School of Computer Science
Carnegie Mellon

10-701 Introduction to Machine Learning

Reinforcement Learning

Readings:
Mitchell Ch. 13

Matt Gormley
Lecture 22
November 30, 2016

Reminders

- Poster Session
  - Fri, Dec 2: 2:30pm – 5:30 pm
- Final Report
  - due Fri, Dec 9

REINFORCEMENT LEARNING

What is Learning?

- Learning takes place as a result of interaction between an agent and the world, the idea behind learning is that

  - Percepts received by an agent should be used not only for understanding/interpreting/prediction, as in the machine learning tasks we have addressed so far,

    but also for **acting**, and further more for improving the agent's ability to behave optimally in the future to achieve the goal.

Types of Learning

- Supervised Learning
  - A situation in which sample (input, output) pairs of the function to be learned can be perceived or are given
  - You can think it as if there is a kind teacher

    - Training data: (X,Y). (features, label)

    - Predict Y, minimizing some loss.

    - Regression, Classification.

- Unsupervised Learning

    - Training data: X. (features only)

    - Find "similar" points in high-dim X-space.

    - Clustering.

Example of Supervised Learning

- Predict the price of a stock in 6 months from now, based on economic data. (Regression)
- Predict whether a patient, hospitalized due to a heart attack, will have a second heart attack. The prediction is to be based on demographic, diet and clinical measurements for that patient. (Logistic Regression)
- Identify the numbers in a handwritten ZIP code, from a digitized image (pixels). (Classification)

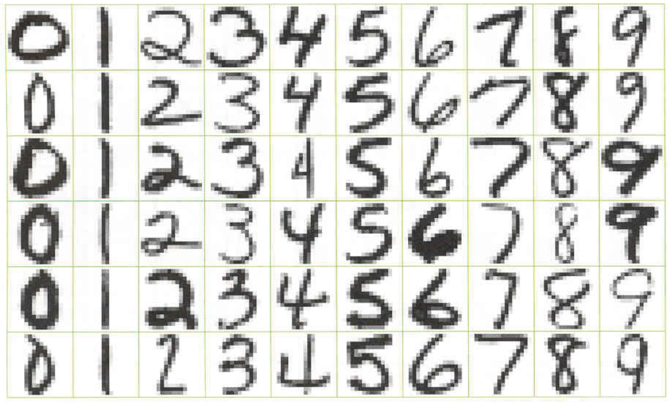
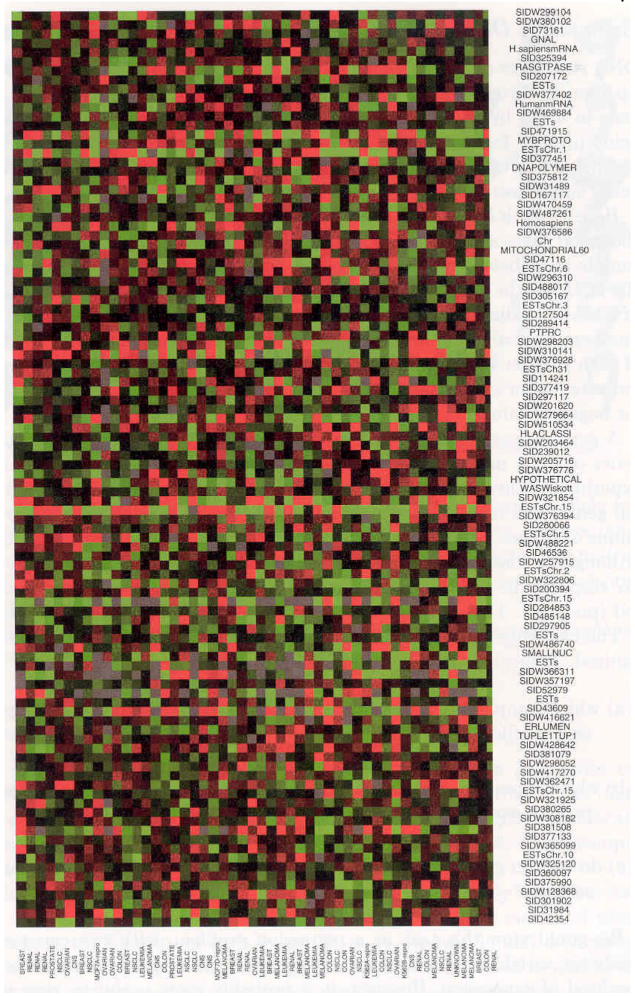


FIGURE 1.2. Examples of handwritten digits from U.S. postal envelopes.

Example of Unsupervised Learning

- From the DNA micro-array data, determine which genes are most “similar” in terms of their expression profiles. (Clustering)

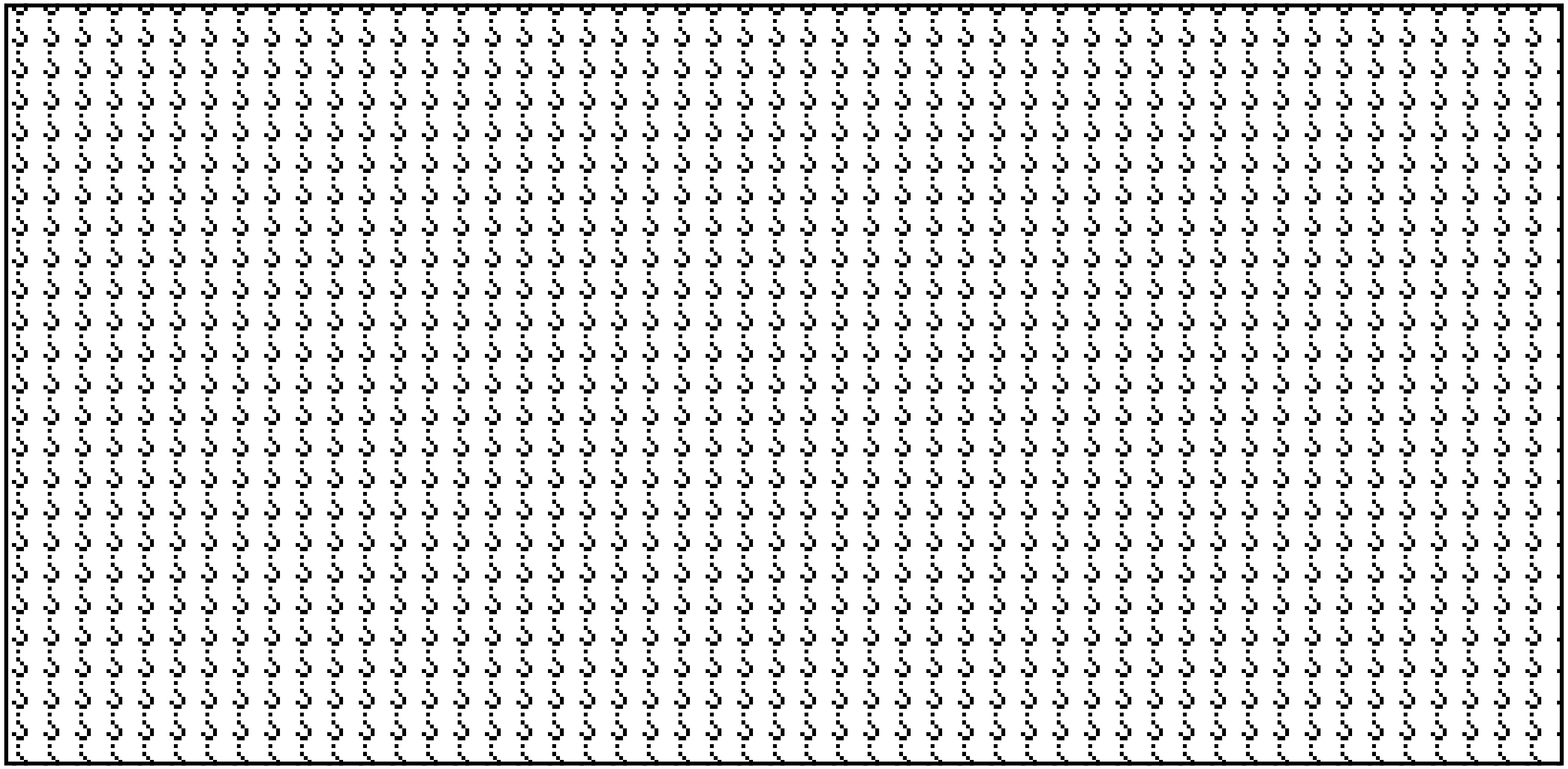


Types of Learning (Cont'd)

- Reinforcement Learning
  - in the case of the agent acts on its environment, it receives some evaluation of its action (reinforcement), but is not told of which action is the correct one to achieve its goal

  - Training data: (S, A, R). (State-Action-Reward)
  - Develop an optimal policy (sequence of decision rules) for the learner so as to maximize its long-term reward.
  - Robotics, Board game playing programs.

