Specialization project

# Evaluation of REINFORCE on benchmark tasks for Panda in RLBench

Krishnan Jothi Ramalingam
Matrikelnummer: *******

August 24, 2021

Technische Universität Braunschweig
Institut für Robotik und Prozessinformatik
Mühlenpfordtstraße 23 · 38106 Braunschweig

| | |
|---|---|
| Prüfer: | Prof. Dr. Jochen Steil |
| Betreuer: | Prof. Dr. Jochen Steil |

## Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Specialization project mit dem Titel

„Evaluation of REINFORCE on benchmark tasks for Panda in RLBench“

selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Braunschweig, August 24, 2021

****** ****** ****** ****** ******

Krishnan Jothi Ramalingam

# Abstract

This specialization project aims to develop the policy gradient REINFORCE algorithm in python, deploy it on a simulated Frank Emika Panda with 7 degrees of freedom, and train the Panda robot to perform a benchmark task in the RLBench environment, and evaluate the performance of the trained agent. The agent is blind, i.e., the inputs are only proprioceptive with no vision data. Feedforward neural network is employed to learn the non-linearity involved while training the stochastic policy. To validate the developed algorithm, and to gain some insights into the learning dynamics involved, the benchmark task is simplified and then the algorithm is deployed for training. To understand the benefits of baseline, the agent is trained both with and without baseline. Since the learning rate decay has hardly been studied in reinforcement learning, experiments are carried out to understand how learning rate decay helps the algorithm in avoiding early deterministic behavior. After validating the algorithm, the agent is trained on the actual task in the same simulated environment, with some gained insights to improvise the training. Finally, the training results, the limitations of the algorithm, possible methods to improve the capabilities, and the directions of future research are discussed. The python code is available at `https://github.com/KrishnanJothi/Projects/tree/main/RL_PandaRobot`

**Keywords:** Reinforcement Learning, Robot Learning, Monte Carlo Policy Gradient, REINFORCE, Learning Rate Decay, Proprioceptive Inputs.

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **MDP** | Markov Decision Process |
| **DP** | Dynamic Programming |
| **DRL** | Deep Reinforcement Learning |
| **POMDP** | Partially Observable Markov Decision Process |
| **RNN** | Recurrent Neural Netwrok |
| **CNN** | Convolutional Neural Network |
| **AI** | Artificial Intelligence |
| **ANN** | Artificial Neural Network |
| **MLP** | Multilayer Perceptron |
| **DoF** | Degree of Freedom |
| **HRI** | Human-Robot Interaction |
| **PDF** | Probability Distribution Function |
| **GMM** | Gaussian Mixture Model |

# 1 Introduction

Apart from learning, there are multiple challenges involved in creating robots, that are intelligent enough to interact directly with the environment to achieve their goals. Initially, section 1.1 presents a broad overview of the research areas, which deal with current challenges in robot manipulation. Since the main focus of this research is on learning, section 1.2 explains the challenges, especially in robot learning for manipulation. Even though many machine learning methods exist, this research focuses on the policy-gradient reinforcement learning technique, hence chapter 2 starts with a formal background, and then the derivation for REINFORCE and the baseline function are presented, and finally, the challenges in applying reinforcement learning to robotics are explained. Once after introducing deep reinforcement learning and its challenges, the motivation to choose REINFORCE and proprioceptive inputs, and the related works are described in chapter 3. Chapter 4 describes the implementation details, and finally, the results and discussion, and the conclusion and future works are presented in chapters 5 and 6 respectively.

## 1.1 Robot Manipulation and its Challenges

Recent immense progress in machine learning along with the advancement in hardware development and design led to the encapsulation of uncertainties, and the progress in adaptive and robust control, through which robots have gained new levels of dexterity. But still, achieving dexterous manipulation capabilities in robots remains an open problem. Robotic manipulation is a proxy for human dexterity, and the robotic industry still strives for multipurpose fine manipulation by overcoming the following challenges,

### 1.1.1 Hardware and Design

It is challenging to fit all the required sensors, actuators, and mechanical structures in the anthropomorphic hand such that it satisfies the payload conditions using the limited space. Due to the placement constraints of touch sensors, contact sensing is not possible at the joints [Nic16]. Advances in material science and electronics led to the design of soft actuation, which enables contact sensing on the entire hand. On-going research is to develop cheap and robust, easily integrable robotic hands, which can also function in other fluids such as water and oil without being damaged [BK19].

### 1.1.2 Robot Perception

Robot vision helps to evaluate the forces that have to be applied to manipulate the object, and as a result, it helps to modify the aperture of the hand (anthropomorphic end effector) or any end effector in general. Despite research in the past few years, robots still struggle to recognize objects that are partially occluded [Yul14], particularly when viewed from a moving camera or if the object in the robot hand is still in motion [Shi15]. Research in the field of interactive perception shows that perception and control are strongly interconnected [Jea17].

### 1.1.3 Robot Control

Control algorithms are used to guide the grasp by updating the values of the object's physical properties, which were poorly estimated through vision. The grasping process is not only object dependent, but also robot dependent, i.e., the number of degrees of freedom of the robot is directly proportional to the complexity of the control. Robot manipulation requires control of the arm, torso, and ultimately the entire body [Mas18], which becomes more difficult in the case of humanoids while maintaining their balance [KK18]. To simplify the control and to execute an optimal stable grasp by exploiting postural synergies, robots need cognitive capabilities to predict the reason for executing the grasp [BK19].

### 1.1.4 Robot Learning

To overcome the dynamics in the interaction process, the uncertainties in perception, and to generate grasps even for unknown and non-rigid objects, robots need generalization capabilities. Therefore recent approaches are mostly data-driven [Jea14], which also avoids the computation of optimal grasp whenever the robot encounters an object and provides fast and online grasp generation. Learning helps to embed representations of stable grasps [OBK10] to regenerate grasps at run time after verifying stability [Seu14].

Research is going on in the fields of mechanical design, electronics, perception, control, software, and learning, to let robots behave optimally, engage in collaborative tasks, manipulate objects jointly with humans, and adapt themselves to unexpected human behavior [BK19]. This specialization project focuses more on the learning part for robotic manipulations.

## 1.2 Challenges in Robot Learning for Manipulation

Robot learning is inspired by humans, which enables the robot to learn continuously, adapt their perception, and control unfamiliar objects. Learning can happen either through autonomous self-exploration or through guidance from a human teacher. Since real-world manipulation is costly and may damage the robot, simulated environments are preferred for learning. The following are the challenges in robot learning for manipulation,

#### Lack of Accurate Simulators:

Learning to grasp and manipulate fast-moving and non-rigid objects in a simulated environment requires accurate modeling of contact dynamics. Even existing simulators are unable to accurately model contact dynamics for soft and deformable objects [BK19].

#### Better Modeling Strategies:

Since creating accurate models of the robot, the environment, and manipulating objects in simulation is a challenging task, better modeling strategies have to be developed, especially for objects whose states change markedly after manipulation (e.g., an onion after being chopped, deformable terrain, cluttered environments, fluid-solid interaction) [Gab19].

#### Theory and Planning:

Planning and generation of appropriate intermediate grasping and manipulation tasks require a thorough description of task goals that reflect the significance of theory and planning even for data-driven approaches, which demands better strategies to model soft bodies and generate scenarios that include force and torque information. [BK19].

#### Learning Transition Models:

Transition models capture the change in the object's state in the robot's environment due to robot manipulation. Learning transition models becomes difficult if the environment is highly stochastic, and hence suitable exploration strategies must be adopted, for acquiring samples. [Oli19].

### Skill Transfer:

Skills that are learned by the robot must be transferable because, in a certain family of tasks, there is a change in only a few of the task contexts, while all the other semantics of the task remains the same [Cal18]. Transfer learning enables skill re-use, increasing the efficiency of learning.

### Partial Observability and Non-stationarity:

Even though the agent is trained successfully in a simulated environment, it is not guaranteed that the learned behavior transfers to the real world due to the partial observability and non-stationarity (e.g., In recommendation systems, mental state observations of the users are unpredictable, and on physical systems, the observations of the wear and tear on hardware components are not available) [Gab19].

### Model Refinement in Real-world:

By overcoming the challenges in section 1.1, the quality of the input data required for simulated training improves, and then the learned behavior can be further refined in the real world. [al18a].

Even though learning is important, it has certain limitations, and solely relying on it is not a viable solution for every problem in robotics. [BK19].

# 2 Reinforcement Learning

Reinforcement learning is an area of machine learning, in which an agent interacts with its environment and tries to maximize the accumulated reward over its lifetime without relying on exemplary supervision or complete knowledge of the environment model. It is a computational approach that involves goal-directed learning and decision-making. As a result, a policy is learned to maximize the numerical reward by making optimal decisions. Robotic tasks associated with reinforcement learning are usually either episodic or continuing with infinite horizons.

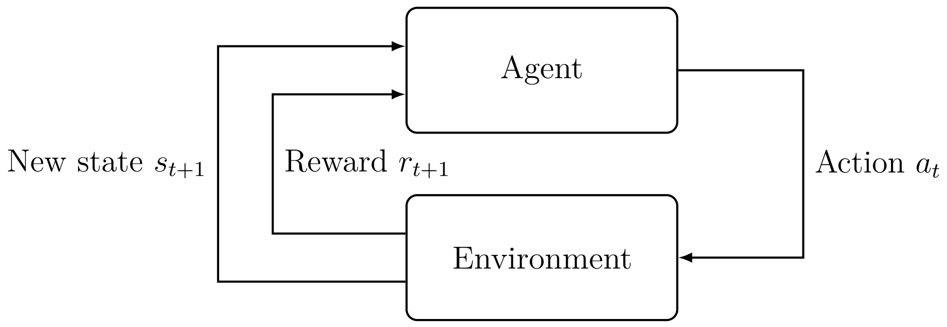


Figure 2.1: Agent-Environment Interaction [Bac13]

Even without learning, it is possible to obtain the optimal behavior by implementing the well mathematically developed Dynamic Programming (DP) concepts like policy and value iteration. But to do that, a complete and accurate model of the environment is required, i.e., the state transitions and environmental outcomes should be known, and hence they are called model-based methods. But in most real-world problems, it is practically impossible to obtain an accurate model of the environment. Once after formally introducing Markov Decision Process (MDP) in section 2.1, learning strategies to overcome the constraints of model-based methods are introduced.

## 2.1 Markov Decision Process

MDP formally describe an environment for reinforcement learning, in which all states are Markov (obey Markov property), i.e., the current state $S_t$ contains sufficient statistics to predict the future, $P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$.

Markov Property - *"The future is independent of the past given the present"*

MDP is a discrete-time stochastic control process used for a variety of optimization problems where the outcome is partially random and partially under the control of the decision-maker.

MDP is characterized by a 5-tuple $\langle S, A, P, R, \gamma \rangle$ [CMa18]

where,
$S$ is a finite set of states,
$A$ is a finite set of actions,
$P$ is a state transition probability matrix, i.e., the probability that action $a_t \in A$ in state $s_t \in S$ at time step $t$ will lead to state $s_{t+1}$ at time step $t+1$,
$R$ is the reward function, i.e., the immediate reward after the transition from state $s_t$ to state $s_{t+1}$,
$\gamma$ is a discount factor representing the difference in importance between the present and the future rewards, $\gamma \in [0, 1]$

The goal of MDP is to find a policy $\pi$ that specifies the action $A_t$ for a given state $S_t$ at the time step $t$, such that it maximizes some cumulative function, typically the expected discounted sum of rewards over a potentially infinite horizon. MDP policies fully define the behavior of an agent, it depend only on the current state (due to Markov property), and are time-independent. The policy which maximizes the function is called an Optimal policy $\pi^*$, and it is also possible that multiple distinct optimal policies can exist for a particular MDP. Reinforcement learning as a MDP assumes both the state and reward is Markovian. As a result of the Markovian assumption, the obtained reward and the transition probability from state $S_t$ to $S_{t+1}$ due to the performed action $a_t$ are stationary, hence

$$P[S_{t+1}|S_t, a_t] = P[S_{t+1}|S_1, a_1..., S_t, a_t], \qquad P[r_{t+1}|S_t, a_t] = P[r_{t+1}|S_1, a_1..., S_t, a_t]$$

Extensions to MDPs also exist. The state or action space can be countably infinite (Infinite MDP) or continuous (Continuous MDP). if the states of the MDP are hidden, then it refers to Partially Observable Markov Decision Process (POMDP), where an additional finite set of observations exists. Undiscounted and average reward MDPs are other extensions of it.

## 2.2 Policy Gradient Methods

Due to the lack of full knowledge of the MDP in many real-world applications, model-free methods like Monte-Carlo and Temporal-Difference learning are used for prediction and control, which are experience sampling approaches. In case of large-scale problems, the state and action space becomes too large (or even continuous), which leads to storage issues in memory, and also learning the value of each state individually becomes too slow. To overcome these issues, value function approximation and policy gradient methods are used, which use function approximators to estimate the value function or directly parameterize the policy. Policy gradient methods learn a parameterized policy $\pi(a|s,\theta)$,

$$\pi(a|s,\theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\} \tag{2.1}$$

which gives the probability of choosing the action $a$ at time $t$ given that the environment is in state $s$, at time $t$, with parameter $\theta$, such that $\theta \in \mathbb{R}^d$, $\pi(a|s,\theta) \in (0,1)$ and $\nabla\pi(a|s,\theta)$ exists [SB18]. Policy gradient reinforcement learning is an optimization problem and the goal is to search for a local maximum in the policy objective function $J(\theta)$ by ascending the gradient of the policy with respect to parameter $\theta$.

### 2.2.1 Stochastic Policy

Stochastic policy shown in the equation 2.1 is represented by a conditional probability distribution. Since the sampled action $a_t$ is a continuous variable, it is more suitable for continuous-control environments. Estimating the policy gradient of a stochastic policy does not require a system model, and a detailed explanation is given in subsection 2.2.2. Without manual hard coding, the stochastic policies can handle the exploration/exploitation trade-off, which is explained in subsection 2.3.4. Due to incomplete perception, multiple indistinguishable states require different optimal actions, which refers to Perceptual Aliasing. Stochastic policies are capable of handling this problem effectively.

