《强化学习与控制》

--

Deep RL

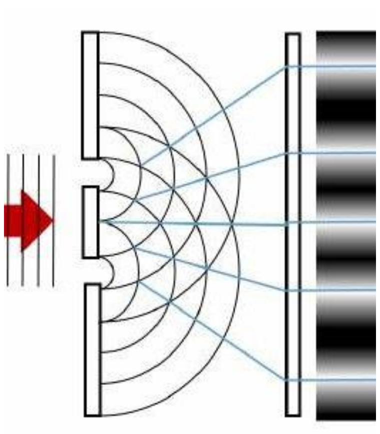
Shengbo Eben Li
(李升波)

Intelligent Driving Lab (*i*DLab)

Tsinghua University

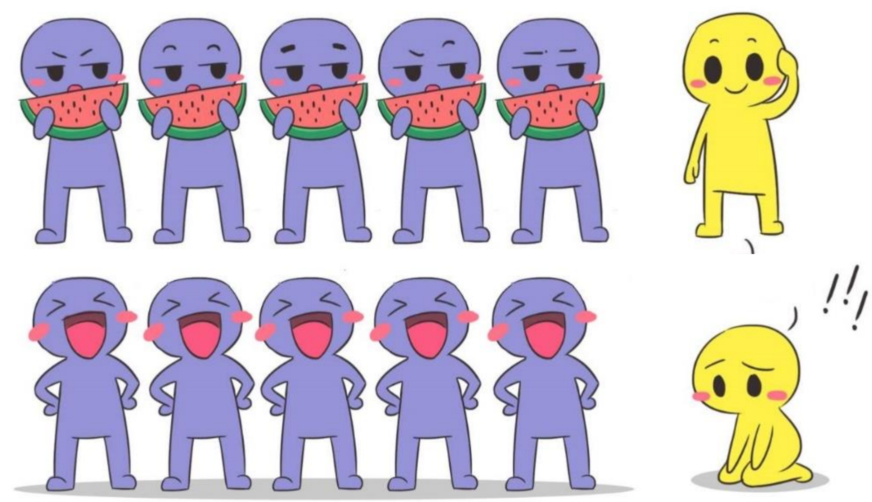
Braid in your heart is invisible!

There is no law except the law that there is no law.



- John A. Wheeler

Wheeler's Delayed Choice Experiment



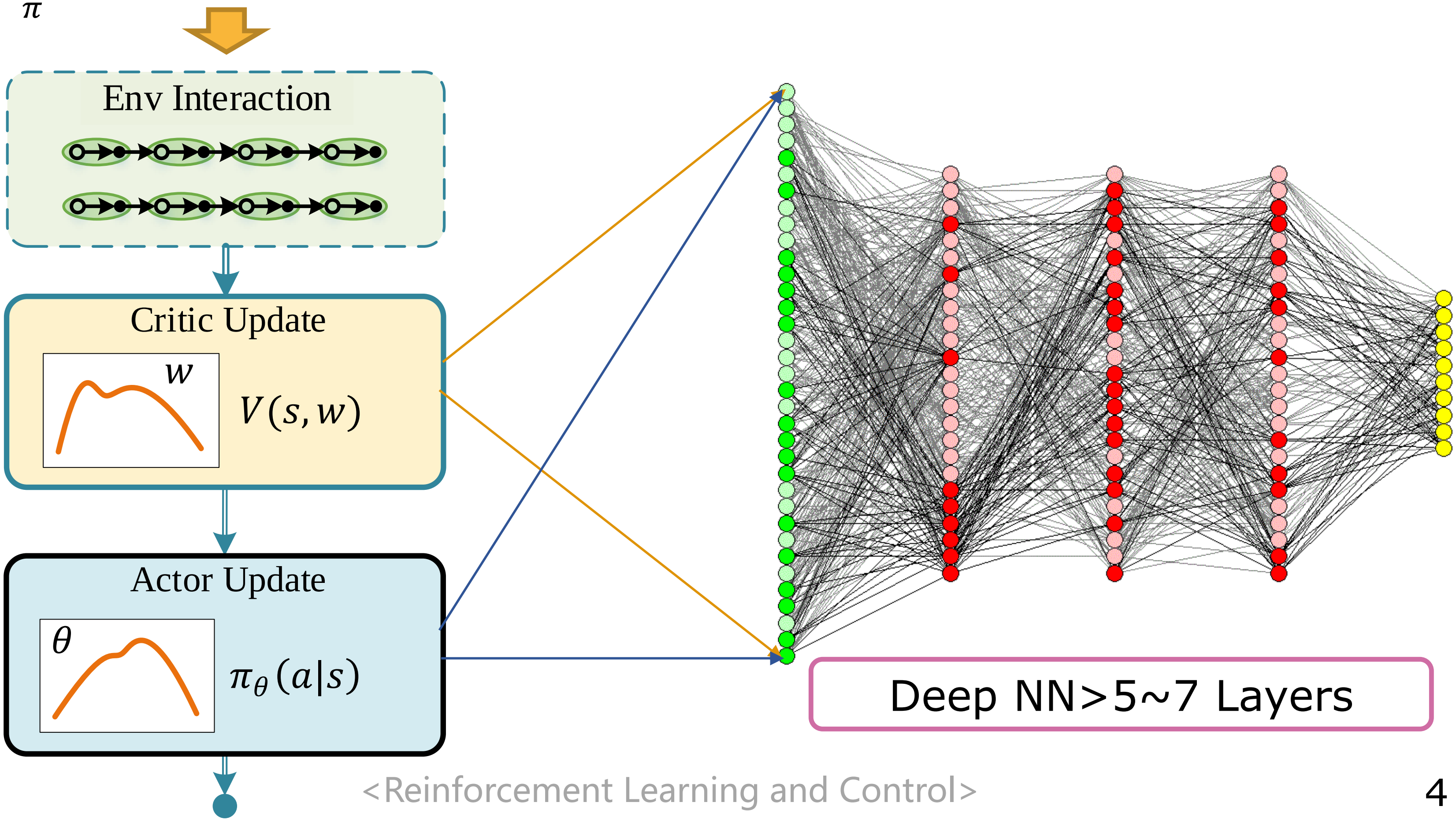
Outline

1. Motivation of Deep RL
2. Deep Neural Network
3. Challenges to be Solved
4. Typical DRL Algorithms

What is Deep RL?

Deep RL = RL + Deep NN

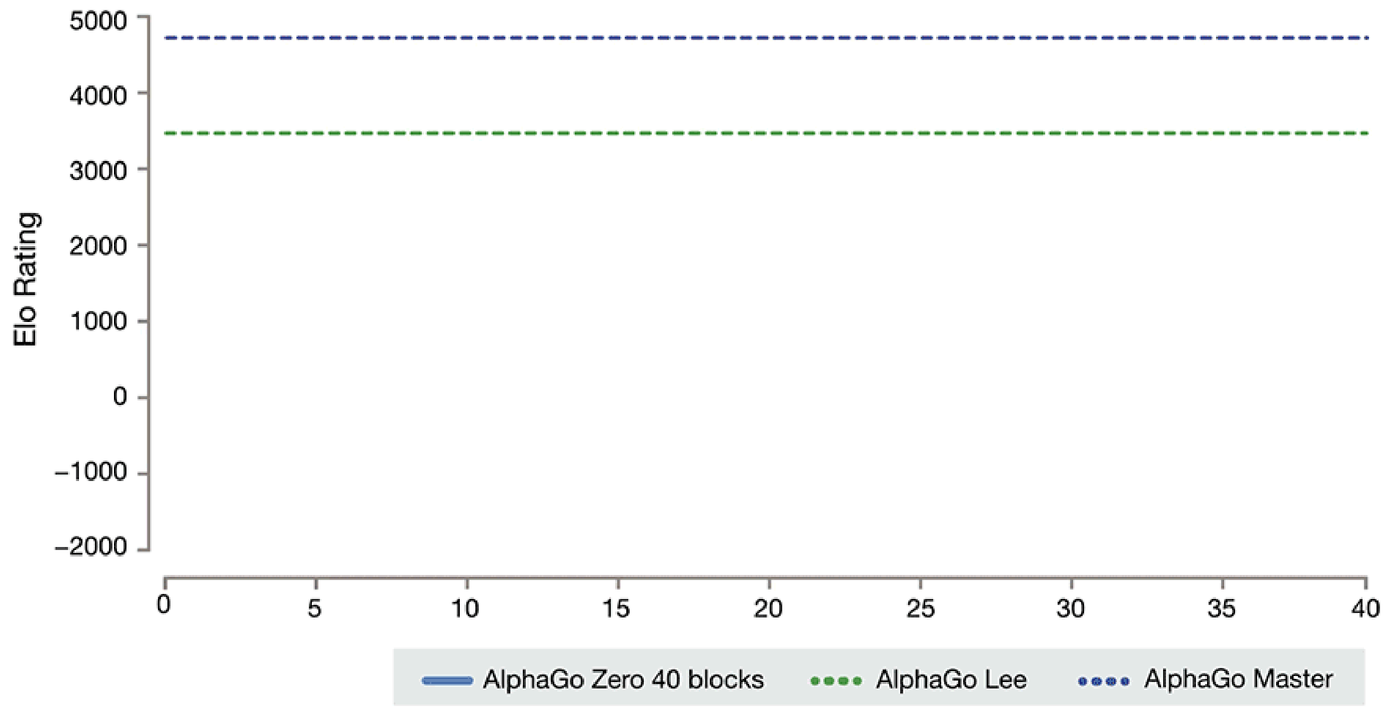
$$\max_{\pi}/\min J(\pi) = \mathbb{E}_{s \in d_{\text{init}}(s)}\{v^{\pi}(s)\}$$



Motivation of Deep RL

Deep RL in games

- **Japanese chess:** train for 2 hours to surpass Elmo
- **Chess:** train for 4 hours to surpass Stockfish
- **Go:** train for 3 days (AlphaZero) to surpass AlphaGo Lee



Motivation of Deep RL

Issues to be solved

- (1) Non-iid data breaks the guarantee of NN convergence
- (2) Mutable update targets lead to diverged policy/value
- (3) Inhomogeneous overestimation negatively affects policy
- (4) Low efficiency to sample the whole state-action space

Tricks in Deep RL

- (1) Experience replay
- (2) Parallel exploration
- (3) Separated target network
- (4) Delayed policy updates
- (5) Constrained policy updates
- (6) Clipped actor criterion
- (7) Double Q-functions
- (8) Bounded double Q-functions
- (9) Distributional return function
- (10) Entropy regularization
- (11) Soft value function

