Research Internship

Specialization: Information and Communication Sciences and Technology

Academic Year: 2019

Generative Replay for Continual Learning in Robotics

This is a non-confidential and publishable report

Author:
Bunthet SAY

Promotion:
2020

ENSTA Paris Tutor:
David Filliat

Intership Tutor:
Natalia Diaz Rodriguez

Internship from 28/05/2019 to 09/08/2019

Host Organization: ENSTA Paristech

Address: 828, Boulevard des Maréchaux, 91762 Palaiseau Cedex

Confidentiality Notice

This is a non-confidential report and publishable on the Internet.
The codes associated with this report can be used freely.

Abstract

Humans and animals are capable of continually gain and transfer knowledge trough their life. This ability has inspired research in Artificial Intelligence, especially in reinforcement learning. The goal is to have an agent that acquire continuously streams of information and skill and can master them.However while continually acquire information,often,the agent began to forget the previous information. This phenomenon is called catastrophic forgetting, a crucial challenge toward continual learning.

Policy distillation is a method that lets an agent(student) acquire knowledge from many agents (teachers) without catastrophic forgetting by training on the union of "on-policy" data sets of all the teachers[14]. However, in order to have the "on-policy" data set, we need to take the agent to the environment and collect data. So we need to have access to the environment. This is not always easy and practical or it may cost a lot in real life.

This project propose the method of generative replay on each teacher environment. There are two main experiments. In the first experiment, we use generative replay to generate "on-policy" data set of each teacher environment. This generated data set is given by a generative model trained on the real "on-policy" data set of corresponding teacher's environment. We use Conditional Variational Autoencoder(CVAE) and Conditional Generative Adversarial Network (CGAN) conditioned on action of the agent and a feature of each environment as generative model.

In the second experiment, we use generative replay to generate data set of each teacher environment without any policy.We use CVAE conditioned on 2 features of each environment as generative model.

We train generative models with several configurations, hyper paremeters and architectures . The results of our experiments are not quite satisfied, however we also leave some ideas for possible cause and future improvement.

All the codes of the experiments can be found in this Github Repository and the experiments process is described in this Read_me.md.

Keywords :

Generative Replay, Policy Distillation, Continual Learning, Catastrophic Forgetting

Résumé

Les êtres humains et les animaux sont capables d'acquérir et de transférer continuellement des connaissances au cours de leur vie. Cette capacité a inspiré la recherche en Intelligence Artificielle, en particulier dans l'apprentissage par renforcement. L'objectif de l'enjeu est d'avoir un agent qui acquiert en continu des flux d'informations et de compétences en les maîtrisant. Cependant, tout en continuant à acquérir des informations, il a commencé souvent à oublier les informations précédentes. Ce phénomène s'appelle l'oubli catastrophique, un défi crucial pour l'apprentissage continu.

La distillation de politique est une méthode qui permet à un agent (étudiant) d'acquérir des connaissances auprès de nombreux agents (enseignants) sans oubli catastrophique grâce à l'entraînement sur l'union de jeu de données "on-policy" de tous les enseignants[14]. Toutefois, pour avoir le jeu de données "on-policy", nous devons amener l'agent dans l'environnement et collecter des données. Nous devons donc avoir accès à l'environnement. Ce n'est pas toujours facile et pratique ou cela peut coûter cher dans la vie réelle.

Ce projet propose la méthode de "Generative Replay" sur chaque environnement enseignant. Il y a deux expériences principales.Pour la première expérience, nous utilisons "Generative Replay" pour générer le jeu de données "on-policy" de chaque environnement enseignant. Cet ensemble de données généré est donné par un modèle génératif entraîné sur le vrai jeu de données "on-policy" de l'environnement enseignant correspondant. Nous utilisons "Conditional Variational Autoencoder" (CVAE) et "Conditional Generative Adversarial Network" (CGAN) conditionnés par l'action de l'agent et par une caractéristique de chaque environnement en tant que modèle génératif.

Dans la deuxième expérience, nous utilisons "Generative Replay" pour générer un ensemble de données de chaque environnement d'enseignant sans aucune politique. Nous utilisons CVAE contditionnée sur 2 caractéristiques de chaque environnement en tant que modèle génératif.

Nous entraînons des modèles génératifs avec plusieurs configurations, hyperparemètres et architectures sur le jeu de données "on-policy" de deux environnements. Les résultats de nos expériences ne sont pas tout à fait satisfaisants, mais nous laissons également quelques idées sur les causes possibles et les améliorations futures.

Tous les codes des expériences se trouveent dans cette Répositoire de Github et le processus d'expériences est décrit dans ce Read_me.md.

Mots-Clés :

Generative Replay, Distillation de politique, Apprentissage Continu, Oubli catastrophique

Acknowledgements

The 11-weeks internship at U2IS ENSTA was a great opportunity for my insertion in research. It help me to be more familiar with research papers and to learn novel scientific methods especially in the field of deep learning and robotic. I also improve my scientific experimental skill such as note taking, data processing, data analysis, evaluation , visualization, etc. In addition, I learn how to efficiently use computer resources to optimize my project's experiments' process. Beside scientific skills, I also learn some soft skills like communication, negotiation, collaboration, group working, time management and tasks scheduling.

In this occasion, I would like to express my big gratitude firstly to Prof. David Filliat for offering me this internship opportunity, as well as Prof. Natalia Diaz Rodriguez who has been tutoring me throughout my internship. Secondly, I would like to thank to René Traoré, Hugo Caselles-Dupré, Timothée Lesort, Lu Lin, Sun Te for introducing me the project and it's framework, suggesting ideas and technical methods, revising my experiments' results and helping me with coding. Finally, I would also like to thanks to other interns in the laboratory for the discussion over various research topics and their encouragement. It lets me know more about different topics in which some are quite helpful for my project.

Contents

Abstract . . . 5
Acknowledgements . . . 8
Contents . . . 10

1 Introduction **11**
1.1 General background . . . 11
1.2 Structure of the report . . . 11

2 Context **12**
2.1 Continual Learning . . . 12
2.2 Policy Distillation . . . 12
2.3 Generative Replay . . . 13
2.3.1 Variational Autoencoder . . . 13
2.3.2 Conditional Variational Autoencoder . . . 14
2.3.3 Generative Adversarial Network . . . 15
2.3.4 Conditional Generative Adversarial Network . . . 16

3 Experiment details and Results **17**
3.1 Train RL Models . . . 17
3.2 Train Generative Models . . . 20
3.2.1 Experiment 1 . . . 20
3.2.2 Experiment 2 . . . 21
3.3 Evaluate Generative model . . . 22
3.3.1 Experiment 1 . . . 22
3.3.2 Experiment 2 . . . 27
3.4 Policy Distillation . . . 27
3.4.1 Experiment 1 . . . 28
3.4.2 Experiment 2 . . . 29
3.5 Evaluate Distilled Model . . . 31

4 Conclusion **33**
4.1 Challenges . . . 33
4.2 Improvement and Future Work . . . 34
Annexes . . . 35

Bibliography 38
List of Figures 39

1 - Introduction

1.1 General background

In recent year, several robotic research interest in teaching an agent/robot to achieve incrementally several tasks without forgetting the previous learned skills. This is the ability of continual learning discovered in humans and animals.[5]

With the arrival of artificial intelligence and deep learning, we have witnessed great application in robotic. We have trained agent that is specialize in solving a particular task using technique like deep reinforcement learning. We would like to extend this ability sequentially by continual learning.

Three major approaches have contributed in continual learning : *Policy distillation*[11] that compressed incrementally previous models into one; *Elastic Weight Consolidation*[2] that add constraint of the new network on the principle gradient direction to preserve information of previous structures and *Progressive neural network*[12], through which we pass inputs through all previous models and gather the internal output to feed a new trained network.

Policy distillation has been a promising candidate in continual learning's technique.With this method, agent(student) can acquire knowledge from many agents (teachers) without catastrophic forgetting by training of the union of "on-policy" data sets of all the teachers. However, in order to have the "on-policy" data set, we need have the access to the environment which is not always easy and practical.

1.2 Structure of the report

Before stepping to the experiments, we begin in chapter 1 general background of the research matter. In chapter 2, we give brief introduction and concept of continual learning, policy distillation, generative replay and several generative models that we are going to use in our experiments. In chapter 3, we detail about the process, configuration, and results of of the experiments. We conclude in chapter 4 with some challenges and future improvement of our experiment setting.

