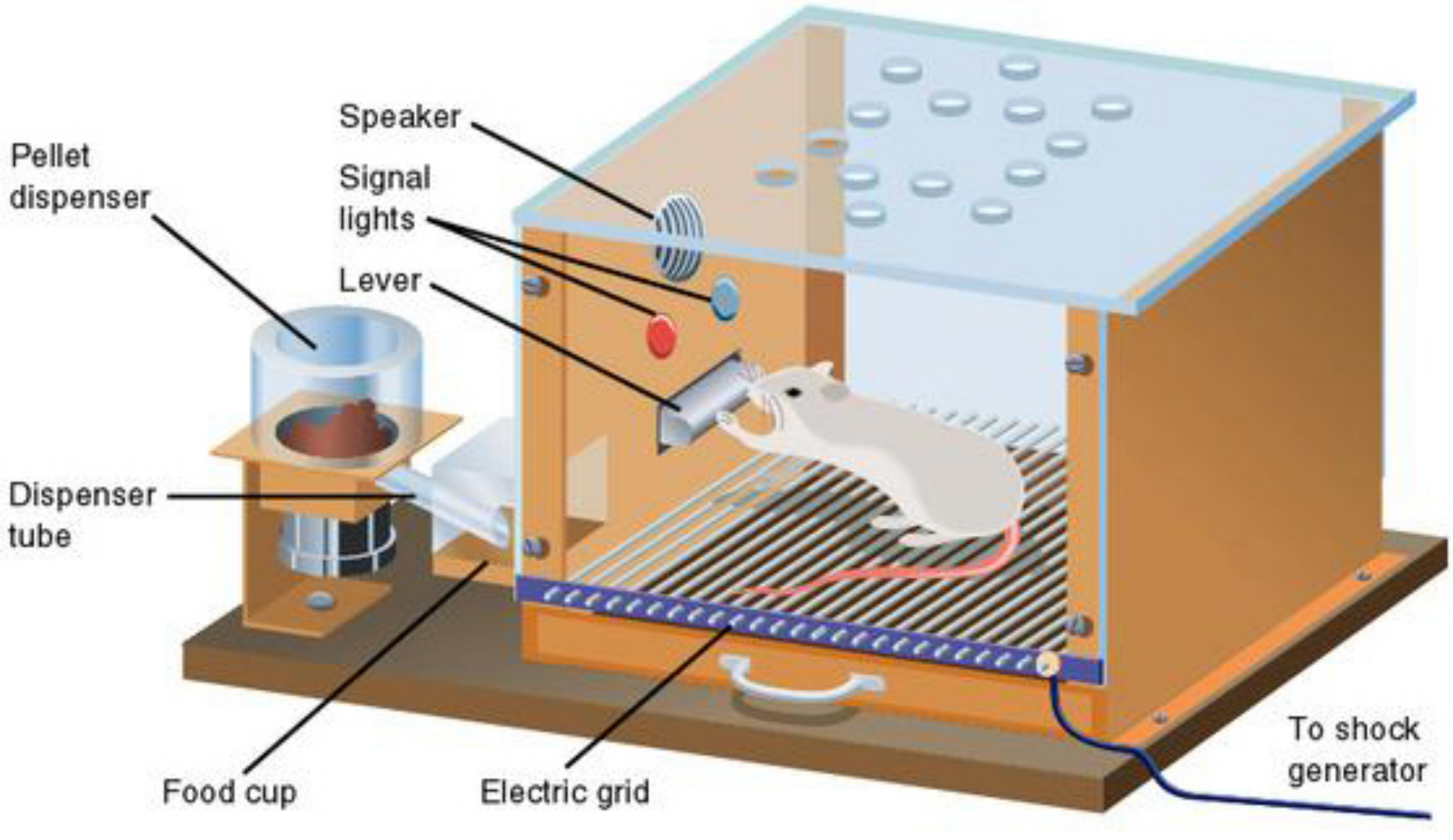
- Learning by conditioning
- Learning by trial and error
  - trial: (state,action)
  - error: (reward)
- Learn by probing

Pavlov

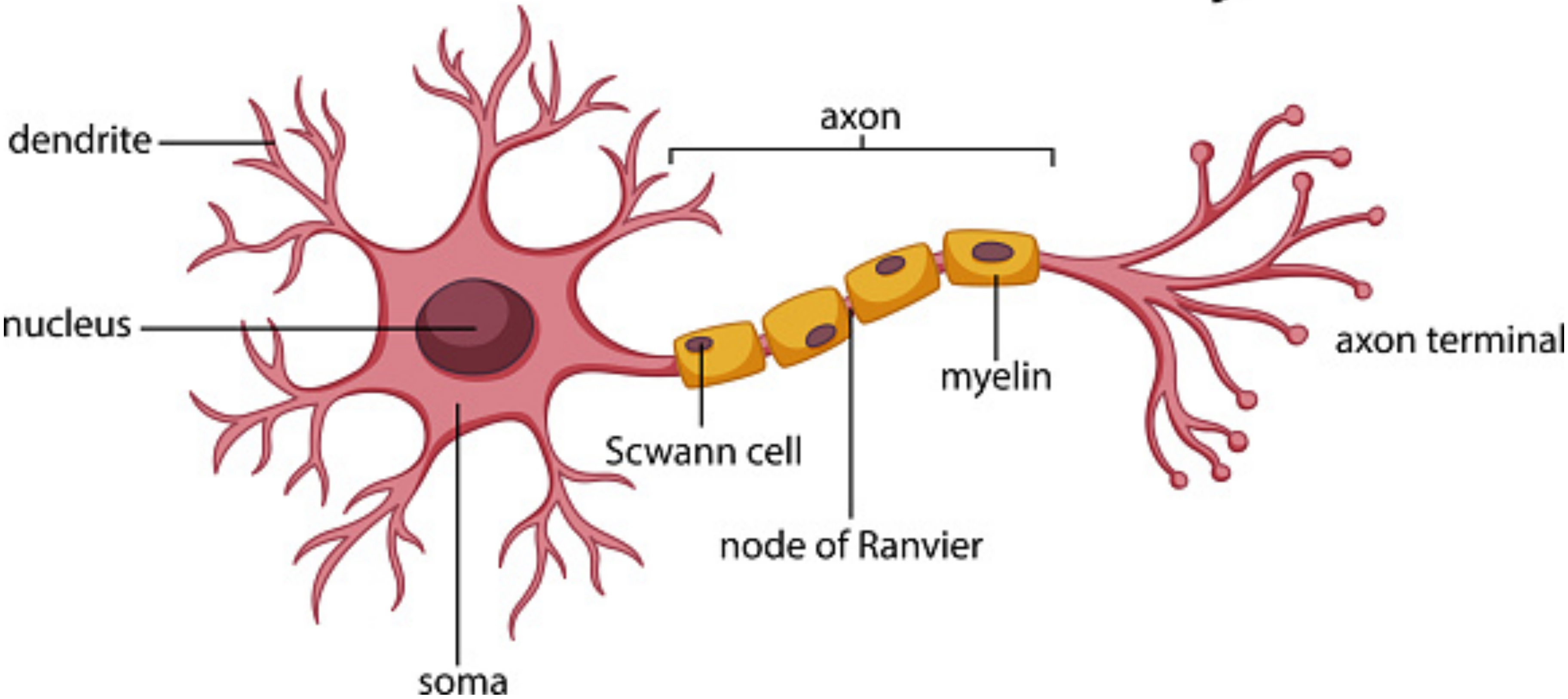


B.F. Skinner

"The Father of Operant Conditioning"