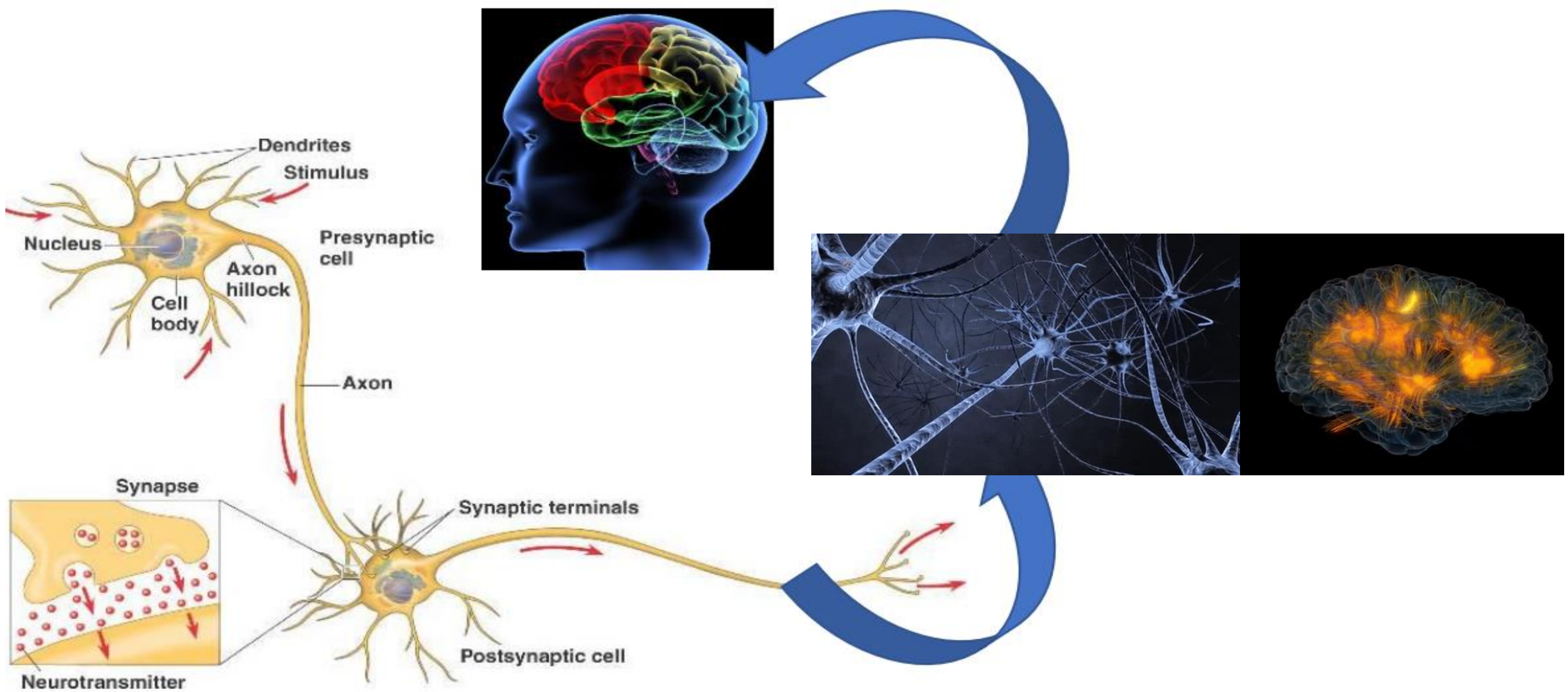
Outline

1. Motivation of Deep RL
2. Deep Neural Network
3. Challenges to be Solved
4. Typical DRL Algorithms

Neurons in Human Brain

Human Brain

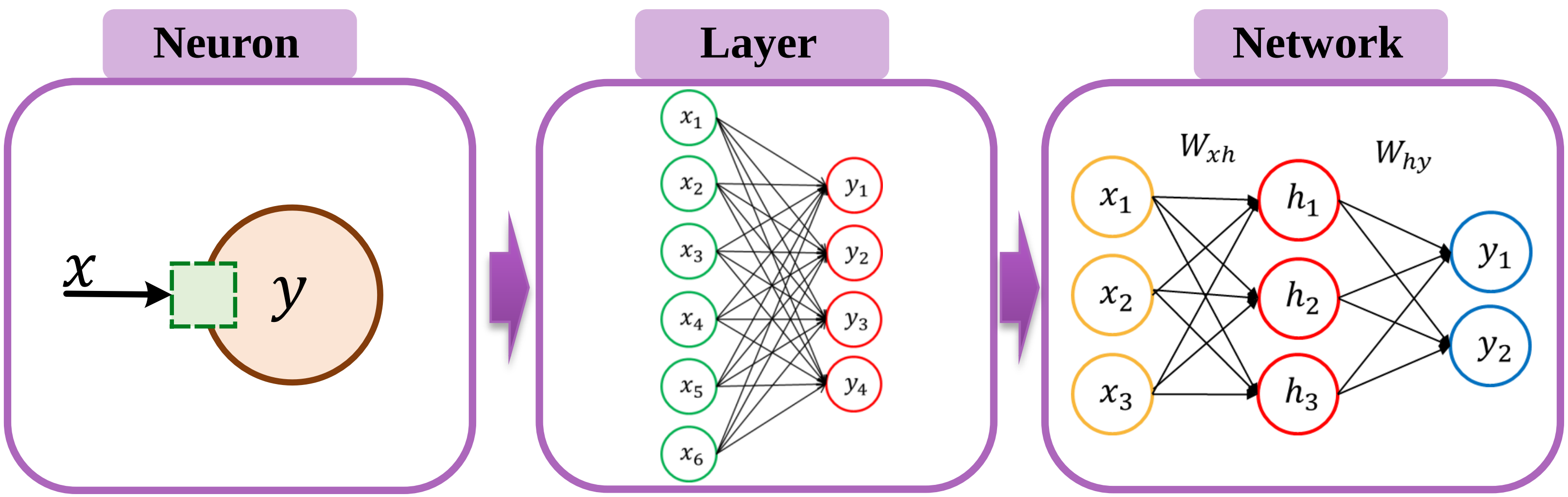
- Large number (about 80 billons) of interconnected neurons
- Each neuron has about 10 thousands connections on average
- Learning occurs by changing the effectiveness of synapses



Artificial Neural Network

Layered Structure

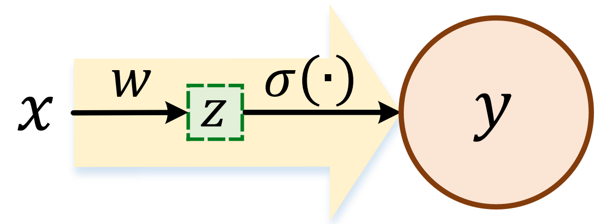
- Neuron: basic computing unit of neural network
- Layer: combination of neurons in the width direction
- Network: connect layers along the depth direction



Artificial Neural Network

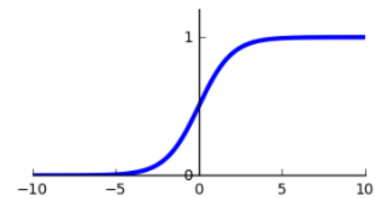
Neuron

- (1) Affine transformation
  - Linear transformation + offset
  - Superposition of linear functions is still linear
- (2) Non-linear activation function

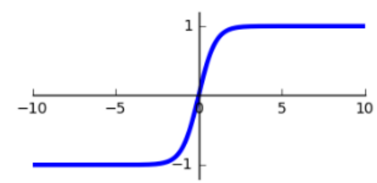


Activation function

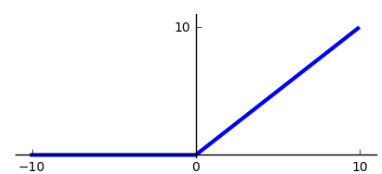
- logistic $\sigma(z) = \dfrac{1}{1 + \mathrm{e}^{-z}}$



- Tanh $\sigma(z) = \dfrac{\mathrm{e}^{z} - \mathrm{e}^{-z}}{\mathrm{e}^{z} + \mathrm{e}^{-z}}$



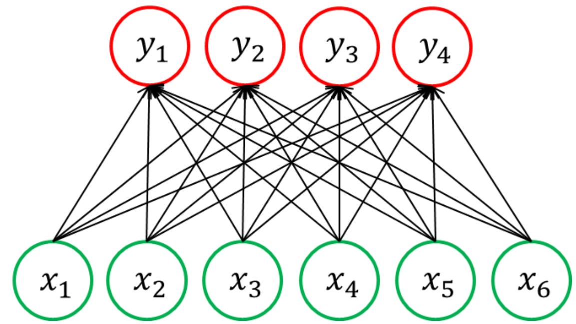
- ReLU $\sigma(z) = \begin{cases} 0, z \leq 0 \\ z, z > 0 \end{cases}$



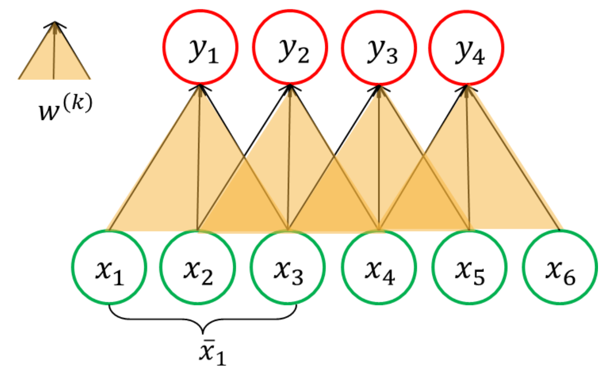
Artificial Neural Network

Layers

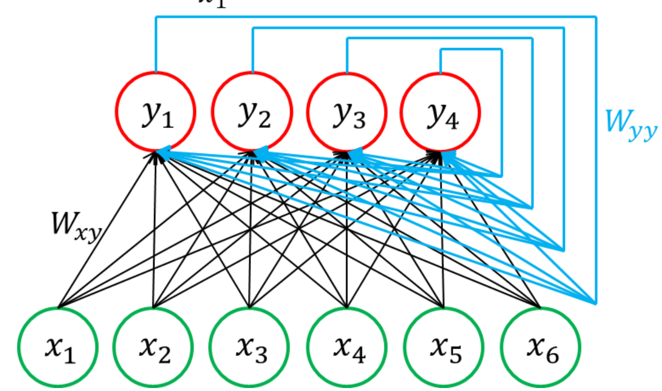
- Fully connected (FC) layer



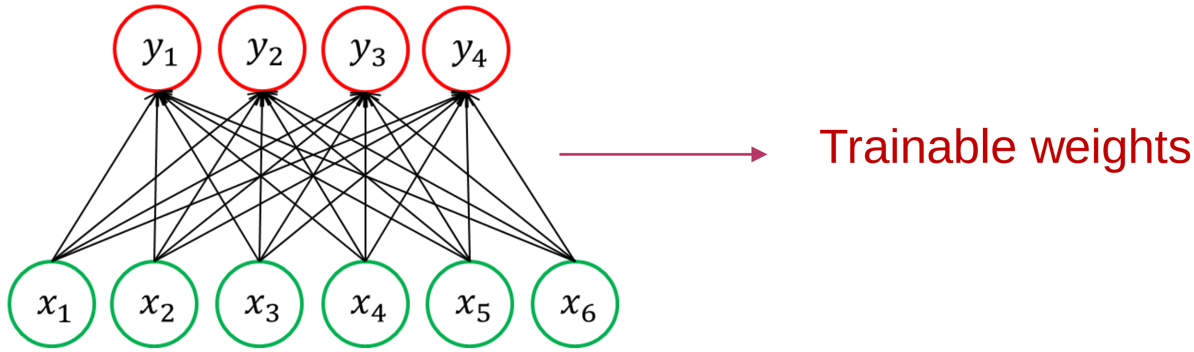
- Convolutional (CONV) layer



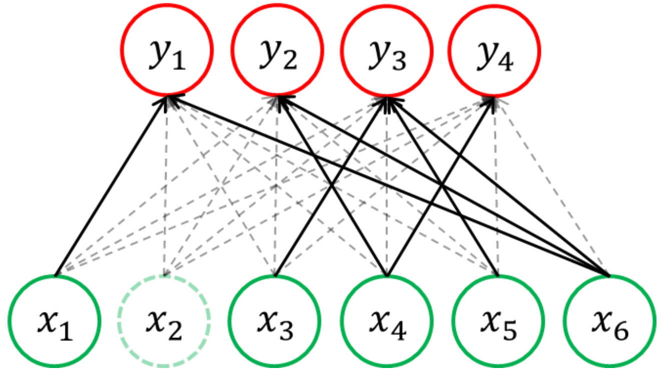
- Recurrent (REC) layer



Fully connected (FC) layer



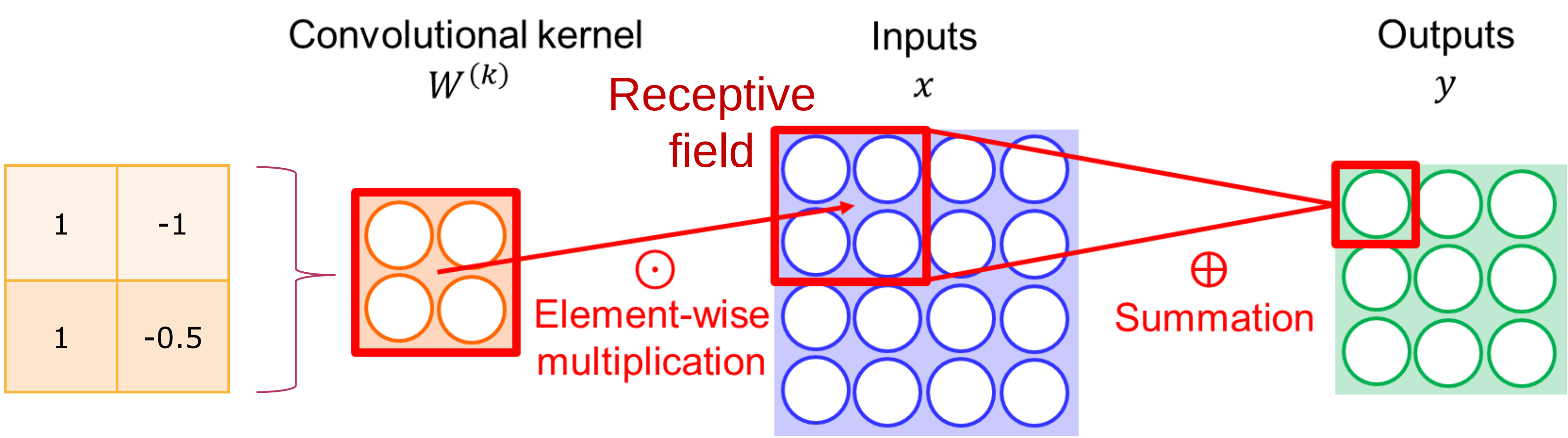
- "Dropout technique" to conquer the overfitting issue
  - Randomly remove some neurons
  - Randomly remove some connections



Sparse connection

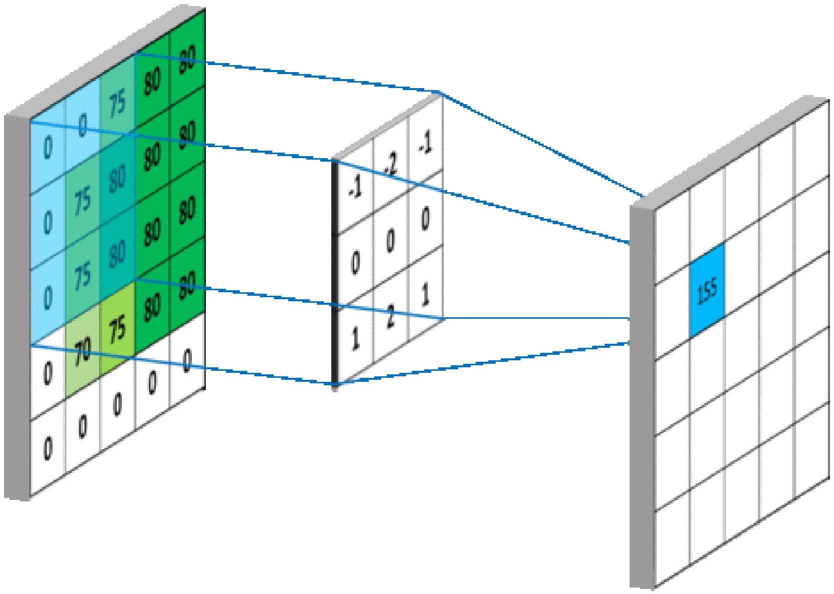
Convolutional (CONV) layer

- Convolution kernels in CONV = Trainable weights in FC
- Receptive field: convolution only works with a restricted subarea
  - Local connectivity: sparse local connection
  - Weight sharing: each receptive field shares same kernels
- Largely reduce the number of trainable weights

