Lecture 1: Introduction

Hado van Hasselt

Admin

- Lectures on RL mostly 9am-11am on Thursdays
- Backgroud material:
  - An Introduction to Reinforcement Learning, Sutton & Barto, 2017
  - `http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html`

Outline

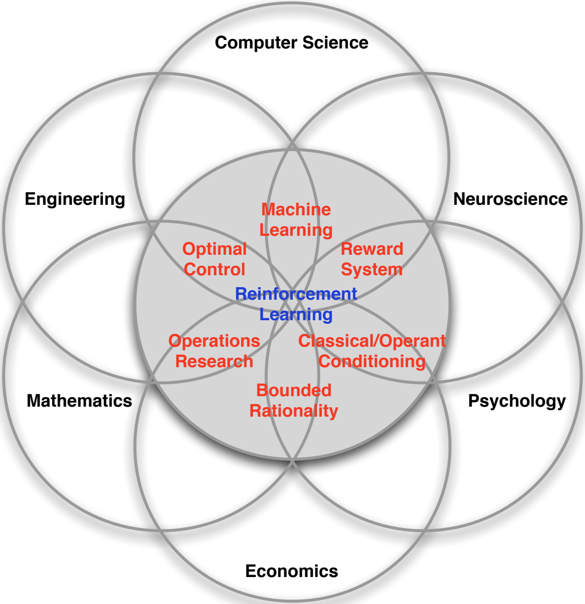
What is reinforcement learning?

What is Reinforcement Learning?

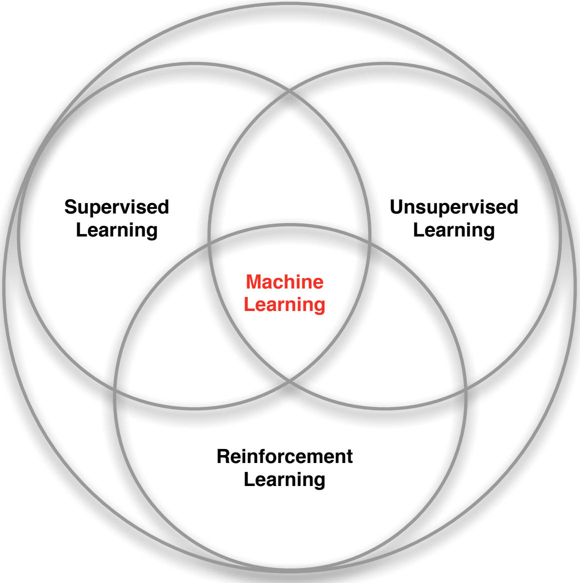
- Science of learning to make decisions, from experience
- This requires us to think about
  - ...predicting (long-term) consequences of actions
  - ...time
  - ...gathering experience
  - ...dealing with uncertainty
- Huge potential applicability

$$\text{RL} = \text{AI?}$$

Many Faces of Reinforcement Learning



Branches of Machine Learning



Characteristics of Reinforcement Learning

How does reinforcement learning differ from other machine learning paradigms?

- No supervision, only a reward signal
- Feedback can be delayed, not instantaneous
- Time really matters (sequential, non-i.i.d data)
- Agent's actions affect the subsequent data it receives
- Can be considered more general than supervised learning

Motivation

- First, we started automating physical solutions
  - Industrial revolution (1750 - 1850) and Machine Age (1870 - 1940)
- Second, we started automating repetitive mental solutions
  - Digital revolution (1960 - now) and Information Age
- Next step: allow machines to find solutions themselves
  - AI revolution (now - ????)
- This requires learning how to make decisions

Examples of decision problems

- Examples:
  - Fly stunt manoeuvres in a helicopter
  - Manage an investment portfolio
  - Control a power station
  - Make a robot walk
  - Playing Atari games better than humans
  - Defeat the world champion at Go
- The RL framework applies to all such problems
  (Whether you use classical RL algorithms or not)

