UNIVERSITY of KHARTOUM

Deep Reinforcement Learning for Robotic Hand Manipulation

by

Mohammed Nagdi Bakri Mohammed 143119

Muhammed Yahia Gaffar Saeed 144082

Supervisors

Assoc Prof. Benjamin Rosman, University of the Witwatersrand, South Africa

Dr. Hiba Hassan, University of Khartoum

A thesis submitted in partial fulfillment for the degree of Bachelor of Science

in the

Faculty of Engineering

Department of Electrical and Electronic Engineering

October 29, 2020

Declaration of Authorship

We, Mohammed Nagdi Bakri Mohammed, and Muhammed Yahia Gaffar Saeed, declare that this thesis titled, 'Deep Reinforcement Learning for Robotic Hand Manipulation' and the work presented in it are our own. we confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by ourselves jointly with others, We have made clear exactly what was done by others and what we have contributed ourselves.

Signed: Mohammed Nagdi, Muhammed Yahia

Date: 10/10/2020

Abstract

Researchers have made a lot of progress in combining the advances in Deep Learning and the generalization and applicability of Reinforcement learning to the sequential decision-making process and introduce Deep Reinforcement Learning, which allows using Reinforcement Learning in high dimensional input space environments. Deep Reinforcement Learning achieved notable results in Atari-Games, continuous control tasks such as Robotics. In this project we benchmark the performance of three different deep reinforcement Learning techniques Deep Deterministic Policy Gradient "DDPG", Deep Deterministic Policy Gradient with Hindsight Experience Replay "DDPG+HER" and state-of-art policy gradient method Proximal Policy optimization "PPO", on multi-goal continuous control environments -Fetch task and HandManipulate tasks, we benchmarked the three algorithms on six different environments using sparse and dense reward settings. Deep Deterministic Policy Gradient with Hindsight Experience Replay achieves the best success-rate overall the environments when applied with sparse rewards, while both Proximal Policy Optimization and Deep Deterministic Policy Gradient were able to converge only on FetchReach environment.

المستخلص

حقق الباحثون تقدمًا كبيرًا في الجمع بين التطورات في التعلم العميق وتعميم وتطبيق التعلم المعزز في عملية صنع القرار المتسلسل , و ابتكروا التعلم المعزز العميق ، والذي يسمح باستخدام التعلم المعزز في بيئات ذات مساحة إدخال عالية الأبعاد. حقق التعلم المعزز العميق نتائج ملحوظة في ألعاب الاتاري ، ومهام التحكم المستمر مثل الروبوتات. في هذا المشروع ، قمنا بقياس أداء ثلاث تقنيات مختلفة للتعلم المعزز العميق التدرج العميق للسياسة الحتمية ، وتدرج السياسة الحتمية العميقة مع إعادة تشغيل تجربة الإدراك المتأخر وأحدث أسلوب التدرج في السياسة تحسين السياسة التقريبية في بيئات التحكم المستمر متعددة الأهداف - مهمة الجلب ومهام التلاعب باليد ، قمنا بقياس الخوارزميات الثلاثة في ست بيئات مختلفة باستخدام إعدادات المكافآت المتفرقة والكثيفة. التدرج الحتمي العميق في السياسة مع تجربة إعادة تشغيل تجربة الإدراك المتأخر يحقق أفضل معدل نجاح بشكل عام للبيئات عند تطبيقه بمكافآت متفرقة ، في حين أن كل من تحسين السياسة التقريبية وتدرج السياسة الحتمية العميقة كانا قادرين على التقارب فقط في بيئة الجلب و الوصول.

Acknowledgements

Prima facie, We are grateful to God for the excellent health and wellbeing that were necessary to complete this Thesis.

Our sincere gratitude to our research supervisors, **Prof.Benjamin Rosman**, and **Dr.Hiba Hassan**. For their motivation, enthusiasm, and immense knowledge. Without their assistance and dedicated involvement in every step throughout the process, this work would have never been accomplished.

Contents

List of Figures

List of Tables

Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
DDPG	**D**eep **D**eterminstic **P**olicy **G**radient
DQN	**D**eep **Q**-**N**etwork
HER	**H**indsight **E**xperience **R**eplay
IL	**I**mitation **L**earning
MC	**M**onte **C**arlo
MDP	**M**arkov **D**ecision **P**rocess
MuJoCo	**Mu**lti-**Jo**int dynamics with **Co**ntact
PG	**P**olicy **G**radient
PG	**P**olicy **G**radient
RL	**R**einforcement **L**earning
TD	**T**emporal **D**ifference

Chapter 1

Introduction

This chapter provides an insight into the field of dexterous manipulation of robotic arm, the motivation behind this work, a brief description of the problem that face the development of this field, the objective of the study.

1.1 Overview

For grown up human beings grasping object – mug, book, smart phone, small jar …etc. - is such an easy task, for new born babies grasping an object is the first step for drawing writing and other dexterous hand manipulations, but it will take him at least a year to develop the coordination to pick up and hold things securely in his hands. He'll start working intensively on this skill by 3 months and make leaps with each passing month, but for a robot it's another story, The Robotics in today's industrial environments are not intelligent, for example the welding and painting robots are "hard-coded" - they are designed and programmed to do specific tasks in specific environment - today's grippers are extremely powerful for gripping predetermined objects, In the past decade, robots have become very good at working in tightly controlled conditions, such as on car-assembly lines. You can build a robotic system for one specific task — picking up a car part, for example, you know exactly where the part is going to be and where the arm needs to be, because the robot picked up the same thing from the same place for the last thousand time. But the world is not a predefined assembly line, for human it's not hard to interact with the countless objects and environments found beyond the factory gates a trivial task, it is tremendously difficult for robots. For

the robot to be intelligent it must possess three attributes, first perception is the process of acquiring, interpreting, selecting and organizing sensor information, second decision making is the process of choosing the action a robot should perform given its inner representation of the world state and third action selection, the main problem for action selection is complexity of controlling the multi-degree of freedom joints and the stochasticity of the real world – to clarify the stochasticity of the real world takes into consideration the following example: you are driving a car in slippery icy road and you select action "turn left" but due to the slippery icy road the car doesn't turn left.

1.2 Problem Statement

We want to teach an intelligent robot gripper and arm to do dexterous manipulation such as sliding, pick-and-place, fetch-reach …etc. using different deep reinforcement learning algorithms to solve the task of dexterous manipulation and then benchmark the results of each algorithm, and conclude why each algorithm succeeded or failed to solve the task.

1.2.1 Objective

- To perform literature survey of previous methods used in the field of dexterous manipulation using deep reinforcement learning.
- To benchmark different deep reinforcement learning algorithms to do dexterous manipulation using robotic gripper and arm environment, and conclude why each algorithm succeeded or failed to solve the task, achieving this would open up the possibilities of robot agent to learn to interact with the environment without prior knowledge.

1.3 Methodology

The focus of this work is to use deep reinforcement learning methods such as policy gradient methods for dexterous manipulation in Mujoco Physics Simulator. Three deep reinforcement learning algorithms were used: deep deterministic policy

gradient (DDPG), proximal policy optimization (PPO), Deep Deterministic Policy Gradient with Hindsight Experience Replay (DDPG-HER).

1.4 Thesis Layout

The rest of this thesis is structured as follows:

- **Chapter 2 – Literature Review:**

  Introduces a critical reviewing of previous related research in the field along with our selected algorithm used for benchmarking.

- **Chapter 3 – Theoretical Background:**

  Gives a detailed background of the field of deep learning and deep Reinforcement learning algorithm that touch the basics needed for implementing our proposed solution.

- **Chapter 4 – Methodology:**

  Provides a detailed description of the approaches being used in this study along with their implementation.

- **Chapter 5 – Result and Discussion:**

  Presents and discusses the results and findings obtained during the development of the solution.

- **Chapter 6 – Conclusion:**

  Provides the Conclusion and an insight into the possible future of the work.

1.5 Contribution

1. Mujoco Installation Blog on Medium "https://medium.com/@mohammed.yahia3/mujoco-installation-5fb47687843a"

2. Submitted Paper with title "Deep Reinforcement Learning for Robotic Hand Manipulation" on International Conference on Computer, Control, Electrical, Electronics and Engineering (ICCCEEE20)

Chapter 2

Literature Review

2.1 Introduction

What is reinforcement learning [6] ?

Reinforcement learning is a part of machine learning concerned with building a software agent with the goal of getting maximum cumulative reward or reaching the"optimal policy". Reinforcement Learning model, consist an agent and environment connected through three signals - state, rewards and action - at each timestep the state for both the agent and the environment could be represented as $s \in S$, for this state s an action $a \in A$ chosen by the agent, the environment then provide the agent with reward $r \in R$. The agent maps each state into action using policy π, the goal of the agent it to reach the optimal policy and hence maximize total cumulative reward.

- set of discrete/continuous environment states, S
- set of discrete/continuous actions A
- set of scalar rewards R

The agent lives in a stochastic world so taking the same action a in the same state s in different time steps t_1 and t_2 can result in different reward and next state.
Reinforcement Learning - RL - is one of three basic machine learning approaches, the other two are **supervised learning and unsupervised learning**. The Reinforcement Learning agent has no labels as in the case of supervised learning, Reinforcement learning is not as blind as in the case of unsupervised learning,

where the agent takes another signal reward besides the observations and uses it to reach the optimal policy.

Supervised learning : y = f(x)

We have features - data - "x" and labels for those features "y" The goal of supervised learning model is to find a function approximation which maps those data into the corresponding labels y, with the goal to generalize well to unseen data.

Reinforcement Learning: y = f(x) , z

We have some information about the state the agent is currently in - observations - , the rewards the environment provided to the agent "z" , after the agent performs an action "y" on the environment. The goal of the RL agent is to find optimal policy - map from state into action - with the goal of maximizing the total cumulative reward.

The Generality and flexibility of reinforcement learning approach comes with price, Reinforcement learning is considered to be more tricky than Supervised and unsupervised Learning, some challenges when using Reinforcement Learning[7]:

First, Exploration-Exploitation problem:

The agent should not only be stuck and exploit the policy he has learned, but from time to time has to explore new actions from each state because maybe there are better actions than the action the agent has already learned from its policy

Note:

There must be balance between exploration and exploitation, too much exploration may decrease the total cumulative reward and much exploitation the agent may not be able to reach the optimum policy

Second, Credit Assignment Problem:

Without credit assignments all actions in the trajectory- all states, actions rewards from start of the environment until we reach the terminal 'final' state - will be updated with the same weights but generally the last actions in the trajectory contributed more to the reward than the first action in the episode. So it's logical to assign more weights to the actions near the terminal rewards and provide more discounted weights to first actions in the trajectory.

2.2 Reinforcement Learning in Robotic

A remarkable number of various problems in robotics may be naturally phrased as problems of reinforcement learning. Reinforcement learning empowers a robot to autonomously discover an ideal behavior through trial-and-error interaction with it's environment. Rather than expressly detailing the solution of the problem,the designer of the control task gives a one step feedback in term of scalar objective function that measure the performance of the robot.

