Autoencoder and Embedding

Seeing through water

Alexei A. Efros*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
efros@cs.cmu.edu

Volkan Isler, Jianbo Shi and Mirkó Visontai
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
{isleri,jshi,mirko}@cis.upenn.edu

Figure 1: Fifteen consecutive frames from the video. The experimental setup involved: a transparent bucket of water, the cover of a vision textbook "Computer Vision/A Modern Approach".

Figure 2: Ground truth image and reconstruction results using mean and median

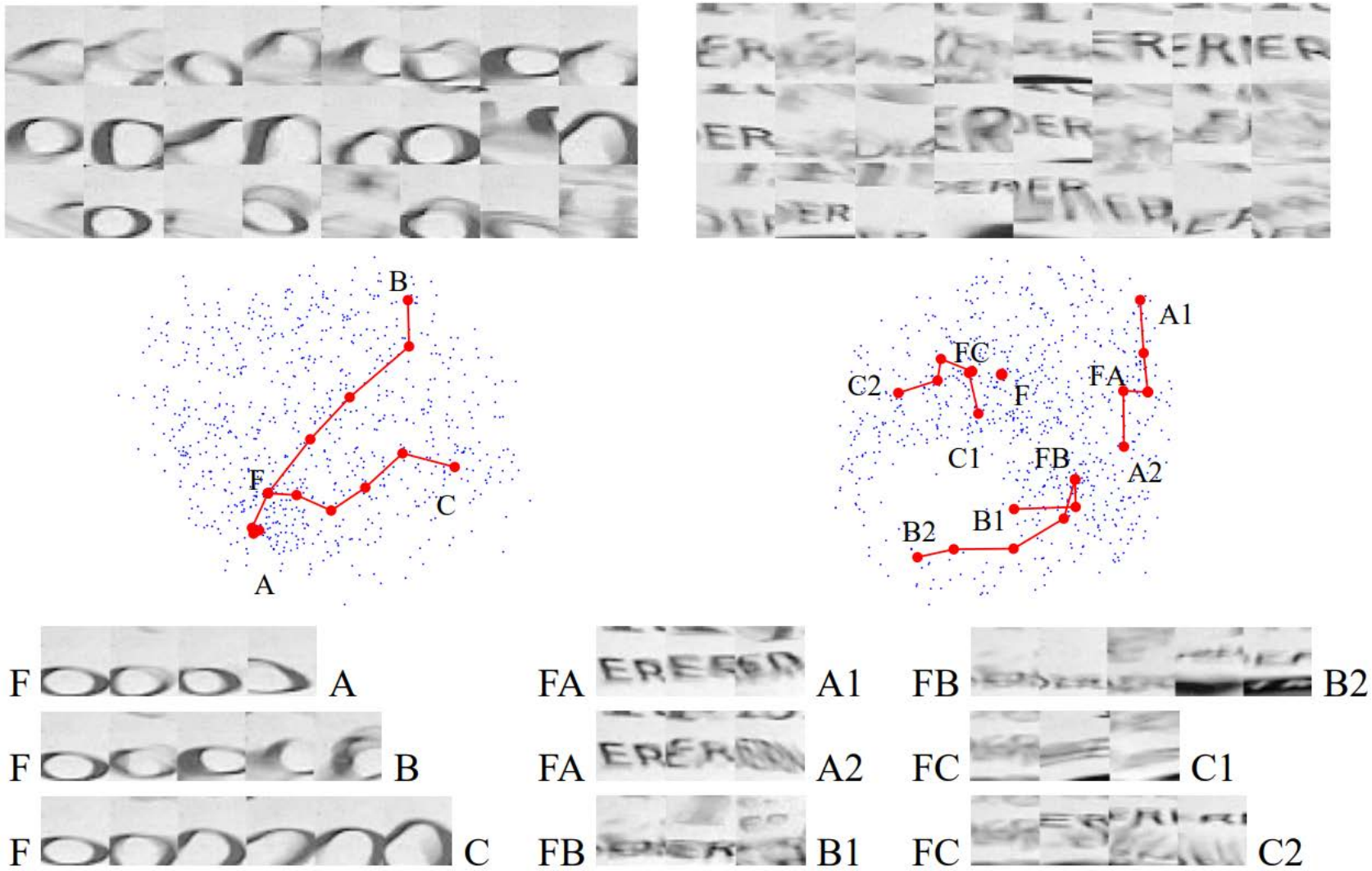
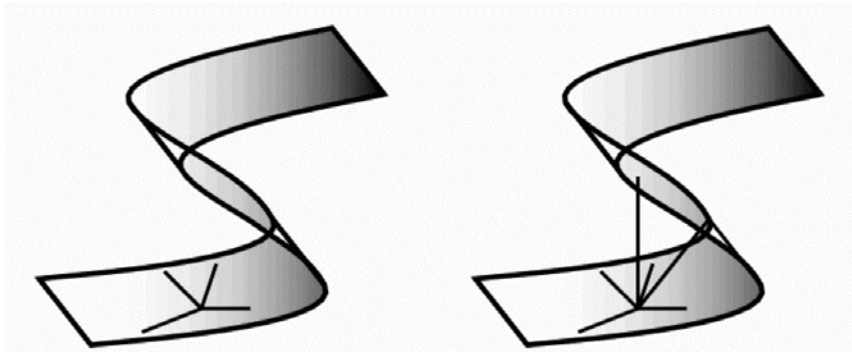
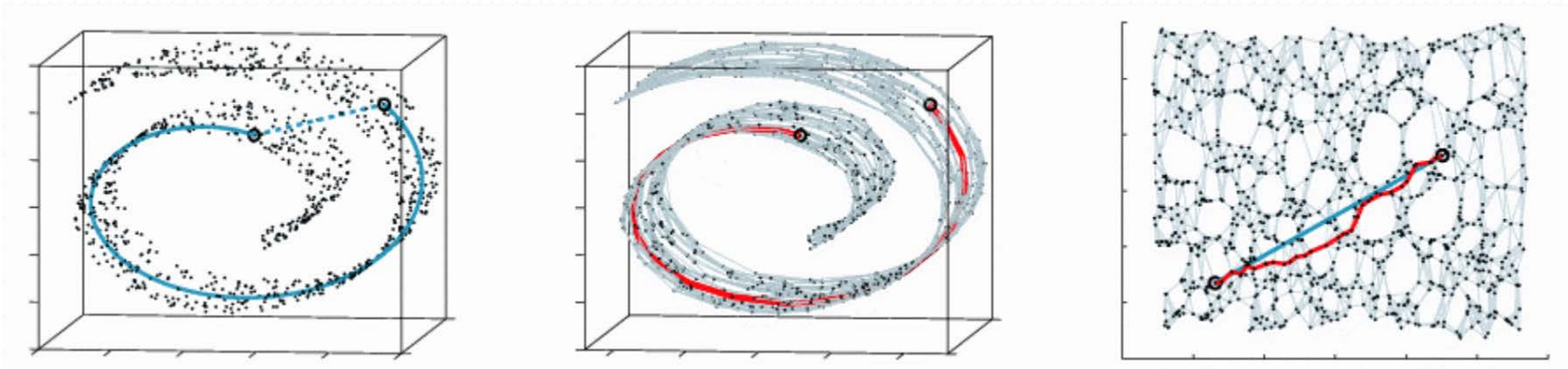


Figure 7: **Top row:** sample patches (two different locations) from 800 frames, **Middle row:** Convex flow embedding, showing the transition paths. **Bottom row:** corresponding patches (A, B, C, A1, A2, B1, B2, C1, C2) and the morphing of them to the centers F F, FA, FB, FC respectively

IsoMap

Find estimated geodesic distances between all pairs in X:





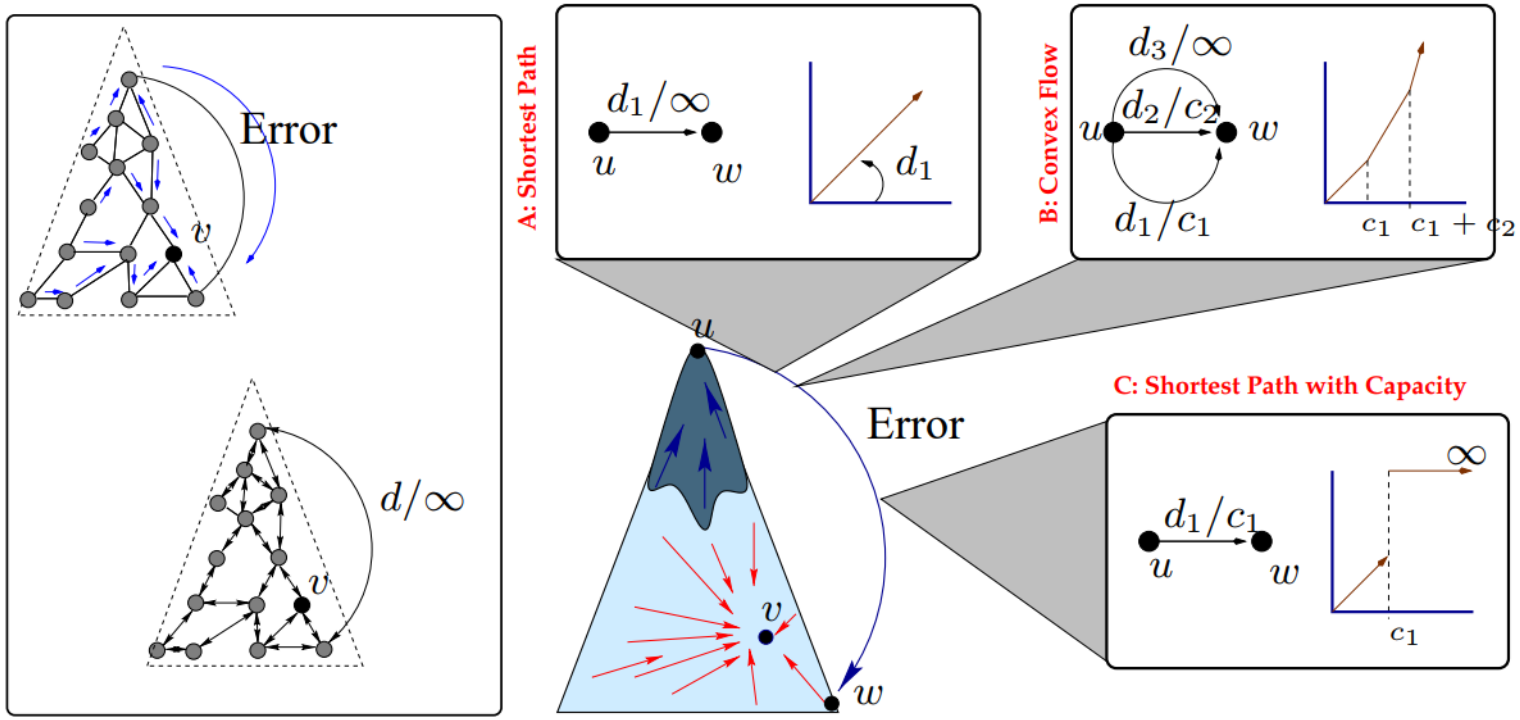


Figure 4: The leakage problem. **Left:** Equivalence of shortest path leakage and uncapacitated flow leakage problem. **Bottom-middle:** After the erroneous edge is inserted, the shortest paths from the top of the triangle to vertex v go through this edge. **Boxes A-C:**Alternatives for charging a unit of flow between nodes u and w. The horizontal axis of the plots is the amount of flow and the vertical axis is the cost. **Box A:** Linear flow. The cost of a unit of flow is d_1 **Box B:** Convex flow. Multiple edges are introduced between two nodes, with fixed capacity, and convexly increasing costs. The cost of a unit of flow increases from d_1 to d_2 and then to d_3 as the amount of flow from u to w increases. **Box C:** Linear flow with capacity. The cost is d_1 until a capacity of c_1 is achieved and becomes infinite afterwards.

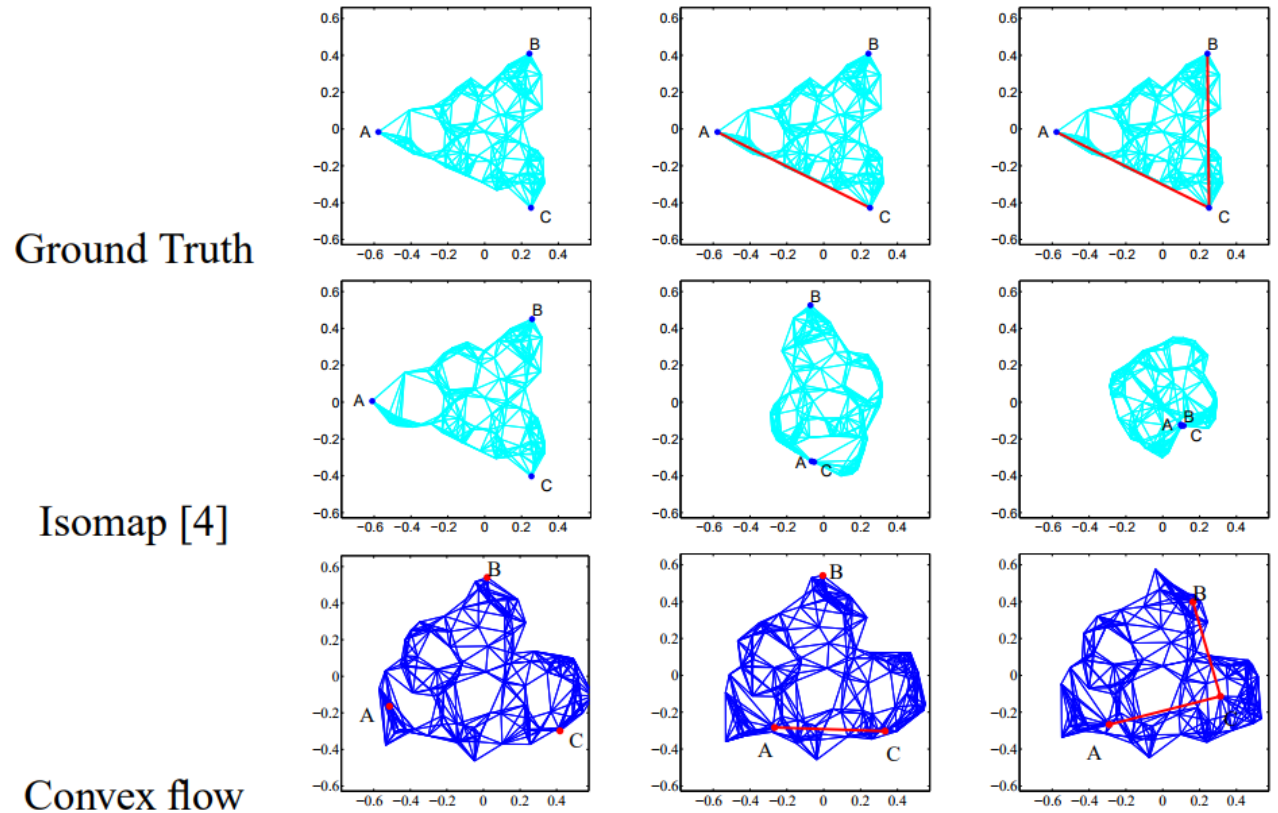


Figure 5: **Top row:** Ground truth. After sampling points from a triangular disk, a kNN graph is constructed to provide a local measure for the embedding (left). Additional erroneous edges AC and CB are added to perturb the local measure (middle, right). **Middle row:** Isomap embedding. Isomap recovers the manifold for the error-free cases (left). However, all-pairs shortest path can "leak" through AC and CB, resulting a significant change in the embedding. **Bottom row:** Convex flow embedding. Convex flow penalized too many paths going through the same edge – correcting the leakage problem. The resulting embedding is more resistant to perturbations in the kNN graph.

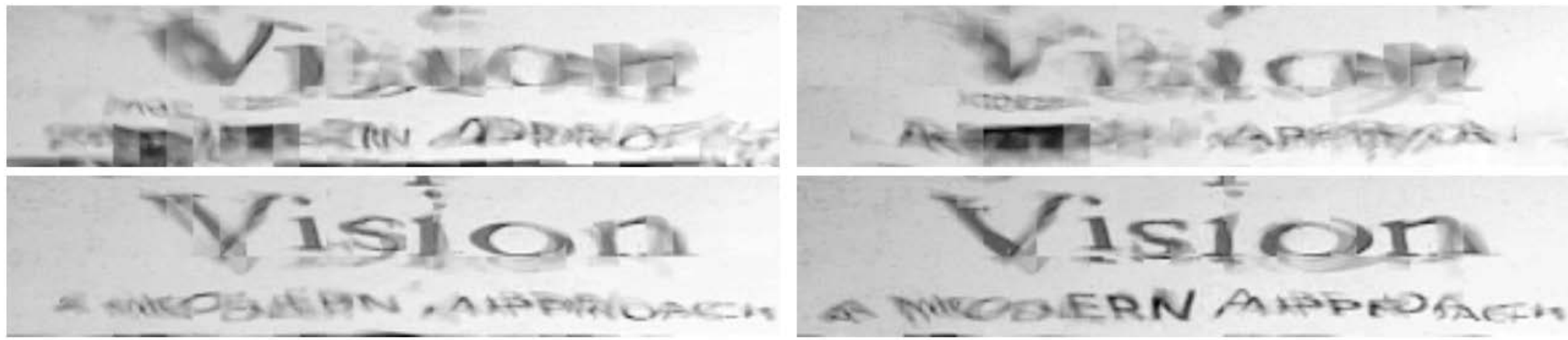
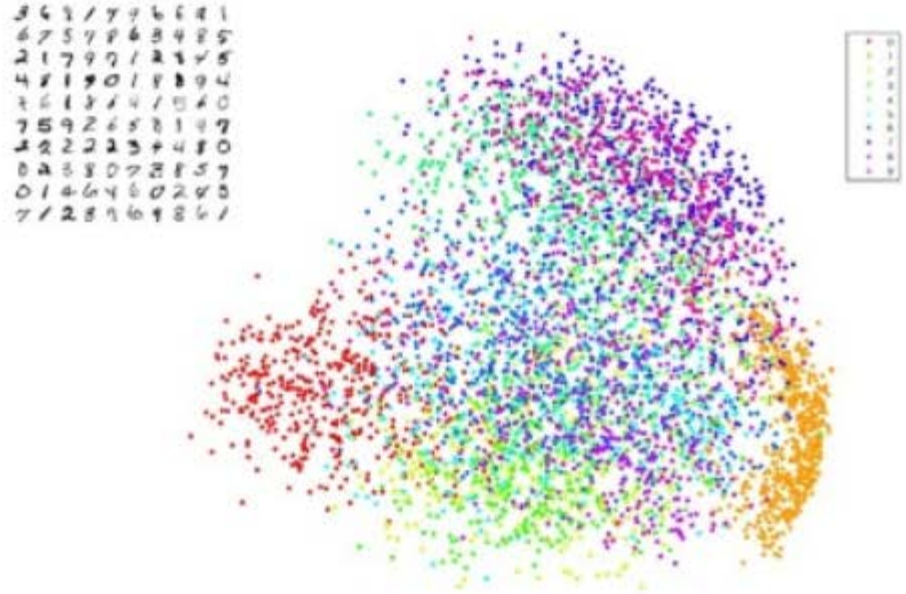
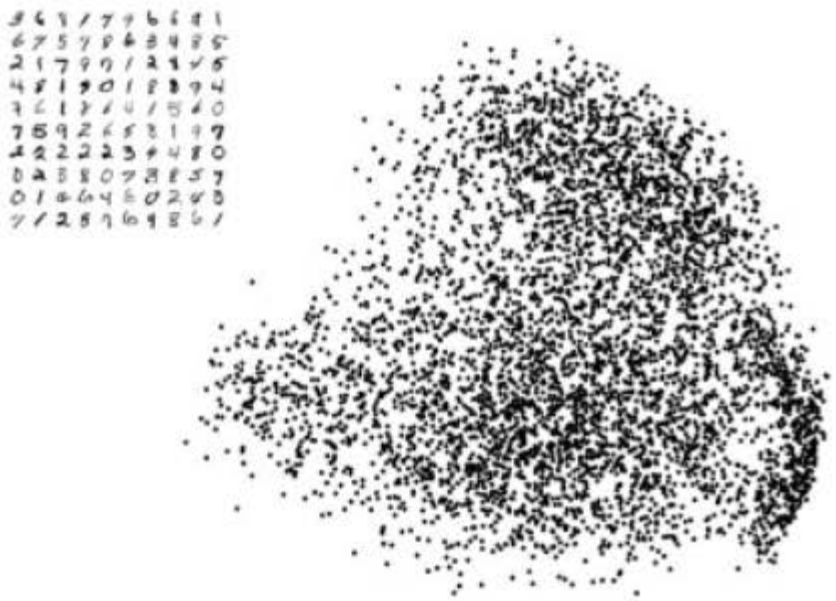


Figure 6: Comparison of reconstruction results of different methods using the first 800 frames, **top**: patches stitched together which are closest to mean (**left**) and median (**right**), **bottom**: our results using a single (**left**) and three (**right**) centers

Factorized embedding





This PCA visualization is terrible

Factorized embedding

Dimensionality Reduction
contd …

Aarti Singh

Machine Learning 10-601
Nov 10, 2011

Slides Courtesy: Tom Mitchell, Eric Xing, Lawrence Saul

Factorized embedding

Principal Components are the eigenvectors of the matrix of sample correlations XX^T of the data

