DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Reinforcement Learning for Path Planing of Robotic Arms

Anton Mai

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Reinforcement Learning for Path Planing of Robotic Arms

Bestärkendes Lernen zur Pfadplanung von robotischen Armen

Author:	Anton Mai
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Dr Zhenshan Bing
Submission Date:	February 17, 2020

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, February 17, 2020 Anton Mai

Acknowledgments

Abstract

Contents

1 Introduction

Lee Sedol, one of the best Go players in the world, was beaten by the Go engine AlphaGo in a match. The engine was clearly stronger. AlphaGo only knew the rules at the beginning and got stronger only by playing with itself. Artificial Intelligence is quite popular nowadays because of its many use cases: Self-Driving cars, playing atari games, robotics and more. But how do these engines learn how to get so good at their areas ? The answer is reinforcement learning, an area of machine learning.

The idea of reinforcement learning is to have a state and actions that an agent can choose from. Each action results in different rewards and states. Rewards are used by agent to measure how good an action was. This process is repeated which results in the agent learning which actions in each state are better. Imagine you are a soccer player. You are standing in front of the goal (which is the state you are in). You can either shoot or pass the ball (which are your available actions). You choose to shoot, but the ball is blocked by the goalkeeper (you got a low reward). So the next time you are in front of the goal again, you will more probably try to pass the ball. This time your teammate scored a goal (you got a high reward). From this experience you learn that it is probably better to pass the ball if you are standing in front of the goal. The concept of reinforcement learning can be used in a variety of environments, for example robotic arms.

Already in the 14th century, Leonardo da Vinci made blueprints of robotic arms.

A robotic arm resembles a human arm. It consists of segments which are connected by joints. The number of joints correspond to what is called Degrees of Freedom. A robotic arm with 5 joints would have 5 Degrees of Freedom because it can pivot in 5 ways. Each joint is connected to a step motor. Step motors make the robot move very precisely. The equivalent to a human hand is the end effector. The end effector can vary depending on the tasks.

Robotic arms have many advantages. They are very accurate and consistent which is why they are mostly used for repetitive tasks or tasks that require high accuracy which

are hard for humans. This is the main reason why they are used in laboratories and hospitals for surgeries. They can also be used automatically without any human which is why they are used for manufacturing and assembly lines.

Humans still have to teach the robotic arms how to move when setting them up. For path planning of the robotic arm, a sequence of actions has to be found that solves the task. This sequence is saved and repetitively executed by the robotic arm. Finding the path still requires human labor. Either by testing or by using linear algebra a path can be found. A robotic arm needs 6 Degrees of Freedom to be able to move its end effector in every direction and orientation. This also means that robotic arms with more degrees of freedom do not have a unique path to solve the tasks. There are different paths which can vary in length and energy consumption. To improve the quality of the path and to do path planning without a human, using reinforcement learning for robotic arms is a logical approach.

...

There is an issue that prevents robotic arms to learn with reinforcement learning. It is hard to construct a suitable reward function for tasks where robotic arms are used. For example ... So either a suitable reward function has to be constructed by hand, or the simplest reward function, a binary sparse reward function has to be used. Both approaches have some issues. Constructing a reward function can be quite complicated. Also, for each task an individual reward function has to be made. So someone has to do this work which defeats the purpose of using reinforcement learning for robotic arms over path planning by hand. Depending on the case it might be easier to just plan the path without reinforcement learning. Using only a sparse reward for robotic arms is as follows. a reward is given, when the goal is reached, no reward is given when the goal is not reached. Robotic arms have usually many Degrees of Freedom, so there are many actions that can be taken by the robotic arm. It is quite unlikely for robotic arms to fulfill the task by doing random movements. Tasks like moving an object are near impossible to solve with random actions. So it is very unlikely for the robotic arm to earn a reward and learn. It takes a very long time to train a robotic arm with sparse rewards. But recently hindsight experience replay has been introduced. Hindsight experience replay allows a high learning rate even with sparse rewards.