Neuron Anatomy



Mathematical Model

Mathematics

- Markov Decision Process
- Optimization Processes

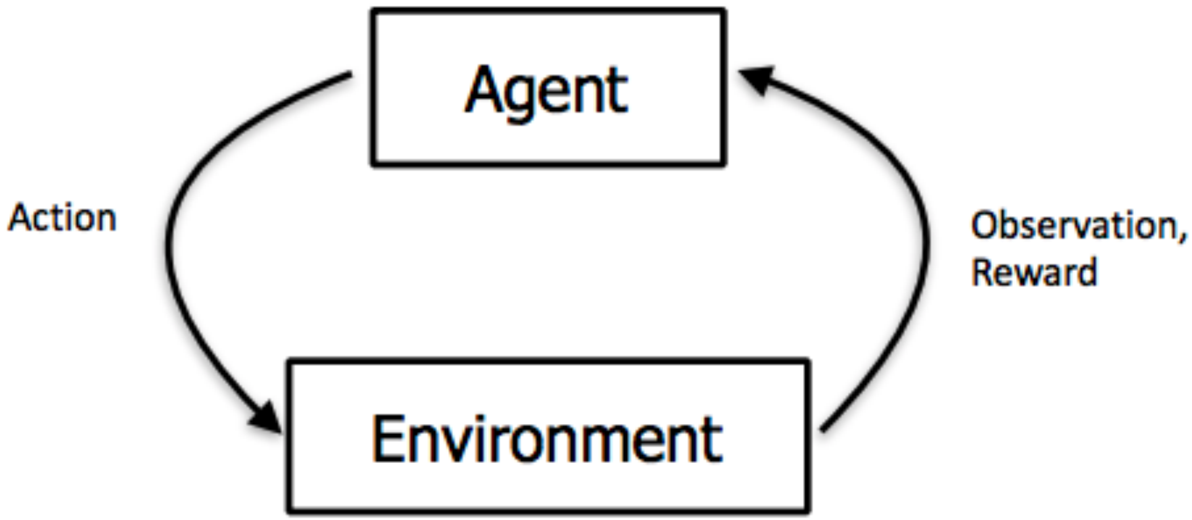
Sequential Decision Problems

- Animal conditioning is a single step problem
- RL is typically used for sequential decision problems
- RL is typically modeled as a Markov decision process

Sequential Decision Problems

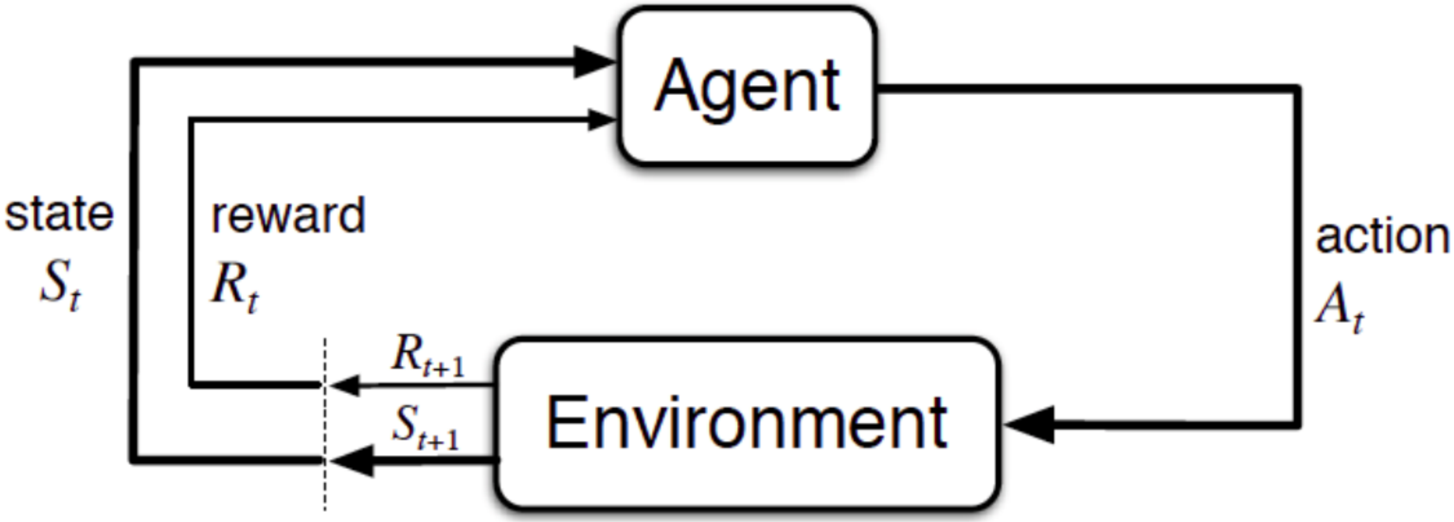
Reinforcement Learning

- Is learning by interaction with an environment with a single reward



Reinforcement Learning

- is learning by interaction
- (state, action) -> reward value



The agent-environment interaction in reinforcement learning. (Source: Sutton and Barto, 2017)

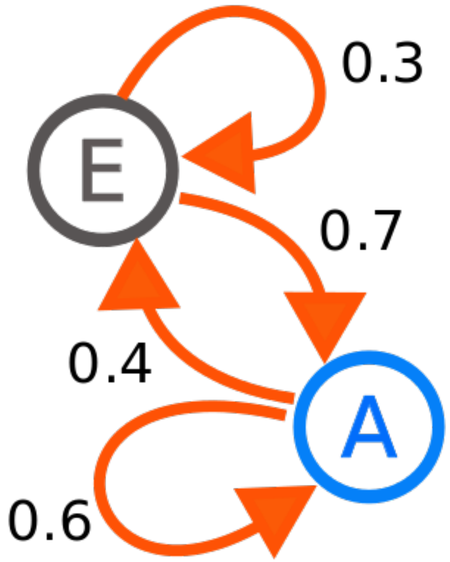
SL - RL

- database
- ((example, label), correct?)
- unordered batch examples
- categories
- classification/regression
- memoization/deep learning

- probing
- ((state,action), reward)
- sequence of examples
- behavior
- action in state (“policy”)
- memoization/deep learning

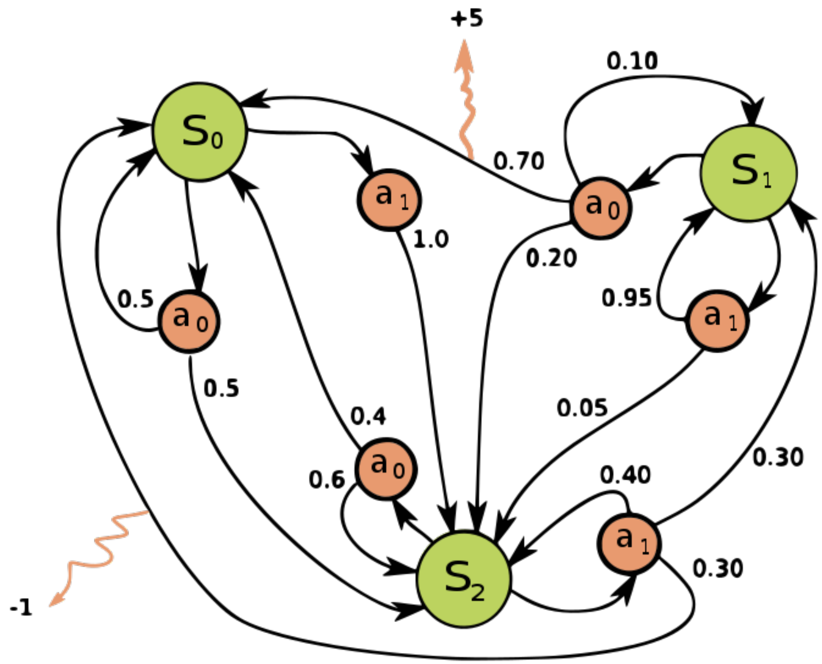
Markov Decision Process

- Andrey Markov 1856-1922
- Formalism for reinforcement learning
- Markov property: “No Memory”
  Future state is solely determined by current state + action (previous states do not matter)
- MDP is extension of Markov Chain: actions and rewards



