ulm university universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Neuroinformatik

DRL - Continuous Robotic Control

Abschlussarbeit an der Universität Ulm

Vorgelegt von:
Marco Deuscher
marco.deuscher@uni-ulm.de
766668

Gutachter:
Prof. Dr. Daniel Braun

Betreuer:
Heinke Hihn M. Sc.

2022

Fassung February 26, 2022

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

Reinforcement learning has gained popularity in recent years especially due to advances in the field of artificial intelligence. In past years it was successfully demonstrated how reinforcement learning approaches can be applied to the task of continuous robotic control. In this work the Trust Region Policy Optimization algorithm is discussed as it forms the basis of Proximal Policy Optimization algorithm. In the replication study the Proximal Optimization algorithm is considered with the goal of reproducing some of the original results. I was able to reproduce some of the results achieved by the original algorithm on specific environments whereas on other environments I was unable to achieve similar results.

Contents

1 Introduction

In the past years great strides have been made in the field of machine learning. The field of machine learning is commonly divided into further subfields including supervised and unsupervised learning. In supervised learning a model is presented with data samples and their corresponding label. This includes, in the most general case, tasks such as classification and regression. The subfield of unsupervised learning contains tasks such as clustering. However, there is an additional field with rising popularity in recent years due to advances in the field of artificial intelligence which is reinforcement learning. Different to the previously mentioned tasks there is no dataset in reinforcement learning, instead an agent interacts with an environment and bases its decision on the interaction with the environment.
In this replication study the main focus will be on reinforcement learning, more precisely on continuous robotic control using policy gradient methods. The goal is the replication of the original results achieved by the Proximal Policy Optimization algorithm for continuous robotic control tasks [8].

2 Methods

This chapter introduces the methods and theoretical background that is required to understand the Proximal Policy Optimization algorithm. Section 2.1 introduces the task of continuous robotic control, Section 2.2 provides an introduction to the basics of reinforcement learning, one of the fundamental papers for the PPO algorithm is introduced in Section 2.3. Section 2.4 goes in depth on the PPO algorithm.

2.1 Continuous Robotic Control

Continuous robotic control in the context of reinforcement learning refers to the task of maximizing the reward of an agent on a specific environment by providing a control signal to the robot that acts as the agent. In many reinforcement learning problems the state space and especially the action space are discrete. For continuous robotic control commonly both the state space, and the action space are continuous as they model the physical environment such as torques applied to actuators. Due to the continuity of the action space, approaches such as Deep Q-Networks cannot be applied to continuous tasks without modification as they suffer from the curse of dimensionality [**journals/corr/LillicrapHPHETS15**]. Thus, the need for more general algorithms that do not suffer from a high dimensional action space are of interest. The problem formulation for the task of continuous robotic control is generally in the realm of control theory. Especially optimal control theory is closely related to reinforcement learning [4]. Further, the considered tasks often contain very high dimensional state and action spaces. One such example are the 3D robotic environments in the openAI gyms, e.g. the Humanoid-v2 [1]. For such high-dimensional problems especially with many actuators the computation of a solution using optimal control theory may become time-consuming as well as missing guarantees of the validity of the solution [4]. Thus, reinforcement learning

based solutions are of interest as their computational complexity during inference mainly consists in the inference time of the policy without having to solve an optimization problem online.

2.2 Reinforcement Learning Basics

This section introduces the basic reinforcement learning framework. In reinforcement learning the agent interacts with the environment by issuing actions and observing the resulting state transition and reward [10]. This is visualized in Figure 2.1. Further, the environment is commonly described by a Markov Decision Process

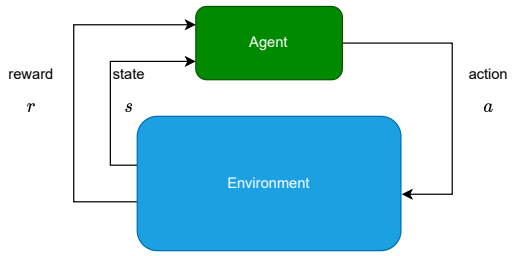

Figure 2.1: Todo

which is given by the tuple $\{S, A, T, R, p(s_0), \gamma\}$ where S is the set of states, A the set of actions, $T : S \times A \to p(S)$ a transition function, R a reward function, $p(s_0)$ is the distribution for the initial state and γ is the discount factor [5]. Based on this, we define the policy as a function $\pi : S \times A \to p(S)$ and introduce the expected discounted reward [9]

$$\eta(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right] \tag{2.1}$$

where $s_0 \sim p(s_0)$, $a_t \sim \pi(a_t|s_t)$ and $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. Further, the state-action value function, value function and advantage function respectively are defined as

$$Q_\pi(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k})\right] \tag{2.2}$$

$$V_\pi(s_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k})\right] \tag{2.3}$$