Hindsight experience replay works as follows.

This thesis is structured as follows: Chapter 2 describes the theoretical background on robotic arms, reinforcement learning and algorithms like deep deterministic policy gradients and hindsight experience replay. Chapter 3 explains the methodology used for this thesis. Chapter 4 gives an overview of the simulation environment. In chapter 5, the experiments are presented and the results are discussed. In the last chapter, the results are summarized and suggestions for further word is provided.

2 Theoretical Background

This chapter explains the concepts needed to understand this thesis. The theory behind reinforcement learning, deep deterministic policy gradients, hindsight experience replay and hindsight goal generation are explained in this chapter.

2.1 Reinforcement Learning

Reinforcement learning is one of the main learning models of Machine learning next to Supervised Learning and Unsupervised Learning. In Supervised learning some input data is given to the learning agent. The agent is expected to come up with some output which is then compared with the expected output. If the output given by the agent and the expected output matches, then the agent was correct. An use case for Supervised Learning is sorting mails into regular mail and spam mail. The agent is given a mail and it should decide whether the mail is regular or spam based on the content. Supervised Learning is used for classification and regression problems. It is mainly useful when the expected output is already known, so the learning agent can learn to do recognize these. In Unsupervised Learning there is no expected up. The learning agent is fed with data so it can figure out interesting features and similarities between different data. Unsupervised Learning is often used to cluster data, often pictures, based on similarities. In Reinforcement Learning the agent is learning through rewards that are given through interaction with an environment. The goal of a task is clear, but the path of actions to reach the goal is not trivial. Reinforcement Learning is used to find the best action in each situation. It is often used for games because they are already set up to have a clear task and goal, but the optimal way to reach it is not clear. Supervised Learning is limited in performance because the agent can only learn to become as good the expected output that we set. So in games like chess, with Supervised Learning the agent can only become as good as the best players it learns from. But Reinforcement Learning is not limited by that. The agent can improve on its own only by exploring his options. In games like Go and Chess, engines that use reinforcement learning have already far surpassed the best human players.

This section will explain the theory behind reinforcement learning. Reinforcement learning is usually modeled as a Markov Decision Process. The Markov Decision Process for Reinforcement Learning consists of following elements

- A set of states S
- A set of actions A
- The transition probability Pa(s,s') from state s to s' under action a
- The immediate reward Ra(s,s') of that transition
- rules that describe the agents observation

The agent and environment are in a state s. The agent chooses an action a from its set of possible actions A to interact with the environment. The environment reacts by transitioning to another state and returning a reward R and an observation to the agent. Depending on the model the transitions might be stochastic or deterministic. The aim of the agent is to earn the maximal total reward possible. To reach this aim, the agent interacts with the environment to gain knowledge about the environment through the gained rewards and observations. Through this process, the agent learns in which state which actions are better to gain more reward. This is illustrated by Figure 2.1.

In each state there is an action that the agent considers best due to its current knowledge about the expected rewards of each action. This set of actions is known as the policy The goal of getting maximal reward can be interpreted as finding the best actions in each state that give the most reward, which is finding the optimal policy. The policy can also be either deterministic or stochastic, depending on the transition probability of the environment.

A value function is used to measure how good a state or action is. Two types of value functions are used for the states and the actions. the state value function is denoted as V(s). The value of a state is the expected reward when acting according to the policy V(s) is defined as follows:

The actions value function is denoted as Q(s,a) and is defined as follows:

When determining the value of a state or action, a discount factor is used to discount future rewards towards immediate rewards. The idea is that a state s is not only as good as the reward you get when transitioning to that state. Future rewards from states

