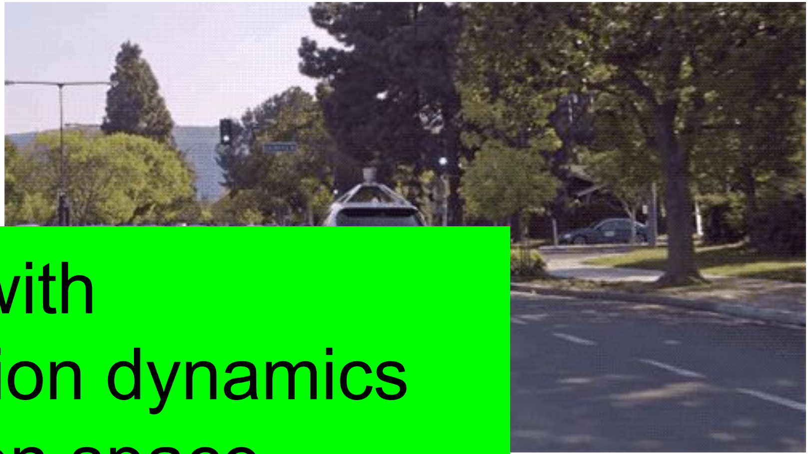
Continuous Control With Deep Reinforcement Learning

Timothy P. Lillicrap∗ , Jonathan J. Hunt∗ , Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra

Presenters:
Anqi (Joyce) Yang
Jonah Philion

Jan 21 2020

Robotics in 2020



Formalism: MDPs with

- Unknown transition dynamics
- Continuous action space

Can reinforcement learning solve robotics?

Alpha Go Zero (Silver et al, Nature, 2017)

selecting among possible moves for that piece. We represent the policy $\pi(a|s)$ by a $8 \times 8 \times 73$ stack of planes encoding a probability distribution over 4,672 possible moves. Each of the 8×8 positions identifies the square from which to "pick up" a piece. The first 56 planes encode

Dota 5 (OpenAI et al, 2019, https://cdn.openai.com/dota-2.pdf)

floats and categorical values with hundreds of possibilities) each time step. We discretize the action space; on an average timestep our model chooses among 8,000 to
pending on hero). For comparison Chess requires around one thousand valu

Alpha Star (Vinyals et al, Nature, 2019)

observe and act next (Fig. 1a). This representation of actio
in approximately 10^{26} possible choices at each step. Similar

Extended Data Table 2 | Agent action space

Field	Description
Action type	Which action to execute. Some examples of actions are moving a unit, training a unit from a building, moving the camera, or no-op. See PySC2 for a full list[7]
Selected units	Entities that will execute the action
Target	An entity or location in the map discretised to 256x256 targeted by the action
Queued	Whether to queue this action or execute it immediately
Repeat	Whether or not to issue this action multiple times
Delay	The number of game time-steps to wait until receiving the next observation

DDPG
(Lillicrap et al, 2015)

A first “Deep” crack at RL with continuous action spaces

Deterministic Policy Gradient

DPG (Silver et al., 2014)

- Finds deterministic policy
- Applicable to continuous action space

Deterministic Policy Gradient

DPG (Silver et al., 2014)

- Finds deterministic policy
- Applicable to continuous action space

- Not learning-based, can we do better?

DDPG

DDPG (Deep DPG) in one sentence:

- Extends **DPG** (Deterministic Policy Gradients, Silver et al., ‘14) using deep learning,

- borrowing tricks from **Deep Q-Learning** (Mnih et al., ‘13)

DDPG

DDPG (Deep DPG) in one sentence:

- Extends **DPG** (Deterministic Policy Gradients, Silver et al., '14) using deep learning,

- borrowing tricks from **Deep Q-Learning** (Mnih et al., '13)

- Contribution: model-free, off-policy, actor-critic approach that allows us to better learn deterministic policies on continuous action space

A Taxonomy of RL Algorithms

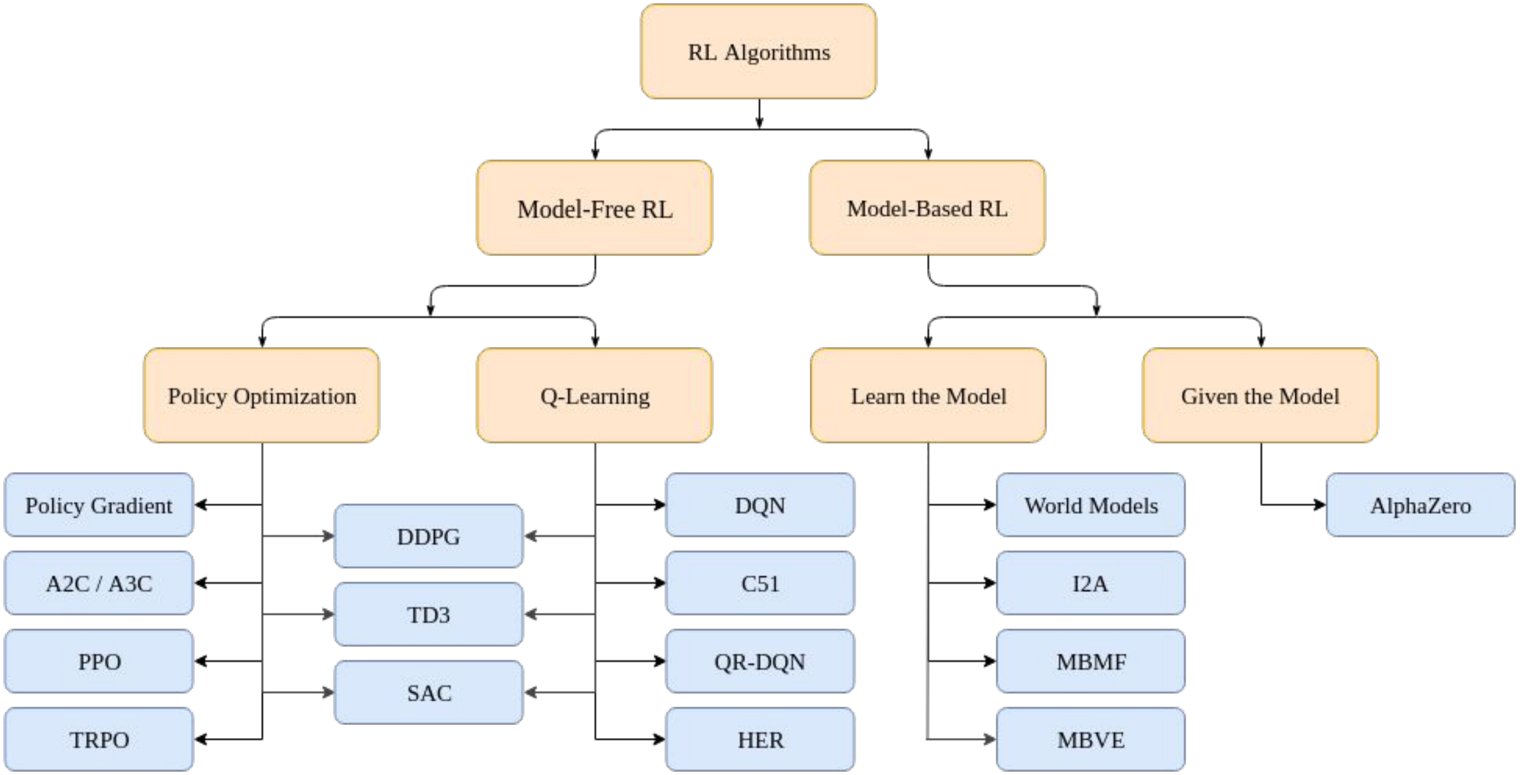


Image credit: OpenAI Spinning Up, https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#id20

