10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Reinforcement Learning: Q-Learning

Matt Gormley
Lecture 24
Apr. 13, 2020

Reminders

- **Homework 8: Reinforcement Learning**
  - Out: Fri, Apr 10
  - Due: Wed, Apr 22 at 11:59pm

- **Today's In-Class Poll**
  - http://poll.mlcourse.org

VALUE ITERATION

Value Iteration

Algorithm 1 Value Iteration

procedure VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
    Initialize value function $V(s) = 0$ or randomly
    while not converged **do**
        for $s \in \mathcal{S}$ **do**
            $V(s) = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')$
    Let $\pi(s) = \operatorname{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \ \forall s$
    return π

Variant 2: without Q(s,a) table

Value Iteration

Algorithm 1 Value Iteration

procedure ValueIteration($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)

  Initialize value function $V(s) = 0$ or randomly

  while not converged **do** $Q(s,a) = 0$

    for $s \in \mathcal{S}$ **do**

      for $a \in \mathcal{A}$ **do**

        $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')$

      $V(s) = \max_a Q(s,a)$

  Let $\pi(s) = \operatorname{argmax}_a Q(s,a), \ \forall s$

  return π

Variant 1: with Q(s,a) table

Synchronous vs. Asynchronous Value Iteration

Algorithm 1 Asynchronous Value Iteration

procedure ASYNCHRONOUSVALUEITERATION($R(s,a), p(\cdot|s,a)$)
    Initialize value function $V(s)^{(0)} = 0$ or randomly
    $t = 0$
    while not converged **do**
        for $s \in \mathcal{S}$ **do**
            $V(s)^{(t+1)} = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')^{(t)}$
            $\boxed{t = t + 1}$
    Let $\pi(s) = \operatorname{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \ \forall s$
    return π

asynchronous updates: compute and update V(s) for each state one at a time

Algorithm 1 Synchronous Value Iteration

procedure SYNCHRONOUSVALUEITERATION($R(s,a), p(\cdot|s,a)$)
    Initialize value function $V(s)^{(0)} = 0$ or randomly
    $t = 0$
    while not converged **do**
        for $s \in \mathcal{S}$ **do**
            $V(s)^{(t+1)} = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')^{(t)}$
        $\boxed{t = t + 1}$
    Let $\pi(s) = \operatorname{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \ \forall s$
    return π

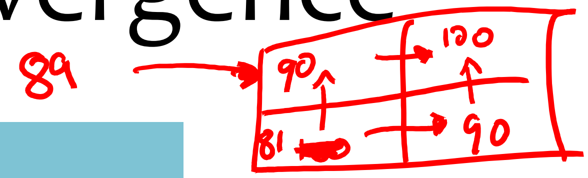
synchronous updates: compute all the fresh values of V(s) from all the stale values of V(s), then update V(s) with fresh values

Value Iteration Convergence

very abridged

Theorem 1 (Bertsekas (1989))
V converges to V^*, if each state is visited infinitely often

Holds for both asynchronous and sychronous updates



Theorem 2 (Williams & Baird (1993))
if $max_s|V^{t+1}(s) - V^t(s)| < \epsilon$
then $max_s|V^{t+1}(s) - V^*(s)| < \frac{2\epsilon\gamma}{1-\gamma}, \forall s$

Provides reasonable stopping criterion for value iteration

Theorem 3 (Bertsekas (1987))
greedy policy will be optimal in a finite number of steps (even if not converged to optimal value function!)

Often greedy policy converges well before the value function

A

B

C = certainty

Value Iteration Variants

Question:

True or False: The value iteration algorithm shown below is an example of **synchronous** updates

---

Algorithm 1 Value Iteration

---

procedure ValueIteration($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
    Initialize value function $V(s) = 0$ or randomly
    while not converged **do**
        for $s \in \mathcal{S}$ **do**
            for $a \in \mathcal{A}$ **do**
                $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')$
            $V(s) = \max_a Q(s,a)$
    Let $\pi(s) = \operatorname{argmax}_a Q(s,a), \ \forall s$
    return π

POLICY ITERATION

Policy Iteration

Algorithm 1 Policy Iteration

procedure POLICYITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)

  Initialize policy π randomly

  while not converged **do**

    Solve Bellman equations for fixed policy π

$$V^\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,\pi(s)) V^\pi(s'),\ \ \forall s$$

    Improve policy π using new value function

$$\pi(s) = \operatorname*{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s')$$

  return π

Policy Iteration

Algorithm 1 Policy Iteration

procedure POLICYITERATION($R(s,a$ … n, $p(\cdot|s,a)$ transition probabilities)

Initialize policy π randomly

while not converged **do**

Solve Bellman equations for fixed policy π

$$V^\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,\pi(s))V^\pi(s'), \;\; \forall s$$

Improve policy π using new value function

$$\pi(s) = \operatorname*{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V^\pi(s')$$

return π

Compute value function for fixed policy is easy

System of |S| equations and |S| variables

Greedy policy w.r.t. current value function

Greedy policy might **remain the same** for a particular state if there is no better action

Policy Iteration Convergence

Q2

In-Class Exercise: $|S|$ $|A|$

How many policies are there for a finite sized state and action space?

