UVA CS 4774: Machine Learning

S5: Lecture 26: Reinforcement Learning

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Course Content Plan → Regarding Tasks

- ~~Regression (supervised)~~ ← Y is a continuous
- ~~Learning theory~~ ← About f()
- ~~Classification (supervised)~~ ← Y is a discrete
- ~~Unsupervised models~~ ← NO Y
- ~~Graphical models~~ ← About interactions among Y,X1,. Xp
- Reinforcement Learning ← Learn to Interact with environment

Outline

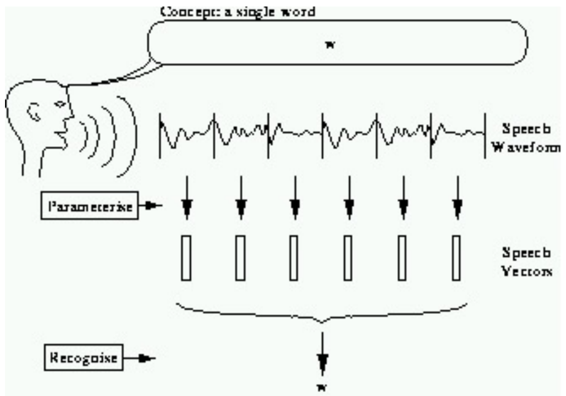
- Examples of RL applications

- Defining an RL problem
  - Markov Decision Processes

- Solving an RL problem
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning

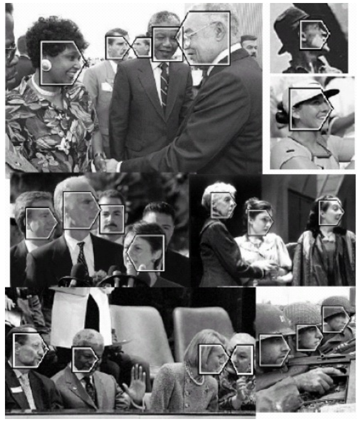
credit: Geoff Hulten

Where Machine Learning is being used or can be useful?

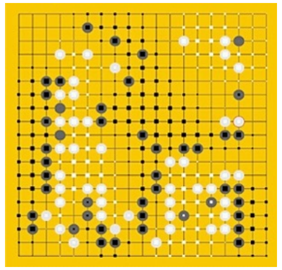


Speech recognition

Information retrieval

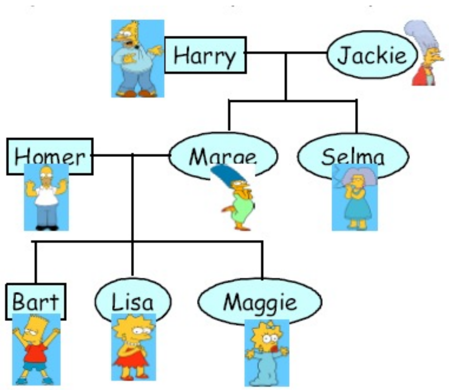


Computer vision

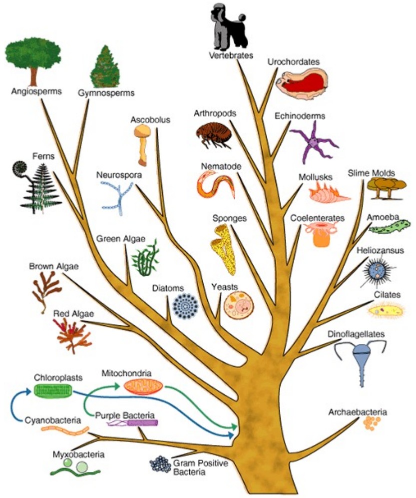


Games

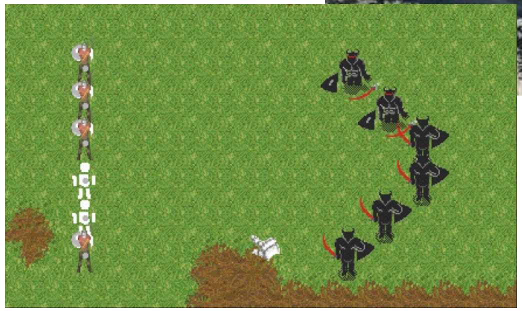
Robotic control



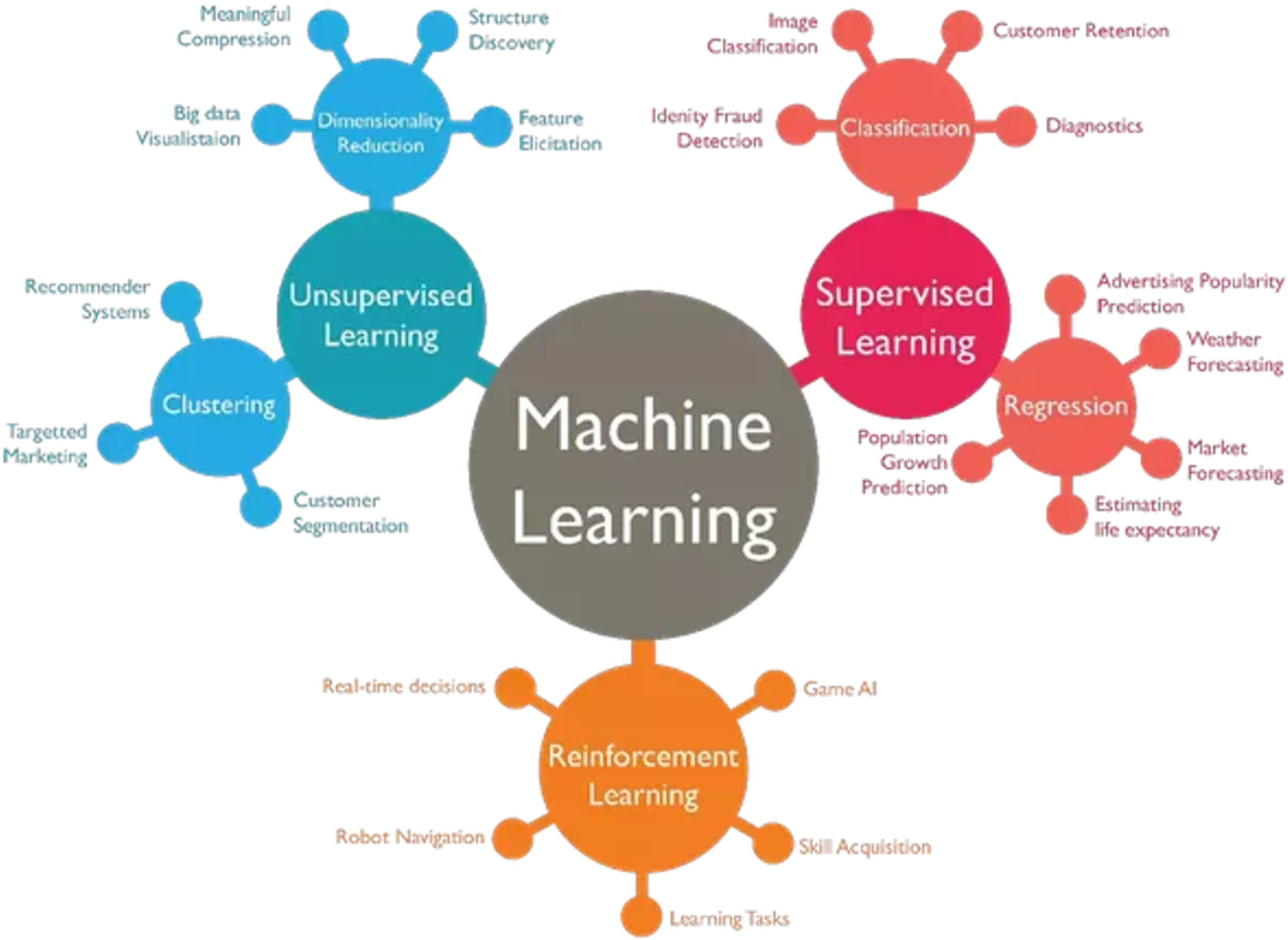
Pedigree



Evolution



Planning


Meaningful Compression
Structure Discovery
Big data Visualistaion
Dimensionality Reduction
Feature Elicitation
Image Classification
Customer Retention
Idenity Fraud Detection
Classification
Diagnostics
Recommender Systems
Unsupervised Learning
Supervised Learning
Advertising Popularity Prediction
Weather Forecasting
Clustering
Regression
Targetted Marketing
Machine Learning
Population Growth Prediction
Market Forecasting
Customer Segmentation
Estimating life expectancy
Real-time decisions
Game AI
Reinforcement Learning
Robot Navigation
Skill Acquisition
Learning Tasks

Classes of Learning Problems

Supervised Learning:

Data: (x, y)

x is data, y is label

Goal: Learn function to map x → y

Example:

This thing is an apple.

Unsupervised Learning:

Data: x

x is data, no labels!

Goal: Learn underlying structure

Example:

This thing is like the other thing.

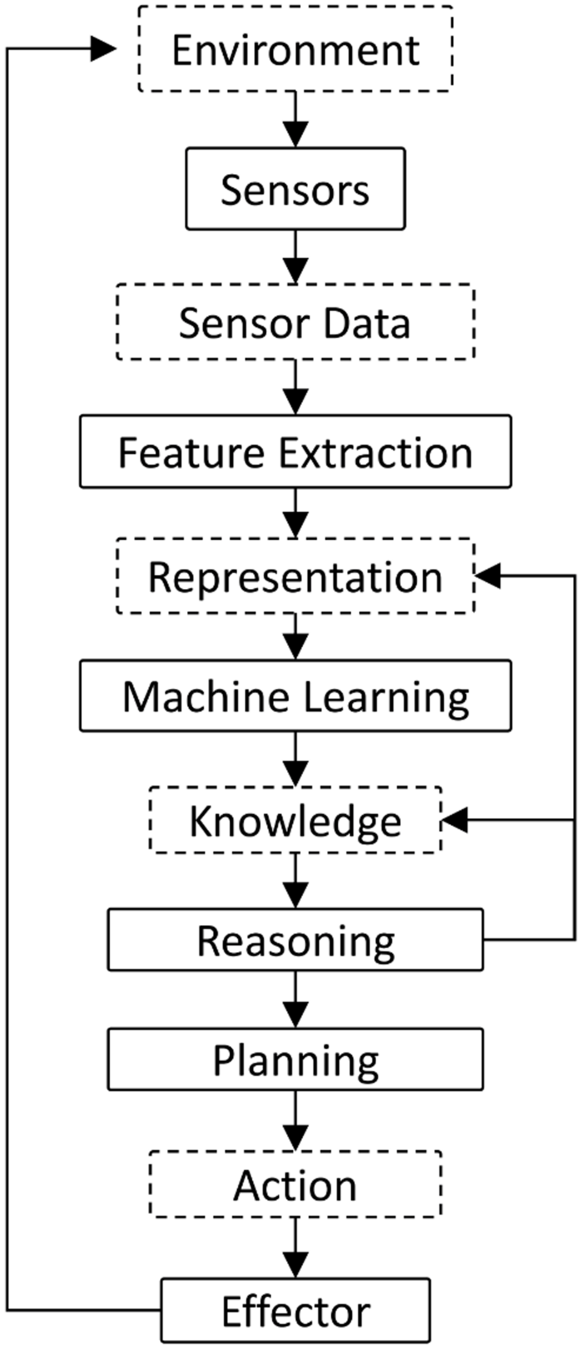
Reinforcement Learning:

Data: state-action pairs

Goal: Maximize future rewards over many steps

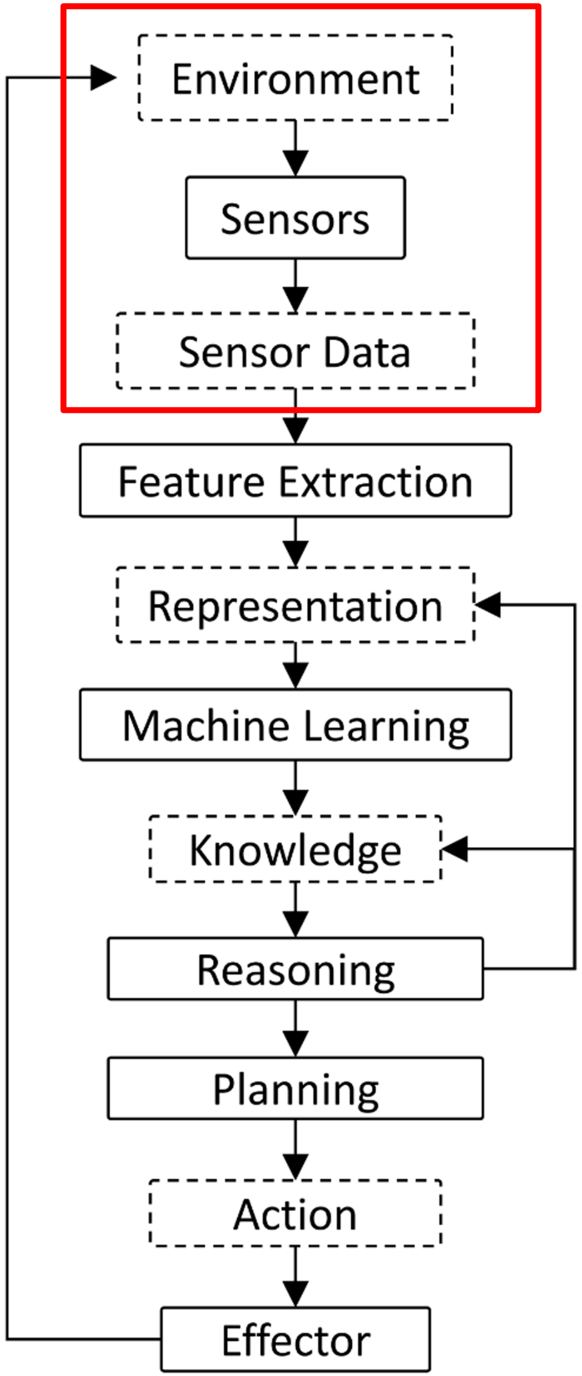
Example:

Eat this thing because it will keep you alive.

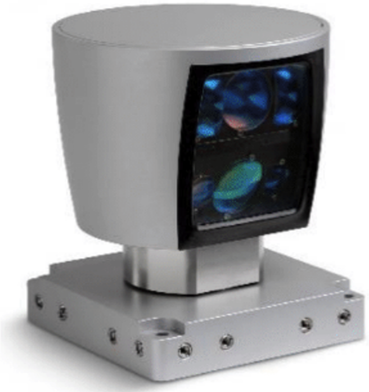


The Machine Learning Stack!

What can be learned?



