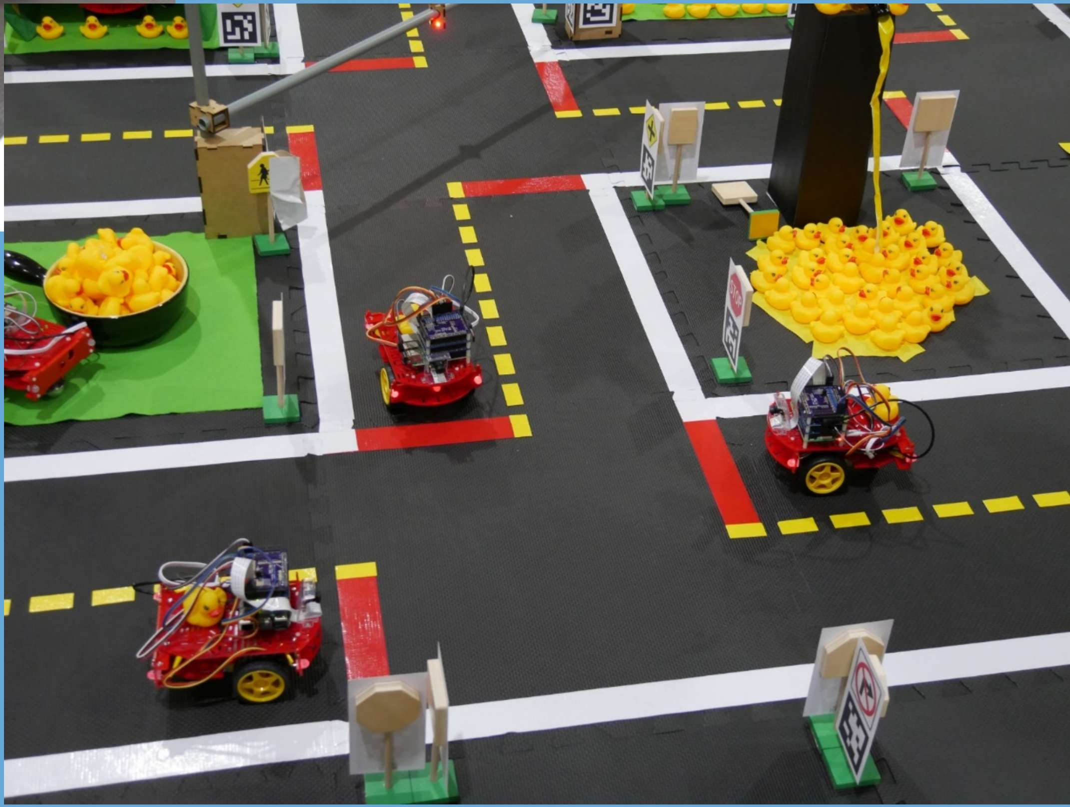
CS 3630



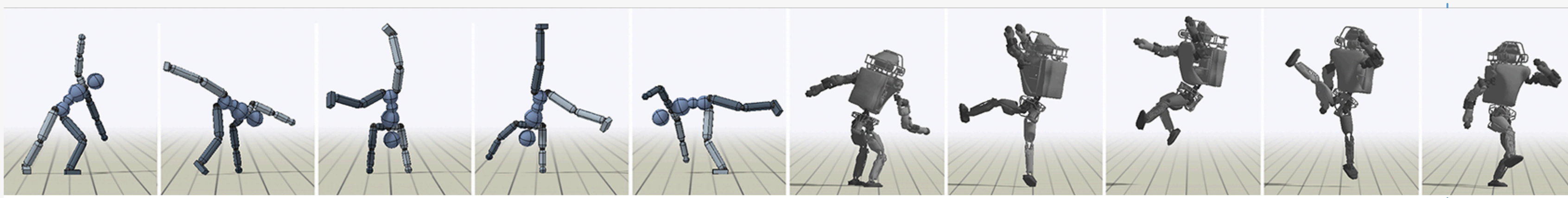
Lecture 24:

A brief introduction to Deep Reinforcement Learning

This lecture borrows heavily from the *Deep RL Boot Camp* slides, in particular the slide decks by Peter Abbeel, Rocky Duan, Vlad Mnih, Andrej Karpathy, and John Schulman

Topics

1. **Recap: MDP and RL Methods**
2. **Deep Q-Learning**
3. **Policy Optimization**



Motivation

- Deep learning success in perception
- MDP and RL frameworks
  - Well understood
  - Early successes (backgammon)
  - Not great on more complex problems
- Can deep learning make RL really work?
  - Evidence points to yes!

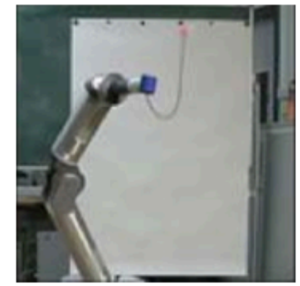
Image from DeepMimic paper, Peng et al 2018

Some RL Success Stories

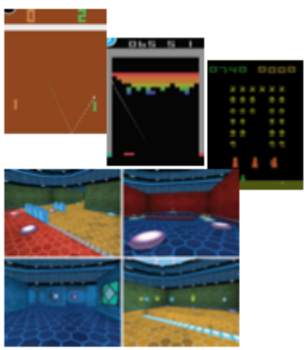
Kohl and Stone, 2004

Ng et al, 2004

Tedrake et al, 2005

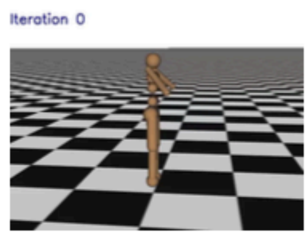


Kober and Peters, 2009

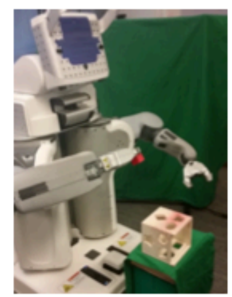


Mnih et al 2013 (DQN)
Mnih et al, 2015 (A3C)

