Continuous Deep Q-Learning with Model-based Acceleration

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, Sergey Levine
Google Brain

Presented by: Frederieke Lohmann

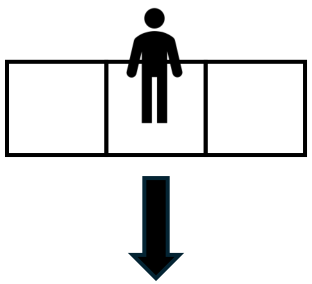
Agenda

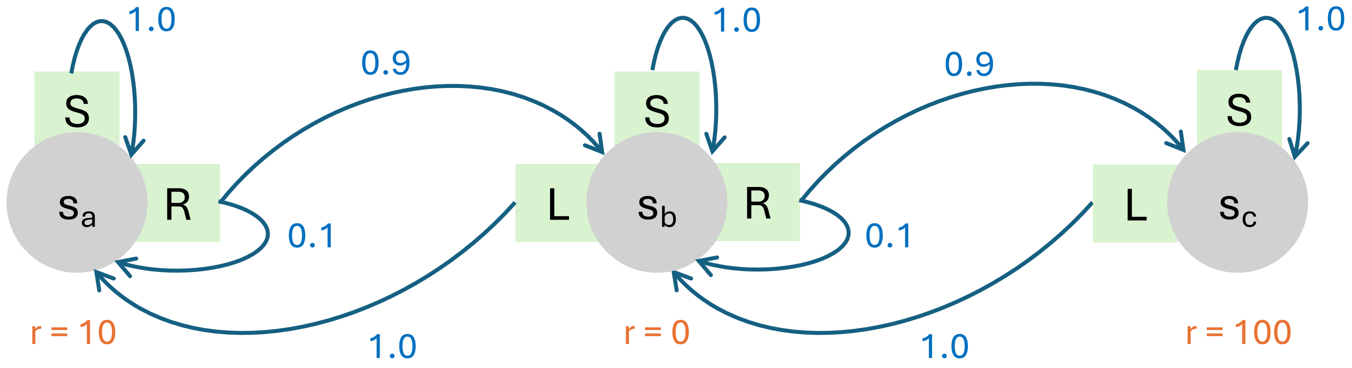
1. Theoretical background
2. Algorithm: Normalized Advantage Function
3. Model-based Acceleration of NAF
4. Experimental Results
5. Outlook
6. Critique

Reinforcement Learning Recap

*An agent learns how to **take actions** in a dynamic, unknown environment in order to **maximize** some cumulative reward*

Environment modeled as a **Markov Decision Process**:



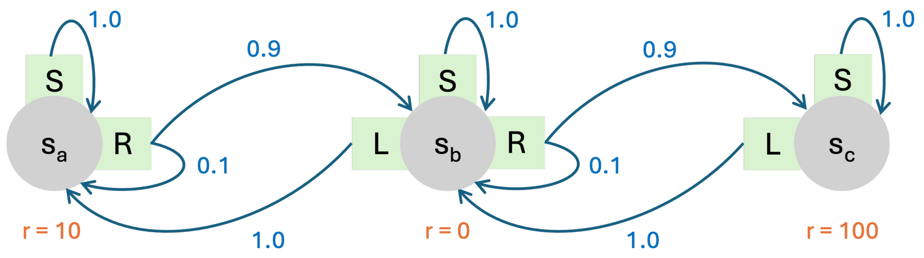


Objective:

$$\underset{\pi(a|s)}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0} \gamma^t r_t \,\middle|\, \pi(a|s), s_0\right]$$

- States $s \in \{s_a, s_b, s_c\}$
- Actions $a \in \{R, L, S\}$
- Transition probabilities $p(s'|s, a)$
- Rewards r

Terminology



- Policy $\pi(a|s)$
  - Strategy which action a to take in state s
- Transition (s, a, r, s')
- Trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, \dots, s_T)$
- Value function V^π
  - $V^\pi(s_0) = \mathbb{E}\left[\sum_{t=0} \gamma^t r_t \,\middle|\, \pi, s_0\right]$
  - Defined w.r.t. some policy
  - Cumulative reward when following the policy from this state
- State-Action Value function Q^π
  - $Q^\pi(s_0, a_0) = \mathbb{E}\left[\sum_{t=0} \gamma^t r_t \,\middle|\, \pi, s_0, a_0\right]$
  - Take any action in the current state
  - Then follow the policy afterwards

