PENALIZED BOOTSTRAPPING FOR REINFORCEMENT LEARNING IN ROBOT CONTROL

Christopher Gebauer and Maren Bennewitz

Humanoid Robots Lab, University of Bonn, Bonn, Germany
{cgebauer,maren}@cs.uni-bonn.de

ABSTRACT

The recent progress in reinforcement learning algorithms enabled more complex tasks and, at the same time, enforced the need for a careful balance between exploration and exploitation. Enhanced exploration supersedes the requirement to hardly constrain the agent, e.g., with complex reward functions. This seems highly promising as it reduces the work for learning new tasks, while improving the agents performance. In this paper, we address deep exploration in reinforcement learning. Our approach is based on Thompson sampling and keeps multiple hypotheses of the posterior knowledge. We maintain the distribution over the hypotheses by a potential field based penalty function. The resulting policy is more performant in terms of collected reward. Furthermore, is our method faster in application and training than the current state of the art. We evaluate our approach in low-level robot control tasks to back up our claims of a more performant policy and faster training procedure.

KEYWORDS

Deep Reinforcement Learning, Deep Exploration, Thompson Sampling, Bootstrapping

1. INTRODUCTION

The outstanding performance of deep reinforcement learning firstly shown by Mnih *et al.* [1] has opened a wide range of possibilities. Especially in the field of robot control this is promising, as it heavily reduces the requirements to implicitly state the desired behavior in more complicated situations. For example, popular non-learning methods are based on the dynamic window approach for navigation [2] or on differential dynamic programming for low-level control [3]. These approaches solve the task to interact with the environment sufficiently, but need to be carefully fine-tuned and are limited to the designed representation of the environment. The lack of understanding and integration of correlations between events in the interaction with the environment lead to a very stable reactive behavior but instability when more foresightedness is required. This is especially given for sequences of actions that rather dependent on high-level decisions as in human-robot interaction, when social acceptable behavior is addressed.

A reinforcement learning agent with enhanced exploration skills is very promising, especially in tasks including more elaborated behavior. While classical approaches are able to avoid collision with humans [4], learning is rather able to solve this task in a socially compliant manner [5]. Even though the results are promising and lead to good navigation policies, the process of learning is inert, due to the absence of deep exploration. With lack of deep exploration, the agent favors to greedily search in the known action space for the optimal solution instead of deeply explore the unknown capabilities first. As gradient-based methods have been established in recent years, the greedy behavior is originated in the local convergence given by the basic assumptions of their derivation. In navigation tasks, the resulting policy of poorly explored local optima rarely results in the expected behavior. Even though Chen *et al.* [5] showed the promising opportunities of

Dhinaharan Nagamalai et al. (Eds): CSEIT, WiMoNe, NCS, CIoT, CMLA, DMSE, NLPD - 2020
pp. 155-165, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101114

reinforcement learning in socially-aware navigation tasks, more research is required for robust end-to-end machine learning solutions [6].

While neural networks of great size inherit the potential to map almost any non-linear function, the major lack is to efficiently force the agent to explore its capabilities as well as the environment itself. Common methods address this problem by a detailed reward design including preprocessing [7] or guided learning steps with increasing difficulty [8]. Another approach includes network extensions to learn auxiliary tasks [9] or an internal reward based on reconstruction error of the current state [10].

In this paper, we introduce a novel boostrapped version of twin delayed deep deterministic policy gradient (TD3) [11] based on Thompson sampling [12] to increase deep exploration and increase the performance of the resulting neural network in terms of maximizing the expected return. Thompson sampling addresses the balance between exploitation and exploration by randomly sampling the policy parameters from a posterior distribution and acting according to it for one episode. The uncertainty inherited in the posterior distribution naturally induces exploration due to the resulting uncertainty in the optimal action. Furthermore, we penalize the similarity of the hypotheses to maintain the posterior distribution. In comparison to Zheng *et al.* [13], our novel bootstrapping method reduces the required computational resources while still improving the performance of the resulting agent. All the claims are backed up with an experimental evaluation.

2. Related Work

To address the problem of deepening the exploration, multiple approaches have been developed in the last few years. The most widely used one is curriculum learning [8], which improves the final policy by increasing the difficulty of the task over time. For example, Kulhanek *et al.* [14] applied this method by increasing the complexity of the environment at given milestones during training of a vision-based navigation policy. Nevertheless is this approach not addressing the learning algorithm itself but modifies the information stream of the training samples and therefore is always usable as an extension.

