SAPIENZA
UNIVERSITÀ DI ROMA

State Of Art Of Generative Models in robotics Applications
Master thesis in engineering in computer science

Facoltà Ingegneria dell'informazione, informatica e statistica

Corso di Laurea Magistrale in Ingegneria Informatica

Candidate

Milad Kiwan
ID number 1164659

Thesis Advisors

Prof. Daniele Nardi
Dr. Francesco Riccio

Academic Year 2019/2020

Thesis defended on 20 April 2020
in front of a Board of Examiners composed by:

Prof. Lenzerini Maurizio (chairman)
Prof. Nardi Daniele
Prof. Giorgio Grisetti
Prof. Comminiello Danilo
Prof. Massimo Mecella
Dr. Scardapane Simone
Dr. Anagnostopoulos Aristidis

State Of Art Of Generative Models in robotics Applications Master thesis in engineering in computer science
Master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: April 18, 2020

Author's email: miladkiwan@hotmail.com

Dedication.

Abstract

Keywords: *Generative models, Generative Adversarial Networks, Reinforcement Learning, Variational Autoencoders, Robotics*

This thesis makes an overview on the most used architectures that have been designed to achieve generative models, and show some of the recent frameworks and algorithms in robotics applications that make use of these generative models. It presents as well how these models assisted the existing frameworks like reinforcement learning that had the lead in succession in robotics applications in the last years. The combination between generative models and reinforcement learning in problems like imitation learning and learn from demonstration has led to achieve better performing frameworks.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my Professor and advisor Prof. Daniele Nardi and his assistent Francesco Riccio for the sustained support of my masters study and related research, motivation, and immense knowledge.

Also, I would like to thank all my friends for their motivation and nice refresh breaks during the day.

A big thanks to my parents for their endless encouragement, support, and patience for being far from them.

Contents

Introduction

The use of AI and machine learning is already pervasive in our daily lives, all humans who have to do with computers or smart phones make use of it, sometimes even without knowing it, using real-time navigation to surfing the web, checking Google Maps before venturing out to gauge current traffic and fastest routes to the destination, or by simply turning on the camera to take a picture. Machine learning algorithms are often categorized as supervised or unsupervised. Supervised learning can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. In contrast, unsupervised machine learning algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system does not figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data. Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training. Then there is reinforcement learning which is a learning method that interacts with its environment by producing actions and discovers errors or rewards. The last and most recent field in machine learning is the generative

modeling which is a branch of unsupervised learning that describes how a dataset is generated, in terms of a probabilistic model. By sampling from this model, we are able to generate new data. ″Generative model‟is also used to describe a model that outputs variable or instances that has nothing to do with input probability distributions. The main difficulty the developers face implementing the is the training process, generative modeling requires large amount of data and it is very hard to train, though when it works, it works really good. The trick is that the neural networks, that the generative models are composed of, have a number of parameters significantly smaller than the amount of data we train them on, so the models are forced to discover and efficiently internalize the essence of the data in order to generate it. There is an immense effort in machine learning and statistics to develop accurate and scalable probabilistic models of data. Such models are called upon whenever we are faced with tasks requiring probabilistic reasoning, such as prediction, missing data imputation and uncertainty estimation; or in simulation-based analyses, common in many scientific fields such as genetics, robotics and control that require generating a large number of independent samples from the model. The benefit of employing a generative model lies in its wide applicability to the problem of inference. For example, many robotic tasks include the problem of evaluating the likelihood of an observation of the environment given some piece of relevant information, such as the location of the camera, a particular object model hypothesis, or even raw images. However, generative models in robotics are used to solve various issues faced during the implementation of applications in robotics field to design intelligent machines that can help and assist humans in their day-to-day lives. Such as learn from demonstrations or sometimes called imitation learning, which is a paradigm for enabling robots to autonomously perform new tasks, help reinforcement learning to learn from high-dimensional state space, lean wild set of tasks, and improve polices either with or without demonstrations. The lack of data as well is one of the main efficiency and stability is one of the main challenges in end-to-end methods like those of *″reinforcement learning‟*. It is hard to talk about robotics without mention reinforcement learning, which as a field, has had the major success in past few years. RL offers to robotics a set of tools and algorithms that can

assist to design complicated robotic behavior to achieve sophisticated tasks. More recently, another field in machine learning that had the lead in success as well is *"Generative Adversarial Networks"* this great idea of gaming networks, invented by Ian Goodfellow, has been exploit in RL to achieve more complicated frameworks that, as we will see later in this work.

During this work we will talk about the main techniques and algorithms used to perform generative models that have assisted the robotics applications to achieve the respective tasks, and go through some of the frameworks and papers that we deemed most interesting, and showed the importance role the generative modeling plays in the most recent robotics applications. Afterward we will discuss these frameworks explaining their functioning and how the generative models were employed to become useful in the various settings. Then we will show the results obtained from this employment, and argue the effects and the limits on these applications. At the end we will talk about the conclusions extracted during the extension of this survey, and offer some thoughts about how might the generative models in robotics extend and where would be the direction of the future works.

1 Background

Variational Auto-Encoder (VAE) Structure

One of the reasons for which the generative models have been employed in different type of applications is the powerful and utility of VAE, where they are used to either solve issues in AI like image reconstruction and generation, achieve better results, reduce the computational complexity due to the high dimensionality of the data, find latent space, reduce dimensionality, extract and represent features or learn density distribution of the dataset. In this session it is given an overview on how a VAE network is structured and what are the main techniques applied to make it useful to each of the issues just mentioned.
Before starting to talk about the usage VAEs, it is mandatory to go through the structure of the auto-encoder which is essentially a neural network with a bottleneck in the middle Fig 1.1 designed to reconstruct the original input in an unsupervised way, in other words, it learns an identity function by first reducing the dimension of the data to the bottleneck so as to extract more efficient and compressed representation. Surprisingly The idea was originated in the 1980s, and later promoted by the seminal paper by Hinton and Salakhutdinov [2006].

The Auto-Encoder consists of tow connected networks that could be any kind of neural networks (convolutional, or multi-layer perceptron etc) depends on the data

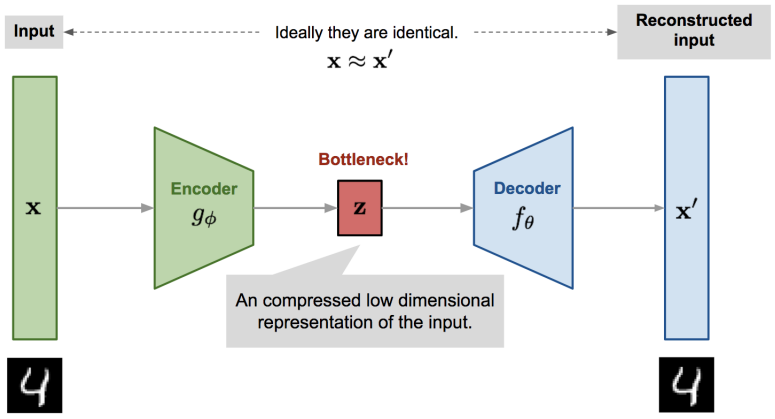

Figure 1.1. Autoencoder

it has to deal with, which are:

- Encoder network: gets the high-dimension input and transform it to into a low-dimension code in the bottleneck, or we can call it representation, latent or features as well again depends on what the usage are we making of the auto-encoder.

- Decoder network: gets the output of the encoder and does essentially the inverse process, or we can say reconstruct the data, likely with larger and larger layers to the last one that outputs the reconstructed original data.