2 - Context

2.1 Continual Learning

Continual learning is an ability that has been discovered in humans and animals of being able to continuously acquire and transfer knowledge.This ability, referred to as lifelong learning. It has inspired lot of research in neuroscience and artificial intelligence to equip this ability to an AI. In autonomous system, we are looking forward to have an autonomous agents interacting in the real be able to learn incrementally with stream of information. Some has intended to train a RL model on first task (information), then on second tasks and so on. However, this sequential learning remains a challenge for machine learning and neural network models since the continual acquisition of incrementally information generally leads to catastrophic forgetting or interference [5].

Catastrophic forgetting is a well-known phenomenon of a neural network experiencing a rapid overriding of previously learned knowledge when trained sequentially on new data.

2.2 Policy Distillation

Policy distillation is a method that extract the policy of a reinforcement learning agent and train a new network that performs at the expert level while being dramatically smaller and more efficient. Furthermore, the same method can be used to consolidate multiple task-specific policies into a single policy. [10]

It is a method of transferring knowledge from *teacher* models to a *student* model in the lifelong learning manner without catastrophic forgetting. The framework of a simple policy distillation is similar to supervised learning. The student acquires knowledge from the teacher. The only different is that in policy distillation, a student is supervised by more than one teachers.

Concretely,if we have K Reinforcement Learning models that represent K trained agents learn to solve individually K different tasks, the student model that distill perfectly the knowledge from all the teachers will be able to solve all the teachers' task while being relatively small.

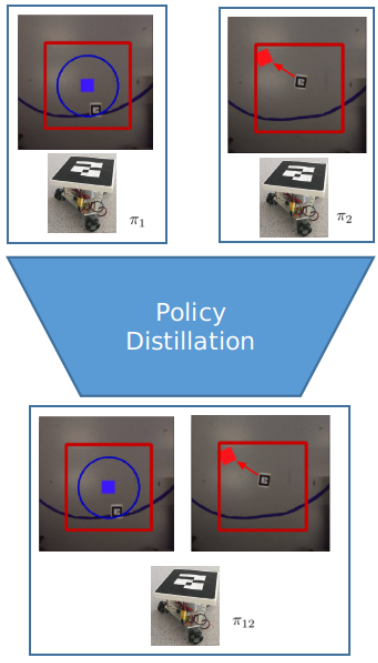

Figure 2.1: Policy Distillation Schema

2.3 Generative Replay

Generative Replay is a technique of data generation that aim to maintain past knowledge. It can do so by training a model called "generative model" on a data set of an environment and by sampling from a distribution, mostly normal distribution, an ideal generative model can generated data set that has the same distribution to training data set. Basically, it keeps the relevant information and knowledge about an environment.

This technique appears to improve the data set generation in an environment. Classically, in order to generate data set we need to access to the environment which is not practical and not always easy. Generative model lets us generate data set from each environment with no need to access to it. Variational Auto-Encoder(VAE) and Generative Adversarial Network(GAN) are 2 basic generative models and in our experiments, we use the conditional form of this two family as generative model.

2.3.1 Variational Autoencoder

In neural net language, a Variational Autoencoder consists of an encoder, a decoder, and a loss function.

The encoder is a neural network. Its input is a data point x, its output is a

hidden representation z, and it has weights and biases θ.z is typically referred to as a 'bottleneck' because the encoder must learn an efficient compression of the data into this lower-dimensional space. Let's denote the encoder by $q_\theta(z|x)$.

The decoder is another neural network. Its input is the latent variable z, it outputs the parameters to the probability distribution of the data, and has weights and biases ϕ. Let's denote the decoder by $p_\phi(x|z)$.

The loss function of the Variational Autoencoder is the negative log-likelihood with a regularizer. We can decompose the loss function into only terms that depend on a single data point X_i. The total loss equals then $\sum_{i=1}^{N} l_i$ for N total data points. The loss function l_i for each data point X_i is expressed as:

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|X_i)}[\log p_\phi(X_i \mid z)] + \mathbb{KL}(q_\theta(z \mid X_i) \mid\mid p(z))$$

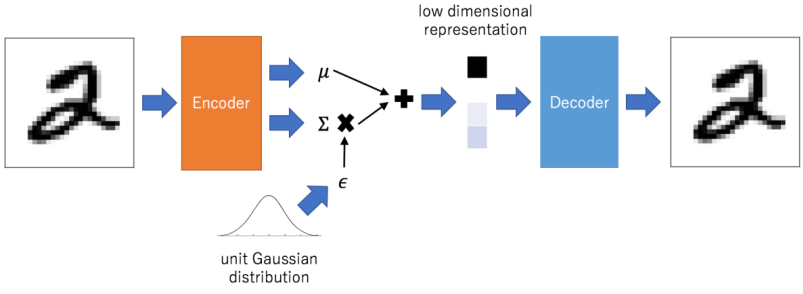

Figure 2.2: Variational Autoencoder schema

The first term is the reconstruction loss, or expected negative log-likelihood of the i^{th} data point.The second term is the Kullback-Leibler divergence between the encoder's distribution This is the Kullback-Leibler divergence between the encoder's distribution $q_\theta(z \mid x)$ and $p(z)$. It measures how much information is lost when using q to represent p, i.e how close q is to p.

2.3.2 Conditional Variational Autoencoder

Conditional Variational Autoencoder (CVAE) is an extension of VAE aiming to complete the uncontrolability of VAE in term of data generation process when the training data set has several numbers of class. It can generate some specific class of data while VAE can't.

It can do so by conditioning the encoder and the decoder of VAE with a class c. The encoder is now conditioned to two variables X and c : $q_\theta(z|X, c)$.Similarly, the decoder is now conditioned to two variables z and c : $p_\phi(X|z, c)$. Hence, the loss function of CVAE is equal to $\sum_{i=1}^{N} l_i$ for N total data points where l_i equals :

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|X_i, c_i)}[\log p_\phi(X_i \mid z, c_i)] + \mathbb{KL}(q_\theta(z \mid X_i, c_i) \mid\mid p(z|c_i))$$

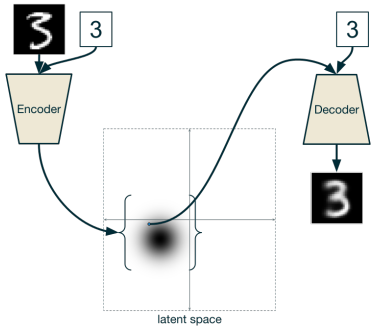

Figure 2.3: Conditional Variational Autoencoder schema

2.3.3 Generative Adversarial Network

In neural network language, Generative Adversarial Network (GAN) is composed of two neural network: Generator(G) and Discriminator(D) trained simultaneously. The generator captures the data distribution, and the discriminator estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a mini-max two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multi-players perceptrons, the entire system can be trained with back-propagation.There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples.

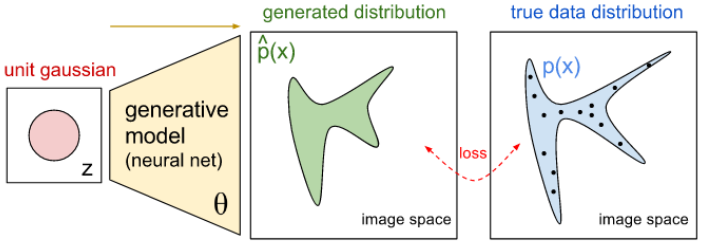

Figure 2.4: Generative Adversarial Network schema

The loss function of GAN is the Mini-Max loss between D and G:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_x(z)}[\log(1 - D(G(z)))] \text{ [1]}$$

2.3.4 Conditional Generative Adversarial Network

Conditional Generative Adversarial Network (CGAN) is a deep learning method where a condition is applied to the standard GAN, meaning that both the generator and discriminator are conditioned on some sort of auxiliary information such as class labels. As a result, the ideal model can learn multi-modal mapping from inputs to outputs by feeding it with different contextual information.

