Day 5:

Speaker: Prof. Dinesh Jayaraman

Title: Tutorial for Reinforcement Learning: ML for decision making and control

Tutorial on Reinforcement Learning: ML for Decision Making and Control

Dinesh Jayaraman

CVIT Summer School, Aug 2021

Ask Questions at: PollEv.com/dineshjayaraman521

Penn Engineering | GRASP Laboratory

General Robotics, Automation, Sensing & Perception Lab

Perception, Action, & Learning Group

1

The RL Problem Setting: *Sequential Decision Making*

Actions: muscle contractions
Observations: sight, smell
Rewards: food

Actions: motor current or torque
Observations: camera images
Rewards: task success measure (e.g., running speed)

Actions: what to purchase
Observations: inventory levels
Rewards: profit

Images: Image: https://ozrobotics.com/tag/atlas-robot/
https://www.bccourier.com/global-inventory-management-software-market-size-to-register-high-growth-prospect-and-future-aspects-by-2026/

3

Any RL Problem

Actions Observations Rewards

Learning Through Trial and Error

The aim of RL is to learn to make sequential decisions in an environment:

- Driving a car
- Cooking
- Playing a videogame
- Controlling a power plant
- Treating a trauma patient
- Riding a bicycle

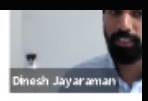
How does an RL agent learn to do these things?

- Assume only occasional feedback, such as a tasty meal, or a car crash, or video game points.
- Assume very little is known about the "environment" in advance.
- Learn through trial and error.

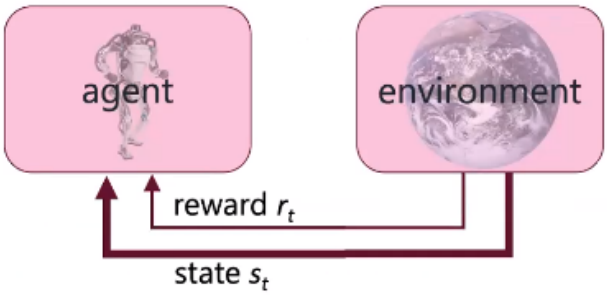
* Very few assumptions of the env. ↓ so can be applied to a large class of problems

* Reward → How well the agent is doing?

The Standard Reinforcement Learning Interface



- Agent receives observations (state $s_t \in S$) and feedback (reward r_t) from the world



Environment

① S_t is not provided to agent automatically but is estimated via some camera etc

② a_t from agent change the state from $S_t \rightarrow S_{t+1}$

Work of agent **maximize Rewards**

The Standard Reinforcement Learning Interface

- Agent receives observations (state $s_t \in S$) and feedback (reward r_t) from the world
- Agent takes action $a_t \in A$
- Agent receives updated state s_{t+1} and reward r_{t+1}

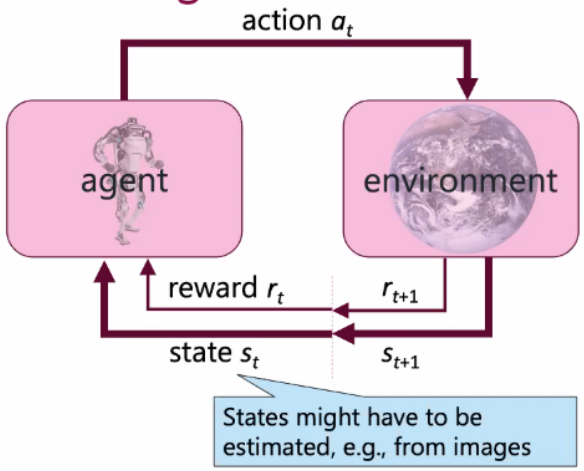
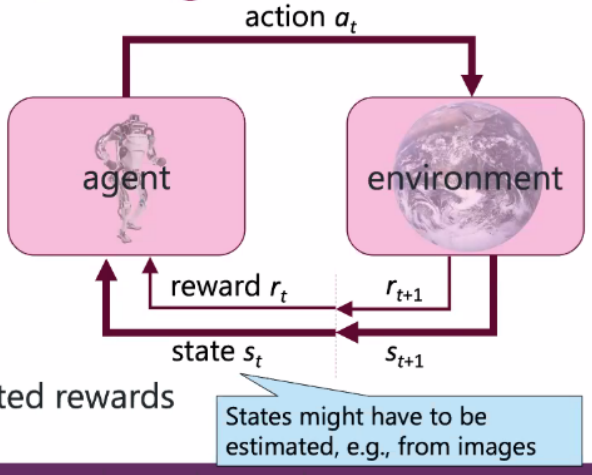


Image: https://robots.ieee.org/robots/pr2/ 5

The Standard Reinforcement Learning Interface

- Agent receives observations (state $s_t \in S$) and feedback (reward r_t) from the world
- Agent takes action $a_t \in A$
- Agent receives updated state s_{t+1} and reward r_{t+1}
- Agent's goal is to maximize expected rewards



S→A (Policy fun^n)

Goal of RL is to learn a **policy** $\pi(s): S \rightarrow A$ for acting in the environment

e.g. state s_t = robot pose, action a_t= motor torques, r_t= running speed

Image: https://robots.ieee.org/robots/pr2/ 5

∴ RL in simple words is a mapping from states, s_t to actions, a_t

How is RL Different from Supervised Learning (SL)?

Goal of SL is to model a function $h(x): X \rightarrow Y$, given training (x, y) pairs

Goal of RL is to learn a **policy** $\pi(s): S \rightarrow A$ for acting in the environment

Supervised Learning

- Target labels for h are directly available in the training data
- Train to regress* from x to y in the training data

Reinforcement Learning

- Optimal action labels a for states s are not given to us
- Train by trying various action sequences in an environment, and observing which ones produce good rewards over time.