Reinforcement learning has been generally utilized in many different robotics containing mission about continuous control tasks, running in a fast physics simulator [8][9], bipedal and quadrupedal locomotion[10][11][12] and acquiring dynamic character controllers directly from monocular video through a combination of pose estimation and deep reinforcement learning[13]. Manipulation in robotics, mainly inspired by human hand, referred to as dexterous manipulation is a topic that heavily studied and linked to Reinforcement Learning one definition from[14]: Dexterous Manipulation is an area of robotics in which multiple manipulators, or fingers, cooperate to grasp and manipulate objects.It differs from traditional robotics mainly in that the manipulation is object-centered. The problem designed in terms of the object manipulated, how it should behave, and what forces they should exert upon it and it's divided into two parts: grasping which means holding the object in a fixed position regarding the hand, and internal manipulation which means the controlled movement of the grasped object in the gripper or hand workspace. Many of the ancient work on dexterous manipulation focuses on simplifying the task mechanically by using a low degree of freedom gripper and applying it on a simple task like grasping and rolling object and with the help of new deep reinforcement learning methods it drastically changed the task that can be achieved by gripper without simplifying the control the problem at the expense of reduced flexibility.

One of the most fundamental and branching point when solving reinforcement learning problem is whether we had a access to the model of the environment. The model of the environment in reinforcement learning consist of two part transition probability along with the reward function.

In this Literature Review we identify four main themes for the past work in solving manipulation problem and robotic in general :

1. **Model Based Reinforcement Learning:** Model based trajectory optimization methods have demonstrated impressive results in simulated domains, see section 2.3.

2. **Model Free Reinforcement Learning:** Model-free RL methods do not require a model of the dynamics, and instead optimize the policy directly, see section 2.4.

3. **Imitation learning :** In imitation learning, demonstrations of successful behavior are used to train policies that imitate the expert providing these successful trajectories,see section 2.5

4. **Combining RL with demonstrations :** These methods have been used to effectively combine demonstrations and RL to enable faster learning, see section 2.6.

2.3 Model Based Reinforcement Learning

In model based Reinforcement Learning we have to learn the explicit transition and/or reward model and plan based on the model.

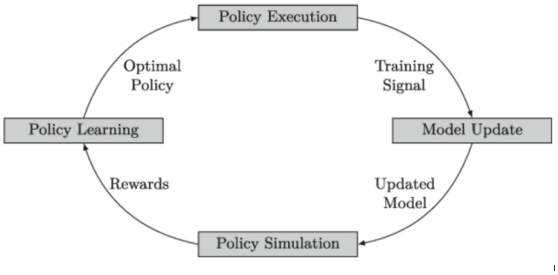


FIGURE 2.1: A flow diagram of model-based RL

As seen in figure 2.1 the agent perform two task when it receive the state and reward form the environment it first update the model and based on it's model it is going to plan to choose what may be a good action, the next task is to update it's policy and value function. In practice it's considered to more computationally expensive and time consuming which it's main drawback but when we do it right it is believed to have increase in sample efficiency which means it need less data and less interaction with the environment to arrive to the optimal policy, another important benefit of model based approaches is transferability and generality which means a model can be reused for achieving different tasks.

In[15] the used model based Reinforcement Learning to manipulate Deformable Linear Object (DLO), such as cables, ropes, ...etc , which used widely in daily life and industry. The absence of universal model that specify DLO regardless of the material and environment make it difficult to achieve target manipulation for a robot. The RL problem is presented as follows : a DLO configuration that is stochastically modeled with input and outputs. Whereas the action is input to the robot the output is the new DLO configuration that resulted from the action as well as cost given to the robot, to solve the high dimensional planning problem they used sample efficient model based method called PILCO (Probabilistic Inference for Learning Control)- reference here- . The task is to manipulate a rope so that the two end's of the are approaching each other and for simplicity they set the rope in 2-D space, they used Baxter robot (Rope manipulator) and one meter rope and built a rope manipulator system using Robot Operating System (ROS) with three main component : a Central PC , Control point detector and Rope manipulator. They conducted three experiment with 25 iteration each and found that the Baxter robot employed the optimal policy that minimizes the distance between the two end points.

In[16] they introduced one shot learning – the agent ability to perform the task successfully from one or few attempts- of manipulation skill, They used model-based reinforcement learning algorithm that uses online adaptation of its dynamics model combined with a coarse prior model obtained from experience on other prior tasks. Also used Model Predictive Contol (MPC) method based on iterative linear quadratic regulator (LQR) under the current adapted model to choose the action needed, which is a local linear approximation to the nonlinear dynamics of the system, The neural network prior must be trained on previous interaction data in order to provide a helpful prior model of the system dynamics. This methods is able to perform different manipulation tasks such as inserting a toy nail into a toolbench, placing a wooden ring on a tight-fitting peg and many other task as presented in the paper.

In survey of model based RL application in robotic[17] the author stated that its main disadvantage is that model-based RL algorithms heavily depend on the model's ability to accurately represent the transition dynamics hence it's important to carefully choose the transition model that accurately represent the task, most of the literature have in common is that they solve simplistic task and struggle to translate to real-world manipulation since learning complex models on real

world systems with significant contact dynamics that require reliability and robustness is very difficult.

2.4 Model Free Reinforcement Learning

Model Free reinforcement learning is when we learn directly from interacting with the environment, where we don't the transition probability distribution and the reward function which collectively called the model of the environment and hence the name model-free, a model-free algorithm can be thought as an explicit trial and error.

The idea is to move toward sampling because we want to get rid of the assumption that we have an accurate model with the goal is to optimize the value function and the policy with an unknown MDP. Advantages of the model-free RL are:

- It's computationally less complex as compare to the mode-based.
- There is no need for accurate representation of the environment to be effective which makes is more fundamental than model-based method

On the other hand it needs more experience gathered from the environment which makes exploration more dangerous in robotic another disadvantage that it doesn't have an explicit plan of how the dynamic of the environment affects the system, especially in response to action that took previously.

The two main approaches used to represent and train agent in model-free Reinforcement Learning are:

1. **Policy Optimization.** The Methods in this family represent the policy explicitly as $\pi(a|s, \theta)$ They optimize the parameters θ ither directly by gradient ascent on the performance objective $J(\pi_\theta)$ or indirectly, by maximizing local approximations of $J(\pi_\theta)$. This optimization is almost always performed on-policy, which means that each update only uses data collected while acting according to the most recent version of the policy. Policy optimization also usually involves learning an approximator $V_\phi(s)$ for the on-policy value function $V^\pi(s)$, which gets used in figuring out how to update the policy.

2. **Q-Learning** Methods in this family learn an approximator $Q_\theta(s, a)$ for the optimal action-value function $Q^*(s, a)$.Typically they use an objective function based on the Bellman equation. This optimization is almost always performed off-policy, which means that each update can use data collected at any point during training, regardless of how the agent was choosing to explore the environment when the data was obtained. The corresponding policy is obtained via the connection between Q^* and π^* he actions taken by the Q-learning agent are given by:

$$a(s) = \underset{a}{\operatorname{argmax}}\, Q_\theta(s, a) \tag{2.1}$$

In this work[18] they successfully adapted the distributional perspective on Reinforcement Learning to continuous control setting. They combine it within a distributed framework for off policy learning in order to develop what the called a Distributed Distributional Deep Deterministic Policy Gradient or D4PG for short.

The work on this paper is based on Deep Deterministic Policy Gradient (DDPG) algorithm[2], DDPG is a model free algorithm where it combine Deterministic Policy Gradient(DPG) with Deep Q-Network (DQN) where DQN stabilize the learning of the Q-function by experience replay and frozen target network , and DDPG extend the learning to the continuous space with the actor-critic framework while Learning a deterministic policy. D4PG applies a set of improvement to DDPG to make it run on a distributional fashion by implying a Distributional Critic, Multiple Distributed Parallel Actor and prioritized experience replay.

In their experiment, they used a simulated hand model implemented with MuJoCo, which consists of 13 actuators that control 22 degrees of freedom. For these experiments see fig 2.2, the wrist site attached to a fixed location in space, about which it is allowed to rotate axially. That enables the hand to pick objects and rotate the palm position to manipulate them. The first task is to catch a falling cylinder dropped onto the hand from random height refer to as (catch) in MuJoCo, the second task (Pick and Place) that requires the agent to pick up an object from the tabletop and then maneuver it to a target position and orientation. The final task (rotate-in-hand) is one wherein a broad cylinder must be rotated in-hand to match a target orientation.

The result found in this experiment using the D4PG algorithm indicates that the biggest performance gain came from the use of the N-step return that added to

FIGURE 2.2: Sequences of frames illustrating the dexterous manipulation tasks. Top to bottom: 'catch', 'pick-up-and-orient', 'rotate-in-hand'. The translucent objects shown in 'pick-up-and-orient' and 'rotate-in-hand' represent the goal states.adapted from Lillicrap et al.

the DDPG, they also found that the use of priority was less crucial to the overall D4PG algorithm especially on harder problems. While the use of prioritization was able to increase the performance of the D3PG algorithm, They found that it can also lead to unstable updates.

In[19] they introduced a novel technique called Hindsight Experience Replay, which allows sample-efficient learning from binary or sparse reward function hence avoid the need for complex reward shaping. The main idea in HER is to replay each episode with a different goal other than the one the task is designed to achieve, e.g. one of the goals already achieved in this episode.

The experiment also demonstrated the task of manipulating an object with a robotic arm. Which they used MuJoCo environment on three different tasks: sliding, pushing, and pick-and-place with sparse reward function that only indicate whether we achieved the task successfully or not at the end of the episode. To test their technique on MuJoCo gripper they compared the performance of DDPG with and without HER. Then they checked if HER improves performance in the single-goal setup. They analyzed the effects of using shaped reward functions. After that, they compared different strategies for sampling additional goals for HER.

From the result obtained, there is some key point:

- DDPG without HER is unable to solve any of the tasks, with the addition of HER the algorithm solved all tasks almost perfectly which confirms that HER is a crucial element in learning sparse rewards.
- DDPG+HER performs much better than pure DDPG even if the goal state is identical in all episodes.
- HER learns faster if the training episodes contain multiple goals.

In[1] they proposed a new policy gradient algorithm for Reinforcement Learning, that alternate between optimizing the surrogate objective function using stochastic gradient ascent and sampling data through interaction with the environment.

The idea is to clip the ratio between the new and the policy denoted by $r(\theta)$ to optimize the objective function $J(\theta)$ that forcing the $r(\theta)$ to stay within a small interval around 1, precisely $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyper-parameter. PPO has been tested on a set of benchmark tasks (MuJoCo reacher and humanoid) and proved to produce good results with much greater simplicity.

In[20] used RL to learn dexterous in-hand manipulation policies that can perform vision-based reorientation on a physical Shadow Dexterous Hand -see figure- without relying on any human demonstration.

The method employed in the paper are Generalized advantage estimator[3] and Proximal Policy Optimization(PPO), They used the distribution of simulations with randomized parameters and appearances to collect data for both the control policy and vision-based pose estimator.

MuJoCo physics engine used to simulate the physical system and used Unity to render the images for training the vision-based pose estimator. For hardware Shadow Dexterous Hand used- is a humanoid robotic hand with 24 DoFs-, RGB cameras, PhaseSpace tracking, and joint sensor tracking.

From the result, they found that the policy exhibit the natural ways of grasp found in humans. The policy also discovers the many strategies for dexterous in-hand manipulation described by the robotics community, such as finger pivoting, finger gaiting, multi-finger coordination, the controlled use of gravity, and coordinated application of translational and torsional forces to the object.

