Exploratory Project on Generative Modelling and Generative Adversarial Networks

Kshitij Parwani, kshitijparwani.mat18@iitbhu.ac.in
Akshat Agrawal, akshatagrawal.mat18@iitbhu.ac.in

under the guidance of Dr. Pratik Chattopadhyay

November 13, 2019

Abstract

Generative Adversarial Network (GANs) have only been around for about 4 years but they have certainly taken the world by storm.They have been successfully trained and applied to domains like image generation, image to image translation, facial attribute manipulation etc.Much of the state-of-the-art performance in these domains has only been possible due to GANs.In this paper we first build up the motivation for generative modelling by considering a few models and also analyzing their performance on the MNIST dataset.Then we move onto GANs and look at the different types of GANS in use today.We address some of the problems faced when training GANs thereby also motivating the need for more sophisticated GAN models.

Introduction

GANs were first proposed in the paper: "Generative Adversarial Nets" (Goodfellow et al)[1] in 2014. They are a type of generative models which infact consist of two models (usually deep neural architectures): a generator and a discriminator. The job of the generator is to successfully create samples that resemble those from the original dataset and it's adversary : the dicriminator is charged with telling the real and generated images apart. So these two adversaries play a kind of a min-max game where one tries to outsmart the other and vice-versa. We train both the networks simultaneously because both adversaries cannnot learn without each other. This kind of simultaneous training often lead to problems which cause instability in the training process which are discussed later.

1 Generative Modelling

1.1 What is Generative Modelling?

A generative model is any model that takes in an input generated from a probability distribution say $\mathrm{P}_{\mathrm{true}}$ and tries to guess the generating distribution, say $\mathrm{P}_{\mathrm{pred}}$. This guess may be explicit as in a mathematical function or implicit as in it may only be able to generate outputs which resemble the input sample. Models like the Naive Bayes try to model this true distribution as a frequency distribution over each individual dimension. Note that this includes the assumption of all input dimensions being independent. On the other hand, models like variational autoencoders try to approximate $\mathrm{P}_{\mathrm{true}}$ by assigning

[1] arXiv:1406.2661v1

each point a gaussian distribution in the latent space (basically a lower dimensional space but more on that later). One must note that this is different from our traditional predictive models which try to model $P\left(y|X,\Theta\right)$ or the probability y of a label given some input X. These two tasks are quite analogous to 'telling if a painting is painted by Van Gogh' vs 'Actually painting in the style of Van Gogh'.

1.2 Need for Generative Modelling

One might ask why do we need to go through so much effort of training a model to actually fabricate data, when our current discriminative models are already doing a pretty good job of correct labelling and understanding data through supervised learning. Well, there are a couple of reasons:

1. Feature selection and creation has always been a long standing problem in machine learning. Being successfully able to train such a generative model is a proof that we are successfully able to extrapolate and manipulate some latent features in the data we have. This means that given high-dimensional data we are able to map it to lower–dimensional latent space which governs most observable aspect of our raw, unorganised output. For eg. for Images, these latent features might be the angle of photography, the lighting conditions, the subject of the photo etc. These latent features are if learnt correctly are indispensible in understanding and using the data that we have.
2. Secondly, as stated in the 'NIPS 2016 Tutorial on GANs by Ian Goodfellow'[2], Generative models, especially GANs, help to model multi-modal outputs. This means in situations which there are multiple correct outputs for any given input, GANs are particularly helpful. This may include tasks like:
   (a) Image generation given some textual pretext.
   (b) Predicting the next frame in a video
3. In some specialised domains, especially creative ones like music and art it is directly useful to get samples resembling a particular dataset but not exactly same as the input samples. This may include tasks like learning the distribution of all paintings painted in a particular era and therefore generate more of them. Or the task maybe to try to imitate the style of a composer and generate more similar but original compostions. No wonder, GANs are already helping artists in these fields create new and original art by providing inspiration for diverse but relevant results.

2 Types of Generative Models

2.1 Naive Bayes

This is the most naive generative model that comes to mind. It uses the dimensional frequency of the in-sample observations to model the probability of each dimension. It assumes that each dimension of the X variable is independent. We try to model $P(X)$ as follows (Consider X to be a vector of n-dimensions):

$$P(X) = P(x_1,x_2,...,x_n) = \prod_{i=1}^{n} P(x_i)$$

The dimensional probabilites are modelled as follows:

$$\hat{\theta}_{il} = \frac{n_{il}}{N}$$

where :

[2] arXiv:1701.00160v4

$\hat{\theta}_{il} = P(x_i = l)$,
n_{il} = No. of times $x_i = l$ in the dataset and N = Total no. of observations

Though this model performs decently for datasets which have high level categorical features, it fails miserably for regressive features.

