Topics in Deep Learning

Dr. Shylaja S S

Director of Cloud Computing & Big Data (CCBD), Centre for Data Sciences & Applied Machine Learning (CDSAML)

Department of Computer Science and Engineering

shylaja.sharath@pes.edu

Ack: Devang Saraogi,
Teaching Assistant

PES UNIVERSITY

Unit 3: Generative Models and Meta Learning

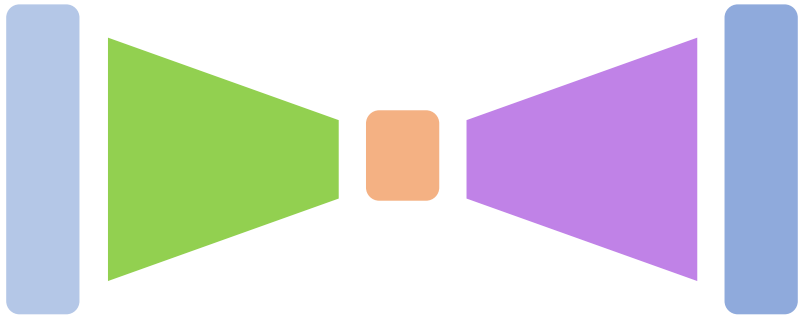
Autoencoders

Ack: Devang Saraogi,
Teaching Assistant

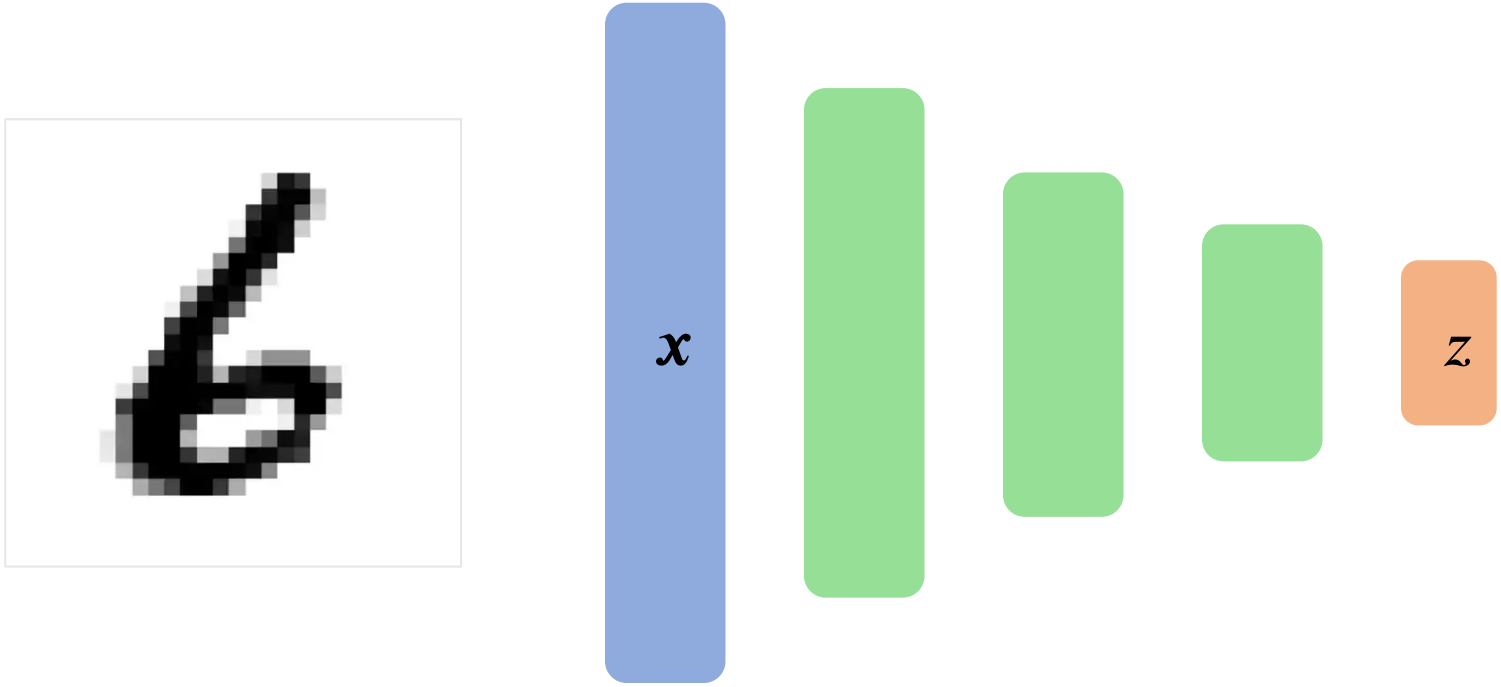
Autoencoders

Autoencoder is a special type of neural network model whose goal is to learn a compressed representation of the input data that captures its most salient features. This compressed representation (latent space) can be used for a variety of tasks such as data compression, denoising, image generation and anomaly detection.

