



Session #1: Introduction to the Course

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)

What is Reinforcement Learning ?

- reward based learning / feedback based learning
- not a type of NN nor it is an alternative to NN. Rather it is an approach for learning
- Autonomous driving, gaming

Why Reinforcement Learning ?

- a goal-oriented learning based on interaction with environment



Course Objectives

Course Objectives:

1. Understand
 - a. the conceptual, mathematical foundations of deep reinforcement learning
 - b. various classic & state of the art Deep Reinforcement Learning algorithms
2. Implement and Evaluate the deep reinforcement learning solutions to various problems like planning, control and decision making in various domains
3. Provide conceptual, mathematical and practical exposure on DRL
 - a. to understand the recent developments in deep reinforcement learning and
 - b. to enable modelling new problems as DRL problems.



Learning Outcomes

1. understand the fundamental concepts of reinforcement learning (RL), algorithms and apply them for solving problems including control, decision-making, and planning.
2. Implement DRL algorithms, handle challenges in training due to stability and convergence
3. evaluate the performance of DRL algorithms, including metrics such as sample efficiency, robustness and generalization.
4. understand the challenges and opportunities of applying DRL to real-world problems & model real life problems



Course Operation

- **Instructors**

Prof. S.P.Vimal

Dr. Sangeetha Viswanathan

- **Textbooks**

1. Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, Second Ed. , MIT Press
2. Foundations of Deep Reinforcement Learning: Theory and Practice in Python (Addison-Wesley Data & Analytics Series) 1st Edition by Laura Graesser and Wah Loon Keng



Course Operation

- Evaluation

Three Quizzes for 5% each; Best 2 towards final grading; No Makeup; → 10%

Two Assignments - Tensorflow/ Pytorch / OpenAI Gym Toolkit → 20 %

Mid Term Exam - 30%

Comprehensive Exam - 40%

- Webinars/Tutorials

4 tutorials : 2 before mid-sem & 2 after mid-sem

- Teaching Assistants



Course Operation

- **How to reach us ?** (for any question on lab aspects, availability of slides on portal, quiz availability , assignment operations)

Prof. S.P.Vimal - vimalsp@wilp.bits-pilani.ac.in

Dr.Sangeetha Viswanathan - sangeetha.viswanathan@pilani.bits-pilani.ac.in

- **Plagiarism**

All submissions for graded components must be the result of your original effort. It is strictly prohibited to copy and paste verbatim from any sources, whether online or from your peers. The use of unauthorized sources or materials, as well as collusion or unauthorized collaboration to gain an unfair advantage, is also strictly prohibited. Please note that we will not distinguish between the person sharing their resources and the one receiving them for plagiarism, and the consequences will apply to both parties equally.

In cases where suspicious circumstances arise, such as identical verbatim answers or a significant overlap of unreasonable similarities in a set of submissions, will be investigated, and severe punishments will be imposed on all those found guilty of plagiarism.



Reinforcement Learning

Reinforcement learning (RL) is based on rewarding desired behaviors or punishing undesired ones. Instead of one input producing one output, the algorithm produces a variety of outputs and is trained to select the right one based on certain variables – Gartner

When to use RL?

RL can be used in large environments in the following situations:

- 1.A model of the environment is known, but an analytic solution is not available;
- 2.Only a simulation model of the environment is given (the subject of simulation-based optimization)
- 3.The only way to collect information about the environment is to interact with it.

(Deep) Reinforcement Learning

<u>Paradigm</u>	 Supervised Learning	 Unsupervised Learning	 Reinforcement Learning
<u>Objective</u>	$p_{\theta}(y x)$	$p_{\theta}(x)$	$\pi_{\theta}(a s)$
<u>Applications</u>	<ul style="list-style-type: none">→ Classification→ Regression	<ul style="list-style-type: none">→ Inference→ Generation	<ul style="list-style-type: none">→ Prediction→ Control

Types of Learning

Criteria	Supervised ML	Unsupervised ML	Reinforcement ML
<i>Definition</i>	Learns by using labelled data	Trained using unlabelled data without any guidance.	Works on interacting with the environment
<i>Type of data</i>	Labelled data	Unlabelled data	No – predefined data
<i>Type of problems</i>	Regression and classification	Association and Clustering	Exploitation or Exploration
<i>Supervision</i>	Extra supervision	No supervision	No supervision
<i>Algorithms</i>	Linear Regression, Logistic Regression, SVM, KNN etc.	K – Means, C – Means, Apriori	Q – Learning, SARSA
<i>Aim</i>	Calculate outcomes	Discover underlying patterns	Learn a series of action
<i>Application</i>	Risk Evaluation, Forecast Sales	Recommendation System, Anomaly Detection	Self Driving Cars, Gaming, Healthcare



Characteristics of RL

- No supervision, only a real value or reward signal
- Decision making is sequential
- Time plays a major role in reinforcement problems
- Feedback isn't prompt but delayed
- The following data it receives is determined by the agent's actions



Types of Reinforcement learning

- **Positive Reinforcement** - an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.
 - Maximizes Performance
 - Sustain Change for a long period of time
 - Too much Reinforcement can lead to an overload of states which can diminish the results

Types of Reinforcement Learning

Negative Reinforcement - strengthening of behavior because a negative condition is stopped or avoided. Advantages of negative reinforcement learning:

- Increases Behavior
- Provide defiance to a minimum standard of performance
- It Only provides enough to meet up the minimum behavior

Manager
praises the
employee

**Positive
Reinforcement**

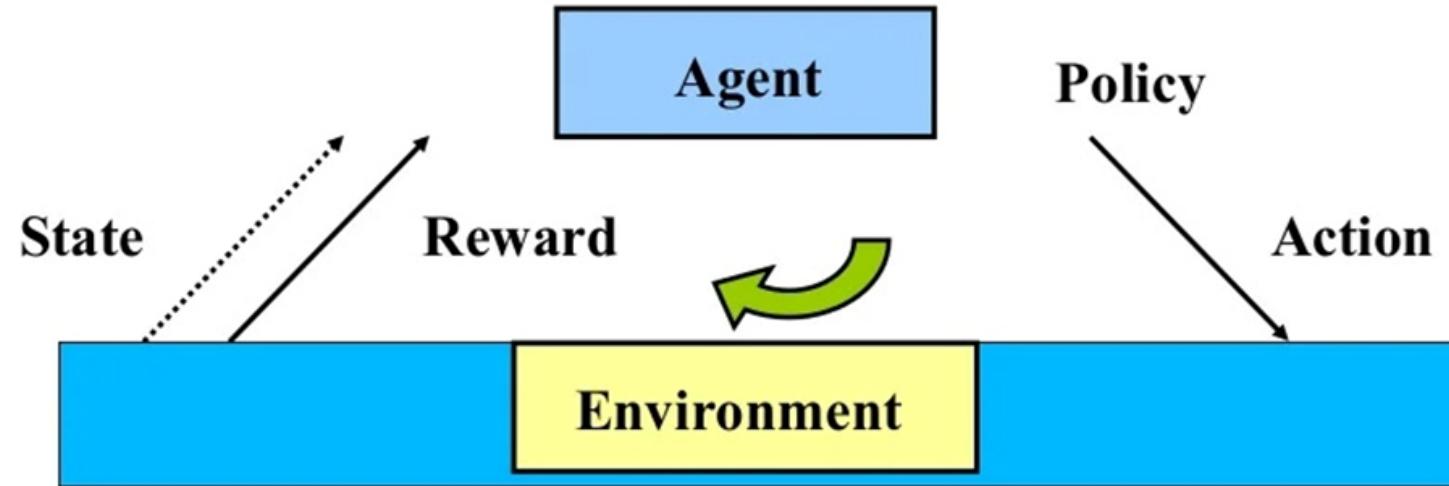
- Positive behavior followed by positive consequences

**Negative
Reinforcement**

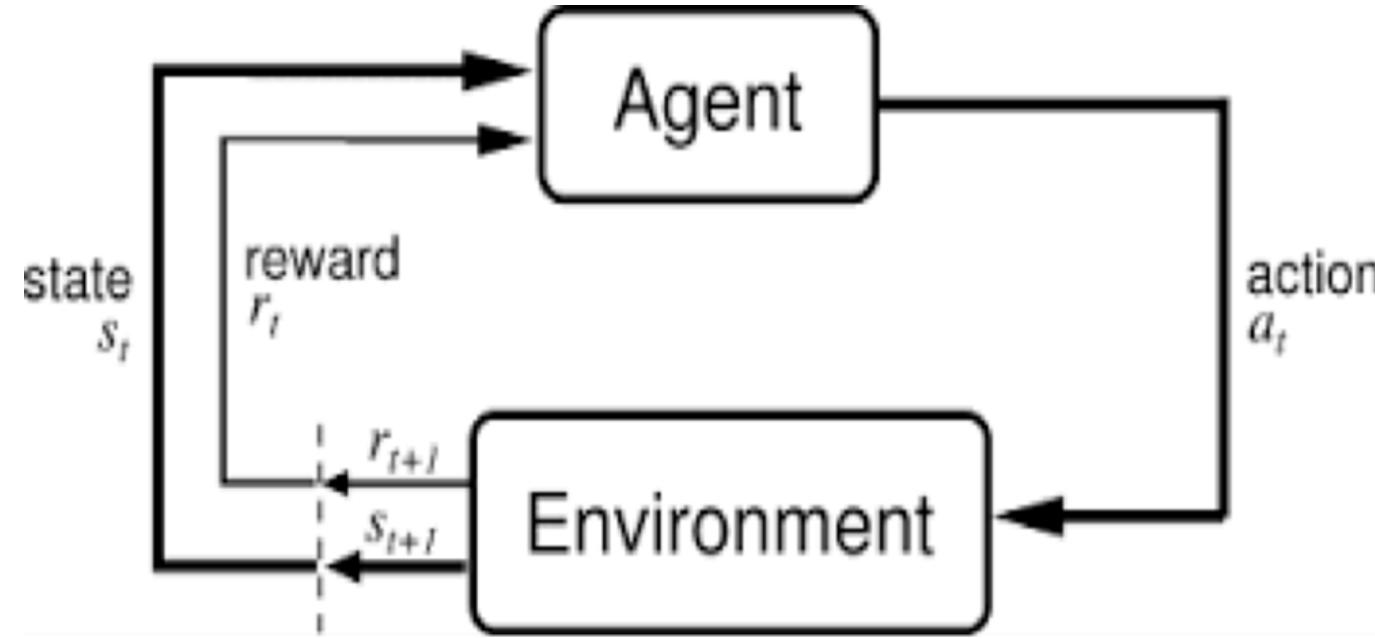
- Positive behavior followed by removal of negative consequences

Manager
stops nagging
the employee

Elements of Reinforcement Learning



Elements of Reinforcement Learning



Beyond the agent and the environment, one can identify four main sub-elements of a reinforcement learning system: *a policy*, *a reward*, *a value function*, and, optionally, *a model* of the environment.



Elements of Reinforcement Learning

- **Agent**

- an **entity** that tries to learn the best way to perform a specific task.
- In our example, the child is the agent who learns to ride a bicycle.

- **Action (A) -**

- **what the agent does** at each time step.
- In the example of a child learning to walk, the action would be “walking”.
- A is the set of all possible moves.
- In video games, the list might include running right or left, jumping high or low, crouching or standing still.



Elements of Reinforcement Learning

•State (S)

- **current situation** of the agent.
- After doing performing an action, the agent can move to different states.
- In the example of a child learning to walk, the child can take the action of taking a step and move to the next state (position).

•Rewards (R)

- feedback that is given to the agent based on the action of the agent.
- If the action of the agent is good and can lead to winning or a positive side then a positive reward is given and vice versa.



Elements of Reinforcement Learning

- **Environment**

- outside world of an agent or physical world in which the agent operates.

- **Discount factor**

- The **discount factor** is multiplied by future rewards as discovered by the agent in order to dampen these rewards' effect on the agent's choice of action. Why? It is designed to make future rewards worth less than immediate rewards.

Often expressed with the lower-case Greek letter gamma: γ . If γ is .8, and there's a reward of 10 points after 3 time steps, the present value of that reward is $0.8^3 \times 10$.



Elements of Reinforcement Learning

Formal Definition - ***Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward.***



Elements of Reinforcement Learning

- **Goal of RL** - maximize the total amount of rewards or cumulative rewards that are received by taking actions in given states.
- Notations –
 - a set of states as S ,
 - a set of actions as A ,
 - a set of rewards as R .

At each time step $t = 0, 1, 2, \dots$, some representation of the environment's state $S_t \in S$ is received by the agent. According to this state, the agent selects an action $A_t \in A$ which gives us the state-action pair (S_t, A_t) . In the next time step $t+1$, the transition of the environment happens and the new state $S_{t+1} \in S$ is achieved. At this time step $t+1$, a reward $R_{t+1} \in R$ is received by the agent for the action A_t taken from state S_t .



Elements of Reinforcement Learning

- Maximize cumulative rewards, Expected Return G_t

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \end{aligned}$$

- Discount factor γ is introduced here which forces the agent to focus on immediate rewards instead of future rewards. The value of γ remains between 0 and 1.



Elements of Reinforcement Learning

- **Policy (π)**

- Policy in RL decides which action will the agent take in the current state.
- It tells the *probability that an agent will select a specific action from a specific state*.
- Policy is a function that maps a given state to probabilities of selecting each possible action from the given state.

- If at time t , an agent follows policy π , then $\pi(a/s)$ becomes the probability that the action at time step t is $a_{t=a}$ if the state at time step t is $S_{t=s}$. The meaning of this is, the probability that an agent will take an action a in state s is $\pi(a/s)$ at time t with policy π .



Elements of Reinforcement Learning

- **Value Functions**

- a simple measure of how good it is for an agent to be in a given state, or how good it is for the agent to perform a given action in a given state.

- Two types

- state-value function
- action-value function



Elements of Reinforcement Learning

• State-value function

- The *state-value function* for policy π denoted as v_π determines the *goodness of any given state for an agent who is following policy π* .
- This function gives us the value which is the expected return starting from state s at time step t and following policy π afterward.

$$\begin{aligned} v_\pi(s) &= E_\pi[G_t \mid S_t = s] \\ &= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]. \end{aligned}$$



Elements of Reinforcement Learning

•Action value function

- determines the goodness of the action taken by the agent from a given state for policy π .
- This function gives the value which is the expected return starting from state s at time step t , with action a , and following policy π afterward.
- The output of this function is also called as *Q-value* where q stands for Quality. Note that in the *state-value function*, we did not consider the action taken by the agent.

$$\begin{aligned} q_\pi(s, a) &= E_\pi[G_t \mid S_t = s, A_t = a] \\ &= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]. \end{aligned}$$

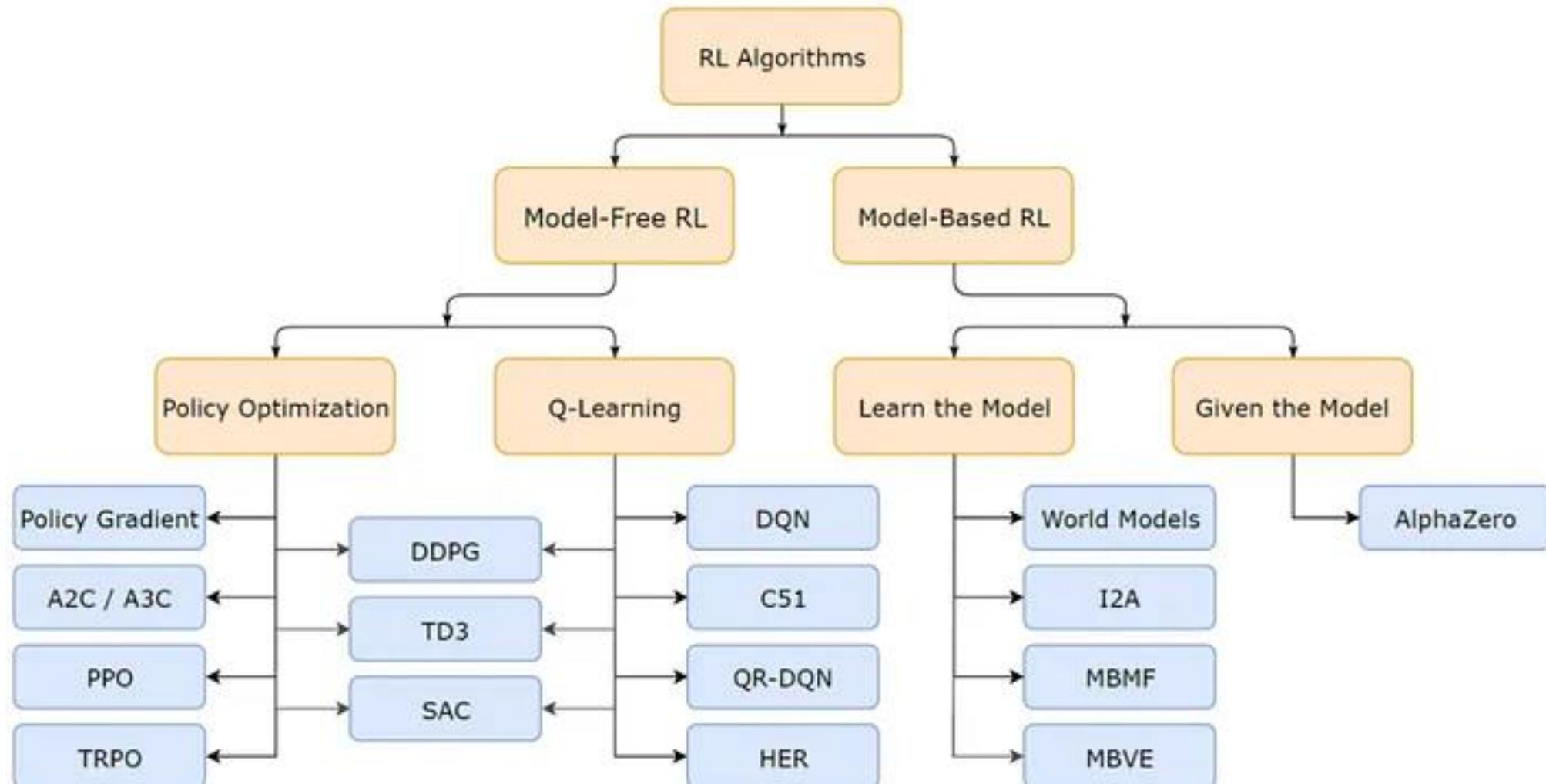


Elements of Reinforcement Learning

- **Model of the environment**

- mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave.
- For example, given a state and action, the model might predict the resultant next state and next reward.
- Models are used for planning

(Deep) Reinforcement Learning





Advantages of Reinforcement Learning

- solve very complex problems that cannot be solved by conventional techniques
- achieve long-term results
- model can correct the errors that occurred during the training process.
- In the absence of a training dataset, it is bound to learn from its experience
- can be useful when the only way to collect information about the environment is to interact with it
- Reinforcement learning algorithms maintain a ***balance between exploration and exploitation***. Exploration is the process of trying different things to see if they are better than what has been tried before. Exploitation is the process of trying the things that have worked best in the past. Other learning algorithms do not perform this balance



An example scenario - Tic-Tac-Toe

Two players take turns playing on a three-by-three board. One player plays Xs and the other Os until one player wins by placing three marks in a row, horizontally, vertically, or diagonally

Assumptions

- playing against an imperfect player, one whose play is sometimes incorrect and allows you to win

Aim

How might we construct a player that will find the imperfections in its opponent's play and learn to maximize its chances of winning?



Reinforcement Learning for Tic-Tac-Toe

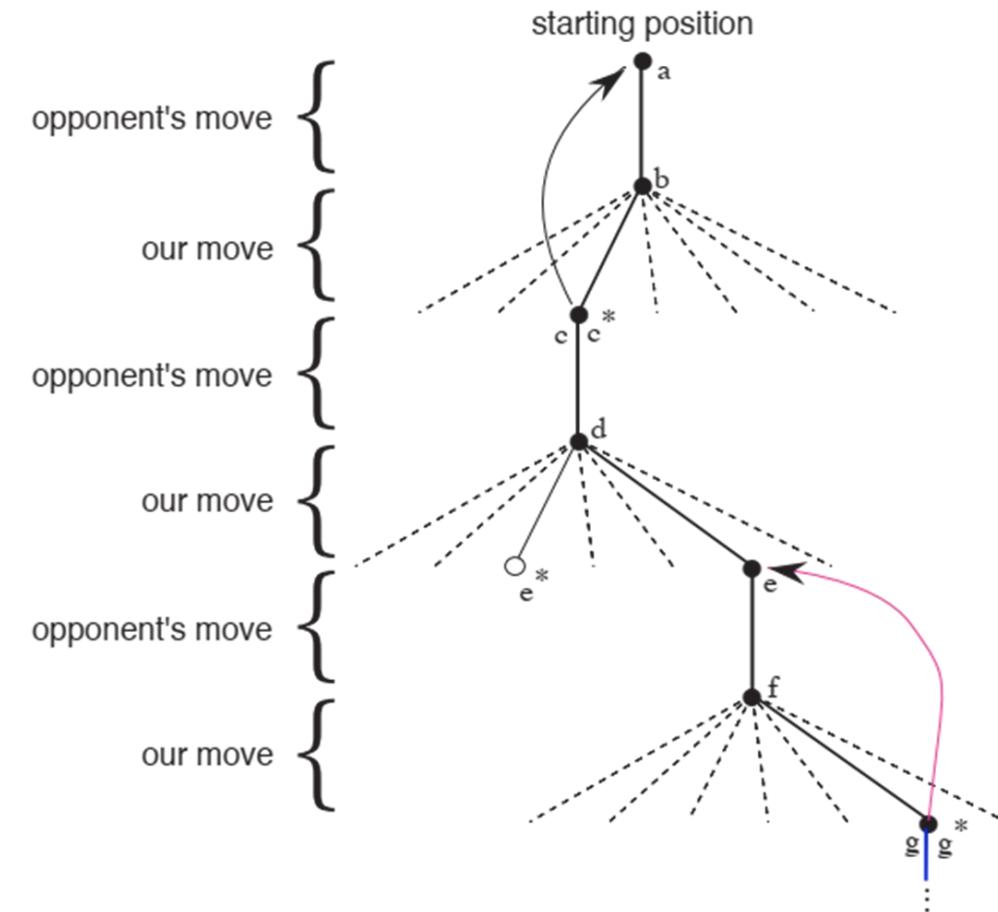
- We set up a table of numbers, one for each possible state of the game. Each number will be the latest estimate of the probability of our winning from that state.
- We treat this estimate as the state's value, and the whole table is the learned value function.
- State A has higher value than state B, or is considered better than state B, if the current estimate of the probability of our winning from A is higher than it is from B.

Reinforcement Learning for Tic-Tac-Toe

- Assuming we always play X's, three X = probability is 1, three O = probability =0 .
Initial = 0.5
- Play many games against the opponent. To select our moves we examine the states that would result from each of our possible moves and look up their current values in the table.
- Most of the time we move greedily, selecting the move that leads to the state with greatest value, i.e, with the highest estimated probability of winning.
- Occasionally, however, we select randomly from among the other moves instead. These are called **exploratory moves** because they cause us to experience states that we might otherwise never see.

Reinforcement Learning for Tic-Tac-Toe

- The solid lines represent the moves taken during a game; the dashed lines represent moves that we (our reinforcement learning player) considered but did not make.
- Our second move was an exploratory move, meaning that it was taken even though another sibling move, the one leading to e^* , was ranked higher.



Reinforcement Learning for Tic-Tac-Toe

- Once a game is started, our agent computes all possible actions it can take in the current state and the new states which would result from each action.
- The values of these states are collected from a ***state_value vector***, which contains values for all possible states in the game.
- The agent can then choose the action which leads to the state with the highest value(exploitation), or chooses a random action(exploration), depending on the value of epsilon.



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



Deep Reinforcement Learning

2022-23 Second Semester, M.Tech (AIML)

Session #2-3: Multi-armed Bandits

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)



Agenda for the class

- Recap
- k-armed Bandit Problem & its significance
- Action-Value Methods
 - Sample Average Method & Incremental Implementation
- Non-stationary Problem
- Initial Values & Action Selection
- Gradient Bandit Algorithms [Class #3]
- Associative Search [Class #3]



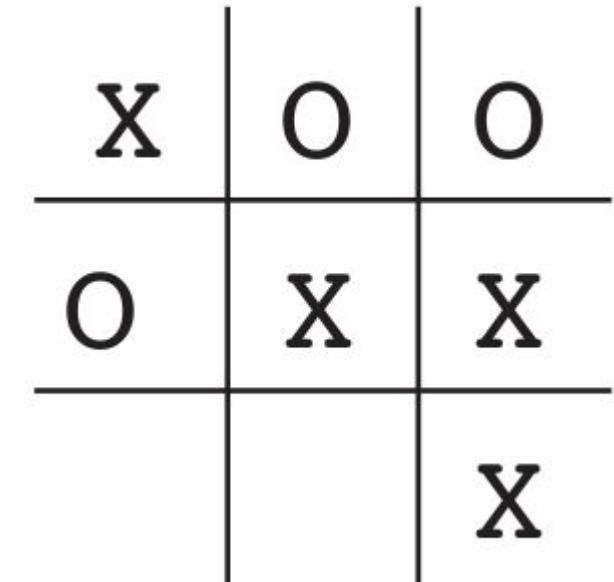
Tic-Tac-Toe (from prev. class)

X	O	O
O	X	X
		X

Tic-Tac-Toe

States	Initial Values
$[X]$	0.5
$[X \ O \ O]$ $\quad X$	0.5
$[X \ O \ O]$ $\quad X \quad X$	1.0
$[X \ O]$ $X \ O$ $\quad X \ O$	0
...	...

Learning Task: Play as many times against the opponent and learn the values



Set up a table of states initial values

Tic-Tac-Toe (prev. class)

States

Initial Values

$[X]$	0.5
-------	-----

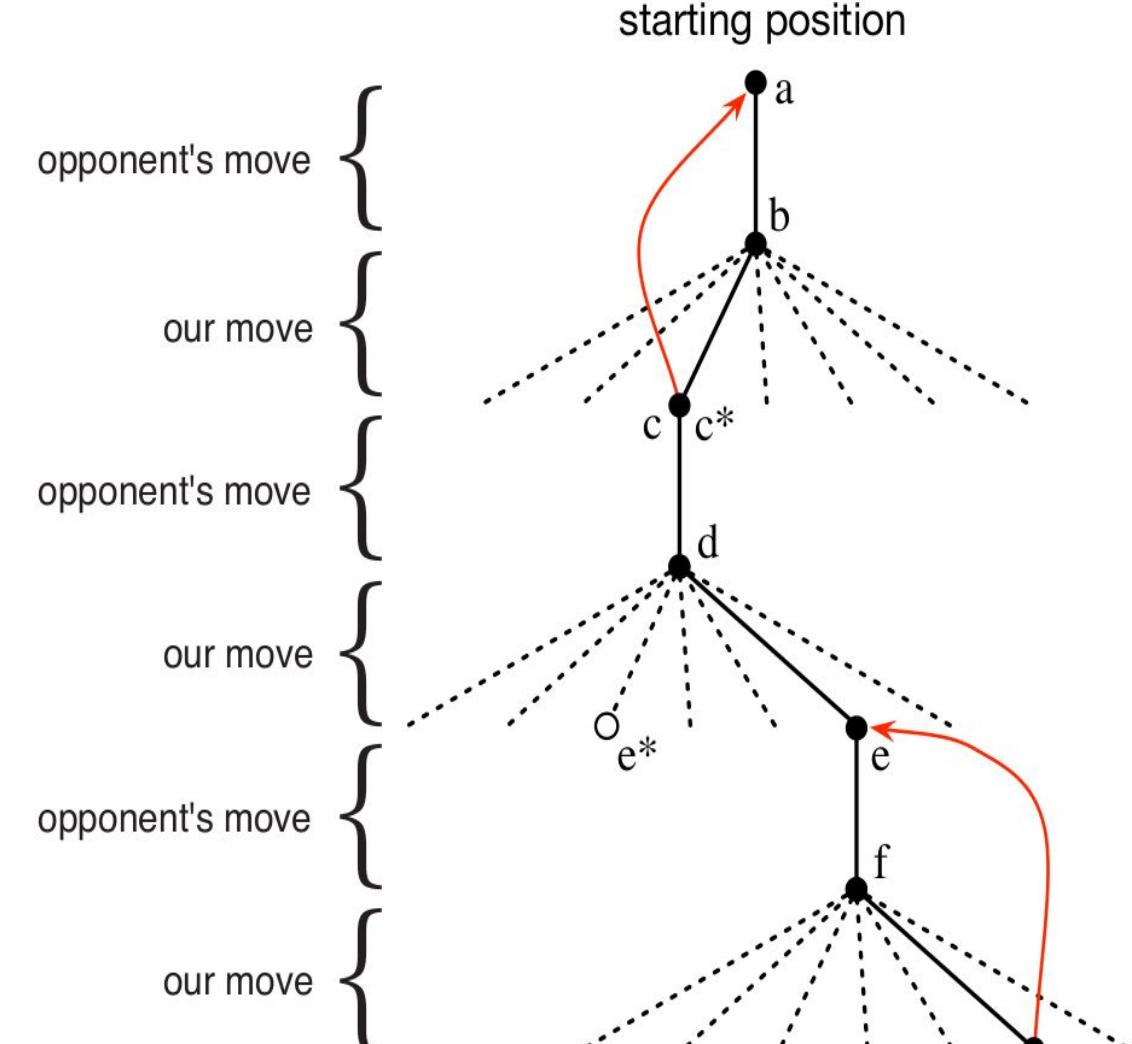
$[X \ O \ O]$ $\quad X$	0.5
----------------------------	-----

$[X \ O \ O]$ $\quad X \quad X$	1.0
------------------------------------	-----

$[X \ O]$ $\quad X \quad X \ O$	0
------------------------------------	---

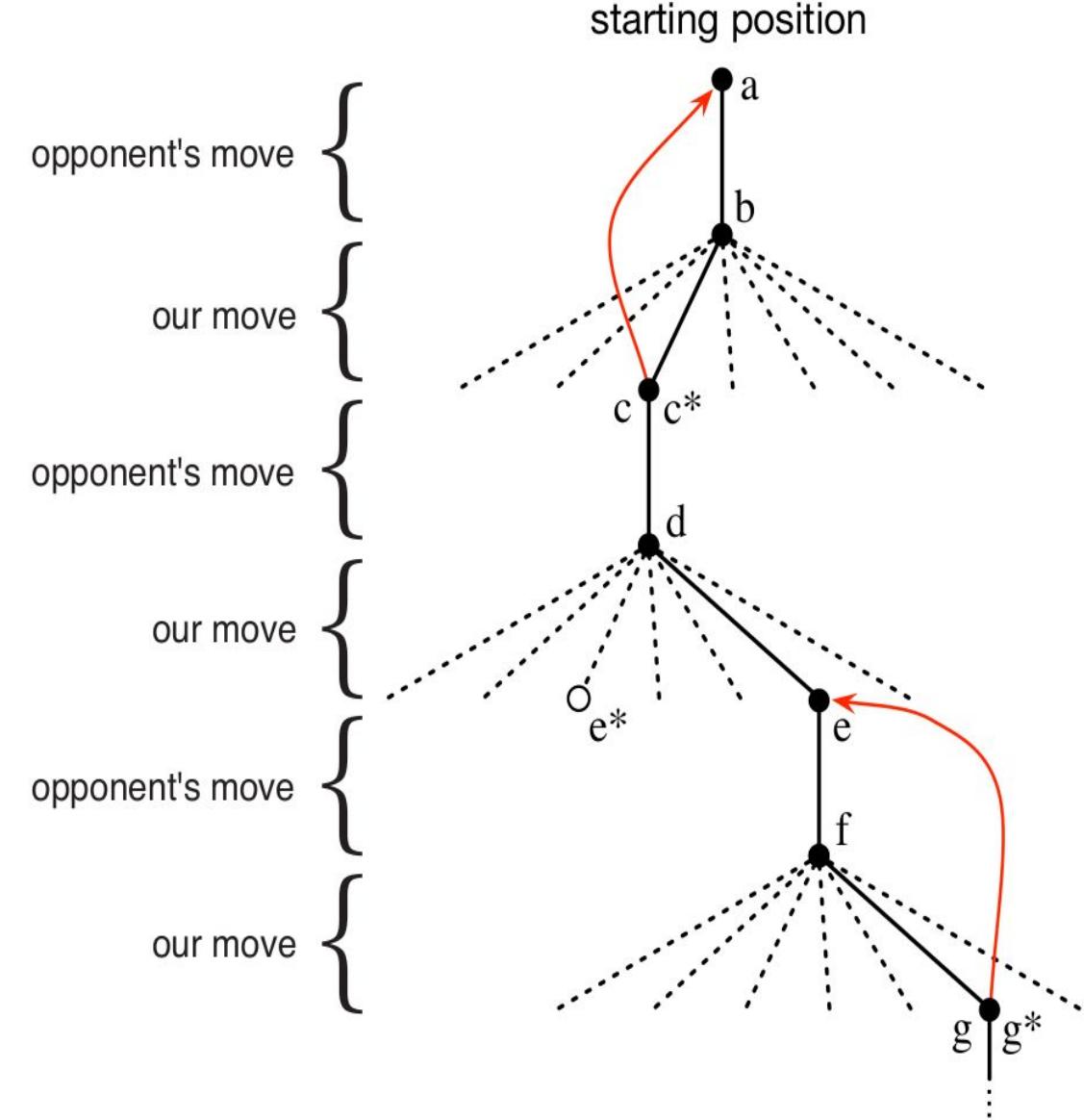
...

S_t - state before greedy move
 S_{t+1} - state after greedy move



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

Tic-Tac-Toe (prev. class)



Temporal Difference Learning Rule

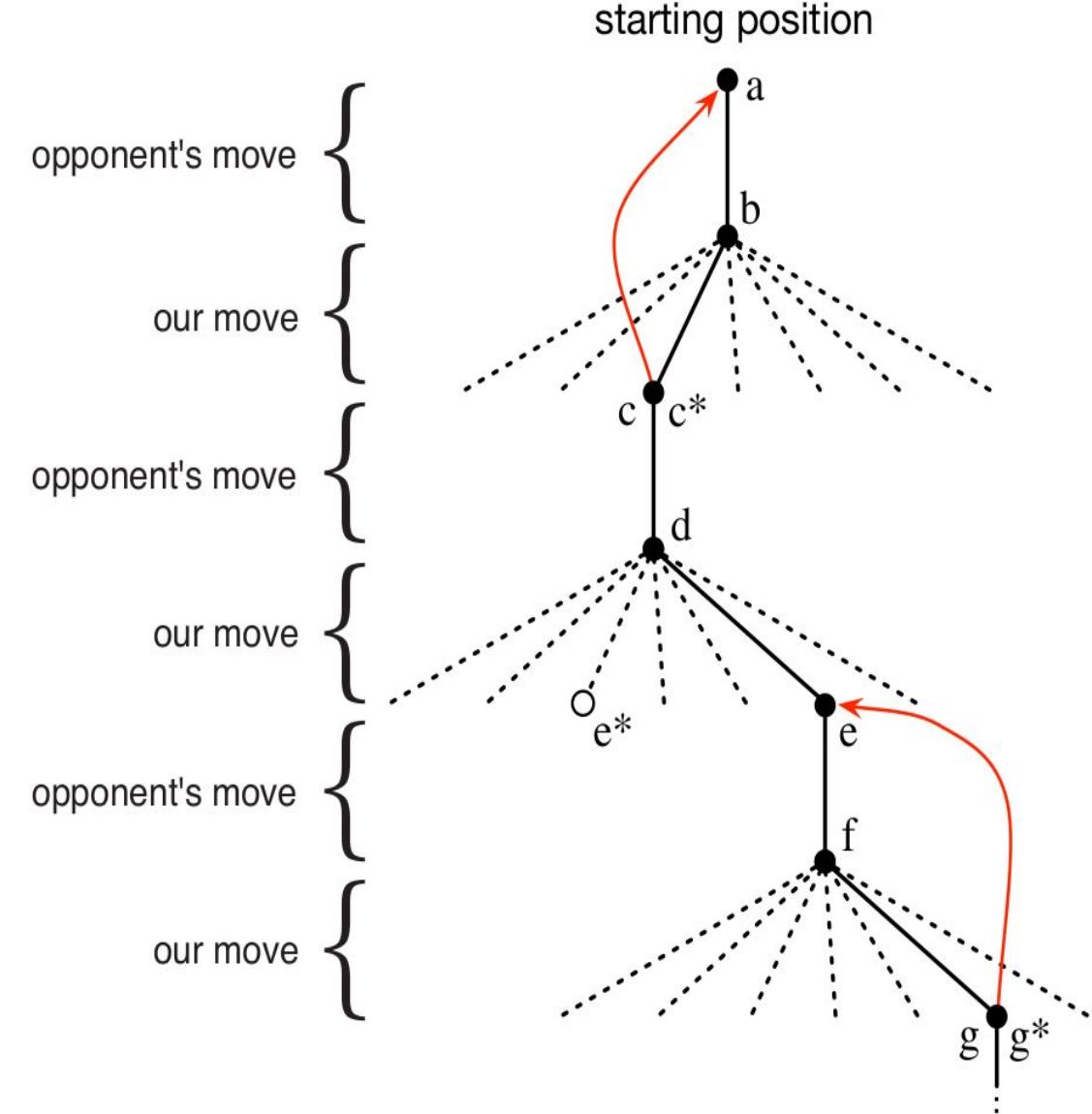
$$V(S_t) \leftarrow V(S_t) + \alpha [V(S_{t+1}) - V(S_t)]$$

α - Step Size Parameter

Tic-Tac-Toe (prev. class)

Questions:

- (1) What happens if α is gradually made to 0 over many games with the opponent?
- (2) What happens if α is gradually reduced over many games, but never made 0?
- (3) What happens if α is kept constant throughout its life time?



Temporal Difference Learning Rule

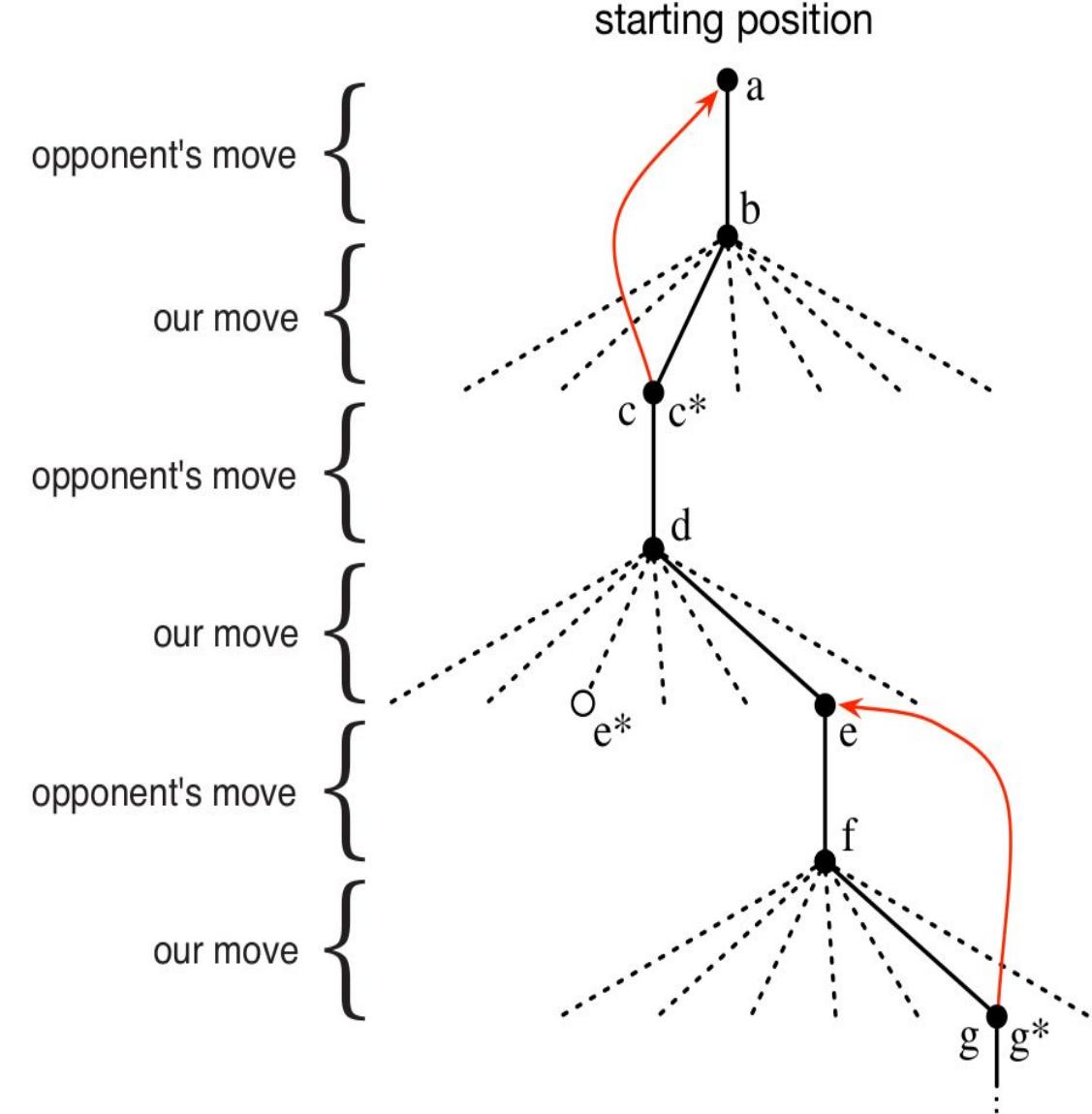
$$V(S_t) \leftarrow V(S_t) + \alpha [V(S_{t+1}) - V(S_t)]$$

α - Step Size Parameter

Tic-Tac-Toe (prev. class)

Key Takeaways:

- (1) Learning while interacting with the environment (opponent).
- (2) We have a clear goal
- (3) Our policy is to make moves that maximizes our chances of reaching goal
 - o Use the values of states most of the time (exploration) and explore rest of the time.



Temporal Difference Learning Rule

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

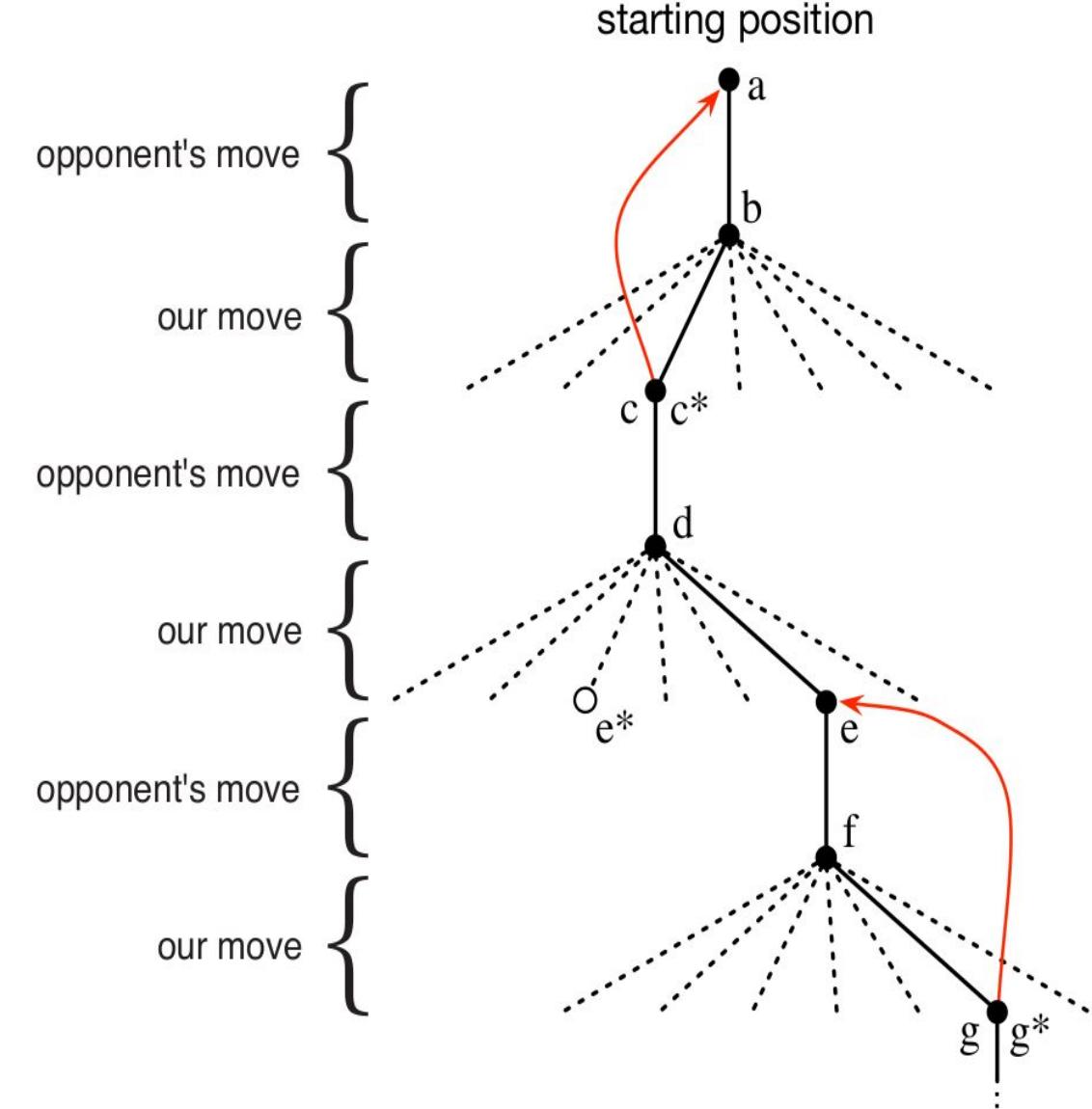
α - Step Size Parameter

Tic-Tac-Toe (prev. class)

Reading Assigned:

Identify how this reinforcement learning solution is different from solutions using minimax algorithm and genetic algorithms.

Post your answers in the discussion forum;



Temporal Difference Learning Rule

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

α - Step Size Parameter



K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward
 - Reward is chosen from a stationary probability distribution that depends on the selected action
- **Objective** : to *maximize the expected total reward over some time period*

K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward

Reward is chosen from a stationary probability distribution that depends on the selected action

- **Objective** : to *maximize the expected total reward over some time period*



K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward

Reward is chosen from a stationary probability distribution that depends on the selected action

- **Objective** : to *maximize the expected total reward over some time period*



Strategy:

- Identify the best lever(s)
- Keep pulling the identified ones

Questions:

- How do we define the **best ones**?
- What are the best levers?

K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward

Reward is chosen from a stationary probability distribution that depends on the selected action

- **Objective** : to *maximize the expected total reward over some time period*



Strategy:

- Identify the best lever(s)
- Keep pulling the identified ones

Questions:

- How do we define the **best ones**?
- What are the best levers?

K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward

Reward is chosen from a stationary probability distribution that depends on the selected action

- **Objective** : to *maximize the expected total reward over some time period*



$$\mathbb{E}[a] = -\$0.5$$



$$\mathbb{E}[b] = -\$0.2$$



$$\mathbb{E}[c] = \$0.1$$



$$\mathbb{E}[d] = \$0.11$$

- **Expected Mean Reward** for each action selected
→ call it **Value** of the action

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$$

K-armed Bandit Problem

$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a]$$

- A_t - action selected on time step t
- $Q_t(a)$ - estimated value of action a at time step t
- $q_*(a)$ - value of an arbitrary action a

Note: If you knew the value of each action, then it would be trivial to solve the k -armed bandit problem: you would always select the action with highest value :-)

K-armed Bandit Problem



$$E[a] = -\$0.5$$



$$E[b] = -\$0.2$$



$$E[c] = \$0.1$$



$$E[d] = \$0.11$$

K-armed Bandit Problem



-1, -1, 5
 $\hat{E}[a] = 1$



-0.2, -0.2
 $\hat{E}[b] = -0.2$



-0.5, -0.5, -0.5
 $\hat{E}[c] = -0.5$



-2, -2
 $\hat{E}[d] = -2$

Keep pulling the levers; update the estimate of action values;

K-armed Bandit Problem



-1, -1, 5
 $\hat{E}[a] = 1$



-0.2, -0.2
 $\hat{E}[b] = -0.2$



-0.5, -0.5, -0.5
 $\hat{E}[c] = -0.5$



-2, -2
 $\hat{E}[d] = -2$

K-armed Bandit Problem

- How to maintain the estimate of expected rewards for each action?
Average the rewards actually received !!!

$$\begin{aligned} Q_t(a) &\doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} \\ &= \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \end{aligned}$$

- How to use the estimate in selecting the right action?

Greedy Action Selection $A_t \doteq \arg \max_a Q_t(a)$

K-armed Bandit Problem

- How to use the estimate in selecting the right action?

Greedy Action Selection

$$A_t \doteq \arg \max_a Q_t(a)$$

Actions which are inferior by the value estimate upto time t, could be indeed better than the greedy action at t !!!

- Exploration vs. Exploitation?

ϵ -Greedy Action Selection / near-greedy action selection

Behave greedily most of the time; Once in a while, with small probability ϵ select randomly from among all the actions with equal probability, independently of the action-value estimates.

K-armed Bandit Problem



$\{-1, -1, 5\}$

$$N_{11}(a)=3$$

$$q_*(a)=-\$0.5$$

$$Q_{11}(a)=1$$



$\{-0.2, -0.2\}$

$$N_{11}(b)=2$$

$$q_*(b)=-\$0.2$$

$$Q_{11}(b)=-0.2$$



$\{-0.5, -0.5, -0.5\}$

$$N_{11}(c)=3$$

$$q_*(c)=\$0.1$$

$$Q_{11}(c)=-0.5$$



$\{-2, -2\}$

$$N_{11}(d)=2$$

$$\text{q*}(d)=\$1$$

$$Q_{11}(d)=-2$$

K-armed Bandit Problem

Greedy Action



K-armed Bandit Problem

Action to Explore



$\{-1, -1, 5\}$

$$N_{11}(a)=3$$

$$q_*(a)=-\$0.5$$

$$Q_{11}(a)=1$$



$\{-0.2, -0.2\}$

$$N_{11}(b)=2$$

$$q_*(b)=-\$0.2$$

$$Q_{11}(b)=-0.2$$



$\{-0.5, -0.5, -0.5\}$

$$N_{11}(c)=3$$

$$q_*(c)=\$0.1$$

$$Q_{11}(c)=-0.5$$



$\{-2, -2\}$

$$N_{11}(d)=2$$

$$\text{q*}(d)=\$1$$

$$Q_{11}(d)=-2$$

K-armed Bandit Problem

ϵ -Greedy Action Selection / near-greedy action selection

```
epsilon = 0.05 // small value to control exploration
def get_action():
    if random.random() > epsilon:
        return argmaxa(Q(a))
    else:
        return random.choice(A)
```

- In the limit as the number of steps increases, every action will be sampled by ϵ -greedy action selection an infinite number of times. This ensures that all the $Q_t(a)$ converge to $q_*(a)$.
- Easy to implement / optimize for epsilon / yields good results



Ex-1: In ϵ -greedy action selection, for the case of two actions and $\epsilon = 0.5$, what is the probability that *the greedy action* is selected?

Ex-1: In ϵ -greedy action selection, for the case of two actions and $\epsilon = 0.5$, what is the probability that *the greedy action* is selected?

$p(\text{greedy action})$

$$= p(\text{greedy action AND greedy selection}) + p(\text{greedy action AND random selection})$$

$$= p(\text{greedy action} \mid \text{greedy selection}) p(\text{greedy selection})$$

$$+ p(\text{greedy action} \mid \text{random selection}) p(\text{random selection})$$

$$= p(\text{greedy action} \mid \text{greedy selection}) (1-\epsilon) + p(\text{greedy action} \mid \text{random selection}) (\epsilon)$$

$$= p(\text{greedy action} \mid \text{greedy selection}) (0.5) + p(\text{greedy action} \mid \text{random selection}) (0.5)$$

$$= (1)(0.5) + (0.5)(0.5)$$

$$= 0.5 + 0.25$$

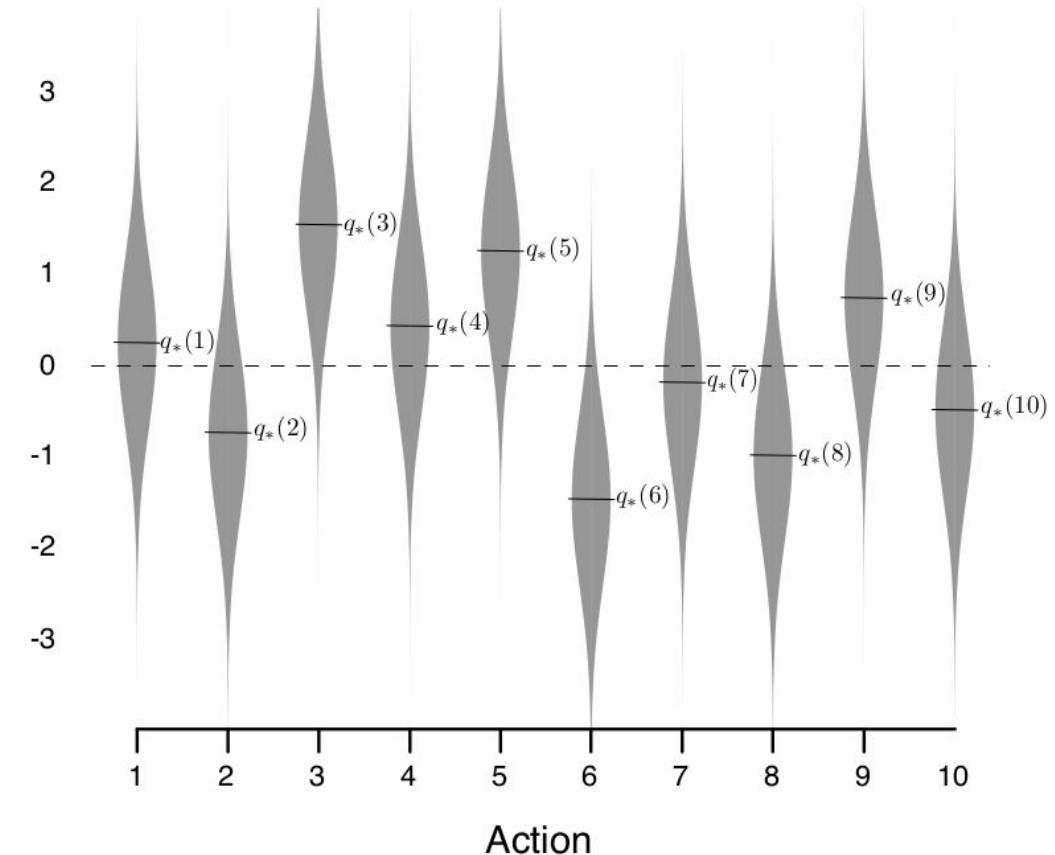
$$= 0.75$$

10-armed Testbed

Example:

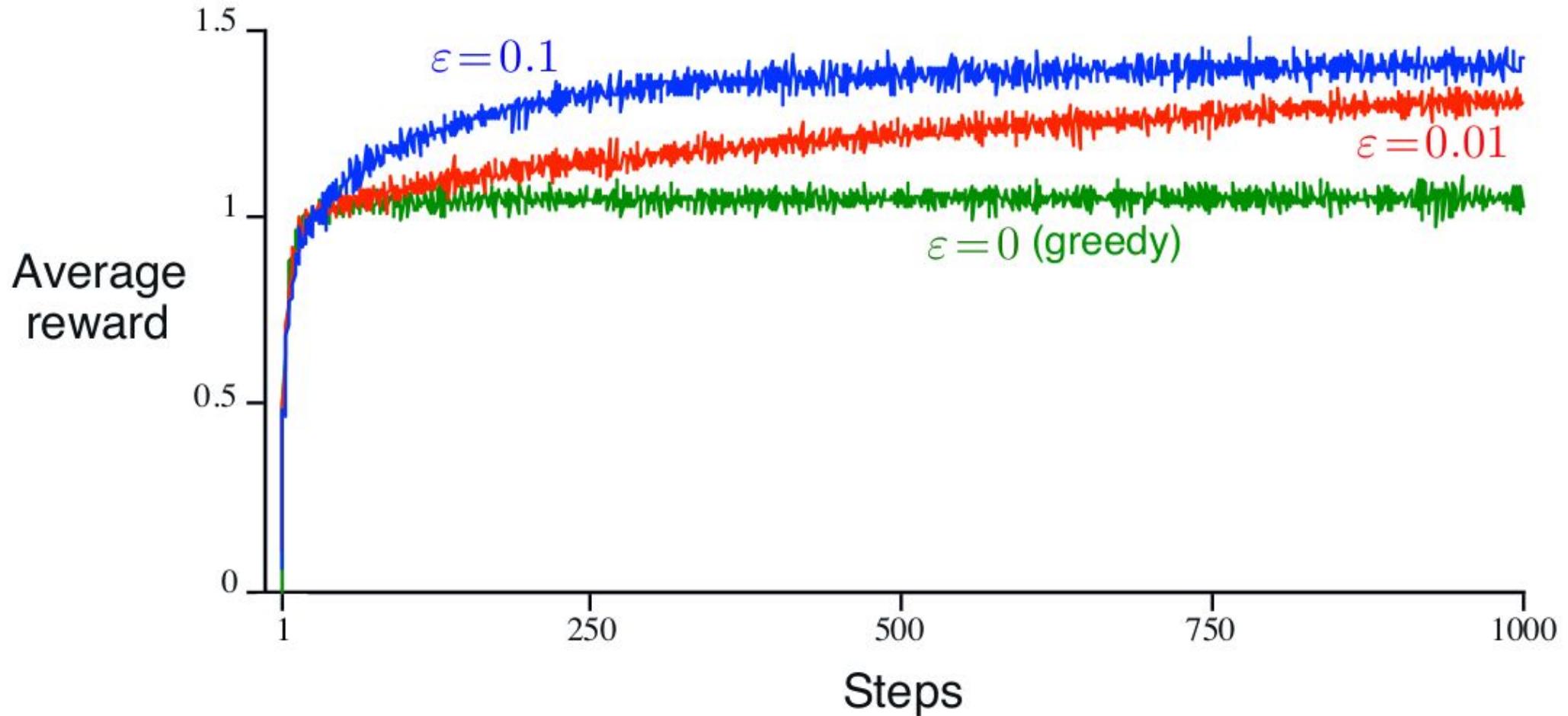
- A set of 2000 randomly generated k -armed bandit problems with $k = 10$
- Action values were selected according to a normal (Gaussian) distribution with mean 0 and variance 1.
- While selecting action A_t at time step t , the actual reward, R_t , was selected from a normal distribution with mean $q_*(A_t)$ and variance 1
- **One Run :** Apply a method for 1000 time steps to one of the bandit problems
- Perform 2000 runs, each run with a different bandit problem, to get an algorithms average behavior

Reward distribution

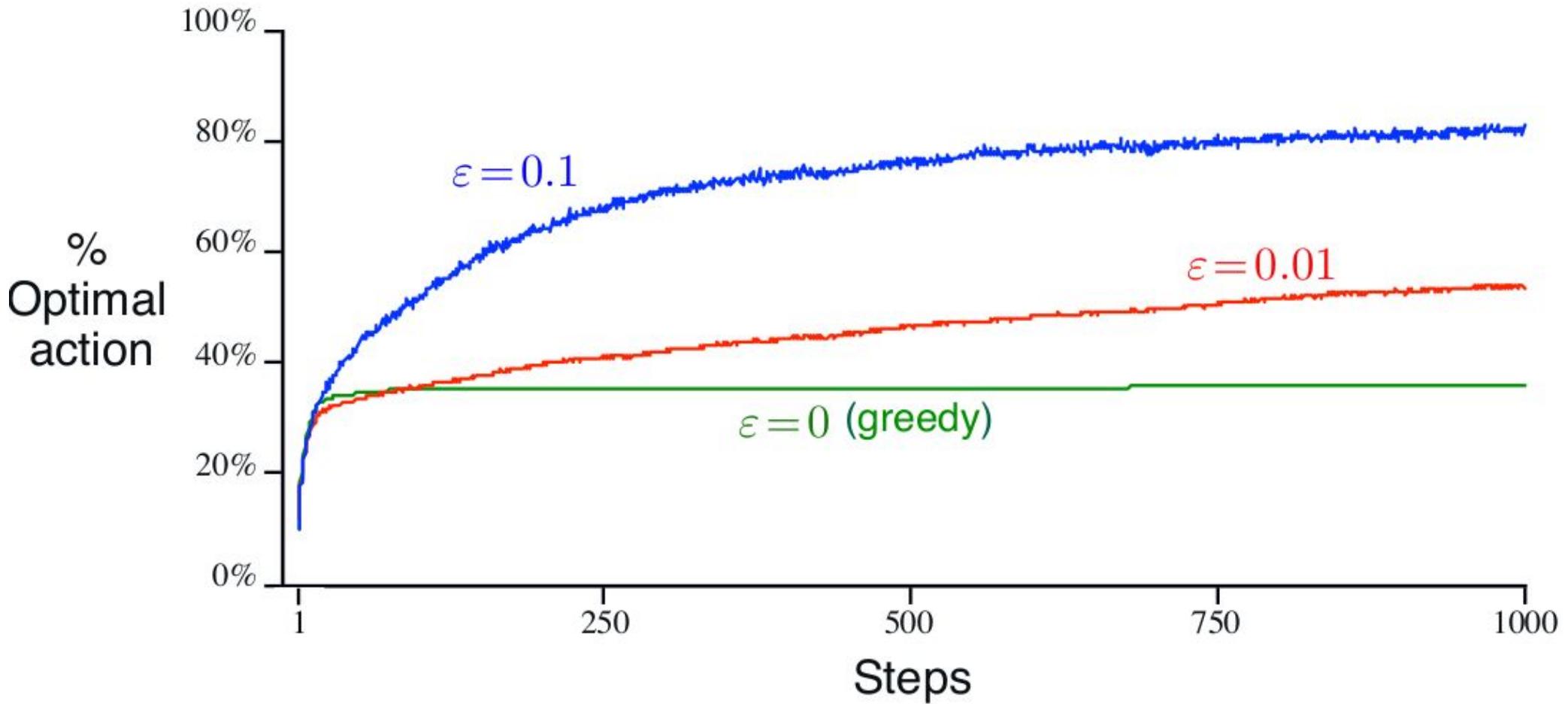


An example bandit problem from the 10-armed testbed

Average performance of ϵ -greedy action-value methods on the 10-armed testbed



Average performance of ϵ -greedy action-value methods on the 10-armed testbed





Discussion on Exploration vs. Exploitation

- 1) What if the reward variance is
 - a. larger, say 10 instead of 1?
 - b. zero ? [deterministic]
- 2) What if the bandit task is non-stationary? [that is, the true values of the actions changed over time]



Ex-2:

Consider a k -armed bandit problem with $k = 4$ actions, denoted 1, 2, 3, and 4.

Consider applying to this problem a bandit algorithm using ϵ -greedy action selection, sample-average action-value estimates, and initial estimates of $Q_1(a) = 0$, for all a .

Suppose the initial sequence of actions and rewards is $A_1 = 1, R_1 = 1, A_2 = 2, R_2 = 1, A_3 = 2, R_3 = 2, A_4 = 2, R_4 = 2, A_5 = 3, R_5 = 0$.

On some of these time steps the ϵ case may have occurred, causing an action to be selected at random.

On which time steps did this definitely occur? On which time steps could this possibly have occurred?