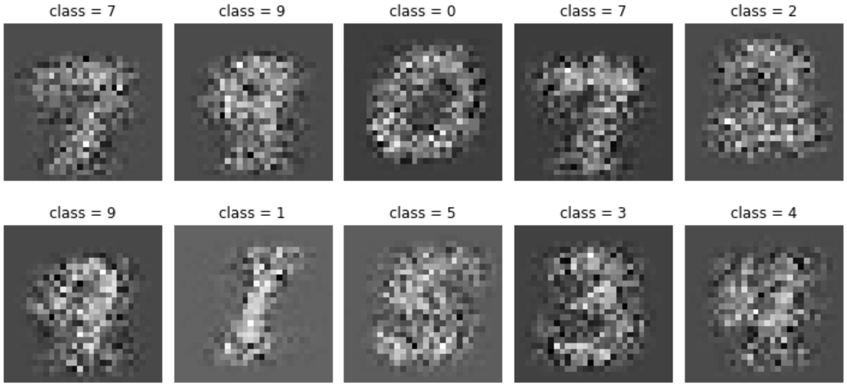
2.2 Gaussian Bayes

Drawing from the last model, this model is a bit smarter as it tries to model each dimension as being sampled from a gaussian distribution. Note that the assumption of feature independence still remains. Sampling it from a gaussian distribution enables us to generate samples which somewhat different from the original dataset. But it is not able to generate sharp and distinct output in case when the input dimensional space is large and thus of a lower-level.

$P(X) = P(x_1,x_2,...,x_n) = \prod_{i=1}^{n} P(x_i)$
$x_i \sim N(\mu_i, \sigma_i)$
where μ_i and σ_i are the mean and standard deviation of x_i across the original dataset. So in essence we are learning 'n' gaussians, one for every feature.

Below we have shown the results for this type of model on the MNIST dataset. For this we have have used labels to improve the modelling process. In essence we have done the above-mentioned process differently for each class. So we have learnt 784(input dimension for MNIST) gaussians for 10 classes (0-9). The sampling function takes in a class and input and samples an image for that digit. Fig 1 shows the image obtained for various classes:

Figure 1: Gaussian Bayes for MNIST dataset

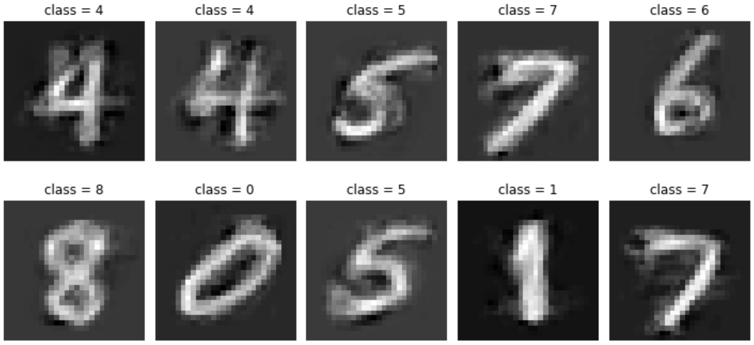


2.3 Gaussian Mixture Models

Gaussian Mixture are a slight upgrade on Gaussian Bayes in the sense that even for a single class, they try to learn a number of localised clusters. In case of MNIST this may represent different styles of writing a single digit. Imagine looking at some images of handwritten 2's and being told to split them into a number of groups depeding on how they are written. You may use criteria such as size of loop, roundness of the digits etc. Gaussian Mixture models do a similar thing but in a unsupervised manner.

This is done using the Expectation-Maximization algorithm. We don't go into much of the details, but the results for GMMs are shown in Fig2. We can already see that we get much better pictures than Gaussian Bayes. This hints in case of handwritten digits, there is a meaningful and inherent structure of numerous clusters and thus learning multiple Gaussians for each digit (here a maximum of 10) greatly improves the generative process.