$|S|^{(|A|)}$ $\boxed{|A|^{(|S|)}}$

Q3

In-Class Exercise:

Suppose policy iteration is shown to improve the policy at every iteration. Can you bound the number of iterations it will take to converge? If yes, what is the bound? If no, why not?

iters $< |A|^{|S|}$

Value Iteration vs. Policy Iteration

solving the Bellman equation

- Value iteration requires $O(|A|\,|S|^2)$ computation per iteration

- Policy iteration requires $O(|A|\,|S|^2 + |S|^3)$ computation per iteration
- In practice, policy iteration converges in fewer iterations

---

Algorithm 1 Value Iteration

---

procedure ValueIteration($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
  Initialize value function $V(s) = 0$ or randomly
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      for $a \in \mathcal{A}$ **do**
        $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')$
      $V(s) = \max_a Q(s,a)$
  Let $\pi(s) = \text{argmax}_a\, Q(s,a),\ \forall s$
  return π

Algorithm 1 Policy Iteration

---

procedure PolicyIteration($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
  Initialize policy π randomly
  while not converged **do**
    Solve Bellman equations for fixed policy π

$$V^\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,\pi(s))V^\pi(s'),\ \forall s$$

    Improve policy π using new value function

$$\pi(s) = \operatorname*{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V^\pi(s')$$

  return π

Learning Objectives

Reinforcement Learning: Value and Policy Iteration

You should be able to…

1. Compare the reinforcement learning paradigm to other learning paradigms
2. Cast a real-world problem as a Markov Decision Process
3. Depict the exploration vs. exploitation tradeoff via MDP examples
4. Explain how to solve a system of equations using fixed point iteration
5. Define the Bellman Equations
6. Show how to compute the optimal policy in terms of the optimal value function
7. Explain the relationship between a value function mapping states to expected rewards and a value function mapping state-action pairs to expected rewards
8. Implement value iteration
9. Implement policy iteration
10. Contrast the computational complexity and empirical convergence of value iteration vs. policy iteration
11. Identify the conditions under which the value iteration algorithm will converge to the true value function
12. Describe properties of the policy iteration algorithm

Today's lecture is brought you by the letter….

Q

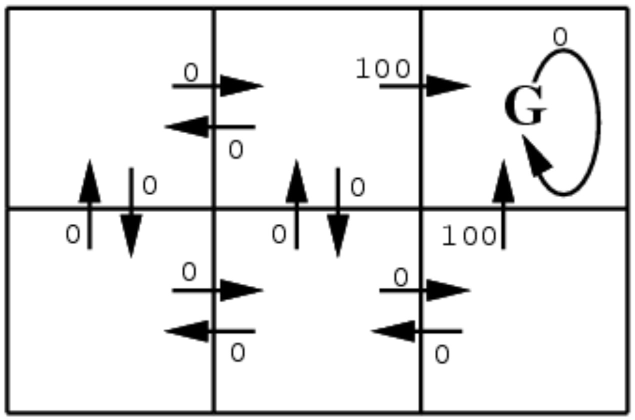
Q-LEARNING

Q-Learning

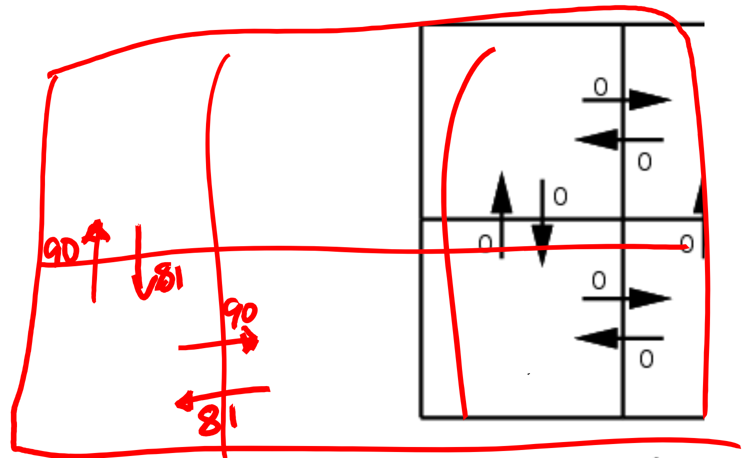
Whiteboard

- Motivation: What if we have zero knowledge of the environment?
- Q-Function: Expected Discounted Reward