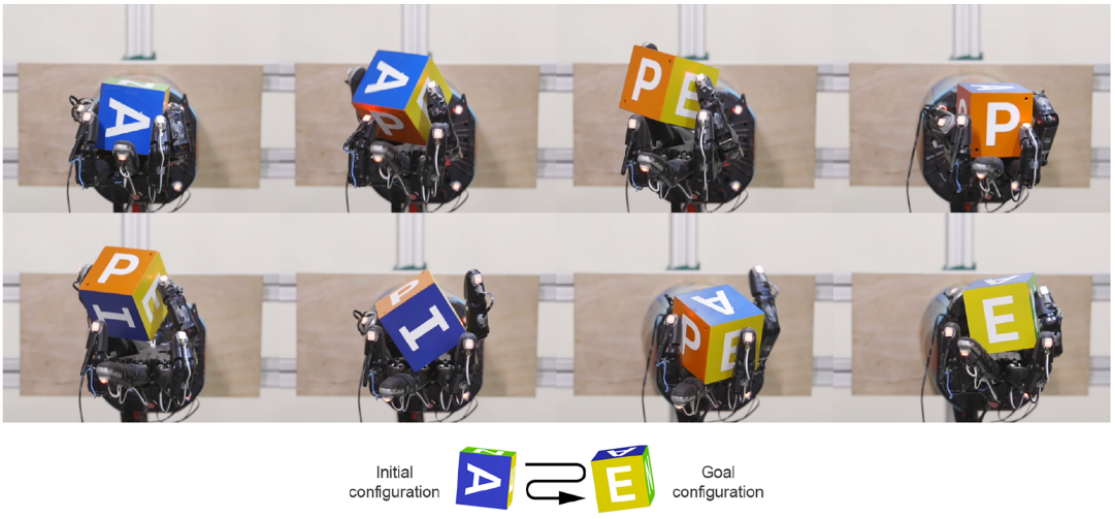


FIGURE 2.3: A five-fingered humanoid hand trained with reinforcement learning, adopted from Schulman et al.

2.5 Imitation learning

The process of imitation Learning is one by which an expert or teacher provide a set of demonstration- sequences of state and action- to the agent to learn a policy that solves a given task. We use it to help us bootstrap or completely learn a new policy and it's applied in robots as a technique to reduce the complexity of search spaces for learning. Robot Imitation Learning was developed to solve multiple problems that is associated with instantiating the controllers. This approach is in contrast to explicit programming which associated with the inability of human programmer to identify every scenario and explicitly determine what course of action must be associated with every state, and it differ from Reinforcement Learning in which the user give the agent the reward without demonstrating the appropriate behavior specially in large state space where the exploration is expensive like to learn how to fly a helicopter. Imitation Learning is useful when it's easier to demonstrate the desired behavior rather than:

1. Specifying a reward that would generate such a behavior.
2. Specifying the desired policy directly.

Imitation Learning go throw 4 major stages :

1. **Source of demonstrations :** Which is the starting stage where the agent capture the action that it learn from , which is performed using multiple methods like on teacher sensors, on learner sensors or external sensor.

2. **Feature representation :** The data then is extracted and used to represent the states and the surrounding of the agent.

3. **Learning from demonstration:** The features extracted is then used to learn the agent to mimic the behavior shown by the teacher demonstration, the three major techniques used in this step are Classification, Regression and Apprenticeship learning.

4. **Refine Policy:** Finally we can use this extra step to act on the policy and refine it based on it's performance, this step may use the input from the teacher for evaluation. Many of the techniques used in this step fall into the umbrella of Reinforcement Learning, some think of this step as post-learning but usually this step happen in conjunction to learning from demonstrations.

The field of imitation Learning is considered a young field, a number of surveys has been exist that review the work to date[21][22], that the task of learning by imitation can be further broken down.The simplest form of imitation learning is behavior cloning (BC). It is a method by which human sub-cognitive skills can be captured and reproduced in a computer program. One of the most important research example is ALVINN[23] which is an vehicles that take input data from camera and and learn to map these data from sensors into steering and driving autonomously. Training has been conducted using simulated road images, this is the first project used Imitation Learning in general it was carried out in 1989 by Dean Pomerleau.

The way behavioral cloning works is rather simple. Given the expert's demonstrations, we divide these into state-action pairs, we treat these pairs as i.i.d. examples and finally, we apply supervised learning. The loss function can depend on the application. Although BC has applied successfully in many project it also suffer from problem where there is a long term planning, pure imitation learning methods cannot exceed the capabilities of the demonstrator since they lack a notion of task performance.

2.6 Combining RL with demonstrations

Reinforcement Learning can be used on its own to learn a policy that works efficiently for the robotics task as demonstrated in previous sections. However, if a policy is learned from demonstration like in pure imitation Learning, reinforcement learning can be applied in order to improve and fine-tune the parameters. The policy enhancement by RL can be necessary if there is a physical discrepancy between the agent and the teacher to reduce the error while acquiring the demonstration. Reinforcement learning can be used to find a policy that performs a task in a way that doesn't look normal for the human observer but achieves better results, an impressive example of this is the application of RL with demonstration is training an agent to play the board game of "GO" that rival and outperform human expert[24]. Moreover, RL can be useful to train a policy for an unseen situation that the teacher doesn't cover in the demonstration.

This work[25] introduce a continuous actor-critic reinforcement learning for the control of multi-function myoelectric devices. By the use of simulated upper-arm robotic prosthesis. Using a simulated upper-arm prosthesis to demonstrate the ability to control limb from myoelectric data using binary human training signal. The work on myoelectric control developed on this paper was the first to introduce this approach that encourages the online learning of new amputee-explicit movements dependent on a one-dimensional input signal given by the user of the prosthesis. They proposed an adaptive algorithmic approach to generalized myoelectric control that pairs online human training of controllers with a statistical machine learning algorithm called continuous actor-critic reinforcement learning (ACRL).

Two experiments conducted, first learning a control policy using a fixed goal-based reward, the second experiment tested the ability to learn a control policy from a human-delivered reward signal. The method used was a continuous actor-critic algorithm and EMG data acquisition in a robotic arm. The ACRL system was trained online with repeated cycles employed through the training EMG data.

From the result obtained, they found the Advantage of using ACRL that it doesn't require manual shaping of the input signal parameter and can develop accurate velocity control policy, but the reward and feedback must be applied consistently to the framework. The system was capable of learning a two joint velocity control task, and demonstrated the ability that it could change over time depending on the

continuous user input, and set one of the first incremental steps toward adaptable intelligent myoelectric prostheses.

As a development based on the last work, a work [4] they developed a method that could enable someone with an amputation to use their non-amputated arm to teach their prosthetic arm to learn how to move in a natural and coordinated way, to overcome a fundamental issue of how to overcome the mismatch between the number of function that an amputee can achieve in any moment and the number of function available in a modern powered prosthesis. Such development also could exploit the muscle synergies already learned by the user.

We can think of a myoelectric prosthesis as a wearable robot that responds to EMG signal for control, hence that the desired behavior could learn from demonstration (LFD) provided by the user.

In the experiment, they explored prosthetic LFD with able-bodied participant then extended to a participant with someone with an amputation, the robotic arm used is Bento Arm that had a 5 DOF including shoulder rotation, elbow flexion/extension, wrist pronation/supination, wrist flexion/extension and hand open/close. With 16 channel Desley Tringo Lab to record EMG signal for Data Acquisition and the motion capture glove for joint angle configuration for 3 DOF that represent wrist flexion/rotation($\theta^*_w f$) and hand open/close (θ^*_h). The learned control policy as seen in figure 2.4 should pick actions such that the actuators instantaneous position matches the joint configuration demonstrated by the subject using the motion capture glove.

The result obtained from learning from demonstration using Actor-Critic Reinforcement Learning. The ACRL learner showed continuous improvement in performance over learning. Most importantly, the performance at the end of the learning phase was consistent with the performance during actual user control in the testing phase for all subjects, though the system mostly captures the intended behavior of the user, it doesn't encapsulate the finer movement of the user and this problem may be solved by a larger dataset (training over multiple sessions).

In this work[5] they showed a model free reinforcement learning that can effictively scale up to complex dexterous manipulation tasks using high dimensional human-like five fingered hand, and try to solve them with and without human demonstration. Four manipulations tasks employed for this experiment that exhibit most of the technical challenges: Object relocation, In-hand Manipulation, Manipulating

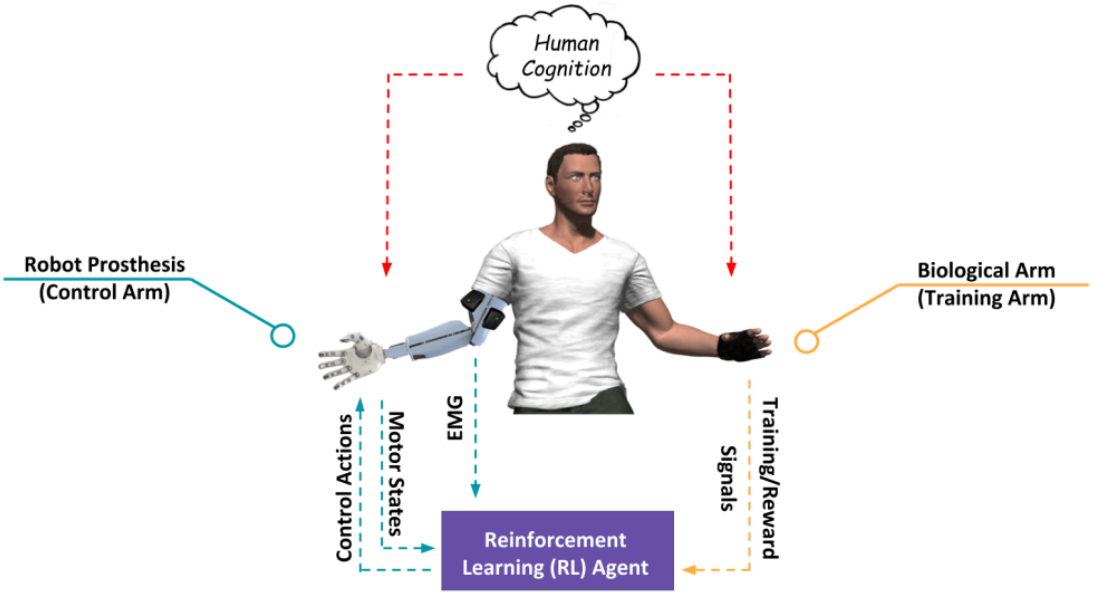


FIGURE 2.4: Schematic showing the flow of information, adopted from Vasan and Pilarski

Environmental Props which involves modification of the environment like opening doors or moving furniture, and Tool Use which require coordination between the arm and the fingers to use the tools.

As for the algorithm used they proposed a method DAPG (Demo Augmented Policy Gradient) that uses a combination of Natural Policy Gradient (NPG) and Behavior Cloning to reduce the sample complexity and help with exploration. The goal is to compare model-free RL from scratch using two state-of-the-art deep reinforcement learning methods, DDPG and NPG, against demonstration-based counterparts, DDPGfD(DDPG from demonstration) and our DAPG algorithm.