A Taxonomy of RL Algorithms

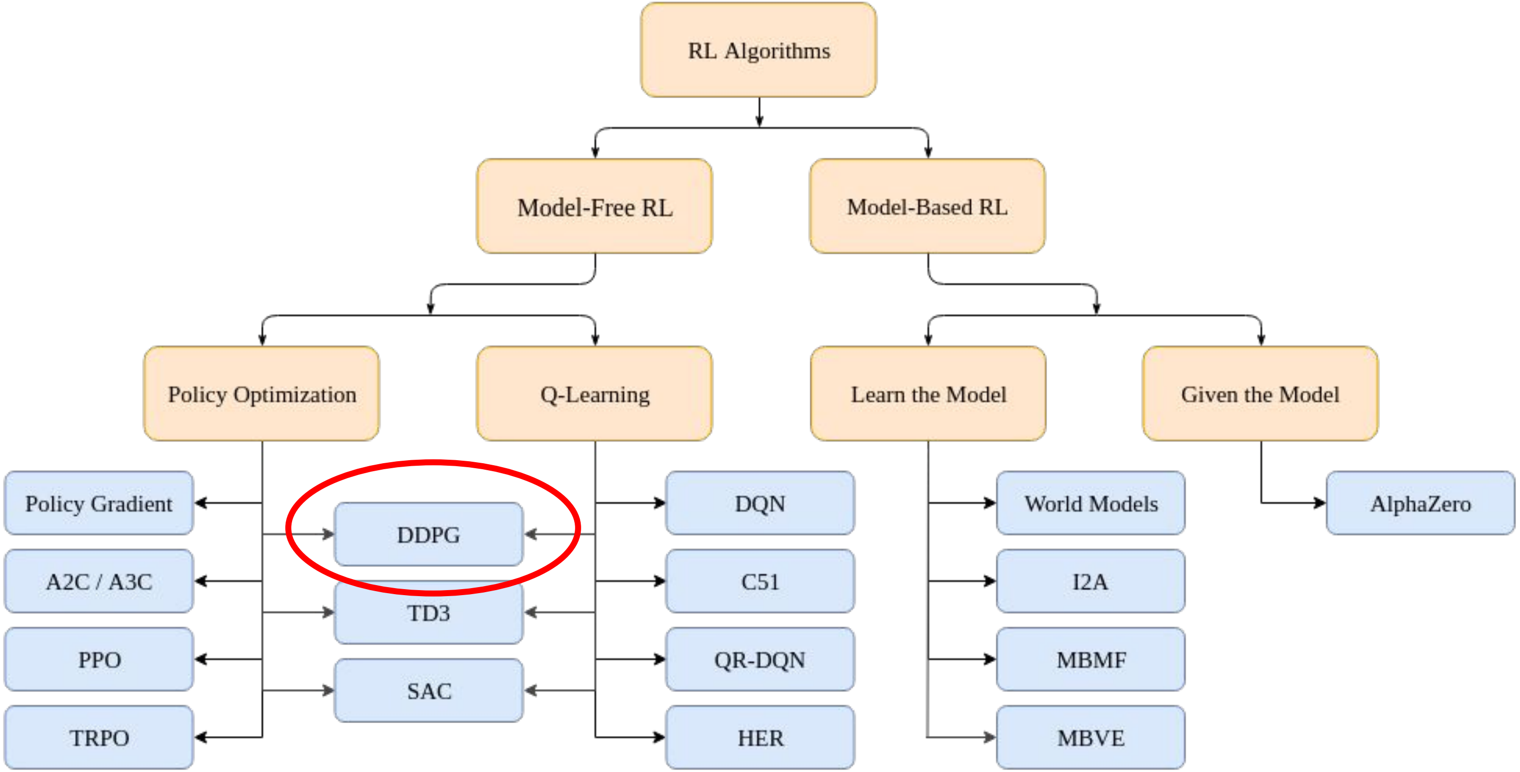


Image credit: OpenAI Spinning Up, https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#id20

DDPG

DDPG (Deep DPG) is a model-free, off-policy, actor-critic algorithm that combines:

- **DPG** (Deterministic Policy Gradients, Silver et al., ‘14): works over continuous action domain, not learning-based

- **DQN** (Deep Q-Learning, Mnih et al., ‘13): learning-based, doesn’t work over continuous action domain

Background - DPG

Background - DPG

Background - DPG

In Q-learning, we find deterministic policy by

$$\mu^{k+1}(s) = \arg\max_{a} Q^{\mu^k}(s, a)$$

Background - DPG

In Q-learning, we find deterministic policy by

$$\mu^{k+1}(s) = \boxed{\arg\max_a} Q^{\mu^k}(s, a)$$

Problem: In large discrete action space or continuous action space, we can't plug in every possible action to find the optimal action!

Background - DPG

In Q-learning, we find deterministic policy by

$$\mu^{k+1}(s) = \boxed{\arg\max_a} Q^{\mu^k}(s, a)$$

Problem: In large discrete action space or continuous action space, we can't plug in every possible action to find the optimal action!

Solution: Learn a function approximator for argmax, via gradient descent

$$\mu^{k+1}(s) = \pi_{\boxed{\theta}}(s)$$

Background - DPG

- Goal:

  Derive a gradient update rule to learn deterministic policy π_θ

Background - DPG

- Goal:

  Derive a gradient update rule to learn deterministic policy π_θ

- Idea:

  Adapt the **stochastic policy gradient** formulation for deterministic policies

Background - DPG

- Vanilla Stochastic Policy Gradient:

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient:

$$\theta^{\star} = \arg\max_{\theta} \underbrace{E_{\tau \sim p_{\theta}(\tau)} \left[\sum_{t} r(\mathbf{s}_t, \mathbf{a}_t) \right]}_{J(\theta)}$$

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient:

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient:

$$\theta^{\star} = \arg\max_{\theta} \underbrace{E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]}_{J(\theta)}$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient:

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

model-free

Not trivial to compute!

