# Exploring reinforcement learning based techniques for robot control

Akshay Iyer
*RBE Dept, WPI*
abiyer@wpi.edu

Amey Kulkarni
*RBE Dept, WPI*
askulkarni2@wpi.edu

Chinmay Burgul
*RBE Dept, WPI*
cmburgul@wpi.edu

Tyagaraja Ramaswamy
*RBE Dept, WPI*
tramaswamy@wpi.edu

***Abstract*—Reinforcement learning (RL) combined with function approximators have gained a lot of attention recently by beating the world champion in the game of Go as well as human experts in various Atari games. In general, RL algorithms look to solve problems where there is an agent interacting with an environment and aim to maximize rewards received from the environment. In dynamic manipulation tasks, using model based trajectory optimization methods(classical control concepts) makes it difficult to obtain the accurate dynamic model and state estimates. To overcome this challenge, in this work, we explore state of the art model free reinforcement learning based techniques which learn the dynamic model itself by sampling state space from exploring the environment. We are exploring two algorithms - Deep Deterministic Policy Gradient (DDPG) and Hindsight Experience Replay (HER) and trying it on multiple environments. We have been able to successfully implement vanilla DDPG and DDPG with HER and obtained satisfactory results on environments like the inverted pendulum and bit flippers. We have tried several variations and parameter tuning for the Fetch environment and found that variations like normalizing inputs and using OU noise instead of Gaussian noise have shown improvements in performance.**

***Index Terms*—Reinforcement Learning, Fetch Robot, DDPG, HER, Mujoco**

## I. Introduction

One of the long-standing goal of a roboticist is to create a general purpose robotic system that can perform wide variety of tasks in human-centered environments such as hospitals and homes. For a robot to perform a wide variety of tasks requires high-versatility like Dexterous Manipulation in small work volume, Object Recognition in a cluttered environment, Navigating in an unknown environments.

In this paper we focus on versatility in terms of Manipulation. The widely known approach of controlling and planning robot manipulation is model based approach. In which, we create a mathematical representation of a dynamic model of a system and linearize to represent it in a state-space format, which introduce an error to it, and then we apply control law to control the output and compensate the error by introducing PD and PID gains to it. This approach is simple and used for many applications. When the system gets dynamically complex and have multiple degrees of freedom like anthropomorphic hand gripper it is hard to represent the accurate dynamic model of the system because of the varying contact forces, and contact frictions comes into picture. Controlling high degrees of freedom action space becomes very complex.

In this project we are following a model-free approach in which we are not aware of the mathematical representation of the dynamical model of the system. The system explores itself in the environment and learns the exact dynamics in terms of transition probability which is saved in terms of weights in a neural network. In this project we are using a Model free Reinforcement Learning Algorithm of Deep Deterministic Policy Gradient to train the agent to perform certain tasks.

The model based approach on the fetch robot will work perfectly if we give a goal point to reach or a trajectory to follow. But it is a complex task to perform the dynamic tasks like pushing a puck to the goal position. The model free approach is one of the solution for such dynamic tasks.

## II. Related Work

*1) Model based trajectory optimization:* Dexterous Manipulation tasks have been successfully achieved by using model based trajectory optimization methods [4] [5] [6] in simulation domains especially when the dynamics can be adjusted and relaxed to make them more tractable. These methods depends on accurate dynamic models and state estimates which are difficult to obtain for contact rich manipulation tasks, in the real world. And these approaches struggle to translate to real world manipulation as learning complex models on real world systems with significant contact dynamics is difficult.

*2) Model Free Reinforcement Learning:* Model Free Reinforcement Learning Methods [10] [8] and Deep Reinforcement Learning Methods [2] [12] do not require dynamic modeling of the system. They will know the configuration of the system by exploring and function approximating a neural network. Infact, they are function approximating the dynamical model of the system by learning from experiences. But for this they require a huge sample space. Few methods overcome this need by using simple policy representation [13] [14]. But to solve real world robotics tasks more complex representation is required to feed rich sensory information.

*3) Imitation Learning:* In this, demonstrations of successful behaviour are used to train policies to imitate the expert [22]. One approach of Imitation Learning is Behavior Cloning(BC), which learns a policy through supervised learning to mimic the demonstration state-action pairs. BC has been applied successfully in autonomous driving [19] [20], quad-copter navigation [21], locomotion [17] [23]. But it suffers from problems related to distributional drift and outside the space

of demonstration data. Dataset Aggregation (DAGGER) [24] mixes the learned and expert policies iteratively and augments the dataset to solve the accumulating error problem. It requires the expert to be available during training to provide additional feedback to the agent and makes it difficult to use. Pure Imitation Learning or BC are limited and cannot exceed the capabilities of the demonstrator since they lack the notion of task performance and environment. They under-perform or perform similar to the demonstrations.

