Introduction to Deep Generative Models

Jes Frellsen

Department of Applied Mathematics and Computer Science

Technical University of Denmark

ProbAI, 2024

https://github.com/frellsen/ProbAI-2024

Slides co-developed with

Jakub Tomczak
TU/e

Pierre-Alexandre Mattei
INRIA

Menu of the day

- Brief introduction to generative models
- Deep latent variable models
  - Exercise 1
- Normalising Flows
  - Exercise 2

Introduction to generative models

Part I

Generative modelling*

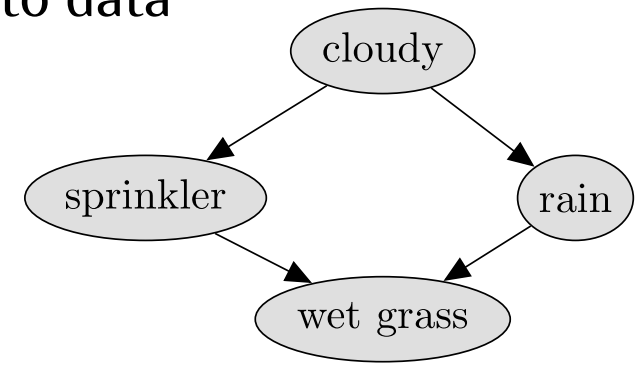
- We **observe** some data $x_1, \ldots, x_N \in \mathcal{X}$, e.g., the CelebA dataset
- **Goal:** learn an approximation of the data distribution
- **Outcome:** be able to generate new and plausible images!

*) Or unsupervised learning
Images from: https://www.tensorflow.org/datasets/catalog/celeb_a_hq

Why probabilistic generative models?

A forward generative model "describes the process assumed to rise to data"[1]

- In unsupervised learning, we model the joint density $p(\mathbf{x})$.
- Typically, we assume some factorisation of $p(\mathbf{x})$.



"Hard part" "Easy part"

Once we have learned the joint, we can answer queries

- Sampling $\mathbf{x} \sim p(\mathbf{x})$
- Density $p(\mathbf{x})$
- Marginalization $p(\mathbf{x}_i) = \int p(\mathbf{x}_i, \mathbf{x}_{\neg i})\, \mathrm{d}\mathbf{x}_{\neg i}$
- Conditioning $p(\mathbf{x}_i|\mathbf{x}_{\neg i}) = \frac{p(\mathbf{x}_i, \mathbf{x}_{\neg i})}{p(\mathbf{x}_{\neg i})}$ E.g., classification or missing data imputation
- Expectation $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[f(\mathbf{x})]$
- Maximization $\arg\min_{x} p(\mathbf{x})$ E.g., moments

Why probabilistic?

- Randomness required for "generation"
- Probabilities are consistent for reasoning under uncertainty

[1]MacKay DJ. Information theory, inference and learning algorithms. Cambridge university press, 2003.
Inspired by slides from Robert Peharz: https://github.com/robert-peharz/robert-peharz.github.io/blob/master/doc/GeMSS23ProbabilisticCircuits.pdf

Why on earth would we want to generate new and plausible images?

Because it is cool!

Why on earth would we want to generate new and plausible images?

Because it can impact society unexpectedly quickly!

← THE BEST INVENTIONS OF 2022 TIME

Artificial Imagination

OpenAI DALL-E 2

The Economist

Science & technology | Generative AI

Large, creative AI models will transform lives and labour markets

The New York Times

OPINION

GUEST ESSAY

Noam Chomsky: The False Promise of ChatGPT

March 8, 2023

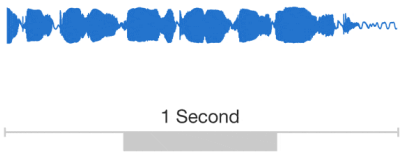
It's not just pretty images and texts!

WaveNet: A Generative Model for Raw Audio

Aäron van den Oord **Sander Dieleman** **Heiga Zen**†

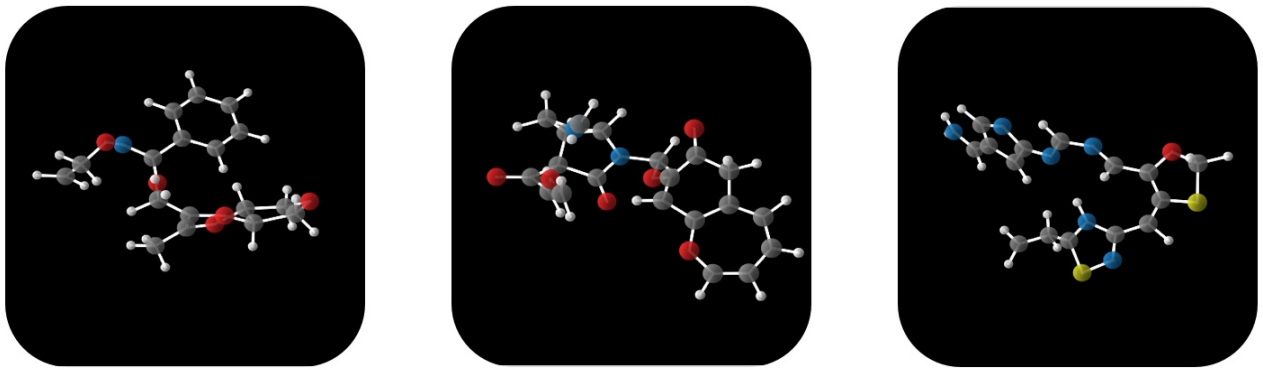
Karen Simonyan **Oriol Vinyals** **Alex Graves**

Nal Kalchbrenner **Andrew Senior** **Koray Kavukcuoglu**



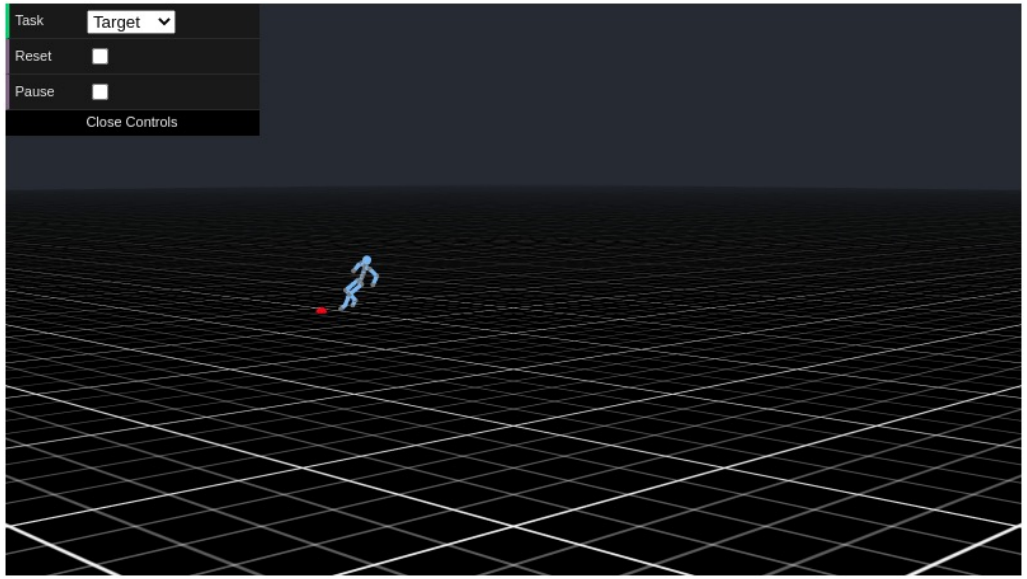
Equivariant Diffusion for Molecule Generation in 3D

Emiel Hoogeboom [*1] **Victor Garcia Satorras** [*1] **Clément Vignac** [*2] **Max Welling** [1]



Character Controllers Using Motion VAEs

HUNG YU LING, University of British Columbia, Canada
FABIO ZINNO, Electronic Arts Vancouver, Canada
GEORGE CHENG, Electronic Arts Vancouver, Canada
MICHIEL VAN DE PANNE, University of British Columbia, Canada

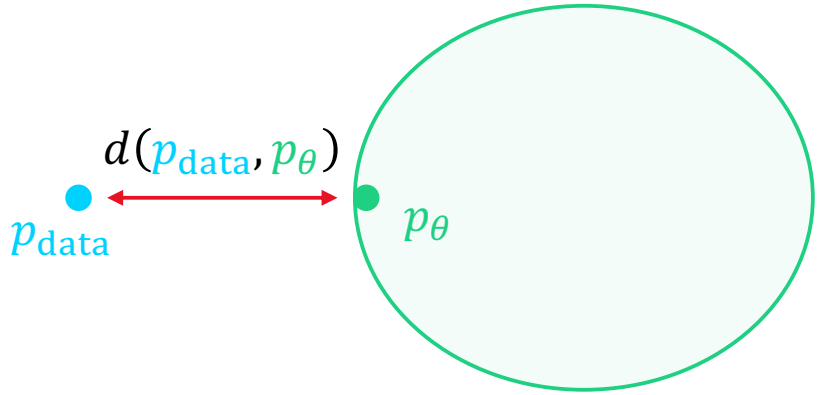


Towards maximum likelihood

- Consider some model family $\{p_\theta\}_{\theta \in \Theta}$
- We want to **find a good** $\hat{\theta} \in \Theta$ such that $p_{\hat{\theta}} \approx p_{\text{data}}$
- Given a distance d between models, we want to find

$$\hat{\theta} \in \arg\min_{\theta \in \Theta} d(p_{\text{data}}, p_\theta)$$

$\mathbf{x}_i \sim p_{\text{data}}$



$\hat{\theta} \in \Theta$
Model family

Illustration based on: Ermon S, Song Y. CS236 , lecture 11.

The Kullback-Leibler divergence

- A commonly used measure of how similar two distributions p and q are

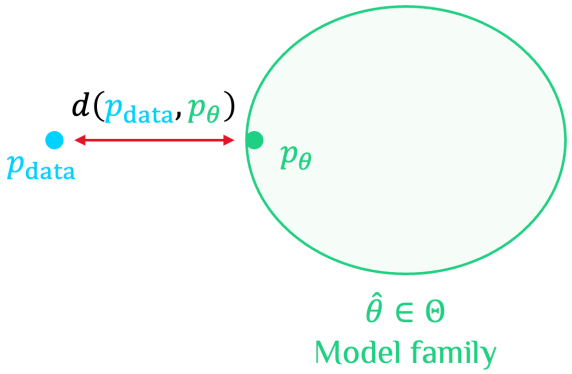
$$\mathrm{KL}[p \parallel q] = \int p(x) \log \frac{p(x)}{q(x)} \mathrm{d}x = \mathbb{E}_{x \sim p(x)} \left[\log \frac{p(x)}{q(x)} \right]$$

- Measure the average difference of the log-densities
- Asymmetrical, but with some nice "distance-like" properties:
  1. $\mathrm{KL}[p \parallel q] \geq 0$
  2. $\mathrm{KL}[p \parallel q] = 0 \Leftrightarrow p = q$

Finding the best model according to KL

We want to **find a good** $\hat{\theta} \in \Theta$ such that $p_{\hat{\theta}} \approx p_{\text{data}}$

$$\hat{\theta} \in \arg\min_{\theta \in \Theta} \mathrm{KL}[p_{\text{data}} \,\|\, p_\theta]$$



where

- $\mathrm{KL}[p_{\text{data}} \,\|\, p_\theta] = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log p_{\text{data}}(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log p_\theta(\mathbf{x})]$ (Does not depend on θ)
- $\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log p_\theta(\mathbf{x})] \approx \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i)$ for samples $\mathbf{x}_1, \ldots, \mathbf{x}_N \sim p_{\text{data}}(\mathbf{x})$

So we want to find

Log-likelihood function

$$\hat{\theta} \in \arg\max_{\theta \in \Theta} \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i) = \arg\max_{\theta \in \Theta} \ell(\theta)$$

Maximum likelihood estimation

- Given
  - a model family $\{p_\theta\}_{\theta \in \Theta}$
  - observations $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$,
- Our objective for learning θ is **log-likelihood function**

$$\ell(\theta) = \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i)$$

- We would like to find the **ML estimate**: $\hat{\theta} = \arg\max_{\theta \in \Theta} \ell(\theta)$.

Modern deep generative model

- DLVMs / VAEs: $p_\theta(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{x})\,\mathrm{d}\mathbf{x}$
- Autoregressive models: $p_\theta(\mathbf{x}) = \prod_{d=1}^{D} p_\theta(x_d|\mathbf{x}_{<d})$
- Normalising Flows: $p_{\mathrm{x}}(\mathbf{x}) = p_{\mathrm{u}}(\mathbf{u})|\det J_T(\mathbf{u})|^{-1}$
- Energy based models: $p_\theta(\mathbf{x}) = \frac{e^{-E_\theta(\mathbf{x})}}{Z_\theta}$
- Diffusion models

This afternoon

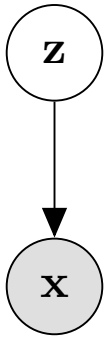
Deep latent variable models

Part II

Latent variable models

Generative process

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$



Manifold hypothesis: $\mathbf{z} \in \mathbb{R}^M$ and $\mathbf{x} \in \mathbb{R}^D$ where $M < D$

Latent factors, e.g.:

- Colour, angle, background, etc.