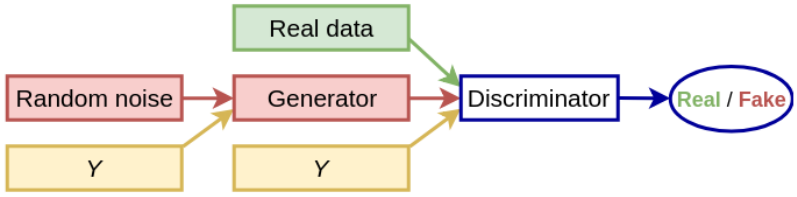

Figure 2.5: Conditional Generative Adversarial Network training schema

3 - Experiment details and Results

All the experiments are done in RL-SRL-toolbox. It is a toolbox capable of evaluating State Representation Learning methods using Reinforcement Learning. It integrates various RL algorithms along with different SRL methods in an efficient way.

3.1 Train RL Models

We train 2 reinforcement learning policies for 2 tasks in an environment called "Omnirobot_Env". The input image is a top-down view of the floor and the robot is identified by a black QR code. The room where the real-life robotic experiments are performed is lighted by surroundings windows and artificial illumination and is subject to illumination changes depending on the weather and time of the day. The robot uses 4 high level discrete actions (move left/right, move up/down in a Cartesian plane relative to the robot) rather than motor commands.

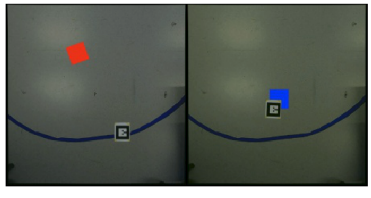

Figure 3.1: One observation of "Omnirobot_Env" in $Task_1$(left) and in $Task_2$(right).

The robot has to do two tasks:

- $Task_1$**: Target Reaching**

  From any initial position, the robot have to reach a target placed at any position on the ground in fastest possible way. The robot gets at each time step t a positive reward +1 for reaching the target (red square), a negative reward -1 for bumping in the boundary, and no reward otherwise.

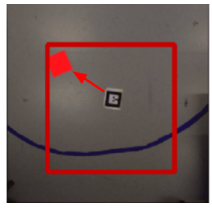

Figure 3.2: $Task_1$: Random Target Reaching

- $Task_2$**: Target Circling**

  From any initial position, the robot have to move to a distance R from the center of the image and keep navigating on the circle of radius R located at the center of image (blue tag). The robot gets at each time stet t a reward R_t defined in Eq. 3.1 (where z_t is the 2D coordinate position with respect to the center of the circle). This reward reach the highest value when the agent is on the circle and has been moving for the previous k steps. An additional penalty term of -1 is added to the reward function in case of bump with boundary. A coefficient $\lambda = 10$ is introduced to balance the behaviour.

$$R_t = \lambda * (1 - \lambda(||z_t|| - r_{circle})^2) * ||z_t - z_{t-k}||_2^2 + \lambda^2 * R_{t,bump} \tag{3.1}$$

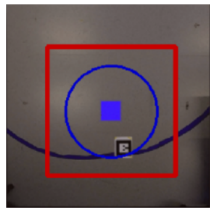

Figure 3.3: $Task_2$: Circular Movement

The RL policy of each task are trained with PPO2 algorithm for 5 millions time steps on the latent space of a State Reinforcement Learning (SRL) model. SRL model is an unsupervised learning method aims at extracting relevant features from raw sensor data. An SRL model of an environment is trained on a data set of that environment. It learns to retain useful information from the observation space for each environment, filter out irrelevant parts and thus fasten the RL convergence.In our experiment, it learn to encode the robot and target position, two enough information for the two tasks.

The data set to train the SRL model consists of frames in 250 episodes where in each episode of each task, the robot takes random actions given different initial configurations.

Base on the baseline in [9], we use splitting model for training the SRL model as following:

- $Task_1$: *autoencoder*:1:198 *reward*:1:-1 *inverse*:1:2
- $Task_2$: *autoencoder*:1:198 *inverse*:1:2

Splitting SRL model is a model compose of several models (base model and auxiliary models) calculated on preferred number of dimensions of the state representation s. To use splitting SRL model, we mention the name of the model, the weight of it's loss and the number of dimensions that it learns in the form: " $< name >:< weight >:< dimension >$ ". If "*dimension*" of a model equals -1, it uses the same dimension of the preceding model.

So for $Task_1$, the state dimension is 200 and the Total loss: $L_{total} = 1 * L_{reconstruction} + 1 * L_{Inverse} + 1 * L_{Reward}$ where Auto-encoder model (with $L_{reconstruction}$) and reward model learn from the same 198 state's dimensions and reward model learns from the left 2 dimensions. Similarly for $Task_2$.

	$Task_1$	$Task_2$
name	SRL_Task_1	SRL_Task_2
model	*srl_plits*: *autoencoder/inverse/reward*	*srl_plit*: *autoencoder/inverse*
dimensionality	*autoencoder*: 198 *inverse*: 2 *reward*: 198	*autoencoder*: 198 *autoencoder*: 2
Losses' weight	*autoencoder*:1 *inverse*:1 *reward*:1	*autoencoder*1 *inverse*:1
Batch size	32	32
State's dimensions	200	200
Training epoch	20	20
optimizer	Adam	Adam
Adam parameters	*beta_1*=0.5,*beta_2*=0.9	*beta_1*=0.5,*beta_2*=0.9
Learning rate	0.005	0.005

Table 3.1: Name, model, dimensionality, losses' weight and hyperparameters for SRL models extracting feature from both environment to serve reinforcement learning of both tasks. *beta_1* and *beta_2* are the exponential decay rate for the first and second moment estimates respectively.

	$Task_1$	$Task_2$
Algorithm	PPO2	PPO2
Srl_model	SRL_Task_1	SRL_Task_2
Number of timesteps	5 millions	5 millions
Image's size	(3,64,64)	(3,64,64)
Seed	0	0

Table 3.2: Arguments for training reinforcement learning model of both tasks.

As a result, we find the optimal policy for each task: π_{Task_1} and π_{Task_2}.

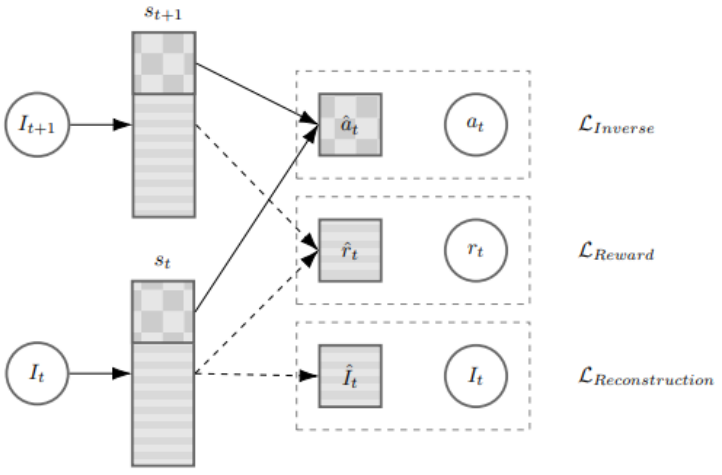

Figure 3.4: SRL Splits model: combines a reconstruction of an image I, a reward (r) prediction and an inverse dynamic models losses, using two splits of the state representation s. Arrows represent model learning and inference, dashed frames represent losses computation, rectangles are state representations, circles are real observed data, and squares are model predictions.[9]

3.2 Train Generative Models

The are two main experiments in this project:

- **Experiment 1**:Train conditional generative model on 'on-policy' data set $D_{\pi_{Task_i}}$. $D_{\pi_{Task_i}}$ is the data set generated with policy π_{Task_i} trained on $Task_i$.An on-policy π data set (D_π) is a set of tuples of 2 elements: observation and probability of action given that observation under policy π. The goal of this experiment is to find a generative model that can generate a good data set to replace an on-policy data set for later doing policy distillation.

- **Experiment 2**: Train condition generative model on $D_{R_{Task_i}}$.$D_{R_{Task_i}}$ is the data set generated by a random policy on $Task_i$. The conditions are *robot_position* and *target_position*. The goal of this experiment is to find a generative model in each environment capable of generating observations with the robot and target at any position we want.

3.2.1 Experiment 1

From π_{Task_1} and π_{Task_2}, we generate their on-policy data set: $D_{\pi_{Task_1}}$ and $D_{\pi_{Task_2}}$ for 5000 short episodes. A short episode for $Task_1$ is an episode where

the environment stops to generated the observations after the robot has reached 2 contacts with the target. A short episode for $Task_2$ is an episode where the environment stops to generated the observations after the robot has reached 75 steps (long enough to make the robot navigate a complete revolution). This setting aims at preventing many overlapping observations in each episodes and also fasten the training.

