Technical Report

IRI-TR-19-01

Learning cloth manipulation with demonstrations

Rishabh Jangir
Carme Torras
Guillem Alenyà

February, 2019

Institut de Robòtica i Informàtica Industrial

Abstract

Recent advances in Deep Reinforcement learning and computational capabilities of GPUs have led to variety of research being conducted in the learning side of robotics. The main aim being that of making autonomous robots that are capable of learning how to solve a task on their own with minimal requirement for engineering on the planning, vision, or control side. Efforts have been made to learn the manipulation of rigid objects through the help of human demonstrations, specifically in the tasks such as stacking of multiple blocks on top of each other, inserting a pin into a hole, etc. These Deep RL algorithms successfully learn how to complete a task involving the manipulation of rigid objects, but autonomous manipulation of textile objects such as clothes through Deep RL algorithms is still not being studied in the community.
The main objectives of this work involve, 1) implementing the state of the art Deep RL algorithms for rigid object manipulation and getting a deep understanding of the working of these various algorithms, 2) Creating an open-source simulation environment for simulating textile objects such as clothes, 3) Designing Deep RL algorithms for learning autonomous manipulation of textile objects through demonstrations.

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain

Tel (fax): +34 93 401 5750 (5751)
`http://www.iri.upc.edu`

Contents

Chapter 1

Rigid Object Manipulation

1.1 Motivation and Objectives

Reinforcement learning (RL) combined with neural networks has recently led to a wide range of successes in learning policies for sequential decision-making problems. This includes simulated environments, such as playing Atari games [1], and defeating the best human player at the game of Go [2], as well as robotic tasks such as helicopter control [3], hitting a baseball [4], screwing a cap onto a bottle [5], or door opening [6].

However, a common challenge, especially for robotics, is the need to engineer a reward function that not only reflects the task at hand but is also carefully shaped [7] to guide the policy optimization. Robotics researchers have used cost functions consisting of complicated terms which need to be carefully weighted to form the reward function in order to train a policy to perform a specific robotics task. The necessity of cost engineering limits the applicability of RL in the real world because it requires both RL expertise and domain-specific knowledge. Moreover, it is not applicable in situations where we do not know what admissible behaviour may look like. It is therefore of great practical relevance to develop algorithms which can learn from unshaped reward signals, e.g. a binary signal indicating successful task completion.

1.2 Algorithms studied and implemented

1.2.1 Hindsight Experience Replay

The technique of Hindsight Experience Replay or HER [8] allows us create algorithms that can learn through sparse reward formulations when combined with any off-policy RL algorithm. It is applicable whenever there are multiple goals which can be achieved, e.g. achieving each state of the system may be treated as a separate goal. Not only does HER improve the sample efficiency in this setting, but more importantly, it makes learning possible even if the reward signal is sparse and binary. Our approach is based on training universal policies which take as input not only the current state, but also a goal state. HER allows training policies which push, slide and pick-and-place objects with a robotic arm to the specified positions in a simulation while the vanilla RL algorithm fails to solve these tasks.

1.2.2 Overcoming exploration in HER with demonstrations

The HER algorithm works fine, but it is fairly slow to converge as it requires a lot of time and interactions with the environment. Using demonstrations to overcome the exploration problem can help in successfully learning to perform long-horizon, multi-step robotics tasks with continuous control such as stacking blocks with a robot arm. The method builds on top of Deep Deterministic Policy Gradients and Hindsight Experience Replay, and provides an order of magnitude of speedup over RL on simulated robotics tasks. It is simple to implement and makes only the additional assumption that we can collect a small set of demonstrations. Furthermore, this method is able to solve tasks not solvable by either RL or behavior cloning alone, and often ends up outperforming the demonstrator policy.

1.3 Simulation Frameworks used

Physics based simulation engines provide a good platform to test and train Deep RL algorithms in Robotics. The constant trial and error methodology involved in RL leads the agent into taking several actions that may be high risk on the robotic platform in the real world, thus training in simulation and then developing strategies to adapt the learned behavior onto the real platform is the way to go for now.

For training in simulations we used variety of simulation platforms available in the robotics community, that can simulate a robotic arm with its gripper. Here we focused on solving rigid block manipulation tasks with the robotic arm inside the simulation, such as picking and placing, stacking.

1.3.1 Barret WAM robotic arm simulation in Gazebo

Gazebo is an open source community maintained physics based simulation platform which is very popular in the Robotics community. IRI provided us with Barret WAM robotic arm simulation in Gazebo, with action servers written in ROS. To make the simulation compatible with the above mentioned algorithm implementations we integrated the barret WAM Gazebo simulation with OpenAI gym with the help of Gym-gazebo package, thus the simulation environment in Gazebo can be used as a standalone gym environment with all the functionalities of a gym based environment.

1.3.2 Fetch arm simulation in Mujoco

MuJoCo is a physics engine aiming to facilitate research and development in robotics, biomechanics, graphics and animation, and other areas where fast and accurate simulation is needed. OpenAI gym based environments such as Fetch arm simulation run in Mujoco. We would be reporting results on the Fetch Robotics environments available from OpenAI gym. Important

functionalities that Mujoco offers are GPU support, simulation without rendering (1000x faster than real time), and multiple environment initialization, all of which prove to be extremely beneficial in the case of training a Deep RL model which requires multiple interactions with the environment at training time.