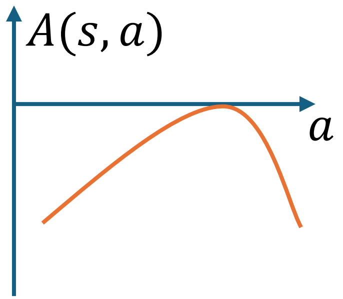
Advantage Function

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

Quantify changes in rewards when going off-policy

If $\max_a A(s,a) = 0$:

$$\max_a Q(s,a) = V(s)$$



Bellman Equation

Recursive way of writing the Value function:

$$V^{\pi}(s_0) = \mathbb{E}_{a_0 \sim \pi(a_0|s_0)} \overbrace{\mathbb{E}_{s_1 \sim p(s_1|s_0,a_0)}\Big[\big(r(s_0, a_0, s_1) + \gamma V^{\pi}(s_1)\big)\Big]}^{Q^{\pi}(s_0, a_0)}$$

Bellman Optimality: $V^*(s_0) = \max_a \left(\mathbb{E}_{s_1 \sim p(s_1|s_0,a_0)}\Big[\big(r(s_0, a_0, s_1) + \gamma V^*(s_1)\big)\Big]\right)$

How can we determine the optimal policy?

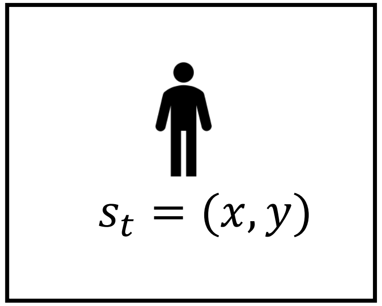
Value Iteration

Use the Bellman optimality!

$$V_{t+1}(s_0) \leftarrow \max_{a} \left(\sum_{s_1} \boxed{p(s_1|s_0,a)} \left(r(s_0,a,s_1) + \gamma V_t(s_1) \right) \right)$$

Model-based

Need to know the model dynamics



Optimal policy: $\pi^*(a|s) = \operatorname*{argmax}_{\hat{a}} Q^*(s,\hat{a})$ (Greedy policy)

TD-Learning

What can we do instead?

- Collected transitions (s,a,r,s') indirectly encode the model dynamics
- Use them to obtain **bootstrap estimate** of the Q-function

$$Q^*(s,a) = \mathbb{E}_{s_1 \sim p(s_1|s,a)} \left(r(s,a,s_1) + \gamma \max_{a_1} Q^*(s_1,a_1) \right) \approx r + \gamma \max_{a'} Q^*(s',a')$$

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha \left(\underbrace{r + \gamma \max_{a'} Q_t(s',a') - Q_t(s,a)}_{\text{TD-Error}} \right)$$

Model-free

Q-Learning

A TD-Learning algorithm

Model-free

Off-policy

1. Collect transitions by following explorative policy (e.g. ε-greedy)
2. Update Rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \underset{\text{Learning rate}}{\alpha}\underbrace{(\overbrace{(r + \gamma \max_a Q(s_{t+1}, a))}^{\text{Target}} - Q(s_t, a_t))}_{\text{TD-Error}}$$

3. Optimal policy: Greedy policy $\pi(a|s) = \underset{a\prime}{\operatorname{argmax}} Q(s, a')$

Until convergence

- Update rule looks like Gradient Ascent!
- Approximate Q-function with a Neural Network ⟶ Deep Q-Learning
  - $Q(s, a) \approx Q(s, a \,|\theta)$
  - Loss $L = \frac{1}{N}\sum_i^N \left(\underbrace{\left(r^{(i)} + \gamma \max_a Q\left(s_{t+1}^{(i)}, a|\theta\right)\right)}_{\text{Label (fixed)}} - \underbrace{Q\left(s_t^{(i)}, a_t^{(i)}|\theta\right)}_{\text{Backprop through this term}}\right)^2$

$s_t = (x, y)$

Taxonomy

Collect transitions (s_t, a_t, r, s_{t+1}) *by …*

On-policy

… following the policy that is optimized

Off-policy

… following a different policy than the optimized one

\+ higher sample efficiency

Optimize the policy …

Model-based

… on a learnt model of the environment

\+ Need fewer real-world rollouts

\- Policy quality limited by quality of learned model

Model-free

… directly on the real environment

\+ Can handle complex systems

Difficulties of Q-Learning

Need to compute $\underset{a}{\operatorname{argmax}}\, Q(s_t, a)$ to collect rollouts and $\max_a Q(s_t, a)$ at every gradient step

⬇

Infeasible to do naively in continuous action spaces

⬇

Actor-Critic Methods

Need to experience good and **bad** transitions **in the beginning** to learn good policy

⬇

Dangerous in safety-critical applications

Need to **interact** with environment **many times** to collect rollouts

⬇

Less sample-efficient than model-based approaches

Deep Deterministic Policy Gradient

Deep RL in continuous action spaces

- Q-Function network and Policy network → 2 sets: Online and Target
- Replay Buffer R stores all transitions

Collect a trajectory (=episode)

Collect the next transition
- Store in R

Update Q-Function and Policy
- On a random batch from R

→ Online networks: Hard update (Gradient Descent)
Target networks: Soft update (Moving Average of Online)

