**CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING**

<https://arxiv.org/pdf/1509.02971.pdf>

**0. INTRODUCTION**

|  |
| --- |
| DQN은 continuous domain에서 적용하기 어렵다.   * DQN을 continuous domain에 적용시키기 위해서는 **action space를 discretize하면 되는데, 여기에는 많은 제약 사항**이 있다.   + **Curse of Dimensionality:** 예를 들어, freedom system이 7 degree이고 각 차원의 값이 {-k, 0, k} 중 하나이면 총 경우의 수는 2187가지가 된다.   여기서는 **model-free하고, off-policy한 actor-critic 알고리즘**을 이용한다. 이 알고리즘은 high-dimensional하고 연속적인 action space에 적용할 수 있다.   * **DPG (deterministic policy gradient)** 알고리즘에 기반한다. |

**1. BACKGROUND**

|  |
| --- |
| Action-value function은 많은 reinforcement learning algorithm에서 사용될 수 있는데, 이것은 **state 에서 action 를 실행했을 때의 expected return**을 설명한다. 여기서 **정책** 는 다음과 같이 나타낸다.  Reinforcement Learning에서의 많은 접근 방법은 다음과 같은 **recursive relationship**을 이용한다.  Target policy가 deterministic하면 이것을 다음과 같이 로 나타낼 수 있다.  Q-learning은 흔히 이용되는 off-policy 알고리즘이고, greedy policy 를 이용한다. 여기서는 **로 parameterize**된 function approximator를 생각한다.  따라서 역시 에 dependent하며, 이것은 일반적으로 ignore된다. |

**1. BACKGROUND (Cont.)**

|  |
| --- |
| 이때 **크고 non-linear한 function approximator**를 이용하여 value 또는 action-value function을 학습하는 것은 기피되어 왔으며, 이는 **이론적인 성능의 보장이 불가능**하기 때문이다.   * 최근에는 **Q-learning 알고리즘**이 큰 neural network의 사용을 효율적으로 하는 function approximator로 사용되고 있다. * Q-learning 알고리즘은 **Atari game을 pixel 단위로 플레이**할 수 있다. * **Replay buffer**와 **separate target network**를 이용하여 를 계산하는 방법이 도입되었다. * 여기서는 **DDPG (Deep Deterministic Policy Gradient)** 개념을 이용하고자 한다. |

**2. ALGORITHM**

|  |
| --- |
| Q-learning을 연속적인 action space에 바로 적용하는 것은 불가능하며, 이는 **연속적인 space에서 greedy policy를 찾는 것**은 **모든 timestep에서 의 최적화를 요구**하기 때문이다.   * DPG 알고리즘은 parameterized **actor function** 를 유지관리하고, 이것은 현재 policy를 **state를 특정한 action으로 deterministic하게 mapping**시키는 방법으로 specify한다. * **Critic function Q(s, a)**는 **Q learning의 Bellman Equation**을 이용하여 학습된다. * Actor는 다음과 같은 chain rule을 이용하여 update된다.   Q learning과 함께 **non-linear function approximator**를 도입하는 것은 convergence가 더 이상 보장되지 않는 것을 의미한다. 그러나 이러한 접근은 **large state space를 학습하고 일반화하기 위해 essential**하다.  이 논문의 기여는 DPG (Deterministic Policy Gradient)를 DQN을 기반으로 수정하여, **neural network function approximator가 큰 action space에서 학습**할 수 있도록 하는 것이다. |

**3. RESULTS**

|  |
| --- |
| **DDPG 알고리즘의 성능**은 각 환경에서 다음과 같으며, 5 replicas의 평균이다.    모든 task에서 **low-dimensional state description과 high-dimensional rendering을 둘 다 이용하여 실험을 진행**하였다.   * DQN에서는 문제를 fully observable하게 하기 위해서 **action repeat**을 사용했다. * Agent의 각 timestep에서 **simulation을 3 timestep 동안 진행**하였다. * observation은 **9개의 feature map**을 갖는다. (3개의 rendering의 RGB 값)   **exploration noise 없이 테스트**하기 위해서, training을 하는 동안 **정책을 주기적으로 evaluate**하였다.   * 다음 그림은 **환경 선택에 따른 performance curve**이다. |

**3. RESULTS (Cont.)**

|  |
| --- |
| 다음 그림은 **task에서 사용된 일부 환경에서의 rendering**을 보여준다.    다음 그림은 **간단한 task**에서는 **DDPG가 systematic bias 없이 정확히 return**한다는 것을 보여준다. |

**4. SUPPLEMENTARY**

|  |
| --- |
| **<Experiment Details>**  Neural network 파라미터를 학습시키기 위해 **Adam 알고리즘**을 적용하였다.  **<Planning Algorithm>**  Planning 알고리즘은 **model-predictive controller**로 구현되었다.  **<Environment Details>**  Torcs 환경에서 각 step에서 **track에 project되는 자동차의 속도에 대해 positive reward**를 제공하는 reward function을 사용하였다.  **<MuJoCo Environments>**  Physical control task에서 **모든 step에서 feedback을 제공**하는 reward function을 이용하였다.   * 모든 task에서 **reward는 작은 action cost를 포함**한다. * 모든 task는 **static한 goal state**를 갖는다. * **간단한 threshold에 의해 결정되는 fall**들에 대해 **negative reward와 early termination**을 적용하였다. |