Key Problems Specific to RL:

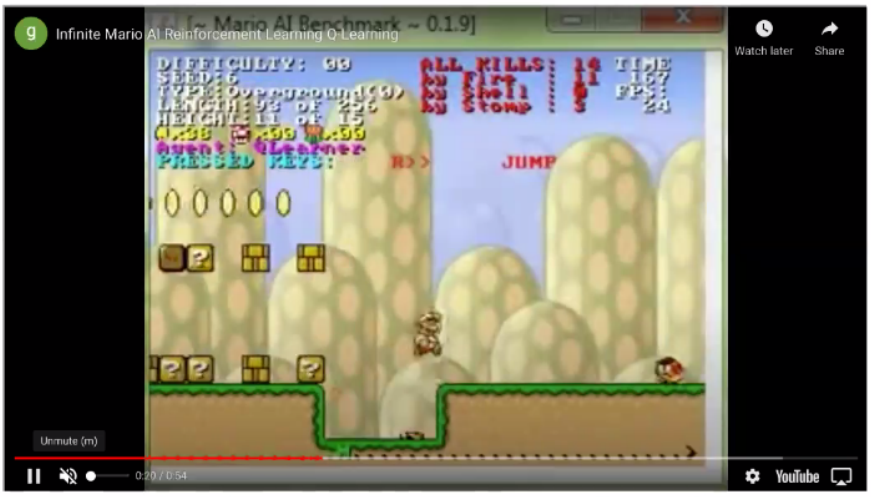
- **Credit assignment:** Which actions in a sequence were the good/bad ones?*
- **Exploration vs Exploitation:** Yes, trial-and-error, but how to pick what to try?*

6

* If we are performing a sequence of actions then after getting good/bad results we don't know which actions caused that

* What do we try? Random things! What?

Examples: Reinforcement Learning for Mario



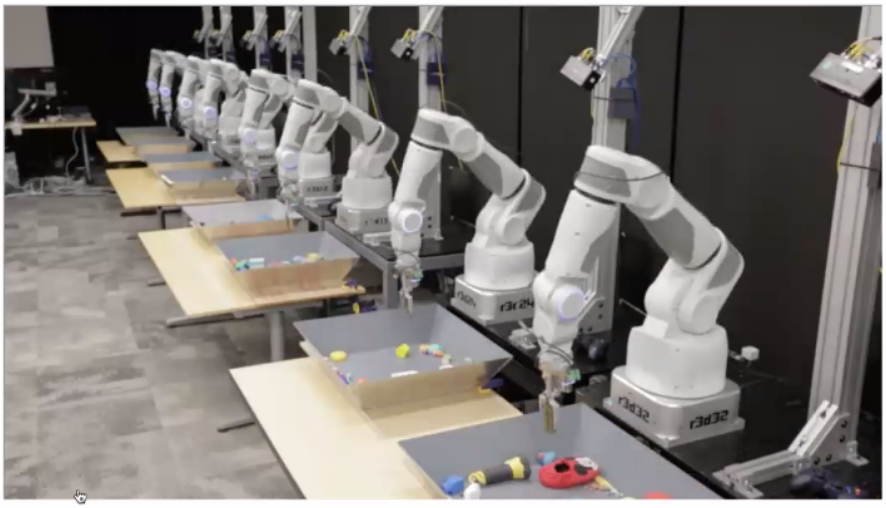
https://www.youtube.com/watch?v=4cgWya-wjgY

→ Trained to max. rewards in the game

Examples: Reinforcement Learning to Play Go

AlphaGo, an RL-based autonomous agent defeated the world champion at the board game Go!

Examples: Reinforcement Learning to Grasp



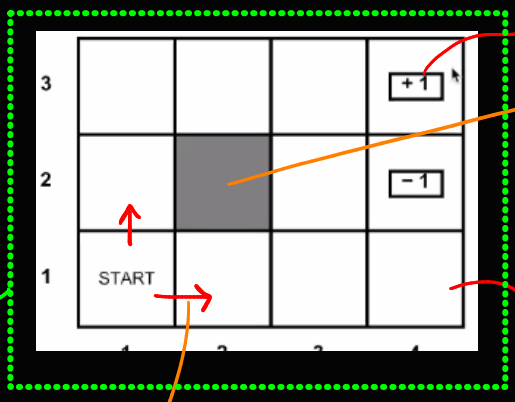
https://www.youtube.com/watch?v=iaF43Ze1oeI

Examples: Reinforcement Learning Parkour

https://www.youtube.com/watch?v=hx_bgoTF7bs

Prerequisites and Preliminaries

- Basic supervised machine learning knowledge, e.g. from an introductory ML course.
- Some basic knowledge of probability (screenshot for reference!):
  - Expectation of a random variable X
    - $\mathbb{E}[X] = \sum_x x\, P_X(x)$, where x is any valid value of X
  - Expectation (under X) of a random variable $Y(X)$
    - $\mathbb{E}_X[Y(X)] = \sum_x y(x)\, P_X(x)$
  - If not discrete variables, then integrations instead of summations.



Rewards

Can't surpass (solid Box)

Can move to transparent boxes

Grid World [sample of MDP]

actions

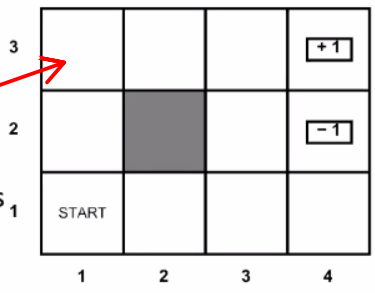
* Each cell represents a state.

* MDP → Markov Decision Process

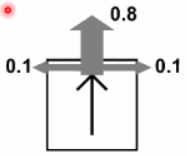
+1 & -1 in the Grid World are terminal rewards i.e once we reach that state we can't do anything more

Sample MDP: Grid World

- Agent operates in a grid with solid and open cells
- Each timestep, the agent receives a small negative "living" reward
- There are two bigger magnitude rewards at terminal states that end an episode
- The agent can move North, East, South, West
  - The agent remains where it is if it tries to move into a solid cell or outside the world
  - The chosen action succeeds 80% of the time (for an open cell)
  - 10% of the time, the agent ends up 90° off
  - Another 10% of the time, the agent ends up -90° off
    - For example, an agent surrounded by open cells and moving North will end up in the northern cell 80% of the time, in the eastern cell 10% of the time, and in the western cell 10% of the time
- Goal: maximize sum of rewards (i.e., maximize expected utility)



∴ Reach +1 quickly.