### 2.2.2 The Policy Gradient Theorem

The policy gradient theorem gives an exact expression whose expectation approximates the gradient of the objective function, i.e., the expectation of the sample gradient approximates the actual gradient of the objective function with the parameter $\theta$,

$$\nabla J(\theta) = \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a)\nabla\pi(a|S_t,\theta)\right] \tag{2.2}$$

$$q_\pi(S_t,a) = \mathbb{E}_\pi\left[G_t|S_t,A_t\right] \tag{2.3}$$

where $\pi(a|s,\theta)$ is the parameterized policy and $q_\pi(S_t,a)$ is the action-value function, i.e., the expected return from the state $S_t$, by taking the action $A_t$, and then following policy $\pi(a|s,\theta)$. The policy gradient theorem leads us to stochastic gradient-ascent algorithm,

$$\theta_{t+1} = \theta_t + \alpha\sum_a \hat{q}(S_t,a,\mathbf{w})\nabla\pi(a|S_t,\theta_t) \tag{2.4}$$

where $\hat{q}$ is a learned approximation of the action-value function $q_\pi$ with weights $\mathbf{w}$. The equation 2.4 is called as *all-actions* method, since the update of the policy parameter $\theta$ involves all of the actions [SB18]. In the following subsection 2.2.3, we will take a look at a specific algorithm that comes under the family of policy gradient methods, which will be the major focus of this specialization project.

### 2.2.3 REINFORCE

The Monte Carlo policy gradient algorithm is the classical algorithm in the family of policy gradient methods which is commonly referred to as REINFORCE [Wil92]. This algorithm uses the policy gradient theorem, but unlike *all-actions* method, the update at time step $t$ involves only one action $A_t$ actually taken at time step $t$. The following are the steps involved in deriving REINFORCE.

1. In equation 2.2, a weighting term (likelihood ratio) is introduced by multiplying and dividing the summed terms by $\pi(a|S_t, \theta)$ such that the equality does not change.

$$\nabla J(\theta) = \mathbb{E}_\pi \left[ \sum_a \pi(a|S_t, \theta) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \right] \tag{2.5}$$

2. Summation over all the actions is replaced by a sample action $A_t \sim \pi$

$$\nabla J(\theta) = \mathbb{E}_\pi \left[ q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \tag{2.6}$$

3. Since $\mathbb{E}[G_t|S_t, A_t] = q_\pi(S_t, A_t)$, the equation 2.6 is modified as,

$$\nabla J(\theta) = \mathbb{E}_\pi \left[ G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \tag{2.7}$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.8}$$

   where $G_t$ is the return, $R_t$ is the reward obtained in the state $S_t$ as a result of the performed action $A_t$ at the time step $t$, and $\gamma \in [0, 1]$ is the discount factor. $\gamma$ close to 0 leads to myopic evaluation, i.e., immediate rewards are more preferred, and $\gamma$ close to 1 leads to far-sighted evaluation, i.e., future rewards are more preferred.

4. The term inside the brackets in the equation 2.7 is the required expression that can be sampled on each time step whose expectation approximates the gradient. This sample is used to formalize our stochastic gradient ascent REINFORCE update formula,

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{2.9}$$

5. Since $\nabla \ln \pi(A_t|S_t, \theta) = \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$, the compact expression is as follows,

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla \ln \pi(A_t|S_t, \theta_t) \tag{2.10}$$

   $\nabla \ln \pi(A_t|S_t, \theta)$ is referred as eligibility vector or score function. The eligibility vector points in the direction steepest ascent that most increases the probability of repeating the action $A_t$ on future visits to state $S_t$.

To compute the return at each time step $t$, REINFORCE uses the discounted sum of all future rewards till the end of the episode. This algorithm is well defined only for the episodic case. After generating one complete episode, the update of the parameter $\theta$ takes place. Pseudocode for this algorithm is given below [SB18]

**Pseudocode for REINFORCE**

```
Input: A differentiable parameterized policy $\pi(a|s,\theta), \theta \in \mathbb{R}^d$

Initialization:
Algorithm parameters: step size $\alpha > 0$, $\gamma \in [0,1]$
Initialize policy parameter $\theta$ arbitrarily.

for each episode:
  Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T \sim \pi(.|.,\theta)$
    for each step of the generated episode $t = 0, 1, ..., T-1$:
      $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
      $\theta_{t+1} \leftarrow \theta_t + \alpha G_t \nabla \ln \pi(A_t|S_t, \theta_t)$
    end for
end for
```

REINFORCE belongs to the class of algorithms which puts a lower bound on the expected return [Fre13] and this method is guaranteed to converge to the true gradient at the fastest theoretically possible pace [Jan06a]. The main issue with this algorithm is high variance and thus produces slow learning [SB18].

Other advanced deep reinforcement learning algorithms also exist, which can process the visual input directly in an end-to-end fashion, and approximate the policy. Deep reinforcement learning is introduced in the chapter 3, and the motivation to choose REINFORCE algorithm with proprioceptive inputs is discussed in the section 3.2.

### 2.2.4 REINFORCE with Baseline

The baseline $b(S_t)$ can be any function that is not a function of action $a_t$. $b(S_t)$ is a function when introduced in an expectation, it does not introduce bias but affects the variance and the learning speed significantly. Since $b(S_t) = 0$ is plausible, it is a strict generalization of REINFORCE. Due to the subtraction of baseline, the equation 2.7 gets modified as,

$$\nabla J(\theta) = \mathbb{E}_\pi \left[ (G_t - b(S_t)) \nabla \ln \pi(A_t|S_t, \theta) \right] \tag{2.11}$$

$$\nabla J(\theta) = \mathbb{E}_\pi \left[ G_t \nabla \ln \pi(A_t|S_t, \theta) \right] - \mathbb{E}_\pi \left[ b(S_t) \nabla \ln \pi(A_t|S_t, \theta) \right] \quad (\mathit{linearity}) \tag{2.12}$$

$$\mathbb{E}_\pi \left[ b(S_t) \nabla \ln \pi(A_t|S_t, \theta) \right] = b(S_t) \mathbb{E}_\pi \left[ \nabla \ln \pi(A_t|S_t, \theta) \right] \quad (\mathit{linearity}) \tag{2.13}$$

The expectation for the applied log derivative trick (used in the equation 2.10) is computed below,

$$\mathbb{E}_\pi \left[ \nabla \ln \pi(A_t|S_t, \theta) \right] = \nabla .1 = 0 \tag{2.14}$$

$$\mathbb{E}_\pi \left[ (G_t - b(S_t)) \nabla \ln \pi(A_t|S_t, \theta) \right] = \mathbb{E}_\pi \left[ G_t \nabla \ln \pi(A_t|S_t, \theta) \right] \tag{2.15}$$

Equation 2.15 shows that baseline does not modify the expected value and not introduces bias. The following derivation shows the optimal baseline function that minimizes the variance and hence speeds up the learning.

$$Var[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{2.16}$$

where $X$ is a random variable. The variance of the equation 2.11 is evaluated as follows,

$$\begin{aligned} Var[\nabla J(\theta)] = \mathbb{E}_\pi & \left[ ((G_t - b(S_t)) \nabla \ln \pi(A_t|S_t, \theta))^2 \right] \\ & - \mathbb{E}_\pi \left[ (G_t - b(S_t)) \nabla \ln \pi(A_t|S_t, \theta) \right]^2 \end{aligned} \tag{2.17}$$

Equating the derivative to zero, to find the optimal $b(S_t)$ [Lex13]

$$\frac{dVar[\nabla J(\theta)]}{db(S_t)} \Rightarrow 0 \tag{2.18}$$

$$b(S_t) = \frac{\mathbb{E}_\pi \left[ \nabla \ln \pi(A_t|S_t, \theta)^2 G_t \right]}{\mathbb{E}_\pi \left[ \nabla \ln \pi(A_t|S_t, \theta)^2 \right]} \tag{2.19}$$

If $X$ and $Y$ are random variables, and they are independent,

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y], \qquad Cov(X, Y) = 0 \tag{2.20}$$

Assuming independence between the terms in the expectation of the numerator,

$$b(S_t) = \mathbb{E}_\pi[G_t|S_t] \tag{2.21}$$

Equation 2.21 shows that a good choice for the baseline function is an estimate of the state-value, $\hat{v}(S_t, \mathbf{w})$, i.e, the expected return for the given state $S_t$, where $\mathbf{w} \in \mathbb{R}^{d'}$ is the state-value weights. Both the policy parameters $\theta$ and the state-value weights $\mathbf{w}$ are trained using the Monte Carlo method [SB18].

Similar to subsection 2.2.3, the update rule is derived, but with an additional baseline,

$$\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t))\frac{\nabla\pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{2.22}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \beta(G_t - b(S_t))\nabla\hat{v}(S_t, \mathbf{w}) \tag{2.23}$$

where both the learning rates $\alpha$ and $\beta$ are the hyper parameters that has to be tuned. The state-value function $\hat{v}(S_t, \mathbf{w})$ is trained parallelly in a supervised fashion with $G_t$ as the target. Pseudocode for REINFORCE with Baseline algorithm is given below [SB18],

**Pseudocode for REINFORCE with Baseline**

```
**Input:**
A differentiable parameterized policy $\pi(a|s, \theta), \theta \in \mathbb{R}^d$
A differentiable parameterized state-value function $\hat{v}(S_t, \mathbf{w}), \mathbf{w} \in \mathbb{R}^{d'}$

**Initialization:**
Algorithm parameters: step sizes $\alpha > 0$, $\beta > 0$, $\gamma \in [0, 1]$
Initialize policy parameter $\theta$ and state-value weights $\mathbf{w}$ arbitrarily.

**for** each episode:
  Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T \sim \pi(.|., \theta)$
    **for** each step of the generated episode $t = 0, 1, ..., T-1$:
      $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
      $\delta \leftarrow G_t - \hat{v}(S_t, \mathbf{w})$
      $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \beta\delta\nabla\hat{v}(S_t, \mathbf{w})$
      $\theta_{t+1} \leftarrow \theta_t + \alpha\delta\nabla \ln \pi(A_t|S_t, \theta_t)$
    **end for**
**end for**
```

## 2.3 Challenges in Applying Reinforcement Learning to Robotics

Reinforcement learning is robust to changes in a system, such as an actuator malfunctioning, models parameter changes, and external disturbances [El-16]. Difficult or even impossible control problems of supervised learning and traditional DP methods can be solved using the efficient framework of reinforcement learning, but still, it faces the following challenges,

### 2.3.1 Curse of Dimensionality

The exponential explosion of discrete state and action space requires extensive computational power to process the high dimensional state–action space [JP13]. Function approximation approaches are one of the ways of dealing with this issue, and hence REINFORCE algorithm is a suitable candidate that works fine in the continuous control environment.

### 2.3.2 Reward Shaping

Many robotic tasks are multi-objective [Oli19] (e.g., finding the optimal path for robot manipulation and minimizing the energy consumption without collision), hence the difficulty of reward shaping is increased, because the agent must observe the variance in the reward signal to be able to guide and improve a policy while learning. In particular, if the agent receives sparse rewards, it is unlikely to ever succeed in its lifetime [JP13] since it is less intuitive than the intermediate rewards.

### 2.3.3 Curse of Real-world Samples

Even though robots are trained in a simulated environment, fine-tuning the trained model in the real-world setup [Seu14] helps to capture the details in the dynamics and be robust against uncertainties, which reflects the significance of real-world robot manipulation. The wear and tear of the robot, frequently changing environment settings, and the delays in sensing and execution are inherent in physical robotic systems. Therefore sample efficient algorithms are essential, which learn from a small number of trials.

### 2.3.4 Exploration–Exploitation Trade-off

Unlike supervised learning, the reinforcement learning agent must discover the environment in a trial and error fashion to learn the optimal action. The agent needs to explore its surrounding by performing previously unused actions. Well-known actions are safe, since the outcome is predictable, whereas exploration might discover new strategies and lead to an even higher reward. The trade-off between exploration and exploitation is a significant problem in robotics. To maintain a balance between them, suitable exploration strategies must be adopted.

# 3 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a combination of both reinforcement learning and deep learning, which gained a lot of attention in recent years. It involves deep processing of a huge amount of input data (e.g. usually raw vision data) in an end-to-end fashion using dense neural networks, making it feasible to learn better representations than manually handcrafted features.