Incremental Implementation

- Efficient approach to compute the estimate of action-value;

$$Q_n \doteq \frac{R_1 + R_2 + \dots + R_{n-1}}{n - 1}.$$

- Given Q_n and the n th reward, R_n , the new average of all n rewards can be computed as follows

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i \\ &= \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} (R_n + (n-1)Q_n) \\ &= \frac{1}{n} (R_n + nQ_n - Q_n) \\ &= Q_n + \frac{1}{n} [R_n - Q_n], \end{aligned}$$

Incremental Implementation

Note:

- StepSize decreases with each update
- We use α or $\alpha_t(a)$ to denote step size (constant / varies with each step)

Discussion:

Const vs. Variable step size?

$$\begin{aligned}
 Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i \\
 &= \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right) \\
 &= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\
 &= \frac{1}{n} \left(R_n + (n-1)Q_n \right) \\
 &= \frac{1}{n} \left(R_n + nQ_n - Q_n \right) \\
 &= Q_n + \frac{1}{n} [R_n - Q_n],
 \end{aligned}$$

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} [\text{Target} - \text{OldEstimate}]$$

Bandit Algorithm with Incremental Update/ ϵ -greedy selection

Initialize, for $a = 1$ to k :

$$Q(a) \leftarrow 0$$

$$N(a) \leftarrow 0$$

Loop forever:

$$A \leftarrow \begin{cases} \arg \max_a Q(a) & \text{with probability } 1 - \varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases} \quad (\text{breaking ties randomly})$$

$$R \leftarrow \text{bandit}(A)$$

$$N(A) \leftarrow N(A) + 1$$

$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$$



Non-stationary Problem

- Most RL problems are non-stationary !
- Give more weight to recent rewards than to long-past rewards !!!

$$Q_{n+1} \doteq Q_n + \alpha [R_n - Q_n]$$

Non-stationary Problem

- Most RL problems are non-stationary !
- Give more weight to recent rewards than to long-past rewards !!!

$$Q_{n+1} \doteq Q_n + \alpha [R_n - Q_n]$$

Exponential recency-weighted average

$$\begin{aligned}
 Q_{n+1} &= Q_n + \alpha [R_n - Q_n] \\
 &= \alpha R_n + (1 - \alpha) Q_n \\
 &= \alpha R_n + (1 - \alpha) [\alpha R_{n-1} + (1 - \alpha) Q_{n-1}] \\
 &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 Q_{n-1} \\
 &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 \alpha R_{n-2} + \\
 &\quad \cdots + (1 - \alpha)^{n-1} \alpha R_1 + (1 - \alpha)^n Q_1 \\
 &= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} R_i.
 \end{aligned}$$



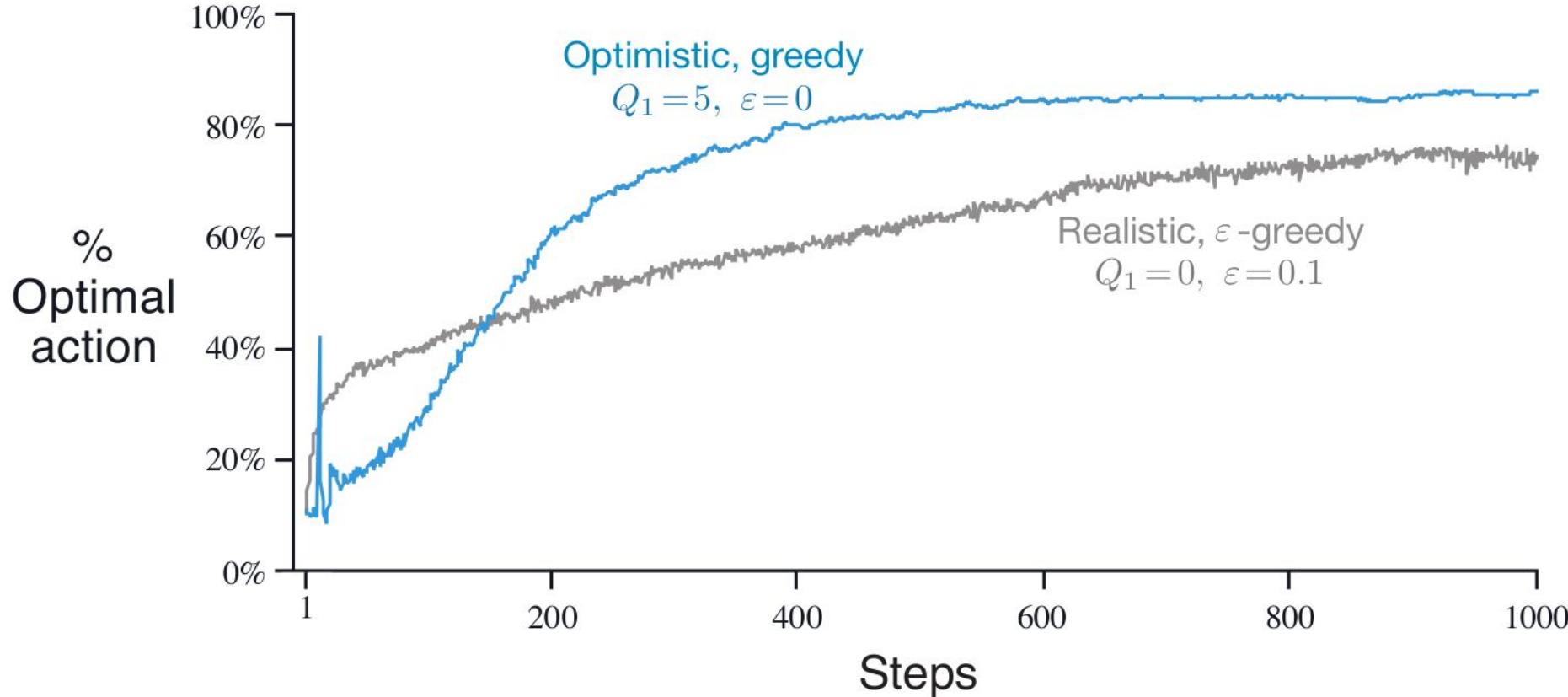
Optimistic Initial Values

- All the above discussed methods are **biased** by their initial estimates
- For sample average method the bias disappears once all actions have been selected at least once
- For methods with constant α , the bias is permanent, though decreasing over time
- Initial action values can also be used as a simple way of encouraging exploration.
- In 10 armed testbed, set initial estimate to +5 rather than 0.

This can encourage action-value methods to explore.

Whichever actions are initially selected, the reward is less than the starting estimates; the learner switches to other actions, being disappointed with the rewards it is receiving. The result is that all actions are tried several times before the value estimates converge.

Optimistic Initial Values



Caution:

Optimistic Initial Values can only be considered as a simple trick that can be quite effective on stationary problems, but it is far from being a generally useful approach to encouraging exploration.

Question:

Explain how in the non-stationary scenario the optimistic initial values will fail (to explore adequately).

The effect of optimistic initial action-value estimates on the 10-armed testbed.
Both methods used a constant step-size parameter, $\alpha = 0.1$

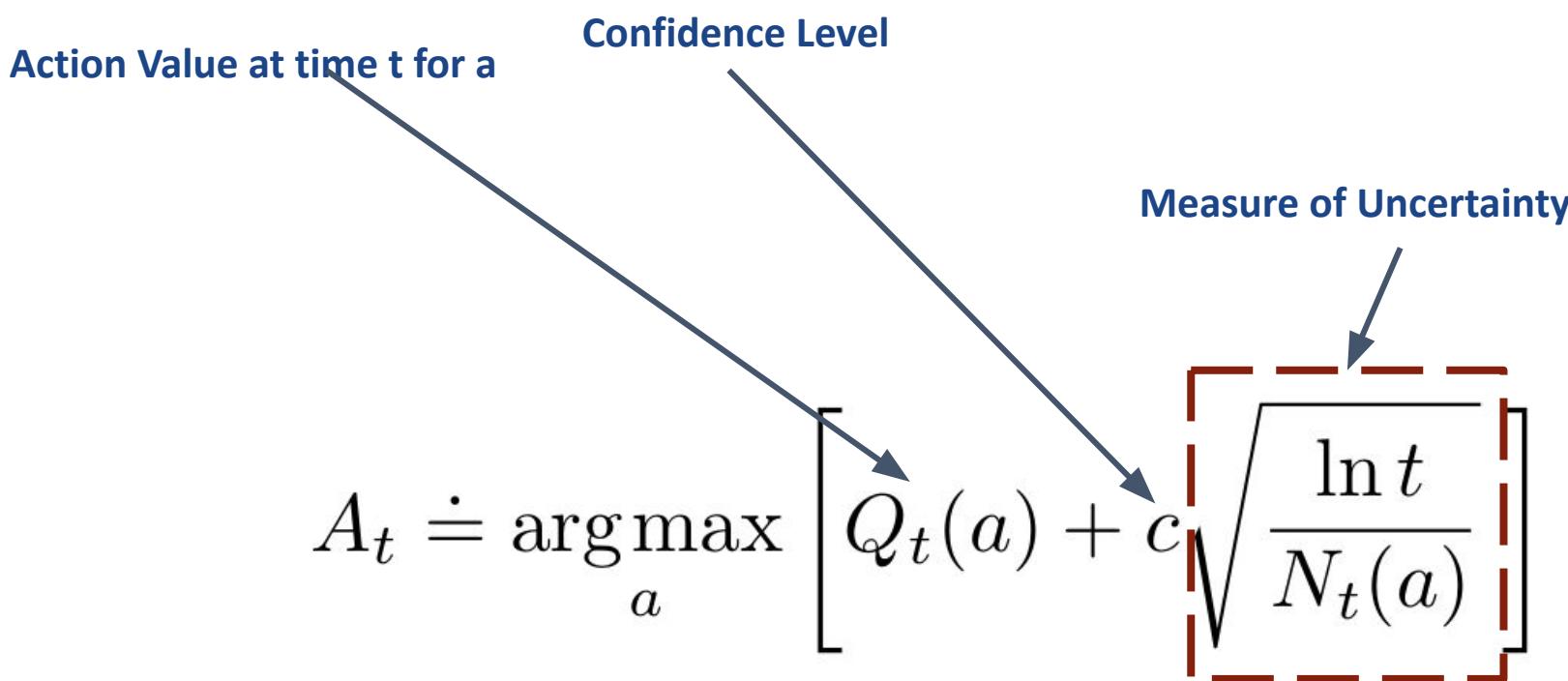
Upper-Confidence-Bound Action Selection

- **ϵ -greedy action selection forces the non-greedy actions to be tried,**
Indiscriminately, with no preference for those that are nearly greedy or particularly uncertain
- It would be better to select among the non-greedy actions according to their potential for actually being optimal
Take into account both how close their estimates are to being maximal and the uncertainties in those estimates.

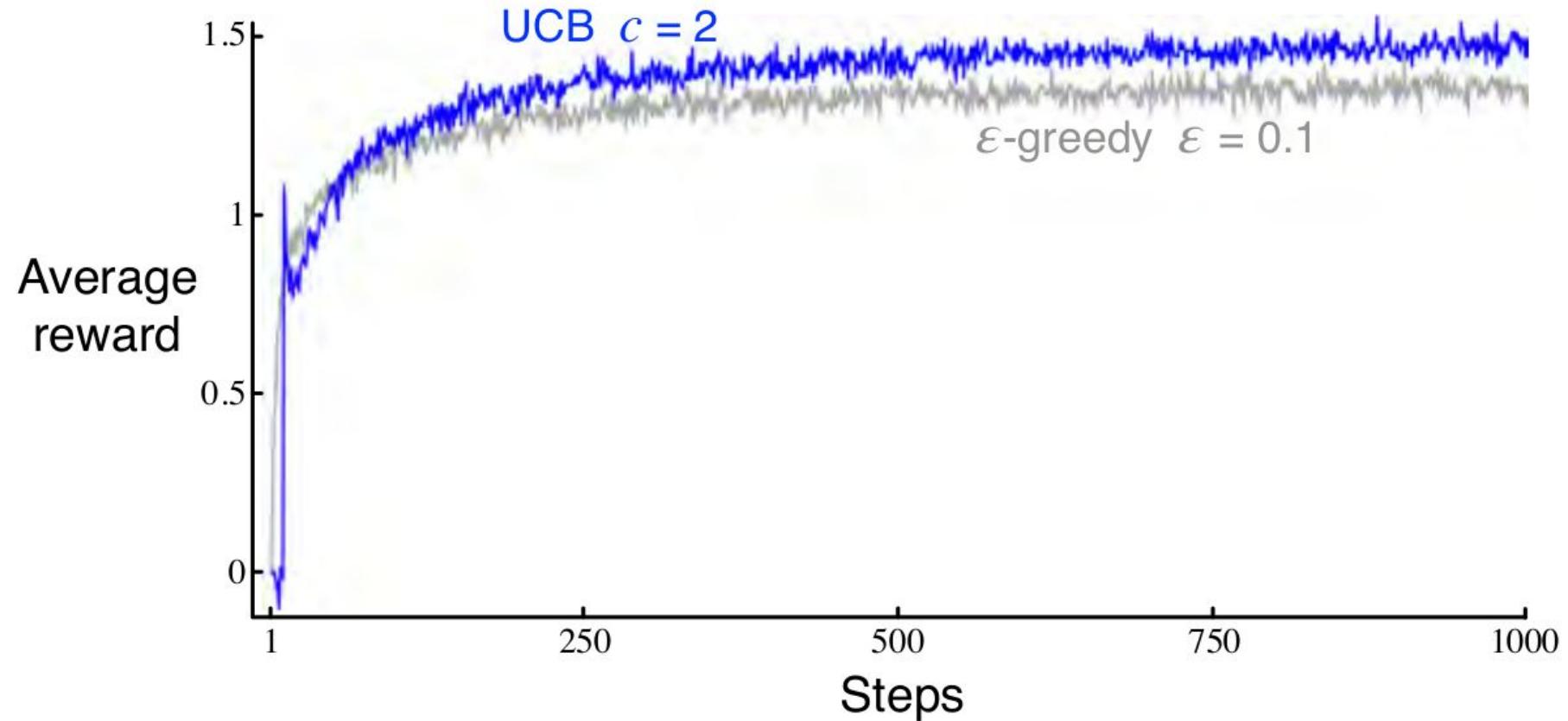
$$A_t \doteq \arg \max_a \left[Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right]$$

Upper-Confidence-Bound Action Selection

- Each time a is selected the uncertainty is presumably reduced
- Each time an action other than a is selected, t increases but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.
- Actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time



Upper-Confidence-Bound Action Selection



UCB often performs well, as shown here, but is more difficult than " ϵ -greedy" to extend beyond bandits to the more general reinforcement learning settings

Policy-based algorithms

- Forget about action-value (Q) estimates, we don't really care about them
- We care about what actions to choose
 - Let's assign a preference to each action and tweak its value
- Define $H_t(a)$ as a numerical preference value associated with action a
- Which action is selected?
 - $A_t = \underset{a}{\operatorname{argmax}}[H_t(a)]$
 - Hardmax results in no exploration -- deterministic action selection



Softmax function

- **Input:** vector of preferences

- **Output:** vector of probabilities forming a valid distribution

- $\Pr\{a_t = a\} = \frac{e^{H_t(a)}}{\sum_{a' \in A} e^{H_t(a')}} = \Pr(a)$

- $H_t \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{bmatrix} 6 \\ 9 \\ 2 \end{bmatrix}$

- $\text{softmax} \begin{pmatrix} 6 \\ 9 \\ 2 \end{pmatrix} = \begin{bmatrix} 0.047 \\ 0.952 \\ 0.00087 \end{bmatrix}$

- That is, with $\Pr(0.95)$ choose a_2 , $\Pr(0.05)$ choose a_1 , and $< \Pr(0.01)$ choose a_3

Softmax function

- Exploration – checked!
- Softmax provides another important attribute – a **differentiable policy**
- Say that we learn that action a_1 results in good relative performance
- Hardmax: $A_t = \underset{a}{\operatorname{argmax}}[H_t(a_1), H_t(a_2), H_t(a_3)]$
 - Change $H(a_1)$ such that $\Pr(a_1)$ is increased, $\frac{\partial \Pr(a_1)}{\partial H(a_1)} = NA$
- Softmax: $\Pr(a_1) = \frac{e^{H_t(a)}}{\sum_{a' \in A} e^{H_t(a')}}$
 - Change $H(a_1)$ such that $\Pr(a_1)$ is increased -> update towards $\frac{\partial \Pr(a_1)}{\partial H(a_1)}$

Gradient ascend

- $H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)$ $\frac{\partial \Pr(A_t)}{\partial H(A_t)}$?
- $\forall a \neq A_t, H_{t+1}(a) = H_t(a) + \alpha(R_t - \bar{R}_t)$ $\frac{\partial \Pr(a)}{\partial H(a)}$
- Update the preferences based on observed reward and a baseline reward (\bar{R}_t)
- If the observed reward is larger than the baseline:
 - Increase the preference of the chosen action, A_t
 - Decrease the preference of all other actions, $\forall a \neq A_t$
- Else do the opposite

Update Rule

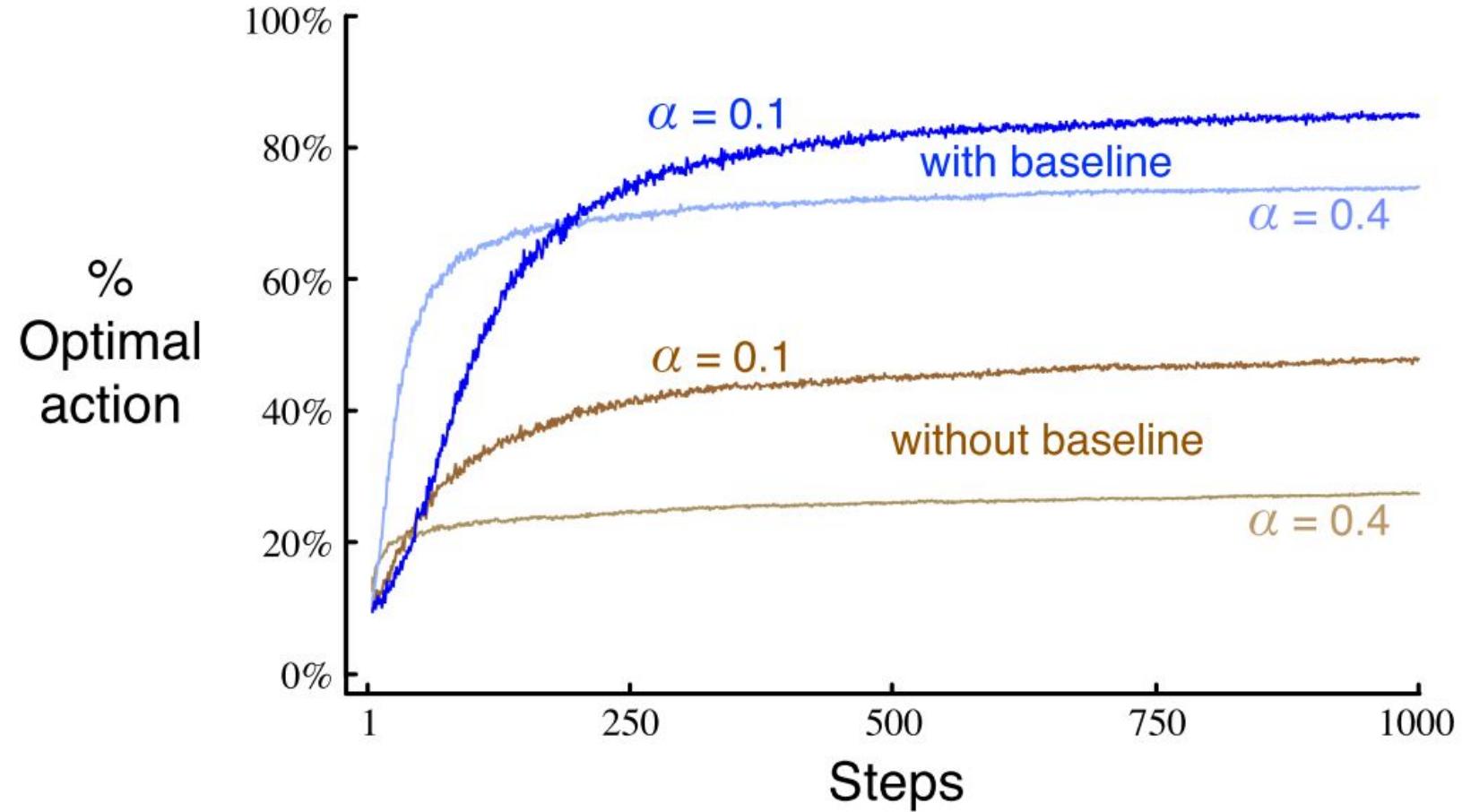
On each step, after selecting action A_t and receiving the reward R_t ,
Update the action preferences :

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \Pr(A_t))$$

$$\forall a \neq A_t, H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t) \Pr(a)$$

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), \quad \text{and}$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), \quad \text{for all } a \neq A_t$$



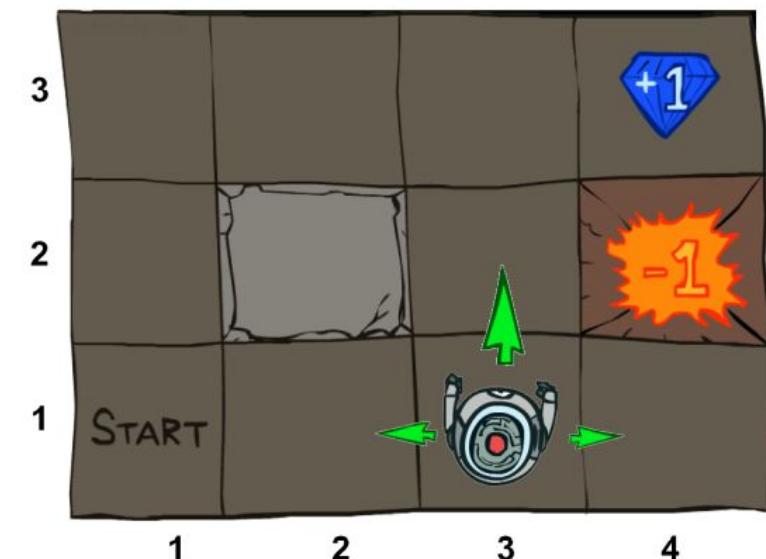


What did we learn?

- **Problem:** choose the action that results in highest expected reward
- **Assumptions:** 1. actions' expected reward is unknown, 2. we are confronted with the same problem over and over, 3. we are able to observe an action's outcome once chosen
- **Approach:** learn the actions' expected reward through exploration (value based) or learn a policy directly (policy based), exploit learnt knowledge to choose best action
- **Methods:** 1. greedy + initializing estimates optimistically, 2. epsilon-greedy, 3. Upper-Confidence-Bounds, 4. gradient ascend + soft-max

A different scenario

- Associative vs. Non-associative tasks ?
- Policy: A mapping from situations to the actions that are best in those situations
- (discuss) How do we extend the solution for non-associative task to an associative task?
 - **Approach**: Extend the solutions to non-stationary task to non-associative tasks
 - Works, if the true action values changes slowly
 - What if the context switching between the situations are made explicit?
 - How?
 - Need Special approaches !!!





Required Readings

1. Chapter-2 of Introduction to Reinforcement Learning, 2nd Ed., Sutton & Barto
2. A Survey on Practical Applications of Multi-Armed and Contextual Bandits, Djallel Bouneffouf , Irina Rish [<https://arxiv.org/pdf/1904.10040.pdf>]



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you !



Session #3: Markov Decision Processes (1/2)

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)

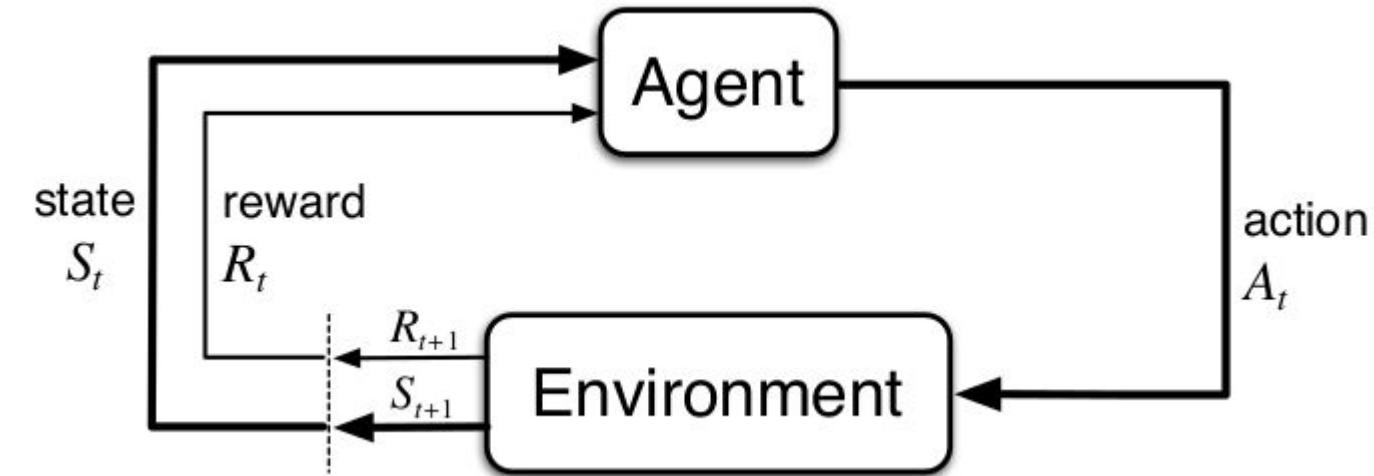


Agenda for the class

- Agent-Environment Interface (Sequential Decision Problem)
- MDP

Agent-Environment Interface

- **Agent** - Learner & the decision maker
- **Environment** - Everything outside the agent
- **Interaction:**
 - Agent performs an action
 - Environment responds by
 - presenting a new situation (change in state)
 - presents numerical reward
- **Objective (of the interaction):**
 - Maximize the return (cumulative rewards) over time

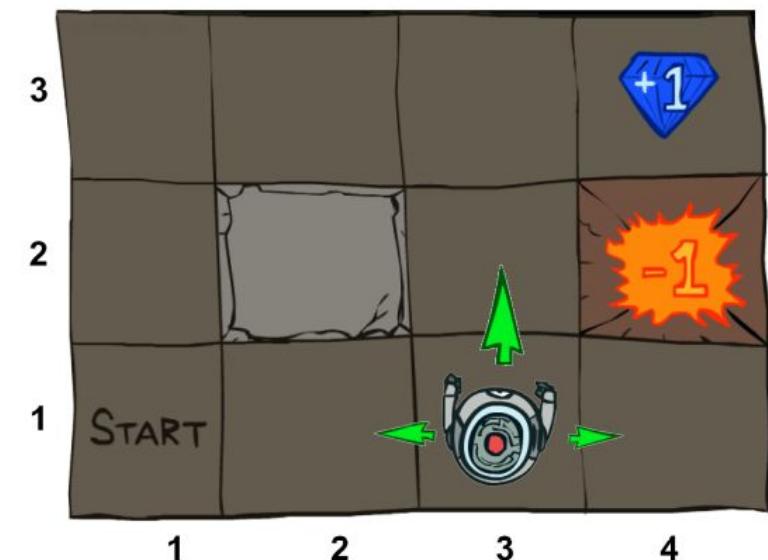


Note:

- Interaction occurs in discrete time steps
- $$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

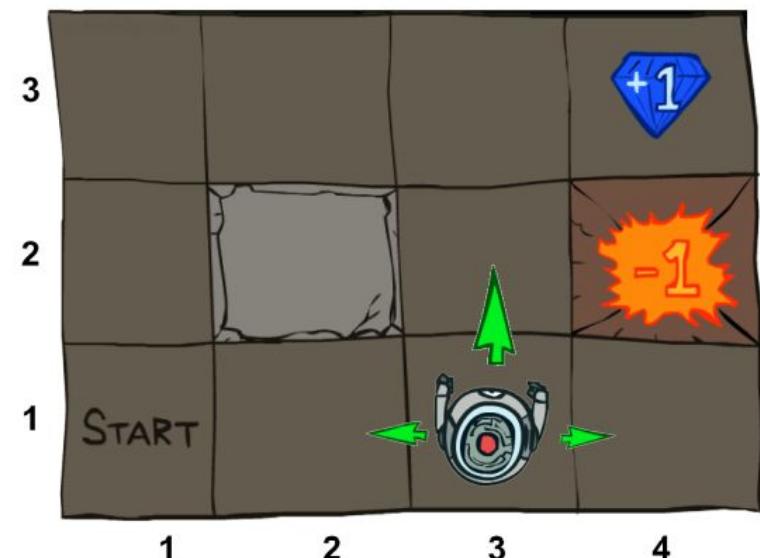
Grid World Example

- A maze-like problem
 - The agent lives in a grid
 - Walls block the agent's path
- Noisy movement: actions do not always go as planned
 - 80% of the time, the action North takes the agent North (if there is no wall there)
 - 10% of the time, North takes the agent West; 10% East
 - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
 - -0.1 per step (battery loss)
 - +1 if arriving at (4,3) ; -1 for arriving at (4,2) ;1 for arriving at (2,2)
- Goal: maximize accumulated rewards



Markov Decision Processes

- An MDP is defined by
 - A set of **states**
 - A set of **actions**
 - **State-transition probabilities**
 - Probability of arriving to after performing at
 - Also called the **model dynamics**
 - **A reward function**
 - The utility gained from arriving to after performing at
 - Sometimes just or even
 - **A start state**
 - **Maybe a terminal state**





Markov Decision Processes

Model Dynamics

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

State-transition probabilities

$$p(s' | s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

Expected rewards for state-action-next-state triples

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$



Markov Decision Processes - Discussion

- *MDP framework is abstract and flexible*
 - Time steps need not refer to fixed intervals of real time
 - The actions can be
 - at low-level controls or high-level decisions
 - totally mental or computational
 - States can take a wide variety of forms
 - Determined by *low-level sensations* or *high-level and abstract* (ex. symbolic descriptions of objects in a room)
- *The agent–environment boundary represents the limit of the agent's absolute control*, not of its knowledge.
 - *The boundary can be located at different places for different purposes*

Markov Decision Processes - Discussion

- *MDP framework is a considerable abstraction of the problem of goal-directed learning from interaction.*
- It proposes that *whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between an agent and its environment:*
 - *one signal to represent the choices made by the agent (the actions)*
 - *one signal to represent the basis on which the choices are made (the states),*
 - *and one signal to define the agent's goal (the rewards).*

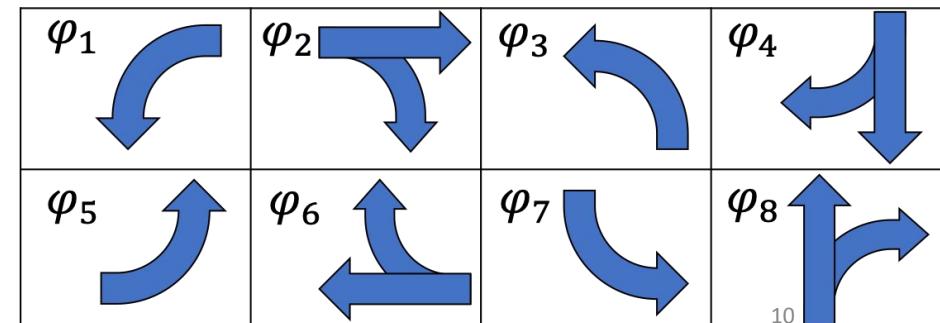
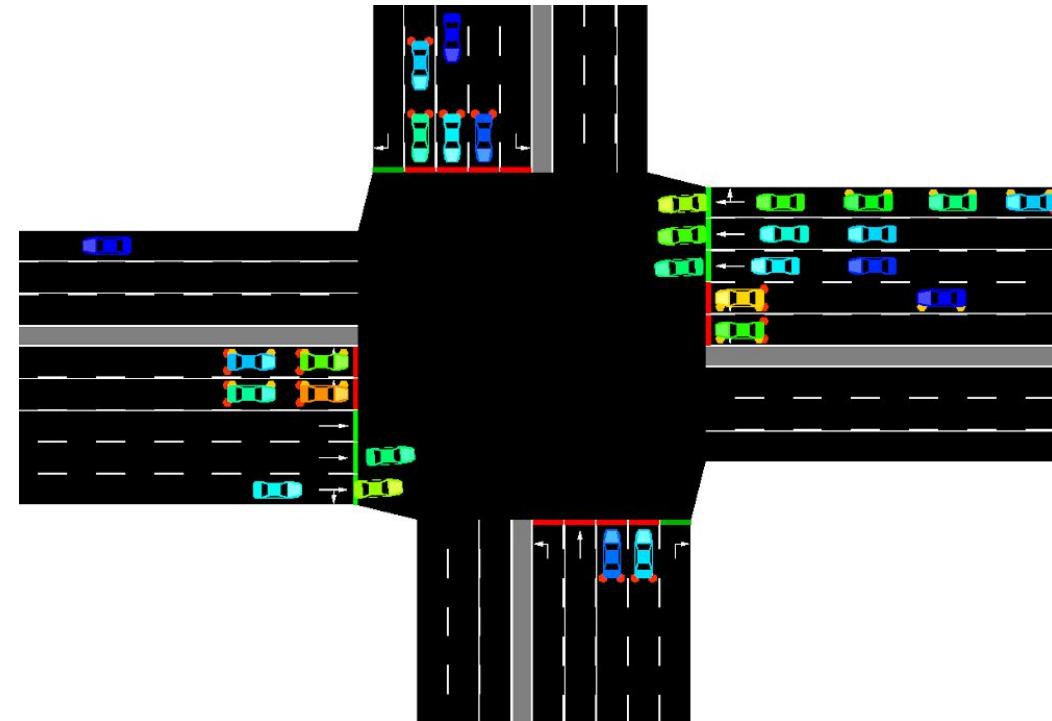
MDP Formalization : Video Games

- *State:*
 - raw pixels
- *Actions:*
 - game controls
- *Reward:*
 - change in score
- *State-transition probabilities:*
 - defined by stochasticity in game evolution



MDP Formalization : Traffic Signal Control

- ***State:***
 - Current signal assignment (green, yellow, and red assignment for each phase)
 - For each lane: number of approaching vehicles, accumulated waiting time, number of stopped vehicles, and average speed of approaching vehicles
- ***Actions:***
 - signal assignment
- ***Reward:***
 - Reduction in traffic delay
- ***State-transition probabilities:***
 - defined by stochasticity in approaching demand



MDP Formalization : Recycling Robot (Detailed Ex.)

- *Robot has*
 - *sensors for detecting cans*
 - *arm and gripper that can pick the cans and place in an onboard bin;*
- *Runs on a rechargeable battery*
- *Its control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper*
- **Task for the RL Agent:** *Make high-level decisions about how to search for cans based on the current charge level of the battery*



MDP Formalization : Recycling Robot (Detailed Ex.)

- ***State:***
 - Assume that only two charge levels can be distinguished
 - $S = \{\text{high}, \text{low}\}$
- ***Actions:***
 - $A(\text{high}) = \{\text{search}, \text{wait}\}$
 - $A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$
- ***Reward:***
 - Zero most of the time, except when securing a can
 - Cans are secured by searching and waiting, but $r_{\text{search}} > r_{\text{wait}}$
- ***State-transition probabilities:***
 - [Next Slide]



MDP Formalization : Recycling Robot (Detailed Ex.)

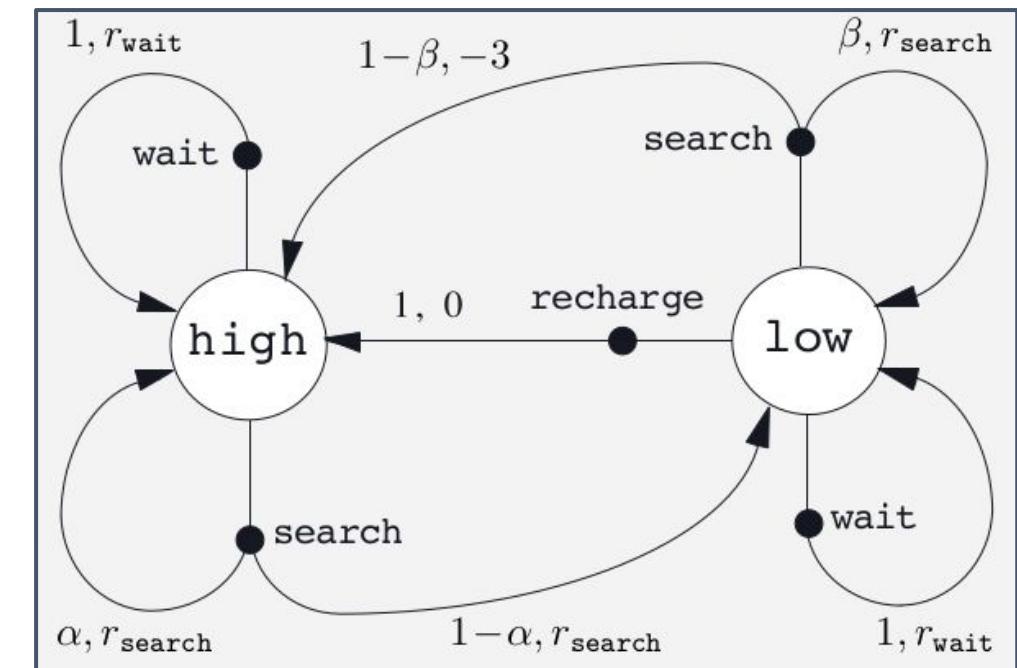
- *State-transition probabilities (contd...):*

s	a	s'	$p(s' s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-

MDP Formalization : Recycling Robot (Detailed Ex.)

- State-transition probabilities (contd...):*

s	a	s'	$p(s' s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-



Note on Goals & Rewards

- Reward Hypothesis:

That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

- *The rewards we set up truly indicate what we want accomplished,*
 - *not the place to impart prior knowledge on how we want it to do*
- *Ex: Chess Playing Agent*
 - *If the agent is rewarded for taking opponents pieces, the agent might fall for the opponent's trap.*
- *Ex: Vacuum Cleaner Agent*
 - *If the agent is rewarded for each unit of dirt it sucks, it can repeatedly deposit and suck the dirt for larger reward*

Returns & Episodes

- *Goal is to maximize the expected return*
- *Return (G_t) is defined as some specific function of the reward sequence*
- *When there is a notion of final time step, say T , return can be*

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

- *Applicable when agent-environment interaction breaks into episodes*
- *Ex: Playing Game, Trips through maze etc. [called episodic tasks]*

Returns & Episodes

- Goal is to maximize the expected return
- Return (G_t) is defined as some specific function of the reward sequence
- When there is a notion of final time step, say T , return can be

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

- Applicable when agent-environment interaction breaks into episodes
- Ex: Playing Game, Trips through maze etc. [called episodic tasks]

- Generally $T = \infty$
 - What if the agent receive a reward of +1 for each timestep?
 - Discounted Return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Note: γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate.

Returns & Episodes

- *What if γ is 0?*
- *What if γ is 1?*
- *Computing discounted rewards incrementally*

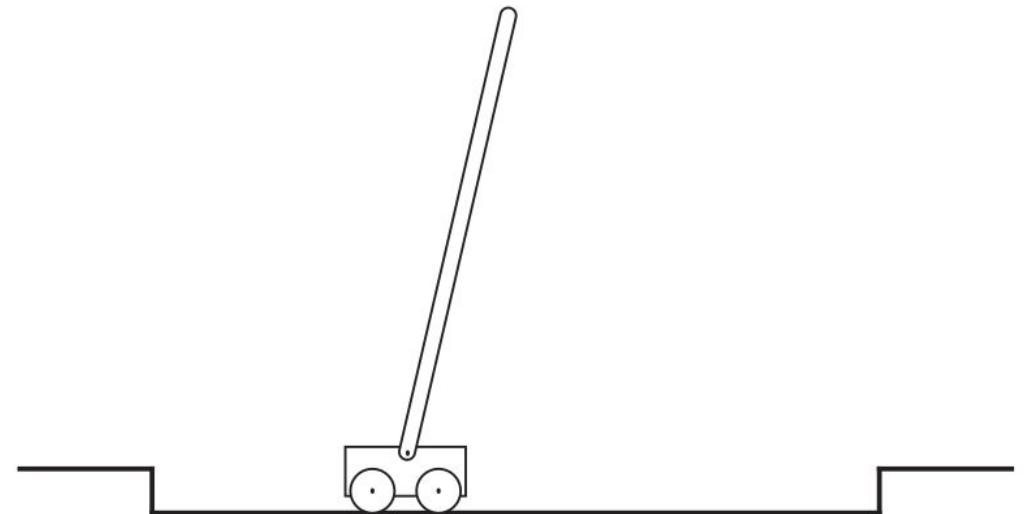
$$\begin{aligned}G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\&= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\&= R_{t+1} + \gamma G_{t+1}\end{aligned}$$

- *Sum of an infinite number of terms, it is still finite if the reward is nonzero and constant and if $\gamma < 1$.*
- *Ex: reward is +1 constant*

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1 - \gamma}.$$

Returns & Episodes

- **Objective:** *To apply forces to a cart moving along a track so as to keep a pole hinged to the cart from falling over*
- **Discuss:**
 - *Consider the task as episodic, that is try/maintain balance until failure.
What could be the reward function?*
 - *Repeat prev. assuming task is continuous.*





Next Class

Solving MDP

Refresh Dynamic Programming !!!

Teaching Assistants

Partha Pratim Saha {parthapratim@wilp.bits-pilani.ac.in}

P. Anusha {anusha.p@wilp.bits-pilani.ac.in}



Required Readings

1. Chapter-3 of Introduction to Reinforcement Learning, 2nd Ed., Sutton & Barto



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Reinforcement Learning
2022-23 Second Semester, M.Tech (AIML)

Session #4: Markov Decision Processes

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)



Agenda for the class

- Agent-Environment Interface (Sequential Decision Problem)
- MDP
 - Defining MDP,
 - Rewards,
 - Returns, Policy & Value Function,
 - Optimal Policy and Value Functions
- Approaches to solve MDP

Announcement !!!

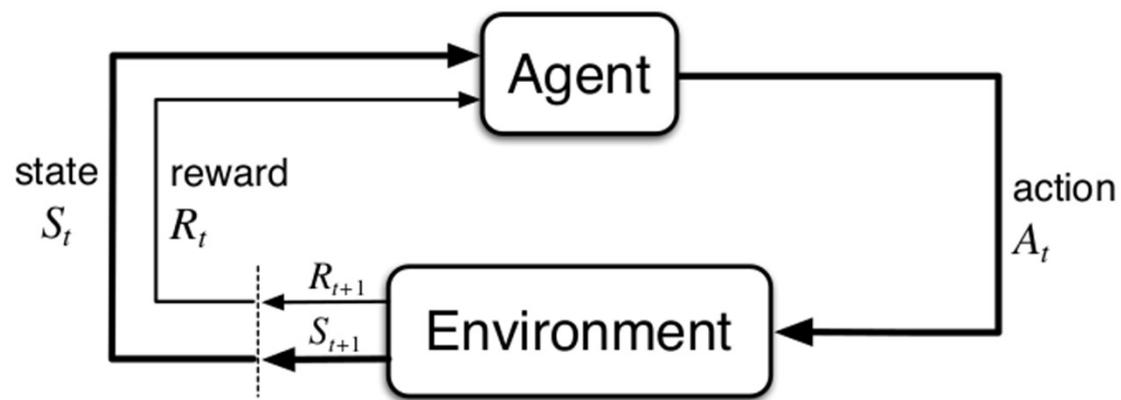
We have our Teaching Assistants now !!!

1. Partha Pratim Saha {parthapratim@wilp.bits-pilani.ac.in}
2. P. Anusha {anusha.p@wilp.bits-pilani.ac.in}



Agent-Environment Interface

- **Agent** - Learner & the decision maker
- **Environment** - Everything outside the agent
- **Interaction:**
 - Agent performs an action
 - Environment responds by
 - presenting a new situation (change in state)
 - presents numerical reward
- **Objective (of the interaction):**
 - Maximize the return (cumulative rewards) over time



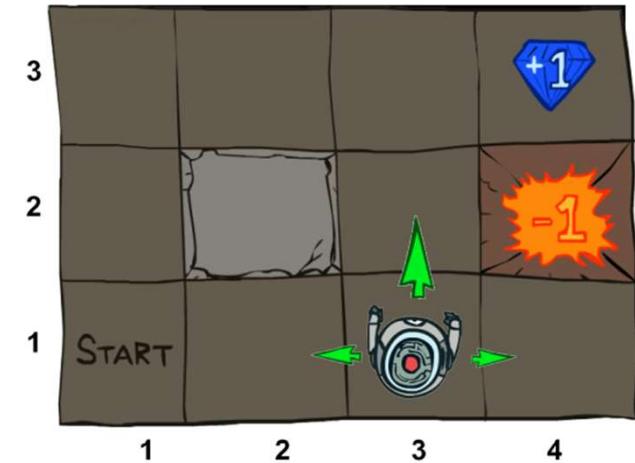
Note:

- Interaction occurs in discrete time steps
- $$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$



Grid World Example

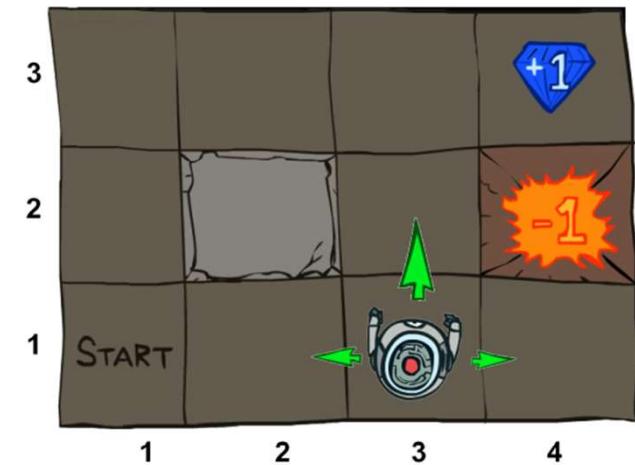
- A maze-like problem
 - The agent lives in a grid
 - Walls block the agent's path
- Noisy movement: actions do not always go as planned
 - 80% of the time, the action North takes the agent North (if there is no wall there)
 - 10% of the time, North takes the agent West; 10% East
 - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
 - -0.1 per step (battery loss)
 - +1 if arriving at (4,3) ; -1 for arriving at (4,2) ; 1 for arriving at (2,2)
- Goal: maximize accumulated rewards





Markov Decision Processes

- An MDP is defined by
 - A set of **states**
 - A set of **actions**
 - **State-transition probabilities**
 - Probability of arriving to after performing at
 - Also called the **model dynamics**
 - **A reward function**
 - The utility gained from arriving to after performing at
 - Sometimes just or even
 - **A start state**
 - **Maybe a terminal state**





Markov Decision Processes

Model Dynamics

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

State-transition probabilities

$$p(s' | s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

Expected rewards for state–action–next-state triples

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$



Markov Decision Processes - Discussion

- *MDP framework is abstract and flexible*
 - **Time steps** need not refer to fixed intervals of real time
 - The **actions** can be
 - at low-level controls or high-level decisions
 - totally mental or computational
 - **States** can take a wide variety of forms
 - Determined by *low-level sensations* or *high-level and abstract* (ex. symbolic descriptions of objects in a room)
- *The agent–environment boundary represents the limit of the agent's absolute control*, not of its knowledge.
 - *The boundary can be located at different places for different purposes*



Markov Decision Processes - Discussion

- *MDP framework is a considerable abstraction of the problem of goal-directed learning from interaction.*
- It proposes that *whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between an agent and its environment:*
 - *one signal to represent the choices made by the agent (the actions)*
 - *one signal to represent the basis on which the choices are made (the states),*
 - *and one signal to define the agent's goal (the rewards).*



MDP Formalization : Video Games

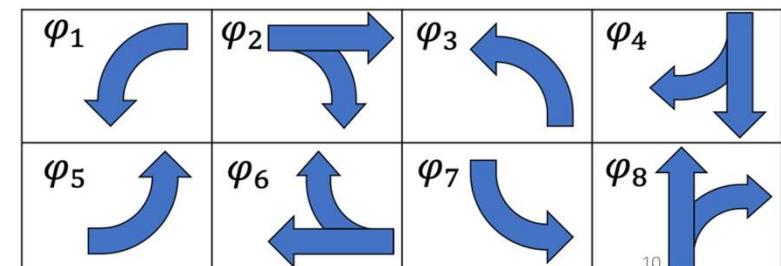
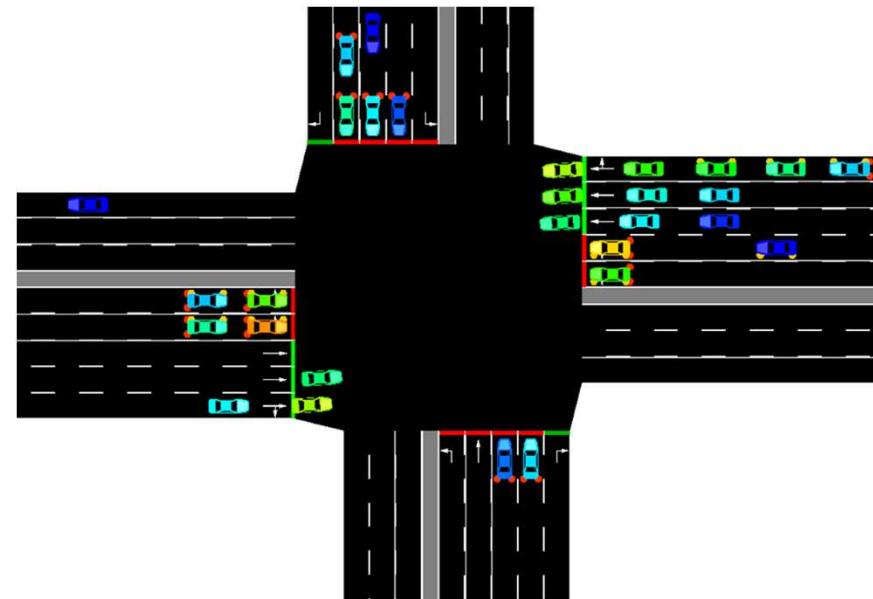
- *State:*
 - raw pixels
- *Actions:*
 - game controls
- *Reward:*
 - change in score
- *State-transition probabilities:*
 - defined by stochasticity in game evolution



MDP Formalization : Traffic Signal Control

- ***State:***
 - Current signal assignment (green, yellow, and red assignment for each phase)
 - For each lane: number of approaching vehicles, accumulated waiting time, number of stopped vehicles, and average speed of approaching vehicles
- ***Actions:***
 - signal assignment
- ***Reward:***
 - Reduction in traffic delay
- ***State-transition probabilities:***
 - defined by stochasticity in approaching demand

Ref: "Learning an Interpretable Traffic Signal Control Policy", Ault et al., 2020



MDP Formalization : Recycling Robot (Detailed Ex.)

- Robot has
 - *sensors for detecting cans*
 - *arm and gripper that can pick the cans and place in an onboard bin;*
- *Runs on a rechargeable battery*
- *Its control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper*
- **Task for the RL Agent:** *Make high-level decisions about how to search for cans based on the current charge level of the battery*



MDP Formalization : Recycling Robot (Detailed Ex.)

- ***State:***
 - Assume that only two charge levels can be distinguished
 - $S = \{\text{high}, \text{low}\}$
- ***Actions:***
 - $A(\text{high}) = \{\text{search}, \text{wait}\}$
 - $A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$
- ***Reward:***
 - Zero most of the time, except when securing a can
 - Cans are secured by searching and waiting, but $r_{\text{search}} > r_{\text{wait}}$
- ***State-transition probabilities:***
 - [Next Slide]





MDP Formalization : Recycling Robot (Detailed Ex.)

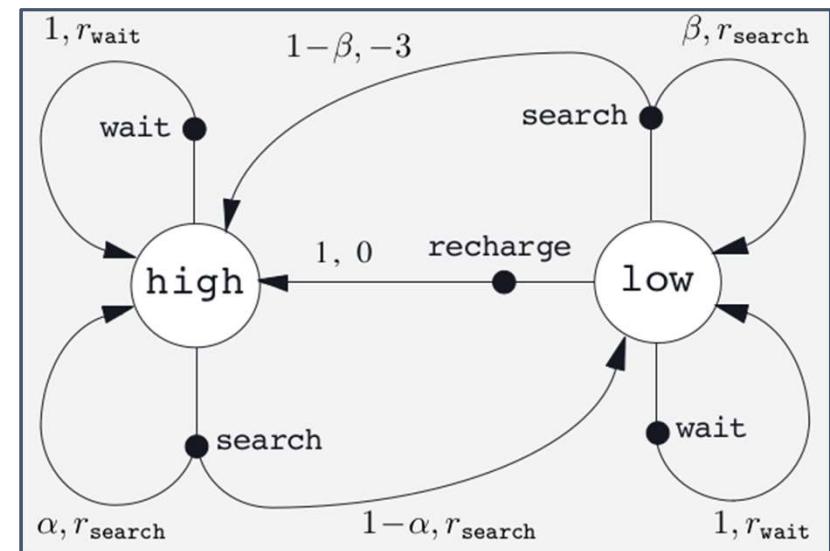
- *State-transition probabilities (contd...):*

s	a	s'	$p(s' s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-

MDP Formalization : Recycling Robot (Detailed Ex.)

- State-transition probabilities (contd...):*

s	a	s'	$p(s' s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-





Note on Goals & Rewards

- Reward Hypothesis:

All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called **reward**).

- The rewards we set up truly indicate what we want accomplished,
 - not the place to impart prior knowledge on how we want it to do
- Ex: **Chess Playing Agent**
 - If the agent is rewarded for taking opponents pieces, the agent might fall for the opponent's trap.
- Ex: **Vacuum Cleaner Agent**
 - If the agent is rewarded for each unit of dirt it sucks, it can repeatedly deposit and suck the dirt for larger reward



Returns & Episodes

- Goal is to maximize the expected return
- Return (G_t) is defined as some specific function of the reward sequence
- Episodic tasks vs. Continuing tasks
- When there is a notion of final time step, say T , return can be

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

- Applicable when agent-environment interaction breaks into episodes
- Ex: Playing Game, Trips through maze etc. [called episodic tasks]

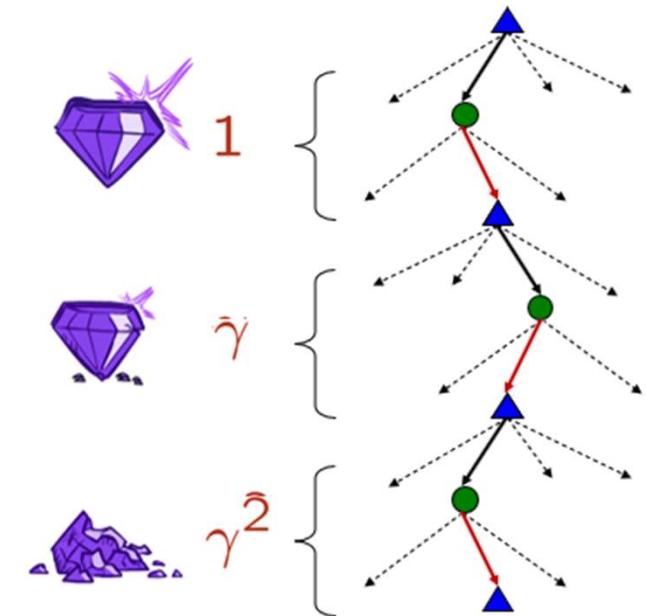
Returns & Episodes

- Generally $T = \infty$
 - What if the agent receives a reward of +1 for each timestep?
 - Discounted Return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Note: γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount rate*.

- Discount rate determines the present value of future rewards





Returns & Episodes

- *What if γ is 0?*
- *What if γ is 1?*
- *Computing discounted rewards incrementally*

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

- *Sum of an infinite number of terms, it is still finite if the reward is nonzero and constant and if $\gamma < 1$.*
- *Ex: reward is +1 constant*

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1 - \gamma}.$$

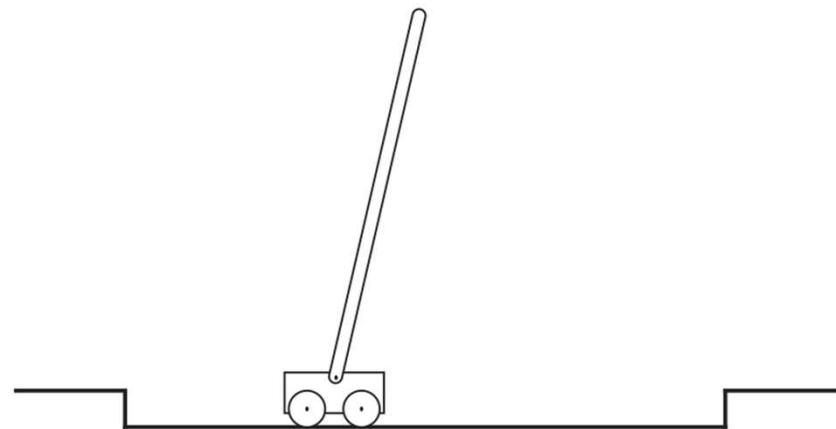


Returns & Episodes

→ **Objective:** *To apply forces to a cart moving along a track so as to keep a pole hinged to the cart from falling over*

→ **Discuss:**

- Consider the task as episodic, that is try/maintain balance until failure.
What could be the reward function?
- Repeat prev. assuming task is continuous.





Policy

- A mapping from states to probabilities of selecting each possible action.
 - $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$
- The purpose of learning is to improve the agent's policy with its experience





Defining Value Functions

State-value function for policy π

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \text{ for all } s \in \mathcal{S},$$

Action-value function for policy π

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$



Defining Value Functions

State Value function in terms of Action-value function for policy π

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) q_\pi(s, a)$$

Action Value function in terms of State value function for policy π

$$q_\pi(s, a) = \sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) [r + \gamma v_\pi(s')]$$



Bellman Equation for V_π

May skip to the next slide !

- Dynamic programming equation associated with discrete-time optimization problems
 - Expressing V_π recursively i.e. relating $V_\pi(s)$ to $V_\pi(s')$ for all $s' \in \text{succ}(s)$

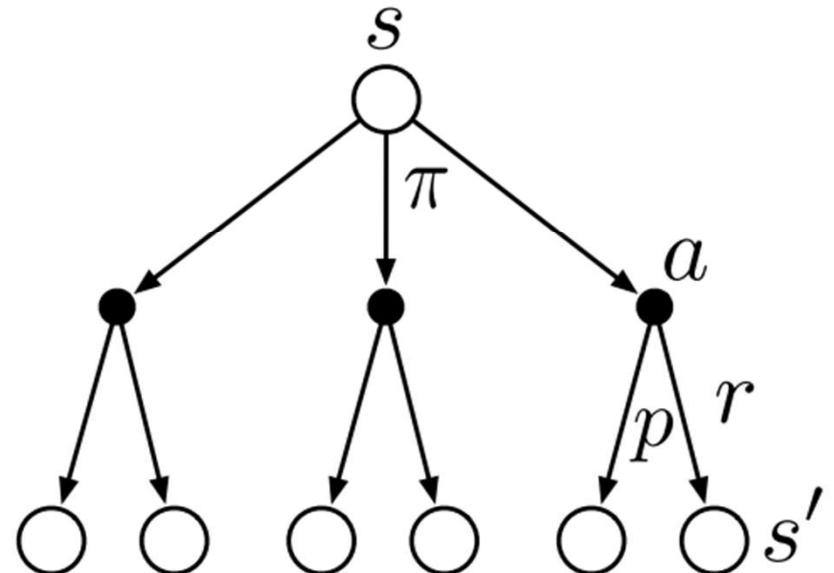
$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_\pi(s') \right], \quad \text{for all } s \in \mathcal{S}. \end{aligned}$$



Bellman Equation for V_π

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')]$$

Value of the start state must equal
**(1) the (discounted) value of the expected next state,
plus**
(1) the reward expected along the way

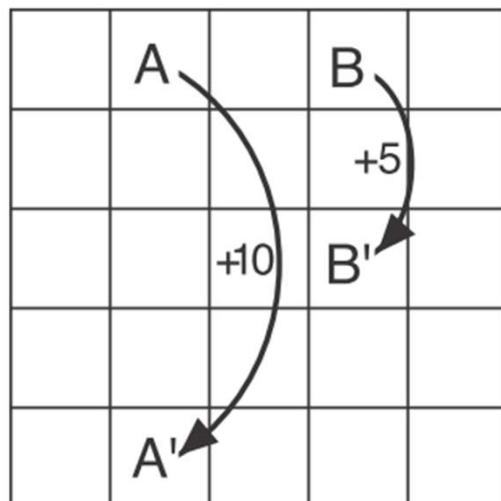


Backup Diagram

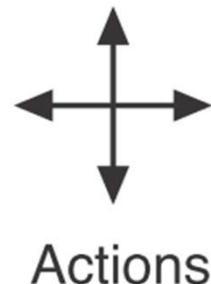
Understanding $V_{\pi}(s)$ with Gridworld

Reward:

- -1 if an action takes agent off the grid
- Exceptional reward from A and B for all actions taking agent to A' and B' resp.
- 0, everywhere else



Exceptional reward dynamics



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

State-value function for the equiprobable random policy with $\gamma = 0.9$



Understanding $V_\pi(s)$ with Gridworld

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_\pi(s') \right].$$

Verify $V_\pi(s)$ using Bellman equation for this state with $\gamma = 0.9$, and equiprobable random policy

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0



Understanding $V_\pi(s)$ with Gridworld

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')]$$

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')] \\ &= \sum_a 0.25 \cdot [0 + 0.9 \cdot (2.3 + 0.4 - 0.4 + 0.7)] \\ &= 0.25 \cdot [0.9 \cdot 3.0] = 0.675 \approx 0.7 \end{aligned}$$

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0



Ex-1

Recollect the reward function used for Gridworld as below:

- -1 if an action takes agent off the grid
- Exceptional reward from A and B for all actions taking agent to A' and B' resp.
- 0, everywhere else

Let us add a constant c (say 10) to the rewards of all the actions. Will it change anything?



Optimal Policies and Optimal Value Functions

- $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$
- There is always at least one policy that is better than or equal to all other policies \rightarrow optimal policy (denoted as π_*)
 - There could be more than one optimal policy !!!

Optimal state-value function $v_*(s) \doteq \max_{\pi} v_{\pi}(s)$.