MDP

- S - State
- A - Action
- T - probability of Transitioning
- R- Reward (can be positive and negative)
- γ - Discount factor

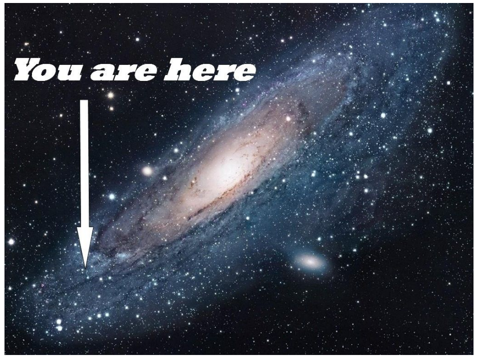


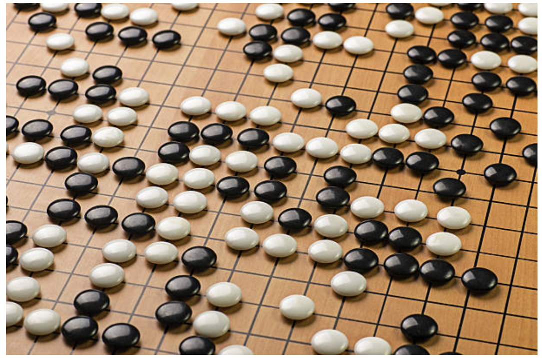
Goal of RL

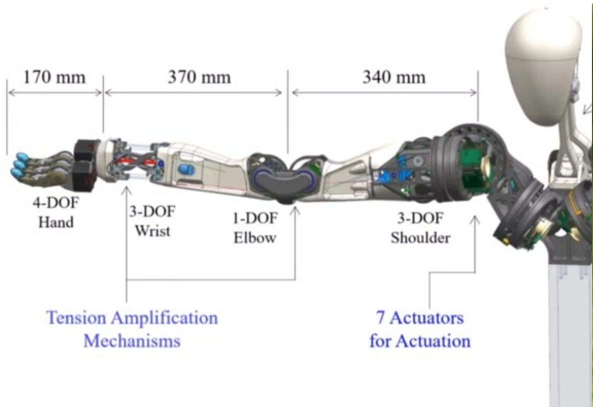
- What action to take in a state?
- Find the optimal policy π^*
  find in each state the actions that maximize the expected cumulative future reward

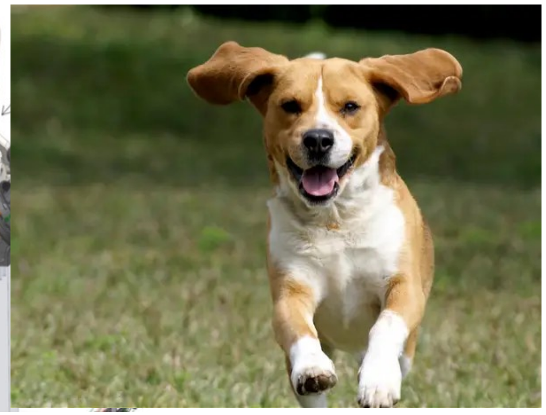
State

- Uniquely represent the state of the environment at time t
  - location on a map
  - pieces on a board
  - angles of joints
  - pixel values in a grid





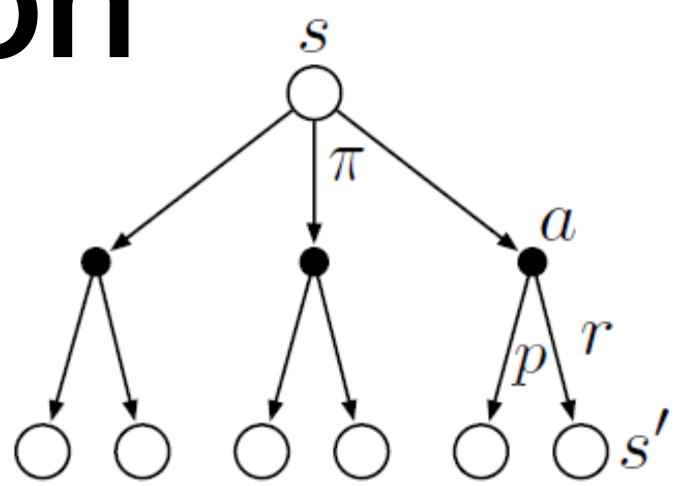




Action

- state s -> action a -> state s’
- discrete action:
  an small integer number
  move pawn e2 to e4
- continuous action:
  bet $1234,56
  move joint A to 56,7 degrees
- discrete policy π(s) -> a
- stochastic policy π(a|s) -> probability distribution over actions

Transition



Backup diagram for v_π

- state s -> action a -> state s'

- $T_a(s,s')$ is the probability that action a in state s will transition to state s' in the environment

- of s->a->s'
  the s->a part is chosen by the agent (policy)
  the a->s' part is chosen by the environment

- T is known by the environment, not by the agent

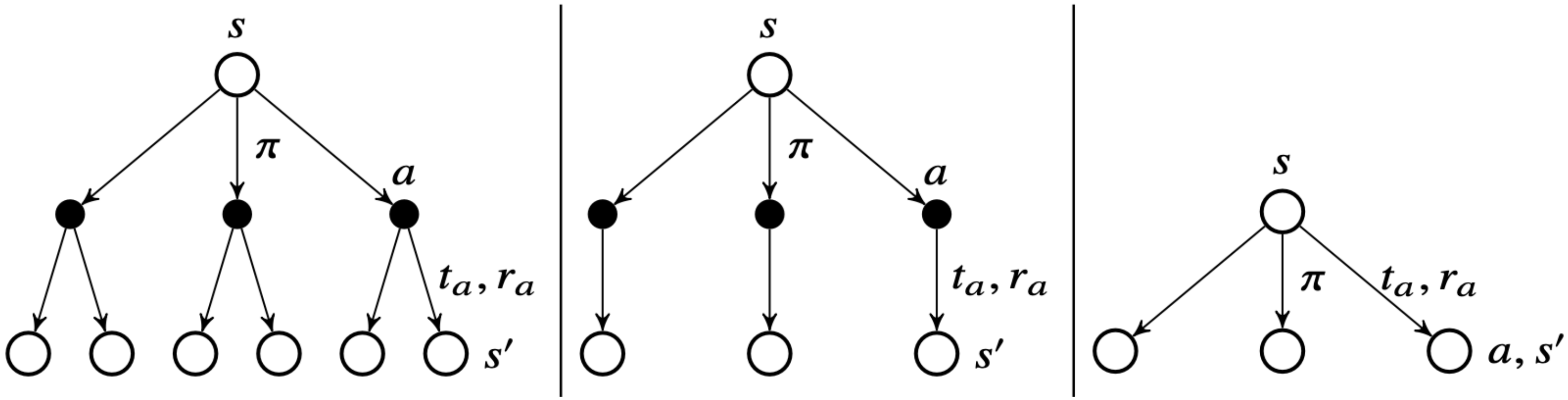
Transition Model

- T is known by Environment only: Model-free methods
  For example: Q-learning

- Agent has local (approximation of) T: Model-based methods
  For example: Dyna

Deterministic Transitions

- In some environments one state follows an action
  For example: Grid World, Puzzles