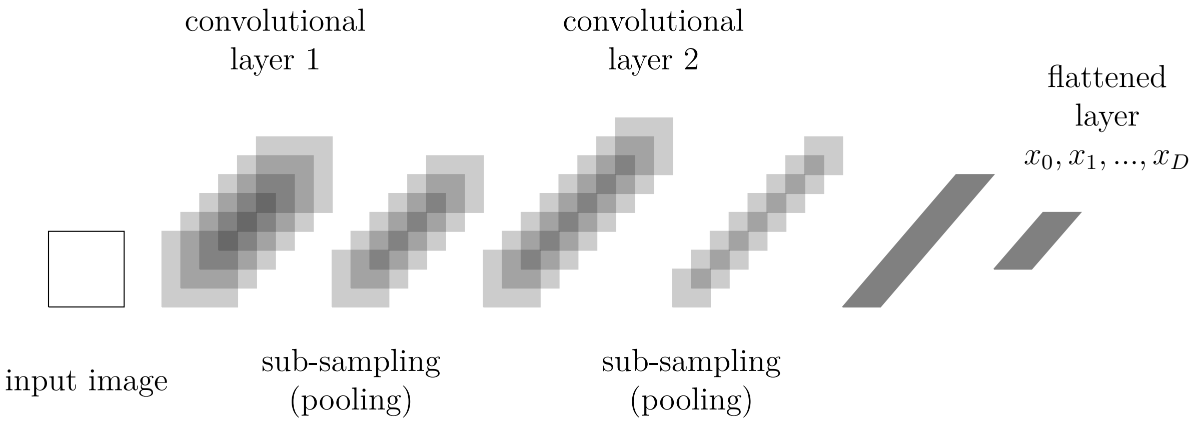


Figure 3.1: Feature extraction in a traditional Convolutional Neural Network (CNN) [Stu20]

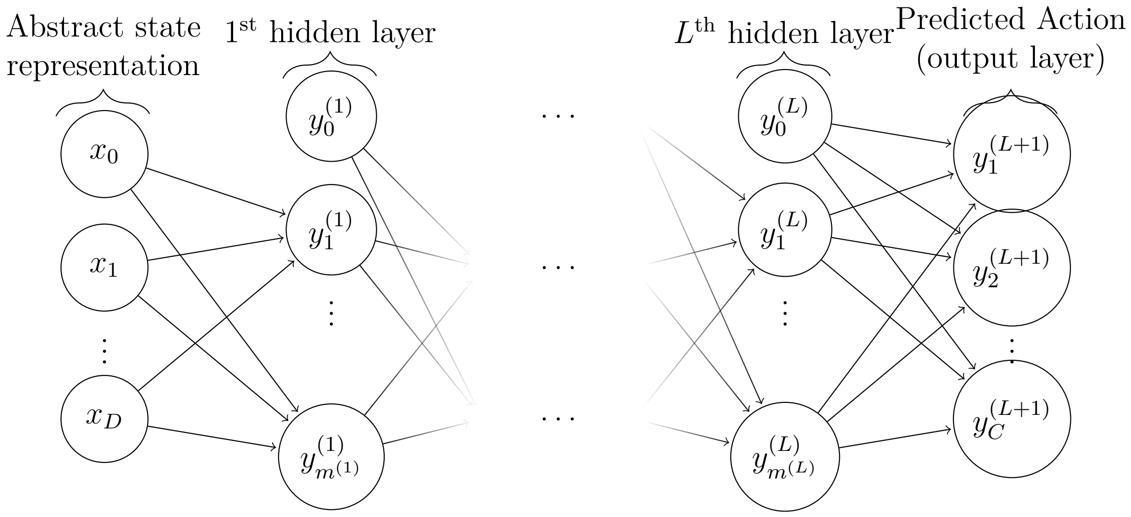


Figure 3.2: Fully connected Artificial Neural Network (ANN) with $L$ hidden layers predicting the action to be performed for the given extracted feature of the visual input [Stu20].

CNN employs a mathematical operation called convolution to extract feature representations from input images using convolution and pooling layers, and finally, it outputs a flattened layer which is the input to a fully connected deep network. The flattened output layer in figure 3.1 is the abstract state representation, which is the input to the deep neural network (e.g., Fully connected neural network depicted in figure 3.2). The output of the network predicts the action to be taken for the given abstract state representation. Apart from the Multilayer Perceptron (MLP) model shown in figure 3.2, DRL techniques also use other advanced ANN architectures like Recurrent Neural Netwrok (RNN) and its variants, and autoencoders. Similarly, depending on the architectural details like the number of convolution and pooling layers, the size of the kernel, and the stride value, different types of CNN exist.

Recently DRL was able to master difficult control policies for seven Atari 2600 games directly from high-dimensional sensory inputs [al13]. The input states are assumed not to be complete and do not contain all necessary information to be Markov, which led to the POMDP assumption. The screenshots, i.e., the frames of the games are the input observations for the agent. To overcome the problem of partial observability, a history of the last 4 frames is stacked and provided as input. RNN is used as the function approximator, which provides the condensed feature representation of the past events.

14 different robotic manipulators were able to grasp unseen objects and discover unconventional grasping strategies once after training over 800,000 grasp attempts. To learn the hand-eye coordination for robotic grasping, a large CNN was trained with visual inputs to make successful grasps, independent of the camera calibration [al16b]. These are a few of the successful research outcomes in the field of DRL.

## 3.1 Challenges in Deep Learning and Deep Reinforcement Learning

Even though DRL methods enables an agent to have a good perception of its environment, learning optimal control policies with raw vision data is one of the major challenges in DRL. Deep neural networks deal well with high-dimensional input issues in reinforcement learning problems by hierarchical feature extraction and learning abstracted representations. Still, DRL is at its initial stage of development since there are many unexplored aspects of reinforcement and deep learning combination, and many challenges exist in real-world applications such as robotics.

#### Lack of Reasoning Capability:

Deep learning models abruptly forget the previously learned behavior and unable to question the provided input information. Without any questioning capability, it believes everything blindly, which is fed during training. Even though it can extract hierarchical features and learn abstracted representations, learning fails if the input vision data is irrelevant to the target task.

### Data-hungry and not Transparent:

Deep learning is data-hungry and is not sufficiently transparent due to the lack of intuitive interpretation of the parameters in complex networks. Proper visualization techniques to analyze these nodal contributions might be a great advantage, especially in domains like financial trades or medical diagnosis [Mar18].

### Prior Knowledge Integration:

The knowledge of the deep learning models is mainly from the correlation of the feature vector, and it is unclear how to integrate the prior knowledge in end-to-end networks (e.g., laws of physics are not explicitly encoded when training a robot to walk). Research is going on, to integrate prior knowledge into an end-to-end deep learning system effectively [Mar18].

### Causality Representation:

Causality is highly significant in some approaches to Artificial Intelligence (AI) since causation takes a step further than correlation and provides more insight into the problem to find better solutions. Deep neural networks can learn the complex correlation between the features, but with no inherent representation of causality (e.g., even though 'rain' and 'wet floor' are correlated, causation shows that rain causes wet floor) [Pea19].

### Dimensionality Reduction and Function Approximation:

Despite recent research in the field of DRL, it is still necessary to understand better by investigating further on deep network architectures for end-to-end learning, which directly approximates non-linear control policies, generates the deep state representation, and significantly reduces the dimension of the input vision data to low dimensional representations, before processing it through deep networks [SH18].

### Transfer Learning:

Transfer learning is one of the challenges mentioned in the section 1.2, which has not yet been addressed in depth in the field of DRL, i.e., using the features learned by the deep end-to-end networks for different tasks without modifying the network architectures [SH18].

**High-end Requirements:**

DRL requires high-performance hardware since it is computationally expensive, and deep learning models are difficult to engineer[Mar18] because it requires subject matter expertise of mathematics, computer science, and the field of application.

## 3.2 Motivation to Choose REINFORCE and Proprioceptive Inputs

Since the real-world tasks are of various difficulty levels, it is feasible to approach simple problems with traditional reinforcement learning algorithms without the need for large-size visual data sets, high computational power, and the above-mentioned challenges can also be omitted. Even though developments happen in the field of robotics, not all day-to-day simple tasks are automated by a robot yet, which reflects the necessity of better robot control policy call for better approaches. When the task becomes highly complicated, deep processing of visual inputs should also be considered as an option since it might learn better representations than handcrafted features.

Recently a 'blind' quadrupedal robot is trained in a simulated environment by privileged learning [al19], which consists of a teacher and student policies. Both are trained in a two-stage training process. The controller successfully trots through mud, moss, snow, and a variety of other off-road terrains and demonstrates remarkable zero-shot generalization from simulation to real-world [al 5]. Even though the training strategy is complex, the task involved is very challenging, and the controller uses only a stream of proprioceptive signals. It reflects the successful learning ability of complex behaviors without any visual inputs to the agent, which motivated me to evaluate the performance of the classical policy gradient algorithm on a 7 Degree of Freedom (DoF) robot in a simulated environment using proprioceptive inputs to perform a benchmark task.

This research is conducted on a 'blind' Frank Emika Panda agent, using the Monte Carlo policy gradient REINFORCE algorithm on a benchmark task in RLBench environment, with non-linear function approximators like ANN to represent the policy. The motive is to understand the ability of the agent to learn to map optimal actions with only proprioceptive inputs, the limitations of REINFORCE, the possible measures to improve the capabilities, and the directions of future research. The agent is initially trained on a simplified benchmark task, with the prime motivation to validate the developed algorithm. Experiments (e.g., visualizing the benefits of subtracting the baseline from the obtained return while calculating the gradient) are carried out on the simplified benchmark task to gain insights into the learning dynamics.

In the field of reinforcement learning, the effect of learning rate decay has hardly been studied [CSW21], which further motivated me to compare and analyze the learning dynamics of REINFORCE algorithm, with both constant and decaying learning rates.

Finally, the agent is trained on the actual (not simplified) task in RLBench, and the results are presented and discussed in chapter 5. A few of the research works related to policy gradient REINFORCE are given in subsection 3.2.1.

### 3.2.1 Related Works

Policy gradient algorithms, especially Monte Carlo REINFORCE have yielded the most real-world robotics results [Jan06b], the following are a few of them. A hardware biped robot learned dynamic walking without any previous knowledge about its dynamic model [Ham97]. The QRIO Humanoid robot is trained using the policy gradient approach to achieve biped walking in a simulated environment, and it demonstrated robust walking in the real world [End05]. A zebra zero robot arm was able to learn successfully, the peg-in-hole insertion task and the ball balancing task, using a model-free policy gradient approach [Vij94]. Using the CPG-actor-critic policy gradient based reinforcement learning model, the CPG controller is trained for the autonomous acquisition of biped locomotion by a biped robot simulator [Tak04].

# 4 Implementation

## 4.1 Simulation in Robotics

Despite the challenges mentioned in section 1.2, simulated environments generate large amounts of training data at low cost, accelerate the engineering design cycle, provide a safe and fully controlled virtual environment for testing and verification, facilitate the research of Human-Robot Interaction (HRI) (e.g., semiautonomous or autonomous operation), and development of better representations of the psycho-social nature of HRI and more intelligent robots [Gab19]. Unless stated otherwise, all the environments are assumed to behave Markovian.

### 4.1.1 RLBench

RLBench[SD19] is a benchmark and learning environment for robot learning which can be used as a standard in place for comparing robotic manipulation methods. RLBench is built around CoppeliaSim/V-REP [EF13] and PyRep [Ste19] interface, which features several improvements including speed, rendering, flexible robot control, and scene manipulation. By default, the agent in the CoppeliaSim scene is Franka Emika Panda, a 7 DoF arm. It includes 100 completely unique, hand-designed tasks, with an array of both proprioceptive observations and visual observations. RLBench seems to be a promising platform for evaluating current and future reinforcement learning algorithms on real-world tasks. One of the key properties of RLBench is the tiered difficulty, i.e., the tasks that are designed are of different difficulty levels starting from easy tasks like reaching to challenging tasks, which can even stress-test well-known state-of-the-art algorithms in use today.

### 4.1.2 Why RLBench?

Now we will take a look at some of the other simulation environments and argue why the choice of RLBench is more suitable for our problem.

### Easy-to-use Task Building Tools:

Most of the tasks featured by the other benchmarks such as OpenAI Gym [al16a] and DeepMind Control Suite [al18b] do not resemble the real-world problems and these benchmarks do not provide any tools to build tasks easily [SD19]. Due to these reasons, researchers created their manipulation tasks in XML using the MuJoCo interface, to evaluate their algorithm. The process of designing the task itself introduces another hyperparameter in the system, which makes comparisons difficult. In the hybrid approach, Simitate [RP19], even though the real-world observations are combined with a simulated environment for benchmarking, it still faces the challenges like time-consuming calibration and motion capturing, whenever there is a need to add new tasks. But whereas RLBench is designed with diversity, reproducibility, scalability, extensibility, realism, and tiered difficulty in mind, which makes it easier to build diverse tasks in a scalable manner.

### Benchmark Platform for Several Research Areas:

RoboNet [al20a] consists of 15 million video frames of different table-top robots, which can be used for pretraining the reinforcement learning model, and then transfer knowledge to a new test environment, whereas RLBench not only seems to be a standard in place for comparing continuous-control reinforcement algorithms involving real-world tasks and but can also serve as a benchmark platform for several research areas such as reinforcement learning, imitation learning, multi-task learning, sim-to-real transfer, and explicit model-based representations [SD19].

### Diverse Range of Tasks

Even though the ACRV Picking Benchmark [al17] for robotic picking and stowing can be reproduced in a laboratory setup, RLBench consists of 100 unique tasks, many of which involve some kind of picking and placing. Each task comes with a number of textual descriptions and an infinite set of demonstrations [SD19].

### Deals With the Complete Robotic Pipeline:

Other few benchmarks focus on sub-problems of manipulation such as object detection, grasp selection, state estimation, and planning [al15c; al14c; al14b; al14a; YF18; al11b; al11a; al15a], on the other hand, RLBench deals with the complete robotic pipeline. Due to the lack of large-scale data beforehand, it was difficult in the RoboCup@Home competition [al15b] to implement reinforcement learning and other end-to-end approaches, whereas RLBench unifies and compares both old and new methods on the same field.

**Access To Visual and Proprioceptive Observations:**

In RLBench, the proprioceptive data of the agent include joint angles, velocities, torques, and the gripper pose, and the robot's visual inputs include RGB, depth, and segmentation masks from an over-the-shoulder stereo camera and an eye-in-hand monocular camera. Given the large number of eye-in-hand camera observations provided, it facilitates research in partial observability and bootstrapping reinforcement learning policies with demonstrations to reduce sample complexity [SD19].

## 4.2 Algorithm Development and Deployment

The algorithm and the policy are the two components of an agent shown in figure 2.1. As a result of trial-and-error interactions with the environment, the agent collects information about the environment. After each trial, the learning algorithm updates the policy parameters with the obtained trajectory information to increase the expected future returns. The updated mapping between the observations and actions guides the future interactions of the agent with the environment.

