Deep Learning & Applications

Vicenç Gómez & Anders Jonsson

Session 2A: Latent variable models

Introduction

Autoencoders

Normalizing Flows

Session 2A (2nd hour)

Introduction

Autoencoders

Normalizing Flows

Learning Problems

Supervised Learning:

- Learn model using dataset with **data-label pairs** $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$
- Examples: Classification, regression, structured prediction

Unsupervised Learning:

- Learn model using dataset **without labels** $\{\mathbf{x}_i\}_{i=1}^N$
- Examples: Clustering, dimensionality reduction, generative models
- Some of them use latent variables to capture structure $\rightarrow$ topic for today

Latent Variable Models

LVMs map between **observation space** $\mathbf{x} \in \mathbb{R}^D$ and **latent space** $\mathbf{z} \in \mathbb{R}^Q$:

$$(f_{\mathbf{w}} : \mathbf{x} \mapsto \mathbf{z}) \qquad g_{\mathbf{w}} : \mathbf{z} \mapsto \hat{\mathbf{x}}$$

- One **latent variable** gets associated with each data point in the training set
- The latent vectors are smaller than the observations ($Q < D$) $\Rightarrow$ **compression**
- Models are linear or non-linear, deterministic or stochastic, with/without encoder

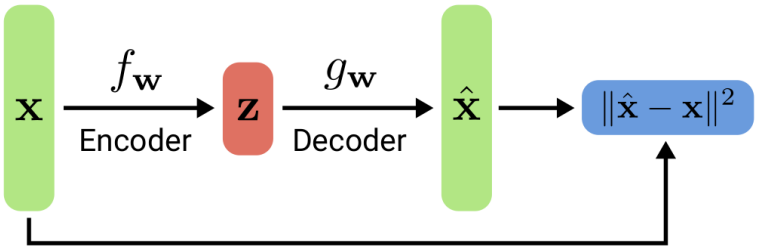
A little taxonomy:

	Deterministic	*Probabilistic*
Linear	Principal Component Analysis	Probabilistic PCA
Nonlinear, Noninvertible	**Deep Autoencoder**	**Variational Autoencoder**
Nonlinear, Invertible		**Normalizing flows**

Autoencoders

Autoencoders comprise an **encoder** $f_{\mathbf{w}}$ as well as a **decoder** $g_{\mathbf{w}}$:

$$f_{\mathbf{w}} : \mathbf{x} \mapsto \mathbf{z} \qquad g_{\mathbf{w}} : \mathbf{z} \mapsto \hat{\mathbf{x}}$$

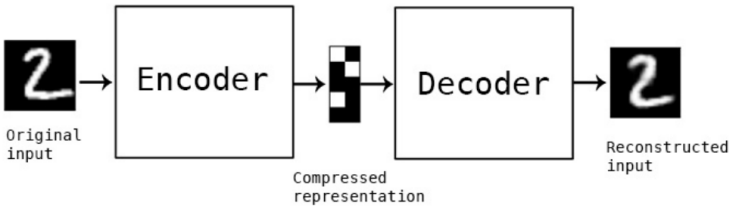


- Models of this type are called **autoencoders** as they predict their input as output

Autoencoders

Autoencoders comprise an **encoder** $f_{\mathbf{w}}$ as well as a **decoder** $g_{\mathbf{w}}$:

$$f_{\mathbf{w}} : \mathbf{x} \mapsto \mathbf{z} \qquad g_{\mathbf{w}} : \mathbf{z} \mapsto \hat{\mathbf{x}}$$



- Models of this type are called **autoencoders** as they predict their input as output

Generative Models

- Generative modeling is a broad area of ML which deals with models of **probability distributions** $p(\mathbf{x})$ over data points $\mathbf{x}$

Generative Models

- Generative modeling is a broad area of ML which deals with models of **probability distributions** $p(\mathbf{x})$ over data points $\mathbf{x}$
- The generative model's task is to capture dependencies / structural regularities in the data

Generative Models

- Generative modeling is a broad area of ML which deals with models of **probability distributions** $p(\mathbf{x})$ over data points $\mathbf{x}$
- The generative model's task is to capture dependencies / structural regularities in the data
- **Generative latent variable models** capture the structure in the **latent variables**

Generative Models

- Generative modeling is a broad area of ML which deals with models of **probability distributions** $p(\mathbf{x})$ over data points $\mathbf{x}$
- The generative model's task is to capture dependencies / structural regularities in the data
- **Generative latent variable models** capture the structure in the **latent variables**
- Intuitively, we are trying to establish a **theory** for what we observe

Generative Models

- Generative modeling is a broad area of ML which deals with models of **probability distributions** $p(\mathbf{x})$ over data points $\mathbf{x}$
- The generative model's task is to capture dependencies / structural regularities in the data
- **Generative latent variable models** capture the structure in the **latent variables**
- Intuitively, we are trying to establish a **theory** for what we observe
- Some generative models (normalizing flows) allow for computing $p(\mathbf{x})$

Generative Models

- Generative modeling is a broad area of ML which deals with models of **probability distributions** $p(\mathbf{x})$ over data points $\mathbf{x}$
- The generative model's task is to capture dependencies / structural regularities in the data
- **Generative latent variable models** capture the structure in the **latent variables**
- Intuitively, we are trying to establish a **theory** for what we observe
- Some generative models (normalizing flows) allow for computing $p(\mathbf{x})$
- Others (VAEs) only approximate $p(\mathbf{x})$

Generative Models

- Generative modeling is a broad area of ML which deals with models of **probability distributions** $p(\mathbf{x})$ over data points $\mathbf{x}$
- The generative model's task is to capture dependencies / structural regularities in the data
- **Generative latent variable models** capture the structure in the **latent variables**
- Intuitively, we are trying to establish a **theory** for what we observe
- Some generative models (normalizing flows) allow for computing $p(\mathbf{x})$
- Others (VAEs) only approximate $p(\mathbf{x})$
- All generative latent variable models allow to draw samples from $p(\mathbf{x})$

