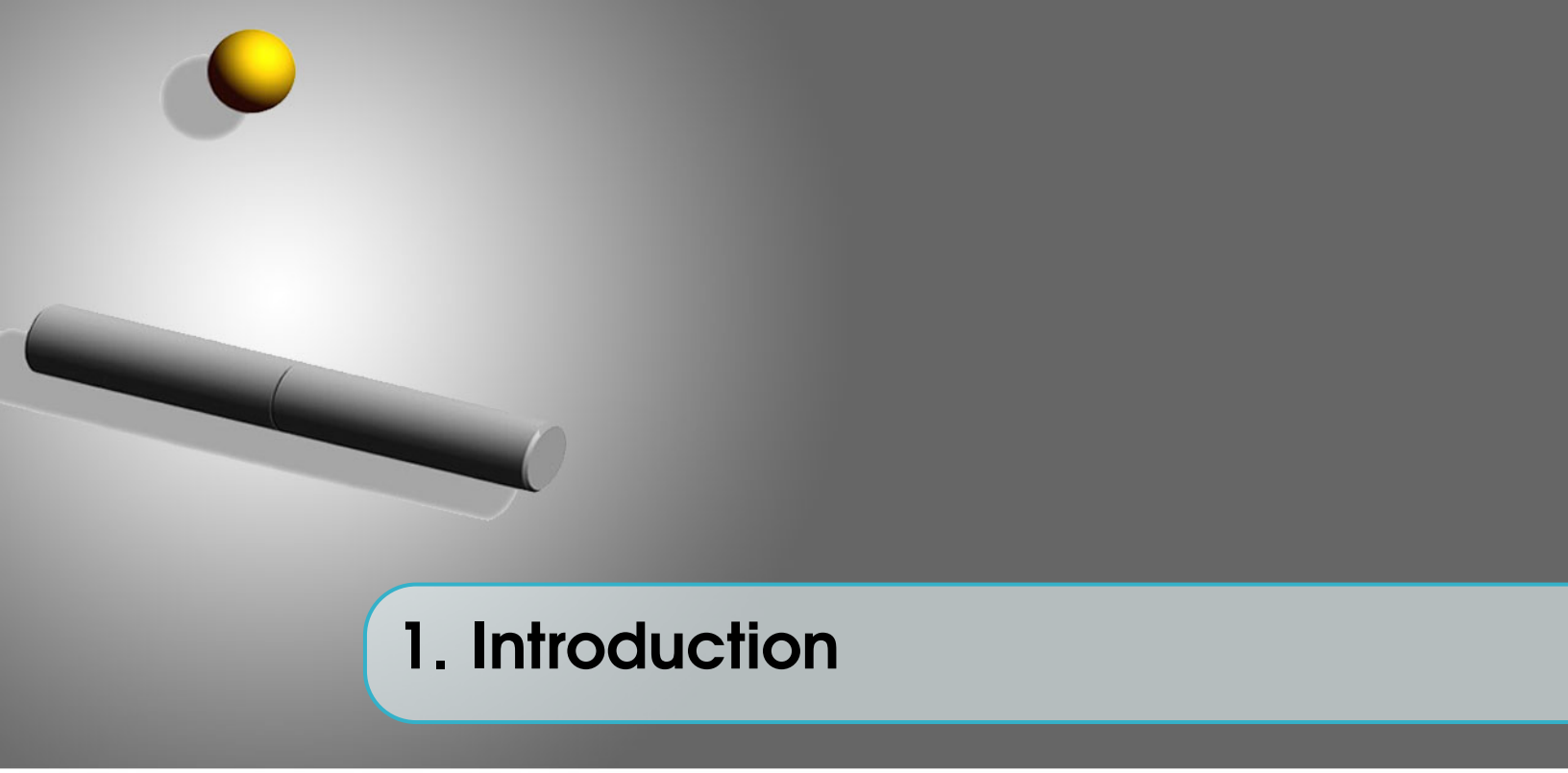
# Deep Reinforcement Learning on a Multi-joint Robot