Another concept is based on outputs from additional heads using the same encoded state as the policy. The encoded state represents, e.g., the output of a convolutional neural network and is shared among all consequential networks. A head is the neural network that uses the encoded state to generate any desired output, e.g., action commands. The purpose of additional heads is to influence the shared network structure without direct usage of the head's output. Jaderberg *et al.* [15] introduced this idea as auxiliary tasks and optimized the additional heads via self-supervised learning using available quantities from the environment. Mirowski *et al.* [9] improved this method by introducing further auxiliary tasks, especially to predict the depth based on an RGB image. Both approaches require the agent to receive such quantities from the environment as well as are strongly bounded to a specific task.

Another method is based on internal usage of auxiliary outputs from the neural network. For example, Pathak *et al.* [10], building upon Stadie *et al.* [16], focused on reconstructing the relevant information of the next state by learning the dynamics of the environment. Exploration is induced by adding a bonus reward on states that have been reconstructed worse, representing its novelty. The approach is known as curiosity learning and was applied very effectively to robot navigation by Zhelo *et al.* [17]. Variational information maximizing exploration [18] uses curiosity learning and extends it by directly maximizing the expected information gain due to the corresponding action. All these methods have especially the problem to be unable to differentiate between stochastic dynamics and uncertainty, as it is not directly approximating latter.

Blundell *et al.* [19] used Bayesian neural networks, building upon Hinton *et al.* [20] and Graves *et*

al. [21]*, to introduce uncertainty into the current weights. This naturally induces exploration due* to the resulting distribution over possible actions with respect to the current state and the uncertainty towards the inherited weights. Nevertheless is this method currently limited to applications where a network is trained to fit a known target value. Henderson *et al.* [22] applied this concept to different actor-critic algorithms by training the critic under weight uncertainty. In general, the critic evaluates the actions, which are mapped directly from states using the actor. This uncertainty, even though not applied to the actor itself, heavily improved training stability, as the actor depends on the critics performance.

Another class of approaches is based on Thompson sampling, which addresses exploration by considering multiple hypotheses of the next optimal action conditioned by a given state and the posterior knowledge. Osband *et al.* [23] applied this technique to deep reinforcement learning, which is known as bootstrapped deep Q-learning. The key concept is a shared network and multiple instances that return the action-value function Q, representing the heads. Thompson sampling is applied by drawing a specific head before each episode and acting according to it. The multi-head structure ensures an estimation of the posterior knowledge inherited by the neural networks to introduce uncertainty. Furthermore is each head trained on its own subset of the complete dataset $\mathcal{D}$. Zheng *et al.* [13] extended this to the actor-critic method by replicating multiple actor-critic pairs, known as double bootstrapped deep deterministic policy gradient (DBDDPG). Both approaches achieve a clear increase in performance compared to non-bootstrapped agents. Our approach is based on the actor-critic methods as well, but extends DBDDPG by adjusting the critic structure and penalizing the similarity of each bootstrapped actor. Furthermore, we reduce the computational cost in context of training and application, while still improving the resulting performance regarding the expected return.

3. OUR APPROACH

In this section we first describe all preliminaries regarding the reinforcement learning setting and describe our approach in detail afterwards.

3.1. Preliminaries

We model the problem as a Markov Decision Process (MDP), where an agent interacts with the environment. Based on the current state $s_t \in \mathcal{S}$ the agent applies action $a_t \in \mathcal{A}$, according to the policy $\pi(\theta) : \mathcal{S} \rightarrow \mathcal{A}$ defined by its parameters θ. At the end of timestep t, the agent receives the reward $r_t \in \mathcal{R}$ and the next state s_{t+1} according to the state-transition probability distribution $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. The discounted return is defined by $R_t = \sum_{i=0}^{T} \gamma^i r_{t+i}$, where $t + T$ represents the final time step and $\gamma \in [0, 1]$ is the discount factor.

