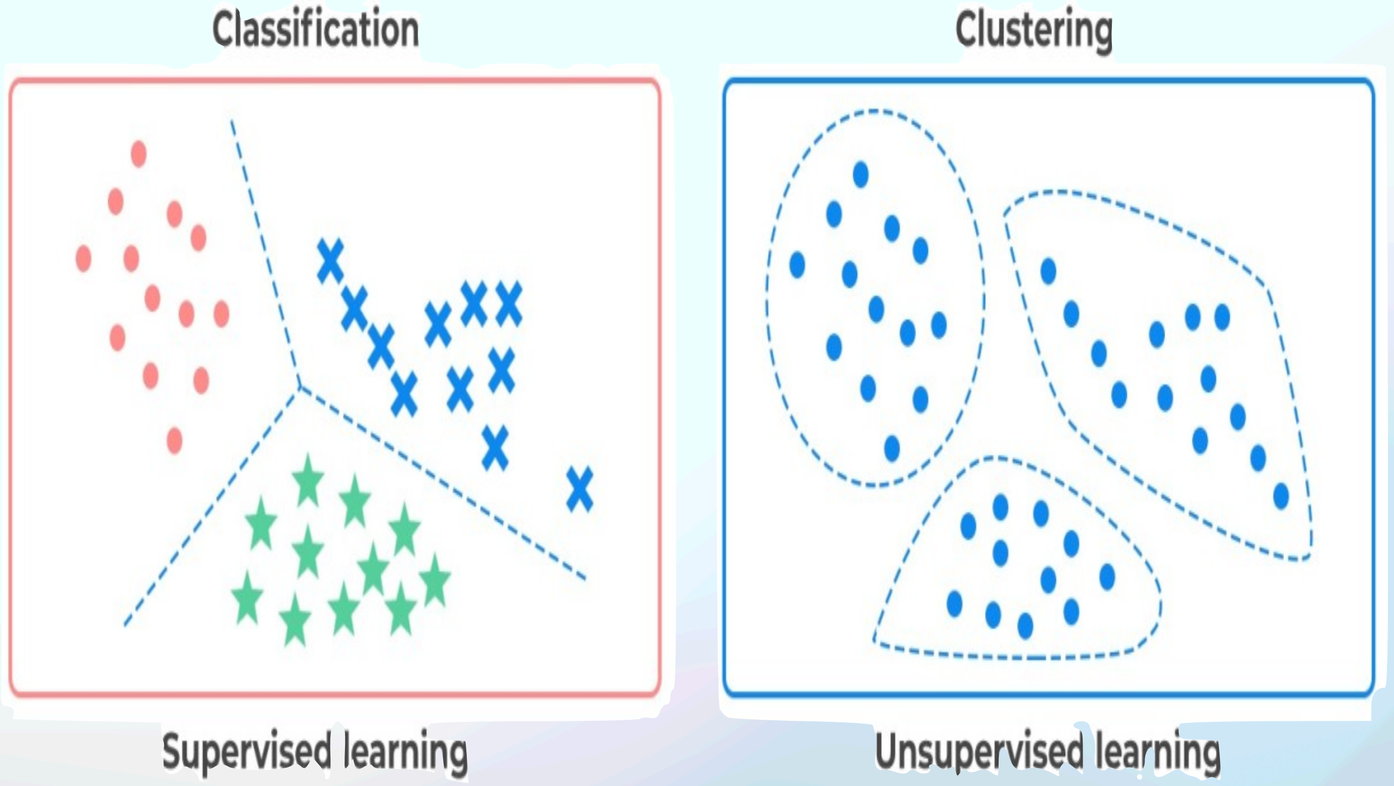
AutoEncoders

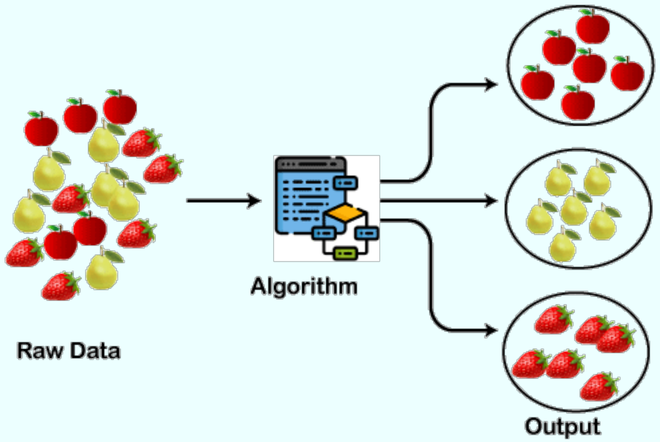
Dr. Thomas Abraham
SCOPE, VIT Chennai

Unsupervised Learning

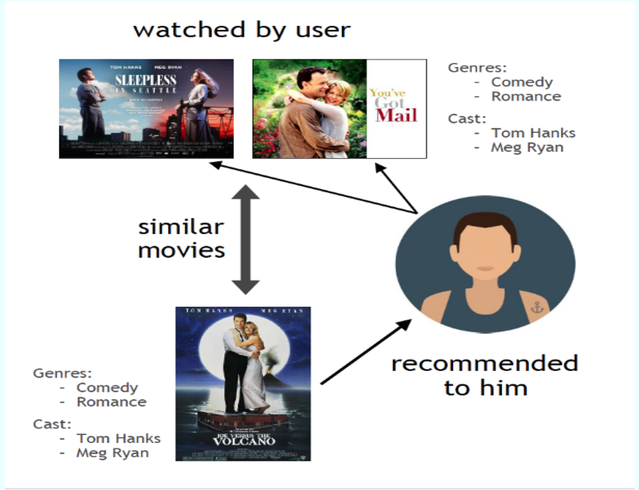
- Used when there is input data (X) and no corresponding output label (Y)
- Objective is to explore the data and find some pattern in it
- Very useful when there is lot of data but very less label
- Example tasks such as association rule, clustering, compression, dimensionality reduction, data generation