ROBO-FISH

Nambi Srivatsav - 1211266430



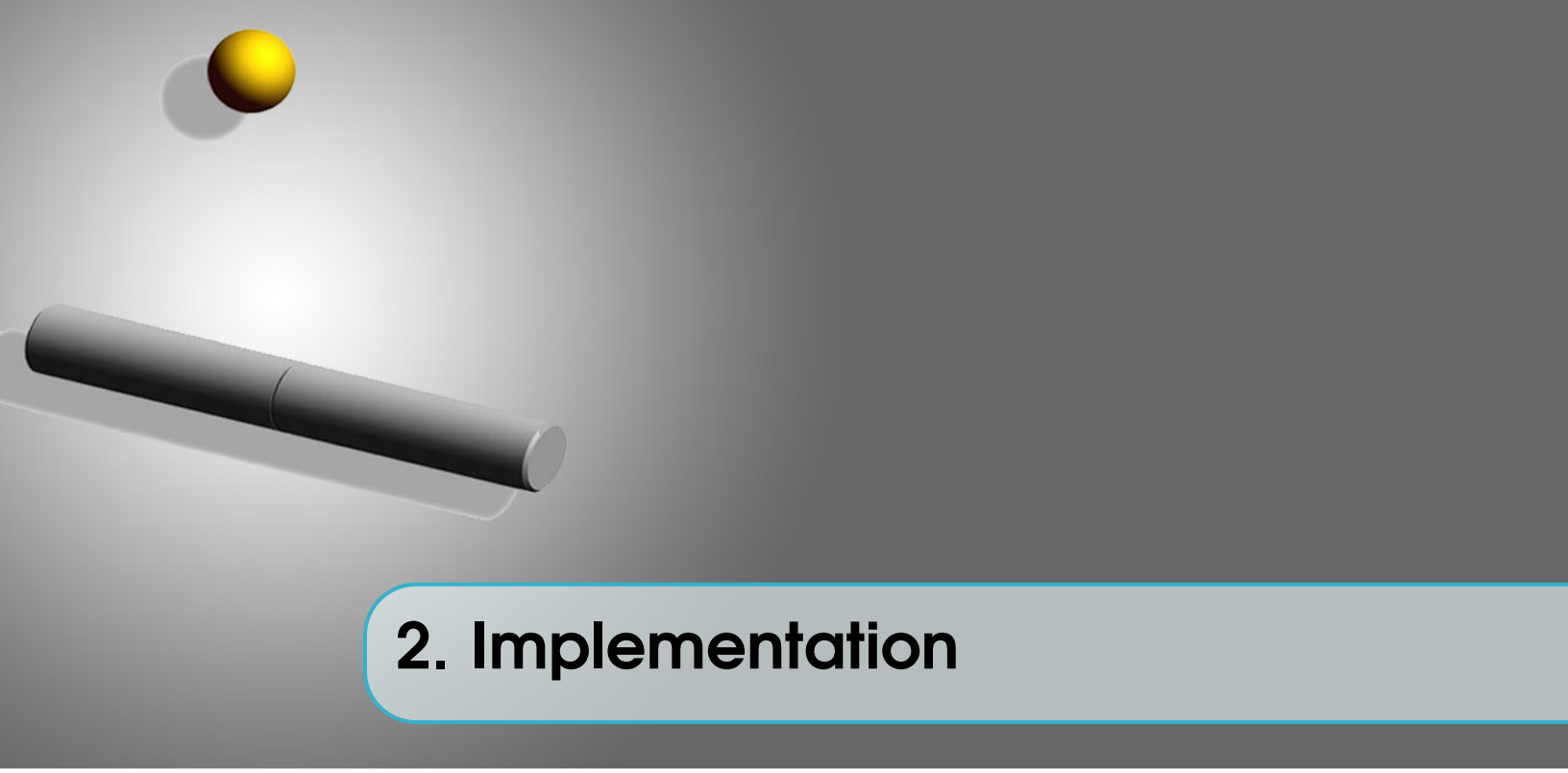
# 1. Introduction

## 1.1 Motivation

There has been huge success with Deep Deterministic Policy Gradient(DDPG) over continuous action domain. The main intention behind this project is to use DDPG algorithm to solve various robotic tasks modeled over Mujoco and OpenAI wrappers. Most notable tasks of the physics control have continuous and high dimensional action spaces. Deep DPG (DDPG) can learn competitive policies for most the tasks tried using low-dimensional observations (cartesian coordinates,joint angles,etc) using the same hyper-parameters and network structure. In order to evaluate this method constructed a two types of challenging physical control problems that involve multi-joint movements, unstable and rich contact dynamics. The main feature about this approach is its simplicity. It can be implemented by straightforward actor-critic architecture and learning algorithm and easier to scale to more difficult problems and larger networks.

## 1.2 Objective

The project consists of three main tasks. First, use the DDPG algorithm on two joint arm to reach a ball that is randomly placed in arena of fixed area.Second, got the learning from the two joint arm and use it to train the three joint arm and observe how fast the the three joint arm is learning. Third, used the same model and same network architecture on different task, swimming.



