UNIT-5

AUTOENCODERS & DEEP GENERATIVE MODELS

Definition of Autoencoders:

Autoencoders are powerful because their capacity is reduced—the number of nodes in the hidden layers is reduced along with the number of nodes in the information bottleneck. It is that way because even if the bottleneck consists of only one dimension, it's still possible for the autoencoder to copy the input to the output, without extracting any information, when the capacity of the model is high.

Types of autoencoders:

There are, basically, 7 types of autoencoders:

1. Undercomplete Autoencoder
2. Sparse Autoencoder
3. Denoising Autoencoder
4. Contractive Autoencoder
5. Deep Autoencoder
6. Convolutional Autoencoder
7. Variational Autoencoder

1. Undercomplete Autoencoder:

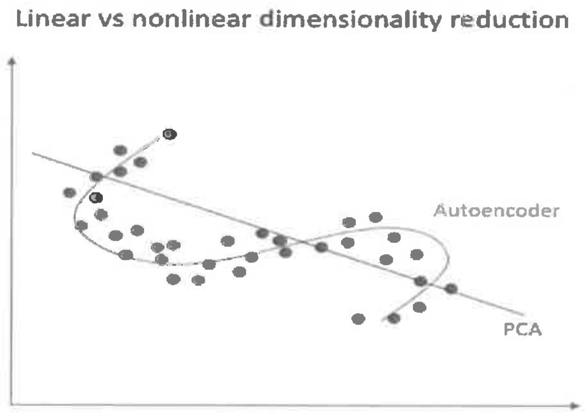
The objective of undercomplete autoencoder is to capture the most important features present in the data. Undercomplete autoencoders have a smaller dimension for hidden layer compared to the input layer. This helps to obtain important features from the data. It minimizes the loss function by penalizing the g(f(x)) for being different from the input x.

$$L(x, g(f(x))),$$

Where both *f* and *g* are nonlinear functions.

An undercomplete autoencoder is one of the simplest types of autoencoders. The way it works is very straightforward— Undercomplete autoencoder takes in an image and tries to predict the same image as output, thus reconstructing the image from the compressed bottleneck region. Undercomplete autoencoders are truly unsupervised as they do not take any form of label, the target being the same as the input. The primary use of autoencoders like such is the generation of the latent space or the bottleneck, which forms a compressed substitute of the input data and can be easily decompressed back with the help of the network when needed. This form of compression in the data can be modeled as a form of **dimensionality reduction**.

When we think of dimensionality reduction, we tend to think of methods like PCA (Principal Component Analysis) that form a lower-dimensional hyperplane to represent data in a higher-dimensional form without losing information. However— PCA can only build linear relationships. As a result, it is put at a disadvantage compared with methods like undercomplete autoencoders that can learn non-linear relationships and, therefore, perform better in dimensionality reduction. This form of nonlinear dimensionality reduction where the autoencoder learns a non-linear manifold is also termed as ***manifold learning***. Effectively, if we remove all non-linear activations from an undercomplete autoencoder and use only linear layers, we reduce the undercomplete autoencoder into something that works at an equal footing with PCA..



Visual representation of the difference between Autoencoder and PCA

The loss function used to train an undercomplete autoencoder is called ***reconstruction loss***, as it is a check of how well the image has been reconstructed from the input data.

Advantages-

- Undercomplete autoencoders do not need any regularization as they maximize the probability of data rather than copying the input to the output.

Drawbacks-

- Undercomplete autoencoders are not versatile and they tend to overfit.

Regularized Auto encoder: Regularised autoencoders are designed based on data complexity, and they address the problems of Undercomplete autoencoders. Regularization in Deep Learning is very important to overcome overfitting.

Regularised autoencoders use a loss function for properties like:

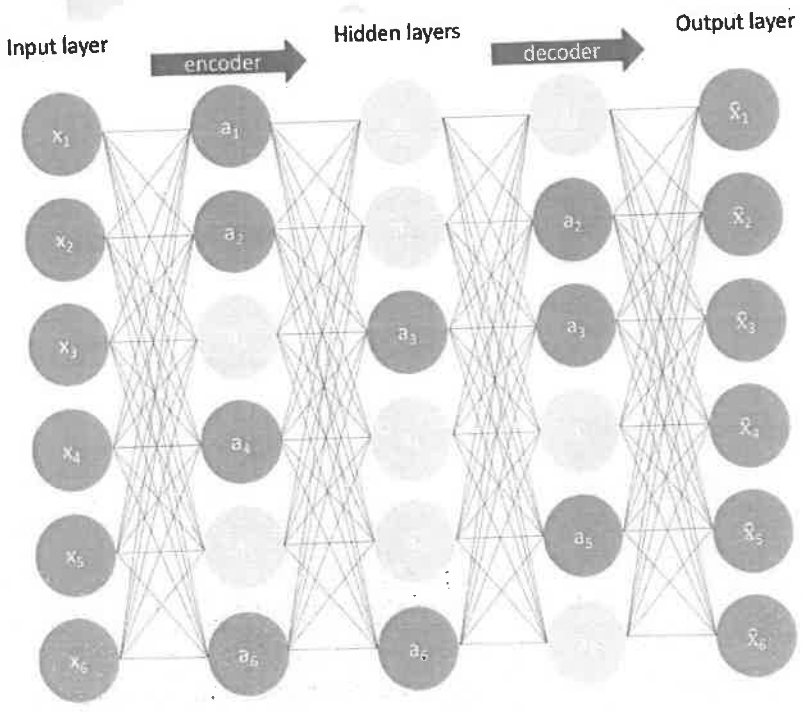
1. The ability to reconstruct the output from the input through approximation.
2. The sparsity of representation.
3. The smallness of the derivative of the representation.
4. Robustness to noise, outliers, or missing inputs.