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{2.4}$$

where a_t and s_{t+1} are sampled from the respective distributions [9]. There are many possible ways to learn in the above shown framework. Trust Region Policy Optimization and Proximal Policy Optimization both based on Policy Gradient methods which is why will only discuss those in the following. Now consider a policy π_θ which is parameterized by a set of parameters θ which must be differentiable w.r.t. the parameters [11]. The basic premise of policy gradient methods is to compute an estimate of the policy gradient and use it to perform gradient ascent optimization. One such estimator is given by

$$\hat{g} = \mathbb{E}\left[\nabla_\theta \log(\pi_\theta(a_t|s_t))\hat{A}_t\right] \tag{2.5}$$

where ∇_θ denotes the gradient w.r.t. the parameters and $\hat{A}_t$ the estimate of the advantage function [8].

2.3 Trust Region Policy Optimization

Trust Region Policy Optimization plays a vital role in understanding the Proximal Policy Optimization algorithm. Based on the framework introduced in Section 2.2 a monotonic improvement guarantee can be formulated. One of the main contributions is the proof that such a monotonic improvement guarantee can be obtained not just for mixture policies but also general stochastic policies which includes neural networks [9]. This leads to the following theorem [9]: Let $\alpha = \max_s \left\{\frac{1}{2}\sum_i |\pi(\cdot|s)_i - \tilde{\pi}(\cdot|s)_i|\right\}$ be the total variation divergence between the old and the new policy. Then the

following bound holds

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - \frac{4\varepsilon\gamma}{(1-\gamma)^2}\alpha^2 \tag{2.6}$$

where π is the new policy, $\tilde{\pi}$ the old policy, γ the discount factor, $\varepsilon = \max_{s,a}\{|A_{\pi}(s,a)|\}$ and $L(\cdot)$ is a local approximation of the discounted reward η given by

$$L_{\pi}(\tilde{\pi}) = \eta(\pi) + \sum_{s} p_{\pi}(s) \sum_{a} \tilde{\pi}(a|s) A_{\pi}(s,a) \tag{2.7}$$

The same can be formulated using the Kullback-Leibler divergence instead of the total variation divergence which results in [9]

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - CD_{KL}^{max}(\pi, \tilde{\pi}) \tag{2.8}$$

where $C = \frac{4\varepsilon\gamma}{(1-\gamma)^2}$. This allows for a chain of monotonically improving policies π. To obtain a practical algorithm two further problems must be solved. The policy cannot be evaluated at every state s which is why the optimization problem is posed as follows

$$\max_{\tilde{\theta}} L_{\pi_\theta}(\pi_{\tilde{\theta}}) \tag{2.9}$$

subject to $\mathbb{E}\left[D_{KL}(\pi_\theta || \pi_{\tilde{\theta}})\right] \leq \delta$. Here δ controls the maximum update step size of the policy. Lastly, the formulation of the optimization problem must be converted to sampled-based estimation of the objective function and the constraint of the optimization problem. A practical formulation of the Trust Region Policy Optimization algorithm is shown in Algorithm 1 Note that the conjugate gradients procedure

Algorithm 1 Trust Region Policy Optimization - Practical Algorithm

Require: Hyperparameter $N, T, \ldots$
  while termination criterion not met **do**
    for actor=$1, \ldots, N$ **do**
      Collect rollout buffer for T steps
      Compute Monte-Carlo estimates of Q-values
    end for
    Compute sample-based expectations of the objective and constraint
    Solve constrained optimization (conjugate gradients and line search)
  end while

requires second derivatives which are computationally expensive to obtain.

2.4 Proximal Policy Optimization

The fundamental goal of the Proximal Policy Algorithm is to preserve the advantages of TRPO while using simple first-order optimization methods which results in a simple and more efficient implementation [8]. PPO builds on the previously introduced policy gradient methods and TRPO by introducing the concept of a surrogate function which is optimized instead. The surrogate objective I used in the implementation and that achieved the best results is the clipped surrogate objective which is given by

$$L_{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \cdot \hat{A}_t \right] = \hat{\mathbb{E}} \left[r_t(\theta) \hat{A}_t \right] \tag{2.10}$$

where $\hat{A}_t$ is the advantage function estimate. This is however missing the in TRPO introduced limit for the policy update which leads to excessively large policy updates [8]. Thus, the clipped function is given by

$$L_{CLIP}(\theta) = \hat{\mathbb{E}} \left[\min \left\{ r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon) \hat{A}_t \right\} \right] \tag{2.11}$$

