Exploratory Project (CSE-291)
Report

Generative Modelling and GANs

Report submitted in fulfillment of the requirements for the Exploratory Project of

Second Year B.Tech.
in
Computer Science and Engineering

by

21075059	**Neerudu Nikhil**
21075069	**Ravi**
21075100	**Vriddhi Shri**

Under the guidance of

Dr. Pratik Chattopadhyay

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
Varanasi 221005, India

Semester IV - May 2023

Dedicated to

Our beloved Parents, Teachers

and IIT-BHU.

Declaration

We certify that

1. The work contained in this report is original and has been done by ourselves under the general supervision of our supervisor.

2. The work has not been submitted for any project.

3. Whenever we have used materials (data, theoretical analysis, results) from other sources, we have given due credit to them by citing them in the text of the thesis and giving their details in the references.

4. Whenever we have quoted written materials from other sources, we have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: May 18, 2023

Neerudu Nikhil, Ravi and Vriddhi Shri
B.Tech. IInd Year Students
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Department of Computer Science and Engineering
Indian Institute of Technology (BHU) Varanasi
Varanasi, INDIA 221005.

Certificate

*This is to certify that the work contained in this report entitled "**Generative Modelling and GANs**" being submitted by **Neerudu Nikhil (21075059), Ravi (21075069), Vriddhi Shri (21075100)** , carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bonafide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: May 18, 2023

Dr. Pratik Chattopadhyay
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Acknowledgments

We would like to extend our heartfelt respect and gratitude to our supervisor, Dr. Pratik Chattopadhyay, Assistant Professor in the Department of Computer Science and Engineering at the Indian Institute of Technology (BHU) Varanasi. Dr. Chattopadhyay's invaluable guidance, expertise, and enthusiastic involvement have been instrumental in the planning and development of this project. We are deeply indebted to his wisdom, vision, and persistent encouragement throughout our journey.

We also gratefully acknowledge his painstaking efforts in thoroughly going through and improving the manuscripts without which this work could not have been completed. We are also highly obliged to Prof. Pramod Kumar Jain, Director, Indian Institute of Technology (BHU) Varanasi, and Prof. Dr. Sanjay Kumar Singh, Head of Department, CSE for providing all the facilities, help and encouragement for carrying out this exploratory project work. We also thank Dr. Hari Prabhat Gupta, Assistant Prof., CSE, as a convener of this course and the other faculty members for their timely help and cooperation extended throughout the course of investigation.

We are also obliged to our parents for their moral support, love, encouragement and blessings to complete this task. Finally, we are indebted and grateful to the Almighty for helping us in this endeavor.

Place: IIT (BHU) Varanasi

Date: May 18, 2023

Neerudu Nikhil

Ravi and Vriddhi Shri

Abstract

Generative modelling has emerged as a prominent field within machine learning, enabling the creation of realistic and diverse samples from complex datasets. In this project, we begin by understanding the distinction between generative models and discriminative models and explore why generative models tend to be more challenging than discriminative models.

To compare different generative models, we utilize the MNIST dataset, which contains simpler features that are suitable for initial models like Naive Bayes, Gaussian Naive Bayes, and Mixture Models. We examine the outcomes and identify the strengths and weaknesses of these simpler models.

Furthermore, we delve into other advanced generative techniques, such as Variational Autoencoders (VAEs), and discuss the motivations behind their development from traditional Autoencoders. We analyze how VAEs improve upon the limitations of their predecessors and enhance the quality of generated samples.

Lastly, we explore elegant and efficient models known as Generative Adversarial Networks (GANs), specifically Deep Convolutional GANs (DCGANs) and Conditional GANs. We apply these models to datasets like CIFAR-10 and CelebA, and we investigate their ability to interpolate the latent space, resulting in smooth transitions between generated images.

Contents

List of Figures

Abbreviations

Abbreviations	Description
CGAN	Conditional Genrative Adversarial Network
CNN	Convolutional Nerual Network
DCGAN	Deep Convolutional Generative Adversarial Networks
EM	Expectation-Maximization
GAN	Generative Adversarial Networks
GDA	Gaussian Disciminant Analysis
GMM	Gaussian Mixture Models
GNB	Gaussian Naive Bayes
KL Divergence	Kullbach-Leibler Divergence
MLE	Maximum Likelihood Estimation
VAE	Variational Autoencoder

Chapter 1

Introduction

Machine learning involves two main models: Generative and Discriminative models. Understanding the significance of generative modeling requires a grasp of the distinction between these two approaches.

Generative Models generate new data similar to original data by learning from training data. It does not require the presence of labels in dataset (Unsupervised). This model learns to estimate the probability of the given observation X i.e. $P(X)$. For e.g. from different paintings of M.F. Hussain it can generate new painting.

Discriminative models classify among different data. It requires the presence of labels for in training data (Supervised). This model estimate label Y for a given variable X i.e. $P(Y|X)$. For e.g. It can differentiate which painting is made by M.F. Hussain.

The crucial point is that even if we are to construct a flawless discriminative model for identifying Hussain paintings, this model would still lack the ability to create a painting resembling his style. Its sole function would be to provide probabilities for existing images based on its training. To generate new images that bear resemblance to the original training dataset, we must instead employ a generative model. In essence, generative modeling holds the key to unlocking a more sophisticated form of artificial intelligence that surpasses the capabilities of a discriminative model.

Chapter 2

Generative Modelling

2.1 What is Generative Model ?

Generative modelling aims to model the underlying probability distribution of a given dataset to generate new samples that resemble the original data. It can be broadly classified into two categories:

1. Explicit generative models: These models explicitly learn the probability distribution of the training data and generate new samples by directly sampling from this distribution. Examples of explicit generative models include Gaussian Mixture Models (GMMs) and Variational Autoencoders (VAEs).

2. Implicit generative models: These models do not explicitly model the probability distribution but instead learn to approximate it through a learning process. They learn to generate new samples by learning the mapping from a noise distribution (e.g., Gaussian noise) to the data distribution. Examples of implicit generative models include Generative Adversarial Networks (GANs).

Generative models offer powerful capabilities for data generation and unsupervised learning, but it also comes with a lot of challenges, such as it requires large and diverse datasets to be trained upon. So it becomes computationally expensive and

harder to train. Also, there is no medium to ensure that the generated data is realistic and high-quality. In other words, its performance cannot be easily evaluated. Also there may have chances of replication of the original data without introducing new variations. Ongoing research aims to address these limitations and advance the field of generative modelling.

2.2 Need of Generative Modelling

Generative models help in various applications such as :

- Data Augmentation: It can be used to augment existing datasets by generating additional samples. This is particularly useful when the original dataset is limited or imbalanced, allowing the model to learn from a more diverse range of examples.

