A comprehensive survey and analysis of generative models in machine learning

Article *in* Computer Science Review · July 2020

DOI: 10.1016/j.cosrev.2020.100285

Harshvardhan Gm
KIIT University
11 PUBLICATIONS **29** CITATIONS
SEE PROFILE

Mahendra Kumar Gourisaria
KIIT University
45 PUBLICATIONS **118** CITATIONS
SEE PROFILE

Manjusha Pandey
109 PUBLICATIONS **449** CITATIONS
SEE PROFILE

A Comprehensive Survey and Analysis of Generative Models in Machine Learning

Abstract

Generative models have been in existence for many decades. In the field of machine learning, we come across many scenarios when directly learning a target is intractable through discriminative models, and in such cases the joint distribution of the target and the training data is approximated and generated. These generative models help us better represent or model a set of data by generating data in the form of Markov chains or simply employing a generative iterative process to do the same. With the recent innovation of Generative Adversarial Networks (GANs), it is now possible to make use of AI to generate pieces of art, music, etc. with a high extent of realism. In this paper, we review and analyse critically all the generative models, namely Gaussian Mixture Models (GMM), Hidden Markov Models (HMM), Latent Dirichlet Allocation (LDA), Restricted Boltzmann Machines (RBM), Deep Belief Networks (DBN), Deep Boltzmann Machines (DBM), and GANs. We study their algorithms and implement each of the models to provide the reader some insights on which generative model to pick from while dealing with a problem. We also provide some noteworthy contributions done in the past to these models from the literature.

Keywords: generative models; machine learning; deep learning; neural networks.

Contents

Introduction

Modern day machine learning classifiers mainly include logistic regression, support vector machine (SVM), supervised feed-forward deep neural networks, nearest neighbour, conditional random fields (CRFs), etc. All of these models focus on the discriminative classification process – where they only model the decision boundary between the classes by learning directly from the training data. However, with generative models, this is not the case as these models assume that data is created by a probability distribution which is then estimated and a distribution very similar to the original one is generated. The probability of the target variable conditioned on the given input variable is then calculated based upon this generated distribution. Fundamentally, both discriminative and generative classifiers perform the same task because of the last step of calculating the conditional probability of the target variable. Mathematically speaking, if we have two variables X and Y as the independent and target variable respectively, discriminative classifiers only estimate the parameters of $P(Y|X)$ whereas generative models *estimate* the distribution given by $P(X|Y)$ and $P(Y)$ with particular algorithms, finally using Bayes' rule to calculate $P(Y|X)$. The Naïve Bayes' classifier is based on this principle, which is also considered to be generative. One may arbitrarily sample from $P(Y|X)$, compute the modes of the distribution $argmax_Y P(Y|X)$, or find expectation of the distribution $P(Y|X)$. Before estimation, this joint distribution may be constrained to having lower degrees of freedom which may be done by structurally studying the conditional independencies between all the variables of the joint distribution as illustrated by Fig. 1 for n variables $X_1, X_2, \dots, X_n$ of the joint distribution $P(X_1, X_2, \dots, X_n)$.

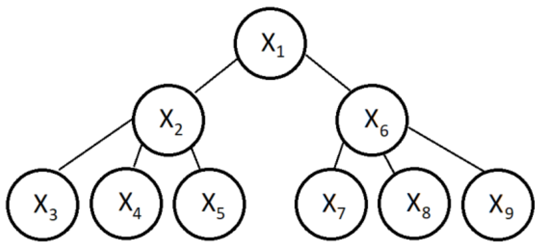


Fig. 1 A directed graph model.

Fig. 1 demonstrates the factorization product of conditional distributions over variables conditioned on parents π_i for variable X_i as,

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^{n} P(X_i|X_{\pi_i}) \quad \textbf{(1)}$$

In many cases, it is difficult to directly estimate the probability of Y conditioned on X hence we use generative models to produce a distribution which resembles the original distribution of the probability of X conditioned on Y to *then* run a classifier similar approach and calculate the probability of the target Y conditioned on X. Training generative models, especially deep generative models, takes longer than discriminative models since creating a probability distribution resembling the original involves a substantially higher number of correlations to learn as opposed to simply labelling instances to their most probable classes as discriminative models do. For example, a convolutional neural network (CNN) classifier only has to spot a few tell-tale differences between the images of cats and dogs to differentiate them as opposed to a deep convolutional generative adversarial network (DCGAN) which has to generate images of cats and dogs by learning all the features, even the

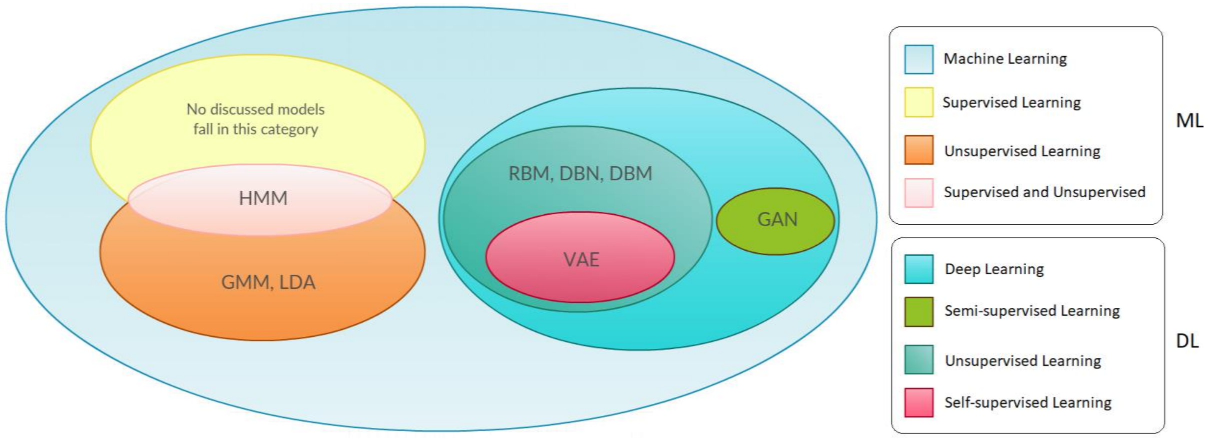


Fig. 2: Classification of different generative models discussed in this paper with respect to ML and DL (Machine Learning and Deep Learning).

ones that the CNN may have skipped. Fundamentally, discriminative models only draw the decision boundary in a data space, whereas generative models learn the overall distribution of the data. Generative models, as we shall see, have their contemporary uses as powerful feature extraction tools [36], or in regression, clustering and classification [35], pattern recognition followed by generation [10, 11], recommendations [15], topic modelling, text generation, etc. Classifiers have recently found useful applications in object tracking and detection through algorithms like the Single Shot Detection [101], YOLO (You Only Look Once) [102], Faster R-CNN [103] and Masked R-CNN [104] which take the normal classification of visual data one step further by detecting objects within a picture with apt bounding boxes. These approaches are used in cutting-edge research in the field of self-driving cars and computer vision based automation in the industry. Classifiers also find extensive use in biological applications in diagnosis of diseases [105], weather prediction [106], assessment of potential high-risk loan applicants for banks [107], and even used in sorting important and spam mail in every modern day emailing services like Gmail, Yahoo!, etc.

We notice that there is a huge focus on discriminative modelling, in other words, classifier-based approaches in problem solving while generative models have not yet sought comparable prominence. The use of generative models is of paramount importance, the reason being two-fold: a) they may be used to select indicative features and act as feature selection tools to facilitate classification and increase model accuracy, and b) they can be applied to generate realistic data samples which is not something the discriminative models are capable of. The motivation of this survey arises from these two pertinent aspects of generative models, combined with the fact that the amount of research done on them does not do justice to their capabilities.

In this paper, we introduce all the generative models (and thus put forward an exhaustive list) section-wise and review the work done on them as: 1. *Gaussian Mixture Models (GMM)*, 2. *Hidden Markov Models (HMM)*, 3. *Latent Dirichlet Allocation (LDA)*, 4. *Boltzmann Machines (BM)*, 5. *Variational Autoencoders (VAE)*, 6. *Generative Adversarial Networks (GAN)*, and 7. *Analysis and Discussion*, where we analyse and implement all the models discussed in this paper and compare them so the reader can gain insights on which model (if they are looking to implement one of the generative models for a task) to pick from to best suit their needs. The models discussed in this paper are differently classified under machine learning as shown in Fig. 2.

1. Gaussian Mixture Models

A Gaussian distribution is simply another name for a normal distribution which is continuous for a real valued random variable and is symmetric about its mean. The probability of occurrence of data is more likely in the vicinity (left and right sides) of the mean and tapers off at the edges tending to be zero or becoming asymptotic to a real value like zero. In certain situations, we deal with different Gaussians (or curves that are normally distributed) in a single graph (Fig. 3). The magnitude of these different Gaussians can be represented by *weights* $\pi = \{\pi_1, \pi_2, \pi_3, \ldots, \pi_n\}$, *mean* $\mu = \{\mu_1, \mu_2, \mu_3, \ldots, \mu_n\}$, *variance* $\rho = \{\rho_1, \rho_2, \rho_3, \ldots, \rho_n\}$, for a mixture of n Gaussians. In our case, $n = 3$ and $0 \leq \pi_i \leq 1$, where $i \in \{1, 2, 3\}$. The weights are such that,

$$\sum_{i=1}^{n} \pi_i = 1 \quad \textbf{(2)}$$

The magnitudes of the weights associated to each Gaussian sum to 1 as they represent the *prior* probability of finding that cluster i when taking into consideration the set of all the data.

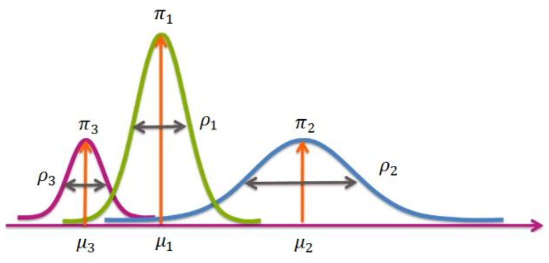


Fig. 3 Mixture of three Gaussians where π signifies the weight associated to the Gaussian and hence also the probability of the data belonging to the i^{th} cluster or Gaussian, μ specifies the position of the Gaussian with the mean, ρ signifies the 'spread' of the Gaussian over the overall distribution by the variance.

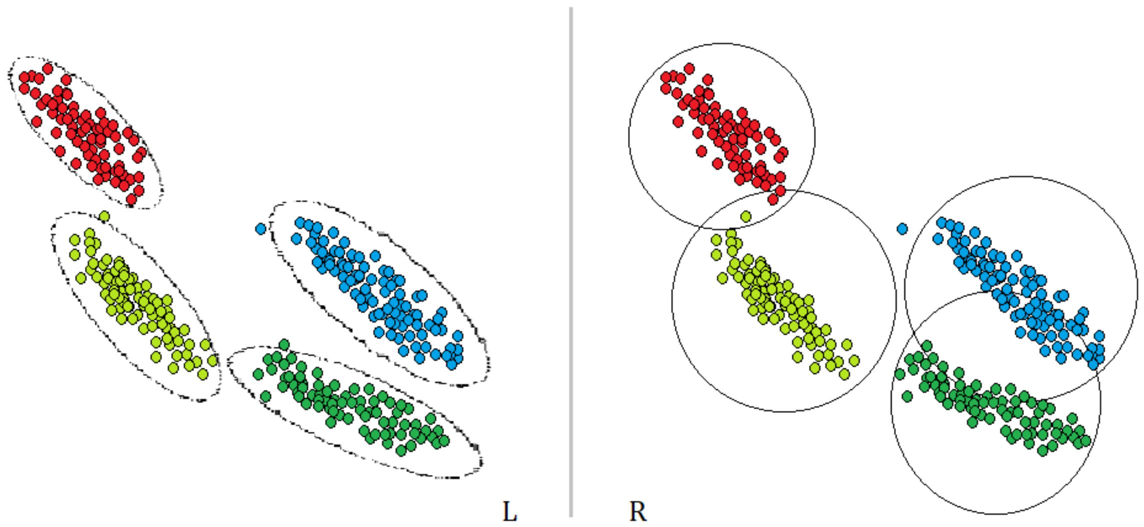


Fig. 4 (**Left**): Data clustering done by GMMs for oblong data (oval in this case). If visualized in 3D, these datapoints may be coplanar and lying on the surfaces of a circular planes inclined at different angles. (**Right**): Data clustering done by K-means algorithm which is a special case of GMM clusters for circular clustering. It is clearly visible that GMMs outperform K-means clustering algorithms for a generalized dataset.

Since our Gaussian mixture model is in one dimension, each unique cluster can be identified by the 3-tuple format given by: $\{\pi_i, \mu_i, \rho_i^2\}$. In 2D, π_i is a vector and ρ_i a covariance matrix. The prior probability of any random data point z_k from a dataset of d points (where $1 \leq k \leq d$) to belong to a particular cluster i is given by the equation,

$$P(z_k = i) = \pi_i, \quad \textbf{(3)}$$

where the argument of the probability P is the cluster assignment for observation z_k. However, if we are given that z_k is from a cluster i, the likelihood of observing x_k is given as,

$$P(x_k \mid z_k = i, \mu_i, \rho_i) = N(x_k \mid \mu_i, \rho_i) \quad \textbf{(4)}$$

$N(x_k \mid \mu_i, \rho_i)$ is the likelihood which is a single Gaussian with a mean μ_i and variance ρ_i. Another equation that represents a weighted

Gaussian mixture model with N components can be written as,

$$P(x \mid \vartheta) = \sum_{k=1}^{N} w_i b(x \mid u_k, \rho_k), \qquad \textbf{(5)}$$

where ϑ is the 3-tuple format used to identify clusters as discussed before and $b(x \mid u_k, \rho_k)$ are the component Gaussian densities for $k = 1, 2, \ldots, N$ defined by,

$$b(x \mid u_k, \rho_k) = \frac{1}{(2\pi)^{M/2}|\rho_k|^{1/2}} e^{-\frac{1}{2}(x- \mu_k)(\Sigma_k(x- \mu_k)^{-1}} \qquad \textbf{(6)}$$

for a mixture of Gaussians in an M dimensional space.

GMMs are considered a generalization of the K-means clustering algorithm [1] because the latter is only prominent at detecting clusters of a circular shape in 2D (and hyper-sphere in higher dimensions). However, GMMs can form clusters of oblong nature as illustrated in Fig. 4.

Albeit GMM can be termed as a clustering algorithm, it is more correct to call it an algorithm for density estimation. When the GMM fits on a data, it is a *generative probabilistic model* giving us the recipe to generate new data distributed similar to the distribution to which the GMM was fit. [2] mentions the Kullback Leibler (KL) Divergence technique which is a tool used for statistics and pattern recognition. The KL divergence is used to measure the similarity between two density functions $p(x)$ and $q(x)$ defined by:

$$D(p||q) = \int p(x) log \frac{p(x)}{q(x)} dx \qquad \textbf{(7)}$$

$D(p||q)$ is always ≥ 0 and is 0 iff $p = q$. This method is not analytically tractable and hence [2] puts forward two methods namely variational approximation and variational upper bound to measure the similarity between GMMs. GMMs are also used in language identification systems as described in [3] where the two approaches mentioned use shifted delta cepstra (SDC) feature vectors. The first approach is based on acoustic scoring which is an identification system composed of a pre-processor to extract features, a backend classifier and a Gaussian mixture of all the target languages. The second approach is done through GMM tokenization as shown in Fig. 5 which comprises a parallel sequence of GMM tokenizers which feed a bank of tokenizer dependent interpolated (unigram and bigram) language models. A sequence of symbols is generated by each tokenizer which corresponds to the per frame indices of the Gaussian component having the highest score. Now, the likelihood of each of these sequences is fed to the language models which generate scores for the corresponding language. These scores are given to the final backend classifier to get the results.

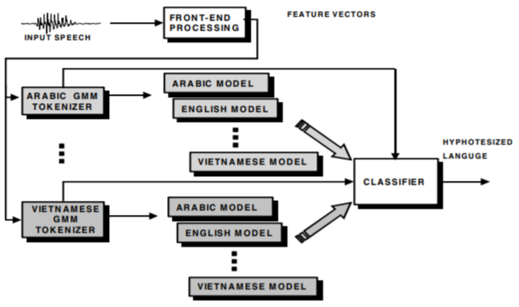


Fig. 5 *P. A. Torres-Carrasquillo et al.* The implementation of GMM tokenization system.

GMMs have been used for speech recognition [3] and even more sophisticated tasks like accent recognition [4]. The implementation for GMMs is demonstrated in Section 7.2.1.

2. Hidden Markov Models

Hidden Markov models are used to generate sequences named as *Markov chains* that are a series of states having certain state-transition probabilities which generate state sequences having corresponding symbol-emission probabilities (Fig. 6). The series of states are said to be Markov chains because the probability of reaching the next state is dependent on the transition function of the current state. HMMs are used in statistical modelling for linear problems involving time series or sequences and have many common links with probabilistic non-deterministic finite automata. In fact, as proven in [5], HMMs are equivalent to probabilistic automata with no final probabilities which can be converted into equivalent probabilistic non-deterministic finite automata. Hidden Markov models describe a probability distribution over a non-finite number of possible sequences.

The probabilistic automata as shown in Fig. 6 generate hidden state sequences based on the state transition probabilities. Each state *emits* residues or symbols based on their symbol-emission probabilities and finally we have an observable symbol sequence. On the contrary, the Markov chain (state sequence) that led to these emissions are not observable; they are hidden and hence the name *hidden* Markov models. HMMs are a probabilistic representation of a system they model, so these states and their transitions need to be constructed which aptly describe the behaviour of the system.

Hidden Markov models have been extensively used for speech recognition [6-9] and in other instances, for biological sequence modelling [10, 11] and optical character recognition (OCR) [12]. The implementation for HMMs is demonstrated in Section 7.2.3.

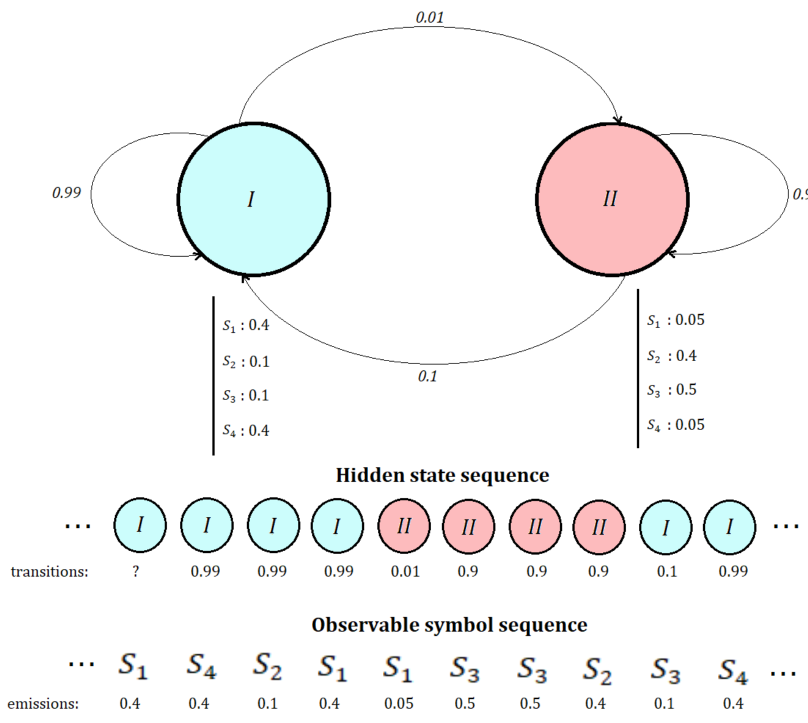

0.01
0.99
I
II
0.9
S_1 : 0.4
S_2 : 0.1
S_3 : 0.1
S_4 : 0.4
0.1
S_1 : 0.05
S_2 : 0.4
S_3 : 0.5
S_4 : 0.05
Hidden state sequence
transitions: ? 0.99 0.99 0.99 0.01 0.9 0.9 0.9 0.1 0.99
Observable symbol sequence
S_1 S_4 S_2 S_1 S_1 S_3 S_3 S_2 S_3 S_4
emissions: 0.4 0.4 0.1 0.4 0.05 0.5 0.5 0.4 0.1 0.4

Fig. 6 A simplified HMM with no initial and final states for the sake of simplicity. Let there be a set of symbols defined by $S = \{S_1, S_2, S_3, S_4\}$. The two states that generate the Markov chain are labelled as I and II. State I generates sequences comprising S_1 and S_4 more frequently whereas state II generates sequences comprising S_2 and S_3 more frequently (each state's symbol emission probabilities are stated below the respective state). All the state-transitions are implemented through arrows with their corresponding probabilities. Finally, the probability of the observable symbol sequence is the product of state-transition and symbol emission probabilities.

3. Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) [13] is a generative technique mainly used for topic modelling, although, more broadly, is considered a dimensionality reduction technique. Topic modelling is the process of making a machine predict the relevant *topics* associated with the input text. We next describe the process followed by LDA to achieve this.

