Deep Reinforcement Learning

Lecture 3: RL and Function Approximation

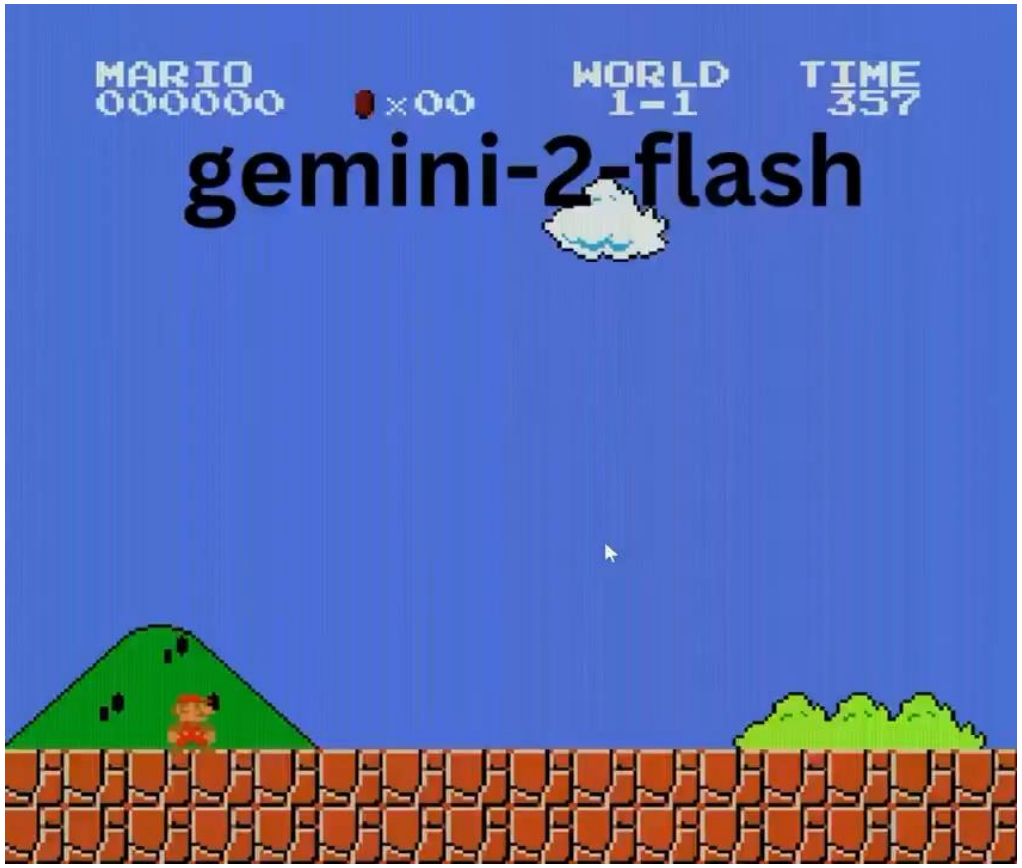
Huazhe Xu

Tsinghua University

HW2 will be released soon!

AI This Week

Physical Intelligence



https://github.com/lmgame-org/GamingAgent

Alert (again)!

- In this lecture, you might hear something like neural networks, convolutional networks, etc.
- If you have zero knowledge about deep learning, it might be hard to follow.
- Try finish Stanford CS231N lectures if you are still on the way to the deep world.

In Lec3

1. From Estimation to Policy
2. Function Approximation
3. Deep Q Learning

So far…

- We have MC, TD, TD(λ) to estimate values.
- If we can find a good policy to follow high value states, then the job is done.

But can we apply policy improvement?

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

Q-learning

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') V(s')$$
$$= R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

- We can perform updates after each action just like in TD.
  - After taking action a in state s and reaching state s' do:

(note that we directly observe reward R(s))

$$Q(s,a) \leftarrow Q(s,a) + \alpha(\underbrace{R(s) + \gamma \max_{a'} Q(s',a')}_{\text{(noisy) sample of Q-value based on next state}} - Q(s,a))$$

Q-Learning (formally)

- Start with initial Q-function (e.g. all zeros)
- Take action from an exploration/exploitation policy giving new state s' (should converge to greedy policy, i.e. GLIE)
- Perform TD update

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

- Goto 2

Note: If an RL algorithm does not depend on the actual next action(s), it is off-policy. The greedy action is neither the current policy nor the policy that collected the sample. If it follows another behavior policy, an importance sampling term might be required.

Q-Learning Convergence

- Theorem: Q-learning converges to the optimal Q-value function in the limit with probability 1, if
  - Every state-action pair is visited infinitely often

Proof Sketch

- Assumption 1: Finite states and actions in an MDP
- Assumption 2:Infinite visits of (s, a) pairs
- Exploration is enabled.
- Robins-Monro Condition: $\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \quad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty$

Theorem The random process $\{\Delta_t\}$ taking values in $\mathbb{R}^n$ and defined as

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x)$$

converges to zero w.p.1 under the following assumptions:

- $0 \leq \alpha_t \leq 1, \sum_t \alpha_t(x) = \infty$ and $\sum_t \alpha_t^2(x) < \infty$;
- $\|\mathbb{E}[F_t(x) \mid \mathcal{F}_t]\|_W \leq \gamma \|\Delta_t\|_W$, with $\gamma < 1$;
- $\mathbf{var}[F_t(x) \mid \mathcal{F}_t] \leq C\left(1 + \|\Delta_t\|_W^2\right)$, for $C > 0$.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation, 6(6):1185–1201, 1994.

