Deep Unsupervised Learning
Autoencoder Family
Variational Autoencoder Family

DEEP LEARNING COURSE – 2023

E. FATEMIZADEH

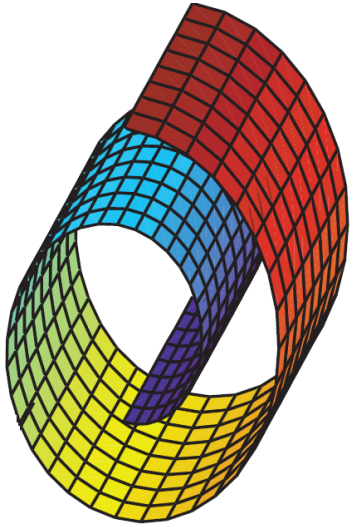
Sharif University of Technology, School of Electrical Engineering

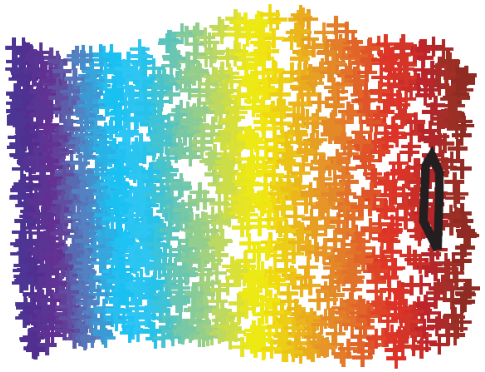
Deep Unsupervised Learning

- Main Area:
  - Dimensionality Reduction+
  - Clustering-
  - Probability Density Estimation+
  - Generative Models+

Dimensionality Reduction

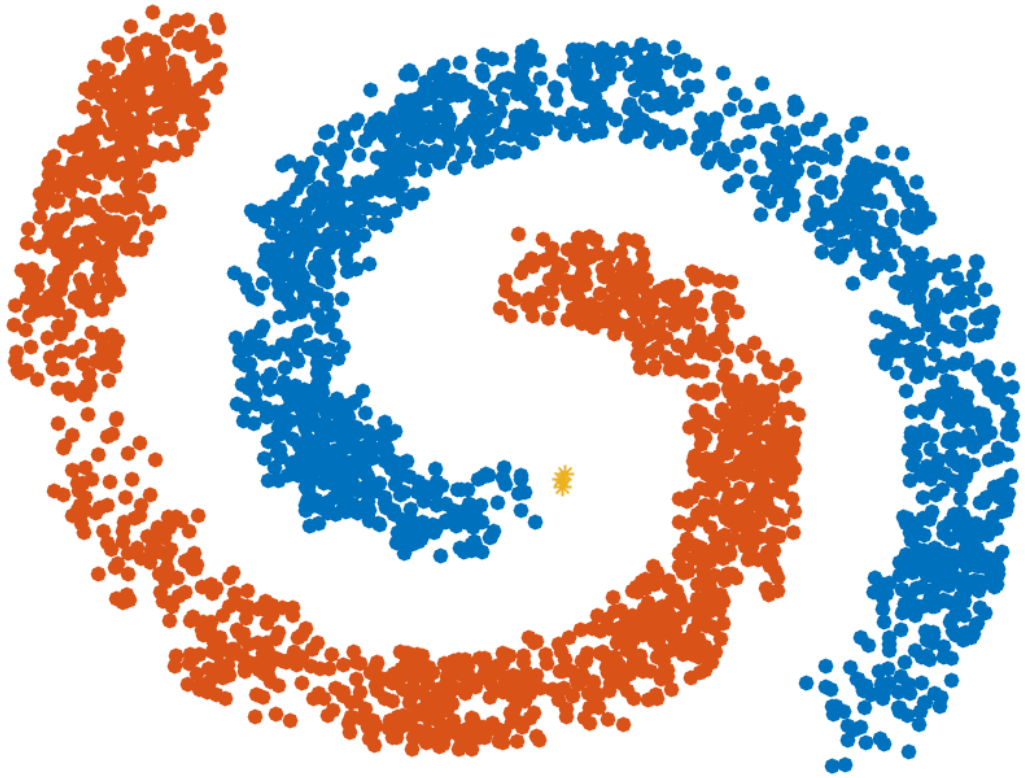
- Remove features redundancy (feature reduction), Compression, Hashing, …





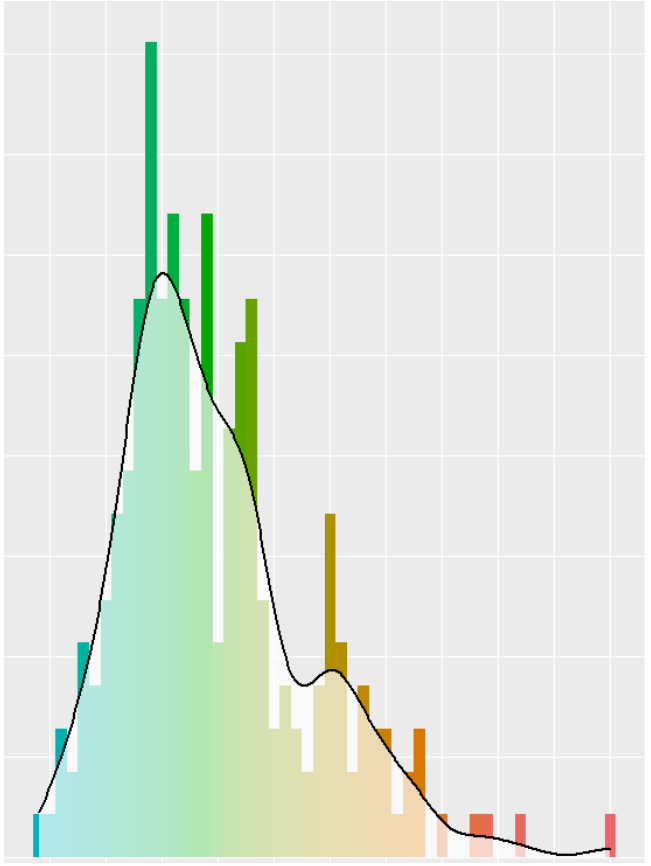
Clustering

- Segmentation (Medical/Commercial)
- WEB pages analysis
- Anomaly Detection
- …



Probability Density Estimation

- Model pdf:
  - Parametric
    - MLE, MAP, EM, …
  - Nonparametric
    - Histogram
    - KDE (Kernel Density Estimation)



Dimensionality Reduction Methods

- Supervised: LDA
- Unsupervised:
  - Principal Component Analysis (PCA)
  - Kernal PCA
  - ICA
  - Manifold Learning
  - Non-negative matrix factorization (NMF)
  - …

PCA Formulation

- Analysis:

$$Y = A^T(X - m_x) \text{ or } Y = A^T X$$

- Whitening matrix A is built using C_{xx} or R_{xx}
- Y: White random vector (zero or non-zero mean) with diagonal covariance (C_{xx}) or correlation (R_{xx}) matrix.
- Synthesize:

$$X = AY + m_x \text{ or } X = AY$$

- PCA is a generative model, generate arbitrary $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ from $N(\mathbf{0}, \boldsymbol{I})$.

PCA for Dimension Reduction

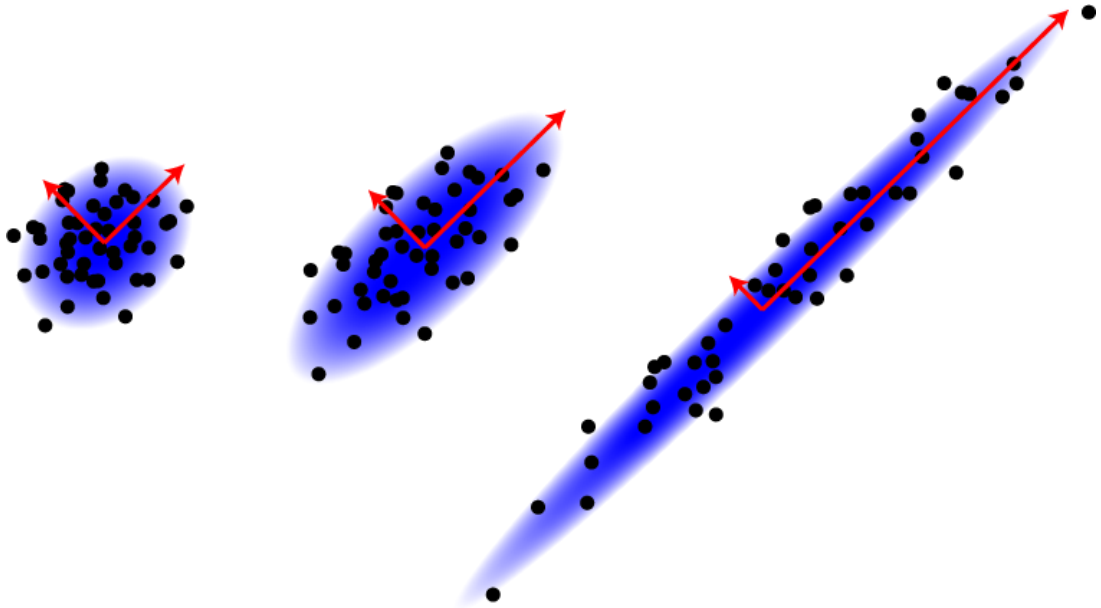
- Dimensionality Reduction:

$$\hat{X} = A_k Y_k + m_x \text{ or } \hat{X} = A_k Y_k$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \end{bmatrix}$$

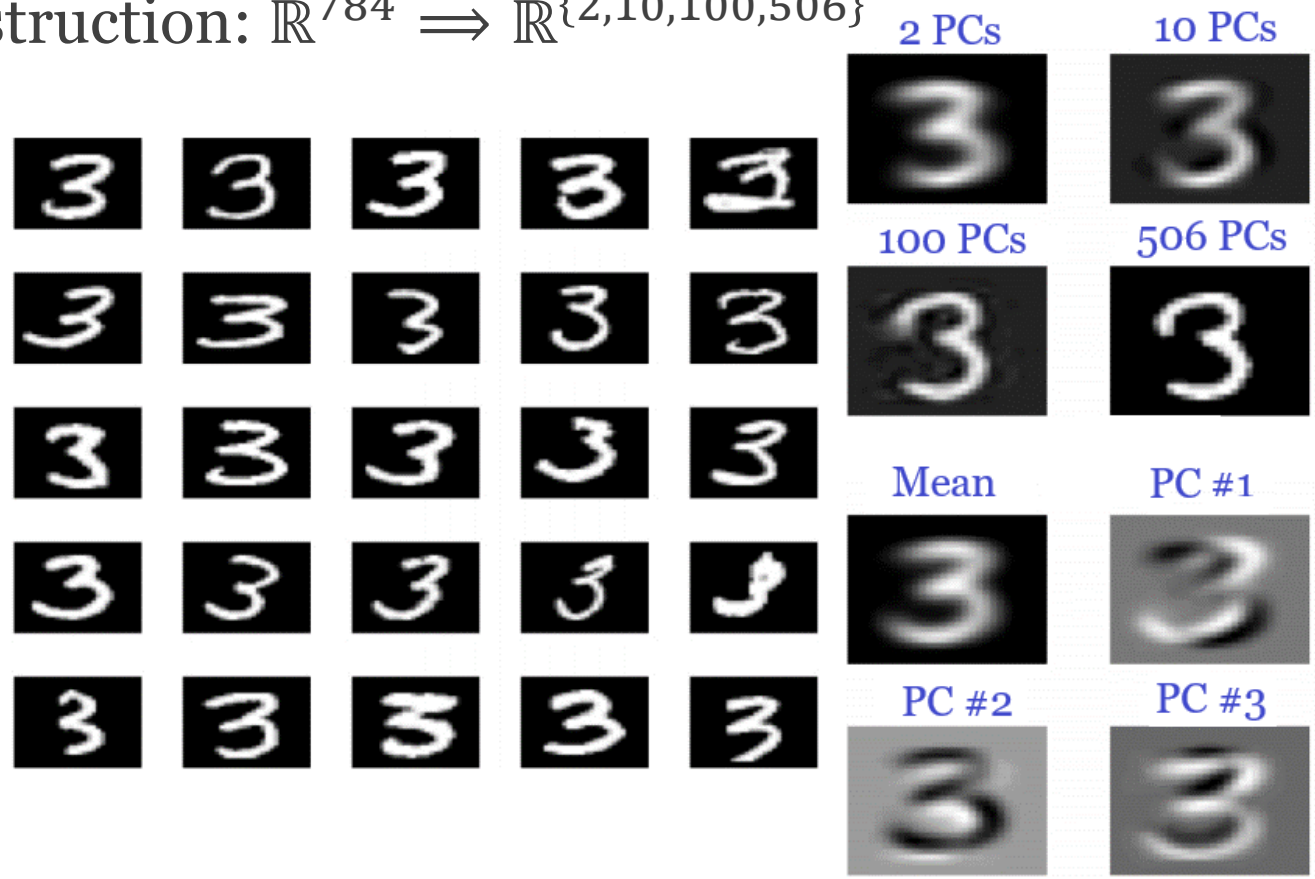
Major and minors axes

- Illustration:



PCA - Example

- PCA Reconstruction: $\mathbb{R}^{784} \Longrightarrow \mathbb{R}^{\{2,10,100,506\}}$



PCA – MSE Perspective

- PCA alternative (optimization) formulation:
- $X \in \mathbb{R}^{N \times d}$: input training matrix $(N \times d)$:

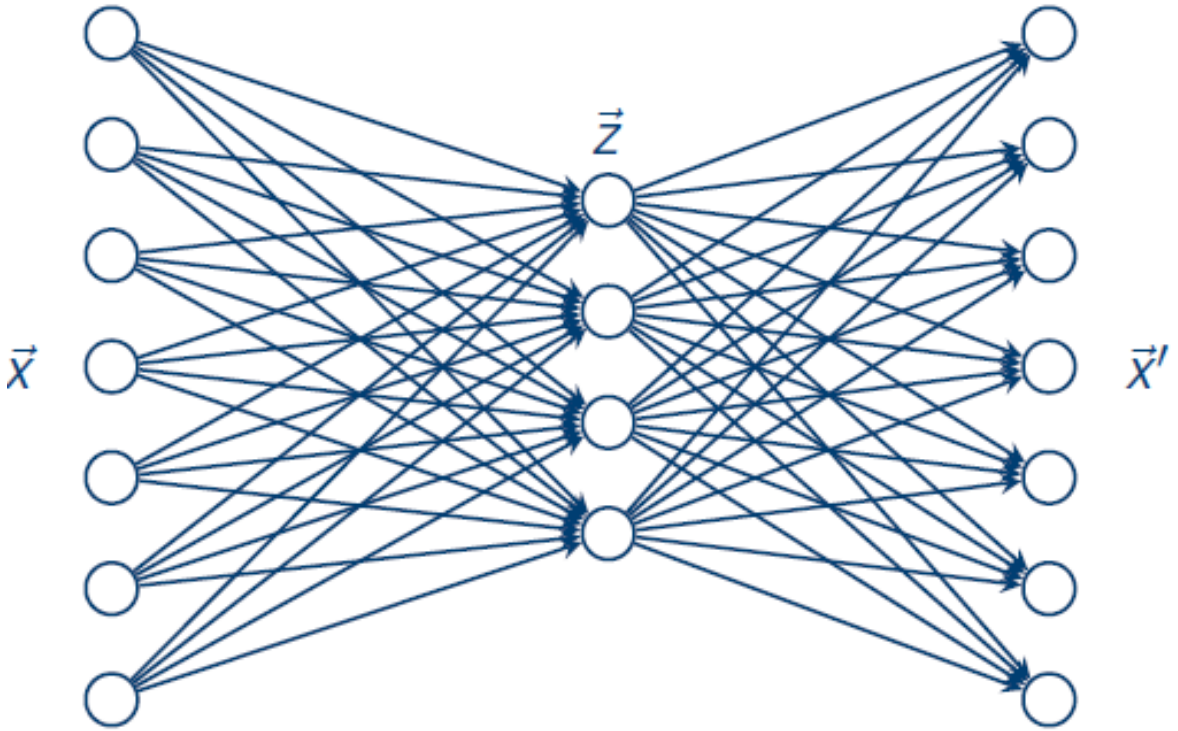
$$\min_{W} \|X - (XW)W^T\|_F^2, \qquad s.t., W^T W = I_{k \times k}, \qquad W \in \mathbb{R}^{d \times k}$$

$$\|X - (XW)W^T\|_F^2 = \left\|X - \hat{X}\right\|_F^2$$

- XW : Map to low dimension
- $(XW)W^T$: Back to high dimension

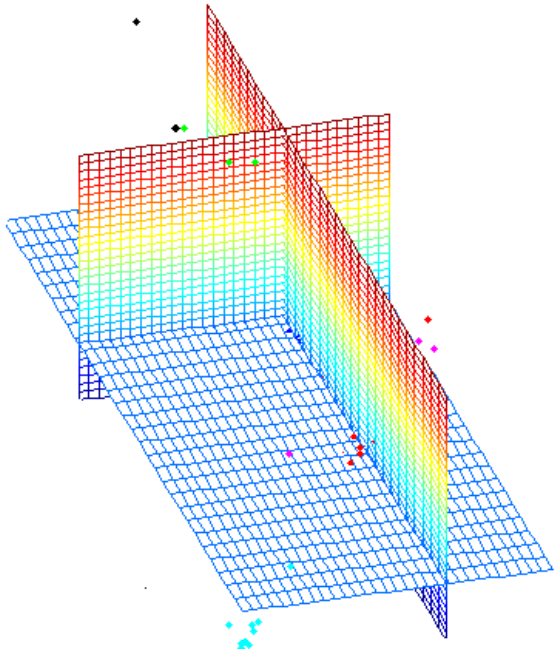
PCA as MLP

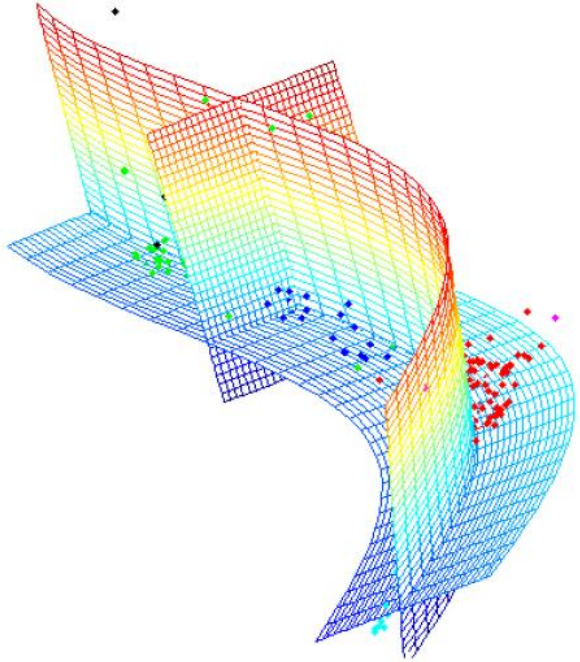
- PCA ~ Two Layer Perceptron, with constrain on weights!



PCA Limitation

- Only Capture **LINEAR** dependency!





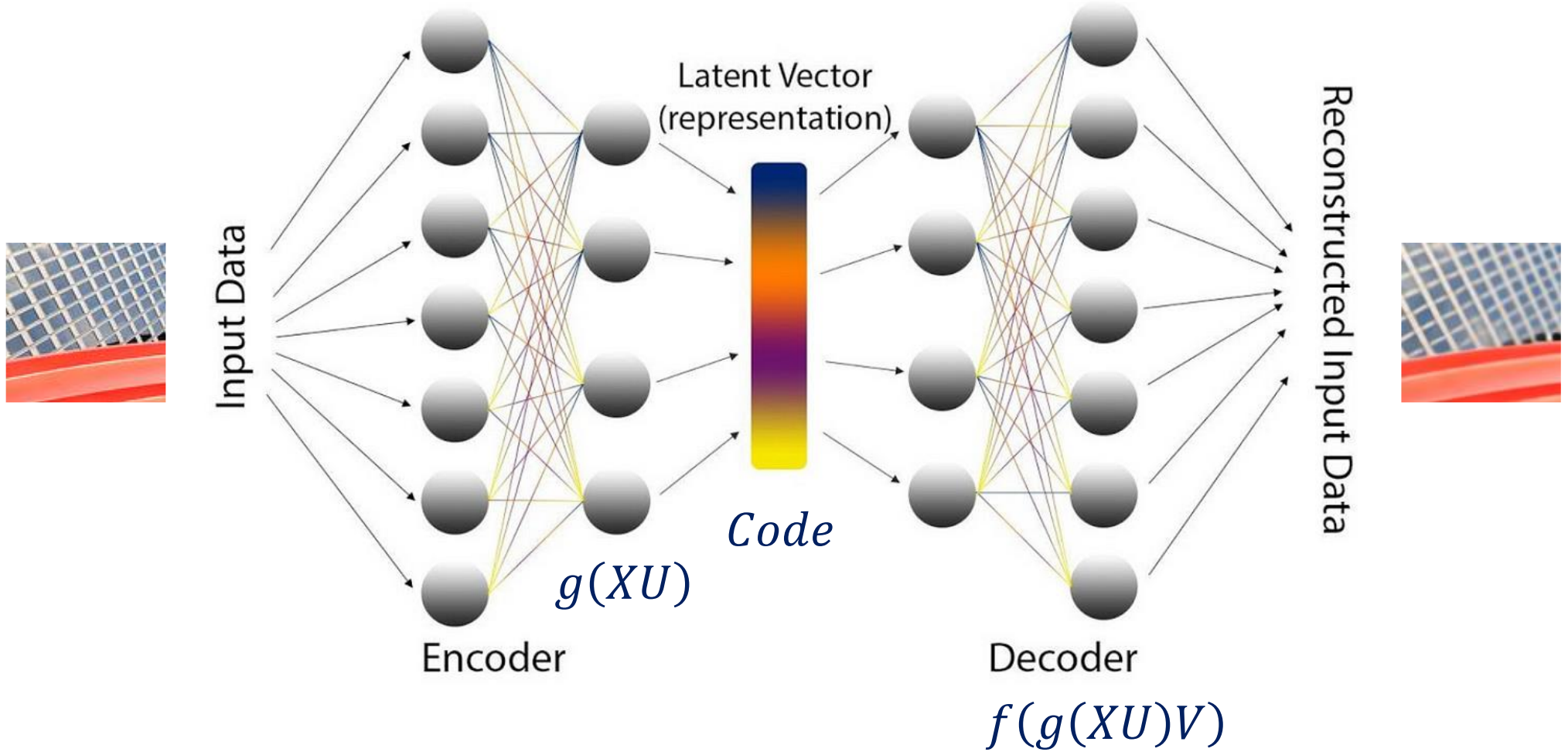
Autoencoder (AE)

- PCA conceptual generalization:

$$\min_{W} \|X - (XW)W^T\|_F^2, \qquad st.\, W^T W = I_{k\times k}$$

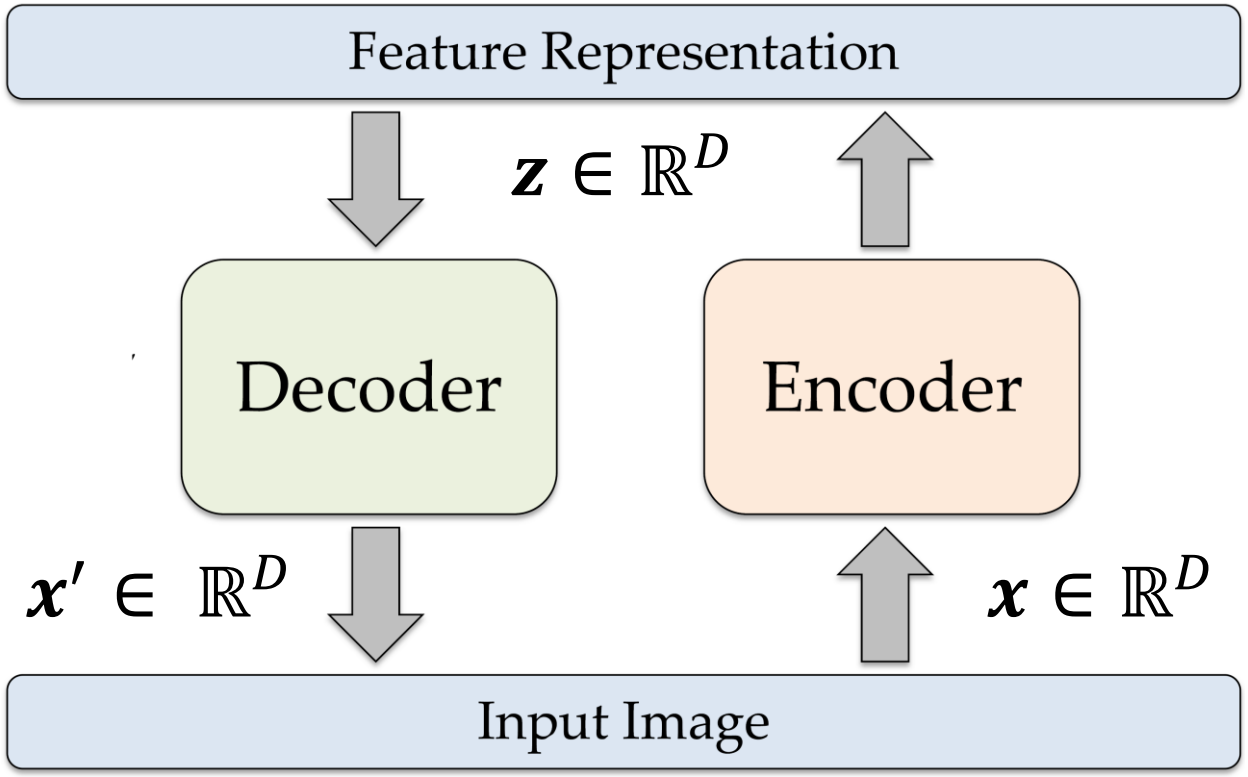
$$\min \|X - f(g(XU)V)\|_F^2$$

Autoencoder Idea



Vanilla Autoencoder

- General Framework



Vanilla Autoencoder

- Input-Code-Output dimensions:
  - Map $\boldsymbol{x} \in \mathbb{R}^D$ to nonlinear *low* dimensional manifold, $\boldsymbol{z} \in \mathbb{R}^d$ (Encoder)

$$\boldsymbol{z} = g(\boldsymbol{W}_{enc}\boldsymbol{x} + \boldsymbol{b}_{enc}), \boldsymbol{W}_{enc} \in \mathbb{R}^{d \times D}$$

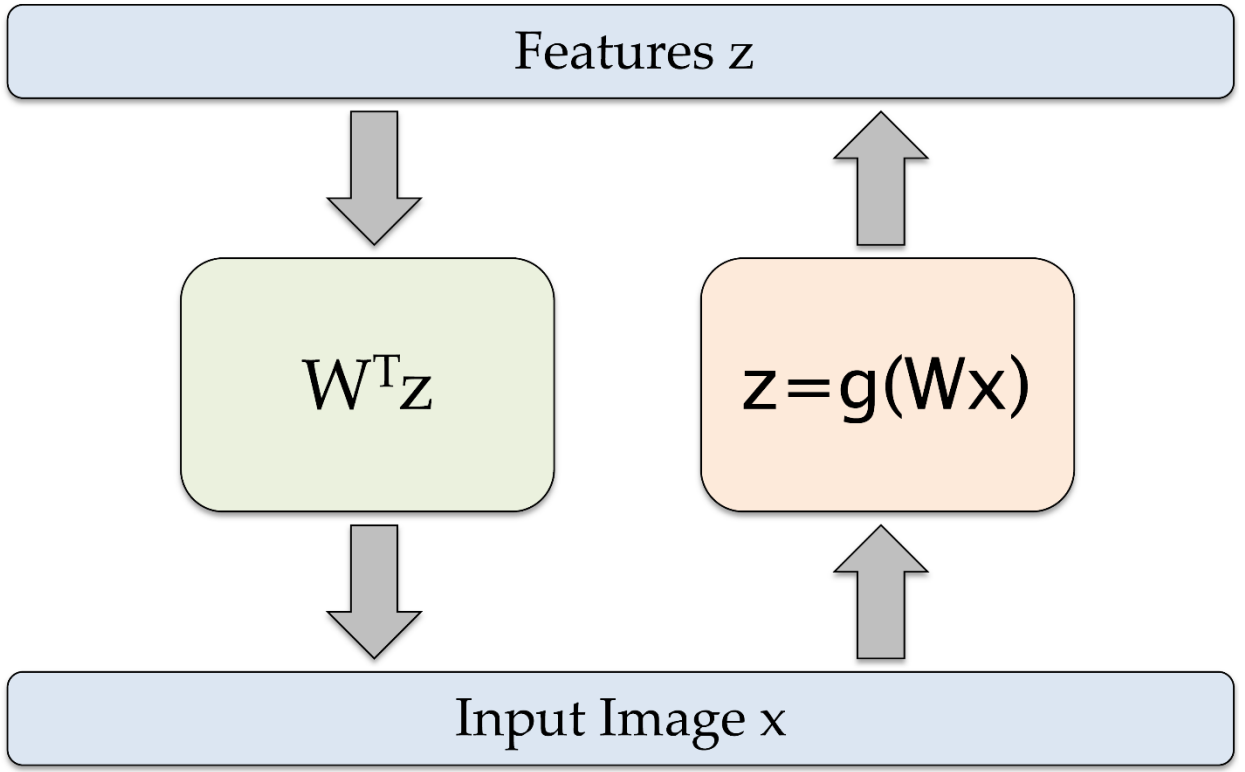
  - $g(\cdot) \in \{Relu, Sigmoid, tanh, \dots\}$,
  - Map $\boldsymbol{z} \in \mathbb{R}^d$ to nonlinear *original* dimensional manifold $\boldsymbol{x}' \in \mathbb{R}^D$ (Decoder)

$$\boldsymbol{x}' = f(\boldsymbol{W}_{dec}\boldsymbol{z} + \boldsymbol{b}_{dec}), \boldsymbol{W}_{dec} \in \mathbb{R}^{D \times d}$$