Generative Latent Variable Models

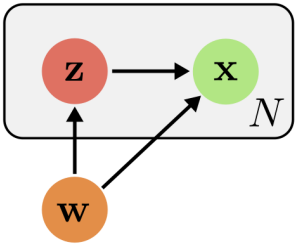
Generative latent variable models often consider a simple **Bayesian model:**

$$p(\mathbf{x}) = \int_{\mathbf{z}} \underbrace{p(\mathbf{z})p(\mathbf{x}|\mathbf{z})}_{\text{Generative Process}} \; d\mathbf{z} = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} p(\mathbf{x}|\mathbf{z})$$

- $p(\mathbf{z})$ is the **prior** over the **latent variable** $\mathbf{z} \in \mathbb{R}^Q$
- $p(\mathbf{x}|\mathbf{z})$ is the **likelihood** (= decoder that transforms $\mathbf{z}$ into a distribution over $\mathbf{x}$)
- $p(\mathbf{x})$ is the **marginal** of the joint distribution $p(\mathbf{x}, \mathbf{z})$

The goal is to maximize $p(\mathbf{x})$ for a given dataset $\mathcal{X}$ by learning the two models $p(\mathbf{z})$ and $p(\mathbf{x}|\mathbf{z})$ such that the latent variables $\mathbf{z}$ best capture the latent structure of the data.

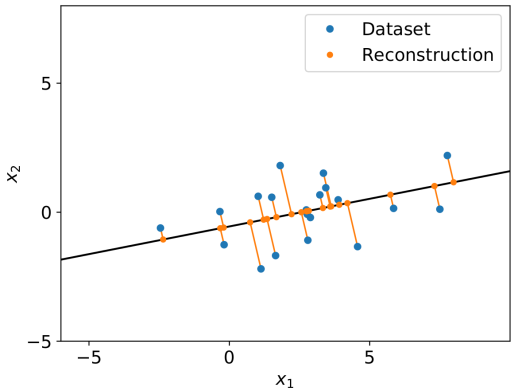
Generative Latent Variable Models



Representation as Graphical Model in Plate Notation:

- Variables inside plates are replicated (we have N data points to explain)
- Each data point $\mathbf{x}$ is associated with a latent variable $\mathbf{z}$
- In contrast, parameters are global (exist only once)
- Remark: We use a single $\mathbf{w}$ to refer to all model parameters

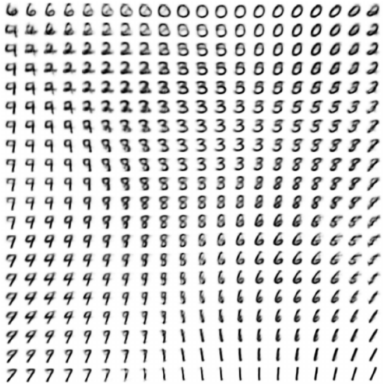
Example: 1D Manifold in 2D Space



Original data $\mathbf{x}$, Reconstruction $\hat{\mathbf{x}}$

Example: Natural Image Manifolds

(a) Learned Frey Face manifold



(b) Learned MNIST manifold

- Visualizing the latent space gives insights into the learned semantics

Session 2A (2nd hour)

Introduction

Autoencoders

Normalizing Flows

Preliminaries

- Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top \in \mathbb{R}^{N \times D}$ be a dataset of observations $\mathbf{x}_i \in \mathbb{R}^D$

Preliminaries

- Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top \in \mathbb{R}^{N \times D}$ be a dataset of observations $\mathbf{x}_i \in \mathbb{R}^D$
- Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^\top \in \mathbb{R}^{N \times Q}$ be corresponding latent vars $\mathbf{z}_i \in \mathbb{R}^Q$

Preliminaries

- Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top \in \mathbb{R}^{N \times D}$ be a dataset of observations $\mathbf{x}_i \in \mathbb{R}^D$
- Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^\top \in \mathbb{R}^{N \times Q}$ be corresponding latent vars $\mathbf{z}_i \in \mathbb{R}^Q$
- While $\mathbf{X}$ is observed, $\mathbf{Z}$ is unobserved and needs to be inferred

Preliminaries

- Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top \in \mathbb{R}^{N \times D}$ be a dataset of observations $\mathbf{x}_i \in \mathbb{R}^D$
- Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^\top \in \mathbb{R}^{N \times Q}$ be corresponding latent vars $\mathbf{z}_i \in \mathbb{R}^Q$
- While $\mathbf{X}$ is observed, $\mathbf{Z}$ is unobserved and needs to be inferred
- Typically, $Q < D$, i.e., we want to obtain a **compressed** representation

Preliminaries

- Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top \in \mathbb{R}^{N \times D}$ be a dataset of observations $\mathbf{x}_i \in \mathbb{R}^D$
- Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^\top \in \mathbb{R}^{N \times Q}$ be corresponding latent vars $\mathbf{z}_i \in \mathbb{R}^Q$
- While $\mathbf{X}$ is observed, $\mathbf{Z}$ is unobserved and needs to be inferred
- Typically, $Q < D$, i.e., we want to obtain a **compressed** representation
- Goal: learn a **bidirectional mapping** $\mathcal{Z} \leftrightarrow \mathcal{X}$ such that as much information of $\mathcal{X}$ as possible is retained in $\mathcal{Z}$

Preliminaries

- Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top \in \mathbb{R}^{N \times D}$ be a dataset of observations $\mathbf{x}_i \in \mathbb{R}^D$
- Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^\top \in \mathbb{R}^{N \times Q}$ be corresponding latent vars $\mathbf{z}_i \in \mathbb{R}^Q$
- While $\mathbf{X}$ is observed, $\mathbf{Z}$ is unobserved and needs to be inferred
- Typically, $Q < D$, i.e., we want to obtain a **compressed** representation
- Goal: learn a **bidirectional mapping** $\mathcal{Z} \leftrightarrow \mathcal{X}$ such that as much information of $\mathcal{X}$ as possible is retained in $\mathcal{Z}$
- In other words, encode $\mathbf{x} \rightarrow \mathbf{x}$ such that decoding $\mathbf{z} \rightarrow \hat{\mathbf{x}}$, leads to a good approximationg of the original $\mathbf{x}$

