Machine Learning (CAP6610) Project: Deep Generative Models with an emphasis on Variational Autoencoders and Generative Adversarial Networks

Bhargav Ram K S
UFID: 40328191

Deepak Maniyani
UFID: 93994924

Srividya Vaishnavi Surampudi
UFID: 64803004

Abstract

In this group project, we will focus on a getting a background on recent advances in Deep Generative Models, with special emphasis on two popular models, namely Variational Autoencoders(VAE) and Generative Adversarial Network(GAN). We shall first describe the density estimation problem, and how VAE and GAN are both different varieties of solutions for density estimation. We shall describe fundamental theoretical principles and motivations behind these models, their training and optimization, advantages and pitfalls in the models and training process. We will then survey some popular applications of VAE and GAN. We will examine the results of image generation by training both Variational Autoencoders and Deep Convolutional GAN (DCGAN) on MNIST and Fashion MNIST datasets, and interpret the results.

1 Introduction

Generative models involve learning models of distribution $p(\boldsymbol{x})$, where $\boldsymbol{x}$ is a vector. Generally, the idea of learning a model involves capturing inter-dependencies between features of the $\boldsymbol{x}$. Generative models learn a model estimate p_{model} of a distribution by using samples of the original distribution p_{data}, usually in the form of a training dataset. A good generative model decides that a datapoint which highly resembles real data gets high probability, and a datapoint which is randomly generated gets low probability. One of the focal topic of our project is to create new datapoints which highly resemble real data, rather than discriminate between datapoints which are likely to be real from random noise. We explore the use of generative models for the task of creating new unseen data, highly similar to existing data. We assume there is an oracle distribution p_{data} from which the existing data is sampled. We want to learn a p_{model}, which is highly similar to p_{data} in terms of sampling behavior. That is, samples generated from p_{model} look like real data sampled out of p_{data}.

As part of Machine Learning class, we have covered supervised generative models, such as Discriminant Analysis methods and unsupervised generative models, such as Gaussian Mixture Models. In context of supervised learning, we use generative models as a way to model the joint distribution $p(\boldsymbol{x}, y)$. In context of unsupervised learning, we can use generative models address Density Estimation, a key problem in unsupervised learning. Density Estimation involves learning the distribution $p_{model}(\boldsymbol{x})$ over a random vector $\boldsymbol{x}$. Once we model the density, we can sample new plausible values of data distribution, or calculate the likelihood of a new data point. Now, we shall look at models which estimate the data distribution on the basis of maximum-likelihood estimation[22]. If we are given a training set $\{\boldsymbol{x}_i\}_{i=1}^{I}$, then our ideal parameters of p_{model} is given by,

$$\hat{\boldsymbol{\phi}} = \text{argmax}_{\phi} \left[\sum_{i=1}^{I} \log \left[p\left(\boldsymbol{x}_i | \boldsymbol{\phi} \right) \right] \right]$$

Density estimation models are classified into two types, explicit and implicit density estimation models. Variational Autoencoder is an example of an explicit density estimation model, whereas Generative Adversarial Network is an example of implicit density estimation model. In case of explicit density models, we make an attempt to directly define a density model $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$, whereas in implicit density models, we are more interested in generating samples from the data distribution rather than directly modelling it. A key challenge is when we want to design a computationally tractable explicit density model which captures complexity of data to a high degree, in case of which we can follow two strategies. Firstly, we can carefully design models with a structure that guarantees tractability. Otherwise, we can design models whose likelihood admits computationally tractable approximations. Variational methods falls into the second category and it operates by defining a tractable lower bound $\mathcal{L}$ for the data likelihood. That is,

$$\mathcal{L}(\boldsymbol{x}; \boldsymbol{\theta}) \leq \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$$

Once we define a tractable $\mathcal{L}$, we can maximize it instead of the original log-likelihood, which is bounded below by $\mathcal{L}$. We will now describe Variational auto-encoders, which define $\mathcal{L}$ by exploiting properties of neural networks as complex function approximators.

2 Variational Autoencoders

For our survey on Variational Autoencoders, we have referred the original paper by Kingma et. al. [17] and a tutorial paper by Doersch et al.[8]. Another immensely helpful source for understanding the theoretical foundations behind Variational Autoencoders was a blog post by Borealis AI [27].

A Variational Autoencoder can be described as a non-linear latent variable model. We have seen examples of latent variable models, such as Gaussian Mixture Models, Factor Analysis and probabilistic PCA. Let us now briefly visit latent variable models and understand how they might be useful to create models which enable us to get high fidelity samples from distributions modelled to be statistically similar to the data distribution.

2.1 Latent Variable Models

When we want to synthesize new data by training a generative model, the complexity of the model and requisite computational power needed for training increases with the interdependency between the dimensions of the data. If we consider a model for generating realistic human faces, there is a strong relationship between different parts of the face, and to generate realistic samples requires a complicated model. The model needs to consider factors like skin-tone, symmetry, continuity and smooth transition across the image of the generated face. To simplify the process, the model can make an intermediate decision in the form of a latent variable, corresponding to the kind of face it wants to generate (example: a face of a hispanic male) and use this decision to actually synthesize a sample. Once we have inferred the latent variable model parameters, we can then sample $\boldsymbol{h}^*$ from the latent variable via $Pr(\boldsymbol{h})$ and later use the conditional, $Pr(\boldsymbol{x}|\boldsymbol{h} = \boldsymbol{h}^*)$ to sample $\boldsymbol{x}^*$. The latent variable sample $\boldsymbol{h}^*$ helps us capture important features we want in our sample and conditioned on that, we can then sample $\boldsymbol{x}^*$.

In a latent variable model, we express the likelihood of a point via marginalization over a latent variable[1], i.e. $\Pr(\boldsymbol{x}) = \int \Pr(\boldsymbol{x}, \boldsymbol{h}) d\boldsymbol{h} = \int \Pr(\boldsymbol{x}|\boldsymbol{h}) \Pr(\boldsymbol{h}) d\boldsymbol{h}$. We have previously seen examples of linear latent variable models in Factor Analysis, where we describe the model as,

$$\begin{aligned}
\boldsymbol{y} &\sim \mathcal{N}(0, \boldsymbol{I}) \\
\boldsymbol{x}|\boldsymbol{y} &\sim \mathcal{N}\left(\boldsymbol{A}\boldsymbol{y} + \boldsymbol{\mu}, \text{Diag}\left(\sigma_1^2, \ldots, \sigma_m^2\right)\right)
\end{aligned}$$

Here, we describe the mean $\boldsymbol{\mu}_{x|y} = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{\mu}$, a linear function in the latent $\boldsymbol{y}$. However, to model more complex data distributions, we can exploit the properties of neural networks as non-linear function approximators to define non-linear latent variable models, which is exactly what a Variational Autoencoder does. The prior and likelihood are defined as:

$$\begin{aligned}
\boldsymbol{h} &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi} &\sim \mathcal{N}\left(\mathbf{f}[\boldsymbol{h}, \boldsymbol{\phi}], \sigma^2 \mathbf{I}\right)
\end{aligned}$$

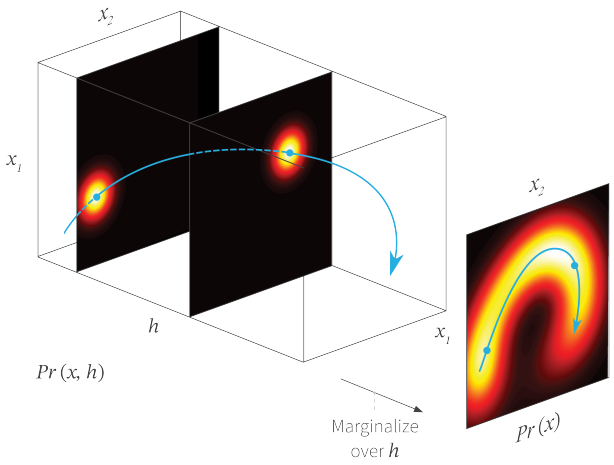

Figure 1: Non-Linear Latent variable model as infinite weighted mixture of gaussians. Source: `https://www.borealisai.com/en/blog/tutorial-5-variational-auto-encoders/`

Here, $\mathbf{f}[\boldsymbol{h}, \boldsymbol{\phi}]$ is a neural network with parameters $\boldsymbol{\phi}$. This model can be visualized in Figure 1. At each value of $\boldsymbol{h}$, $\boldsymbol{x}|\boldsymbol{h}$ is defined by a spherical gaussian with mean as an output of neural network. The model now can be thought of as a weighted mixture of infinite number of isotropic Gaussians with different means, as shown in Figure 1.