  - $f(\cdot) \in \{Linear, Relu, Sigmoid, \dots\}$, sigmoid for binary data (e.g., MNIST)
  - $f(\cdot)$ and $g(\cdot)$ are data dependent
  - Reconstruction goal: $\boldsymbol{x}' \cong \boldsymbol{x}$

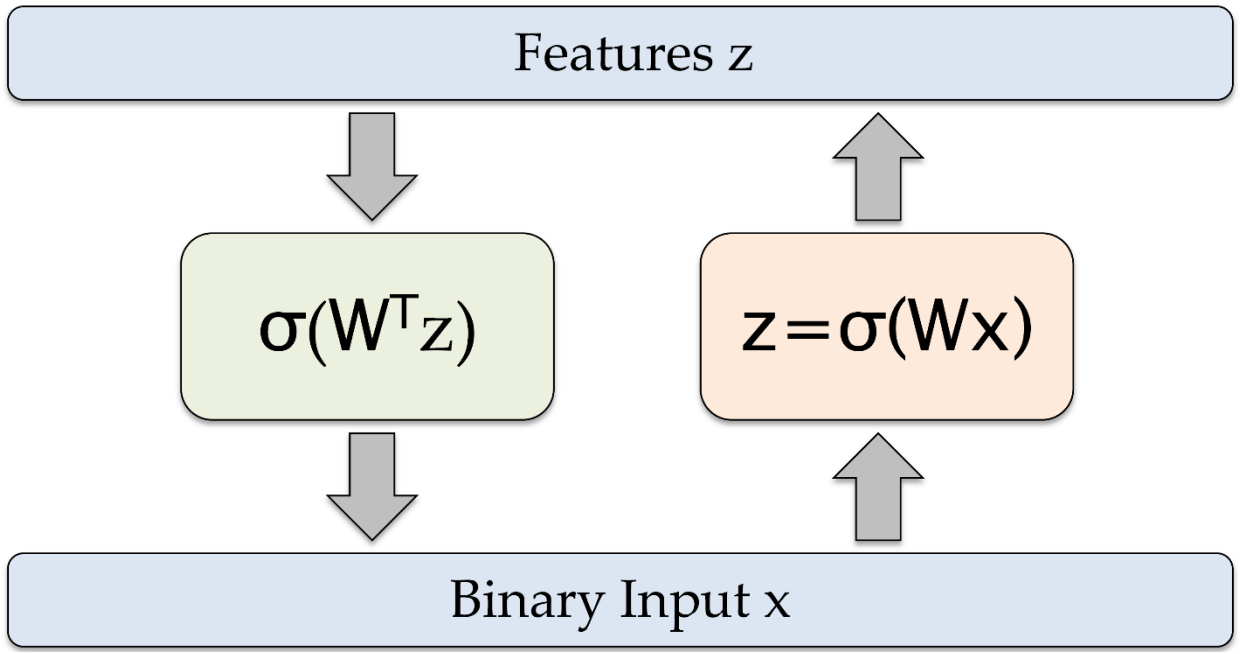
Vanilla Autoencoder

- Linear output (non-binary)

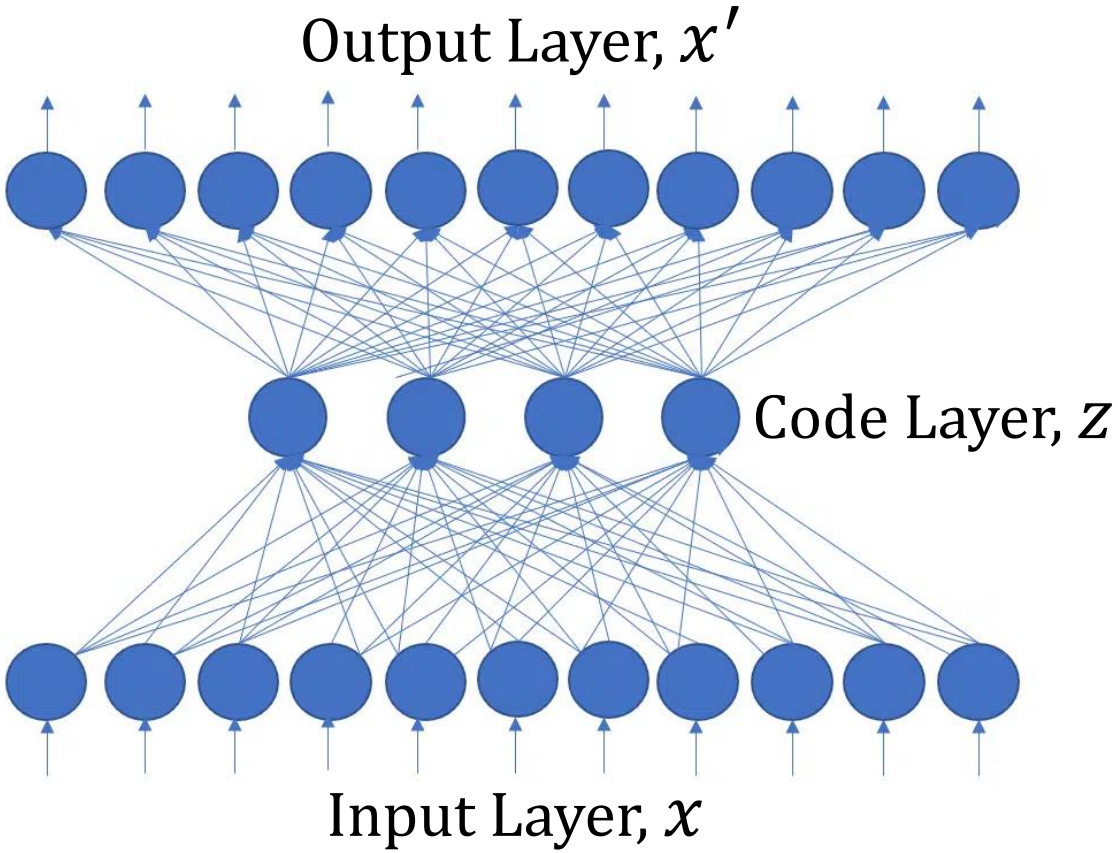


Vanilla Autoencoder

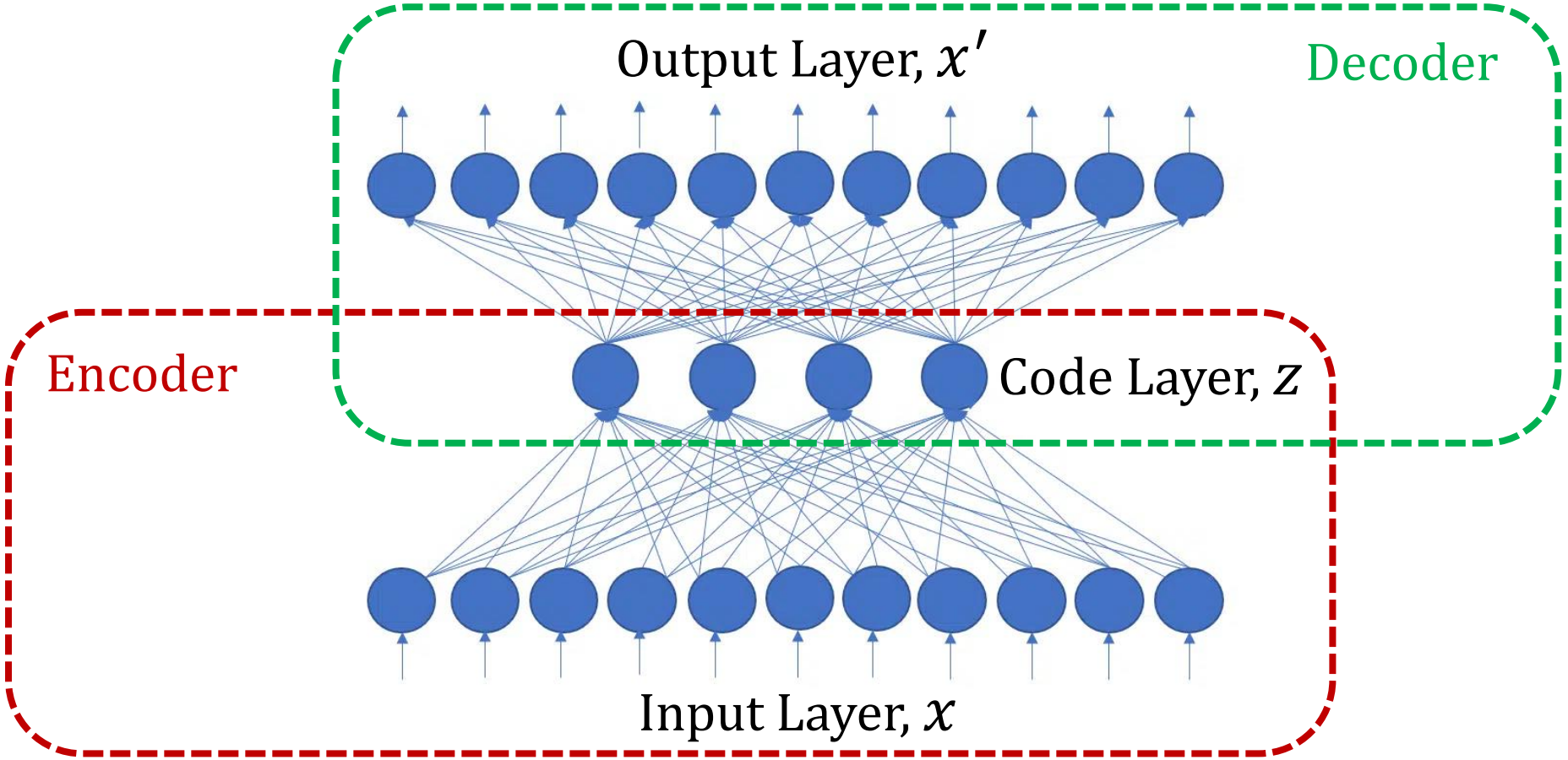
- Linear output (non-binary)



Vanilla Autoencoder



Vanilla Autoencoder



Autoencoder - Loss

- Some times and for simplification: $W_{dec} = W_{enc}^T$
- Loss functions:
  - $\mathcal{L}_2(\boldsymbol{x}, \boldsymbol{x}') = \sum_i \|\boldsymbol{x}_i - \boldsymbol{x}_i'\|_2^2$
  - $\mathcal{L}_1(\boldsymbol{x}, \boldsymbol{x}') = \sum_i \|\boldsymbol{x} - \boldsymbol{x}'\|_1$
  - $\mathcal{L}_{\text{CE}}(\boldsymbol{x}, \boldsymbol{x}') = -\sum_i \left(\boldsymbol{x}_i log \boldsymbol{x}_i' + (1 - \boldsymbol{x}_i) log(1 - \boldsymbol{x}_i')\right)$
  - $\mathcal{L}_{\text{EXP}}(\boldsymbol{x}, \boldsymbol{x}') = \tau exp\left(\frac{1}{\tau}\sum_i (\boldsymbol{x}_i - \boldsymbol{x}_i')^2\right)$
  - $\mathcal{L}_{\text{MCC}}(\boldsymbol{x}, \boldsymbol{x}') = -\sum_i \mathcal{H}_\sigma(\boldsymbol{x}_i - \boldsymbol{x}_i'), \mathcal{H}_\sigma(s) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(\frac{s^2}{2\sigma^2}\right)$

Autoencoder - Regularization

- Regularization

$$J(\boldsymbol{W}_{enc}, \boldsymbol{W}_{dec}, \boldsymbol{b}_{enc}, \boldsymbol{b}_{enc}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}') + \lambda g(\boldsymbol{W}_{enc}, \boldsymbol{W}_{dec})$$

$$g(\boldsymbol{W}_{enc}, \boldsymbol{W}_{dec}) = 0.5(\|\boldsymbol{W}_{enc}\|_F^2 + \|\boldsymbol{W}_{dec}\|_F^2)$$

Vanilla Autoencoder

- Sample configuration in Keras® for MNIST:

```
input_img = keras.Input(shape=(784,))
encoded = layers.Dense(32, activation='relu')(input_img)
decoded = layers.Dense(784, activation='sigmoid')(encoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

- Original (top) and reconstructed (bottom) digits

Autoencoder for Dimension Reduction

- Autoencoder vs PCA:
- MNIST original data:

- Shallow AE (784-32–784)

- Deep AE (Model:784–128–64–32–64–128–784)

Autoencoder for Dimension Reduction

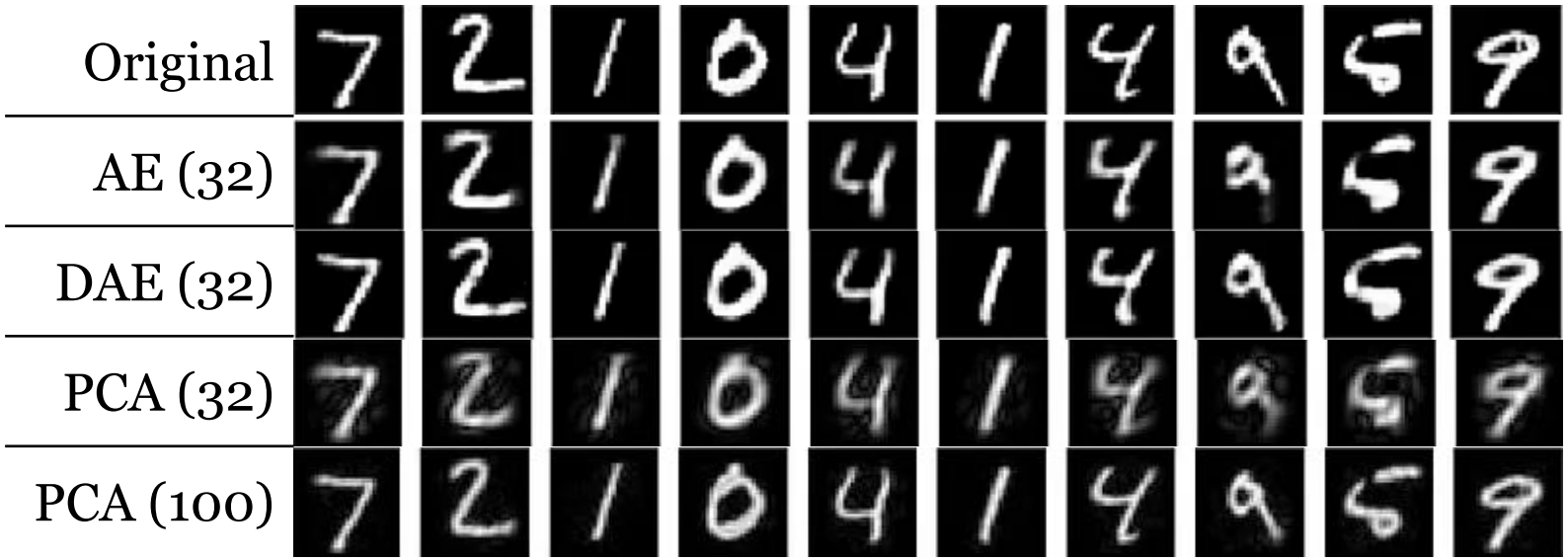
- Autoencoder vs PCA:
- MNIST original data:

- PCA (32):

- PCA (100):

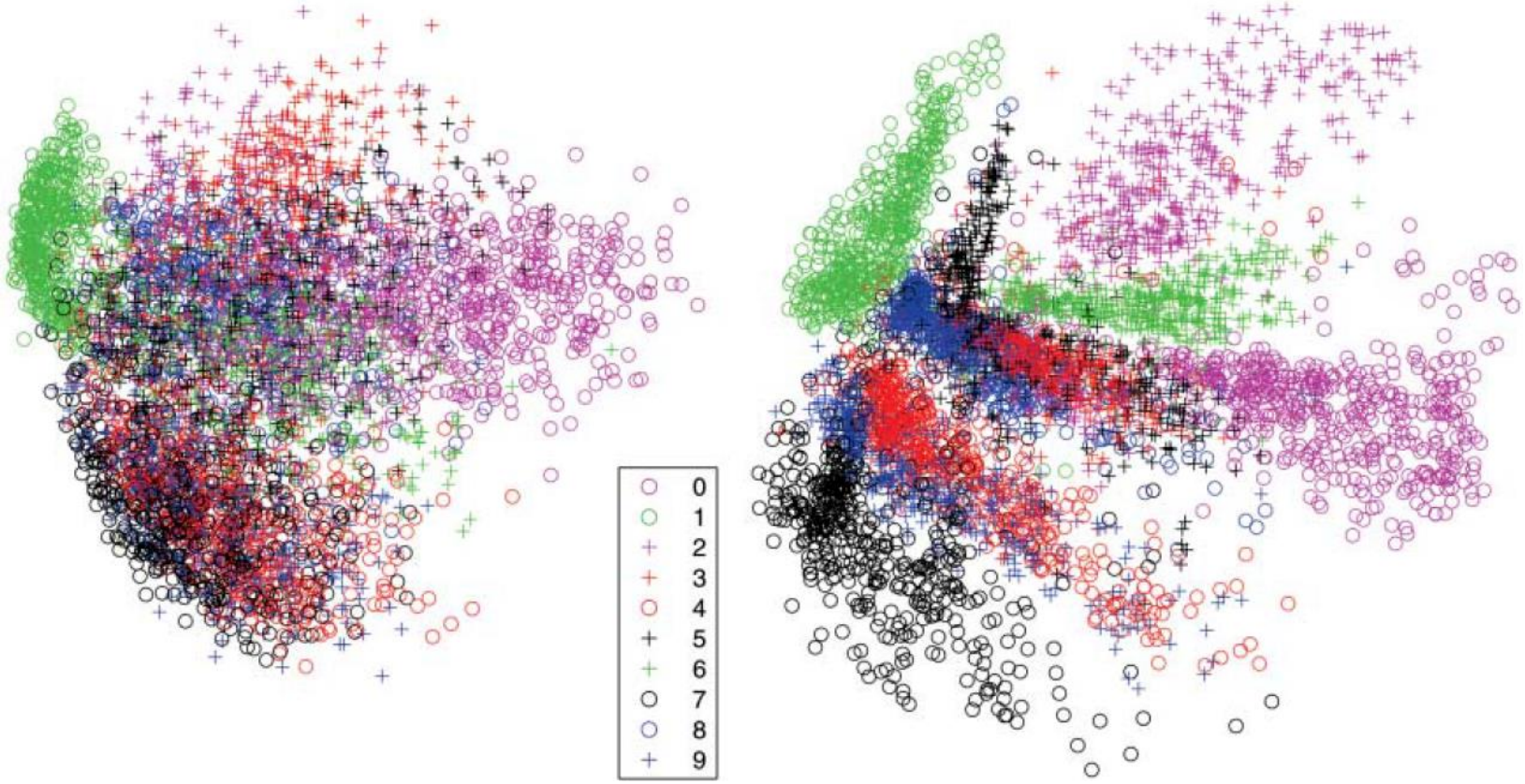
Autoencoder for Dimension Reduction

- Autoencoder vs PCA:



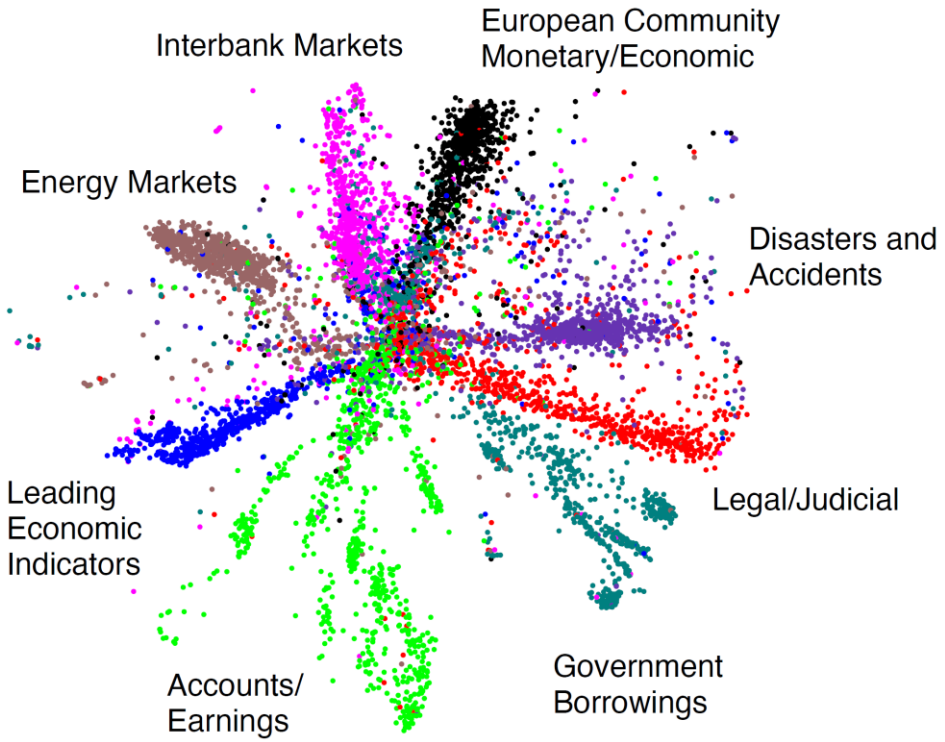
Autoencoder for Dimension Reduction

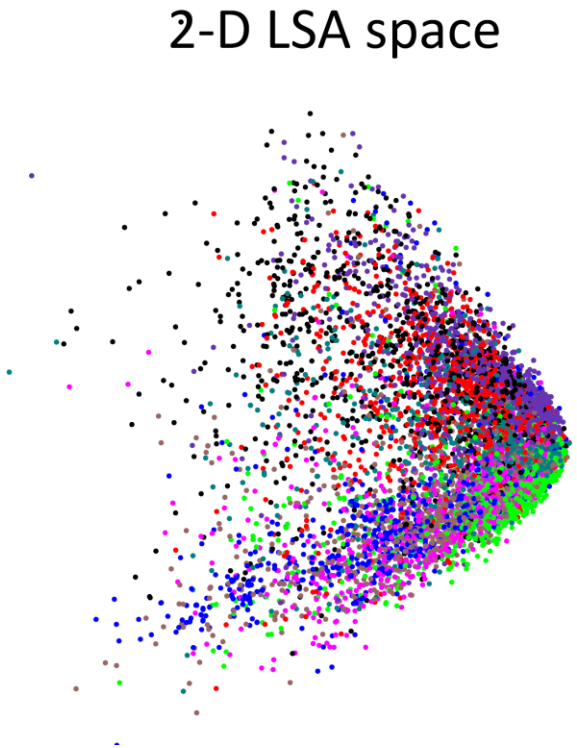
- Left (PC#1 and PC #2), Right: AE (784-1000-500-250-2)



Autoencoder for Dimension Reduction

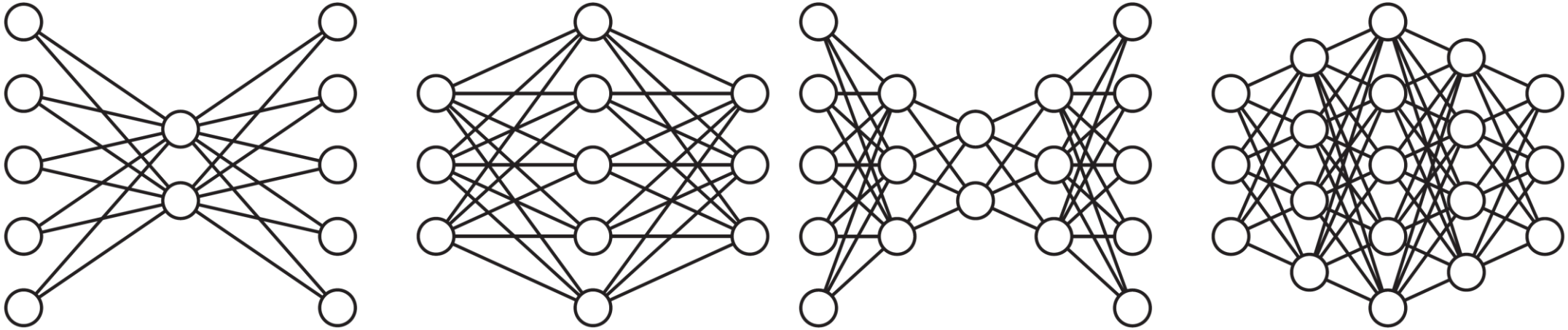
- Article coded by 2000D Bag-of-Word vectors (AE vs LSA)
- AE: 2000-500-250-125-2





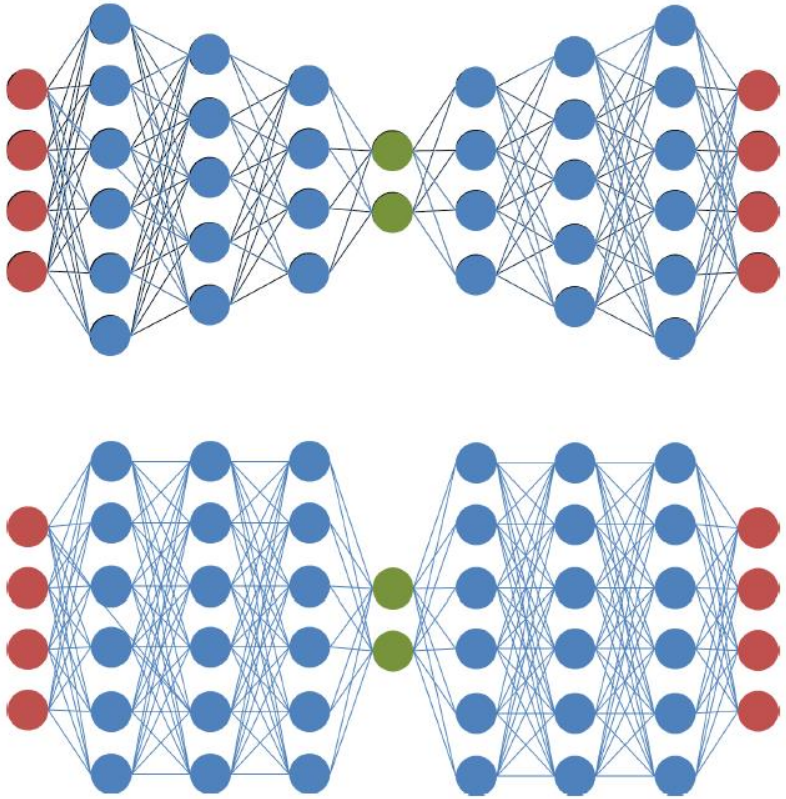
AE Architectures

- Shallow/Deep Undercomplete/Overcomplete AE



AE Architectures

- Deep Undercomplete:



Deep Autoencoder

- Sample configuration in Keras® for MNIST:

```
input_img = keras.Input(shape=(784,))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(32, activation='relu')(encoded)
decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded = layers.Dense(784, activation='sigmoid')(decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

Autoencoder for Dimension Reduction

- Autoencoder vs PCA:
- MNIST original data:

- Shallow AE (784-32–784)

- Deep AE (Model:784–128–64–32–64–128–784)

Autoencoder for Dimension Reduction

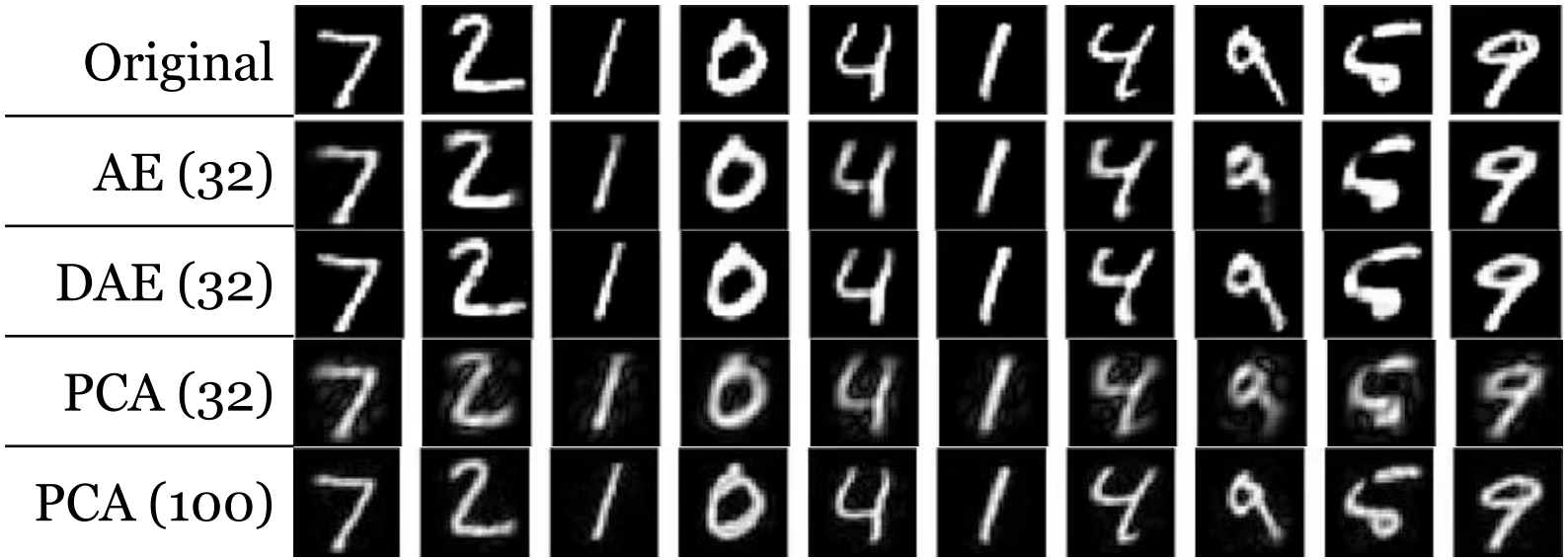
- Autoencoder vs PCA:
- MNIST original data:

- PCA (32):

- PCA (100):

Autoencoder for Dimension Reduction

- Autoencoder vs PCA:

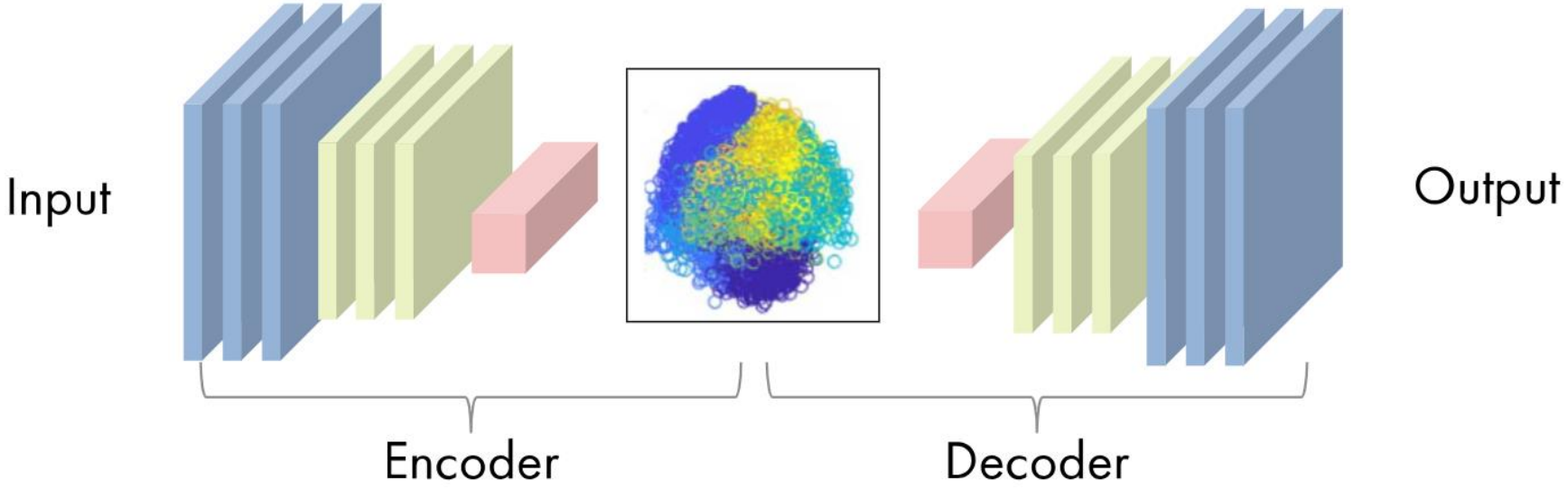


AE Taxonomy:

- Dimensionality:
  - Vanilla (discussed before)
  - Convolutional AE (CAE)
- Regularization:
  - Sparse AE
  - Contractive AE
- Noise:
  - Denoising
  - Robust

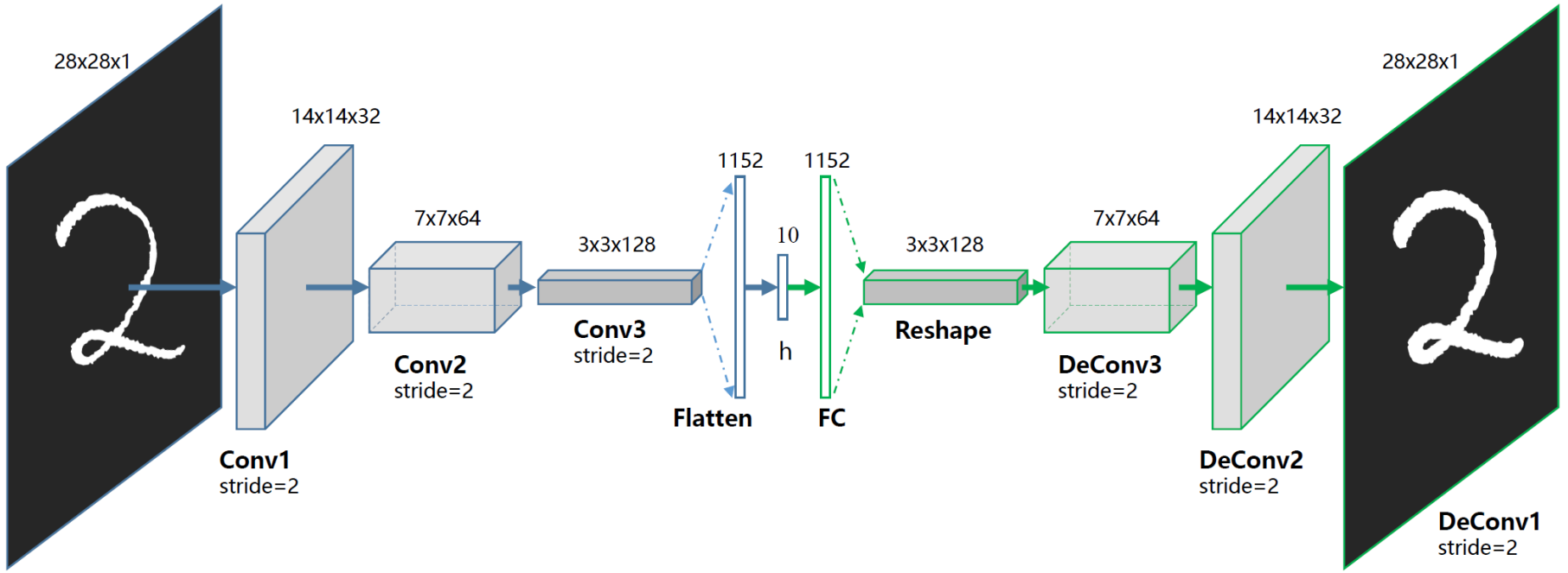
Deep Convolutional Autoencoder (CAE)

- Image/Signal friendly



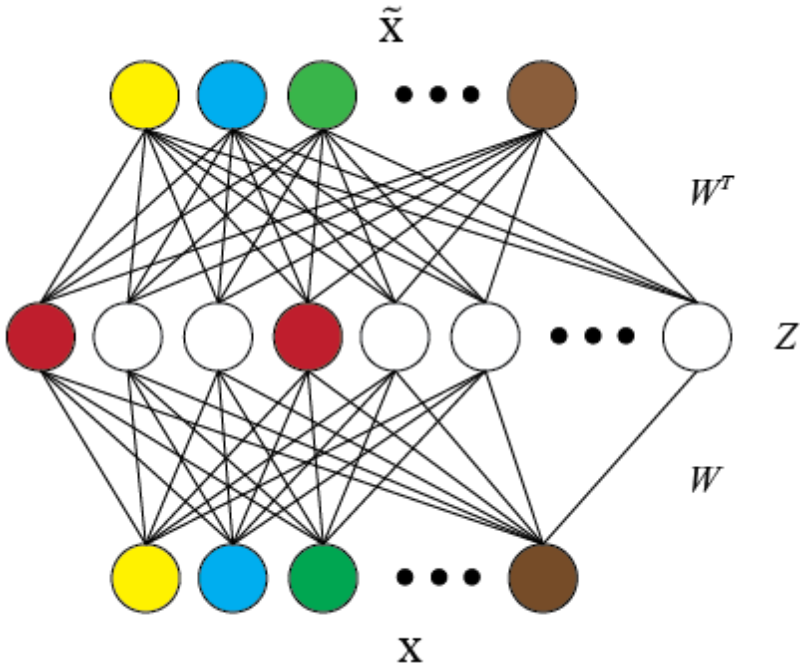
Deep Convolutional Autoencoder (CAE)

- Image/Signal friendly



Sparse Autoencoder (SAE)

- A sparse autoencoder (overcomplete) is simply an autoencoder whose training criterion involves a sparsity penalty $\Omega(\boldsymbol{h})$ on the code layer $\boldsymbol{h}$.



Sparse Autoencoder (SAE)

- Sparsity Loss (assume sigmoid in hidden layer):

$$\hat{\rho}_i = \frac{1}{|S|}\sum_{x \in S} h_i\,(x)$$

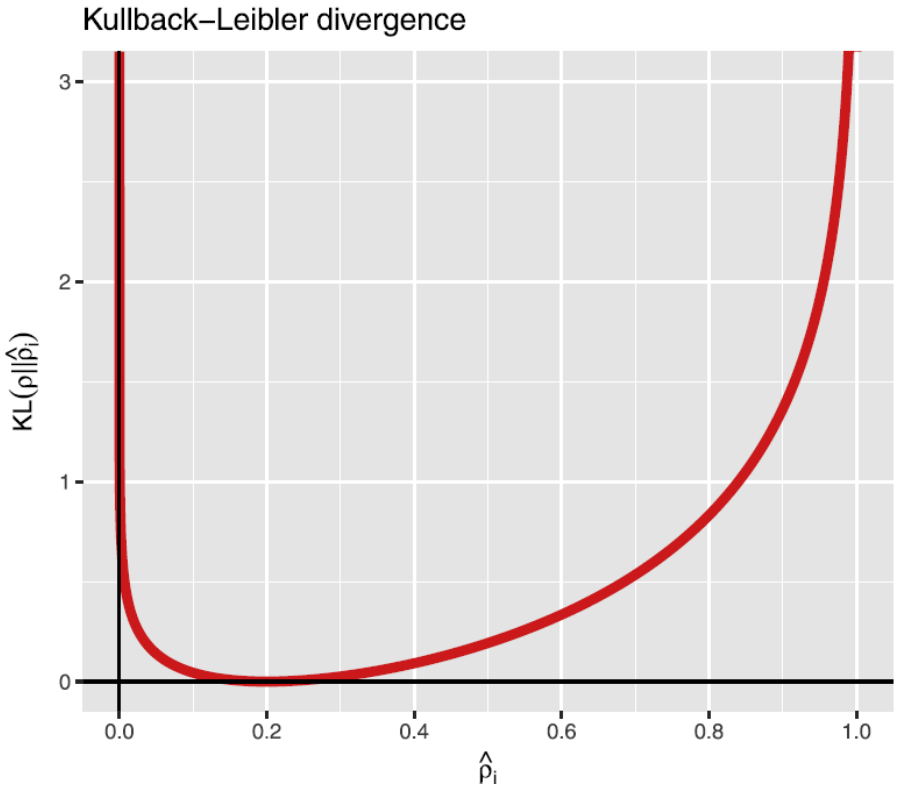
- where S is batch set, and i is neuron index, and $h_i(x)$ is activation value.
- Extra regularization term:

$$\mathrm{KL}(\rho \parallel \hat{\rho}_i) = \sum_{i=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_i} + (1-\rho)\log \frac{1-\rho}{1-\hat{\rho}_i}$$

- where k is latent space dimension, ρ is sparsity parameter ($\rho = 0.05$)

Sparse Autoencoder (SAE)

- Sparsity loss ($\rho = 0.2$)



K-sparse Autoencoder

- Training:
  - Perform the feedforward phase and compute $\boldsymbol{z} = f(W^T \boldsymbol{x} + \boldsymbol{b})$
  - Find the k largest activations of $\boldsymbol{z}$ and set the rest to zero ($\boldsymbol{z}^*$)
  - Compute the output and the error using the sparsified $\boldsymbol{z}^*$ (weight are tied)

$$\hat{\boldsymbol{x}} = g(W\boldsymbol{z}^* + \boldsymbol{b}'), \qquad E = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2$$

  - Backpropagate the error through the k largest activations and iterate.
- Sparse Encoding:
  - Compute the features $\boldsymbol{z} = f(W^T \boldsymbol{x} + \boldsymbol{b})$. Find its αk ($\alpha > 1$) largest activations and set the rest to zero.

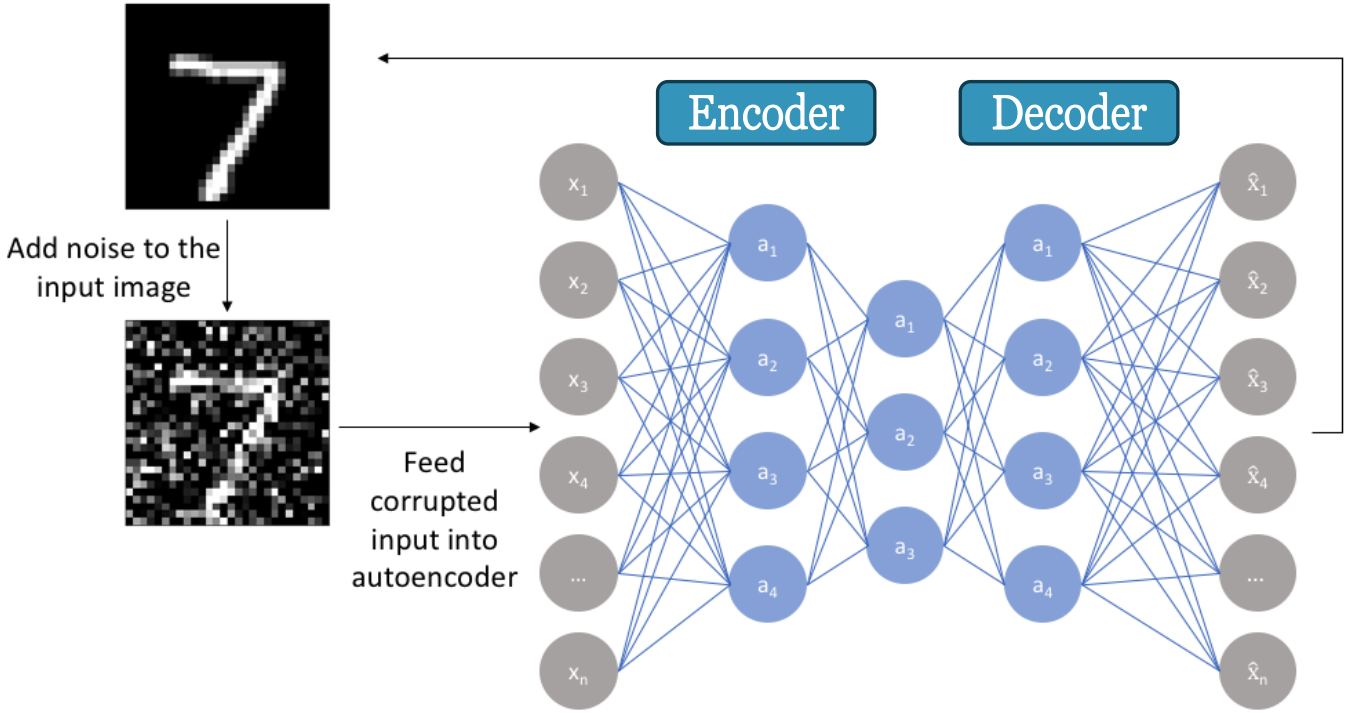
Contractive Autoencoder (CAE)

- Make the learned representation to be robust towards small changes around the training examples
- Sensitivity for small changes in the input can be measured as the Frobenius norm of the Jacobian matrix of the encoder, and use as *regularizer*

$$\|J_h(x)\|_F^2 = \sum_{j=1}^{d} \sum_{i=1}^{c} \left(\frac{\partial h_i(x)}{\partial x_j} \right)^2 , \Omega_{\text{CAE}}(W, b; S) = \sum_{x \in S} \|J_h(x)\|_F^2 .$$

Denoising Autoencoder (DAE)

- Learn robust feature to able reconstruct data from an input of corrupted data (input: noisy data, output: clean data)

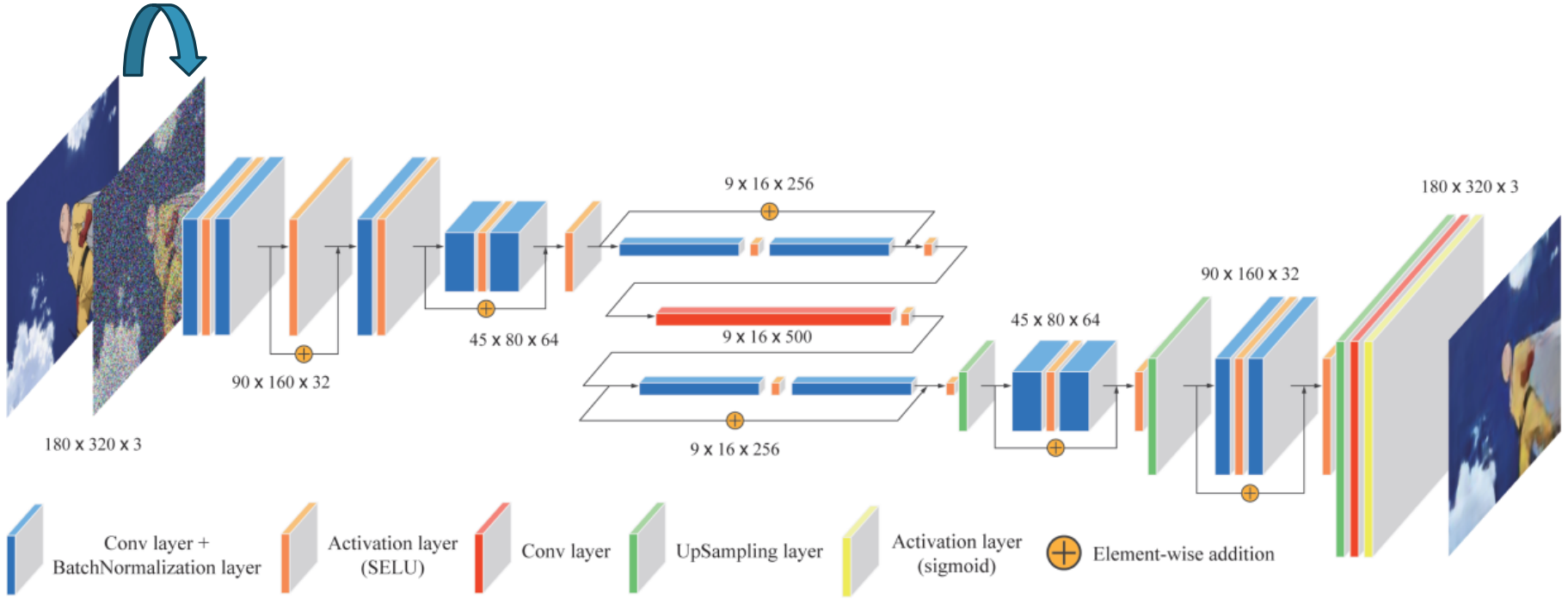


Denoising Autoencoder (DAE)

- How to corrupt input data:
  - Add white Gaussian noise to data $\sim N(0, \sigma^2 I)$
  - Masking noise: some elements of $\boldsymbol{x}$ (randomly) set to zero
  - Salt-and-pepper noise: some elements of $\boldsymbol{x}$ (randomly) set to min/Max value
  - Merge with dropout strategy

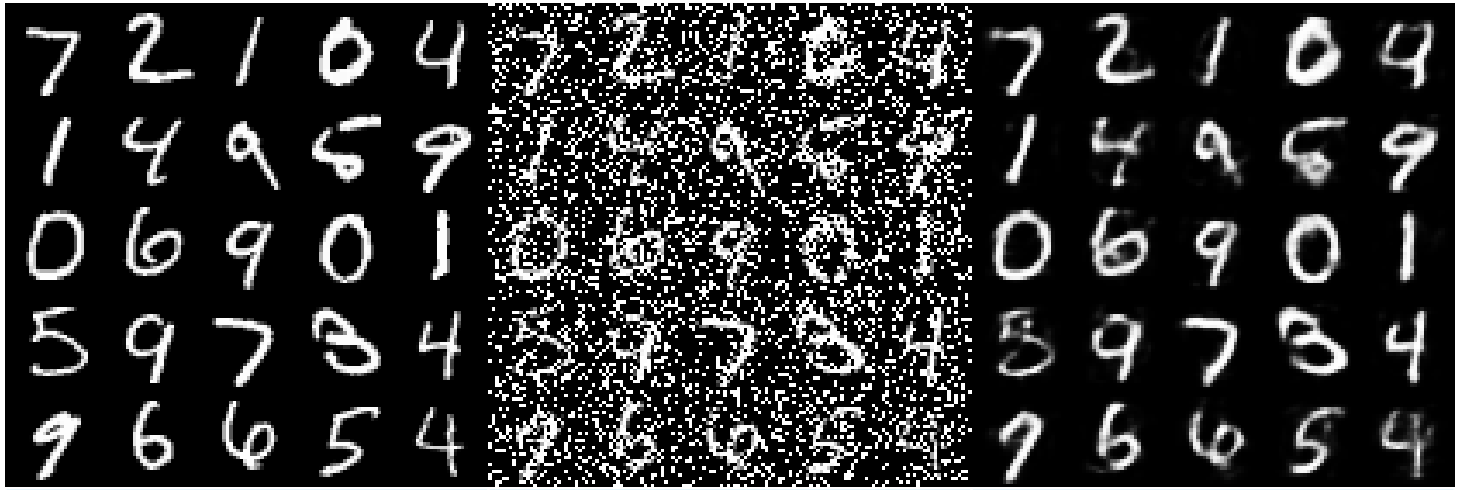
Denoising Autoencoder (DAE)

- Learn robust feature to able reconstruct data from an input of corrupted data (input: noisy data, output: clear data)



Denoising Autoencoder (DAE)

- Example:



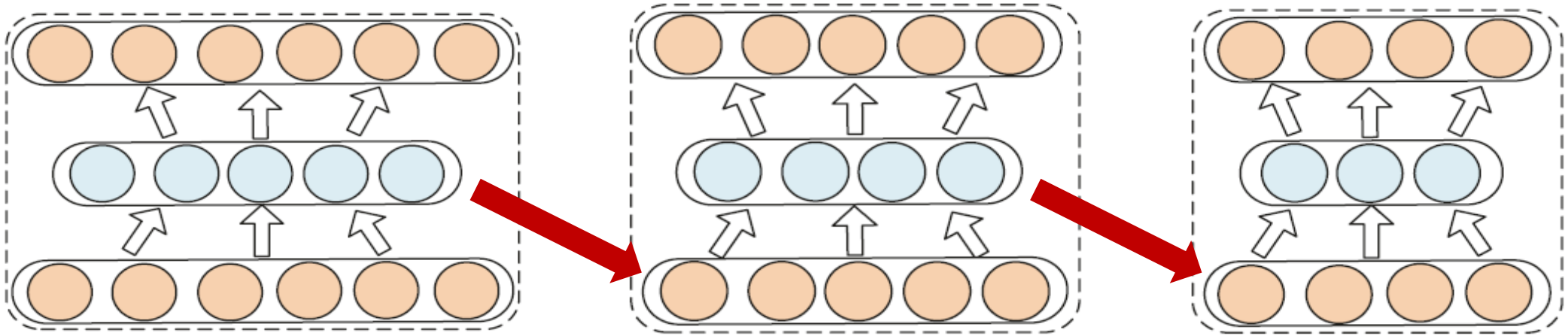
Stacked Autoencoder (SAE)

- Structure is Deep, but layer-by-layer learning!
- 1^{st} layer to n^{th} layer: simple (low level) to complex (high level) features
- Train the 1^{st} AE by input data and obtain the learned feature vector;
- The feature vector of the former layer is used as the input for the next layer, and this procedure is repeated until the training completes.
- After all the hidden layers are trained, backpropagation algorithm(BP) is used to minimize the cost function and update the weights with labeled training set to achieve fine tuning.

Stacked Autoencoder (SAE)

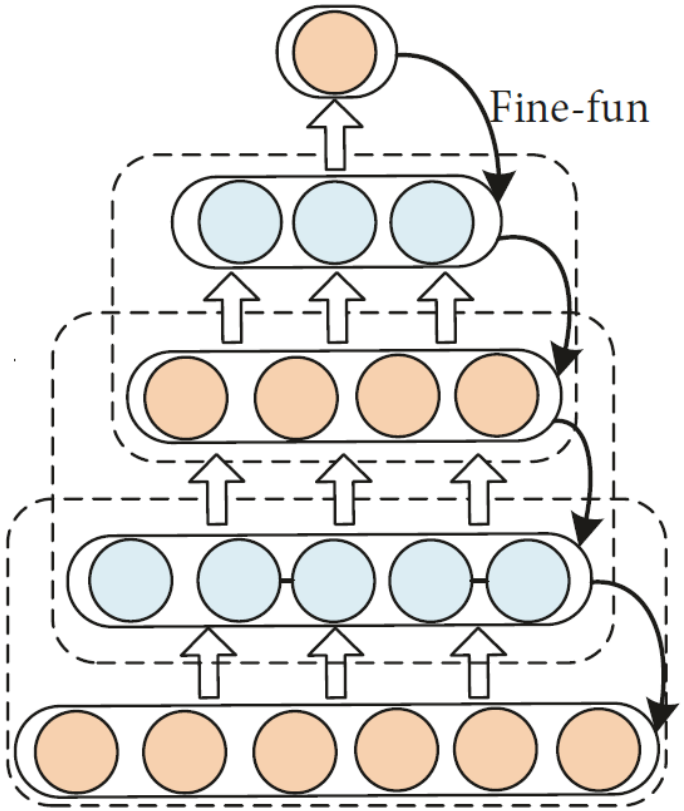
- Learning stages:

$$\{x \in \mathbb{R}^6\} \Longrightarrow \{z_1 \in \mathbb{R}^5\} \Longrightarrow \{z_2 \in \mathbb{R}^4\} \Longrightarrow \{z_3 \in \mathbb{R}^3\}$$



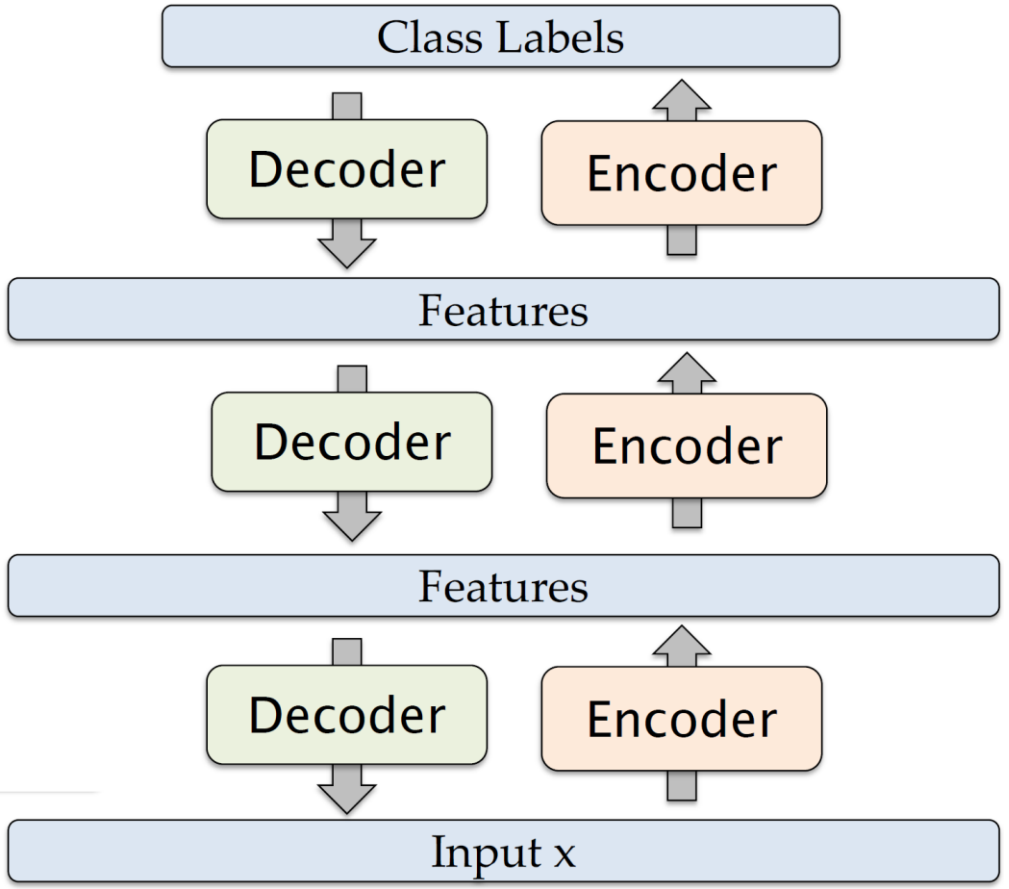
Stacked Autoencoder (SAE)