Actually the on-policy data sets $D_{\pi_{Task_1}}$ and $D_{\pi_{Task_2}}$ distribution are difficult for our generative model to learn since they are the result of stochastic policies π_{Task_1} and π_{Task_2}.

For simplicity, we generated a simplified version of $D_{\pi_{Task_1}}$ in such a way that at each episode, we restrict the robot position to be aligned horizontally or vertically to target position. So the robot are likely to take the same action toward the target. Thus the policy seem more deterministic and the on-policy data set is simpler for generative model to learn. For $D_{\pi_{Task_2}}$, we don't make change.

The generative model used in our experiment are CVAE and CGAN with two condition action a and target position t_{pos}.
We also try with different architecture:

- CVAE Resnet (see Table 4.1 and Table 4.2)
- CGAN
  - Deep Convolutional (DC)(see Table 4.3 and Table 4.4)
  - Resnet (see Table 4.5 and Table 4.6)

The hyperparameters used for the experiment are listed in the following table:

Hyperparameters	CVAE	CGAN
Batch size	128	128
Training epoch	20	100
optimizer	Adam	Adam (Both G and D)
Adam parameters	$beta\_1$=0.5,$beta\_2$=0.9	$beta\_1$=0.5,$beta\_2$=0.9

Table 3.3: Hyperparameters for training generative model for both tasks. $beta\_1$ and $beta\_2$ are the exponential decay rate for the first and second moment estimates respectively.

3.2.2 Experiment 2

We begin the experiment by generating $D_{R_{Task_1}}$ and $D_{R_{Task_1}}$. Each data set has position of robot ($ground\_truth\_states$) and $target\_position$ corresponding to each observation. The generative model used in this experiment is $CVAE$ (see Tab 4.8 and Tab 4.9 for the architecture) conditioned on $ground\_truth\_states$ and $target\_position$.

	$D_{R_{Task_1}}$	$D_{R_{Task_2}}$
Policy	Random	Random
Number of episode	250	250
Image's size	(3,64,64)	(3,64,64)

Table 3.4: Information on each data set.

We then train $CVAE_{Task_1}$ and $CVAE_{Task_2}$ on each corresponding data set.

Hyperparameters	$CVAE_{Task_1}$	$CVAE_{Task_2}$
Training data set	$D_{R_{Task_1}}$	$D_{R_{Task_2}}$
Batch size	128	128
Training epoch	100	80
optimizer	Adam	Adam
Learning Rate	0.005	0.005
State dimension	200	200
Adam parameters	$beta\_1$=0.5,$beta\_2$=0.9	$beta\_1$=0.5,$beta\_2$=0.9
Losses weight	KL_loss: 100 Reconstruction_loss: 1	KL_loss: 100 Reconstruction_loss: 1

Table 3.5: Hyperparameters for training $CVAE_{Task_1}$ and $CVAE_{Task_2}$. $beta\_1$ and $beta\_2$ are the exponential decay rate for the first and second moment estimates respectively.

3.3 Evaluate Generative model

3.3.1 Experiment 1

Since the ideal generative model that we want in this experiment is the one having capacity to generate a data set that can replace an on-policy data set D_π, so we evaluate its performance base on the similarity between the action a_G of the generated on-action image $O^{G(a_G)}$ and the action A_i such that $P(a_\pi = A_i|O^{G(a_G)})$ is the highest for A_i belong to action space $A = \{0, 1, 2, 3\}$.

Notations

- a_G : an action forwarded to the generator (or decoder) of the generative model.
- $O^{G(a_G)}$: the image generated by the Generator G (or decoder) in class a_G.
- $P(a_\pi = A_i|O)$: the probability of action A_i given observation O under policy π.

$Task_1$: Target Reaching

Model	LR	Overall Accuracy	Accuracy per Action
CVAE	10^{-4}	26.15%	[11.04%, 36.01%, 12.09%, 44.20%]
CVAE split	10^{-4}	25.10%	[07.20%, 39.20%, 14.00%, 40.00%]

Table 3.6: Benchmark of CVAE for $Task_1$ with $Random\_seed = 0$.

Model	LR_G	LR_D	Overall Accuracy	Accuracy per Action
DC	2.10^{-4}	2.10^{-4}	26.15%	[19.03%, 33.90%, 38.20%, 13.20%]
DC	10^{-5}	5.10^{-5}	24.90%	[04.10%, 50.60%, 01.40%, 43.50%]
Resnet	2.10^{-4}	2.10^{-4}	25.12%	[98.60%, 00.00%, 00.40%, 01.50%]
Resnet	10^{-5}	5.10^{-5}	24.60%	[03.40%, 58.90%, 25.30%, 10.80%]

Table 3.7: Benchmark of CGAN for $Task_1$ with $Random\_seed = 0$.

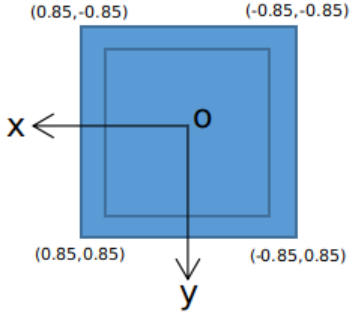

Figure 3.5: Coordinate of the environment

CVAE or CVAE split,does not generated a visible robot, however it generate generally clear target with preferable position as shown in Fig. 3.6.

CGAN (Resnet or DC) can generate visible target and little blur robot. However in some observations, the target and/or the robot appear twice.

$Task_2$: Target Circling

Model	LR	Overall Accuracy	Accuracy per Action
CVAE	10^{-4}	21.95%	[77.00%, 00.10%, 10.30%, 00.40%]
CVAE split	10^{-4}	21.60%	[73.80%, 00.00%, 12.00%, 00.60%]

Table 3.8: Benchmark of CVAE for $Task_2$ with $Random\_seed = 0$.

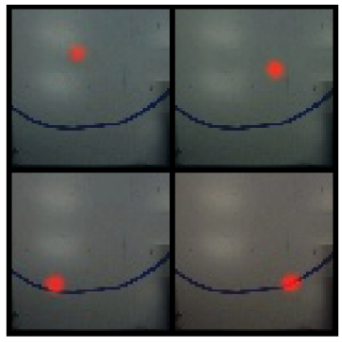

Figure 3.6: Four generated observations of $Task_1$ by CVAE split model correspond to four target positions: (0.27,-0.63), (0.63,0.48),(-0.34,-0.39),(-0.59,0.46) respectively form left to right and from up to down.

Model	LR_G	LR_D	Overall Accuracy	Accuracy per Action
DC	2.10^{-4}	2.10^{-4}	25.78%	[05.90%, 02.70%, 00.75%, 93.80%]
DC	10^{-5}	5.10^{-5}	25.57%	[05.30%, 02.70%, 00.55%, 93.75%]
Resnet	2.10^{-4}	2.10^{-4}	24.75%	[00.00%, 34.60%, 62.00%, 02.40%]
Resnet	10^{-5}	5.10^{-5}	24.92%	[00.20%, 20.20%, 78.90%, 00.40%]

Table 3.9: Benchmark of CGAN for $Task_2$ with $Random\_seed = 0$.

CGAN (Resnet or DC) generates in general clear target and robot position, however there are not varies. It can possibly be the collapse mode in training GAN.