# 2. Implementation

## 2.1 Algorithm

DDPG is an actor-critic algorithm as well. It mainly consists of two neural networks, one for the actor and one for the critic. Each of the networks compute action predictions for the current state and generate a temporal-difference (TD) error signal each time step of the episode. The critic's output is simply the estimated Q-value of the current state and of the action given by the actor. The critic's output is the estimated Q-value of the current state and of the action given by the actor network. The input of the actor network is the current state, and the output is a 1 Dimensional array of 3 real real values representing an action chosen from a continuous action space. The deterministic policy gradient theorem provides the update rule for the weights of the actor network. The critic network is updated from the gradients obtained from the TD error signal.

The main key factors in this implementation are follows:

1. The TD error signal is excellent at compounding the variance introduced by your bad predictions over time. Therefore, this algorithm has a technique called experience replay buffer. I have used a replay buffer to store the experiences of the agent during training, and then sampled the experiences randomly to use for learning. It is used to break the temporal correlations within different training episodes.

2. The usage of target networks, It has been observed that directly updating critic and action network weights with the gradients obtained from the TD error signal that was computed from both your replay buffer and the output of the actor and critic networks causes your learning algorithm to diverge. The better solution to address this issue is to use a set of target networks to generate the targets for the TD error computation. This approach increases the stability of the learning.

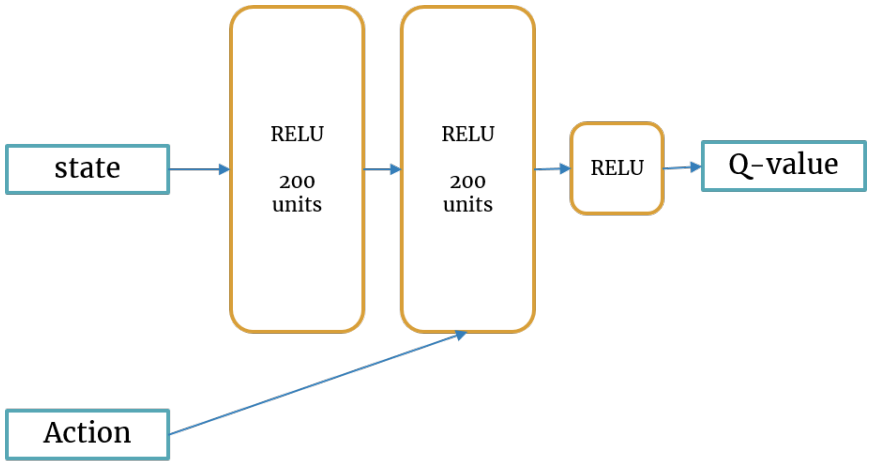
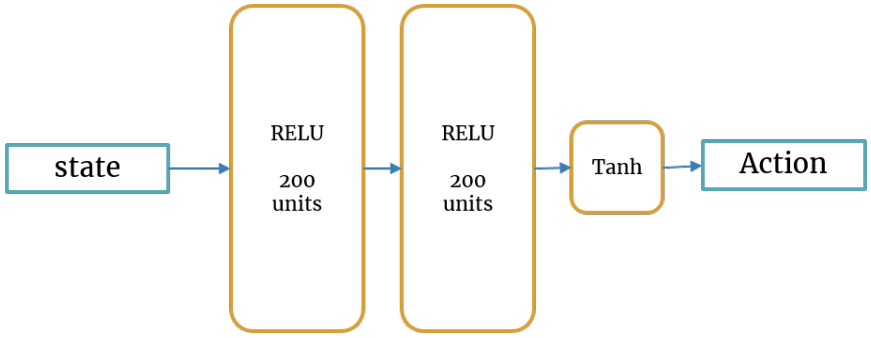