We only observe $\mathbf{x}$, and the marginal density of $\mathbf{x}$ is

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

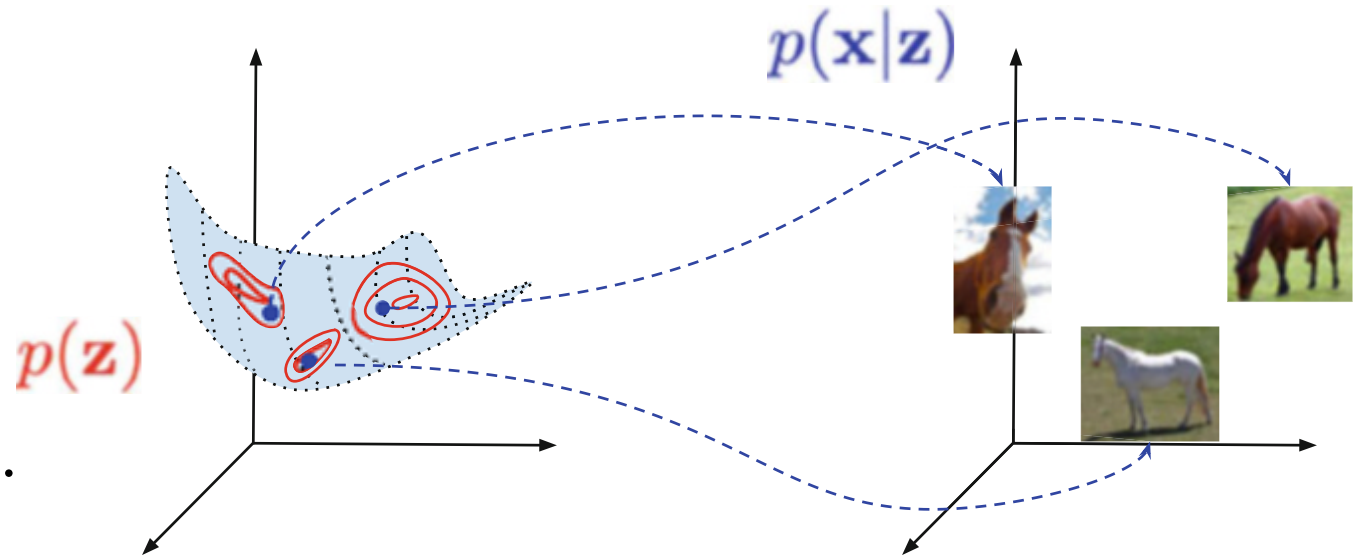


Figure from: Tomczak JM. Deep generative modeling. Springer, 2022.

Deep latent variable models (DLVMs)

- Probabilistic generative models:
  - + Well-founded framework for model building, inference and prediction
  - - Limited complexity: Mainly conjugate and linear models
  - - Learning is computational expensive
- Deep neural networks:
  - + Rich non-linear models
  - + Scalable learning using stochastic gradient decent
  - - Only give point estimates
  - - Typically, discriminative and non-probabilistic

Deep latent variable models combine the approximation abilities of deep neural networks and the statistical foundations of generative models.

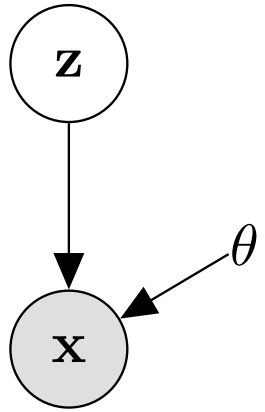
Based on tutorial by Shakir Mohamed and Danilo Rezende: https://auai.org/uai2017/media/tutorials/shakir.pdf

Deep latent variable models (DLVMs)

Assume $\mathbf{x}, \mathbf{z}$ are random variables driven by the model:

$$\mathbf{z} \sim p(\mathbf{z}) \qquad \text{(prior)}$$

$$\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z}) = \Phi(x|f_\theta(z)) \qquad \text{(observation model)}$$



where

- $\mathbf{z} \in \mathcal{Z}^M$ is the continuous latent variable,
- $\mathbf{x} \in \mathcal{X}$ is the observed variable,
- the function $f_\theta : \mathcal{Z}^M \to H$ (deep) neural network called the decoder,
- $\{\Phi(\cdot\,|\eta)\}_{\eta \in H}$ is a parametric family called the output density.

You can think of a DLVM as non-linear PPCA

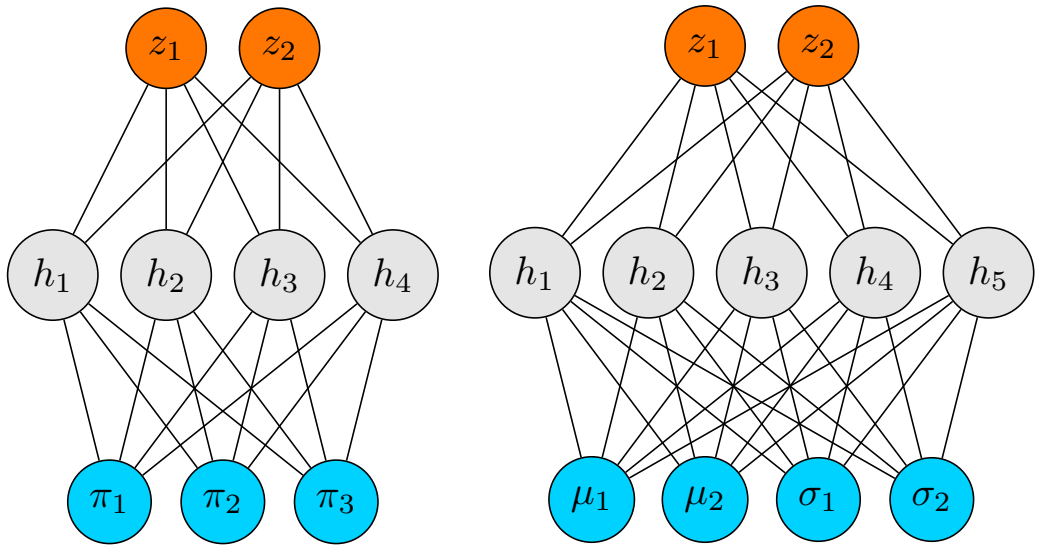
You can think of a DLVM as a MM, but with continuous $\mathbf{z}$

The role of the decoder and output density

$$p_\theta(\mathbf{x}|\mathbf{z}) = \text{Categorical}(\mathbf{x}|\boldsymbol{\pi} = f_\theta(\mathbf{z})) \quad \text{for } \mathbf{x} \in \{0, \dots, 255\}^D$$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma} = f_\theta(\mathbf{z})) \quad \text{for } \mathbf{x} \in \mathbb{R}^D$$

- Typically, we just have one **decoder with multiple output heads**.
- The **output of the decoder** has to lie in the **parameter space**
  - For the **categorical**, we apply a softmax at the output.
  - For the **Gaussian**, we typically use a **diaoginal covariance** $\boldsymbol{\Sigma} = \text{diag}\left(\sigma_\theta^2(\mathbf{z})\right)$ or $\boldsymbol{\Sigma} = \text{diag}(\exp(f_\theta(\mathbf{z})))$



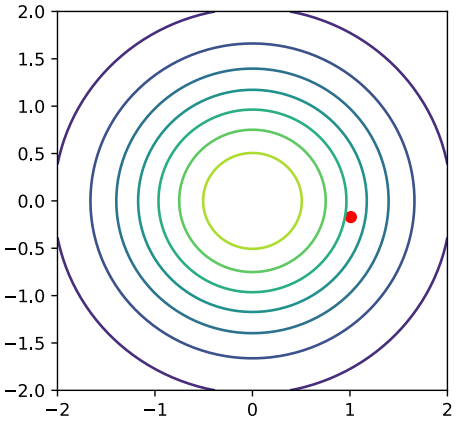
Illustrative example of a DLVM

Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ binary MNIST

Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

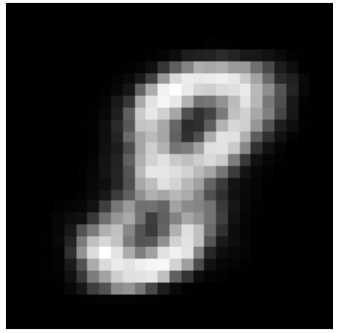
$$\mathbf{z} = (1.0097, -0.1711)$$



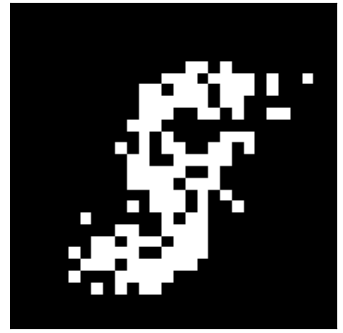
Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0,1\}^{28\times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

$f(z)$



$x^{j,k} \sim \text{Bern}(f^{j,k}(z))$



Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V}\tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

"You haven't mentioned an encoder yet?"

"In fact, you haven't even said variational autoencoder"

Spend 3 minutes **discussing with your neighbour**:

- Q: How has the presentation, so far, differed from the "usual" presentation of VAEs?
  - A: I have only focused on the (generative) model description
- Q: Why haven't I mentioned an encoder?
  - A: That is because the encoder is "only" used for learning the model
  - In fact, you can learn in DLVM without an encoder[1,2]
- Q: What insights does it give not mentioning an encoder yet?
  - A: It decouples model and inference assumptions

[1]Schuster V, Krogh A. The Deep Generative Decode. arXiv:2110.06672, 2021.
[2]Hoffman MD. Learning Deep Latent Gaussian Models with Markov Chain Monte Carlo. ICML, 2018

Learning objective

- We want to **learn the parameters θ of the decoder**
- Given observations $\{x_1, \dots, x_N\}$, a natural objective is **log-likelihood function**

$$\ell(\theta) = \sum_{n=1}^{N} \log p(\mathbf{x}_n) \quad \text{where} \quad p(\mathbf{x}_n) = \int p_\theta(\mathbf{x}_n|\mathbf{z}) p(\mathbf{z}) \, \mathrm{d}\mathbf{z}$$

- We would like to find the **ML estimate**: $\hat{\theta} = \arg\max_\theta \ell(\theta)$.
- However, for non-linear decoders (generally)
  - $p(\mathbf{z}|\mathbf{x}_n)$ is **intractable** rendering **expectation–maximization (EM) intractable**
  - $p(\mathbf{x}_n)$ is **intractable** rendering ***direct*** **MLE intractable**
  - The **MC estimate** $p(\mathbf{x}_n) = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[p_\theta(\mathbf{x}_n|\mathbf{z})] \approx \frac{1}{K}\sum_k p(\mathbf{x}_n|\mathbf{z_k})$ has high variance
    - **Curse of dimensionality**: number of samples grows exponentially with latent dim. M

Amortized variational inference (VI)

You learned the details yesterday

- VI approximatively maximises the log-likelihood, $\ell(\theta)$, by maximising the ELBO

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z})}\right]$$

Otherwise use this form (you'll try later)

$$= \overbrace{\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}^{\text{reconstruction term}} - \overbrace{\mathrm{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\right]}^{\text{regularisation term}} \leq \ell(\theta)$$

Closed form for Gaussians

- We consider an **amortized variational posterior** (i.e. conditional)

$$q_\phi(\mathbf{z}|\mathbf{x}) = \Psi\left(\mathbf{z}\middle|g_\phi(\mathbf{x})\right)$$

  where
  - $\{\Psi(\cdot\,|\kappa)\}_{\kappa \in K}$ is the **variational family** over $\mathcal{Z}^M$ (for $\mathbb{R}^M$ usually Gaussians)
  - $g_\phi : \mathcal{X} \to K$ is a neural net called the **encoder** or **inference network**
  - g_ϕ transforms a datapoint $\mathbf{x}$ into parameters of the variational distribution
- The variational distribution is a **tractable approximation** of the posterior $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$
- Low variance gradients, $\nabla_{\theta,\phi}\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[f_{\theta,\phi}(\mathbf{z})]$, calculated using the **reparameterization trick**

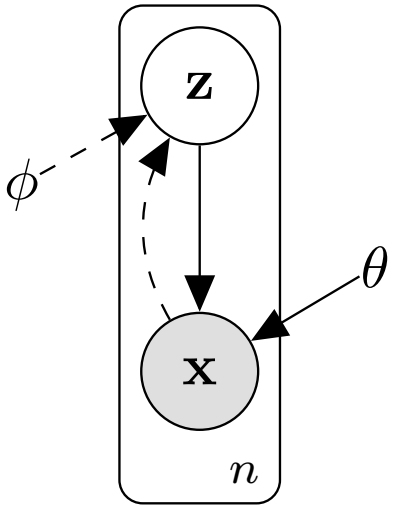
Reparameterization trick

- Also known as the **pathwise gradient estimator**, split into
  1. Stochastic parameter free part
  2. A deterministic parametric transformation
- **Many distributions** can be reparameterized
  - Including: Gaussian, Beta, Gamma, StudentT, and Dirichlet
  - Implemented as `rsample()` in `torch.distributions`
  - **Does not work for discrete variable** (only permutations possible)
    - Solved by relaxed discrete distribution, e.g. the Gumbel-softmax trick[1,2]
- Other **Monte Carlo gradient estimation** exist
  - Including score function estimators (REINFORCE) and measure-valued gradient estimators, see Mohamed et al. (2020)[3] for more details

[1]Jang E, Gu S, Poole B. Categorical Reparameterization with Gumbel-Softmax. ICLR 2017.
[2]Maddison C, Mnih A, Teh Y. The concrete distribution: A continuous relaxation of discrete random variables. ICLR, 2017.
[3]Mohamed S, Rosca M, Figurnov M, Mnih A. Monte Carlo Gradient Estimation in Machine Learning. JMLR, 2020.

The variational autoencoder (VAEs)[1,2]

- A DLVM combined with AVI is called a **variational autoencoder**
  - The **prior** or **marginal distribution** $p(\mathbf{z})$
  - The **likelihood, observation model**, or **stochastic decoder** $p_\theta(\mathbf{x}|\mathbf{z})$
  - The **decoder** $f_\theta : \mathcal{Z}^M \to H$
  - The **variational distribution/posterior** or **stochastic encoder** $q_\phi(\mathbf{z}|\mathbf{x})$
  - The **encoder** $g_\phi : \mathcal{X} \to K$
  - The **ELBO** objective
$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}\big[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\big]$$
  - Optimisation, $\max_{\theta,\phi} \mathcal{L}(\theta, \phi)$, using stochastic gradient descent, where gradient are found using the **reparameterization trick**



[1]Kingma DP, Welling M. Auto-encoding variational Bayes. ICLR, 2014.
[2]Rezende DJ, Mohamed S, Wierstra D. Stochastic backpropagation and approximate inference in deep generative models. ICML 2014.

