Deep Unsupervised AutoEncoder Variational Auto Encoder

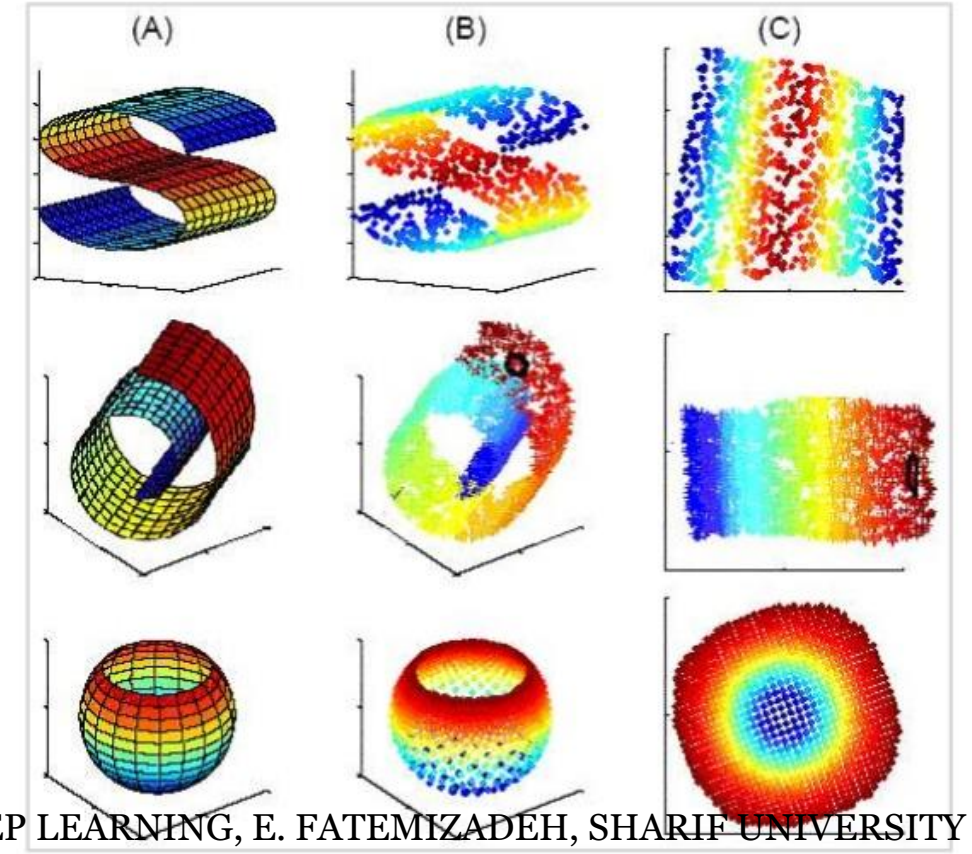
Deep Learning Course
Sharif University of technology
Fall 2021
E. Fatemizadeh

Deep Unsupervised

- Main Area:
  - Dimensionality Reduction
  - Clustering
  - Probability Density Estimation

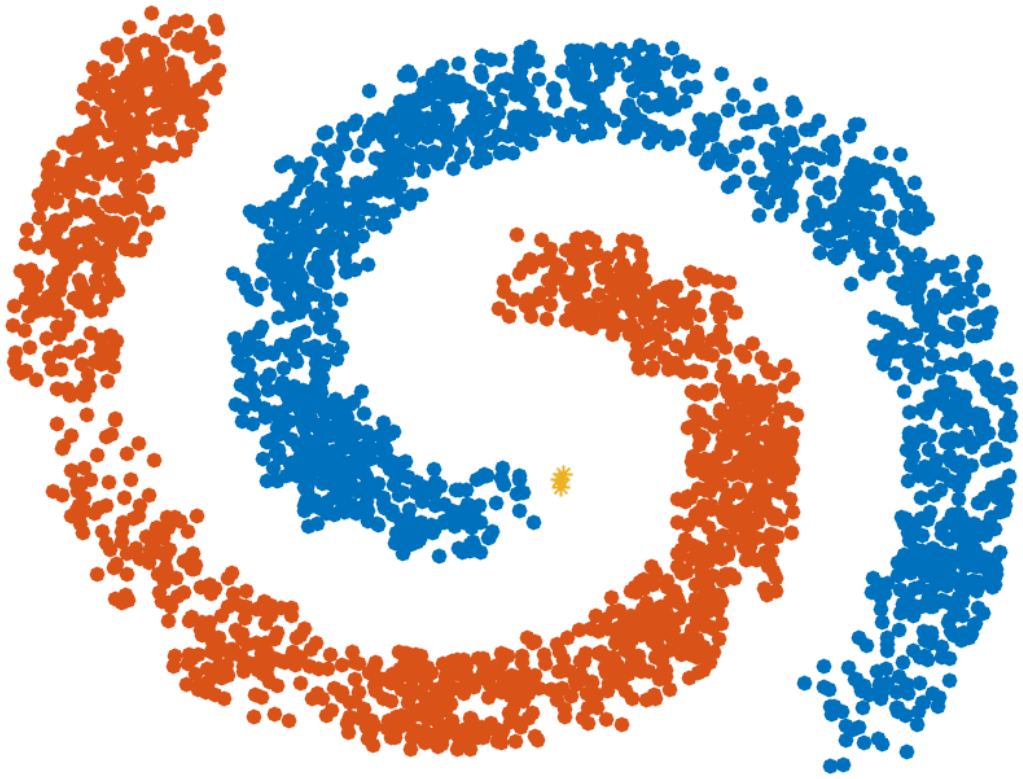
Dimensionality Reduction

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



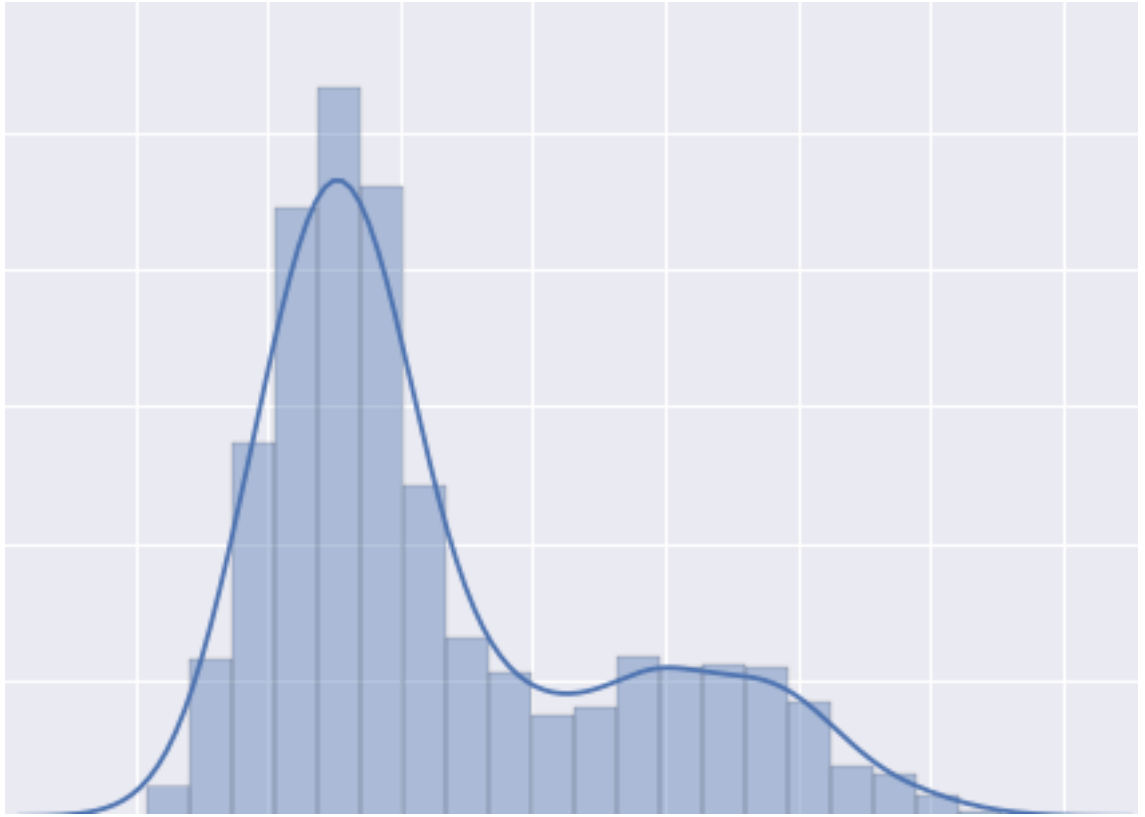
Clustering

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



Probability Density Estimation

- Model pdf:
  - Parametric
  - Nonparametric

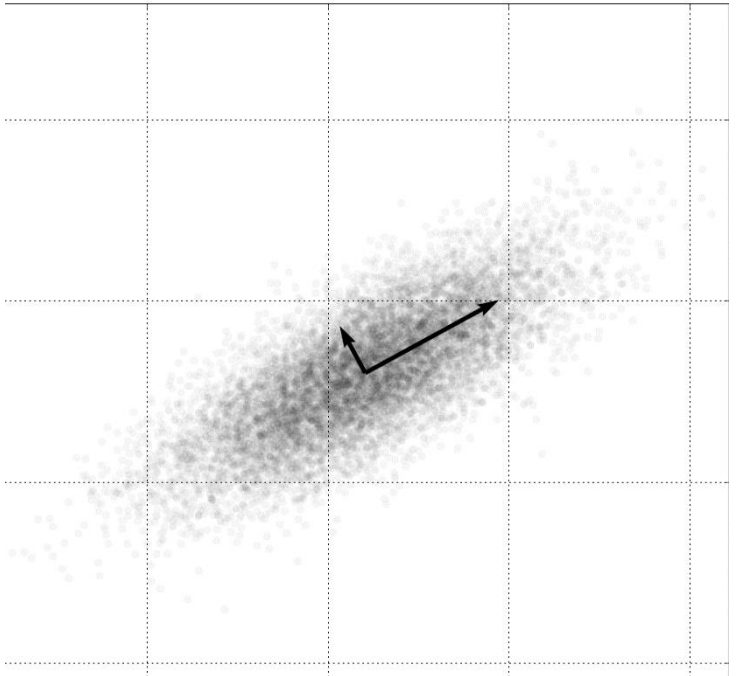


Dimensionality Reduction

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

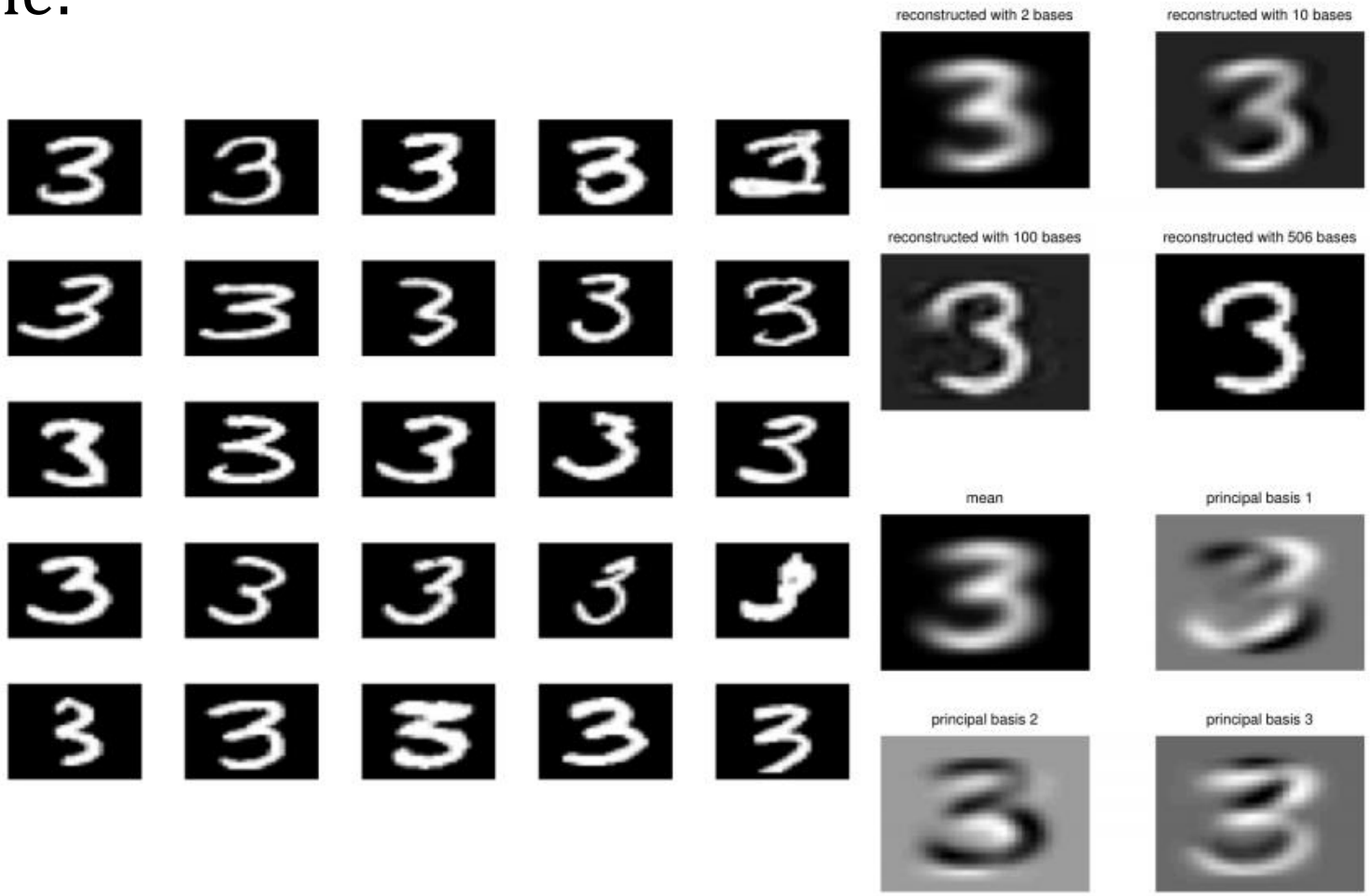
Dimensionality Reduction

- PCA:
  - $Y=A^T(X-m_x)$ or $Y = A^TX$, Y: uncorrelated (diagonal covariance matrix)
  - $X=AY+m_x$ or $X=AY$,



Dimensionality Reduction

› PCA Example:



Dimensionality Reduction

- PCA:
  - $Y=A^T(X-m_x)$ or $Y = A^TX$, Y: uncorrelated (diagonal covariance matrix)
  - $X=AY+m_x$ or $X=AY$,
- Dimensionality Reduction:
  - $\hat{X} = A_k Y_k$

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

Dimensionality Reduction

- PCA Optimization Formulation:
- X: input training matrix ($N \times d$)

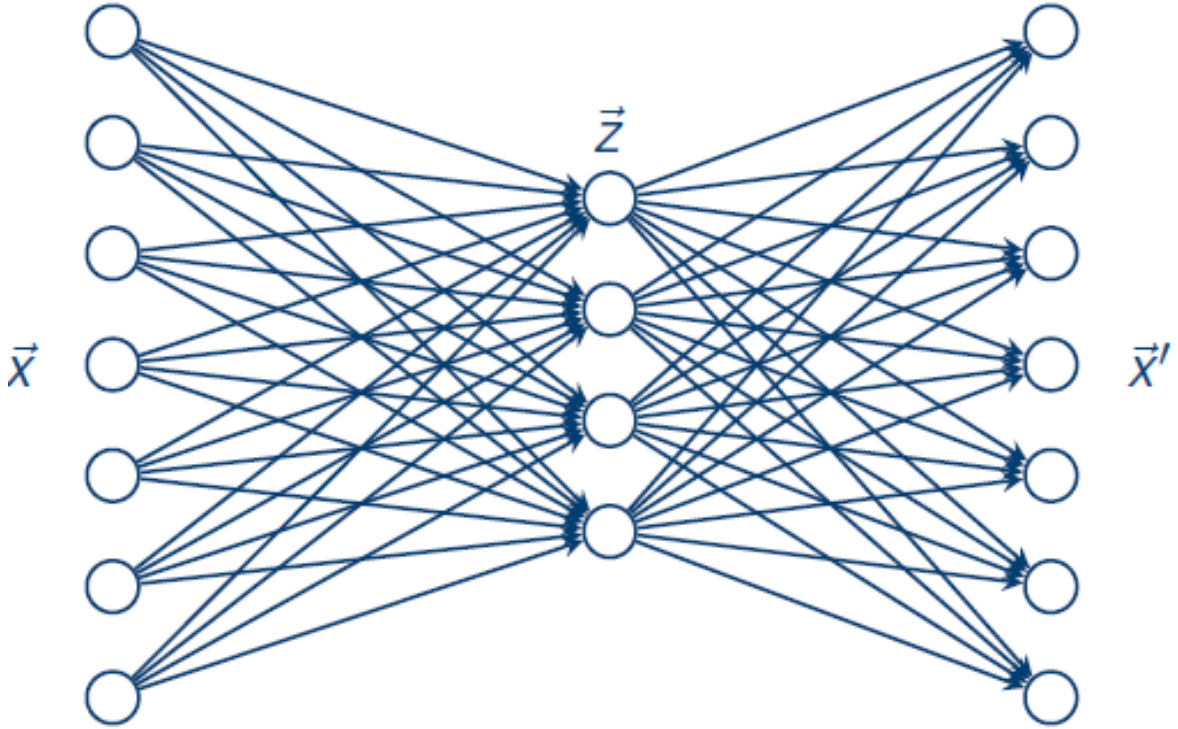
$$\min_{W^T W = I} \|X - (XW)W^T\|_F^2 = \|X - \hat{X}\|_F^2$$

$$W \in \mathbb{R}^{d \times k}$$

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

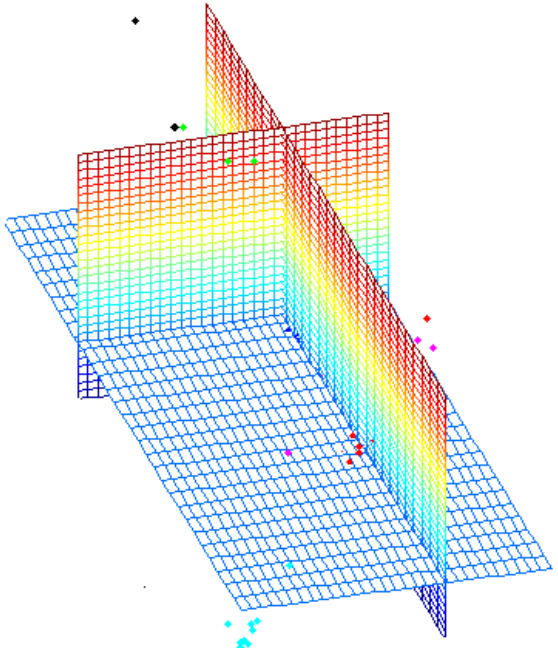
Dimensionality Reduction

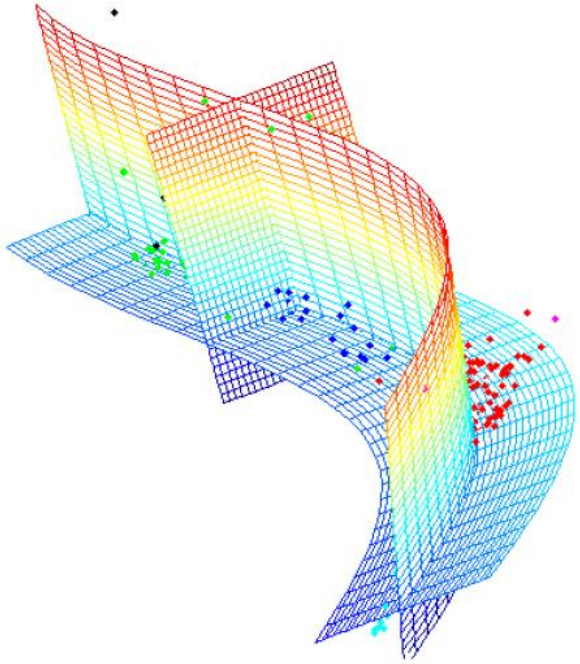
- PCA ~ Two Layer Perceptron
  - Constrain on weights!



Dimensionality Reduction

- PCA Limitation:
  - Only Capture **LINEAR** dependency!





Dimensionality Reduction

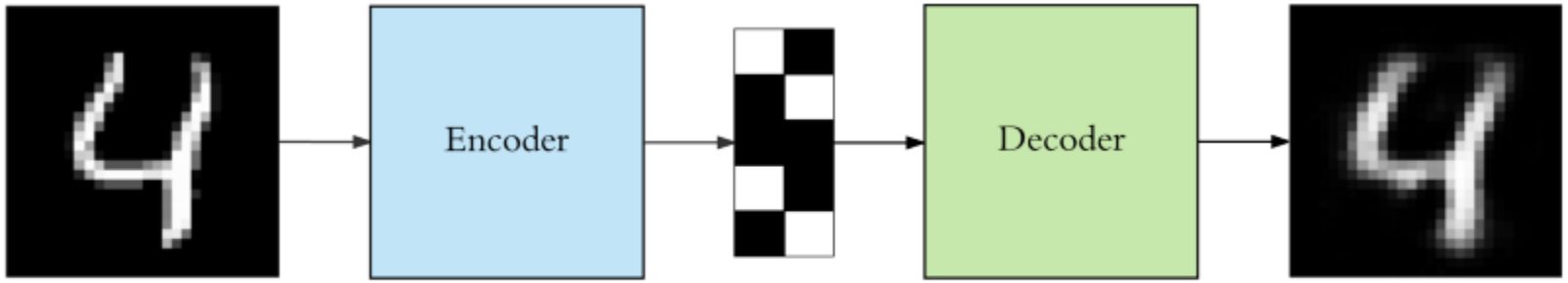
› A conceptual generalization

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

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

AutoEncoder

› AutoEncoder idea:

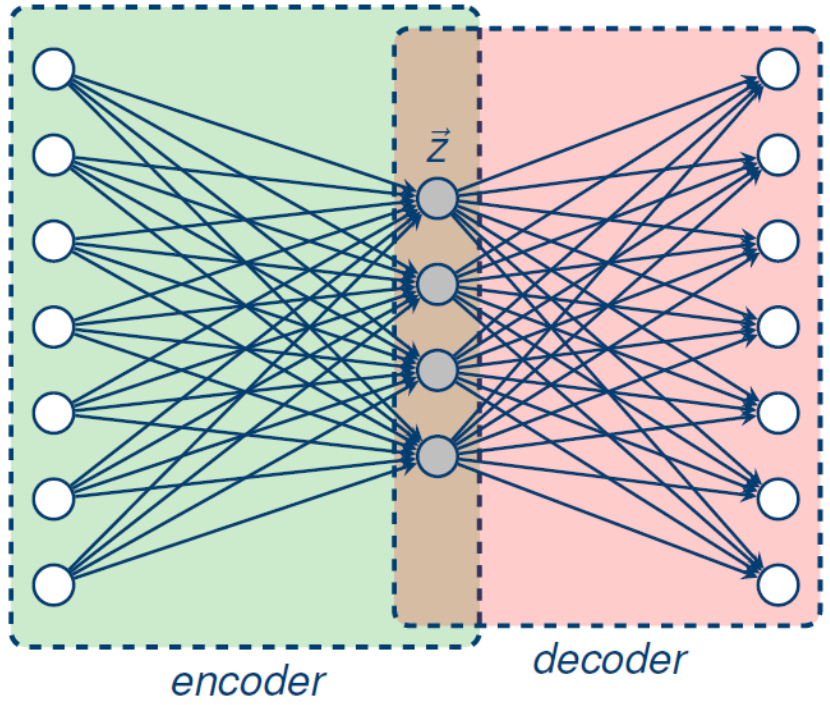


$$f(XU) \qquad g(f(XU)V)$$

AE (AutoEncoder)