Silver et al, 2014 (DPG)
Lillicrap et al, 2015 (DDPG)



Schulman et al, 2016 (TRPO + GAE)



Levine*, Finn*, et al, 2016 (GPS)

Silver*, Huang*, et al, 2016 (AlphaGo)

Overview

This lecture will *not* go in depth

Provide rough overview, pointers to excellent starting points

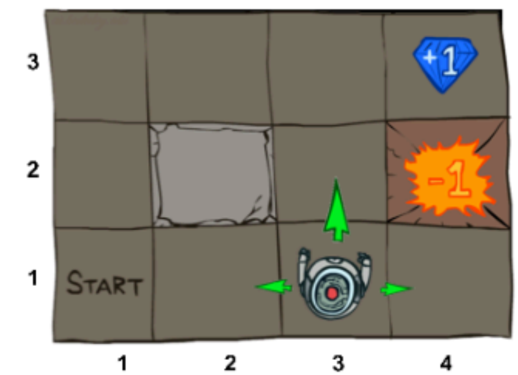
Lecture materials all cribbed from Deep RL Boot Camp, took place August 2017 in Berkeley

All slides/lectures are online: https://sites.google.com/view/deep-rl-bootcamp/home

Recap: Markov Decision Processes (MDP)

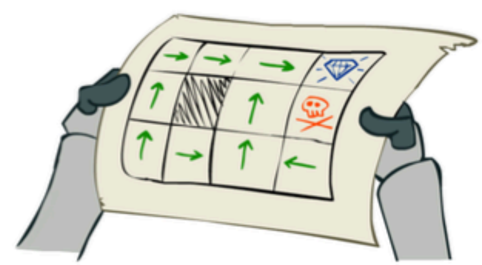
An MDP is defined by:

- Set of states S
- Set of actions A
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



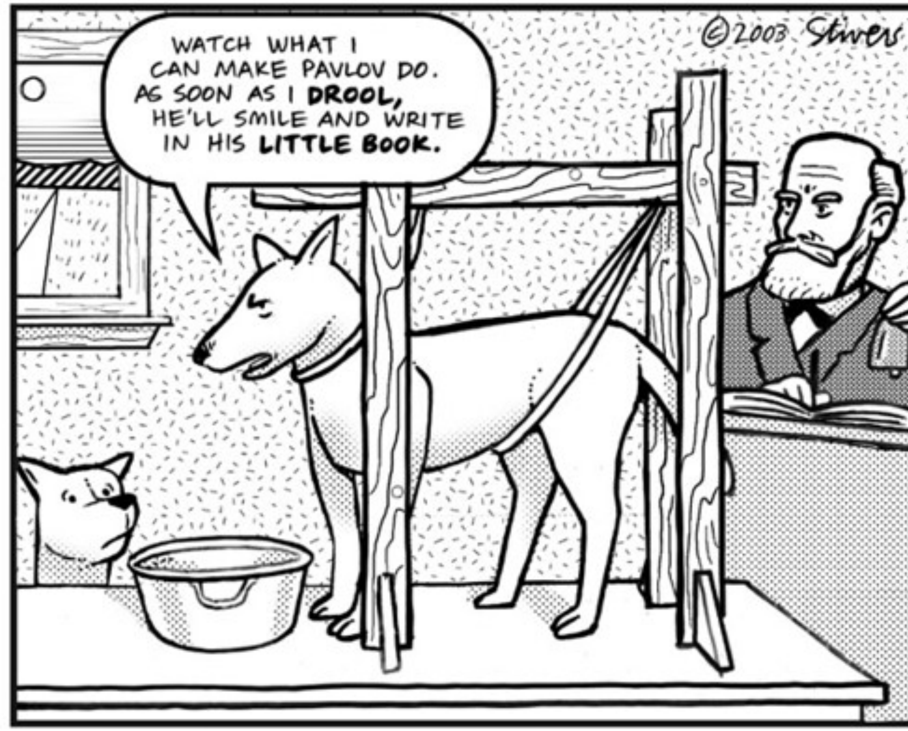
Goal: $max_\pi \mathrm{E}[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1}) | \pi]$

π:



* *P(s'|s,a) == T(s,a,s')* from Seth's lectures

A note on Rewards



- Three different ways to do the reward:
  - *R(s)*: reward is function of state only (Seth's lectures)
  - *R(s, a)*: reward in given state depends on action you take
  - *R(s, a, s')*: reward *also* depends on in what state you land
    - Most general
    - Matches Deep RL Bootcamp slides

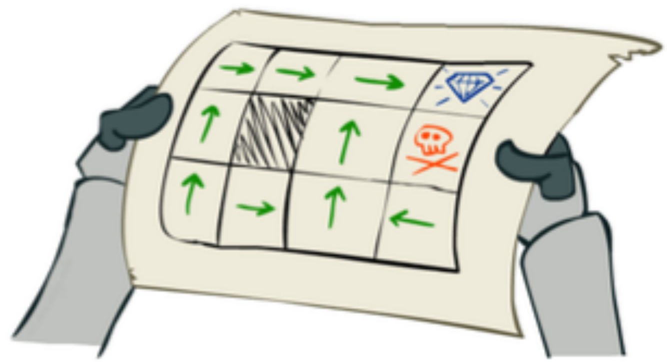
Recap: Policy and Value Function

- Policy:
  - *S -> A*
  - *Optimal* π^*
- Value function:
  - Satisfies Bellman equations $V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$ if actions $a(s) = \pi^*(s)$, i.e. actions are chosen chosen according to optimal policy π^*:

