autoencoder

stats403_deep_learning

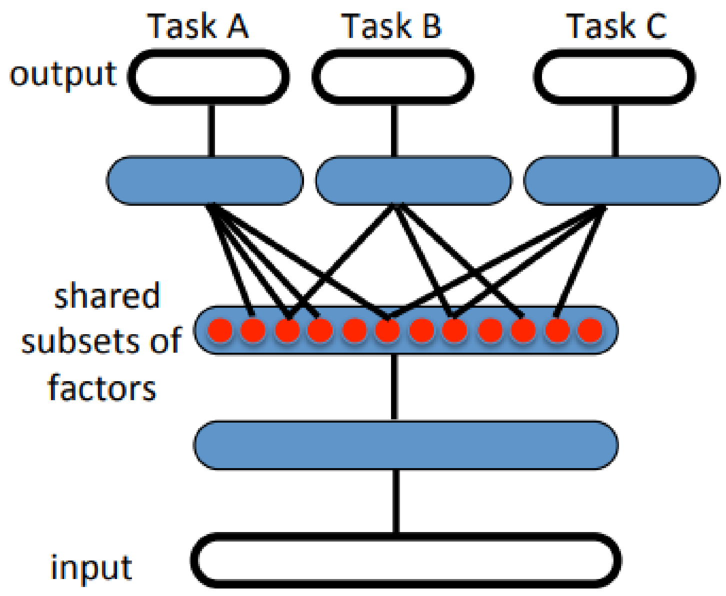
spring_2025

lecture_3

3.1 representation learning and PCA

representation

- the success of machine learning algorithms depends heavily on data representation
- speech recognition, object recognition, natural language processing, transfer learning, …
- human engineered features are costly
- representation learning algorithms learn representations that capture underlying factors for particular tasks



Bengio et al. (2013)


ESPRESSO
ESPRESSO
DOPPIO
DOUBLE ESPRESSO
AMERICANO
HOT WATER
ESPRESSO
CORTADO
STEAMED MILK
ESPRESSO
FLAT WHITE
STEAMED MILK
ESPRESSO
MACCHIATO
MILK FOAM
ESPRESSO
CAPPUCCINO
MILK FOAM
STEAMED MILK
ESPRESSO
LATTE
MILK FOAM
STEAMED MILK
ESPRESSO
MOCHA
MILK FOAM
STEAMED MILK
HOT CHOCOLATE
ESPRESSO
CON PANNA
CREAM
ESPRESSO
AFFOGATO
ESPRESSO
ICE CREAM
IRISH COFFEE
CREAM
WHISKEY
ESPRESSO
yuzubakes.com

- coffee = coefficient × basis

- basis ∈ {espresso, hot water, steamed milk, milk foam, hot chocolate, cream, ice cream, whiskey}

- e.g., mocha = (1,0,1,1,1,0,0,0)

simple representation: principal component analysis (PCA)

- unsupervised representation learning

- each data point $\boldsymbol{x} \in \mathbb{R}^D$ can be decomposed as $\boldsymbol{x} = z_1 \boldsymbol{a}_1 + \cdots + z_D \boldsymbol{a}_D$.
- here, $\boldsymbol{a}_1, \dots, \boldsymbol{a}_D$ are bases determined by the complete dataset which is regarded as a new chart compared to the standard coordinate chart.
- $z_1, \dots, z_D$ are coefficients which represent the data point.

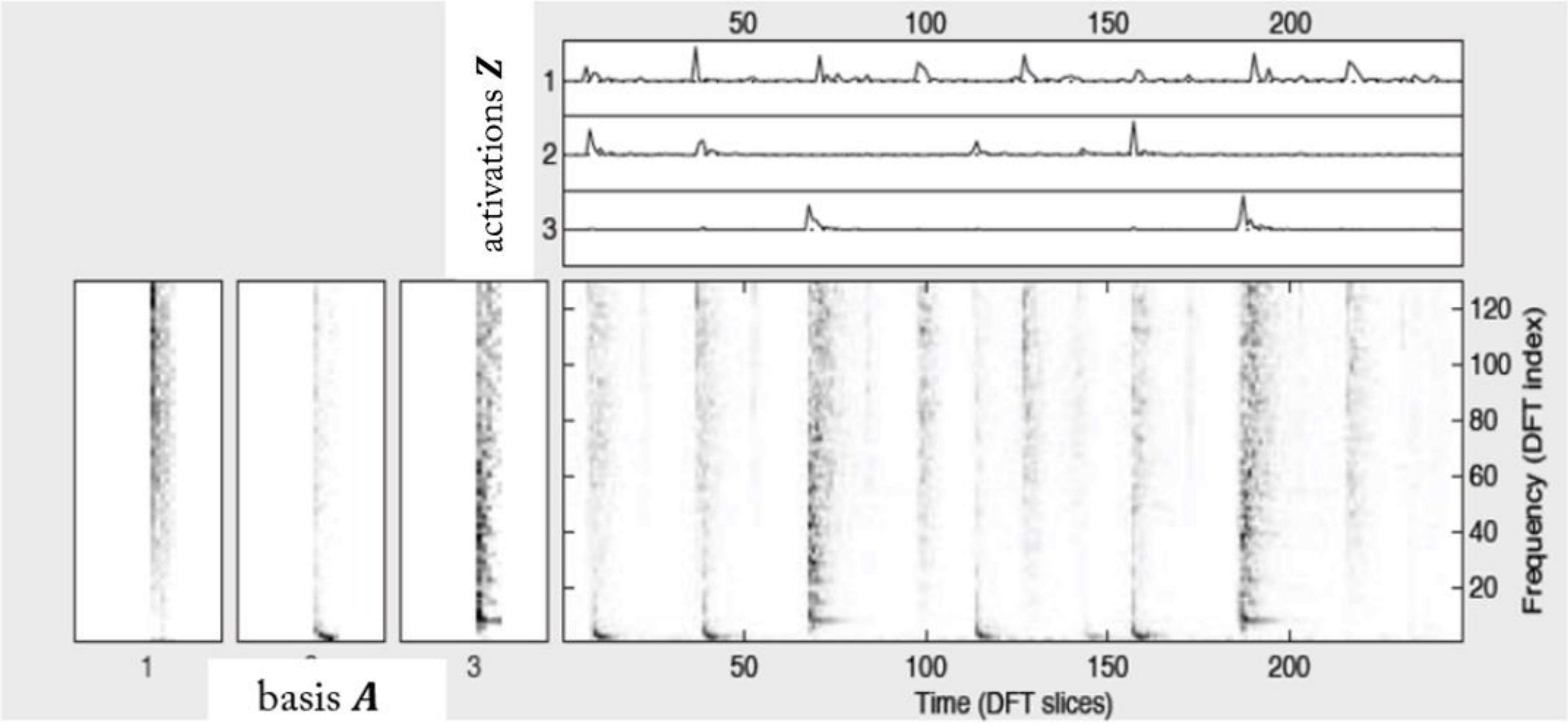
- Suppose only $\boldsymbol{a}_1, \dots \boldsymbol{a}_M$ $(M < D)$ are important (principal) components, $\boldsymbol{x}$ can be roughly represented as $\boldsymbol{x} = z_1 \boldsymbol{a}_1 + \cdots + z_M \boldsymbol{a}_M$