Why is it called a VAE?

$$-\frac{1}{2}\left\|\mathrm{x}-f_\theta\left(g_\phi(\mathbf{x})+\boldsymbol{\epsilon}\right)\right\|^2-\log Z$$

- Let's revisit the ELBO for Gaussian output

$$\begin{aligned}\mathcal{L}(\theta,\phi) &= \overbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}^{\text{reconstruction term}} - \overbrace{\mathrm{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\right]}^{\text{regularisation term}} \\ &= \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0},\mathbf{I}_M)}\left[\log \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}=f_\theta\left(g_\phi(\mathbf{x})+\boldsymbol{\epsilon}\right), I_D\right)\right] - \mathrm{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel \mathcal{N}(\mathbf{z})\right]\end{aligned}$$

  - Here, we only encode the mean of a Gaussian variational distribution
- And the loss of an autoencoder (AE), i.e.,

$$L(\theta,\phi) = \left\|\mathbf{x}-\mathrm{Dec}_\theta\left(\mathrm{Enc}_\phi(\mathbf{x})\right)\right\|^2$$

- The VAE can be viewed as a AE with
  - a noisy encoder
  - a Gaussian regularisation of the latent space

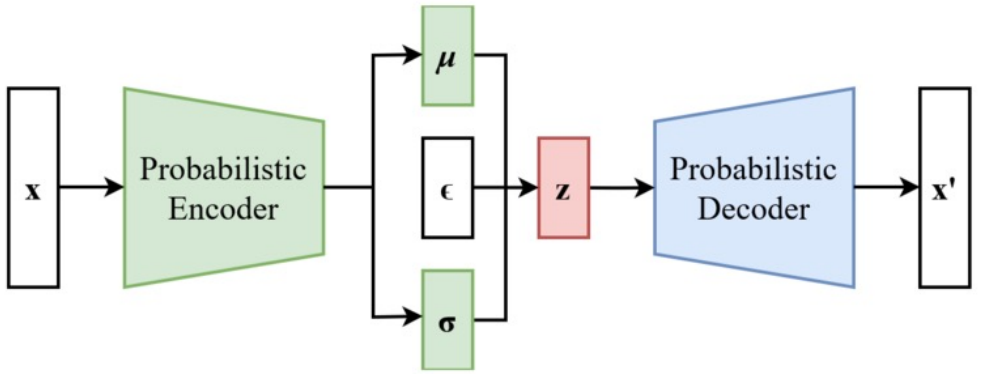


Figure from: https://commons.wikimedia.org/wiki/File:Reparameterized_Variational_Autoencoder.png

Latent representation

$p(\mathbf{z}|\mathbf{x}) \approx q(\mathbf{z}|\mathbf{x})$ gives a **stochastic latent representation** of data, e.g., for MNIST:

Aggregated posterior

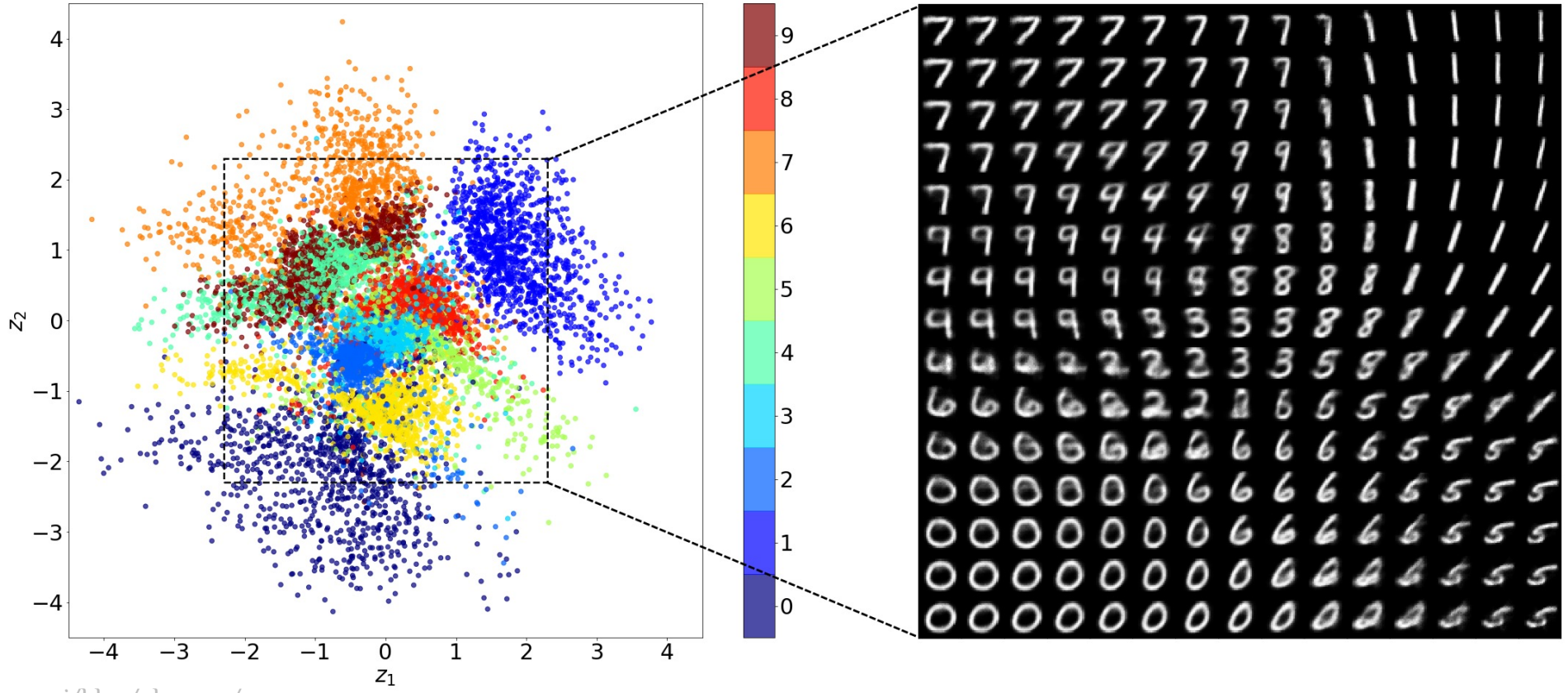
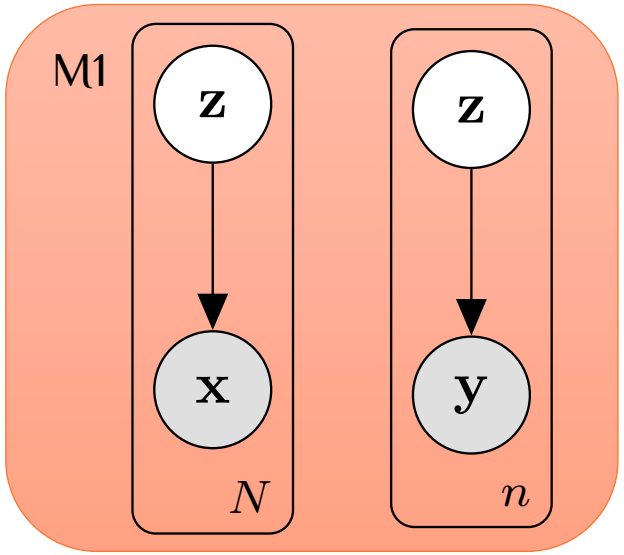
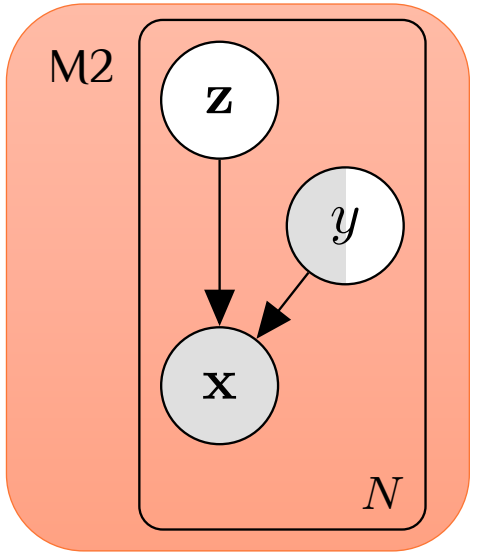


Figure from: https://argmax.ai/blog/vhp-vae/

What can we use VAEs for?





- As data **generators**, e.g., for
  - Generating pretty images 😀
  - Data augmentation
- As a **density model**, e.g., for
  - Missing data imputation[1,2,3]
  - Information acquisition /active sequential variable selection[3]
- For dimensionality reduction or **representation learning**, e.g., for
  - Data visualisation
  - Semi-supervised learning[4]

[1]Nazabal A, Olmos PM, Ghahramani Z, Valera I. Handling incomplete heterogeneous data using VAEs. arXiv:1807.03653, 2018.
[2]Mattei P-A, Frellsen J. MIWAE: Deep Generative Modelling and Imputation of Incomplete Data Sets. ICML, 2019.
[3]Ma C, Tschiatschek S, Palla K, Hernandez Lobato JM, Nowozin S, Zhang C. EDDI: Efficient Dynamic Discovery of High-Value Information with Partial VAE. ICLR, 2019.
[4]Kingma DP, Mohamed S, Rezende DJ, Welling M. Semi-supervised Learning with Deep Generative Models. NeurIPS, 2014.

Are VAEs Bayesian?

- NO! They just borrow a lot of **thermology** from Bayesian inference
- **We learn the model by approximate MLE**
- The prior $p(\mathbf{z})$ is not a Bayesian prior: we don't update it
  - It does not change after we have seen data
  - We may learn it by MLE (as we will see soon)
- The **posterior $p(\mathbf{z}|\mathbf{x}) \approx q(\mathbf{z}|\mathbf{x})$** corresponds to the **component responsibility** in a mixture model
- **You can make Bayesian VAEs** by putting Bayesian priors on, e.g., decoder parameters θ[1]

[1]Daxberger E, Hernández-Lobato JM. Bayesian Variational Autoencoders for Unsupervised Out-of-Distribution Detection. arXiv:1912.05651, 2019.

Issues with VAEs (1/3)

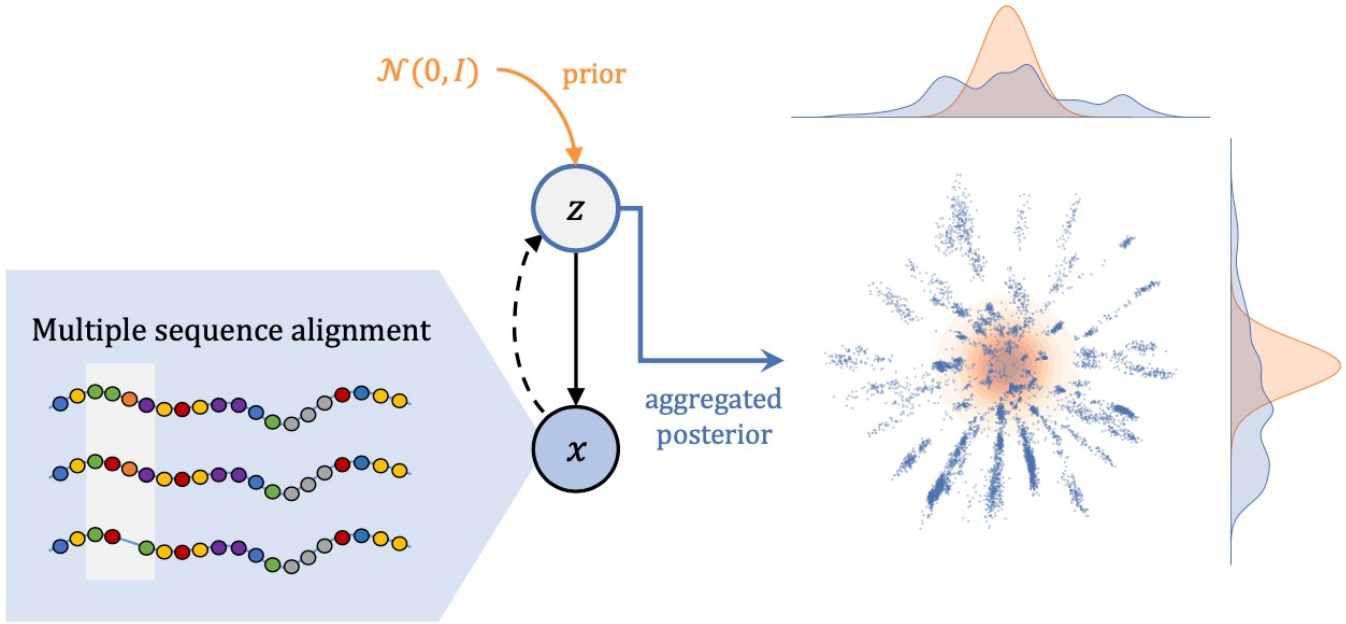
- Posterior collapse[1] (e.g.)
  - For a non-trainable prior, e.g., standard Gaussian, the **regularisation term** will be minimized if $\forall \mathbf{x} : q_\phi(\mathbf{z}|\mathbf{x}) = p(\mathbf{z})$
  - A powerful decoder can learn to **ignore** $\mathbf{z}$ (e.g. ARM like a LSTM)
  - Can partially be addressed by warm-up of the KL-term[2,3] (e.g.)

$$\mathcal{L}(\theta, \phi) = \overbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}^{\text{reconstruction term}} - \overbrace{\mathrm{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\right]}^{\text{regularisation term}}$$

[1] Lucas J, Tucker G, Grosse R, Norouzi M. Understanding Posterior Collapse in Generative Latent Variable Models. 2019.
[2] Higgins I, Matthey L, Pal A, Burgess C, Glorot X, Botvinick M, Mohamed S, Lerchner A. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. ICLR, 2017.
[3] Sønderby CK, Raiko T, Maaløe L, Sønderby SK, Winther O. Ladder Variational Autoencoders. NeurIPS, 2016.