From the result obtained they found that with the reward shaping NPG achieved high success percentage compared to DDPG. DDPG is considered to be sample-efficient but requires considerable hyper-parameter tuning though, which can explain the difficulty to scale to complex dexterous manipulation tasks. Although the incorporation of human knowledge via reward shaping is thought to be helpful, the resulting model-free policies often exhibit unnatural looking behaviors and are too sample inefficient to use it on the physical hardware. On the other hand, the DAPG algorithm acquires policies that not only exhibit more human-like motion but are also substantially more robust and achieved a better result than DDPGFD and can be 30x more sample efficient than model-free RL see the table2.1 below

Method	DAPG	(sp)	RL	(sh)	RL	(sp)
Task	N	hours	N	hours	N	hours
Relocation	52	5.77	880	98	∞	∞
Hammer	55	6.1	448	50	∞	∞
Door	42	4.67	146	16.2	∞	∞
Pen	30	3.33	864	96	2900	322

TABLE 2.1: comparison between the used algorithms in Rajeswaran et al.

that compare between DAPG with sparse reward to RL (Natural Policy Gradient) from scratch with shaped (sh) and sparse task completion reward (sp).

Another work [26] that study how model-free RL can be scaled up to learn a variety of dexterous manipulation tasks with multi-fingered hands directly in the real world, using general-purpose neural network policy without manual controller or policy class design.

In the experiment conducted, to demonstrate the generalizable nature of the model-free deep RL algorithms, they considered two different hardware platforms:

1. Dynamixel Claw: A custom-built 3 fingered hand, it is a powerful, low latency, position controlled 9 DoF manipulator.

2. Allegro Hand: The Allegro hand is a 4 fingered anthropomorphic hand, with 16 degrees of freedom, and can handle payloads of up to 5 kg.

The three distinct tasks for evaluating the robotic hand are valve rotation, box flipping, and door opening. The algorithms used for solving these tasks are: model-free algorithm called vanilla policy gradient (REINFORCE) and Demo Augmented Policy Gradient(DAPG).

From the result obtained they showed that the tasks required can be solved using model-free on-policy reinforcement learning algorithms . This requires 7 hours for valve turning, 4 for box flipping and 16 hours for door opening, with almost 100 success rate over 10 evaluation roll-outs for both systems and were able to quickly learn the right behavior with the same hyper-parameters. While they found that model-free deep RL is generally practical in the real world, the number of samples can be further reduced by employing demonstrations. With only a few demonstrations, they found that demonstration augmented policy gradient (DAPG) can speed up learning significantly, dropping learning time by 2x.

Chapter 3

Theoretical Background

3.1 Reinforcement Learning Framework The Problem

Reinforcement Learning (RL) is used to solve sequential decisions problems. Generally in Reinforcement Learning, the agent's goal is to get maximum cummulative reward

3.1.1 Markov Decision Process:

Markov decision problem theory and computation is based on using backward induction (dynamic programming) to recursively evaluate expected rewards, so the problem at hand must be described as Markov Decision Process "MDP".

3.1.2 Markov Decision process properties:

1- Markovian Property:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_{1,\dots,t}) \tag{3.1}$$

The future is independent of the past given the present. Which means the current state is enough to get the transition probabilities and immediate rewards following

the action a.

2-Stationary:

The physics - rules - of the environment don't change, the transition probability from states doesn't change, and the reward distribution function also doesn't change. The agent is the only element of change in the model.

3.1.3 Components of Markov Decision Process

Assuming finite MDPs, then Markov Decision Process is ideal to formally describe an environment for reinforcement learning and consist of the following components:

- State $s \in S$ where S is a finite set of states, and the state describes the environment.
- Actions $a \in A$, where A is a finite set of actions.
- Reward

$$R^{a}_{ss'} = r(s, a, a^{'}) \tag{3.2}$$

$$r_{t+1} = r(s_{t+1}, s_t, a_t) \tag{3.3}$$

- Transition model

$$P^{a}_{ss'} = P(s_{t+1}|s_t, a_t) \tag{3.4}$$

The probability of transition from state s to state $s^{'}$ when the agent takes action a.

- Discount Factor

$$\gamma \in [0, 1] \tag{3.5}$$

- Policy represents the "master plan" to solve the problem. It is represented by mapping from states into actions π(a|s) where for each state s it outputs the best course of actions to follow.

- π^* is the optimal policy, the optimal mapping from states into actions and hence the optimal solution to an MDP.

3.2 Reward hypothesis:

The goal of the agent can be expressed as maximizing the **total expected cumulative reward**.
Utility "U": is the sum of rewards the agents observe in its **trajectory** -sequence of states transitions from the start state until the terminal state - but problem arises how can we determine which of two policies is better given infinite utility ?!!

Sequence of Rewards Assumptions:
It's essential to specify how the agent takes the future into account when deciding action to take in the present, there are **three different models** to judge this subject.

- **First, The finite-horizon model,**

the agent has to optimise its expected reward for the next h steps, given the agent at time-step t = t_0.

$$E(\sum_{t=t_0}^{h} r_t) \tag{3.6}$$

When using the finite-horizon model the agents policy becomes function of both current state and remaining time-steps $\pi(a|s,h)$ The agent ends up taking different actions at the same state depending on how many time-steps are left • **Second, Average-reward model**

$$\lim_{h\to\infty} E(\frac{1}{h}\sum_{t=t_0}^{h} r_t) \tag{3.7}$$

One problem of this model is no way to distinguish between two policies with the same utility -return - one policy gains a large amount of reward in first steps and the other gains large amounts of steps in later steps.

- **Third, Infinite-horizon discounted model.**

It takes all of the trajectory rewards into account with geometrically discount factor γ where $\gamma \in [0,1]$

$$E(\sum_{t=t_0}^{\infty} \gamma^t r_t) \tag{3.8}$$

Discount factor γ can be seen as an interest rate, a probability of living another step, or mathematical trick to bound the infinite sum into finite and ensures the agent model will converge. In the case of infinite-discounted model the maximum return at each trajectory is

$$\frac{R_{max}}{1-\gamma} \tag{3.9}$$

3.3 Policies

A policy, representing the agent's behavior, mapping states to deterministic actions or probability distributions "the probability to take each action given state". The policy π can be

Stochastic Policy a $= \pi(a|s)$ Returns the probability of action executed in the given state.

Deterministic Policy a $= \pi(a|s)$ Tells the agent with certainty which action to take. Generally the goal of the reinforcement learning is to find an optimal policy π^* helps the agent to choose the best course of actions to maximize total cumulative reward.

State Value function Value function [27] - also known as the state value function - yields the **expected** return if the agent starts in state S, then follows policy π, $V_\pi(s)$.

$$v_\pi = E_\pi[G_t|S_t = s] \tag{3.10}$$

$$V_\pi(s) = \sum_{a\in A} \pi(s|a)[R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_{\pi(s')}] \tag{3.11}$$

3.4 Bellman Equations:

The value of any state in the MDP can be calculated as the sum of the immediate reward and the discounted value of the next state [27] .

$$G_{discount} = R_{t+1} + \gamma.R_{t+2} + \gamma^2.R_{t+3} + \gamma^3.R_{t+4}..where\gamma \in [0,1] \tag{3.12}$$

we know

$$v_\pi = E_\pi[G_t|S_t = s] \tag{3.13}$$

and using bellman recursive operation we have

$$V(s_t) = r(_{t+1}) + \gamma * v(s_{t+1}) \tag{3.14}$$

3.5 Action Value Function:

Action value function Q(s,a) [27] , the agent start in state S takes action a and then follows policy π. **For** $s \in S$:

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] \tag{3.15}$$

$$v_\pi(s) = q_\pi(s, \pi(s)) \tag{3.16}$$

3.6 Solving MDP:

The goal of the reinforcement learning agent is to find an optimal policy ,determine the optimal policy π^* given the infinite-discounted horizon.

Interactions $\rightarrow$ rewards and value function $\rightarrow$ policy

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)[R_s^a + \gamma \sum_{a \in A} P_{ss'}^a v_\pi(s^{'})] \tag{3.17}$$

The optimal value of state, is the reward the agent get start at state s and following the **optimal policy** π^* thereafter,

$$V^*(s) = max_\pi V^\pi(s) \tag{3.18}$$

Also optimal value function can be defined,

$$V^*(s) = max_a(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')), s \in S. \tag{3.19}$$

$$\pi^*(s) = argmax_a(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')), s \in S. \tag{3.20}$$

3.6.1 Dynamic Programming

We assume the agent has full knowledge of the environment(Markov Decision Process) that characteristics the environment.

3.6.1.1 Policy Evaluation

Takes a policy π and returns an estimate to the state-value function V_π this approach the bellman update is applied to the state-value function until the value-function changes are nearly unnoticeable [28].
Policy Evaluation algorithm

Algorithm 1 Policy Evaluation [28]

Input: MDP, policy π small positive number θ
Output: V $\approx v_\pi$
Initialize V arbitrarily (e.g, V(s) = 0 for all s $\in S^+$)
repeat
$\triangle \leftarrow 0$
for s $\in S$ **do**
    v $\longleftarrow V(s)$
    $V(s) \leftarrow \sum_{a \in A(s)} \pi(a|s) \sum_{s',r \in R}$ p(s$'$, $r|s,a)(r + \gamma V(s')$
    $\triangle \leftarrow$max($\triangle$,|v -V(s))**end for**
until $\triangle < \theta$

3.6.1.2 Policy Improvement

Takes an estimate V of the state-value function V_π corresponding to policy and returns an improved - or equivalent - policy π' , where $\pi' >= \pi fors \in S$. This algorithm first constructs an action-value function estimate Q-table. Then for each state $s \in S$ the algorithm selects the action a which maximize Q(s,a),in other words $s \in S$ [28].

$$\pi' = argmax_{a \in A(s)} Q(s,a) \tag{3.21}$$

Policy Improvement Algorithm

Algorithm 2 Policy Improvement [28]

Input: MDP, value function V
Output: policy $\pi^{'}$
for $s \in S$ **do**
    for $a \in A(s)$ **do**
        Q(s,a) $\leftarrow \sum_{s' \in S, r \in R} P(s^{'}, r|s, a)(r + (s^{'}))$
    end for$\pi^{'} \leftarrow argmax_{a \in A(s)} Q(s, a)$
end for
return π

3.6.1.3 Policy Iteration

Policy iterations proceeds as a sequence of policy evaluation and improvement steps, and is guaranteed to converge to the optimal policy after infinite interactions with the environment (for an arbitrary finite MDP) [28].
Policy Iteration Algorithm

Algorithm 3 Policy Iteration [28]

Input: MDP, small positive number θ
Output: policy $\pi \approx \pi^*$
Initialize π arbitrarly (e.g, $\pi(a|s) = \frac{1}{A(s)}$ for all $\in S$ and $a \in A(s)$)
policy-stable $\leftarrow$ false
repeat
    $V \leftarrow$ **Policy Evaluation**(MDP, π, θ)
    $\pi^{'} \leftarrow$ **Policy Improvement**(MDP, V)
    if $\pi = \pi^{'}$ **then**
        policy-stable $\leftarrow$ true
    end if
    $\pi \leftarrow \pi^{'}$
until `policy-stable = true`
return π

3.6.1.4 Value iteration

Takes as input policy π to estimate the state-value-function V_π and returns an improved policy, in this approach every sweep over the state space simultaneously performs policy evaluation and policy improvement [28].

Value iteration Algorithm

Algorithm 4 Value Iteration [28]

Input: MDP, small Positive number θ
Output: policy $\pi \approx \pi^*$
Initialize V arbitrarily (e.g V(s) = 0 for all $s \in S^+$)
repeat
    $\triangle \leftarrow 0$
    for $s \in S$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow max_{a \in A(s)} \sum_{s' \in S, r \in R} P(s', r|s, a)(r + \gamma V(s'))$
        $\triangle \leftarrow max(\triangle, |v - V(s)|)$
    end for
until $\triangle < \theta$
$\pi \leftarrow$ **Policy Improvement(MDP,V)**
return π

3.6.2 Model-Free Reinforcement

Tabular Methods In the previous section we discussed the methods of obtaining the optimal policies for an MDPs with assumption the model its well-known - states transition probabilities and the reward function R(s,a)- Most of the time the model is unknown and the agent must interact with the environment directly to obtain the information, model free allows learning a policy - controller - without learning the model.

