CSCI 699: Robotic Perception

Yue Wang

October 15th, 2025

Recap: Policy/Value Iterations

Policy Iteration using V

- Can also do Q value function (policy evaluation is more complex but improvement is easier)

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

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value Iteration

- Policy iteration involves an iterative policy evaluation
- Can we remove it?
- Yes! Think about Bellman optimality equation:

$$V^*(s) = \max_a Q^*(s,a) = \max_a \sum_{s'} p(s', r \mid s, a)\,[\, r + \gamma \cdot V^*(s')\,]$$

- Idea: directly update the value function!

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

Summary

- Bellman optimality equations are nonlinear
- In tabular case with known dynamics, we can do DP-based policy improvement
- In general RL settings, approximately solve the Bellman optimality equations!

$$V^*(s) = \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot V^*(s') \right]$$

$$Q^*(s, a) = \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot \max_{a'} Q^*(s', a') \right]$$

Continuous state/action space (need function approximation for value functions)

Unknown dynamics (must be approximated, usually using sampling)

Max over continuous action space

Today: $p(s' \mid s, a)$ is unknown

Q Learning

Q-Value Iteration

- Will converge to Q^* because of the contraction

$$Q_{k+1}(s,a) = \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma \cdot \max_{a'} Q_k(s',a')\right], \quad \forall s,a$$

- However, need to know $p(s',r \mid s,a)$ for dynamic programming
- Algorithm
  - Initialize Q_0
  - $Q_{k+1} = \mathcal{B}Q_k$ ($\mathcal{B}$ is called the Bellman operator)
  - Once converged, $\pi(s) = \text{argmax}_a Q_K(s,a)$
- What if we don't know $p(s',r \mid s,a)$?
  - Sampling from interacting with the environment!
- What if $|S|$ and $|A|$ are large, or (s,a) is in continuous space?
  - Function approximation!

Q-Value Iteration

- Will converge to Q^* because of the contraction

$$Q_{k+1}(s,a) = \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma \cdot \max_{a'} Q_k(s',a')\right], \quad \forall s,a$$

- The size of $Q(s,a) : |S| \times |A|$
  - How many states in an Atari game?
  - Curse of dimensionality! (from Richard Bellman)

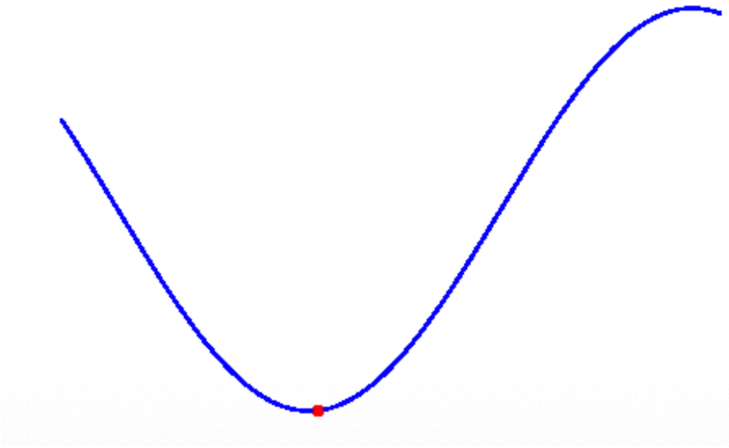
$$|\mathcal{S}| = (255^3)^{200\times200}$$