### 4.2.1 Task Selection

RLBench includes 100 unique tasks ranging from easy to more challenging ones, which can even stress-test well-known state-of-the-art algorithms in use today. Since RLBench was designed with a tiered difficulty feature, it is flexible for the user to choose a task depending on the need and the algorithm capability. The policy gradient approach typically converges to a local optimum rather than a global optimum. As a result of the high variance, policy gradient methods produce slow and unstable learning since the policy improvement happens in small steps and converges slowly. With these challenges, the goal is to select the most simple task available in RLBench. The reach-target task is the most simple one, and the task goal is to reach the target sphere with the panda gripper.

### 4.2.2 Stochastic Policy Representation

Stochastic policy $\pi(a|s, \theta)$ shown in equation 2.1 is represented using simple Gaussian distribution, i.e., for each input state, the actions are assumed to be normally distributed. When the uncertainty of the environment and the complexity of the task involved is high, a combination of multiple Gaussians, i.e., Gaussian Mixture Model (GMM) might be a suitable choice to represent more complex distributions. Many other distributions are available apart from Gaussian and GMM, e.g., beta distribution, etc. The architecture of stochastic policy is depicted in figure 4.1.

Since there are no visual inputs involved, the input data is purely proprioceptive. The input layer of the ANN architecture receives the proprioceptive data, and it is passed forward through the dense layers of the network after feature scaling. The size of the output layer and the complexity of the architecture depends upon the type of distribution needed, e.g., the diagonal covariance matrix consists of off-diagonal zeros, which results in fewer neurons in the output layer, whereas GMM consists of multiple Gaussians along with their weights, which leads to more complex ANN architecture.

Assuming Gaussian, the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ can be build from the output of the ANN. The architecture of the ANN in figure 4.1 is feed-forward. $\varphi(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ represents the normal distribution for the obtained mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. The action $A_t$ can be sampled from the distribution $\varphi(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The likelihood of the Probability Distribution Function (PDF) can be interpreted as the action sampling probability, even though they are not mathematically the same. ANN combined with the process of building the distribution refers to the policy $\pi(a|s, \theta)$ of the agent.

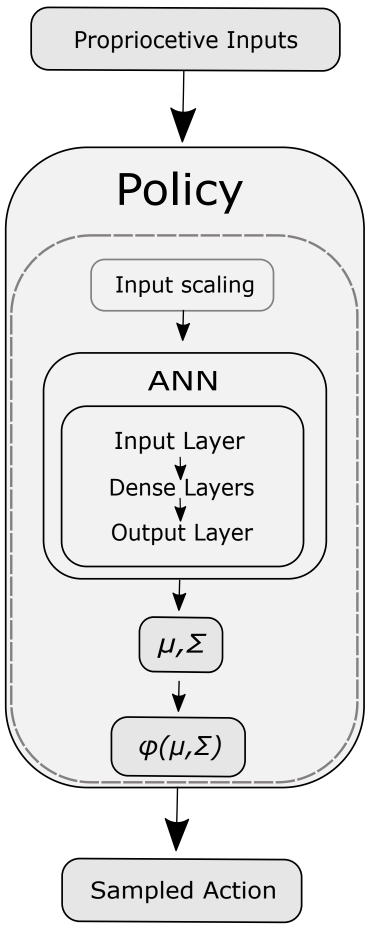


Figure 4.1: Stochastic policy representation

### 4.2.3 Reward Shaping

Reward shaping is a technique inspired by animal training where providing supplemental rewards makes a learning problem easier [E11]. An effective technique for incorporating domain knowledge into reinforcement learning is reward shaping since it speeds up convergence to an optimal policy. Due to human cognitive bias, the transformation of human knowledge into numeric reward values is often not ideal [al20b]. Engineering a reward function depends on the goal and the difficulty level of the task. Based on the selected reach-target task, the inverse of the euclidean distance between the gripper and the target is chosen as the reward function with a maximum reward value to avoid Infinity during training.

### 4.2.4 Trajectory Generation

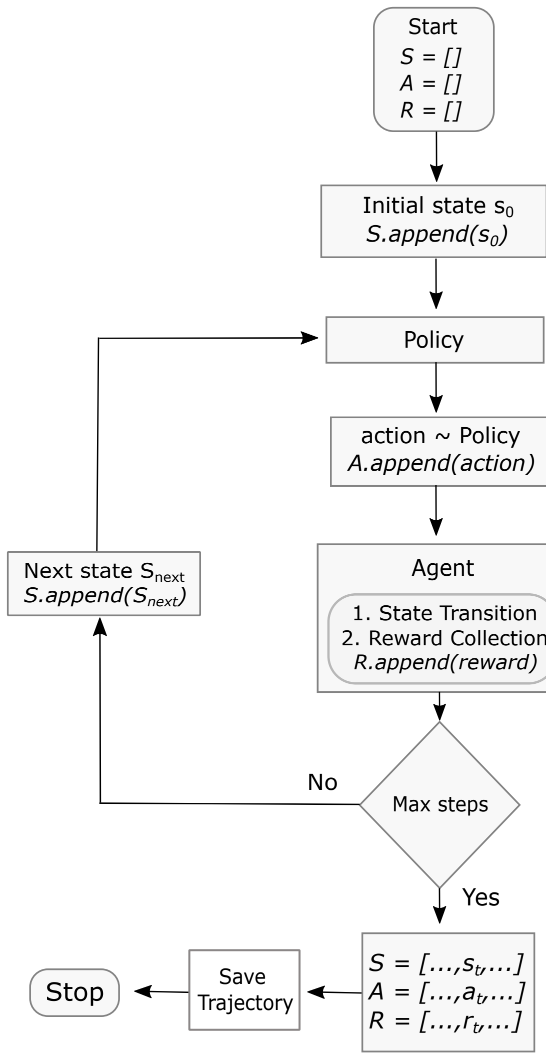


Figure 4.2: Episode generation flowchart

To calculate the return $G_t$ at time $t$ shown in equation 2.8, all the future rewards until the end of the episode are required, i.e., REINFORCE is well defined only for the episodic case. The episode ends only if the predefined maximum number of steps per episode is reached. The flowchart in figure 4.2 depicts the process of episode generation. At the start of each episode, three empty lists are created to store the state, action, and reward for every step the robot takes. $s_0$ refers to the initial state of the robot at the start of episode generation. An action is sampled from the distribution predicted by the policy for the given input state. The sampled action is executed by the robot, as a result, state transition and reward collection take place. The obtained reward, state (before transition), and the action performed, are appended to their respective lists. Due to the state transition, the state of the robot is updated, and then the conditions for the episode termination are verified. If yes, the episode is terminated, or else the current state is given as the input to the policy, and then the same process continues. After the episode termination, the saved trajectory is used by the algorithm to update the policy parameters, which is explained in detail in subsection 4.2.5.

### 4.2.5 Algorithm Deployment

After generating an episode, a list of returns $G$ is computed for all the time steps of the episode from the list of obtained rewards using equation 2.8. The lists of states, actions, and returns are fed into the algorithm to update the parameters of the policy. Equation 2.10 refers to the update formula for REINFORCE, and equations 2.22 and 2.23 refer to the update formulas when there is a baseline function involved. Before the start of training, the maximum number of episodes to be trained is predefined. If the number of generated episodes becomes greater than or equal to the predefined limit, then the training is terminated, or else the process of episode generation and updating the policy continues. If the predefined number of training episodes is not enough, then the training can be continued from where it got terminated. But to do that, the policy, the baseline network, and the optimizer state need to be saved at regular intervals.

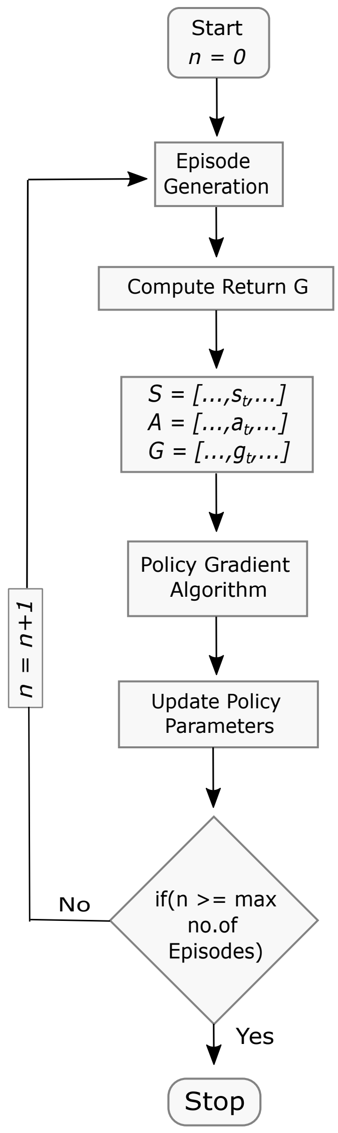

Start
$n = 0$
Episode Generation
Compute Return G
$S = [\ldots, s_t, \ldots]$
$A = [\ldots, a_t, \ldots]$
$G = [\ldots, g_t, \ldots]$
Policy Gradient Algorithm
Update Policy Parameters
if(n >= max no.of Episodes)
No
$n = n+1$
Yes
Stop

Figure 4.3: Algorithm deployment flowchart

# 5 Results and Discussion

The motivation of this specialization project is to implement and evaluate the classical Monte Carlo policy gradient REINFORCE algorithm on benchmark tasks in RLBench and understand how well the agent can learn to map optimal actions with only proprioceptive inputs. The developed algorithm is implemented initially on a simplified setting, and experiments are conducted to understand the effects of baseline and learning rate decay on training. And then the algorithm is deployed on the actual benchmark task in RLBench. The results of both cases are presented and discussed in the following sections.

## 5.1 Architecture Details

The architecture of the ANN proposed in various literature depends on multiple factors like the algorithm used, DoF of the robot, and the task to be trained. The ANN architecture chosen for this use case is similar to the ones used in other relevant literature [LVH17; al15d]. The proprioceptive inputs are normalized in [-1, 1] to speed up training and reduce the chances of getting stuck in local optima, and then they are fed as inputs to the feedforward actor-network as shown in figure 4.1. The proprioceptive input array for the reach-target task in RLBench is of length 40, which contains information about the low dimensional robot state and task state. The number of neurons in the input layer of the actor-network is equal to the array size of proprioceptive inputs. Three hidden layers, each containing 100 neurons, are connected using the relu activation function. Since the robot has 7 DoF, the first 7 neurons of the output layer correspond to the mean vector elements of the Gaussian distribution, with the tanh activation. Assuming multivariate Gaussian with diagonal covariance, the last 7 neurons of the output layer (out of total 14 neurons) represent the diagonal elements of the covariance matrix, with the softplus activation since the covariance matrix is symmetric positive semidefinite with strictly-positive diagonal variances of the random vector [Bek88].

The baseline function explained in subsection 2.2.4 is represented using a feedforward ANN. The size of the input layer is equal to the array size of proprioceptive inputs. Three hidden layers, each containing 20 neurons, are connected using the relu activation function. The output layer consists of a single node, with softplus activation since it approximates the value of the state, which cannot be negative. Both actor and baseline network uses HeNormal initialization [He15] to initialize the kernel weights of the network.

Research shows that more advanced optimization algorithms provide more robust policy gradient learning [Mni+16].Since the Adam optimizer [KB17] takes advantage of momentum by using the moving average of the gradient, it is applied for both the networks instead of the basic gradient descent algorithm.

The maximum number of steps per episode shown in figure 4.2 is chosen to be 40. 0.99 is assigned to the value of the discount factor $\gamma$ shown in equation 2.8, to compute the returns. The results of multiple experiments with constant and decayed learning rates, with and without baseline function, are presented in the following sections. Table 5.1 shows different rates used for the experiments, and the given values support the explanation in the following sections. The absence of $\beta$ represents training without baseline.

Table 5.1: Learning rates of actor and baseline networks to support the explanation in the following sections

| Actor | Baseline |
|---|---|
| $\alpha_1$ = 1e-5 | $\beta_1$ = 1e-3 |
| $\alpha_2$ = 1e-6 | $\beta_2$ = 1e-4 |
| $\alpha_3$ = 1e-5 | - |
| $\alpha_4$ = 1e-6 | - |
| $\alpha_5$ = 1e-7 | - |
| $\alpha_6$ = 5e-7 | $\beta_6$ = 5e-5 |

Since the agent explores and exploits to figure out the optimal policy by maximizing the rewards, the average episodic return, i.e., the mean of the returns of an episode, should tend to increase, and the uncertainty of actions chosen by the agent should decrease, in order to properly learn the expected behavior. During training, the percentage of every 20 generated trajectories that succeeded the task is computed on the fly, which provides more insight into the learning process. The trained agents are validated on 500 roll-outs in a regular interval of training episodes.

## 5.2 Simplified RLBench Benchmark Task

The developed algorithm is first deployed on a simplified benchmark task to fulfill the following,

- Validate the developed algorithm.
- Explain the effect of baseline on the variance of the gradient estimates and the number of training episodes.
- Explain the exploration issues associated with learning rates.

As stated in subsection 4.2.1, the reach-target task is chosen from the list of available benchmark tasks in RLBench. The simplification of the problem is done by fixing the target position at [0.3, 0, 0.8] during training, such that each training trajectory is equally difficult since the position of the goal is constant. For smoothing the data in the plots, the exponential moving average is used, with 0.99 as the smoothing factor.

### 5.2.1 Effect of Baseline

Studies [Zha+11] confirm that the subtraction of the optimal baseline reduces the variance of gradient estimates, improves the stability and learning speed, as explained in subsection 2.2.4. But still, visualizing the benefit with the obtained results helps in validating the algorithm implementation. In this subsection, two agents are trained, one with the baseline, and the other without the baseline.