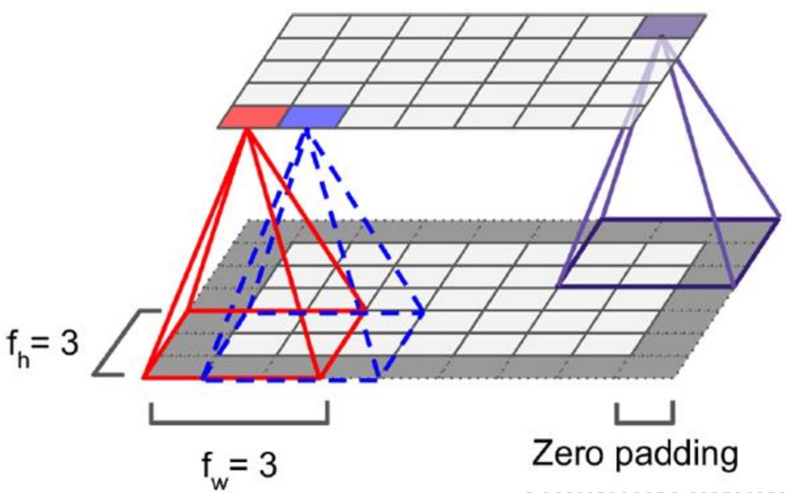


2D convolution: single kernel channel

Key parameters of convolutional layer



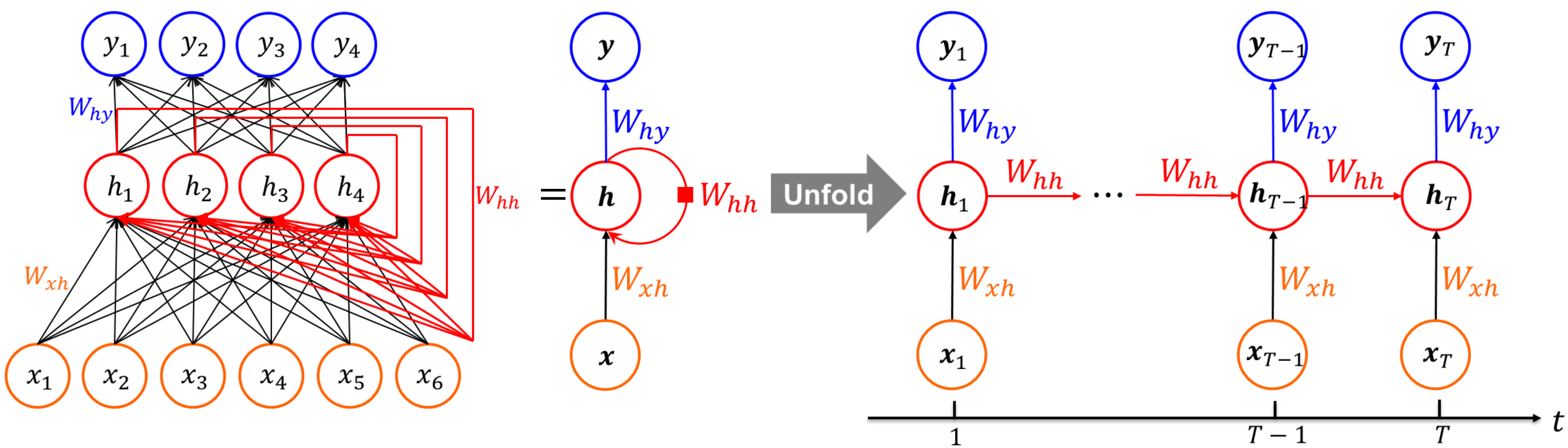
- number of kernels
- number of kernel channels
- kernel size
- sliding stride
- padding size



3X3 kernel,
One sliding stride
Zero padding

Recurrent (REC) layer

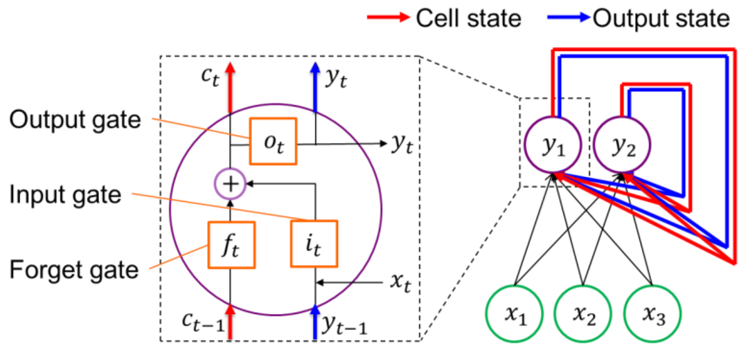
- The output of previous step is fed as the input to current step
- Hidden state carries pertinent information of previous inputs
- Once unfolded in time, feedforward networks with every layer sharing weights
- When training, gradients easily explode or vanish



Recurrent (REC) layer

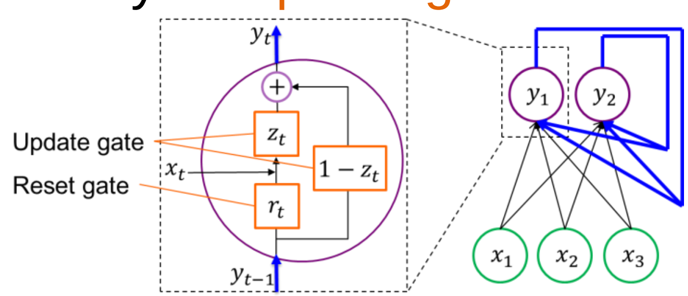
(1) Long short-term memory (LSTM)

- Gradients neither explode nor vanish through cell path
- A cell is composed by input, forget and output gates
- Gates regulate the information flow into and out of the cell



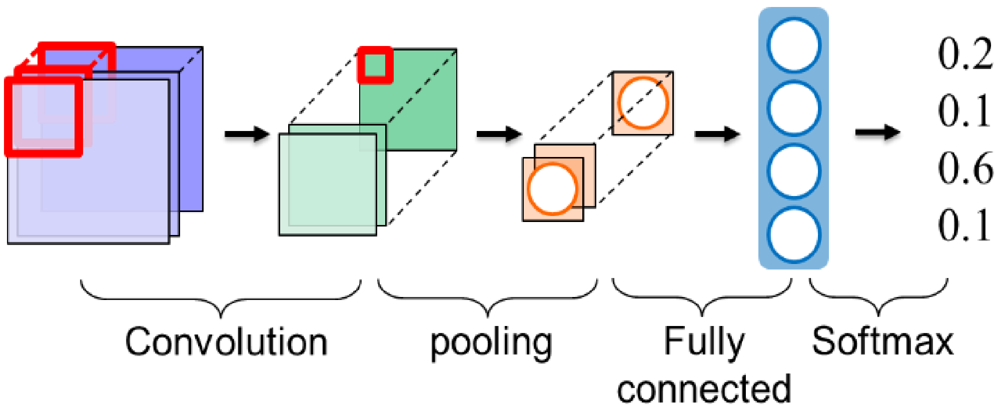
(2) Gated recurrent unit (GRU)

- A cell is composed by an update gate and a reset gate



Typical neural networks

- Convolutional neural networks (CNNs) specialize in processing a grid of values with spatial information like images
- Recurrent neural networks (RNNs) specialize in processing a sequence of values or temporal information like languages



	Num of conv layer	Kernel Size	Num of kernel	Num of kernel channels	sliding stride
LeNet	2	5	20,50	1,20	1
AlexNet	5	3,5,11	96~384	3~256	1,4
GoogLeNet	21	1,3,5,7	16~384	3~832	1,2
VGG-16	13	3	64~512	3~512	1
ResNet-152	151	1,3,7	64~2048	3~2048	1,2
MobileNet	27	1,3	32~1024	3~1024	1,2

Loss Functions in Training NN

Cross entropy

- Measure the error between predicted distribution p and target distribution p^{target}
- Often used for classification where outputs are interpreted as membership probabilities

$$J \stackrel{\text{def}}{=} \mathcal{H}(p^{\text{target}}, p) = -\sum_i p_i^{\text{target}} \log(p_i)$$

Mean square error (MSE)

- The average of the squared differences between predicted values y and target values y^{target}
- Often used for regression where outputs are quantities

$$J \stackrel{\text{def}}{=} \text{MSE}(y^{\text{target}}, y) = \frac{1}{n}\sum_{i=1}^{n}(y_i^{\text{target}} - y_i)^2$$

Training of Neural Network

Gradient descent (GD)

- A first-order optimization algorithm that adjusts trainable weights according to the network error

$$w \leftarrow w - \eta \frac{\partial J(y^{\text{target}}, y)}{\partial w}$$

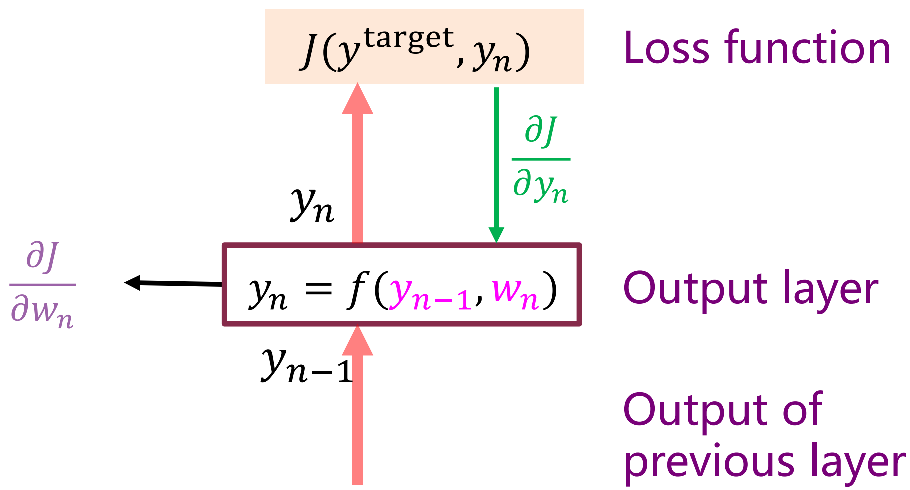
$$\frac{\partial J(y^{\text{target}}, y)}{\partial w} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial w}$$

Backpropagation (BP)

- Allow the error information from the loss function to flow backward through each layer
- Compute the gradients of each layer by the chain rule with a specific order of operations

Training of Neural Network

Backpropagation (BP) - for output layer



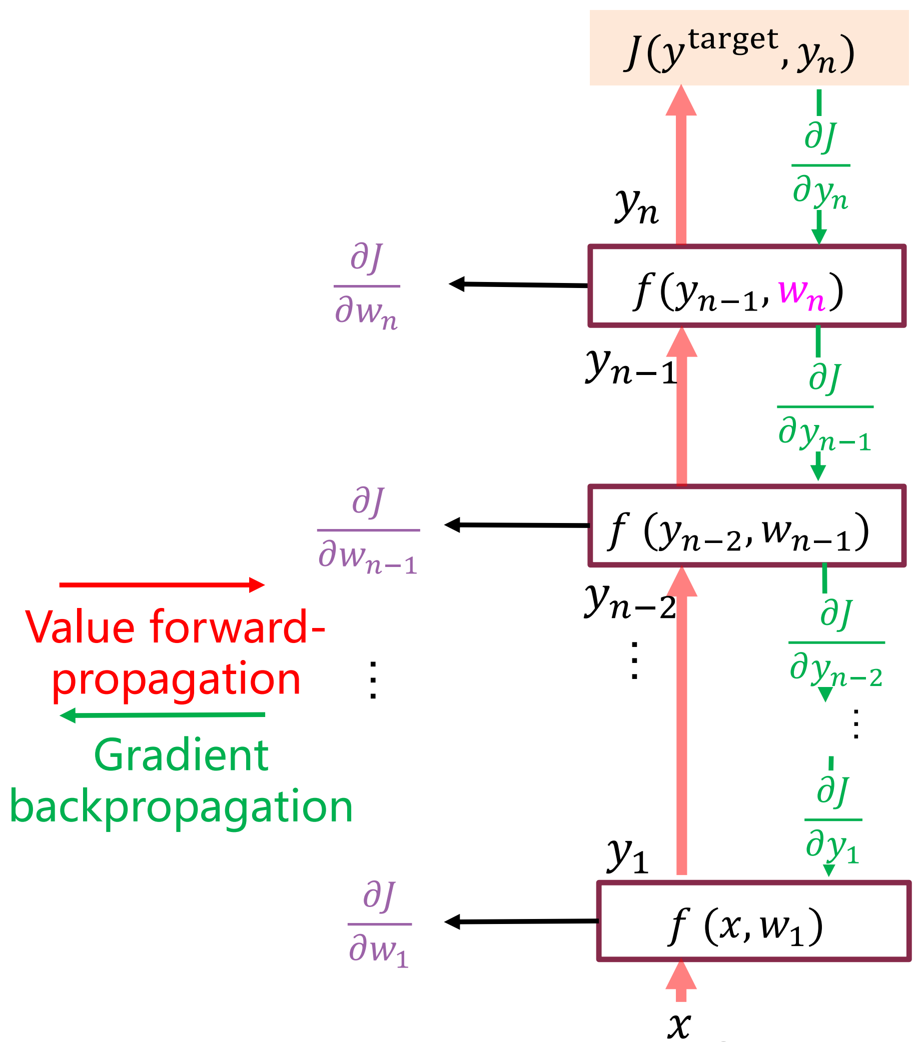
Only use in output layer

$$\frac{\partial J}{\partial w_n} = \frac{\partial J}{\partial y_n} \frac{\partial y_n}{\partial w_n}$$

$$\frac{\partial J}{\partial y_{n-1}} = \frac{\partial J}{\partial y_n} \frac{\partial y_n}{\partial y_{n-1}}$$