- Fine Tuning:



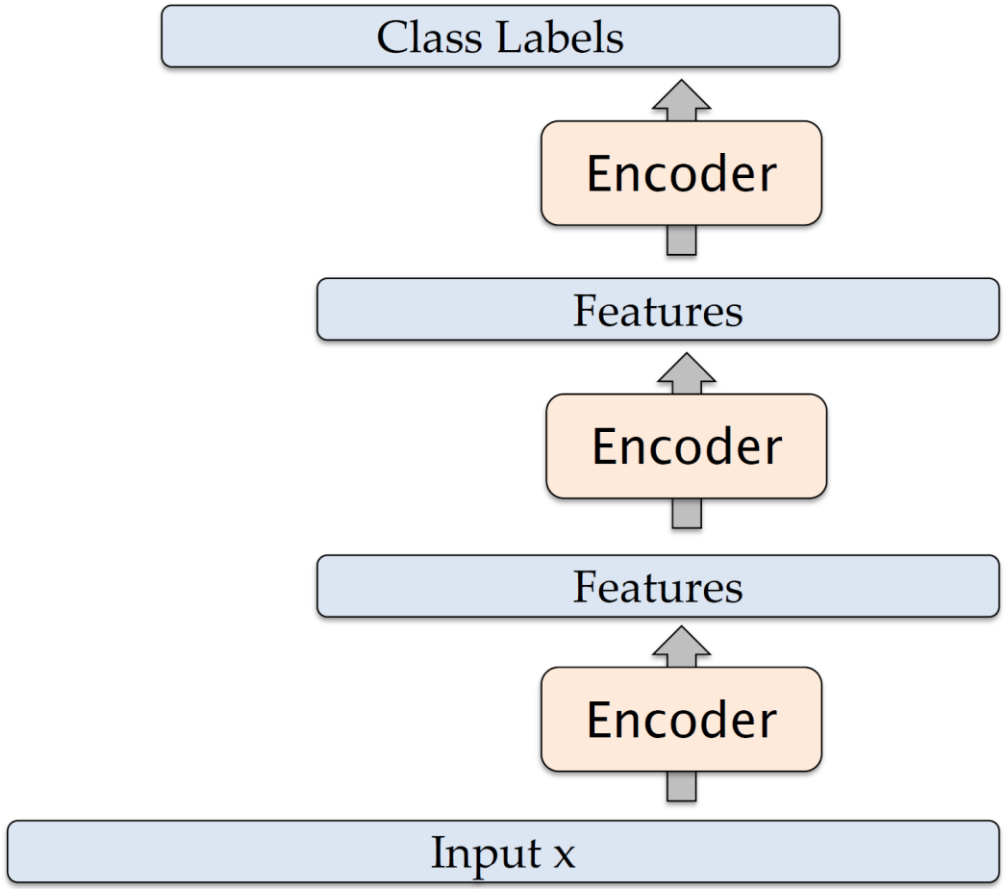
Stacked Autoencoder (SAE)

- Greed Layer-wise Learning



Stacked Autoencoder (SAE)

- Fine-Tuning for main task

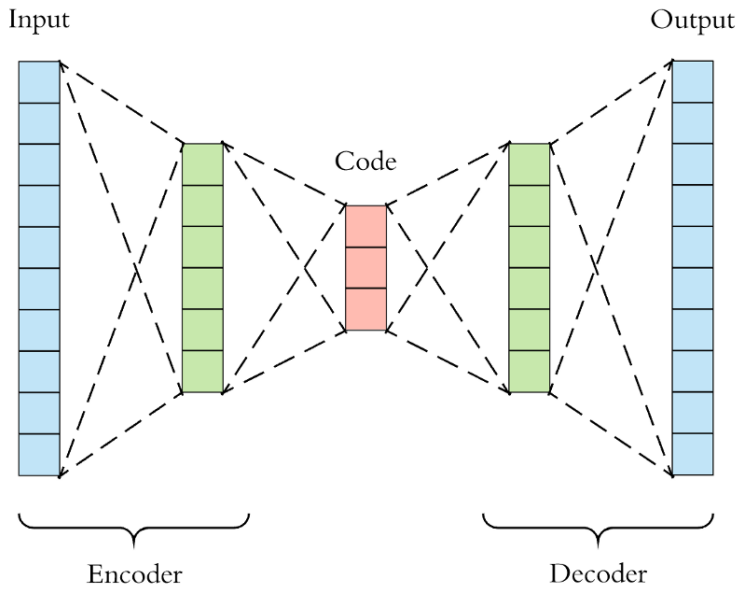


Stacked Autoencoder (SAE)

- Variants:
  - Stacked Denoising Autoencoder (SDA):
  - Stacked Convolutional Autoencoder (SCA)

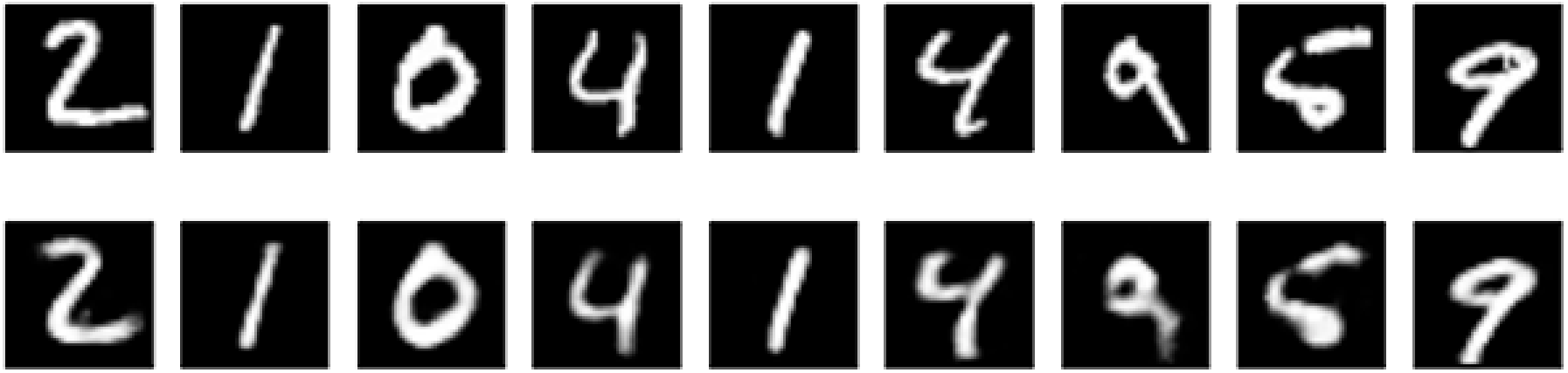
Examples and Applications

- Feature Reduction
  - Input: 28×28 (784D), hidden (100D), Code Size (32D):



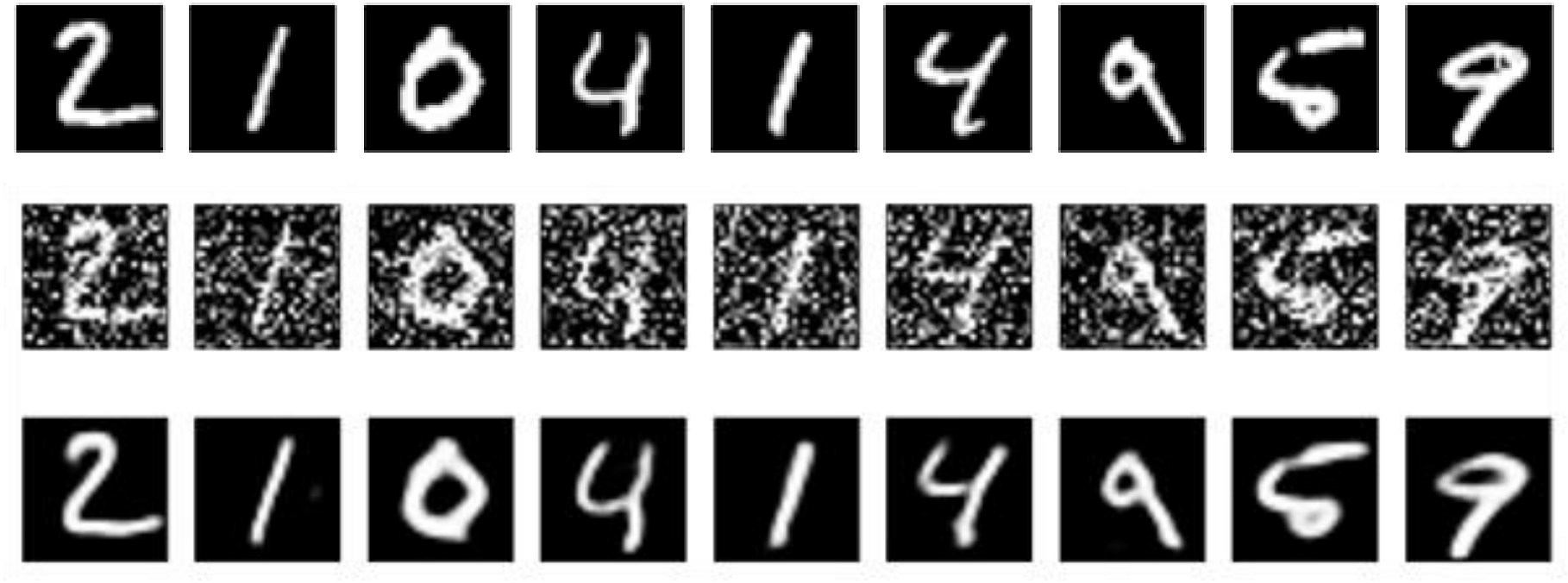
Examples and Applications

- Feature Reduction
  - Input: 28×28 (784D), hidden (784-128-64-32-64-128-784), Code Size (32D):



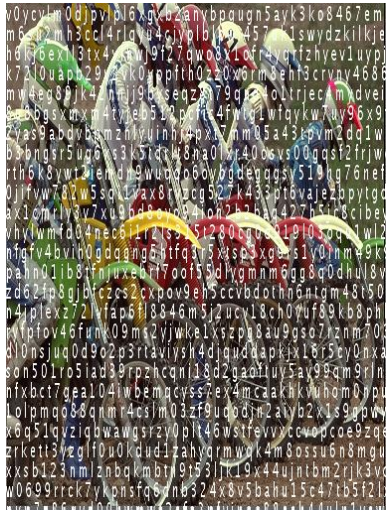
Examples and Applications

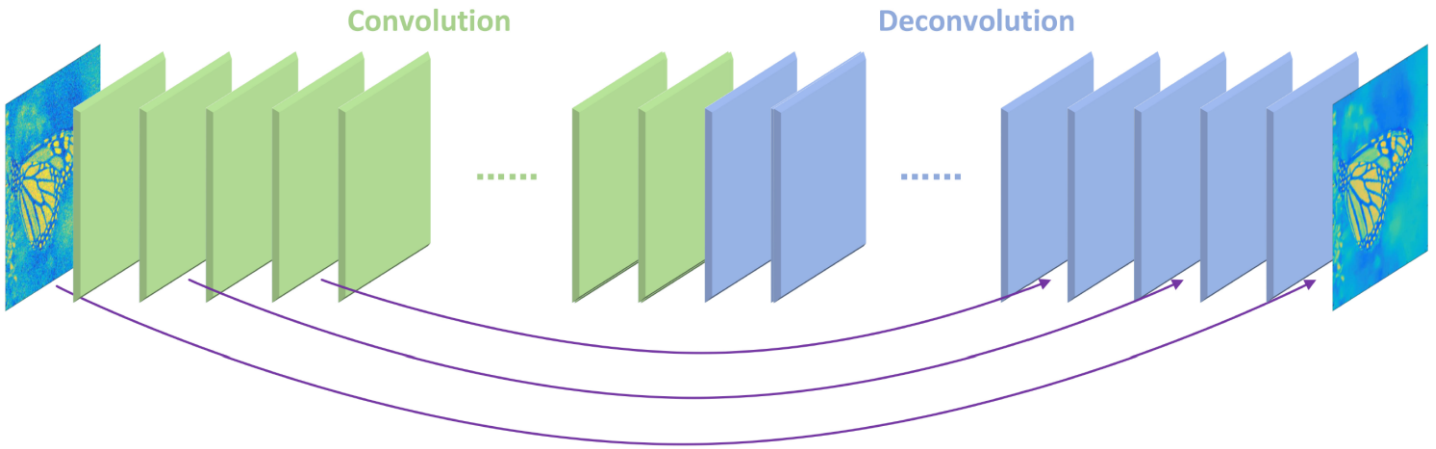
- Denoising AE
  - Same setting



Examples and Applications

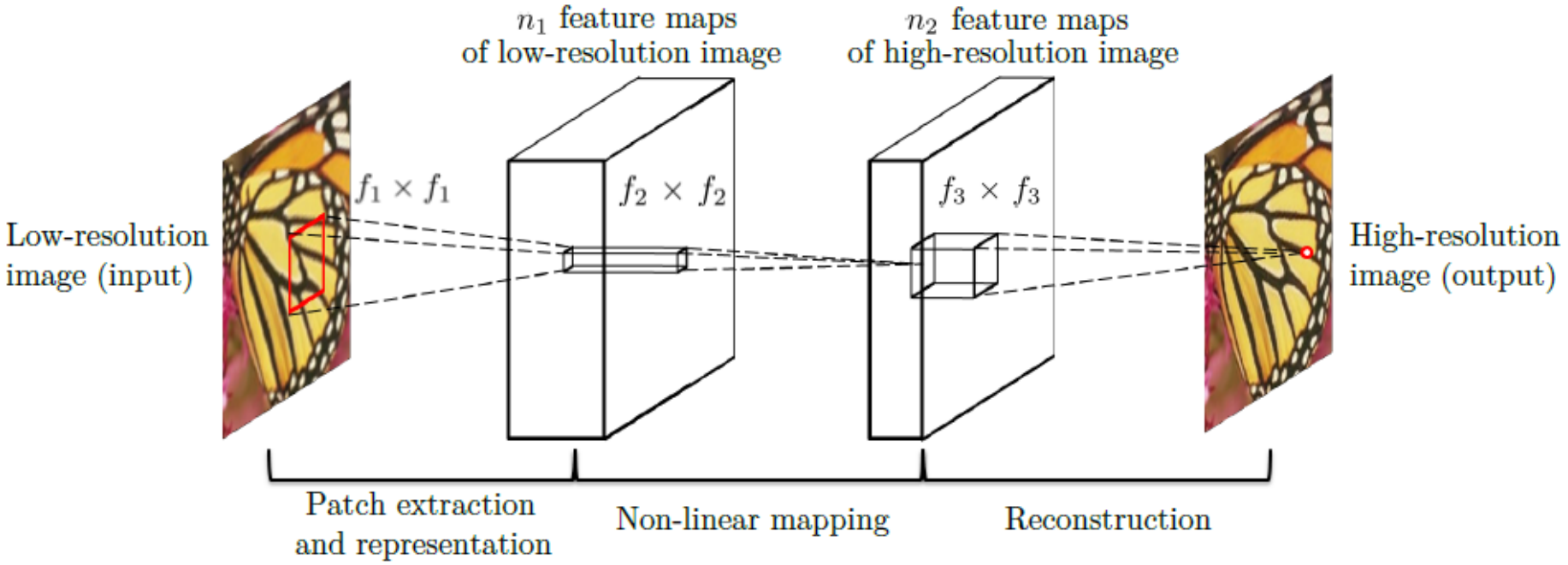
- Image reconstruction
  - Input: corrupted image, output; clean image





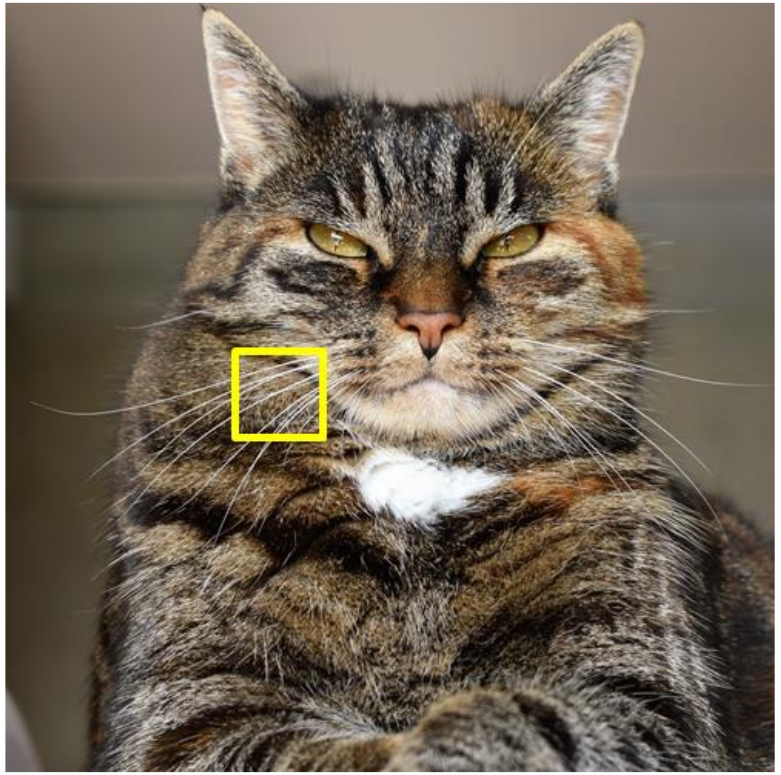
Examples and Applications

- Image Super Resolution



Examples and Applications

- Image Super Resolution



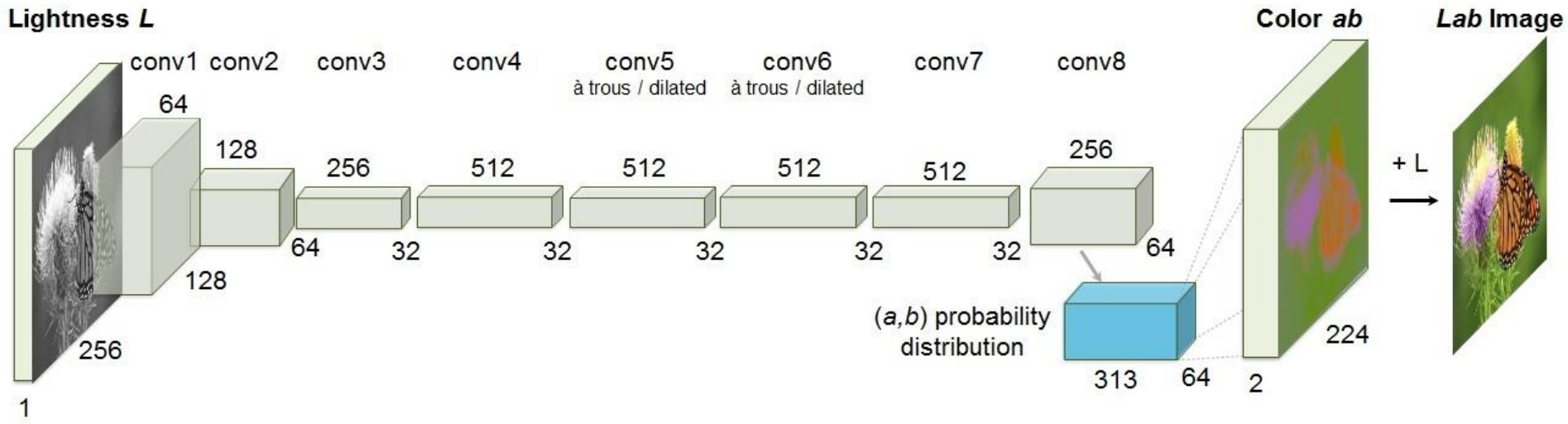
High Resolution

Bicubic

Conv-AE

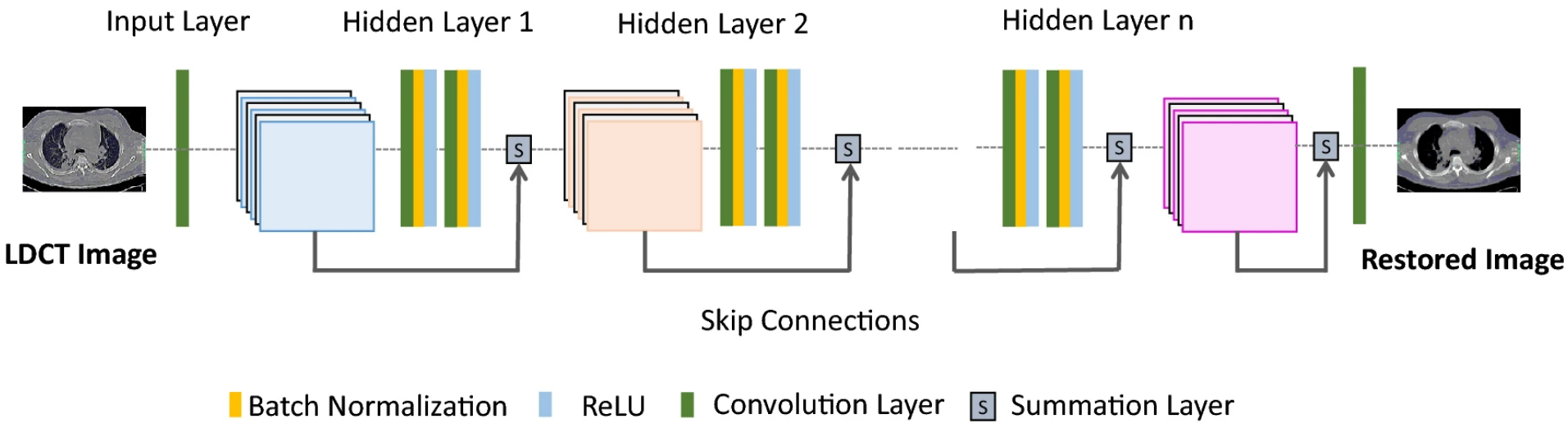
Examples and Applications

- Image Colorization



Examples and Applications

- Low Dose CT



Variational Autoencoder (VAE)

- Preliminary:
  - Discriminative Model
  - Generative Model
  - Generating Arbitrarily Distributed Random Variables
  - KL Distance and Maximum Likelihood

Discriminative Model

- Discriminative Model (X: input, Y: Target):
  - A Discriminative models, model the decision boundary between the classes:
    - Assume some functional form for $P(Y|X)$
    - Estimate parameters of $P(Y|X)$ directly from training data
  - Example:
    - Traditional neural networks
    - Nearest neighbor
    - Support Vector Machine
    - ….

Generative Model

- Generative Model:
  - A Generative Model explicitly models the actual distribution of each class:
    - Assume some functional form for $P(Y)$, $P(X|Y)$
    - Estimate parameters of $P(X|Y)$, $P(Y)$ directly from training data
    - Use Bayes rule to calculate $P(Y|X)$
  - Example:
    - Naïve Bayes
    - Hidden Markov Models (HMM)
    - ….

Generate Random Variables

- Generating Arbitrarily Distributed Random Variables:
  - Suppose we have r.v.'s: "$u \sim uni[0,1]$" and need x with special pdf, $p(x)$
  - Step #1: Calculate (CDF):

$$Prob\ (X<x) = \mathrm{P}(x) = \int_{-\infty}^{x} p(x)dx$$

  - Step #2:

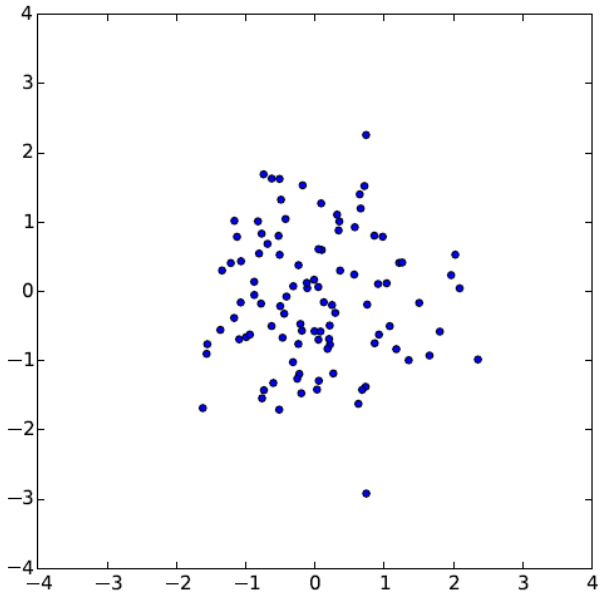
$$x = P^{-1}\ (u) \sim p(x)$$

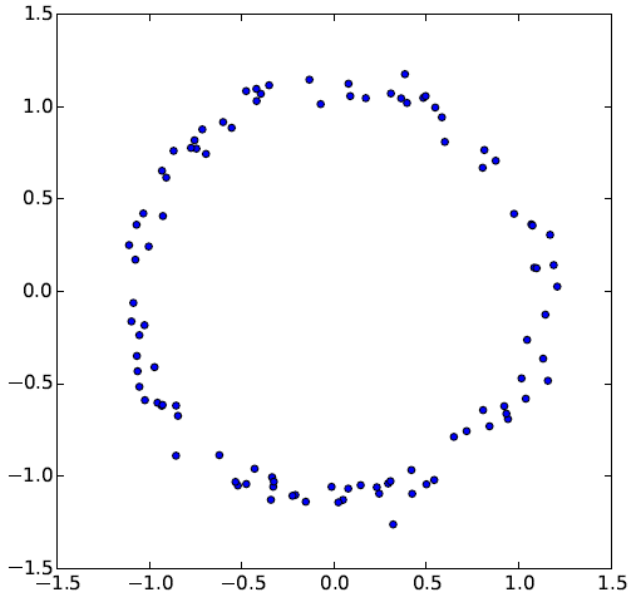
- How to travel from $p(x)$ to $g(x)$

$$G^{-1}(P(x))$$

Generate Random Variables

- Generating Arbitrarily Distributed Random Variables:
  - z: 2D Gaussian r.v. → $\frac{z}{10} + \frac{z}{\|z\|}$ follow $circ(z)$





Kullback–Leibler (KL) Distance/Divergence

- Measure of distance (Divergence) between two pdfs
- Continues random variables

$$D_{KL}(P\|Q) = \int_{-\infty}^{\infty} p(x) log\left(\frac{p(x)}{q(x)}\right) dx$$

- Discrete random variables:

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

- Properties:
  - Non-Negative
  - Non-Symmetric

KL Distance and ML

- Minimizing $D_{KL}[P(x|\theta^*)\|P(x|\theta)]$is equivalent to ML ($*$: true pdf)

$$D_{KL}[P(x|\theta^*)\|P(x|\theta)] = \mathbb{E}_{x\sim P(x|\theta^*)}\left[log\frac{P(x|\theta^*)}{P(x|\theta)}\right]$$

$$= \mathbb{E}_{x\sim P(x|\theta^*)}[logP(x|\theta^*) - logP(x|\theta)]$$

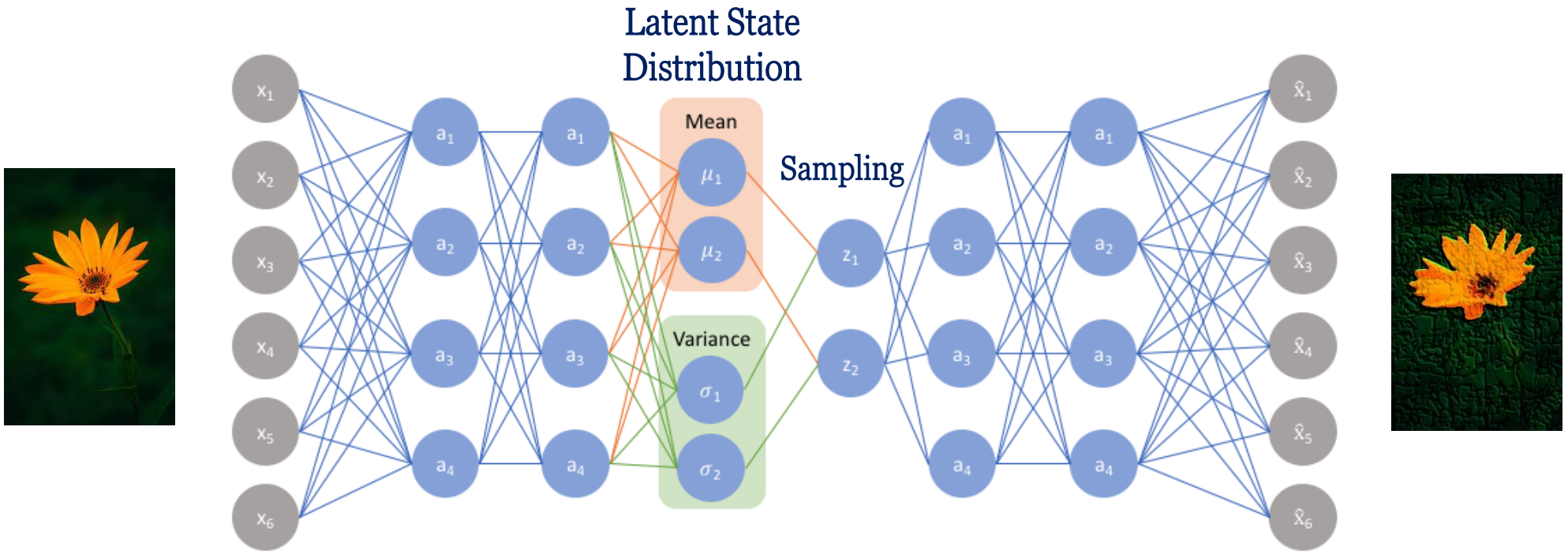
$$= \mathbb{E}_{x\sim P(x|\theta^*)}[logP(x|\theta^*)] - \mathbb{E}_{x\sim P(x|\theta^*)}[logP(x|\theta)]$$

- Law of Large Numbers (Left-Side is Negative Likelihood)!

$$-\mathbb{E}_{x\sim P(x|\theta^*)}[logP(x|\theta)] = -\frac{1}{N}\sum_{i}^{N} log\, P(x_i|\theta)$$

Variational Autoencoder (VAE)

- A Generative model to estimate $p(x)$
  - With sampling (sample generation) we may produce x (image, for example)
  - Novel image synthesis from random samples



VAE Mathematics

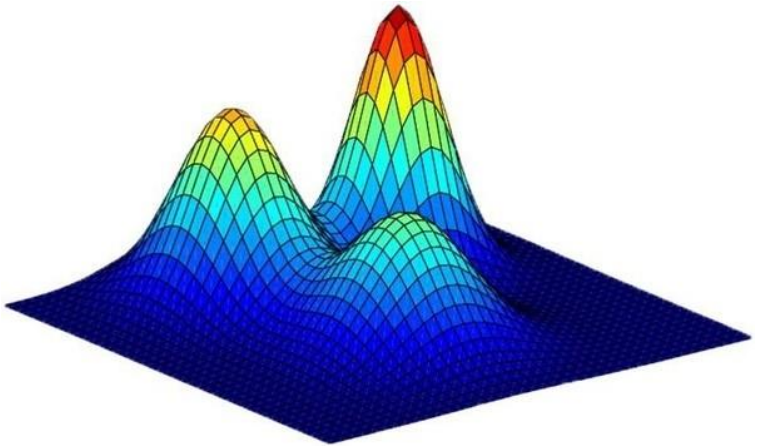
- VAE key point: latent variable!
- Example:
  - Class index in Mixture Model!

