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 Actor-critic methods with deep neural networks were of growing interest in the reinforcement learning community in the last years. The *Deep Deterministic Policy Gradient* algorithm evolved as a powerful tool from the *Deep Q-Learning* algorithm combined with the *Deterministic Policy Gradient* theorem to learn a fully deterministic policy. We explain these components and how they merge together to an algorithm achieving good performance for many simulated and physical tasks before we head to some possible improvements for it, which could be applied to overcome the algorithms lacking data efficiency and instability issues during its training progress.

Keywords DDPG · DQN · DPG · D4PG · Parameter Noise

1 Introduction

The field of reinforcement learning deals with solving problems that are accessible through the interaction of an agent with its environment. Such problems can be defined as *Markov Decision Processes* (Howard 1960), 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

Yannik P. Frisch
TU Darmstadt, Germany
E-mail: yannik_phil.frisch@stud.tu-darmstadt.de

Tabea A. Wilke
TU Darmstadt, Germany
E-mail: tabeaalina.wilke@stud.tu-darmstadt.de

Maximilian A. Gehrke
TU Darmstadt, Germany
E-mail: maximilian_alexander.gehrke@stud.tu-darmstadt.de

to make an agent more or less farsighted. The goal of an agent can then be defined by finding an optimal policy $\pi^* : S \rightarrow A$ 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 $Q^* : S \times A \rightarrow \mathbb{R}$ that defines the value of state-action pairs (s, a).
For many applications, the environment details, i.e. R and P, are not available and the agent can only observe the state s of his environment, perform an action a in it and observe an immediate reward $r(s, a)$ and a successor state s'. 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. The internal lookup-table representation of this algorithm is not tractable for large state-action spaces. This problem is addressed by *value function methods*, e.g. the *Deep Q-Learning* (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, known as *policy-gradient methods*. *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 with the *Deterministic Policy Gradient* (DPG) (Silver et al. 2014) to an actor-critic algorithm using neural networks to approximate the policy and the action-value function while 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 5.

2 Deep Q-Learning

The general goal of a 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 is greedily choosing actions that maximize the optimal action-value function $Q^*(s, a)$ that can be defined by

greedily chooses oder is greedily if choosing

$$Q^*(s, a) = \max_{\pi^*} Q^{\pi^*}(s, a) = \mathbb{E}\left[\sum_{k=0}^{T} \gamma^k r(s_{t+k}, a_{t+k}) | s_t = s, a_t = a, \pi^*\right]$$

where t is the current time-step and T the final time-step ending the episode. 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. The function can be rewritten as

Eigenname Bellman

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

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 becomes intractable for large state-action spaces.
The *Deep Q-Learning* approach (DQN) (Mnih et al. 2013) combines the approximation power of neural networks with traditional Q-learning to overcome this issue. 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. This 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 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 yields the gradient

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

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 its environment. Therefore, an *experience 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 improved further 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).
Using the approximated action-value function $Q(s,a|\theta)$ directly 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). These 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

3 Deterministic Policy Gradient

Many model-free reinforcement learning algorithms adapt the *generalized policy iteration* which consist of interleaving steps of *policy evaluation* 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

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

Though most problems in reinforcement learning consist of a continuous action space what 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 $Q(s, a)$ w.r.t. the policy's parameters:

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

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

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

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

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

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

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

where $\rho^\pi(s)$ is distribution of visited states depending on the current (parameterized) policy in both cases.
Using only 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 an actor-critic architecture combining the above mentioned methods to a state-of-the-art reinforcement algorithm.

4 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 that 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) resulting in the loss function

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

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 *experience 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 directly combining the deep neural network approximation and the DPG theorem, 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 because using an older set of parameters to generate the targets adds a delay between the time we update our functions and the time these updates affect our targets. This makes divergence or oscillations during the learning process much more unlikely (Mnih et al. 2015).
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 those 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.
The most challenging issues of the approach are his still poor sample efficiency and instabilities during learning. We present some possible extensions to DDPG in the next chapter which might improve on these issues.