Notation

- Overall Accuracy: Overall accuracy of correct-labeled images generation(over 4000 times of the 4 actions)
- Accuracy per Action: Accuracy of correct-action [0,1,2,3] images generation (over 100 times on each action). (0:"move left" ,1:"move right" ,2:"move down": ,3:"move up")
- CVAE split: is a method of decoupling features with 2 others auxiliary models: inverse and reward model. The decoupling feature of CVAE split is given in the form : –losses 1:CVAE:198 1:reward:-1 1:inverse:2

Training GAN

Many GAN models suffer the following major problems:

- Non-convergence: the model parameters oscillate, destabilize and never converge,

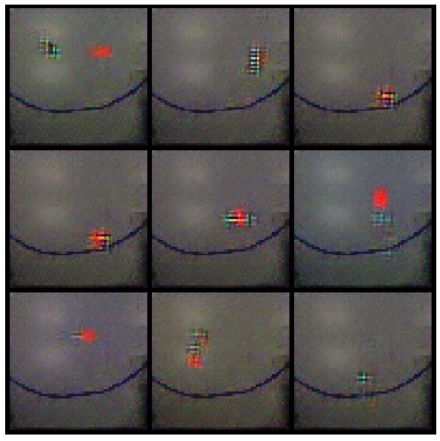

Figure 3.7: Generated observation by CGAN (Resnet) with $LR_D = 5.10^{-5}$ and $LR_G = 10^{-5}$

- Mode collapse: the generator collapses which produces limited varieties of samples,
- Diminished gradient: the discriminator gets too successful that the generator gradient vanishes and learns nothing,Unbalance between the generator and discriminator causing over-fitting, and highly sensitive to the hyper-parameters selections.

In [13], Soumith propose several tips and tricks in training GAN to prevent from the problems above.Some of his tips used in our experiment are :

- Normalize the inputs
- Use a spherical Z
- Batch Normalization
- Avoid Sparse Gradients: ReLU, MaxPool
- Use label smoothing
- Use the ADAM Optimizer
- Track failures early
- Use labels for training (action and target position)

Inspired from [3], we also include *Spectral Normalization* to the discriminator and generator to stabilize the training.

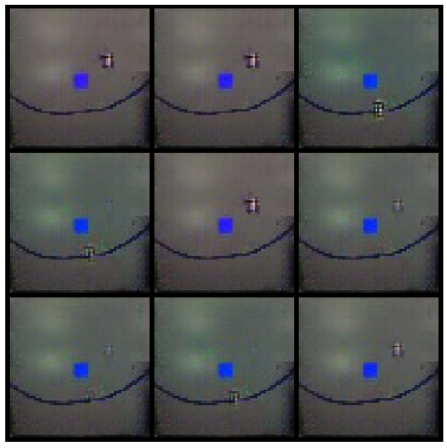

Figure 3.8: Generated observation by CGAN (Resnet) with $LR_D = 5.10^{-5}$ and $LR_G = 10^{-5}$

The learning rate is the most important hyperparameter for tuning neural networks. $LR_G = LR_D = 0.0002$ is inspired by the experiment in [8] for the same batch size and image size.

In [6], Marco Pasini recommend choosing a higher learning rate for the discriminator and a lower one for the generator: in this way the generator has to make smaller steps to fool the discriminator and does not choose fast, not precise and not realistic solutions to win the adversarial game.

3.3.2 Experiment 2

To validate the model, we use cross validation.

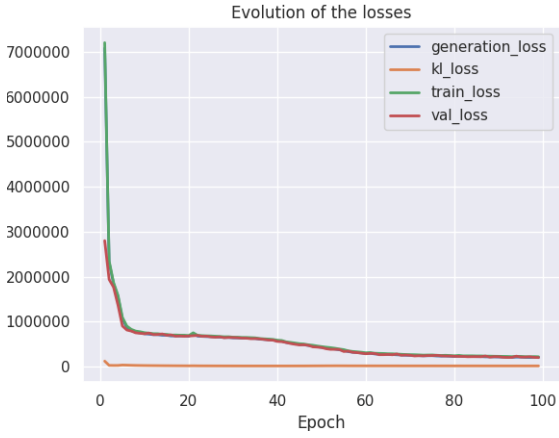

Figure 3.9: Losses evolution of training $CVAE_{Task_1}$ in experiment 2.

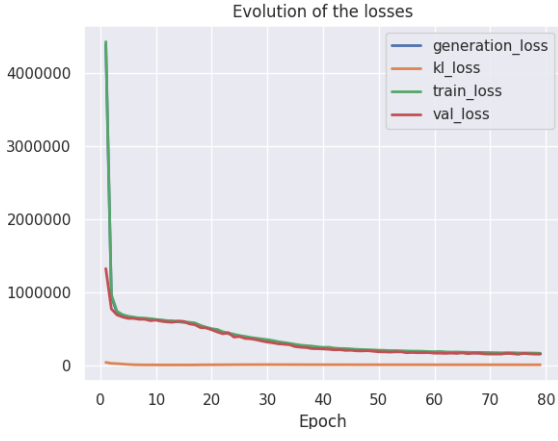

Figure 3.10: Losses evolution of training $CVAE_{Task_2}$ in experiment 2.

3.4 Policy Distillation

There are 3 steps in doing policy distillation:

1. **Generating data set from replay of both tasks**
2. **Merging data sets**
3. **Train policy distillation**

3.4.1 Experiment 1

1. **Generating data set from replay of both tasks:**
   For both tasks, we use CGAN resnet as generative model since it generates the the clearest target and robot among other models. After the training, we have $CGAN_{\pi_{Task_1}}$ and $CGAN_{\pi_{Task_2}}$. From each model, we generate a "generated_on-policy" data set of 8000 frames: $D_{CGAN_{\pi_{Task_1}}}$ for $Task_1$ and $D_{CGAN_{\pi_{Task_2}}}$ for $Task_2$.

   $D_{CGAN_{\pi_{Task_1}}}$ has the same type of element as "on-policy" data set ($D_{\pi_{Task_1}}$). It is composed of tuples of 2 elements: observation and probability of action given that observation under policy π_Task_1.Each observation is an RGB image of size (3,64,64) generated by $CGAN_{\pi_{Task_1}}$ on condition to an action and a target position. The target positions are sampling from a uniform distribution between the minimum and maximum of target's abscissa and ordinate.The actions are chosen equally between each movement: 2000 for each movement. The probability of action of each observation is estimated by the policy π_{Task_1} given that observation. $D_{CGAN_{\pi_{Task_2}}}$ is generated in the same manner.

	Constants of *Omnirobot_env*
Target's boundary	Target_MIN_X = -0.7 Target_MAX_X = 0.7 Target_MIN_Y = -0.7 Target_MAX_Y = 0.7
Robot's boundary	MIN_X = -0.85 MAX_X = 0.85 MIN_Y = -0.85 MAX_Y = 0.85

Table 3.10: Constants of *Omnirobot_env*.

	$D_{\pi_{Task_1}}$	$D_{\pi_{Task_2}}$
Number of actions	action "0": 2000 action "1": 2000 action "2": 2000 action "3": 2000	
State sampling	$z_i \sim Normal(0,1)$	
Target's position sampling	$x \sim Uniform[Target\_MIN\_X, Target\_MAX\_X)$ $y \sim Uniform[Target\_MIN\_Y, Target\_MAX\_Y)$	$x = 0$ $y = 0$
Robot's position sampling	$x \sim Uniform[MIN\_X, MAX\_X)$ $y \sim Uniform[MIN\_Y, MAX\_Y)$	
Image's size	(3,64,64)	

Table 3.11: Actions sampling and image's size of each generated data set.

2. **Merging data sets:**
   We merge $D_{CGAN_{\pi_{Task_1}}}$ and $D_{CGAN_{\pi_{Task_2}}}$ as $D_{CGAN_{\pi_{Task_{12}}}}$.

3. **Train policy distillation:**
   We train policy distillation on $D_{CGAN_{\pi_{Task_{12}}}}$ with the following configurations:

   - Adaptive temperature = False
   - 20 epochs
   - raw pixel
   - Policy model: CustomCNN (See Table 4.7)
   - BATCH_SIZE = 8
   - Learning rate = 1e-3
   - FINE_TUNING = False

   As a result, we have $\pi_{Task_{12}}$ that is supposed to do both tasks.

3.4.2 Experiment 2

1. **Generating data set from replay of both tasks**

	$D_{\pi_{Task_1}}$	$D_{\pi_{Task_2}}$
Random seed	0	0
Number of episode	100	100
State sampling	$z_i \sim Normal(0,1)$	
Target's position sampling in each episode	$x \sim Uniform[Target\_MIN\_X, Target\_MAX\_X)$ $y \sim Uniform[Target\_MIN\_Y, Target\_MAX\_Y)$	$x = 0$ $y = 0$
Robot's position sampling in each episode	grid walker	grid walker
Grid walker's step	0.1	0.1
Image's size	(3,64,64)	

Table 3.12: Sampling method of each data set in experiment 1.

