UCS2703 DEEP LEARNING

Unit-5

Autoencoders and Generative Models

Autoencoders - Undercomplete autoencoders - regularized autoencoders - stochastic encoders and decoders - Learning with autoencoders - Deep Generative Models : Variational autoencoders - generative adversarial networks

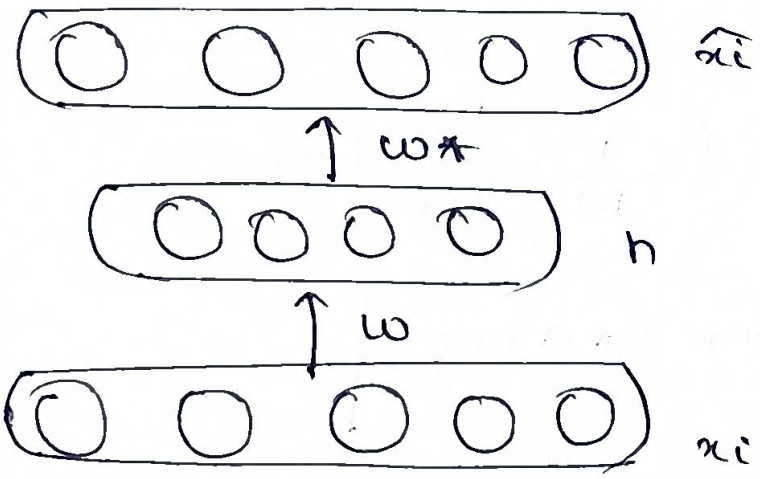
* Autoencoders

→ An autoencoder is a special type of feed forward neural network that :

(i) encodes it input x_i into a hidden representation h

(ii) decodes the hidden representation back into the input space to produce $\hat{x_i}$

eg. Image compression while retaining essential features



$h = g(wx_i + b)$

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

Loss Functions

→ The autoencoder is trained to minimize the reconstruction loss:

$$L(x_i, \hat{x_i}) = \|x_i - \hat{x_i}\|^2$$

→ This ensures that the reconstructed output is close as possible to the original input. The choice of loss function depends on the type of data being processed.

(i) Binary Data - inputs are binary (0 or 1)

(ii) Real-valued data - input can take any continuous value

For Binary Data → The best loss function is Binary Cross Entropy

$$L(x_i, \hat{x}_i) = \frac{-1}{N} \sum_{i=1}^{N} \left[x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i) \right]$$

where

x_i = original binary input

$\hat{x}_i$ = reconstructed output

N = number of samples

→ BCE penalizes incorrect predictions, while ensuring outputs remain between 0 and 1.

For real-valued data - MSE

$$L(x_i, \hat{x}_i) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2$$

Activation Functions

For Binary data → When using binary data, the decoder's activation function should also reflect this:

(i) Logistic Function : $\hat{x}_i$ = logistic (w * h + c)

(ii) Sigmoid Function

* Types of Autoencoders

A. Undercomplete Autoencoder - dimensionality of hidden layer is less than input

B. Over complete Autoencoder - dimensionality of hidden layer is greater than or equal to input

C. Regularized Autoencoders :

(i) Denoising Autoencoders - learn to reconstruct clean inputs from noisy versions - eg. remove noisy data from signals in audio

(ii) Sparse Autoencoders - encourage sparsity in the hidden layer - eg. feature selection in text data

(iii) Contractive Autoencoders - enforce robustness by penalizing the sensitivity of the hidden representations to input variations - eg. robust feature extraction in image data

* Regularized Autoencoders and their Loss Functions

→ Regulared autoencoders are designed to prevent overfitting and encourage the model to learn more meaningful representations

→ This is achieved by adding a regularization term to the loss function.

→ Common regularization techniques used are:

(i) L1 Regularization

(ii) L2 Regularization

(iii) Sparsity Constraints

→ In general, the loss function for a regularized autoencoder can be expressed as:

$$L(x, \hat{x}) = L_{reconstruction}(x, \hat{x}) + \lambda R(w)$$

where $L_{reconstruction}(x, \hat{x})$ = reconstruction loss ($^{eq}_{MSE}$)

$R(w)$ = regularization term applied to the weights w

λ = a hyperparameter that controls the strength of the regularization.

