Continuous Control With Deep Reinforcement Learning

Timothy P. Lillicrap∗ , Jonathan J. Hunt∗ , Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa,

David Silver & Daan Wierstra

Google DeepMind

ICML 2016

Presented by Sanghyeon Lee

Motivation

Problem

1. While DQN solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces
2. Large, non-linear function approximators for Q function has often been avoided since theoretical performance guarantees are impossible

➔ ***Using Deep Deterministic Policy Gradient method***

Method

Notation

E: An Environment in discrete timesteps

x_i: An observation received at each timestep t

$R_t = \sum_{\mathrm{k=t}}^{T} \gamma^{k-t} r(s_k, a_k)$: Total Discount Return

$V^{\pi}(s) = E[R_t | S_1 = s; \pi]$: Value function, Expected total discounted reward

$Q^{\pi}(s, a) = E[R_t | S_1 = s, A_1 = a; \pi$: Action Value Function

$\mu_{\theta}(s)$: Deterministic target policy

μ': Different behavior policy (in off-policy setting)

Method

Review & Background

*Bellman Equation

$$Q^{\pi}(s_t, a_t) = \mathrm{E}_{\mathrm{r_t, s_{t+1}} \sim \mathrm{E}}[r(s_t, a_t) + \gamma \mathrm{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]]$$

*Bellman Equation with Deterministic policy

$$Q^{\pi}(s_t, a_t) = \mathrm{E}_{\mathrm{r_t, s_{t+1}} \sim \mathrm{E}}[r(s_t, a_t) + \gamma \mathrm{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, \mu(s_{t+1})]]$$

*Critic Loss Function

$$L(\theta^{Q}) = \mathrm{E}_{\mu'}[(Q_{\theta}(s_t, a_t) - y_t)^2], \qquad s.t\ \ y_t = r(s_t, a_t) + \gamma Q_{\theta}(s_{t+1}, \mu(s_{t+1}))$$

Directly implement Q learning with $L(\theta^{Q})$ to be unstable. Q is also updated by target value y_t is also function of parameter θ, but it is typically ignored

*Off-Policy Deterministic Actor-Critic (OPDAC)

$$J_{\beta}(\mu_{\theta}) = \int_{S} \rho^{\beta} V^{\mu}(s) ds = \int_{S} \rho^{\beta}(s) Q^{\mu}(s, \mu_{\theta}(s)) ds$$

$$\nabla_{\theta} J_{\beta}(\mu_{\theta}) \approx \int_{S} \rho^{\beta}(s) \nabla_{\theta}\mu_{\theta}(a|s) Q^{\mu}(s, a) ds = \mathrm{E}_{s \sim \rho^{\beta}}[\nabla_{\theta}\mu_{\theta}(s) \nabla_{\mathrm{a}}\, \mathrm{Q}^{\mu}(s, a)|_{a=\mu_{\theta}(s)}]$$

$$\delta_t = r_t + \gamma Q^{w}(s_{t+1}, a_{t+1}) - Q^{w}(s_t, a_t)$$
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^{w}(s_t, a_t)$$
$$\theta_{t+1} = \theta_t + \alpha_{\theta} \nabla_{\theta}\mu_{\theta}(s_t)\ \nabla_a Q^{w}(s_t, a_t)|_{a=\mu_{\theta}(s)}$$

Method

Algorithm

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 gradient:

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

    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

Update Actor

$$\nabla_\theta \mu_\theta \approx \mathrm{E}_{s\sim\mu'}\left[\nabla_\theta Q\left(\mathrm{s},\mathrm{a}\middle|\theta^{Q}\right)\right|_{\mathrm{s}=\mathrm{s_t},\mathrm{a}=\mu_\theta(\mathrm{s_t})} = \mathrm{E}_{s\sim\mu'}\left[\nabla_\theta Q\left(\mathrm{s},\mathrm{a}\middle|\theta^{Q}\right)\right|_{\mathrm{s}=\mathrm{s_t},\mathrm{a}=\mu_\theta(\mathrm{s_t})}$$

Review – Deterministic Policy Gradient Theorem

$$J(\mu_\theta) = \int_S \rho^\mu(s) r\big(s,\mu_\theta(s)\big) ds = \mathrm{E}_{s\sim\rho^\mu}[\mathrm{r}(\mathrm{s},\mu_\theta(s))]$$

$$\nabla_\theta J(\mu_\theta) = \int_S \rho^\mu(s)\nabla_\theta(s)\nabla_a Q^\mu(s,a)\Big|_{a=\mu_\theta(s)} ds = \mathrm{E}_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s)\nabla_\mathrm{a} \mathrm{Q}^\mu(\mathrm{s},\mathrm{a})\Big|_{\mathrm{a}=\mu_\theta(s)}]$$