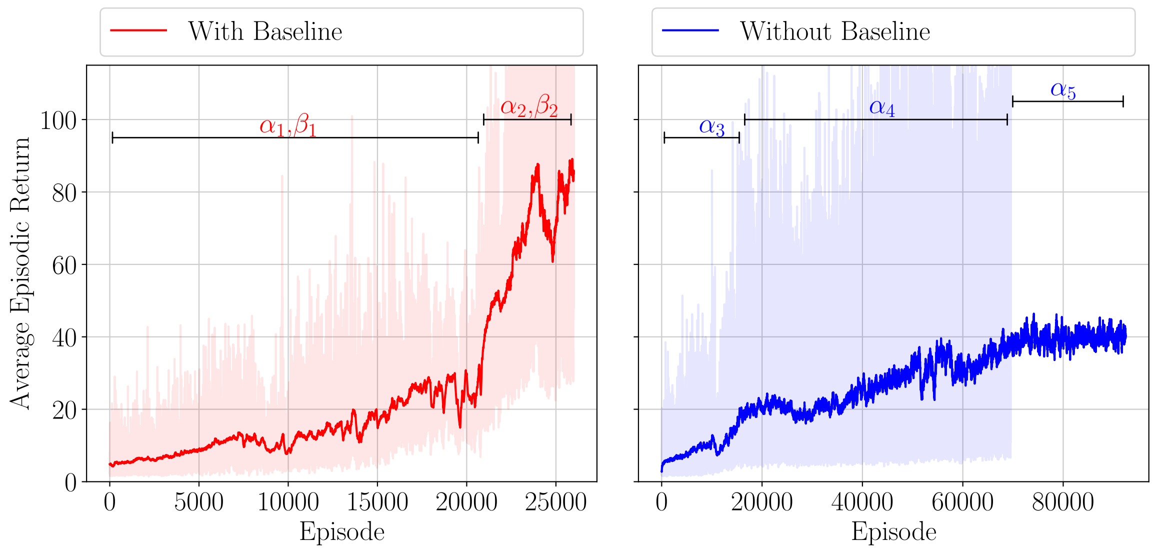


Figure 5.1: Average episodic return. The exponential moving average with a factor of 0.99 is used to smooth the data.

While training with baseline, the learning rate is modified at the 20800th episode. In the absence of a baseline, the learning rate is modified at 16000th and 69400th episodes. The values for the learning rates are shown in table 5.1, and the reason for reducing the learning rate during training is explained in subsection 5.2.2. Figure 5.1 shows a significant reduction in the variance of the average episodic return due to the addition of the baseline function, and as a result, learning speed is notably improved.

From figure 5.2, one can observe that, with a baseline, the agent was able to learn the target behavior almost within 26000 episodes with a 100% success rate, but whereas in the absence of baseline, even after 90000 episodes, the success rate is around 70%.

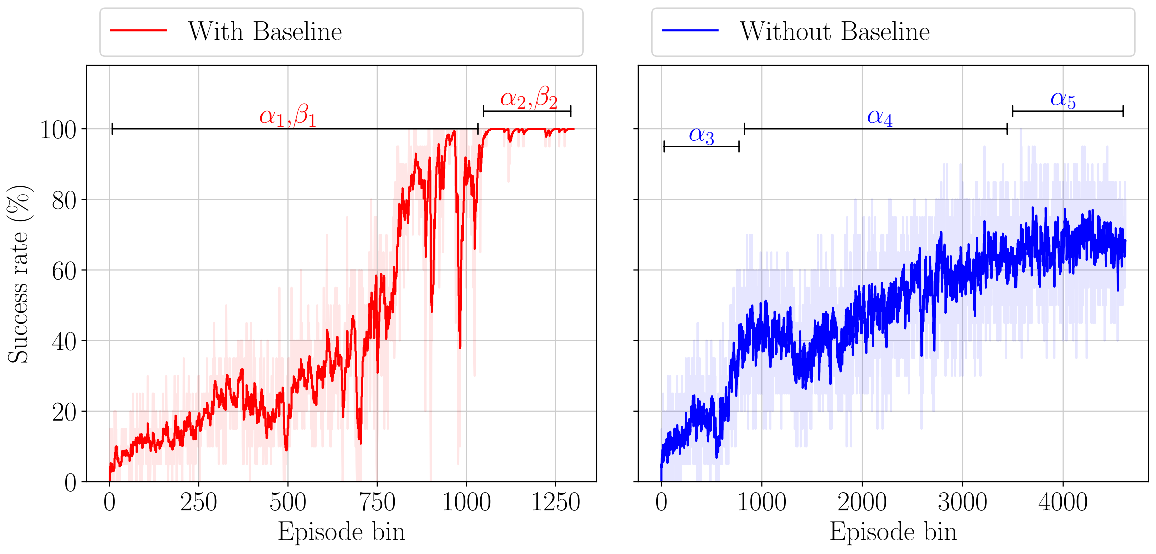


Figure 5.2: The success rate is measured on the fly during training, with 0.99 as a smoothing factor (exponential moving average). Each episode bin in the x-axis represents 20 episodes, starting from ((Episode bin*20)-20) to (Episode bin*20), e.g., the 5th episode bin represents the episodes from 80 to 100, and the respective y-axis value represents the percentage of those 20 episodes that succeeded the task. Even if the agent reaches the target once in a trajectory, then it is considered a success.

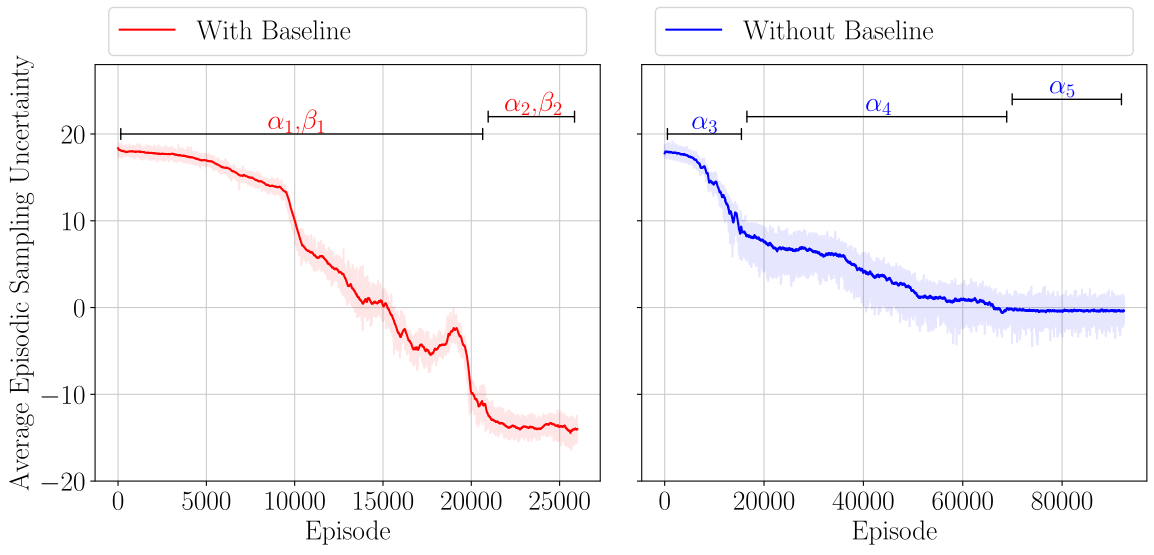


Figure 5.3: In the y-axis, $-ln\ \ L(\boldsymbol{\mu}_{S_t}, \boldsymbol{\Sigma}_{S_t}|A_t)$ is averaged over all the actions $A_t$ of an episode, where $\boldsymbol{\mu}_{S_t}$ and $\boldsymbol{\Sigma}_{S_t}$ represents the mean vector and the covariance matrix built from the actor network predictions, for the given proprioceptive input $S_t$, and $L(\boldsymbol{\mu}_{S_t}, \boldsymbol{\Sigma}_{S_t}|A_t)$ represents the likelihood of a Gaussian with mean vector $\boldsymbol{\mu}_{S_t}$ and covariance matrix $\boldsymbol{\Sigma}_{S_t}$, given the action $A_t$. The exponential moving average with a factor of 0.99 is used to smooth the data.

The successful learning not only reflects the benefit of baseline but also validates the implementation of the algorithm. Even after attaining a 100% success rate, the agent can maintain it, which can be observed in the last 5000 episodes. This is feasible because the agent became more certain about its actions since the variance of the action distribution became low, leading to a high likelihood value which can be visualized in figure 5.3. In the presence of a baseline, the rate of reduction in uncertainty of action sampling is comparatively high.

### 5.2.2 Exploration Associated With Learning Rate

The learning rate decay is a common technique in deep learning to improve the fine-tuning and the learning capabilities of the model to learn complex patterns [You+19]. But still, in the field of reinforcement learning, almost all researches use fixed learning rates, and the effect of learning rate decay has hardly been studied in the past [CSW21]. In recent research, experiments were carried out with Q-learning and Double Q-learning algorithms, and the results show that the best performances are achieved by using the decaying learning rates, and future research is directed to understand the effect of learning rate decay on other reinforcement learning methods [CSW21]. This motivated me to compare and analyze the learning dynamics of REINFORCE algorithm with constant and decaying learning rates.

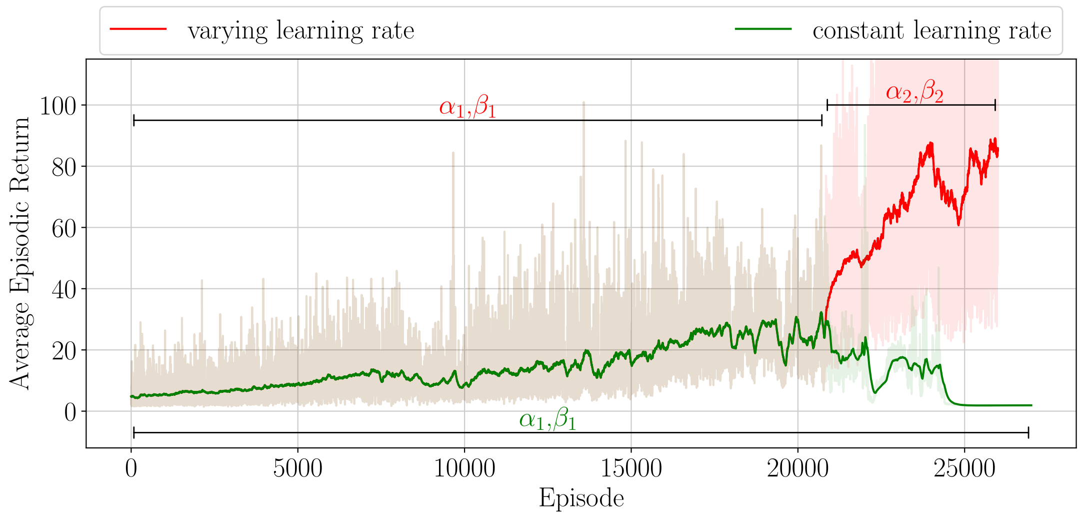


Figure 5.4: Average episodic return. The exponential moving average with a factor of 0.99 is used to smooth the data.

Experiments are carried out, with both constant and decaying learning rates in the presence of a baseline. The plots show the change in learning dynamics after reducing the learning rate at the 20800th episode, and the curves in the plots overlap when they both have the same learning rate.

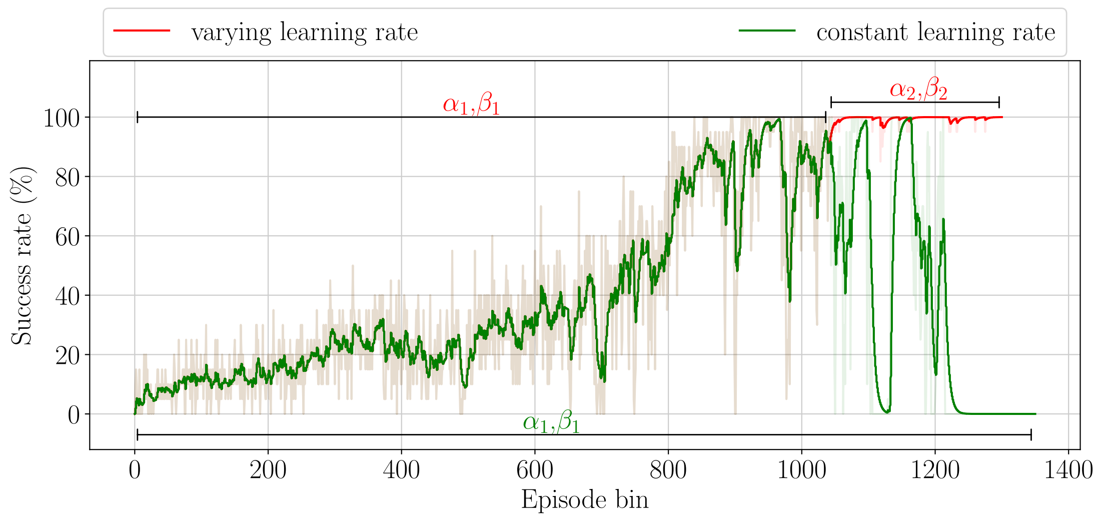


Figure 5.5: The success rate is measured on the fly during training in the presence of a baseline, with 0.99 as a smoothing factor (exponential moving average). Each episode bin in the x-axis represents 20 episodes, starting from ((Episode bin*20)-20) to (Episode bin*20), e.g., the 5th episode bin represents the episodes from 80 to 100, and the respective y-axis value represents the percentage of those 20 episodes that succeeded the task. Even if the agent reaches the target once in a trajectory, then it is considered a success.