RL is learning from interaction



Examples of Reinforcement Learning

- How should a robot behave so as to optimize its "performance"? (Robotics)

- How to automate the motion of a helicopter? (Control Theory)

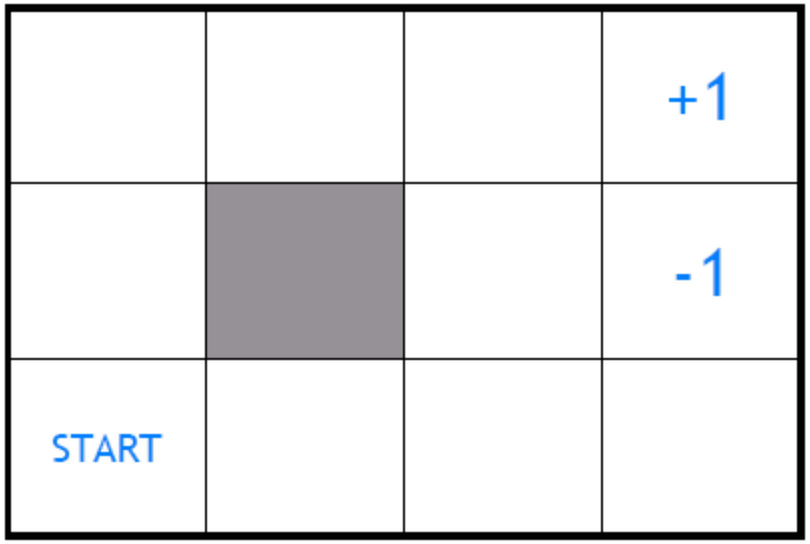
- How to make a good chess-playing program? (Artificial Intelligence)

Autonomous Helicopter

Video:

https://www.youtube.com/watch?v=VCdxqn0fcnE

Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

80%	move UP
10%	move LEFT
10%	move RIGHT

- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step

- what's the strategy to achieve max reward?
- what if the actions were NOT deterministic?

Pole Balancing

- Task:
  - Move car left/right to keep the pole balanced
- State representation
  - Position and velocity of the car
  - Angle and angular velocity of the pole

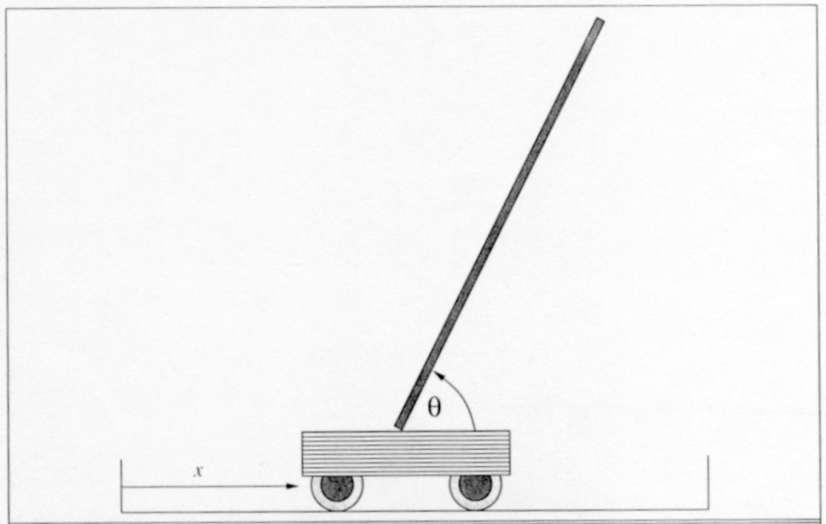


Figure 20.14 Setup for the problem of balancing a long pole on top of a moving cart. The cart can be jerked left or right by a controller that observes x, θ, $\dot{x}$, and $\dot{\theta}$.

History of Reinforcement Learning

- Roots in the psychology of animal learning (Thorndike,1911).
- Another independent thread was the problem of optimal control, and its solution using dynamic programming (Bellman, 1957).
- Idea of temporal difference learning (on-line method), e.g., playing board games (Samuel, 1959).
- A major breakthrough was the discovery of Q-learning (Watkins, 1989).

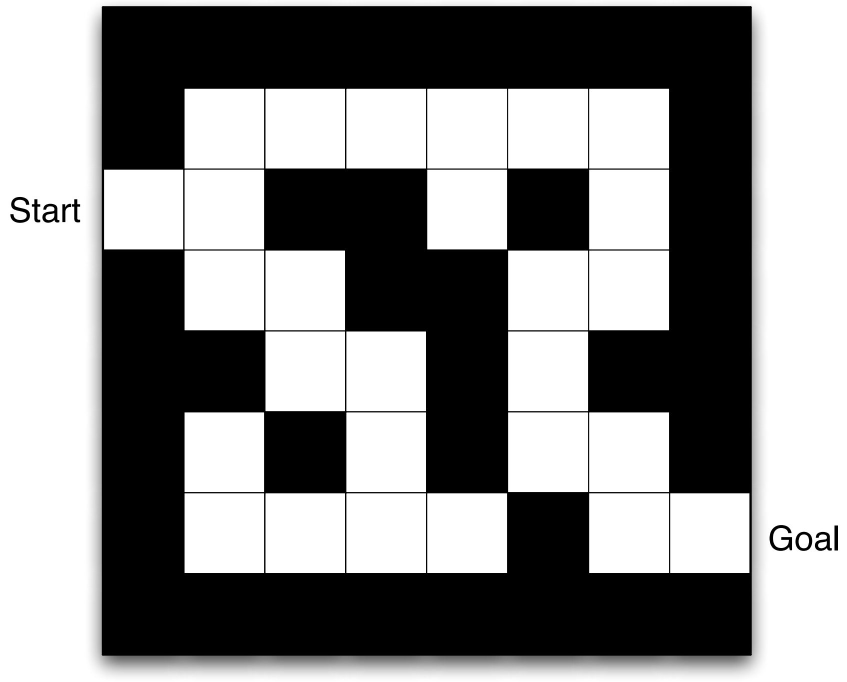
What is special about RL?

- RL is learning how to map states to actions, so as to maximize a numerical reward over time.
- Unlike other forms of learning, it is a multistage decision-making process (often Markovian).
- An RL agent must learn by trial-and-error. (Not entirely supervised, but interactive)
- Actions may affect not only the immediate reward but also subsequent rewards (Delayed effect).

Elements of RL

- A policy
  - A map from state space to action space.
  - May be stochastic.
- A reward function
  - It maps each state (or, state-action pair) to a real number, called reward.
- A value function
  - Value of a state (or, state-action pair) is the total expected reward, starting from that state (or, state-action pair).