Variational autoencoders act as good models for generating real samples of data and one of the contributing factor is the latent variable formulation of VAE. Once we learn parameters of the network, the latent variable $\boldsymbol{h}$ has a good compact representation of important features of the data $\boldsymbol{x}$. We can then use the relatively simple Gaussian formulations of latent variable prior and the conditional $p(\boldsymbol{x}|\boldsymbol{h})$ to sample new, realistic data. Now we look at a few operations we are interested in performing once we define the non-linear latent variable model. We might be interested in computing the likelihood of a new point via marginalization, namely,

$$\begin{aligned}
\Pr(\boldsymbol{x}) &= \int \Pr(\boldsymbol{x}, \boldsymbol{h}|\boldsymbol{\phi}) d\boldsymbol{h} \\
&= \int \Pr(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi}) \Pr(\boldsymbol{h}) d\boldsymbol{h} \\
&= \int \mathcal{N}\left(\mathbf{f}[\boldsymbol{h}, \boldsymbol{\phi}], \sigma^2 \mathbf{I}\right) \mathcal{N}(\mathbf{0}, \mathbf{I}) d\boldsymbol{h}
\end{aligned}$$

But since there is a dependence of the function on the nonlinear function mean $\mathbf{f}[\boldsymbol{h}, \boldsymbol{\phi}]$, the integral is computationally intractable. Now that calculation of $\Pr(\boldsymbol{x})$ is intractable, the posterior calculation, $\Pr(\boldsymbol{h}|\boldsymbol{x}) = \frac{\Pr(\boldsymbol{x}|\boldsymbol{h}) \Pr(\boldsymbol{h})}{\Pr(\boldsymbol{x})}$ is also a difficult task. A helpful observation at this stage is that using sampling to obtain a data point $\boldsymbol{x}^*$ is not difficult. We can sample a $\boldsymbol{h}^*$ of the latent variable, and based on this we can sample $\boldsymbol{x}^*$ from the conditional $\boldsymbol{x}|(\boldsymbol{h} = \boldsymbol{h}^*) \sim \mathcal{N}\left(\mathbf{f}[\boldsymbol{h}^*, \boldsymbol{\phi}], \sigma^2 \mathbf{I}\right)$.

2.2 Evidence Lower Bound (ELBO)

Previously, we saw that $Pr(\boldsymbol{x})$ is an integral with no closed form expression. This realization is problematic in two ways. The main reason of introducing non-linear latent variable model was to express a complex $Pr(\boldsymbol{x})$ as a marginalization using simple distributions of $Pr(\boldsymbol{h})$ and $Pr(\boldsymbol{x}|\boldsymbol{h})$, which we now find out to be a hard task. The second reason is that being able to compute closed form $Pr(\boldsymbol{x})$ will help us in parameter estimation using negative log likelihood. We now look at a method by which we can lower bound the log likelihood (i.e. $\log Pr(\boldsymbol{x})$) and utilize the lower bound expression for learning model parameters. Remember that our original objective was $\max_{\boldsymbol{\phi}} \left[\sum_{i=1}^{I} \log\left[p\left(\boldsymbol{x}_i|\boldsymbol{\phi}\right)\right]\right]$. Let us look at the task of lower bounding $\log\left[p\left(\boldsymbol{x}_i|\boldsymbol{\phi}\right)\right]$. We now take advantage of the concaveness of $\log$ and introduce an auxiliary $q(\boldsymbol{h})$ defined over latent variable to

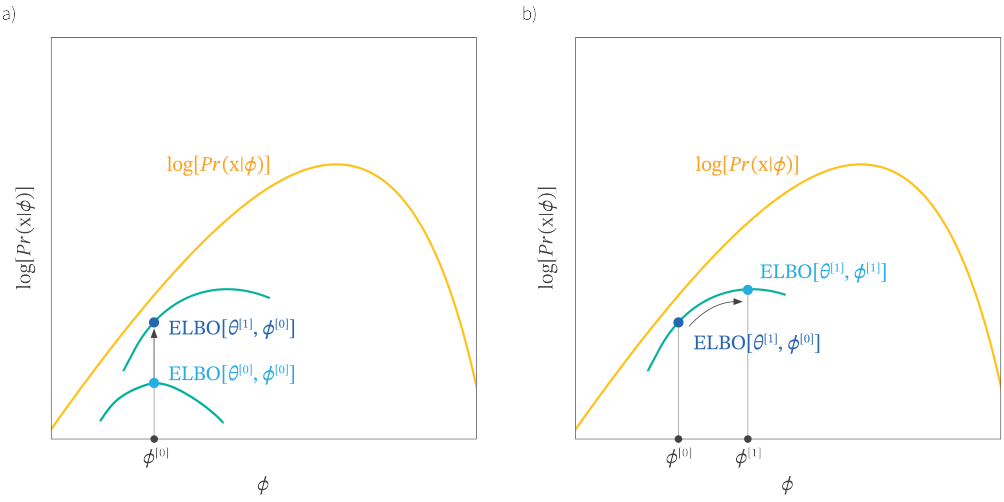

Figure 2: The Evidence Lower Bound (ELBO) is a function of parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ and lies below the likelihood everywhere. Source: `https://www.borealisai.com/en/blog/tutorial-5-variational-auto-encoders/`

lower bound the log likelihood of a point.

$$\begin{aligned}\log[\Pr(\boldsymbol{x}|\boldsymbol{\phi})] &= \log\left[\int \Pr(\boldsymbol{x},\boldsymbol{h}|\boldsymbol{\phi})d\boldsymbol{h}\right] \\ &= \log\left[\int q(\boldsymbol{h})\frac{\Pr(\boldsymbol{x},\boldsymbol{h}|\boldsymbol{\phi})}{q(\boldsymbol{h})}d\boldsymbol{h}\right]\end{aligned}$$

Now if we use the fact that the log likelihood function is concave, we can lower bound the function as follows using Jensen's Inequality. This step is similar to the one where we use Jensen's inequality for constructing the Majorization function for Expectation-Maximization algorithm in class. We have,

$$\log\left[\int q(\boldsymbol{h})\frac{\Pr(\boldsymbol{x},\boldsymbol{h}|\boldsymbol{\phi})}{q(\boldsymbol{h})}d\boldsymbol{h}\right] \geq \int q(\boldsymbol{h})\log\left[\frac{\Pr(\boldsymbol{x},\boldsymbol{h}|\boldsymbol{\phi})}{q(\boldsymbol{h})}\right]d\boldsymbol{h} = \text{ELBO}[\boldsymbol{\theta},\boldsymbol{\phi}]$$