Atari

Core concepts

Rewards

- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximize cumulative reward

$$R_{t+1} + R_{t+2} + R_{t+3} + \ldots$$

Reinforcement learning is based on the reward hypothesis

Definition (Reward Hypothesis)

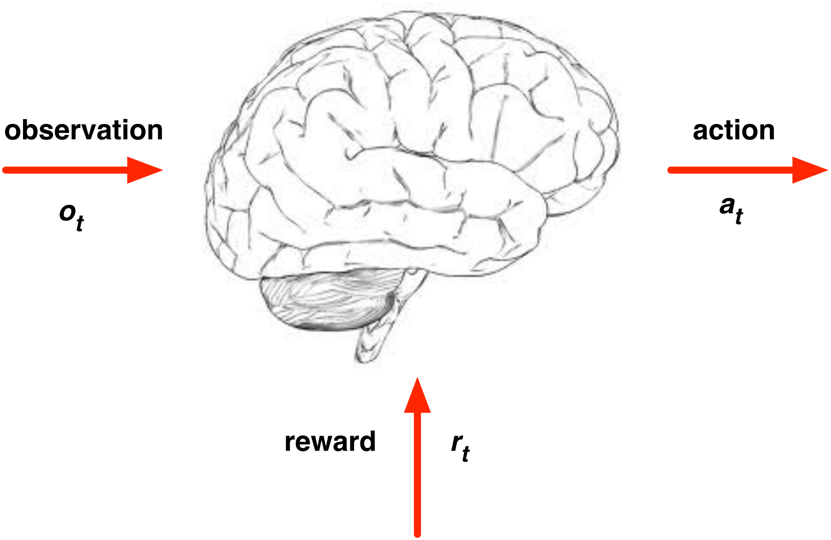
Any goal can be formalized as the outcome of maximizing a cumulative reward

Do you agree?

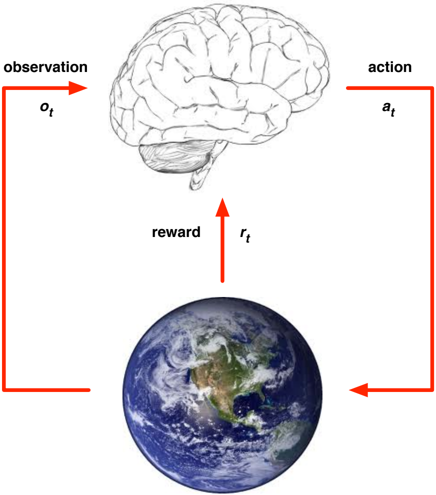
Sequential Decision Making

- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - A financial investment (may take months to mature)
  - Refueling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

Agent and Environment



Agent and Environment



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_t
  - Emits scalar reward R_t
- Rewards could be intrinsic
  (The agent defines its goals)

History and State

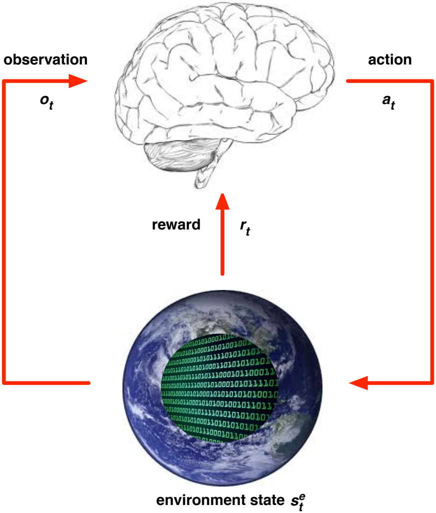
- A history is a sequence of observations, actions, rewards

$$H_t = O_0, A_0, R_1, O_1, ..., O_{t-1}, A_{t-1}, R_t, O_t$$

- i.e. the sensorimotor stream of a robot
- State is the information used to determine what happens next
- Formally, state is a function of the history:

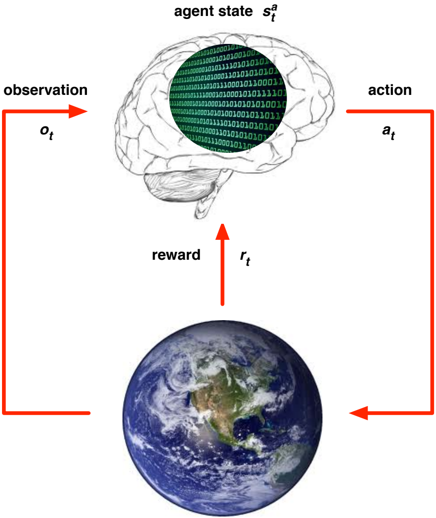
$$S_t = f(H_t)$$

Environment State



- The environment state S_t^e is the environment's private representation
- i.e. whatever data the environment uses to pick the next observation/reward
- The environment state is not usually visible to the agent
- Even if S_t^e is visible, it may contain irrelevant information

Agent State



- The agent state is the agent's internal representation
- i.e. whatever information the agent uses to pick the next action
- i.e. it is the information used by reinforcement learning algorithms
- It can be any function of history:

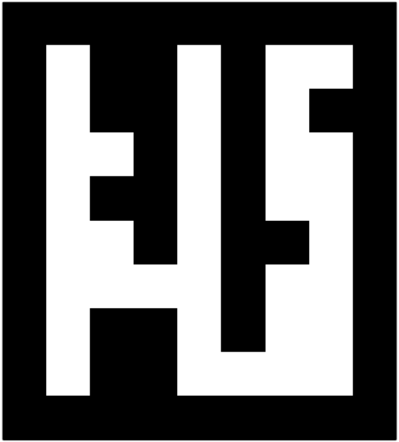
$$S_t^a = f(H_t)$$

Agent State

- The agent state is typically smaller than the environment state
- Need to consider carefully how to construct

Agent State

The full state of a maze

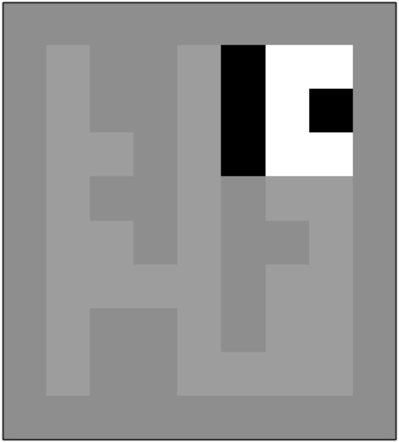


Agent State

A limited view (a potential observation)

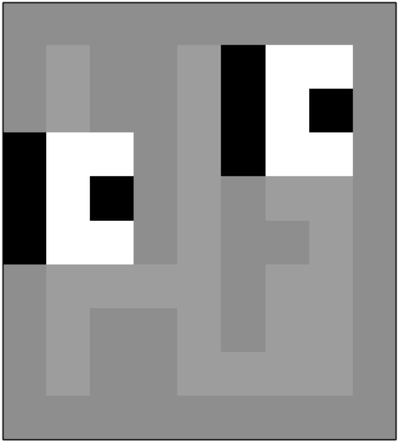
Agent State

Same view in a different location



Agent State

The two observations are indistinguishable



Information State

An information state (a.k.a. Markov state) contains all useful information from the history.

Definition

A state S_t is Markov if and only if

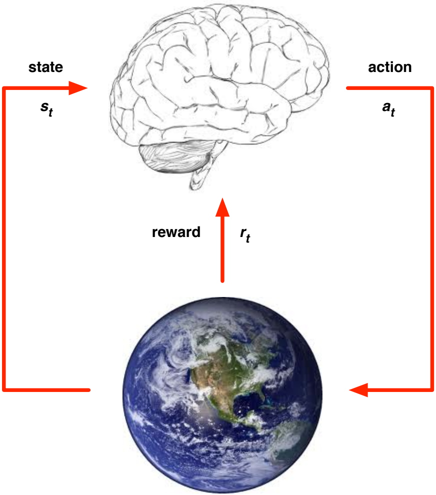
$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- "The future is independent of the past given the present"

$$H_t \rightarrow S_t \rightarrow H_{t+1}$$

- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
- The environment state S_t^e is Markov
- The history H_t is Markov

