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

Application of Reinforcement Learning Methods

Group 19 - Final Project Report

Yannik Frisch · Tabea Wilke · Maximilian Gehrke

the date of receipt and acceptance should be inserted later

1 Introduction

We shortly present two state-of the art reinforcement learning algorithms, the *Deep Deterministic Policy Gradient* and the *Natural Actor Critic*. Both algorithms are evaluated on the Quanser Robots platforms on the simulated quanser systems platforms *BallBalancerSim-v0*, *CartPoleSwingShort-v0* and *Qube-v0*. We furthermore present the results of training both algorithms on the *BallBalancerSim-v0* and evaluating it on the pyhiscal *BallBalancerRR-v0* platform. Finally, we let a pretrained *Natural Actor Critic* agent continue learning on the physical version of *CartPoleSwingShort-v0*. We close with a discussion of the results.

Address(es) of author(s) should be given

2 Deep Deterministic Policy Gradient

The *Deep Deterministic Policy Gradient* approach [2] is an application of the *Deep Q-Learning* algorithm [3] to actor-critic methods [1] in combination with the *Deterministic Policy Gradient* [4]. It is a model-free and off-policy algorithm, learning a deterministic policy. A replay buffer with a total size of 1e6 samples is used to sample independently and identically distributed mini-batches, randomly selected to temporarily decorrelate them. Target networks are used, which are constrained to slow changes with rate τ to improve the stability of learning.

2.1 Evaluation on BallBalancerSim-v0

The Quanser Robots *BallBalancerSim-v0* environment consist of a plate whose angles can be controlled by the input actions. The goal is to balance a ball on the plate, receiving a maximum reward of 1.0 per time-step for balancing it in the middle of the plate. The environment ends after a maximum of 1000 time-steps.
We started our evaluations with using the same network structures for the actor and critic as [2] did. We used 2 hidden layers with 100 and 300 hidden neurons for the actor and the critic networks and their targets. The learning rates are also set to $\alpha_{actor} = 1e{-}4$ and $\alpha_{critic} = 1e{-}3$. In the left row of figure 2 one can find our first acceptable results. The discounting is set to $\gamma = 0.99$, we did small target updates ($\tau = 1e - 4$), used a mini-batch size of 64 and a total replay buffer size of 1e6. We slightly increased the noise to $\sigma_{OU} = 0.2$ and $\theta_{OU} = 0.25$ as the environments action space has an higher amplitude compared to the *Pendulum-v0*.
The algorithm did learn to balance the ball, but was not very stable, which can also be read from the learning process plot. To further increase the stability, we increased th mini-batch size used to sample from the replay buffer, and reduced the noise again. Using weight regularization did not seem to be helpful, so we set it to zero.
Figure 1 shows the training process where discounting is set to $\gamma = 0.2$ compared to $\gamma = 0.99$. One can see discounting is crucial to solve this environment.

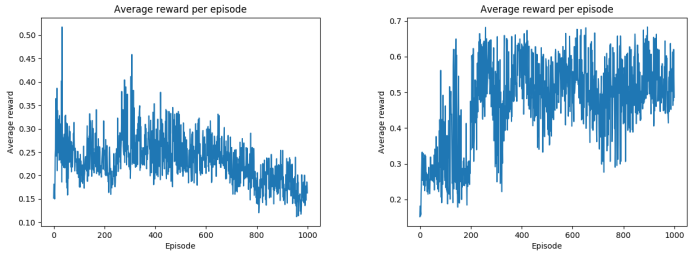

Fig. 1 The left figure shows the cumulative reward per episode during the training process with γ set to 0.2. The right one displays the process for $\gamma = 0.99$. Using discounting close to 1 was very important.

We tried to reduce the computational effort by only using a single hidden layer with 100 hidden neurons instead of two layers. The impact on the performance is shown in the middle row of figure 2. The learning suffered from instabilities, so we decided to weight the stability of using two hidden layers higher than the performance loss.
Our best results can be found in the right row of figure 2 where we set the OU action noise equal to the one used in the original paper with $\sigma_{OU} = 0.15$ and $\theta_{OU} = 0.2$. We used slightly harder updates with $\tau = 1e-3$, and achieved an average cumulative reward of about 650 for 25 episodes of evaluation. The learning process took about 3 hours for 1000 episodes. Further evaluations are needed to improve the training even more.

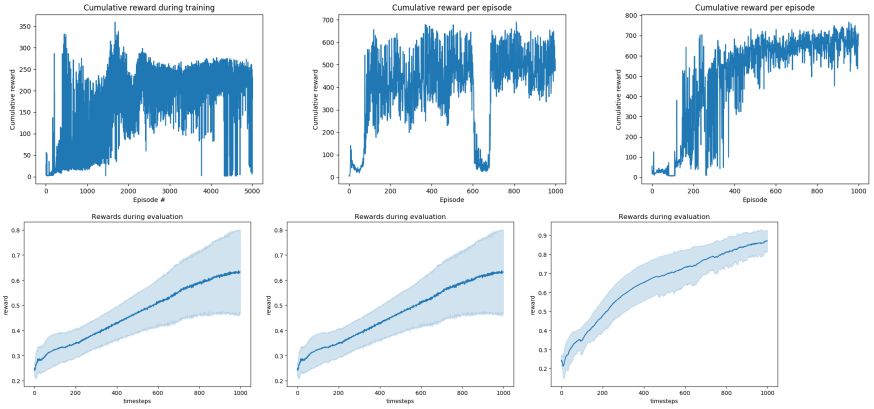

Fig. 2 The figure displays the training process in the first row and the evaluation results in the second row. Early learning successes can be found in the left column, while the middle column shows the influence of using only a single hidden layer. The right column gives the plots of our best training result.