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient with Monte-Carlo Sampling:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient with Monte-Carlo Sampling:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \boxed{\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)}$$

Problem: Point Estimate - High Variance!

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$
$$= \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log_{\pi_\theta}(a|s) Q^{\pi_\theta}(s, a)]$$

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t,\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t)\right)\right]$$
$$= \mathbb{E}_{s\sim\rho^\pi,a\sim\pi_\theta}[\nabla_\theta \log_{\pi_\theta}(a|s)\boxed{Q^{\pi_\theta}(s,a)}]$$

True value function is still not trivial to compute

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Vanilla Stochastic Policy Gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

$$= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log_{\pi_\theta}(a|s)\boxed{Q^{\pi_\theta}(s,a)}]$$

True value function is still not trivial to compute, but we can approximate it with a parameterized function:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\boxed{w}}(s,a)\right]$$

Source: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf

Background - DPG

- Stochastic Policy Gradient (Actor-Critic)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^w(s,a)\right]$$

Background - DPG

- Stochastic Policy Gradient (Actor-Critic)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^w(s,a)\right]$$

Actor: Policy function π_θ

Background - DPG

- Stochastic Policy Gradient (Actor-Critic)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^w(s,a)\right]$$

Actor: Policy function π_θ

Critic: Value function Q^w, which provides guidance to improve the actor

Background - DPG

- Deterministic Policy Gradient (Actor-Critic)

Background - DPG

- Deterministic Policy Gradient (Actor-Critic)

Objective:

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) r(s, \pi_\theta(s)) ds$$
$$= \mathbb{E}_{s \sim \rho^\pi}[r(s, \pi_\theta(s))]$$

Background - DPG

- Deterministic Policy Gradient (Actor-Critic)

Objective:

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) r(s, \pi_\theta(s)) ds$$
$$= \mathbb{E}_{s \sim \rho^\pi}[r(s, \pi_\theta(s))]$$

Policy Gradient:

$$\nabla_\theta J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \nabla_\theta \pi_\theta(s) \nabla_a Q^w(s, a)|_{a=\pi_\theta(s)} ds$$
$$= \mathbb{E}_{s \sim \rho^\pi}[\nabla_\theta \pi_\theta(s) \nabla_a Q^w(s, a)|_{a=\pi_\theta(s)}]$$

Background - DPG

Deterministic Policy Gradient (Actor-Critic)

Objective:

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) r(s, \pi_\theta(s)) ds$$
$$= \mathbb{E}_{s \sim \rho^\pi} \left[\boxed{r(s, \pi_\theta(s))} \right]$$

Policy Gradient:

$$\nabla_\theta J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \nabla_\theta \pi_\theta(s) \nabla_a \boxed{Q^w(s,a)|_{a=\pi_\theta(s)}} ds$$
$$= \mathbb{E}_{s \sim \rho^\pi} \left[\nabla_\theta \pi_\theta(s) \nabla_a Q^w(s,a)|_{a=\pi_\theta(s)} \right]$$

Background - DPG

Stochastic Policy Gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi,\boxed{a\sim\pi_\theta}}[\nabla_\theta \log \pi_\theta(a|s) Q^w(s,a)]$$

Deterministic Policy Gradient:

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

Background - DPG

Stochastic Policy Gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi,\boxed{a\sim\pi_\theta}}[\nabla_\theta \log \pi_\theta(a|s) Q^w(s,a)]$$

Deterministic Policy Gradient:

DDPG: Use deep learning to learn both functions!

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

Background - DQN

Background - DQN

How do we learn a value function with deep learning?

Background - DQN

How do we learn a value function with deep learning?

Q-Learning:

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - Q(S,A) \right]$$

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

How do we learn a value function with deep learning?

Q-Learning:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - Q(S, A) \right]$$

Parameterize Q with a neural network:

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

How do we learn a value function with deep learning?

Q-Learning:

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left[\boxed{R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')} - Q(S,A) \right]$$

Parameterize Q with a neural network:

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

How do we learn a value function with deep learning?

Q-Learning:

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left[\boxed{R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')} - Q(S,A) \right]$$

Parameterize Q with a neural network:

$$t \leftarrow r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha (t - Q(\mathbf{s}, \mathbf{a})) \nabla_{\boldsymbol{\theta}} Q(\mathbf{s}_t, \mathbf{a}_t).$$

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

How do we learn a value function with deep learning?

Q-Learning:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[\boxed{R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')} - Q(S, A) \right]$$

Parameterize Q with a neural network:

Problem: t is parameterized by theta too! Moving target

$$t \leftarrow r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}} \boxed{Q(\mathbf{s}_{t+1}, \mathbf{a})}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(t - Q(\mathbf{s}, \mathbf{a}))\nabla_{\boldsymbol{\theta}} Q(\mathbf{s}_t, \mathbf{a}_t).$$

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

How do we learn a value function with deep learning?

Q-Learning:

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left[\boxed{R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')} - Q(S,A) \right]$$

Parameterize Q with a neural network:

Solution: Use a "target" network with frozen params

$$t \leftarrow r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}} \boxed{Q(\mathbf{s}_{t+1}, \mathbf{a})}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(t - Q(\mathbf{s}, \mathbf{a}))\nabla_{\boldsymbol{\theta}} Q(\mathbf{s}_t, \mathbf{a}_t).$$

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

Deep Q-Learning:

Trick #1: Use a **target network**

3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\boxed{\phi'}}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ': copy ϕ every N steps

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

Another problem: Sample Inefficiency

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

Another problem: Sample Inefficiency

Trick #2: Use a **replay buffer** to store past transitions and rewards

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ': copy ϕ every N steps

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background - DQN

Another problem: Sample Inefficiency

Trick #2: Use a **replay buffer** to store past transitions and rewards

Replay buffer also allows the algorithm to be **off-policy**, since we are sampling from the buffer instead of sampling a new trajectory according to current policy each time

Note that this trick is only possible with deterministic policies

Source: http://www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/slides/lec10-slides.pdf

Background Summary

- DPG: Formulates an update rule for deterministic policies, so that we can learn deterministic policy on continuous action domain

Background Summary

- DPG: Formulates an update rule for deterministic policies, so that we can learn deterministic policy on continuous action domain

- DQN: Enables learning value functions with neural nets , with two tricks:
  - Target Network
  - Replay Buffer

Background Summary

- DPG: Formulates an update rule for deterministic policies, so that we can learn deterministic policy on continuous action domain

  Model-Free, Actor-Critic

- DQN: Enables learning value functions with neural nets , with two tricks:
  - Target Network
  - Replay Buffer - Off-Policy

Background Summary

- DPG: Formulates an update rule for deterministic policies, so that we can learn deterministic policy on continuous action domain

  Model-Free, Actor-Critic

- DQN: Enables learning value functions with neural nets , with two tricks:
  - Target Network
  - Replay Buffer - Off-Policy

- DDPG: Learn both the policy and the value function in DPG with neural networks, with DQN tricks!