Fully Observable Environments



Full observability: agent directly observes environment state

$$O_t = S^a_t = S^e_t$$

- Agent state = environment state = information state
- This is a Markov decision process (MDP)
- A useful mathematical framework to reason about sequential decisions

Partially Observable Environments

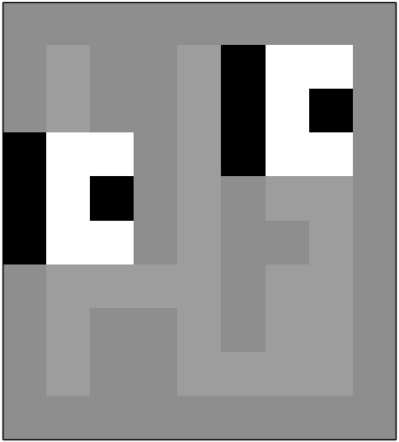
- Formally, MDPs fulfill

$$\mathbb{P}[S_{t+1}, R_{t+1} \mid S_t, A_t] = \mathbb{P}[S_{t+1}, R_{t+1} \mid H_t, A_t]$$

- Partial observability: agent gets partial information
  - A robot with camera vision isn't told its absolute location
  - A poker playing agent only observes public cards
- Now observation is not Markov
- Formally this is a partially observable Markov decision process (POMDP)

Agent State

These two states are not Markov



How would you construct a Markov state?

Partially Observable Environments

- Agent can construct a state representation S_t^a, e.g.
  - Last observation: $S_t^a = O_t$
  - Complete history: $S_t^a = H_t$
  - Beliefs of environment state: $S_t^a = (\mathbb{P}[S_t^e = S^1], ..., \mathbb{P}[S_t^e = S^n])$
  - Recurrent neural network: $S_t^a = f(S_{t-1}^a, O_t)$
- Constructing a Markov agent state may not be feasible
- This is the common case!
- It is more important thing is that the state carries sufficient information to define a good policy, or make good predictions

Core components

Major Components of an RL Agent

- An agent may include one or more of these components:
  - Policy: agent's behaviour function
  - Value function(s): predictions about the future
    (Typically cumulative reward, but can be more general)
  - Model: agent's representation of the environment

Policy

- A policy is the agent's behaviour
- It is a map from agent state to action
- Deterministic policy: $A = \pi(S)$
- Stochastic policy: $\pi(A|S) = \mathbb{P}[A|S]$

Value Function

- Value function is the expected future reward

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s, \pi\right]$$

- Depends on a policy
- Used to evaluate the goodness/badness of states
- Can be used to select between actions
- $\gamma \in [0, 1]$ is called a discount factor
  - Trades off importance of immediate vs long-term rewards
- If we would know the value of each policy, we could act optimally by selecting the policy with highest value. This has the optimal value

$$v_*(s) = \max_\pi v_\pi(s)$$

Value Functions

- Value function for a policy π has a recursive form:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s, \pi\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma^2 R_{t+3} + ...\right) \mid S_t = s, \pi\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, \pi\right] \end{aligned}$$

- This is known as a Bellman equation (Bellman 1957)
- A similar equation holds for the optimal policy:

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

- We heavily exploit such equalities, and use them to create algorithms
- More in the next lecture

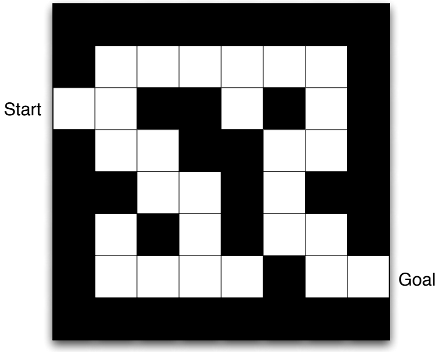
Model

- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}^a_{ss'} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'\right]$$

