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

Lab Report - Reinforcement Learning Project

Janosch Moos · Kay Hansel · Cedric Derstroff

Received: date / Accepted: date

Keywords Furuta Pendulum · Cartpole · Inverted Pendulum · Ball Balancer · Classical Control · Artificial Intelligence · Reinforcement Learning · NPG · NES

1 Introduction

The field of Reinforcement Learning is progressing further and further. A lot of algorithms have been published over the past years using various approaches to solve black box problems [9]. For this project, we have been assigned the task to implement the Natural Policy Gradient (NPG) [6, 5] and the Natural Evolution Strategies (NES) [9]. To evaluate their performance we also have been assigned two platforms, the Cartpole and the Furuta Pendulum. Three tasks have to be solved on these platforms including the Cartpole swing up, the Cartpole double pendulum stabilization and the Furuta pendulum swing up [4]. We will discuss these platforms and tasks further in subsection 1.2 to show how they work. In this report we will shortly introduce the algorithms, platforms. Afterward, we will present our results and findings.

1.1 Algorithms

Reinforcement Learning can be defined by various different optimization problems. One of them is to use a trajectory-based approach

$$J(\pi) = \int_{\mathbb{T}} p^{\pi}(\tau) r(\tau) d\tau. \tag{1}$$

J. Moos
E-mail: janosch.moos@stud.tu-darmstadt.de

C. Derstroff
E-mail: cedric.derstroff@stud.tu-darmstadt.de

K. Hansel
E-mail: kay.hansel@stud.tu-darmstadt.de

By formulating the derivatives and applying the logarithm likelihood trick we can derive the "vanilla gradient" or REINFORCE algorithm [10]

$$\nabla_\theta J(\pi) = \int_{\mathbb{T}} p^\pi(\tau) \nabla_\theta \log p^\pi(\tau) r(\tau) d\tau \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p^\pi(\tau_i) r(\tau_i). \tag{2}$$

If we now add a constraint to regularize the gradients we can rewrite the optimization problem as

$$\begin{aligned} \max_{\delta\theta} \hat{J}(\theta + \Delta\theta) &= J(\theta) + \nabla_\theta J^T \Delta\theta \\ s.t.\ D(p_{\theta+\Delta\theta} || p_\theta) &= \Delta\theta^T F(\theta) \Delta\theta \leq \epsilon \end{aligned} \tag{3}$$

with the Fisher information matrix F. This leads to the natural gradient $F^{-1}\nabla_\theta J$ [6]

$$\Delta\theta = \alpha F^{-1} \nabla_\theta J = \sqrt{\frac{\delta}{\nabla_\theta J^T F^{-1} \nabla_\theta J}} F^{-1} \nabla_\theta J. \tag{4}$$

As a first setup, we implemented the NPG as described in [6] which however led to a lot of problems while calculating the Fisher information matrix (FIM). Not only is calculating the FIM computationally expensive, but it can also result in uninvertible matrices [3]. To get around these problems we decided to use conjugate gradients to compute the natural gradient [3]. To further increase the efficiency we also added importance sampling, which brought us from NPG to the Trust Region Policy Optimization (TRPO) [7, 1]. In this context, we switched to neural network based gaussian policy. To prevent confusion we will still denote to this as NPG in this report.

In contrast of the NPG, the algorithms of the NES family utilize a search distribution over the parameters θ. Each episode a batch of sample parameters z is drawn from this distribution. These samples are used to run a trajectory which is evaluated with a fitness function $f(z)$. We can formalize this as an optimization function

$$J(\theta) = \int f(z) \pi(z|\theta) dz \tag{5}$$

which leads to an estimate of the search gradients from samples $z_1 ... z_\lambda$

$$\nabla_\theta J \approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(z_k) \nabla_\theta \log \pi(z_k|\theta) \tag{6}$$

with population size λ [9].
The first version we implemented was the Canonical NES [9]. Similar to the NPG it adds a constraint to the optimization to regularize the gradients as in Equation 3. However as already described above we ran into problems while calculating the FIM. To avoid the same struggles as in the NPG we decided to switch to another version of the NES called Separable NES (SNES), which changes the update complexity from quadratic to linear [9].

1.2 Platforms

The algorithms have to be tested and evaluated on at least three different tasks. Two of these, the Cartpole swing up and the double pendulum stabilization, are executed on the Cartpole platform. The third task runs on the Furuta Pendulum [4].

