DH^2 : A Deep Reinforcement Learning Algorithm for Robotic Control Applications

Anusha Nemilidinne
Department of Computer Science
University of Houston
anemilidinne@uh.edu

Teja Salapu
Department of Computer Science
University of Houston
tsalapu@uh.edu

Abstract

One of the major challenges for Reinforcement learning in training a Robotic Arm is sparse rewards and complicated Reward engineering. Depending on the degree of freedom for the Robotic joints, we end up with a huge Action space /Action dimensionality.

Our method is built over Deep Deterministic Policy Gradient and Hindsight Experience Replay. To overcome the problem of sparse rewards, we sampled data from an additional buffer containing human Demonstrations data and virtual goals data by Hindsight Experience Replay, which drastically speedups the Robotic tasks. We run our Experiments on tasks like Push, Slide, Pick and place in simulated environment and compare our method with baseline DDPG. Based on the human intuition, we combine our prior knowledge of the environment with trial and error to learn a task. Likewise, we are sampling from Human demonstrations and virtual goals to manipulate a robotic task.

1 Introduction

In the recent Era, we have seen many advances in the field of Reinforcement Learning combined with Neural networks especially in the field of gaming, such as Alpha Go [2] which defeated world best Go player and playing Atari games [1]. Consider the game of pong [3], even though we get good results after many epochs of training but initially, the network faces sparse reward problem. When we try to train a mountain car to reach a flag, it does not know that it needs velocity to speed up and reach the flag. So the program terminates after successive steps of receiving negative rewards. This resulted in Reward engineering concept [3] to shorten agents learning time. The necessity of Reward shaping hinders the RL in Robotic tasks learning in different environments.

In Robotic tasks like Reaching or sliding an object to a specific goal location, we have a huge action space. Consider a Robot Arm - 3 joints with 7 degree of freedom results in 3^7 action states. This number of Action space increases more when we consider complex tasks like Fetch and place or stacking a block. These tasks require multiple steps to accomplish a task like reaching a goal because in case of long horizon and high dimension, its not feasible to assign a reward for just reaching the goal. Andrychowicz et al [22] proposed Hindsight experience replay to add virtual goals into the buffer and reward them even though they are not the actual goal to encourage Non-Zero rewards based on human intuition where we learn from failure i.e when playing a football if we kick the ball slightly to the left of the goal, we learn from the experience. HER can be applied to any off policy method.

For continuous control tasks like this DQN cant be applied because of the dimensionality and as it only relies on finding the action that maximizes the reward. Deterministic policy gradient [4] was applied for continuous spaces but naive application of this actor-critic method with neural function approximators is unstable for challenging problems.

Silver et al [4] proposed Deep Deterministic policy gradient for Continuous control that is stable because of the innovations 1. the network is trained off-policy with samples from a replay buffer to minimize correlations between samples 2. the network is trained with a target Q network to give consistent targets during temporal difference backups.

One significant challenge is slow training in the above approach. The Key feature of our method is we employ additional buffers for Human Demonstration data and virtual goal data i.e Hindsight experience replay-additional goals which are sampled to the buffer to encourage Non-zero rewards and faster training as discussed in the Implementation section.

2 Related Work

Learning algorithms for decision making problems such as robotics can be broadly classified into two types: imitation learning and reinforcement learning(RL). Imitation Learning can also be called as learning from demonstrations since the agent receives demonstrations from an expert and attempts to solve the task by copying the expert's behavior. Whereas in RL, the agent tries to maximise the expected future rewards by interacting with the environment.

Reinforcement Learning: Reinforcement Learning has been used in the field of gaming from ages but it lacks behind in the field of robotics because of complexity and huge action space, However, autonomy in robotic tasks can only be accomplished using this methodology. Reinforcement learning is all about taking a suitable action that maximizes the reward for a particular task. It differs from supervised learning in a way that in supervised learning the training data contains the answer key so the model is trained with correct answers but in reinforcement learning, there are no answers instead the agent is given a reward for every action it takes. In RL, agent learns from experience. Through RL, agents have learned to play tennis [16], swing up a cartpole, and balance a unicycle [17]. There has been huge success in applying RL to Atari games using deep RL. Deep RL uses function approximators like neural networks to learn control from either raw pixels or through experience buffer. Several off-policy RL algorithms like Deep Q-Networks (DQN) and Deep Deterministic Policy Gradients(DDPG) has achieved huge success in the field of robotics like manipulation tasks [18], grasping [19], and opening a door [20].

