Deep Reinforcement Learning (2)

Dec 4^{th} & 7^{th}, 2017

ADL x MLDS

YUN-NUNG (VIVIAN) CHEN

HTTP://ADL.MIULAB.TW

HTTP://MLDS.MIULAB.TW

國立臺灣大學
National Taiwan University

Slides credited from Dr. David Silver & Hung-Yi Lee

Review

Reinforcement Learning

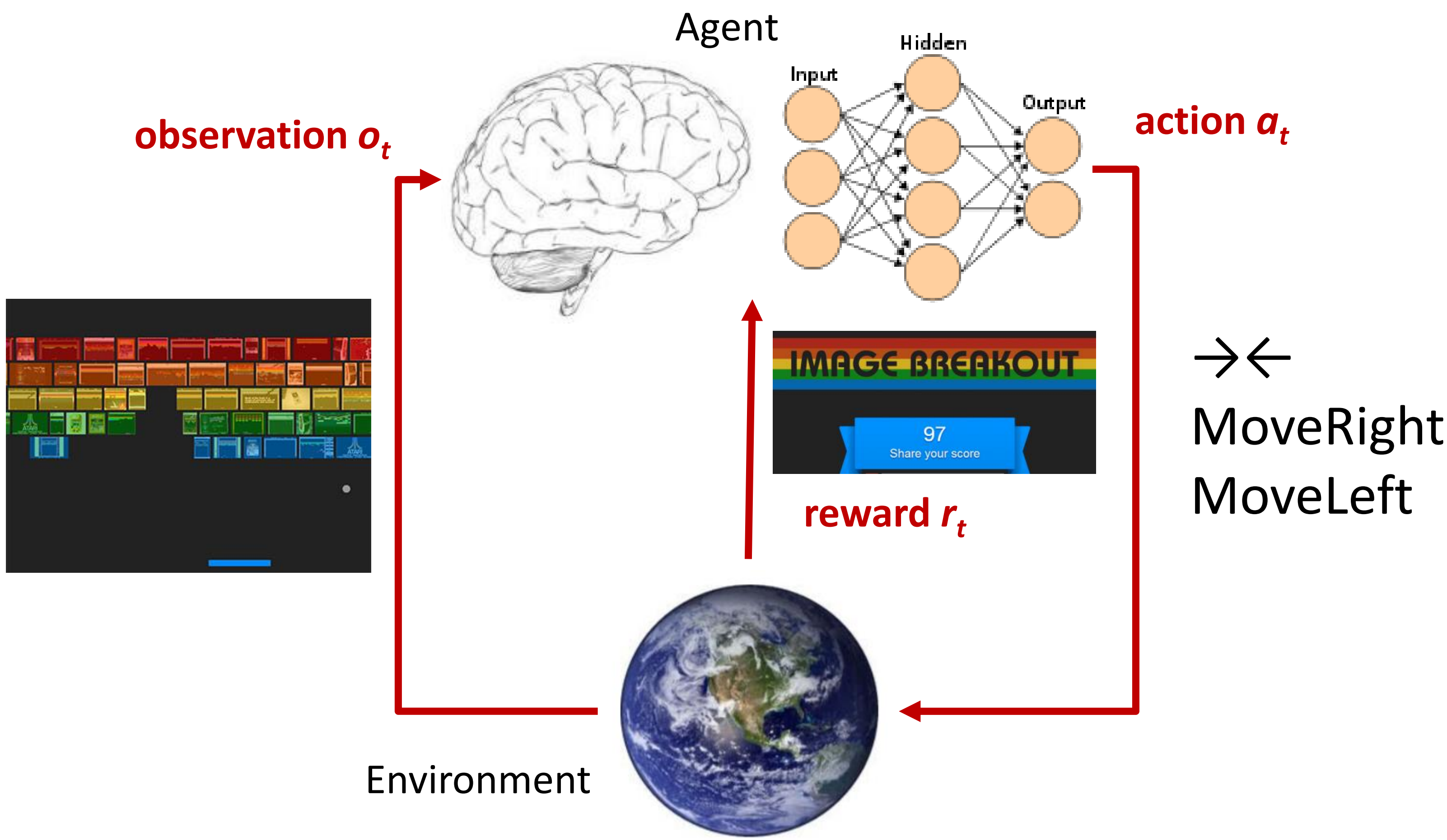
Reinforcement Learning

RL is a general purpose framework for **decision making**

- RL is for an *agent* with the capacity to *act*
- Each *action* influences the agent's future *state*
- Success is measured by a scalar *reward* signal

Big three: action, state, reward

Agent and Environment



Major Components in an RL Agent

An RL agent may include one or more of these components

- **Value function**: how good is each state and/or action
- **Policy**: agent’s behavior function
- **Model**: agent’s representation of the environment

Reinforcement Learning Approach

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$

$Q^*(s, a)$ is maximum value achievable under any policy

Policy-based RL

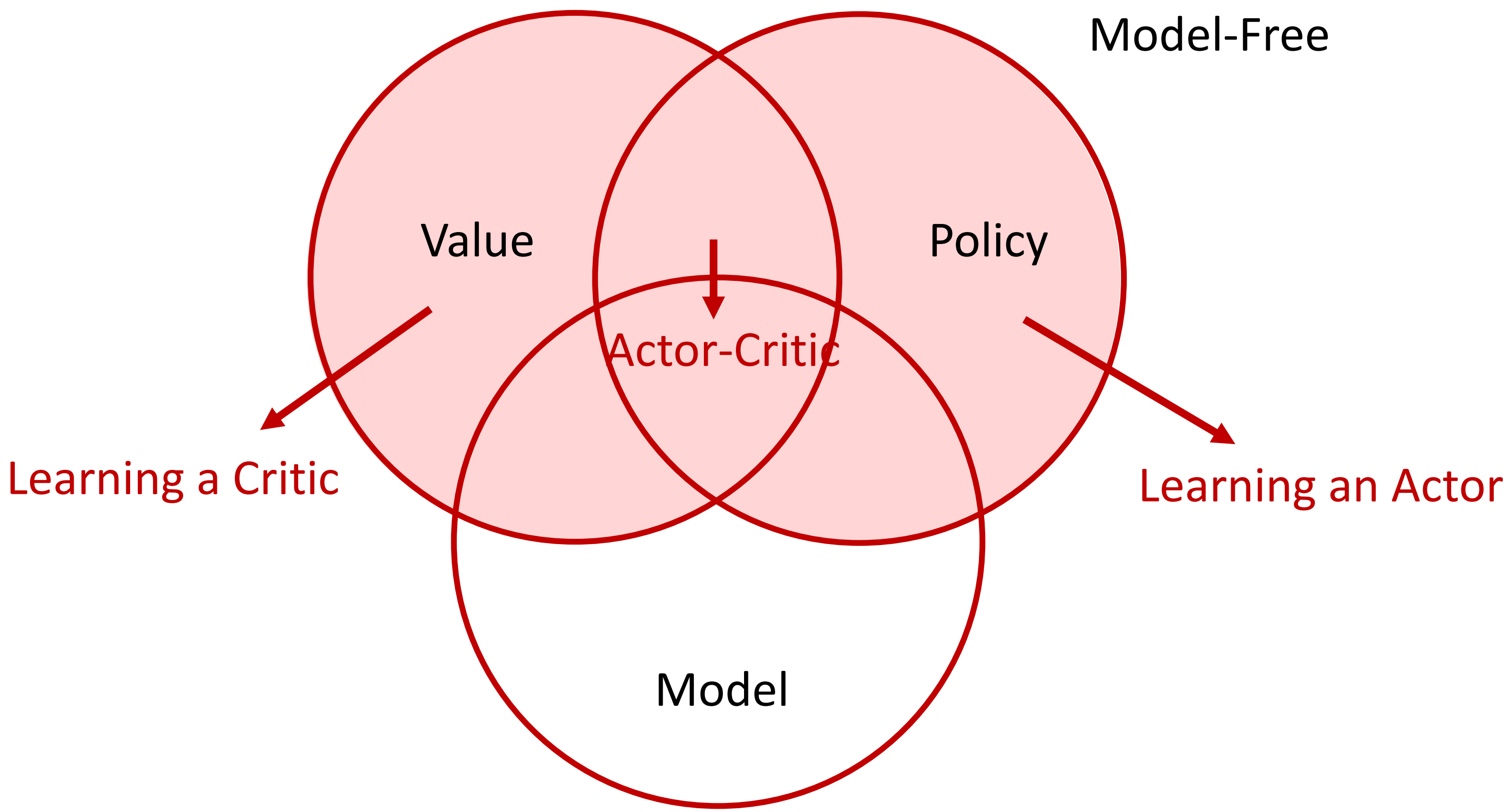
- Search directly for optimal policy π^*

π^* is the policy achieving maximum future reward

Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

RL Agent Taxonomy



Deep Reinforcement Learning

Idea: deep learning for reinforcement learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimize loss function by SGD