Optimal action-value function $q_*(s, a) \doteq \max_{\pi} q_{\pi}(s, a)$

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

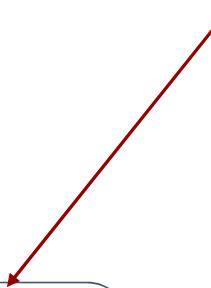


Optimal Policies and Optimal Value Functions

Bellman optimality equation - expresses that *the value of a state under an optimal policy must equal the expected return for the best action from that state*

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')]. \end{aligned}$$

Bellman optimality equation for V_*





Optimal Policies and Optimal Value Functions

Bellman optimality equation - expresses that *the value of a state under an optimal policy must equal the expected return for the best action from that state*

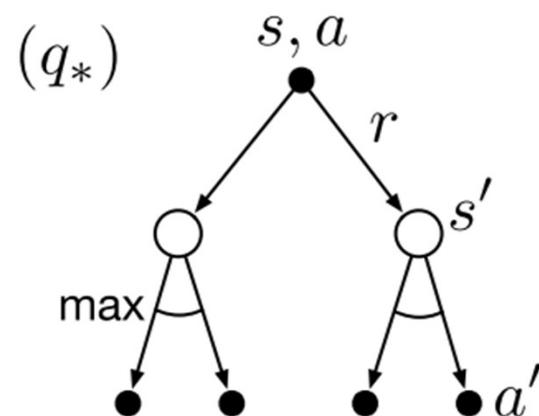
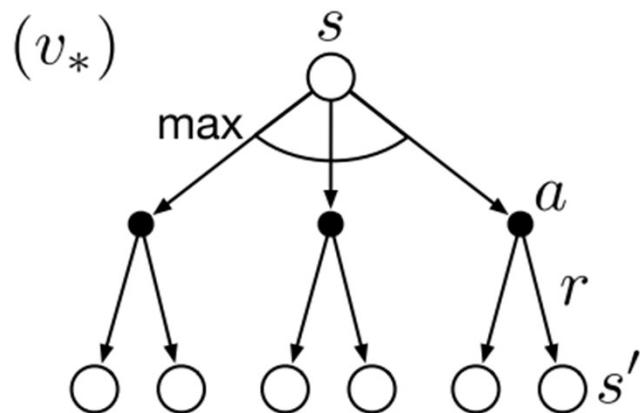
Bellman optimality equation for q_*

$$\begin{aligned} q_*(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\ &= \sum_{s', r} p(s', r | s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]. \end{aligned}$$



Optimal Policies and Optimal Value Functions

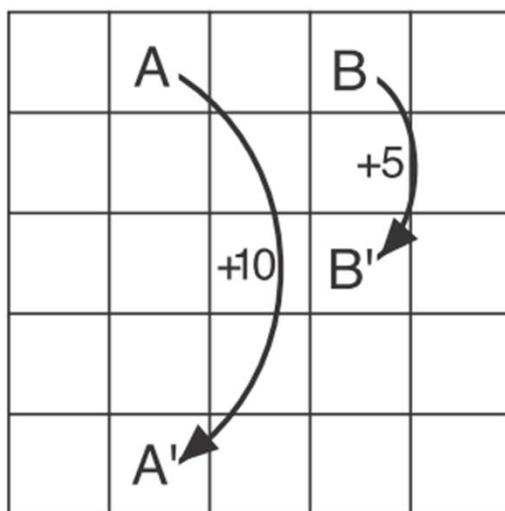
Bellman optimality equation - expresses that *the value of a state under an optimal policy must equal the expected return for the best action from that state*



Backup diagrams for v^* and q^*



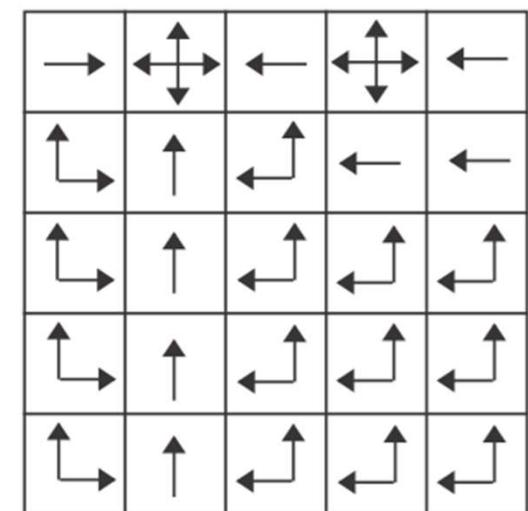
Optimal solutions to the gridworld example



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

Backup diagrams for v^* and q^*



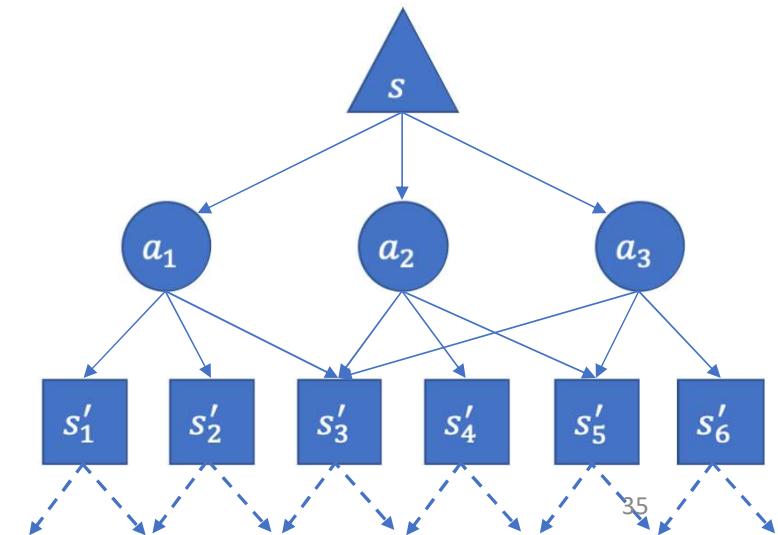
Break for 5 mins

MDP - Objective

A set of states $s \in \mathcal{S}$
 A set of actions $a \in \mathcal{A}$
 State-transition probabilities $P(s'|s, a)$
 A reward function $R(s, a, s')$

- Compute a policy: what action to take at each state
 - $\pi: S \rightarrow A$
- Compute the **optimal** policy: maximum expected reward, π^*
- $\pi^*(s) = ?$
- $= \underset{a}{\operatorname{argmax}} [\sum_{s'} P(s'|s, a) R(s, a, s')]$
 - Must also optimize over the future (next steps)
- $= \underset{a}{\operatorname{argmax}} [\sum_{s'} P(s'|s, a) (R(s, a, s') + \mathbb{E}_{\pi^*}[G|s'])]$

$$v^*(s') = \sum_{s'} P(s'|s) (R(s, a, s') + \mathbb{E}_{\pi^*}[G|s'])$$





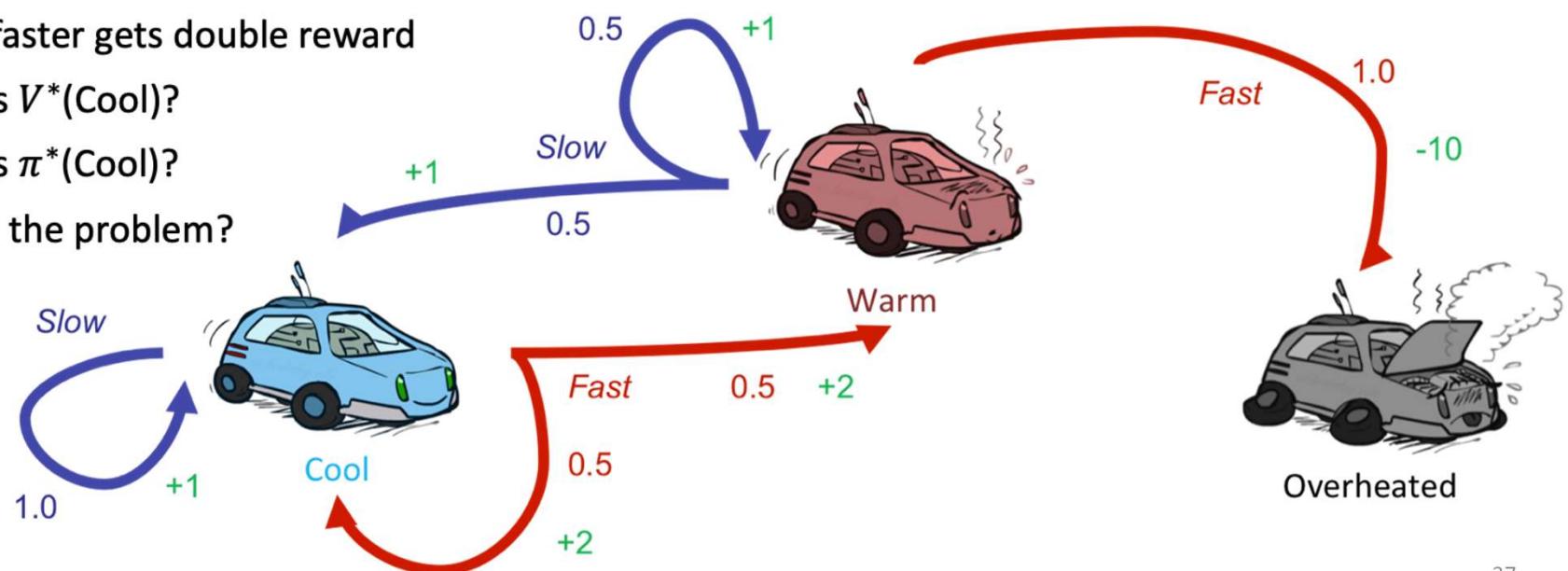
Notation

- π^* - a policy that yields the maximal expected sum of rewards
- G - observed sum of rewards, i.e., $\sum r_t$
- $v^*(s)$ - the expected sum of rewards from being at s then following π^*
 - $= \mathbb{E}_{\pi^*} [G|s]$

Race car example

- A robot car wants to travel far, quickly
 - Three states: **Cool**, **Warm**, Overheated
 - Two actions: *Slow*, *Fast*
 - Going faster gets double reward
 - What is $V^*(\text{Cool})$?
 - What is $\pi^*(\text{Cool})$?
 - What's the problem?

$$\max_a \left[\sum_{s'} P(s'|s, a) (R(s, a, s') + v^*(s')) \right]$$



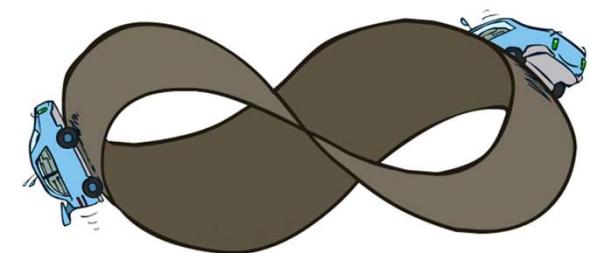


Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
 - Finite horizon: (similar to depth-limited search)
 - Terminate episodes after a fixed T steps (e.g. life)
 - Gives nonstationary policies (π depends on time left)
 - Discounting: use $0 < \gamma < 1$

$$U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1 - \gamma)$$

- Smaller γ means smaller “horizon” – shorter term focus





Discount factor

- As the agent traverse the world it receives a sequence of rewards
- Which sequence has higher utility?
 - $\tau_1 = +1, +1, +1, +1, +1, +1\dots$ $\sum_{t=0}^{\infty} 1 = \sum_{t=0}^{\infty} 2 = \infty$
 - $\tau_2 = +2, +2, +2, +2, +2, +2\dots$
- Let's decay future rewards exponentially by a factor, $0 \leq \gamma < 1$

$$\sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1 - \gamma} \quad \text{Geometric series}$$

- Now τ_1 yields higher utility than τ_2



Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- Discount factor: values of rewards decay exponentially



1

Worth Now



γ

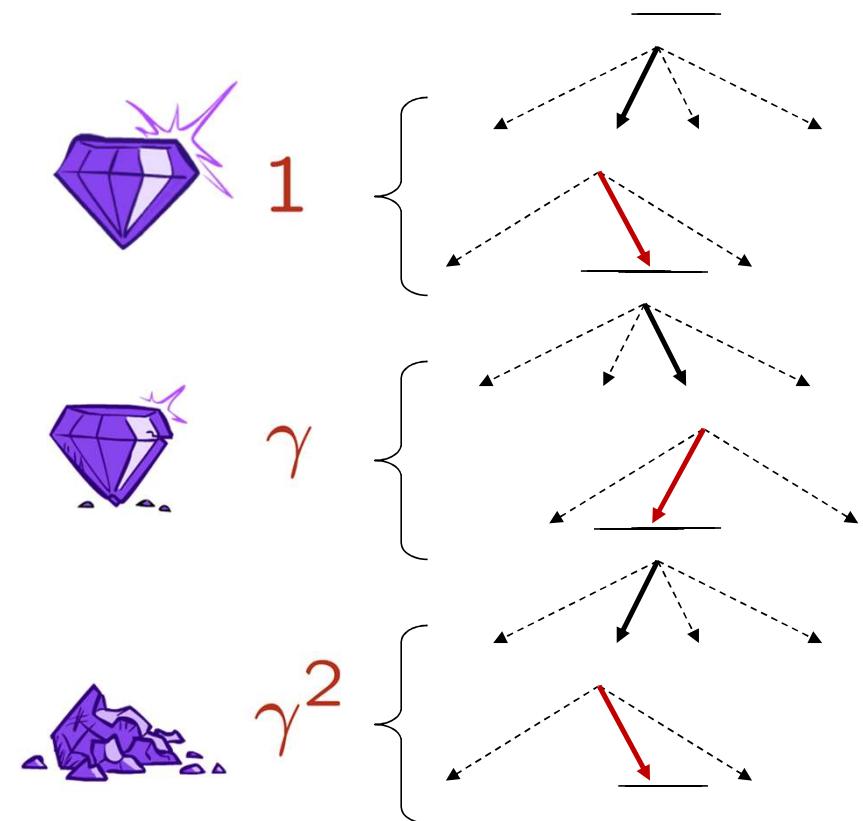


γ^2

Worth In Two Steps

Discounting

- How to discount?
 - Each time we descend a level, we multiply in the discount once
- Why discount?
 - Sooner rewards probably do have higher utility than later rewards
 - Also helps our algorithms converge
- Example: discount of 0.5
 - $G(r=[1,2,3]) = 1*1 + 0.5*2 + 0.25*3$
 - $G([1,2,3]) < G([3,2,1])$





Quiz: Discounting

- Given grid world:

10				1
a	b	c	d	e

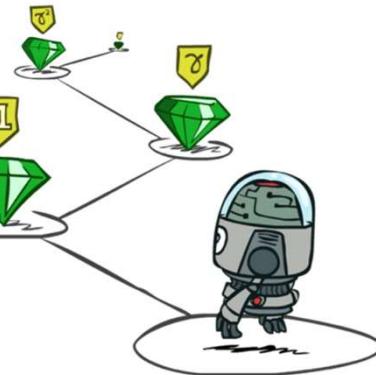
- Actions: East, West, and Exit ('Exit' only available in terminal states: a, e)
- Rewards are given only after an exit action
- Transitions: deterministic
- Quiz 1: For $\gamma = 1$, what is the optimal policy?

10				1
----	--	--	--	---

- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

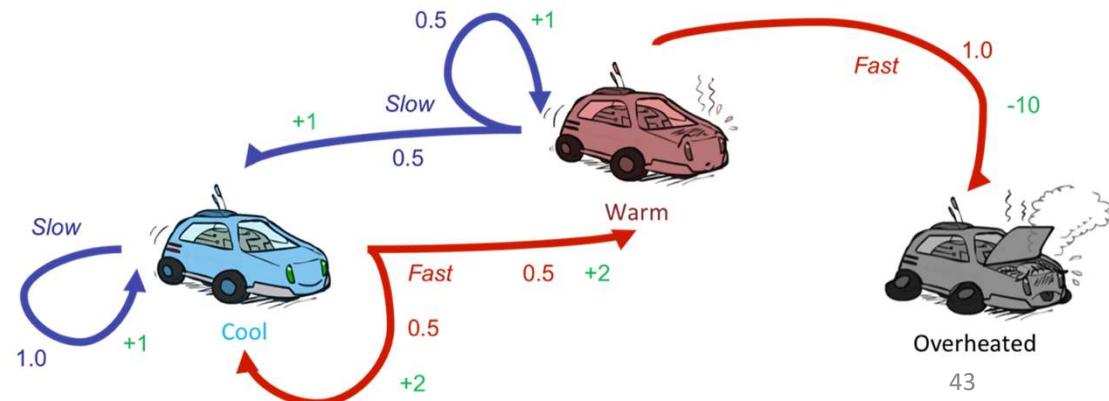
10				1
----	--	--	--	---

- Quiz 3: For which γ are West and East equally good when in state d?



Race car example

- Consider a discount factor, $\gamma = 0.9$
- What is $v^*(Cool)$
- $= \max_a [r(s, a) + \sum_{s'} p(s'|s, a) \gamma v^*(s')]$
- $= \max[1 + 0.9 \cdot 1v^*(Cool), 2 + 0.9 \cdot 0.5v^*(Cool) + 0.9 \cdot 0.5v^*(Warm)]$
 - Computing...
 - ...Stack overflow
- Work in iterations





Value iteration

Value iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$



Value Iteration

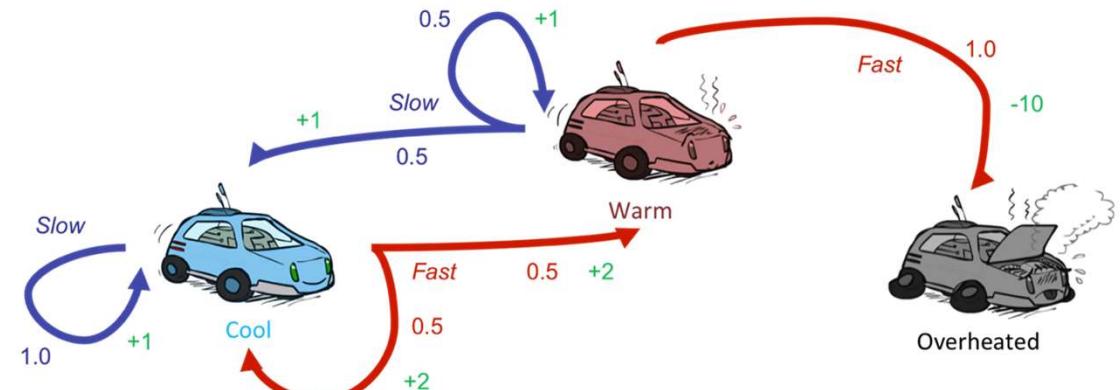


V_0	0	0	0
-------	---	---	---

V_1	2	1	0
-------	---	---	---

V_2	3.35	2.35	0
-------	------	------	---

Check this computation on paper.

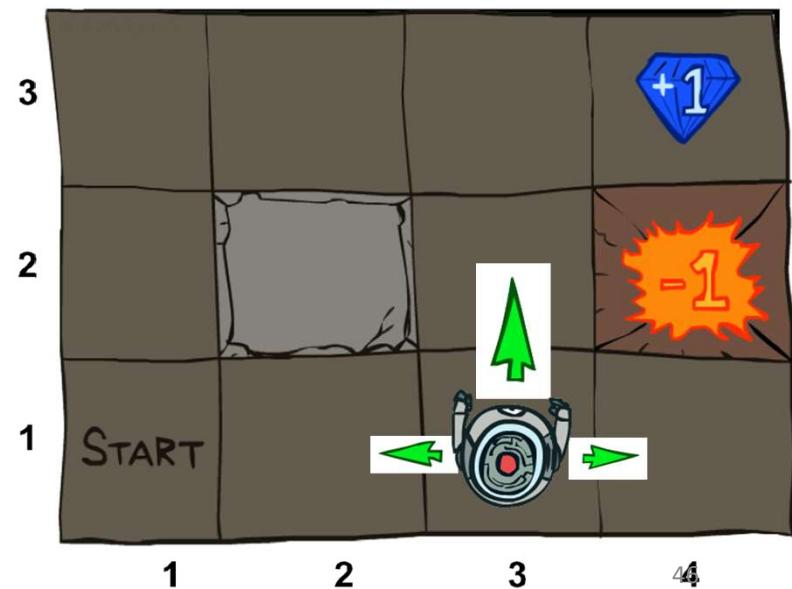
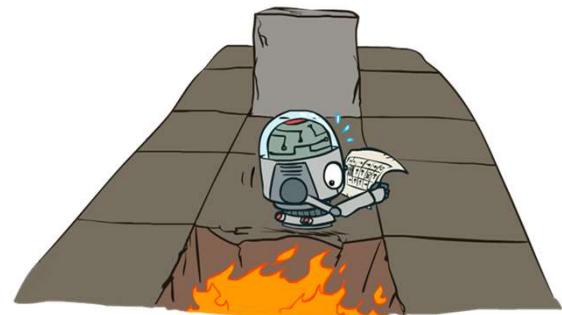


$$v_{k+1}(s) \doteq \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]$$



Example: Grid World

- A maze-like problem
 - The agent lives in a grid
 - Walls block the agent's path
- Noisy movement: actions do not always go as planned
 - 80% of the time, the action North takes the agent North
 - 10% of the time, North takes the agent West; 10% East
 - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
 - Small negative reward each step (battery drain)
 - Big rewards come at the end (good or bad)
- Goal: maximize sum of (discounted) rewards

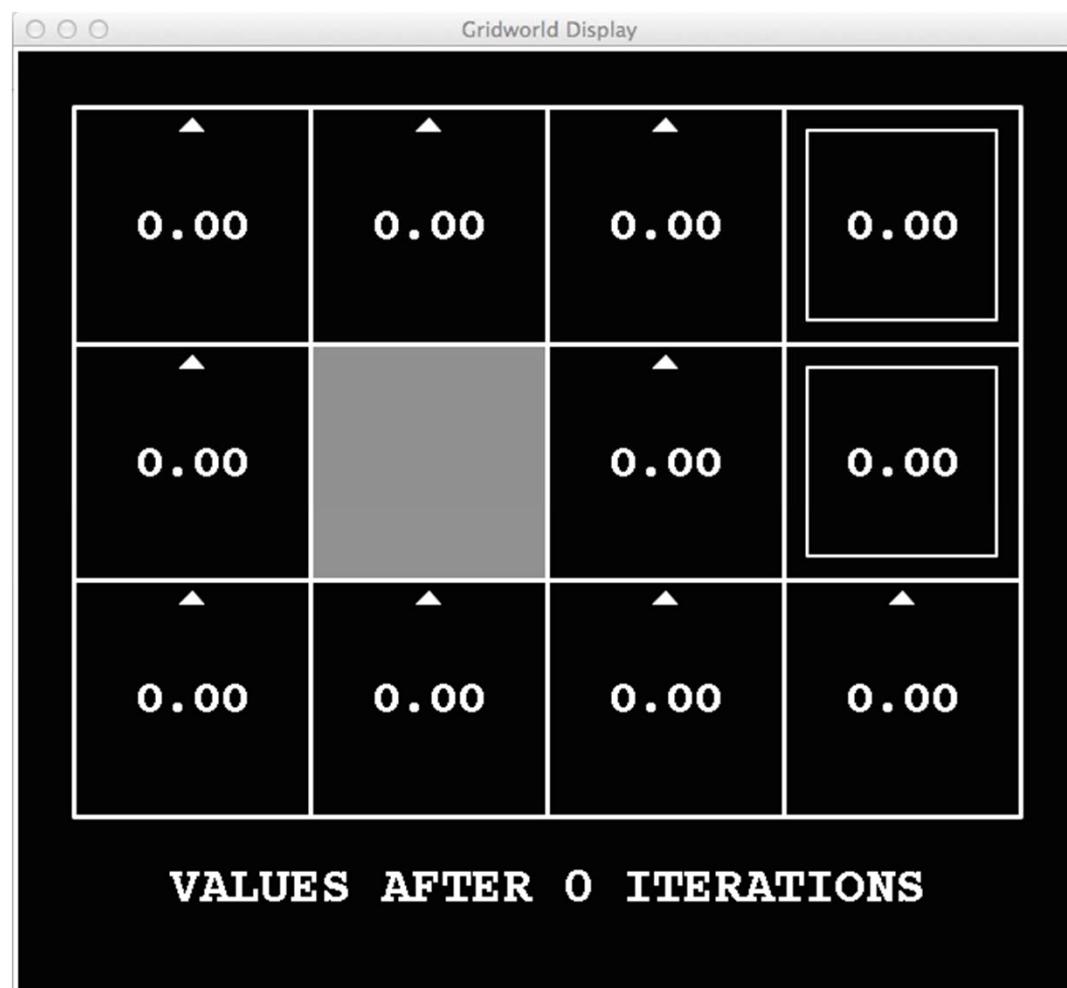


46



k=0

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=1

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=2

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=3

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=4

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=5

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=6

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=7

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=8

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=9

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=10

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=11

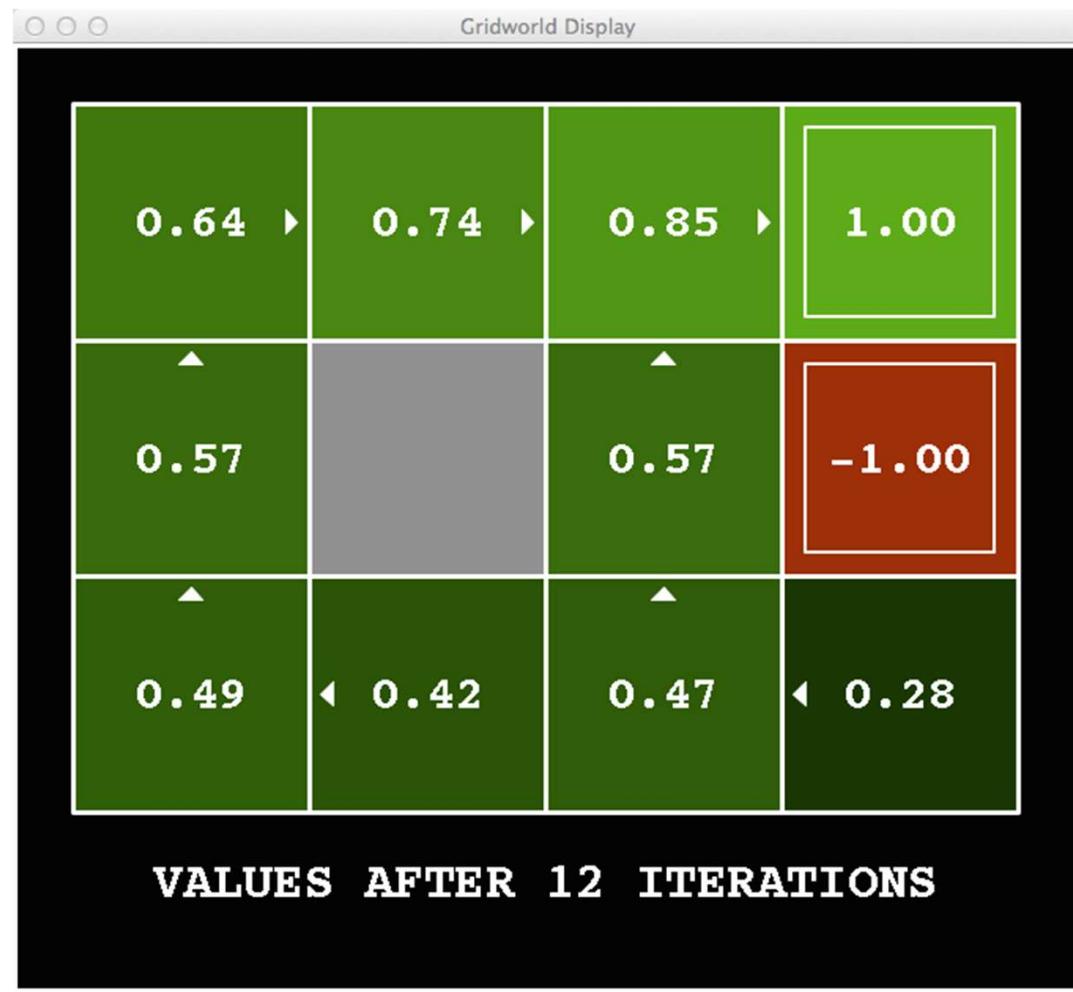
$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=12

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





k=100

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$





Problems with Value Iteration

- Value iteration repeats the Bellman updates:

$$\begin{aligned} V_{k+1}(s) &\leftarrow \max_a [R(s, a) + \sum_{s'} P(s'|s, a) \gamma V_k(s')] \\ &= \max_a \left[\sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V_k(s')) \right] \end{aligned}$$

- **Issue 1:** It's slow – $O(S^2A)$ per iteration
 - Do we really need to update every state at every iteration?
- **Issue 2:** A policy cannot be easily extracted
 - Policy extraction requires another $O(S^2A)$
- **Issue 3:** The policy often converges long before the values
 - Can we identify when the policy converged?
- **Issue 4:** requires knowing the model, $P(s'|s, a)$, and the reward function, $R(s, a)$
- **Issue 5:** requires discrete (finite) set of actions
- **Issue 6:** infeasible in large state spaces



Solutions (briefly, more later...)

- **Issue 1:** It's slow – $O(S^2A)$ per iteration
 - Asynchronous value iteration
- **Issue 2:** A policy cannot be easily extracted
 - Learn q (action) values
- **Issue 3:** The policy often converges long before the values
 - Policy-based methods
- **Issue 4:** requires knowing the model and the reward function
 - Reinforcement learning
- **Issue 5:** requires discrete (finite) set of actions
 - Policy gradient methods
- **Issue 6:** infeasible for large (or continuous) state spaces
 - Function approximators

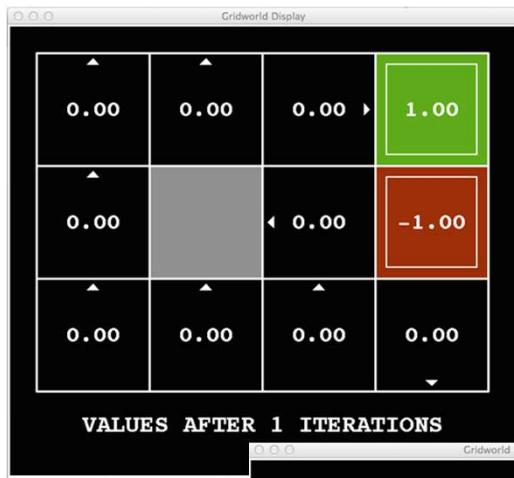


Issue 1: It's slow – $O(S^2A)$ per iteration

- Asynchronous value iteration
- In value iteration, we update every state in each iteration
- Actually, *any* sequences of Bellman updates will converge if every state is visited infinitely often regardless of the visitation order
- Idea: prioritize states whose value we expect to change significantly



Asynchronous Value Iteration



- Which states should be prioritized for an update?

A single update per iteration



Algorithm 3 Prioritized Value Iteration

```
1: repeat
2:    $s \leftarrow \arg \max_{\xi \in S} H(\xi)$ 
3:    $V(s) \leftarrow \max_{a \in A} \{R(s, a) + \gamma \sum_{s' \in S} Pr(s'|s, a)V(s')\}$ 
4:   for all  $s' \in SDS(s)$  do
5:     // recompute  $H(s')$ 
6:   end for
7: until convergence
```

$$SDS(s) = \{s' : \exists a, p(s|s', a) > 0\}$$

For the home assignment set:

$$H(s') = \left| V(s') - \max_a \left\{ R(s', a) + \gamma \sum_{s''} \Pr(s''|s', a)V(s'') \right\} \right|$$



Double the work?

- Computing priority is similar to updating the state value (W.R.T computational effort)
- Why do double work?
 - If we computed the priority, we can go ahead and update the value for free
- Notice that we don't need to update the priorities for the entire state space
- For many of the states the priority doesn't change following an updated value for a single state s
- Only states s' with $\sum_a p(s'|s, a) > 0$ require update

For the home assignment set:

$$H(s') = \left| V(s') - \max_a \left\{ R(s', a) + \gamma \sum_{s''} \Pr(s''|s', a) V(s'') \right\} \right|$$



Issue 2: A policy cannot be easily extracted

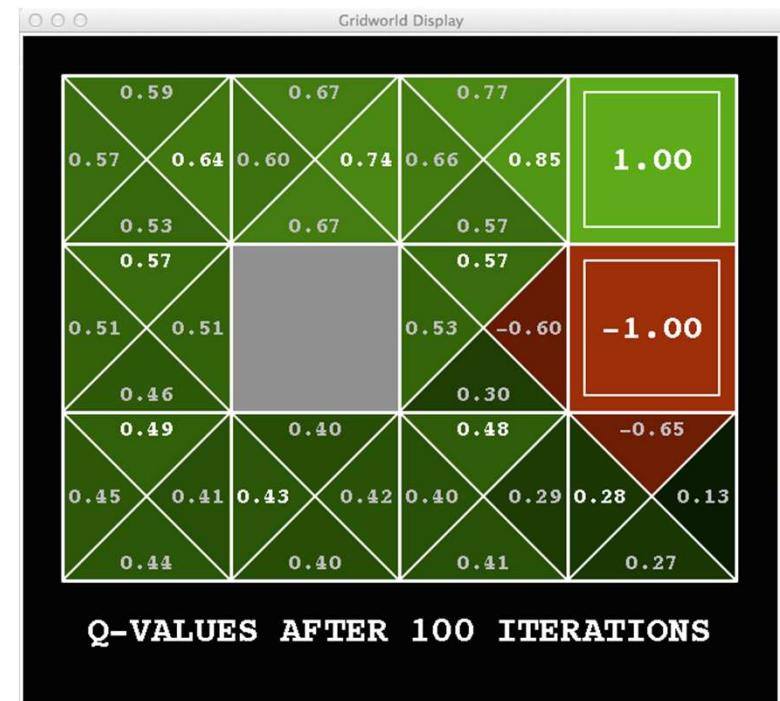
- Given state values, what is the appropriate policy?
 - $\pi(s) \leftarrow \operatorname{argmax}_a [R(s, a) + \sum_{s'} P(s'|s, a) \gamma V_k(s')]$
 - Requires another full value sweep: $O(S^2A)$
- Learn q (action) values instead
- $Q^*(s, a)$ - the expected sum of rewards from being at s , taking action a and then following π^*





Q-learning

- $Q^*(s, a)$ - the expected sum of rewards from being at s , taking action a and then following π^*
- $\pi^*(s) \leftarrow \underset{a}{\operatorname{argmax}}[Q^*(s, a)]$
- Can we learn Q values with dynamic programming?
 - Yes, similar to value iteration





Q-learning as value iteration

- $V^*(s) := \max_a [\sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V^*(s'))]$
- $V^*(s) := \max_a [Q^*(s, a)]$
- $Q^*(s, a) := \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V^*(s'))$
- $Q^*(s, a) = \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma \max_a [Q^*(s', a)])$
- **Solve iteratively**
 - $Q_{k+1}(s, a) = \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma \max_a [Q_k(s', a)])$
 - Can also use Asynchronous learning



Issue 3: The policy often converges long before the values

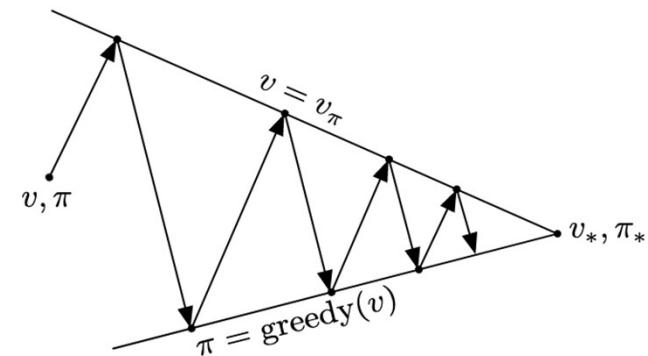
- Value iteration converges to the true utility value: $V_{k \rightarrow \infty} \rightarrow V^*$
- V^* implies the optimal policy: π^*
- Can we converge directly on π^* ?
 - Improve the policy in iteration until reaching the optimal one





Policy Iteration

- **Compute V_π :** calculate state value for some fixed policy (not necessarily the optimal values, $V_\pi \neq V^*$)
 - **Update π :** update policy using one-step look-ahead with the resulting (non optimal) values
 - Repeat until policy converges
(optimal values and policy)
- Guaranteed converges to π^*
 - $\forall s, V_{k>0}(s) \leq V_{k+1}(s)$ i.e., π_i improves monotonically with i
 - A fixed point, $\forall s, V_k(s) = V_{k+1}(s)$, implies π^*



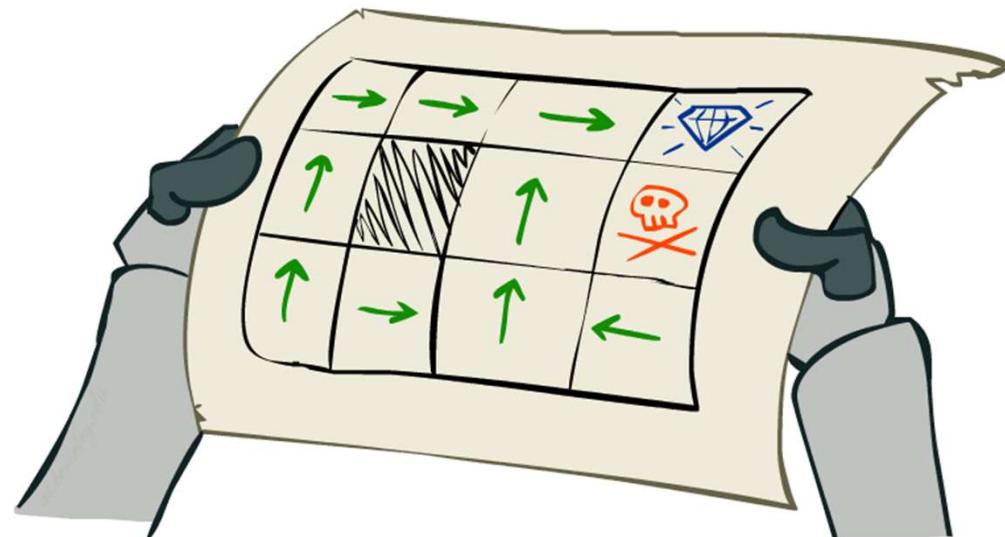


Policy Evaluation

- Why is calculating V_π easier than calculating V^* ?
 - Turns non-linear Bellman equations into linear equations
- $v^*(s) = \max_a [\sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma v^*(s'))]$
- $v_\pi(s) = \sum_{s'} P(s'|s, \pi(s)) (R(s, \pi(s), s') + \gamma v^*(s'))$
- Solve a set of linear equations in $O(S^2)$
 - Solve with Numpy (`numpy.linalg.solve`)
 - Required for your home assignment
 - See: <https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html#numpy.linalg.solve>

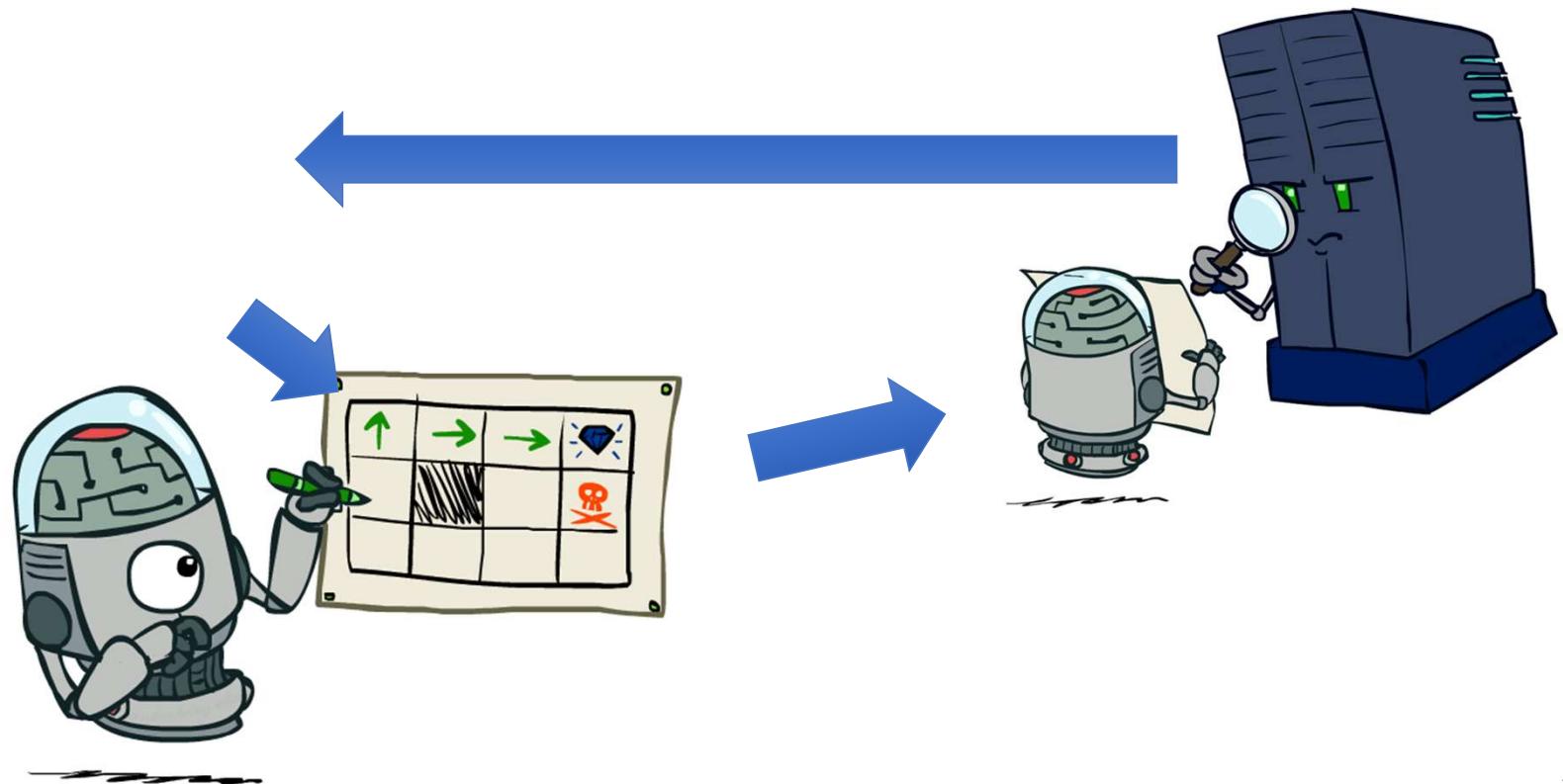
Policy value as a Linear program

- $v_{11} = 0.8 \cdot (-0.1 + 0.95 \cdot v_{12}) + 0.1 \cdot (-0.1 + 0.95 \cdot v_{21}) + 0.1 \cdot (-0.1 + 0.95 \cdot v_{11})$
- $v_{12} = 0.8 \cdot (-0.1 + 0.95 \cdot v_{13}) + 0.2 \cdot (-0.1 + 0.95 \cdot v_{12})$
- ...
- $v_{42} = -1$
- $v_{43} = 1$





Policy iteration



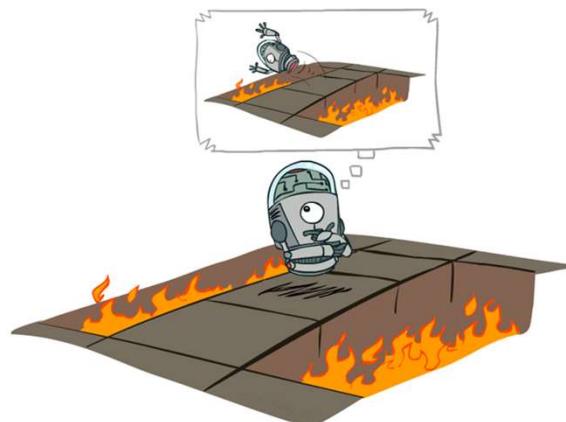


Comparison

- Both value iteration and policy iteration compute the same thing (optimal state values)
- In value iteration:
 - Every iteration updates both the values and (implicitly) the policy
 - We don't track the policy, but taking the max over actions implicitly define it
- In policy iteration:
 - We do several passes that update utilities with fixed policies (each pass is fast because we consider only one action, not all of them)
 - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
 - The new policy will be better (or we're done)

Issue 4: requires knowing the model and the reward function

- We will explore online learning (reinforcement learning) approaches
- How can we learn the model and reward function from interactions?
- Do we even need to learn them? Can we learn V^* , Q^* without a model?
- Can we do without V^* , Q^* ? Can we run policy iteration without a model?



Offline optimization



Online Learning



Issue 5: requires discrete (finite) set of actions

- We will explore policy gradient approaches that are suitable for continuous actions, e.g., throttle and steering for a vehicle
- Can such approaches be relevant for discrete action spaces?
 - Yes! We can always define a continuous action space as a distribution over the discrete actions (e.g., using the softmax function)
- Can we combine value-based approaches and policy gradient approaches and get the best of both?
 - Yes! Actor-critic methods



Issue 6: infeasible in large (or continues) state spaces

- Most real-life problems contain very large state spaces (practically infinite)
- It is infeasible to learn and store a value for every state
- Moreover, doing so is not useful as the chance of encountering a state more than once is very small
- We will learn to generalize our learning to apply to unseen states
- We will use value function approximators that can generalize the acquired knowledge and provide a value to any state (even if it was not previously seen)



Notation

- π^* - a policy that yields the maximal expected sum of rewards
- $V^*(s)$ - the expected sum of rewards from being at s then following π^*
- $V_\pi(s)$ - the expected sum of rewards from being at s then following π
- $Q^*(s, a)$ - the expected sum of rewards from being at s , taking action a and then following π^*
- $Q_\pi(s, a)$ - the expected sum of rewards from being at s , taking action a and then following π
- G_t - observed sum of rewards following time t , i.e., $\sum_{k=t}^T r_k$



Required Readings

1. Chapter-3,4 of Introduction to Reinforcement Learning, 2nd Ed., Sutton & Barto



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Reinforcement Learning
2022-23 Second Semester, M.Tech (AIML)

Session #4: Markov Decision Processes

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)



Dynamic Programming

Value Iteration

Value iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

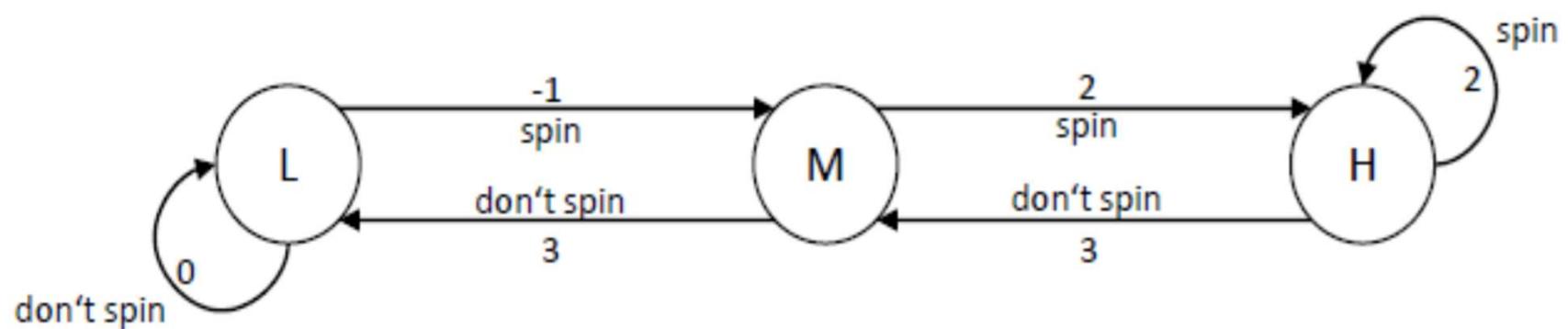


Numerical Example

Consider the following problem :

Consider a rover that operates on a slope and uses solar panels to recharge. It can be in one of three states: high, medium and low on the slope. If it spins its wheels, it climbs the slope in each time step (from low to medium or from medium to high) or stays high. If it does not spin its wheels, it slides down the slope in each time step (from high to medium or from medium to low) or stays low. Spinning its wheels uses one unit of energy per time step. Being high or medium on the slope gains three units of energy per time step via the solar panels, while being low on the slope does not gain any energy per time step. The robot wants to gain as much energy as possible.

- a) Draw the MDP graphically.
- b) Solve the MDP using value iteration with a discount factor of 0.8.
- c) Describe the optimal policy.



where $L = \text{low}$, $M = \text{medium}$ and $H = \text{high}$.

Starting with 0 as initial values, value iteration calculates the following:

At each Iteration, the value of the state is the value of the maximizing action in that state (since we are trying to maximize the energy) and is marked with an asterisk.



$$v1(L, \text{spin}) = -1 + 0.8 \times v0(M) = -1$$

$$v1(L, \text{dont}) = 0 + 0.8 \times V0(L) = 0$$

$$V1(L) = \max(v1(L, \text{spin}), v1(L, \text{dont})) = 0$$

Likewise,

$$v4(L, \text{spin}) = -1 + 0.8 \times v3(M) = 4.06$$

$$v4(L, \text{dont}) = 0 + 0.8 \times v3(L) = 2.02$$

$$v4(L) = \max(v1(L, \text{spin}), v1(L, \text{dont})) = 4.06$$

$$v1(M, \text{spin}) = 2 + 0.8 \times v0(L) = 2$$

$$v1(M, \text{dont}) = 3 + 0.8 \times V0(H) = 3$$

$$V1(L) = \max(v1(M, \text{spin}), v1(M, \text{dont})) = 3$$

Likewise,

$$v4(M, \text{spin}) = 2 + 0.8 \times v3(H) = 7.216$$

$$v4(M, \text{dont}) = 3 + 0.8 \times v3(L) = 5.016$$

$$v4(L) = \max(v1(M, \text{spin}), v1(M, \text{dont})) = 7.216$$



Value Iteration table:

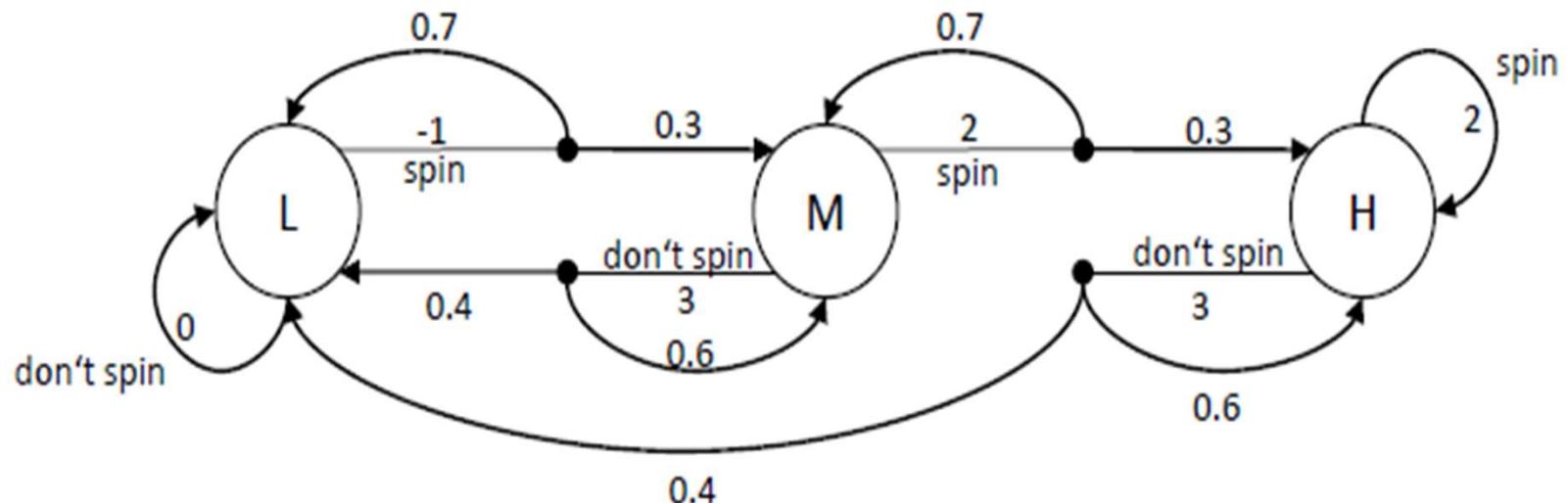
ITR	L		M		H	
	spin	don't	spin	don't	spin	don't
1	-1.00	0.00*	2.00	3.00*	2.00	3.00*
2	1.40*	0.00	4.40*	3.00	4.40	5.40*
3	2.52*	1.12	6.32*	4.12	6.32	6.52*
4	4.06*	2.02	7.22*	5.02	7.22	8.06*
5	4.77*	3.24	8.44*	6.24	8.44	8.77*
6	5.76*	3.82	9.02*	6.82	9.02	9.76*
7	6.21*	4.60	9.80*	7.60	9.80	10.21*
8	6.84*	4.97	10.17*	7.97	10.17	10.84*
9	7.14*	5.47	10.67*	8.47	10.67	11.14*
10	7.54*	5.71	10.91*	8.71	10.91	11.54*

The optimal policy is to spin when the rover is low or medium on the slope and not to spin when it is high on the slope.



Numerical example 2

Now answer the three questions above for the following variant of the robot problem: If it spins its wheels, it climbs the slope in each time step (from low to medium or from medium to high) or stays high, all with probability 0.3. It stays where it is with probability 0.7. If it does not spin its wheels, it slides down the slope to low with probability 0.4 and stays where it is with probability 0.6. Everything else remains unchanged from the previous problem.



Starting with 0 as initial value , calculate value iteration.

In this variation, for example in iteration 4 , the value of $v_4(L)$ is computed as

$$v_4(L, \text{spin}) = -1 + 0.8 \times (0.3 \times v_3(M) + 0.7 \times V_3(L)) = -1 + 0.8(0.3 \times 5.55 + 0.7 \times 0.07) = 0.37$$

$$v_4(L, \text{dont}) = 0 + 0.8 \times v_3(L) = 0.8 \times 0.07 = 0.056$$

$$v_4(L) = \max(v_4(L, \text{spin}), v_4(L, \text{dont})) = 0.37$$



Value Iteration table

ITR	L		M		H	
	spin	don't	spin	don't	spin	don't
1	-1.00	0.00*	2.00	3.00*	2.00	3.00*
2	-0.28	0.00*	4.40	4.44*	4.40	4.44*
3	0.07*	0.00	5.55*	5.13	5.55*	5.13
4	0.37*	0.05	6.44*	5.69	6.44*	5.69
5	0.75*	0.30	7.15*	6.21	7.15*	6.21
6	1.14*	0.60	7.72*	6.67	7.72*	6.67
7	1.49*	0.91	8.18*	7.07	8.18*	7.07
8	1.80*	1.19	8.54*	7.40	8.54*	7.40
9	2.06*	1.44	8.83*	7.68	8.83*	7.68
10	2.27*	1.65	9.07*	7.90	9.07*	7.90

The optimal policy is to spin, wherever the rover is on the slope



Required Readings

1. Chapter-3 of Introduction to Reinforcement Learning, 2nd Ed., Sutton & Barto



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



Session #6-7: Monte Carlo Methods

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)

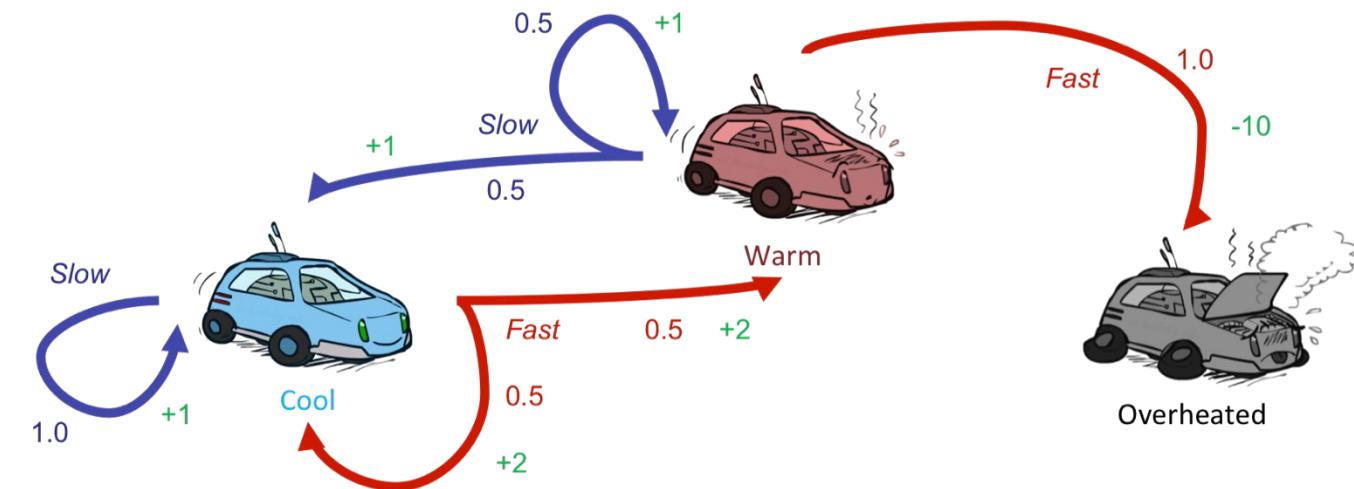


Agenda for the class

- Introduction
- On-Policy Monte Carlo Methods
- Off-Policy Monte Carlo Methods

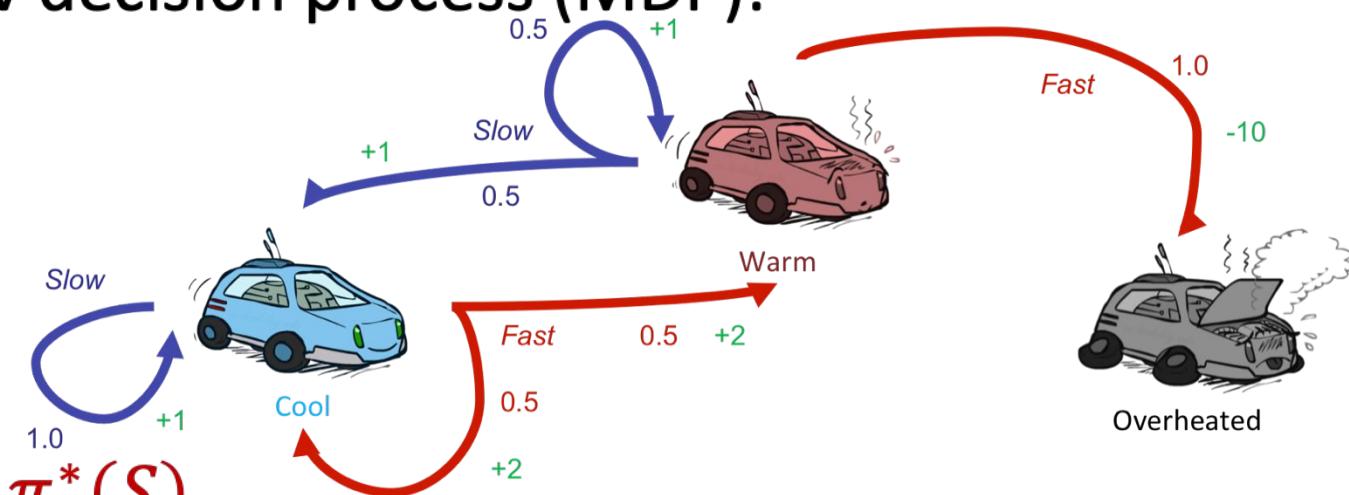
Introduction

- Recollect the problem
 - We need to learn a policy that takes us as far and as faster possible;



Introduction

- Still assume an underlying Markov decision process (MDP):
 - A set of states $s \in S$
 - A set of actions A
 - A model $P(s'|s, a)$
 - A reward function $R(s, a, s')$
 - A discount factor γ
 - Still looking for the best policy $\pi^*(S)$

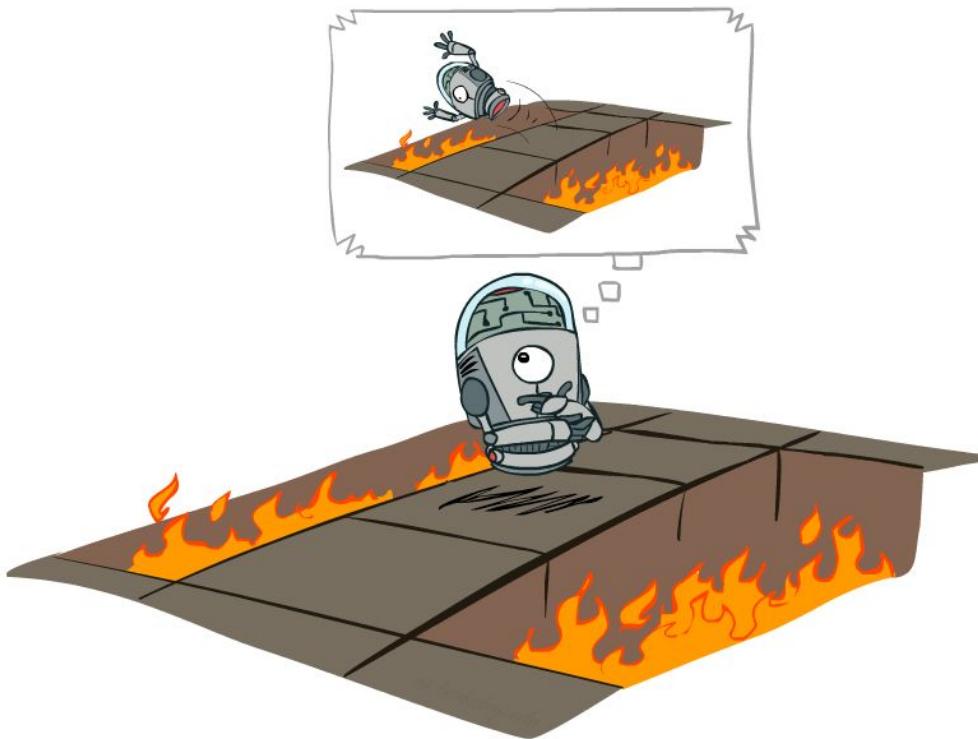


Introduction

- Still assume an underlying Markov decision process (MDP):
 - A **set of states** $s \in S$
 - A **set of actions** A
 - A **model** $P(s'|s, a)$
 - A **reward function** $R(s, a, s')$
 - A discount factor γ
 - Still looking for the best policy $\pi^*(S)$
- New twist: **don't know the model and the reward function**
 - That is, we don't know the actions' outcome
 - Must interact with the environment to learn



(Aside) Offline vs. Online (RL)



Offline Optimization



Online Learning



Monte Carlo Methods

- Monte Carlo methods are a **broad class of computational algorithms** that *rely on repeated random sampling to obtain numerical results*
- The underlying concept is to obtain unbiased samples from a complex/unknown distribution through a random process
- They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to compute a solution analytically
 - Weather prediction
 - Computational biology
 - Computer graphics
 - Finance and business
 - Sport game prediction



First-visit Monte-Carlo Policy Evaluation

[estimate $V\pi(s)$]

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

(a) Generate an episode using π

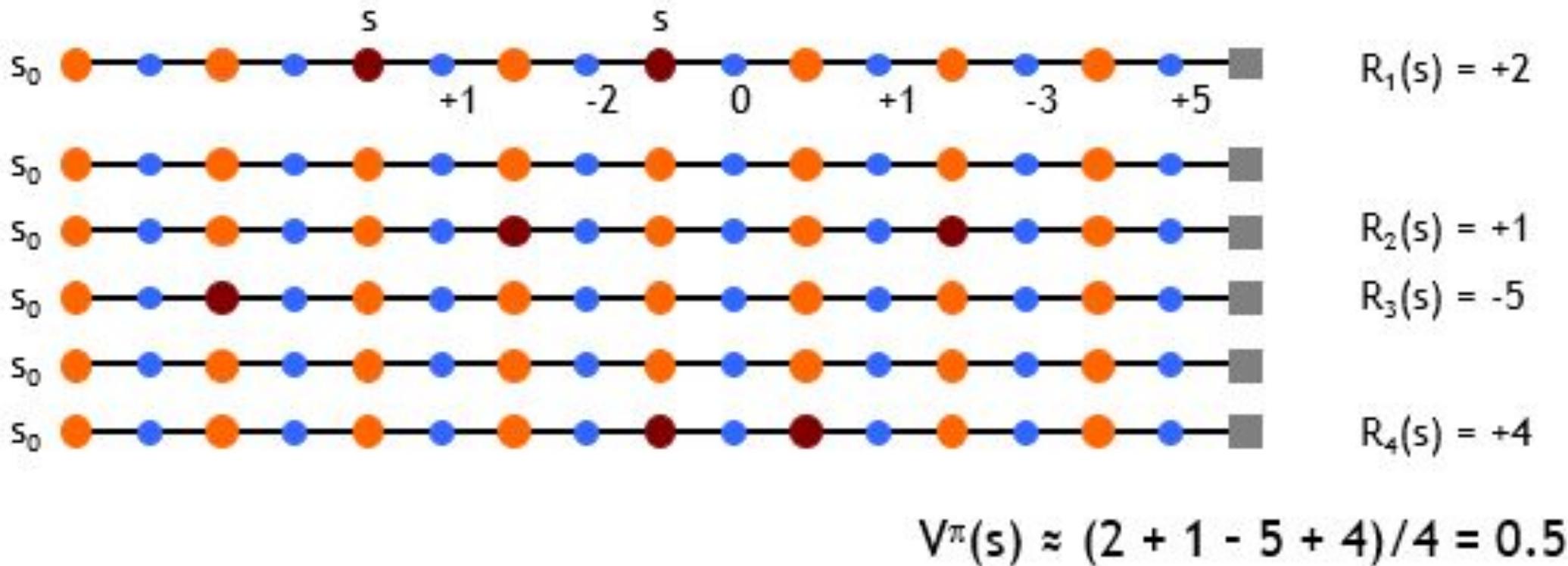
(b) For each state s appearing in the episode:

$R \leftarrow$ return following the first occurrence of s

Append R to $Returns(s)$

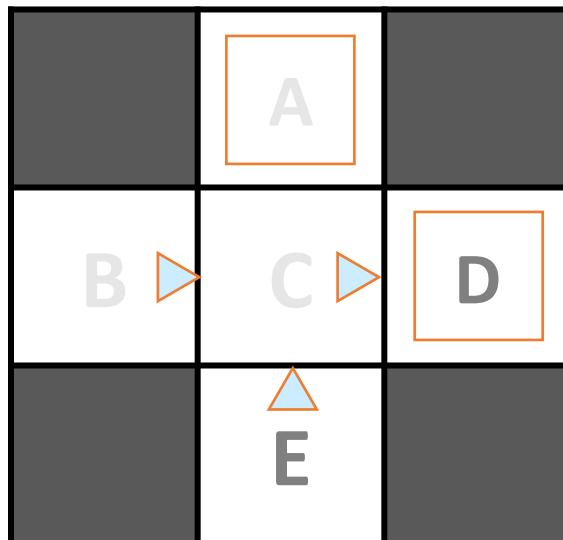
$V(s) \leftarrow \text{average}(Returns(s))$

Ex-1:First-visit Monte-Carlo Policy Evaluation [estimate $V\pi(s)$]



Ex-2: First-visit Monte-Carlo Policy Evaluation [estimate $V\pi(s)$]

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, , +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, , +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, , +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, , -10

Output Values

	-10	
A	+8	+4
B	C	D
	-2	
E		

Problems with MC Evaluation

- What's good about direct evaluation?
 - It's easy to understand
 - It doesn't require any knowledge of the underlying model
 - It converges to the true expected values
- What bad about it?
 - It wastes information about transition probabilities
 - Each state must be learned separately
 - So, it takes a long time to learn

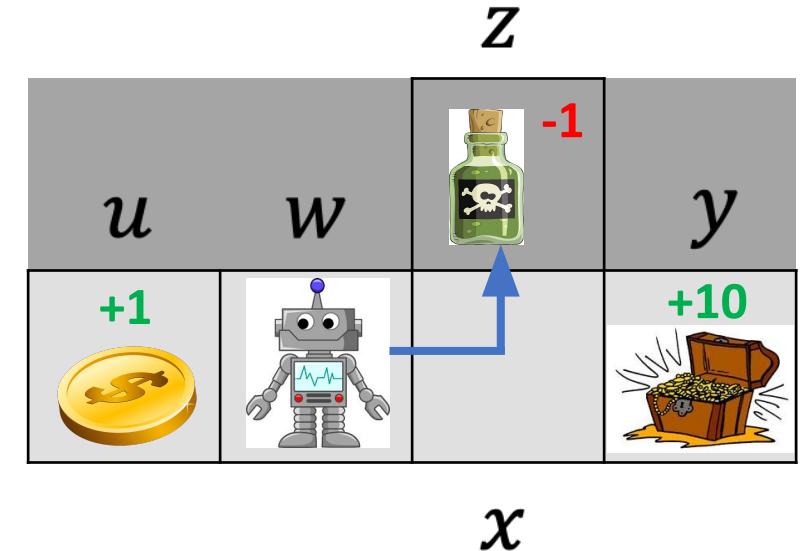
Output Values

	-10 A	
+8 B	+4 C	+10 D
	-2 E	

Think : If B and E both go to C with the same probability, how can their values be different?