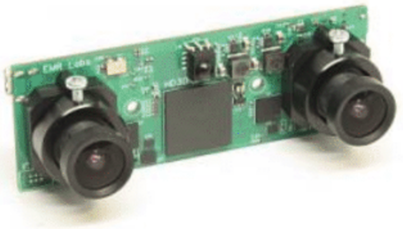
Sensors


Lidar


Camera
(Visible, Infrared)

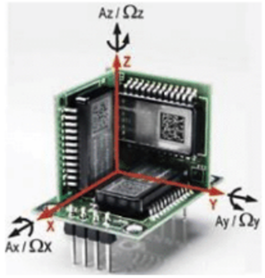

Radar


GPS

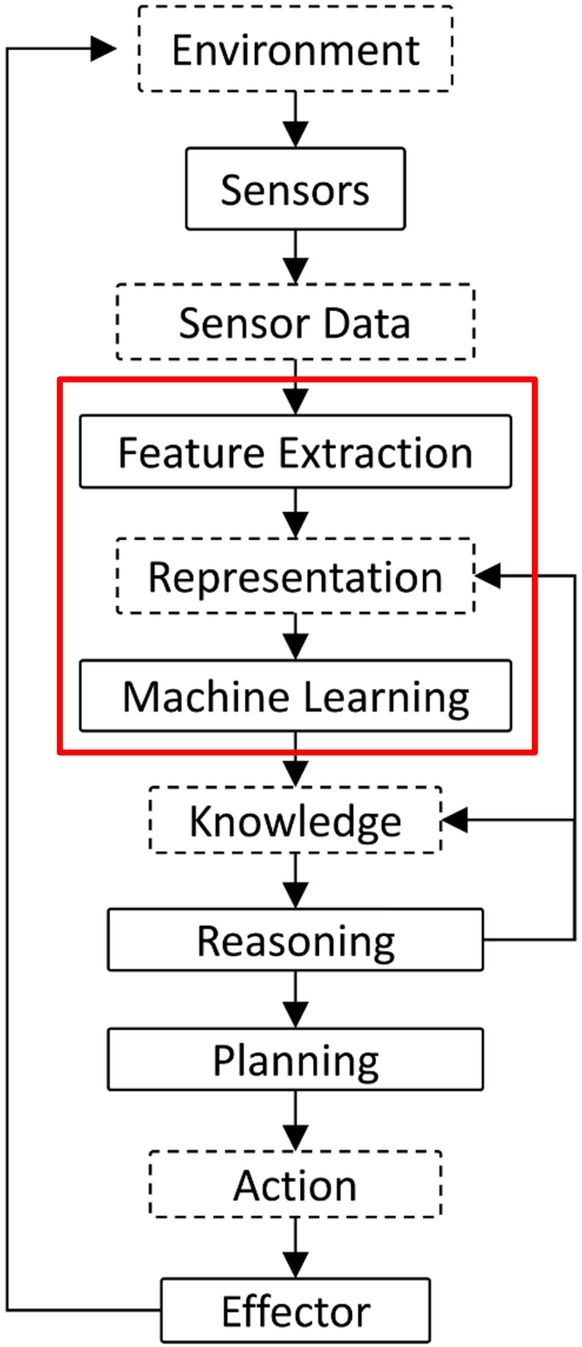

Stereo Camera

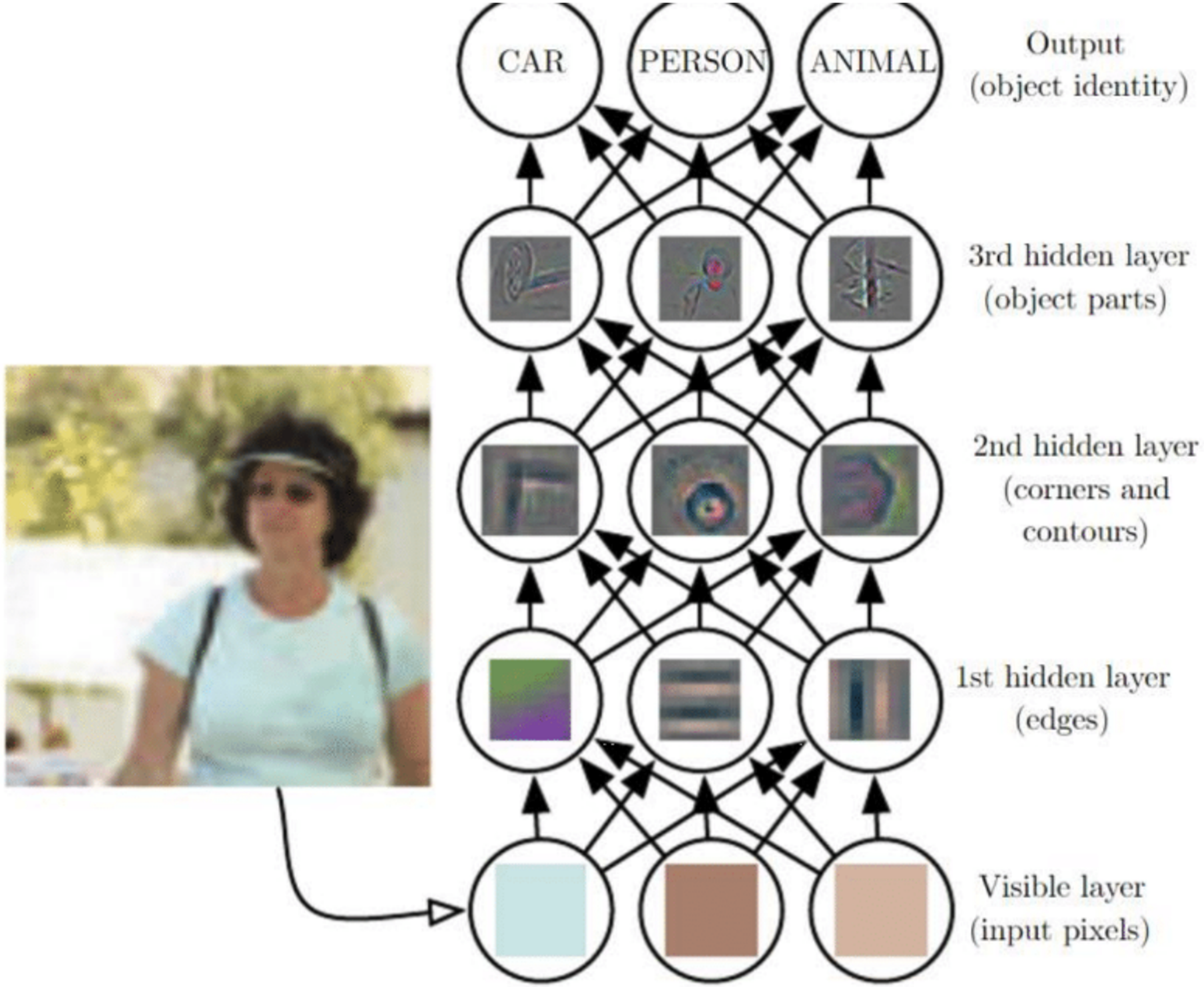

Microphone

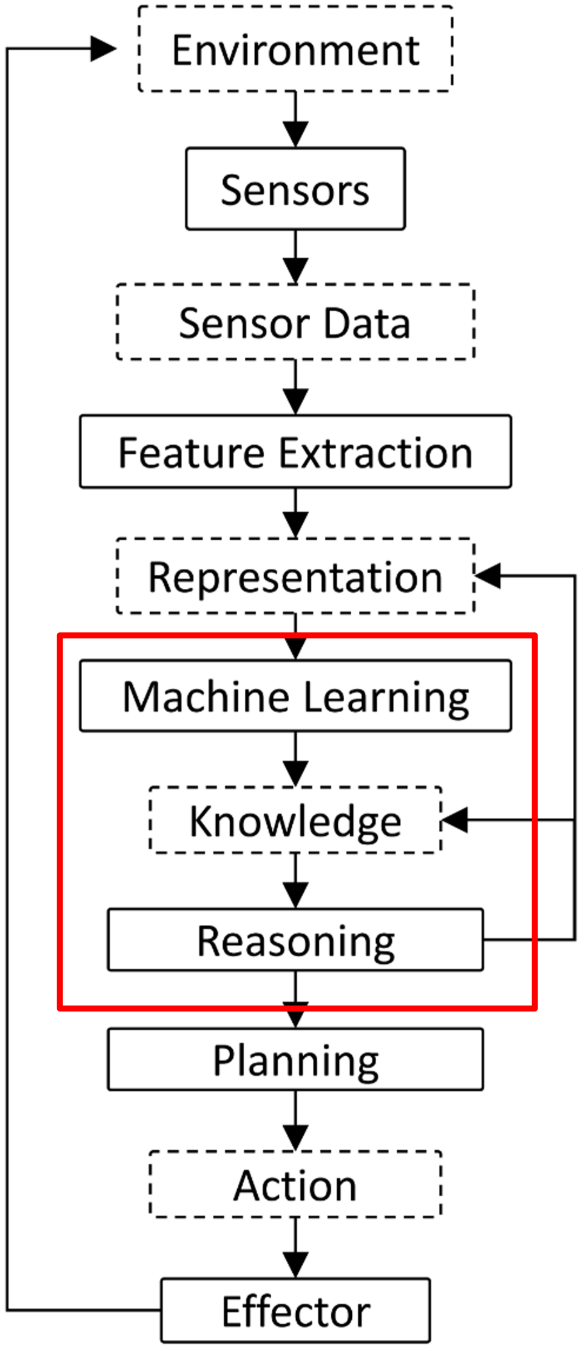

Networking
(Wired, Wireless)


IMU

Representations



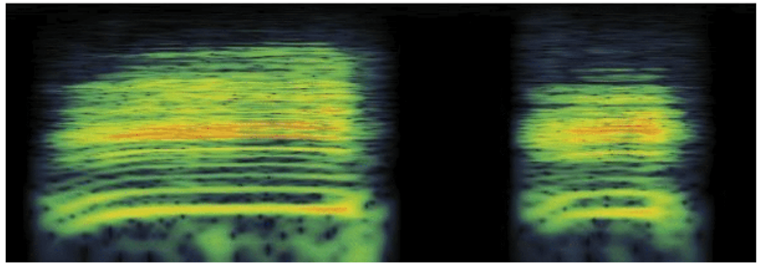




Knowledge / Reasoning

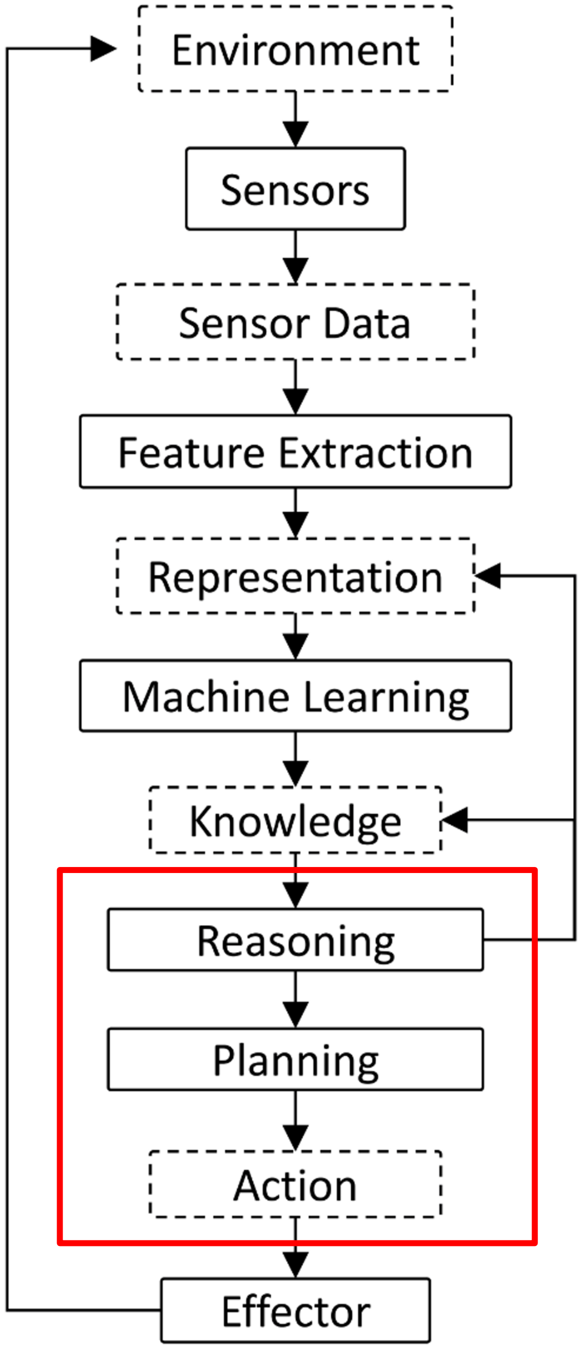
Image Recognition:
If it looks like a duck