Types of Regularized Auto encoder:

2. Sparse Autoencoder
3. Denoising Autoencoder
4. Contractive Autoencoder

2) Sparse Autoencoder:

A sparse autoencoder is simply an autoencoder whose training criterion involves a **sparsity penalty**.Sparse autoencoders have hidden nodes greater than input nodes. They can still discover important features from the data. A generic sparse autoencoder is visualized where the insignificance of a node corresponds with the level of activation. Sparsity constraint is introduced on the hidden layer. This is to prevent output layer copy input data. Sparsity may be obtained by additional terms in the loss function during the training process, either by comparing the probability distribution of the hidden unit activations with some low desired value, or by manually zeroing all but the strongest hidden unit activations. There are actually two different ways to construct our sparsity penalty:
L1 regularization and **KL-divergence.**



Dr. Prathyusha. Kuncha
NRIIT

Advantages-

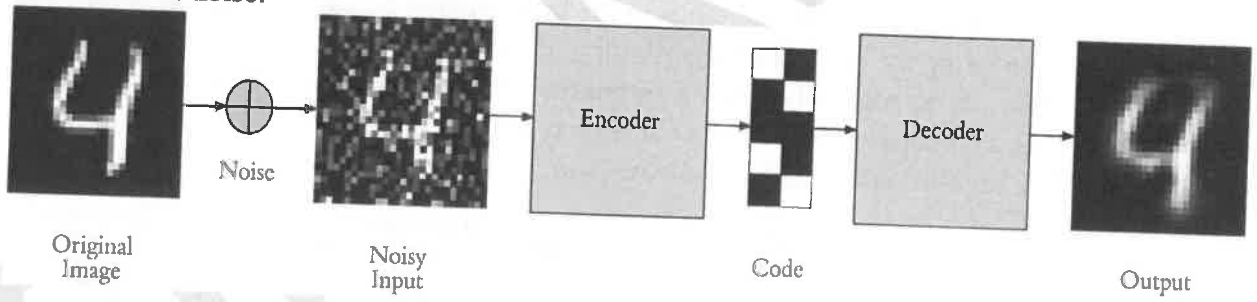
- Sparse autoencoders have a sparsity penalty, a value close to zero but not exactly zero. Sparsity penalty is applied on the hidden layer in addition to the reconstruction error. This prevents overfitting.
- They take the highest activation values in the hidden layer and zero out the rest of the hidden nodes. This prevents autoencoders to use all of the hidden nodes at a time and forcing only a reduced number of hidden nodes to be used.

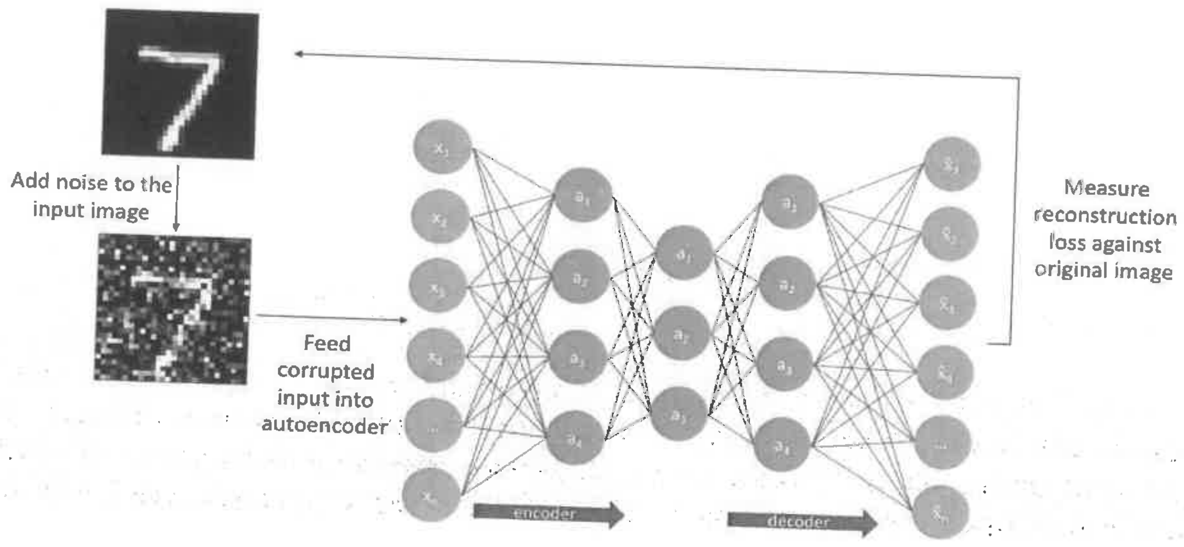
Drawbacks-

- For it to be working, it's essential that the individual nodes of a trained model which activate are data dependent, and that different inputs will result in activations of different nodes through the network.

3. Denoising Autoencoder:

Denoising autoencoders create a corrupted copy of the input by introducing some noise. This helps to avoid the autoencoders to copy the input to the output without learning features about the data. These autoencoders take a partially corrupted input while training to recover the original undistorted input. The model learns a vector field for mapping the input data towards a lower dimensional manifold which describes the natural data to cancel out the added noise.




Add noise to the input image
Feed corrupted input into autoencoder
Measure reconstruction loss against original image
encoder
decoder

Dr. Prathyusha. Kuncha
NRIIT

Advantages-

- Corruption of the input can be done randomly by making some of the input as zero. Remaining nodes copy the input to the noised input.
- Setting up a single-thread denoising autoencoder is easy.

Drawbacks:

- To train an autoencoder to denoise data, it is necessary to perform preliminary stochastic mapping in order to corrupt the data and use as input.
- This model isn't able to develop a mapping which memorizes the training data because our input and target output are no longer the same.

4) Contractive Autoencoder:

Contractive autoencoder is another regularization technique just like sparse and denoising autoencoders. The objective of a contractive autoencoder is to have a robust learned representation which is less sensitive to small variation in the data. Robustness of the representation for the data is done by applying a penalty term to the loss function. However, this regularizer corresponds to the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input. Frobenius norm of the Jacobian matrix for the hidden layer is calculated with respect to input and it is basically the sum of square of all elements. The contractive autoencoder also has a regularization term to prevent the network from learning the identity function and mapping input into output. To train a model that works along with this constraint, we need to ensure that the derivatives of the hidden layer activations are small concerning the input.

$$\delta h / \delta x$$

Contractive autoencoders work on the basis that similar inputs should have similar encodings and a similar latent space representation. It means that the latent space should not vary by a huge amount for minor variations in the input. While the reconstruction loss wants the model to tell differences between two inputs and observe variations in the data, the frobenius norm of the derivatives says that the model should be able to ignore variations in the input data.