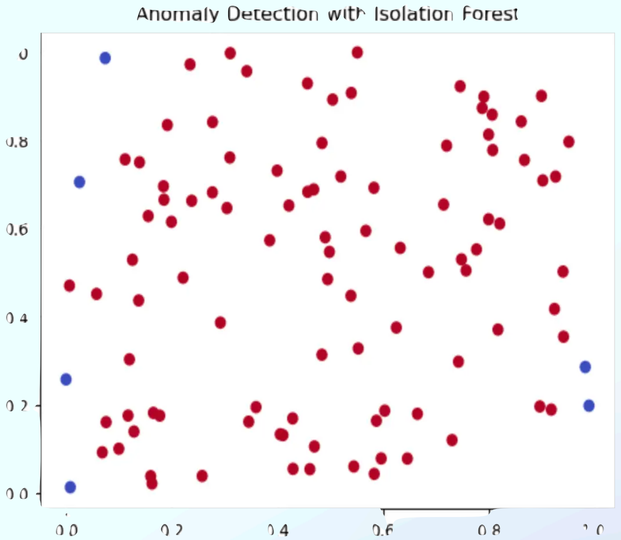
Applications of UL



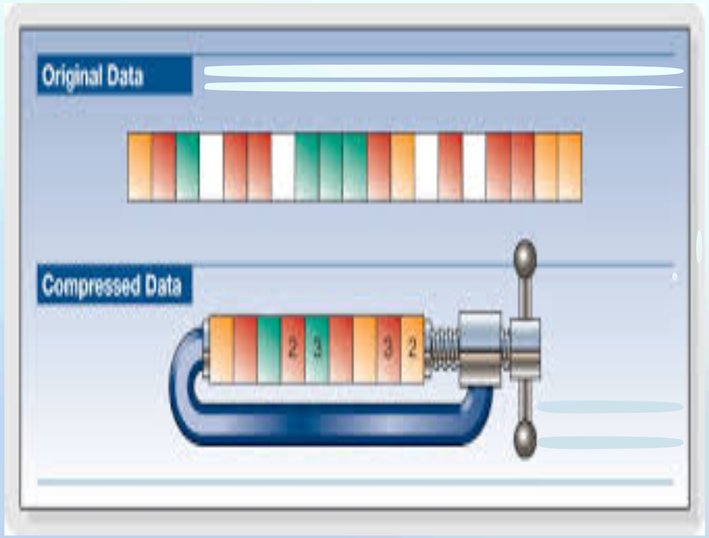
Clustering



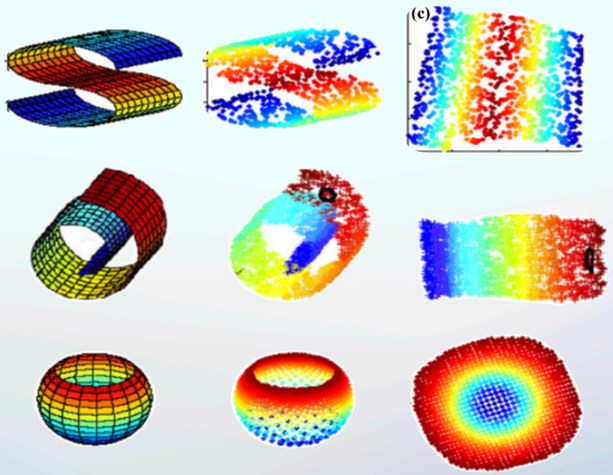
Recommendation System



Anamoly Detection



Data Compression

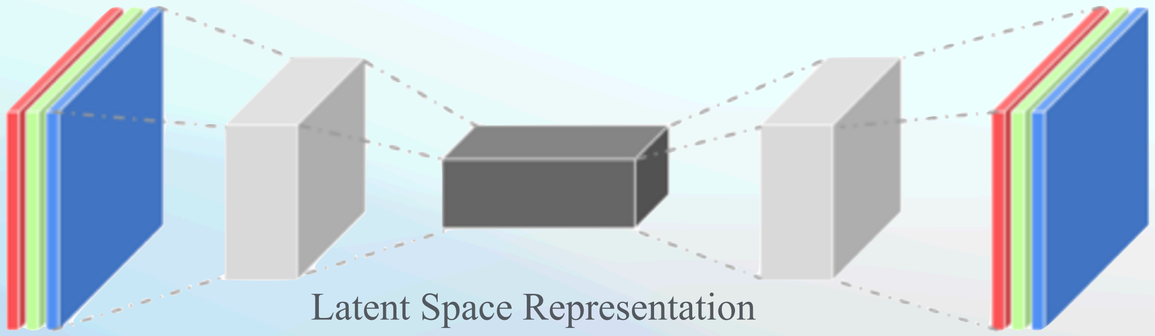


Dimensionality Reduction

Data Generation

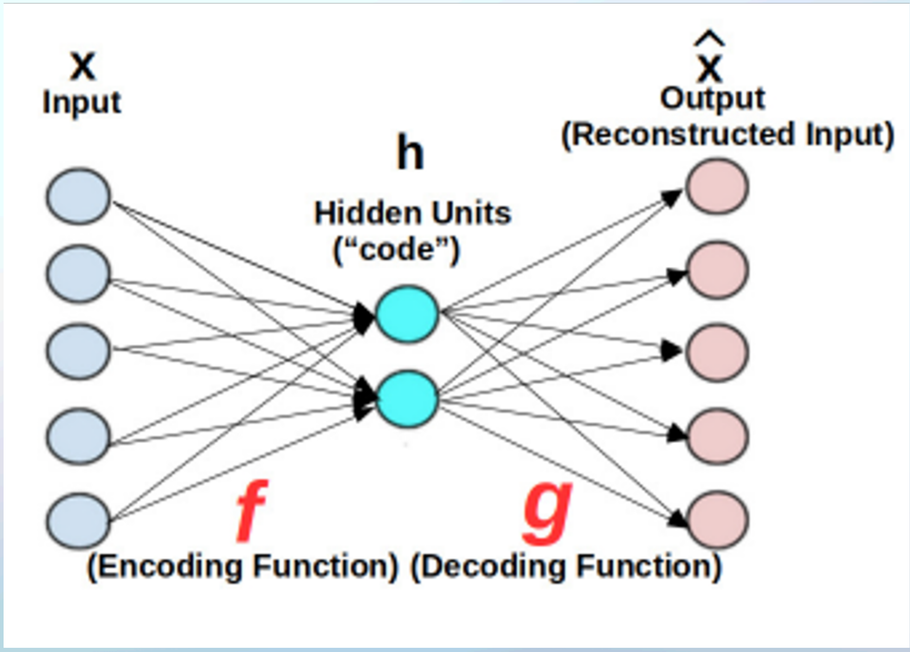
Autoencoder

- An autoencoder is a special type of neural network that is trained to copy its input to its output.
- Autoencoders are an unsupervised learning technique in which we leverage neural networks for the task of ***representation learning.***
- An autoencoder learns to compress the data while minimizing the reconstruction error.



Input Image

Output Image



Autoencoder

- ***For higher dimensional data, autoencoders are capable of learning a complex representation of the data (manifold) which can be used to describe observations in a lower dimensionality and correspondingly decoded into the original input space.***

- For example, given an image of a handwritten digit, an autoencoder first encodes the image into a lower dimensional latent representation, then decodes the latent representation back to an image.

- Suppose we have a set of data points. $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$, where each data point has many dimensions

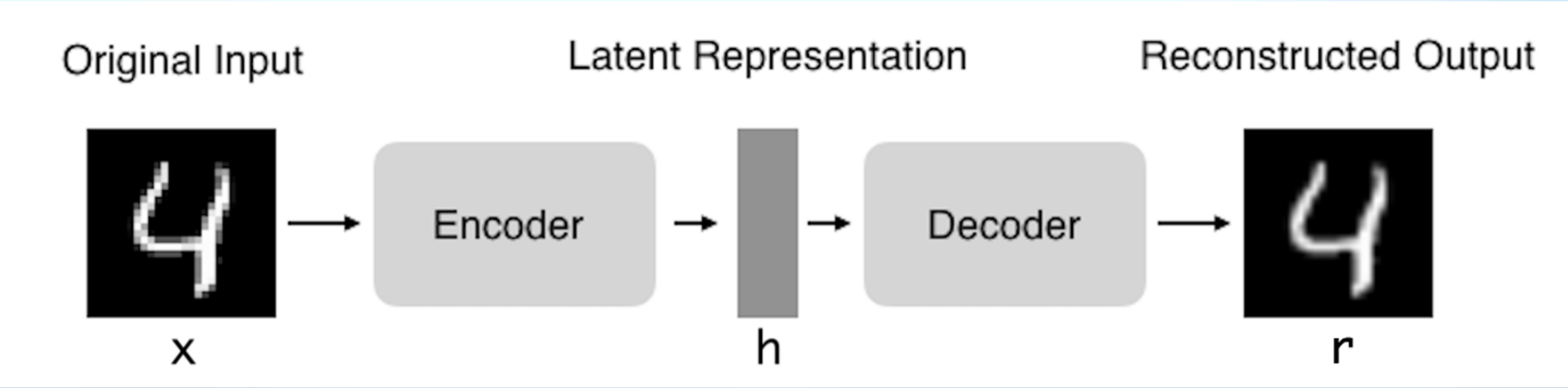
- to map to another set of data points $\{z^{(1)}, z^{(2)}, ..., z^{(m)}\}$ where z's have lower dimensionality than x's and z's can faithfully reconstruct x's.

