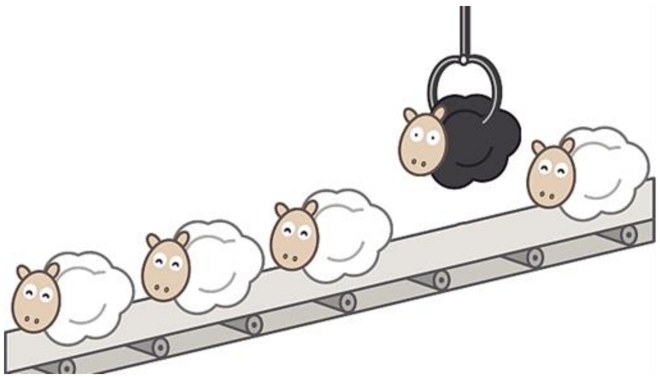
Machine Learning – COMS3007

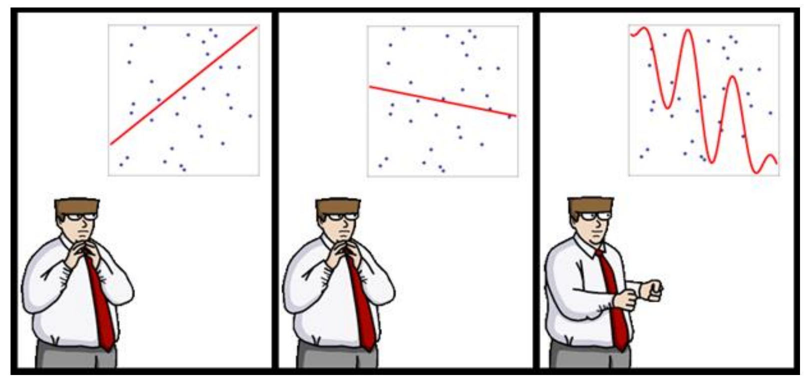
Reinforcement Learning

Benjamin Rosman

Previously on ML…

- We’ve seen how to solve many cool problems around supervised and unsupervised learning





- Why do we make predictions on data?
    - Usually so a human can **make better decisions**
- Decision making itself is important in intelligence
    - How do we automate this?

And now for something completely different…

- Reinforcement learning is the branch of machine learning relating to learning in **sequential decision making settings**

- Also think of as behaviour learning

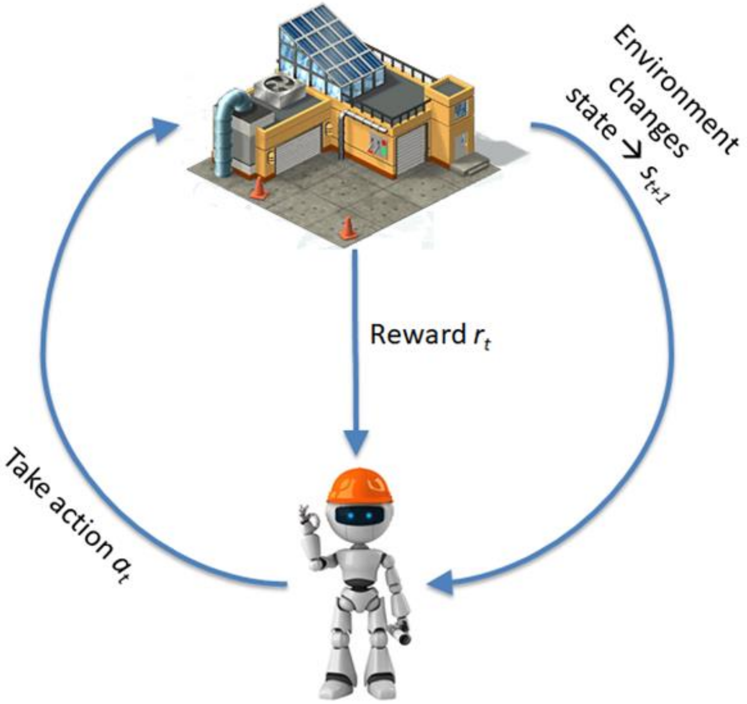
- Supervised learning: single decision point

- Multiple decision points
  - How do I know if I'm doing the right thing?
  - How do my decisions now impact the future?
  - Actions affect the environment!

When do we need this?

Interacting with the environment

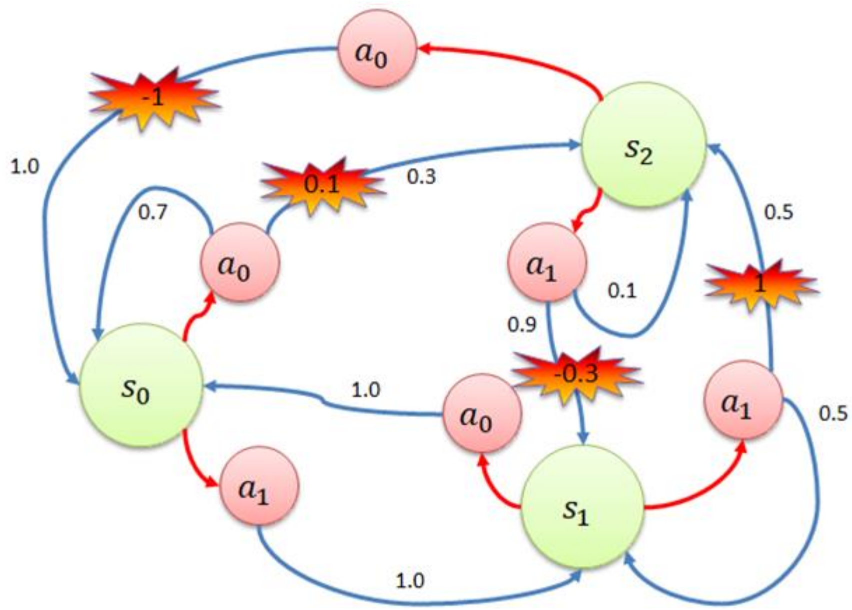
- Decision maker (agent) exists within an environment
- Agent takes action a_t based on the environment state s_t
- Environment state updates $s_{t+1} \leftarrow s_t$
- Agent receives feedback as rewards r_t



Modeling the problem

"Future is independent of the past, given the present"

- Markov Decision Process (MDP)