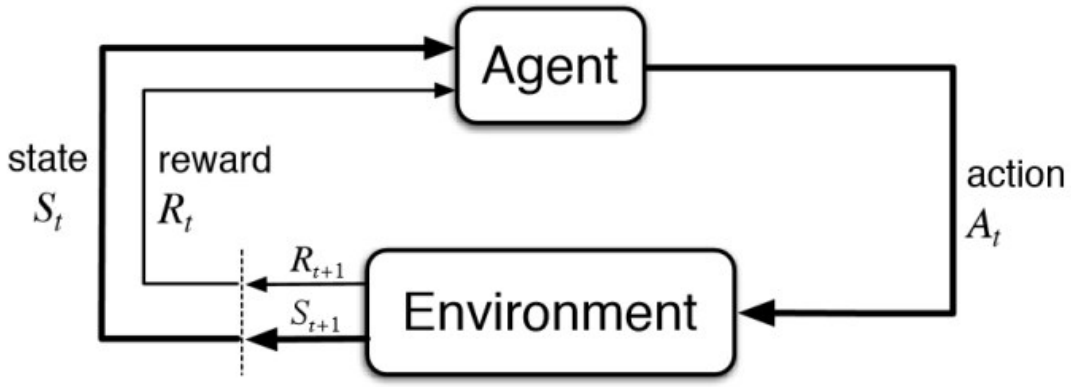

Figure 2.1: Reinforcement Learning. An agent chooses an action to interact with the environment and gets a reward and an observation of his new state back.

that are reachable from state s should also be considered. The value of a state consists of the reward that you get by transitioning to that state and the potential rewards that can be gained by transitioning from that state. Because future rewards are not as certain as immediate rewards, the discount factor is used. The farther a reward is in the future, the more it is discounted. The Bellman equations are a set of equations that convert the value functions into the immediate and future reward:

The goal is to find the actions that return the maximal reward. This is displayed by the Bellman optimality equations:

To calculate the optimal values of each state and action, Dynamic Programming could be used, if we know the model fully. But even if we knew the model fully, usually the main issue is the huge state and action space, which makes it impossible to use Dynamic Programming. For reinforcement learning, neural networks can be used to approximate the value functions.

2.2 Artificial Neural Networks

Artificial Neural Networks are inspired by the human brain. The Neural Network consists of layers of neurons. Each neuron is connected to the next layer of neurons.