simple representation: principal component analysis (PCA)

$$\boldsymbol{X} = [\underbrace{\boldsymbol{x}_1 \ldots \boldsymbol{x}_N}_{\text{list of data}}] = [\underbrace{\boldsymbol{a}_1 \ldots \boldsymbol{a}_M}_{\text{bases}}][\underbrace{\boldsymbol{z}_1 \ldots \boldsymbol{z}_N}_{\text{list of coefficients}}] = \boldsymbol{A}\,\boldsymbol{Z}$$

simple representation: principal component analysis (PCA)

known as Karhunen-Loève Transform (KLT) in audio processing

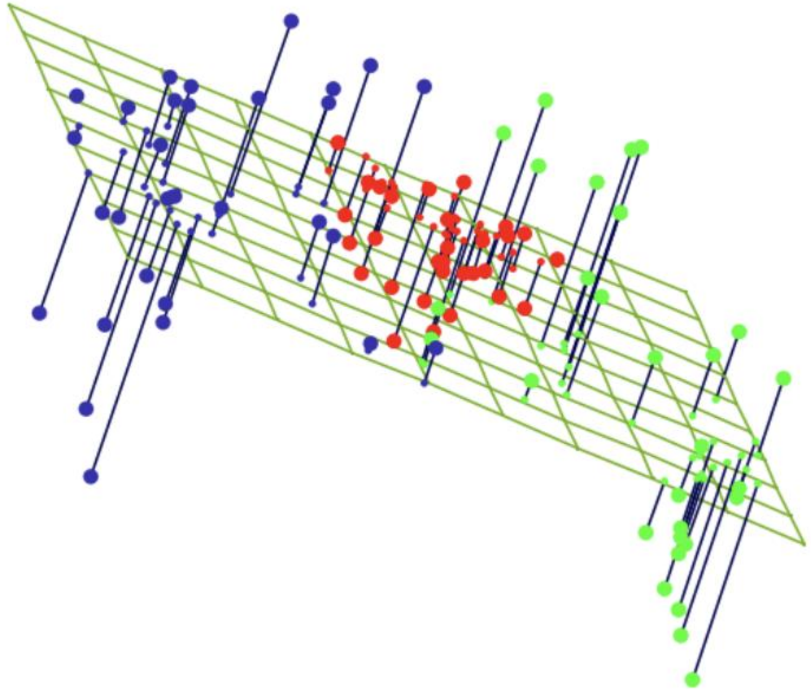


example: audio basis (see link)

- unsupervised method
- provides a geometric representation
- based on feature matrix factorization
- closely related to Latent Semantic Analysis
- simplest: PCA / Karhunen—Loève Transform
- more sophisticated methods exist (NMF, ICA, etc.)

simple representation: principal component analysis (PCA)

- projects data onto a lower dimensional subspace in a way that is optimal in $\sum||\boldsymbol{Az}-\boldsymbol{x}||^2$ sense
- can be efficiently estimated using SVD



simple representation: principal component analysis (PCA)

- for each data point $\boldsymbol{x}$, we can regard the coefficients $\boldsymbol{z}$ as a vector of latent "code"
- then the representation is

$$g(\boldsymbol{z}) = \boldsymbol{A}\boldsymbol{z} \quad \text{where} \qquad \boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$$

- the optimal code should be close to $\boldsymbol{x}$:

$$\boldsymbol{z}^* = \underset{\boldsymbol{z}\in\mathbb{R}^M}{\operatorname{argmin}} \|\boldsymbol{x} - g(\boldsymbol{z})\|^2 = \underset{\boldsymbol{z}\in\mathbb{R}^M}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{A}\boldsymbol{z}\|^2$$

simple representation: principal component analysis (PCA)

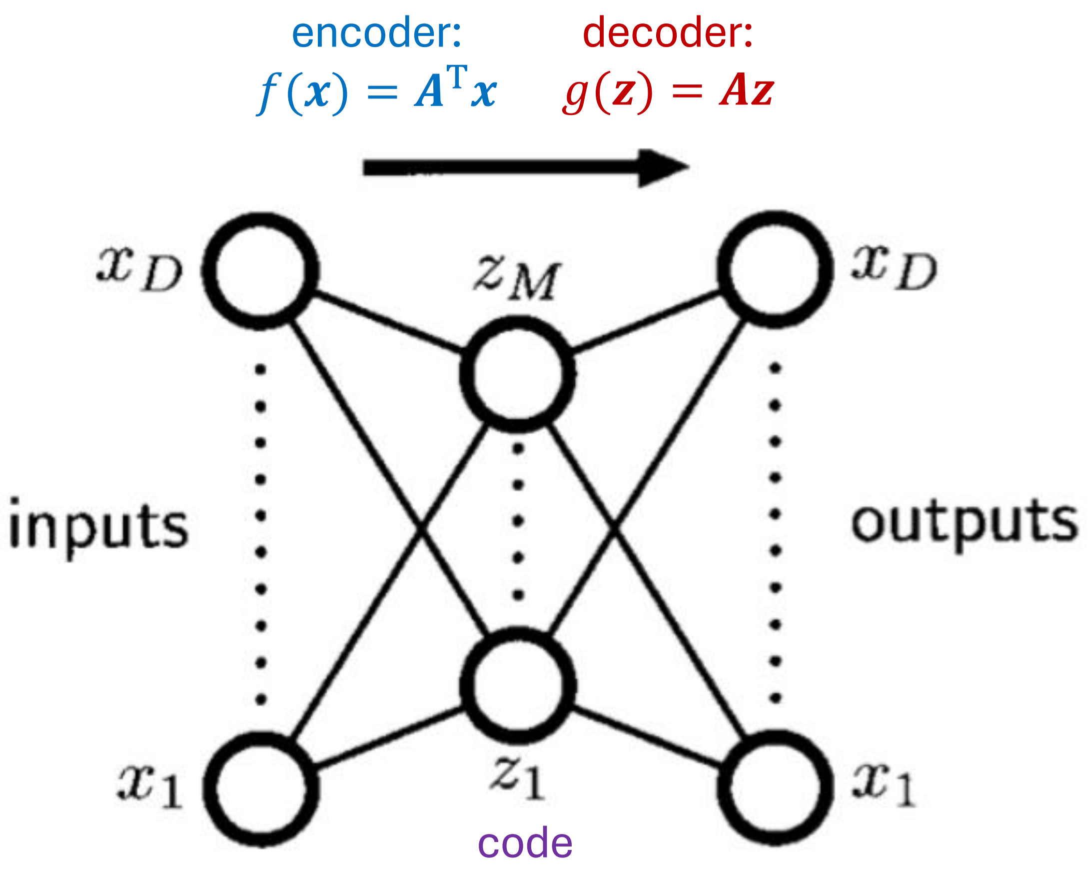
- solving $\min_{\boldsymbol{z} \in \mathbb{R}^M} \|\boldsymbol{x} - \boldsymbol{A}\boldsymbol{z}\|^2$:

simple representation: principal component analysis (PCA)

- therefore, the representation is $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{A}\boldsymbol{A}^T\boldsymbol{x} = g \circ f(\boldsymbol{x})$

- in this case, it is not hard to derive that $\boldsymbol{A} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_M]$ where the columns are right singular vectors of the data matrix corresponding to the M largest singular values

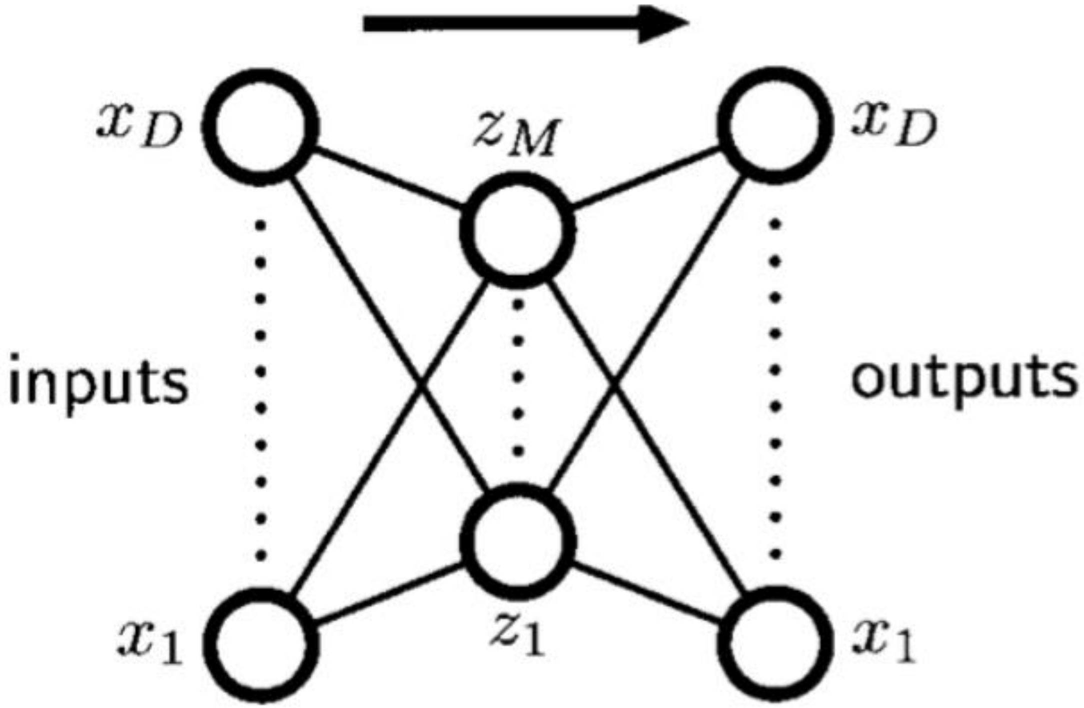
- we can call f an encoder and g a decoder



3.2 autoencoder (AE)

autoencoder

- the above view of PCA gives a linear autoencoder (AE)
- in general, both the encoder and the decoder of AE can be implemented using neural networks (NN)
- when lifting the restriction of "linearity", AE gains expressivity while still being constrained by the dimensionality



autoencoder

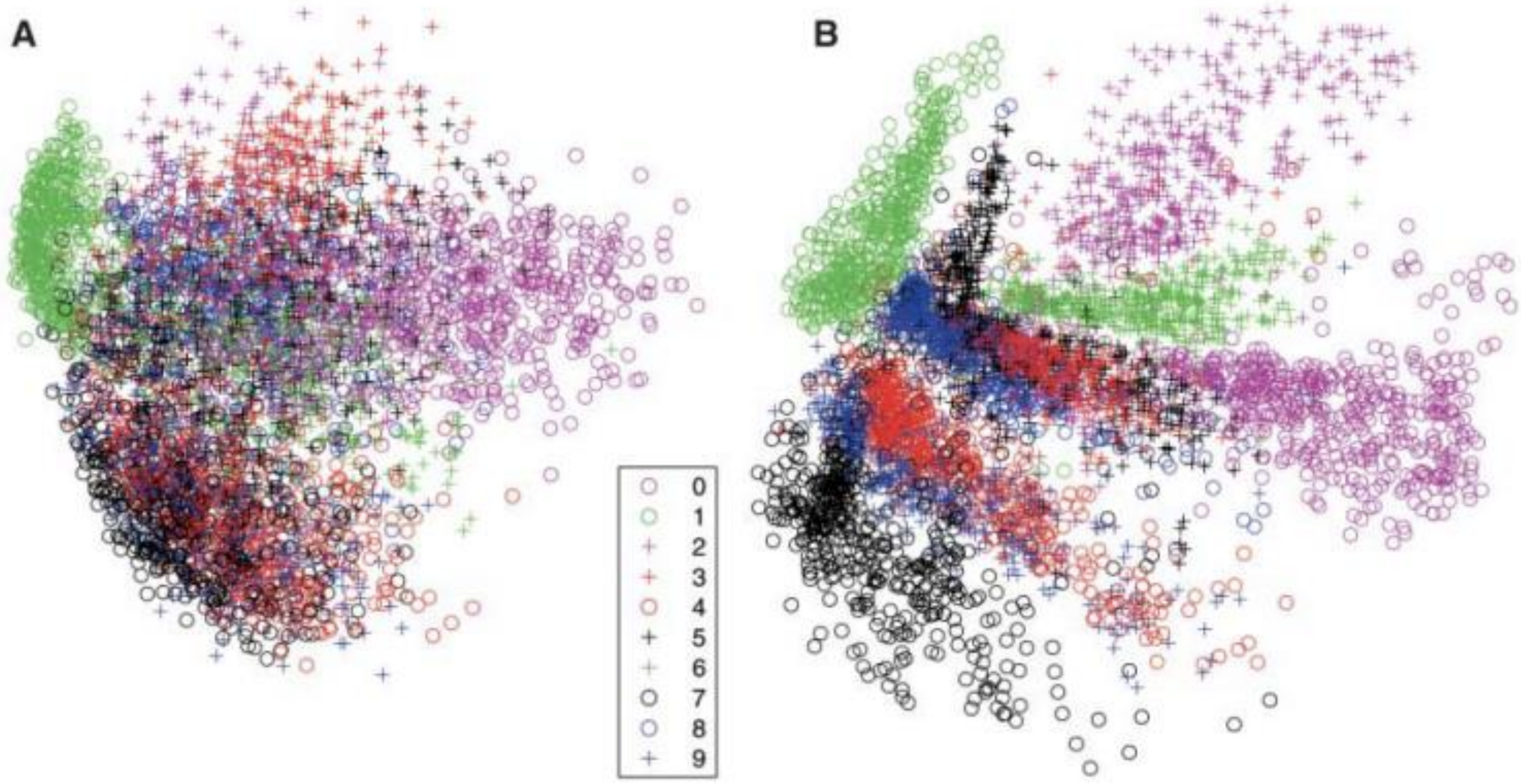
- with NNs, one can create "deep" encoders f and "deep" decoders g
- the loss function is the "reconstruction error", which represents the overall quality of representation:

$$\sum_{\boldsymbol{x}} \|\boldsymbol{x} - g \circ f(\boldsymbol{x})\|^2$$

- in addition to expressivity, using deep NNs may offer other advantages
  - allowing us to enforce constraints (sparsity etc.)
  - reducing computational cost and the amount of training data needed
  - yielding much better compression compared to shallow/linear AE

autoencoder

- two-dimensional codes by [A] linear autoencoder (PCA) and [B] NN-implemented autoencoder for a 10-class digit dataset



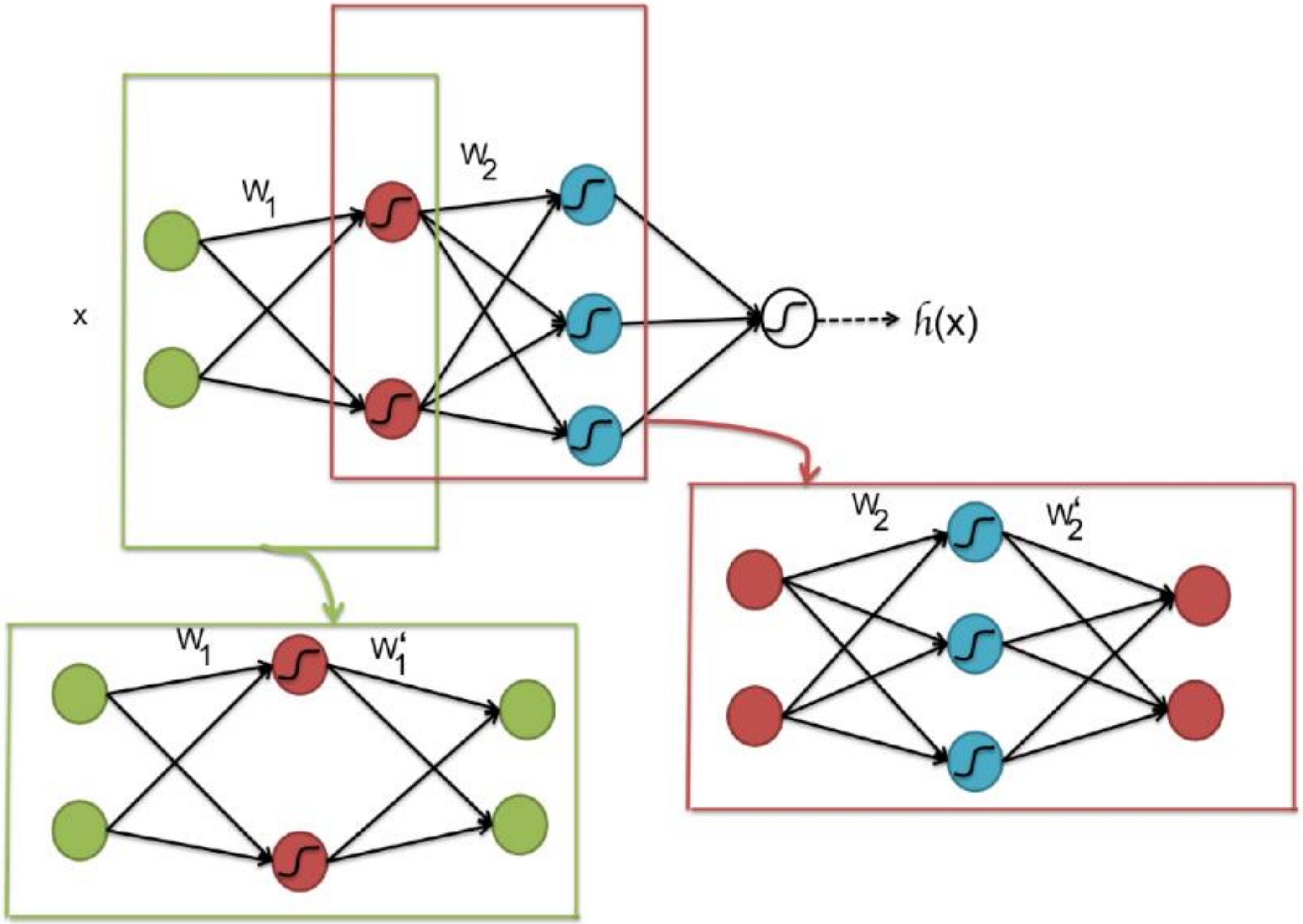
application of AE

- dimension reduction
  - lower-dimensional representations can improve performance on many tasks such as classification

