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

Deep Deterministic Policy Gradients: Components and Extensions

Yannik Frisch · Tabea Wilke · Maximilian Gehrke

Received: date / Accepted: date

Abstract TODO

Keywords DDPG · DQN · DPG

1 Introduction

The field of reinforcement learning deals with solving problems that are accessible through the interaction of an agent with it's environment. Such problems can be defined as Markov Decision Processes [x], which consist of a tuple (S, A, R, P, γ), where S is the state-distribution, A is the action-distribution, $R : S \times A \to \mathbb{R}$ is the reward function mapping states and actions to a scalar reward $r \in \mathbb{R}$, $P : S \times A \to S$ is the state transition function mapping states and actions to states, and γ is the discount factor used to make an agent more or less farsighted.
The goal of an agent can then be defined by finding an optimal policy $\pi^* : S \to A$ that determines when to take which action in order to maximize the observed reward. This is equal to taking the action that maximizes the optimal action-value function $Q^* : S \times A \to \mathbb{R}$, defining the value of state-action pairs.
For many applications the environment details, i.e. R and P, are not available. This requires the use of so called model-free algorithms, e.g. Q-Learning (Watkins and Dayan 1992), which updates an internal representation of the action-value function $Q(s, a)$ by the temporal-difference error between the current and the successor state after performing an action. This update converges

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

S. Author
second address

towards the optimal action-value function, which can be used to determine an optimal policy by simply greedily choosing the action that maximizes it.
The internal lookup-table representation of this function is not tractable for large state-action spaces. This problem is addressed by value function methods, e.g. the DQN-Algorithm (Mnih et al. 2013), which is an adaption to Q-Learning where the action-value function is approximated with a deep neural network. Instead of approximating the value-function, one could also approximate the policy $\pi(s|a)$ directly, which is known as policy-gradient methods. A deterministic version was developed in (Silver et al. 2014). Actor-critic methods (Konda and Tsitsiklis 2000) approximate the value-function as well as the policy.
Finally, the Deep Deterministic Policy Gradient (DDPG) approach (Lillicrap et al. 2015) combines the above mentioned methods to an actor-critic algorithm using neural network function approximation for the policy and the action-value function and learning a deterministic policy.
We will give more detailed insights into the mentioned algorithms and how DDPG evolved from them in the next sections, before we describe some possible extensions for it in section 4.

2 Preliminaries

The general goal of an reinforcement learning algorithm is to find an optimal behavior policy $\pi^*(a|s)$, or $\pi^*(s)$ in the deterministic case, which maximizes the expected total reward an agent collects while following it during an episode. Such an optimal policy can be defined by

$$\pi^*(a|s) = \max_a Q^*(s,a) = \max_a \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

[EDIT: REFORMULATE EQUATION] where t is the current time-step and T the final time-step ending the episode. This policy is just greedily choosing the action maximizing the optimal action-value function $Q^*(s,a)$. By definition this optimal value function yields the bellman equation (Sutton and Barto 2018) and can be reinterpreted as maximizing the current reward and the discounted action-value of the resulting state. In formula this gives:

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a')|s,a\right]$$

2.1 Q-Learning

The Q-Learning approach (Watkins and Dayan 1992), one of the earliest reinforcement learning methods, uses lookup-tables to represent the policy $\pi(a|s)$

and action-value function $Q(s, a)$. The optimal action-value function is calculated by applying the bellman equation using the temporal difference error from current to successor states. The iterative update rule is

$$Q^{(i+1)}(s, a) \leftarrow (1 - \alpha)Q^{(i)}(s, a) + \alpha \left(r + \gamma \max_{a'} Q^{(i)}(s', a')\right)$$

where α is a learning rate hyper-parameter. This update rule is proven to converge to the optimal action-value function using a lookup-table representation [x], but this becomes intractable for large state-action spaces.