- AutoEncoder (Simple One):
  - Map $x \in \mathbb{R}^D$ to Nonlinear **Low** Dimensional Manifold, $z \in \mathbb{R}^d$ (Encoder)
  - Map $z \in \mathbb{R}^d$ to Nonlinear **Original** Dimensional Manifold $\hat{x} \in \mathbb{R}^D$ (Decoder)
  - $z_{out}^{(d)} = x_{in}$

$$\mathbf{h} = \sigma(\mathcal{W}_v \mathbf{x} + \mathbf{b}).$$



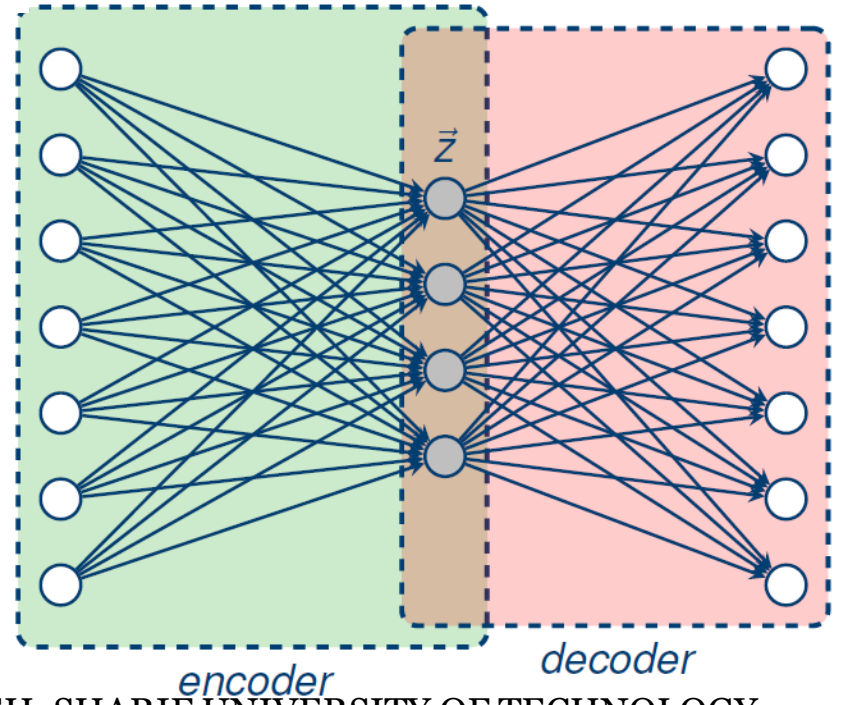
$$\mathbf{z} = \delta(\mathcal{W}_h \mathbf{h} + \mathbf{b}_h)$$

AE (AutoEncoder)

- AutoEncoder (Simple One):
- Some times (for simplification):
  - $W_v = W_h^T$

$$\mathbf{h} = \sigma(\mathcal{W}_v \mathbf{x} + \mathbf{b}).$$

$$\mathbf{z} = \delta(\mathcal{W}_h \mathbf{h} + \mathbf{b}_h)$$



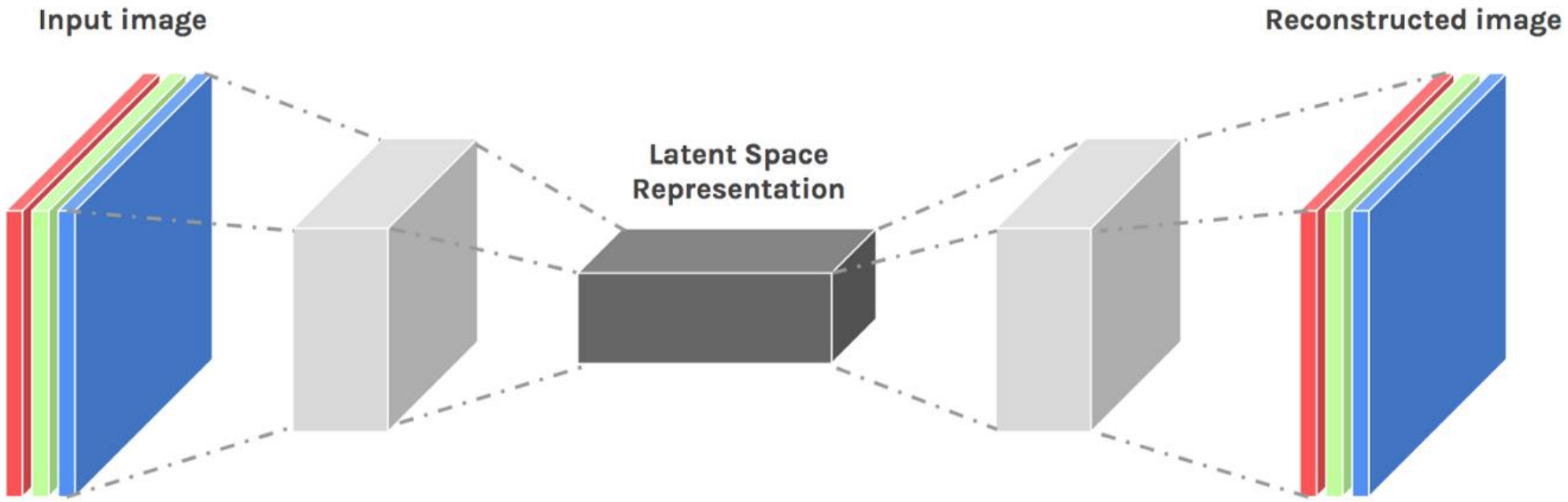
- Loss:
  - MSE or CrossEntropy (σ)

$$\mathcal{J}(\mathcal{W}, \mathbf{b}_v, \mathbf{b}_h) = \mathcal{L}(\mathbf{x}, \mathbf{z}) + \lambda g(\mathcal{W}),$$

$$g(\mathcal{W}) = 0.5\left(\|\mathcal{W}_v\|_F^2 + \|\mathcal{W}_h\|_F^2\right)$$

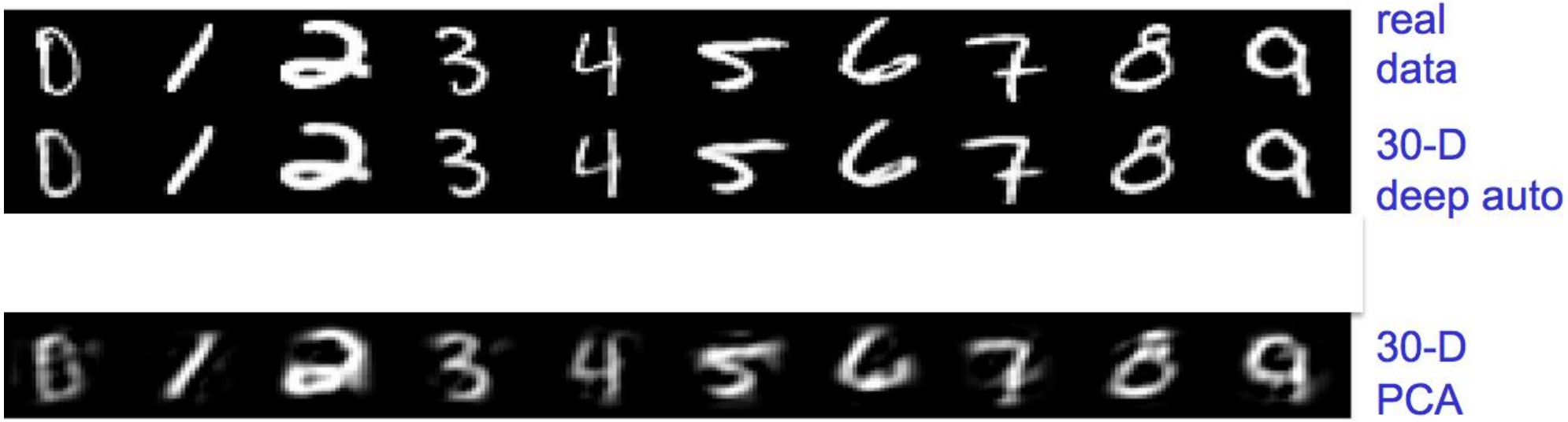
AE (AutoEncoder)

› General (a bit) AE:



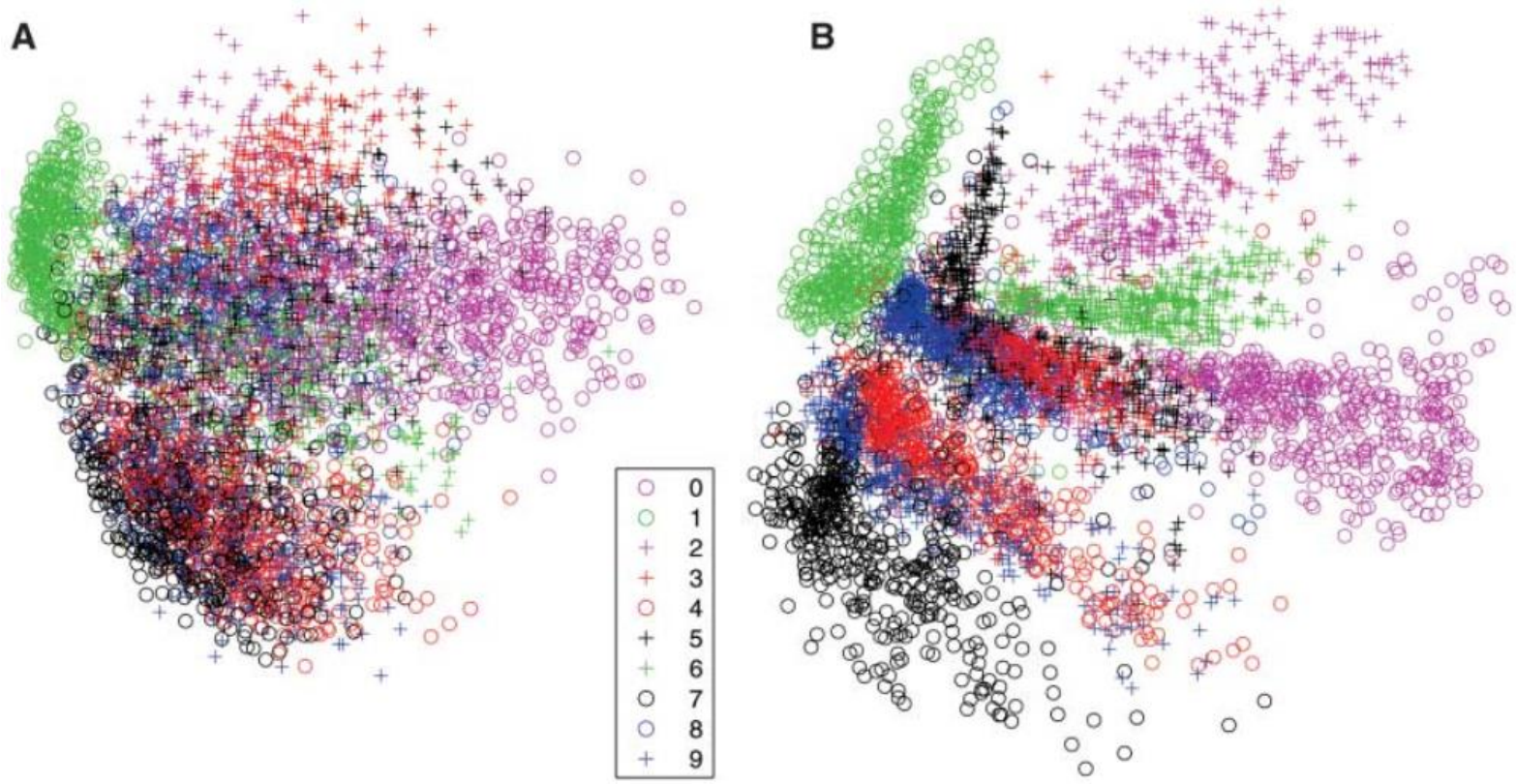
Dimensionality Reduction

› AutoEncoder/PCA Example:



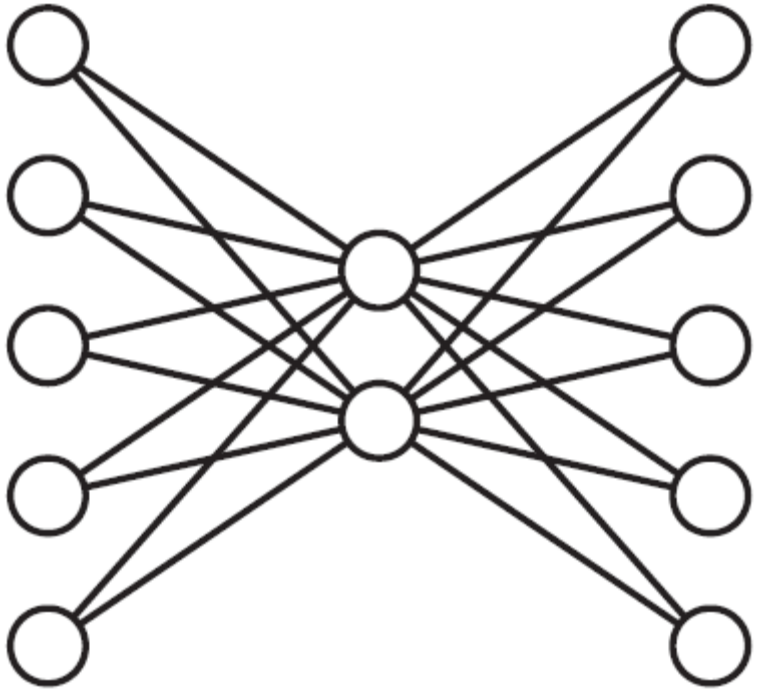
Dimensionality Reduction

› AutoEncoder (Right)/PCA (Left) Example:



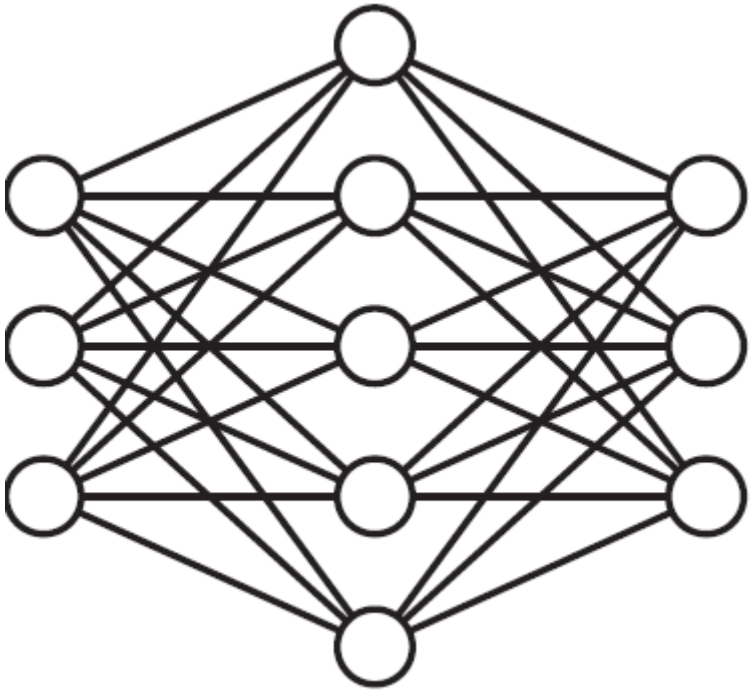
AE Architectures

› Shallow UnderComplete:



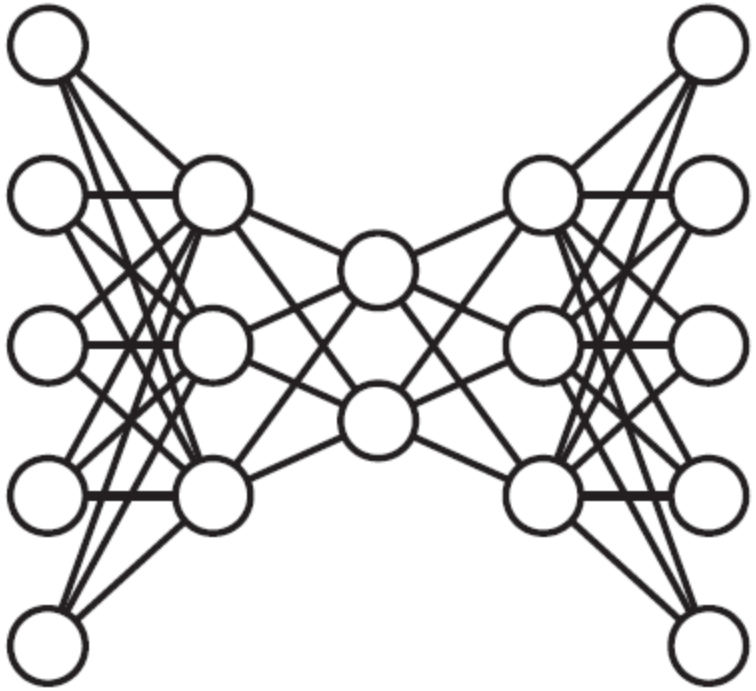
AE Architectures

› Shallow OverComplete:



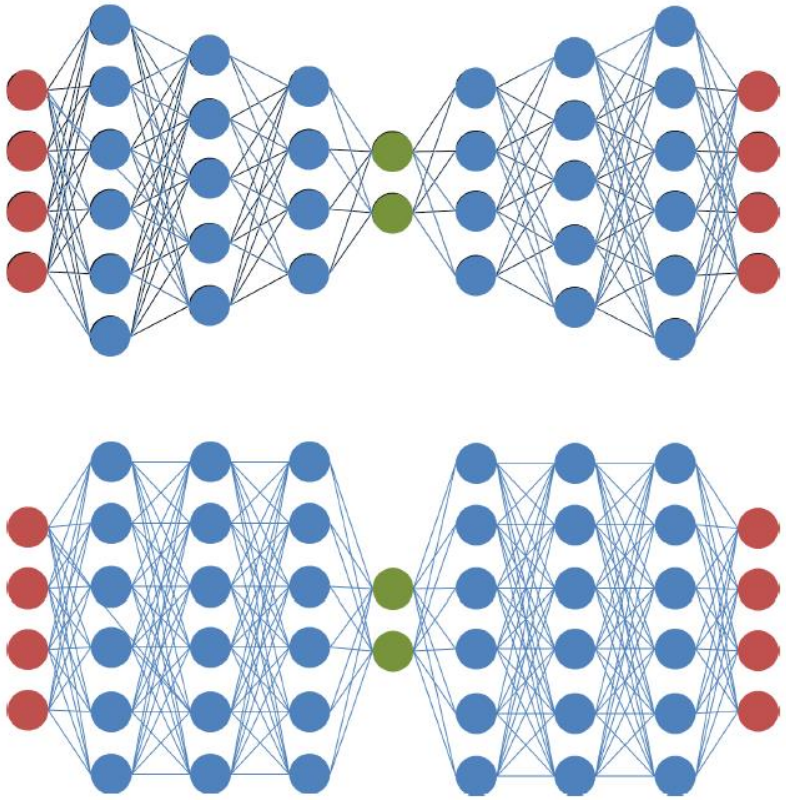
AE Architectures

› Deep UnderComplete:



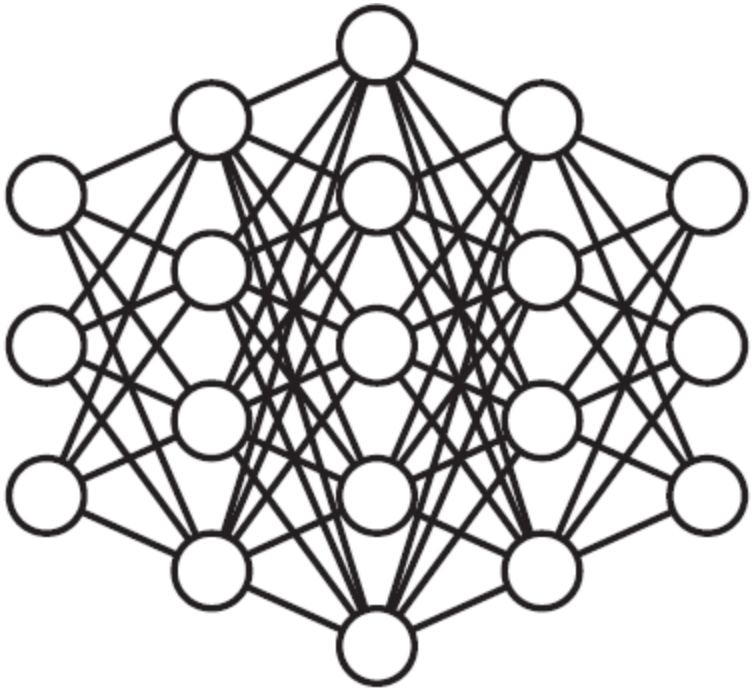
AE Architectures

› Deep UnderComplete:



AE Architectures

› Deep OverComplete:



AE (AutoEncoder)

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

AE (AutoEncoder)

- Convolutional AE (CAE)
  - Image friendly, weight sharing!
  - *k*-th feature map:

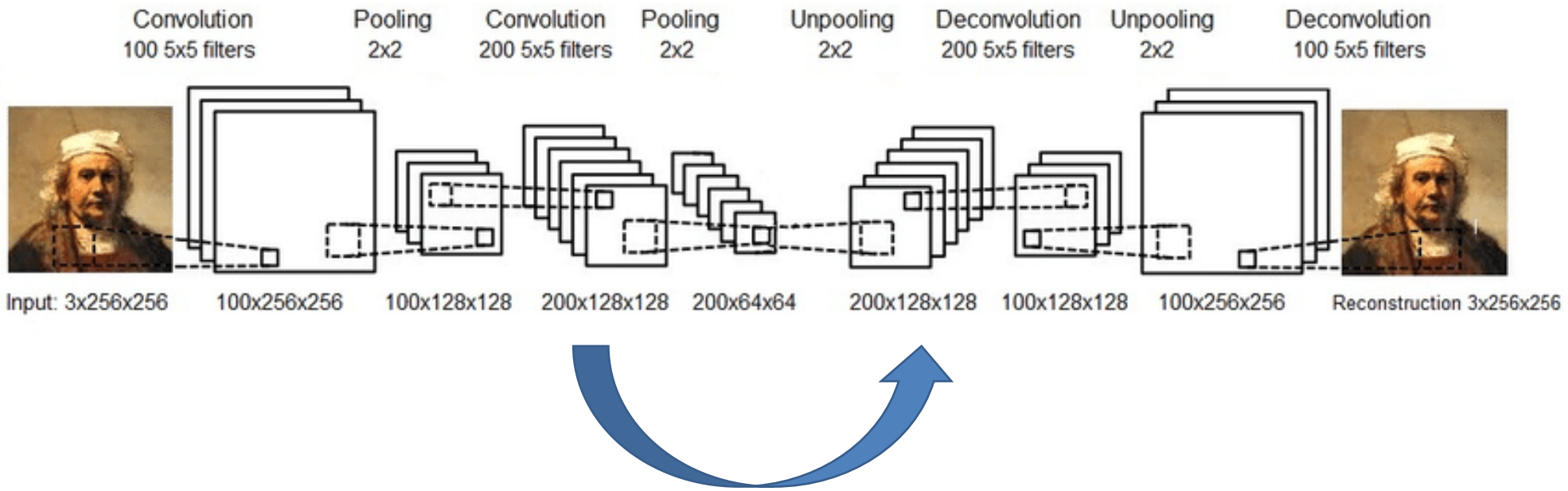
$$h^k = \sigma(\mathrm{x} * \mathrm{W}^{\mathrm{k}} + b^k)$$

  - Reconstructed output:

$$y = \sigma(\sum_{k \in H} \mathrm{h}^{\mathrm{k}} * \tilde{\mathrm{W}}^{\mathrm{k}} + c)$$