Method - DDPG

DDPG Problem Setting

DDPG Problem Setting

$$\mu(s|\theta^{\mu})$$

Policy (Actor) Network
Deterministic, Continuous Action Space

DDPG Problem Setting

$$\mu(s|\theta^\mu)$$

Policy (Actor) Network
Deterministic, Continuous Action Space

$$Q(s, a|\theta^Q)$$

Value (Critic) Network

DDPG Problem Setting

$$\mu(s|\theta^{\mu})$$

Policy (Actor) Network
Deterministic, Continuous Action Space

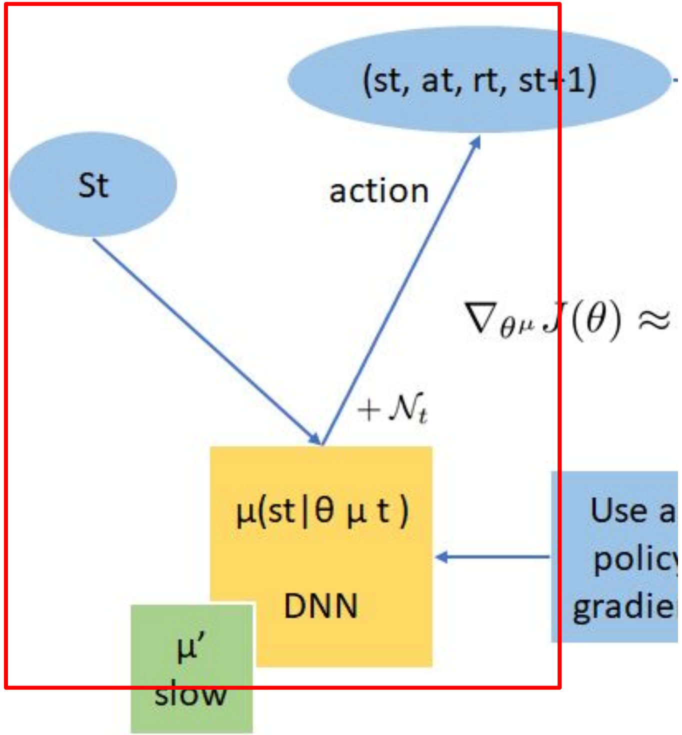
$$Q(s, a|\theta^{Q})$$

Value (Critic) Network

$$\mu'(s|\theta^{\mu'}), Q'(s, a|\theta^{Q'})$$

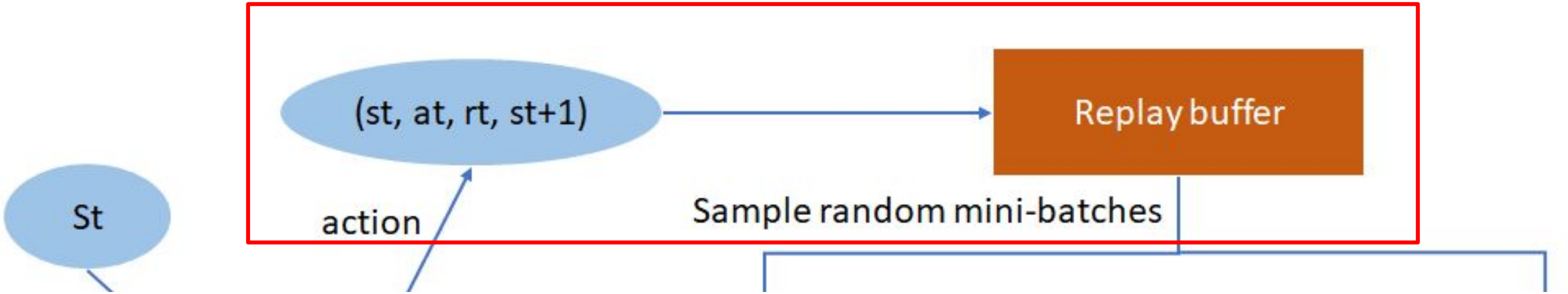
Target Policy and Value Networks

Method



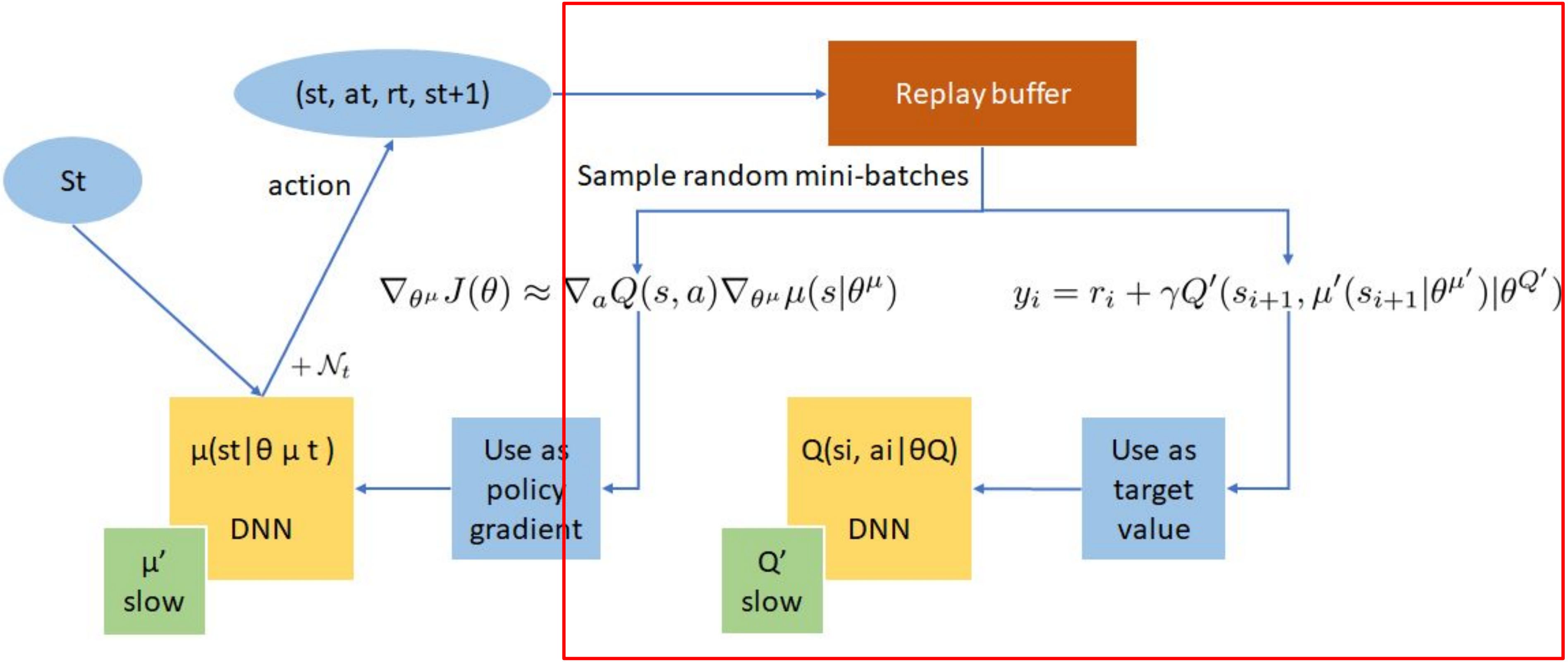
Credit: Professor Animesh Garg

Method



Credit: Professor Animesh Garg

Method



Credit: Professor Animesh Garg

Method

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**
    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
    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:

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

    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

Method

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**
    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
    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:

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

    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

Replay buffer

“Soft” target network update

Method

Add noise for exploration

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**
    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
    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:

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

    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

Method

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**
    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
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

Value Network Update

    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:

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

    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

Method

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**
    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
    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:

Policy Network Update

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

    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

Method

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Method

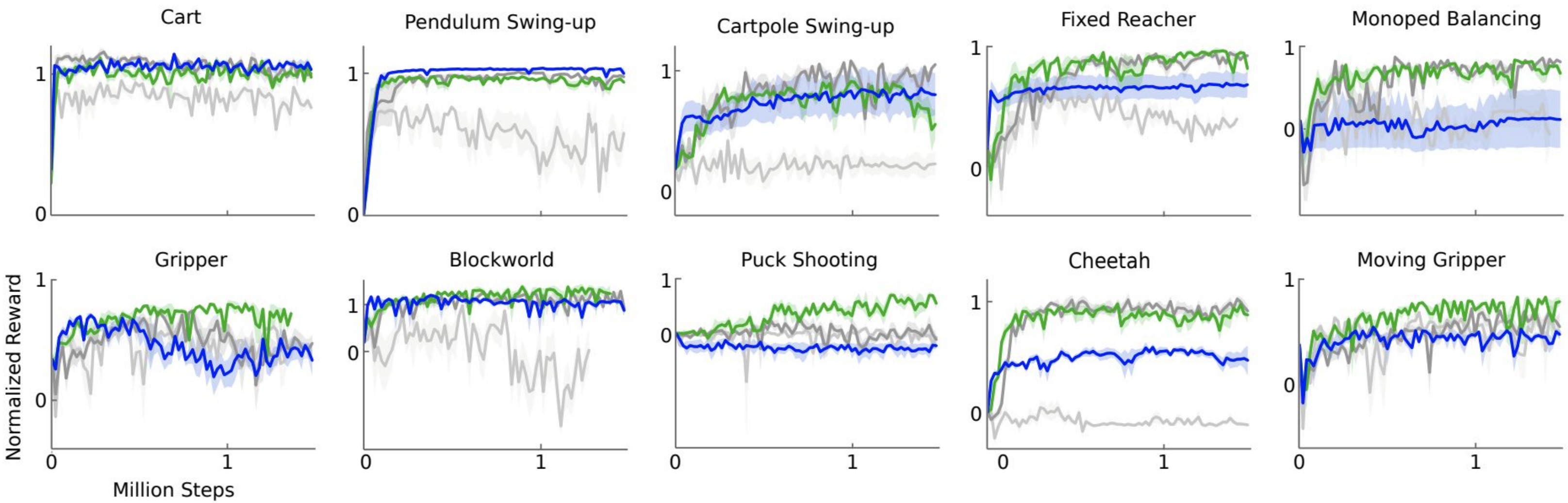
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**

DDPG: Policy Network, learned with Deterministic Policy Gradient

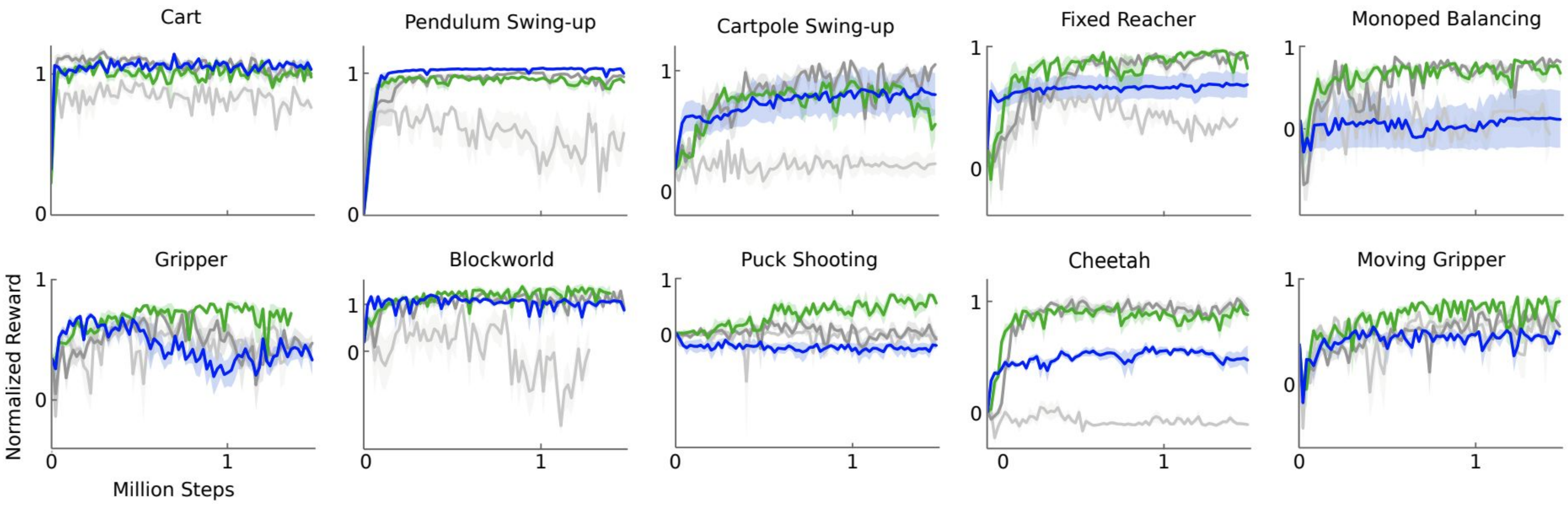
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Experiments



Light Grey: Original DPG
Dark Grey: Target Network
Green: Target Network + Batch Norm
Blue: Target Network from pixel-only inputs

Experiments Do target networks and batch norm matter?



Light Grey: Original DPG
Dark Grey: Target Network
Green: Target Network + Batch Norm
Blue: Target Network from pixel-only inputs

Experiments

Is DDPG better than DPG?

	DDPG				DPG	
environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Experiments

Is DDPG better than DPG?

	DDPG				DPG	
environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Experiments

DDPG DPG

Is DDPG better than DPG?

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Experiments

DDPG DPG

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Is DDPG better than DPG?

Experiments

DDPG DPG

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Is DDPG better than DPG?