Value-Based Approach

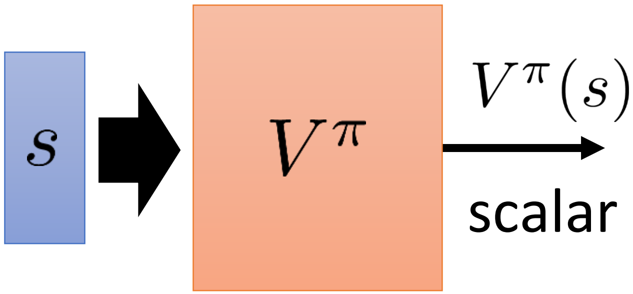
LEARNING A CRITIC

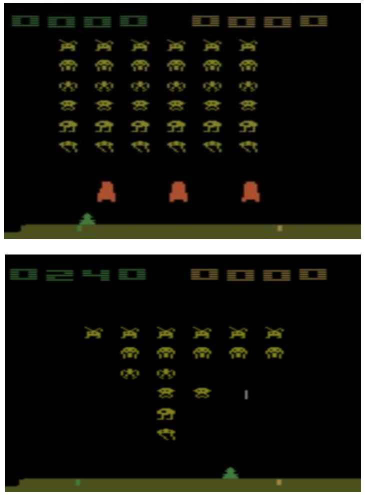
Critic = Value Function

Idea: how good the actor is

State value function: when using actor π, the *expected total reward* after seeing observation (state) s

$$V^{\pi}(s) \ \forall s \quad = \mathbb{E}[G_t \mid s_t = s]$$





larger $V^{\pi}(s)$

smaller $V^{\pi}(s)$

A critic does not determine the action
An actor can be found from a critic

Monte-Carlo for Estimating $V^{\pi}(s)$

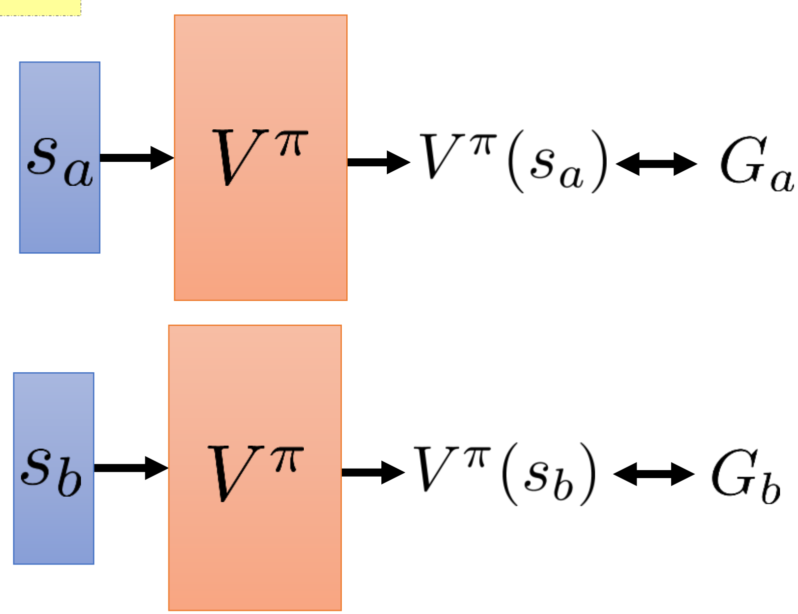
Monte-Carlo (MC)

- The critic watches π playing the game
- MC learns directly from *complete* episodes: no bootstrapping

Idea: value = *empirical mean* return

After seeing s_a,
until the end of the episode,
the cumulated reward is G_a

After seeing s_b,
until the end of the episode,
the cumulated reward is G_b



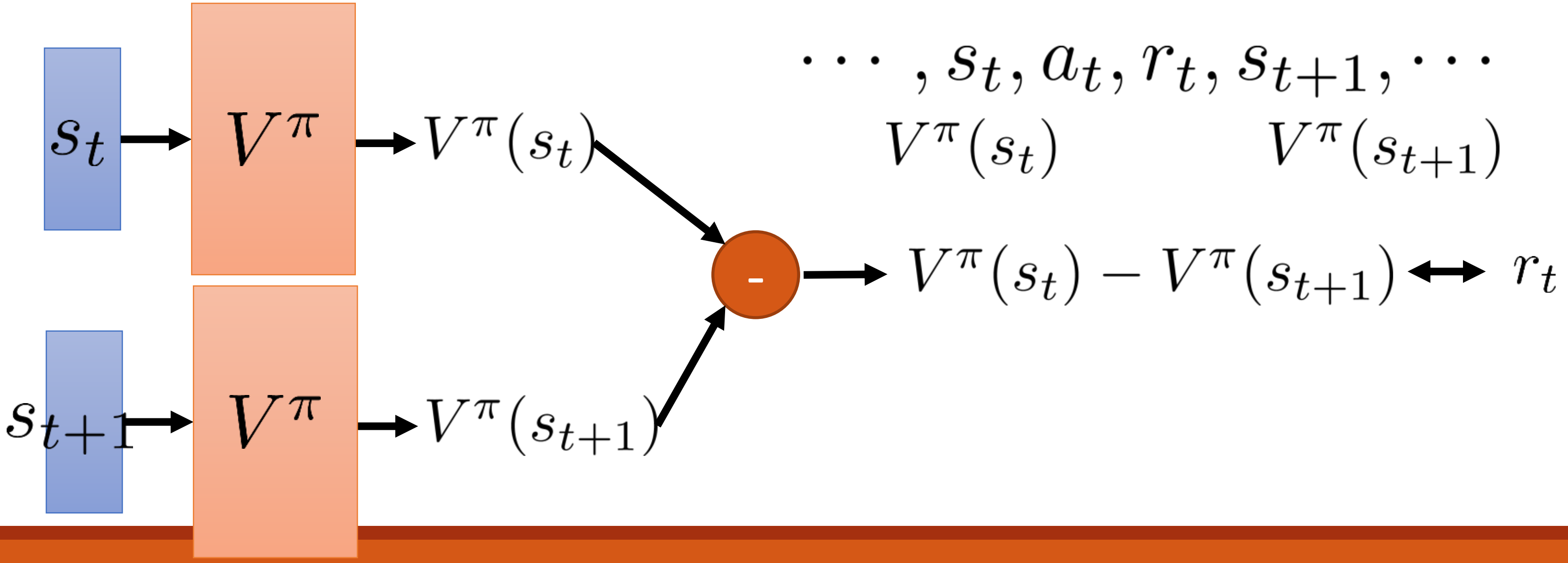
Issue: long episodes delay learning

Temporal-Difference for Estimating $V^\pi(s)$

Temporal-difference (TD)

- The critic watches π playing the game
- TD learns directly from *incomplete* episodes by *bootstrapping*
- TD updates a guess towards a guess

Idea: update value toward *estimated* return



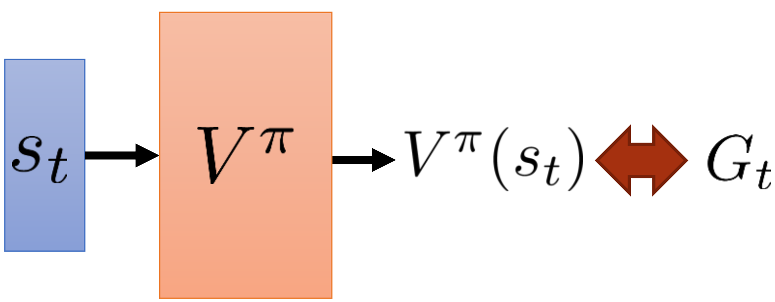
MC v.s. TD

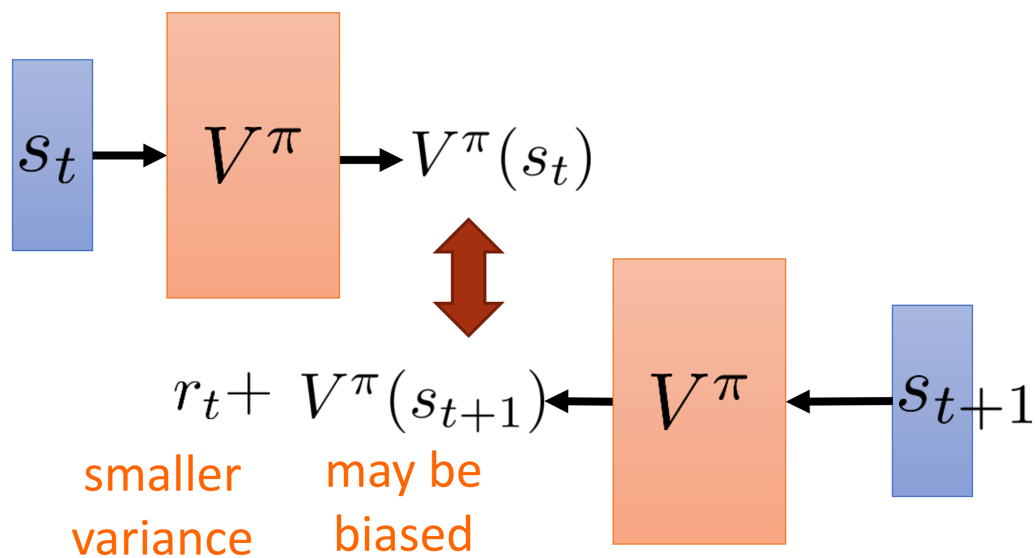
Monte-Carlo (MC)

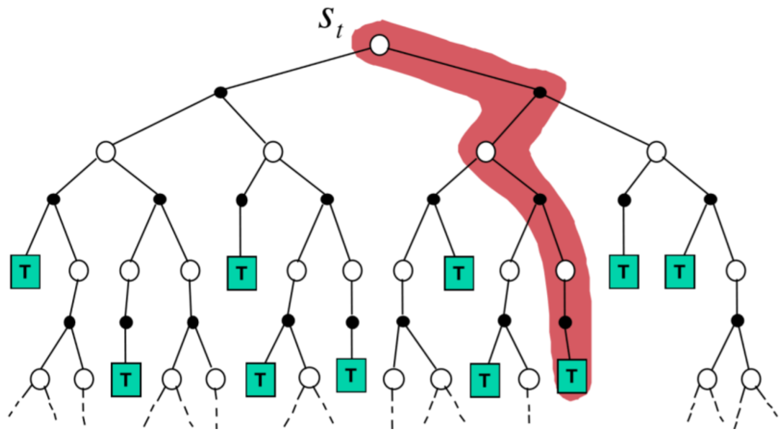
- Large variance
- Unbiased
- No Markov property

Temporal-Difference (TD)