For using

den Satz verstehe ich iwie nicht, gehört

its

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

5 Improvements for DDPG

Despite its good performance on many simulated tasks there is still some room to improve the algorithm's data efficiency and overcome its instabilities during training. We present some possible extensions for it in this section.
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 that are more important over ones that have less value for the training process. This could be implemented by using non-uniform priorities, as in (Schaul et al. 2015; Barth-Maron et al. 2018).
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 its current observation about its environment. This parameter noise makes an agent's exploration more consistent and results in a more effective exploration, increased performance and smoother behavior. It can be seen as a less extreme case of the use of *Evolutionary Strategies* to approximate the gradient of our objective function (Salimans et al. 2017). These would not require back-propagation at all and might still be competitive with state-of-the-art

reinforcement learning algorithms.
The actor and critic updates rely completely on sampling from the experience replay buffer. Therefore, this process could be parallelized using multiple independent actors, all writing to the same buffer. Significant time-savings could be achieved by this parallelization in the *Distributed Distributional Deep Deterministic Policy Gradient* (D4PG) algorithm (Barth-Maron et al. 2018).
This algorithm did furthermore adopt a distributional version of the critic update from (Bellemare et al. 2017). These distributions model the randomness due to intrinsic factors, including the underlying uncertainty from using function approximation in a continuous space. These distributional updates result in better gradients and therefore improve the performance and stability of the learning progress (Barth-Maron et al. 2018).
Instead of only using the return of the successor state, one could also use the *n-step return* when estimating the temporal difference error (Barth-Maron et al. 2018). We hypothesize one could go even further and utilize the *TD(λ)-return* by using *eligibility traces* (Tesauro 1995).
The last point of this section addresses the use of neural networks to approximate the actor and the critic. The mentioned papers use relatively simple network architectures that leave a lot of room for improvement, especially when using convolutional neural networks for processing high-dimensional image data in pixel space. A lot of techniques have been implemented in the deep-learning community which might also help in our case, e.g. the use of *weight sharing* (Nowlan and Hinton 1992), *pooling layers* (Zeiler and Fergus 2013) or *dropout layers* (Srivastava et al. 2014). One could experiment with much more complex network architectures, e.g. the *AlexNet*, *VGGNet* or *GoogLenet*, with *ResNet* (He et al. 2016), or even with *recurrent neural networks* (Haykin 1994) to process sequences of states instead of single ones.

Significant time-savings were achieved

ohne Klammern

Glaube entweder GoogleNet oder GoogLeNet, G

6 Conclusion

We presented a state-of-the-art reinforcement learning algorithm DDPG that evolved from the successful DQN algorithm by combining it in an actor-critic fashion with the DPG theorem. The algorithm is already able to solve a lot of simulated and real physical tasks and has several advantages over other reinforcement learning methods, e.g. the output of a fully deterministic policy and the ability to deal with the exploration-exploitation dilemma completely separated from the learning algorithm by implementing suiting action noises for exploration. Though it still suffers from a poor sample efficiency and instabilities during the learning progress. These issues could be addressed by our presented improvements, many of which have already been successfully realized in the D4PG algorithm. Further evaluations are needed to demonstrate the effectiveness of these enhancements.

them

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

Bellemare MG, Dabney W, Munos R (2017) A distributional perspective on reinforcement learning. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, pp 449–458

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

Haykin S (1994) Neural networks, vol 2. Prentice hall New York

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778

Howard RA (1960) Dynamic programming and markov processes.

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

Nowlan SJ, Hinton GE (1992) Simplifying neural networks by soft weight-sharing. Neural computation 4(4):473–493

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

Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. arXiv preprint arXiv:151105952

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

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1):1929–1958

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

Tesauro G (1995) Temporal difference learning and td-gammon. Communications of the ACM 38(3):58–69

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

Zeiler MD, Fergus R (2013) Stochastic pooling for regularization of deep convolutional neural networks. arXiv preprint arXiv:13013557