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

Session #1:
Introduction to the Course

Instructors :
1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Dr. V Chandra Sekhar (chandrasekhar.v@wilp.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. V Chandra Sekhar

- **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**

  Two Quizzes for 5% each; **Best of two will be taken for 5%** (in final grading);

  Whatever be the points set for quizzes, the score will be scaled to 5%

  NO MAKEUP, for whatever be the reason. Ensure to attend at least one of the quizzes.

  Two Assignments - Tensorflow/ Pytorch / OpenAI Gym Toolkit → 25 %

  Assignment 1: Partially Numerical + Implementation of Classic Algorithms - 10%

  Assignment 2: Deep Learning based RL - 15%

  Mid-Term Exam - 30% [Only to be written in A4 pages, scanned and uploaded]

  Comprehensive Exam - 40% [Only to be written in A4 pages, scanned and uploaded]

- **Webinars/Tutorials**

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

- Teaching Assistants will be introduced to you in the next class

Course Operation

•Schedule of Schedule of Quizzes

Sunday, January 14, 2024	7:00:00 PM	Monday, January 15, 2024	7:00:00 PM
Sunday, March 10, 2024	7:00:00 PM	Monday, March 11, 2024	7:00:00 PM

•Schedule of Assignments

Assignment - #1: 02 Jan, 2024 7:00 PM 18 Jan, 2024 11:59 PM
Assignment - #2: 01, Mar 2024 7:00 PM 15, Mar 2024 11:59 PM

•Schedule of Webinars

21-Dec-23; 10-Jan-24; 22-Feb-24; 13-Mar-24

Course Operation

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

1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Dr. V Chandra Sekhar (chandrasekhar.v@wilp.bits-pilani.ac.in)

- **Plagiarism [Important]**

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

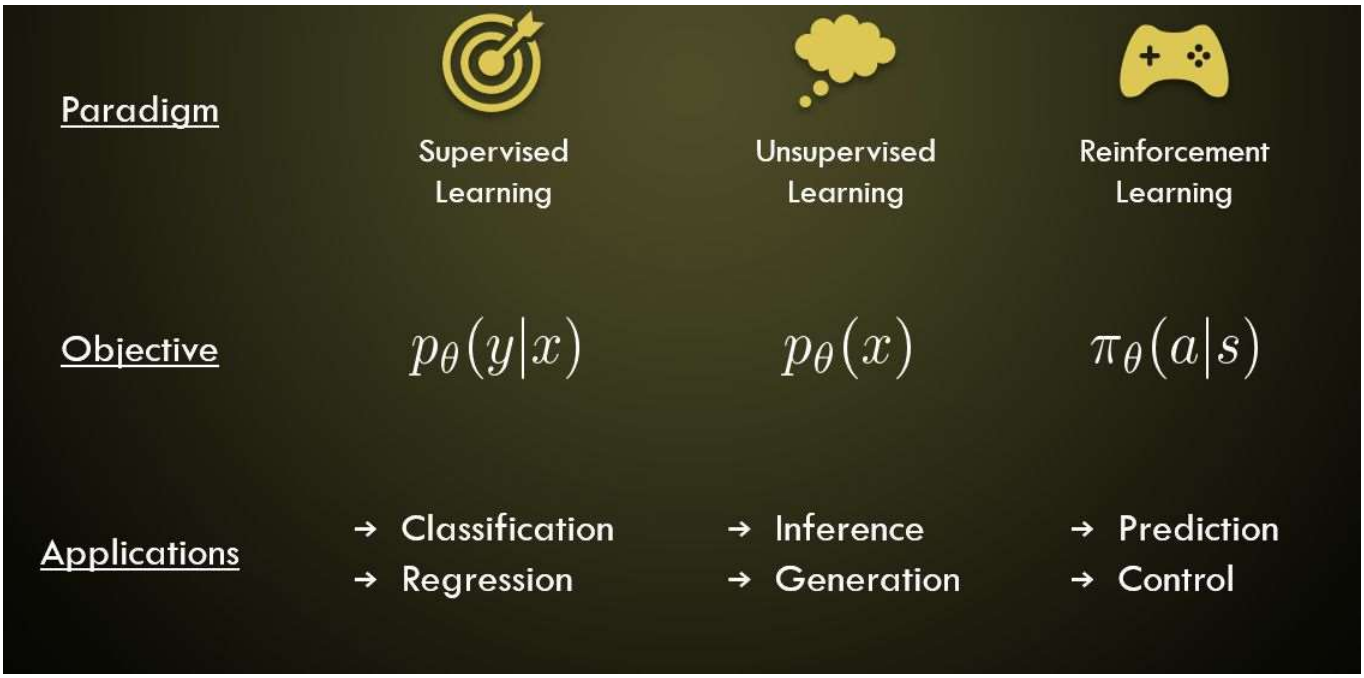
Reinforcement learning (RL) is based on rewarding desired behaviors or punishing undesired ones. Instead of one input producing one output, the algorithm produces a variety of outputs and is trained to select the right one based on certain variables – Gartner

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



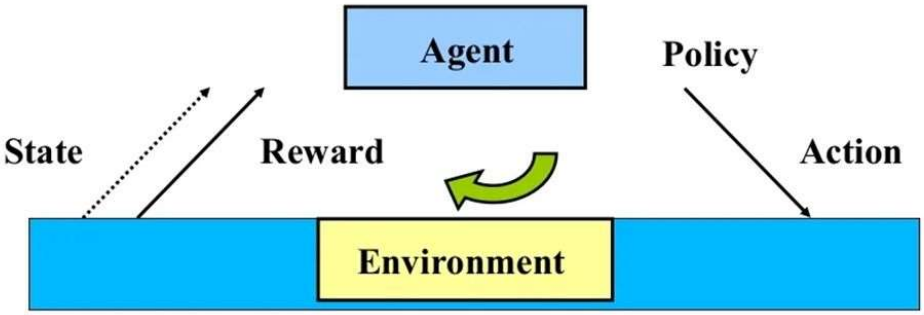
Types of Learning

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

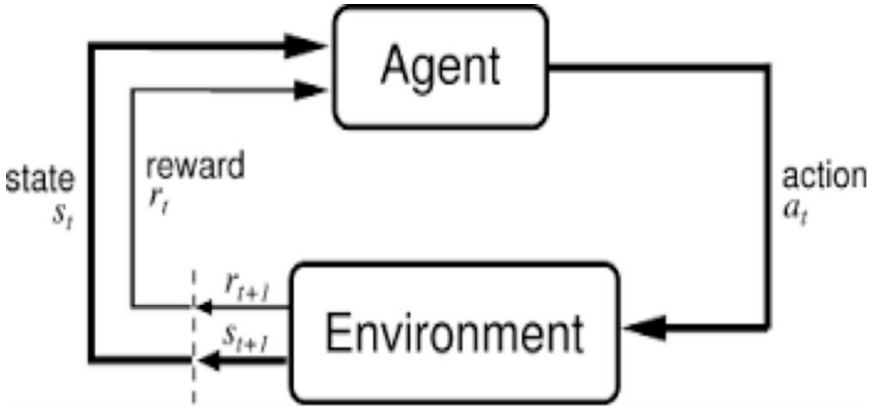
Characteristics of RL

- 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

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**.*

End of Session #1

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, …, 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

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

•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 (ΤΤ)

- 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 ***π(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 ***π(a|s)*** at time ***t*** with policy **ΤΤ**.

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 $\mathbf{v}_{\boldsymbol{\pi}}$ 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.

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

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.

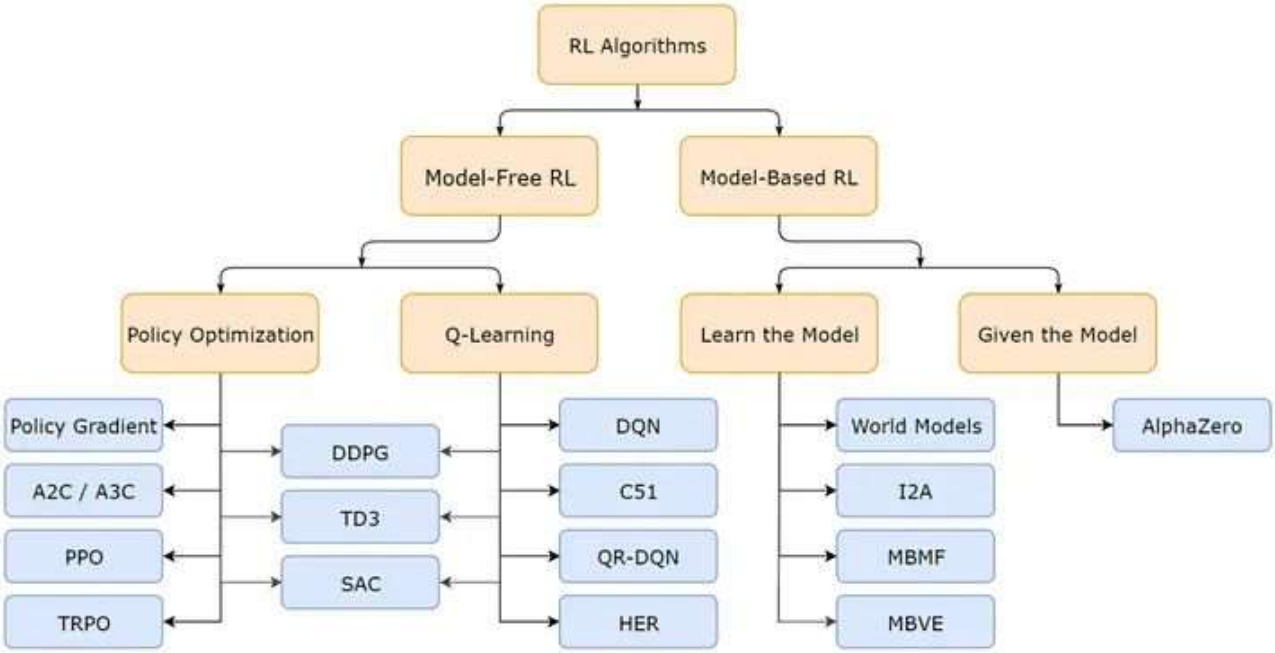
$$q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a]$$
$$= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right].$$

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



From OpenAI

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.

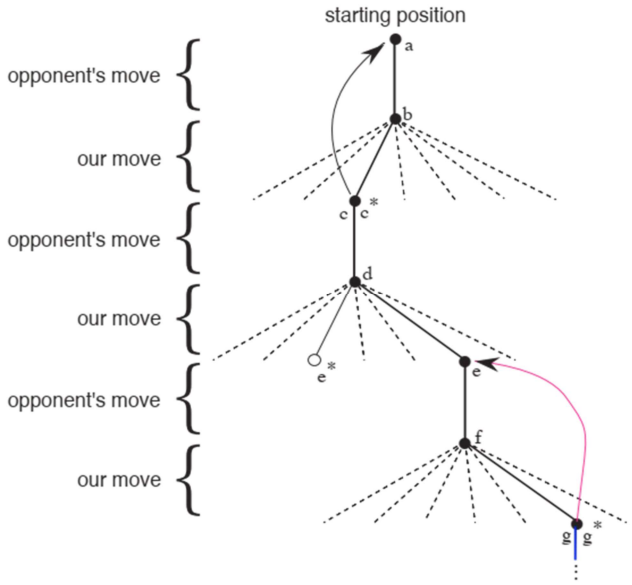
Reinforcement Learning for Tic-Tac-Toe

- 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)

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Session #2-3:
Multi-armed Bandits

Instructors :
1. Prof. S. P. Vimal (vimalsp@wilp.bits-pilani.ac.in),
2. Dr. V Chandra Sekhar (chandrasekhar.v@wilp.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]

Acknowledgements: Some of the slides were adopted *with permission* from the course CSCE-689 (Texas A&M University) by Prof. Guni Sharon

Tic-Tac-Toc

X	O	O
O	X	X
		X

Tic-Tac-Toc

States	Initial Values
$\begin{bmatrix} X & & \\ & & \\ & & \end{bmatrix}$	0.5
$\begin{bmatrix} X & O & O \\ & X & \\ & & \end{bmatrix}$	0.5
$\begin{bmatrix} X & O & O \\ & X & \\ & & X \end{bmatrix}$	1.0
$\begin{bmatrix} X & & O \\ X & & O \\ & X & O \end{bmatrix}$	0
...	...