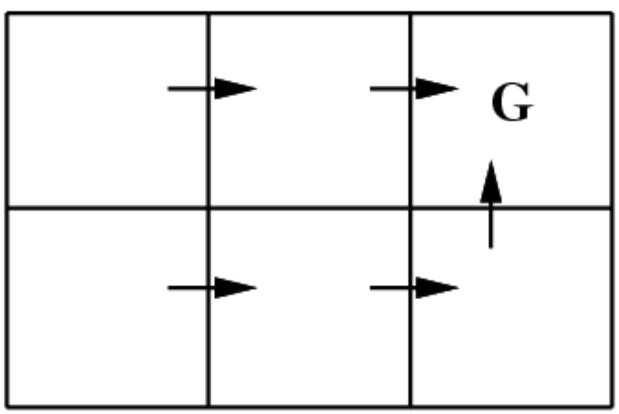
Example: Robot Localization



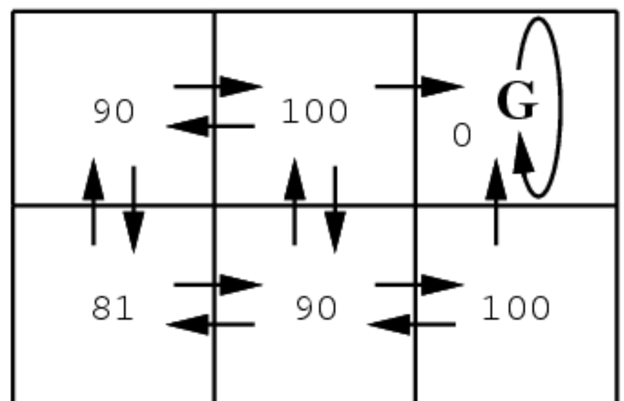
$r(s,a)$ (immediate reward) values



$Q^*(s,a)$ values $r(s,$



One optimal policy



$V^*(s)$ values

Figure from Tom Mitchell

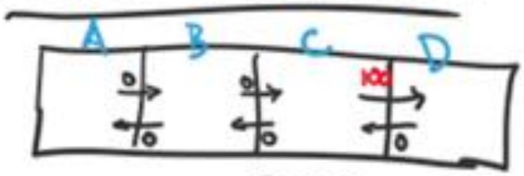
Q-Learning

Whiteboard

- Q-Learning Algorithm
  - Case 1: Deterministic Environment
  - Case 2: Nondeterministic Environment
- Convergence Properties
- Exploration Insensitivity
- Ex: Re-ordering Experiences
- ϵ-greedy Strategy

Reordering Experiences

Ex: Easiest Maze Ever!



arrows are $R(s,a)$

$\gamma = 0.9$
$S = \{A, B, C, D\}$
$A = \{E, W\}$
$Q(s,a) = 0$ at start

① suppose we visit

i	s	a	r	s'
1	A	E	0	B
2	B	E	0	C
3	C	E	100	D

$Q(A,E) = 0$
$Q(B,E) = 0$
$Q(C,E) = 100$

② suppose we visit in reverse order

i	s	a	r	s'
1	C	E	100	D
2	B	E	0	C
3	A	E	0	B

$Q(C,E) = 100$
$Q(B,E) = 90$
$Q(A,E) = 81$

diff updates

Designing State Spaces

Q: Do we have to retrain our RL agent every time we change our state space?

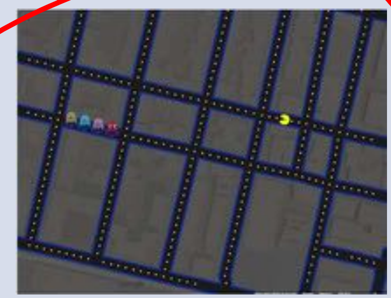
A: Yes. But whether your state space changes from one setting to another is determined by your design of the state representation.

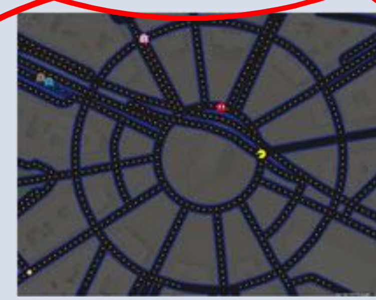
Two examples:

- State Space A: <x,y> position on map
  e.g. s_t = <74, 152>
- State Space B: window of pixel colors centered at current Pac Man location
  e.g. s_t =