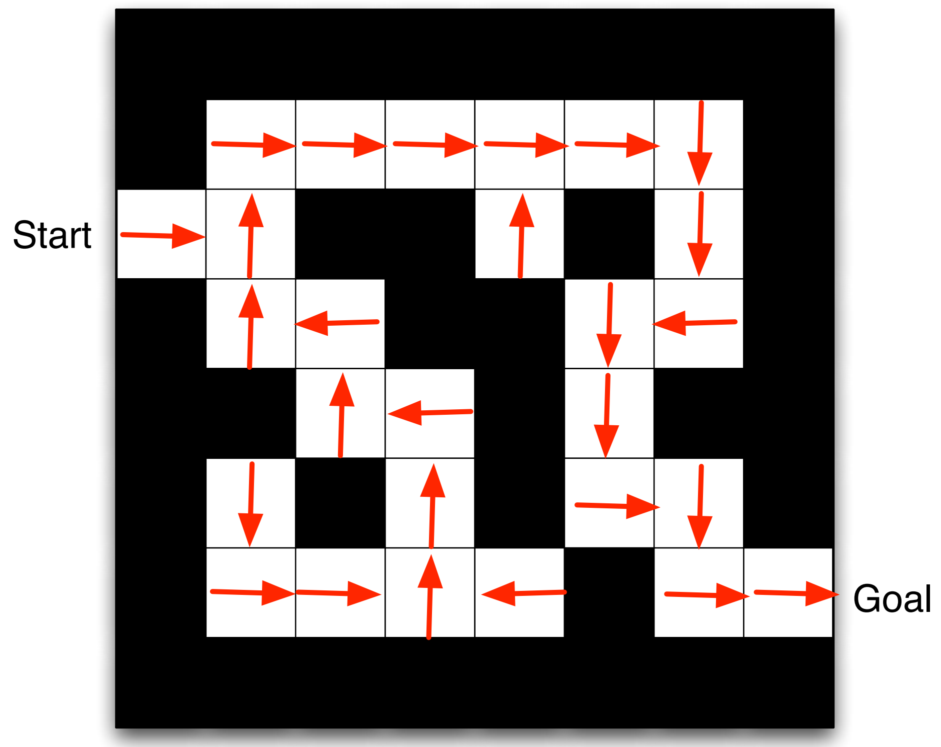
Maze Example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example

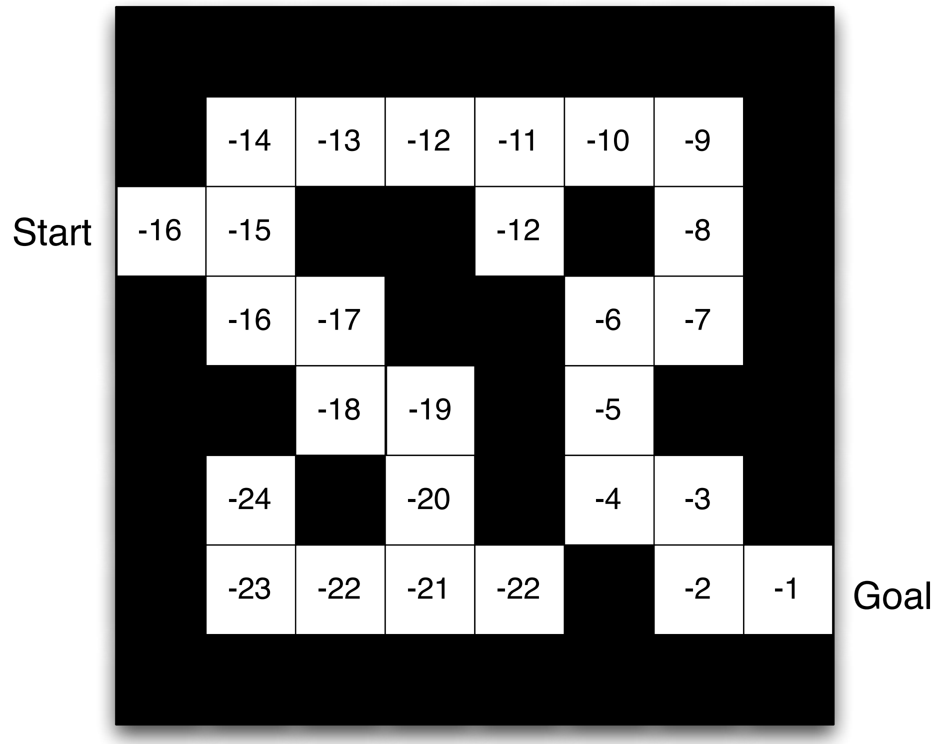
Policy:



- Arrows represent policy $\pi(s)$ for each state s

Maze Example

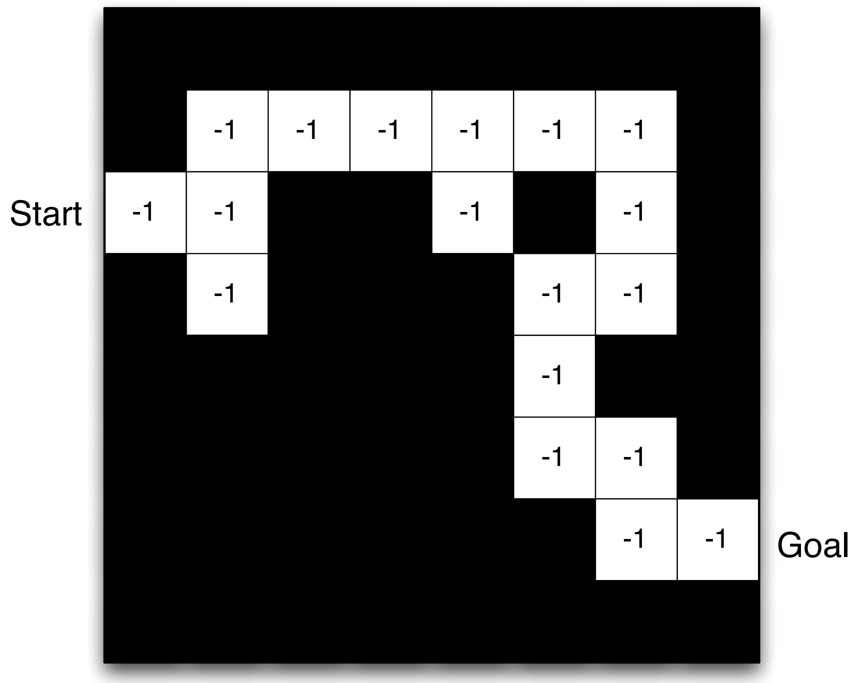
Value Function: (Expected Future Reward)



- Numbers represent value $v_\pi(s)$ of each state s

Maze Example

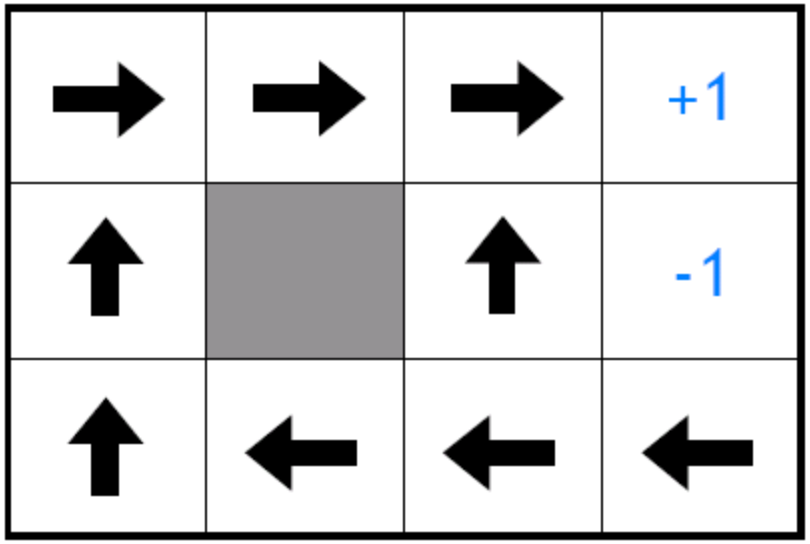
Model:



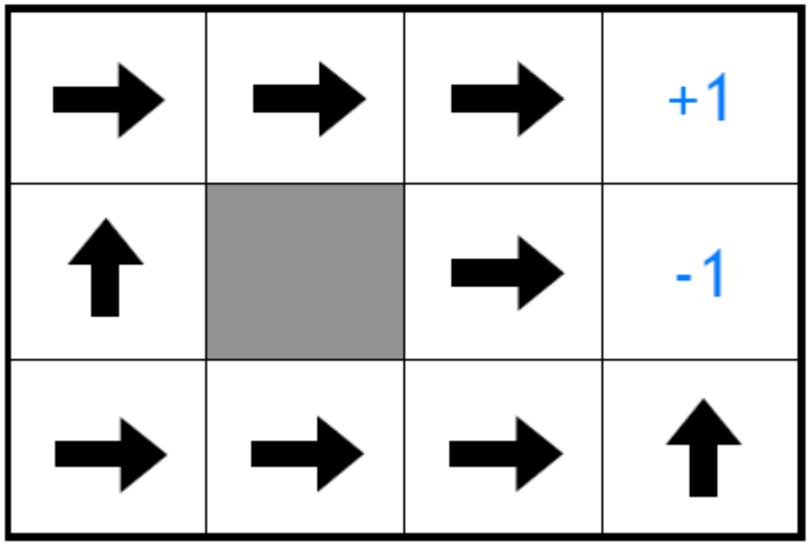
- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