Reuse in subsequent operations

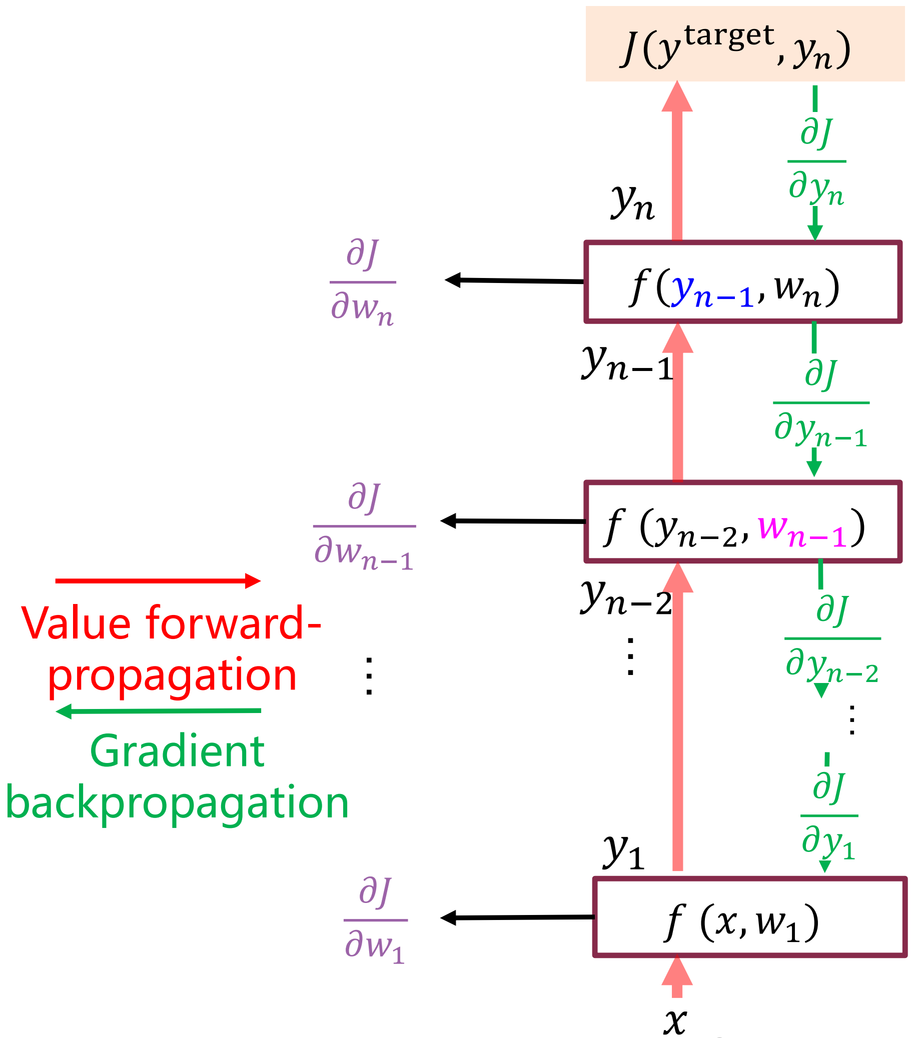
Backpropagation (BP)



$$\frac{\partial J}{\partial y_n} = \frac{\partial J(y^{\text{target}}, y_n)}{\partial y_n}$$

$$\frac{\partial J}{\partial w_n} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_n}$$

Backpropagation (BP)



$$\frac{\partial J}{\partial y_n} = \frac{\partial J(y^{\text{target}}, y_n)}{\partial y_n}$$

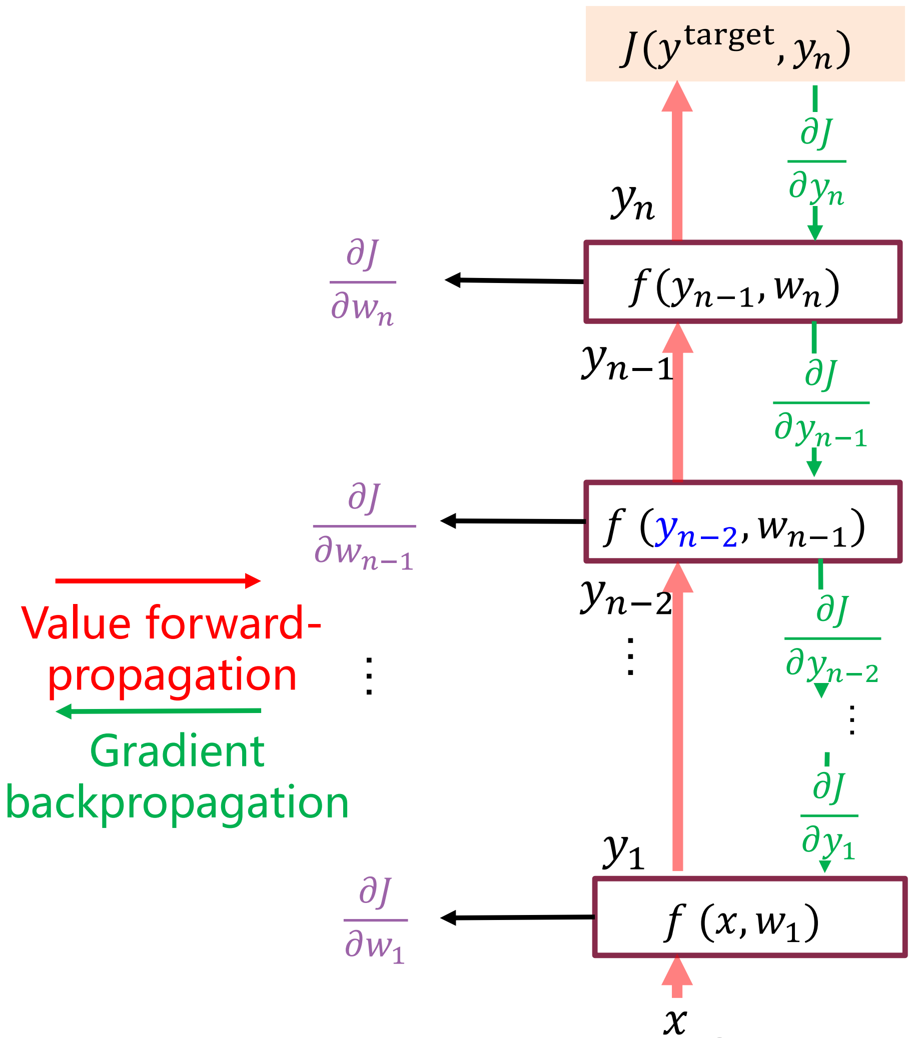
$$\frac{\partial J}{\partial w_n} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_n}$$

Only use in output layer

$$\frac{\partial J}{\partial y_{n-1}} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial y_{n-1}}$$

$$\frac{\partial J}{\partial w_{n-1}} = \frac{\partial J}{\partial y_{n-1}}\frac{\partial y_{n-1}}{\partial w_{n-1}}$$

Backpropagation (BP)



$$\frac{\partial J}{\partial y_n} = \frac{\partial J(y^{\text{target}}, y_n)}{\partial y_n}$$

$$\frac{\partial J}{\partial w_n} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_n}$$ ➡ Only use in output layer

$$\frac{\partial J}{\partial y_{n-1}} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial y_{n-1}}$$

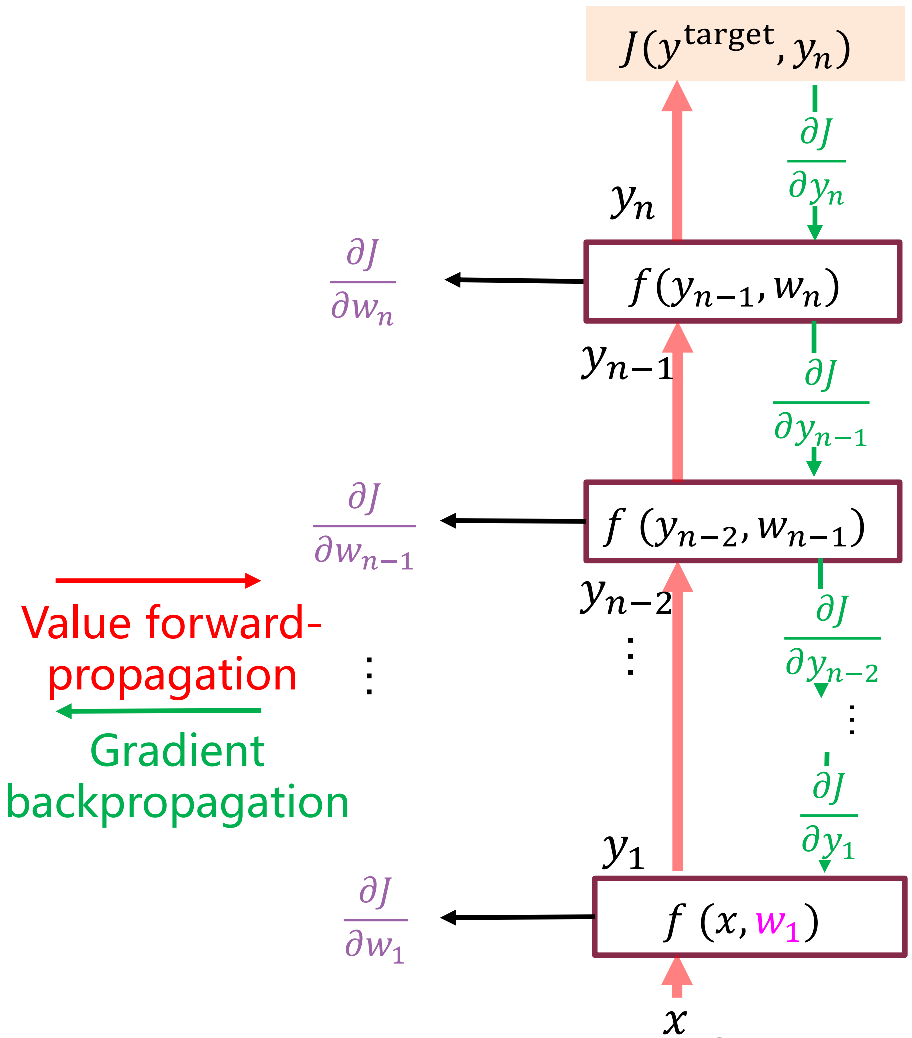
$$\frac{\partial J}{\partial w_{n-1}} = \frac{\partial J}{\partial y_{n-1}}\frac{\partial y_{n-1}}{\partial w_{n-1}}$$ ➡ Only use in Second last layer

$$\frac{\partial J}{\partial y_{n-2}} = \frac{\partial J}{\partial y_{n-1}}\frac{\partial y_{n-1}}{\partial y_{n-2}}$$

$$\frac{\partial J}{\partial w_{n-2}} = \frac{\partial J}{\partial y_{n-2}}\frac{\partial y_{n-2}}{\partial w_{n-2}}$$

Training of Neural Network

Backpropagation (BP)



$$\frac{\partial J}{\partial w_n} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_n}$$

$$\frac{\partial J}{\partial w_{n-1}} = \frac{\partial J}{\partial y_{n-1}}\frac{\partial y_{n-1}}{\partial w_{n-1}}$$