The RHS term is now defined as $\text{ELBO}[\boldsymbol{\theta},\boldsymbol{\phi}]$ and this is the entity that we want to now maximize. We can now use $\text{ELBO}[\boldsymbol{\theta},\boldsymbol{\phi}]$ as sort of a proxy for the original objective and maximize it instead and obtain optimal parameters $\boldsymbol{\theta},\boldsymbol{\phi}$. We will now try to get some intuition regarding ELBO via Figure 2. Consider θ to be fixed. Now ELBO is a function lying entirely below the original objective and it can be maximized by varying $\boldsymbol{\phi}$. Depending on how we fix $\boldsymbol{\theta}$, the $\text{ELBO}[\boldsymbol{\phi}]$ can be closer or farther from the original objective. We are now interested in getting a tighter lower bound for a given $\boldsymbol{\phi}$ by choosing such a $\boldsymbol{\theta}$ that the ELBO and the likelihood objective are as close as possible. More precisely, among the family of ELBO functions with fixed $\boldsymbol{\phi}$, we want to find $\boldsymbol{\theta}$ to result in that function which is closest to the original log likelihood. Mathematically, we have,

$$\begin{aligned}\text{ELBO}[\boldsymbol{\theta},\boldsymbol{\phi}] &= \int q(\boldsymbol{h}|\boldsymbol{\theta})\log\left[\frac{\Pr(\boldsymbol{x},\boldsymbol{h}|\boldsymbol{\phi})}{q(\boldsymbol{h}|\boldsymbol{\theta})}\right]d\boldsymbol{h} \\ &= \int q(\boldsymbol{h}|\boldsymbol{\theta})\log\left[\frac{\Pr(\boldsymbol{h}|\boldsymbol{x},\boldsymbol{\phi})\Pr(\boldsymbol{x}|\boldsymbol{\phi})}{q(\boldsymbol{h}|\boldsymbol{\theta})}\right]d\boldsymbol{h} \\ &= \int q(\boldsymbol{h}|\boldsymbol{\theta})\log[\Pr(\boldsymbol{x}|\boldsymbol{\phi})]d\boldsymbol{h} + \int q(\boldsymbol{h}|\boldsymbol{\theta})\log\left[\frac{\Pr(\boldsymbol{h}|\boldsymbol{x},\boldsymbol{\phi})}{q(\boldsymbol{h}|\boldsymbol{\theta})}\right]d\boldsymbol{h} \\ &= \log[\Pr(\boldsymbol{x}|\boldsymbol{\phi})] + \int q(\boldsymbol{h}|\boldsymbol{\theta})\log\left[\frac{\Pr(\boldsymbol{h}|\boldsymbol{x},\boldsymbol{\phi})}{q(\boldsymbol{h}|\boldsymbol{\theta})}\right]d\boldsymbol{h} \\ &= \log[\Pr(\boldsymbol{x}|\boldsymbol{\phi})] - \mathrm{D}_{KL}[q(\boldsymbol{h}|\boldsymbol{\theta})\|\Pr(\boldsymbol{h}|\boldsymbol{x},\boldsymbol{\phi})]\end{aligned}$$

Here, $D_{KL}(.||.)$ term is the Kullback-Leibler divergence which is always non-negative and attains zero when $q(\boldsymbol{h}|\boldsymbol{\theta}) = \Pr(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\phi})$. That is, if we choose a distribution $q(\boldsymbol{h}|\boldsymbol{\theta})$ as close as possible to the posterior $\Pr(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\phi})$, better the lower bound approximation over log-likelihood we get.

2.2.1 First shot at a solution: EM algorithm

Perhaps, we can now apply Expectation Maximization algorithm. At every iteration, we can choose parameters $\boldsymbol{\theta}$ to make $q(\boldsymbol{h}|\boldsymbol{\theta}) = \Pr(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\phi})$ and then via maximization step, we can infer $\boldsymbol{\phi}$ to maximize lower bound. However, this method fails for our non-linear latent variable model because $\Pr(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\phi})$ is intractable as discussed earlier, hence making the expectation step prohibitive.

2.2.2 Reconstruction error minus KL on the prior

Now, we express ELBO alternately, as the real formulation relevant to the Variational Autoencoder.

$$\begin{aligned}
\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] &= \int q(\boldsymbol{h}|\boldsymbol{\theta}) \log \left[\frac{\Pr(\boldsymbol{x}, \boldsymbol{h}|\boldsymbol{\phi})}{q(\boldsymbol{h}|\boldsymbol{\theta})} \right] d\boldsymbol{h} \\
&= \int q(\boldsymbol{h}|\boldsymbol{\theta}) \log \left[\frac{\Pr(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi}) \Pr(\boldsymbol{h})}{q(\boldsymbol{h}|\boldsymbol{\theta})} \right] d\boldsymbol{h} \\
&= \int q(\boldsymbol{h}|\boldsymbol{\theta}) \log[\Pr(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi})] d\boldsymbol{h} + \int q(\boldsymbol{h}|\boldsymbol{\theta}) \log \left[\frac{\Pr(\boldsymbol{h})}{q(\boldsymbol{h}|\boldsymbol{\theta})} \right] d\boldsymbol{h} \\
&= \underbrace{\int q(\boldsymbol{h}|\boldsymbol{\theta}) \log[\Pr(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi})] d\boldsymbol{h}}_{\text{Reconstruction Error}} - \underbrace{\mathbf{D}_{KL}[q(\boldsymbol{h}|\boldsymbol{\theta}), \Pr(\boldsymbol{h})]}_{\text{KL to prior}}
\end{aligned}$$

The first term is the reconstruction error, which measures the on-average agreement between the data and the hidden variable that generated the data. That is, we assign higher mass to a latent variable which resulted in the datapoint generation. The second term suggests that degree of match between the auxiliary distribution and the prior.

2.3 Variational Approximation

We have seen before that if we choose $q(\boldsymbol{h}|\boldsymbol{\theta})$ to be the posterior $\Pr(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\phi})$, we get a tight bound for the ELBO, but the posterior is intractable. Now the variational step includes choosing $q(\boldsymbol{h}|\boldsymbol{\theta})$ as $q(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta})$ instead. (After all, $q(\boldsymbol{h}|\boldsymbol{\theta})$ was introduced in just the lower bounding step to use Jensen's inequality, so this decision on making it conditional on $\boldsymbol{x}$ isn't arbitrary). Since $\Pr(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\phi})$ is conditioned on $\boldsymbol{x}$, it would make sense to make the q distribution also conditioned on $\boldsymbol{x}$. Now in the variational approximation, we approximate the intractable $\Pr(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\phi})$ by $q(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta})$, a simple Gaussian with it's mean and variance decided by a neural network $g[\boldsymbol{x}, \boldsymbol{\theta}]$ with parameters $\boldsymbol{\theta}$. That is,

$$q(\boldsymbol{h}|\boldsymbol{\theta}, \boldsymbol{x}) \sim \mathcal{N}\left(g_\mu[\boldsymbol{x}|\boldsymbol{\theta}], g_\sigma[\boldsymbol{x}|\boldsymbol{\theta}]\right)$$

2.4 Actual formulation of Variational Autoencoder

Now after the formulation of $q(\boldsymbol{h}|\boldsymbol{\theta}, \boldsymbol{x})$, we have our ELBO as,

$$\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] = \int q(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta}) \log[\Pr(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi})] d\boldsymbol{h} - \mathbf{D}_{KL}[q(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta}), \Pr(\boldsymbol{h})]$$

We now use sampling to approximate the first integral. We sample a $\boldsymbol{h}^*$ from the approximation $q(\boldsymbol{h}|\boldsymbol{\theta}, \boldsymbol{x})$, and we write the integral using this one single sample. The ELBO is now,

$$\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] \approx \log\left[\Pr\left(\boldsymbol{x}\,|\boldsymbol{h}^*|\,\boldsymbol{\phi}\right)\right] - \mathbf{D}_{KL}[q(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta}), \Pr(\boldsymbol{h})]$$

The KL term can be written using an identity[3] for KL divergence between two Gaussians. The prior of $\boldsymbol{h}$ is standard Gaussian and $q(\boldsymbol{h}|\boldsymbol{\theta}, \boldsymbol{x}) \sim \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$. We can write,

$$\mathbf{D}_{KL}[q(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta}), \Pr(\boldsymbol{h})] = \frac{1}{2}\left(\text{Tr}[\boldsymbol{\Sigma}] + \boldsymbol{\mu}^T\boldsymbol{\mu} - D - \log[\det[\boldsymbol{\Sigma}]]\right)$$

The whole process can be summarized as follows. We take an input $\boldsymbol{x}$ and use it to compute mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ of the q distribution via the neural network $g[\boldsymbol{x}, \boldsymbol{\theta}]$. We then sample a $\boldsymbol{h}^*$ from the q