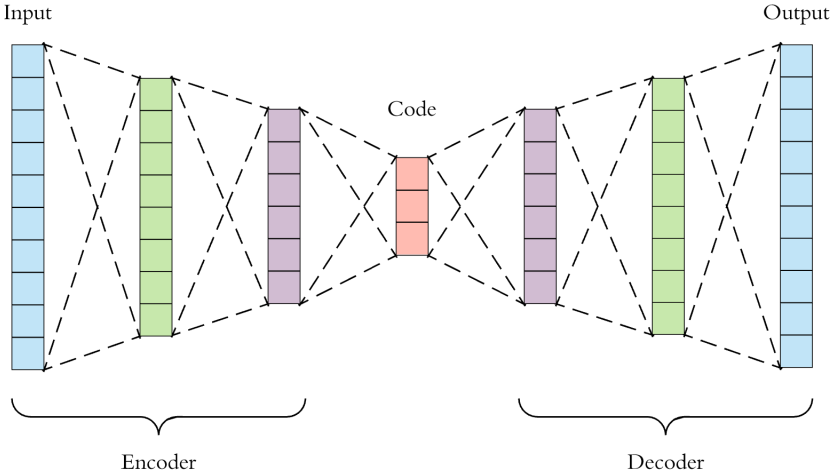
Figure 2: Using Gaussian Mixture Models on MNIST database:



2.4 Autoencoders

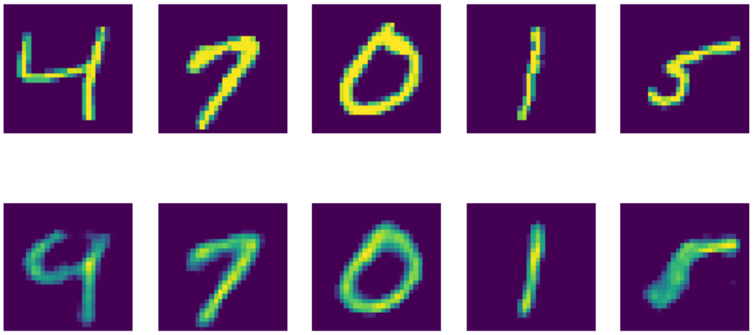
Autoencoders are deep neural architectures consisting of two networks : an encoder and a decoder. The job of the encoder is to take a higher dimensional vector as input and generate a lower-dimensional representation. This basically means assigning every point in the higher dimensional space to a point in a lower dimensional space. The job of the decoder is to take that lower dimensional representation and recreate the higher dimensional input. So basically it is a collection of two neural networks that train to do the inverse of each other.

Figure 3: The architecture of a typical Autoencoder



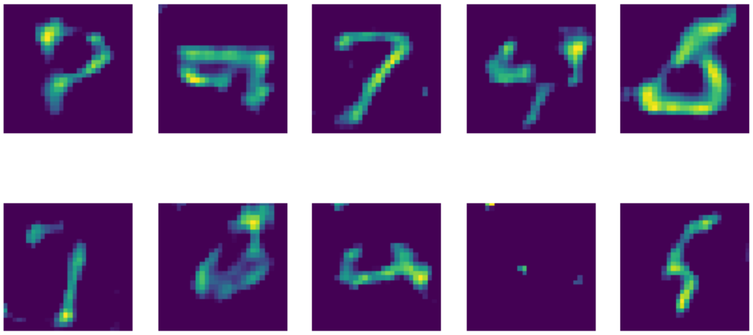
When both are trained simultaneously with the reconstruction loss as the loss function (MSE between the original and reconstructed image in this case), the system learns to compress the information of images in an efficient way so as to provide good reconstructed images. Now in order to generate images we take the decoder and pass it random input in the lower-dimensional space and hope with enough training, it will give good quality images as as output.

Some reconstructions for certain digits:
(The first row shows the actual images and the second row shows the reconstructed images)



The reconstruction seems to be quite good for the most part implying our model was able to fit the set effectively. The following figure shows some generated output for random input:

Some generated images for random input:



As evident the generated images are not satisfactory. This is because of following reasons:

1. When training the autoencoder, our loss function did not incentivize the model for generating high quality and diverse images from random input. This is a major problem in the above mentioned models which is addressed by models like GANs. Though good reconstruction is guaranteed because of the MSE function used, it is offers no advantage when generating images from random input.
2. Secondly, we have not made any efforts to make sure that our latent space is continuous. For eg, if in the latent space we move from one digit to other and if we don't get numbers along the way, it means the latent space of digits that our model has learned is not continuous. This issue is addressed by Variational Autoencoders.