Autoencoder

- Encoder: compress input into a latent-space of usually smaller dimension. h = f(x)
- Decoder: reconstruct input from the latent space. r = g(f(x)) with r as close to x as possible
- To map data back and forth, z and $\tilde{x}$ are functions

$$z^{(i)} = W_1 x^{(i)} + b_1$$

- $$\tilde{x}^{(i)} = W_2 z^{(i)} + b_2$$



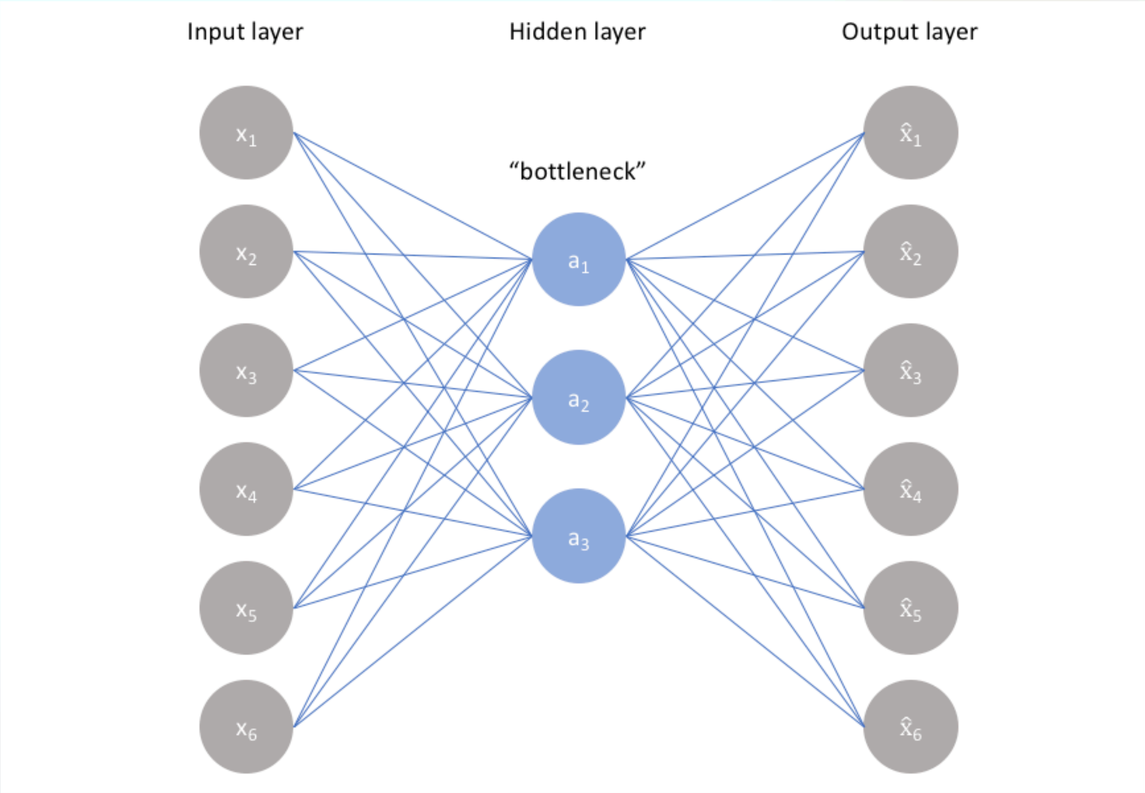
Autoencoder

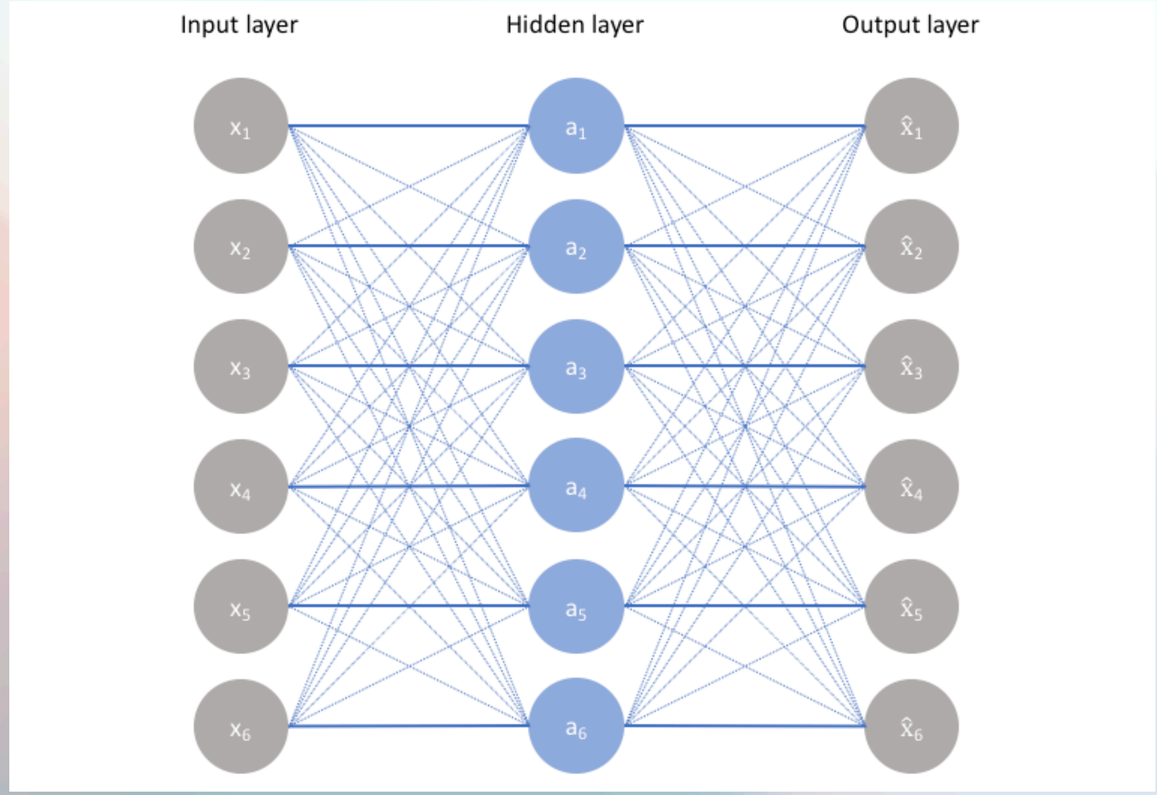
- Our goal is to have $\tilde{x}^{(i)}$ to approximate $x^{(i)}$ using objective function, which is the sum of squared diferences between $\tilde{x}^{(i)}$ and $x^{(i)}$
- $$\begin{aligned} J(W_1, b_1, W_2, b_2) &= \sum_{i=1}^{m} \left(\tilde{x}^{(i)} - x^{(i)} \right)^2 \\ &= \sum_{i=1}^{m} \left(W_2 z^{(i)} + b_2 - x^{(i)} \right)^2 \\ &= \sum_{i=1}^{m} \left(W_2 \left(W_1 x^{(i)} + b_1 \right) + b_2 - x^{(i)} \right)^2 \end{aligned}$$
- which can be minimized using stochastic gradient descent

Properties of Autoencoders

- Data-specific: Autoencoders are only able to compress data similar to what they have been trained on.
- Lossy: The decompressed outputs will be degraded compared to the original inputs.
- Learned automatically from examples: It is easy to train specialized instances of the algorithm that will perform well on a specific type of input.

Reconstruction Error





A bottleneck constrains the amount of information that can traverse the full network, forcing a learned compression of the input data.