1.4 Algorithm implementations

We implemented the paper [9] "Overcoming exploration in Reinforcement learning with demonstrations", and with the implementation trained robotic arms in multiple simulation environments to solve rigid block manipulation tasks. OpenAI baselines is an open-source repository that provides high quality implementations of many state-of-the-art RL algorithms. Starting from the HER implementation in Baselines, we modified and improved the implementation to train the policies from expert demonstrations if available, as described in the paper [9]. The four main ideas of the paper include, 1. Maintaining a secondary buffer called the demo buffer for storing demonstrations and training from this data as well, 2. Introducing an auxiliary behavior cloning loss for training on the demo data, 3. Using Q-value filtering to account for sub-optimal demonstrations in the demo buffer, 4. Resetting the training from favorable states in the environment.

We open-sourced the implementation on Github and reported better results in performance and convergence as compared to vanilla HER. Link to Github repository here. Also, contributed to OpenAI Baselines by submitting a pull request to add a new improved functionality to HER Baseline, which got accepted with encouraging comments.

1.5 Tasks solved with the implementations

The learning algorithm we developed is agnostic of the simulation environment used, and we solved multiple manipulation tasks with our algorithm.

1.5.1 Pick place objects with Barret WAM

In this case we were working in the Joint space of the robotic arm and the action space was of the order 8 consisting of the angles of the 7 DOF robotic arm and the gripper state, the observation space consisted of the positions and orientations of the blocks on table and end effector position of the robotic arm. Figure 1.1 shows picture of the simulation. We solved the following tasks on a Barret WAM arm simulation:

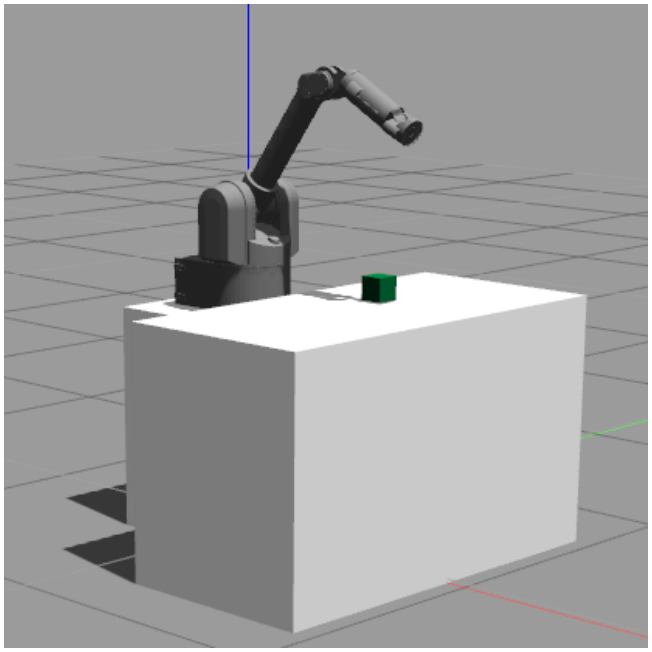

Figure 1.1: Barret WAM arm in simulation.

- Reaching, i.e. Learning Inverse Kinemantics (learning how to reach a particular point inside the workspace).
- PicknPlace, i.e. Learning to grasp a block and take it to a given goal inside the workspace.
- Stack, i.e. Learning to stack a block on top of another block.

1.5.2 Stacking blocks with Fetch Arm

In this case we were working in the Cartesian space and the action space was of the order 4, the x, y, z position of the end effector and the gripper state (open/close), the observation space

consisted of the positions, velocities and orientations of the blocks on table and end effector position of the robotic arm. We solved the following tasks on a Fetch arm simulation, Figure 1.2 shows picture of the simulation:

- Reaching, i.e. learning learning how to make the end effector reach a particular point in space (Working in the Cartesian space)
- PicknPlace, i.e. Learning to grasp a block and take it to a given goal inside the workspace.
- Stack, i.e. Learning to stack a block on top of another block
- Stack 3 blocks on top of each other

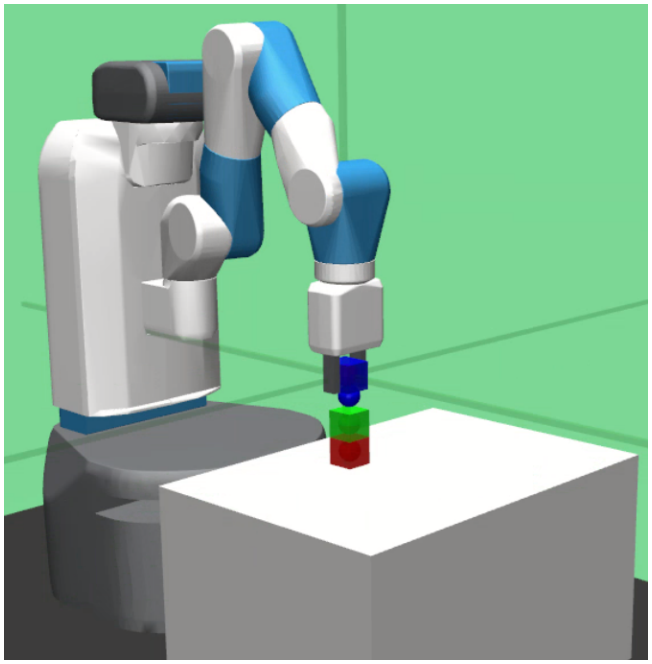

Figure 1.2: Fetch arm stack 3 blocks.

1.6 Generating Demonstrations

For solving the above tasks with learning from demonstrations to overcome exploration paradigm we generated demonstrations for each of the above described tasks using hard-coded python scripts. Not all the generated demonstrations were perfect, which is good as our algorithm uses a Q-filter which accounts for all the bad demonstration data.