Figure 2.1: ACTOR NETWORK




## 2.2 Architecture

The architecture consists of four neural networks actor network, critic network, target actor network and critic network respectively. It has been hosted over a linux system over 8GPUs NVIDIA Tesla K80, 60GB RAM and 16CPUs of Intel Haswell over Google Cloud.

Modeling in MUJOCO : It is a physics engine aiming to facilitate research and development in robotics, graphics and animation, and other areas where fast and accurate simulation is needed. It offers a unique combination of speed, accuracy and modeling power, yet it is not merely a better simulator.

Open AI wrappers : The models are build on OpenAI gym's control environment. OpenAI, It is toolkit for developing and comparing reinforcement learning algorithms. The wrappers provide the observation space and action space.

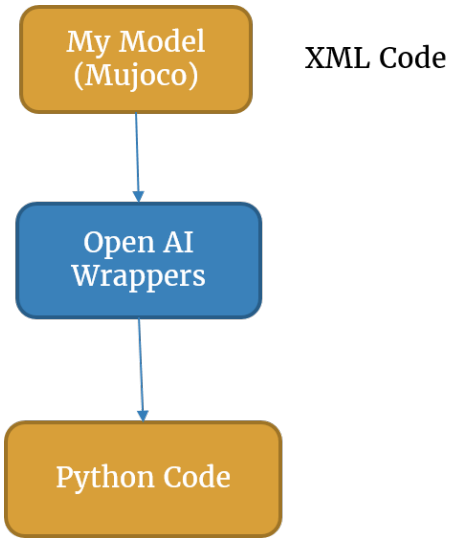



Figure 2.2: Basic Overview

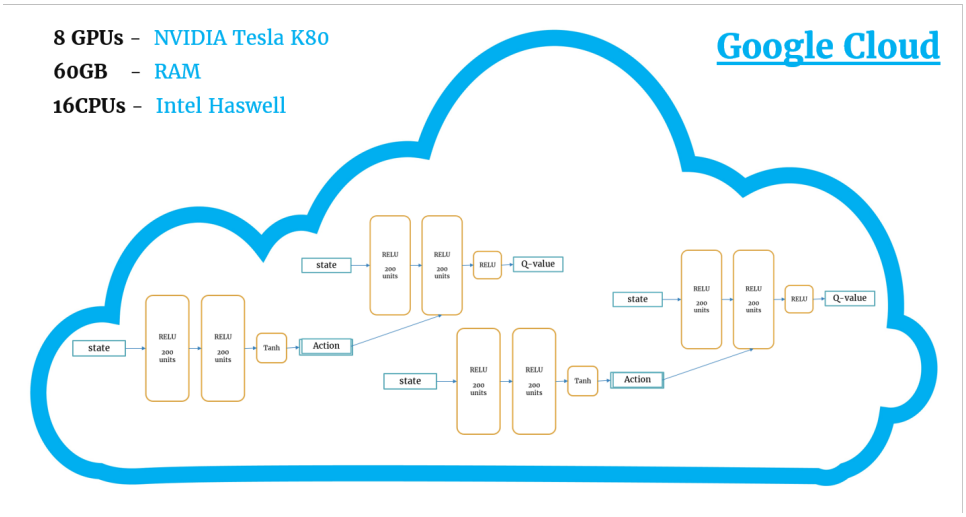
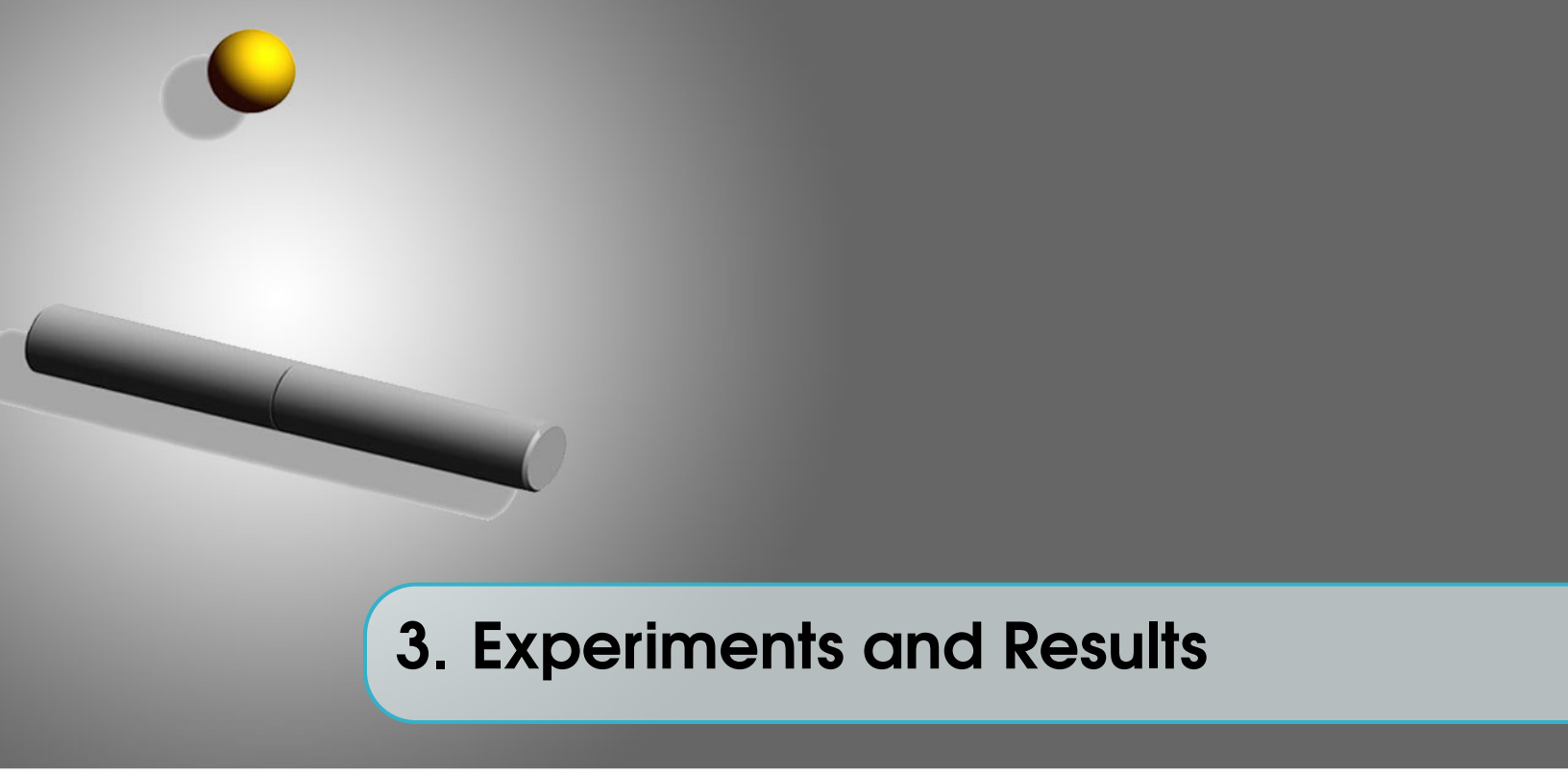