Proof Sketch

- Assumption 1: Finite states and actions in an MDP
- Assumption 2:Infinite visits of (s, a) pairs
- Exploration is enabled.
- Robins-Monro Condition: $\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \quad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty$

$$Q_{t+1}(x_t, a_t) = (1 - \alpha_t(x_t, a_t))Q_t(x_t, a_t) + \alpha_t(x_t, a_t)\left[r_t + \gamma \max_{b \in \mathcal{A}} Q_t(x_{t+1}, b)\right]$$

$$\Delta_t(x, a) = Q_t(x, a) - Q^*(x, a)$$

$$\Delta_t(x_t, a_t) = (1 - \alpha_t(x_t, a_t))\Delta_t(x_t, a_t) + \alpha_t(x_t, a_t)\left[r_t + \gamma \max_{b \in \mathcal{A}} Q_t(x_{t+1}, b) - Q^*(x_t, a_t)\right]$$

State-Action-Reward-State-Action (SARSA)

$$Q_{n+1}(s_n, a_n) = Q_n(s_n, a_n) + \alpha_n[r(s_n, a_n) + \lambda Q_n(s_{n+1}, a_{n+1}) - Q_n(s_n, a_n)]$$

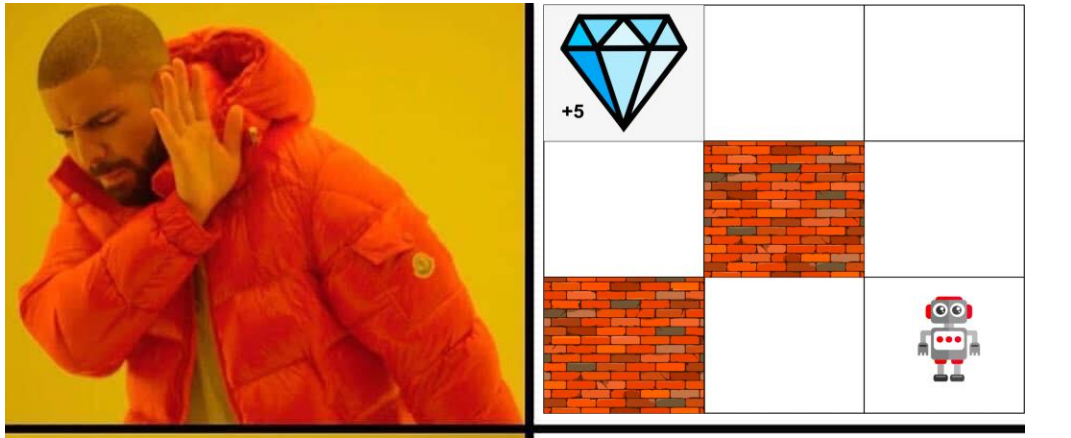
on-policy: The Q-value update depends on the actual next action

What would be next?

- Q Learning is smart, but is it perfect?
- Any other ways to do RL?
- You promised robot dog and dota, but I have grid world.

How can we scale up?

- Value Approximation for RL so that we can deal with real problems



In Lec3

1. From Estimation to Policy
2. Function Approximation
3. Deep Q Learning

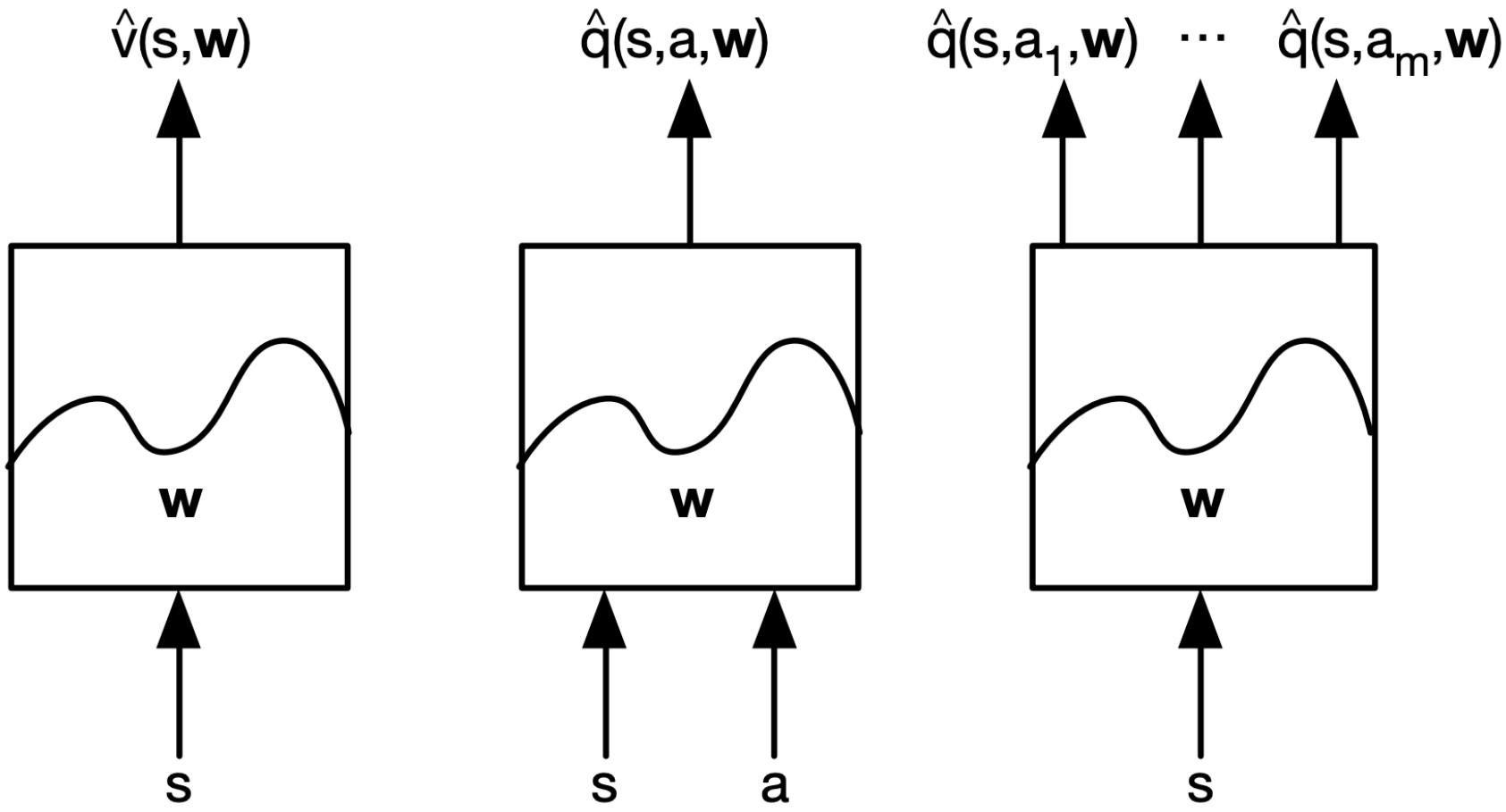
How many states do applications have?