- Imputation of Missing Data: It can fill in missing or incomplete data points by learning the underlying patterns and generating plausible values for the missing entries.

- Synthetic Data Generation: It enables the creation of synthetic data that can be used for various purposes, such as testing algorithms, simulating scenarios, or preserving privacy in data sharing. Synthetic data can help mitigate privacy concerns, as it does not contain personally identifiable information.

- Creative Applications: It has been used in creative domains such as art, music, and literature to generate novel and imaginative outputs. They can assist artists, musicians, and writers in generating new ideas or exploring unconventional possibilities.

- Anomaly Detection: By learning the normal distribution of a dataset, generative models can identify anomalies or outliers that do not conform to the learned

patterns. This can be valuable in detecting fraudulent transactions, identifying unusual behaviour in a system, or flagging anomalies in medical diagnoses.

- Data Compression: Generative models can learn compact representations of data by capturing the essential features and underlying structure. These representations can be used for efficient storage, transmission, or compression of data.

- Image and Text Generation: Generative models, such as generative adversarial networks (GANs) and variational autoencoders (VAEs), have shown impressive capabilities in generating realistic images and text.

- Domain Adaptation and Style Transfer: It can learn the style or distribution of one dataset and apply it to another. This allows for domain adaptation, where a model trained on a source domain can generate samples in a target domain while preserving certain desired characteristics.

Additionally, this model has the capability to simulate environments, allowing users to explore and evaluate various potential futures without the need for direct real-world experiences. This functionality proves especially valuable in the field of Reinforcement Learning, enabling informed decision-making by virtually navigating different scenarios. By leveraging these simulations, one can effectively prevent undesirable conditions or unforeseen factors from contributing to incidents. Moreover, these simulations find practical application in tasks such as image denoising, video generation, and image restoration, further showcasing their versatility and usefulness. Overall, generative modelling provides a powerful framework for understanding and manipulating complex data distributions, offering a wide range of practical applications across various domains.

Chapter 3

Types of Generative Modelling

3.1 Multinomial Distribution

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Figure 3.1: Play Tennis Dataset

The multinomial distribution is a probability distribution that describes the probabilities of observing different discrete outcomes or categories. It is often used to model data where each observation can belong to one of several categories, and the goal is to estimate the underlying probability distribution of those categories. It is a

generalization of the binomial distribution. To estimate the parameters of the multinomial distribution from data, maximum likelihood estimation (MLE) is often used. MLE involves finding the values of the parameters that maximize the likelihood of observed data under the assumed multinomial distribution.

Let us understand it by taking a dataset of Play Tennis. This dataset helps in determining whether a particular weather condition is suitable ("Yes") or not Suitable ("No") for Playing Tennis.

- We consider that the data has been generated from distribution p_{data} (unknown).
- The aim of the model is to use these observations to produce the p_{data}.
- In Figure 3.1, total number of observations are 14.
- No. of features in outlook, temperature, humidity and wind is 3, 3, 2 and 2 respectively. Hence total $3 * 3 * 2 * 2 = 36$ possible combinations of features i.e. 36 sample points (36 different points in sample space).
- If θ_i is the parameter which represents the probability of the i^{th} sample point, then we have 36 different parameters each connected with one of the features (36 minus 1, as the probability associated with the last feature, will be constrained to be 1 – the sum of probabilities of other features).

Using Maximum Likelihood Estimation (MLE) to estimate θ_i we have :

$$\theta_i = \frac{n_i + 1}{N + 1}$$

where, N is the total no. of observations, here 14. n_i is no. of occurrence of i^{th} observation in the given dataset. One is added as a smoothing factor to assign a non-zero probability to the observation which is not present in the original dataset. Hence every parameter gets a non-zero probability according to which it can be sampled from the sample space. However, this still fails to be a satisfactory Generative Model.

3.1.1 Shortcomings:

- The multinomial distribution becomes increasingly challenging to model as the number of categories (parameters) increases.[1] The number of parameters required to properly specify the distribution grows exponentially with the number of categories, leading to high dimensionality. This can make estimation and inference computationally expensive and data-hungry.
- The probability of estimating a point which is not in the original dataset is always a constant. If we try to use such a model to generate car image, it will assign equal weight to a random colorful pixels as to a replica of this car image that differ very minutely from a original image.

We want ideally to have our generative model in a way that it up weights points of sample space that it believes are most likely, due to some inherent structure learned from the data, rather than simply putting all probabilistic weights on data that are available in the training dataset. To achieve this, we need to choose a different parametric model.

3.2 Naive Bayes

Naive Bayes is a simple probabilistic machine learning algorithm based on Bayes' theorem. Unlike Multinomial distribution, this parametric model makes use of a simple assumption that drastically reduces the number of parameters we need to estimate.[2] Naive Bayes' Assumptions :

1. Each feature of a dataset are independent of each other. This means that the presence or absence of a particular feature does not affect the presence or absence of any other feature. Mathematically, for feature x_i and x_j :

$$P(x_i|x_j) = P(x_i)$$

2. Features are assumed to be binary.

To apply these assumptions, we first make use of the chain rule of probability to write the density function as a product of conditional probabilities.

$$p(x) = p(x_1, ..., x_k)$$

$$p(x) = p(x_2, ..., x_k|x_1)p(x_1)$$

$$p(x) = p(x_3, ..., x_k|x_1, x_2)p(x_2|x_1)p(x_1)$$

$$p(x) = \prod_{k=1}^{K} p(x_k|x_1, ..., x_{k-1})$$

where K is the total number of features. We now apply the Naive Bayes assumption to simplify the last line:

$$p(x) = \prod_{k=1}^{k} p(x_k)$$

This is the Naive Bayes model. The problem is reduced to estimate the parameters $\theta_{k,l} = p(x_k = l)$for each feature separately and then multiplying these to find the joint probability for any possible combination.

Now the Maximum Likelihood Estimation (MLE) is given as :

$$\hat{\theta}_{k,l} = \frac{n_{k,l}}{N}$$

where N is the total number of observations in the dataset and $n_{k,l}$ is number of times x_k is l in the training dataset.

Moreover, a model based on Naive Bayes can handle missing values by ignoring the instance during probability estimation. Also, the model trained using this algorithm is robust to isolated noise points because such points are averaged out when estimating conditional probabilities from data. But this algorithm only works with discrete features making us to think of a different advanced algorithm that can deal with

continuous features also.

