DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Branch Dueling Deep Q-Networks for Robotics Applications

Baris Yazici

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Branch Dueling Deep Q-Networks for Robotics Applications

Branchenduell tiefe Q-Netzwerke für Robotikanwendungen

Author:	Baris Yazici
Supervisor:	Prof. Dr. Alois Knoll
Advisor:	Msc. Mahmoud Akl
Submission Date:	15.07.2020

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

Munich, 15.07.2020 Baris Yazici

Acknowledgments

Abstract

Kurzfassung

Contents

1. Introduction

1.1. Objectives

Manipulation of objects is one the most inherent action of the humankind. Humans never stood and plan about how to manipulate an object. It is a natural instict for humans to grab objects in certain ways. Even an infant human can easily manipulate different shapes and colors of objects. Manipulation helps us to use tools, gadget to achieve tasks and provide service. Therefore, manipulation skills will be central for robots of any kind. From rehabilitation to service robots, tool usage is vital to enable them to achieve their objective. Our daily manipulation tasks are shown in figure 1.1. For a complicated manipulation task first of all grapsing the object is essential. We need to first firmly grasp a water bottle to drink it. Without a firm grasp the safety of the manipulation process is projected to risk.

Nowadays, every kinds of robots are helping us produce in the industry. However, those robots at the manufacture facilities are only capable of doing same repetitive task. The task they are responsible of expands the car manufacturing to high precision microchip producing. On all those tasks robots are extremely efficient and precise. On the other hand, if a the car company decides to manufacture a new model of car with a slightly different chasis the whole process needs to be changed and program of the robots needs to be hardcoded from zero. Hardcoding every move of the robot in a slight change of a task is fundementally opposes the very nature of the robots. Main objective of this master's thesis is to enable robotic manipulators to incrementally learn to grasp tools and generalize the learned policy to unseen objects. Rather than overfitting to a certain set of objects, we measure the success rate on unseen dataset. This preassumption promises a robust actor that can adapt well on unseen object and environments.

1.2. Contribution

This projects provides two main boiler plate for developers. Firstly well documented and tested robot gripper environment based on Open AI Gym interface[**openai gym**]. Gym Environment provides easy to use interface to interact. Moreover, gripper environment complies with the most standard and well documented libraries such as python3, pybullet and numpy. Secondly, we present a novel implementation of BDQ reinforcement algorithm algorihtm adopted to most used Stable Baselines project[**stable-baselines**].

Figure 1.1.: Different manipulation skill adopted to robotic manipulators [**Kroemer201**]

1.3. Challenges

1.4. Report Layout

2. Background

2.1. Manipulation and Grasping

2.2. Reinforcement Learning

The simplest examples of learning come from our own life; we learn to walk, speak the language, or to cook. All those activities span our entire life, it influences who we are and the decisions we take in life. We know that living animals such as mammals learn from their social and asocial interactions with the environment. We know that living animals such as mammals learn from their social and asocial interactions with the environment [1]. Although we have not yet developed a full-scale theory of animal learning, we have developed computational objectives for machines to learn [2]. This computational approach can be categorized as Supervised, Unsupervised or Reinforcement Learning. This computational approach falls into three categories as Supervised, Unsupervised, or Reinforcement Learning.

In this chapter, we will consider the Reinforcement Learning objectives and problem formulation. Reinforcement Learning provides a systematic approach to maximize the reward by linking observations to actions. A reinforcement learning agent creates its data by interacting with the environment. Therefore, it is fundamentally different from supervised and unsupervised learning, where the data is already provided [2]. Another important difference is the inherent goal-oriented approach. Reinforcement learning agent maximizes the rewards for its inherent general goal. Other machine learning approaches lack this goal [2]. For instance, supervised learned software recognizing faces can be used for security reasons to detect criminals or can be easily used to unlock phones. However, a reinforcement learning agent trained to drive a car autonomously can only drive a car. In a sense, reinforcement learning provides us end-to-end learning.

A core feature of reinforcement learning is that it acts on uncertain environments and, in return, receives the observation and reward. Fundamentally, a learning agent collects this experience and tunes its action to increase the expected reward. The expected reward term refers to the end of the horizon. For example, a chess-playing agent can choose to sacrifice the queen in the next move for a checkmate in the move after. In this case, the reward would decrease when the agent loses a queen, but the goal of the agent will be satisfied by terminating the game. For a well-defined reinforcement learning system, we can speak of four main components: Policy to decide the actions, a reward to maximize the expected reward in the horizon, and a model of the environment, telling which directions the chess pieces can move. The components of reinforcement learning

are formulated based on Markov Decision Process. In the MDP chapter, we will detailly explain RL components. In the next chapters first Markov Chains, the simple version of MDP, then MDP, the slightly advanced version of Markov Chains, will be explained by some simple modifications [3].

