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 blackbox problems [11]. For this project we have been assigned the task to implement the Natural Policy Gradient (NPG) [9,8] and the Natural Evolution Strategies (NES) [11]. 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 [5]. We will discuss these platforms and tasks further in 1.2 to show how they work. In this report we will shortly introduce the algorithms and platforms and 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

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

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 F being defined to be

$$F = \frac{1}{T}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)^T \tag{4}$$

This leads to the natural gradient in [9]

$$\Delta\theta = \alpha F^{-1}\nabla_\theta J = \sqrt{\frac{\delta}{\nabla_\theta J^T F^{-1} \nabla_\theta J}} F^{-1}\nabla_\theta J \tag{5}$$

As a first setup we implemented the NPG as described in [9] which however lead to a lot of problems while calculating the fisher information matrix (FIM). Not only is calculating the fisher matrix very computationally expensive, it can also result in uninvertable matrices [4]. To get around calculating the FIM directly we decided to use conjugate gradients to compute the natural gradient [4]. To further increase the efficiency we also added importance sampling through which we basically ended up implementing the Trust Region Policy Optimization [10,1]. In this context we switched to neural network based policy representing a gaussian distribution. To prevent confusion we will still denote to this as NPG in this report.

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

$$J(\theta) = \int f(z)\pi(z|\theta)dz \tag{6}$$

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

$$\nabla_\theta J \approx \frac{1}{\lambda}\sum_{k=1}^{\lambda} f(z_k)\nabla_\theta \log \pi(z_k|\theta) \tag{7}$$

with population size λ [11].
The first version we implemented was the Canonical NES [11]. 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 devided to switch to another version of the NES called Seperable NES (SNES) [11].
ADD EXPLANATION OF SNES

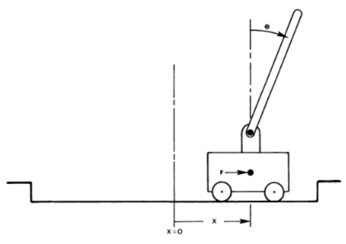

Fig. 1: Cartpole Model [2]

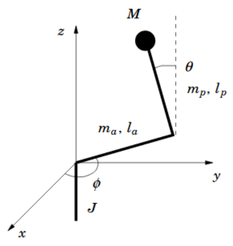

Fig. 2: Furuta Pendulum Model [6]

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 [5].

The Cartpole is a fairly simply mechanical system. It basically is a sledge sitting on a rail which can move left and right. Connected to the sledge is a stiff rod swinging freely whenever the sledge moves [2]. There are several variations of tasks that can be done 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. Even more complex is the swing up. Starting in a hanging position the rod has to be swung up and stabilizing afterwards. The complexity can be increased further by stacking more rods similar as before.

The Furuta Pendulum is a rotary inverted pendulum invented by Katsuhisa Furuta and his colleagues in 1991 [5]. 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 advantage such as using less experimental space and having fewer uncertanties in the mechanical transmission system [5]. The task here is to use the rotation of the motor to swing up the second rod and stabalize it. Similar to the cartpole the complexity could be increased by stacking more rods with free links onto the second one in the same direction.

2 Results

There is a huge difference between simulating a system and running the physical platform. Often there will be policies running flawlessly in simulation but are unable to solve the problem even closely in reality. Further, the algorithms behave quite differently during their training. Thus, in order to present and discuss our findings we will split this section in two parts, Simulation and Physical System.