New set of axes $V = [v_1, v_2, \ldots, v_D]$ where $XX^T = V\Lambda V^T$

- Geometrically: centering followed by rotation
  - Linear transformation

Original representation of data points

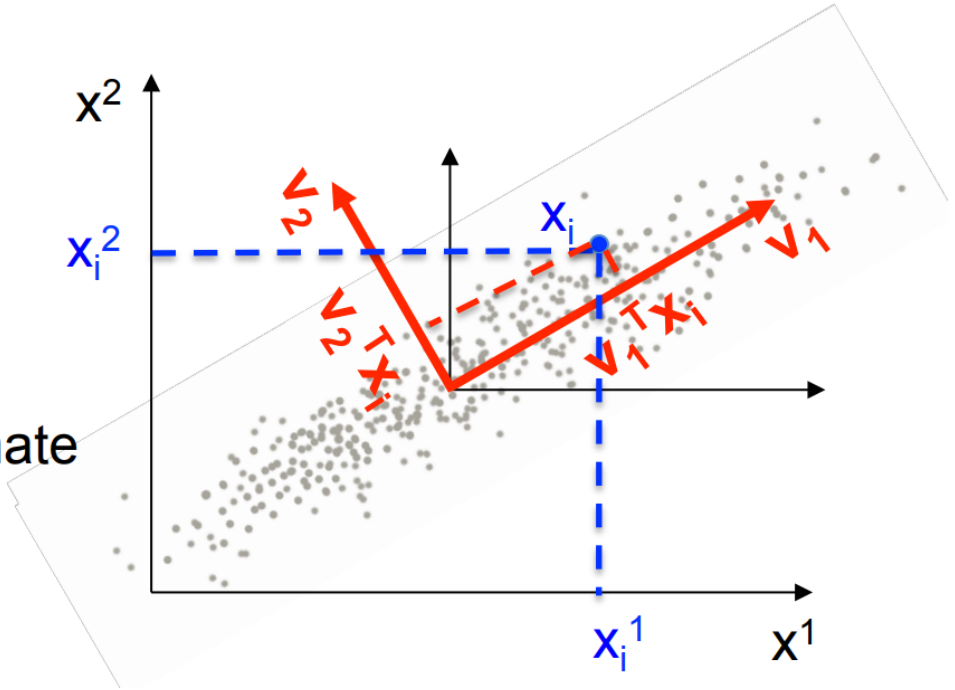
$x_i = [x_i^1, x_i^2, \ldots, x_i^D]$

$x_i^j = e_j^T x_i$ where $e_j = [0 \ldots 0\ \underset{\uparrow}{1}\ 0 \ldots 0]$

j^{th} coordinate

Transformed representation of data points

$[v_1^T x_i, v_2^T x_i, \ldots v_D^T x_i]$



Factorized embedding

Original Representation $[x_i^1, x_i^2, \ldots, x_i^D]$ (D-dimensional vector)

$$x_i = \sum_{j=1}^{D} x_i^j e_j = \sum_{j=1}^{D} (e_j^T x_i) e_j$$

$(x_i^j)^2 = (e_j^T x_i)^2$ = energy/variance of data point i along coordinate j

Transformed representation $[v_1^T x_i, v_2^T x_i, \ldots v_D^T x_i]$ (D-dimensional vector)

$$x_i = \sum_{j=1}^{D} (v_j^T x_i) v_j$$

$(v_j^T x_i)^2$ = energy/variance of data point i along principal component v_j

$$\lambda_j = \sum_{i=1}^{n} (v_j^T x_i)^2 = \text{energy/variance of all points along } v_j$$

Dimensionality reduction $[v_1^T x_i, v_2^T x_i, \ldots v_d^T x_i]$ (d-dimensional vector)

$$\hat{x}_i = \sum_{j=1}^{d} (v_j^T x_i) v_j$$

Only keep data projections onto principal components which capture enough energy/variance of the data $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_D$

Factorized embedding

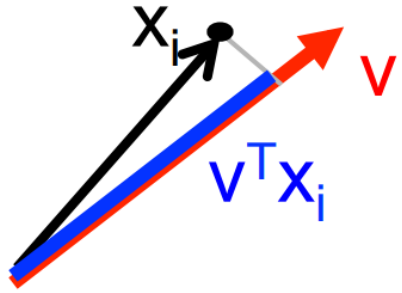
Maximum Variance Subspace: PCA finds vectors v such that projections on to the vectors capture maximum variance in the data

$$\sum_{i=1}^{n} (\mathbf{v}^T \mathbf{x}_i)^2 = \mathbf{v}^T \mathbf{X}\mathbf{X}^T \mathbf{v}$$

Minimum Reconstruction Error: PCA finds vectors v such that projection on to the vectors yields minimum MSE reconstruction

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \underbrace{(\mathbf{v}^T \mathbf{x}_i)\mathbf{v}}\|^2$$

One direction approximation



Recall: $\mathbf{x}_i = \sum_k (\mathbf{v}_k^T \mathbf{x}_i) \cdot \mathbf{v}_k$

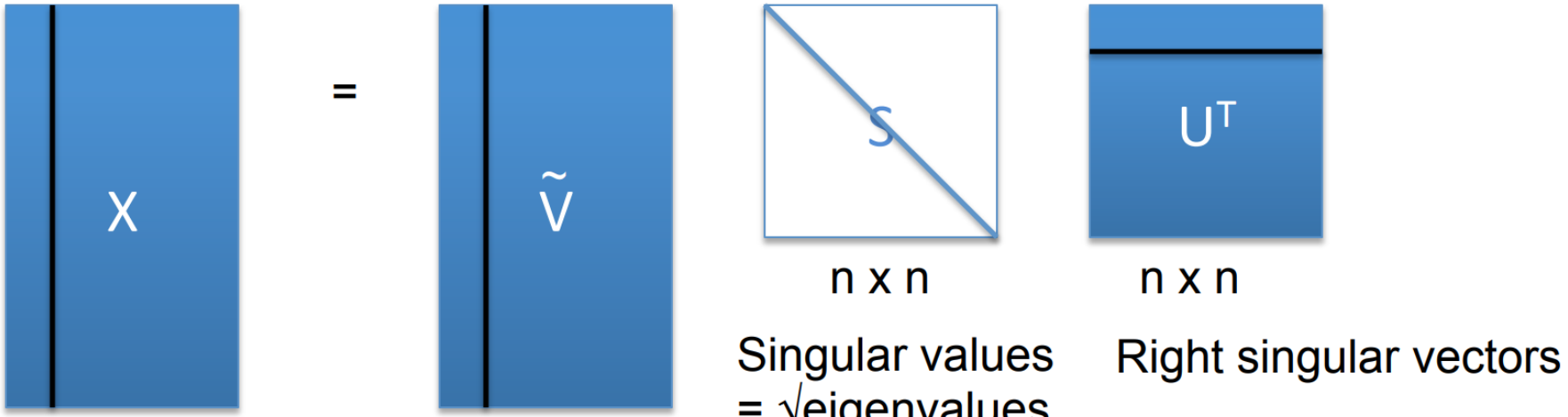
Factorized embedding

Principal Components – Eigenvectors of XX^T (D x D matrix)

Problematic for high-dimensional datasets!

Another way to compute PCs: **Singular Vector Decomposition (SVD)**

$$X = \tilde{V} S U^T \Rightarrow XX^T = \tilde{V} S U^T U S \tilde{V}^T = \tilde{V} S^2 \tilde{V}^T = \tilde{V} \Lambda \tilde{V}^T$$



D x n

D x n

n x n

n x n

Singular values = √eigenvalues

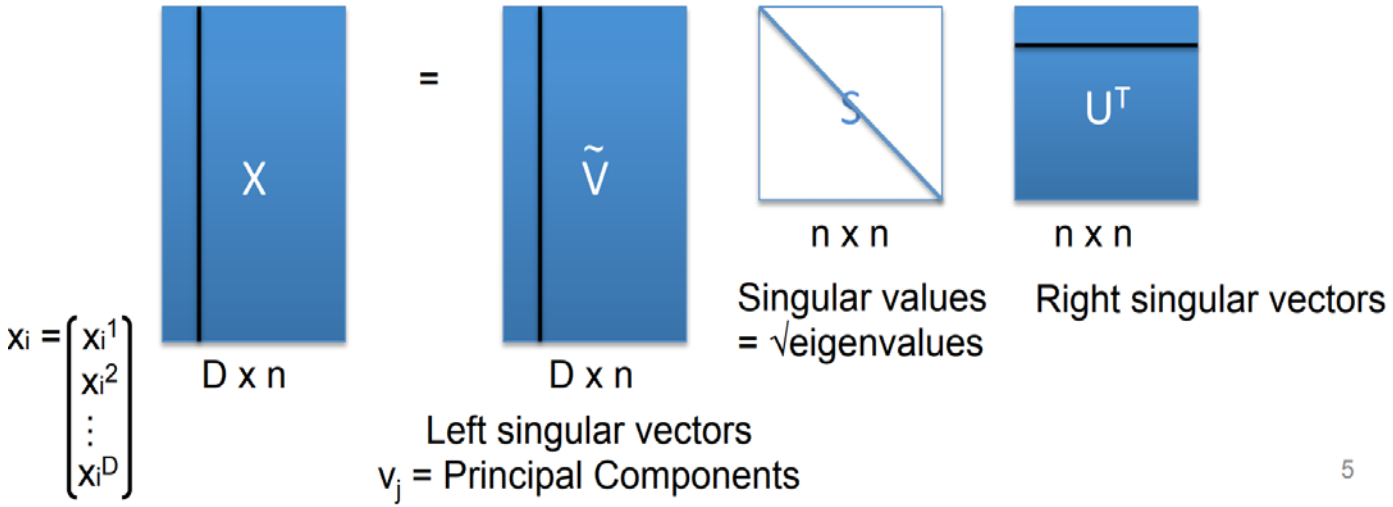
Right singular vectors

$x_i = \begin{bmatrix} x_i^1 \\ x_i^2 \\ \vdots \\ x_i^D \end{bmatrix}$

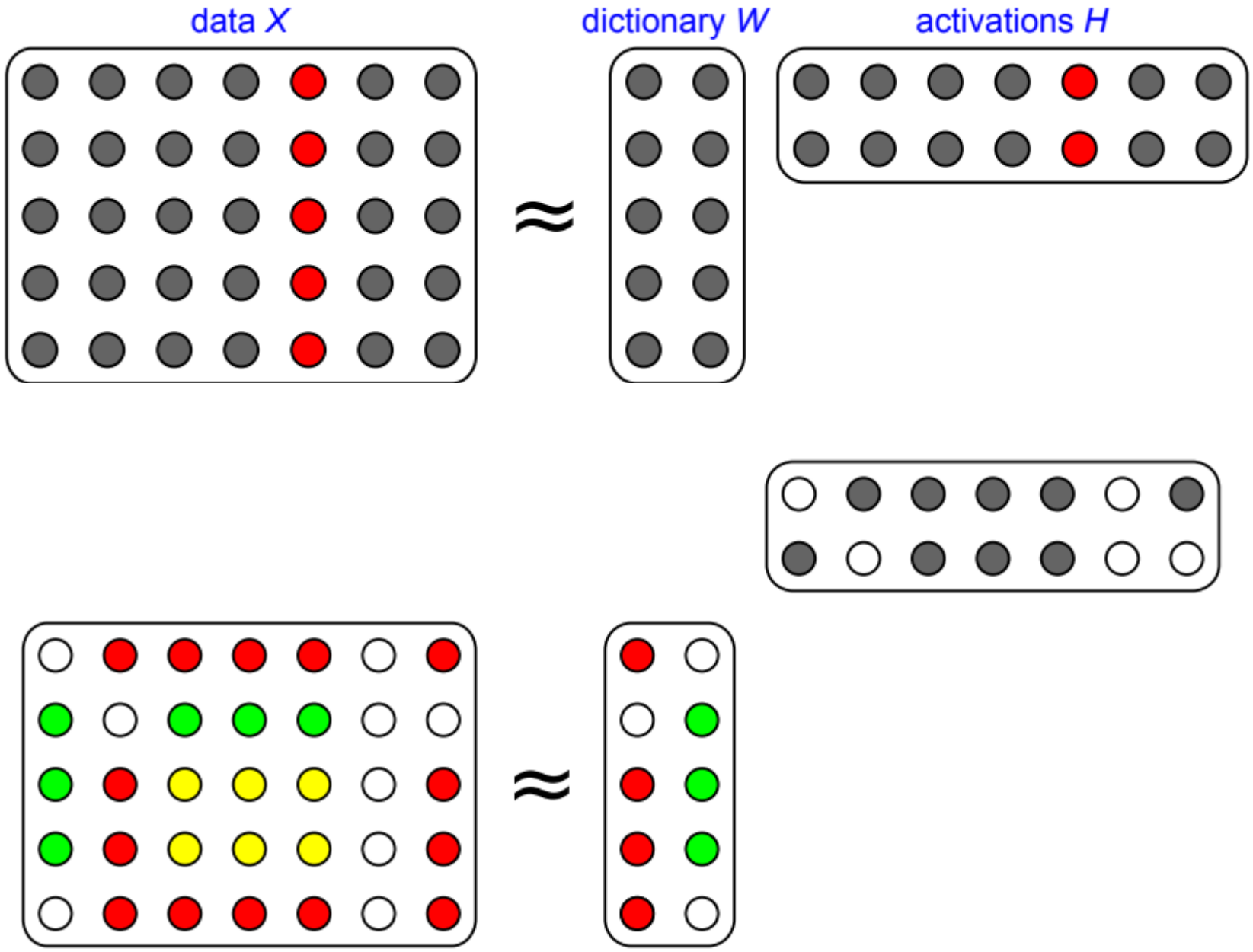
Left singular vectors
v_j = Principal Components

Factorized embedding

$$X = \tilde{V}SU^T \Rightarrow XX^T = \tilde{V}SU^TUS\tilde{V}^T = \tilde{V}S^2\tilde{V}^T = \tilde{V}\Lambda\tilde{V}^T$$



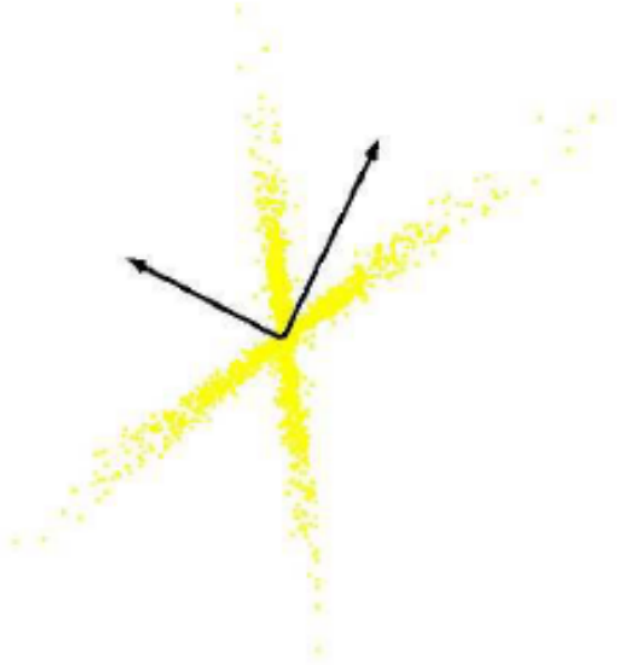
Factorized embedding



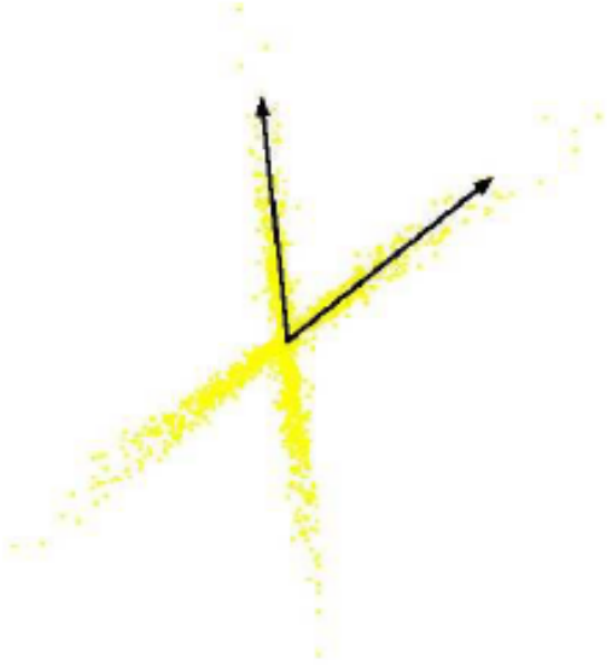
Factorized embedding

- PCA seeks “orthogonal” directions that capture maximum variance in data, or that minimize squared reconstruction error.

- ICA seeks “statistically independent” directions in the data



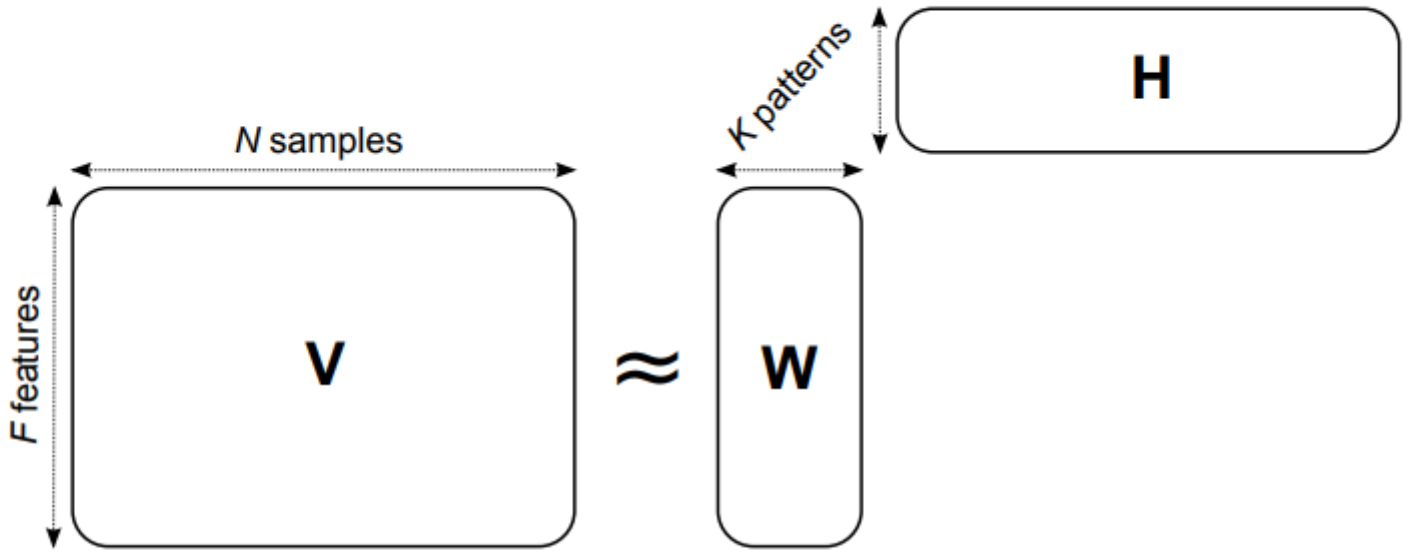
PCA
(orthogonal coordinate)



ICA
(non-orthogonal coordinate)

Factorized embedding

Nonnegative matrix factorisation

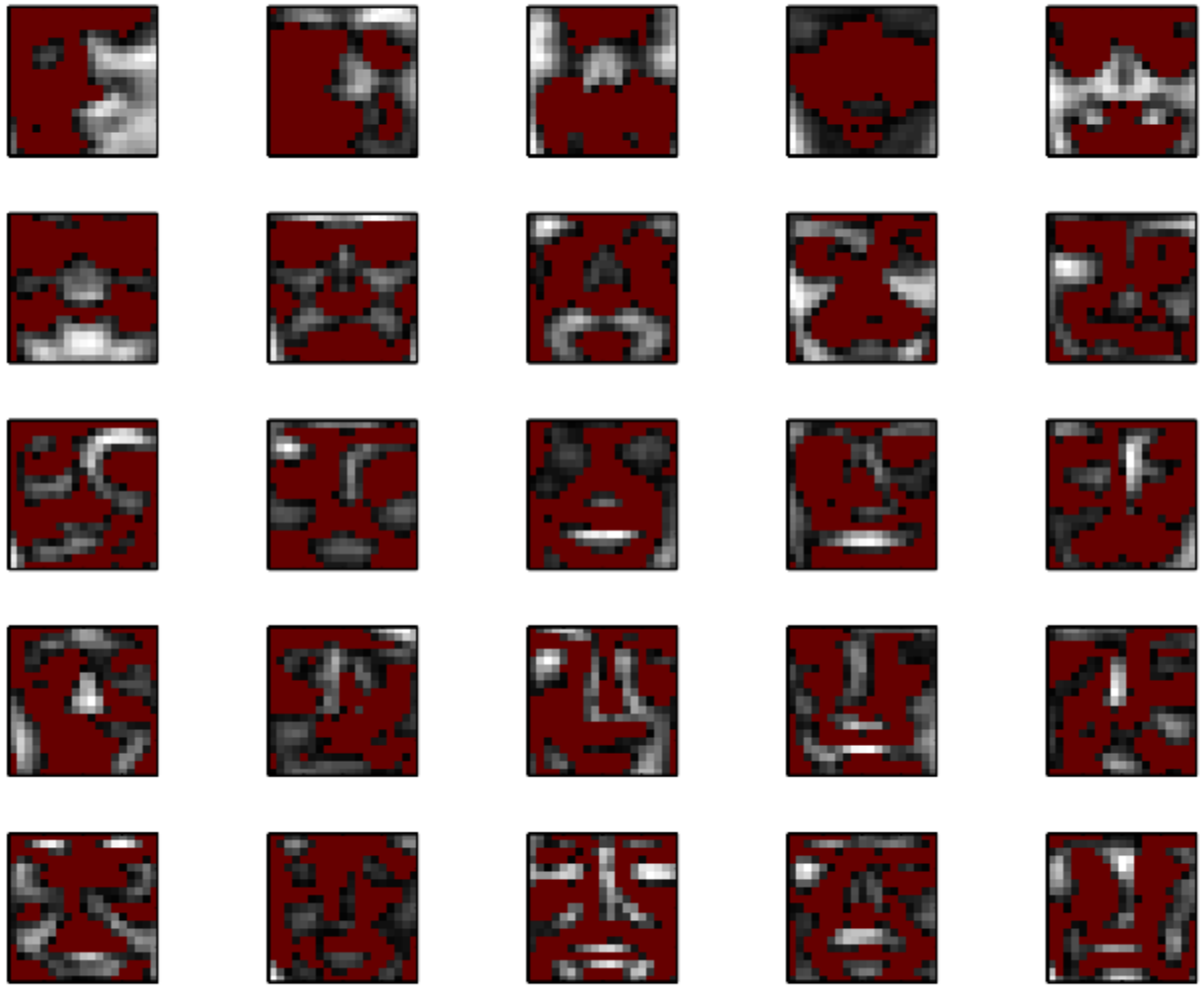


- data **V** and factors **W**, **H** have nonnegative entries.
- nonnegativity of **W** ensures interpretability of the dictionary, because patterns $\mathbf{w}_k$ and samples $\mathbf{v}_n$ belong to the same space.
- nonnegativity of **H** tends to produce part-based representations, because subtractive combinations are forbidden.