What is Mixture Model

- Input: unlabeled data: $\{\boldsymbol{x}_i\}_{i=1}^{N}$

$$p(\boldsymbol{x}) = \sum_{k=1}^{m} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \sum_{k=1}^{m} \pi_k = 1$$

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{m} \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$



Definition

- X: Data that we wish to model,
- z: latent variable,
- $P(X)$: pdf of the data,
- $P(z)$: pdf of latent variable,
- $P(X|z)$: Distribution of generated data given latent variable.

VAE Mathematics

- Objective:

$$P(X) = \int P(X|z)P(z)dz$$

- VAE idea: infer $P(z)$ using $P(z|X)$: Estimate latent variable likely using our data

- What about: $P(z|X)$:
  - A well know method: "Variational Inference", VI
  - Reformulate problem to an optimization problem:
    - Simple/Parametric modelling of $P(z|X)$ and minimize distance between model and data.

VAE Mathematics

- Let infer $P(z|X)$ using $Q(z|X)$ via KL distance:

$$D_{KL}[Q(z|X)\|P(z|X)] = \sum_z Q(z|X)\, log\, \frac{Q(z|X)}{P(z|X)}$$

$$= \mathbb{E}_{z\sim Q(z|X)}\left[log\, \frac{Q(z|X)}{P(z|X)}\right] = \mathbb{E}_z\left[log\, \frac{Q(z|X)}{P(z|X)}\right]$$

$$= \mathbb{E}_z[logQ(z|X) - logP(z|X)]$$

- Let use $P(X)$, $P(X|z)$, and $P(z)$, via Bayes rule.

VAE Mathematics

- Let use $P(X)$, $P(X|z)$, and $P(z)$, via Bayes rule.
- $D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logQ(z|X) - logP(z|X)]$
- $D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z\left[logQ(z|X) - log\frac{P(X|z)P(z)}{P(X)}\right]$
- $D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z\left[logQ(z|X) - \left(logP(X|z) + logP(z) - logP(X)\right)\right]$
- $D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logQ(z|X) - logP(X|z) - logP(z) + logP(X)]$
- Expectation is over z
- $D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logQ(z|X) - logP(X|z) - logP(z)] + logP(X)$

VAE Mathematics

- Expectation is over z

$$D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logQ(z|X) - logP(X|z) - logP(z)] + logP(X)$$

- Move $logP(X)$ to left side:

$$D_{KL}[Q(z|X)\|P(z|X)] - logP(X) = \mathbb{E}_z[logQ(z|X) - logP(X|z) - logP(z)]$$

- In next step we will transform right-hand to get another KL distance,
- $logP(X) - D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z\big[logP(X|z) - \big(logQ(z|X) - logP(z)\big)\big]$
- $logP(X) - D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logP(X|z)] - \mathbb{E}_z[logQ(z|X) - logP(z)]$
- $logP(X) - D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logP(X|z)] - D_{KL}[Q(z|X)\|P(z)]$

VAE Mathematics

- This is defined as **Ev**idence **L**ower **BO**und (ELBO):

$$logP(X) - D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logP(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$

$$logP(X) \geq \mathbb{E}_z[logP(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$

$$logP(X) \geq \mathbb{E}_z[logP(X|z)] - \mathbb{E}_z\left[log\frac{Q(z|X)}{P(z)}\right]$$

$$logP(X) \geq \mathbb{E}_z\left[log\frac{P(X|z)P(z)}{Q(z|X)}\right]$$

$$logP(X) \geq \mathbb{E}_z\left[log\frac{P(X,z)}{Q(z|X)}\right]$$

VAE Mathematics

- VAE objective: ELBO Maximization (Note that $logP(X)$ is constant)

$$logP_\theta(X) - D_{KL}\left[Q_\varphi(z|X)\middle\|P_\theta(z|X)\right] = \mathbb{E}_z[logP_\theta(X|z)] - D_{KL}\left[Q_\varphi(z|X)\middle\|P_\theta(z)\right]$$

- $Q_\varphi(z|X)$: Project data X into latent variable ⇝ {*Encoder Network*}
- z : Latent variable ⇝ {*Generated Code*}
- $P_\theta(X|z)$: Generate data given latent variable ⇝ {*Decoder Network*}
- φ and θ are network parameters
- In fact: *VAE tries to find the lower bound of* $logP_\theta(X)$ (except some error, left-hand KL distance), $D_{KL}[Q(z|X)\|P(z|X)]$

VAE Mathematics

- VAE objective:

$$logP_\theta(X) - D_{KL}\left[Q_\varphi(z|X)\middle\|P_\theta(z|X)\right] = \mathbb{E}_z[logP_\theta(X|z)] - D_{KL}\left[Q_\varphi(z|X)\middle\|P_\theta(z)\right]$$

- ELBO Maximization ~ Log-Likelihood P(X) under decoding error
- Objective: *Maximizing* $logP_\theta(X|z)$ and *minimizing* KL distance between our simple distribution $Q_\varphi(z|X)$ and the true latent distribution $P(z)$.
- Maximizing $logP_\theta(X|z)$: Any discriminative supervised-parametric model (SVM/MLP/...), Any classifier with z as input and X as output, and a proper loss.

VAE Mathematics

- VAE objective (omit φ and θ for simplification):

$$logP(X) - D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logP(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$

- minimizing $D_{KL}[Q(z|X)\|P(z)]$:
- First computation, $D_{KL}[Q(z|X)\|P(z)]$:
- $P(z)$: Latent variable pdf, we will sample it to generate X: $\mathcal{N}(0, \boldsymbol{I})$
- Easy to make $Q(z|X)$ as-close-as possible (KL minimizing) via network.

VAE Mathematics

- $Q(z|X)$ computation: Another simplification:

$$\mathcal{N}\left(\mu(X), \Sigma(X)\right), \text{ and } \textit{diagonal}, \Sigma(X) = diag\left(\sigma_k^2(X)\right)$$

  - why Gaussian assumption: Easy to compute KL between two Gaussian:

$$\mathcal{N}(0, \boldsymbol{I}) \text{ and } \mathcal{N}\left(\mu(X), \Sigma(X)\right)$$

$$D_{KL}(\mathcal{N}_1 \| \mathcal{N}_2) = \frac{1}{2}\left(tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1) - d + log\frac{|\Sigma_2|}{|\Sigma_1|}\right)$$

  - where d is data dimension.

VAE Mathematics

- In VAE $z \sim \mathcal{N}(0, \boldsymbol{I})$

$$logP(X) - D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_z[logP(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$

$$D_{KL}\left(\mathcal{N}\left(\mu(X), \Sigma(X)\right)\middle\|\mathcal{N}(0, \boldsymbol{I})\right) = \frac{1}{2}\left(tr\left(\Sigma(X)\right) + \mu(X)^T\mu(X) - d - log|\Sigma(X)|\right)$$

- Diagonal assumptions:
  - $tr\left(\Sigma(X)\right) = \sum_{k=1}^{d} \Sigma_{kk}(X) = \sum_{k=1}^{d} \sigma_k^2(X)$
  - $\mu(X)^T\mu(X) = \sum_{k=1}^{d} \mu_k^2(X)$
  - $d = \sum_{k=1}^{d} 1$
  - $log|\Sigma(X)| = \sum_{k=1}^{d} log\sigma_k^2(X)$

VAE Mathematics

- Substituations:

$$D_{KL}\left(\mathcal{N}\left(\mu(X),\Sigma(X)\right)\|\mathcal{N}(0,\boldsymbol{I})\right)=\frac{1}{2}\left(\sum_{k=1}^{d}\sigma_k^2(X)+\sum_{k=1}^{d}\mu_k^2(X)-\sum_{k=1}^{d}1-\sum_{k=1}^{d}log\sigma_k^2(X)\right)$$

$$D_{KL}\left(\mathcal{N}\left(\mu(X),\Sigma(X)\right)\|\mathcal{N}(0,\boldsymbol{I})\right)=\frac{1}{2}\sum_{k=1}^{d}\left(\sigma_k^2(X)+\mu_k^2(X)-1-log\sigma_k^2(X)\right)$$

- A numerical trick: Model $\sigma_k^2(X)$ as $log\sigma_k^2(X)$, numerically stable to take *exponent* compared to *log*

$$D_{KL}\left(\mathcal{N}\left(\mu(X),\Sigma(X)\right)\|\mathcal{N}(0,\boldsymbol{I})\right)=\frac{1}{2}\sum_{k=1}^{d}\left(exp\left(\sigma_k^2(X)\right)+\mu_k^2(X)-1-\sigma_k^2(X)\right)$$

VAE Mathematics

- Total Loss

$$Loss = logP(X) - D_{KL}[Q(z|X)\|P(z|X)] = \underbrace{\mathbb{E}_z[logP(X|z)]}_{Maximize} - \underbrace{D_{KL}[Q(z|X)\|P(z)]}_{minimize}$$

$$Loss = \mathbb{E}_z[logP(X|z)] - \frac{1}{2}\sum_{k=1}^{d}\left(exp\left(\Sigma_{kk}(X)\right) + \mu_k^2(X) + 1 - \Sigma_{kk}(X)\right)$$

$$Loss_{VAE} = \underbrace{-\mathbb{E}_{z\sim Q_\varphi(z|X)} logP_\theta(X|z)}_{minimize} + \underbrace{D_{KL}\left[Q_\varphi(z|X)\middle\|P_\theta(z)\right]}_{minimize}$$

VAE Mathematics

- Practical Loss:

$$Loss_{VAE} = \mathcal{L}\left(X, \hat{X}\right) + D_{KL}[Q(z|X) \| P(z)]$$

$$Loss_{VAE} = -\mathbb{E}_{z\sim Q_\varphi(z|X)} log P_\theta(X|z) + D_{KL}\left[Q_\varphi(z|X) \middle\| P_\theta(z)\right]$$

VAE Mathematics

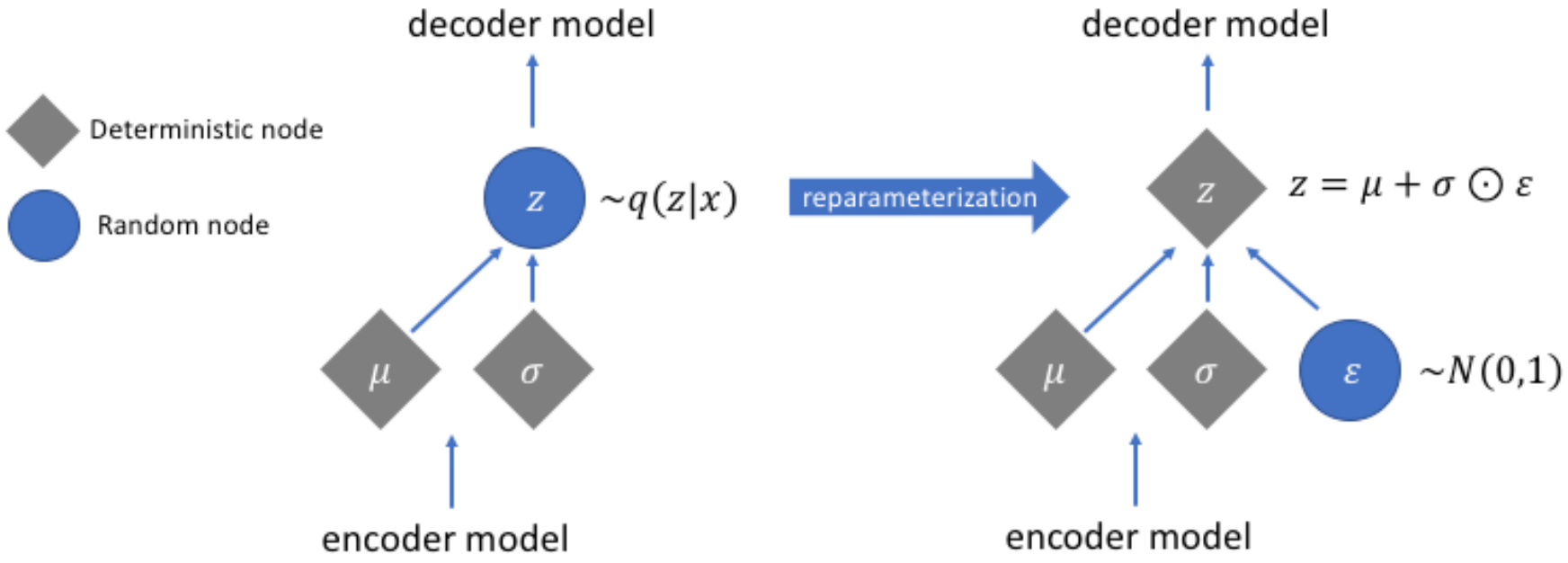
- How to get (sample) z (a r.v.) from encoder output?
  - A Naïve solution: sample $\mathcal{N}(\mu(X), \Sigma(X))$ is easy.
  - But sampling is not easy to handle in training phase with SGD family
  - In other word sampling is not differentiable
- Reparameterization Trick (*makes the network differentiable*):
  - Generate it from standard $\mathcal{N}(0, \mathbf{I})$:

$$z = \mu(X) + \Sigma^{0.5}(X)\boldsymbol{\varepsilon} = \mu(X) + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}$$

- if $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I})$ → $z \sim \mathcal{N}(\mu(X), \Sigma(X))$

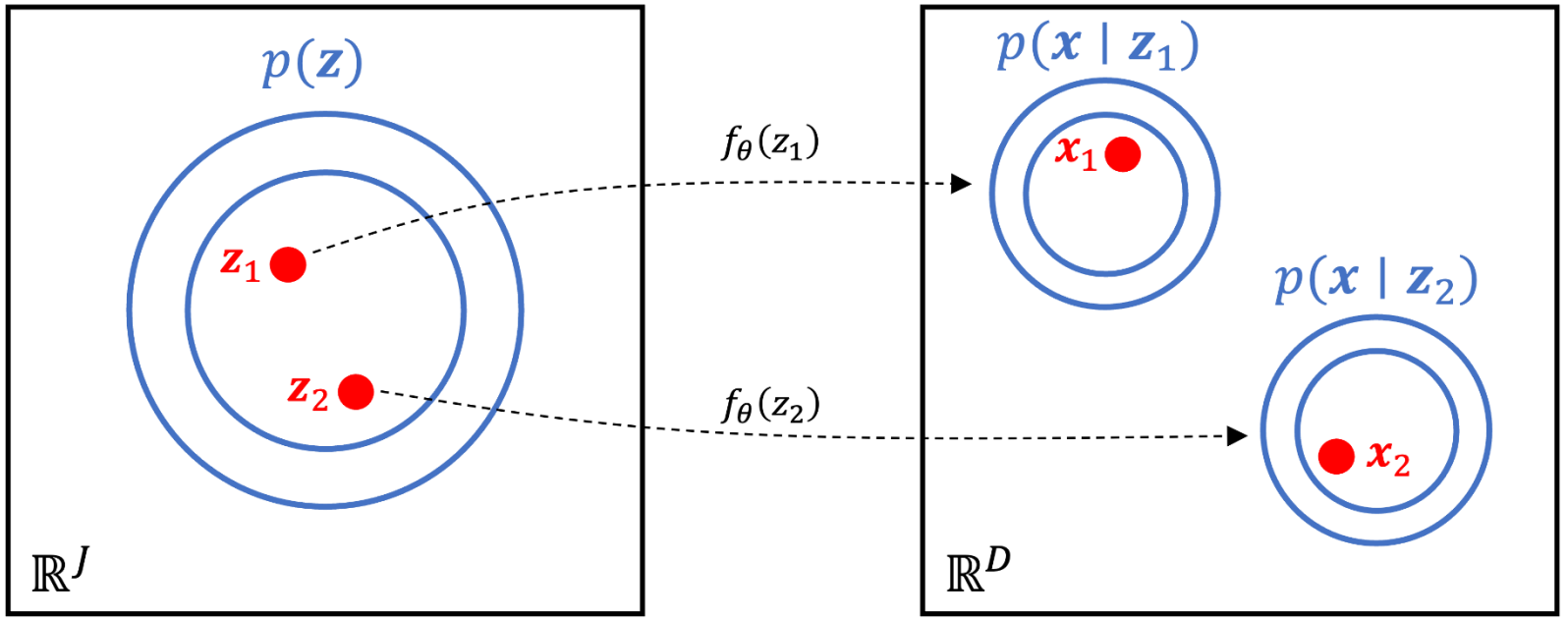
VAE Mathematics

- Reparameterization Trick (*makes the network differentiable*):



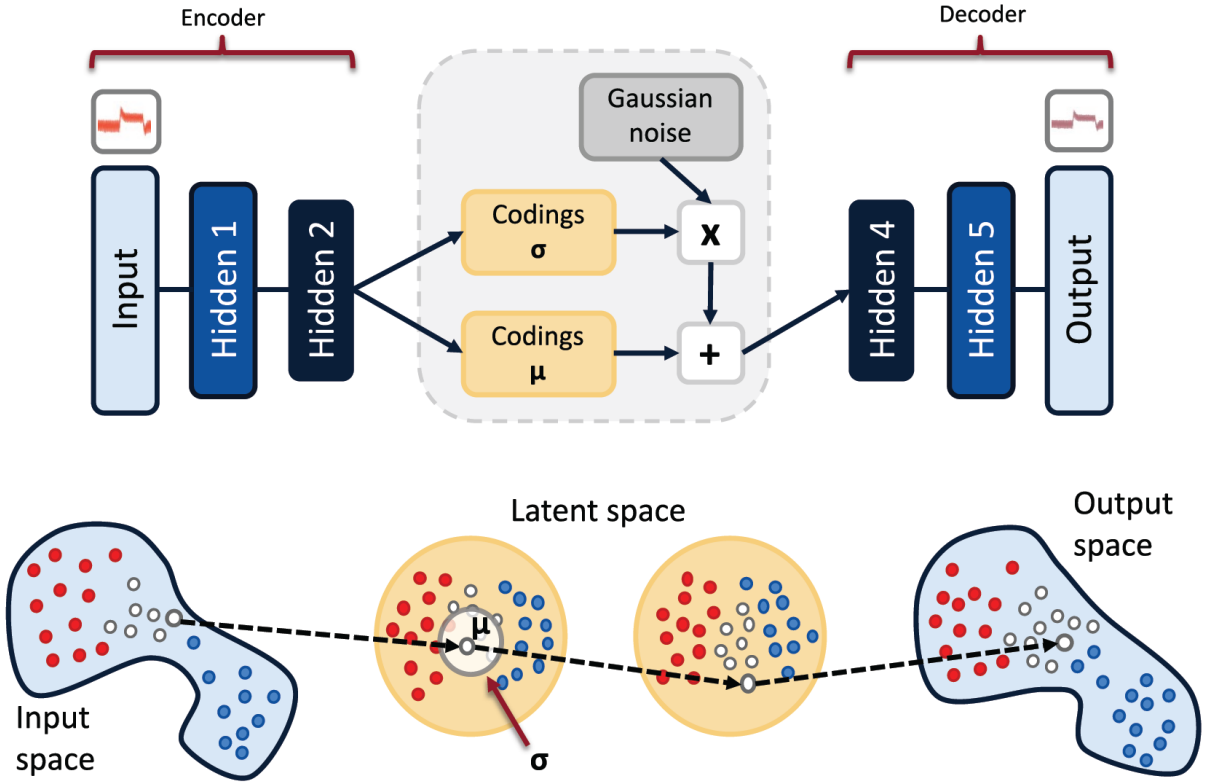
VAE Inference

- VAE mechanism



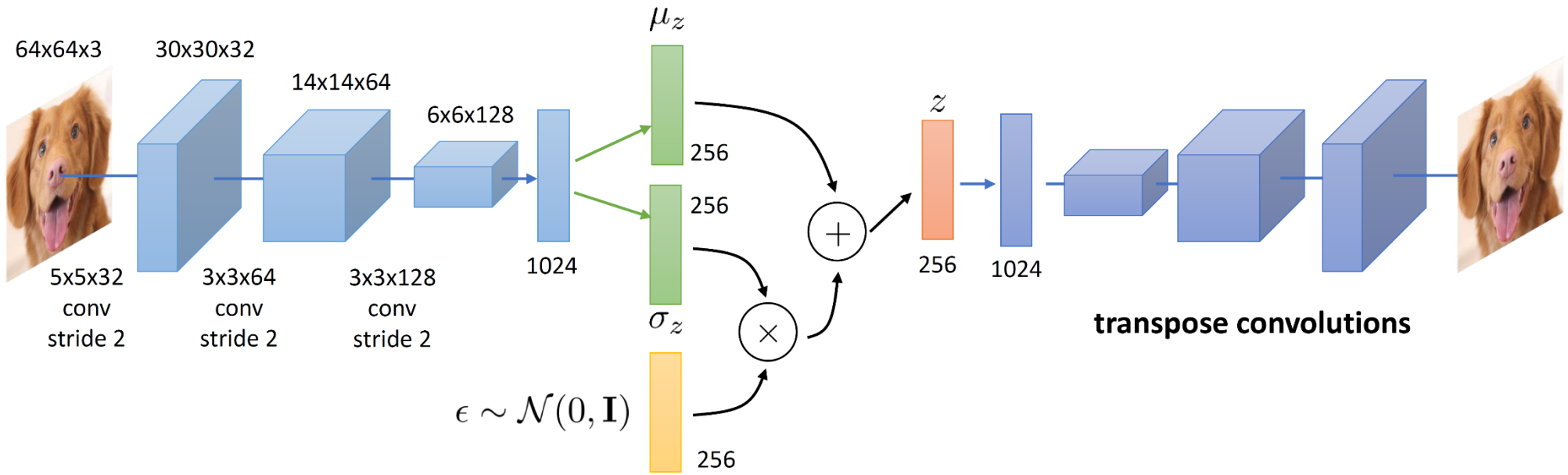
VAE Inference

- VAE mechanism



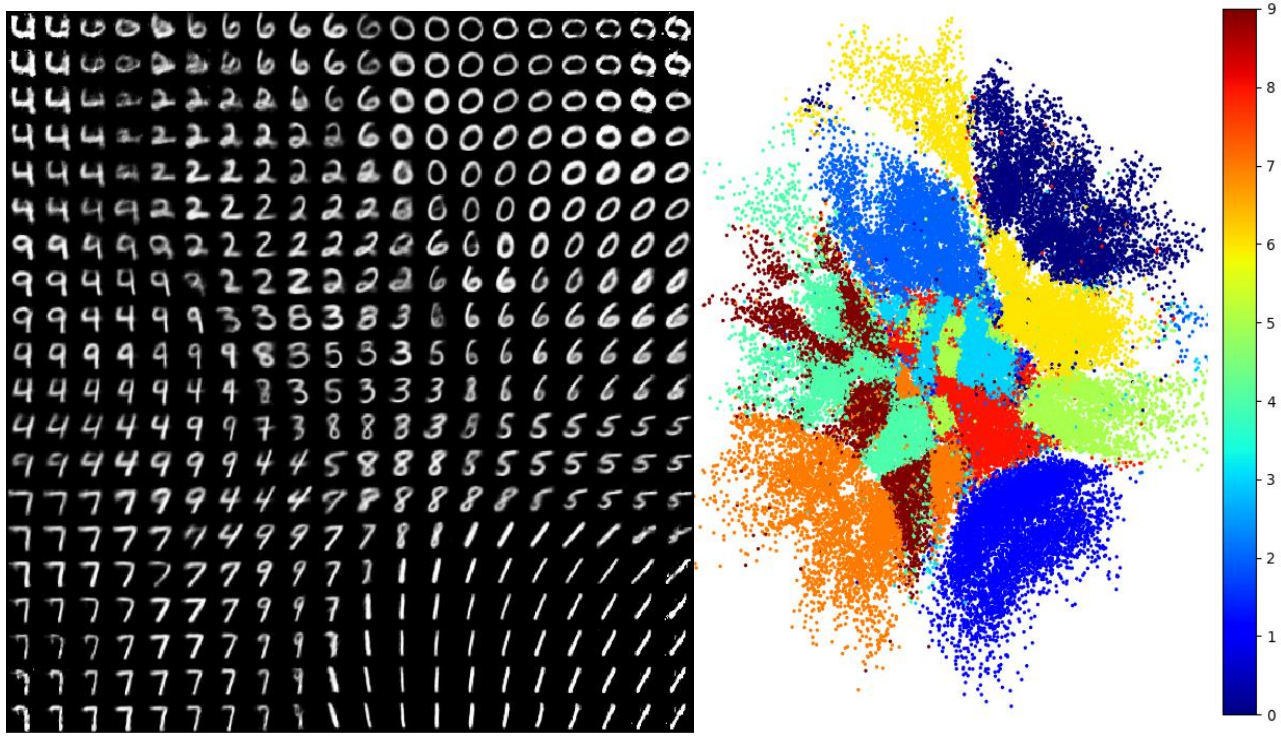
Convolutional VAE

- VAE mechanism



Example

- Dataset: MNIST, Dataset architecture: 500-500-2-500-500



Autoencoder Implementation

- See:
  - https://blog.keras.io/building-autoencoders-in-keras.html (AEs and Vanilla VAE)
  - https://keras.io/examples/generative/vae/ (Convolutional VAE)

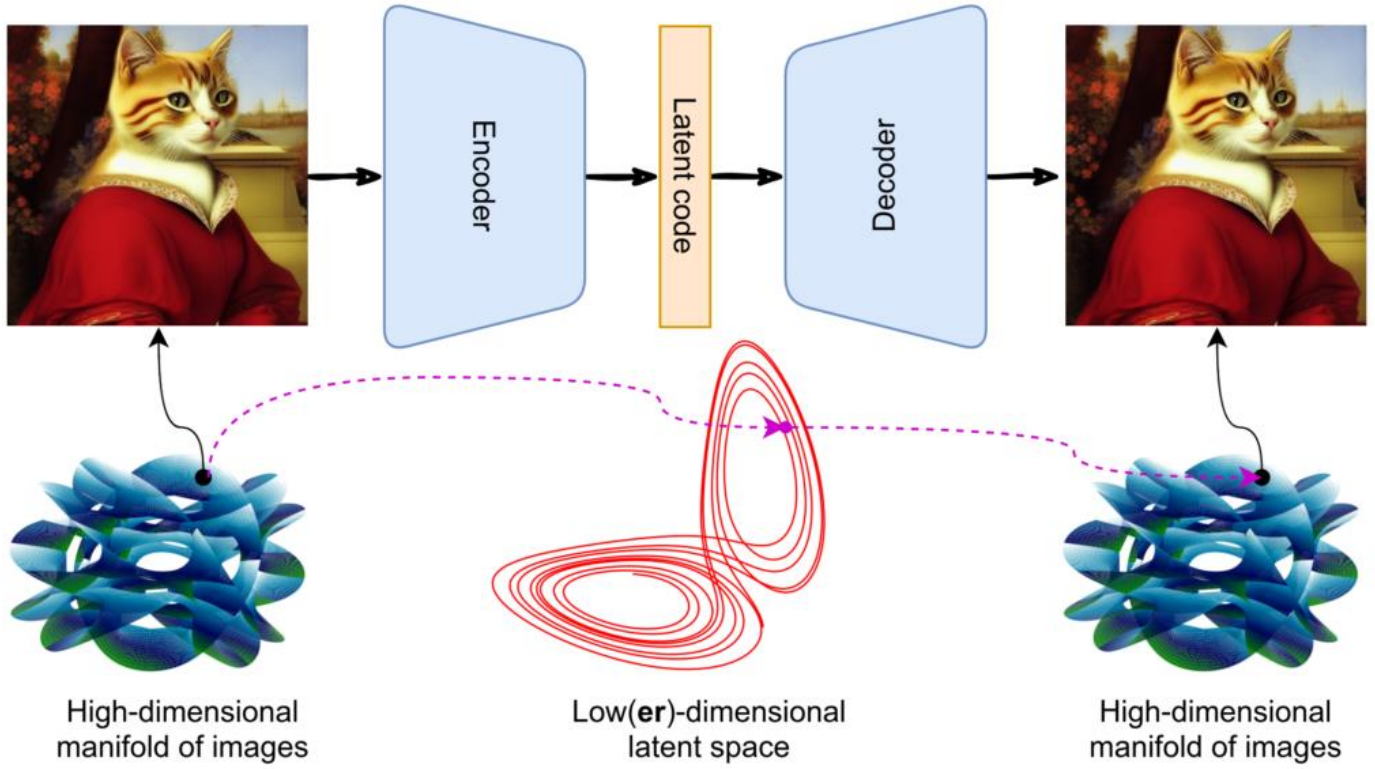
Examples

- Dataset: MNIST, Dataset architecture: 500-500-2-500-500
- Output (bottom) by feed the data (top) into overall VAE net:

- Feed a sample $z \sim \mathcal{N}(0, \mathbf{I})$ into the decoder module:

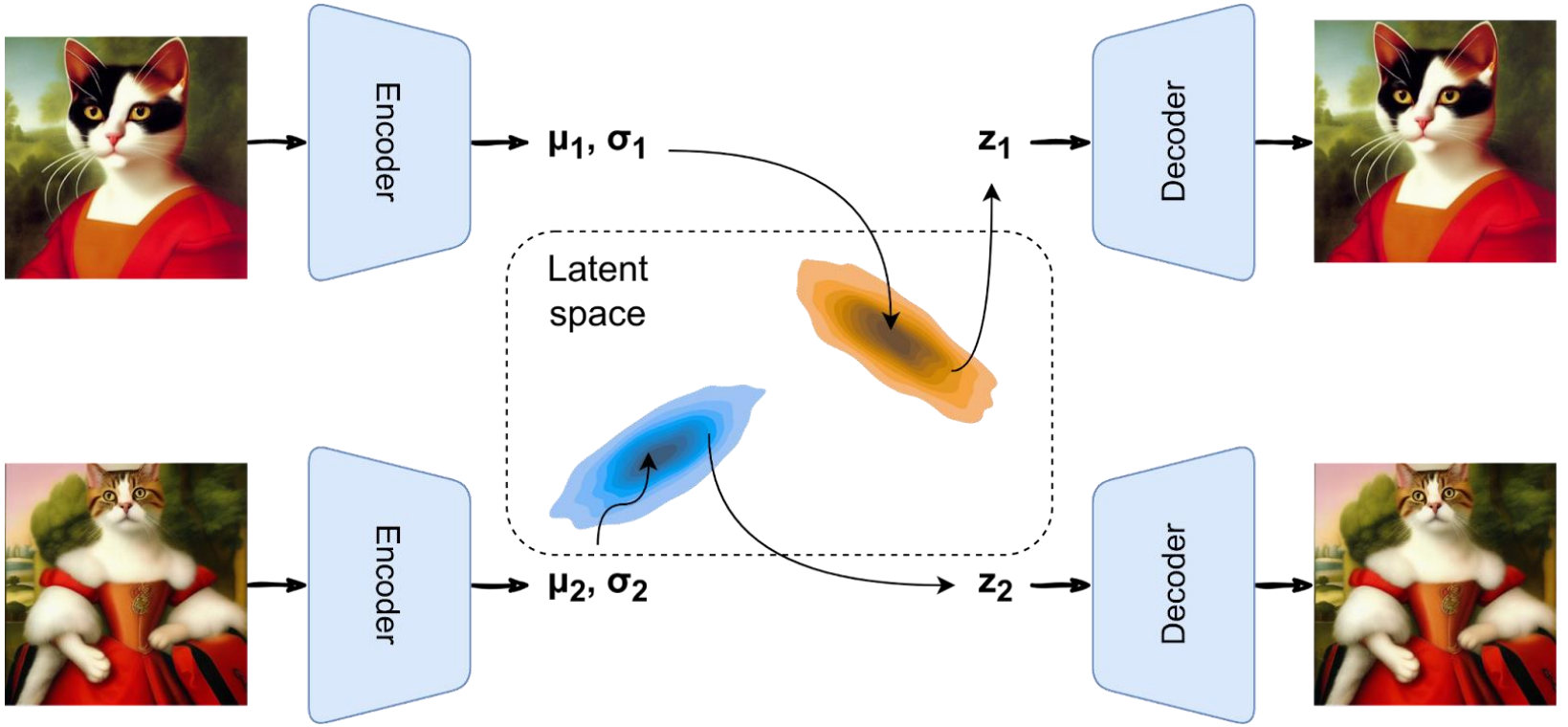
VAE Training phase

- Training phase

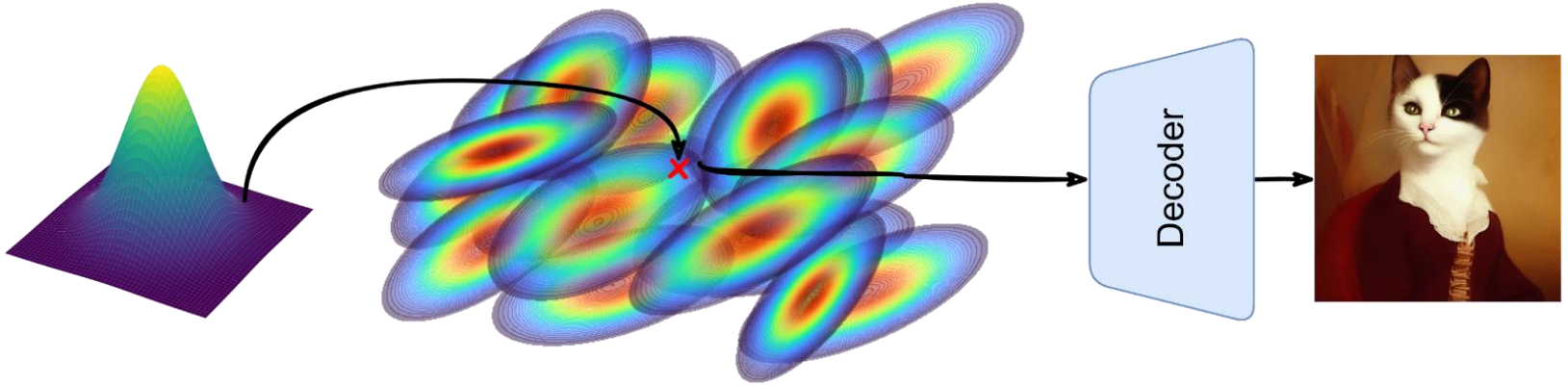


VAE Training + Inference

- Training + Inference



Latent space dimension effect

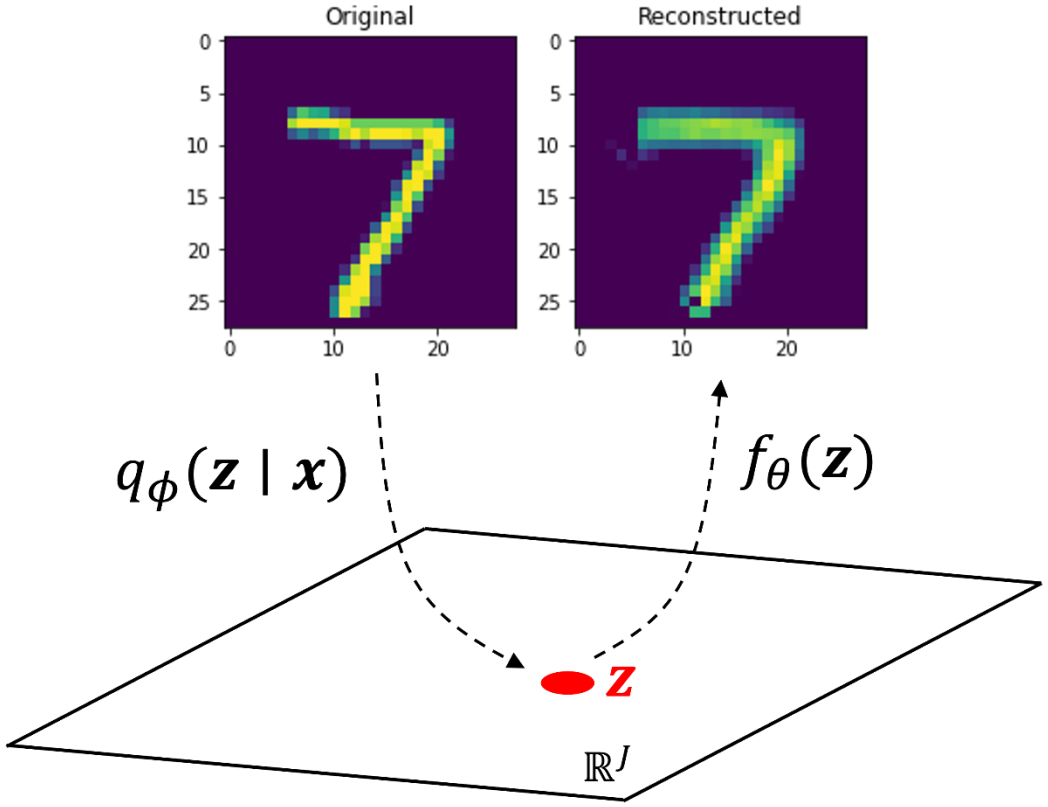


VAE Numerical Examples

- Intermediate Layer Example:
  - $\mu = [0.1 \quad 1.2 \quad 0.2 \quad \ldots]$
  - $\sigma = [0.2 \quad 0.5 \quad 0.8 \quad \ldots]$
  - $sampling = [\mathcal{N}(0.1, 0.2^2) \quad \mathcal{N}(1.2, 0.5^2) \quad \mathcal{N}(0.2, 0.8^2) \quad \ldots]$
  - $sampled\ vector = [0.28 \quad 1.65 \quad 0.92 \quad \ldots]$

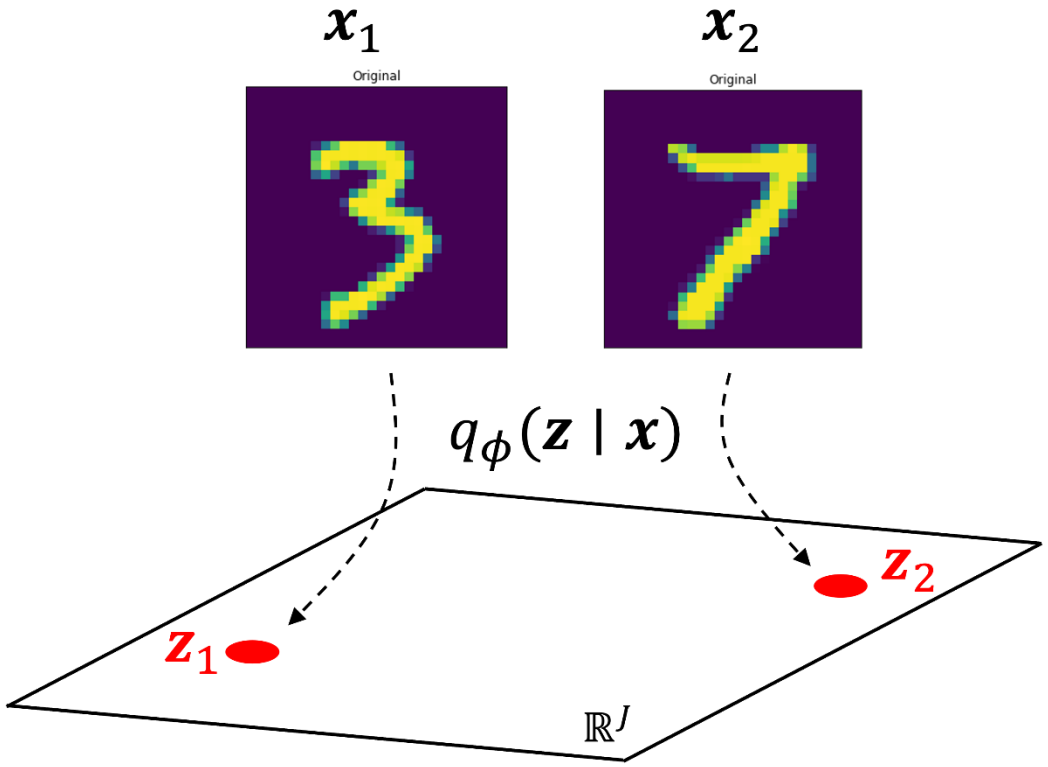
VAE Illustration

- VAE during training:



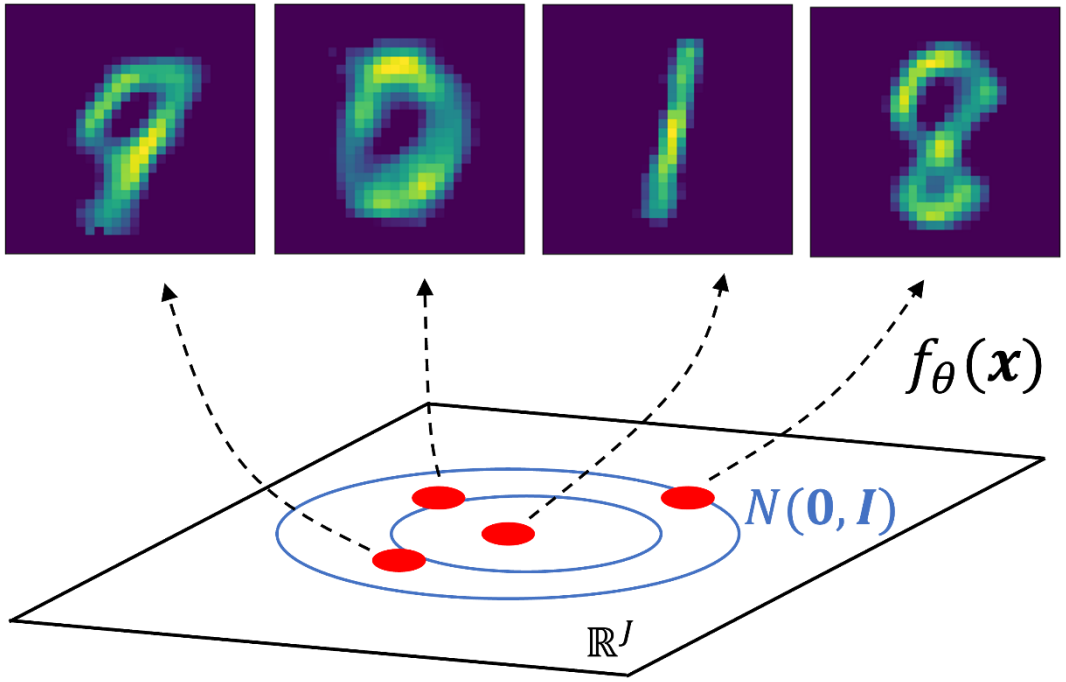
VAE Illustration

- VAE during training:



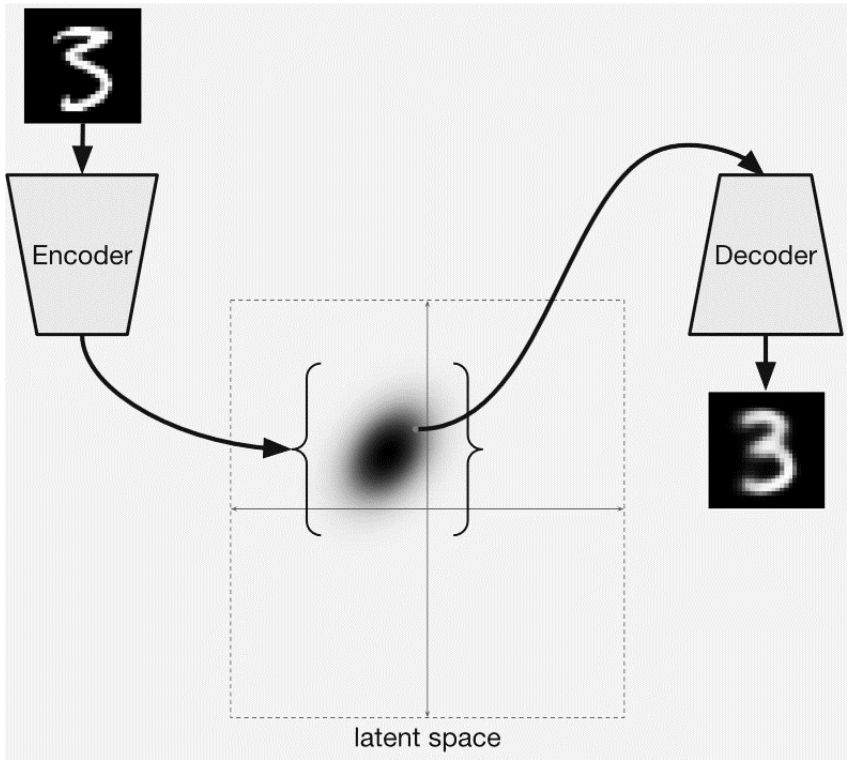
VAE Illustration

- VAE during Inference:



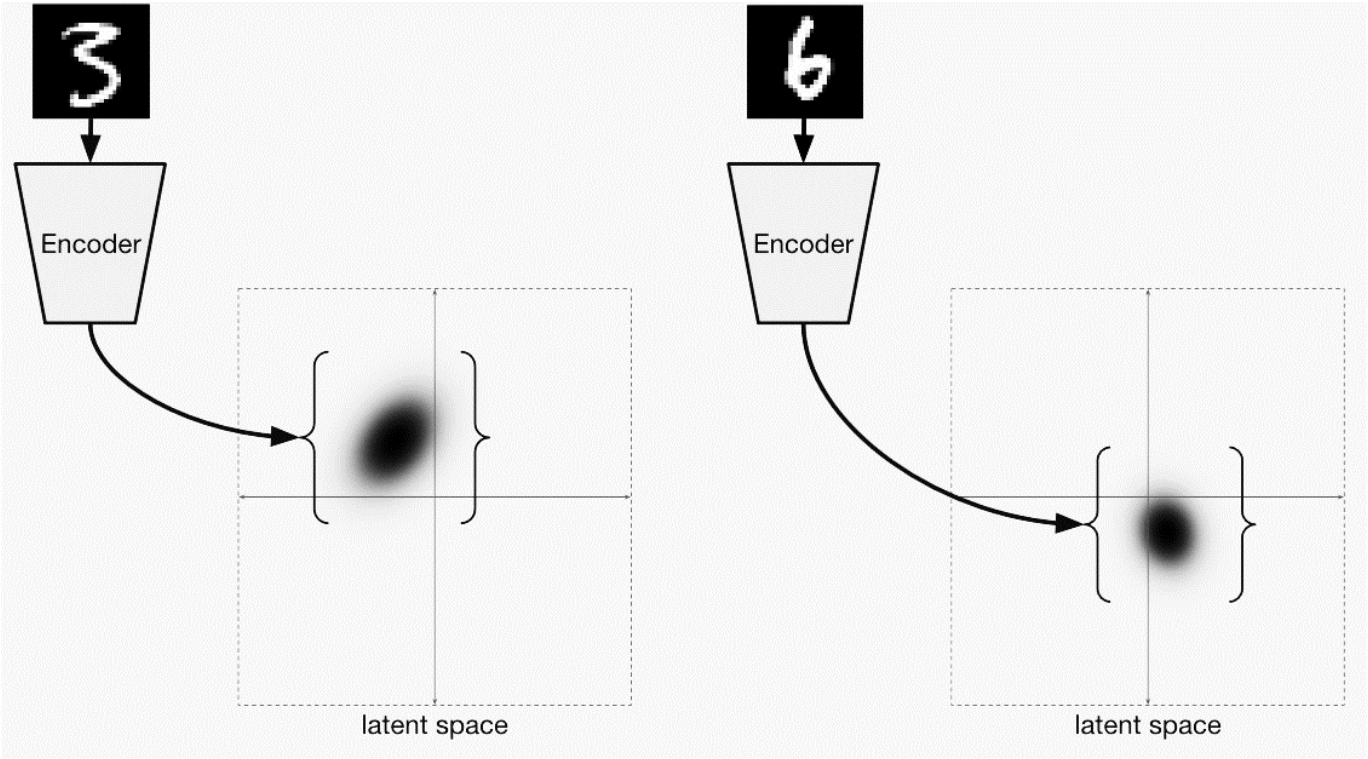
VAE Illustration

- VAE during training:



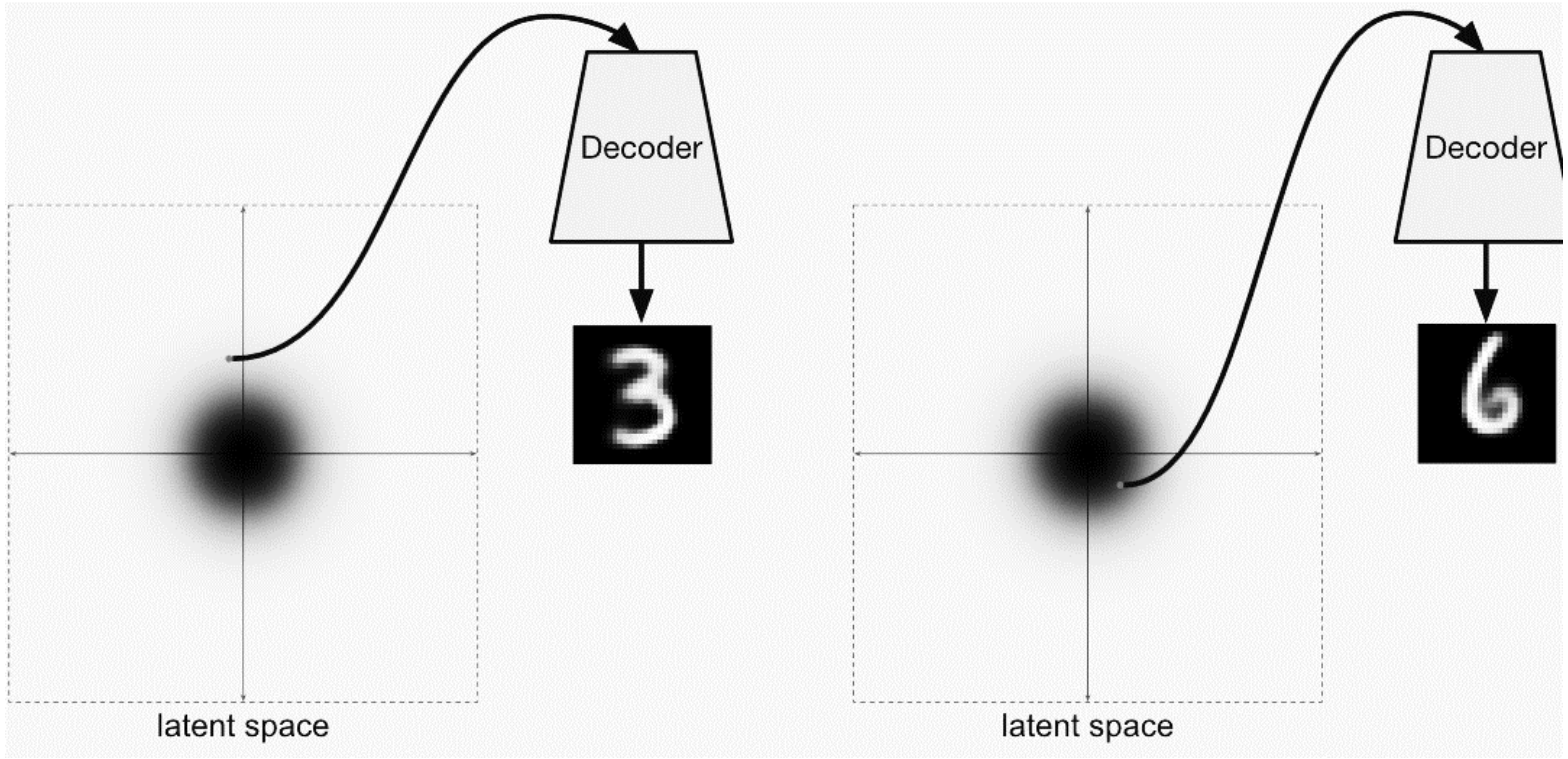
VAE Illustration

- VAE Encoder:



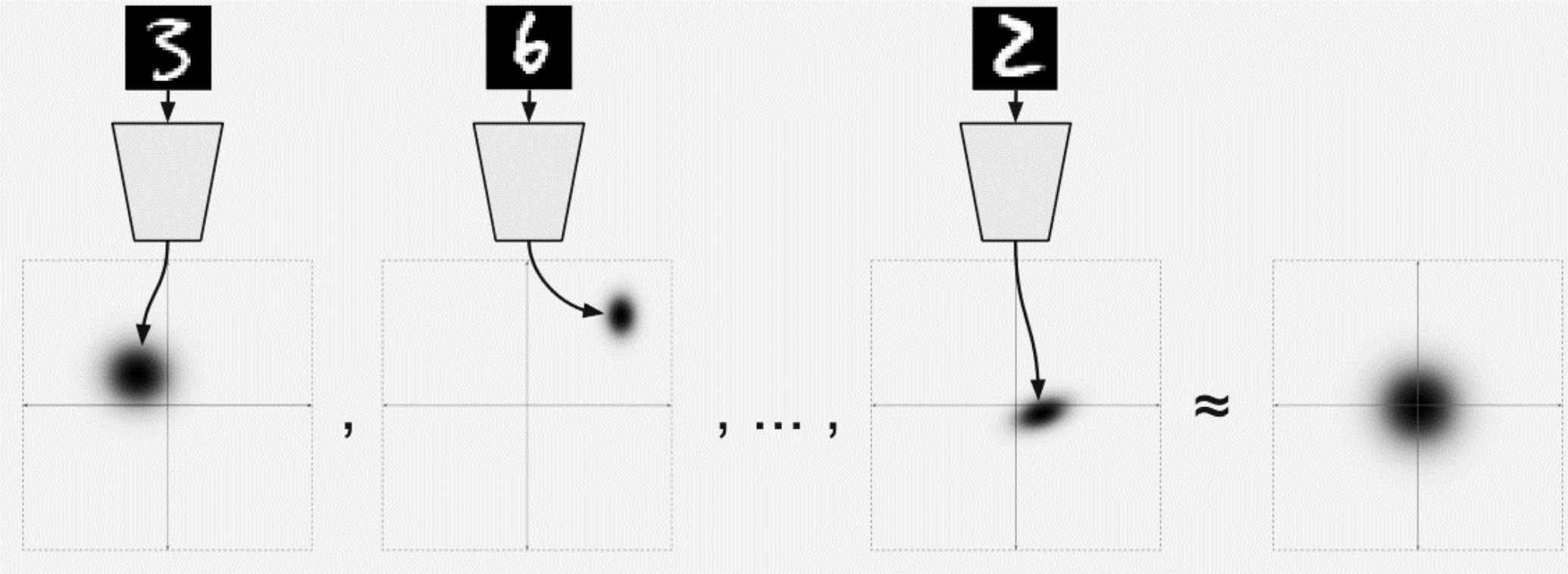
VAE Illustration

- VAE Decoder:



VAE Illustration

- VAE Encoder:



Conditional Variational Autoencoder (CVAE)

- Problem with VAE:
  - No control on the data generation process
  - Consider (encoder input: Alphabetic ASCII code, output: Handwritten Character)

$$z^* \rightarrow \mathcal{A}$$

  - After training which input generate $\mathcal{H}$
- Encoder models, z, based on X, not different type of X
- Decoder models, X, based on z only

Conditional Variational Autoencoder (CVAE)

- In CVAE:
  - Conditioning *Encoder* on an extra variable: $Q(z|X,c)$
  - Conditioning *Decoder* on an extra variable: $P(X|z,c)$.
- CVAE Loss function:

$$P(X|c) - D_{KL}[Q(z|X,c)\|P(z|X,c)] = \mathbb{E}\big(logP(X|z,c)\big) - D_{KL}[Q(z|X,c)\|P(z|c)]$$

- The *real* latent variable, z, is distributed under $P(z|c)$
  - For each c we have one $P(z)$
- c may comes from categorical distribution expressing the label of data

Conditional Variational Autoencoder (CVAE)

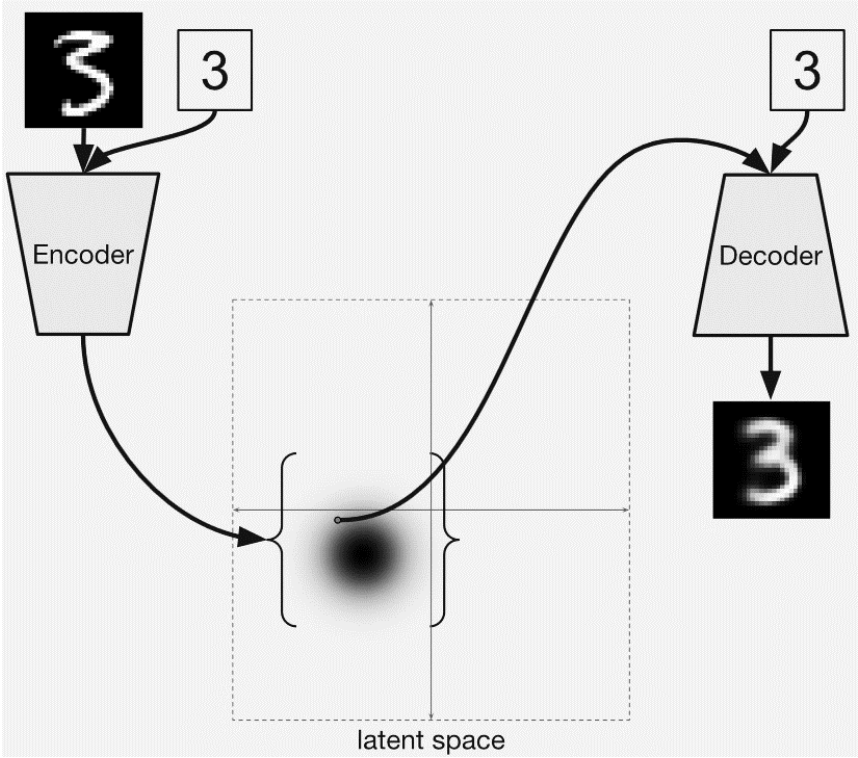
- How to incorporate c (suppose: a *one-hot* vector of *label*):

$$c = [0\ 0\ 0\ 1\ 0\ 0\ 0]$$

- Main idea: concatenation!