Autoencoder

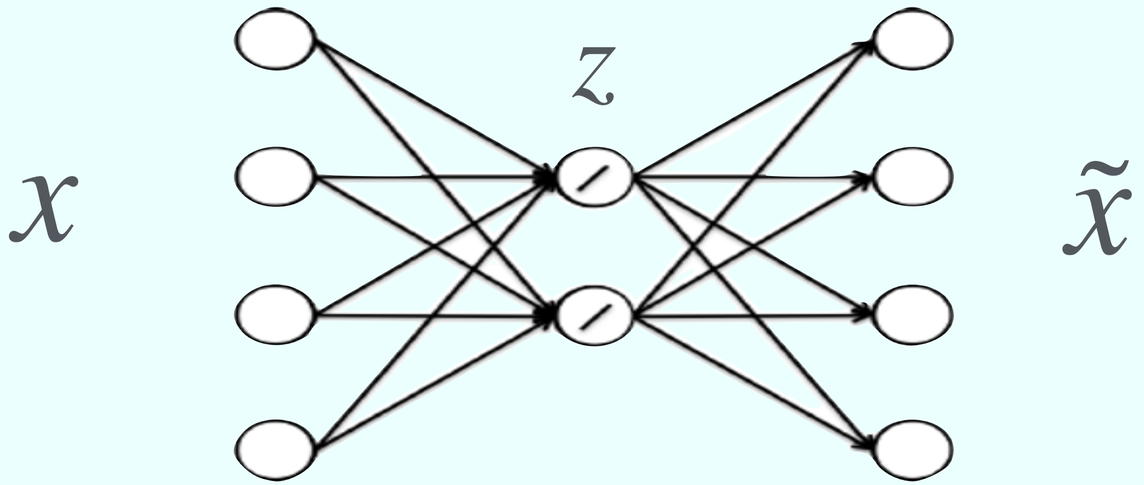
The ideal autoencoder model balances the following:

- 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.

Undercomplete autoencoder

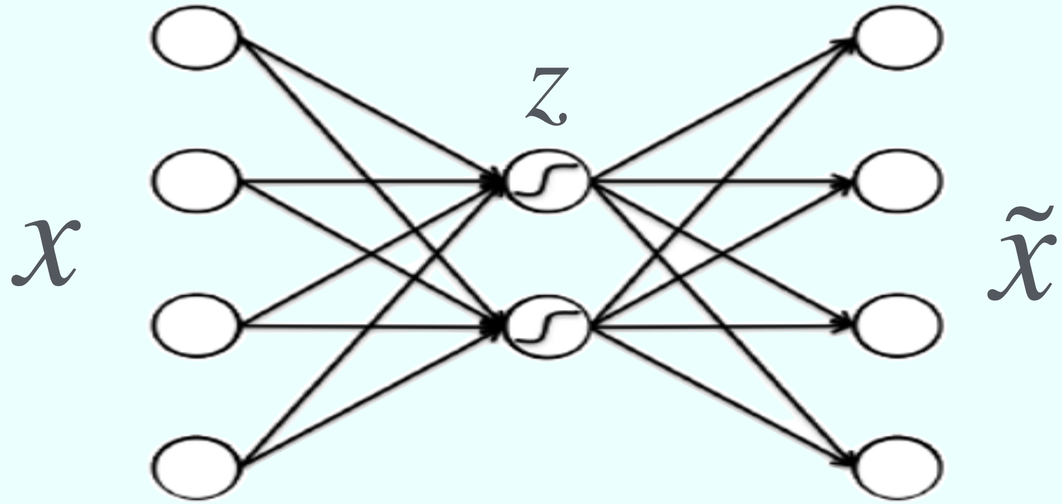
- Important to train the autoencoder to perform the input copying task will result in h (hidden unit) taking on useful properties from input
- features from the autoencoder is to constrain h to have a smaller dimension than x
- An autoencoder whose code dimension is less than the input dimension is called undercomplete.

Linear Autoencoder



- autoencoder maps data from 4 dimensions to 2 dimensions, with one hidden layer, The activation function of the hidden layer is linear
- works for the case where the data lie on a linear surface

Non-linear Autoencoder



- If the data lie on a nonlinear surface it makes more sense to use a nonlinear autoencoder
- If the data is highly nonlinear, one could add more hidden layers to the network to have a deep autoencoder

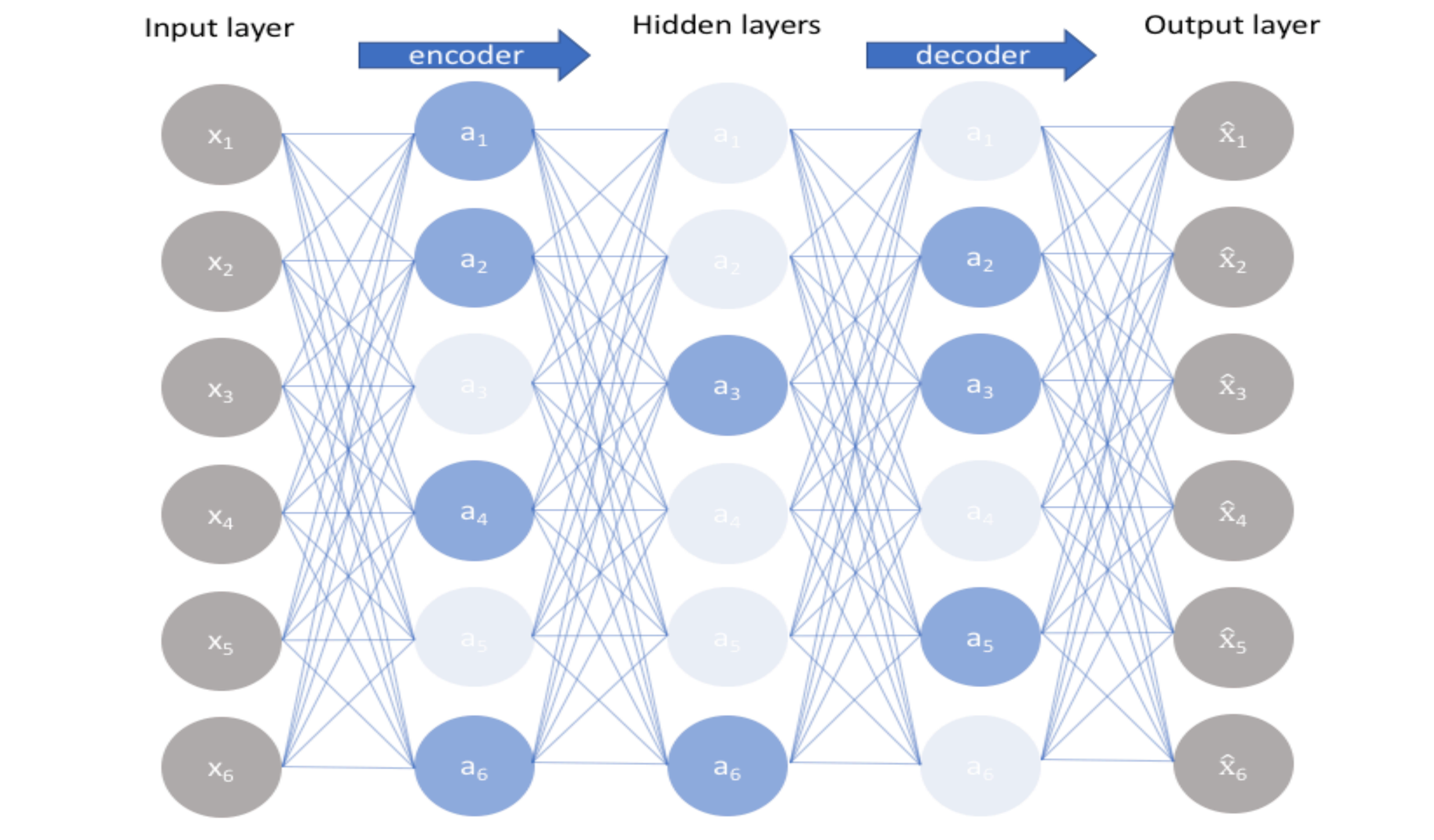
Types of Autoencoders

- Undercomplete autoencoders
- Sparse Autoencoders
- Denoising autoencoders
- Variational Autoencoders (for generative modelling)
- Contractive Autoencoders(CAE)