⬇

Stabilize Training

Deep Deterministic Policy Gradient

Deep RL in continuous action spaces

Model-free

Off-policy

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**

Collect transition

    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R

Update networks

    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

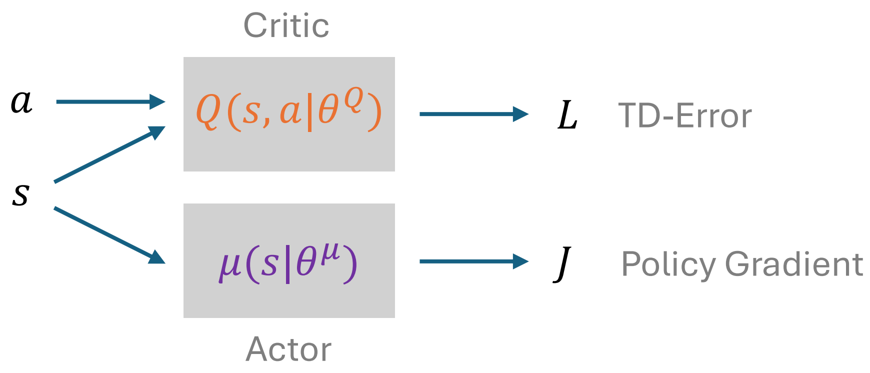
$$J \approx \frac{1}{N}\sum_i Q(s, \mu(s|\theta^\mu)\,|\theta^Q) \longrightarrow \nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

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

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for



$\longrightarrow$ $Q(s, a|\theta^Q)$ and $\mu(s|\theta^\mu)$ trained on different objectives

Does DDPG solve all our problems?

Need to compute $\underset{a}{\operatorname{argmax}}\, Q(s_t, a)$ to collect rollouts and $\max_a Q(s_t, a)$ at every gradient step

⬇

Infeasible to do naively in continuous action spaces

⬇

Need to experience good and **bad** transitions **in the beginning** to learn good policy

⬇

Dangerous in safety-critical applications

⬇

Need to **interact** with environment **many times** to collect rollouts

⬇

Less sample-efficient than model-based approaches

⬇

Continuous Deep Q-Learning with Model-based Acceleration

Normalized Advantage Function

Adapting Q-Learning to continuous action spaces

Model-free

Off-policy

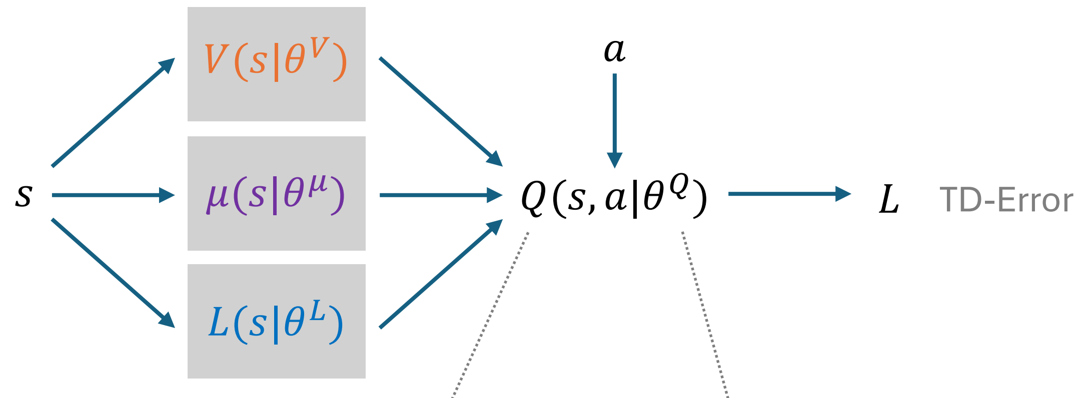
Algorithm 1 Continuous Q-Learning with NAF

Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$.
for episode=1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1 \sim p(\boldsymbol{x}_1)$
  for t=1, T **do**

Collect transition
    Select action $\boldsymbol{u}_t = \mu(\boldsymbol{x}_t|\theta^\mu) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe r_t and $\boldsymbol{x}_{t+1}$
    Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1})$ in R

Update networks
    for iteration=1, I **do**
      Sample a random minibatch of m transitions from R
      Set $y_i = r_i + \gamma V'(\boldsymbol{x}_{i+1}|\theta^{Q'})$ $= r_i + \gamma \max Q'(x_{i+1}, a|\theta^{Q'})$
      Update θ^Q by minimizing the loss: $L \overset{a}{=} \frac{1}{N}\sum_i (y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
      Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    end for
  end for
end for