- Grid layout represents transition model $\mathcal{P}_{ss'}^{a}$
- Numbers represent immediate reward $\mathcal{R}_{s}^{a}$ from each state s (same for all a)

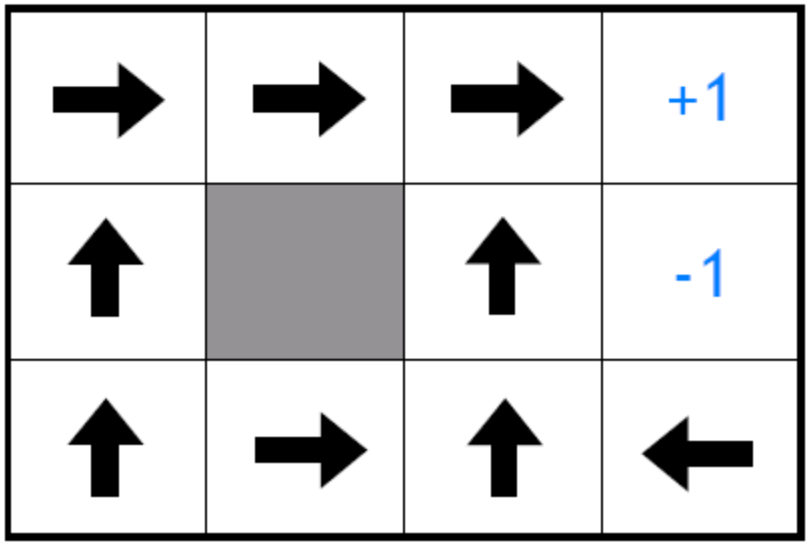
Policy



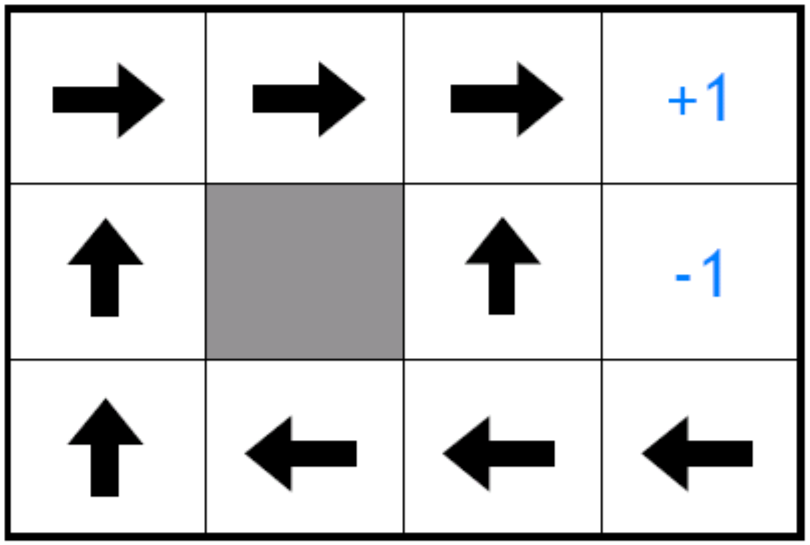
Reward for each step -2



Reward for each step: -0.1



Reward for each step: -0.04



The Precise Goal

- To find a policy that maximizes the Value function.
  - transitions and rewards usually not available

- There are different approaches to achieve this goal in various situations.

- Value iteration and Policy iteration are two more classic approaches to this problem. But essentially both are dynamic programming.

- Q-learning is a more recent approaches to this problem. Essentially it is a temporal-difference method.

MARKOV DECISION PROCESSES

Markov Decision Processes

A Markov decision process is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$ where:

- S is a set of **states**. (For example, in autonomous helicopter flight, S might be the set of all possible positions and orientations of the helicopter.)
- A is a set of **actions**. (For example, the set of all possible directions in which you can push the helicopter's control sticks.)
- P_{sa} are the state transition probabilities. For each state $s \in S$ and action $a \in A$, P_{sa} is a distribution over the state space. We'll say more about this later, but briely, P_{sa} gives the distribution over what states we will transition to if we take action a in state s.
- $\gamma \in [0, 1)$ is called the **discount factor**.
- $R : S \times A \mapsto \mathbb{R}$ is the **reward function**. (Rewards are sometimes also written as a function of a state S only, in which case we would have $R : S \mapsto \mathbb{R}$).

The dynamics of an MDP

- We start in some state s_0, and get to choose some action $a_0 \in A$
- As a result of our choice, the state of the MDP randomly transitions to some successor state s_1, drawn according to $s_1 \sim P_{s0a0}$
- Then, we get to pick another action a_1
- …

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$$

The dynamics of an MDP, (Cont'd)

- Upon visiting the sequence of states $s_0, s_1, \ldots$, with actions $a_0, a_1, \ldots$, our total payoff is given by

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots$$

- Or, when we are writing rewards as a function of the states only, this becomes

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

  - For most of our development, we will use the simpler state-rewards R(s), though the generalization to state-action rewards R(s; a) offers no special diffculties.

- Our goal in reinforcement learning is to choose actions over time so as to maximize the expected value of the total payoff:

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots\right]$$

FIXED POINT ITERATION

Fixed Point Iteration for Optimization

- Fixed point iteration is a general tool for solving systems of equations
- It can also be applied to optimization.

$$J(\boldsymbol{\theta})$$

$$\frac{dJ(\boldsymbol{\theta})}{d\theta_i} = 0 = f(\boldsymbol{\theta})$$

$$0 = f(\boldsymbol{\theta}) \Rightarrow \theta_i = g(\boldsymbol{\theta})$$

$$\theta_i^{(t+1)} = g(\boldsymbol{\theta}^{(t)})$$

1. Given objective function:
2. Compute derivative, set to zero (call this function f).
3. Rearrange the equation s.t. one of parameters appears on the LHS.
4. Initialize the parameters.
5. For i in $\{1,\ldots,K\}$, update each parameter and increment t:
6. Repeat #5 until convergence

Fixed Point Iteration for Optimization

- Fixed point iteration is a general tool for solving systems of equations
- It can also be applied to optimization.

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2+2}{3} = g(x)$$

$$x \leftarrow \frac{x^2+2}{3}$$

1. Given objective function:
2. Compute derivative, set to zero (call this function f).
3. Rearrange the equation s.t. one of parameters appears on the LHS.
4. Initialize the parameters.
5. For i in $\{1,\ldots,K\}$, update each parameter and increment t:
6. Repeat #5 until convergence