3.3 Gaussian Naive Bayes

Gaussian naive Bayes is a case of naive Bayes method with an assumption of having a Gaussian distribution with mean μ_k and variance σ_k^2 on attribute values given the class label l. It is the extension for the Naive Bayes, where the parameters now are no longer binary. This algorithm assumes that the probability of each attribute belonging to a given class l value does not depend on other attribute values.[3]. The covariance matrix of each class is assumed to be the same, and the reason is that the decision boundary would be linear.

$$\hat{\mu_k} = \frac{\sum_{i=1}^{m} I_E \left\{ y^{(i)} = p \right\} * x^{(i)}}{\sum_{i=1}^{m} I_E \left\{ y^{(i)} = p \right\}}$$

$$\sigma_k^2 = \frac{\sum_{i=1}^{m} I_E \left\{ y^i = 1 \right\} (x_i - \hat{\mu_k})^2}{n}$$

$$\Sigma = \frac{\sum_{i=1}^{m} (x_i - \mu_{\hat{C}_k})(x_i - \mu_{\hat{C}_k})^T}{n}$$

3.3.1 Insights:

- The decision boundary function we obtain would be a linear function, because of our initial assumption that co-variances are the same for both classes.
- Class of testing data x depends on the density that is given by the distributions of the two classes.
- If the co-variance matrices are constant matrices, then contours will be circular, and the decision line will be perpendicular bisector joining centres.
- If the variances are different, then it results for different characteristics of contours. So, the decision boundary is no longer a linear function but rather a

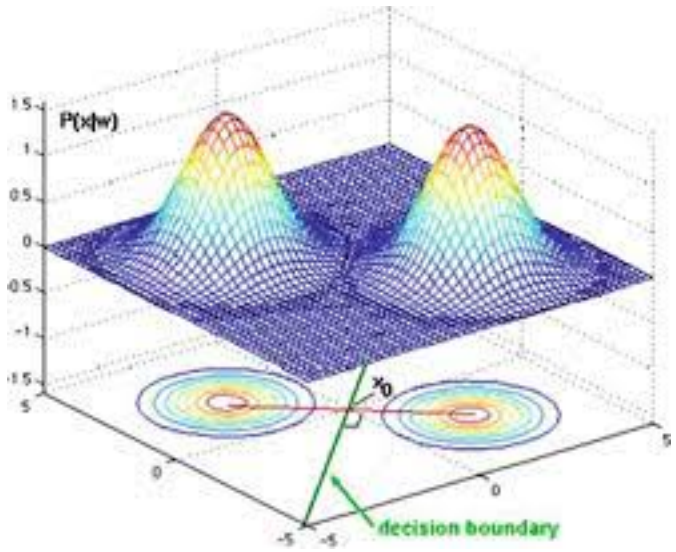

Figure 3.2: Decision boundary considering same constant co-variance matrix for both classes

quadratic function.

3.3.2 Shortcomings:

- Assumption that each feature in the dataset is independent of each other might not hold in many real-world scenarios, where features often exhibit dependencies. For example, if we want to generate a white cat, then neighbouring features of color are highly correlated. So they can't be independent of each other.

- It is not well-suited for handling categorical features or discrete variables as it assumes that the features follow a Gaussian distribution.

- Data scarcity and zero probabilities: When training GNB on datasets with limited samples or rare combinations of feature values, it can lead to zero probabilities for certain classes or feature values. This issue is known as the "zero frequency" problem, and it can cause the model to fail during classification, as it cannot handle unseen feature combinations.

3.4 Gaussian Discriminant Analysis:

Gaussian Bayes results for a poor outcome due to the feature independence assumption. So comes into the picture to overcome this limitation is a classification algorithm that makes assumptions about the probability distribution of the features in each class know to be Gaussian discriminant analysis *(GDA)*. GDA assumes that the features in each class follow a multivariate normal (Gaussian) distribution. Covariance matrix Σ of each class is assumed to be same, and the reason being that the decision boundary would be linear.

$$\phi = \frac{\sum_{i=1}^{m} y^{(i)}}{m} = \frac{\sum_{i=1}^{m} I_E(y^{(i)} = k)}{m}$$

$$\mu_k = \frac{\sum_{i=1}^{m} I_E(y^{(i)} = k) * x^{(i)}}{\sum_{i=1}^{m} I_E y^{(i)} = k}$$

$$\Sigma_k = \frac{\sum_{i=1}^{m} (X^{(i)} - \mu_y^{(i)})(X^{(i)} - \mu_y^{(i)})^T}{m}$$

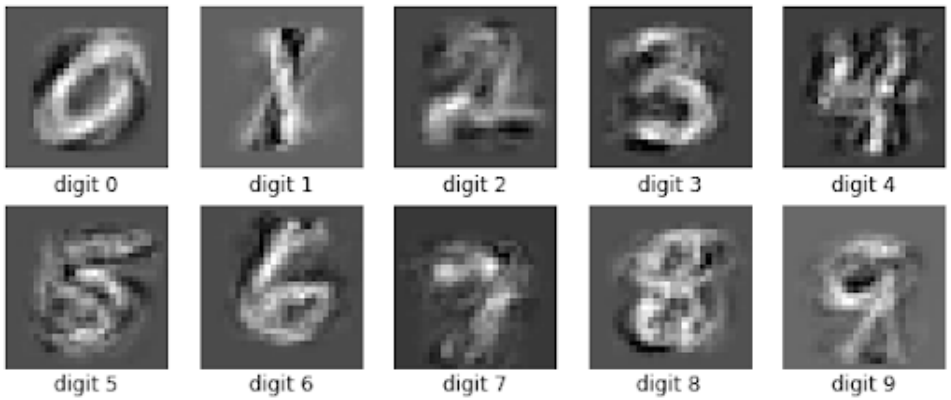

Figure 3.3: Generation of MNIST digits by Gaussian Discriminant Analysis

3.4.1 Insights:

- If $\Sigma_{ij} = 0$ for $i \neq j$ then matrix Σ is a diagonal matrix, it implicates that the feature i is independent of every feature j.

- Comparing Gaussian discriminant analysis to logistic regression, we can end up to a sigmoid function for calculating $P(Y = 1/X)$.

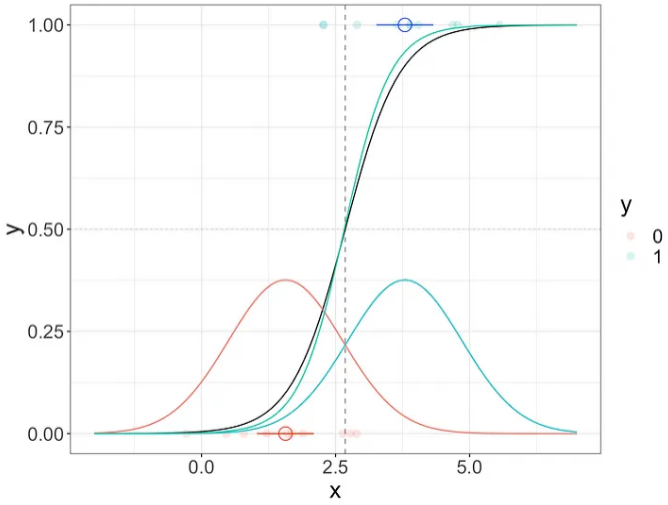

Figure 3.4: GDA ending upto Sigmoid

- GDA makes stronger modelling assumptions, and is more data efficient (i.e., requires less training data to learn "well") when the modelling assumptions are correct or at least approximately correct.