LDA assumes there to be a vocabulary with W distinct words, each word described by W_j where $0 \leq j \leq W-1$ and T different topics. Each topic T_i, where $0 \leq i \leq T-1$, is described by a probability distribution ω_{T_i} over all the W words each with Dirichlet prior β. Then ω_{T_i,W_j} denotes the probability of topic T_i being represented by word W_j. If there are D documents (by documents, we do not mean the entirety of an actual document containing many paragraphs, instead, it is implied that a document can be seen as a block of text or a paragraph) then essentially β is defined as the distribution of words W over all documents D. Similarly, another Dirichlet prior α is defined as the distribution of topics T over all documents D. Let z notate each topic which is assigned to each word thereby making each document a mixture of these topics. Let there be N_{D_k} words in each document D_k where $0 \leq k \leq D-1$ and probability distribution of each document over all the topics given by φ_{D_k} drawn from the Dirichlet distribution with parameter α. So, we can say φ_{D_k,T_i} is the probability that document D_k is associated with topic T_i. Assuming α, β as scalars (which, however, we take as vectors in Fig. 7 and while defining the Dirichlet distribution later), LDA involves iterating through each document D_k having N_{D_k} words. For word W_j, a topic assignment is drawn z_{D_k,W_j} from the categorical distribution φ_{D_k}, and then a word V_{D_k,W_j} is drawn from the categorical distribution $\omega_{z_{D_k,W_j}}$. The algorithm is summarized as follows:

1. Draw $\varphi_{T_i} \sim Dir(\beta)$ for each $0 \leq i \leq T-1$.
2. Consider D_k for each $0 \leq k \leq D-1$:

(i) Draw $\omega_T \sim Dir(\alpha)$

(ii) Draw $z_{D_k,W_j} \sim Cat(\varphi_{D_k})$ for each $0 \leq k \leq D-1$

(iii) Draw $V_{D_k,W_j} \sim Cat(\omega_{z_{D_k,W_j}})$ for each $0 \leq k \leq D-1$,

where Cat denotes the categorical distribution and Dir denotes the Dirichlet distribution whose argument is either one of the Dirichlet priors α or β. The Dirichlet distribution parameterized by vector $\boldsymbol{\alpha}$ is given by:

$$Dir(\varphi_{D_k}|\boldsymbol{\alpha}) = \frac{1}{\mu(\boldsymbol{\alpha})}\prod_{k=0}^{D-1} D_k^{\alpha_k - 1}, \qquad \textbf{(8)}$$

where, μ is the Beta distribution function given by:

$$\mu(\boldsymbol{\alpha}) = \frac{\prod_{k=0}^{D-1} \tau(\alpha_k)}{\tau(\sum_{k=0}^{D-1} \alpha_k)}, \boldsymbol{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{D-1}) \qquad \textbf{(9)}$$

$\tau(x)$ in eqn. (9) is the complete gamma function defined by

$$\tau(x) = (x-1)! \qquad \textbf{(10)}$$

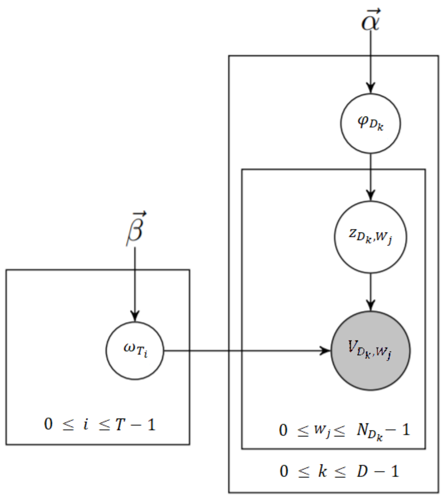


Fig. 7 Graphical plate notation for Latent Dirichlet Allocation. The grey shaded portion signifies an observable entity.

This method can be used for various tasks like web-spam filtering [14], tag recommendation [15], bug localization [16] which involves locating potential buggy files in a software. LDA, apart from tasks revolving topic modelling, has also been used for annotating satellite images into various regions like residential regions, golf courses, deserts, commercial and urban areas as shown in [17]. The implementation for LDA is demonstrated in Section 7.2.2.

4. Boltzmann Machines

In this section, we will be discussing about three different kinds of Boltzmann machines as *4.1 Restricted Boltzmann Machines (RBM)*, *4.2 Deep Belief Networks (DBNs)*, and *4.3 Deep Boltzmann machines (DBMs)*. Why Boltzmann machines (BMs) on their own are not discussed here is because BMs are impractical to deploy due to various computational constraints.

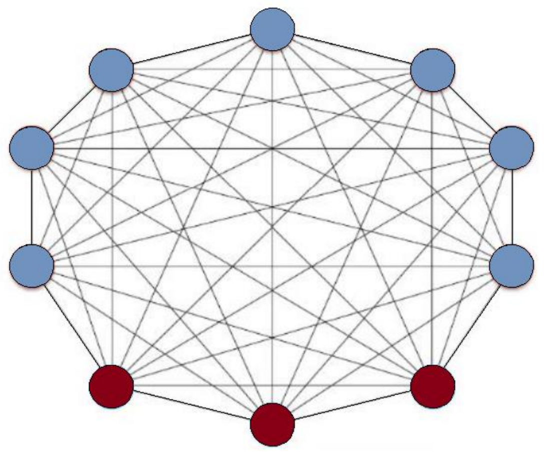


Fig. 8 The Boltzmann Machine where blue-grey nodes are hidden and maroon nodes are visible.

Boltzmann machines are undirected networks composed of many nodes linked with each other via weighted connections. BMs represent a class of unsupervised neural networks which don't try to minimize a loss or achieve a target, instead, they generate data to form a system (which usually is a probability distribution) that closely resembles the original system. There are certain visible and hidden nodes chosen for our convenience where visible nodes are used as the input and output, shown in Fig. 8. This is because after feeding the visible nodes, through *contrastive divergence* [18] which involves Gibbs' sampling (a Monte Carlo algorithm), the visible nodes iteratively feed the hidden nodes through weights, and in return, the hidden nodes feed the visible nodes. This one iteration can also be called a single Monte Carlo Markov Chain walk as a Markov chain is generated at the visible nodes layer.

Pragmatically, it is not easy to sample each iteration when all the nodes are connected to every other node. Hence, the Restricted Boltzmann Machine was proposed.

4.1 Restricted Boltzmann Machine

In RBMs, connections between visible-visible and hidden-hidden (feature detector) nodes are forbidden and hence we end up with a structure as shown in Fig. 9.

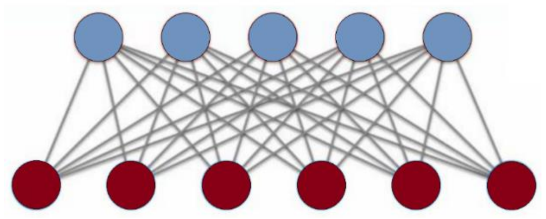


Fig. 9 The Restricted Boltzmann Machine

RBMs (and BMs), in general, are energy based models [19] where the energy of the joint configuration of visible and hidden nodes $E(\boldsymbol{v},\boldsymbol{h})$ is given by [20] as:

$$E(\boldsymbol{v},\boldsymbol{h}) = -\sum_{i\in visible} p_i v_i - \sum_{j\in hidden} q_j h_j - \sum_i \sum_j v_i w_{i,j} h_j, \quad \textbf{(11)}$$

where v_i and h_j are the states of the visible node i and hidden node j, p_i, q_j are their biases and weights between them is denoted by $w_{i,j}$. The RBM makes use of the following formula to assign a probability between each hidden and visible vector pair,

$$p(\boldsymbol{v},\boldsymbol{h}) = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}, \quad \textbf{(12)}$$

where the denominator of the RHS is also called as the partition function. Contrastive divergence follows the gradient for the learning as given by

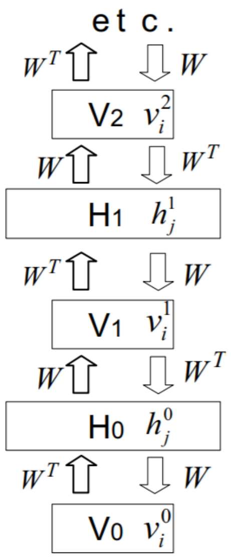
$$\frac{\delta \log p\left(v^{initial}\right)}{\delta\, w_{i,j}} = \langle v_i^{initial} h_j^{initial} \rangle - \langle v_i^{final} h_j^{final} \rangle \quad \textbf{(13)}$$

Through this gradient formula, the lowest energy state is achieved by adjusting the weights.

RBMs have been used for collaborative filtering in the field of recommender systems [21, 22], facial recognition [23], phone recognition [24-26] where [25] replaces each Gaussian mixture in a traditional spectral model with an RBM to outperform the former by modelling the joint probability distributions of the source and target spectral features. [26] uses conditional RBMs (cRBMs) to outperform previously made attempts at phone recognition through HMMs, and sentiment analysis and aspect extraction [27] among various other applications. The implementation for RBMs is demonstrated in Section 7.1.1.

4.2 Deep Belief Networks

Extending the idea of RBMs further, DBNs can be called as a network of stacked up RBMs (an RBM is a single level DBN). However, training DBNs is not very simple as there is a phenomenon called "explaining away" in Bayesian networks that takes place while inferring the hidden variables in the hidden layers as the posterior distribution over the hidden variables is intractable. Monte Carlo Markov Chains (MCMC) can be used to sample from these intractable posterior distributions, however, they are very time consuming. Explaining away occurs when one of the causes of an effect explains the effect entirely thereby reducing the probability of other causes to be responsible. Another problem with training DBNs is when the prior assumes independence at the deepest hidden layer with initial randomized weights. It would be convenient to



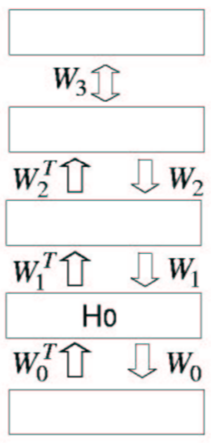


Fig. 10 Hinton *et al.* (**Left)**: An infinite logistic belief net having tied weights. The upward arrows are not part of the generative model as they are only used for sampling inference from the posterior distribution at every hidden layer when the data vector is given to v_o. Whereas, the downward arrows represent the generative model. (**Right)**: the hybrid model with undirected connections between the top two layers representing an RBM and directed top-down connections below representing the generative model whereas the bottom-up connections infer a factorial representation in the layer from the layer below it. In the greedy learning process initially the top-down and bottom-up weights are tied.

eliminate the explaining away effect and the independence of the prior both to train the DBNs more quickly and efficiently.

We take a logistic belief network with stochastic binary units [28] to generate data, where the probability of activating a unit i is a logistic function of its immediate prior neighbours j and the weights between them $w_{i,j}$ (the connections are directed to i from j).

$$P(s_i = 1) = \frac{1}{1 + e^{(-b_i - \sum_j s_j w_{i,j})}} \quad \textbf{(14)}$$

The bias of unit i is given by b_i. Considering the belief network to have only one hidden layer, we can say that this hidden layer is factorial (which means that the hidden units are conditionally independent). However, the posterior is not independent because of the likelihood term which comes from the input vectors fed to the visible layer. [29] proposes that the explaining away in the hidden layer can be eliminated by creating a complementary prior having exactly the opposite correlations to those in the likelihood term originating from the data. This is done so that the product of the prior and the likelihood term yields us a factorial posterior.

The way data is generated in an infinite belief net as shown in Fig. 10; left panel and explained in [29] is the same as using an RBM to generate data. Notice that the alternating connections between hidden and visible layers with tied weights can be reduced down to a two-layer $(\boldsymbol{v}, \boldsymbol{h})$ RBM consisting of infinite walks of contrastive divergence. Finally, in both cases, the generation process converges at a stationary point also called as the equilibrium point of the Markov chain. The main contribution done by [29] was to put forward a greedy algorithm to make the DBN learn layer-by-layer.

First we train an RBM which models the data x. Let $R(\boldsymbol{c}^1|\boldsymbol{c}^0)$ be the posterior over $\boldsymbol{c}^1$ where $\boldsymbol{c}^0$ is the input data vector x. What we can achieve from this is an empirical distribution $\hat{d}^1$ over the layer $\boldsymbol{c}^1$ when we sample $\boldsymbol{c}^0$ from $\hat{d}$ [30].

$$\hat{d}^1(\boldsymbol{c}^1) = \sum_{\boldsymbol{c}^0} \hat{d}(\boldsymbol{c}^0)\, R(\boldsymbol{c}^1|\boldsymbol{c}^0) \qquad \textbf{(15)}$$

After training the RBM, we insert this RBM on top of the DBN where the number of layers is $(n+1)$. Thus, $R(\boldsymbol{c}^{n-1}|\boldsymbol{c}^n)$ corresponds to $D(\boldsymbol{c}^{n-1}|\boldsymbol{c}^n)$ where $D(.)$ is the posterior distribution associated with the DBN. Hence, we are effectively using $R(\boldsymbol{c}^{n-1}|\boldsymbol{c}^n)$ to be an approximation of the posterior $D(\boldsymbol{c}^{n-1}|\boldsymbol{c}^n)$. This eliminates the deepest hidden layer to have an absolutely independent prior. Using eqn. (15), we can write,

$$\hat{d}^n(\boldsymbol{c}^n) = \sum_{\boldsymbol{c}^{n-1}} \hat{d}^{n-1}(\boldsymbol{c}^{n-1}) R(\boldsymbol{c}^n|\boldsymbol{c}^{n-1}) \qquad \textbf{(16)}$$

Through eqn. (16) the samples $\boldsymbol{c}^{n-1}$ with distribution $\hat{d}^{n-1}$ are stochastically transformed to $\boldsymbol{c}^n$ with distribution $\hat{d}^n$. We can now sample in an unbiased manner from $\boldsymbol{c}^n$ and feed it downward stochastically by $R(\boldsymbol{c}^i|\boldsymbol{c}^{i-1})$. By doing so, all the lower hidden units acquire an approximated posterior of data x clamped at $\boldsymbol{c}^0$. Further, using mean-field approximation (as optionally proposed in [29]), we can approximate posteriors $D(\boldsymbol{c}^i|\boldsymbol{c}^0)$ by transforming each individual samples c_j^{i-1} from level $i-1$ where $j \in \{0, 1, \dots, m^i - 1\}$ as m^i is the number of units in layer i. Each sample can be transformed by their mean-field expected value E_j^{i-1} where,

$$E^i = \sigma(b_j^i + W^i E^{i-1}), \text{ where} \qquad \textbf{(17)}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \textbf{(18)}$$

Eqn. (18) is called the sigmoid function used in logistic models. W^i is the weight matrix of i^{th} layer, b_j^i are biases for unit j existing in the layer i. The general equation of deep belief networks put forward by [29] is as follows:

$$D(c^i|c^{i+1}) = \prod_{j=1}^{m^i} D(c_j^i|c^{i+1}), \qquad \textbf{(19)}$$

and

$$D(c_j^i = 1|c^{i+1}) = \sigma\left(b_j^i + \sum_{k=1}^{m^{i+1}} W_{kj}^i c_k^{i+1}\right) \qquad \textbf{(20)}$$

As they are models with a powerful feature extraction ability, DBNs have been employed for a large spectrum of different tasks viz. breast cancer classification [31], time-series forecasting [32], classifying audio from different sources (voice activity detection) [33], and through convolutional DBNs [34], hyperspectral spatial data classification [35], facial expression recognition through boosted DBNs [36], etc. The implementation for DBNs is demonstrated in Section 7.1.2.

4.3 Deep Boltzmann Machines

The main difference between DBMs [37] and DBNs is that in the former all the connections are undirected (see Fig. 11). DBMs are also used to capture hidden complex underlying features in the data making it suitable for tasks like speech and object recognition. DBMs, as opposed to DBNs, use an approximate inference procedure with an additional bottom-up pass initially to accelerate learning and incorporate top-down feedback which makes the DBM deal well with ambiguous inputs.

DBMs can also be pretrained layer-wise in a greedy fashion. Having a DBM with 3 layers, the pretraining is done by learning a stack of RBMs with the modification that the bottom-most vector of inputs and the top-most layer are doubled (see Fig. 12; left panel). This is done to respectively compensate for the scarce top-down input on $\boldsymbol{h}^1$ and the scarce bottom-up input on $\boldsymbol{h}^2$. Lastly, all the weights in the intermediate RBMs are doubled. These three components when composed together form a single deep Boltzmann machine (Fig. 12; right panel). An algorithm for the procedure may be written as:

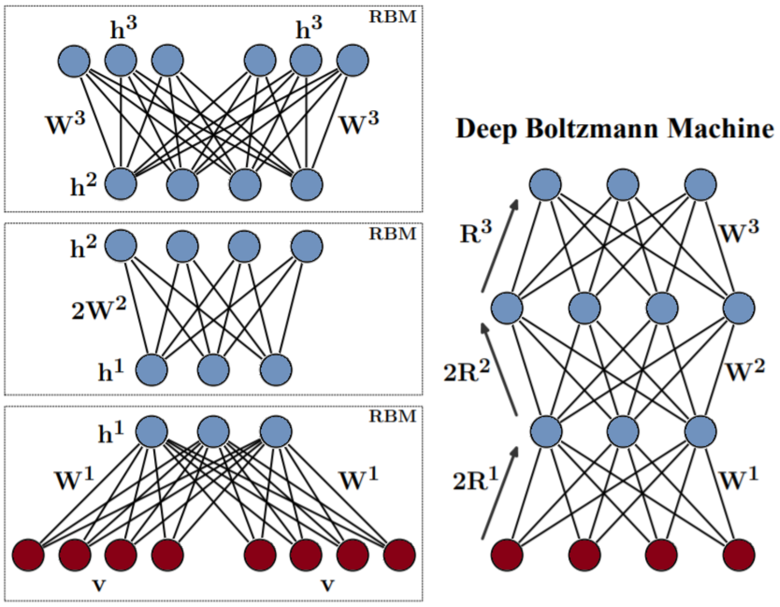


Fig. 12 (**Left**): Pretraining of DBM involves individual training of stacked RBMs with the lowest and topmost layers doubled and the intermediate weights doubled. (**Right**): The final structure done through composition of the weights obtained through modified pretraining process as described on the left.

1. Duplicate visible vector and tie $\boldsymbol{W}^1$. Fit this RBM comprising $\boldsymbol{v}$ and $\boldsymbol{h}$ to the data.
2. Freeze $\boldsymbol{W}^1$. Use 1st layer of features $\boldsymbol{h}^1$ through $P(\boldsymbol{h}^1|\boldsymbol{v}, 2\boldsymbol{W}^1)$ and fit the intermediate RBM having weights vector $2\boldsymbol{W}^2$.
3. Freeze $\boldsymbol{W}^2$. Use 2nd layer of features $\boldsymbol{h}^2$ through $P(\boldsymbol{h}^2|\boldsymbol{v}, 2\boldsymbol{W}^2)$ and fit the intermediate RBM having weights vector $2\boldsymbol{W}^3$.
4. Duplicate top-level RBM's hidden vector $\boldsymbol{h}^3$ and tie $\boldsymbol{W}^3$.
5. Utilize $\boldsymbol{W}^1, \boldsymbol{W}^2, \boldsymbol{W}^3$ to compose a DBM.

The traditional process of training BMs uses random initialization to approximate gradients of the likelihood function [38] for the input data which is not the quickest approach. To tackle this, [37] proposed a variational technique making use of mean-field inference to approximate expectations correlated to data with a Markov chain based estimation procedure to estimate the model's expected required statistics. This procedure involves Markov chains initializing the weights to appropriate values (as in solving the mean-field fixed point equations for each update in the parameters of the DBM) to facilitate joint learning of all layers. However, this is very expensive when compared to the pretraining of DBNs where inference is done through a single bottom-up pass. Thus, inference of DBMs is accelerated, as proposed in [39], with the use of recognition weights. The set of recognition weights $\{\boldsymbol{R}^1, \boldsymbol{R}^2, \boldsymbol{R}^3\}$ are initialized to the weights $\{\boldsymbol{W}^1, \boldsymbol{W}^2, \boldsymbol{W}^3\}$ which are obtained after the greedy pretraining process. With an input clamped on $\boldsymbol{v}$ the recognition weights are applied to reconstruct $\boldsymbol{v} = \{\boldsymbol{v}^1, \boldsymbol{v}^2, \boldsymbol{v}^3\}$ of the approximating posterior distribution which is fully factorized:

$$D^{rec}(\boldsymbol{h}|\boldsymbol{v};\, \boldsymbol{\mu}) = \prod_{p=1}^{m^1}\prod_{q=1}^{m^2}\prod_{r=1}^{m^3} d^{rec}(h_p^1)\, d^{rec}(h_q^2)\, d^{rec}(h_r^3), \tag{21}$$

where, $d^{rec}(h_i^l = 1) = v_i^l$ for $l = \{1, 2, 3\}$ and $\boldsymbol{\mu} = \{\mu^{\mathbf{1}}, \mu^{\mathbf{2}}, \mu^{\mathbf{3}}\}$ are the mean field parameters.