- One of the key advantages of an autoencoder is that **they can learn useful features from raw data without the need for explicit labels or supervision**. This makes them particularly useful in cases where labelled data is scarce or expensive to obtain.
- They are also used to pretrain deep neural networks, which can improve the performance of supervised learning tasks.



- We pass raw input data (x) into a series of neural network layers.
- The output is an encoded representation of the underlying features of (z) and is referred to as the low dimensional latent space.



"**Encoder**" learns the mapping from the data, (x), to a low-dimensional latent space, (z)

Question?

Why do we have the latent vector (z) in a low dimensional space?

- We are able to compress the input data into a small feature representation vector that does not require too much memory.
- It is a very efficient, compact encoding of the rich high dimensional data that we originally pass into the encoder.

Remember,

The primary objective of an autoencoder is to learn a **compressed representation** of the input data that captures its **most salient features**. This compressed representation, can be used for a variety of tasks such as **data compression**, **denoising**, **image generation**, and **anomaly detection**.

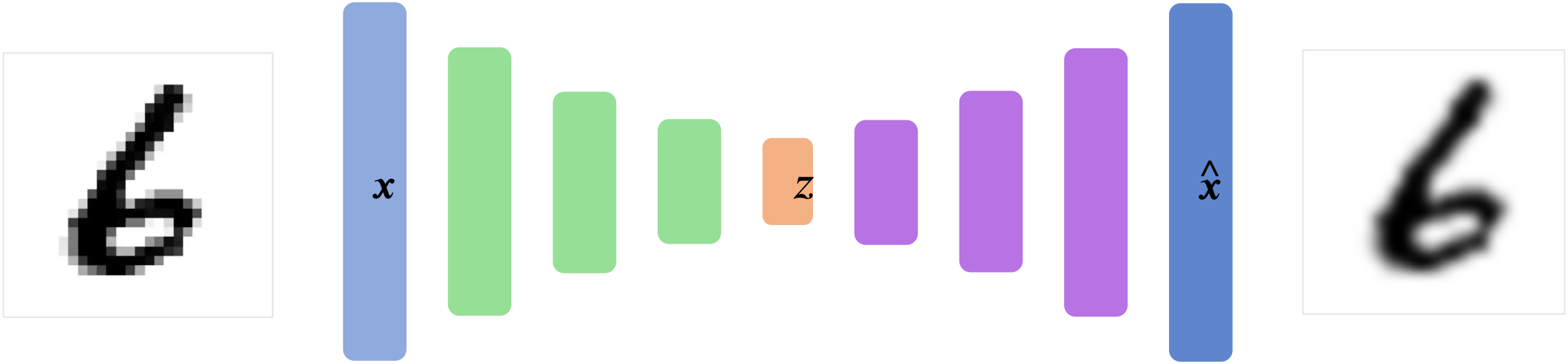
Training an Autoencoder

We cannot explicitly observe the latent variable (z) because there is not training data. So, **how do we train the network to learn this latent variable vector?**

- The autoencoder builds a way to **reverse** the encoding process by **decoding the latent variable vector back into the original data space.** In our case the autoencoder tries to reconstruct original image from the compressed efficient latent encoding.
- We use a series of convolutional or fully connected layers to map the lower dimensional space data back into the original dimensional space.
- The generated output is denoted by $(\hat{x})$. It is an **imperfect reconstruction** of our input data.
- **Our goal is to reconstruct the input data from the latent representation.** To train the network we compare the (reconstructed) output with the input.

How can we learn the latent space?

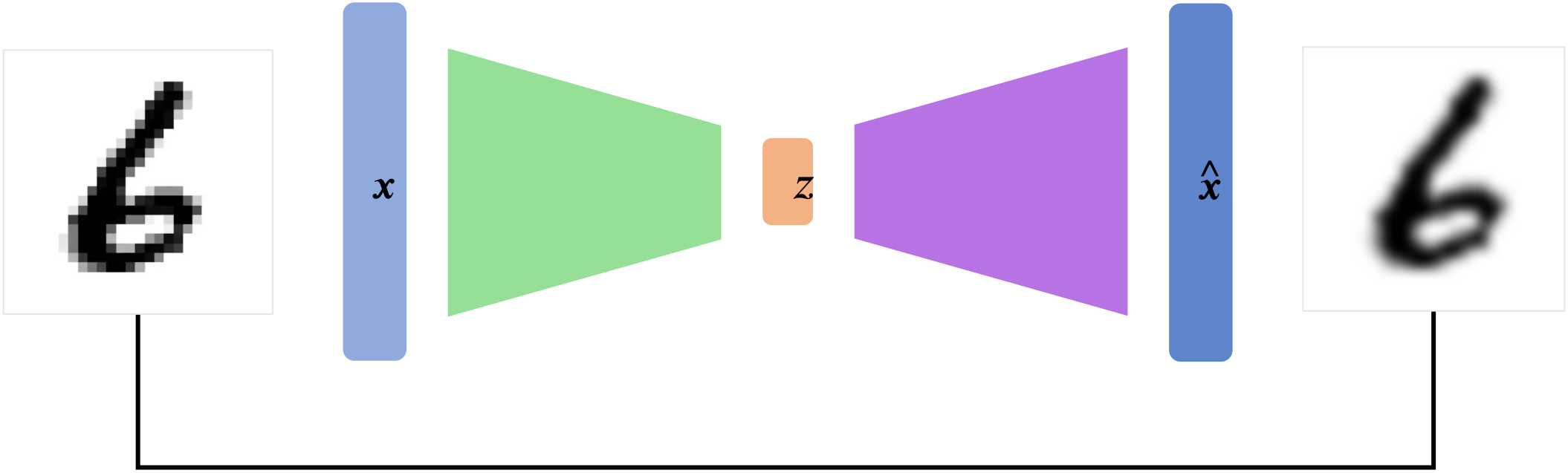
We train the model to use the features to reconstruct the original data.