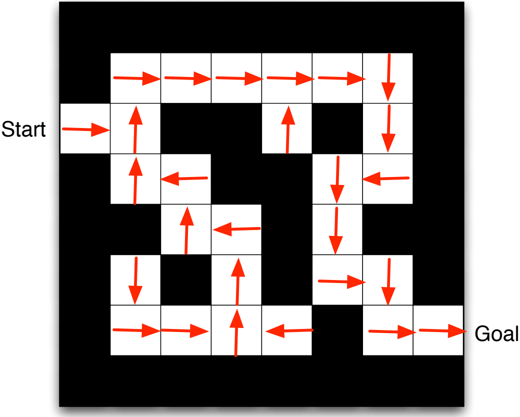
- A model does not immediately give us a good policy - we would still need to plan

Maze Example



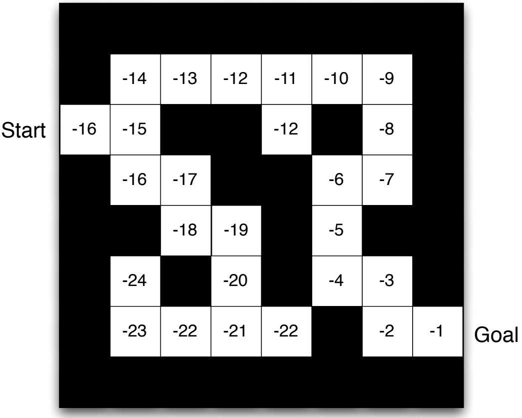
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



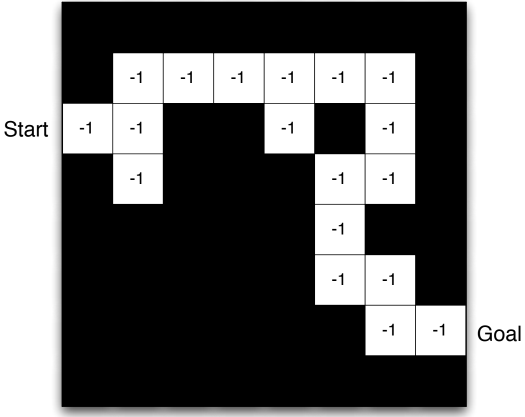
- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



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

Maze Example: Model



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

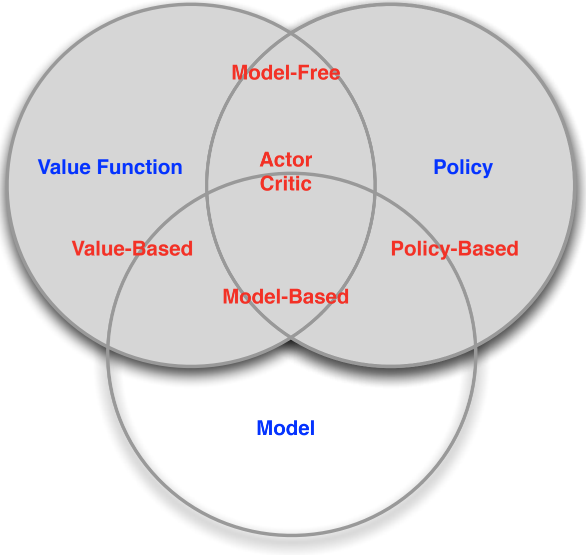
Categorizing RL agents (1)

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function

Categorizing RL agents (2)

- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Optionally Policy and/or Value Function
  - Model

RL Agent Taxonomy



Challenges in reinforcement learning

Learning and Planning

Two fundamental problems in reinforcement learning

- Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
- Planning:
  - A model of the environment is given
  - The agent plans in this model (without external interaction)
  - a.k.a. reasoning, pondering, thought, search, planning

