**Noname manuscript No.**
(will be inserted by the editor)

# Extensions for DDPG and analysis of its components

## subtitle here

**Maximilian Gehrke · Tabea Wilke · Yannik Frisch**

Received: date / Accepted: date

**Abstract** TODO

**Keywords** DDPG · DQN · DPG

## 1 Introduction

Deep Deterministic Policy Gradients (DDPG) arises from Deterministic Policy Gradients (DPG) and Deep Q-Learning (DQN). In the following we describe the underlying algorithms DPG and DQN and which aspects DDPG uses of both of them.

### 1.1 Deep Q-Learning (DQN)

DQN is the combination of neural networks and q-learning. It works on a deterministic environment with the goal to achieve the optimal action-value function. This means finding the best action with respect to the rewards also in the future to a given state. In terms of a formula it is represented by

$$Q^*(s_t, a_t) = \max_{\pi} E\left[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} | s_t = s, a_t = a, \pi\right]$$

with $\lambda$ as discount factor smaller but close to 1, so the agent takes also future reward into account. The rewards of the future will have impact on the result

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

but the influence decreases with time. For estimating the action-value function a deep network is used. Furthermore, a replay buffer is used which will save samples of the environment. Therefore, it is possible to achieve a non correlated batch. There are different ways of estimating the expected q-values. Either with a target network with the same structure as the network for the action-value function or the normal network. If a target network is used, the target weights need to be updated after some training steps.
The loss is calculated through the mean-squared-loss of the expected q-value and the estimated q-value:

$$L_i(\theta_i) = E_{(s,a,r,s')}\left[\left(r + \gamma \max_{a'} Q(s^{'}, a^{'}; \theta_i^{'}) - Q(s, a; \theta_i)\right)^2\right]$$

DQN can only handle discrete and low-dimensional action spaces.

**Algorithm 1** Deep Q-Learning (DQN)

**Initialize:** Replay buffer $D$ with high capacity
**Initialize:** Neural network for action-value function $Q$ with random weights $\theta$
**Initialize:** Neural network for target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
  **for** episode 1 **to** $M$ **do**
    reset environment to state $s_1$
    **for** $t = 1$ **to** $T$ **do**
      **if** random $i \leq \epsilon$ **then**
        random action $a_t$
      **else**
        $a_t = \text{argmin}_a Q(s_t, a; \theta)$
      **end if**
      execute $a_t \rightarrow$ reward $r_t$ and next state $s_{t+1}$
      save $(s_t, a_t, r_t, s_{t+1})$ in $D$
      sample minibatch $(s_i, a_i, r_i, s_{i+1})$ from $D$
      $q_i = \begin{cases} r_i & \textit{if episode terminates at step i+1} \\ r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a_i; \theta^-) & \textit{else} \end{cases}$
      perform gradient descent on $(q_i - Q(s_i, a_i; \theta))^2_\theta$
      every $C$ steps update $\hat{Q} = Q$
    **end for**
  **end for**

### 1.2 Deterministic Policy Gradient (DPG)

The most problems in reinforcement learning have an continuous action space, which makes it very difficult to choose the best action given a policy. In a stochastic point of view, there is a probability distribution which represents the policy. The policy is obtained through $\pi_\theta(a|s) = P[a|s; \theta]$. The goal is to achieve the best possible return and therefore choose the best action by solving the problem with a gradient over the total reward. This is only possible through solving an integral over all actions and actions. In the deterministic view the policy is a discrete mapping from state to actions and thus only one integration over the state space is sufficient. The integration is done via importance sampling. As a result of this, stochastic policy gradients need much more samples than deterministic policy gradients. To handle the exploration-exploitation dilemma, the idea of Silver et al. is to use a stochastic behaviour policy and a deterministic target policy. The behaviour policy should ensure that the exploration is big enough and the target policy should exploit enough based on the policy gradient. The algorithm they used was an off-policy actor-critic method. Thereby, the action-value function is estimated with a function approximation. The policy parameters will be updated in direction of the gradient of the action-value function. The direction of the gradient underlies the *policy gradient theorem*:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_S \rho^\pi(s) \int_A \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) \mathrm{d}a \mathrm{d}s \\ &= E_{s\sim\rho^\pi,\, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)] \end{aligned}$$

### 1.3 Actor-Critic Methods

The advantage of actor-critic methods is that they learn policies as well as value functions. To get an intuition about these methods the following figure illustrates the update-cycle:

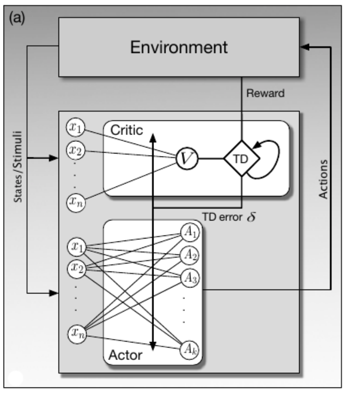


**Fig. 1** Intuition about actor-critic methods (figure from [5])

The actor is responsible for the change of the policies, the critic has to update the parameter of the state-value function. Updating the actors' and the critics' parameter follows the TD-error of the critic which is produced through the reward and the current error of the estimated state values. As Fig. 1 illustrates, the actor has no information about the current reward and the critic has no direct influence on the actions. The update of actor and critic can be formulated as follows:

**One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$**

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

**Fig. 2** Intuition about actor-critic methods (figure from [5])

### 1.4 Deep Deterministic Policy Gradients (DDPG) [1]

DDPG is a model-free and off-policy algorithm which arises from the combination of DQN and DPG. As a result of a continuous action space which leads to a not possible application of Q-learning. Therefore, it is a policy gradient algorithm which uses actor-critic methods with a deterministic target policy and deep Q-learning. Both the actor and the critic will be done with neural networks to represent the parameters. The actor will choose an action from a continuous action space.

- DQN uses deep networks to estimate the action-value function
  - it can only handle discrete and low-dim action spaces
- discretizing the action space often suffers from the course of dimensionality
- PolicyGradientTheorem from continous space to discrete space presented in DPG paper
- naive extension of DPG with nns turns out to be unstable for challenging problems
- Deep DPG (DDPG): combination of DQN and DPG, where:
  - networks are trained off-policy with samples from a replay buffer to minimize the temporal correlations between samples
  - the networks are trained with target networks to give consistent targets during temporal difference backups
  - batch normalization is used
- DDPG is able to learn from low dim observations (torques etc.), aswell as from high dim observations in pixel space

## 2 Extensions to the Algorithm

## References


1. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971
2. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602
3. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529
4. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: ICML
5. Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press