Issues with VAEs (2/3)

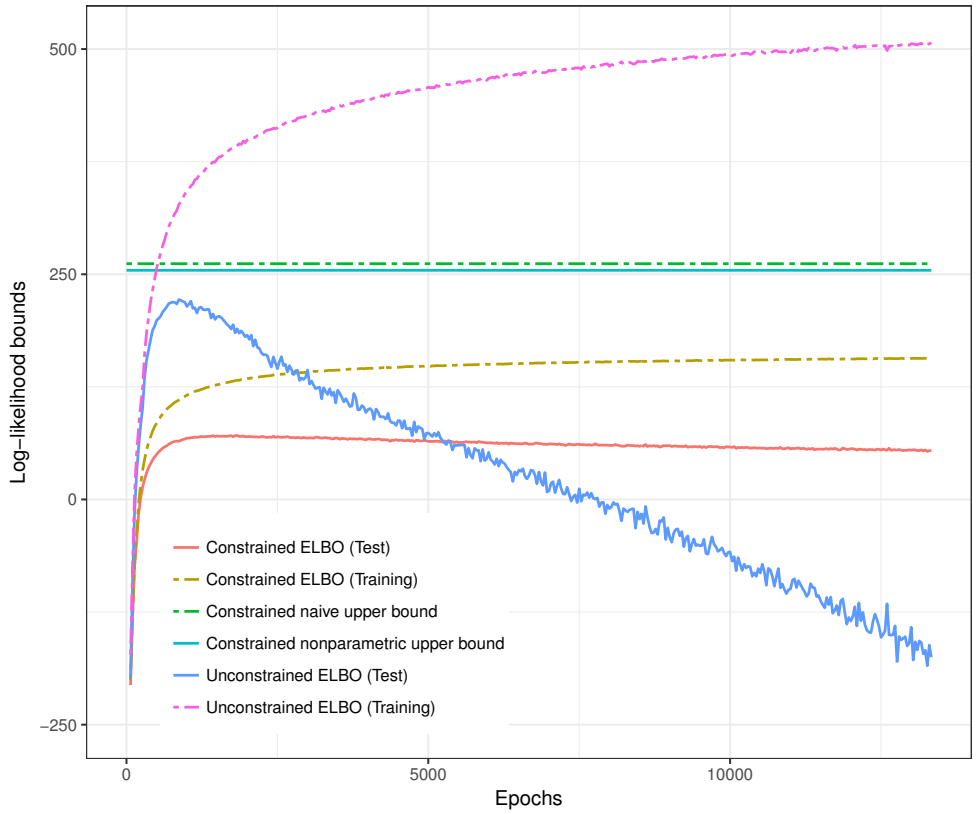
- The **hole problem**[1]
  - Mismatched between the prior $p(\mathbf{z})$ and the aggregated posterior
    $$q_\phi(\mathbf{z}) = \frac{1}{N}\sum_n q_\phi(z|x_n)$$
  - Sampling from these holes gives unrealistic $\mathbf{z}$'s ⇒ $p(\mathbf{x}|\mathbf{z})$ is low quality



[1]Rezende DJ, Viola F. Taming VAEs. arXiv:1810.00597, 2018.
Figure from: Arts ME, Bergamin F, Boomsma W, Frellsen J. Sampling quality in deep generative models of protein sequences. Unpublised, 2023.

Issues with VAEs (3/3)

- **Unbounded likelihood**[1]
  - As for GMMs, **MLE is ill-posed** for VAEs with Gaussian output density
  - ***The likelihood is unbounded***
  - ***Detrimental to generalisation***
  - **Solution**: regularise, clamp or Bayesian about co-variance

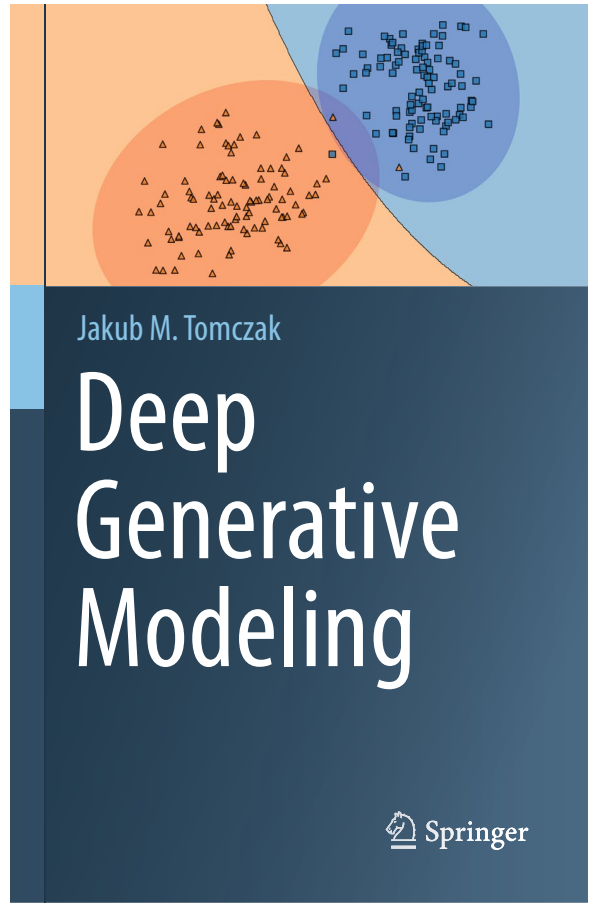


[1]Mattei PA, Frellsen J. Leveraging the Exact Likelihood of Deep Latent Variable Models. NeurIPS, 2018.

Improving VAEs

- We can improve all the components of a VAE
  - Prior
  - Inference
  - Variational distribution
- Here, we focus on the prior

... but there is much more



Improving the prior

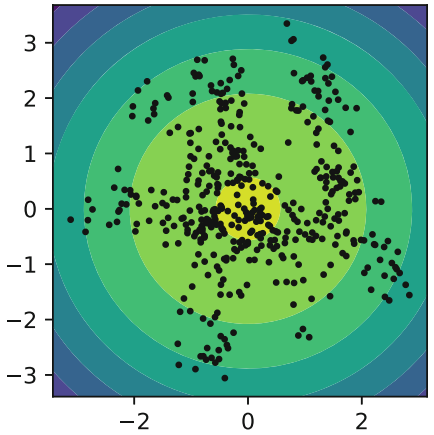
$$\mathcal{L}(\theta, \phi) = \overbrace{\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}^{\text{reconstruction term}} - \overbrace{\mathrm{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\lambda(\mathbf{z})\right]}^{\text{regularisation term}}$$

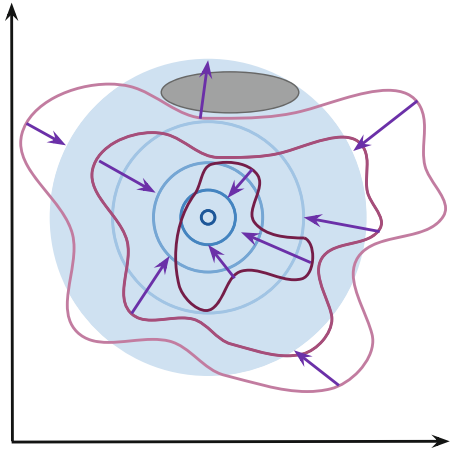
- We can rewrite the **regularization** term as

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}\left[-\mathrm{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\lambda(\mathbf{z})\right]\right] = -\mathbb{CE}\left[q_\phi(\mathbf{z}) \parallel p_\lambda(\mathbf{z})\right] + \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}\left[\mathbb{H}\left[q_\phi(\mathbf{z}|\mathbf{x})\right]\right]$$

  where $q_\phi(\mathbf{z}) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[q_\phi(\mathbf{z}|\mathbf{x})] \approx \frac{1}{N}\sum_n q_\phi(\mathbf{z}|\mathbf{x}_n)$ is the **aggregated posterior**

- **Difficulties in optimising the CE** wrt. q_ϕ for a fixed prior gives rise **holes**
- The cross entropy, is minimised wrt. $p_\lambda(\mathbf{z})$ when $p_\lambda(\mathbf{z}) = q_\phi(\mathbf{z})$
- This can be addressed by learning the prior $p_\lambda(\mathbf{z})$





Figures from: Tomczak JM. Deep generative modeling. Springer, 2022.

Mixture of Gaussians (MoG) prior

You'll try this later!

- What is then the best prior? The **aggregated posterior**:

$$p_\lambda(\mathbf{z}) = \frac{1}{N}\sum_{n=1}^{N} q_\phi(\mathbf{z}|\mathbf{x}_n)$$

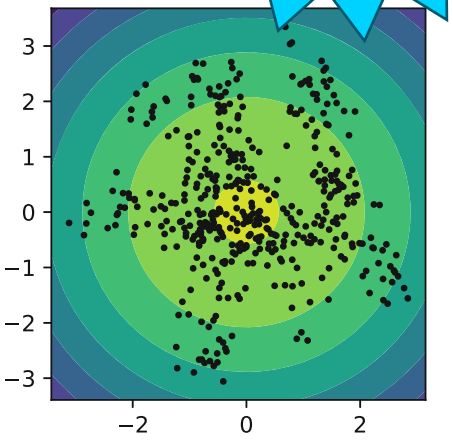
  It minimizes CE, but is not feasible for large N

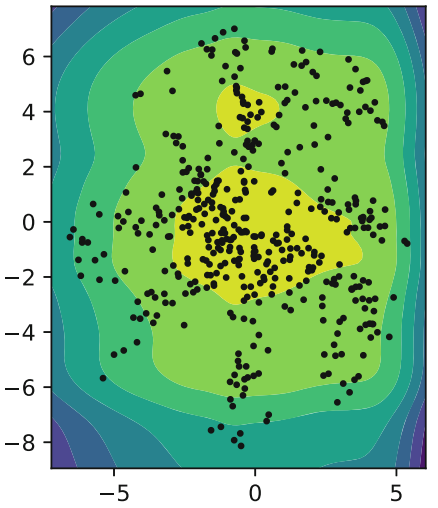
- Notice that this is a mixture distribution!
- So, we could use a **MoG prior** with $K < N$ components

$$p_\lambda(\mathbf{z}) = \frac{1}{K}\sum_{k=1}^{K} w_k \mathcal{N}\left(\mathbf{z}|\boldsymbol{\mu}_k, \mathrm{diag}(\boldsymbol{\sigma}_k^2)\right)$$

  were $\lambda = \left\{\{w_k\}, \{\boldsymbol{\mu}_k\}, \{\boldsymbol{\sigma}_k^2\}\right\}$ are learnable parameters

- *Doesn't this introduce a new latent variable?*
- **This also allows for end-to-end clustering with VAEs**





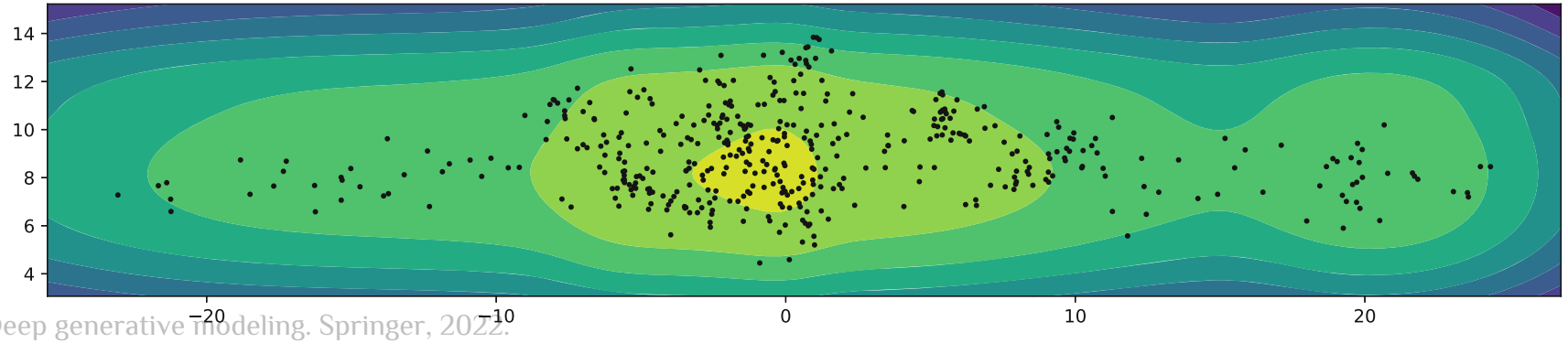
Figures from: Tomczak JM. Deep generative modeling. Springer, 2022.

Variational mixture of posterior prior (VampPrior)[1]

- We can improve on the MoG prior by using a **mixture of the approximate posterior** with $K < N$ pseudo-inputs

$$p_\lambda(\mathbf{z}) = \frac{1}{K}\sum_{k=1}^{K} q_\phi(\mathbf{z}|\mathbf{u}_k)$$

  where $\lambda = \{\{\mathbf{u}_k\}, \phi\}$ are learnable parameters
- Improves the quality of the model (**fewer holes**)
- Challenging initializing pseudo-inputs



Figures from: Tomczak JM. Deep generative modeling. Springer, 2022.
[1]Tomczak J, Welling M. VAE with a VampPrior. AISTATS, 2018.

Hierarchical latent variable models[1,2]

- Can we build a **flexible** prior using an **inductive bias**?
  - Common (ML) **hypothesis**: our world can be organized **hierarchically**
- We can make a VAE, with multiple hierarchical latent variable, e.g.