"Decoder" learns mapping from the latent space, (z), to a reconstructed observation, $\left(\hat{x}\right)$

How can we learn the latent space?

We compare the (reconstructed) output with the input.



$$L = \left\| x - \hat{x} \right\|^2$$

Autoencoder

Autoencoders consists of two parts: an **encoder** and a **decoder**.

- The encoder takes an input vector($\boldsymbol{x}$)and maps it to a compressed representation or the latent space ($\boldsymbol{z}$) using an encoder function $\boldsymbol{f}(\boldsymbol{x})$.

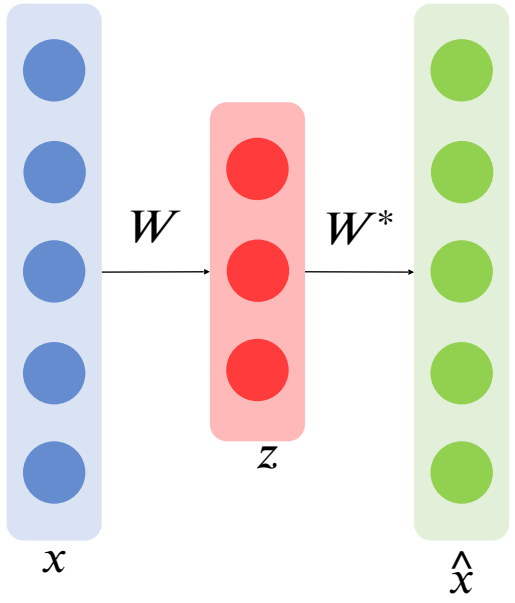
$$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) \quad \Rightarrow \quad \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

- The decoder takes the compressed representation ($\boldsymbol{z}$) and maps it back to the original input vector ($\boldsymbol{x}$), using a decoder function $\boldsymbol{g}(\boldsymbol{x})$.

$$\hat{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{z}) \quad \Rightarrow \quad \hat{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{W}^*\boldsymbol{z} + \boldsymbol{c})$$

- The goal of training an autoencoder is to minimize the difference between the input vector ($\boldsymbol{x}$) and the reconstructed vector $\left(\hat{\boldsymbol{x}}\right)$. This is done by minimizing the mean squared error between the two vectors:

$$\boldsymbol{L} = \left\|\boldsymbol{x} - \hat{\boldsymbol{x}}\right\|^2$$

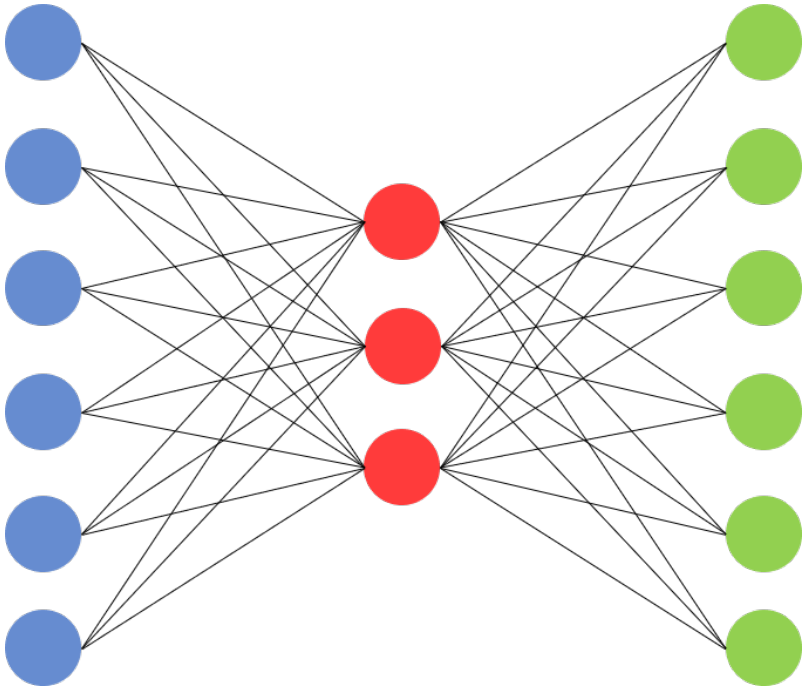


Information Bottleneck

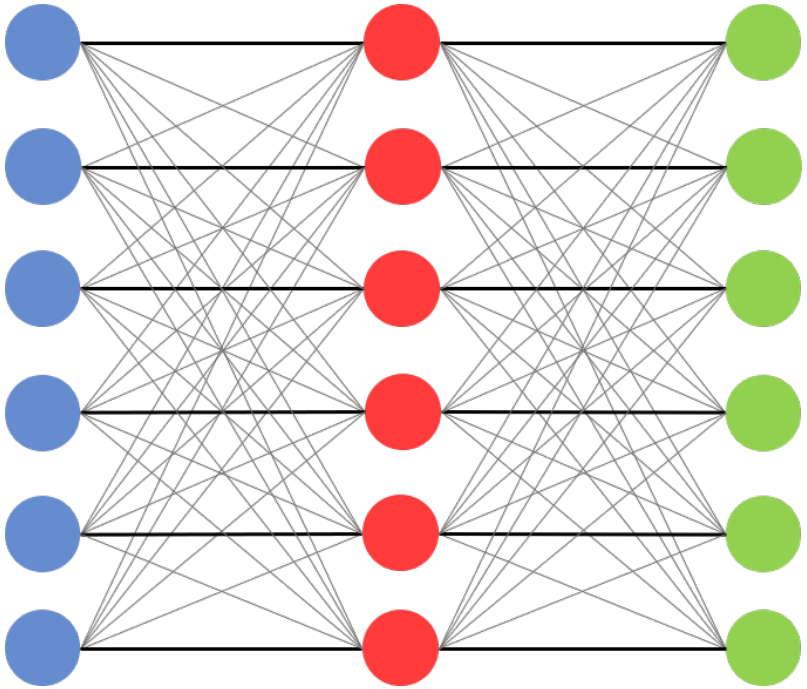
- The autoencoder is designed in such a way that the network **forces** a compressed knowledge representation of the original input.
- If the input features were independent, the compression and reconstruction would be very difficult. However, if the input data contained some sort of **underlying structure** (i.e. correlations between input features), the structure could be **learned** by the network and then used to compress inputs into the lower dimensional space.

A bottleneck controls the amount of information that can travel through the full network, forcing a learned compression of the input data.

**Without the information bottleneck, the network would easily learn to simply memorize the input values by passing these values along through the network. (see following slides)*



Bottleneck



No Bottleneck

Regularization

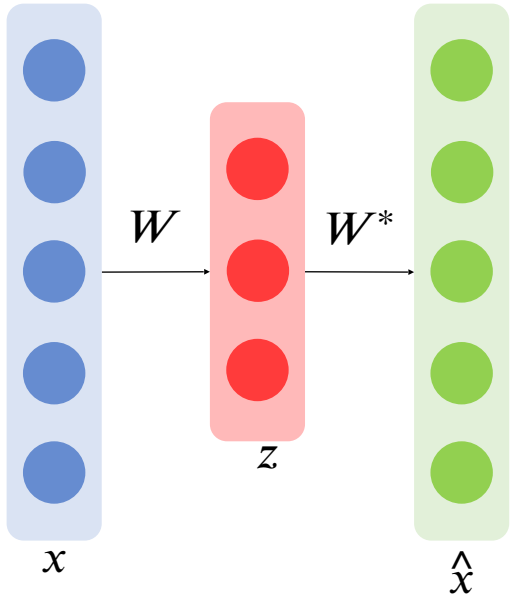
The **ideal autoencoder** model balances the following:

- **Sensitive** to the **inputs enough to accurately build** a reconstruction.
- **Insensitive** enough to the **inputs that the model doesn't simply memorize** or **overfit the training data**.

This trade-off forces the model to maintain only the variations in the data required to reconstruct the input without holding on to redundancies within the input. For most cases, this involves constructing a loss function

$$L\left(x, \hat{x}\right) + regularizer$$

- where one term encourages the model to be sensitive to the inputs **(i.e. reconstruction loss)**
- and a second term discourages memorization/overfitting **(i.e. an added regularizer)**

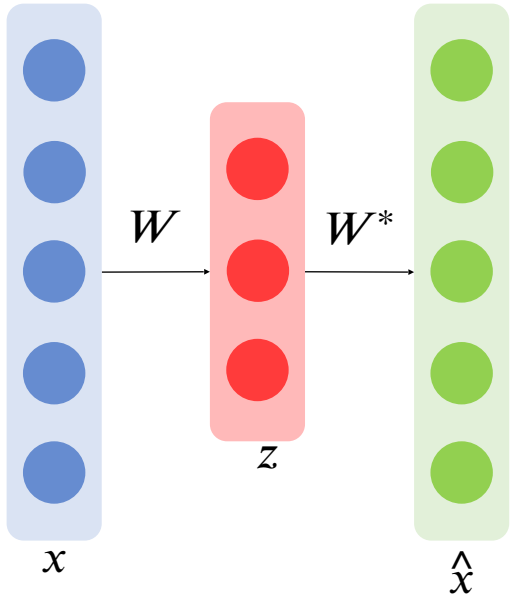


Regularization

As stated earlier, the model can simply learn to copy (x) to (z) and then (z) to $(\hat{x})$. Introducing a regularizer will help to avoid poor generalizations.

