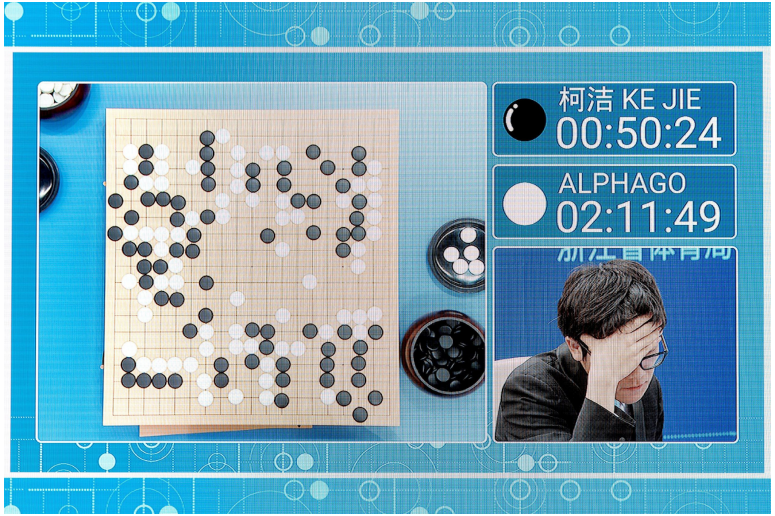
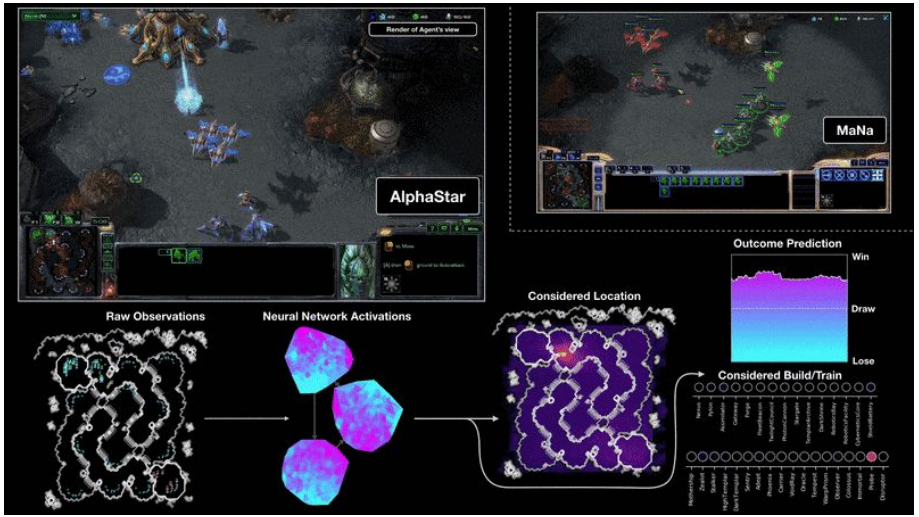
Introduction to Reinforcement Learning

Simon Fan
Ashish Gaurav
UWDSC W19

Lots of hype


柯洁 KE JIE
00:50:24
ALPHAGO
02:11:49


AlphaStar
MaNa
Raw Observations
Neural Network Activations
Considered Location
Outcome Prediction
Win
Draw
Lose
Considered Build/Train

Brief timeline

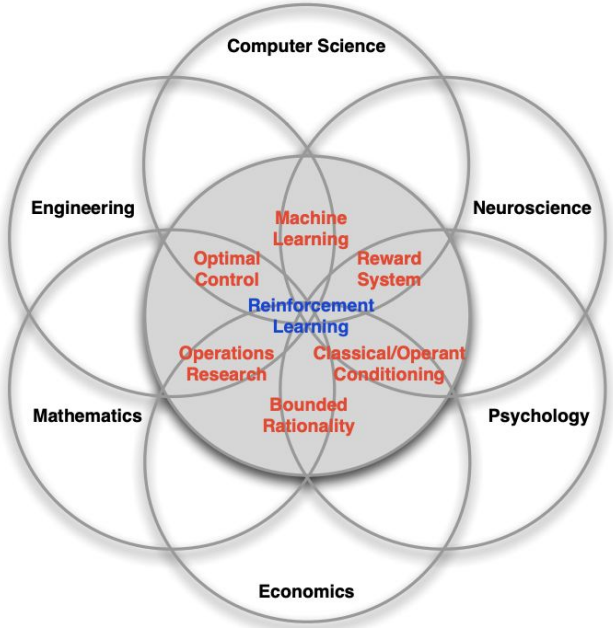
- 1963: RL plays tic-tac-toe
- 1970: AI winter
- 1989: Q-learning
- 2010: Deep Learning
- 2016: AlphaGo (supervised learning + deep RL + monte carlo + tricks)
- 2017: AlphaGo Zero (deep RL + monte carlo + more tricks)
- 2019: AlphaGo Star (supervised learning + deep RL + monte carlo + more and more tricks)

What is RL?