Sparse Autoencoder

- In an autoencoder , may the number of hidden units is large (perhaps even greater than the number of input pixels).
- It can still discover interesting structure, by imposing other constraints on the network.
- In particular, impose a sparsity constraint on the hidden units, then the autoencoder will still discover interesting structure in the data, even if the number of hidden units is large
- ρ is a sparsity parameter, typically a small value close to zero.
- we would like the average activation of each hidden neuron to be close to 0.05 (ρ)
- To achieve, add an extra penalty term to our optimization objective that penalizes $\hat{\rho}_j$ deviating significantly from ρ

Sparse Autoencoder (cont'd)



Sparse Autoencoder (cont'd)

- An autoencoder with sparsity penalty Ω(h) on the code layer h , in addition to the reconstruction error while training $L(x, g(f(x))) + \Omega(h)$
- where g(h) is the decoder output, h =f(x), the encoder output
- Incorporating sparsity forces more neurons to be inactive
- Sparse autoencoders attempt to enforce the constraint i.e sparsity parameter(Ω(h)).
- This penalizes the neurons that are too active, forcing them to activate less
- forces the model to only have a small number of hidden units being activated at the same time

Sparse Autoencoder (cont'd)

- sparsity penalty can yield a model that has learned useful features as a byproduct
- sparsity penalty function, prevents the neural network from activating more neurons and serves as a regularizer
- There are two main ways by which we can impose this sparsity constraint; both involve measuring the hidden layer activations for each training batch and adding some term to the loss function in order to penalize excessive activations. These terms are:
- L_1 **Regularization**: add a term to the loss function that penalizes the absolute value of the vector of activations α in layer h for observation i, scaled by a tuning parameter λ,

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

Sparse Autoencoder (cont'd)

- KL-Divergence: In essence, KL-divergence is a measure of the difference between two probability distributions. We can define a sparsity parameter ρ which denotes the average activation of a neuron over a collection of samples. This expectation can be calculated as $\hat{\rho}_j = \frac{1}{m} \sum_i \left[a_i^{(h)}(x) \right]$ where the subscript j denotes the specific neuron in layer h, summing the activations for training observations denoted individually as x.

- In essence, by constraining the average activation of a neuron over a collection of samples we're encouraging neurons to only fire for a subset of the observations.

- ρ can be described as a Bernoulli random variable distribution such that we can leverage the KL divergence (expanded next) to compare the ideal distribution to the observed distributions over all hidden layer nodes $\mathscr{L}\left(x, \hat{x}\right) + \sum_j KL\left(\rho \,||\, \hat{\rho}_j\right)$

KL Divergence

- A measure of how one probability distribution Q is different from a second, reference probability distribution P.

- The KL divergence between two probability distributions P and Q is the sum, over all possible outcomes x, of the probability of x under distribution PP multiplied by the logarithm of the ratio of the probability of x under P to the probability of x under Q.

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

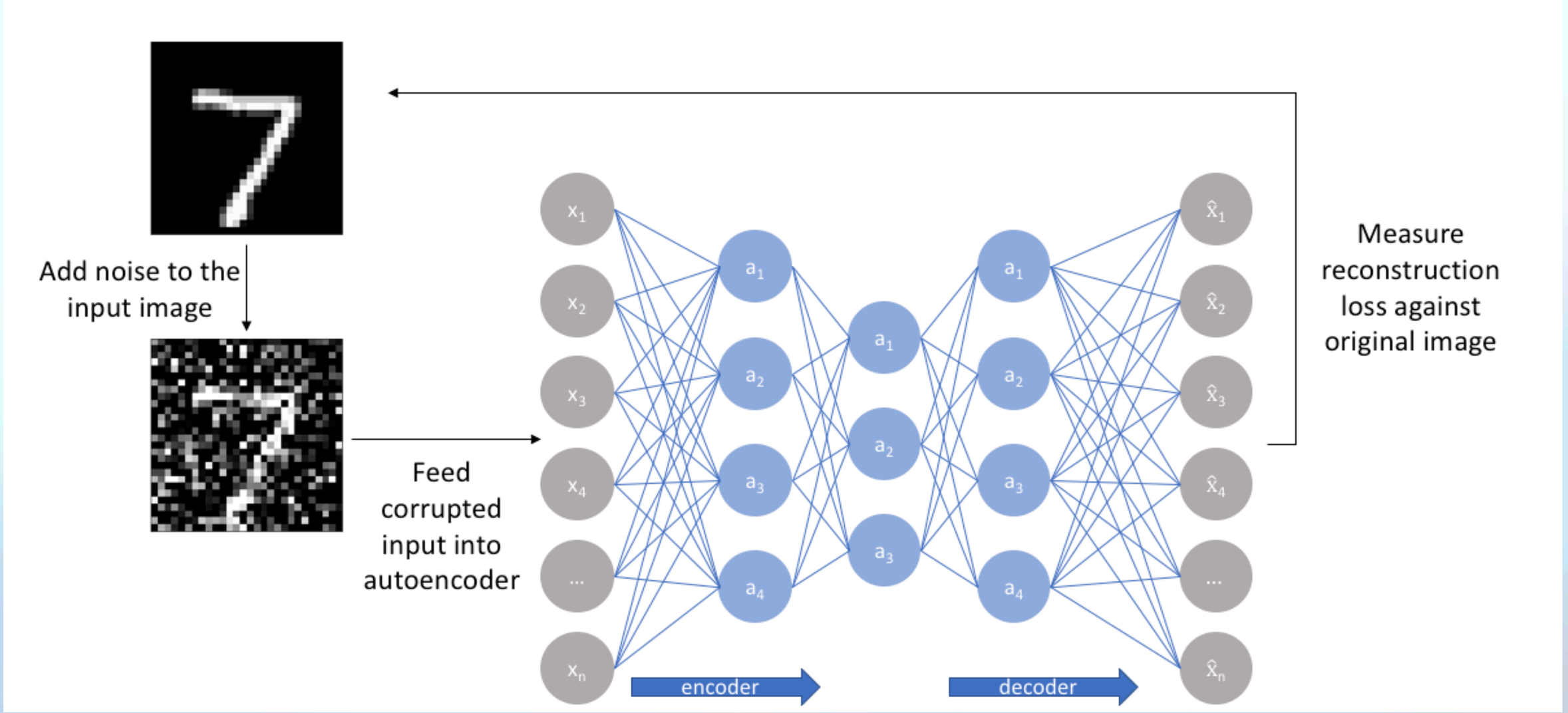
Sparse Autoencoder (cont'd)

- Note: A Bernoulli distribution is "*the probability distribution of a random variable which takes the value 1 with probability* p *and the value 0 with probability* $q = 1 - p$*"*. This corresponds quite well with establishing the probability a neuron will fire.