1.7 Results

Training with demonstrations helps overcome the exploration problem and achieves a faster and better convergence. The following graph in Figure 1.3 contrasts the difference between training with and without demonstration data, I report the the mean Q values vs Epoch and the Success Rate vs Epoch:

Clearly, the use of demonstrations enables a faster and better convergence in the Q values as apparent from the graphs. Also the success condition is achieved much faster reaching up to 100% performance just around the 400^{th} epoch whereas in the case without demonstrations even after 1000 iterations the agent hardly reaches 70% success rate. Future work in this direction would include solving much more complex tasks and improving the algorithm further to enable a better and efficient usage of demonstrations data.

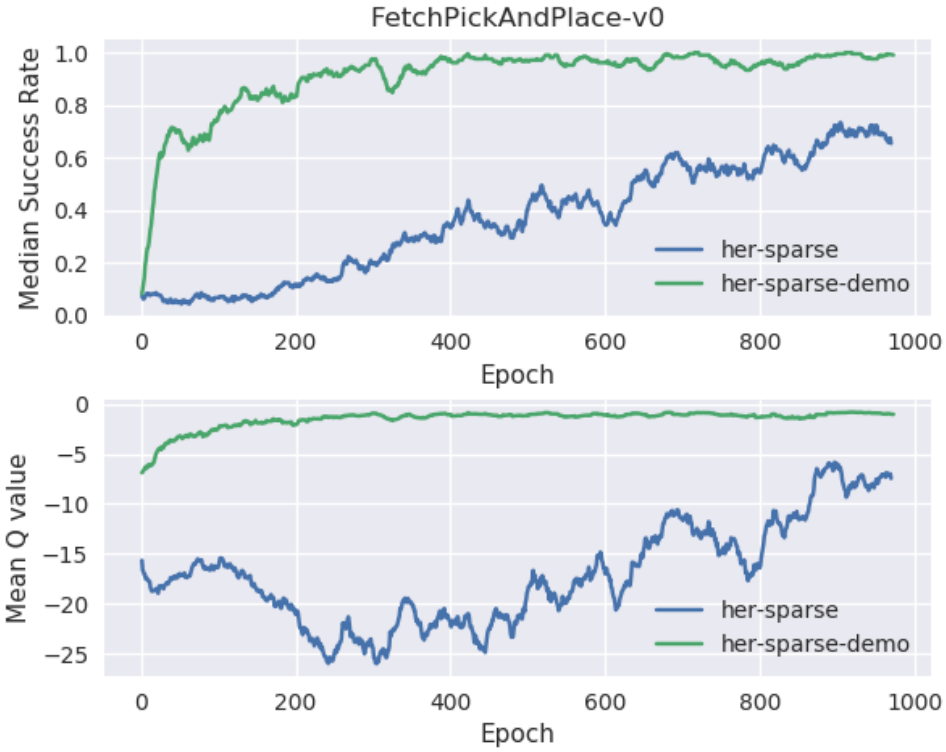

Figure 1.3: Training results for Fetch Pick and Place task contrasting between training with and without demonstration data.

Chapter 2

Textile object simulation

2.1 Introduction

The scarcity of the availability of textile object simulation platforms in the robotics learning community led us to create our own framework using SOFA, a physics based simulation engine primarily designed for medical simulation. We created a textile simulation environment to test and train our learning from demonstrations based Deep RL algorithm and to design algorithms based on these tests that can solve textile object manipulation tasks. This chapter talks about the simulation environment that we created, the challenges faced and the problems solved.

2.2 SOFA simulation Framework

SOFA is an efficient framework dedicated to research, prototyping and development of physics-based simulations. It is free, open-source and community driven. Simulating a textile object in sofa involves defining a mesh of the object and the physical properties associated with the kind of deformation wanted. A SOFA simulation can be run without rendering which proves to be useful for Deep RL research, also it provides GPU compatibility.

The SofaPython plugin exposes a lot of major set of sofa functionality to python, and we could

use it to create iteration able programs for our use-case in python itself.

2.2.1 Textile simulation in SOFA

After a thorough introduction of the software's capabilities we developed a simulation environment for the case of a cloth lying on a table and a gripper available to grasp and manipulate the cloth. Currently we are working with the task of learning how to fold the cloth from vertex to the diagonally opposite vertex. The simulation environment can be described to have the following properties:

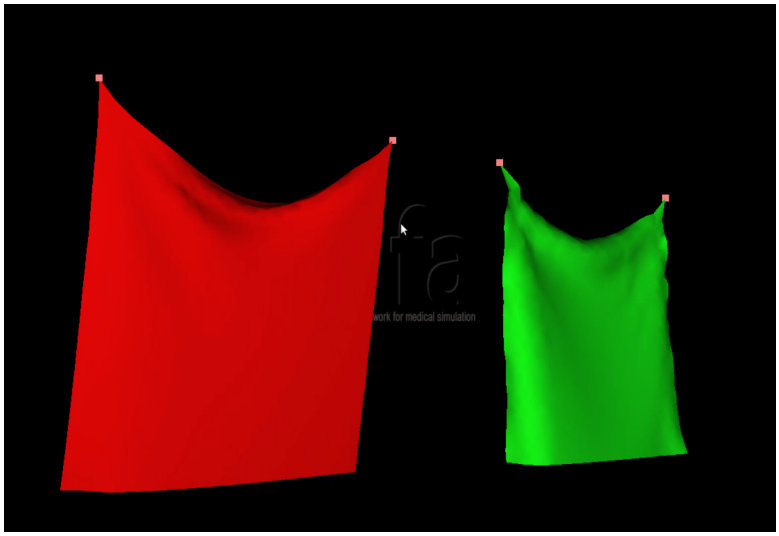

Figure 2.1: A typical textile simulation in SOFA.

