Poppy Arm Control using Deep Reinforcement Learning

REPORT for
PROBLEM DESCRIPTION and MODELING

by

Mohamed Fares Abid
Bo Huang
Maxime Kirgo
Rémi Laumont
Dominik Meinzer
Jiawen Xu

Applied Reinforcement Learning
Lehrstuhl für
DATENVERARBEITUNG
Technische Universität München

Prof. Dr.-Ing. Klaus Diepold

Contents

Chapter 1

Introduction

In recent years, model-free Deep Reinforcement Learning (DRL) using Q-learning has accomplished impressive results in achieving superhuman capabilities in Atari 2600 games [6, 7, 5]. Besides the game domain, many Deep Q-learning methods have been developed for control in Robotics [1, 2], where the robots learn in little time, without demonstration or human interaction, how to open a door. Q-learning [9, 10] is a *model-free* Temporal Difference (TD) method that can be used to learn an optimal policy to any given task. The method can learn in an *off-policy* manner, i.e. an optimal policy is learned regardless of the current agent's behavior. The model-free property allows to learn the underlying model by sampling from the environment and the possibility to train off-policy makes the method data efficient [2]. However, a basic, iterative learning approach, where a policy is learned in a tabular fashion for every state-action pair, is often impractical in real-world applications [3]. Using a non-linear function approximator, like a deep Neural Network (NN) and feeding enough data into the network produces good feature representations which are learned directly from the data. As a consequence, deep NNs have become a popular function approximator and were successfully applied in complex situations. Inspired by the aforementioned works and the will to use and learn about a very basic algorithm in DRL, we aim to control the arm motion of Poppy, a humanoid robot, in this work using the Deep Q-Network (DQN) algorithm. Applying DRL in this project will give us necessary experience for future projects, even outside of this lecture.
In the following report, we will first concretize our the problem of manipulation of the Poppy robot. Second, we explain our modeling of the problem and how we can simplify the problem to make learning easier and faster.

Chapter 2

Problem Description

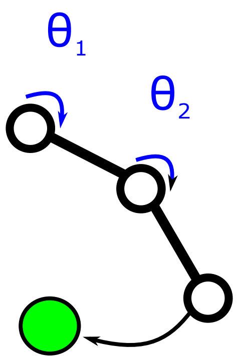

Figure 2.1: Arm Control Task

In this work, we use a humanoid robot, called Poppy. Typical tasks for humanoid robots with many degrees of freedom is to learn motion control. In our case, we will focus on arm motion. The problem of controlling a robotic manipulator can often be solved by using Inverse Kinematics (IKs), but have drawbacks like singularities, joint limitations and maybe unknown kinematic parameters, such as arm lengths. However, manipulation provides us with a simple example to use the DQN method. In theory, Q-learning allows us to learn any task based on the model-free property. In addition, using NNs allows us to learn necessary features for accomplishing the task instead of manually generating good features, which can be very useful for more difficult and complicated problems. Another advantage of the NN is the generalization inside the action-state space, because reaching and trying all different configurations infinitely often to learn good Q-values is impossible, even in a simple manipulation example. Nevertheless, the DQN algorithm should be able to learn a simple reaching task, i.e. how to control the arm to a given target position, while the robot will be able to cope with singularities, joint limitations and unknown arm lengths. Based on the idea of Q-learning, we can even extend the task to different scenarios, such as using both arms, drawing triangles or other shapes, depending on the time of the project. Although the problem of arm motion seams simple, we would like to emphasize that the DRL technique, the DQN algorithm is not, as non-linear function approximators, such as deep NN often tend to diverge in DRL. Achieving a smart agent in the end will be the most challenging part of the project. Mnih et al. [7] proposed the DQN algorithm, including the ideas of a replay memory and a target network, which will be the starting point for our algorithm. In addition, we introduce our own exploration policy, because ϵ-greedy is not suitable for exploration on real hardware and will not result in a short learning time. The exploration is based on an IKs approach to guide the robot arm into the direction of the target and explore randomly around this trajectory. The details will be explained in Sec. 3. Finally, we will have to tune many hyperparameters in order to make the robot move as we expect.