Early work by Paatero and Tapper (1994), landmark *Nature* paper by Lee and Seung (1999)

Factorized embedding

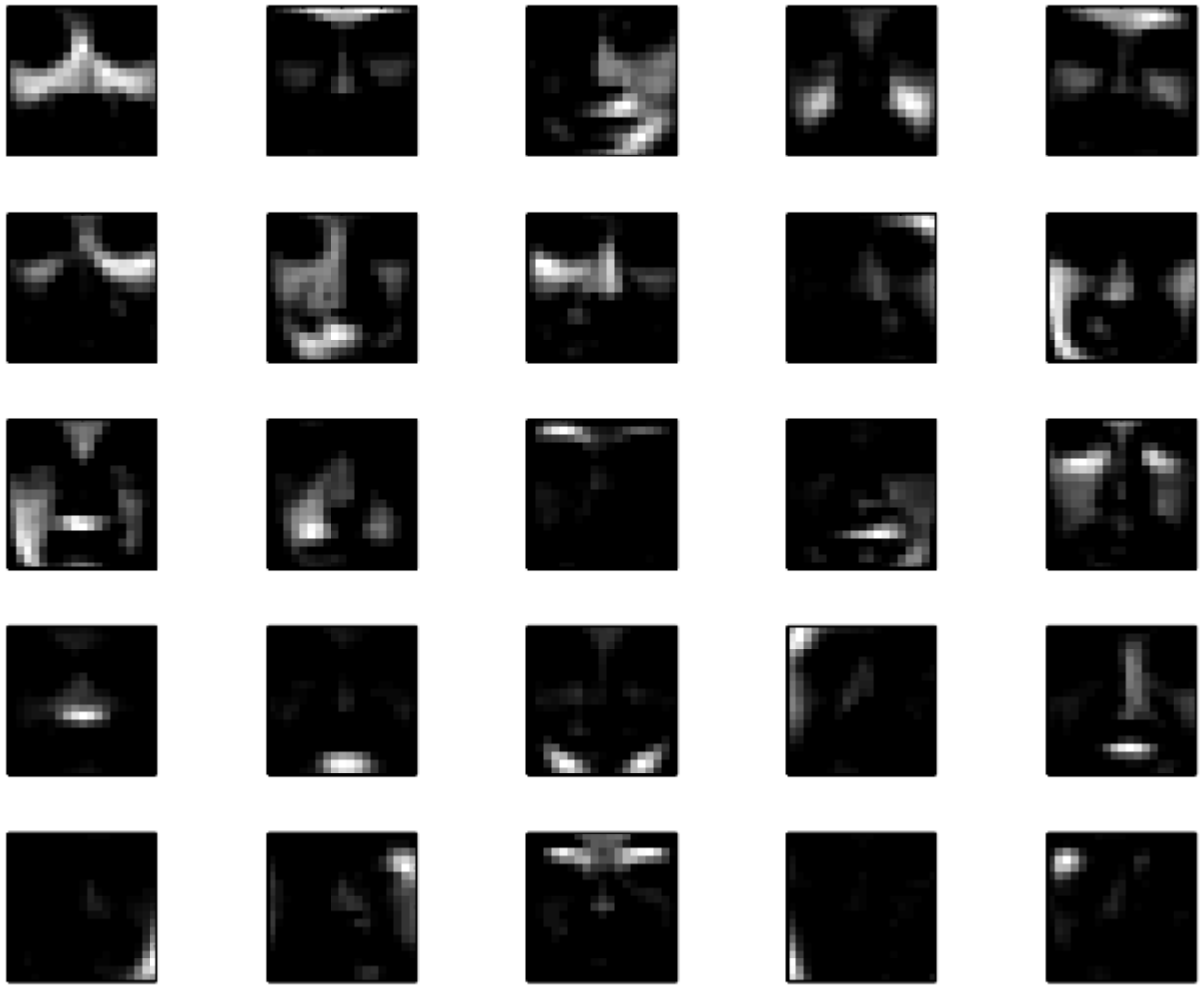
PCA dictionary with $K = 25$



red pixels indicate negative values

Factorized embedding

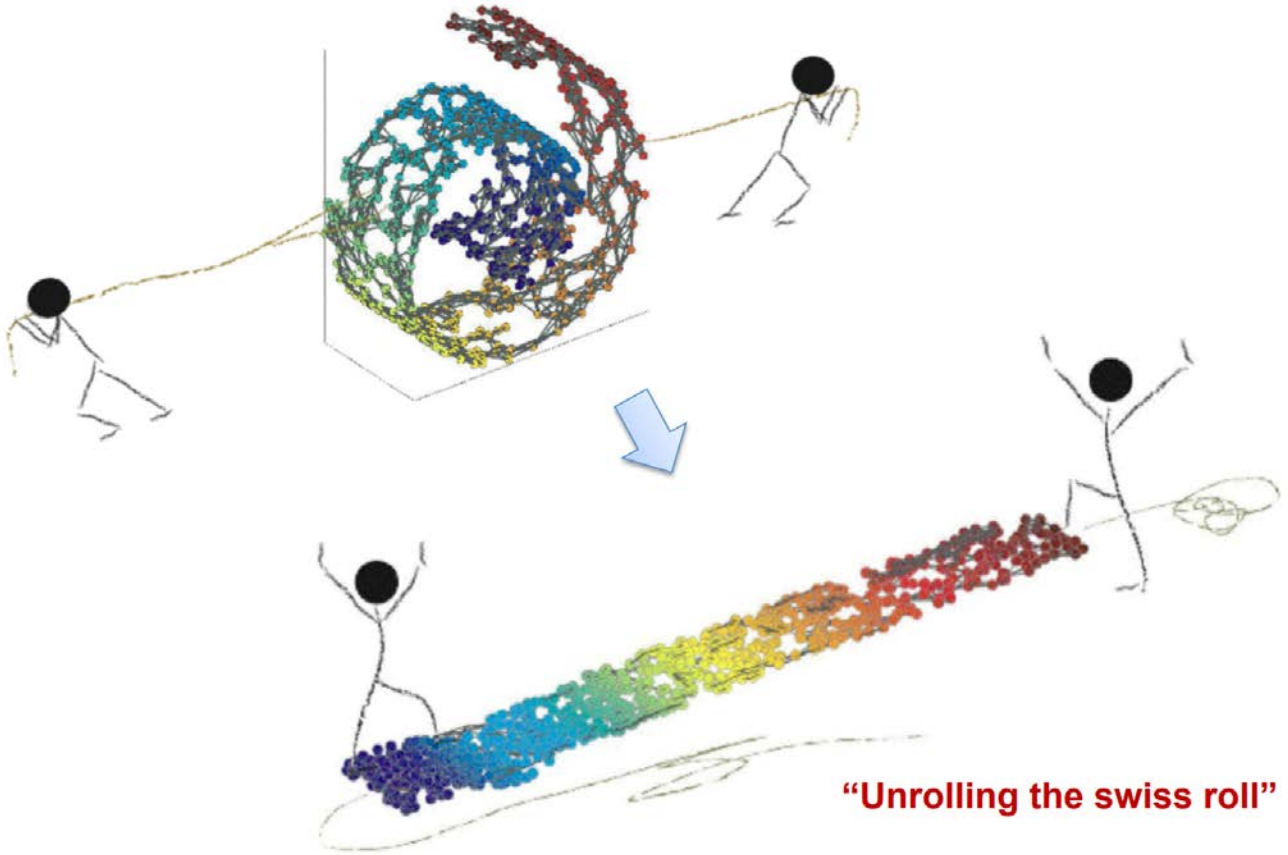
NMF dictionary with $K = 25$



experiment reproduced from (Lee and Seung, 1999)

unsupervised embedding

Dimensionality Reduction

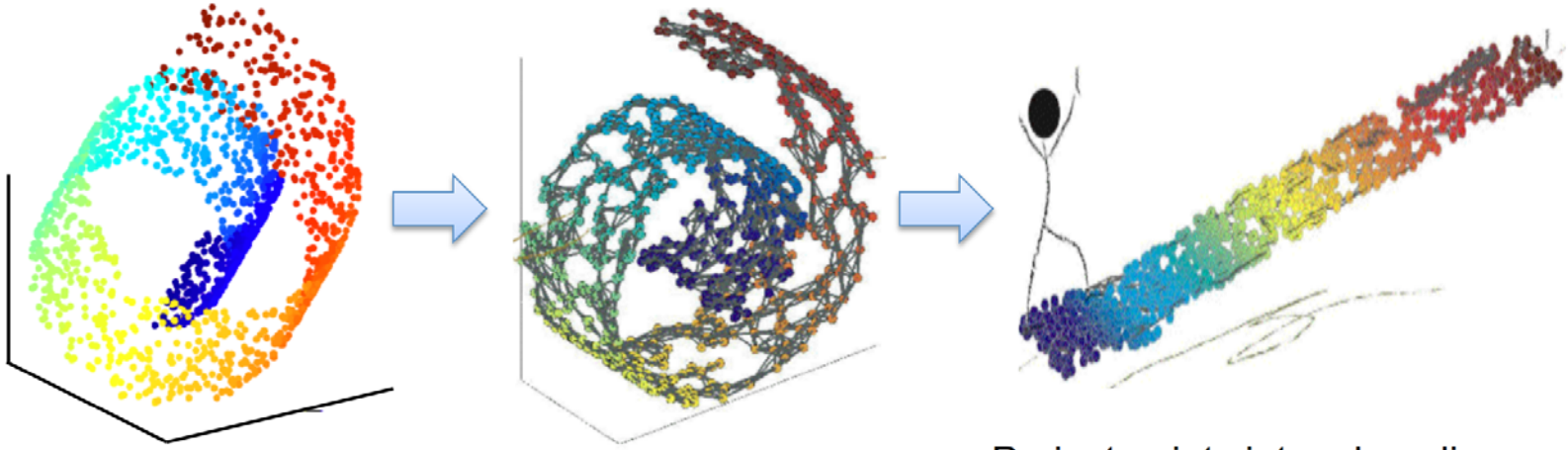


unsupervised embedding

Laplacian Eigenmaps

Linear methods – Lower-dimensional linear projection that preserves distances between **all** points

Laplacian Eigenmaps (key idea) – preserve **local** information only

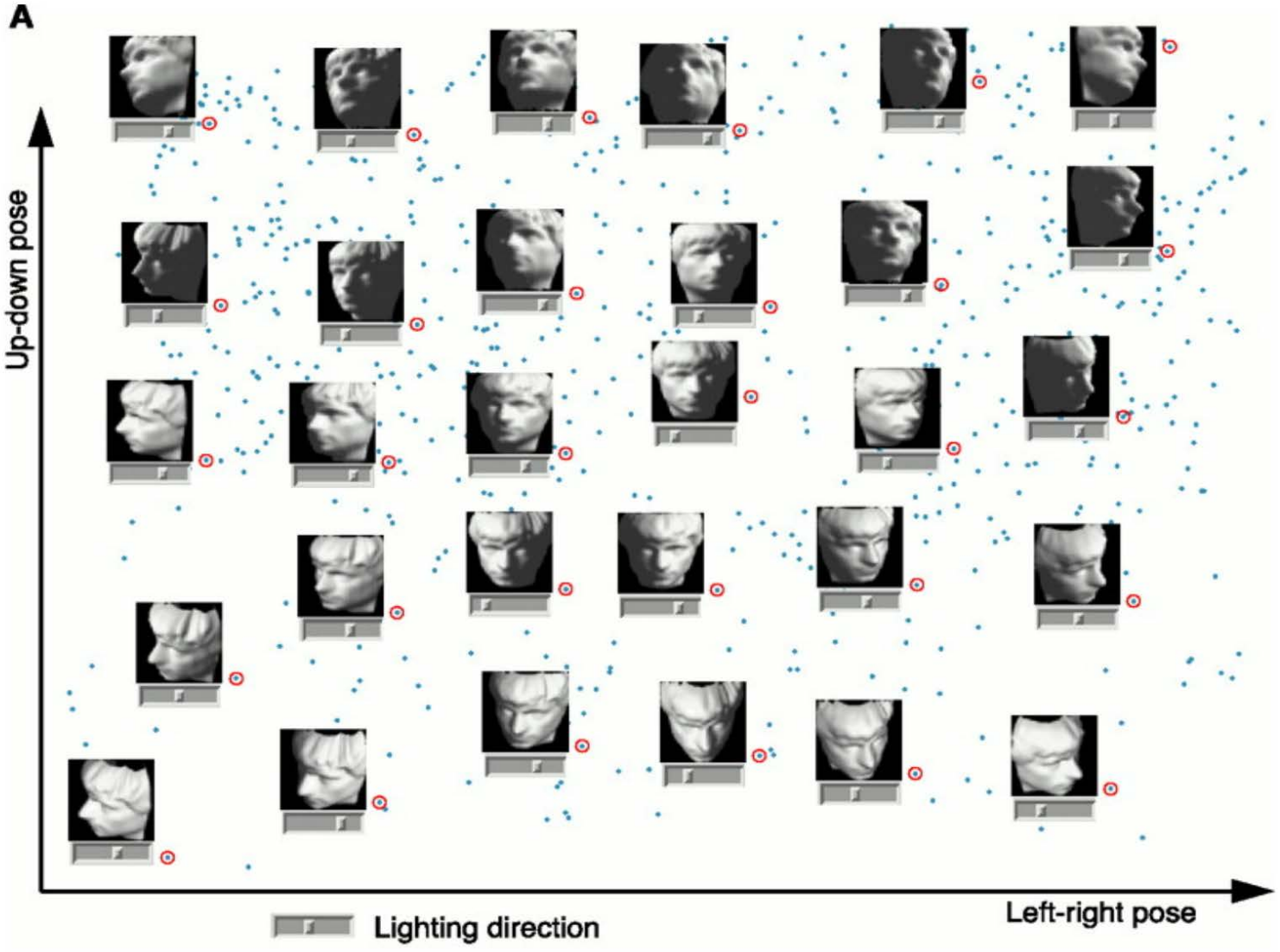


Construct graph from data points (capture local information)

Project points into a low-dim space using “eigenvectors of the graph”

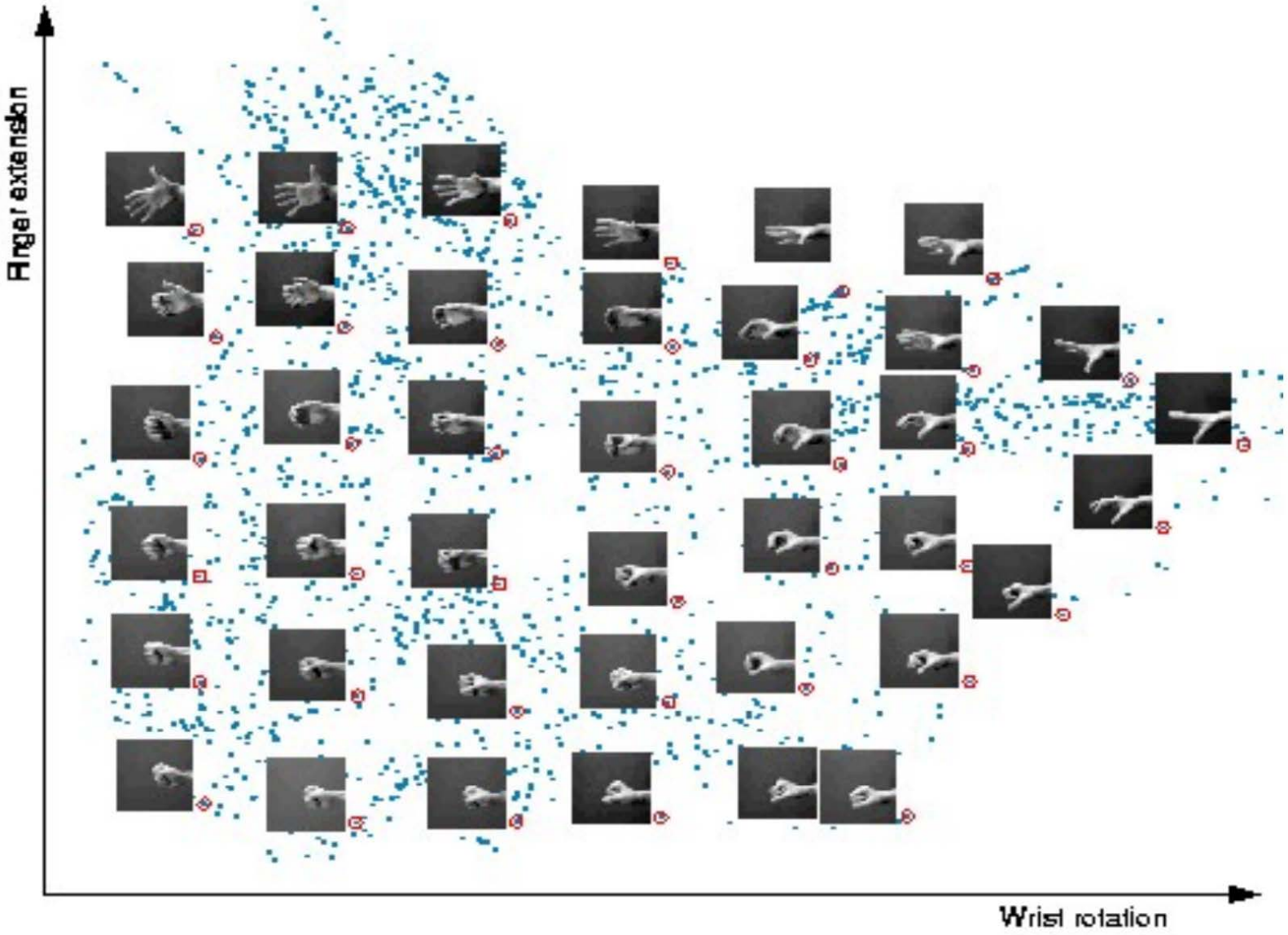
unsupervised embedding

Nonlinear Embedding Results



unsupervised embedding

Nonlinear Embedding Results



unsupervised embedding

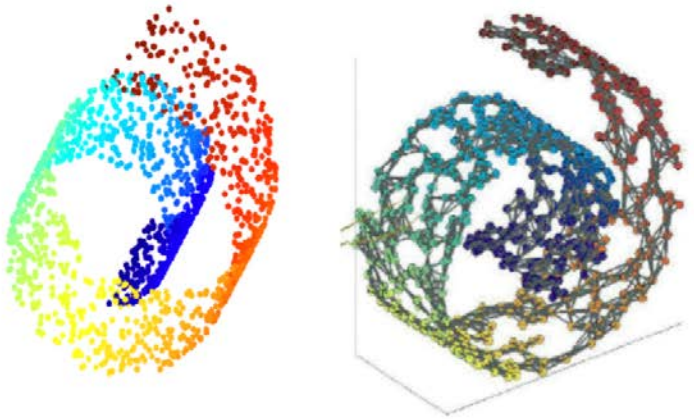
Step 1 - Graph Construction

Similarity Graphs: Model local neighborhood relations between data points

G(V,E,W)

V – Vertices (Data points)

E – Edges



(1) E – Edge if $||x_i - x_j|| \leq \varepsilon$ (ε – neighborhood graph)

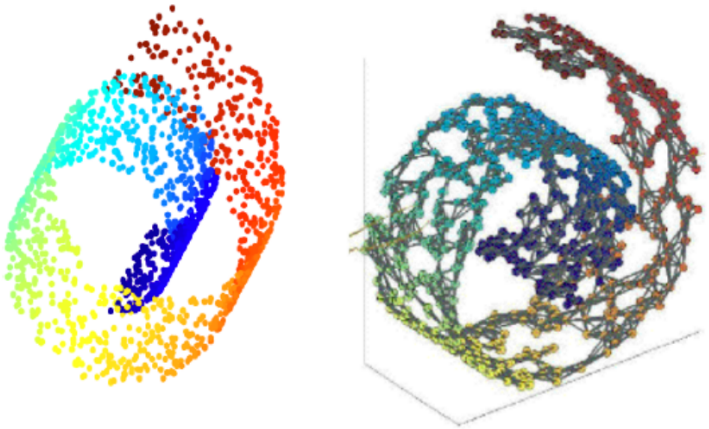
(2) E – Edge if k-nearest neighbor (k-NN graph)

unsupervised embedding

Step 1 - Graph Construction

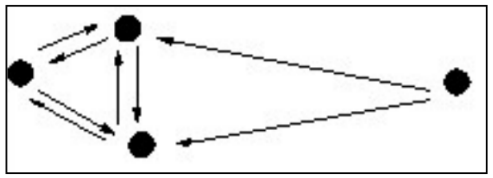
Similarity Graphs: Model local neighborhood relations between data points