→ Transition function (Stochastic Transitions)

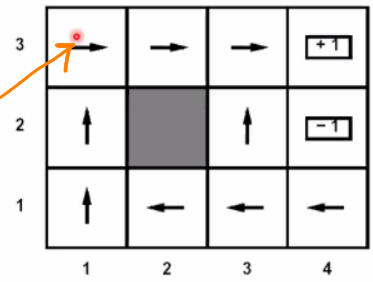
Based on slide by Dan Klein 14

Desired Outcome of RL: Optimal Policies

Goal: find the optimal policy $\pi^*(s): S \rightarrow A$

- "Optimal" $\Longrightarrow$ Following π^* maximizes total reward/utility (on average)

Example optimal policy π^*



Optimal directions ($S \rightarrow A$)

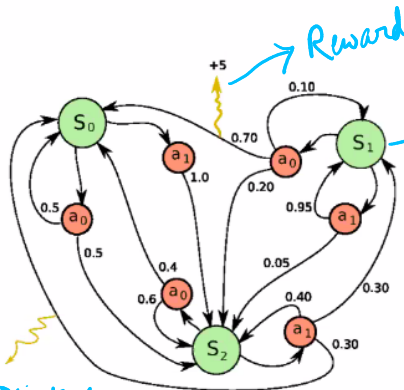
Optimal policy when $R(s,a,s') = -0.03$ for all non-terminal states

Based on slide by Dan Klein 15

Markov Decision Process

An MDP (S, A, P, R, γ) is defined by:

- Set of states $s \in S$
- Set of actions $a \in A$
- Transition function $P(s' \mid s, a)$
  - Probability $P(s' \mid s, a)$ that a from s leads to s'
  - Also "dynamics model" / just "model"
- Reward function $r_t = R(s,a,s')$ or $R(s)$
- Discount factor γ (<1)
- Goal: maximize *discounted* reward sum $\sum_t \gamma^t r_{t+1}$



Reward

MDP

associated with every transition

Preference for immediate or late reward

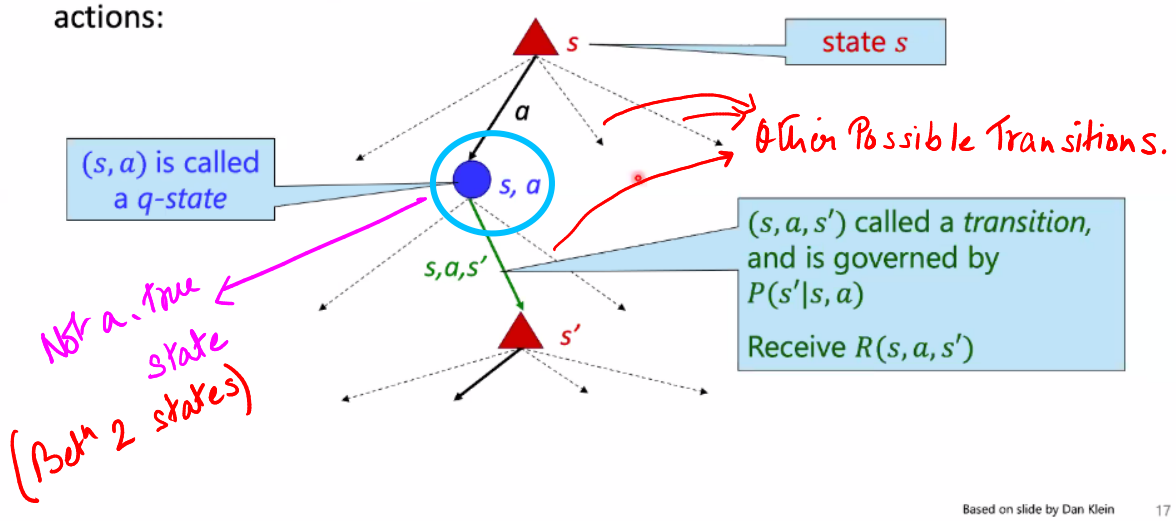
Image: https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690 16

* In RL, we do not assume knowledge of the true functions $P(\cdot)$ or $R(\cdot)$

i.e we don't have the graph apriori.

Depicting state→action→next-state trajectories

- Each MDP state has an associated tree of future outcomes from various actions:



Other Possible Transitions.

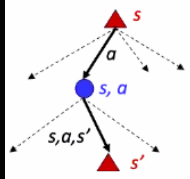
After a true state (Both 2 states)

Engineering a good reward fun^n is difficult!!

Value fun^n's

State Value Functions $V(s)$ of Policies

Given MDP (S, A, P, R, γ):



Value of a state s under policy π : (Given we know the actions already (may not be optimal)) i.e π-known

$V^\pi(s)$ = expected utility when starting in s and acting according to π

$$V^\pi(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s\right)$$

Sequence of rewards generated by following π

Optimal value of a state s :

$V^*(s)$ = expected utility when starting in s and acting optimally

$$V^*(s) = V^{\pi^*}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s\right)$$

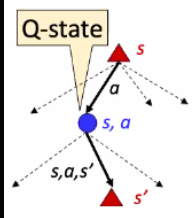
Rewards generated by following π^*

→ when we know the optimal policy π^*

Action Value Functions $Q(s, a)$ of Policies

→ Value fun^n of a Q state → (s,a)

- It is also helpful to define action-value functions



Q-value of taking action a in state s then following policy π :

$Q^\pi(s, a)$ = expected utility when taking a in s and then following π

$$Q^\pi(s, a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s, A_0 = a\right)$$

Optimal Q-value: $Q^*(s, a) = Q^{\pi^*}(s, a)$ π^*

π^* can be greedily determined from Q^*: $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a)$ → find the action that maximizes the Q^*

In other words, knowing/learning Q^* would be sufficient to act optimally.

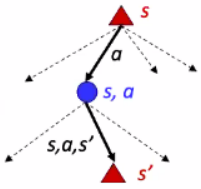
* Bellman Equation

Bellman Equation for (arbitrary) Q^π functions

Q-value of taking action a in state s then following policy π :
$Q^\pi(s,a)$ = expected utility taking a in s and then following π

$$Q^\pi(s,a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s, A_0 = a\right)$$

- The Bellman equations connect value functions at consecutive timesteps:



$$Q^\pi(s,a) = \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma Q^\pi(s', \pi(s'))]$$

expected value over successor state s' | current reward + discounted future reward

Stochastic Process ∴ Expectation

future reward

23

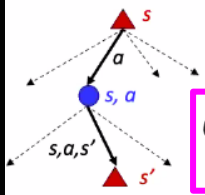
$$Q^\pi(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q^\pi(s', \pi(s'))]$$

* Bellman Equation defines the relationship between two consecutive Q fun^n's.