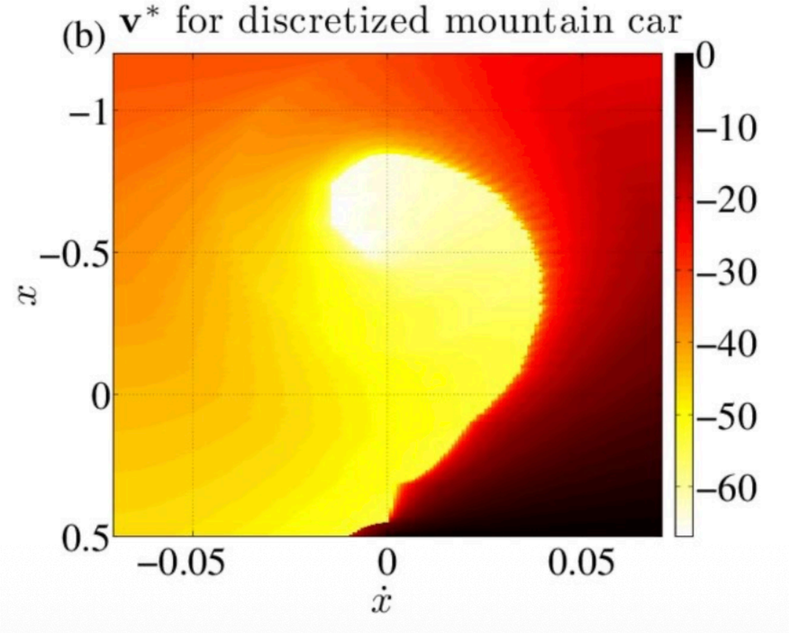
Q-Value Iteration

- Will converge to Q^* because of the contraction

$$Q_{k+1}(s,a) = \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma \cdot \max_{a'} Q_k(s',a')\right], \quad \forall s,a$$

  - Good news: $Q(s,a)$ has some patterns!
    - This is exactly what deep learning is good at.



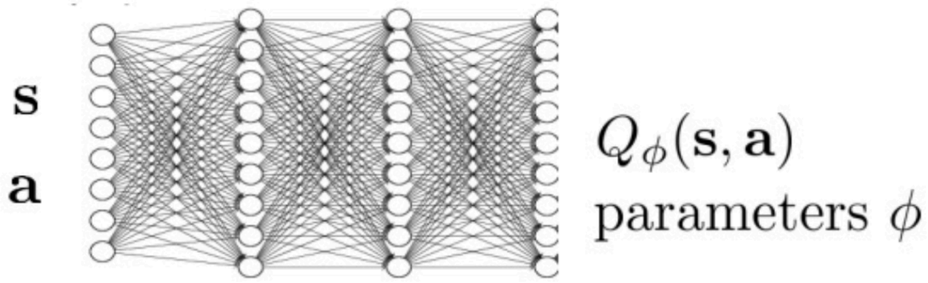


Fitted Q-Iteration (FQI)

- Standard Q-value iteration:

$$Q_{k+1}(s,a) = \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot \max_{a'} Q_k(s', a') \right], \quad \forall s, a$$

  - Fitted Q Iteration (an offline algorithm):
    - Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using "some policy" (not optimal)
    - Repeat
      - $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
      - $\phi \leftarrow \arg\min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$



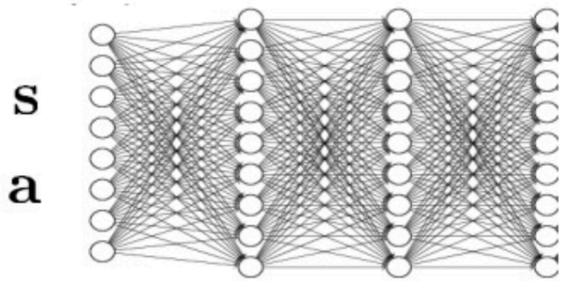
  - Convergence guarantee under some assumptions
    - "Some policy" has good "coverage"
    - Realizability: Q_ϕ can represent Q^*
    - ϕ has some good property

Fitted Q-Iteration (FQI)

- Standard Q-value iteration:

$$Q_{k+1}(s,a) = \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot \max_{a'} Q_k(s', a') \right], \quad \forall s, a$$

  - Fitted Q Iteration (an offline algorithm):
    - Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using "some policy" (not optimal)
    - Repeat
      - $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
      - $\phi \leftarrow \arg\min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$



