

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 simulated Quanser Robots platforms *BallBalancerSim-v0*, *CartPoleStabShort-v0* and *Qube-v0*. We furthermore present the results of training both algorithms on the *BallBalancerSim-v0* and evaluating it on the physical *BallBalancerRR-v0* platform. Finally, we let a pretrained *Natural Actor Critic* agent continue learning on the physical version of *CartPoleStabShort-v0* and close with a discussion of the results.

2 Deep Deterministic Policy Gradient

The *Deep Deterministic Policy Gradient* approach [6] is an application of the *Deep Q-Learning* algorithm [7] to actor-critic methods [5] in combination with the *Deterministic Policy Gradient* [9]. It is a model-free and off-policy algorithm, learning a deterministic policy. We separated the learning process completely from the algorithm’s other components. The critic is updated using the temporal difference error, calculated from using the target networks, which are constrained to slow changes with rate τ to improve the stability of learning. The actor is trained by using the integrated update rule from [6] as it’s loss function. Both networks are addressed and can be modified in a separate file, same as the replay buffer, which is used to sample independently and identically distributed mini-batches, randomly selected to temporarily decorrelate them. The action noise to ensure exploration was also kept completely independent from the training algorithm as intended by [6] and is just added to the output of the actor. We clip this output to ensure it suits the environment.

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 two hidden layers with 100 and 300 hidden neurons for the actor and the critic networks and their targets. The learning rates are 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. Discounting is set to $\gamma = 0.99$, we did soft target updates with $\tau = 1e-4$, 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* trained on in [6].

The algorithm did learn to balance the ball, but was not very stable, which can also be read from the learning progress plot. To further increase the stability, we increased the 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.

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

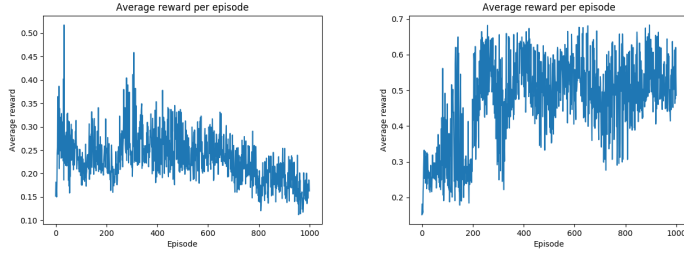


Fig. 1 The left figure shows the cumulative reward per episode during the training process with γ set to 0.2. The right one displays the process for $\gamma = 0.99$. Using discounting close to 1 was very important.

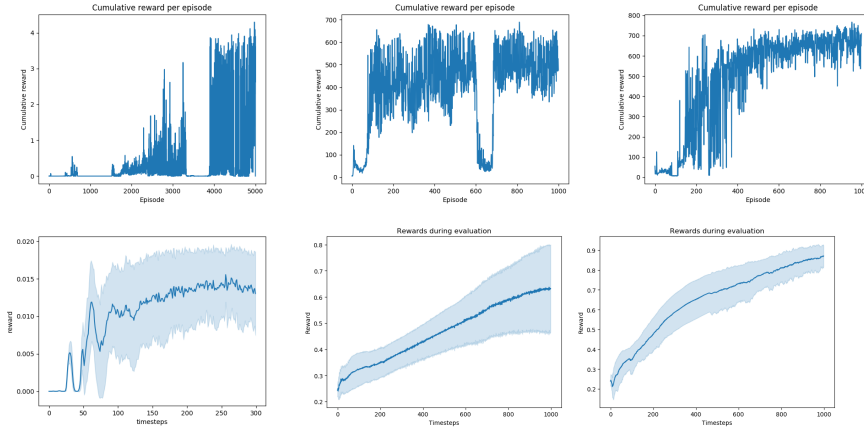


Fig. 2 The figure displays the training process in the first row and the evaluation results in the second row. Early learning successes can be found in the left column, while the middle column shows the influence of using only a single hidden layer. The right column gives the plots of our best training result.[EDIT: ALIGN PLOTS + LEFT PLOT IS WRONG (WRONG ENV...)]