M = ⟨***S***, ***A***, ***T***, ***R***⟩

- States: encode world configurations
- Actions: choices made by agent
- Transition function: how the world evolves under actions

$$T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$

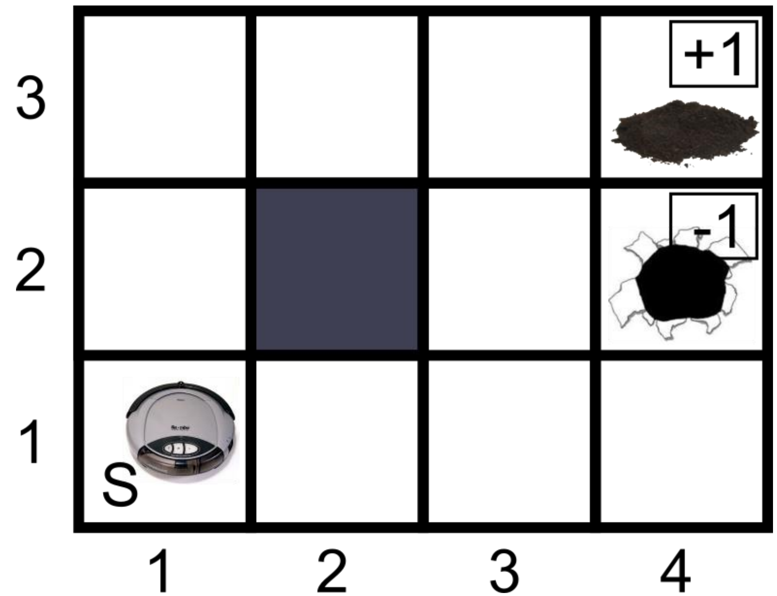
- Rewards: feedback signal to agent

$$R(s, a) = E[r_t | s_t = s, a_t = a]$$

E[.] = "expected" (think of as the average reward)

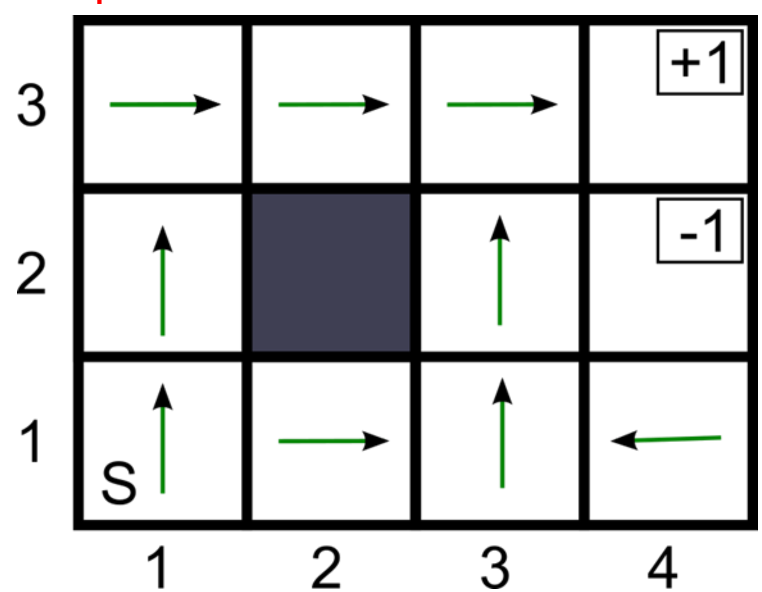
An example

- Cleaning Robot
- States:
  - Position on grid e.g. S is (1,1), goal (4,3)



- Actions: 
- Reward:
  - +1 for finding dirt
  - -1 for falling into hole
  - -0.001 for every move

Optimal behaviour:



Policies

- A **policy** (or behaviour or strategy) $\boldsymbol{\pi}$ is any mapping from states to actions
  - Deterministic or stochastic

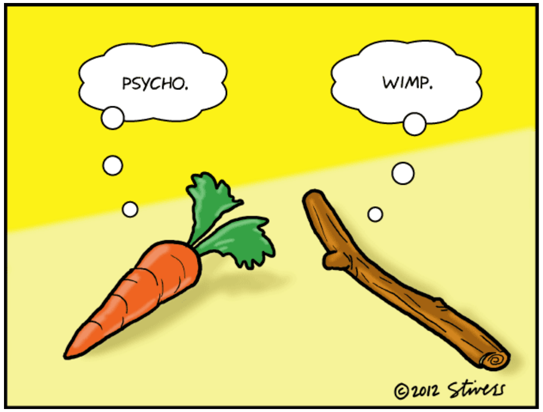
$$\pi(a|s) = P(a_t = a|s_t = s)$$

- Optimal policy π*
  - Accumulates maximal rewards over a trajectory
  - This is what we want to learn!

Rewards

Scalar feedback signal

Encode (un)desirable features of behaviours:
Winning/losing, collisions, taking expensive actions, ...



But, typically:

- Sparse
- Delayed

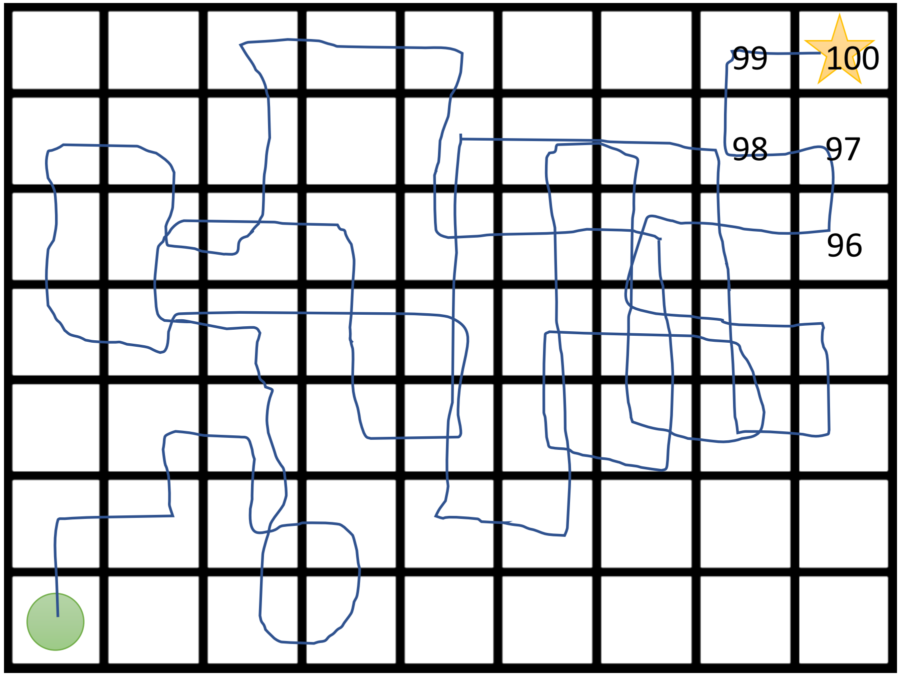
The Rats of Hanoi

Immediate vs delayed rewards

- Cannot just rely on the instantaneous reward function
  - Tradeoff: don't just act myopically (short term)

- Notion of **value** to codify the goodness of a state, considering a policy running into the future
  - Represented as a value function