Trajectory

- Episodic problems have an end
- Continuous problems continue for ever
- Trajectory/Trace/Episode is the sequence of state/action/reward from start to finish

$$\tau_t^n = \{s_t, a_t, r_t, s_{t+1}, .., a_{t+n}, r_{t+n}, s_{t+n+1}\}$$

Reward

- $R_a(s,s')$ is the Reward received after action *a* transitions from state *s* to state *s*'

- $R(\tau)$ is the Return: the cumulative reward of a trace

- $V^{\pi}(s)$ is the state-Value: the expected cumulative reward of a state for following the policy from *s*

- $Q^{\pi}(s,a)$ is the state-action-Value: the expected cumulative reward of a state for following action *a* from state *s* and then the policy from *s*'

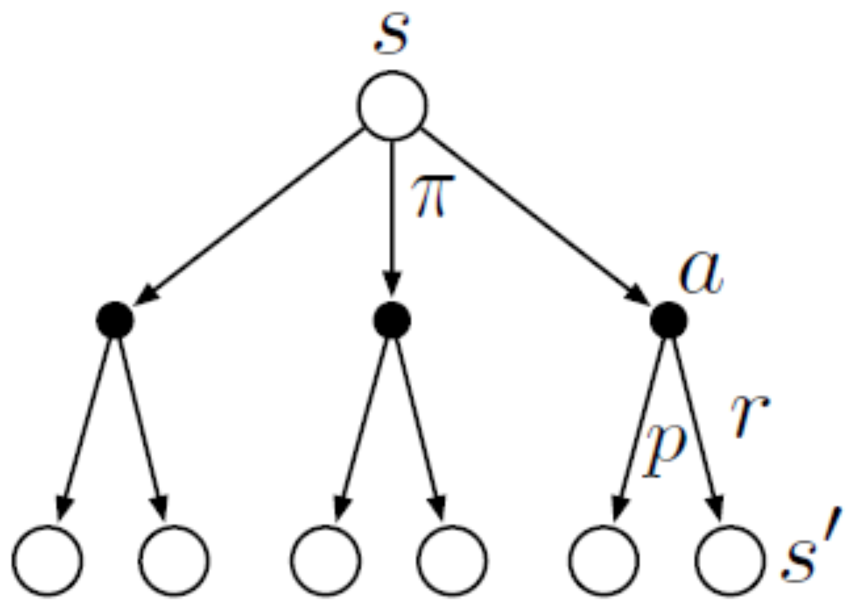
γ

- Gamma is the discount factor, discounting the importance of future rewards
- Especially important in continuous problems
- Sometimes ignored (γ=1) in episodic problems

Solution Methods

Select Down, Learn Up

- Policy is of central importance
- Solution algorithms (finding the optimal policy) travel down and up the tree repeatedly
- It is used to select which action to take in state s
  "**Selecting down**"
- It is also the data structure that is updated when rewards come in
  "**Learning up**"



Backup diagram for v_π

Functions*

- Value V(s)
- Action Value Q(s,a)
- Policy π(s)
- It may help to think of these functions as arrays that can be updated

Bellman

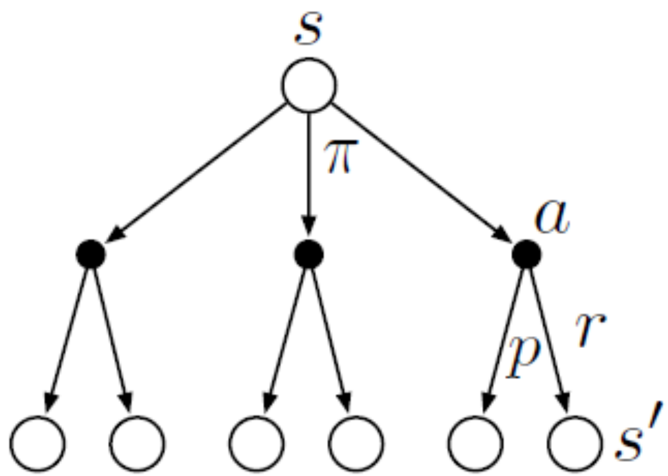
- Bellman equation recursively defines value (assuming transition function P and policy are given)

- Discounted future reward

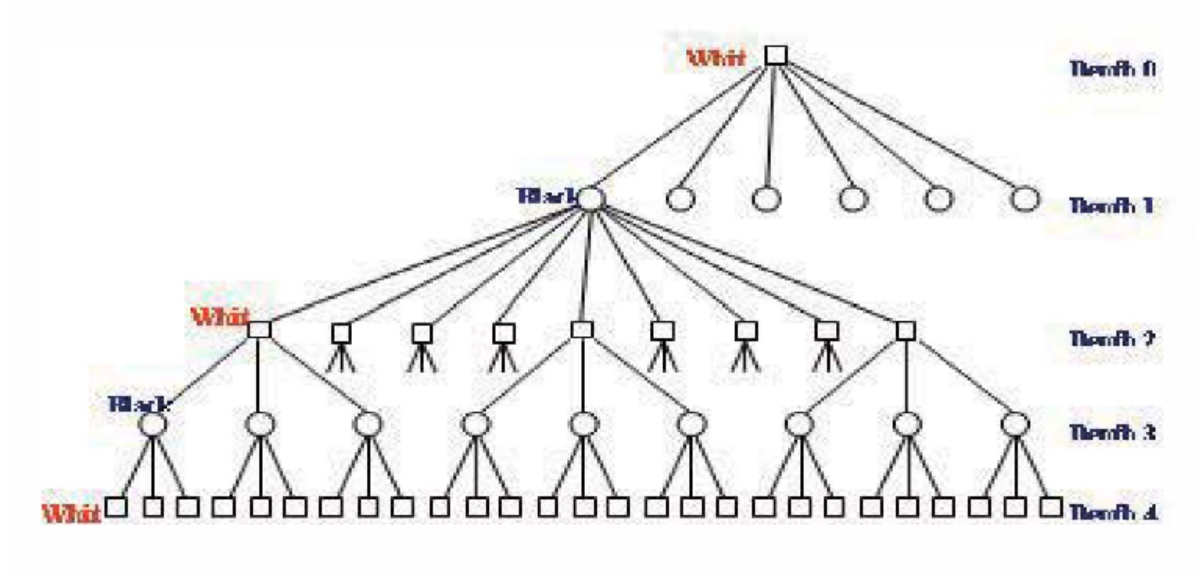
$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s')$$

- Needs Reward, Policy, and Transition P
  It is nice to have this recursive equation, but, unfortunately we typically do not have the transition function

Bellman Backup



Backup diagram for v_π



Value Iteration

Initialize V(s) to arbitrary values

Repeat until V(s) converge

For all states

For all actions

$$Q(s,a) \leftarrow \sum_{s} P_{ss'}^{a}(r(s,a) + \gamma V(s'))$$

$$V(s) \leftarrow \max_{a} Q(s,a)$$

What if we do not have the transition function?