0	1	0
0	0	0
1	1	1

Level 1



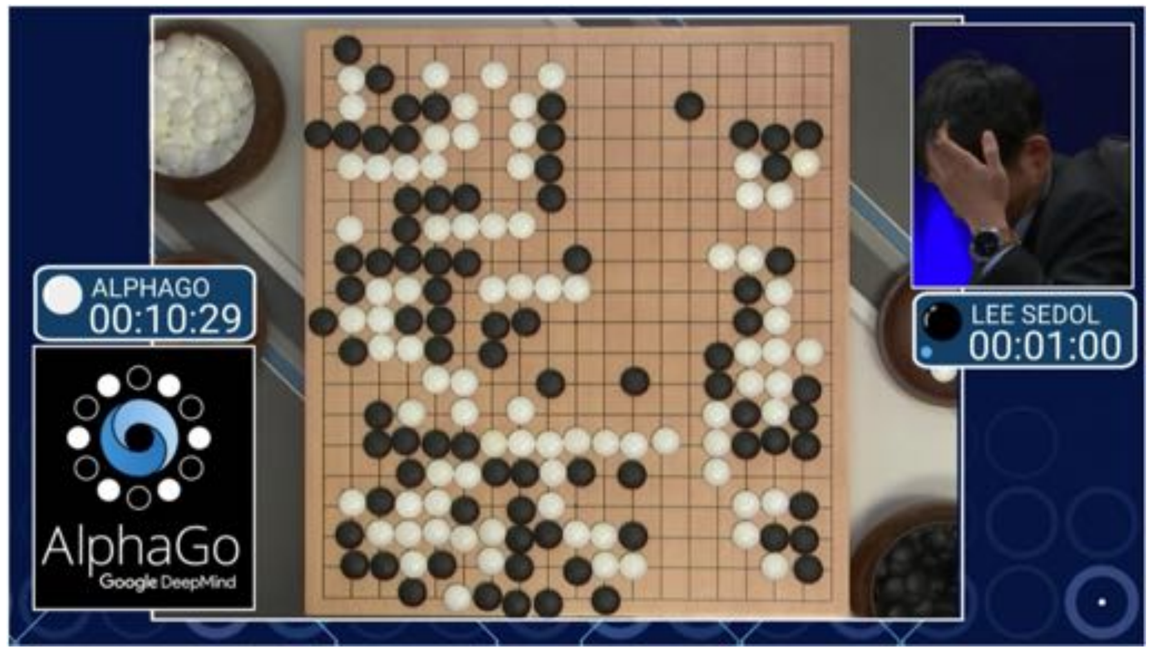


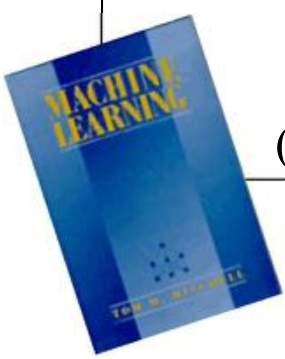
Level 2

DEEP RL EXAMPLES

TD Gammon → Alpha Go

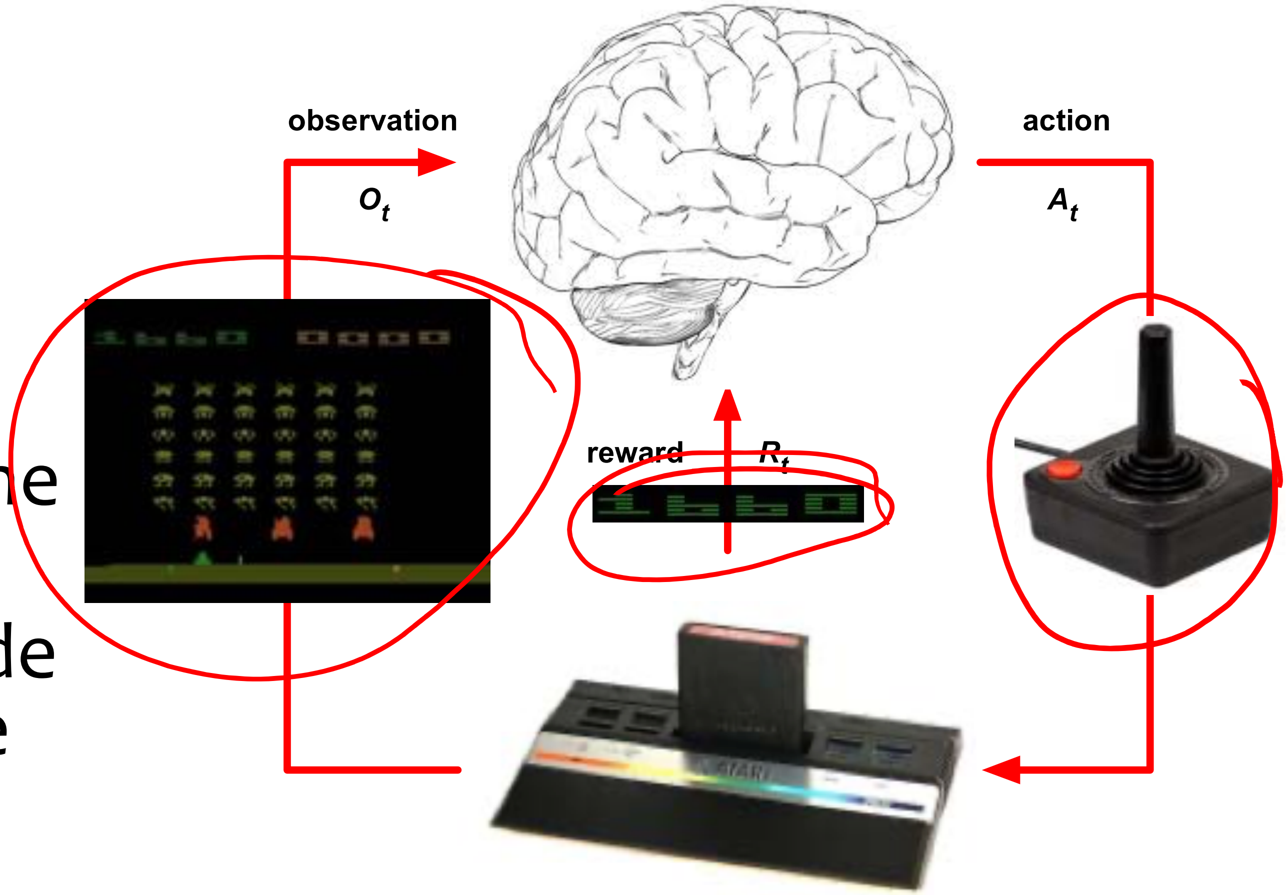
Learning to beat the masters at board games