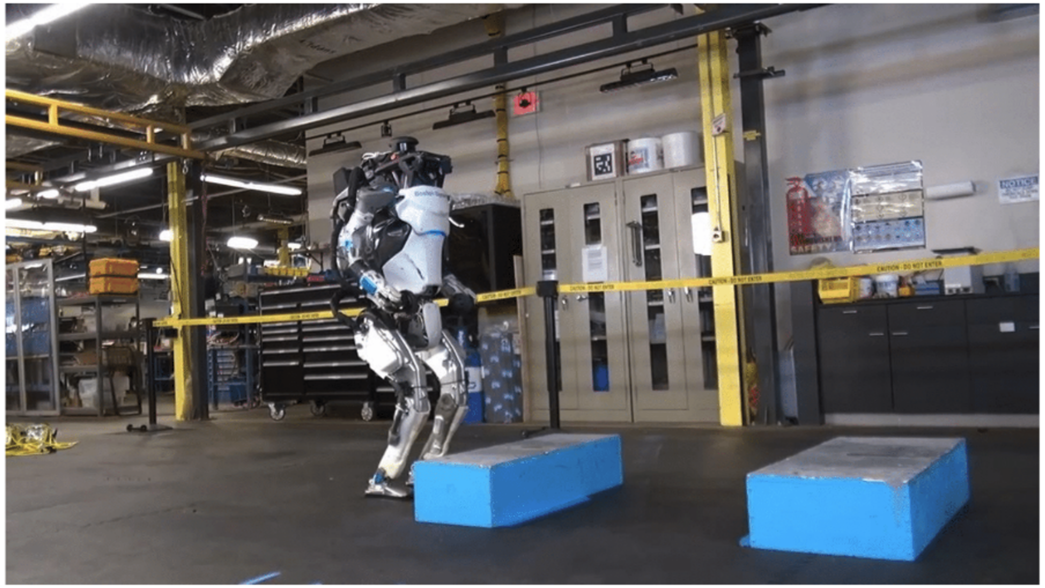
Audio Recognition:
Quacks like a duck

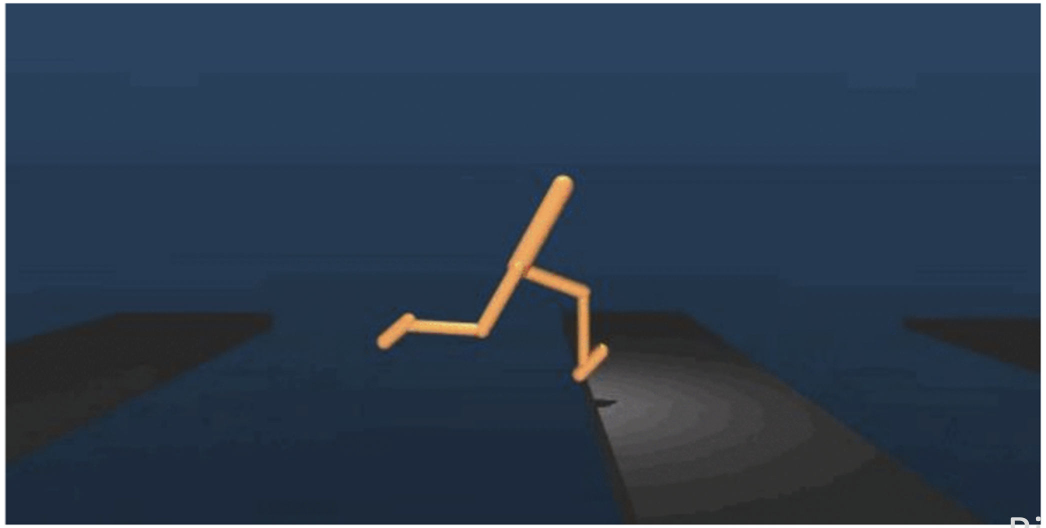


Activity Recognition:
Swims like a duck

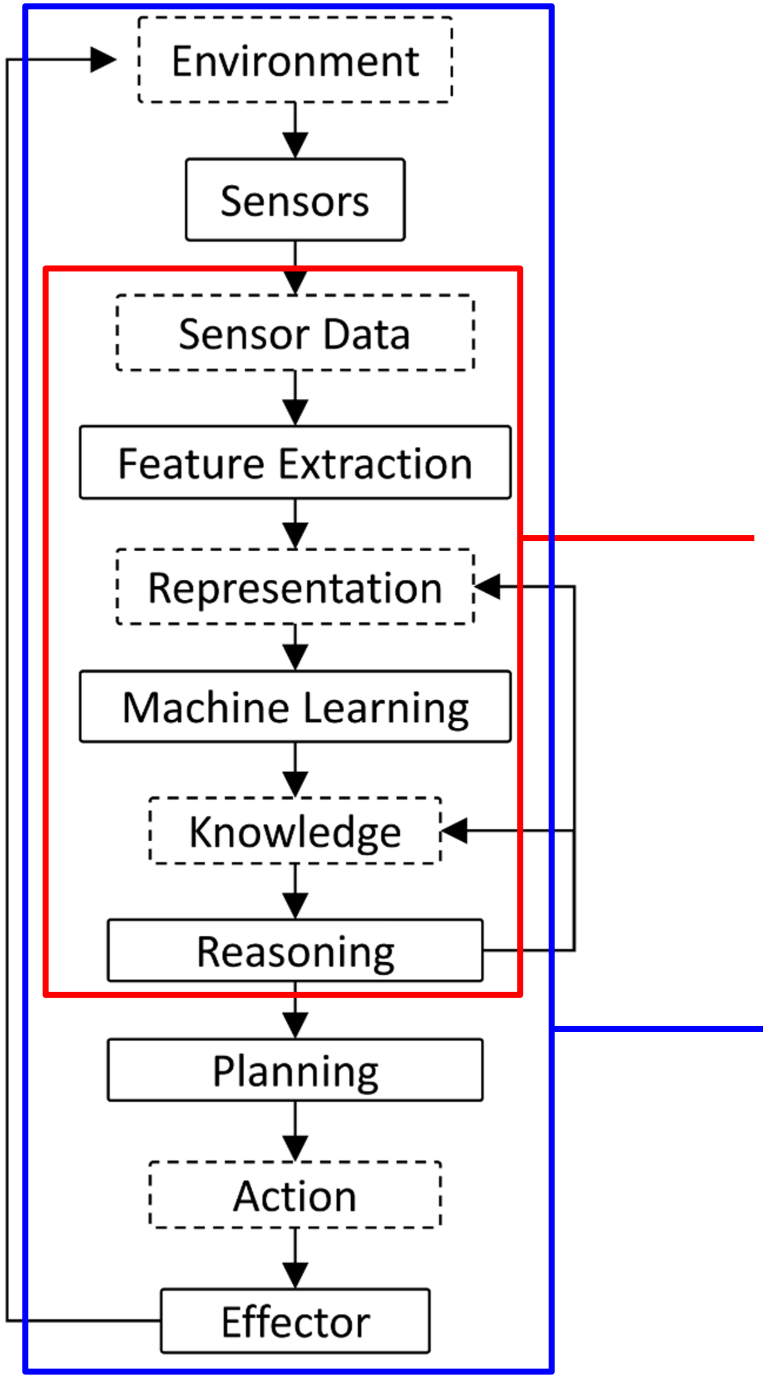
Actions


Environment
Sensors
Sensor Data
Feature Extraction
Representation
Machine Learning
Knowledge
Reasoning
Planning
Action
Effector





The Full Stack



The promise of
Deep Learning

The promise of
Deep Reinforcement Learning

Reinforcement Learning

- Learning to interact with an environment
  - Robots, games, process control
  - With limited human training
  - Where the ‘right thing’ isn’t obvious

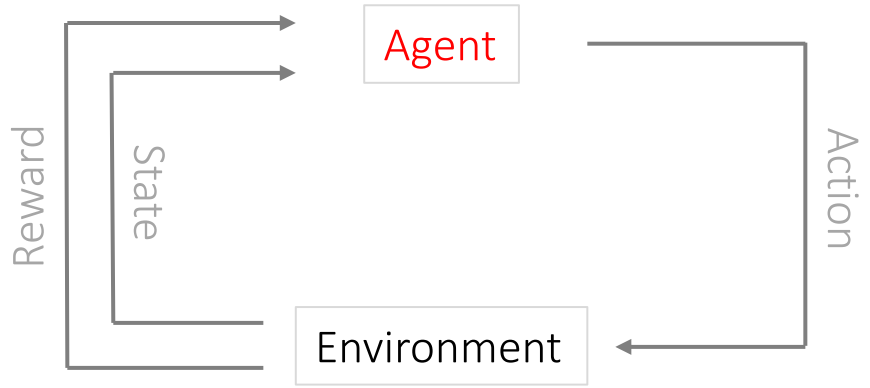
- Supervised Learning:
  - Goal: $f(x) = y$
  - Data: $[< x_1, y_1 >, \ldots, < x_n, y_n >]$

- Reinforcement Learning:
  - Goal:

    Maximize $\sum_{i=1}^{\infty} Reward(State_i, Action_i)$

  - Data:

    $Reward_i, State_{i+1} = Interact(State_i, Action_i)$

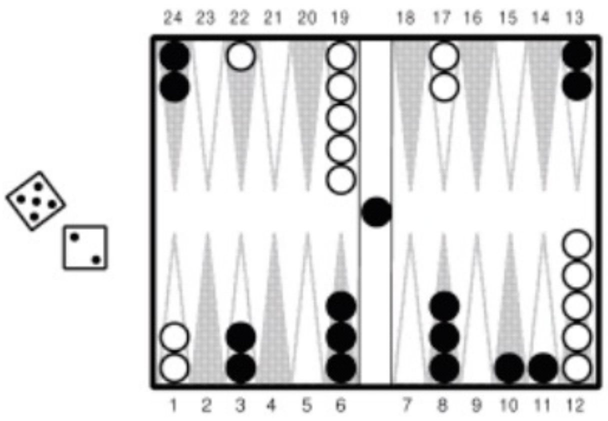


credit: Geoff Hulten

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).

A Success Story



- TD Gammon (Tesauro, G., 1992)

  - A Backgammon playing program.

  - Application of temporal difference learning.

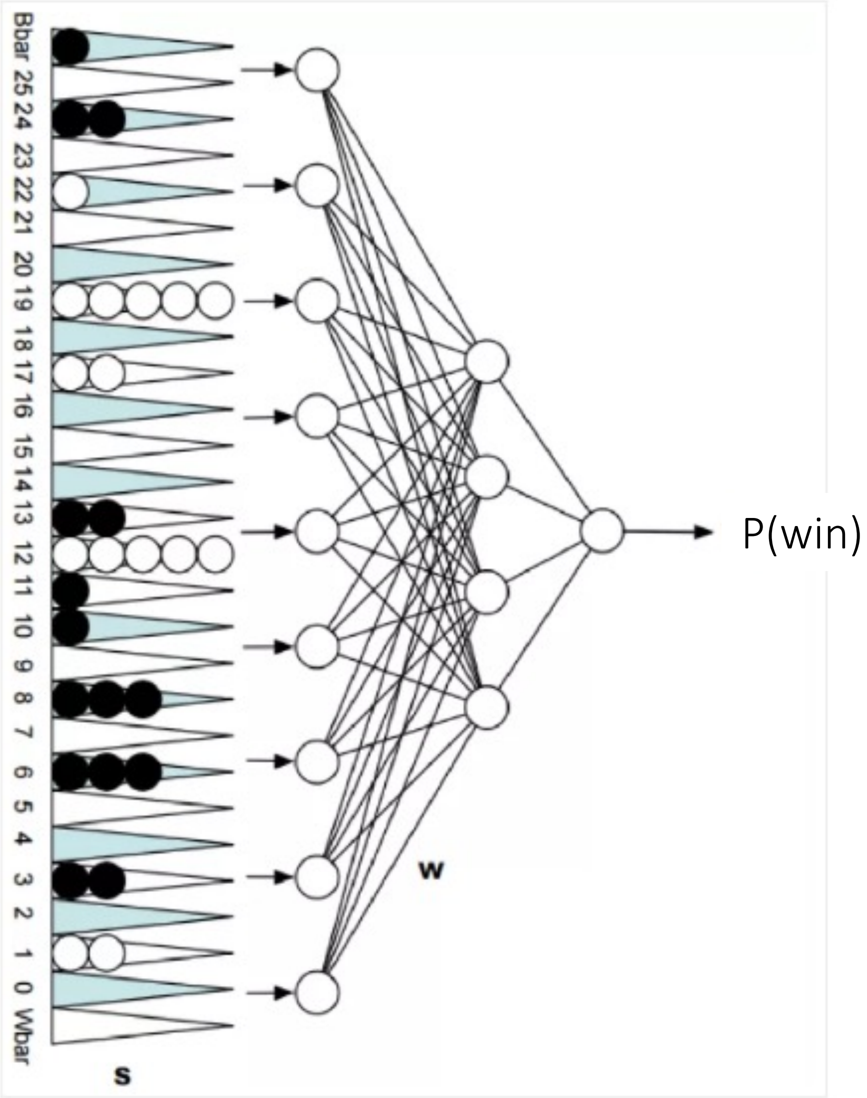
  - The basic learner is a neural network.

  - 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

TD-Gammon – Tesauro ~1995

State: Board State
Actions: Valid Moves
Reward: Win or Lose



- Net with 80 hidden units, initialize to random weights
- Select move based on network estimate & shallow search
- Learn by playing against itself
- 1.5 million games of training
  -> competitive with world class players

credit: Geoff Hulten

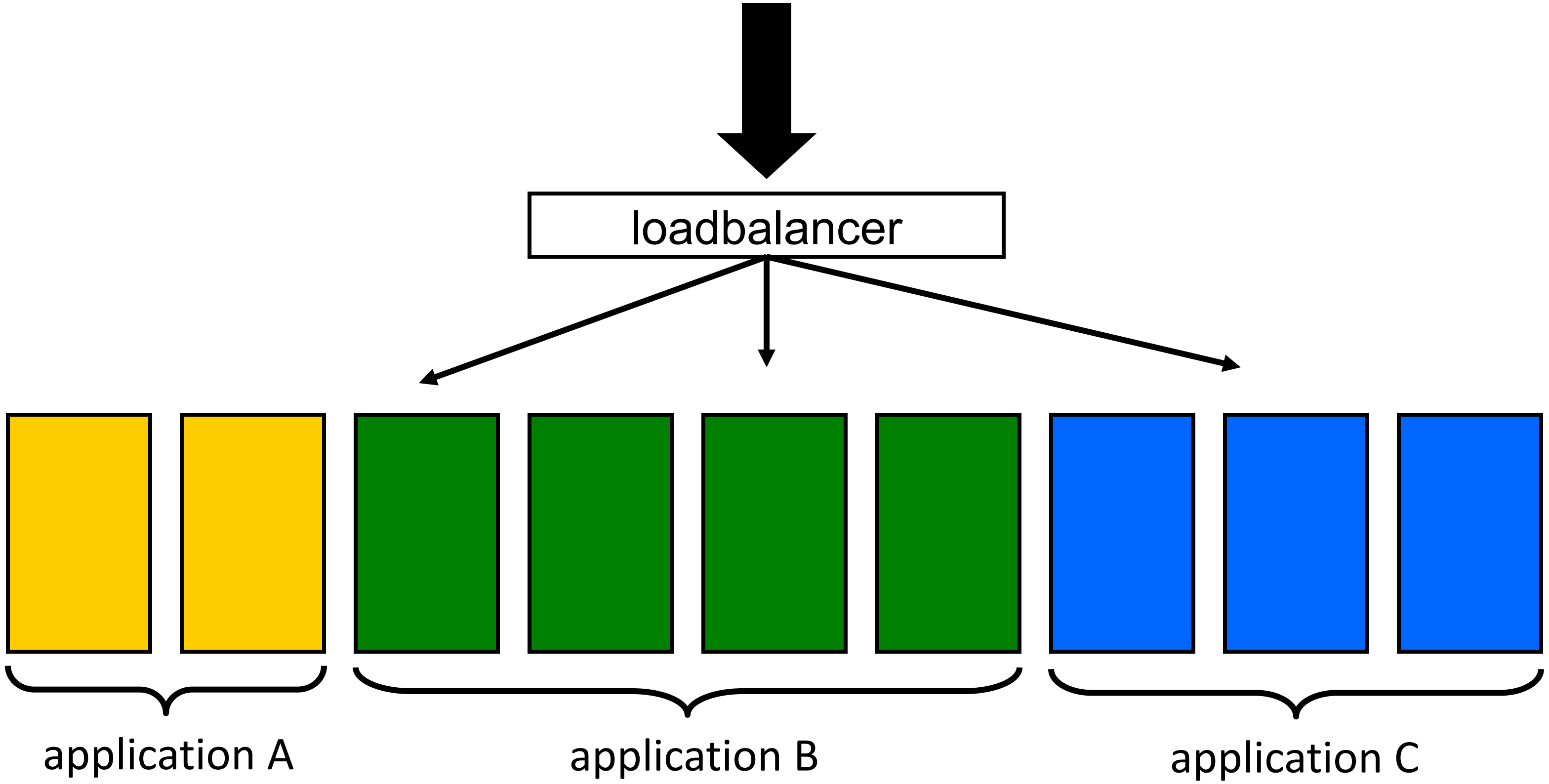
Examples of Reinforcement Learning

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

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

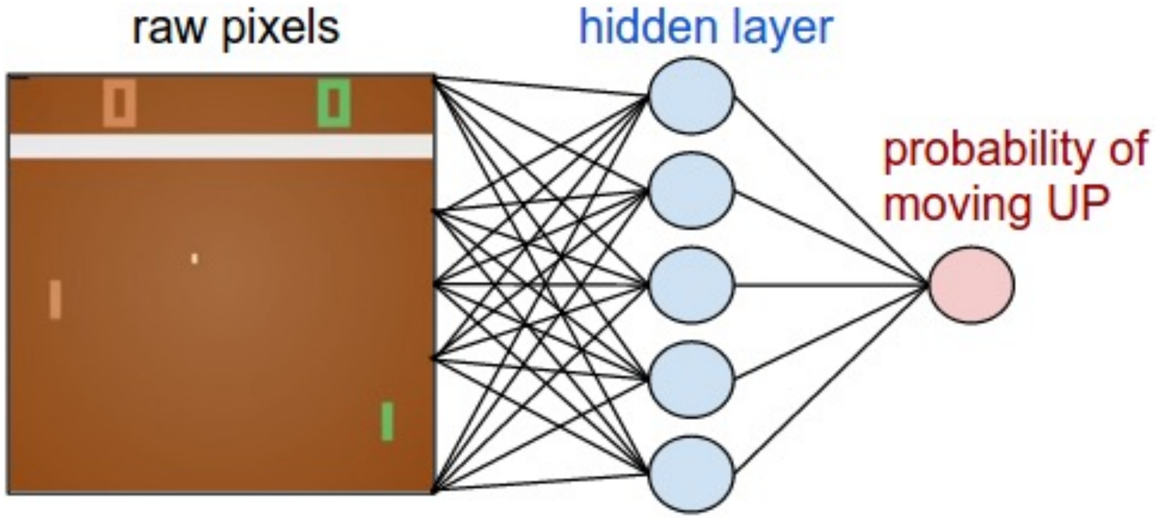
- How to make a good chess-playing program? (Artificial Intelligence)