Bellman Equation for optimal Q functions

$$Q^\pi(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q^\pi(s', \pi(s'))]$$

Optimal values are what we get by picking the optimal action



$$Q^*(s,a) = Q^{\pi^*}(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma \max_{a'} Q^*(s', a')]$$

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

24

1/3 Recap

- Markov Decision Processes (S, A, P, R, γ)
- RL wants to find the optimal policy π^* to maximize $\sum_t \gamma^t r_{t+1}$
- One way to do this is to find the optimal Q function, Q^*
- Q^* satisfies a recursive equation, called the Bellman equation

→ Sum of discounted future rewards

Coming up next:

- How to compute Q^* if we knew the full MDP (S, A, P, R, γ)?
  - "Q iteration"
- How to *learn* Q^* from experience if we didn't know P, R?
  - "Q learning"

25

The Q Iteration: Finding Q^* & π^* in "known environments"

Solving MDPs when P, R are known

Q-Policy Iteration

Key Idea: To find Q^*, solve iteratively via dynamic programming

1. Start with a random guess, e.g., $Q_0^*(s,a) \leftarrow 0$ for all states s and actions a
2. Iterate (incrementing i, till convergence):
   - **Policy Improvement:**
     - Update the guess for optimal policy π^* to be compatible with Q_i:
       $$\pi_i(s) \leftarrow \underset{a}{\operatorname{argmax}}\, Q_i(s,a)$$
   - **Policy Evaluation:**
     - Update your guess for Q^* to be compatible with π_i:
       $$Q_{i+1}(s,a) \leftarrow Q^{\pi_i}(s,a)$$

easy to do

How to compute?

29

Q-Policy Evaluation

How do we calculate the $Q^\pi(s,a)$ for some policy $\pi(s)$?

- Recall, Bellman Equation gives us a recursive definition of the optimal value:
  $$Q^\pi(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q^\pi(s', \pi(s'))]$$
- Idea: **convert the Bellman equation into an update rule**

1. $Q_0^\pi(s,a) \leftarrow 0$
2. $Q_{j+1}^\pi(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q_j^\pi(s', \pi(s'))]$

} Iterate over and over to get the correct Q^π fun^n

30

Putting it together: Q-Policy Iteration

- Start with a random guess, e.g., $Q_0^*(s,a) \leftarrow 0$ for all states s and actions a
- Iterate (incrementing i, till convergence):
  - **Policy Improvement:**
    - Compute the corresponding policy $\pi_i^*(s) \leftarrow \underset{a}{\text{argmax}}\, Q_i^*(s,a)$
  - **Policy Evaluation:**
    - Iterate (incrementing j, till convergence?):
      - $Q_j^{\pi_j^*}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q_j^{\pi}(s', \pi(s'))]$

Can also run only a single update: "Q value iteration", or "Q iteration"

- More concise expression for Q iteration:

$$Q_{i+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

Bellman Equation for optimal Q* converted to an update rule!

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

31

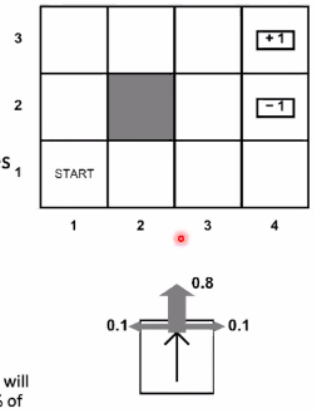
Q-Iteration example

* From previous slide

*

Grid World Environment

- Agent operates in a grid with solid and open cells
- Each timestep, the agent receives a small negative "living" reward
- There are two bigger magnitude rewards at terminal states that end an episode
- The agent can move North, East, South, West
  - The agent remains where it is if it tries to move into a solid cell or outside the world
  - The chosen action succeeds 80% of the time (for an open cell)
  - 10% of the time, the agent ends up 90° off
  - Another 10% of the time, the agent ends up -90° off
    - For example, an agent surrounded by open cells and moving North will end up in the northern cell 80% of the time, in the eastern cell 10% of the time, and in the western cell 10% of the time
- Goal: maximize sum of rewards (i.e., maximize expected utility)