THEN	NOW
"…the world's top computer program for backgammon, TD-GAMMON (Tesauro, 1992, 1995), learned its strategy by playing over one million practice games against itself…" (Mitchell, 1997)	



Playing Atari with Deep RL

- Setup: RL system observes the pixels on the screen
- It receives rewards as the game score
- Actions decide how to move the joystick / buttons



Figures from David Silver (Intro RL lecture)

Playing Atari with Deep RL

Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Videos:

- Atari Breakout: https://www.youtube.com/watch?v=V1eYniJ0Rnk
- Space Invaders: https://www.youtube.com/watch?v=ePv0Fs9cGgU

Figures from Mnih et al. (2013)

Playing Atari with Deep RL

Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	-20.4	157	110	179
Sarsa [3]	996	5.2	129	-19	614	665	271
Contingency [4]	1743	6	159	-17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	-3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	-16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

Figures from Mnih et al. (2013)

Deep Q-Learning

Question: *What if our state space S is too large to represent with a table?*

Examples:

- s_t = pixels of a video game
- s_t = continuous values of a sensors in a manufacturing robot
- s_t = sensor output from a self-driving car

Answer: Use a parametric function to approximate the table entries

Key Idea:

1. Use a neural network $Q(s,a; \theta)$ to approximate $Q^*(s,a)$
2. Learn the parameters θ via SGD with training examples $\underbrace{< s_t, a_t, r_t, s_{t+1} >}$

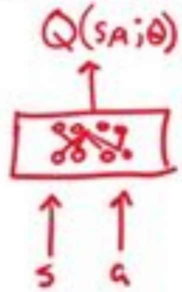
Deep Q-Learning

Whiteboard

- Strawman loss function (i.e. what we cannot compute)
- Approximating the Q function with a neural network
- Approximating the Q function with a linear model
- Deep Q-Learning
- function approximators
  (<state, action_i> → q-value
  vs.
  state → all action q-values)

not-so-Deep Q-Learning

Approx w/NN:



$|A| = K = \#$ actions

Approx w/Linear Regression:

Represent state as a feature vector:

$\vec{s}_t = [0,1,1,0,1,\dots,1]^T \qquad |\vec{s}_t| \in \mathbb{R}^M$

$M = \#$ features

$Q(\vec{s},a;\Theta) = \vec{\Theta}_a^T \vec{s}$

$$\Theta = \begin{bmatrix} \text{---} & \vec{\Theta}_1^T & \text{---} \\ \text{---} & \vec{\Theta}_2^T & \text{---} \\ & \vdots & \\ \text{---} & \vec{\Theta}_K^T & \text{---} \end{bmatrix} \qquad \Theta \in \mathbb{R}^{K \times M}$$

$$\nabla_{\vec{\Theta}_b} Q(\vec{s},a;\Theta) = \ ?$$

$$= \begin{cases} \vec{s} & \text{if } a = b \\ \vec{0} & \text{if } a \neq b \end{cases}$$

$$= \vec{s}\,\mathbb{1}(a=b)$$

$$\nabla_\Theta Q(s,a;\Theta) = \begin{array}{c} \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & & & \vdots \\ \text{---} & \vec{s}^T & & \text{---} \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \end{array} \leftarrow a^{th} \text{ row}$$

Experience Replay

- **Problems** with online updates for Deep Q-learning:
  - not i.i.d. as SGD would assume
  - quickly forget rare experiences that might later be useful to learn from
- **Uniform Experience Replay** (Lin, 1992):
  - Keep a *replay memory* $D = \{e_1, e_2, \dots, e_N\}$ of N most recent experiences $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  - Alternate two steps:
    1. Repeat T times: randomly sample e_i from D and apply a Q-Learning update to e_i
    2. Agent selects an action using epsilon greedy policy to receive new experience that is added to D
- **Prioritized Experience Replay** (Schaul et al, 2016)
  - similar to Uniform ER, but sample so as to prioritize experiences with high error