Inverse optimal control or Inverse Reinforcement Learning (IRL) [25] is other approach of Imitation learning in which agent learns the reward function from optimal demonstrations. Few achievements of IRL are navigation [26], autonomous helicopter flight [27] and manipulation [28]. Our focus is not just to use imitation learning rather use imitation learning for bootstrapping the process of reinforcement learning. This bootstrapping helps to overcome exploration challenges of RL and RL learns the policy to improve based on actual task objective.

*4) Leveraging Reinforcement Learning with demonstrations:* In this research, we are planning to combine imitation learning with Reinforcement learning. The Imitation learning bootstraps the initial learning process and gives a idea of task objective this overcomes the exploration challenges and the need to explore the whole task space, while RL fine tuning allows the policy to improve based on actual task objective.

Methods based in dynamic movement primitives (DMPs) [13] [14] [17] have been used to effectively combine demonstrations and RL to enable fast learning. Most of the methods use trajectory-centeric policy representations, which is well suited for imitation but not enable feedback on rich sensory inputs. Such methods have been applied to some dexterous manipulation tasks [18], the tasks are primitive and simpler. Using expressive function approximators allow for complex, non linear ways to use sensory feedback, making them well-suited to dexterous manipulation. [1] [2] have successfully demonstrated the work on Leveraging Reinforcement Learning with demonstration.

*5) Multi Goal Reinforcement Learning:* Instead of the standard RL setting, we train agents with parameterised goals, which lead to more general policies and have recently been shown to make learning with sparse rewards easier. Goals describe the task we expect the agent to perform in the given episode, in our case they specify the described positions of all objects [9]. We sample the desired positions of all objects. We sample the goal g at the beginning of every episode. The function approximator takes the current goal as an additional input.

## III. Problem Description

In this project, we intend to accomplish tasks in a continuous action space using controllers generated by Reinforcement Learning Algorithms. The task could range from pushing Fig.1 or sliding an object Fig.2 to balancing an inverted pendulum. Our assumption is that the dynamic model of the robot is unknown to us.

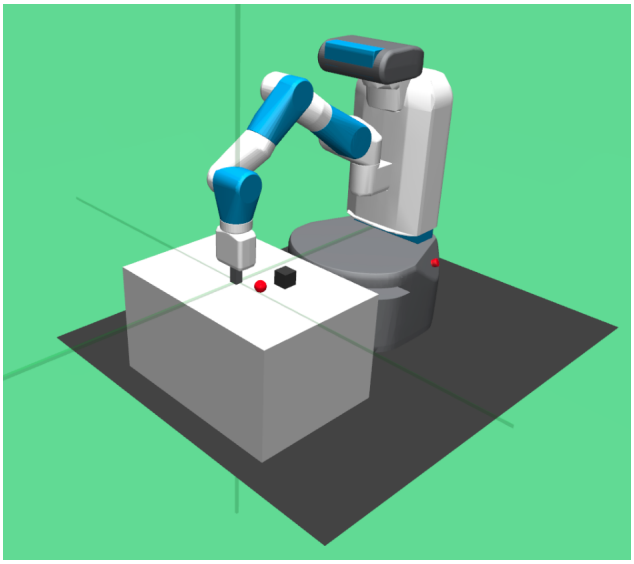

Fig. 1. FetchPush Environment with object

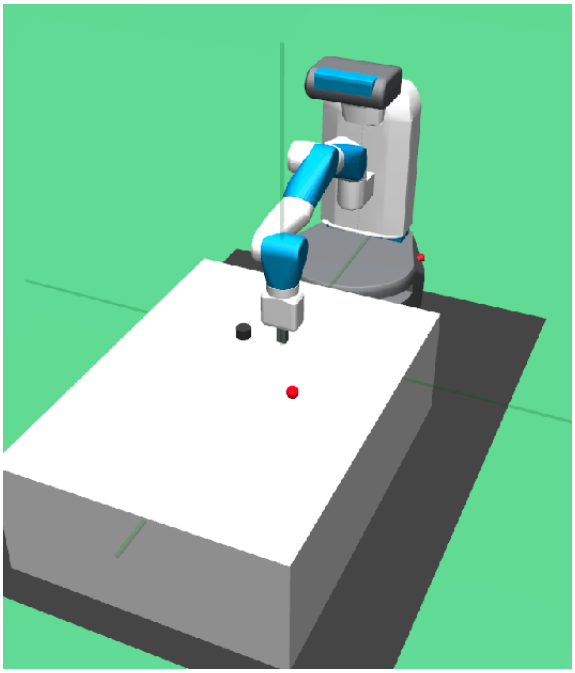

Fig. 2. FetchSlide Environment with puck

## IV. Methodology

Before choosing the algorithm, we need to get the bird's eye view of the dynamics of the environment. Reinforcement Learning involves the robot being at a particular state and taking an action to get some reward from the environment. The states and the actions of the robots in our case are not discrete. Every action of the robot leads to a new state and the environment calculates the reward based on whether the object has reached the destination. Each action gets a reward of 0 if it results in the object reaching the desired goal position and -1 otherwise.