$Q_\phi(\mathbf{s}, \mathbf{a})$
parameters ϕ

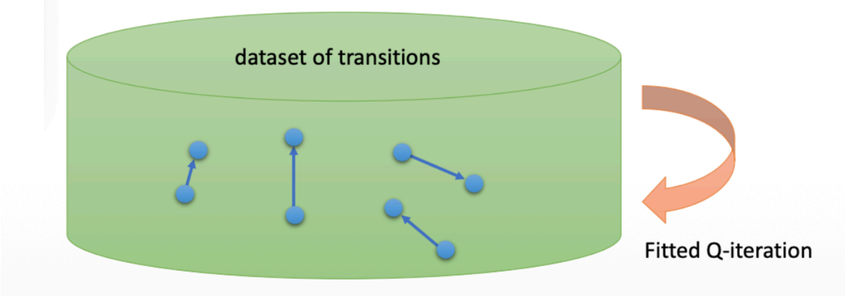
- Main idea 1: using samples $\{(s_i, a_i, s_i', r_i)\}$ to approximate the dynamic programming step
- Main idea 2: using function approximation Q_ϕ

Definition of On/Off-Policy

- Off-policy: the learned policy ≠ the data policy π
- On-policy: the policy used to gather data is the same policy being improved upon
- FQI is off-policy! The learned policy $\text{argmax} Q_\phi$ ≠ the data policy π
- Why is FQI off-policy?
  - Given s_i, a_i, this update is independent of π!

$$y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$$

  - π only influences data distribution!



Fitted Q Iteration (FQI)

- Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using "some policy" (not optimal)
- Repeat
  - $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
  - $\phi \leftarrow \arg\min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$
- Why FQI makes sense? It minimizes the following error:

$$\mathbb{E}_{(s,a,s',r)\sim\pi,p}\left[\left(Q_\phi(s,a) - r(s,a) - \gamma \max_{a'} Q_\phi(s',a')\right)^2\right]$$

- Recall Bellman optimality equation:

$$Q^*(s,a) = \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \cdot \max_{a'} Q^*(s',a')\right]$$

Fitted Q Iteration (FQI)

- Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using "some policy" (not optimal)
- Repeat
  - $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
  - $\phi \leftarrow \arg\min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$
- Problem of FQI:
  - It is an offline algorithm — $\{(s_i, a_i, s_i', r_i)\}$ is pre-collected and fixed.
    - If the data collection policy is too random: inefficient learning
    - If the data collection policy is too deterministic: bad coverage
  - The optimization $\arg\min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$ is hard.

Online Q-Learning

- Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using "some policy" (not optimal)
- Repeat
  - $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
  - $\phi \leftarrow \arg\min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$
- Online Q-Learning: Repeat (online):
  - Take "some" action a and observe (s, a, s', r)
  - $y \leftarrow r + \gamma \max_{a'} Q_\phi(s', a')$
  - Update: $\phi \leftarrow \phi - \eta\,(Q_\phi(s, a) - y)\,\nabla_\phi Q_\phi(s, a)$
- Questions:
  - How to design the data collection policy? How to explore?
  - Can we do a batch update rather than single data point?
  - Does it converge/work?