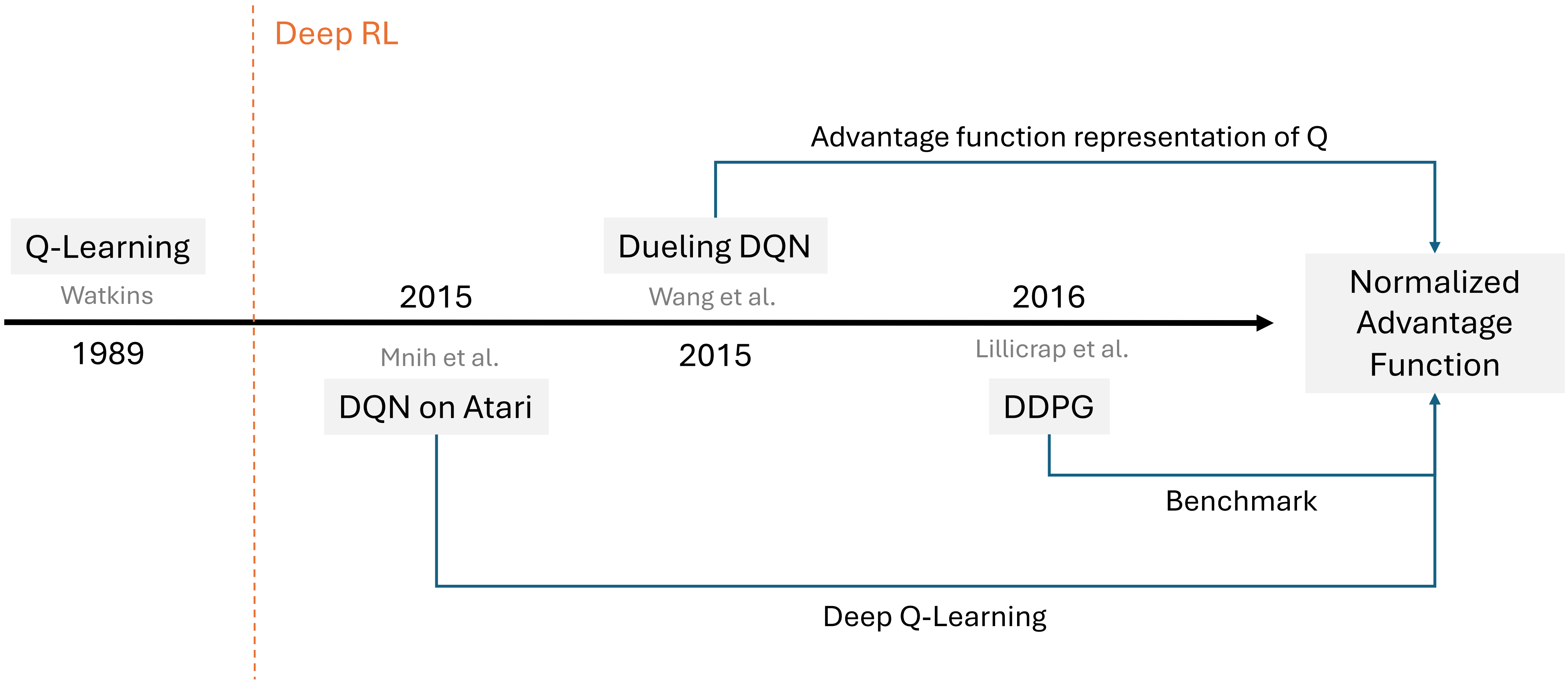
Parametrization of the Q-Function:



$$\underbrace{Q(s,a|\theta^V,\theta^\mu,\theta^L)}_{\theta^Q} = \underbrace{V(s|\theta^V) - \frac{1}{2}(a-\mu(s|\theta^\mu))^T L(s|\theta^L) L(s|\theta^L)^T (a-\mu(s|\theta^\mu))}_{\text{Advantage Function } A(s,a)}$$

➡ $\mu(s|\theta^\mu)$ is trained on same objective as $V(s|\theta^V)$ and $L(s|\theta^L)$

A brief history of (Deep) RL



Does NAF solve all our problems?

Need to compute $\underset{a}{\operatorname{argmax}}\, Q(s_t, a)$ to collect rollouts and $\max_a Q(s_t, a)$ at every gradient step

Infeasible to do naively in continuous action spaces

Need to experience good and **bad** transitions **in the beginning** to learn good policy

Dangerous in safety-critical applications

Need to **interact** with environment **many times** to collect rollouts

Less sample-efficient than model-based approaches

Continuous Deep Q-Learning with Model-based Acceleration

Idea

- What if we had more experience to learn from, especially in the beginning?
- Can we get more experience without the danger of taking bad actions in the real environment?
- Can we leverage the advantages of Model-based learning?