We try to solve the above mentioned problem statement using Deep Reinforcement Learning (DRL) technique. We use Deep Deterministic Policy Gradient (DDPG), the popular DRL technique that works well with continuous spaces, and we use it along with Hindsight Experience Replay(HER). The methods along with their advantages and disadvantages have been explained in the following subsections.

### *A. Deep Deterministic Poilicy Gradient (DDPG)*

DDPG is a popular technique in Deep Reinforcement Learning developed by Lilicrap et. al, which can be used for high-dimensional, continuous action spaces as explained in [12]. This algorithm is an off-policy technique and uses two actor-critic neural networks.

The first actor-critic network, or the target network, generates a set of states, actions, rewards and next states and stores in a memory called Replay Buffer. The input layer of this network is the state of the robot and the output is the next action. Some noise is added to the generated actions in order to avoid repetitive actions and also help the robot explore different options. Once we have sufficient data, we sample from these and feed it to the second actor-critic network.

The second actor-critic network uses the sampled batch of the state, action, reward, new state and tries to find the optimal policy to carry out the task successfully. The critic finds the maximum value from all the available actions by minimizing the loss function which is the least squared error with the target critic value. Using this maximum value it finds the optimal policy. The pseudo code from the original paper [12] is mentioned in Fig[4]

The advantages of DDPG:

1) It is model-free
2) Knowledge about dynamics of the robot is not necessary
3) Works well with high dimensional continuous action spaces
4) It takes care of the singularity configuration of the robot
5) Neural network helps in getting multiple sets of state, action, reward very easily.

The problems with DDPG:

1) Requires high sample space

However the main problem with using DDPG alone is that it only leads to failed cases in the beginning when we are in the beginning stage of learning the dynamics of the system. This results in more training time.

### *B. Hindsight Experience Replay (HER)*

In order to mitigate the problem mentioned in the previous subsection, we use Hindsight Experience Replay Fig[3] algorithm to train the Fetch robot arm in a model-free manner. The motivation behind this algorithm is to emulate the human ability to learn from a unfavourable incident as much as from a favourable one. This algorithm is usually combined with an off-policy algorithm, which is DDPG in our case.

The rewarding structure of the algorithm is binary. This means a robot will receive a reward of -1 until it successfully achieves its objective which will lead to earning a reward of

**Algorithm 1** Hindsight Experience Replay (HER)

**Given:**
- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer $R$
**for** episode = 1, $M$ **do**
  Sample a goal $g$ and an initial state $s_0$.
  **for** $t = 0, T-1$ **do**
    Sample an action $a_t$ using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation
    Execute the action $a_t$ and observe a new state $s_{t+1}$
  **end for**
  **for** $t = 0, T-1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g,\ a_t,\ r_t,\ s_{t+1}||g)$ in $R$ ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}$(**current episode**)
    **for** $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g',\ a_t,\ r',\ s_{t+1}||g')$ in $R$ ▷ HER
    **end for**
  **end for**
  **for** $t = 1, N$ **do**
    Sample a minibatch $B$ from the replay buffer $R$
    Perform one step of optimization using $\mathbb{A}$ and minibatch $B$
  **end for**
**end for**

Fig. 3. Hindsight Experience Replay Pseudo Code

**Algorithm 1** DDPG algorithm

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

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

    Update the target networks:

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

  **end for**
**end for**

Fig. 4. DDPG Pseudo Code

a positive value. This algorithm also helps to convert a failed episode into a successful one. In our case, while training when we give a desired goal location to the robot, it might not always reach there because of less matured weights of the neural networks. The experience replay feature of this algorithm will help to revisit this episode, allow the model to change the desired goal of this episode to the goal which was achieved. This is extremely advantageous as it allows for generalizing a policy for multi-goals.

While training, after each episode, the experience replay buffer stores every time step transition. This buffer can be replayed using DDPG to train multi-goal episodes using the stored intermediate goals. This is graphically depicted in Fig. 9.

## V. Training Environment

Experimentation in Reinforcement Learning is constrained to the existing environments available for training [31]. OpenAI Gym provides standardized environments as a test bed

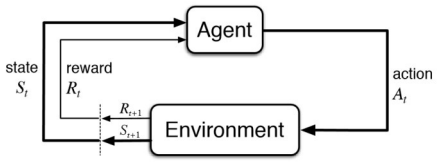


Fig. 5. Framework

for trying out a variety of algorithms. Gym provides multiple environments which run on different physics engines. A few examples are Fetch Robot, Lunar Lander, Inverted Pendulum, and Hopper. This project performs investigation on the Fetch and the Pendulum environment. An environment provides the basic necessities to accomplish a task. For example, four environments are offered for the Fetch robot

- To pick and place a puck to the desired location on a table(FetchPickAndPlace-v1)
- To push a puck to the desired location on a table(FetchPush-v1)
- To move the end effector of the robot to a desired location(FetchReach-v1)
- To slide a puck to the desired location on the table(FetchSlide-v1)

Similarly, the Pendulum-v0 offers an environment to swing a pendulum and make it stay upright. This is a classic problem in the control literature. Both Fetch and Pendulum environments run on the Mujoco physics engine.