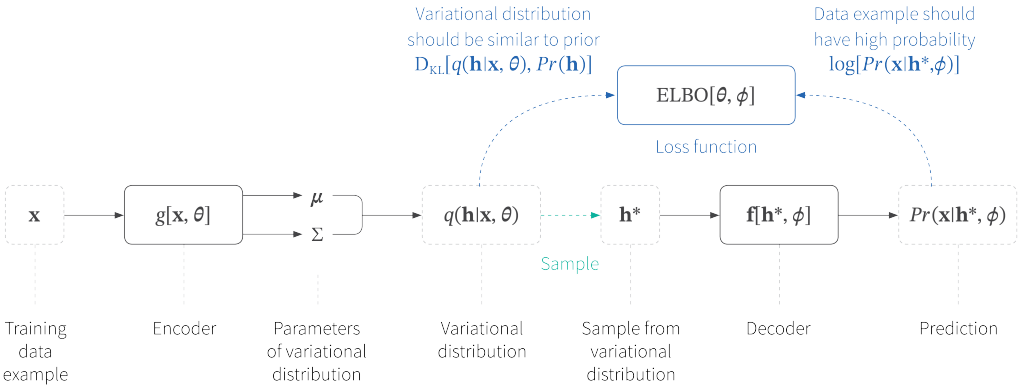

Figure 3: VAE initial architecture. Notice that in the network, there is a sampling step for $\boldsymbol{h}^*$ which makes gradient updates through back-propagation impossible. Source: `https://www.borealisai.com/en/blog/tutorial-5-variational-auto-encoders/`

distribution. Using this, we compute the KL term using estimated means $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$, and the sampling term using the definition of $\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi} \sim \mathcal{N}\left(\mathbf{f}[\boldsymbol{h}, \boldsymbol{\phi}], \sigma^2\mathbf{I}\right)$, and later combine both to compute the ELBO. This process is illustrated in the architecture diagram in Figure 3. By this diagram, we can see that the architecture is an autoencoder, trying to reconstruct the original data by using a non-linear latent variable model. Hence the name, Variational Autoencoder. It is also variational, because we make an approximation for latent variable posterior with a Gaussian whose mean and covariance is decided by a neural network.

2.5 Re-parametrization Trick

If we observe the architecture diagram in Figure 3, we note that to update the parameters $\boldsymbol{\theta}$ of the encoder $g[\boldsymbol{x}, \boldsymbol{\theta}]$ we must back-propagate the gradients via a sampling stage, which is not possible. However, we can use what is called a re-parametrization trick, which uses Cholesky decomposition to write any Gaussian sample as a affine function of a sample from a standard normal Gaussian.

$$\boldsymbol{h}^* = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\epsilon^*$$

This will allow us to decouple sampling from the main pipeline where we need gradient back-propagation to update parameters. We can now sample from a standard normal and then infer $\boldsymbol{h}^*$ by using mean and variance from encoder neural network. We will not face any problem while back-propagation now and we can infer all parameters. The resulting tweak is shown in architecture in Figure 4. The training procedure is now straightforward and it can be done using Stochastic Gradient Descent.

2.6 Benefits and Drawbacks of Variational Autoencoders

Previous models aimed at solving the task of sampling new data by modelling data distribution had the following shortcomings. One of the drawbacks is that these models required us to make strong structural assumptions of data, and hence they were suboptimal. Some of these models also depend on computationally expensive methods such as Markov Chain Monte Carlo. Variational Autoencoder, on the other hand, overcome the disadvantages faced by previous models. VAEs make weak assumptions about the structure of the data and has a faster training stage compared to previous models due to use of backpropagation in neural networks. VAEs also make use of deep neural networks as non-linear function approximators and this makes it powerful.

VAEs also have some drawbacks. Since the calculation of $\Pr(\boldsymbol{x})$ involves an integral over the latent variable, it is hard to compute a likelihood of a point in VAEs. We will have to resort to computationally expensive procedures like Markov-Chain Monte-Carlo (MCMC) methods. Generally, samples generated through VAE are imperfect. This is maybe because of the naive choice of the conditional as spherically gaussian. There are also issues where during training of the model, where the parameters can hit a local minimum, and after which the encoder always puts out mean and

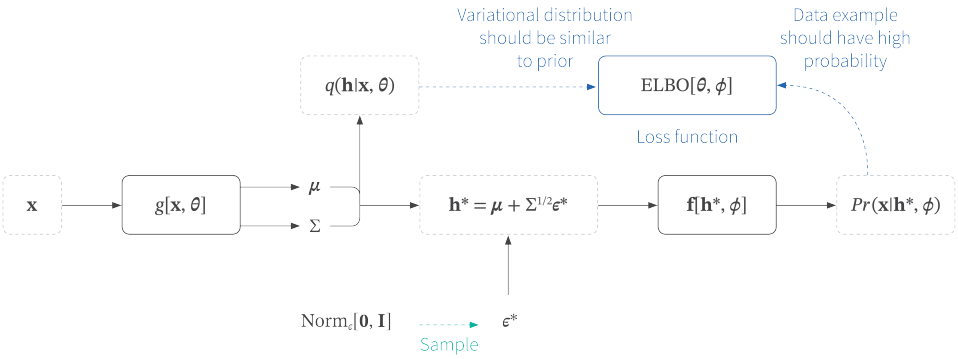

Figure 4: VAE final architecture. Notice that in the network, the sampling step is decoupled from the main pipeline. Parameters of neural networks, $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ can be updated from backpropagation. Source: `https://www.borealisai.com/en/blog/tutorial-5-variational-auto-encoders/`

covariance corresponding to the uninformative latent variable prior. This phenomenon is known as posterior collapse. One proposed solution[11, 5] is to introduce the KL divergence term in the ELBO formulation only gradually. That is, we follow a annealing schedule where β(see below equation) is gradually increased from 0 to 1 as training progresses.

$$\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] = \int q(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta}) \log[\Pr(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi})] d\boldsymbol{h} - \beta \mathbf{D}_{KL}[q(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta}), \Pr(\boldsymbol{h})]$$

2.7 Applications of Variational Autoencoders

We will describe a few applications of VAEs for vision and language modeling problems. Deep Recurrent Attentive Writer (DRAW) architecture, proposed by Gregor et. al.[14] uses Variational Autoencoder along with spatial attention mechanism to mimic the foviation of human eye to iteratively generate better images. Maaløe et. al.[19] extended deep generative models with auxiliary latent variables to achieve state-of-the-art results on several semi-supervised learning problems, where only a small portion of the dataset is labelled. Bowman et. al.[4] utilized Variational Autoencoders for a Recurrent Neural Network based language model, which was able to generate sentences with particular style, topic and other syntactic properties. They were able to navigate the latent space to successively obtain sentences which meaningfully integrate in-between other sentences.

3 Generative Adversarial Networks(GAN)

Previously, we described Variational Autoencoders, an explicit density estimation framework where we defined a variational lower bound approximation $\mathcal{L}$ to the intractable log likelihood term. Our previous aim was to actually approximate the density function in a closed form. However, there are implicit density models, a class of models whose aim is not to model the density function explicitly, but rather be able to sample accurately from the original model distribution, $p_{model}(\boldsymbol{x})$. Generative Adversarial Nets are one of the most popular models under this category. GAN has some advantages over VAE, namely its ability to produce better and clearer samples in case of tasks like image synthesis. Also, in case of GAN, we do not have to do a variational lower bounding step. For Generative Adversarial Networks, we have referred the original paper by Goodfellow et. al.[13] and a NIPS tutorial paper by Goodfellow[12], where he offers a broad explanation of the model.