2.2 Deep Q-Learning

The Deep Q-Network approach (DQN) (Mnih et al. 2013) combines the approximation power of neural networks with traditional Q-learning. The algorithm is an off-policy, model-free approach and is able to find a close to optimal action-value function for many cases (Mnih et al. 2015), and from this a close to optimal policy. For approximating the action-value function $Q(s, a|\theta) \approx Q(s, a)$ the approach uses a deep neural network with parameters θ, called the Q-Network. The Q-Network can be trained by sequentially minimizing the loss function $L_i(\theta)$, depending on the parameters

$$L^{(i)}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'|\theta^{(i-1)}) - Q(s, a|\theta^{(i)})\right)^2\right]$$

This loss function is similar to the classical temporal-difference loss used in Q-Learning, but with approximated action-value functions instead of lookup-tables. Derivating this loss w.r.t. the approximation's weights gives

$$\nabla_{\theta^{(i)}} L^{(i)}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'|\theta^{(i-1)}) - Q(s, a|\theta^{(i)})\right) \nabla_{\theta^{(i)}} Q(s, a|\theta^{(i)})\right]$$

The expectation can be approximated by sampling from an environment and this gradient can be used to optimize the loss function by using stochastic gradient descent.
Most optimization algorithms for neural networks assume independently and identically distributed data, which does not hold for experience samples sequentially generated by an agent interacting with it's environment. Therefore a replay buffer is used which stores a fixed amount of samples of the environment. This allows random mini-batch sampling, which temporarily decorrelates the samples and ensures a fixed size of updates, unlike using whole trajectories of the agent. It furthermore improves the data efficiency, as single episodes might potentially be used in several update steps. Sampling mini-batches also enables the efficient use of hardware optimization. This could be further improved by using empowered derivatives of stochastic gradient descent, e.g. *RPROP* as in the *neural fitted Q-Learning* approach (Riedmiller 2005) or *ADAM update* (Kingma and Ba 2014).
Directly using the approximated action-value function $Q(s, a|\theta)$ to calculate

the target for the update of itself, turned out to be unstable in many cases (Mnih et al. 2015). This problem can be addressed by using a target network, which is initialized with the same structure and parameters as the Q-network. This target network is only updated after a fixed amount of time-steps.
A pseudo-code for the DQN approach can be found in algorithm 1. It was able to significantly outperform earlier learning methods despite incorporating almost no prior knowledge about the inputs (Mnih et al. 2013). However, it is limited by the disability to cope with continuous and high-dimensional action spaces due to the max operator in the action selection (Lillicrap et al. 2015). This limitations can be addressed by combining the approach with the Deterministic Policy Gradient (Silver et al. 2014), which is described in the following section.

Algorithm 1 Deep Q-Learning (DQN)

Initialize: Replay buffer D with high capacity
Initialize: Neural network for action-value function Q with random weights θ
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
        $a_t = \text{argmax}_a \, Q(s_t, a|\theta)$
      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 mini-batch $(s_i, a_i, r_i, s_{i+1})_k$ of size k from D
      $q_i = \begin{cases} r_i & \textit{if episode terminates at step i+1} \\ r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'|\theta^-) & \textit{else} \end{cases}$
      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

2.3 Deterministic Policy Gradient

Many model-free reinforcement learning algorithms adapt the *generalized policy iteration* which consist of interleaving steps of *policy evlatuation* and *policy improvement* (Sutton and Barto 2018). The most common approach to the policy improvement step for a deterministic policy is to greedily select the action maximizing the (estimated) action-value function

$$\pi(s) = \arg\max_a Q(s, a)$$

Most problems in reinforcement learning consist of a continuous action space which makes it very difficult to greedily choose the best action given a policy,

because the max operator in the policy improvement step would require a global optimization at every iteration.
Rather than trying to maximize the action-value function $Q(s, a)$ globally by greedy improvements of the policy, one could move the policy in the direction of the gradient of $Q(s, a)$:

$$\nabla_{\theta^\pi} J \approx \mathbb{E}\left[\nabla_{\theta^\pi} Q(s, a|\theta^Q)\right]$$