Preliminaries

- Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top \in \mathbb{R}^{N \times D}$ be a dataset of observations $\mathbf{x}_i \in \mathbb{R}^D$
- Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^\top \in \mathbb{R}^{N \times Q}$ be corresponding latent vars $\mathbf{z}_i \in \mathbb{R}^Q$
- While $\mathbf{X}$ is observed, $\mathbf{Z}$ is unobserved and needs to be inferred
- Typically, $Q < D$, i.e., we want to obtain a **compressed** representation
- Goal: learn a **bidirectional mapping** $\mathcal{Z} \leftrightarrow \mathcal{X}$ such that as much information of $\mathcal{X}$ as possible is retained in $\mathcal{Z}$
- In other words, encode $\mathbf{x} \rightarrow \mathbf{x}$ such that decoding $\mathbf{z} \rightarrow \hat{\mathbf{x}}$, leads to a good approximationg of the original $\mathbf{x}$
- Simplest autoencoder: a **linear** bidirectional mapping $\rightarrow$ Principal Component Analysis (**PCA**)

Principal Component Analysis

- Principal Component Analysis (PCA) reconstructs the input using a linear model

$$\hat{\mathbf{x}}_i = \bar{\mathbf{x}} + \sum_{j=1}^{Q} z_{ij}\mathbf{v}_j$$

Both the PCA decoder and encoder are simple **linear mappings:**

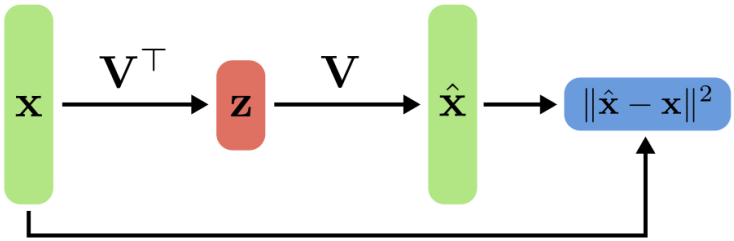
Decoder:

$$\mathbf{x} = \mathbf{V}\mathbf{z} + \bar{\mathbf{x}}$$

Encoder:

$$\mathbf{z} = \mathbf{V}^\top(\mathbf{x} - \bar{\mathbf{x}})$$

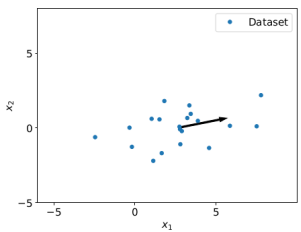
Principal Component Analysis

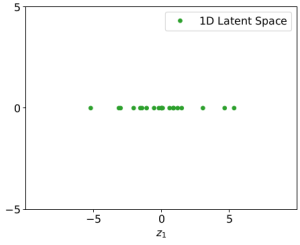


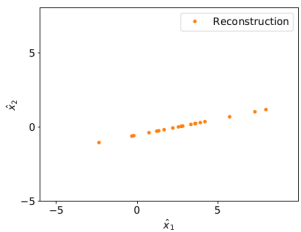
PCA Recipe:

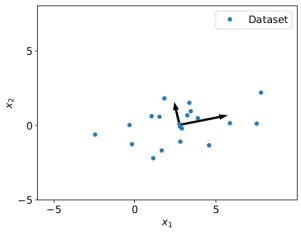
- Given a dataset $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of observations $\mathbf{x}_i \in \mathbb{R}^D$
- Compute the **data mean** $\bar{\mathbf{x}}$ and **scatter matrix** $\mathbf{S} = \sum_{i=1}^{N} (\bar{\mathbf{x}} - \mathbf{x}_i)(\bar{\mathbf{x}} - \mathbf{x}_i)^\top$
- Compute the **eigen decomposition** of $\mathbf{S}$
- Select the Q eigenvectors corresponding to the Q largest eigenvalues for $\mathbf{V}$

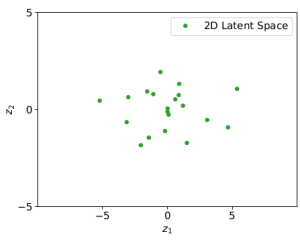
Results in Synthetic 2D Dataset

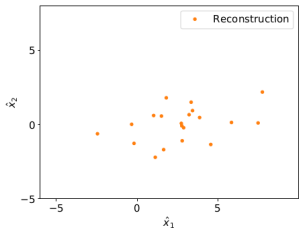




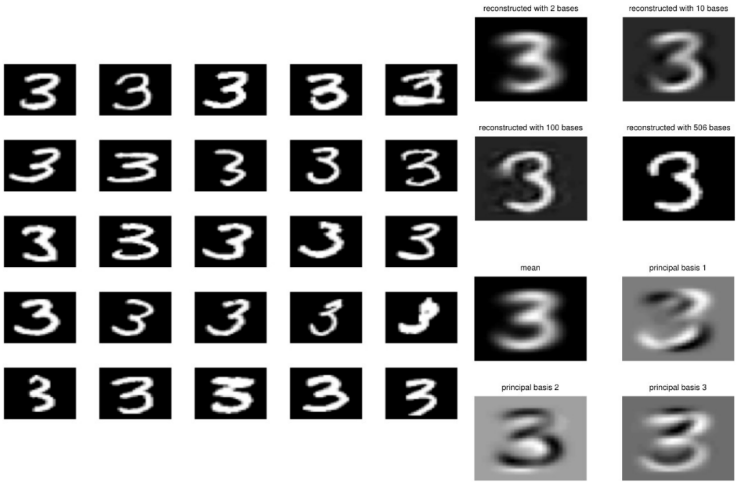








Results on MNIST



Results on Faces

PCA on Face Images:

- PCA on 2429 19x19 grayscales images (CBCL data)
- Yields good reconstructions with only 3 components:

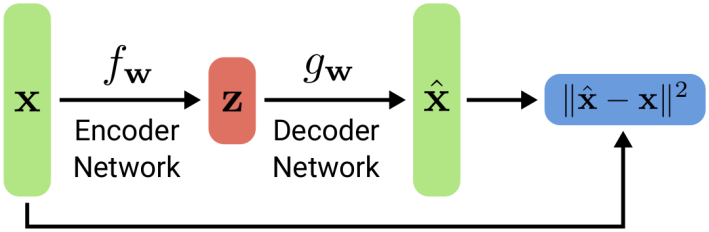
- We can apply a classifier directly on this latent representation
- PCA with 3 components obtains 79 % accuracy on face/non-face discrimination on test data vs. 76.8 % for a Gaussian Mixture Model with 84 states
- Also commonly used for analyzing the latent properties of a dataset

Turk and Pentland: Face recognition using eigenfaces. CVPR, 1991.