Alpha Go

Game of Go (圍棋)

- 19x19 **board**
- Players alternately play black/white **stones**
- **Goal** is to fully encircle the largest region on the board
- **Simple** rules, but **extremely complex** game play

Game 1
Fan Hui (Black), AlphaGo (White)
AlphaGo wins by 2.5 points

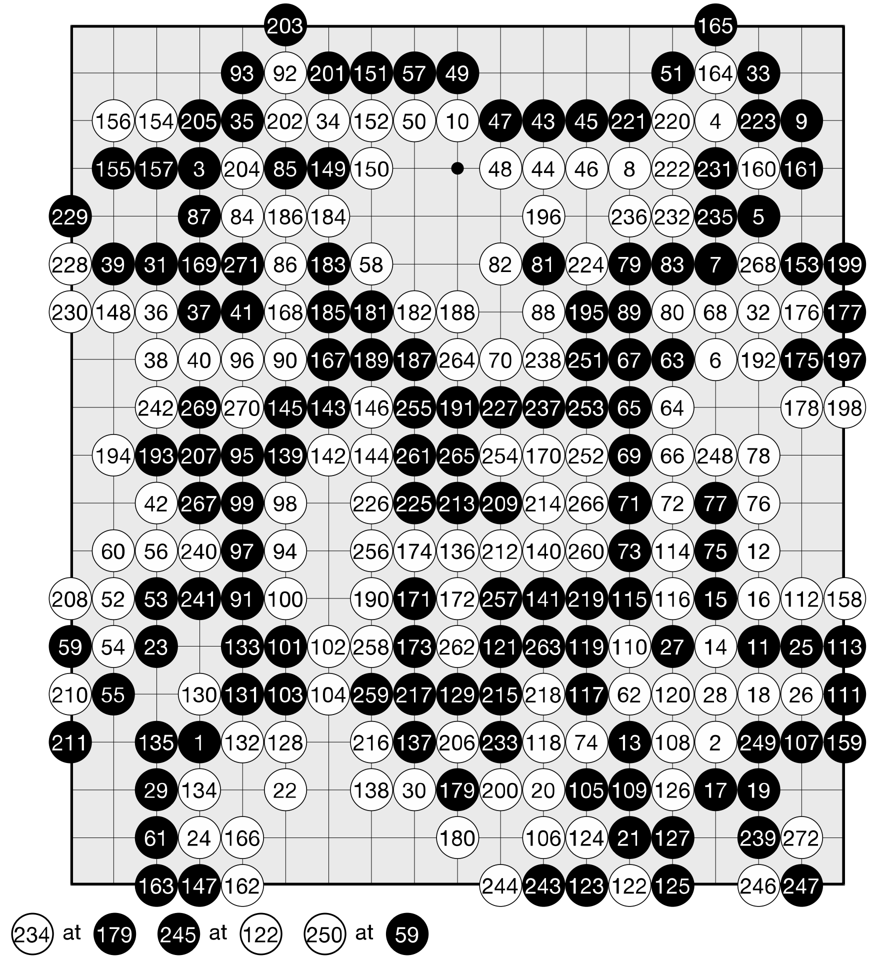


Figure from Silver et al. (2016)

Alpha Go

- State space is too large to represent explicitly since # of sequences of moves is $O(b^d)$
  - Go: $b=250$ and $d=150$
  - Chess: $b=35$ and $d=80$
- Key idea:
  - Define a neural network to approximate the value function
  - Train by policy gradient

