



**SAPIENZA**  
UNIVERSITÀ DI ROMA

Master Thesis in Engineering in Computer Science

**Title**

Milad Kiwan

Advisor Prof. Daniele Nardi

March 2020

Dipartimento di Ingegneria Informatica, Automatica e Gestionale (DIAG)  
Sapienza Università di Roma

*Dedication.*

# Abstract

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

# Acknowledgements

Firstly, I would like to express my sincere gratitude to my teacher and advisor Prof. Daniele Nardi and his assistant 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

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>Introduction</b>	<b>1</b>
<b>1 Variational Auto-Encoder (VAEs)</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 VAE Structure . . . . .	2
1.2.1 Reparameterization Trick: . . . . .	4
1.3 Employment of VAEs in generative models for robotics . . . . .	4
<b>2 Gaussian mixture models (GMMs)</b>	<b>6</b>
2.1 Introduction . . . . .	6
2.2 Structure and Learning algorithm . . . . .	6
<b>3 Reinforcement Learning (RL)</b>	<b>9</b>
3.1 What is RL . . . . .	9
3.2 Exploitation of VAE and GMM in RL . . . . .	10
<b>4 Generative Adversarial Networks (GANs)</b>	<b>12</b>
4.1 Whats a GAN . . . . .	12
4.2 GAN Algorithm overview . . . . .	12
<b>Conclusion</b>	<b>15</b>
<b>Bibliography</b>	<b>17</b>

# Introduction

intro

# Variational Auto-Encoder (VAEs)

## Introduction

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 its 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 above.

## VAE Structure

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 it has to deal with, which are:

- Encoder network: gets the high-dimension input and transform it to into

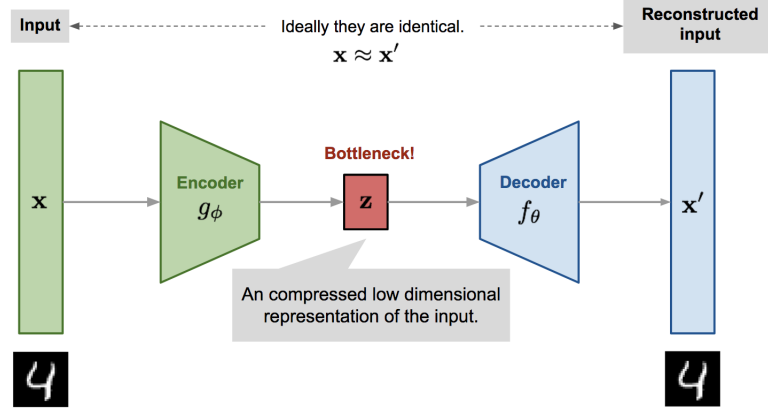


Figure 1.1: Autoencoder

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  $q_\theta$  instead of a fixed vector  $z$ , where  $q_\theta$  is parameterized by  $\theta$ , 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  $q_\theta$  has good parameters to enable the generative process. The way that makes the encoder to be able to produce  $q_\theta$  is by composing the bottleneck or the output of a mean  $\mu$  and a covariance matrix  $\Sigma$  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 KullbackLeibler (KL) divergence-which measures the distance between tow distributions- is introduced between the distribution produced by the encoder  $q_\theta(z \sim x_i)$  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:

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

Where  $z$  is sampled from  $q_\theta$  and  $\phi$  the decoder parameters, the purpose of the first term in poor words mean how much the decoder output is similar to