3.1 Model Description of GAN

GAN is based on a two-player game between a generator and a discriminator, both usually defined as neural networks. The intention of the generator is to progressively generate samples(say images) that are likely to have been generated from the original model distribution, $p_{model}(\boldsymbol{x})$. The discriminator, on the other hand tries to examine real training samples and the samples from the generator and carry

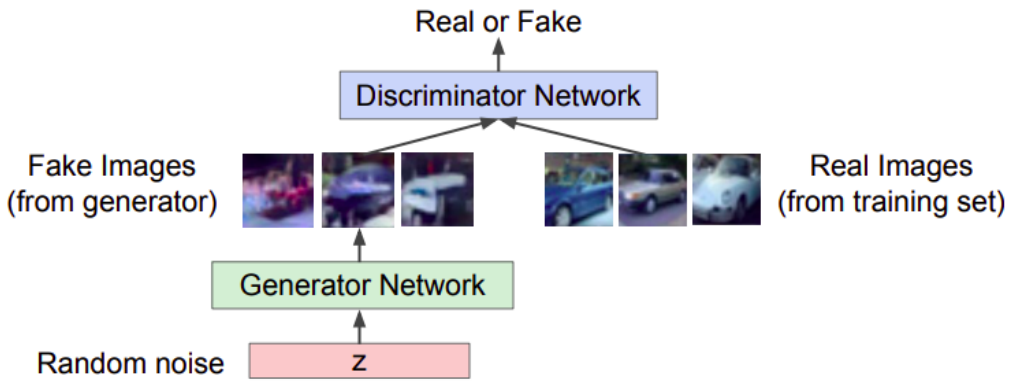

Figure 5: Architecture of a Generative Adversarial Network. Source: `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf`

out a supervised learning problem classifying these samples as either real or fake. For a generator to truly succeed in the two-player game, it has to progressively learn to generate samples from the $p_{model}(\boldsymbol{x})$.

The generator and discriminator functions are represented by G and D respectively. Generally, G and D are neural networks with parameters/weights $\boldsymbol{\theta}_G$ and $\boldsymbol{\theta}_G$ respectively. The cost function or objective of the generator is given by $J_G(\boldsymbol{\theta}_G, \boldsymbol{\theta}_D)$ and that of discriminator is denoted by $J_D(\boldsymbol{\theta}_G, \boldsymbol{\theta}_D)$. Notice that although $\boldsymbol{\theta}_G$ and $\boldsymbol{\theta}_D$ are involved in both objectives, the generator controls only $\boldsymbol{\theta}_D$ and discriminator controls only $\boldsymbol{\theta}_D$. Due to these constraints, rather than defining the final objective as an optimization problem, it is more prudent to describe it as a game whose solution is a local minimum in the parameter $(\boldsymbol{\theta}_G, \boldsymbol{\theta}_D)$ space. The required solution is a Nash Equilibrium, which constitutes a parameters tuple $(\boldsymbol{\theta}_G, \boldsymbol{\theta}_D)$ which is a local minimum of J_D with respect to $\boldsymbol{\theta}_D$ and a local minimum of J_G with respect to $\boldsymbol{\theta}_G$.

3.1.1 The Generator

The generator is represented by a simple differentiable function $G(z)$. It accepts a latent variable z sampled from a simple prior and yields an $\boldsymbol{x}_{fake} = G(z)$ as a sample drawn to mimic data from original data distribution $p_{model}(\boldsymbol{x})$. G is generally represented by a Deep Neural Network and the methods by which z is accepted to the network G is also highly customizable. For example, it is possible to feed a part of the input z as input to the first hidden layer, but the rest of z can be given as input to some intermediate layers. Hence the inputs to G can be provided in any layer of the Deep Neural Network and the methods of interaction of prior sample z with G are very flexible.

3.1.2 The Discriminator

The discriminator, like the generator, is also a differentiable function $D(\boldsymbol{x})$ which takes a candidate $\boldsymbol{x}$ as input and outputs a probability, that is a number between 0 and 1. The higher the output probability, more probable that the input $\boldsymbol{x}$ to the discriminator is real. $D(\boldsymbol{x})$ is also modelled as a neural network with parameters $\boldsymbol{\theta}_D$. Architecture of GAN is given in Figure 5.

3.2 Description of Two-player game in GAN

The two players in GAN are the discriminator and generator, each with parameters $\boldsymbol{\theta}_D$ and $\boldsymbol{\theta}_G$ of the respective neural networks. The two separate scenarios of the game can be described as below:

a) Scenario 1: We sample a real image $\boldsymbol{x}$ from training data and we feed it to $D(\boldsymbol{x})$. Since the image is real, $D(\boldsymbol{x})$ strives to learn parameters to output a probability as close as possible to 1.

b) Scenario 2: In this scenario, both the generator and discriminator participate in the game. We sample a z from the latent variable prior $p(z)$ and get an output from the generator network $G(z)$. Then the sample $G(z)$ is an input to D and we obtain $D(G(z))$. The generator tries to fool the discriminator and it's job is to learn neural network weights $\boldsymbol{\theta}_G$ such that $D(G(z))$ is as close as possible to 1. However, the discriminator tries to do it's best job to detect that $G(z)$ is a fake sample and tries to learn parameters $\boldsymbol{\theta}_D$ to push down $D(G(z))$ towards 0.

If both generator and discriminator networks have adequate capacity (number of hidden layers, width of layers etc.), the Nash equilibrium tuple $(\boldsymbol{\theta}_G, \boldsymbol{\theta}_D)$ are such that $D(\boldsymbol{x}) = \frac{1}{2}$ for all $\boldsymbol{x}$, whether $\boldsymbol{x}$ is a real sample from training data or if $\boldsymbol{x} = G(z)$ a fake sample from generator output.

3.3 Cost Function of GAN

The games designed for GAN have the same cost for the discriminator under various formulations of the game, and the formulation of this cost is motivated by two reasons. We want to maximize the probability of D assigning a training sample as real, and we want D to assign a sample generated by $G(z)$ as fake. Below is the cross entropy loss formulation for J_D that we want to minimize.

$$J_D\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log(1 - D(G(z)))$$

The loss/cost formulation of generators has several possible choices depending on the formulation of the game. However, the original paper on GAN by Goodfellow et al.[13] chooses Minimax zero-sum game formulation, in case of which the cost of the generator is simply negative of that of the discriminator. This choice was made in the original GAN paper because it is best amenable to theoretical analysis, which we will summarize later. In this case, the loss of entire GAN model with respect to both generator and discriminator parameters $\boldsymbol{\theta}_G$ and $\boldsymbol{\theta}_D$ turns out to be

$$\min_{\theta_G}\max_{\theta_D} V(G, D) = \min_{\theta_G}\max_{\theta_D}\left[\mathbb{E}_{x\sim p_{\text{data}}} \log D_{\theta_D}(\boldsymbol{x}) + \mathbb{E}_{z\sim p(z)} \log\left(1 - D_{\theta_D}\left(G_{\theta_G}(z)\right)\right)\right]$$

where $V(G, D)$ is the common loss term which applies both to generator and discriminator, since the losses are negatives of each other in minimax formulation. To elaborate, we simultaneously maximize $V(G, D)$ with respect to the discriminator, but minimize $V(G, D)$ with respect to generator.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 6: Algorithm for training a Generative Adversarial Network. Source: `https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`

3.4 Training Procedure of GAN

The training procedure of GAN involves alternating between gradient ascent update of Discriminator and descent update of Generator. Generally, we do k steps of gradient ascent update for the discriminator and one step of gradient descent update for generator. For each gradient update step, two sets of samples of size m each are sampled from the training data-set and from the latent variable prior $p(z)$. It was earlier considered that we need $k > 1$ steps of gradient ascent of discriminator for a single gradient descent step of generator. However, setting $k = 1$, that is simultaneous stochastic gradient ascent and descent for discriminator and generator is noted to not be a bad choice by Goodfellow[12].

3.5 Algorithm for training of GAN

Figure 6 gives the pseudocode for the algorithm for GAN training. The image of the algorithm has been taken from Goodfellow et al.[13].

3.6 Intuitive understanding of GAN training process

We will now choose the above minimax formulation of GAN cost and describe some key findings. Before diving into summarizing theoretical results on GAN, let us get some intuition on how we can recover the original p_{data} distribution by training GAN, given that the neural networks G and D are of sufficient capacity. We will consider Figure 7.