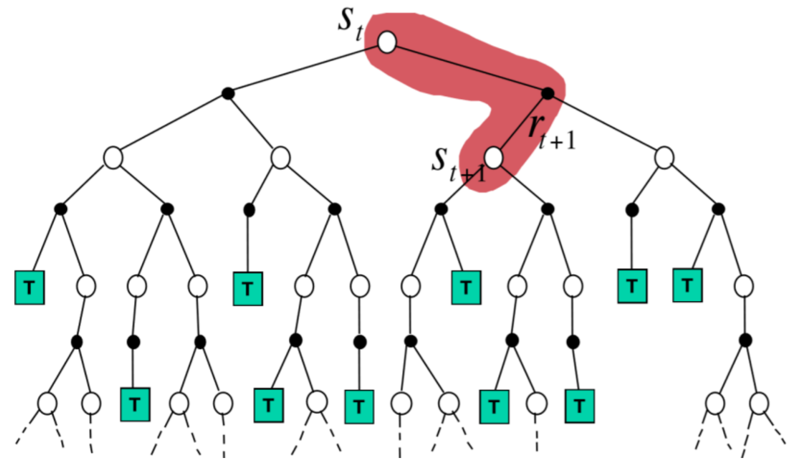
- Small variance
- Biased
- Markov property







$$V'^{\pi}(s_t)$$
$$= V^{\pi}(s_t) + \alpha(G_t - V^{\pi}(s_t))$$



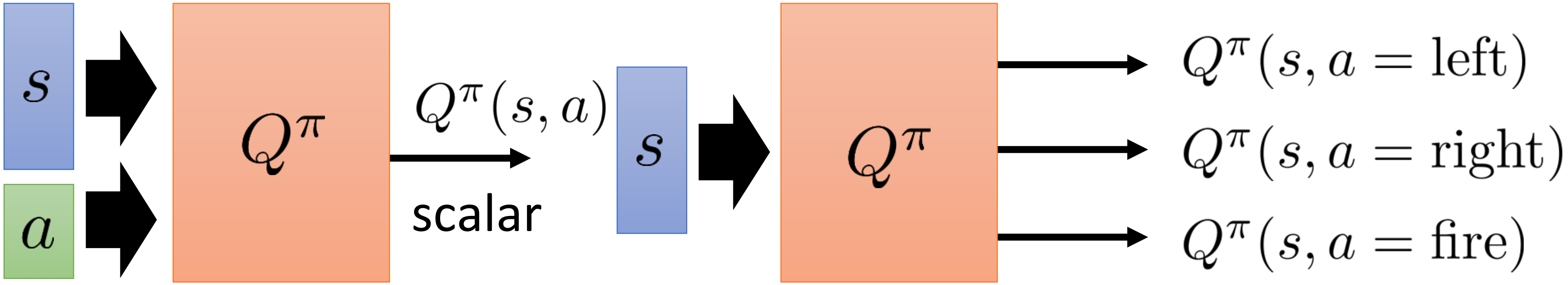
$$V'^{\pi}(s_t)$$
$$= V^{\pi}(s_t) + \alpha(r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t))$$

MC v.s. TD

Critic = Value Function

State-action value function: when using actor π, the *expected total reward* after seeing observation (state) s and taking action a

$$Q^\pi(s,a)\ \forall s,a = \mathbb{E}[G_t \mid s_t = s, a_t = a]$$



for discrete action only

Q-Learning

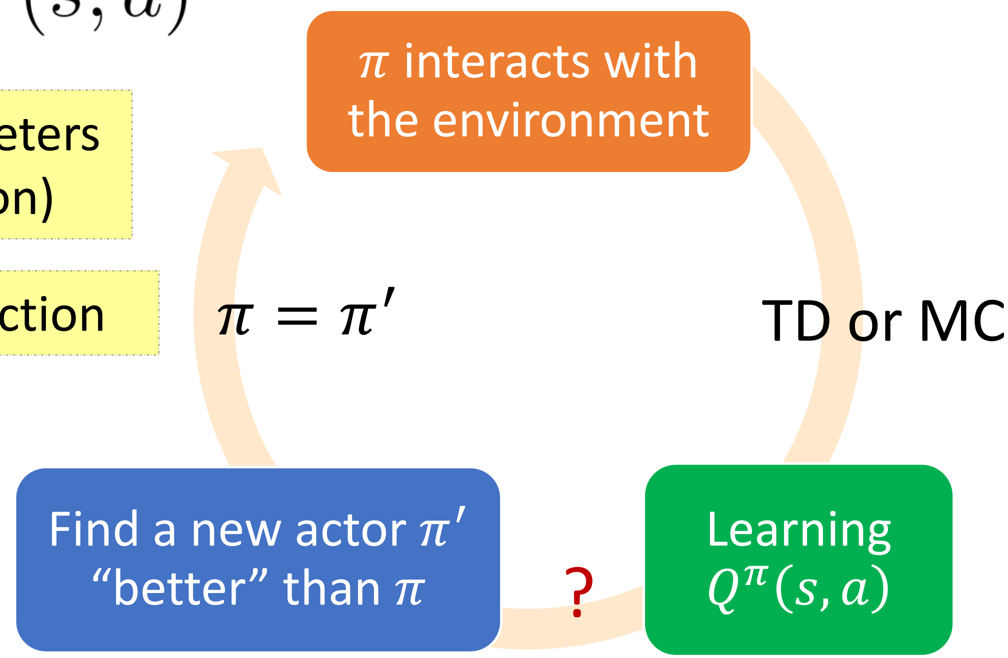
Given $Q^{\pi}(s,a)$, find a new actor π' "better" than π

$$V^{\pi'}(s) \geq V^{\pi}(s) \ \forall s$$

$$\pi'(s) = \arg\max_{a} Q^{\pi}(s,a)$$

π' does not have extra parameters (depending on value function)

not suitable for continuous action



Q-Learning

Goal: estimate optimal Q-values

- Optimal Q-values obey a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$

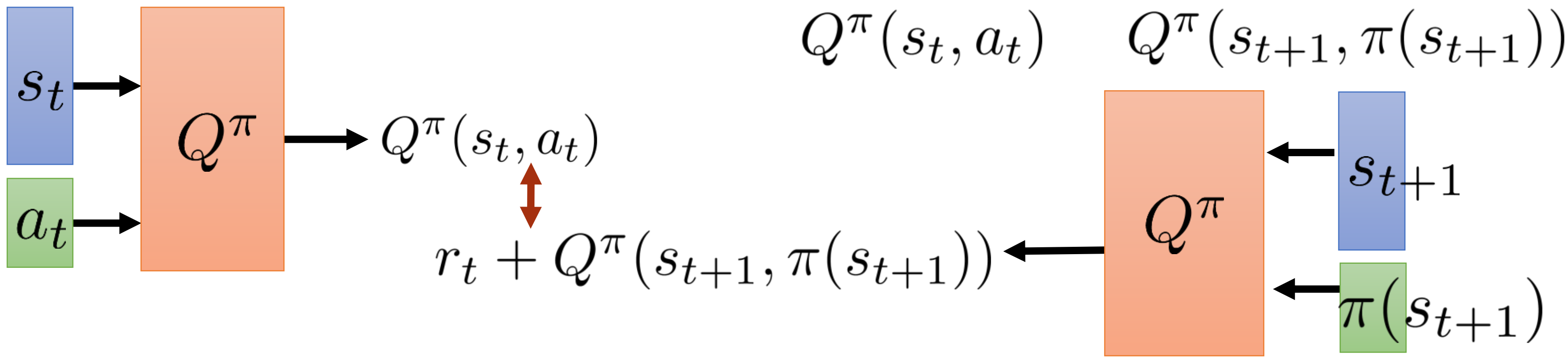
learning target

- *Value iteration* algorithms solve the Bellman equation

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q_i(s',a') \mid s,a\right]$$

Deep Q-Networks (DQN)

Estimate value function by TD $\cdots, s_t, a_t, r_t, s_{t+1}, \cdots$



Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

Objective is to minimize MSE loss by SGD

$$\mathcal{L}(w) = \mathbb{E}\Big[\Big(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\Big)^2\Big]$$

Deep Q-Networks (DQN)

Objective is to minimize MSE loss by SGD

$$\mathcal{L}(w) = \mathbb{E}\Big[\Big(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \Big)^2 \Big]$$

Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\Big[\Big(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \Big) \frac{\partial Q(s, a, w)}{\partial w} \Big]$$

Issue: naïve Q-learning oscillates or diverges using NN due to:
1) correlations between samples 2) non-stationary targets

Stability Issues with Deep RL

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid (independent and identically distributed)
2. Policy changes rapidly with slight changes to Q-values
   - Policy may oscillate
   - Distribution of data can swing from one extreme to another
3. Scale of rewards and Q-values is unknown
   - Naive Q-learning gradients can be unstable when backpropagated

Stable Solutions for DQN

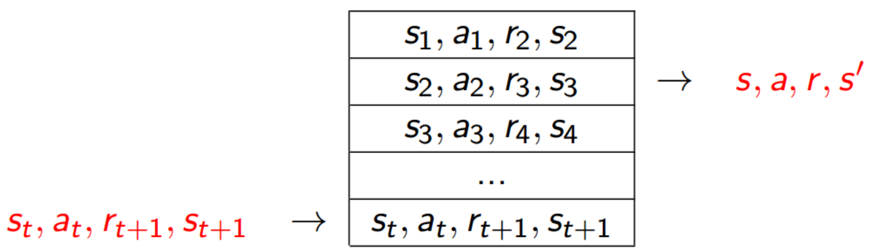
DQN provides a stable solutions to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
2. Freeze target Q-network
   - Avoid oscillation
   - Break correlations between Q-network and target
3. Clip rewards or normalize network adaptively to sensible range
   - Robust gradients

Stable Solution 1: Experience Replay

To remove correlations, build a dataset from agent's experience

- Take action at according to ϵ-greedy policy small prob for exploration
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D