What about exploration?

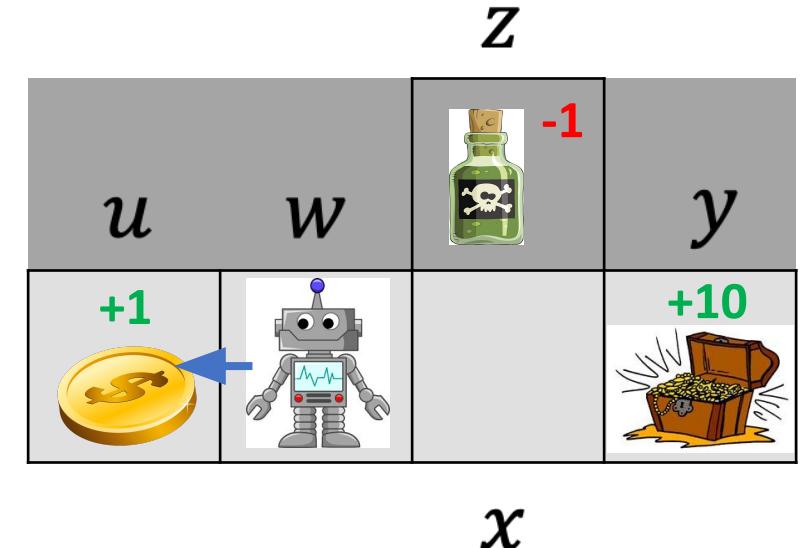
- $S = \{u, w, x, y, z\}$
- $A = \{N, E, S, W, \text{exit}\}$
- Reward:
 - $r(u, \text{exit}) = 1$
 - $r(z, \text{exit}) = -1$
 - $r(y, \text{exit}) = 10$



State						
		NA	NA	NA	NA	0
		0	0 -1	0	0	NA
		0 -1	0	0	0	NA
		NA	NA	NA	NA	0
		NA	NA	NA	NA	0 -1

What about exploration?

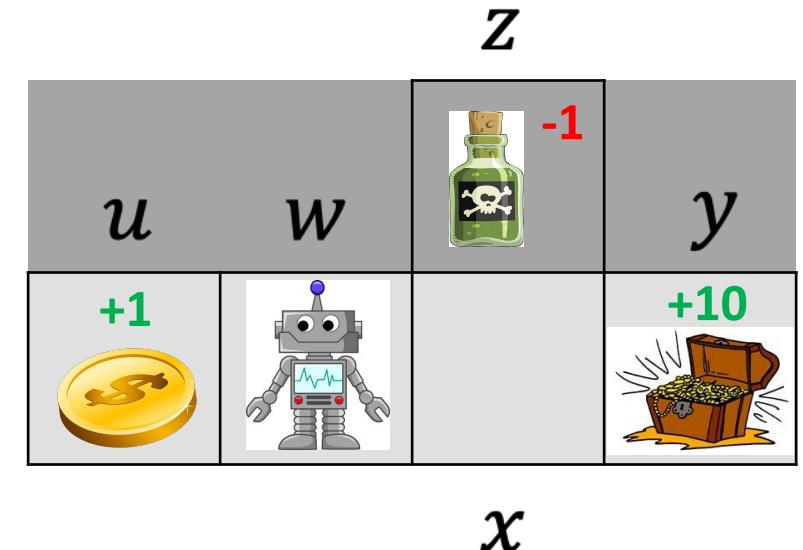
- $S = \{u, w, x, y, z\}$
- $A = \{N, E, S, W, \text{exit}\}$
- Reward:
 - $r(u, \text{exit}) = 1$
 - $r(z, \text{exit}) = -1$
 - $r(y, \text{exit}) = 10$



State							
		NA	NA	NA	NA	0	-1
	W	0	-1	0	0	1	NA
	E	-1	0	0	0	NA	
		NA	NA	NA	NA	0	
		NA	NA	NA	NA	NA	-1

What about exploration?

- $S = \{u, w, x, y, z\}$
- $A = \{N, E, S, W, \text{exit}\}$
- Reward:
 - $r(u, \text{exit}) = 1$
 - $r(z, \text{exit}) = -1$
 - $r(y, \text{exit}) = 10$



State						
		NA	NA	NA	NA	1
		0			1	NA
		-1			0	NA
		NA	NA	NA	NA	0
		NA	NA	NA	NA	-1

We converged on a local optimum!

Must explore!

- Hard policy (insufficient): $\pi(s) = a$, $\pi: S \rightarrow \mathcal{A}$
- Soft policy: $\pi(a|s) = [0,1]$, $\pi: S \times \mathcal{A} \rightarrow p$
 - At the beginning $\forall a$, $\pi(a|s) > 0$ to allow exploration
 - Gradually shift towards a deterministic policy
- For instance: select a random action with probability ε
 - $\forall a \neq A^*$, $\pi(s, a) = \frac{\varepsilon}{|\mathcal{A}(s)|}$
 - Else select the greedy action: $\pi(s, A^*) = 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$

ε -greedy MC control

On-policy first-visit MC control (for ε -soft policies), estimates $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

$\pi(a|s) \leftarrow$ an arbitrary ε -soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow$ the return that follows the first occurrence of s, a

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

(with ties broken arbitrarily)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

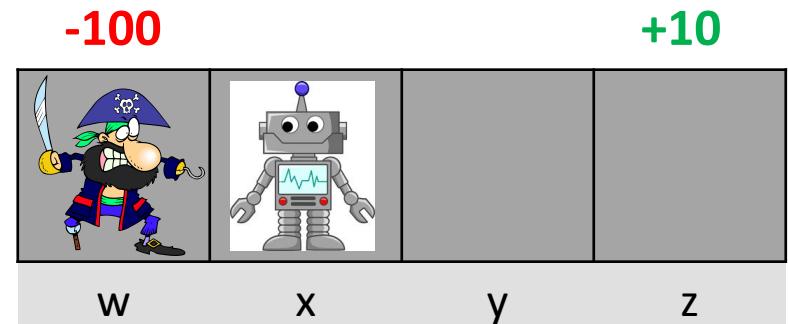
$$\gamma = 0.9$$

MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline 5 & 4,3 & 2,1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $Returns = \begin{array}{|c|c|c|c|} \hline - & -, - & -, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $\varepsilon \cdot \text{Random}$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$ (with tie-breaking rule)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

$\gamma = 0.9$

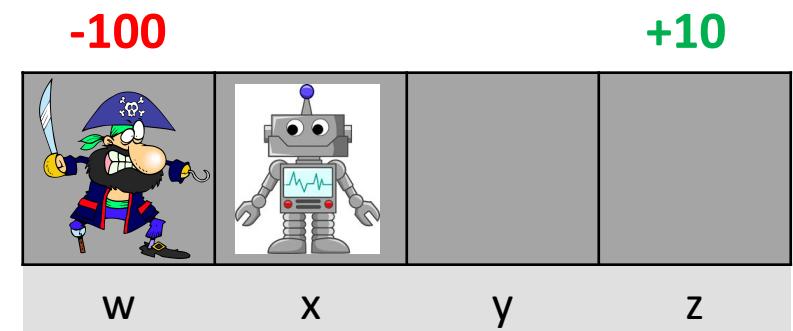
MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline 5 & 4,3 & 2,1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $Returns = \begin{array}{|c|c|c|c|} \hline - & -, - & -, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $\varepsilon \cdot \text{Random}$

- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

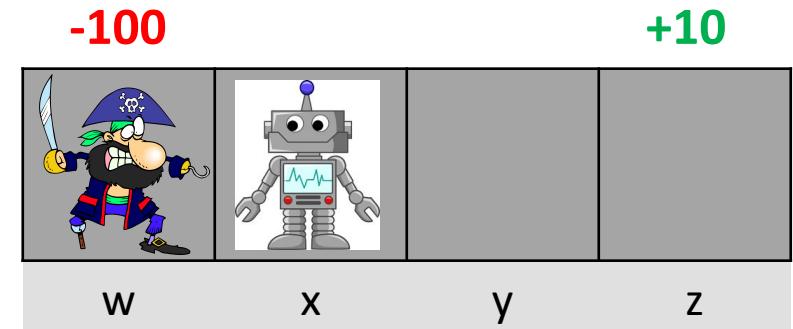
$A^* \leftarrow \arg \max_a Q(s, a)$ (with tie-breaking rule)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline 5 & 4,3 & 2,1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\overline{Returns} = \begin{array}{|c|c|c|c|} \hline -100 & -90,0 & -, - & - \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\varepsilon \cdot \text{Random}$
 - $\tau = x, \leftarrow, 0, w, \text{exit}, -100$



On-policy first-visit MC control (for ϵ -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

Returns(s, a) \leftarrow empty list

$\pi(a|s) \leftarrow$ an arbitrary ε -soft policy

Repeat forever:

- (a) Generate an episode using π
 - (b) For each pair s, a appearing in the episode:
 - $G \leftarrow$ the return that follows the first occurrence of s, a
 - Append G to $Returns(s, a)$
 - $Q(s, a) \leftarrow$ average($Returns(s, a)$)
 - (c) For each s in the episode:
 - $A^* \leftarrow \arg \max_a Q(s, a)$ (with tie)
 - For all $a \in \mathcal{A}(s)$:
 - $\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$

$$\gamma = 0.9$$

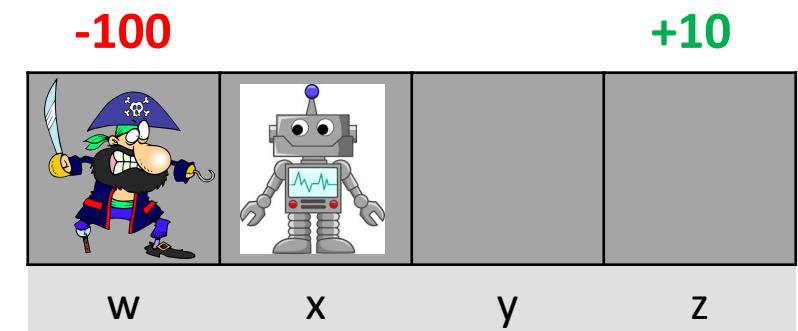
MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $Returns = \begin{array}{|c|c|c|c|} \hline -100 & -90,0 & -, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\varepsilon \cdot \text{Random}$

- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

(with tie)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

$$\gamma = 0.9$$

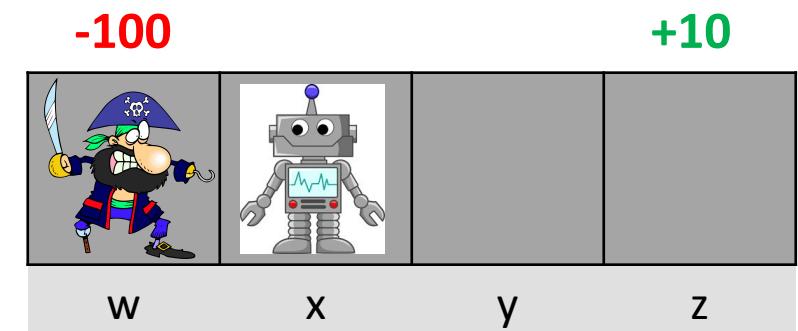
MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $Returns = \begin{array}{|c|c|c|c|} \hline -100 & -90,0 & -, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\varepsilon \cdot \text{Random}$

- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$
- $A^* = [\rightarrow, \text{exit}]$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

(with tie)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

$$\gamma = 0.9$$

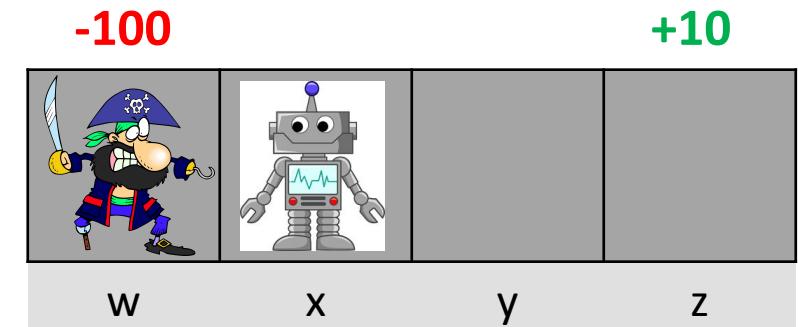
MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $Returns = \begin{array}{|c|c|c|c|} \hline -100 & -90,0 & -, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\varepsilon \cdot \text{Random}$

- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$
- $A^* = [\rightarrow, \text{exit}]$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

(with tie)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

$\gamma = 0.9$

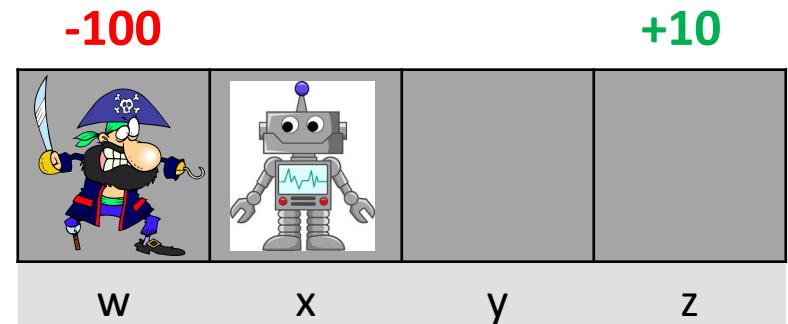
MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $Returns = \begin{array}{|c|c|c|c|} \hline -100 & -90,0 & -, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $\varepsilon \cdot \text{Random}$

- $\tau = x, \rightarrow, 0, y, \leftarrow, 0, x, \leftarrow, 0, \text{exit}, -100$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

(with tie)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

$$\gamma = 0.9$$

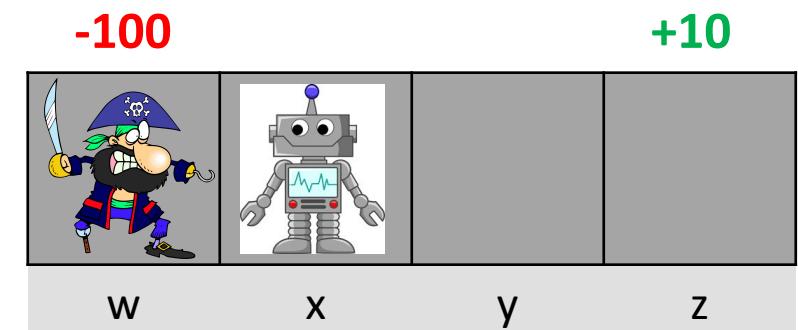
MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90, -72.9 & -81, 1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $Returns = \begin{array}{|c|c|c|c|} \hline -100 & -90, -72.9 & -81, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\varepsilon \cdot \text{Random}$

- $\tau = x, \rightarrow, 0, y, \leftarrow, 0, x, \leftarrow, 0, \text{exit}, -100$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

(with tie)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

$$\gamma = 0.9$$

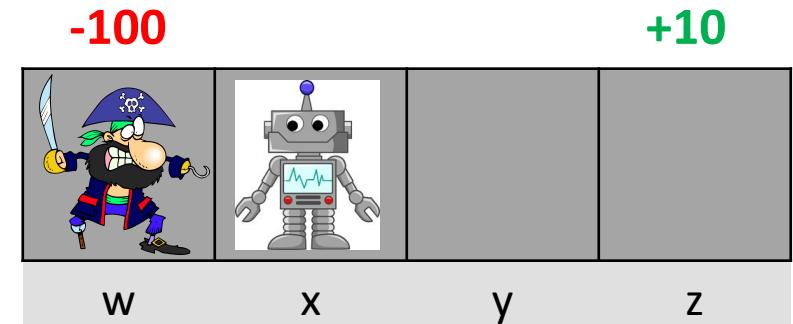
MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90, -72.9 & -81, 1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $\overline{Returns} = \begin{array}{|c|c|c|c|} \hline -100 & -90, -72.9 & -81, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\varepsilon \cdot \text{Random}$

- $\tau = x, \rightarrow, 0, y, \leftarrow, 0, x, \leftarrow, 0, \text{exit}, -100$
- $A^* = [\rightarrow, \rightarrow, \text{exit}]$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

(with tie-breaking rule)

For all $a \in \mathcal{A}(s)$:

$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$

$$\gamma = 0.9$$

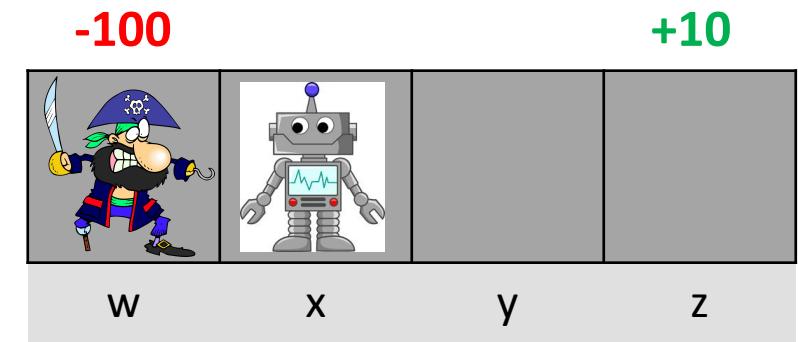
MC control - example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90, -72.9 & -81, 1 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$

- $\overline{Returns} = \begin{array}{|c|c|c|c|} \hline -100 & -90, -72.9 & -81, - & - \\ \hline w & x & y & z \\ \hline \end{array}$

- $\pi(a|s) = (1 - \varepsilon) \cdot \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
 - $\varepsilon \cdot \text{Random}$

- $\tau = x, \rightarrow, 0, y, \leftarrow, 0, x, \leftarrow, 0, \text{exit}, -100$
- $A^* = [\rightarrow, \rightarrow, \text{exit}]$



On-policy first-visit MC control (for ε -soft policies),

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

$\pi(a|s) \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow \text{the return that follows the first occurrence of } s, a$

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

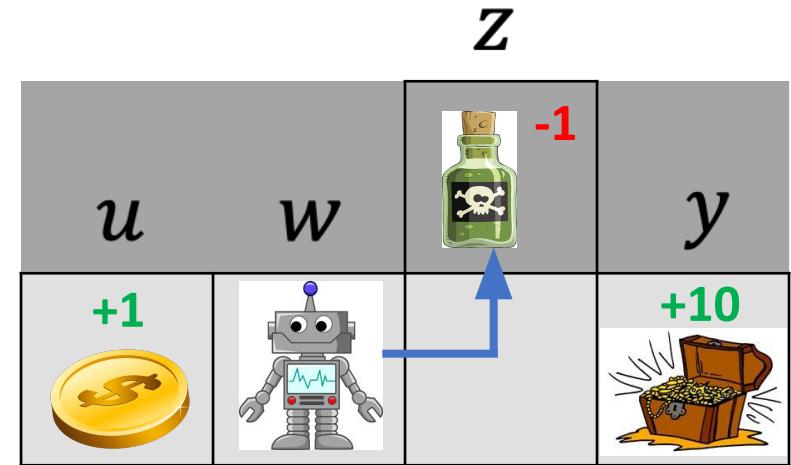
$A^* \leftarrow \arg \max_a Q(s, a)$

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

On-policy learning

- | Estimation | True value |
|---|------------|
| $Q_\pi(w, \rightarrow) = q_\pi(w, \rightarrow) = -1$ (correct!) | |
| $q^*(w, \rightarrow) = ?$ | |
| $q_{\pi \neq b} = ?$ | |
| Observation drawn from π are useful for evaluating q_π | |
| Once the policy is changed these observations are irrelevant | |
| This is not sample efficient! | |



State			
		NA	NA
		0	-1
		-1	0
		NA	NA
		NA	NA

Quick Recap !

On-policy vs. Off-policy Learning



Off-policy learning

- We would like to use observations drawn from some policy b to evaluate q_π where $\pi \neq b$, specifically, we want to evaluate q_{π^*}
- We strive for full utilization of previous experience
- Off-policy learning allows us to optimize a ***target policy*** while following another ***behavior policy***
- **Pros:** sample efficient
- **Cons:** greater variance in value estimations



Off-policy learning conditions

- **Objective:** use episodes from b to estimate values for π
- For off-policy learning we must assume ***coverage***
 - $\forall s, a, \pi(a|s) > 0 \Rightarrow b(a|s) > 0$
- If this is true, then by running b repeatedly we will eventually discover all possible trajectories for π
- If coverage is violated for some s, a , then no inference is possible regarding that state-action value

Trajectory probability

- An agent following policy b sampled the following trajectory
 - $\tau = \{S_0, A_0, R_1, S_1, A_1, R_2, \dots, A_{T-1}, R_T, S_T\}$
- What is the probability of sampling this trajectory?
 - $\Pr\{\tau|b\} = b(A_0|S_0)p(S_1|S_0, A_0)b(A_1|S_1)p(S_2|S_1, A_1) \dots b(A_{T-1}|S_{T-1})p(S_T|S_{T-1}, A_{T-1})$
 - $= \prod_{k=0}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)$
- Assume MC control, how should $Q_b(S_t, A_t)$ be updated?
 - $G_t = \sum_{k=t+1}^T \gamma^{k-t+1} R_k$
 - Append G to $Returns(S_t, A_t)$ with weight $\Pr_t\{\tau|b\}$
 - $Q_b(S_t, A_t) = \text{Weighted_AVG}(Returns(S_t, A_t))$
- Computing the weighted average based on the sample probability would reduce variance (eliminate impact of noisy sampling)
- But the model, $p(S_{K+1}|S_K, A_K)$, is unknown! So can't compute $\Pr\{\tau|b\}$
- Can we say anything about $\Pr\{\tau|\pi \neq b\}$?

$$\mathbb{E}[X \sim p] = \sum_x p(x)x$$

Importance sampling

- Given a trajectory τ drawn by running b
 - We can **define** (not compute) the probability $\Pr\{\tau|b\}$
 - We can also **define** $\Pr\{\tau|\pi\}$
 - Define the ***importance sampling ratio*** as: $\rho_t = \frac{\Pr\{\tau_t|\pi\}}{\Pr\{\tau_t|b\}}$
 - Can we **compute** ρ without a model, $p(S_{K+1}|S_K, A_K)$?
 - $$\rho_t = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k) \cancel{p(S_{K+1}|S_K, A_K)}}{\prod_{k=t}^{T-1} b(A_k|S_k) \cancel{p(S_{K+1}|S_K, A_K)}} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$
 YES!

Importance sampling

- **Fact:** $\mathbb{E}_{\tau \sim b}[G_t | S_t = s] = v_b(s)$
- Importance sampling allows us to compute an unbiased estimate of $v_\pi(s)$ by running b
- **Claim:** $\mathbb{E}_{\tau \sim b}[\rho_t G_t | S_t = s] = v_\pi(s)$
- We set $v_\pi(s)$ to be a weighted average of observed returns (weighted by the importance ratio)
- Assume visiting state s over M episodes using policy b
 - s is first visited during time step t^m during each episode, $m \in M$
- $v_\pi(s) = \frac{\sum_{m \in M} \rho_{t^m} G_{t^m}^m}{M}$

Importance sampling: proof

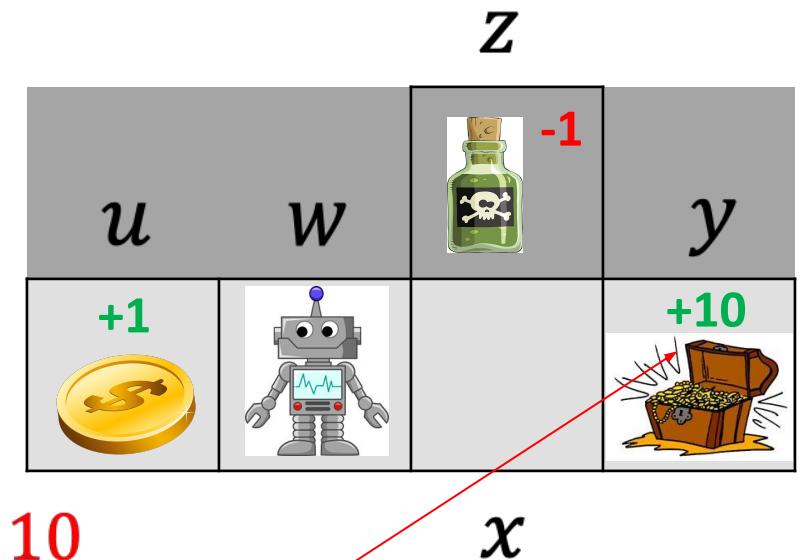
- We would like to estimate $\mathbb{E}[X]$ in our case when $X = v_\pi(s)$
- By definition: $\mathbb{E}[X] = \sum_x \Pr_X(x)x$
 - for the continuous case replace the sum with an integral
- If we don't know $\Pr(x)$ we can take a sample-based approach
- $\mathbb{E}[X] = \sum_x \Pr_X(x)x = \frac{\sum_{i=0}^n x_i}{n}$
 - This is an unbiased estimation because the samples are coming from the same distribution that defines \Pr_X
- But what if our samples come from a different distribution \Pr_Y ?

Importance sampling: proof

- Assume we know: $\mathbb{E}[Y] = \sum_y \Pr(y)y = \frac{\sum_{i=0}^n y_i}{n}$
 - Can we use this to compute $\mathbb{E}[X]$?
 - Yes! if we assume that all possible values of X exist in Y (*converge*)
 - $\mathbb{E}[X] = \sum_x \Pr_X(x)x = \sum_y \Pr_X(y)y$
 - $= \sum_y \frac{\Pr_X(y)}{\Pr_Y(y)} \Pr_Y(y)y$
 - $= \frac{\Pr_X(y)}{\Pr_Y(y)} \cdot \frac{\sum_{i=0}^n y_i}{n}$
- Importance ratio

(ordinary) Importance sampling - example

- $\because b(s|a) = \begin{cases} \rightarrow, p(0.5) \\ \leftarrow, p(0.5) \end{cases}$
- $\bullet \pi(s|a) = \begin{cases} \rightarrow, p(0.99) \\ \leftarrow, p(0.01) \end{cases}$
- $\bullet \tau_1 = \{w, \rightarrow, 0, x, \rightarrow, 0, y, exit, 10\}$
- $\bullet v_\pi(w) = \frac{\sum_{m \in M} \rho_t^m G_t^m}{M} = \frac{0.99}{0.5} * \frac{0.99}{0.5} * 10 = \textcolor{red}{3.96 * 10}$
- $\bullet \tau_2 = \{w, \leftarrow, 0, u, exit, 1\}$
- $\bullet v_\pi(w) = \frac{\sum_{m \in M} \rho_t^m G_t^m}{M} = \frac{3.96*10+0.02*1}{2}$

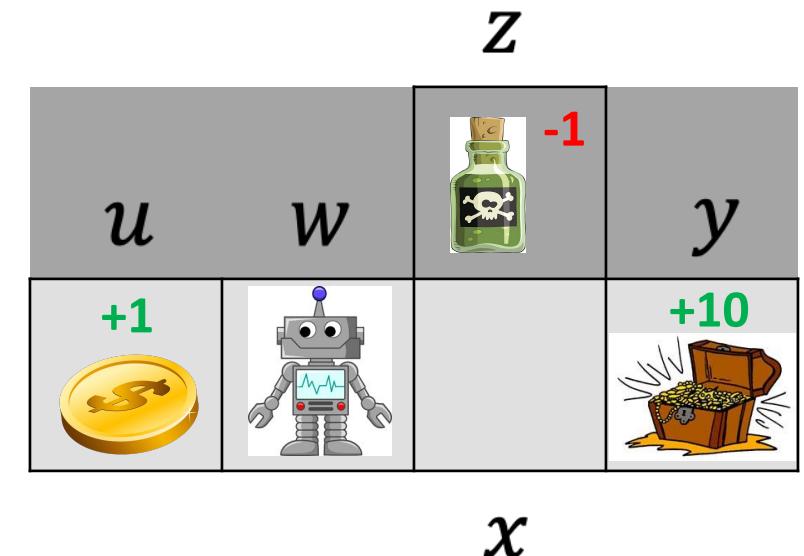


39.6 ??
 Ordinary Importance
 sampling is unbiased yet
 high variance

Weighted importance sampling

- $v_\pi(s) = \frac{\sum_{m \in M} [\rho_t^m G_t^m]}{\sum_{m \in M} \rho_t^m}$
- $b(s|a) = \begin{cases} \rightarrow, p(0.5) \\ \leftarrow, p(0.5) \end{cases}$
- $\pi(s|a) = \begin{cases} \rightarrow, p(0.99) \\ \leftarrow, p(0.01) \end{cases}$
- $\tau_1 = \{w, \rightarrow, 0, x, \rightarrow, 0, y, exit, 10\}$
- $v_\pi(w) = \frac{\sum_{m \in M} [\rho_t^m G_t^m]}{\sum_{m \in M} \rho_t^m} = \frac{3.96*10}{3.96}$
- $\tau_2 = \{w, \leftarrow, 0, u, exit, 1\}$
- $v_\pi(w) = \frac{\sum_{m \in M} [\rho_t^m G_t^m]}{\sum_{m \in M} \rho_t^m} = \frac{3.96*10+0.02*1}{3.98}$

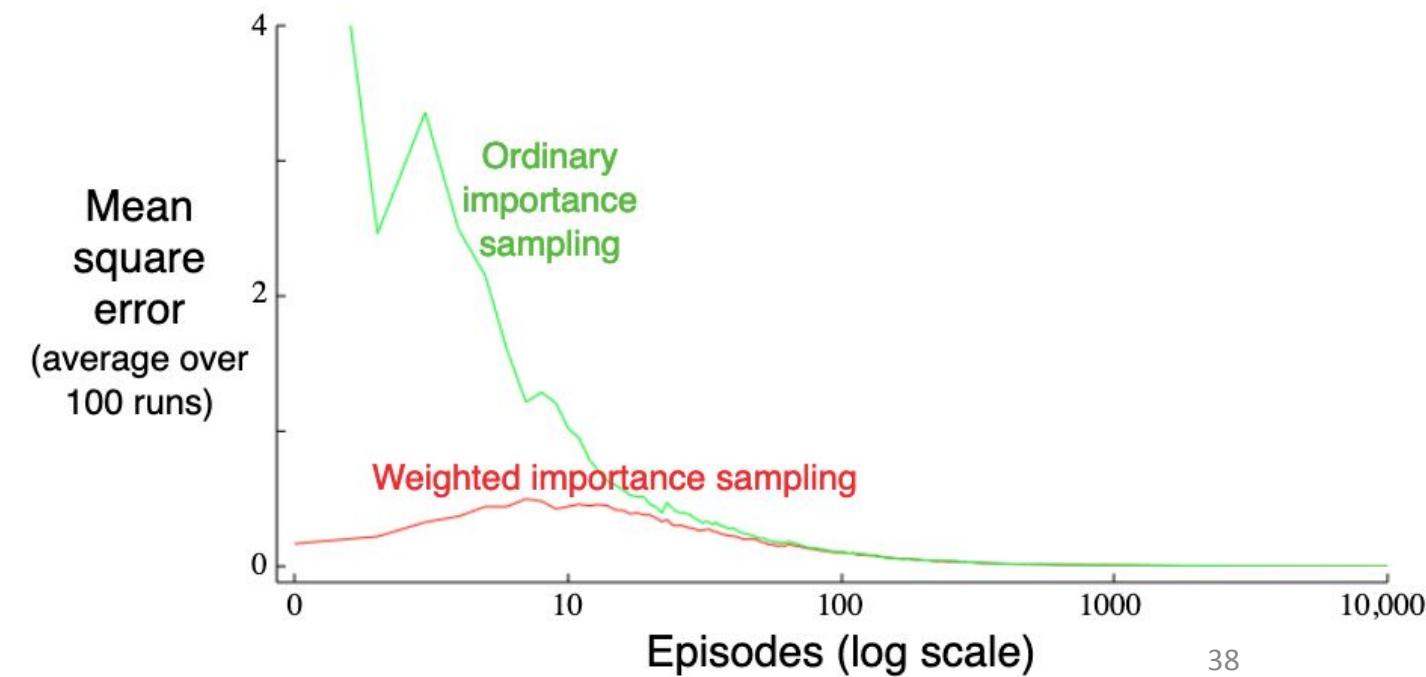
Trick: normalize by the sum of importance ratios



Ordinary Importance sampling is unbiased while the weighted version is biased (initially). Ordinary Importance sampling results in high variance while the weighted version has a bounded variance

Ordinary vs weighted importance sampling

- Estimating a black-jack state
- Target policy: hit on 19 or below
- Behavior policy: random (uniform)
- Both approaches converge to the true value
- weighted importance sampling is much better initially





Announcement !!!

Revised Evaluation Scheme

- Assignment - 25 %
 - Quizzes - 5%
 - 2 Quizzes will be conducted; Each for 5% ;
 - Best score will be taken towards grading
 - Mid Semester Exam - 30 %
 - Comprehensive Exam - 40%
- [EC-1 : 30% ; EC-2: 30%; EC-3- 40%]



Mid Semester Exams & Prep

- Weight – 30%
- Number of Questions – 4 (will have sub-parts)
- Topics:

Introduction to Reinforcement Learning (RL); Multi-armed Bandit Problem; MDP: Framework; Dynamic Programming Solution; Monte Carlo (MC) Methods (MC prediction, MC control, incremental MC.); Temporal-Difference (TD) Learning



To Do in the Tutorial... (Off-Policy MC control and Demonstration)

Revisit : Incremental Implementation (Slide taken from Multi-armed Bandit)

Note:

- StepSize decreases with each update
- We use α or $\alpha_t(a)$ to denote step size (constant / varies with each step)

Discussion:

Const vs. Variable step size?

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i \\ &= \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} \left(R_n + (n-1)Q_n \right) \\ &= \frac{1}{n} \left(R_n + nQ_n - Q_n \right) \\ &= Q_n + \frac{1}{n} [R_n - Q_n], \end{aligned}$$

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} [\text{Target} - \text{OldEstimate}]$$

MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Accumulated
reward after step t

MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$$b \leftarrow \text{any soft policy}$$

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

1 over Joint probability over observed actions from following b after step t . This equals the IS ratio here because the target policy is deterministic.



MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$$b \leftarrow \text{any soft policy}$$

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Going back in time

MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Discount future rewards and add immediate reward

MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Cumulative sum of IS weights affiliated with S_t, A_t (for weighted IS)

MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Incremental update of Q values (waited moving average)

MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Update target policy
(greedy)

MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$$b \leftarrow \text{any soft policy}$$

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Since π is deterministic,
once we diverge from it all
IS weights of earlier
actions will be 0

MC control + importance sampling

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$$b \leftarrow \text{any soft policy}$$

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Update the joint prob by multiplying by ρ_t . Notice that $\pi(S_t) = 1$ in this example (deterministic target policy)

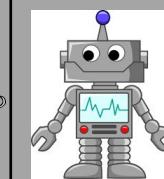
$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{c|c|c|c} - & 4,3 & 2,1 & - \\ \hline w & x & y & z \end{array}$
- $C = \begin{array}{c|c|c|c} - & 0,0 & 0,0 & - \\ \hline w & x & y & z \end{array}$
- $\pi(s) = \begin{array}{c|c|c|c} \text{exit} & & & \text{exit} \\ \hline w & x & y & z \end{array}$

-100

+10



w

x

y

z

Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow \text{any soft policy}$

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

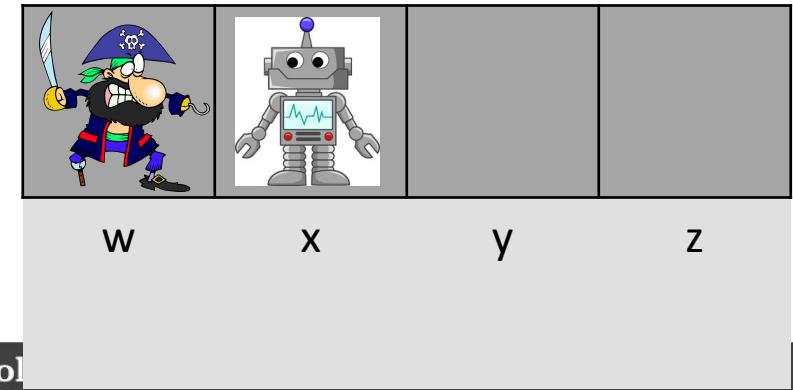
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{c|c|c|c} - & 4,3 & 2,1 & - \\ \hline w & x & y & z \end{array}$
- $C = \begin{array}{c|c|c|c} - & 0,0 & 0,0 & - \\ \hline w & x & y & z \end{array}$
- $\pi(s) = \begin{array}{c|c|c|c} \text{exit} & & & \text{exit} \\ \hline w & x & y & z \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$

$$\begin{array}{c} -100 \\ +10 \end{array}$$



Off-policy

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

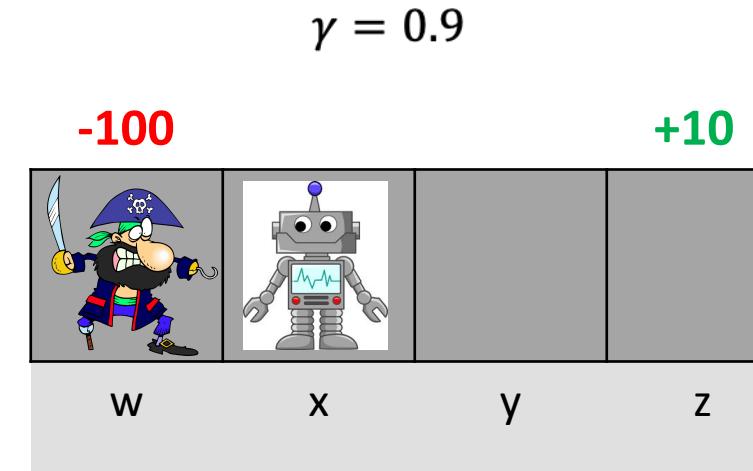
If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

- $Q = \begin{matrix} & 5 & 4,3 & 2,1 & 5 \\ w & x & y & z \end{matrix}$
- $C = \begin{matrix} & 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{matrix}$
- $\pi(s) = \begin{matrix} \text{exit} & & & \text{exit} \\ w & x & y & z \end{matrix}$

- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$G \leftarrow 0$

$W \leftarrow 1$

For $t = T-1, T-2, \dots$ down to 0:

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$

If $A_t \neq \pi(S_t)$ then exit For loop

$W \leftarrow W \frac{1}{b(A_t | S_t)}$

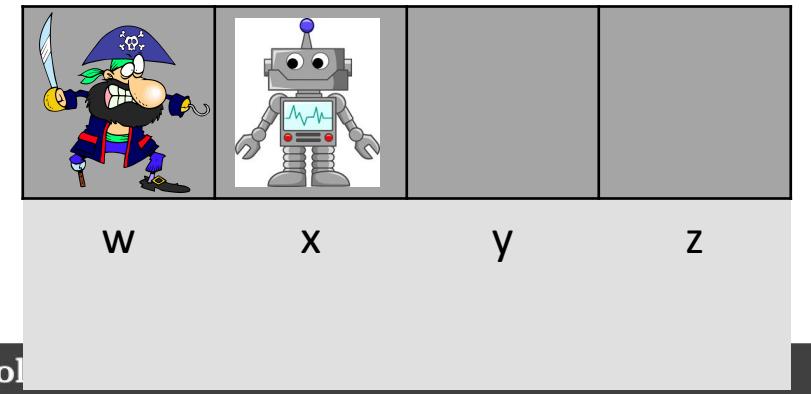
$\gamma = 0.9$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline 5 & 4,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 0 & 0,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$

- $G = -100$
- $W = 1$
- $t = 1$

-100 +10



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

$$\bullet Q = \begin{array}{|c|c|c|c|} \hline & 5 & 4,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$$

$$\bullet \ C = \begin{array}{|c|c|c|c|} \hline & 1 & 0,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$$

- $\pi(s) = \boxed{\text{exit} \quad \quad \quad \quad \quad \text{exit}}$

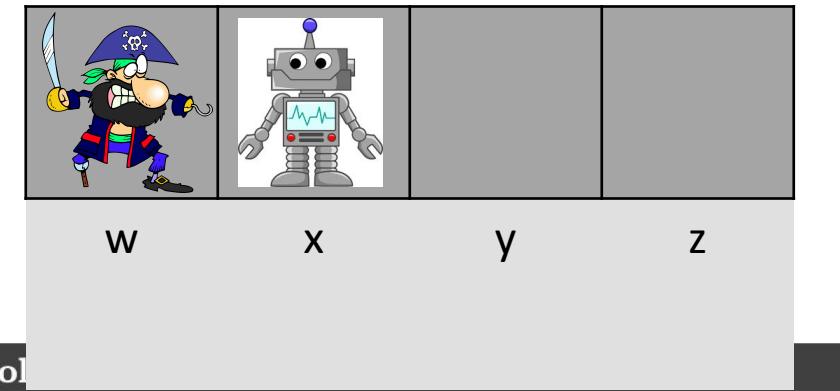
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, exit, -100$

- $G = -100$
- $W = 1$
- $t = 1$

$$\gamma = 0.9$$

-100

+10



Off-po

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \operatorname{argmax}_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using *b*:

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

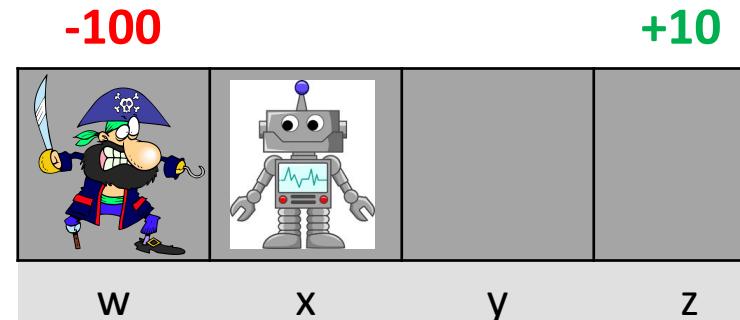
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & 4,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 0,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$

- $G = -100$
- $W = 1$
- $t = 1$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

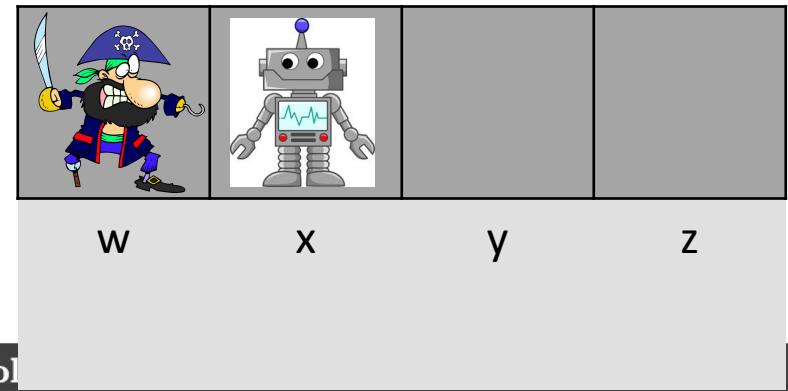
$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & 4,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 0,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$

- $G = -100$
- $W = 1$
- $t = 1$

$$\begin{array}{ccccc} & & \color{red}{-100} & & \color{green}{+10} \\ & & & & \\ \end{array}$$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

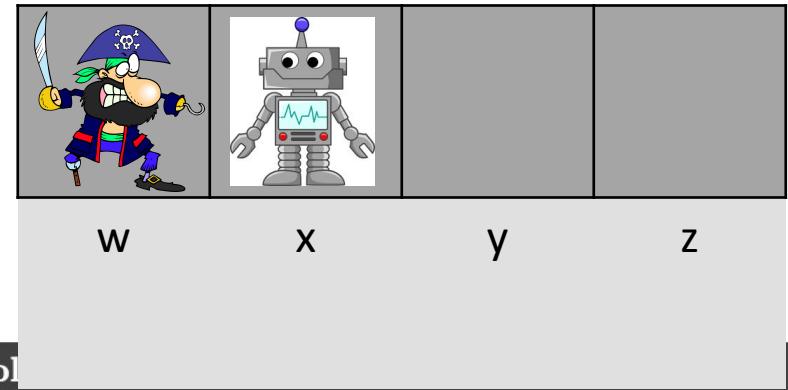
$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & 4,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 0,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$

- $G = -100$
- $W = 2$
- $t = 1$

$$-100 \quad +10$$



Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

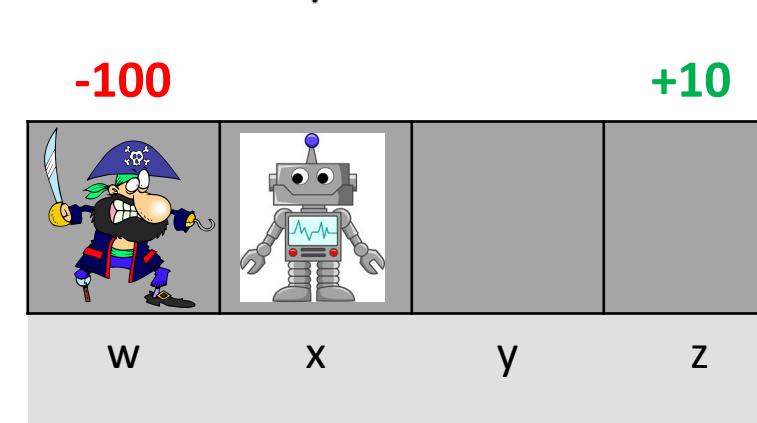
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$\gamma = 0.9$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & 4,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 0,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$

- $G = -90$
- $W = 2$
- $t = 0$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

$$\bullet Q = \begin{array}{|c|c|c|c|} \hline & -100 & 4,3 & 2,1 & 5 \\ \hline w & & x & y & z \\ \hline \end{array}$$

\bullet	$C =$	<table border="1"> <tr> <td>1</td><td>2,0</td><td>0,0</td><td>0</td></tr> <tr> <td>w</td><td>x</td><td>y</td><td>z</td></tr> </table>	1	2,0	0,0	0	w	x	y	z
1	2,0	0,0	0							
w	x	y	z							

- $\pi(s) = \boxed{\text{exit} \quad \quad \quad \quad \quad \text{exit}}$

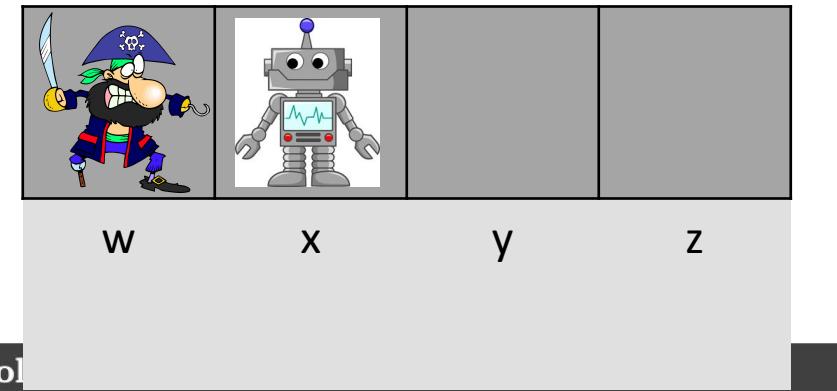
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, exit, -100$

- $G = -90$
- $W = 2$
- $t \equiv 0$

$$\gamma = 0.9$$

-100

+10



Off-po

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \operatorname{argmax}_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using *b*:

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

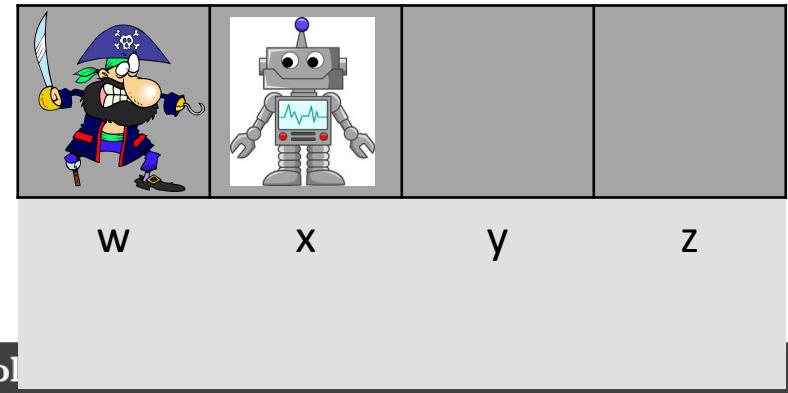
$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$

- $G = -90$
- $W = 2$
- $t = 0$

-100 +10



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

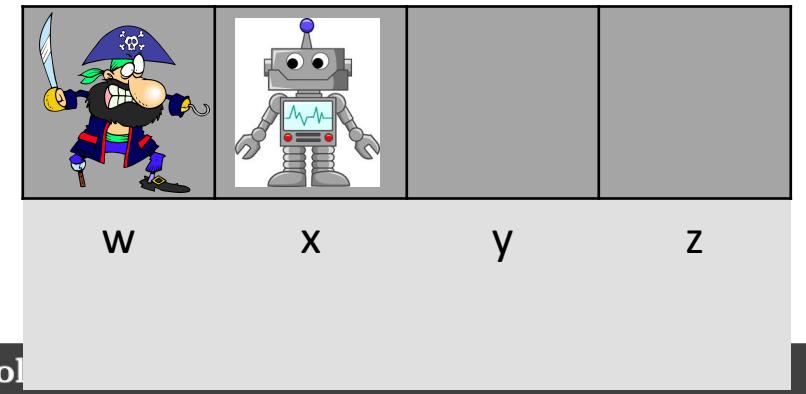
$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, \text{exit}, -100$

- $G = -90$
- $W = 2$
- $t = 0$

-100 +10



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

$$\bullet Q = \begin{array}{|c|c|c|c|} \hline & -100 & -90,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$$

\bullet	$C =$	<table border="1"> <tr> <td>1</td><td>2,0</td><td>0,0</td><td>0</td></tr> <tr> <td>w</td><td>x</td><td>y</td><td>z</td></tr> </table>	1	2,0	0,0	0	w	x	y	z
1	2,0	0,0	0							
w	x	y	z							

- $\pi(s) = \boxed{\text{exit} \quad \quad \quad \quad \quad \text{exit}}$

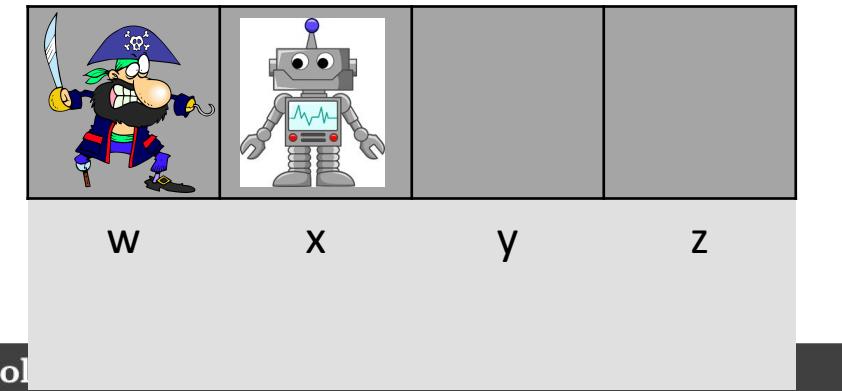
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \leftarrow, 0, w, exit, -100$

- $G = -90$
- $W = 2$
- $t = 0$

$$\gamma = 0.9$$

-100

+10



Off-po

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$$C(s, a) \leftarrow 0$$

$\pi(s) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using *b*:

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

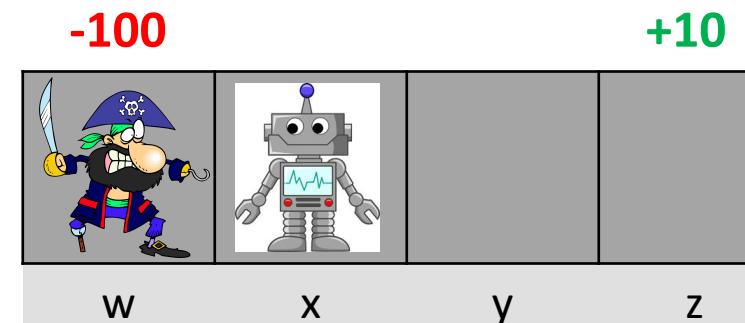
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = -90$
- $W = 2$
- $t = 0$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

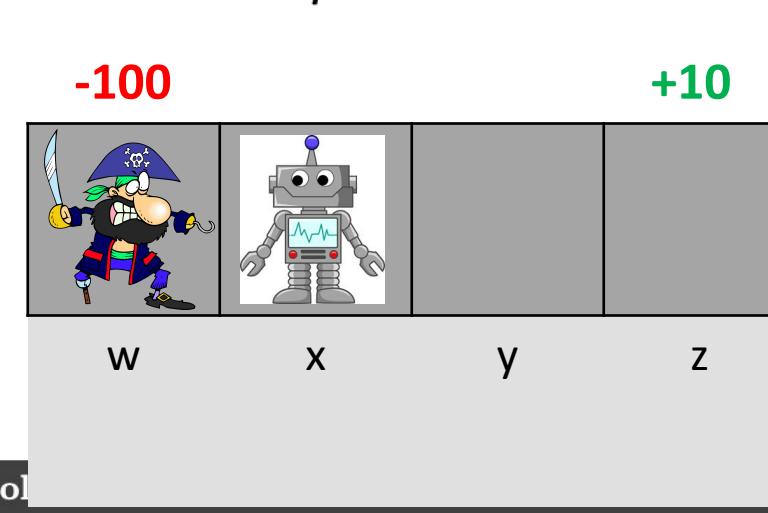
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,0 & 0 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 10$
- $W = 1$
- $t = 2$



Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

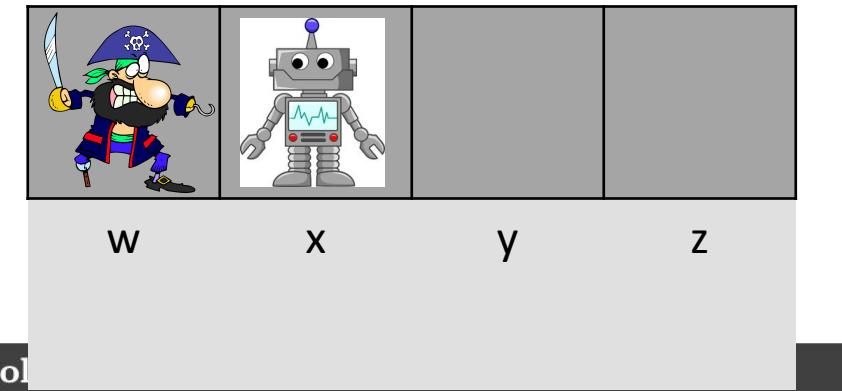
- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 5 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,0 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 10$
- $W = 1$
- $t = 2$

$$\gamma = 0.9$$

-100

+10



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

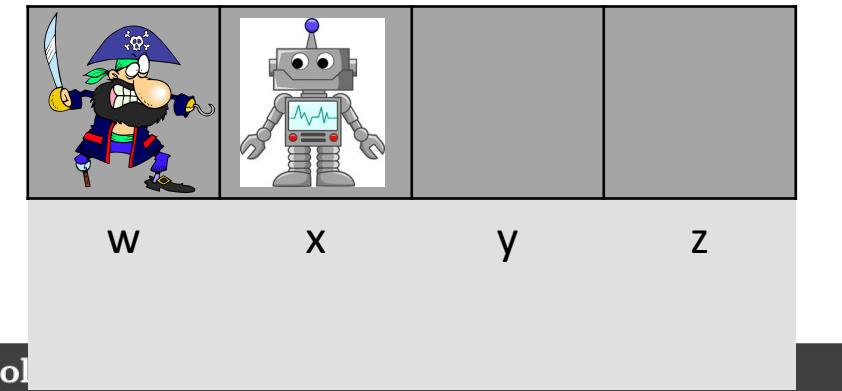
- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,0 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 10$
- $W = 1$
- $t = 2$

$$\gamma = 0.9$$

-100

+10



Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

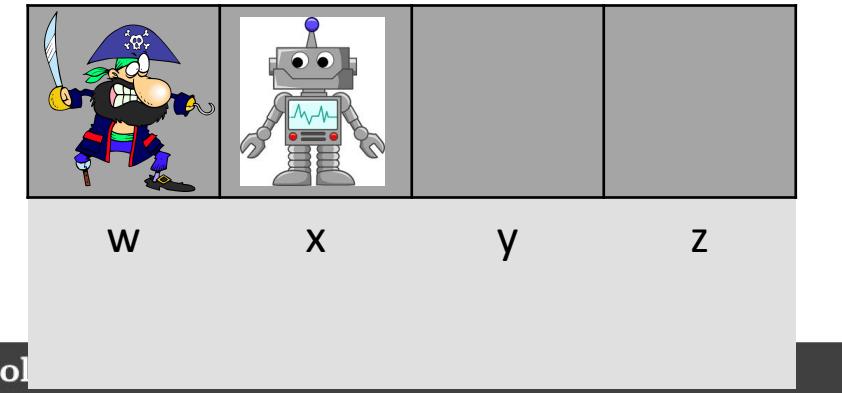
- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,0 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 10$
- $W = 1$
- $t = 2$

$$\gamma = 0.9$$

-100

+10



Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

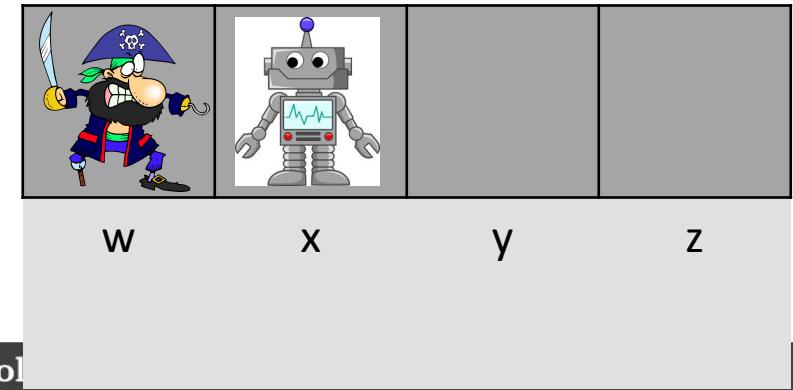
$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,0 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 10$
- $W = 2$
- $t = 2$

-100 +10



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

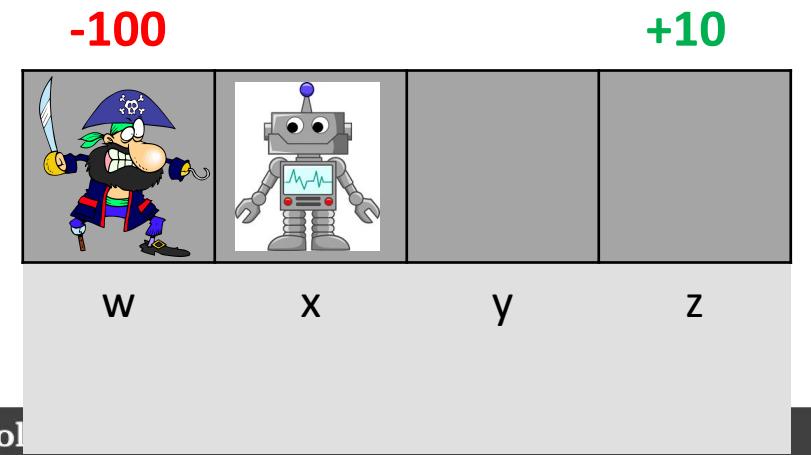
- $Q =$

-100	-90,3	2,1	10
w	x	y	z
 - $C =$

1	2,0	0,0	1
w	x	y	z
 - $\pi(s) =$

exit			exit
w	x	y	z
 - $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
 - $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, exit,$

- $G = 9$
 - $W = 2$
 - $t = 1$



Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \operatorname{arg\,max}_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using *b*:

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$W \leftarrow$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a)$ (with ties broken consistently)

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

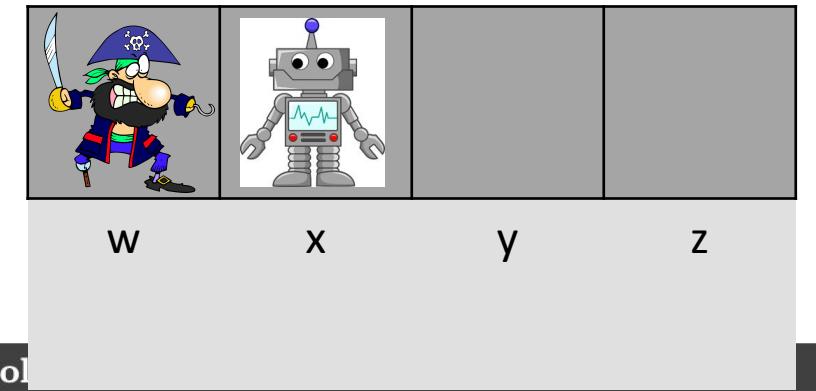
- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,1 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 9$
- $W = 2$
- $t = 1$

$$\gamma = 0.9$$

-100

+10



Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

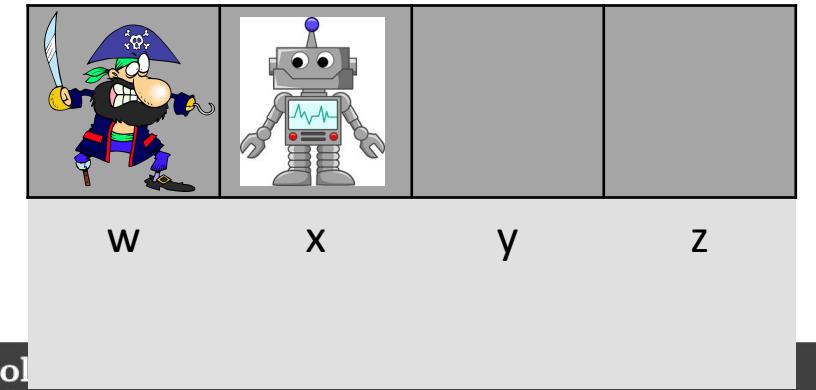
- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,9 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 9$
- $W = 2$
- $t = 1$

$$\gamma = 0.9$$

-100

+10



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

MC control + IS example

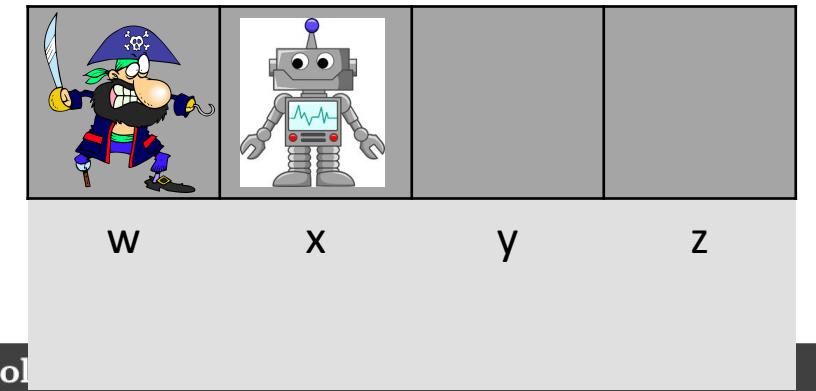
- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,9 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 9$
- $W = 2$
- $t = 1$

$$\gamma = 0.9$$

-100

+10



Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T - 1, T - 2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

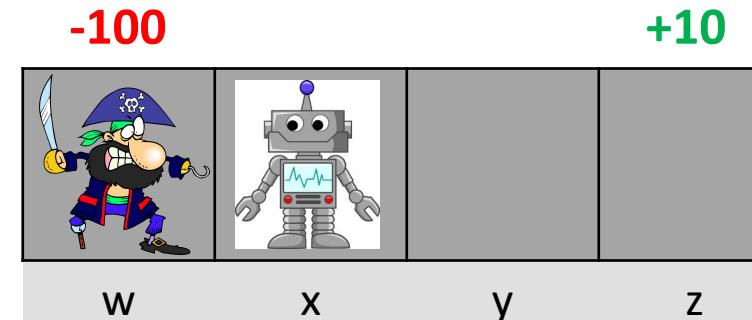
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,9 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 9$
- $W = 4$
- $t = 1$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

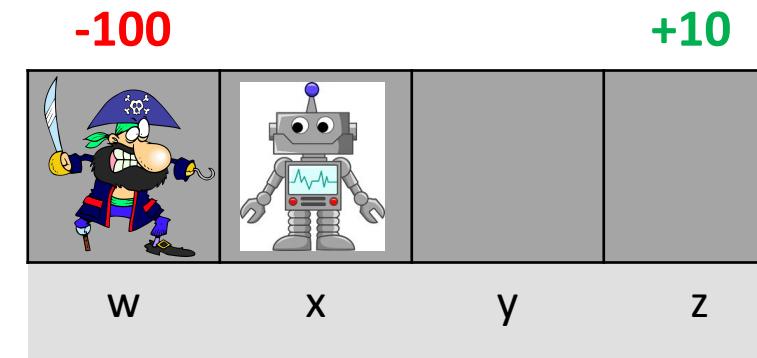
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,9 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,0 & 0,2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 8.1$
- $W = 4$
- $t = 0$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

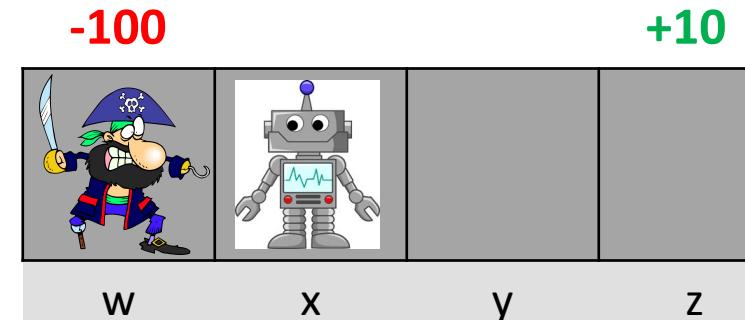
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90,3 & 2,9 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2,4 & 0,2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 8.1$
- $W = 4$
- $t = 0$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

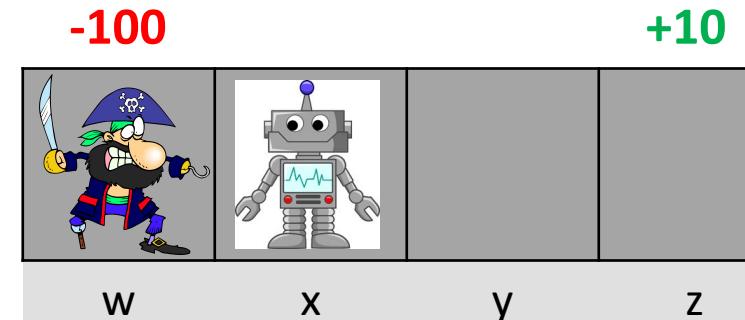
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90, 8.1 & 2, 9 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2, 4 & 0, 2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 8.1$
- $W = 4$
- $t = 0$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

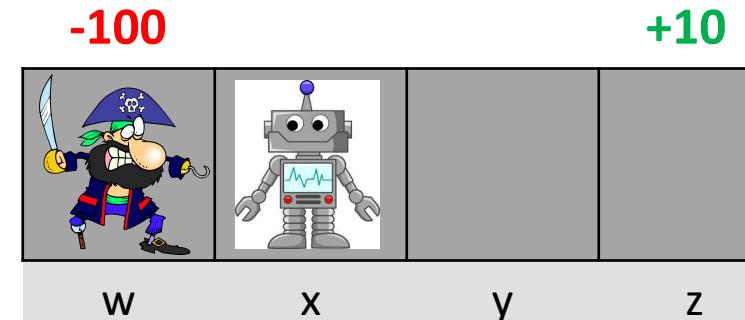
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90, 8.1 & 2, 9 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2, 4 & 0, 2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 8.1$
- $W = 8$
- $t = 0$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

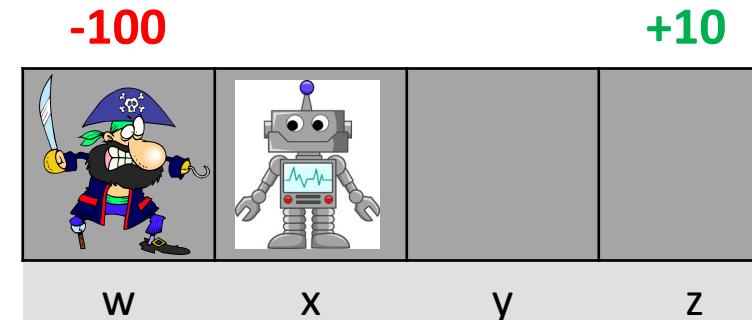
$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

$$\gamma = 0.9$$

MC control + IS example

- $Q = \begin{array}{|c|c|c|c|} \hline -100 & -90, 8.1 & 2, 9 & 10 \\ \hline w & x & y & z \\ \hline \end{array}$
- $C = \begin{array}{|c|c|c|c|} \hline 1 & 2, 4 & 0, 2 & 1 \\ \hline w & x & y & z \\ \hline \end{array}$
- $\pi(s) = \begin{array}{|c|c|c|c|} \hline \text{exit} & & & \text{exit} \\ \hline w & x & y & z \\ \hline \end{array}$
- $b(s) = \forall s \{0.5 \leftarrow, 0.5 \rightarrow\}$
- $\tau = x, \rightarrow, 0, y, \rightarrow 0, z, \text{exit}, 10$

- $G = 8.1$
- $W = 8$
- $t = 0$



Off-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

For $t = T-1, T-2, \dots$ down to 0:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit For loop

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

What did we learn?