$$\cdots\ \cdots$$

$$\frac{\partial J}{\partial w_1}$$

Loss function

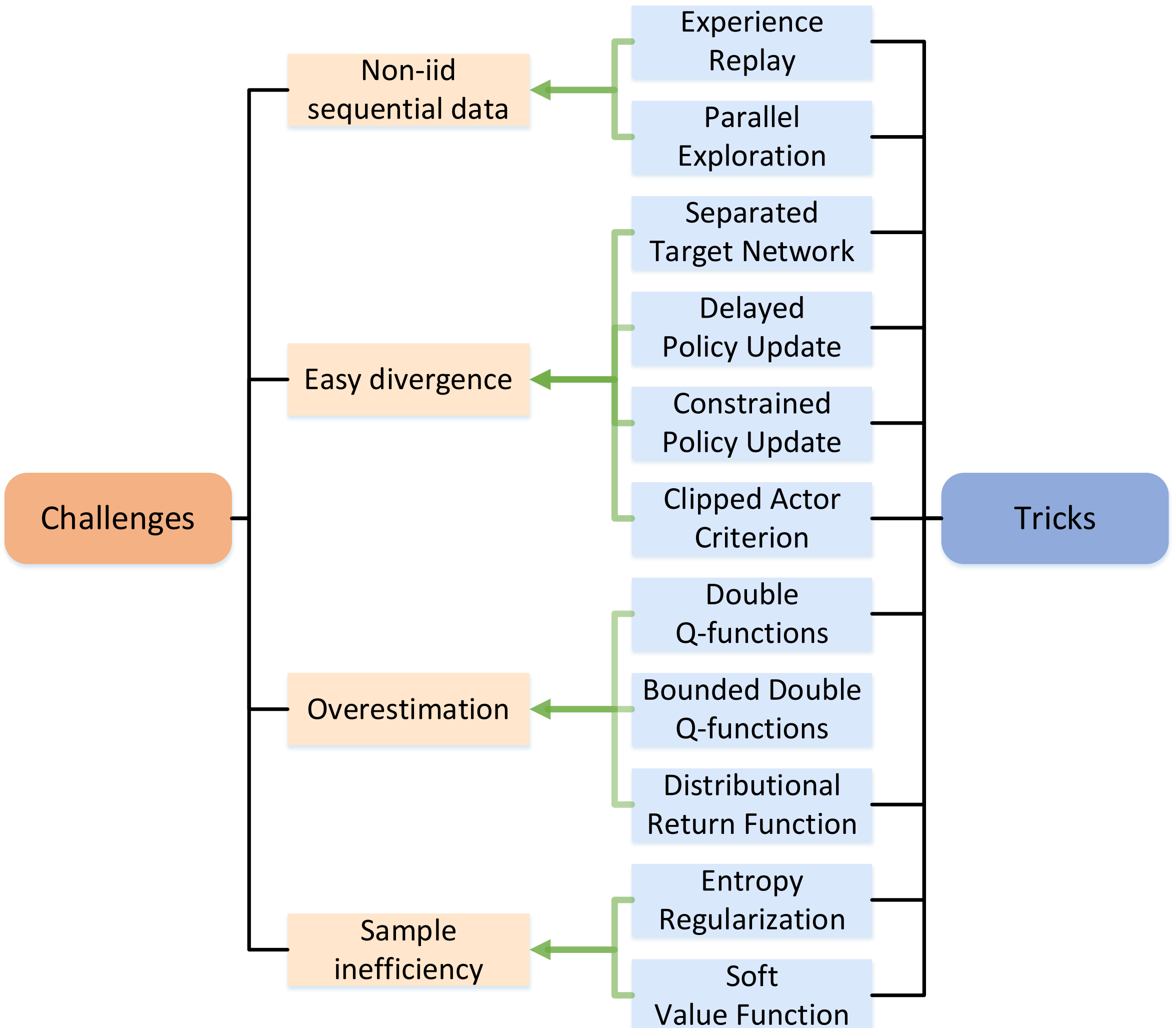
Weight of 1st layer

Outline

1. Motivation of Deep RL
2. Deep Neural Network
3. Challenges to be Solved
4. Typical DRL Algorithms

Challenges to be Solved

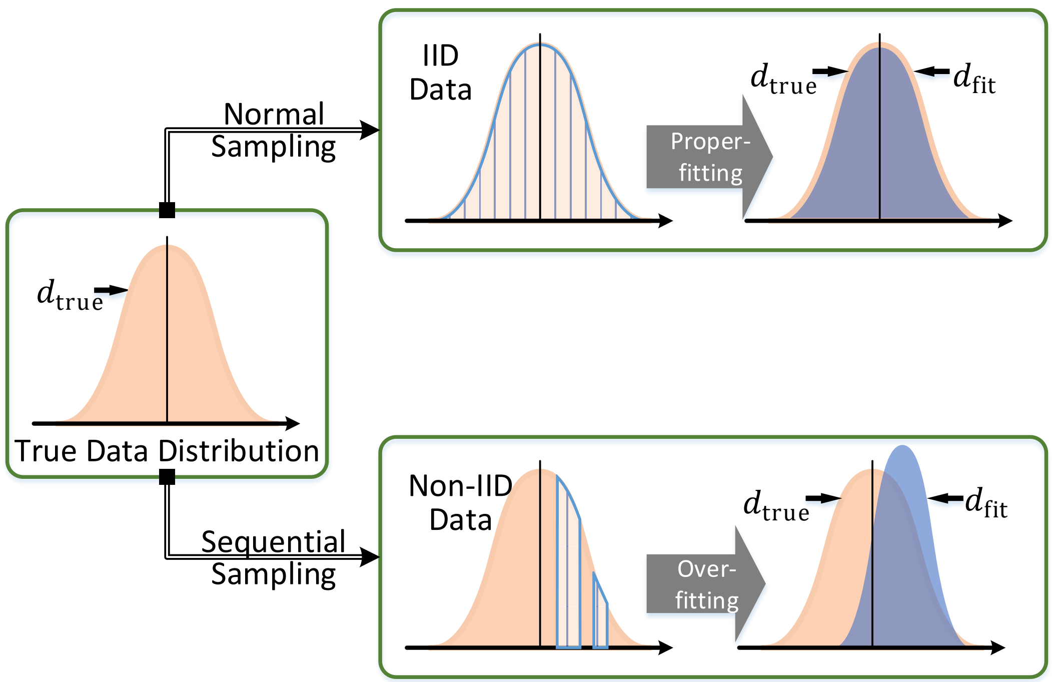
Challenges and tricks of Deep RL



Challenges to be Solved

(A) Challenge: non-iid sequential data

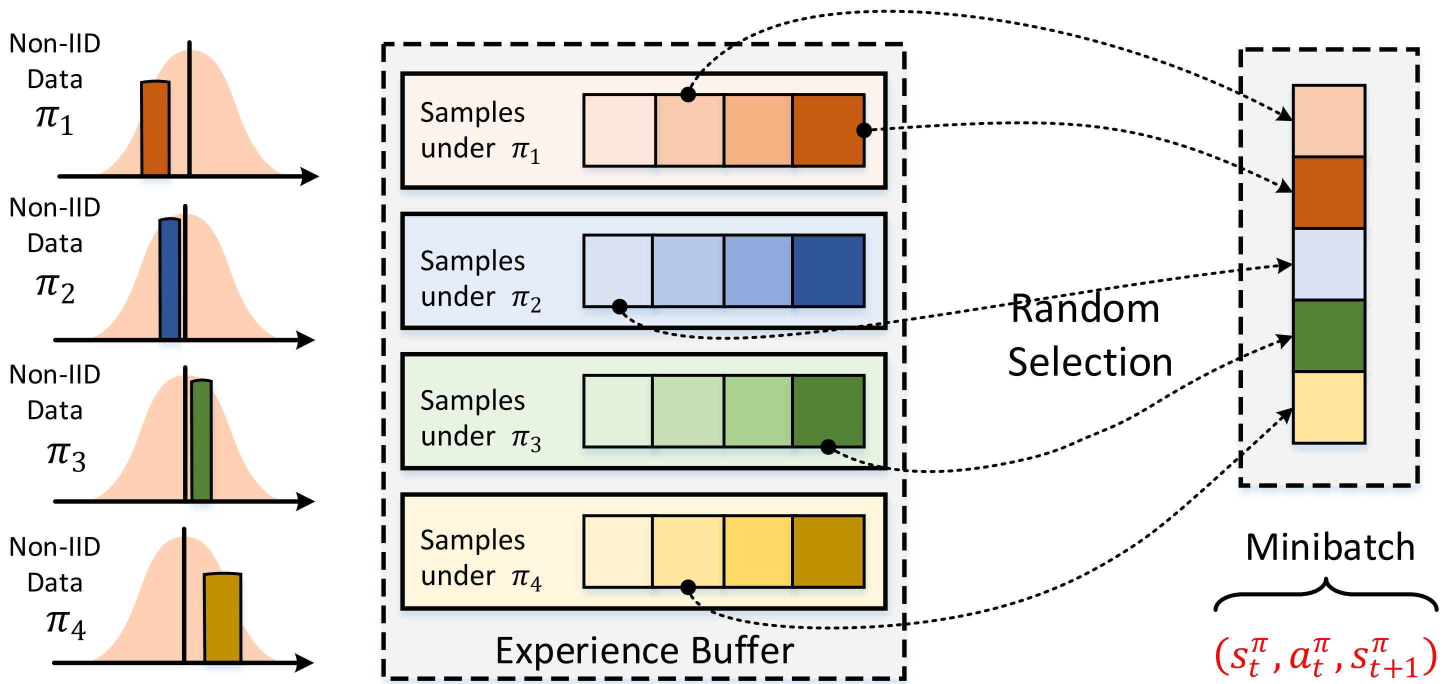
- Samples are needed to be independent and identically distributed (iid) in deep learning
- Samples are explored sequentially in reinforcement learning
- Overfitting: generalization ability is negatively affected
- Training distribution changes with diverse sequential data



Challenges to be Solved

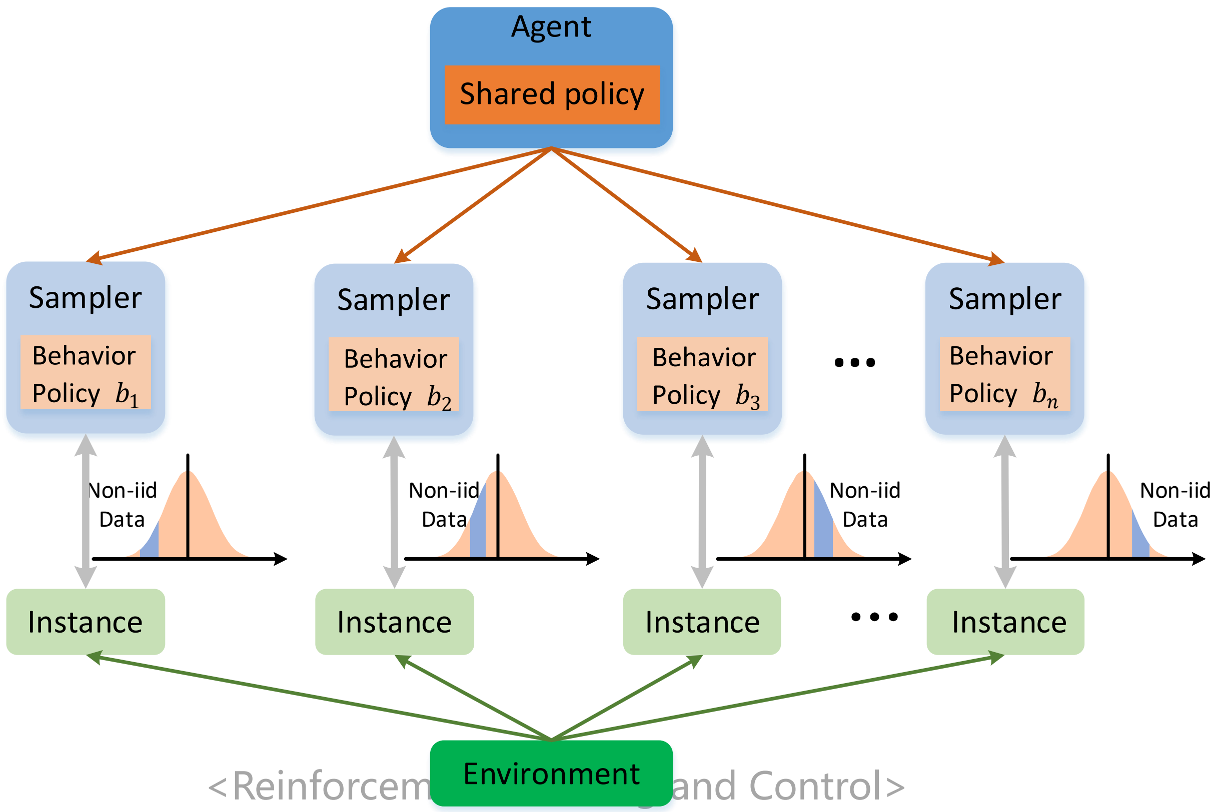
Trick 1: experience replay (ExR)

- Only suitable for off-policy RL
- Store samples in a replay buffer; reuse randomly sampled mini-batch to update
- Average training distribution over previous experiences



Challenges to be Solved

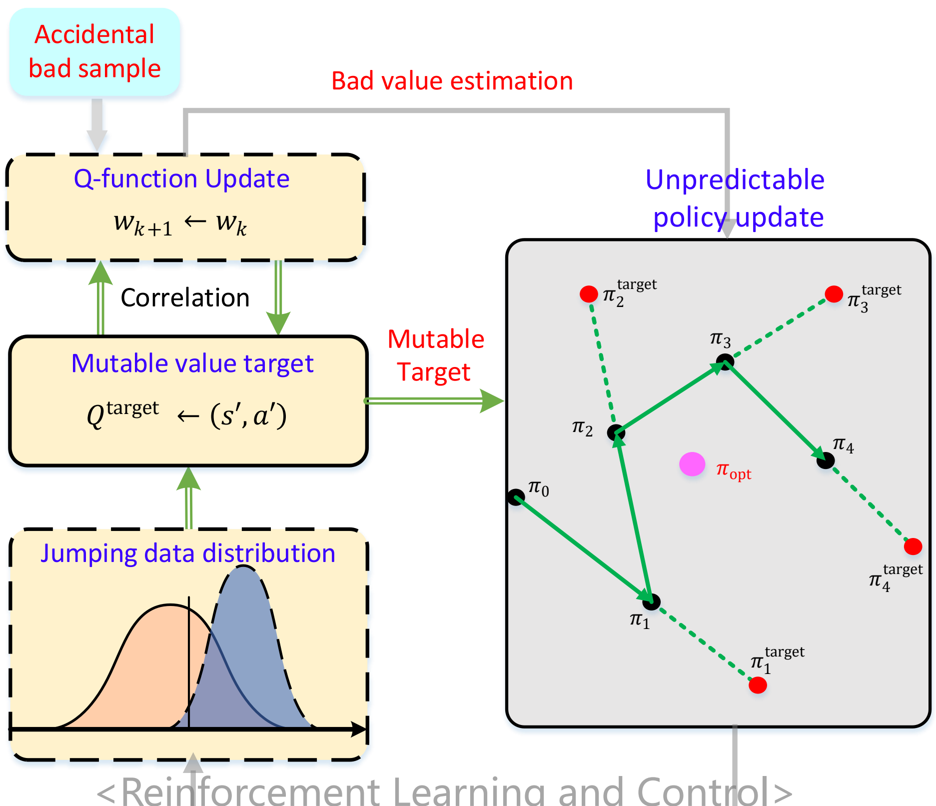
- **Trick 2: parallel exploration (PEx)**
  - Suitable for both on-policy RL & off-policy RL
  - Collect data from different parts of environment to alleviate the sample correlation and average the training distribution
  - Off-policy can maximize the diversity of training data



Challenges to be Solved

(B) Challenge: easy divergence

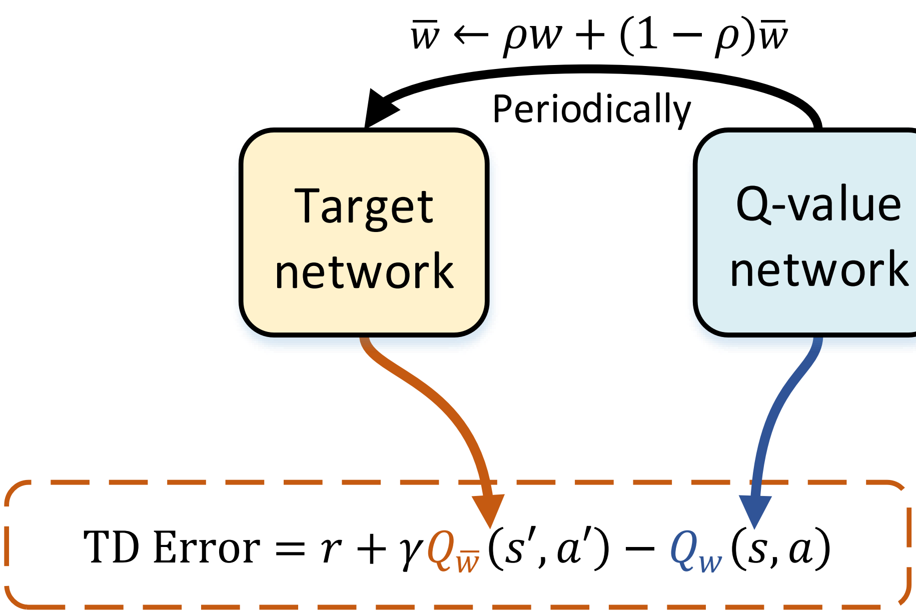
- Oscillating target values and unstable policy updates leads to oscillation of learning process and non-convergence policy
  - Bad value estimation provides wrong direction of policy update
  - Improper policy improvement deteriorates the critic quality



Challenges to be Solved

Trick 3: separated target network (STN)

- The target value is obtained from separated target network rather than from online value network
- The target is fixed during most updates of online value network
- Reduce tight correlation between Q-value and its target



Challenges to be Solved

- **Trick 4: delayed policy updates (DPU)**
  - Learn the policy w.r.t more precise value network
  - The policy update waits until value approximation error becomes small enough