Learned Basis (Eigenfaces)

Turk and Pentland: Face recognition using eigenfaces. CVPR, 1991.

Deep Autoencoders



- An autoencoder is a **neural network** whose outputs are its own inputs
- The input $\mathbf{x} \in \mathbb{R}^D$ is compressed to a latent code $\mathbf{z} \in \mathbb{R}^Q$
- The goal is to **minimize the reconstruction error** (as in PCA)

PCA as special case of Autoencoders

Let $f_{\mathbf{w}} : \mathbf{x} \mapsto \mathbf{z}$ denote the encoder and $g_{\mathbf{w}} : \mathbf{z} \mapsto \hat{\mathbf{x}}$ the decoder network.
Let's assume both mappings to be **linear** (without activation function):

$$\mathbf{z} = f_{\mathbf{w}}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{a} \qquad \hat{\mathbf{x}} = g_{\mathbf{w}}(\mathbf{z}) = \mathbf{B}\mathbf{z} + \mathbf{b}$$

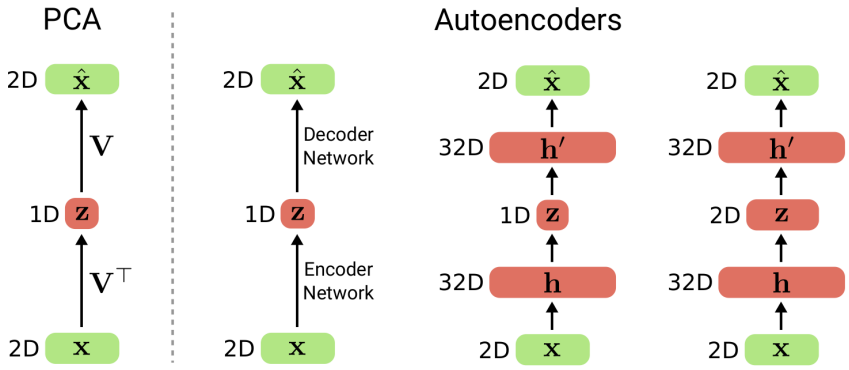
In this case, we have:

$$\hat{\mathbf{x}} = g_{\mathbf{w}}(f_{\mathbf{w}}(\mathbf{x})) = \underbrace{\mathbf{B}\mathbf{A}}_{=\mathbf{C}}\mathbf{x} + \underbrace{\mathbf{a} + \mathbf{b}}_{=\mathbf{c}}$$

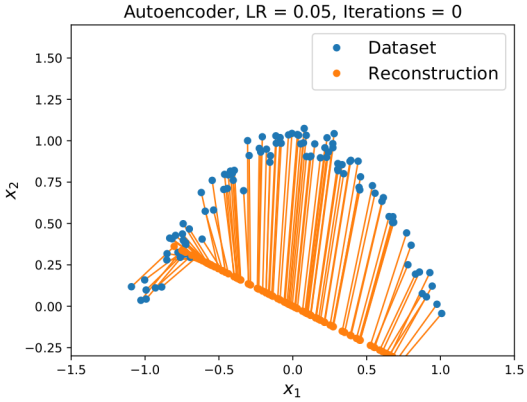
Thus, the optimal solution $\mathbf{w}^*$ is given by **PCA:**

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \| \underbrace{\mathbf{C}\mathbf{x}_i + \mathbf{c}}_{=\hat{\mathbf{x}}_i} - \mathbf{x}_i \|^2$$

Results on Synthetic 2D Dataset

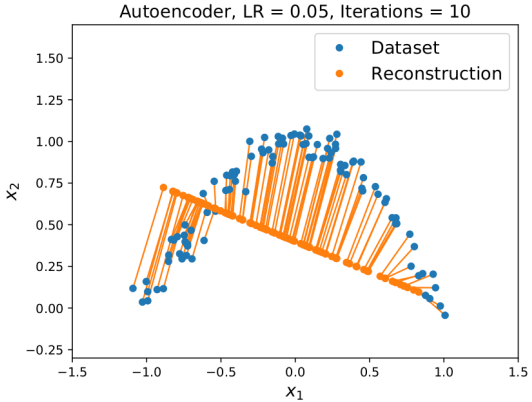


Results on Synthetic 2D Dataset



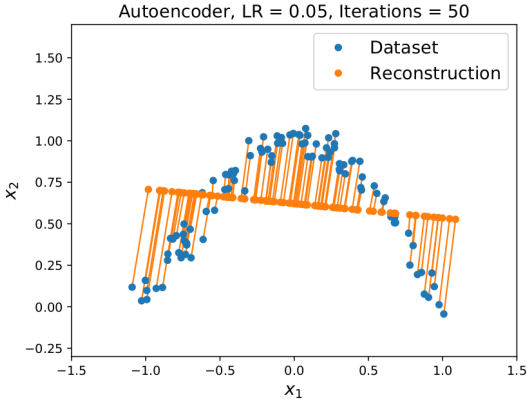
- **Autoencoder reconstruction** with one linear enc/dec layer ($Q = 1$)