Online Q-Learning

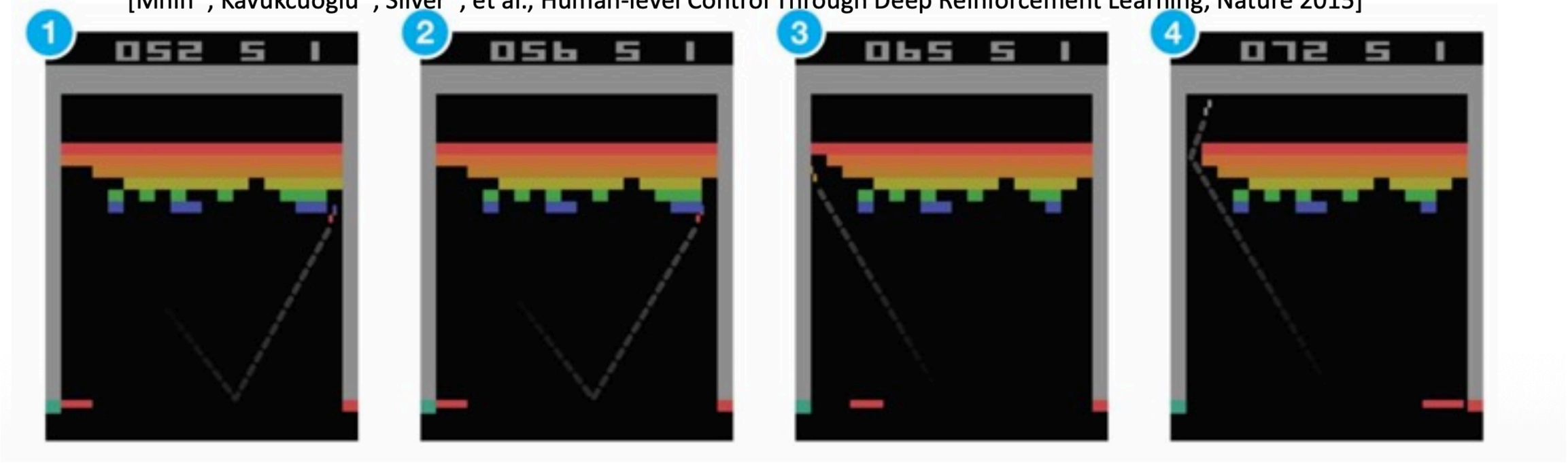
- Online Q-Learning: Repeat (online):
  - Take "some" action a and observe (s, a, s', r)
  - $y \leftarrow r + \gamma \max_{a'} Q_\phi(s', a')$
  - Update: $\phi \leftarrow \phi - \eta \left(Q_\phi(s, a) - y\right) \nabla_\phi Q_\phi(s, a)$
- Q learning is NOT gradient descent (GD)!
  - $y = r(s, a) + \gamma \max_{a'} Q_\phi(s', a')$ is fixed
  - Full GD will consider $\nabla_\phi y$
- It is a semi-gradient method
  - Theoretical analysis is hard…
  - But we can make it work well in practice!

Deep Q-Network (DQN)