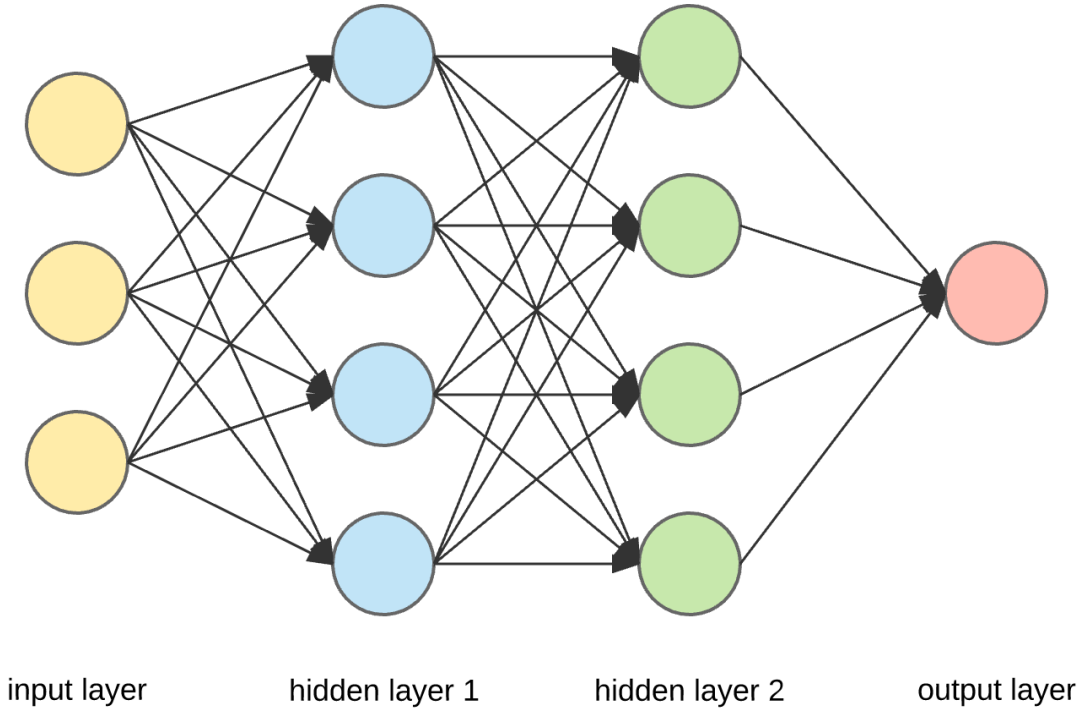

Figure 2.2: A neural network.

There is one input layer and one output layer at the beginning and end of the layer of neurons. The layers between the input and output layers are called hidden layer. The hidden layer can consist of only one or more layers. The idea is to train the neural network to take inputs and produce outputs. To approximate the value functions the input would be states and actions, the output should be the correct and optimal value of these states and actions. An example of an artificial neural network is shown in Figure 2.2.

The learning process of the neuronal network is as follows. Each neuron obtains inputs xi by the the output of the neurons in the layer before it. each input value is weighed and then added. A bias b is also added to support the learning process. After using an activation function on the sum, the value is output to the next layer of neurons. The activation function is a simple function that either reduces the output of the neuron to 0 if the value is below a certain threshold, otherwise the value is output unfiltered.

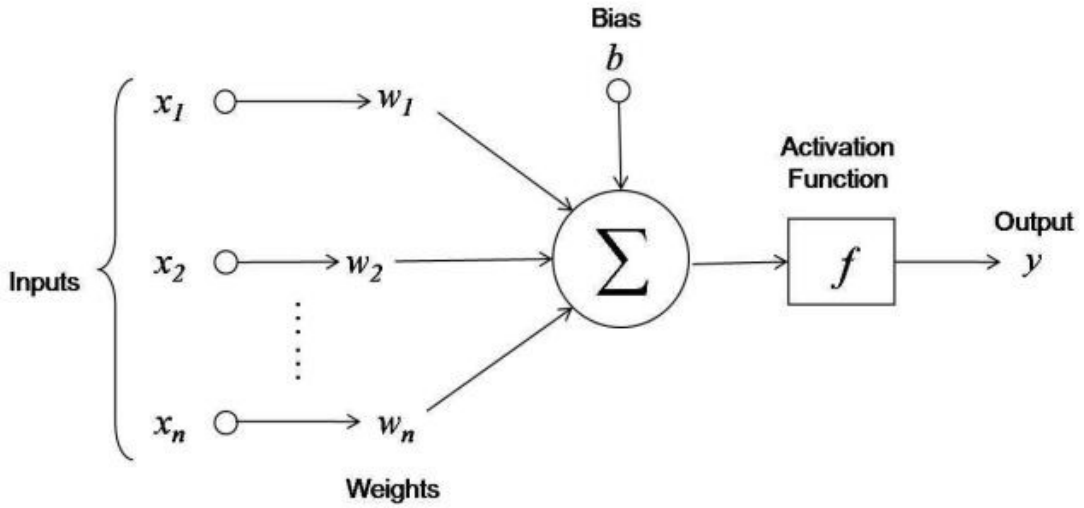

Figure 2.3: A neuron.

The training process of the neuron can be seen in Figure 2.3.

When the neural network outputs a value, a process called Backpropagation is used to further improve the neural network.

2.3 Deep Deterministic Policy Gradients

The algorithm Deep Deterministic Policy Gradient learns concurrently a Q-function (the action-value function) and a policy. When learning the optimal action value function Q*(s,a), the optimal action in each state can be solved by using following equation:

To solve the equations for a discrete action space, the Q-values of each action could be calculated and compared to find the biggest value. But in a continuous action space it is not possible to calculate the Q-value for each action. DDPG uses the fact that the action space is continuous and so Q*(s,a) is expected to be differentiable in respect to the action argument. This way it is possible to approximate the Q-values and policy.