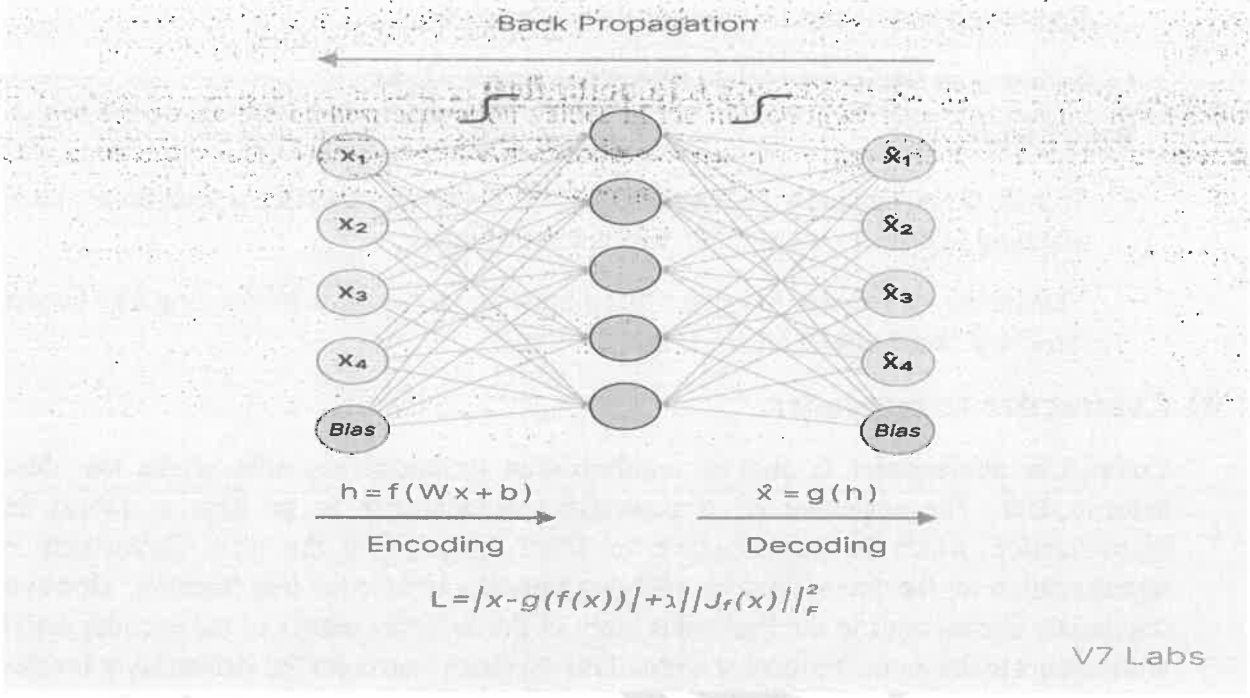
The total loss function can be mathematically expressed as:

$$L = \left|x - \widehat{x}\right| + \lambda \sum_i \left|\left|\nabla_x a_i^{(h)}(x)\right|\right|^2$$

Where *h>* is the hidden layer for which a gradient is calculated and represented with respect to the input x as $\nabla_x a_i^{(h)}(x)$.

Dr. Prathyusha. Kuncha
NRIIT

The gradient is summed over all training samples, and a frobenius norm of the same is taken.



Advantages-

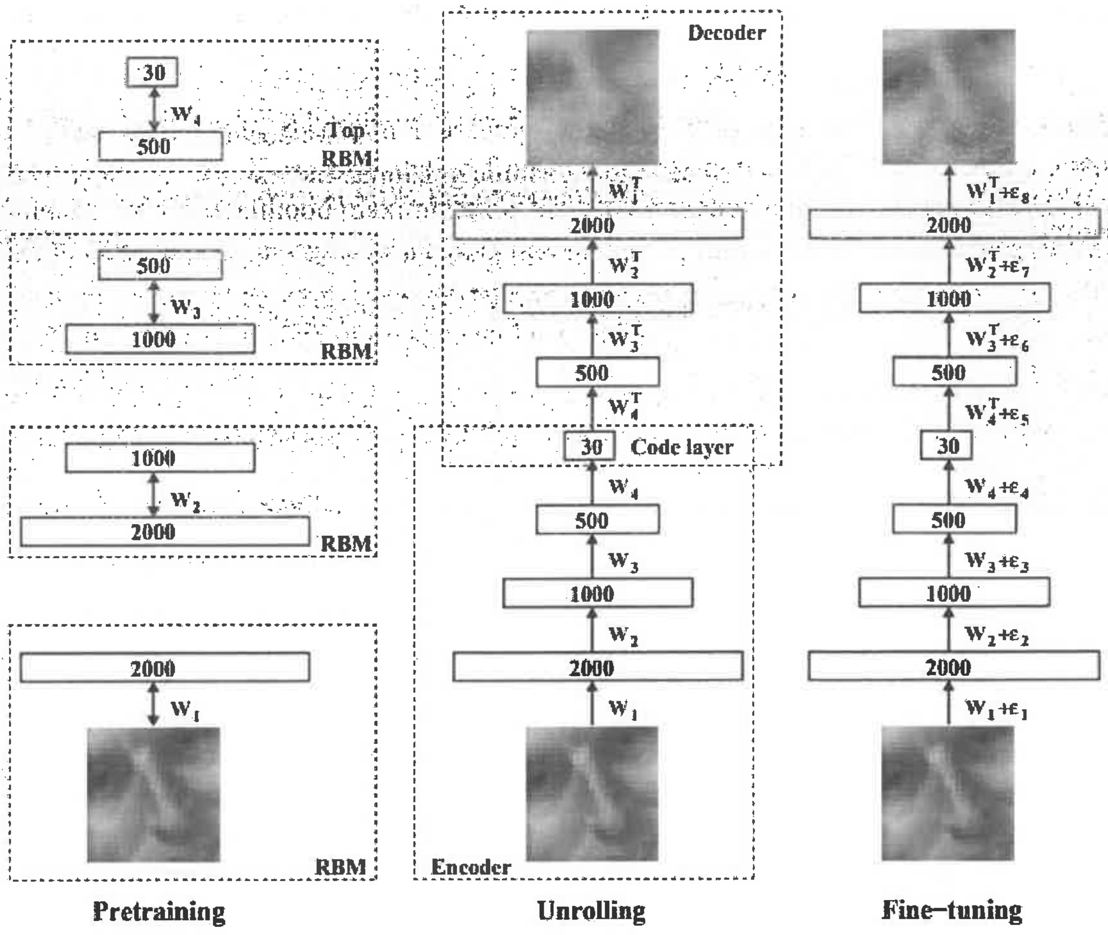
- Contractive autoencoder is a better choice than denoising autoencoder to learn useful feature extraction.
- This model learns an encoding in which similar inputs have similar encodings. Hence, we're forcing the model to learn how to contract a neighborhood of inputs into a smaller neighborhood of outputs.