- Online evaluation through the Monte-Carlo approach
 - Update V or Q estimations based on observed returns
- On policy Monte-Carlo control
 - Beware of local optimum. Must explore!
 - Consider using a soft policy $\pi(a|s)$
 - Assign soft policy values that mimic an ε -greedy strategy
 - On policy learning is not sample efficient!
- Off policy Monte-Carlo control
 - Define target policy (e.g., $\arg \max_a Q(S_t, a)$) that can be different than the behavior policy
 - Utilize weighted importance sampling to train a target policy
- $v_\pi(s) = \frac{\sum_{m \in M} [\rho_{tm} G_t^m]}{\sum_{m \in M} \rho_{tm}}$



Required Readings

1. Chapter-3,4 of Introduction to Reinforcement Learning, 2nd Ed., Sutton & Barto



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Reinforcement Learning 2022-23 Second Semester, M.Tech (AIML)

Session #9: Temporal Difference Learning

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)



Agenda for the class

- Temporal Difference Learning
 - TD(0)
 - SARSA
 - Q-Learning

Acknowledgements: Some of the slides were adopted with permission from the course [CSCE-689](#) (Texas A&M University) by Prof. Guni Sharon



Solving MDPs so far

Dynamic programming

- ✓ Off policy
- ✓ local learning, propagating values from neighbors (Bootstrapping)
- ✗ Model based

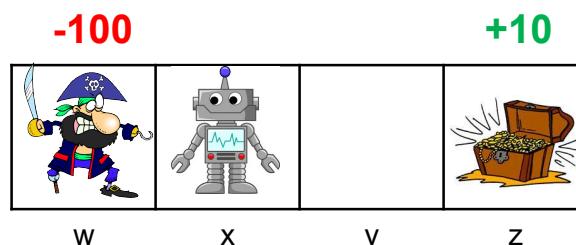
Monte-Carlo

- ✗ On-policy (though importance sampling can be used)
- ✗ Requires a full episode to train on
- ✓ Model free, online learning

- $Q(z, \text{exit}) = 10$
- $Q(y, \rightarrow) = 0 + \gamma \max_a Q(z, a)$
- $Q(x, \rightarrow) = 0 + \gamma \max_a Q(y, a)$

$$q^*(s, a) = \sum_{s'} p(s'|s, a) (r(s, a, s') + \gamma \max_a [q^*(s', a)])$$

$$\gamma = 0.9$$



- Episode = $\{x, y, z, \text{exit}\}$
- $Q(z, \text{exit}) = 10$
- $Q(y, \rightarrow) = 9$
- $Q(x, \rightarrow) = 8.1$



Fuse DP and MC

Dynamic programming

- ✓ Off policy
- ✓ local learning, propagating values from neighbors
(Bootstrapping)
- ✗ Model based

Monte-Carlo

- ✗ On-policy (though importance sampling can be used)
- ✗ Requires a full episode to train on
- ✓ Model free, online learning

TD Learning

- ✓ Off policy
- ✓ local learning, propagating values from neighbors
(Bootsraping)
- ✓ Model free, online learning



Temporal difference learning

- $Q(s, a) = Q(s, a) + \alpha \left(R_{t+1} + \gamma \max_{a'} [q^*(s', a')] - Q(s, a) \right)$
 - But we don't know $q^*(s', a)$
 - Use the learned estimation $Q(s', a)$
 - $Q(s, a) = Q(s, a) + \alpha \left(R_{t+1} + \gamma \max_{a'} [Q(s', a')] - Q(s, a) \right)$
- 
Temporal difference error: δ

A	B	C	D	E
B	C	D	②(C, South, E, 4)	
E			③(C, East, A, 6)	④(B, East, C, 2)

①(B, East, C, 2) [s, a, s', r(s, a, s')] what are the learned value q states after 4 transactions.

initial value = 0 $\gamma = 1, \alpha = 0.5$

$v(s) = v(s) + \alpha (R + \gamma (v(s')) - v(s))$

• Transitions

	A	B	C	D	E
initial	0	0	0	0	0
①	0	1	0	0	0
②	0	1	2	0	0
③	0	1	4	0	0
④	0	3.5	4	0	0

$$v(C) = 2 + 0.5(6 + 1(0) - 2) \\ = 2 + 0.5(4) = 4$$

$$v(B) = 1 + 0.5(2 + 1(4) - 1) \\ = 1 + 0.5(5) = 3.5$$



Temporal difference learning

- $$Q(s, a) = Q(s, a) + \alpha \left(R_{t+1} + \gamma \max_{a'}[Q(s', a')] - Q(s, a) \right)$$
 - $$V(s) = V(s) + \alpha(R_{t+1} + \gamma V(s') - V(s))$$
- Update estimate based on other estimates
- Model free
- Online, incremental learning
- Guaranteed to converge to the true value!
 - Some conditions on the step size, α (see slide #19 in 2Multi_armed_bandits.pptx)
- Usually converges faster than MC methods



SARSA: On-policy TD Control

- $$Q(s, a) = Q(s, a) + \alpha \left(R_{t+1} + \gamma \max_{a'}[Q(s', a')] - Q(s, a) \right)$$
- Replace $\max_{a'}[Q(s', a')]$ with values from the observed transition
 - $< s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, >$
- $$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$
- SARSA converges with probability 1 to an optimal policy and action-values as long as all state-action pairs are visited infinitely often, and the policy converges in the limit to the greedy policy
 - Which can be arranged, for example, with ϵ -greedy policies by setting $\epsilon = 1/t$



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

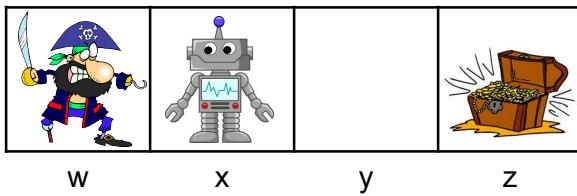
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10





SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

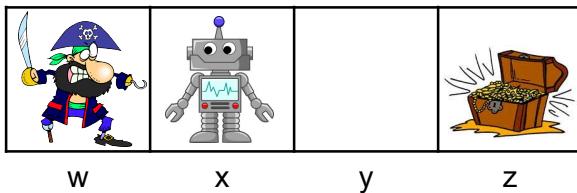
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



$$Q = \begin{array}{cccc} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{array}$$

10



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

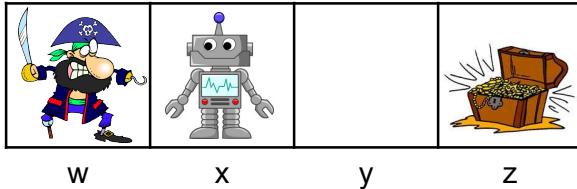
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



x	\leftarrow	0	w	<i>null</i>
S	A	R	S'	A'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{bmatrix}$$



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$\underline{Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]}$$

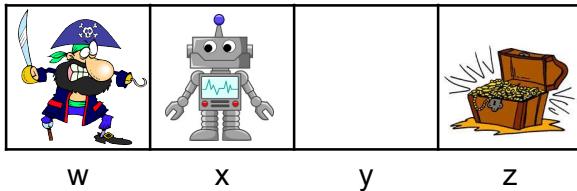
$\underline{S \leftarrow S'; A \leftarrow A'}$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



x	\leftarrow	0	w	<i>exit</i>
S	A	R	S'	A'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{bmatrix}$$



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

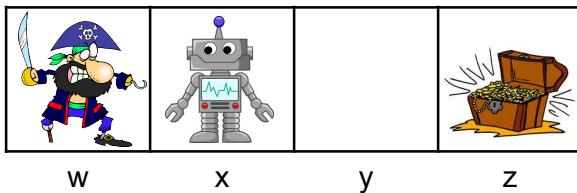
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



x	\leftarrow	0	w	<i>exit</i>
S	A	R	S'	A'

$$Q = \begin{bmatrix} 0 & \mathbf{0,0} & \mathbf{0,0} & 0 \\ w & x & y & z \end{bmatrix}$$



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

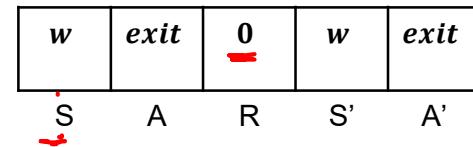
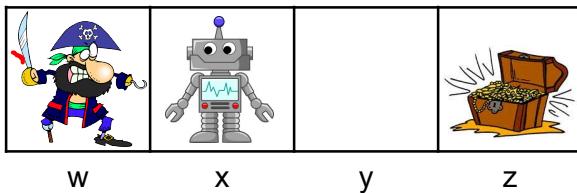
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{bmatrix}$$



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

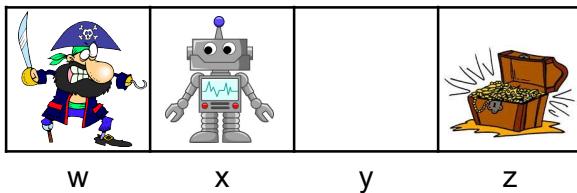
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



w	exit	-100	ter	exit
S	A	R	S'	A'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{bmatrix}$$



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

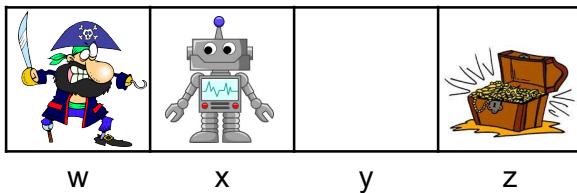
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



w	exit	-100	ter	.
S	A	R	S'	A'

-100	0,0	0,0	0
w	x	y	z



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

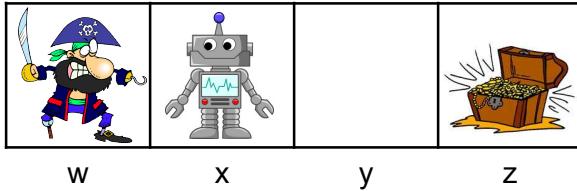
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



x	\leftarrow	-100	<i>ter</i>	.
S	A	R	S'	A'

$Q =$	-100	0,0	0,0	0
	w	x	y	z



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal



$$\gamma = 0.9$$

-100



w

+10



x

y



z

x	\leftarrow	0	w	exit
S	A	R	S'	A'

$$Q =$$

-100	0,0	0,0	0
w	x	y	z



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

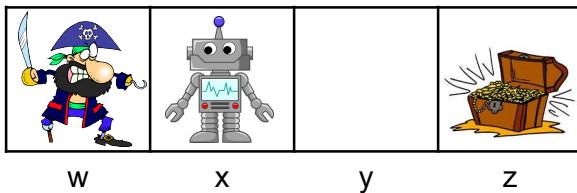
$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$\gamma = 0.9$$

-100

+10



x	\leftarrow	0	w	<i>exit</i>
S	A	R	S'	A'

$$Q = \begin{matrix} -100 & - & 0,0 & 0 \\ w & x & y & z \end{matrix}$$



SARSA: On-policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

Repeat (for each step of episode):

Take action A , observe R, S'

Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$$S \leftarrow S'; A \leftarrow A';$$

until S is terminal

And so on...

-100	$\gamma = 0.9$	$+10$
A cartoon illustration of a pirate wearing a blue tricorn hat with a feather, a black coat with gold buttons, and blue pants. He has a mustache and is holding a cutlass.	A simple grey robot with a rectangular head, two large eyes, and a small mouth. It has a small blue antenna on its head and two grey cylindrical arms.	An open treasure chest filled with gold coins, with a few coins spilling out onto the ground in front of it.

w	<i>exit</i>	0	w	<i>exit</i>
S	A	R	S'	A'

$$Q = \begin{array}{|c|c|c|c|} \hline & -100 & - & 0,0 \\ \hline w & 90,0 & & 0 \\ \hline x & & & \\ \hline y & & & \\ \hline z & & & \\ \hline \end{array}$$



Q-learning: Off-policy TD Control

- Use the original TD update rule
- $$Q(s, a) = Q(s, a) + \alpha \left(R_{t+1} + \gamma \max_{a'} [Q(s', a')] - Q(s, a) \right)$$
- Approximates the state-action value for the optimal policy, i.e., q^*
 - Assuming that every state-action pair is visited infinitely often
- Follows from the proof of convergence for the Bellman function
 - See slides MDPs+DP



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

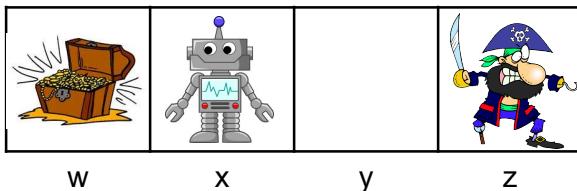
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal

$$\gamma = 0.9$$

+10



x	null	null	null
S	A	R	S'

0	0,0	0,0	0
w	x	y	z



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

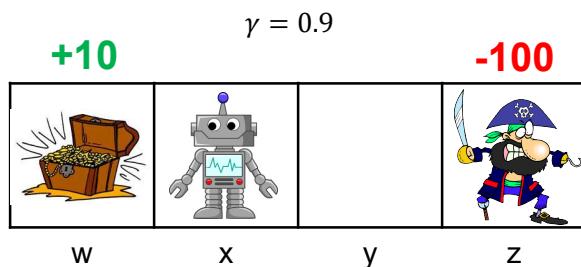
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal



x	\rightarrow	0	y
S	A	R	S'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{bmatrix}$$



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

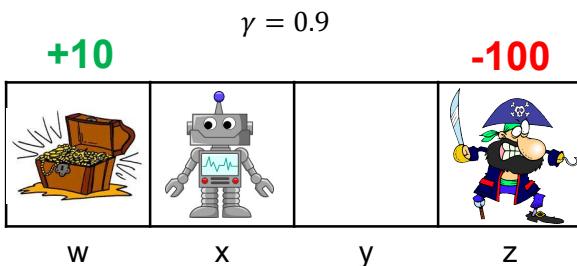
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal



x	\rightarrow	0	y
S	A	R	S'

$Q =$

0	0,0	0,0	0
w	x	y	z



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

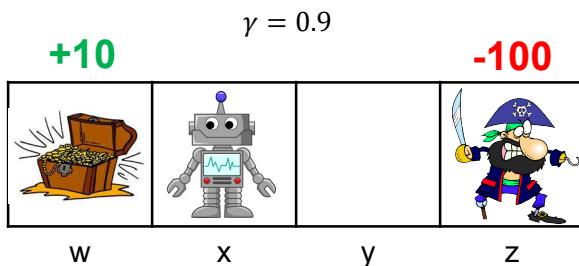
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal



y	\rightarrow	0	y
S	A	R	S'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{bmatrix}$$



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

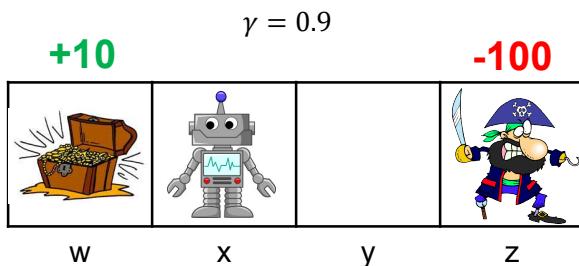
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal



z	\rightarrow	0	z
S	A	R	S'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{bmatrix}$$



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

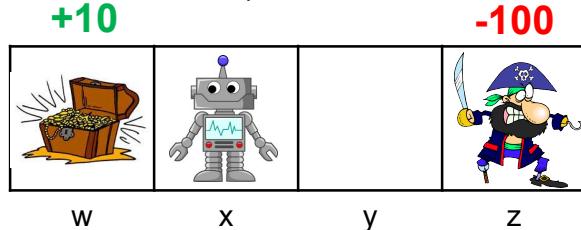
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal



$$\gamma = 0.9$$



z	<i>exit</i>	-100	.
S	A	R	S'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & 0 \\ w & x & y & z \end{bmatrix}$$



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

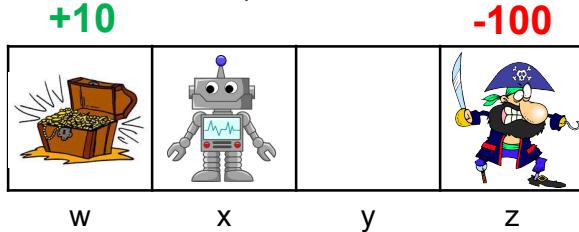
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal



$$\gamma = 0.9$$



z	<i>exit</i>	-100	.
S	A	R	S'

Q =	0	0,0	0,0	-100
	w	x	y	z



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

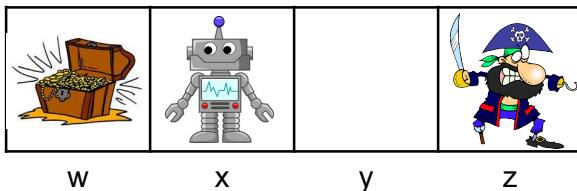
$$S \leftarrow S'$$

 until S is terminal

$$\gamma = 0.9$$

+10

-100



x	<i>exit</i>	-100	.
S	A	R	S'

0	0,0	0,0	-100
w	x	y	z



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

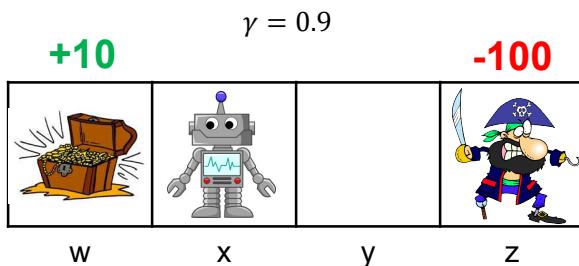
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal



x	\rightarrow	0	y
S	A	R	S'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & -100 \\ w & x & y & z \end{bmatrix}$$



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

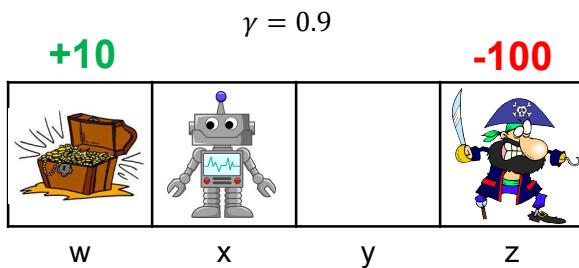
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal



y	\rightarrow	0	z
S	A	R	S'

$$Q = \begin{bmatrix} 0 & 0,0 & 0,0 & -100 \\ w & x & y & z \end{bmatrix}$$



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

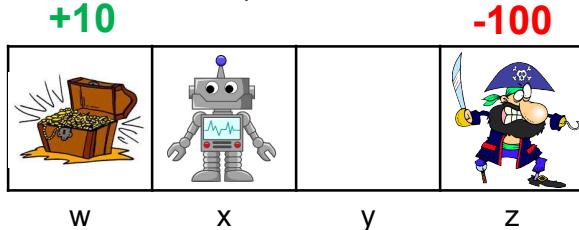
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$

 until S is terminal



$$\gamma = 0.9$$



y	\rightarrow	0	z
s	a	r	s'

$Q =$	0	$0,0$	$0,-90$	-100
	w	x	y	z



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

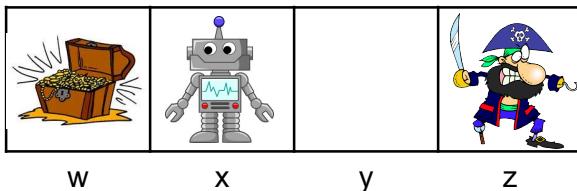
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal

$$\gamma = 0.9$$

+10



x	→	0	z
s	A	R	s'

0	0,0	0,-90	-100
w	x	y	z



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

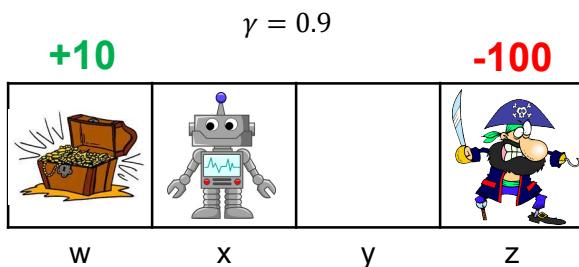
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal



x	→	0	y
S	A	R	S'

$Q =$

0	0,0	0,-90	-100
w	x	y	z



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

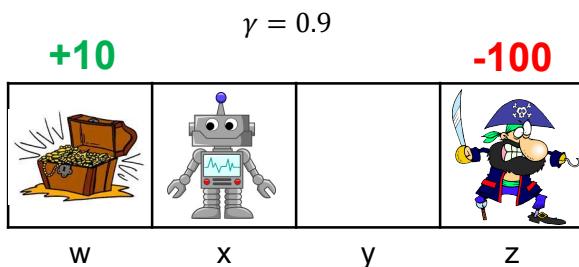
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal



x	\rightarrow	0	y
S	A	R	S'

$Q =$

0	0,0	0,-90	-100
w	x	y	z



Q-learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

until S is terminal

And so on...

	$\gamma = 0.9$
+10	-100
	
w	x
	
	y
	z

x	\rightarrow	0	y
S	A	R	S'

$$Q = \begin{array}{|c|c|c|c|} \hline & 0 & 0,0 & 0,-90 & -100 \\ \hline w & & & & \\ x & & & & \\ y & & & & \\ z & & & & \\ \hline \end{array}$$



Required Readings

1. Chapter-6 of Introduction to Reinforcement Learning, 2nd Ed., Sutton & Barto



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



Deep Reinforcement Learning
2022-23 Second Semester, M.Tech (AIML)

Session #10-11-12: On Policy Prediction with Approximation

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)

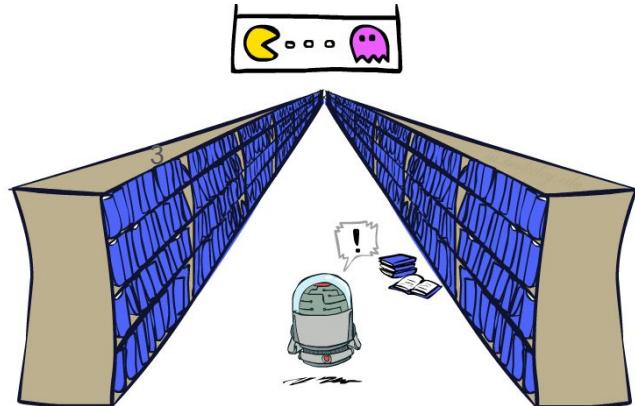
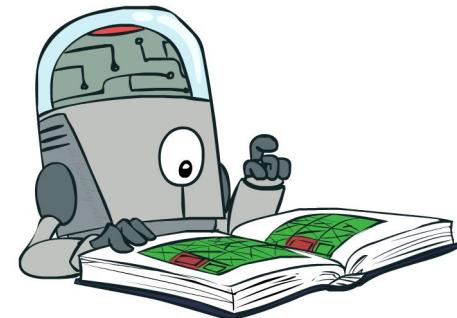


Agenda for the classes

- Introduction
- Value Function Approximation
- Stochastic Gradient, Semi-Gradient Methods
- Role of Deep Learning for Function Approximation;
- Feature Construction Methods

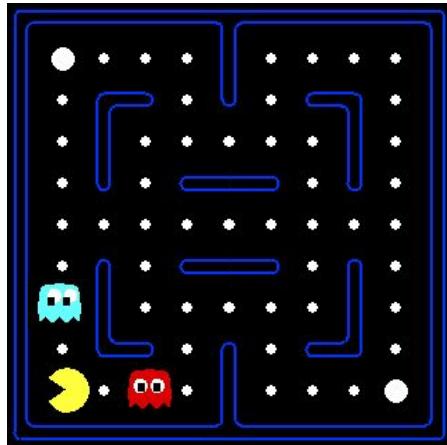
Generalizing Across States

- Tabular Learning keeps a table of all state values
- In realistic situations, we cannot possibly learn about every single state!
 - Too many states to visit them all during training
 - Too many states to hold a value table in memory
- Instead, we want to generalize:
 - Learn about some small number of training states from experience
 - Generalize that experience to new, similar situations
 - This is a fundamental idea in machine learning

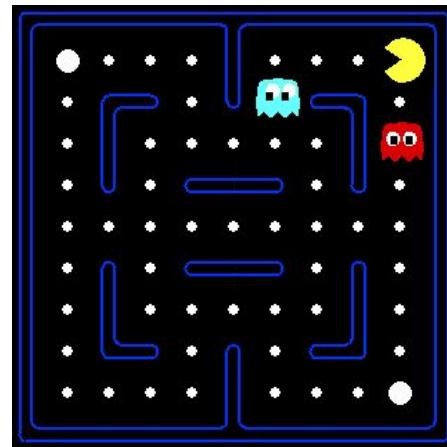


Example: Pacman

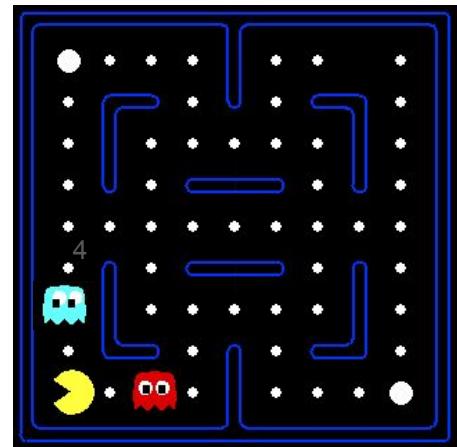
Let's say we discover through experience that this state is bad:



In naïve tabular-learning, we know nothing about this state:



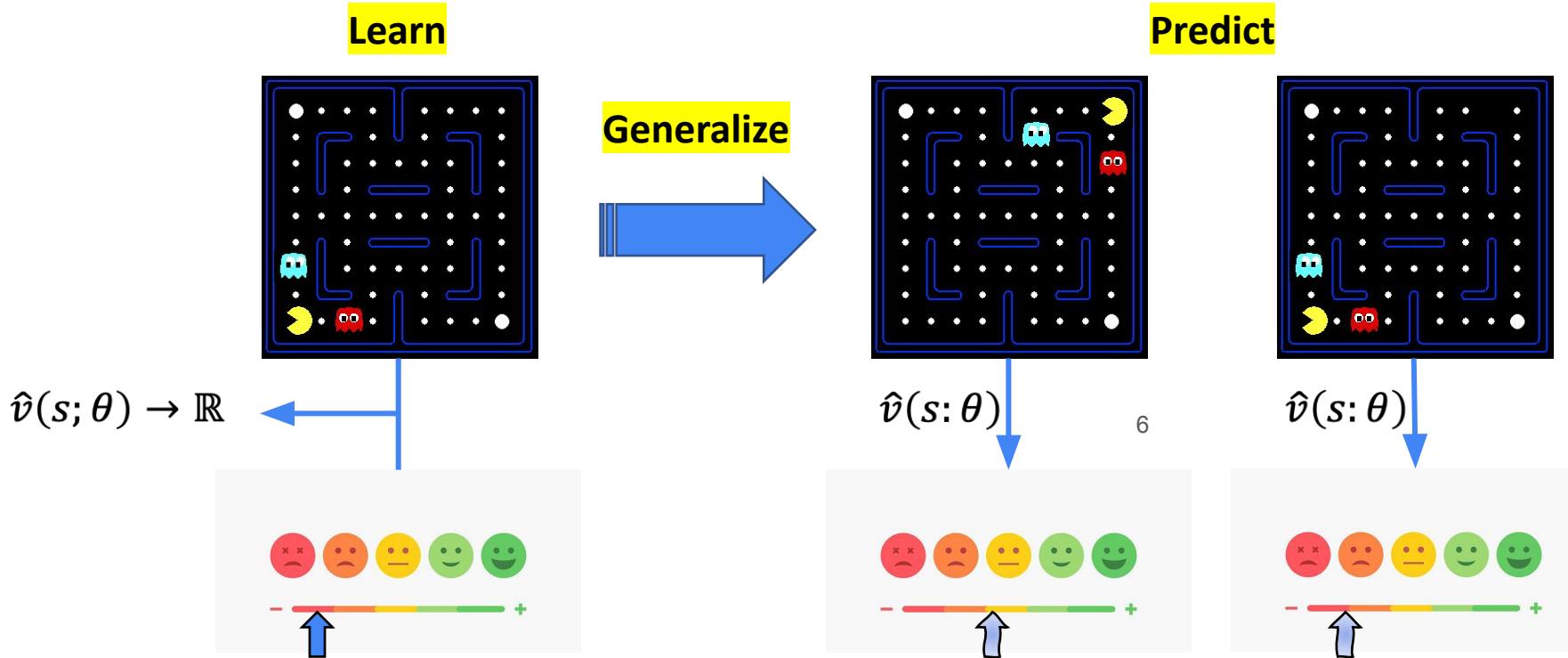
Or even this one!



- Naïve Q-learning
- After 50 training episodes



Learn an approximation function

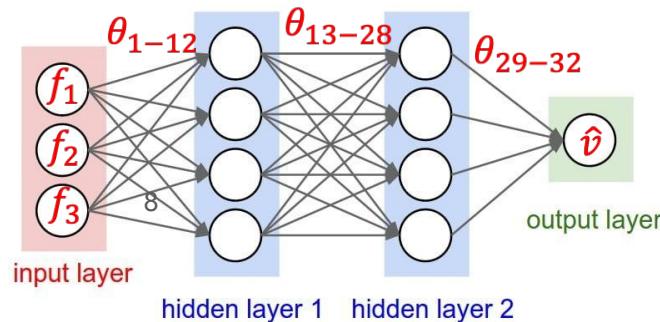


- Q-learning with function approximator



Parameterized function approximator

- Assume that each state is vector of features (f_1, f_2, \dots, f_n) , e.g.,
 - Packman location, Ghost1 location , Ghost2 location, food location
 - Or even screen pixels
- A parametrized value approximator $\hat{v}(s; \theta)$ might look like this:
 - $= \sum_i \theta_i f_i$ or maybe like this $= \prod_i f_i^{\theta_i}$ or even this
- Assume we know the true value for a set of states:
 - $v(S_1) = 5, v(S_2) = 8, v(S_3) = 2$
 - How can we update θ to reflect this information?



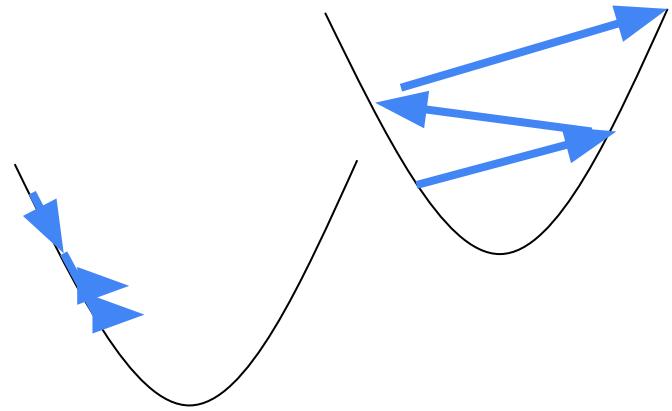
Gradient Descent

- Given: $v(S_1) = 5, v(S_2) = 8, v(S_3) = 2$
- We want to set θ such that $\forall s, \hat{v}(s; \theta) = v(s)$
 - Not possible in the general case, why?
 - Instead we'll try to minimize the errors: loss = $\sum_s |v(s) - \hat{v}(s; \theta)|$
 - Partial derivative of the loss with respect to θ_i = how to change θ_i such that loss will increase the most
 - Go the other way -> decrease loss
 - Ooops! Absolute value is not differentiable -> can't compute gradients
 - Simple fix: loss = $\frac{1}{2} \sum_s [v(s) - \hat{v}(s; \theta)]^2$ = squared loss function

Gradient Decent

- loss = $\frac{1}{2} \sum_s [v(s) - \hat{v}(s; \theta)]^2$

- For each i
 - Push θ_i towards a direction that minimizes loss
 - $\theta_i = \theta_i - \frac{\partial \text{loss}}{\partial \theta_i}$
- More generally $\theta = \theta - \alpha \nabla \text{loss}$
 - $\nabla \text{loss} = \left(\frac{\partial \text{loss}}{\partial \theta_1}, \frac{\partial \text{loss}}{\partial \theta_2}, \dots, \frac{\partial \text{loss}}{\partial \theta_n} \right)$
 - α is the learning rate, requires tuning per domain, too large learning diverges to small results in slow learning or even premature convergence

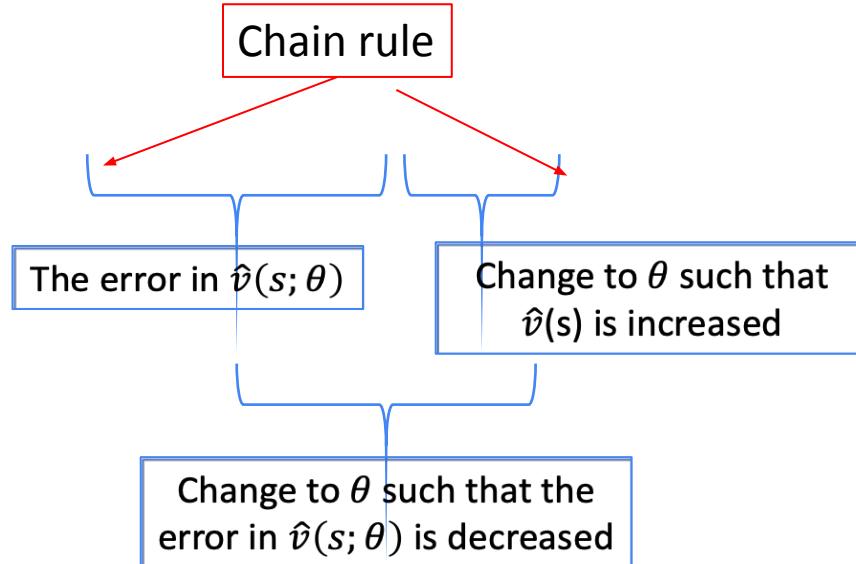


Stochastic Gradient Descent

Gradient Decent

- loss = $\frac{1}{2} \sum_s [v(s) - \hat{v}(s; \theta)]^2$

- Gradient Decent: $\theta = \theta - \alpha \nabla_{\theta} loss = \theta - \alpha \nabla_{\hat{v}} loss \cdot \nabla_{\theta} \hat{v}$
 - $\theta = \theta + \alpha \sum_s [v(s) - \hat{v}(s; \theta)] \nabla \hat{v}(s; \theta)$



Gradient Descent

- Idea:
 - Start somewhere
 - Repeat: Take a step in the gradient direction

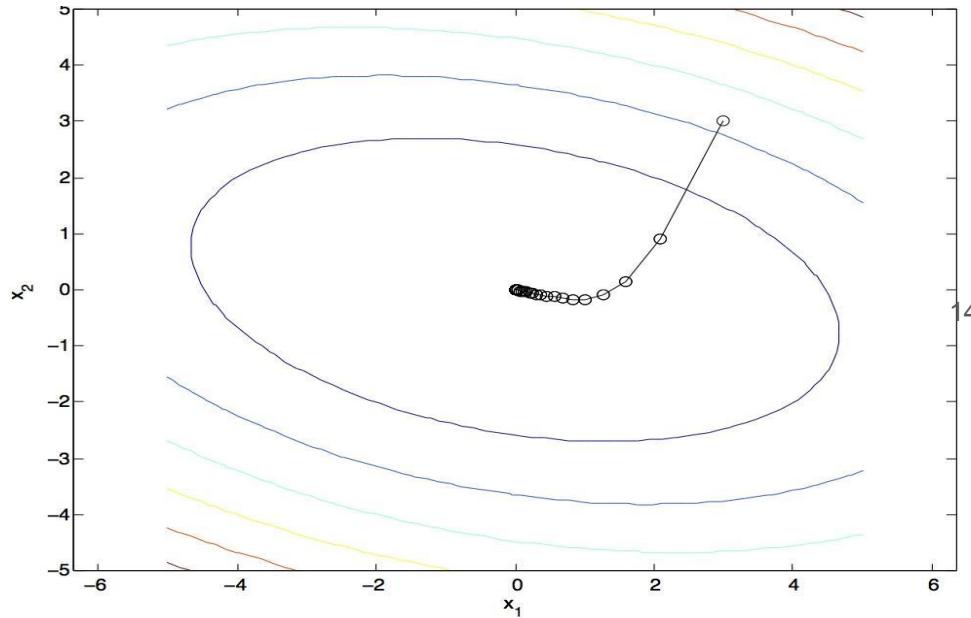


Figure source: Mathworks

Batch Gradient Decent

Minimize squared loss: $l(\theta) = \frac{1}{2} \sum_s [v(s) - \hat{v}(s; \theta)]^2$

- **init**
- **for iter = 1, 2, ...**
 - $\theta = \theta + \alpha \sum_s [v(s) - \hat{v}(s; \theta)] \nabla \hat{v}(s; \theta)$

Stochastic Gradient Decent (SGD)

Minimize squared loss: $l(\theta) = \frac{1}{2} \sum_s [v(s) - \hat{v}(s; \theta)]^2$

Observation: once gradient on one training example has been computed, might as well incorporate before computing next one

- `init θ`
- `for iter = 1, 2, ...`
 - `pick random j`
 - $\theta = \theta + \alpha [v(s_j) - \hat{v}(s_j; \theta)] \nabla \hat{v}(s_j; \theta)$

Mini-Batch Gradient Decent

Minimize squared loss: $l(\theta) = \frac{1}{2} \sum_s [v(s) - \hat{v}(s; \theta)]^2$

Observation: gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- init θ
- for iter = 1, 2, ...
 - pick random subset of training examples J^{17}
 - $\theta = \theta + \alpha \sum_{s \in J} [v(s) - \hat{v}(s; \theta)] \nabla \hat{v}(s; \theta)$

SGD for Monte Carlo estimation

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Initialize value-function weights \mathbf{w} as appropriate (e.g., $\mathbf{w} = \mathbf{0}$)

Repeat forever:

Generate an episode $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$ using π

For $t = 0, 1, \dots, T - 1$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

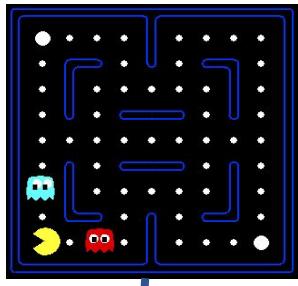
w are the tunable parameters of the value approximation function

18

- Guaranteed to converge to a local optimum because G_t is an unbiased estimate of $v_\pi(S_t)$

Example

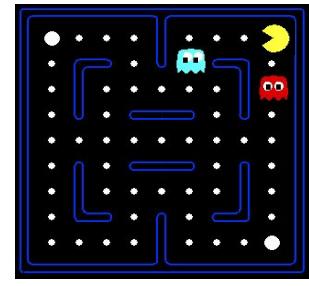
$$f(S) = [2,2,1]$$



10

- $S = \{f_1(S), f_2(S), f_3(S)\}$
 - $f_{1,2}$ =distance to ghost 1,2, f_3 =distance to food
- $\hat{v}(s) = \sum_i \theta_i f_i(s)$
 - init: $\theta = [0,0,0]$
- $\theta = \theta + \alpha(G_t - \hat{v}(s; \theta))\nabla \hat{v}(s; \theta)$
- $\theta = [0,0,0] + 0.1(10 - [0,0,0] \cdot [2,2,1])[2,2,1]$
 - $\theta = [2,2,1]$
- $\hat{v}(S') = f(S') \cdot \theta = [2,4,1] \cdot [2,2,1] = 13$

$$f(S') = [2,4,1]$$



19

Learning approximation with bootstrapping

- Can we update the value approximation function at every step?
- Yes, define SGD as a function of the TD error
 - Tabular TD learning: $\hat{v}(s_t) = \hat{v}(s_t) + \alpha(r_t + \gamma \hat{v}(s_{t+1}) - \hat{v}(s_t))$
 - Approximation TD learning: $\theta = \theta + \alpha(r_t + \gamma \hat{v}(s_{t+1}; \theta) - \hat{v}(s_t; \theta)) \nabla \hat{v}(s_t; \theta)$
- Known as Semi-gradient methods
- **NOT** guaranteed to converge to a local optimum because $\hat{v}(s_{t+1}; \theta)$ is a biased estimate of $v_\pi(s_{t+1})$
- Semi-gradient (bootstrapping) methods do not converge as robustly as (full) gradient methods

Semi-gradient methods

- They do converge reliably in important cases such as the linear approximation case
- They offer important advantages that make them often clearly preferred
- They typically enable significantly faster learning, as we have seen in Chapters 6 and 7
- They enable learning to be continual and online, without waiting for the end of an episode
- This enables them to be used on continuing problems and provides computational advantages

Semi-gradient TD(0)

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Initialize value-function weights \mathbf{w} arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose $A \sim \pi(\cdot | S)$

 Take action A , observe R, S'

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})] \nabla \hat{v}(S, \mathbf{w})$$

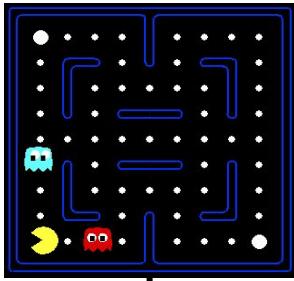
$S \leftarrow S'$

 until S' is terminal

What's the difference
from the tabular case?

Example

$$f(S) = [2,3,1]$$



$$R = +10$$

$$f(S') = [1,2,1]$$



- $S = \{f_1(S), f_2(S), f_3(S)\}$

- $f_{1,2}$ =distance to ghost 1,2, f_3 =distance to food

- $\hat{v}(s) = \sum_i \theta_i f_i(s)$

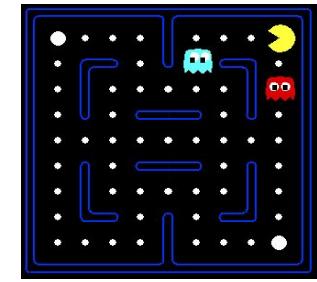
- init: $\theta = [0,0,0]$

- $\theta = \theta + \alpha(R + \gamma \hat{v}(S'; \theta) - \hat{v}(S; \theta)) \nabla \hat{v}(S; \theta)$

- $\theta = [0,0,0] + 0.1(10 + [1,2,1] \cdot [0,0,0] - [2,3,1] \cdot [0,0,0])[2,3,1]$
 - $\theta = [2,3,1]$

- $\hat{v}(U) = f(U) \cdot \theta = [2,4,1] \cdot [2,3,1] = 17$

$$f(U) = [2,4,1]$$



²³

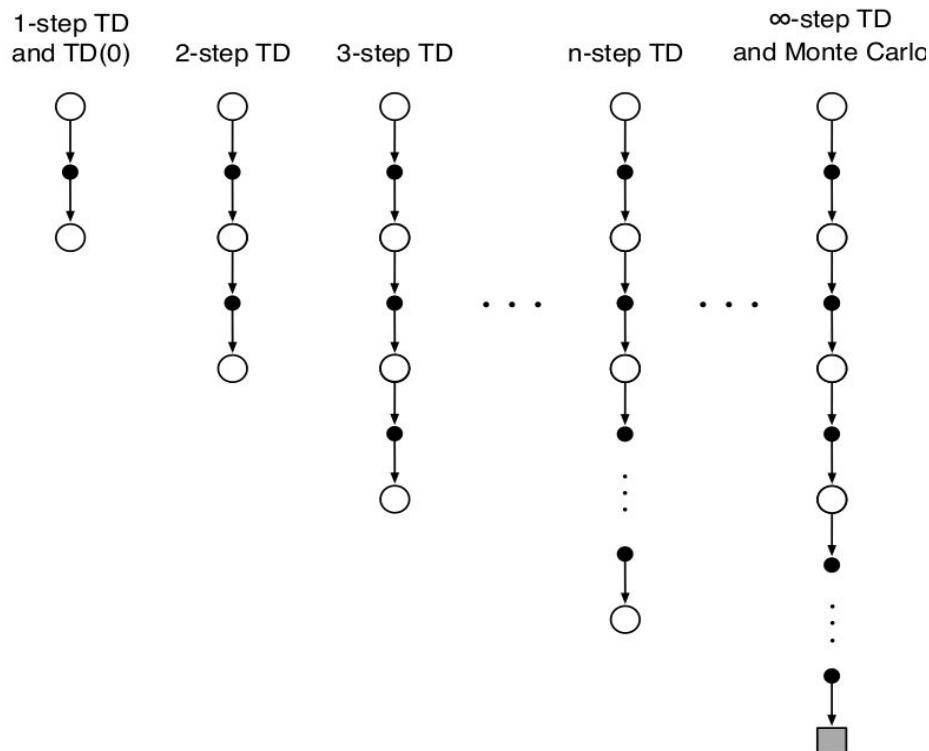
[Review] n-step TD Prediction

One-step return:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

Two-step return:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$



Ref Section 7.1 of TB

[Review] n-step TD Prediction

One-step return:

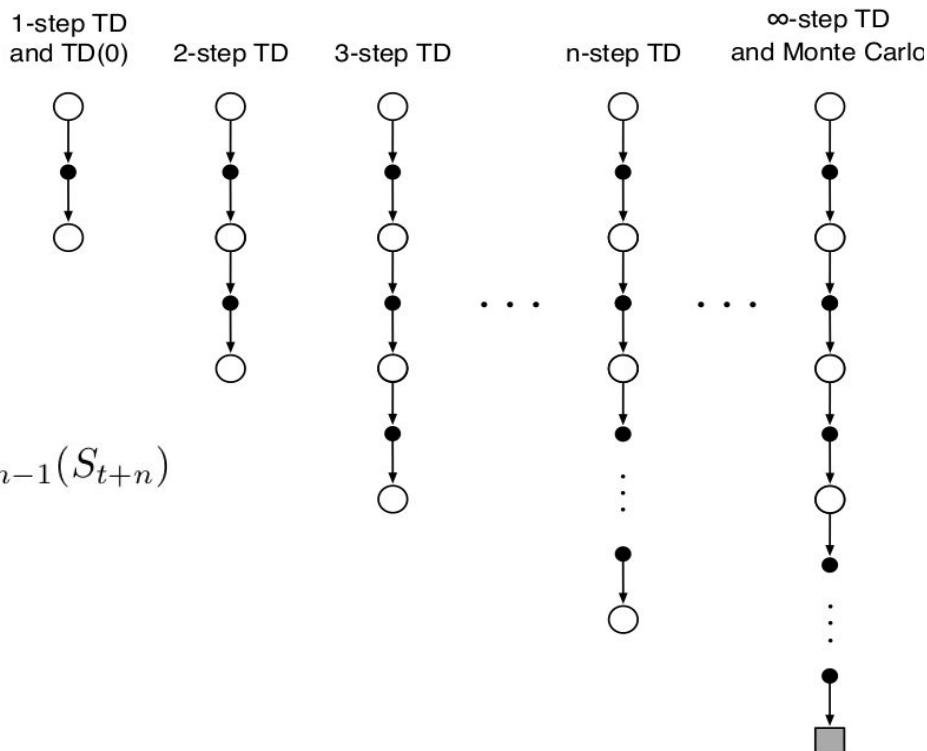
$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

Two-step return:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

n-step return:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$



Ref Section 7.1 of TB

[Review] n-step TD Prediction

One-step return:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

Two-step return:

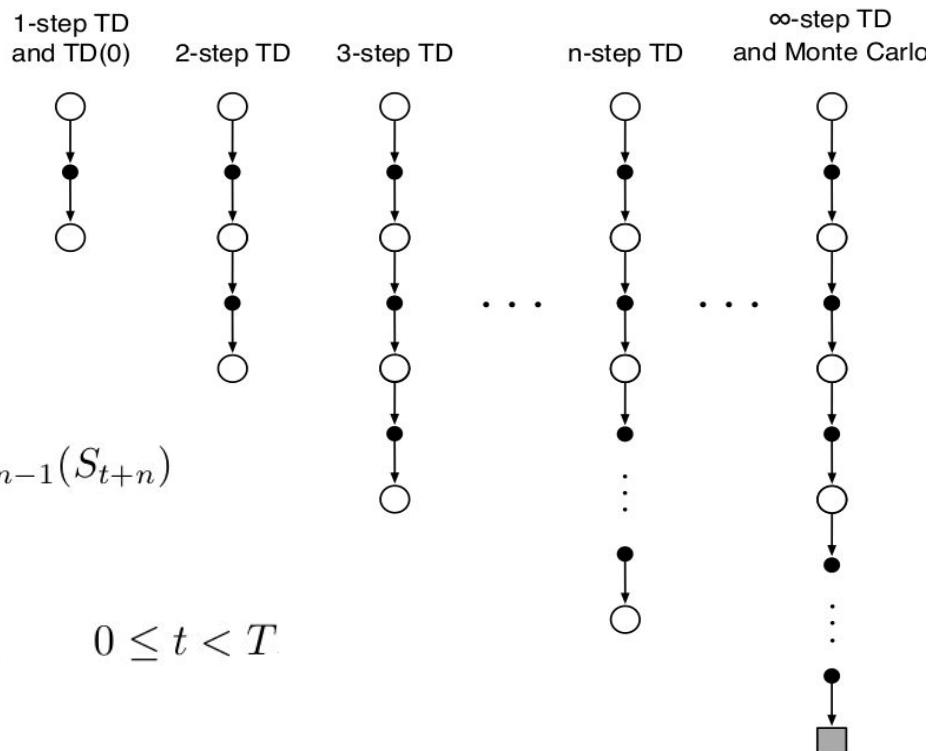
$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

n-step return:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

State-value learning algorithm for using n-step returns:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha [G_{t:t+n} - V_{t+n-1}(S_t)], \quad 0 \leq t < T$$



Ref Section 7.1 of TB

[Review] n-step TD Prediction

n-step TD for estimating $V \approx v_\pi$

Input: a policy π

Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n

Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$

All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:

Initialize and store $S_0 \neq \text{terminal}$

$$T \uparrow \infty$$

Loop for $t = 0, 1, 2, \dots$:

| If $t < T$, then:

Take an action according to $\pi(\cdot | S_t)$

Observe and store the next reward as R_{t+1} and the next state as S_{t+1} .

If S_{t+1} is terminal, then $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)

If $\tau \geq 0$:

$$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$$

If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$

$$V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$$

Until $\tau = T - 1$

n-step return

n -step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Parameters: step size $\alpha \in (0, 1]$, a positive integer n

All store and access operations (S_t and R_t) can take their index mod n

Initialize value-function weights \mathbf{w} arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Repeat (for each episode):

Initialize and store $S_0 \neq \text{terminal}$

$T \leftarrow \infty$

For $t = 0, 1, 2, \dots$:

If $t < T$, then:

Take an action according to $\pi(\cdot|S_t)$

Observe and store the next reward

If S_{t+1} is terminal, then $T \leftarrow t + 1$

$$\tau \leftarrow t - r$$

$$G \leftarrow \sum_{i=1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} B_i$$

If $\tau + n \leq T$ then: $G \leftarrow G + \gamma^n \hat{v}(S_{\tau+1:n} - \mathbf{w})$

(G)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G - \hat{v}(S_{-}, \mathbf{w})] \nabla \hat{v}(S_{-}, \mathbf{w})$$

Until $\tau = T - 1$

- Again, only a simple modification over the tabular setting

n-step return

n -step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Parameters: step size $\alpha \in (0, 1]$, a positive integer n

All store and access operations (S_t and R_t) can take their index mod n

Initialize value-function weights \mathbf{w} arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Repeat (for each episode):

Initialize and store $S_0 \neq$ terminal

$$T \leftarrow \infty$$

For $t = 0, 1, 2, \dots$

If $t < T$, then:

Take an action according to $\pi(\cdot|S_t)$

Observe and store the next reward as R_{t+1} and the next state as S_{t+1}

If S_{t+1} is terminal, then $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)

If $\tau > 0$:

$$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$$

If $\tau + n < T$, then: $G \leftarrow G + \gamma^n \hat{v}(S_{\tau+n}, \mathbf{w})$

$$(G_{\tau:\tau+n})$$

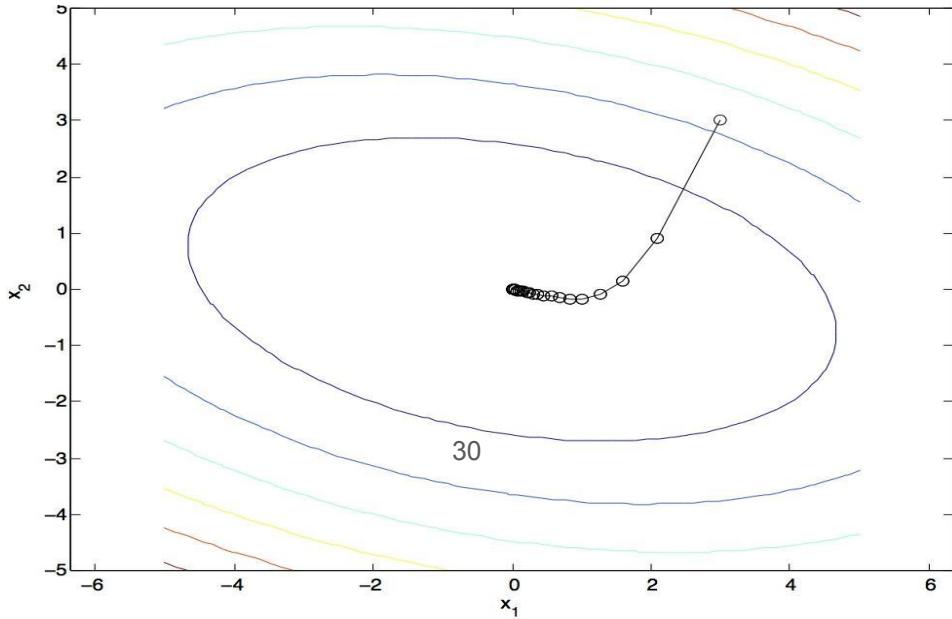
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G - \hat{v}(S_\tau, \mathbf{w})] \nabla \hat{v}(S_\tau, \mathbf{w})$$

Until $\tau \equiv T = 1$

- Again, only a simple modification over the tabular setting
 - Weight update instead of tabular entry update

Another optimization approach

- Approach1: Gradient decent
 - Update θ in iterations
 - Find a fixed point
 - If: $\theta_{t+1} = \theta_t$
 - Then: converged to a local optimum
- Approach2: Compute the fixed point directly
 - Solve: $\theta_{t+1} = \theta_t$



Compute fixed point over θ

- Assume a linear function approximator $\hat{v}(f(s); \theta) = f(s) \cdot \theta$
- TD update: $\theta_{t+1} = \theta_t + \alpha[R_{t+1} + \gamma\hat{v}(S_{t+1}) - \hat{v}(S_t)]\nabla\hat{v}(S_t)$
- $= \theta_t + \alpha[R_{t+1} + (\gamma f(S_{t+1}) - f(S_t))\theta_t]f(S_t)$
- $= \theta_t + \alpha(b - A\theta_t)$
- Where: $b = \mathbb{E}[R_{t+1}f(S_t)]$, $A = \mathbb{E}\left[f(S_t)(f(S_t) - \gamma f(S_{t+1}))^\top\right]$
- Fixed point at: $\mathbb{E}[\theta_{t+1}] = \mathbb{E}[\theta_t]$
 - $b - A\theta_t = 0$ TD-error = 0
 - $b = A\theta_t$
 - $\theta_t = A^{-1}b$

For a linear
approximator

Least squares TD

- Approximate A and b online, solve $\theta = A^{-1}b$
- Where: $b = \mathbb{E}[R_{t+1}f(S_t)]$, $A = \mathbb{E}\left[f(S_t)(f(S_t) - \gamma f(S_{t+1}))^\top\right]$
- $\hat{A} = \sum_{k=0}^{t-1} f(S_t)(f(S_t) - \gamma f(S_{t+1}))^\top$
- But \hat{A} is not guaranteed to be invertible (it might have '0' on diagonal)
- So add a small constant (ε) to the diagonal
 - $\hat{A} = \sum_t f(S_t)(f(S_t) - \gamma f(S_{t+1}))^\top + \varepsilon I$
- $\hat{b} = \sum_t R_{t+1}f(S_t)$

Least squares TD

LSTD for estimating $\hat{v} \approx v_\pi$ ($O(d^2)$ version)

Input: feature representation $\mathbf{x}(s) \in \mathbb{R}^d$, for all $s \in \mathcal{S}$, $\mathbf{x}(\text{terminal}) \doteq \mathbf{0}$

$$\widehat{\mathbf{A}}^{-1} \leftarrow \varepsilon^{-1} \mathbf{I}$$
$$\widehat{\mathbf{b}} \leftarrow \mathbf{0}$$

Store the inverse of A
instead of A

An $d \times d$ matrix

An d -dimensional vector

Repeat (for each episode):

Initialize S ; obtain corresponding \mathbf{x}

Repeat (for each step of episode):

Choose $A \sim \pi(\cdot|S)$

Take action A , observe R, S' ; obtain corresponding \mathbf{x}'

$$\mathbf{v} \leftarrow \widehat{\mathbf{A}}^{-1}^\top (\mathbf{x} - \gamma \mathbf{x}')$$

$$\widehat{\mathbf{A}}^{-1} \leftarrow \widehat{\mathbf{A}}^{-1} - (\widehat{\mathbf{A}}^{-1} \mathbf{x}) \mathbf{v}^\top / (1 + \mathbf{v}^\top \mathbf{x})$$

$$\widehat{\mathbf{b}} \leftarrow \widehat{\mathbf{b}} + R \mathbf{x}$$

$$\boldsymbol{\theta} \leftarrow \widehat{\mathbf{A}}^{-1} \widehat{\mathbf{b}}$$

$$S \leftarrow S'; \mathbf{x} \leftarrow \mathbf{x}'$$

until S' is terminal

Least squares TD

LSTD for estimating $\hat{v} \approx v_\pi$ ($O(d^2)$ version)

Input: feature representation $\mathbf{x}(s) \in \mathbb{R}^d$, for all $s \in \mathcal{S}$, $\mathbf{x}(\text{terminal}) \doteq \mathbf{0}$

$$\widehat{\mathbf{A}}^{-1} \leftarrow \varepsilon^{-1} \mathbf{I}$$

An $d \times d$ matrix

$$\widehat{\mathbf{b}} \leftarrow \mathbf{0}$$

An d -dimensional vector

Repeat (for each episode):

 Initialize S ; obtain corresponding \mathbf{x}

 Repeat (for each step of episode):

 Choose $A \sim \pi(\cdot|S)$

 Take action A , observe R, S' ; obtain corresponding \mathbf{x}'

$$\mathbf{v} \leftarrow \widehat{\mathbf{A}}^{-1}^\top (\mathbf{x} - \gamma \mathbf{x}')$$

$$\widehat{\mathbf{A}}^{-1} \leftarrow \widehat{\mathbf{A}}^{-1} - (\widehat{\mathbf{A}}^{-1} \mathbf{x}) \mathbf{v}^\top / (1 + \mathbf{v}^\top \mathbf{x})$$

$$\widehat{\mathbf{b}} \leftarrow \widehat{\mathbf{b}} + R \mathbf{x}$$

$$\boldsymbol{\theta} \leftarrow \widehat{\mathbf{A}}^{-1} \widehat{\mathbf{b}}$$

$$S \leftarrow S'; \mathbf{x} \leftarrow \mathbf{x}'$$

until S' is terminal

34

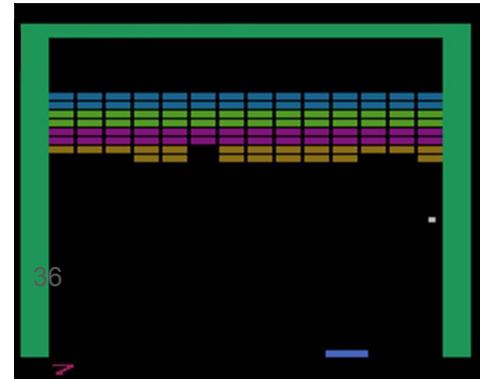
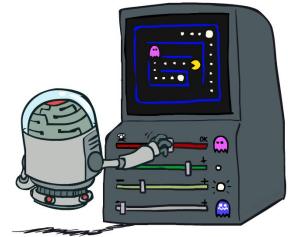
Incremental updates
(no need to store all
previous transitions)

Least squares TD

- Directly computing the TD fixed point
- Most data efficient form of linear TD(0), but it is also more expensive computationally
 - Semi-gradient linear TD(0) requires memory and per-step computation that is only $O(d)$ where d is the number of state features
- In the incremental update version, \hat{A} is an outer product (a column vector times a row vector) and thus requires a matrix update
- The update computational complexity is $O(d^2)$, and the memory complexity is $O(d^2)$

Feature selection

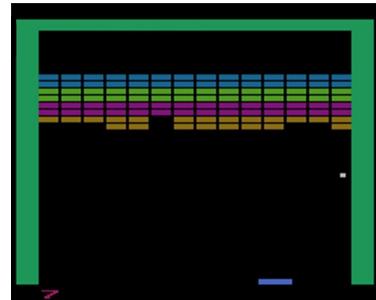
- Assume a linear function approximator $\hat{v}(f(s); \theta) = f(s) \cdot \theta$
- What relevant features should represent states?