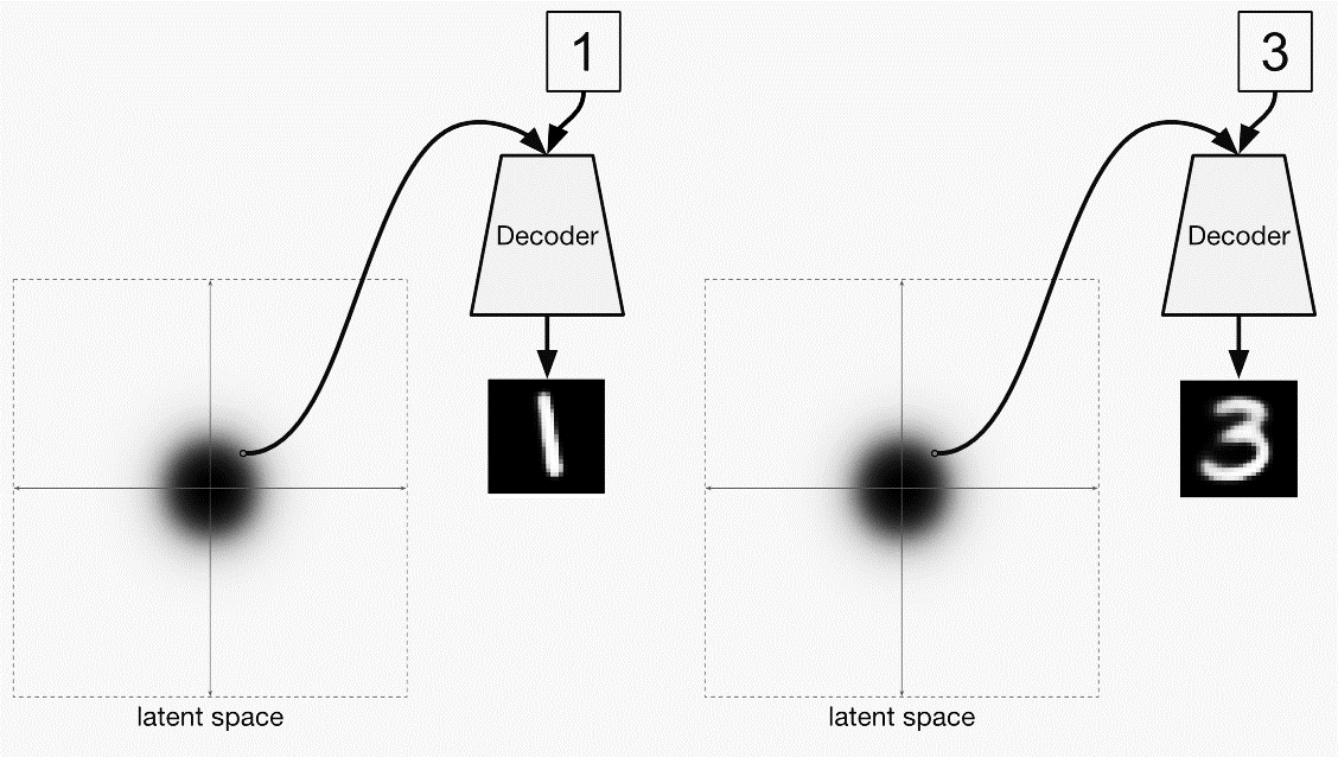
Conditional Variational Autoencoder (CVAE)

- CVAE Training:



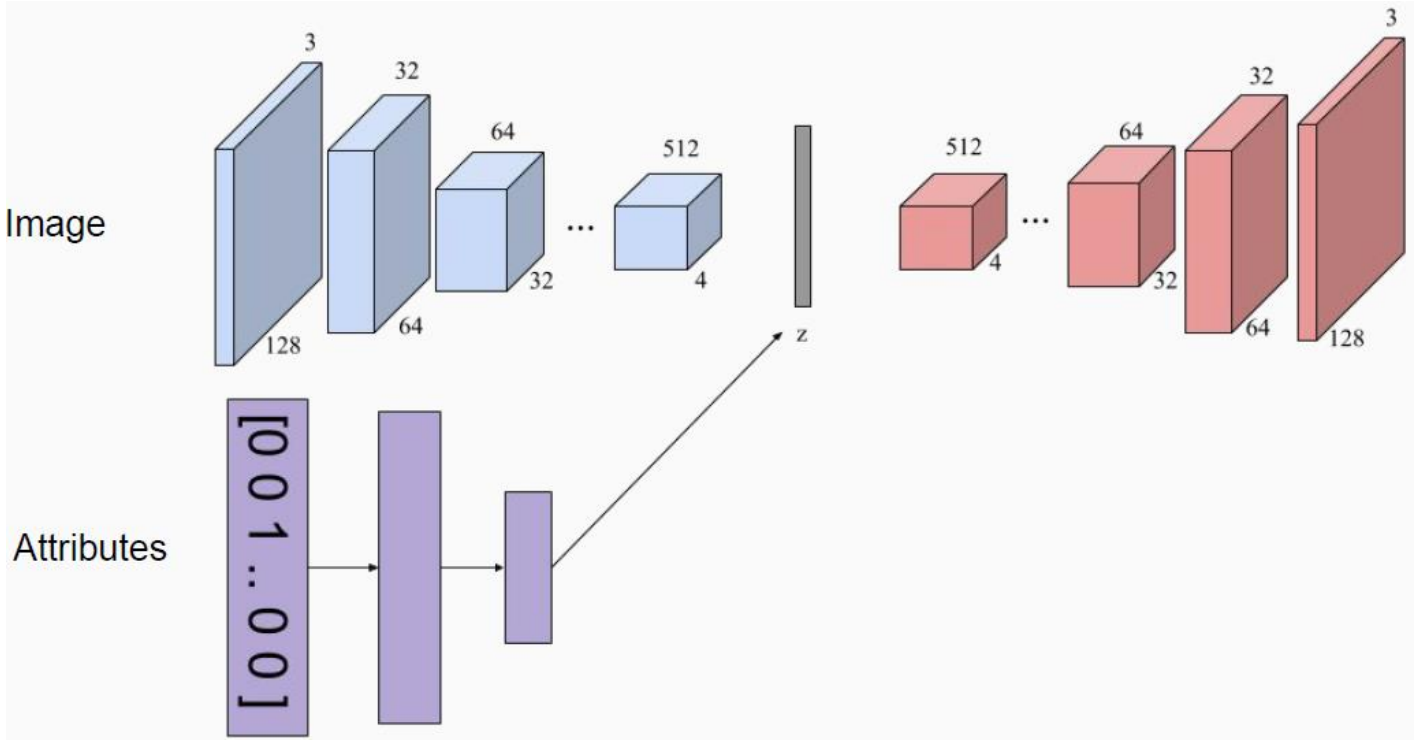
Conditional Variational Autoencoder (CVAE)

- CVAE Inference:



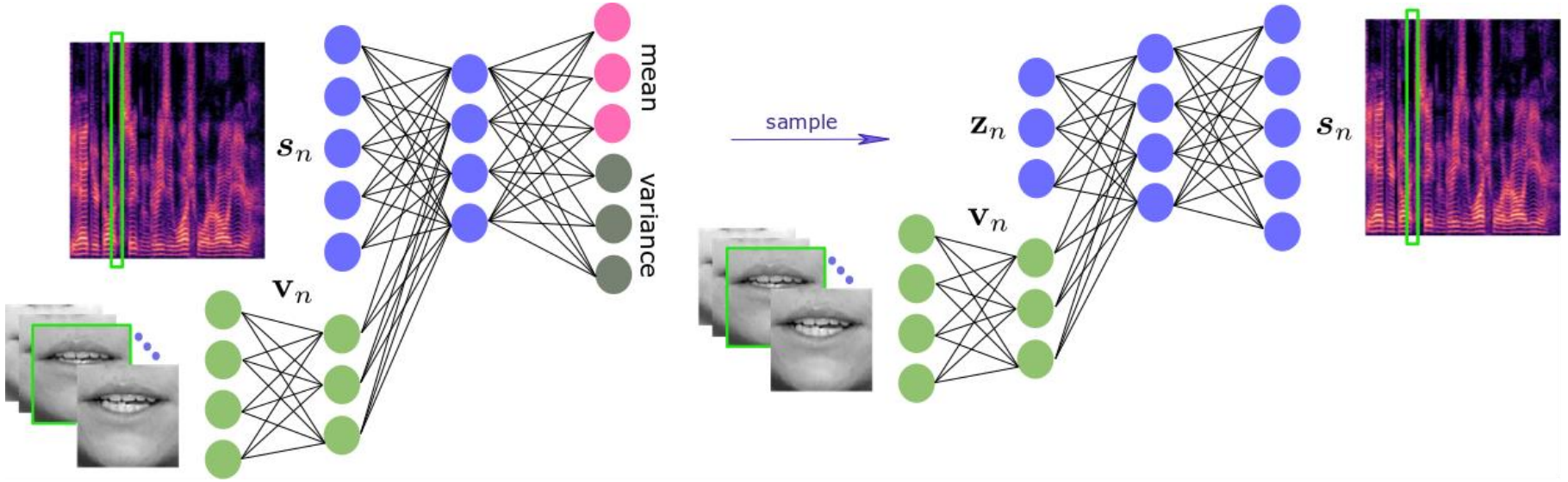
Conditional Variational Autoencoder (CVAE)

- An alternative:



Conditional Variational Autoencoder (CVAE)

- An alternative:



Conditional Variational Autoencoder (CVAE)

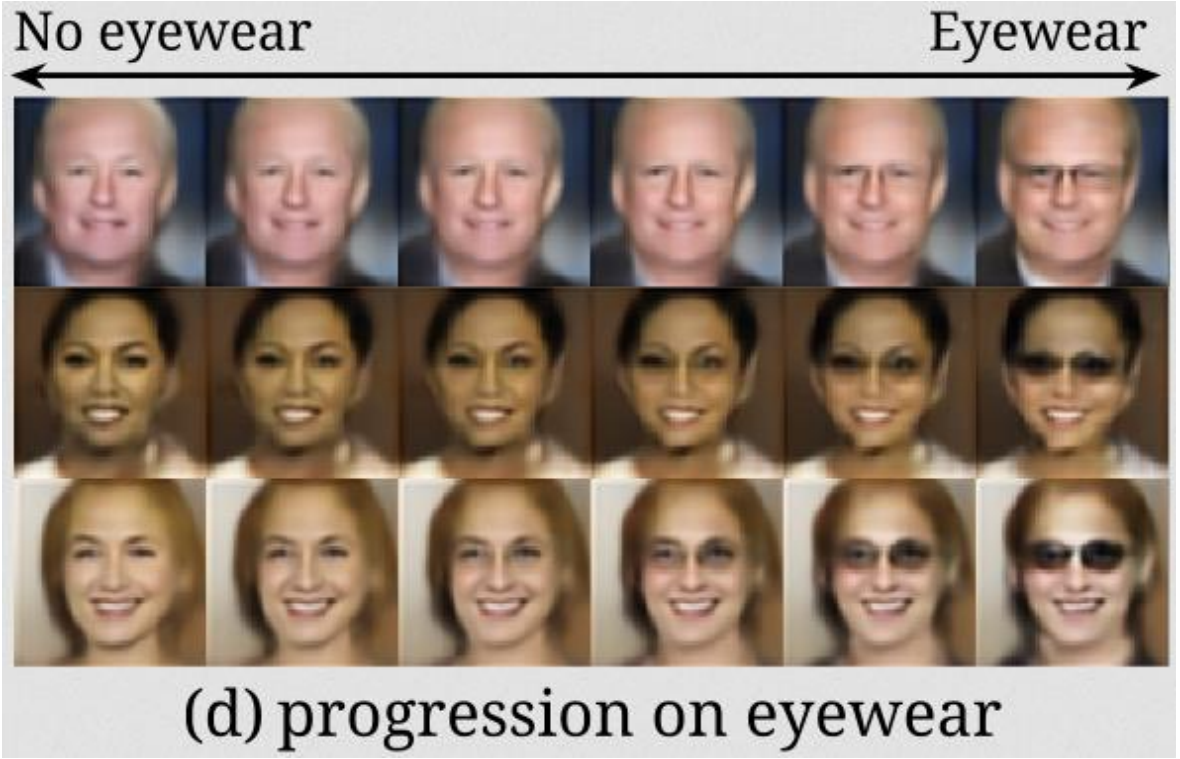
- Sample application: Conditional Image Generation from Visual Attributes.

  - *Attribute2Image*

- c: involve image attribute
  - *A young man in the street wearing glasses with black hair is crying.*
  - c = [**1.3 1.0 101 0 1.3 -1.2**]
- Attribute2Image
  - One CVAE for foreground
  - One CVAE for background

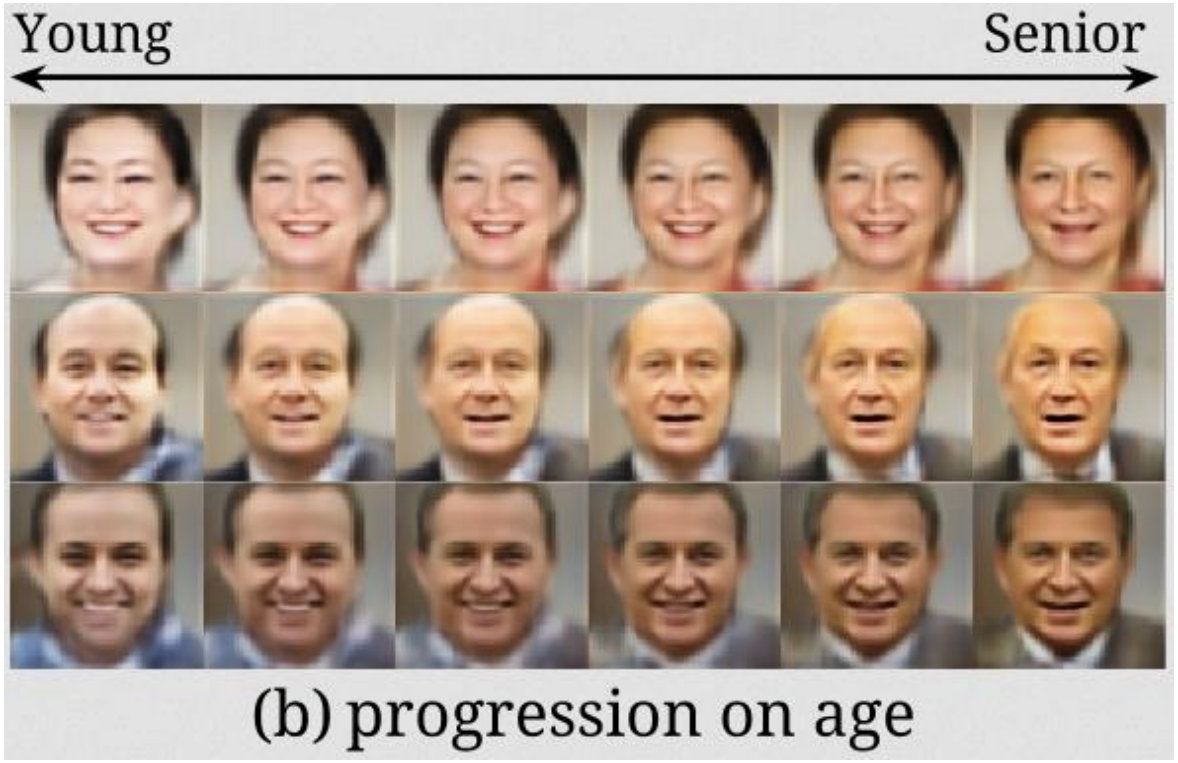
Conditional Variational Autoencoder (CVAE)

- Attribute2Image



Conditional Variational Autoencoder (CVAE)

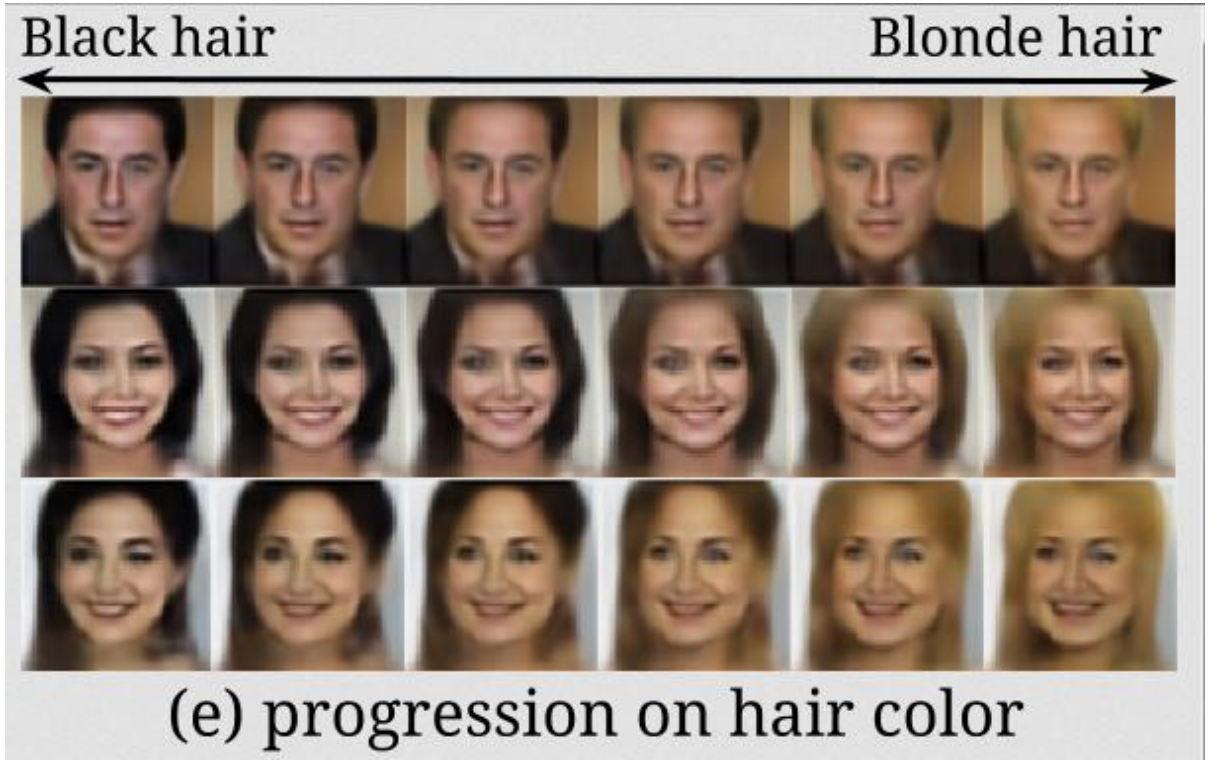
- Attribute2Image



(b) progression on age

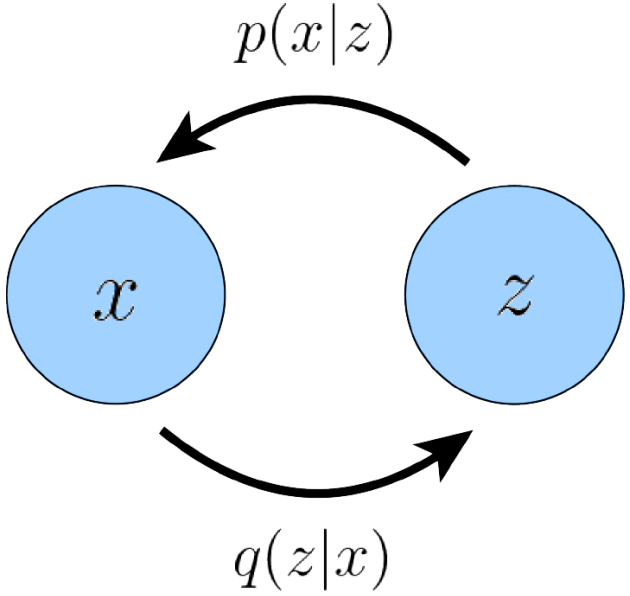
Conditional Variational Autoencoder (CVAE)

- Attribute2Image



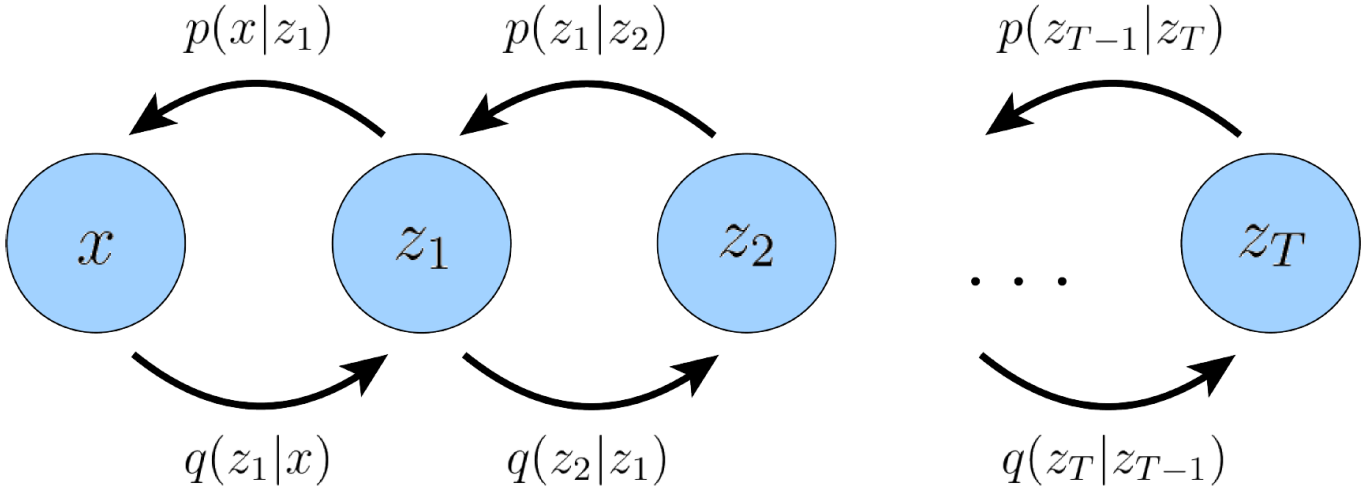
VAE variants

- Graphic of VAE:



Hierarchical Variational Autoencoders (HVAE)

- A Hierarchical Variational Autoencoder (HVAE) is a generalization of a VAE that extends to multiple hierarchies over latent variables.
- Under this formulation, latent variables themselves are interpreted as generated from other higher-level, more abstract latents.



Hierarchical Variational Autoencoders (HVAE)

- The inference model can then be defined in two ways:
  - bottom-up

$$Q_{1:T}(\boldsymbol{z}|X) = Q(z_1|X)\prod_{k=2}^{T} Q(z_k|z_{k-1})$$

$$Loss = \mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[log p_\theta(\boldsymbol{x}|z_1)] - D_{KL}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x})\middle|p(\boldsymbol{z})\right)$$

$$D_{KL}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x})\middle|p(\boldsymbol{z})\right) = \left[\sum_{i=1}^{T-1} log\frac{p_\theta(z_i|z_{i+1})}{q_\varphi(z_i|z_{i-1})} + log\frac{p_\theta(z_T)}{q_\varphi(z_T|z_{T-1})}\right]$$

Hierarchical Variational Autoencoders (HVAE)

- The inference model can then be defined in two ways:
  - top-down is similar

$$Q_{1:T}(z|X) = Q(z_T|X) \prod_{k=T-1}^{1} Q(z_k|z_{k+1})$$

β-VAE

- β-VAE has special emphasis to discover disentangled latent factors. As VAE, the goal is to maximize the probability of generating real data, while keeping the distance between the real and estimated posterior distributions is small:

$$\max_{\theta,\varphi} \mathbb{E}_{X\sim\mathcal{D}}\left[\mathbb{E}_{z\sim Q_\varphi(z|X)}[logP_\theta(X|z)]\right], \qquad s.t.\, D_{KL}\left[Q_\varphi(z|X)\middle\|P_\theta(z)\right] < \delta$$

- Using Lagrange multiplier:

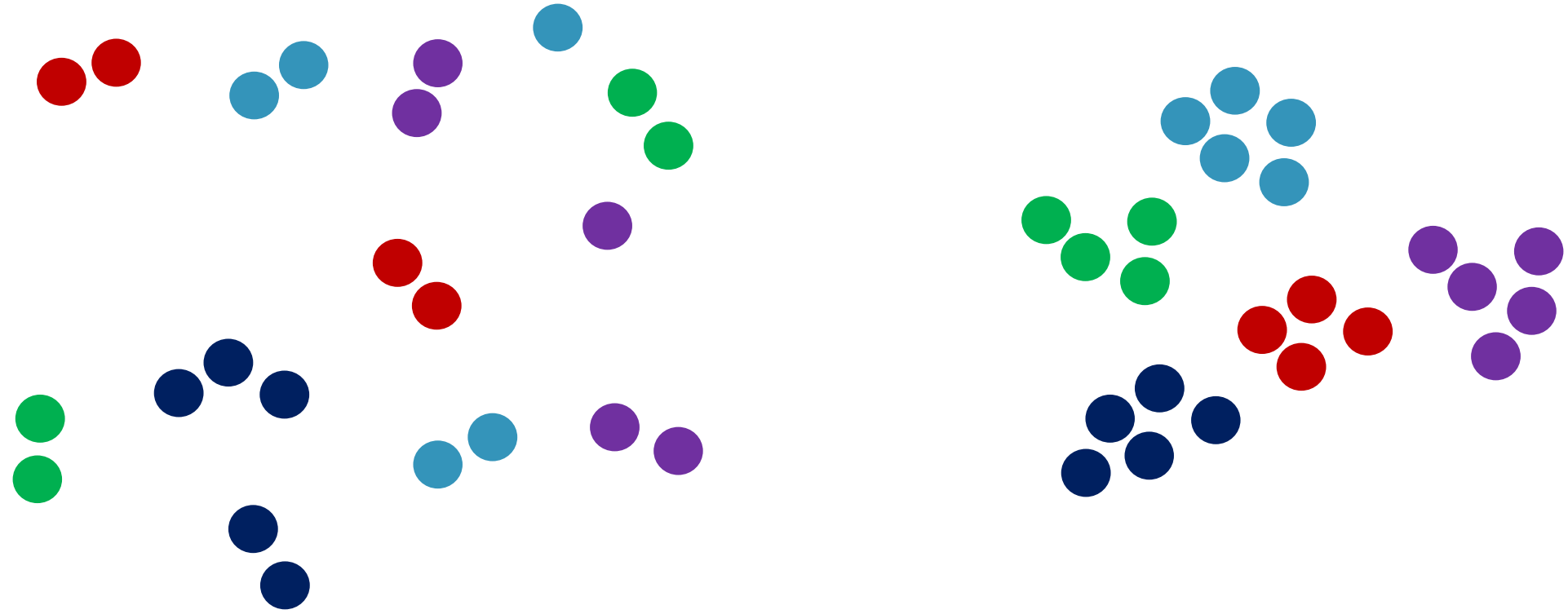
$$Loss_{VAE} = -\mathbb{E}_{z\sim Q_\varphi(z|X)} logP_\theta(X|z) + \beta D_{KL}\left[Q_\varphi(z|X)\middle\|P_\theta(z)\right], \beta > 1$$

Vector Quantized VAE (VQ-VAE)