Drawback:

Suffers with major drawback i.e: Reconstruction Error.

5. Deep Autoencoder:

A deep autoencoder is a feed-forward neural network that uses hidden layers to transfer input neurons to output neurons. A deep autoencoder is composed of two symmetrical deep-belief networks having four to five shallow layers. They have more layers than a simple autoencoder and thus are able to learn more complex features. The layers are restricted Boltzmann machines, the building blocks of deep-belief networks. It allows for more efficient code learning without increasing the number of nodes per layer. This reduces the amount of training data needed, which in turn reduces training computation. Processing the benchmark dataset MNIST, a deep autoencoder would use binary transformations after each RBM. Deep autoencoders are useful in topic modeling, or statistically modeling, abstract topics that are distributed across a collection of documents. They are also capable of compressing images into 30 number vectors.



Advantages-

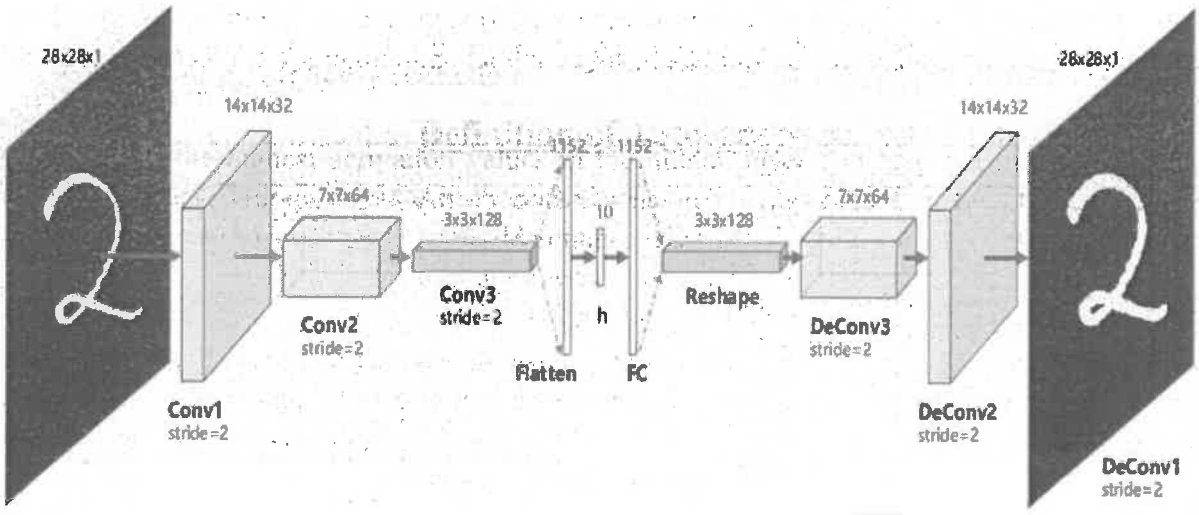
- Deep autoencoders can be used for other types of datasets with real-valued data, on which you would use Gaussian rectified transformations for the RBMs instead.

Drawbacks-

- Chances of underfitting to occur.