- **Trick 5: constrained policy updates (CPU)**
  - Stabilize policy update by constraining the policy change

$$\|\pi_{k+1} - \pi_k\| \le \delta_\pi$$

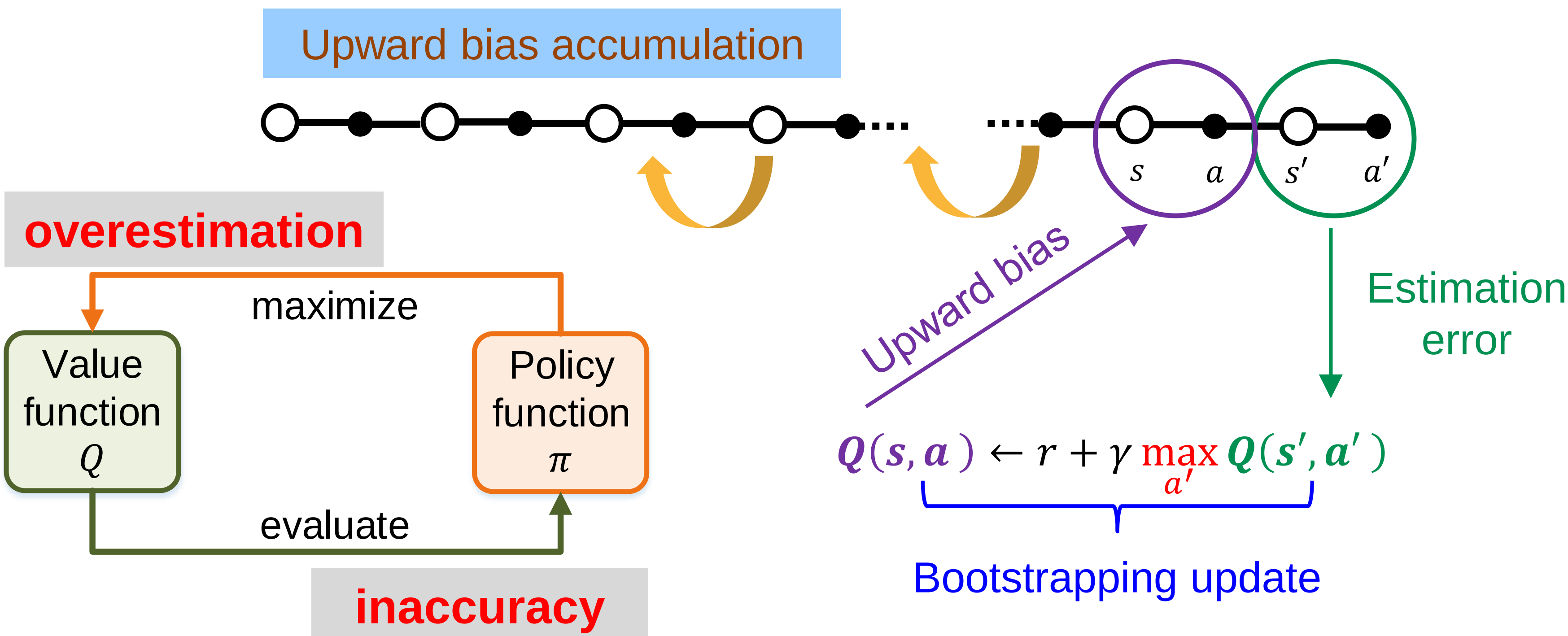
- **Trick 6: clipped actor criterion (CAC)**
  - Stabilize policy update by constraining actor criterion change

$$0 \le J_{\text{Actor}}(\pi_{k+1}) - J_{\text{Actor}}(\pi_k) \le \delta_J$$

Challenges to be Solved

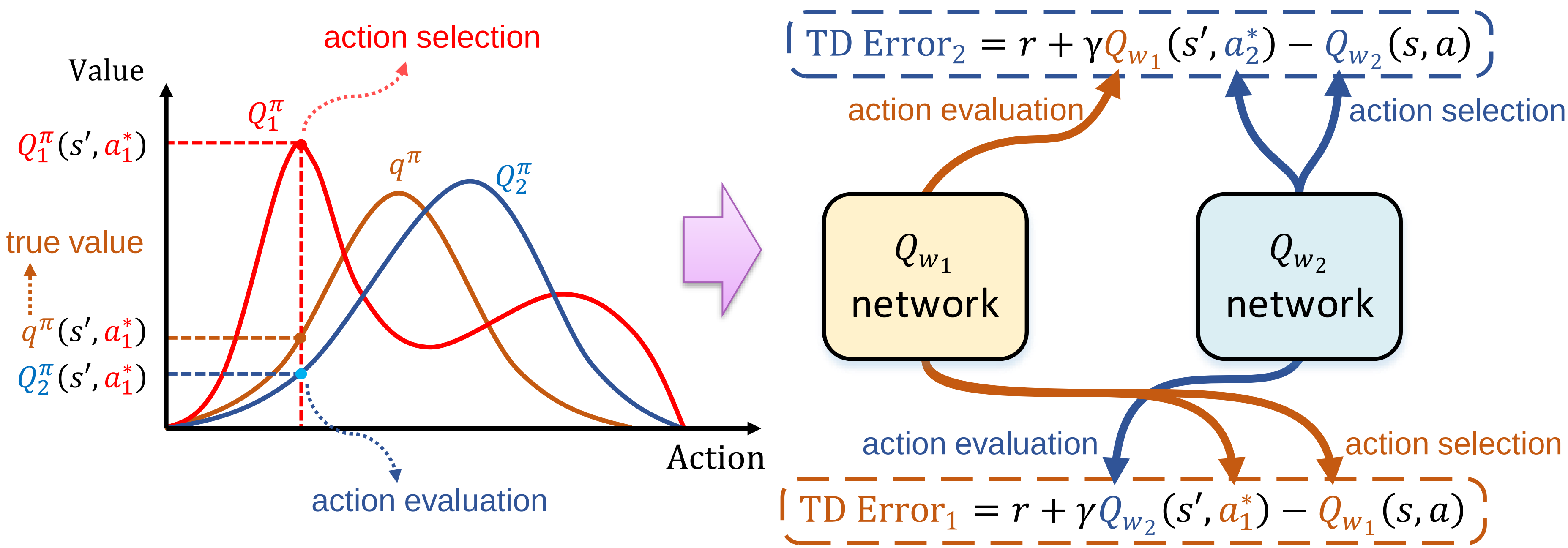
(C) Challenge: overestimation

- When maximizer acts on a value function, any estimation error will cause an upward bias
- Upward bias will accumulate through value network update, i.e., bootstrapping update



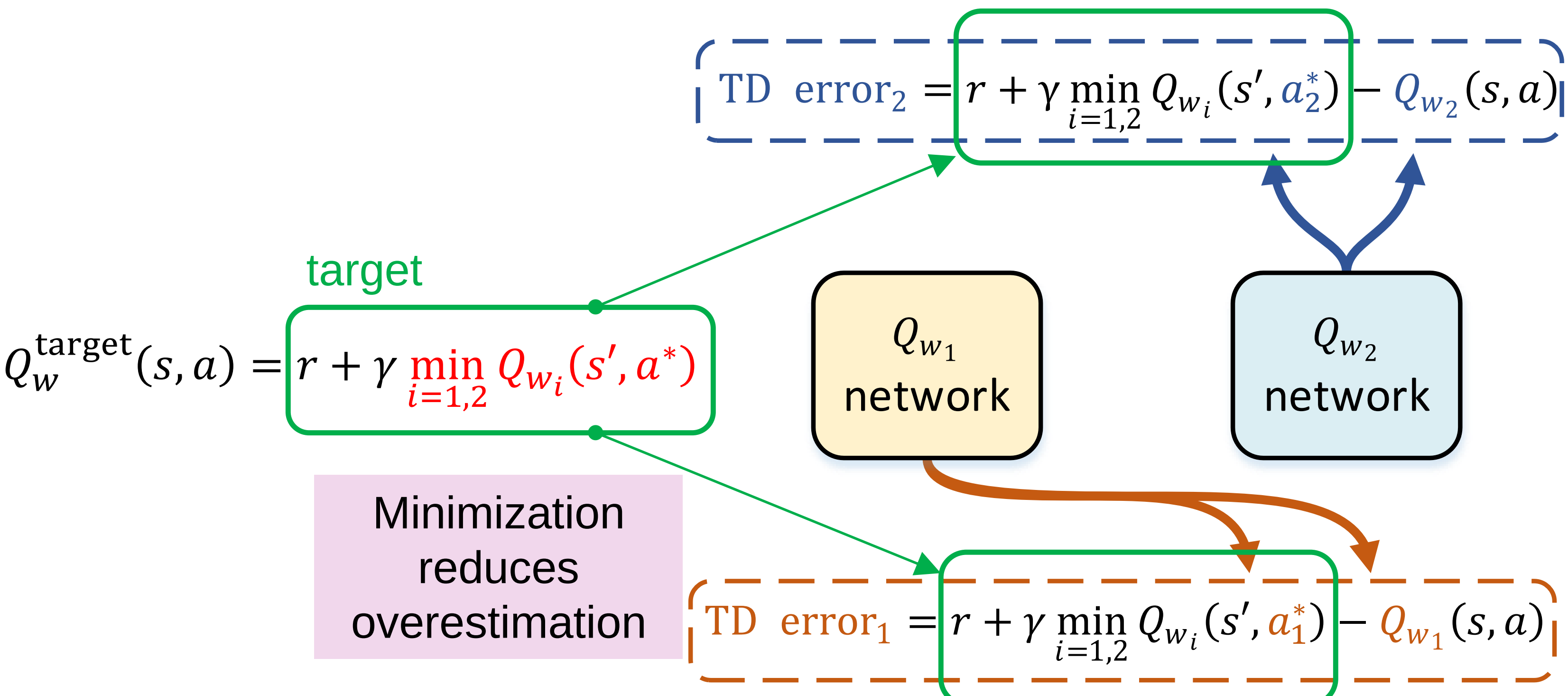
Trick 7: double Q-functions (DQF)

- Decouple maximizer into action selection and action evaluation
- Use two independent value functions: action is selected by one value function, and then evaluated by the other value function



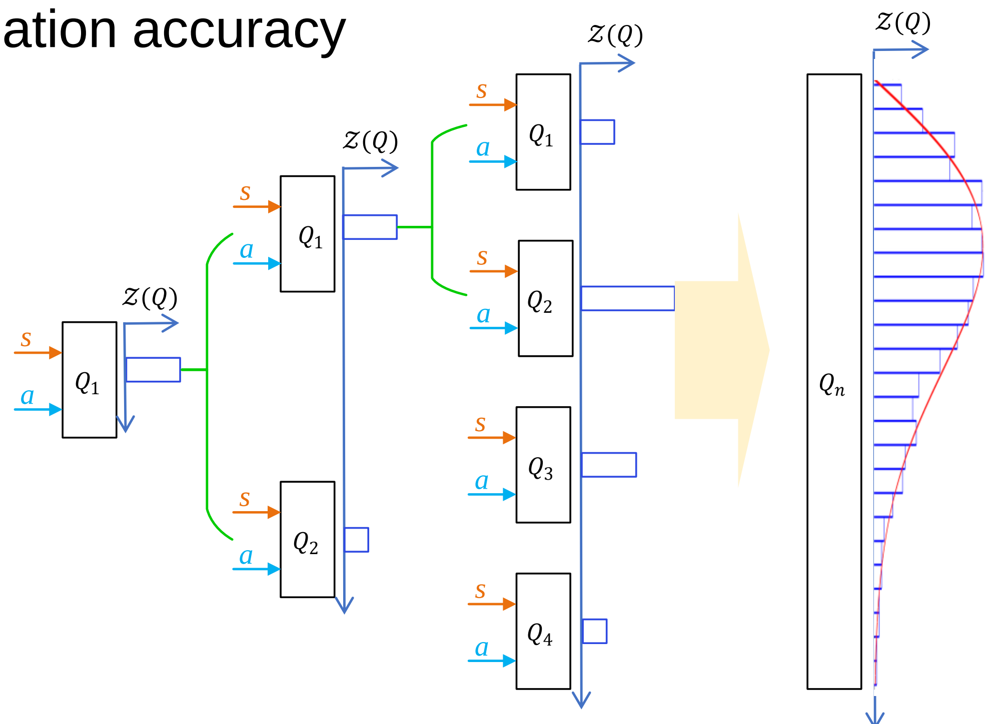
Trick 8: bounded double Q-functions (BDQ)

- Any two value functions are not exactly independent since each creates the target for the other value function
- Obtain target from the minimum of two value functions



Trick 9: distributional return function (DRF)

- More independent action-value functions can better alleviate overestimation
- Learn infinite number of action-value functions to maximize the estimation accuracy



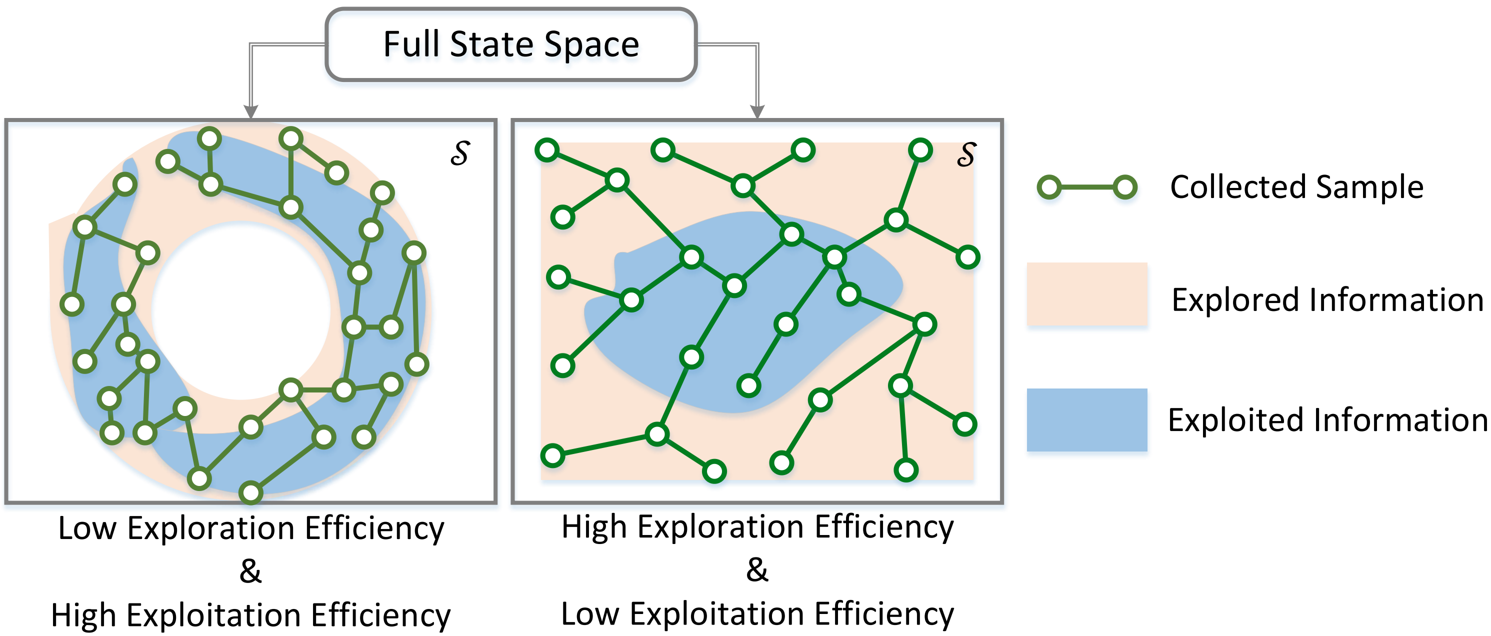
$$Z(s,a) \sim \mathcal{Z}_w(\cdot|s,a)$$