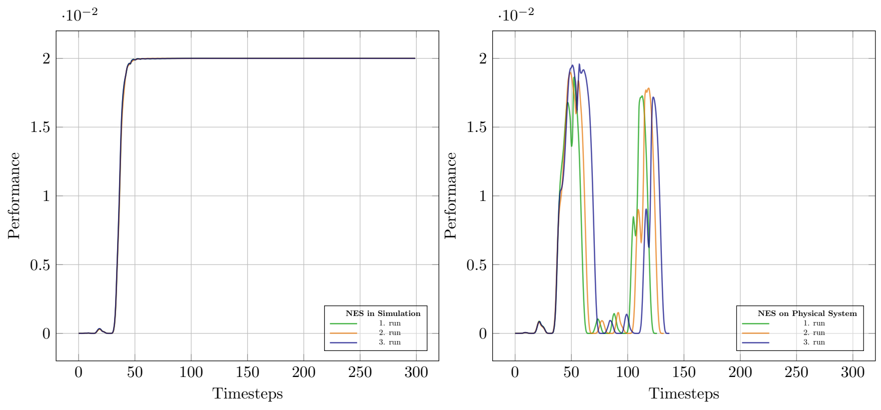

Fig. 3: This figure is a placeholder showing our struggles

2.1 Simulation

Training with the NPG turned out to be quite difficult. The biggest hurdle is finding suitable hyperparameters. Figure 3 shows the training process of the NPG with two different sets of hyperparameters. For these plots we chose to change INSERT PARAMETER HERE, however this is only one of many hyperparameters that can be changed in order to solve the optimization problem. The most important of them are:

1. δ:
   Increasing δ increases the step of the policy update which can lead to faster convergence. However this can also result in worse performance because the algorithm does not converge and instead jumps between suboptimal solutions.

2. γ:
   γ can be used to set more focus to short or long term rewards. Increasing the value also increases the importance of long term reward and vice versa [7].

3. λ:
   λ is used to realize a tradeoff between bias and variance in the advantage estimation [7].

4. Number of roll outs:
   Increasing the number of roll outs per episode decreases the variance during roll outs.

5. Policy:
   The policy is an important factor. Depending on the tasks to solve different policies can be applied [9]. For our implementation we only use a neural network representing a gaussian distribution. Thus we can only change the network size. The activation function is fixed as *tanh*, but still the size itself is

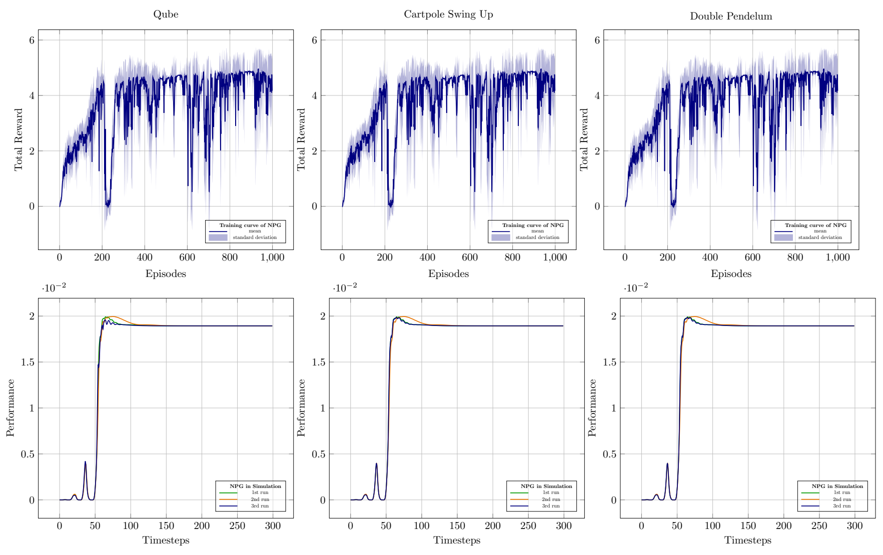

Fig. 4: This figure shows the training process of the NPG as well as a benchmark with the final policy on each of the three tasks described in section 1. "Qube" is refering to the Furuta Pendulum.

already a huge factor.

6. Baseline:
   The baseline, estimating the state-value function, is also represented with a neural network. DISCUSS size, epochs, batches, optimizer

Figure 4 shows our final results on the given tasks using the NPG. All of the graphs were created during simulations, which is easier in most cases because of simplifications that are usually made in the derivations of the dynamics in such systems [3]. As such the simulations run more smoothly and can even end up with unrealisticly good policies. We will discuss this more in subsection 2.2. Still, we were only able to solve the Qube with the NPG. Based on the fact that we managed to solve the Qube, we assume that the algorithm was implemented correctly and we were only unable to find suitable parameters as there are a lot that can change the training process tremendously which we already described above.
Further as Figure 4 shows the NPG can be quite unstable during its' training process. At around 200, 600 and 700 episodes there are huge jumps in the performance. These jumps can be reduced or even be prevented by reducing the learning rate δ, which in contrast also decreases convergence speed or might even cause the algorithm to get stuck in poor optima.
Another important insight are the benchmark plots in Figure 4. On these plots always only three runs are shown to increase visability. In reality we did around 100 runs for each policy to verify its' performance. This was necessary because, depending on the environment, the starting position can vary. This might lead to false positive results when to few runs are performed, which can also be seen in the plots of the Cartpole tasks in Figure 4. INSERT WHAT CAN BE SEEN