2.5 Variational Autoencoders

Variational autoencoders are different from autoencoders because instead of mapping each image to a point in the lower dimensional space, VAEs map each image to a normal distribution with a certain μ and σ. This serves two purposes :

1. It ensures that the latent space is continuous, ie if a certain point has a high probability of being a 9, then the points surrounding it are also similar looking in the higher-dimensional image space. This is because now, the probability at a certain point is determined by a super-position of various normal distributions nearby.

2. Because now each image is mapped to a normal distribution, we use a random sample drawn from normal distributions to generate images. This is because now our decoder is trained specifically to produce good images from normal input and thus it does a better job in reconstruction.

How does the decoder get its input then? It is only given the μ and σ by the encoder and it gets its point by sampling from this distribution. This introduces a degree of uncertainty in our predictions.

VAEs feature a different kind of component in the loss function when training. This is called the Kullback-Leibler (KL) divergence. It is way of measuring, how different two probability distributions are. We want our VAE to generate images from input randomely generated from the standard normal distribution. So we add a term which measures the difference between our assigned normal distribution : $N(\mu, \sigma)$ and the standard normal distribution : $N(0, 1)$.

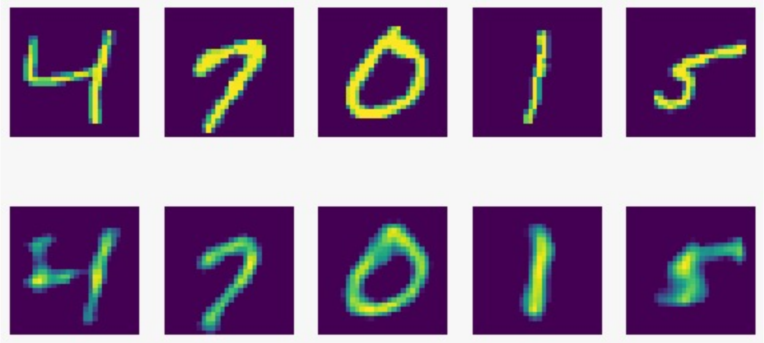
$$D_{KL}[N(\mu, \sigma)||N(0,1)] = \frac{1}{2}\sum(1 + \log(\sigma^2) - \mu^2 - \sigma^2)$$

So now our loss function becomes :

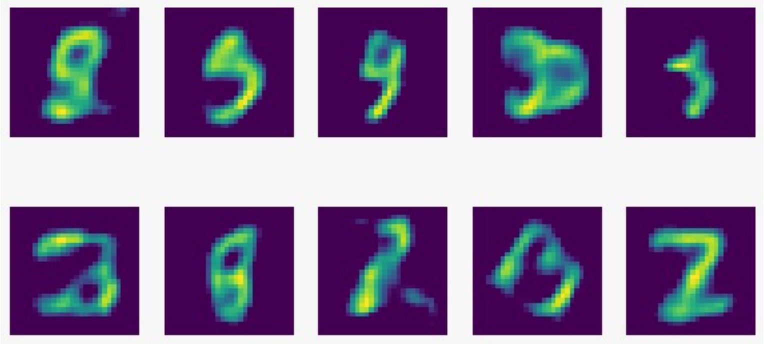
$$\text{Loss function} = D_{KL} + r.R_L$$

where R_L is the reconstruction loss and,
r is the reconstruction factor (which determines how much we want to weight the reconstruction loss against the KL loss.)

Reconstructed Images for VAE:



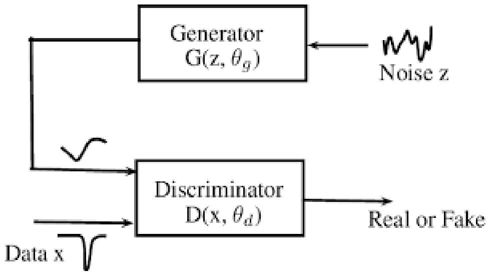
Some generated images for random input generated from the standard normal distribution:



2.6 Generative Adversarial Networks (GANs)

Finally, we come to GANs. This is a fairly recent development (2014) and has been called by many as the greatest idea in the AI community in the recent years. GANs are an adversarial system of two networks: a generator and a discriminator. The generator is tasked with generating high quality images given random input and the discriminator is trained to tell the real and generated images apart.