- Chess: $\sim 10^{111}$
- Go: $\sim 10^{170}$
- Robot Dog: Continuous
- Value Iteration or Q learning: it only solves tabular games.
- Any ideas about the challenges?

Challenges & Solutions in largescale applications

- Challenges:
  - Memory
  - Computation
- Solutions:
  - Many states look similar, we might assign similar value to similar states
  - Generalize from seen states to unseen states
  - We can use parameters to perform estimation!
    - Instead of calculating $\mathrm{v}_\pi(s)$ directly, we use function approximators.

How do we parameterize them?



How do you approximate a function (other than neural nets)?

- Linear combinations of features [Huazhe guesses: better explainability]
- Decision Tree [Huazhe guesses: stronger when data amount is small]
- Nearest Neighbor [Huazhe guesses: done by NNs already]
- Fourier / wavelet bases [capture frequency information, especially useful when you care about it]

We now consider the differentiable ones!

Incremental Method: Gradient Descent

- Goal: find parameter vector w minimizing mean-squared error between approximate value $\hat{v}(s)$ and true value $v_\pi(s)$.

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

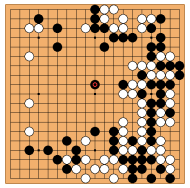
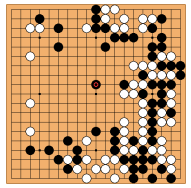
- Gradient Descent

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- You may remove the expectation sign by using samples. Stochastic gradient descent!

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

A concrete example

- $\hat{v}($  $) = v_\pi($  $)$

- For v_π, we assume it can be computed or given by a supervisor.
- For $\hat{v}$, you need some feature vectors.

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- Go pieces configuration (can also be robot joint angles, etc.)

Linear Value Function Approximation

- Represent value functions with a linear combination of features.

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters w

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

- Update with SGD

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

How to get $\mathbf{v}_{\pi}$ in the first place?

- There is no supervisor or oracle in this world.

For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

For TD(λ), the target is the λ-return G_t^{λ}

$$\Delta \mathbf{w} = \alpha(G_t^{\lambda} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

The same concept applies to Q value approximation.

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimize your mean squared error:

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right]$$

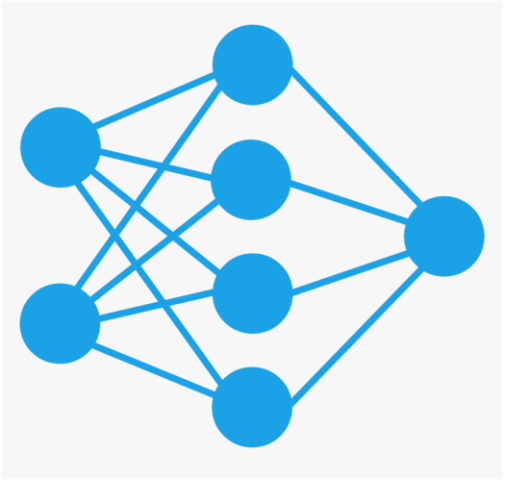
- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

In modern DRL

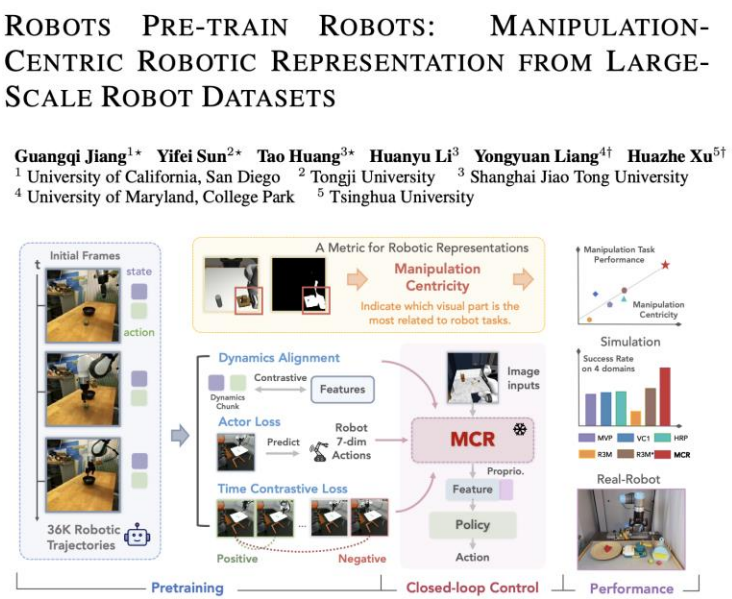
- The parameter w usually refers to a neural network.
- The feature vectors can be high-dimensional sensory information.

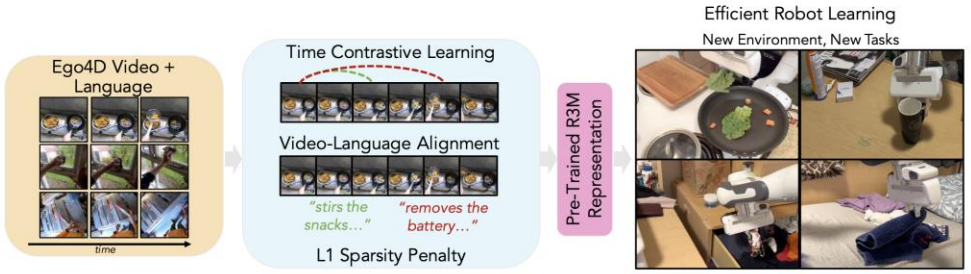


$\hat{v}$

Features are sometimes called representations! And it is popular research topics in DRL.