The Cartpole is a fairly simple mechanical system. It consists of a sled which can move left and right on a rail. Connected to the sled is a stiff rod swinging freely whenever the sled moves [2]. There are several variations of tasks that can be performed on the Cartpole platform ranging from simple stabilization of a single rod to more complex tasks like stabilizing two, three or even four rods stacked on top of each other connected by free links. An even more challenging task is the swing up. Starting in a hanging position the rod has to be swung up and stabilized afterward. The complexity can be increased further by stacking more rods similar to before.

The Furuta Pendulum is a rotary inverted pendulum invented by Katsuhisa Furuta and his colleagues in 1991 [4]. It is an underactuated, non-linear system with a stiff rod attached to a single motor by a rigid link. The motor rotates the rod in a horizontal plane. Attached to the other end of the rod is a free link with a second rod perpendicular to the first. When the motor moves the first rod, the second one rotates in a vertical plane always perpendicular to the first rod. Compared to the Cartpole system it provides multiple advantages such as using less experimental space and having fewer uncertainties in the mechanical transmission system [4]. The task here is to use the rotation of the motor to swing up the second rod and stabilize it. Similar to the Cartpole the complexity can be increased by stacking more rods with free links onto the second one in the same direction.

2 Results

During our experiments, we observed that there is a huge difference in the policy behavior between the simulation and physical system. Often there will be policies running flawlessly in simulation but are unable to solve the problem even closely in reality. Further, NPG and NES behave quite differently during their training. Thus, in order to present and discuss our findings, we will split this section into two parts, Simulation and Physical System.

2.1 Simulation

Training a policy with the NPG turned out to be quite difficult. The biggest hurdle is finding suitable hyperparameters. Figure 1 shows the training process of the NPG on the left and NES on the right with two different sets of hyperparameters each. For the NPG we chose to change the discount factor γ, however, this is only one of many hyperparameters that can be changed in order to solve the optimization problem.

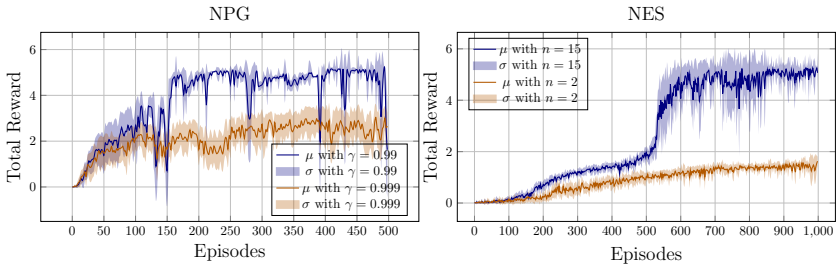

Fig. 1: Comparison of the training process with different hyperparameters. The plots show the huge impact even a single hyperparameter can have on the training process.

The most important parameters are:

- Trust Region δ:
  δ regulates the step size of the policy update which can lead to faster convergence. However, this can also result in worse performance because the algorithm may not converge to a global optimum and instead jumps among suboptimal solutions. We also observed that for varying δ we can get large jumps in performance during the training process as evident in the top left of Figure 2
- Discount Factor γ:
  γ determines the focus to short or long term rewards. Increasing the value also increases the importance of long term reward and vice versa [8].
- λ:
  λ is used to realize a tradeoff between bias and variance in the advantage estimation [8].
- Number of rollouts n:
  A higher number of rollouts per episode decreases the variance between episodes as the distribution over possible starting positions is averaged more realistically.
- Policy π:
  The policy maps states to actions. As such depending on the tasks to solve different policy classes can be applied [6]. For our implementation, we only use a neural network based gaussian policy, which not only the NPG, as implemented at this point, is efficient for, but also makes feature construction unnecessary. For our project, we decided to use the Adam optimizer as well as the tanh activation function, however, the activation function can also be changed if needed. The main difficulty is to find the right network size.
- Baseline b:
  The baseline estimates the state-value function. Possibly every regression algorithm can be used to realize the estimation provided suitable features are used. To simplify the implementation, we decided to represent the baseline with a neural network as well. In order to increase its performance, we added batch normalization. As such the baseline itself has multiple parameters that can be changed. The network size and number of epochs are our main concern. The batch size is set to 64 as default but can be adjusted as well as the learning rate