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

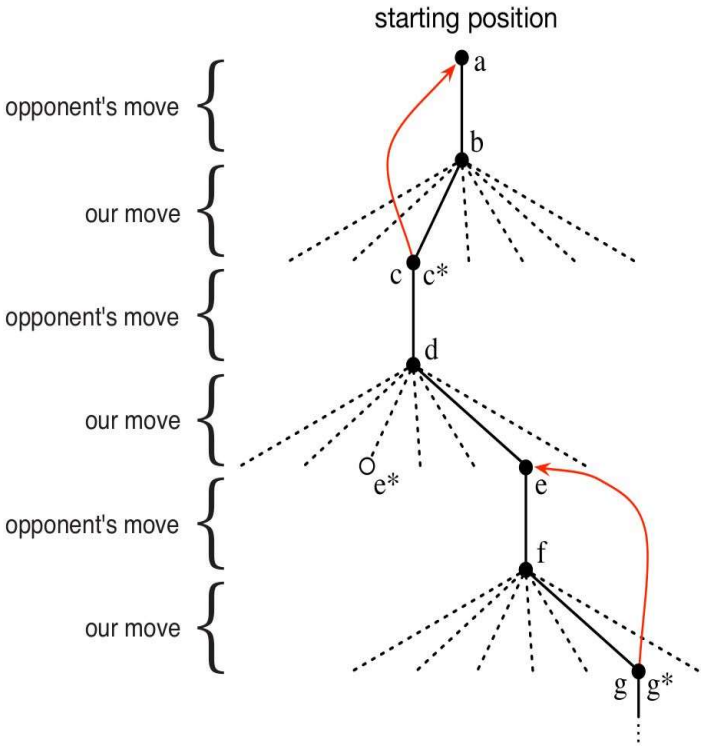
X	O	O
O	X	X
		X

Set up a table of states initial values

Tic-Tac-Toc (prev. class)

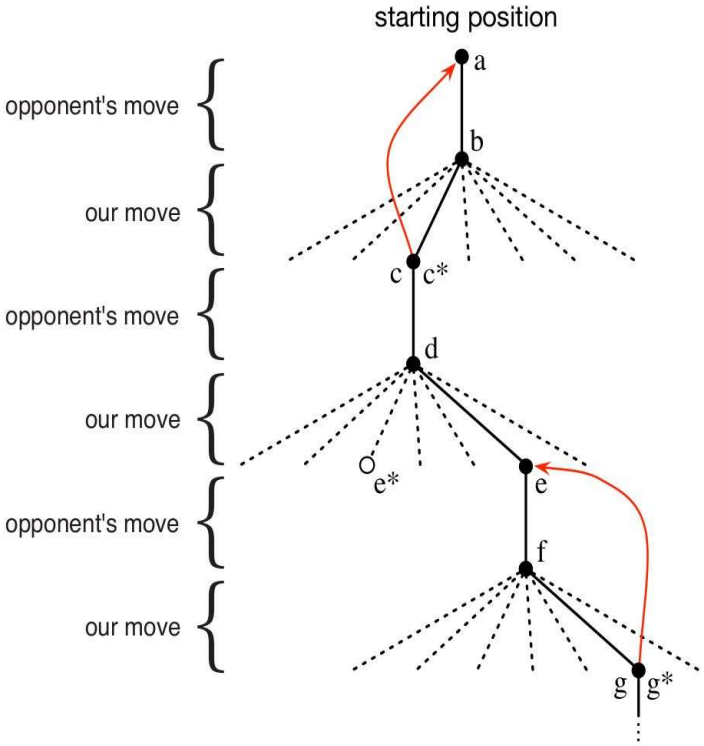
States	Initial Values
$\begin{bmatrix} X & & \\ & & \\ & & \end{bmatrix}$	0.5
$\begin{bmatrix} X & O & O \\ & X & \\ & & \end{bmatrix}$	0.5
$\begin{bmatrix} X & O & O \\ & X & \\ & & X \end{bmatrix}$	1.0
$\begin{bmatrix} X & & O \\ X & & O \\ & X & O \end{bmatrix}$	0
...	...

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



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

Tic-Tac-Toc (prev. class)



Temporal Difference Learning Rule

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

α- Step Size Parameter

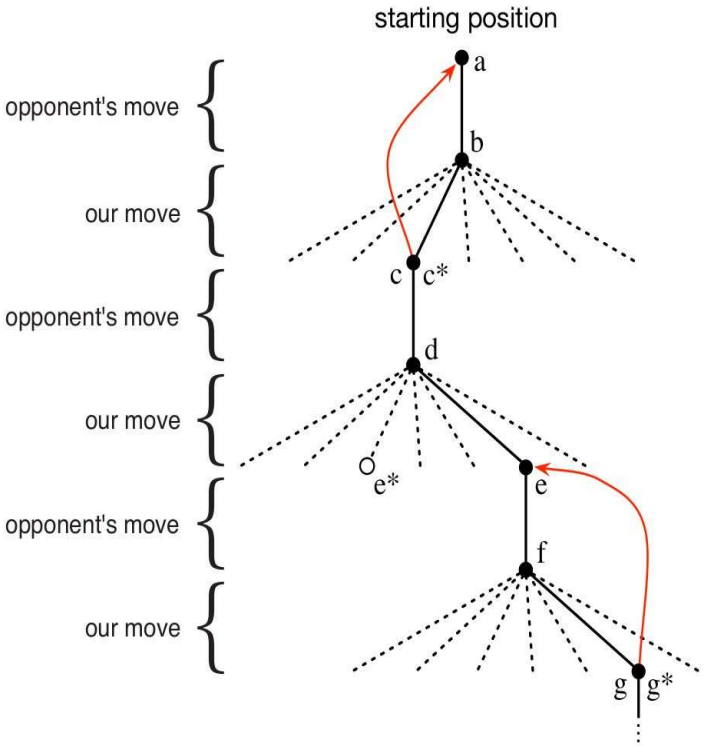
Tic-Tac-Toc (prev. class)

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

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

α- Step Size Parameter

Tic-Tac-Toc (prev. class)

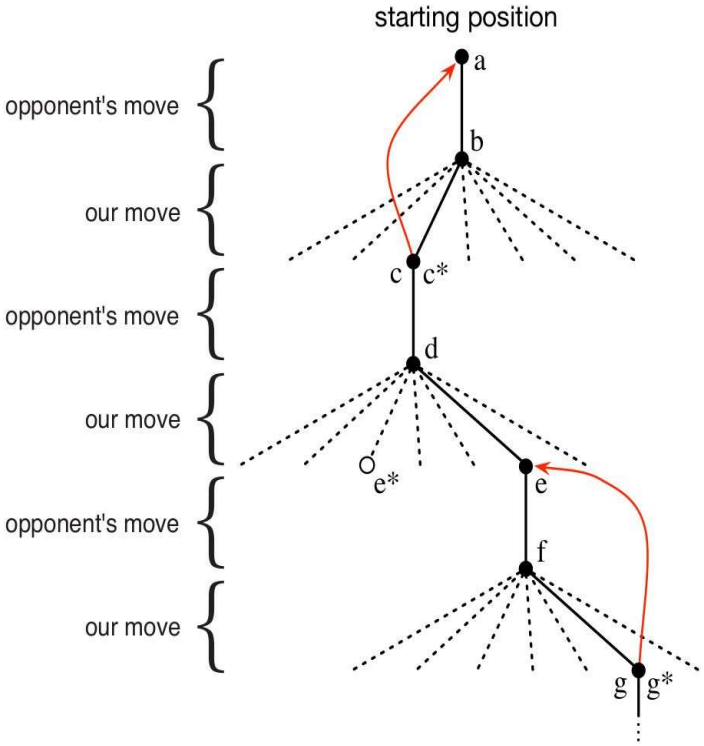
Key Takeaways:

(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

- Use the values of states most of the time (exploration) and explore rest of the time.



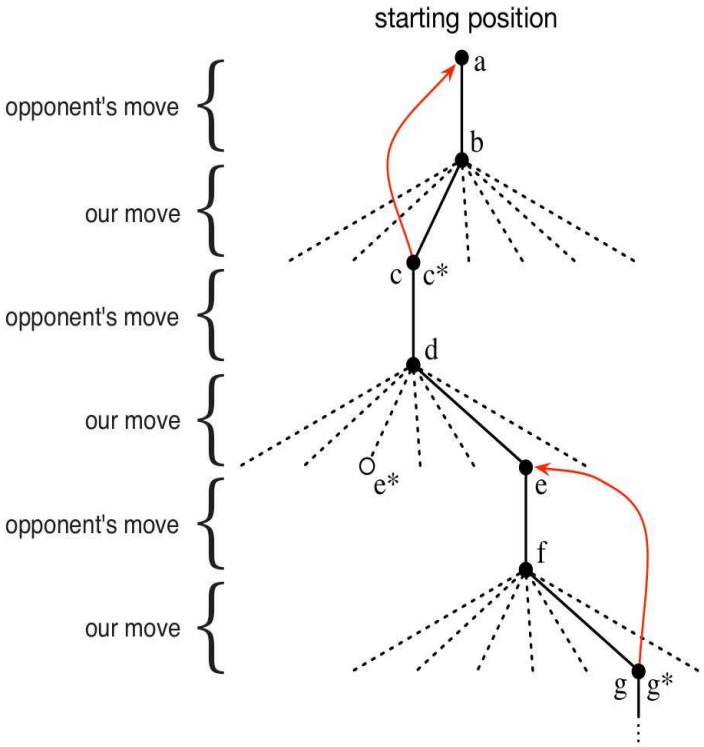
Temporal Difference Learning Rule

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

α- Step Size Parameter

Tic-Tac-Toc (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

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

α- Step Size Parameter

K-armed Bandit Problem

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 one*s?
- 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 one*s?
- 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

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

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

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

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

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

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

K-armed Bandit Problem

-1, -1, 5

$\hat{\mathbb{E}}[a] = 1$

-0.2, -0.2

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

-0.5, -0.5, -0.5

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

-2, -2

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

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

K-armed Bandit Problem

-1, -1, 5

$\widehat{\mathbb{E}}[a]=1$

-0.2, -0.2

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

-0.5, -0.5, -0.5

$\widehat{\mathbb{E}}[c]=-0.5$

-2, -2

$\widehat{\mathbb{E}}[d]=-2$

K-armed Bandit Problem

1. How to maintain the estimate of expected rewards for each action?

   Average the rewards actually received !!!

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}$$

$$= \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

1. How to use the estimate in selecting the right action?

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

K-armed Bandit Problem

2. 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 !!!

3. 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}

$N_{11}(d)=2$

$q_*(d)=\$1$

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

K-armed Bandit Problem

Greedy Action

{-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}

$N_{11}(d)=2$

$q_*(d)=\$1$

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

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}

$N_{11}(d)=2$

$q_*(d)=\$1$

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

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 Qt (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 ε = 0.5, what is the probability that *the greedy action* is selected?