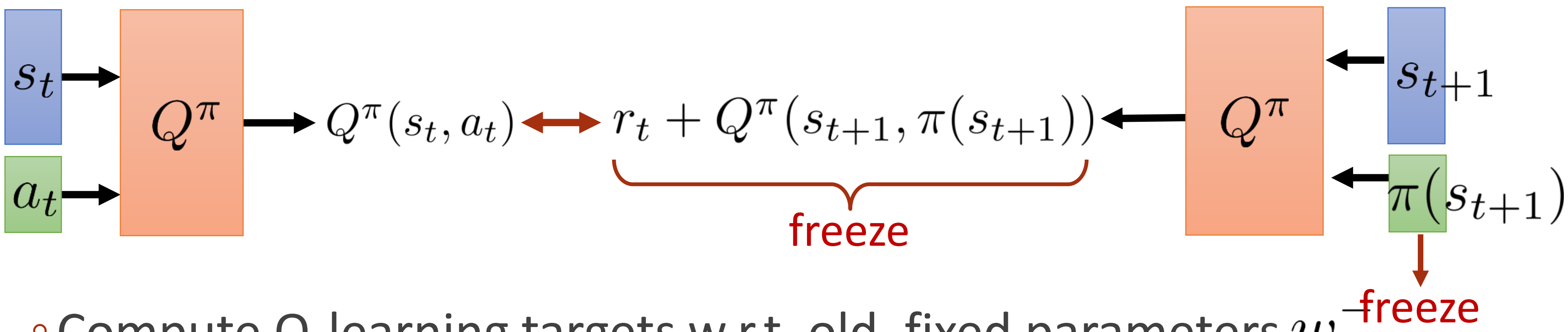


- Optimize MSE between Q-network and Q-learning targets

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

Stable Solution 2: Fixed Target Q-Network

To avoid oscillations, fix parameters used in Q-learning target



- Compute Q-learning targets w.r.t. old, fixed parameters w^-

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

- Optimize MSE between Q-network and Q-learning targets

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

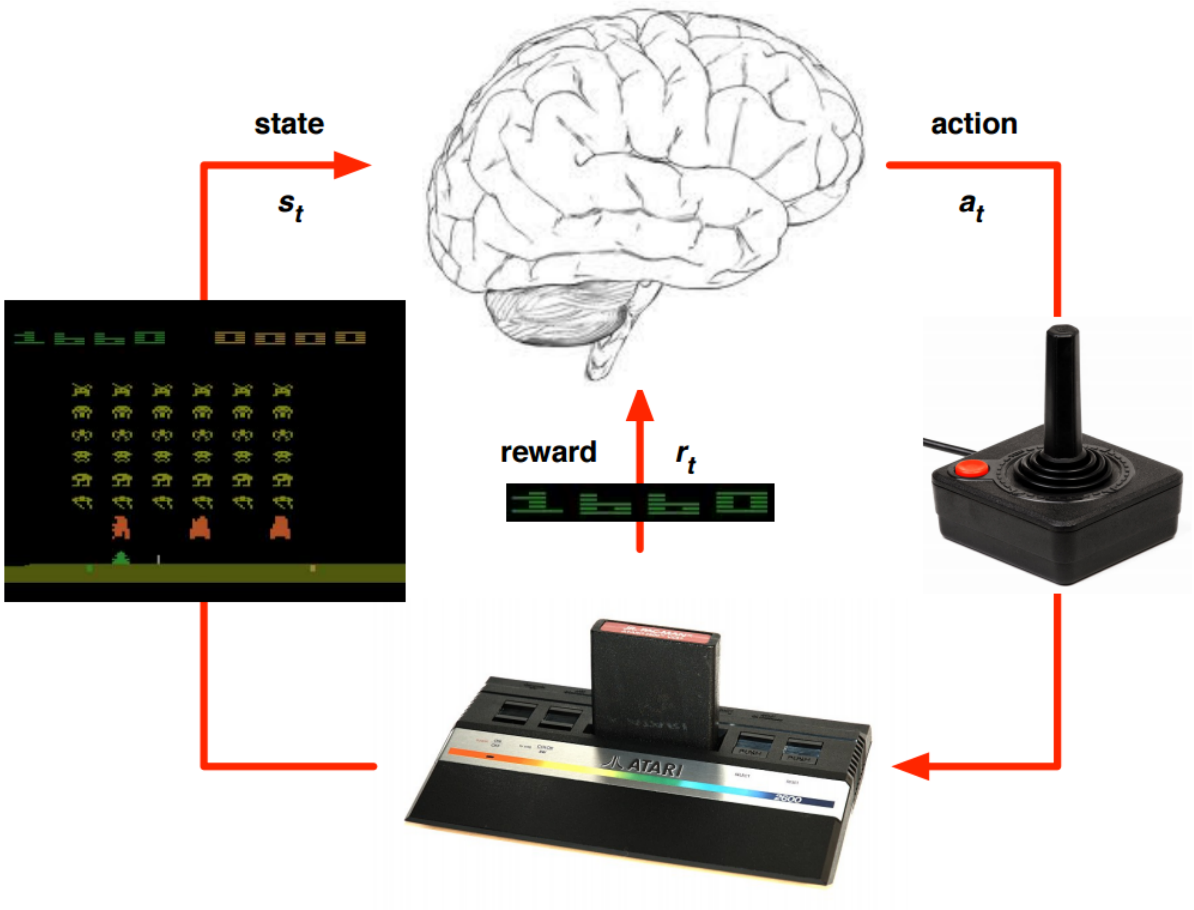
- Periodically update fixed parameters $w^- \leftarrow w$

Stable Solution 3: Reward / Value Range

To avoid oscillations, control the reward / value range

- DQN clips the rewards to $[-1, +1]$
  - Prevents too large Q-values
  - Ensures gradients are well-conditioned

Deep RL in Atari Games

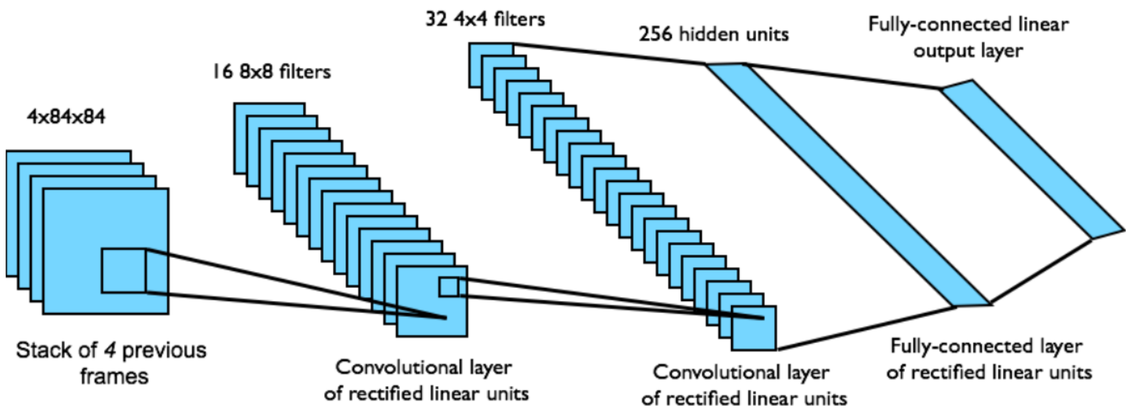


DQN in Atari

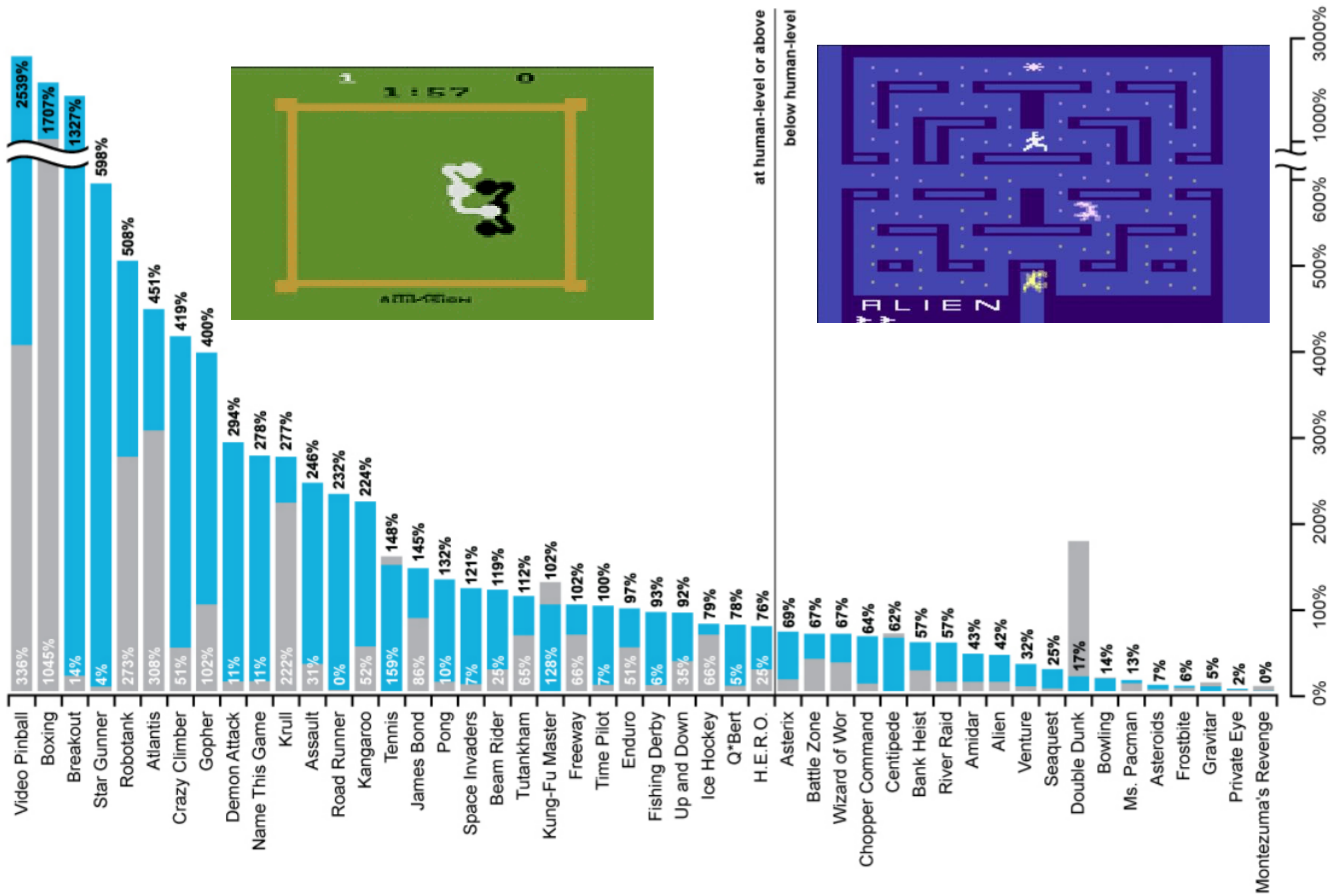
Goal: end-to-end learning of values $Q(s, a)$ from pixels

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

- Input: state is stack of raw pixels from last 4 frames
- Output: $Q(s, a)$ for all joystick/button positions a
- Reward is the score change for that step