(i) **L1 Regularization** $\quad R(w) = \|w\|_1 = \sum_i |w_i|$

encourages sparsity in the wt. matrix

(ii) **L2 Regularization** $\quad R(w) = \|w\|_2^2 = \sum_i w_i^2$

penalizes large weights, promotes smaller wt. values

(iii) **Sparsity Constraints** encourages a certain percentage of neurons to be inactive during training

$$R(h) = \|h\|_1$$

* Regularized Autoencoders - Denoising Autoencoders

→ a variant of standard autoencoders designed to remove

noise from input data

→ They learn to reconstruct clean inputs from noisy versions, making them effective for tasks like image denoising.

→ Objective : minimize difference between the original clean image and the reconstructed image

→ Training : on pairs of noisy and clean images

Architecture : Input layer - noisy data x_n

Encoder - map noisy data to hidden representation h

Decoder - reconstruct output from the hidden representation to produce $\hat{x}_c$

$$h = g(W x_n + b)$$

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

Training Process - 1. Add noisy to input data

2. Use the noisy input to train the autoencoder to predict the clean output

3. Minimize the loss function:

$$L(x_c, \hat{x}_c) = \| x_c - \hat{x}_c \|^2$$

- This encourages the model to learn robust features that are invariant to noise.

Applications - Image Processing - remove noise from photos and medical images

Speech Enhancement - improve audio quality by filtering out background noise

Data preprocessing - cleaning datasets before feeding them into ML models

* Autoencoders vs. PCA

→ Both aim to reduce dimensionality while preserving info.

→ Autoencoders can learn non-linear mappings unlike PCA

* Applications of Autoencoders

→ Image denoising

→ Anomaly detection

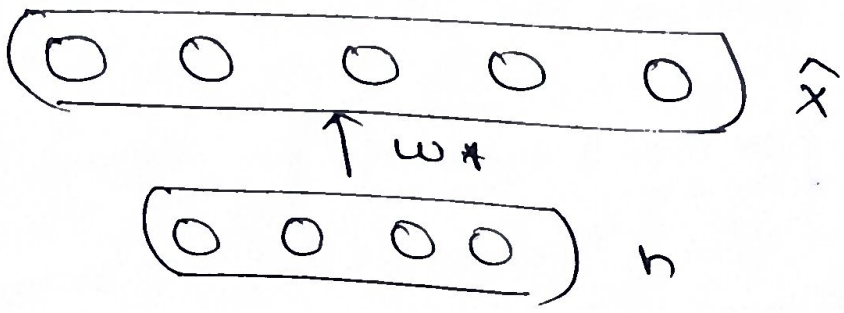
→ Dimensionality reduction

* Variational Autoencoders

* Generation with autoencoders

→ Once the autoencoder is trained, remove the encoder, feed a hidden representation h to the decoder and decode x from it.

→ h is a high dimensional vector and only a few vectors in this space would correspond to meaningful latent representations of the input.



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

→ Out of all the possible values of h, feed those values of x which are highly likely.

i.e Sample from $P(h|X)$ - pick only those h's which have a high probability

→ However, autoencoders do not have such a probabilistic interpretation.

→ They learn a hidden representation h, but not a distribution

$P(h|X)$

→ Similarly, the decoder is also deterministic and does not learn a distribution over X.

Solution : use a VAE

* Restricted Boltzmann Machines (RBMs)

→ a type of neural neural networks, used for unsupervised learning, particularly for generative models.

→ RBMs, like autoencoders, can take an input and reconstruct it by learning a good abstraction of that input

→ Beyond just abstraction, RBMs can also generate new data.

→ They belong to a class of models called energy-based models.

→ In these models, the goal is to minimize a certain energy

function, which is related to the likelihood of the data.

→ Lower energy corresponds to higher probability of the data.