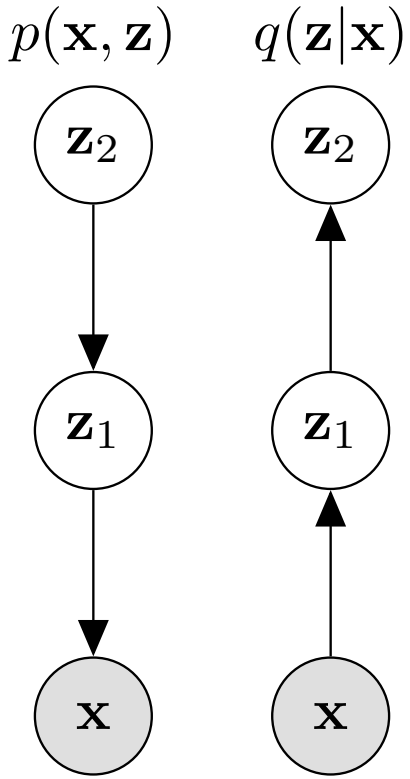
$$p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2) = p(\mathbf{x}|\mathbf{z}_1)p(\mathbf{z}_1|\mathbf{z}_2)p(\mathbf{z}_2)$$

- A natural variational posterior is **bottom-up**

$$q(\mathbf{z}_1, \mathbf{z}_2|\mathbf{x}) = q(\mathbf{z}_2|\mathbf{z}_1)q(\mathbf{z}_1|\mathbf{x})$$

- However, we easily get posterior collapse for $\mathbf{z}_1$

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{z}_1,\mathbf{z}_2|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z}_1) + \log \frac{p(\mathbf{z}_1|\mathbf{z}_2)}{q(\mathbf{z}_1|\mathbf{x})} - \mathrm{KL}[q(\mathbf{z}_2|\mathbf{z}_1)||p(\mathbf{z}_2)]\right]$$



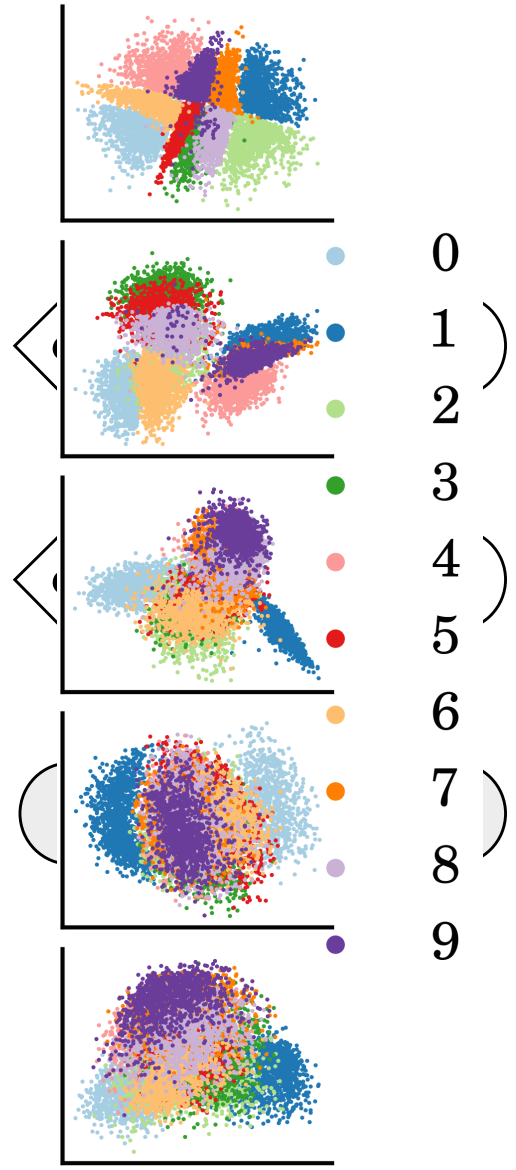
[1]Rezende DJ, Mohamed S, Wierstra D. Stochastic backpropagation and approximate inference in deep generative models. ICML 2014.
[2]Sønderby CK, Raiko T, Maaløe L, Sønderby SK, Winther O. Ladder Variational Autoencoders. NeurIPS, 2016.

Hierarchical latent variable models

- Instead, we can write the variational posteriors **top-down**

$$q(\mathbf{z}_1, \mathbf{z}_2|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{z}_2, \mathbf{x})q(\mathbf{z}_2|\mathbf{x})$$

- The decoder is **shared** between the variational posterior and the prior
- This helps **preventing posterior collapse**
- Warm-up (gradually turning on the KL-term) is often still required



Figures from: Sønderby CK, Raiko T, Maaløe L, Sønderby SK, Winther O. Ladder Variational Autoencoders. NeurIPS, 2016.

Very deep VAEs (48 layers) on FFHQ-256[1]

[1]Child R. Very deep VAEs generalize autoregressive models and can outperform them on images. arXiv:2011.10650, 2020.

Take-away!

- DLVM: a power framework for building generative model
- VAE: a DLVM learned with amortized variational inference
- Allows for **representation learning** and **semi-supervised** learning
- Key components we can **improve**
  - Prior
  - Inference
  - Variational distribution

Exercise 1: Variational autoencoders

- Jupiter notebook with VAE implementation
  - Runs in Google Colab
- Do exercise 1.1-1.3
  - Exercise 1.1: Understand the code
  - Exercise 1.2: Carucate test-ELBO and plot the aggregate posterior
  - Exercise 1.3: Implement a MoG prior (you may not get to here)
- Available at:

`https://github.com/frellsen/ProbAI-2024`

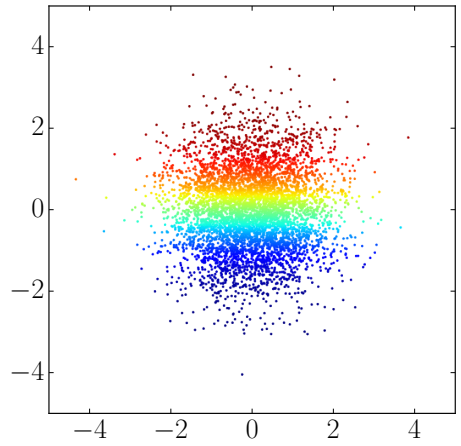
Normalising Flows

Part III

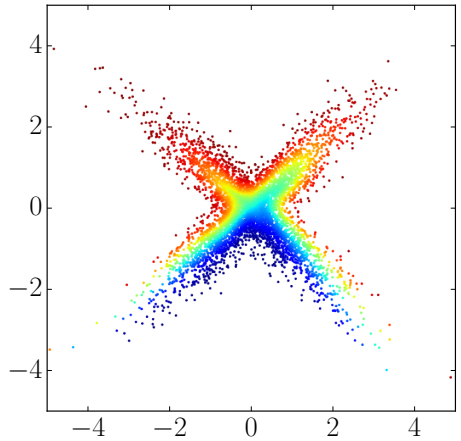
Flow-based models

How is that different from a VAE?

$u \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$



$T(\mathbf{u})$



Generative process is

$\mathbf{u} \sim p(\mathbf{u})$

$\mathbf{x} = f(\mathbf{u})$

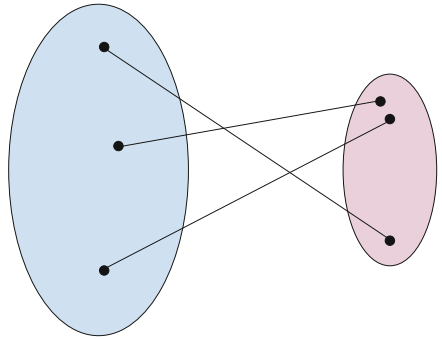
- If we want to calculate the density of $\mathbf{x}$, we need to use the **change-of-variable formular** for probability densities
- This requires f to **invertible** (bijection) and **differentiable**

VAE generative process

$z \sim p(\mathbf{z})$

$\eta = f(\mathbf{z})$

$\mathbf{x} \sim p(\mathbf{x}|\eta)$



In VAEs $\dim \mathbf{z} < \dim \mathbf{x}$, hence f not bijective

Figure from: Papamakarios G, Nalisnick E, Rezende DJ, Mohamed S, Lakshminarayanan B. Normalizing Flows for Probabilistic Modeling and Inference. JMLR, 2021.

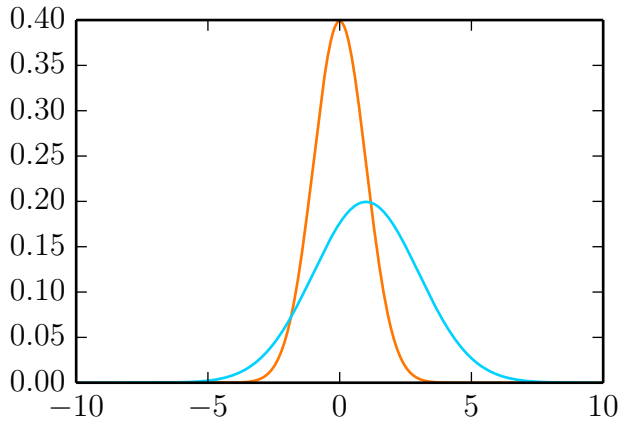
A simple (well-known) example

For $u \in \mathbb{R}$, let $p(u) = \mathcal{N}(u|0,1)$ and $x = T(u) = 2u + 1$

Q: What is the density of x?

A: The density is $p_x(x) = \mathcal{N}(x|1,2^2)$

How did we get there?



The **change of variable formular** for scalar densities

$$p_x(x) = p_u(u)\left|\frac{du}{dx}\right| = p_u\left(T^{-1}(x)\right)\left|\frac{d}{dx}T^{-1}(x)\right|$$

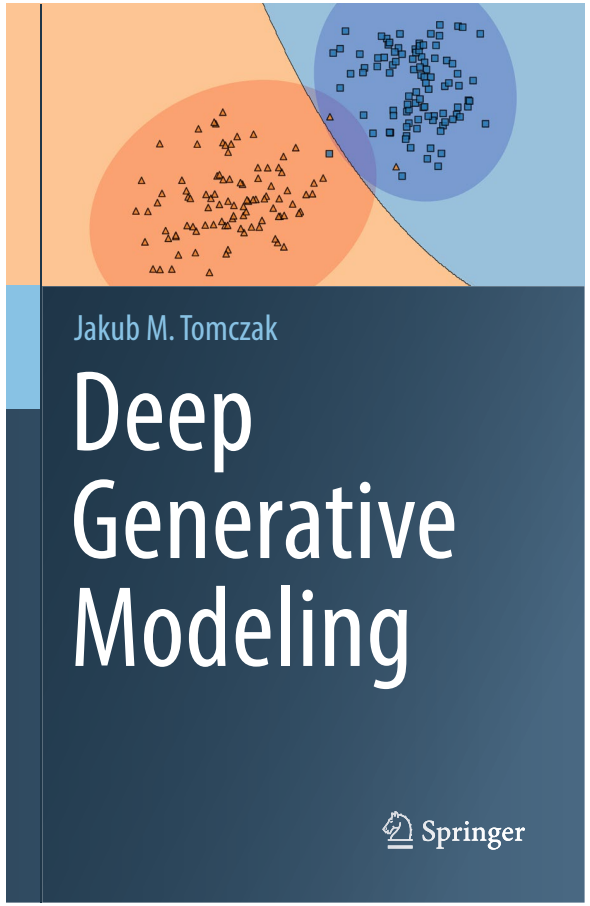
We have that

$$\mathcal{N}(u|0,1) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}u^2\right)$$

$$T^{-1}(x) = \frac{x-1}{2} \quad \text{and} \quad \left|\frac{d}{dx}T^{-1}(x)\right| = \frac{1}{2}$$

$$p_x(x) = p_u\left(\frac{x-1}{2}\right)\frac{1}{2} = \frac{1}{2}\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}\frac{(x-1)^2}{2^2}\right) = \mathcal{N}(x|1,2^2)$$

Literature



Normalizing Flows for Probabilistic Modeling and Inference

George Papamakarios* GPAPAMAK@GOOGLE.COM
Eric Nalisnick* ENALISNICK@GOOGLE.COM
Danilo Jimenez Rezende DANILOR@GOOGLE.COM
Shakir Mohamed SHAKIR@GOOGLE.COM
Balaji Lakshminarayanan BALAJILN@GOOGLE.COM
DeepMind

Editor: Ryan P. Adams

Abstract

Normalizing flows provide a general mechanism for defining expressive probability distributions, only requiring the specification of a (usually simple) base distribution and a series of bijective transformations. There has been much recent work on normalizing flows, ranging from improving their expressive power to expanding their application. We believe the field has now matured and is in need of a unified perspective. In this review, we attempt to provide such a perspective by describing flows through the lens of probabilistic modeling and inference. We place special emphasis on the fundamental principles of flow design, and discuss foundational topics such as expressive power and computational trade-offs. We

Normalizing Flows (NFs)[1,2]

Let $\mathbf{u}, \mathbf{x} \in \mathbb{R}^D$ be random variables driven by the model:

$$\mathbf{u} \sim p_{\mathrm{u}}(\mathbf{u}) \quad \text{and} \quad \mathbf{x} = T(\mathbf{u})$$

where

- $p_{\mathrm{u}}(\mathbf{u})$ is the **base distribution**
- The **transformation T is a diffeomorphism** (invertible and T, T^{-1} differentiable)
- p_{u} is parametrized by ϕ
- T is parametrized by ψ

...and it **evaluable!**
"Exact likelihood model"

The density of is $\mathbf{x}$ is well-defined by the change of variable

$$p_{\mathrm{x}}(\mathbf{x}) = p_{\mathrm{u}}(\mathbf{u}) |\det J_T(\mathbf{u})|^{-1} \quad \text{where} \quad \mathbf{u} = T^{-1}(\mathbf{x})$$

$$J_T(\mathbf{u}) = \begin{bmatrix} \frac{\partial T_1}{\partial \mathrm{u}_1} & \cdots & \frac{\partial T_1}{\partial \mathrm{u}_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_D}{\partial \mathrm{u}_1} & \cdots & \frac{\partial T_D}{\partial \mathrm{u}_D} \end{bmatrix}$$

[1]Esteban TG, Vanden-Eijnden E. Density estimation by dual ascent of the log-likelihood. Commun. Math. Sci., 2010.
[2]Papamakarios G, Nalisnick E, Rezende DJ, Mohamed S, Lakshminarayanan B. Normalizing Flows for Probabilistic Modeling and Inference. JMLR, 2021.