Each hidden layer is activated with a bottom-up pass by:

$$v_p^1 = \sigma\left(\sum_{i=1}^{m^0} 2R_{ij}^1 v_i\right), \tag{22}$$

$$v_q^2 = \sigma\left(\sum_{p=1}^{m^1} 2R_{pk}^2 v_p\right), \tag{23}$$

$$v_r^3 = \sigma\left(\sum_{q=1}^{m^2} R_{km}^3 v_k\right) \tag{24}$$

As noticed in eqns. (22) and (23) the recognition weights are doubled to compensate for the lack of top-down feedback (as also shown in Fig. 12; right panel). However, in the top layer there is no top-down feedback, hence the recognition weights are not doubled. After this step, k iterations of mean-field approximation are applied which initialize at $\boldsymbol{\mu} = \boldsymbol{v}$. These mean-field parameters thus obtained are used in the training updates for the DBM. Finally, the recognition weights are updated in such a manner that the Kullback-Leibler divergence (defined by eqn. (7)) between mean-field posterior $D^{mf}(\boldsymbol{h}|\mathbf{v};\, \boldsymbol{\mu})$ and factorial posterior $D^{rec}(\boldsymbol{h}|\mathbf{v}; \boldsymbol{v})$ is minimized as illustrated in eqn. (25).

$$KL(D^{mf}(\boldsymbol{h}|\mathbf{v};\, \boldsymbol{\mu})||D^{rec}(\boldsymbol{h}|\mathbf{v}; \boldsymbol{v})) = -\sum_i \mu_i log v_i - \sum_i (1 - \mu_i)\log(1 - v_i) + C\ (const.) \tag{25}$$

Further on, fine-tuning of the DBM may be done discriminatively (in a supervised manner) by feeding a few samples of labelled data. DBMs have been applied on topic modelling to outperform LDA [40], multimodal learning [41], spoken query detection [42], state-of-the-art 3D model recognition [43, 44], face modelling [45], etc. The implementation for DBMs is demonstrated in Section 7.1.3.

5. Variational Autoencoders

In this section, we first describe what autoencoders are in 5.1 *Autoencoders*, and then move on to 5.2 *Variational Autoencoders* to describe how these models generate data.

5.1 Autoencoders

Normal autoencoders comprise three layers (Fig. 13). The input layer, where $\{x^i\}_{i=1}^N \in \boldsymbol{X}$, the middle layer also known as the *coding* or the *bottleneck* layer $\boldsymbol{Z}$, and lastly the output layer $\hat{X}$.

Autoencoders are an unsupervised approach to learning lower dimensional feature representations from unlabelled data. The inputs are encoded into feature extracted representations and stored in $\boldsymbol{Z}$ through weights. Similarly, an output similar to $\boldsymbol{X}$ at $\widehat{\boldsymbol{X}}$ is generated after decoding of vector $\boldsymbol{Z}$. There is a mapping function for encoding which may be stated as

$$\boldsymbol{Z} = f(\boldsymbol{WX} + b) \tag{26}$$

where b is the bias and $\boldsymbol{W}$ is the vector of weights. The loss is calculated through a simple L2 loss function at the end after one epoch,

$$L = ||x - \hat{x}||^2 \tag{27}$$

After calculating the loss, the error is backpropagated through the network and the weights are adjusted like in a normal artificial neural network. Autoencoders can be used to initialize supervised classification models where the decoder is replaced with a classifier and this classifier runs on the extracted feature vector $\boldsymbol{Z}$ to classify only based on the important encoded features. Autoencoders are mainly thought to be used for compression tasks as the cardinality of vector $\boldsymbol{Z}$ is very low as compared to that of vector $\boldsymbol{X}$ and hence data is converted into a compressed format with all the important features still intact. Autoencoders however, are used for more purposes such as collaborative filtering [46], hashing for fast image search through binary autoencoders [47], and audio generation [48], etc.

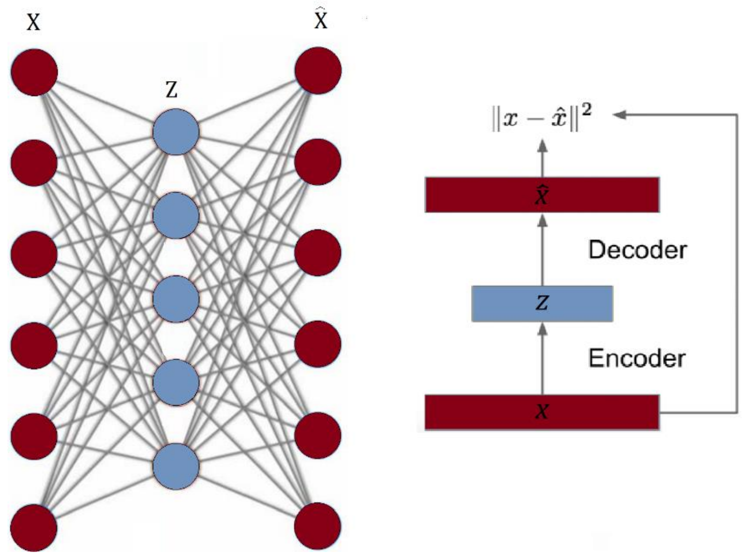


Fig. 13 (**Left**): An autoencoder network. (**Right**): Training process of the autoencoder with an L2 loss function $||x - \hat{x}||^2$.

5.2 Variational Autoencoders

In order to generate data, we must be allowed to sample from the autoencoder model. We first assume that our data $\{x^i\}_{i=1}^N$ is generated by a true prior latent distribution $\boldsymbol{z}$ (which is assumed to be a Gaussian) given by $p_\theta(\boldsymbol{z})$ where θ are the parameters of our model. To generate data, we must now sample from $\boldsymbol{x}$ by the true conditional $p_\theta(\boldsymbol{x} \mid z^i)$ and estimate the true parameters of θ. This conditional can be represented by a neural network. Generally, for training generative models we maximize the likelihood of training data,

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{z}) p_\theta(\boldsymbol{x}|\boldsymbol{z}) dz \tag{28}$$

The problem with eqn. (28) is that the conditional $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ over the integral is intractable. Moreover, the posterior density given by,

$$p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{x})} \tag{29}$$

is also intractable due to the denominator $p_\theta(\boldsymbol{x})$ which we know from eqn. (28) is intractable. As a solution, we can approximate $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ through an inference network $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ which allows us to derive a tractable lower bound which can be maximized by proper optimization. Fig. 14 describes this new probabilistic model where $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ may be called the generator network. The logarithm of the data likelihood can then be expressed as the expectation $\mathbb{E}$ with respect to $\boldsymbol{z}$ sampled from $q_\varphi(\boldsymbol{z}|x^i)$.

$$\log p_\theta(x^i) = \mathbb{E}_{\boldsymbol{z}\sim q_\varphi(\boldsymbol{z}|x^i)}[\log p_\theta(x^i)] \qquad \textbf{(30)}$$

$p_\theta(x^i)$ is independent of $\boldsymbol{z}$

$$= \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{p_\theta(x^i|\boldsymbol{z})p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{z}|x^i)}\right], \qquad \textbf{(31)}$$

from Bayes' rule

Multiplying with a constant, we get,

$$\mathbb{E}_{\boldsymbol{z}}\left[\log \frac{p_\theta(x^i|\boldsymbol{z})p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{z}|x^i)}\frac{q_\varphi(\boldsymbol{z}|x^i)}{q_\varphi(\boldsymbol{z}|x^i)}\right] \qquad \textbf{(32)}$$

$$\begin{aligned} &= \mathbb{E}_{\boldsymbol{z}}[\log p_\theta(x^i|\boldsymbol{z})] \\ &- \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{q_\varphi(\boldsymbol{z}|x^i)}{p_\theta(\boldsymbol{z})}\right] \\ &+ \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{q_\varphi(\boldsymbol{z}|x^i)}{p_\theta(\boldsymbol{z}|x^i)}\right] \end{aligned} \qquad \textbf{(33)}$$

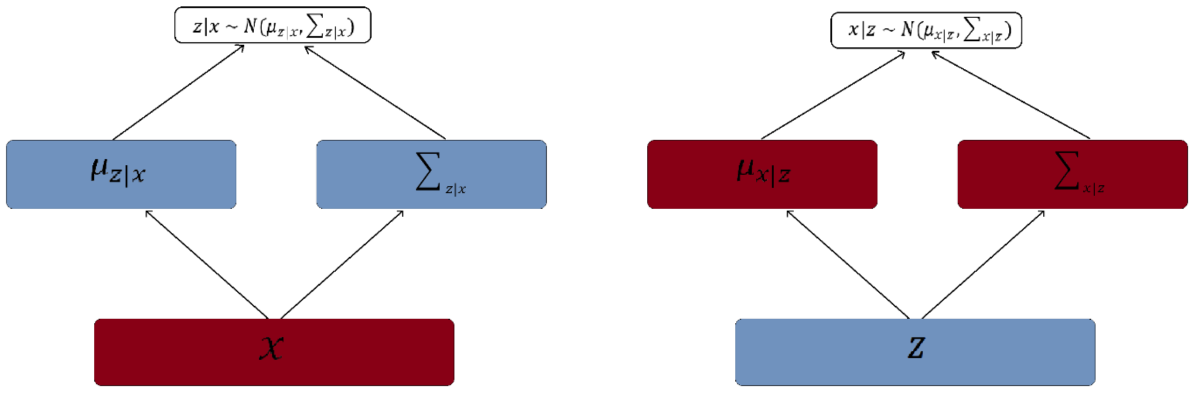

$z|x \sim N(\mu_{z|x}, \sum_{z|x})$
$\mu_{z|x}$
$\sum_{z|x}$
x
$x|z \sim N(\mu_{x|z}, \sum_{x|z})$
$\mu_{x|z}$
$\sum_{x|z}$
z

Fig. 14 (**Left**): The inference network represented by $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ which outputs mean and diagonal covariance vectors of $\boldsymbol{z}|\boldsymbol{x}$ from which we sample $\boldsymbol{z}|\boldsymbol{x}$. (**Right**): The generator network (or decoder) represented by $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ which also output the mean and diagonal covariance vectors of $\boldsymbol{x}|\boldsymbol{z}$ from which the distribution $\boldsymbol{x}|\boldsymbol{z}$ can be sampled.

$$\begin{aligned} &= \mathbb{E}_{\boldsymbol{z}}[\log p_\theta(x^i|\boldsymbol{z})] \\ &- KL(q_\varphi(\boldsymbol{z}|x^i)||p_\theta(\boldsymbol{z})) \\ &+ KL(q_\varphi(\boldsymbol{z}|x^i)||p_\theta(\boldsymbol{z}|x^i)) \end{aligned} \qquad \textbf{(34)}$$

As we can see, term III comprises $p_\theta(\boldsymbol{z}|x^i)$ which we know is intractable. However, we also know that the KL divergence is always ≥ 0 which is useful for optimizing the likelihood. Also, term I can be estimated by sampling that is differentiable when we use the reparameterization trick described in [49], term II is composed of two Gaussians and their KL divergence gives us a closed-form solution. To maximize the likelihood we have to maximize I and minimize II. Term I describes the reconstruction of input data which requires the expectation to be high (or reconstruct the data very well), whereas term II needs to be minimized, which essentially means the posterior distribution must be as similar as possible to the prior, as by doing this, the Kullback-Leibler divergence is minimized.

We define a tractable lower bound (because term III is ≥ 0), $\varepsilon(x^i, \theta, \varphi)$ whose gradient can be acquired and optimized,

$$\varepsilon(x^i, \theta, \varphi) = \mathbb{E}_{\boldsymbol{z}}\left[\log p_\theta(x^i|\boldsymbol{z})\right] - KL(q_\varphi(\boldsymbol{z}|x^i)||p_\theta(\boldsymbol{z})) \quad \textbf{(35)}$$

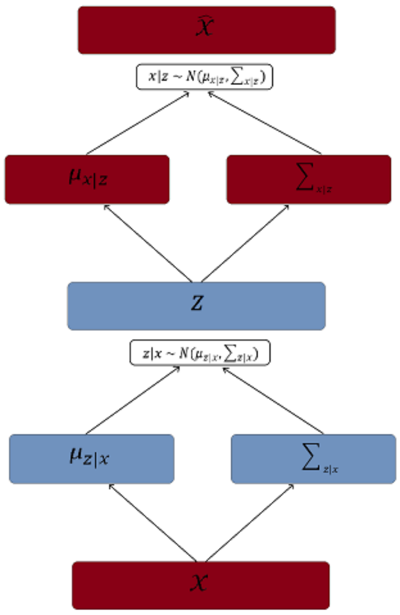
Therefore, we get a variational lower bound,

$$\varepsilon(x^i, \theta, \varphi) \leq \log p_\theta(x^i) \quad \textbf{(36)}$$

While training, we attempt to estimate the parameters θ' and φ' by maximizing $\varepsilon(x^i, \theta, \varphi)$ as

$$\theta', \varphi' = \arg\max \sum_{i=1}^{N} \varepsilon(x^i, \theta, \varphi) \quad \textbf{(37)}$$

Fig. 15 shows the entire variational autoencoder network.



$\hat{X}$
$x|z \sim N(\mu_{x|z}, \sum_{x|z})$
$\mu_{x|z}$
$\sum_{x|z}$
z
$z|x \sim N(\mu_{z|x}, \sum_{z|x})$
$\mu_{z|x}$
$\sum_{z|x}$
x

Fig. 15. The Variational Autoencoder (VAE) when all the parts are pieced together. From the bottom-up, it comprises of the inference network and the generator network which gives us the output $\hat{X}$.

VAEs have been used for trajectory prediction from static images [50], collaborative filtering [51], recreation of music [52], modelling frame-wise spectral envelopes in speech processing [53], speech emotion classification [54], molecule generation [55], etc. Later, in Section 7.1.4, we demonstrate how CVAEs (Convolutional Variational Autoencoders) can be applied to generate artificial images from a base distribution of similar images.

6. Generative Adversarial Networks

VAEs generate data similar to the original data to an extent, however, they are not the most accurate. In the case of generating images, one can notice blurriness in the generated images. This is overcome by GANs [56], proposed by Goodfellow *et al.* (2014), which are the most recent addition to the modern generative models and also achieve high accuracy in generating data. GANs offer a solution of training generative models without the usual procedure of maximizing a log likelihood (as we saw earlier with VAEs) which are usually intractable and required numerous approximations. Neither do GANs require any Markov chains, as in the case of Boltzmann machines.

In this section, we shall first discuss GANs and the basic building blocks that they comprise along with their ideology of operation. Then, we discuss some of the most relevant and popular derivatives of GANs (although, in practice, there are literally more than a thousand derivatives invented so far) viz. 6.1 *Deep Convolutional GANs (DCGANs)*, 6.2 *Fully Connected and Convolutional GANs (FCC-GANs)*, 6.3 *Conditional GANs (CGANs)*, 6.4 *Stack GANs (SGAN)*.

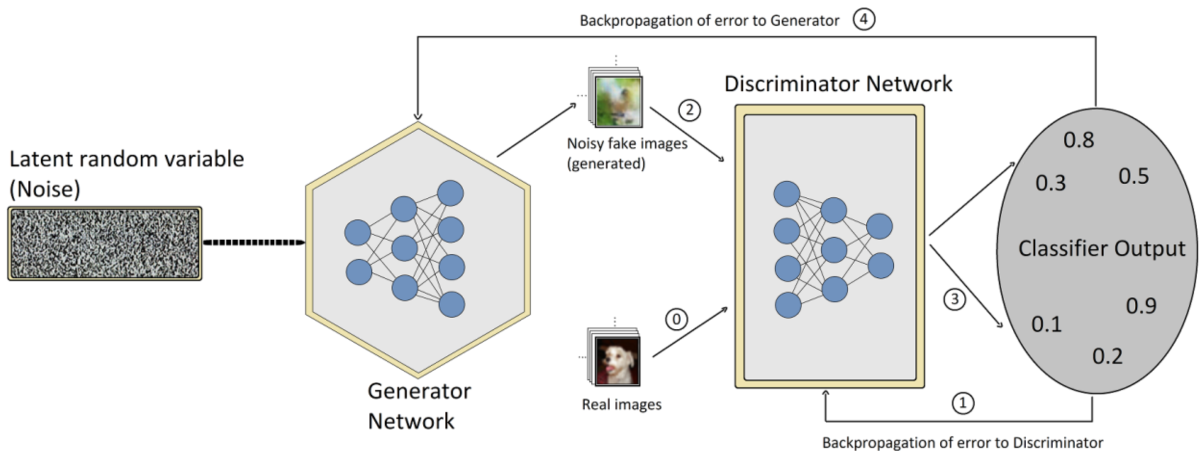
GANs have two components namely the *discriminator* and the *generator*. Both these components work in tandem (contrary to the term ‘adversarial’ – which gives the idea of a competitive environment) and learn features together rather than one of them being pretrained. Fig. 16 describes the whole training process and the ideology of operation in a pedagogical manner.

Formally, let us denote the distribution of generator G by p_G over genuine data $\boldsymbol{x}$. We define a prior of input noise (latent random variable) $p_{\boldsymbol{z}}(\boldsymbol{z})$. Note that G is a differentiable function as it operates on non-discrete data $\boldsymbol{z}$

with parameters θ_G whose data space is represented by $G_{\theta_G}(\boldsymbol{z})$. Similarly, we represent the discriminator D's data space as $D_{\theta_D}(\boldsymbol{x})$ having parameters θ_D which is the probability that the data came from genuine data $\boldsymbol{x}$ and isn't fake (that is, not from p_G). One can think of D as a simple Convolutional Neural Network (CNN) which discriminates between real and fake images and outputs softmax probabilities between 0 and 1 regarding whether the data is real or fake. The general equation of GANs is given by a value function $f(G, D)$ or a minimax objective function given by:

$$\begin{aligned} & min_{\theta_G}\, min_{\theta_D}\, f(D, G) \\ & = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})} \log D_{\theta_D}(\boldsymbol{x}) \\ & + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})} \log(1 \\ & - D_{\theta_D}(G_{\theta_G}(\boldsymbol{z}))) \end{aligned} \qquad \textbf{(38)}$$

D_{θ_D} tends to maximize objective by increasing $D_{\theta_D}(\boldsymbol{x}) \approx 1$ and decreasing $D_{\theta_D}(G_{\theta_G}(\boldsymbol{z})) \approx 0$, while on the other hand, G_{θ_G} tends to minimize objective by increasing $D_{\theta_D}(G_{\theta_G}(\boldsymbol{z})) \approx 1$. In other words, D attempts to discriminate more properly between real and fake data, and conversely, G attempts to make D output higher values close to 1 for generated data in order to fool D. This minimax game terminates at a "saddle point" (the Nash equilibrium) where $f(D, G)$ is minimum with respect to G's strategy and maximum with respect to D's strategy. The Nash equilibrium is a solution concept in game theory which involves two players competing each other (non-cooperative) where each player knows the equilibrium strategies of other players, with the given constraint that no player can reach their equilibrium by changing only their own strategy. Further, the training of GANs



Backpropagation of error to Generator ④
Latent random variable (Noise)
Generator Network
Noisy fake images (generated)
②
Real images
⓪
Discriminator Network
③
Classifier Output
0.8
0.3
0.5
0.9
0.1
0.2
①
Backpropagation of error to Discriminator

Fig. 16 Generative Adversarial Network. The working procedure is as follows:
Step 0: The discriminator D (a multilayer perceptron (MLP)) is trained first by feeding it real images.
Step 1: The error is backpropagated through D and its weights are adjusted.
Step 2: D is trained further by feeding it some random primitive stage generations by generator G (note that G takes noise input and is also an MLP).
Step 3: D outputs values close to 0 because of pretraining of real images.
Step 4: The output of D is subtracted by 1 (to train G appropriately) and backpropagated through G and its weights are adjusted.
Further, G generates images with noise input again with readjusted weights, resulting in more realistic looking images. These images are fed to D along with real images and we get outputs which are backpropagated through D so it can better discriminate next time. The outputs for fake images are also backpropagated through G for the betterment of quality of generated images. These steps are repeated for many epochs until D cannot discriminate between generated and real images.

alternates between k steps of optimizing D and one step of optimizing G as follows:

$$argmax_{\theta_D}[\mathbb{E}_{\boldsymbol{x}\sim p_{data}(x)} \log D_{\theta_D}(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})} \log(1 - D_{\theta_D}(G_{\theta_G}(\boldsymbol{z})))], \quad \textbf{(39)}$$

and,

$$argmin_{\theta_G}[\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})} \log(1 - D_{\theta_D}(G_{\theta_G}(\boldsymbol{z})))] \quad \textbf{(40)}$$

Eqn. (39) describes the gradient ascent on D while eqn. (40) describes the gradient descent on G.

However, as reported in [56], in practice, initially it may be difficult to train GANs due to insufficient gradient for G. This happens because initially when G is weak, it is easy for D to detect generated data with high confidence thus making $\log(1 - D_{\theta_D}(G_{\theta_G}(\boldsymbol{z})))$ saturate. Instead, we choose to maximize $\log D_{\theta_D}(G_{\theta_G}(\boldsymbol{z}))$,

$$argmax_{\theta_G}[\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})} \log D_{\theta_D}(G_{\theta_G}(\boldsymbol{z}))] \quad \textbf{(41)}$$

This deals with the problem of a flat gradient early on in the training for G by increasing the gradient which helps G to train properly. Intuitively, this modification serves the same purpose as before when G was tending to minimize the objective. We now discuss a few major contributions done on the field of GANs.

6.1 Deep Convolutional GANs (DCGAN)

DCGANs [57] were proposed with some structural changes to the original GANs which were unstable to train in the sense that the network could collapse after certain epochs, where the generator produced nonsensical outputs. There were five significant modifications done, which we shall discuss now.