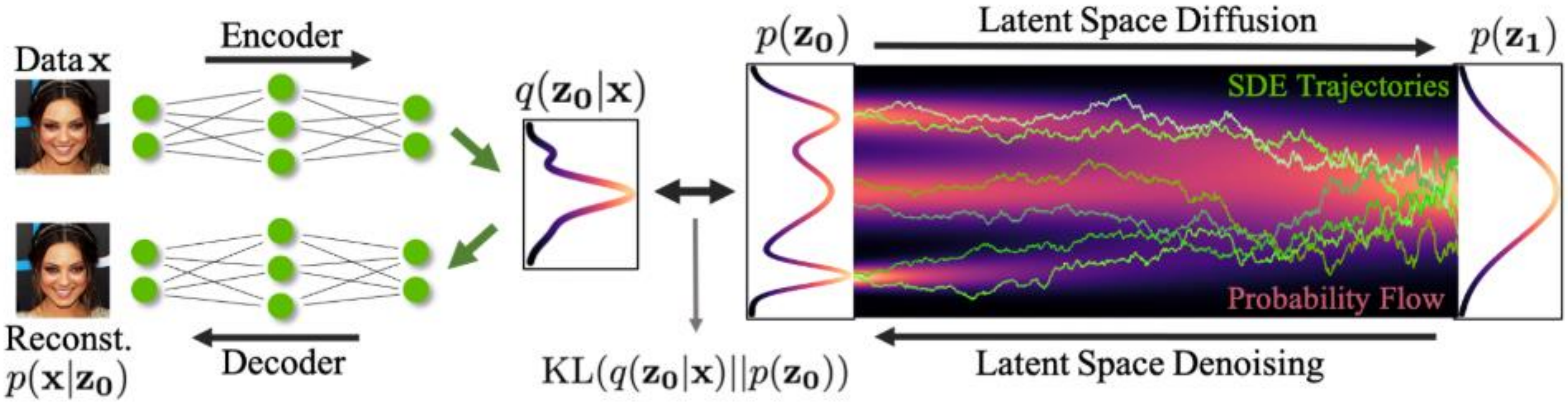
- pre-training

- hashing and information retrieval
  - switching entries (0,1) codes quickly reveals similar database entries

application of AE



application of AE



3.3 probabilistic PCA (PPCA)

why probabilistic?

- unsupervised learning learns good data representations

- this requires we understand the data distribution since a dataset is considered as examples sampled from the data distribution

- we "understand the data distribution" if we can
  - either describe the density $p(\mathbf{x})$ where data are sampled from
  - or generate new examples from $p(\mathbf{x})$

probabilistic PCA

- suppose $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$

- also suppose $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}\left(\mathbf{x}\middle|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I}\right)$
- equivalently, give the random vector $\mathbf{z}$, we can get $\mathbf{x}$ following

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\varepsilon} \quad \text{with} \quad \boldsymbol{\varepsilon} \sim \mathcal{N}\left(\boldsymbol{\varepsilon}\middle|\mathbf{0}, \sigma^2\mathbf{I}\right)$$

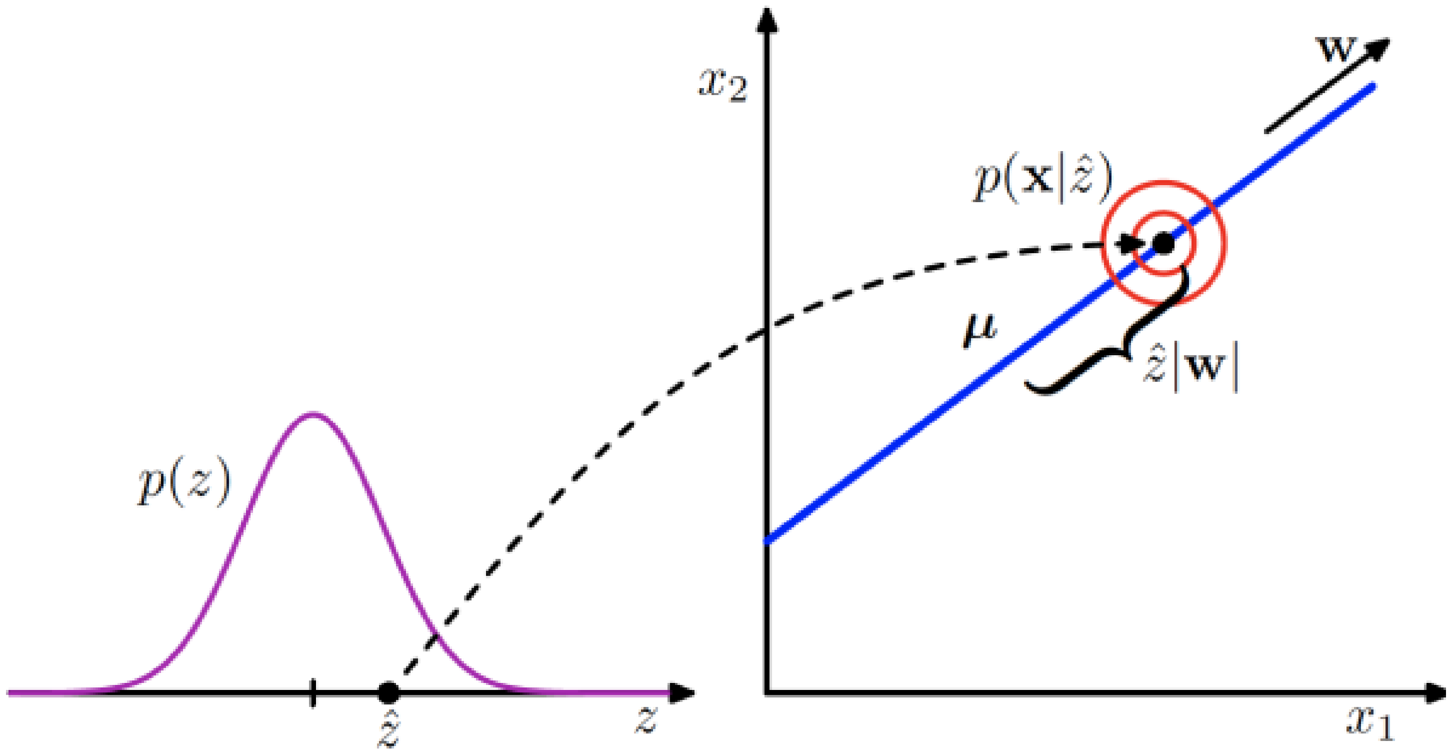
probabilistic PCA

- $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\varepsilon}$ can be read in a generative manner

1. sample a "code" $\mathbf{z}$ from $\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$
2. apply the transformation $\mathbf{W}\mathbf{z} + \boldsymbol{\mu}$ to the sampled code
3. add a random noise $\boldsymbol{\varepsilon} \sim \mathcal{N}\left(\boldsymbol{\varepsilon}\middle|\mathbf{0}, \sigma^2\mathbf{I}\right)$