In Figure 7, part a), the blue dotted line represents the discriminative distribution i.e. $p_D(\boldsymbol{x})$. The black dotted line represents the p_{data}, the data generating distribution. The green line represents generative distribution $p_G(\boldsymbol{x})$. The lower horizontal line represents sampling from latent z and it is mapped to a horizontal line representing the domain of $\boldsymbol{x}$. The mapping $\boldsymbol{x} = G(z)$ imposes a non-uniform distribution $p_G(\boldsymbol{x})$ over outputs of the generator. Part b) represents the state after gradient ascent update of discriminator, where it is updated to recognize real training images from the ones from the generator. Hence we see a change in discriminative distribution $p_D(\boldsymbol{x})$. In part c) we have an update to the generator parameters $\boldsymbol{\theta}_G$ to generate better samples closed to original distribution. We can see that the green line (generative distribution $p_G(\boldsymbol{x})$) is closer to dotted line(data distribution $p_{model}(\boldsymbol{x})$). In part d) we can see that at the stage of convergence of parameters, the generator distribution $p_G(\boldsymbol{x})$ is almost equal to the model distribution $p_{model}(\boldsymbol{x})$. We can also see that the discriminative distribution is now equal to a constant 1/2, i.e. it has attained a stage where it will get fooled by samples from the generator and cannot distinguish it from real training data.

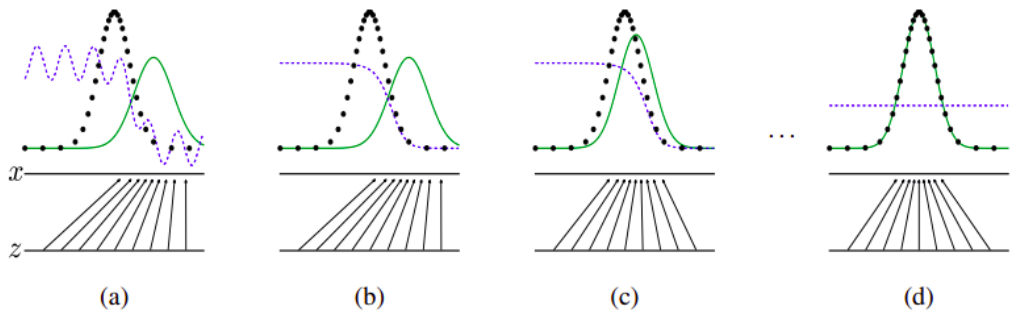

Figure 7: Pictorial description of GAN training. Source: `https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`

3.7 Summary of theoretical results on minimax formulation of GAN

The first theoretical result proven by Goodfellow et al.(2014)[13] regarding minimax formulation of GAN loss is regarding the optimal discriminator for a fixed generator. For a given fixed generator

parameters $\boldsymbol{\theta}_G$, the optimal discriminator is proven to follow,

$$D_G^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

If $p_G(\boldsymbol{x}) \approx p_{\text{data}}(\boldsymbol{x})$, $D_G^* \approx \frac{1}{2}$. Later, we will see that this configuration is this stage of convergence, and at this point, the discriminator is unable to tell the real training image from generated fake image (both get assigned probability of $\frac{1}{2}$). At this point of time, we can define a Virtual Training Criterion/Objective $C(G)$ as,

$$\begin{aligned}
C(G) &= \max_D V(G, D) \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log D_G^*(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z} \sim p_z} \left[\log \left(1 - D_G^*(G(\boldsymbol{z}))\right)\right] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log D_G^*(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{x} \sim p_G} \left[\log \left(1 - D_G^*(\boldsymbol{x})\right)\right] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}\left(\boldsymbol{x}\right)}{P_{\text{data}}\left(\boldsymbol{x}\right) + p_G(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_G} \left[\log \frac{p_G(\boldsymbol{x})}{p_{\text{data}}\left(\boldsymbol{x}\right) + p_G(\boldsymbol{x})}\right]
\end{aligned}$$

The next step is the important result that $C(G)$ attains a global minimum when $p_{data} = p_G$. Goodfellow et al.(2014)[13] prove that $C(G)$ can be expressed as,

$$\begin{aligned}
C(G) &= -\log(4) + D_{KL}\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_G}{2}\right) + D_{KL}\left(p_G \| \frac{p_{\text{data}} + p_G}{2}\right) \\
&= -\log(4) + JSD(p_{data} || p_G)
\end{aligned}$$

This means that the global minimum of $C(G)$ is obtained only when $p_{data} = p_G$, since the Jensen-Shannon divergence term, $JSD(.||.)$ is non-negative and is equal to zero when the two input distributions $p_{data} = p_G$, i.e. they are the same distributions. Next, the authors Goodfellow et al.(2014)[13] prove that, given the generator and discriminator networks have sufficient capacity (depth of the network and width of layers), and at each step of algorithm, the discriminator is allowed to configure itself to the state of being an optimal discriminator (as described above), GAN can converge to a stage where the model distribution is equal to the data distribution.

3.8 Benefits and Drawbacks of GANs

GANs are highly aligned and streamlined with the goal of producing realistic data and give state-of-the-art results in image synthesis of various types of data such as faces, chairs, animals or nature visuals. This is mainly because the GAN optimization problem where the generator progressively learns to produce realistic artificial data is tailored for the image synthesis problem. GANs are also excellent in semi-supervised learning tasks as shown by Salimans et. al.[26], which require us to train good classifiers with dataset where only a portion of data is labelled. The GAN optimization problem does not involve lower bounding via variational approximation like in VAEs. Because of this, there is no deterministic bias introduced in the GAN optimization problem, which leads to sharper feature learning. GANs require only one forward pass through the model to generate a sample, unlike earlier models like Boltzmann Machines, where generating a new sample required us to do unknown number of iterations through a Markov chain.

GANs are disadvantageous in that the training process is complicated. The optimization formulation is non-standard and we need find an Nash equilibrium point for the parameters of the model. The optimization tools for minimax problems are not advanced enough and and there is no good algorithm for finding equilibrium point for GAN. Hence, the training is unstable, compared to that of Variational Autoencoder. It is also difficult to use GAN for discrete data such as text or speech. The training process requires a lot of human supervision to prevent the model from getting stuck in local minima. It is also difficult to reverse engineer the posterior $p(\boldsymbol{z}|\boldsymbol{x})$, to get back the latent variable/noise that generated the image sample.

3.9 Architectural variants of GAN

The GAN architecture was first discussed in the paper by Goodfellow et al.[13] and there have been various model novelties over the years and can be classified into the below categories[6].

3.9.1 Fully Connected GANs

Fully Connected GANs are the first GAN architecture and for both generator and discriminator use fully connected neural network.This architecture is implemented for simple image datasets like MNIST, CIFAR-10 (natural images) and the Toronto Face Dataset (TFD) [6, 2].

3.9.2 Convolutional GANs

One constant criticism behind neural networks is the lack of algorithm based approach and use of black-box methods. For convolutional neural networks Zeiler et. al.[28] presented approximate purpose of each convolution filter in the CNN using deconvolutions and by the filtering the maximal activations. Similar work [6] performs gradient descent on the inputs to examine the ideal image that activates certain subsets of filters.These developments inspired different models of convolutional GANs mainly LAPGAN model and DCGAN model.

3.9.3 Laplacian pyramid of adversarial networks (LAPGAN)

The authors of LAPGAN [7] proposed a model combining the Laplacian pyramid representation with a conditional GAN model. This model iteratively upscales low resolution generated images which are more reliable in modelling. The Laplacian pyramid consists of a grouth truth image decomposed with the conditional convolutional GAN is used in training the generation process using multiple scales producing each layer from the previous layer. The structure is explained in the Figure 8. The G and D networks receive additional label information which improves the image quality. This model finds great applications in generating coarse-to-fine fashion images and has been explored for datasets like MNIST, CIFAR, LSUN.