    Optimal value $V^*(s)$

- Exercise: check Bellman equations with given *noise 0.2* (actions only work 80% of the time) and *discount factor* 0.9

π:



Noise =
Discount

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, ... , H:

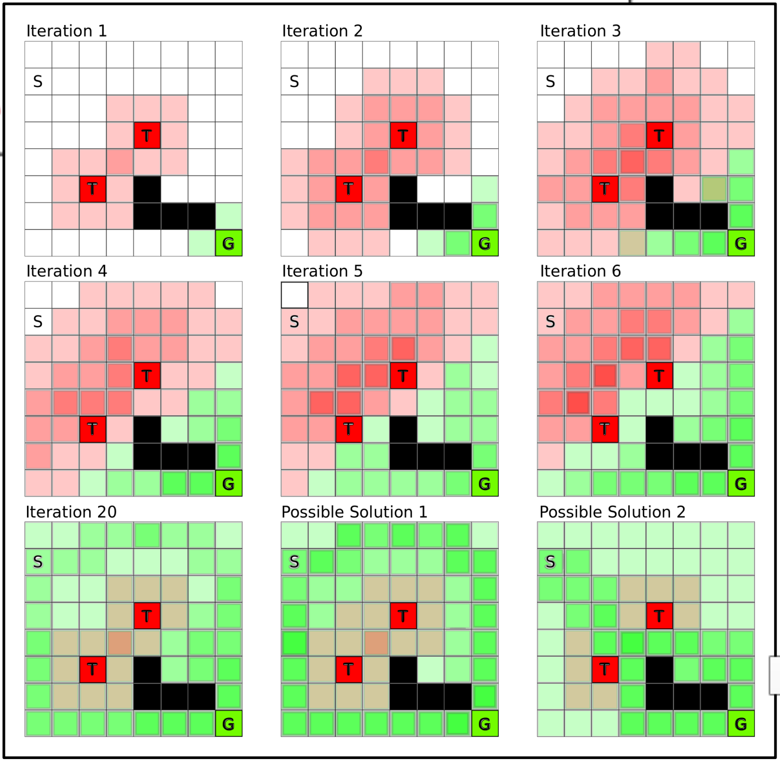
For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

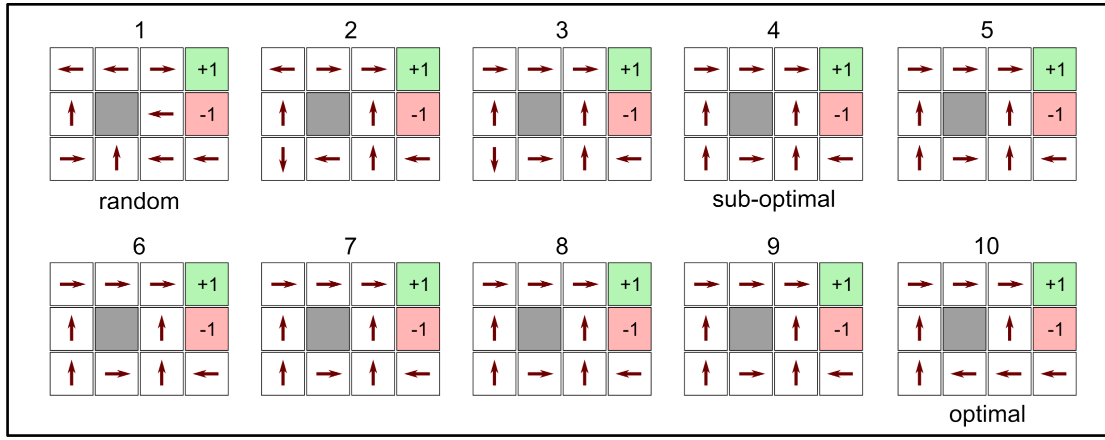
This is called a value update

Recap: Value Iteration



[Image borrowed from (useful!) Nvideo blog post on RL: https://devblogs.nvidia.com/deep-learning-nutshell-reinforcement-learning/](https://devblogs.nvidia.com/deep-learning-nutshell-reinforcement-learning/)

Recap: Policy Iteration



- Policy evaluation for current policy π_k :
  - Iterate until convergence

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s,\pi_k(s)) \left[R(s,\pi(s),s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: find the best action according to one-step look-ahead

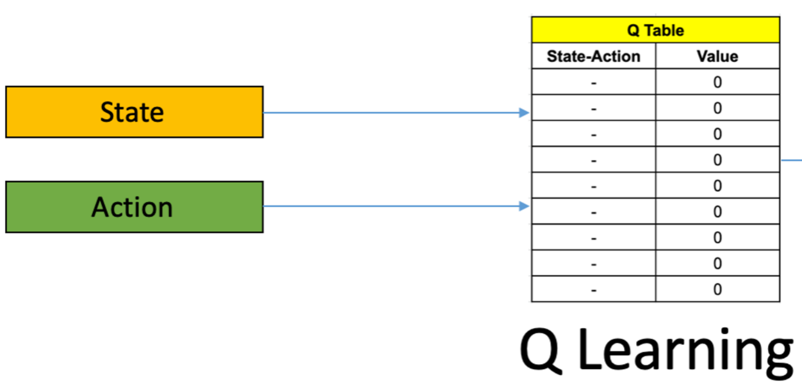
$$\pi_{k+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V^{\pi_k}(s')\right]$$

- Repeat until policy converges
- At convergence: optimal policy; and converges faster than value iteration under some conditions

Image borrowed from (useful!) blog post at https://mpatacchiola.github.io/blog/2016/12/09/dissecting-reinforcement-learning.html

Recap: Reinforcement *Learning*

- Passive:
  - Direct utility estimation
  - Adaptive Dynamic Programming
  - Temporal Difference Learning (model-free!)
- Active:
  - Model-based: exploitation vs. exploration
  - Q-learning (model-free!)