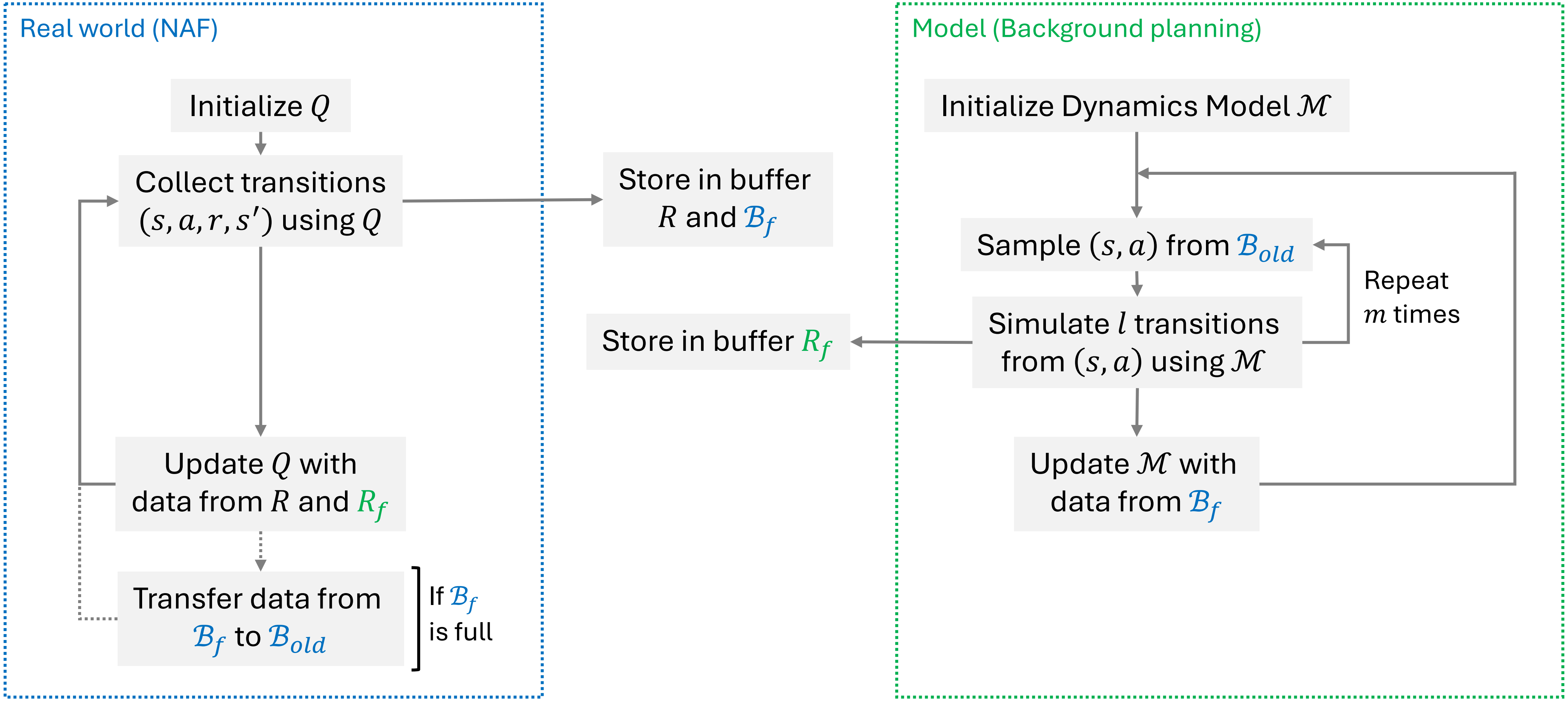
Simulate rollouts using a model of the environment!

Dyna-Q

Model-based

Off-policy

Incorporate experience simulated by a Model (Sutton 1990)



Fitting the Dynamics Model

The iLQG algorithm (Tassa et al. 2012, Levine & Koltun 2013, Levine & Abbeel 2014)

- Collect $n = 5$ episodes $(s_0, a_0, s_1, \dots, s_T)$ in $\mathcal{B}_f$
- For every timestep t:

1. Fit Gaussian to Dynamics:

$$\left\{\left(s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)}\right)_{i=1,\dots,n}\right\} \approx p_t(s_{t+1}, a_t, s_t) \xrightarrow{\text{Condition on } (a_t, s_t)} p_t(s_{t+1} | a_t, s_t)$$

2. Compute Gaussian policy:

Q and V are locally quadratic under the Dynamics model

Policy $\pi_t^{iLQG}(a_t|s_t) = \mathcal{N}(\hat{a}_t + k_t + K_t(s_t - \hat{s}_t), c\Sigma)$ maximizes the locally quadratic Q

Depend on partial derivatives of Q

NAF + Model-based Acceleration

Algorithm 2 Imagination Rollouts with Fitted Dynamics and Optional iLQG Exploration

Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$ and fictional buffer $R_f \leftarrow \emptyset$.
Initialize additional buffers $B_f \leftarrow \emptyset, B_{old} \leftarrow \emptyset$ with size nT.
Initialize fitted dynamics model $\mathcal{M} \leftarrow \emptyset$.
for $episode = 1, M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1$
  Select $\mu'(\boldsymbol{x}, t)$ from $\{\mu(\boldsymbol{x}|\theta^\mu), \pi_t^{iLQG}(\boldsymbol{u}_t|\boldsymbol{x}_t)\}$ with probabilities $\{p, 1-p\}$
  for $t = 1, T$ **do**
    Select action $\boldsymbol{u}_t = \mu'(\boldsymbol{x}_t, t) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe r_t and $\boldsymbol{x}_{t+1}$
    Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1}, t)$ in R and B_f
    if $\mod(episode \cdot T + t, m) = 0$ and $\mathcal{M} \neq \emptyset$ **then**
      Sample m $(\boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}_{i+1}, i)$ from B_{old}
      Use $\mathcal{M}$ to simulate l steps from each sample
      Store all fictional transitions in R_f
    end if
    Sample a random minibatch of m transitions $I \cdot l$ times from R_f and I times from R, and update $\theta^Q, \theta^{Q'}$ as in Algorithm 1 per minibatch.
  end for
  if B_f is full **then**
    $\mathcal{M} \leftarrow$ FitLocalLinearDynamics(B_f)
    $\pi^{iLQG} \leftarrow$ iLQG_OneStep$(B_f, \mathcal{M})$
    $B_{old} \leftarrow B_f, B_f \leftarrow \emptyset$
  end if
end for

Collect transition

Update networks

Switch between real-world rollouts from Greedy Policy and iLQG Policy

Simulate Rollouts

Update networks on real-world rollouts and simulated rollouts

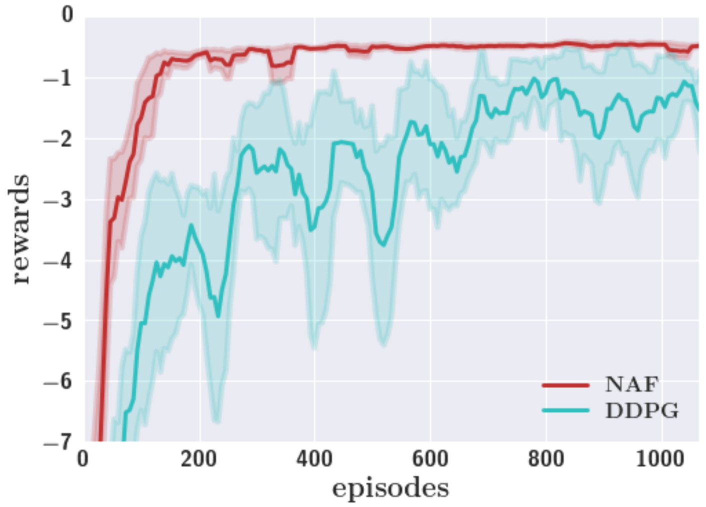
Update the Dynamics Model

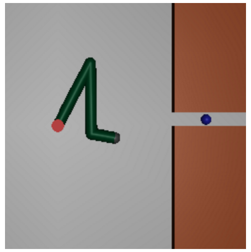
Update the iLQG Policy

Experimental Results

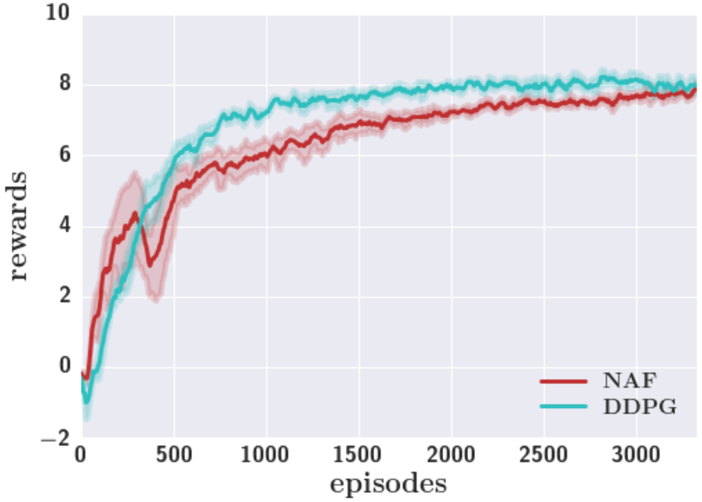
How good is NAF compared to DDPG?