2.2 Evaluation of the Pretrained Model on the Real Ball Balancer System

Evaluating our best model trained in simulation on the real ball balancer was not successful. The chosen actions were too big and the plate tilted. Also, it was not possible for us to reset the environment between the evaluation episodes. The result of a single episode of evaluation can be found in figure [x].

2.3 Evaluation on CartPoleStabShort-v0

The Quanser Robots *CartPoleStabShort-v0* environment consists of a movable car with a singular pendulum attached. The car can be controlled by input actions. The goal is to balance the pendulum, starting from a vertical position and the reward is depending on the angle, with a maximum of 2.0 per time-step for balancing the pendulum straight upright.
We achieved some progress using [x]. The results are displayed in [x].

2.4 Evaluation on Qube-v0

The Quanser Robots *Qube-v0* environment implements the Furuta Pendulum, which consists of a motor controlling one horizontal arm. One end of the joint is attached to the motor, the other end is attached to another vertical arm, which can only be controlled indirectly by controlling the first arm. More details about the Furuta Pendulum can be found in our paper about it.
The goal is to balance the second arm in upright position, receiving a maximum reward of 0.02 per time-step. The environment stops after 300 time-steps. We did not get any useful results re-using the parameters we found for *BallBalancer-v0*. Using an Ornstein Uhlenbeck seemed not to help to deal with with the local optima of the environment. Setting the randomness σ_{OU} too small resulted in to less exploration. Choosing a higher σ_{OU} resulted in better exploration but much less stability. Examples are displayed in figure 3.
No stable training process could be achieved. To address this issue we started using a consistent gaussian noise. Unfortunately this did also not help the training process, and seemed to be even more unstable. Figure 3 shows a result of using gaussian noise in the right plot. The algorithm seemed to learn well for a period of time, but tended to overwrite these progress. Lowering the target update rate τ did not help, nether did lowering the actor and critic learning rates.

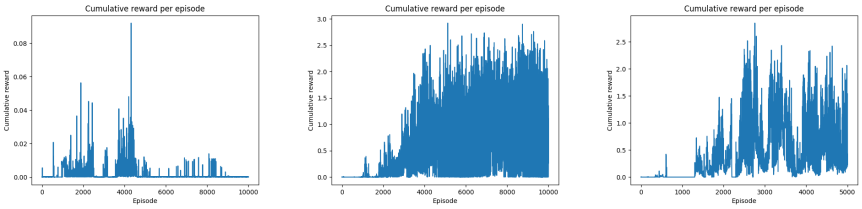

Fig. 3 Assuming the parameters from our best result on *BallBalancerSim-v0*, we achieved the performance shown in the left plot on *Qube-v0*. The middle plot shows the changes when using more OU noise, i.e. $\sigma = 3.2$. The right plot displays the result of training with a gaussian noise with mean zero and standard deviation $\sigma = 0.7$. The mini-batch size was also increased to 256.

Our experiments induced several things: Using a higher batch size seemed to increase the stability to a certain level, but also highly increased the computational effort. 256 seemed to be a good batch-size. The decisions harder or softer updates, controlled by τ, and the amount of action noise for exploration seemed to be very important but also very hard to tune. E.g. choosing $\tau = 1e-2$ resulted in an agent tending to overwrite what he already learned, while setting $\tau = 1e-4$ prevented him from learning anything. So we chose $\tau = 1e-3$, but this alone was not sufficient for effective learning. Reducing the action noise added to the actor output did also not help. Further evaluations are needed to find the right set of hyper-parameters for this environment.

3 Natural Actor Critic

3.1 Evaluation on CartPoleSwingShort-v0

3.2 Evaluation on the BallBalancerSim-v0

3.3 Evaluation of the Pretrained Model on the Real Ball Balancer System

3.3.1 Learning from the Physical Cart Pole System with pretrained parameters

4 Discussion

We implemented the DDPG and NAC algorithms and evaluated them on the Quanser Robots environments *BallBalancerSim-v0*, *CartPoleStabShort-v0* and the *Qube-v0*. Both algorithms were able to learn a well performing policy for the first two environments, where the NAC needed less computational time and was more sample efficient. Nonetheless, both algorithms did have troubles to learn a close to optimal policy for the *Qube-v0* environment. They suffered from a very difficult exploration / exploitation trade-off, often with stable

learning in the beginning which is overwritten later, or not having enough exploration drive to escape local optima. Further evaluation is needed to optimize the algorithms especially for this environment. Due to technical difficulties it was not really possible to evaluate our pretrained models on the real Quanser Robots systems. Further investigation is needed.

References

1. Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms. In: Advances in neural information processing systems, pp 1008–1014
2. 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
3. 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
4. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: ICML