We can see already how the auto-encoder networks can give us an efficient way to impressively represent the data and in lower dimension. So the accomplishment of solutions for the problematics we talked about at beginning of this session, is all about about how we build the bottleneck layer or what will call from now on vector z. The VAE Kingma and Welling [2013] basically is an auto-encoder but the structure of vector z is quite different. For instance what if we need to map the input into a probability distribution q_θ instead of a fixed vector z, where q_θ is parameterized by θ, from which we sample or generate z, this is what make the VAE to be recognized as a generative model. Where the training is regularized to avoid eventual overfitting that might occur with auto-encoder architecture and ensure that the distribution q_θ has good parameters to enable the generative process. The way that makes the encoder to be able to produce q_θ is by composing the bottleneck or

the output of a mean μ and a covariance matrix Σ the problem here is that nothing would prevent the this distribution to be extremely narrow, or effectively a single value. To escape the issue, the Kullback-Leibler (KL) divergence-which measures the distance between tow distributions- is introduced between the distribution produced by the encoder $q_\theta(z \mid x_i)$ and a unit Gaussian distribution $p(z)$(mean 0, covariance matrix is the identity matrix) and tell us how much information is lost when using q to represent p, this KL divergence is then introduced as a penalty to the loss function li, which consists of another term as well that is the expected negative likelihood of the i-th datapoint x_i as follow:

$$l_i(\theta, \phi) = -E_{z \sim q_\theta(z|x_i)}[\log_{p_\phi}(x_i \mid z)] + KL(q_\theta(z \mid x_i)||p(z)) \tag{1.1}$$

Where z is sampled from q_θ and ϕ the decoder parameters, the purpose of the first term in poor words mean ″how much the decoder output is similar to original datapoint x_i‶. It intuitively leads the decoder to learn to reconstruct the data. The last important part left to talk about is the training one, we can use the gradient descent to optimize the loss with respect to the parameters of the encoder and decoder θ and ϕ respectively. For stochastic gradient descent with step size ρ, the encoder parameters are updated using $\theta = \theta - \frac{\partial l}{\partial \theta}$ and the decoder is updated similarly.

Reparameterization Trick:

As we can notice at this point that there would be a problem doing the backpropagation step of the gradient descent optimizer, because it does not go through the random node z, therefore we have to implement some trick to circumvent this issue. The reparameterization trick Kingma and Welling [2013] is essentially done by introducing an auxiliary variable (noise) ε that allows us to reparameterize z in a way that allows backpropagate to flow through the deterministic nodes as shown in Fig. 1.2, we are basically expressing the random variable z as a deterministic

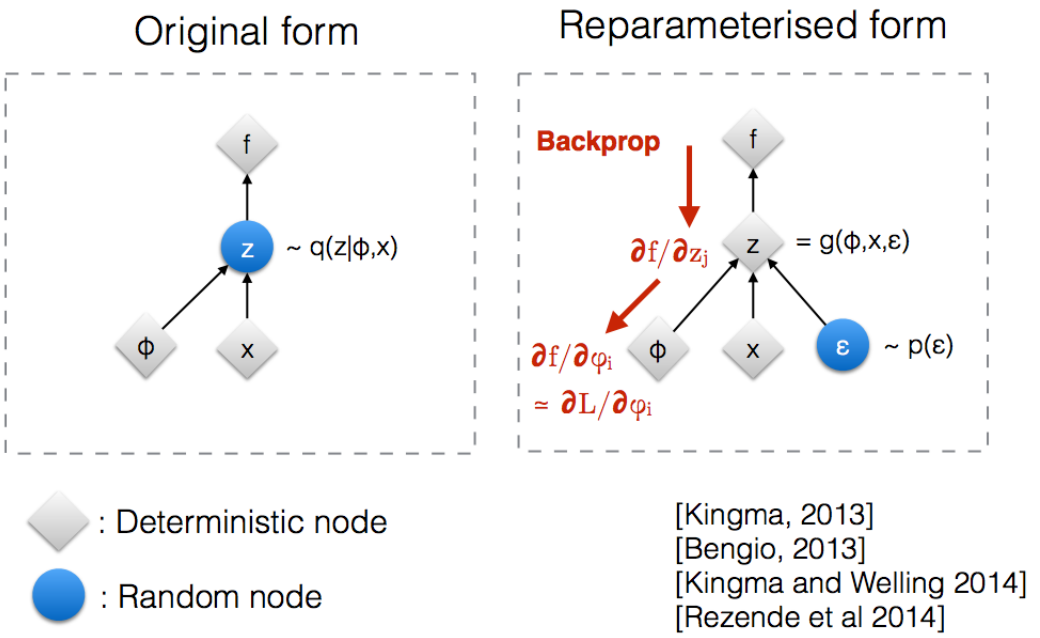

Figure 1.2. reparameterization trick

Gaussian Mixture Models (GMMs) structure and learning algorithm

GMM is a probabilistic model for representing normally distributed subpopulations within an overall population. Mixture models in general don't require knowing which subpopulation a data point belongs to, allowing the model to learn the subpopulations automatically. Since subpopulation assignment is not known, this constitutes a form of unsupervised learning. GMMs have been used for feature extraction from speech data, and have also been used extensively in object tracking of multiple objects, where the number of mixture components and their means predict object locations at each frame in a video sequence. The model is parameterized by two types of values, the mixture component weights are defined as θ_k and the component means μ_k and variances σ_k or covariances (for the multivariant case) Σ, the mixture component weights has a constraint that is: $\sum_{i=1}^{K} \phi_i = 1$ so that the total probability distribution normalizes to 1. The numerical technique used to maximize the likelihood estimation is the Estimation maximization (EM) which consists of tow steps:

- E-step: consist of calculating the the expectation of the component assignments $P(C_k|x_i)$ for each data point $x_i \in X$ given the model parameters ϕ_k, μ_k, and σ_k.

- M-step: which consists of maximizing the expectations calculated in the E step with respect to the model parameters. This step consists of updating the values ϕ_k, μ_k, and σ_k.

The entire process iteratively repeats until the algorithm converges, before it starts some initializations are made as follows: Randomly assign samples without replacement from the dataset $X = x_1, ..., x_N$, to the component mean estimates $\mu_1, ..., \mu_k$. E.g. for K=3 and N=100, set $\mu_1 = x_{45}$, $\mu_2 = x_{32}$, $\mu_3 = x_{10}$. Set all component variance estimates to the sample variance $\sigma_1^2, ..., \sigma_k^2 = \frac{1}{N}\sum_{i=1}^{K}(x_i - \hat{x})^2 = 1$, where $\hat{x} = \frac{1}{N}\sum_{i=1}^{N}(x_i)$ is the sample mean. Set all component distribution prior estimates to the uniform distribution $P(C_k) = \phi_1, ..., \phi_k = \frac{1}{K}$ while the E-step computes the probability that x_i is generated by component C_k:

$$p(C_j \mid x_i) = \frac{p(x_i \mid C_j)p(C_j)}{p(x_i)} = \frac{p(x_i \mid C_j)p(C_j)}{\sum_i p(x_i \mid C_j)p(C_j)} \tag{1.2}$$

which will be used in the M-step where the parameters are updated as follow:

$$\mu_j = \frac{\sum_i p(C_j \mid x_i)x_i}{\sum_i p(C_j \mid x_i)} \tag{1.3}$$

$$\sigma_j^2 = \frac{\sum_i p(C_j \mid x_i)(x_i - \mu_j)(x_i - \mu_j)^T}{\sum_i p(C_j \mid x_i)} \tag{1.4}$$

$$p(C_j) = \frac{\sum_i p(C_j \mid x_i)}{N} \tag{1.5}$$

Originally GMM is employed for classification and clustering tasks, but as we can deduce that it is also a suitable model when recovering the distribution of the data is needed, since it can produce more complexed distribution composed of jointed k gaussians as in Fig. 1.3, for example if we have different sources from which the data is provided. Back to our main argument, GMM has been used in several robotics applications, like in Gaussian Mixture Model for Robotic Policy Imitation Pignat and Calinon [2019] where different robots had to learn from few

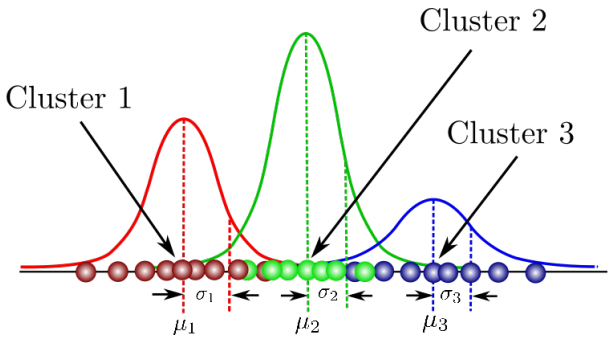

Figure 1.3

amount of demonstrations to complete various tasks such as avoid obstacles, or insert a peg in a moving hole. This approach (GMM) illustrates the advantages of learning a distribution of policies instead of trajectories and can be used in a variety of tasks. On the other hand in some work as in Zhang et al. [2016] the GMM was benefited in robot obstacle avoidance learning as a base for a generative model, to generate trajectories, by Gaussian Mixture Regression (GMR), The trajectory obtained not only can avoid obstacles but also can be executed by robots due to its good smooth property. The same idea of Zhang et al. [2016] was implemented in Reiley et al. [2010] in which GMM encodes the expert's underlying motion structure. GMR is then used to extract a smooth reference trajectory to reproduce a trajectory of the task. This GMM/GMR generative model was trained on expert data, then tested by classifying the generated trajectories to be either coming from expert, intermediate, or novice surgeons. The classification algorithm Hidden Markov Models (HMMs) trains three (expert, intermediate and novice) from five new unseen trials for each skill level. The results of the classifier show that each trajectories generated by GMM/GMR are closest to the expert model. To conclude this session it is right and proper to say that the use of GMM has remarkable impact to improve the model performance in presence of lack of data issue.