In general the objective of reinforcement learning is to maximize the expected return $J(\pi) = \mathbb{E}[R_0|\pi]$, where the gradient $\nabla_\theta J(\pi)$ is used to update the policy. For deterministic policy gradient (DPG) [24] the gradient is not defined to directly depend on the policy, but rather points in the direction of the action-value function's gradient ∇Q at the current sample point

$$\nabla_\theta J(\pi) \quad = \quad \mathbb{E}[\nabla_a Q(s, a)|_{a=\pi(s_t)} \nabla_\theta \pi(s)]. \tag{1}$$

The action-value function Q is defined by $Q(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$ with its parameters θ_Q and updated using temporal difference:

$$\begin{aligned} L(Q) &= \mathbb{E}\left[(Q(s, a) - y_t)^2\right] \\ \text{with} \quad y_t &= r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})) \end{aligned} \tag{2}$$

The target y_t and therefore the loss function L depend on the policy and the action-value function itself. For deep deterministic policy gradient (DDPG) [25], the neural network based extension

of DPG, this introduces an instability towards the update as the numerous number of network parameters are adjusted concurrently. To overcome this, Mnih *et al.* [1] introduced a target network that is updated less frequently by partially copying the network parameters. The target network is denoted by a quotation sign, as, e.g., Q', and applied to all trained models.

Another problem is the overestimation bias of the action-value function excessively increasing over training. As the policy update depends on the stability of the action-value function, the overestimation leads to a general instability of the training. Twin delayed deep deterministic policy gradient (TD3) [11] countermeasures this effect by adding another critic and taking the minimal action-value estimate as target. Both of the action-value functions are trained separately with identical, but modified target action-value:

$$y_t = r_t + \gamma \min_{i=1,2} Q'_i(s_{t+1}, \pi'(s_{t+1}) + \epsilon_t) \tag{3}$$

This improves stability by almost eliminating the overestimation bias during training. To further improve generalization, as deterministic policies usually tend to naturally overfit, the smoothing target noise term ϵ_t is added to the action estimation of the target policy in the target function [11]. It is drawn from a clipped Gaussian distribution. TD3 is the underlying algorithm used for our novel boostrapping approach based on Thompson sampling.

3.2. Bootstrapped Actor

The benefit of Thompson sampling is originated in the optimization of multiple correlated hypotheses based on the given dataset $\mathcal{D}$. Each hypothesis corresponds to its own set of function parameters $\theta_i \in \theta$ and trains on a subset $\mathcal{D}_i$ to ensure the maintenance of the distribution over the posterior knowledge $\hat{P}(\theta|\mathcal{D})$. The hat denotes the approximation of the true posterior distribution, as we do not know the true distribution, but rather draw multiple hypotheses assuming to be distributed according to P. Before each interaction with the environment, a hypothesis is chosen and acted upon on. This introduces an uncertainty and, therefore, a more elaborated exploration compared to greedy policies.

Our core contribution is a boostrapped version of TD3 based on Thompson sampling, in the following referred to as Multi-TD3. We instantiate a number $N \in \mathbb{N}$ of actors with a random set of parameters θ_i, where $i \in N$, drawn from the prior distribution $P(\theta)$. These actors represent our hypotheses and are forming the distribution $\hat{P}(\theta|\mathcal{D})$, which then is dependent on the prior distribution $P(\theta)$. To maintain the distribution $\hat{P}(\theta|\mathcal{D})$ we update each actor separately based on its subset $\mathcal{D}_i$ defined by the mask m_t drawn from a Bernoulli distribution [23]. The Bernoulli distribution is defined by the masking probability p, a new hyperparameter. The mask m_t has the size of N and indicates for each actor whether the specific sample is part of its subset or not. In our case, this results in a dataset $\mathcal{D}$ defined by the collection of tuples $\{s_t, a_t, s_{t+1}, r_t, m_t\}$. While each actor only depends on its subset $\mathcal{D}_i$, we combine the loss of all actors to update the shared network and therefore consider the whole dataset $\mathcal{D}$. The shared network is hindrance for clear separation of the different hypotheses, however, it is usually intractable to train complete copies of the same network in parallel.