The simplest solution is to add an L2-regularization term to the objective function.

$$\min_{\theta, w, w^*, b, c} \frac{1}{m} \Sigma_{i=1}^{m} \Sigma_{j=1}^{n} \left(\hat{x}_{ij} - x_{ij} \right)^2 + \lambda \|\theta\|^2$$



Another trick is to tie the weights of the encoder and decoder i.e., $W^* = WT$

- when weights are tied, the decoder's weights are constrained to be the transpose of the encoder's weights
- the constraint reduces the model's capacity to learn new weights thus avoiding overfitting

Hyperparameters

Some of the most common hyperparameters that can be tuned when optimizing autoencoders are:

- number of layers in the encoder and decoder network
- number of nodes for each of these layers
- loss function (e.g., binary cross-entropy or mean squared error)
- size of the latent space (the smaller, the higher the compression, acting, therefore as a regularization mechanism)

Dimensionality of Latent Space

It has been observed that the dimensionality of the latent space has a huge impact on the quality of the generated reconstructions and how compressed the information bottleneck is.

2D Latent Space

5D Latent Space

Ground Truth

Types of Autoencoders

Over the years, different types of Autoencoders have been developed:

- Undercomplete and Overcomplete Autoencoders
- Denoising Autoencoder
- Sparse Autoencoder
- Contractive Autoencoder
- Variational Autoencoder
- Stacked Autoencoder
- Convolutional Autoencoder
- Masked Autoencoder
- Video Autoencoder

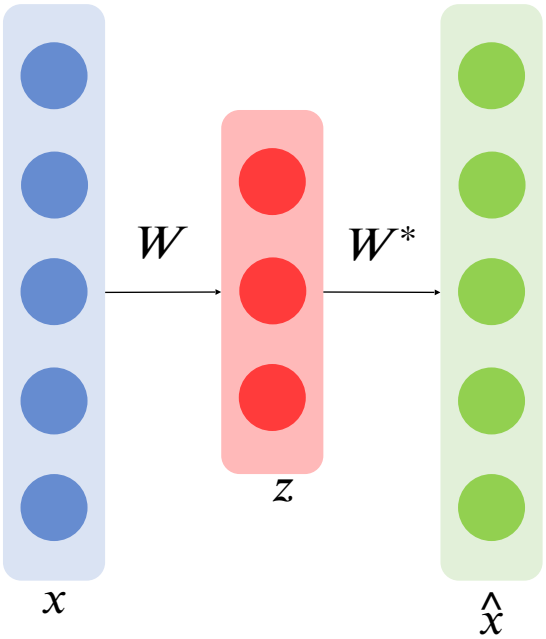
Undercomplete Autoencoder

Undercomplete Autoencoder

- An autoencoder is said to be **undercomplete** if the dimensionality of the latent space (i.e., the number of neurons in the encoder output layer) is lesser than the dimensionality of the input space.

$$\dim(z) < \dim(x)$$

- In other words, the encoder is forced to learn a compressed representation of the input data that captures only the most essential features.
- This can help **prevent overfitting** and encourage the autoencoder to learn a more general representation of the input data.
- However, an undercomplete autoencoder may not be able to capture all the essential features of the input data.



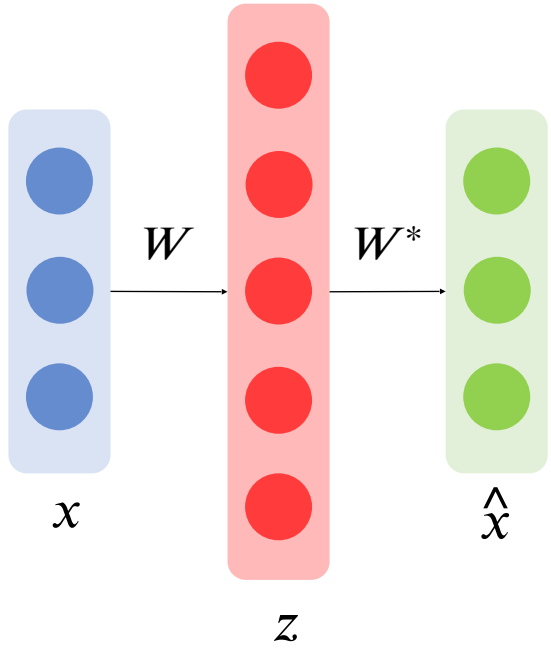
Overcomplete Autoencoder

Overcomplete Autoencoder

- An autoencoder is said to be **overcomplete** if the dimensionality of the latent space (i.e., the number of neurons in the encoder output layer) is greater than the dimensionality of the input space.