State-action return   Action-value return distribution

Challenges to be Solved

(D) Challenge: sample inefficiency

- High-dimensional and continuous state-action space in DRL
- Sample efficiency: how many samples are required when reaching a certain policy performance
  - Exploration efficiency & Exploitation efficiency



Challenges to be Solved

Trick 10: entropy regularization (EnR)

- Policy entropy: A measure of policy randomness
- Augment overall RL objective function with policy entropy

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left\{v^{\pi_\theta}(s) + \alpha\mathcal{H}\big(\pi_\theta(\cdot\,|s)\big)\right\}$$

Trick 11: soft value function (SVF)

- Augment each reward signal with policy entropy

$$r_{\text{aug}}(s,a,s') = r(s,a,s') + \alpha\mathcal{H}\big(\pi(\cdot\,|s)\big)$$

- Maximum entropy RL framework

$$v^{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{i=0}^{\infty}\gamma^i\Big(r_{t+i} + \alpha\mathcal{H}\big(\pi(\cdot\,|s_{t+i})\big)\Big)\middle|\, s_t = s\right\}$$

Outline

1 Motivation of Deep RL

2 Deep Neural Network

3 Challenges to be Solved

4 Typical DRL Algorithms

Typical DRL Algorithms

Deep RL Algorithms

Algorithm	ExR	PEx	STN	DPU	CPU	CAC	DQF	BDQ	DRF	EnR	SVF	π
DQN	★		★									off
Dueling DQN	●		●									off
DDQN	●		●				★					off
TRPO					★							on
PPO						★						on
A3C		★								★		on
DDPG	●		●				●					off
TD3	●		●	●			●	★				off
SAC	●		●				●	●			★	off
DSAC	●	●	●	●			●		★		●	off

★: Formally proposed for the first time

●: Inherited tricks from previous DRLs

Typical DRL Algorithms

Deep Q-Network (DQN)

- Suitable for continuous state, discrete action
- 2 Q-networks (only 1 needs BP)
- **Trick 1: experience replay**
- **Trick 3: separated target network**

$$J(w) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{Replay}}} \left\{ \left(r + \gamma \max_{a'} Q_{\bar{w}}(s', a') - Q_w(s,a) \right)^2 \right\}$$

Q-network weight (w) · Replay buffer ($\mathcal{D}_{\text{Replay}}$) · Target network weight ($\bar{w}$)

Atari Games	w/o Trick	Sep. Target Network	Exp. Replay	Sep. Target Network + Exp. Replay
Breakout	3	10	241	317
Enduro	29	142	831	1006
River Raid	1453	2868	4103	7447
Seaquest	276	1003	823	2894
Space Invaders	302	373	826	1089

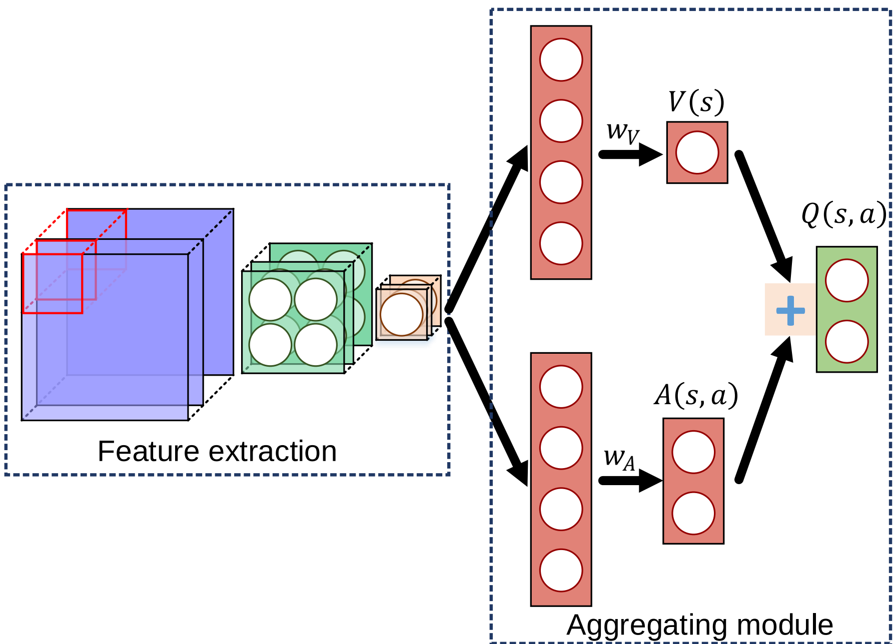
* Game score

* Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature, 2015

Typical DRL Algorithms

Dueling DQN

- 2 Q-networks (only 1 needs BP)
- **Trick 1: experience replay**
- **Trick 3: separated target network**
- **Dueling Q-network:** Split Q-value into V-value and A-value



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

* Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning. ICML 2016, New York, USA.

Typical DRL Algorithms

Dueling DQN

- Unidentifiable estimation

$$Q(s,a;w_V,w_A) = V(s;w_V) + A(s,a;w_A)$$

  - Given Q, we cannot recover V and A uniquely
  - $V' = V + 10$, $A' = A - 10$, $Q' = Q$
  - Results in oscillating training process

$$\mathbb{E}_{a\sim\pi}\{A^{\pi}(s,a)\} = 0$$

when π is greedy

- Identifiable estimation

$$Q(s,a;w_V,w_A) = V(s;w_V) + A(s,a;w_A) - \max_{\hat{a}} A(s,\hat{a};w_A)$$

  - Identifiable condition: $Q(s,a^*;w_V,w_A) = V(s;w_V)$

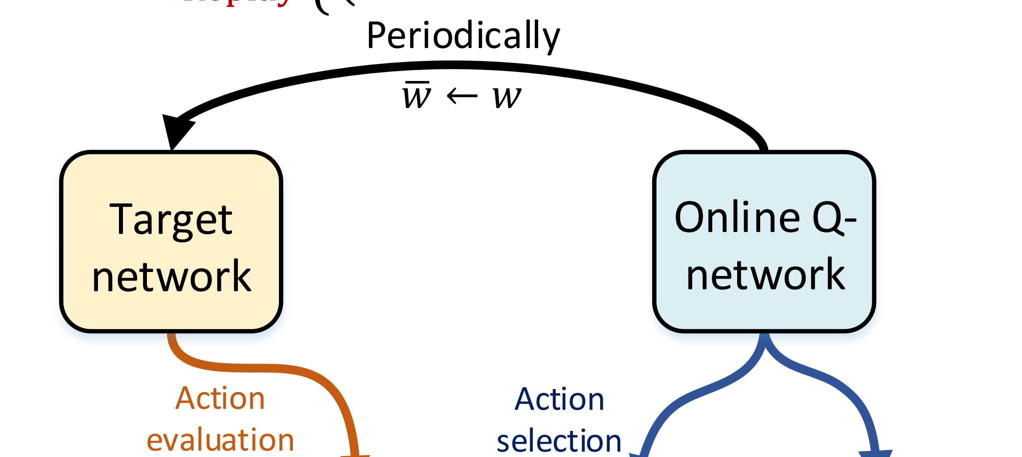
- Pros: (1) Better value estimation; (2) More robust performance

Typical DRL Algorithms

Double DQN (DDQN)

- 2 Q-networks (only 1 needs BP)
- **Trick 1: experience replay**
- **Trick 3: separated target network**
- **Trick 7: double Q-functions**
  - The target network in DQN provides a natural candidate for action evaluation

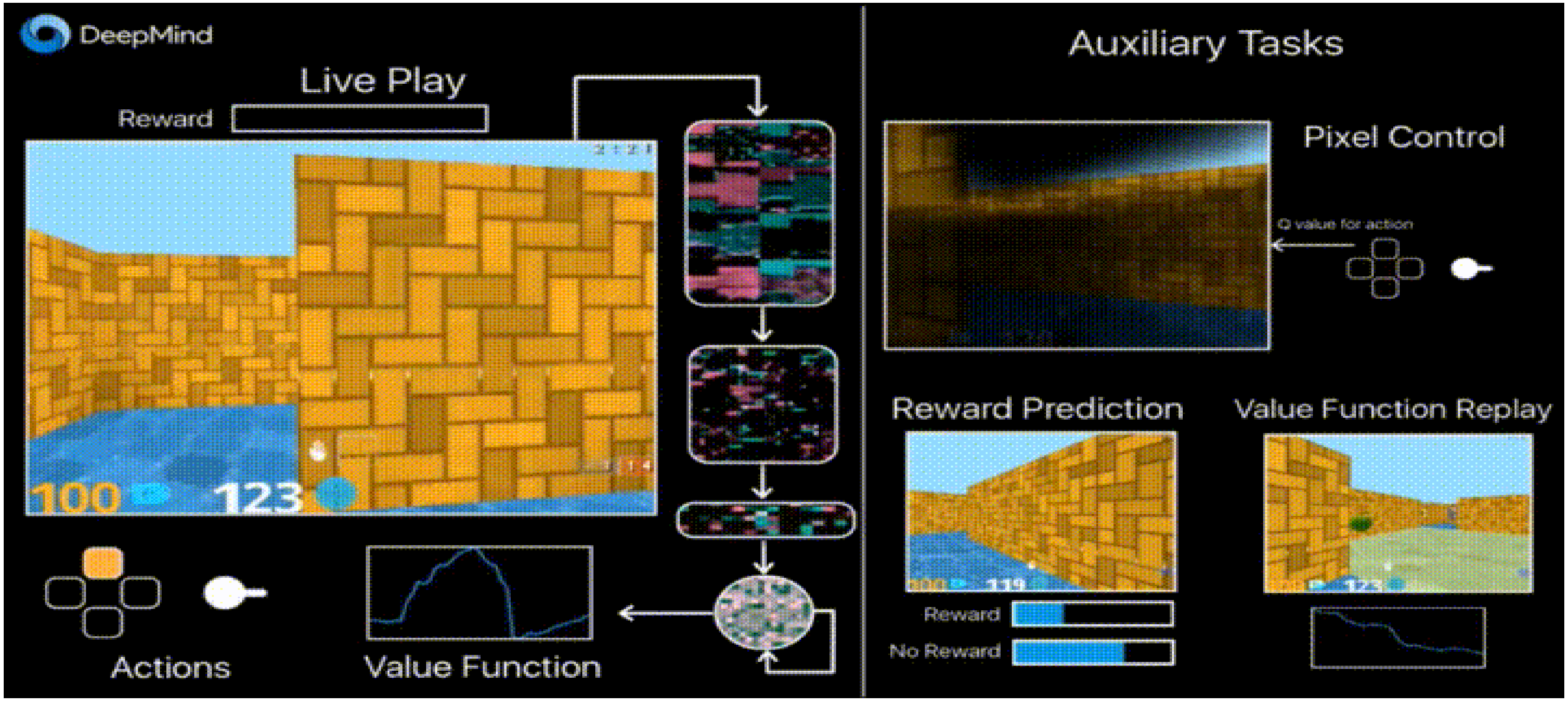
$$J(w) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\mathbf{Replay}}} \left\{ \left(r + \gamma Q_{\overline{w}}(s', a^*(s')) - Q_w(s,a) \right)^2 \right\}$$



$$\text{TD Error} = r + \gamma Q_{\overline{w}}\left(s', \underset{a'}{\operatorname{argmax}}\, Q_w(s', a')\right) - Q_w(s,a)$$

Deep RL examples

- DeepMind Demo of Double DQN



Typical DRL Algorithms

Trust Region Policy Optimization (TRPO)

- 1 V-network, 1 stochastic policy network (both need BP)
- Minorize-maximization (MM) optimization
- **Trick 5: constrained policy updates**
  - Convert the penalty term into a trust region constraint

$$\theta \leftarrow \arg\max_{\theta} \mathbb{E}_{s\sim d_{\pi_{\text{old}}},a\sim\pi_{\text{old}}}\left\{\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}A^{\pi_{\text{old}}}(s,a)\right\}$$

<u>Subj. to:</u>

$$D_{\text{KL}}\left(\pi_{\text{old}},\pi_\theta\right) \le \delta_\pi$$

  - Use average KL divergence to replace max KL divergence

$$\bar{D}_{\text{KL}} = \mathbb{E}_{s\sim d_{\pi_{\text{old}}}}\{D_{\text{KL}}(\pi_{\text{old}},\pi_\theta)\} \le \delta_\pi$$

$$(\theta-\theta_{\text{old}})^{\text{T}}H(\theta-\theta_{\text{old}}) \le \delta_\pi$$

$$H = \mathbb{E}\left\{\frac{\partial^2 \bar{D}_{\text{KL}}}{\partial\theta^2}\right\}$$

Computationally expensive!

* Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization. ICML 2015, Lille, France.

Typical DRL Algorithms

Proximal Policy Optimization (PPO)

- 1 V-network, 1 stochastic policy network (both need BP)
- Pure first-order optimization that holds the monotonic improvement property
- **Trick 6: clipped actor criterion**

$$\theta \leftarrow \arg\max_{\theta} \mathbb{E}_{s\sim d_{\pi_{\text{old}}}, a\sim\pi_{\text{old}}} \left\{ \min\left(\rho_{t:t} A^{\pi_{\text{old}}}(s,a), \rho_{\text{clip}} A^{\pi_{\text{old}}}(s,a) \right) \right\}$$

$$\rho_{\text{clip}} \stackrel{\text{def}}{=} \text{clip}(\rho_{t:t}, 1-\epsilon, 1+\epsilon)$$

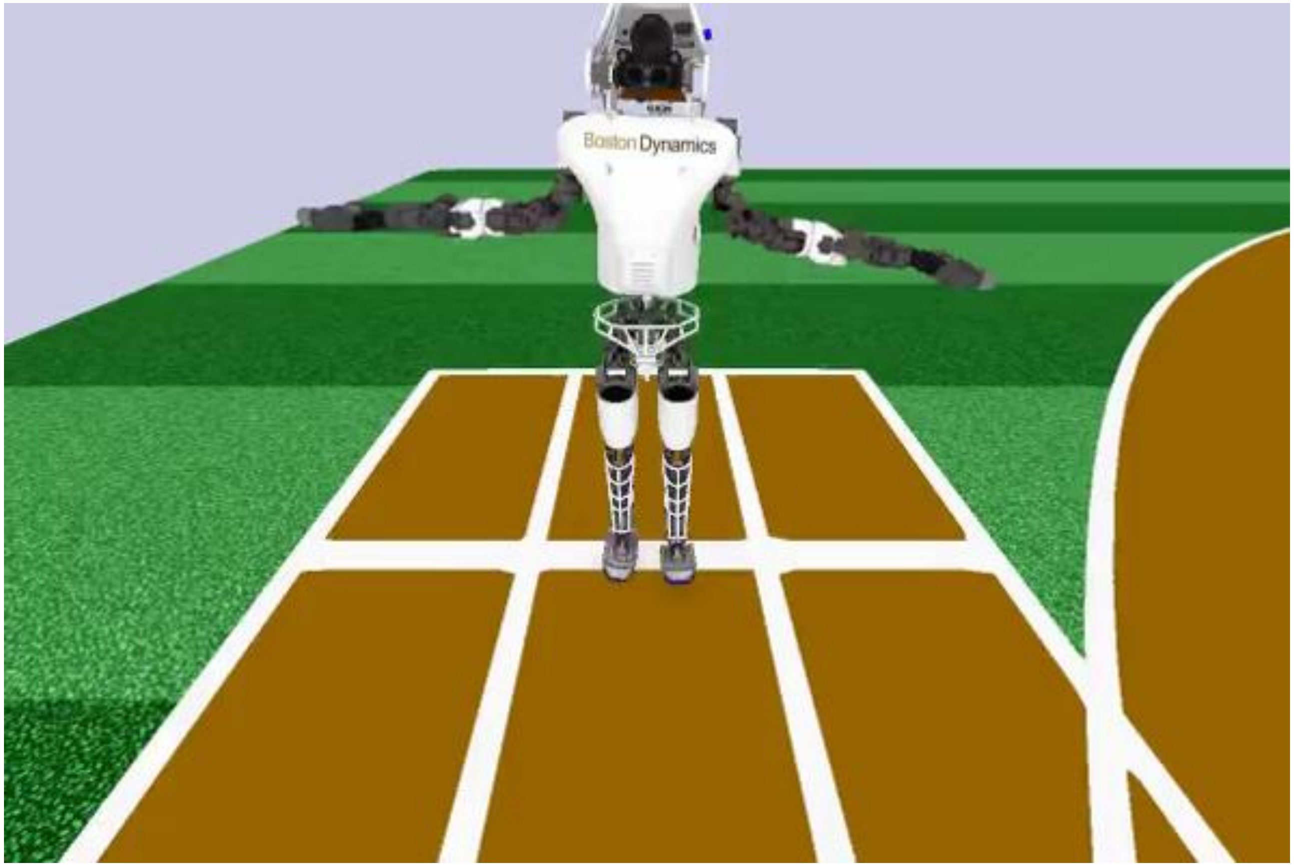
Advantage function $\quad A^{\pi}(s,a) = r + \gamma V^{\pi}(s') - V^{\pi}(s)$

- The minimum of clipped and unclipped criteria is a lower bound of original objective function (i.e., surrogate function)

* Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. arXiv preprint, 2017.

PPO: Atlas model from Boston Dynamics

- Include 30 distinct joints (versus 17 in bipedal robot)



*https://openai.com/blog/openai-baselines-ppo/

Typical DRL Algorithms

Deep Deterministic Policy Gradient (DDPG)

- Suitable for continuous state & action space
- 2 Q-networks (1 BP), 2 policy networks (1 BP)
- **Trick 1: experience replay**
- **Trick 3: separated target network**
- **Trick 7: double Q-functions**
  - Action evaluation and action selection are implemented by different networks

$$J_{\text{Critic}}(w) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{Replay}}} \left\{ \left(r + \gamma Q_{\bar{w}} \left(s', \pi_{\bar{\theta}}(s') \right) - Q_w(s,a) \right)^2 \right\}$$

- Learn a deterministic policy $\pi_\theta(s)$ which maximizes $Q_w(s,a)$

$$J_{\text{Actor}}(\theta) = \mathbb{E}_{s \sim \mathcal{D}_{\text{Replay}}} \left\{ Q_w \left(s, \pi_\theta(s) \right) \right\}$$

Typical DRL Algorithms

Twin Delayed DDPG (TD3)

- 4 Q-networks (2 BP), 2 policy networks (1 BP)
- **Trick 1: experience replay**
- **Trick 3: separated target network**
- **Trick 7: double Q-functions**
- **Trick 8: bounded double Q-functions**

$$Q_{\bar{w}}^{\text{target}}(s,a) = r + \gamma \min_{i=1,2} Q_{\bar{w}_i}\left(s', \pi_{\bar{\theta}}(s')\right)$$

$$J_{\text{Critic}}(w_1) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{Replay}}}\left\{\left(Q_{\bar{w}}^{\text{target}}(s,a) - Q_{w_1}(s,a)\right)^2\right\}$$

$$J_{\text{Critic}}(w_2) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{Replay}}}\left\{\left(Q_{\bar{w}}^{\text{target}}(s,a) - Q_{w_2}(s,a)\right)^2\right\}$$

- **Trick 4: delayed policy updates**. TD3 delays the policy update, i.e., the weights of policy network change less frequently

* Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods. ICML 2018, Stockholm, Sweden.

Typical DRL Algorithms

Soft Actor-Critic (SAC)

- 4 soft Q-networks (2 BP), 1 stochastic policy networks (1 BP)
- **Trick 1: experience replay**
- **Trick 3: separated target network**
  - SAC concurrently learns a stochastic policy and two Q-networks. Both of two Q-networks have related target networks, while the policy has no target
- **Trick 7: double Q-functions**
- **Trick 8: bounded double Q-functions**
- **Trick 11: soft value function**
  - SAC uses maximum entropy RL framework, where each reward signal is augmented with a policy entropy term
  - SAC trains a stochastic policy to strike a balance between expected return and accumulated policy entropy

Typical DRL Algorithms

Soft Actor-Critic (SAC) – Soft Q-function

- Use common target to update two Q-networks (like TD3)

$$J_{\text{Critic}}(w_i) = \mathbb{E}_{s,a,s'\sim\mathcal{D}_{\text{replay}}}\left\{\left(Q_{\bar{w}}^{\text{target}}(s,a) - Q_{w_i}(s,a)\right)^2\right\}, \forall i = 1,2$$

- Common target is calculated through soft Q-function

$$Q_{\bar{w}}^{\text{target}}(s,a) = r + \gamma\mathbb{E}_{a'\sim\pi_\theta}\left\{\min_{i=1,2} Q_{\bar{w}_i}(s',a') - \alpha \log \pi_\theta(a'|s')\right\}$$

- Either high return or high policy entropy would lead to high soft Q-function, thus the corresponding policy has the motivation to explore the state space with larger uncertainties

Typical DRL Algorithms

Soft Actor-Critic (SAC) – Stochastic policy

- Maximize soft V-function

$$J_{\text{Actor}}(\theta) = \mathbb{E}_{s\sim\mathcal{D}_{\text{replay}},a\sim\pi_\theta}\{Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)\}$$

temperature parameter

- Reparameterization trick removes the pain point of the action distribution depending on policy weights and leads to a lower variance estimate

- SAC automatically update the temperature parameter, where the policy entropy is enforced to be no less than a lower bound

* Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. ICML 2018, Stockholm, Sweden.

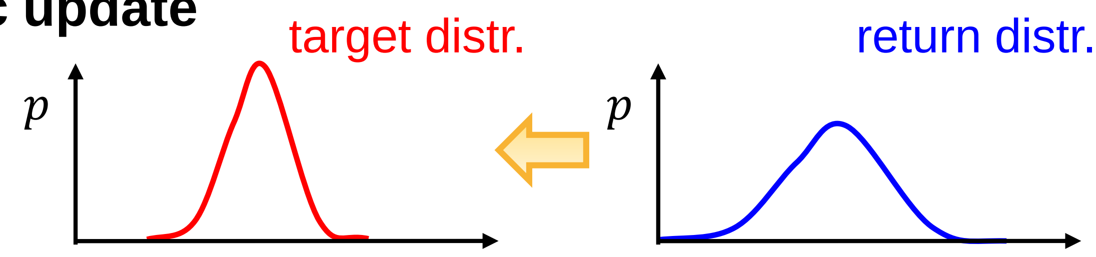
Typical DRL Algorithms

Distributional SAC (DSAC)

- 2 distributional Q-networks (1 BP), 2 stochastic policy networks (1 BP)
- DSAC integrates distributional return function into maximum entropy RL framework
- **Trick 1: experience replay**
- **Trick 2: parallel exploration**
- **Trick 3: separated target network**
- **Trick 4: delayed policy updates**
- **Trick 7: double Q-functions**
- **Trick 9: distributional return function**
  - Instead of learning two independent Q-networks, DSAC learns a single distributional return function for more accurate action-value estimation
- **Trick 11: soft value function**

DSAC - Distributional policy iteration

- **Critic update**



$$J_{\text{Critic}}(w) = \mathbb{E}_{s,a\sim\mathcal{D}_{\text{replay}}}\left\{D_{\text{KL}}\left(\mathcal{Z}_{\bar{w}}^{\text{target}}(\cdot\,|s,a),\mathcal{Z}_{w}(\cdot\,|s,a)\right)\right\}$$

Contraction mapping theorem

- **Actor update**

$$J_{\text{Actor}}(\theta) = \mathbb{E}_{s\sim\mathcal{D}_{\text{replay}},a\sim\pi_\theta}\left\{\text{Perc}\left(\mathcal{Z}_{w}(\cdot\,|s,a)\right) - \alpha\log\pi_\theta(a|s)\right\}$$

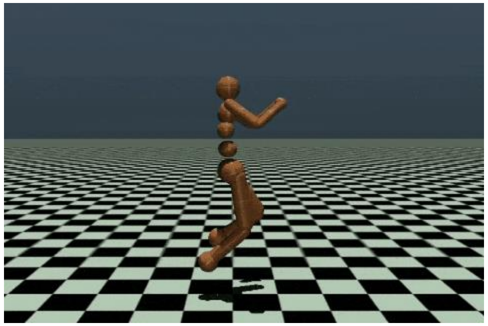
Policy performance improves monotonically

* Duan J, Guan Y, Li S E*, et al. Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors. IEEE Trans Neural Networks and Learning Systems, 2021 (First version in Arxiv, Jan 2020)

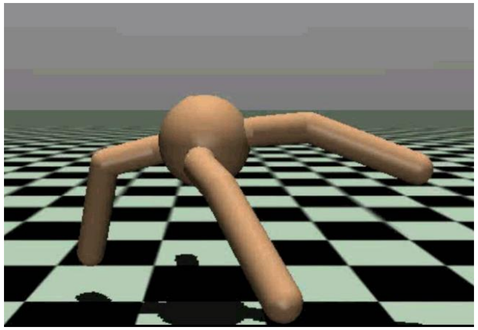
Deep RL examples

Mujoco Tasks

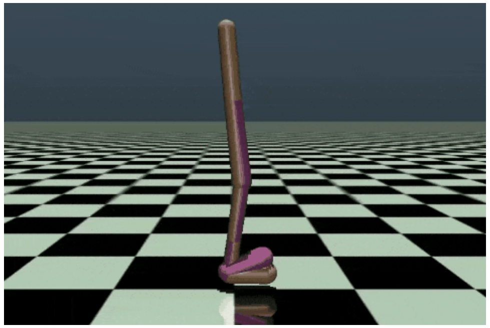
- DSAC (Tsinghua iDLab, 2021)



More like human posture

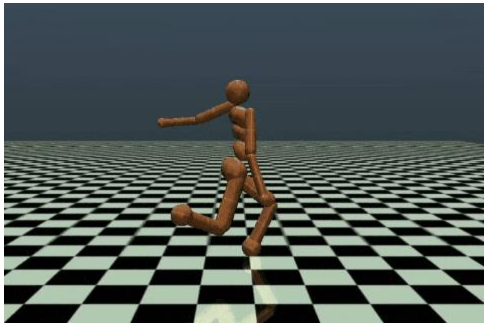


Faster crawl with 3 legs

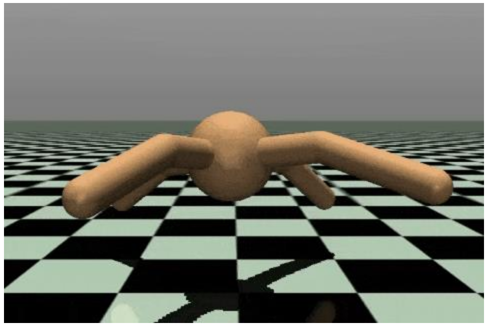


More reasonable gravity center

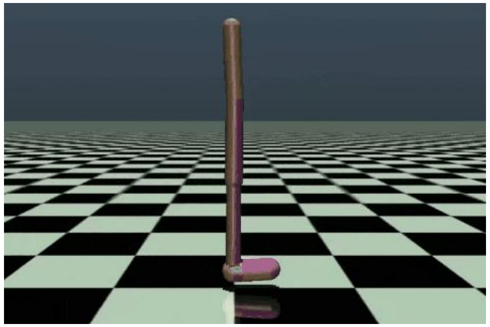
- SAC (UC Berkeley, 2019)



Lean back & Small stride



Slower crawl with 4 legs



Gravity center moving forward

<Reinforcement Learning and Control>