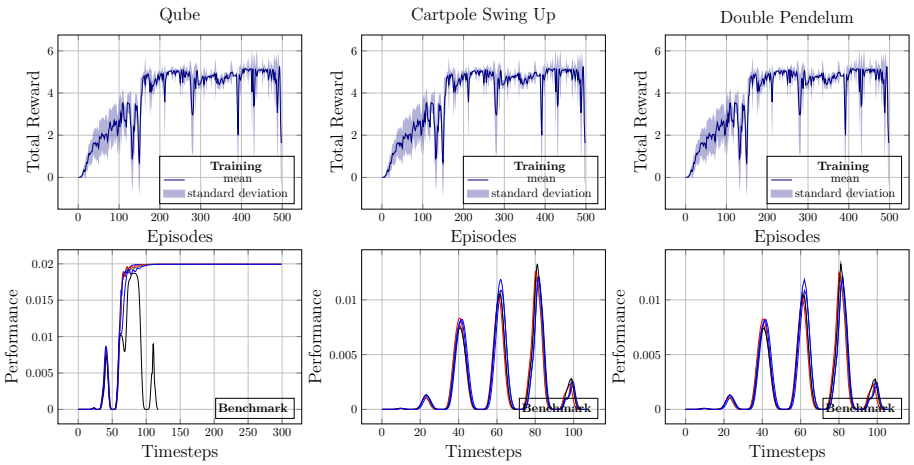

Fig. 2: This figure shows the training process of the NPG as well as a benchmark with the final policy on each of the three tasks described in section 1. "Qube" is referring to the Furuta Pendulum.

which is set to 10^{-3}. The optimizer and activation is the same as in the policy network. Despite fixing the optimizer, we noticed during our experiments that the optimizer has a huge impact. Adam behaves very differently to e.g. the SGD.

Figure 2 shows our final results on the given tasks using the NPG. We recognized that the most important parameters are γ and λ of the advantage function. We have had a lot of trouble finding suitable parameters for the given tasks and, unfortunately, have not managed to solve the Cartpole swing up.
Another important insight is provided by the benchmark plots, which are shown in the lower row of Figure 2. Each plot illustrates ten of the 100 runs of the benchmark test. It is important, depending on the environment, to perform a representative amount of runs, because the starting position can vary, which might lead to false positive results as pictured in the bottom left plot in Figure 2. During one of the runs the simulation was terminated early because the robot hit the restriction in the movement space and did not manage to perform the swing up. To ensure the best performance the exploration was set to zero for the benchmarks.
In the produced experiments we have even encountered policies whose exploration converged to almost zero and thus the total reward of the last training episodes was close to the benchmark performance. This is the desired behavior of gaussian policies, which should converge to an optimal policy without further exploration after sufficient training with suitable hyperparameters (see Figure 3 Double Pendulum).

As already mentioned in section 1 with the Canonical NES we had similar problems as with the first version of the NPG. Inverting the FIM often resulted in singular matrices, but in addition, both were very slow. In the case of the NES, we decided to switch to the SNES, which turned out to be a huge success. Figure 3 illustrates our final results with the SNES on all tasks. We managed to

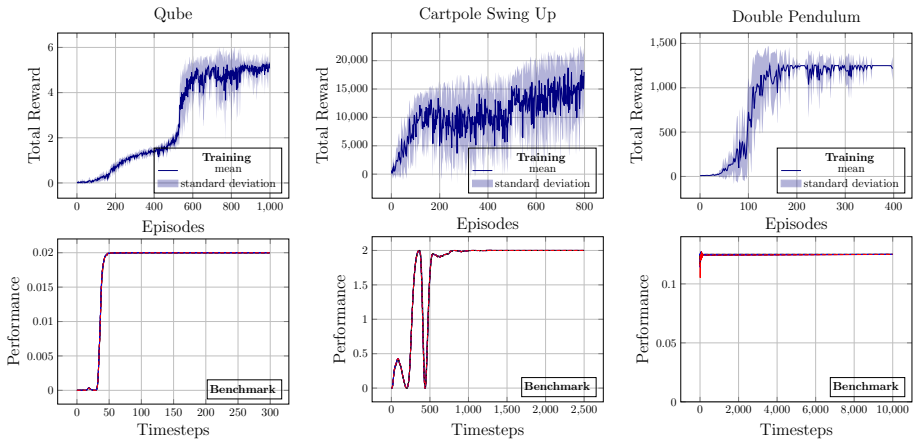

Fig. 3: Final results for given tasks using SNES

solve all platforms due to the fact that the SNES compared to the NPG has fewer hyperparameters that need to be adjusted. The only ones being:

- Population size λ:
  The population size λ defines the number of samples which are drawn from the search distribution for each episode. For all our solutions we used the calculation for λ suggested by [9].
- Number of rollouts n:
  In the case of the NES, this parameter does not define the number of rollouts per episode but rather the rollouts performed for each policy evaluation. Thus, for each episode $\lambda \cdot n$ simulations are executed while resetting the environment seed between each sample to a random value chosen at the beginning of each episode. Increasing the number of rollouts decreases the variance during rollouts similar to the NPG. Our experiments have shown that for most platforms and tasks $n = 2$ turned out to be enough to find good solutions. To solve the Furuta Pendulum, however, we had to use $n = 15$ as it was much more unstable than the Cartpole These settings have also been used to produce the plots in Figure 3.
- Policy π:
  Similar to the NPG the policy can be chosen arbitrarily depending on tasks and setup. We decided to use the greedy version of the policy we used for the NPG as the NES already incorporates exploration in the search distribution over the policy parameters. Additionally, we avoid the same problems using a neural network as described for the NPG. As such we have the same adjustable parameters for the NN, which however turned out not to be as impactful as in the NPG. For all tasks, we used a neural network with a single layer and 10 nodes and the default tanh activation function.
- Fitness function $f(z)$:
  The fitness function is crucial to evaluate the performance of the sample policies π_z. There a many possible options to choose from. We decided to go with one of the easiest by using $f(z) = \sum_{k=1}^{n} \sum_{i=1}^{T} r_{k,t}$.

In the end the SNES was not only easier to implement but also much easier to handle and set up. It has a lot less crucial hyperparameters that need to be adjusted and in addition, it is also much more invariant to small changes in the hyperparameters. We have observed that even though the SNES takes much longer than the NPG to reduce its exploration to near zero it manages to converge much faster to a policy of high quality.

2.2 Physical System

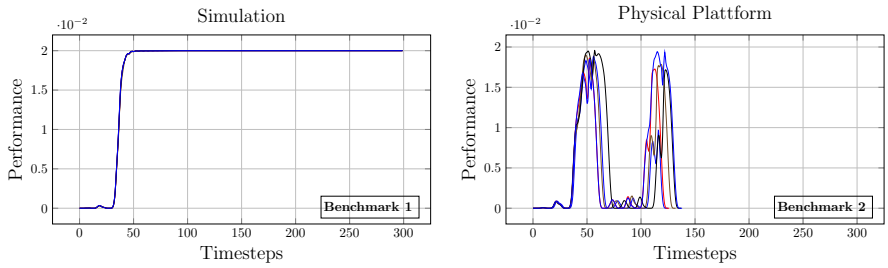

Fig. 4: Trained policy using NES in simulation and applied to physical Qube

Learning a policy in simulations is only the first step. In the end, we need to control a real physical system. Anyhow, applying a policy trained in simulation to a physical system can lead to tremendous drops in performance. Figure 4 pictures this drop in performance really well. As can be seen, the benchmark ran flawlessly in the simulation while crashing spectacularly on the physical system. We got a similar result for the NPG on the Furuta Pendulum. One reason we noticed was that the state space was slightly smaller in reality due to an obstacle mounted on the Qube to prevent damage when using suboptimal controllers. In order to apply policies without having this issue, we decided to clip the state space of the simulation to $[-\pi, \pi]$ and retrained the policies with both algorithms.

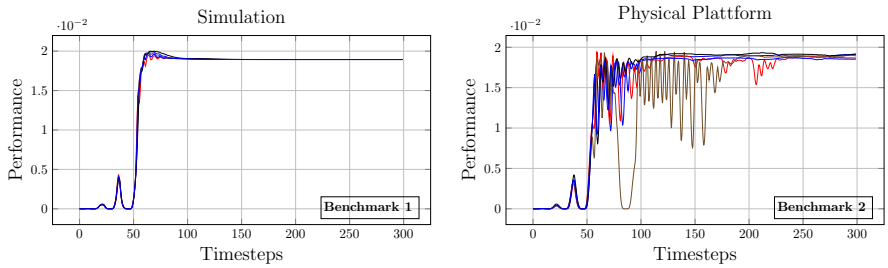

Fig. 5: Policy performance using NPG and clipped state space

Applying the new policy learned with the NPG worked surprisingly well. Figure 5 displays the benchmark results for our policy trained in the new state space.