- **Observation space :** The simulation provides with the 3 dimensional position and velocity of all the points of the mesh of the cloth and the position of the gripper as observations.

- **Action space :** The 3 dimensional motion in x, y and z with the gripping action constitutes the action space.

- **Episode length :** A maximum episodic length of 150 steps is chosen for the particular task of cloth folding vertex to diagonally opposite vertex.

- **Reward :** The aim of the work is to train the agent to perform a described task with minimum reward information possible (sparse rewards), the best functional form of reward function would be determined during the experiments.

To give an intuition about the structure of the reward function, a sparse reward function for the task of folding a cloth vertex to vertex could be giving a reward of 0 when the position of vertex being manipulated is within a defined minimum distance of the goal vertex position and -1 otherwise.

2.3 Gym like environment for SOFA

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. This is the gym open-source library, which gives you access to a standardized set of environments. Gym makes no assumptions about the structure of the learning agent, and is compatible with any numerical computation library, such as TensorFlow or Theano. We can use it from Python code, and soon from other languages. Our implementation of the learning to overcome exploration through demonstration paper was based on the gym API.

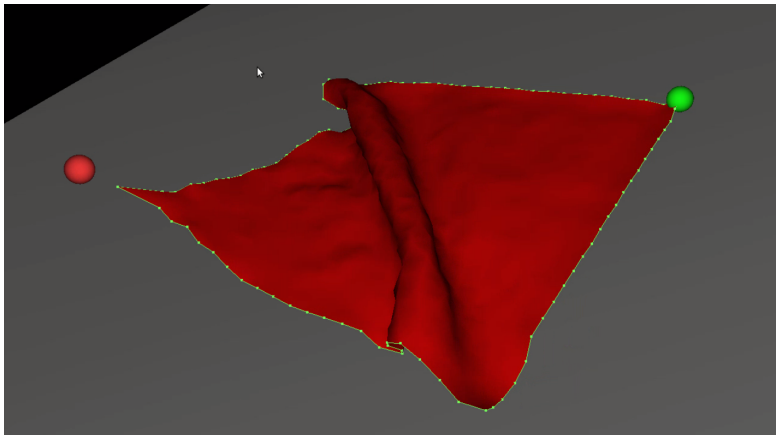

Figure 2.2: Our cloth simulation environment in SOFA

Thus, it made complete sense to create a gym environment of our SOFA cloth simulation. But the problem was with the different versions of python in the SofaPython code (Python 2) and the implementation of the paper was in Python 3. To overcome this problem, we created a

piping framework by writing the data to be passed into files through one module and reading the JSON formatted data in the other module. Indeed this method slowed down the speed of the implementation but it seemed to be the best possible way currently. There are efforts in the sofa community to port SofaPython code to Python 3, but it is still in the build phase.Figure 2.2 shows an image of our simulation environment.

2.4 Challenges

The simulation of a cloth looks good in real time, but there are still some questions left unanswered about the friction behavior of the cloth with the surface and the stiffness of the mesh of the cloth which gets deformed more than expected from a real cloth. We hope to overcome these design problems soon.

Chapter 3

Learning cloth manipulation

After successfully creating a textile object simulation environment which is capable of GPU rendering and compatible with our learning algorithm implementation, we now move on to studying the necessary changes required to learn cloth manipulation tasks, in particular the vertex to diagonally opposite vertex folding task as shown in Figure 3.1. This chapter talks about the paradigm which we have used to train our manipulating agent, the inferences we have made and the challenges we have in the near future.

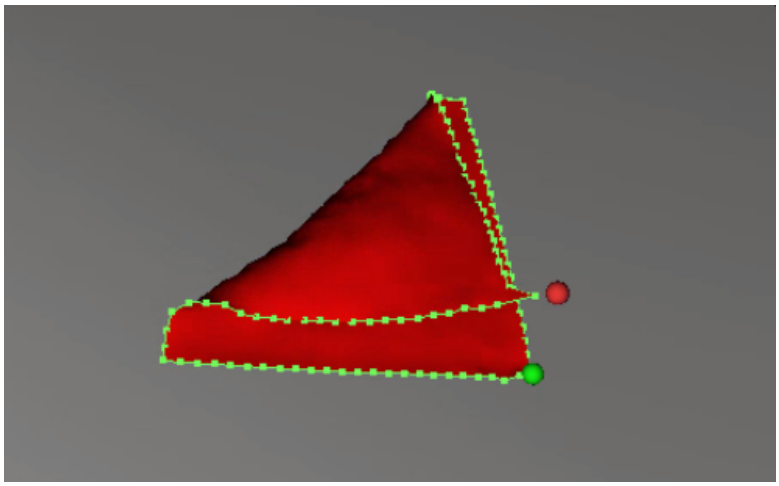

Figure 3.1: Folding vertex to diagonally opposite vertex

3.1 Generating demonstrations

Our training algorithm requires demonstrations from an expert, although not all of these demonstrations have to be perfect. In order to generate demonstrations for the respective task we will be using hard-coded python scripts and saving the observation data and actions later to be used in the learning algorithm. An important point to mention here is the generation of demonstrations with variance in the goal position and variance in trajectories as well, this allows the network to generalize well. Showing similar data to the network in every loop would be detrimental to learning.