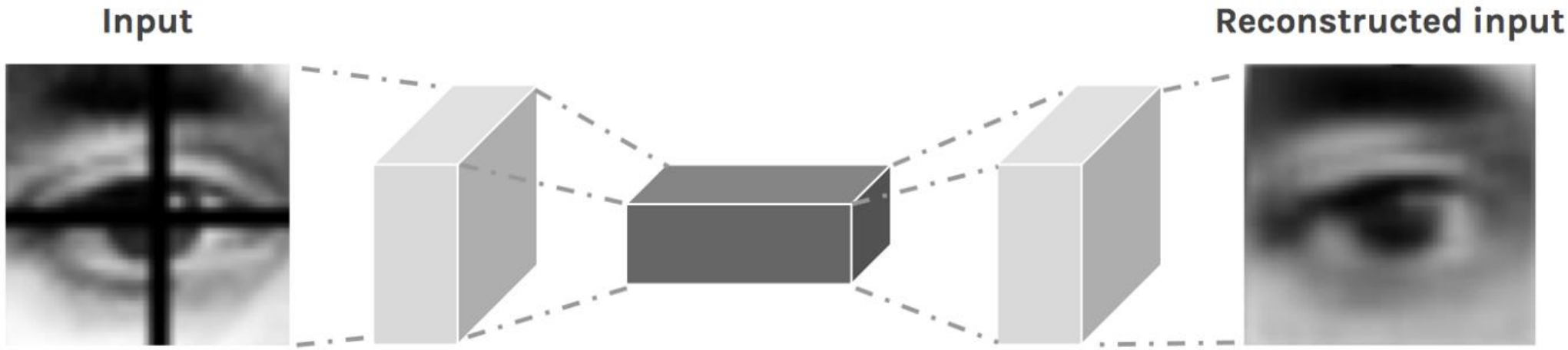
AE (AutoEncoder)

› Convolutional AE (CAE):



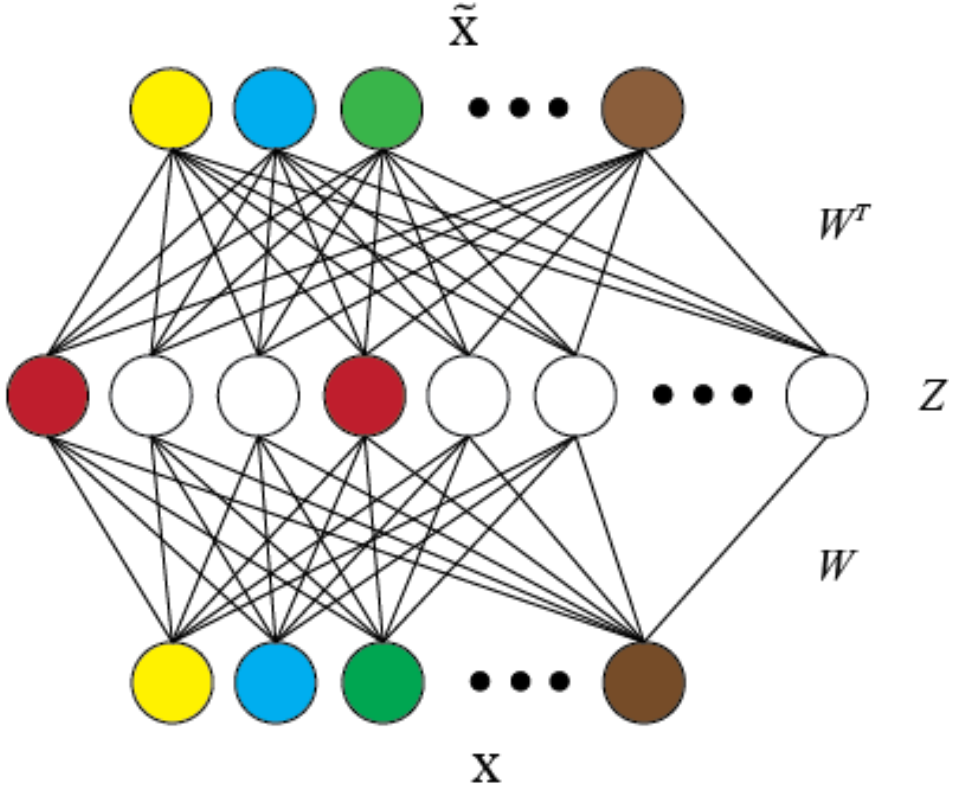
AE (AutoEncoder)

- Convolutional AE (CAE)
  - Visual features
  - Blurry/low quality outputs



AE (AutoEncoder)

› Sparse AE (SAE)
  – *OverComplete* structure and add sparsity term in Loss



AE (AutoEncoder)

- Sparsity Loss:
  - Average of activation in hidden layer:
  - S: Batch Set, i: Neuron index $\hat{\rho}_i = \frac{1}{|S|} \sum_{x \in S} f_i(x)$
  - ρ=acceptable sparsity (0.05)

$$\mathrm{KL}(\rho \| \hat{\rho}_i) = \rho \log \frac{\rho}{\hat{\rho}_i} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_i}$$

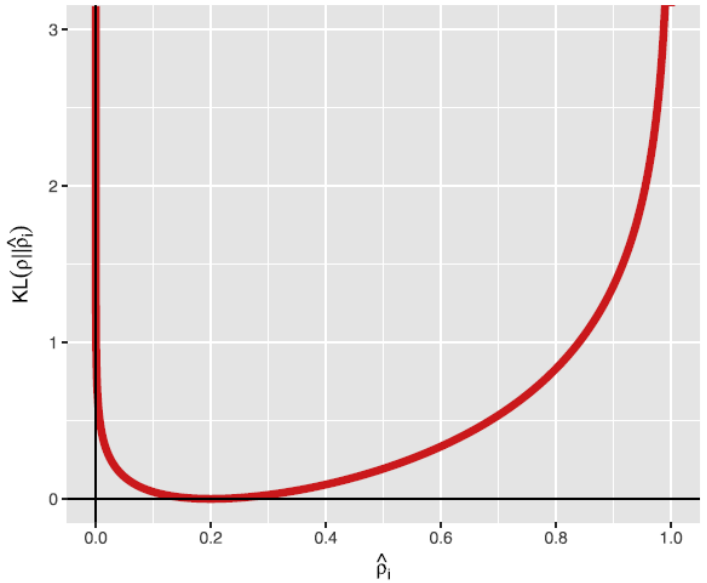
$$\Omega_{\mathrm{SAE}}(W, b; S) = \sum_{i=1}^{c} \mathrm{KL}(\rho \| \hat{\rho}_i),$$

  - c: Hidden layer size

AE (AutoEncoder)

› Sparsity Loss (for ρ=0.2):

$$\mathrm{KL}(\rho||\hat{\rho}_i) = \rho \log \frac{\rho}{\hat{\rho}_i} + (1-\rho)\log \frac{1-\rho}{1-\hat{\rho}_i}$$



AE (AutoEncoder)

› Overall Loss:

$$J_{SAE} = J(W) + \beta \sum_{i=1}^{c} KL(\rho \parallel \rho_i)$$

AE (AutoEncoder) - k-Sparse Autoencoders, 2013

› K-sparse AE:
  – Keep only *k-largest* active neuron in hidden layer (EBP only through these)
  – $\alpha \geq 1$

k-Sparse Autoencoders:

Training:

1) Perform the feedforward phase and compute

$$\boldsymbol{z} = W^\top \boldsymbol{x} + \boldsymbol{b}$$

2) Find the k largest activations of $\boldsymbol{z}$ and set the rest to zero.

$$\boldsymbol{z}_{(\Gamma)^c} = 0 \quad \text{where} \quad \Gamma = \text{supp}_k(\boldsymbol{z})$$

3) Compute the output and the error using the sparsified $\boldsymbol{z}$.

$$\hat{\boldsymbol{x}} = W\boldsymbol{z} + \boldsymbol{b}'$$
$$E = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2$$

3) Backpropagate the error through the k largest activations defined by Γ and iterate.

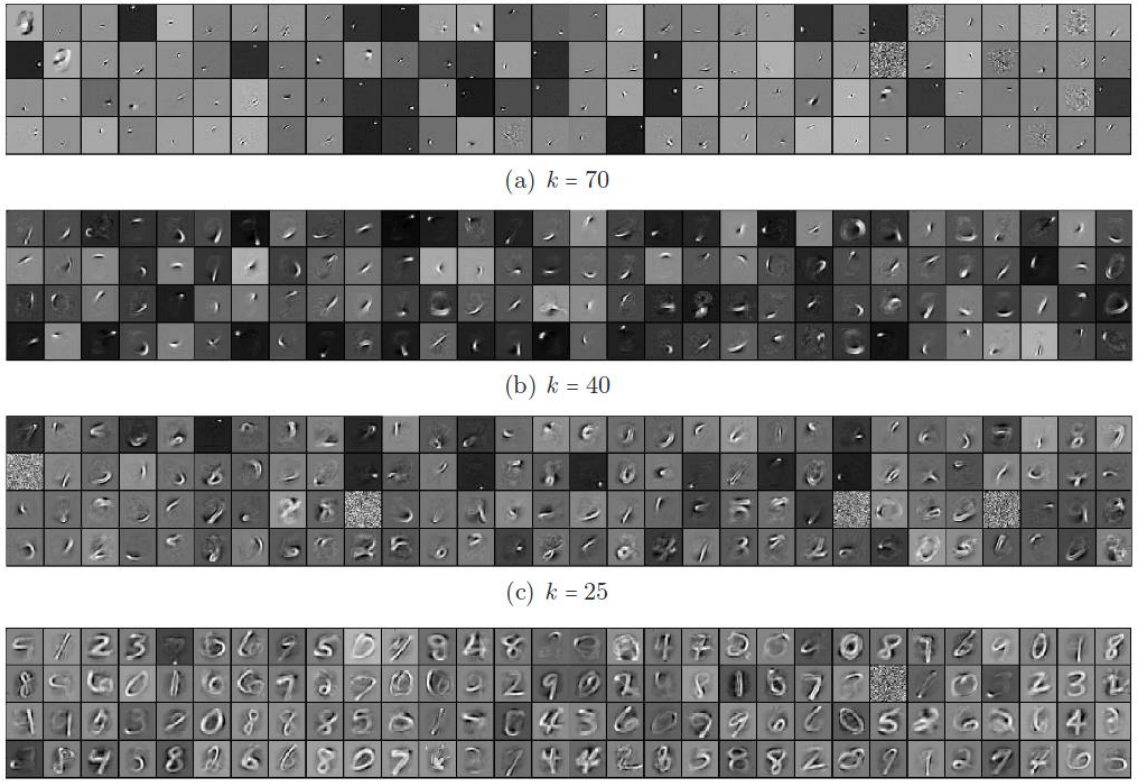
Sparse Encoding:

Compute the features $\boldsymbol{h} = W^\top \boldsymbol{x} + \boldsymbol{b}$. Find its αk largest activations and set the rest to zero.

$$\boldsymbol{h}_{(\Gamma)^c} = 0 \quad \text{where} \quad \Gamma = \text{supp}_{\alpha k}(\boldsymbol{h})$$

AE (AutoEncoder)

› K-sparse AE:
  – Keep only *k*-largest active neuron in hidden layer (backprop. only these)



(a) $k = 70$

(b) $k = 40$

(c) $k = 25$

AE (AutoEncoder) - Contractive Auto-Encoders: Explicit Invariance During Feature Extraction, ICML2011

› Contractive AE:

Make the learned representation to be **robust** towards **small** changes around the **training** examples

› Loss + Sensitivity term:

$$\mathcal{J}_{\mathrm{CAE}}(\theta) = \sum_{x \in D_n} \left(L(x, g(f(x))) + \lambda \| J_f(x) \|_F^2 \right)$$

$$\| J_h(X) \|_F^2 = \sum_{ij} \left(\frac{\partial h_j(X)}{\partial X_i} \right)^2$$

› Encourages the mapping to the feature space to be contractive in the neighborhood of the training data.

AE (AutoEncoder) - Contractive Auto-Encoders: Explicit Invariance During Feature Extraction, ICML2011

› Contractive AE:

$$Z_j = W_i X_i$$

$$h_j = \phi(Z_j)$$

$$\frac{\partial h_j}{\partial X_i} = \frac{\partial \phi(Z_j)}{\partial X_i}$$

$$= \frac{\partial \phi(W_i X_i)}{\partial W_i X_i} \frac{\partial W_i X_i}{\partial X_i}$$

$$= [\phi(W_i X_i)(1 - \phi(W_i X_i))] W_i$$

$$= [h_j(1 - h_j)] W_i$$

$$\frac{\partial h}{\partial X} = diag[h(1-h)] W^T$$

AE (AutoEncoder) - Contractive Auto-Encoders: Explicit Invariance During Feature Extraction, ICML2011

› Contractive AE:

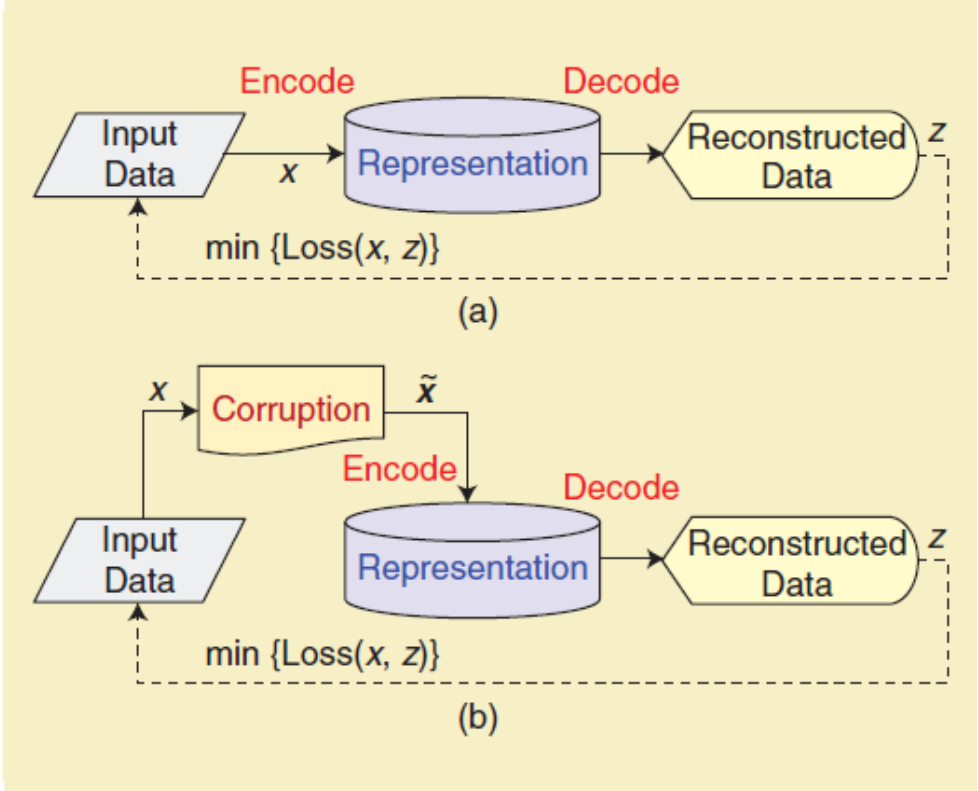
$$\|J_h(X)\|_F^2 = \sum_{ij}\left(\frac{\partial h_j}{\partial X_i}\right)^2$$

$$= \sum_i \sum_j [h_j(1-h_j)]^2 (W_{ji}^T)^2$$

$$= \sum_j [h_j(1-h_j)]^2 \sum_i (W_{ji}^T)^2$$

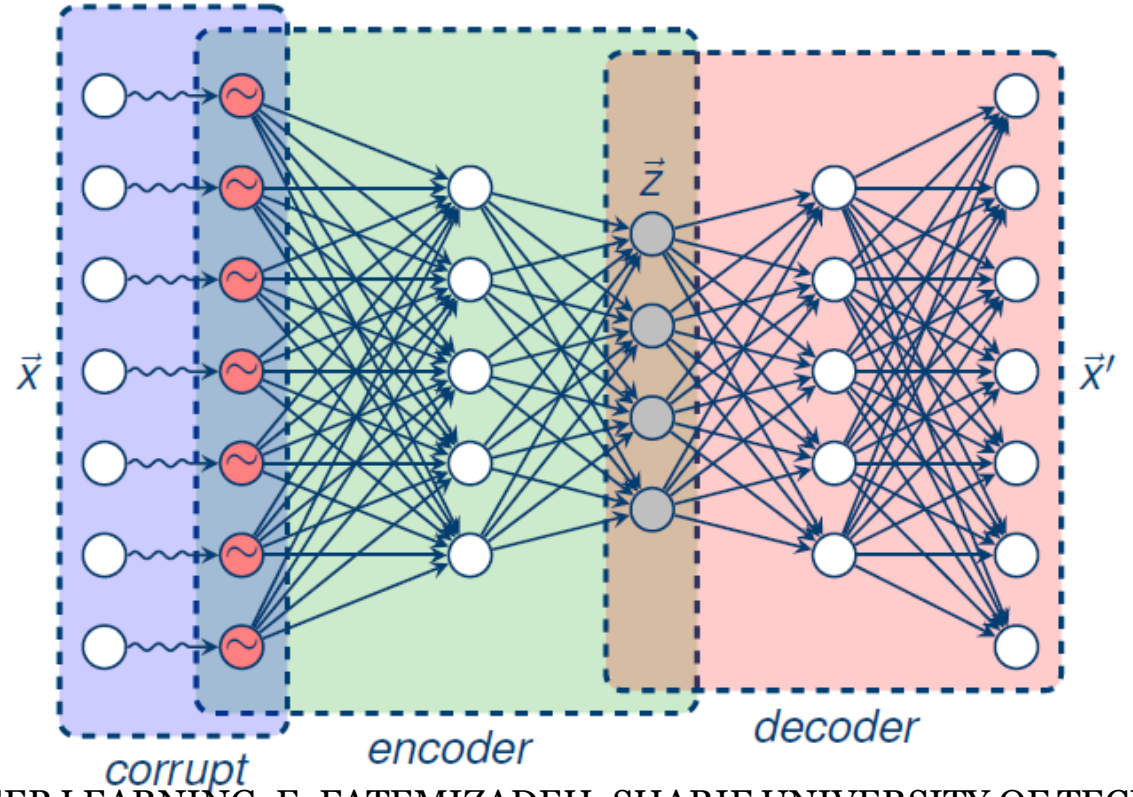
AE (AutoEncoder)

› Denoising AE:
  – Learn robust feature to able reconstruct data from an input of corrupted data



AE (AutoEncoder)

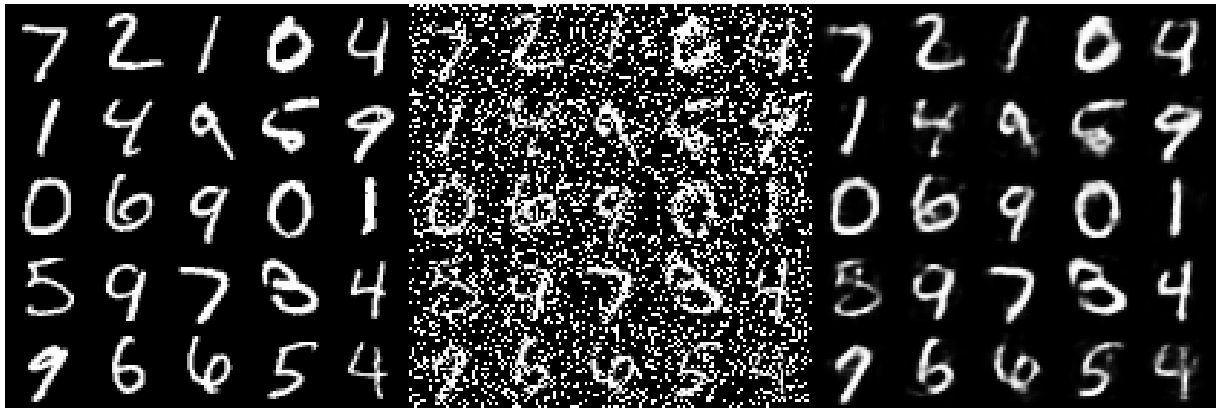
› Denoising AE:
  – Learn robust feature to able reconstruct data from an input of corrupted data



AE (AutoEncoder)

› Denoising AE:

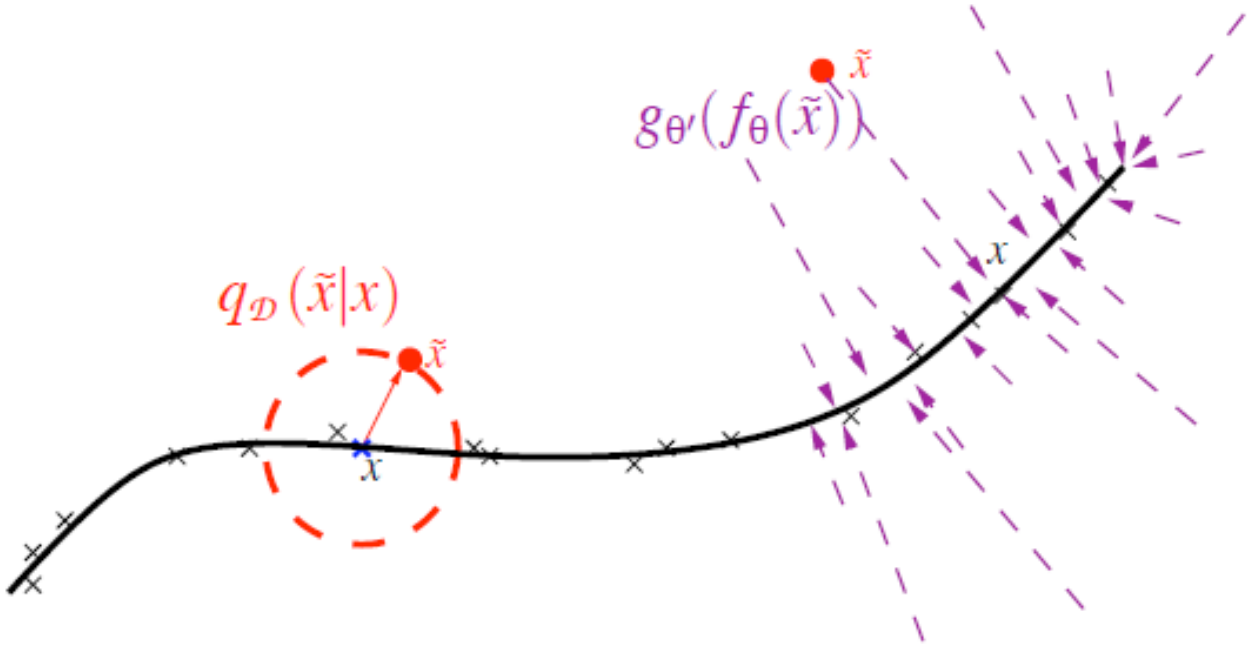
› Corrupted data (target-input-output)



AE (AutoEncoder)

› Denoising AE:

› Manifold representation:

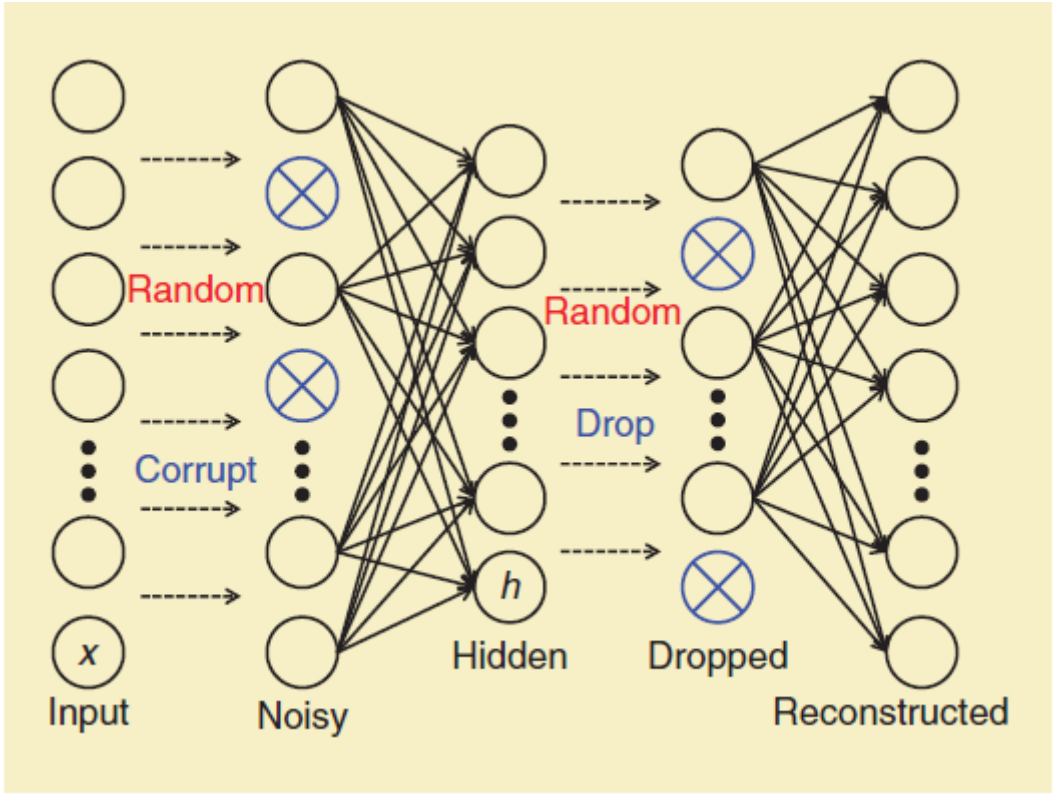


AE (AutoEncoder)