- Assumptions of logistic regression could be achieved by considering the assumptions of GDA.

3.4.2 Shortcomings:

- The assumption of equal covariance matrices in GDA, where variability and correlation between variables are the same for all classes, may not apply in practical analysis. Violations of this assumption occur when different classes exhibit varying covariance structures.

- GDA can be computationally expensive for high-dimensional datasets due to the need to estimate the covariance matrix. Estimating the covariance matrix

accurately requires a large number of parameters, making the computation and memory requirements substantial.

- GDA faces difficulties in accurately classifying infrequent classes within unbalanced datasets. The model tends to prioritize learning the majority class, resulting in lower performance for the minority or infrequent classes. This imbalance in class representation can hinder GDA's ability to effectively classify rare classes.

3.5 Gaussian Mixture Models:

A Gaussian Mixture is a function that is comprised of several Gaussians, each identified by k $\in \{1,\ldots, K\}$, where K is the number of clusters of our dataset. All the above mentioned algorithms assume one single class for each digit, but there could be several ways of writing the digit. Each cluster k is a Multivariate Gaussian distribution comprised of the following parameters:

- A mean μ that defines its centre.
- A covariance Σ that defines its width.
- A mixing probability π that defines how big or small the Gaussian function will be.

Our clustering task amounts to inferring the latent component z_i responsible for each x_i. For the moment, we will ignore the fact that we do not know the parameters of the GMM and imagine how we would carry out the clustering task given (π , μ_{1_k} , Σ_{1_k}).

The maximum likelihood principle would choose estimates of (π , μ_{1_k} , Σ_{1_k}) that maximize this expression. For k>1, the log likelihood no longer simplifies and

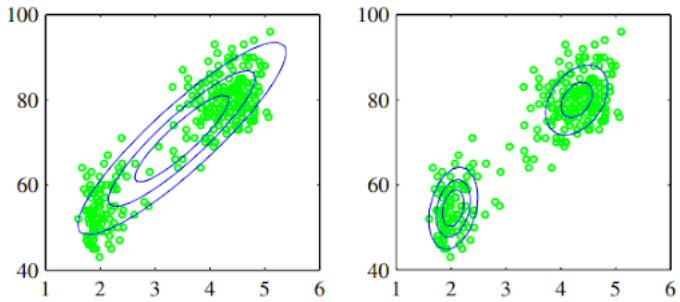

Figure 3.5: Clustering of data points

no longer yields closed-form solutions. Hence we choose an iterative optimization technique that operates locally, known as the Expectation-Maximization algorithm that is used to find the latent variable, which is basic for many of the unsupervised clustering algorithms. The EM algorithm consists of two main steps: The E-step and the M-step.

- E-step involves computing the posterior probabilities of the missing variables or latent variables, conditioned on the observed data and the current parameter estimates. The responsibilities that belong to each Gaussian component using current GMM parameters, are evaluated.

$$\gamma_k(x) = \frac{\pi_k \times N(x/\mu_j, \Sigma_j)}{\Sigma_{j=1}^{K}(\pi_k \times N(x/\mu_j, \Sigma_j))}$$

- In the M-step, the algorithm updates the model parameters (computed in the E-step) to maximize the expected log-likelihood, subject to any constraints imposed by the model.

$$\pi_j = \frac{\Sigma_{n=1}^{N} Y_j(X_n)}{N}$$

$$\mu_j = \frac{\Sigma_{n=1}^{N} Y_j(X_n \times X_n)}{\Sigma_{n=1}^{N} Y_j(X_n)}$$

$$\Sigma_j = \frac{\Sigma_{n=1}^{N}(X_n - \mu_n)(X_n - \mu_m)^T}{\Sigma_{n=1}^{N} Y_j(X_n)}$$

- This iterates between these two steps until convergence, that is until the change

in the log-likelihood between successive iterations falls below a certain threshold.

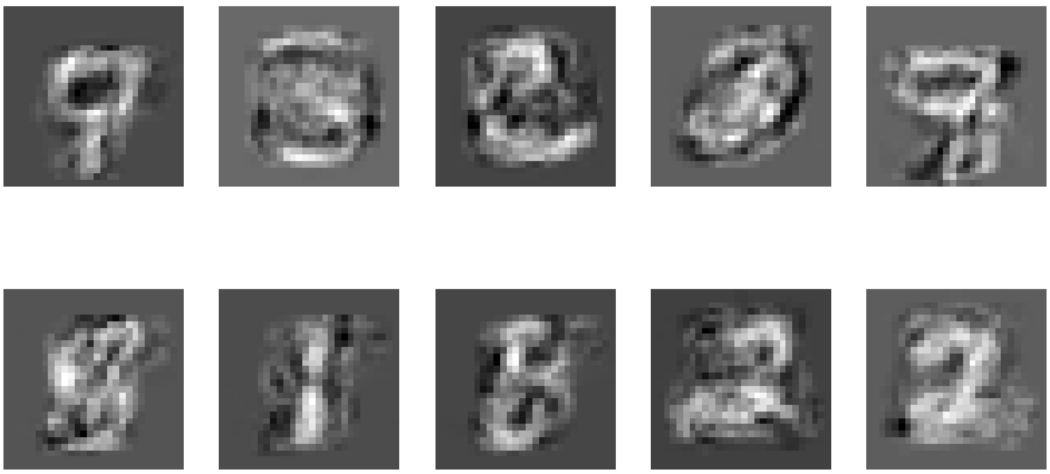

Figure 3.6: Generation of MNIST digits by Gaussian Mixture Models

3.6 Autoencoders

An autoencoder is a special type of feed forward neural network which

- encodes its input x_i into a hidden representation h .

- decodes the input again from this hidden representation.