- Remember the example I gave when you have full knowledge about the states, but the representation really matters? (Mario-Princess example)
- R3M: A Universal Visual Representation for Robot Manipulation Nair et al., CoRL'22





In Lec3

1. From Estimation to Policy
2. Function Approximation
3. Deep Q Learning

Q-learning w/ Non-linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy
3. Perform TD update for each parameter

$$\mathcal{L}_i(w_i) = \left(r + \gamma \max_{a'} Q(s', a'; w_i) - Q(s, a; w_i) \right)^2$$

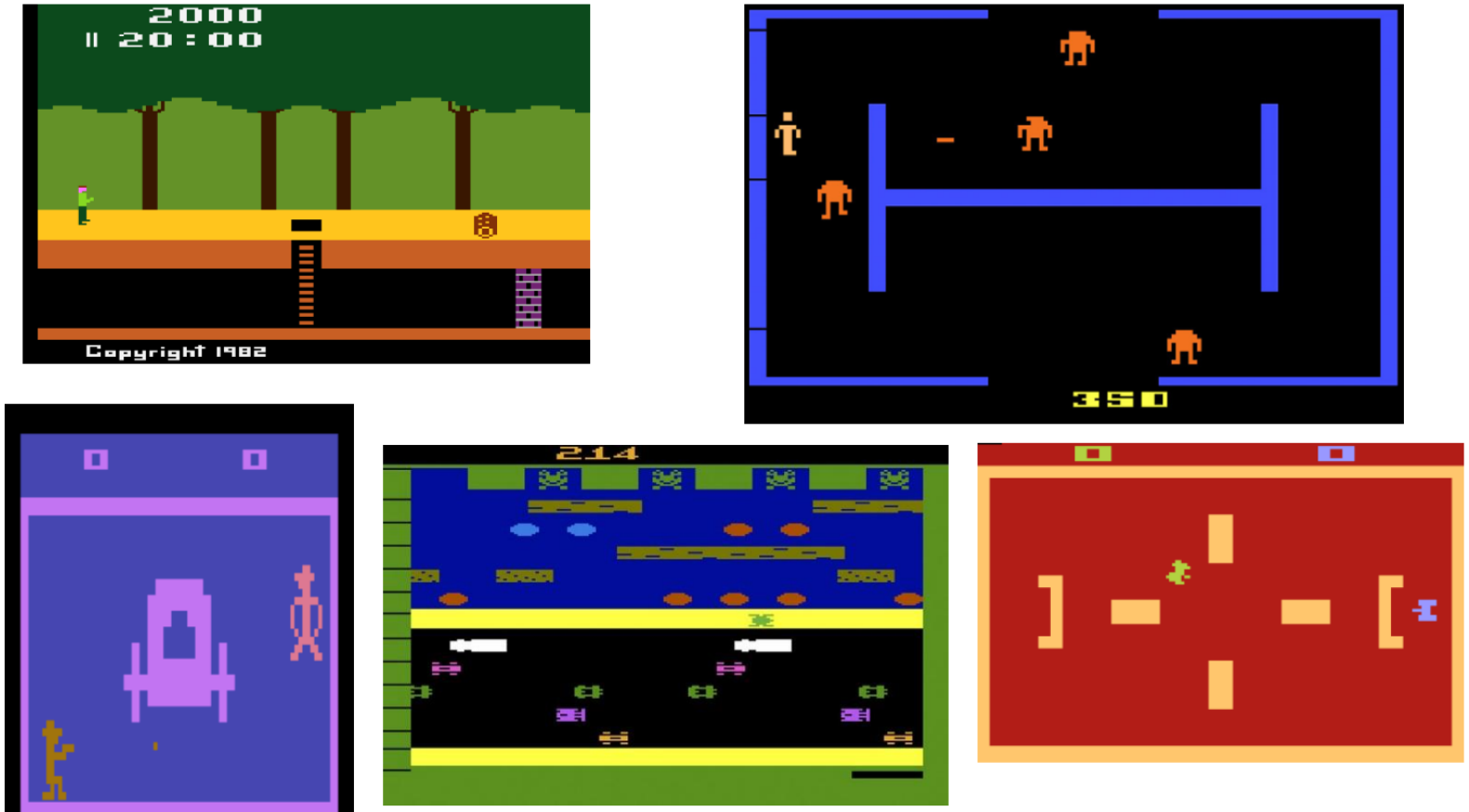
4. Goto 2

No convergence guarantee, but it works! (If you improve this a bit!)

DL + RL = Magic!

- RL defines the objective (Q-value function)
- DL learns the hierarchical feature representations

If you were Vlad Mnih, what would you do to improve?



Human-level control through deep reinforcement learning

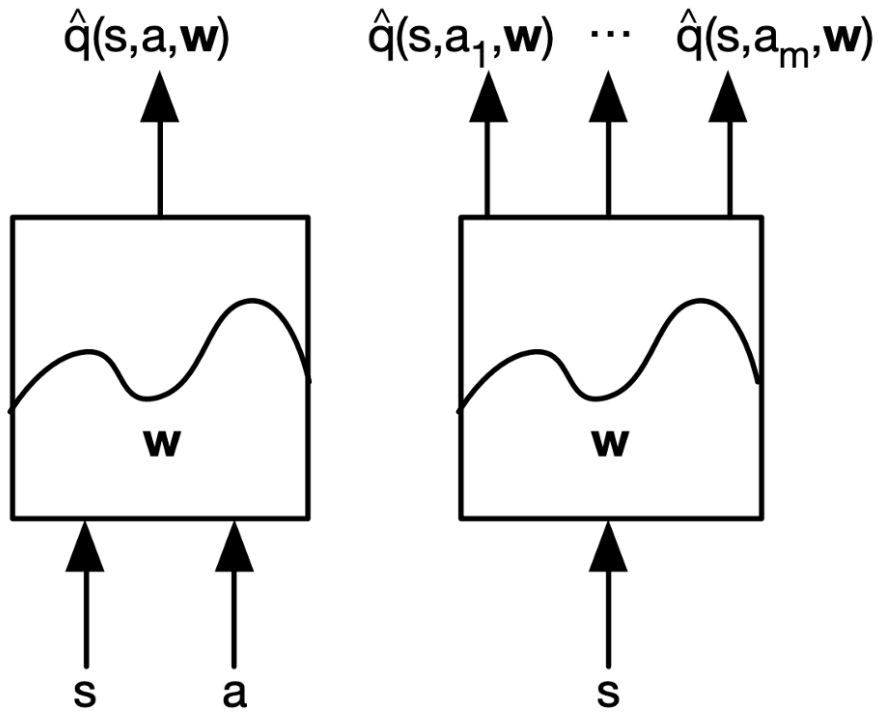
Nature 2015.