We apply Thompson sampling by drawing randomly from the pool of actors before each episode, representing a hypothesis based on the posterior knowledge. In contrast to pure Thompson sampling, we choose during the entire episode the action according to the drawn policy π_i, neglecting its optimality, instead of redrawing before each interaction as suggested by Osband *et al.* [23]. While the randomness due to neglection of the argmax operator in the deterministic action inference is desired for data collection during training, the policy shall be greedy during evaluation. Therefore, we search greedily in each timestep for the optimal parameter

$$\theta^* = \underset{\theta_i}{argmax}\, Q_1(s_t, \pi(s_t, \theta_i)). \tag{4}$$

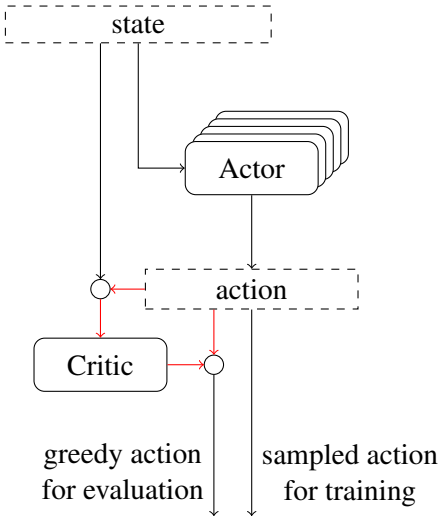

Figure 1: General information flow in Multi-TD3, illustrating the action generation in evaluation and during training.

Type your text

Algorithm 1: Multi-TD3

Input: Number of heads N, replay buffer $\mathcal{D}$, masking probability p, exploration noise ϵ, smoothing target noise ϵ_t

Initialize parameter $\theta_{Q_{j=1,2}}$, $\theta_{i \in N}$ from $P(\theta)$

for *each iteration* **do**

  if *new episode* **then**

    Pick actor $n \sim$ uniform{N}

  Apply $a_t \sim \pi_n(s_t) + \epsilon$

  Receive r_t and s_{t+1}

  Sample mask $m_t \sim$ Bernoulli$\{p\}$

  Store $\{s_t, a_t, s_{t+1}, r_t, m_t\}$ in $\mathcal{D}$

  Sample minibatch $\mathcal{B}$ from $\mathcal{D}$

  for $i \in N$ **do** $\hat{a}_i \sim \pi'_i(\mathcal{B}) + \epsilon_t$

  $y \leftarrow \max_{i \in N}\{ \min_{j=1,2} \ Q'_j(\mathcal{B}, \hat{a}_i)\}$

  Update each critic according to Eq. (2)

  Update each actor according to Eq. (6) using Eq. (1)

  Update the target networks

Q_1 refers to one of the two critics used in TD3, where either are usable. Eq. (4) satisfies the assumption on greedy maximization using argmax regarding the action in DPG. In Fig. 1 we illustrate the comparison of the information flow for action generation in training and evaluation. The additional loop, marked in red, for the greedy search of optimality brings in an overhead due to the linear cost in N. The greatest speedup in comparison to DBDDPG [13] is achieved by uniformly sampling an action during data collection for training instead of searching greedily for it. Additionally we are speeding up the greedy network inference by not increasing the number of critics to N, which results in a reduction of the runtime complexity from quadratic with N to linear.

As mentioned, the critic structure is not modified and stays identical to TD3. However, we modify the target function in Eq. (3) to still minimize over both critics and additionally to maximize over all predicted next actions according to each actor

$$y_t = r_t + \gamma \max_{i \in N} \left[\min_{j=1,2} Q'_i(s_{t+1}, \pi'_i(s_{t+1}) + \epsilon_t) \right]. \tag{5}$$

The resulting complete optimization problem is given by

$$\begin{aligned} \min_{\theta_i, \theta_{Q_j}} - \sum_{i=1}^{N} \mathbb{E}\left[Q_1(s, \pi(s, \theta_i))\right] + \sum_{j=1,2} \mathbb{E}\left[(Q_j(s, a, \theta_{Q_j}) - y_t)^2\right] \\ + \beta \sum_{i=1}^{N} \sum_{j=1, i \neq j}^{N} \frac{1}{||\pi_i - \pi_j||_2^2}, \end{aligned} \tag{6}$$

and summarized in Alg. 1 as well as visualized in Fig. 1. The last term is a penalty to further support the maintenance of the distribution $\hat{P}(\theta|\mathcal{D})$. It is inspired by the entropy cost used in, e.g., Schulman *et al.* [26]. It forces the distribution of the possible actions, given the current state, towards a uniform distribution. Since we apply deterministic policies, entropy is not applicable,