Reinforcement Learning RL

This field of machine learning deals with how an agent ought to behave in an environment in order to maximize the reward. It differs from supervised learning in

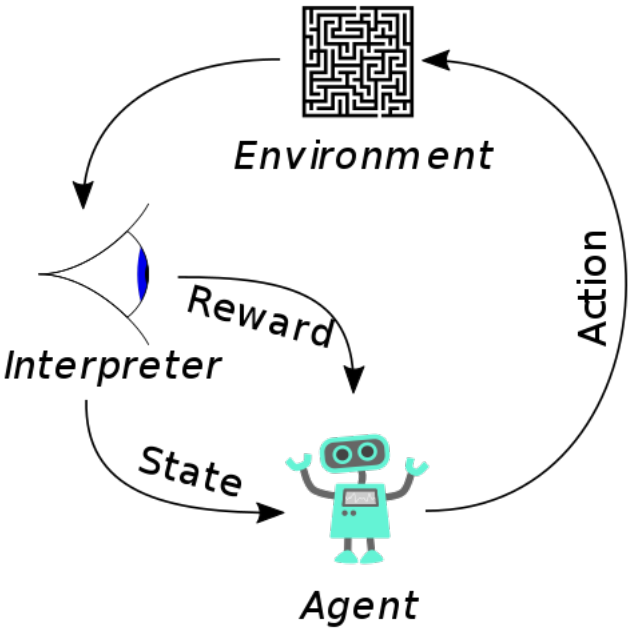

Figure 1.4. Reinforcement learning diagram

not needing of labeled input/output pairs and from unsupervised learning in getting guidance from the environment by performing actions and learning from the errors or rewards. Typically the environment takes the form of a Markov Decision Process (MDP) is a mathematical system used for modeling decision making. We use a tuple (S, A, P, R, γ) to define a MDP. Where S denotes the state space, a finite set of states. A denotes a set of actions the actor can take at each time step t. P denotes the probability that taking action a at time step t in state s_t will result in state s_{t+1}. $R_a(s, \acute{s})$ is the expected reward from taking action a and transitioning to $\acute{s}$. $\gamma \in [0, 1]$ is a discount factor, to discount the future reward. The general diagram of Reinforcement learning is shown in Fig. 1.4 There are tow notions about the environment where the algorithm that implement RL that should be mentioned which are:

- ″model-based″algorithms: who are employed when the environment is a priori known, in other words, when we know the transition probability matrix P between states, so the agent can make predictions about the next state and reward before it takes each action.

- ″model-free″algorithms: for which there is no assumption about the world.

While about the techniques the algorithm uses to lean the policy are divided as follow:

- "Off-policy": is that it updates its Q-values using the Q-value of the next state s`and the greedy action a`. In other words, it estimates the return (total discounted future reward) for state-action pairs assuming a greedy policy were followed despite the fact that it is not following a greedy policy.
- "On-policy": is that it updates its Q-values using the Q-value of the next state s`and the current policy`s action a. It estimates the return for state-action pairs assuming the current policy continues to be followed.

In this work, all the algorithms referred to are model-free since in robotics applications usually the software agent can not make any prediction about the environment, and no assumption is made whether it is on-policy or off-policy.

Going through the various algorithms of RL you can realize that in most cases there is not best algorithm, it all depends on task, environment, discrete or continuous spaces, and the data itself and its size. During my studies I have implemented different algorithms in RL which are Deep Q Learning (DQN), Deep Deterministic Policy Gradient (DDPG) and Trust Region Policy Optimization (TRPO). Basing on my modest experience I realized is that as long as we have simple and well-defined environment, picking the algorithm who is more fit to the task taking into account the domain spaces of actions and states, eventually will get good results, the agent will learn a close-to-optimal policy to behave in the environment. But when the task (policy) to be learned is more complicated in respect of the lack of resources and data and its quality, then it is more than convenient making some process on the input data to make the learning policy process more efficient computationally and of course in terms of results which are our aim first of all, for example find the latent space of the data, or define a probability distribution that describe the dataset. That what I found out while doing my survey about generative models in robotics, where RL is strongly present regardless on which algorithm has been employed, actually most of times the algorithm used was not mentioned.

Generative Adversarial Networks (GANs)

Generative adversarial networks (GAN) is algorithmic architecture that uses two neural networks, pitting one against the other (thus the ″adversarial‟) in order to generate new, synthetic instances of data that can pass for real data. They are used widely in image generation, video generation and voice generation. it was introduced firstly by Goodfellow et al. [2014] to create a new framework for estimating generative models via an adversarial process that corresponds to tow-player game, the tow networks could have arbitrary architecture and they are trained simultaneously, one neural network, called the discriminator, is designed as classifier network to evaluate the authenticity distinguishing between fake and real data instances, while the other one, called generator, is trained to generate data as close to the authentic ones. Meanwhile, the generator is creating new, synthetic instances that it passes to the discriminator. It does so in the hopes that they, too, will be deemed authentic, even though they are fake. The goal of the generator is to generate passable instances to lie without being caught. The goal of the discriminator is to identify those coming from the generator as fake. GANs are a clever way to train a generative model in the same manner of a supervised learning problem.

To describe the GAN algorithm Fig. 1.5 it is possible to start from either the generator, or the discriminator, because as mentioned in the previous section it corresponds to a tow-player game, like in chess, conventionally the while starts, but even if black starts that does not change the essence of the game. However, let's start with the more interesting one with is the generative model.

The generator model takes a fixed-length random noise vector as input and generates a sample in the domain. The vector is drawn randomly from a Gaussian distribution, and the vector is used to seed the generative process. This vector space is referred to as a latent space, or a vector space comprised of latent variables. Latent variables, or hidden variables, are those variables that are important for a domain but are not directly observable. It is often referred to latent variables, or a latent space, as a projection or compression of a data distribution. That is, a latent space provides a compression or high-level concepts of the observed raw data such

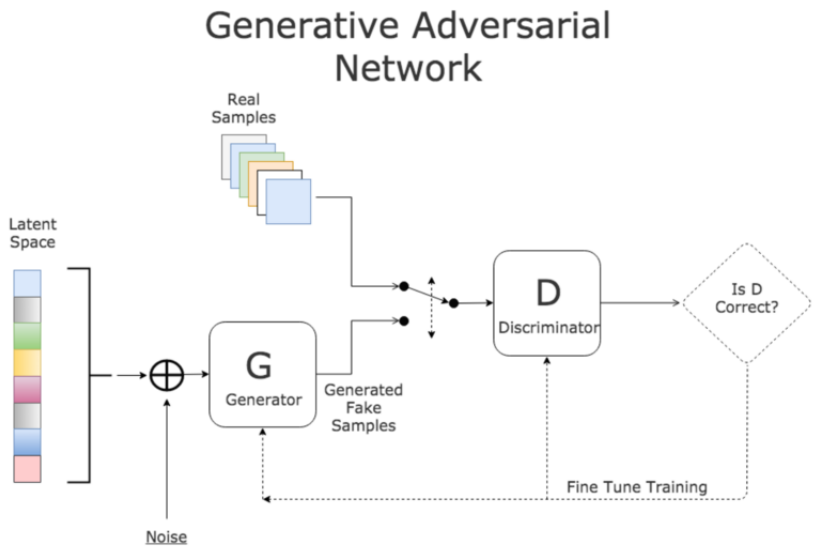

Figure 1.5. GAN Architecture

as the input data distribution. In the case of GANs, the generator model applies meaning to points in a chosen latent space, such that new points drawn from the latent space can be provided to the generator model as input and used to generate new and different output examples, after training, the generator model is kept and used to generate new samples. Sometimes, the generator can be repurposed as it has learned to effectively extract features from examples in the problem domain. Some or all of the feature extraction layers can be used in transfer learning applications using the same or similar input data.