Applying the chain rule to this equation gives the *deterministic policy gradient (DPG) theorem*:

$$\nabla_{\theta^\pi} J \approx \mathbb{E}\left[\nabla_a Q(s, a|\theta^Q)|_{a=\pi(s|\theta^\pi)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)\right]$$

where the expectation can again be approximated by sampling from an environment.
The deterministic policy gradient is potentially more sample efficient, especially for large action spaces, as it only requires integrating over the state space

$$\nabla_\theta J(\theta) = \int_S \rho^\pi(s) \nabla_\theta \pi(s|\theta) \nabla_a Q(s, a)|_{a=\pi(s)} ds$$

while the stochastic policy gradient (Sutton and Barto 2018) requires integrating over the state and the action space

$$\nabla_\theta J(\theta) \int_S \rho^\pi(s) \int_A \nabla_\theta \pi(a|s, \theta) Q(s, a) da ds$$

where $\rho^\pi(s)$ is state distribution depending on the current (parameterized) policy in both cases.
Only using deterministic action outputs will make an agent fully exploiting what it already knows, so one needs to make sure there still is exploration. This is realized by using an off-policy approach which follows a stochastic policy while learning a deterministic policy and could be implemented by simply adding some noise to the policy output.
The following section describes a typical structure of how to use deep neural networks for function approximation in reinforcement learning. Together with this section this led to the algorithm described in chapter 3.

2.4 Actor-Critic Methods

A lot of recent success in reinforcement learning is based on *actor-critic* methods (Konda and Tsitsiklis 2000). In contrast to value-function or policy-gradient methods, they parameterize both, the value function $Q(s, a) \approx \hat{Q}(s, a|\theta^Q)$, also known as the *critic*, and the policy $\pi(s|a) \approx \hat{\pi}(s|a, \theta^\pi)$, called the *actor*. While the actor learns how to choose the right action and is responsible to update his policy, the critic has to learn and update the parameters of the state-value function. The actor's and the critic's parameters can be updated following the TD-error of the critic which is computed from the observed reward and the

current error of the estimated state values in every time-step. The actor has no information about the current reward and the critic has no direct influence on the actions.

3 Deep Deterministic Policy Gradient

The combination of above approaches led to the *Deep Deterministic Policy Gradient (DDPG)* approach (Lillicrap et al. 2015), which is a model-free and off-policy algorithm. It uses actor-critic methods, a deterministic target policy and deep Q-Learning. Both, the actor and the critic, are realized by deep neural networks. The pseudo-code for DDPG can be found in 2.
It consists of a parameterized deterministic policy, the actor, $\pi(s|\theta^\pi)$ and a parameterized action-value function $Q(s, a|\theta^Q)$, the critic. The critic is updated using the *bellman equation* with a TD-error similar to Q-Learning (Watkins and Dayan 1992), which results in the loss function

$$L = \frac{1}{N}\sum_i((r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'}) - Q(s_i, a_i|\theta^Q))^2$$

The actor is updated using the DPG theorem:

$$\nabla_{\theta^\pi} J \approx \mathbb{E}\left[\nabla_a Q(s, a|\theta^Q)|_{a=\pi(s|\theta^\pi)} \nabla_{\theta^\pi}\pi(s|\theta^\pi)\right]$$

The use of neural networks to parameterize the above functions implies that convergence guarantees do not hold anymore. Therefore the Actor-Critic DPG approach is combined with recent successes from DQN:
To ensure independently and identically distributed data, the authors use a replay buffer and sample random mini-batches from it. This again decorrelates the samples and allows the efficient use of hardware optimization.
To address instability issues from applying deep neural network approximation to Q-Learning they also use *target networks* which are copies of the actor $\pi'(s|\theta^{\pi'})$ and the critic $Q'(s, a|\theta^{Q'})$. These target-networks track the learned networks and are constrained to slow changes by using soft updates: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. These consistent targets might slow down the learning process but greatly improve the stability of it.
Using low dimensional feature input might give very different scales for the single states, e.g. the angle of a joint which is bounded to $[-\pi, \pi]$ and the angular velocity which is potentially unbounded. This can lead to problematic learning for the neural networks and is addressed by using batch normalization which normalizes each dimension across the samples in a mini-batch.
To ensure exploration while using a deterministic policy, a noise process N is added to the action output of the actor network. This noise process can be chosen to suit the environment, e.g. a time-dependent Ornstein-Uhlenbeck process (Ricciardi and Sacerdote 1979) or a consistent gaussian noise (Barth-Maron et al. 2018).
The algorithm was evaluated on more than 20 simulated physical tasks using the same algorithm, network structures and hyper-parameters, including

classic control problems like the cart-pole environment. Using low-dimensional feature input, it was able to find close to optimal policies for most of the tasks. Their performance is competitive with these found by a controller with full access to the environment. The algorithm was also able to find good policies using high dimensional pixel input. For simple tasks this turned out to be as fast as using low dimensional state features.