Model-free

- The recursion idea to find the Value is useful
- But what if the agent does not have the Transition function, can it use the Environment to sample from?

Temporal Difference

- Temporal Difference Learning [Sutton]

- Solution method that **samples** from environment, estimating the policy, when **no transition** probabilities are given

$$V(s) \leftarrow V(s) + \alpha[R' + \gamma V(s') - V(s)]$$

- Gamma is discount rate, Alpha is learning rate

Temporal Difference

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

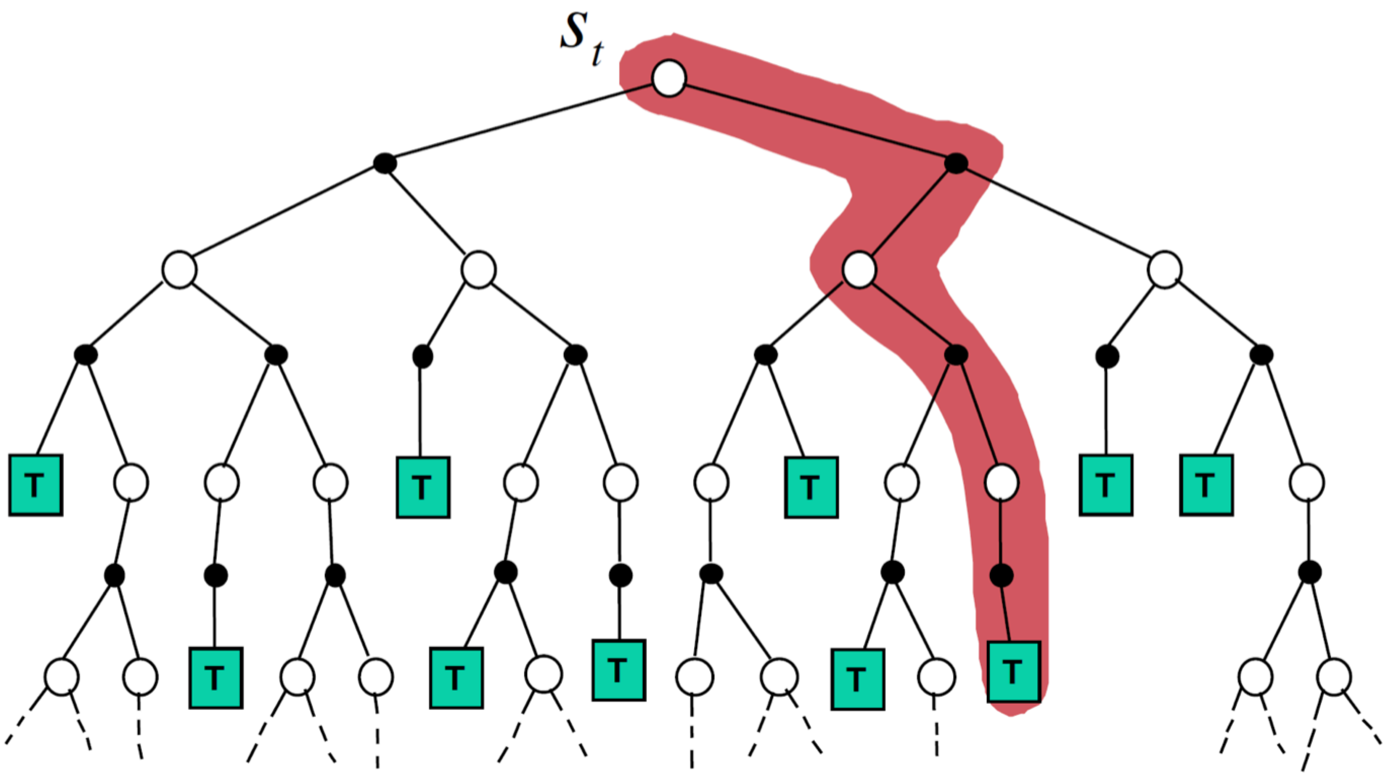
Compare

- Monte Carlo (full-episode)
- Temporal Difference (partial-episode)
- Dynamic Programming (given transition function)

Monte Carlo

Monte-Carlo Backup:

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

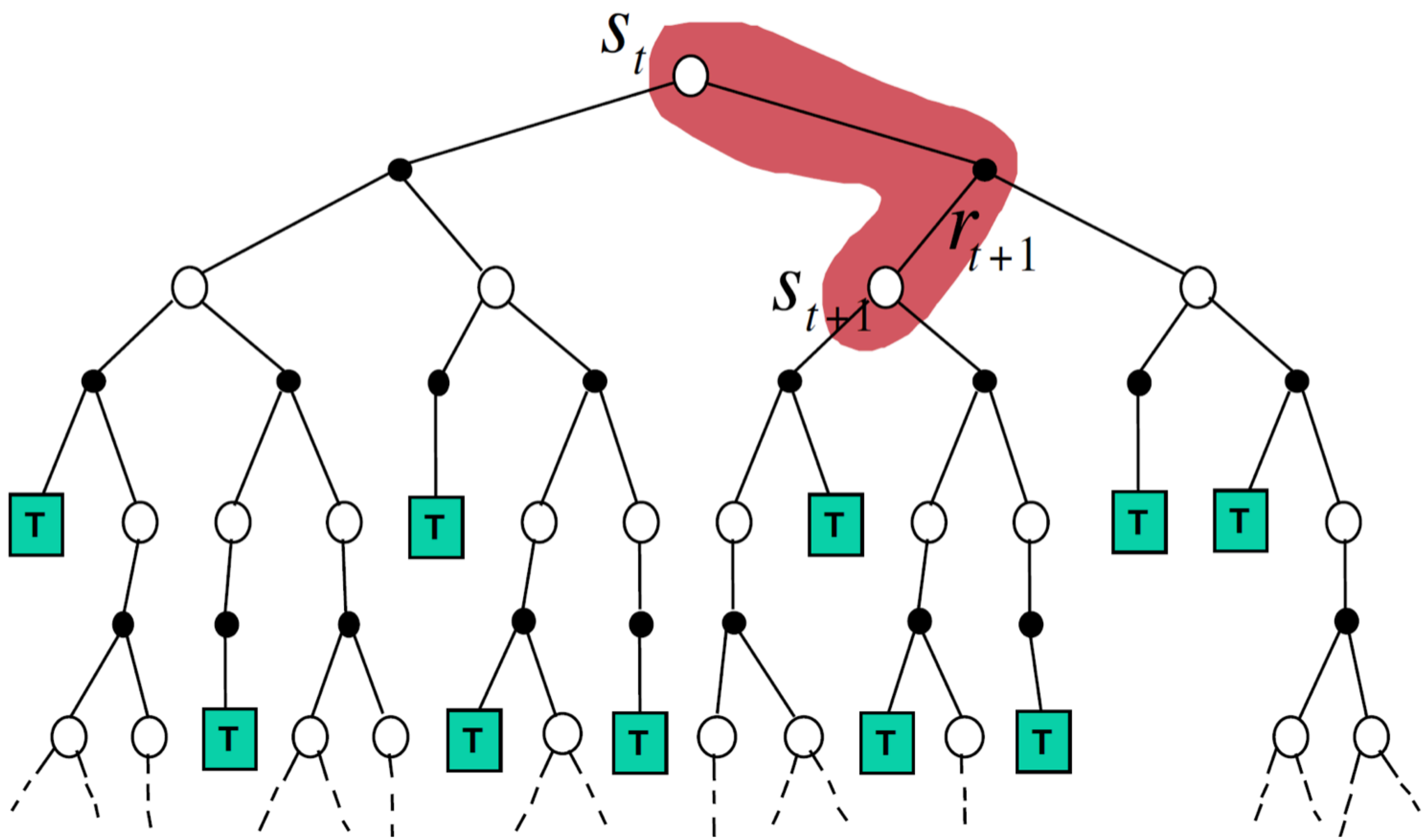


In Monte-Carlo we are basically traversing one random path of states which eventually leads to a terminating state. Hence, it will traverse through the **depth** and end with a terminating state.