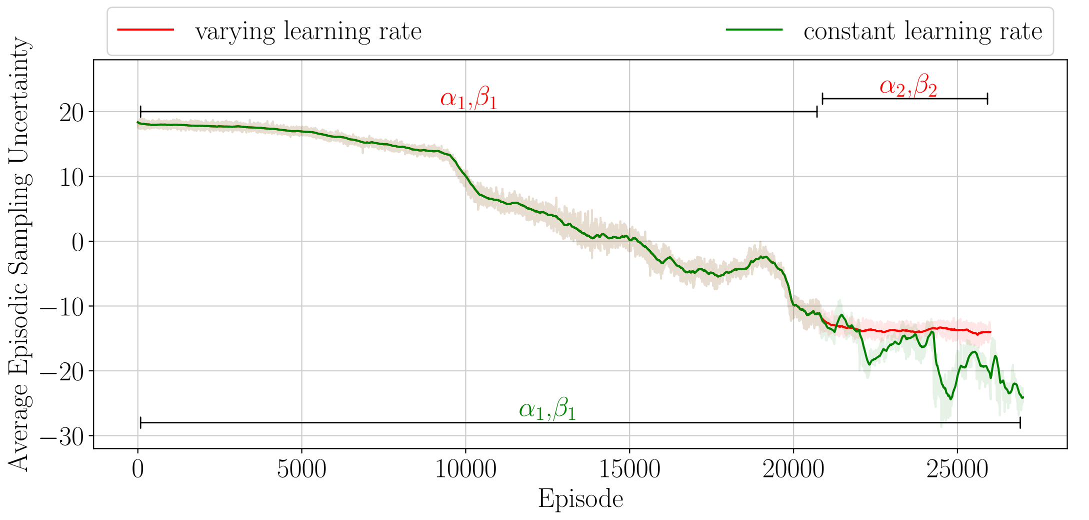


Figure 5.6: In the y-axis, $-ln\ \ L(\boldsymbol{\mu}_{S_t}, \boldsymbol{\Sigma}_{S_t}|A_t)$ is averaged over all the actions $A_t$ of an episode, where $\boldsymbol{\mu}_{S_t}$ and $\boldsymbol{\Sigma}_{S_t}$ represents the mean vector and the covariance matrix built from the actor network predictions, for the given proprioceptive input $S_t$, and $L(\boldsymbol{\mu}_{S_t}, \boldsymbol{\Sigma}_{S_t}|A_t)$ represents the likelihood of a Gaussian with mean vector $\boldsymbol{\mu}_{S_t}$ and covariance matrix $\boldsymbol{\Sigma}_{S_t}$, given the action $A_t$. The exponential moving average with a factor of 0.99 is used to smooth the data.

During training, a continuous drop (green curve) in the average episodic return as shown in figure 5.4 indicates that the deployed learning rate became too high after 20800 episodes, therefore the optimization problem started to diverge instead of converging to an optimal policy. The experiment is, if there is a continuous drop in the performance from a particular point in time during training, then the agent is retrained from that point with a lower learning rate (red curve). In deep learning, a similar technique is deployed to train the model for the image recognition task. The learning rate is divided by 10 when the error plateaus, i.e., if the change in the loss with respect to the training iterations is less than a threshold value. This learning rate decay scheduler resulted in better performance [He+15].

After a certain point in time during training, if the learning rate is high, then the agent takes big steps and jumps around, and it becomes almost impossible for the agent to learn minute variations of the network parameters to fine-tune the model. Due to the big steps taken, the optimization problem diverges, and the agent learns a non-sense policy, which leads to a continuous drop in the average episodic return. And as a result, even if the agent is able to obtain 100% success rate with a constant learning rate (green curve), it is unable to maintain its performance since the learning rate became high after some point in time, and started to fluctuate and ended at 0% as shown in figure 5.5.

In figure 5.4, in the case of the constant learning rate, one can observe around 25000 episodes, the variance of the average episodic return started to become very small, and hence it is barely visible in the plot. And after that, the curve became almost flat, without any fluctuations or growth. Similarly, in figure 5.5, in the case of the constant learning rate, after reaching a 0% success rate at the end of the training, there is no further improvement in its performance. These behaviors can be reasoned out by observing figure 5.6, which shows that after 20800 episodes, due to big learning steps (green curve), the agent is not only learning non-optimal policy but also becoming highly certain with its non-sense actions, i.e., in other words, the policy quickly becomes deterministic without sufficiently exploring. Being an on-policy algorithm, REINFORCE evaluates and improves the same policy, and hence, if the learned policy doesn't make sensible actions to attain success, or if it became almost deterministic, then the agent will be unable to explore the search space properly. So, the early deterministic behavior, without enough exploration, can be avoided by decaying the learning rate during training.

### 5.2.3 Validation of the Trained Agent

In this subsection, we will validate the trained models, which are saved during training. As shown in figures 5.2 and 5.5, computing the success rate on the fly during training gives a good overview of the change in learning dynamics, but still, evaluating the trained model on many roll-outs can predict the success rate with comparatively high confidence. It's because of the stochastic nature of the agent's policy. Since validating the trained model requires many roll-outs, it is not feasible to save the model after generating each episode and validate it parallelly during training. Due to this downside, trained models are saved at regular intervals and then validated by evaluating the performance on multiple roll-out trajectories.

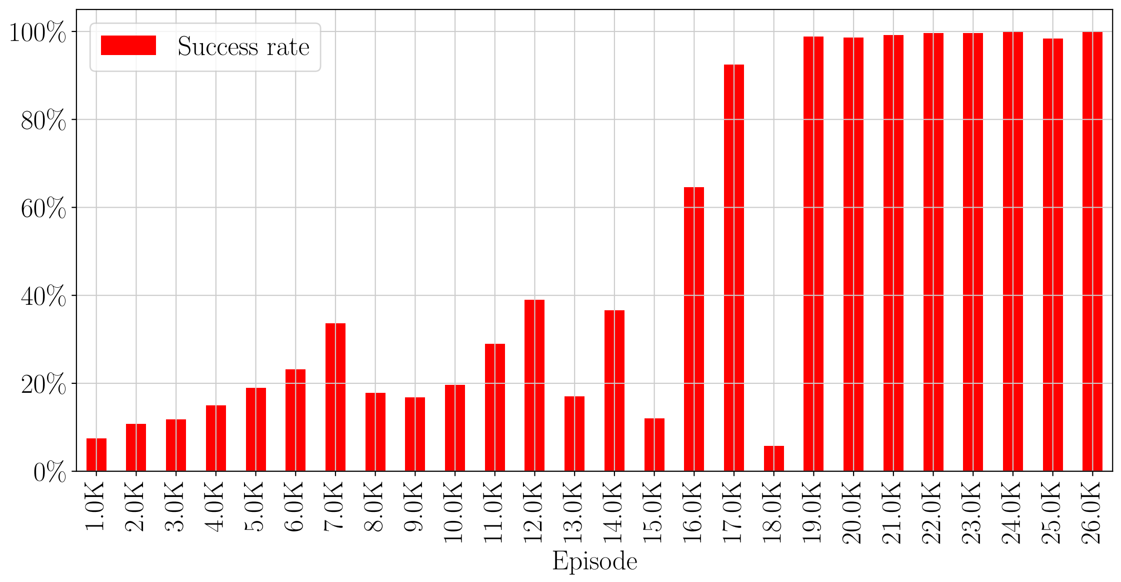
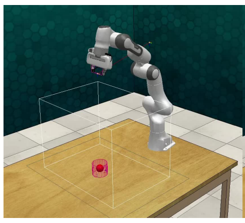
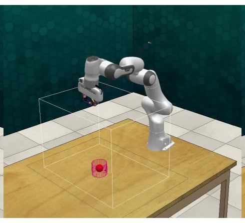
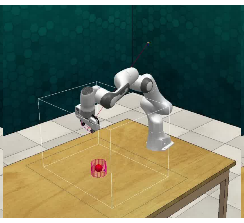
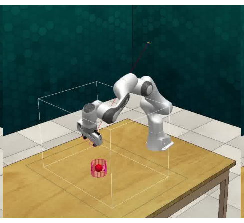
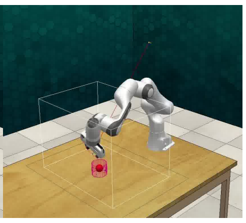
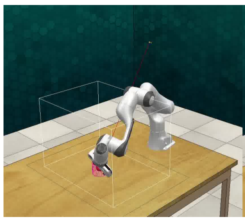
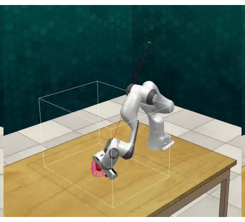
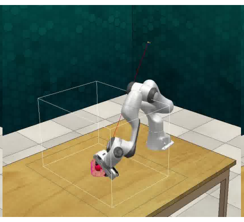
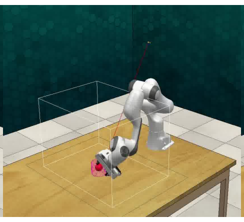
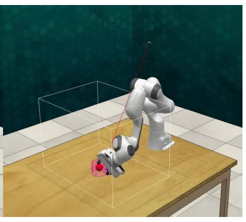


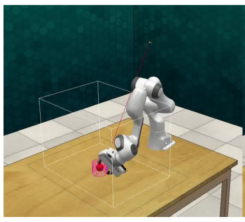
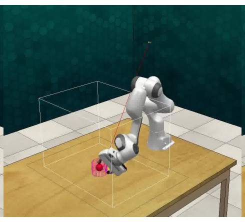
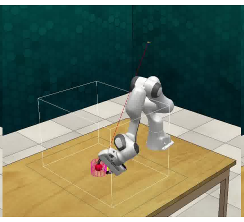
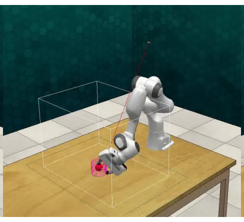
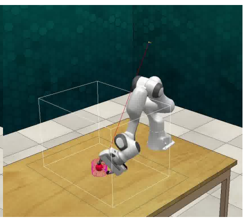
Figure 5.7: For every 1000 episodes, the agent is validated on 500 roll-outs, and the success rate is evaluated. Even if the agent reaches the target once in a roll-out, then it is considered a success.

Figure 5.7 shows the validation results of the agent from subsection 5.2.2 (red curve), which was trained successfully with baseline and decaying learning rate. From figure 5.7, one important thing to note is that the agent was able to attain almost 100% success rate even around 19000 episodes, but from figure 5.4, one can observe that the average episodic return is only around 30 at 19000 episodes, and it went up approximately till 90 at the end of training. But even after obtaining a cent percent success rate, why there is a significant increase in average episodic return? What kind of additional behavior did the agent learn in between 19000 and 26000 episodes?

In order to find explanations for the above questions, let us look at figure 5.8, which represents a sample roll-out trajectory simulation through 20 timeframes, executed by the trained agent. As explained in section 5.1, the number of steps per episode is 40, i.e., even though the agent reaches the target beforehand, still it should complete the total 40 timesteps (do not confuse timesteps with timeframes) to terminate an episode. Even if the agent reaches the target once in an episode, then it is considered a success. An interesting behavior is observed in figure 5.8, that is even after reaching the target (8th timeframe), the agent learned to stay at the target till the termination of the episode, in order to maximize the average episodic return. This was the behavior the agent leaned in between 19000 and 26000 episodes, and as a result, there was a significant increase in the average episodic return.


(a) Timeframe 1

(b) Timeframe 2

(c) Timeframe 3

(d) Timeframe 4

(e) Timeframe 5


(f) Timeframe 6

(g) Timeframe 7

(h) Timeframe 8

(i) Timeframe 9

(j) Timeframe 10


(k) Timeframe 11

(l) Timeframe 12

(m) Timeframe 13

(n) Timeframe 14

(o) Timeframe 15

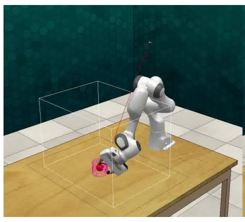
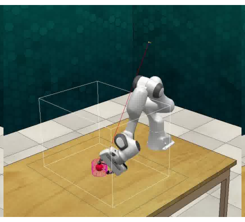
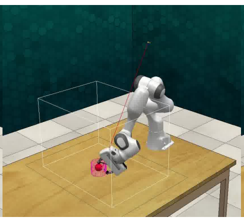
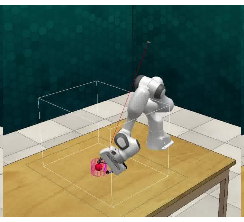
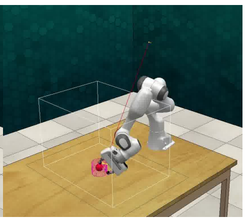

(p) Timeframe 16

(q) Timeframe 17

(r) Timeframe 18

(s) Timeframe 19

(t) Timeframe 20

Figure 5.8: Timeframes of a CoppeliaSim simulation of a sample roll-out executed by an agent, which was successfully trained with baseline, on 26000 episodes. The red ball represents the target.

## 5.3 RLBench Benchmark Task

Apart from the validation of the developed algorithm, the other key takeaways from the experiments in subsections 5.2.1 and 5.2.2 include the following,

- As an effect of the baseline function, the variance of the average episodic return decreases, and as a result, the learning speed is notably improved.
- By decaying the learning rate during training, the early deterministic behavior of the agent can be avoided.

In this section, the insights gained from the previous experiments are used to improvise the training. Since this section deals with the actual benchmark task (unlike section 5.2), the goal position is not fixed during training, and the learning happens in the presence of the baseline. The architecture of the algorithm is the same as explained in section 5.1. For smoothing the data in the plots, the exponential moving average is used, with 0.99 as the smoothing factor.

### 5.3.1 Need to Maintain Stochasticity

In this subsection, the learning rate is modified at 16400th and 47900th episodes during training in the presence of baseline. By using the insights from subsection 5.2.2, it can be noticed from figure 5.9, that the agent did not experience any continuous drop in the performance during training. Even though the agent did not become deterministic by learning a non-sense policy, it still suffers from the exploration issue. In figure 5.9, the average episodic return of the agent is around 10, even if the number of training episodes reaches 200k, but whereas in figure 5.4, the maximum value at the end of the successful training is around 90. Even though figure 5.4 deals with a simplified case, it gives a good estimate of the average episodic return at the end of the successful training. Figure 5.10 shows that the maximum success rate measured on the fly during training is only around 10%, and figure 5.11 shows that the maximum success rate measured on 500 roll-outs during validation is also around 10%, which reflects that there is still room for improvement. But unfortunately, the uncertainty of action sampling in figure 5.12 became very low, and the learning rates $\alpha_6$ and $\beta_6$ are also very low, which makes it difficult for the agent to explore unseen regions in the search space. That's where the need to maintain the stochasticity of the action sampling during training arises, to explore the action space properly, without becoming certain about its non-sense actions in advance. The following subsections deal with a few of the capabilities that REINFORCE algorithm lacks, and as a result, it is unable to maintain the stochasticity of the action sampling.