$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q^*_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

Image by Praphul Singh, https://blogs.oracle.com/datascience/reinforcement-learning-deep-q-networks

2. Deep Q-learning

A simple (tabular) Q-learning algorithm:

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s’

If s’ is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s’

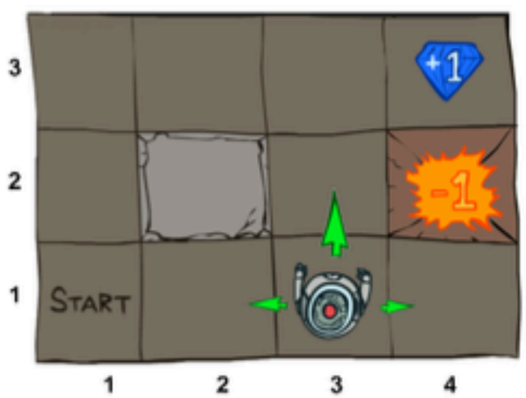
else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

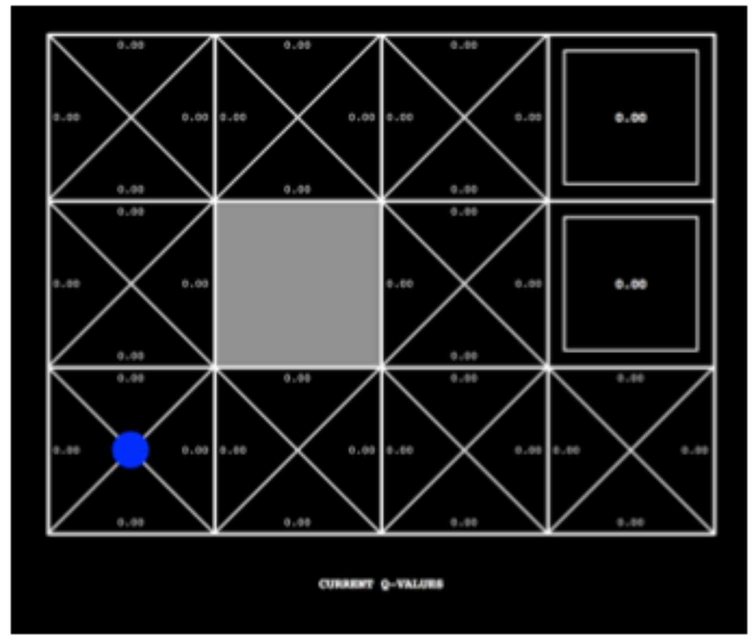
$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \, [\text{target}]$$

$$s \leftarrow s'$$

Q-learning on gridworld

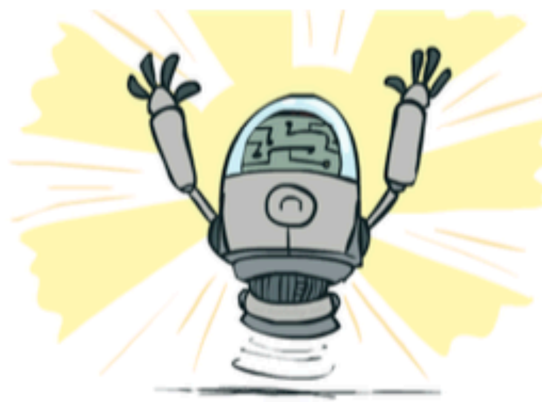


- States: 11 cells
- Actions: {up, down, left, right}
- Deterministic transition function
- Learning rate: 0.5
- Discount: 1
- Reward: +1 for getting diamond, -1 for falling into trap



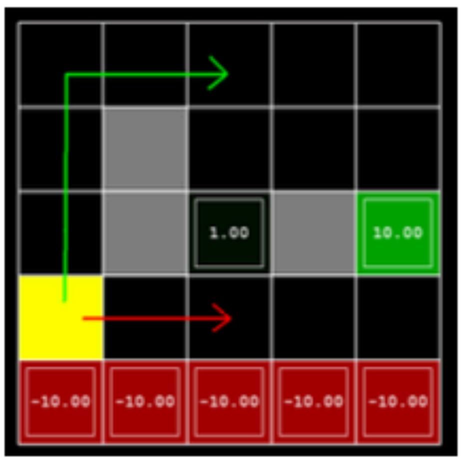
Q-learning properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly

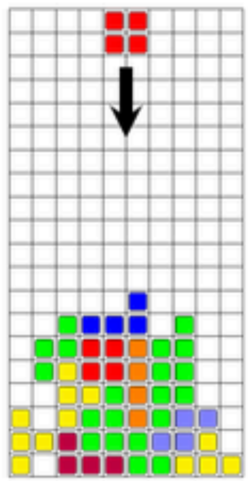


Can tabular Q-learning scale?

- Discrete environments



Gridworld
10^1

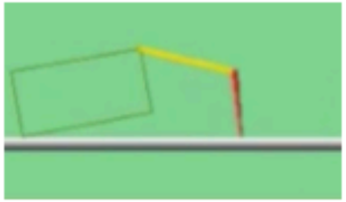


Tetris
10^60

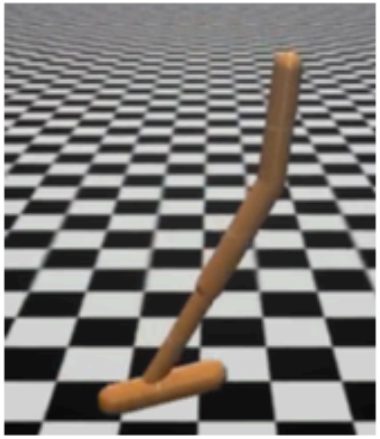
Atari
10^308 (ram)   10^16992 (pixels)

Can tabular Q-learning scale?