Chapter 3

Modeling

Due to the short project time, we want to start with a very simplified version of the problem. Consequently, we focus on only arm motion of the robot. Additionally, we decrease the degrees of freedom of the Poppy arm from 4 to 2, see Fig. 3.1. This means that we will restrict the control to motors *l_shoulder_x* and *l_elbow_y*, called in the following θ_1 and θ_2 respectively. Motor l_shoulder_y is set to $-90°$ and motor l_arm_z is hold at $0°$ in order to allow movements of the hand only in the x-axis (sideways) and y-axis (forward/backwards). This restriction should help us to learn a reasonable behavior of the robot arm in shorter training time.
The original DQN algorithm uses images as input to the Q-Network and discrete joystick actions. In order to simplify the state space, we will not use images as input, but a description of the robot and the environment in only 6 states. Nevertheless, this description fulfills the Markovian property. These 6 states consist of the two joint angles of the robot arm $\vec{\theta} = [\theta_1, \theta_2]$, the end-effector position in the x-y plane $\vec{x}_{arm} = [x_{arm}, y_{arm}]$ and the goal position in the x-y plane $\vec{x}_{goal} = [x_{goal}, y_{goal}]$. This means that our state s is:

$$s = [x_{arm}, y_{arm}, \theta_1, \theta_2, x_{goal}, y_{goal}] \tag{3.1}$$

where the arm joint angles θ_i can be measured by the internal sensors of the robot, the position of the end-effector can be computed by simple forward kinematics and the goal position is set by ourselves manually at the beginning of the project. If everything is working out as planned, we may include vision to determine the goal.
The action space in our project is planned to be 4 discrete actions. The agent controls two motors and can move each motor with $\pm 1°$. A larger action space will be considered during the experiments, but may be harder to train. The reward signal r is designed as the negative distance between the end-effector position and the goal position, i.e.

$$r = -\left\|(\vec{x}_{arm} - \vec{x}_{goal})\right\|^2 \tag{3.2}$$

This reward design is punishing the agent more when moving away from the target and punishing the agent less when approaching the target. Additionally, punishing

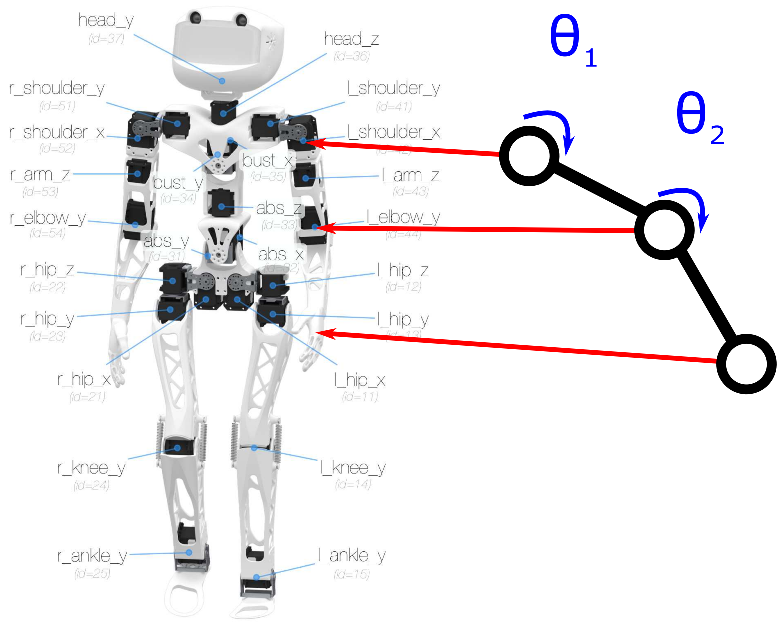

Figure 3.1: Arm Modeling of Poppy

the agent in every step is forcing the agent to get to the goal as fast as possible.