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.

2.2 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-

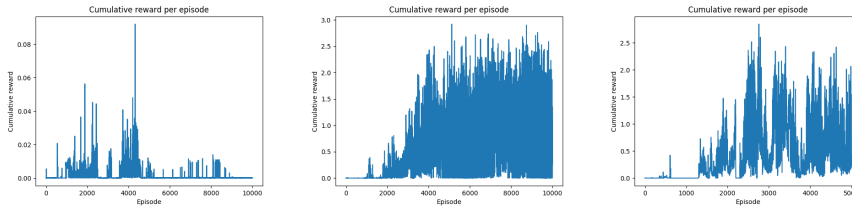


Fig. 3 Assuming the parameters from our best result on *BallBalancerSim-v0*, we achieved the performance shown in the left plot on *Qube-v0*. The middle plot shows the changes when using more OU noise, i.e. $\sigma = 3.2$. The right plot displays the result of training with a gaussian noise with mean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

step for balancing the pendulum straight upright.

We achieved some progress using [x]. The results are displayed in [x].

2.3 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 similar to the one used in citeplillicrap2015continuous, did not seem to help to deal with the local optima of the environment. Setting the randomness σ_{OU} too small resulted in too 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. 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

The *Natural Actor Critic (NAC)* [8] is a model-free, off-policy algorithm, which learns the optimal policy of a reinforcement learning problem by applying the natural gradient [1] on an actor-critic [10] approach. We implemented the episodic NAC [8] which takes a fixed number of steps (equals one or more trajectories) in the environment before applying updates to the actor and critic. The actor is an approximation of the policy and the critic is an approximation of the value function, which is used to determine the policy. In this case, we implemented both by using neural networks in tensorflow.

3.1 The Critic Network

For the critic, we introduced a neural network with a single hidden layer. The input layer equals the state dimension of the environment in use. It is fully connected to the hidden layer which again is fully connected to a single output node and uses a ReLu activation function [2].

To train the network, we calculated the discounted return for all states we visited since the last update and compared it to the approximation of the critic. As loss we took the least squares error between the two arrays and minimized it by applying an Adam optimizer [4].

Parameters: We used a hidden layer with 10 nodes and a learning rate of 0.1 for all environments. We tried to add more hidden nodes and experimented with the learning rates, but could not depict any significant improvements.

3.2 The Actor Network

The actor network has an input and an output layer which are fully connected and run through a softmax activation function. The input layer matches the dimensions of the state, the output layer matches the dimensions of the action space. If we have multidimensional actions, each permutation is represented in the action space. If we put an observation into the network, the softmax layer takes care that we end up with a probability for each action. To select which action to take next, we stochastically sample one action w.r.t these probabilities.

To train the network, we use the critic to calculate the advantages for all states we visited since the last update and feed it along with the states and actions into the actor network. Then we compute the cost function, it's gradient w.r.t the actor network parameters and the inverse of the Fisher information matrix. To get the inverse of the Fisher information matrix, we

applied singular value decomposition [3]. Finally, we multiply the cost function gradient, the inverse of the Fisher information matrix and a learning rate and add the resulting term to the old actor network parameters.

Parameters: We trained all of the environments without a hidden layer and with a learning rate of 0.001. We tried to add a hidden layer and change the learning rate, but did not get any improved results.

3.3 Evaluation on the BallBalancerSim-v0

The NAC algorithm learned to solve the ball balancer extremely well. Already with 300 updates and 2000 steps per update, the ball balancer manages to hold the ball on the platform and navigate it slowly into the middle. The more steps the environment was allowed to do between updates, the greater became the cumulative reward. The ball balancer learned the faster it rolls the ball into the middle, the higher is the reward. We managed to run a simulation with 1000 updates 7000 steps, which returned an cumulative average trajectory reward of 600. We believe that we would get an even higher average cumulative trajectory reward with even higher steps per update.