Interactions $\rightarrow$ rewards & value function $\rightarrow$ policy

3.6.2.1 Monte-Carlo Methods

Monte Carlo [29](**MC**) works by averaging the returns the agent receives from the environment on an episodic basis.
Note: Reinforcement learning tasks can either be episodic with well defined ending points - like end of the game, crash of car - or continuous ones with no ending points like modeling the economy of some country. In episodic task every state-action pair is called **visit**.

Monte-Carlo Prediction First Visit for Action Values:

Algorithm 5 Monte-Carlo Prediction First Visit for Action Values [28]

Input: policy π, positive integer num-episodes
Output: value function Q($\approx q_\pi$ if num-episodes is large enough)
Initialize returns-sum(s,a) = 0 for all $s \in S$, $a \in A(s)$
Initialize returns-sum(s,a) = 0 for all $s \in S$, $a \in A(s)$
for $i \leftarrow 1$ `to num-episodes` **do**
    Generate an episode $S_0, A_0, R_1, ..., ST$ using π
    for $t \leftarrow 0$ `to` $T-1$ **do**
        if (S_t, A_t) `is first visit (with return` G_t`)` **then**
            $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
            return-sum$(S_t, A_t) \leftarrow$ return-sums$(S_t, A_t) + G_t$
        end if
    end for
end for
Q(s,a) $\leftarrow$ return-sum(s,a)/N(s,a) for all $s \in S, a \in A$
return Q

3.6.2.2 Temporal Difference Learning

Unlike MC methods that wait until the end of the episode to learn, TD [30] learning the agent is not allowed to take a break for evaluation and improvement, TD Learning waits for only one time-step and uses temporal errors to update the value.

TD(0) prediction:

Algorithm 6 TD(0) [28]

Input: policy π, positive integer num-episodes
Output: value function V($\approx v_\pi$ if num-episodes is large enough)
Initialize V arbitrarily (e.g V(s) = 0 for all $s \in S^+$)
for $i \leftarrow 1$ `to num-episodes` **do**
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy π
        Take action A_t and observe R_{t+1}, S_{t+1}
        $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
        $t \leftarrow t + 1$
    until S_t is terminal
end for

3.7 Deep Learning

Deep learning has turned out to be broadly applicable technique. It has been found useful in everything from hedge-fund algorithms to bioinformatics. Deep Learning techniques utilize the same building blocks as simpler neural networks.

3.7.1 Biological Basis

Each nerve cell in the brain is know as neuron. Neurons in the brain form a networks, by connections known as synapses.

3.7.2 Artificial Neural Network

Feed-Forward Neural Networks The signal generally moving in one direction through the networks, followed by back-propagation which determines errors at the end of each input signal through the structure and distribute the error back through the network's weights.

Neurons the smallest unit in neural networks, it holds a vector of weights. At each time-step a vector of input Is passed to the neuron. The neurons apply dot product operation to the input and its weights then gives the result of the dot

product into an activation function, usually the activation function is non-linear transformer – to allow the neural network to generalize to non-linear situations

Layers neurons in feed-forward neural networks are organized in layers, each layer consist of a certain number of neurons, in feed-forward neural networks the signal will pass from one layer into the following. Every neuron in each layer is connected to every neuron in the previous layer. The first layer is known as the input layer. It receives signals from some external source, the last layer is known as the output layer, layers in between are known as hidden layers.

Back-propagation:
Uses Multi-chain rule and gradient descent calculates the partial derivative of the error with respect to all the hidden layers weights to determine how much each neuron was responsible for the incorrect output in the output layer so optimizing the weights of the neural network.

3.8 Deep Reinforcement Learning

We have covered model-free reinforcement learning and Deep learning, now its time to combine both and get deep reinforcement learning. The tasks that reinforcement learning solves are mostly in domains in which the agent observes low-dimensional state spaces, however using function approximation - as deep learning- allows the agent to be successful at learning policies directly from high-dimensional state input.

3.8.1 Deep Q-networks:

Deep Q-networks [31] algorithms outperform human experts in Atari games,

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \gamma * (R_{t+1} + \gamma * max_a Q(s_{t+1}, a) – Q(s_t, a_t)) \tag{3.22}$$

There are **three main techniques** Deepmind used in Deep Q-network.
First Technique instead of passing the state and the action to the q-network we just pass the state and get output from the neural network with length equal to the number of possible actions so the function now can be assigned as $Q_A(s)$.

Second Technique Experience Replay: unless at all episodes the environments were initialized to the same states and locations the agent fail to solve the problem, this problem is known as catastrophic forgetting – the agent just end up memorizing the states and if the game started at random locations each episode then the agent will fail.

Third Technique Target Q-Network:
the action-value function update equation is

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \gamma.(R_{t+1} + \gamma * max_a Q(s_{t+1}, a) – Q(s_t, a_t)) \tag{3.23}$$

If we keep updating the parameters of the $Q-network$ after each subset of replay buffer, this might cause instability [27].

Algorithm 7 Deep Q-Learning Algorithm [27]

Initialize replay memory D with capacity N
Initialize $action-value$ function $\hat{q}$ with random weights w
Initialize $target-value$ weights $w^{-} = w$
for `the episode e = 1 to M` **do**
  Initial input frame x_1
  Prepare initial state: S = $\phi(< x_1 >)$
  for `time step t = 1 to T` **do**
    Sample
    choose action A from State S using policy $\pi = \epsilon - greedy(\hat{q}(S, A, w))$
    Take action A, observe reward R and next input frame x_{t+1}
    Prepare next state: $S^{'} = \phi(< x_{t-2}, x_{t-1}, x_t, x_{t+1} >)$
    Store experience tuple$(S, A, R, S^{'})$ in the replay memory D
    $S = S^{'}$
    Learn
    Obtain random mini-batch of tuples (S_j, a_j, r_j, s_{j+1}) from D
    set target $y_j = r_j + \gamma max_a \hat{q}(s_{j+1}, a, w^{'})$
    update: $\triangle w = \alpha(y_j - \hat{q}(s_j, a_j, w)).\nabla_w q(s_j, a_j, w)$
    Every C steps reset $w^{-} = w$
  end for
end for

3.8.2 Policy Gradients methods "Learning to pick the best policy":

Policy-based methods are better for continuous action spaces than value-based methods, because in discrete space we can easily generate an action vector then pick the action with the maximum value but for continuous case the ***max*** operation turned out into an optimization problem.

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \gamma * (R_{t+1} + \gamma * max_a Q(s_{t+1}, a) – Q(s_t, a_t)) \tag{3.24}$$

The most beneficial actions will have the highest chance of being selected from random sampling, since they are assigned the highest probability.

3.8.2.1 Policy function approximation

Function approximation can extend policy-based methods to cover large and continuous state spaces. To do so, we optimize the policy with parameters θ.

Discrete spaces:

$$a = \pi(s, a, \theta) = [a|s, \theta] \tag{3.25}$$

for example, an approximation function would simply be the linear combination of features.

$$f(s, a, \theta) = x(s, a)^T.\theta \tag{3.26}$$

by applying a soft-max function, we can then turn this into a probability distribution. Only works for a discrete set of actions.

Continuous spaces:

$$f(s, a, \theta) = x(s, a)^T.\theta \tag{3.27}$$

$$a = \pi(s, a, \theta) = N(\mu, \sigma^2) \tag{3.28}$$

we can determine the parameters of our Gaussian by

$$\mu = f(s, a, \theta) \tag{3.29}$$

and fix the variance σ^2

3.8.2.2 Stochastic policy gradient

The policy outputs vector representing probability distribution over actions,in the first episodes the agent can select two different actions from the same state, but since the env

3.8.2.3 Define an objective Function and Reinforcing good actions

The policy network denoted by π parameterized by parameters θ Given the state we are in. We encourage the policy to take actions that maximize the total cumulative reward[32]. We have to make the updates small because we want stochastic action selection to ensure exploration, also we want to be able to weight how much we assign credit to each action in the trajectory.

Our objective function

$$\gamma_t * G_t * \log \pi(a|\theta) \tag{3.30}$$

The surrogate objective function $\log \pi(a|s, \theta)$ has large dynamic range (-∞, θ) this allows the optimizer to operate in larger range, still we have to solve the problem of credit assignment, so we add parameter G_t and γ_t , we want to discount more distant actions more than more recent ones

$$G_t = r_t + r_{t+1}... + rT - 1 + r_T \tag{3.31}$$

$$\gamma_t = \gamma(T–t) \tag{3.32}$$

3.8.3 Actor Critic (Temporal Difference update):

Since MC -Monte Carlo Methods - are not sample efficient people generally use bootstrapping - estimate the value using estimates- instead of updating $V(S)$ by Monte-Carlo Sampling, and get advantage function like

$$\delta \longleftarrow R - V(s) \tag{3.33}$$

The Advantage Function when bootstrapping with temporal difference update

$$\delta \longleftarrow r_t + \gamma.V_w(s_{t+1}) - V_w(s_n) \tag{3.34}$$

Baseline with temporal Difference is called ***Actor Critic***

Algorithm 8 Actor Critic (s_0, π_0)

Initialize π_θ to anything
Initialize Q_w to anything
loop `For each Episode`
    Initialize s_0 and set $n \longleftarrow 0$
    while `s is not terminal (for each time step t)` **do**
        Sample a_t $\pi_\theta(a_t|s_t)$
        Execute a_t an observe s_{t+1}, r_t
        A(s,a)$\longleftarrow r_t + \gamma.V_w(s_{t+1}) - V_w(s_t)$
        Update value network: $w \longleftarrow w + \alpha_w.\gamma^t.A(s,a)\nabla_w V(s_t)$
        Update Policy network: $\theta \longleftarrow \theta + \alpha_\theta.\gamma^t.A(s,a).\nabla_\theta \log \Pi_\theta(a_t|s_t)$
    end while
end loop
Return π_θ

Chapter 4

Methodology

4.1 Proximal Policy Optimization"PPO"

A policy gradient algorithm for Reinforcement Learning, that alternate between optimizing the surrogate objective function using stochastic gradient ascent and sampling data through interaction with the environment [1].