Flows: the Jacobian

- The **inverse function theorem**, $J_{T^{-1}} = J_T^{-1}$, gives the equivalent formulations

$$p_{\mathrm{x}}(\mathbf{x}) = p_{\mathrm{u}}(\mathbf{u})|\det J_T(\mathbf{u})|^{-1} = p_{\mathrm{u}}(\mathbf{u})|\det J_{T^{-1}}(\mathbf{x})|$$

- $|\det J_T(\mathbf{u})|$ quantifies the **relative change of volume** around $\mathbf{u}$ due to T
- When $|\det J_T(\mathbf{u})| = 1$, then transformation is "volume preserving"

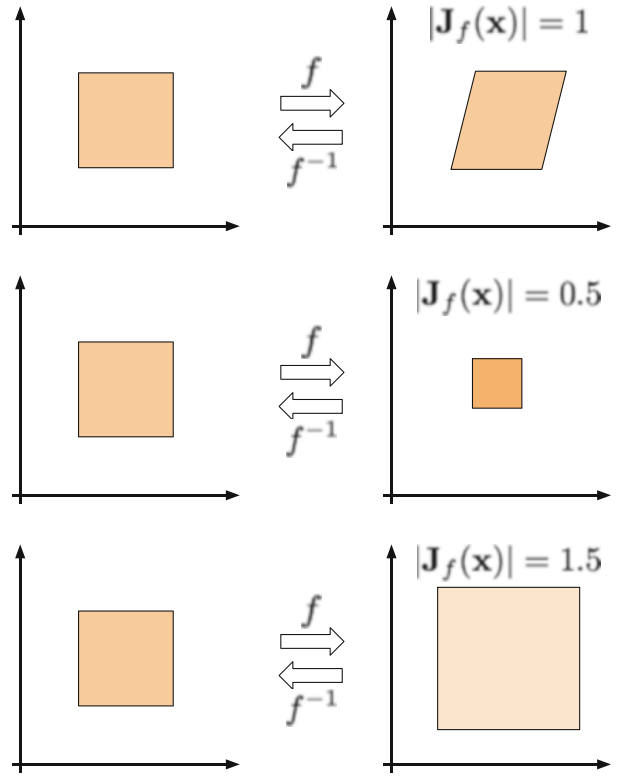


Figure from Tomczak JM. Deep generative modeling. Springer, 2022.

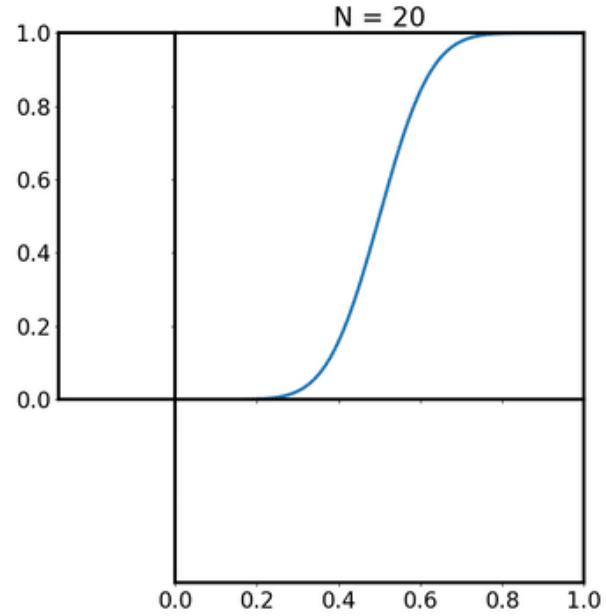
Another simple (well-known) example

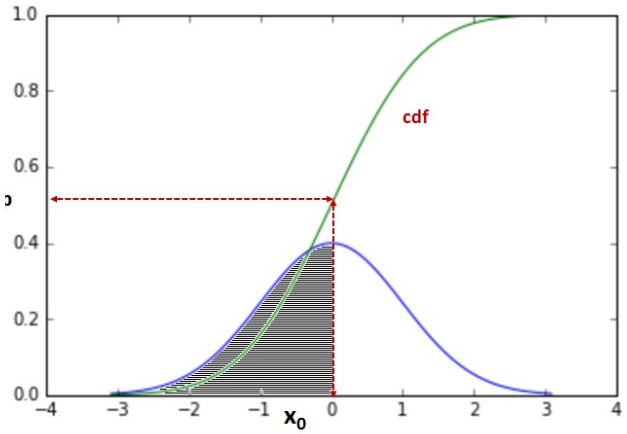
- Recall **inverse transform sampling:**
  - Procedure from for sampling from $p(x)$
  - $p(x)$ is **any* continuous univariate probability distribution**
  - Utilize the cumulative distribution function (CDF): $F(x) = \int_{-\infty}^{x} p(x')\,\mathrm{d}x'$
- We obtain samples x from $p(x)$ by the procedure

$$u \sim U(0,1)$$
$$x = F^{-1}(u)$$

- This is a flow with $p(x) = U(0,1)$ and $T(u) = F^{-1}(u)$!
- Assuming CDF is invertiable, transforms $U(0,1)$ to any distribution
- **So, a flow can represent nearly any distribution!**
  - A similar argument can be in **multiple dimensions**

CDF for $\mathcal{N}(0,1)$





Figures from https://en.wikipedia.org/wiki/Inverse_transform_sampling and http://pytolearn.csd.auth.gr/d1-hyptest/11/distros.html

[1]Papamakarios G, Nalisnick E, et al.. Normalizing Flows for Probabilistic Modeling and Inference. JMLR, 2021.

Learning NFs from data

- We want to **learn the parameters ψ of T** and possibly ϕ of p_{u}
- Given observations $\{x_1, \ldots, x_N\}$, a natural objective is the **log-likelihood function**

$$\ell(\psi, \phi) = \sum_{n=1}^{N} \log p_{\mathrm{u}}(T^{-1}(\mathbf{x}_n; \psi); \phi) + \log|\det J_{T^{-1}}(\mathbf{x}_n; \psi)|$$

- We would like to find the **ML estimate**: $\hat{\psi}, \hat{\phi} = \arg\max_{\psi,\phi} \ell(\psi, \phi)$.
- We can do that by SGD, but we need to:
  - Evaluate $T^{-1}(\mathbf{x}_n; \psi)$ and its log Jacobian-determinant
  - Evaluate $p_{\mathrm{u}}(\mathbf{u}; \phi)$
  - ... and **their gradients** (i.e., backprop though them)

Note: For MLE, we just need:
- T^{-1}
- $J_{T^{-1}}$ (or J_T)

We don't need T
- But T is need for **sampling**

Challenge in specifying the transformation

- We usually use a standard Gaussian as base, $p_{\mathbf{u}}(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\mathbf{0}, I_D)$
- We would like a flexible **transformation T**, i.e., a neural net (NN)
- But how do we specify a NN, where
  1. The **function is invertible**
  2. The **log Jacobian-determinant is easy to evaluate**
- These properties are **not** fulfilled by **standard feedforward NN**
- However, we formulate **simple NN-based layers** fulfilling these
  - ... and then **we can stack them** to get expressive power!

Picture from https://themodernproper.com/buttermilk-pancakes

Composable transformation

- We can construct a flow by composing **simpler transformations**

$$T = T_K \circ \cdots \circ T_1$$

Composition is also invertible

- Each transformation T_k fulfil the properties above (invertible and Jacobian)
- We can iteratively evaluate both the **forward** and **inverse** transformations

$$\mathbf{z}_k = T_k(\mathbf{z}_{k-1}) \quad \text{for } k = 1, \dots, K$$

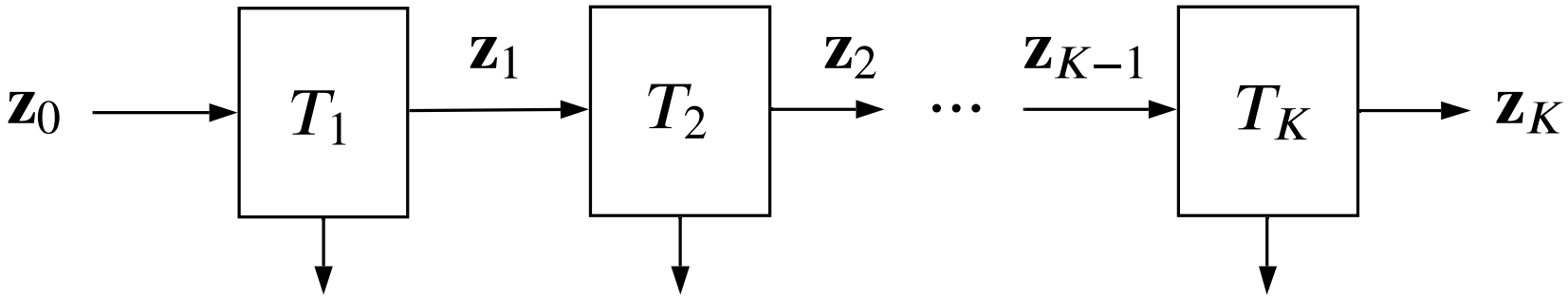
$$\mathbf{z}_{k-1} = T_k^{-1}(\mathbf{z}_k) \quad \text{for } k = K, \dots, 1$$

- ... and the log Jacobian-determinant

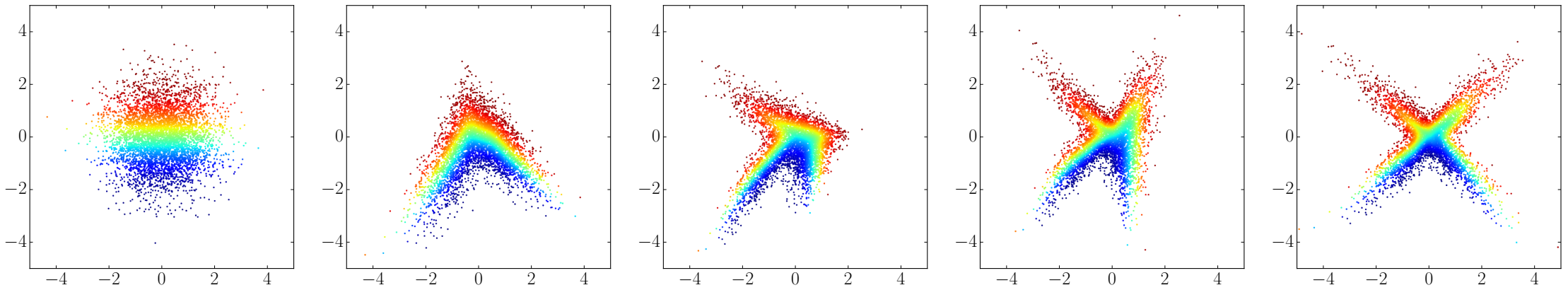
$$\log|\det J_T(\mathbf{x})| = \log\left|\prod_{k=1}^{K} \det J_{T_k}(\mathbf{x})\right| = \sum_{k=1}^{K} \log|\det J_{T_k}(\mathbf{x})|$$

Chain rule for Jacobian

Composable transformation



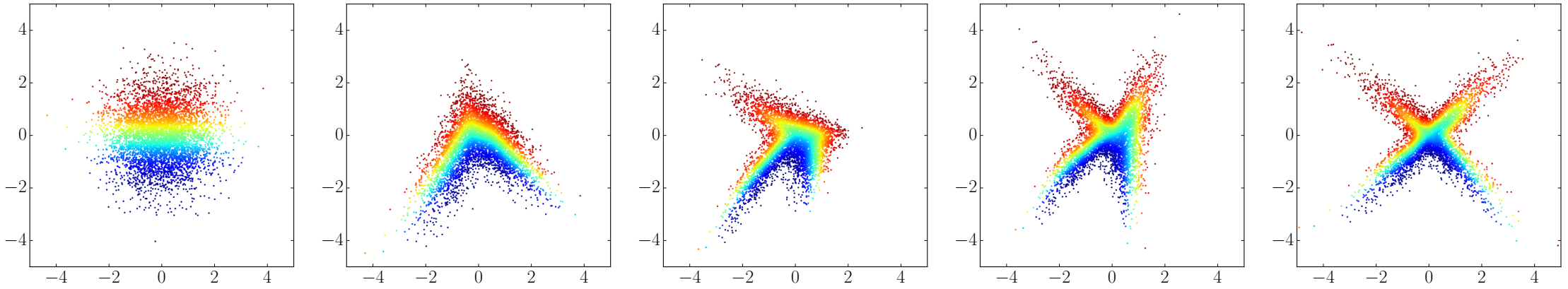
$$\log|\det J_{T_1}(\mathbf{z}_0)| + \log|\det J_{T_2}(\mathbf{z}_1)| + \cdots + \log|\det J_{T_K}(\mathbf{z}_{K-1})| = \log|\det J_T(\mathbf{z}_0)|$$



Figures from Papamakarios G, Nalisnick E, Rezende DJ, Mohamed S, Lakshminarayanan B. Normalizing Flows for Probabilistic Modeling and Inference. JMLR, 2021.

Why are the called "Normalizing Flows"?