Value functions



Value functions

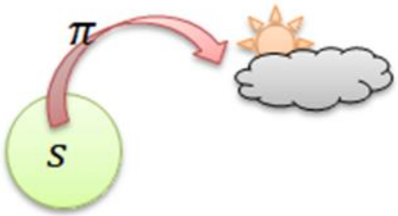
- Value function:
  - The expected return (R) starting at state s and then executing policy π

accumulated reward

Discounting: future values count less

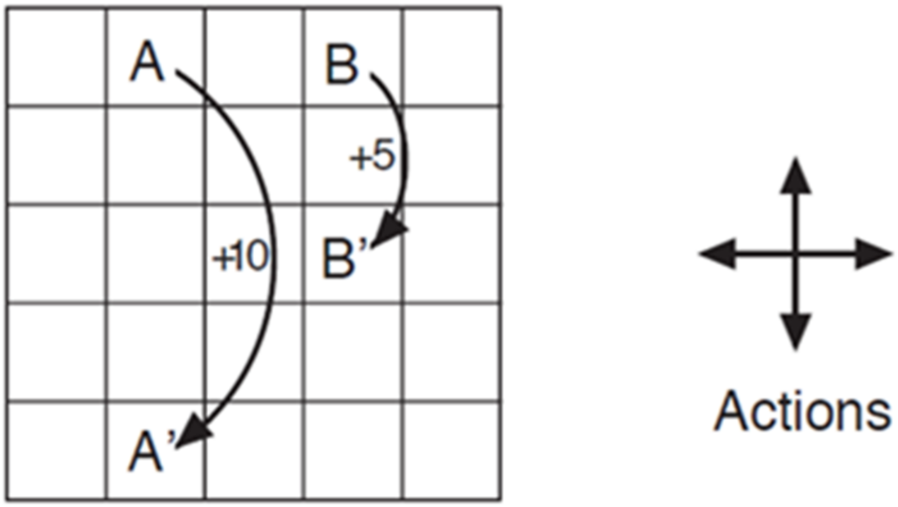
$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t, s_{t+1})\}$$

  - "How good is s under π?"



Example

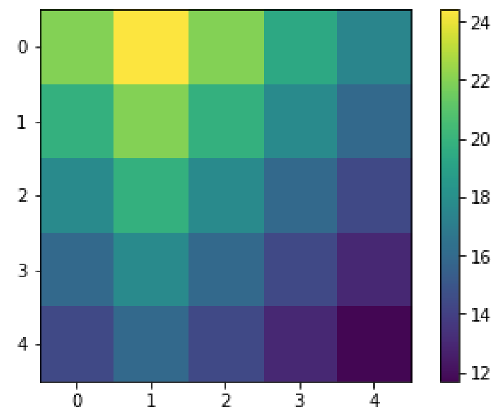
- Reward -1 for every move

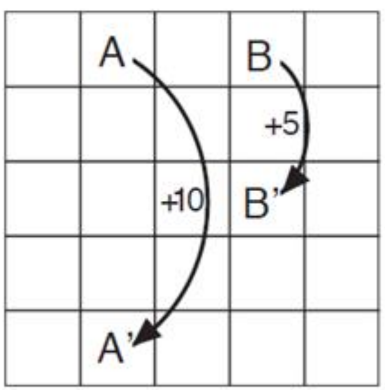


(a)

Example

- Optimal policy

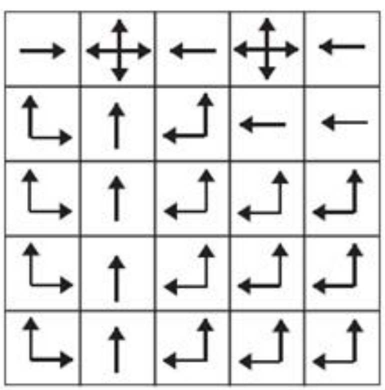




a) gridworld

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_*

Value functions: recursion

- V(s) ⇒ expected return starting at s and following π
  - Suggests dependence on V(s') from next state s'

- Bellman Equation:

$$\boxed{V^{\pi}(s)} = \boxed{R(s, \pi(s))} + \gamma \boxed{\sum_{s'}} \boxed{T(s, \pi(s), s')} \boxed{V^{\pi}(s')}$$

value of s

immediate reward

for all possible next states

the probability of reaching that state with π

value of s'

Value functions: optimality

- Similarly, for an optimal policy π* with optimal value function V*:

- Bellman Optimality Equation:

$$\boxed{V^*(s)} = \boxed{\max_a}\{R(s,a) + \gamma \sum_{s'} T(s,a,s') \boxed{V^*(s')}$$

take the best possible action

- Note: the optimal value function V* gives us the optimal policy π*
  - Choose the action that leads to the best next state

Solving Bellman

- Given the Bellman equation:

$$V^*(s) = \max_a \{R(s,a) + \gamma \sum_{s'} T(s,a,s')V^*(s')$$

- Solve this as a large system of value function equations
  - But: non-linear (max operator)
  - So: solve iteratively

- What are we trying to do here?
  - Learn how good each state of the world is, when looking perfectly into the future

Dynamic programming

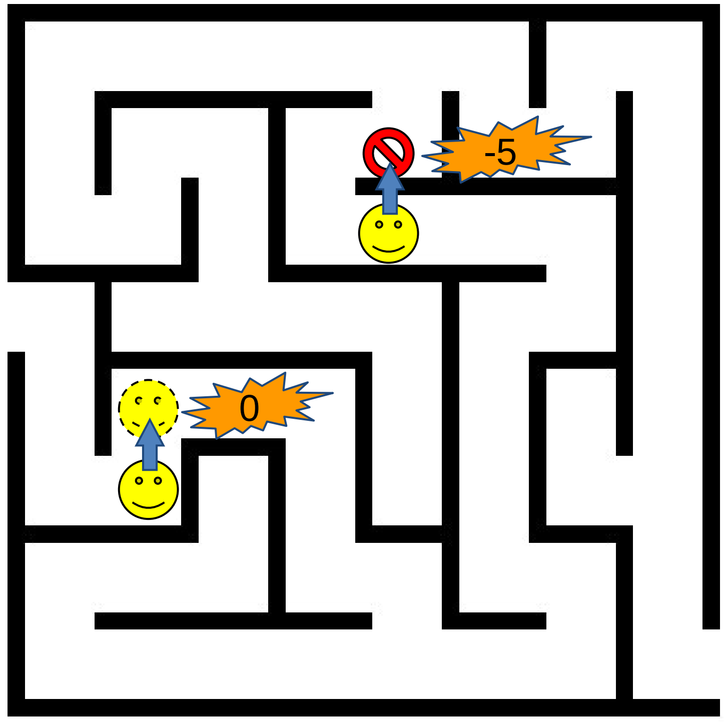
- Value Iteration: dynamic programming
- Iteratively update V (synchronous version)
  - At each iteration i:
    - For all states $s \in S$:
      - Update $V(s)$:

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V_i(s') \right) \right\}$$

- But: this requires the full MDP!!
  - In general, T and R are unknown

Learning from experience

- T and R unknown!
- Instead, generate samples of training data (s, a, r, s') from environment
- Learn from experience
- We need:
  - A way to choose actions
  - A model to store knowledge
    - Value function



Action selection

- How do we collect data from the environment?
  - Run the best policy we have at the moment
  - But how does that learn anything new??