Imitation Learning: In imitation learning, the agent receives a series of illustrations from an expert. It is believed to learn a policy that generalizes well to unseen states. Some of the problems where imitation learning can be used include helicopter flight through apprenticeship learning, learning how to put a ball in a cup, playing table tennis , performing human-like reaching motions like fetch, pick and place, sliding among others. Humans learn much faster by observing and imitating the behavior of others. That we provide labelled instances to the agent. This method is called learning from demonstrations(LfD). Most common forms of imitation learning are behavior cloning and inverse reinforcement learning.

Behavioral Cloning(BC)is a kind of supervised learning, where the policy is learned from demonstration state-action pairs provided by domain experts. It directly maps states to actions. The agent is capable of learning through demonstrations without having to interact with the environment. BC can only be applied to scenarios where actions of the demonstrator are available. There are scenarios where we do not have knowledge of internal control signals that the expert used instead we just see the outcome of the actions [21]. BC has been applied in autonomous driving [5], quadcopter navigation [6], locomotion [7]. One limitation of BC is that it does not take into account the task or environment.

Inverse Reinforcement Learning(IRL) algorithms learn a reward function through demonstrations from experts [8], [9]. This reward function can then be used to learn a policy using the normal reinforcement learning [10]. But some IRL methods alternate between forward/standard and inverse reinforcement learning [11], [12]. However, using IRL methods on high-dimensional observations like images has been proven to be difficult. In RL, we try to hand-craft the rewards using reward shaping or reward engineering and it is often by trial and error or tweaking the rewards until a specific or desired behavior of the agent is achieved. So IRL tries to solve this problem of manually shaping or engineering the reward function for a task. One problem of IRL is that there might be many reward functions that the expert is trying to maximize using the demonstrations. IRL has been applied to navigation [13], automonous helicopter flight [14], and manipulation [15].

Deep RL off-policy algorithms like DQN and DDPG use experience replay buffer to store the experiences of an agent when it interacts with the environment. These experiences are randomly sampled from the buffer and used to train the neural networks. This technique of experience replay was used along with DQN agent to play Atari games (Mnih et al., 2015). Tasks like robotic arm movements contain sparse rewards which makes learning difficult. Sparse reward is one of the biggest challenges in RL. There are several types of experience replays each of which aims to solve a specific challenge. One such technique that deals with the problem of sparse and binary rewards is Hindsight Experience Replay (HER). This can be combined with any of the off-policy RL algorithm mentioned above. Our project combines the aspects of deep RL, imitation learning and HER to solve tasks with continuous action spaces like robotic arm movements.

3 Background

3.1 Reinforcement Learning

We consider a standard reinforcement learning setup where an agent interacts with the environment E. In this setup, we consider discrete time steps. The agent receives an observation x_t, performs an action a_t and receives a reward of r_t at time step t. Discrete action spaces mean there are finite possibilities to complete the task. We consider fully observable environment where the agent can see the entire state of environment at any given point of time. That is it does not require memory to make a decision. Agent's behavior is defined by policy π which maps states to probability distribution over actions $\pi : S \to P(A)$ [26].

The environment E might be stochastic which means there is a certain degree of uncertainity associated. Hence we model the environment as a Markov Decision Process(MDP) with state space S, action space A, and a reward function $r(s_t, a_t)$. The agent always chooses the action that maximizes expected discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$. Gamma is the discount factor which is in the range of $[0, 1]$. That is we give more weightage to the rewards gained in the inital stages of the game. Since the reward depends on the action chosen which in turn depends on the policy, rewards are stochastic. The primary goal in reinforcement learning is to maximize the expected discounted future reward from the start distribution $J = E_{r_i, s_i \sim E, a_i \sim \pi}[R_0]$. Many approaches have been proprosed to solve this problem. Q-learning is one such algorithm which attempts to find an action-value function $Q^\pi(s_t, a_t) = E_{r_i, s_i \sim E, a_i \sim \pi}[R_t | s_t, a_t] = E_{r_i, s_i \sim E[r_t + \gamma E_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]]}$. This equation is called Bellman equation which is solved recursively using dynamic programming paradigm. Q-learning is an off-policy algorithm which means the learned policy is different from the policy actually followed that is the optimal policy is independent of the actions followed. Q-learning uses a greedy policy $\mu(s) = argmax_a Q(s, a)$.