The model will be trained to minimize a certain loss function which will ensure $\hat{x_i}$ is close to x_i. Autoencoders may be trained end-to-end or gradually layer by layer. In the latter case, they are "stacked" together, which leads to a deeper encoder.[4]

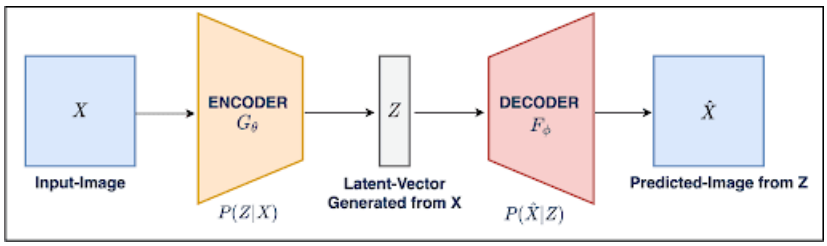

Figure 3.7: Schematic Diagram of an Autoencoder

$$h = g(W \times x_i + b)$$

$$\hat{x_i} = f(W^* \times h + c)$$

- If $\dim(h_i) < \dim(\ x_i\ )$, and if it can reconstruct (y_i), from h such h is known to be loss free encoding of (x_i), the encoder is called under complete autoencoder.
- If $\dim(h_i) > \dim(\ x_i\ )$, such encoder learns the trivial encoding and it is known as an over-complete autoencoder.

We choose our loss function to minimise difference between x and $\hat{x}$. For real inputs we consider the mean square error (MSE) while for the binary inputs it is considered to be the binary cross entropy (BCE places a heavy penalty if the choice of pixel is extremely wrong, which leads for a less vibrant images.)

$$L(\theta) = \min\left\{\frac{\sum_{i=1}^{m}(\hat{x_i} - x_i)^T(\hat{x_i} - x_i)}{m}\right\}$$

$$L(\theta) = \min\left\{-\sum_{j=1}^{m} x_{ij} log(\hat{x_{ij}}) + (1 - x_{ij}) log(1 - \hat{x}_{ij})\right\}$$

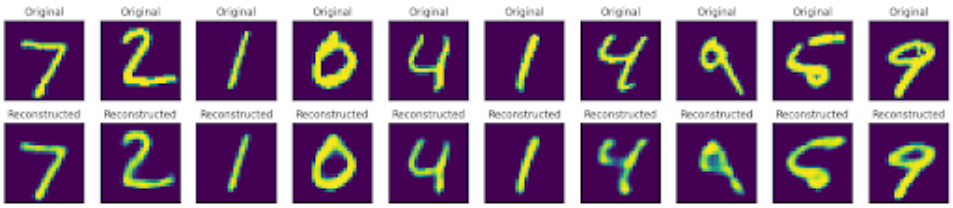

Figure 3.8: Generation of MNIST digits by Autoencoder

Autoencoders can be used for image denoising. For, this purpose, some random noise is added to the original image to form a partially corrupted image x_i', which is passed through the autoencoder to generate image $\hat{x_i}$.

3.6.1 Regularisation of Autoencoders:

- Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting

or underfitting. For regularisation, the simplest solution is to add an L2 regularisation term to the function.

$$L(\theta) = \min \left\{ \frac{\sum_{i=1}^{m} \sum_{i=1}^{n} (\hat{x}_i - x_i)^2}{m} \right\} + \lambda \|\theta\|^2$$

- Parameters of the encoder and decoder are reduced by considering $W^* = W_T$.

3.6.2 Shortcomings:

- Autoencoders have a tendency to overfit by memorizing the input data, resulting in limited generalization capabilities. Instead of learning meaningful features, they simply store and reproduce the training samples, potentially leading to poor performance on unseen data. Regularization techniques like dropout and early stopping can help mitigate this issue and encourage feature learning.[5]

Figure 3.9: Discontinuous Latent space of Autoencoder spread over large area

- The latent space of an autoencoder can exhibit a discontinuous nature, where slight variations in the input may lead to abrupt changes in the corresponding encoded representations. This discontinuity can sometimes limit the smooth interpolation of features within the latent space.

- If input data is complex, then training the autoencoder is computationally expensive.

3.7 Variational Autoencoder

Some works have made tremendous progress in training neural networks as powerful function approximators through backpropagation [6]. These advances have given rise to promising frameworks which can use back propagation-based function approximators to build generative models.[7] Variational autoencoder describes an observation in latent space in a probabilistic manner, which brings continuity to the latent space, which is the major limitation of an autoencoder.

When decoding from the latent state, we'll randomly sample from each latent state distribution to generate a vector as input for our decoder model. Actual encoding is calculated by sampling from Gaussian with that Mean and Variance it varies somewhat during each pass.[8] Encoder of VAE learns the points nearby to the actual output mean and variance of a class also as that particular class with little variation. So decoder will map the near by points to that class and returns an image closer to that class with little variation. This overcomes the latent space discontinuity limitation that traditional autoencoders posses.

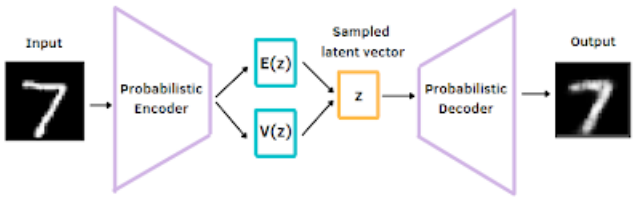

Figure 3.10: Block diagram representing Variational Autoencoder

3.7.1 KL Divergence:

KL Divergence loss of the encoder is minimised with the standard normal distribution ensuring it is concentrated at the origin. The loss function of VAEs is composed of two parts: one is just the log-likelihood of the reconstruction, while the second one is a term aimed to enforce a known prior distribution $P(z)$ of the latent space - typically a spherical normal distribution. Technically, this is achieved by minimizing the Kullbach Leibler distance between $Q(z|X)$ and the prior distribution $P(z)$; as a side effect, this will also improve the similarity of the aggregate inference distribution $Q(z) = E_X Q(z|Z)$ with the desired prior, that is our final objective.[9]
The loss function of VAE expressed as

$$L(\theta, \phi) = -E_{z \sim q_\theta(z|x)} \left[log_{p_\phi}(x|z) \right] + KL(q_\theta(z|x)||p(z))$$

Here θ and ϕ represent the parameters of our decoder and encoder respectively.$(\|x - \hat{x}\|)^2$ is minimized through the first term of the loss function, while the second function ensures that our encoding distribution is close to the standard normal distribution. This overcomes the other limitation of traditional autoencoder, where the encodings are distributed asymmetrically about the origin and spread over a large area.