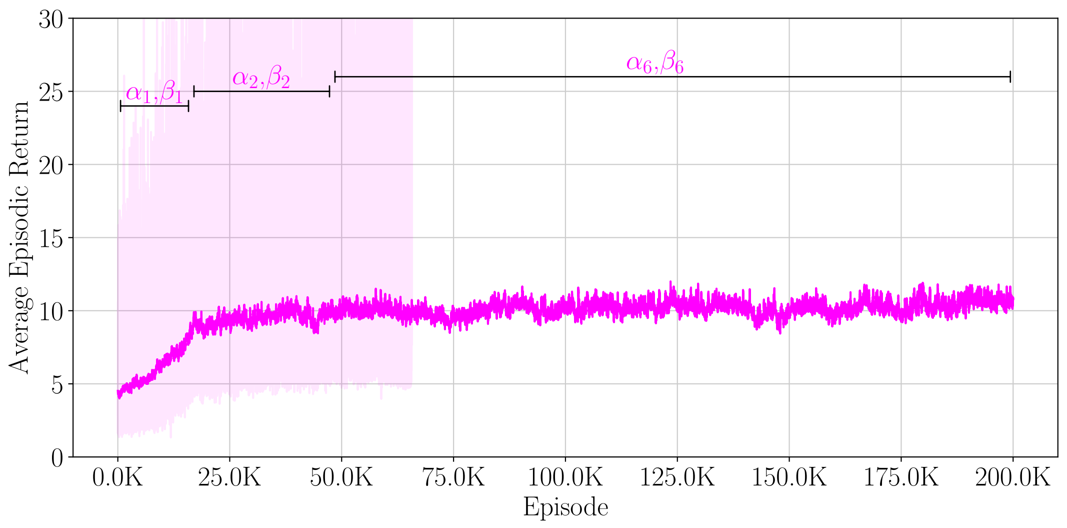


Figure 5.9: Average episodic return. The exponential moving average with a factor of 0.99 is used to smooth the data.

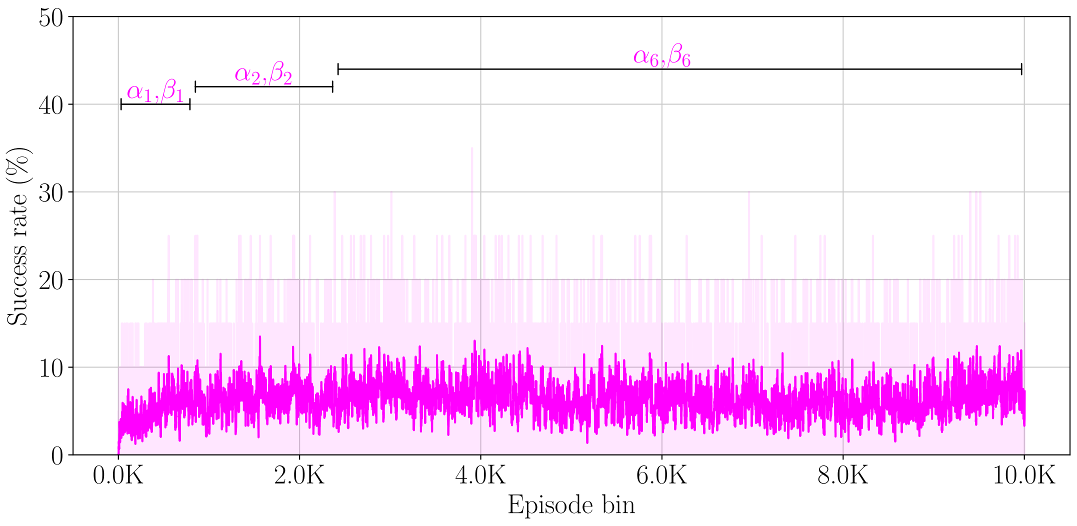


Figure 5.10: The success rate is measured on the fly during training in the presence of a baseline, with 0.99 as a smoothing factor (exponential moving average). Each episode bin in the x-axis represents 20 episodes, starting from ((Episode bin*20)-20) to (Episode bin*20), e.g., the 5th episode bin represents the episodes from 80 to 100, and the respective y-axis value represents the percentage of those 20 episodes that succeeded the task. Even if the agent reaches the target once in a trajectory, then it is considered a success.

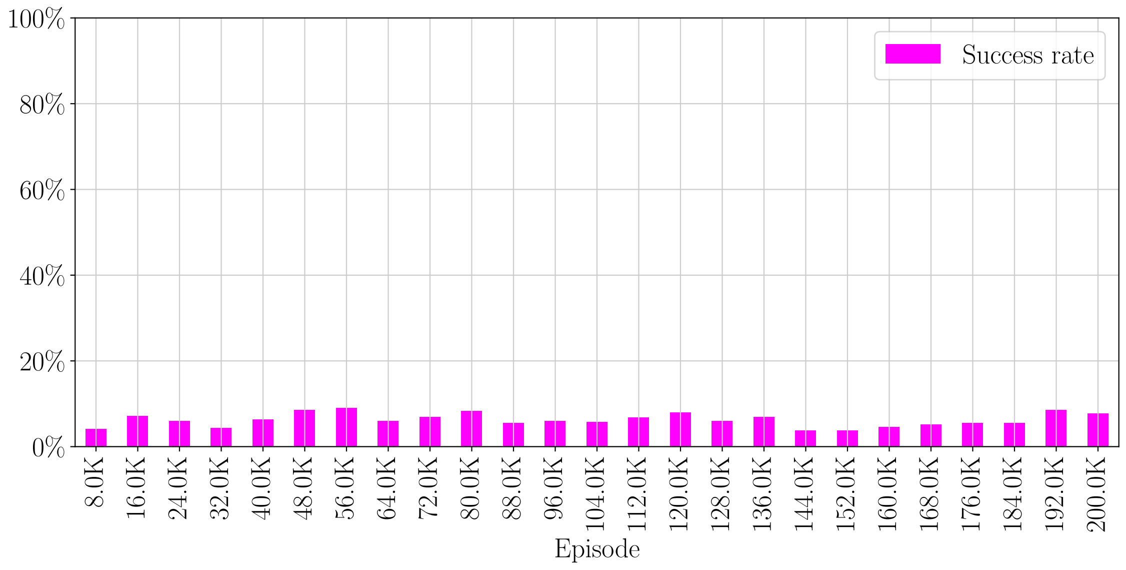


Figure 5.11: For every 8000 episodes, the agent is validated on 500 roll-outs, and the success rate is evaluated. Even if the agent reaches the target once in a roll-out, then it is considered a success.

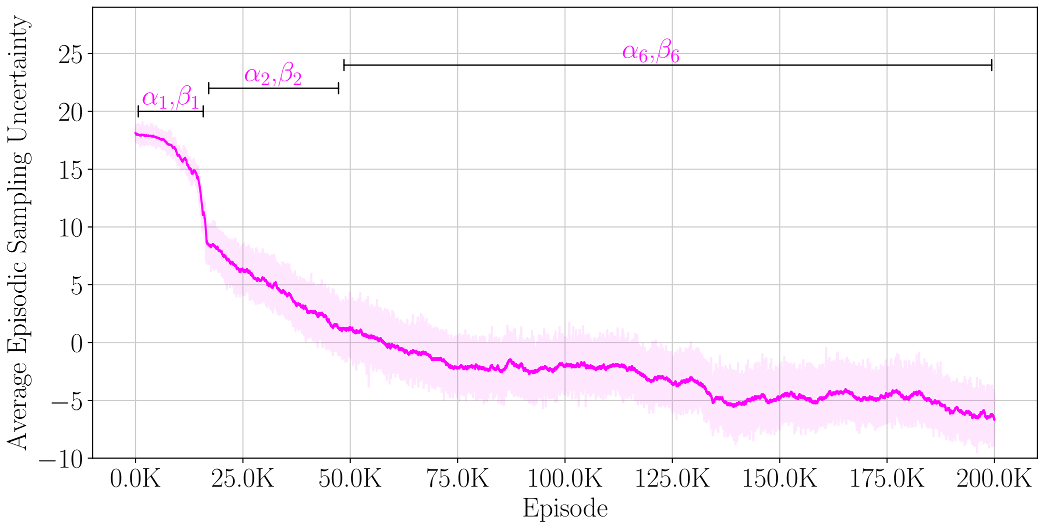


Figure 5.12: In the y-axis, $-ln\ L(\boldsymbol{\mu}_{S_t}, \boldsymbol{\Sigma}_{S_t}|A_t)$ is averaged over all the actions $A_t$ of an episode, where $\boldsymbol{\mu}_{S_t}$ and $\boldsymbol{\Sigma}_{S_t}$ represents the mean vector and the covariance matrix built from the actor network predictions, for the given proprioceptive input $S_t$, and $L(\boldsymbol{\mu}_{S_t}, \boldsymbol{\Sigma}_{S_t}|A_t)$ represents the likelihood of a Gaussian with mean vector $\boldsymbol{\mu}_{S_t}$ and covariance matrix $\boldsymbol{\Sigma}_{S_t}$, given the action $A_t$. The exponential moving average with a factor of 0.99 is used to smooth the data.

### 5.3.2 Addressing Premature Convergence

Premature convergence is a common problem, which means that the agent got stuck in a local optimum during optimization. Even if we can perfectly estimate the gradient, the reason for the premature convergence is that the policy optimization in reinforcement learning would still be difficult due to the geometry of the objective function. [Ahm+18]. Since the exact solution to tackle this problem is unknown, research is going on in multiple directions [DT12; Zon97; Vik+18]. In this subsection, let us look at a few of the approaches, which are more relevant to our use case in the context of reinforcement learning, to overcome premature convergence.

#### Off-Policy Learning:

REINFORCE is an on-policy learning algorithm since it evaluates and improves the same policy, which is being used to select actions, i.e., both the target policy and the behavior policy are the same. In off-policy learning, the target policy and the behavior policy are different. Even though off-policy learning has its own challenges [SB18], it possesses multiple advantages when compared to on-policy learning.

- Since the behavior policy is different from the target, the agent can explore continuously, whereas on-policy learns sub-optimal policy, and hence off-policy algorithms give flexibility in dealing with the trade-off between exploration and exploitation.
- Initializing multiple simulated environment objects in parallel can speed up the learning by generating multiple trajectories.
- Off-policy learning facilitates multitask learning, by using one stream of experience to solve multiple tasks simultaneously.
- The agent can even learn from demonstrations when certain training scenarios require human guidance in the form of kinesthetic teaching.

#### Entropy Regularization:

Adding the entropy of the policy $\pi$ to the objective function improves exploration by discouraging premature convergence to suboptimal policies [Mni+16].

$$\mathcal{H}^{\pi}(.|S_t) = -\sum_{a_t} \pi(a_t|S_t, \theta) \ln \pi(a_t|S_t, \theta) \tag{5.1}$$

Equation 5.1 represents the entropy of the policy $\pi$, given the state $S_t$ [WW19]. Due to the addition of the entropy regularization term, equation 2.11 representing the gradient of the objective function is modified as [Mni+16],

$$\nabla J(\theta) = \mathbb{E}_{\pi}\left[(G_t - b(S_t))\nabla \ln \pi(A_t|S_t,\theta) + \tau\nabla\mathcal{H}^{\pi}(.|S_t)\right] \tag{5.2}$$

$\tau >= 0$ is the user-specified temperature parameter that controls the strength of entropy regularization, if $\tau \to 0$, then it is the same as equation 2.11, without entropy term. Several advantages of regularizing the entropy are listed below,

- Since entropy encourages exploration, regularizing entropy helps the agent avoid situations that might lead to a sub-optimal convergence.
- The optimization problem with the additional entropy term results in a smoother objective function that can be optimized with a larger learning rate, and hence it speeds up the training [Ahm+18].
- Entropy regularization facilitates the agent to re-utilize the learned behavior to fine-tune the policy for a more advanced task, instead of learning from scratch [TL17].
- Since the exploration is encouraged due to the maximum-entropy policy, the agent becomes more robust to rare states of the environment.
- Recent research in reinforcement learning deploys entropy maximization, which led to the formulation of new approaches like Soft Actor-Critic [Haa+18] and A3C [Mni+16].

Even though there are multiple advantages, there is no assurance that the entropy regularization results in significant solution improvements in all situations. The influence of entropy is not guaranteed to change the landscape of the objective function significantly, in order to find better solutions at all times. Its because the impact of entropy on the objective function depends both on the problem and the environment. [Ahm+18].

# 6 Conclusion and Future Works

REINFORCE Algorithm is developed and deployed to train agents on both the simplified and actual versions of a benchmark task, using the same architecture. Baseline decreased the variance of the gradient estimates and significantly reduced the training time. Experiments show that decaying the learning rate during training results in better performance and helps to avoid early deterministic behavior. In the future, the agent performance needs to be analyzed with different learning rate schedulers to understand its impact on training. With the help of baseline function and the learning rate reduction, the agent was able to learn the task successfully in the case of a simplified setting, by not only reaching the target but also learned to stay at the target till the termination of the episode, whereas the occurrence of premature convergence is observed during training of the actual task.

Even though the simulated agent performed well in the simplified setting of the task, as a part of the future work, the feasibility of executing the learned behavior on a real Panda robot needs to be verified. Despite the simplicity of the task considered in this research, it is a persistent research problem to understand when and how well such training on simulated data succeeds and how it transfers to the real world.