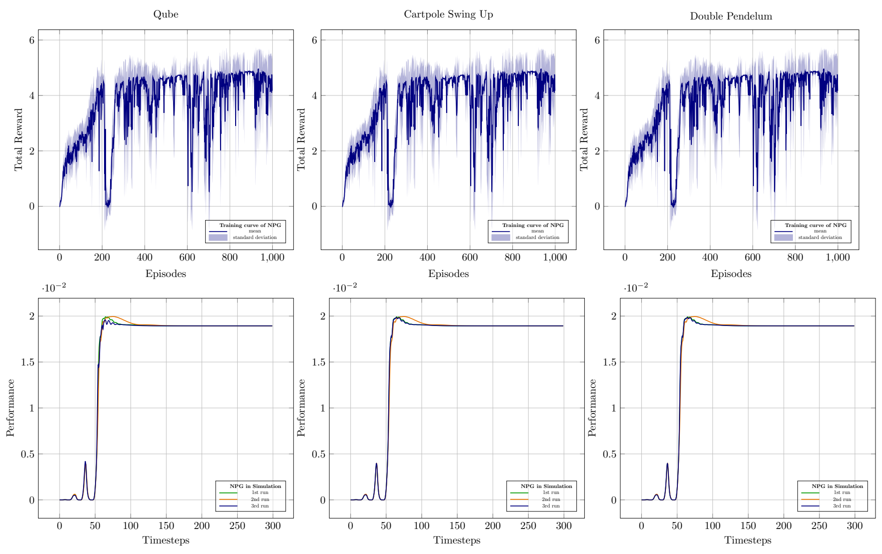

Fig. 5: This figure shows the performance of a policy trained in simulation. On the left you can see three runs done in simulation and on the right the same policy applied to the physical system

ON THE PLOTS. In contrast the well trained policy performed well in all runs. During these benchmark the policies behaved greedily to ensure the best possible actions where executed.
During our tests, which are not shown as Figures in this report we have also seen policies converge so well that almost no exploration was left and thus the performance in the last few episodes of the training process was almost similar to the benchmark performance. This outcome is the optimal situation as in theory a gaussian policy like the one we used for this project should converge to an optimum without further exploration if it is trained long enough with suitable hyperparameters.

As already described in section 1 the Canonical NES had similar problems as the first version of the NPG. Inverting the FIM often resulted in singular matrices, but in addition both were very slow. For the NPG we advanced towards TRPO in order to solve this issue. In case of the NES we decided to switch to the SNES, which turned out to be a huge success. Figure 5 shows our results of the NES on all tasks. We managed to solve all platforms quite quickly with a single exception being the Furuta Pendulum. The biggest reason why is that the SNES compared to the NPG/TRPO has much less hyperparameters that need to be adjusted. The only ones being:

1. λ:
   The population size λ defining the amount of samples which are drawn from the search distribution for each episode. For all our solution we used the suggested calculation in [11] to set λ which is also defined as the default setup in our project.

2. Number of roll outs:
   In case of the NES this parameter not quite defines the number of roll outs per episode but rather the roll outs performed for each policy evaluation. Thus for each episode $n = \lambda * number\ of\ roll\ outs$ simulations are done while resetting the environment seed between each sample to a random value chosen at the beginning of each episode. Increasing the number of roll outs decreases the variance during roll outs which is identical to the NPG. For all platforms except the Furuta Pendulum this was set to one and turned out to be enough to find good solutions. To solve the Furuta Pendulum however, we had to use 15 roll outs as it was much more unstable than the cartpole.