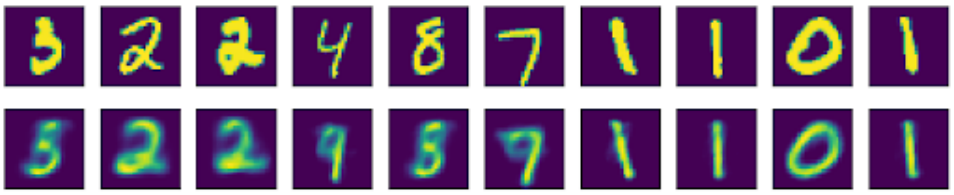

Figure 3.11: Generation of MNIST digits using Variational Autoencoder

3.7.2 Reparameterization trick:

The reparameterization trick plays a pivotal role in Variational Autoencoders (VAEs), facilitating effective training and sample generation from the latent space. By decou-

pling the stochasticity from the model's parameters, this technique allows for efficient gradient-based optimization during training while preserving the ability to generate diverse and meaningful samples. This elegant approach ensures both the smooth convergence of the VAE model and the production of high-quality outputs from the latent space. [10]

$$z \sim N(\mu, \sigma^2) \implies z = \mu + \sigma.\epsilon, \epsilon \sim N(0, 1)$$

3.7.3 KL loss annealing:

The employed loss function $L = RL + KL$, consisting of the reconstruction loss (RL) and the KL divergence loss (KL), posed a challenge during the initial stages of training. The substantial magnitude of the KL loss term compared to the RL loss led to an imbalance in the feedback signal during backpropagation.[11] As a consequence, the network parameters tended to converge to local minima where the KL loss was low, but the RL loss remained high. This undesirable outcome highlighted the need to address this issue and find a balanced approach that promotes the optimization of both components to ensure improved overall performance. So a modified loss equation is used: $L = r.RL + KL$.

Initially, we set r=0, then it is linearly increased to 1 over a regular period of training. This helps in avoiding local optima and provides with better outputs.

Chapter 4

Generative Adversarial Networks (GANs)

4.1 Generative Adversarial Network (GAN)

4.1.1 Introduction

Generative Adversarial Networks (GANs) have emerged as a powerful class of models in the field of deep learning. They have revolutionized the field of generative modelling by enabling the synthesis of realistic and high-quality samples from complex and high-dimensional data distributions. GANs were introduced by [12], and since then, they have gained significant attention and achieved remarkable results in various domains, including image synthesis, text generation, and video synthesis.

4.1.2 GAN Architecture

The fundamental idea behind GANs is to train two neural networks simultaneously: a generator network and a discriminator network. The generator network generates synthetic samples from random noise, while the discriminator network learns to dis-

tinguish between real and fake samples. The two networks engage in a two-player minimax game, where the generator aims to generate samples that fool the discriminator, while the discriminator aims to accurately classify real and fake samples.

4.1.3 Adversarial Training

GANs employ an adversarial training 'process, where the generator and discriminator are trained simultaneously in a competitive manner. The training procedure can be summarized as follows:

- The generator takes random noise as input and generates synthetic samples.
- The discriminator takes both real and generated samples as input and predicts whether each sample is real or fake.
- The discriminator's predictions are compared with the ground truth labels (real or fake) to compute a discriminator loss.
- The generator's objective is to generate samples that fool the discriminator, so its loss is based on the discriminator's output for generated samples.
- The gradients from the discriminator's loss are backpropagated to update the discriminator's weights, while the gradients from the generator's loss are backpropagated to update the generator's weights.
- This adversarial game between the generator and discriminator continues iteratively until both networks reach a stable equilibrium, where the generator produces samples that are indistinguishable from real samples, and the discriminator is unable to accurately classify them.

4.1.4 Mathematics of GANs

Mathematically, let's consider the training data generated from an unknown distribution called p_{data}. We have a generator that samples from a known distribution called $p_z(z)$ and transforms it into the image space using the function G. If $G(z)$ follows a probability distribution function called p_g, our goal is to approximate p_{data} with p_g. The discriminator, represented by a neural network, examines images from p_{data} and p_g and categorizes them as either real ($D(X) \approx 0$) or fake ($D(X) \approx 1$).

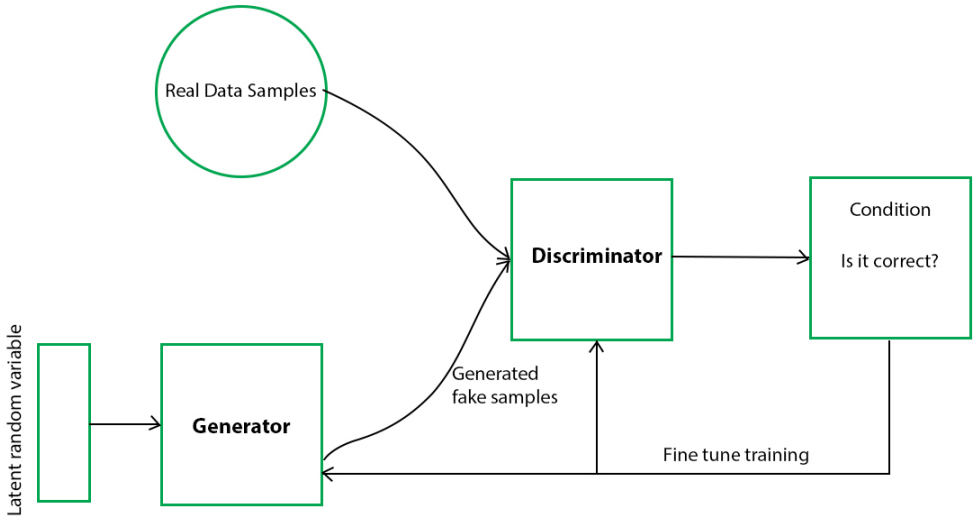

Figure 4.1: Schematic Diagram of a Generative Adversarial Network

Source: Generative Adversarial Network Architecture and its Components

The discriminator (D) and the generator (G) play the min-max game. The objective of the discriminator is to minimize the value of $V(D, G)$, while the generator aims to maximize it.

(Source: *Generative Adversarial Nets*, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua

Bengio., p. 3)

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

This equation can be understood intuitively. When the image is real ($x \sim p_{data}(x)$), the discriminator seeks to classify it as real ($D(x) \approx 1$), maximizing the first term. When the image is fake ($G(z) \sim p_g$) or equivalently ($z \sim p_z(z)$), the discriminator wants to classify it as fake ($D(G(z)) \approx 0$), maximizing the second term. Conversely, the generator aims to minimize the second term by deceiving the discriminator into predicting the fake image as real ($D(G(z)) \approx 1$).

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 4.2: Algorithm for the training of GANs

(**Source:** Generative Adversarial Nets, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio., p. 4) [12]

Mathematically, for any given generator G, the optimal value of the discriminator D_G^* can be obtained by minimizing $V(D, G)$ with respect to D. By solving the equation $\frac{\delta V}{\delta D} = 0$ and verifying that $\frac{\delta^2 V}{\delta^2 D} < 0$ (indicating a maximum) at D_G^*, the optimal discriminator is found to be:

(Source: *Generative Adversarial Nets*, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio., p. 5-6)

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Similarly, the optimal value of the generator can be found by minimizing $V(D_G^*, G)$ with respect to G. In other words, we aim to find $G^* = \arg\min_G [V(D_G^*, G)]$, which reduces to $G^* = \arg\min_G [-\log 4 + 2 * JSD(p_{data} \parallel p_g)]$, where JSD represents the Jensen-Shannon divergence between p_{data} and p_g.