These robots(agents) communicate with their environments as shown in 5. The agent performs an action and the environment returns the log of the transition from the former state to the latter one. The environment also returns a reward for performing the action.

An experience tuple is stored in every episode of the training which contains all the information which is necessary for training. The content of this experience tuple is

- state($S$) : Current state
- actions($A$) : The action that the robot performs
- rewards($R$) : rewards are specific to environments
- next-state($S$‘) : The state after taking the action
- done : 1 or 0, if the goal is achieved or not, or if the episode has ended or not
- Experience Tuple : ($S$, $A$, $R$, $S$‘, done)

The main advantage of using these established environments is that they provide the observations and the inbuilt facility to perform actions on the robot. Whenever an action is performed, the environment outputs what it observes about its surroundings or the robot itself from the new state. The observation space of both the environments is explained below.

- The observation space of the Pendulum environment is of size [3,1] and contains X position of its end effector, Y position of the end effector and the angular velocity of the pendulum.
- The observation space of the Fetch robot environment is of size [25,3] and contains the joint angles, the joint angular velocities, the position of the object or the end effector(in the case when object is absent), The linear and angular velocity of the end effector, and the desired goal position.

The rewards of an action are used to find the Q value of the critic framework using the Bellman equation 1. This equation calculates the sum of the reward for the performed action and the maximum possible reward in the new state multiplied by a discount factor. The discount factor gamma gives more weight to the possible Q values in the near future and lesser weights to the values in the farther future. In the Bellman Equation, $s'$ is the state we ended up in and $a'$ is the action resulting in maximum Q value from this state.

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi\left(a'|s'\right) q_\pi\left(s', a'\right) \quad (1)$$

The rewards of both the environments are explained below.

- The rewarding strategy of the Pendulum environment follows the equation-

$$\mathcal{R}_s^a = -(\theta^2 + 0.1 * \dot{\theta}^2 + 0.001 * a^2) \quad (2)$$

  The angle $\theta$ is normalized between $-\pi$ and $\pi$. Hence the minimum and maximum rewards are -16.27 and 0. In short, the goal is to remain upright(at zero angle) with the least rotational velocity.
- The rewards of the Fetch environment are sparse i.e every action gets either -1 or 0 as the reward. If the next state is the goal state, then the reward for the action is 0, otherwise it is -1. This strategy is very popular for robotics applications.

The action space of each environment is also different and it is determined by the degrees of freedom of the robot.

- The action space of the Pendulum robot is of size 1. It basically contains the torque used to modify the single joint of the pendulum.
- The action space of the Fetch robot environment contains 4 elements i.e the joint angles of the robot.

## VI. Experiments

For all experiments, we use PyTorch as our deep learning framework and use an Ubuntu 16 environment running on a Nvidia 1080Ti GPU. We perform several experiments and the parameters common to all the experiments are mentioned in the table below. Experiment specific parameters are mentioned in the respective sections.

```
Initial State:  [0 0 0 0 0 1 1 0 0 1 1 1 1 1 1]
Goal:   [1 0 0 0 0 0 0 0 1 1 1 1 0 0 0]
State at step 0: [1 0 0 0 0 1 1 0 0 1 1 1 1 1 1]
State at step 1: [1 0 0 0 0 1 1 0 1 1 1 1 1 1 1]
State at step 2: [1 0 0 0 0 1 1 0 1 1 1 1 0 1 1]
State at step 3: [1 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1]
State at step 4: [1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 1]
State at step 5: [1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0]
State at step 6: [1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0]
Success!
Press enter...
```

Fig. 6. Bit Flipping Simulation

Common parameters used in the various experiments

| Parameter | Value |
|---|---|
| Epochs | 7000 |
| Timesteps | 100 |
| Buffer Size | 1e6 |
| Optimizer | Adam |
| Actor Learning Rate | 0.0001 |
| Critic Learning Rate | 0.001 |
| Discount Factor (gamma) | 0.98 |
| Polyak weights | 0.999 |
| Batch Size | 128 |
| Test steps | 50 |