- Exploration/exploitation tradeoff!
  - Sometimes exploit what we have already learned
  - Other times try something new

- ϵ-Greedy $(0 < \epsilon \leq 1)$:
  - With probability $1 - \epsilon$ **exploit**
    - Choose the best action for a state from π
  - With probability ϵ **explore**
    - Randomly choose action

Temporal difference learning

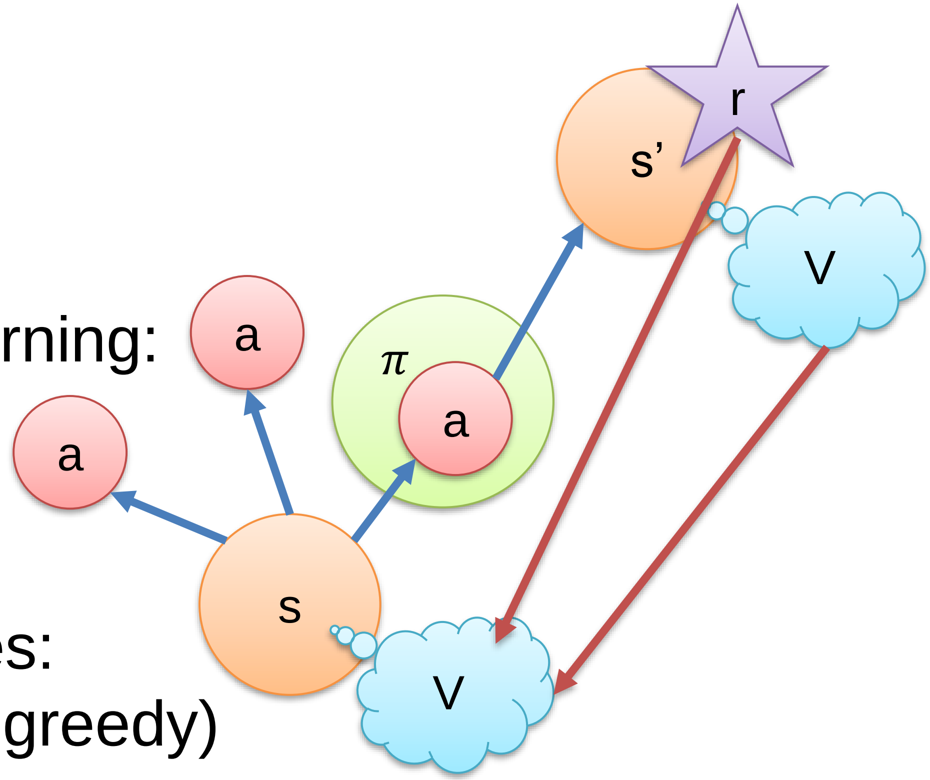
How to learn:

Update toward estimated return

$$V(S_t) \leftarrow V(S_t) + \alpha({\color{red}R_{t+1} + \gamma V(S_{t+1})} - V(S_t))$$

Update estimate of goodness of current state

TD learning



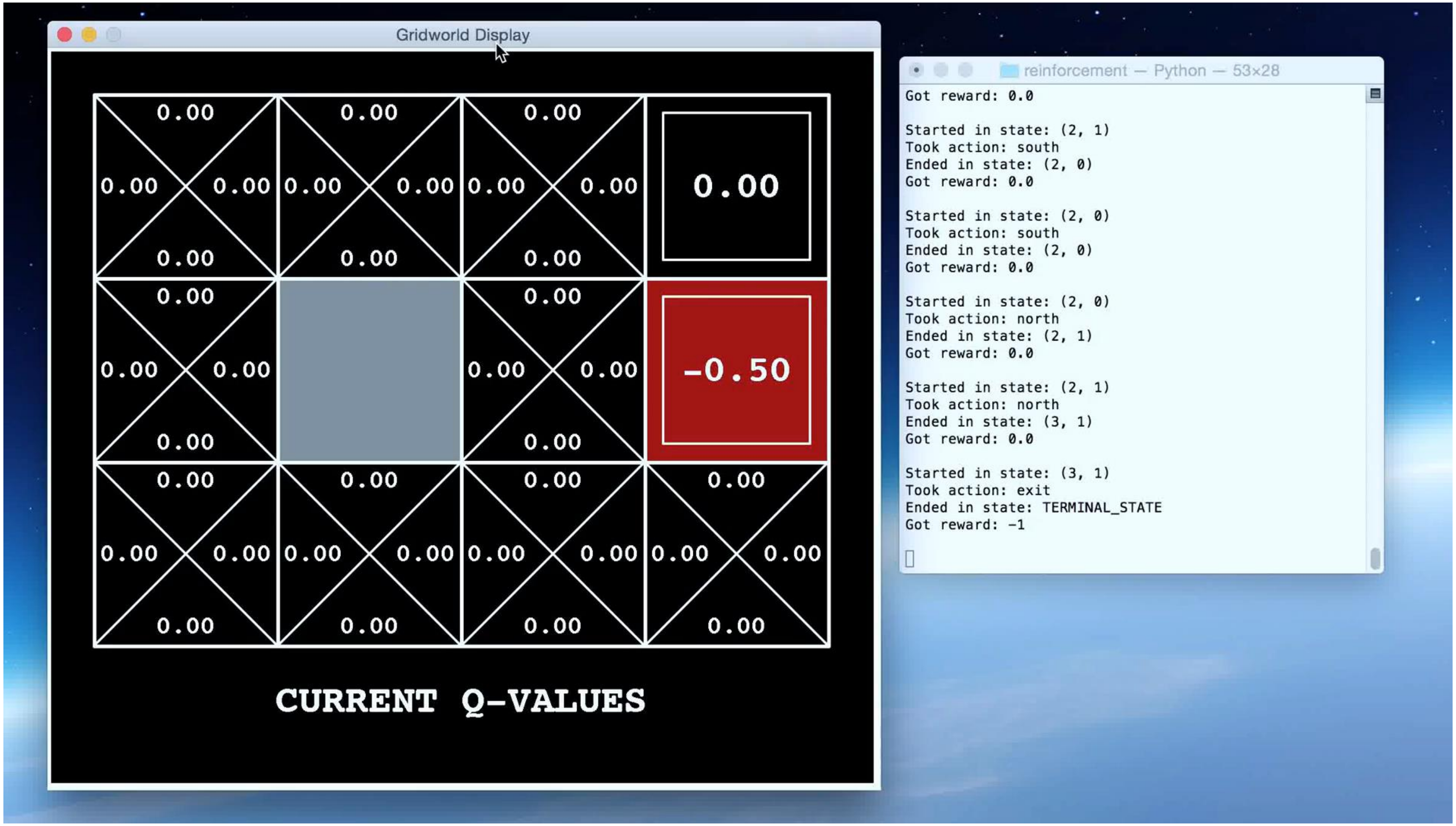
- Temporal Difference (TD) Learning:
  - Initialise V for all $s \in S$
  - For each episode:
    - Reset state s
    - Until episode terminates:
      - Choose action a (ϵ-greedy)
      - Execute a
      - Receive new state s' and reward r
      - Update V using (s, r, s'):

$$V_{i+1}(s) \leftarrow V_i(s) + \alpha\underbrace{(\overbrace{r + \gamma V_i(s')}^{learnt\ value} - V_i(s))}_{\text{TD error}}$$