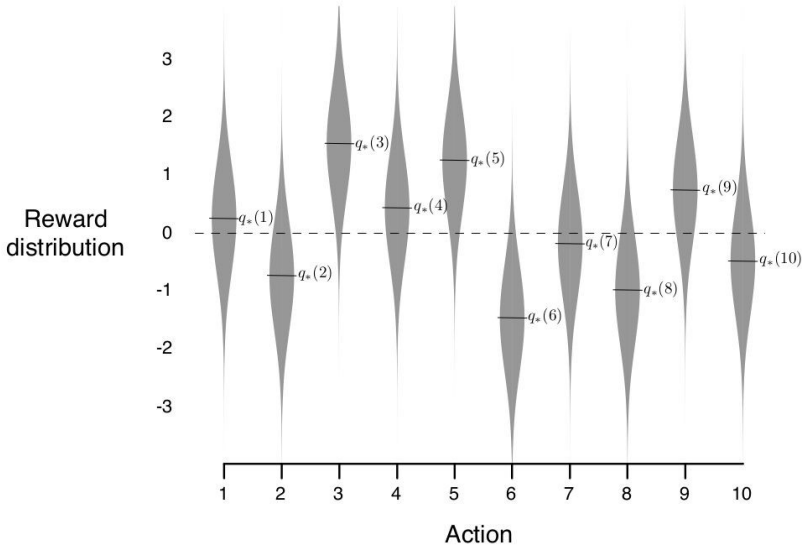
Ex-1: In ε-greedy action selection, for the case of two actions and ε = 0.5, what is the probability that *the greedy action* is selected?

p (greedy action)
= p (greedy action AND greedy selection) + p (greedy action AND random selection)
= p (greedy action | greedy selection) p (greedy selection)
+ p (greedy action | random selection) p (random selection)
= p (greedy action | greedy selection) (1-ε) + p (greedy action | random selection) (ε)
= p (greedy action | greedy selection) (0.5) + p (greedy action | random selection) (0.5)
= (1) (0.5) + (0.5) (0.5)
= 0.5 + 0.25
= 0.75

10-armed Testbed

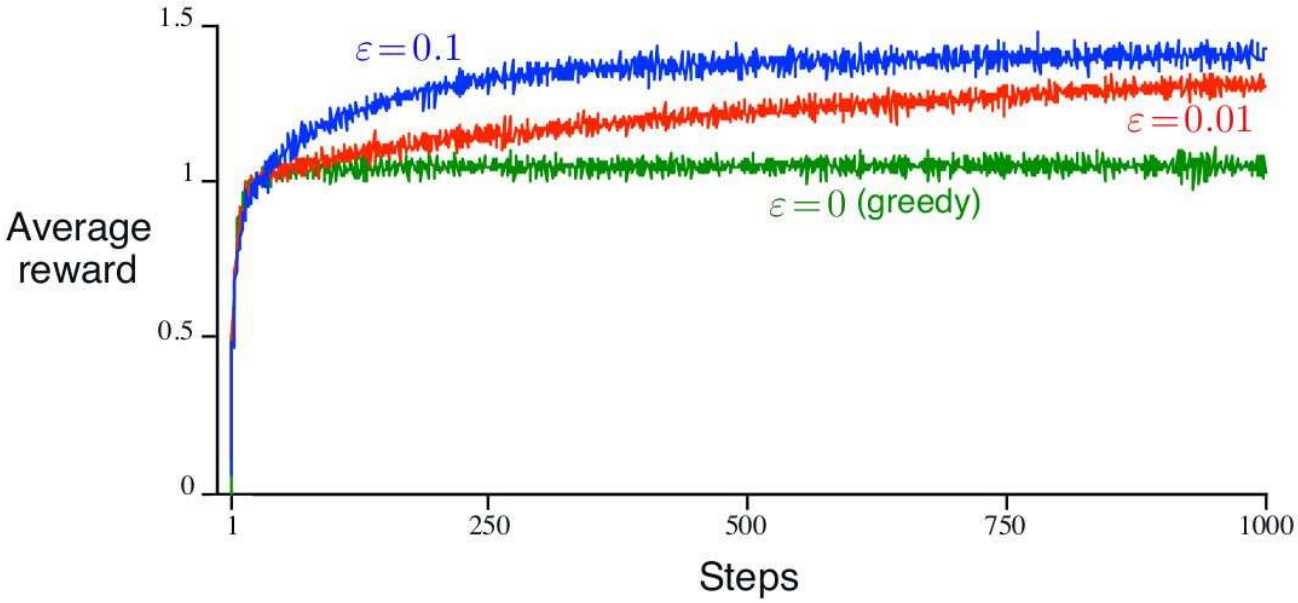
Example:

- 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

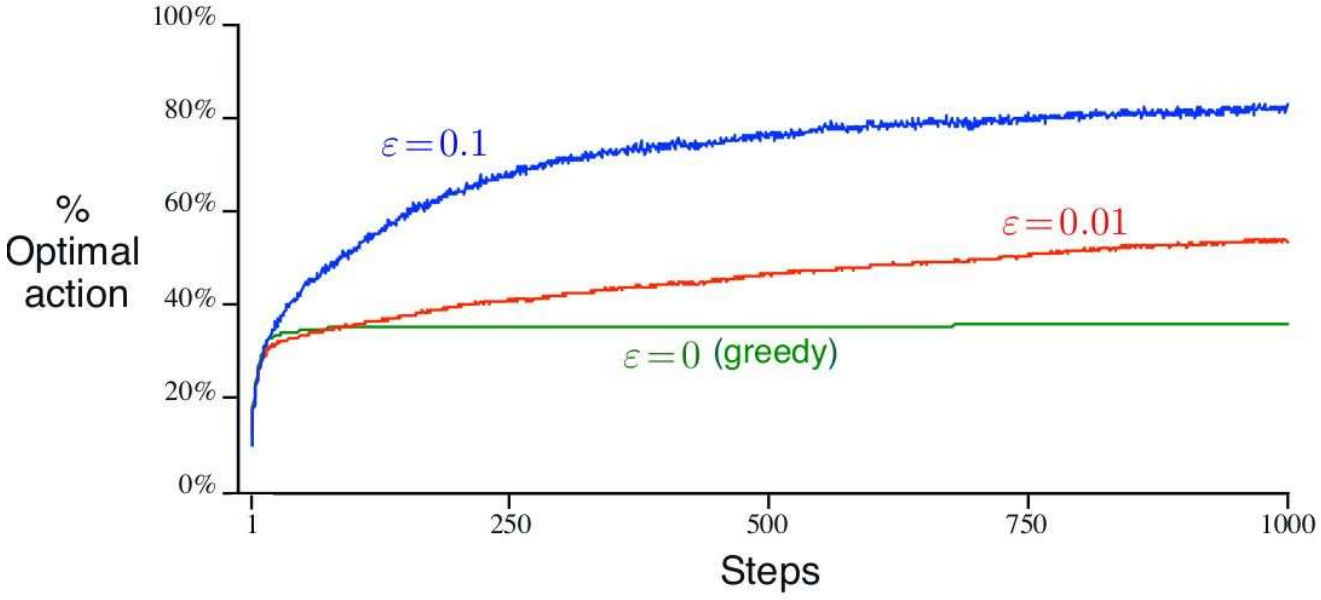


An example bandit problem from the 10-armed testbed

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 Q1 (a) = 0, for all a.

Suppose the initial sequence of actions and rewards is A1 = 1, R1 = 1, A2 = 2, R2 = 1, A3 = 2, R3 = 2, A4 = 2, R4 = 2, A5 = 3, R5 = 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;

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

- Given Qn and the nth reward, Rn , the new average of all n rewards can be computed as follows

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

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?

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

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \Big[Target - OldEstimate\Big]$$

Bandit Algorithm with Incremental Update/ ε-greedy selection

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

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

$R \leftarrow bandit(A)$

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

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

Non-stationary Problem

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

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

Non-stationary Problem

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

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

Exponential recency-weighted average

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

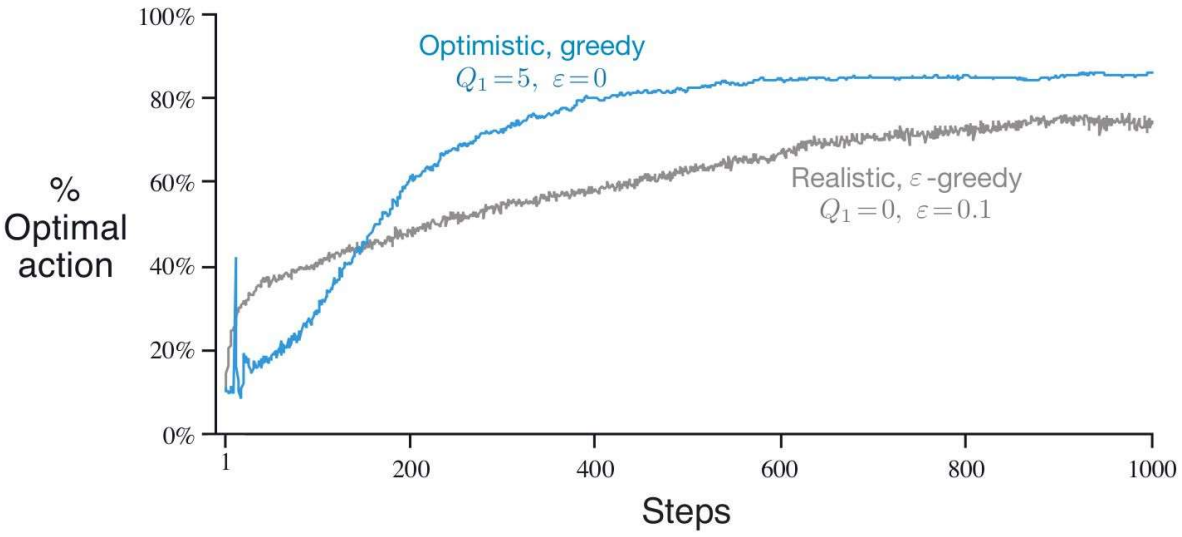
Optimistic Initial Values

- 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, α = 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

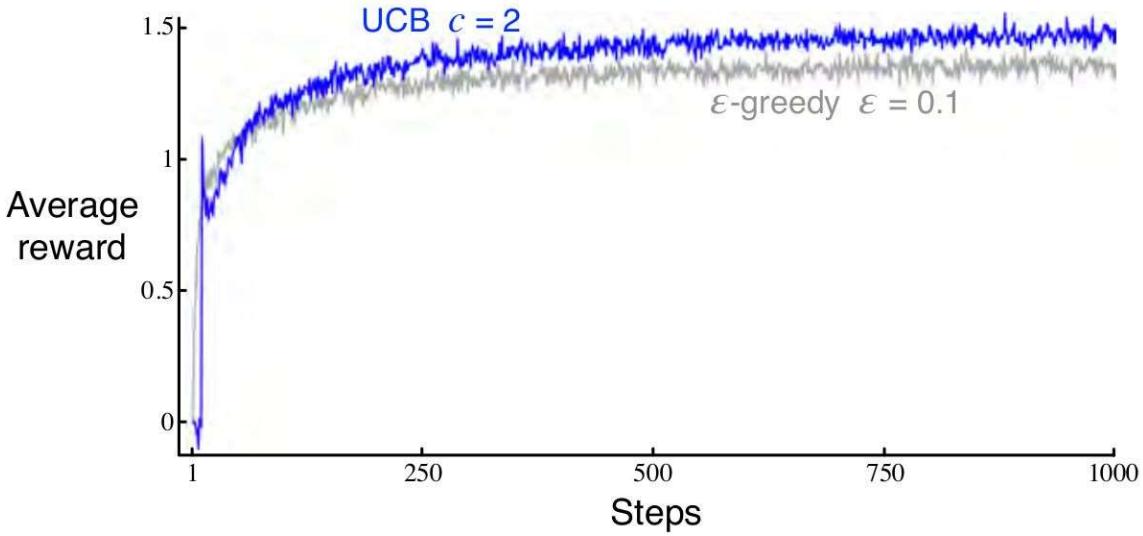
Action Value at time t for a

Confidence Level

Measure of Uncertainty

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

Upper-Confidence-Bound Action Selection



UCB often performs well, as shown here, but is more difficult than "-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 chose

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?
  - $A_t = \underset{a}{\text{argmax}}[H_t(a)]$
    - Hardmax results in no exploration -- deterministic action selection

Softmax!