3.2 Sub-tasks considered

The task of folding a cloth in a way that the grasped vertex meets the diagonally opposite vertex can be viewed as multiple sub-tasks such as, in order, 1. Gripping the vertex to be manipulated, 2. Moving this vertex to a specified point in space while not dropping it, 3. Releasing the vertex near to the goal vertex. Thus Before diving into the complete task, it is beneficial to consider learning these sub tasks in order to see how certain hyper-parameters must be modified to account for the additional challenge of deformation inside the objects in comparison to the rigid object manipulation task.

3.2.1 Grasping the vertex

The grasping action in this case is binary, unlike the rigid object problem where we had an actual robotic gripper simulation, that used to work in steps of opening and closing. Thus the problem of grasping in a binary case had to be studied, in order to see if the neural network is capable of comprehending the difference in data distribution when most of the action is 0.0 and only a single step out of 150 steps is 1.0 (the gripping action). One important result that we determined was that giving a value of range(0.9, 1) for the gripping action rather than a straight 1.0 works better in practice. This is probably because it is easier for a function approximator

to learn changes in data that are distributed (properly defined gradients) rather than a spike in data (infinite slope).

3.2.2 Moving grasped vertex to a point

Because of the physics and working of the simulator the length of a single episode in the case of cloth simulator is much longer than the rigid object case. Thus, to account for a longer horizon, we tried experimenting with the discount factor gamma. Similarly we experimented with a bunch of important hyper-parameters of the actor-critic network that we are using for learning.

Hyper-parameter studies

For understanding the behavior and the learning process of the network, we ran experiments with a combination of different hyper-parameters. This study was performed on the task of grasping a vertex and taking this vertex to a specified goal position.

- **Learning rate :** We varied the learning rate for both actor and critic networks and reported the results in Figure 3.2

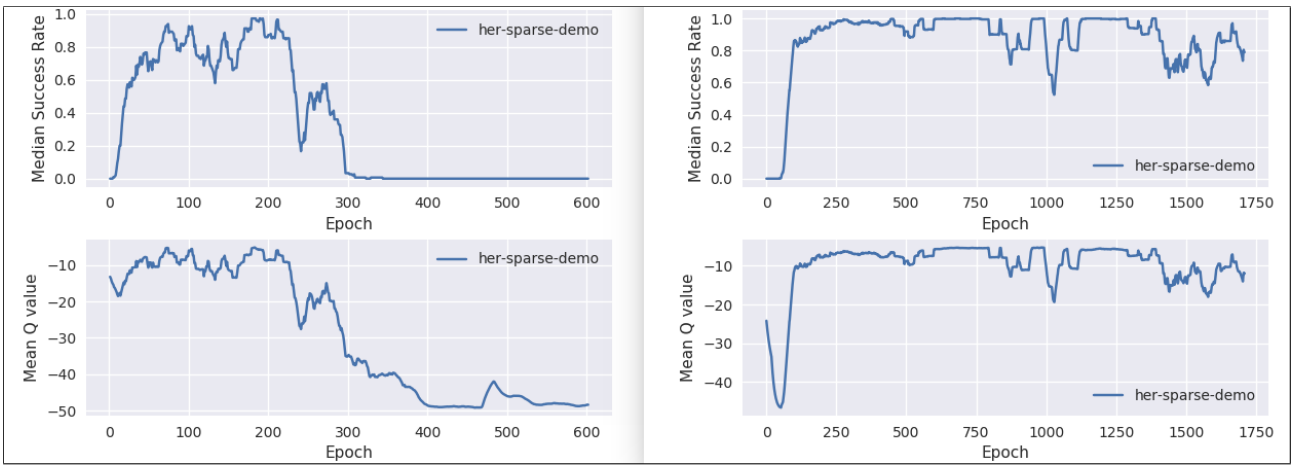

Figure 3.2: Learning rate comparison, left. Higher learning rate, right. Smaller learning rate

- **Discount factor :** We trained the RL algorithm with different discount factors and reported the results in Figure 3.3