To learn the Q-values, the Bellman equation for the action value is used. To approxi-

mate the Q-values, a mean squared Bellman error function is used:

Minimizing the MSBE loss function is equal to approximating the current Q-values to the optimal Q-values.

For DDPG an experience replay buffer is also used. The replay buffer is a set of experiences. This can be used to replay old experiences. When only using new experiences, the neural network might be overfitted to those experiences. Experience replay is useful to prevent that. But a too large buffer can cause the learning process to slow down. The right balance has to be found.

DDPG also uses target networks. The following term is called target: When minimizing the MSBE loss function, there is the problem that the target is also dependent on the parameters that are trained. When changing the parameters, the target would also change which is problematic. That is why the target network, a cpoy of the neural network is used. The update of the target network is delayed to avoid this conflict.

To find the optimal policy, simply gradient ascent can be used to find the maximal Q-values.

2.4 Hindsight Experience Replay

Sparse rewards are a big issue in Reinforcement Learning, especially in tasks for robotic arms often the rewards are sparse. Having sparse rewards means that most of the samples used for training will not successful and therefore will not bring any useful reward. For example, the task to move an object to a certain point would have a sparse reward for a robotic arm because very precise movements are needed which the robotic arm has to learn first.

Andrychowicz et al. have shown that Hindsight Experience Replay can be used to deal with this issue for robotic arms. Hindsight Experience Replay can learn efficiently from sparse rewards and can also be combined with any off-policy Reinforcement Learning algorithm. This technique is inspired by the ability of humans to learn from failures as least as much as from successes.

Hindsight Experience Replay works as follows. After an episode of gaining experiences, all transitions between the states in each training sample is stored in a replay buffer, but the goal that was not achieved is extended to a set with a goal that is reached. This can also be further extended to a set of more goals that can be achieved in the terminating state of the training sample. If the goal was to move an object to point x, but it was pushed to point y, the replay buffer would use the same transitions but

change the the goal we wanted to achieve to y. So when replaying the same experience, the agent would be successful and earn an useful reward. This does not help the agent learn how to reach the goal it wanted to reach initially, but it learns to reach other goals. Reaching those other achieved goals might be beneficial in learning how to reach the goal it actually wanted to achieve. Hindsight Experience Replay is mainly used for tasks with multiple goals, but it was shown that it also improves the training of tasks with only a single goal. Interestingly, they have shown Hindsight Experience Replay performs has problems when using shaped rewards

2.5 Hindsight Goal Generation*

3 Methodology

To provide a sound answer to how hindsight experience replay performs on harder tasks in contrast to easier tasks, the obvious approach is a quantitative approach. In order to collect data, a simulation environment is built for tasks of different difficulties. The robotic arm is trained for those tasks with hindsight experience replay. The performance for the training period is measured. The data will show the performance on the tasks over time and characteristics like learning rate and consistency is shown through the data. Data between easier and harder tasks is evaluated and compared to show how hindsight experience replay performs on different tasks. In some cases, it is clear that hindsight experience replay fails for the harder tasks. Possible reasons for the lack of performance are presented. An alternative extension for hindsight experience replay, hindsight goal generation will be used in addition to show possible approaches on solving the tasks with her for harder tasks. Hindsight goal generation will also be used on the easier tasks to make them comparable. Validity is obviously given, because the data measured is exactly the performance and learning rate of the robotic arm which is the measurement needed to evaluate the performance of hindsight experience replay. Similar results can be reproduced when repeating the data collection. The results are not necessarily exactly the same when reproduced. This is due to the nature of reinforcement learning as there is some randomness in the Markovian decision process when choosing an action. The law of large numbers states that with rising amount of samples the results will converge towards the expected probability. In context of reinforcement learning, the training time to learn needed might vary slightly, but the end performance should converge towards the same value with rising training time.

4 Simulation Environment

This section describes the environment and tools used for the experiments.