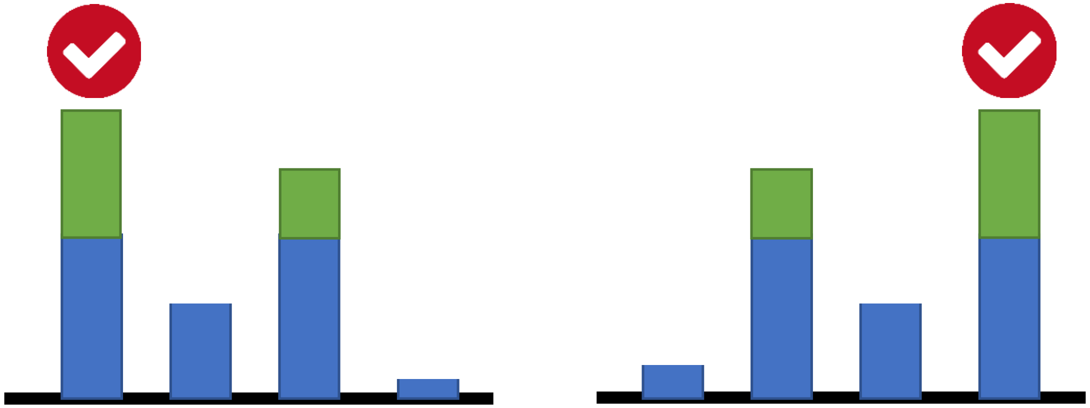


DQN in Atari


at human-level or above
below human-level
0%
100%
200%
300%
400%
500%
600%
1000%
3000%
Video Pinball 2539% 336%
Boxing 1707% 1045%
Breakout 1327% 14%
Star Gunner 598% 4%
Robotank 508% 273%
Atlantis 451% 308%
Crazy Climber 419% 51%
Gopher 400% 102%
Demon Attack 294% 11%
Name This Game 278% 11%
Krull 277% 222%
Assault 246% 31%
Road Runner 232% 0%
Kangaroo 224% 52%
Tennis 148% 159%
James Bond 145% 86%
Pong 132% 10%
Space Invaders 121% 7%
Beam Rider 119% 25%
Tutankham 112% 65%
Kung-Fu Master 102% 128%
Freeway 102% 66%
Time Pilot 100% 7%
Enduro 97% 51%
Fishing Derby 93% 6%
Up and Down 92% 35%
Ice Hockey 79% 66%
Q*Bert 78% 5%
H.E.R.O. 76% 25%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pacman 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
1:57
ALIEN

Other Improvements: Double DQN

Nature DQN $Q(s_t, a_t) \longleftrightarrow r_t + \gamma \max_a Q(s_{t+1}, a)$



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

Issue: tend to select the action that is over-estimated

Other Improvements: Double DQN

Nature DQN

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

Double DQN: remove upward bias caused by $\max_{a} Q(s, a, w)$

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

- Current Q-network w is used to select actions
- Older Q-network w^- is used to evaluate actions

Other Improvements: Prioritized Replay

Prioritized Replay: weight experience based on surprise

- Store experience in priority queue according to DQN error

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

Other Improvements: Dueling Network

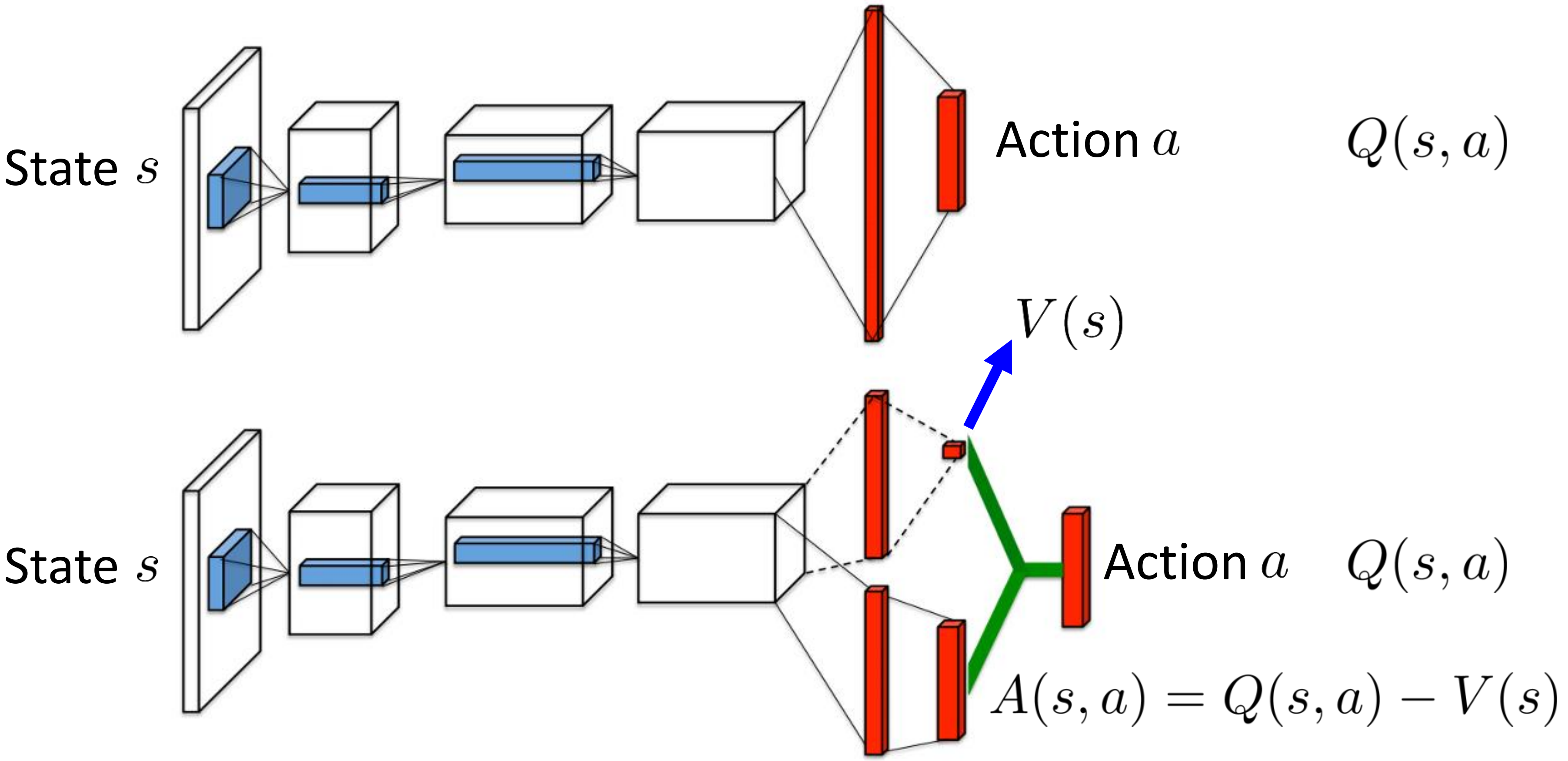
Dueling Network: split Q-network into two channels

$$Q(s, a) = V(s) + A(s, a)$$

- Action-independent value function $V(s)$
  - Value function estimates how good the state is
- Action-dependent advantage function $A(s, a)$
  - Advantage function estimates the additional benefit

Wang et al., "Dueling Network Architectures for Deep Reinforcement Learning", arXiv preprint, 2015.

Other Improvements: Dueling Network



Policy-Based Approach

LEARNING AN ACTOR

On-Policy v.s. Off-Policy

On-policy: The agent learned and the agent interacting with the environment is the same

Off-policy: The agent learned and the agent interacting with the environment is different

Goodness of Actor

An episode is considered as a trajectory τ

- $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$
- Reward: $R(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r_t$

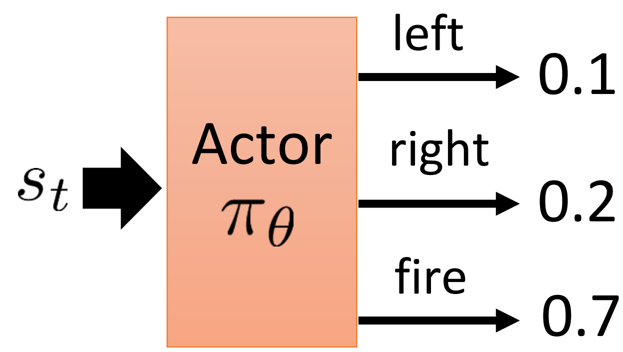
$$P(\tau \mid \theta) =$$

$$p(s_1)p(a_1 \mid s_1, \theta)p(r_1, s_2 \mid s_1, a_1)p(a_2 \mid s_2, \theta)p(r_2, s_3 \mid s_2, a_2)\cdots$$

$$= p(s_1) \prod_{t=1}^{T} p(a_t \mid s_t, \theta) p(r_t, s_{t+1} \mid s_t, a_t)$$

not related to your actor

control by your actor



$$p(a_t = \text{fire} \mid s_t, \theta) = 0.7$$

Goodness of Actor

An episode is considered as a trajectory τ

- $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$
- Reward: $R(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r_t$

We define $\mathcal{R}(\theta)$ as the *expected value* of reward

- If you use an actor to play game, each τ has $P(\tau|\theta)$ to be sampled

$$\mathcal{R}(\theta) = \sum_{\tau} R(\tau) P(\tau \mid \theta) \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n)$$

  - Use π_θ to play the game N times, obtain $\{\tau^1, \tau^2, \cdots, \tau^N\}$
  - Sampling τ from $P(\tau|\theta)$ N times

sum over all possible trajectory

Deep Policy Networks

Represent policy by deep network with weights

Objective is to maximize total discounted reward by SGD

$$\mathcal{R}(\theta) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \mid \pi(\cdot, \theta)\right]$$

Update the model parameters iteratively

$$\theta^* = \arg\max_\theta \mathcal{R}(\theta)$$

$$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

Policy Gradient $\mathcal{R}(\theta) = \sum_\tau R(\tau) P(\tau \mid \theta)$

Gradient assent to maximize the expected reward

$$\nabla \mathcal{R}(\theta) = \sum_\tau R(\tau) \nabla P(\tau \mid \theta) = \sum_\tau R(\tau) P(\tau \mid \theta) \frac{\nabla P(\tau \mid \theta)}{P(\tau \mid \theta)}$$

do not have to be differentiable

can even be a black box

$$= \sum_\tau R(\tau) P(\tau \mid \theta) \nabla \log P(\tau \mid \theta)$$

$$\frac{d \log f(x)}{dx} = \frac{1}{f(x)} \frac{df(x)}{dx}$$

use π_θ to play the game N times, obtain $\{\tau^1, \tau^2, \cdots, \tau^N\}$

$$\approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \underline{\nabla \log P(\tau^n \mid \theta)}$$

Policy Gradient $\nabla \log P(\tau \mid \theta)$

An episode trajectory $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$

$$P(\tau \mid \theta) = p(s_1) \prod_{t=1}^{T} p(a_t \mid s_t, \theta) p(r_t, s_{t+1} \mid s_t, a_t)$$