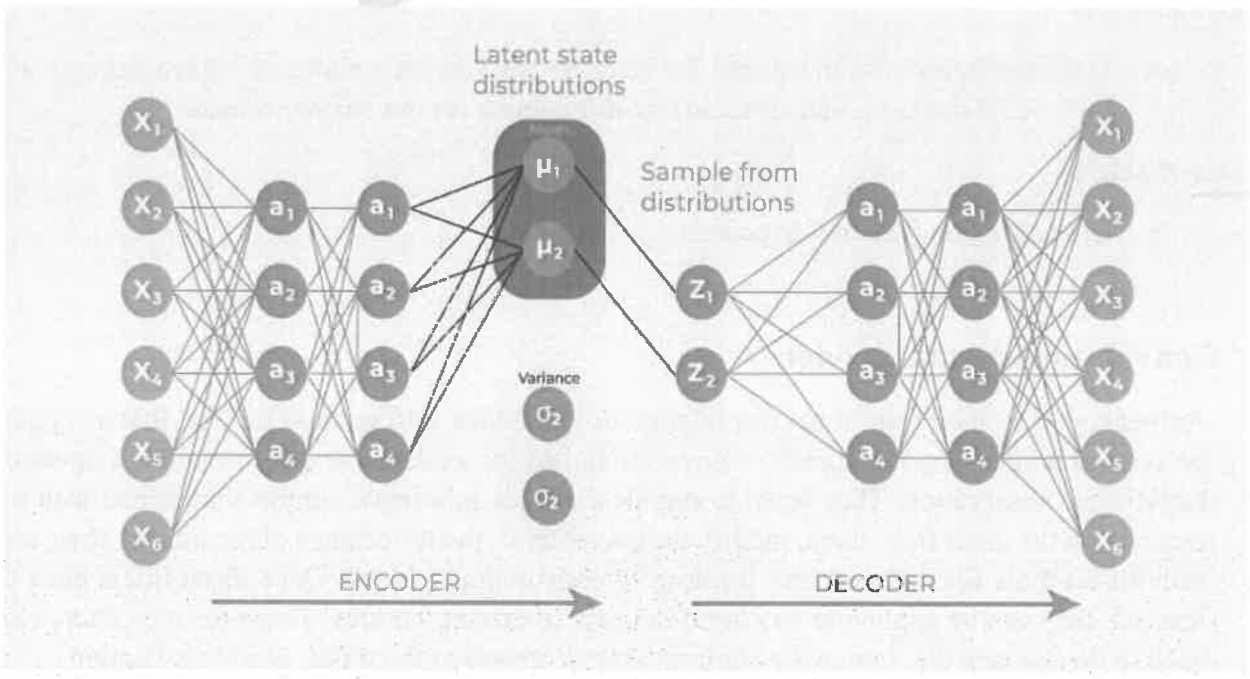
7. Convolutional Autoencoder

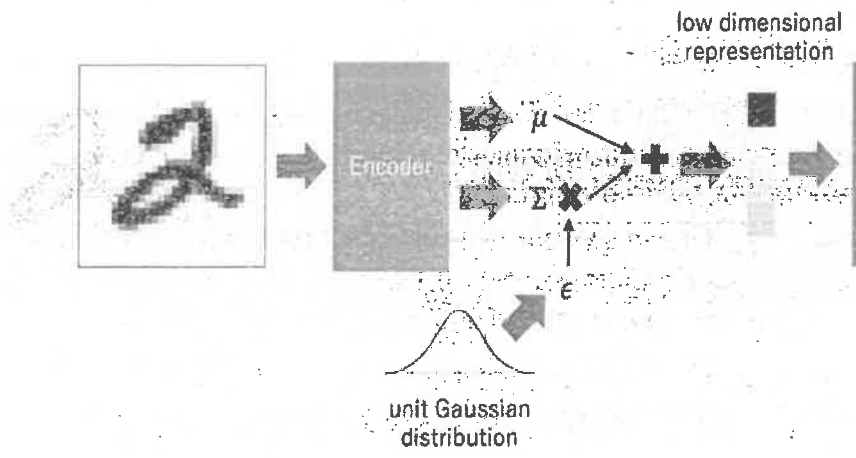
Autoencoders in their traditional formulation does not take into account the fact that a signal can be seen as a sum of other signals. Convolutional Autoencoders use the convolution operator to exploit this observation. They learn to encode the input in a set of simple signals and then try to reconstruct the input from them, modify the geometry or the reflectance of the image. They are the state-of-art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input in order to extract features. These features, then, can be used to do any task that requires a compact representation of the input, like classification.



8. Variational Autoencoder:

A variational autoencoder (VAE) provides a probabilistic manner for describing an observation in latent space. Thus, rather than building an encoder that outputs a single value to describe each latent state attribute, we'll formulate our encoder to describe a probability distribution for each latent attribute. It has many applications, such as data compression, synthetic data creation, etc.Variational autoencoder is different from an autoencoder in a way that it provides a statistical manner for describing the samples of the dataset in latent space. Therefore, in the variational autoencoder, the encoder outputs a probability distribution in the bottleneck layer instead of a single output value.





Advantages

1. Variational Autoencoders are used to generate new data points that resemble the original training data. These samples are learned from the latent space.
2. Variational Autoencoder is probabilistic framework that is used to learn a compressed representation of the data that captures its underlying structure and variations, so it is useful in detecting anomalies and data exploration.

Disadvantages

1. Variational Autoencoder use approximations to estimate the true distribution of the latent variables. This approximation introduces some level of error, which can affect the quality of generated samples.
2. The generated samples may only cover a limited subset of the true data distribution. This can result in a lack of diversity in generated samples.

Applications:

- Image generation
- Data generation
- Anomaly detection
- Data imputation, and more.

Stochastic Encoders and Decoders

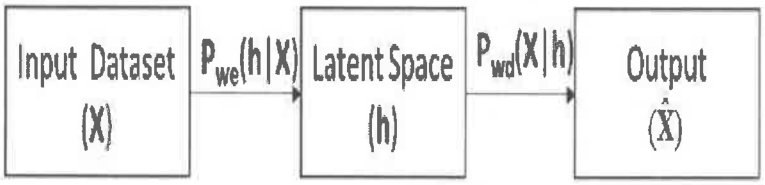
Building and comparing stochastic encoders and decoders

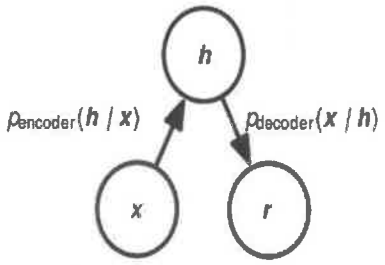
Stochastic encoders fall into the domain of generative modeling, where the objective is to learn joint probability P(X) over given data X transformed into another high-dimensional space. For example, we want to learn about images and produce similar, but not exactly the same, images by learning about pixel dependencies and distribution. One of the popular approaches in generative modeling is **Variational autoencoder (VAE)**, which combines deep learning with statistical inference by making a strong distribution assumption on h ~ P(h), such as Gaussian or Bernoulli. For a given weight W, the X can be sampled from the distribution as $P_w(X|h)$. The cost function of VAE is based on log likelihood maximization. The cost function consists of reconstruction and regularization error terms:

$$\text{Cost} = \text{Reconstruction Error} + \text{Regularization Error}$$

Stochastic encoder

We can also generalize the notion of an encoding function $f(\boldsymbol{x})$ to an encoding distribution $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$





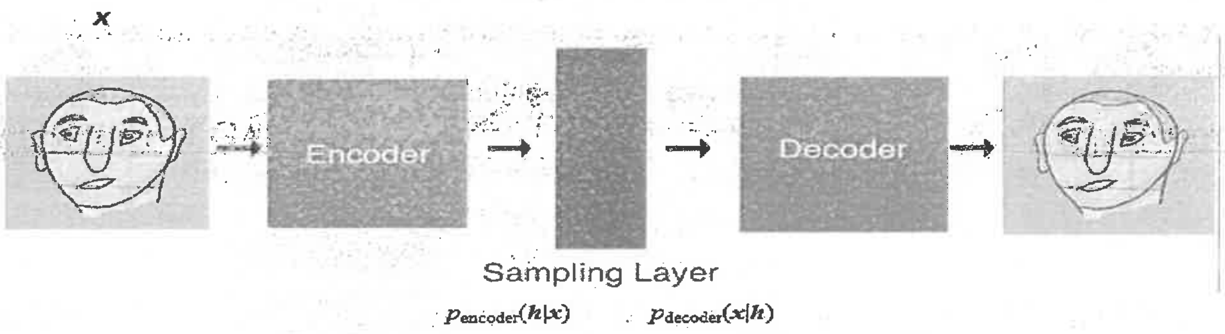
Loss function for Stochastic Decoder

- Given a hidden code $\boldsymbol{h}$, we may think of the decoder as providing a conditional distribution $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- We train the autoencoder by minimizing $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- The exact form of this loss function will change depending on the form of $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$

Relationship to joint distribution

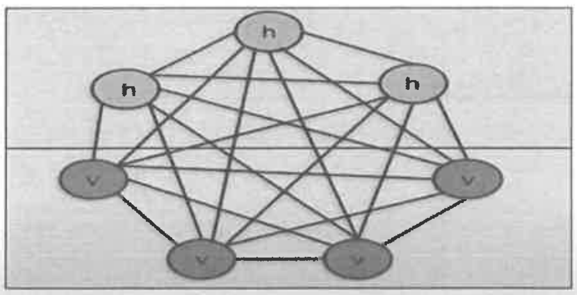
- General strategy for designing the output units and loss function of a feedforward network is to
  - ➢ Define the output distribution $p(\boldsymbol{y}|\boldsymbol{x})$
  - ➢ Minimize the negative log-likelihood $-\log p(\boldsymbol{y}|\boldsymbol{x})$
  - ➢ In this setting $\boldsymbol{y}$ is a vector of targets such as class labels
- In an autoencoder $\boldsymbol{x}$ is the target as well as the input.
- Any latent variable model $p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$ defines a stochastic encoder $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})=p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$
- And a stochastic decoder $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})=p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$

