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 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

intro

1

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

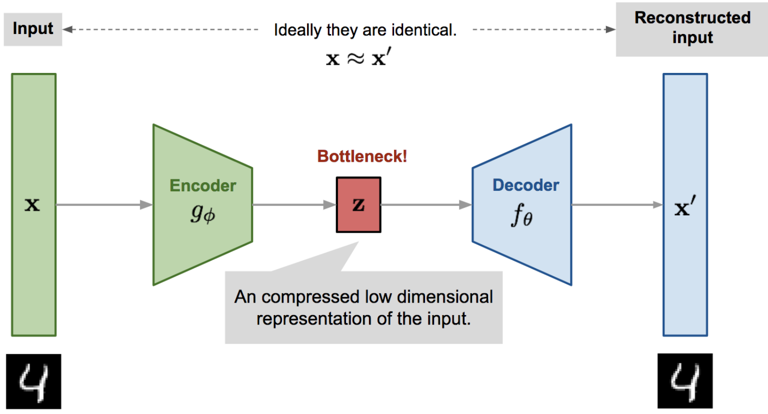

Figure 1.1: Autoencoder

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_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 KullbackLeibler (KL) divergence-which measures the distance between tow distributions- is introduced between the distribution produced by the encoder $q_\theta(z \ V \ x_i)$ 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:

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

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

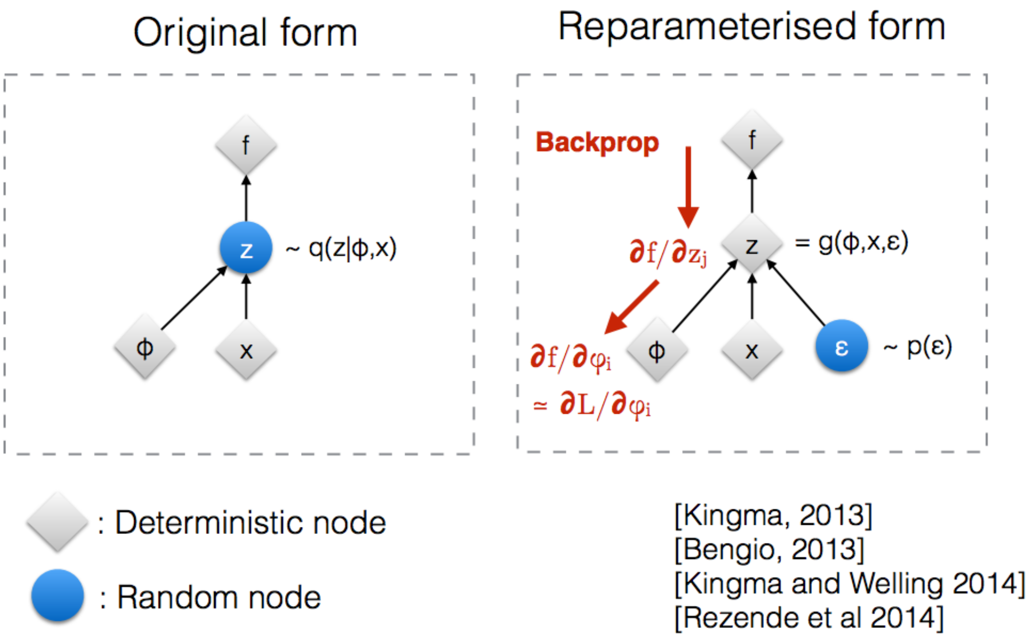

Figure 1.2: reparameterization trick

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 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

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.

2

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 k and the component means k and variances k or covariances (for the multivariant 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 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.

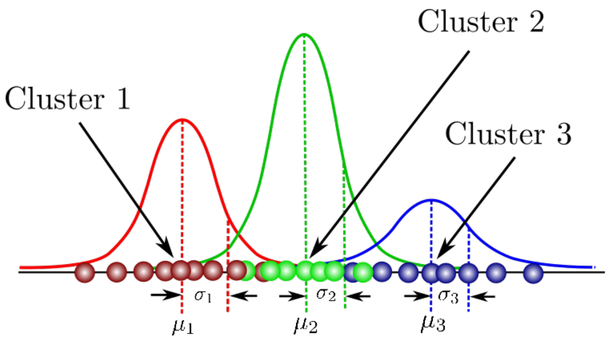

Figure 2.1

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 $X = x_1, ..., x_N$, 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:

$$p(C_j \mid x_i) = \frac{p(x_i \mid C_j)p(C_j)}{p(x_i)} = \frac{p(x_i \mid C_j)p(C_j)}{\sum_i p(x_i \mid C_j)p(C_j)} \tag{2.1}$$

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

$$\mu_j = \frac{\sum_i p(C_j \mid x_i)x_i}{\sum_i p(C_j \mid x_i)} \tag{2.2}$$

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

$$p(C_j) = \frac{\sum_i p(C_j \mid x_i)}{N} \tag{2.4}$$

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.

3

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, 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 st 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 cant 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 soft-max activation function that consists of tow operations on the last convolutional layer of the encoder as follow:

$$s_{cij} = \frac{e^{\frac{a_{cij}}{\alpha}}}{\sum_{\acute{i}\acute{j}} e^{\frac{a_{c\acute{i}\acute{j}}}{\alpha}}} \tag{3.1}$$

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

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

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 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.

4

Generative Adversarial Networks (GANs)

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.

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.

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.

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