as no distributions over actions is available. However, we reformulated this to minimize the repellent force between the deterministic policies induced by their potential field, known from identically charged point particles in the field of electrostatic [27]. This equals minimizing the inverted Euclidean norm and results in a uniform spread of the policies across the action space. Therefore, maximizing the potential of the actions according to the deterministic policies is similar to maximizing the entropy of the policy's distribution. To prevent a division by zero, a lower bound is set for the Euclidean norm. The parameter β scales the cost and is a newly introduced hyperparameter.

4. EXPERIMENTAL EVALUATION

The main focus of this work is to increase deep exploration in the application of robot control tasks. In Sec. 4.1. we introduce the neural network structure and its hyperparameters we used in our experiments. The complete network structure, as well as the optimization algorithm is implemented using Tensorflow [28]. In Sec. 4.2. we compare our novel bootstrapping approach to the state-of-the-art bootstrapping method for deterministic actor-critic methods [13], as well as common unbootstrapped deterministic policies [25], [11]. We evaluated the performance in different low-level control tasks simulated in PyBullet [29] and wrapped with OpenAI gym [30]. In Sec. 4.3. we evaluate the reduced runtime complexity and especially the reduced training time in comparison to DBDDPG [13] as well as the overhead towards the unbootstrapped TD3 [11].

4.1. Deep Network Architecture

For DDPG and TD3 two different sets of hyperparameters turned out to be superior in our experiments. The one applied to DDPG and DBDDPG is similar to Lillicrap *et al.* [25]. We apply two dense layers with $\{\mathbf{512}$,$256\}$, while the bold number represents the shared layer for DBDDPG. The structure for the actor and critic is identical. The hyperparameters are given by the discount factor $\gamma = 0.95$, actor learning rate $l_{r,\pi} = 10^{-4}$, critic learning rate $l_{r,Q} = 3 \times 10^{-4}$, the exploration noise $\epsilon = 0.2$, the target update factor $\tau = 0.01$ and the size of the minibatch $\mathcal{B}$ with 128. We apply kernel and bias regularization via L2-regularization with $l_2 = 10^{-4}$. The replay buffer has a size of 10^6 for all approaches. For DBDDPG the degree of bootstrapping is given by $N = 5$, as suggested in the original paper [13], and the masking probability by $p = 0.5$.

For TD3 and our approach we choose slightly different hyperparameters, which have been tuned during hyperparameter validation. In general, the modified hyperparameters increase the performance with the cost of more unstable training. Instability refers to higher variance over the course of training or even the absence of any progress. DDPG and DBDDPG did not train with the modified hyperparameters. TD3 did train with the basic hyperparameter setting, but performed much better with our hyperparameters. The hyperparameters have been tuned for the unbootstrapped algorithms and remain unchanged when boostrapping is applied.

The network still consists of two dense layer, but with $\{\mathbf{256}$,$256\}$ nodes. Again the bold number represents the shared layer for our approach. The modified hyperparameters are the discount factor $\gamma = 0.99$, critic learning rate $l_{r,Q} = 8 \times 10^{-4}$, the exploration noise $\epsilon = 0.1$, the target update factor $\tau = 0.005$ and the size of the minibatch $\mathcal{B}$ with 256. The smoothing target noise is given by $\epsilon_t = 0.2$ and $\epsilon_{t,clip} = 0.5$. We train our agent with a bootstrapping degree of $N = 10$ and a masking probability of $p = 0.3$. The potential penalization factor β varies across the environments and needs to carefully tuned. However, mostly a value of $\beta = 10^{-6}$ is a good start for hyperparameter search.