Sampling $p_{\text{model}}(h|x)$



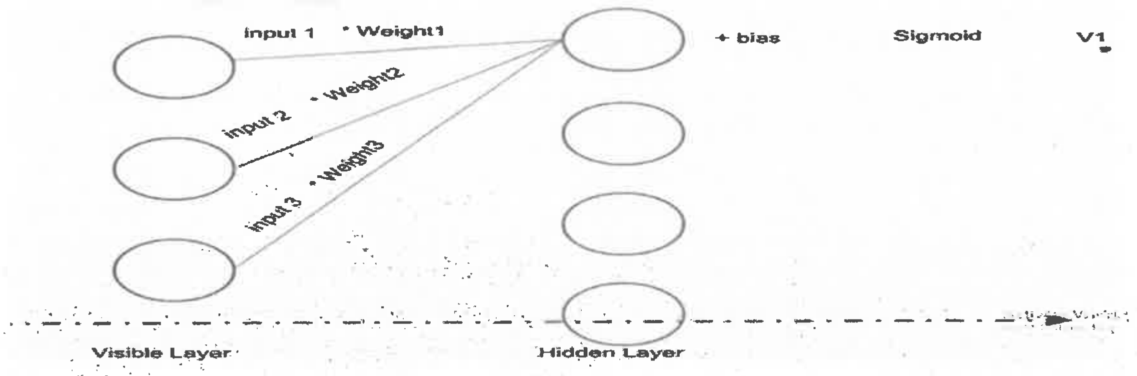
Boltzmann Learning:

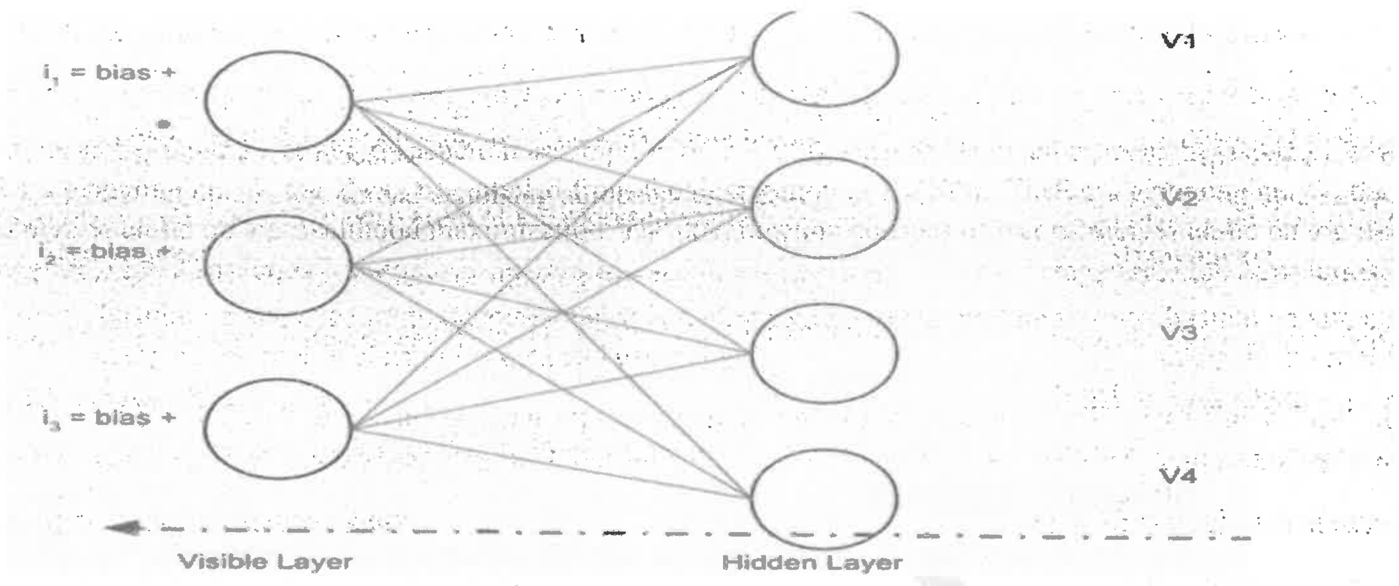
A neural network based on Boltzmann learning rule is called as a Boltzmann Machine A BM is a network of symmetrically connected, neurons like units that make stochastic (Random Probability Distribution) decisions about whether to be on or off. It operates by randomly choosing a neuron and flipping its state. These machines are not deterministic deep learning models, they are stochastic or generative deep learning models. They are representations of a system. A Boltzmann machine is an unsupervised deep learning model in which every node is connected to every other node. It is a type of recurrent neural network, and the nodes make binary decisions with some level of bias.



v - visible nodes, h - hidden nodes

How BM learns





The Boltzmann machine is characterised by an energy function

$$E = -\frac{1}{2}\sum_{j}\sum_{k} w_{kj}x_k x_j$$

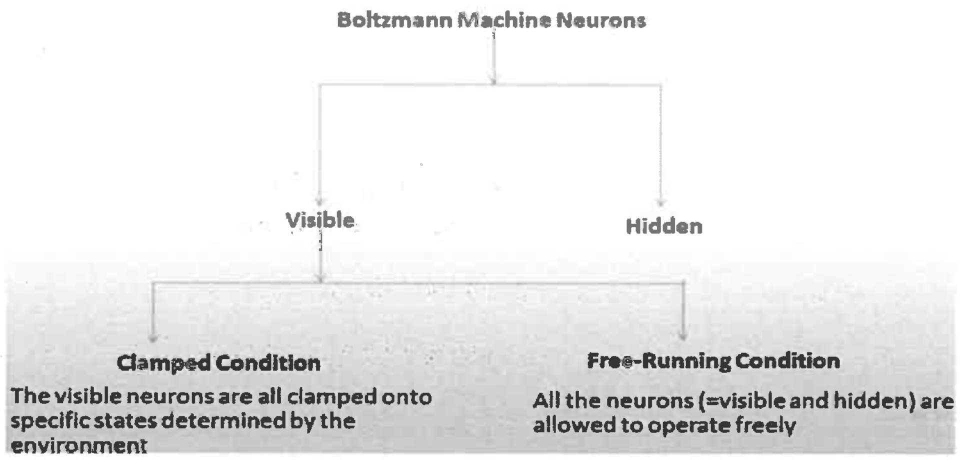
$$where,\ j \neq k$$

$$x_k \text{ is state of neuron, } k$$

Probability of State change

$$P = \frac{1}{1 + exp(^{-\Delta E_k/T})}$$

Operating Conditions



Clamped Condition
The visible neurons are all clamped onto specific states determined by the environment

Free-Running Condition
All the neurons (=visible and hidden) are allowed to operate freely

Dr. Prathyusha. Kuncha
NRIIT

The Boltzmann learning rule:

$$\Delta w_{kj} = \eta(\rho^{+}_{kj} - \rho^{-}_{kj}),\ j \neq k,$$

η is the learning rate parameter

note that both ρ^{+}_{kj} and ρ^{-}_{kj} range in value from –1 to +1.