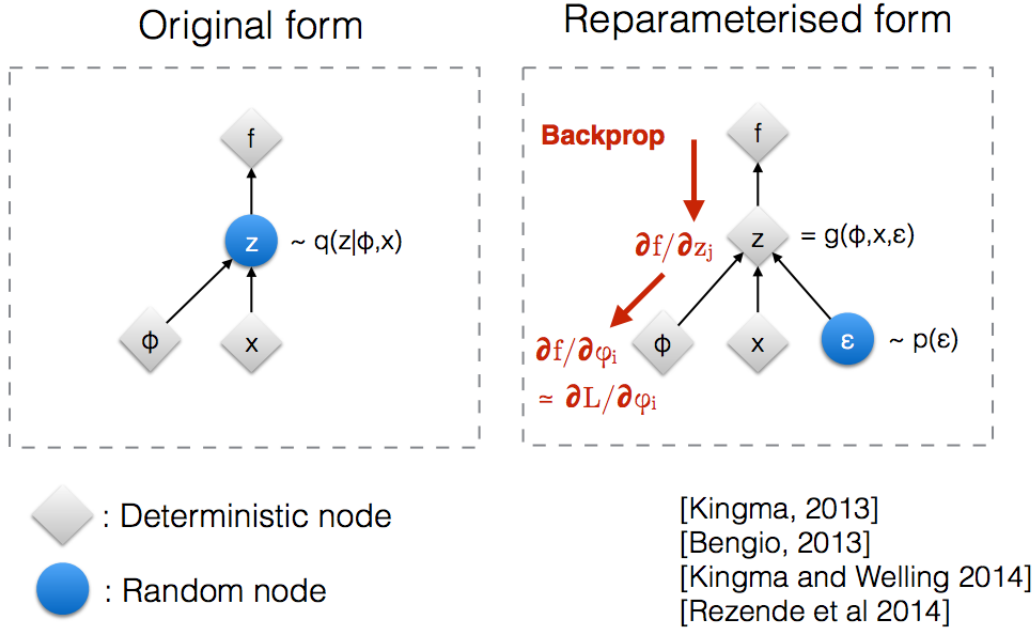


Figure 1.2: reparameterization trick

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  $\theta$  and  $\phi$  respectively. For stochastic gradient descent with step size  $\rho$ , 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)  $\epsilon$  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

## Employment of VAEs 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 is 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 cubes 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 image (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.

## Gaussian mixture models (GMMs)

### Introduction

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.

### Structure and Learning algorithm

The model is parameterized by two types of values, the mixture component weights are defined as  $\phi_k$  and the component means  $\mu_k$  and variances  $\sigma_k$  or covariances (for the multivariate case), 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 two 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  $\phi_k$ ,  $\mu_k$ , and  $\sigma_k$ .
- 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  $\phi_k$ ,  $\mu_k$ , and  $\sigma_k$ .

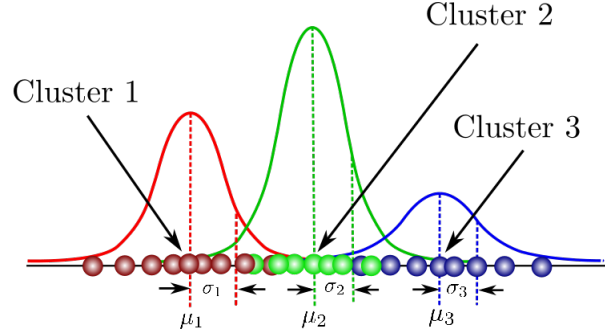


Figure 2.1

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  $X = x_1, \dots, x_N$ , to the component mean estimates  $\mu_1, \dots, \mu_k$ . 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  $\sigma_1^2, \dots, \sigma_k^2 = \frac{1}{N} \sum_{i=1}^K (x_i - \hat{x})^2 = 1$ , 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  $P(C_k) = \phi_1, \dots, \phi_k = \frac{1}{K}$  while the E-step computes the probability that  $x_i$  is generated by component  $C_k$ :

$$p(C_j | x_i) = \frac{p(x_i | C_j)p(C_j)}{p(x_i)} = \frac{p(x_i | C_j)p(C_j)}{\sum_i p(x_i | C_j)p(C_j)} \quad (2.1)$$