Learning rate

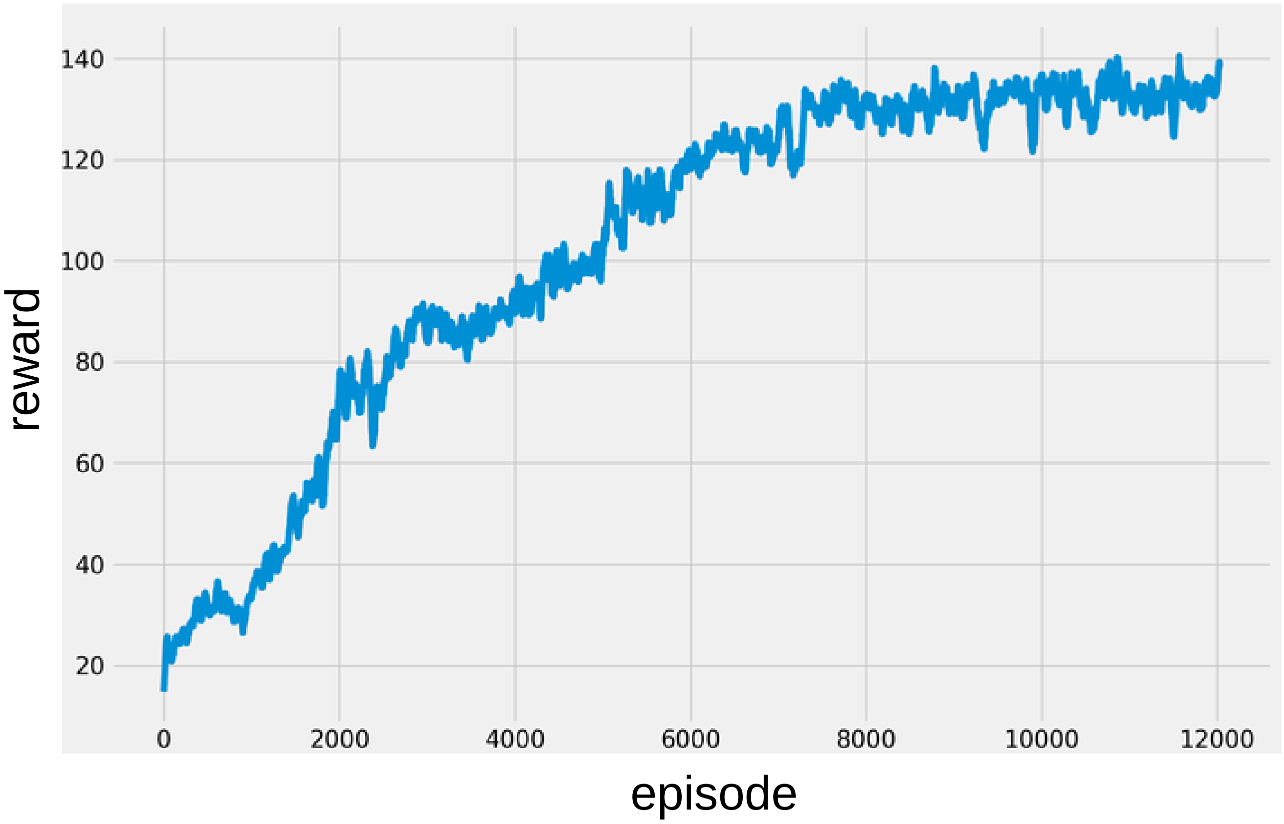
      - $s \leftarrow s'$

Q-Learning demo



Shreyas Skandan: https://www.youtube.com/watch?v=RTu7G0y4Os4

Learning curves



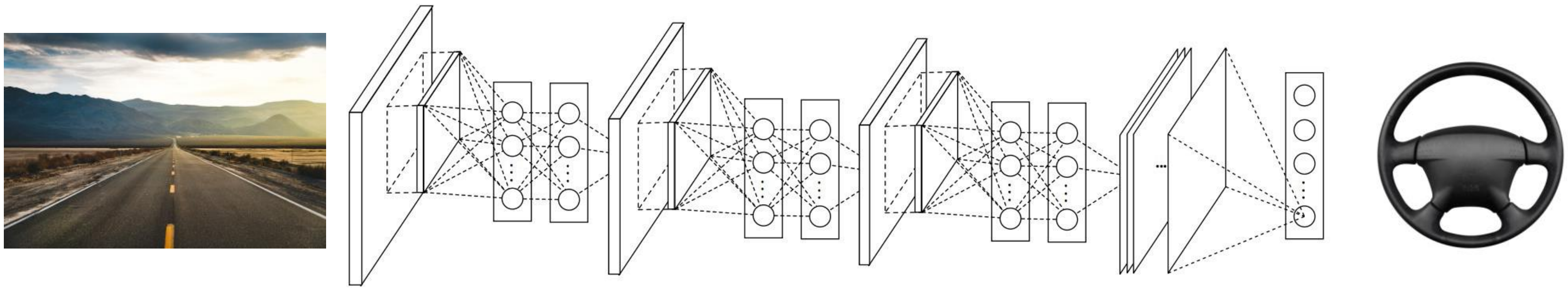
But the world is continuous!

Function approximation

Instead of learning the best action for every state…

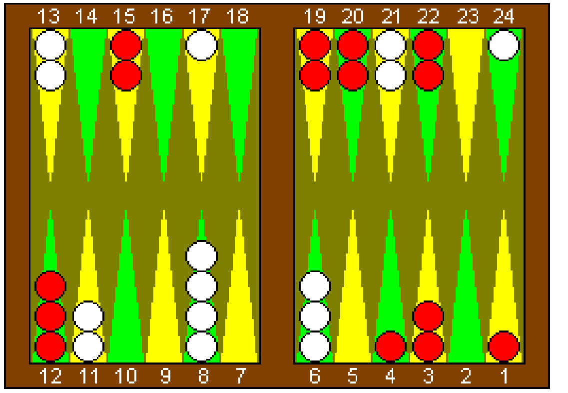
Use a **neural network** to learn a representation of the value function