Nonetheless, we can see in figure ?? that even for our best run, the average reward per timestep increased very slowly. This reflects that the environment moves the ball only very slowly to the middle of the ball balancer, where the rewards get higher. Our idea was to apply a basis function to the rewards which pulls apart the high rewards from the low rewards even further, but did not have any success with this idea yet.

The number of updates was not as important as the step size between updates. Figure ?? shows that already after 300 steps we reached trajectories with cumulative rewards of 600. But the longer we update the more stable the algorithm got, meaning that nearly all trajectories reached a reward of approximately 600 after 800 updates, whereas after 300 updates the cumulative reward per trajectory still fluctuated a lot.

Regarding the discounting rate when calculating the discounted return for a state, we used 0.99 or 1 which work very well and is very reasonable for the specific problem.

3.4 Evaluation on CartPoleStabShort-v0

We managed to get a nearly perfect result for the cart pole balancing task. The average cumulative trajectory reward of our model is 19987.75, which is just slightly less than the possibly maximum of 20000. We used 300 updates with 2000 steps each and a discount factor of 0.99 to achieve these results.

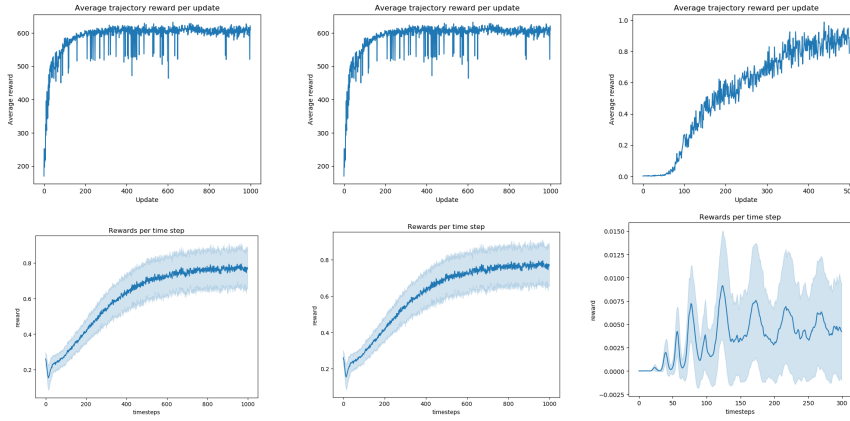


Fig. 4 Hallo mein name ist p

3.5 Evaluation on Qube-v0

The results we achieved for the Qube were not very good. Our best run had an average cumulative trajectory reward of 0.92278 which is not very high. The Qube manages to swing up the pendulum from time to time, but does not hold it in an upright position.

4 Evaluation on real systems

4.1 Evaluation of Pretrained Models on the Real Ball Balancer System

Evaluating our best DDPG model trained in simulation on the real ball balancer was not successful. The chosen actions were too big and the plate instantly tilted. Additionally, it was not possible for us to reset the environment between the evaluation episodes.

Also with our best pretrained NAC model, the actions of our policy seemed to big. We tried to execute a model with 10 times smaller actions, but got the same disappointing results.

4.2 Evaluation of the Pretrained models on the Real Cart Pole System

Evaluating our best NAC model with nearly perfect rewards on the real cart pole system was not successful. The actions of the cart had to much power, which is why we clipped the actions from originally $[-24, 24]$ to $[-6, 6]$, but the cart was still far to jumpy to hold the stick upright.

5 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. Amari SI (1998) Natural gradient works efficiently in learning. *Neural computation* 10(2):251–276
2. Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp 315–323
3. Golub G, Kahan W (1965) Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2(2):205–224
4. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*
5. Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms. In: *Advances in neural information processing systems*, pp 1008–1014
6. 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:1509.02971*
7. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*
8. Peters J, Schaal S (2008) Natural actor-critic. *Neurocomputing* 71(7-9):1180–1190
9. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: *ICML*
10. Sutton RS, Barto AG, et al. (1998) *Introduction to reinforcement learning*, vol 135. MIT press Cambridge