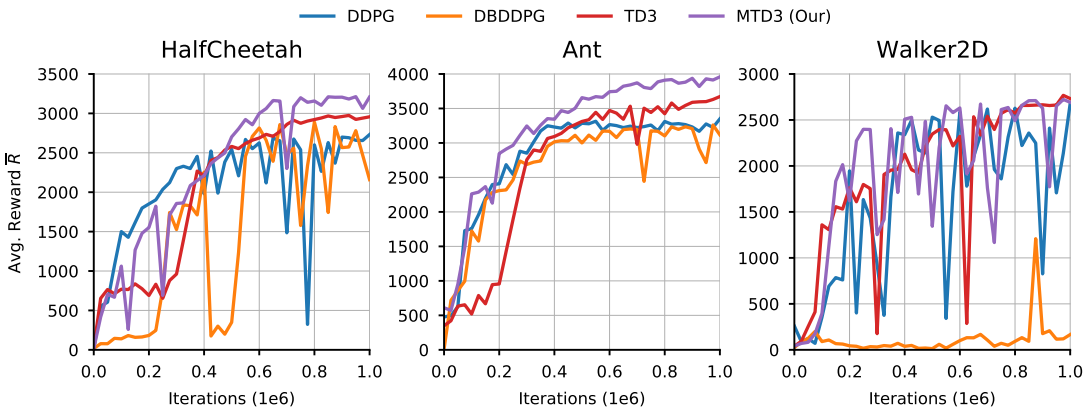

Figure 2: Averaged return over the course of training for a variety of low-level control tasks from the OpenAI gym [30] simulated with PyBullet [29]. As can be seen, enhanced exploration due to our approach clearly improves the performance of the agent.

4.2. Bootstrapped Performance

In this section, we evaluate the improved exploration by comparing the evolution of the received reward during the course of training. Each environment represents a low-level control task of a simplified robot. The received observation consists of the joint angles and joint velocities. The actions correspond to the applied torques, one for each joint. The reward is calculated based on the forward traveling speed and a fix alive bonus to consider the length of the episode. Additionally, a penalty for higher effort is applied, which consists of the torque magnitude and the impact forces on the ground. The episode ends and the environment is reset, when the policy leads the robot into an absorbing state or a maximum length of 1000 timesteps is reached. An absorbing state is defined as a state that the robot is unable to leave within the given action space. These robot environments first appeared in Schulmann *et al.* [31]. The underlying engine to simulate the kinematics of the robot and the contacts with the groundplane is PyBullet [29], while the environment is wrapped and accessed by the agent using Gym [30].

The baselines are represented by DDPG and TD3, while TD3 clearly outperforms DDPG. This is originated in the superior target function and stabilized critic optimization. Due to DBDDPG being based on DDPG, not only the absolute difference to our approach regarding the received reward is of interest, but also the difference in relative improvement towards unbootstrapped methods. Where absolute refers to the direct comparison and relative to the comparison of the improvement towards the unbootstrapped base. To reduce statistical drawbacks due to randomness in initialization regarding the comparability of all algorithms, each experiment is conducted four times for each algorithm under identical conditions and the best trials are compared in Fig. 2. The data is generated by evaluating the agent during the course of training and averaging the final return over multiple episodes.

Our approach achieves a clear increase in performance regarding the expected return. We are outperforming all of the other approach due to a more elaborated exploration. However, when no beneficial effect due to bootstrapping is noticeable, the overhead produced by our approach does not decrease the final performance and achieves the same results as TD3. This especially can be seen in the Walker2D environment, when compared to the effect of DBDDPG. In direct comparison DBDDPG, or DDPG in general, suffers from greater instability and therefore high variance over the course of training. Furthermore does DBDDPG not manage to increase the performance in comparison to DDPG, when the hyperparameters from DDPG are applied as shown in our experiments. Therefore, our approach not only outperforms DBDDPG in absolute measure