Types of Boltzmann machines:

There are three types of Boltzmann machines. These are:

1. Restricted Boltzmann Machines (RBMs)
2. Deep Belief Networks (DBNs)
3. Deep Boltzmann Machines (DBMs)

Applications of BM:

- Dimensionality reduction
- Classification
- Collaborative filtering
- Feature learning
- Topic modelling, etc...

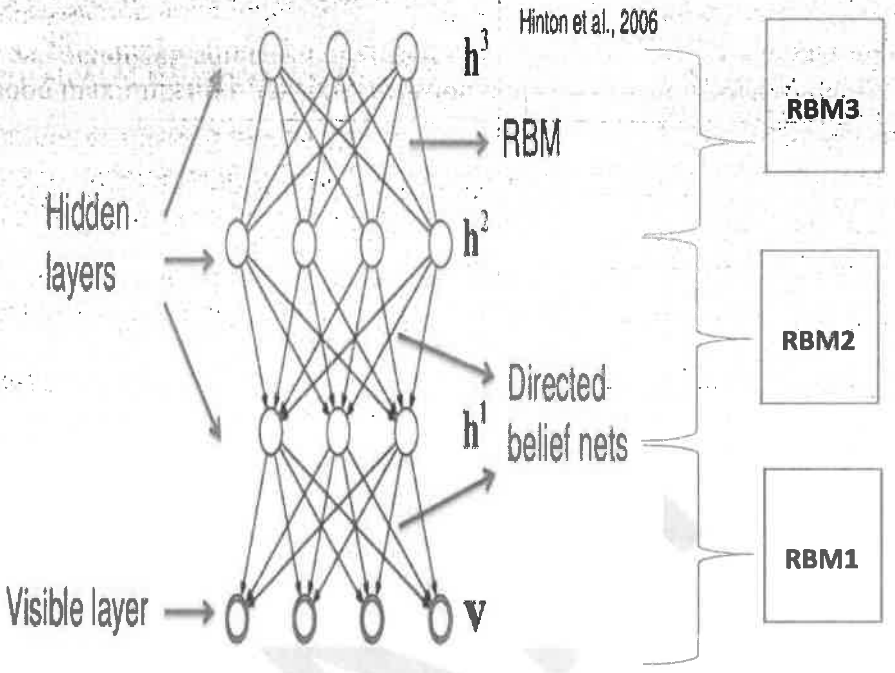
Deep Belief Networks:

Architecture of DBN:

- **Network Structure:** DBN identical to MLP.
- **Training:** DBN differs with MLP.

One problem with traditional multilayer perceptron's / ANN is that backpropagation can often lead to "local minima". DBN solve this problem by using an extra step called pre-training. The connection in the top layers are undirected and associative memory is formed from the connections between them. The connections in the lower levels are directed. Each RBM layer learns the entire input. The idea of pre-training came from work on DBN's. It is a composition of stack of unsupervised networks such as RBM. The hidden layer in stack 1 is the visible layer for the stack 2. DBN has connections within RBM an not between RBMs. Each RBM network is trained independently with the greedy algorithm.

DBN structure



Applications:

- Image recognition
- Video recognition
- Motion – capture data

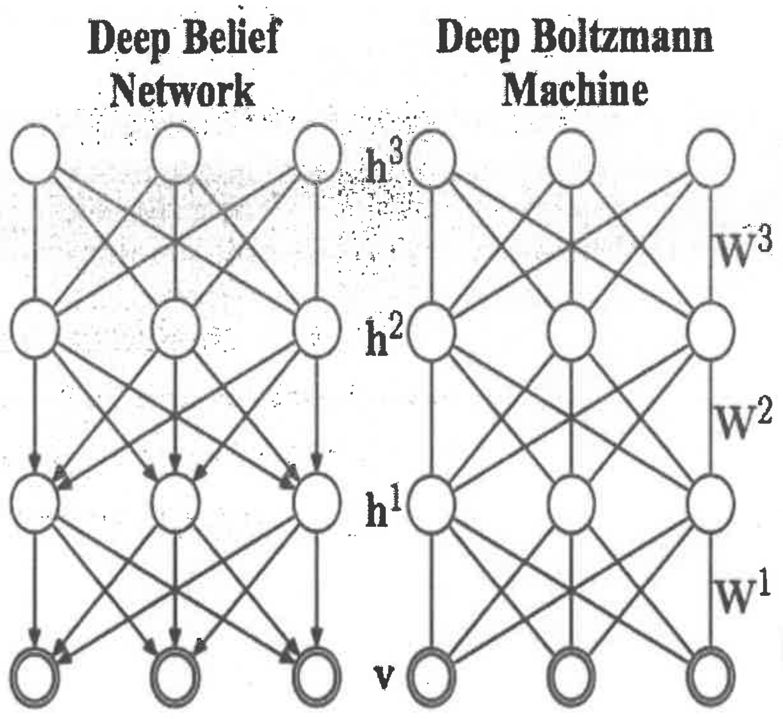
Deep Boltzmann Machines:

A **Deep Boltzmann Machine (DBM)** is a three-layer generative model. It is similar to a Deep Belief Network, but instead allows bidirectional connections in the layers. DBM is Unsupervised, probabilistic, generative model with entirely undirected connections between different layers. Deep Boltzmann machines expand upon restricted Boltzmann machines by adding additional hidden layers. The architectural properties of entirely symmetrical edges, as well as no connections between nodes in the same layer are retained. Its energy function is as an extension of the energy function of the RBM:

$$E(v,h) = -\sum_{i}^{i} v_i b_i - \sum_{n=1}^{N}\sum_{k} h_{n,k} b_{n,k} - \sum_{i,k} v_i w_{ik} h_k - \sum_{n=1}^{N-1}\sum_{k,l} h_{n,k} w_{n,k,l} h_{n+1,l}$$

for a DBM with N hidden layers.