Fixed Point Iteration for Optimization

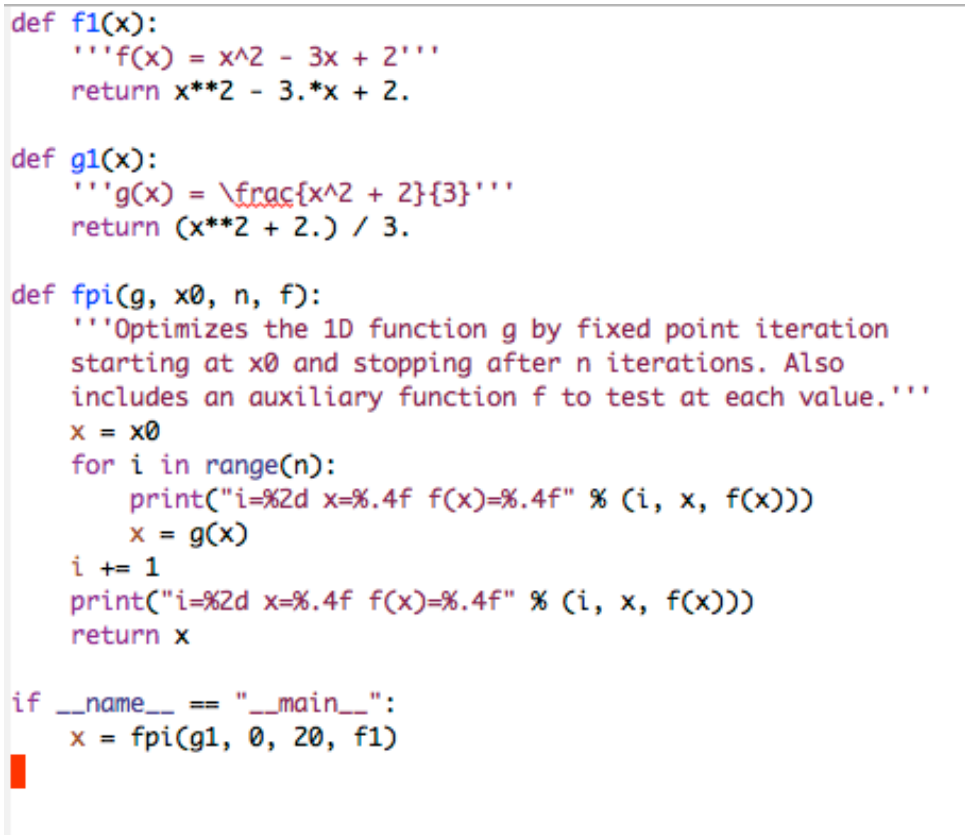
We can implement our example in a few lines of python.

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2+2}{3} = g(x)$$

$$x \leftarrow \frac{x^2+2}{3}$$



```
def f1(x):
    '''f(x) = x^2 - 3x + 2'''
    return x**2 - 3.*x + 2.

def g1(x):
    '''g(x) = \frac{x^2 + 2}{3}'''
    return (x**2 + 2.) / 3.

def fpi(g, x0, n, f):
    '''Optimizes the 1D function g by fixed point iteration
    starting at x0 and stopping after n iterations. Also
    includes an auxiliary function f to test at each value.'''
    x = x0
    for i in range(n):
        print("i=%2d x=%.4f f(x)=%.4f" % (i, x, f(x)))
        x = g(x)
    i += 1
    print("i=%2d x=%.4f f(x)=%.4f" % (i, x, f(x)))
    return x

if __name__ == "__main__":
    x = fpi(g1, 0, 20, f1)
```

Fixed Point Iteration for Optimization

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2 + 2}{3} = g(x)$$

$$x \leftarrow \frac{x^2 + 2}{3}$$

```
$ python fixed-point-iteration.py
i= 0 x=0.0000 f(x)=2.0000
i= 1 x=0.6667 f(x)=0.4444
i= 2 x=0.8148 f(x)=0.2195
i= 3 x=0.8880 f(x)=0.1246
i= 4 x=0.9295 f(x)=0.0755
i= 5 x=0.9547 f(x)=0.0474
i= 6 x=0.9705 f(x)=0.0304
i= 7 x=0.9806 f(x)=0.0198
i= 8 x=0.9872 f(x)=0.0130
i= 9 x=0.9915 f(x)=0.0086
i=10 x=0.9944 f(x)=0.0057
i=11 x=0.9963 f(x)=0.0038
i=12 x=0.9975 f(x)=0.0025
i=13 x=0.9983 f(x)=0.0017
i=14 x=0.9989 f(x)=0.0011
i=15 x=0.9993 f(x)=0.0007
i=16 x=0.9995 f(x)=0.0005
i=17 x=0.9997 f(x)=0.0003
i=18 x=0.9998 f(x)=0.0002
i=19 x=0.9999 f(x)=0.0001
i=20 x=0.9999 f(x)=0.0001
```

VALUE ITERATION

Elements of RL

- A policy
  - A map from state space to action space.
  - May be stochastic.
- A reward function
  - It maps each state (or, state-action pair) to a real number, called reward.
- A value function
  - Value of a state (or, state-action pair) is the total expected reward, starting from that state (or, state-action pair).

Policy

- A policy is any function $\pi \ : \ S \mapsto A$ mapping from the states to the actions.

- We say that we are executing some policy if, whenever we are in state s, we take action $a = \pi(s)$.

- We also define the value function for a policy π according to

$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots \mid s_0 = s, \pi\right]$$

  - $V^{\pi}(s)$ is simply the expected sum of discounted rewards upon starting in state s, and taking actions according to π.

Value Function

- Given a fixed policy π, its value function V^π satisfies the **Bellman equations**:

$$V^\pi(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \gamma \underbrace{\sum_{s' \in \mathcal{S}} P_{s\pi(s)}(s') V^\pi(s')}_{\text{expected sum of future discounted rewards}}$$

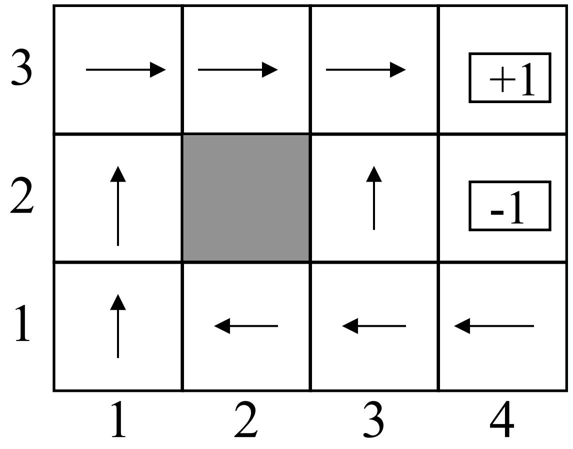
- Bellman's equations can be used to efficiently solve for V^π (see later)

The Grid world

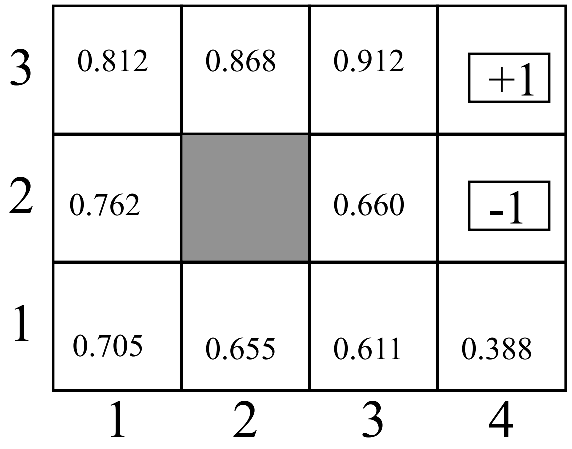
M = 0.8 in direction you want to go
0.2 in perpendicular $<$ 0.1 left / 0.1 right

Policy: mapping from states to actions

An optimal policy for the stochastic environment:



utilities of states:



	1	2	3	4
3	0.812	0.868	0.912	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388

Environment $<$ Observable (accessible): percept identifies the state / Partially observable

Markov property: Transition probabilities depend on state only, not on the path to the state.
Markov decision problem (MDP).
Partially observable MDP (POMDP): percepts does not have enough info to identify transition probabilities

Optimal value function

- We define the optimal value function according to

$$V^*(s) = \max_\pi V^\pi(s) \qquad (1)$$

  - In other words, this is the best possible expected sum of discounted rewards that can be attained using any policy

- There is a version of Bellman's equations for the optimal value function:

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s') \qquad (2)$$

  - Why?

Optimal policy

- We also define a policy : $\pi^* : S \mapsto A$ as follows:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \qquad (3)$$

- Fact:
  - $$V^*(s) = V^{\pi^*}(s) \geq V^\pi(s)$$
  - Policy π^* has the interesting property that it is the optimal policy for all states s.
    - It is not the case that if we were starting in some state s then there'd be some optimal policy for that state, and if we were starting in some other state s_0 then there'd be some other policy that's optimal policy for s_0.
    - The same policy π^* attains the maximum above for all states s. This means that we can use the same policy no matter what the initial state of our MDP is.