Deep Q-Learning

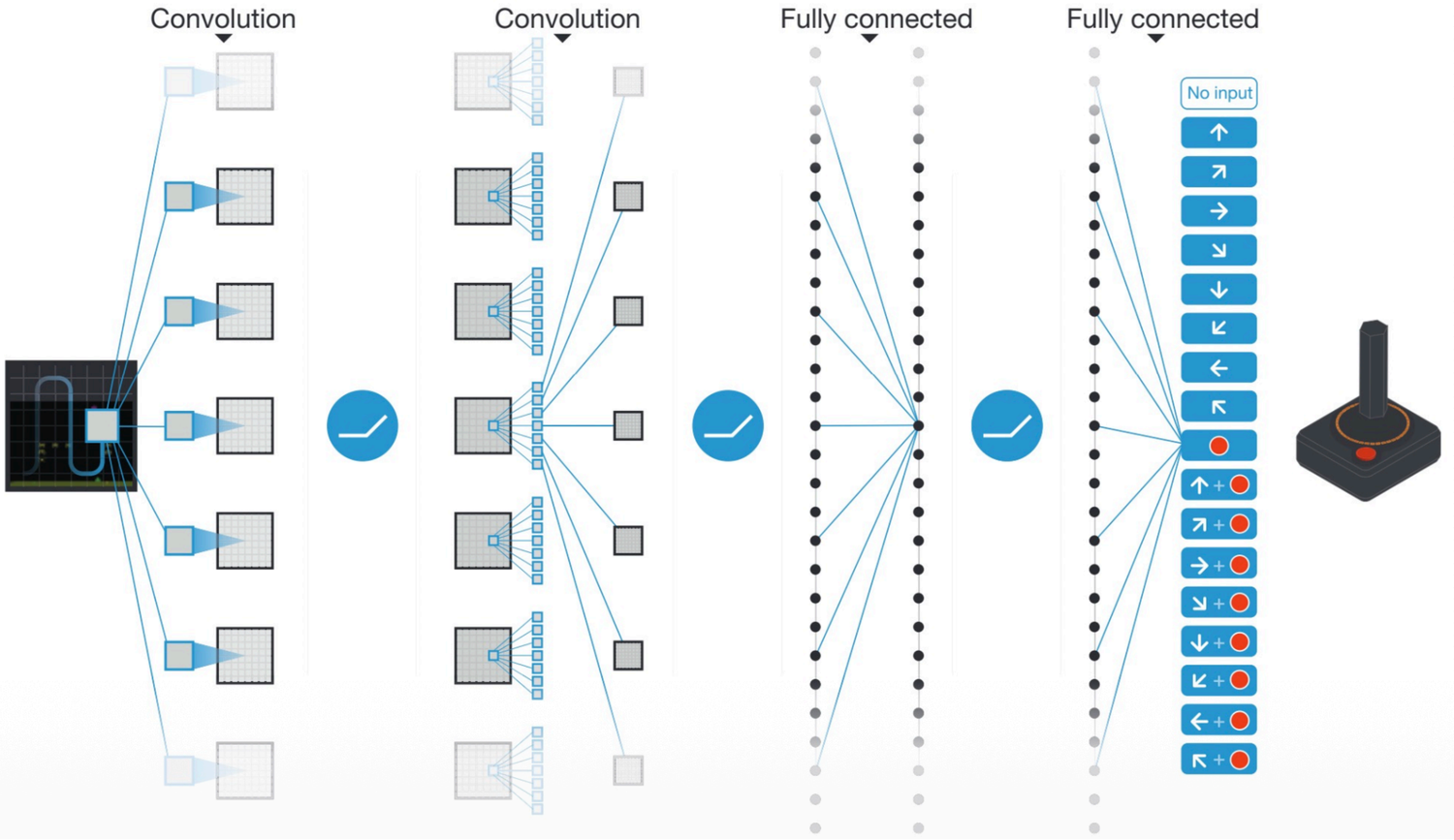
- DQN: Deep Q-Networks
  - DeepMind showed in a 2013 workshop paper (later a Nature 2015 paper) how to train an agent to play Atari from raw pixels and environment reward.
  - Improvements and extensions: Double, Prioritized, Dueling, DDPG, …

[Mnih et al., Playing Atari with Deep Reinforcement Learning, NeurIPS workshop 2013]
[Mnih*, Kavukcuoglu*, Silver*, et al., Human-level Control Through Deep Reinforcement Learning, Nature 2015]



Deep Q-Learning

- Q network:
  - Input: consecutive game frames (4 for Atari)
  - Output: Q values for all actions (discrete)



$$\begin{bmatrix} Q_\theta(\mathbf{s}, \mathbf{a}^{(1)}) \\ Q_\theta(\mathbf{s}, \mathbf{a}^{(2)}) \\ \vdots \\ Q_\theta(\mathbf{s}, \mathbf{a}^{(k)}) \end{bmatrix}$$