Train an **agent**, who interacts with an **environment** by performing **actions** which result in **rewards**.

A **policy** (which action to perform and when) is learnt by the agent and we can compute the cumulative reward the agent receives by following the policy.

The goal of RL is for the agent to learn the **optimal policy** to follow, in order to maximize the cumulative reward (value).



Contrast with other forms of learning

Suppose there is a distribution **X** with true labels **y**, we approximate the label mapping function with:

- Supervised learning
  - Given ***m*** samples of <***x***,***y***>, assume correct
  - find ***f: x***→***y*** w.r.t. some objective function
- Unsupervised learning
  - Given ***m*** samples of ***x***
  - find ***f: x***→ ***y***
- Reinforcement learning
  - Given ***m*** samples of ***x*** and some reward function ***z***, assume correct
  - learn ***f: x***→***y*** w.r.t. the reward function ***z***, i.e. maximize the cumulative reward
  - ***z*** looks a lot like ***y*** in supervised learning

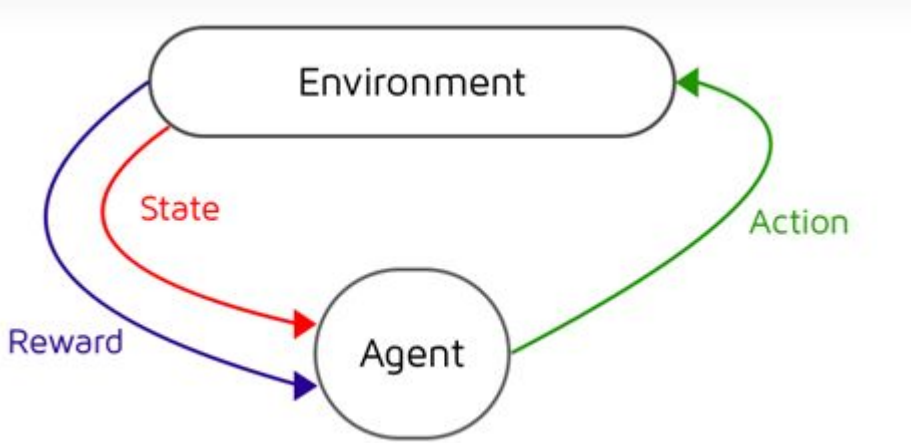
Parallel with Psychology

Positive reinforcement (positive reward)

Negative reinforcement (negative reward)

RL definitions

- Agent: the one who interacts with the environment
- State (*s*): unique, observed status of the environment
- Possible actions (A): state to actions mapping
- Action (*a=A(s)*): action that the agent selected
- Reward (*R*): <state, action> to reward mapping
- Policy (π): state to action mapping