The Discriminator Model takes an example from the domain as input (real or generated) and predicts a binary class label of real or fake (generated).The real example comes from the training dataset. The generated examples are output by the generator model. The discriminator is a normal (and well understood) classification model. After the training process, the discriminator model is discarded as we are interested in the generator.
The generator and the discriminator have different training processes, and it proceeds in alternating periods:

1. The discriminator trains for one or more epochs.

2. The generator trains for one or more epochs.

3. Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

Indeed, as the generator improves with training, the discriminator get worse, because it becomes more difficult to recognize the authentic instances rather than the generated one, which means in accuracy terms that if the generator succeeds perfectly then the discriminator has a 50% accuracy, same as flipping a coin to predict the label of the current instance. This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse, which produces limited varieties of samples. Contrarily. If the discriminator gets too successful that the generator gradient vanishes and learns nothing. Anyways, training GANs is noted as hard to obtain but still there several techniques to make the training more stable, which are out of the boundaries of this work. GANs try to replicate a probability distribution. They should therefore use loss functions that reflect the distance between the distribution of the data generated by the GAN and the distribution of the real data. Of course, there are tow loss functions for each of the tow networks as introduced in Goodfellow et al. [2014], the generator tries to minimize the following function while the discriminator tries to maximize it:

$$E_x[\log D(x)] + E_z[\log (1 - D(G(z)))] \tag{1.6}$$

where $D(x)$ is the discriminator's estimate of the probability that real data instance x is real. E_x is the expected value over all real data instances. $G(z)$ is the generator's output when given noise z. $D(G(z))$ is the discriminator's estimate of the probability that a fake instance is real. E_z is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances $G(z)$). The formula derives from the cross-entropy between the real and generated distributions. Since the generator can not directly affect $\log D(x)$ so, for the generator, minimizing the loss is equivalent to minimizing $log(1 - D(G(z)))$.

2 Generative Modeling in Robotics

Employment of VAEs and DNNs in generative models for robotics

Lets go now through some papers to see where and how the VAEs have been employed and show their effectiveness in various applications of generative models in robotics. The first one Eslami et al. [2016] where a framework called by the authors *Attend, infer, repeat (AIR)* the VAE structure here is quite different that the encoder was implemented as a Recurrent Neural Network (RNN), since its purpose is to learn to detect and generate objects, specifically ″where are the objects, what are they and how many are they‟. The additional recurrence to the structure is basically to detect how many objects are present in the input data. Experiments were designed initially on 2D data particularly on multiple MNIST digits, and reliably the model were able to detect and generate the constituent digits from scratch, it shows advantages over state-of-art generative models computationally and also in terms of generalization to unseen datasets. Other Experiments on 3D datasets, considering scenes consisting of only one of three objects: a red cube, a blue sphere, and a textured cylinder. The network accurately and reliably infers the identity and pose of the object, on the other hand, an identical network trained to predict the ground-truth identity and pose values of the training data has much more difficulty in accurately determining the cube's orientation.

Moreover, in robotics, grasping and manipulation of various objects is a critical and challenging problem, Veres et al. [2017] has proposed another concept referred to as the grasp motor imagery (GMI) that combines object perception and a learned prior over grasp configurations, to synthesize new grasps to apply to a different object. At the core of GMI is the autoencoder structure, taking advantage of Deep Learning (DL), particularly building both encoder and decoder as Convolutional Neural Networks (CNN). The approach followed is intuitive: based on perceptual information about an object, and an idea of how an object was previously grasped, collecting a object/grasp pair dataset of successful, cylindrical precision robotic grasps using the V-REP simulator Rohmer et al. [2013], and object files provided by Kleinhans et al. [2015] on a simulated picking task. The VAE was trained on this dataset to generate grasps for novel objects. GMI integrates perceptual information and grasp configurations using deep generative models. Applying it to a simulated grasping task, has demonstrated the capacity of these models to transfer learned knowledge to novel objects under varying amounts of available training data.

Furthermore, a novel data generation pipeline for training a deep neural network (DNN) approach was introduced in Domain randomization and generative models for robotic grasping Tobin et al. [2018], that perform the grasp planning for randomized objects domain. The model architecture of this paper shown in Fig. 2.1, is composed of tow parts that are trained separately, the second part is grasp evaluation model f that acts as a classifier taking as input a single observation image from the robot and outputs a single scalar value corresponding to the likelihood of success of that grasp returned from the planner. While the more interesting part is first one which is the grasp planning module $\gamma(I) = \beta \circ \alpha(I)$ Where β consists of n (the dimensionality of the grasp) small NNs, each of them takes as input a representation of the image s and outputs a softmax over possible values of g as follows:

$$\beta(s)(g) = \prod_{i=1}^{n} \beta_i(g_1, ..., g_n, s) \tag{2.1}$$

In other words, what actually the planner outputs is a probability distribution over possible grasps.

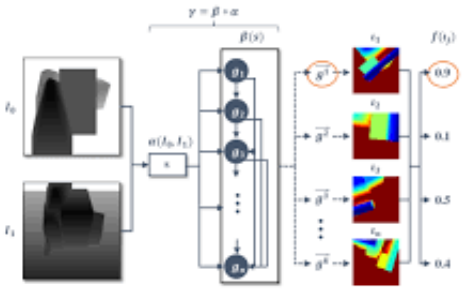

Figure 2.1. An overview of sampling from our model architecture

This architecture was trained on randomly unrealistic generated objected based on than 40,000 object meshes found in the ShapeNet object dataset Chang et al. [2015], the experiments of this framework showed its ability to scale to real world objects. Where this learned model has been tested on 30 previously unseen objects chosen to capture the diversity of object dataset obtaining a overall success rate of 80%.

Exploitation of VAE and GMM in RL

As mentioned in the previous section, most of the time applying RL algorithm directly on high-dimensional data does not lead to a good performance, so in this kind of situation it is advantageous to make use of the techniques that permit to reduce the dimensionality by representing the data in more suitable way.
One of the frameworks I went through has exploit both VAE and GMM to neatly make it feasible to fit the dynamics even when the number of samples is much lower than the dimensionality of the system. This what Finn et al. [2016] does, where initially RL algorithm run on robot with initial random policy to collect N (5 for that experiment) samples, then use them to fit GMM to learn the environment dynamics or the policy controller without vision but using only the robot's configuration as the state. In a second phase a VAE is trained to encode image dataset with unsupervised learning to produce a low-dimensional bottleneck vector that is a natural choice for learned feature representation or feature points for each image that concisely describes the configuration of objects in the scene, the interesting part

of this VAE is that it is forced to encode spatial features rather than values. This is obtained basically by applying spatial soft-max activation function that consists of tow operations on the last convolutional layer of the encoder as follow:

$$s_{cij} = \frac{e^{\frac{a_{cij}}{\alpha}}}{\sum_{i'j'} e^{\frac{a_{ci'j'}}{\alpha}}} \tag{2.2}$$

where the temperature α is a learned parameter. Then, the expected 2D position of each softmax probability distribution s_c is computed according to:

$$f_c = (\sum_i i s_{cij}, \sum_j j s_{cij}) \tag{2.3}$$

which forms the autoencoder's bottleneck and essentially it is the learned spatial feature point representation, that will therefore be capable of directly localizing objects in the image. The third and final phase of this framework is same as the first one, but the difference here is that the controller is trained on the feature points of the encoder using same trajectory-centric reinforcement learning algorithm.
The experiments of this method showed that it could be used to learn a wild range of manipulation skills that require close coordination between perception and action, and uses a spatial feature representation of the environment, which is learned as a bottleneck layer in an autoencoder. This allows us to learn a compact state from high-dimensional real-world images. Furthermore, since this representation corresponds to image-space coordinates of objects in the scene, it is particularly well suited for continuous control. The trajectory-centric RL algorithm we employ can learn a variety of manipulation skills with these spatial representations using only tens of trials on the real robot.