- i.e. a mapping from states to value/actions



Import the whole deep learning toolbox into RL

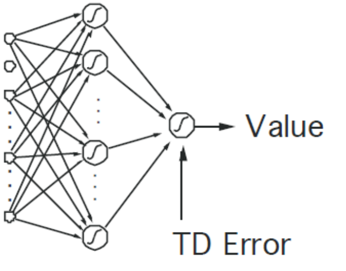
Backgammon



TD-Gammon: Tesauro (1992-1995)

- Learn to play Backgammon through self-play
- 1.5 million games
- Neural network function approximator

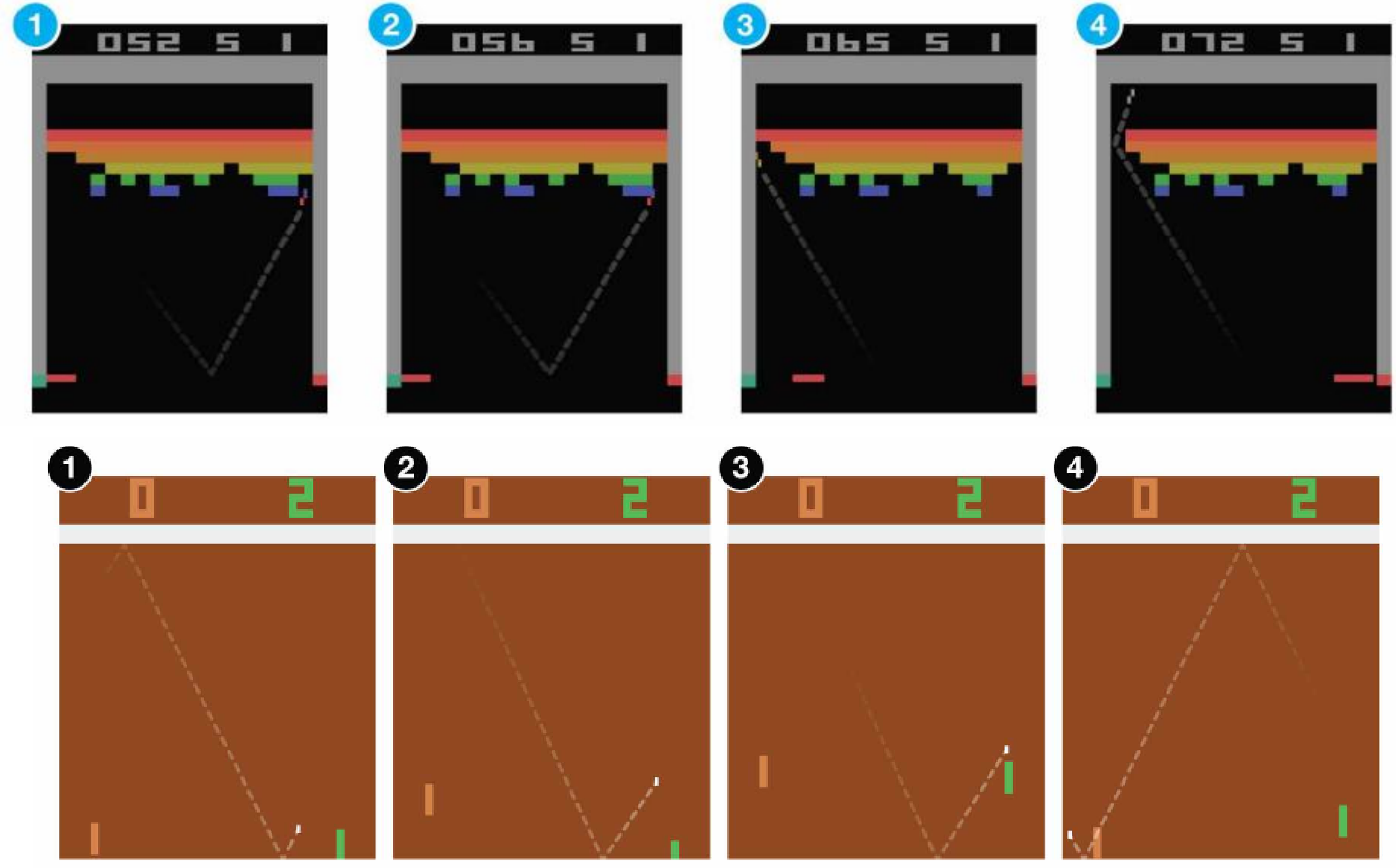
States = board configurations ($\approx 10^{20}$)
Actions = moves
Rewards = $\begin{cases} 1 \; win \\ -1 \; lose \\ 0 \; else \end{cases}$



At/near best human play

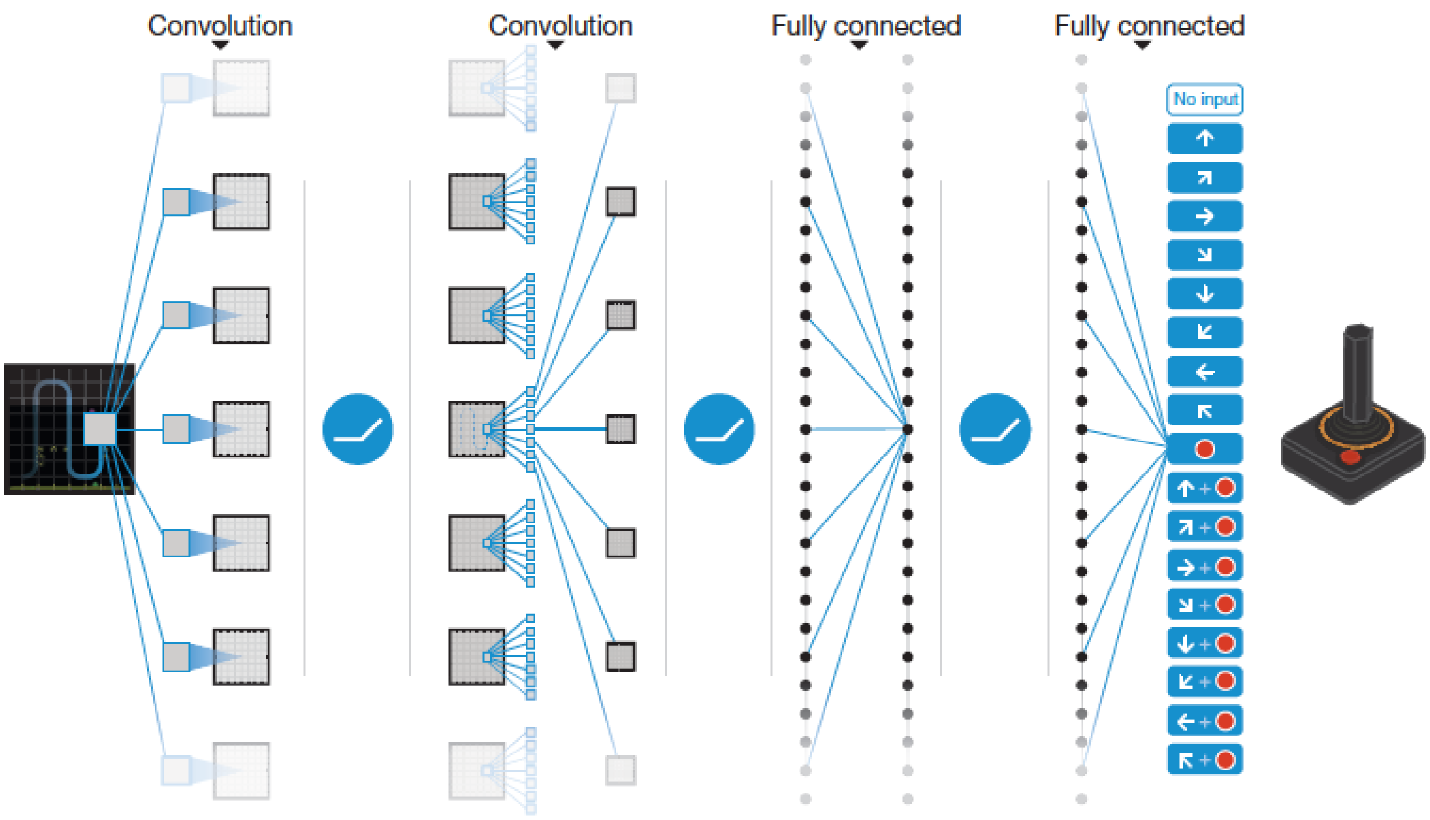
Changed the way the best human players played