4.1.1 Noise Reduction :

$$\nabla_\theta U(\theta) = g = \sum_\tau P(\tau;\theta)(R_\tau(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t))) \tag{4.1}$$

$$g = \frac{1}{N}\sum_{i=1}^{N} R_i \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{4.2}$$

also there is another Bonus

$$\mu = \frac{1}{N}\sum R_i \tag{4.3}$$

$$R_i \longleftarrow \frac{R_i - \mu}{\sigma} \tag{4.4}$$

$$\sigma = \sqrt{\frac{1}{N}\sum (R_i - \mu)^2} \tag{4.5}$$

4.1.2 Credit Assignment:[1]

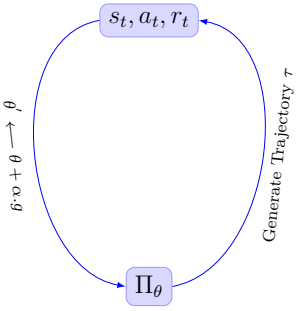
$$\theta \longleftarrow \theta + \alpha_\theta.\gamma^n A(s_n, a_n).\nabla.\log \pi_{theta}(a_n|s_n) \tag{4.6}$$

The Markov property tells us the agent in state s_t only has control on rewards $r_{t+1} + r_{t+2} + ...$ so we change the Return R in the gradient above into R^{future}

$$g = \sum_t R_t^{future} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{4.7}$$

$$g = \sum (\underbrace{\cdots + r_t}_{R_{past}} + \underbrace{r_t + \cdots}_{R_{future}}) \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{4.8}$$

4.1.3 Importance Sampling:



we start with policy π_θ using that policy we generate some trajectories then use those trajectories to update weights of π_θ so we end up with new policy $\pi_\theta^{'}$ and then discard those trajectories, but isn't this sounds like wasteful why do we discard those trajectories after updating the policy?! Recycle Those trajectories and use them to update new policy, mathematically this is equivalent to adding up all the $f(\tau)$, weighted by the probability of sampling each trajectory under new policy [1].

$$\sum P(\tau;\theta^{'}) f(\tau) \tag{4.9}$$

$$P(\tau|\theta) \longrightarrow s_t, a_t, r_t, s_{t+1}, a_{t+1}, \cdots underpolicy \pi_\theta \tag{4.10}$$

now we can modify this equation by multiplying and dividing with $\pi(\tau|\theta)$

$$\sum_\tau \underbrace{P(\tau;\theta)}_{SampleUnderoldPolicy} \cdot \underbrace{\frac{P(\tau;\theta^{'})}{P(\tau|\theta)}}_{Re-weightFactor} .f(\tau) \tag{4.11}$$

$$U(\theta^{'}) = \sum_{\tau} P(\tau;\theta^{'}) f(\tau) \tag{4.12}$$

This tells we can use of old trajectories for computing averages for new policy, as long as this extra re-weight factor, which takes into account how under or over-representative each trajectory is under the new policy compared to the old one [1].

4.1.4 The re-weight Factor

Let's take a closer look at the re-weight factor[1]

$$\frac{P(\tau;\theta^{'})}{P(\tau;\theta)} = \frac{\pi_{\theta'}(a_1|s_1).\pi_{\theta'}(a_2|s_2)\pi_{\theta'}(a_3|s_3)\cdots}{\pi_{\theta'}(a_1|s_1).\pi_{\theta'}(a_2|s_2).\pi_{\theta'}(a_3|s_3)\cdots} \tag{4.13}$$

Because each Trajectory contains many steps, the probability contains a chain of products of each at different time-steps, in practice we want to make sure the re-weight factor is not too far from 1 when we utilize importance sampling [1].

4.1.5 The Surrogate Objective Function

we are trying to update our current Policy $\pi_{\theta'}$. To do that we need to estimate gradient g. But we only have trajectories generated by an older policy π_{θ}, to compute the gradient we utilize importance sampling and the gradient would be the same old gradient formula times re-weight factor [1].

$$g = \frac{P(\tau;\theta^{'})}{P(\tau;\theta)}. \sum_{t} \frac{\nabla_{\theta}\pi_{\theta'}}{\pi_{\theta'}(a_t|s_t)}.R_t^{future} \tag{4.14}$$

4.1.6 Clip Surrogate Objective Function

The clipped surrogate objective function is designed to improve training stability by limit the change to each policy at each step

Algorithm 9 PPO [1]

for *iterationn = 1,2,...* **do**
  for *actor = 1,2,...,N* **do**
    Run policy $pi(\theta)_{old}$ in environment for T time-steps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  end for
  Optimize surrogate L wrt θ with K epochs and mini-batch size $M \leq NT$
  $\theta_{old} \longleftarrow \theta$
end for

4.2 Deep Deterministic Policy Gradient

4.2.1 Introduction

Deterministic policy gradient - DPG - gradient is an off-policy Actor critic method. previous actor critic methods uses *stochastic policy* returns the probability distribution over actions - in discrete action space - or parameters of normal distribution - mean and standard deviation- in case of continuous action space. DPG has deterministic policy in other words it directly tell the agent what action to take from the state [33].
In continuous action space each action is a number, and since DPG uses deterministic policy we will always get the same action for the specific input state[33].

4.2.2 DDPG

DDPG belongs to Adavantage actor critic family, but it uses Deterministic policy and its the policy network objective function differ from A2C

A2C policy network objective function

$$\nabla J(\theta) = \frac{1}{N}(\sum_{1}^{N} A(s,a)\nabla log\pi(a|s)) \tag{4.15}$$

DDPG policy network objective function

$$\nabla J(\theta) = \frac{1}{N}(\sum_{1}^{N} \nabla_a Q(s,a)\nabla \mu_\theta(s)) \tag{4.16}$$

DDPG uses four networks, $Q - network$, deterministic policy network, target $Q - network$ target policy network, the actor directly maps states to actions instead of outputting probability distribution across a discrete action space. Target networks are $time - delayed$ copies of the original networks, using targets network greatly improve the stability in learning.

4.2.3 DDPG learning:

Algorithm 10 DDPG algorithm [2]

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s, \theta^\mu)$ with weights θ^Q and θ^Q
Initialize target network $Q^{'}$ and $\mu^{'}$ with weights $\theta^{Q^{'}} \longleftarrow \theta^Q$, $\theta^{\mu^{'}} \longleftarrow \theta^\mu$
Initialize replay Replay buffer R
for `episode = 1 , M do:` **do**
    Initialize a random process η for action exploration
    Receive initial observation state s_1
    for `t = 1, T do` **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \eta_t$ according to the current policy and exploration noise
        Execute action a_t and observation reward r_t and observe new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random mini-batch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
        Set $y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu'})|\theta^{Q^{'}})$
        Update critic by minimizing the loss: L = $\frac{1}{N}\sum_i(y_i - Q(s_i, a_i|\theta^Q))))^2$
        Update the actor policy using the sampled policy gradient:

$$\nabla_{theta\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta\mu}\mu(s|\theta^\mu)|s_i$$

        Update the target networks:

$$\theta^{Q^{'}} \longleftarrow \tau\theta^Q + (1-\tau)\theta^{Q^{'}}$$
$$\theta^{\mu^{'}} \longleftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu^{'}}$$

    end for
end for

The value network is updated similarity as is done in Q-Learning. The updated Q value is obtained by bellman equation:

$$y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu^{'}})|\theta^{Q^{'}}) \tag{4.17}$$

$$Loss = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{4.18}$$

4.2.4 Actor(policy) and Critic(Value) updates:

The Critic network is updated as in case of Q-learning.The updated Q value is obtained by bellman equation [2]:

$$y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu^{'}}(|\theta^{Q^{'}}))) \tag{4.19}$$

$$Loss = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{4.20}$$

$$\nabla_{\theta\mu} J(\theta) \approx \frac{1}{N}\sum_{i}[\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i} \nabla_{\theta\mu}\mu(s|\theta^{\mu}|_{s=s_i}] \tag{4.21}$$

4.2.5 Target Network Updates [2]

$$\theta^{Q^{'}} \longleftarrow \tau\theta^Q + (1-\tau)\theta^{Q^{'}} \tag{4.22}$$

$$\theta^{\mu^{'}} \longleftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu^{'}} \tag{4.23}$$

where

$$\tau << 1 \tag{4.24}$$

4.2.6 Exploration [2]

For continuous action space exploration is done via adding noise to the action [2].

$$\mu^{'}(s_t) = \mu(s_t|\theta_t^{\mu}) + \eta \tag{4.25}$$

4.2.7 Hindsight Experience Replay "HER"

Dealing with sparse rewards is one of the biggest challenges in RL, our ability as humans to learn almost as much from achieving undesired goal outcome as from

desired one is unlike any generation of model-Free RL algorithms.
The idea is very simple we want to use every single experience we have as much as we can, so if we did not reach the $desired - goal$ instead of telling the agent you failed to achieve the $desired - goal$ we tell the agent you have managed to reach the $achieved - goal$ and then change the weights of the neural network as if the original goal was the achieved goal [19].

Algorithm 11 `Hindsight experience Replay` [19]

Given:
 * an $off - policy$ RL algorithm A,
 * a strategy S for sampling goals for replay
 * reward function $r : SxAxG \longrightarrow R$
Initialize A "neural network"
Initialize replay buffer R
for `episode = 1, M do` **do**
    Sample an action a using the behavioral policy from A:
    $a_t \longleftarrow \pi_b(s_t||a_t)$
    Execute the action a_t and observe a new state s_{t+1}
end for
for `t = 0, T-1 do` **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition in the standard experience replay$(s_t||g, a_t, r_t, s_{t+1}||g)$ in R
    sample set of additional goals for replay $G = S(currentepisode)$
    for $g^{'}G$ **do**
        $r^{'} := r(s_t, a_t, g^{'})$
        Store the transition $(s_t||g^{'}, a_t, r^{'}, s_{t+1}||g^{'})$ in R
    end for
end for
for `t = 1, N do` **do**
    Sample a $mini - batch\ B$ from the replay buffer R
    Perform one step of optimization using **A** and $mini - batch$ **B**
end for

4.3 Simulation Environment

We are using Mujoco physics engine simulator, Mujoco is model-based physics engine standing for Multi-Joint dynamics with contact., Mujoco uses XML and UDRF files to represents the models. Mujoco codes sare written using C / C++. Mujoco is a product, and for students it provides one year license for free. We

chose Mujoco since it provides stable and fast simulation environment compared to the other third-party simulators also there is wrapper "$mujoco-py$" which allows us to run mujoco as one of OpenAI Gym this wrapper allows us to write our code using Python and its deep learning frameworks such as Pytorch or TensorFlow. OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms, its compatiable with any numerical computation library and make no assumption about your agent. OpenAI Gym categories:

1-Atari 2- Box2d 3-Classic Control 4-Mujoco 5-Robotics

There are other simulation environments available such as Robot operating System "ROS", Gazebo Simulator, Unity3D and VREP. ROS and Gazebo are full of bugs, we have suffered many installation problems for gazebo and ROS in our machines, and we are only able to use Gazebo-ROS In Virtual machine which was too slow, and reinforcement learning depends on sampling high frequency per frames which we can't Achieve using virtual Machine. V-Rep has large range of features for its environments which support different programming langauges such as C/C++, Matlab,Octave ,Python, Java but V-REP simulator is still developing and many bugs has been reported while using.

4.3.1 Fetch environments

The fetch reach environments are based on the robotic arm with 2 fingered parallel gripper and consist of 7 degrees of freedom, the goal is 3 dimensional and desired positions of the object. The experiment tested in both sparse and dense rewards. Actions are 4-dimensional: 3 dimensions specify the desired gripper movement in Cartesian coordinates and the last dimension controls opening and closing of the gripper. Observations include the Cartesian position of the gripper, its linear velocity as well as the position and linear velocity of the robot's gripper. If an object is present, we also include the object's Cartesian position and rotation using Euler angles, it's linear and angular velocities, as well as its position and linear velocities relative to the gripper.

In our experiment we try out four different tasks

- **Reaching (FetchReach)** The task is to move the gripper to a target position. This task is used for benchmarking to ensure that a new idea works at all.
- **Pushing (FetchPush)** A box is placed on a table in front of the robot and the task is to move it to a target location on the table. The robot fingers are locked to prevent grasping. The mixture of pushing and rolling defines suitable learned behavior.
- **Sliding (FetchSlide)** A puck is placed on a long slippery table and the target position is outside of the robot's reach so that it has to hit the puck with such a force that it slides and then stops at the target location due to friction.
- **Pick and Place (FetchPickAndPlace)** The task is to grasp a box and move it to the target location which may be located on the table surface or in the air it.