Let us reinvent DQN together!

- Put the current image into the network because it is the state. (True or False, Scientific or Engineering)
- Stack 2 images and put them into the network. (True or False, Scientific or Engineering)
- Rescale the image a bit.

Reinvent DQN

- Put s, a into the network or put s into the network with |a| heads? (Scientific or Engineering)



Now I found that my Q value oscillates a lot!

- Correlated data distribution, the NNs might overfit
  - Inefficient data usage
- Q values might change a lot, and small Q value change may induce large policy change
  - Data distributions might swing between extremes
- Unstable gradients if rewards are too large

Sol1 to Correlated data distribution, the NNs might overfit

- Experience replay

- Take action a_t
- Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\boldsymbol{D}$
- Sample random mini-batch of transitions (s, a, r, s') from replay memory $\boldsymbol{D}$
- Optimize MSE between Q-network and Q-learning targets

Sol2 to Q values might change a lot, but small Q value change may induce large policy change

- Target Q Networks

Compute Q-learning targets w.r.t. old, fixed parameters w^-
Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; w_i^-\right) - Q(s, a; w_i)\right)^2\right]$$

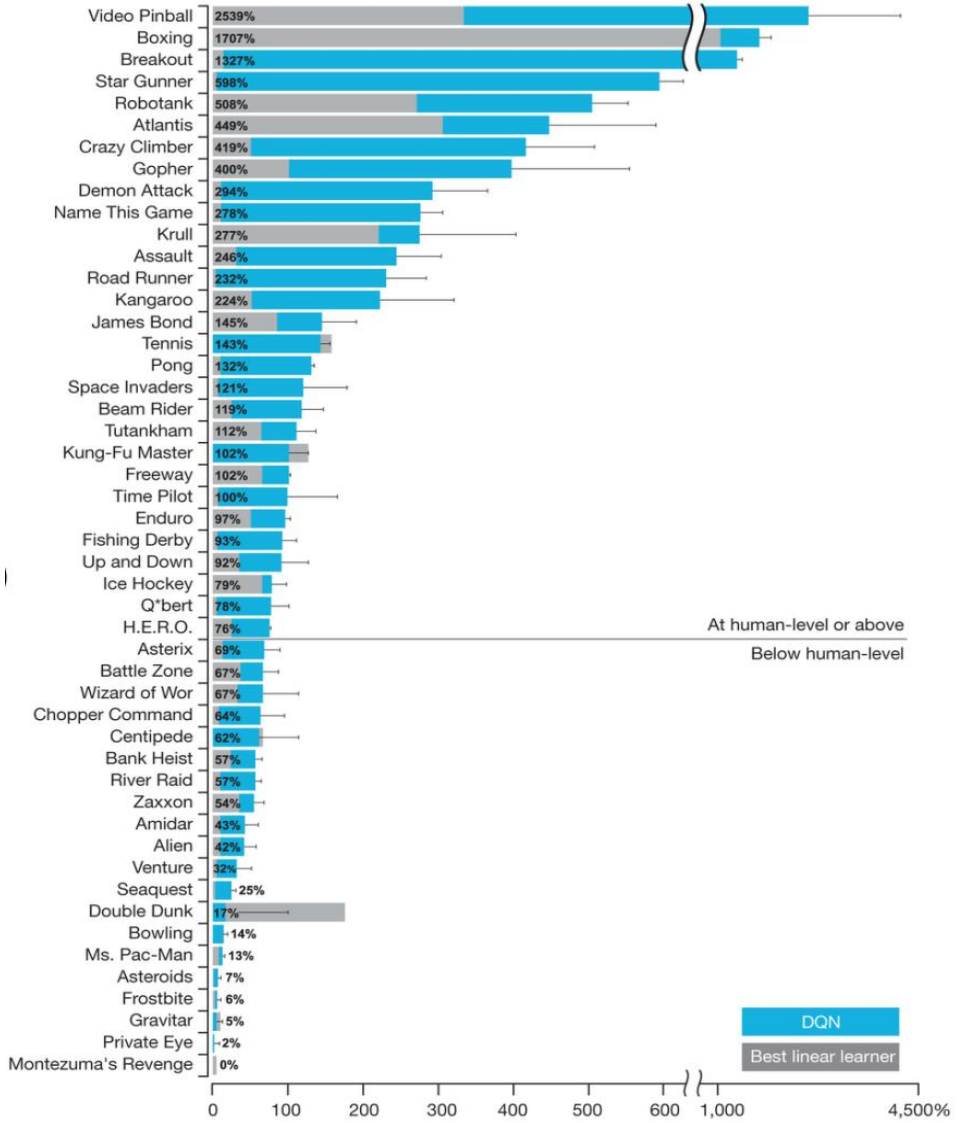
Sol3 to Unstable gradients if rewards are too large

- To limit impact of any one update, control the reward / value range
- DQN clips the rewards to [-1, +1]
  - Prevents too large Q-values
  - Ensures gradients are well-conditioned

Training Details

- 49 Atari 2600 games
- Use RMSProp algorithms with minibatches 32
- Use 50 million frames (38 days)
- Replay memory contains 1 million recent frames
- Agent select actions on every 4th frames
- In DQN, they trusted.

Human-level performance



But remember? Always question the convention!

Overcoming the Spectral Bias of Neural Value Approximation, Yang et al., ICLR'22 [Target network is the convention, I try to remove it with the cost of inventing smarter algorithms.]

