Ranking Policy Gradient

Kaixiang Lin
July 14th

Key ideas towards sample-efficiency

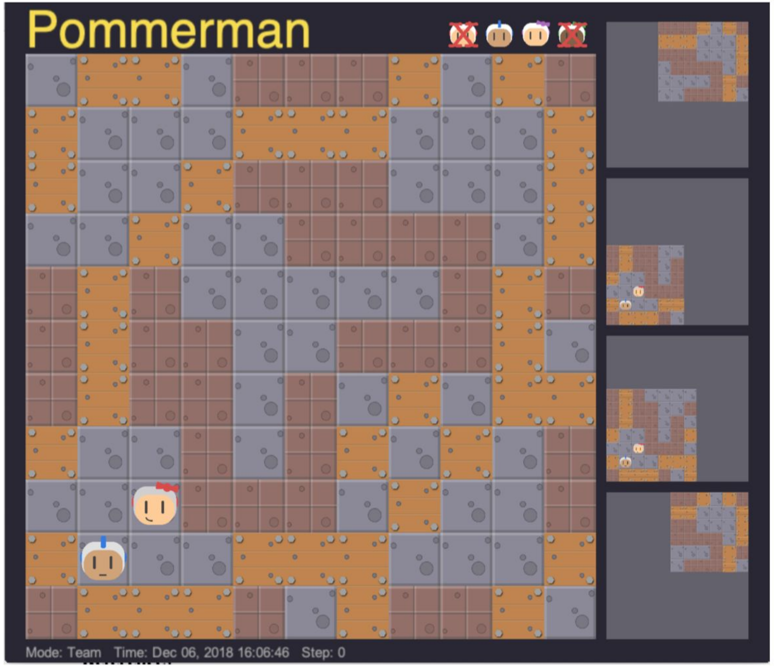
- Learning optimal rank of actions
- Disentangle exploration and exploitation
- Off-policy learning as supervised learning

Motivating Example

- **Pommerman**
  - Goal: kill the opponent.
  - State: board
  - Action: move around or lay a bomb.
  - Reward: -2 for each step,
    100 for killing opponent.
  - Horizon: finite.

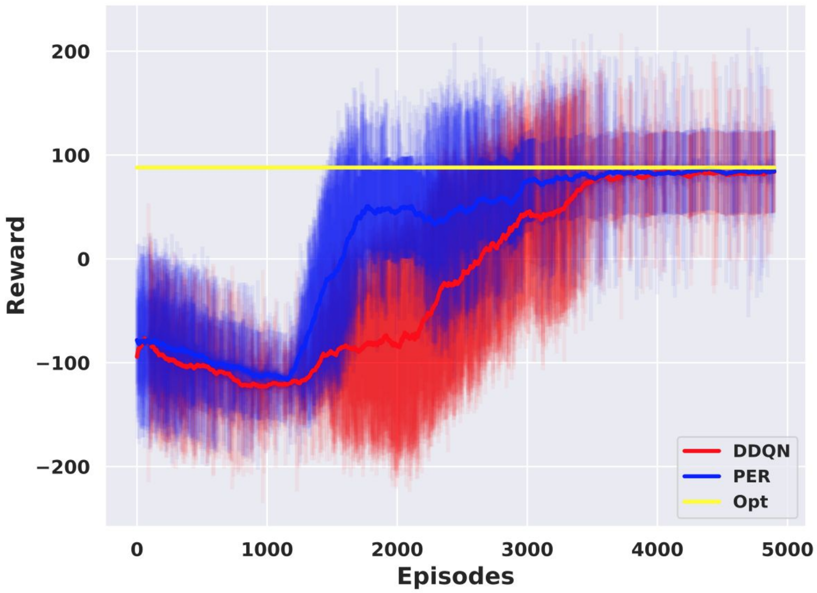
Motivating Example

- **Pommerman**
  - Goal: kill the opponent.
  - State: board
  - Action: move around or lay a bomb.
  - Reward: -2 for each step,
    100 for killing opponent.
  - Horizon: finite.



Motivating Example

- DDQN: Double DQN with regular replay buffer.
- PER: Double DQN with prioritized experience replay.
- Opt: best possible performance.



Revisit DQN

Minimizing the Bellman error:

$$\min_{\theta} \mathbf{E}_{(s,a,s')\sim\mathcal{D}}[Q_\theta(s,a) - (r(s,a) + \gamma \max_{a'} Q_\phi(s',a'))]^2$$

Revisit DQN

Minimizing the Bellman error:

$$\min_{\theta} \mathbf{E}_{(s,a,s')\sim\mathcal{D}}[Q_\theta(s,a) - (r(s,a) + \gamma \max_{a'} Q_\phi(s',a'))]^2$$

Choose the action greedily:

$$\pi(s) = \arg\max_a Q(s,a)$$

Revisit DQN

Minimizing the Bellman error:

Moving target

$$\min_{\theta} \mathbf{E}_{(s,a,s')\sim\mathcal{D}}[Q_\theta(s,a) - (r(s,a) + \gamma \max_{a'} Q_\phi(s',a'))]^2$$

non-stationary

Choose the action greedily:

$$\pi(s) = \arg\max_a Q(s,a)$$

Unstable optimization:

- Deadilly triad: function approximations, experience replay, and bootstrapping. [Sutton, 2018]
- Moving target, non-stationary distribution.
- Hard to quickly adapt to good experience.

Revisit DQN

Minimizing the Bellman error:

$$\min_{\theta} \mathbf{E}_{(s,a,s')\sim\mathcal{D}}[Q_\theta(s,a) - (r(s,a) + \gamma \max_{a'} Q_\phi(s',a'))]^2$$

Choose the action greedily:

$$\pi(s) = \arg\max_a Q(s,a)$$

Do we need/have the accurate estimation of optimal Q(s,a)?:

Revisit DQN

Minimizing the Bellman error:

$$\min_{\theta} \mathbf{E}_{(s,a,s')\sim\mathcal{D}}[Q_\theta(s,a) - (r(s,a) + \gamma \max_{a'} Q_\phi(s',a'))]^2$$

Choose the action greedily:

$$\pi(s) = \arg\max_a Q(s,a)$$

Do we need/have the accurate estimation of optimal Q(s,a)?:

- **No. We do the argmax over Q(s, a).**

Revisit DQN

Minimizing the Bellman error:

$$\min_{\theta} \mathbf{E}_{(s,a,s')\sim\mathcal{D}}[Q_\theta(s,a) - (r(s,a) + \gamma \max_{a'} Q_\phi(s',a'))]^2$$

Choose the action greedily:

$$\pi(s) = \arg\max_a Q(s,a)$$

Do we need/have the accurate estimation of optimal Q(s,a)?:

- No. We do the argmax over Q(s, a).
- In practice we rarely see an accurate estimation of Q-values in DQN.

Revisit DQN

Minimizing the Bellman error:

$$\min_{\theta} \mathbf{E}_{(s,a,s')\sim\mathcal{D}}[Q_\theta(s,a) - (r(s,a) + \gamma \max_{a'} Q_\phi(s',a'))]^2$$

Choose the action greedily:

$$\pi(s) = \arg\max_a Q(s,a)$$

Do we need/have the accurate estimation of optimal Q(s,a)?:

- No. We do the argmax over Q(s, a).
- In practice we rarely see an accurate estimation of Q-values in DQN.
- The relative order of Q(s, a) is more important than the absolute value.

Revisit DQN

Minimizing the Bellman error:

$$\min_{\theta} \mathbf{E}_{(s,a,s')\sim\mathcal{D}}[Q_\theta(s,a) - (r(s,a) + \gamma \max_{a'} Q_\phi(s',a'))]^2$$

Choose the action greedily:

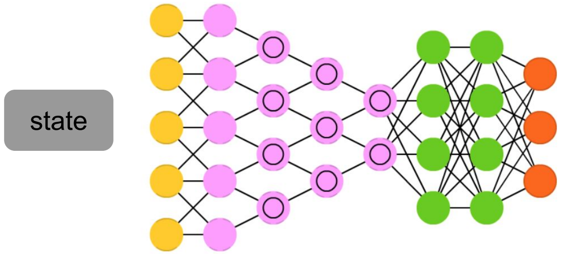
$$\pi(s) = \arg\max_a Q(s,a)$$

Can we learn the relative relationship of actions directly to approach a more sample-efficient algorithm?

$$\arg\max_a \lambda(s,a) = \arg\max_a Q^{\pi_*}(s,a)$$

Ranking Based Reinforcement Learning

The Pairwise Ranking Policy



Policy network

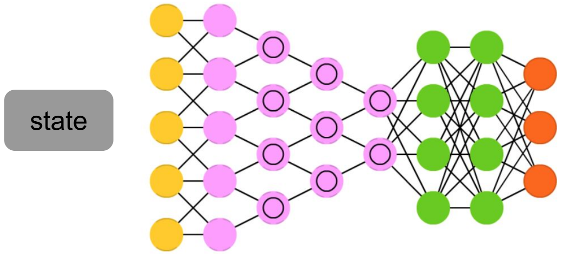
λ_1 a_1

λ_2 a_2

λ_3 a_3

$$a = \arg\max_a \lambda(s, a)$$

The Pairwise Ranking Policy



Policy network

λ_1 a_1

λ_2 a_2

λ_3 a_3

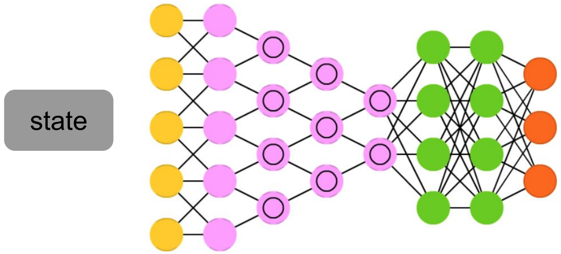
$$a = \underset{a}{\arg\max}\, \lambda(s, a)$$

- Pairwise comparison of actions

  a_1 is better than a_2,

  a_1 is better than a_3.

The Pairwise Ranking Policy



Policy network

λ_1 a_1

λ_2 a_2 $a = \arg\max_a \lambda(s, a)$

λ_3 a_3

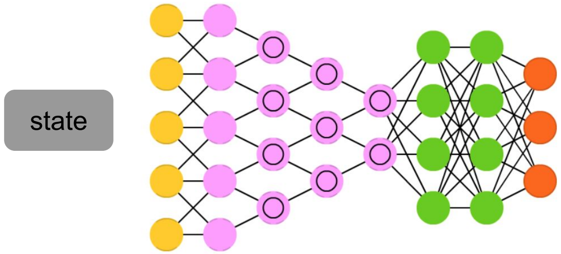
- Pairwise comparison of actions
  a_1 vs a_2, a_1 vs a_3, a_2 vs a_3
- Pairwise learning to rank.
- The probability that action i is ranked higher than action j.

$$p_{ij} = \frac{\exp(\lambda(s, a_i) - \lambda(s, a_j))}{1 + \exp(\lambda(s, a_i) - \lambda(s, a_j))}$$

- The probability that action i to be

$$\pi(a = a_i | s) = \prod_{j=1, j\neq i}^{m} p_{ij}$$

The Pairwise Ranking Policy



Policy network

λ_1 a_1

λ_2 a_2

λ_3 a_3

$$a = \arg\max_{a} \lambda(s, a)$$

- Pairwise comparison of actions
  a_1 vs a_2, a_1 vs a_3, a_2 vs a_3
- Pairwise learning to rank.
- The probability that action i is ranked higher than action j.

$$p_{ij} = \frac{\exp(\lambda(s, a_i) - \lambda(s, a_j))}{1 + \exp(\lambda(s, a_i) - \lambda(s, a_j))}$$

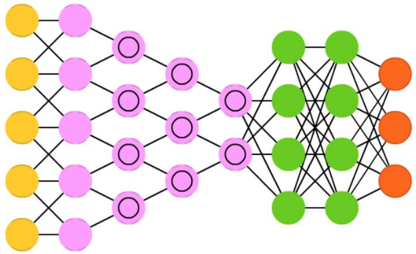
- The probability that action i to be

$$\pi(a = a_i | s) = \prod_{j=1, j \neq i}^{m} p_{ij}$$

Action with highest return ➡ with highest probability ➡ with highest λ

The Pairwise Ranking Policy

state



Policy network

λ_1 $\quad$ a_1

λ_2 $\quad$ a_2 $\qquad a = \arg\max_a \lambda(s, a)$

λ_3 $\quad$ a_3

- Pairwise comparison of actions
  a_1 vs a_2, a_1 vs a_3, a_2 vs a_3
- Pairwise learning to rank.
- The probability that action i is ranked higher than action j.

$$p_{ij} = \frac{\exp(\lambda(s, a_i) - \lambda(s, a_j))}{1 + \exp(\lambda(s, a_i) - \lambda(s, a_j))}$$

- The probability that action i to be

$$\pi(a = a_i | s) = \prod_{j=1, j\neq i}^{m} p_{ij}$$

Action with highest return $\rightarrow$ with highest probability $\rightarrow$ with highest λ

How to optimize this w.r.t episodic reward?

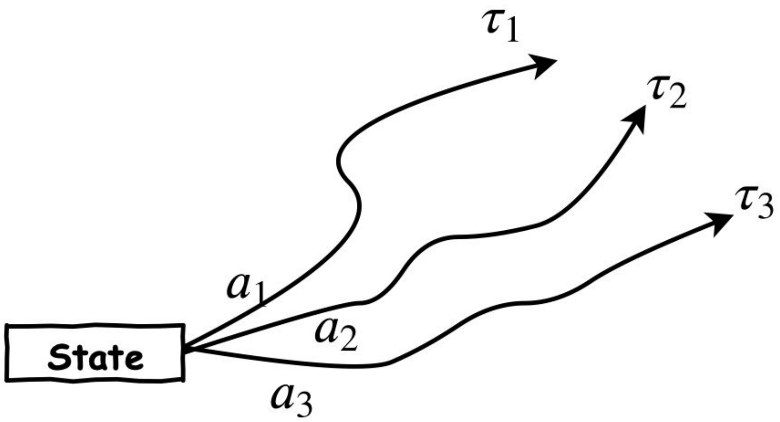
Direct Policy Differentiation

Expected long-term reward

$$J(\theta) = \sum_{\tau} p_\theta(\tau) r(\tau)$$

Trajectory probability $\quad p_\theta(\tau) = p(s_0)\Pi_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$

Trajectory reward $\quad r(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$



Direct Policy Differentiation

Expected long-term reward

$$J(\theta) = \sum_{\tau} p_\theta(\tau) r(\tau)$$

Trajectory probability $\quad p_\theta(\tau) = p(s_0)\Pi_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$

Trajectory reward $\quad r(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$

$$\max_\theta J(\theta)$$ Maximizing long-term reward

$$\theta \leftarrow \theta + \nabla J(\theta)$$ Gradient accent

Direct Policy Differentiation

Expected long-term reward

$$J(\theta) = \sum_{\tau} p_{\theta}(\tau) r(\tau)$$

Trajectory probability $\quad p_{\theta}(\tau) = p(s_0)\Pi_{t=1}^{T} \pi_{\theta}(a_t|s_t) p(s_{t+1}|s_t, a_t)$

Trajectory reward $\quad r(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$

Pairwise ranking policy

$$\pi(a = a_i|s) = \prod_{j=1, j\neq i}^{m} p_{ij}$$

$$\nabla_{\theta} J(\theta) = \mathbf{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^{T} \nabla_{\theta} \left(\sum_{j=1, j\neq i}^{m} (\lambda_i - \lambda_j)/2 \right) r(\tau) \right]$$

Ranking policy gradient (RPG)

$$J(\theta) = \sum_{\tau} p_{\theta}(\tau) r(\tau)$$

$$\Big\downarrow \text{ Pairwise ranking policy}$$

$$\nabla_{\theta} J(\theta) = \mathbf{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^{T} \nabla_{\theta} \left(\sum_{j=1, j \neq i}^{m} (\lambda_i - \lambda_j)/2 \right) r(\tau) \right]$$

$$\Big\downarrow \text{ Deterministic policy}$$

$$a = \arg\max_{a} \lambda(s, a)$$

Ranking policy gradient (RPG)

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau\sim\pi_\theta}\left[\sum_{t=1}^{T} \nabla_\theta \left(\sum_{j=1,j\neq i}^{m} (\lambda_i - \lambda_j)/2\right) r(\tau)\right]$$

$$a = \arg\max_a \lambda(s, a)$$

- Indications:
  - Minimizing trajectory reward-weighted hinge-loss is policy gradient.
  - Policy logits can be used to denote the rank of actions.
  - Policy logits can be used for decision making explicitly.

Ranking policy gradient (RPG)

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau \sim \pi_\theta} \left[\sum_{t=1}^{T} \nabla_\theta \left(\sum_{j=1, j \neq i}^{m} (\lambda_i - \lambda_j)/2 \right) r(\tau) \right]$$

$$a = \arg\max_a \lambda(s, a)$$

- However
  - RPG is not a sample-efficient approach
  - It’s a new type of policy gradient learning relative action values.

Ranking policy gradient (RPG)

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau \sim \pi_\theta}\left[\sum_{t=1}^{T} \nabla_\theta \left(\sum_{j=1, j \neq i}^{m} (\lambda_i - \lambda_j)/2\right) r(\tau)\right]$$

$$a = \arg\max_{a} \lambda(s, a)$$

- However
  - RPG is not sample-efficient approach
  - It's a new type of policy gradient.

$$\pi_1 \rightarrow \theta \leftarrow \theta + \nabla J(\theta) \rightarrow \pi_2$$

D1: Data collected from $\pi_1 \not\rightarrow \pi_2$

Prior works on off-policy learning

- Off-policy via importance sampling
  - Trades off bias and variance.
- Off-policy via value function methods
  - Suffers from unstable optimization procedure.

A general off-policy framework?

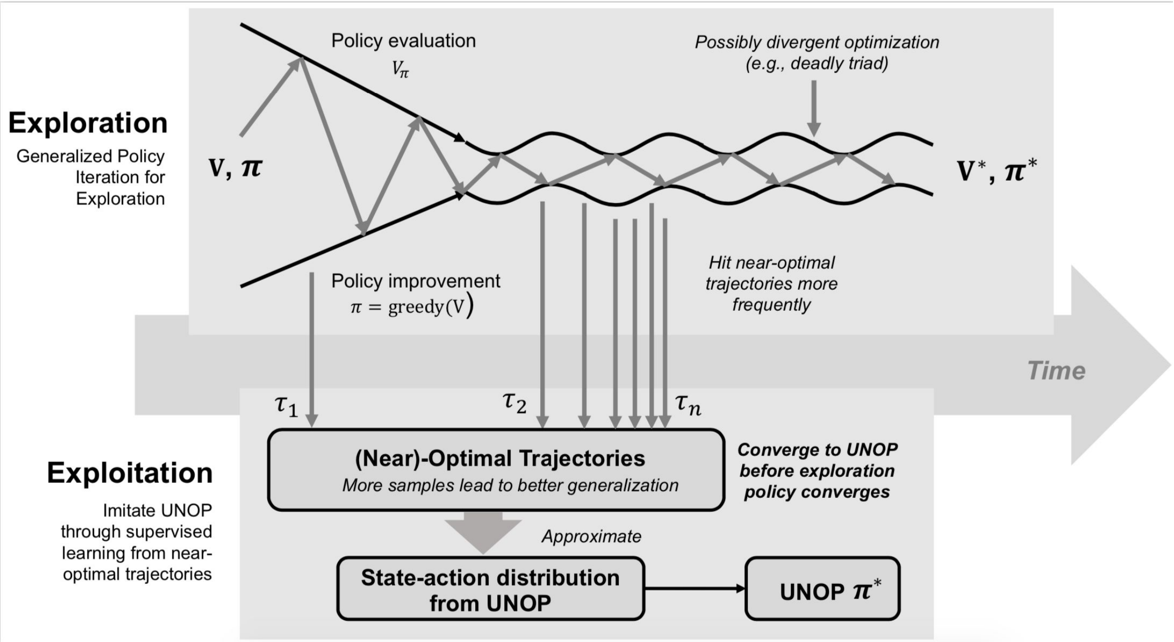
- Stable optimization:
  - stationary target
  - i.i.d. assumption

vs. Q-learning based methods
IQN, DDQN, Rainbow, etc.

- Unbiasedness
- Variance reduction

vs. Importance sampling methods,
ACER, etc.

Two-stage off-policy learning



Two-stage off-policy learning

- Trajectory reward shaping, TRS:

$$w(\tau) = \begin{cases} 1, & \text{if } r(\tau) \geq c \\ 0, & o.w. \end{cases}$$

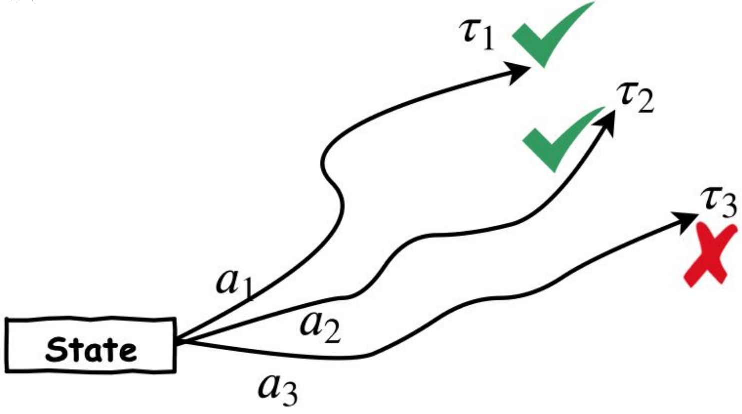
Domain knowledge

- Long-term performance (optimality preserving)

$$\sum_{\tau} p_\theta(\tau) r(\tau) \Rightarrow \sum_{\tau} p_\theta(\tau) w(\tau).$$

- Uniformly (Near)-Optimal Policy, UNOP:

$$p_{\pi_*}(\tau) = \frac{1}{|\mathcal{T}|}, \forall \tau \in \mathcal{T}$$



Reduce RL to Supervised Learning

Long-term reward: $\sum_{\tau} p_\theta(\tau) r(\tau)$

Trajectory reward shaping $\sum_{\tau} p_\theta(\tau) w(\tau)$

Optimizing the lower bound by

$$\arg\max_{\theta} \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}_s} p_{\pi_*}(s, a) \log \pi_\theta(a|s)$$

UNOP

Pairwise Ranking policy

$$\min_{\theta} \sum_{s, a_i} p_{\pi_*}(s, a_i) \left(\sum_{j=1, j \neq i}^{m} \max(0, 1 + \lambda(s, a_j) - \lambda(s, a_i)) \right)$$

Reduce RL to Supervised Learning

Long-term reward: $\sum_\tau p_\theta(\tau) r(\tau)$

Trajectory reward shaping $\sum_\tau p_\theta(\tau) w(\tau)$

Optimality preserving

Optimizing the lower bound by

$$\arg\max_\theta \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}_s} p_{\pi_*}(s, a) \log \pi_\theta(a|s)$$

UNOP

Pairwise Ranking policy

$$\min_\theta \sum_{s, a_i} p_{\pi_*}(s, a_i) \left(\sum_{j=1, j \neq i}^{m} \max(0, 1 + \lambda(s, a_j) - \lambda(s, a_i)) \right)$$

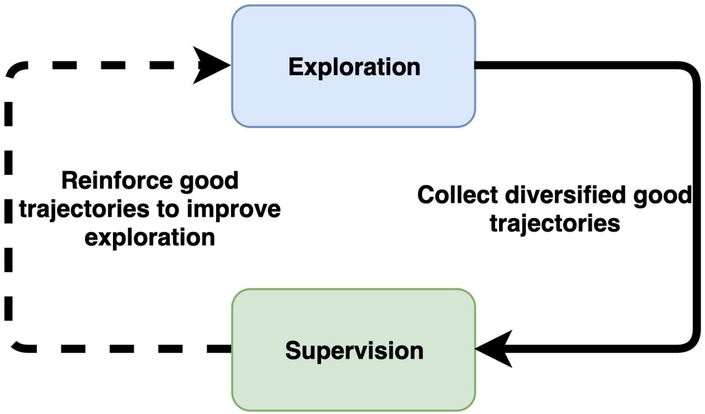
Two-stage off-policy learning framework

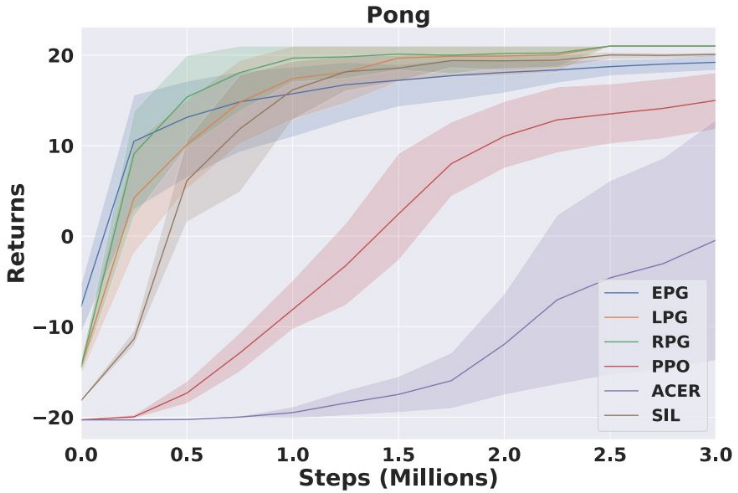
- Exploration stage:
  - To collect different (near)-optimal trajectories asap.
- Supervision stage:
  - Maximize log-likelihood of state-action pairs from near-optimal trajectories. Minimize hinge loss for RPG.

- Empirical evidence
  - Sufficient amount of (near)-optimal samples has been collected, before the state-of-the-art converge to (near)-optimal performance.
- Theoretical advantage:
  - The upper bound of gradient variance is reduced by an order of $O(T^2 R_{max}^2)$

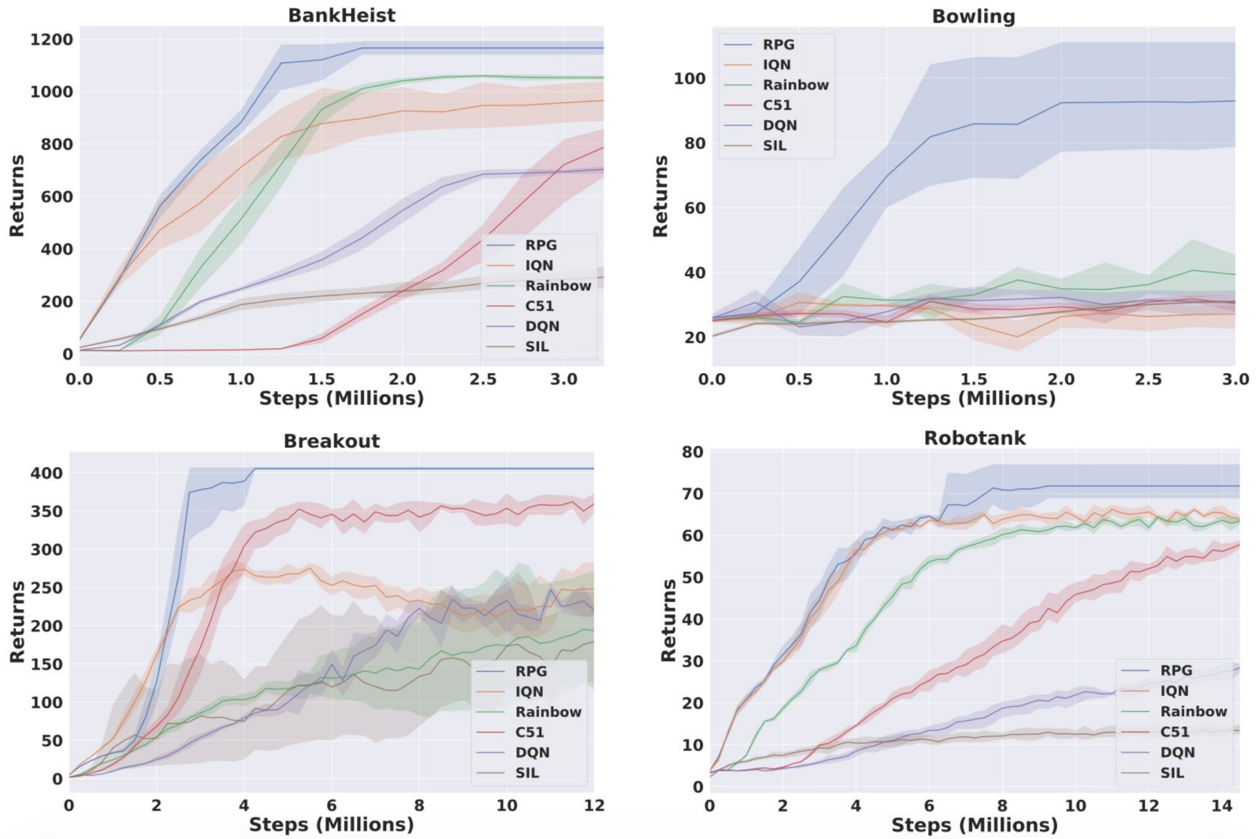
Experimental results

- EPG: VPG + Off-policy learning (stochastic)
- LPG: VPG + Off-policy learning (deterministic)
- RPG: EPG exploration and RPG for learning policy.

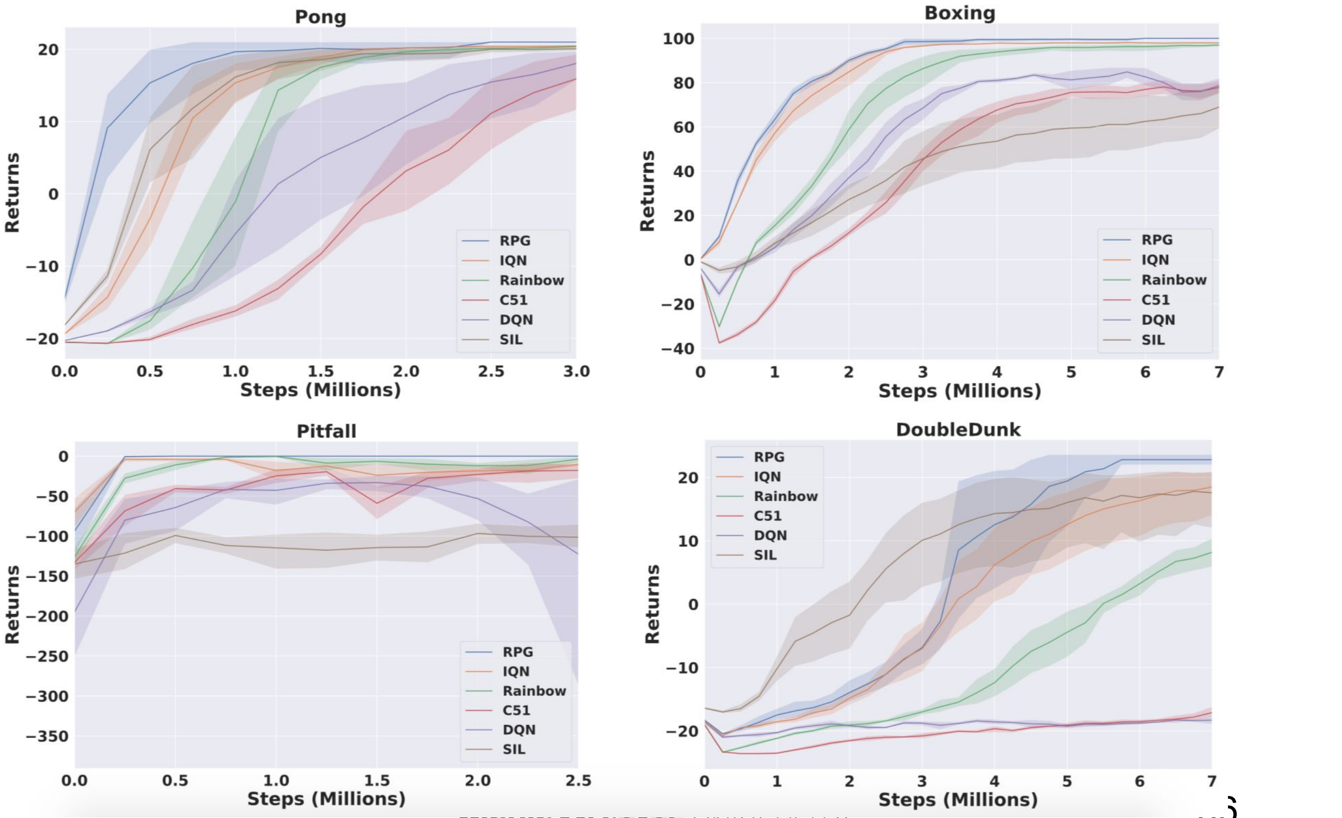




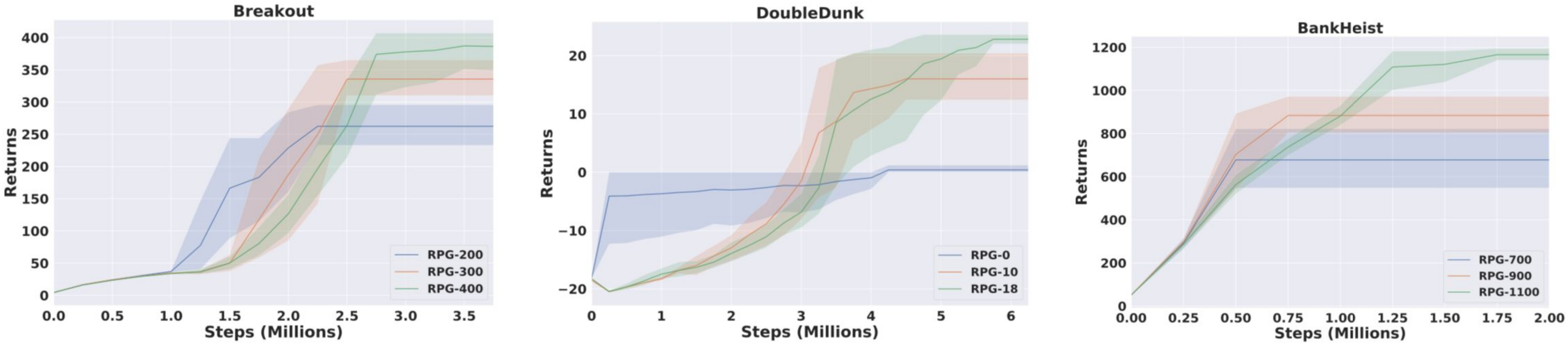
Sample-efficiency



Sample-efficiency



Optimality vs Efficiency



The trajectory reward threshold c trades-off the efficiency and optimality

Conclusions

- We propose the ranking policy gradient (RPG) that learns the optimal rank of actions.
- Formalize policy logits as relative action value denoting the rank of actions.
- Reduce RL as supervised learning, enable off-policy learning, reducing variance, and preserving optimality at the same time.
- Propose a two-stage off-policy reinforcement learning framework that leads to more sample-efficient results.

Q &A

Email: linkaixi@msu.edu

Limitations

- Our task
  - Finite MDP, discrete action space.
  - Episodic task, undiscounted.
- Explicitly
  - Independence of action ranks, for the pairwise ranking policy. e(a1>a2), e(a1>a3).
  - Bounded gradient norm, for the variance reduction.
  - The existence of uniformly (near)-optimal policy, for reducing RL to SL.
- Implicitly
  - Prior knowledge of the trajectory reward threshold.
  - Sufficient amount of near-optimal trajectories can be explored via exploration algorithms.