DQN: Epsilon Greedy

- Epsilon-greedy policy with a decaying epsilon for exploration

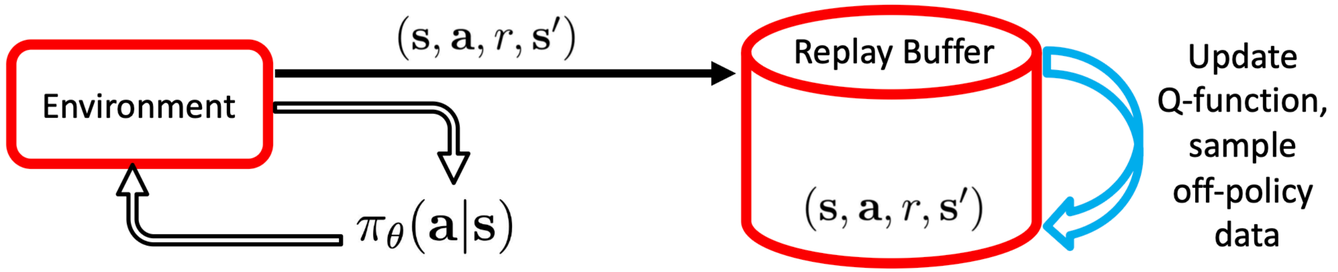
$$\pi_\theta(a \mid s) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q_\theta(s, a') \\ \dfrac{\epsilon}{|\mathcal{A}| - 1}, & \text{otherwise} \end{cases}$$

- Intuition epsilon-greedy:
  - Early in training: Q-values largely a guess, explore to see rewards
  - Later in training: Q-values mostly stable, might be better to exploit
- Vanilla algorithm:
  - Epsilon greedy policy gives a and observe (s, a, s', r)
  - Update $\phi \leftarrow \phi - \eta\left(Q_\phi(s,a) - r - \gamma \max_{a'} Q_\phi(s', a')\right) \nabla_\phi Q_\phi(s,a)$
- BUT …
  - Correlated samples
  - It is NOT GD
  - Target value $r + \gamma \max_{a'} Q_\phi(s', a')$ always changes!

DQN: Replay Buffer

- All algorithmic ideas in DQN (and variants) aim to make Q learning "closer" to standard supervised learning with SGD update

- Idea 1: experience replay
  - Fundamental reason: Q-learning is off-policy



- Q learning with reply buffer:
  - Epsilon greedy policy gives a and observe (s, a, s', r)
  - Update the replay buffer
  - Sample a batch from the buffer and compute the target y_i
  - Update $\phi \leftarrow \phi - \eta \sum_i \left(Q_\phi(s_i, a_i) - y_i\right) \nabla_\phi Q_\phi(s_i, a_i)$

DQN: Target Networks for Stable Regression

- Idea 2: target network
  - Two Q-networks, with the same architecture but a different set of parameters
  - Periodically update the "lagging/target" parameters: $\phi^{-} \leftarrow \phi$

$$\mathbb{E}_{(s,a,s',r)}\left[\left(Q_{\phi}(s,a) - r(s,a) - \gamma \max_{a'} Q_{\phi}(s',a')\right)^2\right]$$

$$\mathbb{E}_{(s,a,s',r)}\left[\left(Q_{\phi}(s,a) - r(s,a) - \gamma \max_{a'} Q_{\phi^{-}}(s',a')\right)^2\right]$$

DQN: Idea 1 + Idea 2

- Repeat:
  - Epsilon greedy policy gives a and observe (s, a, s', r)
  - Update the replay buffer
  - Sample a batch from the buffer and compute the target