$$\dim(z) \geq \dim(x)$$

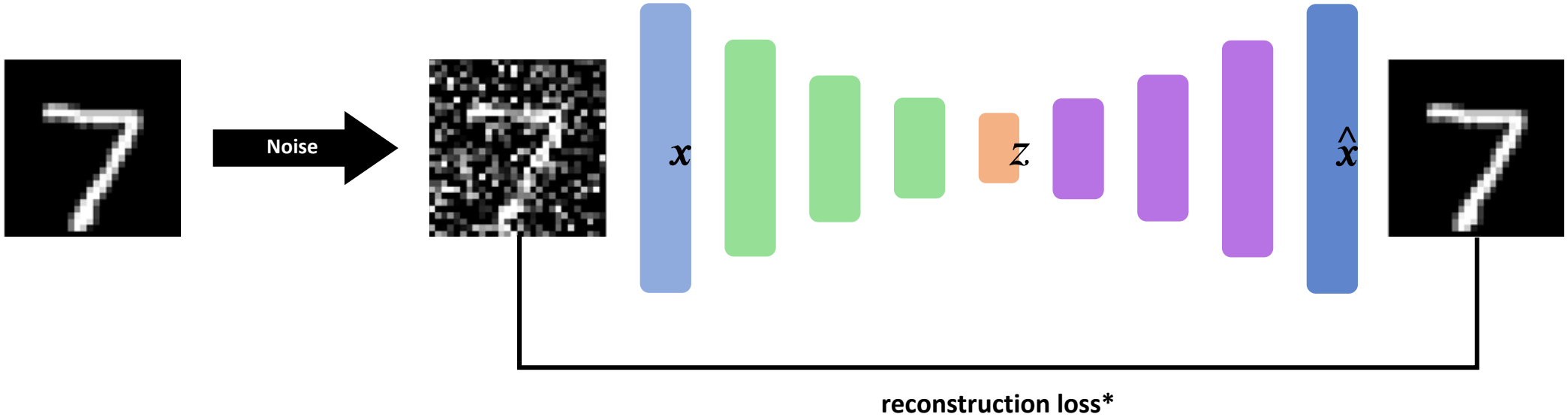
- In other words, there are more hidden units in the encoder than necessary to capture all the essential features of the input data.
- This means that an overcomplete autoencoder can learn **multiple compressed representations** of the input data, each capturing a different aspect of the data.
- However, this can also lead to overfitting, where the autoencoder learns to simply copy the input data to the latent space without capturing meaningful features.



Denoising Autoencoder

Denoising Autoencoder

Denoising Autoencoders are networks that are designed to remove noise from the input data. In contrast to undercomplete autoencoders, denoising autoencoders do not use the input as the ground truth. Instead, some form of noise (e.g. Gaussian Noise) is introduced into the input and the network aims to remove that.



**loss function generally used in these types of networks is L1 or L2 loss*

Denoising Autoencoder

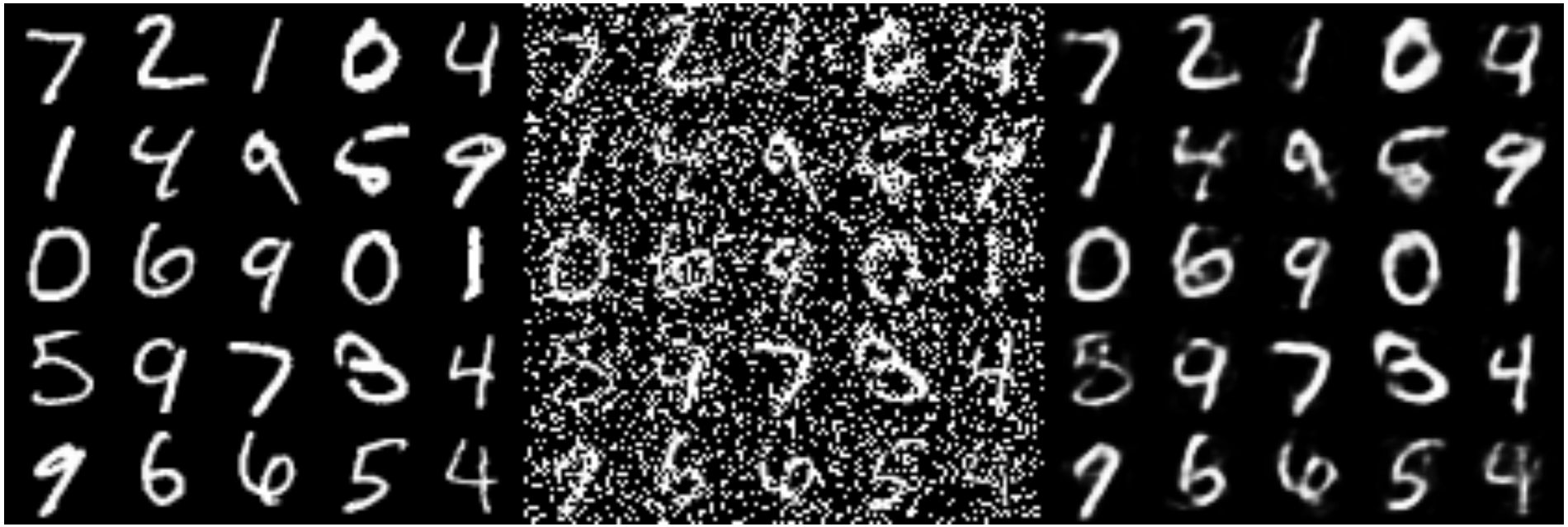
Denoising Autoencoder

The aim is to reconstruct the original input (free of noise) from a corrupted version of the input.