Features are domain depended
requiring expert knowledge

Automatic features extraction

- Consider a game state that is given as a bit map
- Raw data of type $pixel(7,3) = [0,0,0]$ (black)
- Desired features = {ball location, ball speed, ball direction, pan location...}
- How can we translate pixels to the relevant features?



Automatic features extraction for linear approximator

- **Polynomials:** $f_i(s) = \prod_{j=1}^k x_j^{c_{i,j}}$

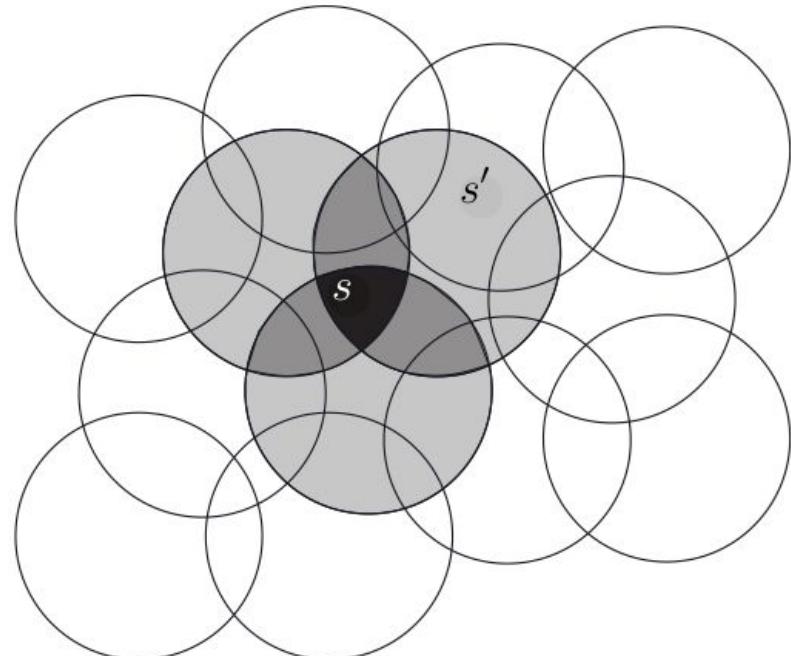
- where each $c_{i,j}$ is an integer in the set $\{0, 1, \dots, n\}$ for an integer $n \geq 0$
- These features makeup the order- n polynomial basis for dimension k , which contains $(n + 1)^k$ different features

- **Fourier Basis:** $f_i(s) = \cos(\pi X^\top c^i)$

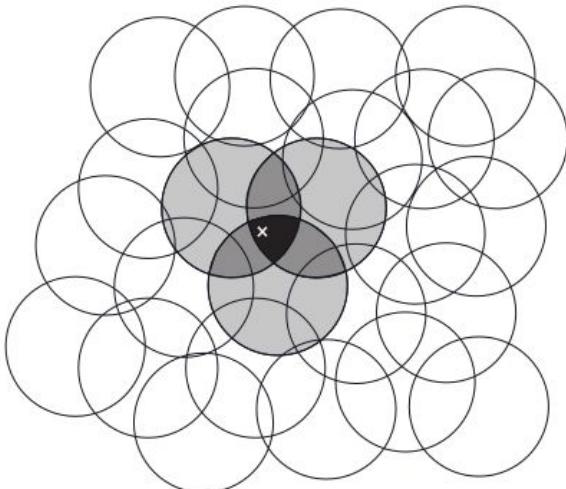
- Where $c^i = (c_1^i, \dots, c_k^i)^\top$, with $c_j^i \in \{0, \dots, n\}$ for $j = \{1, \dots, k\}$ and $i = \{0, \dots, (n + 1)^k\}$
- This defines a feature for each of the $(n + 1)^k$ possible integer vectors c^i
- The inner product $X^\top c^i$ has the effect of assigning an integer in $\{0, \dots, n\}$ to each dimension of X
- This integer determines the feature's frequency along that dimension
- The features can be shifted and scaled to suit the bounded state space of a particular application

Automatic features extraction for linear approximator - Coarse Coding

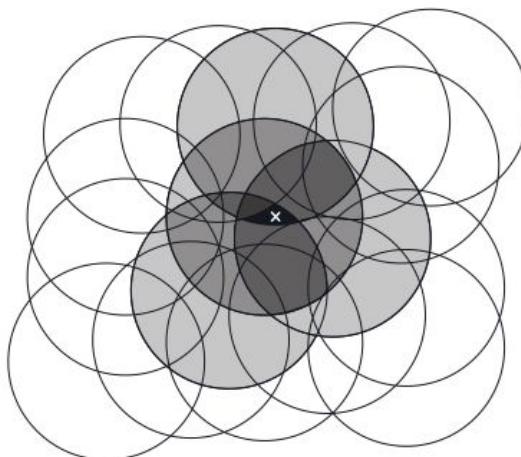
- Natural representation of the state set is continuous
- In 2-d, features corresponding to circles in state space
- Coding of a state:
 - If the state is inside a circle, then the corresponding feature has the value 1
 - otherwise the feature is 0
- Corresponding to each circle is a single weight (a component of w) that is learned
 - Training a state affects the weights of all the intersecting circles.



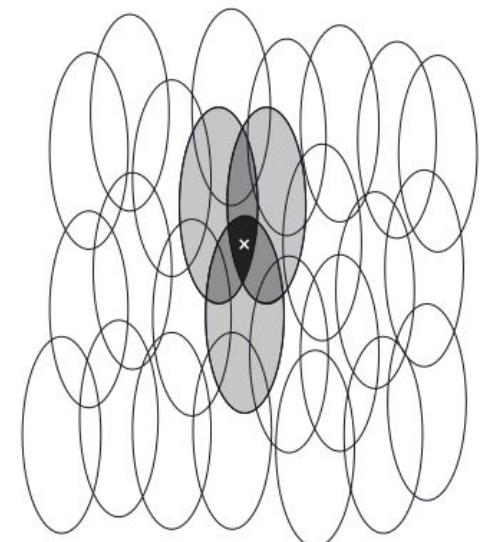
Automatic features extraction for linear approximator - Coarse Coding



Narrow generalization

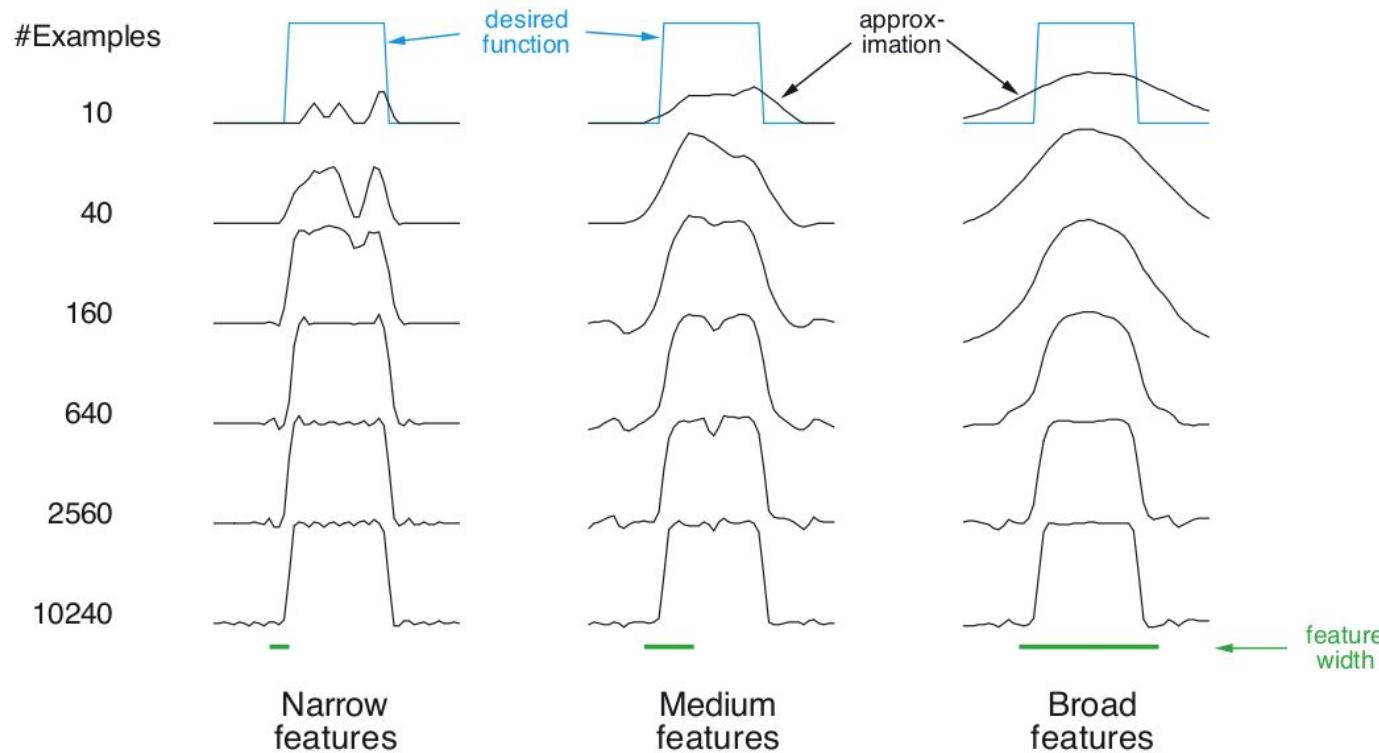


Broad generalization

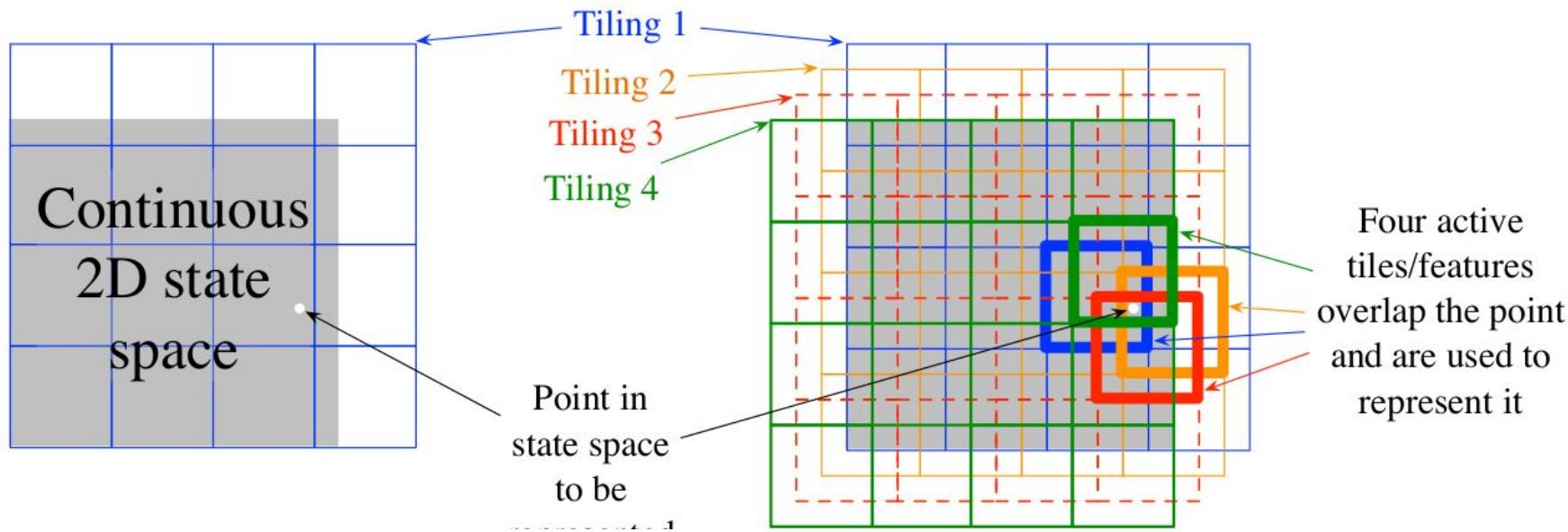


Asymmetric generalization

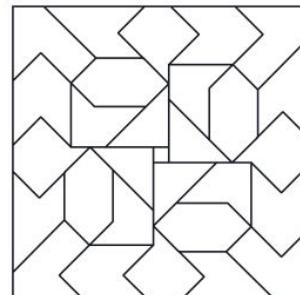
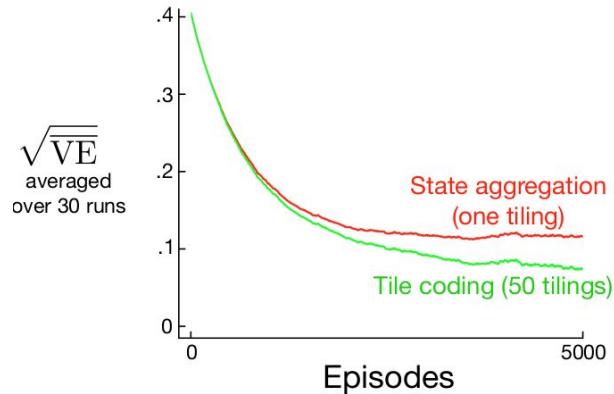
Automatic features extraction for linear approximator - Coarse Coding



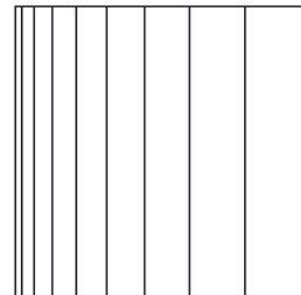
Automatic features extraction for linear approximator - Tile Coding



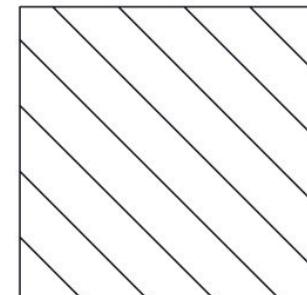
Automatic features extraction for linear approximator - Tile Coding



Irregular

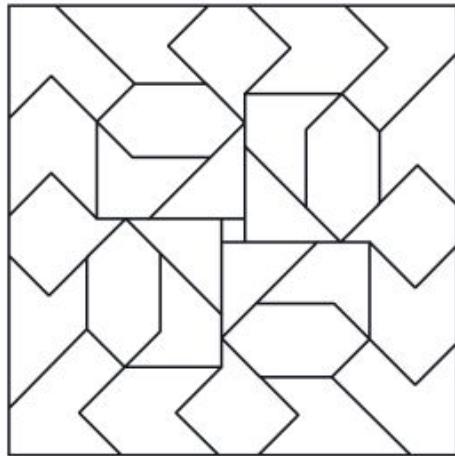


Log stripes

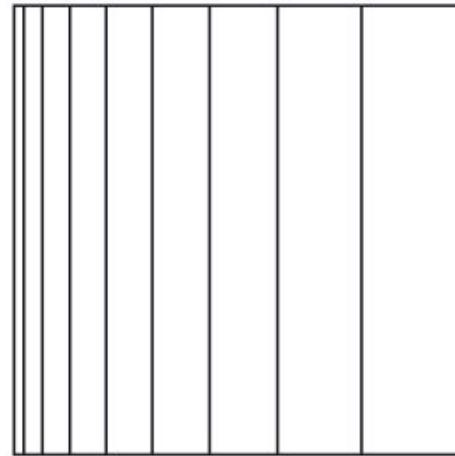


Diagonal stripes

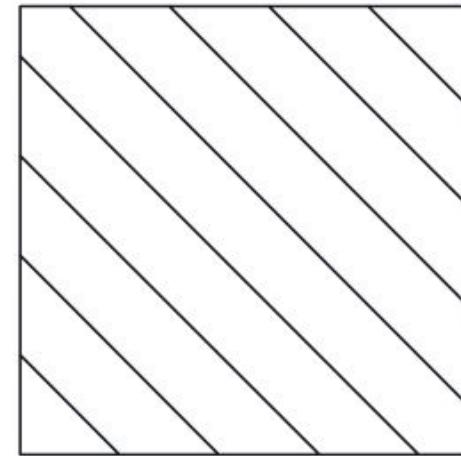
Automatic features extraction for linear approximator - Tile Coding



Irregular



Log stripes

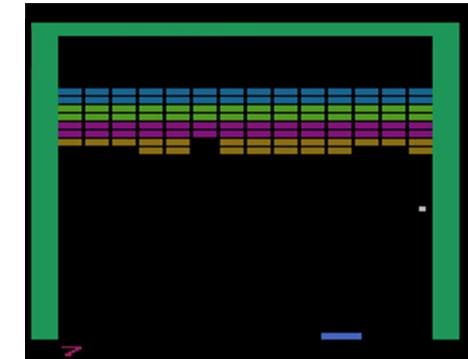


Diagonal stripes

Automatic features extraction for linear approximator

- Other approaches include: Coarse Coding, Tile Coding, Radial Basis Functions (See chapter 9.5 in textbook)
- Each of these approaches defines a set of features, some useful yet most are not
 - E.g., is there a polynomial/Fourier function that translates pixels to pan location?
 - Probably but it's a needle in a (combinatorial) haystack
- Can we do better (generically)
 - Yes, using deep neural networks...

45



What did we learn?

- Reinforcement learning must generalize on observed experience if it is to be applicable to real world domains
- We can use parameterized function approximation to represent our knowledge about the domain state/action values
- Use stochastic gradient descend to update the tunable parameters such that the observed (TD, rollout) error is reduced
- When using a linear approximator, the Least squares TD method provides the most sample efficient approximation



Part-2 DQN

Mnih et al. 2015

- First deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning
- The model is a convolutional neural network, trained with a variant of Q-learning
- Input is raw pixels and output is an action-value function estimating future rewards
- Surpassed a human expert on various Atari video games

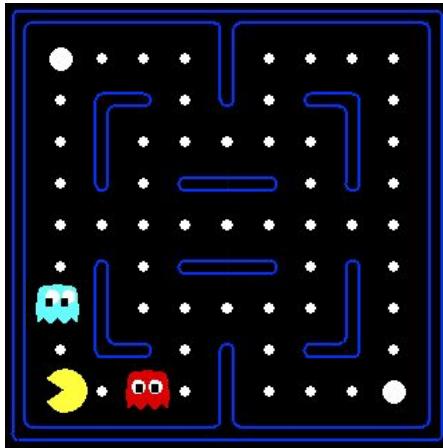
The age of deep learning

- Previous models relied on hand-crafted features combined with linear value functions
 - The performance of such systems heavily relies on the quality of the feature representation
- Advances in deep learning have made it possible to automatically extract high-level features from raw sensory data

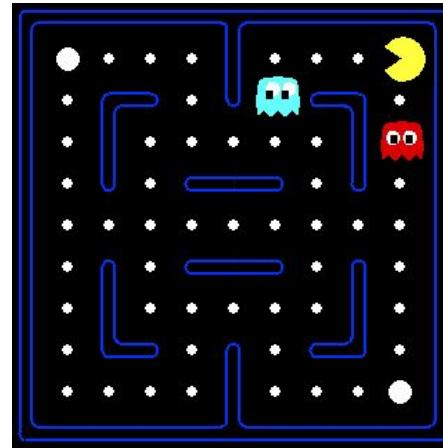


Example: Pacman

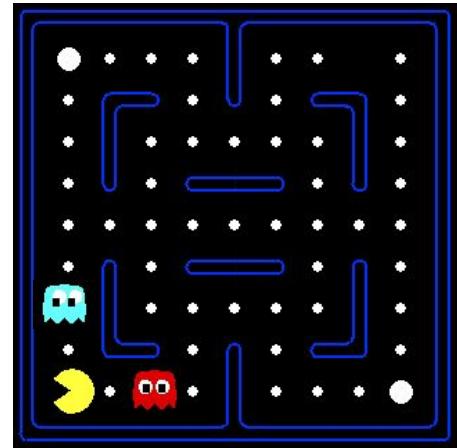
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



Or even this one!



We must generalize our knowledge!

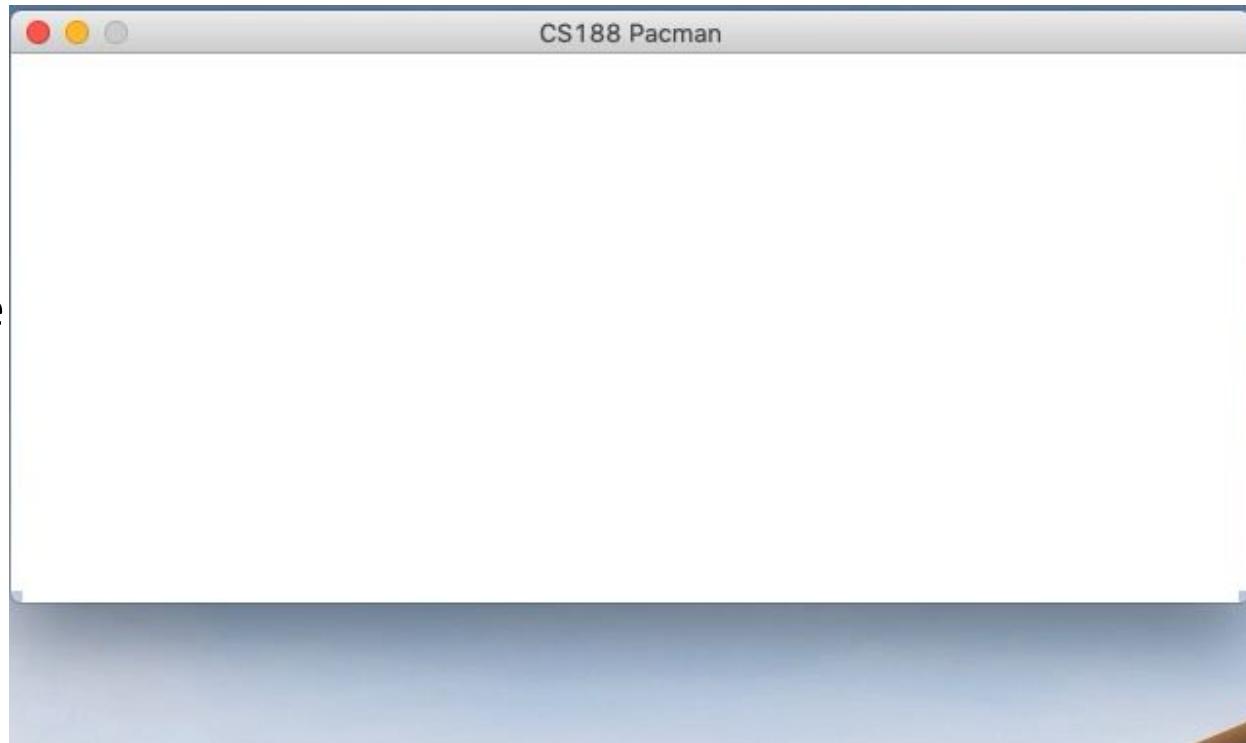
- Naïve Q-learning
- After 50 training episodes



- Generalize Q-learning with function approximator



- Generalizing knowledge results in efficient learning
- E.g., learn to avoid the ghosts



Generalizing with Deep learning

- **Supervised:** Require large amounts of hand-labelled training data
 - **RL** on the other hand, learns from a scalar reward signal that is frequently sparse, noisy, and delayed
- **Supervised:** Assume the data samples are independent
 - In **RL** one typically encounters sequences of highly correlated state
- **Supervised:** Assume a fixed underlying distribution
 - In **RL** the data distribution changes as the algorithm learns new behaviors
- DQN was first to demonstrate that a convolutional neural network can overcome these challenges to learn successful control policies from raw video data in complex RL environments

Deep Q learning [Mnih et al. 2015]

- Trains a generic neural network-based agent that successfully learns to operate in as many domains as possible
- The network is not provided with any domain-specific information or hand-designed features
- Must learn from nothing but the raw input (pixels), the reward, terminal signals, and the set of possible actions

Original Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

$$\theta = \theta + \alpha \left(R + \gamma \max_a \hat{Q}(S', a; \theta) - \hat{Q}(S, A; \theta) \right) \nabla_{\theta} \hat{Q}(S, A; \theta)$$

$$S \leftarrow S'$$

 until S is terminal

Deep Q learning [Mnih et al. 2015]

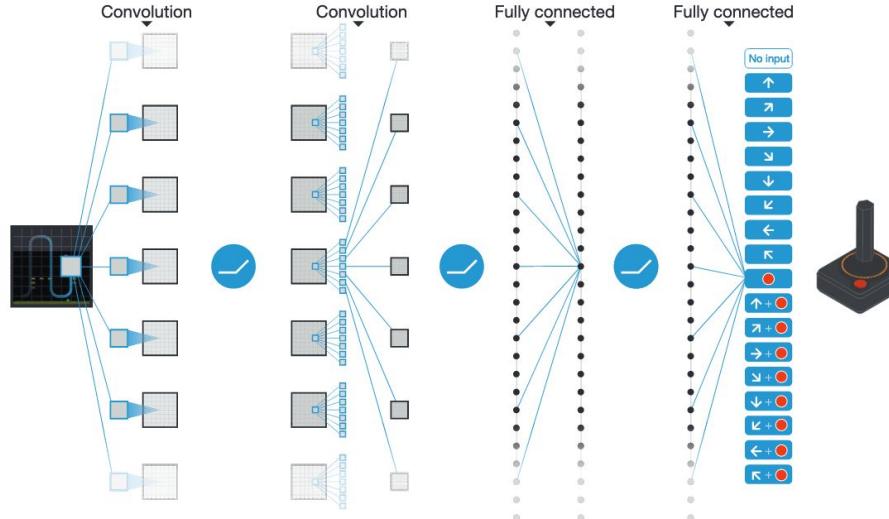
- DQN addresses problems of correlated data and non-stationary distributions
 - Use an experience replay mechanism
 - Randomly samples and trains on previous transitions
 - Results in a smoother training distribution over many past behaviors

Deep Q learning [Mnih et al. 2015]

- Learn a function approximator (Q network), $\hat{Q}(s, a; \theta) \approx Q^*(s, a)$
- Value propagation: $Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \right]$
 - \mathcal{E} represents the environment (underlying MDP)
- Update $\hat{Q}(s, a; \theta)$ at each step i using SGD with squared loss:
- $L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[\left(y_i - \hat{Q}(s, a; \theta_i) \right)^2 \right]$
 - $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_{i-1}) \right]$
 - $\rho(s, a)$ is the behavior distribution, e.g., epsilon greedy
 - θ_{i-1} is considered fix when optimizing the loss function (helps when taking the derivative with respect to θ and with stability)

Deep Q learning [Mnih et al. 2015]

- $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - \hat{Q}(s, a; \theta_i))^2 \right]$ Independent of θ_i (because θ_{i-1} is considered fix)
- $\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot), s' \sim \mathcal{E}} [(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_{i-1}) - \hat{Q}(s, a; \theta_i)) \nabla_{\theta_i} \hat{Q}(s, a; \theta_i)]$



Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Initialize the replay memory and two identical Q approximators (DNN). \hat{Q} is our target approximator.

Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do** ← Play m episodes (full games)

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Start episode from x_1 (pixels at the starting screen).

Preprocess the state (include 4 last frames, RGB to grayscale conversion, downsampling, cropping)

Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do** ← For each time step during the episode

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

With small probability select a random action (explore), otherwise select the, currently known, best action (exploit).

Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Execute the chosen action and store the
(processed) observed transition in the replay
memory

Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Experience replay:
Sample a random minibatch of transitions from replay memory and perform gradient decent step on Q (not on \hat{Q})

Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Once every several steps set the target function, \hat{Q} , to equal Q

Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Such delayed online learning helps in practice:

"This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $Q(s_t, a_t)$ often also increases $Q(s_{t+1}, a)$ for all a and hence also increases the target y_j , possibly leading to oscillations or divergence of the policy" [Human-level control through deep reinforcement learning. Nature 518.7540 (2015): 529.]

Deep Q learning

- *model-free*: solves the reinforcement learning task directly using samples from the emulator \mathcal{E} , without explicitly learning an estimate of \mathcal{E}
- *off-policy*: learns the optimal policy, $a = \underset{a}{\operatorname{argmax}} Q(s, a; \theta)$, while following a different behavior policy
 - One that ensures adequate exploration of the state space

Experience replay

- Utilizing experience replay has several advantages
 - Each step of experience is potentially used in many weight updates, which allows for greater data efficiency
 - Learning directly from consecutive samples is inefficient, due to the strong correlations between the samples; randomizing the samples breaks these correlations and therefore reduces the variance of the updates
 - The behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters
- Note that when learning by experience replay, it is necessary to learn off-policy (because our current parameters are different to those used to generate the sample), which motivates the choice of Q-learning

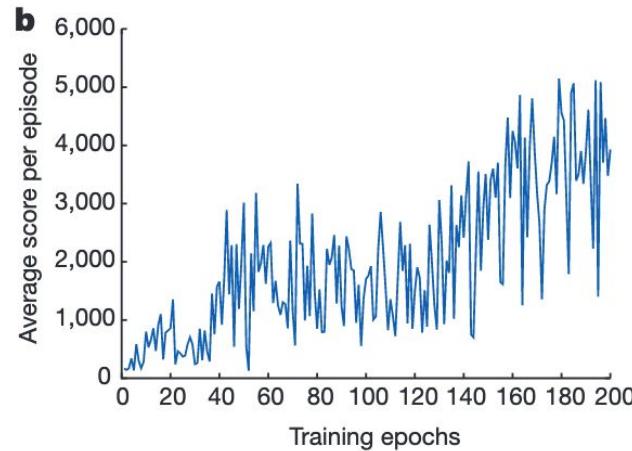
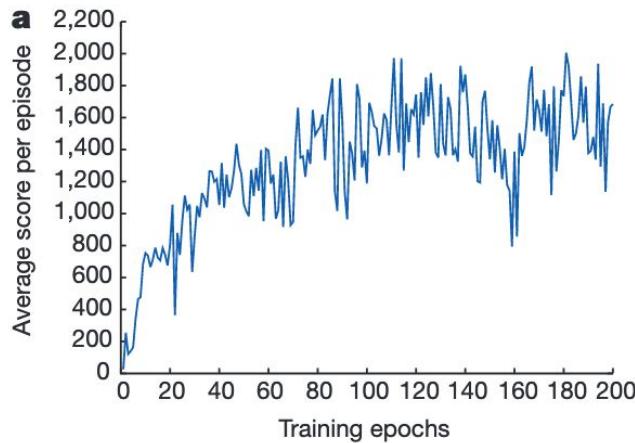
Experience replay

- DQN only stores the last N experience tuples in the replay memory
 - Old transitions are overwritten
- Samples uniformly at random from the buffer when performing updates
- Is there room for improvement?
 - Important transitions?
 - Prioritized sweeping
 - Prioritize deletions from the replay memory
 - see prioritized experience reply, <https://arxiv.org/abs/1511.05952>

**Before training
peaceful swimming**

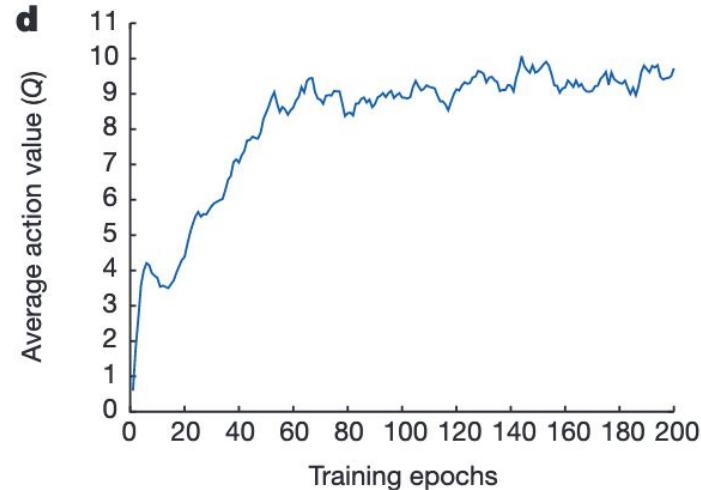
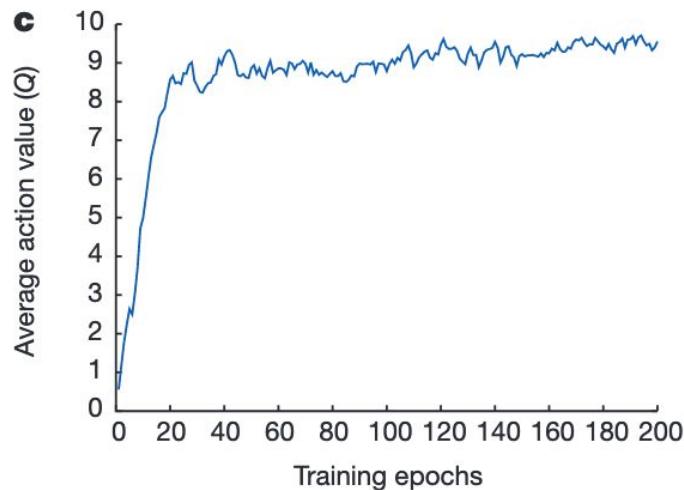
Results: DQN

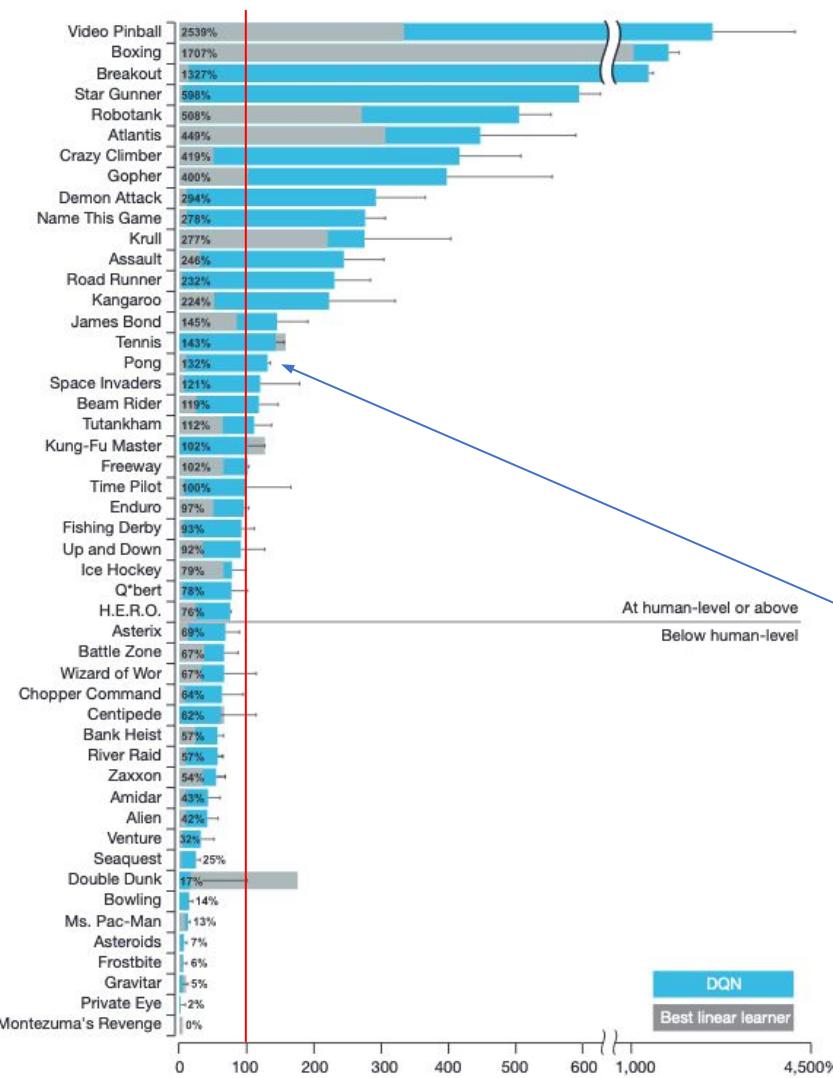
- Each point is the average score achieved per episode after the agent is run with an epsilon-greedy policy ($\varepsilon = 0.05$) for 520k frames on SpaceInvaders (a) and Seaquest (b)



Results: DQN

- Average predicted action-value on a held-out set of states on Space Invaders (c) and Seaquest (d)





- Normalized between a professional human games tester (100%) and random play (0%)
- E.g., in Pong, DQN achieved a factor of 1.32 higher score on average when compared to a professional human player

Maximization bias

- See lecture 5TD_learning.pptx, slide 41
- The max operator in Q-learning uses the same values both to select and to evaluate an action
 - $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_{i-1}) \right]$
 - Makes it more likely to select overestimated values, resulting in overoptimistic value estimates
- We can use a technique similar to the previously discussed Double Q-learning (lecture 5TD_learning.pptx, slide 43)
 - Two value functions are trained. Update only one of the two value functions at each step while considering the max from the other

Double Deep Q networks (DDQN)

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

We have two available Q networks. Let's use them for double learning

Double Deep Q networks (DDQN)

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

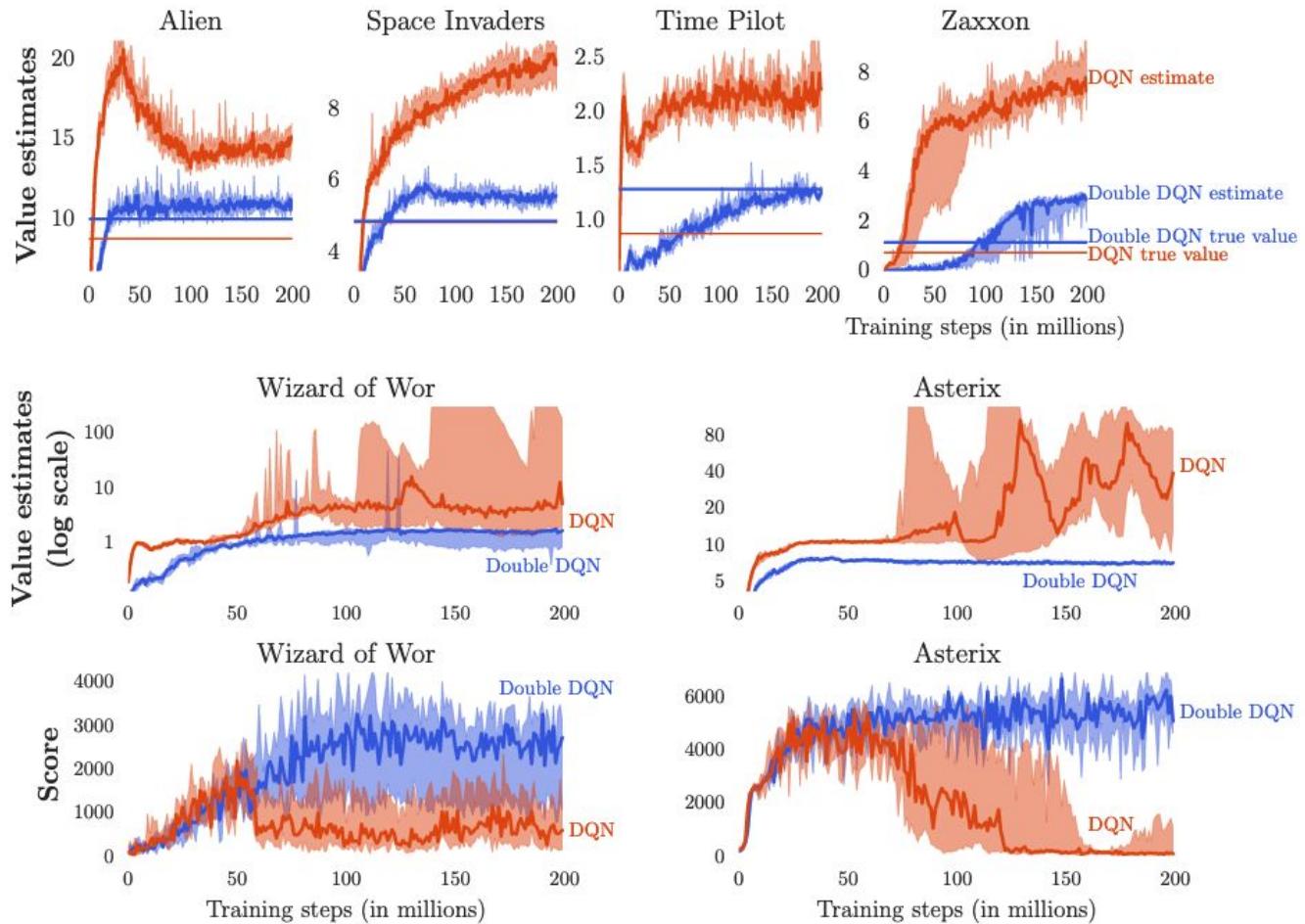
End For

We have two available Q networks. Let's use them for double learning

$$r_j + \gamma \hat{Q}(\phi_{j+1}, \arg \max_{a'} Q(\phi_{j+1}, a'; \theta), \theta^-)$$

- Hasselt et al.
2015

- The straight horizontal orange (for DQN) and blue (for Double DQN) lines in the top row are computed by running the corresponding agents after learning concluded, and averaging the actual discounted return obtained from each visited state. These straight lines would match the learning curves at the right side of the plots if there is no bias.



What did we learn?

- Using deep neural networks as function approximators in RL is tricky
 - Sparse samples
 - Correlated samples
 - Evolving policy (nonstationary sample distribution)
- DQN attempts to address these issues
 - Reuse previous transitions at each training (SGD) step
 - Randomly sample previous transitions to break correlation
 - Use off-policy, TD(0) learning to allow convergence to the true target values (Q^*)
 - No guarantees for non-linear (DNN) approximators



Required Readings

1. Chapter-6 of Introduction to Reinforcement Learning, 2nd Ed., Sutton & Barto



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



Deep Reinforcement Learning
2022-23 Second Semester, M.Tech (AIML)

Session #13: Policy Gradients - REINFORCE, Actor-Critic algorithms

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)

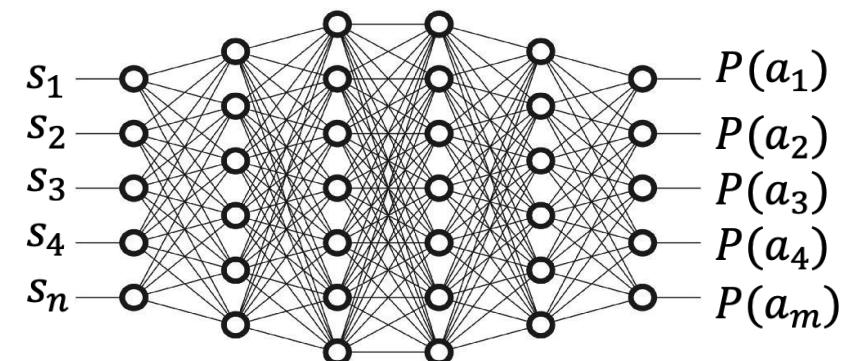


Agenda for the classes

- Introduction
- Policy gradients
- REINFORCE algorithm
- Actor-critic methods
- REINFORCE - example

Notation

- The policy is a parametrized function: $\pi_\theta(a|s)$
 - For policy gradient we need a continuous, differentiable policy... (Softmax activation can be useful for discrete action spaces)
 - $\pi(a|s)$ is assumed to be differentiable with respect to θ
 - E.g., a DNN where θ is the set of weights and biases
- $J(\theta)$ is a scalar policy performance measure (expected sum of discounted rewards) with respect to the policy params



Improving the policy

- SGD: $\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)}$
- Where $\widehat{\nabla J(\theta_t)}$ is a stochastic estimate, whose expectation approximates the gradient of the performance measure
- All methods that follow this general schema we call **policy gradient methods**
 - Might also learn an approximate value function for normalization (baseline)
- Methods that use approximations to both policy and value functions for computing the policy's gradient are called **actor–critic methods** (more on this later)

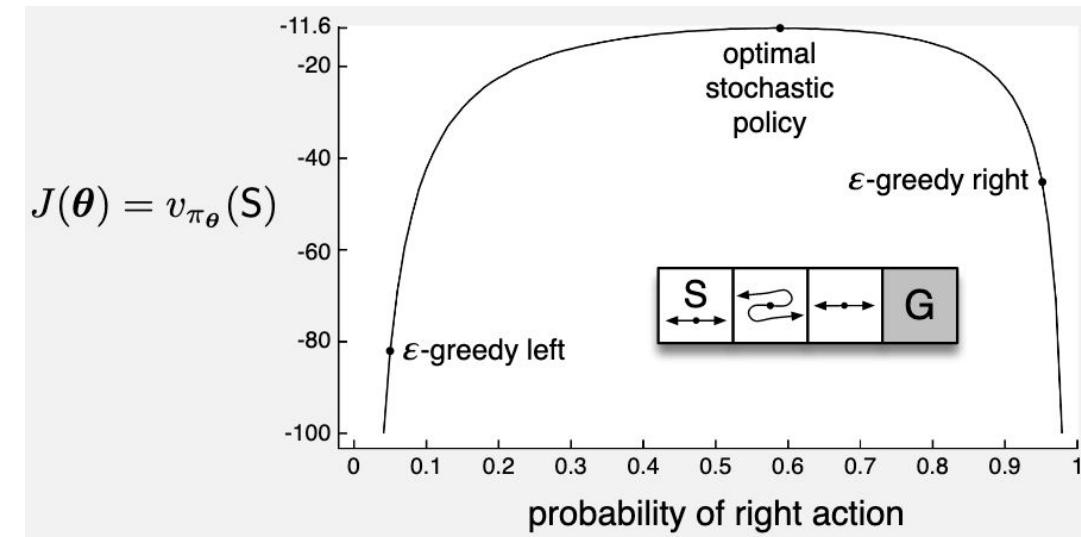


Advantages of PG

- The policy convergence over time as opposed to an epsilon-greedy value-based approach
- Naturally applies to continuous action space as opposed to a Q learning approach
- In many domains the policy is a simpler function to approximate
Though this is not always the case
- Choice of policy parameterization is sometimes a good way of injecting prior knowledge
E.g., in phase assignment by a traffic controller
- Can converge on stochastic optimal policies, as opposed to value-based approaches
Useful in games with imperfect information where the optimal play is often to do two different things with specific probabilities, e.g., bluffing in Poker

Stochastic policy

- Reward = -1 per step
- $\mathcal{A} = \{\text{left}, \text{right}\}$
- Left goes left and right goes right except in the middle state where they are reversed
- States are identical from the policy's perspective
- $\pi^* = [0.41 \quad 0.59]$





Evaluate the gradient in performance

- $\widehat{\nabla_{\theta} J(\theta)} = ?$
- J depends on both the action selections and the distribution of states in which those selections are made
 - Both of these are affected by the policy parameter θ
- Seems impossible to solve without knowing the transition function (or the distribution of visited states)
 - $p(s'|s, a)$ is unknown in model free RL
- The PG theorem allows us to evaluate $\widehat{\nabla_{\theta} J(\theta)}$ without the need for $p(s'|s, a)$



Monte-Carlo Policy Gradient

- Sample-based gradient estimation

- $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s)$
- $\sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s) = \mathbb{E}_\pi [\sum_a q_\pi(S_t, a) \nabla_\theta \pi(a|S_t)]$
- We can now use this gradient estimation for PG steps
 - $\theta_{t+1} = \theta_t + \alpha \sum_a \widehat{q_\pi}(S_t, a) \nabla_\theta \pi(a|S_t; \theta)$
 - Requires an approximation to q_π , denoted \widehat{q} , for every possible action
 - Can we avoid this approximation?

REINFORCE [Williams, 1992]

1. $\theta_{t+1} = \theta_t + \alpha \sum_a \widehat{q}_\pi(S_t, a) \nabla_\theta \pi(a|S_t)$

- Can we avoid this approximation? YES!

2. $\nabla J(\theta) \propto \mathbb{E}_\pi [\sum_a q_\pi(S_t, a) \nabla_\theta \pi(a|S_t)]$

3. $= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t; \theta) q_\pi(S_t, a) \frac{\nabla_\theta \pi(a|S_t)}{\pi(a|S_t)} \right]$ (multiplied and divided by the same number)

4. $= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla_\theta \pi(A_t|S_t)}{\pi(A_t|S_t)} \right]$ ($\sum_x \Pr(x) f(x) = \mathbb{E}_{x \sim \Pr(x)}[f(x)]$)

5. $= \mathbb{E}_\pi \left[G_t \frac{\nabla_\theta \pi(A_t|S_t)}{\pi(A_t|S_t)} \right]$ ($\mathbb{E}_\pi[G_t|S_t, A_t] = q_\pi(S_t, A_t)$)

- We can now use this gradient estimation for PG steps

- $\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t)}{\pi(A_t|S_t)}$

REINFORCE [Williams, 1992]

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t;\theta)}{\pi(A_t|S_t;\theta)}$$

- Each increment is proportional to the product of a return G_t and a vector
 - The gradient of the probability of taking the action actually taken over the (current) probability of taking that action
 - The vector is the direction in parameter space that most increases the probability of repeating the action A_t on future visits to state S_t
- The update increases the parameter vector (*) proportional to the return, and (**) inversely proportional to the action probability
 - (*) makes sense because it causes the parameter to move most in the directions that favor actions that yield the highest return
 - (**) makes sense because otherwise actions that are selected frequently are at an advantage (the updates will be more often in their direction) and might win out even if they do not yield the highest return



REINFORCE [Williams, 1992]

- $\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t | S_t; \theta)}{\pi(A_t | S_t; \theta)}$
- REINFORCE uses G_t : the complete return from time t until the end of the episode
- In this sense REINFORCE is a Monte Carlo algorithm and is well defined only for the episodic case with all updates made in retrospect after the episode is completed

REINFORCE [Williams, 1992]

* In the literature you might encounter "grad log pi" instead of "grad ln pi". That's not a problem since: $\nabla \ln(f(x)) \propto \nabla \log(f(x))$

Specifically: $\nabla \ln(f(x)) = \nabla \log_a(f(x)) \ln(a)$

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$

Repeat forever:

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot| \cdot, \theta)$

For each step of the episode $t = 0, \dots, T - 1$:

$G \leftarrow$ return from step t

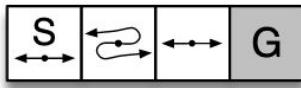
$$\theta \leftarrow \theta + \alpha \gamma^t G \boxed{\nabla_{\theta} \ln \pi(A_t | S_t, \theta)}$$

How¹² did we get here
from $\frac{\nabla_{\theta} \pi(A_t | S_t; \theta)}{\pi(A_t | S_t; \theta)}$?

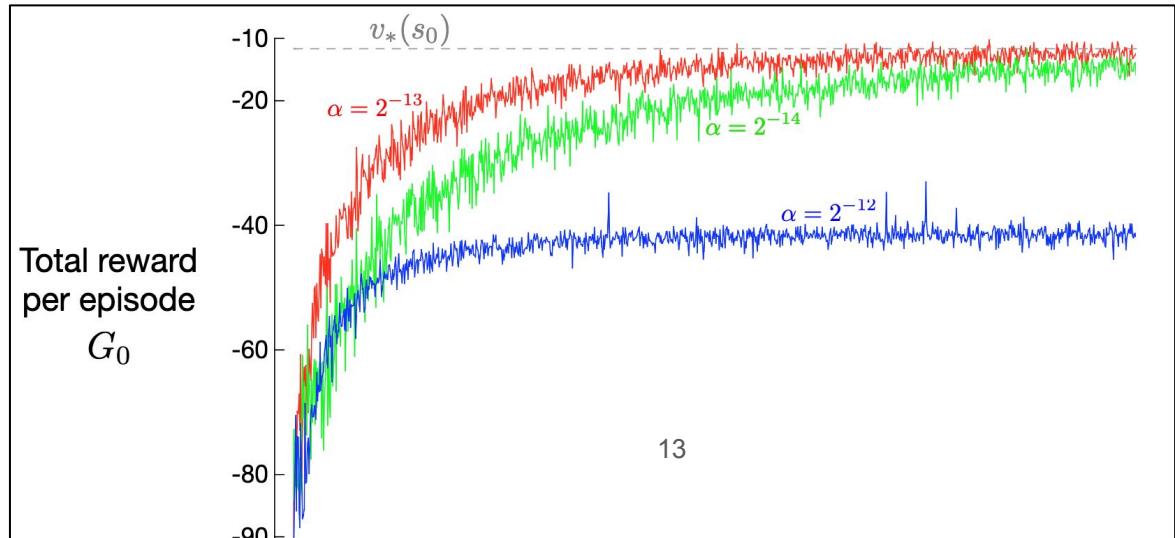
$$\nabla \ln(f(x)) = \frac{\nabla f(x)}{f(x)}$$

REINFORCE [Williams, 1992]

- The crazy corridor domain



- With the right step size, the total reward per episode approaches the optimal value of the start state



Guaranteed to converge to a local optimum under standard stochastic approximation conditions for decreasing α

Calibrating REINFORCE

- Imagine a domain with two possible episode trajectories (rollouts)
 - $\tau_1 = \{S_0, A_0, R_1, S_1, \dots, A_{T-1}, R_T, S_T\}$ with $G = 1001$
 - $\tau_2 = \{S'_0, A'_0, R'_1, S'_1, \dots, A'_{T-1}, R'_T, S'_T\}$ with $G' = 1000$
 - Further assume that $R_{i < T} = R'_i = 0$ and $R_T = G, R'_T = G'$
- Following REINFORCE: $\theta_{t+1} = \theta_t + \alpha \textcolor{red}{G}_t \nabla_\theta \ln \pi(A_t | S_t; \theta)$
- How **will** the policy be updated after sampling each of the trajectories?
 - $\tau_1 ++, \tau_2 ++$
- How **should** the policy be updated after sampling each of the trajectories?
 - $\tau_1 +, \tau_2 -$
- How can we address this gap?

PG with baseline

- Normalize the returns with a baseline!
- $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_\pi(s, a) - b(s)) \nabla \pi(a|s)$
- Are we allowed to do that???
- Can we subtract $\sum_a b(s) \nabla \pi(a|s)$ from $\nabla J(\theta)$?
- Yes! As long as $b(s)$ isn't a function of a
- $\sum_a b(s) \nabla \pi(a|s) = b(s) \nabla \sum_a \pi(a|s) = b(s) \nabla 1 = 0$ 15 (actions' probabilities sum to 1)
- REINFORCE with baseline: $\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t)) \nabla_\theta \ln \pi_\theta(A_t | S_t)$

PG with baseline

- In the bandit algorithms the baseline was the average of the rewards seen so far
- For MDPs the baseline should be different for each states
 - In some states all actions have high (q) values, and we need a high baseline to differentiate the higher valued actions from the less highly valued ones
 - In other states all actions will have low values and a low baseline is appropriate
- What would be the equivalent of average reward (bandits) for a given state (MDP) ?
 - $v_\pi(s)$

PG with baseline

- $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_\pi(s, a) - v_\pi(s)) \nabla \pi(a|s)$
- Actions that improve on the current policy are encouraged
 - $q_\pi(s, a) - v_\pi(s) > 0$
- Actions that damage the current policy are discouraged
 - $q_\pi(s, a) - v_\pi(s) < 0$
- Also known as the **advantage** of action a in state s
- How can we learn $v_\pi(s)$?
- Another function approximator $\hat{v}(s; w)$
 - On top of the policy

REINFORCE with baseline

REINFORCE with Baseline (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value parameterization $\hat{v}(s, w)$

Parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$

Repeat forever:

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot | \cdot, \theta)$

For each step of the episode $t = 0, \dots, T - 1$:

$$G_t \leftarrow \text{return from step } t$$

$$\delta \leftarrow G_t - \hat{v}(S_t, w)$$

$$w \leftarrow w + \alpha^w \gamma^t \delta \nabla_w \hat{v}(S_t, w)$$

$$\theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla_\theta \ln \pi(A_t | S_t, \theta)$$

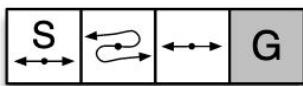
18

Each approximator has its
unique learning rate

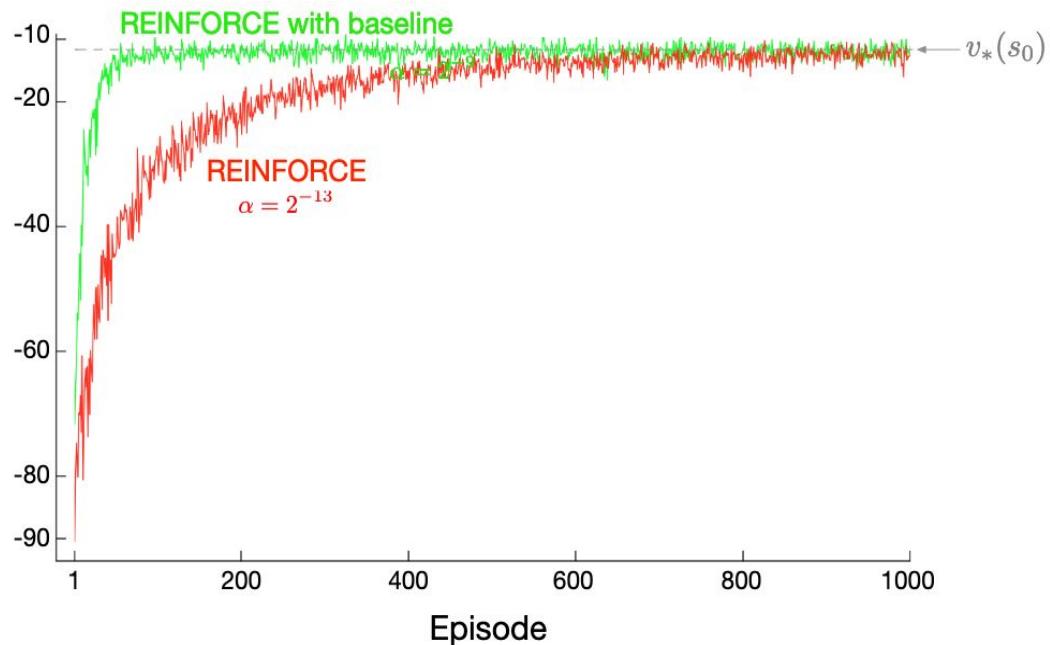
REINFORCE with baseline

- The crazy corridor domain

- $\alpha^\theta = 2^{-9}$
- $\alpha^w = 2^{-6}$



Total reward
per episode
 G_0



Policy Gradient

- $\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)}$
- PG theorem: $\widehat{\nabla J(\theta_t)} \propto (q_\pi(S_t, A_t) - b(S_t)) \nabla_\theta \log \pi(A_t | S_t; \theta)$
 - REINFORCE+baseline: $\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t)) \nabla_\theta \log \pi(A_t | S_t; \theta)$
- **Pros:** approximating the optimal policy directly is more accurate and faster to converge in many domains when compared to value-based approaches
- **Cons:** using G_t as an estimator for $q_\pi(S_t, A_t)$ is noisy²⁰(high variance) though unbiased
 - Unstable learning

Add a critic

- $\widehat{\nabla J(\theta_t)} \propto (q_{\pi}(S_t, A_t) - b(S_t)) \frac{\nabla_{\theta} \pi(A_t | S_t; \theta)}{\pi(A_t | S_t; \theta)}$
- Define a new estimator: $\hat{q}_{\pi}(s, a; \theta) \approx q_{\pi}(s, a)$
 - To be used instead of G_t in REINFORCE
- How should we train $\hat{q}_{\pi}(s, a; \theta)$?
 - Monte-Carlo updates
 - High variance samples
 - Requires a full episode
 - Bootstrapping e.g., Q-learning
 - Lower variance (though introduces bias)

Critic's duties

1. Approximate state or action or advantage ($q(s, a) - v(s)$) values
 2. Trained via bootstrapping, criticizes the action chosen by the actor
adjusting the actor's (policy) gradient
- Is REINFORCE (+ state value baseline) an actor-critic framework?
 - $\theta_{t+1} = \theta_t + \alpha(G_t - \hat{v}_\pi(S_t)) \nabla_\theta \ln \pi(A_t | S_t; \theta)$
 - NO! the state-value function is used only as a baseline, not as a critic.
Moreover, it doesn't utilize bootstrapping (uses high-variance MC updates)²²
 - The bias introduced through bootstrapping is often worthwhile because it reduces variance and accelerates learning

Benefits from a critic

- REINFORCE with baseline is unbiased* and will converge asymptotically to a local optimum
 - * With a linear state value approximator, and when b is not a function of a
 - Like all Monte-Carlo methods it tends to learn slowly (produce estimates of high variance)
 - Not suitable for online or for continuing problems
- Temporal-difference methods can eliminate these inconveniences
- In order to gain the TD advantages in the case of policy gradient methods we use actor–critic methods

Actor+critic

- Actor-critic algorithms are a derivative of policy iteration, which alternates between policy evaluation—computing the value function for a policy—and policy improvement—using the value function to obtain a better policy
- In large-scale reinforcement learning problems, it is typically impractical to run either of these steps to convergence, and instead the value function and policy are optimized jointly
- The policy is referred to as the actor, and the value function as the critic

Advantage function

- Eventually we would like to shift the policy towards actions that result in higher return
- what is the benefit from taking action a at state s while following policy π ?
 - $A_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$
- This resembles PG with baseline but not the same as q_π is approximated:
 - $\widehat{\nabla J(\theta_t)} = A_\pi(s_t, a_t) \nabla_\theta \ln \pi(a_t | s_t; \theta)$
- Actions that improve on the current policy are encouraged
 - $A_\pi(s, a) > 0$
- Actions that damage the current policy are discouraged
 - $A_\pi(s, a) < 0$

One-step actor-critic

- $A_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$
 - Does that mean that approximating the advantage function requires two function approximators (\hat{q} and \hat{v}) ?
 - No since q values can be derived from state values + one step transition
 - $\hat{q}_\pi(s_t, a_t) - \hat{v}_\pi(s_t; w) = \hat{\mathbb{E}}[r_{t+1} + \gamma \hat{v}(s_{t+1}; w)] - \hat{v}(s_t; w)$
- $\theta_{t+1} = \theta_t + \alpha(r_{t+1} + \gamma \hat{v}(s_{t+1}; w) - \hat{v}(s_t; w)) \nabla_\theta \ln \pi(a_t | s_t; \theta)$

$\delta - \text{TD error}$

 - One-step Actor-Critic is a fully online, incremental algorithm, with states, actions, and rewards processed as they occur and then never revisited

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value parameterization $\hat{v}(s, w)$

Parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$$A \sim \pi(\cdot|S, \theta)$$

 Take action A , observe S', R

$$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w) \quad (\text{if } S' \text{ is terminal, then } \hat{v}(S', w) \doteq 0)$$

$$w \leftarrow w + \alpha^w \delta \nabla_w \hat{v}(S, w)$$

$$\theta \leftarrow \theta + \alpha^\theta I \delta \nabla_\theta \ln \pi(A|S, \theta)$$

$$I \leftarrow \gamma I$$

$$S \leftarrow S'$$

Keep track of
accumulated discount

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value parameterization $\hat{v}(s, w)$

Parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$A \sim \pi(\cdot|S, \theta)$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$

 (if S' is terminal, then $\hat{v}(S', w) \doteq 0$)

$w \leftarrow w + \alpha^w \delta \nabla_w \hat{v}(S, w)$

$\theta \leftarrow \theta + \alpha^\theta I \delta \nabla_\theta \ln \pi(A|S, \theta)$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Follow the current
policy

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$

Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$

Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$A \sim \pi(\cdot|S, \boldsymbol{\theta})$

 Take action A , observe S' , R

$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

 (if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S, \boldsymbol{\theta})$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Compute the TD error

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value parameterization $\hat{v}(s, w)$

Parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$A \sim \pi(\cdot|S, \theta)$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$

$w \leftarrow w + \alpha^w \delta \nabla_w \hat{v}(S, w)$

$\theta \leftarrow \theta + \alpha^\theta I \delta \nabla_\theta \ln \pi(A|S, \theta)$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Update the critic without
the accumulated discount.

(The discount factor is
included in the TD error)

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$

Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$

Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$A \sim \pi(\cdot|S, \boldsymbol{\theta})$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

 (if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S, \boldsymbol{\theta})$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Update the actor with
discounting. Early actions
matter more.

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$

Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$

Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$A \sim \pi(\cdot|S, \boldsymbol{\theta})$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ (if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S, \boldsymbol{\theta})$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Update accumulated
discount and progress to
the next state

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value parameterization $\hat{v}(s, w)$

Parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$A \sim \pi(\cdot|S, \theta)$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$

 (if S' is terminal, then $\hat{v}(S', w) \doteq 0$)

$w \leftarrow w + \alpha^w \delta \nabla_w \hat{v}(S, w)$

$\theta \leftarrow \theta + \alpha^\theta I \delta \nabla_\theta \ln \pi(A|S, \theta)$

$I \leftarrow \gamma I$

$S \leftarrow S'$

In practice: training the network at every step on a single observation is inefficient (slow and correlated)

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$

Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$

Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$A \sim \pi(\cdot|S, \boldsymbol{\theta})$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S, \boldsymbol{\theta})$

$I \leftarrow \gamma I$

$S \leftarrow S'$

(if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)

Instead: store all state approx values, log probabilities, and rewards along the episode.
Train once at the end of the episode.