where ε is a hyperparameter controlling the largest possible policy update. A common trick during implementation is to normalize the advantage function estimate to zero-mean and unit-variance to increase stability during training. A second proposed surrogate function is the adaptive KL penalty coefficient surrogate function which is given by

$$L_{KLPEN}(\theta) = \hat{\mathbb{E}} \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \cdot \hat{A}_t - \beta D_{KL}(\pi_{\theta_{old}} || \pi_\theta) \right] \tag{2.12}$$

where the penalty coefficient β is either halfed or doubled based on a target KL-Divergence hyperparameter [8]. The Proximal Policy Algorithm in the Actor-Critic implementation is shown in Algorithm 2 [8] The training loop is repeated until the termination criterion is met which is given by a total number of training steps. In a first step a rollout buffer is collected which contains a list of the visited states, the actions and observed rewards. Based on the collected rollout buffer the advantage function estimate is computed, for this the value function is evaluated, and the advantage function is computed following Equation 2.4. In particular, the rewards-to-go are required for the estimation. The rewards-to-go or discounted rewards are computed for each episode in the rollout buffer by iterating the rewards in reverse

Algorithm 2 Proximal Policy Optimization - Actor-Critic implementation

Require: Hyperparameter $\varepsilon, N, T, K, \ldots$
  while termination criterion not met **do**
    for actor=$1, \ldots, N$ **do**
      Collect rollout buffer using current policy $\pi_{\theta_{old}}$ for T steps
      Compute advantage function estimate $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    for epoch=$1, \ldots, K$ **do**
      Optimize surrogate function L w.r.t. θ
      Update policy $\theta_{old} \leftarrow \theta$
    end for
  end while

order and adding the current reward to the discounted sum. The value function and policy are both given by neural networks which are optimized for K epochs. The policy networks uses the surrogate objective function for gradient ascent. Further, a Huber loss is employed for the optimization of the value function network. The PPO algorithm is very flexible to different implementations regarding the modeling of the policy and value function. For simplicity, I have decided to model both as separate neural networks but parameter sharing between the networks is possible. This influences the computation of the surrogate / loss function. The architecture of the policy network is shown in Figure 2.2. A simple Multi-Layer Perceptron network with three layers is used, each layer using a $\mathrm{Tanh}(\cdot)$ activation [3]. The output of the policy network is a mean-value which is used to construct a multi-variate gaussian distribution from which an action is sampled. Note that the covariance is initialized as a diagonal matrix as it is assumed that there exists no dependencies between the components. In the following the variance used to initialize the covariance matrix is referred to as σ. The architecture is shared by the value function network sans the action sampling and instead outputs a real number.

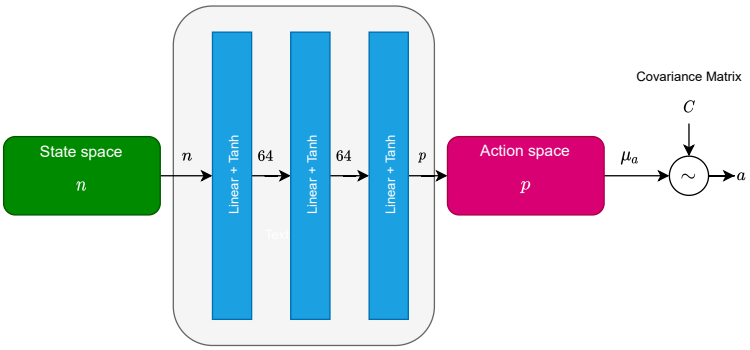

Figure 2.2: Architecture of the policy MLP.

3 Experiments and Evaluation

In this chapter the experimental procedure is described in detail, and the evaluation including the quantitative results are presented.

3.1 Experimental Setup

I implemented the PPO algorithm in Python using PyTorch for the implementation of the neural networks and the optimization [6]. For the environments I used the openAI gym in combination with PyBulletEnv for an open-source implementation of the simulation for the robotic environments [1, 2]. Further, I employed Stablebaselines3 for the normalization of the environment [7]. In particular, I normalized both the state space and rewards. The two considered environments are the Ant and Swing-up pendulum. The training was conducted on an AMD Ryzen 2700X CPU for a total of X training steps. Requiring a total time of Y minutes. The policy and value function are modeled by separate neural networks respectively. The architecture is presented in Section 2.4. Table 3.1 shows the hyperparameters. The naming is in

Hyperparameter	Value
ε-clip	0.2
γ	0.99
σ	0.5
N	2048
T	200
K	10
numeric stability	$1 \cdot 10^{-10}$
learning rate	$2 \cdot 10^{-4}$

Table 3.1: Hyperparameters used during training.

accordance with Algorithm 2. Additionally, σ refers to the variance used to initialize the covariance matrix used in the multivariate gaussian to sample an action from the mean-value provided by the policy network. To ensure numeric stability when normalizing the advantage function estimates the above shown hyperparameter is added during normalization. The learning rate is used for both the value function and policy network.