The Basic Setting for Learning

- Training data: n finite horizon trajectories, of the form $\{s_0, a_0, r_0, \ldots, s_T, a_T, r_T, s_{T+1}\}$.
- Deterministic or stochastic policy: A sequence of decision rules $\{\pi_0, \pi_1, \ldots, \pi_T\}$.
- Each π maps from the observable history (states and actions) to the action space at that time point.

Algorithm 1: Value iteration

- Consider only MDPs with finite state and action spaces $(|S| < \infty, \ |A| < \infty)$
- The value iteration algorithm:

```
1. For each state $s$, initialize $V(s) := 0$.
2. Repeat until convergence {
       • For every state, update
         $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$.
   }
```

  - **synchronous** update
  - **asynchronous** updates
- It can be shown that value iteration will cause V to converge to V^*. Having found V^* , we can then use Equation (3) to find the optimal policy.

Algorithm 2: Policy iteration

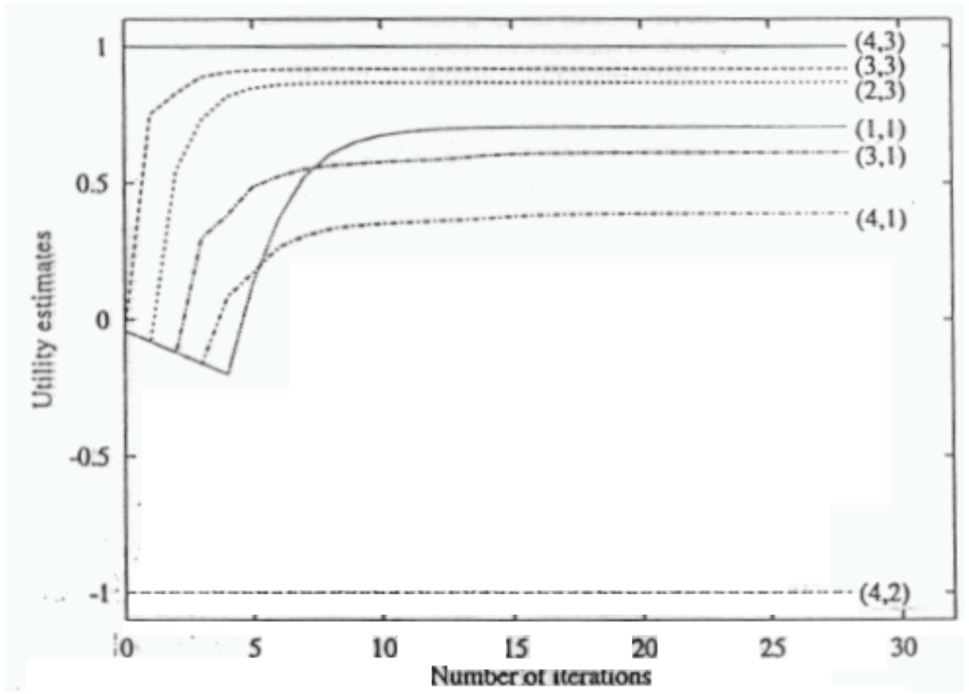
- The policy iteration algorithm:

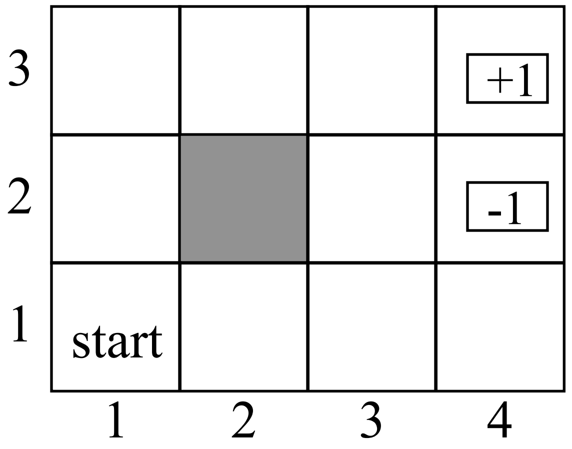
```
1. Initialize $\pi$ randomly.
2. Repeat until convergence {
       • Let $V := V^{\pi}$
       • For each state $s$, let $\pi(s) := \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$.
   }
```

- The inner-loop repeatedly computes the value function for the current policy, and then updates the policy using the current value function.
- Greedy update
- After at most a finite number of iterations of this algorithm, V will converge to V^* , and π will converge to π^*.

Convergence

- The utility values for selected states at each iteration step in the application of VALUE-ITERATION to the 4x3 world in our example





Thrm: As t→∞, value iteration converges to exact U even if updates are done asynchronously & *i* is picked randomly at every step.

Convergence

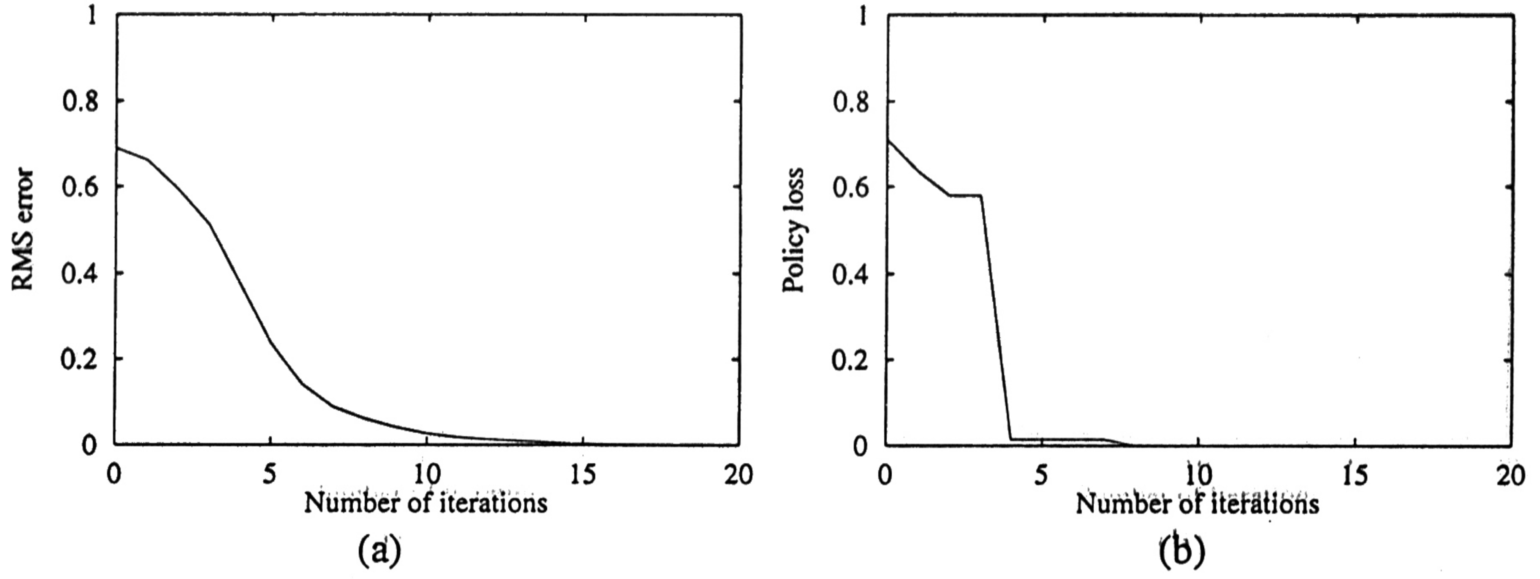


Figure 17.6 (a) The RMS (root mean square) error of the utility estimates compared to the correct values, as a function of iteration number during value iteration. (b) The expected policy loss compared to the optimal policy.

When to stop value iteration?