- steps 1-3 "decodes" $\mathbf{z}$ to obtain $\mathbf{x}$

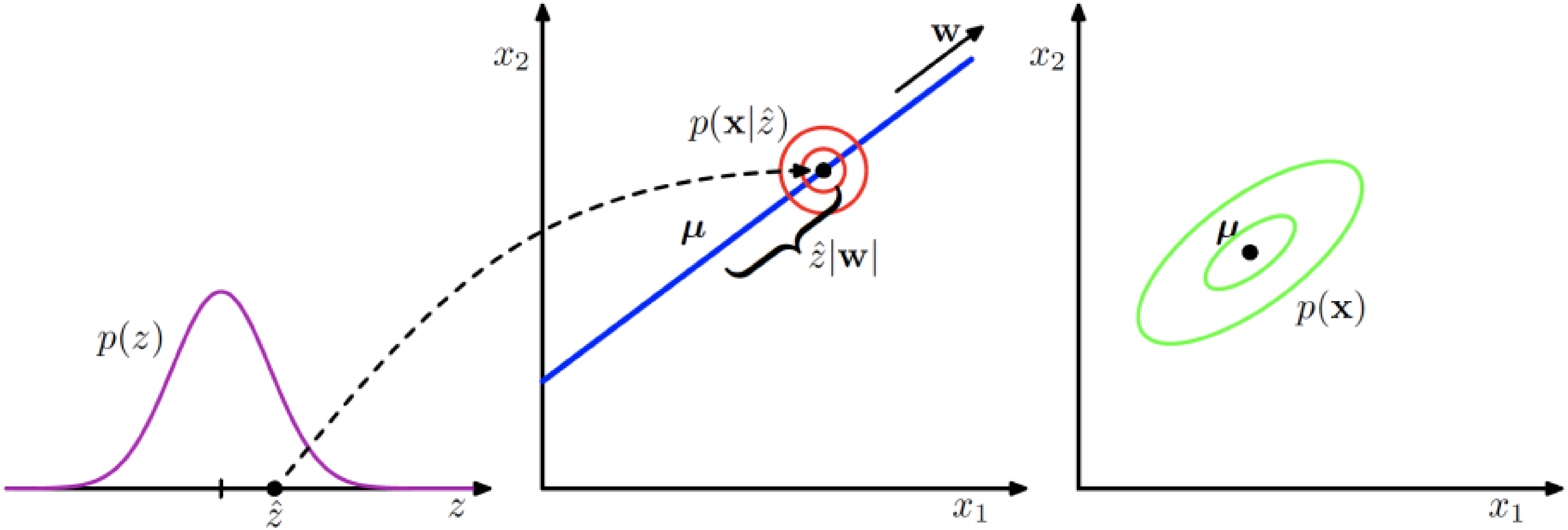
probabilistic PCA



probabilistic PCA

- over all possible **z**,

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$



probabilistic PCA

- thanks to the linear model, it is easy to show that

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{C})$$

where

$$\mathbf{C} = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I}$$

- we can derive that

$$\mathbf{C}^{-1} = \sigma^{-1}\mathbf{I} - \sigma^{-2}\mathbf{W}\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}$$

where

$$\mathbf{M} = \mathbf{W}^{\mathrm{T}}\mathbf{W} + \sigma^2\mathbf{I}$$

- we can then derive the “encoder”

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}\middle|\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}(\mathbf{x} - \boldsymbol{\mu}), \sigma^{-2}\mathbf{M}^{-1}\right)$$

probabilistic PCA

- given a dataset $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^{N}$, we have

$$\begin{aligned}\ln p(\mathbf{X}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2) &= \sum_{n=1}^{N} \ln p(\mathbf{x}_n|\mathbf{W}, \boldsymbol{\mu}, \sigma^2) \\ &= -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{C}| - \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{x}_n - \boldsymbol{\mu})\end{aligned}$$

with which we can derive the maximum likelihood estimation of the parameters $\boldsymbol{\mu}, \mathbf{W}, \sigma^2$

probabilistic PCA

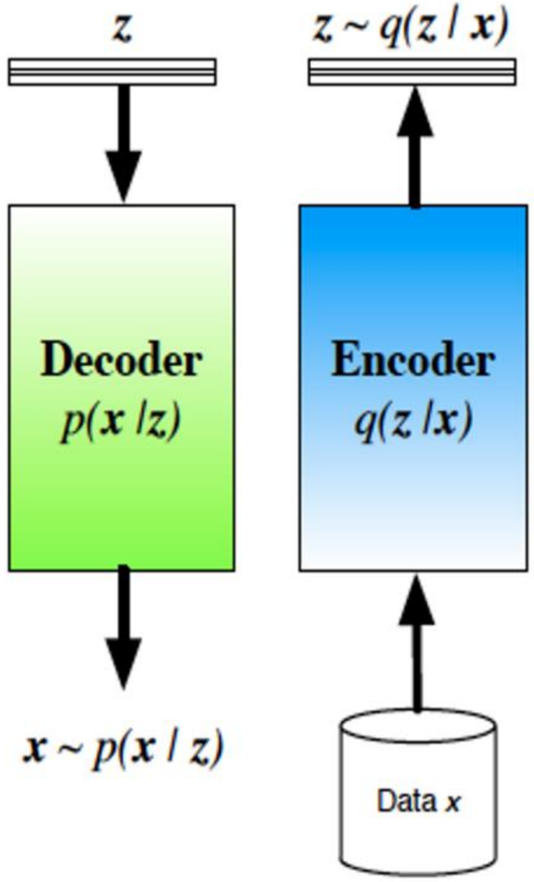
- in the linear case, we can derive $p(\mathbf{z}|\mathbf{x})$ from $p(\mathbf{x}|\mathbf{z})$
- similar to how AE generalizes PCA, we would like to generalize PPCA to nonlinear cases
- of course, we don't expect to express $p(\mathbf{z}|\mathbf{x})$ or $p(\mathbf{x})$ from $p(\mathbf{x}|\mathbf{z})$ once $p(\mathbf{x}|\mathbf{z})$ takes the form of a neural network
- nevertheless, we can learn a posterior, say $q(\mathbf{z}|\mathbf{x})$, to approximate $p(\mathbf{z}|\mathbf{x})$
- we hope it is still possible to derive maximum likelihood estimation of the paramters in that case

3.4 variational autoencoder (VAE)

variational encoder-decoder

- assume that we use neural networks to implement a decoder for $p(\boldsymbol{x}|\boldsymbol{z})$ and an encoder $q(\boldsymbol{z}|\boldsymbol{x})$