One-step Actor-Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value parameterization $\hat{v}(s, w)$

Parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$

Repeat forever:

 Initialize S (first state of episode)

$I \leftarrow 1$

 While S is not terminal:

$A \sim \pi(\cdot|S, \theta)$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$

$w \leftarrow w + \alpha^w \delta \nabla_w \hat{v}(S, w)$

$\theta \leftarrow \theta + \alpha^\theta I \delta \nabla_\theta \ln \pi(A|S, \theta)$

$I \leftarrow \gamma I$

$S \leftarrow S'$

(if S' is terminal, then $\hat{v}(S', w) \doteq 0$)

Instead: store all state approx. values, log probabilities, and rewards along the episode.
Train once at the end of the episode.

values = torch.FloatTensor(values)
Qvals = torch.FloatTensor(Qvals)
log_probs = torch.stack(log_probs)
advantage = Qvals - values

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic \hat{V}_w
- For each episode:
 - Init empty episode memory
 - $s = \text{init env}$
 - For each time step:
 - $a \sim \pi(s)$
 - $s', r_t \sim \text{env}(s, a)$
 - Store (s, a, r) in episode memory
 - $s = s'$
 - Reset $d\theta = 0, dw = 0$
 - Backwards iteration (i from length to 0) over the episode
 - Compute advantage $\delta_t = r_i + \gamma \hat{V}_w(s_{i+1}) - \hat{V}_w(s_i)$
 - Accumulate the policy gradient using the critic: $d\theta \leftarrow d\theta + \delta \nabla_\theta \log \pi_\theta(s_i, a_i)$
 - Accumulate the critic gradient: $dw \leftarrow dw + \delta \nabla_w \hat{V}_w(s_i)$
 - Update the actor and the critic with the accumulated gradients using gradient descent

Add eligibility traces

Actor-Critic with Eligibility Traces (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value parameterization $\hat{v}(s, w)$

Parameters: trace-decay rates $\lambda^\theta \in [0, 1]$, $\lambda^w \in [0, 1]$; step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$

Repeat forever (for each episode):

 Initialize S (first state of episode)

$z^\theta \leftarrow \mathbf{0}$ (d' -component eligibility trace vector)

$z^w \leftarrow \mathbf{0}$ (d -component eligibility trace vector)

$I \leftarrow 1$

 While S is not terminal (for each time step):

$A \sim \pi(\cdot|S, \theta)$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$ (if S' is terminal, then $\hat{v}(S', w) \doteq 0$)

$z^w \leftarrow \gamma \lambda^w z^w + \nabla_w \hat{v}(S, w)$

$z^\theta \leftarrow \gamma \lambda^\theta z^\theta + I \nabla_\theta \ln \pi(A|S, \theta)$

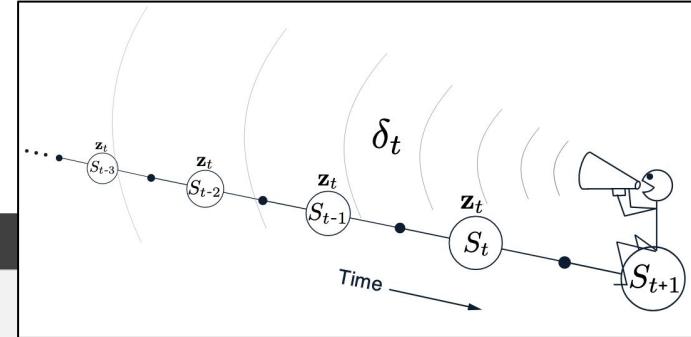
$w \leftarrow w + \alpha^w \delta z^w$

$\theta \leftarrow \theta + \alpha^\theta \delta z^\theta$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Gradient eligibility per tunable parameter for both the actor and the critic approximators



What did we learn?

- In many domains it's more efficient to learn the policy directly
 - Instead of deriving one from state or action values
- Applicable for continuous action spaces and stochastic policies
- Approximate the change to the policy that results in the highest increase in the expected return: $\nabla_{\theta} J(\theta)$
- Such approximation is made possible when by the policy gradient theorem
- Should we reinforce policies that result in high return?
 - Not always, a different policy might yield higher return
 - We need to use a baseline to determine if a policy is **relatively** good

What did we learn?

- REINFORCE:

- $\bullet \widehat{\nabla J(\theta_t)} = \boxed{G_t} \nabla_{\theta} \ln \pi(A_t | S_t; \theta)$

- Q Actor-Critic:

- $\bullet \widehat{\nabla J(\theta_t)} = \boxed{\hat{q}(S_t, A_t; w)} \nabla_{\theta} \ln \pi(A_t | S_t; \theta)$

- REINFORCE + baseline:

- $\bullet \widehat{\nabla J(\theta_t)} = (\boxed{G_t} - \boxed{\hat{v}(S_t; w)}) \nabla_{\theta} \ln \pi(A_t | S_t; \theta)$

- Advantage Actor-Critic:

- $\bullet \widehat{\nabla J(\theta_t)} = (\boxed{R_{t+1} + \gamma \hat{v}(S_{t+1}; w)} - \boxed{\hat{v}(S_t; w)}) \nabla_{\theta} \ln \pi(A_t | S_t; \theta)$



Required Readings

1. Chapter-6 of Introduction to Reinforcement Learning, 2nd Ed., Sutton & Barto



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



Deep Reinforcement Learning
2022-23 Second Semester, M.Tech (AIML)

Session #14: Model Based Algorithms

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)



Agenda for the classes

- Introduction
- Upper-Confidence-bound Action Selection
- Monte-Carlo Tree Search
- AlphaGo Zero
- MuZero, PlaNet



Model Based Algorithms

- Learn the model of an environment's transition dynamics or make use of a known dynamic model
- Once an agent has a model of the environment, $P(s'|s,a)$, it can "imagine" what will happen in the future by predicting the trajectory for a few time steps.
- If the environment is in state s , an agent can estimate how the state will change if it makes a sequence of actions a_1, a_2, \dots, a_n by repeatedly applying $P(s'|s,a)$ all without actually producing an action to change the environment.
- Hence, the predicted trajectory occurs in the agent's "head" using a model.



Model Based Algorithms

- Most commonly applied to games with a target state, such as winning or losing in a game of chess, or navigation tasks with a goal state s^* .
- Do not model any rewards



Model based algorithms - Advantages

- Very appealing : it can play out scenarios and understand the consequences of its actions without having to actually act in an environment.
- Require many fewer samples of data to learn good policies since having a model enables an agent to supplement its actual experiences with imagined ones.



Model based algorithms - challenges

- Models are hard to come by.
- An environment with a large state space and action space can be very difficult to model; doing so may even be intractable, especially if the transitions are extremely complex
- Models are only useful when they can accurately predict the transitions of an environment many steps into the future - prediction errors



Upper Confidence bound action selection

UCB is a deterministic algorithm for Reinforcement Learning that focuses on exploration and exploitation based on a confidence boundary that the algorithm assigns to each machine on each round of exploration. These boundary decreases when a machine is used more in comparison to other machines.

The **Upper Confidence Bound** follows the principle of optimism in the face of uncertainty which implies that if we are uncertain about an action, we should optimistically assume that it is the correct action.

Initially, UCB explores more to systematically reduce uncertainty but its exploration reduces over time. Thus we can say that UCB obtains greater reward on average than other algorithms such as Epsilon-greedy, Optimistic Initial Values, etc.

UCB Algorithm

$$A_t = \operatorname{argmax}_a (Q_t(a) + c \sqrt{\frac{\ln(t)}{N_t(a)}})$$

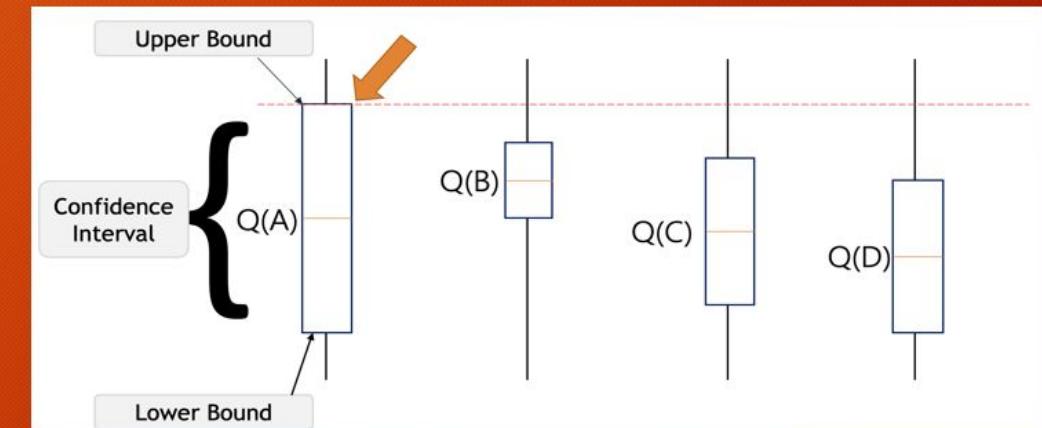
Exploit

Explore

t = timesteps

$N_t(a)$ = no. of times action (a) is taken

$$Q_t(a) = \frac{\text{sum of rewards when (a) taken prior to (t)}}{\text{number of times (a) taken prior to (t)}}$$





Upper Confidence bound action selection

Steps followed in Upper Confidence Bound

1. *At each round n , we consider two numbers for machine m .*
-> $N^{\square}(n)$ = number of times the machine m was selected up to round n .
-> $R^{\square}(n)$ = number of rewards of the machine m up to round n .
2. *From these two numbers we have to calculate,*
 - a. The average reward of machine m up to round n , $r^{\square}(n) = R^{\square}(n) / N^{\square}(n)$.
 - b. The confidence interval $[r^{\square}(n) — \Delta^{\square}(n), r^{\square}(n)+\Delta^{\square}(n)]$ at round n with, $\Delta^{\square}(n)= \text{sqrt}(1.5 * \log(n) / N^{\square}(n))$
3. *We select the machine m that has the maximum UCB, ($r^{\square}(n)+\Delta^{\square}(n)$)*

Monte Carlo Tree Search

- Monte Carlo Tree Search (MCTS) is a heuristic search set of rules that has won big attention and reputation within the discipline of synthetic intelligence, specially in the area of choice-making and game playing.
- MCTS combines the standards of Monte Carlo strategies, which rely upon random sampling and statistical evaluation, with tree-primarily based search techniques.
- **Idea :**
 - build a seek tree incrementally by using simulating more than one random performs (regularly known as rollouts or playouts) from the current recreation nation.
 - These simulations are carried out until a terminal state or a predefined intensity is reached.
 - The results of these simulations are then backpropagated up the tree, updating the records of the nodes visited at some stage in the play, which includes the wide variety of visits and the win ratios.



Monte Carlo Tree Search

- It is a probabilistic and heuristic driven search algorithm that combines the classic tree search implementations alongside machine learning principles of reinforcement learning.
- ***exploration-exploitation trade-off***: exploits the actions and strategies that is found to be the best till now but also must continue to explore the local space of alternative decisions and find out if they could replace the current best.
- exploration expands the tree's breadth more than its depth. But it quickly becomes inefficient in situations with large number of steps or repetitions.
- Exploitation sticks to a single path that has the greatest estimated value. This is a greedy approach and this will extend the tree's depth more than its breadth.



Monte Carlo Tree Search

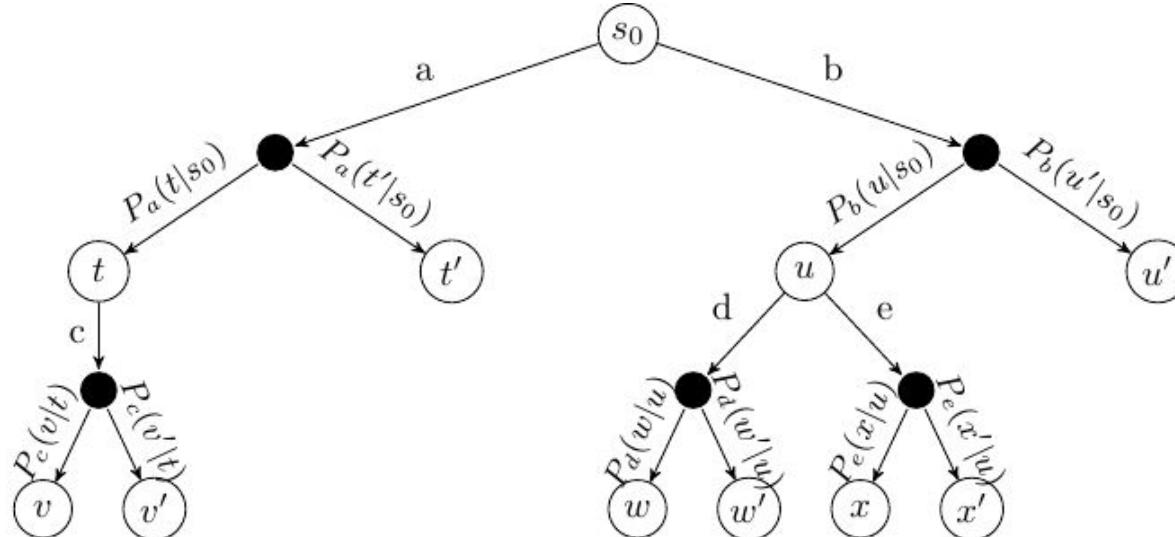
Why use Monte Carlo Tree Search (MCTS) ?

1. Handling Complex and Strategic Games
2. Unknown or Imperfect Information
3. Learning from Simulations
4. Optimizing Exploration and Exploitation
5. Scalability and Parallelization
6. Applicability Beyond Games
7. Domain Independence

Monte Carlo Tree Search

Nodes are the building blocks of the search tree and are formed based on the outcome of a number of simulations.

MDPs can be represented as trees (or graphs), called *ExpectiMax* trees:



The letters $a-e$ represent actions, and letters $s-x$ represent states. White nodes are state nodes, and the small black nodes represent the probabilistic uncertainty: the ‘environment’ choosing which outcome from an action happens, based on the transition function.



Monte Carlo Tree Search – Overview

The algorithm is online, which means the action selection is interleaved with action execution. Thus, MCTS is invoked every time an agent visits a new state.

Fundamental features:

1. The Q-value $Q(s,a)$ for each is approximated using *random simulation*.
2. For a single-agent problem, an ExpectiMax *search tree* is built incrementally
3. The search terminates when some pre-defined computational budget is used up, such as a time limit or a number of expanded nodes. Therefore, it is an *anytime* algorithm, as it can be terminated at any time and still give an answer.
4. The best performing action is returned.
 - This is complete if there are *no* dead-ends.
 - This is optimal if an entire search can be performed (which is unusual – if the problem is that small we should just use a dynamic programming technique such as value iteration).



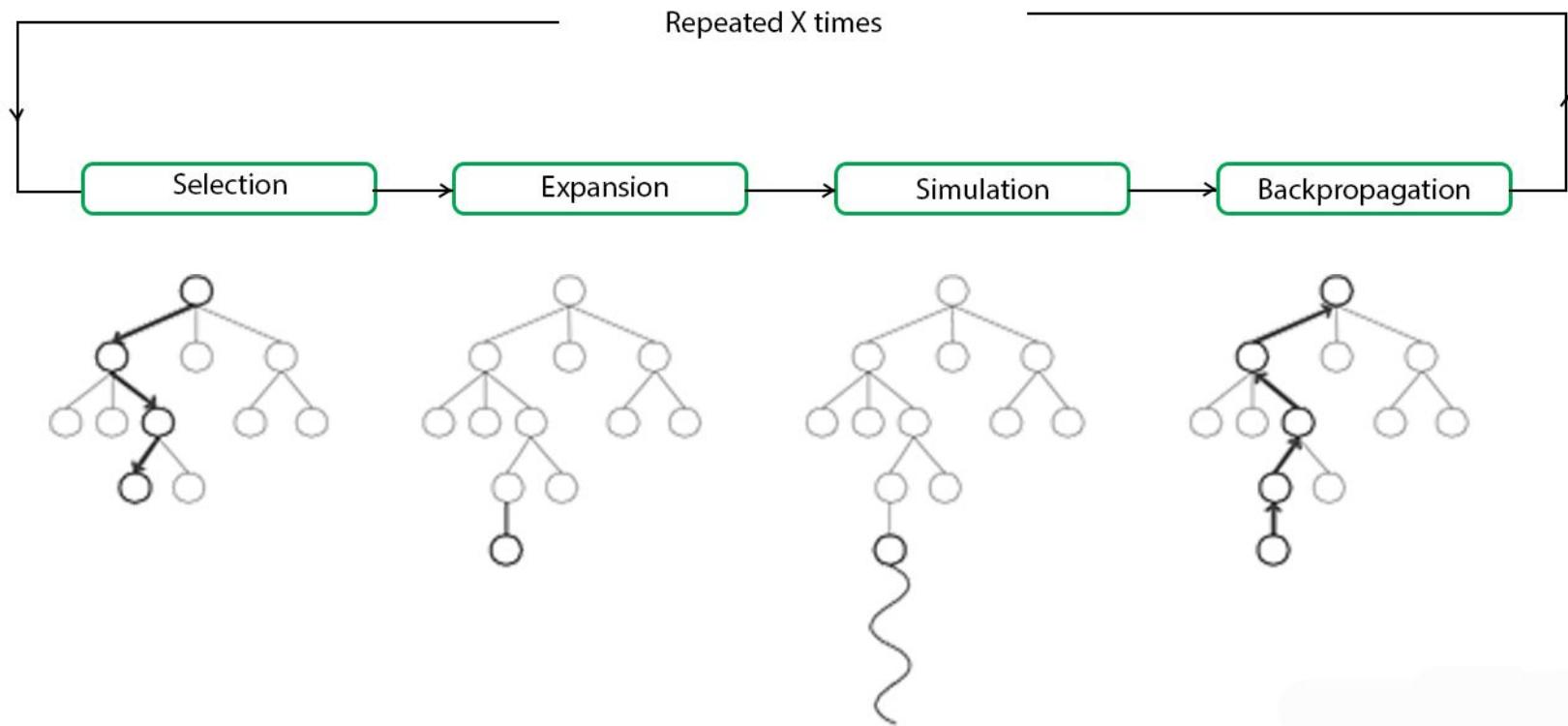
The Framework of MCTS

The basic framework is to build up a tree using simulation. The states that have been evaluated are stored in a search tree. The set of evaluated states is *incrementally* built by iterating over the following four steps:

- **Select:** Select a single node in the tree that is *not fully expanded*. By this, we mean at least one of its children is not yet explored.
- **Expand:** Expand this node by applying one available action (as defined by the MDP) from the node.
- **Simulation:** From one of the outcomes of the expanded, perform a complete random simulation of the MDP to a terminating state. This therefore assumes that the simulation is finite, but versions of MCTS exist in which we just execute for some time and then estimate the outcome.
- **Backpropagate:** Finally, the value of the node is *backpropagated* to the root node, updating the value of each ancestor node on the way using expected value



MCTS Framework



In a basic MCTS algorithm we incrementally build of the search tree. Each node in the tree stores:

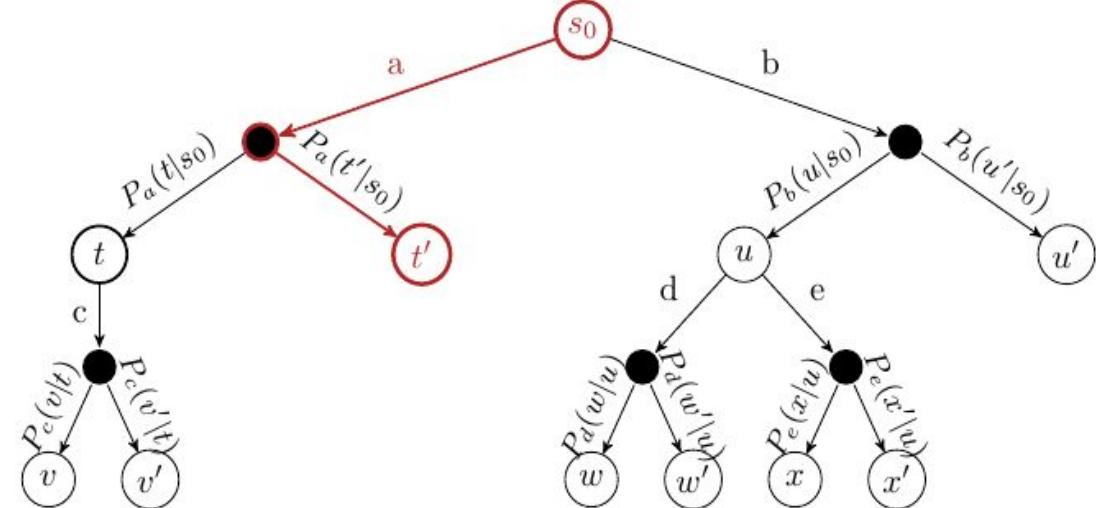
1. a set of children nodes;
2. pointers to its parent node and parent action; and
3. the number of times it has been visited.

i Algorithm – Monte-Carlo Tree Search

Input: MDP $M = \langle S, s_0, A, P_a(s' | s), r(s, a, s') \rangle$, base value function Q , time limit T .

Output: updated Q-function Q

```
while currentTime < T
    selected_node ← Select( $s_0$ )
    child ← Expand(selected_node) – expand and choose a child to simulate
     $G \leftarrow$  Simulate(child) – simulate from child
    Backpropagate(selected_node, child,  $G$ )
return  $Q$ 
```



Selection: The first loop progressively selects a branch in the tree using a multi-armed bandit algorithm using $Q(s|a)$. The outcome that occurs from an action is chosen according to $P(s'|s,a)$ defined in the MDP.

i Function – Select($s : S$)

Input: state s

Output: unexpanded state

while s is fully expanded

 Select action a to apply in s using a multi-armed bandit algorithm

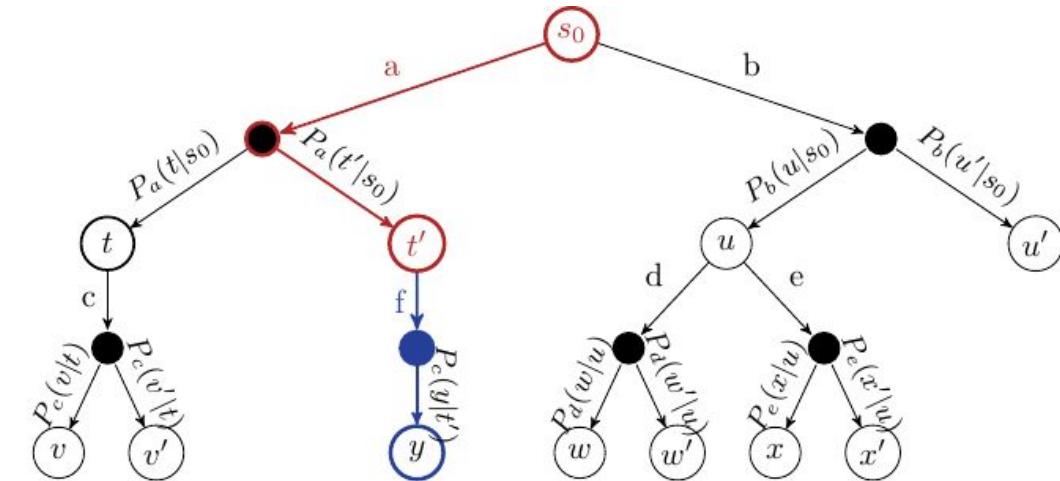
 Choose one outcome s' according to $P_a(s' | s)$

$s \leftarrow s'$

return s

Expansion

Select an action q to apply in state s , either randomly or using an heuristic. Get an outcome state s' from applying action a in state s according to the probability distribution $p(s'|s)$. Expand a new environment node and a new state node for that outcome.



i Function - $\text{Expand}(s : S)$

Input: state s

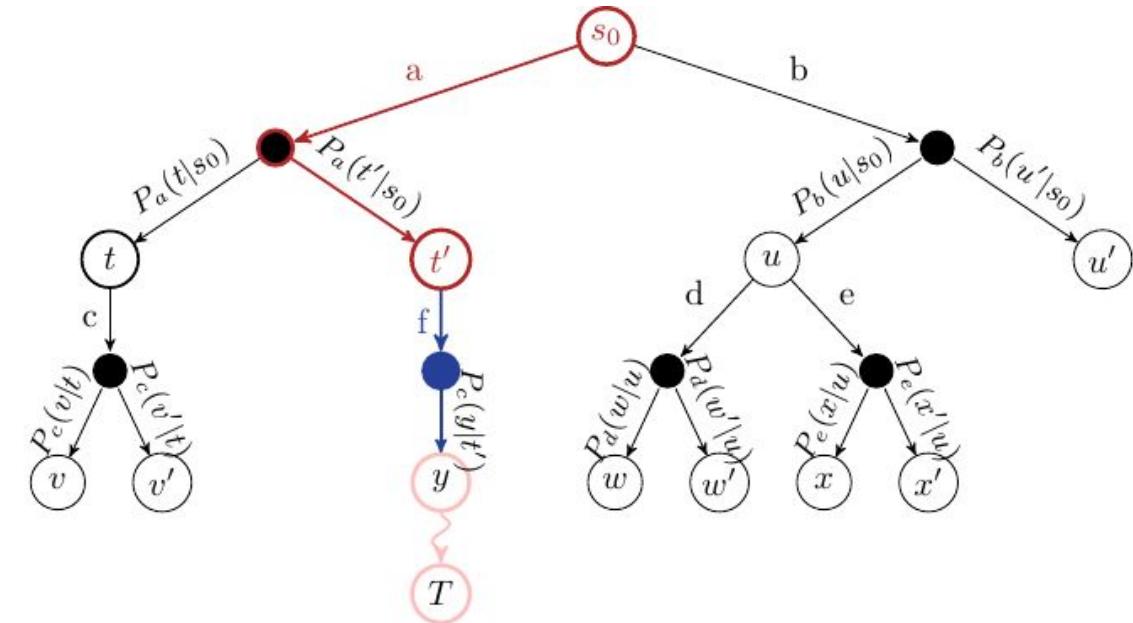
Output: expanded state s'

Select an action a from s to apply

Expand one outcome s' according to the distribution $P_a(s' | s)$ and observe reward r

return s'

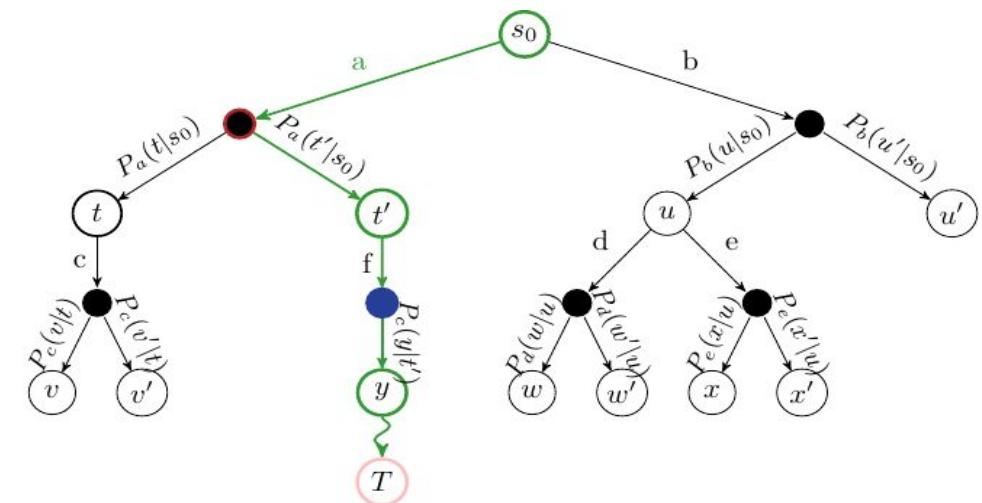
Simulation: Perform a randomised simulation of the MDP until we reach a terminating state. That is, at each choice point, randomly select an possible action from the MDP, and use transition probabilities $P(s'|s)$ to choose an outcome for each action.



Heuristics can be used to improve the random simulation by guiding it towards more promising states. G is the cumulative discounted reward received from the simulation starting at s' until the simulation terminates.

To avoid memory explosion, we discard all nodes generated from the simulation. In any non-trivial search, we are unlikely to ever need them again.

Backpropagation: The reward from the simulation is backpropagated from the selected node to its ancestors recursively. We must not forget the *discount factor*! For each state s and action a selected in the Select step, update the cumulative reward of that state.



i Procedure – Backpropagation($s : S; a : A$)

Input: state-action pair (s, a)

Output: none

do

$$N(s, a) \leftarrow N(s, a) + 1$$

$$G \leftarrow r + \gamma G$$

$$Q(s, a) \leftarrow Q(s, a) + \frac{1}{N(s, a)} [G - Q(s, a)]$$

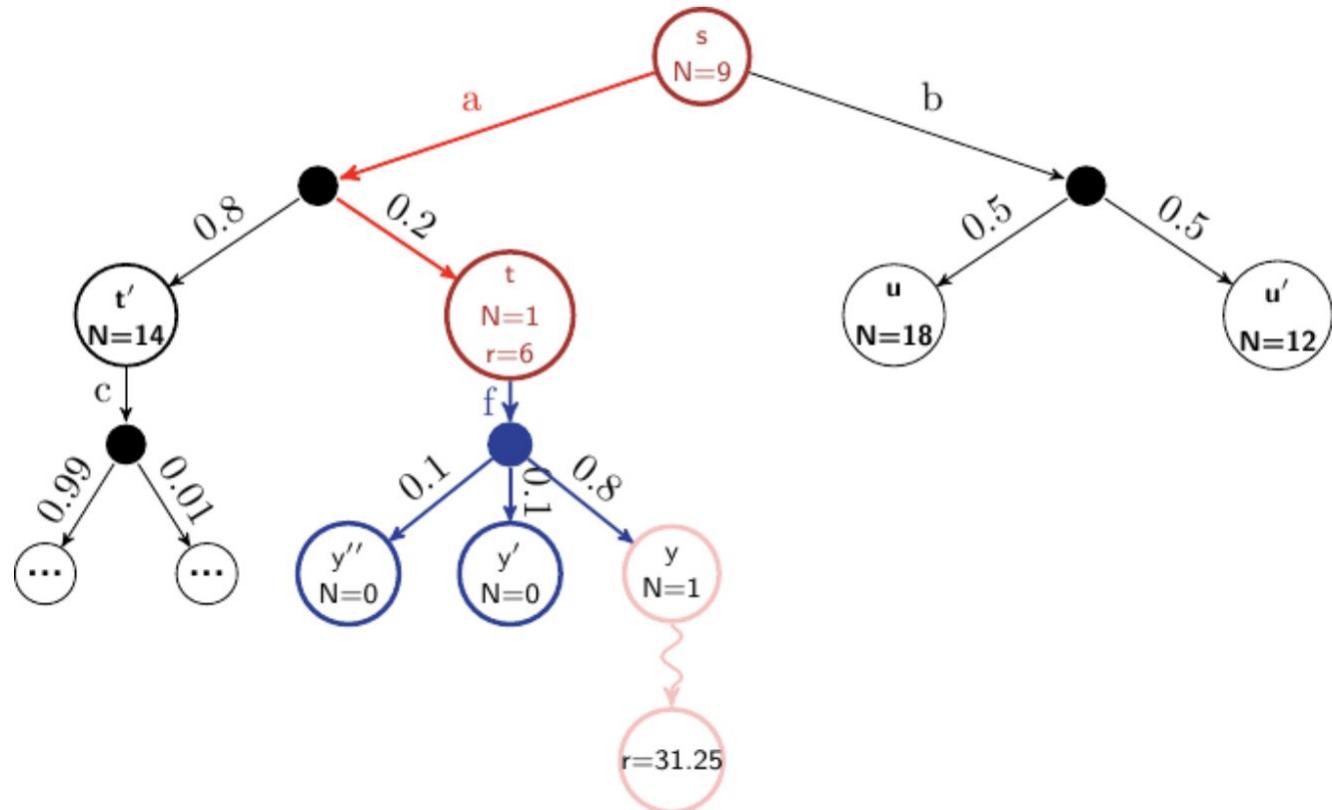
$s \leftarrow$ parent of s

$a \leftarrow$ parent action of s

while $s \neq s_0$

Consider the following ExpectiMax tree that has been expanded several times. Assume $\gamma = 0.8$, $r = X$ represents reward X received at a state, V represents the value of the state (the value $\max_{a' \in \text{children}} Q(s, a')$) and the length of the simulation is 14. After the simulation step, but before backpropagation, our tree would look like this:

Example : Backpropagation





In the next iteration, the red actions are selected, and the blue node is expanded to its three children nodes. A simulation is run from y , which terminates after 3 steps. A reward of 31.25 is received in the terminal state. This would mean that the discounted reward returned at node y would be $\gamma \times 31.25$, which is 20.

Before backpropagation, we have the following:

$$\begin{aligned} Q(s, a) &= 18 \\ Q(t, f) &= 0 \end{aligned}$$

We leave out the remainder of the Q-function as it is not relevant to the example.

The backpropagation step is then calculated for the nodes y , t , and s as follows:

$$\begin{aligned}Q(y, g) &= \gamma^2 \times 31.25 \text{ (simulation is 3 steps long and receives reward of 31.25)} \\&= 20\end{aligned}$$

$$\begin{aligned}Q(t, f) &= Q(t, f) + \frac{1}{N(t, f)} [r + \gamma G - Q(t, f)] \\&= 0 + \frac{1}{2} [0 + 0.8 \cdot 20 - 0] \\&= 8\end{aligned}$$

$$\begin{aligned}Q(s, a) &= Q(s, a) + \frac{1}{N(s, a)} [r + \gamma G - Q(s, a)] \\&= 18 + \frac{1}{5} [6 + 0.8 \cdot (0.8 \cdot 20) - 18] \\&= 18 + \frac{1}{5} [6 + 12.8 - 18] \\&= 18.16\end{aligned}$$



Action selection

Once we have run out of computational time, we select the action that maximises our expected return, which is simply the one with the highest Q-value from our simulations:

$$\operatorname{argmax}_{a \in A(s)} Q(s_0, a)$$

We execute that action and wait to see which outcome occurs for the action.

Once we see the outcome state, which we will call s' , we start the process all over again, except with $s_0 \leftarrow s'$.

However, importantly, we can *keep* the sub-tree from state s' , as we already have done simulations from that state. We discard the rest of the tree (all children of s_0 other than the chosen action) and incrementally build from s' .



Issues in Monte Carlo Tree Search:

1. Exploration-Exploitation Trade-off
2. Sample Efficiency
3. High Variance
4. Heuristic Design
5. Computation and Memory Requirements
6. Overfitting
7. Domain-specific Challenges

	Value iteration	MCTS
Cost	Higher cost (exhaustive)	Lower cost (does not solve for entire state space)
Coverage/ Robustness	Higher (works from any state)	Lower (works only from initial state or state reachable from initial state)



AlphaZero

- Combining MCTS and TD learning: Alpha Zero
- Alpha Zero (or more accurately its predecessor AlphaGo) made headlines when it beat Go world champion Lee Sodol in 2016.
- It uses a combination of MCTS and (deep) reinforcement learning to learn a policy.



AlphaZero - Overview

1. AlphaZero uses a deep neural network to estimate the Q-function. More accurately, it gives an estimate of the probability of selecting action a in state s , $(P(a|s))$ and the *value* of the state $(V(s))$, which represents the probability of the player winning from s .
2. It is trained via *self-play*. Self-play is when the same policy is used to generate the moves of both the learning agent and any of its opponents. In AlphaZero, this means that initially, both players make random moves, but both also learn the same policy and use it to select subsequent moves.
3. In short, AlphaZero is a **game-playing program** that, through a combination of self-play and neural network reinforcement learning (more on that later), is able to learn to play games such as chess and Go from scratch — that is, after being fed nothing more than the rules of said games.

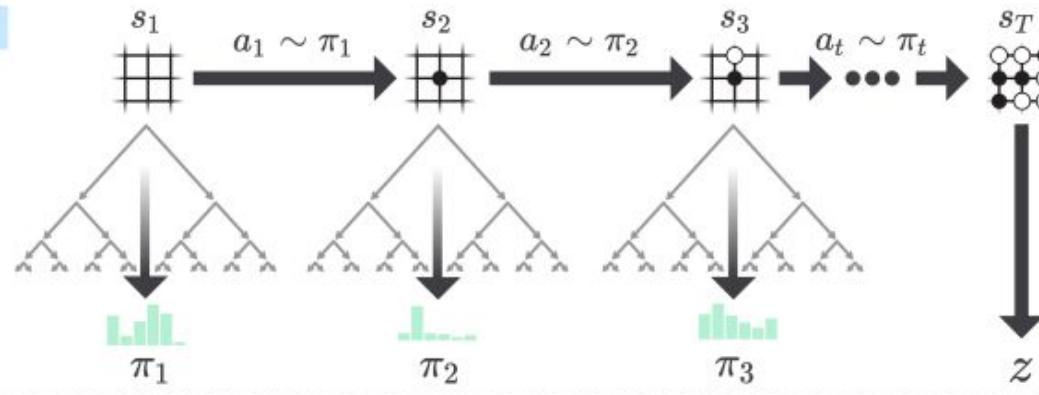
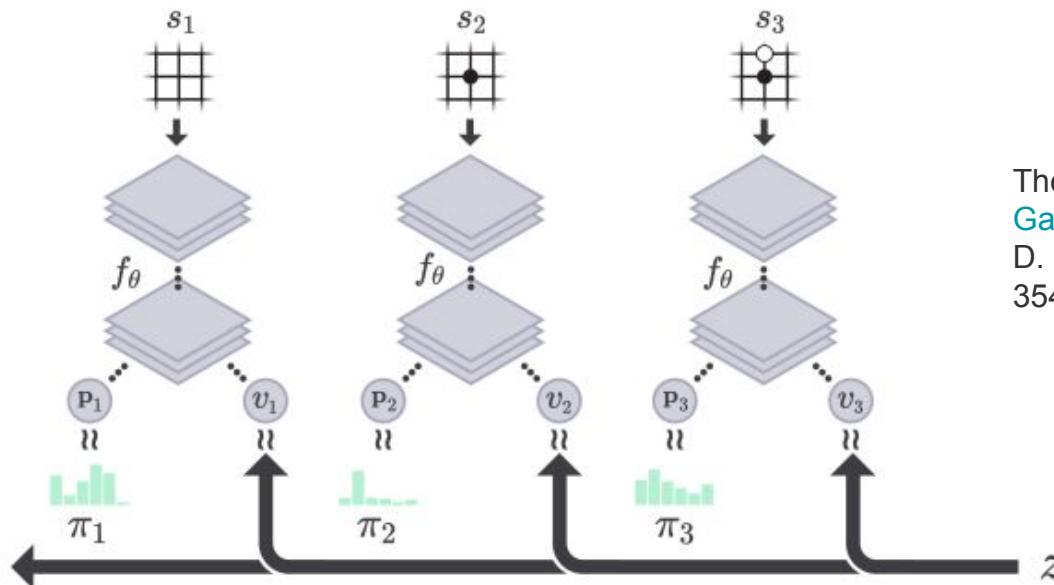


AlphaZero

A DNN initialised to return 2 randomly generated outputs: $v(s)$, and $p(s)$. The network will later be trained to take in position s and predict $v(s)$ and $p(s)$ from it.

At each move, AlphaZero:

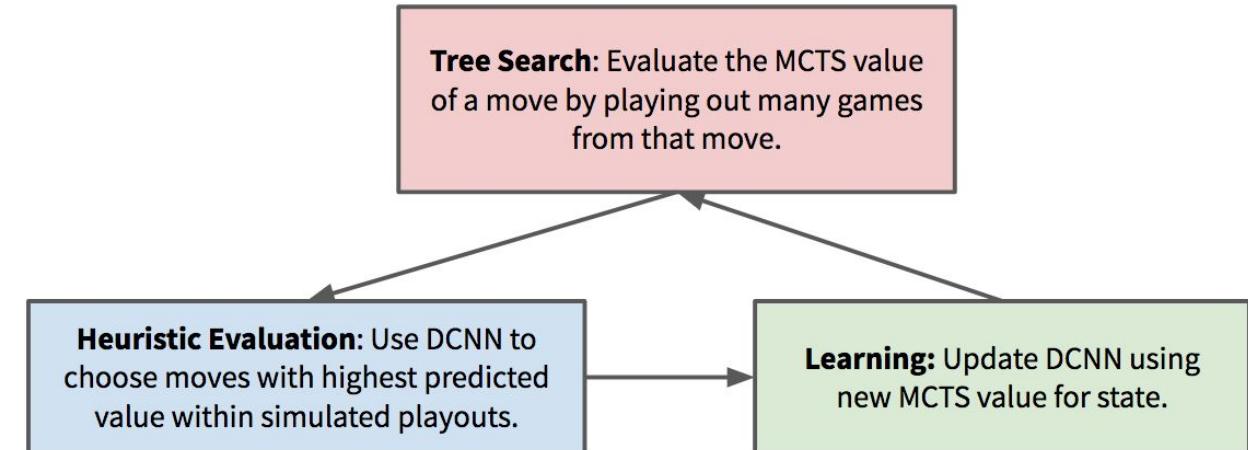
1. Executes an MCTS search using UCB-like selection: $Q(s, a) + P(s, a)/1 + N(s, a)$, which returns the probabilities of playing each move.
2. The neural network is used to guide the MCTS by influencing $Q(s, a)$.
3. The final result of a simulated game is used as the reward for each simulation.
4. After a set number of MCTS simulations, the best move is chosen for self-play.
5. Repeat steps 1-4 for each move until the self-play game ends.
6. Then, feedback the result of the self-play game to update the Q function for each move.

a. Self-Play

b. Neural Network Training


The AlphaZero framework. [Mastering the Game of Go without Human Knowledge](#).
 D. Silver, et al. Nature volume 550, pages 354–359 (2017)

Implications of AlphaZero

- Increase the probability of a win come the end of the game
- AlphaZero's evaluation function is the product of a highly sophisticated neural network's experience accumulated over millions of games
- AlphaZero does not make use of any human knowledge, we can expect it to come up with brand-new ideas previously unknown to mankind.





MuZero

- AlphaZero (2017) - learning to play on its own, without any human data or domain knowledge, or even by mastering three different games (Go, Chess, Shogi) with only one algorithm being only provided with the known set of rules.
- MuZero (2020) - learnt to master these games(Go, Chess, Shogi, Atari) without being provided known rules
- Big step forward towards general-purpose algorithms



MuZero

MuZero models three elements that are key to the planner:

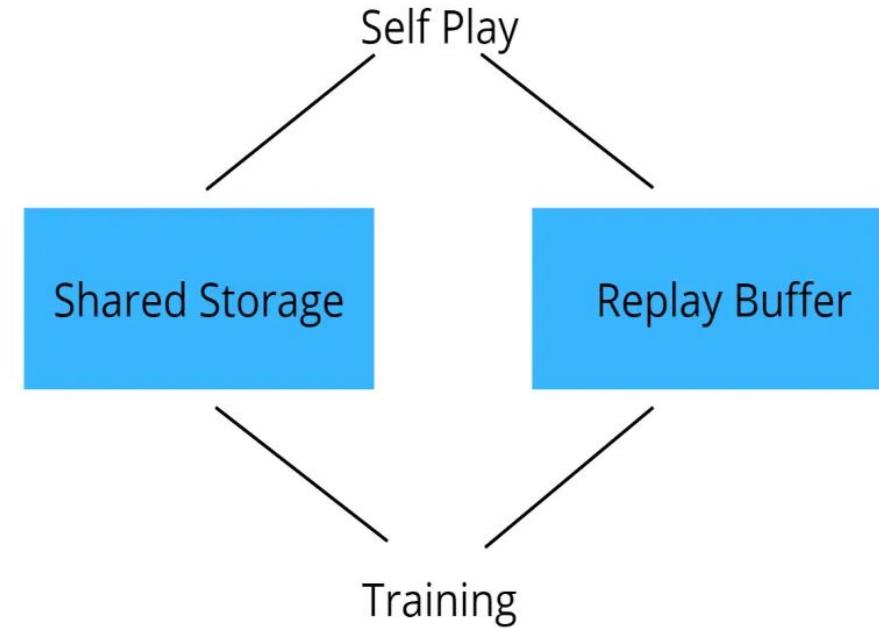
- **Value:** how good is the current state?
- **Policy:** which action should be the next one?
- **Reward:** how good was the last action taken?



MuZero - overview

- The MuZero algorithm aims to accurately predict characteristics and details of the future which it deems important for planning.
- The algorithm initially receives an input, for instance an image of a chess board, which is translated into a hidden state.
- The hidden state then undergoes iterations based on the previous hidden state and a proposed subsequent plan of action.
- Each time the hidden state is updated the model predicts three variables: policy, value function and immediate reward.
- The policy is the next move to be played, the value function is the predicted winner, and the immediate reward is the strength of the move (if it improves the player's position).
- The model is then trained to accurately predict the values of the three aforementioned variables.

- The MuZero algorithm does not receive any rules of the game, for instance in a chess game, the algorithm is unaware of the legal moves or what constitutes as win, draw or a loss.
- MuZero only receives the rewards of its actions when a game is terminated, either by winning, drawing, or losing. However, during its learning, MuZero receives rewards periodically based on how well it is doing in its own mini-games.





Training and Loss function

- function `train_network` repeatedly trains the neural network by making use of the `replayBuffer`.
- The `train_network` function works by looping the total number of training steps, which is set to one million by default.
- The function then samples a batch at every step and uses that data to update the neural network.
- The tuples within a batch are: the current state of the game, a list of actions taken from the current position and lastly the targets used to train the neural networks.
- The targets used to train the neural networks are calculated by using Temporal Difference (TD) Learning .
- The loss function of MuZero is responsible for how the weights of the neural network are updated.



PlaNet

- Deep Planning Network - Google AI & DeepMind Initiative
- PlaNet agent was tasked with ‘planning’ a sequence of actions to achieve a goal like pole balancing, teaching a virtual entity (human or cheetah) to walk, or keeping a box rotating by hitting it in a specific location.
- Common goals between these tasks that the PlaNet needed to achieve:
 1. The Agent needs to predict a variety of possible futures (for robust planning)
 2. The Agent needs to update the plan based on the outcomes/rewards of a recent action
 3. The Agent needs to retain information over many time steps

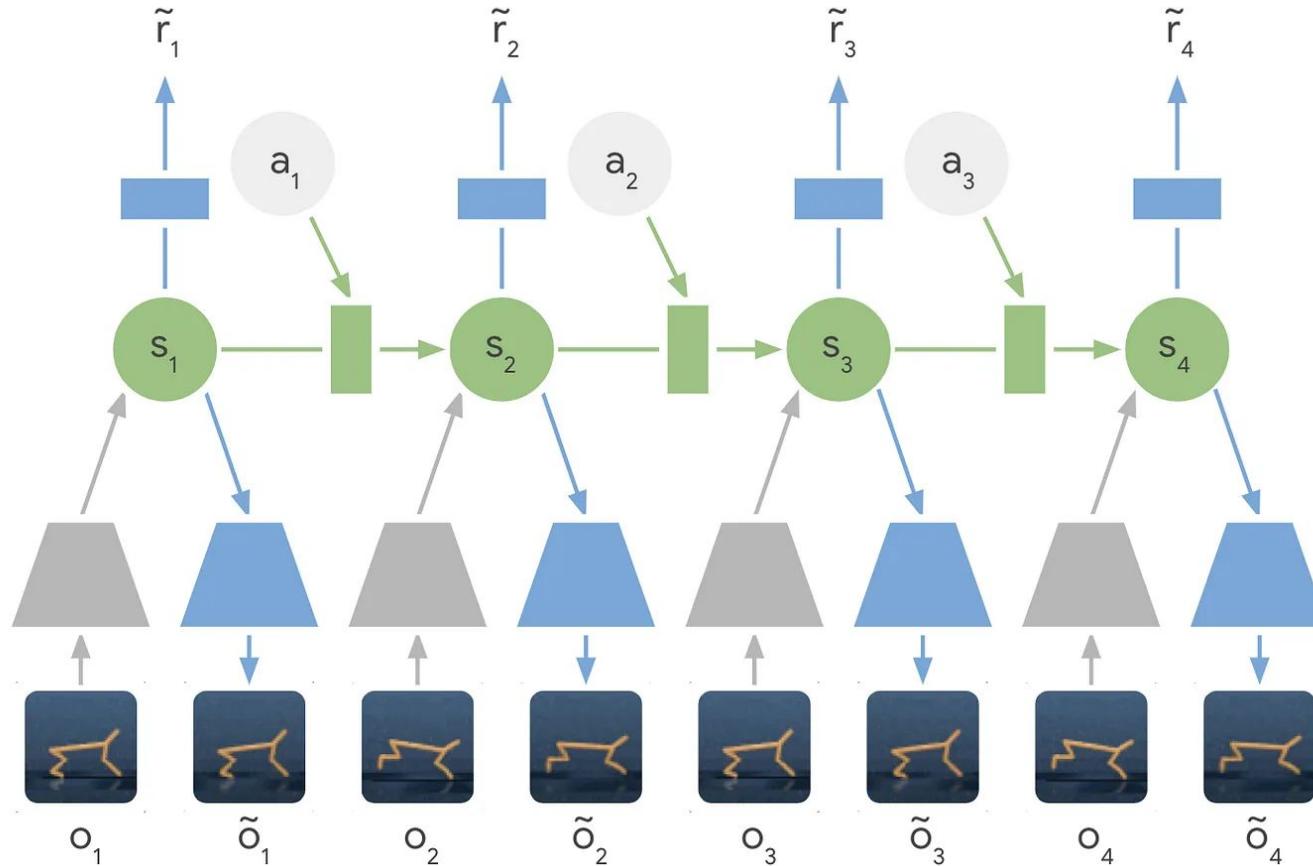
So how did the Google AI team achieve these goals?



PlaNet Overview

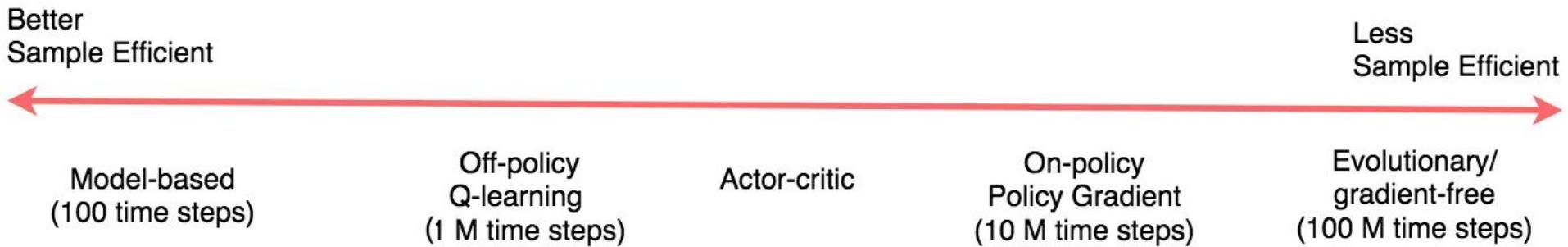
1. **Learning with a latent dynamics model** — PlaNet learns from a series of hidden or latent states *instead of images* to predict the latent state moving forward.
2. **Model-based planning** — PlaNet works without a policy network and instead makes decisions based on continuous planning.
3. **Transfer learning** — The Google AI team trained a single PlaNet agent to solve all six different tasks.

- 1) **Latent Dynamics Model** - Key benefits to using compact latent state spaces are that it allows the agent to learn more abstract representations like the objects' positions and velocities and also avoid having to generate images.



Learned Latent Dynamics Model—Instead of using the input images directly, the encoder networks (gray trapezoids) compress the images' information into hidden states (green circles). These hidden states are then used to predict future images (blue trapezoids) and rewards (blue rectangle).

2. Model based planning



3. Transfer Learning

- one agent for all tasks

The agent is randomly placed into different environments without knowing the task, so it needs to infer the task from its image observations. Without changes to the hyper parameters, the multi-task agent achieves the same mean performance as individual agents.



Required Readings and references

1. <https://rl-lab.com/#play>
2. <https://www.aionlinecourse.com/tutorial/machine-learning/upper-confidence-bound-%28ucb%29>
3. <https://towardsdatascience.com/monte-carlo-tree-search-in-reinforcement-learning-b97d3e743d0f>
4. <https://gibberblot.github.io/rl-notes/single-agent/mcts.html>
5. <https://towardsdatascience.com/alphazero-chess-how-it-works-what-sets-it-apart-and-what-it-can-tell-us-4ab3d2d08867>
6. <https://medium.com/geekculture/muzero-explained-a04cb1bad4d4>
7. <https://towardsdatascience.com/everything-you-need-to-know-about-googles-new-planet-reinforcement-learning-network-144c2ca3f284>
8. <https://blog.research.google/2019/02/introducing-planet-deep-planning.html?m=1>



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



Deep Reinforcement Learning
2022-23 Second Semester, M.Tech (AIML)

Session #15: Imitation Learning

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)



Agenda for the classes

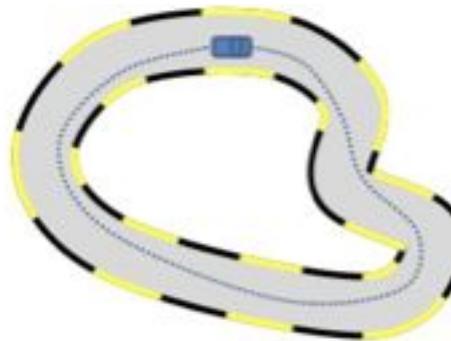
- Introduction
- Upper-Confidence-bound Action Selection
- Monte-Carlo Tree Search
- AlphaGo Zero
- MuZero, PlaNet

Imitation Learning in a Nutshell

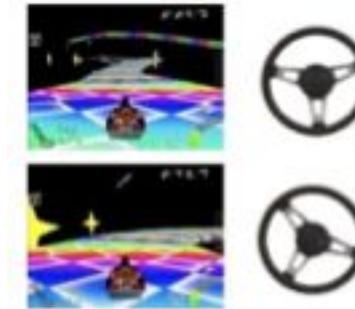
Given: demonstrations or demonstrator

Goal: train a policy to mimic demonstrations

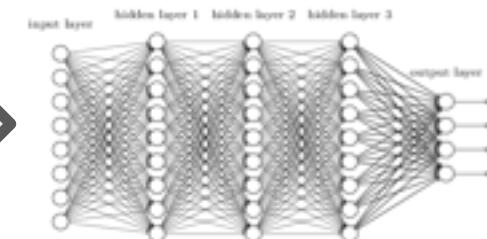
Expert Demonstrations



State/Action Pairs



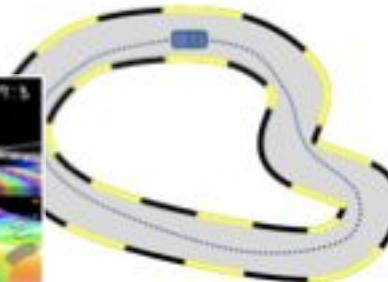
Learning



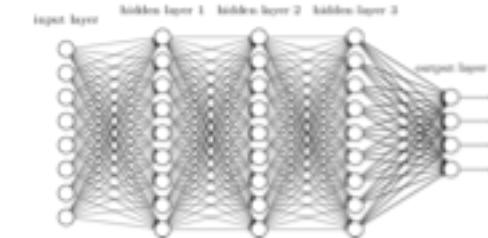
Ingredients of Imitation Learning



Demonstrations or Demonstrator



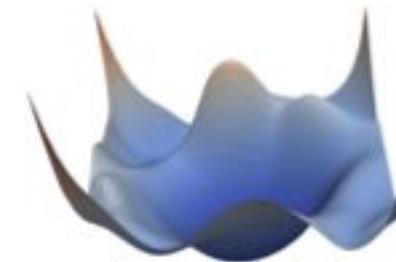
Environment / Simulator



Policy Class



Loss Function



Learning Algorithm



Some Interesting Examples

- **ALVINN**

<https://www.ri.cmu.edu/publications/alvinn-an-autonomous-land-vehicle-in-a-neural-network/>

Dean Pomerleau et al., 1989-1999 <https://www.youtube.com/watch?v=iiP4aPDTBPE>

- **Helicopter Acrobatics**

Learning for Control from Multiple Demonstrations - Adam Coates, Pieter Abbeel, Andrew Ng, ICML 2008

An Application of Reinforcement Learning to Aerobatic Helicopter Flight - Pieter Abbeel, Adam Coates, Morgan Quigley, Andrew Y. Ng, NIPS 2006

<https://www.youtube.com/watch?v=0JL04JJjocc>

- **Ghosting (Sports Analytics) - Next Slide.**

Ghosting



Data Driven Ghosting using
Deep Imitation Learning

Hoang M. Le et al., SSAC 2017

English Premier League
2012-2013

Match date: 04/05/2013

<https://www.youtube.com/watch?v=Wl-WL2cj0CA>



Notation & Set-up

State: s (sometimes x) (**state may only be partially observed)

Action: a (sometimes y)

Policy: π_θ (sometimes h)

- Policy maps states to actions: $\pi_\theta(s) \rightarrow a$
- ...or distributions over actions: $\pi_\theta(s) \rightarrow P(a)$

State Dynamics: $P(s'|s,a)$

- Typically not known to policy.
- Essentially the simulator/environment



Notation & Set-up

Rollout: sequentially execute $\pi(s_0)$ on an initial state

- Produce trajectory $\tau = (s_0, a_0, s_1, a_1, \dots)$

$P(\tau|\pi)$: distribution of trajectories induced by a policy

1. Sample s_0 from P_0 (distribution over initial states), initialize $t = 1$.
2. Sample action a_t from $\pi(s_{t-1})$
3. Sample next state s_t from applying a_t to s_{t-1} (requires access to environment)
4. Repeat from Step 2 with $t=t+1$

$P(s|\pi)$: distribution of states induced by a policy

- Let $P_t(s|\pi)$ denote distribution over t -th state
- $P(s|\pi) = (1/T)\sum_t P_t(s|\pi)$

Example #1: Racing Game

(Super Tux Kart)

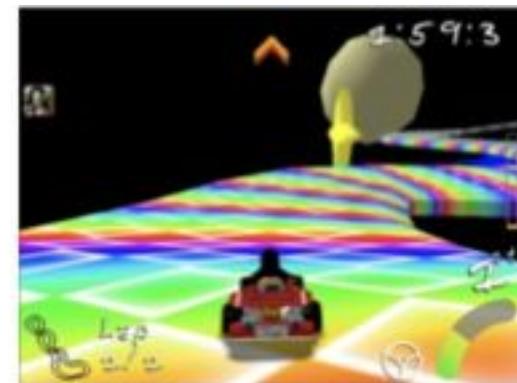
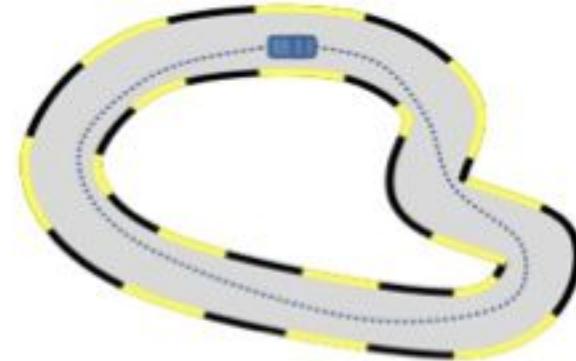
s = game screen

a = turning angle

Training set: $D=\{r:=(s,a)\}$ from π^*

- s = sequence of s
- a = sequence of a

Goal: learn $\pi_\theta(s) \rightarrow a$



Images from Stephane Ross

Example #2: Basketball Trajectories

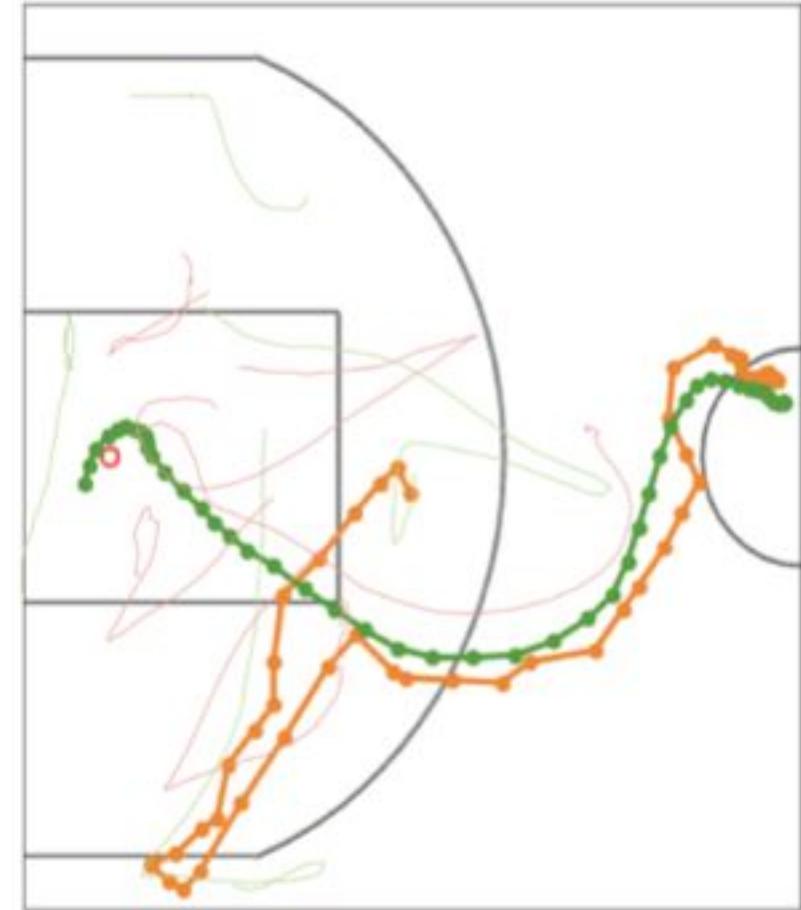
s = location of players & ball

a = next location of player

Training set: $D=\{r:=(s,a)\}$ from π^*

- s = sequence of s
- a = sequence of a

Goal: learn $\pi_\theta(s) \rightarrow a$



Behavioral Cloning = Reduction to Supervised Learning (Ignoring regularization for brevity.)

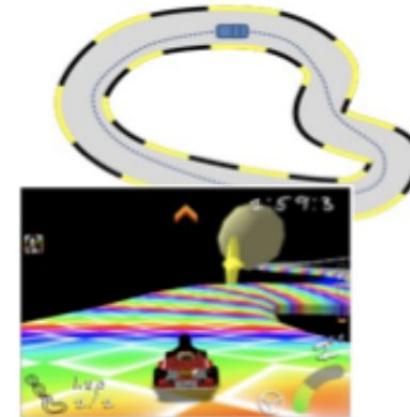
Define $P^* = P(s|\pi^*)$ (distribution of states visited by expert)

(recall $P(s|\pi^*) = (1/T)\sum_t P_t(s|\pi^*)$)

(sometimes abuse notation: $P^* = P(s, a^*=\pi^*(s)|\pi^*)$)

Learning objective:

$$\operatorname{argmin}_{\theta} E_{(s,a^*) \sim P^*} L(a^*, \pi_\theta(s))$$



Interpretations:

1. Assuming perfect imitation so far, learn to continue imitating perfectly
2. Minimize 1-step deviation error along the expert trajectories

Behavioral Cloning vs. Imitation Learning

Behavioral Cloning (Supervised Learning):

$$\operatorname{argmin}_{\theta} E_{(s,a^*) \sim P^*} L(a^*, \pi_\theta(s))$$

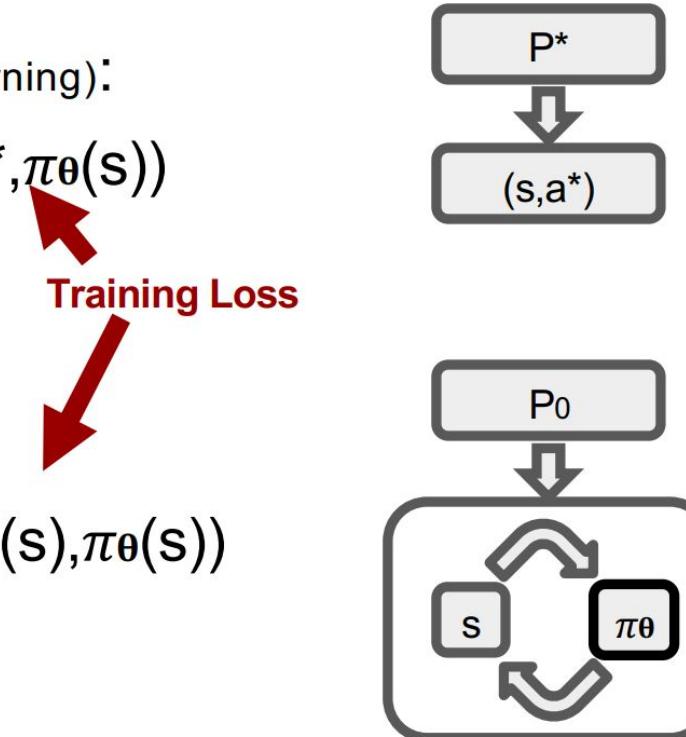
Distribution provided exogenously

(General) Imitation Learning:

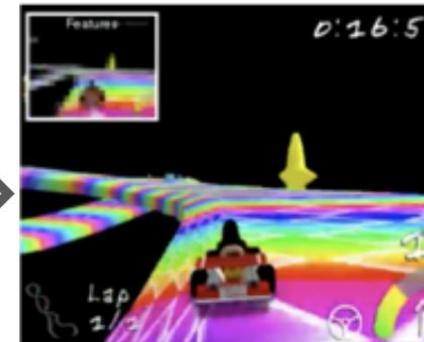
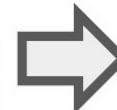
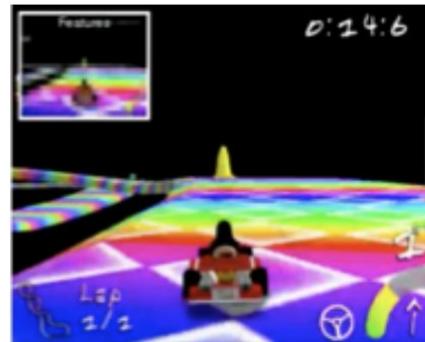
$$\operatorname{argmin}_{\theta} E_{s \sim P(s|\theta)} L(\pi^*(s), \pi_\theta(s))$$

Distribution depends on rollout.