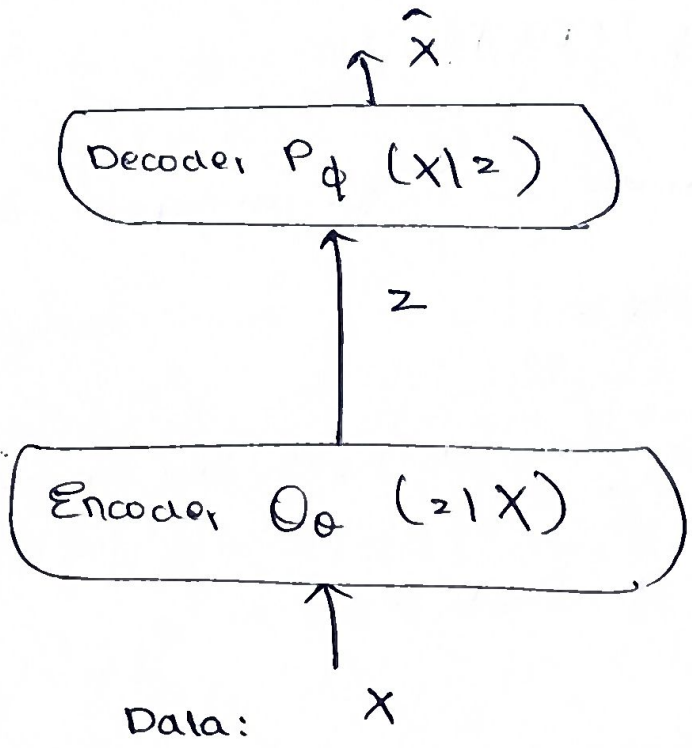
→ RBMs are probabilistic, meaning they learn to model the probability distribution of the input data. They try to find the joint probability distribution that maximizes the likelihood (→ log-likelihood) of the observed data.

* Variational Autoencoders

→ VAEs improve upon regular autoencoders by learning a probabilistic distribution over the hidden (latent) variables.

→ The encoder doesn't just map to a fixed hidden representation, but rather to a distribution, typically a normal (Gaussian) distribution

Goals



θ = parameters of the encoder neural network

ϕ = parameters of the decoder neural network

→ Learn a distribution over the latent variables ($Q(z|x)$)

→ Learn a distribution over the visible variables ($P(x|z)$)

Encoder

→ The encoder's job is to learn the parameters of a probability distribution (often a normal distribution) for the latent variables z given an input X

→ $Q(z|X)$ represents the distribution of latent variables z conditioned on the input X

→ In a VAE, we assume that the latent variables z come from a standard normal distribution. The encoder's job is to predict the parameters (mean and covariance) for this normal distribution

→ Once the encoder provides the distribution parameters, we can sample random values of z from the predicted distribution

→ From a neural network perspective, the encoder takes each input x and predicts a mean vector and diagonal covariance matrix.

→ To train the encoder, we need a loss fn, that includes a reconstruction loss and a regularization term to ensure that the distribution $Q(z|X)$ stays close to the prior $P(z)$.

Decoder

→ The decoder takes a latent variable z and predicts a probability distribution over the input data X, represented as $P(X|z)$

→ The job of the decoder is to predict the mean of this distribution $f_\phi(z)$

Decoder Objective function

→ The goal is to maximize the probability $P(x_i)$ for a given training sample x_i