Fig.3.11 and Fig.3.12 are the result of fixed value of latent space variable z.We observe that each z has effect on the color of background ,target and robot. This benefit *Domain_Randomization*. However,it also affect of the blurriness and the accuracy of the position of robot and target in the generated image.

For now, we decide not to pay attention on *Domain_Randomization*. In other word, we fixed the value of z where we observe good quality and the accuracy on generated images.

We first generate the data set with z sampled from Normal Distribution with $RandomSeed = 0$. We find a frame that seem to fulfill our demand and we choose the value of z used for that frame. For $D_{\pi_{Task_1}}$ and $D_{\pi_{Task_1}}$, we use z at $frame\_107$ and $frame\_14$ respectively.

Figure 3.11: Oberservations in a episod enerated by $CVAE_{Task_1}$ with a fixed value of latent variable z.The sampled target is at position (0.50, 0.49) and the robot's positions are sampled with grid walker of step 0.28. With this step, there are 25 observations in an episode.

2. **Merging data sets**
   We merge $D_{CVAE_{\pi_{Task_1}}}$ and $D_{CVAE_{\pi_{Task_2}}}$ as $D_{CVAE_{\pi_{Task_{12}}}}$.

3. **Train policy distillation**
   We train policy distillation on $D_{CVAE_{\pi_{Task_{12}}}}$ with the following configurations:

   - Adaptive temperature = True
   - Temperature : $Task_1 : 0.1$, $Task_2 : 0.1$
   - 20 epochs
   - raw pixel
   - Policy model: CustomCNN (See Table 4.7)
   - BATCH_SIZE = 8
   - Learning rate = 1e-3

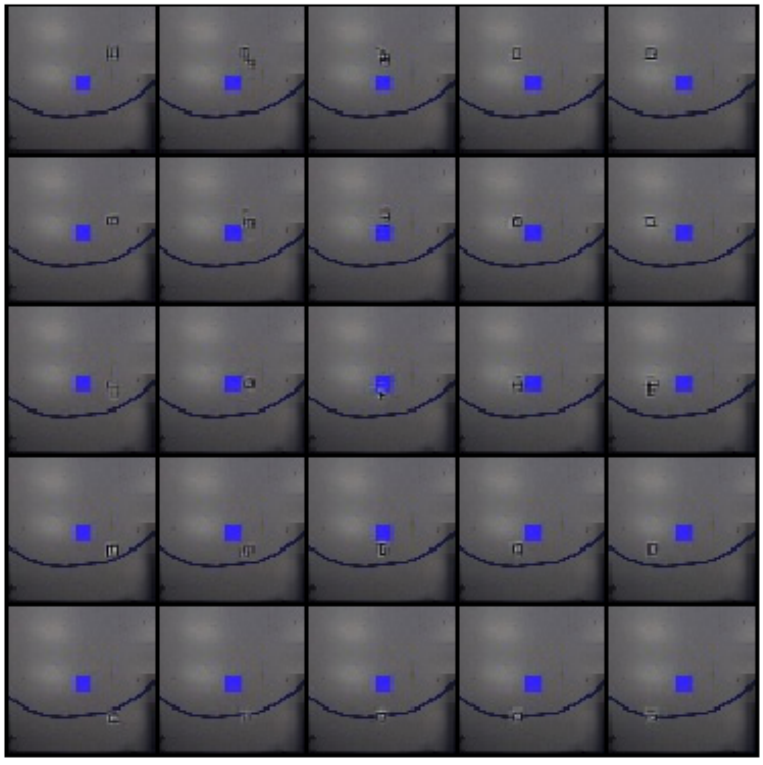

Figure 3.12: Oberservations in a episod enerated by $CVAE_{Task_2}$ with a fixed value of laten variable z. The sampled target is placed at the center of the arena and the robot's positions are sampledwith grid walker of step 0.28. With this step, there are 25 observations in an episode.

- FINE_TUNING = False

As a result, we have $\pi_{Task_{12}}$ that is supposed to do both tasks.

3.5 Evaluate Distilled Model

We evaluate the student model $\pi_{Task_{12}}$ for both tasks and compare its reward with that of the teachers: π_{Task1} and π_{Task_2}.

	2 episodes	4 episodes	6 episodes	8 episodes	9 episodes
$Teacher\ \pi_{Task1}$	211.5	201	211.83	216.75	179.22
$Student\_experiment\_1$	-242.5	-240.75	-241	-241.5	-241
$Student\_experiment\_2$	-240.5	-243.75	-244.33	-243.75	-244.35

Table 3.13: Reward of Teacher, Student_experiment_1 and Student_experiment_2 for $Task_1$: Target_Reaching on 2500 time steps with $random\_seed = 0$

	2 episodes	4 episodes	6 episodes	8 episodes	9 episodes
$Teacher\ \pi_{Task2}$	1824.92	1868.9	1874.85	1878	1881.69
$Student\_experiment\_1$	-24134.34	-24290.95	-24130.59	-24145.42	-24219.86
$Student\_experiment\_2$	-23758.02	-24261.76	-24111.13	-24129.34	-24150.26

Table 3.14: Reward of Teacher, Student_experiment_1 and Student_experiment_2 for $Task_2$: Target_Circling on 2500 time steps with $random\_seed = 0$

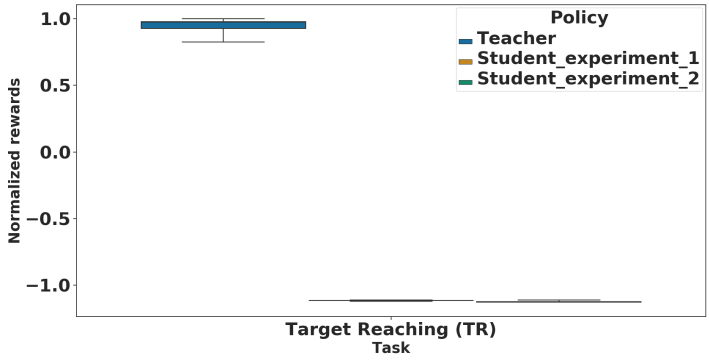

Figure 3.13: Box-plot of the reward of the Teacher, Student_experiment_1 and Student_experiment_2 for $Task_1$ on 2500 time steps with $random\_seed = 0$.(Teacher: left , Student_in_experiment_1: middle,Student_in_experiment_2: right)

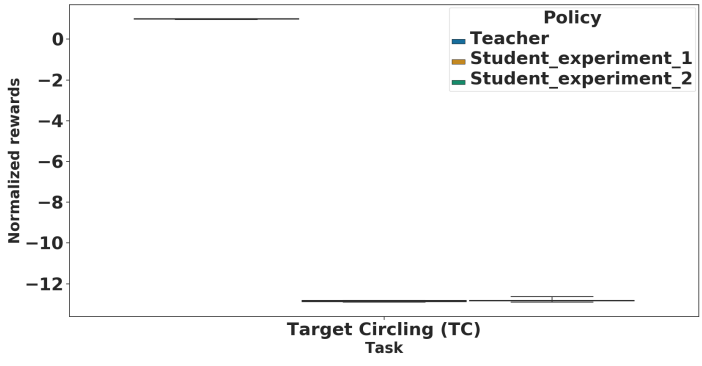

Figure 3.14: Box-plot of the reward of the Teacher, Student_experiment_1 and Student_experiment_2 for $Task_2$ on 2500 time steps with $random\_seed = 0$.(Teacher: left ,Student_experiment_1: middle, Student_experiment_2: right)

4 - Conclusion

In conclusion, for experiment 1, even thought we have not found any good enough model having good generative ability over on-policy data set,we have come to find one (CVAE split) that can generated observations with preferable target's position(see Fig. 3.6) for $Task_1$: $Target\_Reaching$. It ensures that we can generate observations with preferable target's positions which is an indispensable setting for policy distillation. We also have come to find a model (CGAN Resnet) that can generates in general clear target and robot in both tasks (see Fig. 3.7 and Fig. 3.8).

The result of experiment 1, has inspired us to do experiment 2. As a result, we have come to generate observation with target and robot at any position we want.

Furthermore, we have come to scale the models for all experiments (State Representation Learning, Reinforcement Learning, Generative Replay, Policy distillation etc.) with preferable image size. As a matter of fact, we have used the image shape : (3,64,64) instead of (3,224,224) in our experiments.It speeds up the training from 4 to 6 times depending on the model's architecture. In addition, for experiment, we also give the possibility to train the generative model without condition, one condition (action) or two condition (action and target's position) which may help us understand more whether it is caused by the action or target's position or both that make our generative model poor.