*1) Vanilla DDPG implementation:* We begin with a vanilla DDPG implementation. Here we just create an actor-critic network and perform their respective updates as per the DDPG algorithm. The network architecture for both actor and critic is a 4-layer fully connected network with 256 hidden units each. The activation function is ReLU for all layers but the output where it is tanh. The input layer for the actor network took in the observation vector (shape 25) and the goal vector (shape 3) and predicted the actions vector (shape 4). The critic network took as inputs observation (shape 25), goals (shape 3) and actions vector (shape 4). The training ran for 5000 epochs, within every epoch there are 800 timesteps when the environment takes a step and computes the reward. The actor critic networks update loop is run for 50 iterations. The target networks were updated by polyak averaging once every epoch. The code was tested on OpenAI gym Inverted Pendulum Environment and Mujoco Fetch Environments. The simulation is shown in Fig[6] scores vs episodes plot for pendulum is shown in Fig[7], Success rate vs Episodes in Fig[8]

*2) DDPG with HER:* In this variant, we implemented Hindsight Experience Replay along with the vanilla DDPG algorithm implemented in the previous experiment. We use a hindsight replay buffer and we use the 'Final' sampling strategy to sample goals and put in the HER buffer. The final strategy involves the following : after each episode, we take all transitions of the episode and copy them in the HER buffer. Then we replace the desired goal of all states but the last with the achieved goal of the last timestep. We then re-compute rewards for each timestep accordingly and add the HER buffer into the standard replay buffer. In this version we perform

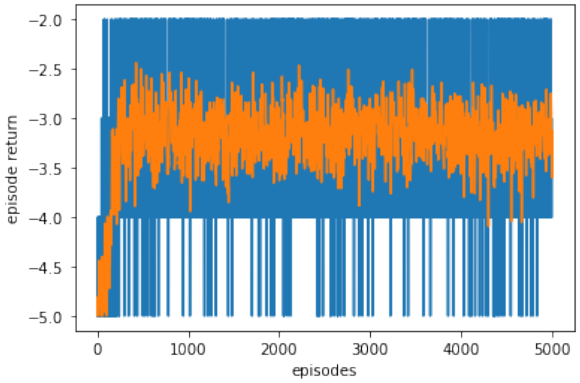


Fig. 7. Bit Flipping Environment Returns vs Results

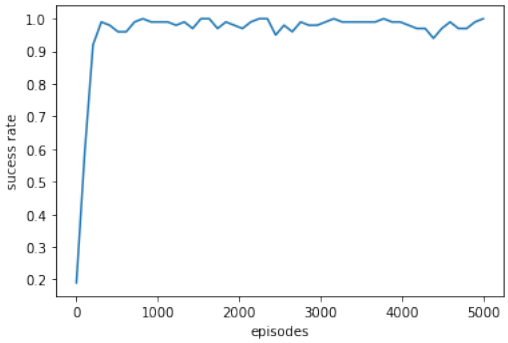


Fig. 8. Bit Flipping Environment Success rate vs Episodes

two variants: 1) we always sample episode transitions into the HER buffer 2) we sample it only when the Fetch robot has touched the puck. The former is how it is typically done in HER tasks. The latter was done since if the robot does not touch the puck at all and still we add it as a valid goal then we are encouraging the robot to not touch the puck at all. We observe improvements using the latter and are documented in the results section. We also implement it successfully on a custom built bit-flipping environment.

In Bit Flipping Environment we randomly sample two one dimensional arrays of n-bits(0 or 1) as Initial and Goal state. For every time step we take an action of flipping a bit at index i = n from the state to match the Goal state. Till the Initial state is not matched with the Goal state the environment will give a reward of -1 and when it is matched it will give a reward of 0 (Sparse Reward). Standard Algorithms like DQN are bound to fail in this environment for $n \geqslant 15$ bits because they will experience a reward of -1 till the both states are matched. The solution for solving this problem is reward engineering. But this will not help much when $n \geqslant 40$ bits. The best solution found for this is Hindsight experience replay. We have implemented DQN + HER and the results are shown in results section.

*3) Varying the type of noise:* The actions carried out by the agent is intentionally corrupted with some element of noise. This is done to encourage better explorations. We have tried

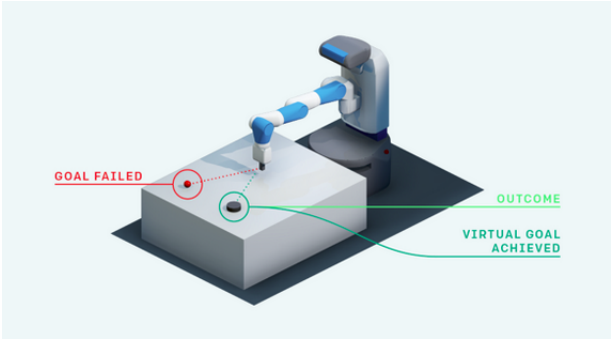


Fig. 9. Hindsight Experience Replay [32]

two types of noise : Gaussian noise and OU noise [29].