- Denoising AE:
- How to corrupt data:
  - Add white Gaussian noise to data $\sim N(0,\sigma^2\mathbf{I})$
  - 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 (zero or one)

AE (AutoEncoder)

› Denoising AE:
  – May be merged with dropout strategy:



AE (AutoEncoder) - Robust feature learning by stacked autoencoder with maximum correntropy criterion, ICASSP 2014

› Robust AE:

› Use "Correntropy: A Localized Similarity Measure" as loss
  – Robust to outlier,
  – MCC: **M**aximum **C**orrEntropy **C**riterion

$$\mathscr{L}_{\mathrm{MCC}}(u, v) = -\sum_{k=1}^{d} \mathscr{K}_{\sigma}(u_k - v_k),$$

$$\text{where } \mathscr{K}_{\sigma}(\alpha) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\alpha^2}{2\sigma^2}\right),$$

AE (AutoEncoder)

› Stacked AutoEncoder (SAE):
  – Structure is Deep, but layer-by-layer learning!
  – 1st layer to nth layer: simple (low level) to complex (high level) features
  – For kth layer:

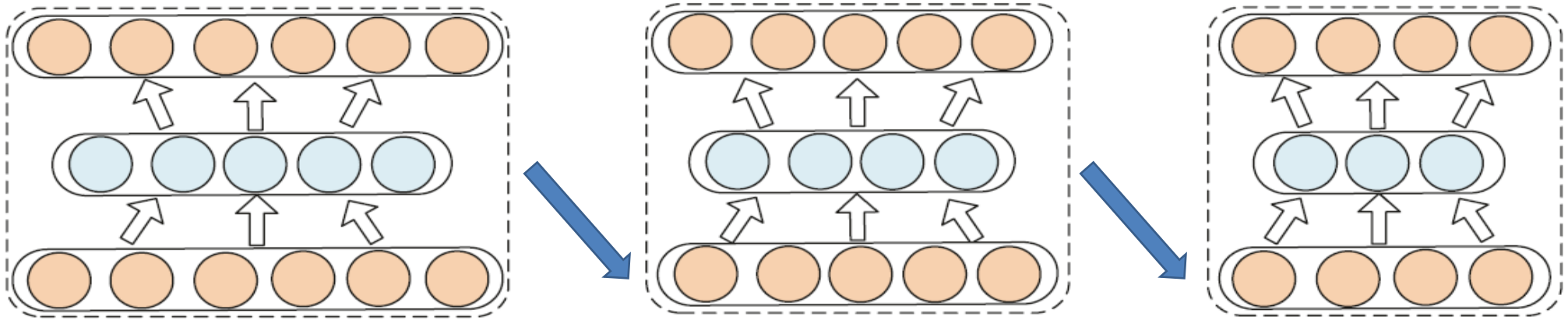
$$\mathbf{h}^{(k)} = \sigma\left(\mathbf{z}^{(k)}\right)$$
$$\mathbf{z}^{(k+1)} = \mathcal{W}_v^{(k)}\mathbf{h}^{(k)} + \mathbf{b}_v^{(k)},$$
$$\mathbf{z}^{(k)} = \mathbf{x}^{(k-1)}$$

AE (AutoEncoder)

- Stacked AutoEncoder (SAE):
- Train the 1st 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.

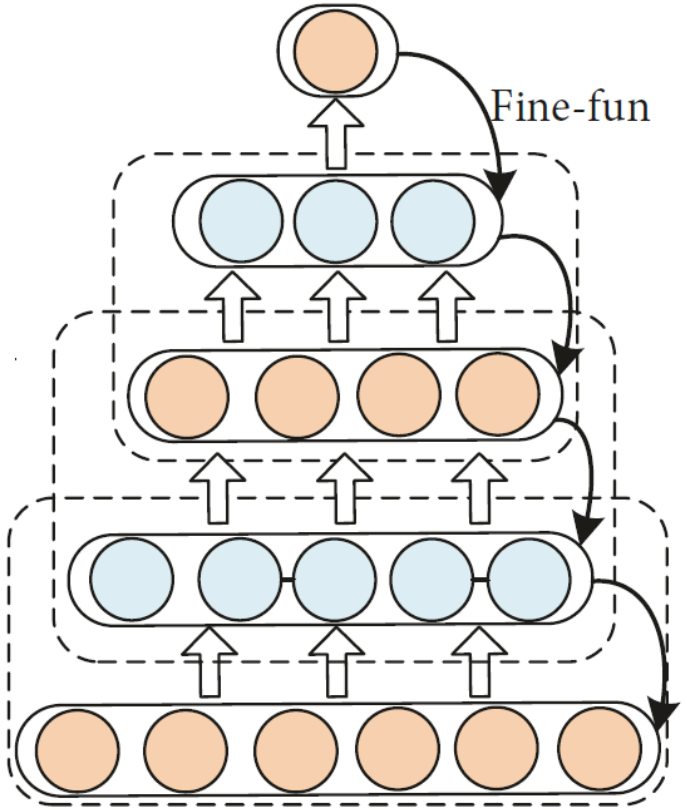
AE (AutoEncoder)

› Stacked AutoEncoder (SAE):



AE (AutoEncoder)

› Stacked AutoEncoder (SAE):

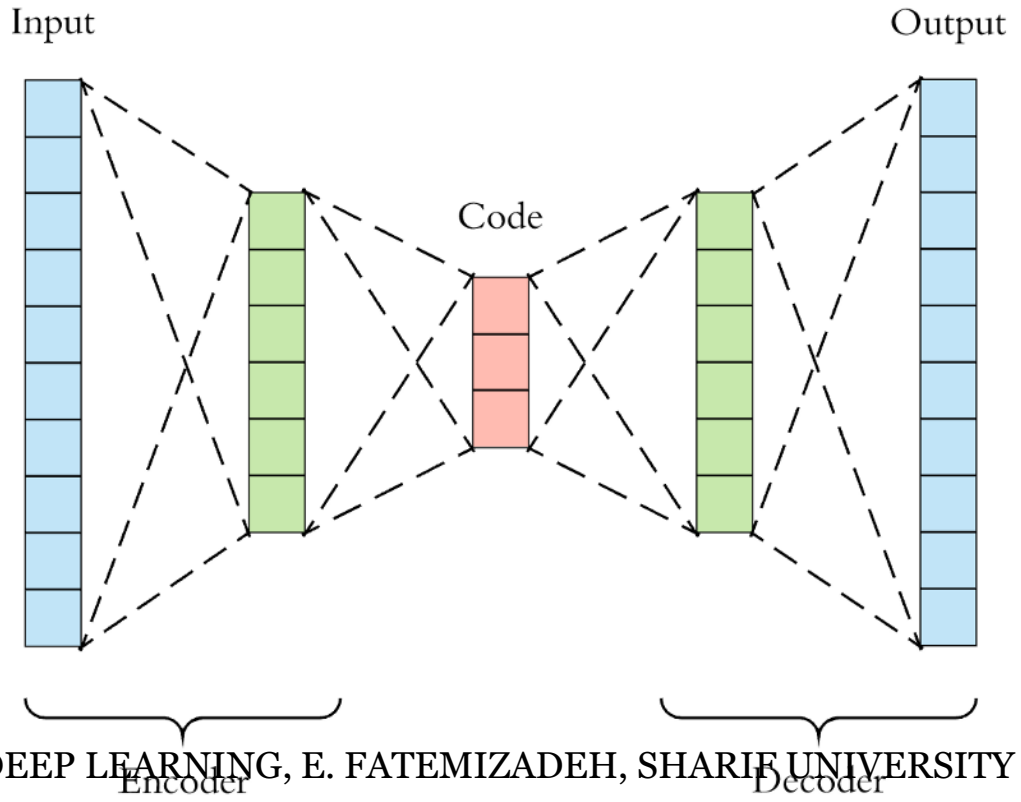


AE (AutoEncoder)

- Stacked Denoising AutoEncoder (SDA):
- Stacked Convolutional AuroEncoder (SCA)

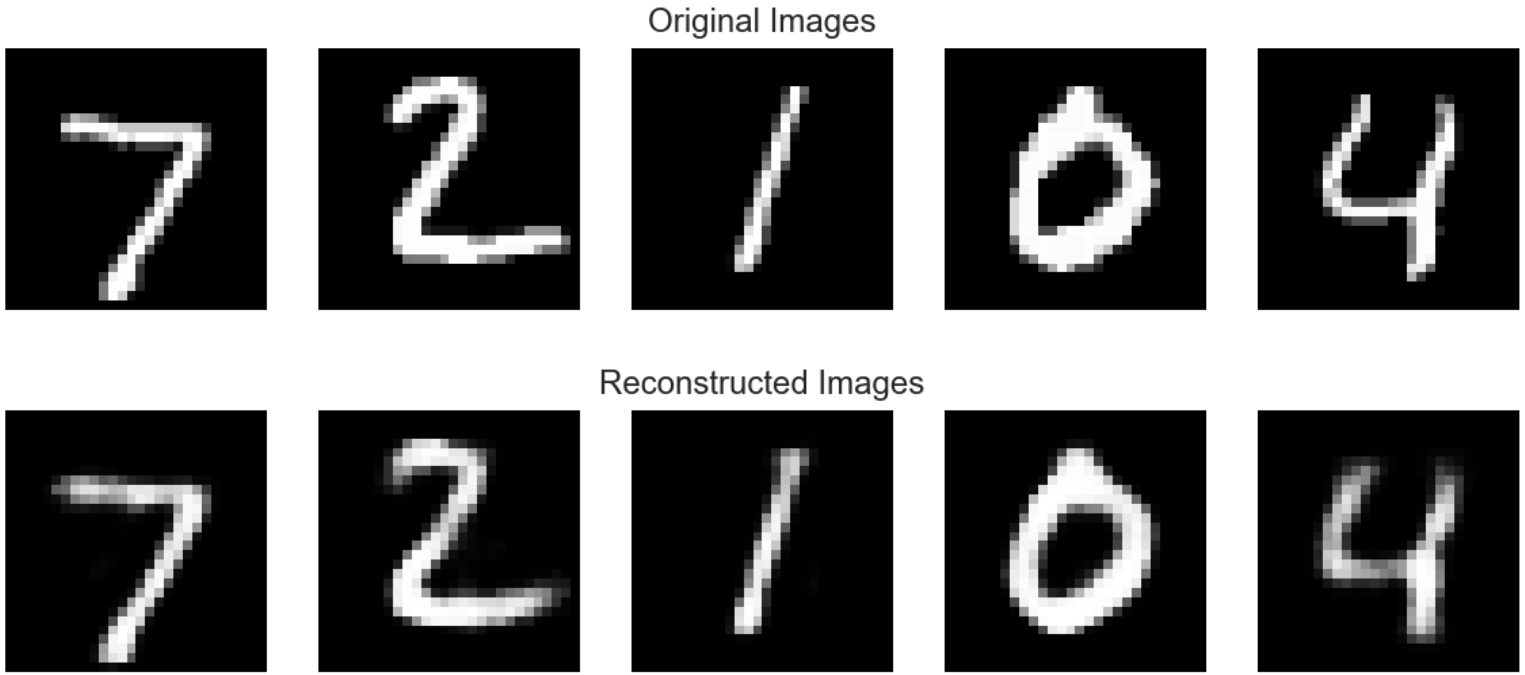
AE (AutoEncoder) – Examples and Applications

› Example #1:
  – Input: 28×28 (784D), hidden (100D), codesize (32D):



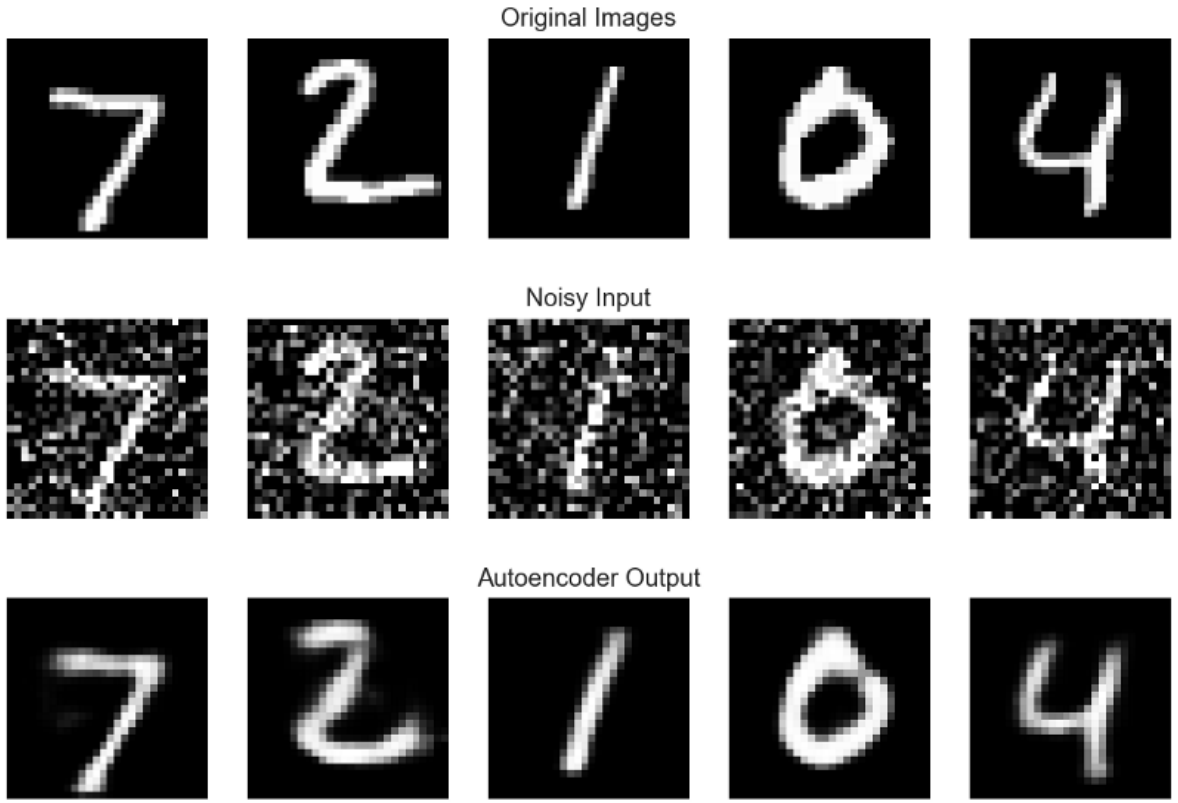
AE (AutoEncoder) – Examples and Applications

› Example #1: Feature Reduction
  – Input: 28×28 (784D), hidden (100D), codesize (32D):



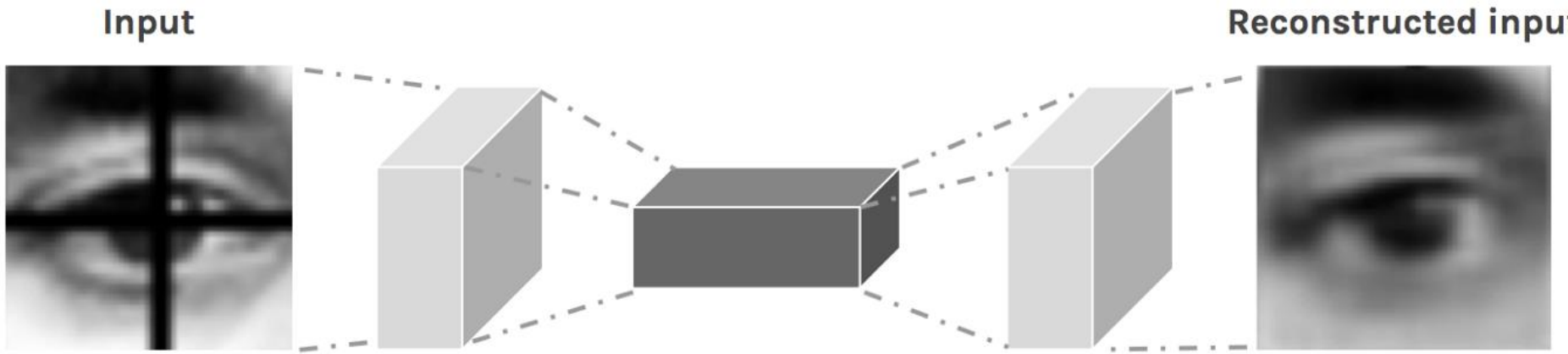
AE (AutoEncoder) – Examples and Applications

› Example #2: Denoising AE
  – Same setting



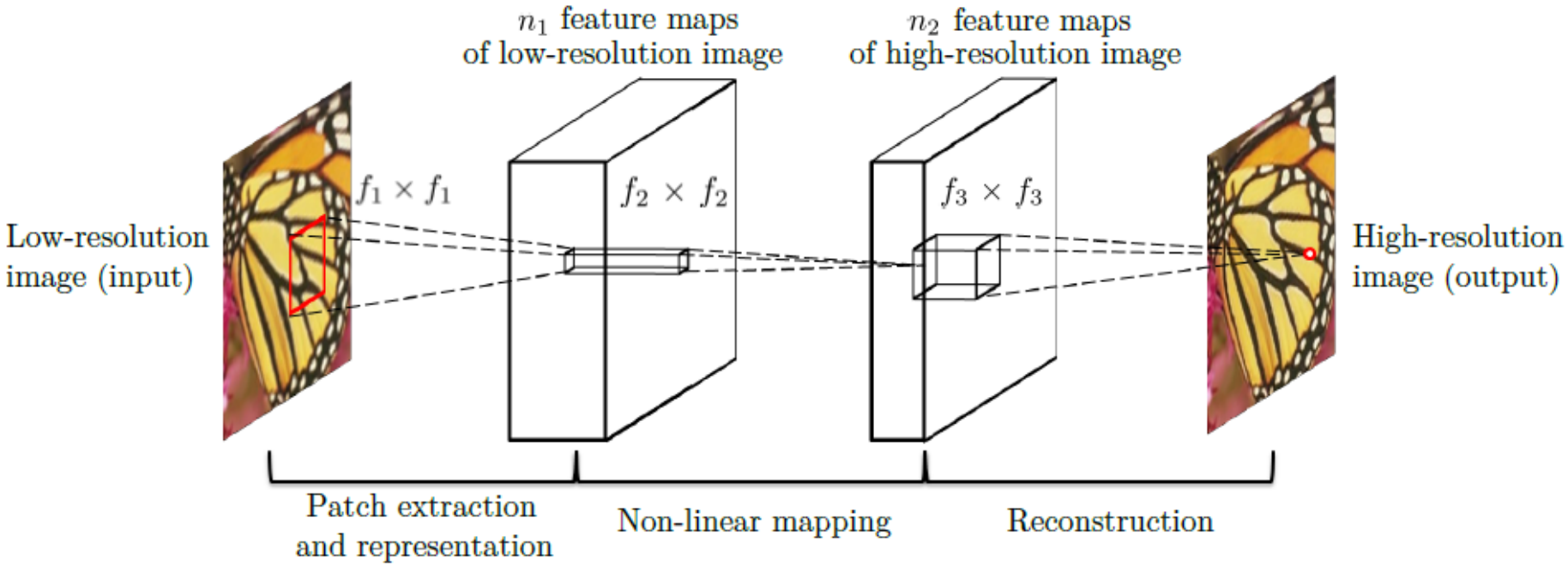
AE (AutoEncoder) – Examples and Applications

› Example #3: image reconstruction
  – Input: corrupted image, output; clean iage



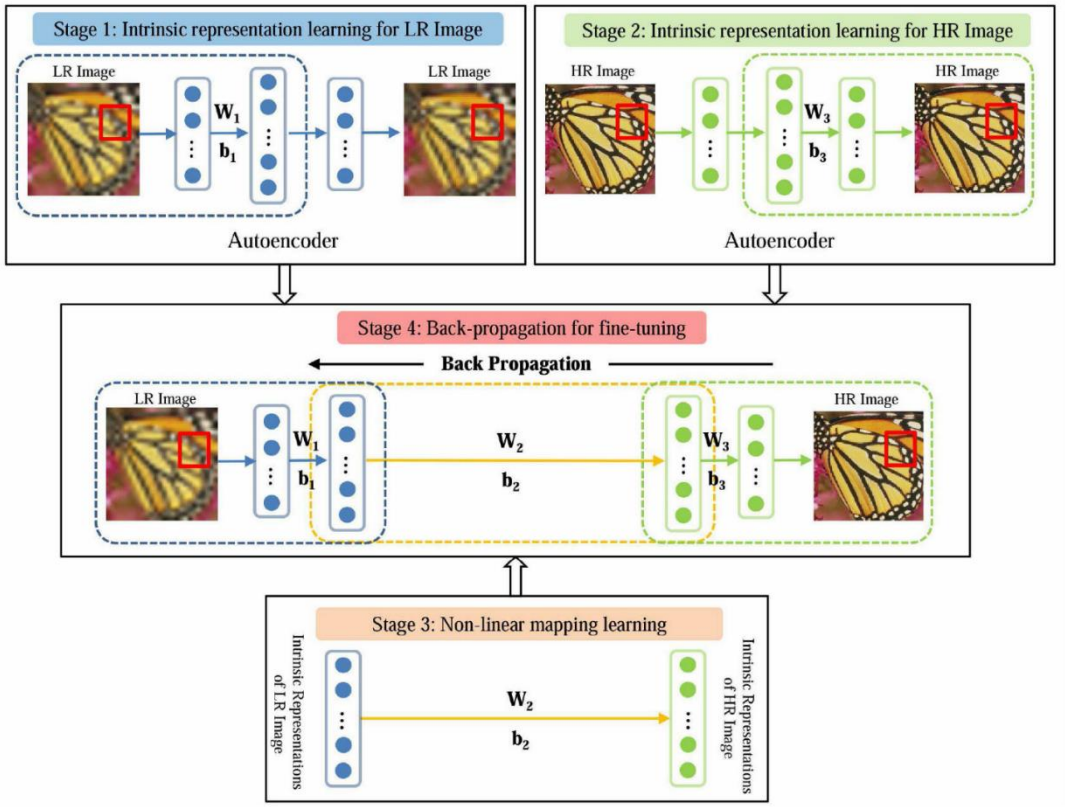
AE (AutoEncoder) – Examples and Applications

› Example #4: image Super Resolution - Coupled Deep Autoencoder for Single Image Super-Resolution, IEEE, ITC, 2015



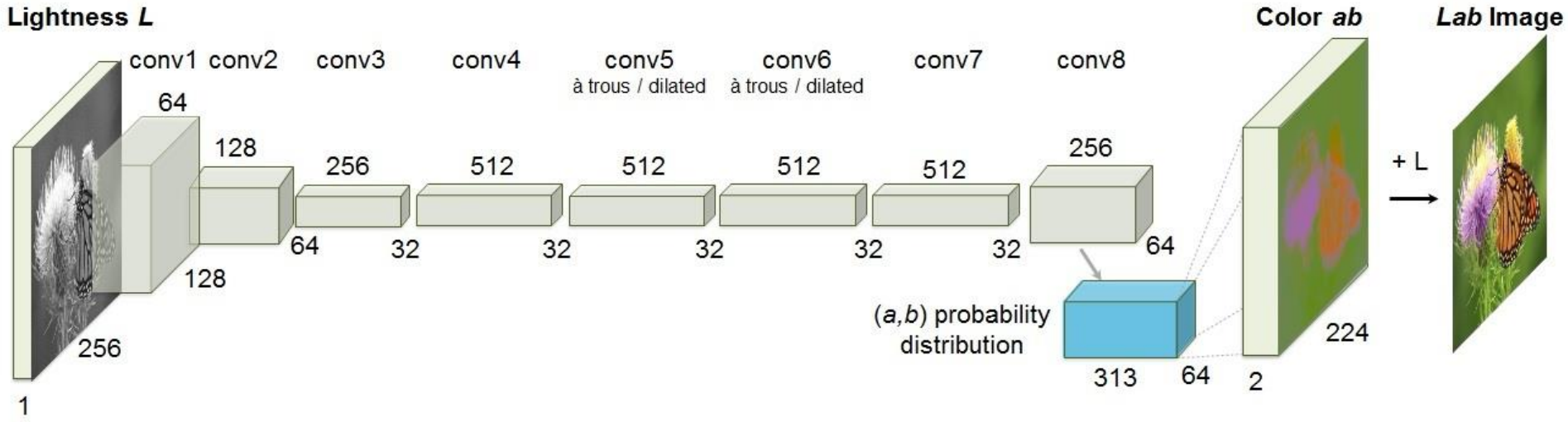
AE (AutoEncoder) – Examples and Applications