4.1 Challenges

There are several challenges we have encountered in experiment 1:

- Difficulty in training GAN:
  GAN is highly sensitive to hyperparameters selection. Without proper learning rates, the training become easily unstable and never converge. Cost function may not converge using gradient decent.

- Complex data set distribution:
  Another challenge we suspected is the complexity of the training data set distribution for conditional model. In our case, it is the "on-policy" data set : D_π. It maybe is too complicated for the generator. The action (one

label) of each data point is estimated under a trained stochastic policy π which is a deep neural network. So, to generate observation that correspond the correct on-policy action, the generative model need to do two tasks: generate natural images and remember the policy π. This ability demands a "proper powerful model". A model with simple architecture may not solve the problem. Furthermore, the action of the robot depend on the target position, so conditioning the generative model only on the action is not sufficient,the target position should also be conditioned. So it becomes a two-conditional model.

4.2 Improvement and Future Work

There are several improving ideas and task that might be interesting for future work:

- Try with deterministic policy for experiment 1:
  Instead of using stochastic policy, we may first try with deterministic policy, so the "on-policy" data set is less complicated.For $Task_1$, it is actually not complicated to find a deterministic policy. For example, if the robot has the same abscissa or ordinate from the target, it takes only one action on ordinate or the abscissa direction to reach the target. If it has different abscissa and ordinate from the target, it need to first navigate on the ordinate direction to have the same abscissa and then navigate on the abscissa direction to reach the target.

- Implement FID and Inception score for GAN for experiment 1:
  We have not implement any metric to measure the performance of GAN. What we have done so far is: we save the model at every epochs during training. We save the generated image of every epochs.At some epochs, the model generates only target.Some other, it generates one target and 2 robots. Some other, it generates 2 targets and two robots. And the robot and the target are not always clear to see at each epoch. We looked at these images and choose the model of an epoch in which the generated image has only one and clear enough robot and only one and clear enough target. After that, we delete the other models to prevent from much disk space occupation. Implementing FID and Inception score may give us better insight of the best GAN model during training.

- Try with other environments for experiment 1:
  Try with other environments like "mobile_robot_extreme" [4] since the environment is less realistic than that "Omnirobot_env".

- Evaluation of models in policy distillation:
  Averaging the reward of each model at least with 5 evaluations of different *Random_seed*

Annexes

Here are the tables that represent the architectures of the models used in our experiments:

Layer	Architecture
1	Conv2d(*img_chanel*+ *action_chanel* (+ *target_pos_chanel*), 64, kernel_size=7, stride=2, padding=3, bias=False) + BN + ReLU
2	MaxPool2d(kernel_size=3, stride=2, padding=1)
3	Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=False) + BN + ReLU
4	MaxPool2d(kernel_size=3, stride=2)
5	Conv2d(64, 64, kernel_size=3, stride=2, padding=1, bias=False) + BN + ReLU
6	MaxPool2d(kernel_size=3, stride=2)
7	Linear(*out_shape_height***out_shape_width**64, *state_dim*)

Table 4.1: Encoder's architecture of CVAE Resnet (CNNCVAE_NEW). It is inspired by the $CNNVAE$ model in this Github Repository [4]. In case we condition the action and target position, we add *target_pos_chanel* on the first layer, otherwise don't. (*img_chanel*=3, *action_chanel*=4 , *target_pos_chanel*=1, $out\_shape\_height = out\_shape\_width = 1$)

Layer	Architecture
1	Linear(*state_dim*+ *action_dim* (+ *target_pos_dim*), *out_shape_height***out_shape_width**64)
2	Reshape(64, *out_shape_height*, *out_shape_width*)
3	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
4	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
5	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
6	ConvTranspose2d(64, *img_chanel*, kernel_size=4, stride=2)+ Tanh

Table 4.2: Decoder's architecture of CVAE Resnet (CNNCVAE_NEW). It is inspired by the $CNNVAE$ model in this Github Repository [4]. In case we condition the action and target position, we add *target_pos_dim* on the first layer, otherwise don't. (*img_chanel*=3, *action_dim*=4 , *target_pos_dim*=2, $out\_shape\_height = out\_shape\_width = 1$)

Conditions	Sub Model	Layer	Architecture
$Both - Cases$	deconv1_1	1	ConvTranspose2d($state\_dim$, 128*4, 4, 1, 0, False)+ BN+ ReLu
$Only\_Action$	deconv1_2	1	ConvTranspose2d($actiom\_dim$, 128*4, 4, 1, 0, False)+ BN+ ReLu
$Action\_and\_Target$	deconv1_2	1	ConvTranspose2d($actiom\_dim$, 128*2, 4, 1, 0, False)+ BN+ ReLu
	deconv1_3	1	ConvTranspose2d($target\_pos\_dim$, 128*2, 4, 1, 0, False)+ BN+ ReLu
$Both - Cases$	deconv2	1	ConvTransposeSN2d(128*8, 128*4, 4, 2, 1, False)+ BN+ ReLu
		2	ConvTransposeSN2d(128*4, 128*2, 4, 2, 1, False)+ BN+ ReLu
		3	ConvTransposeSN2d(128*2, 128, 4, 2, p1, False)+ BN+ ReLu
		4	ConvTransposeSN2d(128, img_chanel, 4, 2, 1, False)+ Tanh

Table 4.3: DC Generator's architecture of CGAN. It is inspired by this Github Repository [7]. There are two cases of conditions: 1) *Only_Action* where *action* is the only one condition 2) *Action_and_Target* where *action* and *targetposition* are conditioned to the model.(img_chanel=3, $action\_dim$=4 , $target\_pos\_dim$=2, $out\_shape\_height = out\_shape\_width = 1$)

Conditions	Sub Model	Layer	Architecture
$Both - cases$	conv1_1	1	Conv2d(img_chanel, 128/2, 4, 2, 1, False)+ LeakyReLu
$Only\_Action$	conv1_2	1	Conv2d($action\_chanel$, 128/2, 4, 2, 1, False)+ LeakyReLu
$Action\_and\_target$	conv1_2	1	Conv2d($action\_chanel$, 128/4, 4, 2, 1, False)+ LeakyReLu
	conv1_3	1	Conv2d($target\_pos\_chanel$, 128/4, 4, 2, 1, False)+ LeakyReLu
$Both - Cases$	conv2	1	Conv2d(128, 128*2, 4, 2, 1, False)+ BN+ LeakyReLu
		2	Conv2d(128*2, 128*4, 4, 2, 1, False)+ BN+ LeakyReLu
		3	Conv2d(128*4, 128*8, 4, 2, 1, False)+ BN+ LeakyReLu
		4	Conv2d(128*8, 1, 4, 1, 0, False)+ Sigmoid

Table 4.4: DC Discriminator's architecture of CGAN. It is inspired by this Github Repository [7]. There are two cases of conditions: 1) *Only_Action* where *action* is the only one condition 2) *Action_and_Target* where *action* and *targetposition* are conditioned to the model.(img_chanel=3, $action\_chanel$=4 , $target\_pos\_chanel$=1, $out\_shape\_height = out\_shape\_width = 1$)

Layer	Architecture
1	Linear($state\_dim$+ $class\_dim$ (+ $target\_pos\_dim$), out_shape_height*out_shape_width*64)
2	Reshape(64, out_shape_height, out_shape_width)
3	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
4	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
5	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
6	ConvTranspose2d(64, img_chanel, kernel_size=4, stride=2)+ Tanh

Table 4.5: Resnet Generator's architecture of CGAN. It is inspired by the *GeneratorResNet* model in this Github Repository [4]. There are two cases of conditions: 1) $Only_Action$ where *action* is the only one condition 2) *Action_and_Target* where *action* and *targetposition* are conditioned to the model.(img_chanel=3, $action\_dim$=4 , $target\_pos\_dim$=2, $outshape = summary(encoder\_conv, img\_shape, False)$, $out\_shape\_height, out\_shape\_width = outshape[-2:]$)