*4) Input Normalization:* Neural networks typically face issues when the inputs have different magnitudes and hence it becomes imperative to scale them properly. Hence we rescale the inputs to the actor and critic networks such that they have zero mean and unit standard deviation. We then clip them to the range [5,5]. Since the inputs here are dynamic, we ensure that the mean and standard deviations are recomputed every time there are new inputs in the training.

*5) Initialization using pre-trained weights:* Fetch environments are very complex in nature due to the enormous number of states and possible actions to take. This makes convergence of algorithms on these environments a very difficult task. It has been observed that vanilla DDPG hardly converge on such environments. Along with HER, there are several optimizations required to obtain satisfactory progress. One reason for the slow convergence is the starting initial weights for the robot. We observed that starting with random initialization show no change in rewards even after 5000 epochs. Hence we use a pre-trained model obtained from [30]. We use these weights as initialization and train it from there.

## VII. Results and Discussion

This section presents the results we obtained and our observations on the various experiments performed. For each experiment, we saved model weights and loaded them during inference time. We observe that our DDPG implementation worked perfectly fine on the inverted pendulum environment. The resulting video is seen in https://bit.ly/2RKzihA . The pendulum is able to balance itself vertically as seen in Fig[11] . Fig[12] shows the plot of scores improving with every episodes. Due to negative rewarding the sum of scores in an episode is always negative and it improves with episodes. However, when we ran the same vanilla DDPG code for the Fetch environments, the rewards were more or less static for more than 5000 epochs and the algorithm did not converge on the desired weights. On implementing DDPG with HER, we observe that the HER samples were sampled 50% of the times, each time. It has worked perfectly on the bit-flipping environment as seen in Fig[6] the simulation of transition from start state to goal state, Fig[7] shows the score return in an episode plotted with episodes the orange plot is a mean of

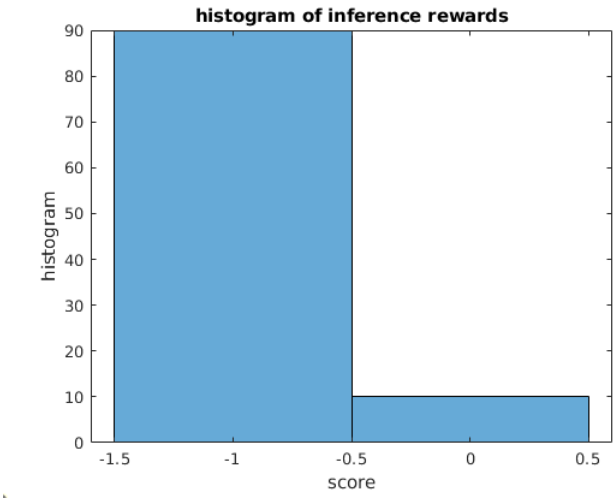


Fig. 10. This figure shows the histogram of the rewards obtained. It can be seen that out of 100 samples 10% of them are successful

returns, Fig[8] shows the success rate vs episodes. When tested on the Fetch Push and Fetch Slide environments and using pretrained weights which had included all optimizations which the original authors had done, we were able to get almost 100% success rates as seen in https://bit.ly/35fR5Rk . But the results obtained using basic DDPG and HER algorithm and training from scratch had very low success rates. We observed that if we sample transitions into HER buffer every time, then it results in deteriorated performance. The reason is that the robot does not touch the puck every time and if we even sample this transition into HER then we would be encouraging the robot to not touch the puck at all. Hence we modified the HER code such that it modifies the HER buffer only when the robot touches the puck. This way, the pseudo-goal in HER makes physical sense as well. We observe that over 5000 epochs, the robot touches the puck 10% of the time. We observe that using OU noise performed better than using Gaussian noise since in the latter the robot almost always takes a similar action irrespective of the position of the puck and the goal. We also observed that a combination of input normalization and training using pretrained weights as initialization gave an improved performance and we were able to obtain about 10% success rate with the same as can be seen in Fig. 10 and the video is at https://bit.ly/35fKMxc . This tells us the importance of a proper initialization of weights.

The original authors of HER from OpenAI themselves could not obtain high success rates with Fetch Slide using the final sampling strategy and obtained sufficient accuracy with Fetch Push after 200 epochs with each epoch containing 50 cycles and each cycle running the policy for 16 episodes and 40 network updates. They implemented sophisticated multi-threading to make the training tractable and quick. Our implementation consumed a lot of computing power and the GPUs got exhausted after about 7000 epochs. Moreover, cross-environment transfer of algorithms which work in one