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: $A_t = \underset{a}{\text{argmax}}[H_t(a_1), H_t(a_2), H_t(a_3)]$
  - 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

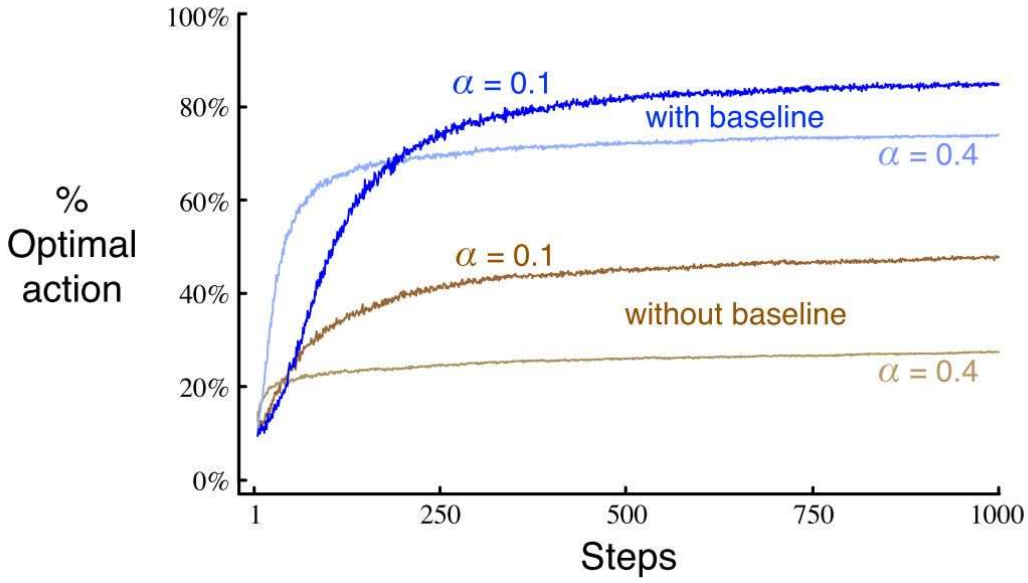
- $H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \overline{R_t})\frac{\partial \Pr(A_t)}{\partial H(A_t)}$?
- $\forall a \neq A_t, H_{t+1}(a) = H_t(a) + \alpha(R_t - \overline{R_t})\frac{\partial \Pr(A_t)}{\partial H(a)}$
- Update the preferences based on observed reward and a baseline reward ($\overline{R_t}$)
- 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 Rt ,
Update the action preferences :

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

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

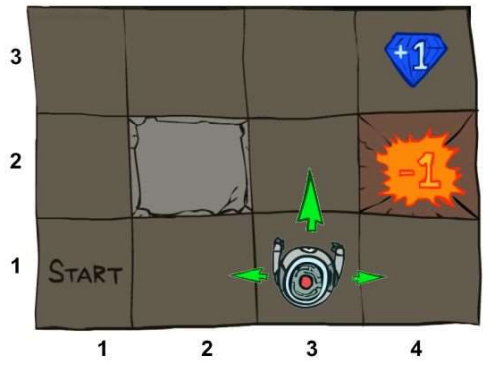


What did we learn?

- **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 !

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

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

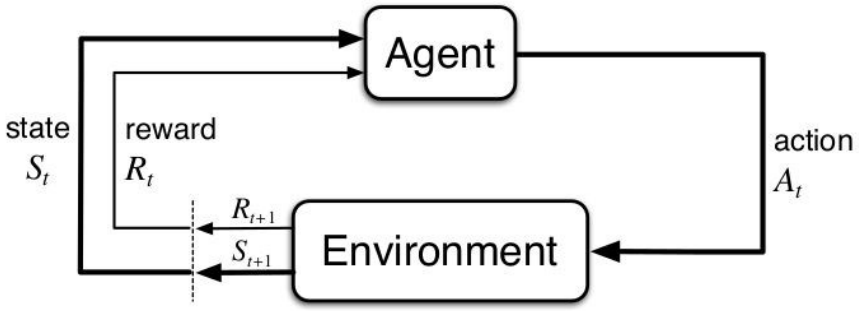
Announcement !!!

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}

Acknowledgements: Some of the slides were adopted *with permission* from the course CSCE-689 (Texas A&M University) by Prof. Guni Sharon

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



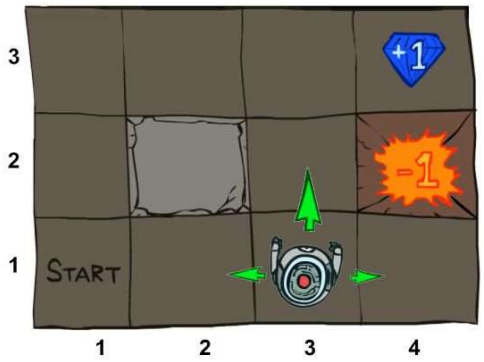
Note:

- Interaction occurs in discrete time steps

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

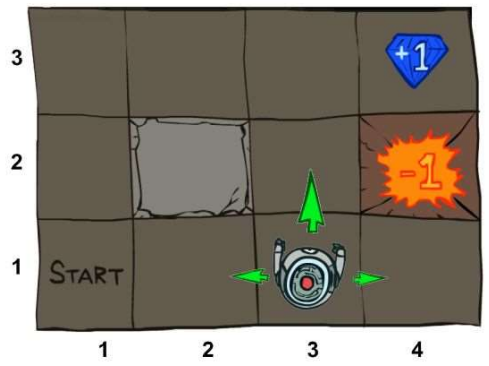
Grid World Example

- 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

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

State-transition probabilities

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

Expected rewards for state–action–next-state triples

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

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).*

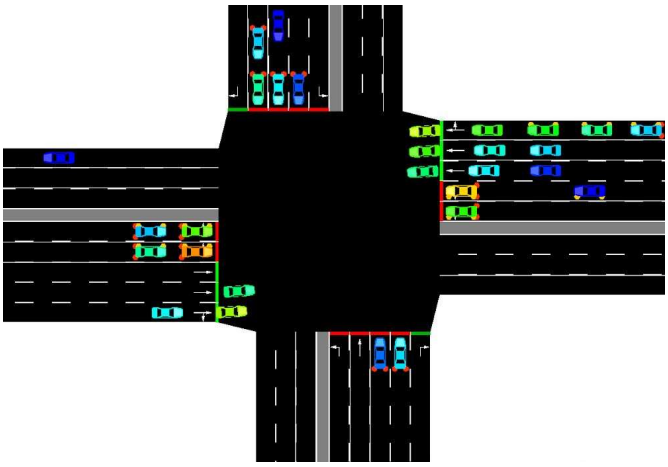
MDP Formalization : Video Games

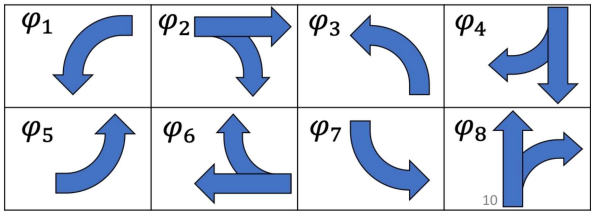
- *State:*
  - *raw pixels*
- *Actions:*
  - *game controls*
- *Reward:*
  - *change in score*
- *State-transition probabilities:*
  - *defined by stochasticity in game evolution*

Ref: Playing Atari with deep reinforcement learning", Mnih et al., 2013

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

MDP Formalization : Recycling Robot (Detailed Ex.)

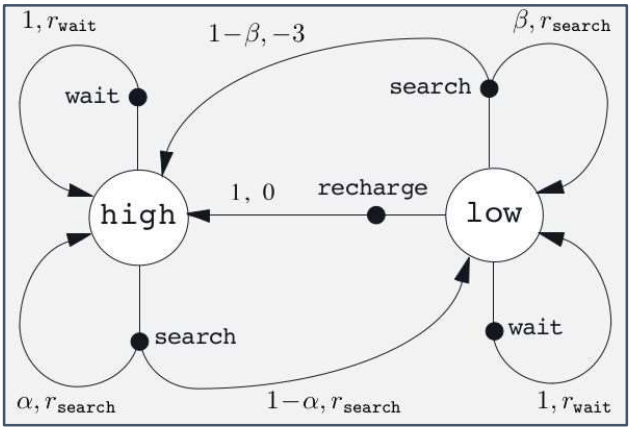
- *State-transition probabilities (contd…):*

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
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'	$p(s' \mid s, a)$	$r(s, a, s')$
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 (Gt) 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]*

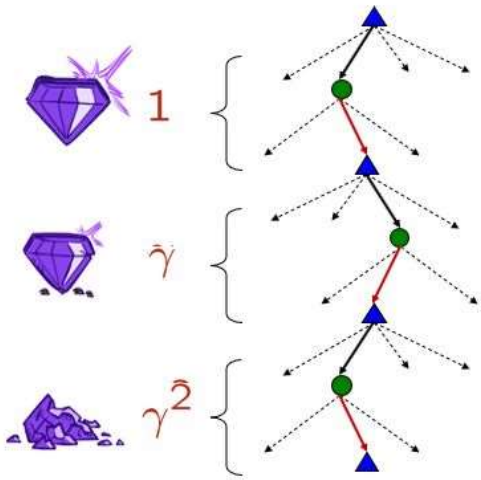
Returns & Episodes

- *Generally T = ∞*
  - *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*

  - *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*

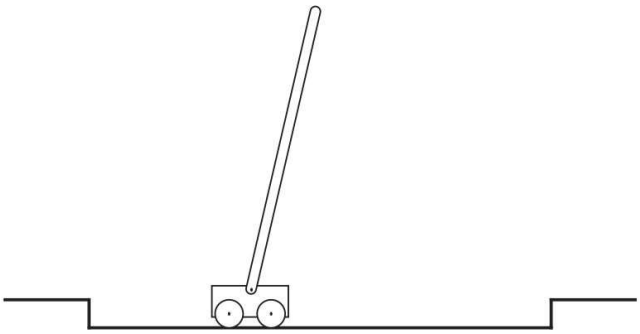
$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

- *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*

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

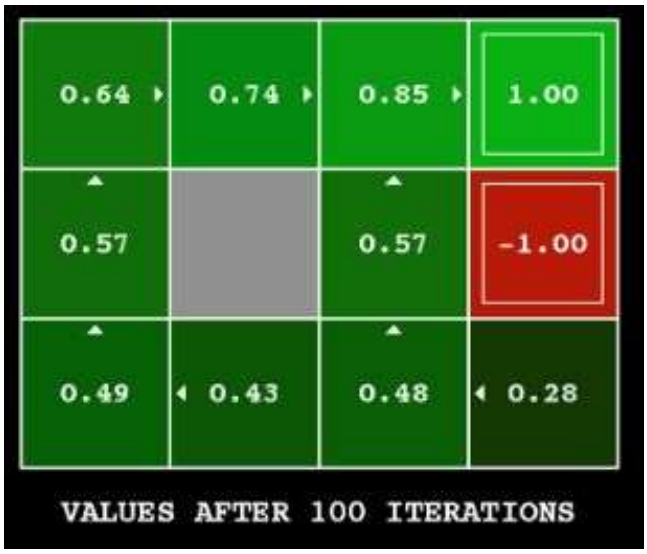
Returns & Episodes

- ➔ *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

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

Action-value function for policy π

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

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

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

Bellman Equation for V_π

May skip to the next slide !