Condition	Sub Model	Layer	Architecture
Both − cases	*modules_list*	1	Conv2d(*img_channel*, 16, kernel_size=4, stride=2, padding=1)+ LeakyReLu
		2	Conv2d(16*1, 16*2, kernel_size=4, stride=2, padding=1)+ LeakyReLu
		3	Conv2d(16*2, 16*4, kernel_size=4, stride=2, padding=1)+ LeakyReLu
		4	Conv2d(16*4, 16*8, kernel_size=4, stride=2, padding=1)+ LeakyReLu
		5	Conv2d(16*8, 16*8, kernel_size=4, stride=2, padding=1)+ LeakyReLu
		6	Conv2d(16*8, 16*4, kernel_size=3, stride=1, padding=1)
	before_last	1	Linear(*in_features*, *state_dim*, bias=True)
Only_Action	*last*	1	Linear(*state_dim*+*action_dim*, 1, bias=True)+ Sigmoid
Action_and_Target	*last*	1	Linear(*state_dim*+*action_dim*+*target_pos_dim*, 1, bias=True)+ Sigmoid

Table 4.6: Resnet Discriminator's architecture of CGAN. It is inspired by the *DiscriminatorResNet* model in this Github Repository [4]. In case we condition the action and target position, we add *target_pos_chanel* on the first layer, otherwise don't. (*img_chanel*=3, *action_chanel*=4 , *target_pos_chanel*=1,*out_shape_height* = *out_shape_width* = 1, *in_features* = 256)

Sub Model	Layer	Architecture
conv_layers	1	Conv2d(*img_chanel*, 64, kernel_size=7, stride=2, padding=3, bias=False) + BN + ReLU
	2	MaxPool2d(kernel_size=3, stride=2, padding=1)
	3	Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=False) + BN + ReLU
	4	MaxPool2d(kernel_size=3, stride=2)
	5	Conv2d(64, 64, kernel_size=3, stride=2, padding=1, bias=False) + BN + ReLU
	6	MaxPool2d(kernel_size=3, stride=2)
linear_layer	1	Linear(*out_shape_height***out_shape_width**64, *state_dim*)

Table 4.7: CustomCNN architecture.(outshape=summary(conv_layers, img_shape, False), out_shape_height = out_shape_width=1)

Layer	Architecture
1	Conv2d(*img_chanel*+ *robot_pos_chanel* + *target_pos_chanel*, 64, kernel_size=7, stride=2, padding=3, bias=False) + BN + ReLU
2	MaxPool2d(kernel_size=3, stride=2, padding=1)
3	Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=False) + BN + ReLU
4	MaxPool2d(kernel_size=3, stride=2)
5	Conv2d(64, 64, kernel_size=3, stride=2, padding=1, bias=False) + BN + ReLU
6	MaxPool2d(kernel_size=3, stride=2)
7	Linear(*out_shape_height***out_shape_width**64, *state_dim*)

Table 4.8: Encoder's architecture of CNNCVAE Resnet for experiment 2. It is inspired by the *CNNVAE* model in this Github Repository [4]. (*img_chanel*=3, *robot_pos_chanel*=1 , *target_pos_chanel*=1, *out_shape_height* = *out_shape_width* = 1)

Layer	Architecture
1	Linear(*state_dim*+ *robot_pos_dim* + *target_pos_dim*, *out_shape_height***out_shape_width**64)
2	Reshape(64, *out_shape_height*, *out_shape_width*)
3	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
4	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
5	ConvTranspose2d(64, 64, kernel_size=3, stride=2)+ BN+ ReLu
6	ConvTranspose2d(64, *img_chanel*, kernel_size=4, stride=2)+ Tanh

Table 4.9: Decoder's architecture of CVAE Resnet for experiment 2. It is inspired by the *CNNVAE* model in this Github Repository [4]. (*img_chanel*=3, *robot_pos_dim*=2, *target_pos_dim*=2, *out_shape_height* = *out_shape_width* = 1)

Bibliography

[1] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1406.2661 (June 2014), arXiv:1406.2661. arXiv: 1406.2661 [stat.ML].

[2] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *CoRR* abs/1612.00796 (2016). arXiv: 1612.00796. URL: http://arxiv.org/abs/1612.00796.

[3] Takeru Miyato et al. “Spectral Normalization for Generative Adversarial Networks”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=B1QRgziT-.

[4] ncble. *srl-zoo*. https://github.com/ncble/robotics-rl-srl. 2019.

[5] German Ignacio Parisi et al. “Continual Lifelong Learning with Neural Networks: A Review”. In: *CoRR* abs/1802.07569 (2018). arXiv: 1802.07569. URL: http://arxiv.org/abs/1802.07569.

[6] Marco Pasini. *10 Lessons I Learned Training GANs for one Year*. 2019. URL: https://towardsdatascience.com/10-lessons-i-learned-training-generative-adversarial-networks-gans-for-a-year-c9071159628.

[7] pytorch. *examples*. https://github.com/pytorch/examples/tree/master/dcgan. 2019.

[8] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016. 2015. URL: http://arxiv.org/abs/1511.06434.

[9] Antonin Raffin et al. “Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics”. In: *CoRR* abs/1901.08651 (2019). arXiv: 1901.08651. URL: http://arxiv.org/abs/1901.08651.

[10] Andrei A. Rusu et al. “Policy Distillation”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016. URL: http://arxiv.org/abs/1511.06295.

[11] Andrei A. Rusu et al. “Policy Distillation”. In: *CoRR* abs/1511.06295 (2016).

[12] Andrei A. Rusu et al. “Progressive Neural Networks”. In: *CoRR* abs/1606.04671 (2016). arXiv: 1606.04671. URL: http://arxiv.org/abs/1606.04671.

[13] soumith. *ganhacks*. https://github.com/soumith/ganhacks. 2016.

[14] René Traoré et al. “DisCoRL: Continual Reinforcement Learning via Policy Distillation”. In: *CoRR* abs/1907.05855 (2019). arXiv: 1907.05855. URL: http://arxiv.org/abs/1907.05855.

List of Figures

2.1 Policy Distillation Schema 13
2.2 Variational Autoencoder schema 14
2.3 Conditional Variational Autoencoder schema 15
2.4 Generative Adversarial Network schema 15
2.5 Conditional Generative Adversarial Network training schema . . 16

3.1 One observation of "Omnirobot_Env" in $Task_1$(left) and in $Task_2$(right). 17
3.2 $Task_1$: Random Target Reaching 18
3.3 $Task_2$: Circular Movement 18
3.4 SRL Splits model: combines a reconstruction of an image I, a reward (r) prediction and an inverse dynamic models losses, using two splits of the state representation s. Arrows represent model learning and inference, dashed frames represent losses computation, rectangles are state representations, circles are real observed data, and squares are model predictions.[9] 20
3.5 Coordinate of the environment 23
3.6 Four generated observations of $Task_1$ by CVAE split model correspond to four target positions: (0.27,-0.63), (0.63,0.48),(-0.34,-0.39),(-0.59,0.46) respectively form left to right and from up to down. 24
3.7 Generated observation by CGAN (Resnet) with $LR_D = 5.10^{-5}$ and $LR_G = 10^{-5}$ 25
3.8 Generated observation by CGAN (Resnet) with $LR_D = 5.10^{-5}$ and $LR_G = 10^{-5}$ 26
3.9 Losses evolution of training $CVAE_{Task_1}$ in experiment 2. 27
3.10 Losses evolution of training $CVAE_{Task_2}$ in experiment 2. 27
3.11 Oberservations in a episod enerated by $CVAE_{Task_1}$ with a fixed value of latent variable z.The sampled target is at position (0.50, 0.49) and the robot's positions are sampled with grid walker of step 0.28. With this step, there are 25 observations in an episode. 30

3.12 Oberservations in a episod enerated by $CVAE_{Task_2}$ with a fixed value of laten variable z. The sampled target is placed at the center of the arena and the robot's positions are sampledwith grid walker of step 0.28. With this step, there are 25 observations in an episode. 31
3.13 Box-plot of the reward of the Teacher, Student_experiment_1 and Student_experiment_2 for $Task_1$ on 2500 time steps with $random\_seed = 0$.(Teacher: left , Student_in_experiment_1: middle,Student_in_experiment_2: right) 32
3.14 Box-plot of the reward of the Teacher, Student_experiment_1 and Student_experiment_2 for $Task_2$ on 2500 time steps with $random\_seed = 0$.(Teacher: left ,Student_experiment_1: middle, Student_experiment_2: right) 32