- The original input is corrupted using a probabilistic process $P\left(\tilde{x}_{ij} \middle| x_{ij}\right)$ before feeding it to the network.
- A simple P could be

$$P\left(\tilde{x}_{ij} = 0 \middle| x_{ij}\right) = q \qquad P\left(\tilde{x}_{ij} = x_{ij} \middle| x_{ij}\right) = 1 - q$$

- It's assumed that the higher-level representations are **relatively stable** and can be easily extracted.
- While removing noise directly from the inputs seems difficult, the autoencoder performs this by mapping the input data into a lower-dimensional manifold (like in undercomplete autoencoders), where filtering of noise becomes much easier.
- Essentially, denoising autoencoders work with the help of **non-linear dimensionality reduction**.



Original Data, Corrupted Data (Input) and Reconstructed Data (Output)

Denoising Autoencoder

Pros

- In the process of training, the network can no longer memorize and copy the inputs to the latent space because the input and the ground truth are not the same anymore.
- Denoising autoencoders is good for learning the latent representation in corrupted data while creating a robust representation of the same, allowing the model to recover true features.

Cons

- It is important to perform preliminary stochastic mapping i.e. add noise randomly to the data and then pass it as input to the network.

Sparse Autoencoder

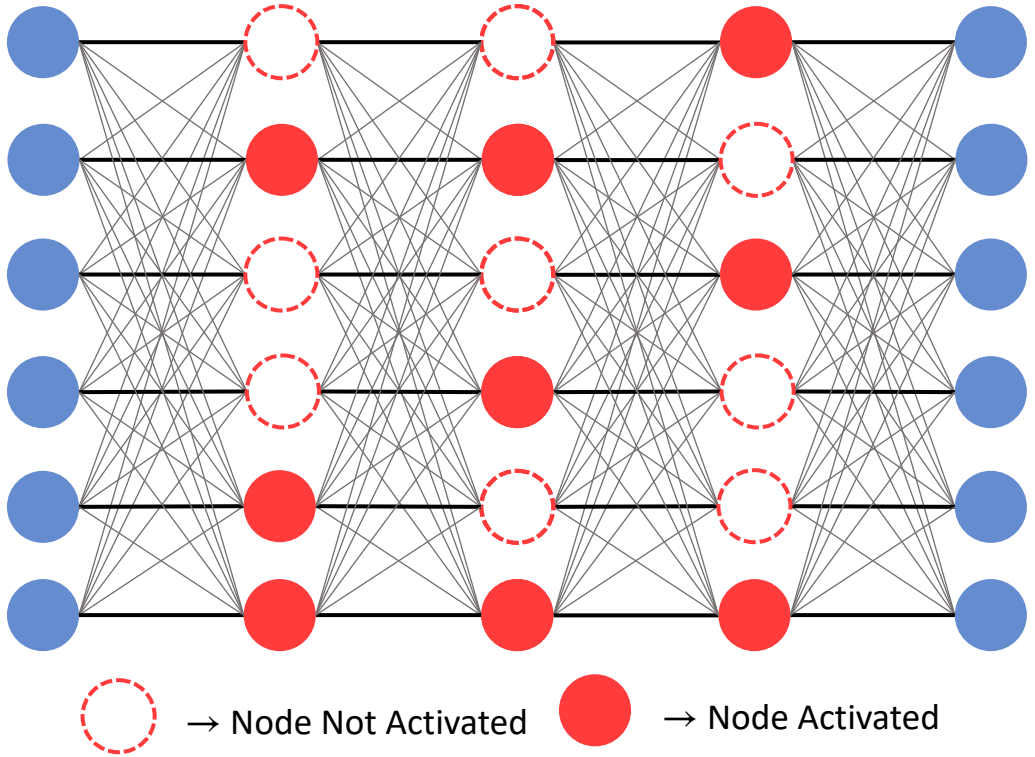
Sparse Autoencoder

Sparse autoencoders and undercomplete autoencoders share similarities in that they both use the inputs as the ground truth. However, they differ significantly in how they regulate the encoding of information.

- Usually fine tuning is done by regulating the size of the bottleneck, the sparse autoencoder is regulated by changing the number of nodes at each hidden layer.
- Since it is not possible to design a neural network that has a flexible number of nodes at its hidden layers, sparse autoencoders work by penalizing the **activation of some neurons*** in hidden layers.
- The loss function has a term that calculates the number of neurons that have been activated and provides a penalty that is directly proportional to that.
- This penalty, called the sparsity function, prevents the neural network from activating more neurons and serves as a regularizer.

** a neuron with the sigmoid activation function is said to be activated when the output is close to 1 and is said to be not activated when the output is close to 0, the sparse autoencoder tries to ensure that a neuron is inactive most of the times.*

- sparsity penalties allow the network to have nodes dedicated to finding specific features in the input data.
- training process treats the regularization problem as a problem separate from the latent space problem.
- latent space dimensionality can be set at the bottleneck without worrying about regularization.