Arcade Learning Environment



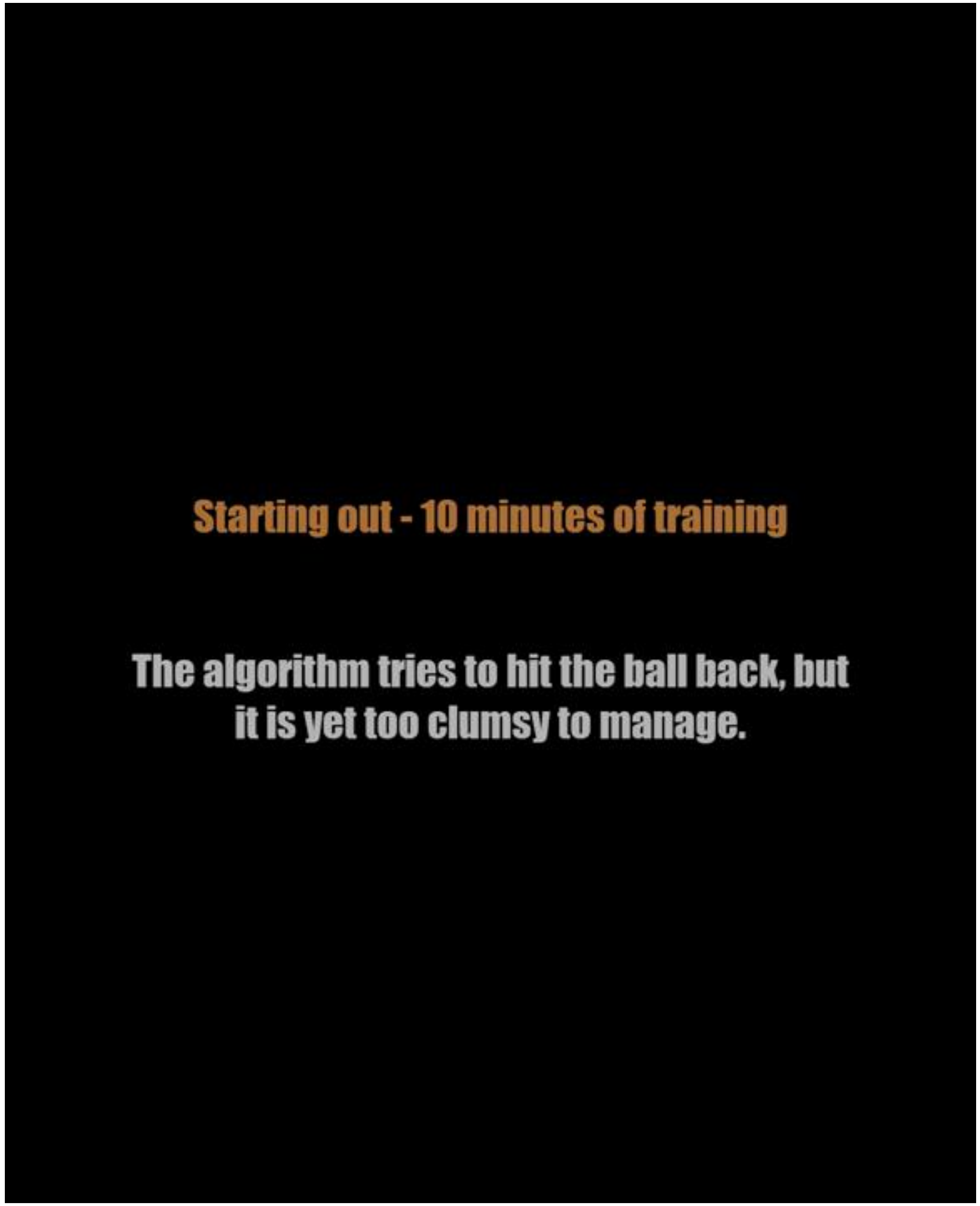
[Bellemare 2013]

Deep Q-Networks



[Mnih et al., 2015]

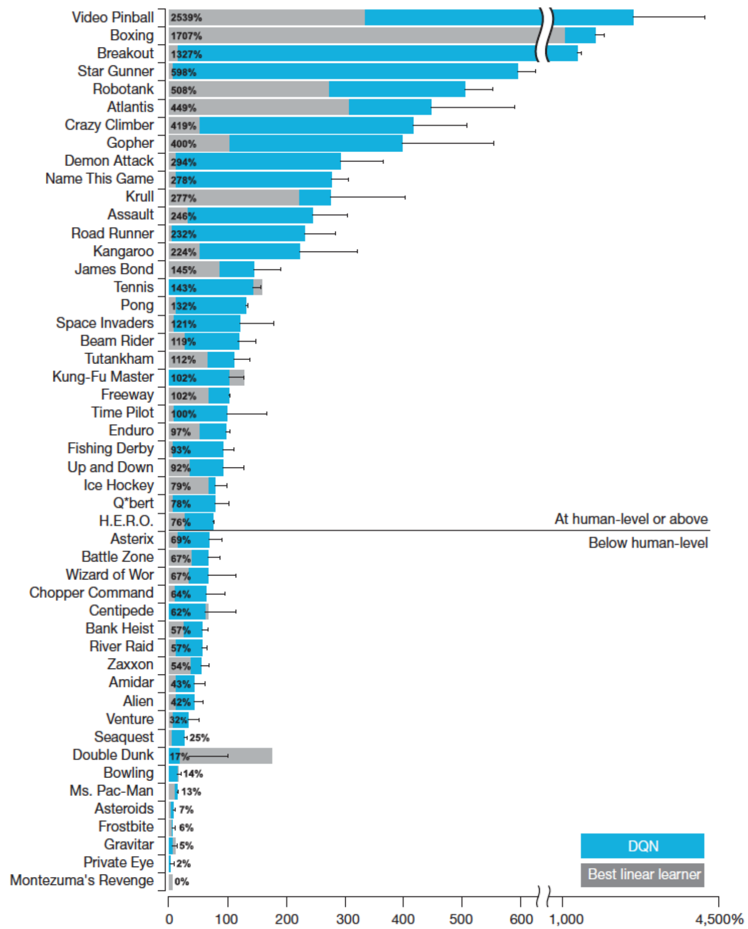
Atari



Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. *Nature*, *518*(7540), pp.529-533.