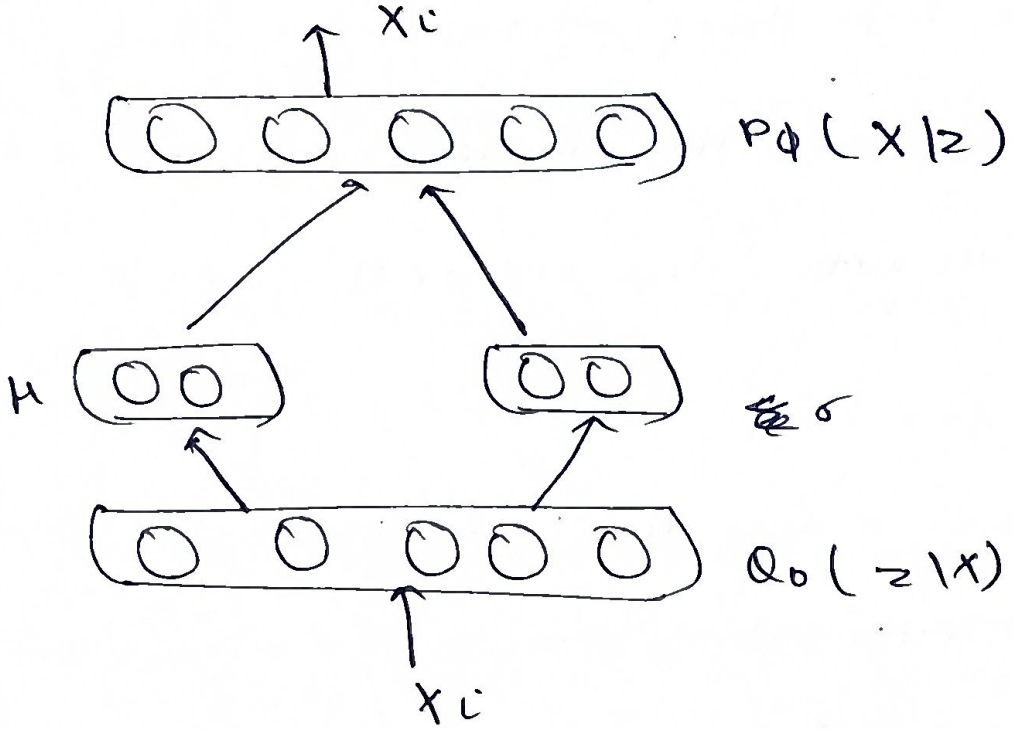
→ To do this calculate the probability $P(x_i)$ for each input x_i, which involves integrating over all the latent variables z

$$P(x_i) = \int P(z)\, P(x_i|z)\, dz$$

$$= E_{z \sim Q_\theta(z|x_i)} [\log P(x_i|z)]$$

→ The total loss is the sum of losses for all data points, which the model tries to minimize.

$$L(\theta) = \sum_{i=1}^{n} l_i(\theta)$$



In addition we want $Q(z|X)$ to be as close to $P(X)$ as possible.

∴ The loss function is modified as:

$$l_i(\theta, \phi) = -E_{z \sim Q_\phi(z|x_i)} [\log P(x_i|z)] + KL(Q_\phi(z|x_i) \,||\, P(z))$$

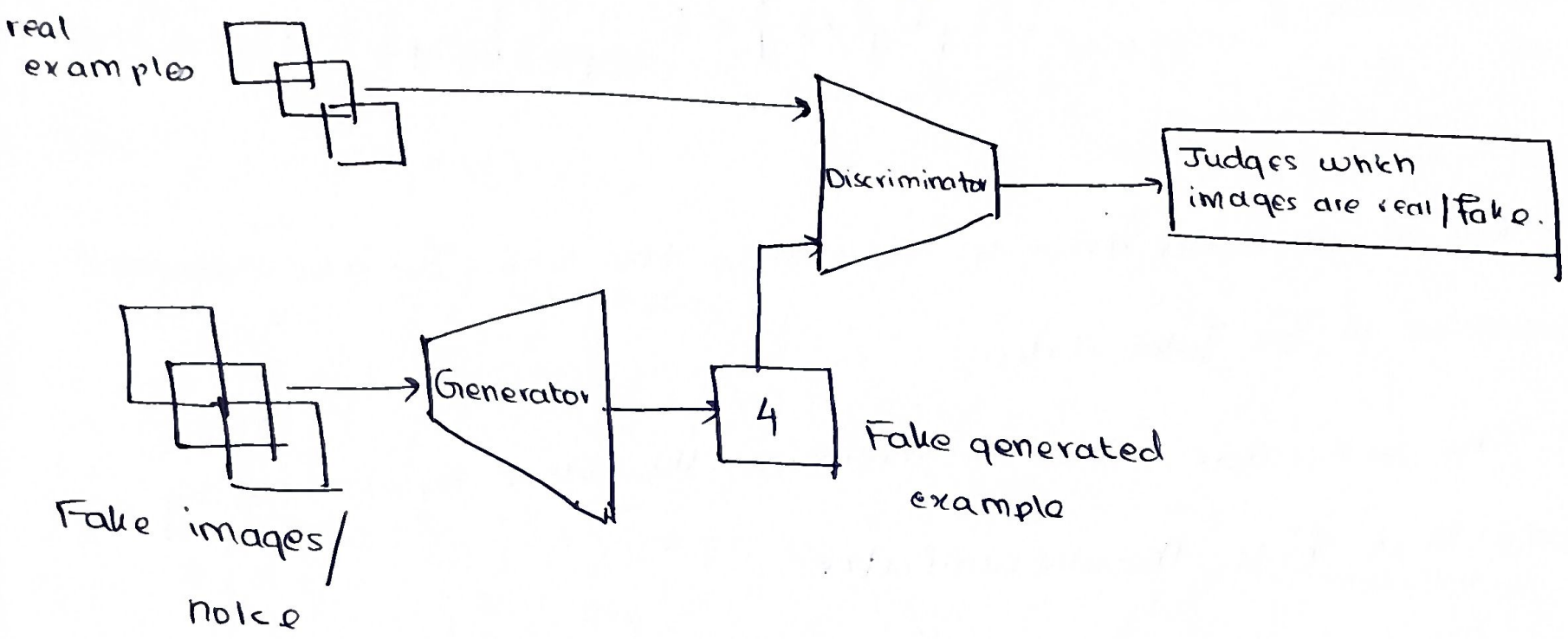
$z = \mu + \sigma \epsilon$
↳ to allow gradients to pass