Sparse Autoencoder

This sparsity constraint can be induced by using L1 Regularization and KL divergence, effectively preventing the model from overfitting.

L1 Regularization

The term penalizes the absolute value of the vector of activations $\boldsymbol{a}$ in layer $\boldsymbol{h}$ for observation $\boldsymbol{i}$, scaled by a tuning parameter λ.

$$L = \left| x - \hat{x} \right| + \lambda \sum_{i} \left| a_i^{(h)} \right|$$

Sparse Autoencoder

This sparsity constraint can be induced by using L1 Regularization and KL divergence, effectively preventing the model from overfitting.

KL Divergence

The activations are considered over a collection of samples at once rather than being considered as a summation like in the L1 Regularization method.

- the average activation of each neuron is constrained
- considering the ideal distribution as a Bernoulli distribution, KL Divergence is included to reduce the difference between the current distribution of the activations and the ideal (Bernoulli) distribution

$$L = \left| x - \hat{x} \right| + \sum_j KL\left(\rho \,||\, \hat{\rho}_j \right)$$

where $\hat{\rho}_j = \frac{1}{m} \sum_i \left[a_i^{(h)}(x) \right]$ and j denote the specific neuron for layer h and a collection of m samples is being considered here, each represented/denoted by x.

Sparse Autoencoder

Pros

- In sparse autoencoders, **overfitting** is prevented by applying a sparsity penalty. The sparsity penalty is applied on both the hidden layers and reconstruction error. This allows the model to be more versatile by increasing the capacity and learning complex topologies.

Cons

- It is important that the nodes are data-dependent since the input vectors are responsible for the activation of different nodes that yields results during training. **Hence, any slight statistical change in the test data can yield different results.**

Contractive Autoencoder

Contractive Autoencoder

One would expect that for **very similar inputs, the learned encoding would also be very similar.** Contractive autoencoders are networks that are explicitly designed to do so. They work on the basis that similar inputs should have similar encodings and a similar latent space representation.

- The latent space **does not vary by a huge amount for minor variations** of the **same** input.
- This is enforced by making sure that the **derivates of the hidden layer activations are very small** with respect to the input.
- **This forces the model to learn how to contract a neighborhood of inputs into a smaller neighborhood of outputs.**
- Instances where a **small change in the input** leads to a **large change in the output** are penalized.

An important thing to note in the loss function (formed from the norm of the derivatives and the reconstruction loss) is that the two terms contradict each other.

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.

Putting these two contradictory conditions into one loss function enables training where the hidden layers now capture only the most essential information. This information is necessary to separate inputs and ignore information that is non-discriminatory in nature, and therefore, not important.

The total loss function can be mathematically expressed as:

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

Where (h) is the hidden layer for which a gradient is calculated and represented with respect to the input (x) as $\nabla_x a_i^{(h)}(x)$. The gradient is summed over all training samples, and a frobenius norm of the same is taken.

**in simple terms, the frobenius norm can be defined as the square root of the sum of the squares of all the matrix entries*

Contractive Autoencoder

Pros

- Contractive autoencoders are a good choice over sparse autoencoders to learn good representation since they **are robust to slight variations** and the nodes are not data-dependent.

Cons

- Contractive autoencoders suffer from a major drawback in its reconstruction error during the encoding and decoding process of the input vectors. **This leads to neglecting finer details worth considering for reconstruction.**

Applications of Autoencoders

Applications of autoencoders include:

- Anomaly detection (identifying data points that deviate from the norm)
- Data denoising (removing noise from data)
- Time Series (analyzing and predicting sequential data)
- Image Generation (generating images from learned distribution)
- Self Supervised Learning (learning representations from unlabeled data)
- Image inpainting (filling missing parts of an image)
- Information retrieval (efficient retrieval of relevant features from a large dataset)
- Disentanglement (separating underlying factors of variation)
- Image Classification (assigning an input image to a predefined category)

PES UNIVERSITY

References

- https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Handout/Lecture7.pdf
- https://www.jeremyjordan.me/autoencoders/
- https://youtu.be/3G5hWM6jqPk?si=q1MmWbFeFBgJ7Px0
- https://www.datacamp.com/tutorial/introduction-to-autoencoders
- https://www.v7labs.com/blog/autoencoders-guide
- https://towardsdatascience.com/a-high-level-guide-to-autoencoders-b103ccd45924
- https://merveenoyan.medium.com/complete-guide-on-deep-learning-architectures-part-2-autoencoders-293351bbe027
- https://tonyjesuthasan.medium.com/autoencoders-a-comprehensive-guide-d0723c2f988b
- https://neptune.ai/blog/autoencoders-case-study-guide
- https://viso.ai/deep-learning/autoencoder/

Thank You

Dr. Shylaja S S

Director of Cloud Computing & Big Data (CCBD), Centre for Data Sciences & Applied Machine Learning (CDSAML)

Department of Computer Science and Engineering

shylaja.sharath@pes.edu

Ack: Devang Saraogi,
Teaching Assistant