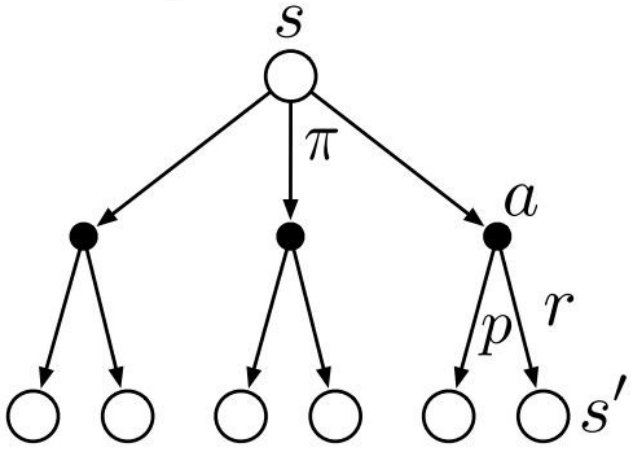
- Dynamic programming equation associated with discrete-time optimization problems
  - Expressing V_π recursively i.e. relating $V_\pi(s)$ to $V_\pi(s')$ for all $s' \in$ succ(s)

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

Bellman Equation for V_π

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

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

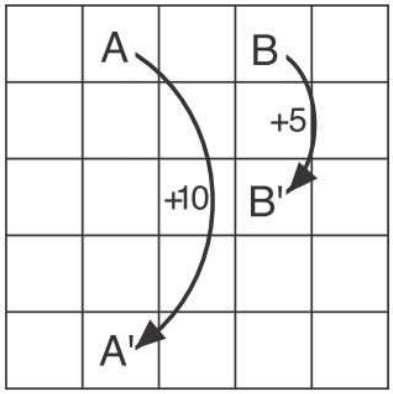


Backup Diagram

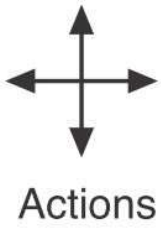
Understanding $V_\pi(s)$ with Gridworld

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

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

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

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 → 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 $q_*(s,a) \doteq \max q_\pi(s,a)$

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

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*

Bellman optimality equation for V_*

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

Optimal Policies and Optimal Value Functions

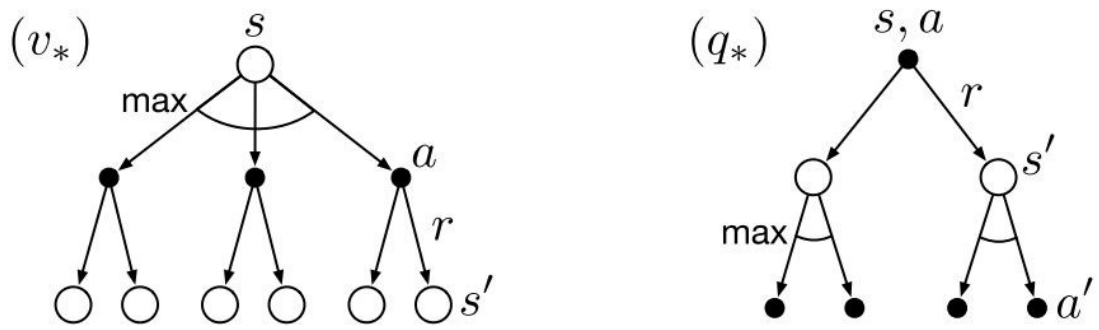
Bellman optimality equation - expresses that *the value of a state under an optimal policy must equal the expected return for the best action from that state*

Bellman optimality equation for q_*

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

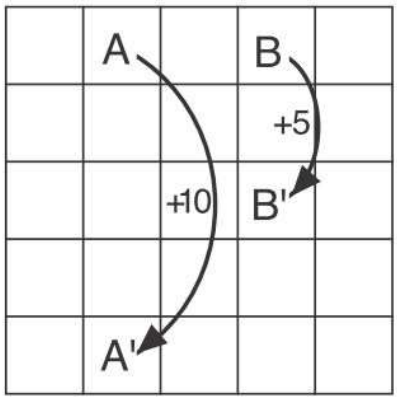
Optimal Policies and Optimal Value Functions

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*

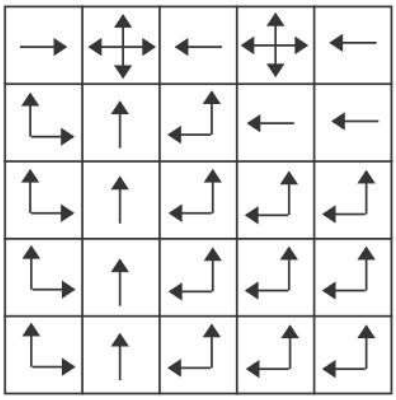
Optimal solutions to the gridworld example



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_*

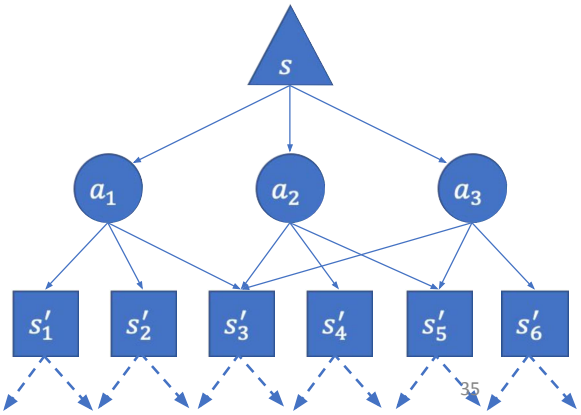
Backup diagrams for v* and q*

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^*
- $\pi^*(s) =?$
- $= \underset{a}{\text{argmax}}[\sum_{s'} P(s'|s,a)R(s,a,s')]$
  - Must also optimize over the future (next steps)
- $= \underset{a}{\text{argmax}}[\sum_{s'} P(s'|s,a)\,(R(s,a,s') + \underbrace{\mathbb{E}_{\pi^*}[G|s']}_{v^*(s')})]$



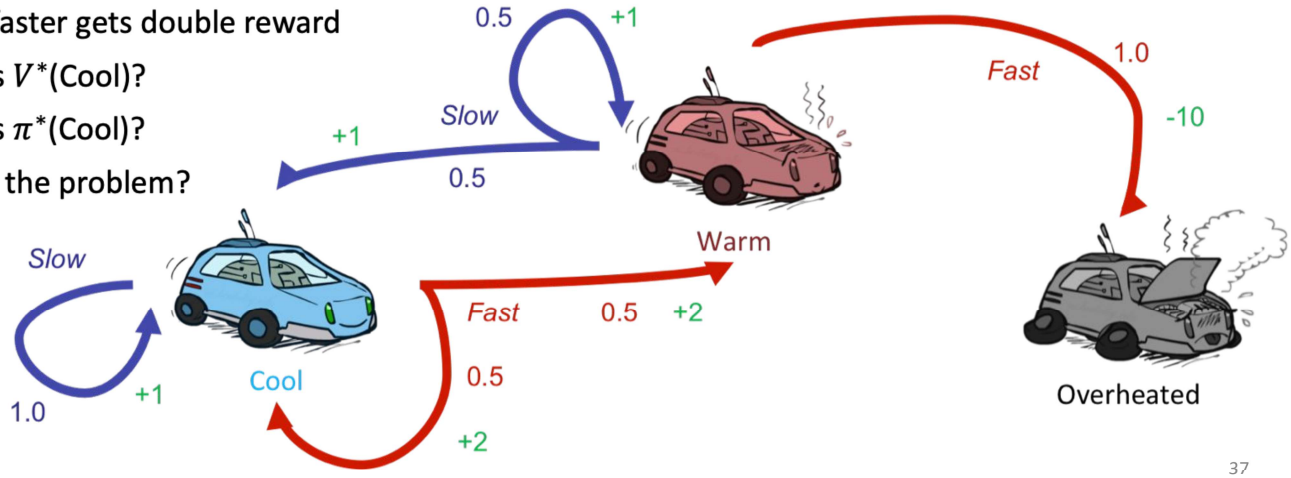
Notation

- 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

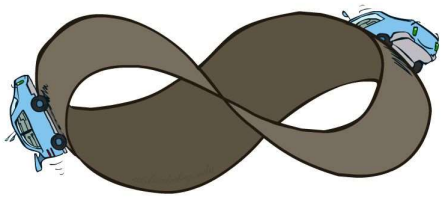
$$\max_a \left[\sum_{s'} P(s'|s,a) \left(R(s,a,s') + v^*(s') \right) \right]$$

- 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^*(Cool)?
- What is π^*(Cool)?
- What's the problem?



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 $0 < \gamma < 1$



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

  - 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?
  - $\tau_1 = +1, +1, +1, +1, +1, +1 \ldots$
  - $\tau_2 = +2, +2, +2, +2, +2, +2 \ldots$

$$\sum_{t=0}^{\infty} 1 = \sum_{t=0}^{\infty} 2 = \infty$$

- Let's decay future rewards exponentially by a factor, $0 \leq \gamma < 1$

$$\sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$$ **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

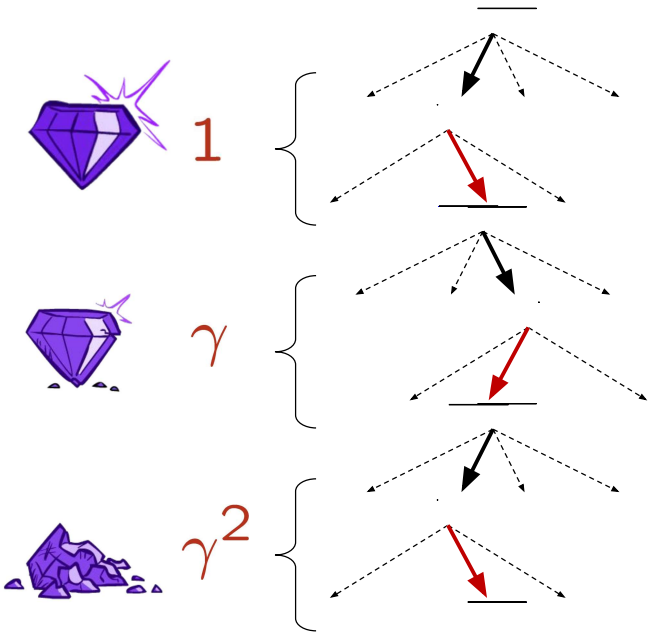
Worth Next Step

γ^2

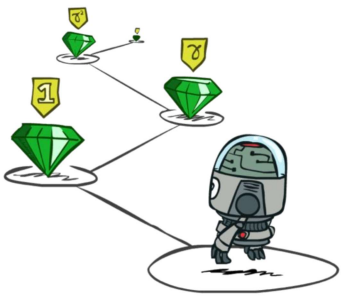
Worth In Two Steps

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 γ = 1, what is the optimal policy?

10				1

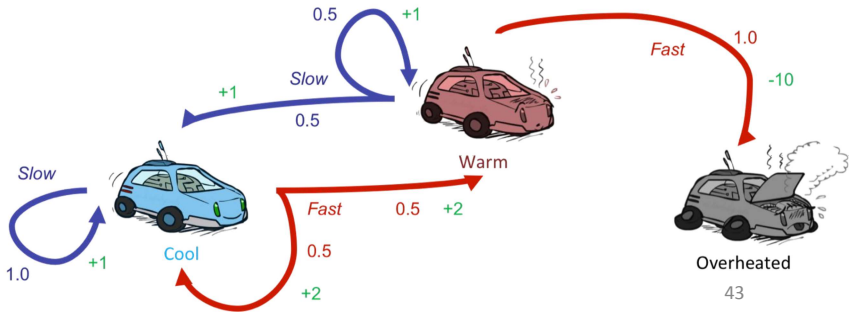
- Quiz 2: For γ = 0.1, what is the optimal policy?