- "**Flow**": Refers to the **sequence** of transformation $T = T_K \circ \cdots \circ T_1$ where a sample $\mathbf{u}$ *flows* from the base to the output $\mathbf{x}$
- "**Normalising**": Refers to the **reverse flow** where a datum $\mathbf{x}$ *flows* back to a **normal distributed base**, i.e., **data is normalised**



Figures from Papamakarios G, Nalisnick E, Rezende DJ, Mohamed S, Lakshminarayanan B. Normalizing Flows for Probabilistic Modeling and Inference. JMLR, 2021.

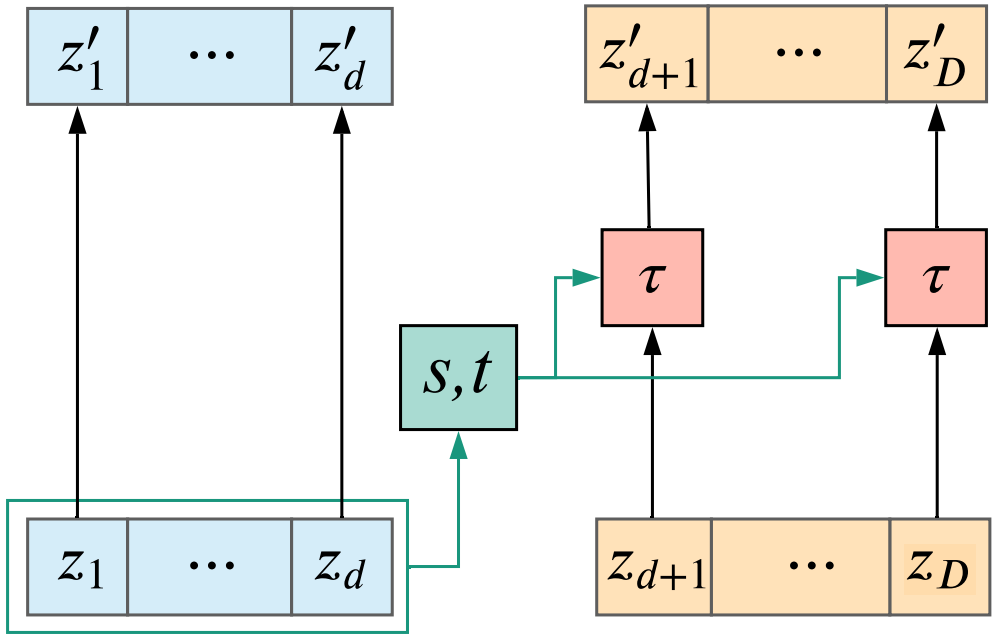
How to construct the transformations?

- Using a **neural net**, we want to construct a transformation that is
  - **Invertible**
  - **Jacobian is easy evaluate**

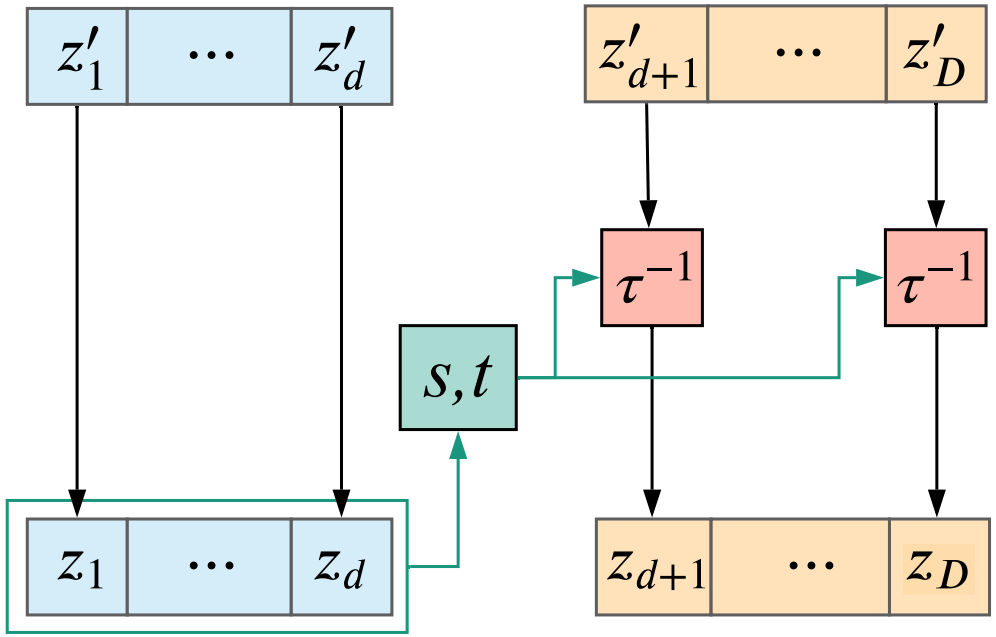
- We will go in **details** with to such transformations
  - **Affine coupling layers**
  - **Permutation layers**

[1]Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP. ICLR, 2017.

Affine coupling layers



Forward



Inverse

τ is the affine transformation

s and t are the scaling and translation

Figures adapted from Papamakarios G, Nalisnick E, Rezende DJ, Mohamed S, Lakshminarayanan B. Normalizing Flows for Probabilistic Modeling and Inference. JMLR, 2021.

Affine coupling layers

$$\forall i \leq d : \mathrm{z}'_i = z_i$$

$$\forall i > d : \mathrm{z}'_i = \exp\big(s(\mathbf{z}_{\leq d})\big)_i z_i + t(\mathbf{z}_{\leq d})_i$$

- **Affine coupling** layers were introduced by Dinh et al. (2015)
  - Key components in **RealNVP**[2] (Real-valued Non-Volume Preserving flows)
- The transformation $T: \mathbf{z} \to \mathbf{z}'$ partitions $\mathbf{z} = [\mathbf{z}_{\leq d}, \mathbf{z}_{>d}]$ at $d < D$:

$$\mathbf{z}'_{\leq d} = \mathbf{z}_{\leq d}$$

$$\mathbf{z}'_{>d} = \exp\big(s(\mathbf{z}_{\leq d})\big) \odot \mathbf{z}_{>d} + t(\mathbf{z}_{\leq d})$$

Arbitrary neural nets

$$s : \mathbb{R}^d \to \mathbb{R}^{D-d}$$
$$t : \mathbb{R}^d \to \mathbb{R}^{D-d}$$

- Since $\mathbf{z}'_{>d}$ is a affine transformation of $\mathbf{z}_{>d}$, its easily invertible

$$\mathbf{z}_{\leq d} = \mathbf{z}'_{\leq d}$$

$$\mathbf{z}_{>d} = \exp\big(-s(\mathbf{z}_{\leq d})\big) \odot \big(\mathbf{z}'_{>d} - t(\mathbf{z}_{\leq d})\big)$$

$$\mathbf{z}'_{>d} = \exp\big(s(\mathbf{z}_{\leq d})\big) \odot \mathbf{z}_{>d} + t(\mathbf{z}_{\leq d})$$

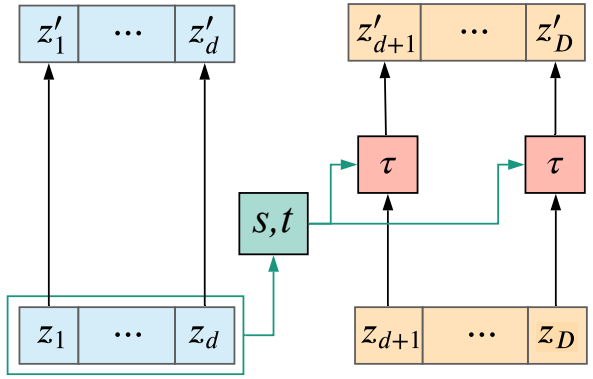
$$\mathbf{z}'_{>d} - t(\mathbf{z}_{\leq d}) = \exp\big(s(\mathbf{z}_{\leq d})\big) \odot \mathbf{z}_{>d}$$

$$\exp\big(-s(\mathbf{z}_{\leq d})\big) \odot \big(\mathbf{z}'_{>d} - t(\mathbf{z}_{\leq d})\big) = \mathbf{z}_{>d}$$

[1]Dinh l, Krueger D, and Bengio Y. NlCE: Non-linear independent components estimation. lCLR workshop track, 2015.
[2]Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP. lCLR, 2017.

Affine coupling layers: the Jacobian (1)

- It turn out, that **the Jacobian is easy**
- Let's write it on the following block form



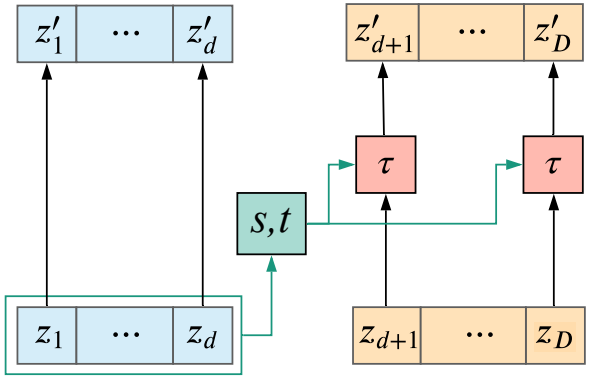
Forward

$$J_T(\mathbf{z}) = \begin{bmatrix} \frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{\leq d}} & \frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{>d}} \\ \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{\leq d}} & \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{>d}} \end{bmatrix}$$

Affine coupling layers: the Jacobian (2)

Recall that $\mathbf{z}'_{\leq d} = \mathbf{z}_{\leq d}$, and so we find

$$\frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{\leq d}} = \begin{bmatrix} \frac{\partial z_1}{\partial z_1} & \cdots & \frac{\partial z_1}{\partial z_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_d}{\partial z_1} & \cdots & \frac{\partial z_d}{\partial z_d} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} = I_{d \times d}$$

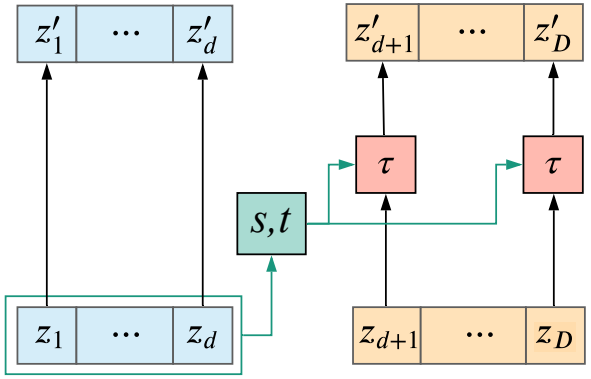


Forward

$$\frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{>d}} = \begin{bmatrix} \frac{\partial z_1}{\partial z_{d+1}} & \cdots & \frac{\partial z_1}{\partial z_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_d}{\partial z_{d+1}} & \cdots & \frac{\partial z_d}{\partial z_D} \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} = 0_{d \times (D-d)}$$

Affine coupling layers: the Jacobian (3)

Recall that $\forall i > d : \mathrm{z}'_{\mathrm{i}} = \exp\big(s(z_{\leq d})\big)_i z_i \; + t(z_{\leq d})_i$



Forward

$$\frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{>d}} = \begin{bmatrix} \frac{\partial z_{d+1}}{\partial z_{d+1}} & \cdots & \frac{\partial z_{d+1}}{\partial z_{d+1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_D}{\partial z_{d+1}} & \cdots & \frac{\partial z_D}{\partial z_D} \end{bmatrix}$$

Q: What is this block of the Jacobian?

$$= \begin{bmatrix} \exp\big(s(\mathbf{z}_{\leq d})\big)_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \exp\big(s(\mathbf{z}_{\leq d})\big)_{D-d} \end{bmatrix}$$

$$= \mathrm{diag}\big(\exp\big(s(\mathbf{z}_{\leq d})\big)\big)$$

Affine coupling layers: the Jacobian (3)

- This mean that

$$J_T(\mathbf{z}) = \begin{bmatrix} I_{d\times d} & 0_{d\times(D-d)} \\ \dfrac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{\leq d}} & \operatorname{diag}\left(\exp\left(s(\mathbf{z}_{\leq d})\right)\right) \end{bmatrix}$$

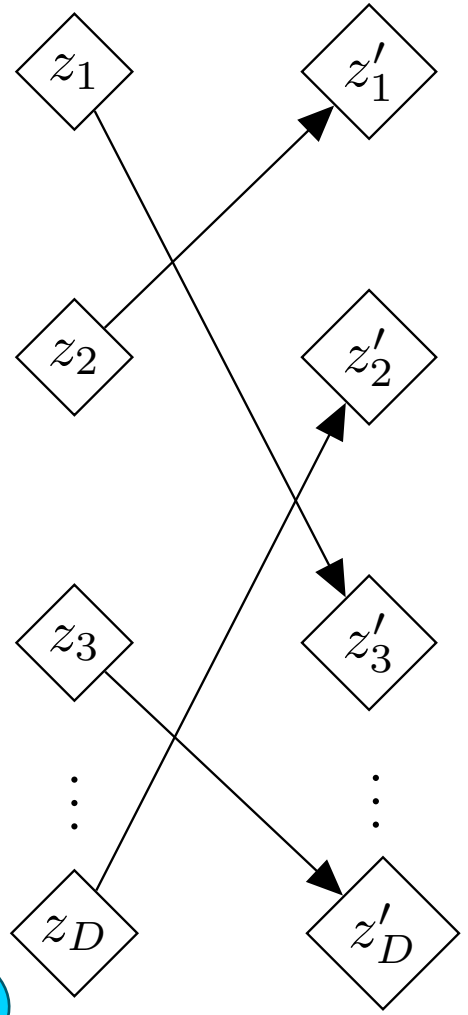
- Since $J_T(\mathbf{z})$ is **triangular**, we have

$$\det J_T(\mathbf{z}) = \prod_{i=1}^{D-d} \exp\left(s(\mathbf{z}_{\leq d})\right)_i = \exp\left(\sum_{i=1}^{D-d} s(\mathbf{z}_{\leq d})_i\right)$$

Recall, for triangular matrix A we have $\det A = \prod_i a_{i,i}$

Permutation layer

- With coupling layers, we need to **permute** the variables
  - If the partitioning is fixed, e.g., $d = \frac{D}{2}$, some variables will never be transformed
  - A permutation is **invertible**
  - A permutation is **volume preserving**, i.e., $|\det J_T(\mathbf{z})| = 1$



Q: What is $|\det J_T(\mathbf{u})|$ for a permutation?