Resource allocation in datacenters

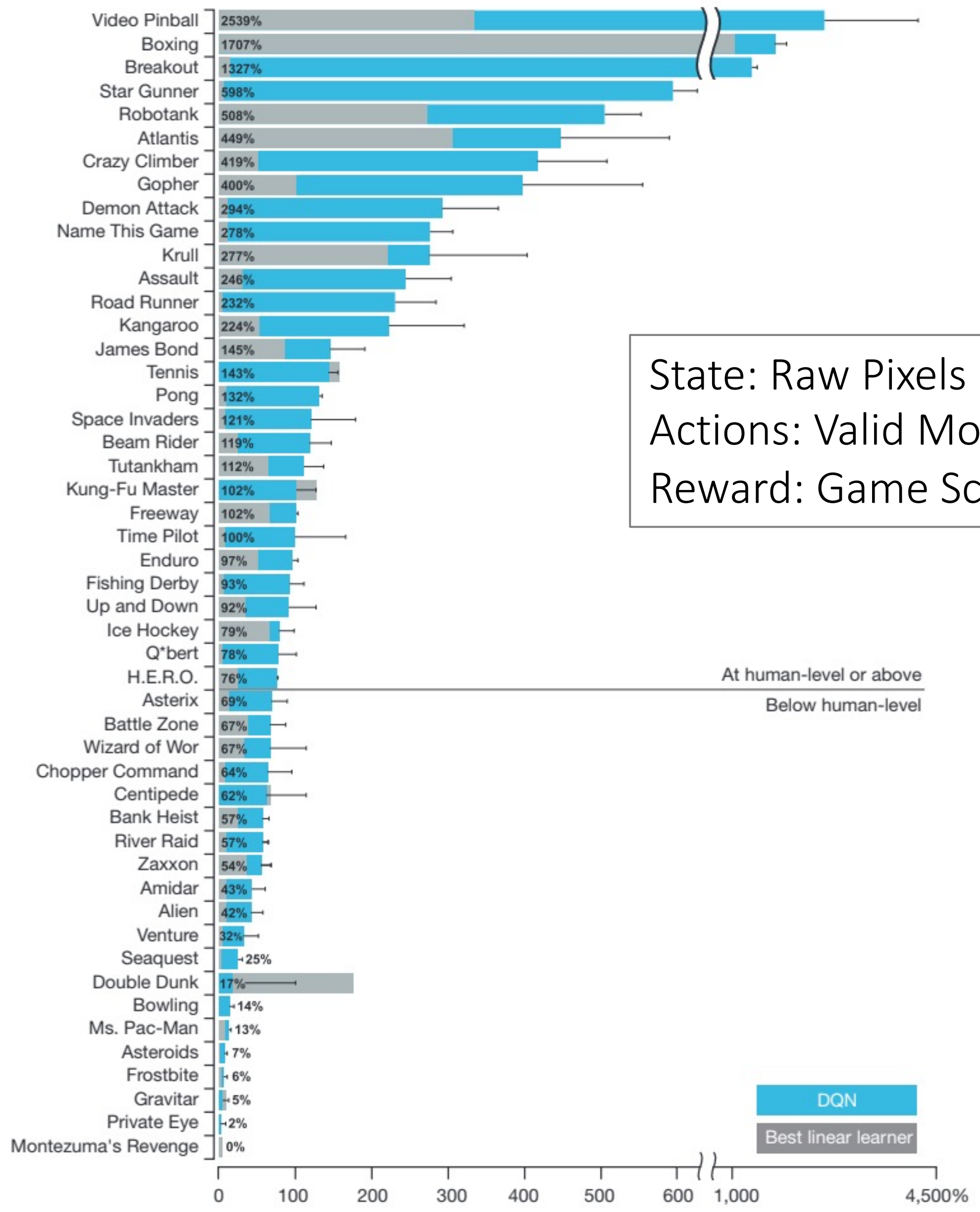


- A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation
  - Tesauro, Jong, Das, Bennani (IBM)
  - ICAC 2006

Atari 2600 games



- Same model/parameters for ~50 games



State: Raw Pixels
Actions: Valid Moves
Reward: Game Score

credit: Geoff Hulten

https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

Robotics and Locomotion

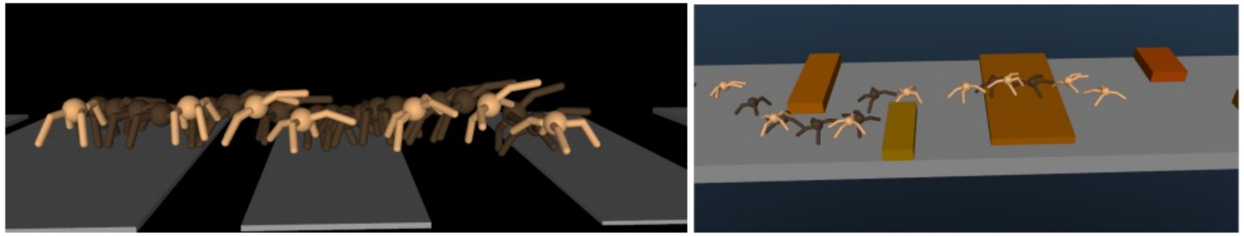
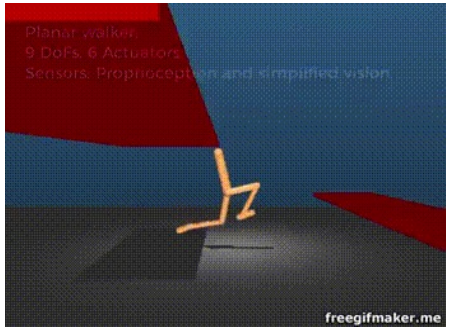
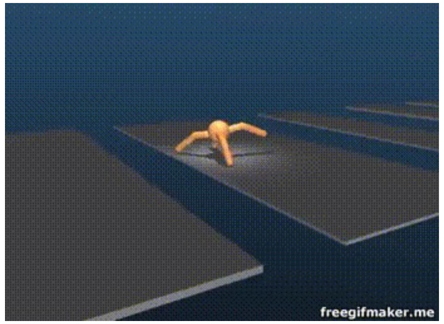
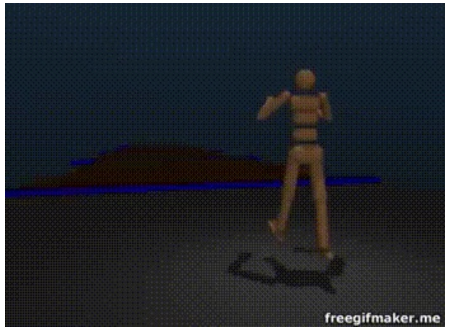


Figure 5: Time-lapse images of a representative *Quadruped* policy traversing gaps (left); and navigating obstacles (right)







State:
- Joint States/Velocities
- Accelerometer/Gyroscope
- Terrain

Actions: Apply Torque to Joints

Reward: Velocity – { stuff }

https://youtu.be/hx_bgoTF7bs

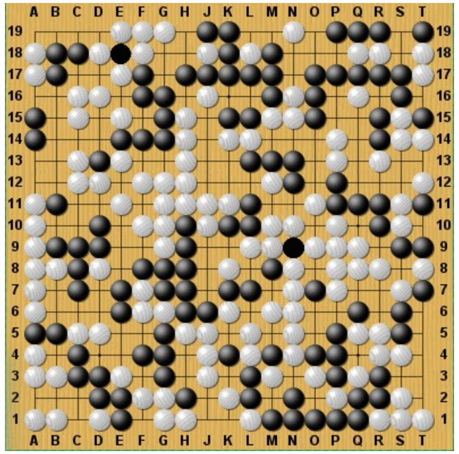
credit: Geoff Hulten

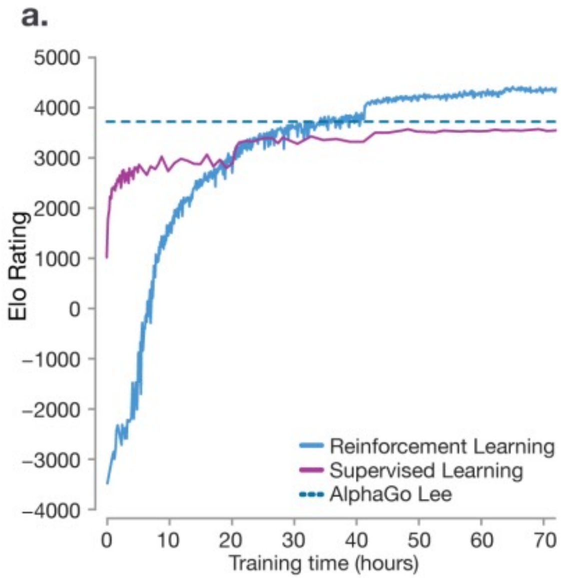
2017 paper https://arxiv.org/pdf/1707.02286.pdf

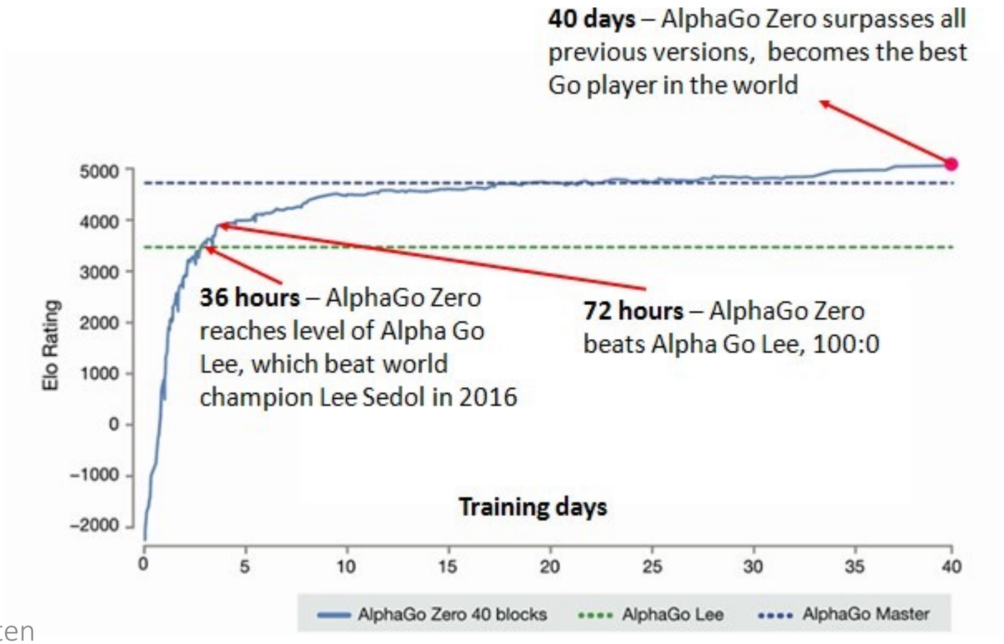
Alpha Go

State: Board State
Actions: Valid Moves
Reward: Win or Lose

- Learning how to beat humans at ‘hard’ games (search space too big)
- Far surpasses (Human) Supervised learning
- Algorithm learned to outplay humans at chess in 24 hours



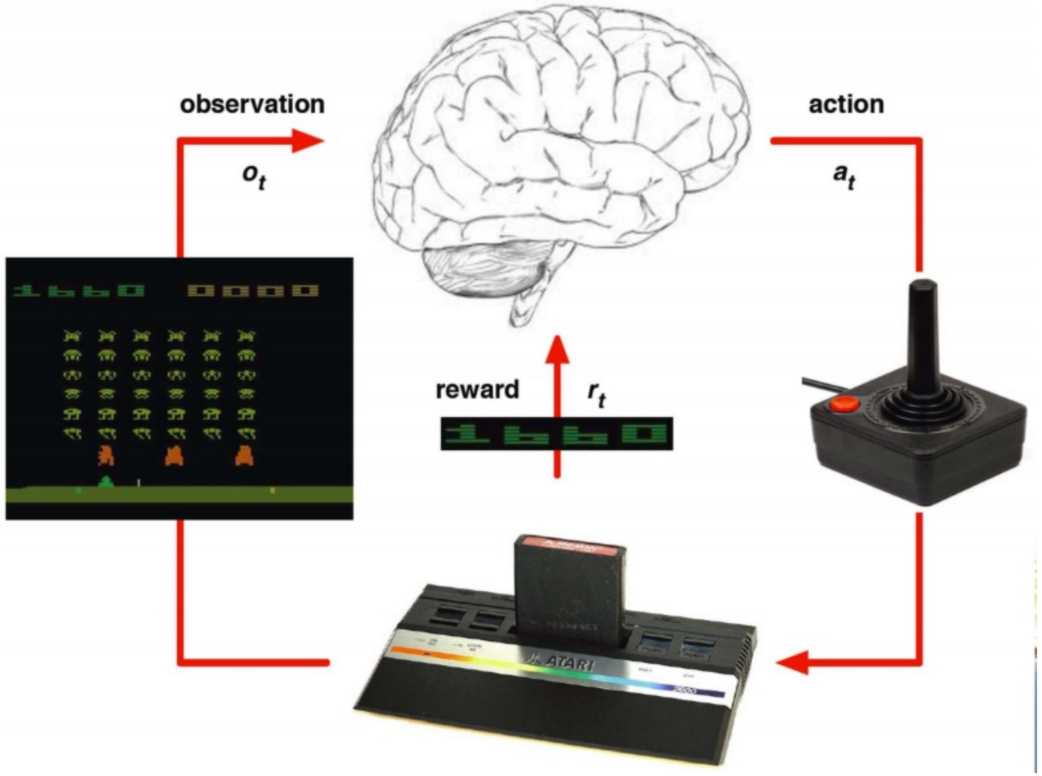




credit: Geoff Hulten

https://deepmind.com/documents/119/agz_unformatted_nature.pdf

Deep Reinforcement Learning



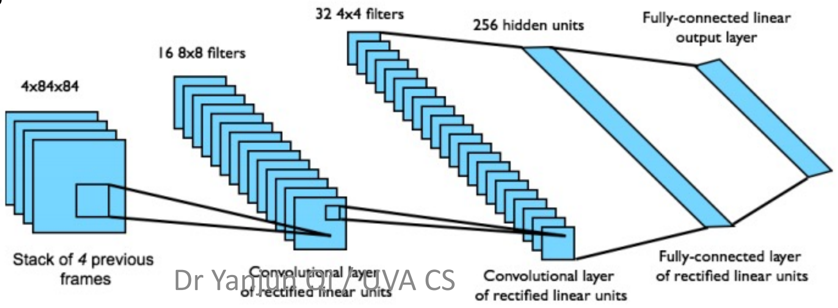
- Human

- So what's **DEEP** RL?