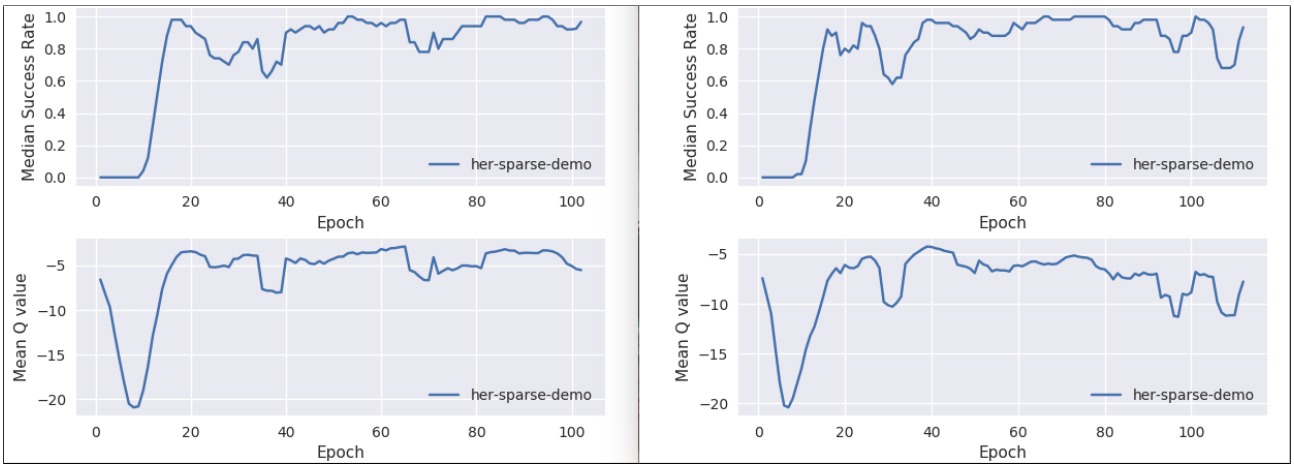

Figure 3.3: Discount factor comparison, left. Gamma=0.98, right. Gamma=0.99

- **Network size :** We varied the network size of both actor and critic networks and reported the results in Figure 3.4

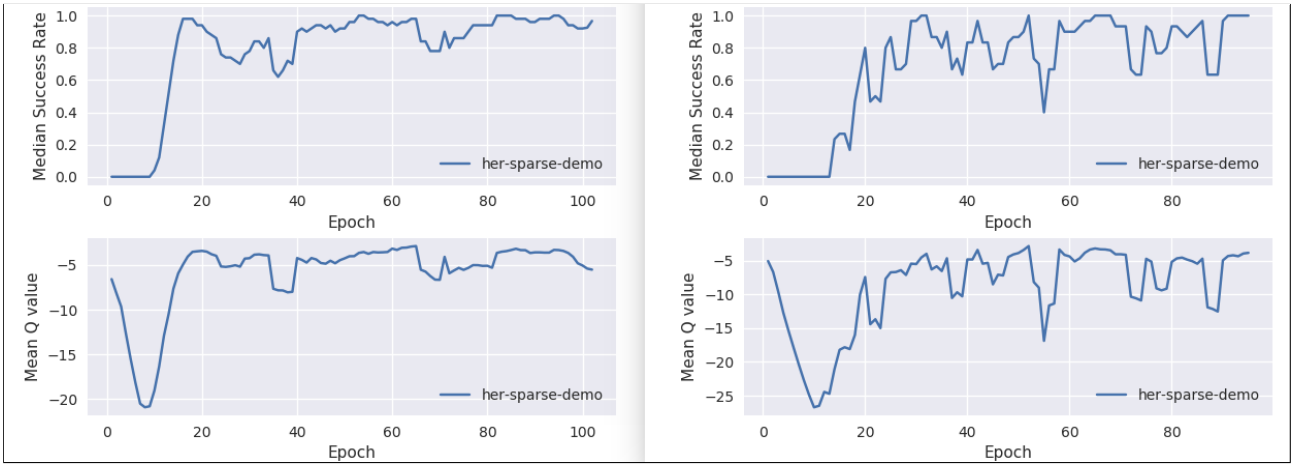

Figure 3.4: Network size comparison, left. 4 hidden layers, right. 3 hidden layers

- **Noise :** We varied the action noise used while training and reported the results in Figure 3.5

- **Distance threshold :** We varied the distance threshold that determines the minimum distance required to achieve goal and reported the results in Figure 3.6

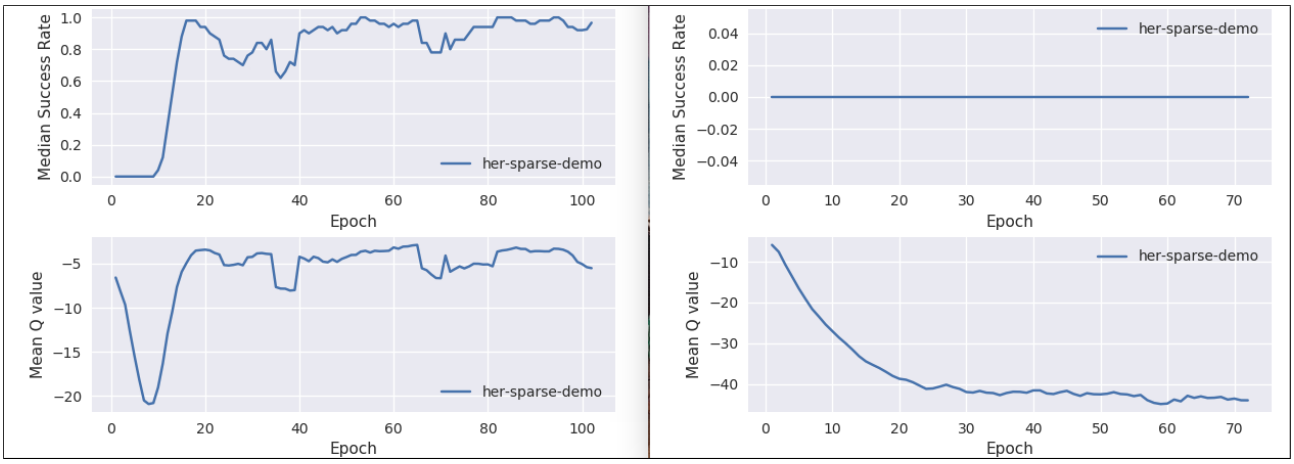

Figure 3.5: Action noise comparison, left. 10% noise in actions, right. 30% noise in actions