› Example #4: image Super Resolution



AE (AutoEncoder) – Examples and Applications

› Example #5: image Colorization

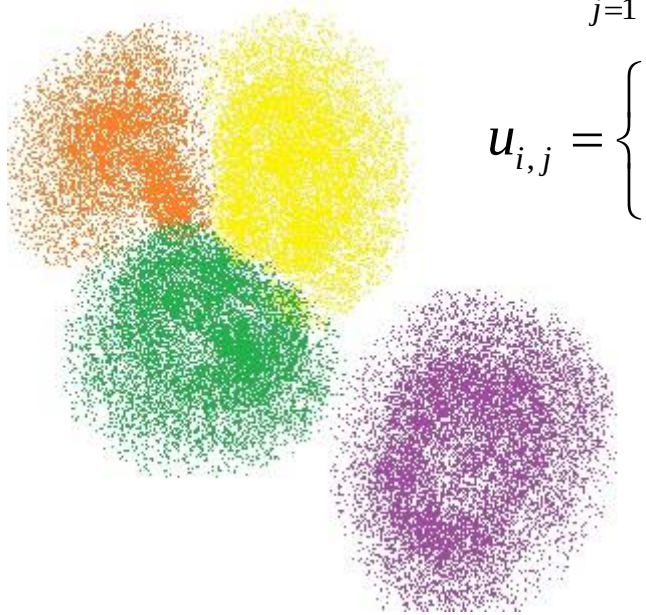


Deep Clustering

› First! Ordinary Clustering:

$$J = \sum_{j=1}^{C} \sum_{i \in \Omega_j} \left\| x_i - \omega_j \right\|^2 = \sum_{j=1}^{C} \sum_{i=1}^{N} u_{i,j} \left\| x_i - \omega_j \right\|^2$$

$$u_{i,j} = \begin{cases} 1 & x_i \in \omega_j \\ 0 & x_i \notin \omega_j \end{cases}$$



Deep Clustering

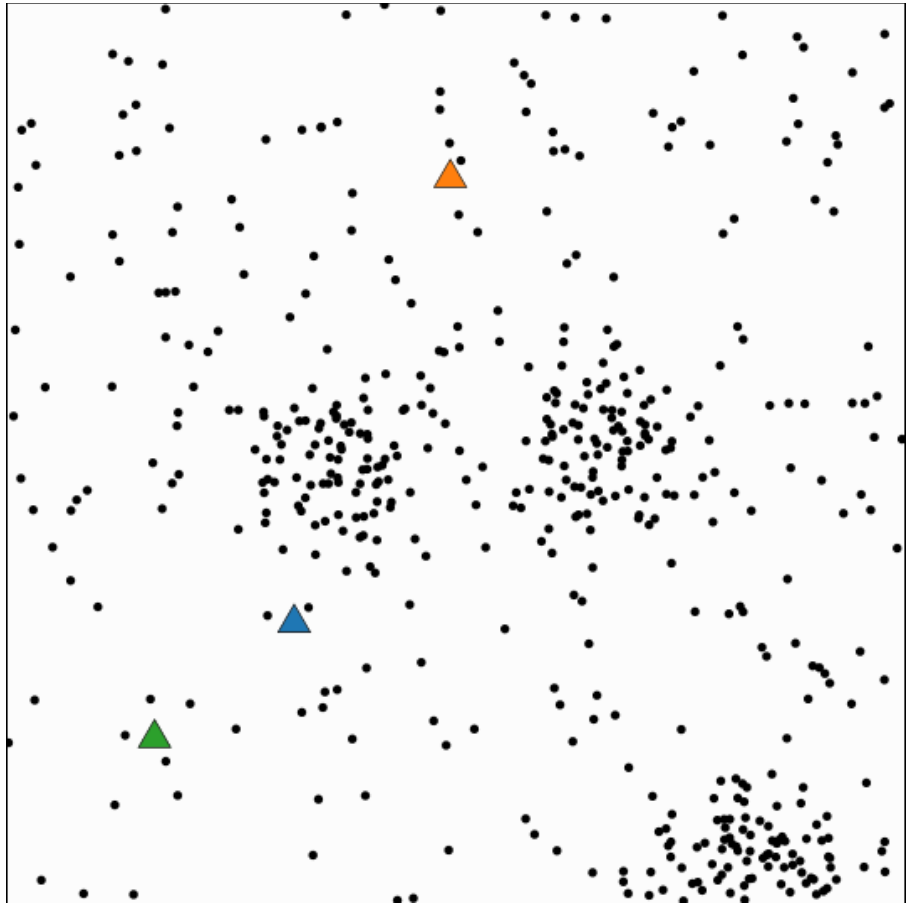
- k-means Clustering:
  - Initial centroids
  - Determine $u_{i,j}$ based on minimum distance
  - Determine w_j based on minimum internal energy (CoG)

$$J = \sum_{j=1}^{C} \sum_{i \in \Omega_j} \left\| x_i - \omega_j \right\|^2 = \sum_{j=1}^{C} \sum_{i=1}^{N} u_{i,j} \left\| x_i - \omega_j \right\|^2$$

$$u_{i,j} = \begin{cases} 1 & x_i \in \omega_j \\ 0 & x_i \notin \omega_j \end{cases}$$

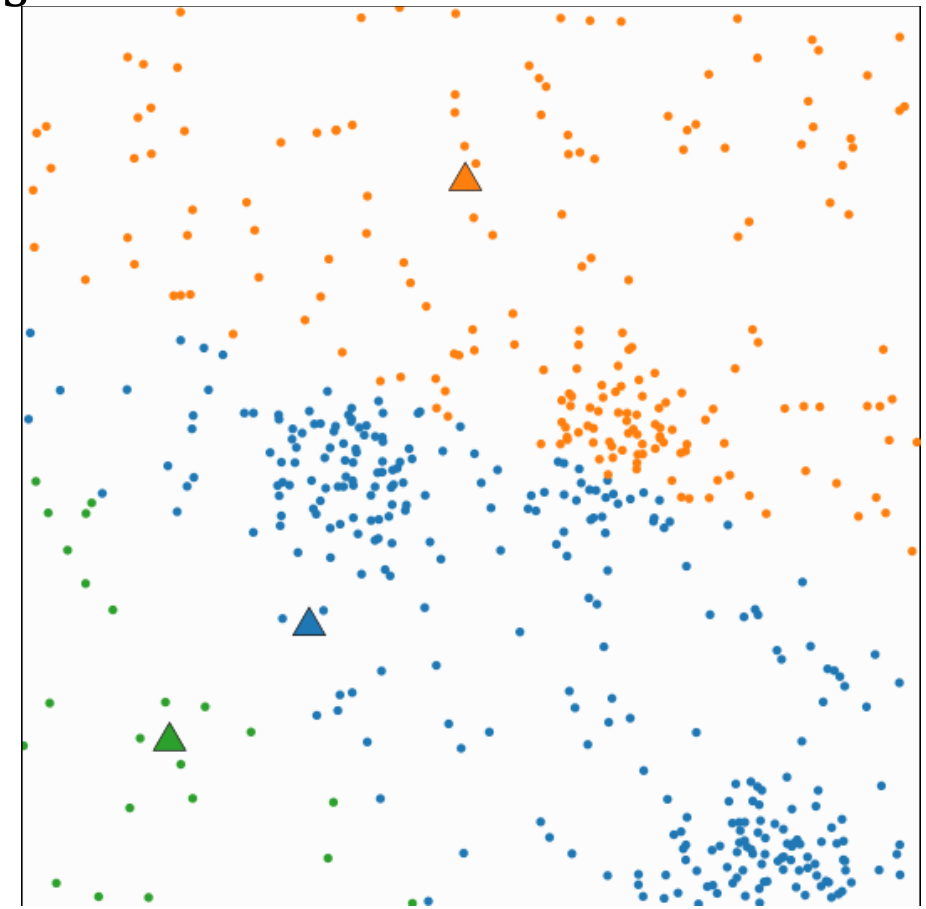
K-means

› Initialization



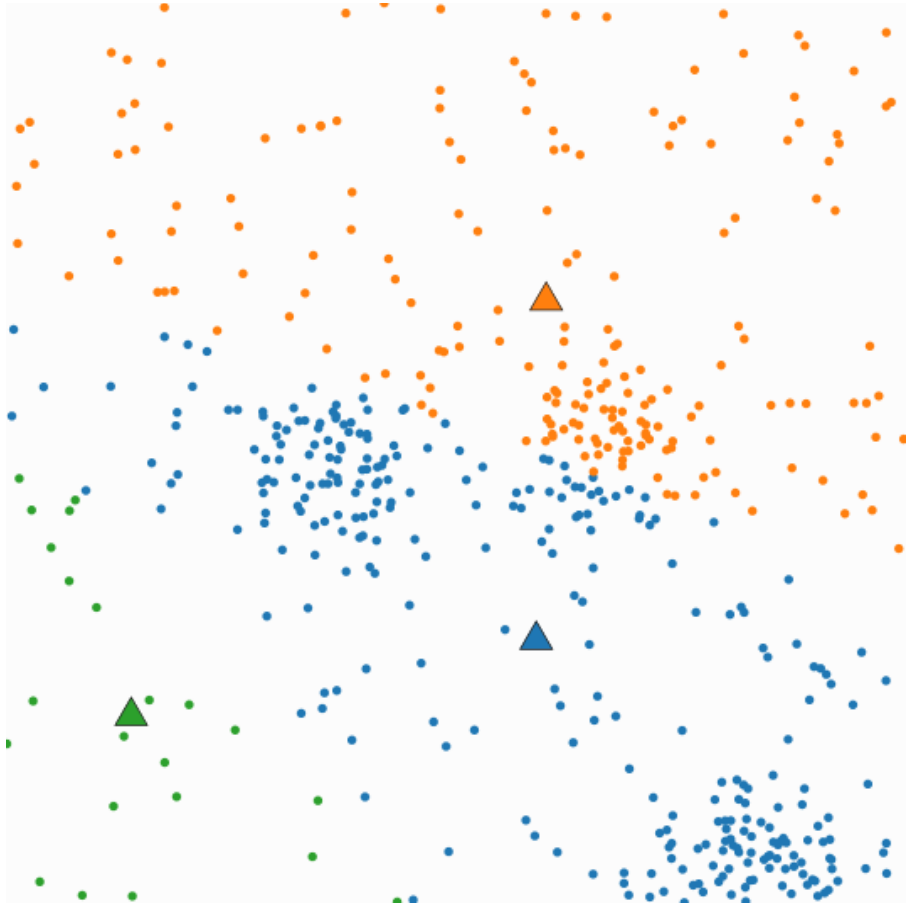
K-means

› Update Clusters



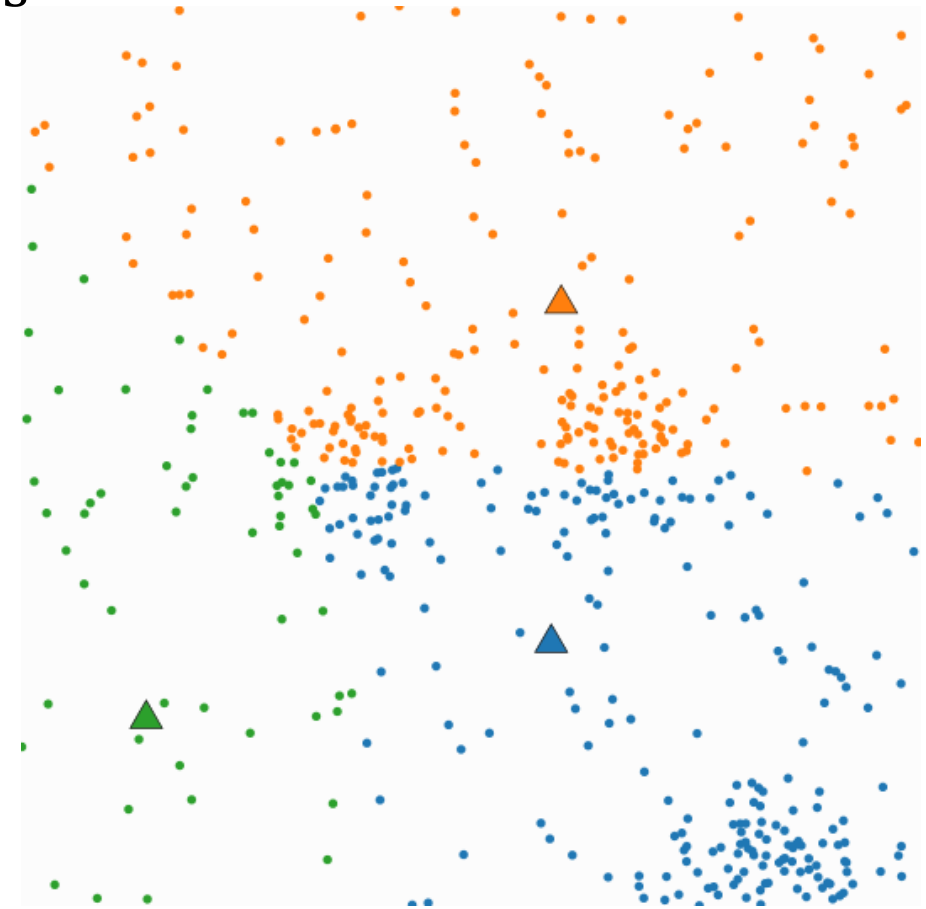
K-means

› Update Centers



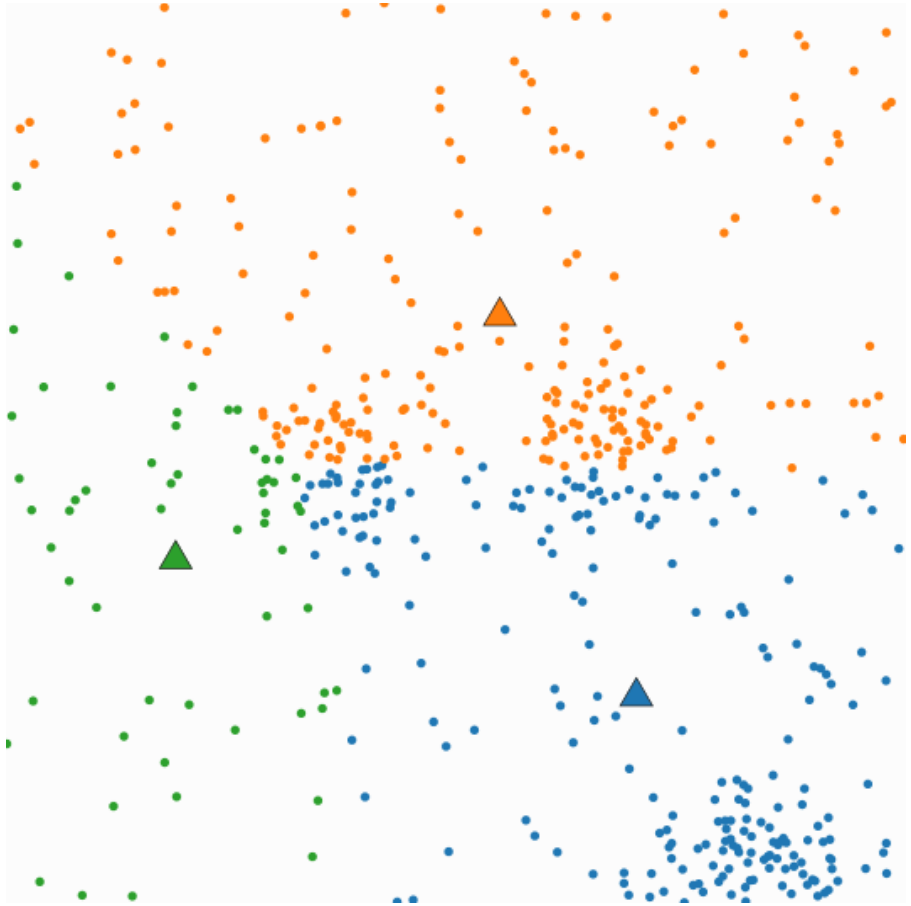
K-means

› Update Clusters



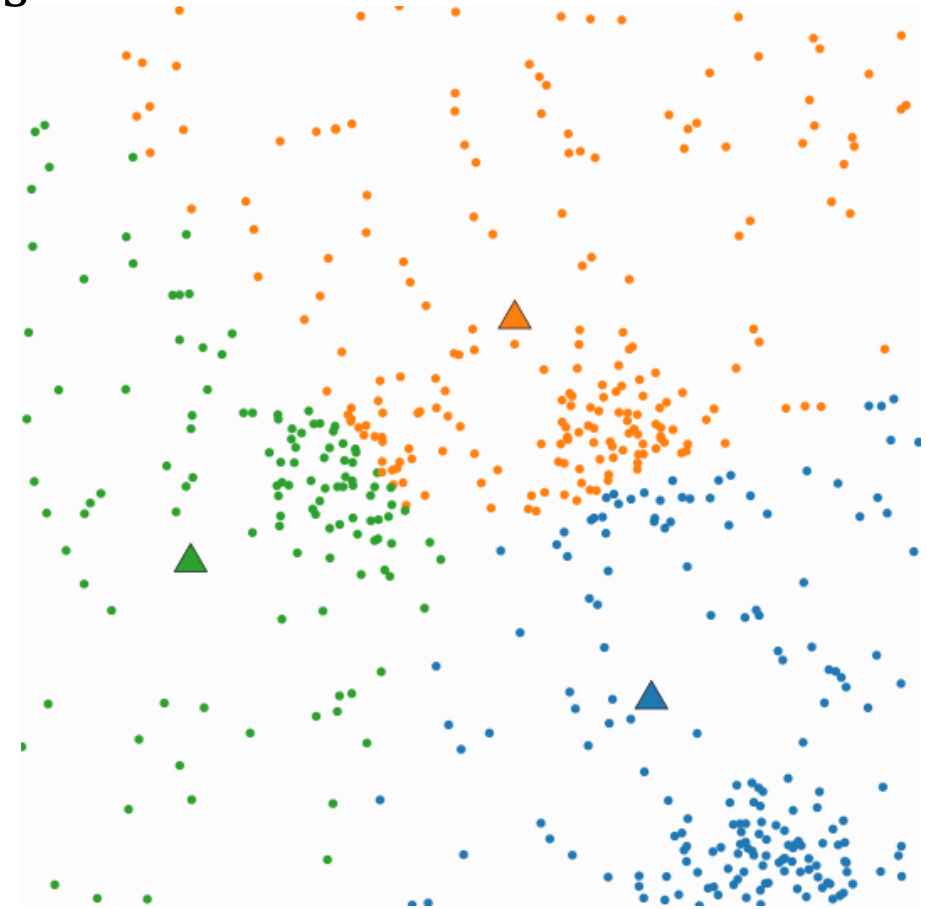
K-means

› Update Centers



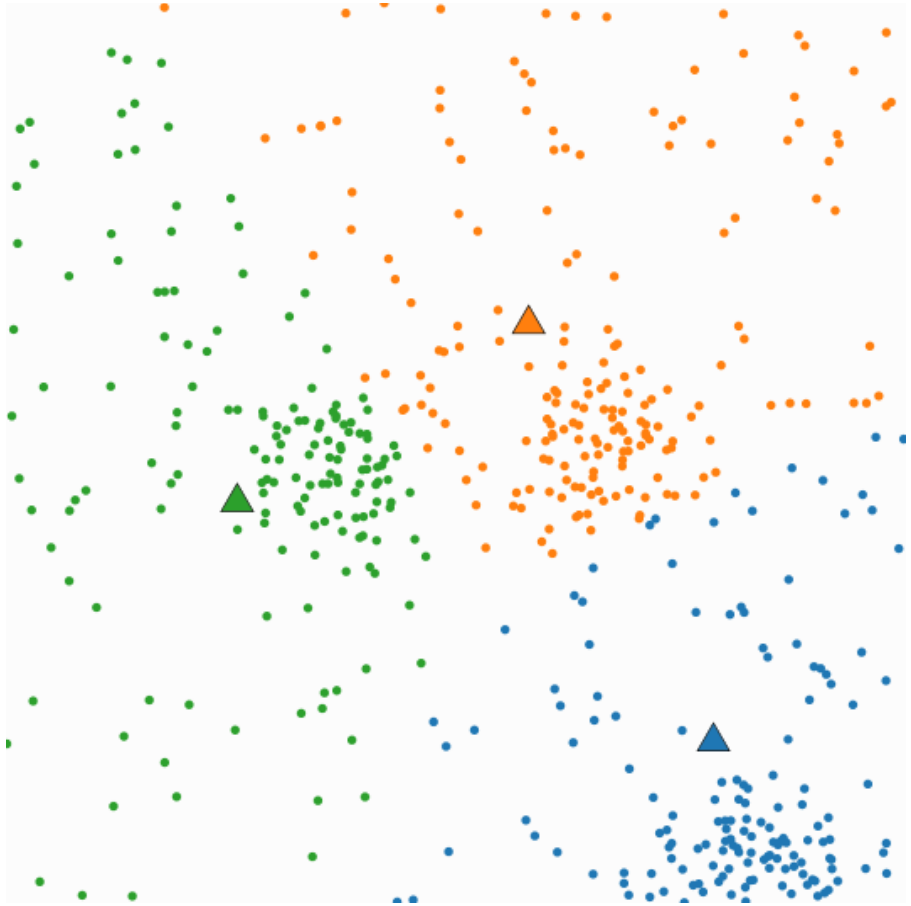
K-means

› Update Clusters



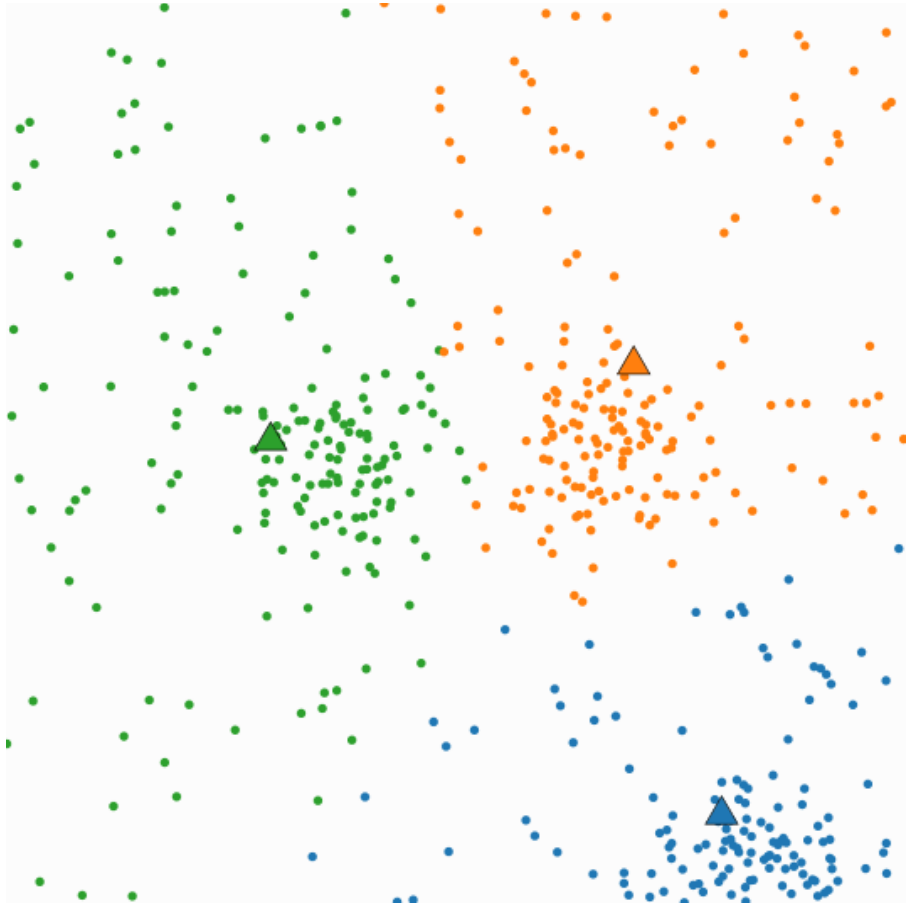
K-means

› Update Centers



K-means

› Final



DCN - Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering, 2017

› K-means Clustering Cost:

$$\min_{\boldsymbol{M}\in\mathbb{R}^{M\times K},\{\boldsymbol{s}_i\in\mathbb{R}^K\}} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{M}\boldsymbol{s}_i\|_2^2$$

$$\text{s.t. } s_{j,i} \in \{0,1\},\ \mathbf{1}^T\boldsymbol{s}_i = 1\ \ \forall i,j,$$