- The KL divergence between two Bernoulli distributions can be written as

$$\sum_{j=1}^{l^{(h)}} \rho \log \frac{\rho}{\hat{\rho}_j} + \left(1 - \rho\right) \log \frac{1 - \rho}{1 - \hat{\rho}_j}$$

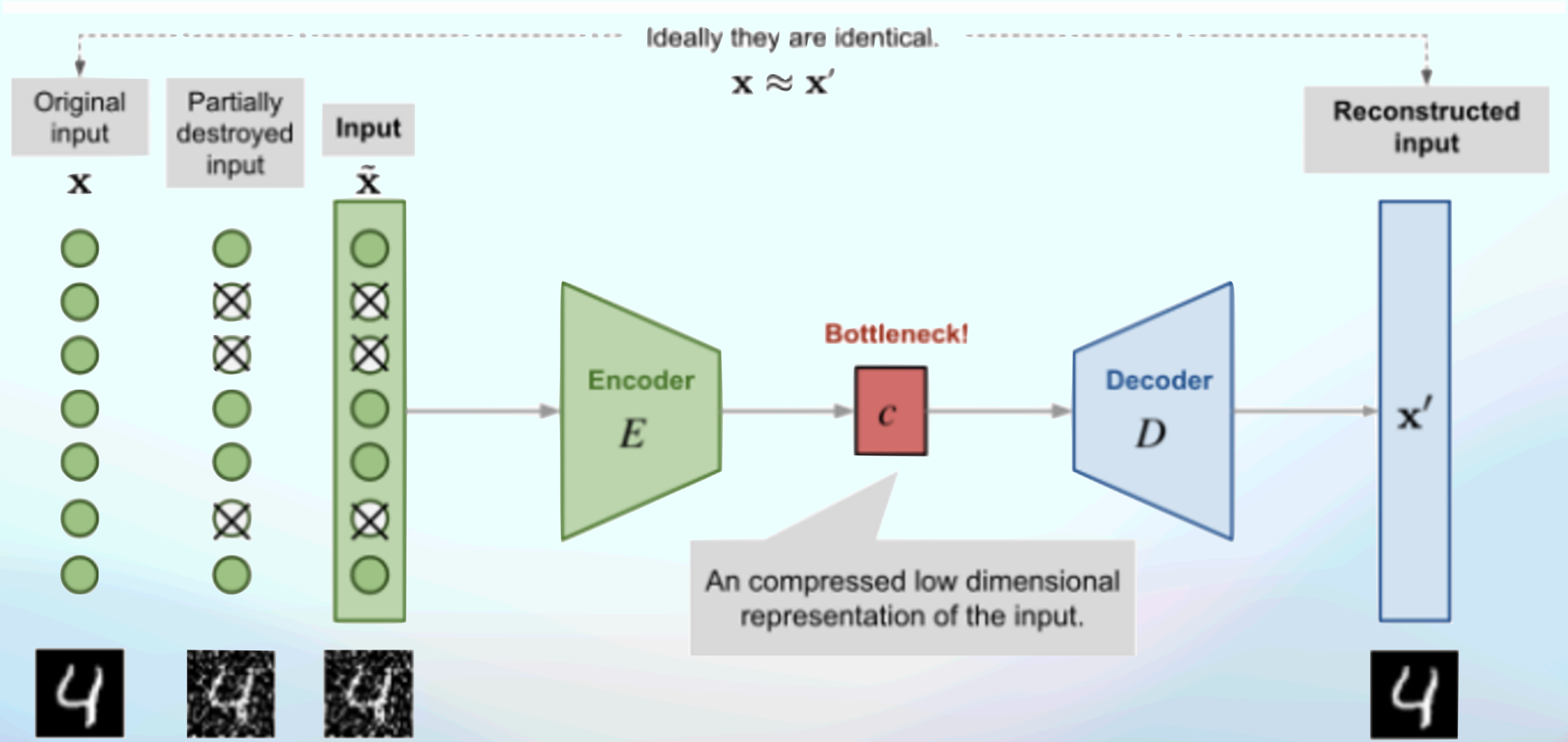
Denoising Autoencoder



Denoising Autoencoder

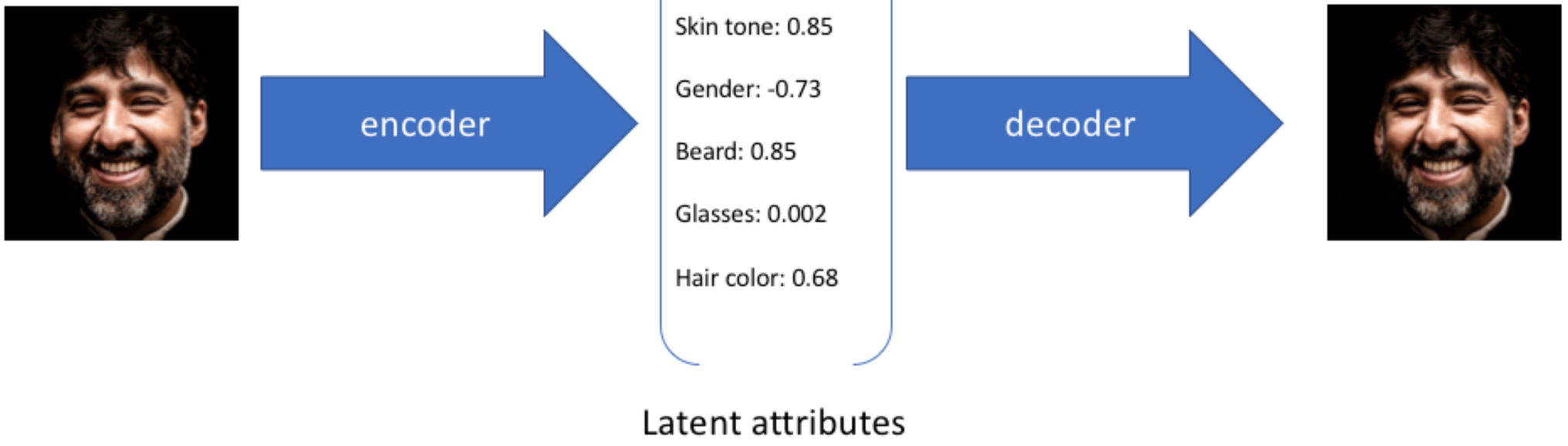
- One approach towards developing a generalizable model is to slightly corrupt the input data but still maintain the uncorrupted data as our target output.
- The denoising autoencoder (DAE) is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output
- we train the autoencoder to reconstruct the input from a corrupted copy of the inputs. This forces the codings to learn more robust features of the inputs
- The input is partially corrupted by adding noises to or masking some values of the input vector in a stochastic manner

Denoising Autoencoder

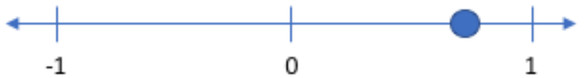


$$L_{\text{DAE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\tilde{\mathbf{x}}^{(i)})))^2$$

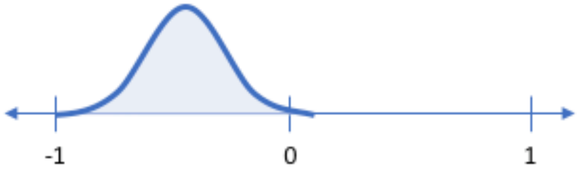
Variational Autoencoder



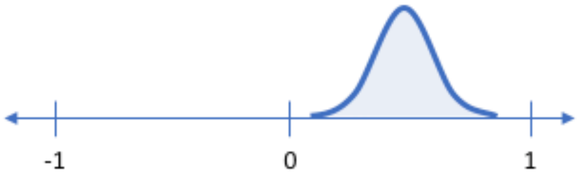
Smile (discrete value)