Algorithm 2 Deep Deterministic Policy Gradient (DDPG)

Initialize: Replay buffer D with high capacity
Initialize: Critic network $Q(s, a|\theta^Q)$ and actor network $\pi(s|\theta^\pi)$ with random weights θ^Q and θ^π
Initialize: Initialize target networks Q' and π' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\pi'} \leftarrow \theta^\pi$
for episode 1 **to** M **do**
  Initialize random process N for action exploration
  Reset environment to state s_1
  for $t = 1$ **to** T **do**
    Select action $a_t = \pi(s_t|\theta^\pi) + N_t$ from local actor
    Execute action a_t and observe reward r_t and next state s_{t+1}
    Save (s_t, a_t, r_t, s_{t+1}) in replay buffer D
    Sample mini-batch $(s_i, a_i, r_i, s_{i+1})_k$ with size k from D
    Set TD-target from target networks:
      $y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'})$
    Update the critic by minimizing the loss:
      $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor using the sampled policy gradient:
      $\nabla_{\theta^\pi} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s=s_i}$
    Update the target networks:
      $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
      $\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$
  end for
end for

The most challenging issues of the approach are his poor sample efficiency and instabilities during learning. We present some possible extensions to DDPG in the next chapter which might improve on these issues.

4 Improvements for DDPG

Despite it's good performance on many simulated tasks there is still some room to improve the algorithm's data efficiency and overcome it's instabilities during training. We present some possible extensions for it in this section.

4.1 Using importance sampling to sample from the replay-buffer

In practice the algorithm is limited by the maximum storage size N of the replay-buffer D. Overwriting older samples by current ones when the maximum size is reached does not differentiate between more or less important

experiences, because uniform random sampling does weight all experiences equally. One could use a technique similar to *prioritized sweeping* (Moore and Atkeson 1993) which uses *importance sampling* (Glynn and Iglehart 1989) to prefer transitions which are more important over ones that have less value for the training process.

4.2 Using Action Noise in Parameter Space

Instead of adding noise to the action space to ensure exploration, one could add adaptive noise directly to the parameters of the neural network (Plappert et al. 2017). This would add some randomness into the parameters of the agent and therefore into the decision it makes, while still always fully depending on it's current observation about it's environment. This parameter noise makes an agent's exploration more consistent and results in a more effective exploration, increased performance and smoother behavior.

4.3 Evolutionary Approaches

One can consider an even more extreme case of the above mentioned extension, which would be the use of *Evolutionary Strategies* to approximate the gradient of our objective function (Salimans et al. 2017). This does not require back-propagation and is competitive with state of the art RL.

4.4 Using Multiple Actors in Parallel

(Barth-Maron et al. 2018) TODO

4.5 Improvements of the Deep Neural Network Architectures

The relatively simple network architectures give room for improvement, especially when using convolutional neural networks. TODO

5 Conclusion

TODO

References

Barth-Maron G, Hoffman MW, Budden D, Dabney W, Horgan D, Muldal A, Heess N, Lillicrap T (2018) Distributed distributional deterministic policy gradients. arXiv preprint arXiv:180408617

Glynn PW, Iglehart DL (1989) Importance sampling for stochastic simulations. Management Science 35(11):1367–1392

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980

Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms. In: Advances in neural information processing systems, pp 1008–1014

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

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

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

Moore AW, Atkeson CG (1993) Prioritized sweeping: Reinforcement learning with less data and less time. Machine learning 13(1):103–130

Plappert M, Houthooft R, Dhariwal P, Sidor S, Chen RY, Chen X, Asfour T, Abbeel P, Andrychowicz M (2017) Parameter space noise for exploration. arXiv preprint arXiv:170601905

Ricciardi LM, Sacerdote L (1979) The ornstein-uhlenbeck process as a model for neuronal activity. Biological cybernetics 35(1):1–9

Riedmiller M (2005) Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In: European Conference on Machine Learning, Springer, pp 317–328

Salimans T, Ho J, Chen X, Sidor S, Sutskever I (2017) Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:170303864

Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: ICML

Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press

Watkins CJ, Dayan P (1992) Q-learning. Machine learning 8(3-4):279–292