MuJoCo is a physics engine used to simulate the physical models of the environment. MuJoCo is currently still in development and is improved, so there are many different versions. For this thesis, MuJoCo 2.0 for Linux (Ubuntu 16.04) is used.

Mujoco-py is used as an interface to allow the usage of MuJoCo in python scripts. Mujoco-py is required for OpenAI Gym to work.

OpenAI developed OpenAI Gym, a toolkit to create and use environments, and use these environments to test and compare algorithms for reinforcement learning. OpenAI Gym only provides the environment part of reinforcement learning, the agent has to be written by the user or by using OpenAI baselines. The robotic environments require MuJoCo. In our case, we will create our own environments and compare them to the existing ones.

OpenAI baselines is a toolkit with high-quality implementations of reinforcement learning algorithms. It supplements the OpenAI Gym toolkit. For each of the environments by OpenAI Gym, any OpenAI baselines reinforcement learning algorithm can be used. In our case, we will focus mainly on Hindsight Experience Replay.

4.1 MuJoCo

MuJoCo stands for "Multi-Joint dynamics with Contact". It is a physics engine for model based control and was developed by Emanuel Todorov. MuJoCo was developed for research in areas with fast and accurate simulation, like robotics. As its name suggests, multi-joint dynamics and contact responses and contact responses are a main focus of the engine. They represented multi-joint dynamics in generalized coordinates and computed them with recursive algorithms. For the contact responses, they wrote algorihhms based on a modern velocity-stepping approach. MuJoCo was developed to be fast and accurate, especially for computationally intensive processes, which are common in simulation of physics. They compared MuJoCo to SD/FAST, another tool to simulate physics of mechanical systems. Even though SD/FAST uses model-specific

code, which was expected to be much fast, MuJoCo was quite comparable. Their tests with a 12-core machine showed 400.000 dynamics evaluations per second for a 3E humanoid with 18 Degrees of Freedom and 6 active contacts. Creating models for MuJoCo is quite simple. For MuJoCo, XML files can be used, which are simple to understand and provide transparency.

To use MuJoCo with Python, mujoco-py was created by OpenAI. Mujoco-py currently supports compatibility of MuJoCo with Python 3.

4.2 OpenAI

OpenAI developed OpenAI Gym and OpenAI Baselines. Both are freely accessible on Github.

4.2.1 OpenAI Gym

OpenAI Gym provides an environment to test a reinforcement learning algorithm with. OpenAI Gym already contains many environments to use, like Atari games, classic control problems and robotics. In this thesis, the robotics environment will be used, as it provides four environments with Fetch robots that use a robotic arm. But OpenAI also allows the user to create his own environments. For the experiments, a few more robotics environments will be created. For the robotics environments, MuJoCo is required. The agent and the reinforcement learning algorithms it uses that are required for reinforcement learning to interact with the environment has to be either written by the user or provided by OpenAI Baselines.

4.2.2 OpenAI Baselines

OpenAI Baselines provides a set of high-quality implementations of reinforcement learning algorithms. It can be used together with OpenAI Gym. Provided algorithms contain Advantage Actor Critic, Actor critic with experience replay, Actor Critic using Kronecker-Factored Trust Region, Deep Deterministic Policy Gradient, Deep Q-Networks, Generative Adversarial Imitation Learning, Proximal Policy Optimization, Trust Region Policy Optimization and Hindsight Experience Replay. For our purposes, Hindsight Experience Replay in conjunction with Deep Deterministic Policy Gradients will be used. OpenAI Baselines also requires Tensorflow to work.

4.3 Model

4.3.1 FetchGolf

4.3.2 FetchToss

5 Experiments

5.1 FetchGolf

5.1.1 Task Description

5.1.2 Environment

Action Space

Observation Space

5.1.3 Results

5.1.4 Discussion

5.2 FetchToss

5.2.1 Task Description

5.2.2 Environment

Action Space

Observation Space

5.2.3 Results

5.2.4 Discussion

6 Conclusion

List of Figures

List of Tables