(2) E – Edge if k-nearest neighbor (k-NN)

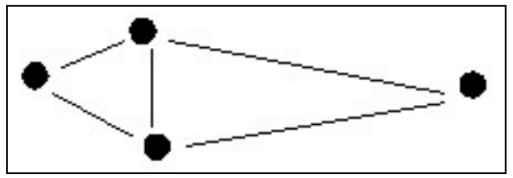


yields directed graph

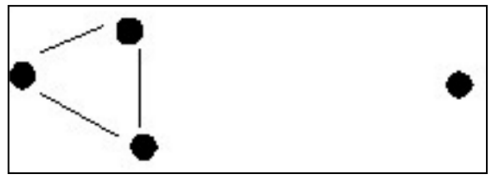
connect A with B if $A \rightarrow B$ OR $A \leftarrow B$ (symmetric kNN graph)
connect A with B if $A \rightarrow B$ AND $A \leftarrow B$ (mutual kNN graph)



Directed nearest neighbors



(symmetric) kNN graph



mutual kNN graph

unsupervised embedding

Step 1 - Graph Construction

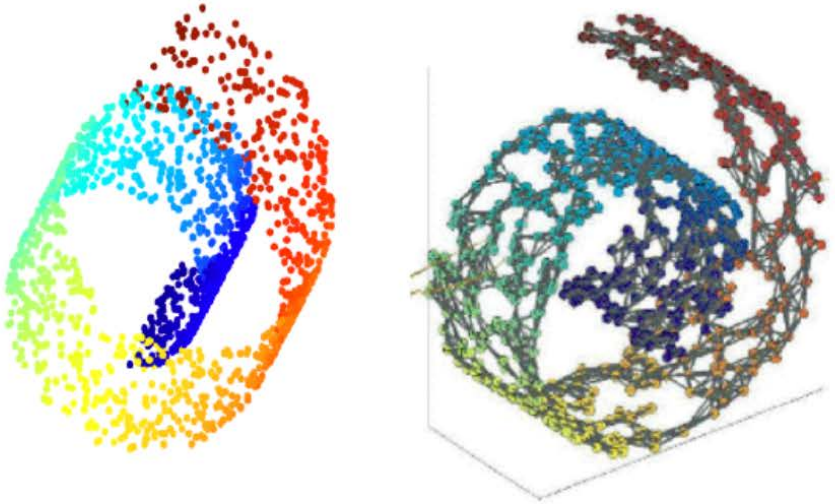
Similarity Graphs: Model local neighborhood relations between data points

G(V,E,W)

V – Vertices (Data points)

E – Edges

W – Weights



(1) W - W_{ij} = 1 if edge present, 0 otherwise

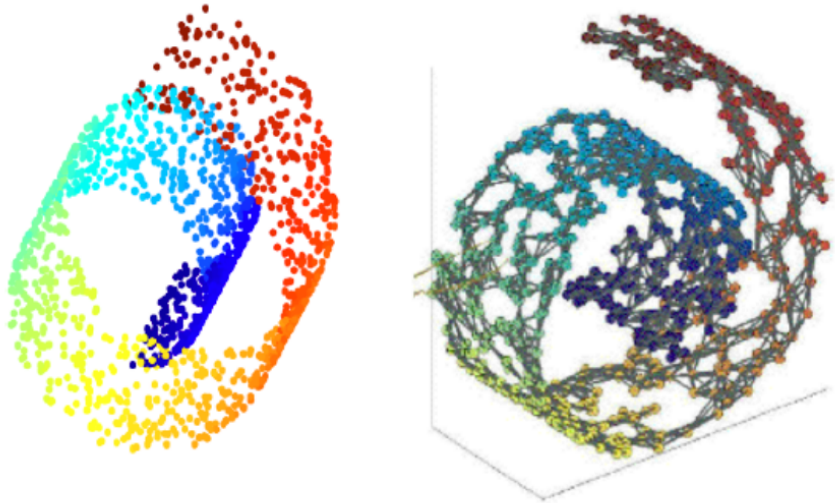
(2) W - $W_{ij} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$ Gaussian kernel similarity function (aka Heat kernel)

unsupervised embedding

Original Representation		Transformed representation
data point		projections
x_i	→	$(f_1(i), \ldots, f_d(i))$
(D-dimensional vector)		(d-dimensional vector)

Basic Idea: Find vector **f** such that, if x_i is close to x_j in the graph (i.e. W_{ij} is large), then projections of the points f(i) and f(j) are close

$$\min_{\mathbf{f}} \sum_{ij} W_{ij}(\mathbf{f}_i - \mathbf{f}_j)^2$$

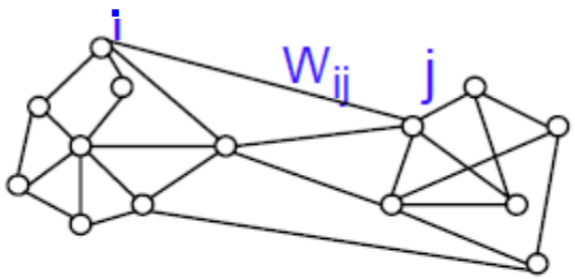


unsupervised embedding

Step 2 – Projection using Graph

- Graph Laplacian (unnormalized version)

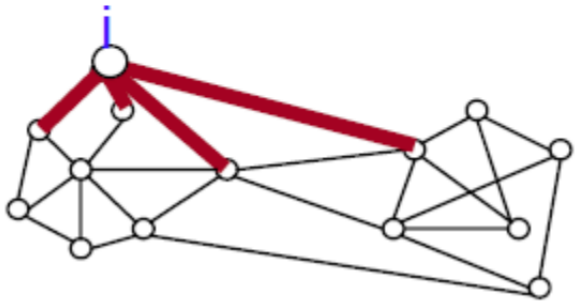
  L = D – W (n x n matrix)



W – Weight matrix

D – Degree matrix = $diag(d_1, \ldots, d_n)$

$d_i = \sum_j w_{ij}$ degree of a vertex



Note: If graph is connected,

1 is an eigenvector with 0 as eigenvalue

$$\mathbf{L1} = \begin{bmatrix} d_1 - \sum_j w_{1j} \\ d_2 - \sum_j w_{2j} \\ \cdots \\ d_n - \sum_j w_{nj} \end{bmatrix} = 0$$

unsupervised embedding

Step 2 – Projection using Graph

- Justification – points connected on the graph stay as close as possible after embedding

$$\min_{\mathbf{f}} \sum_{ij} W_{ij}(\mathbf{f}_i - \mathbf{f}_j)^2 \equiv \min_{\mathbf{f}} \mathbf{f}^T \mathbf{L} \mathbf{f}$$

RHS = $\mathbf{f}^T(D-W)\,\mathbf{f} = \mathbf{f}^T D\,\mathbf{f} - \mathbf{f}^T W\,\mathbf{f} = \sum_i d_i f_i^2 - \sum_{i,j} f_i f_j w_{ij}$

$$= \frac{1}{2}\left(\sum_i (\sum_j w_{ij}) f_i^2 - 2\sum_{ij} f_i f_j w_{ij} + \sum_j (\sum_i w_{ij}) f_j^2\right)$$

$$= \frac{1}{2}\sum_{ij} w_{ij}(f_i - f_j)^2$$ = LHS

unsupervised embedding

Step 2 – Projection using Graph

- Justification – points connected on the graph stay as close as possible after embedding

$$\min_{\mathbf{f}} \sum_{ij} W_{ij}(\mathbf{f}_i - \mathbf{f}_j)^2 \equiv \min_{\mathbf{f}} \mathbf{f}^T \mathbf{L} \mathbf{f} \qquad s.t.\ \mathbf{f}^T \mathbf{f} = 1$$

Similar to PCA with XX^T replaced by L

Lagrangian: $\min_{\mathbf{f}} \mathbf{f}^T \mathbf{L} \mathbf{f} - \lambda \mathbf{f}^T \mathbf{f}$

Wrap constraint into the objective function

$\partial / \partial \mathbf{f} = 0$ $\qquad (\mathbf{L} - \lambda \mathbf{I})\mathbf{f} = 0$ $\qquad \boxed{\mathbf{L}\mathbf{f} = \lambda \mathbf{f}}$

unsupervised embedding

Step 2 – Projection using Graph Laplacian

- Graph Laplacian (unnormalized version)

  $L = D - W$ (n x n matrix)

  Find eigenvectors of the graph Laplacian $\mathbf{Lf} = \lambda\mathbf{f}$

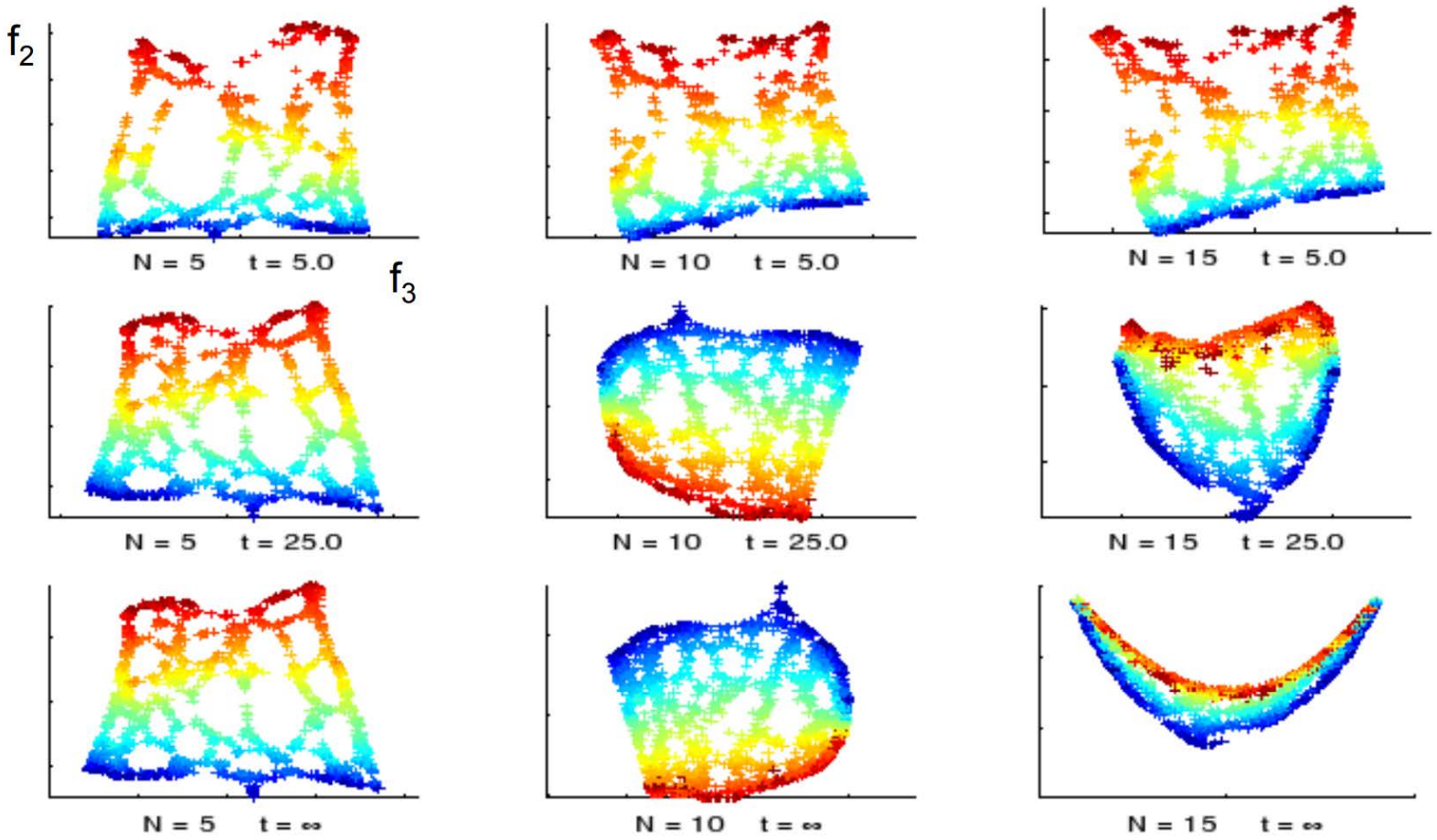
  Ordered eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_n$

  To embed data points in d-dim space, project data points onto eigenvectors associated with $\lambda_1, \lambda_2, \ldots, \lambda_d$

Original Representation		Transformed representation
data point		projections
x_i	→	$(f_1(i), \ldots, f_d(i))$
(D-dimensional vector)		(d-dimensional vector)

unsupervised embedding

Unrolling the swiss roll



N=number of nearest neighbors, t = the heat kernel parameter (Belkin & Niyogi'03)

unsupervised embedding

Dimensionality Reduction Methods

- Feature Selection - Only a few features are relevant to the learning task

  Score features (mutual information, prediction accuracy, domain knowledge)
  Regularization

- Latent features – Some linear/nonlinear combination of features provides a more efficient representation than observed feature

  Linear: Low-dimensional linear subspace projection

  PCA (Principal Component Analysis),
  MDS (Multi Dimensional Scaling),

  Factor Analysis, ICA (Independent Component Analysis)

  Nonlinear: Low-dimensional nonlinear projection that preserves local information along the manifold

  Laplacian Eigenmaps

  ISOMAP, Kernel PCA, LLE (Local Linear Embedding),

  Many, many more …

unsupervised embedding

One the Role and Impact of the Metaparameters

in t-distributed Stochastic Neighbor Embedding

John A. Lee and Michel Verleysen
Machine Learning Group
Université catholique de Louvain
Louvain-la-Neuve, Belgium
michel.verleysen@uclouvain.be

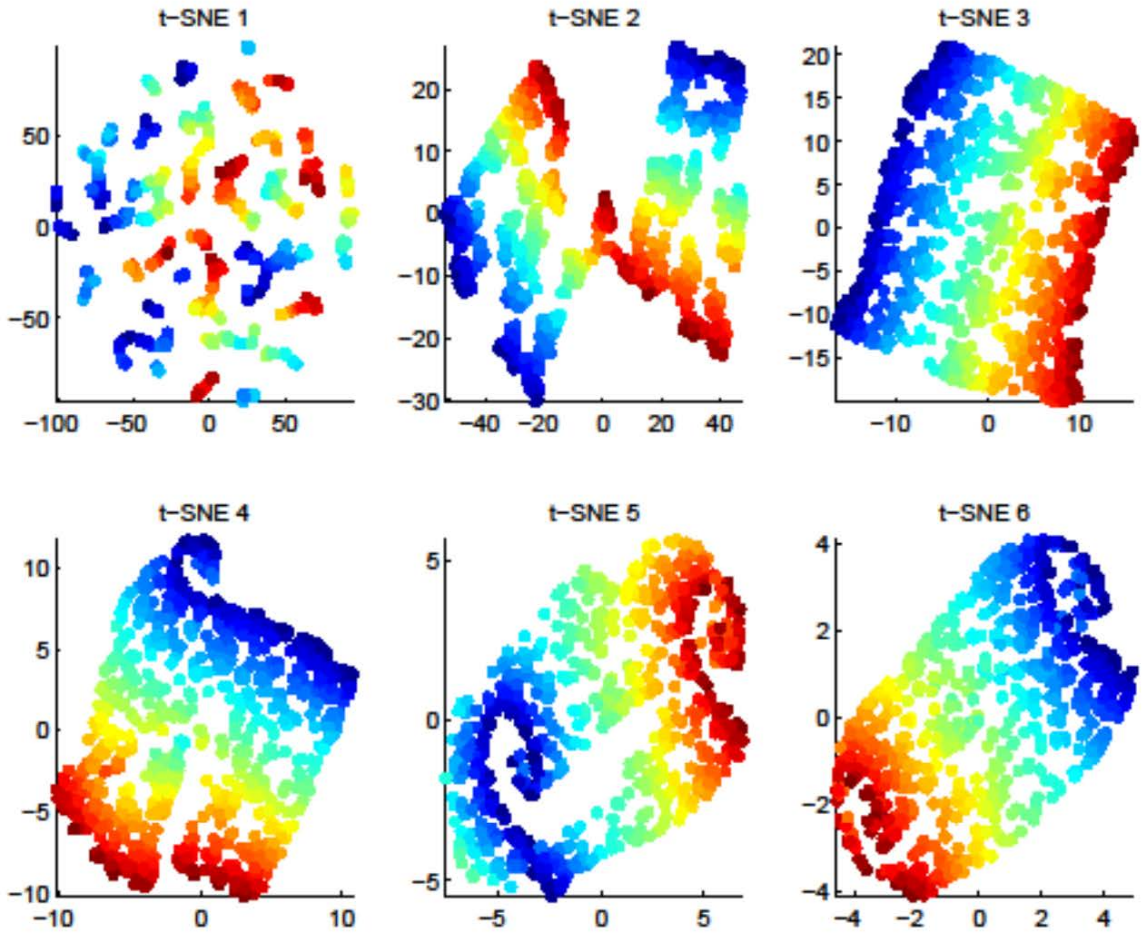
t-SNE Method
for High-Dimensional Data Visualization

David Khosid

2017

unsupervised embedding

Results with Euclidean distances



On the role and impact of the metaparameters in t-distributed SNE

unsupervised embedding

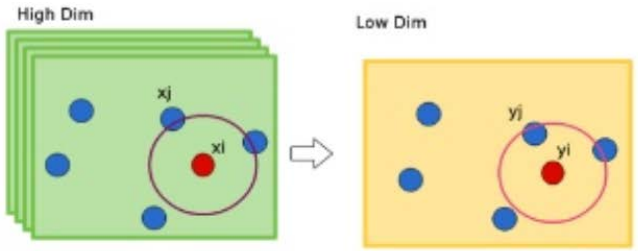
SNE and t-SNE

- In the original space, the similarity between y_i and y_j is defined as

$$p_{j|i}(\lambda_i) = \begin{cases} 0 & \text{if } i = j \\ \dfrac{g(\delta_{ij}/\lambda_i)}{\sum_{k \neq i} g(\delta_{ik}/\lambda_i)} & \text{otherwise} \end{cases} \qquad \left(g(u) = \exp\left(\frac{-u^2}{2} \right) \right)$$

- Similarities are not symmetric (individual widths) !
- $p_{j|i}$ is the empirical probability of y_j to be a neighbor of y_i

unsupervised embedding



Similarity of datapoints ($\mathbf{x}_i$) in data space R^D

$$p_{j|i} = \frac{exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2})}{\sum_{k \neq m} exp(-\frac{\|\mathbf{x}_k - \mathbf{x}_m\|^2}{2\sigma_i^2})}$$

$p_{j|i}$ measures how close $\mathbf{x}_j$ is from $\mathbf{x}_i$, considering Gaussian distribution around $\mathbf{x}_i$ with a given variance σ_i^2.

Similarity of datapoints ($\mathbf{x}_i$) in data space R^D

$$p_{j|i} = \frac{exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2})}{\sum_{k \neq m} exp(-\frac{\|\mathbf{x}_k - \mathbf{x}_m\|^2}{2\sigma_i^2})} \quad (1)$$

Make the similarity metric p_{ij} symmetric. The main advantage of symmetry is simplifying the gradient (learning stage):

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \quad (2)$$

- we set $p_{ii} = 0$, as we interested in pairwise similarities
- σ_i is chosen such that the data point has a fixed **perplexity** (effective number of neighbors).

unsupervised embedding

SNE and t-SNE

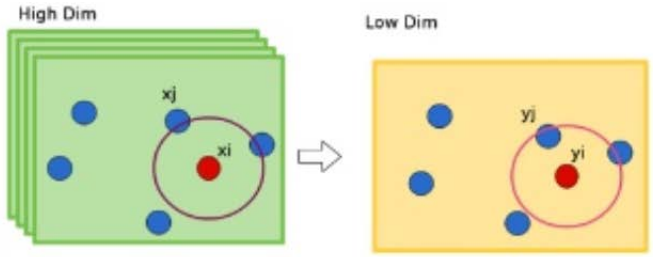
- In the embedding space, the similarity between x_i and x_j is defined as

$$q_{ij}(n) = \begin{cases} 0 & \text{if } i = j \\ \dfrac{t(d_{ij}, n)}{\sum_{k \neq l} t(d_{kl}, n)} & \text{otherwise} \end{cases}$$

$$t(u,n) = \left(1 + \frac{u^2}{n}\right)^{-\frac{n+1}{2}}$$

- Similarities are symmetric
- $t(u,n)$ is proportional to a Student t with n degrees of freedom (n controls the thickness of the tail)
- SNE: $n \to \infty$ *t-SNE:* $n = 1$

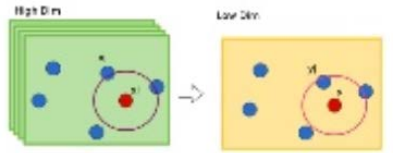
unsupervised embedding



Student t-distribution with 1DoF (same as Cauchy distribution)

$$q_{ij} = \frac{\frac{1}{1+\|\mathbf{y}_i - \mathbf{y}_j\|^2}}{\sum_{k \neq m} \frac{1}{1+\|\mathbf{y}_k - \mathbf{y}_m\|^2}} \tag{3}$$

- we set $q_{ii} = 0$, as we interested in pairwise similarities
- heavy-tail (will be discussed later)
- still closely related to the Gaussian
- computationally convenient (no exponent)



$\mathbf{P}=[p_{ij}]$ is fixed, $\mathbf{Q}=[q_{ij}]$ is flexible.
We want $\mathbf{P}$ and $\mathbf{Q}$ to be as close as possible.