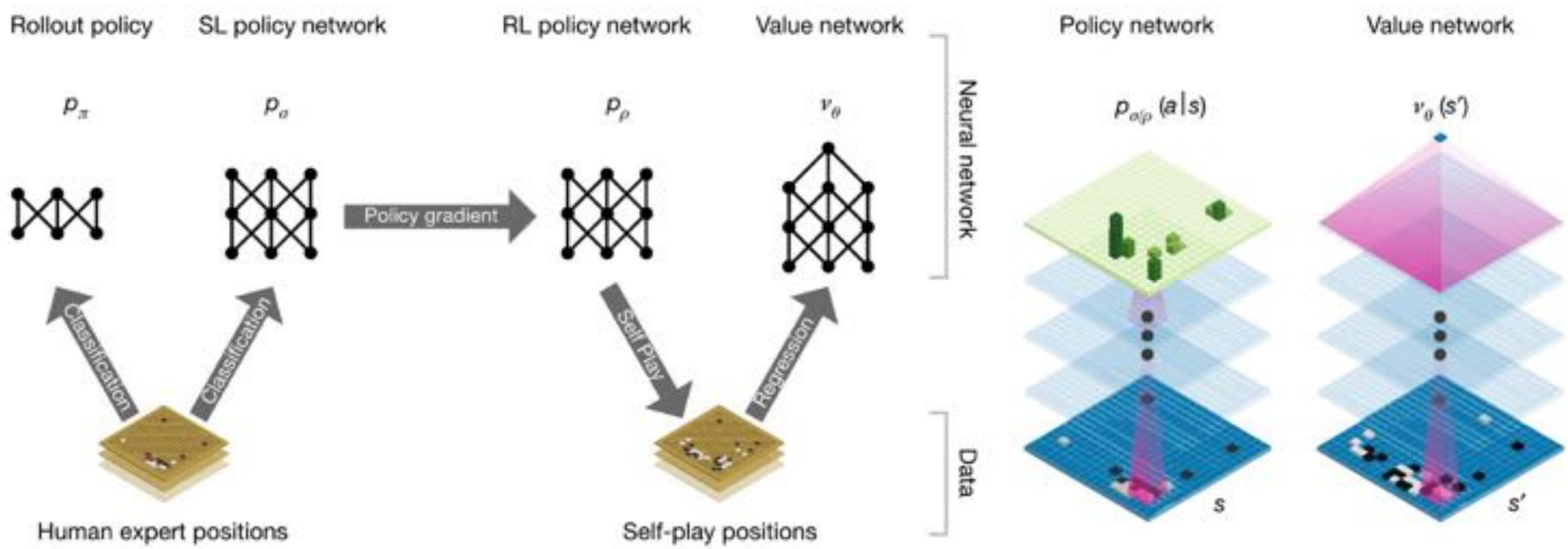


Figure from Silver et al. (2016)

Alpha Go

- Results of a tournament
- From Silver et al. (2016): "a 230 point gap corresponds to a 79% probability of winning"

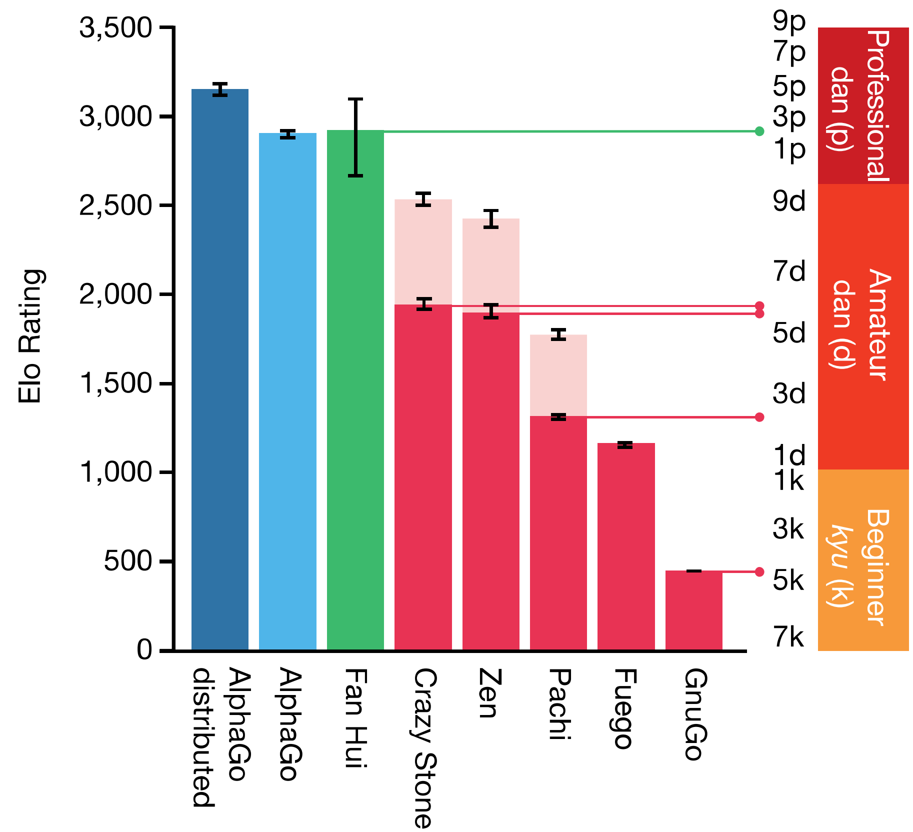


Figure from Silver et al. (2016)

Learning Objectives

Reinforcement Learning: Q-Learning

You should be able to…

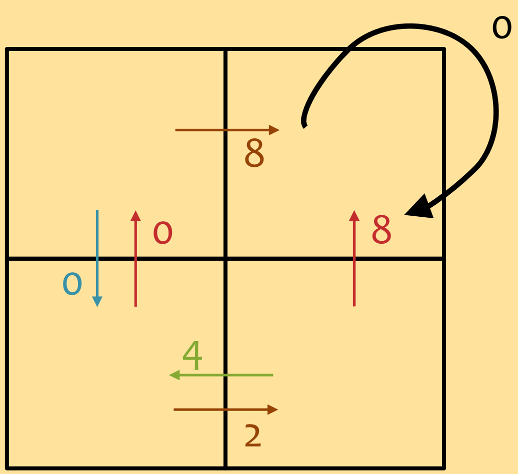
1. Apply Q-Learning to a real-world environment
2. Implement Q-learning
3. Identify the conditions under which the Q-learning algorithm will converge to the true value function
4. Adapt Q-learning to Deep Q-learning by employing a neural network approximation to the Q function
5. Describe the connection between Deep Q-Learning and regression