Q-LEARNING

Q learning

- Define Q-value function

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

- Q-value function updating rule
  - See subsequent slides

- Key idea of TD-Q learning
  - Combined with temporal difference approach

- Rule to chose the action to take

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

Algorithm 3: Q learning

```
For each pair (s, a), initialize Q(s,a)
Observe the current state s
Loop forever
{
    Select an action a (optionally with ε-exploration) and execute it
        $a = \arg\max_a Q(s,a)$
    Receive immediate reward r and observe the new state s'
    Update Q(s,a)
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    s=s'
}
```

Exploration

- Tradeoff between exploitation (control) and exploration (identification)

- Extremes: greedy vs. random acting

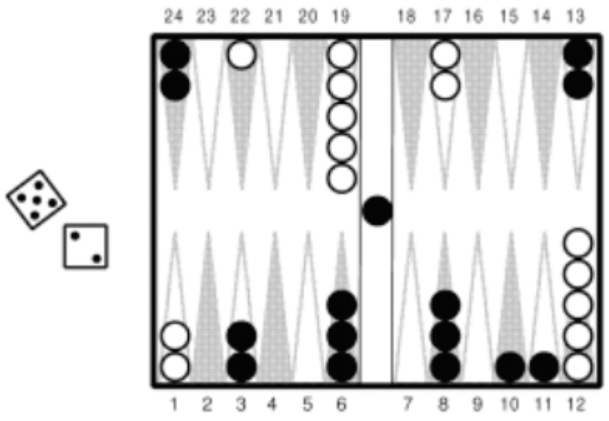
  (n-armed bandit models)

Q-learning converges to optimal Q-values if

- Every state is visited infinitely often (due to exploration),
- The action selection becomes greedy as time approaches infinity, and
- The learning rate a is decreased fast enough but not too fast (as we discussed in TD learning)

RL EXAMPLES

A Success Story



- TD Gammon (Tesauro, G., 1992)

  - A Backgammon playing program.

  - Application of temporal difference learning.

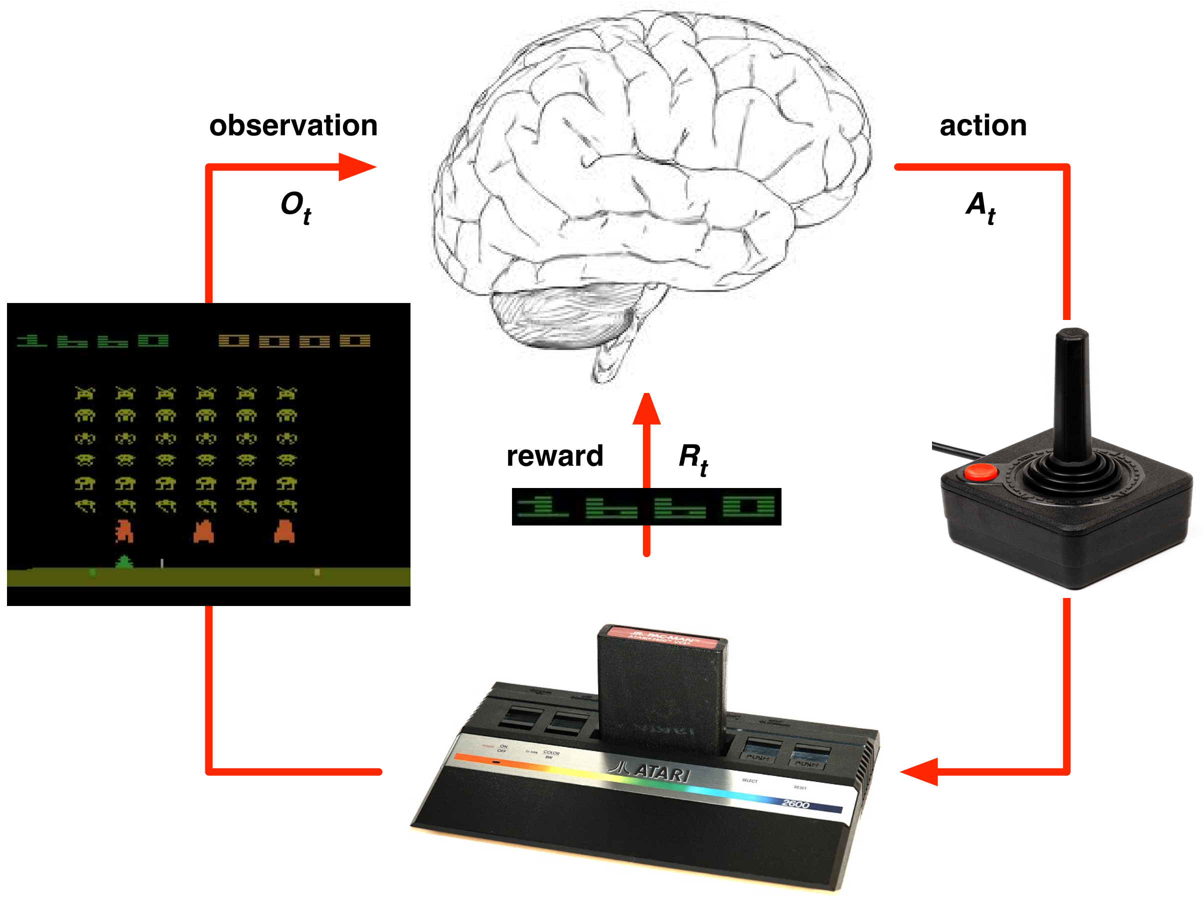
  - The basic learner is a neural network.

  - It trained itself to the world class level by playing against itself and learning from the outcome. So smart!!

  - More information: http://www.research.ibm.com/massive/tdl.html

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

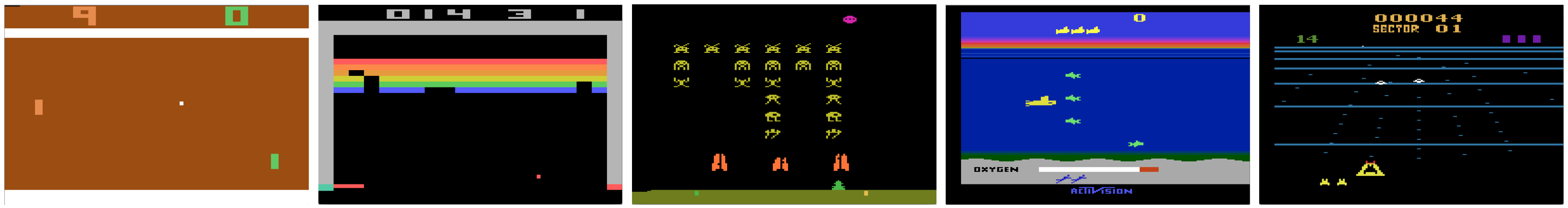


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Videos:

- Atari: 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

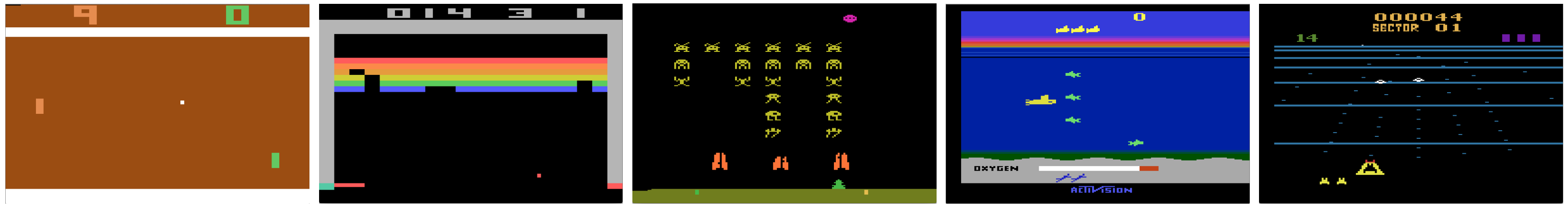


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

	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)