- Continuous environments (by crude discretization)



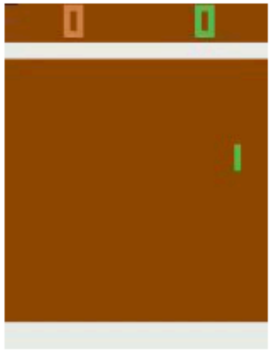
Crawler
10^2



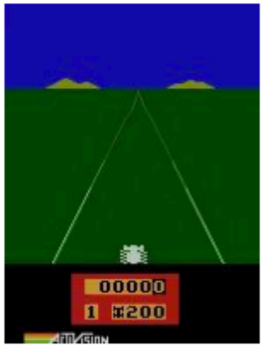
Hopper
10^10

Humanoid
10^100

Enter DQN (2015 *Nature* paper, Mnih et al)

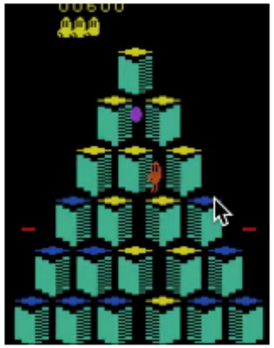


Pong



Enduro

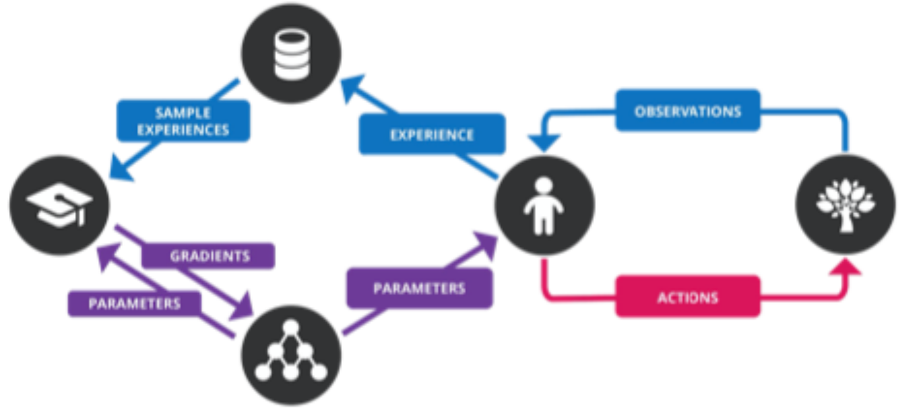
Beamrider



Q*bert

- 49 ATARI 2600 games.
- From pixels to actions.
- The change in score is the reward.
- Same algorithm.
- Same function approximator, w/ 3M free parameters.
- Same hyperparameters.
- Roughly human-level performance on 29 out of 49 games.

DQN overview



- High-level idea - **make Q-learning look like supervised learning**.
- Two main ideas for stabilizing Q-learning.
- Apply Q-updates on batches of past experience instead of online:
  - **Experience replay** (Lin, 1993).
  - Previously used for better data efficiency.
  - Makes the data distribution more stationary.

- Use an older set of weights to compute the targets (**target network**):
  - Keeps the target function from changing too quickly.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r+\gamma \max_{a'} Q(s',a';\theta_i^-)}_{\text{target}} - Q(s,a;\theta_i)\right)^2$$

"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (20

Neural Network to approximate Q-values

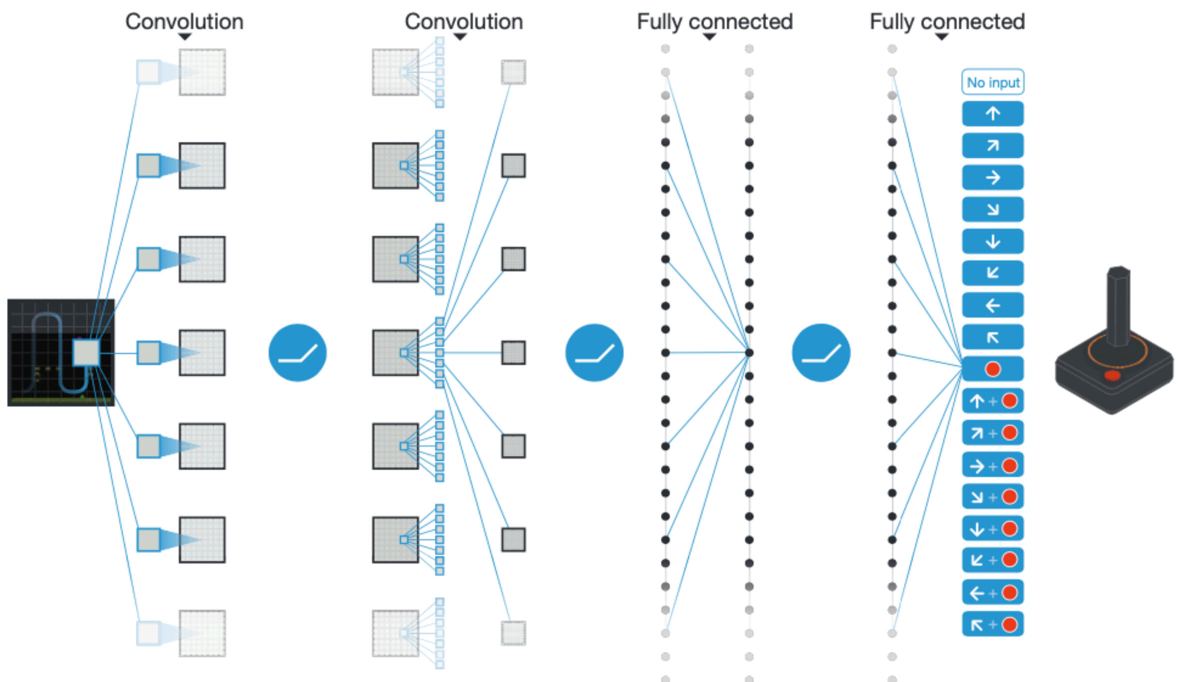
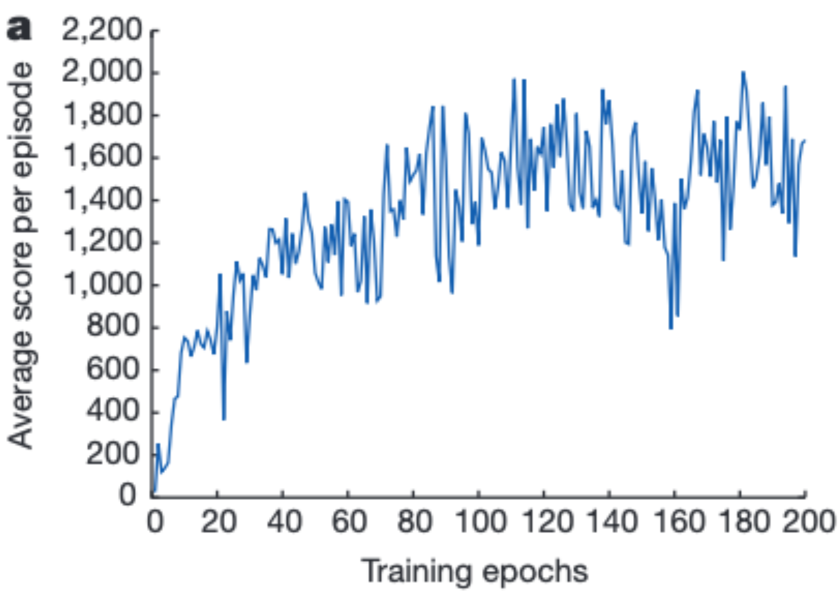