- Backbone of original Dall-E model

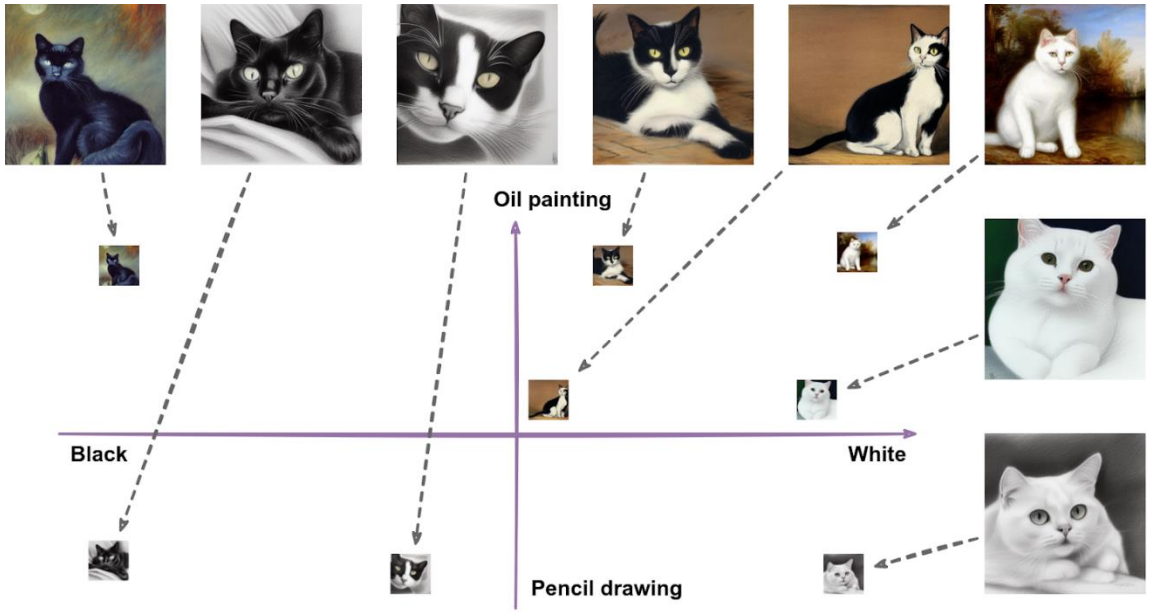
VQ-VAE

- Comparison between Messy-Well distributed



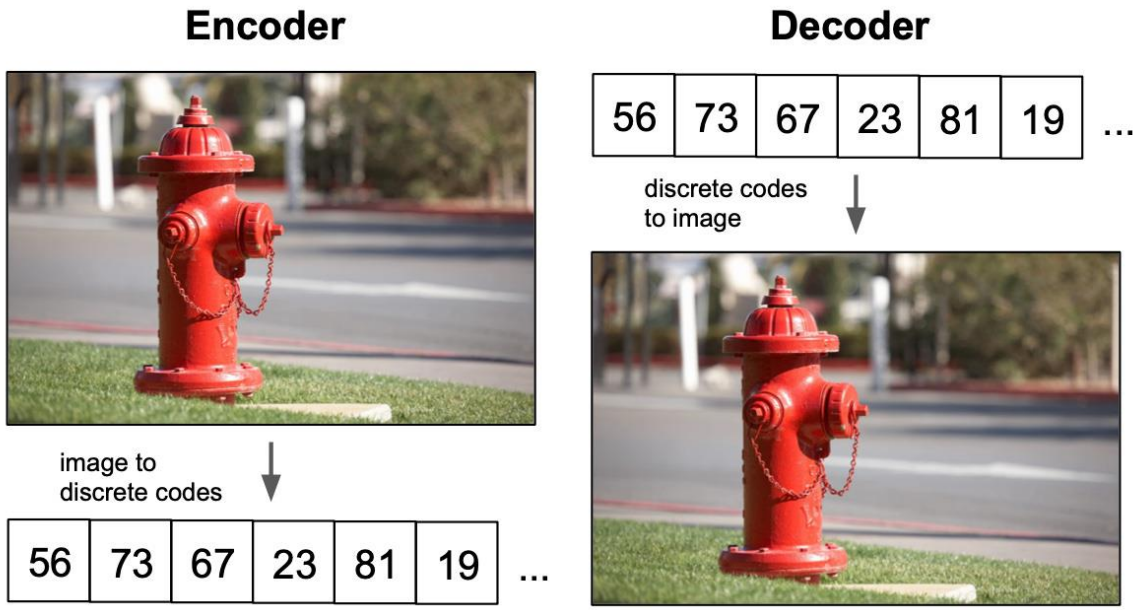
VQ-VAE

- If the autoencoder is designed well, this may result in a latent space where certain dimension in latent space correspond to specific properties of the image, ideally:



VAE vs VQ-AVE

- The fundamental difference between a VAE and a VQ-VAE is that VAE learns a continuous latent representation, whereas VQ-VAE learns a discrete latent representation. (Note: Discrete NOT Digital)

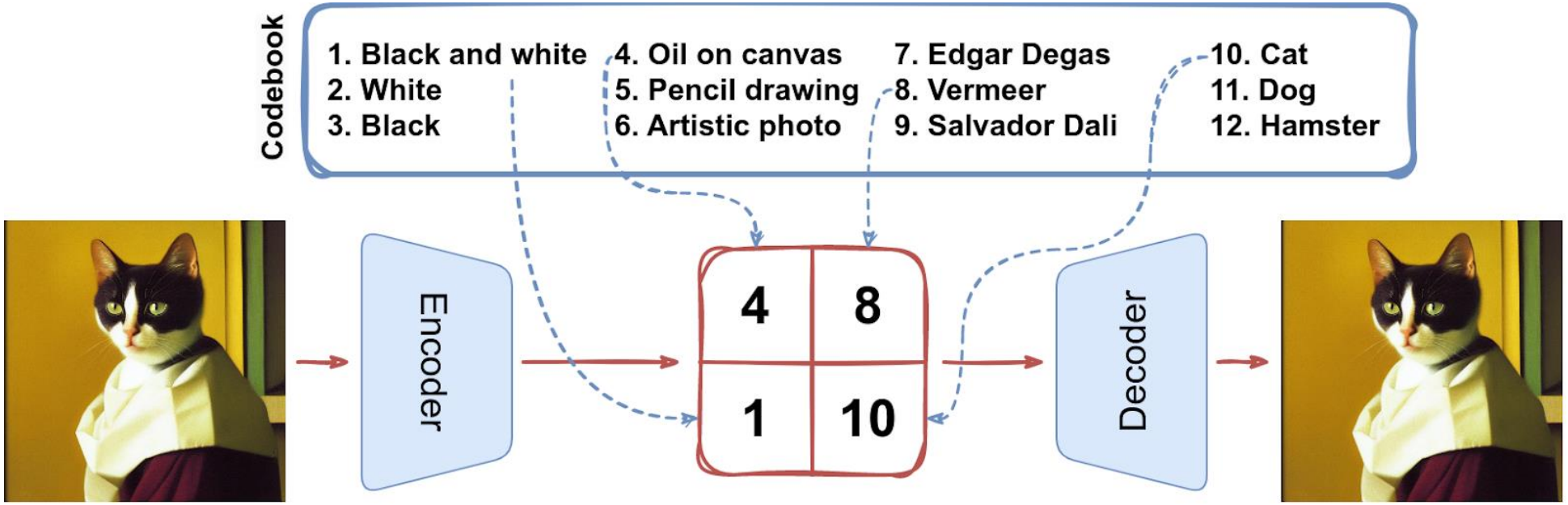


VQ-VAE

- In VQ-VAE latent space is *discrete* at a scale sufficient for general-purpose images.
- VQ-VAE train a *finite* vocabulary (codebook) and encodes images as fixed sets (tensors) of *discrete* codes

VQ-VAE

- A well designed discrete latent space



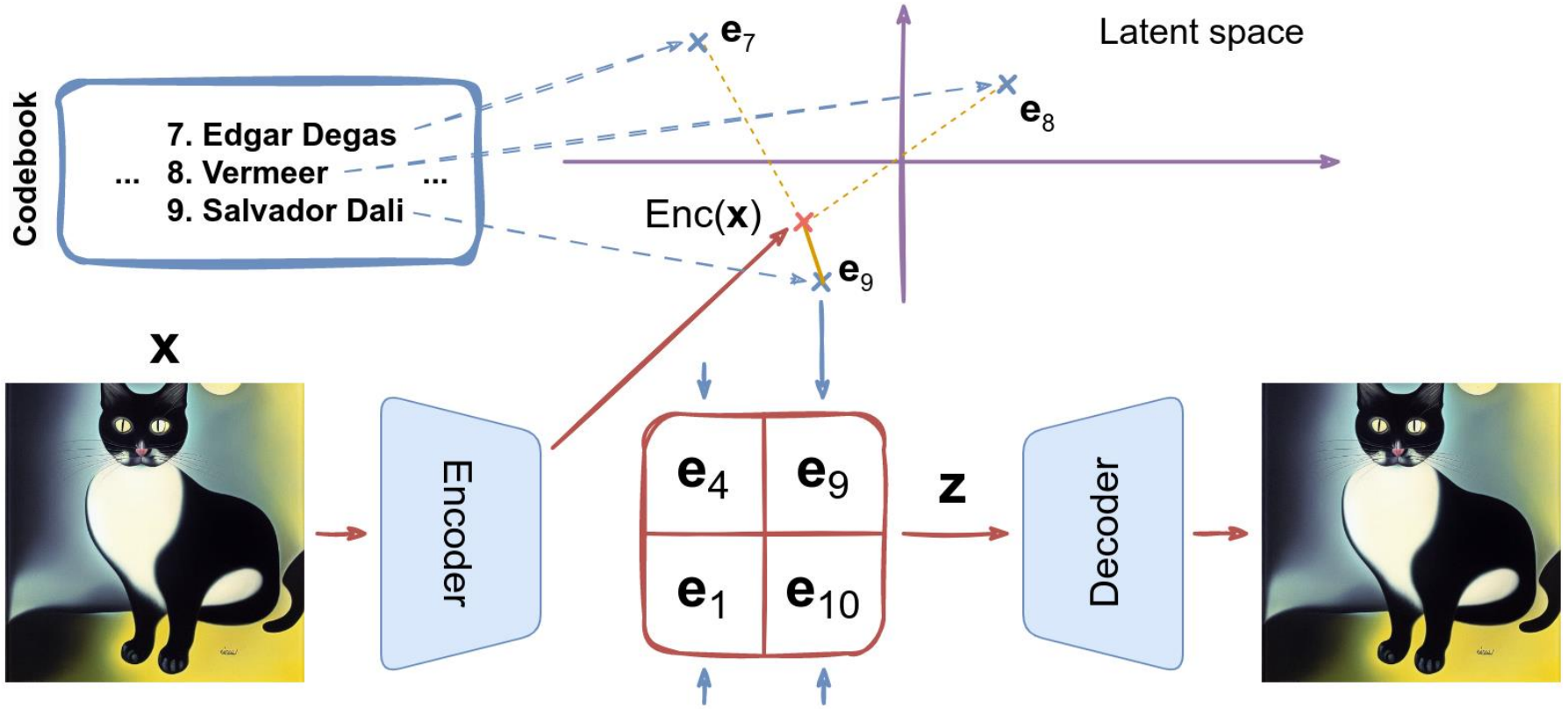
VQ-VAE Steps

- The encoder, takes an image X as input and produces *a set* of latent vectors $Enc(X)$
- For every latent vector, we find the *nearest codebook vector* in the *latent space* and replace it with this codebook vector; the resulting code consists only of codebook vectors;
- The decoder receives as input the tensor of vectors, with the same dimensions as the encoder had produced, but actually the latent code is now *discrete*.

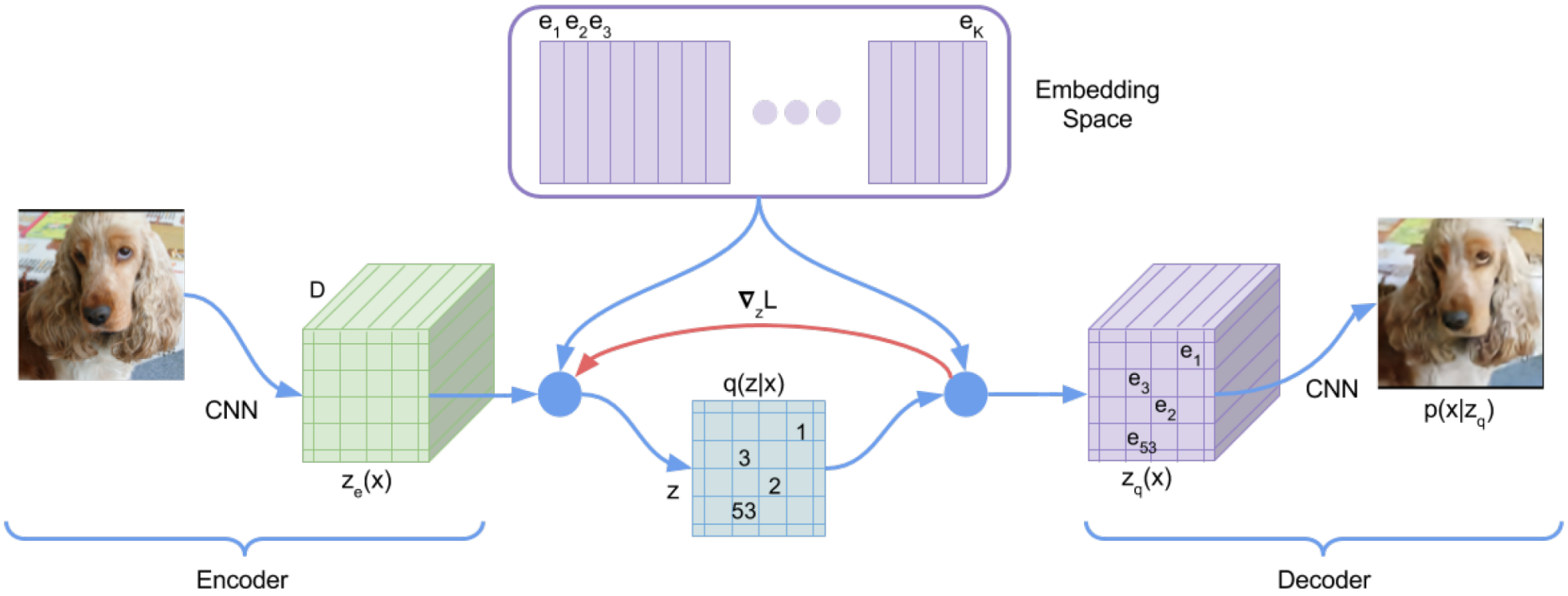
VQ-VAE Steps

- Note:
- While each component of the latent code is still a *continuous vector* of *real* numbers, there's now only a finite number of possibilities for each of the vectors.
- Latent space is a D-dimensional dictionary of K-codebooks
- Latent space: $\{e_i\}_{i=1}^{K}, e_i \in \mathbb{R}^D$ and $Enc(x) \in \mathbb{R}^D$

Latent Space Discretization



VQ-VAE Architecture



VQ-VAE Variational Interpretation

- The posterior categorical distribution $q(z|x)$ probabilities are defined as follows:

$$q(z = k|x) = \begin{cases} 1, & k = argmin_j \|z_e(x) - e_j\| \\ 0, & otherwise \end{cases}$$

- where $z_e(x)$ is the output of the encoder network. VQ-VAE view this model as a VAE in which we can bound $\log p(x)$ with the ELBO. The proposal distribution $q(z = k|x)$ is deterministic, and by defining a simple uniform prior over z we obtain a KL divergence constant and equal to $logK$.

VQ-VAE Details

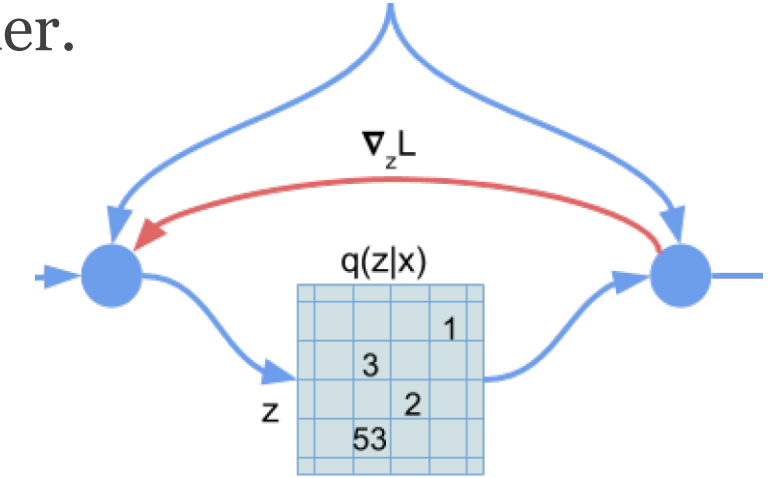
- The representation $z_e(x)$ is passed through the discretisation bottleneck followed by mapping onto the nearest element of embedding e as given:

$$q(z = k|x) = \begin{cases} 1, & k = argmin_j \|z_e(x) - e_j\| \\ 0, & otherwise \end{cases}$$

$$Quantize(z_e(x)) = z_q(x) = e_k, k = argmin_j \|z_e(x) - e_j\|$$

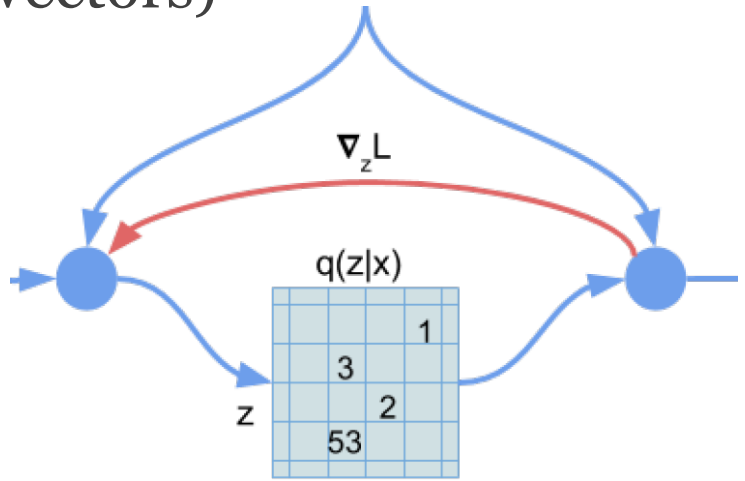
VQ-VAE Details

- Note that there is no gradient defined for $argmin_j \|\cdot\|$, VQ-VAE just copy gradients from decoder input $z_q(x)$ to encoder output $z_e(x)$.
- During forward pass the nearest embedding $z_q(x)$ is passed to the decoder, and during the backwards phase the gradient $\nabla_z Loss$ is passed unaltered to the encoder.



VQ-VAE Details

- This $argmin_{(.)}\|\cdot\|$ operation is non-differentiable with respect to the encoder. But in practice everything seems to work fine if just pass the decoder gradient *directly* through this operation to the encoder (set its gradient to 1 wrt the encoder and the quantized codebook vector; and to 0 wrt all other codebook vectors)



VQ-VAE Details (Capacity)

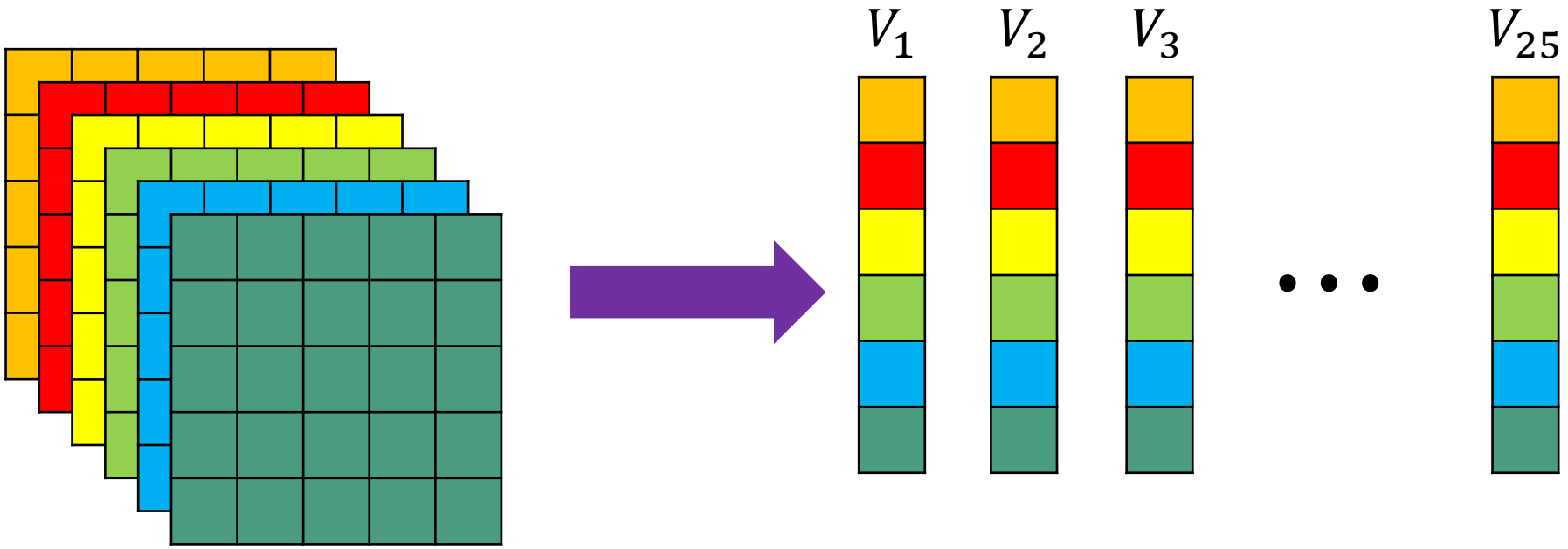
- A puzzle: There are only a finite number (K) of latent codes in dictionary.
- How could one ever expect to generate the *huge* quantity and diversity of possible images when the decoder can only accept the set of codebook vectors as input?
- This would be a valid concern if the encoder generate just *one* vector, but in actual VQ-VAEs the encoder usually produces a series of vectors.

VQ-VAE Details

- Consider an image generator VQ-VAE
- Input: $x \in \mathbb{R}^{(H \times W \times 3) \times B}, H: Height, W: Width, B: Batch\ size$
- Dictionary codebook: $e_i \in \mathbb{R}^{D}$
- Encoder output: $z_e(x) \in \mathbb{R}^{(h \times w \times D) \times B}$, $D = C$ number of filters/channels
- After reshaping we have $(h * w * B)$ vectors each of the dimensionality D
- Now we have $(h * w * B)$ nearest vector in K-codebook dictionary
- In other word we have $(h * w * B)$ quantized vector (e_k)
- Reshape back to $z_q(x) \in \mathbb{R}^{(h \times w \times D) \times B}$ and copy gradient from $z_e(x)$ to $z_q(x)$

VQ-VAE Example

- $(5 \times 5 \times 6) \times 1$ feature map give 25, 6-dimensional quantized vectors
- K^{25} possibilities (for Dall-E v1: $8192^{32\times32}$ possibilities)



VQ-VAE Loss Function

- The proposed loss ($\mathbf{e} = z_q(\mathbf{x})$):

$$\mathcal{L}\big(\mathbf{x}, Dec(\mathbf{e})\big) = \|\mathbf{x} - Dec(\mathbf{e})\|_2^2 + \|sg[Enc(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|sg[\mathbf{e}] - Enc(\mathbf{x})\|_2^2$$

$$\mathcal{L}\big(\mathbf{x}, Dec(\mathbf{e})\big) = \|\mathbf{x} - Dec(\mathbf{e})\|_2^2 + \|sg[z_e(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|sg[\mathbf{e}] - z_e(\mathbf{x})\|_2^2$$

- where sg stands for the stop-gradient operator that is defined as *identity* at *forward* computation time and has *zero partial derivatives*, thus effectively constraining its operand to be a non-updated constant.
- Gradient copying: $\nabla_{z_e(\mathbf{x})}\|\mathbf{x} - Dec(\mathbf{e})\|_2^2 = \nabla_e\|\mathbf{x} - Dec(\mathbf{e})\|_2^2$

VQ-VAE Loss Function

- Loss Function:

$$\mathcal{L}\big(\mathbf{x}, Dec(\mathbf{e})\big) = \|\mathbf{x} - Dec(\mathbf{e})\|_2^2 + \|sg[z_e(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|sg[\mathbf{e}] - z_e(\mathbf{x})\|_2^2$$

- First term: The standard reconstruction loss.

VQ-VAE Loss Function

- Loss Function:

$$\mathcal{L}\big(\mathbf{x}, Dec(\mathbf{e})\big) = \|\mathbf{x} - Dec(\mathbf{e})\|_2^2 + \|sg[z_e(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|sg[\mathbf{e}] - z_e(\mathbf{x})\|_2^2$$

- Second term: Codebook alignment loss, whose goal is to get the *chosen codebook* vector as close to the *encoder output* as possible. There is a *stop gradient* operator on the encoder output because this term is *only* intended to *update* the *codebook*.

VQ-VAE Loss Function

- Loss Function:

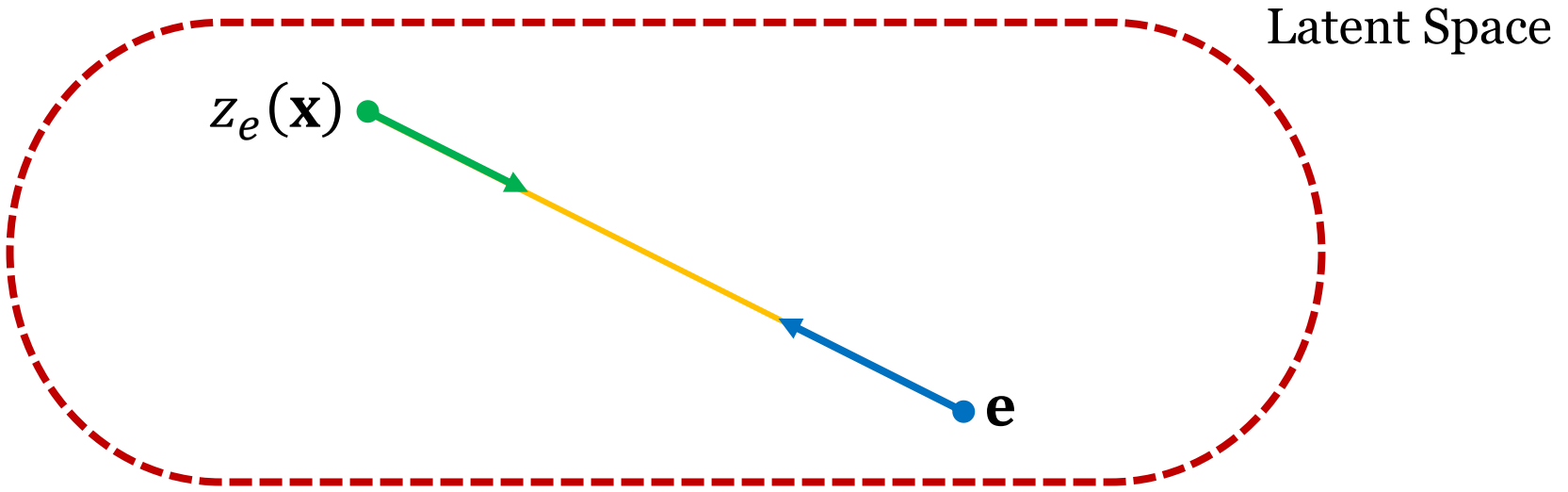
$$\mathcal{L}\big(\mathbf{x}, Dec(\mathbf{e})\big) = \|\mathbf{x} - Dec(\mathbf{e})\|_2^2 + \|sg[z_e(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|sg[\mathbf{e}] - z_e(\mathbf{x})\|_2^2$$

- Third term: It is meant to solve the inverse problem of getting the *encoder output* to commit as much as possible to its *closest codebook* vector. (commitment loss)
- These last two terms are averaged over each quantized vector output from the model, if there is more than one.

VQ-VAE Loss Function

- Loss Function:

$$\mathcal{L}\big(\mathbf{x}, Dec(\mathbf{e})\big) = \|\mathbf{x} - Dec(\mathbf{e})\|_2^2 + \|sg[z_e(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|sg[\mathbf{e}] - z_e(\mathbf{x})\|_2^2$$



VQ-VAE Dictionary Learning

- VQ-VAE dictionary updates with Exponential Moving Averages (EMA).
- This module used instead of second term in loss function:

$$\|sg[z_e(\mathbf{x})] - \mathbf{e}\|_2^2$$

- Let $\{z_{i,k}\}_{k=1}^{n_i}$ be the set of n_i outputs from the encoder that are closest to dictionary codebook e_i, best estimation for e_i is (k-means updates):

$$e_i = \frac{1}{n_i}\sum_{k=1}^{n_i} z_{i,k}$$

- However, we cannot use this update directly when working with minibatches.

VQ-VAE Dictionary Learning

- Let use online update!
- $N_i^{(t)} = \gamma N_i^{(t-1)} + (1-\gamma) n_i^{(t)}$
- $m_i^{(t)} = \gamma m_i^{(t-1)} + (1-\gamma) \sum_{k=1}^{n_i^{(t)}} z_{i,k}^{(t)}$
- $e_i^{(t)} = \frac{m_i^{(t)}}{N_i^{(t)}}$
- $\gamma \cong 0.99$

VQ-VAE

ImageNet $128 \times 128 \times 3$ images

Reconstructions from a VQ-VAE with a $32 \times 32 \times 1$ latent space, and $K = 512$

VQ-VAE 2

- Train a multi level hierarchical VQ-VAE to encode images onto a discrete latent space.
- Fit a multi-stage powerful image generation using multi-headed self-attention .



Any Question