Therefore, G^* is considered optimal when the JSD between p_{data} and p_g is zero, indicating that $p_g(x)$ is the same as $p_{data}(x)$ for all x. This is our objective with the model. In this scenario, $V = -\log 4$.

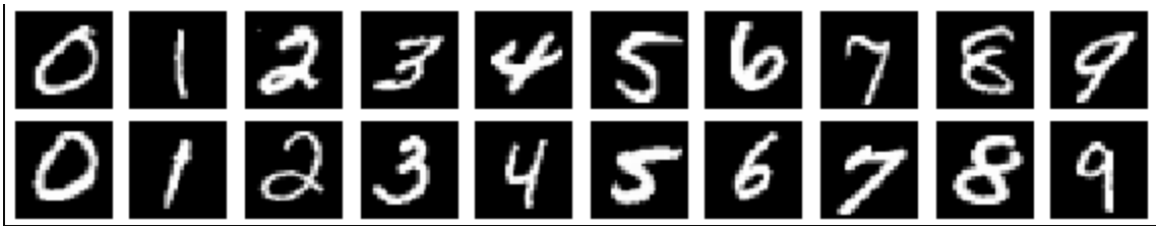

Figure 4.3: MNIST like digits generated using Generative Adversarial Networks

Theoretically, this min-max game ensures that in equilibrium, the probability distribution function of the generated samples becomes identical to the probability distribution function of the real dataset. Consequently, the generator starts producing highly realistic images that the discriminator cannot distinguish from real images. The discriminator outputs a probability of $\frac{p_{data}}{p_g + p_{data}} = \frac{1}{2}$ for any point, classifying it as real with a 50% chance, regardless of its authenticity.

4.1.5 Shortcomings

- **Vanishing Gradients:** If θ represents the weights of the generator, then $\frac{dV}{d\theta} \approx 0$ when $D(G(z)) \approx 0$, which generally happens in initial stages of training. This results in slow training progress. To overcome this problem, a practical solution is to modify the objective function by maximizing $\log(D(G(z)))$ instead of minimizing $\log(1 - D(G(z)))$. Although the objective remains the same, this modification provides improved gradients for backpropagation, particularly in the early stages of training.

- **Mode Collapse:** Mode collapse is a common shortcoming in Generative Adversarial Networks (GANs) where the generator fails to capture the full diversity of the target data distribution and instead produces limited variations or even repetitive samples. It occurs when the generator focuses on generating only a subset of the possible outputs, ignoring the richness and diversity present in the training data. When mode collapse happens, the generator becomes overly specialized in producing a few dominant modes or patterns that may be easier to learn. As a result, it fails to explore and generate samples that cover the entire data distribution. This leads to a loss of diversity and variability in the generated outputs, limiting the quality and richness of the generated samples.

- **Training Instability:** GAN training can be challenging and unstable, requiring careful tuning of hyperparameters and network architectures. The generator and discriminator may struggle to find a stable equilibrium, leading to oscillations and difficulties in convergence. Instability can manifest as sudden shifts in the training dynamics, resulting in fluctuations in loss and difficulties in achieving consistent improvement.

- **Nash Equilibrium and Convergence:** GANs rely on reaching a Nash Equilibrium, where the generator and discriminator reach a stable state. However, achieving convergence to this equilibrium can be difficult, and the training process may become stuck in suboptimal solutions. The equilibrium can be sensitive to the relative capacities of the generator and discriminator, making it challenging to strike the right balance between their capabilities.

4.2 Deep Convolutional GANs (DCGANs)

4.2.1 Introduction

Deep Convolutional Generative Adversarial Networks (DCGANs) are a variant of Generative Adversarial Networks (GANs) that specifically leverage deep convolutional neural networks (CNNs) for both the generator and discriminator models. DCGANs have been widely used for image generation tasks and have shown impressive results in generating realistic and high-quality images.

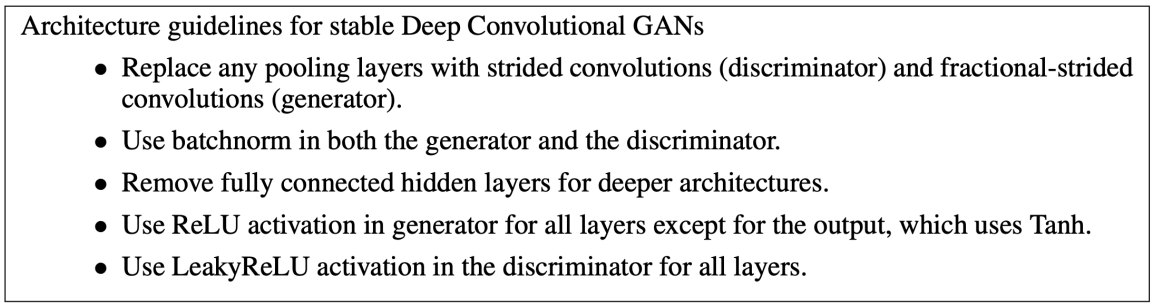

Figure 4.4: Architecture guidelines for stable Deep Convolutional GANs

Source: Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In International Conference on Learning Representations (ICLR).[8]

4.2.2 Key features of DCGANs

- **Convolutional Architecture:** DCGANs utilize convolutional layers in both the generator and discriminator networks, allowing them to effectively capture spatial dependencies and generate high-quality images.

- **Strided Convolutions and Transposed Convolutions:** DCGANs employ strided convolutions in the discriminator network for downsampling and transposed convolutions in the generator network for upsampling. This helps in learning hierarchical representations and generating higher-resolution images.

Figure 4.5: Celebrity Face Generation with Deep Convolution GAN

- **Batch Normalization:** DCGANs incorporate batch normalization, which stabilizes the training process by normalizing the input to each layer. This helps in accelerating convergence and improves the stability of the generated images.

- **Activation Functions:** DCGANs typically use the Rectified Linear Unit (ReLU) activation function for intermediate layers of the generator and discriminator. The generator's output layer often employs the hyperbolic tangent ($tanh$) activation function.

4.3 Conditional GANs

4.3.1 Introduction

Conditional Generative Adversarial Networks (CGANs) are an extension of the traditional Generative Adversarial Networks (GANs) that allow for targeted and controlled generation of data by incorporating additional conditioning information. In CGANs, both the generator and discriminator networks are conditioned on some extra input, such as class labels, text descriptions, or auxiliary data, which guides the generation process.