Temporal Difference

TD Backup:

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

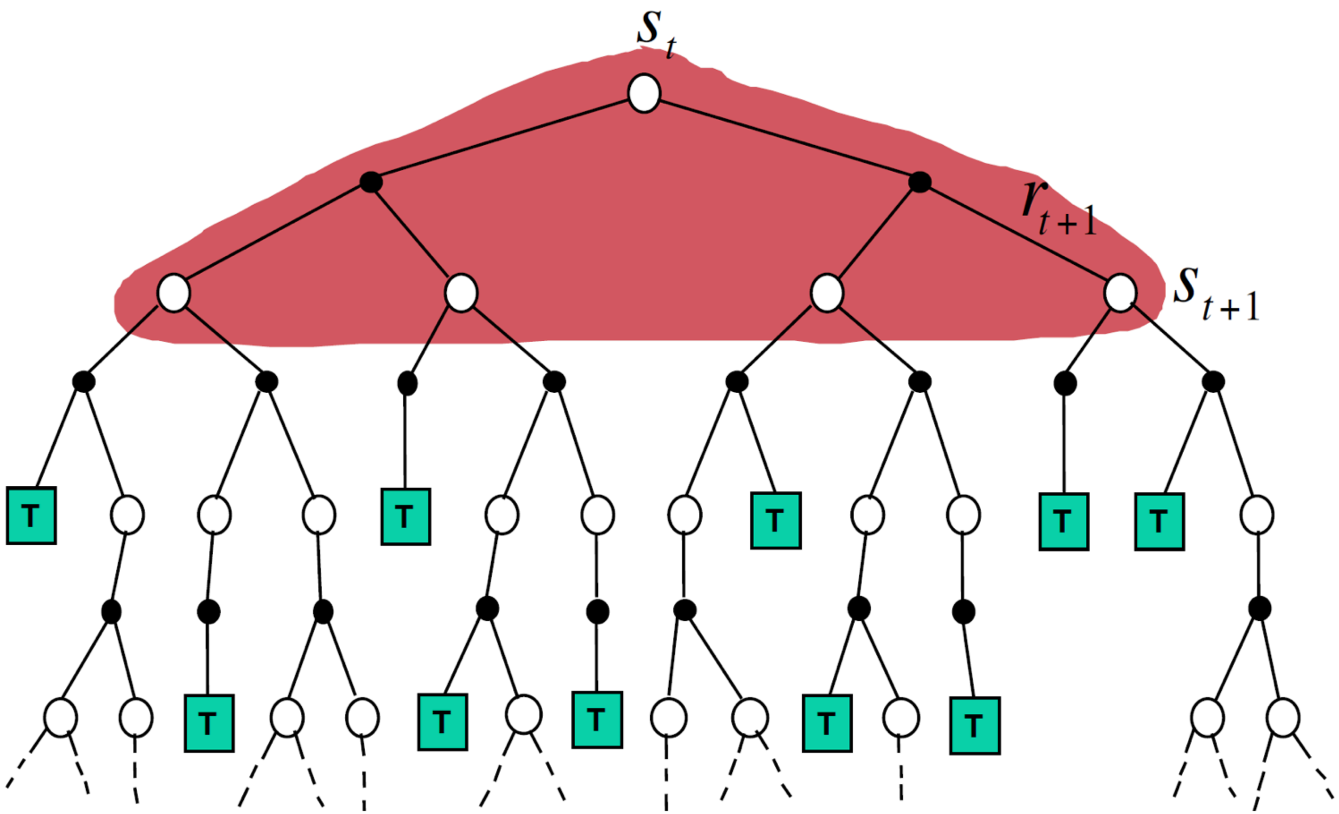


In TD, we only look one step ahead and then estimate the rest. That is R_{t+1} + gamma*(V(S_{t+1}).

Dynamic Programming

Dynamic programming backup:

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1})\right]$$

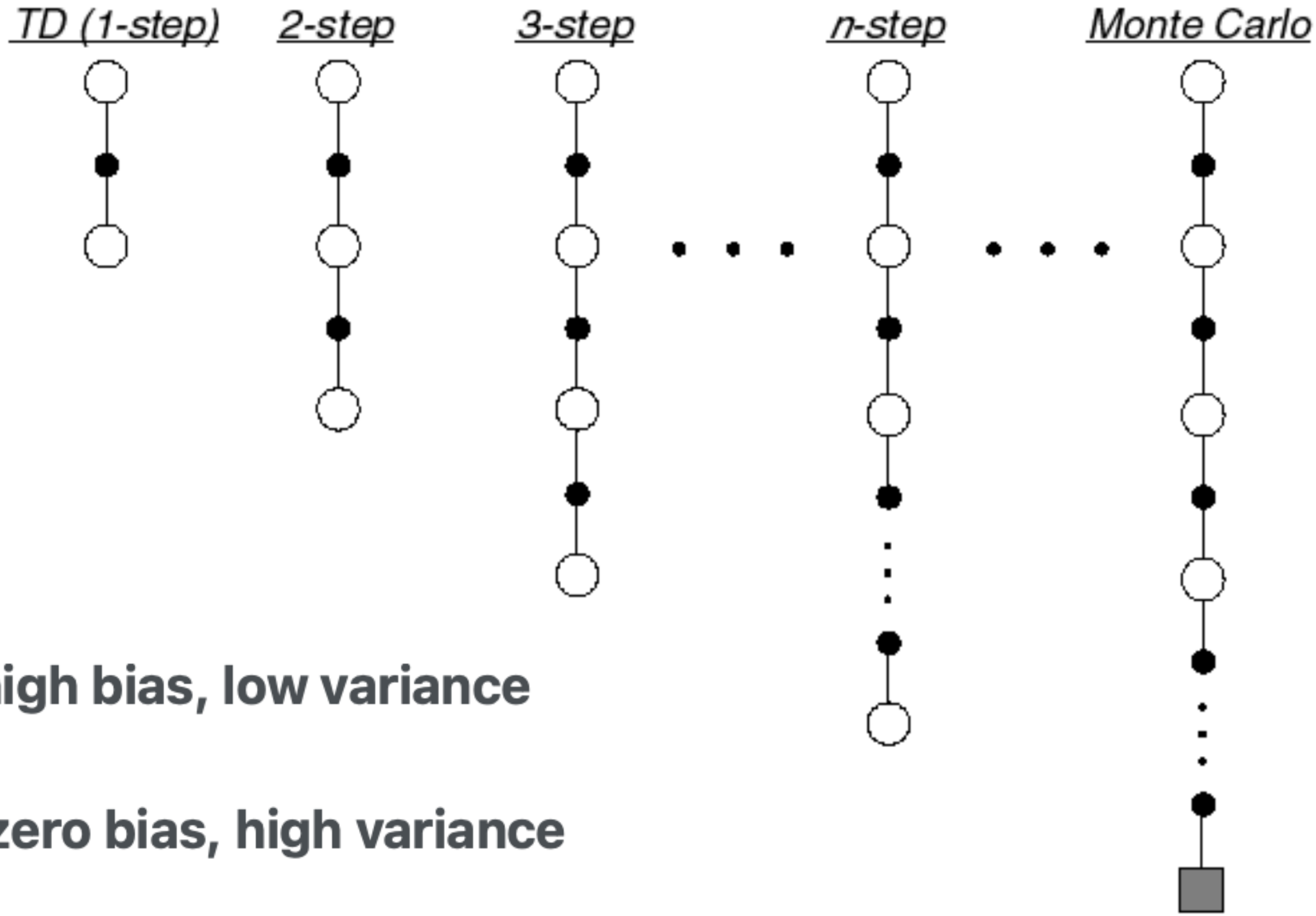


In DP, we used to consider **all possible states** one level ahead, i.e the entire breadth of level+1.