Recently, Q-learning has been adapted to neural networks which act as powerful function approximators to learn the target Q-values [27]. In order to scale Q-learning for large state-action spaces, two majors components have been introduced: replay buffer and a separate target network to estimate the target Q values. Our project makes use of these aspects in DDPG and HER which are explained in the following sections. Our project focuses on employing DDPG from Demonstrations with HER for fetch robotic arm movements using openAI gym.

3.2 DDPG

3.2.1 Policy Gradient

The goal of reinforcement learning is to find an optimal behavior strategy for the agent to obtain maximum rewards. The policy gradient methods model and optimize the policy directly. The policy is modeled using a parameterized function respect to $\theta, \pi_\theta(a|s)$. The value of the reward function depends on this policy and can thereby be optimized further by applying further algorithms.

The reward function is defined as:

$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a)$

where $d^\pi(s)$ is the stationary distribution of Markov chain for $d^\pi(s)$.

As there are infinite number of actions and states to estimate the values for, value based approaches are very expensive computationally in the continuous space.In generalized policy iteration, the policy improvement step $\arg\max_{a \in \mathcal{A}} Q^\pi(s, a)$ requires a full scan of the action space, suffering from the curse of dimensionality.

3.2.2 DDPG

Lillcrap et al., [23] introduced model-free off-policy actor-critic algorithm by stabilizing the Deterministic Policy Gradient using Experience replay and updating target network.At a high level,DDPG learns an action-value function(critic) by minimizing the Bellman error, while simultaneously learning a policy (actor) by directly maximising the estimated action-value function with respect to the parameters of the policy. DDPG extends to continuous space while learning a deterministic policy.Also it uses "soft" target updates, rather than directly copying the weights and have them slowly track the learned networks: $\theta \leftarrow \tau\theta + (1-\tau)\theta$ with $\tau \ll 1$. In this way target network values are constrained to change slowly thereby improving the stability of learning.

In order to deal with generalisation problem across environments with different scales of state value like a environment where robot's positions and velocities are given as input, DDPG uses batch normalization to fix this issue by normalizing every dimension across samples in one minibatch.

To do better exploration, an exploration policy is constructed by adding noise $\mathcal{N}$:

$$\mu'(s) = \mu_\theta(s) + \mathcal{N}$$

Our method is built over DDPG as it can be trained off-policy,so we sampled from two buffers demonstrations and virtual goals thereby encouraging non-zero rewards.

3.3 HER

We trained our model by employing HER additional buffer in our method. Hindsight Experience Replay[23] is based on how humans, when playing a football observe that kicking the ball slightly to the left missed the goal and learn from it but in HER we think of this as imaginary goal and add this transition into the buffer. This is greatly helpful in Robotic tasks because in training the program keeps terminating as a result of negative rewards. Even though this trajectory of additional goal may not directly help us to learn achieve final desired state, it definitely tells something about how to achieve the state ST. By adding these to experience replay we end up with some Non-zero rewards instead of -1 rewards without HER.

suppose our agent performs an episode of trying to reach goal state G from initial state S, but fails to do so and ends up in some state S at the end of the episode. We cache the trajectory into our replay buffer:

$$\{(S_0, G, a_0, r_0, S_1), (S_1, G, a_1, r_1, S_2), \dots, (S_n, G, a_n, r_n, S')\}$$

The idea in HER is to imagine that our goal has actually been S all along, and assigning a positive reward for that additional goal. So, in addition to caching the real trajectory as seen before, we also cache the following imaginary additional trajectory:

$$\{(S_0, S', a_0, r_0, S_1), (S_1, S', a_1, r_1, S_2), \dots, (S_n, S', a_n, r_n, S')\}$$

By introducing the imagined trajectories to our replay buffer, HER helps such that how bad the policy may be,it always have some positive rewards to learn from.