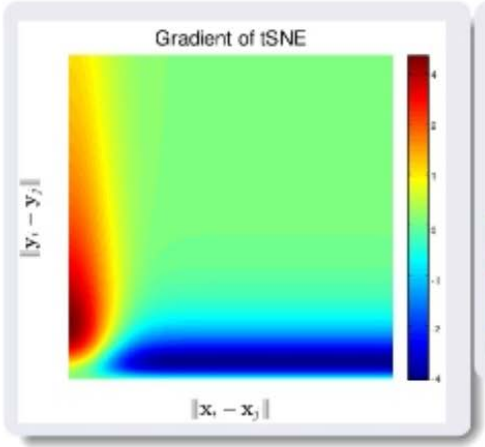
$$C = KL(\mathbf{P}\|\mathbf{Q}) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

KL divergence:

- is not a distance, since it is asymmetric
- large p_{ij} modelled by small $q_{ij} \rightarrow$ large penalty
- Small p_{ij} modelled by large $q_{ij} \rightarrow$ small penalty
- **KL divergence** meaning: cross-entropy

unsupervised embedding

t-SNE algorithm minimizes KL divergence between **P** and **Q** distributions.



$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{i \neq j} (p_{ij} - q_{ij}) \frac{\mathbf{y}_i - \mathbf{y}_j}{1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2} \qquad (5)$$

positive → attraction

negative → repulsion

(dissimilar DPs, similar MPs) → repulsion

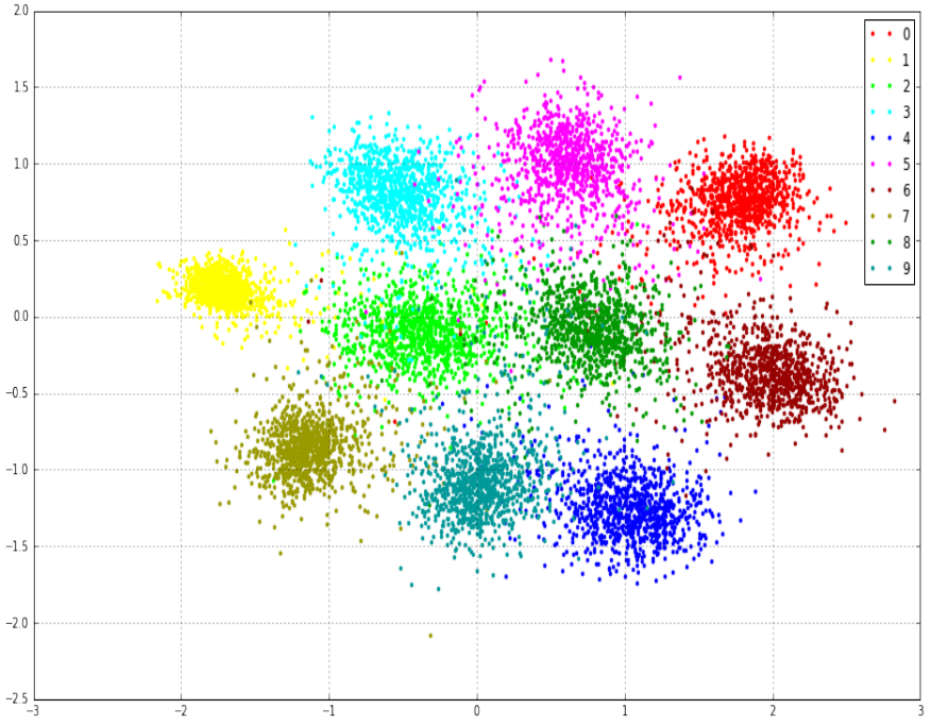
$$\mathbf{Y}^{(t)} = \mathbf{Y}^{(t-1)} + \eta \frac{\partial C}{\partial \mathbf{Y}} + \alpha(t)(\mathbf{Y}^{(t-1)} - \mathbf{Y}^{(t-2)}) \qquad (6)$$

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{i \neq j} (p_{ij} - q_{ij}) \frac{\mathbf{y}_i - \mathbf{y}_j}{1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2}$$

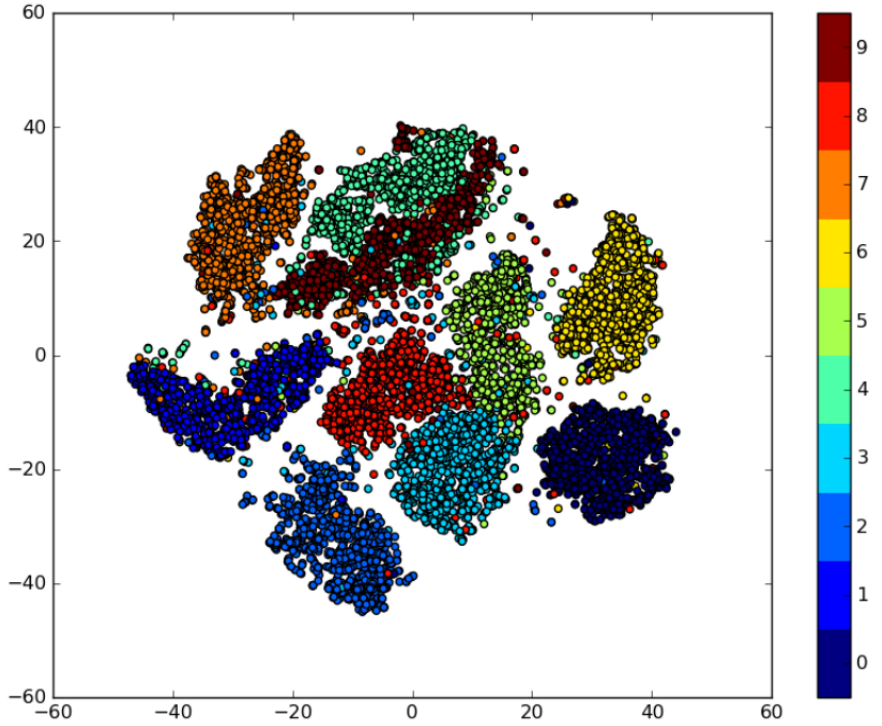
Spring Analogy: $F = -k * (\mathbf{y}_i - \mathbf{y}_j)$, attraction/repulsion

Supervised vs. unsupervised embedding

Toy Example of MNIST Dataset

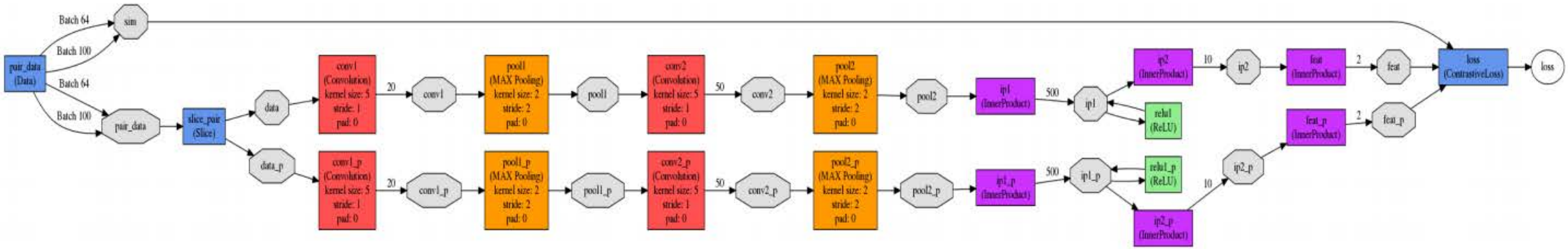


Siamese Embedding



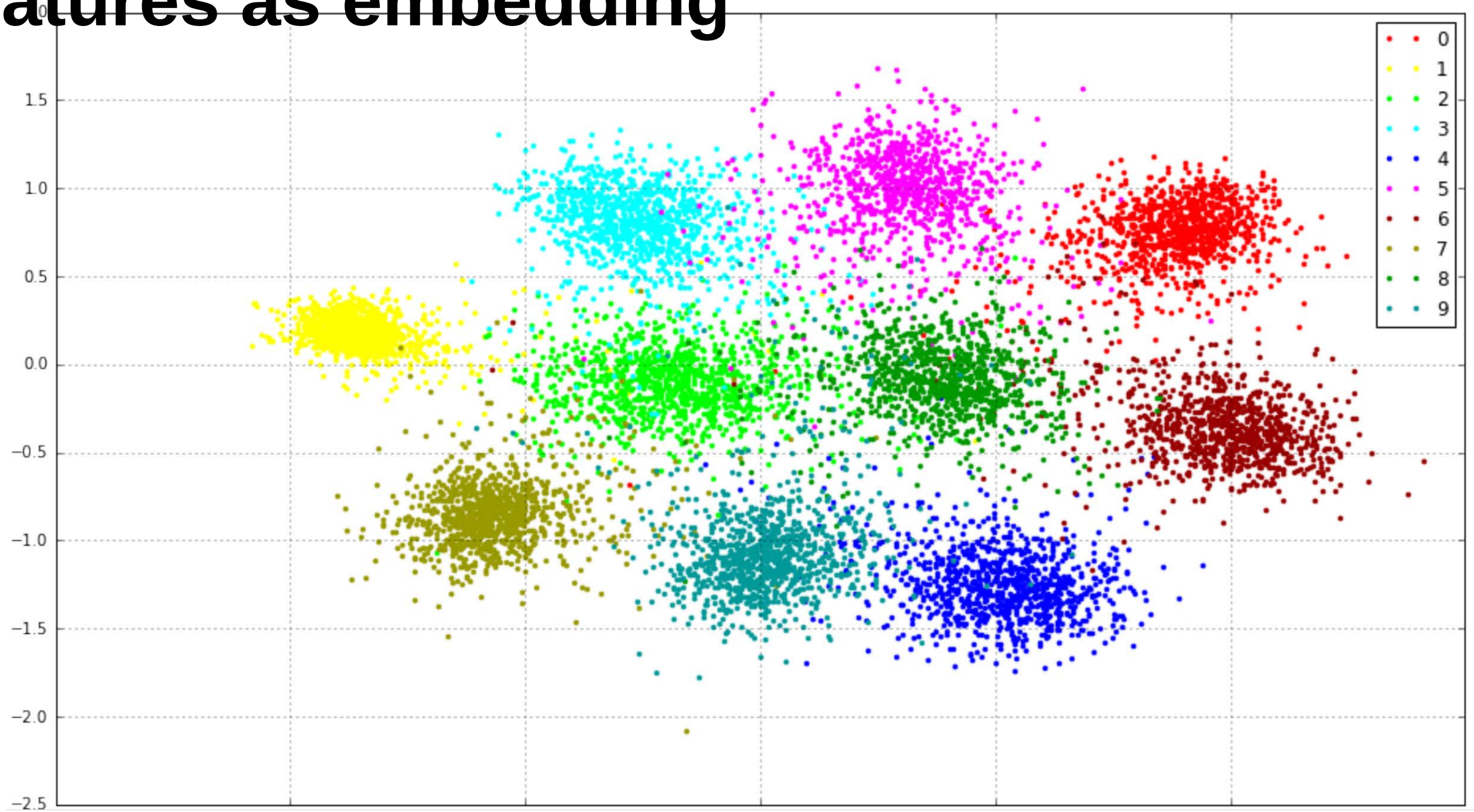
t-SNE Embedding

Learning Features for discriminative embedding



$$d_g(\boldsymbol{a}, \boldsymbol{b}, c) = \mathbb{1}(c = g)d(\boldsymbol{a}, \boldsymbol{b}) + \mathbb{1}(c \neq g)\max(\delta - d(\boldsymbol{a}, \boldsymbol{b}), 0)$$

Features as embedding



Autoencoders

- Generated images need to be sharp
  - Generated images from auto-encoder tend to be blurry.

Autoencoders

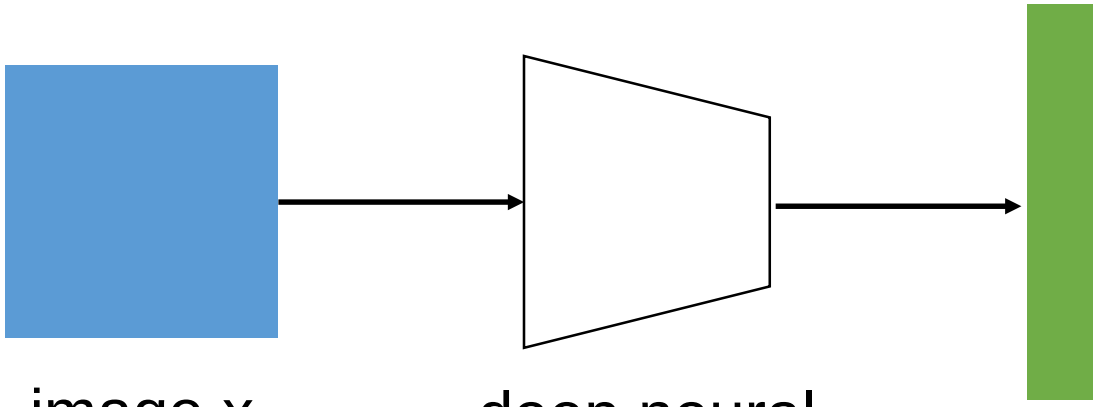


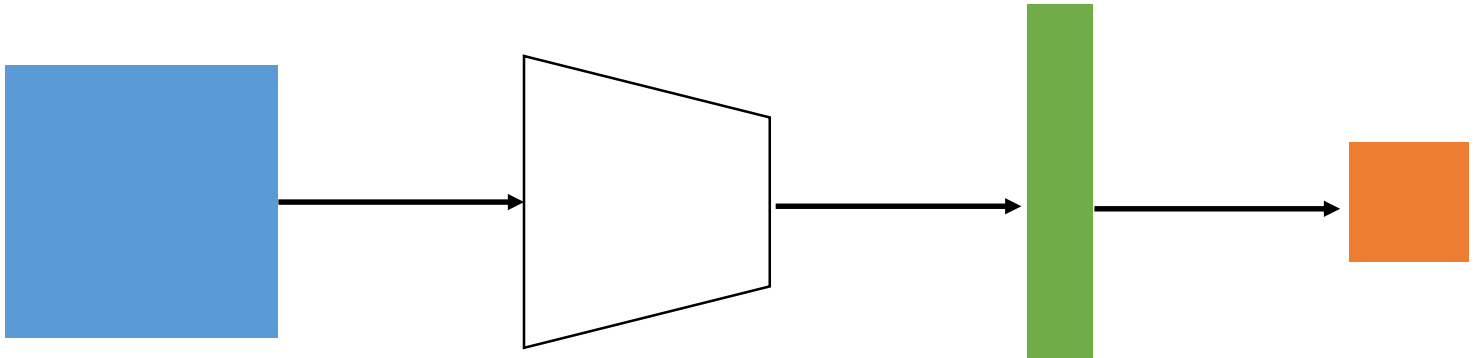
image x

deep neural network

deep feature ϕ
Representation of image x in a condensed version

Autoencoders

Supervised scenario: label y to train the network to get the feature



deep feature ϕ
Representation of image x
Less spatial resolution, more feature descriptions

Autoencoders

Unsupervised scenario: if we don't have any label, how can we train it?

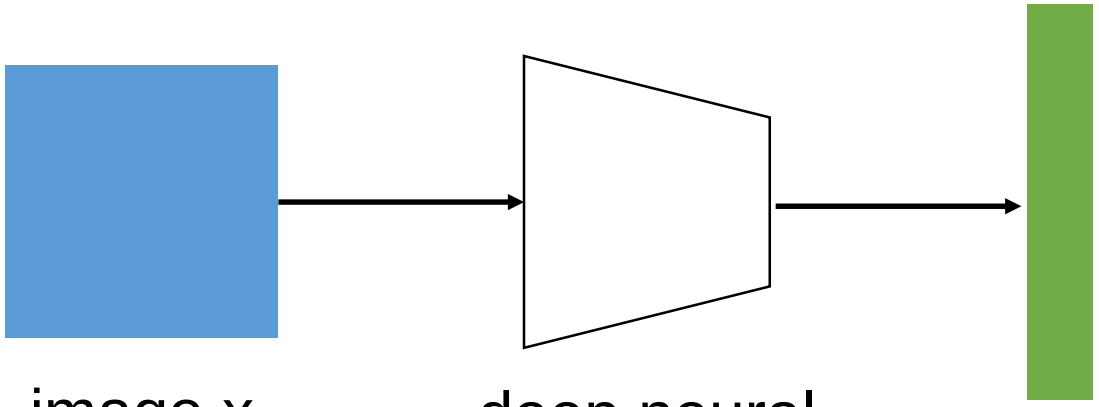


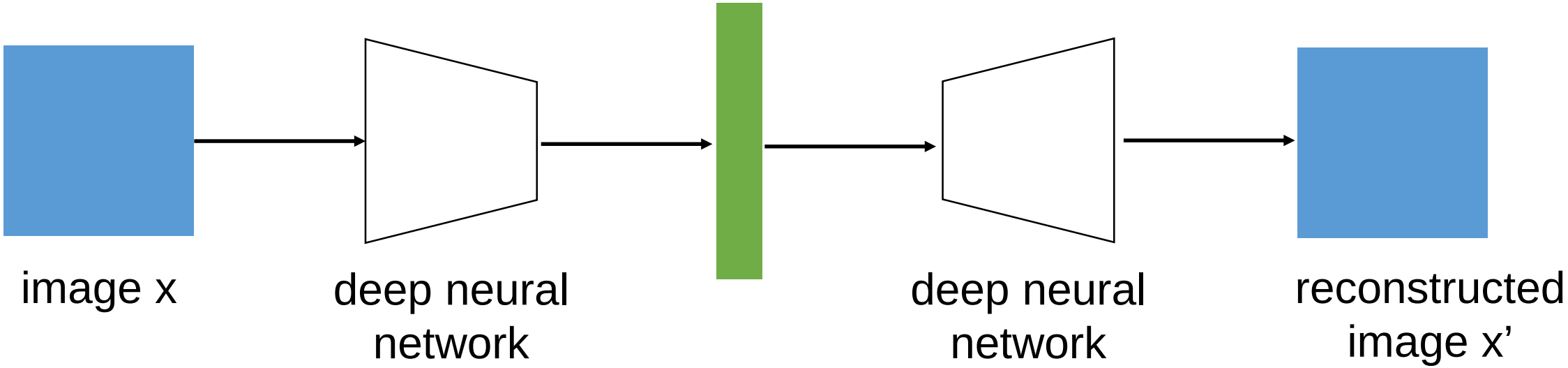
image x

deep neural network

deep feature ϕ
Representation of image x
Less spatial resolution,
more feature descriptions

Autoencoders

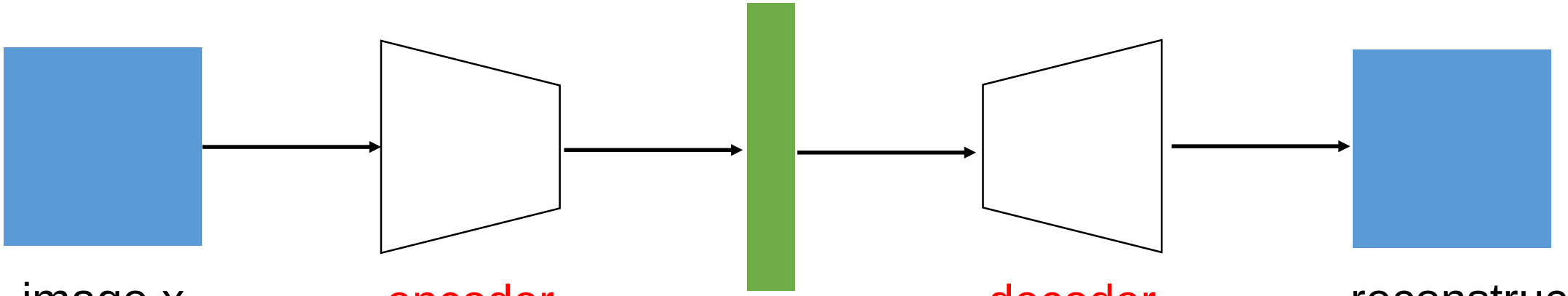
Autoencoders: ||x-x'|| as loss, use back propagation to train the network



deep feature ϕ
Representation of image x
Space vs features: Less spatial resolution, more feature descriptions

Autoencoders

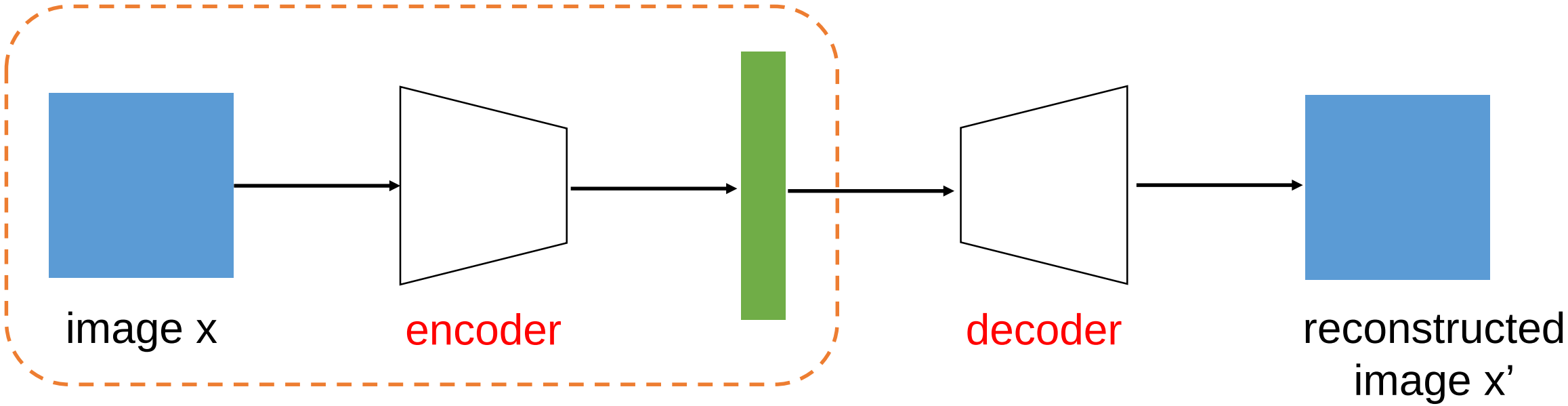
Autoencoders: ||x-x'|| as loss, use back propagation to train the network



deep feature ϕ
Representation of image x
Less spatial resolution, more feature descriptions