$\log P(\tau \mid \theta)$

$$= \log p(s_1) \sum_{t=1}^{T} \log p(a_t \mid s_t, \theta) + \log p(r_t, s_{t+1} \mid s_t, a_t)$$

$$\nabla \log P(\tau \mid \theta) = \sum_{t=1}^{T} \nabla \log p(a_t \mid s_t, \theta)$$

ignore the terms not related to θ

Policy Gradient

Gradient assent for iteratively updating the parameters

$$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

$$\nabla \mathcal{R}(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla \log P(\tau^n \mid \theta)$$

$$= \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n \mid s_t^n, \theta)$$

- If τ^n machine takes a_t^n when seeing s_t^n

$R(\tau^n) > 0$ → Tuning θ to increase $p(a_t^n \mid s_t^n)$

$R(\tau^n) < 0$ → Tuning θ to decrease $p(a_t^n \mid s_t^n)$

Important: use cumulative reward $R(\tau^n)$ of the whole trajectory τ^n instead of immediate reward r_t^n

Policy Gradient

Given actor parameter θ

$$\tau^1: \ (s_1^1, a_1^1) \quad R(\tau^1) \qquad \tau^2: \ (s_1^2, a_1^2) \quad R(\tau^2)$$

$$(s_2^1, a_2^1) \quad R(\tau^1) \qquad (s_2^2, a_2^2) \quad R(\tau^2)$$

$$\vdots \quad \vdots \qquad \vdots \quad \vdots$$

data collection

model update

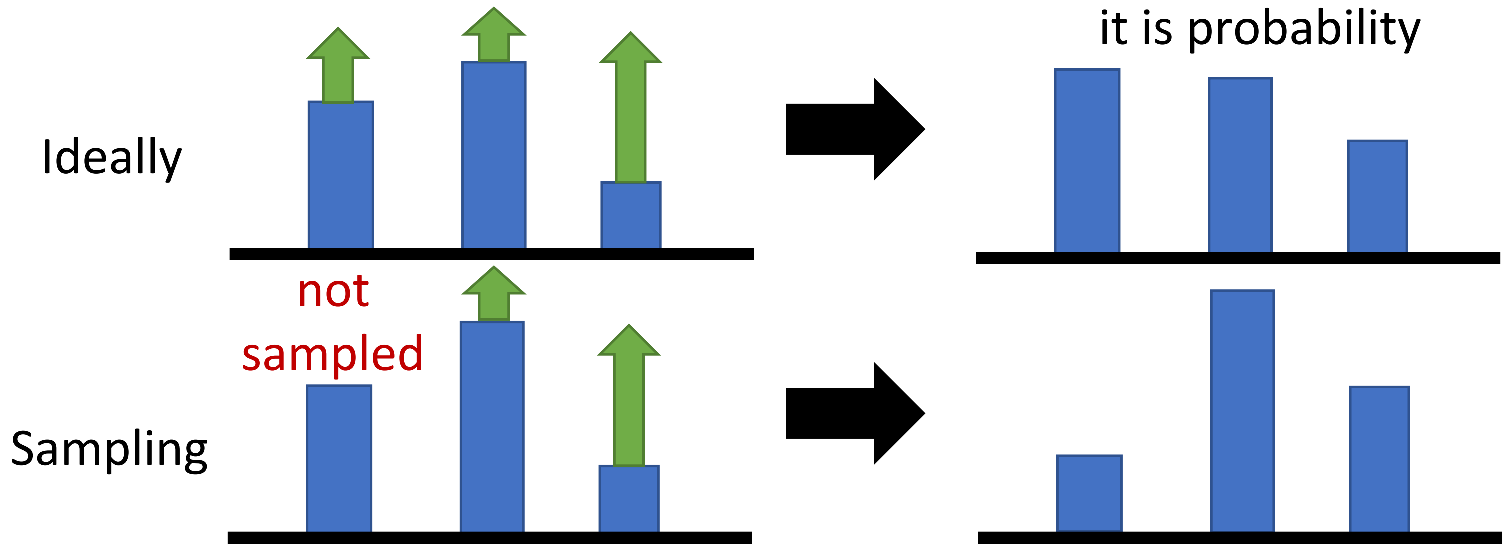
$$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

$$\nabla \mathcal{R}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n \mid s_t^n, \theta)$$

Improvement: Adding Baseline

$$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

$$\nabla \mathcal{R}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (R(\tau^n) - b) \nabla \log p(a_t^n \mid s_t^n, \theta)$$



Issue: the probability of the actions not sampled will decrease

Actor-Critic Approach

LEARNING AN ACTOR & A CRITIC

Actor-Critic (Value-Based + Policy-Based)

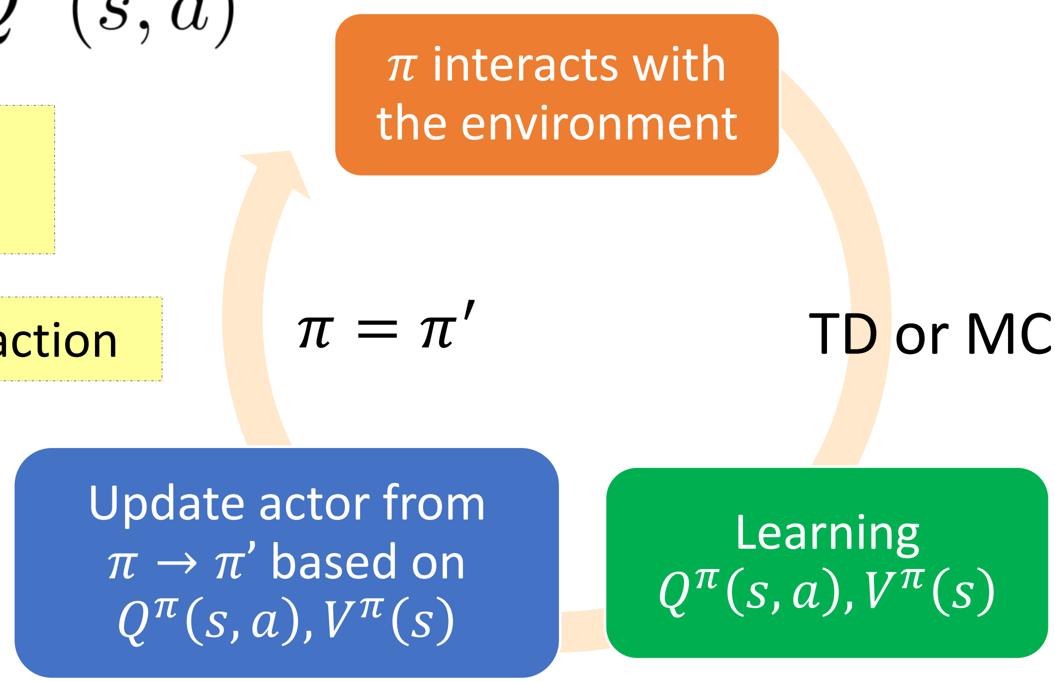
Estimate value function $Q^{\pi}(s,a), V^{\pi}(s)$

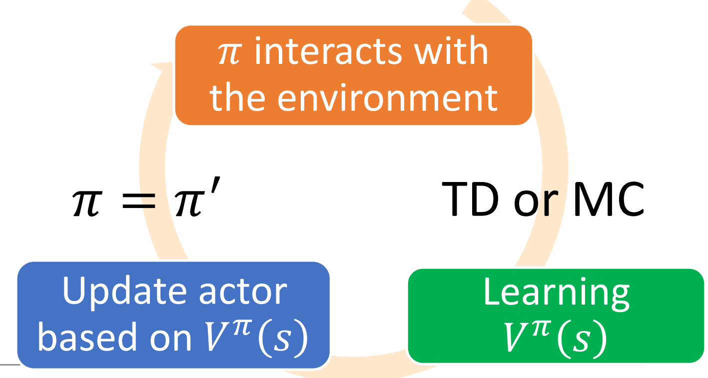
Update policy based on the value function evaluation π

$$\pi'(s) = \arg\max_{a} Q^{\pi}(s,a)$$

π is a actual function that maximizes the value

may works for continuous action





Advantage Actor-Critic

Learning the policy (actor) using the value evaluated by critic

$$\theta^{\pi'} \leftarrow \theta^{\pi} + \eta \nabla \mathcal{R}(\theta^{\pi})$$

$$\nabla \mathcal{R}(\theta^{\pi}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n \mid s_t^n, \theta^{\pi})$$

baseline is added

evaluated by critic

Advantage function: $r_t^n - \left(V^{\pi}(s_t^n) - V^{\pi}(s_{t+1}^n)\right)$

the reward r_t^n we truly obtain when taking action a_t^n

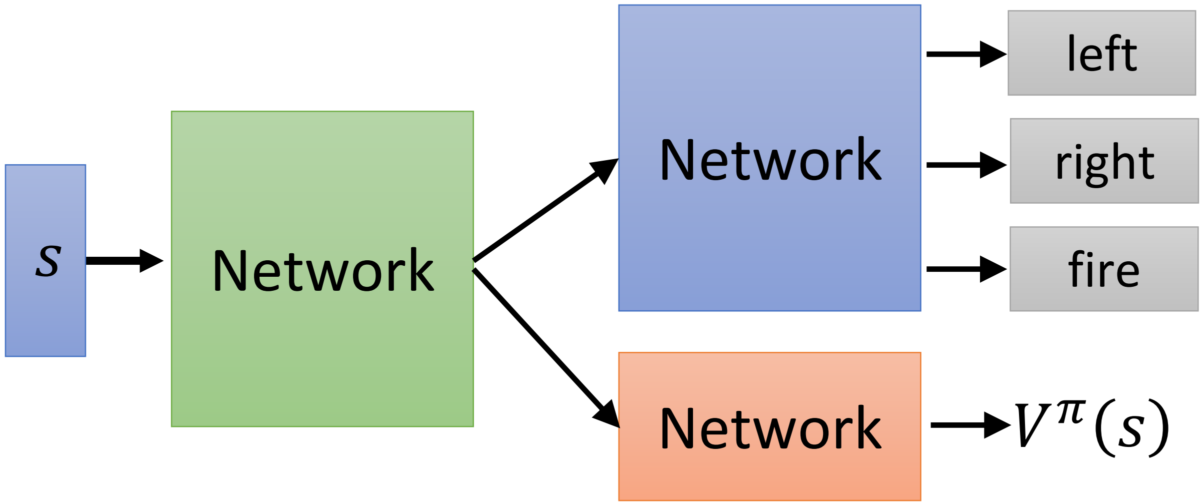
expected reward r_t^n we obtain if we use actor π