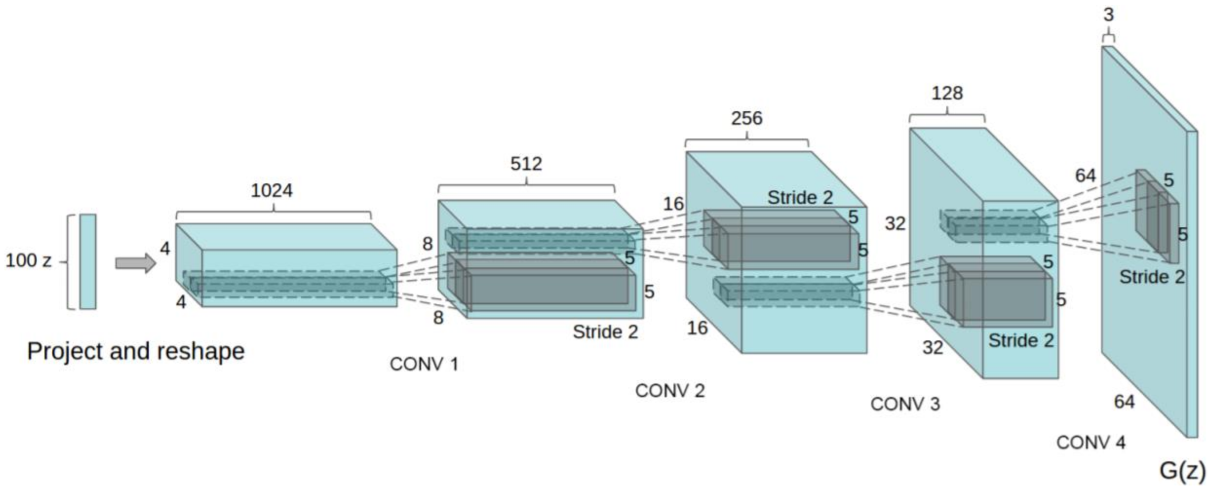


Fig. 17 Radford *et al.* The generator of DCGAN with four sequential fractionally strided convolutional layers.

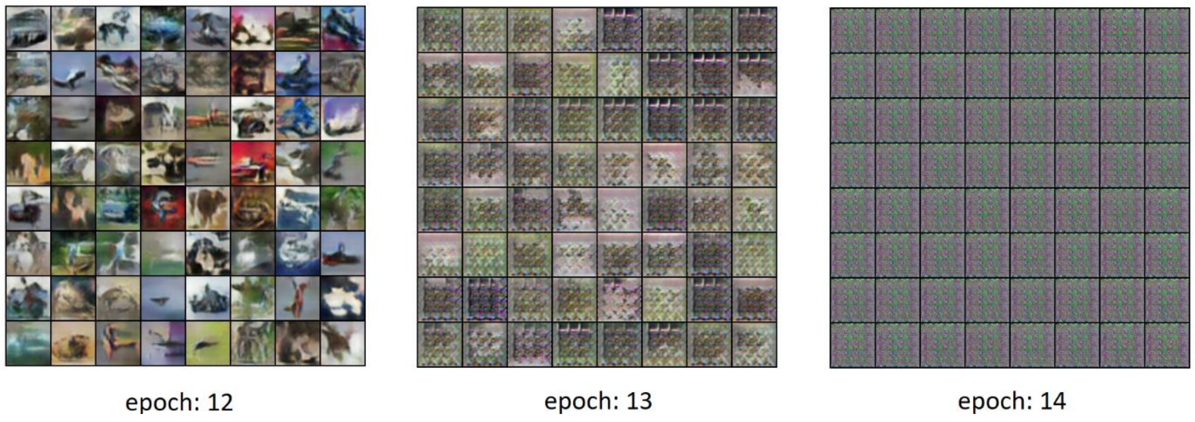


Fig. 18 Collapse of a DCGAN to a single point, particularly visible after 12 epochs with certain parameters on the 32x32 CIFAR-10 [59] dataset.

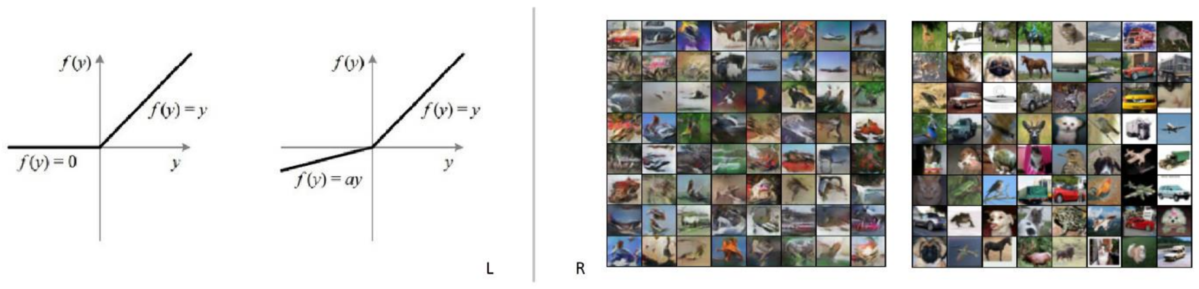


Fig. 19 (**Left**): Rectified Linear Unit (reLU) activation function on the left and leaky reLU on the right. (**Right**): DCGAN generated samples after 11 epochs on 32x32 CIFAR-10 dataset on the left and real samples on the right.

The first change was to use only convolutional layers, instead of the traditional alternating convolution and max-pooling layers followed by full connection with ANNs. This was inspired by [58] which found that a convolutional layer with augmented stride can replace a max-pooling layer without any loss in accuracy. This allowed the generator to learn its own spatial upsampling (through transposed convolutional layers or fractionally strided convolutional layers) and similarly for the discriminator to learn its own spatial downsampling.

Secondly, the full connection on top of the highest convolutional features was eliminated. The highest convolutional features were connected to the input of the generator and the output of the discriminator. Generator G takes a noise distribution $\boldsymbol{z}$ as input which can be called a full connection (because it's only a matrix multiplication) and as for discriminator D, the last convolutional layer is flattened to be fed into a single sigmoid output. Fig. 17 sheds light on the DCGAN architecture of G. DCGANs have been implemented in Section 7.1.5.

6.2 Fully Connected and Convolutional GANs (FCC-GAN)

[63] puts forward a fully connected and convolutional GAN and argues that having full connection along with max pooling layers in both components of the GAN can outperform the traditional DCGANs. FCC-GAN demonstrates improvement over the traditional DCGAN in sample quality, learning speed and stability.

The strided convolution layers in D are replaced by pooling layers having unit stride convolution (Fig. 20).

The FC network of the generator maps noise to intermediate image features which has many advantages. It allows learning of essential non-spatial mapping from the noise vector to intermediate features. It also captures the underlying relationship between different noise vectors that should be mapped to similar features of the same class of images. Finally, it solves the problem of shared weights in the DCGAN architecture which didn't allow the generation of slight variations with respect to different spatial zones in the same convolution filter, thus giving more realistic results.

The problem with the discriminator of the DCGAN is that it extracts high dimensional features from an input image which makes it easier to make a decision boundary between real and fake. This implies that the discriminator loss converges quickly to small values. If D becomes too powerful as compared to G, then the learning gradients for G would become too weak or flat and may even vanish completely. Secondly, the distance between the decision boundary and category regions in high dimensional space may be high, and thus the gradients may point to random directions which is not appropriate to train G. FCC-GANs solve the above problems by reducing the high dimensional features to lower dimensions, thus bringing the decision boundary closer to category regions, and since it is harder to build the decision boundary in lower dimensions, D cannot easily discriminate between real and fake data points. This in turn slows down the convergence rate.

Finally, the last advantage of FCC-GAN over traditional DCGANs is the average pooling in D which boosts performance and acts as a regularization in feature extraction process which is very useful in deep fully connected networks. Fig. 21 shows results of DCGAN (denoted by conventional CNN), FCC-GAN-S (strided convolution), and FCC-GAN-P (pooling).

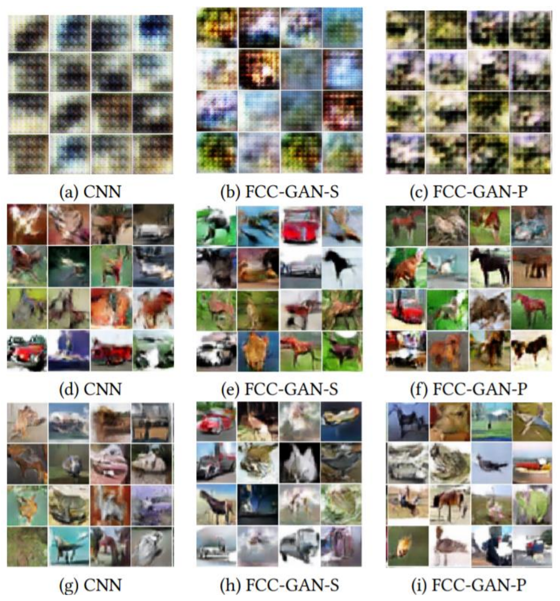


Fig. 21 Barua *et al.* Image generation performance on 32x32 CIFAR-10 image dataset, compared between CNN, FCC-GAN-S and FCC-GAN-P after 1 epoch (a-c), 35 epochs (d-f), and 150 epochs (g-i).

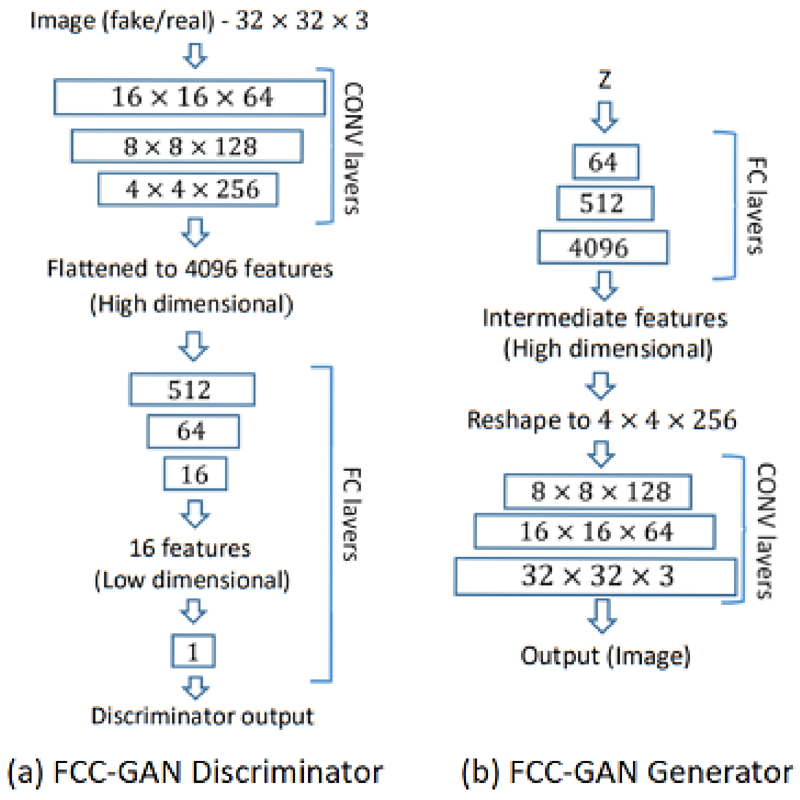


Fig. 20 Barua *et al.* (a) The FCC-GAN discriminator having multiple deep fully connected layers mapping high dimensional features extracted by convolutional layers to a lower dimensional space before max pooling with unit

stride convolution, and (b) the FCC-GAN generator having multiple deep fully connected layers before convolutional layers to convert low dimensional noise distribution to a high dimensional representation of features of an image.

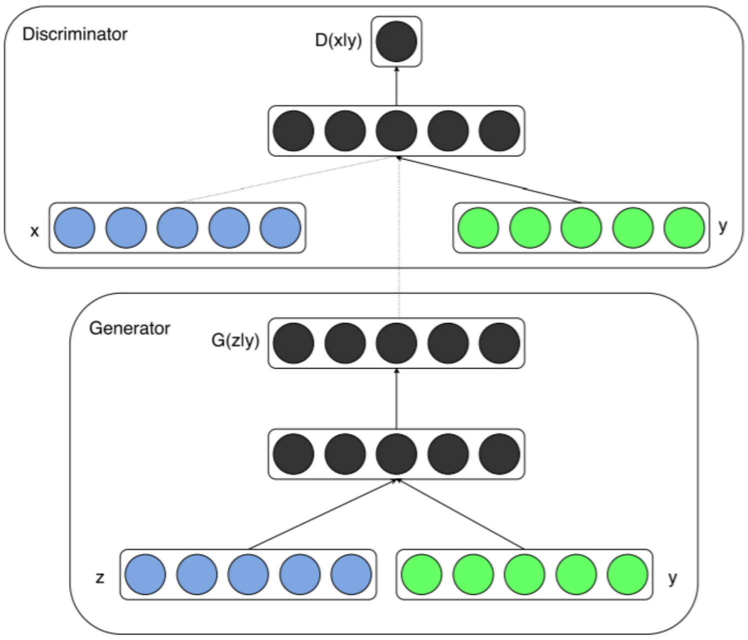


Fig. 22 Mirza *et al.* The conditional generative adversarial network with additional data $\boldsymbol{y}$, ground truth data $\boldsymbol{x}$ and latent hidden noise vector $\boldsymbol{z}$.

6.3 Conditional GANs (CGAN)

CGANs [64] provide a control over the data being generated by conditioning it on additional data (which could be class labels or associated data from a different modality). Let this additional data be denoted by $\boldsymbol{y}$. We feed $\boldsymbol{y}$ to both G and D to condition them both as shown in Fig. 22. The general equation of CGANs is as follows:

$$min_{\theta_G} max_{\theta_D} f(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})} \log D_{\theta_D}(\boldsymbol{x}|\boldsymbol{y}) + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})} \log(1 - D_{\theta_D}(G_{\theta_G}(\boldsymbol{z}|\boldsymbol{y}))) \quad \textbf{(42)}$$

CGANs have been used for many purposes such as face generation [65], transforming face by age [66], removal of rain from images [67], speech enhancement [68], classifying spoken language [69], however, they do not promise to always outperform their non-conditional counter parts. With hyperparameter tuning and changes to the architecture it may be possible to do so, yet CGANs on their own are not always more efficient.

6.4 Stack GANs (SGAN)

Stacked GANs [70] comprise stacks of top-down generators, each of which generate lower level representations while being conditioned on corresponding higher level representations of the data. Similarly, in a bottom-up manner, a series of deep neural networks (DNNs) are connected which form the encoder. The *adversarial loss* ω^{adv} is introduced which is given by each of the intermediate representations of the units (DNNs) in the encoder which forces these intermediate representations to lie on the manifold of the DNN's representation space. There are two more types of losses, namely, *conditional loss* ω^{cond} and *entropy loss* ω^{ent}. Conditional loss arises due to the conditional nature of the generators used which are conditioned on class labels $\boldsymbol{y}$ with noise input $\boldsymbol{z}$ which forces the generators to better utilize the high level conditional information. Entropy loss forces the generators to make the generated representations more varied and diverse. Fig. 23 shows the architecture of SGAN in its entirety.

The encoder E is pretrained initially for classification. Let there be a stack of bottom up deterministic non-linear mappings $h_{i+1} = E_i(h_i),\ i \in \{0, 1, \dots, N-1\}$ where N is the total number of stacks and h_i $(i \neq 0, N\}$ are the intermediate representations between each encoding unit (it is worthy of mention that $h_0 = \boldsymbol{x}$ (original data) and $h_N = \boldsymbol{y}$ which are the predictions). E_i refers to the i^{th} unit of E which is made up of neural network layers like convolutions, pooling, etc.

Each G_i then takes the higher level feature and the noise vector to produce a lower level feature $\hat{h}_i$. As described in Fig. 23, the generators G_i, whose function is to invert a bottom-up mapping of the corresponding E_i, is trained independently first with the conditional input provided by E as $\hat{h}_i = G_i(h_{i+1}, z_i)$, and then jointly when each G_i (except the top-most, which takes input from the prediction $\boldsymbol{y}$) gets input from upper generators as $\hat{h}_i = G_i(\hat{h}_{i+1}, z_i)$. All of the loss functions described henceforth are for independent training which can be represented the same way for joint training by replacing h_{i+1} with $\hat{h}_{i+1}$ wherever applicable. The total loss is calculated as the linear combination of the three losses described earlier for each G_i as:

$$\omega_{G_i} = C_1\omega_{G_i}^{adv} + C_2\omega_{G_i}^{cond} + C_3\omega_{G_i}^{ent}, \qquad \textbf{(43)}$$

where C_1, C_2 and C_3 are the weights corresponding to each of the respective losses, however, these weights are not learnt. [70] specifies that in practical use, these weights are set in such a magnitude that all the losses fall in similar scales. For each G_i there exists a corresponding discriminator D_i that judges the generated sample $\hat{h}_i$ based on real representations given by h_i whose loss function may be given as:

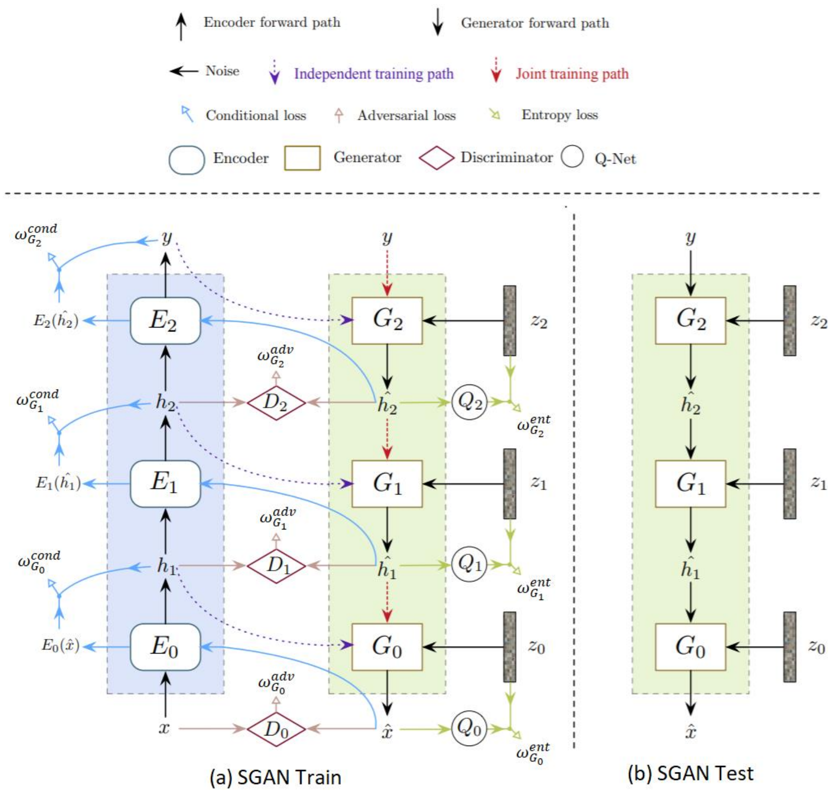


Fig. 23 Huang *et al.* Stack GAN model. (a) The training procedure where each generator G_i is trained individually conditioned on the input given by the encoder and then jointly when receiving input from other upper generators.