4 Methodology

DH^2 (DDPG+HER+Human demonstrations buffer)

Our improvised methodology employs actor-critic based Deep Deterministic Policy Gradient(DDPG) algorithm and Hindsight Experience Replay(HER) along with an additional human demonstrations buffer. In an actor-critic learning algorithm, we implement policy function independently of value function. The policy function is known as actor and the value function is referred to as critic. The actor produces an action for a given state of the environment and critic produces Temporal-Difference(TD) error signal for that state and a resultant reward. For the critic to estimate action-value function $Q(s, a)$, it needs the output of actor.

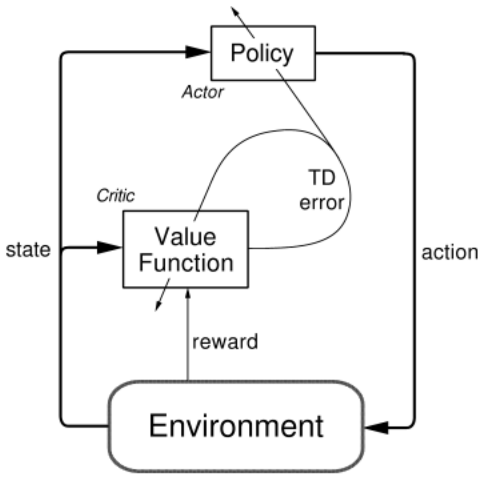

Figure 3:Actor-Critic Model

We implemented two neural networks, one for actor and other for critic. These networks compute action predictions for the current state and generate a TD error signal at each time step. The input of actor network is current state and output is a single real value representing an action from a continuous space. The critic's output is simply an estimated Q-value of the current state and of the action given by the actor. Updating the actor and critic neural network weights with the gradients obtained from TD error signal causes the algorithm to diverge. Hence, we are using another two target networks which are copies of actor and critic networks to generate targets for TD error computation there by regularizing the learning algorithm.

In DDPG, training and evaluating the policy and value function with thousands of temporally-correlated simulated trajectories leads to the introduction of enormous amounts of variance in the approximation of the true Q-function.So DDPG uses Experiences replay buffer to randomly sample experiences for learning in order to break up the temporal correlations within different training episodes. Here are the equations for the TD target y_i and loss function for the critic network:

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

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2) \tag{2}$$

In the above equation,N stands for size of the minibatch which is sampled from Replay buffer and i refers to i^{th} sample.The target y_i is computed from the sum of the immediate reward and the outputs of the target actor and critic networks with weights θ^{μ} and θ^{Q} respectively. Then the critic loss is computed w.r.t the output $Q(s_i, a_i|\theta^Q)$ of the critic network for the i^{th} sample.

4.1 Human demonstration buffer

Along with the experience replay buffer in standard DDPG algorithm, we added an additional buffer to store human demonstrations to improve learning rate. We sample N random minibatches from the human demonstration buffer and these samples are included in actor and critic update.

4.2 Imaginary goals-HER

We employed Hindsight Experience Replay in our project where after experiencing some episode $s_0, s_1, s_2, ..., s_T$, we store in the replay buffer every transition $s_t \rightarrow s_{t+1}$ not only with the original goal used for this episode but also with a subset of these additional imaginary goals. This ensures Non-Zero rewards and speed up in learning rate especially in Robotic tasks where we have long horizon tasks with the problem of sparse rewards and longer learning time.

5 Experiments

We experimented our method on 4 tasks 1. FetchReach 2. FetchPush 3. FetchPickandPlace 4. FetchSlide

5.1 Environments

We evaluated our method on OpenAI Gym Simulated environment on 7 degree of freedom Robotic Arm for the following tasks.

- **FetchReach**: In this task, the robot has to reach a desired target position.
- **FetchPush**: In this task,a box is placed on a table in front of the robot and the task is to move it to the target location on the table.
- **FetchPickandPlace**: In this task, it has to grasp the object and reach the target location
- **FetchSlide**: In this task, a puck is placed on a long slippery table and the target location is outside of the robot's reach so that it has to hit the puck with a force that it slides and stops in the appropriate place due to friction.

5.2 Environment Setup

We implement our project in python. We use pycharm IDE for coding. The following are the software requirements necessary to successfully execute the project.