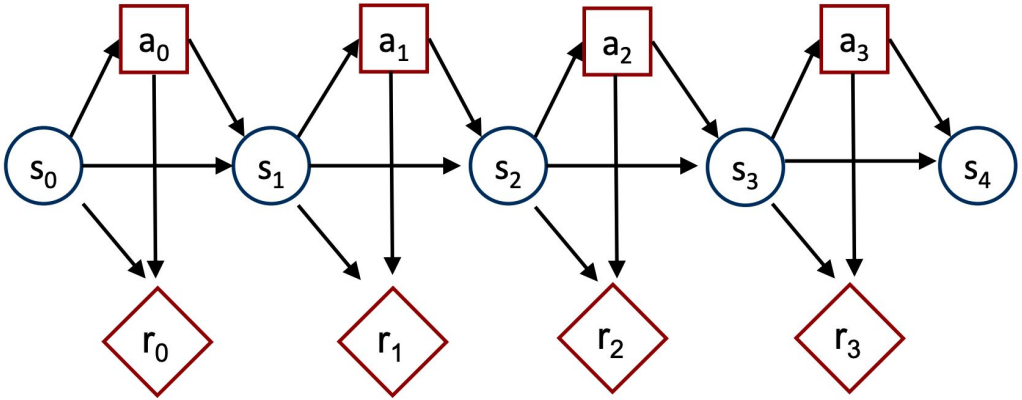


RL definitions

Formally it's a MDP

Markov Decision Process

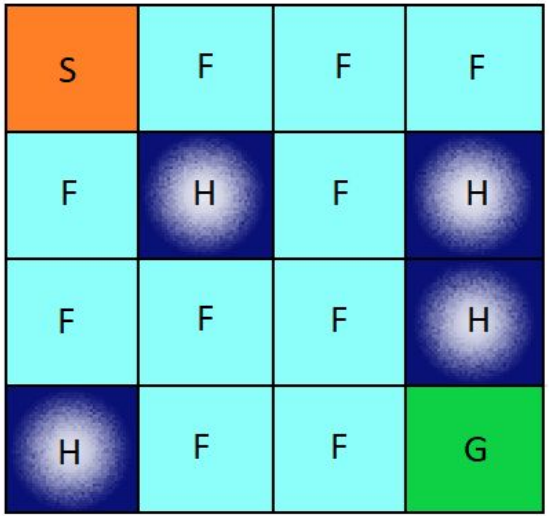
- Markov process augmented with…
  - Actions e.g., a_t
  - Rewards e.g., r_t



An example: walking on a lake

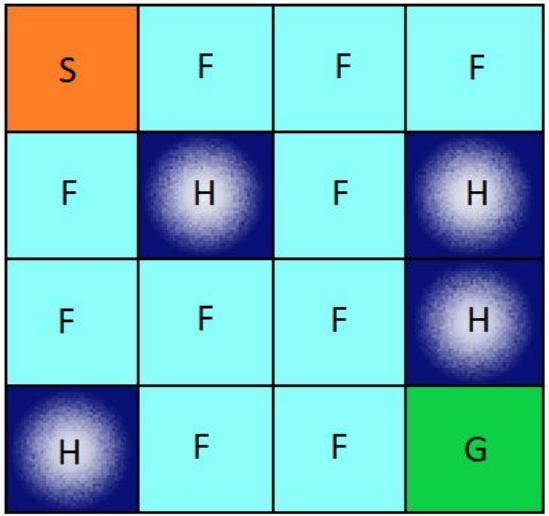
- States: Each index is a state
  - S: start state
  - F: walkable surface
  - H: game ends
  - G: goal state
- Reward: 1 when goal is reached, 0 otherwise
- Agent: Tries to get from S to G such that reward is maximized
- Actions: Up, Down, Left, Right

What is the optimal policy? Are there more than one?



An example: Frozen Lake

- States: Each index is a state
  - S: start state
  - F: walkable surface, but slippery
  - H: game ends
  - G: goal state
- Reward: 1 when goal is reached, 0 otherwise
- Agent: Tries to get from S to G such that reward is maximized
- Actions: Up, Down, Left, Right



A lot harder to write a non RL solver

DEMO IN OPENAI GYM! https://gym.openai.com/

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $temp \leftarrow v(s)$
         $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s))\Big[r(s, \pi(s), s') + \gamma v(s')\Big]$
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $temp \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s, a)\Big[r(s, a, s') + \gamma v(s')\Big]$
      If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

Figure 4.3: Policy iteration (using iterative policy evaluation) for v_*. This algorithm has a subtle bug, in that it may never terminate if the policy continually switches between two or more policies that are equally good. The bug can be fixed by adding additional flags, but it makes the pseudocode so ugly that it is not worth it. :-)

finding optimal value function

Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
      $temp \leftarrow v(s)$
      $v(s) \leftarrow \max_a \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v(s')]$
      $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\pi(s) = \arg\max_a \sum_{s'} p(s'|s, a)\Big[r(s, a, s') + \gamma v(s')\Big]$

Figure 4.5: Value iteration.

one policy update (extract policy from the optimal value function

Uncovered, but important topics

- Continuous state spaces
- Continuous action spaces
- Model vs Model-free
- On-policy vs off-policy
- Exploration vs exploitation tradeoff
  - Superstitions
  - “Solution”: epsilon-greedy, upper confidence bounds, ...
- Reward tuning
  - Horizon
  - Credit assignment
  - “Solutions”: shaping rewards, discount rate, … (careful environment design)