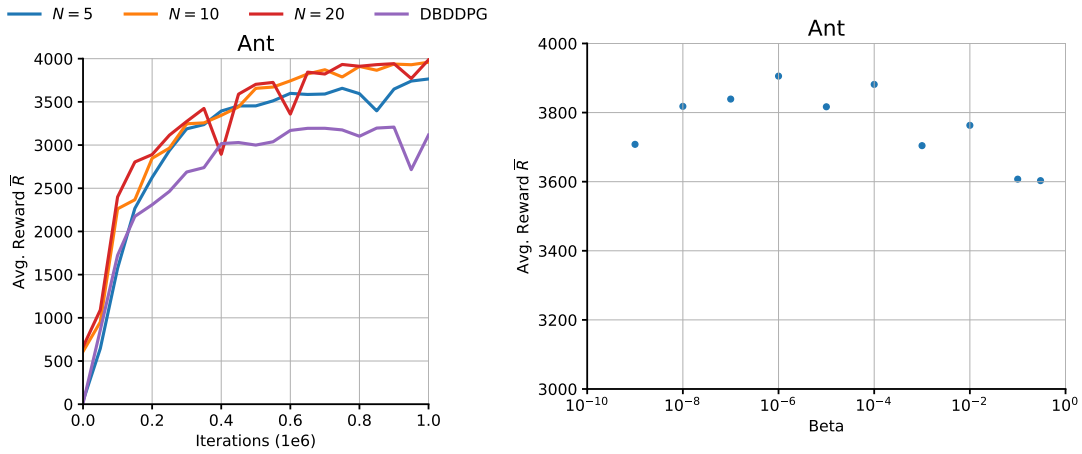

Figure 3: Illustration of the effect of the degree of bootstrapping (left) and the influence of the potential based penalty (right) for the environment Ant. As can be seen, with increasing N the beneficial effect due to enhanced exploration increases. The potential based penalty coefficient improves the expected return until it constraints the training to heavily.

but also in relative measure compared to the unbootstrapped methods. We are still improving the performance of the agent compared to TD3 even though the hyperparameters are optimized explicitly for TD3.

The exploration is especially supported by the potential-based penalty as it ensures a greater maintenance of the posterior distribution. This effect is visible in Fig. 3 on the right for $N = 5$. Each data point represents the mean reward over the last five evaluations given a certain beta. While only the penalty coefficient beta is varying, the average reward starts to increase and decrease after an optimal value of beta. While first being supportive towards the posterior distribution the penalty prevents optimization if too great. Another essential parameter is the degree of bootstrapping N, which increases the number of hypotheses and therefore increases the possible diversity. As Fig. 3 on the left shows, increasing the degree of bootstrapping further enhances the beneficial effect on the resulting performance. However, this effect saturates as can be seen by comparing the curves for $N = 10$ and $N = 20$. All values of N are superior over DBDDPG with $N = 5$.

A major limitation is the source of improvement itself. While exploration is heavily enhanced by uncertainty, at the same time there is no guarantee that each trial under identical condition will benefit from it. The potential-based penalty supports this effect, as it prevents a greater similarity between each hypotheses and emphasizes generality. Furthermore, is it unclear with the current state of the art, how a conjunction of the trained hypothesis, each represented by a neural network, could be possible. Since the local optima usually lay far apart, no strategy for weight combination is commonly known. Even if the optima lay close by each other, a naive combination of the weights will most likely result in a major decrease in performance. Therefore, we evaluate each actor greedily, as explained above, using the critic and act upon the expected to be most beneficial hypothesis, while the conjunction is put to future work.

4.3. Training- and Runtime Analysis

Another major contribution of our bootstrapping approach is the much faster training time in comparison to the current state of the art due to random policy sampling instead of greedy data collection. To evaluate this, we conducted a shortened training including 10,000 iterations and average the time per iteration. No evaluation takes place, therefore the averaged time includes one interaction with the environment and one optimization step of the neural networks. For the averaged inference time during evaluation the network generates 10,000 actions based on random

Table 1: Runtime (average and std. dev.), normalized by our approach for $N = 5$

Application	OUR, $\mathbf{N = 5}$	OUR, $\mathbf{N = 10}$	TD3	DBDDPG
Training	$\mathbf{1.0} \pm 0.129$	2.31 ± 0.139	0.609 ± 0.105	2.817 ± 0.682
Evaluation	$\mathbf{1.0} \pm 0.063$	1.982 ± 0.128	0.144 ± 0.014	3.9603 ± 0.189

states, while the interaction with the environment is neglected. The averaged time only includes the inference, not the sampling of the random states. All data is summarized in Tab. 1 and normalized by our approach with a degree of bootstrapping $N = 5$, represented by bold digits, to be easily comparable.

As expected, is TD3 in comparison the fastest in training, as least networks have to be optimized. Nevertheless is ours only slightly slower due to the efficient data collection, while the overhead is given by the increased number of actor heads that need to be optimized. In evaluation the overhead becomes more obvious, as we are searching greedily for the optimal action and quantify the performance of each actor using the critic. Our approach is still applicable in real time, since one inference for $N = 10$ takes on average less than 9 ms.