2.2.1. Markov Chain

The weak law of large numbers has a tremendous significance on stochastic modeling. This law states that the average of a large number of experimentations converges to the real value of the probability of a particular task. As an example, if one tosses infinite amounts of the coin, the average number of heads should converge to 0.5 [4]. Bernoulli's weak law of large numbers only covers independent events. Markov proved that Bernoulli's law also holds on dependent cases [4]. As the law of large numbers suggests, if one conducts a large number of iterations on this problem, one can deduce the transition matrix. This matrix proves to be the only information one needs to compute the next state. This characteristic defines the famous Markov property; the current state captures all the necessary information one needs to predict the next state. If we extrapolate this example to slightly complex systems, for example, weather forecast, we just need today's weather report to predict tomorrow's forecast, assuming that we know the transition matrix. Naturally, one can formulate other events with Markov Chain lawnmower, and random walk are the straightforward ones in the literature[4].

It is also possible to attach rewards to Markov Chain's formulation. In figure 2.1, the transition between states is represented with the arcs. And each transition has a probability similar to the transition matrix; we defined before. Each state has an immediate reward and a value function. The immediate reward is received directly when the actor moves to that state. The value of a state represents how likely the future actor will end up collecting high rewards.

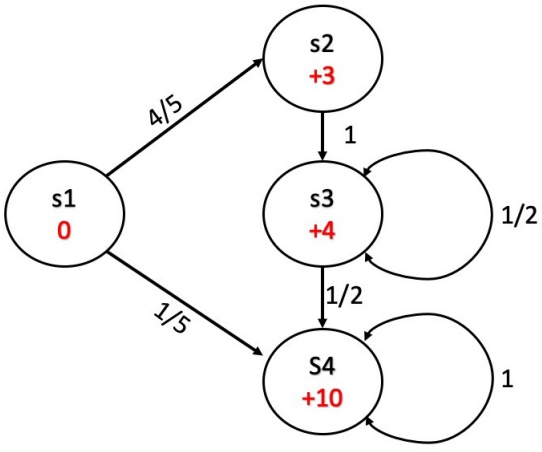

Figure 2.1.: Markov chain with transition probabilties and rewards

The value of a state will be later significant to solve Reinforcement Learning problems

through Value Iteration methods. They are one of the essential algorithms that led to the initial success of Reinforcement Learning research.

2.2.2. Markov Decision Process

Markov decision process is a slightly advanced version of the Markov Chain. It includes action on top of the Markov Chain. Reinforcement learning problems are formalized as a Markov Decision Process rather than Markov Chain because RL agents are free to choose from different actions.

MDPs first came into play as part of optimal control problem by Bellman [5]. Bellman applied dynamic programming methods to solve the MDP problem optimally. However, this methodology was not scalable to larger problems stem from the curse of dimensionality problem [**Sutton**].

The fundamental elements of MDP are as follows:

- Agent: The actor takes action on the environment to learn.
- Environment: The agent interacts with the Environment(Plant).
- Rewards: Environment returns rewards based on the interaction made by the actor.
- State: View of the environment from the eyes of the actor.

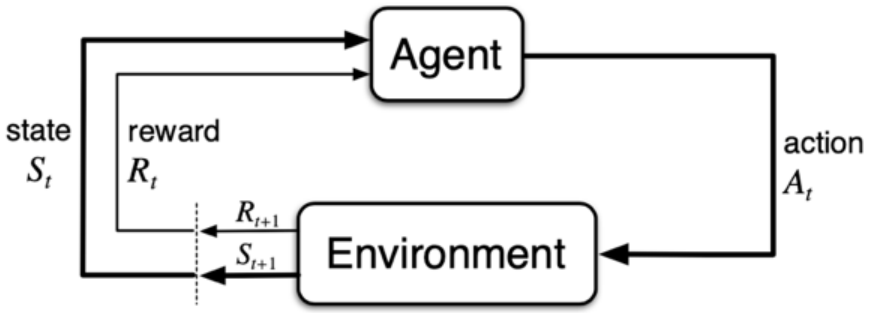

Figure 2.2.: MDP structure

The process of action follows; the agent acts on the environment at time t and environment return the reward and state of the action at time $t+1$. Based on the state of the environment at the $t+1$ agent makes another action A_{t+1}, which results in R_{t+2} and S_{t+2}.

In every MDP system, agents should be reachable to every state through a sequence of actions. The transition between states is of great value for the MDP framework. One can compute the transition probability in MDP, given the inner dynamics of the environment

[**Sutton**]. The below equation defines the internal dynamics probability. It tells, how probable it is to end up in state s' with reward r by taking action a in state s.