3.2 Evaluation

4 Discussion

The primary goal of this work was to replicate the results achieved by the original PPO authors. Thus, in the following I firstly discuss to what extent I was successful in reproducing the results. Afterwards, I discuss the topic of reproducibility in science in general, and my experience in replicating the results

4.1 Results

4.2 Reproducibility in Science

In science almost all work is related to previous publications either by improving established work or comparing novel methods to the current state of the art. Thus, in order to compare your own results to the existing publications' reproducibility is extremely important. In the context of machine learning and reinforcement learning this means that an author can validate the results of others by running the experiments on their own machine. However, not all authors publish their implementation which requires re-implementing their work based on the provided paper. Many papers are relatively vague regarding implementation details and are missing vital information. Thus, replicating results solely based on the paper is often difficult. There are platforms helping with this problem. One such platform is *paperswithcode*[1] grouping publications based on the task or dataset and offering links to available implementations. This demonstrates some difficulties in regard to reproducibility in the field of machine learning. Reinforcement learning introduces additional challenges as the entire framework presented in Section 2.2 is intrinsically based on statistics which further

[1] `https://paperswithcode.com/`

makes replicating exact results difficult.

In the following I'd like to discuss my personal experience reproducing the results of the PPO algorithm. Previous to this project I had no practical experience in the field of reinforcement learning except a brief introduction in a single lecture. So the beginning included a lot of learning about the basics of reinforcement learning, common conventions and an overview of the field. Starting with the environments I found it relatively easy to setup (..) compatible with the existing work . Something I struggled with especially at the beginning of the implementation was the transfer from the formulas given in the paper to an actual implementation, e.g. replacing expectations by their empirical, sampled-based estimator. The PPO paper is vague regarding the modeling of the policy and value function. The original paper suggests the actor-critic method, it is however unclear if it is implemented as separated networks or with parameter sharing. Available implementations also vary widely in this point. Once I had a working implementation, I had some problems with hyperparameters. To achieve optimal performance I tried to use the same hyperparameters as available implementations but depending on the implementation they change their name and also their definition which makes it difficult to copy the hyperparameters when one is not intimately familiar with the common conventions.
Even though I had no previous experience with reinforcement learning, I have some experience in Deep Learning regarding reproducibility from my bachelor thesis and various projects at Team Spatzenhirn. For me, it was significantly more difficult reproducing the results of the PPO algorithm compared to previous projects I worked on such as monocular depth estimation or depth estimation. Something I noticed is that especially when transferring to a new dataset / environment the reinforcement learning models are incredibly sensitive w.r.t. their hyperparameters. I imagine that this makes it difficult to transfer a model from one of the popular dataset to an actual application.

5 Conclusion

Bibliography

[1] Greg Brockman **andothers**. *OpenAI Gym*. cite arxiv:1606.01540. 2016. URL: http://arxiv.org/abs/1606.01540.

[2] Benjamin Ellenberger. *PyBullet Gymperium*. https://github.com/benelot/pybullet-gym. 2018–2019.

[3] Ian Goodfellow, Yoshua Bengio **and** Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[4] Simon Gottschalk **and** Michael Burger. "Differences and similarities between reinforcement learning and the classical optimal control framework". **in***PAMM*: 19 (**november** 2019). DOI: 10.1002/pamm.201900390.

[5] Thomas Moerland, Joost Broekens **and** Catholijn Jonker. *A Framework for Reinforcement Learning and Planning*. **june** 2020.

[6] Adam Paszke **andothers**. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". **in***Advances in Neural Information Processing Systems 32*: **byeditor**H. Wallach **andothers**. Curran Associates, Inc., 2019, **pages** 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[7] Antonin Raffin **andothers**. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". **in***Journal of Machine Learning Research*: 22.268 (2021), **pages** 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[8] John Schulman **andothers**. "Proximal Policy Optimization Algorithms." **in***CoRR*: abs/1707.06347 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17.

[9] John Schulman **andothers**. "Trust Region Policy Optimization". **in***ArXiv*: abs/1502.05477 (2015).

[10] Richard S. Sutton **and** Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[11] Richard S. Sutton **andothers**. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. **in***Proceedings of the 12th International Conference on Neural Information Processing Systems*: NIPS’99. Denver, CO: MIT Press, 1999, 1057–1063.

Name: Marco Deuscher Matrikelnummer: 766668

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Marco Deuscher