In comparison to DBDDPG, our approach is much faster in training and application. The quadratic cost in evaluation is already for $N = 5$ noticeable and especially the greedy collection of the data heavily slows down the training process in DBDDPG. Already for this degree of bootstrapping our approach is more performant regarding the expected return, while being faster in training and evaluation. Even for an increasing degree of bootstrapping our approach is faster, as is shown for $N = 10$.

5. CONCLUSION

In this paper, we presented a novel bootstrapping approach based on Thompson sampling applied to twin delayed deep deterministic policy gradient (TD3). Our approach trains and infers much faster than the current state of the art, which is DBDDPG [13], and still achieves a seriously improved performance. This especially results from applying the critic structure from TD3 instead of bootstrapping the entire actor-critic structure. Another speedup is achieved by sampling the actions during training based on Thompson sampling and not by greedily searching for the optimal action. To benefit from Thompson sampling, it is important to maintain the posterior knowledge inherited in our actor structure. We address this by adding a potential field based penalty, which induces high cost when the hypotheses agree on the same optimality. A well maintained distribution, given the posterior knowledge, naturally induces deep exploration when acted according to it during data collection.

We evaluated our approach in a variety of low-level control tasks, which strongly back up our claims. The experiments show that we outperform DBDDPG in all of the trained environments with a major decrease in computational cost regarding training and evaluation. Furthermore, we add a clear improvement in comparison to TD3 with the drawback of minor computational cost increase. This benefit is mainly caused by the improved exploration during data collection induced by sampling the current policy based on Thompson sampling, especially in combination with our potential field penalty constraint.

ACKNOWLEDGEMENTS

This work has partly been supported by the German Research Foundation under Germany's Excellence Strategy, EXC-2070 - 390732324 (PhenoRob).

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[2] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine (RAM)*, 1997.

[3] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[4] G. Ferrer, A. G. Zulueta, F. Cotarelo, and A. Sanfeliu, "Robot social-aware navigation framework to accompany people walking side-by-side," *Autonomous Robots*, 2017.

[5] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-Robot Interaction: Crowd-Aware Robot Navigation With Attention-Based Deep Reinforcement Learning," *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[6] V. Dhiman, S. Banerjee, B. Griffin, J. M. Siskind, and J. J. Corso, "A Critical Investigation of Deep Reinforcement Learning for Navigation," *CoRR*, 2018.

[7] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum Learning," in *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2009.

[9] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to Navigate in Complex Environments," *CoRR*, 2016.

[10] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," in *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2017.

[11] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2018.

[12] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, 1933.

[13] Z. Zheng, C. Yuan, Z. Lin, Y. Cheng, and H. Wu, "Self-Adaptive Double Bootstrapped DDPG," in *Proc. of the Intl. Jt. Conf. on Artificial Intelligence (IJCAI)*, 2018.

[14] J. Kulhánek, E. Derner, T. de Bruin, and R. Babuška, "Vision-based Navigation Using Deep Reinforcement Learning," in *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2019.

[15] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement Learning with Unsupervised Auxiliary Tasks," *CoRR*, 2016.

[16] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models," *arXiv preprint*, 2015.

[17] O. Zhelo, J. Zhang, L. Tai, M. Liu, and W. Burgard, "Curiosity-driven Exploration for Mapless Navigation with Deep Reinforcement Learning," *CoRR*, 2018.

[18] R. Houthooft, X. D. Chen, Y. M. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "Variational Information Maximizing Exploration," in *Proc. of the Conf. on Neural Information*

Processing Systems (NIPS), 2016.

[19] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2015.

[20] G. E. Hinton and D. van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," in *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, 1993.

[21] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems 24*, 2011.

[22] P. Henderson, T. Doan, R. Islam, and D. Meger, "Bayesian policy gradients via alpha divergence dropout inference," *CoRR*, 2017.

[23] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, "Deep Exploration via Bootstrapped DQN," *CoRR*, 2016.

[24] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2014.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *Intl. Conf. on Learning Representations (ICLR)*, 2016.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms." *CoRR*, 2017.

[27] E. Shech and E. Hatleback, "The material intricacies of coulomb's 1785 electric torsion balance experiment," July 2014.

[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[29] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.

[31] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

© 2020 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.