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 z of the environment given some piece of relevant information q, such as the location of the camera, or a particular object model hypothesis

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.

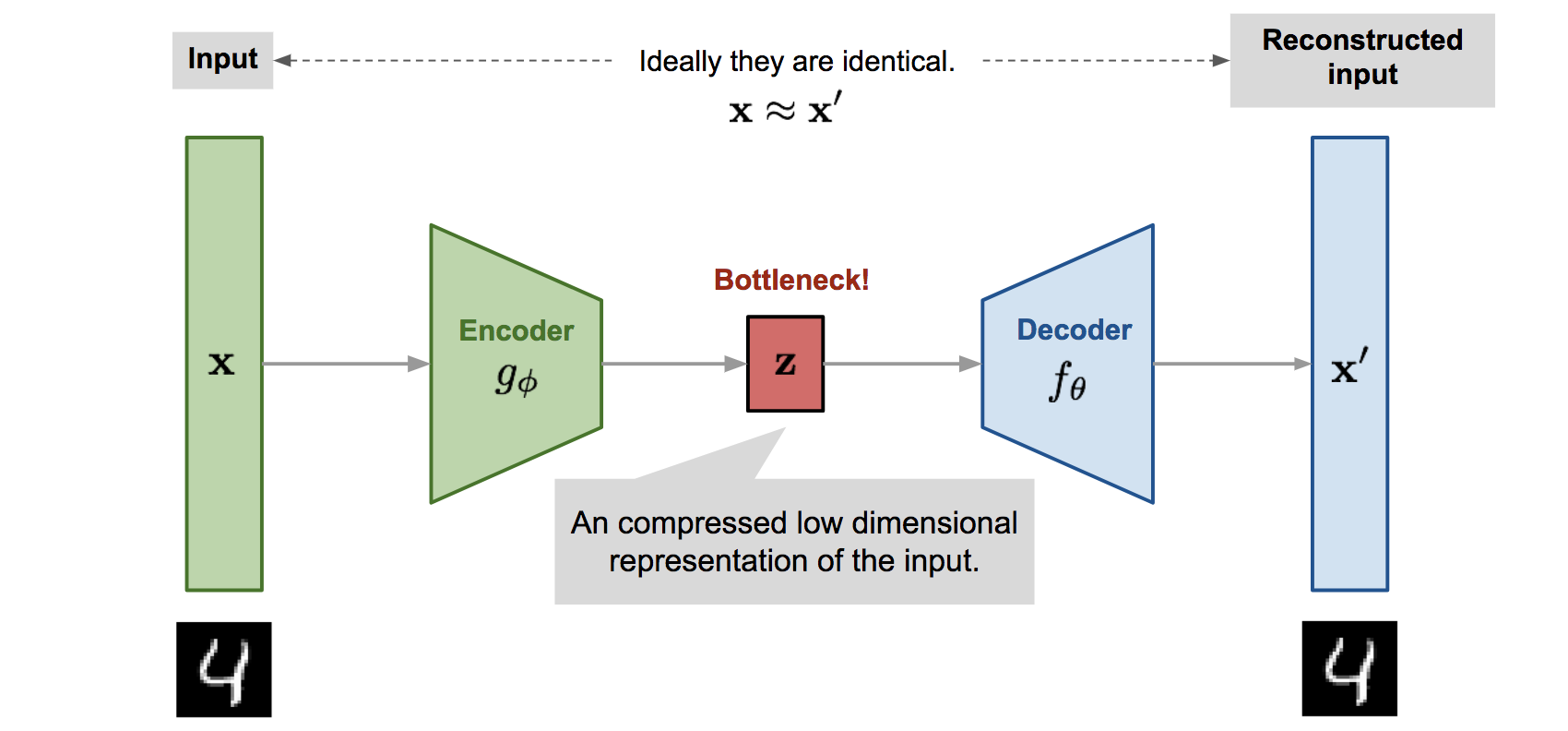
**Variational Auto-Encoder (VAEs):**

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’s 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.

Before starting to talk about the 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() 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 & Salakhutdinov, 2006 [1].

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 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.