Results on Synthetic 2D Dataset



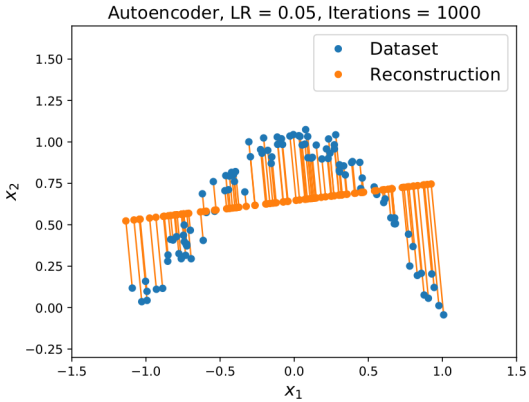
- **Autoencoder reconstruction** with one linear enc/dec layer ($Q = 1$)

Results on Synthetic 2D Dataset



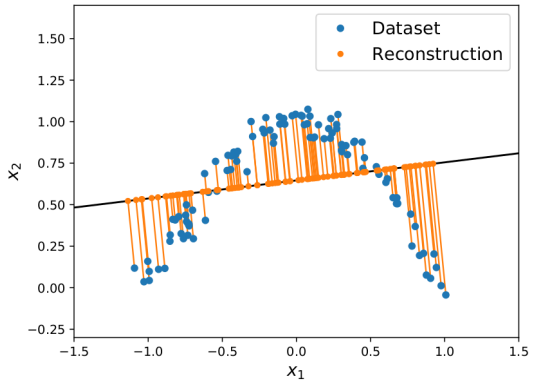
- **Autoencoder reconstruction** with one linear enc/dec layer ($Q = 1$)

Results on Synthetic 2D Dataset



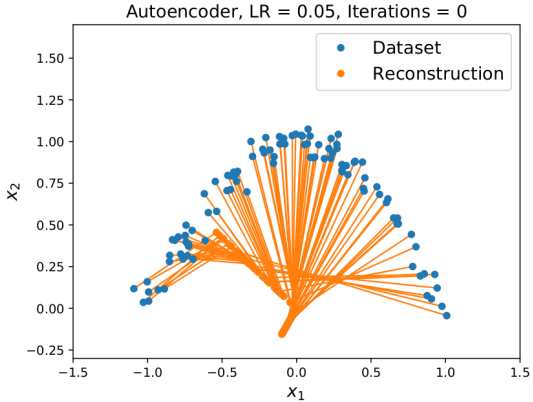
- **Autoencoder reconstruction** with one linear enc/dec layer ($Q = 1$)

Results on Synthetic 2D Dataset



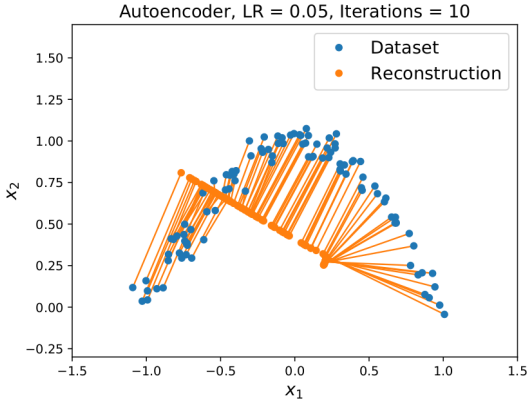
- **PCA reconstructions** on the same dataset with $Q = 1$

Results on Synthetic 2D Dataset



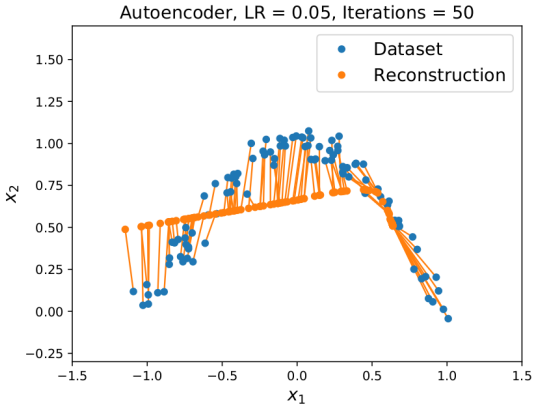
- **Autoencoder reconstruction** with 32D hidden layers ($Q = 1$)

Results on Synthetic 2D Dataset



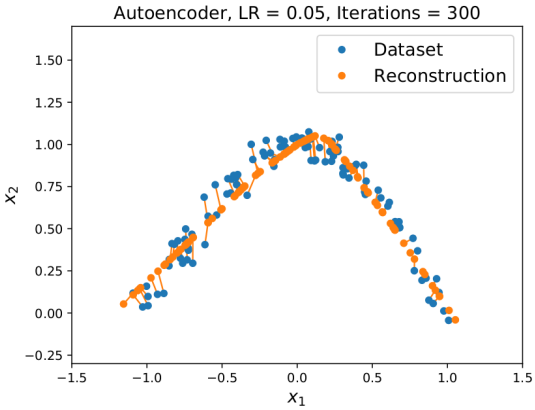
- **Autoencoder reconstruction** with 32D hidden layers ($Q = 1$)

Results on Synthetic 2D Dataset



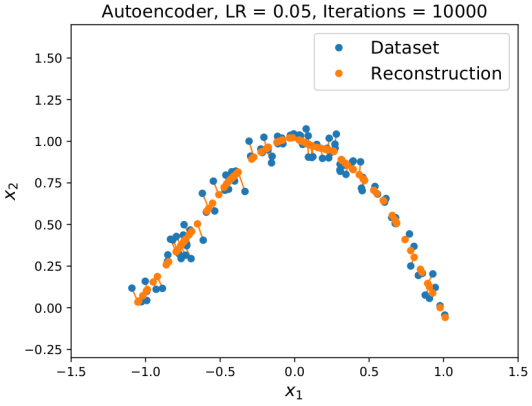
- **Autoencoder reconstruction** with 32D hidden layers ($Q = 1$)