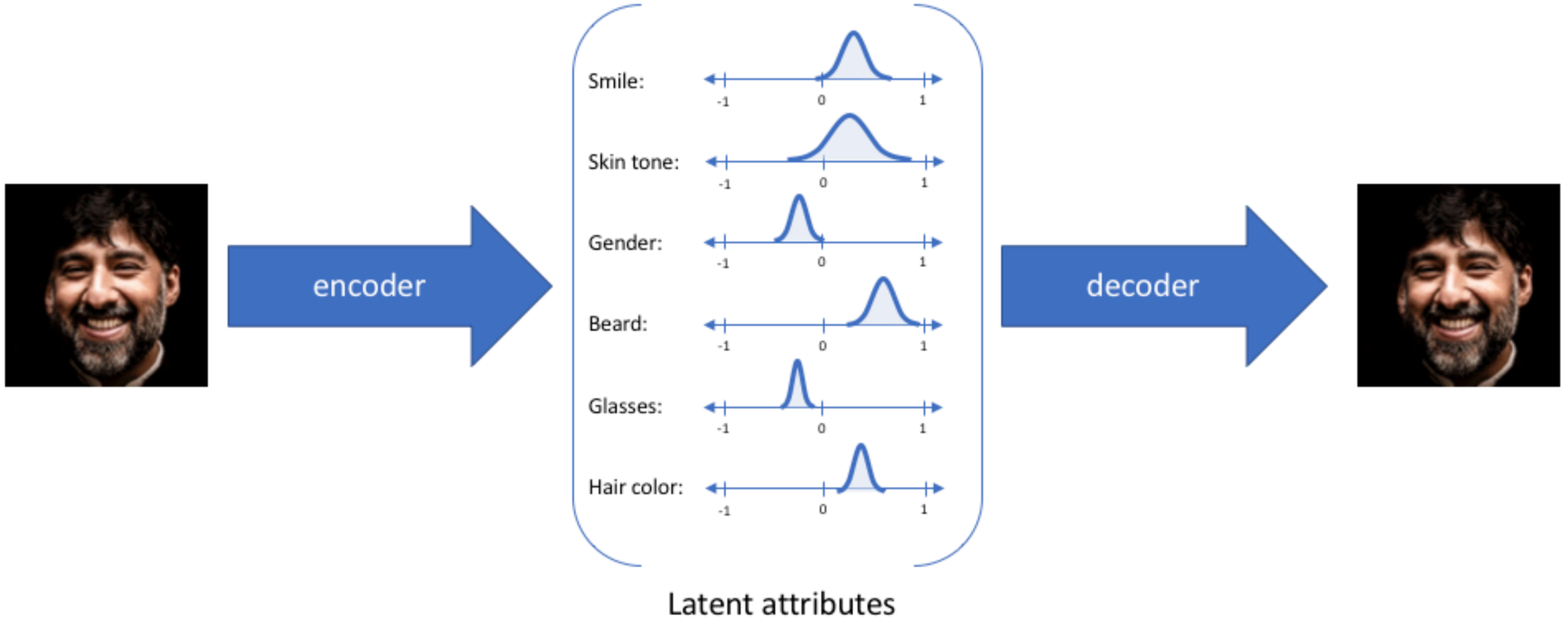


Smile (probability distribution)