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



Value iteration

Value iteration

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

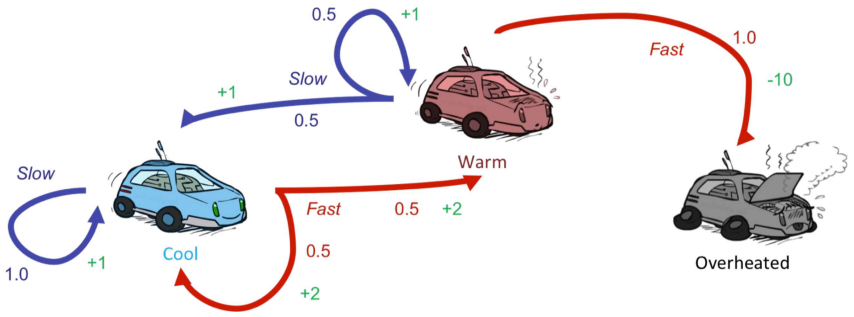
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

Value Iteration

V_0	0	0	0
V_1	2	1	0
V_2	3.35	2.35	0

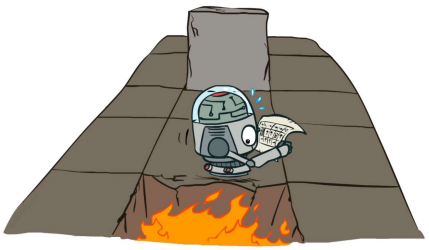
Check this computation on paper.



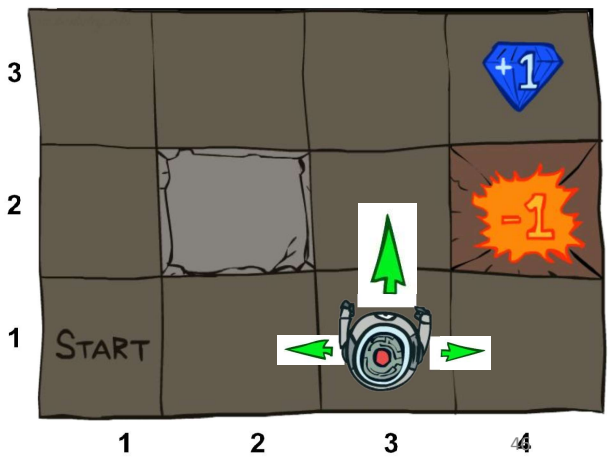
$\gamma = 0.9$

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

Example: Grid World

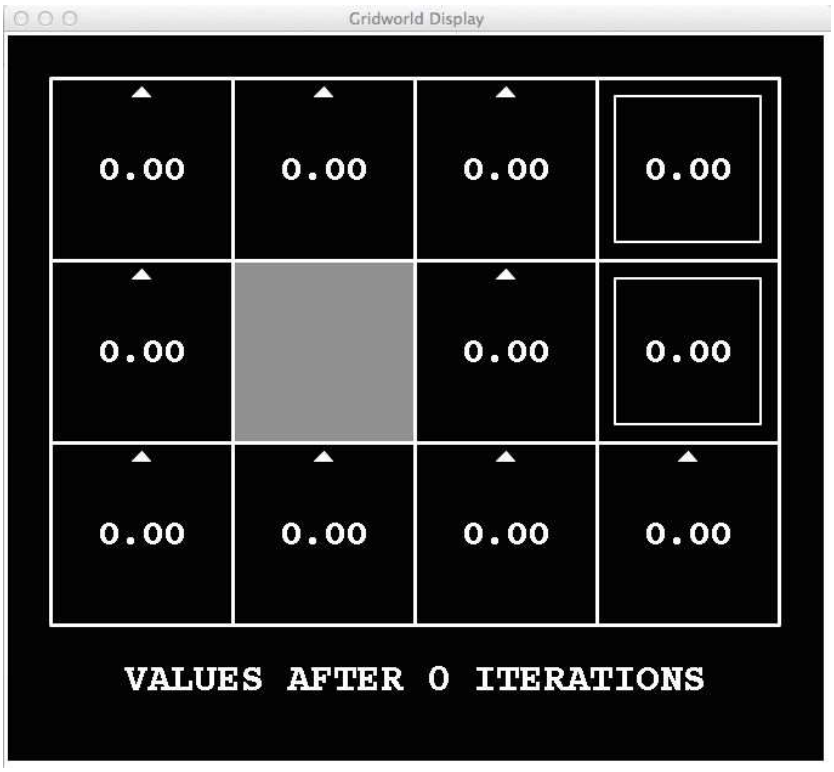


- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the 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



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

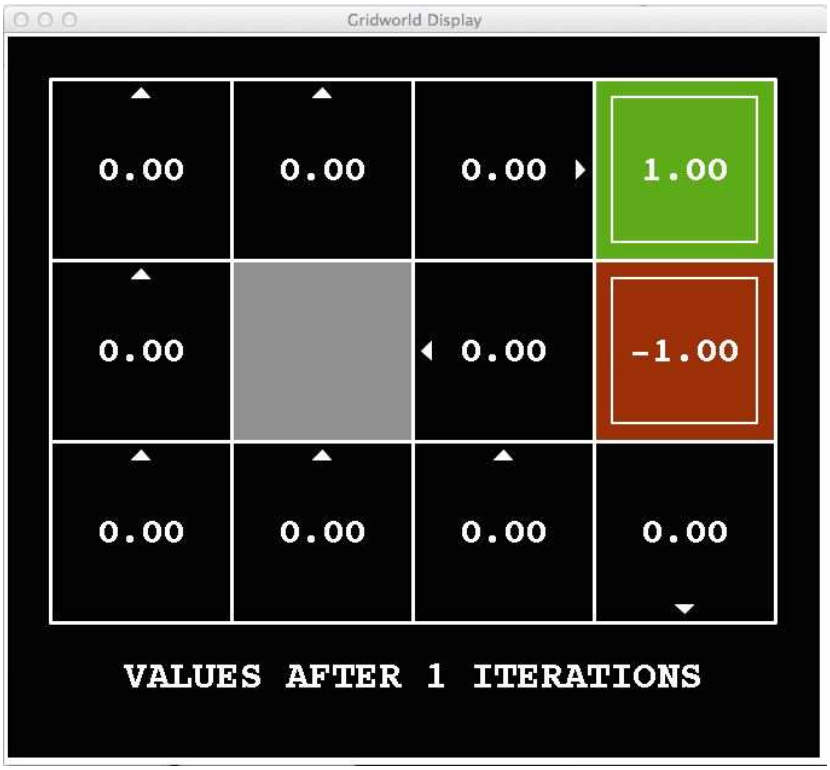
k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

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

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

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

k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

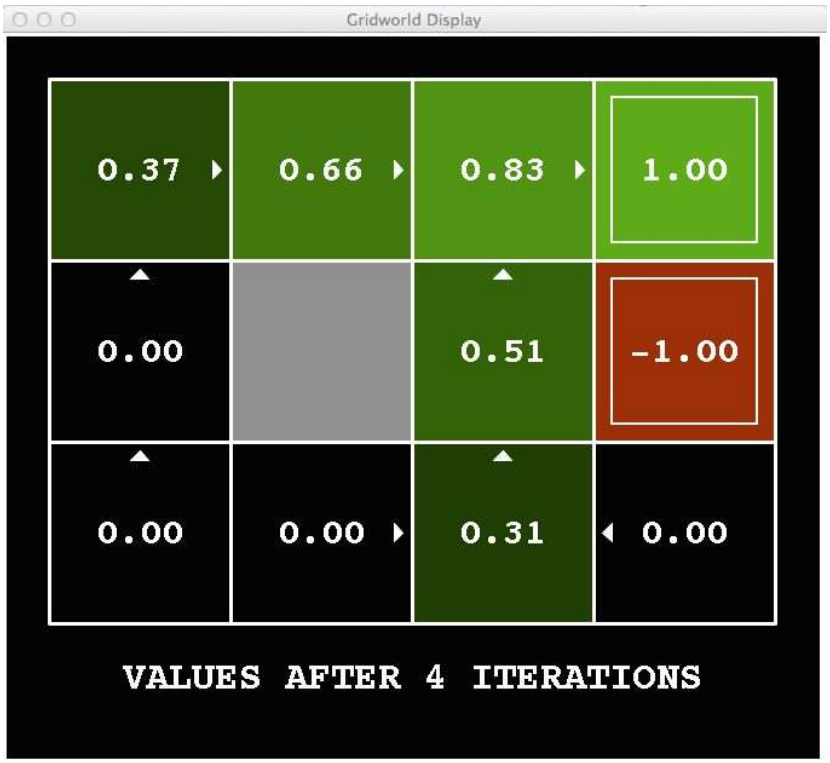
$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a)\left[r+\gamma v_k(s')\right]$$

k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

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

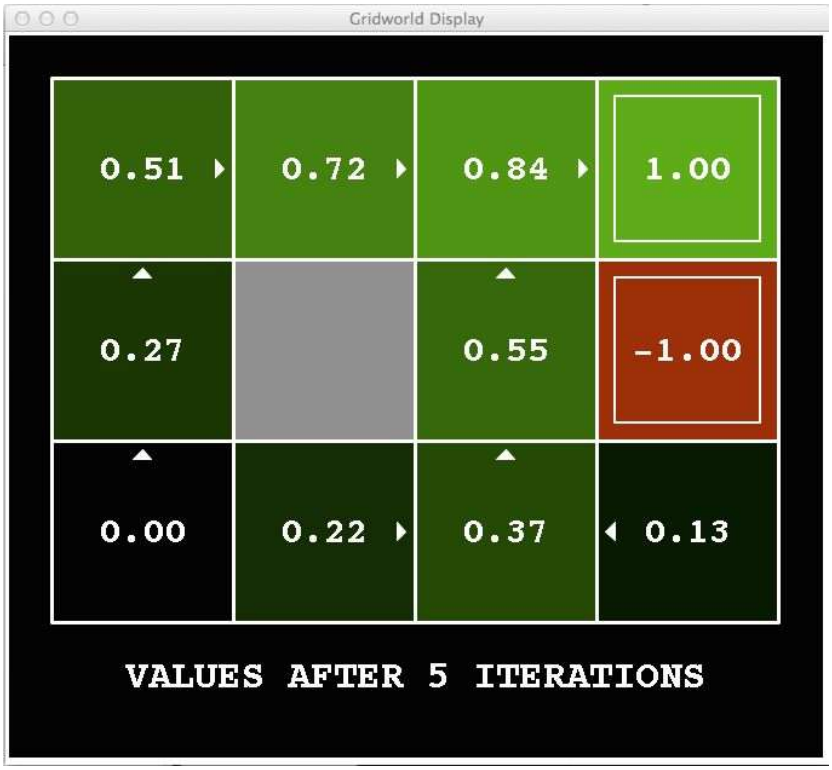
k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