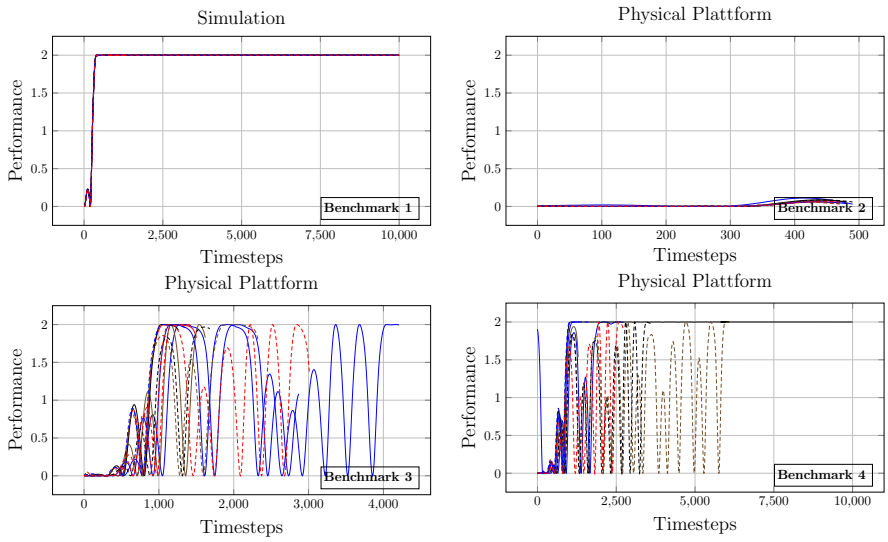

Fig. 6: Tuning of new NES policy on real Cartpole

It is evident that the drops in performance were less sever than before. The NPG even managed to perform the swing up and stabilization of the Furuta Pendulum in all benchmark runs. However, when applying the NES we recognized that this is not the case for all policies as the NES policy did not perform a single swing up. Instead, with this particular NPG policy, we got lucky as it was not as efficient during the swing up period as the others. In Figure 4 and 5 we notice the difference in the swing up of the simulation. The new NPG policy took longer to perform the swing up and used two instead of one swing before getting into an upright position while the new NES policy performed similarly to before as in Figure 4. On that basis we conclude that most policies are trained to a point were they perform unrealistically good and thus are overfitted to the simulation. So in order to train a good controller for real systems we can either train a policy in simulation and tune it on the physical system or train on the physical system from the start. Due to limited time we only used the first method. Though while tuning the new NES policy on the real system we encountered a huge problem. Every few runs the system crashed without stopping the training process. Instead, when crashed, each run ended after few steps with almost no reward which made it nearly impossible to improve the policy. Due to this fact, we decided to try solving the real Cartpole swing up with the NES. Similar to the Qube the NES performed nearly perfectly in simulation and its performance dropped on the real Cartpole. Figure 6 shows the proccess of applying the NES to the Carpole in the first row while the second illustrates the improvement of the policy after 30 and 60 episodes. To prevent damage to the system we have clipped the actions to $[-6, 6]$. Additionally, the state space of the Cartpole was to large for the real system, thus, we decided to adjust the state space as well. We also managed to solve the Ball-Balancer-v0 and Levitation-v0 with the same parameters using the NES, but did not include the results due to a lack of space.

References

1. Ankur Mohan: Efficiently Computing the Fisher Vector Product in TRPO (2018). URL http://www.telesens.co/2018/06/09/efficiently-computing-the-fisher-vector-product-in-trpo/
2. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics **SMC-13**(5), 834–846 (1983). DOI 10.1109/TSMC.1983.6313077. URL http://ieeexplore.ieee.org/document/6313077/
3. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking Deep Reinforcement Learning for Continuous Control. CoRR **abs/1604.0** (2016). URL http://arxiv.org/abs/1604.06778
4. Furuta, K., Yamakita, M., Kobayashi, S.: Swing up control of inverted pendulum. In: Proceedings IECON '91: 1991 International Conference on Industrial Electronics, Control and Instrumentation, pp. 2193–2198. IEEE, Kobe, Japan, Japan (1991). DOI 10.1109/IECON.1991.239008. URL http://ieeexplore.ieee.org/document/239008/
5. Kakade, S.: A Natural Policy Gradient. In: NIPS'01 Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, pp. 1531–1538 (2001)
6. Rajeswaran, A., Lowrey, K., Todorov, E., Kakade, S.: Towards Generalization and Simplicity in Continuous Control (Nips) (2017). DOI 10.1016/j.acra.2014.04.006. URL http://arxiv.org/abs/1703.02660
7. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust Region Policy Optimization. CoRR **abs/1502.0** (2015). URL http://arxiv.org/abs/1502.05477
8. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-Dimensional Continuous Control Using Generalized Advantage Estimation. Journal of Materials Chemistry B **6**(7), 1035–1043 (2015). DOI 10.1039/C7TB02772A. URL http://xlink.rsc.org/?DOI=C7TB02772Ahttp://arxiv.org/abs/1506.02438
9. Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., Schmidhuber, J.: Natural Evolution Strategies. Journal of Machine Learning Research **15**, 949–980 (2014). URL http://jmlr.org/papers/v15/wierstra14a.html
10. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8**(3), 229–256 (1992). DOI 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696