Dr. Prathyusha. Kuncha
NRIIT



Training of DBMs is often done in 2 stages:

- **A pre-training stage:** where every RBM is trained independently.
- **A fine tuning stage:** where the network is trained at once using backpropagation.

Applications:

- Generative modelling
- Unsupervised feature learning
- Representation learning
- Collaborative learning
- NLP
- Image generation and restoration

Generative Adversarial Networks (GANs)

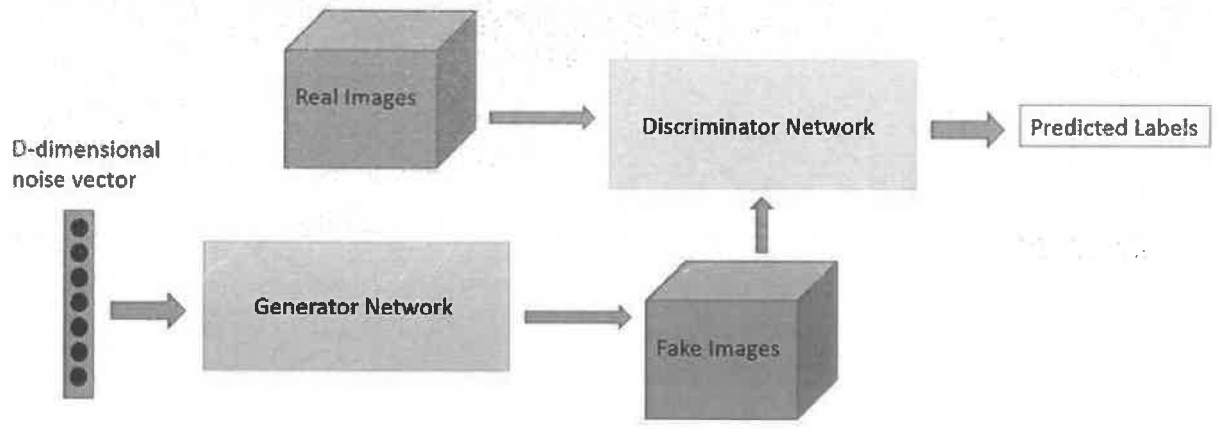
Generative Adversarial Networks (GANs) can be broken down into three parts:

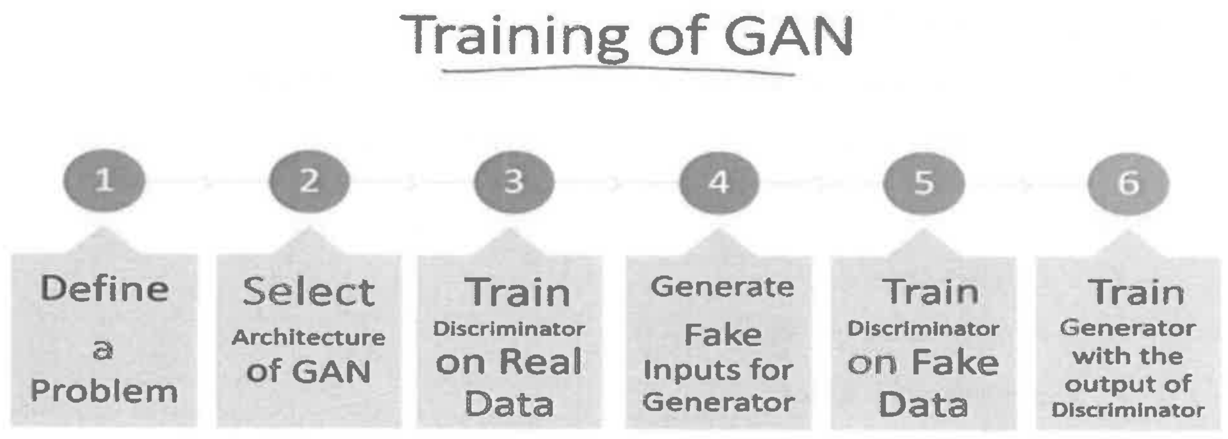
- **Generative:** To learn a generative model, which describes how data is generated in terms of a probabilistic model.
- **Adversarial:** The word adversarial refers to setting one thing up against another. This means that, in the context of GANs, the generative result is compared with the actual

images in the data set. A mechanism known as a discriminator is used to apply a model that attempts to distinguish between real and fake images.

➤ **Networks:** Use deep neural networks as artificial intelligence (AI) algorithms for training purposes.

Architecture of GAN:




Training of GAN
1
Define a Problem
2
Select Architecture of GAN
3
Train Discriminator on Real Data
4
Generate Fake Inputs for Generator
5
Train Discriminator on Fake Data
6
Train Generator with the output of Discriminator

Steps to Train Generative Adversarial Networks

Step 1: Define the problem

- Do you want to generate fake images or fake text. Here you should completely define the problem and collect data for it.

Step 2: Define architecture of GAN

- Define how your GAN should look like. Should both your generator and discriminator be multi layer perceptron's, or convolutional neural networks? This step will depend on what problem you are trying to solve.

Step 3: Train Discriminator on real data for n epochs

- Get the data you want to generate fake on and train the discriminator to correctly predict them as real. Here value n can be any natural number between 1 and infinity.

Step 4: Generate fake inputs for generator and train discriminator on fake data

- Get generated data and let the discriminator correctly predict them as fake.

Step 5: Train Discriminator on Fake Data

- The samples which are generated by Generator will pass to Discriminator and It will predict the data passed to it is Fake or real and provide feedback to Generator again.

Step 6: Train generator with the output of discriminator

- Now when the discriminator is trained, you can get its predictions and use it as an objective for training the generator. Train the generator to fool the discriminator.

Loss function of GAN

Generator tries to minimize the loss function while discriminator tries to maximize the loss function

$$\min_G \max_D V(D, G)$$

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

where,
G = Generator
D = Discriminator
Pdata(x) = distribution of real data
P(z) = distribution of generator
x = sample from Pdata(x)
z = sample from P(z)
D(x) = Discriminator network
G(z) = Generator network

Applications of GAN

- Predicting the next frame in a video
- Increasing Resolution of an image
- Text to Image Generation
- Audio synthesis
- Face aging
- Image modification
- Speech generation