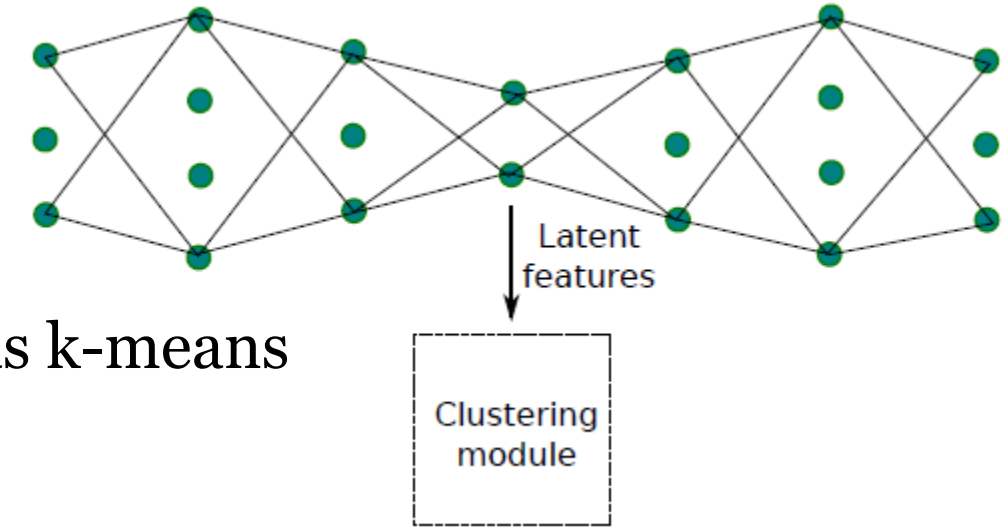
$$\{\boldsymbol{x}_i\}_{i=1,\dots,N} \text{ where } \boldsymbol{x}_i \in \mathbb{R}^M$$

› s_i: one-hot assignment vector for sample i

› m_k (k^{th} column M): CoG of cluster k (CoG: Center of Gravity)

DCN - Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering, 2017

- DCN: AE+k-means
  - Solve in two pass
  - Freeze **M** and $\mathbf{s_i}$ train AE
  - Freeze **AE** and tune **M** ans $\mathbf{s_i}$ as k-means
    - Offline or online k-means

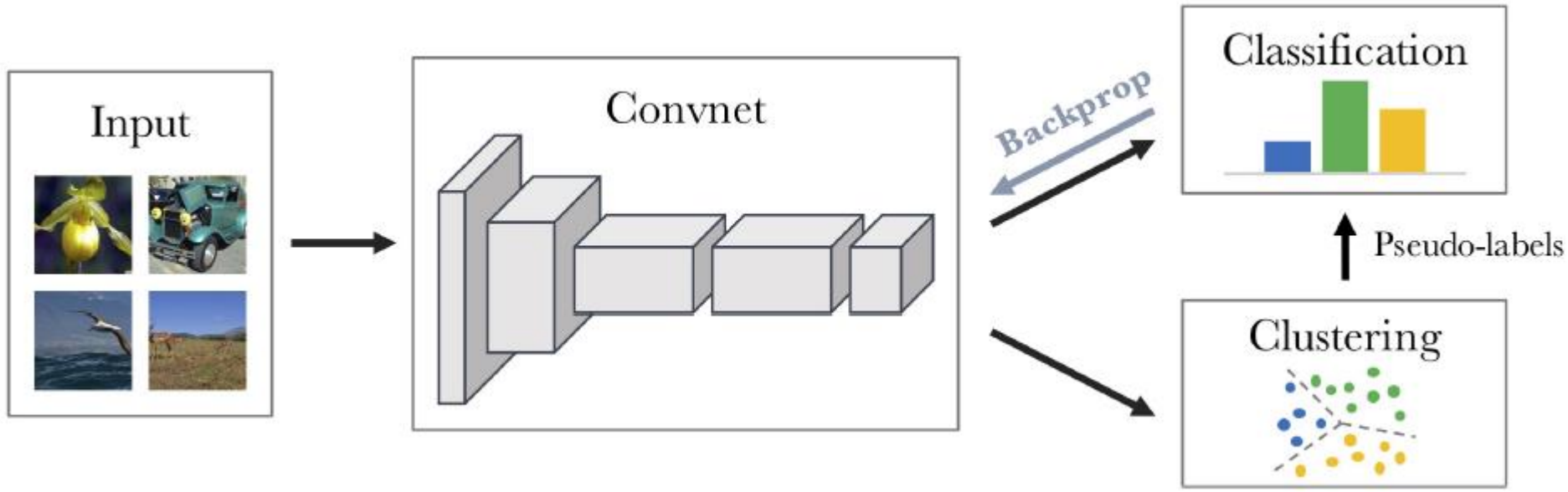


$$\min_{\substack{\mathcal{W},\mathcal{Z},\\ M,\{s_i\}}} \sum_{i=1}^{N} \left(\ell\left(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x}_i)), \boldsymbol{x}_i\right) + \frac{\lambda}{2} \left\| \boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{M}\boldsymbol{s}_i \right\|_2^2 \right) \tag{4}$$

$$\text{s.t. } s_{j,i} \in \{0,1\},\ \mathbf{1}^T s_i = 1\ \ \forall i, j,$$

DeepCluster - Deep Clustering for Unsupervised Learning of Visual Features, 2018, Facebook AI-Lab

› Generate Pseudo label with k-means, then train CNN!



DEC - Unsupervised Deep Embedding for Clustering Analysis, 2016

- Deep Embedded Clustering
  - AE and Clustering criteria embed to one Loss
- Goal: Dimension reduction and find cluster centers simultaneously

$$\{\mu_j \in Z\}_{j=1}^{k}$$

- Clustering Criteria:
  - 1) Soft assignment ($\alpha = 1$):

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\alpha)^{-\frac{\alpha+1}{2}}}, \quad z_i = f_\theta(x_i)$$

  - 2) Clustering criteria (use a target distribution)

$$L = \mathrm{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

DEC - Unsupervised Deep Embedding for Clustering Analysis, 2016

› Target distribution? (may be hard, assign to nearest), but:

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_j \left(q_{ij}^2 / \sum_i q_{ij} \right)}$$

› Optimization:

$$\frac{\partial L}{\partial z_i} = \frac{\alpha+1}{\alpha} \sum_j (1 + \frac{\|z_i - \mu_j\|^2}{\alpha})^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j),$$

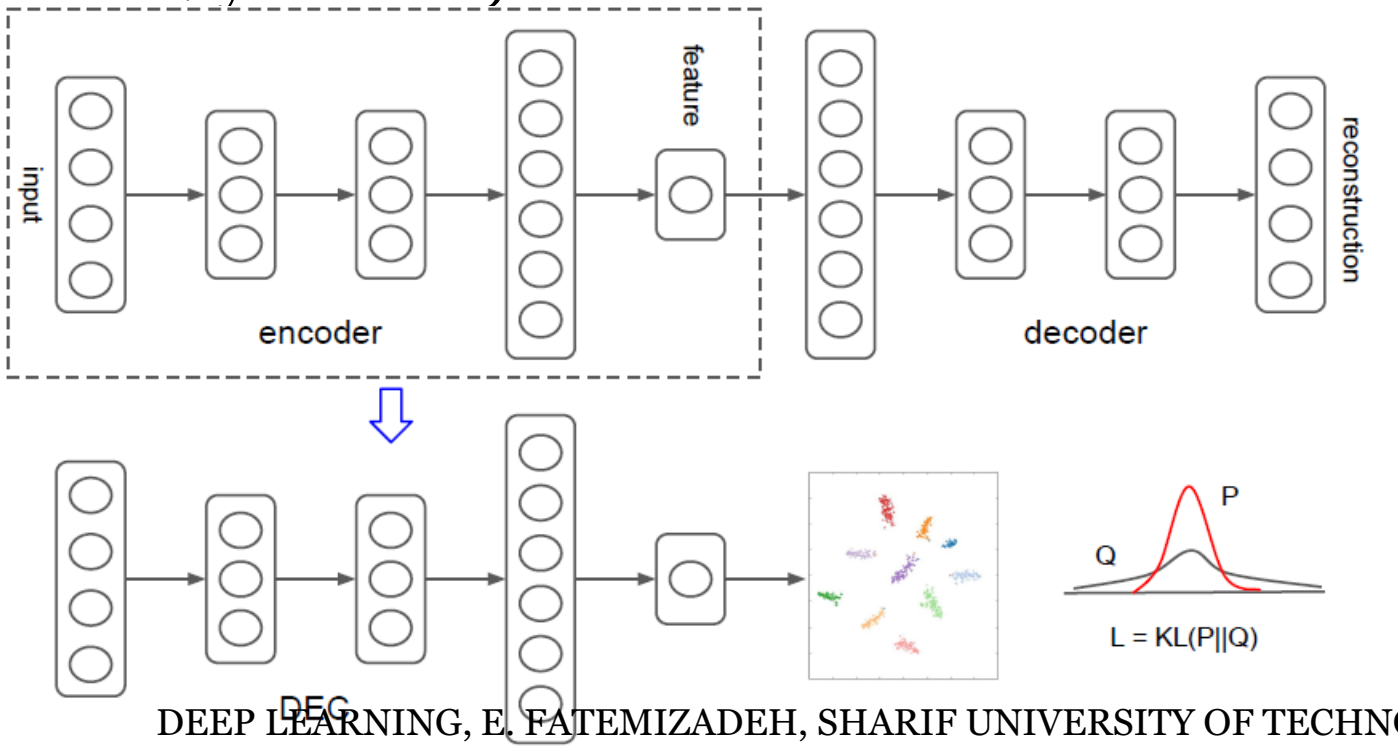
Feed to Encoder

$$\frac{\partial L}{\partial \mu_j} = -\frac{\alpha+1}{\alpha} \sum_i (1 + \frac{\|z_i - \mu_j\|^2}{\alpha})^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j).$$

Just for Clustering

DEC - Unsupervised Deep Embedding for Clustering Analysis, 2016

› Deep Embedded Clustering:
  – AE: Initialize by a Pre-trained SAE (Stacked AutoEncoder)
  – Feed generated code (discard decoder part) to clustering Loss (Initialize by kmeans)



$$\frac{\partial L}{\partial z_i}$$

$$z_i = f_\theta(x_i)$$

$$\frac{\partial L}{\partial \mu_j}$$

Variational AutoEncoder (VAE)

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

Variational AutoEncoder (VAE)

- 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
    - ….

Variational AutoEncoder (VAE)

- 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)
    - ....

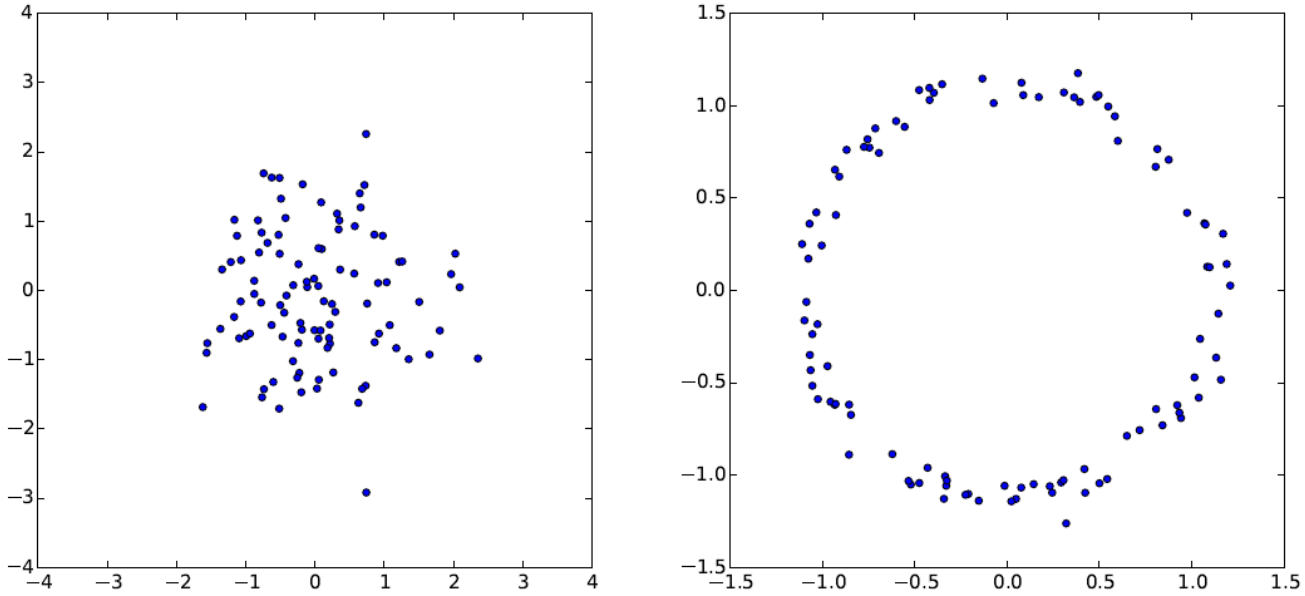
Variational AutoEncoder (VAE)

- Generating Arbitrarily Distributed Random Variables:
  - Suppose we have r.v.'s: "u ~ 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 $g(x)$
  - $G^{-1}(P(x))$

Variational AutoEncoder (VAE)

› Generating Arbitrarily Distributed Random Variables:

– z: 2D Gaussian r.v. → $\frac{z}{10} + \frac{z}{\|z\|}$ follow circ(z)



Variational AutoEncoder (VAE)

› Kullback–Leibler (KL) Distance/Divergence:
  – Measure of distance (Divergence) between two pdfs

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right).$$

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

  – Non-Negative
  – Non-Symmetric

Variational AutoEncoder (VAE)

› KL Distance and ML:

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

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

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

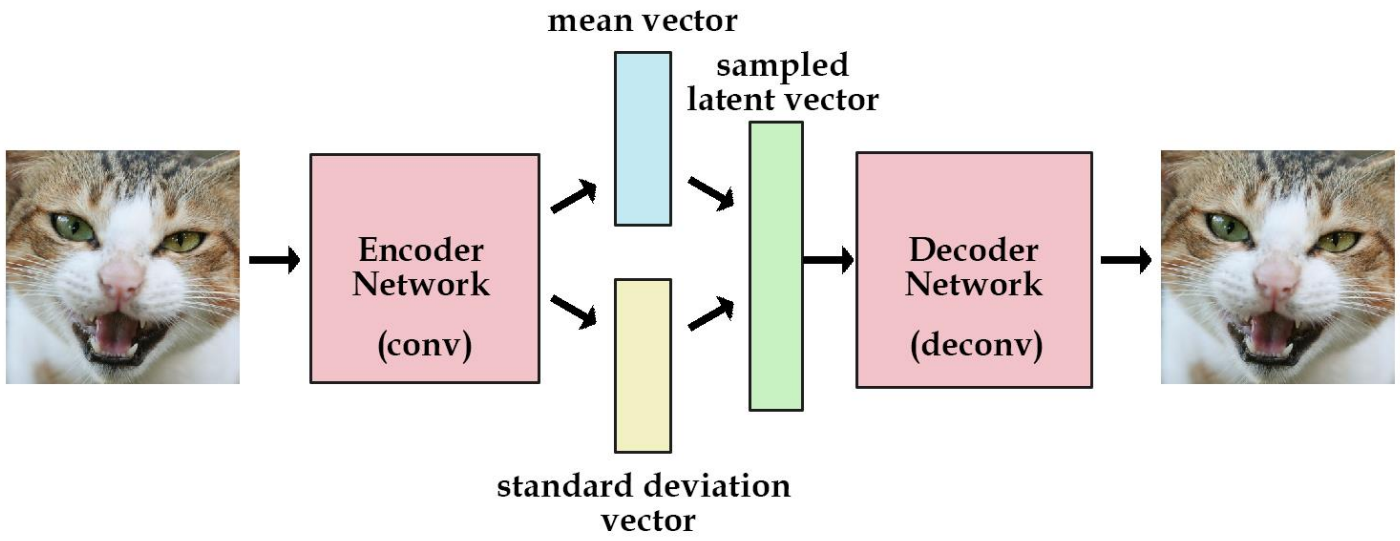
$$= \mathbb{E}_{x\sim P(x|\theta^*)}\left[\log P(x|\theta^*)\right] - \mathbb{E}_{x\sim P(x|\theta^*)}\left[\log P(x|\theta)\right]$$

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

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

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

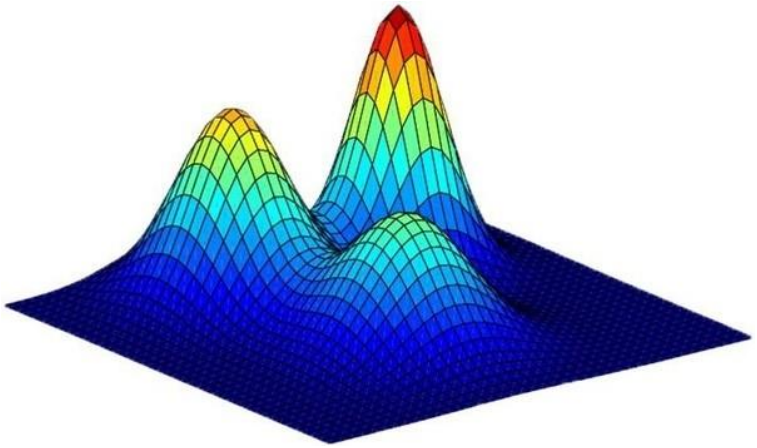


Variational AutoEncoder (VAE)

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

Variational AutoEncoder (VAE)

- What is Mixture Model:
- Input: unlabeled data: $\{x_i\}_{i=1}^{N}$



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

$$\sum_{k=1}^{K} \pi_k = 1$$

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

Variational AutoEncoder (VAE)

- 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 generating data given latent variable.

Variational AutoEncoder (VAE)

› 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
- Convert problem to optimization problem:
  - › Simple/Parametric modelling of $P(z|X)$ and minimize distance between model and data

Variational AutoEncoder (VAE)

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

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

$$= E\left[\log \frac{Q(z|X)}{P(z|X)}\right]$$

$$= E[\log Q(z|X) - \log P(z|X)]$$

› Let use $P(X)$, $P(X|z)$, and $P(z)$, via Bayes Rule

Variational AutoEncoder (VAE)

› Let use P(X), P(X|z), and P(z), via Bayes Rule

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

$$D_{KL}[Q(z|X)\|P(z|X)] = E\left[\log Q(z|X) - \log \frac{P(X|z)P(z)}{P(X)}\right]$$

$$= E[\log Q(z|X) - (\log P(X|z) + \log P(z) - \log P(X))]$$

$$= E[\log Q(z|X) - \log P(X|z) - \log P(z) + \log P(X)]$$

Variational AutoEncoder (VAE)

› Expectation is over z this move P(X) outside of the:

$$D_{KL}[Q(z|X)\|P(z|X)] = E[\log Q(z|X) - \log P(X|z) - \log P(z) + \log P(X)]$$

$$D_{KL}[Q(z|X)\|P(z|X)] = E[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X)$$

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

› We will transform right-hand to get another KL distance!

Variational AutoEncoder (VAE)

› We will transform right-hand to get another KL distance!

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

$$\begin{aligned}\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] &= E[\log P(X|z) - (\log Q(z|X) - \log P(z))] \\ &= E[\log P(X|z)] - E[\log Q(z|X) - \log P(z)] \\ &= E[\log P(X|z)] - D_{KL}[Q(z|X)\|P(z)]\end{aligned}$$

› VAE objective function: ELBO (Evidence Lower Bound).

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

Variational AutoEncoder (VAE)

› So what!! 🤔

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

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

› Q(z|X): Project data *X* into latent variable ~ {Encoder Network}

› z: Latent variable ~ {Generated Code}

› P(X|z): Generate data given latent variable ~ {Decoder Network}

› In fact: *VAE tries to find the lower bound of logP(X)* (except some error, left-hand KL distance)

Variational AutoEncoder (VAE)

› So what!!

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

› ELBO Maximization ~ Log-Likelihood P(X) under decoding error

› Objective: **Maximizing** $logP(X/z)$ and **minimizing** KL distance between our simple distribution $Q(z/X)$ and the true latent distribution $P(z)$.

Variational AutoEncoder (VAE)

› How to solve!

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

› **Maximizing** $logP(X|z)$: Any discriminative supervised-parametric model (SVM/MLP/…):

› Any classifier with z as input and X as output, and a proper loss

Variational AutoEncoder (VAE)

› How to solve!

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

› **minimizing** $D_{KL}[Q(z|X)\|P(z)]$:

› $P(z)$: Latent variable pdf, we will sample it to generate X: $N(0,1)$

– Easy to make $Q(z|X)$ as-close-as possible to it

› $Q(z|X)$: Another simplification: $N(\mu(X), \Sigma(X))$, and diagonal, $\Sigma(X)$!

– Easy to compute KL between two Gaussian!