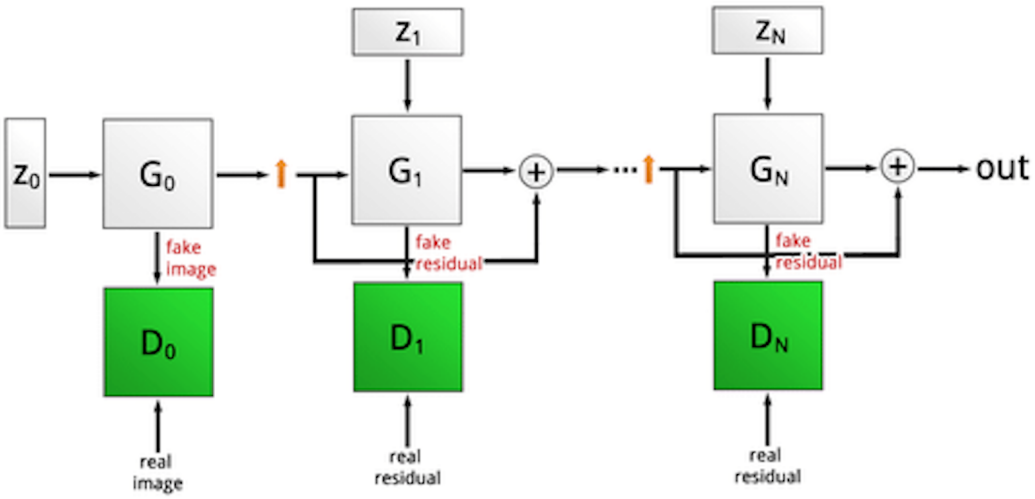

Figure 8: LAPGAN model

However, unlike supervised learning, the generator and discriminator could not be trained with the same level of capacity and representational power.

3.9.4 Deep Convolutional GAN (DCGAN)

DCGAN[23] is a family of network architectures which supports stable training of a pair of deep convolutional generator and discriminator networks. For training, DCGANs utilise convolutions of strided and fractionally-strided to implement the learning of the spatial down-sampling and up-sampling operators. These operators handle the change in sampling rates and locations, a key requirement in mapping from image space to possibly lower-dimensional latent space, and from image space to a discriminator.

3.9.5 Conditional GANs

Conditional GAN[21] architecture is an extension of Generative Adversarial nets to a conditional model where both the G and D are conditioned by feeding an additional input layer of some extra auxiliary information $\boldsymbol{y}$.

In the G, the prior input noise is $\sim p_z(\boldsymbol{z})$ and the $\boldsymbol{y}$ are combined in joint hidden representation. In D, $\boldsymbol{x}$ and $\boldsymbol{y}$ are inputs to a discriminative function.The objective function of a two-player minimax game would be,

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

For example,the pix2pix model [15] learns the mapping from the input image to the output image and constructs a loss function to train this. This is useful in computer vision for semantic segmentation, colorization of black and white images, and generating maps from aerial photos.

3.9.6 GANs with Inference Models

In the previous architectures, GANs lack an important mechanism of inference.The context of inference is the ability to determine the output of $\boldsymbol{z}$ with the given observation $\boldsymbol{x}$. The inference network was introduced in Adversarially Learned Inference (ALI)[10] and Bidirectional GANs[9] where the discriminators examine joint (data, latent) pairs, in which the generator consists of two networks encoder (inference network) and decoder.

The two joint distributions can be expressed as:

$$\textbf{Encoder joint} : q(\boldsymbol{x}, \mathbf{z}) = q(\boldsymbol{x})q(\mathbf{z} \mid \boldsymbol{x})$$

$$\textbf{Decoder joint} : p(\boldsymbol{x}, \mathbf{z}) = p(\mathbf{z})p(\boldsymbol{x} \mid \mathbf{z})$$

The above distributions are jointly trained to fool the discriminator. The discriminator receives pairs of ($\boldsymbol{x}$, $\boldsymbol{z}$) vectors, and has to determine which pair constitutes a genuine tuple consisting of real image sample and its encoding, or a fake image sample and the corresponding latent-space input to the generator. The output or reconstruction of encoding/decoding model is similar to input[6] The value function and the optimal discriminator are formalized in the following

$$\begin{aligned}\min_G \max_D V(D,G) &= \mathbb{E}_{q(\boldsymbol{x})}[\log(D(\boldsymbol{x}, G_z(\boldsymbol{x})))] + \mathbb{E}_{p(\mathbf{z})}[\log(1 - D(G_x(\mathbf{z}), \mathbf{z}))] \\ &= \iint q(\boldsymbol{x})q(\mathbf{z} \mid \boldsymbol{x}) \log(D(\boldsymbol{x}, \mathbf{z}))d\boldsymbol{x}d\mathbf{z} \\ &+ \iint p(\mathbf{z})p(\boldsymbol{x} \mid \mathbf{z}) \log(1 - D(\boldsymbol{x}, \mathbf{z}))d\boldsymbol{x}d\mathbf{z}\end{aligned}$$

$$D^*(\boldsymbol{x}, \mathbf{z}) = \frac{q(\boldsymbol{x}, \mathbf{z})}{q(\boldsymbol{x}, \mathbf{z}) + p(\boldsymbol{x}, \mathbf{z})}$$

However, the fidelity of reconstructed data samples synthesized using the above two GANs are poor.This can be improved by the next architecture.

3.9.7 Adversarial Autoencoders (AAE)

Adversarial Autoencoders (AAE) are probabilistic autoencoders which was first discussed by Makhzani et. at [20]. Adversarial training accomplishes feedforward and ancestral sampling with training applied between the latent space and a desired prior distribution on the latent space. This composition results in a reconstruction and the two mappings are trained to replicate the original image.The architecture is explained in the figure .

The result is a combined loss function of both the reconstruction error and the measure of the difference in distribution of the prior from that produced by candidate encoding network.

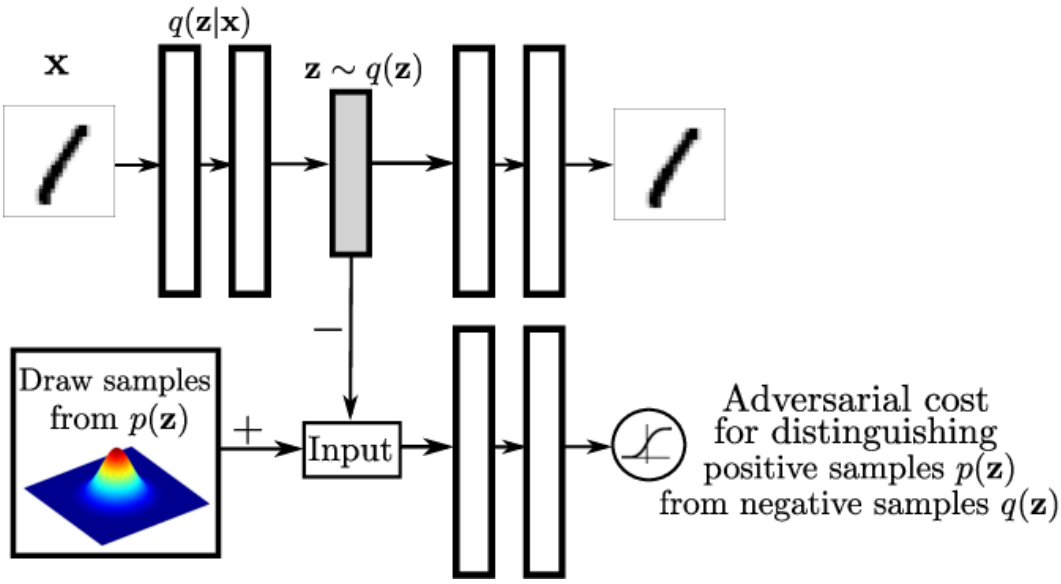

Figure 9: Architecture of Adversarial Autoencoders, Source: https://arxiv.org/pdf/1511.05644.pdf

3.10 Applications of GAN

GANs have made a significant contribution to various fields like Computer Vision, Natural Language Processing, Image processing etc. The most well-studied and vastly explored application is Image synthesis[6, 24] which generates new images from given attributes like class labels. In supervised learning, the networks learn the mapping from the input image to output image as well as from a loss function which helps in performing different loss formulations. The images can also be synthesized from reverse captioning or text descriptions resulting in several possible images that match the description. The model Generative Adversarial What-Where Network (GAWWN)[6, 25] conditions on image location. It consists of an interactive interface that provides tools to intuitively edit images and constructs large images incrementally with textual descriptions of parts and user-supplied bounding boxes.