KL captures distance between 2 distributions — regularizer

* Generative Adversarial Networks

→ a class of machine learning models where 2 neural networks, like the generator and discriminator, are trained simultaneously in a game-like setting.

→ The generator creates fake data, and the discriminator tries to distinguish between real and fake data.



→ GANs consist of two neural networks - generator (G) and discriminator (D)

→ The generator takes noise $z \sim N(0,1)$ and produces data $G_\theta(z)$.

→ The discriminator evaluates the data, assigning a score $D_\phi(G_\theta(z))$ between 0 and 1.

→ The goal is to improve both networks in an adversarial setting.

Generator's objective

minimize :

$$E_{z \sim p(z)} [\log(1 - D_\phi(G_\theta(z)))]$$

for a discrete distribution: $\min_\theta \frac{1}{N} \sum_{i=1}^{N} \log(1 - D_\phi(G_\theta(z)))$

Discriminator's Objective

→ maximize the log-probability of correctly classifying real data.

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

→ Together, the generator and discriminator form a minimax game:

$$\min_{\theta} \max_{\phi} E_{x \sim p(data)} [\log D_{\phi}(x)] + E_{z \sim p(z)} [\log(1 - D_{\phi}(G_{\theta}(z)))]$$

→ The discriminator tries to maximize the score for real data and minimize it for fake data.

→ The generator tries to minimize this loss by generating data that fools the discriminator.

* Deep Convolutional GANs (DCGANs)

→ a GAN architecture that uses convolutional layers in both the generator and discriminator.

→ any CNN-based classifier - VGG, ResNet

→ Guidelines for stable DCGANs:

1. Replace pooling layers with strided convolutions in the discriminator
2. Use fractional-stride convolutions in the generator
3. Use batch normalization in both networks
4. Use ReLU activation in the generator - except for the output

Layer, which uses tanh.

5. Use LeakyReLU in the discriminator.

* Applications of GANs

1. Image generation and enhancement - eg. deepfake technology, super-resolution
2. Text-to-image generation (eg. DALL-E)
3. Data augmentation in training datasets
4. Music & audio synthesis
5. Video prediction and generation

* Variants of GANs

1. Conditional GANs - Incorporates labels to generate class-conditional samples.
2. Deep Convolutional GAN - uses convolutional layers, improving image generation
3. CycleGAN - used for unpaired image-to-image translation
4. StyleGAN - produces high-resolution, realistic images with control over styles
5. Wasserstein GAN - Improves stability by addressing issues like mode collapse.

* Challenges with GANs

1. Mode Collapse - the generator may produce limited varieties of output

2. Training Instability - The adversarial nature makes GAN training difficult.

3 Evaluation Metrics - challenging to quantitatively evaluate the quality of generated data.