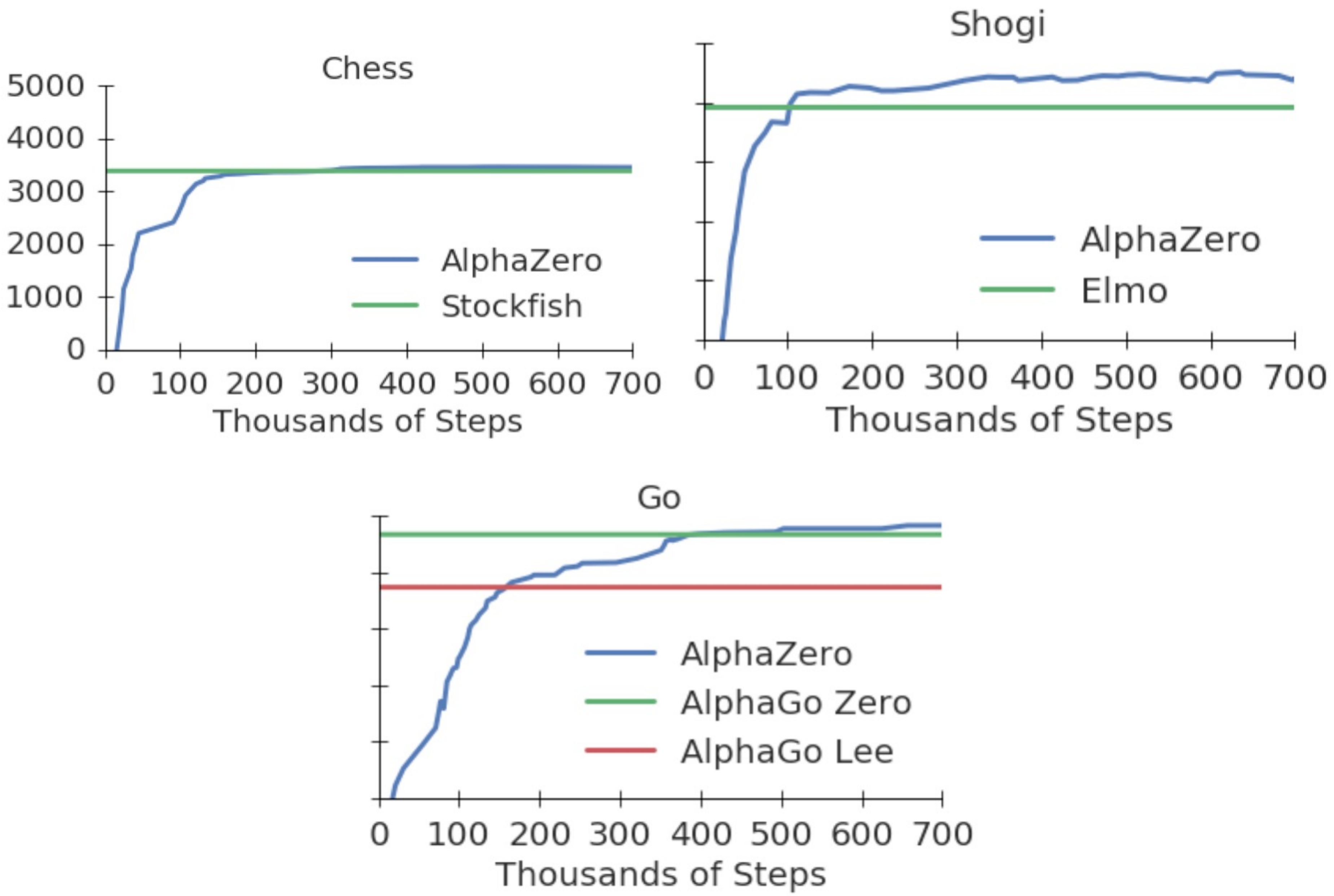
Environment

{Raw Observation, Reward}

{Actions}



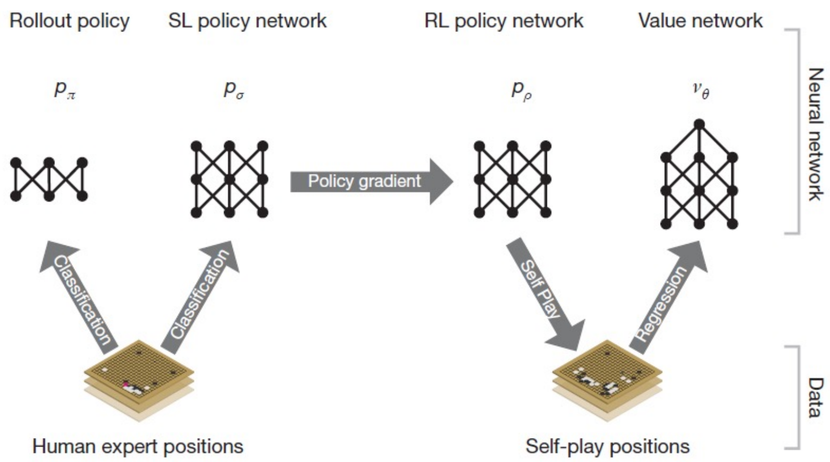
Adapt from Professor Qiang Yang of HK UST



Silver, David et al. (2017b). “Mastering the Game of Go without Human Knowledge”. In: Nature 550.7676, pp. 354–359.

AlphaGO: Learning Pipeline

- Combine Supervised Learning (SL) and RL to learn the search direction in Monte Carlo Tree Search



Silver, David, et al. 2016.

- SL policy Network
  - Prior search probability or potential
- Rollout:
  - combine with MCTS for quick simulation on leaf node
- Value Network:
  - Build the Global feeling on the leaf node situation

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016): 484-489.

AlphaGo {Fan, Lee, Master} × AlphaGo Zero:

- supervised learning from human expert positions × from scratch by self-play reinforcement learning ("tabula rasa")
- additional (auxialiary) input features × only the black and white stones from the board as input features
- separate policy and value networks × single neural network
- tree search using also Monte Carlo rollouts × simpler tree search using only the single neural network to both evaluate positions and sample moves
- (AlphaGo Lee) distributed machines + 48 tensor processing units (TPUs) × single machines + 4 TPUs
- (AlphaGo Lee) several months of training time × 72 h of training time (outperforming AlphaGo Lee after 36 h)

Silver, David et al. (2017b). "Mastering the Game of Go without Human Knowledge". In: Nature 550.7676, pp. 354–359.

OpenAI Spinning Up

latest

Search docs

Benchmarks for Spinning Up Implementations

Table of Contents

We benchmarked the Spinning Up algorithm implementations in five environments from the MuJoCo Gym task suite: HalfCheetah, Hopper, Walker2d, Swimmer, and Ant.

Performance in Each Environment

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 learn by trial-and-error. (Not entirely supervised, but interactive)
- Actions may affect not only the immediate reward but also subsequent rewards (Delayed effect).

Outline

- Examples of RL applications
- Defining an RL problem
  - Markov Decision Processes
- Solving an RL problem
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning

credit: Geoff Hulten

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).

Setup for Reinforcement Learning

Markov Decision Process (environment)

- Discrete-time stochastic control process
- Each time step, s:
  - Agent chooses action a from set A_s
  - Moves to new state with probability:
    - $P_a(s, s')$
  - Receives reward: $R_a(s, s')$
- Every outcome depends on s and a
  - Nothing depends on previous states/actions

Policy

(agent's behavior)

- $\pi(s)$ – The action to take in state s
- Goal maximize: $\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$
  - $a_t = \pi(s_t)$
  - $0 \leq \gamma < 1$ – Tradeoff immediate vs future
- $V^{\pi}(s) =$
  $\sum_{s'} P_{\pi(s)}(s, s') \; *$
  $(R_{\pi(s)}(s, s') + \gamma V^{\pi}(s'))$

Probability of moving to each state

Reward for making that move

Value of being in that state

credit: Geoff Hulten

Simple Example of Agent in an Environment

State:

Map Locations

$\{< 0{,}0 >, < 1{,}0 > \cdots < 3{,}3 >\}$

Actions:

Move within map

Reaching chest ends episode

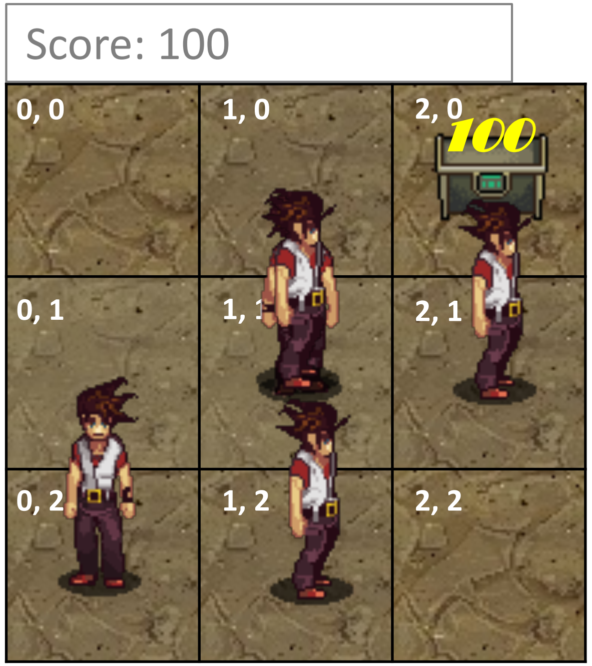
$A_{0,0} = \{\, east, south \,\}$
$A_{1,0} = \{\, east, south, west \,\}$
$A_{2,0} = \{\, \phi \,\}$
...
$A_{2,2} = \{\, north, west \,\}$

Reward:

100 at chest

0 for others

$R_{east}(< 1{,}0 >, < 2{,}0 >) = 100$
$R_{north}(< 2{,}1 >, < 2{,}0 >) = 100$
$R_{*}(*,*) = 0$



credit: Geoff Hulten

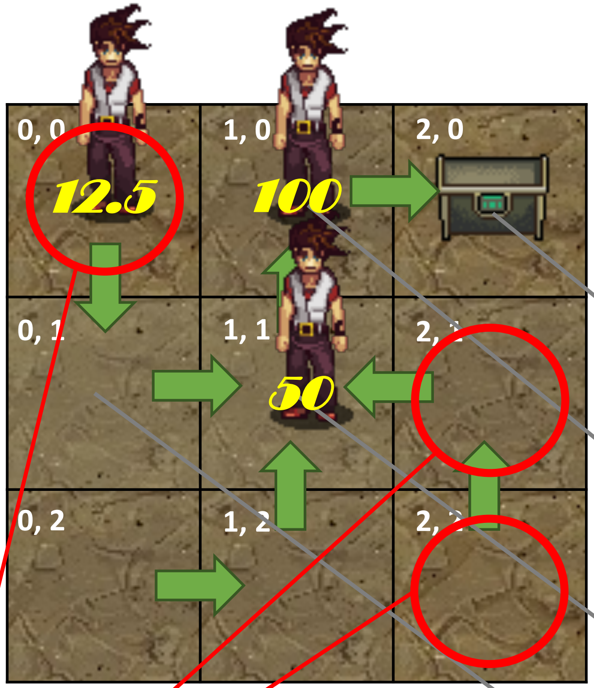
Policies

$$R_{east}(<1{,}0>, <2{,}0>) = 100$$
$$R_{north}(<2{,}1>, <2{,}0>) = 100$$
$$R_{*}(*,*) = 0$$
$$\gamma = 0.5$$

Policy

$$\pi(s) = a$$

$$\pi(<0{,}0>) = \{south\}$$
$$\pi(<0{,}1>) = \{east\}$$
$$\pi(<0{,}2>) = \{east\}$$
$$\pi(<1{,}0>) = \{east\}$$
$$\pi(<1{,}1>) = \{north\}$$
$$\pi(<1{,}2>) = \{north\}$$
$$\pi(<2{,}0>) = \{\phi\}$$
$$\pi(<2{,}1>) = \{west\}$$
$$\pi(<2{,}2>) = \{north\}$$



Policy could be better

Evaluating Policies

$$V^{\pi}(s) = \sum_{i=0}^{\infty} \gamma^{i} r_{i+1}$$

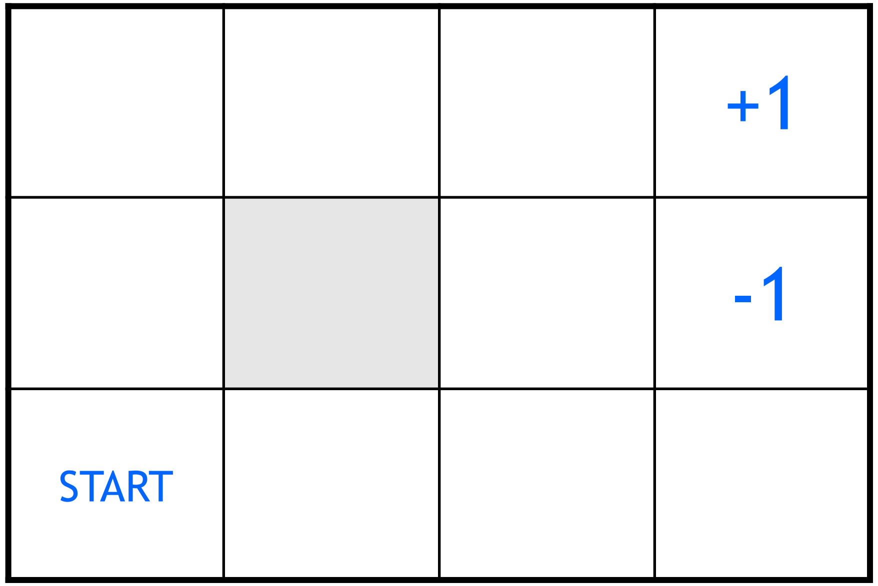
$$V^{\pi}(<1{,}0>) = \gamma^{0} * 100$$

$$V^{\pi}(<1{,}1>) = \gamma^{0} * 0 + \gamma^{1} * 100$$

Move to <0,1> Move to <1,1> Move to <1,0> Move to <2,0>

$$V^{\pi}(<0{,}0>) = \boxed{\gamma^{0} * 0} + \boxed{\gamma^{1} * 0} + \boxed{\gamma^{2} * 0} + \boxed{\gamma^{3} * 100}$$