Results on Synthetic 2D Dataset



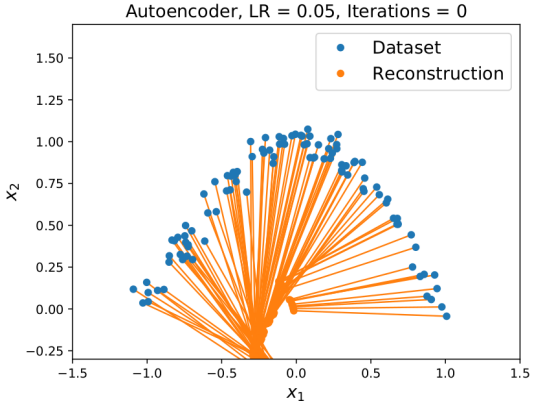
- **Autoencoder reconstruction** with 32D hidden layers ($Q = 1$)

Results on Synthetic 2D Dataset



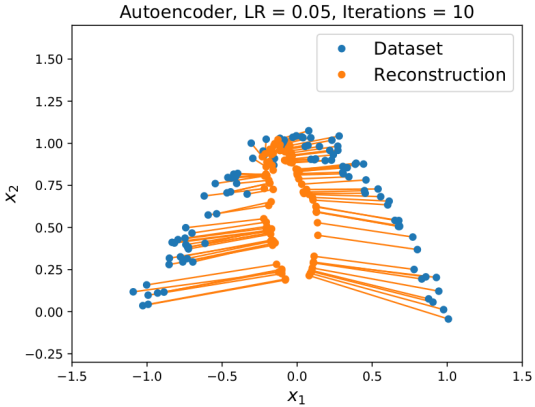
- **Autoencoder reconstruction** with 32D hidden layers ($Q = 1$)

Results on Synthetic 2D Dataset



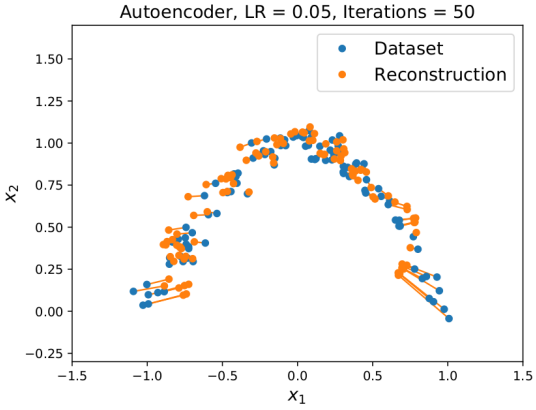
- **Autoencoder reconstruction** with 32D hidden layers ($Q = 2$)

Results on Synthetic 2D Dataset



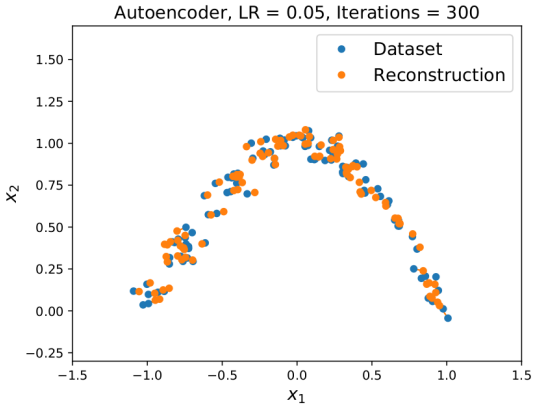
- **Autoencoder reconstruction** with 32D hidden layers ($Q = 2$)

Results on Synthetic 2D Dataset



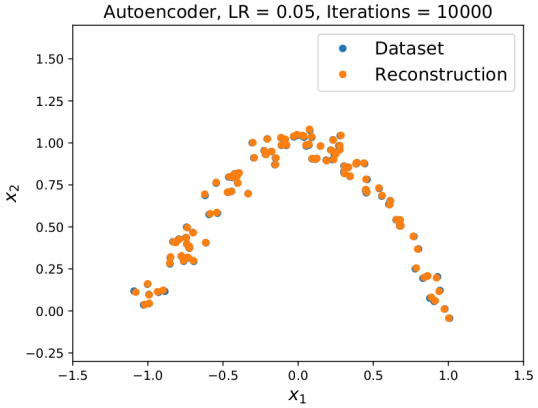
- **Autoencoder reconstruction** with 32D hidden layers ($Q = 2$)

Results on Synthetic 2D Dataset



- **Autoencoder reconstruction** with 32D hidden layers ($Q = 2$)

Results on Synthetic 2D Dataset



- **Autoencoder reconstruction** with 32D hidden layers ($Q = 2$)

Comparing Reconstructions

- Deep autoencoders learn **non-linear** latent embeddings
- They often have **smaller reconstruction errors** compared to PCA with same Q
- In contrast, PCA always learns the best linear mapping

Denoising Autoencoders

- **Denoising Autoencoders** take noisy inputs and predict the original noise-free data
- Higher level representations are relatively stable and robust to input corruption
- Encourages the model to **generalize better** and capture useful structure
- Similar to data augmentation (except that the "label" is the noise-free input)
- `https://blog.keras.io/building-autoencoders-in-keras.html`

Variational Autoencoders

So far, we have discussed deterministic latent variables. We will now take a **probabilistic perspective** on **latent variable models** with autoencoding properties. Consider the following **Bayesian model** of the data $\mathbf{x} \in \mathbb{R}^D$:

$$p(\mathbf{x}) = \int_{\mathbf{z}} \underbrace{p(\mathbf{z})p(\mathbf{x}|\mathbf{z})}_{\text{Generative Process}} \, d\mathbf{z} = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} p(\mathbf{x}|\mathbf{z})$$

- $p(\mathbf{z})$ is the **prior** over the **latent variable** $\mathbf{z} \in \mathbb{R}^Q$
- $p(\mathbf{x}|\mathbf{z})$ is the **likelihood**
- $p(\mathbf{x})$ is the **marginal** of the joint distribution $p(\mathbf{x}, \mathbf{z})$

Variational Autoencoders

Assumptions:

- The **prior** model $p(\mathbf{z})$ can be computed and sampled efficiently

Variational Autoencoders

Assumptions:

- The **prior** model $p(\mathbf{z})$ can be computed and sampled efficiently
- The **likelihood** model $p(\mathbf{x}|\mathbf{z})$ is computable

Variational Autoencoders

Assumptions:

- The **prior** model $p(\mathbf{z})$ can be computed and sampled efficiently
- The **likelihood** model $p(\mathbf{x}|\mathbf{z})$ is computable
- We can **sample** from $p(\mathbf{z})$ and we can compute the proability of $p(\mathbf{z})$ and $p(\mathbf{x}|\mathbf{z})$ for any given $\mathbf{x}$ and $\mathbf{z}$

Variational Autoencoders

Assumptions:

- The **prior** model $p(\mathbf{z})$ can be computed and sampled efficiently
- The **likelihood** model $p(\mathbf{x}|\mathbf{z})$ is computable
- We can **sample** from $p(\mathbf{z})$ and we can compute the proability of $p(\mathbf{z})$ and $p(\mathbf{x}|\mathbf{z})$ for any given $\mathbf{x}$ and $\mathbf{z}$
- We will choose **simple parametric distributions** to achieve this

Variational Autoencoders

Let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}_{i=1}^N$ with $\mathbf{x}_i \in \mathbb{R}^D$ be a dataset and $\mathbf{w}$ the model parameters.
The **Variational Autoencoder** minimizes this bound to the **negative log likelihood:**

$$
\begin{aligned}
\mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmin}} \; \mathbb{E}_{\mathbf{x} \sim p_{data}} \left[-\log p_{\mathbf{w}}(\mathbf{x}) \right] \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \left[-\log p_{\mathbf{w}}(\mathbf{x}_i) \right] \\
&\approx \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{KL(q_{\mathbf{w}}(\mathbf{z}|\mathbf{x}_i), p(\mathbf{z}))}_{\text{Approx. Posterior = Prior}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\mathbf{w}}(\mathbf{z}|\mathbf{x}_i)} \left[-\log p_{\mathbf{w}}(\mathbf{x}_i|\mathbf{z}) \right]}_{\text{Reconstruction Term}}
\end{aligned}
$$

- In a VAE, $q_{\mathbf{w}}(\mathbf{z}|\mathbf{x})$ is a **multivariate Gaussian** parameterized by a neural network
- It thus makes a **variational approximation** $q_{\mathbf{w}}(\mathbf{z}|\mathbf{x})$ to the true posterior $p(\mathbf{z}|\mathbf{x})$

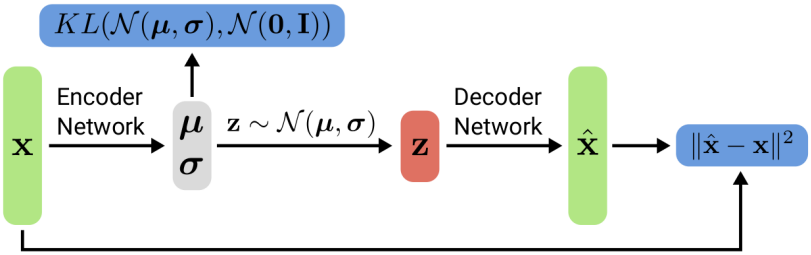
Variational Autoencoders

Let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}_{i=1}^N$ with $\mathbf{x}_i \in \mathbb{R}^D$ be a dataset and $\mathbf{w}$ the model parameters.
The **Variational Autoencoder** minimizes this bound to the **negative log likelihood:**

$$
\begin{aligned}
\mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmin}} \; \mathbb{E}_{\mathbf{x} \sim p_{data}} \left[-\log p_{\mathbf{w}}(\mathbf{x}) \right] \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \left[-\log p_{\mathbf{w}}(\mathbf{x}_i) \right] \\
&\approx \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{KL(q_{\mathbf{w}}(\mathbf{z}|\mathbf{x}_i), p(\mathbf{z}))}_{\text{Approx. Posterior = Prior}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\mathbf{w}}(\mathbf{z}|\mathbf{x}_i)} \left[-\log p_{\mathbf{w}}(\mathbf{x}_i|\mathbf{z}) \right]}_{\text{Reconstruction Term}}
\end{aligned}
$$

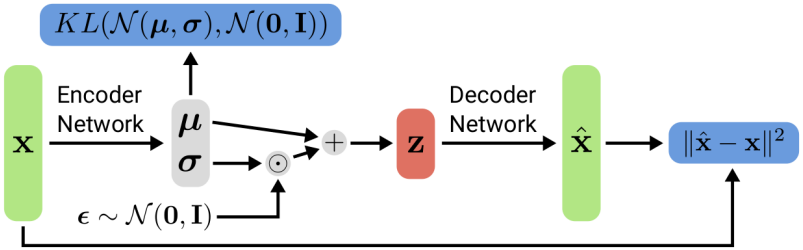
- The **likelihood model** $p_{\mathbf{w}}(\mathbf{x}_i|\mathbf{z})$ is also parameterized by a neural network
- For binary observations **Bernoulli**, for real observations **Gaussian** or **Laplacian**

Learning a VAE



- **Vanilla VAE** formulation which is intractable due to sampling inside the network
- Remark: We assume a Gaussion likelihood $p_{\mathbf{w}}(\mathbf{x}|\mathbf{z})$ in this example

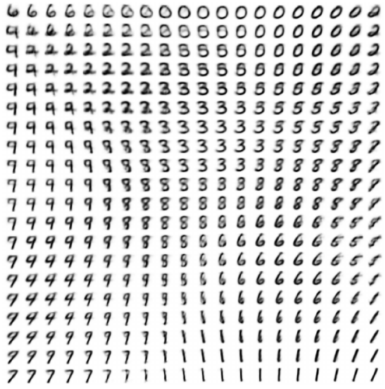
Learning a VAE



- **Reparameterized version** which is tractable as sampling has been moved to input
- This trick works for Gaussians and some other simple distributions (cf., Kingma)

Example: Natural Image Manifolds

(a) Learned Frey Face manifold

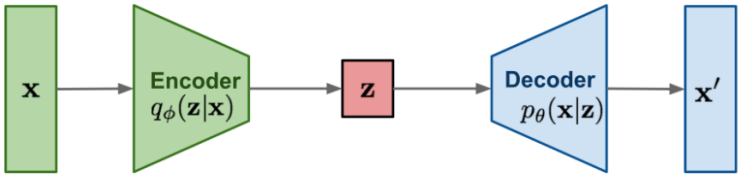


(b) Learned MNIST manifold

Diederik P. Kingma, Max Welling: Auto-Encoding Variational Bayes. ICLR, 2014.