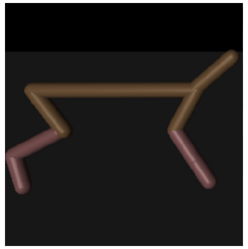
Precision Robotic Tasks





Locomotion Tasks





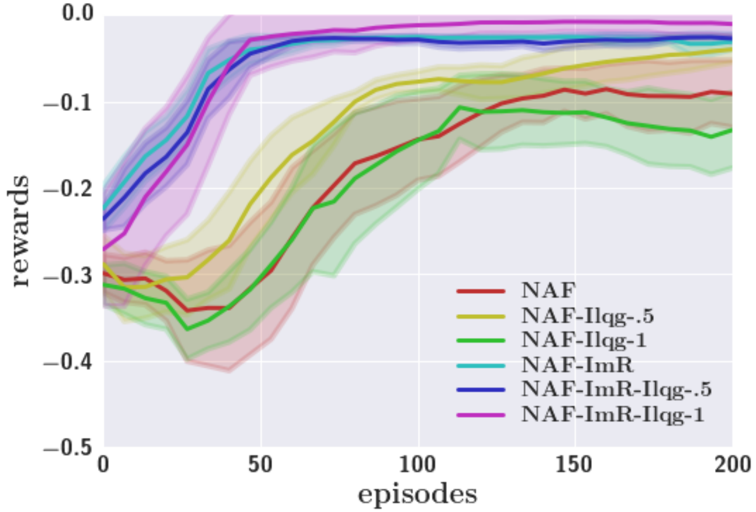
- Faster convergence of NAF
- NAF finds the target precisely
- DDPG fluctuates around target

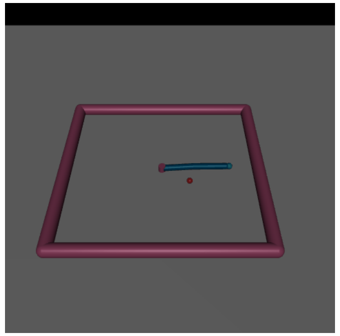
- Similar performance of NAF and DDPG
- Faster convergence of DDPG
    - Mode-seeking behavior of NAF

Experimental Results

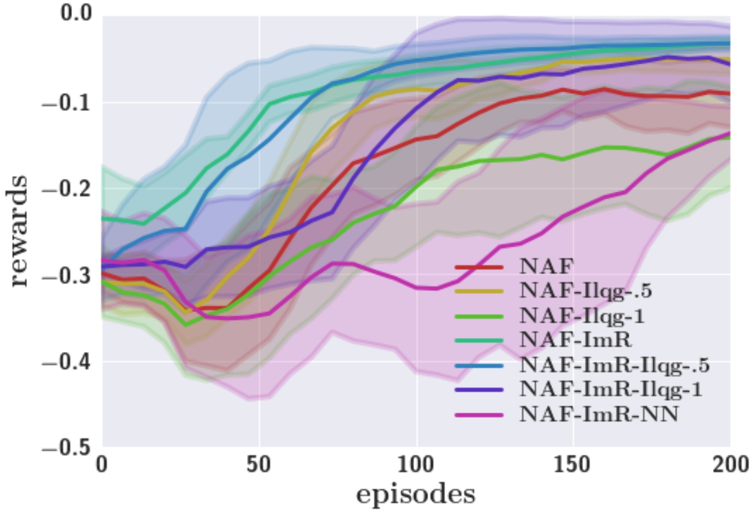
Benefit of Model-based Acceleration

$\mathcal{M}$ = True Dynamics





$\mathcal{M}$ = Fitted Dynamics



- iLQG real-world rollouts provide no significant improvement
  - Need to experience bad actions
- Faster convergence with Imagination rollouts

- Faster convergence with Imagination rollouts
- Most of the benefit of Model-based Acceleration in the beginning

Key Takeaways

- NAF as Q-Learning alternative to Actor-Critic methods in continuous action and state domains
  - ✓ Conceptually simpler than Actor-Critic
  - ✓ Mostly faster convergence than DDPG
  - ✓ Especially suited for high-precision tasks
- Leverage advantages of Model-based + Model-free methods with simulated experience
  - ✓ Need fewer real-world rollouts with Imagination rollouts
  - ✓ Good results with simple dynamics model
  - ✓ Combine the „best of both worlds“

Outlook

Schulman et al. 2017: **PPO**

- Simpler policy gradient method with better performance

Haarnoja et al. 2018: **SAC**

- More sample-efficient and stable Actor-Critic

Chebotar et al. 2017: **PILQR**

- Directly combine Model-based and Model-free updates

Critique of the Paper

Pros

- Very systematic analysis of different approaches
- Include unsuccessful results aswell
  - iLQG exploration has no substantial benefit

Cons

- NAF not simpler than Actor-Critic methods in practice
  - Needs 3 NNs instead of 2 for Actor-Critic
  - But all trained on same objective
- Confusing usage of the terms on-policy vs. off-policy
- Typo in algorithm
- Do not explain iLQG algorithm in detail

Thank you!