Fig. 11. Trained Policy taking actions and making Pendulum Inverted

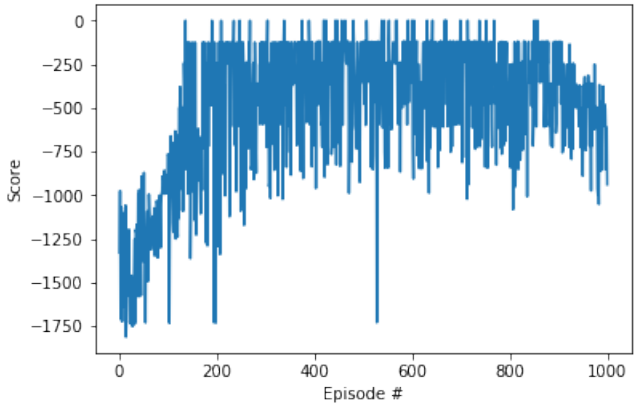


Fig. 12. Pendulum-V0 Score vs Episodes

environment to work in some other environment is an active area of research and requires skilled expertise in reinforcement learning.

## VIII. Timeline

| N.o | Tasks | Dates | Status |
|---|---|---|---|
| 1 | Software Environment setup ready | Oct 26 | Done |
| 2 | Overall understanding of RL algorithms | Nov 2 | Done |
| 3 | Understanding Actor Critic Methods | Nov 9 | Done |
| 4 | Implement DDPG+HER with baselines | Nov 16 | Done |
| 5 | Implement Vanilla DDPG | Nov 23 | Done |
| 6 | Implement DDPG + HER | Nov 30 | Done |
| 6 | Variations and parameter tuning of algorithms | Dec 07 | Done |
| 8 | Documentation | Dec 11 | Done |

## IX. Conclusion and Future Work

In this work, we explored the state of the art reinforcement learning algorithms for continuous space action control. We successfully implemented vanilla DDPG and DDPG with HER. We have been able to successfully complete the tasks prescribed in environments like the inverted pendulum and bit flipping. The inverted pendulum was trained with vanilla DDPG. This trained network achieved success in every inference episode. Additionally, We used the pre-trained weights in our network of the Fetch environment for DDPG and HER and achieved success in all the inference episodes. We have tried several variants on the basic DDPG with HER combination and have been able to obtain 10% success rate for FetchPush after training from scratch. We come up with several observations and potential pitfalls which have been mentioned in the results section and would be useful for someone looking to implement the algorithms on such complex environments. This project was more focused on learning Deep Reinforcement Learning concepts and hence we made an effort to implement the algorithms from scratch. Going forward, one could focus on improved sampling strategies like future sampling, episodic sampling. One could also implement multi-threading for improved performance and hence more training. Prioritized experience replay is another technique which could be explored and merged with DDPG and HER.

## X. Work Contribution

The work contribution of all four students are more or less the same as we are understanding the concept, pseudo code and implementing it all together.

The code of this paper can be found on this link : short-url.at/clqtQ

## References

[1] Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards, Matej and Hester, Todd and Scholz, Jonathan and Wang, Fumin and Pietquin, Olivier and Piot, Bilal and Heess, Nicolas and Roth, Thomas and Lampe, Thomas and Riedmiller, Martin, arXiv preprint arXiv:1707.08817, 2017

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

[3] Learning from Demonstrations for Real World Reinforcement Learning, Todd Hester and Matej Vecer and Olivier Pietquin and Marc Lanctot and Tom Schaul and Bilal Piot and Andrew Sendonaris and Gabriel Dulac-Arnold and Ian Osband and John Agapiou and Joel Z. Leibo and Audrunas Gruslys, ArXiv, 2017, abs/1704.03732

[4] Contact-invariant optimization for hand manipulation, Mordatch, Igor and Popović, Zoran and Todorov, Emanuel, Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, pages 137-144, 2012, Eurographics Association

[5] A direct method for trajectory optimization of rigid bodies through contact,Posa, Michael and Cantu, Cecilia and Tedrake, Russ, The International Journal of Robotics Research,33, pages :69–81, 2014, Sage Publications Sage UK: London, England

[6] Real-time behaviour synthesis for dynamic hand-manipulation, Kumar, Vikash and Tassa, Yuval and Erez, Tom and Todorov, Emanuel, 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 6808–6815, 2014, IEEE

[7] Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost, Henry Zhu and Abhishek Gupta and Aravind Rajeswaran and Sergey Levine and Vikash Kumar, CoRR, 2018, http://arxiv.org/abs/1810.06045, arXiv, https://dblp.org/rec/bib/journals/corr/abs-1810-06045, dblp computer science bibliography, https://dblp.org

[8] A generalized path integral control approach to reinforcement learning, Theodorou, Evangelos and Buchli, Jonas and Schaal, Stefan, Journal of machine learning research, 11 Nov 2010

[9] Hindsight experience replay, Andrychowicz, Marcin and Wolski, Filip and Ray, Alex and Schneider, Jonas and Fong, Rachel and Welinder, Peter and McGrew, Bob and Tobin, Josh and Abbeel, OpenAI Pieter and Zaremba, Wojciech, Advances in Neural Information Processing Systems, 2017