1. python 3.5.2
2. openai-gym
3. mujoco200
4. mjpro131
5. pytorch-1.0.0
6. mpi4py

5.3 Training Details

To train our model, we use Adam optimiser. Actor learning rate and critic learning rate are 0.0001 and 0.001 respectively. τ as 0.001. $\gamma = 0.99$. Buffer size = 100000. Batch size = 32. $\epsilon = 1$ which is the exploration factor and then is decreased by a factor of 100000 as the actor learns the tasks.

5.4 Neural Network Architecture

We use a 4 layer network with 512 hidden units per layer for both actor and critic networks. We use 'relu' and 'sigmoid'. activation functions for the outputs. Since the observations are represented using low dimensional feature vector like positions and velocities which have different physical units, it might be difficult for the networks to learn effectively and makes tuning of hyperparameters difficult. Hence, we use batch normalization where in each feature is normalised across all the samples in a minibatch to have unit mean and variance.

5.5 Results

Figure 1 shows the graphs for success rates of different tasks using the experimental setup and network architecture described in the previous sections.

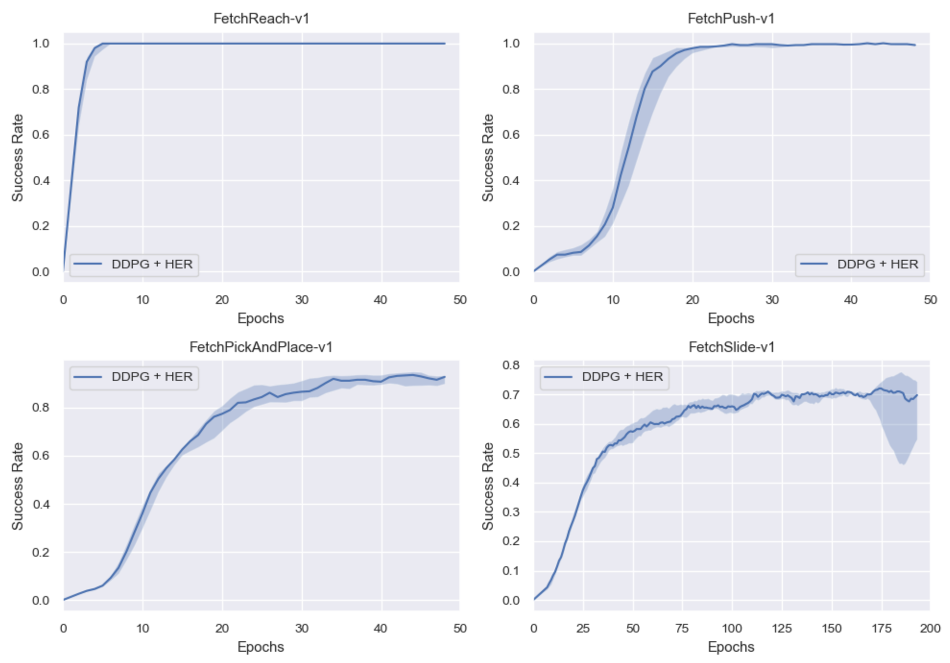

Figure 1: Success rates for the four robotic arm tasks

Figure 2 shows the frames for the tasks mentioned above.

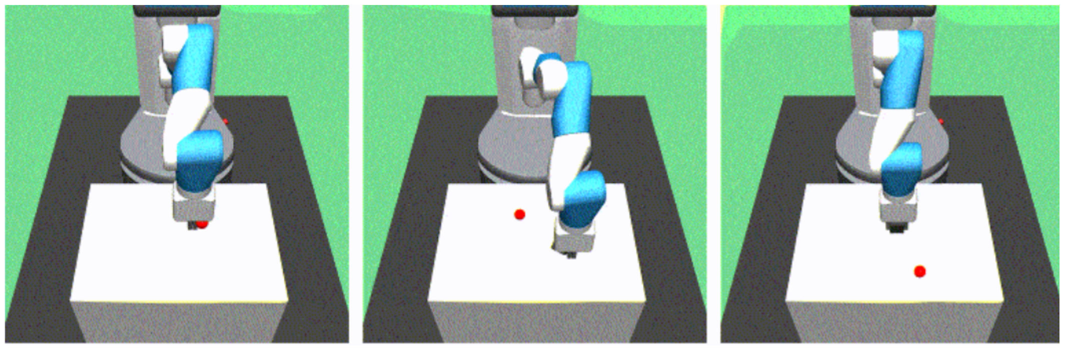

Figure 2: Demo of the tasks

Table 1: Comparison of success rates for our method with baselines

Tasks	our method	Baseline (DDPG + HER)
FetchReach	0.99	0.86
FetchPush	0.99	0.86
FetchPickandPlace	0.86	0.64
FetchSlide	0.72	0.67