4.3.2 Hand environments:

These environments are based on the Shadow Dexterous Hand, which is an anthropomorphic robotic hand with 24 degrees of freedom. Of those 24 joints, 20 can be controlled independently whereas the remaining ones are coupled joints.

In all hand tasks, two types of rewards are used sparse and dense. Actions are 20-dimensional: We use the absolute position control for all non-coupled joints of the hand. Observations include the 24 positions and velocities of the robot's joints. In the case of an object that is being manipulated, we also include its Cartesian position and rotation represented by a quaternion (hence 7-dimensional) as well as its linear and angular velocities. In the reaching task, we include the Cartesian position of all 5 fingertips.

In our experiment we try out 2 different tasks:

- **Block manipulation (HandManipulateBlock)** In the block manipulation task, a block is placed on the palm of the hand. The task is to then manipulate the block such that a target pose is achieved. The goal is 7-dimensional and includes the target position (in Cartesian coordinates) and target rotation (in quaternions).

- **Egg manipulation (HandManipulateEgg)** The objective here is similar to the block task but instead of a block, an egg-shaped object is used. We find that the object geometry makes significant differences in how hard the problem is and the egg is probably the easiest object. The goal is again 7-dimensional and includes the target position (in Cartesian coordinates) and target rotation (in quaternions).

Chapter 5

Results and Discussion

This research consist of two-part experiments, First, Benchmarking three model-free reinforcement learning algorithms in fetch environments to enable the gripper to run as fast as police. The Second one is benchmarking them in a hand environment.

5.1 Benchmarking Parameters of RL algorithms In OpenAI Environments

There are numbers of model-free RL algorithms, in our experiments, we chose three policy gradient algorithm that represents the major state of the art algorithms in model-free RL algorithms DDPG, DDPG with HER, and PPO, we run each experiment for 350 epoch each epoch contain 400 episodes and each episode maximum number of time-steps is 50 in fetch environment and 100 hand environment.

To measure how each experiment improves with time we use three measurement types :

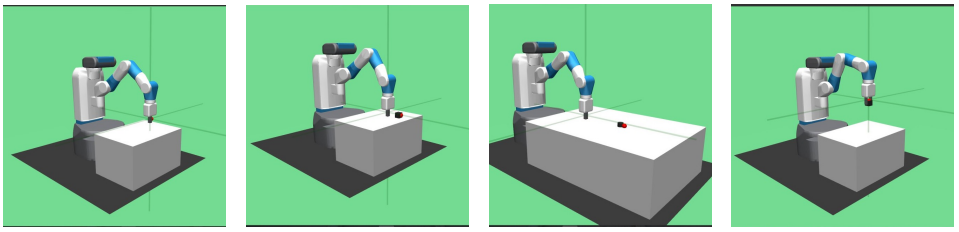
- **Immediate Reward** which is the final reward at the end of the episode.
- **The Epoch Reward** which is the final reward at the end of the epoch, the main issue is it have high degree of variation.
- **The Mean Reward** Measuring the mean of 15 epoch to ease for readability of the graph.

Hyperparameters \RL Algorithms	**DDPG**	**DDPH+HER**	**PPO**
Hidden Layers	3	3	3
Hidden Layers Activation	ReLU	ReLU	ReLU
Learning Rate	0.00101	0.00101	0.00101
Output Layer Neurons (Fetch)	4	4	4
Output Layer Neurons (Hand)	20	20	20
Number of Epoch	350	350	350
Number of Episodes Per Epoch	400	400	400
Test Roll-outs	10	10	10

TABLE 5.1: Hyper-Parameters in all experiments

5.2 Benchmarking the algorithms in Fetch environment

For fetch environment we explored the ability of a 2-fingered gripper to learn by using each algorithm in order to learn how to perform four different task as stated in section 4.3 Reaching, Pushing, Sliding, Pick and Place, as appeared in the same order in fig 5.1.each algorithm runs for 350 epochs with 400 episode each and each episode has a maximum of 50 time-step

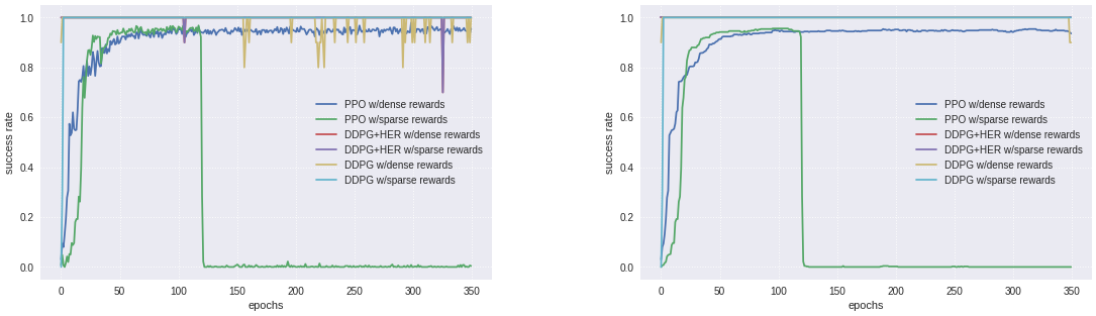


(a) (b) (c) (d)

FIGURE 5.1: The four tasks employed in the experiment, (a)Fetch Reach, (b)Fetch Push, (c)Fetch Slide, (d)Fetch Pick and Place

The results obtained from these experiments expressed in figures below (see figures 5.2,5.3,5.4,5.5) per-epoch vs success rate and vs mean success rate:

1. **Fetch Reach:**

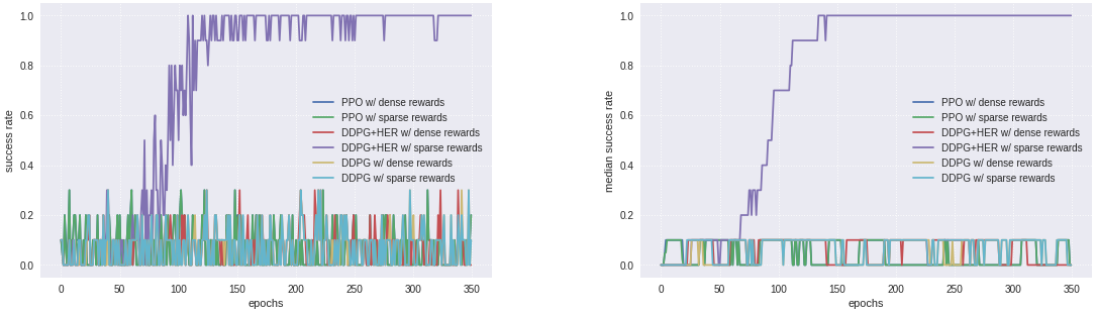


(a) (b)

FIGURE 5.2: (a) Number of Epoch vs Success Rate, and (b) Number of Epochs vs Median Success Rate

We observe from fig 5.2 that all three experiment achieved almost 100% success rate, though PPO drop to zero due to over training.

2. **Fetch Push:**

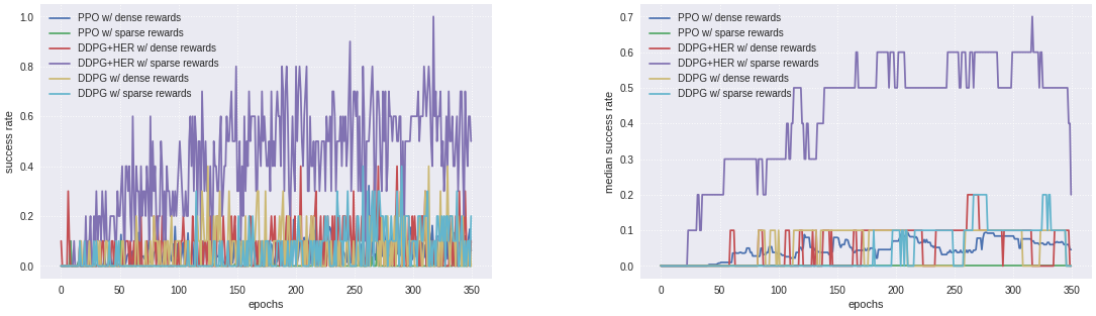


(a) (b)

FIGURE 5.3: (a) Number of Epoch vs Success Rate, and (b) Number of Epochs vs Median Success Rate

From fig5.3 the only algorithm that reached 100% success rate is DDPG with HER when used in sparse reward function, while other algorithm fails to reach even 10% success rate.

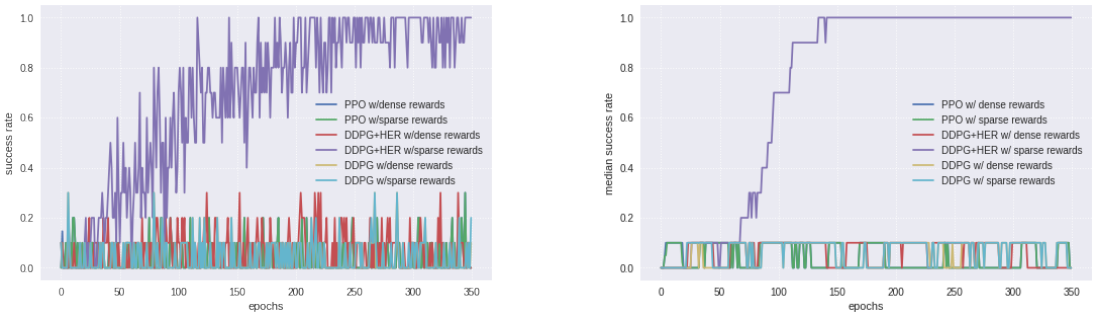
3. **Fetch Slide:**



(a) (b)

FIGURE 5.4: (a) Number of Epoch vs Success Rate, and (b) Number of Epochs vs Median Success Rate

4. **Fetch Pick and Place:**



(a) (b)

FIGURE 5.5: (a) Number of Epoch vs Success Rate, and (b) Number of Epochs vs Median Success Rate

We conclude from figures (5.4,5.5) that in both Slide and Pick and Place the only algorithm that was able to learn is DDPG with HER in sparse reward function.

From the data tables (5.2, 5.3) can conclude that DDPG with HER always yield to maximum success rate much faster compared to DDPG and PPO, the result obtained from PPO and DDPG doesn't converge near 100% success rate in every experiment except for fetch reach which it's the simplest experiment where the only task is to reach for the object in the desired location without the need for manipulating it.