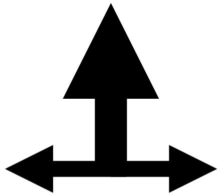
Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

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

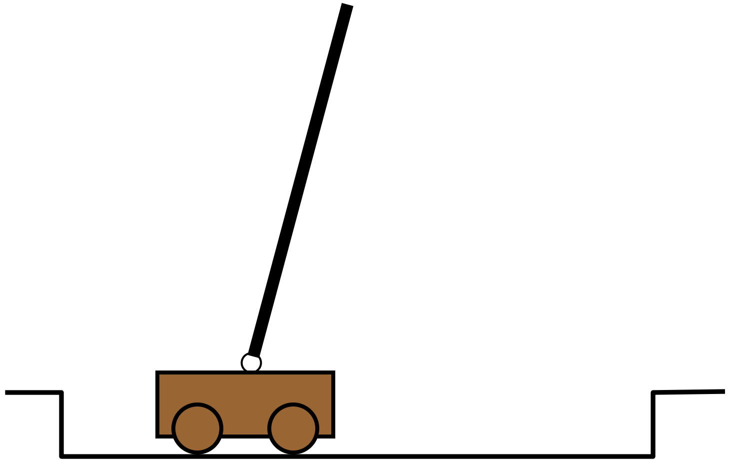


- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step

- what's the strategy to achieve max reward?
- what if the actions were deterministic?

credit: Geoff Hulten

Other examples



- pole-balancing
- TD-Gammon [Gerry Tesauro]
- helicopter [Andrew Ng]

- no teacher who would say “good” or “bad”
  - is reward “10” good or bad?
  - rewards could be delayed

- similar to control theory
  - more general, fewer constraints

- explore the environment and learn from experience
  - not just blind search, try to be smart about it

credit: Geoff Hulten

How Reinforcement Learning is Different

- Delayed Reward
- Agent chooses training data
- Explore vs Exploit (Life long learning)
- Very different terminology (can be confusing)

credit: Geoff Hulten

Outline

- Examples of RL applications

- Defining an RL problem
  - Markov Decision Processes

- Solving an RL problem
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning

credit: Peter Bodí

The Precise Goal / Popular RL Algorithms

- 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.

(1) Dynamic programming

- main idea
  - use value functions to structure the search for good policies
  - need a perfect model of the environment

- two main components
  - policy evaluation: compute V^{π} from π
  - policy improvement: improve π based on V^{π}

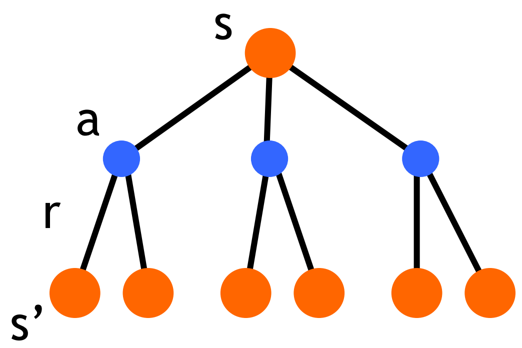
  - start with an arbitrary policy
  - repeat evaluation/improvement until convergence

Value functions

- state value function: $V^{\pi}(s)$
  - expected return when starting in *s* and following π

- state-action value function: Q-function: $Q^{\pi}(s,a)$
  - expected return when starting in *s*, performing *a,* and following π

- useful for finding the optimal policy
  - can estimate from experience
  - pick the best action using $Q^{\pi}(s,a)$



$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V^{\pi}(s') \right] = \sum_{a} \pi(s,a) Q^{\pi}(s,a)$$

- Bellman equation

credit: Peter Bodí

Using DP

- need complete model of the environment and rewards
  - robot in a room
    - state space, action space, transition model
- can we use DP to solve
  - robot in a room?
  - back gammon?
  - helicopter?

credit: Peter Bodí

Outline

- Examples of RL applications
- Defining an RL problem
    - Markov Decision Processes
- Solving an RL problem
    - Dynamic Programming
    - Monte Carlo methods
    - Temporal-Difference learning

credit: Peter Bodí

Monte Carlo methods

- don’t need full knowledge of environment
  - just experience, or
  - simulated experience
- but similar to DP
  - policy evaluation, policy improvement
- averaging sample returns
  - defined only for episodic tasks

credit: Peter Bodí

Computing return from rewards

- episodic (vs. continuing) tasks
  - "game over" after N steps
  - optimal policy depends on N; harder to analyze

- additive rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  - infinite value for continuing tasks

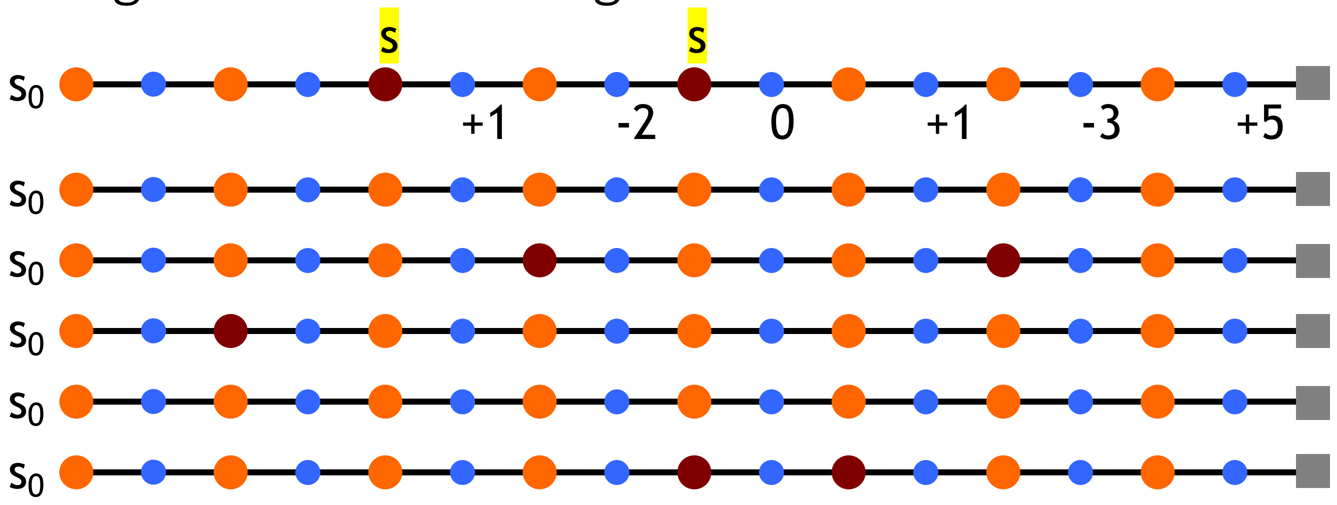
- discounted rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$
  - value bounded if rewards bounded

credit: Peter Bodí

Monte Carlo policy evaluation

- want to estimate $V^\pi(s)$
  = expected return starting from s and following π
  - estimate as average of observed returns in state s

- first-visit MC
  - average returns following the first visit to state s



$R_1(s) = +2$

$R_2(s) = +1$

$R_3(s) = -5$

$R_4(s) = +4$

$V^\pi(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$

credit: Peter Bodí

Maintaining exploration

- deterministic/greedy policy won't explore all actions
  - don't know anything about the environment at the beginning
  - need to try all actions to find the optimal one
- maintain exploration
  - use *soft* policies instead: $\pi(s,a)>0$ (for all s,a)
- ε-greedy policy
  - with probability 1-ε perform the optimal/greedy action
  - with probability ε perform a random action

  - will keep exploring the environment
  - slowly move it towards greedy policy: ε -> 0

credit: Peter Bodí

Simulated experience

- 5-card draw poker
  - s_0: A♣, A♦, 6♠, A♥, 2♠
  - a_0: discard 6♠, 2♠
  - s_1: A♣, A♦, A♥, A♠, 9♠ + dealer takes 4 cards
  - return: +1 (probably)

- DP
  - list all states, actions, compute P(s,a,s')
    - P([A♣,A♦,6♠,A♥,2♠], [6♠,2♠], [A♠,9♠,4]) = 0.00192

- MC
  - all you need are sample episodes
  - let MC play against a random policy, or itself, or another algorithm

credit: Peter Bodí

Summary of Monte Carlo

- don’t need model of environment
  - averaging of sample returns
  - only for episodic tasks

- learn from sample episodes or simulated experience

- can concentrate on “important” states
  - don’t need a full sweep

- need to maintain exploration
  - use soft policies

credit: Peter Bodí

Outline

- Examples of RL applications

- Defining an RL problem
    - Markov Decision Processes

- Solving an RL problem
    - Dynamic Programming
    - Monte Carlo methods
    - Temporal-Difference learning

credit: Peter Bodí

Temporal Difference Learning

- combines ideas from MC and DP
  - like MC: learn directly from experience (don't need a model)
  - like DP: learn from values of successors
  - works for continuous tasks, usually faster than MC

- constant-alpha MC:
  - have to wait until the end of episode to update

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t)\right]$$

target

- simplest TD
  - update after every step, based on the successor

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

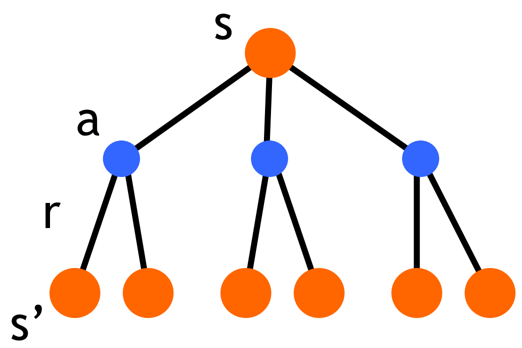
credit: Peter Bodí

Value functions

- state value function: $V^{\pi}(s)$
  - expected return when starting in *s* and following π

- state-action value function: Q-function: $Q^{\pi}(s,a)$
  - expected return when starting in *s*, performing *a*, and following π

- useful for finding the optimal policy
  - can estimate from experience
  - pick the best action using $Q^{\pi}(s,a)$



$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^{\pi}(s') \right] = \sum_a \pi(s,a) Q^{\pi}(s,a)$$

- Bellman equation

credit: Peter Bodí

Optimal value functions

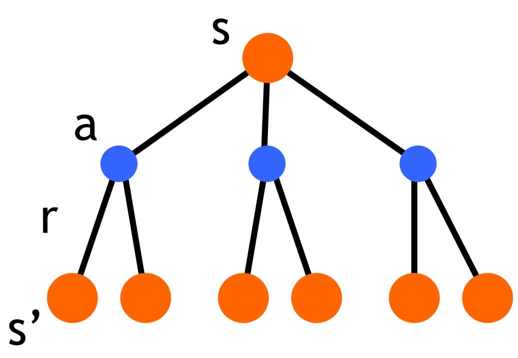
- there's a set of *optimal* policies
  - V^π defines partial ordering on policies
  - they share the same optimal value function

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

- Bellman optimality equation

$$V^*(s) = \max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^*(s') \right]$$

  - system of n non-linear equations
  - solve for V*(s)
  - easy to extract the optimal policy



- having Q*(s,a) makes it even simpler

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

credit: Peter Bodí

Q-learning

- before: on-policy algorithms
  - start with a random policy, iteratively improve
  - converge to optimal

- Q-learning: off-policy
  - use any policy to estimate Q

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

  - Q directly approximates Q* (Bellman optimality eqn)
  - independent of the policy being followed
  - only requirement: keep updating each (s,a) pair

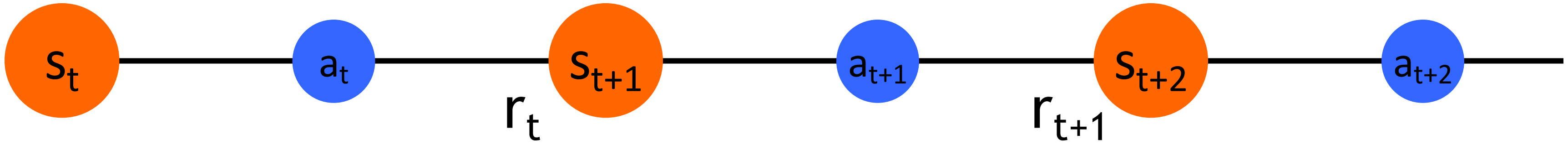
- Sarsa

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

credit: Peter Bodí

Sarsa

- again, need Q(s,a), not just V(s)