[10] Reinforcement Learning: An Introduction, R.S.Sutton and A.G. Barto, volume I and II

[11] Trust region policy optimization, Schulman, John and Levine, Sergey and Abbeel, Pieter and Jordan, Michael and Moritz, Philipp, International conference on machine learning, pages 1889–1897, 2015

[12] Continuous control with deep reinforcement learning, Lillicrap, Timothy P and Hunt, Jonathan J and Pritzel, Alexander and Heess, Nicolas and Erez, Tom and Tassa, Yuval and Silver, David and Wierstra, Daan, arXiv preprint arXiv:1509.02971, 2015

[13] Reinforcement learning of motor skills with policy gradients, Peters, Jan and Schaal, Stefan, Neural networks, volume 21, pages 682–697, 2008, publisher Elsevier

[14] Policy search for motor primitives in robotics, Kober, Jens and Peters, Jan R, Advances in neural information processing systems, pages 849–856, 2009

[15] Computational approaches to motor learning by imitation, Schaal, Stefan and Ijspeert, Auke and Billard, Aude, Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences, 358, number 1431, pages 537–547, 2003, The Royal Society

[16] Learning motor primitives for robotics, Kober, Jens and Peters, Jan, 2009 IEEE International Conference on Robotics and Automation, pages 2112-2118, 2009, IEEE

[17] Movement imitation with nonlinear dynamical systems in humanoid robots, Ijspeert, Auke Jan and Nakanishi, Jun and Schaal, Stefan, Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292), volume 2, 1398-1403, 2002, IEEE

[18] Learning robot in-hand manipulation with tactile features, Van Hoof, Herke and Hermans, Tucker and Neumann, Gerhard and Peters, Jan, 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), 121-127 2015, IEEE

[19] Alvinn: An autonomous land vehicle in a neural network, Pomerleau, Dean A, Advances in neural information processing systems, 305-313, 1989

[20] End to End Learning for Self-Driving Cars, Mariusz Bojarski and Davide Del Testa and Daniel Dworakowski and Bernhard Firner and Beat Flepp and Prasoon Goyal and Lawrence D. Jackel and Mathew Monfort and Urs Muller and Jiakai Zhang and Xin Zhang and Jake Zhao and Karol Zieba, 2016, 1604.07316, arXiv

[21] A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots, A. Giusti and J. Guzzi and D. C. Cireşan and F. He and J. P. Rodríguez and F. Fontana and M. Faessler and C. Forster and J. Schmidhuber and G. D. Caro and D. Scaramuzza and L. M. Gambardella, IEEE Robotics and Automation Letters, 2016, volume 1, 2, pages 661-667, July

[22] Learning from demonstration and adaptation of biped locomotion Nakanishi, Jun and Morimoto, Jun and Endo, Gen and Cheng, Gordon and Schaal, Stefan and Kawato, Mitsuo, Robotics and autonomous systems, volume 47, number 2-3, pages 79-91, 2004, Elsevier

[23] Learning locomotion over rough terrain using terrain templates, Kalakrishnan, Mrinal and Buchli, Jonas and Pastor, Peter and Schaal, Stefan, 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 167-172, 2009, IEEE

[24] A reduction of imitation learning and structured prediction to no-regret online learning, Ross, Stéphane and Gordon, Geoffrey and Bagnell, Drew, Proceedings of the fourteenth international conference on artificial intelligence and statistics, 627-635, 2011

[25] Algorithms for inverse reinforcement learning., Ng, Andrew Y and Russell, Stuart J and others, Icml, volume 1, 2000

[26] Maximum entropy inverse reinforcement learning, Ziebart, Brian D and Maas, Andrew and Bagnell, J Andrew and Dey, Anind K, 2008

[27] Apprenticeship learning via inverse reinforcement learning, Abbeel, Pieter and Ng, Andrew Y, Proceedings of the twenty-first international conference on Machine learning, 2004, ACM

[28] Guided cost learning: Deep inverse optimal control via policy optimization, Finn, Chelsea and Levine, Sergey and Abbeel, Pieter, International Conference on Machine Learning, pages 49-58, 2016

[29] Uhlenbeck, George E., and Leonard S. Ornstein. "On the theory of the Brownian motion." Physical review 36.5 (1930): 823.

[30] https://github.com/TianhongDai/hindsight-experience-replay

[31] Plappert, Matthias and Andrychowicz, Marcin and Ray, Alex and McGrew, Bob and Baker, Bowen and Powell, Glenn and Schneider, Jonas and Tobin, Josh and Chociej, Maciek and Welinder, Peter and others, Multi-goal reinforcement learning: Challenging robotics environments and request for research, arXiv preprint arXiv:1802.09464, 2018

[32] https://openai.com/blog/ingredients-for-robotics-research/