From the Table 1, we can say that our method performed better than the baseline method or pure DDPG. The baseline method performs poorly in the pick and place task because of sparse reward settings as picking a block with random actions is extremely unlikely. Hence employing a human demonstration buffer greatly increases the accuracy of our method. Therefore, combining DDPG with HER along with human demonstration buffer, which is a way to provide prior knowledge, results in increase in learning rate in the early stage of training because of non-zero rewards thereby outperforming the baseline method.

6 Conclusion

We presented a methodology for increasing Non-zero rewards and learning rate using Actor-critic networks along with demonstration samples and additional goals from HER. A major challenge is sample efficiency and availability of demonstrations for complicated tasks. In future we aim to extend this to employ prioritized sampling from human demonstrations data. Also to train a synthetic Neural network for learning Human-readable plans from real-world demonstrations as proposed by Nvidia[24]

References

[1].Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M.,Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning.

[2]. David Silver*, Julian Schrittwieser*, Karen Simonyan*, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, Demis Hassabis. (2016). Mastering the Game of Go without Human Knowledge

[3]. Makarov, I., Kashin, A., Korinevskaya, A.(2017) Learning to play pong video game via deep reinforcement learning.

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

[5] D. A. Pomerleau. (1989) Alvinn: An autonomous land vehicle in a neural network."

[6] A. Giusti et al.,(2015). A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots,.

[7] M. Kalakrishnan et al.,(2009). Learning Locomotion over Rough Terrain using Terrain Templates,

[8] P. Abbeel and A. Y. Ng.(2004). Apprenticeship learning via inverse reinforcement learning,

[9] S. Levine, Z. Popovic, and V. Koltun,(2011). Nonlinear inverse reinforcement learning with gaussian processes,.

[10]N. D. Ratliff, J. A. Bagnell, and M. Zinkevich.(2006) Maximum margin planning,

[11] C. Finn, S. Levine, and P. Abbeel,(2016). Guided cost learning: Deep inverse optimal control via policy optimization,

[12] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal.(2013) Learning objective functions for manipulation,.

[13] B. D. Ziebart et al.,(2008). Maximum Entropy Inverse Reinforcement Learning. in AAAI Conference on Artificial Intelligence".

[14] P. Abbeel and A. Y. Ng.(2004). Apprenticeship learning via inverse reinforcement learning,

[15] C. Finn, S. Levine, and P. Abbeel.(2016). Guided Cost Learning: Deep Inverse Optimal Control via

[16] J. Peters, K. Mulling, and Y. Alt un.(2010). Relative Entropy Policy Search,

[17] M. P. Deisenroth and C. E. Rasmussen,(2011). Pilco: A model-based and data-efficient approach to policy search,.

[18] S. Levine et al.,(2015) End-to-end training of deep visuomotor policies,.

[19] L. Pinto and A. Gupta,(2015) Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours,

[20] S. Gu et al.,(2016). Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates,.

[21] F. Torabi, G. Warrnell, and P. Stone.(2018). Behavioral cloning from observation

[22] M. Andrychowicz et al., Hindsight experience replay, in Advances in neural information processing systems, 2017.

[23]Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess,Tom Erez, Yuval Tassa, David Silver & Daan Wierstra.(2016).CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

[24]Jonathan Tremblay, Thang To, Artem Molchanov, Stephen Tyree, Jan Kautz, Stan Birchfield.(2018).Synthetically Trained Neural Networks for Learning Human-Readable Plans from Real-World Demonstrations

[26] Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. International Conference on Learning Representations (ICLR), 2016.

[27] Yuxi Li. Deep Reinforcement Learning: An Overview.arXiv:1701.07274, 2017