Variational AutoEncoder (VAE)

› KL distance between two Gaussian (in k-dimension):

$$D_{\mathrm{KL}}(\mathcal{N}_0 \parallel \mathcal{N}_1) = \frac{1}{2}\left(\operatorname{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + (\mu_1 - \mu_0)^{\mathsf{T}}\Sigma_1^{-1}(\mu_1 - \mu_0) - k + \ln\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right)\right)$$

› In VAE ($z \sim N(0,1)$):

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

$$D_{KL}[N(\mu(X), \Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\operatorname{tr}(\Sigma(X)) + \mu(X)^T\mu(X) - k - \log\det(\Sigma(X))\right)$$

Variational AutoEncoder (VAE)

› **Minimizing** $D_{KL}[Q(z|X)\|P(z)]$:

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\mathrm{tr}(\Sigma(X)) + \mu(X)^T\mu(X) - k - \log\det(\Sigma(X))\right)$$

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \log\prod_k \Sigma(X)\right)$$

$$= \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \sum_k \log\Sigma(X)\right)$$

$$= \frac{1}{2}\sum_k \left(\Sigma(X) + \mu^2(X) - 1 - \log\Sigma(X)\right)$$

Variational AutoEncoder (VAE)

- **Minimizing** $D_{KL}[Q(z|X)\|P(z)]$:
  - A numerical trick!: Model $\Sigma(X)$ as $log\Sigma(X)$, numerically stable to take *exponent* compared *log*

$$= \frac{1}{2}\sum_k \left(\Sigma(X) + \mu^2(X) - 1 - \log\Sigma(X)\right)$$

$$D_{KL}[N(\mu(X), \Sigma(X))\|N(0,1)] = \frac{1}{2}\sum_k \left(\exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X)\right)$$

Variational AutoEncoder (VAE)

- Encoder Module (sample, may be buggy, code):
  - Input: $Q(z|X)$, Output: $\mu(X)$ and $\Sigma(X)$
  - One Hidden Layer, 2D latent variable (n_z=2), z , Batch size:(m=50)

```
m = 50
n_z = 2
n_epoch = 10
# Q(z|X) -- encoder
inputs = Input(shape=(784,))
h_q = Dense(512, activation='relu')(inputs)
mu = Dense(n_z, activation='linear')(h_q)
log_sigma = Dense(n_z, activation='linear')(h_q)
```

- https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

Variational AutoEncoder (VAE)

› How to get (sample) z (a r.v.) from encoder output?
  – A Naïve solution: sample $N(\mu(X), \Sigma(X))$ is easy!
  – But sampling is not easy to handle in training phase with GD family

› Reparameterization Trick (*makes the network differentiable*):
  – Make it from standard N(0,1):

$$z = \mu(X) + \Sigma^{\frac{1}{2}}(X)\,\epsilon$$

› If $\varepsilon \sim \mathrm{N}(\mathbf{0},\mathbf{I})$ → $z \sim N(\mu(X), \Sigma(X))$

Variational AutoEncoder (VAE)

› Again (sample, may be buggy, code):

```
def sample_z(args):
    mu, log_sigma = args
    eps = K.random_normal(shape=(m, n_z), mean=0., std=1.)
    return mu + K.exp(log_sigma / 2) * eps


# Sample z ~ Q(z|X)
z = Lambda(sample_z)([mu, log_sigma])
```

› https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

Variational AutoEncoder (VAE)

› Decoder Module (Any discriminator):

› Output: $P(X|z)$

```
# P(X|z) -- decoder
decoder_hidden = Dense(512, activation='relu')
decoder_out = Dense(784, activation='sigmoid')

h_p = decoder_hidden(z)
outputs = decoder_out(h_p)
```

› https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

Variational AutoEncoder (VAE)

› Total Loss:

$$D_{KL}[N(\mu(X), \Sigma(X)) \| N(0,1)] = \frac{1}{2} \sum_k \left(\exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X) \right)$$

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

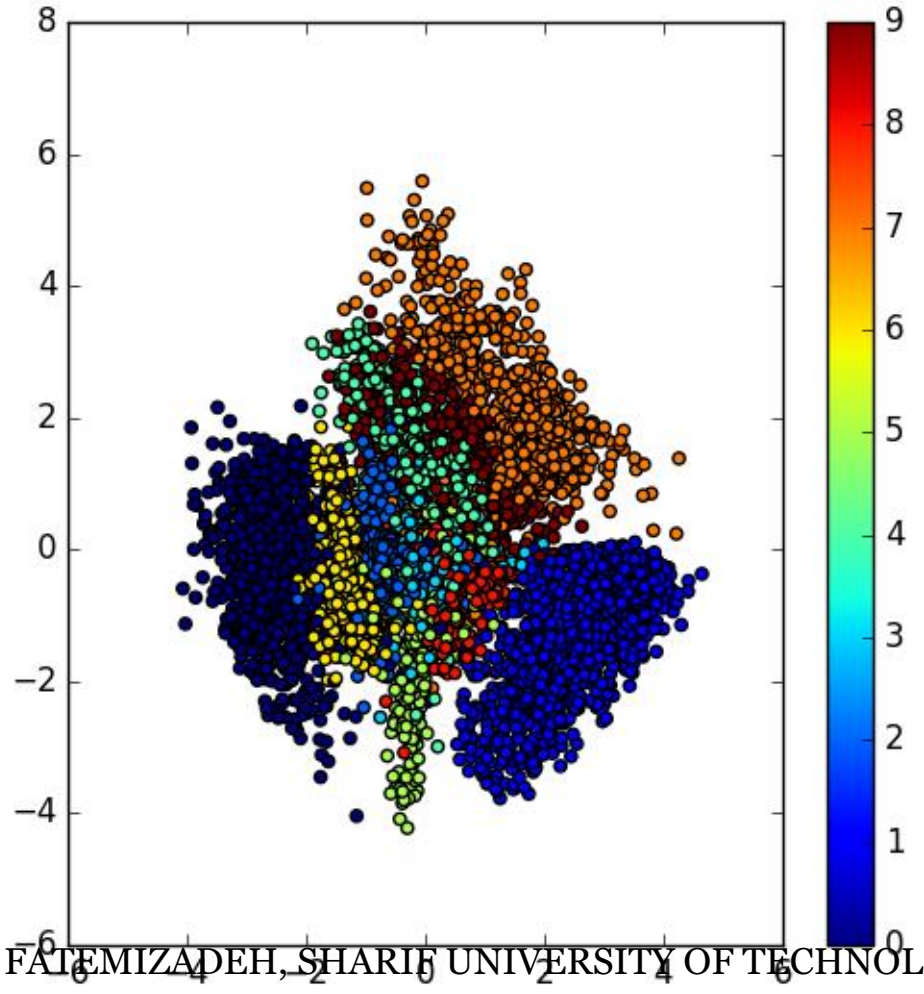
```
def vae_loss(y_true, y_pred):
    """ Calculate loss = reconstruction loss + KL loss for each data in minibatch """
    # E[log P(X|z)]
    recon = K.sum(K.binary_crossentropy(y_pred, y_true), axis=1)
    # D_KL(Q(z|X) || P(z|X)); calculate in closed form as both dist. are Gaussian
    kl = 0.5 * K.sum(K.exp(log_sigma) + K.square(mu) - 1. - log_sigma, axis=1)

    return recon + kl
```

Variational AutoEncoder (VAE)

› Example: MNIST

› Visualize: $Q(z|X)$



Variational AutoEncoder (VAE)

› Example: MNIST

› Output (bottom) by feed the data (top) into overall VAE net:

› Feed a sample $z \sim N(0,1)$ into the decoder module:

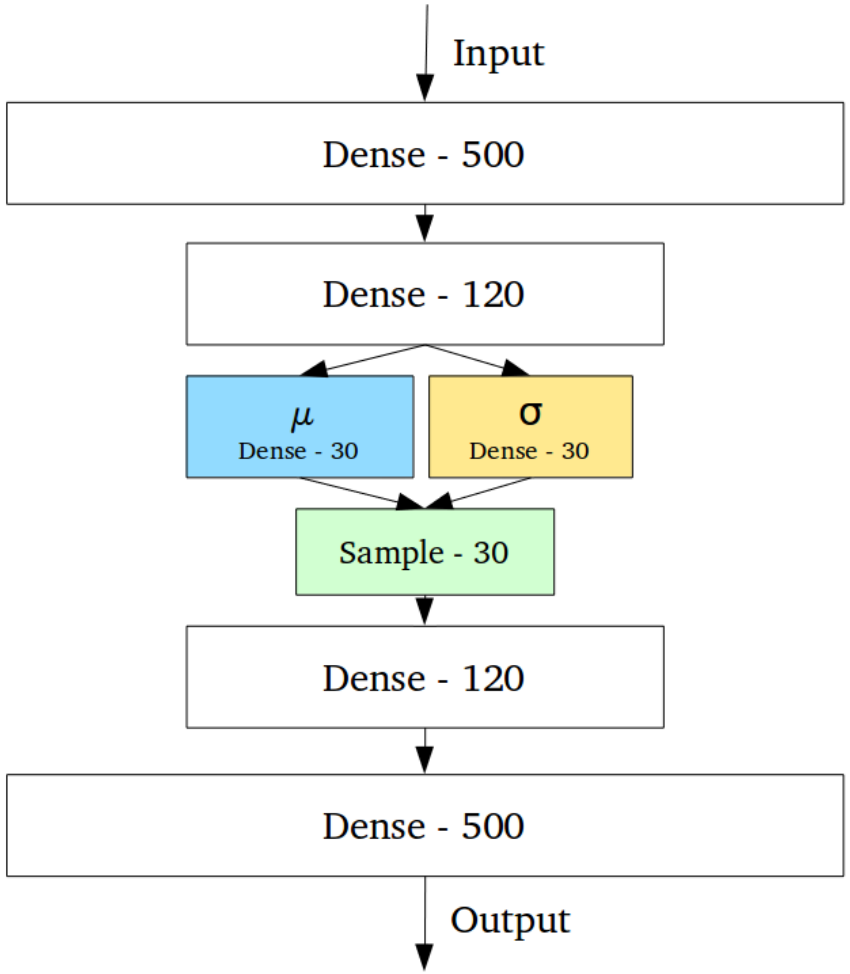
Variational AutoEncoder (VAE)

› An interesting *keras* implementation:

https://blog.keras.io/building-autoencoders-in-keras.html

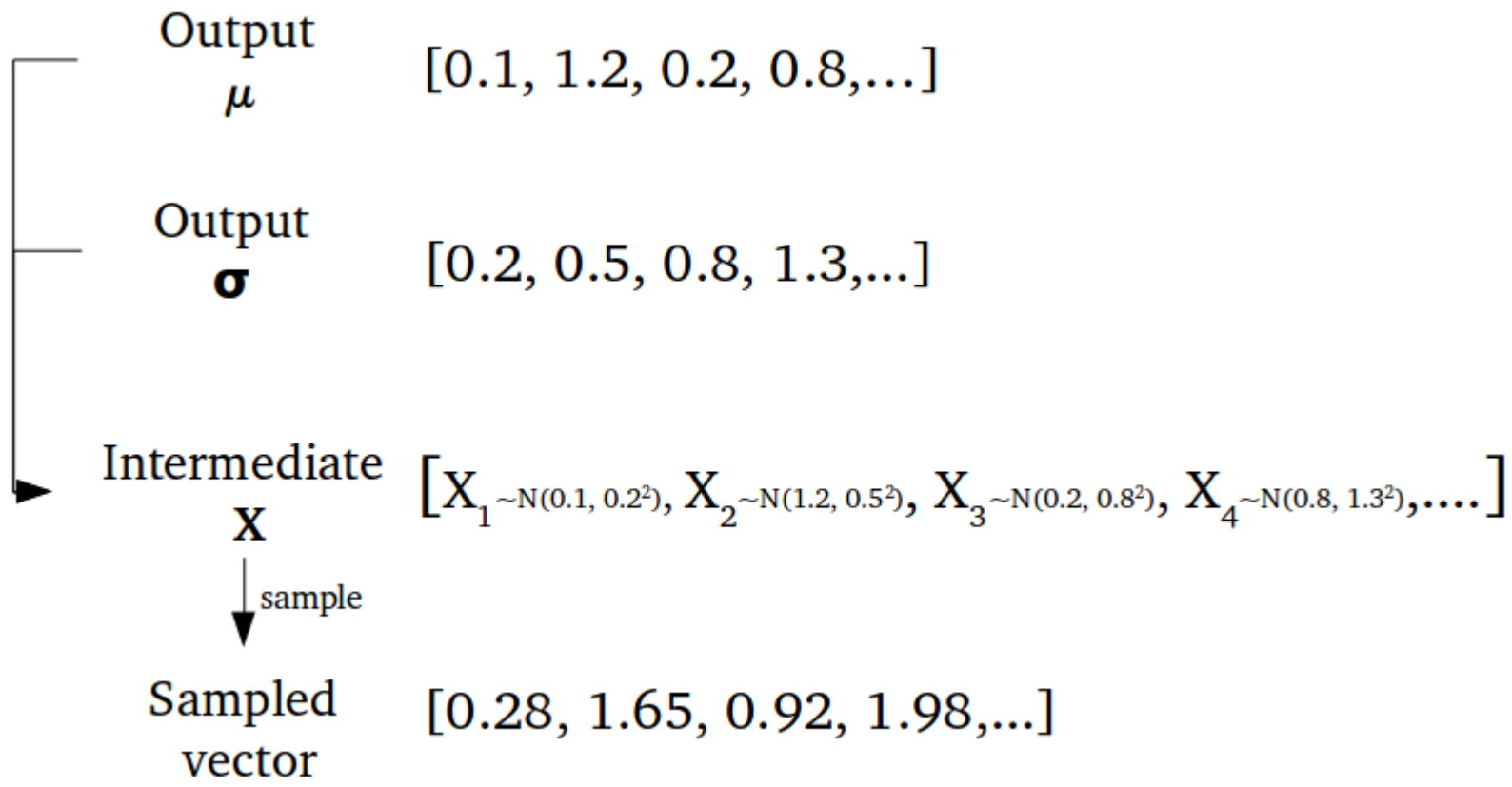
Variational AutoEncoder (VAE)

› An Overall Sy



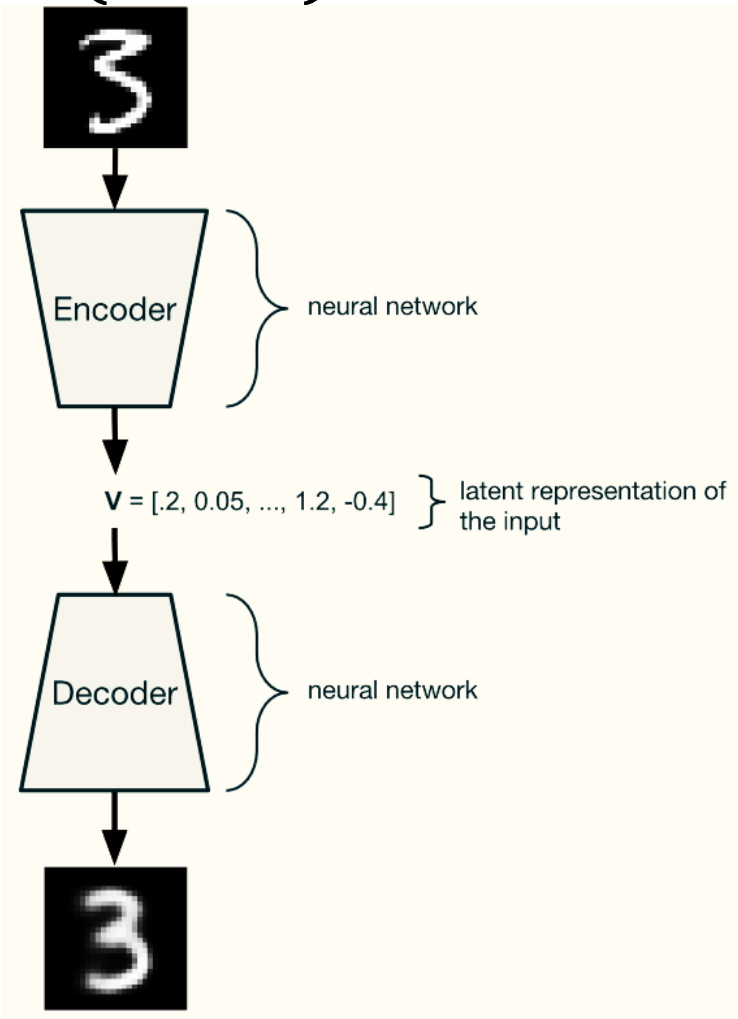
Variational AutoEncoder (VAE)

› Intermediate Layer Example:



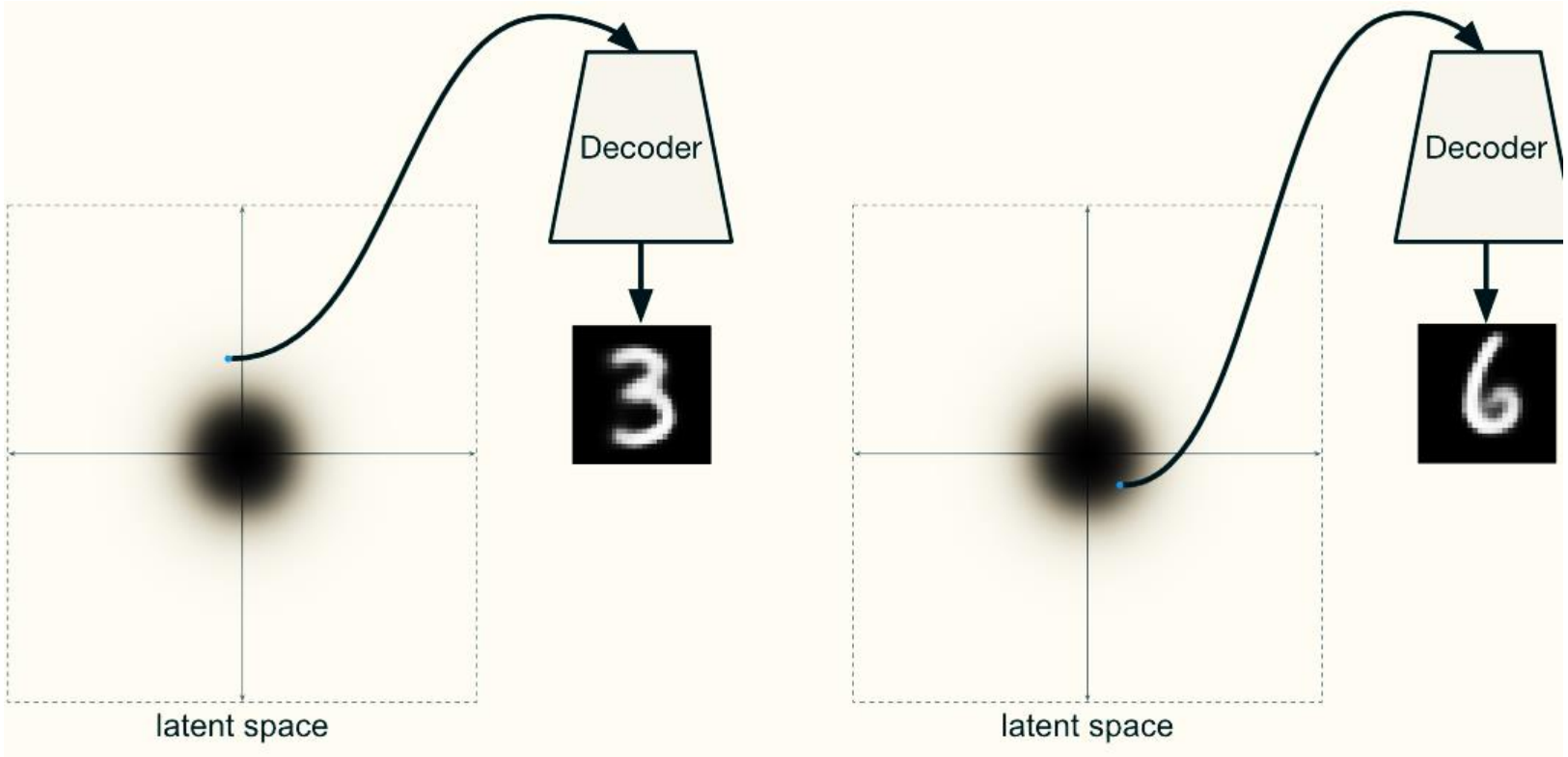
Variational AutoEncoder (VAE)

› Another Representation:
  – Conventional AE Training:



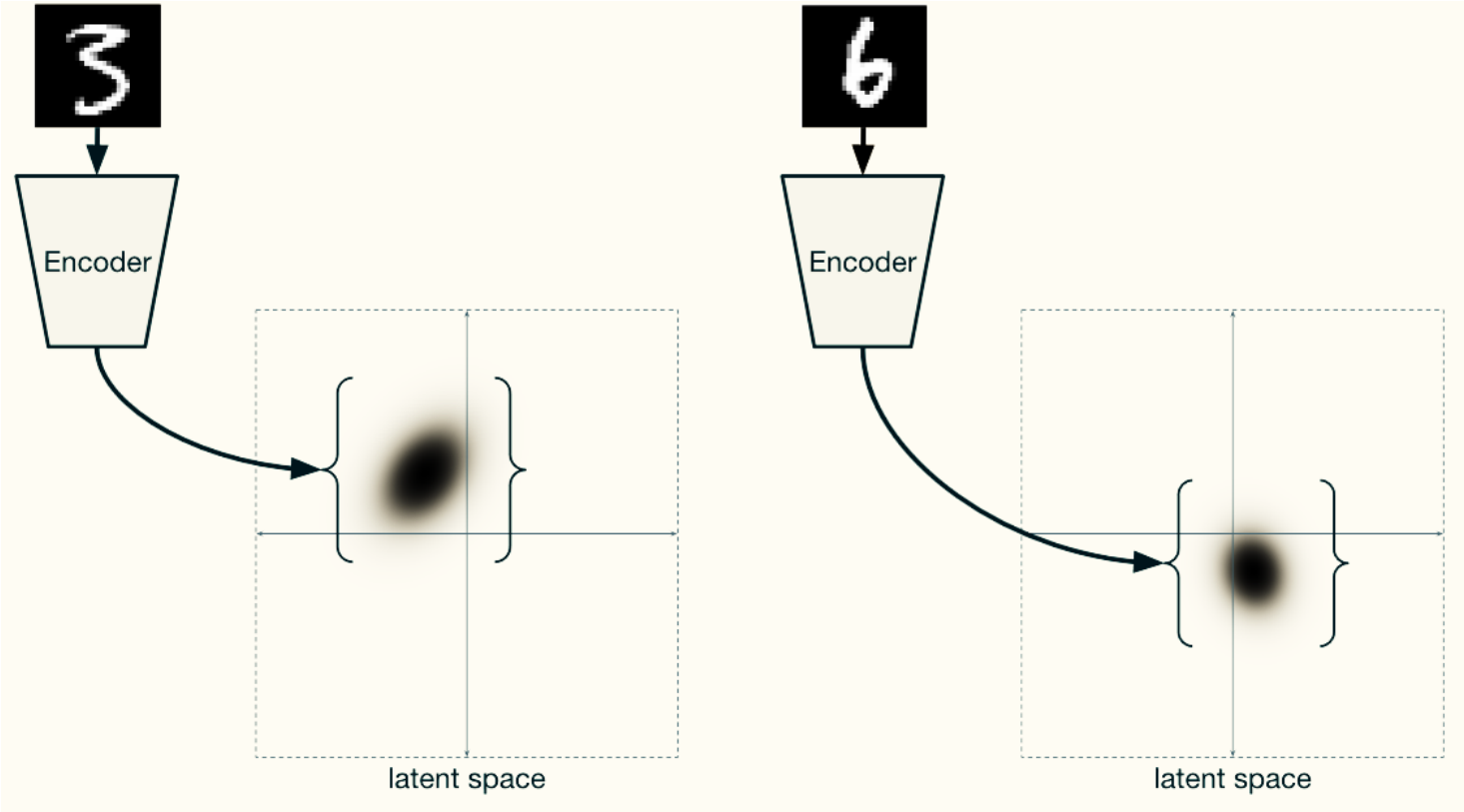
Variational AutoEncoder (VAE)

› VAE Decoder:



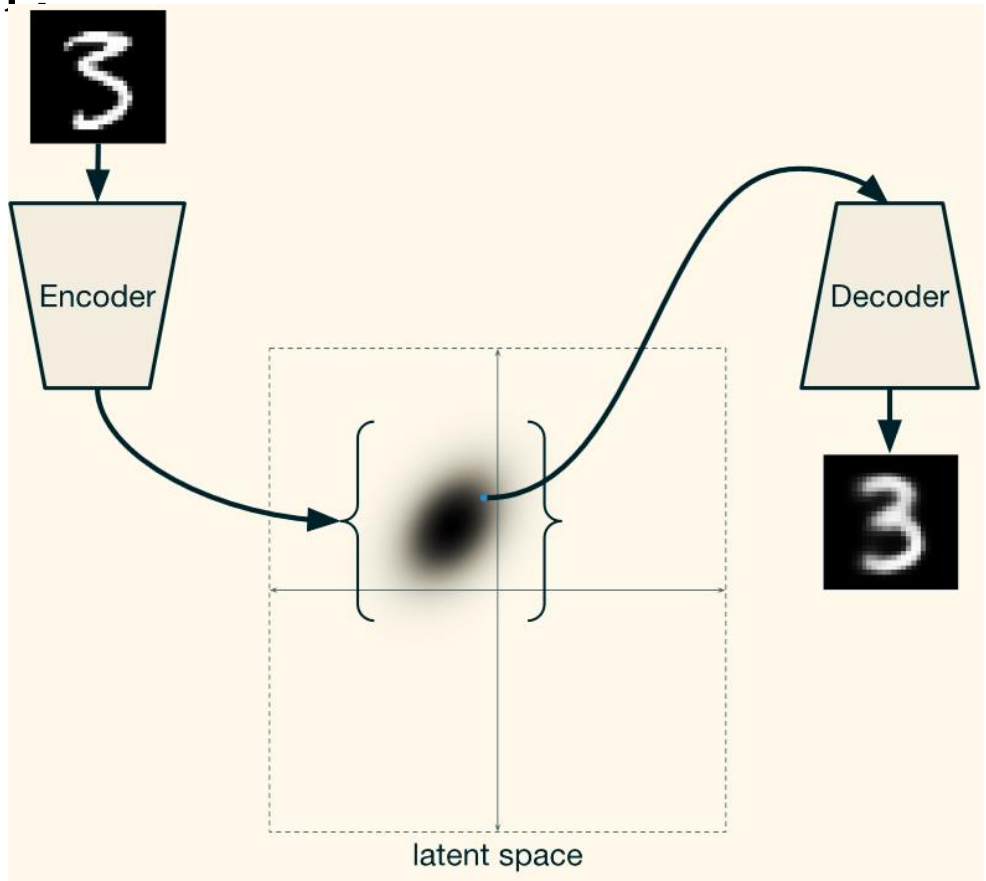
Variational AutoEncoder (VAE)