(b) While testing, $\hat{\boldsymbol{x}}$ is the final generated data which is a result of stacked top-down generators sequentially generating features one-by-one.

$$\omega_{D_i} = \mathbb{E}_{h_i \sim P_{data,E}}[-\log D_i(h_i)] + \mathbb{E}_{z_i \sim P_{z_i},\ h_{i+1} \sim P_{data,E}}[-\log(1 - D_i(G_i(h_{i+1}, z_i)))] \quad \textbf{(44)}$$

Each G_i's adversarial loss is given by the equation,

$$\omega_{G_i}^{adv} = \mathbb{E}_{h_{i+1} \sim P_{data,E},\ z_i \sim P_{z_i}}[-\log(D_i(G_i(h_{i+1}, z_i)))] \quad \textbf{(45)}$$

Conditional loss ω^{cond} serves the purpose of preventing generators G_i to generate data from the lower level representation inputs $\hat{h}_i$ by completely ignoring high level representations h_{i+1}, which shouldn't be the case as G_i is supposed to be conditioned on h_{i+1}. This is a form of regularization applied on G_i. The generated low-level distributions $\hat{h}_i$ are fed back to corresponding encoder unit E_i to compute *recovered higher-level representations*. ω^{cond} is used to force these recovered representations to be close (in Euclidean distance defined by f) to conditional representations as,

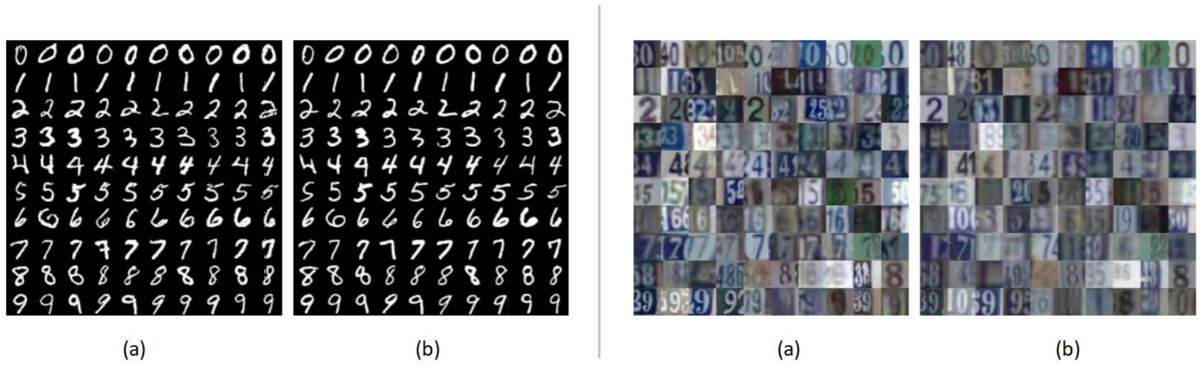


Fig. 24 Huang *et al.* **Left**: (a) SGAN samples conditioned on class labels, (b) ground truth (real) images from MNIST dataset. **Right:** (a) SGAN samples on conditioned on class labels, (b) ground truth (real) images from SVHN dataset.

$$\omega_{G_i}^{cond} = \mathbb{E}_{h_{i+1} \sim P_{data,E},\ z_i \sim P_{z_i}}[f(E_i(G_i(h_{i+1}, z_i)), h_{i+1})] \quad \textbf{(46)}$$

Another problem that arises with conditional loss is that G_i tends to ignore the latent noise distribution $\boldsymbol{z}$ while determining $\hat{h}_i$ conditioned on h_{i+1}. To solve this, it is necessary that the generated representations $\hat{h}_i$ be sufficiently diverse being conditioned on h_{i+1} given by the conditional entropy function $H(\hat{h}_i | h_{i+1})$. Higher entropy means higher diversity, thus desirably one would want to maximize H, but it is intractable. [70] proposes a variational lower bound on H where an auxiliary distribution $Q_i(z_i|\hat{h}_i)$ is used to approximate the true posterior $P_i(z_i|\hat{h}_i)$ through the entropy loss $\omega_{G_i}^{ent}$ given by,

$$\omega_{G_i}^{ent} = \mathbb{E}_{z_i \sim P_{z_i}} \mathbb{E}_{\hat{h}_i \sim G_i(\hat{h}_i|z_i)}[-\log Q_i(z_i|\hat{h}_i)] \quad \textbf{(47)}$$

$\omega_{G_i}^{ent}$ is minimized to maximize the variational lower bound for $H(\hat{h}_i|h_{i+1})$. Q_i is a DNN that estimates the posterior distribution z_i conditioned on $\hat{h}_i$ which estimates only the posterior mean (which is treated to be a diagonal Gaussian distribution) which implies that $\omega_{G_i}^{ent}$ becomes similar to a Euclidean reconstruction error. G_i and Q_i are updated in each iteration to minimize $\omega_{G_i}^{ent}$.

One thing to note is that G_i of the SGAN generate distributions conditioned on class labels $\boldsymbol{y}$ as,

$$P_G(\hat{\boldsymbol{x}}|\boldsymbol{y}) = P_G(\hat{h}_0|\hat{h}_N) \propto P_G(\hat{h}_0, \hat{h}_1, \dots, \hat{h}_{N-1}|\hat{h}_N) = \prod_{i=0}^{N-1} P_{G_i}(\hat{h}_i|\hat{h}_{i+1}) \quad \textbf{(48)}$$

SGAN factorizes $H(x)$ into smaller conditional entropy terms,

$$H(\boldsymbol{x}) = H(h_0, h_1, \dots, h_N) = \sum_{i=0}^{N-1} H(h_i|h_{i+1}) + H(\boldsymbol{y}) \quad \textbf{(49)}$$

SGANs achieve higher sample quality than the traditional DCGANs with either independent or joint training separately. Fig. 24 shows some SGAN generated images.

7. Analysis and Discussion

Table 1 describes the fields in which all the models discussed in this paper are used in. However, while putting together this survey, we noticed that there are common fields amongst two entirely different models that they are applied on. For example, we have seen that LDA is mainly meant for topic modelling, and still, DBNs have outperformed LDA in the same field [40]. We saw that HMMs were good for pattern recognition like the case of speech

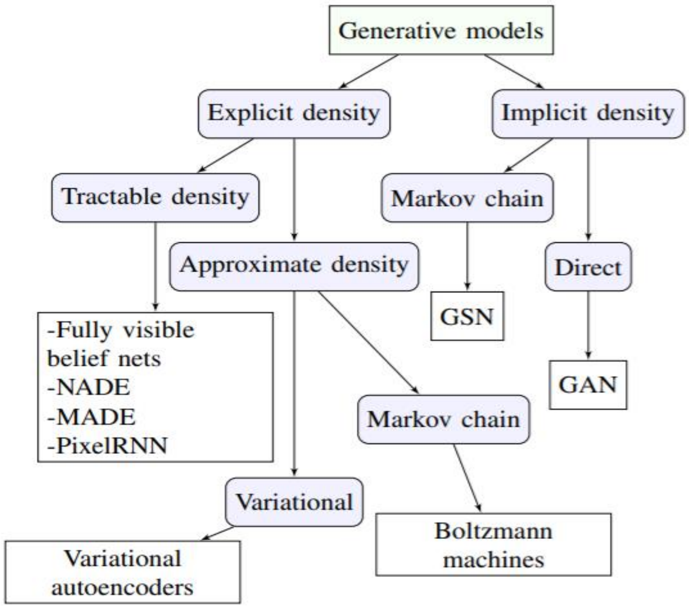


Fig. 25 Adapted from Goodfellow *et al*. [71], the taxonomy of generative models based on the tractability of their density distribution.

recogntion, which was also done effectively by VAEs [54]. Virtually, one may think that any of the generative models can be applied in some form to tackle all the problems that specific classes of generative models usually do.

Having mentioned that, it is still useful to apply only a certain and relevant class of generative models for a specific problem for better efficiency of problem solving, simplicity of model design (in the sense that one should not arbitrarily throw a deep generative network with many layers at data which doesn't require feature extraction at various levels for better learning), for better cost minimization, among various other reasons. In this section, we analyze all the generative models discussed thus far to give the reader a better understanding of the advantages and disadvantages of all the models for any given problem. Additionally, we implement the discussed models on suitable datasets to demonstrate experimental analyses.

Following are the sub-sections: 7.1 *Analysis of Deep Generative Models*, 7.2 *Analysis of pure ML-based Generative Models*, 7.3 *Comparisons*, 7.4 *Difficulty of Analyzing Generated Samples* and 7.5 *Future Directions*.

7.1 Analysis of Deep Generative Models

Deep generative models can be classified as whether they perform explicit density estimation of $P_{data}(x)$ or implicit. Explicit density estimation simply means defining and solving for $P_{model}(x)$, whereas implicit density estimation refers to *learning a model* that can sample from $P_{model}(x)$ without explicitly defining it. Fig. 25 describes the flowchart of deep generative models based on the tractabilities of their density distributions where NADE stands for Neural Autoregressive Distribution Estimation [72], MADE for Masked Autoencoders Distribution Estimation [73], PixelRNN for Pixel Recurrent Neural Networks [74] which is a type of fully visible deep belief network (FVBN) [75, 76], and GSN for Generative Stochastic Networks [77]. The problem with explicit density models is capturing all the complexities in a given data and simultaneously not losing tractability. To tackle this, FVBNs consider careful construction of the model which guarantees tractability, and on the other hand, explicit models that approximate density like VAEs incorporate tractable approximations in the likelihood and its gradients.

We further describe all the explicit density models having tractable density distributions listed in Fig. 25, which the paper does not formally address, in brief. FVBNs use the chain rule of probability to decompose a probability distribution over an M dimensional vector $\boldsymbol{x}$, giving us a product of 1-D probability distributions as,

$$P_{model}(\boldsymbol{x}) = \prod_{k=1}^{M} P_{model}(x_k|x_1, \dots, x_{k-1}) \quad \textbf{(50)}$$

The order of generating a sample in FVBNs is $O(n)$ because each sample has to be generated sequentially as x_1, then x_2 and so on. The advantage that GANs have over FVBNs is in the respect that all the samples are generated *in parallel* yielding a very high generator speed.

NADE, similar to FVBNs, assume that a product of 1D distributions represent a factorized M dimensional distribution $p(\boldsymbol{x})$ in any order o, or permutation of integers $\in [1, M]$,

$$p(\boldsymbol{x}) = \prod_{m=1}^{M} p(x_{o_m}|\boldsymbol{x}_{o_{<m}}) \quad \textbf{(51)}$$

where $o_{<m}$ represent dimensions $\in [1, m-1]$ and $\boldsymbol{x}_{o_{<m}}$ is the corresponding set of dimensions (a subvector) in the order o. NADE specifies a parameterization of all these M conditionals $p(x_{o_m}|\boldsymbol{x}_{o_{<m}})$ using a feed-forward multilayer perceptron given by,

$$p(x_{o_m} = 1|\boldsymbol{x}_{o_{<m}}) = sigm(\boldsymbol{V}_{o_m}, \boldsymbol{h}_m + y_{o_m}) \quad \textbf{(52)}$$

$$\boldsymbol{h}_m = sigm(\boldsymbol{W}_{o_{<m}} \boldsymbol{x}_{o_{<m}} + \boldsymbol{z}) \quad \textbf{(53)}$$

where $sigm(x)$ is the logistic sigmoid function defined by $sigm(x) = 1 / (1 + e^{-x})$ with a total of H hidden units, $\boldsymbol{V} \in \mathbb{R}^{M \times H}$, $\boldsymbol{y} \in \mathbb{R}^{M}$, $\boldsymbol{W} \in \mathbb{R}^{H \times M}$, $\boldsymbol{z} \in \mathbb{R}^{H}$, which are the parameters of NADE. As the hidden layer matrix $\boldsymbol{W}$ and bias $\boldsymbol{z}$ are shared among all hidden layers $\boldsymbol{h}_m$, NADE features parameters of the order $O(HM)$ as opposed to $O(HM^2)$ as in the case of individual neural networks. Training of NADE can be done through minimization of the average negative log-likelihood or by maximum likelihood through SGD (stochastic gradient descent),

$$\frac{1}{N}\sum_{n=1}^{N} -\log p(\boldsymbol{x}^{(n)}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M} -\log p(x_{o_m}^{(n)}|\boldsymbol{x}_{o_{<m}}^{(n)}) \quad \textbf{(54)}$$

MADEs are modified autoencoders which satisfy the *autoregressive* property, which is referred as such because of the sequential prediction (regression of) each dimension of $\boldsymbol{x}$ while computing the negative log-likelihood which we define below by eqn. (56). Let $l(\boldsymbol{x})$ denote the binary cross-entropy loss function given by,

$$l(\boldsymbol{x}) = \sum_{m=1}^{M} [-x_{o_m} \log \hat{x}_{o_m} - (1 - x_{o_m}) \log(1 - \hat{x}_{o_m})] \quad \textbf{(55)}$$

As before, we define $p(x_{o_m} = 1|\boldsymbol{x}_{o<m}) = \hat{x}_{o_m}$, along with the fundamental rule of probability that implies $p(x_{o_m} = 0|\boldsymbol{x}_{o<d}) = 1 - \hat{x}_{o_m}$ we transform eqn. (55) into a negative log-likelihood,

$$-\log p(\boldsymbol{x}) = \sum_{m=1}^{M} -\log p(x_{o_m}|\boldsymbol{x}_{o<m}) = \sum_{m=1}^{M} [-x_{o_m} \log p(x_{o_m} = 1|\boldsymbol{x}_{o<m}) - (1 - x_{o_m}) \log p(x_{o_m} = 0|\boldsymbol{x}_{o<m})] \quad \textbf{(56)}$$

Now, given an autoencoder with hidden representation $h(\boldsymbol{x})$ obtaining a compressed reconstruction $\hat{\boldsymbol{x}}$ defined by,

$$h(\boldsymbol{x}) = f(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}) \quad \textbf{(57)}$$

$$\hat{\boldsymbol{x}} = sigm(\boldsymbol{a} + \boldsymbol{V}\boldsymbol{h}(\boldsymbol{x})) \quad \textbf{(58)}$$

where $\boldsymbol{a}, \boldsymbol{b}$ are vectors and $\boldsymbol{W}$ and $\boldsymbol{V}$ are matrices representing connections from input-to-hidden and hidden-to-output respectively, with f as a non-linear activation function. Transforming this simple autoencoder to satisfy autoregressiveness can be done by knowing that since any outputs $\hat{x}_{o_m}$ depend upon subvector $\boldsymbol{x}_{o<m}$, there is no computational waypoint between $\hat{x}_{o_m}$ and $x_{o_m}, \dots, x_{o_M}$. Hence, it can be said that at least one of these paths represented by matrices $\boldsymbol{W}$ or $\boldsymbol{V}$ equals to null. To nullify connections, a binary mask matrix is applied element-wise and multiplied, the values of which are 0 for the entries that correspond to paths that are null (or non-existent). Using the autoencoder defined by eqn. (57) and eqn. (58),

$$h(\boldsymbol{x}) = f(\boldsymbol{b} + (\boldsymbol{W} \odot \boldsymbol{K}^{\boldsymbol{W}})\boldsymbol{x}) \quad \textbf{(59)}$$

$$\hat{\boldsymbol{x}} = sigm(\boldsymbol{a} + (\boldsymbol{V} \odot \boldsymbol{K}^{\boldsymbol{V}})h(\boldsymbol{x})) \quad \textbf{(60)}$$

where $\boldsymbol{K}^{\boldsymbol{W}}$ and $\boldsymbol{K}^{\boldsymbol{V}}$ are defined to be the masks for matrices $\boldsymbol{W}$ and $\boldsymbol{V}$ respectively which help in transforming the autoencoder satisfy the autoregressive property.

PixelRNN is another neural autoregressive model which models distributions over pixel values which optimizes weights by maximization of likelihood. Given the fragmentation of a joint distribution $p(\boldsymbol{x})$ defined by eqn. (50), each conditional is considered a multinumial and parameterized through a softmax activation layer. Long Short-Term Memory (LSTM) networks are used to model the product of the conditionals (which are also used in reccurent neural networks, RNNs) to learn long-term dependencies between pixels. For a given pixel, the contextual dependencies are learned by iterating the images row by row from top to bottom demonstrated by Fig. 26.

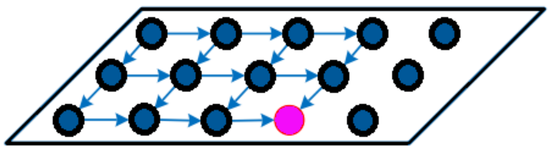


Fig. 26 Autoregressive LSTM-based image modelling.

Explicit density estimation models which require approximations provide an intractable explicit density function which needs approximations to maximize likelihood. VAEs are the variational categorization of approximating density models. In general, the lower bound in variational methods is given by,

$$\varepsilon(\boldsymbol{x}; \theta) \leq \log P_{model}(\boldsymbol{x}; \theta) \quad \textbf{(61)}$$

The problem with VAEs is that when a weak approximation of the posterior distribution or prior distribution is used, no matter how good the optimization of the parameters or how big and diverse the dataset, distance between ε and the true likelihood can make the distribution P_{model} be not similar to P_{data} at all; the model can learn an undesired distribution. Variational methods obtain a very high likelihood but still fail to generate samples of quality as high as that of GANs, and when compared to FVBNs, they are difficult to optimize. Generative models often are evaluated based on the quality of samples they produce which is a subjective opinion and not an empirical fact (we discuss more about evaluation of generative models later in this section).

The other class of generative models which estimate density explicitly by approximation are BMs which use Markov chains to do so. Samples are drawn, $\boldsymbol{x}^{\circ} \sim T(\boldsymbol{x}^{\circ}|\boldsymbol{x})$ repeatedly which simultaneously update $\boldsymbol{x}$ by the transition function T which makes the distribution $\boldsymbol{x}$ eventually converge to be a sample from $P_{model}(\boldsymbol{x})$. Running the Markov chain until it converges is also referred to as *burning in the Markov chain* in literature. However, achieving this convergence is not simple. In literature, the number of steps the Markov chain must run before reaching equilibrium is called the *mixing time*. There is no proper theory regarding predicting when a Markov chain will reach convergence, although it is clear that it definitely will converge at some point. This is the reason why usually Markov chains, wherever used, are made to run for a sufficient amount of time so one can say that it has roughly converged. Mixing time can be very long and thus in practice, $\boldsymbol{x}$ is put to use a long time before it can actually converge to be a sample from P_{model}. Also, efficiency of Markov chains in higher dimensions diminishes. BMs, more specifically DBNs brought the focus back on deep learning when the official DBN paper was released, however, BMs altogether have lost relevance and are moving along the downward curve in terms of popularity. The reason for this could be that the MC approximation techniques fail to outperform their supervised, directed graph model-centric counterparts in terms of accuracy, and another could be the high cost associated with burning in MCs.

Table 1: Configuration settings for training the RBM.

# Visible Nodes	# Hidden Nodes	# Epochs	Batch Size
1682	100	10	100

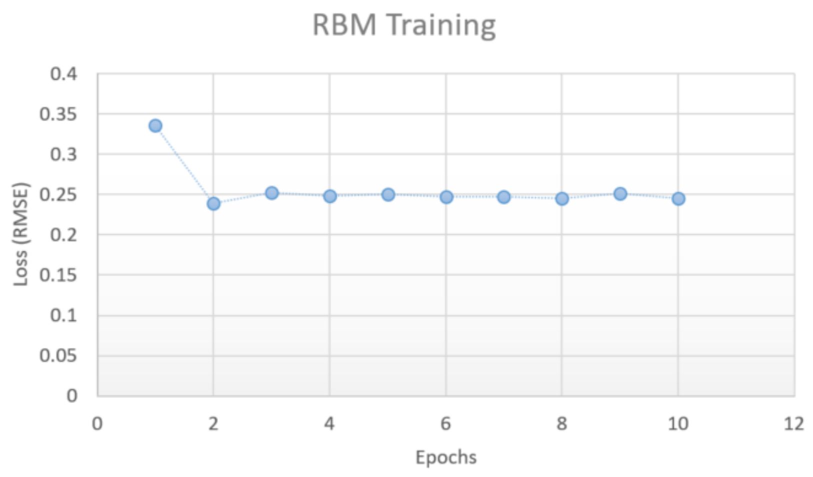


Fig. 27 Reconstruction RMSE loss encountered per epoch while training the RBM.

Finally, the implicit density models positioned on the right branch of the taxonomy interact indirectly with P_{model} while training by sampling from it. This sampling can be done directly as in the case of GANs, or a Markov chain could be run several times before reaching convergence which translates to sampling from P_{model} itself, an example of which is the GSN. However, as discussed earlier, burning in Markov chains is cost intensive and GANs are better in those regards. Moreover, GANs can generate samples in parallel when put in comparison against FVBNs which only generate samples one by one. There are no difficult approximations of intractable probabilistic computations in maximum likelihood estimation, and, the added advantage GANs possess is better sample quality. Yet, GANs come with their own disadvantages; they are difficult to optimize and are very susceptible to collapsing as discussed in section 6. Section 7.4 also discusses the difficulty in properly analyzing whether the GAN has overfit or underfit the training data.

7.1.1 Implementation of Restricted Boltzmann Machine[1]

For this experiment, we demonstrate how RBMs can be used as recommender systems by generation of a set of recommended movie titles for a particular user given their past history of movies that they liked or disliked. We use the MovieLens ml-100k dataset [94] as our training data. The data comprises 100000 ratings of 943 users (in rows) on 1682 movies (in columns). The ratings fall in the range of 1~5 which, at the time of preprocessing, was binarized so that if the rating was over 3, it would be replaced by a 1, otherwise a 0. The data is binarized as RBMs generally are trained on binary data and are suitable for such cases. The data is fed to the visible layer of the RBM row-wise during training which then goes through contrastive divergence stochastically and a regenerated output is acquired from the visible layer. The regenerated outputs are compared to the original ratings of the users and the model's performance is evaluated based on RMSE metrics (root mean square error) defined as,

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i\in N}(r_{ui} - \hat{r}_{ui})^2} \qquad \textbf{(62)}$$

In eqn. (62), u denotes a user, i denotes a movie, r_{ui} denotes the rating of user u on movie i, $\hat{r}_{ui}$ denotes user u's predicted rating on movie i and N is the number of movies. The configuration of the network is as follows: the number of visible layers is set to be equal to N (which, in our case, is 1682). This is done so that the input layer can assign one node to each individual rating made by the user for every movie. Obviously, each user cannot rate all 1682 movies and hence these nodes receive a -1 instead of 0. During training, the negative values are frozen and hence these values are not trained. The configuration settings for RBM training are listed in Table 1. After training, we acquire the reconstruction loss per each epoch as illustrated by Fig. 27 and attain a satisfactory loss on the test set of **0.2439** based on RMSE. Hence, we may conclude that RBMs can be used as recommender systems.

Table 2: DBN configuration settings.

Hidden Layer Structure	Batch Size	Learning Rate (RBM)	# Epochs	Activation Function
{256, 512}	10	0.06	20	Sigmoid

7.1.2 Implementation of Deep Belief Network

In the previous subsection we demonstrated how RBMs can be used for generating data close to the original distribution when applied as recommender systems. Here, we use DBNs for feature extraction and pipeline a logistic regression model after extracting features to classify a dataset of digits [93] ranging from 0 to 9. We compare results of classification when logistic regression is applied on raw pixel data versus when the features are extracted from the same data using DBN followed by the

[1] All source code for each implementation further on is provided in [92].

application of logistic regression. The dataset is augmented 5-fold by shifting the 8x8 images 1px (single pixel) to the left, right, down and up followed by splitting of data into training and testing set of proportions 8:2. The DBN was constructed with specifications provided by Table 2, where {256, 512} implies that the first hidden layer comprises 256 nodes and the second 512 nodes. As mentioned earlier in Section 4.2 the training of DBNs is done by initially training an RBM followed by its insertion at the top of the DBN to eliminate independence of the prior at the top-most layer.