In another work Nair et al. [2018] a RL framework was designed to jointly learns representations and policies from raw sensor inputs that achieve arbitrary goals under this representation by practicing to reach self-specified random goals during training. Here shows up the problem of choosing a suitable goal representations, to deal with this, a goal space Ģ as to be same as the state space Ş. As well the problem of high-dimensional observations such as images arises, to handle it the authors once again the rely on VAE to learn a latent embedding for both Ģ and Ş, by executing a

random policy to collect state observation and optimize Eq. 1.1, an additional online training has been introduced where the VAE is fine-tuned during the policy training each 3000 environment steps on all of the images observed by the policy, because as the policy improve it might visit new state observations where the VAE is not trained on, this additional training helped the performance of the overall algorithm. The final step is to run the RL algorithm which is a value-based one in this work twin delayed deep deterministic policy gradients (TD3) Fujimoto et al. [2018] is used, the thing that should be pointed out here that the negative Mahalanobis distance in the latent space were used as a reward function, but it turned out that minimizing this squared distance was equivalent to maximize the probability of the latent goal. This framework for learning general-purpose goal-conditioned policies that can achieve goals specified with target observations.

GANs vs VAEs

VAEs are a probabilistic graphical model whose explicit goal is latent modeling, and accounting for or marginalizing out certain variables as part of the modeling process. They can make good generative models, though they are ideal in settings where the latent is important. The VAE naturally collapses most dimensions in the latent representations, and generally gets very interpretable dimensions out, its ability to set complex priors in the latent is very nice especially in cases where you know something should make sense or you have a desired latent distribution. As well as we have seen in section 2.1, GAN is explicitly set up to optimize generative tasks, though recently it also gained a set of models with a true latent space. The worry of VAE is that it extends the probability distribution over datapoints that does not make sense, whereas GAN may miss them but as a result for a generative model it looks more reasonable. However it is hard to tell which one is better it all depends on the task, because it is hard measure and test. Speaking of VAEs and GANs, it is therefore appropriate to mention a paper titled Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-To-End Learning from Demonstration Rahmatizadeh et al. [2018], where they developed an approach that

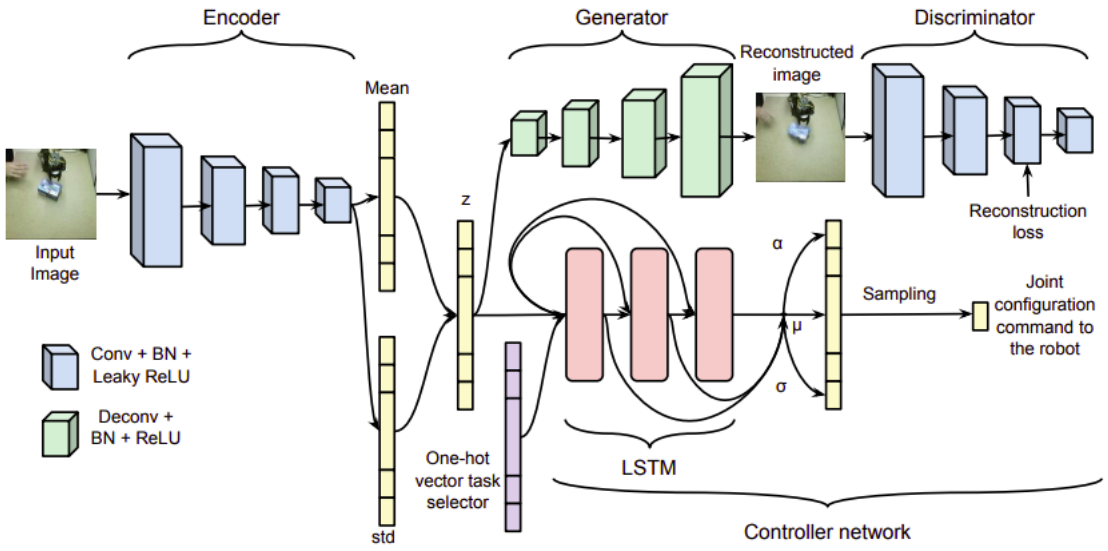

Figure 2.2. multi-task robot manipulation learning

takes as input images of the environment and outputs the next joint configuration of the robot to execute. A combination of VAE-GAN structure and LSTM was composed to build a Robot controller using end-to-end learning from demonstration, where the controller is a LSTM used to generate joint commands to control the robot, and it is based on a GMM, instead of predicting all the outputs (joint configurations), they are factorized into one-dimensional distributions, one for each joint configuration. while the VAE-GAN structure consists of three neural networks. The first network is an encoder that finds the distribution of the data and then sample a latent representation $z = q(z \mid x)$. The second part of the autoencoder is a GAN, the generator tries to reconstruct the original image, while the discriminator that takes a real image or a reconstructed image and tries to predict whether it is real or reconstructed. The objective E_{GAN} is then computed as in eq. 1.6. regarding VAE, the objective in eq. 1.1is changed in this paper where the first term $[\log_{p_\phi}(x_i \mid z)]$ is replaced by E_{GAN}, plus the reconstruction error where they used mean squared error (MSE) between the extracted features from those images in the third convolutional layer of the discriminator as follows: $E_{rec} = MSE(D_3(x), D_3(\tilde{x}))$ whereas the normal prior imposed to the latent distribution p(z) to regularize the encoder $E_{prior} = KL(q_\theta(z \mid x_i)||p(z))$ is kept. Finally, the error of the autoencoder network

can be described as the sum of errors formulated before: $E_{AE} = E_{GAN} + E_{rec} + E_{prior}$. The proposed model with architecture in Fig. 2.2, as the experiments showed, is very powerful and it does not have any assumption about the task or the shape of objects that are involved in each task. It generated smooth trajectories that follow reasonable path in different situations. This is good since we can train the model on a wide variety of tasks. However, we need large number of demonstrations to successfully learn a single task. At the same time it could be used for a wild variety of tasks given its generalization property acquired from the VAE-GAN structure.

Combination between GANs and RL

RL is a powerful technique to train an agent to perform a task, so it is capable of achieving that single task specified via its reward function. It an approach that is hard to be scaled to a situation where the agent needs to perform different set of tasks, such as navigating to varying positions in a room or moving objects to varying locations. Held et al. [2018] has offered an idea that combine RL and GANs that is able to solve this kind of situations, in this framework, instead of learning to optimize a single reward function, an indexed range of reward functions r^g is defined, each goal $g \in G$ corresponds to a set of states $S^g \subset S$, in such way that the goal g is considered to be achieved from any state $s_t \in S$. The policy that should be learned ,given a goal g, must perform optimally with respect to r^g. The framework uses a simple indicator function to define the reward that gives a measure whether the agent has reached the goal $r^g(s_t, a_t, s_{t+1}) = 1\{s_{t+1} \in S^g\}$ where $S^g = \{s_t : d(f(s_t), g) < \epsilon\}$ e $f(.)$ is a function that projects a state into goal space G, $d(.,.)$ is a distance metric, ϵ acceptable tolerance that determines when the goal is reached. it is also defined a uniform distribution $P_g(g)$,although in practice any distribution can be used, over the set of goals G to sample from, so the overall objective function that the authors call it *coverage* will be:

$$\pi^*(a_t \mid s_t, g) = \arg\max_{\pi} E_{g \sim P_g(.)} R^g(\pi) \tag{2.4}$$

Where $R^g(\pi)$ is the success probability of each goal. Sampling goals from $P_g(g)$ is modified to be uniform only over a set of goals grounding on the level of difficulty,

or in better words, Goals of Intermediate Difficulty (GOID):

$$GOID_i := \{g : R_{min} \leqslant R^g(\pi_i) \leqslant R_{max}\} \tag{2.5}$$

Due to sparsity for the reward function the current policy π_i for most goals would not get reward, in the way it will be hard to train the policy, to circumvent this, the sampled goal g should grantee that π_i is able to receive some minimum expected return R_{min}. On the other hand to prevent the policy keeps training on only some of those goals that get very high reward, the R_{max} restriction is added to make the policy train on the goals who are not mastered yet. $GOID_i$ allows the policy to train on wild coverage objective. that was the first part of the algorithm designed in this framework which is partitioned into three stages. The second stage is the Goal-GAN employment, its generator network is used to generate goals that fall in $GOID_i$, from noise z, while the discriminator network distinguish the goals that are in $GOID_i$ from those that are not. a modification has been added to the LSGAN introduced implementation in Mao et al., 2017 this modification allows to train the LSGAN both with positive examples from the distribution we want to approximate and negative examples sampled from a distribution that does not share support with the desired one this gave the chance to improve the accuracy of the generative model even though it was trained on few positive samples. This adoption was made by introducing a binary label y_g that permit to train on "negative samples" when $y_g = 0$ then optimize the LSGAN objectives:

$$min_D V(D) = E_{g \sim pdata(g)}[y_g(D(g) - b)^2 + (1 - y_g)(D(g) - a)^2] + E_{z \hat{a} \acute{L} ij p_z(z)}[(D(G(z)) - a)^2]$$
$$max_G V(G) = E_{z \hat{a} \acute{L} ij p_z(z)}[(D(G(z)) - c)^2] \tag{2.6}$$

using ($a = -1$, $b = 1$, and $c = 0$).