Figure 2.3: Architecture



# 3. Experiments and Results

First Task, Robot arm learns how to reach the ball placed randomly on a arena on the simulator. Action space is 1D with real values and Observation space is 12D. It took around 2 million episodes to reach average reward value of -4.63. The reward results graph along with number of episodes is listed below in Fig 3.1. I have also tested the same algorithm by running it for 4 million episodes to see if there is any improvements in the reward values. There was not any significant improvement. Captured the video using pre-defined rendering functions of OpenAI

Second Task, After training the two joint arm for 2 Million episodes for reaching a constant reward value, a model with three joint arm is trained with pre-trained network of two joint arm. The three joint arm is able to learn pretty fast and able to get a reward value of -8.61. The reward results graph along with number of episodes is listed below in Fig 3.2. When, the same model has been trained from scratch, it was taking 2 million iterations to reach an average reward value of -8.62. It is clearly observable that the model learns fast when used with pre-trained network.

Third Task, was to teach the same planar robot to swim with 3 links and 2 actuated joints in a viscous container, with the goal of moving forward Adversary applied 3D force on the center of the swimmer. Used the same network architecture and same hyper parameters for the networks but need some improvements and more experiments with hyper parameters for better convergence.

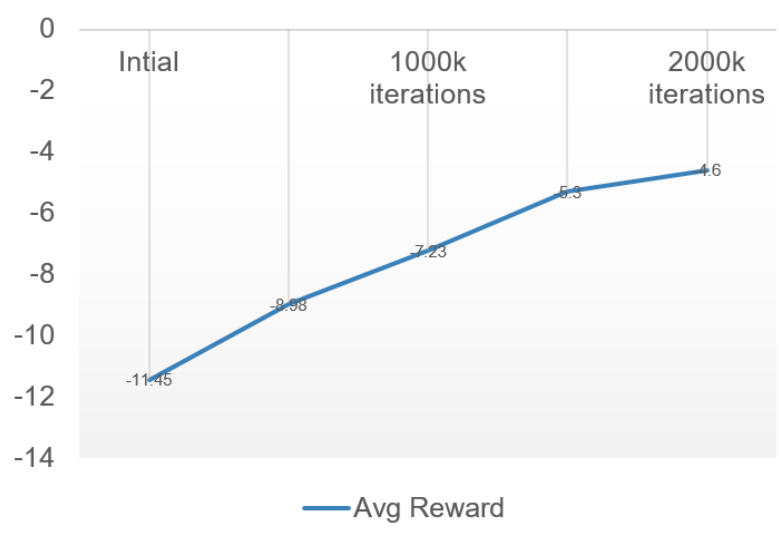



Figure 3.1: Basic Overview

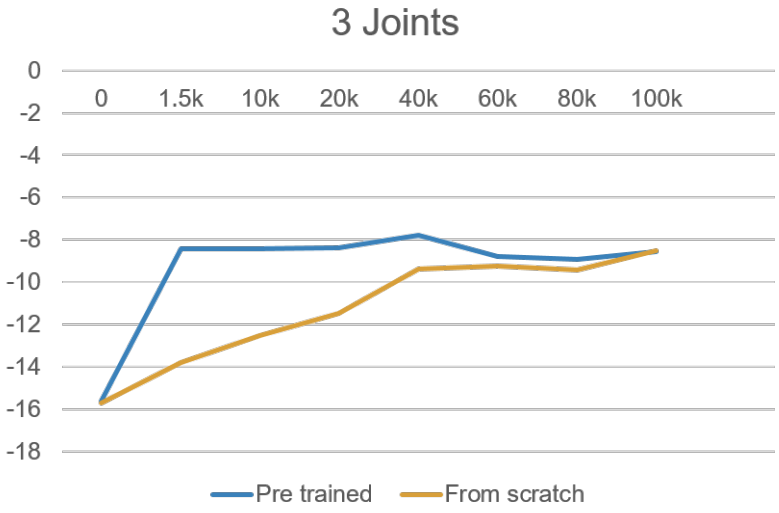



Figure 3.2: Basic Overview