Q-Learning

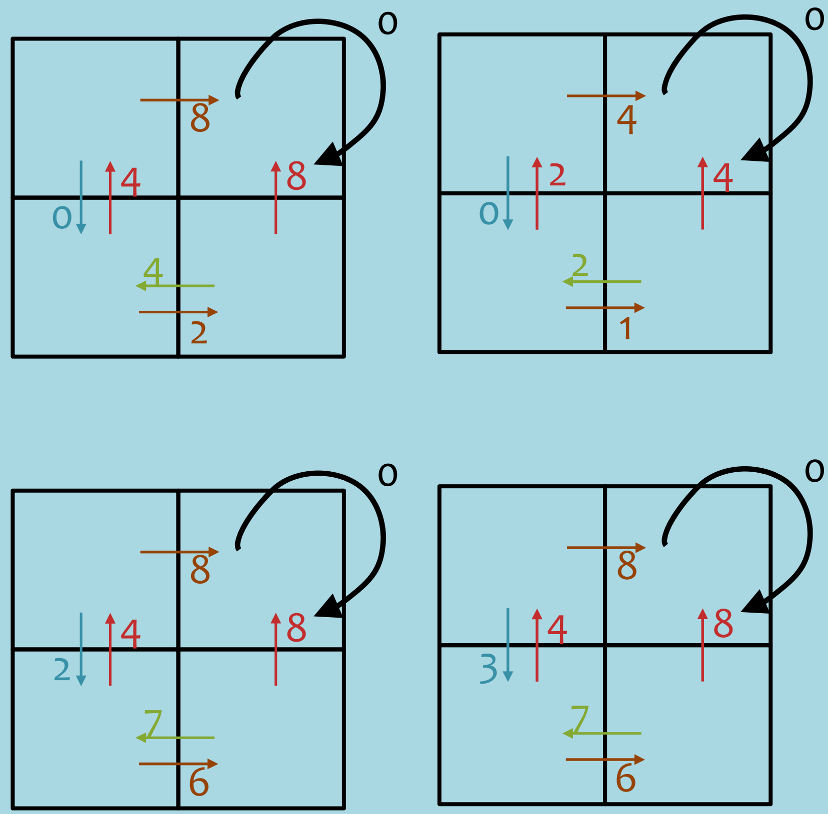
Question:

For the R(s,a) values shown on the arrows below, which are the corresponding Q*(s,a) values?

Assume discount factor = 0.5.



Answer:



ML Big Picture

Learning Paradigms:

What data is available and when? What form of prediction?

- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- active learning
- imitation learning
- domain adaptation
- online learning
- density estimation
- recommender systems
- feature learning
- manifold learning
- dimensionality reduction
- ensemble learning
- distant supervision
- hyperparameter optimization

Problem Formulation:

What is the structure of our output prediction?

boolean	Binary Classification
categorical	Multiclass Classification
ordinal	Ordinal Classification
real	Regression
ordering	Ranking
multiple discrete	Structured Prediction
multiple continuous	(e.g. dynamical systems)
both discrete & cont.	(e.g. mixed graphical models)

Application Areas

Key challenges?

NLP, Speech, Computer Vision, Robotics, Medicine, Search

Theoretical Foundations:

What principles guide learning?

- ❑ probabilistic
- ❑ information theoretic
- ❑ evolutionary search
- ❑ ML as optimization

Facets of Building ML Systems:

How to build systems that are robust, efficient, adaptive, effective?

1. *Data prep*
2. *Model selection*
3. *Training (optimization / search)*
4. *Hyperparameter tuning on validation data*
5. *(Blind) Assessment on test data*

Big Ideas in ML:

Which are the ideas driving development of the field?

- *inductive bias*
- *generalization / overfitting*
- *bias-variance decomposition*
- *generative vs. discriminative*
- *deep nets, graphical models*
- *PAC learning*
- *distant rewards*

Learning Paradigms

Paradigm	**Data**
Supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \qquad \mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$
$\hookrightarrow$ Regression	$y^{(i)} \in \mathbb{R}$
$\hookrightarrow$ Classification	$y^{(i)} \in \{1, \ldots, K\}$
$\hookrightarrow$ Binary classification	$y^{(i)} \in \{+1, -1\}$
$\hookrightarrow$ Structured Prediction	$\mathbf{y}^{(i)}$ is a vector
Unsupervised	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N \qquad \mathbf{x} \sim p^*(\cdot)$
Semi-supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$
Online	$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\}$
Active Learning	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and can query $y^{(i)} = c^*(\cdot)$ at a cost
Imitation Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \ldots\}$
Reinforcement Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \ldots\}$