"

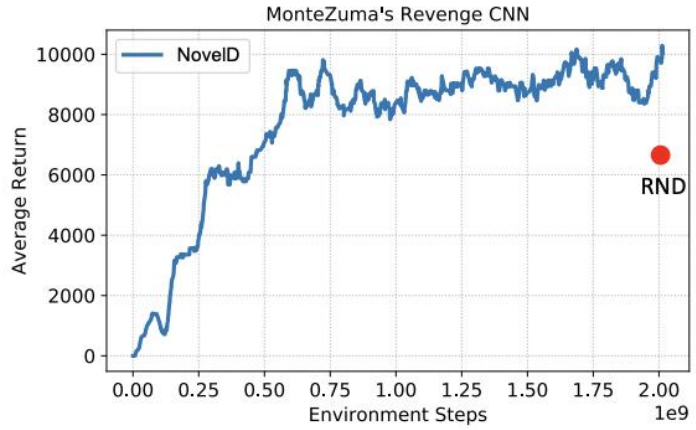
Faster convergence and better off-policy stability also make it possible to remove the target network without suffering catastrophic divergences, which further reduces TD(0)'s estimation bias on a few tasks. Code and analysis available at https://geyang.github.io/ffn.

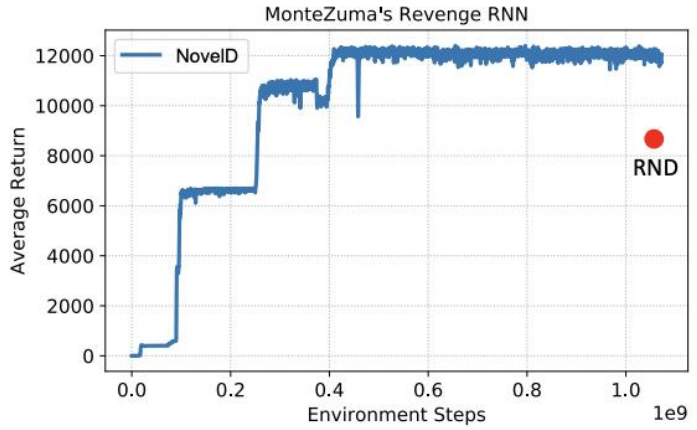
"

The Achilles'Heel of DQN

Montezuma's Revenge is challenging at that time.

- Another research style: always solve the hardest problem.
- Now Montezuma is no longer hard.
  - NovelD: A Simple yet Effective Exploration Criterion, Zhang et al., NeurIPS'21
- If you don't like robots, try Minecraft.





Let's recap here…(If you understand this slide, you are on the right track!)

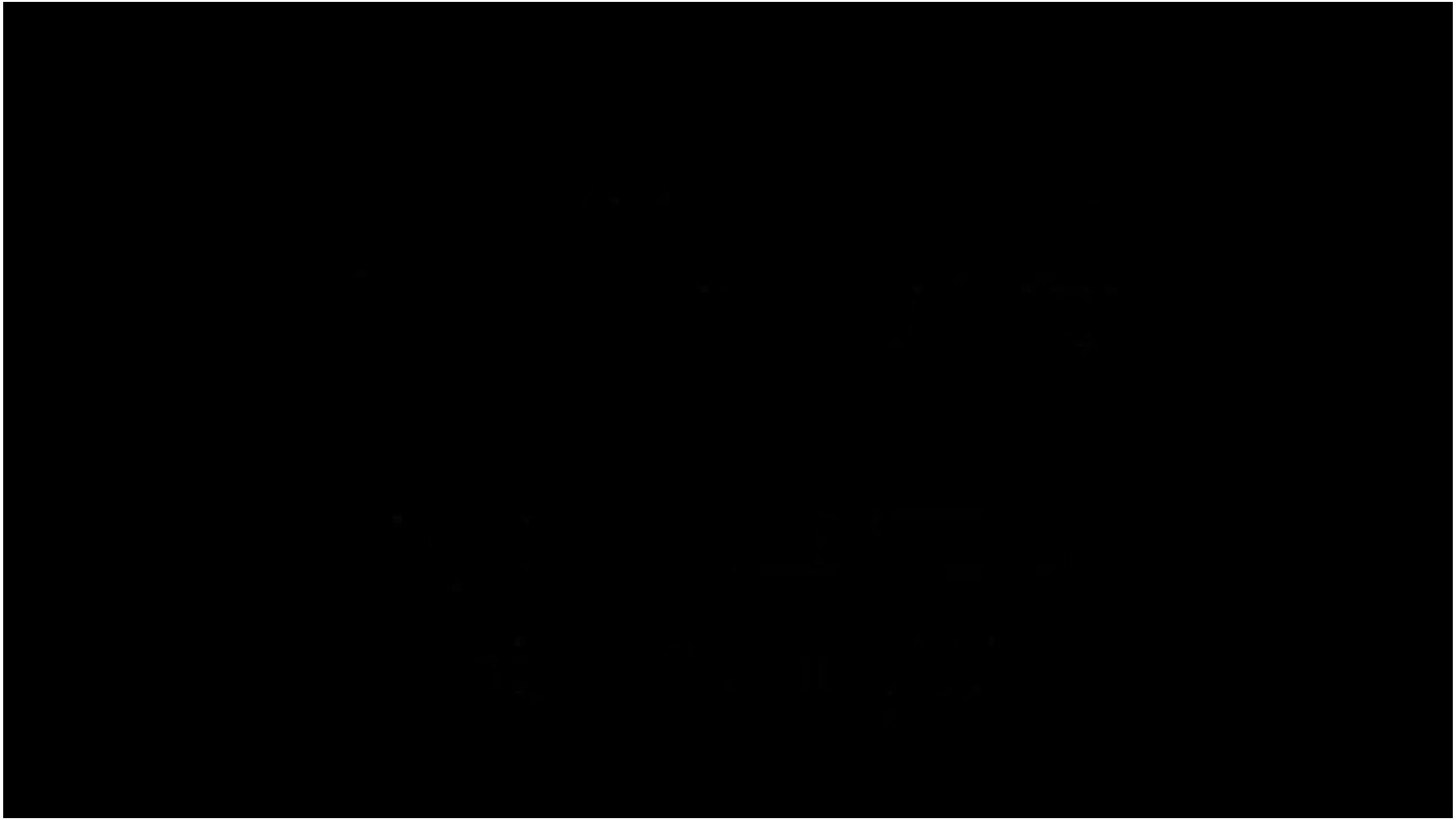
- We've learned value iteration style stuff…
- But sometimes, we don't have transition functions and reward functions.
- We decide to use experience to estimate value: MC, TD
- But it seems it still requires transition function if we need the policy: maybe we should use TD to get Q value
- Even you have Q, you are still passive. You can never know the part you don't know. You need to explore.
- ϵ −greedy for policy to explore
- But you find it is hard to calculate Q values in real games. You decide to use neural networks or other methods to approximate values.
- You add a few tricks such as experience replay, target network, reward clipping. Done!