As opposed to this, in MC and TD we are only considering a limited space.

Bias/Variance

Let TD target look n steps into the future



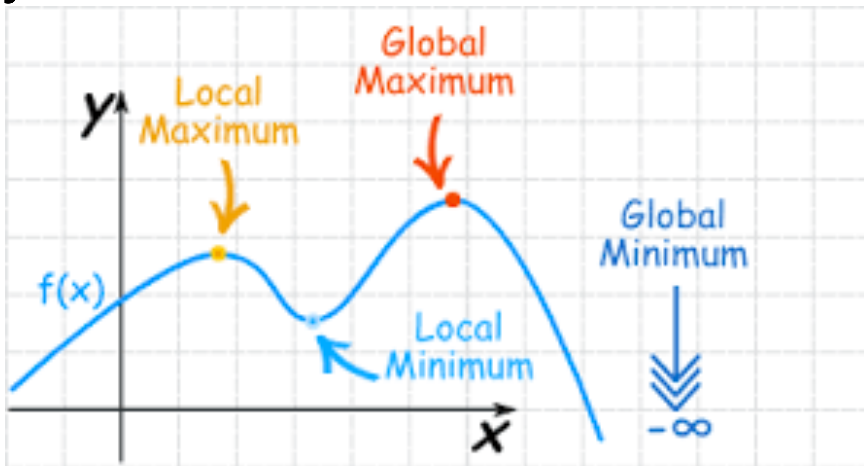
TD – high bias, low variance

MC – zero bias, high variance

Exploration/Exploitation

Exploration/Exploitation

- We now have a recursive formula to compute the value [Learn Up]
- We also have a sampling procedure [Select Down]
- How can we sample in a smart way?
- Exploit the best current action
- Explore to get out of local optima



The Societal Importance of Exploration

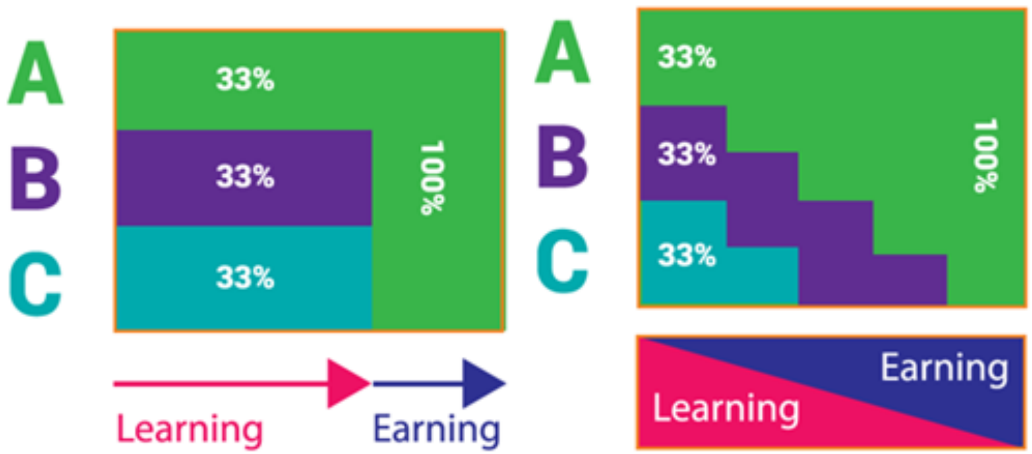
- Sensational news satisfies our immediate desires; thoughtful new directions explores less-direct benefits

- Without sufficient exploration your news will stay inside your filter bubble

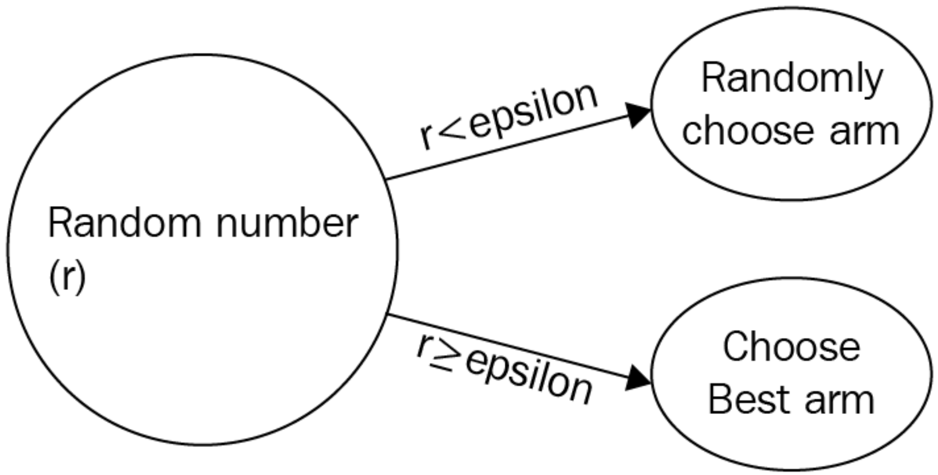
- Without sufficient exploration your democratic processes will get you Trump

Multi-armed Bandit

- Theory for optimal exploration sampling
- Important in Clinical Trials to find minimal regret



Epsilon-Greedy



- Greedy: Exploit best current action
- However, for an epsilon fraction, you explore a random action
- Static, adaptive schemes

Epsilon-Greedy

- When epsilon-greedy explores [Select Down], and finds that the action was indeed non-optimal, Then What [Learn Up]?

- On-policy learning says: use its reward anyway.
[highly consistent, but perhaps slow convergence]

- Off-policy learning says: use the best action instead to learn from.
[may diverge, but may be quicker to converge]

On Policy, Off Policy

- On policy learning samples its behavior from the current (best) policy function as it is updating that current policy function. Even when selection explores non-optimally, it follows that to update policy. As it learns the latest policy, it walks (samples behavior) from this policy -> convergence

- Off-policy learning samples its behavior from a policy but updates from the one with the best rewards. When behavior explores non-optimally, learning exploits; it learns the best policy off the behavior policy -> may not converge, since behavior policy may be not influenced by learning. (But it might be a large database of previous samples, and off-policy is suited for parallelization)

On Policy, Off Policy

- Use Q to select [down] s’ and a’, and then:
- On-behavior-policy learning [up]: SARSA

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

- Off-behavior-policy learning [up]: Q-learning

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

SARSA

- Initialize Q-function
- For All Episodes:
  - Initialize s; Select a ϵ-greedy from Q(s)
  - For All Time Steps in this Episode:
    - Perform a in Environment giving s’ and r
    - Select a’ ϵ-greedy from Q(s) :: **SELECT DOWN**
    - $Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma Q(s',a')-Q(s,a)]$ :: **LEARN UPDATE**
    - $s \leftarrow s'$; $a \leftarrow a'$
- return Q