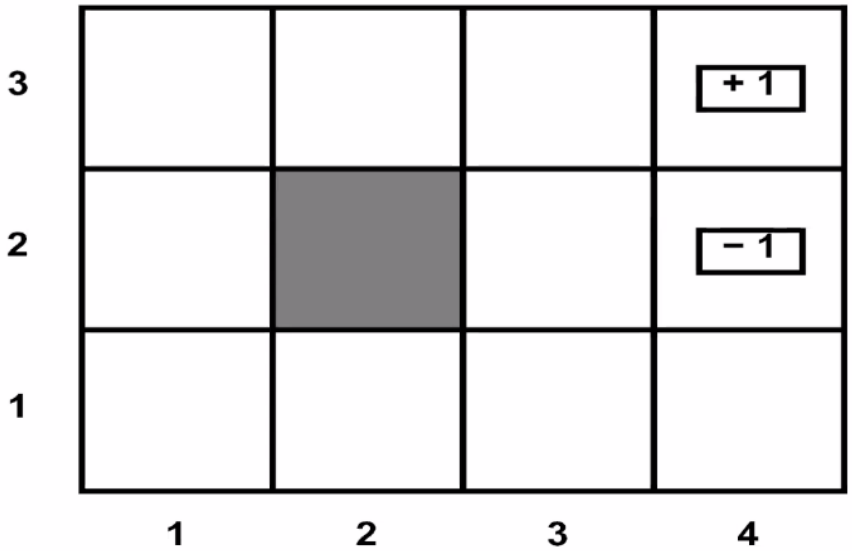
32

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

0 0.9

* Each cell is a state {Refer to previous slide}



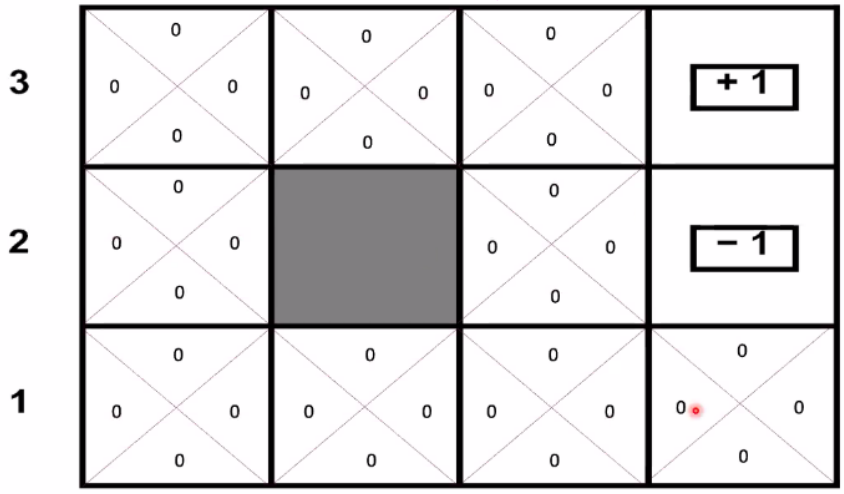
33

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

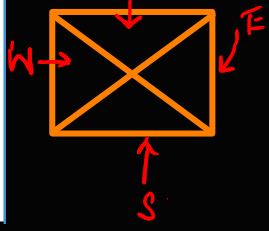
0 — expect for terminal states

0.9 (say)



$Q_0^{\pi}(s,a) = 0$

Q fun^n for being in that state & trying to move 'N'



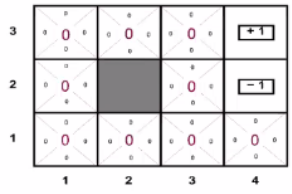
33

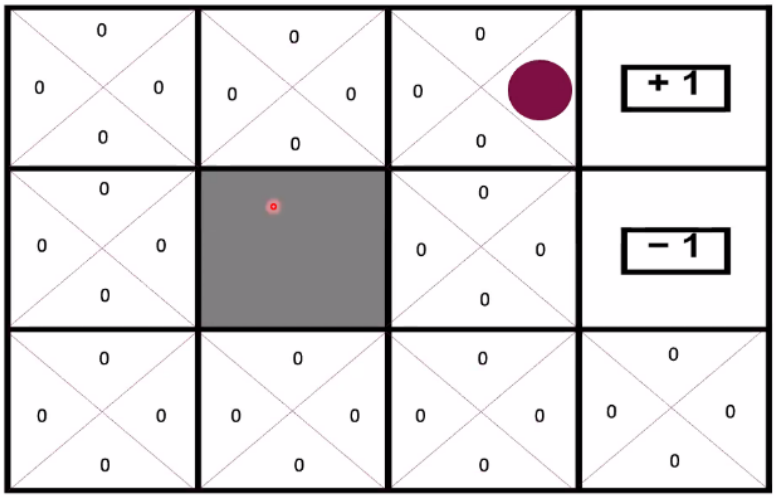
Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

0 0.9

max of Q in a given state (out of 4)



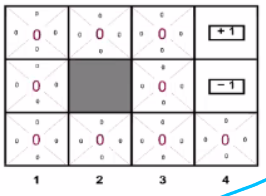


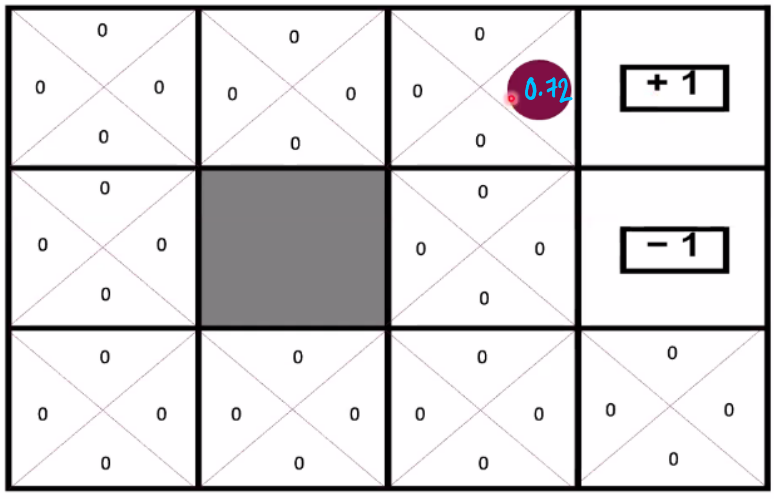
34

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

0 0.9





reward

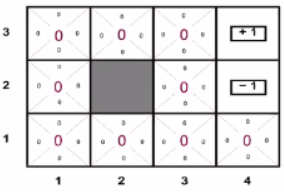
max = 1

Q = 0.8x[0+0.9x1]
+ 0.1x[0 + 0]
+0.1x[0+0]
=0.72

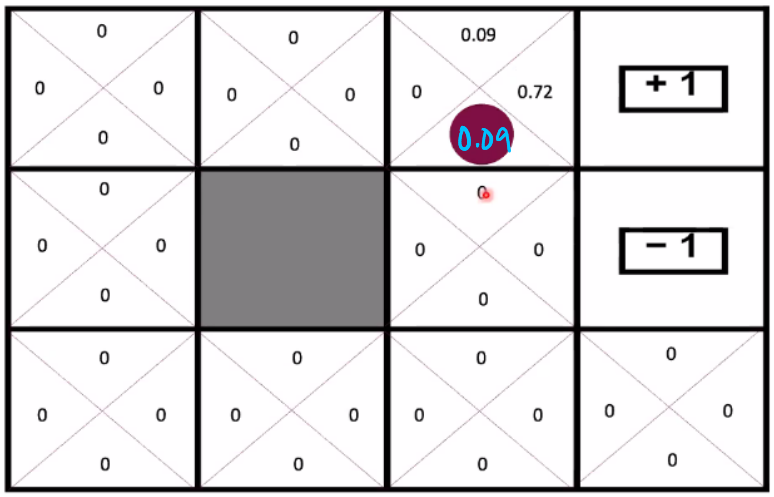
34

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$



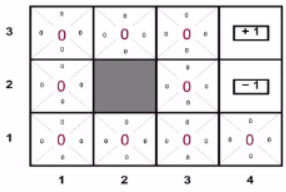
0.8x[0+0]
+ 0.1x[0+0.9x1]
+0.1x[0+0]
=0.09