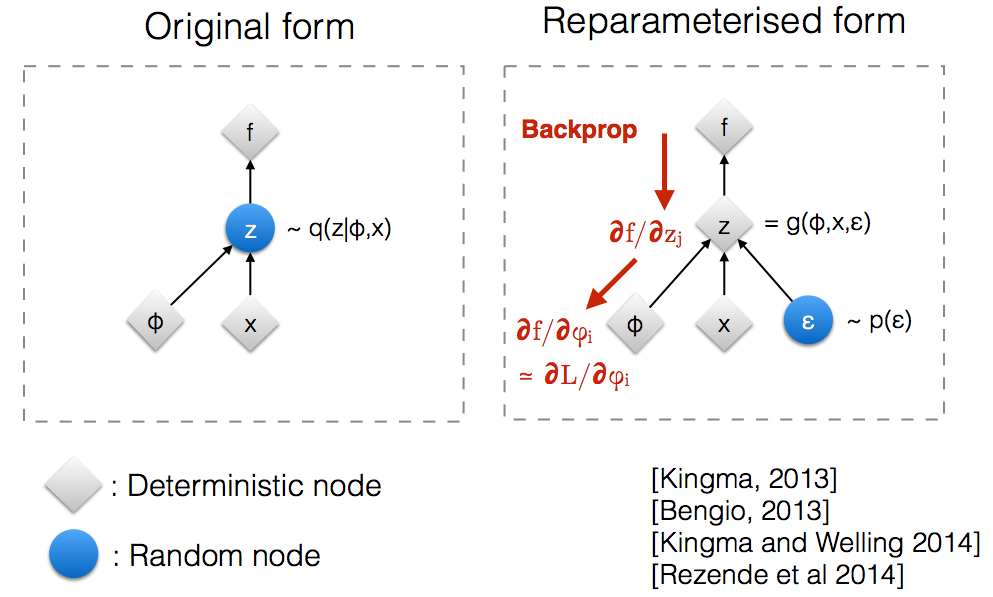


We can see already how the auto-encoder networks can give us an efficient way to impressively represent the data and in lower dimension. So the accomplishment of solutions for the problematics we talked about at beginning of this session, is all about about how we build the bottleneck layer or what will call from now on vector z. The VAE [2] 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 qθ instead of a fixed vector z, where qθ 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 qθ has good parameters to enable the generative process. The way that makes the encoder to be able to produce qθ is by composing the bottleneck or the output of a mean  and a covariance matrix  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 and a unit Gaussian distribution (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 datapointas follow:

Where z is sampled from qθ 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“. 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 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 doesn't go through the random node z, therefore we have to implement some trick to circumvent this issue. The reparameterization trick [2] 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(), we are basically expressing the random variable z as a deterministic variable (mean plus element-wise between standard deviation and ε) where gθ(.) is a differentiable transformation and .



Later in this work we will see where and how the VAEs have been employed and show there effectiveness in various applications of generative models in robotics.

**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 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 modelling 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 st+1. Ra(s,s′) is the expected reward from taking action a and transitioning to s′ . γ ∈ [0, 1] is a discount factor, to discount the future reward.

There are tow different groups of algorithms that implement RL, one of them is the model-based algorithms, they 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. The other one is model-free algorithms, for which there is no assumption about the world.

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.

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 expirience 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 regards on which algorithm has been employed the most, actually most of time the algorithm used is not mentioned.

**Gaussian mixture models (GMMs):**

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 ϕk and the component means *μk* and variances σk or covariances (for the multivariant case) Σ, the mixture component weights has a constraint that is: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 Ck (P(Ck | xi)) for each data point xi ∈X given the model parameters ϕk, *μk,* and σk..
* 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 ϕk, *μk,* and σk..

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={x1, ..., xN}, to the component mean estimates μ1, … , *μk.* E.g. for K=3 and N=100, set *μ1*= x45, *μ2* = x32, *μ3* = x10.
* Set all component variance estimates to the sample variance

σ12, ..., σk2= , where is the sample mean.

* Set all component distribution prior estimates to the uniform distribution

P(Ck) = ϕ1,..., ϕk =

while the E-step the probability that xi is generated by component Ck which will be used in the M-step where the parameters are updated as follow:

as we can deduce that the GMM is suitable model when recovering the distribution of the data is needed, since it can produce more complexed distribution composed of jointed k gaussians, for example if we have different sources from which the data is provided. Back to our main argument

**Generative adversarial networks:**

**References:**

[1] Reducing the Dimensionality of Data with Neural Networks. Science 313, 504 (2006); G. E. Hinton, et al.

[2] Auto-Encoding Variational Bayes, Diederik P Kingma, Max Welling 2013