Research shows that the optimization landscapes are highly problem and environment dependent [Ahm+18]. Since there is a difference in both the problem cases considered in this research, the geometry of the objective functions is different. Intuitively, the increased complexity in the actual problem case might have generated a more complicated optimization landscape with many local optima. In that case, other techniques to maintain the stochasticity of action sampling during training might help the agent to explore unseen regions of search space.

Even though approaches like off-policy behavior and entropy regularization exist, it is not assured to improve the results at all times. Since the optimization landscapes are highly problem and environment dependent, it is highly challenging to develop general-purpose optimization algorithms, which work well for all optimization landscapes. This emphasizes the need to better understand the underlying optimization landscape in direct policy optimization problems by investigating the aspects of the problem and the environment, which induces difficulties in the objective function. Similar to entropy regularization, batch normalization is an objective function smoothing technique, which might lead to improvement in performance [SM19]. Some research should be directed on alternative smoothing techniques to develop advanced methods for smooth convergence even with large learning rates.

# Bibliography

[Ahm+18] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. „Understanding the impact of entropy on policy optimization“. In: *arXiv:1811.11214v5* (2018), pages 1–25.

[al 5] Joonho Lee et al. „Learning Quadrupedal Locomotion over Challenging Terrain“. In: *Science Robotics* (2020, Volume 5), pages 1–22.

[al11a] Mila Popović et al. „Grasping unknown objects using an early cognitive vision system for general scene understanding“. In: *Proc. Int. Conf. Intell. Robots Syst.* (2011), pages 987–994.

[al11b] S. Ulbrich et al. „The OpenGRASP benchmarking suite: An environment for the comparative analysis of grasping and dexterous manipulation“. In: *Proc. Int. Conf. Intell. Robots Syst.* (2011), pages 1761–1767.

[al13] Volodymyr Mnih et al. „Playing Atari with Deep Reinforcement Learning“. In: *arXiv:1312.5602* (2013).

[al14a] Arjun Singh et al. „Bigbird: A large-scale 3D database of object instances“. In: *IEEE Int. Conf. Robot. Autom.* (2014), pages 509–516.

[al14b] Mark Everingham et al. „The PASCAL Visual Object Classes Challenge: A Retrospective“. In: *International Journal of Computer Vision* (2014), 98–136, Volume 111.

[al14c] Tsung-Yi Lin et al. „Microsoft COCO: Common Objects in Context“. In: *European Conference on Computer Vision* (2014), pages 740–755.

[al15a] Berk Calli et al. „Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set“. In: *Robot. Autom. Mag.* (2015), 36–52, Volume 22.

[al15b] LucaIocchi et al. „RoboCup@Home: Analysis and results of evolving competitions for domestic and service robots“. In: *Artificial Intelligence* (2015), 258–281, Volume 229.

[al15c] Olga Russakovsky et al. „ImageNet Large Scale Visual Recognition Challenge“. In: *International Journal of Computer Vision* (2015), 211–252, Volume 115.

[al15d] Timothy P. Lillicrap et al. „Continuous control with deep reinforcement learning“. In: *arXiv:1509.02971v6 [cs.LG]* (2015), pages 1–14.

[al16a] Greg Brockman et al. „OpenAI Gym“. In: *arXiv:1606.01540* (2016), pages 1–4.

[al16b] Sergey Levine et al. „Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection“. In: *arXiv:1603.02199v4* (2016).

[al17] Jürgen Leitner et al. „The ACRV Picking Benchmark (APB): A Robotic Shelf Picking Benchmark to Foster Reproducible Research“. In: *arXiv:1609.05258v2* (2017), pages 1–8.

[al18a] Marcin Andrychowicz et al. „Learning Dexterous In-Hand Manipulation“. In: *CoRR* (2018), pages 1–27.

[al18b] Yuval Tassa et al. „DeepMind Control Suite“. In: *arXiv:1801.00690* (2018), pages 1–24.

[al19] Dian Chen et al. „Learning by Cheating“. In: *arXiv:1912.12294* (2019), pages 1–12.

[al20a] Sudeep Dasari et al. „RoboNet: Large-Scale Multi-Robot Learning“. In: *arXiv:1910.11215* (2020), pages 1–15.

[al20b] Yujing Hu et al. „Learning to Utilize Shaping Rewards: A New Approach of Reward Shaping“. In: *arXiv:2011.02669v1* (2020), pages 1–22.

[Bac13] Pierre-Luc Bacon. In: *https://gist.github.com/pierrelux/6501790.js* (2013).

[Bek88] Paul A. Bekker. „The Positive Semidefiniteness of Partitioned Matrices “. In: *Elsevier Science Publishing Co., Inc* (1988), pages 261–278.

[BK19] Aude Billard and Danica Kragic. „Trends and challenges in robot manipulation“. In: *Science* (2019), 1–8, Volume 364, Issue 6446.

[Cal18] Sylvain Calinon. „Robot learning with task-parameterized generative models“. In: *Proceedings of the 2018 International Symposium on Robotics Research* (2018), pages 111–126.

[CMa18] Dan C.Marinescu. „Big Data, Data Streaming, and the Mobile Cloud“. In: *Cloud Computing* (2018), pages 439–487.

[CSW21] Yifei Chen, Lambert Schomaker, and Marco Wiering. „An Investigation Into the Effect of the Learning Rate on Overestimation Bias of Connectionist Q-learning“. In: *13th International Conference on Agents and Artificial Intelligence* (2021).

[DT12] Kathryn A. Dowsland and Jonathan M." Thompson. „Simulated Annealing“. In: *Springer* (2012), pages 1623–1655.

[E11] Wiewiora E. *Reward Shaping*. Boston, MA, USA: Encyclopedia of Machine Learning., Springer, 2011.

[EF13] S. P. N. Singh E. Rohmer and M. Freese. „V-REP: A versatile and scalable robot simulation framework“. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2013), pages 1321–1326.

[El-16] Mohammed E. El-Telbany. „The Challenges of Reinforcement Learning in Robotics and Optimal Control“. In: *International Conference on Advanced Intelligent Systems and Informatics* (2016), pages 881–890.

[End05] Gen Endo. „Learning cpg sensory feedback with policy gradient for biped locomotion for a full-body humanoid“. In: *American Association for Artificial Intelligence* (2005), pages 1267–1273.

[Fre13] Olivier Sigaud Freek Stulp. „Robot Skill Learning: From Reinforcement Learning to Evolution Strategies“. In: *PALADYN Journal of Behavioral Robotics* (2013), pages 49–61.

[Gab19] Todd Heste Gabriel Dulac-Arnold Daniel Mankowitz. „Challenges of Real-World Reinforcement Learning“. In: *arXiv:1904.12901* (2019), pages 1–13.

[Haa+18] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, et al. „Soft Actor-Critic Algorithms and Applications“. In: *arXiv:1812.05905v2* abs/1812.05905 (2018), pages 1–17.

[Ham97] Judy A. Franklin Hamid Benbrahim. „Biped dynamic walking using reinforcement learning“. In: *Robotics and Autonomous Systems* (1997), pages 283–302.

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. „Deep Residual Learning for Image Recognition“. In: *arXiv:1512.03385v1* (2015).

[He15] Kaiming He. „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification“. In: *IEEE Xplore* (2015), pages 1026–1034.

[Jan06a] Stefan Schaal Jan Peters. „Policy Gradient Methods for Robotics“. In: *International Conference on Intelligent Robots and Systems, Beijing* (2006).

[Jan06b] Stefan Schaal Jan Peters. „Policy Gradient Methods for Robotics“. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing* (2006), pages 2219–2225.

[Jea14] Tamim Asfour Jeannette Bohg Antonio Morales. „Data-Driven Grasp Synthesis—A Survey“. In: *IEEE Transactions on Robotics* (2014), 289–309, Volume 30.

[Jea17] Karol Hausman Jeannette Bohg. „Interactive Perception: Leveraging Action in Perception and Perception in Action“. In: *IEEE Transactions on Robotics* (2017), 1273–1291, Volume 33.

[JP13] J. Andrew Bagnell Jens Kober and Jan Peters. „Reinforcement learning in robotics: A survey“. In: *The International Journal of Robotics Research* (2013), pages 1238–1274.

[KB17] Diederik P. Kingma and Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *arXiv:1412.6980v9 [cs.LG]* (2017).

[KK18] Joris Vaillant Karim Bouyarmane Kevin Chappellet and Abderrahmane Kheddar. „Quadratic Programming for Multirobot and Task-Space Force Control“. In: *IEEE Transactions on Robotics* (2018), 64–77, Volume 35.

[Lex13] Nigel Tao Lex Weaver. „The Optimal Reward Baseline for Gradient · Based Reinforcement Learning“. In: *arXiv:1301.2315* (2013), pages 538–545.

[LVH17] Winfried Lötzsch, Julien Vitay, and Fred Hamker. „Training a deep policy gradient-based neural network with asynchronous learners on a simulated robotic problem“. In: *Gesellschaft für Informatik, Bonn* (2017), pages 2143–2154.

[Mar18] Gary Marcus. „Deep Learning: A Critical Appraisal“. In: *arXiv:1801.00631* (2018), pages 1–27.

[Mas18] Matthew T. Mason. „Toward Robotic Manipulation“. In: *Annual Review of Control, Robotics, and Autonomous Systems* (2018), 1–28, Volume 1.

[Mni+16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, et al. „Asynchronous Methods for Deep Reinforcement Learning“. In: *arXiv:1602.01783v2 [cs.LG]* (2016).

[Nic16] Aude Billard Nicolas Sommer. „Multi-contact haptic exploration and grasping with tactile sensors“. In: *Robotics and Autonomous Systems* (2016), 48–61, Volume 85.

[OBK10] J.Piater O.B.Kroemer R.Detry. „Combining active learning and reactive control for robot grasping“. In: *Robotics and Autonomous Systems* (2010), 1105–1116, Volume 58.

[Oli19] George Konidaris Oliver Kroemer Scott Niekum. „A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms“. In: (2019).

[Pea19] Judea Pear. „CAUSALITY: MODELS, REASONING, AND INFERENCE“. In: *Econometric Theory* (2003, Volume 19), pages 675–685.

[RP19] Viktor Seib Raphael Memmesheimer Ivanna Mykhalchyshyna and Dietrich Paulus. „Simitate: A Hybrid Imitation Learning Benchmark“. In: *arXiv:1905.06002* (2019), pages 1–7.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* second edition. Cambridge, MA, USA: A Bradford Book, 2018.

[SD19] David Rovick Arrojo Stephen James Zicong Ma and Andrew J. Davison. „RLBench: The Robot Learning Benchmark and Learning Environment“. In: *arXiv:1909.12271* (2019), pages 1–8.

[Seu14] Aude Billard Seungsu Kim Ashwini Shukla. „Catching Objects in Flight“. In: *IEEE Transactions on Robotics* (2014), 1049–1065, Volume 30.

[SH18] Michael Schukata Seyed Sajad Mousavi and End Howley. „Deep Reinforcement Learning: An Overview“. In: *Proceedings of SAI Intelligent Systems Conference* (2018), pages 1–17.

[Shi15] Sachin Sakhare Shipra Ojha. „Image processing techniques for object tracking in video surveillance-A survey“. In: *IEEE International Conference on Pervasive Computing* (2015), pages 1–6.

[SM19] Andrew Ilyas Shibani Santurkar Dimitris Tsipras and Aleksander Madry. „How Does Batch Normalization Help Optimization?“ In: *arXiv:1805.11604v5* (2019), pages 1–26.

[Ste19] Andrew J. Davison Stephen James Marc Freese. „PyRep: Bringing V-REP to Deep Robot Learning“. In: *arXiv:1906.11176* (2019), pages 1–4.

[Stu20] David Stutz. In: *https://github.com/davidstutz/latex-resources.git* (2020).

[Tak04] Masa-aki Sato Takeshi Mori Yutaka Nakamura. „Reinforcement Learning for a CPG-driven Biped Robot“. In: *IAmerican Association for Artificial Intelligence* (2004), pages 623–630.

[TL17] Pieter Abbeel Tuomas Haarnoja Haoran Tang and Sergey Levine. „Reinforcement Learning with Deep Energy-Based Policies“. In: *International Conference on Machine Learning* (2017), pages 1–16.

[Vij94] Hamid Benbrahim Vijaykumar Gullapalli Judy A.Franklin. „Aquiring robot skills via reinforcement learning“. In: *IEEE Control Systems* (1994), pages 13–24.

[Vik+18] Adam Viktorin, Roman Senkerik, Michal Pluhacek, and Tomas Kadavy. „Addressing Premature Convergence with Distance based Parameter Adaptation in SHADE“. In: *International Conference on Systems, Signals and Image Processing* (2018), pages 1–5.

[Wil92] Ronald J. Williams. „Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning“. In: *Machine Learning* (1992), 229–256, Volume 8.

[WW19] Shiji Song Wenjie Shi and Cheng Wu. „Soft Policy Gradient Method for Maximum Entropy Deep Reinforcement Learning“. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence* (2019), pages 3425–3431.

[YF18] Venkatraman Narayanan Yu Xiang Tanner Schmidt and Dieter Fox. „PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes“. In: *arXiv:1711.00199v3* (2018), pages 1–10.

[You+19] Kaichao You, Mingsheng Long, Michael I. Jordan, and Jianmin Wang. „How Does Learning Rate Decay Help Modern Neural Networks?“ In: *arXiv:1908.01878v2* (2019).

[Yul14] Ferdous Sohel Yulan Guo Mohammed Bennamoun. „3D object recognition in cluttered scenes with local surface features: A survey“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2014), pages 2270–2287.

[Zha+11] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. „Analysis and Improvement of Policy Gradient Estimation“. In: *Advances in Neural Information Processing Systems* (2011).

[Zon97] Yong Gao Zongben Xu. „Characteristic analysis and prevention on premature convergence in genetic algorithms“. In: *Science China Technological Sciences* (1997), pages 13–125.