At each iteration i the algorithm initially sample the noise vector z to generate a goals $G(z)$ that are used train the RL algorithm this time TRPO with AGE is used as in Schulman et al. [2015] afterwards the is evaluated to assign a label y_g to each goal, and use these labels to train the goal generator and goal discriminator of the GAN following eq. 2.6. Few experiments have been done to test this method, one of them was a Ant navigation in a maze, in this case the goals generated have tow dimensions (x, y) and it is been discovered that the generated goals were concentrated in some

area were the policy is receiving some expected return signals which obviously need improvements, however still the Goal-GAN can shift the distribution of sample goals of the appropriate difficulty dynamically around the origin in a ring growing manner. Similarly another experiment was made on a multi-path point-mass maze here again the generator was able to produce a multi-modal distribution over the appropriate goal of intermediate level of difficulty. In this last experiments, it is been used N-dimensional Point Mass the full state-space of the N-dimensional Point Mass hypercube. However, the Point Mass can only move within a small subset of this state space, also here the performance was good, using the goal-GAN as a generative model even without this prior knowledge, the Goal GAN discovers the feasible portion of the goal space and generate automatically for the policy that are at the appropriate level of difficulty.

One more paper that made use of GAN is Generative Adversarial Imitation Learning (GAIL) Ho and Ermon [2016] where the agent has to imitate an expert behavior, the algorithm make use of the RL algorithm TRPO which at each iteration construct a careful step scheme to update the policy to ensure that the divergence does not occur, simultaneously exploiting the GAN architecture, where the discriminator distinguishes between the expert trajectories and those generated by the generator, in turn, it generates trajectories following TRPO, in other words, the generator acts as policy controller that should be learned. Specifically a trajectory τ_i is generated by executing TRPO, then the discriminator updates its parameters w_i to w_{i+1} with the gradient:

$$E_{\tau_i}[\nabla_w \log D_w(x)] + E_{\tau_E}[\nabla_w \log (1 - D_w(G(z)))]$$

afterwards the generator's update function in this paper is composed of tow terms the first is like TRPO algorithm, by finding a trust region in which la policy can move, preventing it of changing too much due to noise in the policy gradient and move it toward expert-like regions of state-action space, and the second term is the casual cross entropy that is used as a cost function or policy regularizer as the following:

$$E_{\tau_i}[\nabla_\theta \log \pi_\theta(a \mid s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta) \tag{2.7}$$

where $\lambda \geqslant 0$ is a hyperparameter and $Q(s,a) = E_{\tau_i}[log(D_{w+1}(s,a)) \mid s_0 = s, a_0 = a]$

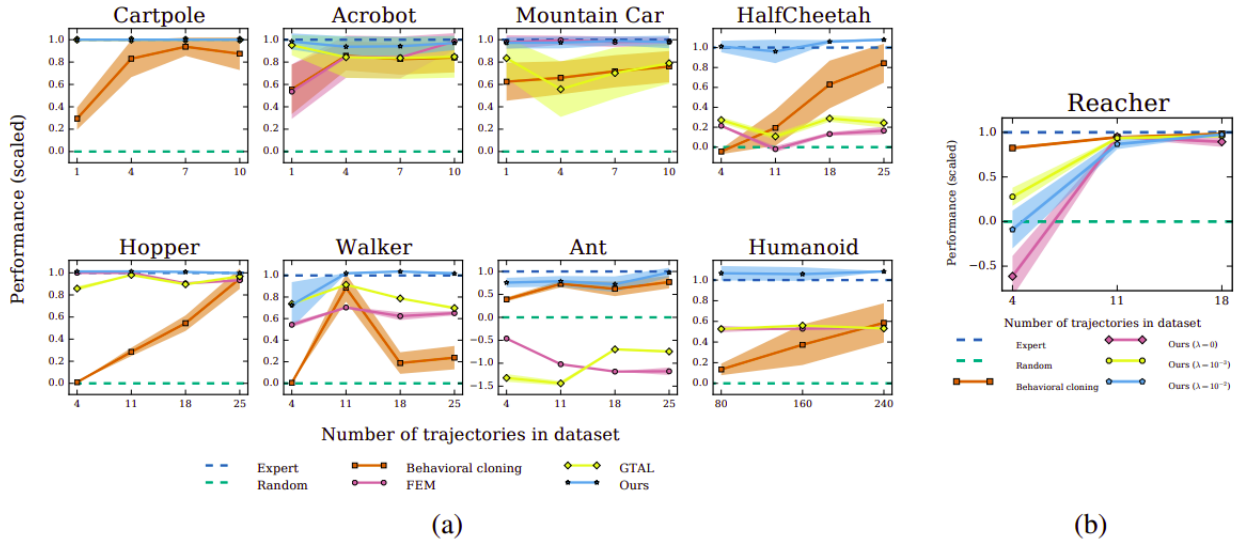

Figure 2.3. (a) Performance of learned policies. The y-axis is negative cost, scaled so that the expert achieves 1 and a random policy achieves θ. (b) Causal entropy regularization λ on Reacher.

The previous iteration repeat until convergence, in other words like in GAN algorithm, it repeats till finding a saddle point (π, D) of the expression 2.7 between the tow approximation functions of the policy and the discriminator.
The algorithm has been tested on 9 physics-based control tasks, varying from low-dimensional control tasks from the classic RL literature for example cartpole, acrobot, and mountain car, to difficult high-dimensional tasks such as a 3D humanoid locomotion, solved only recently by model-free reinforcement learning. the procedure followed for each of these experiments was firstly to generate an expert behavior be running TRPO (not the generator of the GAN of this algorithm) on the true cost function, the algorithm was compared with three baselines such are Behavioral cloning, Feature expectation matching (FEM), Game-theoretic apprenticeship learning (GTAL). All algorithm had the same neural network architecture to train the policy and have been given the same amount of interactions with the environments.

As we can see in Fig. 2.3 this algorithm produced policies performing better than the other baselines except on the Reacher environment, where behavioral cloning performed excellently even though it suffered on the classic control tasks, while we could see in Fig. 2.3b the effect of causal entropy regularization λ we have met in

eq. 2.7, tuning it has changed the performance of the overall algorithm GAIL on the Reacher task. Instead on the other MuJoCo environments, GAIL has achieved at least 70% of the expert behavior for all datasets, as well as performing much better then the other baselines for on all the tasks.

GAIL is not the unique algorithm to take advantage and use of the GAN architecture, Self-Improving Generative Adversarial Reinforcement Learning (SL-GARL) Liu et al. [2019] as well has developed another framework that once again exploits it to imitate the expert. The general concept repeat itself where the discriminator acts as a classifier and gives a high error values to the initial policies and low error values to the improved ones, while the goal of the generator is to generate policies to minimize the error values. In addition SL-GARL employs a black box improvement policy operator, generally speaking this kind of function may get the computational complexity worse, but in this case it helps to shrink the exploration space and a consumption on computational resources. This operator maps a policy and a state value to an improved policy $\hat{\pi} = I(\pi, V_\pi)$ that is supposed to be better then the direct output from the policy neural network, because it produces the expert demonstration of the improved policy. The Self-Improving RL algorithm employs tow neural networks one is for evaluating the state value function $\hat{V}(s, a) = DNN_\theta(s)$ where θ is the neural network parameters. The second neural network is for the policy estimator defined as $\hat{\pi}(a \mid s, w)$ where w is the neural network parameters, this network will be considered as the generator of the GAN. The loss function at this point is going to be the KL-divergence between the improved policy and the agent one as follows:

$$L_I(w, \theta, s) = D_{KL}[I(\hat{\pi}(\cdot \mid s, w), \hat{V}(\theta, s)) \parallel \hat{\pi}(\cdot \mid s, w)] \tag{2.8}$$