video: Two Minute Papers

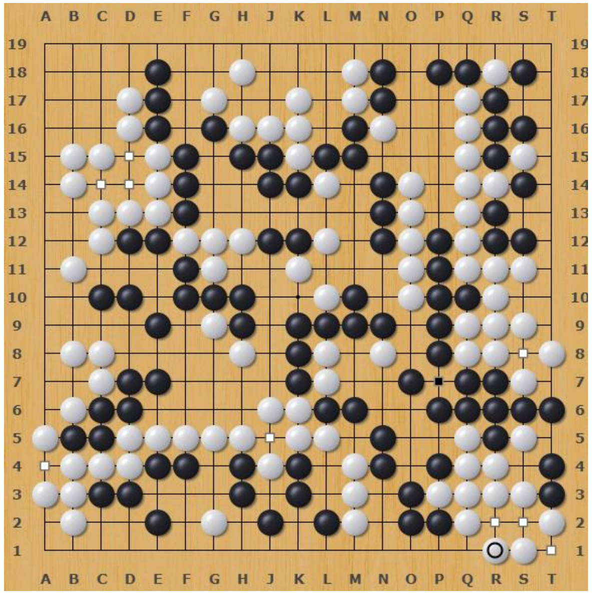
Atari results



Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. *Nature*, *518*(7540), pp.529-533.

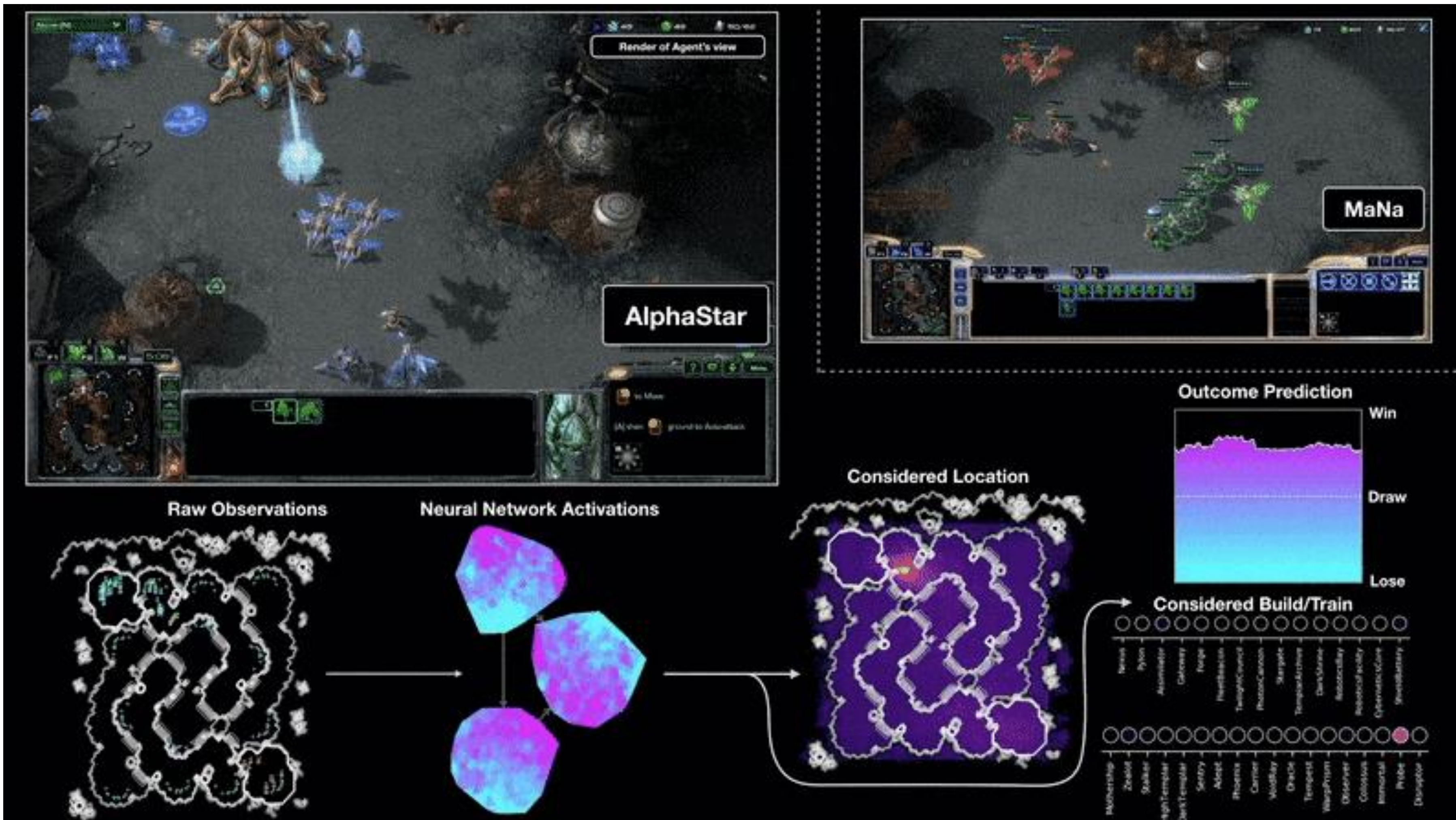
But these are simple games!

- Go
  - 361 moves
  - $\sim 10^{174}$ states
  - Adversarial!



- What is the complexity of real-world decisions?

Just a game?



Render of Agent's view
AlphaStar
MaNa
Outcome Prediction
Win
Draw
Lose
Considered Location
Raw Observations
Neural Network Activations
Considered Build/Train

Increasing complexity

Information Type	Near-Perfect	Perfect	**Imperfect**
Players	Single player	**Multiplayer**	**Multiplayer**
Action Space	17	**361**	**$\sim 10^{26}$**
Moves per game	100's	100's	**1000's**

Recap

- Sequential decision making
- Agent vs environment
- Markov Decision Process
- Policies, rewards, value functions
- Dynamic programming: value iteration
- TD learning
- Function approximation

Also watch the AlphaGo documentary:
https://www.youtube.com/watch?v=WXuK6gekU1Y