Autoencoders

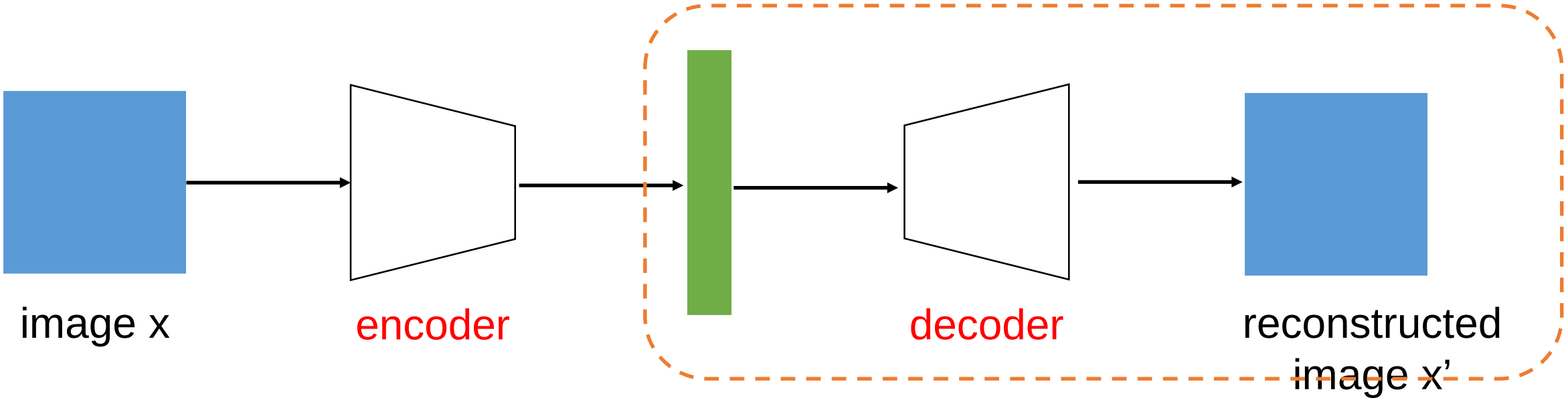
The encoder maps image space to feature space (self-learned)



deep feature ϕ
Representation of image x
Less spatial resolution, more
feature descriptions

Autoencoders

The decoder maps feature space to image space



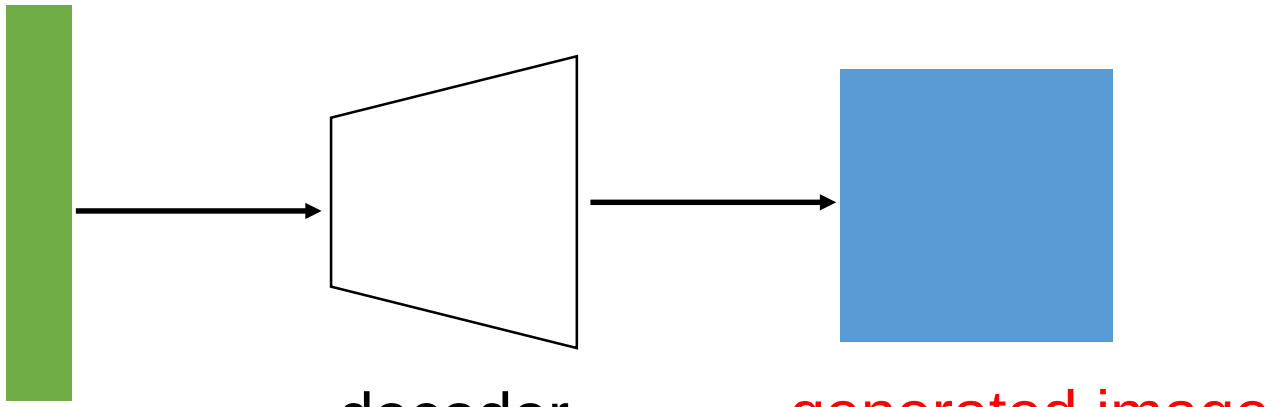
deep feature ϕ
Representation of image x
Less spatial resolution, more feature descriptions

Autoencoders

Can this process be generalized?

Goal of generative model: Generate realistic image from a random vector.

Problem: How to train it?

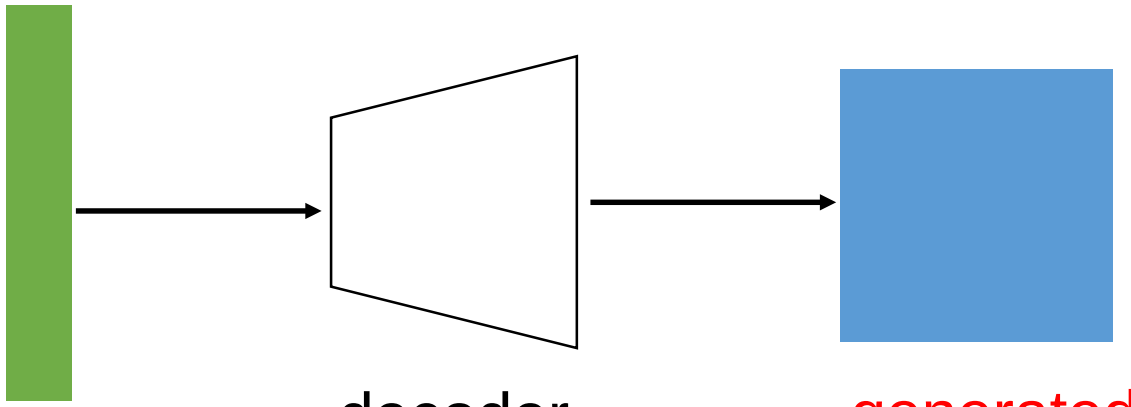


Autoencoders

Can this process be generalized?

Goal of generative model: Generate realistic image from a random vector.

Problem: How to train it?



decoder

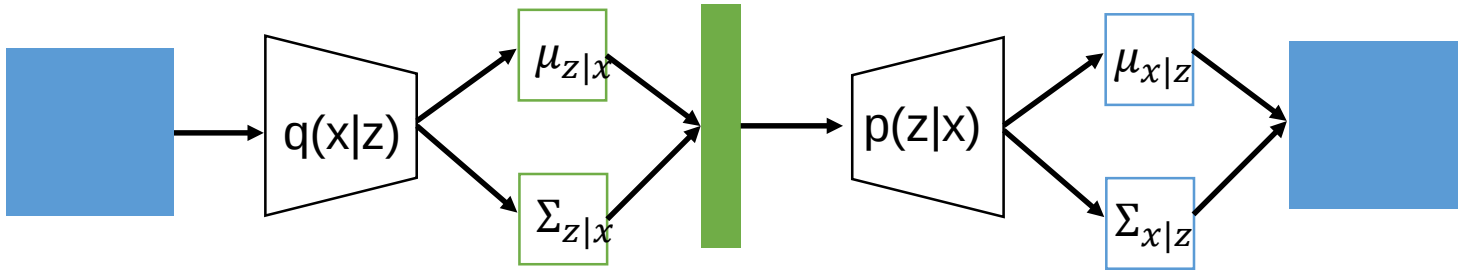
generated image x

random vector z from a predefined distribution (e.g. Gaussian)

Latent variable: attributes of the image

Models

1. Variational Autoencoders

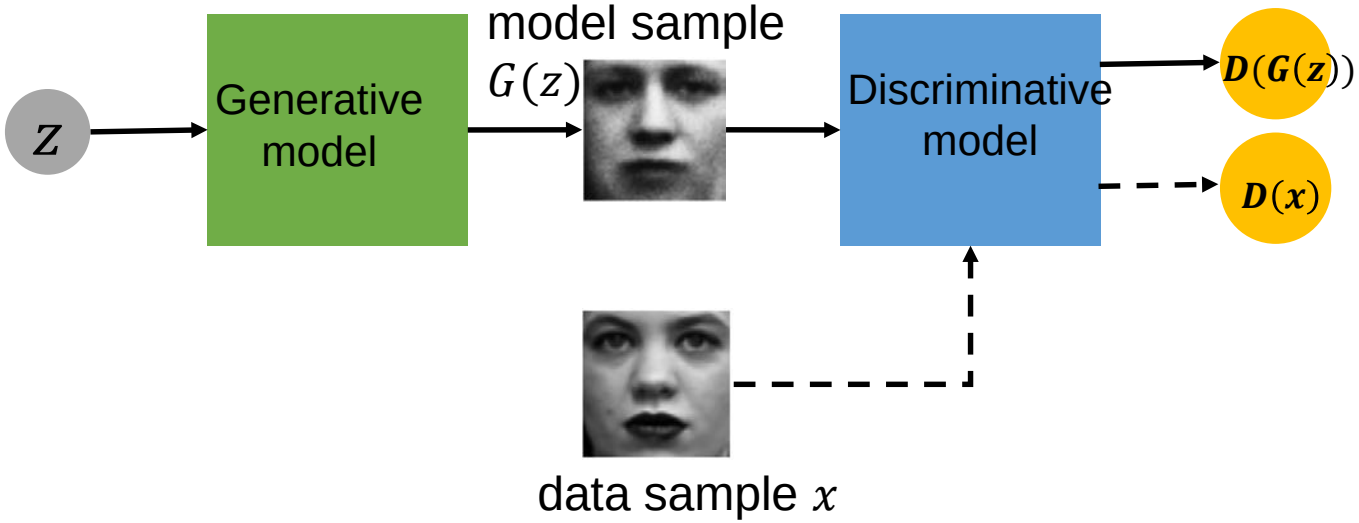


x

sample z from
z|x ~ N($\mu_{z|x}$, $\Sigma_{z|x}$)

sample x from
x|z ~ N($\mu_{x|z}$, $\Sigma_{x|z}$)

2. Generative Adversarial Network

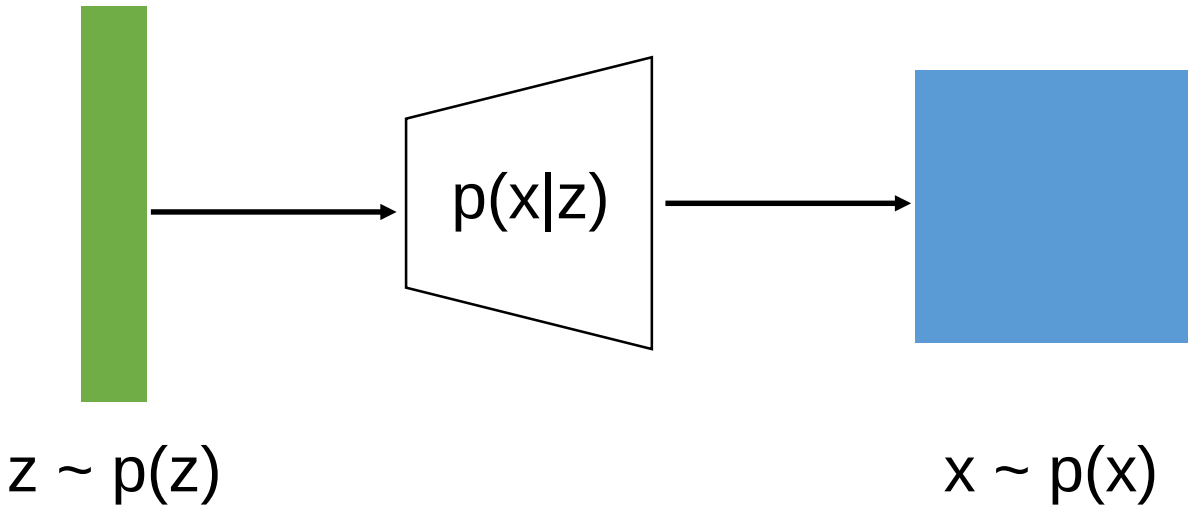


Variational Autoencoders

if we can train a model to approximate p(x|z), then we can generate the image by:

(1) Draw a latent variable z ~ p(z)

(2) Draw a data point x ~ p(x|z)

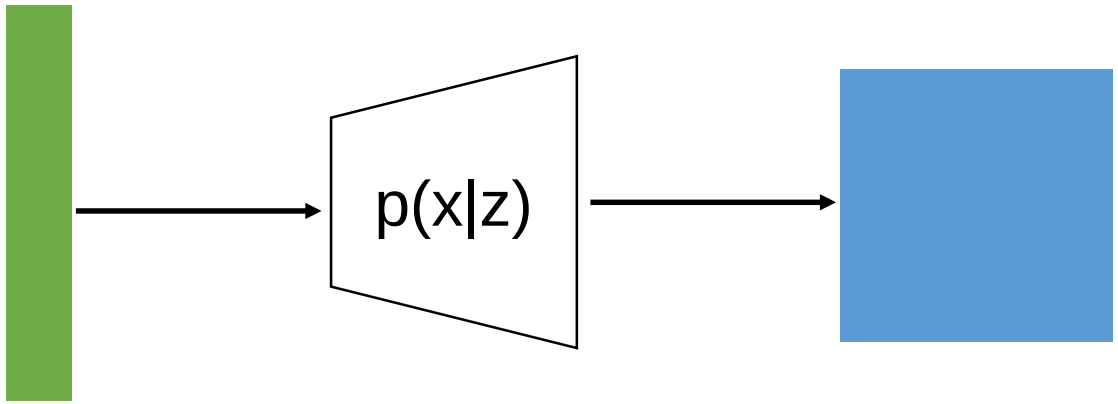


Variational Autoencoders

if we can train a model to approximate p(x|z), then we can generate the image by:

(1) Draw a latent variable z ~ p(z)

(2) Draw a data point x ~ p(x|z)



z ~ p(z)
Simple distribution
(e.g. Multivariate Gaussian)

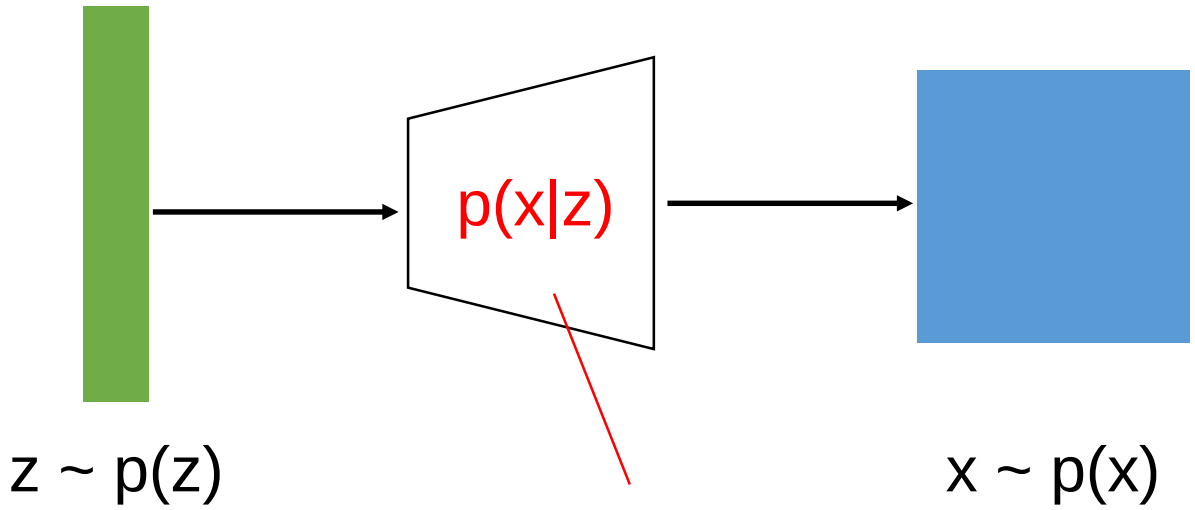
x ~ p(x)

Variational Autoencoders

if we can train a model to approximate p(x|z), then we can generate the image by:

(1) Draw a latent variable z ~ p(z)

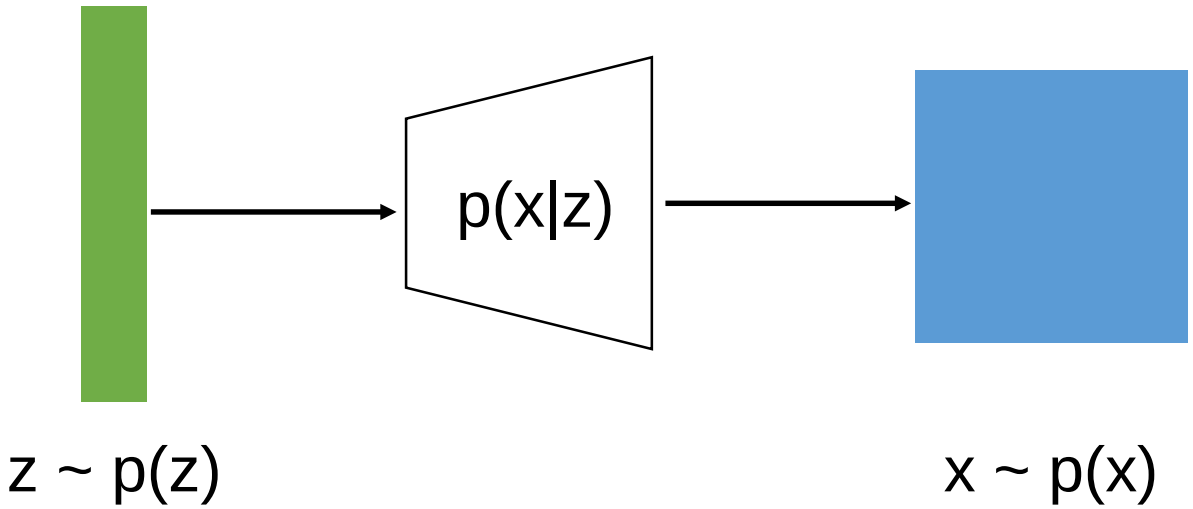
(2) Draw a data point x ~ p(x|z)



Variational Autoencoders

Training strategy:

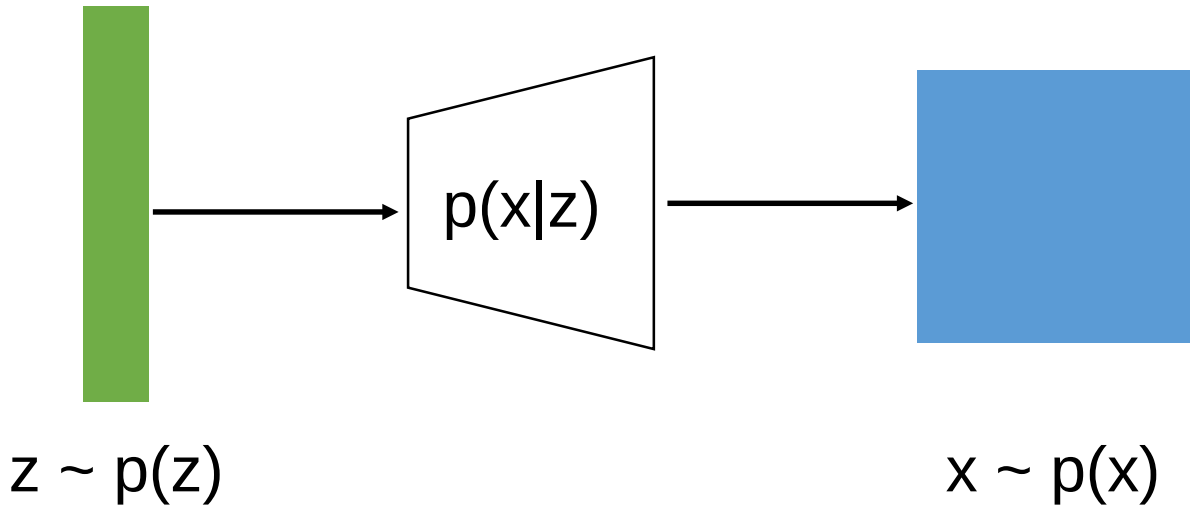
Given a set of training data (images) $x_1, \ldots, x_n$, train the network to maximize the likelihood of training data p(x)



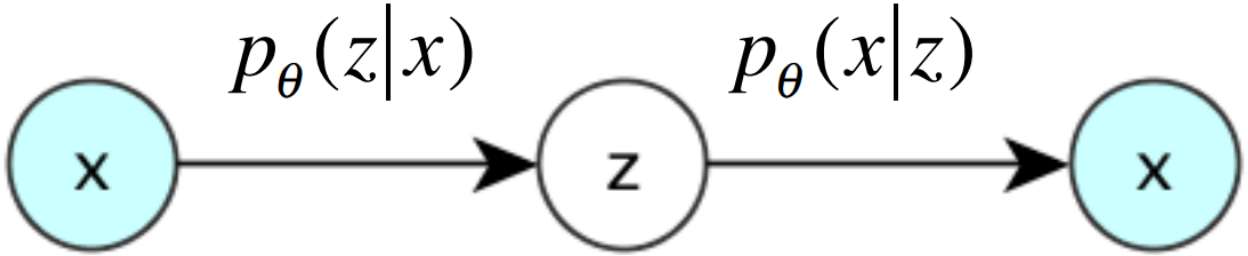
Variational Autoencoders

$$p(x) = \int p(z)p(x|z)dz$$

z is continuous, intractable to compute p(x|z) for every z



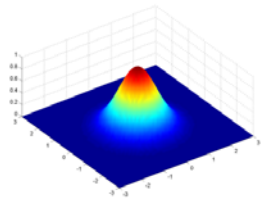
Training as an autoencoder

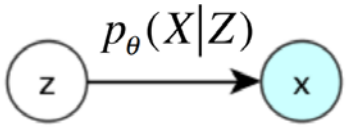


Training use maximum likelihood of p(x) given the training data

$z \sim p(z)$ multivariate Gaussian

$x|z \sim p_\theta(x|z)$





One Example

Problem:

$$p_\theta(z|x)$$

Cannot be calculated:

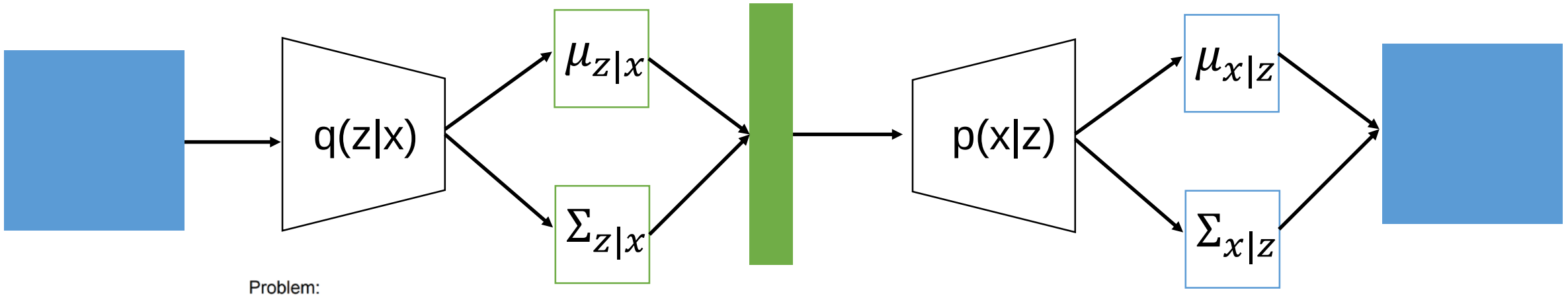
Solution:

- MCMC (too costly)
- Approximate p(z|x) with q(z|x)

Variational Autoencoders

1. Introduce an encoder network to model q(z|x)≈p(z|x)
2. Introduce Gaussian distribution to represent the output of the encoder and decoder

We will show later that, this gives tractable lower bound of p(x)



x

Problem:

$p_\theta(z|x)$

Cannot be calculated:

Solution:
- MCMC (too costly)
- Approximate p(z|x) with q(z|x)

sample z from
z|x ~ N($\mu_{z|x}$, $\Sigma_{z|x}$)

sample x from
x|z ~ N($\mu_{x|z}$, $\Sigma_{x|z}$)

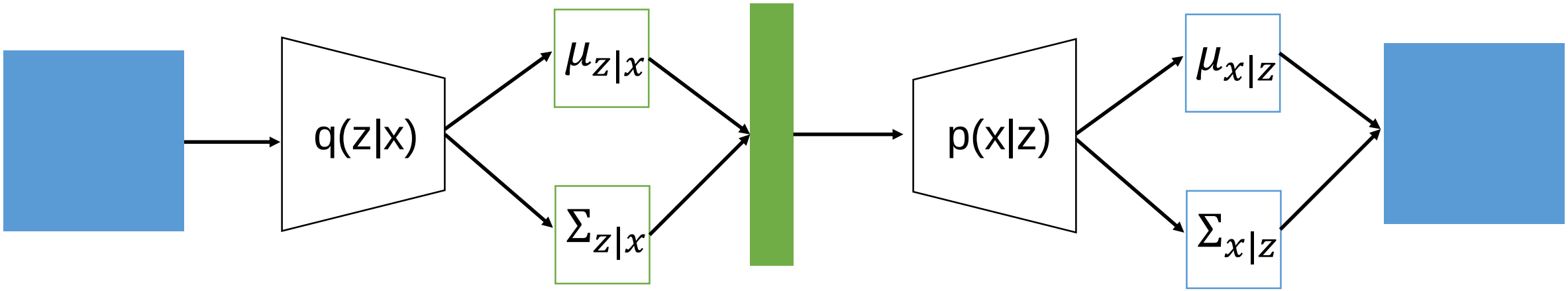
Variational Autoencoders

For each image x from training example :

$$L = \log\big(p(x)\big)$$

$$= \sum_{z} q(z|x) \log(p(x))$$

Multiply with 1



x

sample z from
z|x ~ N($\mu_{z|x}$, $\Sigma_{z|x}$)

sample x from
x|z ~ N($\mu_{x|z}$, $\Sigma_{x|z}$)

Variational Autoencoders

For each image x from training example :

$$
\begin{aligned}
L &= \log\big(p(x)\big) && \\
&= \textstyle\sum_z q(z|x) \log\big(p(x)\big) && \text{Multiply with 1} \\
&= \textstyle\sum_z q(z|x) \log\left(\frac{p(z,x)}{p(z|x)}\right) && \text{Bayes role} \\
&= \textstyle\sum_z q(z|x) \log\left(\frac{p(z,x)}{p(z|x)}\frac{q(z|x)}{q(z|x)}\right) && \text{Multiply with 1} \\
&= \textstyle\sum_z q(z|x) \log\left(\frac{p(z,x)}{q(z|x)}\right) + \sum_z q(z|x) \log\left(\frac{q(z|x)}{p(z|x)}\right) && \text{Logarithms} \\
&= L^v + D_{KL}(q(z|x)||p(z|x)) &&
\end{aligned}
$$

Variational Autoencoders

For each image x from training example :

$$
\begin{aligned}
L &= \log\big(p(x)\big) \\
&= \textstyle\sum_z q(z|x) \log\big(p(x)\big) \\
&= \textstyle\sum_z q(z|x) \log\left(\frac{p(z,x)}{p(z|x)}\right) \\
&= \textstyle\sum_z q(z|x) \log\left(\frac{p(z,x)}{p(z|x)}\frac{q(z|x)}{q(z|x)}\right) \\
&= \textstyle\sum_z q(z|x) \log\left(\frac{p(z,x)}{q(z|x)}\right) + \sum_z q(z|x) \log\left(\frac{q(z|x)}{p(z|x)}\right) \\
&= L^v + D_{KL}(q(z|x)||p(z|x))
\end{aligned}
$$

Multiply with 1

Bayes role

Multiply with 1

Logarithms

Variational lower bound of L

KL divergence >= 0

Variational Autoencoders

For each image x from training example :

$$L^v = \sum_z q(z|x) \log\left(\frac{p(z,x)}{q(z|x)}\right)$$

$$= \sum_z q(z|x) \log\left(\frac{p(x|z)p(z)}{q(z|x)}\right)$$

$$= \sum_z q(z|x) \log\left(\frac{p(z)}{q(z|x)}\right) + \sum_z q(z|x) \log\big(p(x|z)\big)$$

$$= -D_{KL}(q(z|x)||p(z)) + \mathbb{E}_{q(z|x)}(\log(p(x|z)))$$

Bayes role

Logarithms

KL divergence between Gaussian for encoder and prior Gaussian z
Close form solution
Also known as regularization term

Variational Autoencoders

For each image x from training example :

$$L^v = \sum_z q(z|x) \log\left(\frac{p(z,x)}{q(z|x)}\right)$$
$$= \sum_z q(z|x) \log\left(\frac{p(x|z)p(z)}{q(z|x)}\right)$$
$$= \sum_z q(z|x) \log\left(\frac{p(z)}{q(z|x)}\right) + \sum_z q(z|x) \log\big(p(x|z)\big)$$
$$= -D_{KL}(q(z|x)||p(z)) + \mathbb{E}_{q(z|x)}(\log(p(x|z)))$$

Bayes role

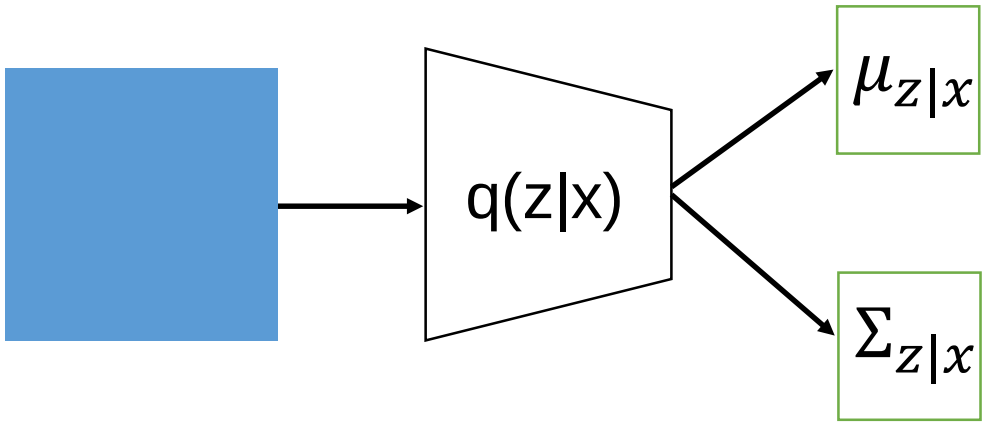
Logarithms

Reconstruction quality from the decoder
Can be estimated by sampling

Variational Autoencoders

$$L^v = -D_{KL}(q(z|x)||p(z)) + \mathbb{E}_{q(z|x)}(\log(p(x|z)))$$

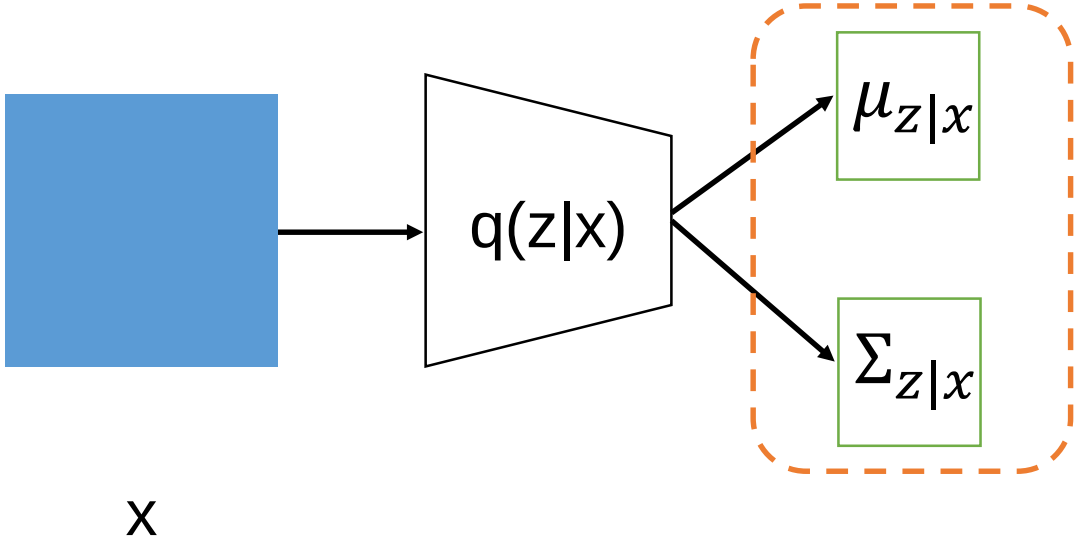
Given a minibatch of training images x, pass through the encoder.



Variational Autoencoders

$$L^v = -D_{KL}(q(z|x)||p(z)) + \mathbb{E}_{q(z|x)}(\log(p(x|z)))$$

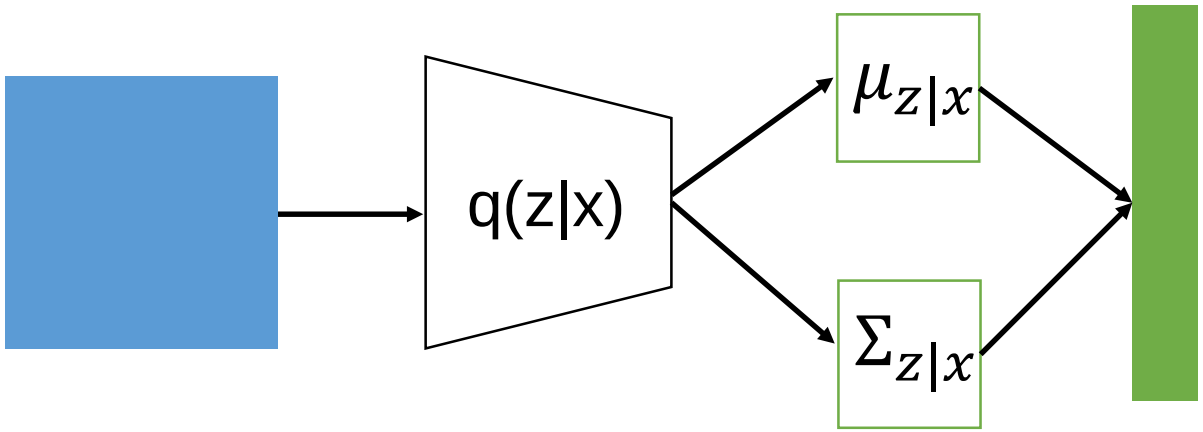
KL divergence between two Gaussians, close form solution.
Avoid the encoder from “cheating” by projecting similar images to far latent vectors in latent space.



Variational Autoencoders

$$L^v = -D_{KL}(q(z|x)||p(z)) + \mathbb{E}_{q(z|x)}(\log(p(x|z)))$$

Sample z from z|x ~ N($\mu_{z|x}$, $\Sigma_{z|x}$)



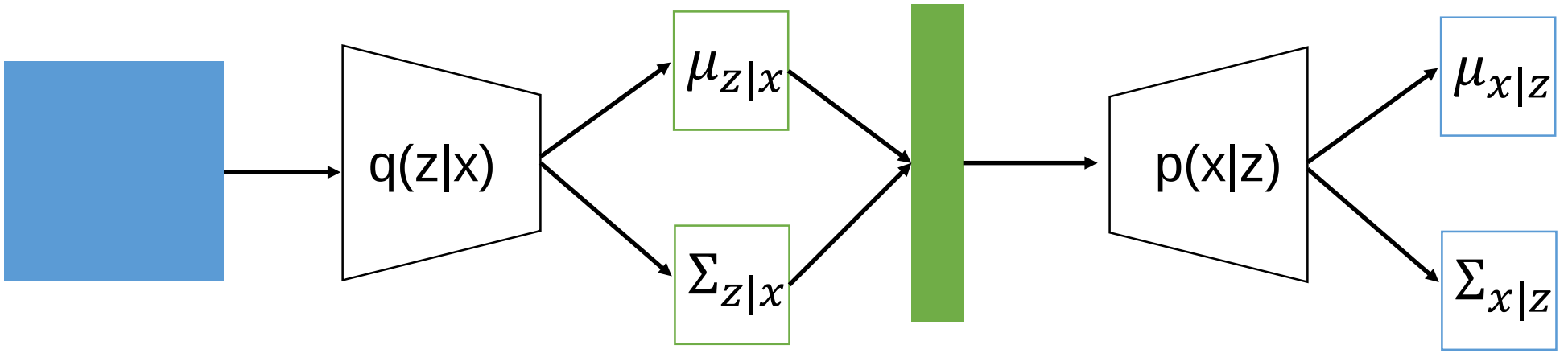
x

sample z from
z|x ~ N($\mu_{z|x}$, $\Sigma_{z|x}$)

Variational Autoencoders

$$L^v = -D_{KL}(q(z|x)||p(z)) + \mathbb{E}_{q(z|x)}(\log(p(x|z)))$$

Pass z through the decoder.



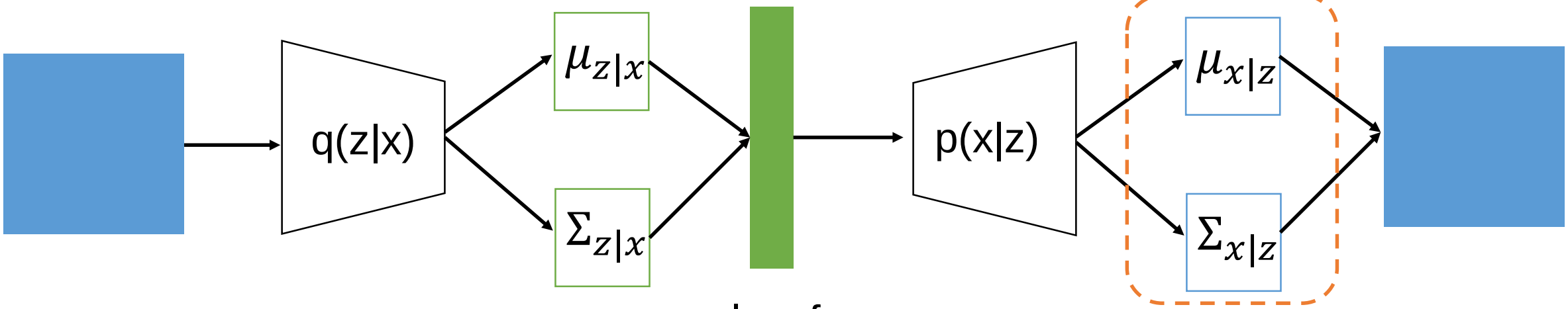
x

sample z from
z|x ~ N($\mu_{z|x}$, $\Sigma_{z|x}$)

Variational Autoencoders

$$L^v = -D_{KL}(q(z|x)||p(z)) + \mathbb{E}_{q(z|x)}(\log(p(x|z)))$$

Approximate by sampling z from q(z|x)
Maximize the likelihood of the original input being reconstructed



x

sample z from
z|x ~ N($\mu_{z|x}$, $\Sigma_{z|x}$)

sample x from
x|z ~ N($\mu_{x|z}$, $\Sigma_{x|z}$)

Sampling to calculate $\mathbb{E}_{q(z|x^{(i)})}\left(\log\left(p(x^{(i)}|z)\right)\right)$

Approximating $\mathbb{E}_{q(z|x^{(i)})}$ with sampling form the distribution $q(z|x^{(i)})$

With $z^{(i,l)}$ $l = 1, 2, \ldots L$ sampled from $z^{(i,l)} \sim q(z|x^{(i)})$

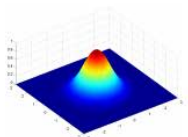
$$L^{v} = -D_{KL}\left(q(z|x^{(i)})||p(z)\right) + \mathbb{E}_{q(z|x^{(i)})}\left(\log\left(p(x^{(i)}|z)\right)\right)$$

$$L^{v} \approx -D_{KL}\left(q(z|x^{(i)})||p(z)\right) + \frac{1}{L}\sum_{i=1}^{L}\log\left(p(x^{(i)}|z^{(i,l)})\right)$$

Example $x^{(i)}$

$q_\phi(z|x^{(i)})$

z



$\log(p_\theta(x^{(i)}|z^{(i,1)}))$ where $z^{(i,1)} \sim N(\mu_z^{(i)}, \sigma_z^{2(i)})$

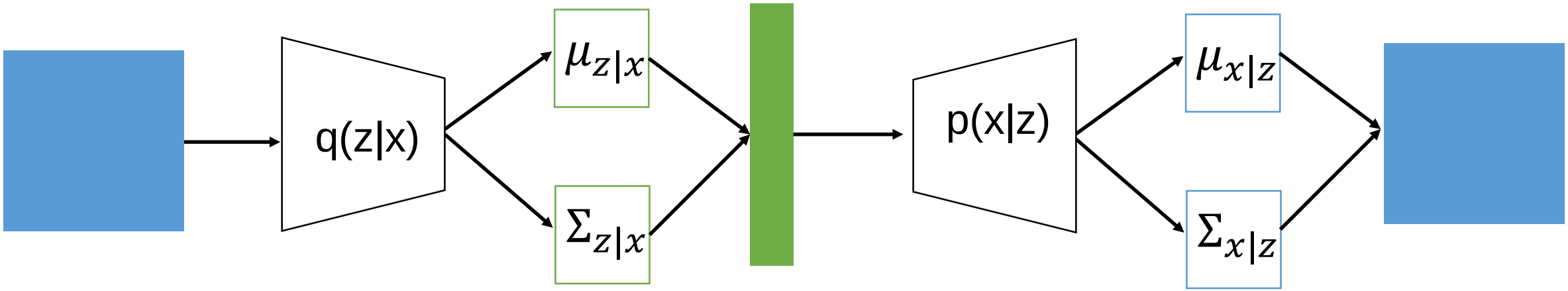
...

$\log(p_\theta(x^{(i)}|z^{(i,L)}))$ where $z^{(i,L)} \sim N(\mu_z^{(i)}, \sigma_z^{2(i)})$

Variational Autoencoders

During training, back propagation to maximize the lower bound of p(x)

$$L^v = -D_{KL}(q(z|x)||p(z)) + \mathbb{E}_{q(Z|X)}(\log(p(x|z)))$$



x

sample z from
z|x ~ N($\mu_{z|x}$, $\Sigma_{z|x}$)

sample x from
x|z ~ N($\mu_{x|z}$, $\Sigma_{x|z}$)

Assume we have a normal distribution q that is parameterized by θ, specifically $q_\theta(x) = N(\theta, 1)$. We want to solve the below problem

$$\min_\theta \quad E_q[x^2]$$

This is of course a rather silly problem and the optimal θ is obvious. However, here we just want to understand how the reparameterization trick helps in calculating the gradient of this objective $E_q[x^2]$.