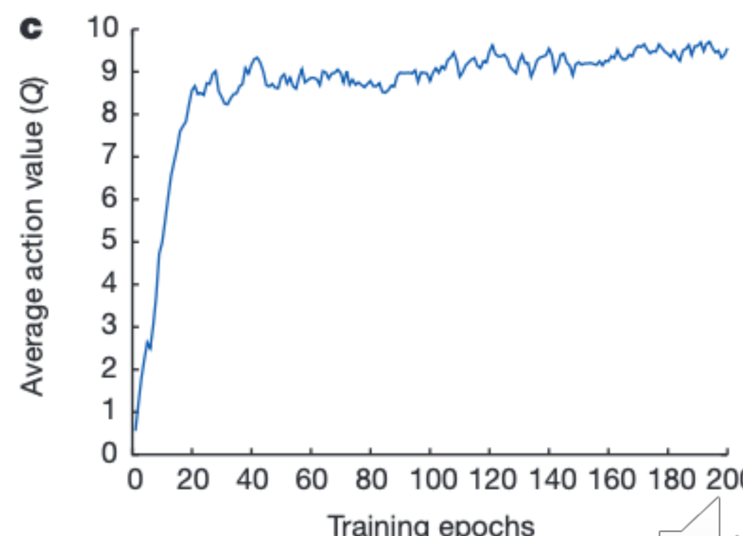
Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

Learning Space Invaders

s

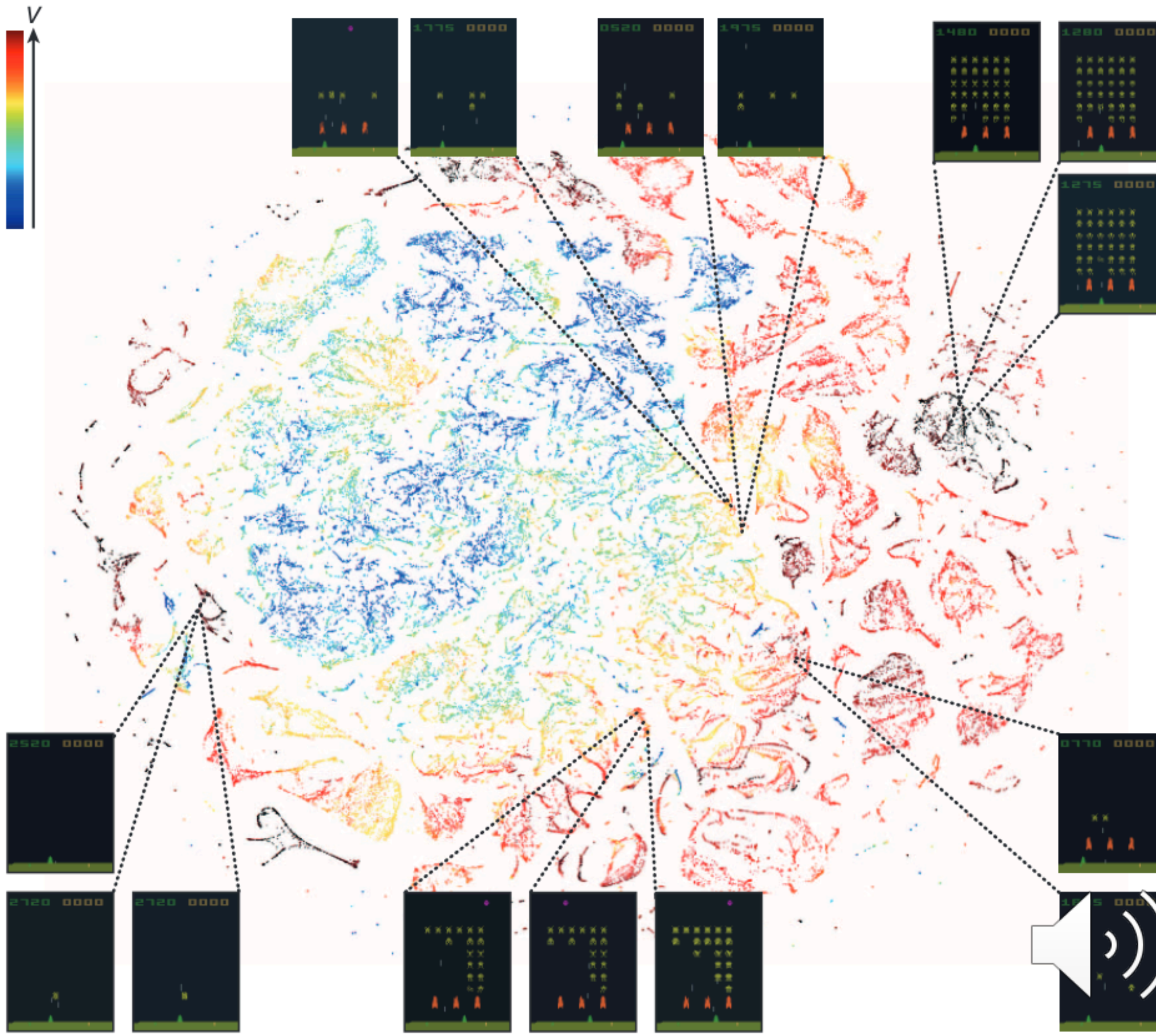
s'