- given a data point $\boldsymbol{x}$, we can implement the encoder to find the corresponding code
- given a code $\boldsymbol{z}$ sampled from the latent distribution $p(\boldsymbol{z})$, we can implement the decoder to recover the data example



variational encoder-decoder

- suppose the parameters of the decoder $p(\boldsymbol{x}|\boldsymbol{z})$ are summarized as $\boldsymbol{\theta}$; the parameters of the encoder $q(\boldsymbol{z}|\boldsymbol{x})$ are summarized as $\boldsymbol{\phi}$

- it is custom to write $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ and $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ to emphasize the parametric representations

- such probabilistic setting of AE is called a variational autoencoder (VAE)

- to derive maximum likelihood estimations of the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, we proceed as follows

$p(\boldsymbol{x})$

evidence lower bound

(we will use KL and D_{KL} interchangeably)

$$
\begin{aligned}
\log p(\boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] + D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}|\boldsymbol{x})) \\
&\geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right]
\end{aligned}
$$

- in Bayesian statistics, considering $p(\boldsymbol{z})$ as the prior, $p(\boldsymbol{x}|\boldsymbol{z})$ as the likelihood, and $q(\boldsymbol{z}|\boldsymbol{x})$ as the posterior, $p(\boldsymbol{x})$ is called the evidence; thus, the lower bound presented above is called the Evidence Lower BOund (ELBO)

evidence lower bound

- in the following, we will derive the best $\boldsymbol{\phi}$ that maximizes the ELBO (so that on one hand, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{z}|\boldsymbol{x})$; on the other hand, $p(\boldsymbol{x})$ will also be maximized w.r.t. the parameters)
- we need to maximize the following:

$$
\begin{aligned}
\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}))}_{\text{prior matching term}}
\end{aligned}
$$

why?

evidence lower bound

- for the prior matching term, we often choose to model the prior as
$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$$

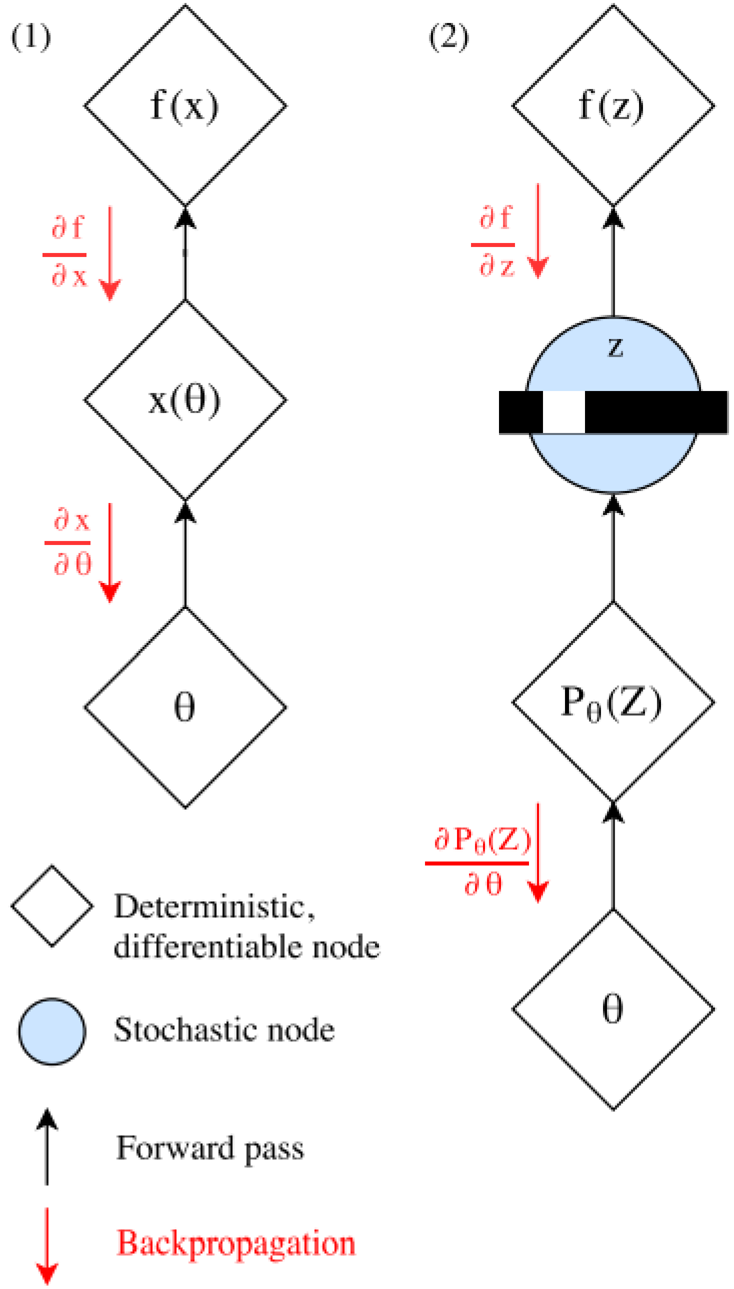
  and the posterior as (Gaussian with a diagonal covariance matrix)
$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\mathbf{z} \mid \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))$$

- with vectors representations of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, the expression of KL divergence is, from an example in STATS303🙀,
$$D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})\right) = -\frac{1}{2}\sum_{j=1}^{M}(1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2)$$