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = d, A_{t-1} = a\} \tag{2.1}$$

One can calculate the transition probability function from the inner dynamics' probability function.

$$p(s'|s, a) = Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in R} p(s', r|s, a) \tag{2.2}$$

Next chapter, we will dive into the definition of the reward and value function. Based on those concepts we will build the logic on how to solve MDPs optimally

2.2.3. Reward and Value function

[(TODO: Ilk paragraf)]

As defined in the Markov Chain section, rewards and value functions are the essence of value iteration algorithms. In this section, we will describe the objective of the RL problem formally. As we mentioned in previous chapters, we want the increase the number of rewards we achieve when we reach the terminal state. If we define the reward at final state T as R_T and the reward at the initial state t is R_t, returns we receive is the sum of rewards in every state.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.3}$$

G_t is the notation of expected return. We will mostly use the discounted version of the expected return calculation. We introduce a discount factor (γ, to value the rewards that the agent receives now, compare to the rewards in the future.

$$G_t = R_{t+1} + \gamma^1 * R_{t+2} + \gamma^2 * R_{t+3} + ... + \gamma^T - 1 * R_T = \sum_{k=0} \gamma^k R_{t+k+1} \tag{2.4}$$

For instance, if a rational human offered 1m Euros now, versus 1m Euros in 50 years, would usually choose 1m Euros now. Therefore, our agent also weighs the rewards it receives now, over 50 steps from the current state. In the meantime, we do not want the agent to undervalue the importance of reaching the terminal state through the highest reward sequence. Given the below example, if we introduce a discount factor of 0, the agent will always try to maximize the immediate reward and take a sequence of s1-a2-s3-a4-s4 and end up with non-optimal greedy algorithm with $G_t = 20$. But with a discount factor, in this example everything between $0 < \gamma < 1$ works, would find the optimal sequence(s1-a1-s2-a3-s4) with $G_t = 25$.

The value function is the expectation of returns while an agent is following the policy (π). Policy namely represents the probability distribution of an agent taking action a in state s. The policy is similar to the transition probability matrix in the Markov Chain section. Using policy, we can define the value function of an agent following the policy pi as below.

2.2.4. Q-Learning

Q-learning algorithms have been the core of the RL research for almost 20 years. It combines the idea of TD learning in approximating action-value function. Through this approach, the computation converges faster than state-value approximation algorithms. Another strength of the Q-learning algorithm is its off-policy nature. The Q-learning agent can learn from the results of different policies. Off-policy nature makes the agent more data-efficient. An agent is not obliged to learn from the outcome of one policy; instead, many policies can contribute to learning.

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))] \tag{2.5}$$

2.2.5. Value-Based Reinforcement Learning

Q-network

Extension of Q-networks

Branch Dueeling Q-networks

2.2.6. Policy-Based Reinforcement Learning

Entropy-Based Approach

Soft Actor Critic

2.2.7. Off-Policy RL

2.2.8. On-Policy RL

2.3. Neural Networks

2.4. Curriculum Learning

One can gradually increase the level of the difficulty of a specific task to be learned to guide the training process and speed up the convergence time. Based on phycological studies, humans learn faster when they subject to the information in the form of a unique curriculum. That is because we are subjected to curriculum learning since they were toddlers.

2.5. Simulator Choice

2.5.1. Mujoco

2.5.2. Gazebo

2.5.3. PyBullet

2.6. State of Art

2.6.1. Dexnet

2.6.2. DQL Google

3. Experimental Setup

3.1. Implementation

3.1.1. BDQ Algorighm Implementation

Object Recognition

3.1.2. Network Architecture

3.1.3. External Libraries

3.1.4. Testing Structure

3.1.5. Run the Code

3.2. Curriculum Integration

3.2.1. Curriculum Parameters

3.3. Hyperparameter Search

3.4. GPU vs. CPU

4. Evaluation

5. Conclusion & Future Work

Use with pdfLaTeX and Biber.

5.1. Conclusion

5.2. Future Work

A. General Addenda

If there are several additions you want to add, but they do not fit into the thesis itself, they belong here.

A.1. Detailed Addition

Even sections are possible, but usually only used for several elements in, e.g. tables, images, etc.

B. Figures

B.1. Example 1

✓

B.2. Example 2

✗

List of Figures

List of Tables

Bibliography

[1] E. L. Thorndike. "Animal Intelligence: Experimental Studies." In: (1911).

[2] R. S. Sutton and A. G. Barto. *Reinforcement Learning, Second Edition: An Introduction - Complete Draft*. 2018, pp. 1–3. ISBN: 9780262039246.

[3] T. Lozano-Pérez and L. Kaelbling. "6.825 Techniques in Artificial Intelligence (SMA 5504)". Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. Fall 2002. URL: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-825-techniques-in-artificial-intelligence-sma-5504-fall-2002/index.htm#.

[4] P. A. Gagniuc. *From Theory to Implementation and Experimentation*. Wiley, 2017. ISBN: 9781119387572.

[5] R. Bellman. "Dynamic programming and stochastic control processes". In: *Information and Control* 1.3 (1958), pp. 228–239. ISSN: 00199958. DOI: 10.1016/S0019-9958(58)80003-0.