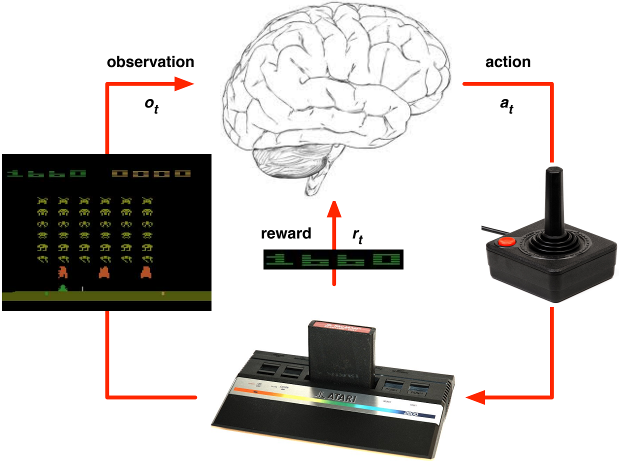
Prediction and Control

- Prediction: evaluate the future
  - Given a policy
- Control: optimize the future
  - Find the best policy
- These are strongly related:

$$\pi_*(s) = \operatorname*{argmax}_{\pi} v_\pi(s)$$

- If we could predict *everything* do we need anything else?

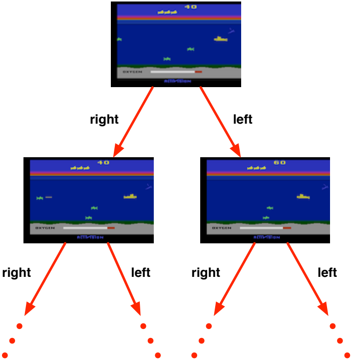
Atari Example: Reinforcement Learning



- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystic see pixels and scores

Atari Example: Planning

- Rules of the game are known
- Can query emulator
  - perfect model inside agent's brain
- If I take action *a* from state *s*:
  - what would the next state be?
  - what would the score be?
- Plan ahead to find optimal policy
  - e.g. tree search



Exploration and Exploitation (1)

- We learn from trial-and-error learning
- The agent should discover a good policy
- ...from its experiences of the environment
- ...without losing too much reward along the way

Exploration and Exploitation (2)

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is important to explore as well as exploit

Examples

- Restaurant Selection

  Exploitation Go to your favourite restaurant
  Exploration Try a new restaurant

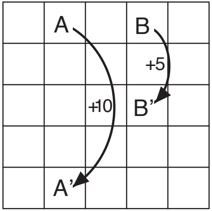
- Oil Drilling

  Exploitation Drill at the best known location
  Exploration Drill at a new location

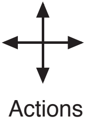
- Game Playing

  Exploitation Play the move you currently believe is best
  Exploration Try a new strategy

Gridworld Example: Prediction



(a)



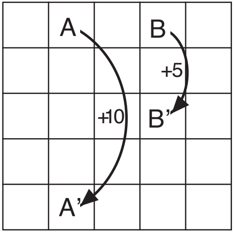
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

Reward is -1 when bumping into a wall, $\gamma = 0.9$

What is the value function for the uniform random policy?

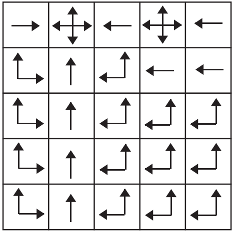
Gridworld Example: Control



a) 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

b) V^*



c) π^*

What is the optimal value function over all possible policies?

What is the optimal policy?

Major Components of an RL Agent

- An agent may include one or more of these components:
  - Policy: $\pi(a|s)$
  - Value function: $v_\pi(s)$, $v_*(s)$, $q_\pi(s, a)$, $q_*(s, a)$
  - Model: $s', r = m(s, a)$
- All of these are functions
- We often represent these as deep neural networks, then use deep learning to optimize these
- However, we also often violate typical assumptions from supervised learning (e.g., i.i.d. data, stationarity)
- Deep reinforcement learning is a rich and active research field

Topics

- Introduction
- Markov decision processes
- Planning by dynamic programming
- Model-free prediction
- Model-free control
- Deep reinforcement learning
- Policy-gradient methods
- Integrating Learning and Planning
- Case study: Deep RL in simulations (Volodymyr Mnih)
- Case study: AlphaGo (David Silver)