One way to calculate $\nabla_\theta E_q[x^2]$ is as follows

$$\nabla_\theta E_q[x^2] = \nabla_\theta \int q_\theta(x) x^2 dx = \int x^2 \nabla_\theta q_\theta(x) \frac{q_\theta(x)}{q_\theta(x)} dx = \int q_\theta(x) \nabla_\theta \log q_\theta(x) x^2 dx$$
$$= E_q[x^2 \nabla_\theta \log q_\theta(x)]$$

For our example where $q_\theta(x) = N(\theta, 1)$, this method gives

$$\nabla_\theta E_q[x^2] = E_q[x^2 (x - \theta)]$$

Reparameterization trick is a way to rewrite the expectation so that the distribution with respect to which we take the gradient is independent of parameter θ. To achieve this, we need to make the stochastic element in q independent of θ. Hence, we write x as

$$x = \theta + \epsilon, \quad \epsilon \sim N(0, 1)$$

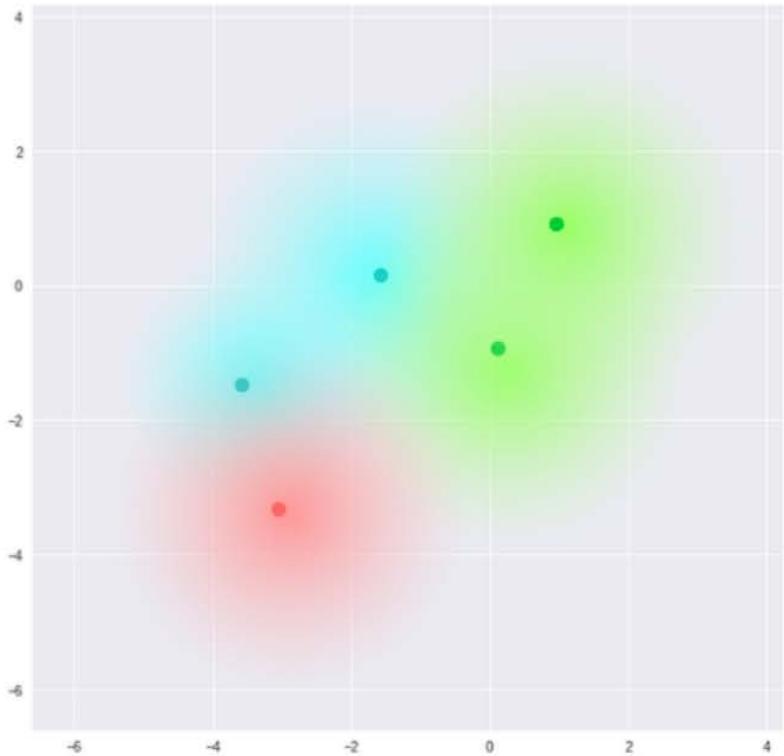
Then, we can write

$$E_q[x^2] = E_p[(\theta + \epsilon)^2]$$

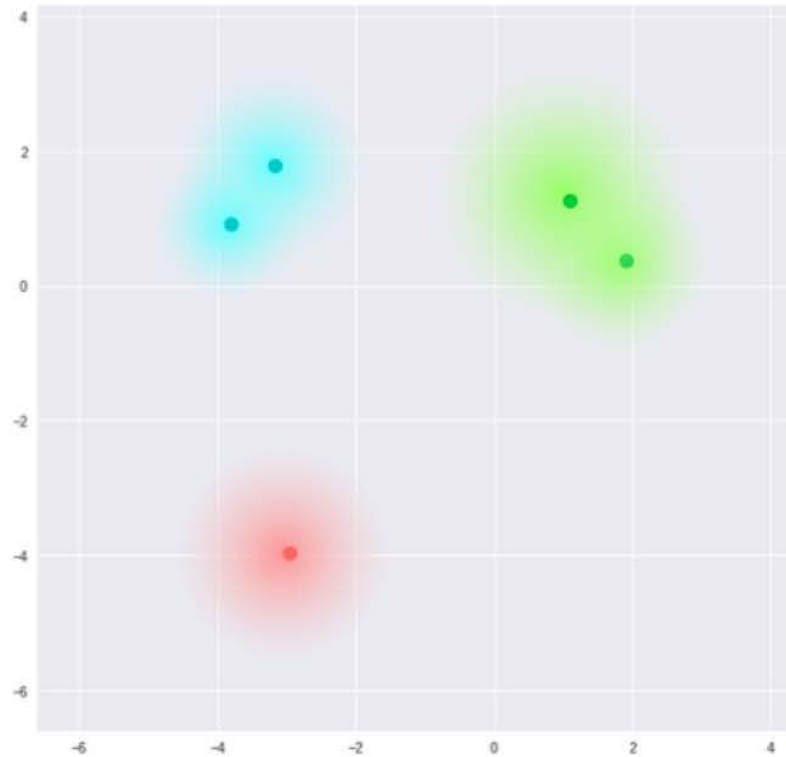
where p is the distribution of ϵ, i.e., $N(0, 1)$. Now we can write the derivative of $E_q[x^2]$ as follows

$$\nabla_\theta E_q[x^2] = \nabla_\theta E_p[(\theta + \epsilon)^2] = E_p[2(\theta + \epsilon)]$$

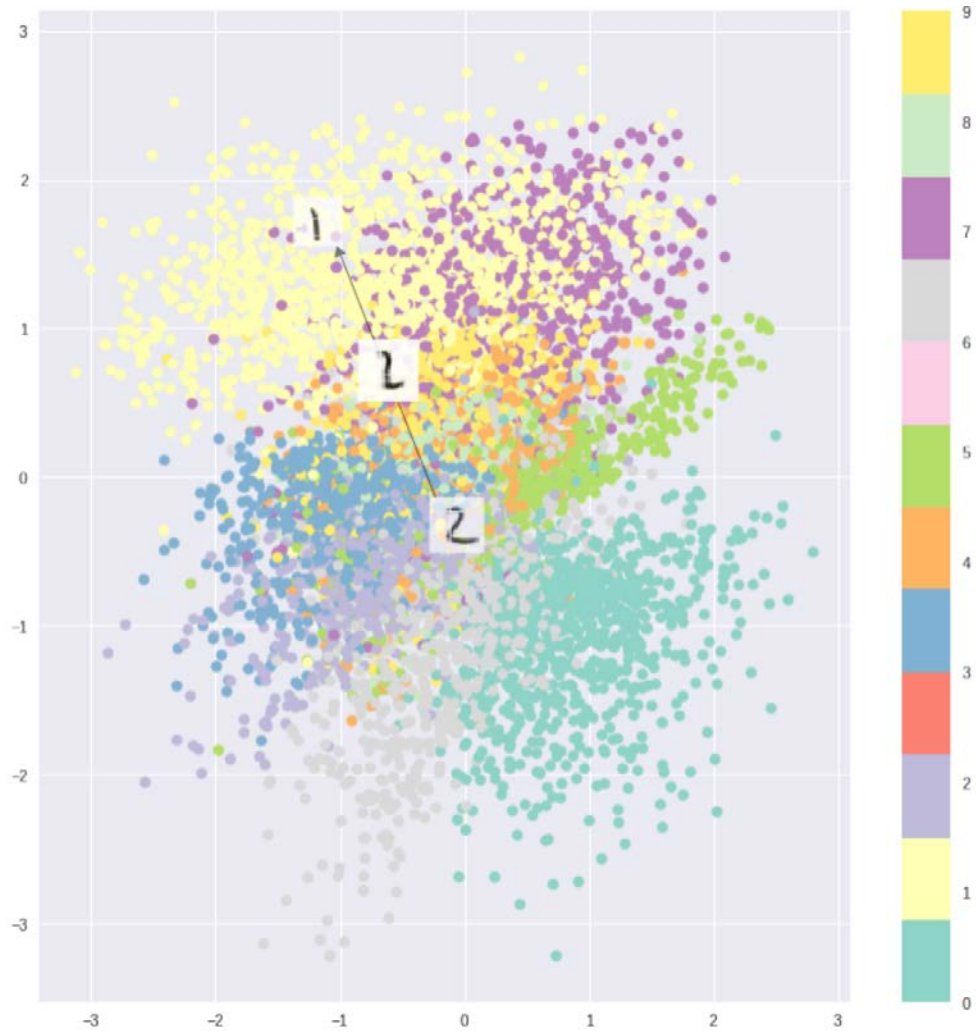
Here is an IPython notebook I have written that looks at the variance of these two ways of calculating gradients.
http://nbviewer.jupyter.org/github/gokererdogan/Notebooks/blob/master/Reparameterization%20Trick.ipynb



What we require



What we may inadvertently end up with

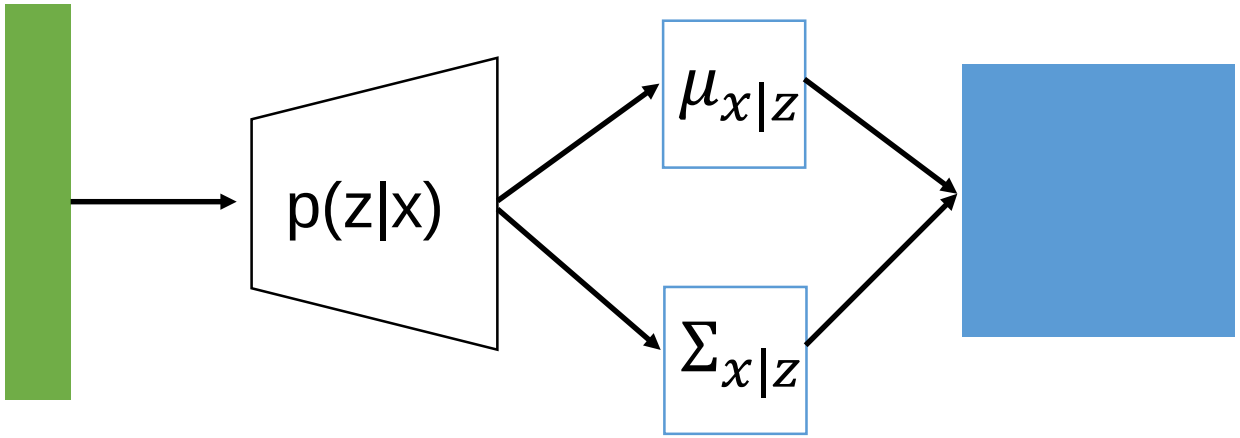


Optimizing using both reconstruction loss and KL divergence loss

Variational Autoencoders

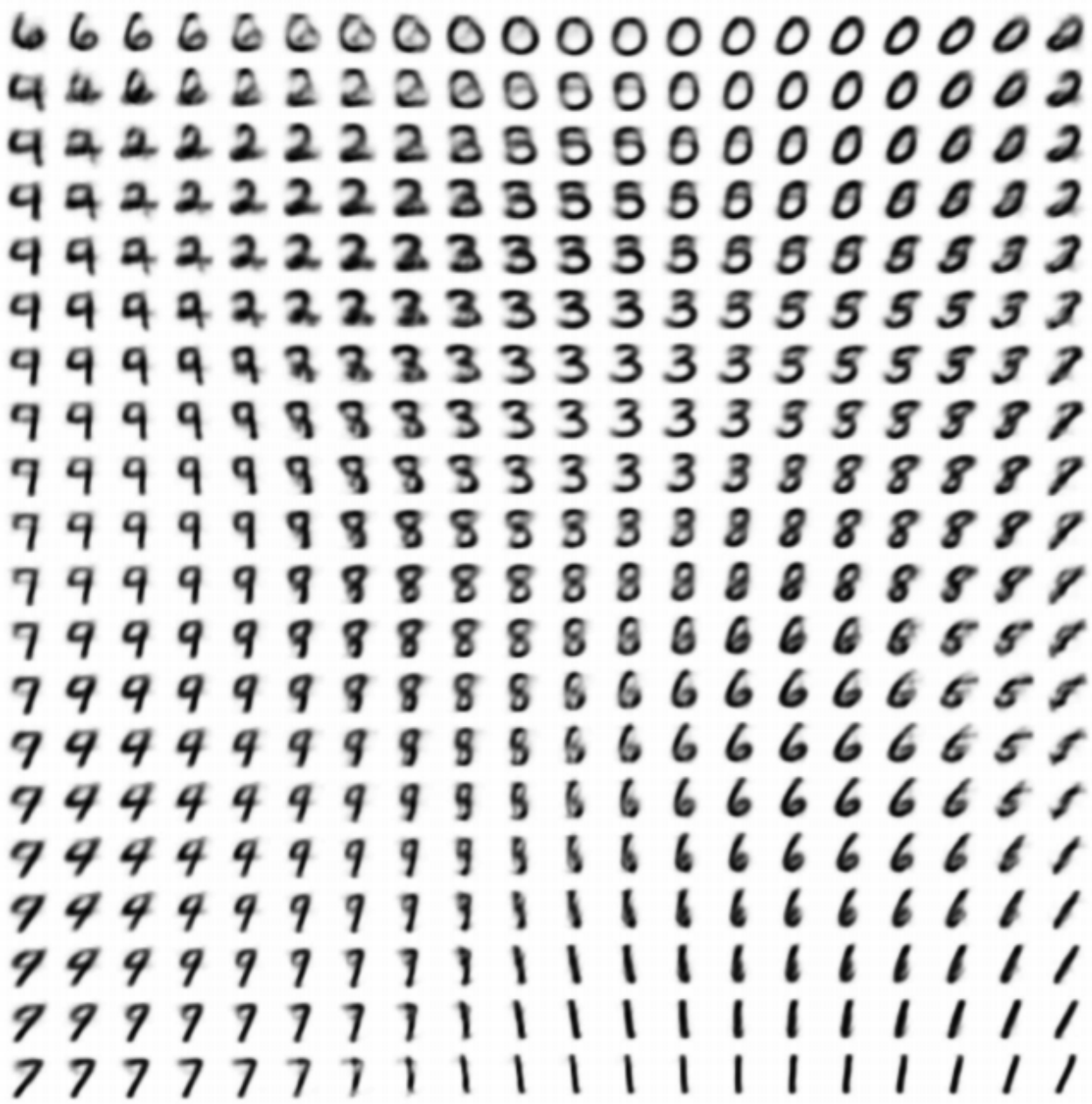
During image generation

1. Sample z from prior distribution (usually N(0,1))
2. Pass through the decoder
3. Sample x from x|z ~ N($\mu_{x|z}$, $\Sigma_{x|z}$) (usually $\mu_{x|z}$)

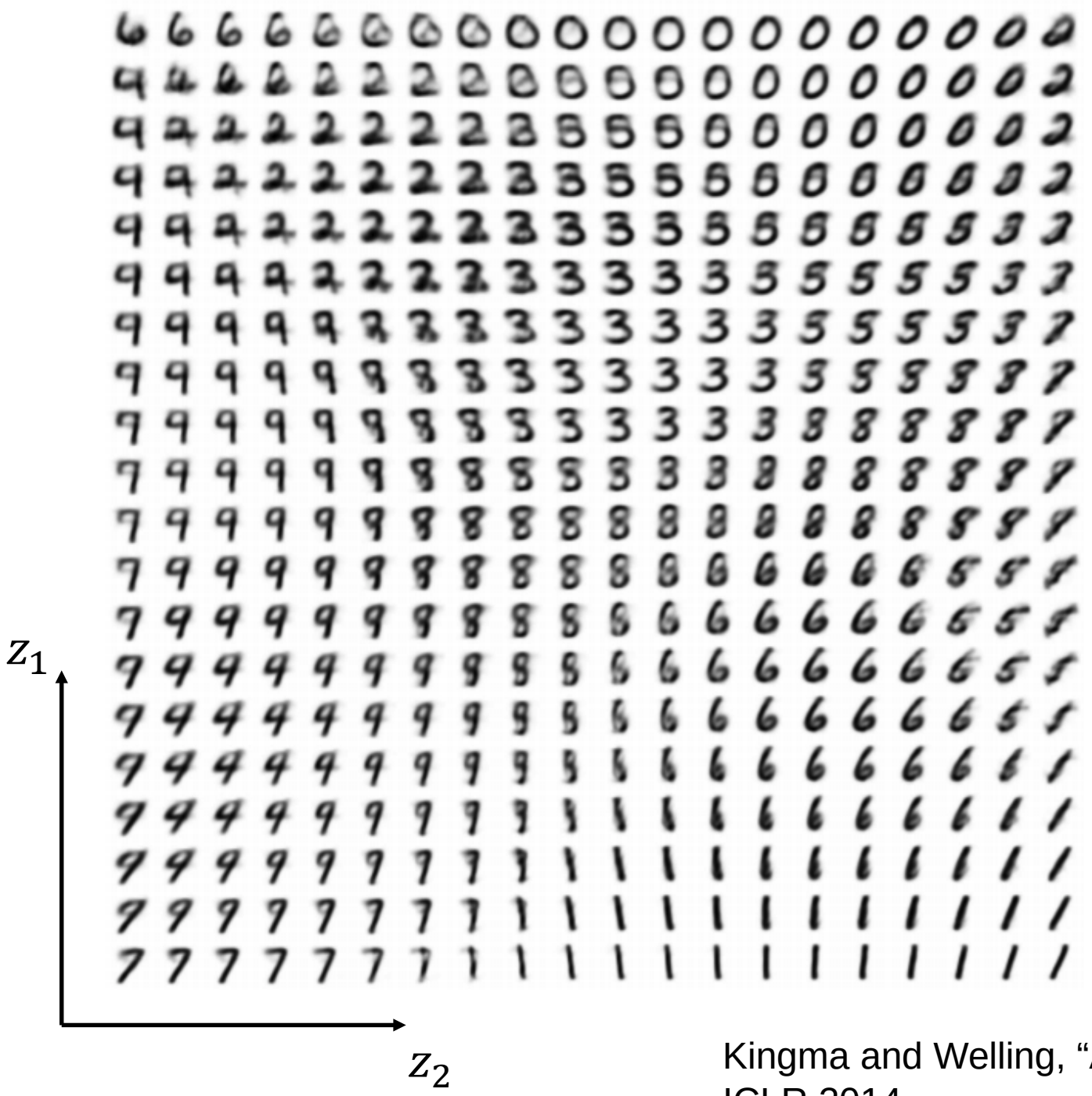


sample z from p(z)

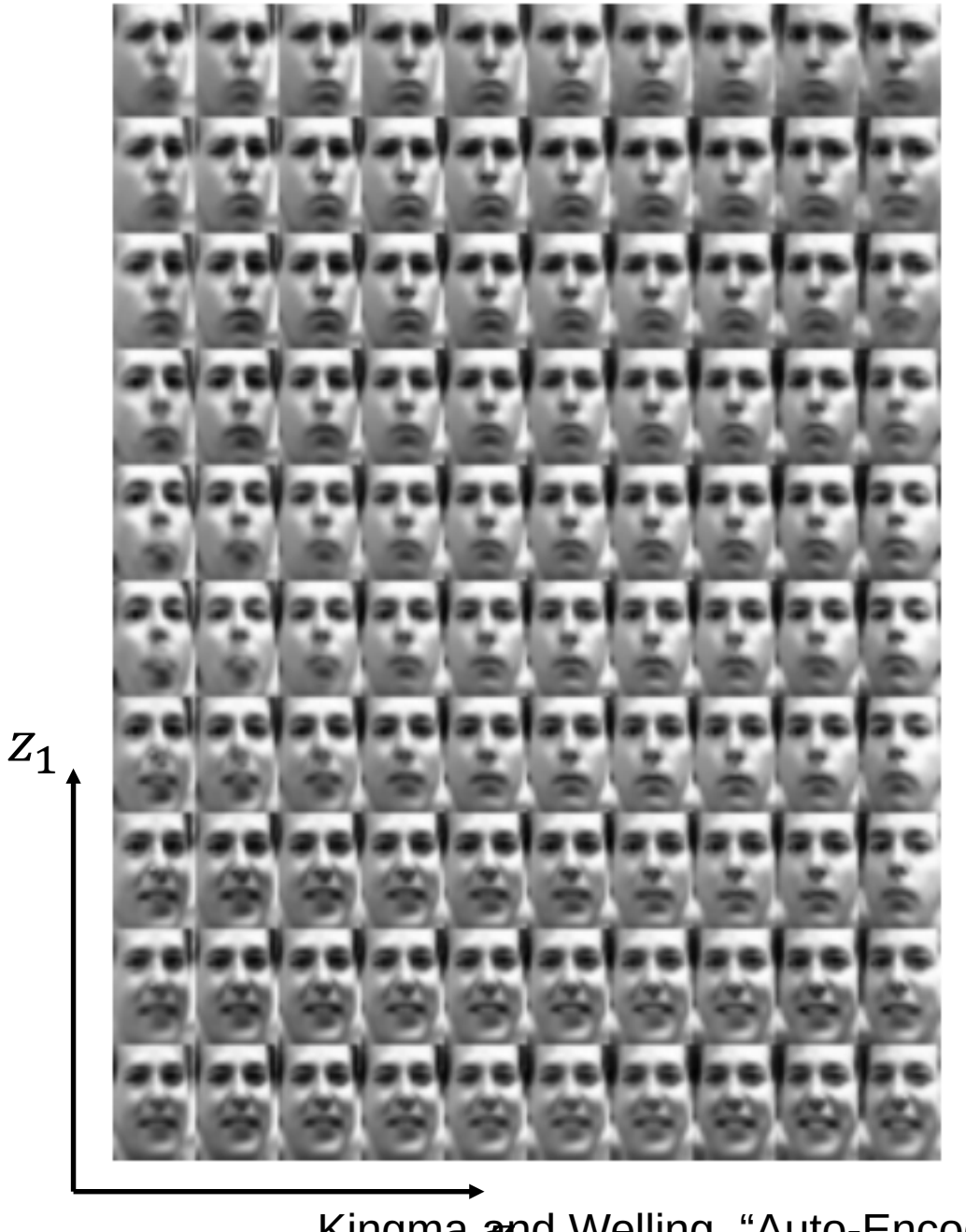
sample x from
x|z ~ N($\mu_{x|z}$, $\Sigma_{x|z}$)



Kingma and Welling, “Auto-Encoding Variational Bayes
ICLR 2014



Kingma and Welling, “Auto-Encoding Variational Bayes
ICLR 2014



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Limit of VAE

Tend to generate blurry images.

Anders Larsen et al. 2