$P(s|\theta)$ = state distribution of π_θ



Limitations of Behavioral Cloning



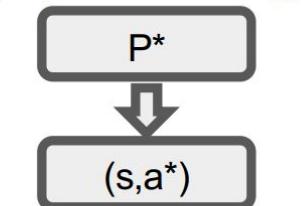
π_θ makes a mistake

New state sampled not from P^* !

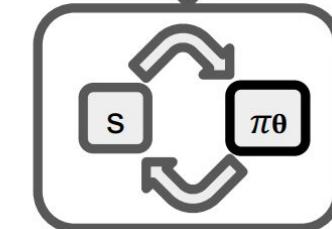
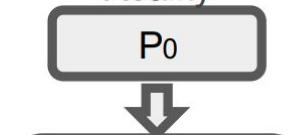
Worst case is catastrophic!

Images from Stephane Ross

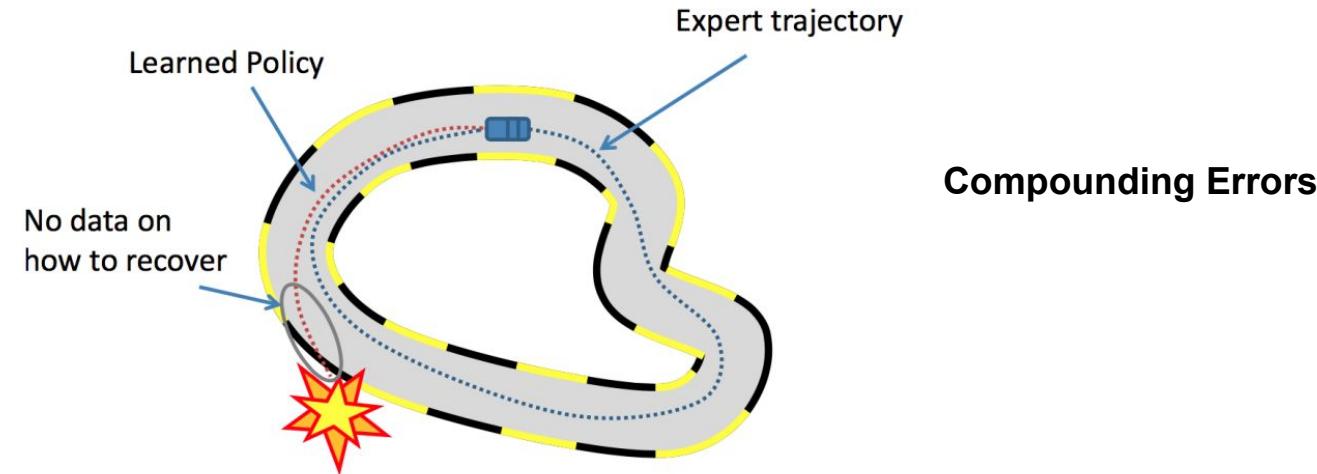
IID Assumption
(Supervised Learning)



Reality



Limitations of Behavioral Cloning



Data distribution mismatch!

In supervised learning, $(x, y) \sim D$ during train **and** test. In MDPs:

- Train: $s_t \sim D_{\pi^*}$
- Test: $s_t \sim D_{\pi_\theta}$



When to use Behavioral Cloning?

Advantages

- Simple
- Simple
- Efficient

Disadvantages

- Distribution mismatch between training and testing
- No long term planning

Use When:

- 1-step deviations not too bad
- Learning reactive behaviors
- Expert trajectories “cover” state space

Don't Use When:

- 1-step deviations can lead to catastrophic error
- Optimizing long-term objective (at least not without a stronger model)

Types of Imitation Learning

Behavioral Cloning

$$\operatorname{argmin}_{\theta} E_{(s,a^*) \sim P^*} L(a^*, \pi_\theta(s))$$

Works well when P^* close to P_θ

Inverse RL

Learn r such that:

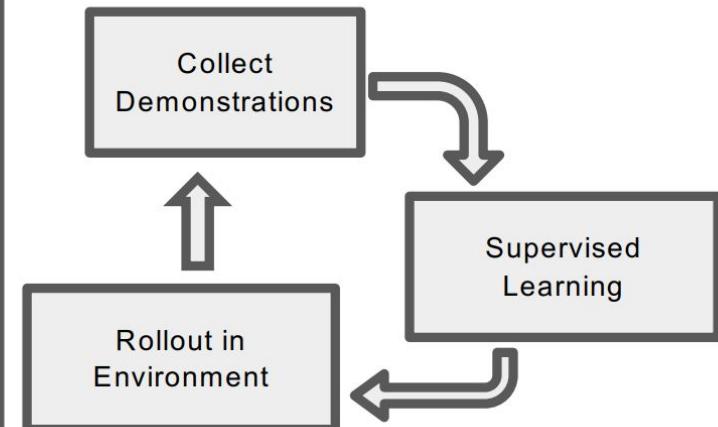
$$\pi^* = \operatorname{argmax}_{\theta} E_{s \sim P(s|\theta)} r(s, \pi_\theta(s))$$

RL problem

Assumes learning r is statistically easier than directly learning π^*

Direct Policy Learning

via Interactive Demonstrator



**Requires Interactive Demonstrator
(BC is 1-step special case)**



Types of Imitation Learning

	Direct Policy Learning	Reward Learning	Access to Environment	Interactive Demonstrator	Pre-collected Demonstrations
Behavioral Cloning	Yes	No	No	No	Yes
Direct Policy Learning (Interactive IL)	Yes	No	Yes	Yes	Optional
Inverse Reinforcement Learning	No	Yes	Yes	No	Yes

Interactive Expert

Can query expert at any state

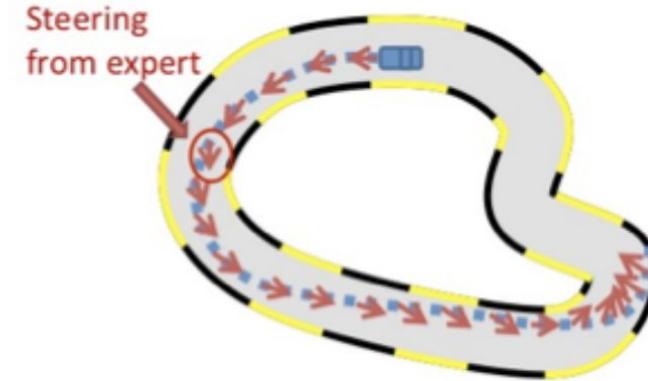
Construct loss function

- $L(\pi^*(s), \pi(s))$

Typically applied to rollout trajectories

- $s \sim P(s|\pi)$

Driving example: $L(\pi^*(s), \pi(s)) = (\pi^*(s) - \pi(s))^2$



Example from Super Tux Kart
(Image courtesy of Stephane Ross)

Interactive Expert

Can query expert at any state

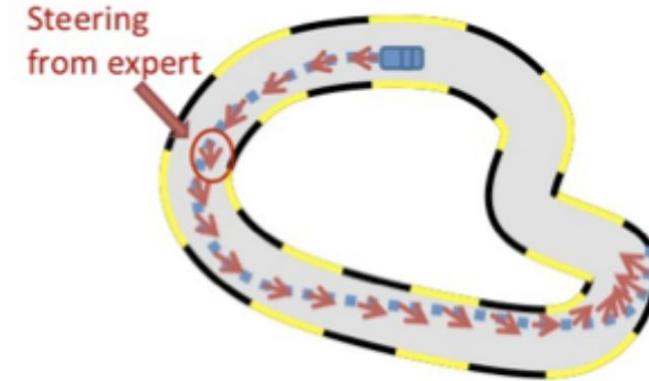
Construct loss function

- $L(\pi^*(s), \pi(s))$

Typically applied to rollout trajectories

- $s \sim P(s|\pi)$

Driving example: $L(\pi^*(s), \pi(s)) = (\pi^*(s) - \pi(s))^2$



Example from Super Tux Kart
(Image courtesy of Stephane Ross)

DAGGER: Dataset Aggregation

Initialize $\mathcal{D} \leftarrow \emptyset$.

Initialize $\hat{\pi}_1$ to any policy in Π .

for $i = 1$ **to** N **do**

 Let $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$.

 Sample T -step trajectories using π_i .

 Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i
 and actions given by expert.

 Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$.

 Train classifier $\hat{\pi}_{i+1}$ on \mathcal{D} .

end for

Return best $\hat{\pi}_i$ on validation.

- Idea: Get more labels of the expert action along the path taken by the policy computed by behavior cloning
- Obtains a stationary deterministic policy with good performance under its induced state distribution



Inverse RL

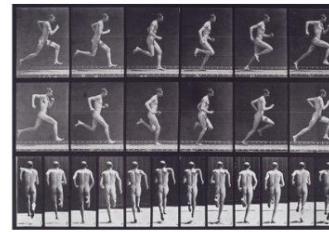
- What if we don't have an online demonstrator?
 - We only have access to an offline set of demonstrated trajectories
- Behavioral cloning is not robust
 - Suffers from overfitting
 - We know what to do in observed states but can't generalize well to other states
- How can we learn to mimic the demonstrator in a general way?
 - Learn the demonstrator's objective (reward) function
 - Apply RL



Inverse RL

- What if we don't have an online demonstrator?
 - We only have access to an offline set of demonstrated trajectories
- Behavioral cloning is not robust
 - Suffers from overfitting
 - We know what to do in observed states but can't generalize well to other states
- How can we learn to mimic the demonstrator in a general way?
 - Learn the demonstrator's objective (reward) function
 - Apply RL

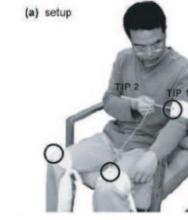
Inverse RL



Muybridge (c. 1870)



Mombaur et al. '09



Li & Todorov '06



Ziebart '08

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)$$

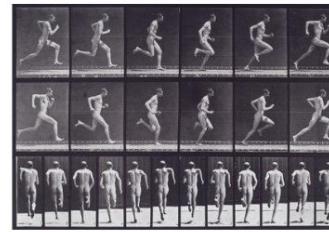
$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

$$\pi = \arg \max_{\pi} E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

$$\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)$$

optimize this to explain the data

Inverse RL



Muybridge (c. 1870)



Mombaur et al. '09



Li & Todorov '06



Ziebart '08

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

$$\pi = \arg \max_{\pi} E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

$$\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)$$

optimize this to explain the data

Inverse RL

The imitation learning perspective

Standard imitation learning:

- copy the *actions* performed by the expert
- no reasoning about outcomes of actions



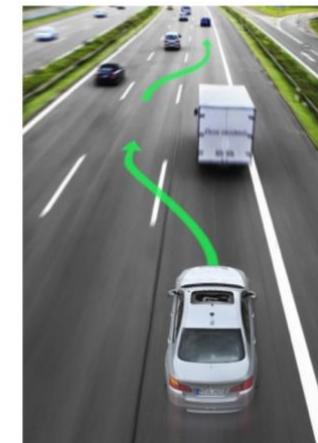
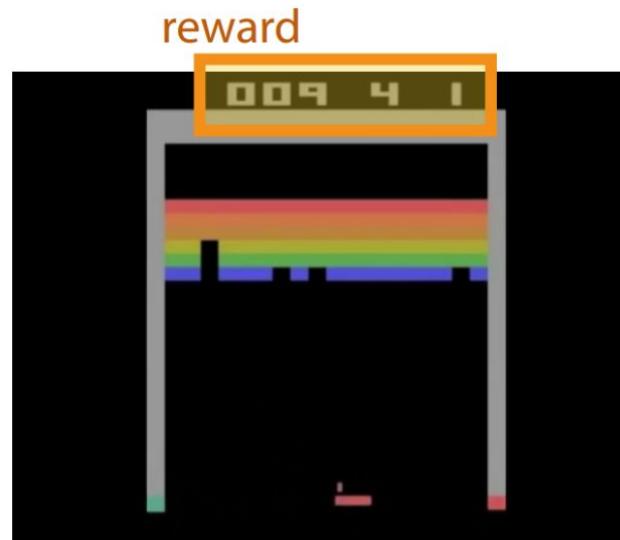
Human imitation learning:

- copy the *intent* of the expert
- might take very different actions!



Inverse RL

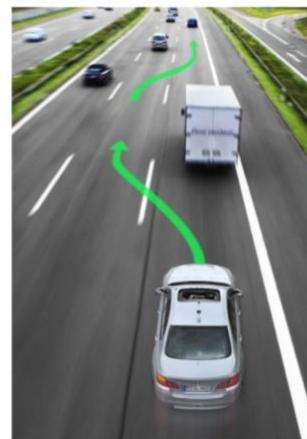
The reinforcement learning perspective



what is the reward?

Inverse RL

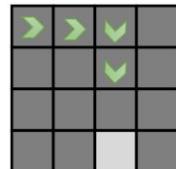
Infer reward functions from demonstrations



$$r(s, a)$$

by itself, this is an **underspecified** problem

many reward functions can explain the **same** behavior



Inverse RL

"forward" reinforcement learning

given:

states $\mathbf{s} \in \mathcal{S}$, actions $\mathbf{a} \in \mathcal{A}$

(sometimes) transitions $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

reward function $r(\mathbf{s}, \mathbf{a})$

learn $\pi^*(\mathbf{a}|\mathbf{s})$

inverse reinforcement learning

given:

states $\mathbf{s} \in \mathcal{S}$, actions $\mathbf{a} \in \mathcal{A}$

(sometimes) transitions $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

samples $\{\tau_i\}$ sampled from $\pi^*(\tau)$

learn $r_\psi(\mathbf{s}, \mathbf{a})$

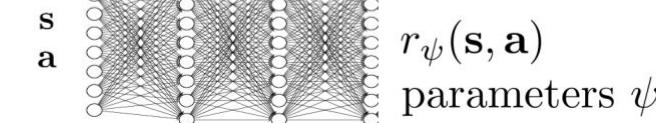
reward parameters

...and then use it to learn $\pi^*(\mathbf{a}|\mathbf{s})$

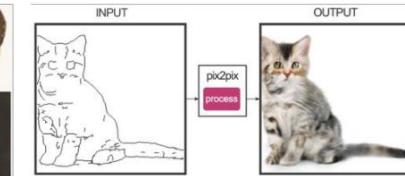
neural net reward function:

linear reward function:

$$r_\psi(\mathbf{s}, \mathbf{a}) = \sum_i \psi_i f_i(\mathbf{s}, \mathbf{a}) = \psi^T \mathbf{f}(\mathbf{s}, \mathbf{a})$$



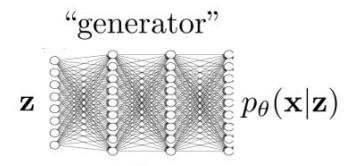
GAN



Zhu et al. '17

Arjovsky et al. '17

Isola et al. '17



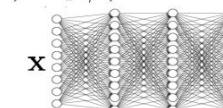
samples from $p_\theta(\mathbf{x})$

data (“demonstrations”)



$D(\mathbf{x}) = p_\psi(\text{real image}|\mathbf{x})$

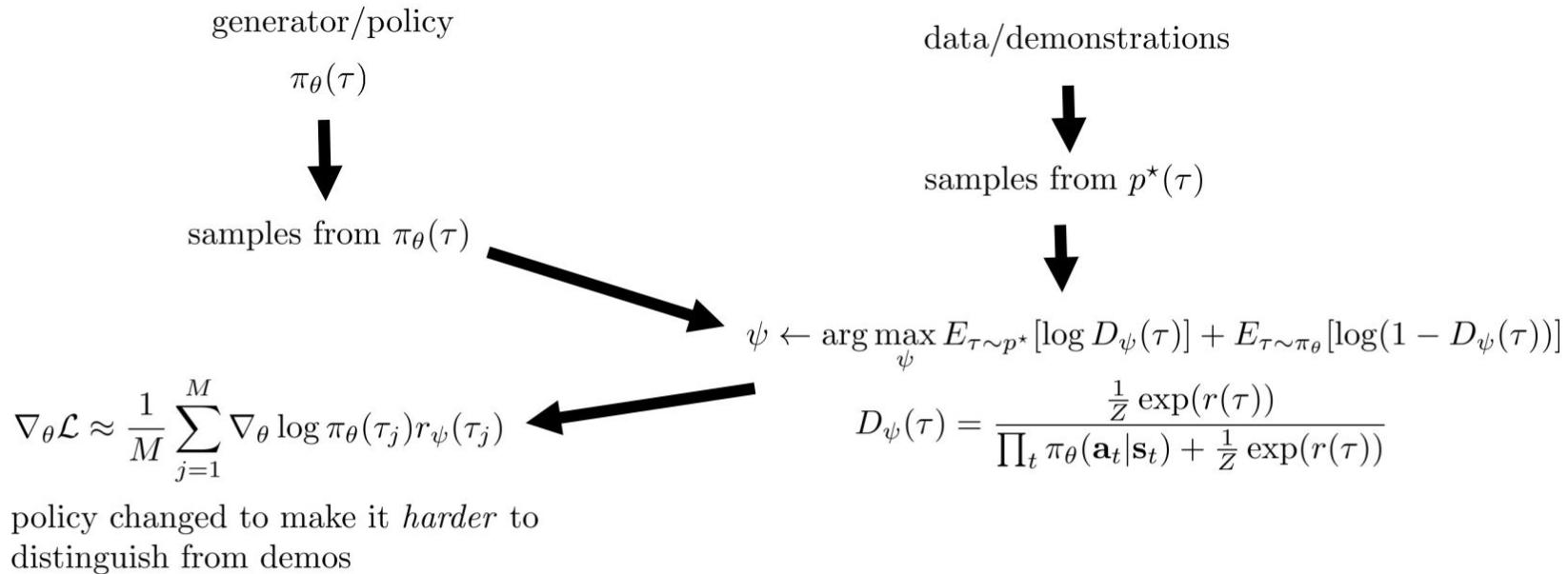
samples from $p^*(\mathbf{x})$



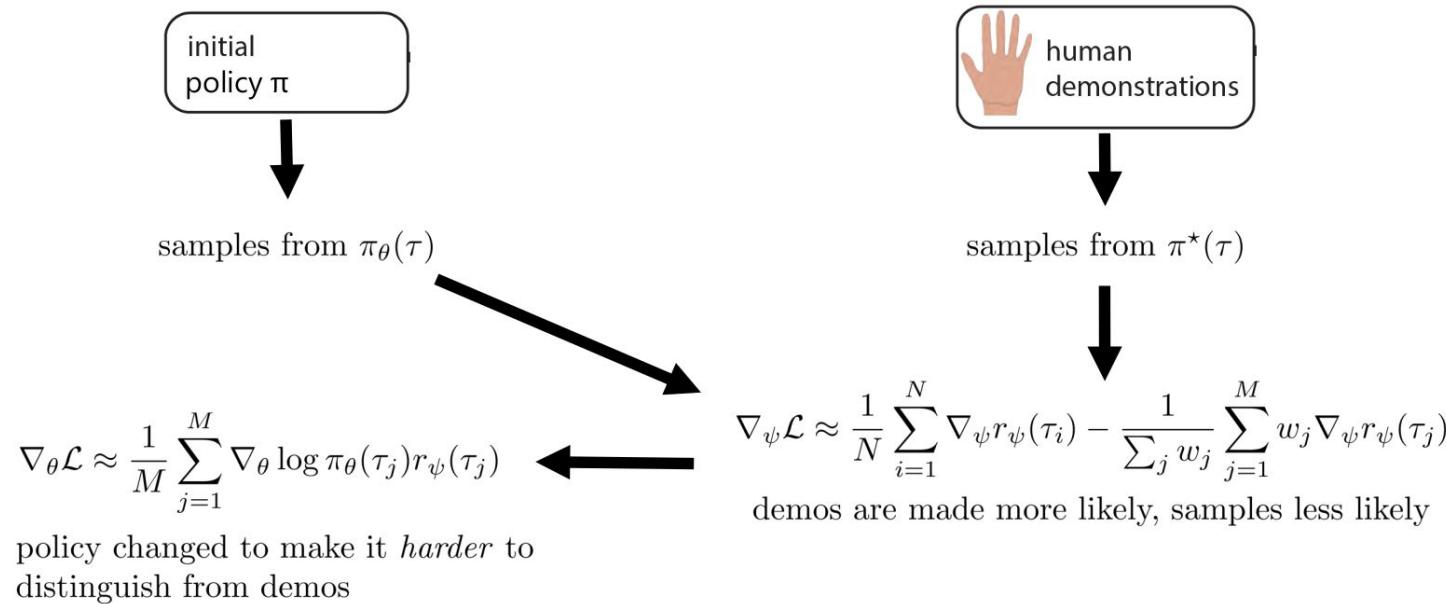
$$\psi = \arg \max_{\psi} \frac{1}{N} \sum_{\mathbf{x} \sim p^*} \log D_\psi(\mathbf{x}) + \frac{1}{M} \sum_{\mathbf{x} \sim p_\theta} \log(1 - D_\psi(\mathbf{x}))$$

$$\theta \leftarrow \arg \max_{\theta} E_{\mathbf{x} \sim p_\theta} \log D_\psi(\mathbf{x})$$

Inverse RL as GAN



Inverse RL as GAN





Model based algorithms - Advantages

- Very appealing : it can play out scenarios and understand the consequences of its actions without having to actually act in an environment.
- Require many fewer samples of data to learn good policies since having a model enables an agent to supplement its actual experiences with imagined ones.



Required Readings and references

1.



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



Deep Reinforcement Learning
2022-23 Second Semester, M.Tech (AIML)

Session #16: Multi-Agent Reinforcement Learning

Instructors :

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Prof. Sangeetha Viswanathan (sangeetha.viswanathan@pilani.bits-pilani.ac.in)



Agenda for the classes

- Introduction
- Cooperative vs Competitive agents,
- centralized vs. decentralized RL ;
- Proximity Primal Optimization (Surrogate Objective Function, Clipping)



Multi-Agent RL (MARL)

Vanilla reinforcement learning is concerned with a single agent, in an environment, seeking to maximize the total reward in that environment.

A robot learning to walk, where its overall goal is to walk without falling.

It receives rewards for taking steps without falling over, and through trial and error, and maximizing these rewards, the robot eventually learns to walk.

In this context, we have a *single* agent seeking to accomplish a goal through maximizing total rewards.



MARL

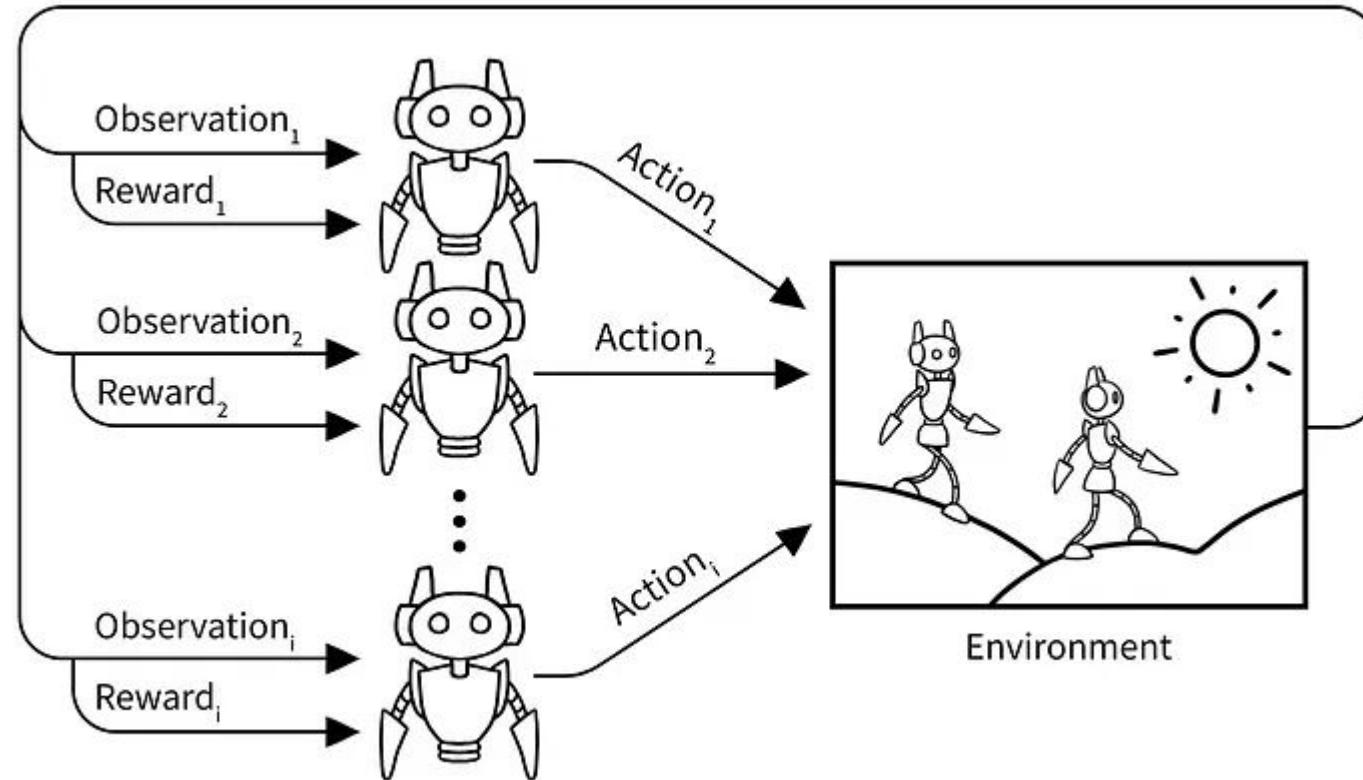
Multi-agent reinforcement learning studies how multiple agents interact in a common environment.

That is, when these agents interact with the environment and one another, can we observe them collaborate, coordinate, compete, or collectively learn to accomplish a particular task.

In general it's the same as single agent reinforcement learning, where each agent is trying to learn its own policy to optimize its own reward.

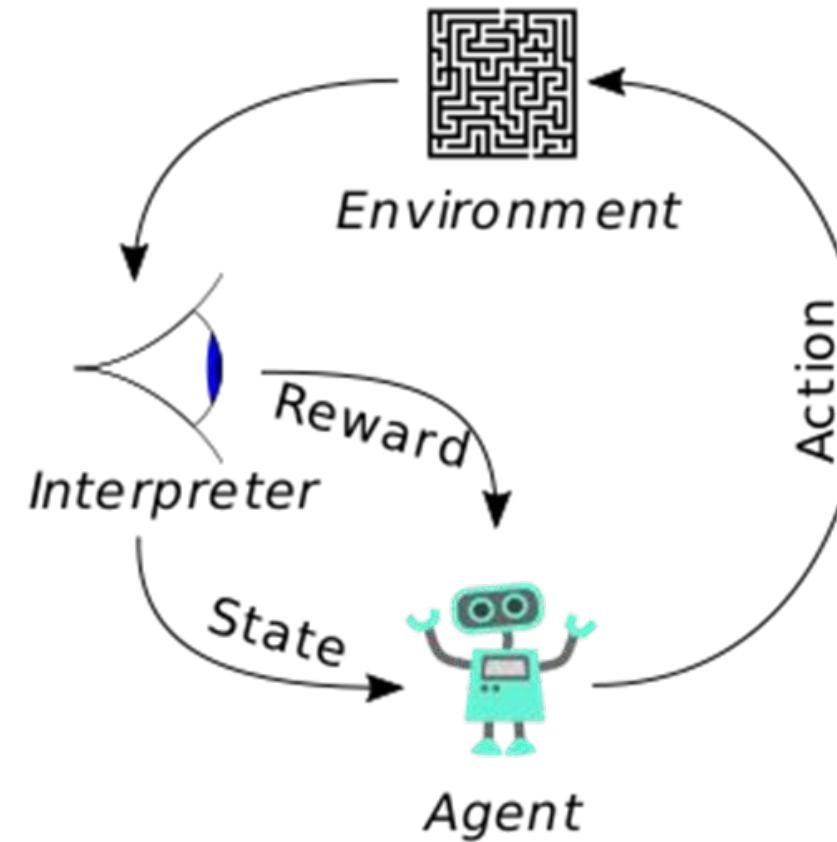
Using a central policy for all agents is possible, but multiple agents would have to communicate with a central server to compute their actions (which is problematic in most real world scenarios), so in practice *decentralized* multi-agent reinforcement learning is used.

MARL

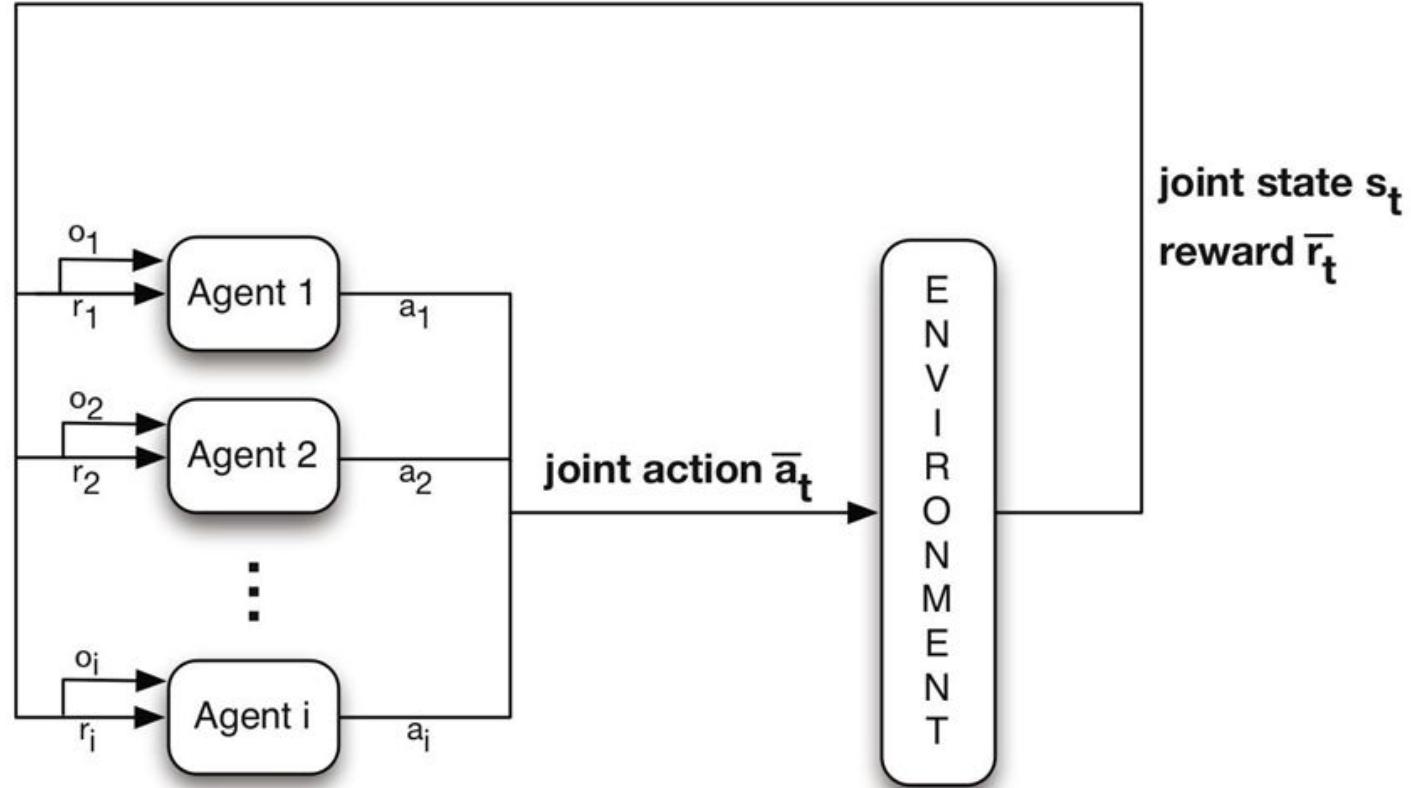




Traditional (Single-Agent) RL



Multiagent RL





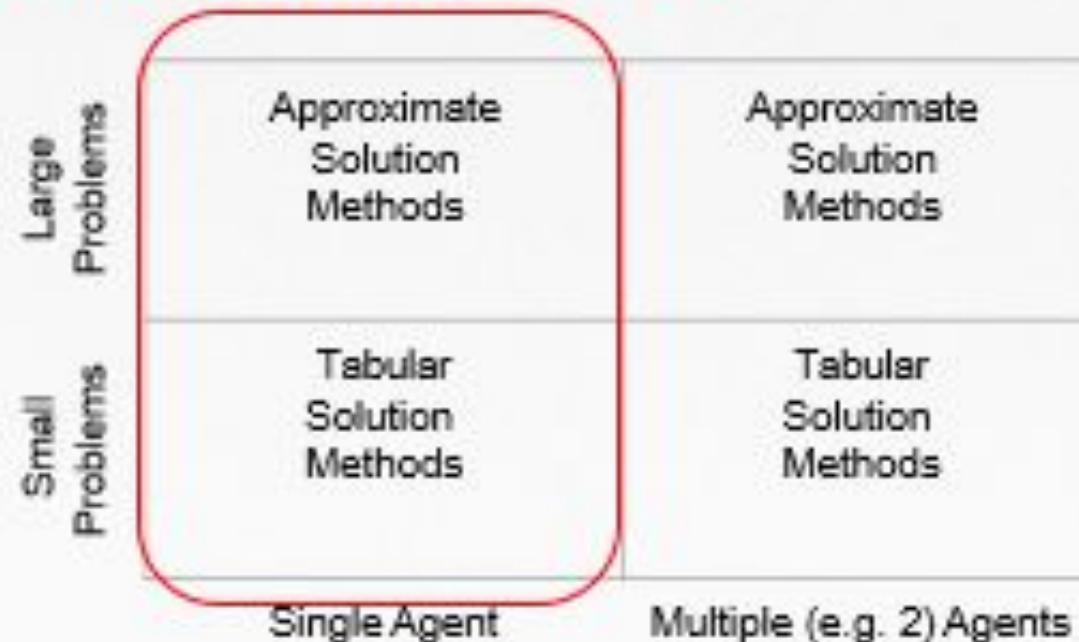
Motivations: Research in Multiagent RL

	Large Problems	Approximate Solution Methods	Approximate Solution Methods
	Small Problems	Tabular Solution Methods	Tabular Solution Methods
Single Agent		Multiple (e.g. 2) Agents	



Motivations: Research in Multiagent RL

Sutton & Barto '98, '18



Motivations: Research in Multiagent RL

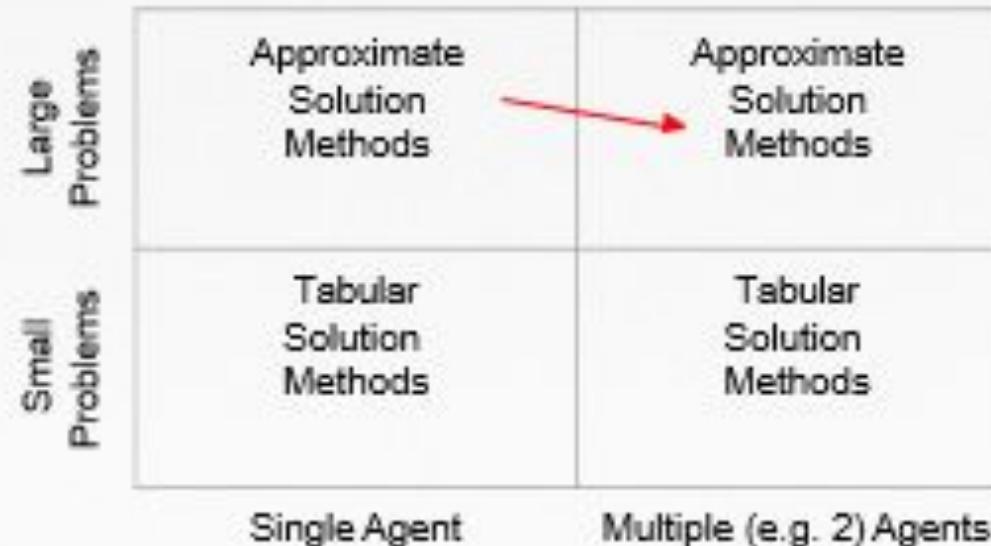
First era of multiagent RL





Motivations: Research in Multiagent RL

Multiagent Deep RL era ('16 - now)





Dimensions of MARL

Centralized:

- One brain / algorithm deployed across many agents

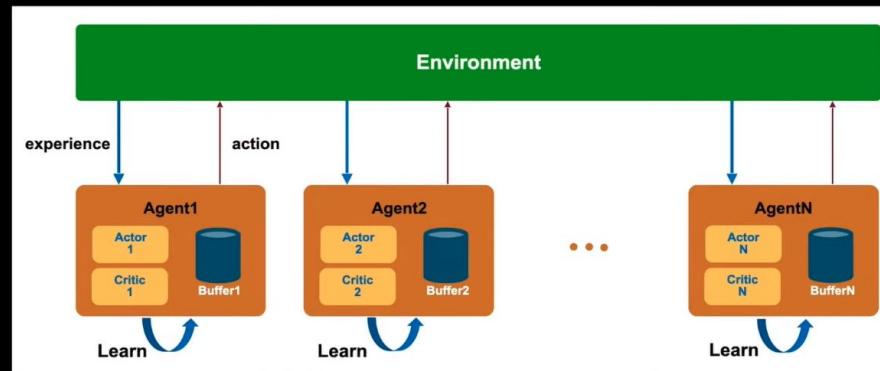
Decentralized:

- All agents learn individually
- Communication limitations defined by environment

Decentralized systems

MARL Approaches

Decentralized



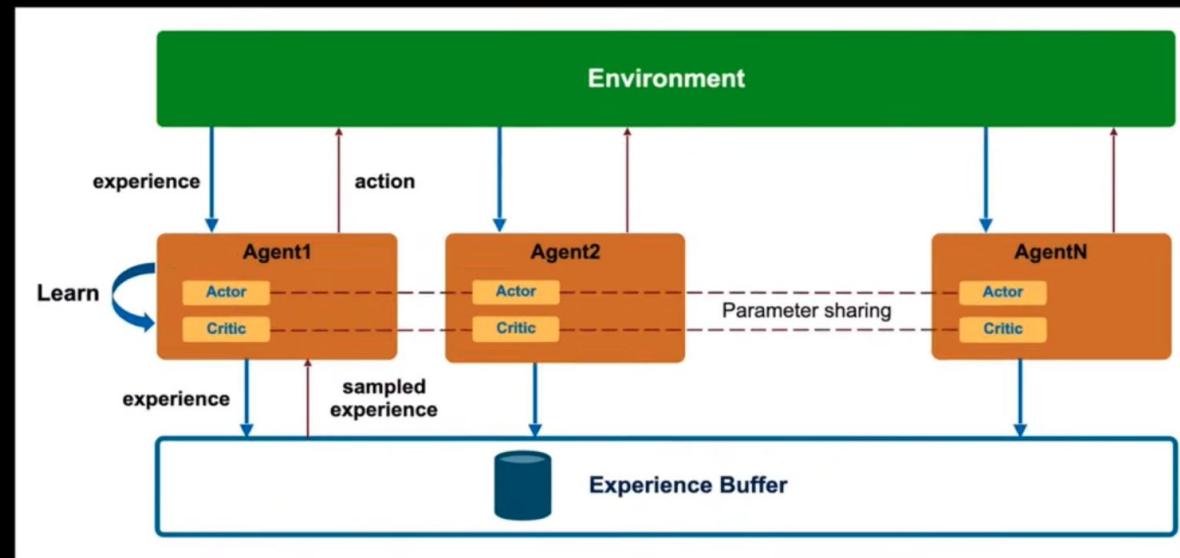


Decentralized systems

- In decentralized learning, each agent is trained independently from the others.
- The benefit is that since no information is shared between agents, these vacuums can be designed and trained like we train single agents.
- The idea here is that our training agent will consider other agents as part of the environment dynamics. Not as agents.
- However, the big drawback of this technique is that it will make the environment non-stationary since the underlying Markov decision process changes over time as other agents are also interacting in the environment. And this is problematic for many Reinforcement Learning algorithms that can't reach a global optimum with a non-stationary environment.

Centralized systems

Centralized





Centralized systems

- In this architecture, we have a high-level process that collects agents' experiences: the experience buffer. And we'll use these experiences to learn a common policy.
- We use that collective experience to train a policy that will move all three robots in the most beneficial way as a whole. So each robot is learning from their common experience. We now have a stationary environment since all the agents are treated as a larger entity, and they know the change of other agents' policies (since it's the same as theirs).



Dimensions of MARL

Prescriptive:

- Suggests how agents *should* behave

Descriptive:

- Forecast how agent *will* behave



Dimensions of MARL

Cooperative: Agents cooperate to achieve a goal

- Shared team reward
- For instance, in a warehouse, robots must collaborate to load and unload the packages efficiently (as fast as possible).

Competitive: Agents compete against each other

- Zero-sum games
- Individual opposing rewards
- For example, in a game of tennis, each agent wants to beat the other agent.

Neither: Agents maximize their utility which may require cooperating and/or competing

- General-sum games
- like in our SoccerTwos environment, two agents are part of a team (blue or purple): they need to cooperate with each other and beat the opponent team.



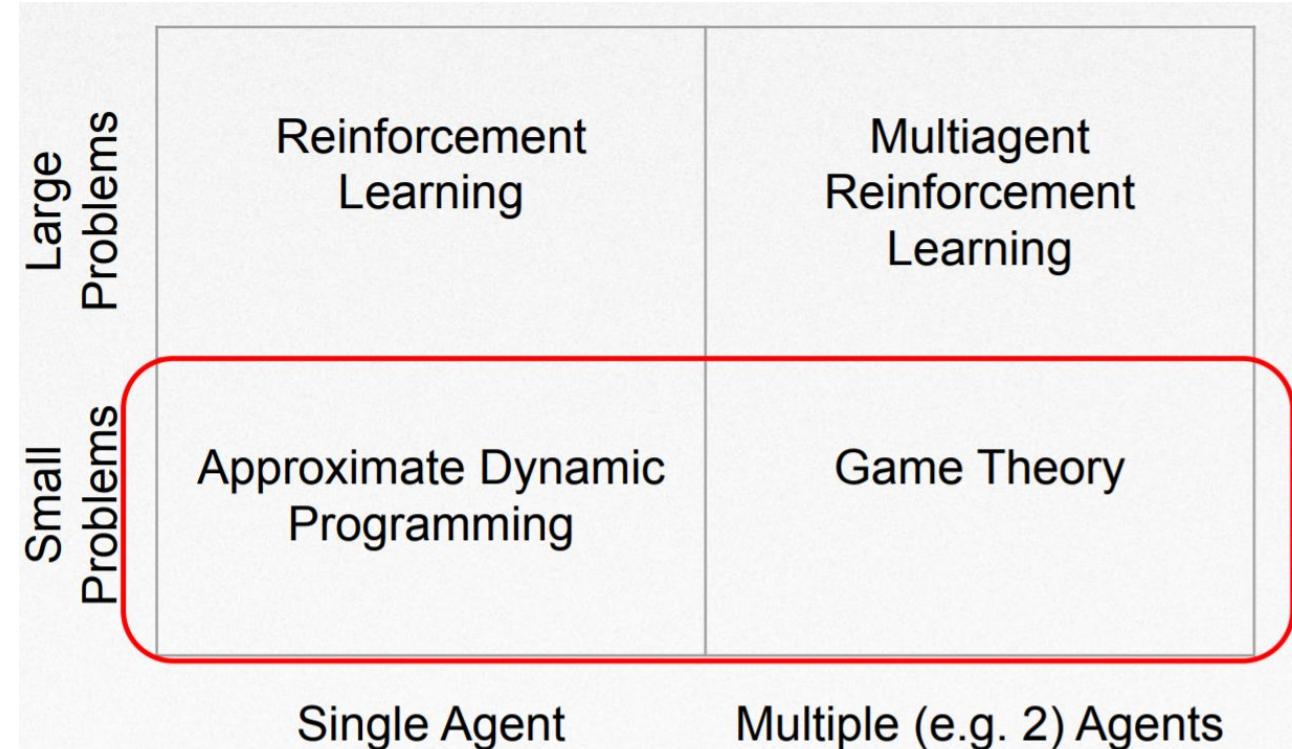
Dimensions of MARL

Numbers of agents

- One (single-agent)
- Two (very common)
- Finite
- Infinite



Foundations of MARL



Benefits of Multi-agent Learning systems

- **Sharing experience**

via communication, teaching, imitation

- **Parallel computation**

due to decentralized task structure

- **Robustness**

redundancy, having multiple agents to accomplish a task



Challenges of Multi-agent learning systems

■ Curse of dimensionality

Exponential growth in computational complexity from increase in state and action dimensions. Also a challenge for single-agent problems.

■ Specifying a good (learning) objective

Agent returns are correlated and cannot be maximized independently.

■ The system in which to learn is a moving target

As some agents learn, the system which contains these agents changes, and so may the best policy. Also called a system with non-stationary or time-dependent dynamics.

■ Need for coordination

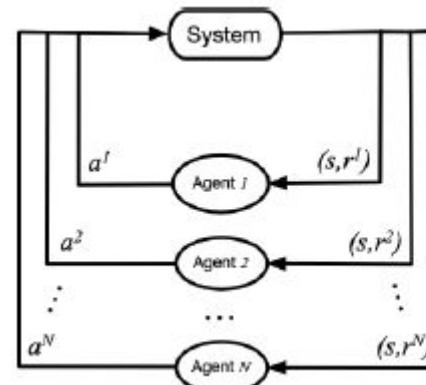
Agent actions affect other agents and could confuse other agents (or herself) if not careful. Also called destabilizing training.

Challenges of Multi-agent learning systems

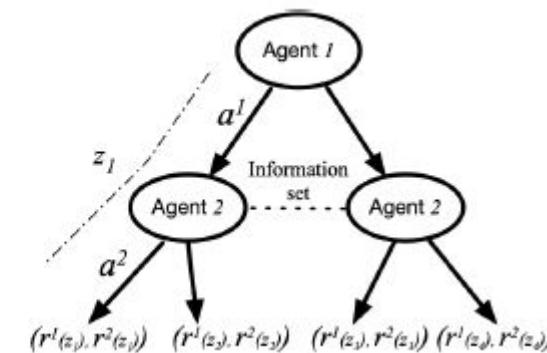
- In single agent RL, agents need only to adapt their behaviour in accordance with their own actions and how they change the environment. In MARL agents also need to adapt to other agents' learning and actions. The effect is that agents can execute the same action on the same state and receive different rewards.
- MARL agents do not always have a full view of the environment and even if they have, they normally cannot predict the actions of other agents and the changes in the environment
- The credit assignment problem - the difficulty of deciding which agent is responsible for successes or failures. How to split the reward signal among the agents and the trade-off between the use of local and global rewards to achieve fast learning or to guarantee to converge to a global optimal policy.

Theoretical framework of MARL

- Markov/Stochastic games - perfect information
- Extensive Form games - imperfect information



Markov game



Extensive-form game



Markov Game

Definition 2.2 A Markov game is defined by a tuple $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i \in \mathcal{N}}, \mathcal{P}, \{R^i\}_{i \in \mathcal{N}}, \gamma)$, where $\mathcal{N} = \{1, \dots, N\}$ denotes the set of $N > 1$ agents, \mathcal{S} denotes the state space observed by all agents, \mathcal{A}^i denotes the action space of agent i . Let $\mathcal{A} := \mathcal{A}^1 \times \dots \times \mathcal{A}^N$, then $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ denotes the transition probability from any state $s \in \mathcal{S}$ to any state $s' \in \mathcal{S}$ for any joint action $a \in \mathcal{A}$; $R^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function that determines the immediate reward received by agent i for a transition from (s, a) to s' ; $\gamma \in [0, 1)$ is the discount factor.

MARL Formulation

- The agents choose actions according to their policies.
- For agent j , the corresponding policy is defined as $\pi^j : S \rightarrow \Omega(A^j)$, where $\Omega(A^j)$ is the collection of probability distributions over agent j 's action space A^j .
- Let $\pi = [\pi^1, \dots, \pi^N]$ - is the joint policy of all agents, then

$$v_\pi^j(s) = v^j(s; \pi) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{\pi, p}[r_t^j | s_0 = s, \pi]$$

- Q-function such that the Q-function $Q_\pi^j : S \times A^1 \times \dots \times A^N \rightarrow \mathbb{R}$ of agent j under the joint policy π :

$$Q_\pi^j(s, \mathbf{a}) = r^j(s, \mathbf{a}) + \gamma \mathbb{E}_{s' \sim p}[v_\pi^j(s')]$$

Nash Q-learning

- In MARL, the objective of each agent is to learn an optimal policy to maximize its value function
- Optimizing the v_π^j for agent j depends on the joint policy π of all agents
- A **Nash equilibrium** is a joint policy π such that no player has incentive to deviate unilaterally. It is represented by a particular joint policy

$$\pi_* = [\pi_*^1, \dots, \pi_*^N]$$

such that for all $s \in S, j \in \{1, \dots, N\}$ it satisfies:

$$v^j(s; \pi_*) = v^j(s; \pi_*^j, \pi_*^{-j}) \geq v^j(s; \pi^j, \pi_*^{-j})$$

Here π_*^{-j} is the joint policy of all agents except j as

$$\pi_*^{-j} = [\pi_*^1, \dots, \pi_*^{j-1}, \pi_*^{j+1}, \dots, \pi_*^N]$$

Nash Q-Learning

- In a Nash equilibrium, each agent acts with the best response π_*^j to others, provided that all other agents follow the policy π_*^{-j}
- For a N -agent stochastic game, there is at least one Nash equilibrium with stationary policies, assuming players are rational
- Given Nash policy π_* , the Nash value function

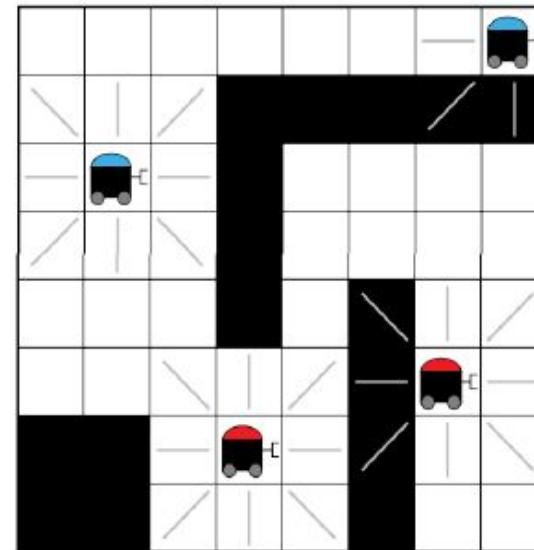
$$\mathbf{v}^{\text{Nash}} = [v_{\pi_*}^1(s), \dots, v_{\pi_*}^N(s)]$$

$$\mathbf{Q}(s, \mathbf{a})^{\text{Nash}} = \mathbb{E}_{s' \sim p}[\mathbf{r}(s, \mathbf{a}) + \gamma \mathbf{v}^{\text{Nash}}(s')]$$

where $\mathbf{r}(s, \mathbf{a}) = [r^1(s, \mathbf{a}), \dots, r^N(s, \mathbf{a})]$

Multi-agent Deep Q network - Eg: Pursuit Evasion

- n pursuit-evasion – a set of agents (the pursuers) are attempting to chase another set of agents (the evaders)
- The agents in the problem are self-interested (or heterogeneous), i.e. they have different objectives
- The **two pursuers** are attempting to catch the **two evaders**



Problem representation

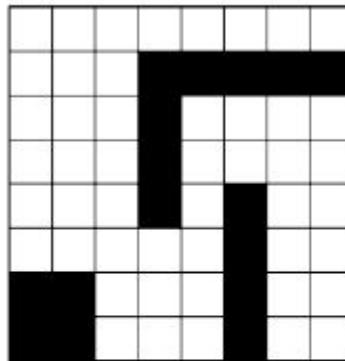
Challenge: defining the problem in such a way that an arbitrary number of agents can be represented without changing the architecture of the deep Q-Network.

Solution (under some assumptions):

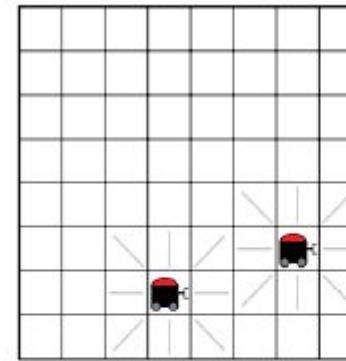
- The image tensor is of size $4 \times W \times H$, where W and H are the height and width of our two dimensional domain and four is the number of channels in the image.
- Channels:
 - **Background Channel:** contains information about any obstacles in the environment
 - **Opponent Channel:** contains information about all the opponents
 - **Ally Channel:** contains information about all the allies
 - **Self Channel:** contains information about the agent making the decision

Chanel Settings

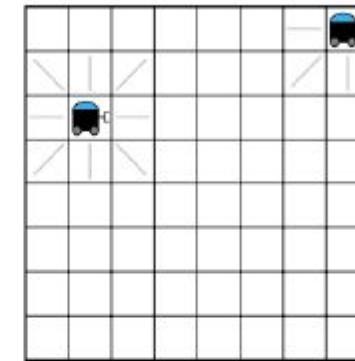
Background Channel



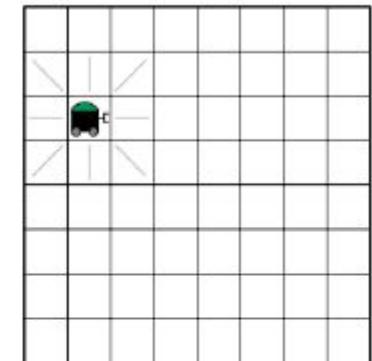
Opponent Channel



Ally Channel



Self Channel



Four Channel Image

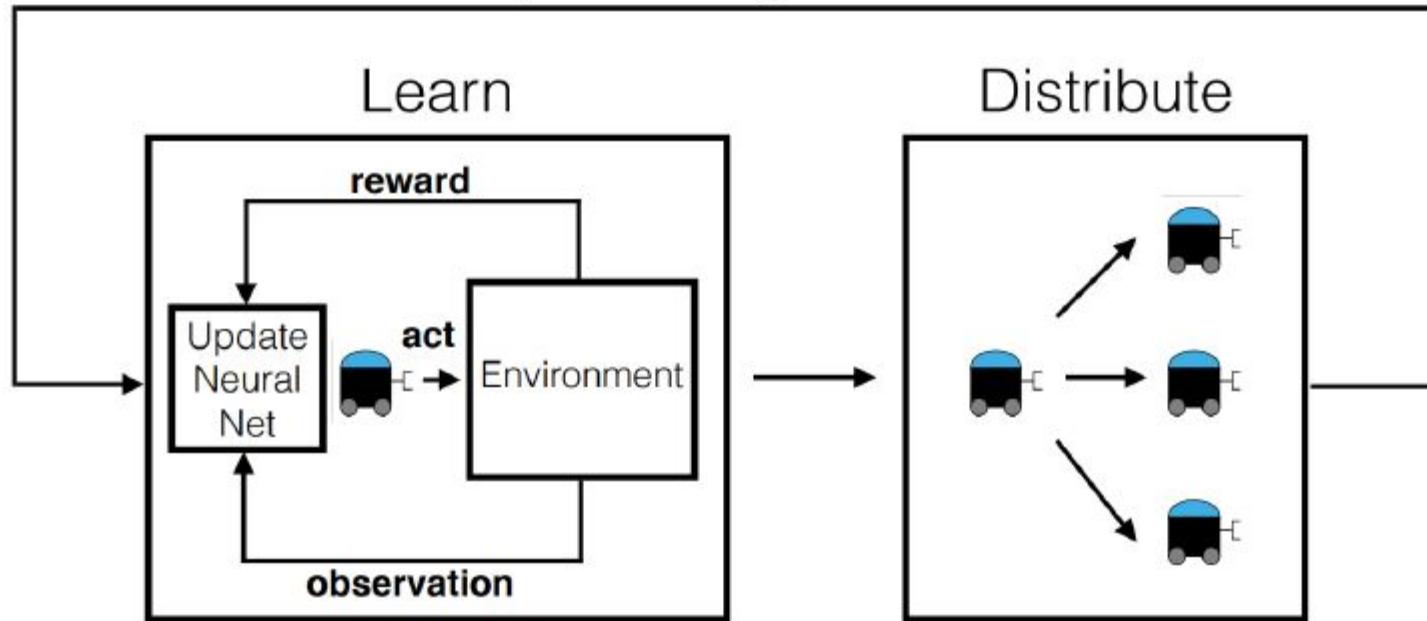


Multi-agent centralized training

- Train one agent at a time, and fix policies of all the other agents
- After a number of iterations distribute the policy learned by the training agent to all the other agents of its type

Multi-agent centralized training

Improved Agent Policies



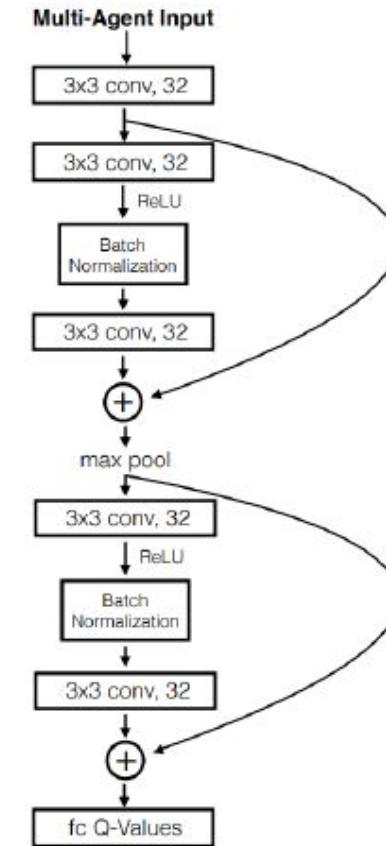


Dealing with agent ambiguity

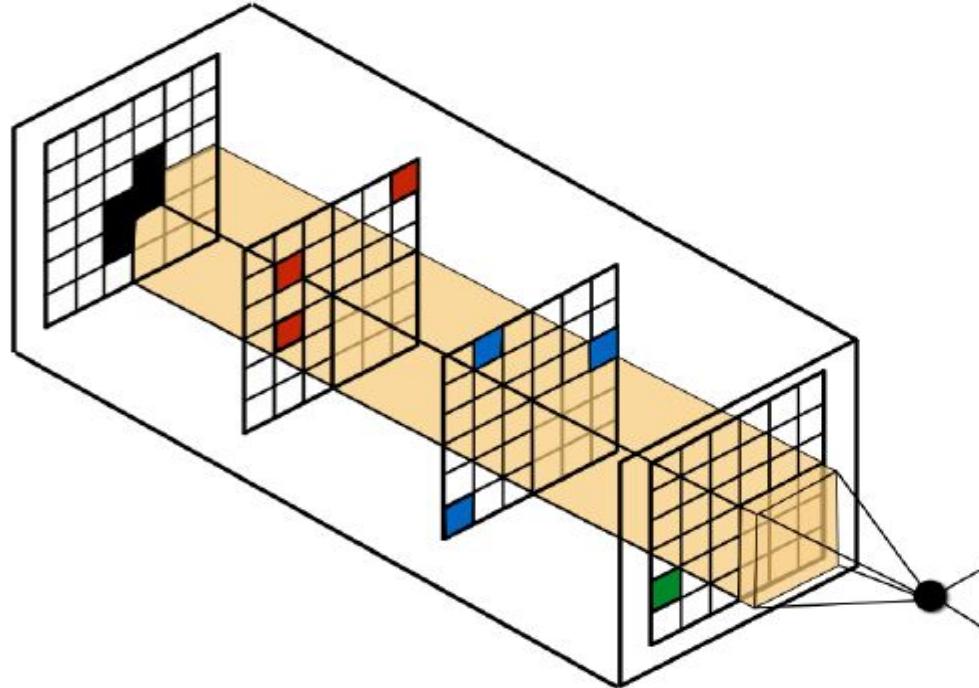
- **Challenge:** When two ally agents are occupying the same position in the environment, the image-like state representation for each agent will be identical, so their policies will be exactly the same.
- **Solution:** To break this symmetry – use a stochastic policy for agents. The actions taken by the agent are drawn from a distribution derived by taking a softmax over the Q-values of the neural network. This allows allies to take different actions if they occupy the same state and break the ambiguity.

MADQN Architecture - Residual network type

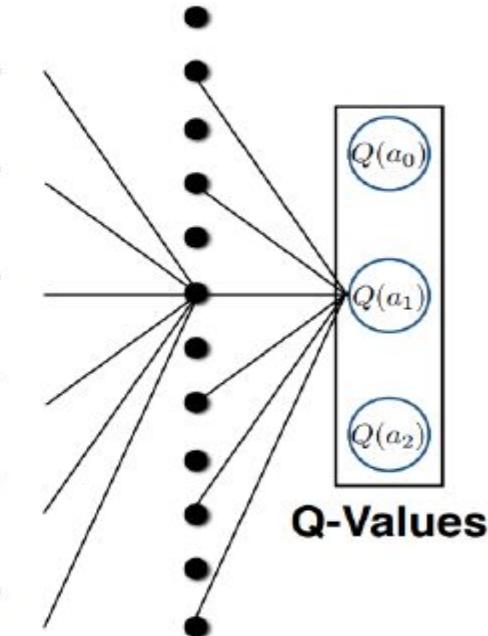
MADQN architecture – a Residual Network type architecture is used to improve gradient flow throughout the network



MADQN Architecture



Convolution



Fully connected



Proximal Policy Optimization (PPO)

The algorithm, introduced by OpenAI in 2017, seems to strike the right **balance between performance and comprehension**.

PPO aims to strike a balance between important factors like ease of implementation, ease of tuning, sample complexity, sample efficiency and trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.

PPO is in fact, a policy gradient method that learns from online data as well. It merely ensures that the updated policy isn't too much different from the old policy to ensure low variance in training.

PPO - Surrogate Objective

- surrogate objective - avoids performance collapse by guaranteeing monotonic policy improvement.

During the optimization process, we search over a sequence of policies $\pi_1, \pi_2, \pi_3, \dots, \pi_n$ within the set of all policies. This is known as the *policy space*¹ Π .

$$\Pi = \{\pi_i\} \tag{7.1}$$

With the policy parametrized as π_θ , we can also conveniently express the parameter space for θ . Each unique θ parametrizes an instance of the policy. The *parameter space* is Θ .

$$\Theta = \{\theta \in \mathbb{R}^m, \text{ where } m \text{ is the number of parameters}\} \tag{7.2}$$



Modifying the objective

- relative policy performance identity

$$J(\pi') - J(\pi) = \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^T \gamma^t A^\pi(s_t, a_t) \right]$$

- The relative policy performance identity $J(\pi) - J(\pi')$ serves as a metric to measure policy improvements. If the difference is positive, the newer policy π' is better than π .
- During a policy iteration, we should ideally choose a new policy π' such that this difference is maximized. Therefore, maximizing objective $J(\pi')$ is equivalent to maximizing this identity, and they can both be done by gradient ascent.

Surrogate objective

Framing the objective this way also means that every policy iteration should ensure a non-negative (monotonic) improvement—that is, $J(\pi') - J(\pi) \geq 0$ —since in the worst case we can simply let $\pi' = \pi$ to get no improvement. With this, there will be no performance collapses throughout the training, which is the property we are looking for.

$$\begin{aligned} J(\pi') - J(\pi) &= \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t \geq 0} A^\pi(s_t, a_t) \right] \\ &\approx \mathbb{E}_{\tau \sim \pi} \left[\sum_{t \geq 0} A^\pi(s_t, a_t) \frac{\pi'(a_t | s_t)}{\pi(a_t | s_t)} \right] = J_\pi^{\text{CPI}}(\pi') \end{aligned}$$



PPO Algorithm

- Page 174 - Foundations of Deep Reinforcement Learning: Theory and Practice in Python (Addison-Wesley Data & Analytics Series) 1st Edition by Laura Graesser and Wah Loon Keng



Required Readings and references

1. Foundations of Deep Reinforcement Learning: Theory and Practice in Python (Addison-Wesley Data & Analytics Series) 1st Edition by Laura Graesser and Wah Loon Keng



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you