	Fetch		Push	
Algorihm	max successs rate	max epoch	max successs rate	max epoch
PPO w/sparse	0.968	104	0.3	7
PPO w/dense	0.98	298	0.3	7
HER w/sparse	1	0	1	108
HER w/dense	1	0	0.3	124
DDPG w/sparse	1	2	0.3	40
DDPG w/dense	1	1	0.3	40

TABLE 5.2: Comparison between RL algorithms in Fetch reach and Fetch Push environments

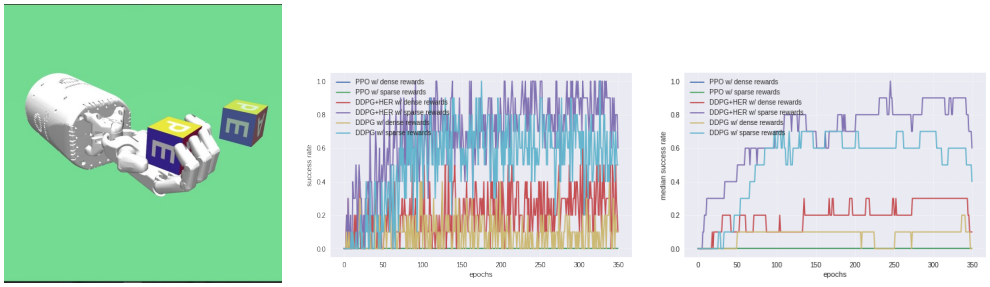
	Slide		Pick and Place	
Algorihm	max successs rate	max epoch	max successs rate	max epoch
PPO w/sparse	0.1	8	0.3	344
PPO w/dense	0.322	262	0.3	344
HER w/sparse	1	317	1	116
HER w/dense	0.4	204	0.3	124
DDPG w/sparse	0.4	275	0.3	6
DDPG w/dense	0.4	125	0.3	6

TABLE 5.3: Comparison between RL algorithms in Fetch Slide and Fetch Pick and Place environments

5.3 Benchmarking the algorithms in Hand environmnets

In hand environment we test the ability of Shadow Dexterous Hand to learn how to manipulate objects using our three algorithm, the tasks we choose for bench-marking are block manipulation and egg manipulation as discussed earlier in section 4.3. Every task runs for 350 epochs each contain 400 episodes and the maximum number of iteration in each episode is 100 time-step.

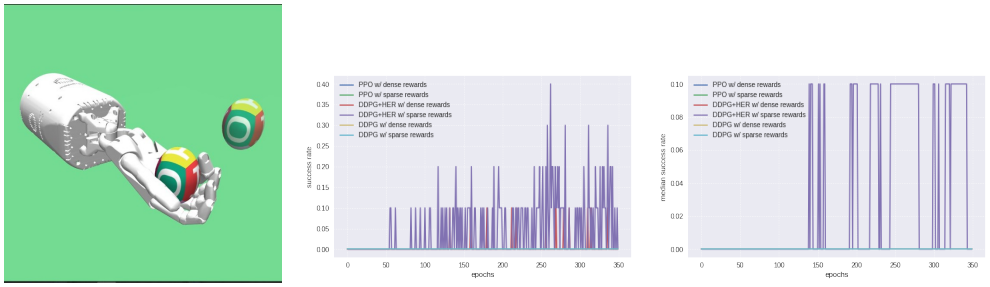
1. **Block manipulation:**



(a) (b) (c)

FIGURE 5.6: This figure represent:(a)The block manipulation task, (b)The epochs versus the success rate of each algorithm, (c)The epochs versus the median success rate.

2. **Egg manipulation:**



(a) (b) (c)

FIGURE 5.7: This figure represent:(a)The egg manipulation task, (b)The epochs versus the success rate of each algorithm, (c)The epochs versus the median success rate.

We conclude form figure(see fig 5.6 and fig 5.7) and the table 5.4 that only DDPG with HER was able to learn in hand environment.

	Block Manipulation		Egg Manipulation	
Algorihm	max successs rate	max epoch	max successs rate	max epoch
PPO w/sparse	0.0	0	0	0
PPO w/dense	0.0	0	0	0
HER w/sparse	1	91	0.4	262
HER w/dense	0.6	133	0.1	132
DDPG w/sparse	1	175	0	0
DDPG w/dense	0.4	97	0	0

TABLE 5.4: Comparison between RL algorithm in Block Manipulate and Egg Manipulate

5.4 Discussion

- The Fetch and Hand environments used in *Open AI Gym* with *MuJoCo* physics simulator are high dimensional state space with continuous action space and state space.
- The MuJoCo simulator is very realistic and very much similar to human hand motion, thus it's very slow compare to other environments in Open AI Gym and it mostly utilizes only the CPU which in turn requires high processing power.
- Value-based algorithms take a very long time to converge in these problems, thus it's better to use policy gradient algorithms.
- Naive policy gradient algorithms such as Reinforce don't work well in this complex environment.
- From result obtained it turns out that both DDPG and PPO don't converge to optimal success rate in all environment except for Fetch Reach task.
- Using DDPG with HER have very high sample efficiency compared to DDPG and PPO has the poorest sample efficiency because it uses online learning. HER utilizes the multi goal feature in these environment.
- DDPG with HER always converges to optimal policy except for Egg Maipulation task.
- DDPG and DDPG with HER works better in *Sparse* reward setting while PPO converges better in *Dense* reward.
- In Fetch environmnet HER converges mostly before 150 epoch.
- In Egg Manipulation task the object geometry make significant difference in the complexity of the problem and makes it very difficult for the algorithm to converge.

Chapter 6

Conclusion, Research Limitations, Recommendation and Future Work

6.1 Conclusion

In this project we applied Deep Reinforcement Learning algorithms to Mujoco's dexterous hand environments. We benchmarked three different Deep Reinforcement Learning algorithms(DDPG, DDPG+HER, PPO) in Mujoco and OpenAI Gym simulation environment, to know which algorithms achieves the maximum reward, we found DDPG+HER to be the best.

When using Reward engineering - shaping the reward from sparse to dense - to test each algorithm in both sparse and dense reward settings, we conclude that changing reward to dense results in compelling the critic to approximate highly non-linear function which may include euclidean distance between the achieved-goal and desire-goal, on the other hand learning in sparse is simpler since the critic differentiate only between success and failure.

6.2 Research Limitation

When applying all algorithms in MuJoCo environment, there was some limitation we faced on our project which were that:

1. Each experiment runs one time, then repeat all the experiments number of times and take the average.
2. The same hyper-parameters were used for all algorithms to achieve benchmarking between algorithms.
3. Only three Reinforcement Learning algorithms were used, two of which belongs to Actor-Critic algorithms, and only one Policy-Gradient method which was sample inefficient.
4. Multi-Goal which means the same algorithm and hyperparameter were used for all environments.
5. The reward function is sparse and the state space is high dimensional.

6.3 Recommendation

6.3.1 Reduce Training Time

To reduce the training time there is multiple ways which significantly improve the speed by which training is accomplished:

1. Increase the computational power by increasing the number of processors and/or cores, this will results in increase of training cost.
2. Stop the rendering in Mujoco.
3. Parallelism which performs many tasks at the same time but requires high degree of synchronization and communication between different tasks and processors or cores.
4. Using Hindsight Experience Replay ensure more sampling efficiency hence reducing the training time.

6.3.2 Reward Engineering

Another way to achieve better reward in MuJoCo dexterous task is by changing how to deal with reward in order to achieve better results which can be accomplished by:

1. Increase the number of episodes, which will increase training time.
2. Select the best hyper-parameters for each algorithms taking into consideration Multi-Goal, but selecting hyper-parameters is trail and error process.
3. Add Layer normalization, Layer normalization stabilizes the learning process in wide range of reward scaling.
4. Use Hindsight Experience replay so allowing more sampling efficient and substitute goal ensures more reward.
5. Select algorithms which will meet the requirements of the target environment.

6.4 Future Work

1. Increase the number of episodes.
2. Run more experiments to select the best hyper-parameters for each algorithm and environment.
3. Bayesian Optimization.
4. Experiment with other Deep Reinforcement Learning algorithms. Transfer it to real-life robot.
5. Evolution Strategy.

Appendix A

Symbols

$S_t \Longrightarrow$ state at time t

$A_t \Longrightarrow$ action at time t

$R_t \Longrightarrow$ reward at time t

$\gamma \Longrightarrow$ discount rate (where $0 \leq \gamma \leq 1$)

$G_t \Longrightarrow$ discount return at time t($\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$)

$S \Longrightarrow$ set of all $non-terminal$ states

$S^+ \Longrightarrow$ set of all states(including terminal States)

$A \Longrightarrow$ set of all actions

$A(s) \Longrightarrow$ set of all actions available in state s

$R \Longrightarrow$ set of all rewards

$P(s^{'}, r|s, a) \Longrightarrow$ probability of next state $S^{'}$ and reward r,
given current state s and current action a

$\pi \Longrightarrow$ policy if deterministic : $\pi \in A(s)$ for all $s \in S$
if stochastic: $\pi(a|s) = P(A_t = a|S_t = s)$ for all $s \in S$ and $a \in A$

$v_\pi \Longrightarrow state-value$ function for policy $\pi(v_\pi = E[G_t|S_t = s]$
for all $s \in S)$

$q_\pi \Longrightarrow action-value$ function for policy $\pi(q_\pi = E[G_t|S_t = s, A_t = a]$
for all $s \in S$ and $a \in A(s))$

$v^* \Longrightarrow$ optimal $value-function(v^*(s) = max_\pi v_\pi(s)$ for all $s \in S)$

$q^* \Longrightarrow$ optimal $action-value$ function $q_\pi(s, a) = max_\pi q_\pi(s, a)$ for $s \in S$ and $a \in A(s)$

Bibliography

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[2] 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.

[3] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[4] Gautham Vasan and Patrick M Pilarski. Learning from demonstration: Teaching a myoelectric prosthesis with an intact limb via reinforcement learning. In *2017 International Conference on Rehabilitation Robotics (ICORR)*, pages 1457–1464. IEEE, 2017.

[5] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[6] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[7] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. Learning to solve the credit assignment problem. *arXiv preprint arXiv:1906.00889*, 2019.

[8] Peter Henderson, Wei-Di Chang, Florian Shkurti, Johanna Hansen, David Meger, and Gregory Dudek. Benchmark environments for multitask learning in continuous domains. *arXiv preprint arXiv:1708.04352*, 2017.

[9] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[10] Hamid Benbrahim and Judy A Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3-4):283–302, 1997.

[11] Jun Morimoto, Gordon Cheng, Christopher G Atkeson, and Garth Zeglin. A simple reinforcement learning algorithm for biped walking. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 3030–3035. IEEE, 2004.

[12] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2619–2624. IEEE, 2004.

[13] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Transactions on Graphics (TOG)*, 37(6):1–14, 2018.

[14] Allison M Okamura, Niels Smaby, and Mark R Cutkosky. An overview of dexterous manipulation. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 255–262. IEEE, 2000.

[15] Haifeng Han, Gavin Paul, and Takamitsu Matsubara. Model-based reinforcement learning approach for deformable linear object manipulation. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 750–755. IEEE, 2017.

[16] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4019–4026. IEEE, 2016.

[17] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[18] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[19] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017.

[20] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[21] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Survey: Robot programming by demonstration. *Handbook of robotics*, 59 (BOOK_CHAP), 2008.

[22] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[23] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[24] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[25] Patrick M Pilarski, Michael R Dawson, Thomas Degris, Farbod Fahimi, Jason P Carey, and Richard S Sutton. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *2011 IEEE international conference on rehabilitation robotics*, pages 1–7. IEEE, 2011.

[26] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3651–3657. IEEE, 2019.

[27] 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.

[28] Richard S Sutton. Introduction: The challenge of reinforcement learning. pages 1–3, 1992.

[29] Che Wang and Keith Ross. On the convergence of the monte carlo exploring starts algorithm for reinforcement learning. *arXiv preprint arXiv:2002.03585*, 2020.

[30] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[31] 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.

[32] Brandon Brown Alexander Zai. *Deep Reinforcement Learning In Action*. 2019.

[33] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.