- Positive advantage function $\leftrightarrow$ increasing the prob. of action a_t^n
- Negative advantage function $\leftrightarrow$ decreasing the prob. of action a_t^n

Advantage Actor-Critic

Tips

- The parameters of actor $\pi(s)$ and critic $V^{\pi}(s)$ can be shared



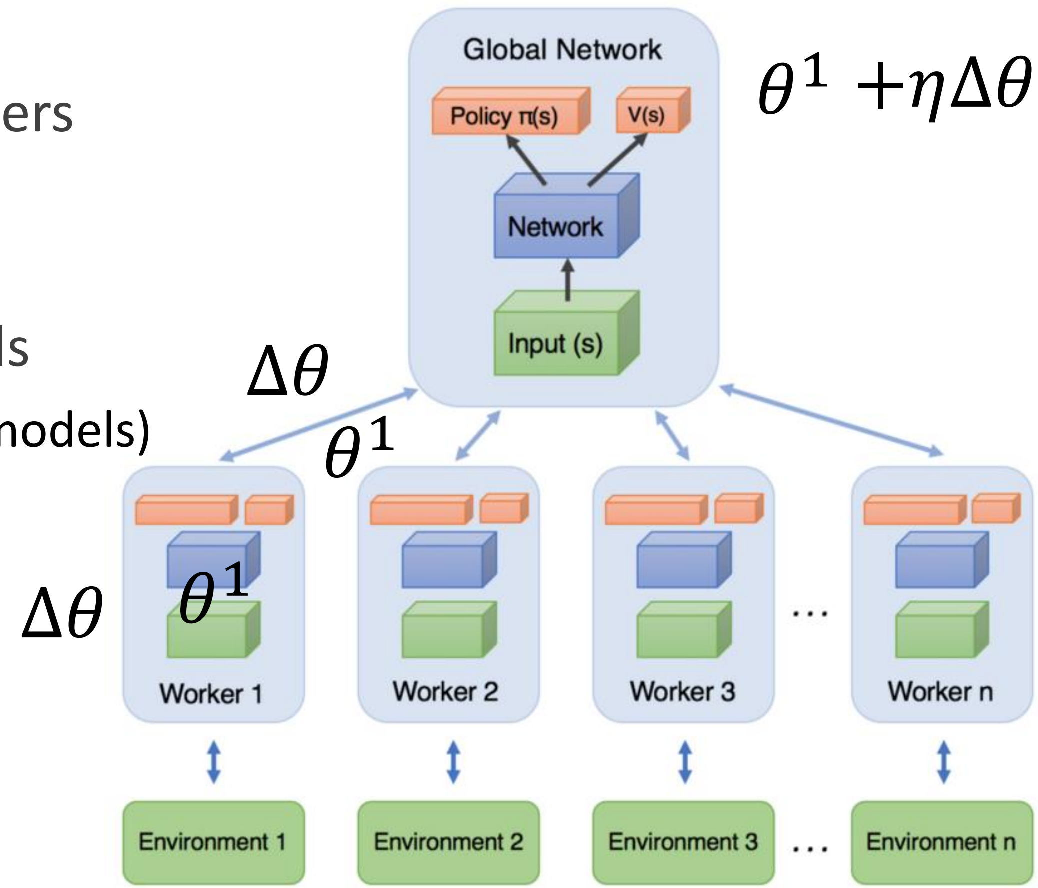
- Use output entropy as regularization for $\pi(s)$
  - Larger entropy is preferred → exploration

Asynchronous Advantage Actor-Critic (A3C)

Asynchronous

1. Copy global parameters
2. Sampling some data
3. Compute gradients
4. Update global models

(other workers also update models)



Pathwise Derivative Policy Gradient

Original actor-critic tells that a given action is good or bad

Pathwise derivative policy gradient tells that which action is good

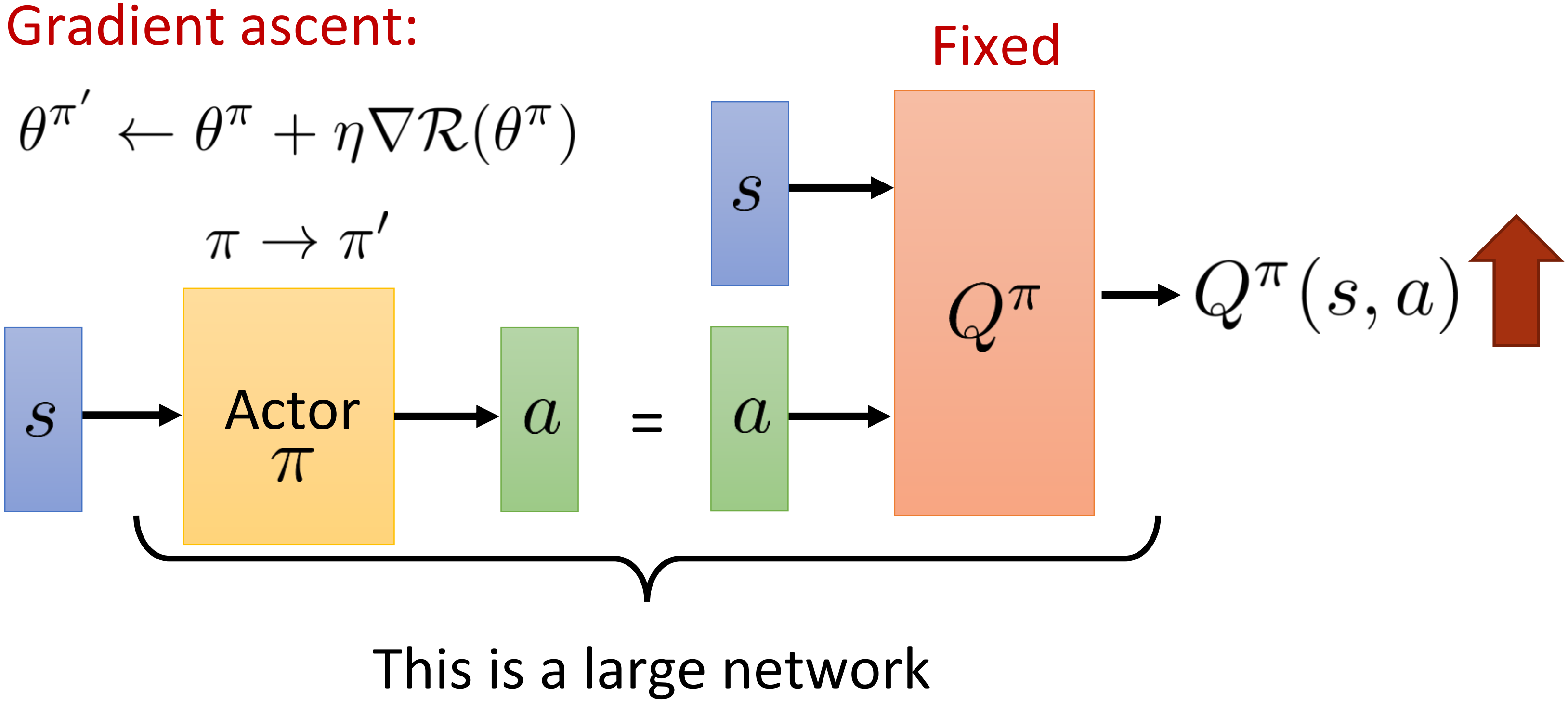
Pathwise Derivative Policy Gradient

$$\pi'(s) = \arg\max_a Q^\pi(s, a)$$ ← an actor's output

Gradient ascent:

$$\theta^{\pi'} \leftarrow \theta^\pi + \eta \nabla \mathcal{R}(\theta^\pi)$$

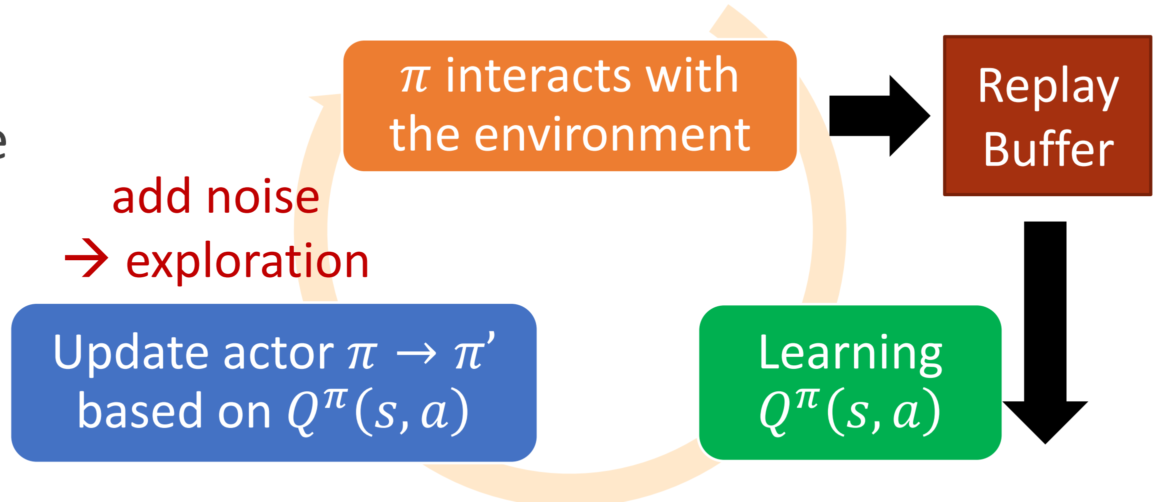
$$\pi \rightarrow \pi'$$



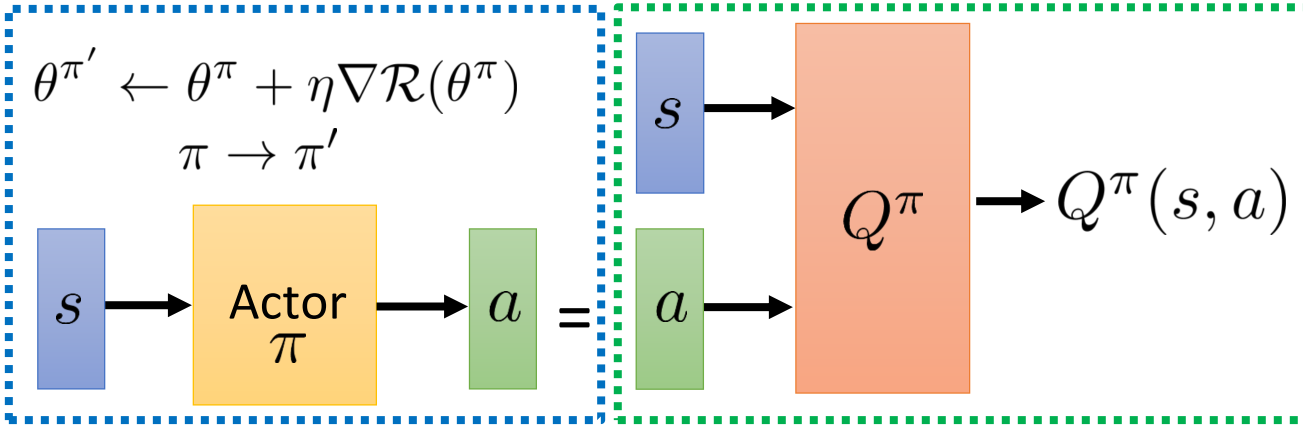
Deep Deterministic Policy Gradient (DDPG)