It is observed after our experiments that feature extraction using DBNs yields superior classification performance as compared to classification without feature extraction. Reconstruction error of the RBM by training on 20 epochs is illustrated by Fig. 28. The evaluation metrics for evaluation of both approaches we use are namely *precision*, *recall*, and *F1 score*. Precision (also referred to as *specificity*) is defined as the ratio of positive observations that the model correctly predicted to the total number of predicted positive observations. Recall (also referred to as *sensitivity*) is the ratio of positive observations that the model correctly predicted to all the observations in the actual class of 'positive'. Finally, F1-score is defined as the weighted average of both precision and recall. Mathematically,

$$Precision = {}^{TP}\!/_{TP + FN} \quad \textbf{(63)}$$

$$Recall = {}^{TP}\!/_{TP + FN} \quad \textbf{(64)}$$

$$F1\ Score = \frac{2(Recall * Precision)}{(Recall + Precision)} \quad \textbf{(65)}$$

We illustrate the results in terms of the above metrics by Fig. 29 for classification by feature selection using DBNs and by Fig. 30 for classification without feature selection (in both cases, using logistic regression).

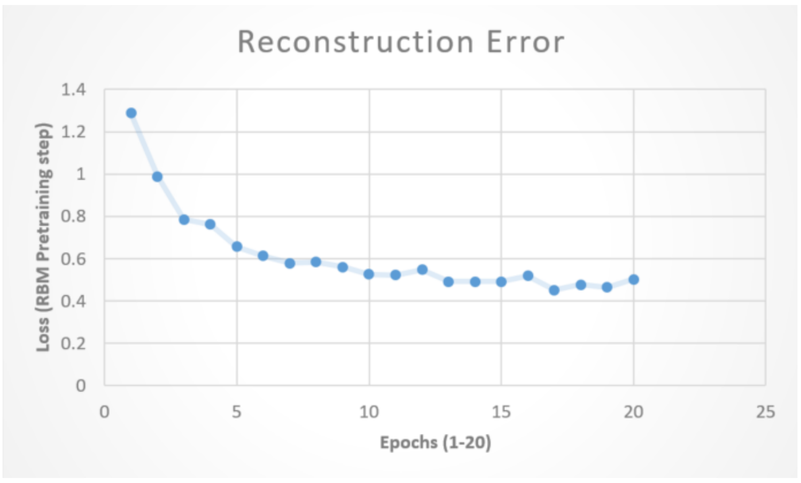


Fig. 28 RBM pre-training reconstruction error at each epoch.

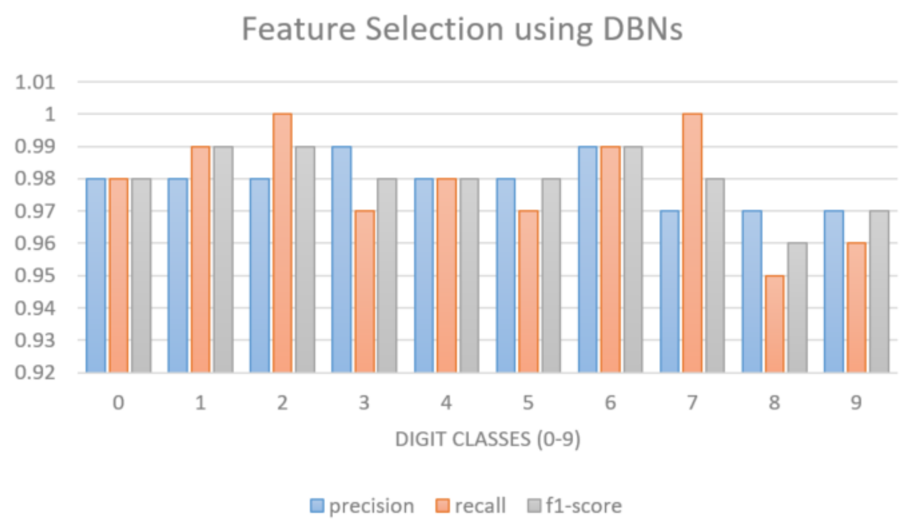


Fig. 29 Metrics precision, recall and F1 score for each of the digit classes as classified by a logistic regressor on DBN extracted features.

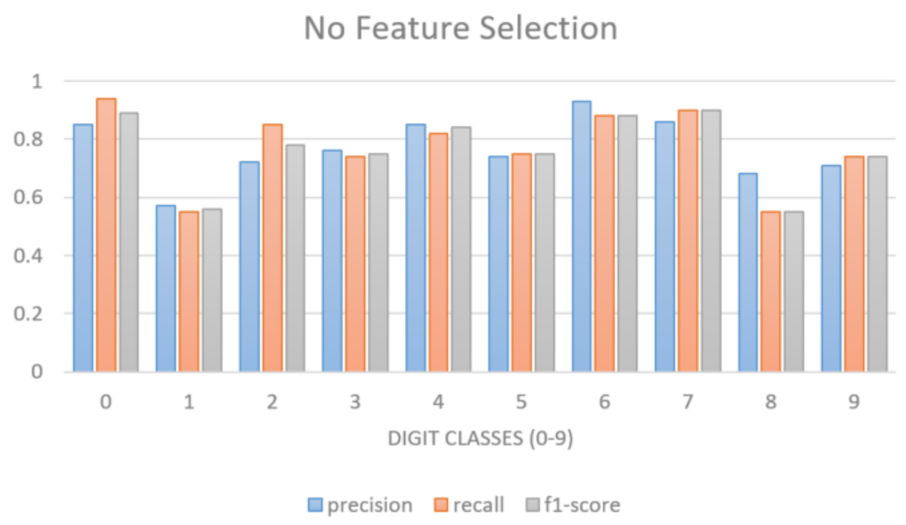


Fig. 30 Metrics precision, recall and F1 score for each of the digit classes as classified by a logistic regressor on raw pixel-value data without feature selection. Clearly, these values are diminished due to absence of feature selection.

Table 3: Configuration settings for training of DBM in MNIST digit recognition dataset.

Dimensions	# Epochs	Learning rate	Batch size
{784, 500, 784}	2	0.01	5

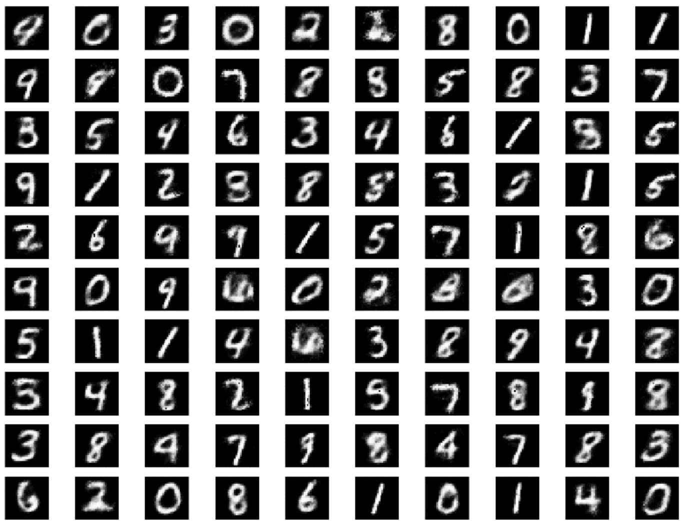


Fig. 31 Regenerated samples of MNIST digit recognition dataset by DBM-based feature selection and plotting.

7.1.3 Implmentation of Deep Boltzmann Machine

We saw how RBMs can be applied as recommender systems and how DBNs may be used as powerful feature extractors. In this implementation, we demonstrate how DBMs can be used to regenerate data in form of images by taking the popular MNIST dataset of digits [95]. Once the data is imported, the pixel values, ranging from 0~255 are divided by 255 for normalization purposes, scaling all pixel values to lie in [0, 1]. The configuration settings used for the training of the DBM are given in Table 3. The generated samples are illustrated by Fig. 31. The dimensions {784, 500, 784} refer to the consecutive enumeration of visible and hidden layers separated by commas. We notice that DBMs can reconstruct simple image data such as the MNIST dataset to a decent extent of good sample quality.

7.1.4 Implementation of Variational Autoencoder

In Convolutional Variational Autoencoders (CVAEs), the encoder and decoder networks comprise convolutional and fractionally strided (or transpose convolutional) layers respectively which help in generating images similar to DCGANs. For this experiment, we use the same dataset as used for DBM-based image generation. The images are resized to 28x28x1 where 1 represents the channels or the *colors*, and since our data is black and white, there is only a single channel that ranges between 0~255. Next all the pixel values are normalized with the division of 255 so the values now lie between [0, 1]. Here, we take 60000 training images as opposed to 10000 test images (6:1 split for training size to testing size) with a batch size of 32. All the configuration settings are provided by Table 4. Latent dimensions refers to the central latent dimensions in CVAE which is the dimensions of data being reconstructed between the

Table 4: Configuration settings for training of CVAE on MNIST digit recognition dataset.

Latent dimensions	# Epochs	Learning rate	Batch size	Conv filters
2	10	0.001	32	32

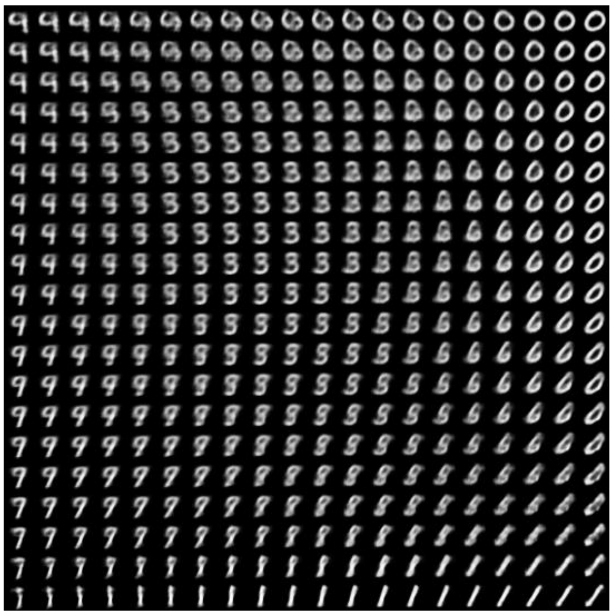


Fig. 32 CVAE generated image of the MNIST digit recognition dataset.

Table 5: Configuration settings for the training of DCGAN.

Input Dimensions	Batch size	Learning rate (D, G)	# Epochs
64x64x3	64	0.0002	25

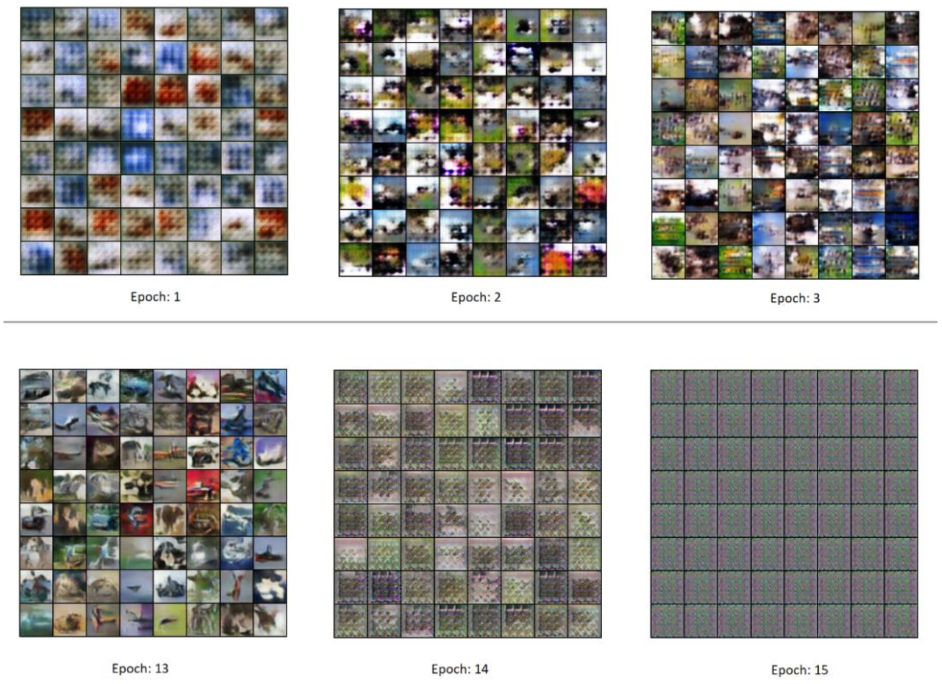


Fig. 33 (**Top**): Output of the DCGAN when initialized from epochs 1 till 3. (**Bottom**): Output of the DCGAN from epochs 13 till 15 where a collapse can be seen in epochs 14 and 15.

encoder and decoder networks. Conv filters refer to the number of filters applied on each convolutional layer. The plotted output image of the CVAE is illustrated by Fig. 32.

7.1.5 Implementation of Generative Adversarial Network

In this implementation, we make use of DCGANs to reconstruct images from the

CIFAR-10 dataset [59]. All input images are set to have the dimensions 64x64x3 where 3 refers to the number of channels, and thus 3 refers to channels *R*, *G* and *B* for red, green and blue which when combined can form any color in the visual spectrum. The configuration settings for training of DCGAN are given in Table 5. Note that learning rate for both, discriminator and generator networks (D, G) as given by Table 5 is set at 0.0002. We notice that while training, the network collapses randomly at epoch 15 (by randomly, we mean that it is not mandatory that this DCGAN with its specific hyperparameters would always collapse at epoch 15 on the same dataset), the effects of which start to become visible from epoch 14 as illustrated by Fig. 33 (bottom). The initial epoch results (epochs 1, 2, 3) are shown by Fig. 33 (top) to illustrate how the DCGAN learns the input distribution (as shown in Fig. 34) and corrects images initially. DCGANs, as mentioned earlier, are susceptible to collapsing, and this experiment verifies that they are indeed unstable and difficult to train.

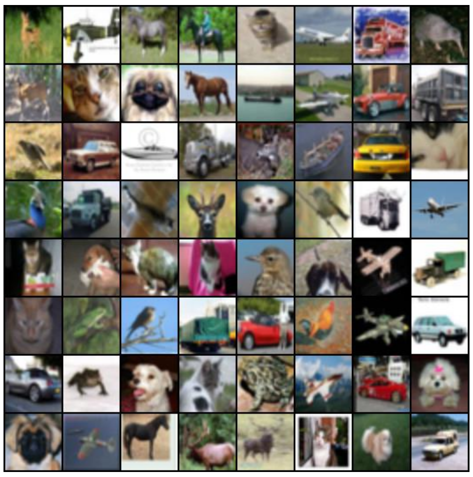


Fig. 34 Real samples of the CIFAR-10 dataset that the DCGAN tries to model.

We now shift our analysis to focus on the generative models discussed in this paper that do not fall in the category of DL (as shown in Fig. 2).

7.2 Analysis of pure ML-based Generative Models

When dealing with problems that require clustering, if the distribution to be modeled has hidden, non observable parameters, a viable option is to use GMM. This is because the model essentially assigns probabilities of each point being in some cluster k and does not assume, as in the case of K-means clustering, that the probability of a certain data point to belong to a certain cluster is 1. GMMs are a lot more flexible in terms of cluster covariance and thus we get more flexible clustering regions (elliptical instead of circular) as observed in Fig. 4. Another advantage of GMMs is that they accommodate mixed membership in the sense that data points can *probabilistically* belong to more than one cluster, as opposed to K-means which forcefully assigns one cluster for each data point. In GMMs, a data point belongs to each cluster to a different extent or degree. Mixed membership can be beneficial in certain cases, for example, when trying to cluster news articles based on their tags. In this case, a news article may contain many tags, hence it may belong to many clusters, however, some tags are more prominent and some others may not be so prominent. This prominence reflects in the probability assignments accordingly. In other cases, mixed membership may not be so beneficial which can be considered a disadvantage of GMMs, for example, when clustering items that cannot fall under more than one cluster (say, organisms, which can only belong to one species). In this case, K-means does a better job at hard defining a single cluster for each data point.

LDA, as discussed in Section 2 is a dimensionality reduction technique in actuality. However, we confine our focus to LDA when applied on natural language processing (NLP) tasks (though it can be applied on many other forms of data) as it has a more intuitive use in this field for topic modelling. It assumes that the corpora given as input contain documents which belong to a number of topics. Each topic is associated with a vocabulary of words, and that each document is the result of a mixture of probabilistic samplings – first over distribution of possible topics of the input documents, and then over the amalgamation of possible words in the selected topic. This general assumption gives LDA the biggest advantage as opposed to other previously employed topic modelling approaches viz. latent semantic indexing (LSI) [78] and probabilistic LSI (pLSI) [79]. This advantage is that LDA can be extended to separate documents into topics for documents that do not belong to the corpora it was trained on. LDA generalizes the model it uses for dimensionality reduction for new corpora. For example, if the LDA model was trained on news articles corpora to group the news articles into categories like politics, crime, sports, etc. it would be allowed to use the same model to categorize newly published articles. This is not possible in LSI. Another advantage of LDA over pLSI is that the number of parameters grow linearly with the number of documents in the corpora in pLSI, whereas on the other hand, the number of parameters in LDA goes linearly with the number of topics which is much lower than pLSI. Hence, LDA can outperform pLSI in terms of speed on much bigger datasets.

There are, however, many disadvantages of using LDA. First, there must exist prior knowledge about the number of topics; the number of topics K is fixed. Secondly, it is not suitable to apply LDA on short texts as found in [80] because LDA exploits statistical inference to discover hidden patterns in the data. Thus, when the data is very less (as in the case of tweets on Twitter), there are very few observations to infer the parameters of the model which hinders with the accuracy of the whole affair. Thirdly, LDA cannot capture correlations between topics. For example, LDA will not be able to use to its advantage that a tag, say, computer science is highly correlated with another tag namely programming language. Even with many more disadvantages, LDA has revolutionalized the field of topic modelling and has become the central idea associated with it.

HMMs have been considered as a specific form of dynamic Bayesian networks and since 1980s have been used for modelling biological sequences (e.g DNA). HMMs assume that the system they try to model is generated by a Markov process with hidden or unknown parameters. Thus, these unknown parameters are tried to be learnt based on the given observable parameters. Not only biological sequence modelling, but they have been used succesfully for pattern recognition in the form of speech recognition, OCR, data mining, classification and structural analysis. More generally, one can infer that an HMM can be applied if the system to be modelled is made up of different stages or states that exist in definite or typical orders. HMMs have strong statistical foundation, are simple conceptually, and are very flexible – they are the most flexible generalization of sequence profiling methods as they can handle inputs of variable length. The problem with these models, however, is that the number of unstructured parameters can be very high quite often. Another problem that arises in sequence labelling with HMMs is that at times, it is required to know the correlation between states that are not immediate neighbours as sequences can have contextual property among states and are also of certain length, all of which is not taken into account by HMMs which only take the immediate next state in the transition function.

7.2.1 Implementation of Gaussian Mixture Models

In the experiment, we randomly generate and plot 400 data points gathered around 4 different centers; 100 data points per center with a standard deviation of 0.6 and initialization of random state by 0. Next, we make another version of this scatter plot by stretching each of the four distributions unidirectionally to better demonstrate results of GMMs and prove how effective they are, as discussed, when the distribution is oblongated. Fig. 35 shows the two randomly generated data distributions. Further, we apply GMM to both these distribution and as an addition, also apply K-means (note that K-means implementation is not provided in [92]) to compare our results. It

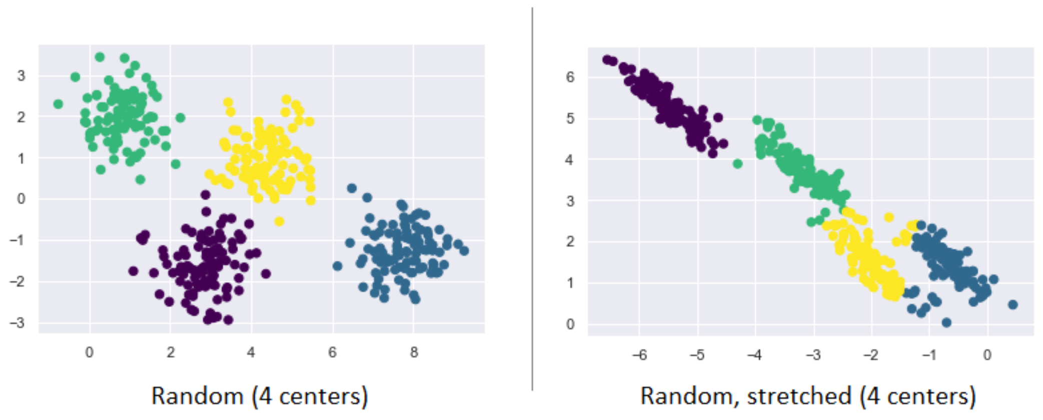


Fig. 35 (**Left**): Randomly generated quad-centric data distribution. (**Right**): Randomly generated quad-centric stretched (oblong) data distribution.