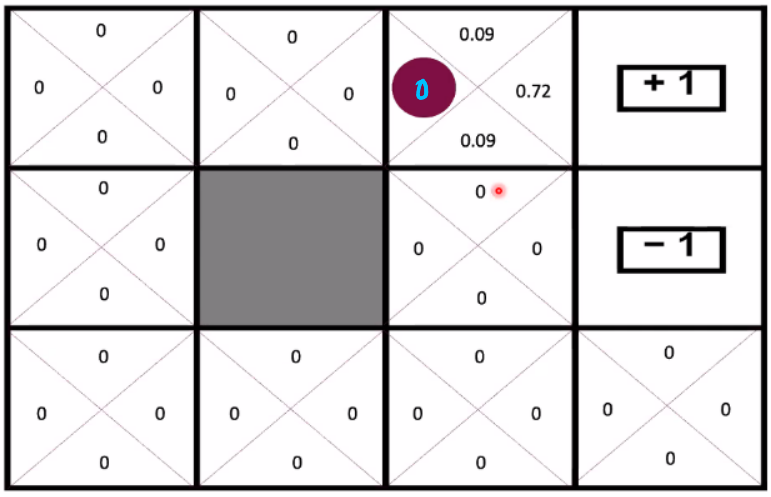
35

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$



0.8x[0+0]
+ 0.1x[0+0]
+0.1x[0+0]
=0

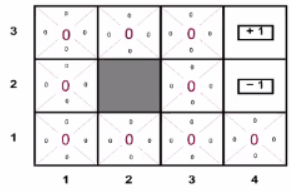


36

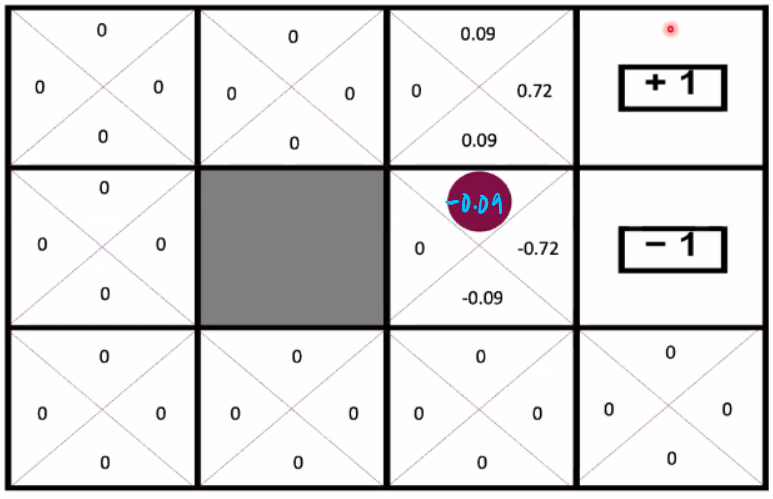
Similarly

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$



0.8x[0+0]
+ 0.1x[0+0.9x-1]
+0.1x[0+0]
=-0.09

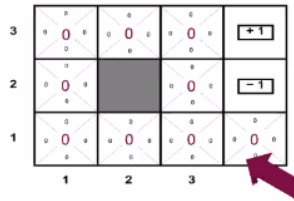


39

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

(0 and 0.9 mark the $R(s,a,s')$ and γ terms)



	1	2	3	4
3	0	0	0.09 / 0 / 0.72 / 0.09	+1
2	0	(wall)	-0.09 / 0 / -0.72 / -0.09	−1
1	0	0	0	-0.72 / -0.09 / -0.09 / 0

Now we have $Q_1(s,a)$ for all (s,a)

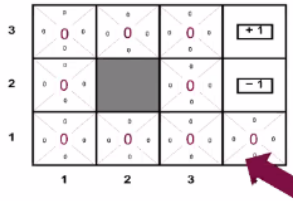
Only the reward boundary states are better.

41

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

(0 and 0.9 mark the $R(s,a,s')$ and γ terms)



	1	2	3	4
3	0	0	0.09 / 0 / 0.72 / 0.09 / .72	+1
2	0	(wall)	-0.09 / 0 / -0.72 / -0.09	−1
1	0	0	0	-0.72 / -0.09 / -0.09 / 0

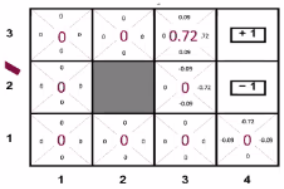
Now we have $Q_1(s,a)$ for all (s,a)

41

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

(0 and 0.9 mark the $R(s,a,s')$ and γ terms)



	1	2	3	4
3	0	0	0.09 / 0 / 0.7848 / 0.09	+1
2	0	(wall)	-0.09 / 0 / -0.72 / -0.09	−1
1	0	0	0	-0.72 / -0.09 / -0.09 / 0

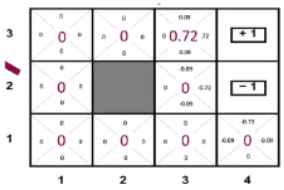
0.8x[0+0.9x1]
+ 0.1x[0+0.9x0.72]
+0.1x[0+0]
=0.7848

42

Q-Iteration example

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

(0 and 0.9 annotated above the terms)



0.8x[0+0]
+ 0.1x[0+0.9x1]
+0.1x[0+0]
=0.09

And so on till convergence...

	1	2	3	4
3	0 / 0 / 0 / 0	0 / 0 / 0 / 0	0.09 / 0 / 0.78 / 0.09	+1
2	0 / 0 / 0 / 0		-0.09 / 0 / -0.72 / -0.09	−1
1	0 / 0 / 0 / 0	0 / 0 / 0 / 0	0 / 0 / 0 / 0	-0.72 / -0.09 / -0.09 / 0

43

Q-Iteration example

(after 1000 sweeps over (s,a))

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

(0 and 0.9 annotated above the terms)

	1	2	3	4
3	0.59 / 0.57 → **0.64** / 0.53	0.67 / 0.60 → **0.74** / 0.67	0.77 / 0.66 → **0.85** / 0.57	+1
2	**0.57** / 0.51 / 0.51 / 0.46		**0.57** / 0.53 / -0.60 / 0.30	−1
1	**0.49** / 0.45 / 0.41 / 0.44	0.40 / **0.43** ← 0.42 / 0.40	**0.48** / 0.40 / 0.29 / 0.41	-0.65 / **0.28** ← 0.13 / 0.27

→ Optimal action → highest Q fun^n

44

* Policy tells us where to move for optimality.