The training follows the learning rules as proposed by Mnih et al. in [6, 7], while our Q-Network will not have convolutional layers and is inspired by the works of Lillicrap et al. in [4]. We plan to use an input layer and two hidden layers with each 100 neurons and ReLU activation function. The dimension of the output layer of the Q-Network is 4, i.e. one for each action, and has a linear activation function. In addition, we plan to use the *soft* update rule for the target network as in [4]. Regarding the DQN algorithm we aim for a modularized and asynchronous implementation, as visualized in Fig. 3.2 and inspired by [8, 5]. In this way, we can interact with different environments in parallel, using seperate threads, and collect more distinct experience samples e_t at timestep t in the form:

$$e_t = [s_t, a_t, r_t, s_{t+1}, terminal] \tag{3.3}$$

where s is the state of the agent, a the action selected by the agent, r the reward signal according to the distance when arrived at state s_{t+1}. The boolean variable *terminal* denoted if the episode finished after this experience sample and is needed for the Q-learning update rules in the terminal state.

Finally, we noticed that ϵ-greedy is not a good exploration policy in robotics and we will try to guide the robot arm into the goal direction by using IK. The main idea of this guidance of the actions is to show the robot almost correct experiences

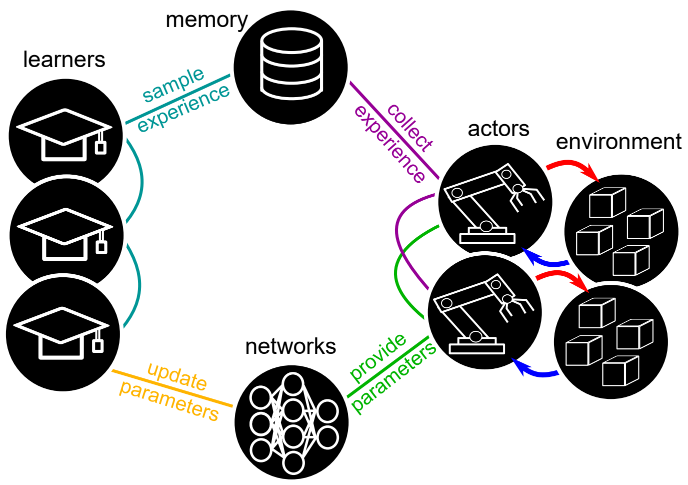

Figure 3.2: Asynchronous architecture for Deep Reinforcement Learning

and to let the agent improve itself around this guidance by randomly selecting other actions. These random actions might improve the policy and the agent's behavior could show better performance than the guidance originally taught to the agent. In detail, the exploration will not be fully ϵ-greedy, but 50% ϵ-greedy and 50% IKs guided. The IKs guidance is done in the following way. The joint angle velocities $\delta\vec{\theta}$ are computed by:

$$\delta\vec{\theta} = J^{-1} \cdot (\vec{x}_{goal} - \vec{x}_{arm}) \tag{3.4}$$

where J is the Jacobian matrix of the robot arm at joint angles $\vec{\theta}$.
Then, the motor to control is determined by selecting the motor with the larger joint angle velocity:

$$\arg\max_{i \in \{1,2\}}(\delta\theta_1, \delta\theta_2) \tag{3.5}$$

Finally, the action is determined by the motor i to move and the sign of the respective $\delta\theta_i$ in order to decide the direction $\pm 1°$.

Draft of a Project Plan

1. Set up the simulation (done)
2. Use the inverse kinematics as a base controller in simulation (done)
3. Use the inverse kinematics as a base controller on the robot
4. Implement the DQN algorithm (done)
5. Include the simulation into DQN framework
6. Train the Q-Network and tune hyperparameters
7. Try the Q-Network on the real robot
8. Optional features: vision for goal detection, draw triangle or other shapes, use both arms, etc.

Bibliography

[1] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 2016.

[2] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *arXiv preprint arXiv:1610.00633*, 2016.

[3] Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

[4] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[8] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[9] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[10] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge England, 1989.

List of Acronyms

DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
IK	Inverse Kinematic
NN	Neural Network
RL	Reinforcement Learning
TD	Temporal Difference