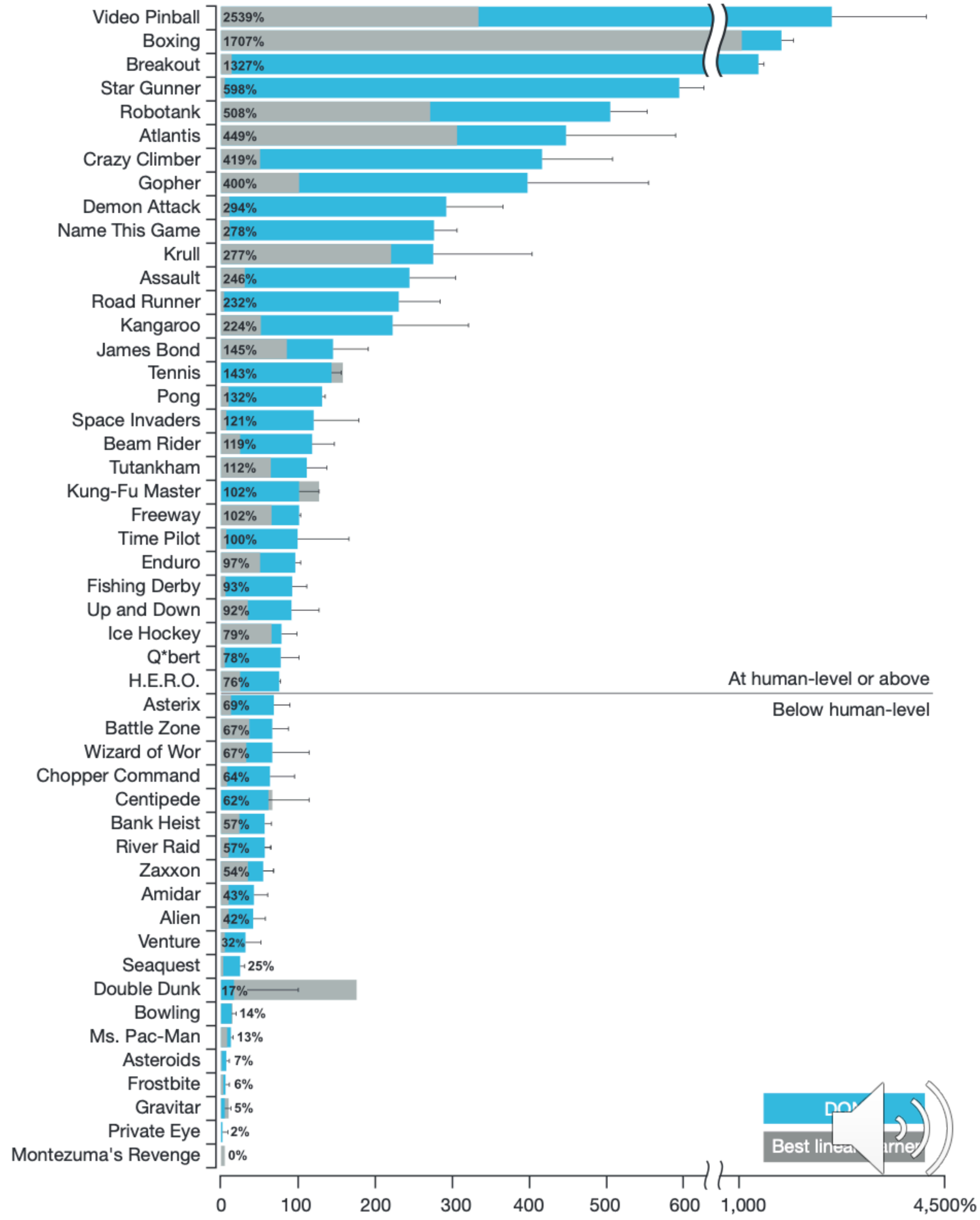
Value function, visualized

- 512-dim state space (final hidden layer)
- "t-SNE" embedding visualizes this in 2D
- Color is value of state



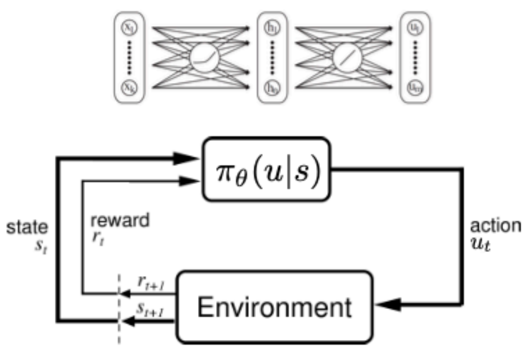
Learning all the games!