Exploration Policy

Discrete action spaces (ε-greedy):

$$\pi(a|s) = \begin{cases} \arg\max_a Q(s,a) & \text{with probability } 1-\varepsilon \\ \text{random action with probability } \varepsilon \end{cases}$$

Continuous action spaces:

$$\pi(a|s) = \arg\max_a Q(s,a) + \epsilon \qquad \epsilon \sim \mathcal{N}$$ (e.g. Ohrnstein-Uhlenbeck process)

NAF vs. DDPG

Domain	Description	Domain	Description
Cartpole	The classic cart-pole swing-up task. Agent must balance a pole attached to a cart by applying forces to the cart alone. The pole starts each episode hanging upside-down.	Reacher	Agent is required to move a 3-DOF arm from random starting locations to random target positions.
Peg	Agent is required to insert the tip of a 3-DOF arm from locally-perturbed starting locations to a fixed hole.	Gripper	Agent must use an arm with gripper appendage to grasp an object and manuver the object to a fixed target.
GripperM	Agent must use an arm with gripper attached to a moveable platform to grasp an object and move it to a fixed target.	Canada2d	Agent is required to use an arm with hockey-stick like appendage to hit a ball initialzed to a random start location to a random target location.
Cheetah	Agent should move forward as quickly as possible with a cheetah- like body that is constrained to the plane.	Swimmer6	Agent should swim in snake-like manner toward the fixed target using six joints, starting from random poses.
Ant	The four-legged ant should move toward the fixed target from a fixed starting position and posture.	Walker2d	Agent should move forward as quickly as possible with a bipedal walker constrained to the plane without falling down or pitching the torso too far forward or backward.

Domains	-	DDPG	episodes	NAF	episodes
Cartpole	-2.1	-0.601	420	-0.604	**190**
Reacher	-2.3	-0.509	1370	**-0.331**	**1260**
Peg	-11	-0.950	690	**-0.438**	**130**
Gripper	-29	1.03	2420	**1.81**	**1920**
GripperM	-90	-20.2	1350	**-12.4**	**730**
Canada2d	-12	-4.64	1040	**-4.21**	900
Cheetah	-0.3	**8.23**	**1590**	7.91	2390
Swimmer6	-325	-174	220	**-172**	**190**
Ant	-4.8	-2.54	2450	-2.58	**1350**
Walker2d	0.3	**2.96**	**850**	1.85	1530

Bellman Equation

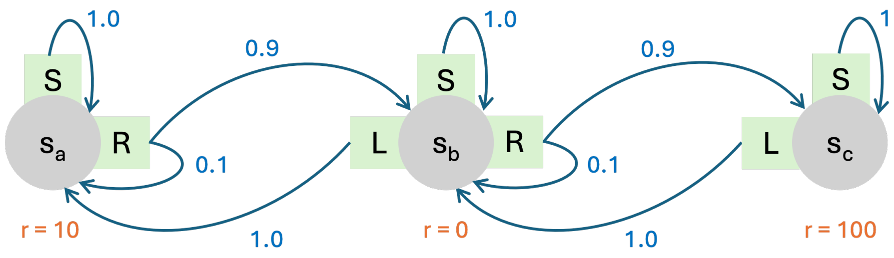
Over all states and actions in the trajectory

$$V^{\pi}(s_0) = \mathbb{E}_{a_0,s_1,a_1,\dots}\left[\sum_{t=0} \gamma^t r_t \,|s_0\right]$$

$$= \mathbb{E}\left[r_0 + \sum_{t=1} \gamma^t r_t \,|s_0\right]$$

Only depends on first action, state and starting state

$$= \sum_{a_0} \pi(a_0|s_0) \sum_{s_1} p(s_1|s_0,a_0)\left(r(s_0,a_0,s_1) + \overbrace{\gamma\mathbb{E}_{a_1,s_2,a_2,\dots}\left[\sum_{t=0}\gamma^t r_{t+1}\,|s_1\right]}^{V^{\pi}(s_1)}\right)$$



$$V^{\pi}(s_0) = \sum_{a_0} \pi(a_0|s_0) \overbrace{\sum_{s_1} p(s_1|s_0,a_0)\left(r(s_0,a_0,s_1) + \gamma V^{\pi}(s_1)\right)}^{Q^{\pi}(s_0,a_0)}$$

Bellman Optimality: $$V^{*}(s_0) = \max_a \left(\sum_{s_1} p(s_1|s_0,a)\left(r(s_0,a,s_1) + \gamma V^{*}(s_1)\right)\right)$$

Hyperparameters

Parameter	Value
Number of Episodes for Model Fitting n	5
Number of simulated steps l	$5, 10$
Batch size m	$?$
Number of updates I	5
Episode length T	$154?$
Number of Episodes M	$\sim 10^2 - 10^3$
Fraction of greedy rollouts p	$0.5, 0$