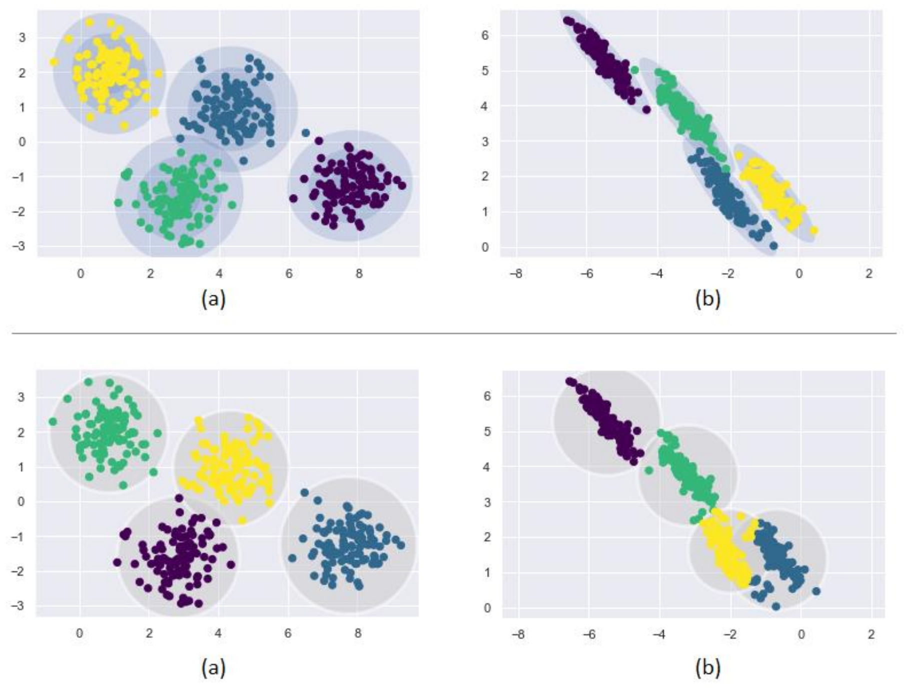


Fig. 36 (**Top**): (a) GMM applied for clustering on randomly generated quad-centric data distribution. (b) GMM applied for clustering on randomly generated quad-centric stretched (oblong) data distribution. (**Bottom**): (a) K-means used on the same distribution for clustering as displayed by Fig. 36 (Top) (a). (b) K-means applied on the same distribution for clustering as displayed by Fig. 36 (Top) (b).

is noticed that results shown by Fig. 36 (Top) (a) and Fig. 36 (Bottom) (a) are quite similar because of a somewhat circular nature of the data distribution. However, the true advantage of GMMs over K-means is realized by Fig. 36 (Top) (b) and Fig. 36 (Bottom) (b) as we notice the clusters generated by GMM is more precise due to its flexible clustering capability as discussed in Section 1.

7.2.2 Implementation of Latent Dirichlet Allocation

For this experiment, we take a textual dataset which, after appropriate preprocessing, contains a raw mixture of author names, abstracts, titles the text corresponding to every research paper presented in NIPS (Neural Information Processing Systems) conferences from the first conference in 1987 till the 2016 conference. To get an estimate of the most popular *phrases* used in the papers, we generate a word cloud as represented by Fig. 37 (left) and also a bar graph for the most popular *words*

Table 6: Top 10 most prominent words generated by LDA over 5 topics.

Topics	Prominent words (top 10)
1	model, network, image, neural, figure, input, time, using, images, neurons
2	state, learning, policy, time, function, value, action, algorithm, optimal, reward
3	data, model, models, distribution, using, 10, set, algorithm, gaussian, number
4	algorithm, matrix, problem, function, theorem, set, 10, let, log, learning
5	learning, training, data, set, networks, neural, network, classification, using, model

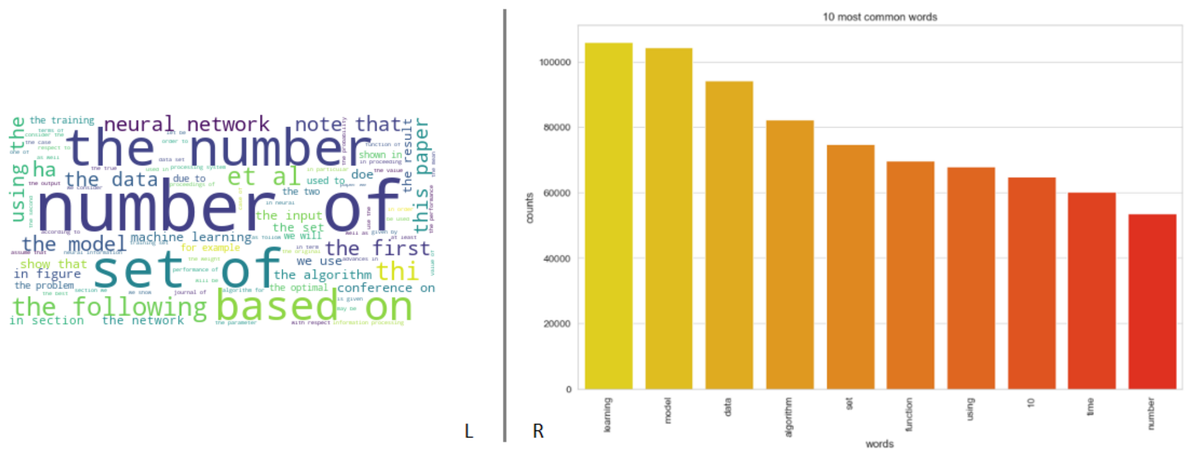
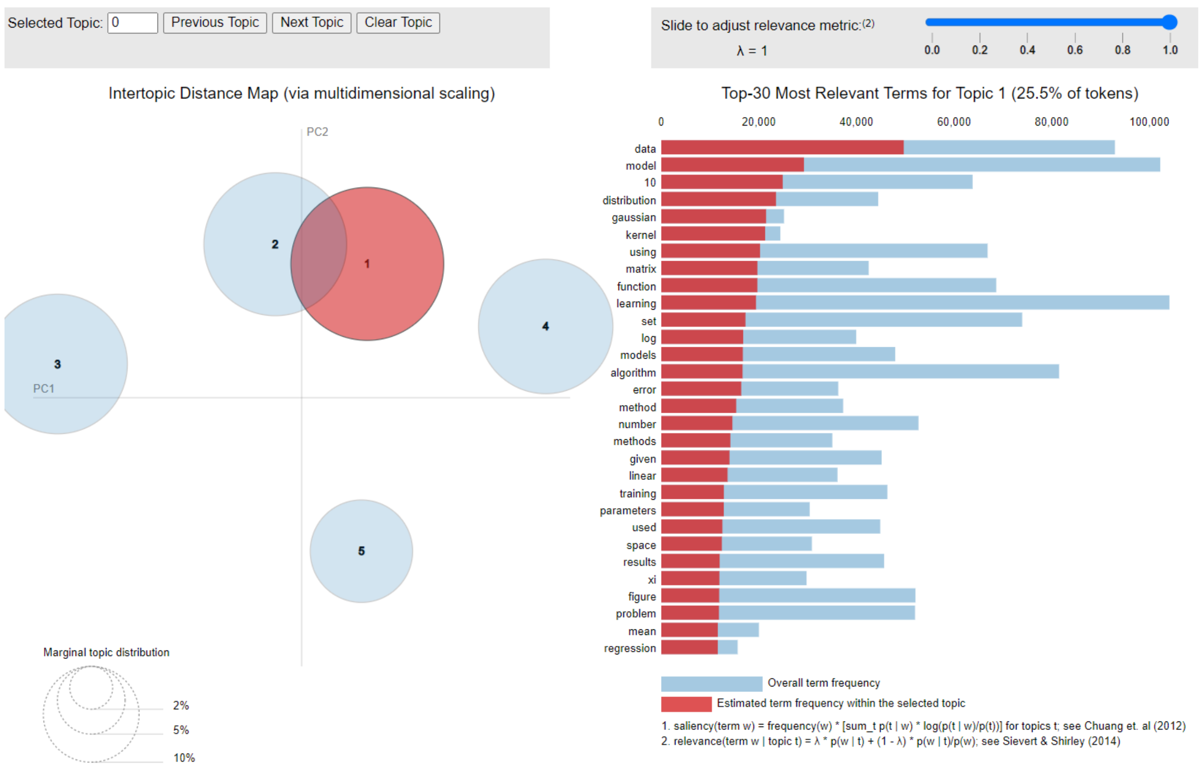


Fig. 37 (**Left**): Generated wordcloud. (**Right**): 10 most common words.



Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
Marginal topic distribution
2%
5%
10%
Top-30 Most Relevant Terms for Topic 1 (25.5% of tokens)
Overall term frequency
Estimated term frequency within the selected topic

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Fig. 38 Inter-topic distance map through multidimensional scaling along with the top 30 salient words (light blue bar) and their relevance in selected topic 1 (red bar) with $\lambda = 1$.

represented by Fig. 37 (right). We apply LDA on this dataset for finding 5 topics and print the 10 most prominent words and the results attained are enlisted in Table 6**.** Further, introduce two terms *saliency* [93] and *relevance* [94] in LDAs defined by,

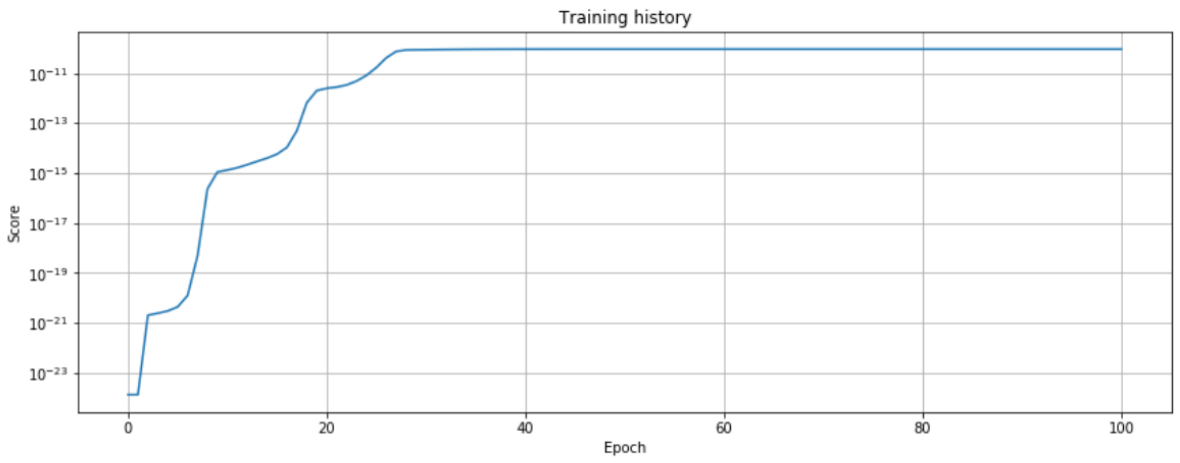


Fig. 39 Training HMM to maximize the score over 100 epochs.

Table 7: Hidden state transition matrix.

	L_1	L_2	L_3	L_4
L_1	0.90	0.08	0.01	0.01
L_2	0.01	0.90	0.05	0.04
L_3	0.03	0.02	0.85	0.10
L_4	0.05	0.02	0.23	0.70

Table 8: State emission probability matrix.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
L_1	0.01	0.01	0.20	0.01	0.30	0.05	0.01	0.40	0.01
L_2	0.01	0.01	0.01	0.10	0.01	0.30	0.05	0.01	0.50
L_3	0.30	0.20	0.01	0.10	0.01	0.30	0.05	0.02	0.01
L_4	0.03	0.01	0.01	0.19	0.01	0.39	0.39	0.01	0.03

Table 9: Initial state probability distribution matrix.

L_1	L_2	L_3	L_3
0	1	0	0

Table 10: Results of the trained HMM on input sequence of emissions (radiations, in our case) based on the predictions of the hidden state (locations).

Time (t)	Input Observable Emission Sequence (Radiation)	Output State Prediction (Location)
1	R_6	L_2
2	R_6	L_2
3	R_6	L_2
4	R_9	L_2
5	R_6	L_2
6	R_6	L_2
7	R_6	L_2
8	R_6	L_2
9	R_3	L_1
10	R_5	L_1
11	R_5	L_1
12	R_5	L_1
13	R_5	L_1

14	R_5	L_1
15	R_4	L_3
16	R_4	L_3
17	R_8	L_4
18	R_8	L_4
19	R_8	L_4
20	R_1	L_4
21	R_7	L_4
22	R_7	L_4
23	R_7	L_4
24	R_2	L_4

$$saliency(w) = f(w) \sum_T P(T|w) \, log \frac{P(T|w)}{P(T)} \quad \textbf{(66)}$$

$$relevance(w|T) = \lambda P(w|T) + (1-\lambda) \frac{P(w|T)}{P(w)} \quad \textbf{(67)}$$

where, w denotes a word from the data vocabulary, T denotes a topic from the set of topics, $P(E)$ denotes the probability of event E and λ denotes a weight parameter ($0 \leq \lambda \leq 1$). *Saliency* is a measure proposed by Chuang *et al.* (2012) [93] that aids rapid classification and disambiguation of topics. *Relevance* is a measure proposed by Sievert *et al.* [94] (2014) of a term that provides users with an understanding of how useful the word is in describing the topic. Based on these two parameters, we use *LDAvis*, a tool made by [94] to visualize inter-topic distances through multidimensional scaling projected on *principal component axes* PC1 and PC2 between the 5 topics. Moreover, we show ranking of the top 30 most salient and relevant words (decorated by light blue and red bars, respectively) in any selected topic with $\lambda = 1$ as demonstrated by Fig. 38.

7.2.3 Implementation of Hidden Markov Models

In this implementation, we assume there to be a particle x that travels to different locations (four) given by the set $L = \{L_1, L_2, L_3, L_4\}$ and can be found at one of the locations in L at any particular time t. We further assume that particle x emits certain (nine) types of radiations defined by the set $R = \{R_1, R_2, R_3, \ldots, R_9\}$ corresponding to the location it resides in at any time t. We define set R to be the set of the observables and L to be the set of hidden states in the HMM. At every time update, the particle transitions from one hidden state to the other, $L_i^{(t)} \rightarrow L_j^{(t+1)}$ and the probabilities of transitions between the hidden states is given by Table 7. Next, we define the probabilities of particle x to emit a certain type of radiation R_i while at any given location L_j by Table 8. Finally, we present the initial state probability distribution matrix by Table 9 which specifies where x resides initially. Note that Table 7, Table 8, and Table 9 specify the true distribution of the system and are provided to demonstrate the likelihood of x to emit a certain event when in a certain state. We apply our HMM on an arbitrary input of observed radiations without the model's prior knowledge of Table 7 and Table 8 and train it on 100 epochs (Fig. 39) to come up with predictions of likelihood of the particle's location as given by Table 10.

Note that since the model was initialized from a random state, there is no control over the naming of hidden states of L which is a convention of the model. In other words, the HMM model does not really know which locations we mean by L_j. In reality, we shall have to swap the output label names which correctly explain the true emission probability distribution that exists.

7.3 Comparisons

Table 11 gives an application-oriented comparison of all the generative models discussed in this paper to give the readers a quick overview on how the models compare

Table 11: Application-oriented comparison of all generative models.

Model	Density estimation	Advantages	Disadvantages
GMM	Explicit	Flexible cluster covariance. Mixed membership of data points.	Mixed membership not always the best choice. Fails to perform in higher dimensional data (e.g > 6-D).
HMM		Strong statistical foundation, simple concept, very flexible.	Large number of unknown parameters in bigger sequencing problems. Fails to understand correlations in sequence.
LDA		Extensible and applicable on new data points. Parameters increase linearly with number of topics – not documents (as in pLSI).	Requires prior knowledge of number of topics. Not suitable for short texts. Fails to capture correlation among topics.
RBM		Expressive at encoding higher-order correlations due to hidden layers. Allows for feature extraction to train other models on top of.	Requires Markov chains – computationally expensive and uncertain convergence time. Partition function estimation is not easy.
DBN		Powerful feature extractor for pattern recognition, can be fine-tuned with a small labeled dataset.	Complex training procedure. Requires Markov chains.
DBM		Efficient feature extraction. Higher performance gain by adding layers.	Requires Markov chains, mean field approximations. Complex training procedure.
VAE		Achieves high data likelihood; precise control over latent representations. Objective is measurable (lower bound on goodness of model)	Generated samples are not sharp; blurry. Limited approximation to true posterior.
(DC, FCC, C, S) GANs	Implicit	Parallel sample generation; fast. High sample quality. No Markov chains needed.	Training requires finding Nash equilibrium – harder problem than optimizing an objective function. Difficult to train due to instability and possible collapse. Difficult to evaluate empirically (easy subjectively).

Table 12: Encountered positive and negative aspects of each model while experimentation.

Model	Pros	Cons
GMM	Strong Python3 library support. Very powerful for complex shaped clusters and easily visualized.	We noticed mixed membership or intersection of different cluster regions which may not always be desirable.
LDA	Over a corpora of text documents, we were able to detect *n* number of topics and the top *m* words related to each topic where *m* and *n* were decided by us. With *LDAvis*, some very useful insights were drawn.	Computation of *LDAvis* to generate the final report took extensive memory and hours of processing. So did LDA in generating the topics and the top 10 words.
HMM	The HMM trained and gave an output prediction of the latent state sequence with only the input of a single observable sequence; the training prerequisite data (input sequence) was very simple.	Implementing HMM in our instance took memory of >8 GB for a very simple model with not many observable and hidden states.
RBM	We provided a lengthy visible layer of 1682 nodes with only a single hidden layer responsible for detecting 100 features (having 100 nodes) with a test loss of 0.249 which is considered optimal for recommender systems.	The data had to be preprocessed into binary form for proper functioning of the RBM.

DBN	Use of DBNs for classification had a huge impact on metrics like precision, recall and F1 score.	We went through a complex and computationally expensive pretraining process due to training the top level RBM for 20 epochs.
DBM	Satisfactory sample quality on MNIST digit recognition dataset.	Similarly complex training procedure as with DBNs.
CVAE	Satisfactory sample quality on the same data as used for DBM, the MNIST digit recognition dataset.	Library support for constructing CVAE was scarce.
DCGAN	We noticed that upto 13 epochs the DCGAN had started generating very realistic models and with some hypertuning it may have avoided a collapse.	Each epoch was computationally expensive and time-taking. Additionally, the GAN collapsed, the signs of which crystallized by the 15th epoch.

Table 13: Comparison of all generative models discussed in this paper.

Model	Inference			Sampling	$P(x)$ evaluation	Design	Deployed in fields
	MCI	VI	LAI				
GMM	✕	✕	✕	No difficulties	Tractable	Simple; mixture of probability distributions	Clustering, acoustics analysis, etc.
HMM	✕	✕	✕	No difficulties	Tractable	Involves hidden states; level-wise simple	DNA sequence analysis, pattern recognition in sound/speech, etc.
LDA	✕	✕	✕	No difficulties	Tractable	Many variables in play; to be estimated	Dimensionality reduction, topic modelling, etc.
RBM	✕	✓	✕	Requires Markov chain	May be approximated via AIS; intractable	Complex, designed carefully to ensure various parameters	Dimensionality reduction, regression/classification, CF, topic modelling, etc.
DBN	✕	✓	✕	Requires Markov chain	May be approximated via AIS; intractable	Complex, designed carefully to ensure various parameters	Classification, speech recognition, information retrieval, drug discovery, etc.
DBM	✕	✓	✕	Requires Markov chain	May be approximated via AIS; intractable	Complex, designed carefully to ensure various parameters	Pattern recognition, information retrieval, regression/classification, etc.
VAE	✓	✕	✕	Requires Markov chain	Not represented clearly, may be estimated via PDE	Any differentiable function permitted	Image forecasting, CF, modelling acoustic features and molecule design, etc.
GAN	✕	✕	✓	No difficulties	Not represented clearly, may be estimated via PDE	Any differentiable function permitted	Drug discovery, anomaly detection, image analysis and transformations, etc.
DCGAN	✕	✕	✓	No difficulties	Not represented clearly, may be estimated via PDE	Any differentiable function permitted	Image generation
FCCGAN	✕	✕	✓	No difficulties	Not represented clearly, may be estimated via PDE	Any differentiable function permitted	High quality image generation, etc.
CGAN	✕	✕	✓	No difficulties	Not represented clearly, may be estimated via PDE	Any differentiable function permitted	Face generation and aging simulation, image feature editing, speech enhancement, language identification, etc.

SGAN	✕	✕	✓	No difficulties	Not represented clearly, may be estimated via PDE	Any differentiable function permitted	Text-to-image generation, image analysis and transformations, etc.

with each other based on their pros and cons. Additionally, we formulated Table 12 to make users aware of the positive and negative aspects that we came across while implementing all the models. A summary and comparison of all the models discussed is given below in Table 13 where MC, VI, LAI, AIS, CF, and PDE respectively stand for Markov chain, variational inference, learnt approximate inference, annealed importance sampling, collaborative filtering and Parzen density estimation.

7.4 Difficulty of Analyzing Generated Samples

We have seen how GANs produce sharp generated images as opposed to VAEs. One may be tempted to say that GANs are better at generating images but that is a wide misconception. Usually, generative models are evaluated on the basis of how realistic the generated samples appear to be when compared with the data distribution; a visual inspection. However, it is possible for a very weak probabilistic model to generate very good samples.