    $$y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_{\phi^-}(s_i', a_i')$$

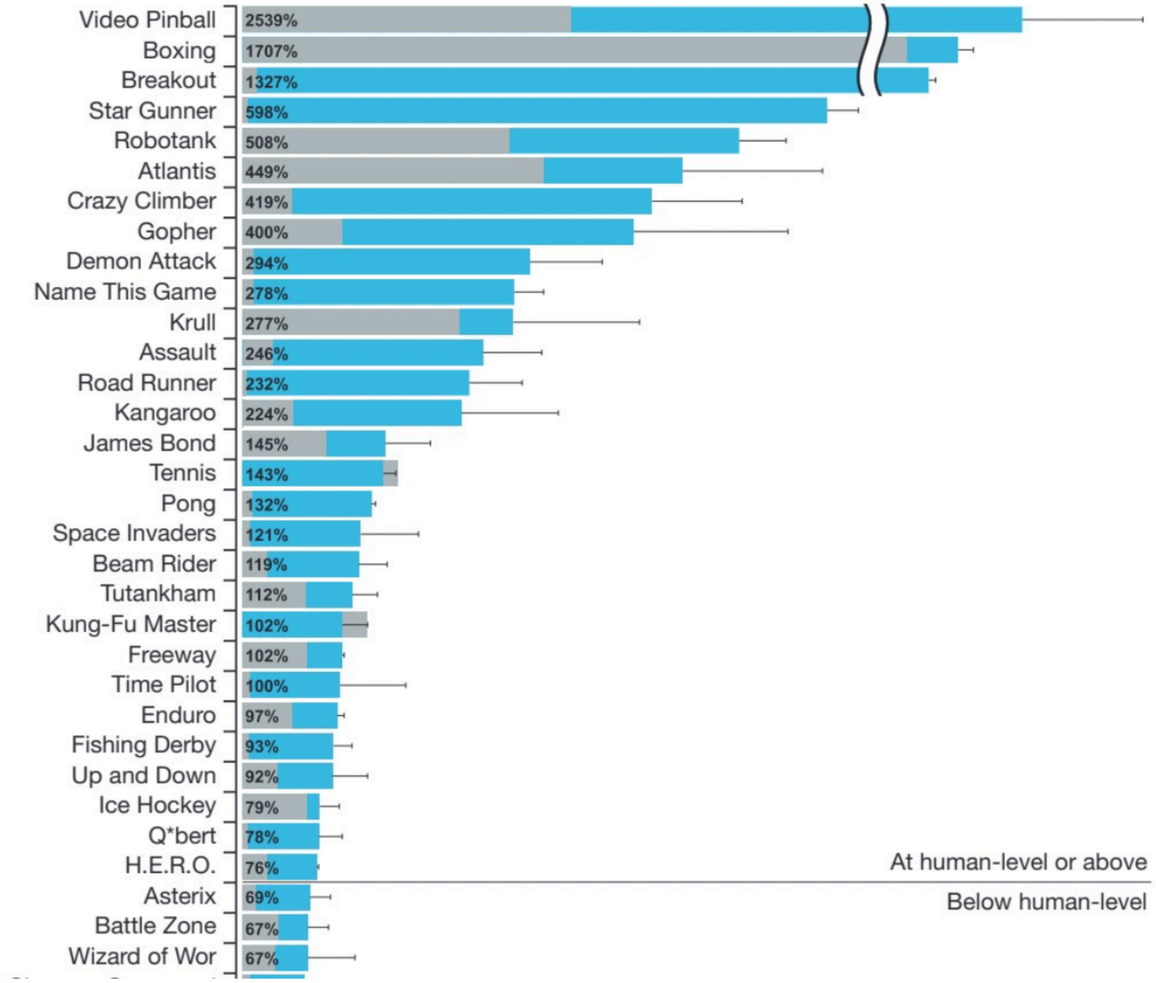
  - Update: $\phi \leftarrow \phi - \eta \sum_i \left(Q_\phi(s_i, a_i) - y_i\right) \nabla_\phi Q_\phi(s_i, a_i)$
  - Periodically update: $\phi^- \leftarrow \phi$