Behavior policy

$$\mu'(s_t) = \mu_{\theta_t^\mu}(s_t) + N\ , N\ is\ a\ noise\ process$$

Noise makes exploration on continuous action spaces

'Soft' target update

$$\theta' = \tau\theta + (1-\tau)\theta'$$

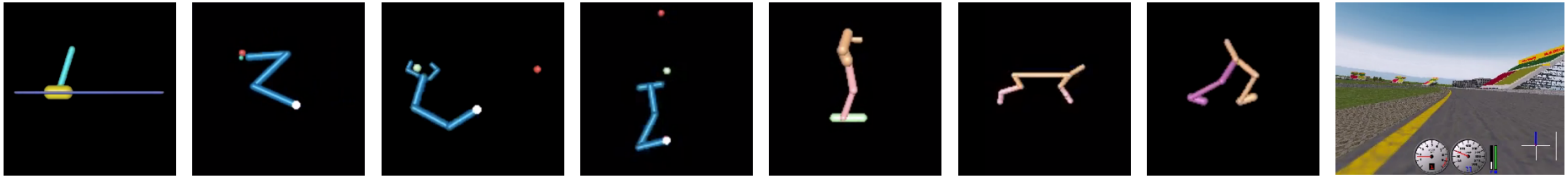
Target values are constrained to change slowly, greatly improving the stability of learning

Experiments

Method

1. Low-dimensional state and high-dimensional state
2. For each timestep of agent, repeating the agent's action and simulate 3 timesteps
   -3*RGB ➔ 9 feature maps, Normalize RGB value into [0,1]
3. Evaluate the policy periodically (In inference, remove the noise $\mathcal{N}$)
4. Ablation study for target value and batch normalization
5. DPG vs DDPG
6. Q-Learning vs DDPG

Results



Experiments

Results

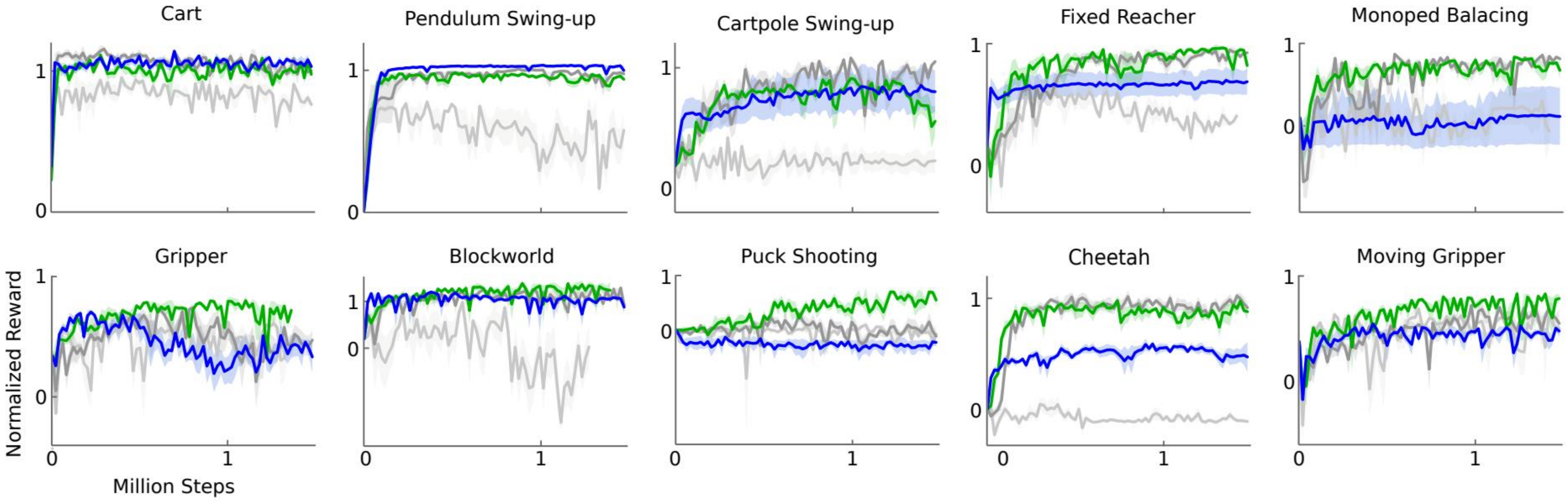


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.

Experiments

Results

ddpg — dpg with replay buffer and batchnorm

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

Normalize the score using two baselines

-Naïve policy (sample action from a uniform distribution)

➔ Mean: 0

-iLQG(Todrov & Li, 2005) ~ planning based solver

➔ Mean:1

Discussion

1. DDPG algorithm robustly solve the challenging problems across a variety of domains, even when using raw pixels for observations
2. Experimental results demonstrate that stable learning without the need for any modifications between environments
3. DDPG needs fewer steps of experience than was used by DQN learning to fund solutions in the Atari domain
4. DDPG requires a large number training episodes to find solutions