Idea

- Critic estimates value of current policy by DQN
- Actor updates policy in direction that improves Q



Critic provides loss function for actor



DDPG Algorithm

Initialize critic network θ^Q and actor network θ^π

Initialize target critic network $\theta^{Q'} = \theta^Q$ and target actor network $\theta^{\pi'} = \theta^\pi$

Initialize replay buffer R

In each iteration

- Use $\pi(s) + \text{noise}$ to interact with the environment, collect a set of $\{s_t, a_t, r_t, s_{t+1}\}$, put them in R
- Sample N examples $\{s_n, a_n, r_n, s_{n+1}\}$ from R
- Update critic Q to minimize $\sum_n (\hat{y}_n - Q(s_n, a_n))^2$

  $\hat{y}_n = r_n + Q'(s_{n+1}, \pi'(s_{n+1}))$ using target networks

- Update actor π to maximize $\sum_n Q(s_n, \pi(s_n))$
- Update target networks:

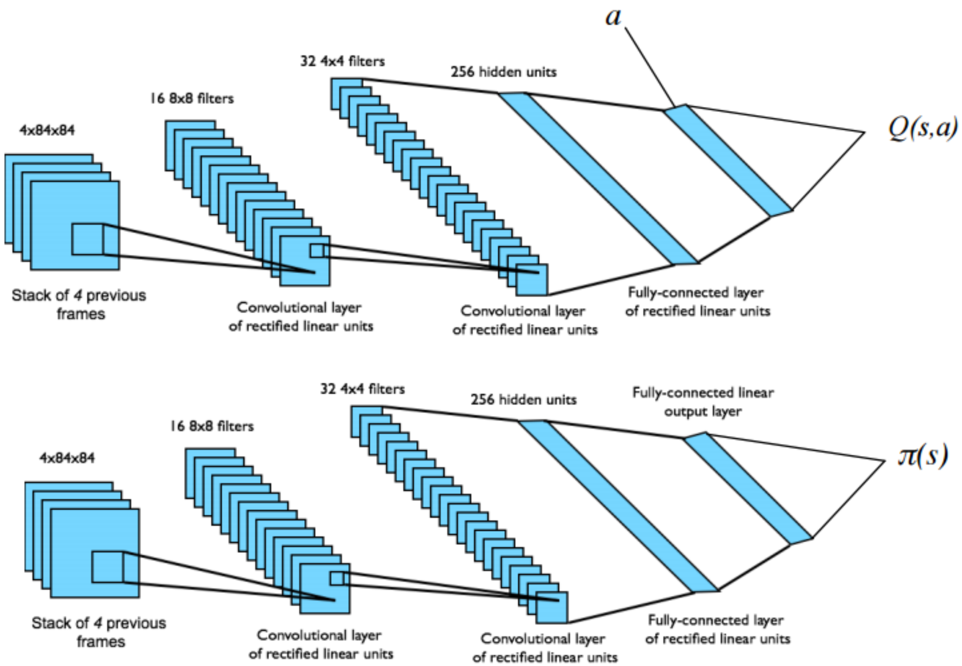
$$\theta^{\pi'} \leftarrow m\theta^\pi + (1-m)\theta^{\pi'}$$
$$\theta^{Q'} \leftarrow m\theta^Q + (1-m)\theta^{Q'}$$

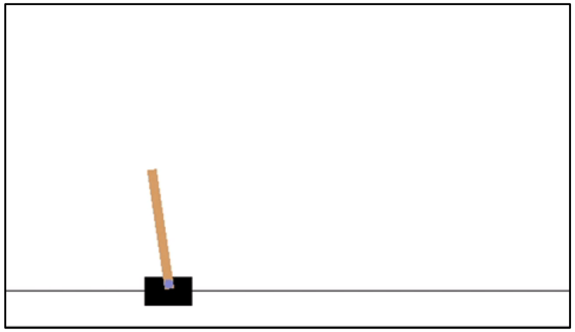
the target networks update slower

DDPG in Simulated Physics

Goal: end-to-end learning of control policy from pixels

- Input: state is stack of raw pixels from last 4 frames
- Output: two separate CNNs for Q and π





Model-Based

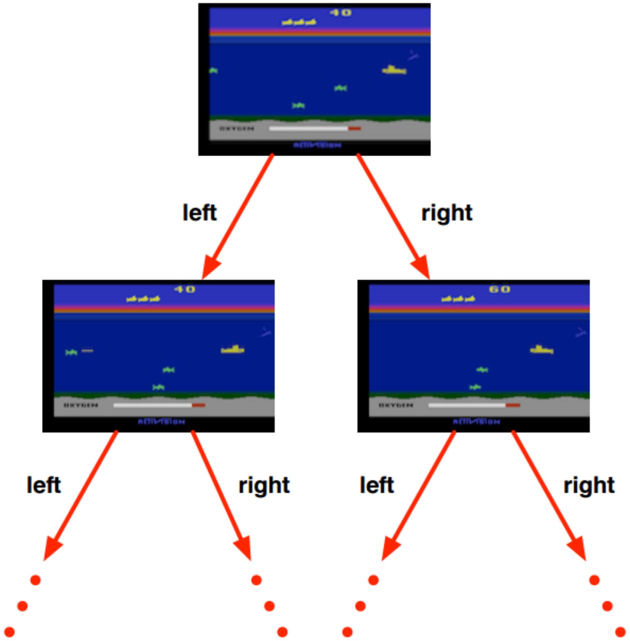
Agent’s Representation of the Environment

Model-Based Deep RL

Goal: learn a transition model of the environment and plan based on the transition model

$$p(r, s' \mid s, a)$$

Objective is to maximize the measured goodness of model



Model-based deep RL is challenging, and so far has failed in Atari

Issues for Model-Based Deep RL

Compounding errors

- Errors in the transition model compound over the trajectory
- A long trajectory may result in totally wrong rewards

Deep networks of value/policy can “plan” implicitly

- Each layer of network performs arbitrary computational step
- n-layer network can “lookahead” n steps

Model-Based Deep RL in Go

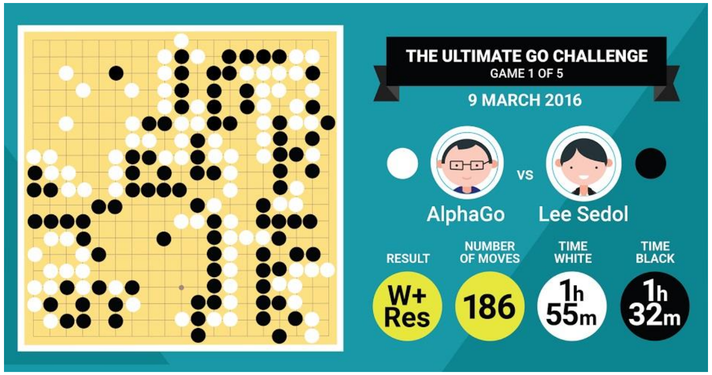
Monte-Carlo tree search (MCTS)

- MCTS simulates future trajectories
- Builds large lookahead search tree with millions of positions
- State-of-the-art Go programs use MCTS

Convolutional Networks

- 12-layer CNN trained to predict expert moves
  - Raw CNN (looking at 1 position, no search at all) equals performance of MoGo with 105 position search tree

1st strong Go program



Silver, et al., “Mastering the game of Go with deep neural networks and tree search,” Nature, 2016.

https://deepmind.com/alphago/

OpenAI Universe

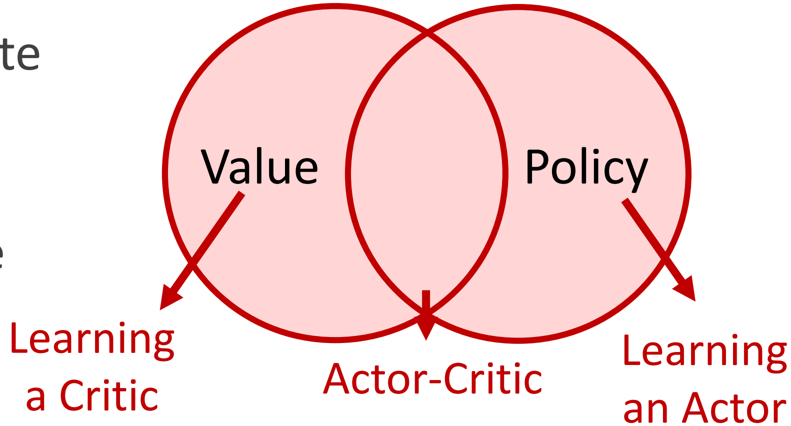
Software platform for measuring and training an AI's general intelligence via the OpenAI gym environment

Concluding Remarks

RL is a general purpose framework for **decision making** under interactions between agent and environment

An RL agent may include one or more of these components

- Value function: how good is each state and/or action
- Policy: agent's behavior function
- Model: agent's representation of the environment



RL problems can be solved by end-to-end deep learning

Reinforcement Learning + Deep Learning = AI

References

Course materials by David Silver: http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html

ICLR 2015 Tutorial: http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

ICML 2016 Tutorial: http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf