

Deep Reinforcement Learning

Outline

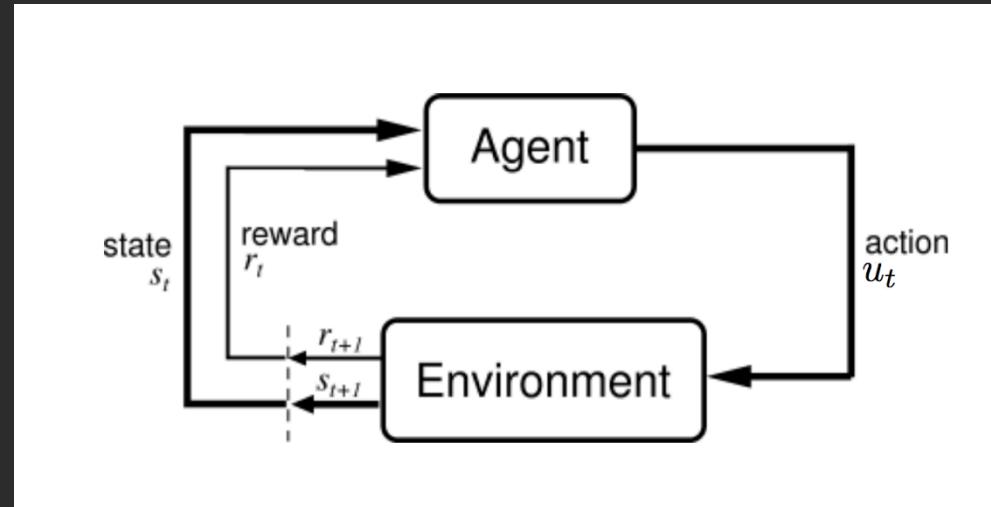
1. Overview of Reinforcement Learning
2. Policy Search
3. Policy Gradient and Gradient Estimators
4. Q-prop: Sample Efficient Policy Gradient and an Off-policy Critic
5. Model Based Planning in Discrete Action Space

Note: These slides largely derive from David Silver's video lectures + slides

<http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>

Reinforcement Learning 101

Agent	Entity interacting with its surroundings
Environment	Surroundings in which the agent interacts with
State	Representation of agent and environment configuration
Reward	Measure of success for positive feedback



Reinforcement Learning 101

Policy	Map of the agent's actions given the state.	Deterministic policy: $a = \pi(s)$ Stochastic policy: $\pi(a s) = \mathbb{P}[A_t = a S_t = s]$
V(S)= Value Function	Expectation Value of the future reward given a specific policy, starting at state S(t)	$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots S_t = s]$
Q = Action-Value Function	Expectation value of the future reward following a specific policy, after a specific action at a specific state.	$q_{\pi}(s, a) = \mathbb{E}_{\pi} [R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) S_t = s, A_t = a]$
Model	Predicts what the environment will do next.	$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' S_t = s, A_t = a]$ $\mathcal{R}_s^a = \mathbb{E} [R_{t+1} S_t = s, A_t = a]$

Policy Evaluation

Run policy iteratively in environment while updating $Q(a,s)$ or $V(s)$, until convergence:

Model Based Evaluation

Learn Model from experience (Supervised Learning).
Learn Value function $V(s)$ from model.

$$\begin{aligned} S_1, A_1 &\rightarrow R_2, S_2 \\ S_2, A_2 &\rightarrow R_3, S_3 \\ \vdots \\ S_{T-1}, A_{T-1} &\rightarrow R_T, S_T \end{aligned}$$

$$\begin{aligned} S_{t+1} &\sim \mathcal{P}_\eta(S_{t+1} | S_t, A_t) \\ R_{t+1} &= \mathcal{R}_\eta(R_{t+1} | S_t, A_t) \end{aligned}$$

Pros: Efficiently learns model and can reason about model uncertainty

Cons: two sources of error from model and approximated $V(s)$

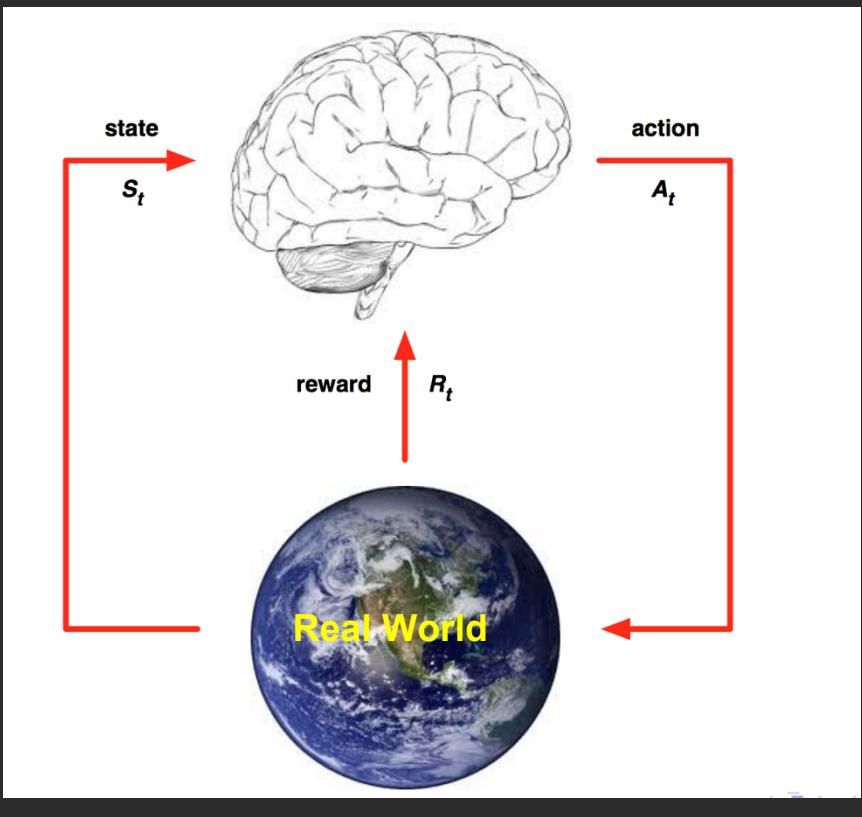
Model Free Evaluation

Learn from experience (sampling). Greedy policy over $V(s)$ requires model

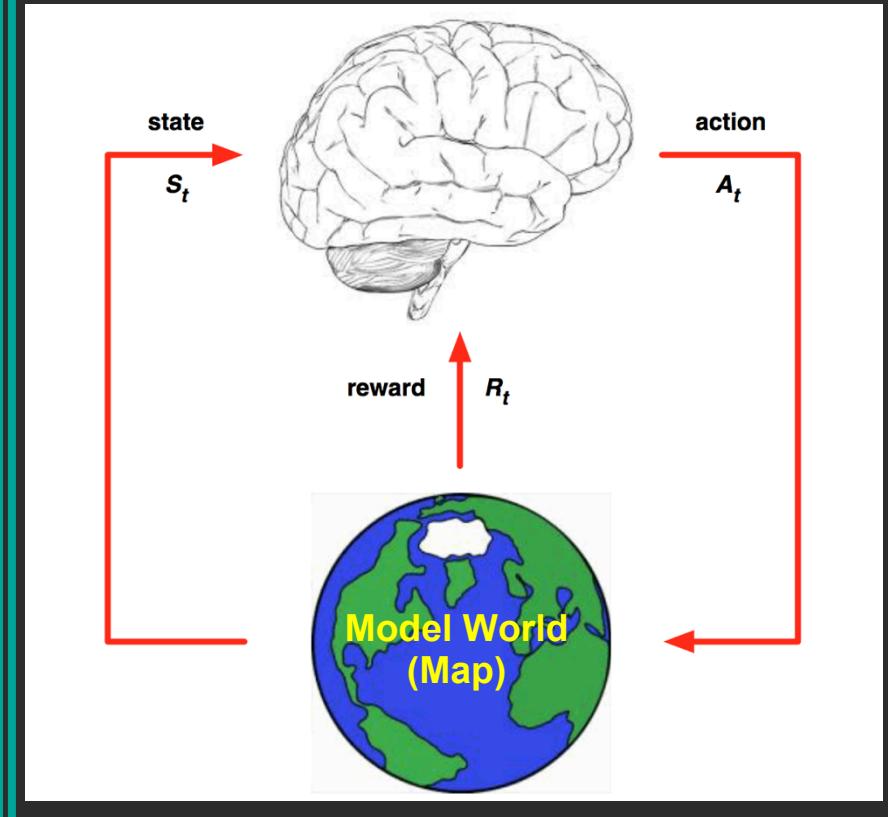
$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

Evaluation over action space: $Q = q_\pi$

Model Based



Model Free



Policy Evaluation Method: Monte Carlo (MC) versus Temporal Dynamics (TD)

Monte Carlo

Return

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

Update Value toward actual return after episode trajectory

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

- Better for non-Markov
- High Variance, no bias
- Only for offline

Temporal Dynamics

Learns directly from incomplete episodes of experience from bootstrapping.

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- Better for Markov
- Low bias, low variance
- Offline and Online

Policy Improvement

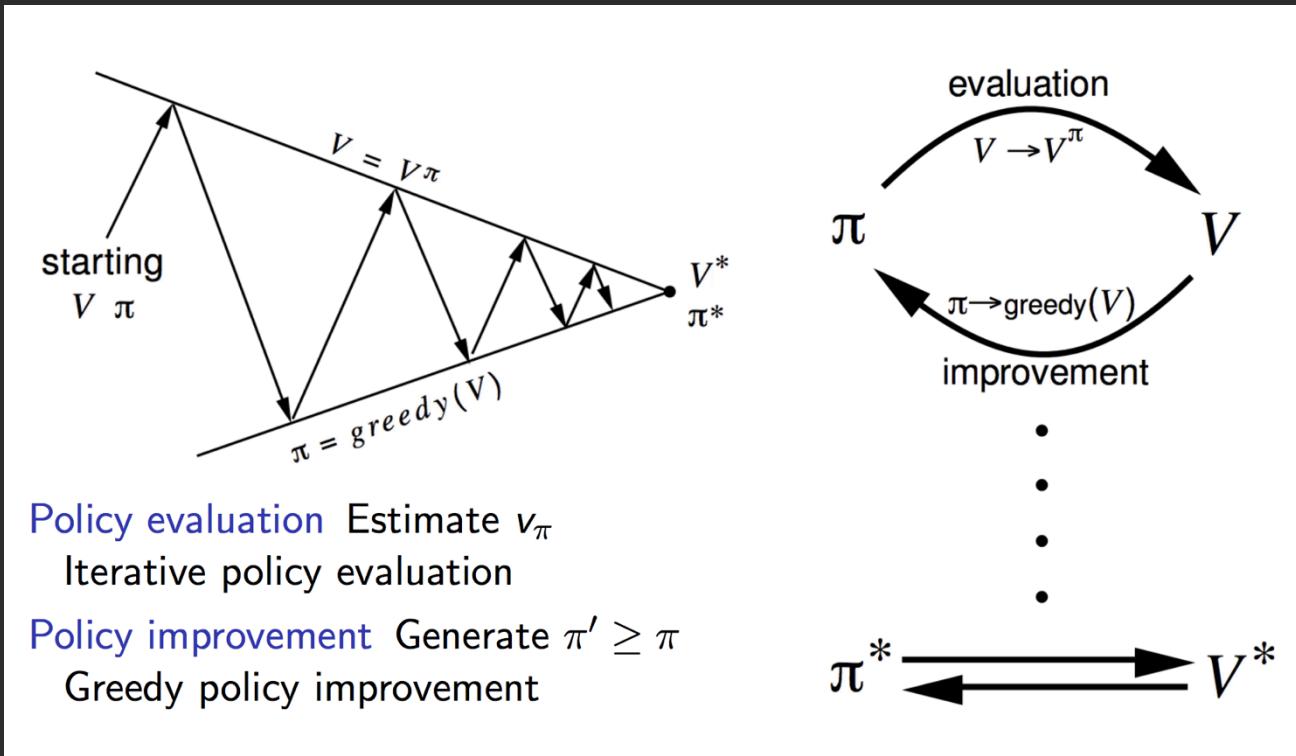
Update policy from the $V(s)$ and/or $Q(a,s)$ after iterated policy evaluation

Epsilon-Greedy

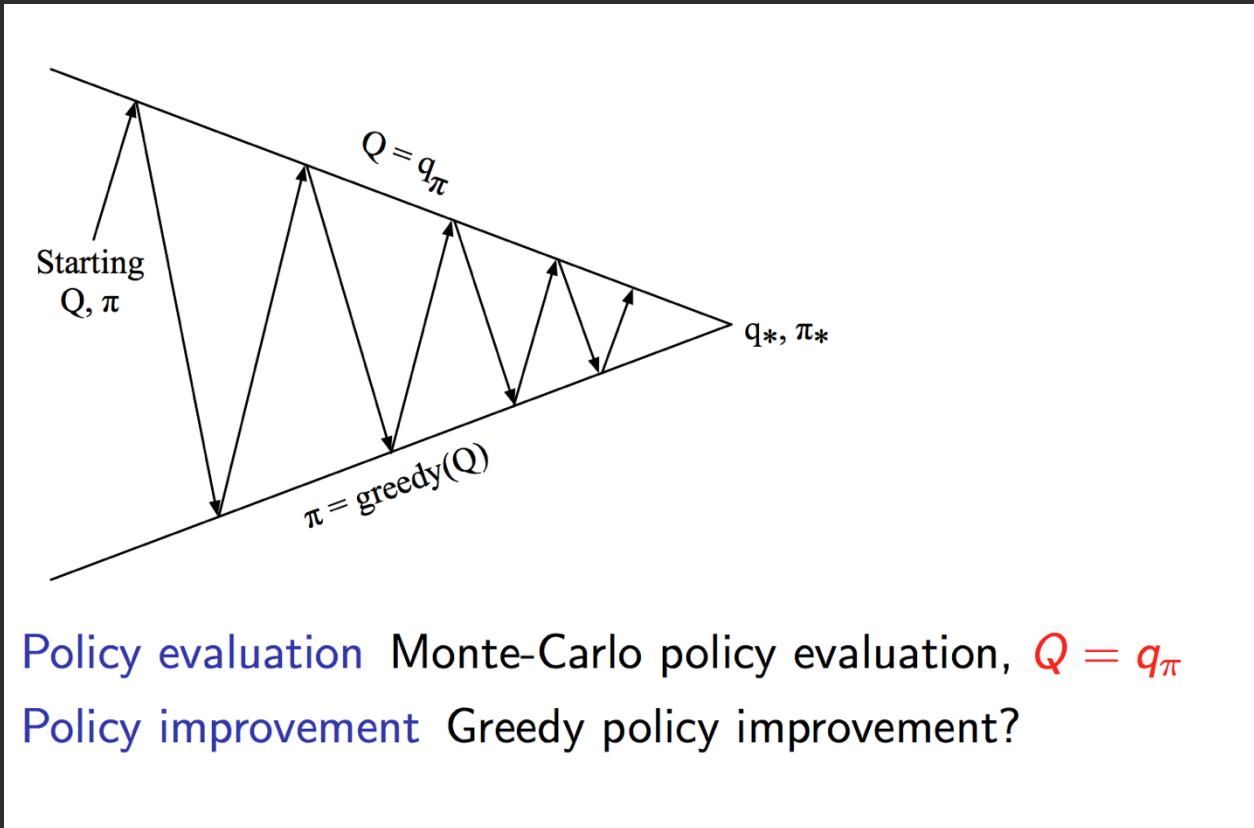
- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Generalized Policy Iteration $V(s)$



Generalized Policy Iteration Q(a,s)



Function Approximation for Large MDP Systems

Problem: Recall every state(s) has an entry $V(s)$ and every action, state pair has an entry $Q(a,s)$. This is problematic for large systems with many state pairs.

Solution: Estimate value function with approximation function. Generalize from seen states to unseen states and update parameter w using MC or TD learning.

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

or

$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

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

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

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

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

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

On-policy and Off-policy Control Methods

- On-policy methods: the agent learns from experiences drawn from its own behavioural policy.
 - Example of on-policy: SARSA, TRPO
- Off-policy methods: the agent optimizes its own policy using samples from another target policy (ex: an agent learning by observing a human).
 - Example of off-policy: Q-learning (next slide)
 - Qualities: Can provide sample efficiency, but can lack convergence guarantees and suffer from instability issues.

Off-policy example: Q-learning

- Target policy acts greedily, behaviour acts epsilon-greedily.
- Bootstrap w.r.t. the target policy in the Q update assignment.

```
Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$   
Repeat (for each episode):
```

```
    Initialize  $S$ 
```

```
    Repeat (for each step of episode):
```

```
        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
```

```
        Take action  $A$ , observe  $R, S'$ 
```

```
        
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

```

```
         $S \leftarrow S'$ ;
```

```
    until  $S$  is terminal
```

Policy Gradient Methods

Idea: Use function approximation on the policy:

$$\pi_{\theta}(s, a) = \mathbb{P}[s, a | \theta]$$

Given its parameterization, we can directly optimize the policy. Take gradient of:

$$J(\theta) = \mathbb{E}_{\pi_{\theta}}[R_0] = \mathbb{E}_{\pi_{\theta}} \left[\sum_{t=0}^{t=\infty} \gamma^t r(s_t, a_t) \right]$$

Policy Gradient Methods: Pros / Cons

Advantages:

- Better convergence properties (updating tends to be smoother)
- Effective in high-dimensional/cts action spaces (avoid working out max)
- Can learn stochastic policies (more on this later)

Disadvantages:

- Converge often to local minima
- Can be inefficient to evaluate policy + have high variance (max operation can be viewed as more aggressive)

Policy Gradient Theorem

Assuming our policy is differentiable, can prove that (Sutton, 1999):

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

Useful formulation that moves the gradient past the distribution over states, providing model-free gradient estimator.

Monte Carlo Policy Gradient Methods

Most straightforward approach = REINFORCE:

```
function REINFORCE
    Initialise  $\theta$  arbitrarily
    for each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$  do
        for  $t = 1$  to  $T - 1$  do
             $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$ 
        end for
    end for
    return  $\theta$ 
end function
```

Problems:

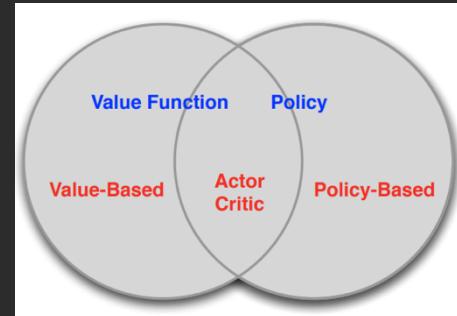
- High variance (can get rid of some through control variate)
- Sample intensive (attempts to use off-policy data have failed).
- Not online (have to calculate the return)

Policy Gradient with Function Approximation

Approximate the gradient with a critic:

$$\nabla J(\theta) \approx \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)]$$

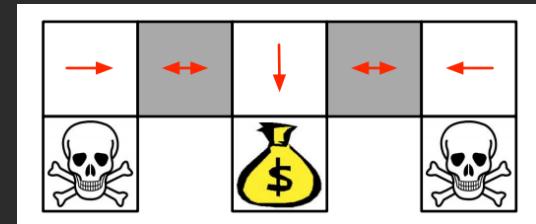
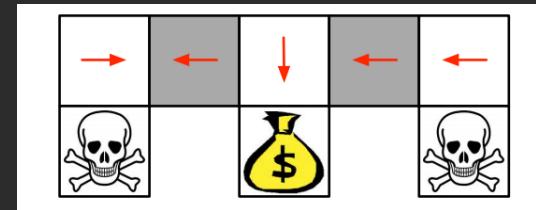
- Employ techniques from before (e.g. Q-learning) to update Q. Off-policy techniques provide sample efficiency.
- Can have reduced variance compared to REINFORCE (replacing full-step mc return with for example one-step TD return).



Deterministic vs. Stochastic Policies

Stochastic policies:

- Can break symmetry in aliased features
- If on-policy, get exploration



Deterministic policies:

- Bad in POMDP/adversarial settings
- More efficient

Why is deterministic more efficient?

- Recall policy gradient theorem:

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \int_S \rho^{\pi}(s) \boxed{\int_A \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) da} ds \\ &= \mathbb{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a)] \quad (2)\end{aligned}$$

- With stochastic policy gradient, the inner integral (red box in 2) is computed by sampling a high dimensional action space. In contrast, the deterministic policy gradient can be computed immediately in closed form.

$$\begin{aligned}\nabla_{\theta} J(\mu_{\theta}) &= \int_S \rho^{\mu}(s) \nabla_{\theta} \mu_{\theta}(s) \left. \nabla_a Q^{\mu}(s, a) \right|_{a=\mu_{\theta}(s)} ds \\ &= \mathbb{E}_{s \sim \rho^{\mu}} \left[\nabla_{\theta} \mu_{\theta}(s) \left. \nabla_a Q^{\mu}(s, a) \right|_{a=\mu_{\theta}(s)} \right] \quad (9)\end{aligned}$$

$a = \mu_{\theta}(s)$

Q-Prop: Sample Efficient Policy Gradient with an Off-Policy Critic

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, Sergey Levine

Q-Prop: Relevance

- Challenges
 - On-policy estimators: sample efficiency, high variance with MC PG methods
 - Off-policy estimators: unstable results, non-convergence emanating from bias
- Related Recent Work
 - Variance reduction in gradient estimators is an ongoing active research area..
 - Silver, Schulman etc. TRPO, DDPG

Q-Prop: Main Contributions

- Q-prop provides a new approach for using off-policy data to reduce variance in an on-policy gradient estimator without introducing further bias.
- Coalesce prior advances in dichotomous lines of research since Q-Prop uses both on-policy updates and off-policy critic learning.

Q-Prop: Background

- Monte Carlo (MC) Policy Gradient (PG) Methods:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \gamma^t R_t \right] = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) (R_t - b(\mathbf{s}_t)) \right], \quad (1)$$

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\mathbf{s}_t \sim \rho_{\pi}(\cdot), \mathbf{a}_t \sim \pi(\cdot | \mathbf{s}_t)} [\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) (R_t - b(\mathbf{s}_t))]. \quad (2)$$

$$\begin{aligned} V_{\pi}(\mathbf{s}_t) &= \mathbb{E}_{\pi}[R_t] = \mathbb{E}_{\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)} [Q_{\pi}(\mathbf{s}_t, \mathbf{a}_t)] \\ Q_{\pi}(\mathbf{s}_t, \mathbf{a}_t) &= r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\pi}[R_{t+1}] = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\rho(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [V_{\pi}(\mathbf{s}_{t+1})] \\ A_{\pi}(\mathbf{s}_t, \mathbf{a}_t) &= Q_{\pi}(\mathbf{s}_t, \mathbf{a}_t) - V_{\pi}(\mathbf{s}_t). \end{aligned} \quad (3)$$

- PG with Function Approximation or Actor-Critic Methods

- Policy evaluation step: fit a critic Q_w (using TD learning for e.g.) for the current policy π
- Policy improvement step: optimize policy π against critic estimated Q_w
- Significant gains in sample efficiency using off-policy (memory replay) TD learning for the critic
 - E.g. method: Deep Deterministic Policy Gradient (DDPG) [Silver et. al. 2014], used in Q-Prop
 - (Biased) Gradient (in policy improvement phase) given by:

$$\nabla_{\theta} J(\theta) \approx \mathbb{E}_{\mathbf{s}_t \sim \rho_{\beta}(\cdot)} [\nabla_{\mathbf{a}} Q_w(\mathbf{s}_t, \mathbf{a})|_{\mathbf{a}=\mu_{\theta}(\mathbf{s}_t)} \nabla_{\theta} \mu_{\theta}(\mathbf{s}_t)] \quad (5)$$

$$w = \arg \min_w \mathbb{E}_{\mathbf{s}_t \sim \rho_{\beta}(\cdot), \mathbf{a}_t \sim \beta(\cdot | \mathbf{s}_t)} [(r(\mathbf{s}_t, \mathbf{a}_t) + \gamma Q(\mathbf{s}_{t+1}, \mu_{\theta}(\mathbf{s}_{t+1})) - Q_w(\mathbf{s}_t, \mathbf{a}_t))^2]$$

$$\theta = \arg \max_{\theta} \mathbb{E}_{\mathbf{s}_t \sim \rho_{\beta}(\cdot)} [Q_w(\mathbf{s}_t, \mu_{\theta}(\mathbf{s}_t))]$$

Q-Prop: Estimator

- Intuition: Q-Prop is simply a Monte Carlo PG estimator with a special form of control variate. Uses the first-order Taylor expansion of critic's Q_w as baseline control variate.

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\rho_{\pi}, \pi} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (\hat{Q}(s_t, a_t) - \bar{Q}_w(s_t, a_t)] + \mathbb{E}_{\rho_{\pi}} [\nabla_a Q_w(s_t, a)|_{a=\mu_{\theta}(s_t)} \nabla_{\theta} \mu_{\theta}(s_t)]. \quad (7)$$

- In terms of advantages for the basic derivation:

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \mathbb{E}_{\rho_{\pi}, \pi} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (\hat{A}(s_t, a_t) - \bar{A}_w(s_t, a_t)] + \mathbb{E}_{\rho_{\pi}} [\nabla_a Q_w(s_t, a)|_{a=\mu_{\theta}(s_t)} \nabla_{\theta} \mu_{\theta}(s_t)] \\ \hat{A}(s_t, a_t) &= \bar{Q}(s_t, a_t) - \mathbb{E}_{\pi_{\theta}} [\bar{Q}(s_t, a_t)] = \nabla_a Q_w(s_t, a)|_{a=\mu_{\theta}(s_t)} (a_t - \mu_{\theta}(s_t)). \end{aligned} \quad (8)$$

- For adapting Q-Prop, weighing variable $\eta(s_t)$ is added:

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \mathbb{E}_{\rho_{\pi}, \pi} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (\hat{A}(s_t, a_t) - \eta(s_t) \bar{A}_w(s_t, a_t))] \\ &\quad + \mathbb{E}_{\rho_{\pi}} [\eta(s_t) \nabla_a Q_w(s_t, a)|_{a=\mu_{\theta}(s_t)} \nabla_{\theta} \mu_{\theta}(s_t)] \end{aligned} \quad (9)$$

- Variance of the estimator is given using a surrogate variance measure for tractability:

$$\begin{aligned} \text{Var}^* &= \mathbb{E}_{\rho_{\pi}} [\text{Var}_{a_t} (\hat{A}(s_t, a_t) - \eta(s_t) \bar{A}(s_t, a_t))] \\ &= \text{Var} + \mathbb{E}_{\rho_{\pi}} [-2\eta(s_t) \text{Cov}_{a_t} (\hat{A}(s_t, a_t), \bar{A}(s_t, a_t)) + \eta(s_t)^2 \text{Var}_{a_t} (\bar{A}(s_t, a_t))]. \end{aligned} \quad (11)$$

- Optimal state dependent factor $\eta(s_t)$ is computed as:

$$\eta^*(s_t) = \text{Cov}_{a_t} (\hat{A}, \bar{A}) / \text{Var}_{a_t} (\bar{A})$$

Adaptive Q-Prop and Variants

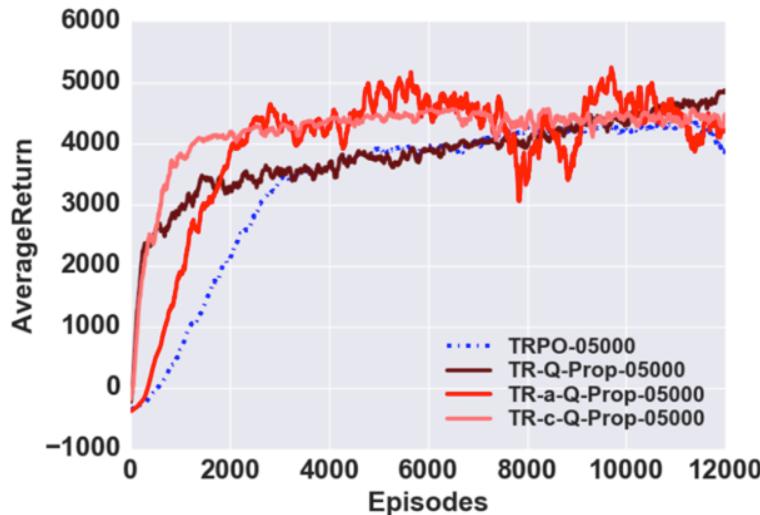
- The various variants of Q-Prop are obtained using variants of the control-variate modulating variable $\eta(s_t)$.
- **Adaptive Q-Prop**: using optimal $\eta^*(s_t)$ for variance reduction (Eq. 11): $\text{Var}^* = \mathbb{E}_{\rho_\pi}[(1 - \rho_{corr}(\hat{A}, \bar{A})^2)\text{Var}_{a_t}(\hat{A})]$
 - Achieves variance reduction if at any state, actor and critic advantage functions are correlated.
- **Conservative Q-Prop** (c-Q-Prop): $\eta(s_t) = 1$ if there is positive covariance, else 0. Effectively disables the control variate for some samples of the states.
- **Aggressive Q-Prop** (a-Q-Prop): $\eta(s_t) = \text{sign}(\text{Cov}(\hat{A}, \bar{A}))$. Control variates are always used regardless of covariance relation

Q-Prop: Algorithm

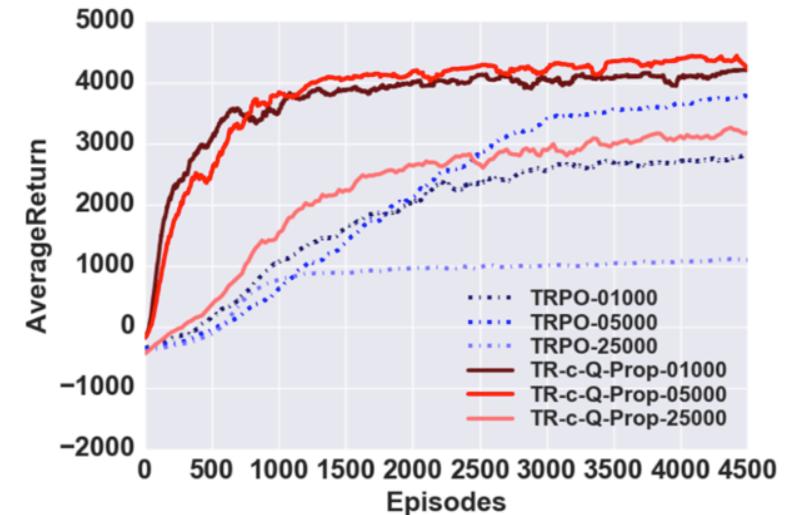
Algorithm 1 Adaptive Q-Prop

- 1: Initialize w for critic Q_w , θ for stochastic policy π_θ , and replay buffer $\mathcal{R} \leftarrow \emptyset$.
- 2: **repeat**
- 3: **for** $e = 1, \dots, E$ **do** ▷ Collect E episodes of on-policy experience using π_θ
- 4: $s_{0,e} \sim p(s_0)$
- 5: **for** $t = 0, \dots, T - 1$ **do**
- 6: $a_{t,e} \sim \pi_\theta(\cdot | s_{t,e}), s_{t+1,e} \sim p(\cdot | s_{t,e}, a_{t,e}), r_{t,e} = r(s_{t,e}, a_{t,e})$
- 7: Add batch data $\mathcal{B} = \{s_{0:T,1:E}, a_{0:T-1,1:E}, r_{0:T-1,1:E}\}$ to replay buffer \mathcal{R}
- 8: Take $E \cdot T$ gradient steps on Q_w using \mathcal{R} and π_θ
- 9: Fit $V_\phi(s_t)$ using \mathcal{B}
- 10: Compute $\hat{A}_{t,e}$ using GAE(λ) and $\bar{A}_{t,e}$ using Eq. 7
- 11: Set $\eta_{t,e}$ based on Section 3.2
- 12: Compute and center the learning signals $l_{t,e} = \hat{A}_{t,e} - \eta_{t,e}\bar{A}_{t,e}$
- 13: Compute $\nabla_\theta J(\theta) \approx \frac{1}{ET} \sum_e \sum_t \nabla_\theta \log \pi_\theta(a_{t,e} | s_{t,e}) l_{t,e} + \eta_{t,e} \nabla_a Q_w(s_{t,e}, a) |_{a=\mu_\theta(s_{t,e})} \nabla_\theta \mu_\theta(s_{t,e})$
- 14: Take a gradient step on π_θ using $\nabla_\theta J(\theta)$, optionally with a trust-region constraint using \mathcal{B}
- 15: **until** π_θ converges.

Q-Prop: Experiments and Evaluations



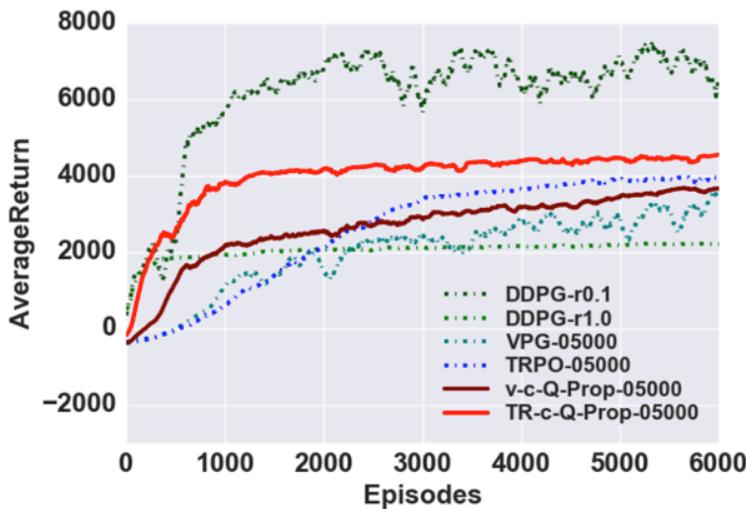
(a) Standard Q-Prop vs adaptive variants.



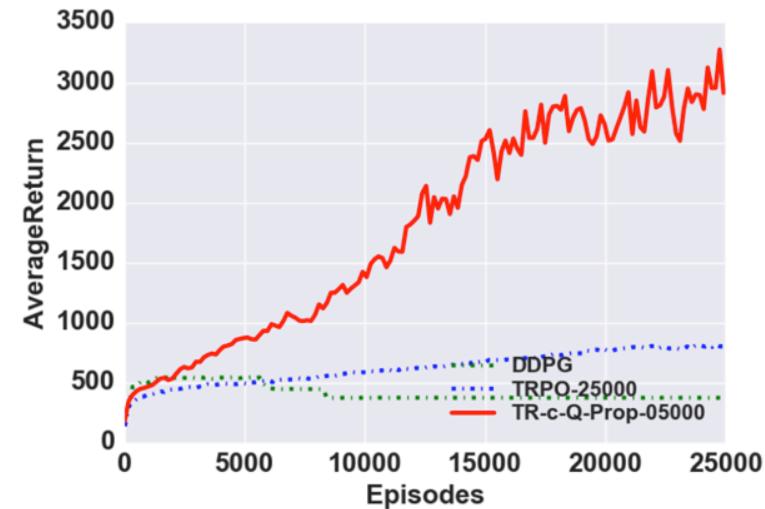
(b) Conservative Q-Prop vs TRPO across batch sizes.

All variants of Q-Prop substantially outperform TRPO in terms of sample efficiency

Q-Prop: Evaluations Across Algorithms



(a) Comparing algorithms on HalfCheetah-v1.



(b) Comparing algorithms on Humanoid-v1.

TR-c-Q-Prop outperforms VPG, TRPO. DDPG is inconsistent (dependent on hyper-parameter settings (like reward scale – r – here))

Q-Prop: Evaluations Across Domains

Q-Prop, TRPO and DDPG results showing the max average rewards attained in the first 30k episodes and the episodes to cross specific reward thresholds.

Domain	Threshold	TR-c-Q-Prop		TRPO		DDPG	
		MaxReturn.	Episodes	MaxReturn	Epsisodes	MaxReturn	Episodes
Ant	3500	3534	4975	4239	13825	957	N/A
HalfCheetah	4700	4811	20785	4734	26370	7490	600
Hopper	2000	2957	5945	2486	5715	2604	965
Humanoid	2500	>3492	14750	918	>30000	552	N/A
Reacher	-7	-6.0	2060	-6.7	2840	-6.6	1800
Swimmer	90	103	2045	110	3025	150	500
Walker	3000	4030	3685	3567	18875	3626	2125

Take away: Q-Prop often learns more sample efficiently than TRPO and can solve difficult domains such as Humanoid better than DDPG.

Q-Prop: Limitations

- Speed: the compute time per episode is bound by the critic training at each iteration. Poses a limitation of usage with fast simulators where data collection is very fast.
 - Possible work around: asynchronous data collection and policy updates to fit Q_w
- Robustness to Bad Critics: estimating off-policy critic's reliability is a fundamental issue that requires further investigation.
 - Possible work around: adopt more stable state-of-the-art critic learning techniques such as Retrace (Munos et. al. 2016)

Q-Prop: Future Work

- Q-Prop was implemented using TRPO-GAE for this paper.
- Combining Q-Prop with other on-policy update schemes and off-policy critic training methods is an interesting direction of future work.

Model-Based Planning in Discrete Action Spaces

By: Mikael Henaff, William F. Whitney, Yann LeCun

Model-based Reinforcement Learning

Recall: model-based RL uses a learned model of the world (i.e. how it changes as the agent acts).

The model can then be used to devise a way to get from a given state s_0 to a desired state s_f , via a sequence of actions.

This is in contrast to the model-free case, which learns directly from states and rewards.

Benefits:

- Model reusability (e.g. can just change reward if task changes)
- Better sample complexity (more informative error signal)
- In *continuous* case, can optimize efficiently

Notation and Learning the Forward Model

θ : Learned forward model parameters

$a = (a_1, \dots, a_r)$: Sequence of actions

$(s, a, s') \sim \mathcal{E}$: Environment transitions

$f(s, a, \theta)$: Predicted state from s after a

$\mathcal{L}(s, \tilde{s})$: Loss function between states

Use example transitions
from the environment E
to learn the forward
model f by minimizing L

E.g. f can be a neural
network

Learned model
parameters:

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(s,a,s') \sim \mathcal{E}} [\mathcal{L}(f(s, a, \theta), s')]$$

Planning in Model-based Reinforcement Learning

Goal: given f , find the sequence of actions a that takes us from a starting state s_0 to a desired final state s_f

$$a^* = \arg \min_a \mathcal{L}(f(s_0, a, \theta), s_f)$$

In the continuous case, this can be done via gradient descent in action space.

But what if the action space is discrete?

Problems in Discrete Action Spaces

Suppose our discrete space is one-hot encoded with dimension d

$$\text{Action Space : } \mathcal{A} = \{e_1, \dots, e_d\}$$

- It is too expensive to enumerate the tree of possibilities and find the optimal path (reminiscent of classical AI search e.g. in games)
- If we treat A as a vector space and naively attempt continuous optimization, it is likely that the resulting action will be *invalid*, i.e. not an allowed action

Can we somehow map this to a differentiable problem, more amenable to optimization?

Handling Discreteness (I): Overview

Two approaches are used to ameliorate the problems caused by discreteness:

1. Softening the action space and relaxing the discrete optimization problem allows back-propagation to be used with gradient descent
2. Biasing the algorithm to producing action vectors that are close to valid, by additive noise (implicit) or an entropy penalty (explicit)

Handling Discreteness (II): Soften & Relax

Define a new input space for the actions, defined by the d-dimensional simplex

$$\Delta^d = \Delta^{|\mathcal{A}|} = \{z : z = \sigma(x) \forall x \in \mathbb{R}^d\}$$

Notice that we can get a softened action from any real vector by taking its softmax

$$a_t = \sigma(x_t)$$

Relaxing the optimization then gives (notice the x's are not restricted):

$$x^* = \arg \min_{x=(x_1, \dots, x_T)} \mathcal{L}(f(s_0, \sigma(x), \theta), s_f)$$

Note: the softmax is applied element-wise

Handling Discreteness (III): Optimization Bias

The paper considers 3 ways to push the “input” x_t ’s towards one-hot vectors during the optimization procedure:

1. Add noise to the input x_t ’s
2. Add noise to the gradients (scaled version of 1.)
3. Add an explicit penalty to the loss function, given by the entropy of the softened action $H(\sigma(x_t))$

This entropy is a good measure for how well this bias (or regularization) is working (since low entropy means furthest from uniform, i.e. more concentration at one value)

Why Does Adding Noise Help?

Adding noise to the inputs x_t implicitly induces the following additional penalty to the optimization objective:

Noise variance
(strength)

Encourage less sensitivity to inputs
(e.g. going to saturated softmax areas)

$$P = \sum_i \epsilon_i \left[\frac{\partial^2 f(s_0, \sigma(x), \theta)}{\partial x_i^2} \frac{\partial \mathcal{L}}{\partial s'} + \left| \frac{\partial f}{\partial x_i} \right|^2 \right]$$

Also less sensitivity, by penalizing low loss but high curvature (e.g. sharp or unstable local minima)

The Overall Planning Algorithm

Algorithm 1 Forward Planner

Require: Trained forward model f , initial state s_0 , desired final state s' , learning rate η .

- 1: Initialize $x_t \in \mathbb{R}^{|\mathcal{A}|}$ from $\mathcal{N}(0, 0.1)$ for $t = 1, \dots, T$.
 - 2: **for** $i = 1 : k$ **do**
 - 3: $x_t \leftarrow x_t + \epsilon$ for $t = 1, \dots, T$. ▷ Add noise to inputs
 - 4: $a_t \leftarrow \sigma(x_t)$ for $t = 1, \dots, T$. ▷ Compute action vectors
 - 5: $s \leftarrow f(s_0, a_1, \dots, a_T, \theta)$ ▷ Predict final state for this action sequence
 - 6: Compute $\mathcal{L}(s, s')$ and ∇s ▷ Forward and backprop through \mathcal{L}
 - 7: Compute ∇a_t for $t = 1, \dots, T$ ▷ Backprop through f using ∇s
 - 8: Compute ∇x_t for $t = 1, \dots, T$ ▷ Backprop through σ using ∇a_t
 - 9: $x_t \leftarrow \text{ADAM}(x_t, \nabla x_t, \eta)$ for $t = 1, \dots, T$. ▷ Update using ADAM
 - 10: **end for**
 - 11: $a_t \leftarrow \sigma(x_t)$ for $t = 1, \dots, T$.
 - 12: $a_t \leftarrow \arg \min_{e_i \in \{e_1, \dots, e_d\}} \|a_t - e_i\|$ for $t = 1, \dots, T$. ▷ Quantize actions to one-hot vectors
 - 13: **return** a_1, \dots, a_T
-

Evaluation: Two New Discrete Planning Tasks

Based on classic Q&A tasks, but “reversed” (here we predict a from s_f)

- (A) Navigate: find discrete moving and turning sequence to reach target position
- (B) Transport: reproduce object locations by agent picking up objects & moving

(A) NAVIGATION TASK

QA TASK	PLANNING TASK
AGENT1 IS AT (8, 4)	AGENT1 IS AT (8, 4)
AGENT1 FACES-E	AGENT1 FACES-E
AGENT1 MOVES-2	*
AGENT1 MOVES-5	*
AGENT1 FACES-S	*
AGENT1 MOVES-5	*
AGENT1 FACES-S	*
AGENT1 MOVES-1	*
AGENT1 MOVES-1	*
AGENT1 MOVES-1	*
Q1: WHERE IS AGENT1?	Q1: WHERE IS AGENT1?
A1: *	A1: (10, 1)

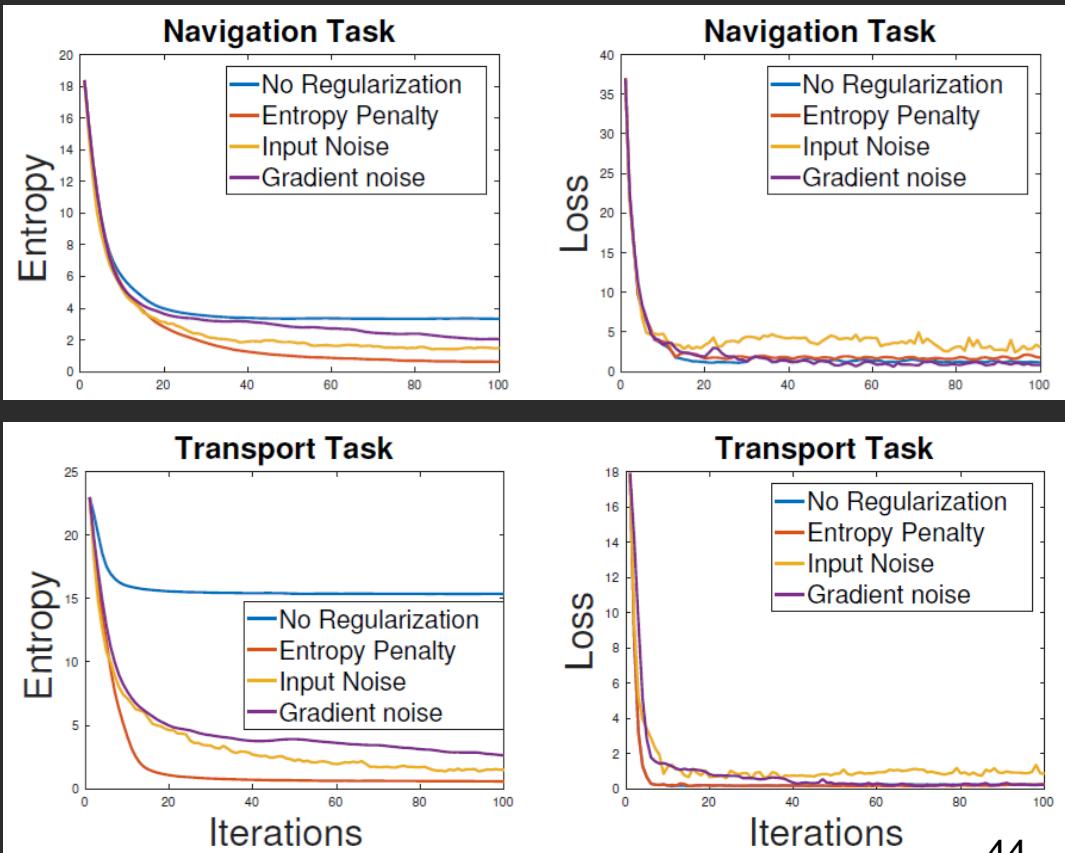
(B) TRANSPORT TASK

QA TASK	PLANNING TASK
OBJECT1 IS AT LOCATION3	OBJECT1 IS AT LOCATION3
OBJECT2 IS AT LOCATION4	OBJECT2 IS AT LOCATION4
OBJECT3 IS AT LOCATION2	OBJECT3 IS AT LOCATION2
JASON WENT TO LOCATION3	JASON WENT TO LOCATION3
JASON PICKED UP THE OBJECTS	*
JASON WENT TO LOCATION2	*
JASON PICKED UP THE OBJECTS	*
JASON WENT TO LOCATION1	*
Q1: WHERE IS OBJECT1?	Q1: WHERE IS OBJECT1?
Q2: WHERE IS OBJECT2?	Q2: WHERE IS OBJECT2?
Q3: WHERE IS OBJECT3?	Q3: WHERE IS OBJECT3?
A1: *	A1: LOCATION1
A2: *	A2: LOCATION4
A3: *	A3: LOCATION1

Results (I): Entropy and Loss over Time

Empirically, adding noise directly to the inputs seems to be the best of the 3 implicit loss regularization methods (possibly helps avoid local minima too)

One can also see that the entropy decreases over time, when regularization is present (right)



Results (II): Performance Comparison

The method (the Forward Planner) was compared to **Q-learning** and an **imitation learner**. It does better at generalizing for longer sequences (outside training data)

Issue: the Forward Planner takes much longer to choose (i.e. plan) its actions. But if even if given less time, it still performs reasonably well.

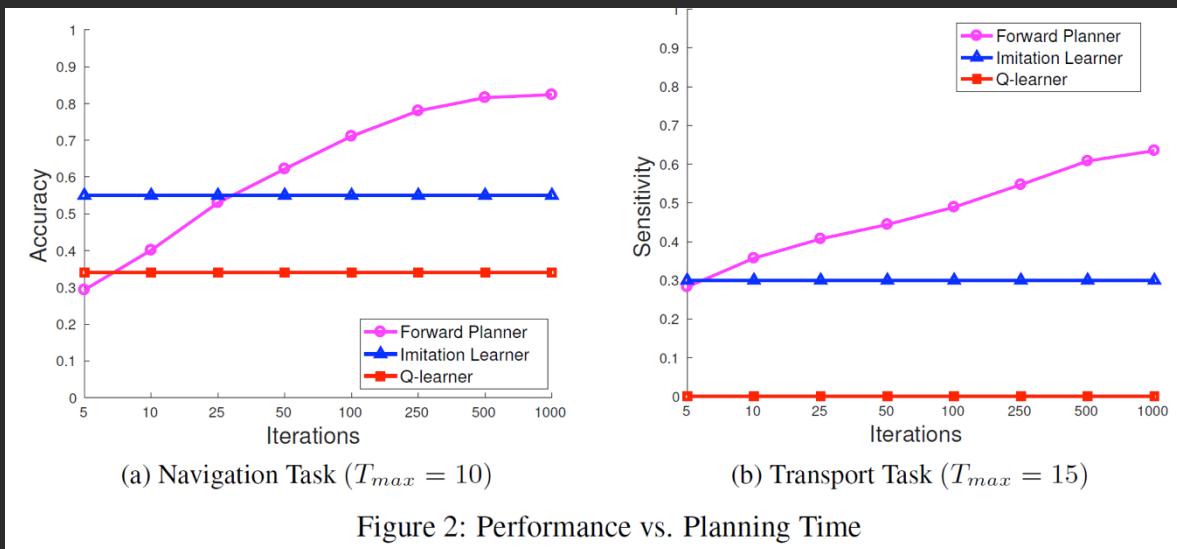


Figure 2: Performance vs. Planning Time

Summary of Paper

- Devise a way to perform model-based planning in discrete actions spaces via gradient-based optimization
 - Combines: (1) relaxation of the problem and action space, and (2) a penalty that biases the algorithm naturally towards preferring low entropy (soft) actions
- Defined two new discrete RL tasks and demonstrated their model's state-of-the-art performance on them

Thank you

Appendix

REINFORCE

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^t R_t \right] = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (R_t - b(s_t)) \right], \quad (1)$$

Related Theorems

- Stochastic Policy Gradient Theorem [Sutton et. al., 1999]

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \int_S \rho^{\pi}(s) \int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a)]\end{aligned}$$

- Deterministic Policy Gradient Theorem [Silver et. al. 2015]

$$\begin{aligned}\nabla_{\theta} J(\mu_{\theta}) &= \int_S \rho^{\mu}(s) \nabla_{\theta} \mu_{\theta}(s) \left. \nabla_a Q^{\mu}(s, a) \right|_{a=\mu_{\theta}(s)} ds \\ &= \mathbb{E}_{s \sim \rho^{\mu}} \left[\nabla_{\theta} \mu_{\theta}(s) \left. \nabla_a Q^{\mu}(s, a) \right|_{a=\mu_{\theta}(s)} \right] \quad (9)\end{aligned}$$

Open AI Gym MuJoCo

- Humanoid Demo
 - <https://www.youtube.com/watch?v=SHLuf2ZBQSw>
- Half Cheetah
 - <https://www.youtube.com/watch?v=EzBmQsiUWB>

Estimating the Advantage Function

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating *both* $V^{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s, a)$
- Using two function approximators and two parameter vectors,

$$V_v(s) \approx V^{\pi_\theta}(s)$$

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

$$A(s, a) = Q_w(s, a) - V_v(s)$$

- And updating *both* value functions by e.g. TD learning

Source: David Silver Lecture slides

Deep Deterministic Policy Gradient (DDPG)

- Policy Gradient Theorem (Sutton et. al. 1999):

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \int_{\mathcal{S}} \rho^{\pi}(s) \boxed{\int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) da} ds \\ &= \mathbb{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a)] \quad (2)\end{aligned}$$

- With stochastic policy gradient, the inner integral (red box in 2) is computed by sampling a high dimensional action space. In contrast, the deterministic policy gradient can be computed immediately in closed form.

$$\begin{aligned}\nabla_{\theta} J(\mu_{\theta}) &= \int_{\mathcal{S}} \rho^{\mu}(s) \nabla_{\theta} \mu_{\theta}(s) \left. \nabla_a Q^{\mu}(s, a) \right|_{a=\mu_{\theta}(s)} ds \\ &= \mathbb{E}_{s \sim \rho^{\mu}} \left[\nabla_{\theta} \mu_{\theta}(s) \left. \nabla_a Q^{\mu}(s, a) \right|_{a=\mu_{\theta}(s)} \right] \quad (9)\end{aligned}$$

$a = \mu_{\theta}(s)$