› VAE Encoder:



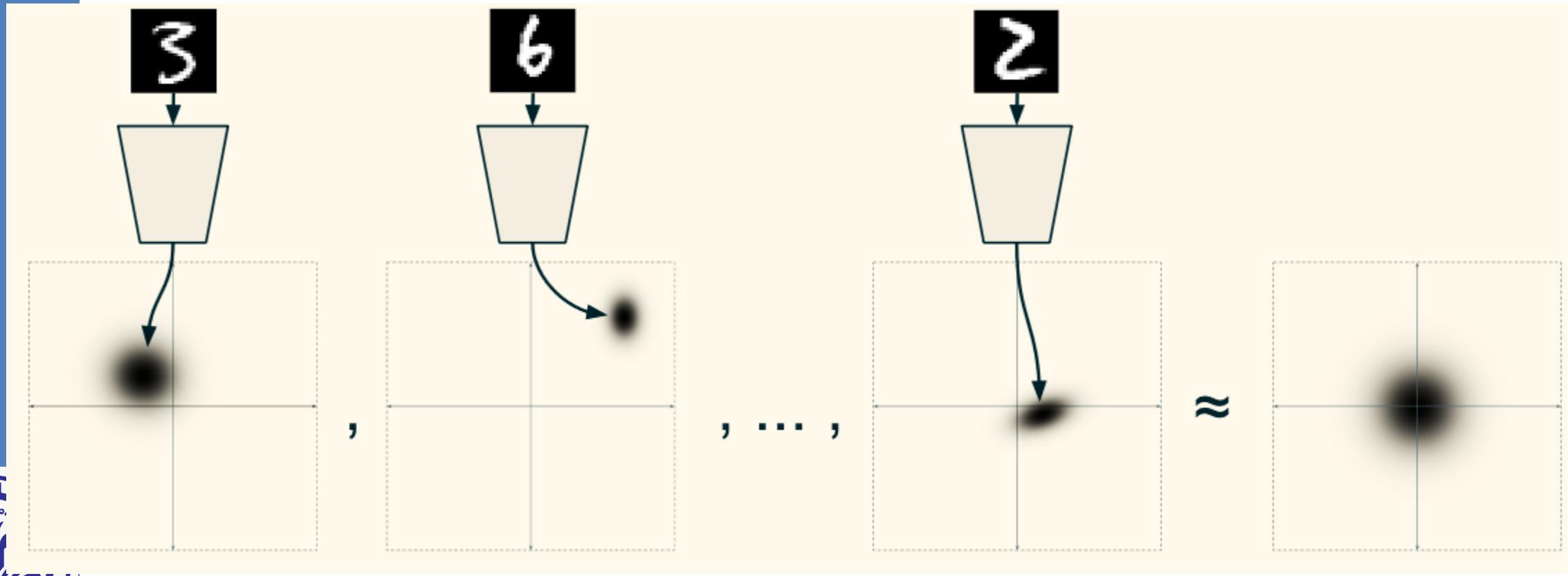
Variational AutoEncoder (VAE)

› VAE Encoder:



Variational AutoEncoder (VAE)

› VAE Encoder:



Conditional Variational AutoEncoder (CVAE)

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

$$01000001 \rightarrow \mathcal{A}$$

  - 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:

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

- 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)

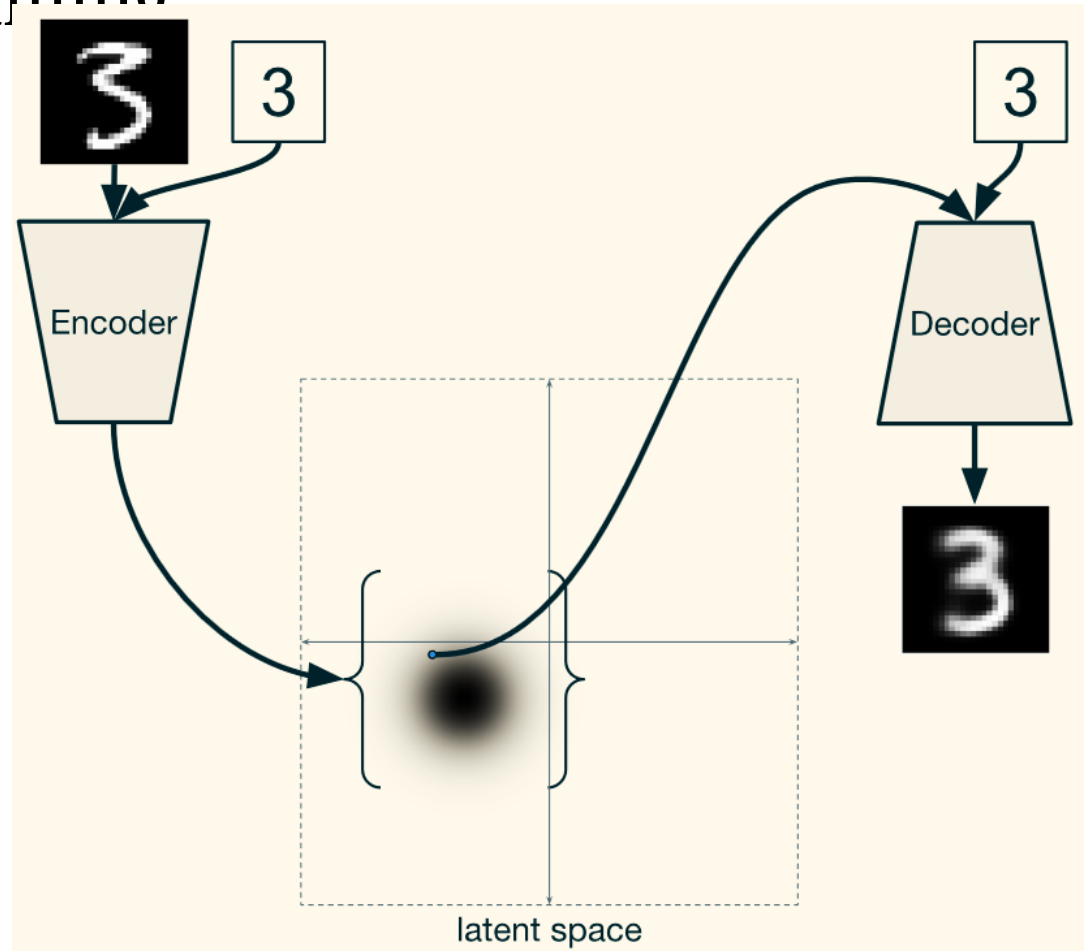
› How to incorporate c (suppose: a *one-hot* vector of *label*):

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

› Main idea: concatenation!

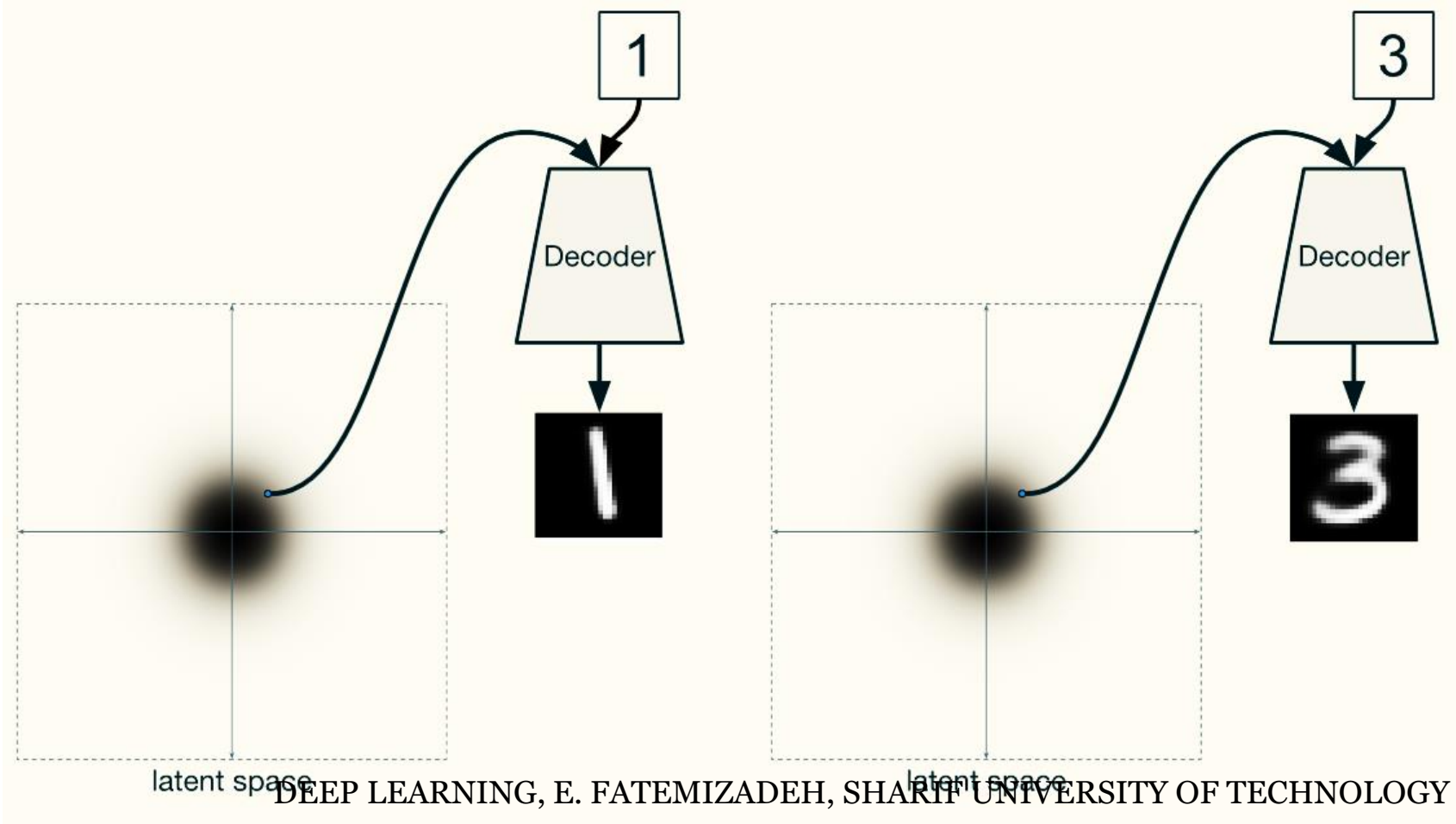
Conditional Variational AutoEncoder (CVAE)

› CVAE Training:



Conditional Variational AutoEncoder (CVAE)

› CVAE Test:



Conditional Variational AutoEncoder (CVAE)

› Let's talk about *concatenation* with code (encoder/decoder):

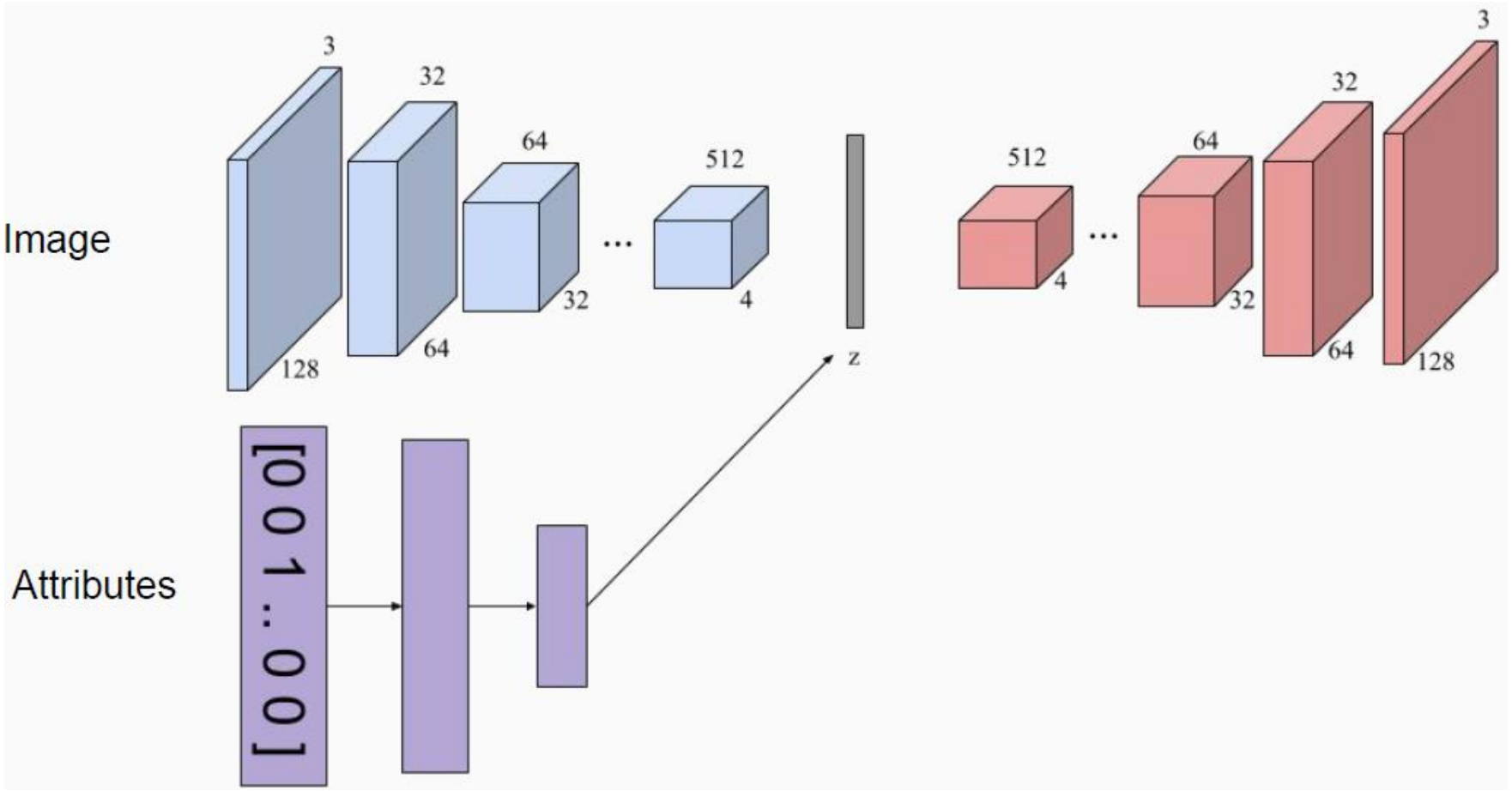
```
# Q(z|X,y) -- encoder
X = Input(batch_shape=(m, n_x))
cond = Input(batch_shape=(m, n_y))


inputs = merge([X, cond], mode='concat', concat_axis=1)


# Sample z ~ Q(z|X,y)
z = Lambda(sample_z)([mu, log_sigma])
z_cond = merge([z, cond], mode='concat', concat_axis=1) # <--- NEW!
```

Conditional Variational AutoEncoder (CVAE)

› An alternative:

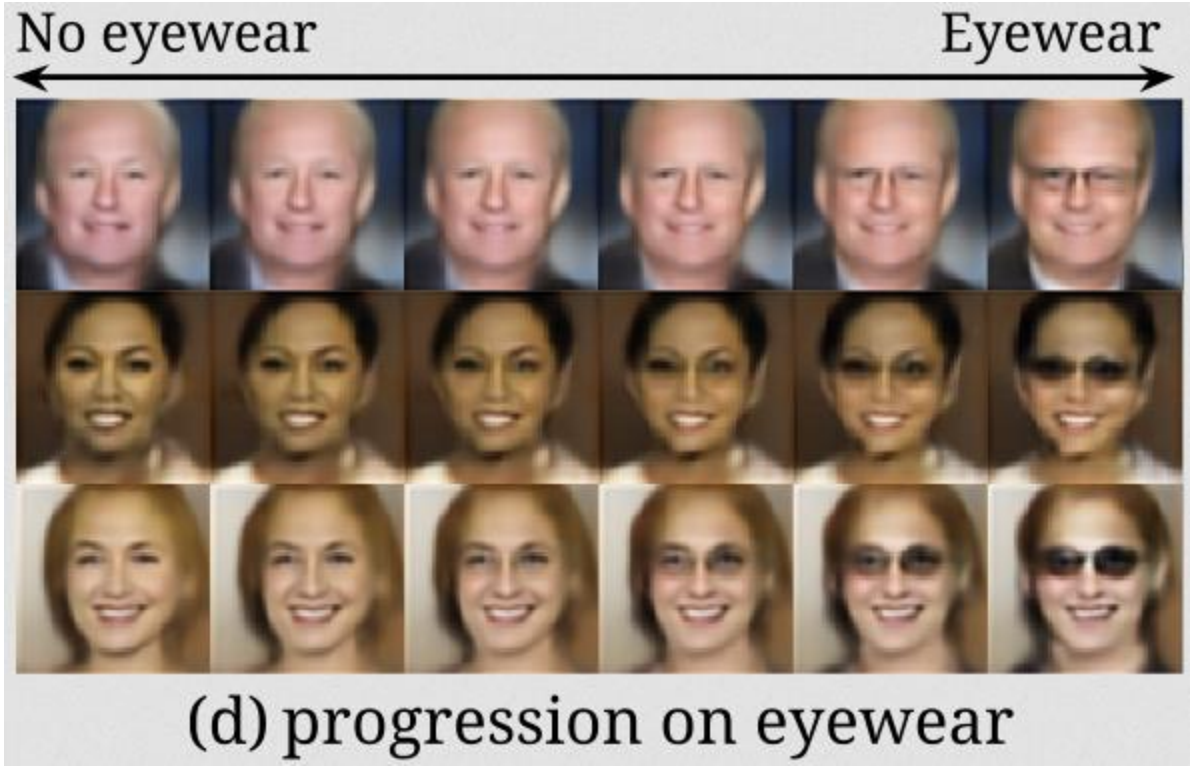


Conditional Variational AutoEncoder (CVAE)

- Crazy 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

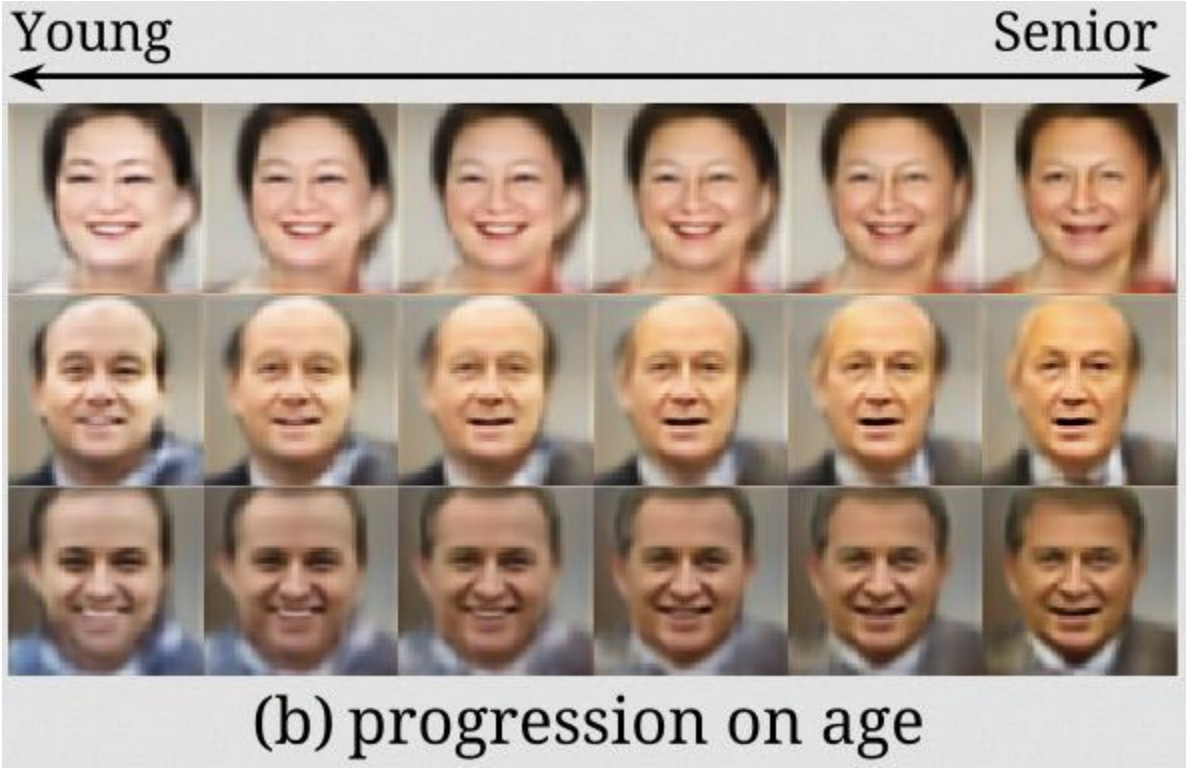
Conditional Variational AutoEncoder (CVAE)

› Attribute2Image



Conditional Variational AutoEncoder (CVAE)

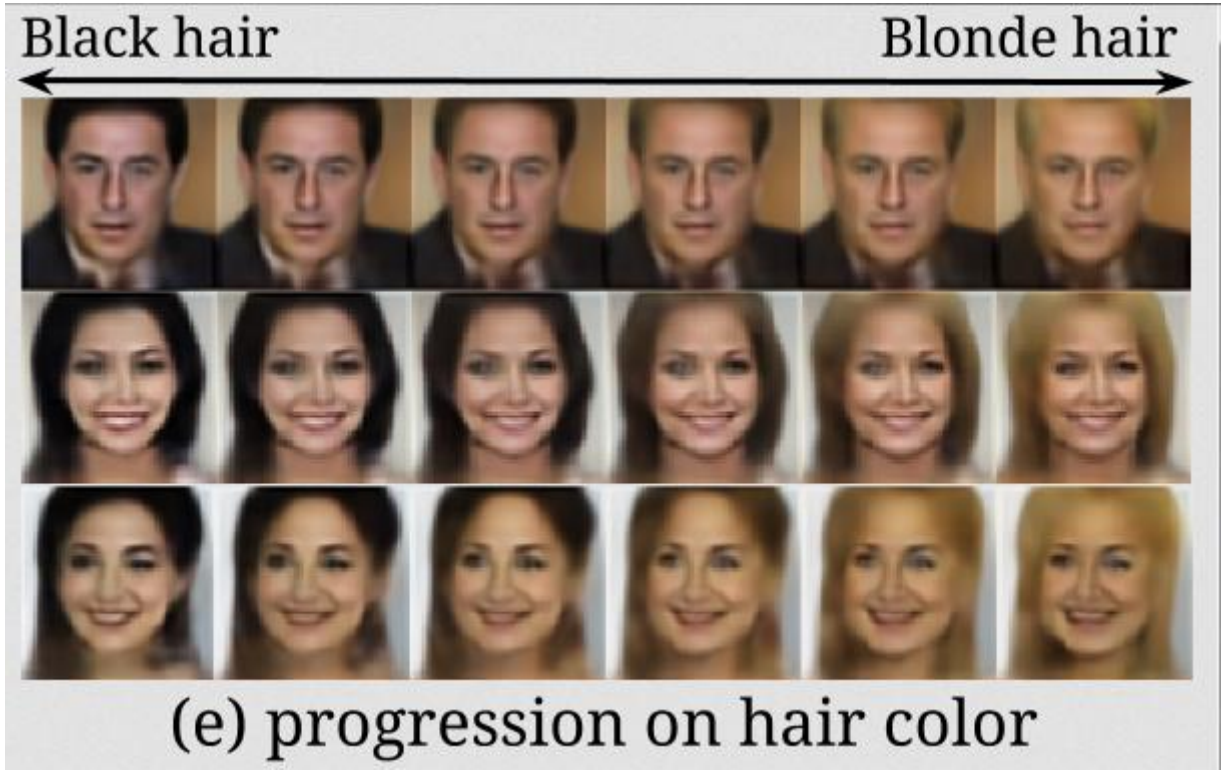
› Attribute2Image



(b) progression on age

Conditional Variational AutoEncoder (CVAE)

› Attribute2Image



VAE & CVAE

› AnY QuEsTiOn?