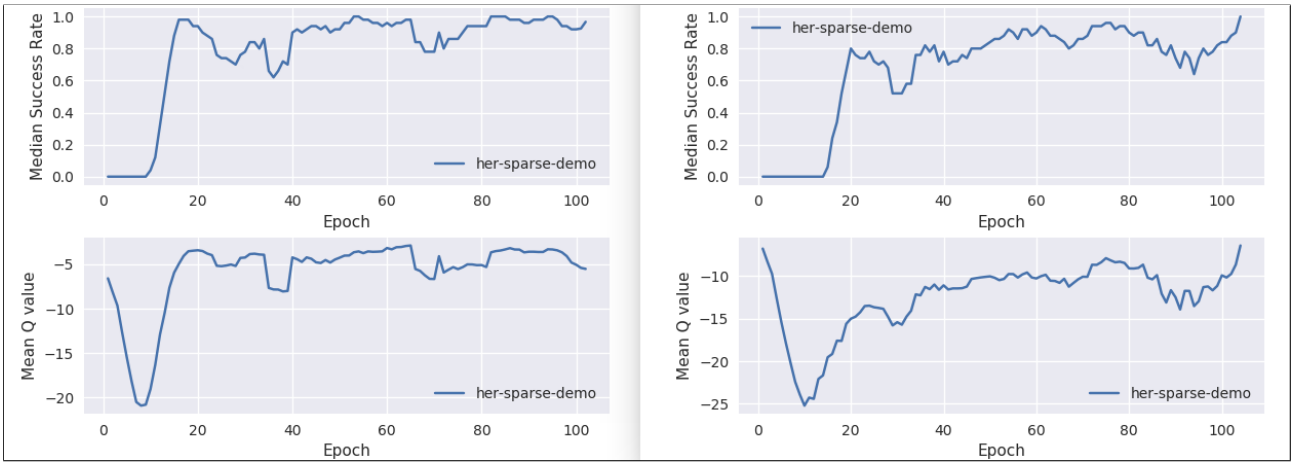

Figure 3.6: Distance threshold comparison, left. Larger dist. threshold (10 units), right. Smaller threshold (5 units)

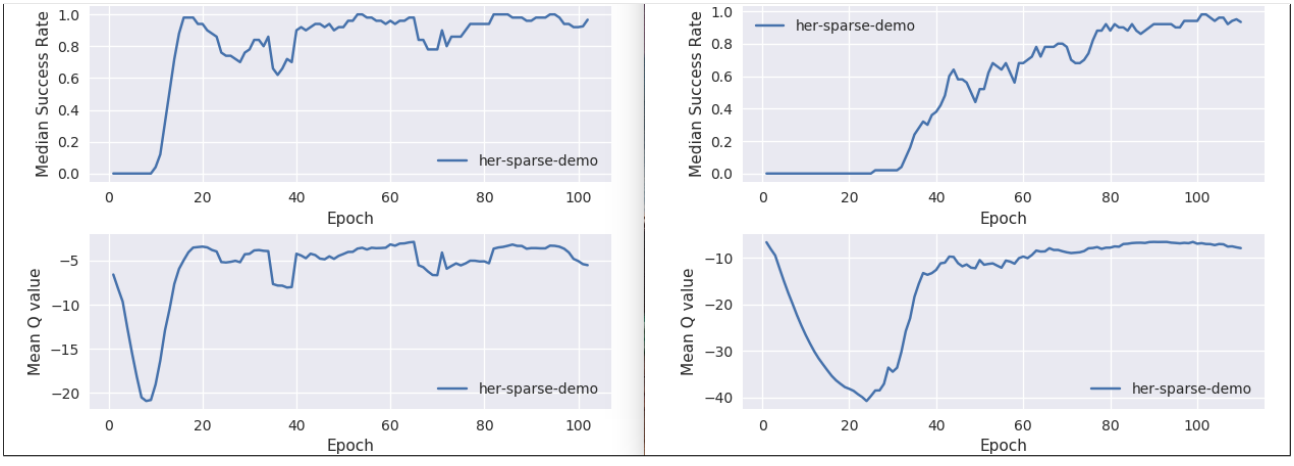

Figure 3.7: Work-space size comparison, left. Smaller work-space, right. Larger work-space (x4)

- **Work-space size :** We varied the work-space size the actuator was working inside and reported the results in Figure 3.7

- **Number of demonstrations :** We varied the number of demonstrations from expert used for training and reported the results in Figure 3.8

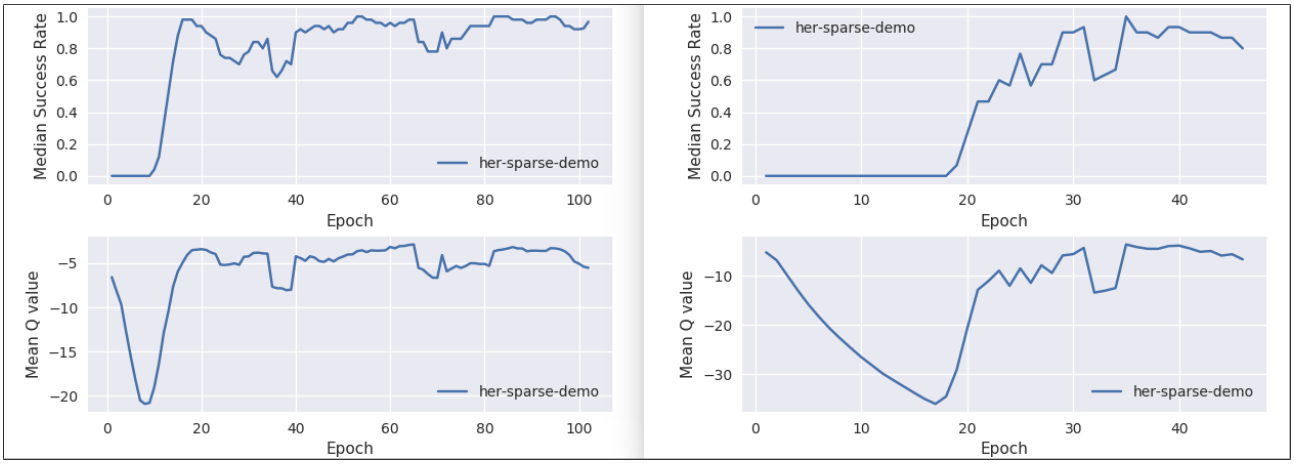

Figure 3.8: Number of demonstrations comparison, left. 40 demonstrations, right. 5 demonstrations