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

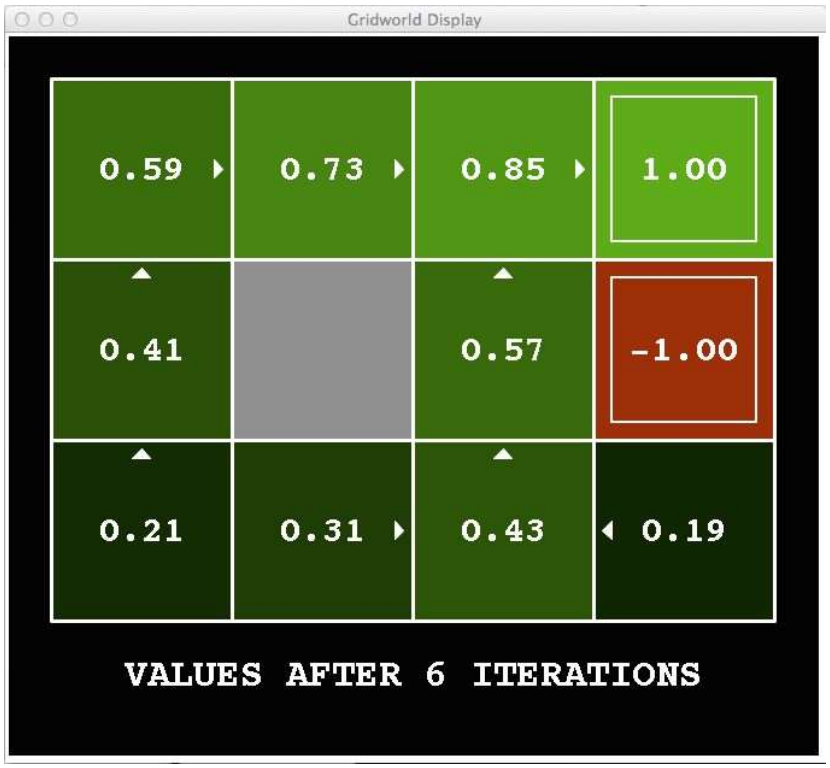
k=5



Noise = 0.2
Discount = 0.9
Living reward = 0

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

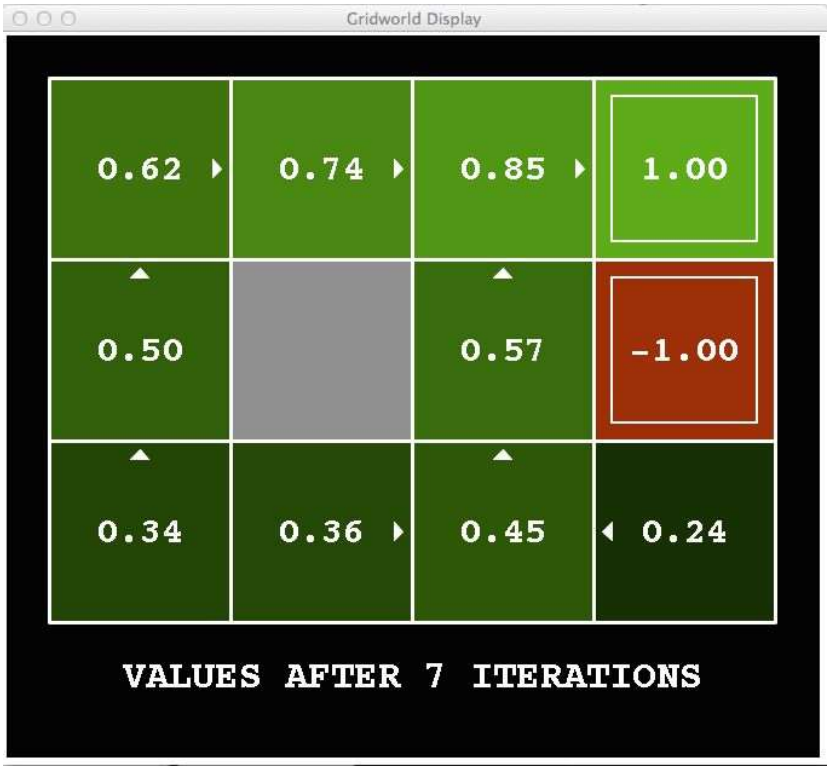
k=6



Noise = 0.2
Discount = 0.9
Living reward = 0

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

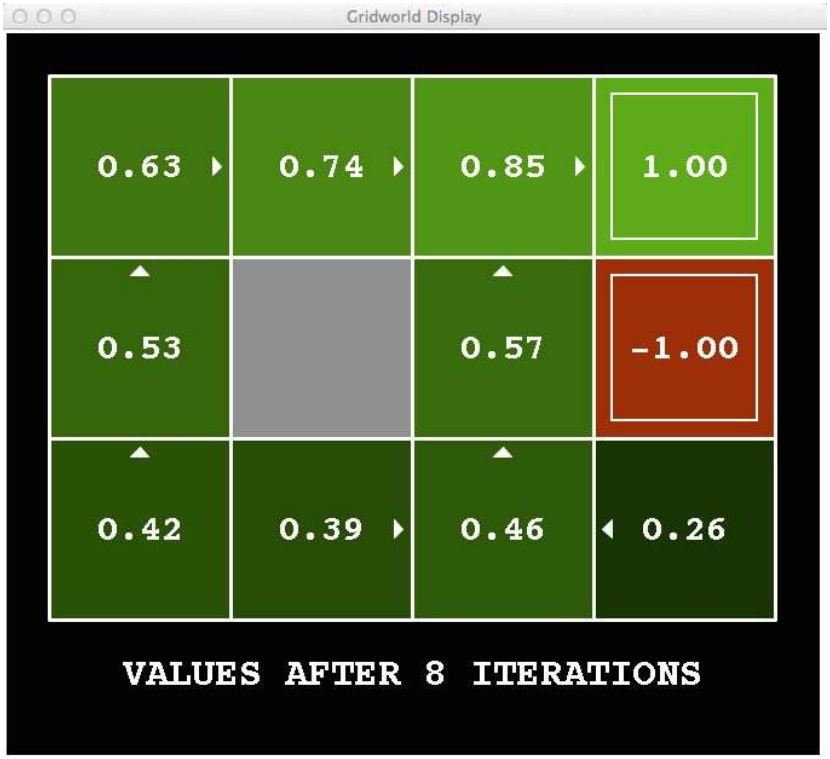
k=7



Noise = 0.2
Discount = 0.9
Living reward = 0

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

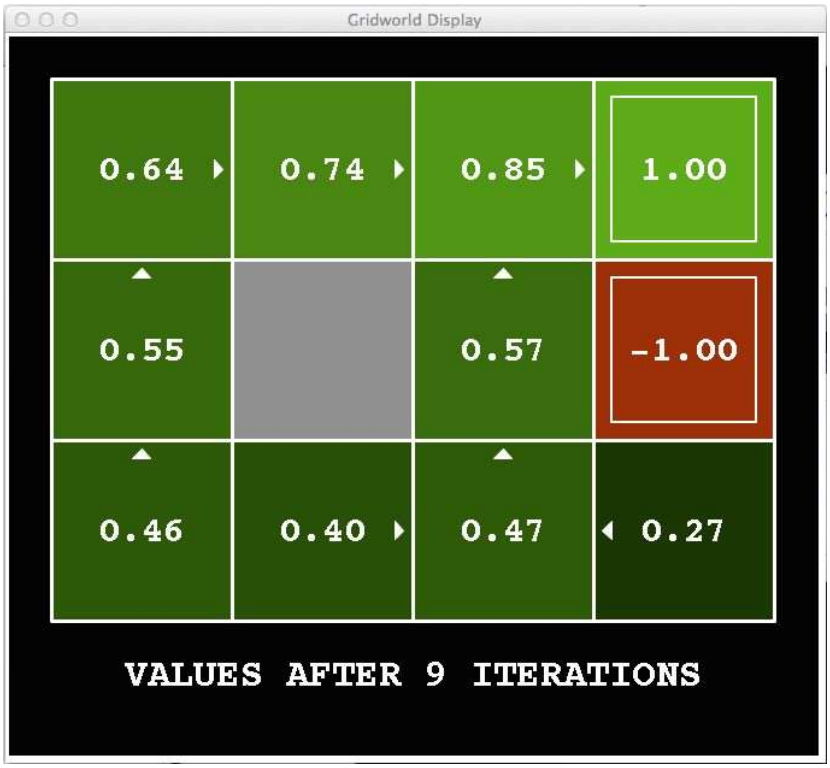
k=8



Noise = 0.2
Discount = 0.9
Living reward = 0

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

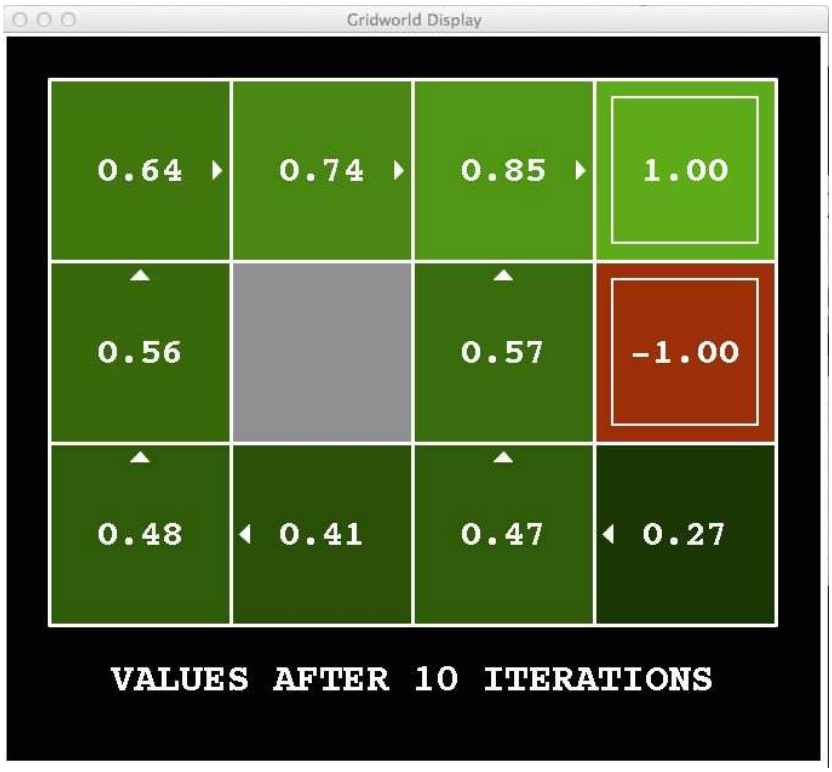
k=9



Noise = 0.2
Discount = 0.9
Living reward = 0

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

k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

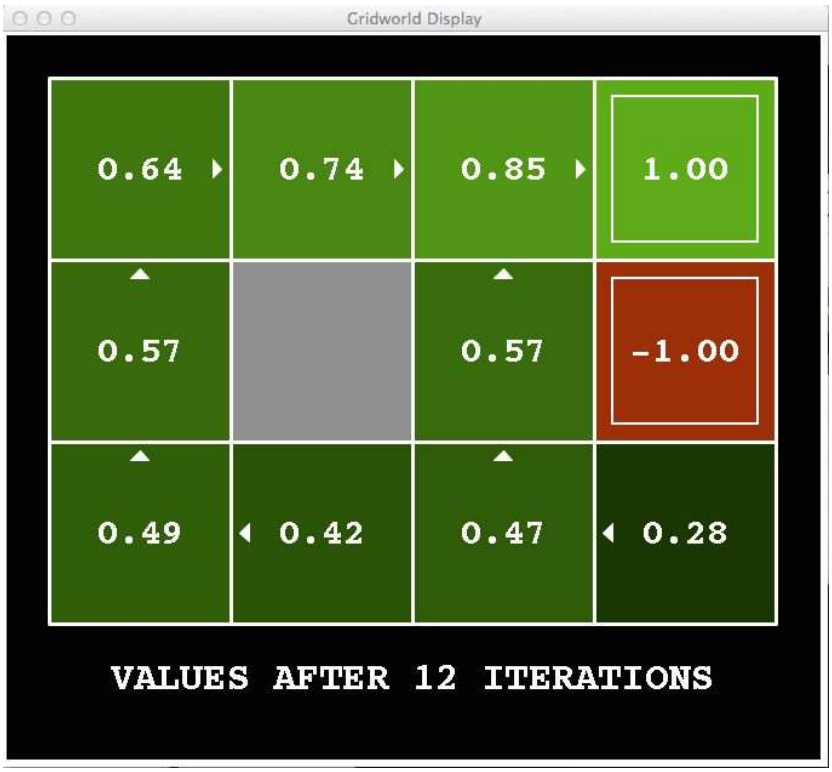
$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a)\left[r+\gamma v_k(s')\right]$$

k=11

Noise = 0.2
Discount = 0.9
Living reward = 0

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

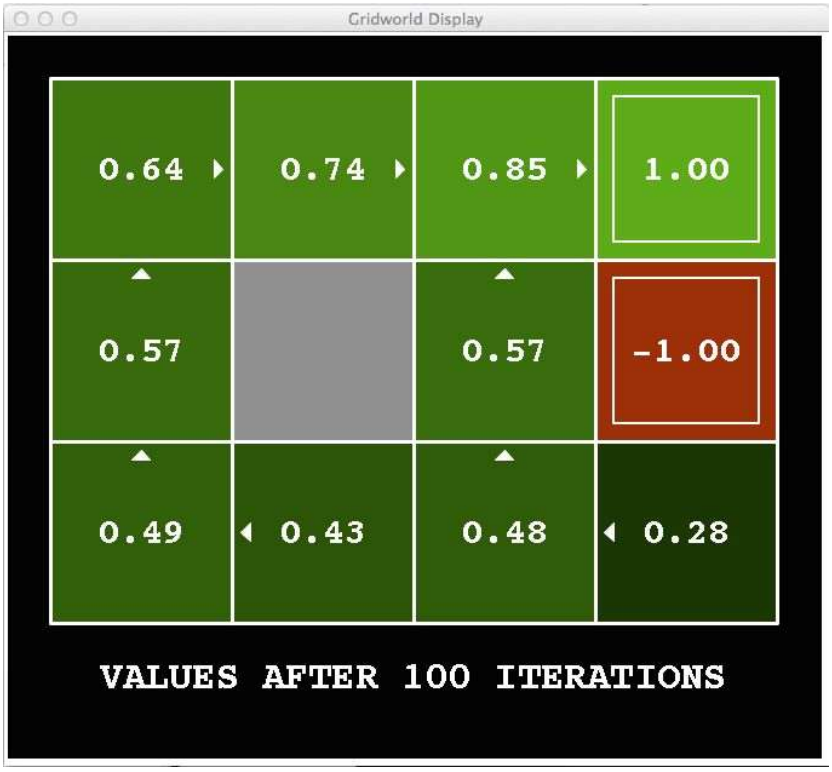
k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