In the field of image processing, there is a growing demand for high-resolution images to improve the accuracy of results during image analysis and pattern recognition. Super Resolution is a set of methods of upscaling video or images with the inference of photo-realistic details during up-sampling training. This is extended in the Super Resolution GAN (SRGAN)[6, 18] model by adding an adversarial loss component that adjusts the images to a manifold of natural images. The component is obtained from a larger loss function with the perceptual loss from a pre-trained classifier and regularization loss of spatially coherent images. These applications based on Computer Vision have been discussed in detail by Alqahtani et al.[2].

4 Experiments: Variational Autoencoders and DCGAN for Image Synthesis

We trained the Variational Autoencoder(implemented in PyTorch) and Deep Convolutional GAN (DCGAN)(implemented in TensorFlow 2.0) models for image generation task on MNIST and Fashion MNIST datasets(Github: `https://github.com/deepakgm/deep-generative-models`). We will briefly describe the model specifics of both VAE and GAN, and later describe the results in the next section. In the VAE model, the encoder and decoder neural networks consist of fully connected layers with ReLU activation. As described before, the encoder neural network has one input of the training data and has two outputs, corresponding to the auxiliary distribution's mean and variance respectively. These outputs are used for generating a sample from the q distribution via reparametrization trick. The generated sample is then used to make a reconstruction of the input and later, the ELBO loss is calculated and the gradients of the loss with respect to model parameters are backpropagated. We train for 10 epochs with a batch size of 128, and we use the Adam [16]

optimizer for the loss function. After every epoch, we generate an image through the decoder so that as the training progresses epoch after epoch, we get a demonstration of how the model learns to generate better and realistic images.

The DCGAN model, as described before, contains convolutional neural network layers for the generator and discriminator. The generator takes as input a random noise and through a series of up-sampling operations via 2d transpose convolutions, and we progressively get spatially enlarged features with decreased volume depth. The regularization in the generator is done via batch normalization [23] and leaky ReLU activation is used. The discriminator consists of downsampling stages via 2d convolutional layers with leaky ReLU activation, and the regularization is achieved via dropout. We trained the DCGAN model for 180 epochs with a batch size of 256. The model uses a cross entropy loss and both generator and discriminator use the Adam optimizer. After each epoch of training, we visualize sample images produced by the generator on a random input. We can see that the model learns to produce better images as training progresses. We now present the results of the VAE and DCGAN models on MNIST and Fashion MNIST datasets.

4.1 Results

Figure 10: Sample images generated by VAE model for MNIST dataset

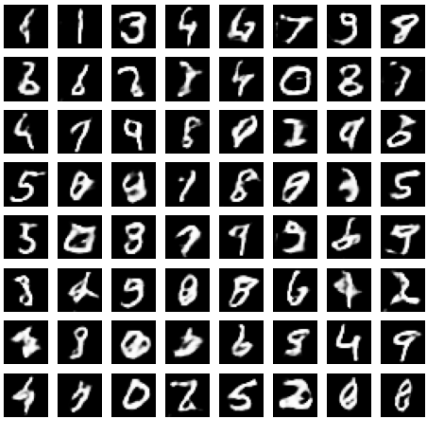

Figure 11: Sample images generated by DCGAN model for MNIST dataset

For the MNIST dataset containing 60000 images, once the training of both VAE and DCGAN are completed, the above images are generated. For the Fashion MNIST dataset containing 60000 images, we get the following images once training is completed for VAE and DCGAN.

Figure 12: Sample images generated by VAE model for Fashion MNIST dataset

Figure 13: Sample images generated by DCGAN model for Fashion MNIST dataset

4.2 Discussion

During the training procedure, we observe that the loss of GAN fluctuates wildly as the iterations progress, which is an indication that the training stage of GAN is somewhat unstable, compared to VAE. In both MNIST and Fashion MNIST datasets, we can observe that the images produced by VAE are blurry and somewhat less clearer than the ones produced by DCGAN. We can conjecture why we might be observing such results as follows. In case of the vanilla VAE that we trained, it is possible that the Gaussians modelled via fully connected neural networks are not expressive enough to capture the full complexity of data distribution.

Another reason why we might observe blurriness might be the maximum likelihood formulation of the problem itself. In case we are maximizing likelihood, we want $D_{KL}\left(p_{\text{data}} \| p_{\text{model}}\right)$ to be small. It is possible that the p_{model} trained by us, in addition to assigning high probability to points from data distribution, also assigns high probability to other points which happen to be blurry. Also, in case of GANs, the job of the generator is to capture all features that might let it fool the discriminator. Hence in the training process, the generator tends to gain ability to learn sharper features, such as boundaries and edges, which helps in producing better quality of images. Another reason why we can observe blurriness in VAE might be because of the definition of $\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{\phi} \sim \mathcal{N}\left(\mathbf{f}[\boldsymbol{h}, \boldsymbol{\phi}], \sigma^2\mathbf{I}\right)$. This isotropic Gaussian definition might not be a good assumption regarding data because effectively we are assuming the different pixels of the image as independent Gaussians, which might be restricting.

5 Conclusion

This project paper explains Deep Generative Models and presents a study of Variational Autoencoders and Generative Adversarial Networks. We first describe the theory and foundations behind both models, and how they leverage the power of neural networks as non-linear function approximators to tackle difficult problems in areas like vision and language modelling. We do a demonstration of performance of VAE and DCGAN on benchmarked datasets in task of image generation, namely MNIST and Fashion MNIST. We interpret these results and pose conjectures and speculations regarding why the results are the way they turned out to be. We observe that in general, VAEs tend to produce blurry image samples on both MNIST and Fashion MNIST.

*Note: In the below references section, the main papers referred are indicated in bold text.

References

[1] A Albert and Emmanuel Lesaffre. Multiple group logistic discrimination. In *Statistical Methods of Discrimination and Classification*, pages 209–224. Elsevier, 1986.

[2] Hamed Alqahtani, Manolya Kavakli-Thorne, and Gulshan Kumar. Applications of generative adversarial networks (gans): An updated review. *Archives of Computational Methods in Engineering*, pages 1–28, 2019.

[3] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer.

[4] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[5] Li Chunyuan. Less pain, more gain: A simple method for VAE training with less of that KL-vanishing agony.

[6] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. **Generative Adversarial Networks: An Overview**. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[7] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[8] Carl Doersch. **Tutorial on Variational Autoencoders**. *arXiv preprint arXiv:1606.05908*, 2016.

[9] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[10] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

[11] Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Celikyilmaz, and Lawrence Carin. Cyclical annealing schedule: A simple approach to mitigate kl vanishing. In *NAACL 2019*, February 2019.

[12] Ian Goodfellow. **NIPS 2016 Tutorial: Generative Adversarial Networks**. *arXiv preprint arXiv:1701.00160*, 2016.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. **Generative adversarial Nets**. In *Advancemakhzani2015adversarials in neural information processing systems*, pages 2672–2680, 2014.

[14] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Diederik P Kingma and Max Welling. **Auto-Encoding Variational Bayes**. *arXiv preprint arXiv:1312.6114*, 2013.

[18] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[19] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

[20] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[21] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[22] Kamalaldin Morad, William Y Svrcek, and Ian McKay. Probability density estimation using incomplete data. *ISA transactions*, 39(4):379–399, 2000.

[23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with regularized deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[24] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[25] Scott E Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. In *Advances in neural information processing systems*, pages 217–225, 2016.

[26] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[27] Prince Simon. Tutorial #5: Variational autoencoders. Available at: https://www.borealisai.com/en/blog/tutorial-5-variational-auto-encoders/.

[28] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.