2/3 Recap

- When transition probabilities P and reward function R are known i.e., the full MDP (S, A, P, R, γ) is known:
  - The Bellman equation for the optimal Q function can be converted to an update rule: **the "Q value iteration", or "Q iteration"**

- Note that we heavily relied on known P and R!

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

(0 and 0.9 annotated above the terms)

Coming up next:

- How to *learn* Q^* from experience if we didn't know P, R?
  - "Q learning"

45

Your Very First RL Algorithm: Q Learning

Replacing known P and R with samples from experience

Q Learning

Q-value iteration: $Q_{i+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$

How to extend this to when the functions $P(s'|s,a)$ and $R(s,a,s')$ are unknown and only revealed gradually through experience?

Note: Every time you take action a from state s, you get one sample from the unknown $P(s'|s,a)$ and the corresponding reward $R(s,a,s')$

Collect Reward Samples

Q Learning

Q-value iteration: $Q_{i+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$

Idea: Treat the single sample you get as a rough estimate of the expectation, and apply an *incremental* update to reduce the "Bellman error":

- Execute a single action a from state s and observe s' and R:

$$sample = R(s,a,s') + \gamma \max_{a'} Q_{old}(s',a')$$

- Now, compare this sample to the LHS, and apply the *incremental* update:

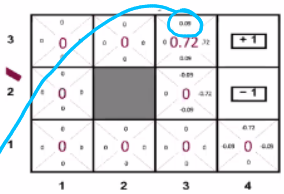
$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(\underbrace{R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)}_{\text{Bellman error}}\right)$$

Thus, we can now get the optimal Q from the agent's trial-and-error experience. **This is called "Q-Learning".**

Q-Learning example step

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

(0.1 = α, 0.9 = γ)



Sample $R + \gamma Q$ =
0+0.9x0.72 = 0.648

New Q =
0.09+0.1X(0.648-0.09)
= 0.1458

Earlier Value

3	0 / 0 / 0 / 0	0 / 0 / 0 / 0	0 / 0 / 0.78 / 0.09 (0.1458)	+1
2	0 / 0 / 0 / 0		0 / -0.09 / -0.72 / -0.09	−1
1	0 / 0 / 0 / 0	0 / 0 / 0 / 0	0 / 0 / 0 / 0	-0.72 / -0.09 / -0.09 / 0
	1	2	3	4

till convergence!

51

Q-Learning example
(after 100,000 actions)

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

3	0.59 / 0.57 → **0.64** / 0.53	0.67 / 0.60 → **0.74** / 0.67	0.77 / 0.66 → **0.85** / 0.57	+1
2	**0.57** / 0.51 ↑ 0.51 / 0.46		**0.57** / 0.53 ↑ -0.60 / 0.30	−1
1	**0.49** / 0.45 ↑ 0.41 / 0.44	0.40 / **0.43** ← 0.42 / 0.40	**0.48** / 0.40 ↑ 0.29 / 0.41	-0.65 / **0.28** ← 0.13 / 0.27
	1	2	3	4

Q Learning works! Same result as with Q iteration, even with unknown P, R.

52

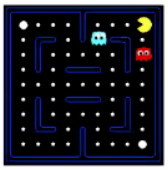
From Tabular to Deep Q Learning

Handling high-dimensional state spaces

High-dimensional states 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 its q states:



- Or even about this one!

similar bad state

Slide by Dan Klein 56

Q-Learning

- In many real situations, we cannot possibly learn about every single state+action!
  - Too many state-action pairs to visit them all in training
  - Too many state-action pairs to hold the q-tables in memory → Computational Expenses
- * Instead, we want to generalize:
  - Learn about some small number of training q-states from experience
  - Generalize that experience to new, similar q-states
  - This is a fundamental idea in machine learning, and we see it over and over again

Based on slide by Dan Klein 57

Solution

Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - ..... etc.
  - Can also describe a q-state (s, a) with features
    - e.g. action moves closer to food

To evaluate correct action.

As we now do in computer vision, can we avoid engineering these features?

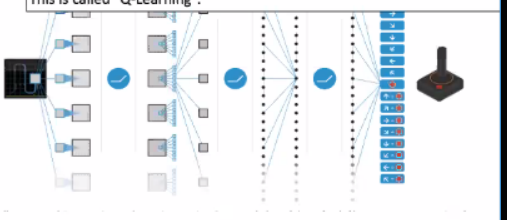
A Neural Network to Predict Q

like CNN

Predict Q-values with a deep neural network

- Execute a single action a from state s and observe s' and R: $sample = R(s,a,s') + \gamma \max_{a'} Q^*(s',a')$
- So, the incremental TD update is: $Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$ (Bellman error)

This is called "Q-Learning".



- **Input:** the state, e.g. an image
- **Output:** Q-values of various actions

- **Learning:**

  gradient descent with the squared Bellman error loss:

$$\left(\left(R(s,a,s') + \gamma \max_{a'} Q(s',a')\right) - Q(s,a)\right)^2$$

o/p of fn NN

As always, the policy action is the one with the highest predicted Q-value

Deep Q-Learning v1

- Execute a single action a from state s and observe s' and R: $sample = R(s,a,s') + \gamma \max_{a'} Q^*(s',a')$
- So, the incremental TD update is: $Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$ (Bellman error)

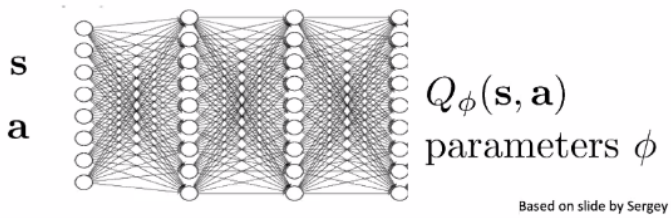
This is called "Q-Learning".

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ → Sample
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i,\mathbf{a}_i)}{d\phi}\left(Q_\phi(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{y}_i\right)$

$= \frac{d}{d\phi}(Q_\phi - y_i)^2$ } Update Rule

To read about * deep Q Learning

Incremental update step → gradient descent* on the squared Bellman error loss!