3. Policy:
   Similar to the NPG the policy can we varied depending of tasks and setup, but we decided to use the same policy as the NPG with the exception that only greedy evaluations are performed as the NES already incorporates exploration in the search distribution over the policy parameters. Still the neural network can be adjusted in size and activation function, 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.

Compared to the NPG though the SNES is harder to interpret during its' training process. Figure 5 shows that e.g. the Furuta Pendulum performed much worse during the last few episodes of the training process than the benchmark. We realized that the SNES takes much longer to converge to almost no exploration. Still in most tasks it managed to converge to a near optimal solution much faster than the NPG.

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 invarient to small changes in the hyperparameters.

2.2 Physical System

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. Cazzolato, B.S., Prime, Z.: On the Dynamics of the Furuta Pendulum. Journal of Control Science and Engineering **2011**(May), 1–8 (2011). DOI 10.1155/2011/528341
4. 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`
5. 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/`

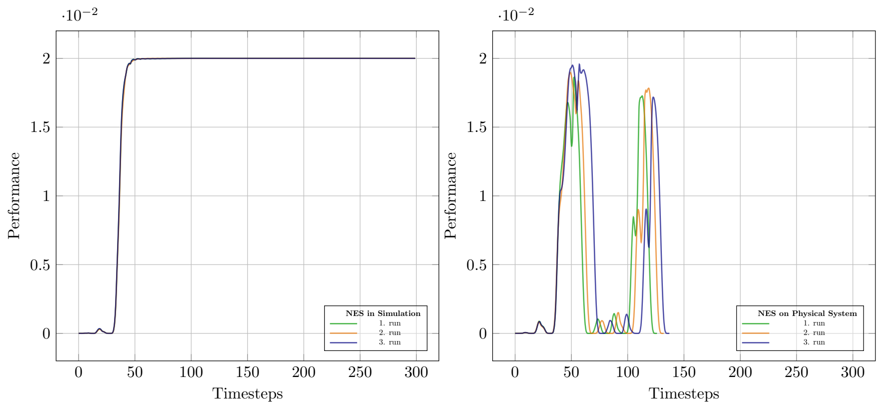

Fig. 6: This figure shows the performance of a policy trained in simulation. On the left you can see three runs done in simulation and on the right the same policy applied to the physical system

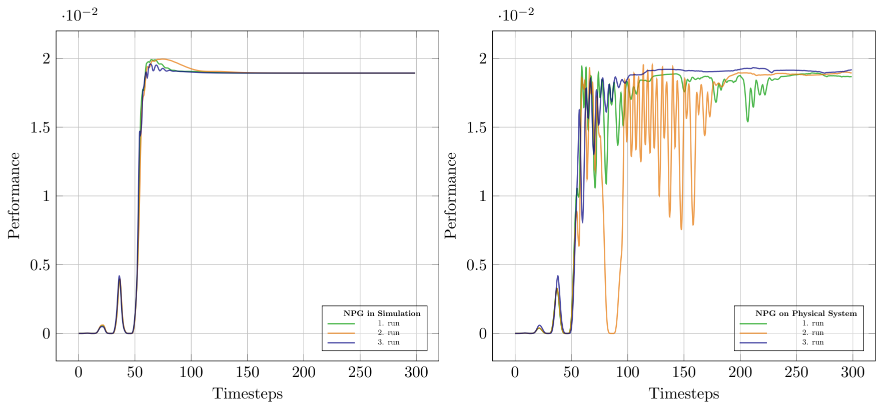

Fig. 7: This figure shows the performance of a policy trained in simulation. On the left you can see three runs done in simulation and on the right the same policy applied to the physical system

6. Gäfvert, M.: Modelling the Furuta pendulum. April. Department of Automatic Control Lund Institute of Technology (1998). URL `http://www.control.lth.se/documents/1998/7574.pdf`
7. He, J., Ai, L., Liu, X., Huang, H., Li, Y., Zhang, M., Zhao, Q., Wang, X., Chen, W., Gu, H.: Plasmonic CuS nanodisk assembly based composite nanocapsules for NIR-laser-driven synergistic chemo-photothermal cancer therapy. Journal of Materials Chemistry B **6**(7), 1035–1043 (2018). DOI 10.1039/c7tb02772a
8. 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)
9. 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`

10. 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`
11. 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`