Q-learning

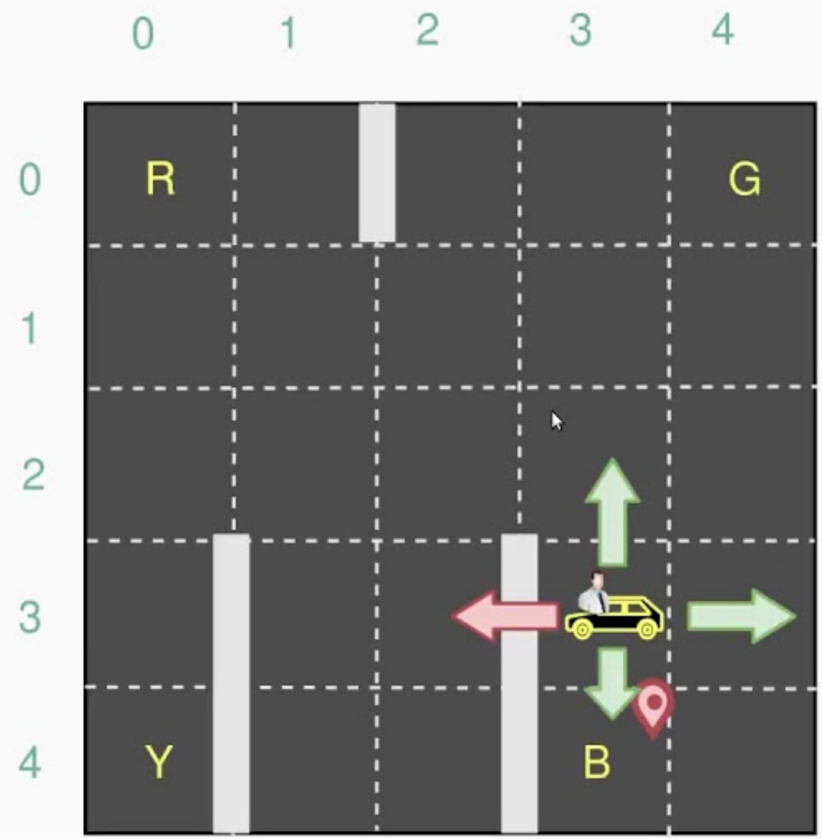
- Initialize Q-function
- For All Episodes:
  - Initialize s
  - For All Time Steps in this Episode:
    - Select a ϵ-greedy from Q(s) :: **SELECT DOWN**
    - Perform a in Environment giving s’ and r
    - $Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma \max_a Q(s',{}_a)-Q(s,a)]$ :: **LEARN UPDATE**
    - $s \leftarrow s'$
- return Q

Practice

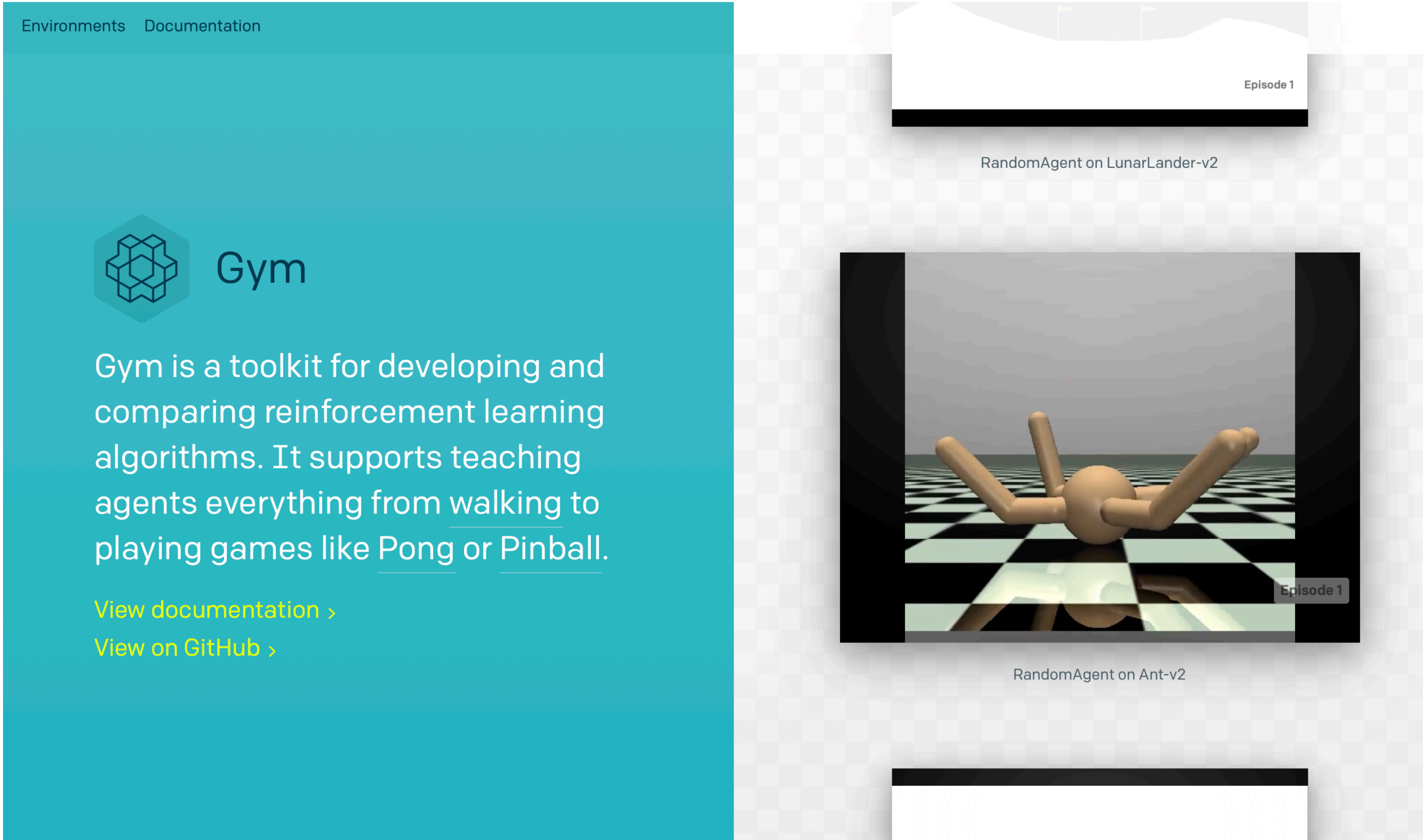
Taxi example

Action Space and the Rewards

- Default reward: -1
- Drop-off at right destination: 20
- Pickup at wrong location: -10
- Drop-off at wrong location: -10



Gym



Environments
Documentation
Episode 1
RandomAgent on LunarLander-v2
Gym
Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.
View documentation ›
View on GitHub ›
Episode 1
RandomAgent on Ant-v2

Gym

Open source interface to reinforcement learning tasks.

The gym library provides an easy-to-use suite of reinforcement learning tasks.

```
import gym
env = gym.make("CartPole-v1")
observation = env.reset()
for _ in range(1000):
  env.render()
  action = env.action_space.sample() # your agent here (this takes random actions)
  observation, reward, done, info = env.step(action)

  if done:
    observation = env.reset()
env.close()
```

We provide the environment; you provide the algorithm.

You can write your agent using your existing numerical computation library, such as TensorFlow or Theano.

Questions?