Hence the update formulas for SI-RL will be:

$$w_{k+1} = w_k + \nabla_w L_I(w_k, \theta_k, s) = w_k - I(\hat{\pi}(\cdot \mid s, w), \hat{V}(\theta, s)) \nabla_w \log \hat{\pi}(\cdot \mid s, w)$$

$$\theta_{k+1} = \theta_k + \nabla_\theta L_I(w_k, \theta_k, s) = \theta_k - I(\hat{\pi}(\cdot \mid s, w), \hat{V}(\theta, s)) \nabla_\theta \log \hat{\pi}(\cdot \mid s, w) + \nabla_\theta \parallel \hat{V}(\theta, s) - R \parallel^2$$

The outputs of these tow networks, The state value $\hat{V}(\theta, s)$ and the initial policy $\hat{\pi}(\cdot \mid s, w)$ are then passed to the operator I. The inclusion of the GAN was by adding a discriminator which distinguish between improved policy which will interact

with the environment and the initial one generated by the policy network, the figure Fig. 2.4 describe the architecture and data flow of the framework. The authors of this paper have evaluated three different algorithms to implement the policy improvement operator:

- **trust region policy optimization (TRPO):** which can be used as policy improvement operator directly without any modification, because of the monotonic improvement guaranteed in TRPO
- **Monte Carlo tree search (MCTS):** Browne et al. [2012] is a search algorithm that uses randomness for deterministic problems difficult or impossible to solve using other approaches, it is based on tree data structure, it can be used as a planning-based police improvement operator. It provides a long term panning ability compared to the policy gradient methods
- **cross entropy method (CEM):** Since all policies are parametrized by some network parameters, and the aim is to modify these parameters in such way that the total discount reward is maximized, this method take this problem as a black box with respect to the parameter θ, and it maintains a distribution over parameter vectors and moves the distribution towards parameters with a higher reward.

Both GAN and self-improving procedures show their potential in multiple test domains either against typical deep reinforcement learning (DRL) algorithms like DQN or the actor-critic algorithms like Asynchronous Actor-Critic Agents (A3C). Since the lack of data is one of the main challenges that RL faces. Recent researches solve the problem resort to supervised learning methods by utilizing human expert demonstrations, or by imitation learning like the previous paper we talked about. Anyway, this paper proposes and implements a novel SI-GARL framework that avoids to face directly to the reward engineering problem in RL. A policy improvement operator is defined to provide flexibility to the framework, and employ the operator as a black box to implement the self-improving procedure, a future work for this might be to select self-improving automatically basing on the task. The of integrating GAN into the SI-GARL framework was in order to further improve the exploration

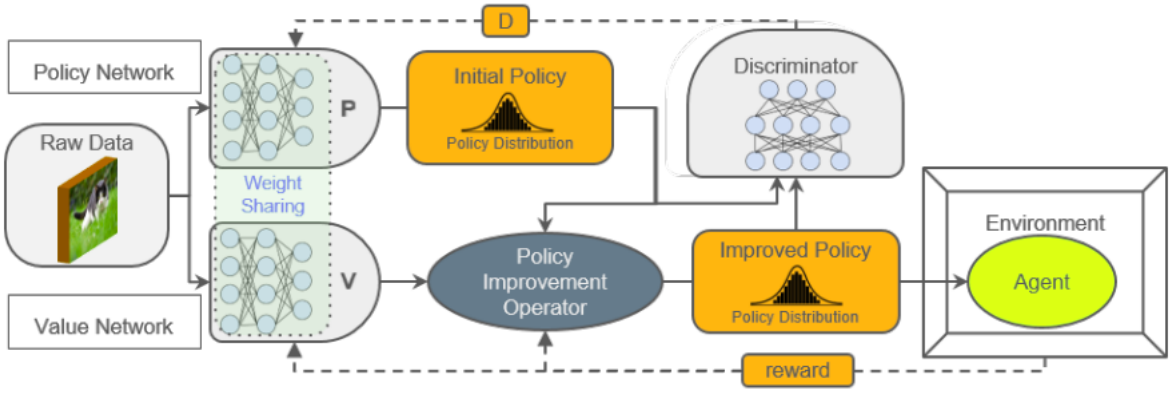

Figure 2.4. The architecture of SI-GARL framework. Arrows with solid lines show how the data transfer between modules, and those with dotted lines show the feedback signal used to update the networks. The orange blocks represent the internal data. The grey circle represents the policy improvement operator seen as a black box

quality and the data efficiency. The imitation step is implemented by adversarial training, which is not a standard GAN implementation, but similar to GAIL Ho and Ermon [2016]. The generator-and-discriminator gaming principle still exists with the same purpose in this framework, even though does not seem to be similar to a classical GAN, but inherits its essential idea of the tow networks who are trying to fool each other. The experiments showed that it is worth to spend more computational cost on this framework due to its outstanding performance. Both GAN and self-improving procedures show their potential in multiple test domains.

3 Discussion

Going through the different frameworks we have seen in the previous section, we could notice that sometimes the goal of some researches are the same, like to find the probability distribution of data, or to maximize the total returned reward, generate new data instances, but the various authors were making different choices of the available techniques to achieve there goal in solving the task. We could explain these choices by making one step back and have a look on the data. Firstly the question that should be made is ″what is the raw data the framework is going to cope with?‶. The data we have seen are of different natures like trajectories, images, sensory data, grasp configuration and so on. Posting this question initially leads to figure out what are the space dimensions to deal with, and gives place to additional questions like ″should we move to latent space?‶in case it is high-dimensional like images. Another main question is ″How big is the dataset?‶this one is very important impact on the choice to make on the technique to use for the generative modeling, I discovered that when the dataset to base on the not large enough, means that it is more opportune to know probability distribution to generate new data instances from, so the choice of VAEs 1.1 with reparameterizing trick 1.1.1 in this case should be a good fit. Nevertheless, VAEs struggles to find a complex probability distribution in situations where the dataset is high-dimensional and composed of variables that vary in wild ranges, to overcome this kind of issue the GMM 1.2 approach may fit better, given the property of GMM to combine K probability distributions components

to represent a final probability distribution that fits more efficiently the dataset. Moreover, in human cloning and/or learn from demonstrations tasks, where data are collect by making an expert to behave or execute actions in environment to achieve the goal of the task, this collected dataset is usually small for both learning process of the agent given some demonstrations, or cloning an expert behavior to achieve the required task. In this kind of situations where the amount of data available is not big enough, VAEs still not a good choice unless we follow the idea of moving to the latent space, then VAEs is the best option taking advantage its nice probabilistic formulation they come with as a result of maximizing a lower bound on the log-likelihood. Working in latent space is another approach that can circumvent the high-dimension real-world data. VAE naturally collapses most dimensions in the latent representations, and generally getting very interpretable dimensions out, even though the training dynamics are generally a bit weird. VAEs are used as well to learn the data representation like in Finn et al. [2016] which is a reinforcement learning application to robotic manipulation tasks. The authors were interested in knowing the spatial feature representation of the environment, these spatial features are the configurations of objects in the scene. The bottleneck of the autoencoder was modified in such way the spatial representation is learned. VAE has shown its ability as well as a generative model like in Eslami et al. [2016] formalizing a framework called *Attend, infer, repeat* for efficient variational inference in latent spaces of variable dimensionality. The key idea of this paper is to treat inference as an iterative process, implemented as a recurrent neural network that attends to one object at a time, and learns to use an appropriate number of inference steps for each image. VAE as a generative model is able to compete with GAN which is the most interesting idea in machine learning of the last 10 as Yann LeCun, Director of AI Research at Facebook AI claims. GANs is a superior model with respect to VAEs because they are better at generating visual features this superiority conduct to say that the adversarial loss is better than mean-squared loss. The difference that products this superiority is that VAE can spread the probability mass to places it might not make sense, whereas GAN models may never explore. Later, a cooperation between VAE and GAN has been done in Rahmatizadeh et al. [2018] mentioned in 2.3

which is one of the most advanced paper I read during this survey, that represent the state of art of robot manipulation tasks. it demonstrated how it is possible to learn complex manipulation tasks from user demonstrations, such as picking up a towel, wiping an object, and depositing the towel to its previous position, entirely from raw images with direct behavior cloning and outputs the joint configuration of the robot.