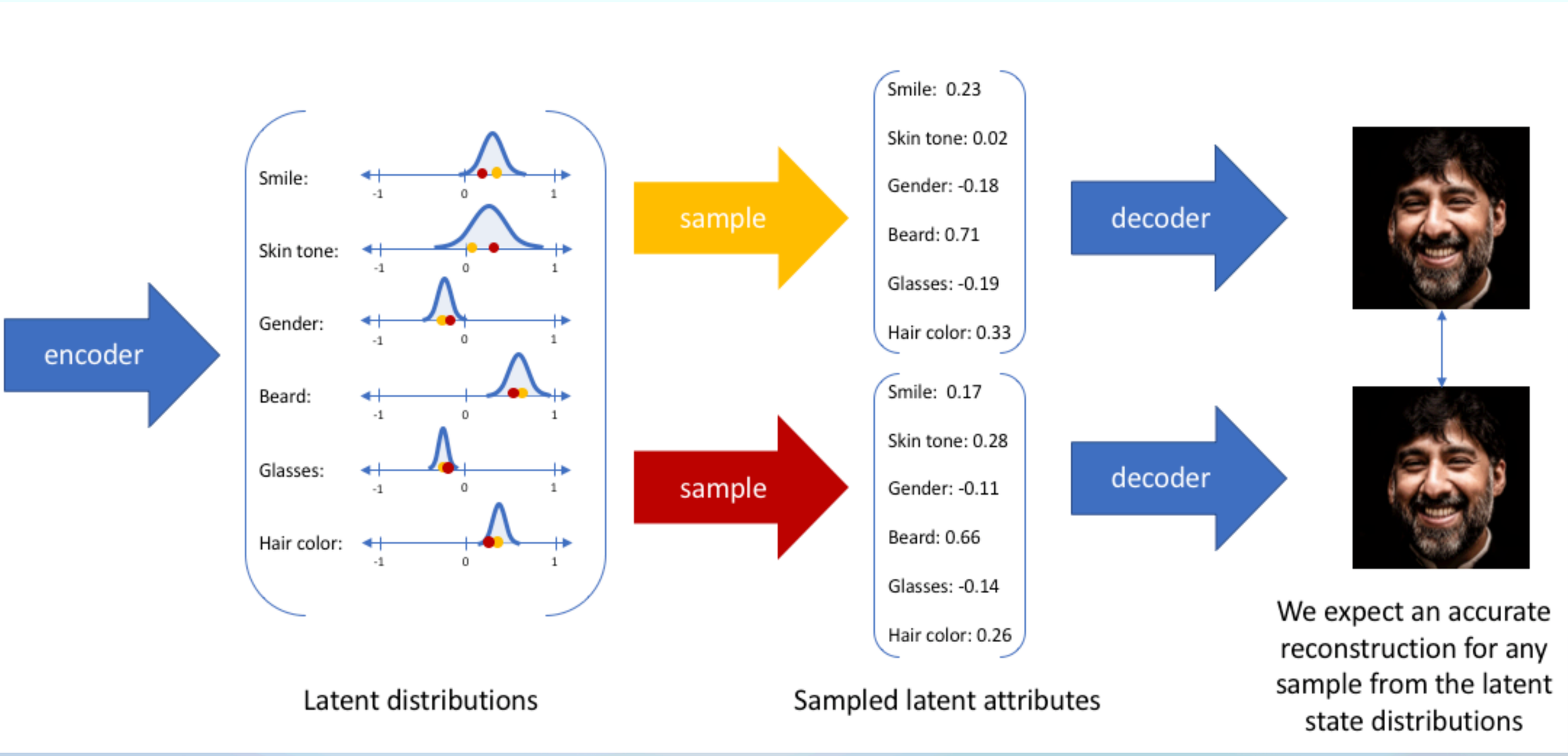


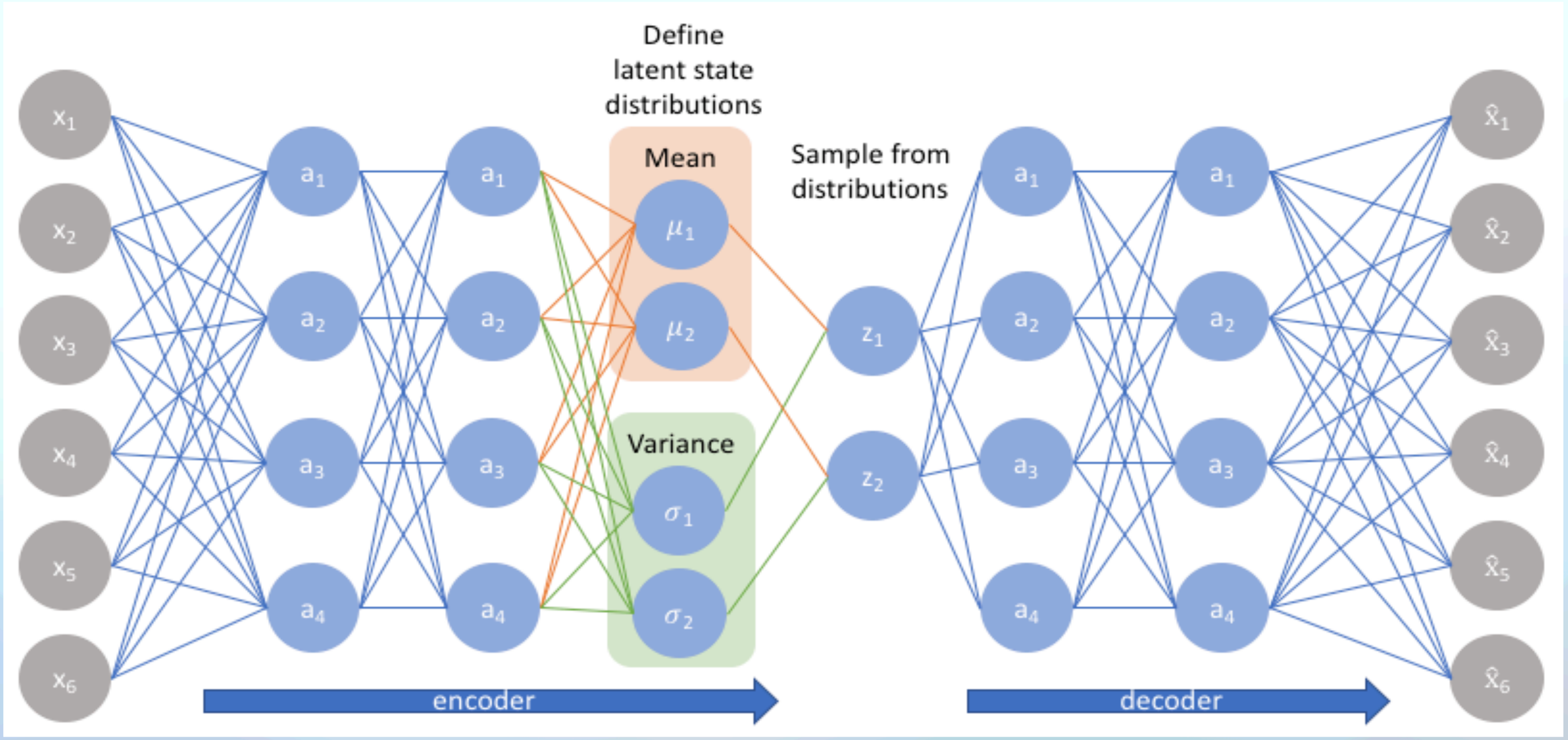
vs.




encoder
Smile:
Skin tone:
Gender:
Beard:
Glasses:
Hair color:
-1
0
1
decoder
Latent attributes

- Variational autoencoder was proposed in 2013 by Diederik P. Kingma and Max Welling at Google and Qualcomm.
- A variational autoencoder (VAE) provides a probabilistic manner for describing an observation in latent space.
- 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.
- It has many applications, such as data compression, synthetic data creation, etc.
- For variational autoencoders, the encoder model is sometimes referred to as the recognition model whereas the decoder model is sometimes referred to as the generative model.


encoder
Smile:
Skin tone:
Gender:
Beard:
Glasses:
Hair color:
-1
0
1
sample
sample
Smile: 0.23
Skin tone: 0.02
Gender: -0.18
Beard: 0.71
Glasses: -0.19
Hair color: 0.33
Smile: 0.17
Skin tone: 0.28
Gender: -0.11
Beard: 0.66
Glasses: -0.14
Hair color: 0.26
decoder
decoder
Latent distributions
Sampled latent attributes
We expect an accurate reconstruction for any sample from the latent state distributions


Define
latent state
distributions
Mean
Variance
Sample from
distributions
encoder
decoder

How does VAE work?

- The encoder network takes raw input data and transforms it into a probability distribution within the latent space.
- The latent code generated by the encoder is a probabilistic encoding, allowing the VAE to express not just a single point in the latent space but a distribution of potential representations.
- The decoder network, in turn, takes a sampled point from the latent distribution and reconstructs it back into data space. During training, the model refines both the encoder and decoder parameters to minimize the reconstruction loss – the disparity between the input data and the decoded output.
- The goal is not just to achieve accurate reconstruction but also to regularize the latent space, ensuring that it conforms to a specified distribution.

- The reconstruction loss compels the model to accurately reconstruct the input, while the regularization term encourages the latent space to adhere to the chosen distribution, preventing overfitting and promoting generalization.

$$\mathscr{L}\left(x,\hat{x}\right)+\sum_{j}KL\left(q_{j}\left(z|x\right)||p\left(z\right)\right)$$

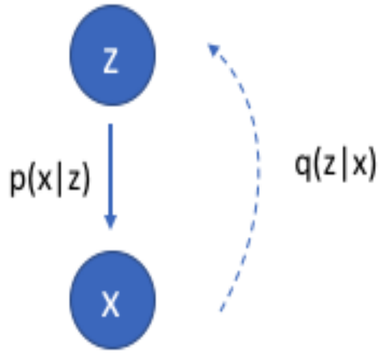
- Where $q\left(z|x\right)$ is the learned distribution and $p\left(z\right)$ is the true prior distribution , which we'll assume follows a unit Gaussian distribution, for each dimension j of the latent space.

Statistical

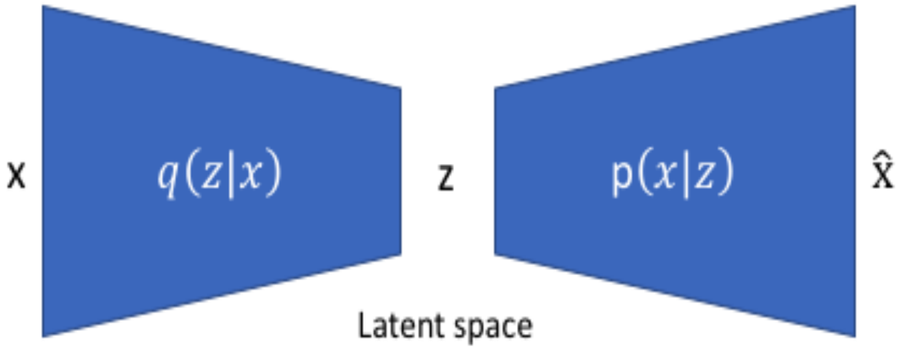
$$p\left(z\,|\,x\right) = \frac{p\left(x\,|\,z\right)p\left(z\right)}{p\left(x\right)}$$

$$p\left(x\right) = \int p\left(x\,|\,z\right)p\left(z\right)dz$$

$$\min KL\left(q\left(z|x\right)||p\left(z|x\right)\right)$$



We'd like to use our observations to understand the hidden variable.



Latent space representation.

Neural network mapping x to z.

Neural network mapping z to x.

$$E_{q(z|x)}\log p\left(x|z\right) - KL\left(q\left(z|x\right)||p\left(z\right)\right)$$

Exercise

- Two GAN generators have learned different distributions over 4 categories:

  Generator G1 :

  PG1=[0.3,0.3,0.2,0.2]

  Generator G2 :

  PG2 =[0.25,0.25,0.25,0.25]

  The real data distribution is:

  Preal=[0.35,0.3,0.2,0.15]

  Compute the KL divergence between Preal and PG1 , and between Preal and PG2 .

  Based on KL divergence, which generator better approximates the real data distribution? Discuss how KL divergence helps compare the quality of the two generators in GAN training.