When something big is out, everyone follows…

Be the first one to dance is hard, but it takes most of the credit! --- the flint!

When you find a lone nut doing something great, have the guts to be the first person to stand up and join in. --- the spark!

Then everyone comes!



Idea 1: Overestimation in Q Learning

$$E(\max(X1, X2, \ldots)) \geq \max(E(X1), E(X2), \ldots)$$

$$\mathcal{L}_i(w_i) = \left(r + \gamma \max_{a'} Q(s', a'; w_i) - Q(s, a; w_i) \right)^2$$

- For each update, you take max first and then average them with multiple samples.

Sol 1: Double DQN

- Dealing with maximization bias of Q-Learning
- Current Q-network w is used to select actions
- Older Q-network w− is used to evaluate actions

$$I = \left(r + \gamma Q\left(s', \underset{a'}{\mathbf{argmax}}\, Q\left(s', a', \mathbf{w}\right), \mathbf{w}^{-} \right) - Q(s, a, \mathbf{w}) \right)^2$$

van Hasselt, Guez, Silver, 2015

Idea 2: You wasted some computation on the samples that are not informative.

- Samples saved in the replay buffer are used randomly.
- It is not efficient.

Sol 2: Prioritized Experience Replay

- Weight experience according to ``surprise" (or error)
- Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

- Stochastic Prioritization

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Schaul, Quan, Antonoglou, Silver, ICLR 2016

Idea & Sol 3: The architecture of DQN can be improved.

- Split Q-network into two channels
- Action-independent value function V(s; w)
- Action-dependent advantage function A(s, a; w)

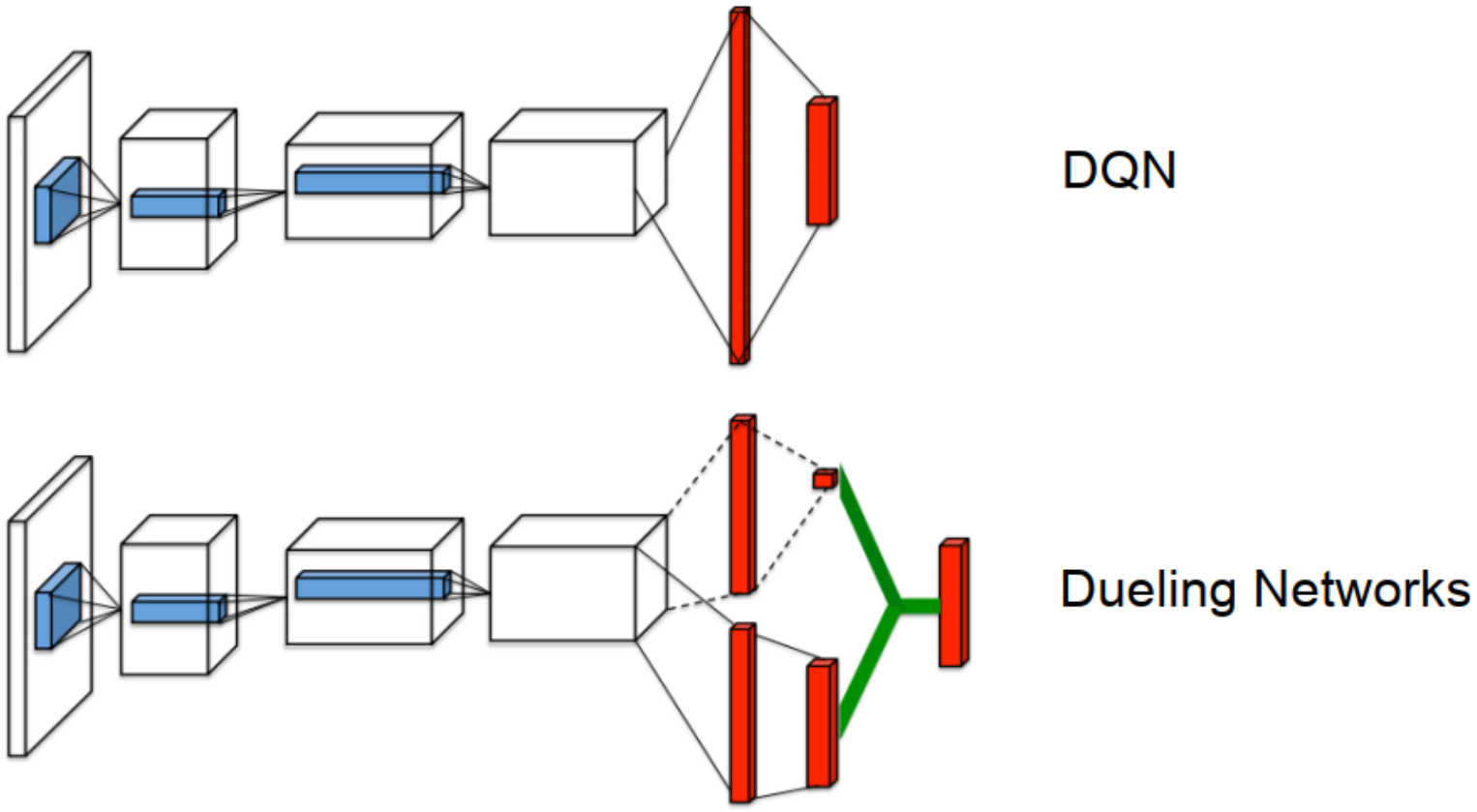
$$Q(s, a; \mathbf{w}) = V(s, \mathbf{w}) + A(s, a; \mathbf{w})$$