- Human = 100%



3. Policy Gradient Method

- Optimize over *stochastic* policies



- Consider control policy parameterized by parameter vector θ

$$\max_{\theta} \quad \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

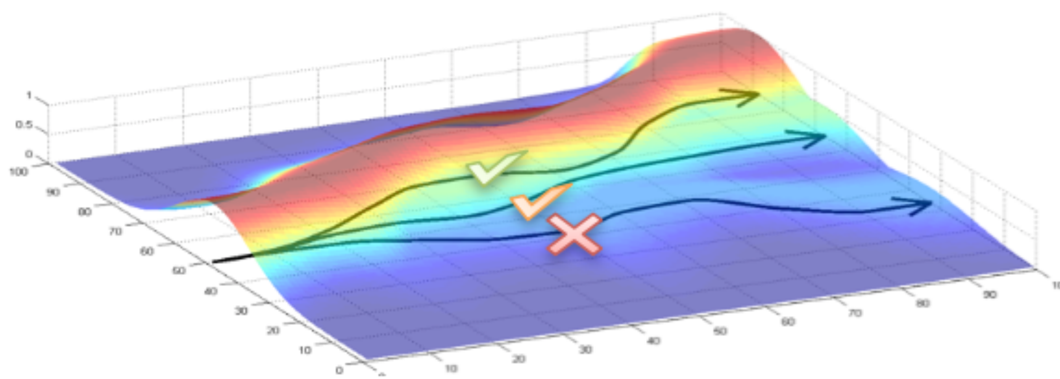
- Stochastic policy class (smooths out the problem):

  $\pi_\theta(u|s)$: probability of action u in state s

Why Policy Optimization

- Often π can be simpler than Q or V
  - E.g., robotic grasp
- V: doesn't prescribe actions
  - Would need dynamics model (+ compute 1 Bellman back-up)
- Q: need to be able to efficiently solve $\arg\max_u Q_\theta(s, u)$
  - Challenge for continuous / high-dimensional action spaces*

Policy leads to *trajectories* τ



We let τ denote a state-action sequence $s_0, u_0, \ldots, s_H, u_H$. We overload notation: $R(\tau) = \sum_{t=0}^{H} R(s_t, u_t)$.

$$U(\theta) = \mathrm{E}[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

In our new notation, our goal is to find θ:

$$\max_{\theta} U(\theta) = \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

Vanilla Policy Gradient

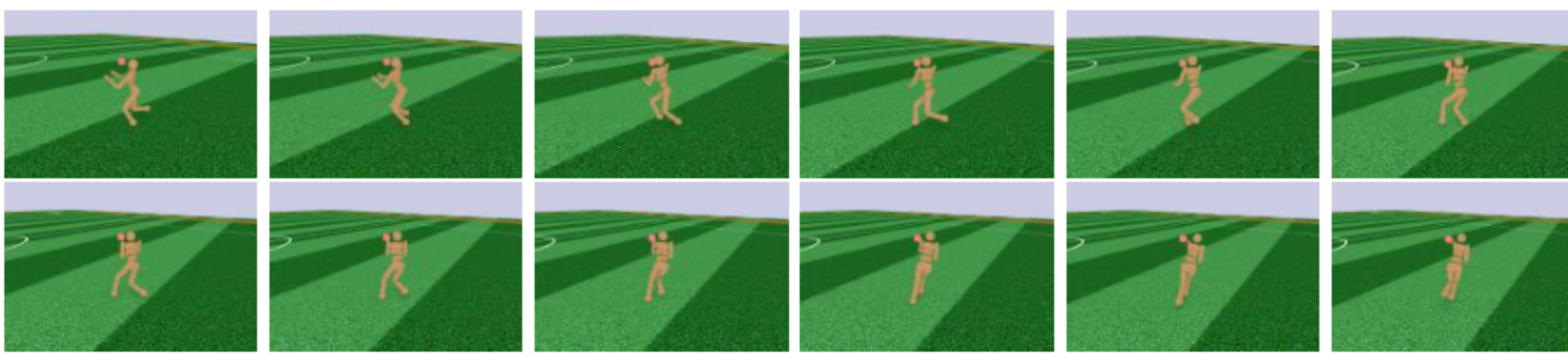
- 1992 (!) REINFORCE algorithm
- Roll out many trajectories
- Compares current policy with a baseline *b*
- Adapt network to improve reward on trajectories that compare advantageously (A_t) to the baseline

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b
for iteration$=1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for

Improvements led to SOTA methods

- Rather complicated and beyond scope
  - DDPG: Deep Deterministic Policy Gradient
  - TRPO: trust-region Policy Optimization
  - PPO: Proximal Policy Optimization
    - See blog post: https://openai.com/blog/openai-baselines-ppo/
- Some successes:
  - OpenAI gym:



  - Atari:

	A2C	ACER	PPO	Tie
(1) avg. episode reward over all of training	1	18	**30**	0
(2) avg. episode reward over last 100 episodes	1	**28**	19	1

Summary

1. Recap: MDP and RL Methods
2. Deep Q-Learning (DQN) beats humans on many Atari games, and is 10K citation Nature paper now
3. Policy Optimization learns a network that takes a state and outputs a stochastic action. Tricky to get to work, lots of heavy math, but great success in robotics-like domains.