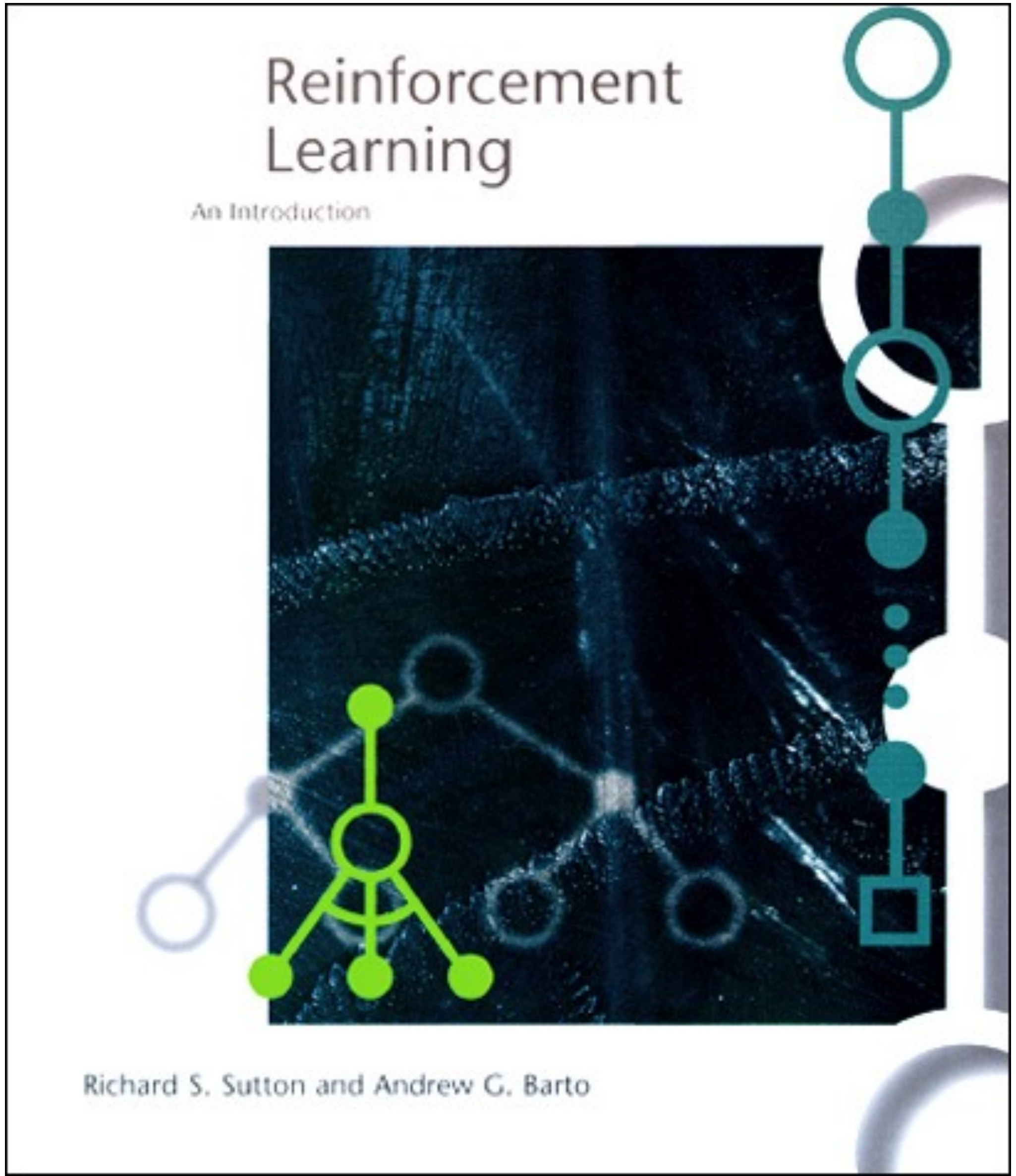


$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right]$$

- control
  - start with a random policy
  - update Q and π after each step
  - again, need ε-soft policies

credit: Peter Bodí

The RL Intro book



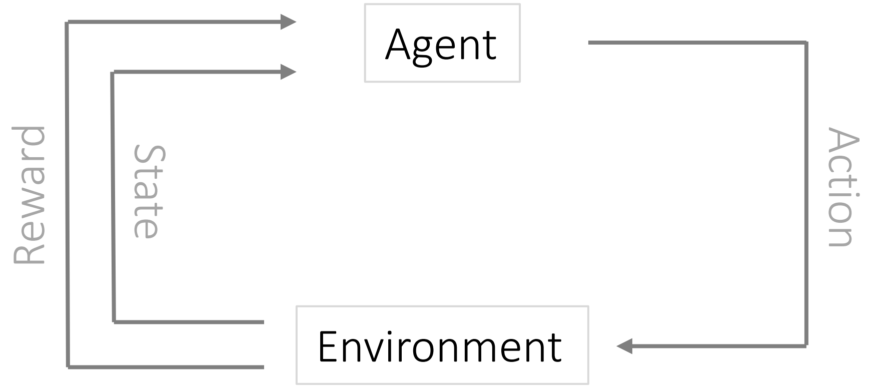
Richard Sutton, Andrew Barto
Reinforcement Learning,
An Introduction

http://www.cs.ualberta.ca/~sutton/book/the-book.html

credit: Peter Bodí

Summary

Reinforcement Learning:
- Goal: Maximize $\sum_{i=1}^{\infty} Reward(State_i, Action_i)$
- Data: $Reward_{i+1}, State_{i+1} = Interact(State_i, Action_i)$



Many (awesome) recent successes:
- Robotics
- Surpassing humans at difficult games
- Doing it with (essentially) zero human knowledge

Challenges:
- When the episode can end without reward
- When there is a 'narrow' path to reward
- When there are many states and actions

(Simple) Approaches:
- Q-Learning $\hat{Q}(s,a)$ -> discounted reward of action
- Policy Gradients -> Probability distribution over A_s
- Reward Shaping
- Memory
- Lots of parameter tweaking…

credit: Geoff Hulten

https://spinningup.openai.com/en/latest/

- Key Papers in Deep RL
  - 1. Model-Free RL
  - 2. Exploration
  - 3. Transfer and Multitask RL
  - 4. Hierarchy
  - 5. Memory
  - 6. Model-Based RL
  - 7. Meta-RL
  - 8. Scaling RL
  - 9. RL in the Real World
  - 10. Safety
  - 11. Imitation Learning and Inverse Reinforcement Learning
  - 12. Reproducibility, Analysis, and Critique
  - 13. Bonus: Classic Papers in RL Theory or Review

credit: Geoff Hulten

References

- RL slides from Rich Nguven
- RL Slides from Geoff Hulten
- RL slides from Eric Xing
- RL slides from Peter Bodik

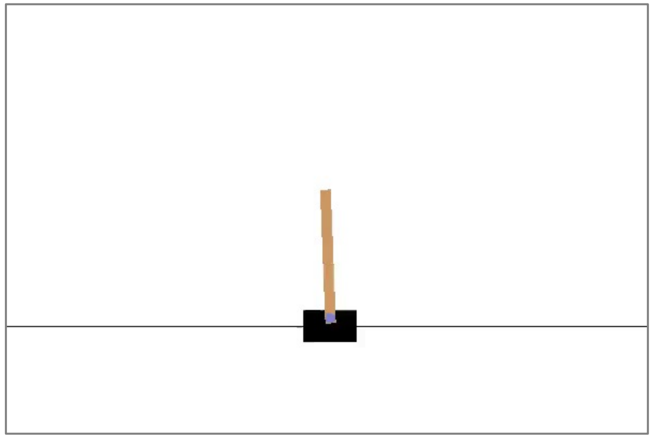
credit: Geoff Hulten

Algorithms Docs

- Vanilla Policy Gradient
  - Background
  - Documentation
  - References
- Trust Region Policy Optimization
  - Background
  - Documentation
  - References
- Proximal Policy Optimization
  - Background
  - Documentation
  - References
- Deep Deterministic Policy Gradient
  - Background
  - Documentation
  - References
- Twin Delayed DDPG
  - Background
  - Documentation
  - References
- Soft Actor-Critic
  - Background
  - Documentation
  - References

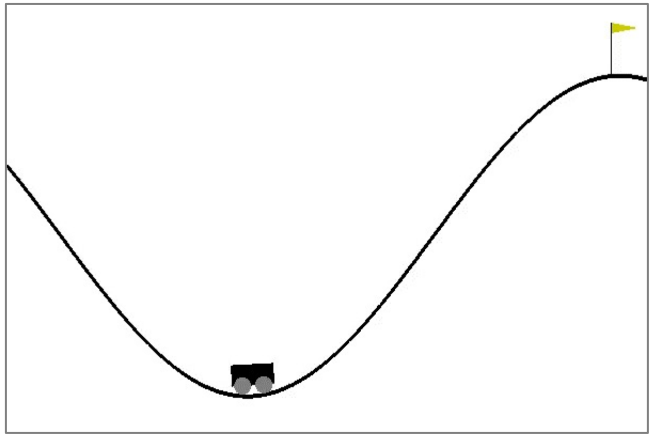
Gym – toolkit for reinforcement learning

CartPole



Reward +1 per step the pole remains up

MountainCar



Reward 200 at flag -1 per step

```python
import gym

env = gym.make('CartPole-v0')

import random
import QLearning # Your implementation goes here...
import Assignment7Support

trainingIterations = 20000

qlearner = QLearning.QLearning(<Parameters>)

for trialNumber in range(trainingIterations):
    observation = env.reset()
    reward = 0
    for i in range(300):
        env.render() # Comment out to make much faster...

        currentState = ObservationToStateSpace(observation)
        action = qlearner.GetAction(currentState, <Parameters>)

        oldState = ObservationToStateSpace(observation)
        observation, reward, isDone, info = env.step(action)
        newState = ObservationToStateSpace(observation)

        qlearner.ObserveAction(oldState, action, newState, reward, …)

        if isDone:
            if(trialNumber%1000) == 0:
                print(trialNumber, i, reward)
            break

# Now you have a policy in qlearner – use it...
```

https://gym.openai.com/docs/

Q learning

Learn a policy $\pi(s)$ that optimizes $V^{\pi}(s)$ for all states, using:

- No prior knowledge of state transition probabilities: $P_a(s, s')$
- No prior knowledge of the reward function: $R_a(s, s')$

Approach:

- Initialize estimate of discounted reward for every state/action pair: $\hat{Q}(s, a) = 0$
- Repeat (for a while):
  - Take a random action a from A_s
  - Receive s' and $R_a(s, s')$ from environment
  - Update $\hat{Q}(s, a)$ = $R_a(s, s') + \gamma \max_{a'} \hat{Q}(s', a')$
  - Random restart if in terminal state

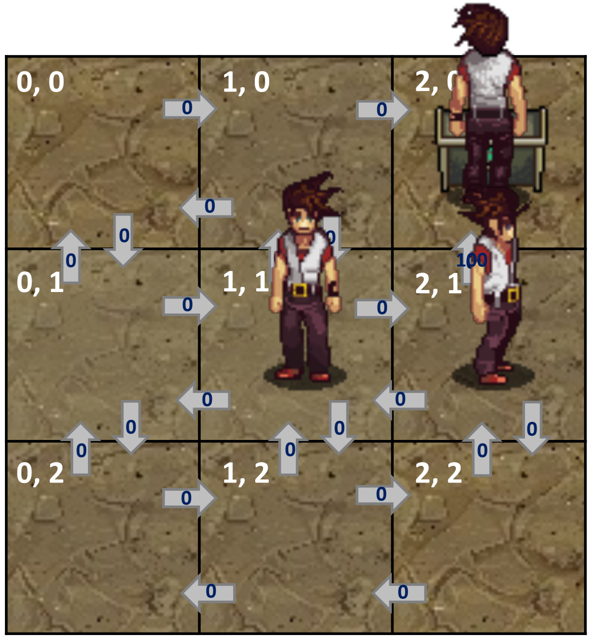
$$\propto_v = \frac{1}{1 + visits(s, a)}$$

Exploration Policy: $P(a_i, s) = \frac{k^{\hat{Q}(s, a_i)}}{\sum_j k^{\hat{Q}(s, a_j)}}$

credit: Peter Bodí

Example of Q learning (round 1)

- Initialize $\hat{Q}$ to 0
- Random initial state = $< 1{,}1 >$
- Random action from $A_{<1,1>} = east$
  - $s' =< 2{,}1 >$
  - $R_a(s, s') = 0$
- Update $\hat{Q}(< 1{,}1 >, east) = 0$
- Random action from $A_{<2,1>} = north$
  - $s' =< 2{,}0 >$
  - $R_a(s, s') = 100$
- Update $\hat{Q}(< 2{,}1 >, north) = 100$
- No more moves possible, start again…

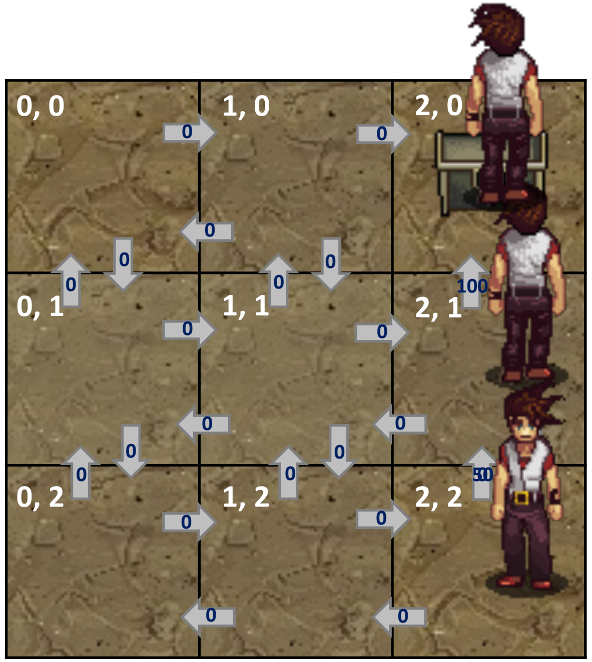


$$\hat{Q}(s, a) = R_a(s, s') + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')$$

credit: Peter Bodí

Example of Q learning (round 2)

- Round 2: Random initial state = $< 2{,}2 >$
- Random action from $A_{<2,2>} = north$
  - $s' =< 2{,}1 >$
  - $R_a(s, s') = 0$
- Update $\hat{Q}(< 2{,}1 >, north) = 0 + \gamma * 100$



- Random action from $A_{<2,1>} = north$
  - $s' =< 2{,}0 >$
  - $R_a(s, s') = 100$
- Update $\hat{Q}(< 2{,}1 >, north) = still\ 100$

- No more moves possible, start again…

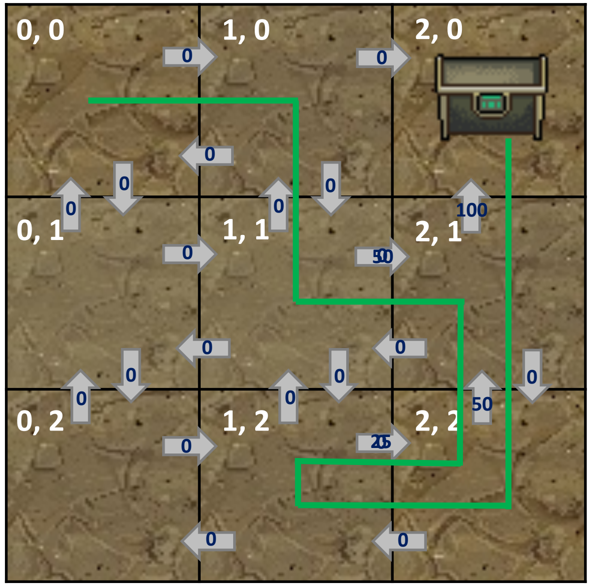
$$\hat{Q}(s,a) = R_a\,(s,s') + \gamma \max_{a'} \hat{Q}_{n-1}(s',a') \qquad \gamma = 0.5$$

credit: Peter Bodí

Example of Q learning (some acceleration…)

$$\hat{Q}(s,a) = R_a(s,s') + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')$$
$$\gamma = 0.5$$

- Random Initial State $< 0{,}0 >$
- Update $\hat{Q}(< 1{,}1 >, east) = 50$
- Update $\hat{Q}(< 1{,}2 >, east) = 25$