0: random policy

1: planning-based policy

Experiments

DDPG DPG

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

DDPG still exhibits high variance

Experiments

How well does Q estimate the true returns?

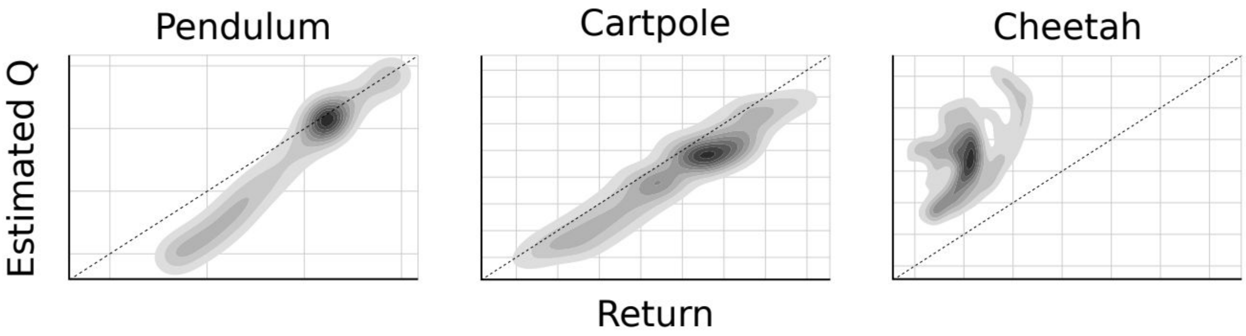


Figure 3: Density plot showing estimated Q values versus observed returns sampled from test episodes on 5 replicas. In simple domains such as pendulum and cartpole the Q values are quite accurate. In more complex tasks, the Q estimates are less accurate, but can still be used to learn competent policies. Dotted line indicates unity, units are arbitrary.

Discussion of Experiment Results

- Target Networks and Batch Normalization are crucial
- DDPG is able to learn tasks over continuous domain, with better performance than DPG
- Q values estimated are quite accurate (compared to the true expected reward) in simple tasks

Discussion of Experiment Results

- Target Networks and Batch Normalization are crucial

- DDPG is able to learn tasks over continuous domain, with better performance than DPG, but the variance in performance is still pretty high

- Q values estimated are quite accurate (compared to the true expected reward) in simple tasks, but not so accurate for more complicated tasks

Toy example

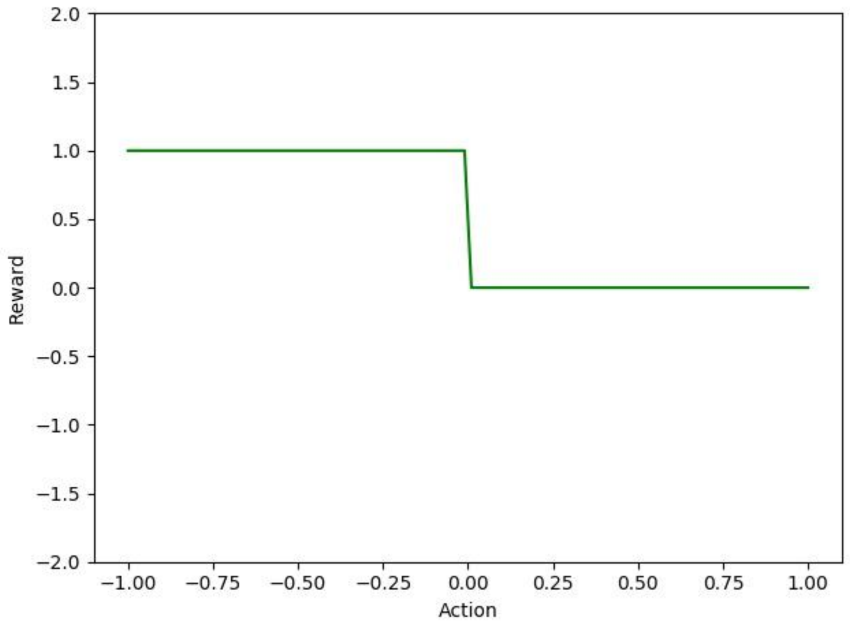
What can we say about Q*(a) in this case?

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

$$Q^*(a) = r(a)$$

Consider the following MDP:

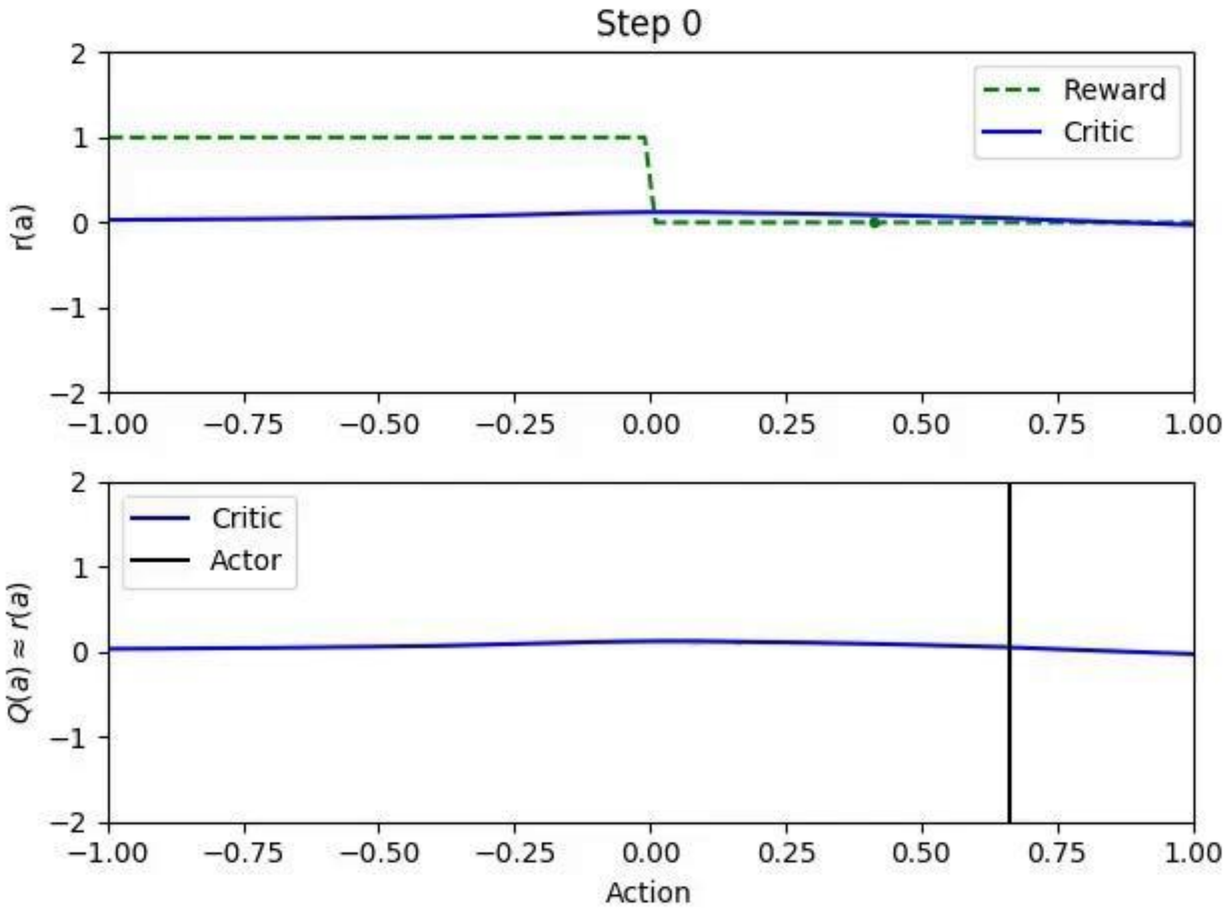
1. Actor chooses action $-1<a<1$
2. Receives reward 1 if action is negative, 0 otherwise