Based on slide by Sergey Levine

2.5/3 Recap

- Q-Learning: We can modify Q iteration when P and R are unknown:
  - Treat each sample from the distribution as a coarse proxy for the mean
  - Make updates *incremental*
- Deep Q-Learning v1:
  - To handle high-dimensional states, replace table by a deep network that maps (s,a) to $Q(s,a)$
  - Convert incremental update to gradient descent

Problems with Deep Q-Learning v1

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

Problems:

- sequential states are strongly correlated (not i.i.d.)
- target value is always changing

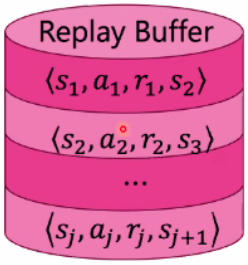
Identical & independently distributive

Solution

Based on slide by Sergey Levine 64

Addressing Correlations: Experience Replay

- Q-Learning is "off-policy": we don't say anything about the specific actions that need to be executed, and we don't need the transitions to be in sequence.
- Maintain buffer of previous experiences
- Perform Q-updates based on a sample from the replay buffer



FIFO or Priority Queue

- Advantages:
  - Breaks correlations between consecutive samples
  - Each experience step may influence multiple gradient updates

* To read →

Based on slide by Sergey Levine 65

Deep Q Learning v2 (with replay buffer $\mathcal{D}$)

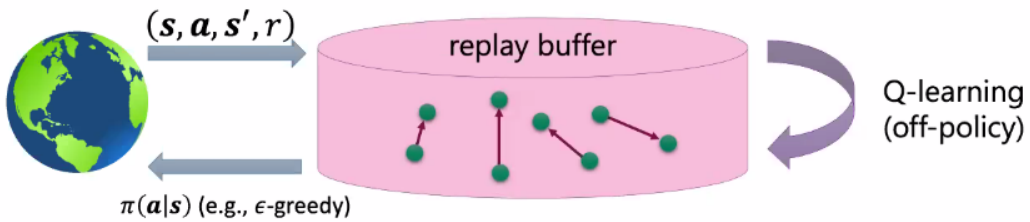
Deep Q Learning v1

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

Deep Q Learning v2

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{D}$
2. Loop K times, do:
3. sample a batch of $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$'s from $\mathcal{D}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

($K\times$)



Based on slide by Sergey Levine 66

Target value changing Problem

Problem: Moving Target for Q-regression

Online Q-iteration:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

no gradient through target value

Problem: Instability (e.g., rapid changes) in Q(·) can cause it to diverge

- Q-learning is *not* gradient descent on any fixed objective!

Solution: use two nets to provide stability

- The Q-network is updated regularly
- The target network is an older version of the Q-network, updated occasionally

$$\left(Q_\phi(s,a) - \left(r_i + \gamma \max_{a'} Q_{\phi'}(s',a') \right) \right)^2$$

computed via Q-network

computed via target network

Based on slide by Sergey Levine

Deep Q Learning v3

To read *

Deep Q Learning v2

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{D}$
2. Loop K times, do:
3. sample a batch of $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$'s from $\mathcal{D}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

Deep Q Learning v3

1. save target network parameters: $\phi' \leftarrow \phi$
2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{D}$
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{D}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

This is the "classic" deep Q Learning algorithm from 2015!*

*(usually K=1)

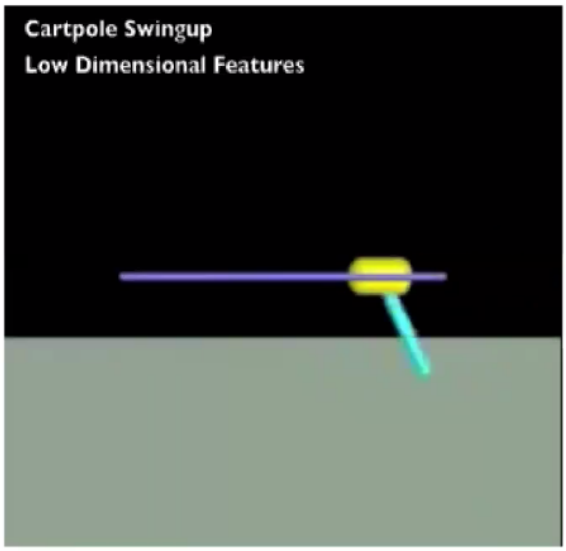
Based on slide by Sergey Levine

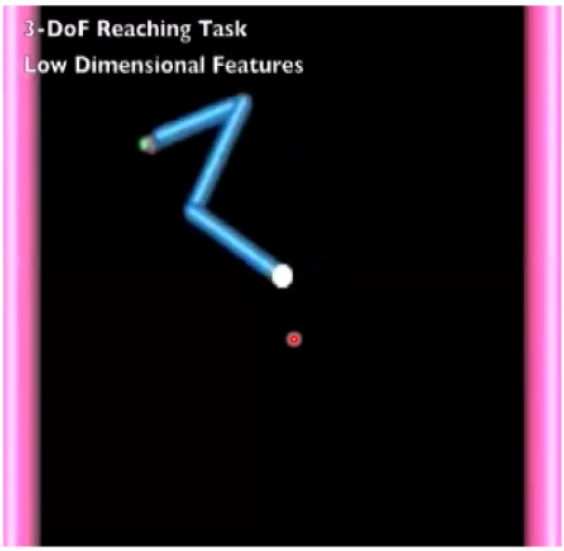
DQN on Atari Games

https://www.youtube.com/watch?v=rQIShnTz1kU

Deep Deterministic Policy Gradients

DQN for Continuous Control (DDPG)





Go forth and explore!

- This is just a 90-minute primer to RL. If you're excited about this, there's lots more to learn!
- Some useful resources:
  - 2-week mini-course in collaboration with Chuning Zhu and the Penn CIS 522 team: interactive ipython notebooks with an embedded variant of this lecture at https://github.com/CIS-522/course-content/tree/main/tutorials/
  - Sergey Levine's Deep RL 1-semester course (public video lectures and slides): http://rail.eecs.berkeley.edu/deeprlcourse/
  - Textbook: Sutton and Barto, http://incompleteideas.net/book/the-book-2nd.html

Lots of Open Research Problems

A short list:

- High-dimensional state spaces (like vision) & action spaces
- Sample complexity
- Learning from logs of an external behavior policy
- Safety during learning and deployment
- Where do rewards come from?
- Delays in perception, policy computation, execution
- Exploration, resets