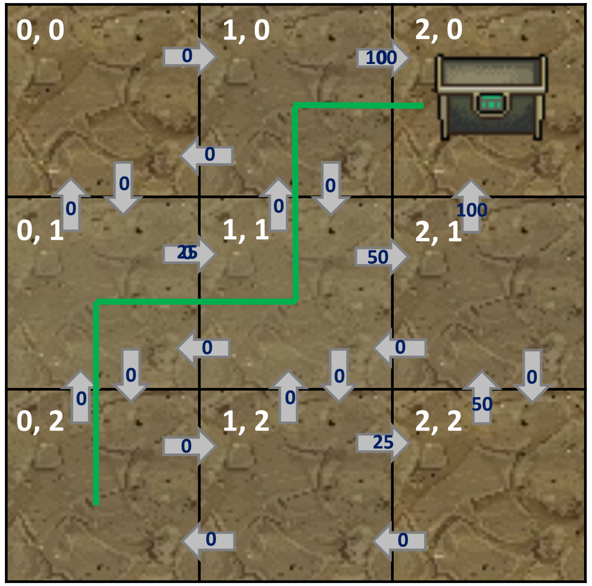


credit: Peter Bodí

Example of Q learning (some acceleration...)

$$\hat{Q}(s,a) = R_a(s,s') + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')$$
$$\gamma = 0.5$$

- Random Initial State $< 0,2 >$
- Update $\hat{Q}(< 0,1 >, east) = 25$
- Update $\hat{Q}(< 1,0 >, east) = 100$



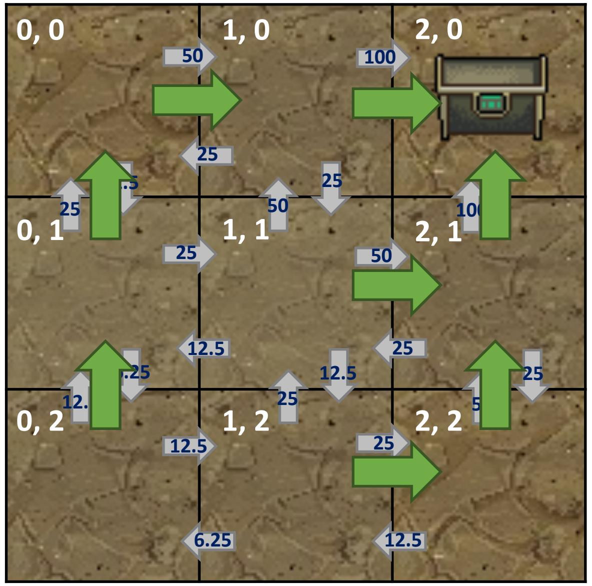
credit: Peter Bodí

Example of Q learning
($\hat{Q}$ after many, many runs…)

- $\hat{Q}$ converged

- Policy is:

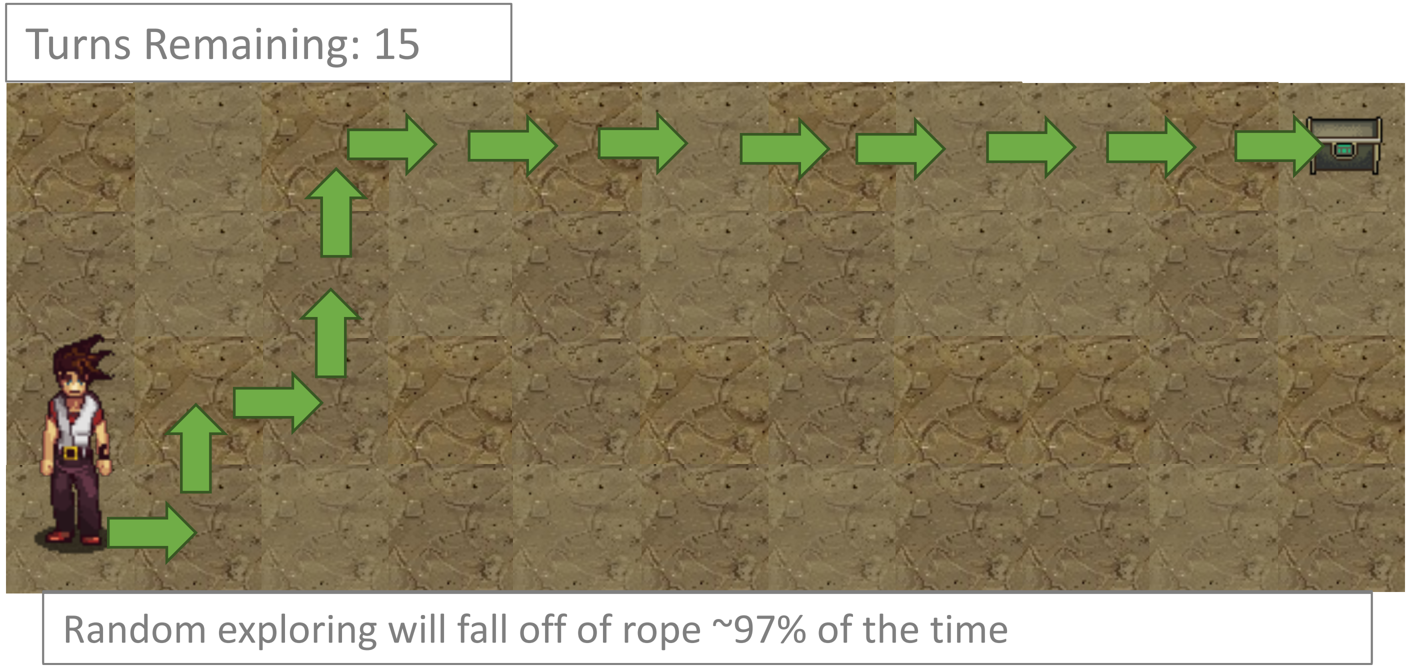
$$\pi(s) = \underset{a \epsilon A_s}{\operatorname{argmax}} \hat{Q}(s, a)$$



credit: Peter Bodí

Challenges for Reinforcement Learning

- When there are many states and actions
- When the episode can end without reward
- When there is a 'narrow' path to reward

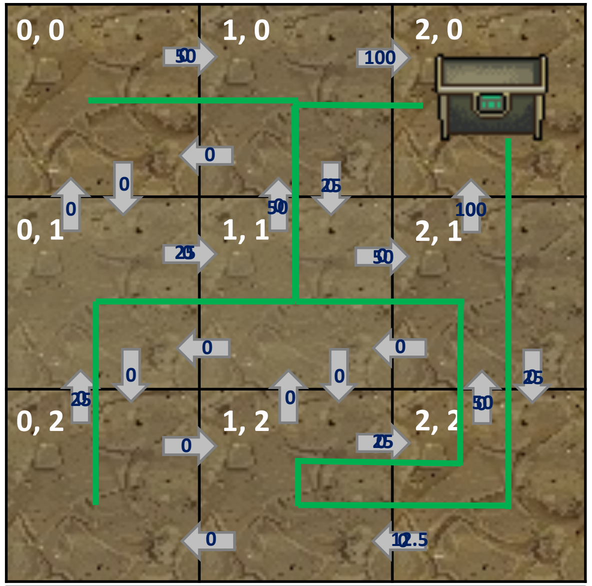


credit: Peter Bodí

Memory

- Retrain on previous explorations
  - Maintain samples of:
$$P_a(s, s')$$
$$R_a(s, s')$$

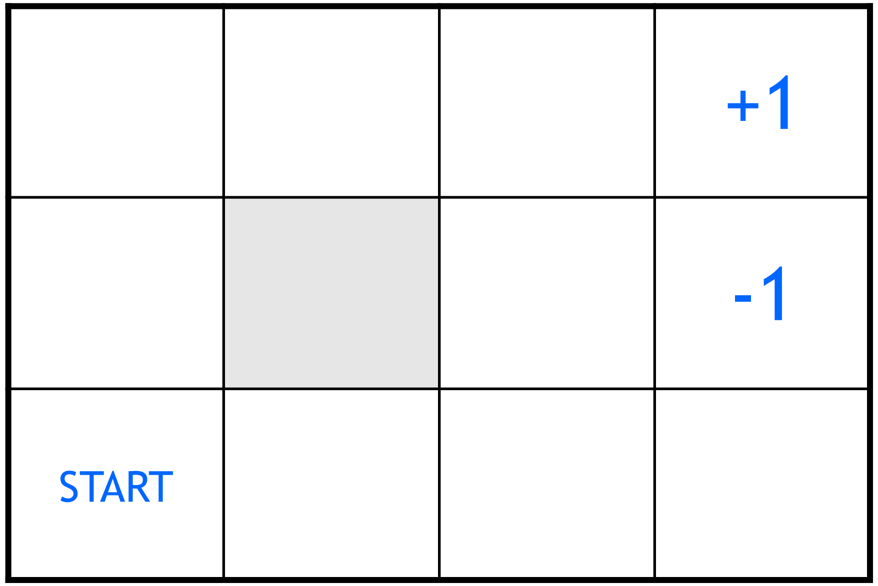
- Useful when
  - It is cheaper to use some RAM/CPU than to run more simulations
  - It is hard to get to reward so you want to leverage it for as much as possible when it happens



Replay it a bunch of times

credit: Peter Bodí

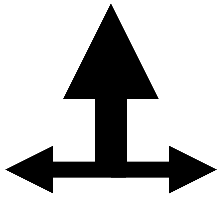
Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

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



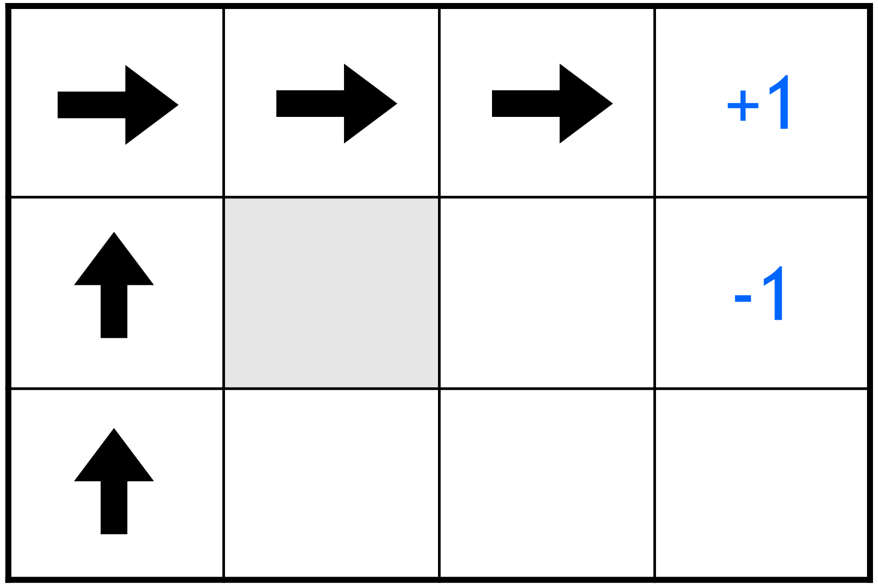
reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards

- what is the solution?

credit: Peter Bodí

Is this a solution?



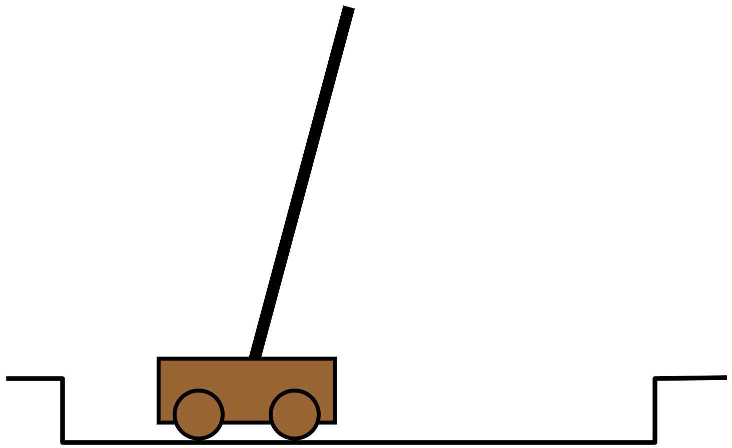
- only if actions deterministic
  - not in this case (actions are stochastic)

- solution/policy
  - mapping from each state to an action

credit: Peter Bodí

State representation

- pole-balancing
  - move car left/right to keep the pole balanced
- state representation
  - position and velocity of car
  - angle and angular velocity of pole
- what about *Markov property*?
  - would need more info
  - noise in sensors, temperature, bending of pole
- solution
  - coarse discretization of 4 state variables
    - left, center, right
  - totally non-Markov, but still works

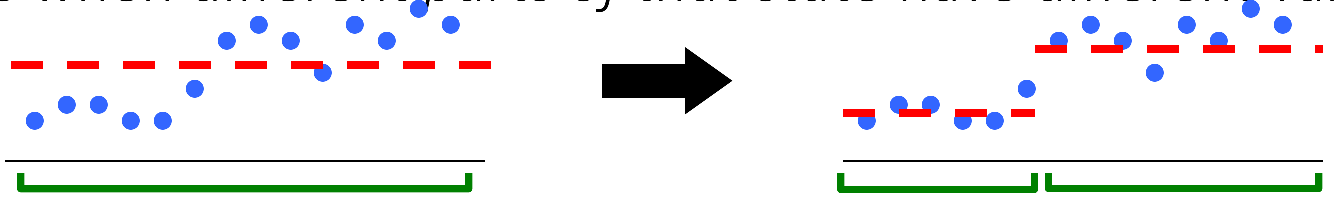


credit: Peter Bodí

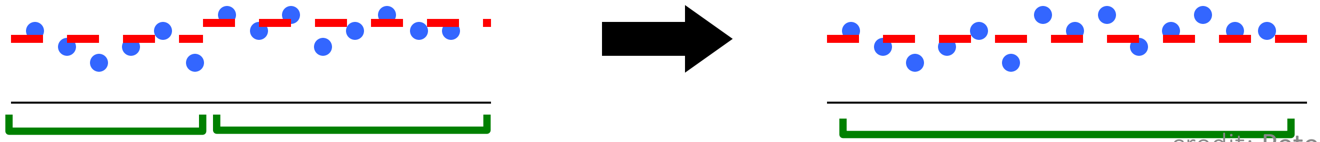
Splitting and aggregation

- want to discretize the state space
  - learn the best discretization during training

- splitting of state space
  - start with a single state
  - split a state when different *parts of that state* have different values



- state aggregation
  - start with many states
  - merge states with similar values



credit: Peter Bodí

Designing rewards

- robot in a maze
  - episodic task, not discounted, +1 when out, 0 for each step
- chess
  - GOOD: +1 for winning, -1 losing
  - BAD: +0.25 for taking opponent's pieces
    - high reward even when lose
- rewards
  - rewards indicate what we want to accomplish
  - NOT how we want to accomplish it

- shaping
  - positive reward often very "far away"
  - rewards for achieving subgoals (domain knowledge)
  - also: adjust initial policy or initial value function

credit: Peter Bodí