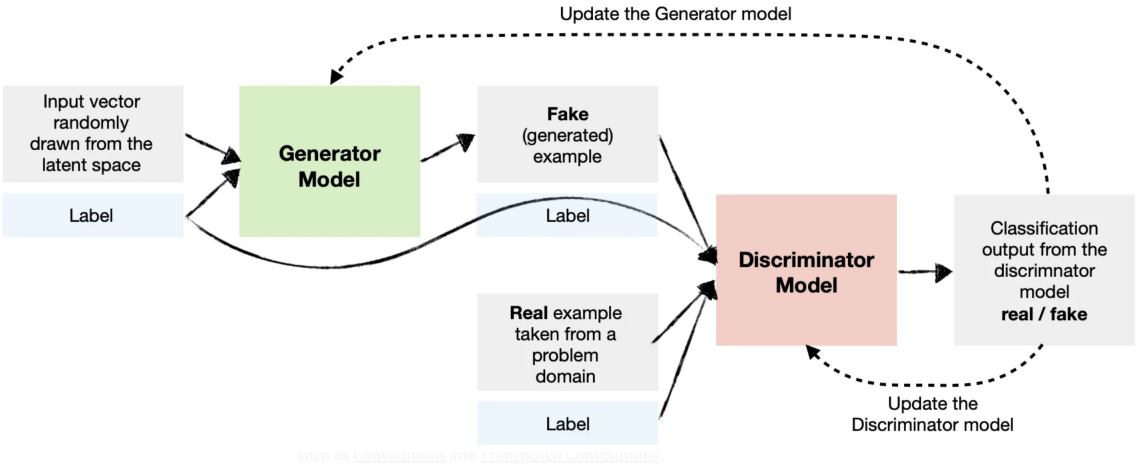

Figure 4.6: Conditional GAN Model Architecture

Source: cGAN: Conditional Generative Adversarial Network — How to Gain Control Over GAN Outputs

The concept of conditioning in CGANs adds an extra level of control and flexibility to the generation process. By conditioning the generator on specific input, CGANs can generate data samples that conform to certain desired characteristics or follow specific criteria. For example, in image generation tasks, CGANs can generate images of different classes or styles based on the provided class labels.

(a) MNIST like digits generated using Conditional GAN

(b) Implementation of Conditional GAN on CIFAR10 Dataset

Figure 4.7: Image Generation using Conditional GAN

4.3.2 Shortcomings

- **Limited Expressive Power:** CGANs heavily rely on the conditioning information provided during training and generation. If the conditioning information is insufficient or does not capture the desired characteristics adequately, the generated samples may not align with the intended criteria. Ensuring the conditioning information effectively captures the desired attributes is crucial to obtain high-quality results.

- **Difficulty in Obtaining Conditional Data:** In some cases, obtaining paired conditional data (e.g., paired images and corresponding labels) for training CGANs can be challenging or costly. The availability of labelled data that precisely matches the desired conditioning information may be limited, hindering the training and performance of CGANs.

Chapter 5

Conclusions and Discussion

In this project, we have explored the fundamental concepts of Generative Modeling right from the basics.

- We began by implementing simple and intuitive approaches such as Explicit Tractable Density Modeling using Multinomial Distribution, Naive Bayes, and Gaussian Discriminant Analysis.

- As we progressed, we delved into more advanced techniques, including Approximate Explicit Density Modeling using Variational Autoencoders (VAEs) and Implicit Density Modeling using Generative Adversarial Networks (GANs).

Throughout our exploration, we acknowledged the limitations of each model, which drove us to explore the next model in the line. It's worth noting that the sequence in which these models were developed may not align perfectly with their application in generative modelling. Despite the computational intensity associated with training generative models, we successfully obtained results from all models (where applicable) on a standard dataset. This enabled us to compare their advantages, disadvantages, and the impact of underlying mathematical principles on their performance.

In addition to generating novel sequences, generative models possess another fascinating property known as latent space interpolation. This property has practical applications across various domains. We have demonstrated this property using VAEs on the MNIST dataset and DCGANs on the Anime dataset. Along the way, we also gained insights into intriguing mathematical concepts such as KL Divergence, and Maximum Likelihood Estimation.

Overall, this project has provided us with a solid foundation in Generative Modeling, allowing us to explore a range of models and understand their mathematical principles. We have also witnessed the practical implications of generative models, including their ability to generate diverse sequences and perform latent space interpolation.

5.1 Future Work

This project has served as a valuable foundation for our understanding of generative modelling. Through the implementation of various techniques such as Naive Bayes, Gaussian Naive Bayes, Gaussian Discriminant Analysis, Gaussian Mixture Models, Autoencoders, Variational Autoencoders, and Generative Adversarial Networks (GANs) including DCGANs and CGGANs, we have gained a solid grasp of the fundamentals. This achievement has further ignited our curiosity and motivation to dig deeper into the vast landscape of generative modelling, inspiring us to explore alternative approaches like advanced Variational Autoencoders, and other GANs like WGAN, CycleGAN, StarGAN, and styleGAN. In addition to our existing implementations, we are thrilled to embark on projects involving video generation from images and speech-to-text conversion. By undertaking these exciting endeavours, we aspire to contribute to the continued advancement of this captivating field.

Bibliography

[1] G. A. F. Seber, *Multinomial Distribution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 882–884. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_388

[2] G. I. Webb, *Naïve Bayes*. Boston, MA: Springer US, 2010, pp. 713–714. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_576

[3] M. C. Belavagi and B. Muniyal, "Performance evaluation of supervised machine learning algorithms for intrusion detection," *Procedia Computer Science*, vol. 89, pp. 117–123, 2016, twelfth International Conference on Communication Networks, ICCN 2016, August 19– 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705091631081X

[4] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *CoRR*, vol. abs/2003.05991, 2020. [Online]. Available: https://arxiv.org/abs/2003.05991

[5] M. Gogoi and S. A. Begum, "Image classification using deep autoencoders," *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp. 1–5, 2017.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[7] C. Doersch, "Tutorial on variational autoencoders," 2021.

[8] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016.

[9] A. Asperti and M. Trentin, "Balancing reconstruction error and kullback-leibler divergence in variational autoencoders," *IEEE Access*, vol. 8, pp. 199 440–199 448, 2020.

[10] L. A. P. Rey, V. Menkovski, and J. Portegies, "Diffusion variational autoencoders," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, jul 2020. [Online]. Available: https://doi.org/10.24963%2Fijcai.2020%2F375

[11] C. Ok, G. Lee, and K. Lee, "Informative language encoding by variational autoencoders using transformer," *Applied Sciences*, vol. 12, no. 16, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/16/7968

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.