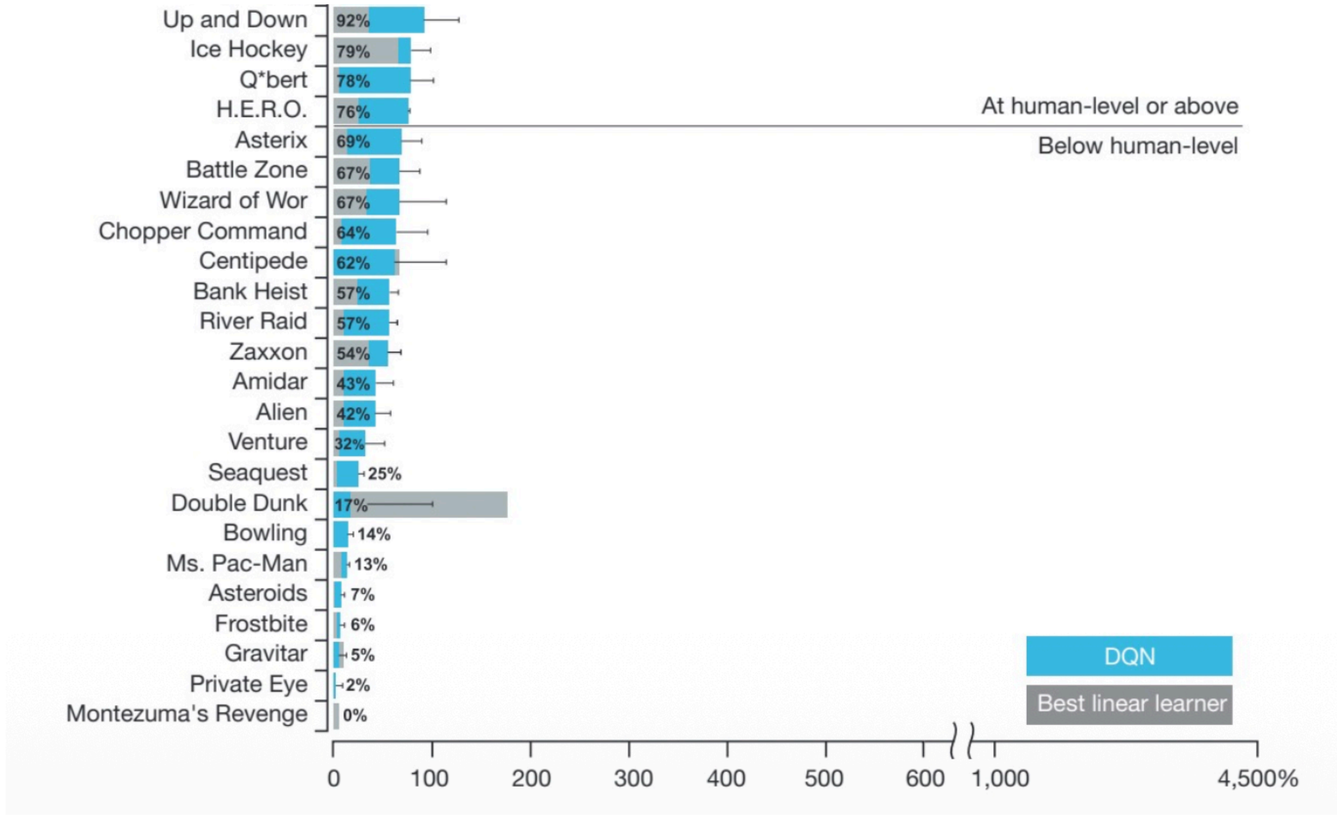
- Alternative target network
  - $\phi^- \leftarrow (1 - \lambda)\phi + \lambda\phi^-$
  - E.g., $\lambda = 0.999$

Practice Tips

- It can take a while to converge (be patient).
- High exploration initially, then reduce
- Test on known / easy tasks first
  - Pong and Breakout are good options.
  - Avoid sparse-reward tasks such as Montezuma’s Revenge
- Use / add more advanced features.
- Use a well tested library (e.g., stable-baselines3).
- The best way to learn about it is to implement it!

DQN results


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
At human-level or above
Below human-level


Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

Acknowledgement

- Many contents are taken from UT Austin CS391, CMU 16-831