After the spread of GANs, their influence reached as well to reinforcement learning algorithms, specially those who follow the actor-critic approach like TRPO and DDPG. However GANS could be interpreted as an actor-critic approach, where the generator should learn the policy and the discriminator acts the critic that returns a feedback in such a way both generator/actor-discriminator/critic improve by the time goes on. Generally speaking this is what happens in all RL approaches that exploit the GANs architecture, like in GAIL Ho and Ermon [2016] which is designed to imitate an expert behavior. The generator in this case is trying to generate trajectories as much similar to the expert ones, until the discriminator could distinguish between them, in other word a saddle point is found. This idea repeats itself each time a framework would like to achieve a cooperation between RL and GAN, as we have seen as well in SI-GARL Liu et al. [2019]. GAIL and SI-GARL are the best examples to make that describes the introduction of GANs in RL, it transforms the actor-critic approach of reinforcement learning to generative adversarial network, where the generator is acting as it is an actor the discriminator criticizes be distinguishing between authentic and generated data. The great impact of this introduction was avoiding the design of a reward function which is one of the main issues RL suffer giving space to implement new multi-task frameworks for robotics that are able to guarantee the very important generalization property of machine learning like for example in Rahmatizadeh et al. [2018]. The initial goal of inventing the generative model was to generate images, achieving great results, afterwards instead of only learn the data and try to find patterns in it as the other machine learning models, it became a way to make the machines first understand the data and then make the decision regard it. The introducing generative modeling in robotics application has had different employments over time, were firstly was used

as a data augmentation instrument when generative modeling was essentially used to generate images, then to a tool that extract more fit representations of the data, latent spaces, or features like spatial features, location, pose, and orientation of the objects in the environment the robot is working in. Given the high performance of the generative models in terms of results, led to get more employments in robotics applications. In the most recent works we have seen generative modeling has almost substitute or at least has become the main framework obviously taking advantage of the existing algorithms, as we have seen in various recent works we have mentioned, where the generator is acting as a policy controller which interacts with the robot, and its parameters are updated according to both generative model loss function and the loss function of the existing algorithm (GAIL uses TRPO RL algorithm). Generative models can substitute RL, as in some papers, by instead of generating new datapoints they generate the action to perform by the robot, such as the joint configurations of the robot, or generating the next state given the current one.
In addition, the high dimensionality of the domain space is never been helpful in robotics applications, because in all the robotics tasks we are interested in some specific informations inside the input, so we can notice that working in latent space or in some low-dimensional representation of the input data is almost a must. One of the best ways to obtain these representations is VAE as we have seen in many papers mentioned in the previous section. The most powerful generative models use deep learning where essentially they are neural networks that exchange elaborated data and errors -even if the goal is to try to fool each other like GAN- to achieve the final model that is able to generate new instances similar to original dataset. This is what now days is called Deep Generative models. However, the experiments of papers we have seen unfortunately were made in simulated environments like GAIL, showing very good results even in the more complex ones like Ant and Humanoid. While Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-To-End Learning from Demonstration has experiments on real robot providing videos that show how the robot behavior in performing different tasks. Furthermore, Tobin et al. [2018] showed how the generated objects has helped the framework to scale to real-word objects to perform grasps.

Conclusion

The goal of this survey is to bring together some of the researches oriented for robots that make use of the knowledge about deep and probabilistic generative models to develop a future cognitive architecture. It also aims at examining the challenges and opportunities emerging from the interdisciplinary research field covering machine learning, and robotics. During this survey it was clear that the most commonly used and efficient approaches to achieve a generative models are Variational Autoencoders (VAE) we have seen in sec. 1.1 and Generative Adversarial Networks (GAN) in sec. 1.4. Taking advantage of the ability of VAEs in either finding the latent space when dealing with high-dimensional input data, or to benefit of the its structure. VAE aims at maximizing the lower bound of the data log-likelihood. However, the skills that VAEs enjoy does not deny the efficiency of Gaussian Mixture Models (GMM) sec. 1.2 in finding the probability distribution of the dataset from which data instances are drawn. GMMs were employed to encode and retrieval trajectories, and imitate demonstrated policies.

In a field like robotics the lack of data is one of the most common problems the researchers meet as I noticed in the different papers I have read, these researchers had to make the robot learn some task or imitate human behavior given a limited amount of demonstration, GMM has shown there ability in finding complex probability distribution. While the powerful of GANs is that can create new content based on guidance of the dataset. There are some issues usually are faced in training GANs is that they need large amount of data, the training process itself, and they struggle

to find data distribution, even though they are powerful to generate visual data instances that are conform with the original dataset. GAN approach is to achieve an equilibrium between Generator and Discriminator, which make the training process confusing, where sometimes it hard to till if the model has reached what we called earlier the saddle point between the generator and discriminator.

The generative modeling in this survey has been done in different spaces as sensory space, more efficiently in latent representations of row observations, or to encode spatial features. The overall utility we found out is that these generative models helped to achieve more efficient frameworks in robotics field, this help was essentially obtained by generating more data for training and testing, or generate behaviors that are similar to an expert demonstration, giving chance to the agent to imitate the expert. This integration of the generative models in robotics applications, has almost opened a new field in machine learning, and gave birth to additional improvements to the various existing algorithms used in robotics, fastening their training process by reducing the number of epochs to train the model, when there are some demonstrations to learn from. technically speaking this integration is done by modifying the loss function both actor and the critic by adding the adversarial components. In addition, taking advantage of GANs has help to avoid to face the problem of formulate the reward function in RL. Finally, to conclude this work, I can tell that the generative model has been employed to either assist the existing approaches applied on robotics, or to introduce new ones that by taking advantage of the structures of the different instruments of generative modeling. However, this introduction has yielded a notable improvement to the robotics applications mainly by decreasing the training process where there is no more need to carry out the whole learning on only real world data, instead it is done using generated data instances. Moreover, the generative models made it easier for robot agents to clone a human behavior, by generating policies that are similar to the demonstrations made by humans, that consequently reduces the number of epochs the learning process should effectuate. Nevertheless, the generative model has of course helped to achieved better results for robotics applications, but still it is not enough though, the improvement margins are still wild. One of the main objectives of robotics

researchers is to achieve robots able to perform wild variety of tasks, the generative modeling can help, though the path toward robotic manipulators that can execute a wide variety of tasks lies in multi-task learning as we have seen in Rahmatizadeh et al. [2018]. We can imagine that, in the future, a large number of end-users could specify their own tasks to their own robots, all what the users have to do is to show their robots some demonstrations of how to behave, with the help of the generative models, the robots could autonomously learn how to perform efficiently the tasks. Moreover, in future work, generative modeling might introduce opportunities for learning to observe not only from similar agents or humans, but also from other agents like robots with different embodiments whose actions are unknown or do not have a known correspondence, nevertheless, contribution would be to learn to transfer across environments.

Bibliography

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pages 3225–3233, 2016.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley,

Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. 2018.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. URL http://arxiv.org/abs/1606.03476.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Ashley Kleinhans, Benjamin S Rosman, Michael Michalik, B Tripp, and Renaud Detry. G3db: A database of successful and failed grasps with rgb-d images, point clouds, mesh models and gripper parameters. 2015.

Yang Liu, Yifeng Zeng, Yingke Chen, Jing Tang, and Yinghui Pan. Self-improving generative adversarial reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 52–60. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pages 9191–9200, 2018.

Emmanuel Pignat and Sylvain Calinon. Bayesian gaussian mixture model for robotic policy imitation. *IEEE Robotics and Automation Letters*, 4(4):4452–4458, 2019.

Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end

learning from demonstration. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3758–3765. IEEE, 2018.

Carol E Reiley, Erion Plaku, and Gregory D Hager. Motion generation of robotic surgical tasks: Learning from expert demonstrations. In *2010 Annual international conference of the IEEE engineering in medicine and biology*, pages 967–970. IEEE, 2010.

Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Josh Tobin, Lukas Biewald, Rocky Duan, Marcin Andrychowicz, Ankur Handa, Vikash Kumar, Bob McGrew, Alex Ray, Jonas Schneider, Peter Welinder, et al. Domain randomization and generative models for robotic grasping. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3482–3489. IEEE, 2018.

Matthew Veres, Medhat Moussa, and Graham W Taylor. Modeling grasp motor imagery through deep conditional generative models. *IEEE Robotics and Automation Letters*, 2(2):757–764, 2017.

Huiwen Zhang, Xiaoning Han, Mingliang Fu, and Weijia Zhou. Robot obstacle avoidance learning based on mixture models. *Journal of Robotics*, 2016, 2016.