DDPG

Critic Perspective

Actor Perspective



Why did this work?

- What is the ground truth deterministic policy gradient?

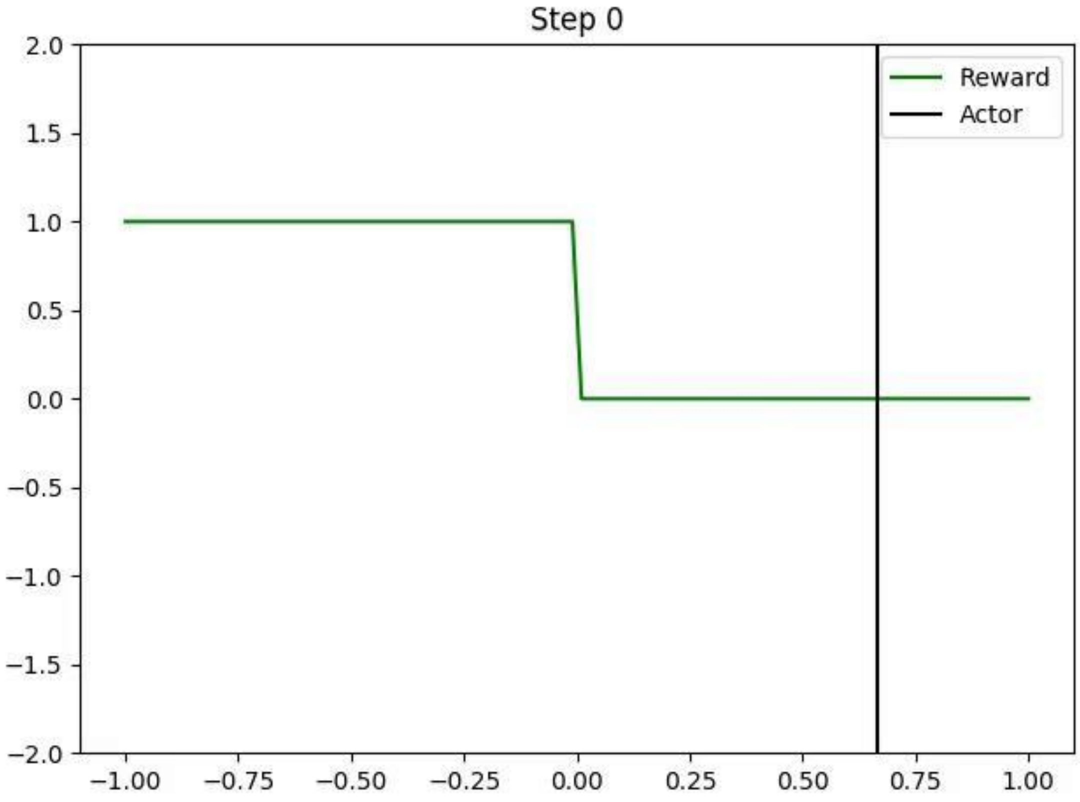
$$Q^*(a) = r(a)$$

0

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t\sim\rho^\beta}\left[\boxed{\nabla_a Q(s,a|\theta^Q)}|_{s=s_t,a=\mu(s_t)} \nabla_{\theta_\mu}\mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned}$$

=> The true DPG is **0** in this toy problem!

Gradient Descent on Q* (true policy gradient)



A Closer Look At Deterministic Policy Gradient

Claim: If in a finite-time MDP

- State space is continuous
- Action space is continuous
- Reward function r(s, a) is piecewise constant w.r.t. s and a
- Transition dynamics are deterministic and differentiable

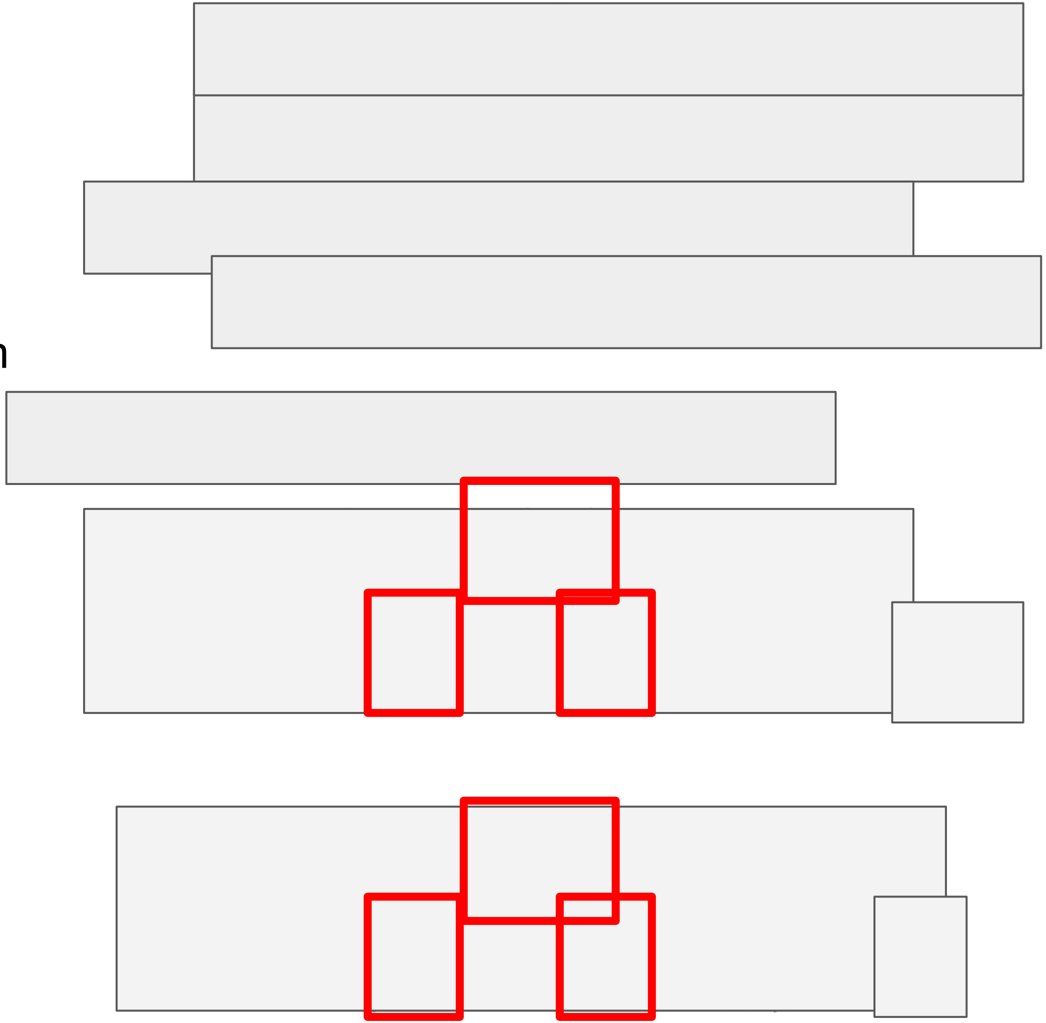
=> Then Q* is also piecewise constant and the DPG is **0**.