Diagram taken from paper by Bajibabu Bollepalli, Lauri Juvela, Paavo Alku [researchgate.com]

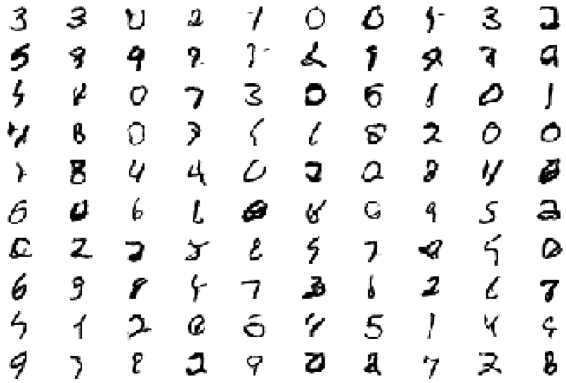


The training procedure for GANs is mainly as follows:

1. Freeze the weights of the generator. Generate a few images from the generator given random input.
2. Take a few images from the real dataset. Concatenate them with the generated images.
3. Feed this to the discriminator model and train the disciminator for a single batch of these images. By labelling generated images as 0 and real images as 1, we train the discriminator to tell these images apart.
4. Now freeze the weights of the discriminator, and train the generator. The generated images are scored by labelling all the images as real and computing the loss function subsequently.
5. Repeat this process for a number of epochs.

Finally, we show the results obtained on the MNIST dataset for GANs. The resuts are much more likely to pass off as handwritten digits.

Generated images for random noise for a DC-GAN (Convolutional GAN):



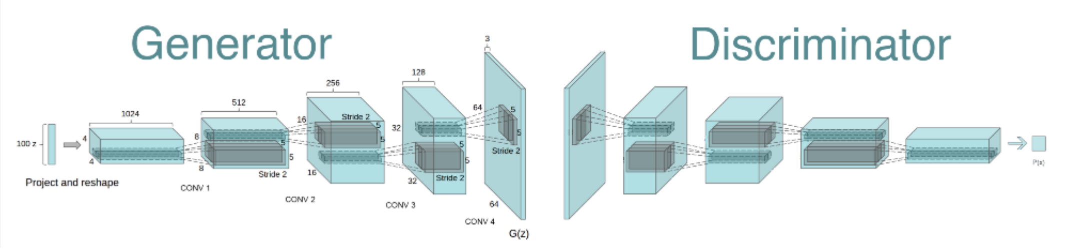
In the next section, we inspect GANs for various domains.

3 Generative Adversarial Networks

3.1 Deep Convolutional Generative Adversarial Network

DCGAN is a flavour of GANs, in which both the generator and discriminator are both convolutional networks.

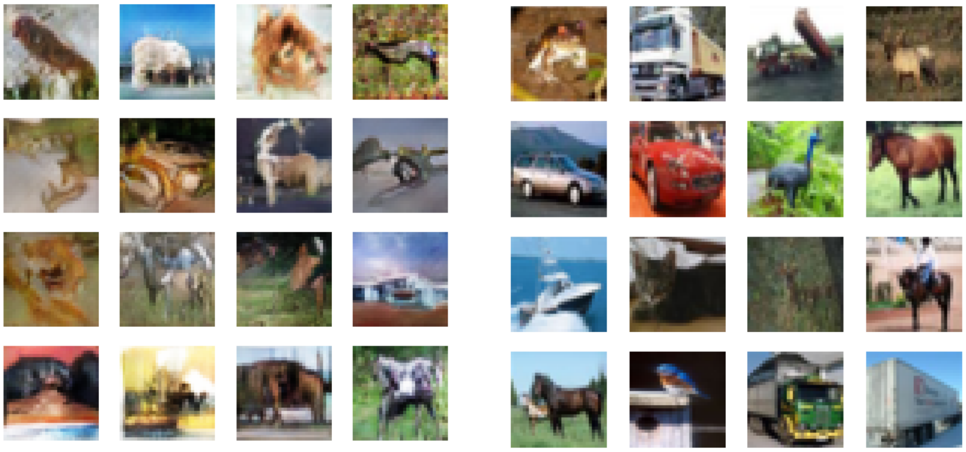
Architecture of a typical DCGAN:



In this particular model, we used transposed convolutions for upsampling and batch normalization for faster training.

Below we show the results of DCGAN on the CIFAR-10 dataset.

Generated random images VS Real images from the dataset



Though the generated images seem a bit blurry, they are still pretty good given the quality of images in the CIFAR-10 dataset.