Variational Autoencoders (Summary)

- Probabilistic latent variable models
- Successful in learning useful low-dimensional representations
- Trained via a bound (ELBO) on the marginal likelihood
- Sampling from $p(\mathbf{x})$ is straightforward
- Approximate evaluation of $p(\mathbf{x})$ is possible



Kingma & Welling Foundations and Trends in Machine Learning: Vol. 12 (2019): No. 4, pp 307-392

Session 2A (2nd hour)

Normalizing flows

What are normalizing flows?:

- A probabilistic latent variable model built on invertible transformations
- Efficient to sample from $p(\mathbf{x})$
- Efficient to evaluate $p(\mathbf{x})$
- Highly expressive
- Useful latent representation
- Straightforward to train

Example: Glow model `https://openai.com/blog/glow/`

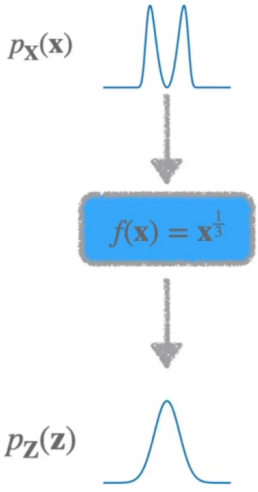
Normalizing flows

Basic idea: **change of variables formula**

$$p_x(\mathbf{x}) = p_z(\underbrace{f(\mathbf{x})}_{\text{invertible}}) \underbrace{|\det Df(\mathbf{x})|}_{\text{volume correction}}$$

where

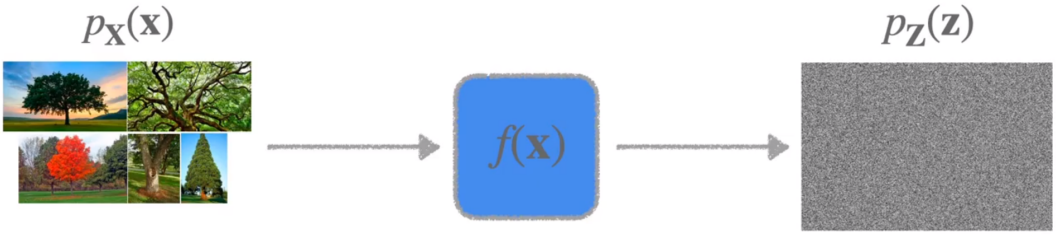
- $\mathbf{z} = f(\mathbf{x})$ with an invertible, differentiable $f(\mathbf{x})$
- $Df(\mathbf{x})$ is the Jacobian of $f(\mathbf{x})$
  (ensures normalization)



Normalizing flows

- Given $p_x(\mathbf{x})$, fix $p_z(\mathbf{z})$ and find an invertible, differentiable $f(\mathbf{x})$ such that

$$p_x(\mathbf{x}) = p_z(f(\mathbf{x}))|\det Df(\mathbf{x})|$$



Two pieces:

- Base measure $p_z(\mathbf{z})$, typically chosen as $\mathcal{N}(\mathbf{0}, \mathbf{I})$
- Flow $f(\mathbf{x})$ must be invertible and differentiable

Normalizing flows

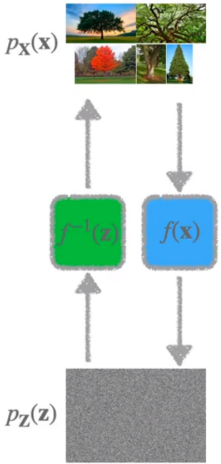
Training via maximum (log-)likelihood

$$\max_{\theta} \sum_{i=1}^{N} \log p_z(f(\mathbf{x}_i;\theta)) + \log|\det Df(\mathbf{x}_i;\theta)|$$

where θ are the parameters of the flow $f(\mathbf{x}_i;\theta)$

If learned properly:

- Exact density evaluation: $p_x(\mathbf{x}) = p_z(f(\mathbf{x}))|\det Df(\mathbf{x})|$
- Sampling
  - Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - Compute $\mathbf{x} = f^{-1}(\mathbf{z})$



Normalizing flows

- **Main challenge:** Design an invertible and differentiable flow that is also

Normalizing flows

- **Main challenge:** Design an invertible and differentiable flow that is also
  - Sufficiently expressive

Normalizing flows

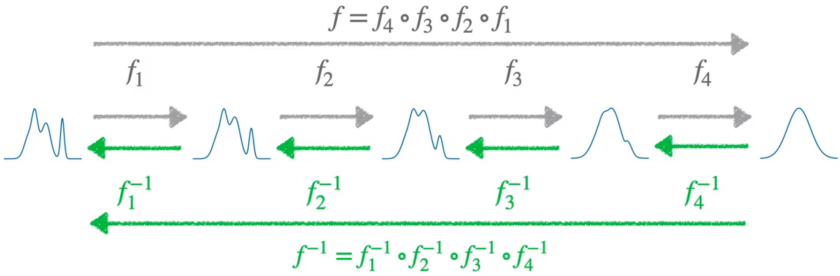
- **Main challenge:** Design an invertible and differentiable flow that is also
  - Sufficiently expressive
  - Efficient computation of inverse and Jacobian determinant

Normalizing flows

- **Main challenge:** Design an invertible and differentiable flow that is also
  - Sufficiently expressive
  - Efficient computation of inverse and Jacobian determinant
- Key idea: Compositions of simpler flows

Normalizing flows

- **Main challenge:** Design an invertible and differentiable flow that is also
  - Sufficiently expressive
  - Efficient computation of inverse and Jacobian determinant
- Key idea: Compositions of simpler flows



Normalizing flows

- The determinant of the Jacobian factorizes

$$\det Df = \det \prod_{k=1}^{K} Df_k = \prod_{k=1}^{K} \det Df_k$$