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

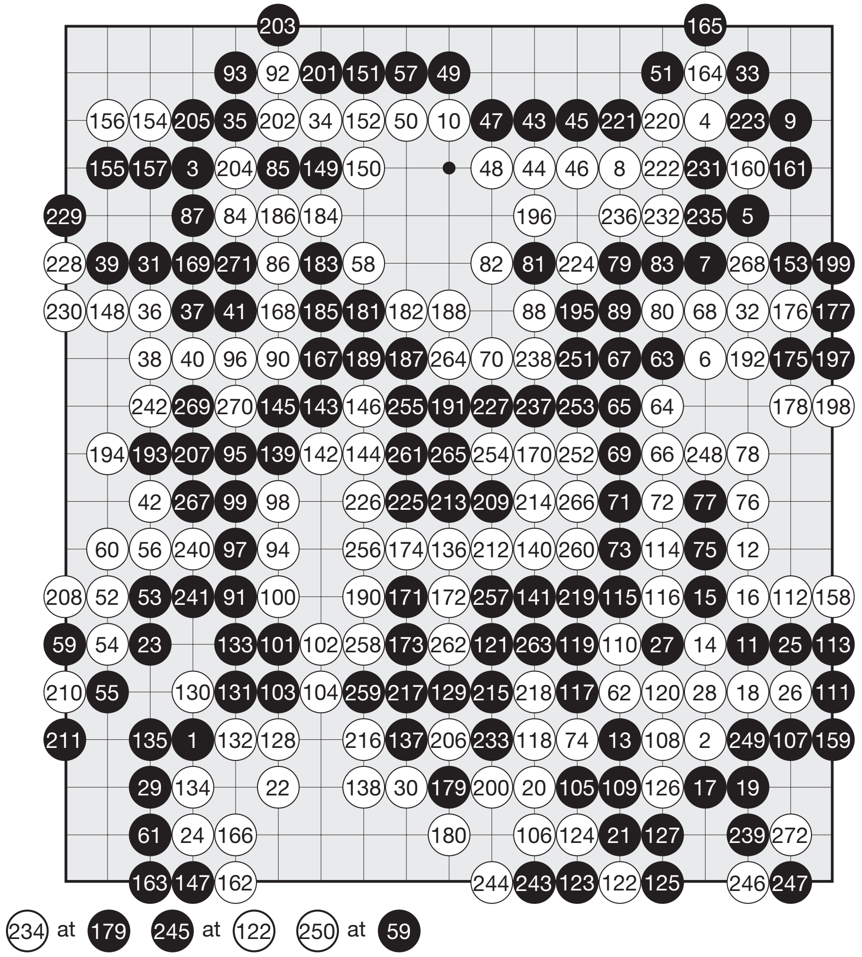


Figure from Silver et al. (2016)

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

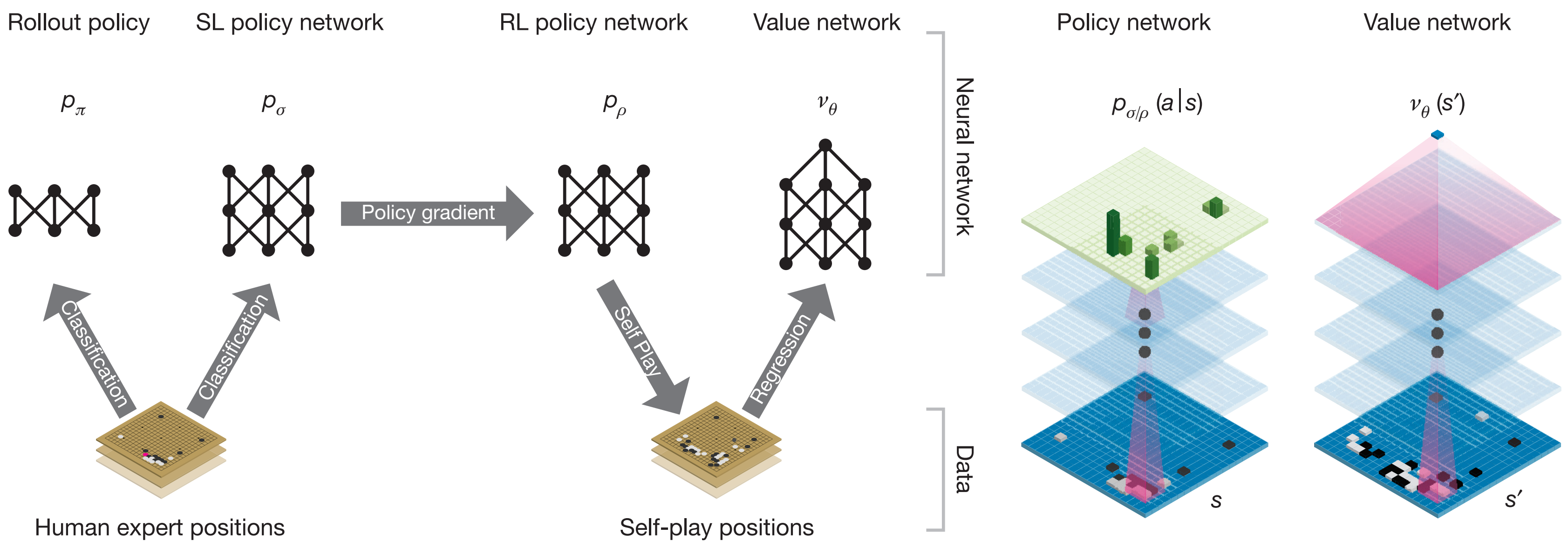


Figure from Silver et al. (2016)

Alpha Go

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

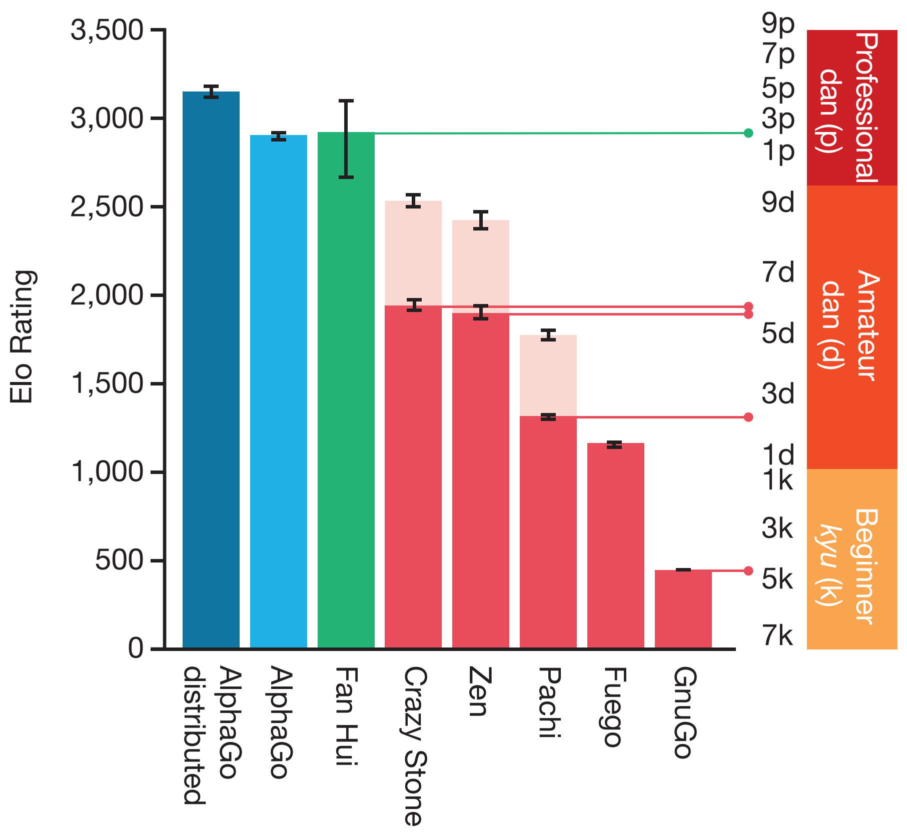


Figure from Silver et al. (2016)

SUMMARY

Summary

- Both value iteration and policy iteration are standard algorithms for solving MDPs, and there isn't currently universal agreement over which algorithm is better.
- For small MDPs, value iteration is often very fast and converges with very few iterations. However, for MDPs with large state spaces, solving for V explicitly would involve solving a large system of linear equations, and could be difficult.
- In these problems, policy iteration may be preferred. In practice value iteration seems to be used more often than policy iteration.
- Q-learning is model-free, and explore the temporal difference

Types of Learning

- Supervised Learning
  - Training data: (X,Y). (features, label)
  - Predict Y, minimizing some loss.
  - Regression, Classification.

- Unsupervised Learning
  - Training data: X. (features only)
  - Find "similar" points in high-dim X-space.
  - Clustering.

- Reinforcement Learning
  - Training data: (S, A, R). (State-Action-Reward)
  - Develop an optimal policy (sequence of decision rules) for the learner so as to maximize its long-term reward.
  - Robotics, Board game playing programs