- Advantage function is defined as:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

Wang et.al., ICML, 2016

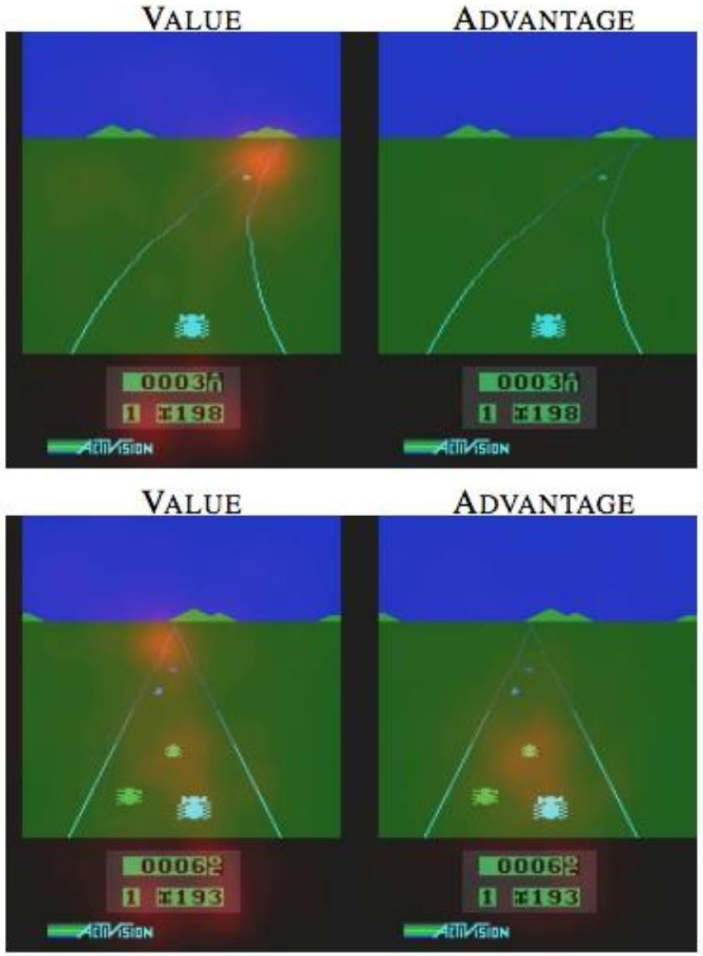
Dueling Networks



Dueling Network

- The value stream learns to pay attention to the road

- The advantage stream: pay attention only when there are cars immediately in front, so as to avoid collisions.



Idea 4: Use n-step return!

- Sometimes, TD(0) does not yield reward in one step.

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$$

$$I = \left(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} Q\left(S_{t+n}, a', \mathbf{w}\right) - Q(s, a, \mathbf{w}) \right)^2$$

Rainbow, Hessel et al

Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel DeepMind

Joseph Modayil DeepMind

Hado van Hasselt DeepMind

Tom Schaul DeepMind

Georg Ostrovski DeepMind

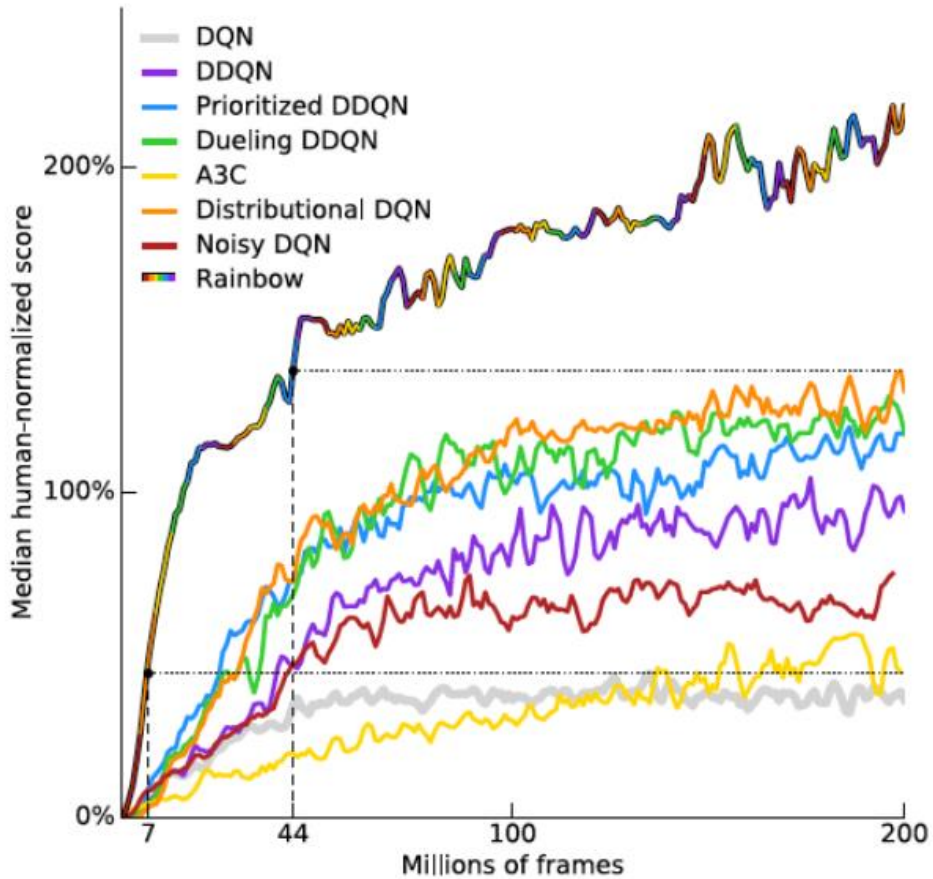
Will Dabney DeepMind

Dan Horgan DeepMind

Bilal Piot DeepMind

Mohammad Azar DeepMind

David Silver DeepMind



In Lec3

1. From Estimation to Policy
2. Function Approximation
3. Deep Q Learning

In our next lecture

- Advanced DQN extensions
- A brand new category of RL algorithms: Policy Gradient