One way to evaluate the models is to map the nearest neighbor of the generated sample to the data distribution by Euclidean distance in data space $\boldsymbol{x}$. Then it would be known if the model is overfitting as the generated sample would appear simply copied from the sample in the training data. In this case, visual inspection would definitely fail as the generated sample can still look very sharp. In another case, the model can also underfit. For example, one can train a generative model on a dataset of trees and houses. However, the model while generation can create samples only showing trees. To the normal person, this is a successful generative model as it generates high quality samples of trees. However, it is required to know by the evaluator that the dataset that the model was trained on also contained images of houses which are not being generated at all. The model does not assign any probability to training images of houses. Realistically, when the model is trained on thousands and thousands of modes (statistically speaking), it may ignore a few modes which would be very difficult to detect by human observation as one cannot remember so many images to detect missing variations in the generated samples.

These reasons also partially explain why VAEs can achieve high data likelihood and be very close to the true posterior distribution and still generate blurry samples as opposed to GANs.

Even if we turn to the evaluation of the log-likelihood that the model assigns to test data we see that this method is not perfect either. Sometimes the log-likelihood may measure unimportant attributes and leave out the ones we want to be measured. Some models may achieve high log-likelihood due to the assignment of low variance to certain portions of the training data (say, background of the images) which will never change. We say it is a good thing to achieve high likelihood, but clearly, in this case, it is not.

Therefore there is a need in the field of generative models in machine learning to not just strive for better-generated samples but also to devise new ways of unbiasedly evaluating them.

7.5 Future Directions

Future directions of generative modelling may point to mixing up two or more generative models (HMM and LDA have been jointly used for stem cell research, topic modelling, and speech emotion recognition) [81-83] for a problem to overcome the drawbacks faced when deploying models individually. More interesting uses of generative models are *text-to-image* generation [84] using SGAN and stacking VAE and GAN for context aware text-to-image generation [85], with more instances of exploiting the VAE-GAN combination [86, 87], which are more examples of combining two generative models.

More research could be done on combining unsupervised generative models with supervised models (semi-supervision) to fine-tune (like we saw in Section 4.3 in DBMs) generation process and make it more efficient. It is noticed that while energy based models like BMs have been used in the field of recommender systems, the newest addition GANs have not been employed except in a few cases [88] and so applying these generative models (not only GANs) to recommendation systems is another future direction that can be taken by researchers. However, perhaps the biggest development required in generative modelling is a better way of interpretation of the generated samples as discussed in Section 7.4. The problem with VAEs is that it may spread probability mass where it may not make sense whereas GANs may miss modes of the true distribution. There have been initial approaches [89, 90] that try to improve this and there has been work that shows that directly optimizing likelihood can also generate high quality samples [91] which show the direction to future research in generative models.

Conclusion

In this paper, we provide a high level overview and analysis of all the generative models used in modern day applications by studying their ideology of operation, properties, advantages and disavantages. We compare all the models juxtaposed with each other based on their inference, sampling, probability evaluation, design and looked at the fields they are employed in to bring out some of the major differences between them. In addition, we implemented each discussed model with details of the results and our findings. It is worthy to mention that all the methods described in this paper are fields of active research in the literature and every day we see these generative models put to newer applications. This paper also points out the flaws in the evaluation of generated samples and provides future directions to the field of generative models. We hope that this survey provides readers a comprehensive, high-level and exhaustive read on all the generative models that exist and give them a fundamental along with a practical understanding of them.

References

[1] A. Likas, N. Vlassis, J. Verbeek, The global k-means clustering algorithm. *Pattern Recognition*, (2003) **36**(2), 451–461. doi:10.1016/s0031-3203(02)00060-2

[2] J.R. Hershey, P.A. Olsen, Approximating the Kullback Leibler Divergence Between Gaussian Mixture Models. *IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, (2007) doi:10.1109/icassp.2007.366913

[3] D. Povey, L. Burget, M. Agarwal, P. Akyazi, K. Feng, A. Ghoshal, *et al.* Subspace Gaussian Mixture Models for speech recognition. *IEEE International Conference on Acoustics, Speech and Signal Processing*, (2010) doi:10.1109/icassp.2010.5495662

[4] T. Chen, C. Huang, E. Chang, J. Wang, Automatic accent identification using Gaussian mixture models. *IEEE Workshop on Automatic Speech Recognition and Understanding*. ASRU '01, (2001) doi:10.1109/asru.2001.1034657

[5] P. Dupont, F. Denis, Y. Esposito, Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, (2005) **38**(9), 1349–1371. doi:10.1016/j.patcog.2004.03.020

[6] L.R. Bahl, P.F. Brown, P.V. de Souza, R.L. Mercer, Estimating hidden Markov model parameters so as to maximize speech recognition accuracy. *IEEE Transactions on Speech and Audio Processing*, (1993) **1**(1), 77–83. doi:10.1109/89.221369

[7] K.F. Lee, On large-vocabulary speaker-independent continuous speech recognition. *Speech Communication*, (1988) **7**(4), 375–379. doi:10.1016/0167-6393(88)90053-2

[8] L. Rabiner, B.H. Juang, Fundamentals of Speech Recognition, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[9] F. Jelinek, Statistical Methods for Speech Recognition, *MIT Press*, Cambridge, MA, 1998.

[10] R. Durbin, S. Eddy, S. Krogh, G. Mitchison, Biological Sequence Analysis, *Cambridge University Press*, Cambridge, 1998.

[11] P. Baldi, S. Brunak, Bioinformatics: A Machine Learning Approach, second ed., MIT Press, 2001.

[12] O.E. Agazzi, S. Kuo, Hidden markov model based optical character recognition in the presence of deterministic transformations. *Pattern Recognition*, (1993) **26**(12), 1813–1826. doi:10.1016/0031-3203(93)90178-y

[13] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent Dirichlet Allocation. *Journal of Machine Learning Research*, (2003) **3**, 993-1022. doi: 10.1162/jmlr.2003.3.4-5.993

[14] I. Bíró, J. Szabó, A.A. Benczúr, Latent dirichlet allocation in web spam filtering. *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web - AIRWeb '08.* (2008). doi:10.1145/1451983.1451991

[15] R. Krestel, P. Fankhauser, W. Nejdl, Latent dirichlet allocation for tag recommendation. *Proceedings of the Third ACM Conference on Recommender Systems - RecSys '09*, (2009). doi:10.1145/1639714.1639726

[16] S.K. Lukins, N.A. Kraft, L.H. Etzkorn, Bug localization using latent Dirichlet allocation. *Information and Software Technology*, (2010) **52**(9), 972–990. doi:10.1016/j.infsof.2010.04.002

[17] M. Lienou, H. Maitre, M. Datcu, Semantic Annotation of Satellite Images Using Latent Dirichlet Allocation. *IEEE Geoscience and Remote Sensing Letters*, (2010) **7**(1), 28–32. doi:10.1109/lgrs.2009.2023536

[18] O. Woodford, Notes on Contrastive Divergence. http://www.robots.ox.ac.uk/~ojw/files/NotesOnCD.pdf , 2006 (accessed 29 March 2020).

[19] Y. LeCun, A Tutorial on Energy-Based Learning. http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf , 2006 (accessed 30 March 2020).

[20] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, (1982) **79**(8), 2554–2558. doi:10.1073/pnas.79.8.2554

[21] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted Boltzmann machines for collaborative filtering. *Proceedings of the 24th International Conference on Machine Learning - ICML '07* (2007). doi:10.1145/1273496.1273596

[22] K. Georgiev, P. Preslav Nakov, A non-iid framework for collaborative filtering with restricted boltzmann machines, *ICML*, (2003) 1148–1156.

[23] Y.W. The, G.E. Hinton, G.E. Rate-coded restricted Boltzmann machines for face recognition. *Advances in Neural Information Processing Systems*, (2001) **13** 908–914.

[24] N. Jaitly, G. Hinton, Learning a better representation of speech soundwaves using restricted boltzmann machines. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* (2011) doi:10.1109/icassp.2011.5947700

[25] L.H. Chen, Z.H. Ling, Y. Song, L.R Dai, Joint spectral distribution modeling using restricted boltzmann machines for voice conversion, *INTERSPEECH-2013*, (2013) 3052-3056.

[26] A. Mohamed, G. Hinton, Phone recognition using Restricted Boltzmann Machines. *IEEE International Conference on Acoustics, Speech and Signal Processing*, (2010). doi:10.1109/icassp.2010.5495651

[27] L. Wang, K. Liu, Sentiment-Aspect Extraction based on Restricted Boltzmann Machines, *ACL*, (2015).

[28] R.M. Neal, Connectionist learning of belief networks. *Artificial Intelligence*, (1992) **56**(1), 71–113. doi:10.1016/0004-3702(92)90065-6

[29] G.E. Hinton, S. Osindero, Y. Teh, A fast learning algorithm for deep belief nets. *Neural Computation*, (2006) **18**, 1527–1554.

[30] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle. Greedy layerwise training of deep networks, *In Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS'07)*, (2007) 153–160.

[31] A.M. Abdel-Zaher, A. M. Eldeib, Breast cancer classification using deep belief networks. *Expert Systems with Applications*, (2016) **46**, 139–144. doi:10.1016/j.eswa.2015.10.015

[32] T. Kuremoto, S. Kimura, K. Kobayashi, M. Obayashi, Time series forecasting using a deep belief network with restricted Boltzmann machines. *Neurocomputing*, (2014) **137**, 47–56. doi:10.1016/j.neucom.2013.03.047

[33] X.L. Zhang, J. Wu, Deep Belief Networks Based Voice Activity Detection. *IEEE Transactions on Audio, Speech, and Language Processing*, (2013) **21**(4), 697–710. doi:10.1109/tasl.2012.2229986

[34] H. Lee, Y. Largman, P. Pham, A.Y. Ng, Unsupervised feature learning for audio classification using convolutional deep belief networks. *In Proceedings of the 22nd International Conference on Neural Information Processing Systems (NIPS'09)* (2009).

[35] Y. Chen, X. Zhao, X. Jia, Spectral–Spatial Classification of Hyperspectral Data Based on Deep Belief Network. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, (2015) **8**(6), 2381–2392. doi:10.1109/jstars.2015.2388577

[36] P. Liu, S. Han, Z. Meng, Y. Tong, Facial Expression Recognition via a Boosted Deep Belief Network. *IEEE Conference on Computer Vision and Pattern Recognition* (2014). doi:10.1109/cvpr.2014.233

[37] R. Salakhutdinov, G. Hinton, Deep Boltzmann Machines. *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, (2009) **5**, 448-455

[38] G. E. Hinton, T. Sejnowski, Optimal perceptual inference. *IEEE conference on Computer Vision and Pattern Recognition* (1983).

[39] R. Salakhutdinov, H. Larochelle, Efficient learning of deep boltzmann machines. *AISTATS. PMLR*, (2010) 693–700.

[40] N. Srivastava, R. Salakhutdinov, G. Hinton, Modeling documents with a Deep Boltzmann Machine. *In Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI'13). AUAI Press*, (2013) 616–624.

[41] N. Srivastava, R. Salakhutdinov, Multimodal learning with deep Boltzmann machines. *JMLR*, (2014) **15**, 1, 2949–2980.

[42] Y. Zhang, R. Salakhutdinov, H.A. Chang, J. Glass, Resource configurable spoken query detection using Deep Boltzmann Machines. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (2012). doi:10.1109/icassp.2012.6289082

[43] B. Leng, X. Zhang, M. Yao, Z. Xiong, A 3D model recognition mechanism based on deep Boltzmann machines. *Neurocomputing*, (2015) **151**, 593–602. doi:10.1016/j.neucom.2014.06.084

[44] W. Liu, R. Ji, S. Li, Towards 3D object detection with bimodal deep Boltzmann machines over RGBD imagery. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (2015) doi:10.1109/cvpr.2015.7298920

[45] C.N. Duong, K. Luu, K.G. Quach, T.D. Bui, Beyond Principal Components: Deep Boltzmann Machines for face modeling. *IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), (2015). doi:10.1109/cvpr.2015.7299111

[46] S. Sedhain, A.K. Menon, S. Sanner, L. Xie, AutoRec. *Proceedings of the 24th International Conference on World Wide Web - WWW '15 Companion*, (2015). doi:10.1145/2740908.2742726

[47] M.A. Carreira-Perpinan, R. Raziperchikolaei, *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (2015) 557-566

[48] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, K. Simonyan, Neural audio synthesis of musical notes with WaveNet autoencoders. *In Proceedings of the 34th International Conference on Machine Learning (ICML'17)*. JMLR.org, (2017) **70**, 1068–1077.

[49] D.P. Kingma, M. Welling, Auto-Encoding Variational Bayes. *ICLR* (2014).

[50] J. Walker, C. Doersch, A. Gupta, M. Hebert, An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders. *Lecture Notes in Computer Science*, (2016) 835–851. doi:10.1007/978-3-319-46478-7_51

[51] D. Liang, R.G. Krishnan, M.D. Hoffman, T. Jebara, Variational Autoencoders for Collaborative Filtering. *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, (2018). doi:10.1145/3178876.3186150

[52] A. Roberts, J. Engel, D. Eck, Hierarchical variational autoencoders for music. *NIPS Workshop on Machine Learning for Creativity and Design*, 2017.

[53] M. Blaauw, J. Bonada, Modeling and transforming speech using variational autoencoders. Morgan N, editor. *Interspeech*; *ISCA*, (2016) 1770-4.

[54] S. Latif, R. Rana, J. Qadir, J. Epps, Variational Autoencoders for Learning Latent Representations of Speech Emotion. ArXiv, abs/1712.08708, 2017.

[55] Q. Liu, M. Allamanis, M. Brockschmidt, A. Gaunt, Constrained graph variational autoencoders for molecule design. *Advances in Neural Information Processing Systems*, (2018) 7795-7804.

[56] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets. *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14). MIT Press*, (2014) **2**, 2672–2680.

[57] A. Radford, L. Metz, S. Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR*, abs/1511.06434, 2015.

[58] J.T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806, 2014.

[dataset] [59] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, 2009. https://www.cs.toronto.edu/~kriz/cifar.html

[60] V. Nair, G. Hinton, Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10). Omnipress*, (2010) 807–814.

[61] A.L. Maas, A.Y Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models. *In Proc. ICML*, (2013) **30**(1), 3.

[62] B. Xu, N. Wang, T. Chen, M. Li, Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853, 2015.

[63] S. Barua, S.M. Erfani, J. Bailey, FCC-GAN: A Fully Connected and Convolutional Net Architecture for GANs. arXiv preprint arXiv:1905.02417, 2019.

[64] M. Mirza, S. Osindero, Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784, 2014.

[65] J. Gauthier, Conditional generative adversarial nets for convolutional face generation. Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester, (2014) **5**, 2.

[66] G. Antipov, M. Baccouche, J.L. Dugelay, Face aging with conditional generative adversarial networks. *IEEE international conference on image processing (ICIP)*, (2017) 2089-2093.

[67] H. Zhang, V. Sindagi, V.M. Patel, Image De-raining Using a Conditional Generative Adversarial Network. *IEEE Transactions on Circuits and Systems for Video Technology*, (2019) **1**, 1. doi:10.1109/tcsvt.2019.2920407

[68] D. Michelsanti, Z.H. Tan, Conditional generative adversarial networks for speech enhancement and noise-robust speaker verification. arXiv preprint arXiv:1709.01703, 2017.

[69] P. Shen, X. Lu, S. Li, H. Kawai, Conditional Generative Adversarial Nets Classifier for Spoken Language Identification. *INTERSPEECH*, (2017) 2814-2818.

[70] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, S. Belongie, Stacked generative adversarial networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (2017) 5077-5086.

[71] I. Goodfellow, Generative adversarial networks. *NIPS tutorial*. arXiv preprint arXiv:1701.00160, 2016.

[72] B. Uria, M.A. Côté, K. Gregor, I. Murray, H. Larochelle, Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, (2016) **17**(1), 7184-7220.

[73] M. Germain, K. Gregor, I. Murray, H. Larochelle, Made: Masked autoencoder for distribution estimation. *International Conference on Machine Learning*, (2015) 881-889.

[74] A.V.D. Oord, N. Kalchbrenner, K. Kavukcuoglu, Pixel recurrent neural networks. arXiv preprint arXiv:1601.06759, 2016.

[75] B. J. Frey, G.E. Hinton, P. Dayan, Does the wake-sleep algorithm produce good density estimators? *Advances in neural information processing systems*, (1996) 661-667.

[76] B.J. Frey, Graphical models for machine learning and digital communication. *MIT Press*, Cambridge, 1998.

[77] Y. Bengio, E. Laufer, G. Alain, J. Yosinski, Deep generative stochastic networks trainable by backprop. *International Conference on Machine Learning*, (2014) 226-234.

[78] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, R. Harshman, Indexing by latent semantic analysis. *Journal of the American society for information science*, (1990) **41**(6), 391-407.

[79] T. Hofmann, Probabilistic latent semantic indexing. *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, (1999) 50-57.

[80] L. Hong, B.D. Davison, Empirical study of topic modeling in twitter. *Proceedings of the first workshop on social media analytics*, (2010) 80-88.

[81] Q. Wu, C. Zhang, Q. Hong, L. Chen, Topic evolution based on LDA and HMM and its application in stem cell research. *Journal of Information Science*, (2014) **40**(5), 611–620. doi:10.1177/0165551514540565

[82] B.J.P. Hsu, J. Glass, J, Style & topic language model adaptation using HMM-LDA. *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, (2006) 373-381.

[83] A. Bansal, S. Chaudhary, S.D. Roy, A Novel LDA and HMM-Based Technique for Emotion Recognition from Facial Expressions. *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction*, (2013) 19–26. doi:10.1007/978-3-642-37081-6_3

[84] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, D.N. Metaxas, Stackgan:

Text to photo-realistic image synthesis with stacked generative adversarial networks. *Proceedings of the IEEE international conference on computer vision*, (2017) 5907-5915.

[85] C. Zhang, Y. Peng, Stacking VAE and GAN for Context-aware Text-to-Image Generation. *IEEE Fourth International Conference on Multimedia Big Data (BigMM)*, 2018. doi:10.1109/bigmm.2018.8499439

[86] A.B.L. Larsen, S.K. Sønderby, H. Larochelle, O. Winther, Autoencoding beyond pixels using a learned similarity metric. arXiv preprint arXiv:1512.09300, 2015.

[87] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, B. Frey, Adversarial autoencoders. arXiv preprint arXiv:1511.05644, 2015.

[88] A.V. Prosvetov, GAN for Recommendation System. *Journal of Physics: Conference Series*, (2019) **1405**(1), 012005.

[89] Y. Li, K. Swersky, R. Zemel, Generative moment matching networks. *International Conference on Machine Learning*, (2015) 1718-1727.

[90] S. Nowozin, B. Cseke, R. Tomioka, f-gan: Training generative neural samplers using variational divergence minimization. *Advances in neural information processing systems*, (2016) 271-279.

[91] L. Dinh, J. Sohl-Dickstein, S. Bengio, Density estimation using real nvp. arXiv preprint arXiv:1605.08803, 2016.

[92] H. G.M., GenerativeModels, (2020), GitHub repository: https://github.com/GM-git-dotcom/GenerativeModels

[93] D. Dua, C. Graff (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[94] F.M. Harper, J.A. Konstan, ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19, December 2019.

[95] Y. LeCun, C. Cortes, C.J. Burges, MNIST handwritten digit database (2010).

[96] W. Fang, F. Zhang, V.S Sheng, Y. Ding, A method for improving CNN-based image recognition using DCGAN. *CMC: Comput. Mater. Continua*, (2018) ***57***(1), 167-178.

[97] H. Heo, Y. Hwang, Automatic Sketch Colorization using DCGAN. In *2018 18th International Conference on Control, Automation and Systems (ICCAS)* (2018, November), 1316-1318.

[98] W. Fang, Y. Ding, F. Zhang, J. Sheng, Gesture recognition based on CNN and DCGAN for calculation and text output. *IEEE Access*, (2019), **7**, 28230-28237.

[99] D.D. Kim, M.T. Shahid, Y. Kim, W.J. Lee, H.C. Song, F. Piccialli, K.N. Choi, Generating Pedestrian Training Dataset using DCGAN. In *Proceedings of the 2019 3rd International Conference on Advances in Image Processing*, (2019, November) 1-4.

[100] P.L. Suárez, A.D. Sappa, B.X. Vintimilla, Infrared image colorization based on a triplet dcgan architecture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, (2017) 18-23.

[101] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, A.C. Berg, Ssd: Single shot multibox detector. In *European conference on computer vision*, (2016) 21-37.

[102] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (2016) 779-788.

[103] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, (2015) 91-99.

[104] K. He, G. Gkioxari, P. Dollár, R. Girshick, Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, (2017) 2961-2969.

[105] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, ..., M.P. Lungren, Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, (2017).

[106] M.A. Nayak, S. Ghosh, Prediction of extreme rainfall event using weather pattern recognition and support vector machine classifier. *Theoretical and applied climatology*, (2013), **114**(3-4), 583-603.

[107] B. Yang, L.X. Li, H. Ji, J. Xu. An early warning system for loan risk assessment using artificial neural networks. *Knowledge-Based Systems*, (2001), **14**(5-6), 303-306.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.