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

$$\mu_j = \frac{\sum_i p(C_j | x_i)x_i}{\sum_i p(C_j | x_i)} \quad (2.2)$$

$$\sigma_j^2 = \frac{\sum_i p(C_j | x_i)(x_i - \mu_j)(x_i - \mu_j)^T}{\sum_i p(C_j | x_i)} \quad (2.3)$$

$$p(C_j) = \frac{\sum_i p(C_j | x_i)}{N} \quad (2.4)$$

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. 2.1, 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 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 experts 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)

## What is 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 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 take the form of a Markov Decision Process (MDP) is a mathematical system used for modeling decision making. We use a tuple  $(S, A, P, R, \gamma)$  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.

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's 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't 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, and picking the algorithm whose more fit to the task taking into account the domain spaces of actions and states, you eventually will get good result, 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. 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 time the algorithm used was not mentioned.

## 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 the 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 which makes 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 robots 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 softmax activation function that consists of tow operations on the last convolutional layer of the encoder as follow:

$$s_{cij} = \frac{e^{\frac{a_{cij}}{\alpha}}}{\sum_{ij} e^{\frac{a_{cij}}{\alpha}}} \quad (3.1)$$

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

$$f_c = (\sum_i i s_{cij}, \sum_j j s_{cij}) \quad (3.2)$$

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 wide 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 learn 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  $\mathbb{G}$  as to be same as the state space  $\mathbb{S}$ . 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  $\mathbb{G}$  and  $\mathbb{S}$ , 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.



# Generative Adversarial Networks (GANs)

## Whats a GAN

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 a two-player game, the two networks could have arbitrary architecture and they are trained simultaneously, one neural network, called the discriminator, is designed as a 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.

## GAN Algorithm overview

To describe the GAN algorithm Fig. 4.1 it is possible to start from either the generator, or the discriminator, because as mentioned in the previous section it corresponds to a two-player game, like in chess, conventionally the white starts, but even if black starts that does not change the essence of the game. However, let's start with the more interesting one which is the generative model.

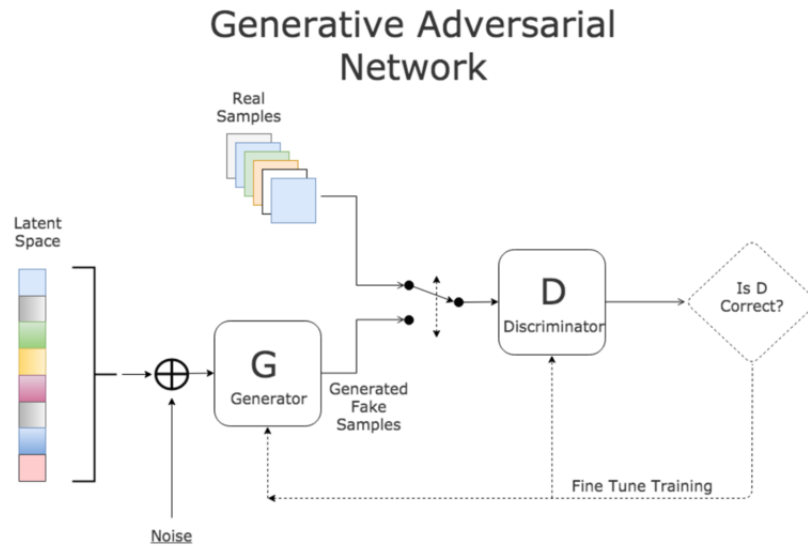


Figure 4.1: GAN Architecture

The generator model takes a fixed-length random noise vector as input or 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 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 gets 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 are 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 two loss functions for each of the two networks as introduced in Goodfellow et al. [2014], the generator tries to minimize the following function while the discriminator tries to maximize it:

$$E_x[\log D(x)] + E_z[\log (1 - D(G(z)))] \quad (4.1)$$

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  $\log(1 - D(G(z)))$ .

## Conclusion

## Bibliography

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