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

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

- Value iteration repeats the Bellman updates:
- $V_{k+1}(s) \leftarrow \max_a \left[R(s,a) + \sum_{s'} P(s'|s,a)\ \gamma V_k(s')\right]$

$$= \max_a \left[\sum_{s'} P(s'|s,a)\ (R(s,a,s') + \gamma V_k(s'))\right]$$

- 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

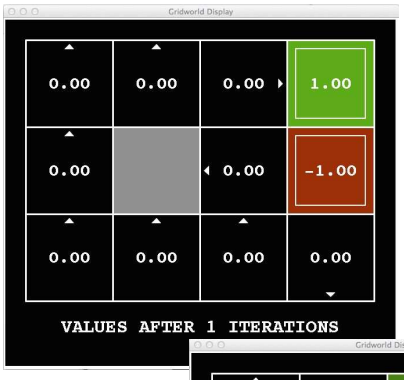
Solutions (briefly, more later…)

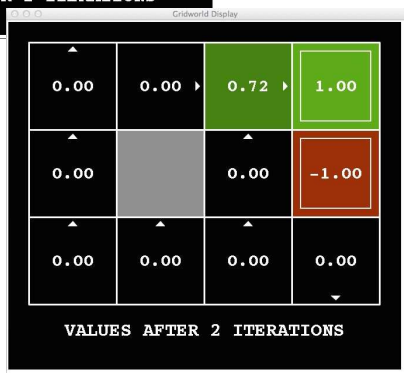
- Issue 1: It's slow – $O(S^2A)$ per iteration
  - Asynchronous value iteration
- Issue 2: A policy cannot be easily extracted
  - Learn q (action) values
- Issue 3: The policy often converges long before the values
  - Policy-based methods
- Issue 4: requires knowing the model and the reward function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

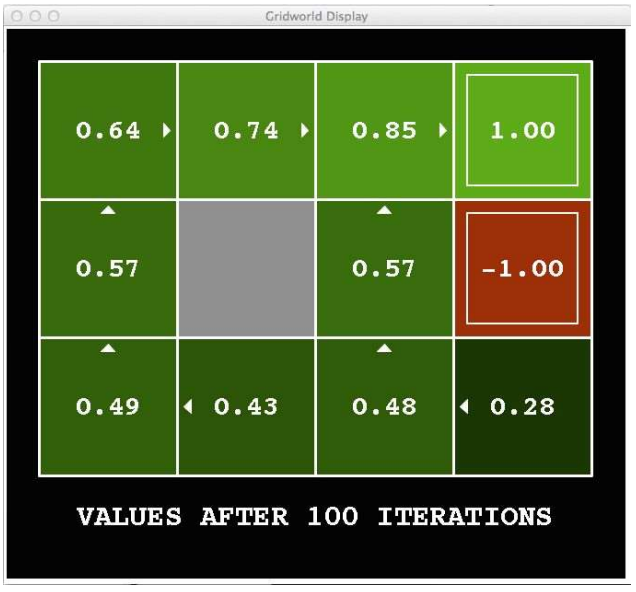
For the home assignment set:

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

- 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

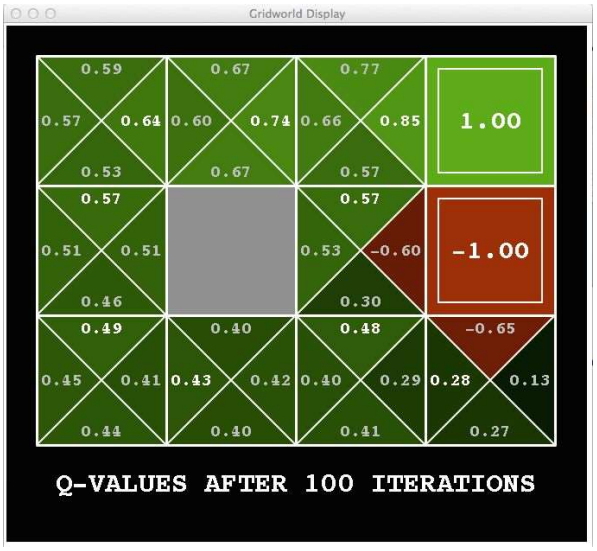
Issue 2: A policy cannot be easily extracted

- Given state values, what is the appropriate policy?
  - $\pi(s) \leftarrow \underset{a}{\text{argmax}}\left[R(s,a) + \sum_{s'} P(s'|s,a)\ \gamma V_k(s')\right]$
  - 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^*
- $\pi^*(s) \leftarrow \underset{a}{\text{argmax}}[Q^*(s, a)]$
- Can we learn Q values with dynamic programming?
  - Yes, similar to value iteration



Q-learning as value iteration

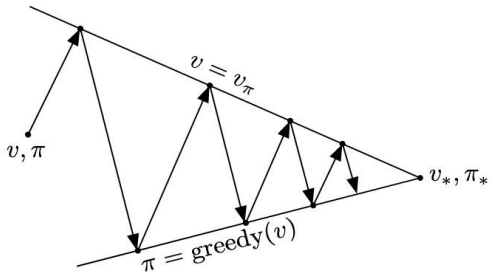
- $V^*(s) \coloneqq \max_a[\sum_{s'} P(s'|s, a)\,(R(s, a, s') + \gamma V^*(s'))]$
- $V^*(s) \coloneqq \max_a[Q^*(s, a)]$
- $Q^*(s, a) \coloneqq \sum_{s'} P(s'|s, a)\,(R(s, a, s') + \gamma V^*(s'))$
- $Q^*(s, a) = \sum_{s'} P(s'|s, a)\,(R(s, a, s') + \gamma \max_a[Q^*(s', a)])$
- Solve iteratively
  -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

Issue 3: The policy often converges long before the values

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

Policy Iteration

1. Compute V_π: calculate state value for some fixed policy (not necessarily the optimal values, $V_\pi \neq V^*$)
2. Update π: update policy using one-step look-ahead with the resulting (non optimal) values
3. Repeat until policy converges
   (optimal values and policy)



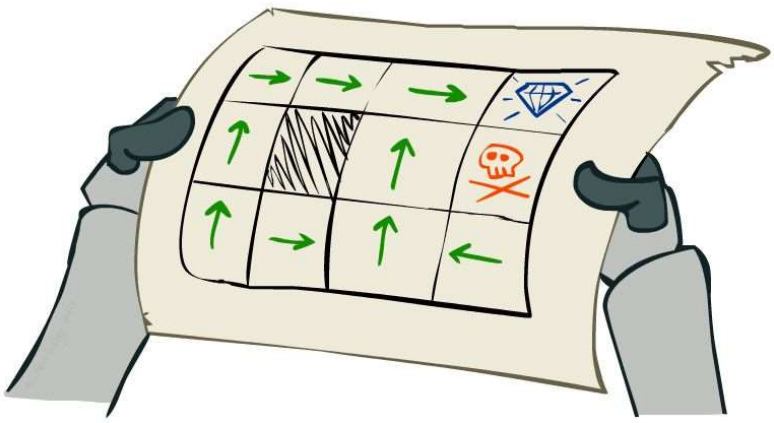
- Guaranteed converges to π^*
  - $\forall s, V_{k>0}(s) \leq V_{k+1}(s)$ i.e., π_i improves monotonically with i
  - A fixed point, $\forall s, V_k(s) = V_{k+1}(s)$, implies π^*

Policy Evaluation

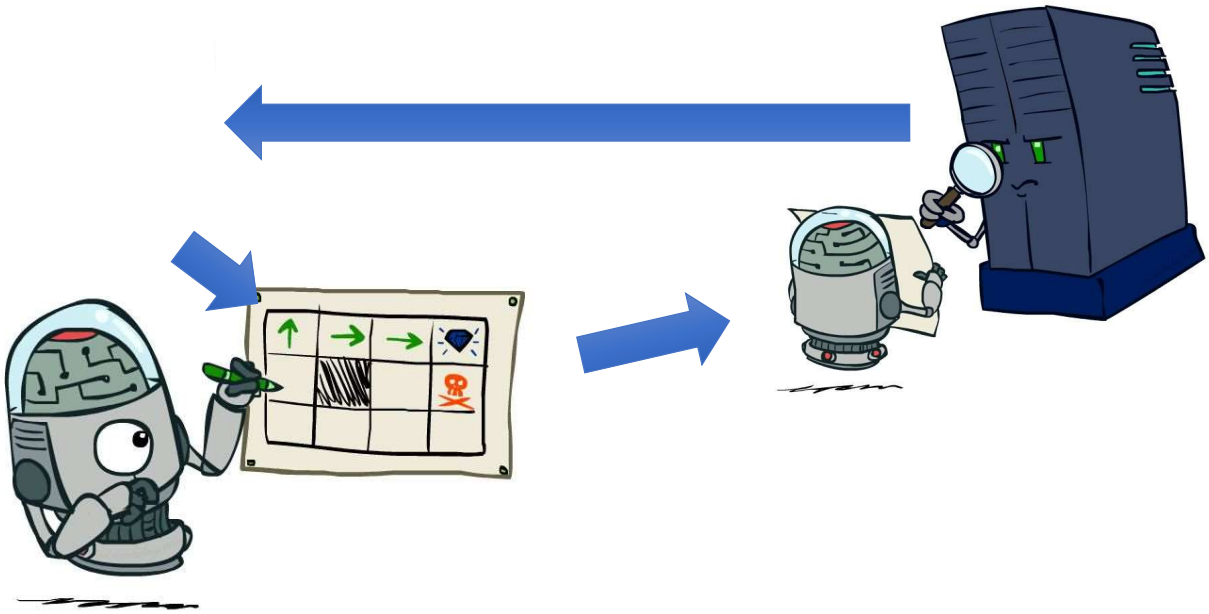
- Why is calculating V_π easier that calculating V^*?
  - Turns non-linear Bellman equations into linear equations
- $v^*(s) = \max_a [\sum_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma v^*(s'))]$
- $v_\pi(s) = \sum_{s'} P(s'|s,\pi(s))\,(R(s,\pi(s),s') + \gamma v^*(s'))$
- Solve a set of linear equations in $O(S^2)$
  - 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

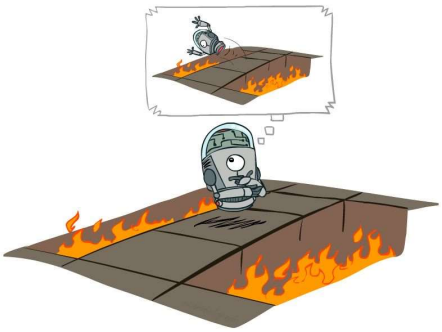


Comparison

- 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 continues action space as a distribution over the discreate 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 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