3.3 Inferences drawn from the above study

The above hyper-parameter study allowed us to come to several important conclusions, some as expected, some not as expected:

- Slowing down the learning process by decreasing the learning rate allows the agent to learn more steadily as we can see from the graph comparisons in Figure 3.2. With a higher learning rate the agent does learn faster, but later in the training it unlearns all this information and the performance degrades,

- Unexpectedly, even for a slightly longer horizon problem, increasing the discount factor to 0.99 from 0.98 makes the agent perform slightly poorly, the difference is not huge as it can be seen from the graphs in Figure 3.3, but the lower discount factor performs better without doubt,

- Decreasing the network size by decreasing the number of hidden layers in both the actor and critic networks leads to a poorer learning as expected as shown in Figure 3.4,

- Introducing higher noise in the actions while training leads to the agent being not able to learn anything, this is because with increased noise the gripping action is affected the most as shown in Figure 3.5,

- Although the difference is not startling, still having a larger distance threshold for evaluating the goal condition facilitates learning in the RL setting as can be seen in Figure 3.6,

- Working in a smaller space is easier and the convergence is achieved faster as can be seen in Figure 3.7, but we can not always control the size of the work space as it is task dependent, but it could be a good idea to reduce the work-space size for trouble-shooting,

- Providing fewer number of demonstrations, does affect the training in a slightly negative way, but the agent is indeed able to learn even from very less number of demonstrations as shown in Figure 3.8, only slower.

3.4 Current task results

After getting important information about how to set up the hyper-parameters we trained the RL agent to learn from provided demonstrations on the task of grasping a vertex and taking it to a random goal position. The goal position is a point randomly sampled on and above the area of the cloth. The agent learned to perform this task with 92% accuracy while learning how to carefully manipulate the grasped point such that the cloth does not crumple. In this task we are currently giving the positions and velocities of the diagonally opposite vertices as observations, so the position of the rest of the cloth is unknown to the agent as of now. Getting good results on a smaller set of observation space would now allow us to move to a larger observation space, while studying the affects of introducing more points on the cloth.

3.5 Comparison with Rigid body manipulation

The problem of manipulating a textile object is different to manipulating multiple rigid objects in the following ways:

- Manipulating a single object's position in a rigid blocks case does not have any effect on other blocks (unless collision), but in the case of a textile object, all the observation points actually lie inside the same body and making changes in the position/velocity of one of the points leads to changes in the position of other points in observation. This change is governed by the physical properties of the textile abject and thus the learning algorithm has an additional task of learning the way the object behaves.

- While manipulating rigid blocks the velocity of the motion of the gripper did not matter when trying to achieve the goal state, but in the case of textile objects the velocity of the gripper plays a crucial role in determining the final state the environment will land in. And this difference in the distribution of velocities can result in very different performances even for the same tasks. Thus, the network has to also learn about how the velocity distribution data affects the task.

Because of the multiple number of states the environment can land in, in the case of clothes we need to have tighter bounds on the reward definition and the success condition. Slight differences in the reward definition can have huge effects on the optimization objective. For example, take the case of folding the cloth vertex to diagonally opposite vertex, in this case the position of the other 2 vertices in the final state does not really affect the definition of the task, but taking them into account inside the reward function definition can help us to make the agent learn a much cleaner and less vague folding of the cloth.

3.6 Challenges faced and Future Work

Determining the best observation space that can represent the task being considered in a succinct way is the main challenge ahead of us. Firstly, because making the observation space too large can have detrimental effects on learning, but at the same time it should contain enough information for the agent to make proper mappings between the observations and the task being solved. Secondly, although we aim to keep the reward function as sparse as possible, there arise multiple cases of final states that would have received the reward but still be unable to complete the task, for example, keeping the other two vertices at place while folding. While making these changes to the reward function and success condition we do need to keep a check on the HER functionality and not violate it.

Bibliography

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

[3] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pages 363–372. Springer, 2006.

[4] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[5] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 156–163. IEEE, 2015.

[6] Yevgen Chebotar, Karol Hausman, Zhe Su, Gaurav S Sukhatme, and Stefan Schaal. Self-supervised regrasping using spatio-temporal tactile features and reinforcement learning. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 1960–1966. IEEE, 2016.

[7] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[8] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[9] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

Acknowledgements

This work has been partially supported by the CHIST-ERA Project I-DRESS PCIN-2015-147, by the ERC Advanced Grant CLOTHILDE ERC-2016-ADG-741930 and the Spanish State Research Agency through the María de Maeztu Seal of Excellence to IRI (MDM-2016-0656).

IRI reports

This report is in the series of IRI technical reports.
All IRI technical reports are available for download at the IRI website
`http://www.iri.upc.edu`.