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

Quick proof: Induct on steps from terminal state

Base case n=0 (aka s is terminal):
Q*(s,a) = r(s,a)
=> Q*(s,a) is piecewise constant in for s terminal because r(s,a) is.

Inductive step:
assume true for states **n-1** steps from terminating and proof for states **n** steps from terminating



If the dynamics are deterministic and the reward function is discrete =>

Deterministic Policies have 0 gradient

(monte carlo estimates become equivalent to random walk)

DDPG Follow-up

- Model the actor as the argmax of a convex function
  - Continuous Deep Q-Learning with Model-based Acceleration (Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, Sergey Levine, ICML 2016)
  - Input Convex Neural Networks (Brandon Amos, Lei Xu, J. Zico Kolter, ICML 2017)
- Q-value overestimation
  - Addressing Function Approximation Error in Actor-Critic Methods (TD3) (Scott Fujimoto, Herke van Hoof, David Meger, ICML 2018)
- Stochastic policy search
  - Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor (Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, ICML 2018)

often used to perform the maximization in Q-learning. A commonly used algorithm in such settings, deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), provides for sample-efficient learning but is notoriously challenging to use due to its extreme brittleness and hyperparameter sensitivity (Duan et al., 2016; Henderson et al., 2017).

ily to very complex, high-dimensional tasks, such as the Humanoid benchmark (Duan et al., 2016) with 21 action dimensions, where off-policy methods such as DDPG typically struggle to obtain good results (Gu et al., 2016). SAC also avoids the complexity and potential instability associ-

A cool application of DDPG: Wayve

Learning to Drive in a Day (Alex Kendall et al, 2018)

We selected a simple continuous action domain model-free reinforcement learning algorithm: deep deterministic policy gradients (DDPG) [8], to show that an off-the-shelf reinforcement learning algorithm with no task-specific adaptation is capable of solving the MDP posed in Section III-A.

DDPG consists of two function approximators: a critic $Q\colon \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, which estimates the value $Q(s, a)$ of the expected cumulative discounted reward upon using action a in state s, trained to satisfy the Bellman equation

$$Q(s_t, a_t) = r_{t+1} + \gamma(1 - d_t)Q(s_{t+1}, \pi(s_{t+1})),$$

under a policy given by the actor $\pi\colon \mathcal{S} \rightarrow \mathcal{A}$, which attempts to estimate a Q-optimal policy $\pi(s) = \text{argmax}_a Q(s, a)$; here $(s_t, a_t, r_{t+1}, d_{t+1}, s_{t+1})$ is an experience tuple, a transition

Conclusion

- DDPG = DPG + DQN
- Big Idea is to bypass finding the local max of Q in DQN by jointly training a second neural network (actor) to predict the local max of Q.
- Tricks that made DDPG possible:
  - Replay buffer, target networks (from DQN)
  - Batch normalization, to allow transfer between different RL tasks with different state scales
  - Directly add noise to policy output for exploration, due to continuous action domain
- Despite these tricks, DDPG can still be sensitive to hyperparameters. TD3 and SAC offer better stability.

Questions

1. Write down the deterministic policy gradient.
    a. Show that for gaussian action, REINFORCE reduces to DPG as sigma->0

2. What tricks does DDPG incorporate to make learning stable?

Thank you!

Joyce, Jonah

Motivation and Main Problem

1-4 slides

Should capture

- High level description of problem being solved (can use videos, images, etc)
- Why is that problem important?
- Why is that problem hard?
- High level idea of why prior work didn’t already solve this (Short description, later will go into details)

Contributions

Approximately one bullet, high level, for each of the following (the paper on 1 slide).

- Problem the reading is discussing

- Why is it important and hard

- What is the key limitation of prior work

- What is the key insight(s) (try to do in 1-3) of the proposed work

- What did they demonstrate by this insight? (tighter theoretical bounds, state of the art performance on X, etc)

General Background

1 or more slides

The background someone needs to understand this paper

That wasn’t just covered in the chapter/survey reading presented earlier in class during same lecture (if there was such a presentation)

Problem Setting

1 or more slides

Problem Setup, Definitions, Notation

Be precise-- should be as formal as in the paper

Algorithm

Likely >1 slide

Describe algorithm or framework (pseudocode and flowcharts can help)

What is it trying to optimize?

Implementation details should be left out here, but may be discussed later if its relevant for limitations / experiments

Experimental Results

>=1 slide

State results

Show figures / tables / plots

Discussion of Results

>=1 slide

What conclusions are drawn from the results?

Are the stated conclusions fully supported by the results and references? If so, why? (Recap the relevant supporting evidences from the given results + refs)

Critique / Limitations / Open Issues

1 or more slides: What are the key limitations of the proposed approach / ideas? (e.g. does it require strong assumptions that are unlikely to be practical? Computationally expensive? Require a lot of data? Find only local optima?)

- If follow up work has addressed some of these limitations, include pointers to that. But don't limit your discussion only to the problems / limitations that have already been addressed.

Contributions / Recap

Approximately one bullet for each of the following (the paper on 1 slide)

- Problem the reading is discussing

- Why is it important and hard

- What is the key limitation of prior work

- What is the key insight(s) (try to do in 1-3) of the proposed work

- What did they demonstrate by this insight? (tighter theoretical bounds, state of the art performance on X, etc)

“Deep” Q learning

Not directly applicable to continuous action space!

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

O(A) if discrete, O(n forward passes of Q) otherwise

O(A) if discrete, O(n forward passes of Q) otherwise

Mnih, 2013, https://arxiv.org/pdf/1312.5602.pdf