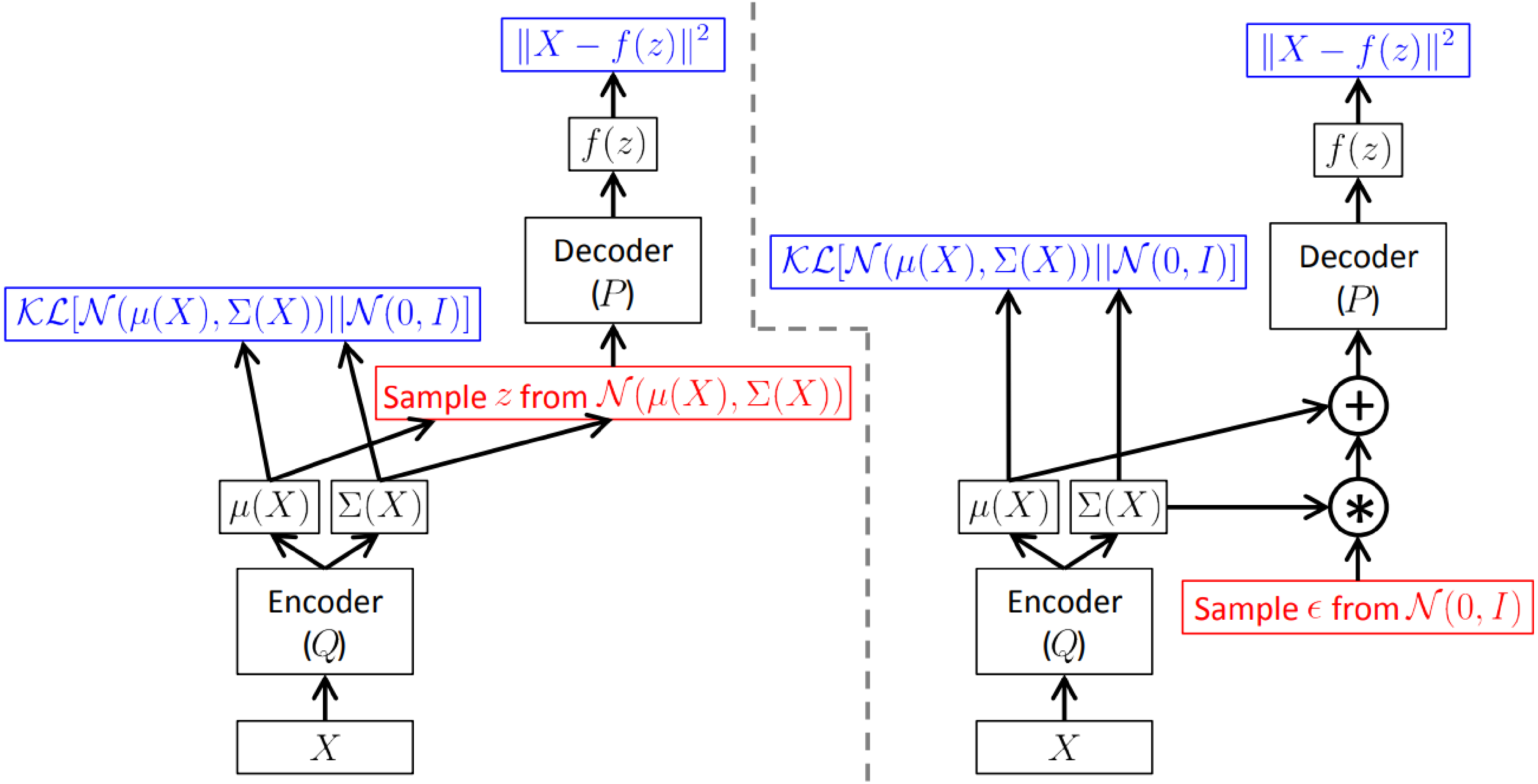
backpropagation for stochastic layers

- in AE, the gradients can be computed via backpropagation since both the encoder and decoder are deterministic and differentiable

- however, in VAE, the presence of stochastic nodes preclude backpropagation as the sampler function does not have a well-defined gradient

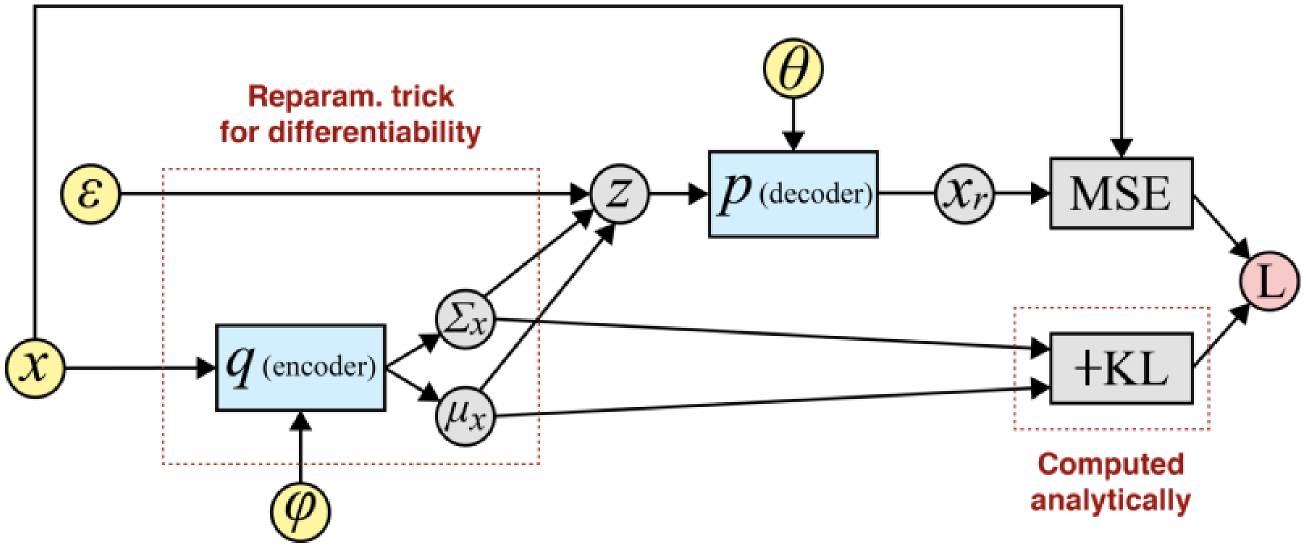


the reparameterization trick



overall VAE regime

$\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x = M(\mathbf{x}), \Sigma(\mathbf{x})$	Push **x** through encoder
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, 1)$	Sample noise
$\mathbf{z} = \boldsymbol{\epsilon}\boldsymbol{\sigma}_x + \boldsymbol{\mu}_x$	Reparameterize
$\mathbf{x}_r = p_\theta(\mathbf{x} \mid \mathbf{z})$	Push **z** through decoder
$\text{recon. loss} = \text{MSE}(\mathbf{x}, \mathbf{x}_r)$	Compute reconstruction loss
$\text{var. loss} = +\text{KL}[\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x) \| \mathcal{N}(0, I)]$	Compute variational loss
$\text{L} = \text{recon. loss} + \text{var. loss}$	Combine losses



pytorch implementation of VAE

see here

Thank you!

Reference

- Ch 2.12, 14.*Deep Learning*.
- Ch 12.2, *Pattern Recognition and Machine Learning*.
- Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.
- https://medium.com/@solopchuk/tutorial-on-active-inference-30edcf50f5dc
- Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, *35*(8), 1798-1828.
- Jang, E., Gu, S., & Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.
- Luo, C. (2022). Understanding diffusion models: A unified perspective. *arXiv preprint arXiv:2208.11970*.