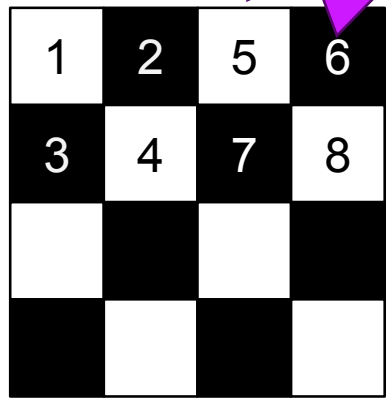
Hint: Recall, if we swap two rows (or columns) in a matrix, the determinant change sign.

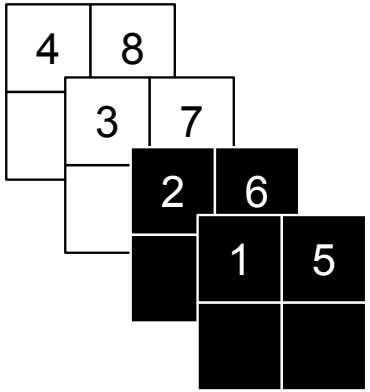
RealNVP[1]

- Masked coupling layers
- No permutation needed

You will implement this later!

- RealNVP employ affine coupling layers
- The partitioning is implemented by masking:
  - Masking variables in a **checkerboard** pattern (alternating)
    - This helps to learn **spatial dependencies**
  - Masking **channel-wise**
    - Ensures the checkerboard masking in **not redundant**
  - Use **squeezing**: reshape input tensor
    - Converts special dimensions to channels
    - Gives a **multi-scale** behaviour



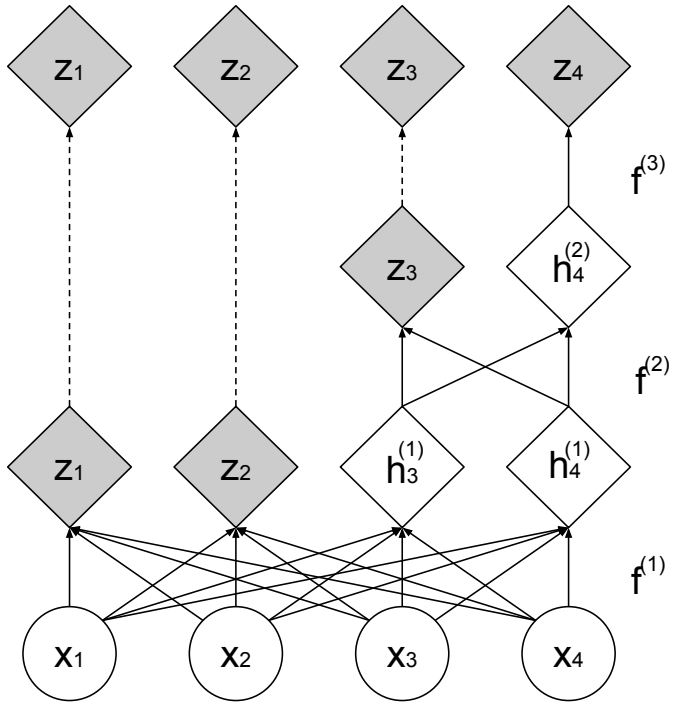


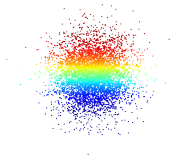
[1]Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP. ICLR, 2017. Figures from [1].

RealNVP[1]: Factoring out variables

- At regular intervals, **half of the dimensions** are "factored out"
  - I.e. they are **not transformed anymore**
- Different pixels are **Gaussianized**
  - In an **earlier layer** (fine scale)
  - In a **later layer** (coarser scale)

$$
\begin{aligned}
h^{(0)} &= x \\
(z^{(i+1)}, h^{(i+1)}) &= f^{(i+1)}(h^{(i)}) \\
z^{(L)} &= f^{(L)}(h^{(L-1)}) \\
z &= (z^{(1)}, \ldots, z^{(L)})
\end{aligned}
$$





[1]Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP. ICLR, 2017. Figures from [1].

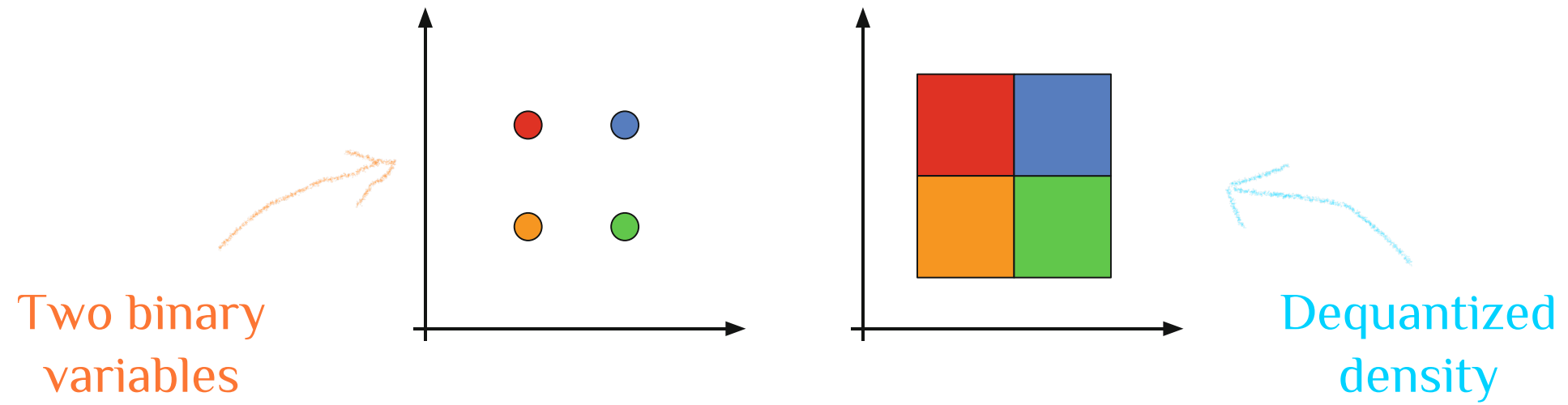
Dequantization

In flows we assume that $\mathbf{x} \in \mathbb{R}^D$, i.e., real-valued

Q: How can we model images $\tilde{\mathbf{x}} \in \{0, 1, \ldots, 255\}^D$

A: We can add noise (dequantize) to $\tilde{\mathbf{x}}$

$$x_i = \tilde{x}_i + u \quad \text{where} \quad u \sim U(-0.5, 0.5)$$

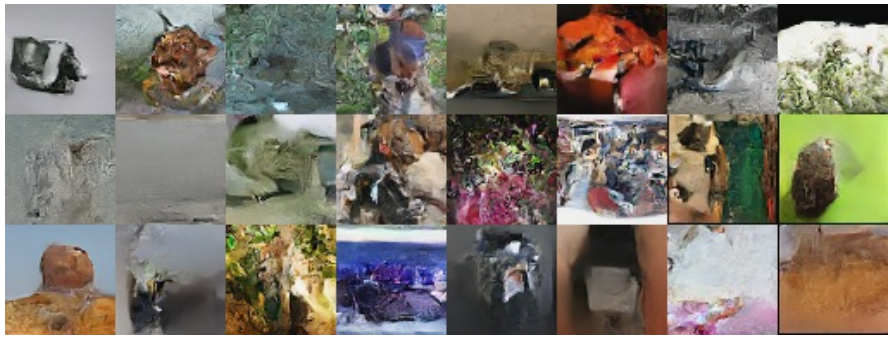


RealNVP[1]

Data

Samples

Imagenet (64x64)



CelebA

LSUN (bedroom)

[1]Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP. ICLR, 2017. Figures from [1].

Glow: samples from model on CelebA HQ[1]

[1] Kingma DP, Dhariwal P. Glow: generative flow with invertible 1× 1 convolutions. NeurIPS, 2018.

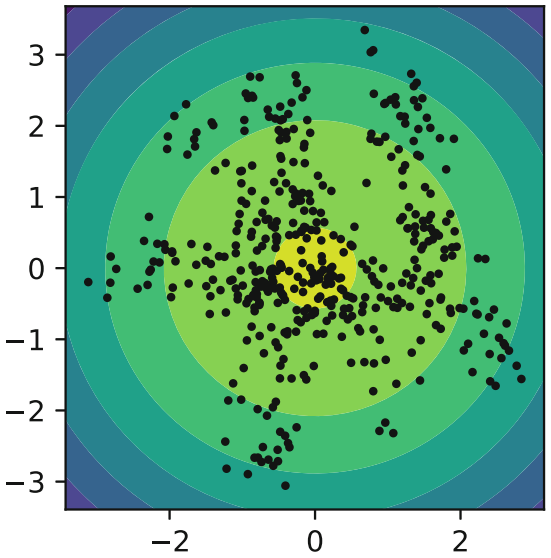
What can we use them for?

- **Density estimation**
  - With **exact likelihood**
  - Can **approximate any distribution**
  - But, layers are simple, so many are required (c.f. Glow)
- **Variational inference**
  - A flow can be a very **flexible approximate posterior**
- **Flexible priors in VAEs**
- ... and more (see section 6 in Papamakarios et al.[1])

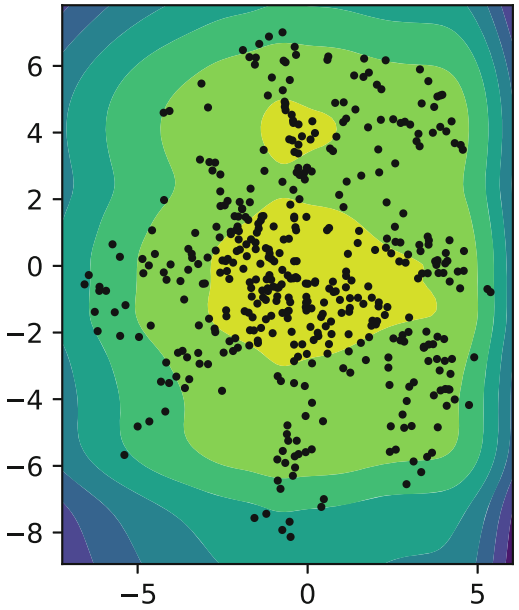
[1]Papamakarios G, Nalisnick E, Rezende DJ, Mohamed S, Lakshminarayanan B. Normalizing Flows for Probabilistic Modeling and Inference. JMLR, 2021.

Flow-based prior on digits dataset[1]

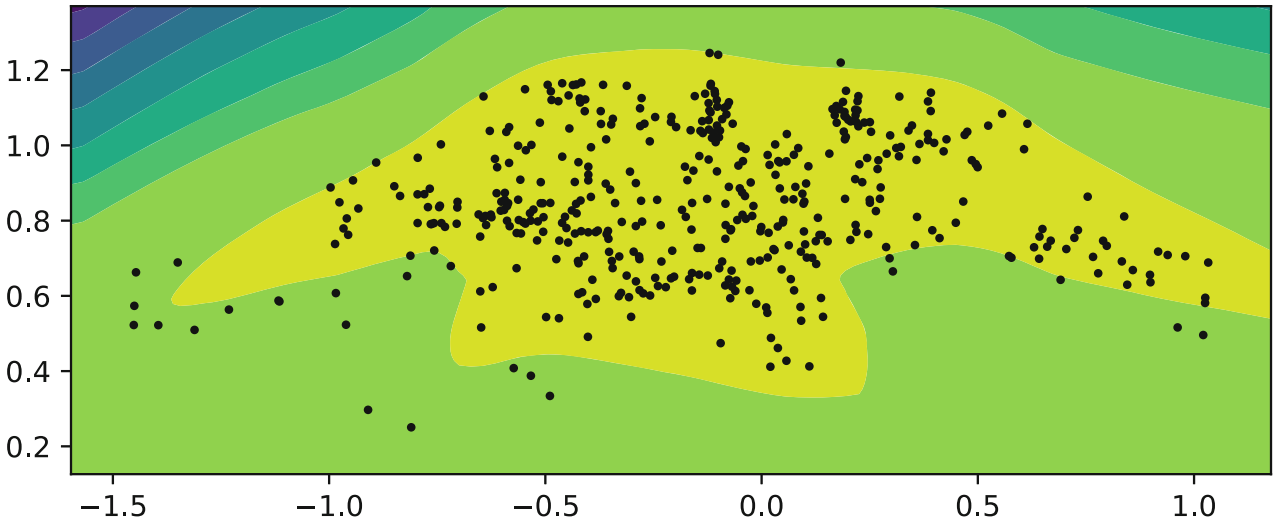
Standard Gaussian prior



MoG prior



RealNVP prior



[1]Tomczak JM. Deep generative modeling. Springer, 2022.

Take-away!

- Flow are models for density estimation
  - With exact likelihood
  - Can approximate any distribution
  - However, layers are simple, so many are required (c.f., Glow)
- We can use them for
  - Flexible posteriors in variational inference
  - Flexible priors in VAEs
  - ... and more

- In a VAE, it is preferable to use a flow prior

Flows vs VAEs[1]

Flows

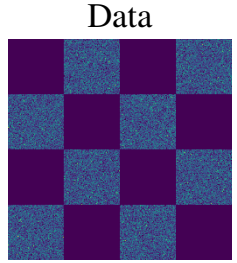
- Exact likelihood
- No **dimensionality reduction**
- Don't model **discrete** data or **discrete nature** data (well)

Performs a bijective transformation of the prior

VAEs

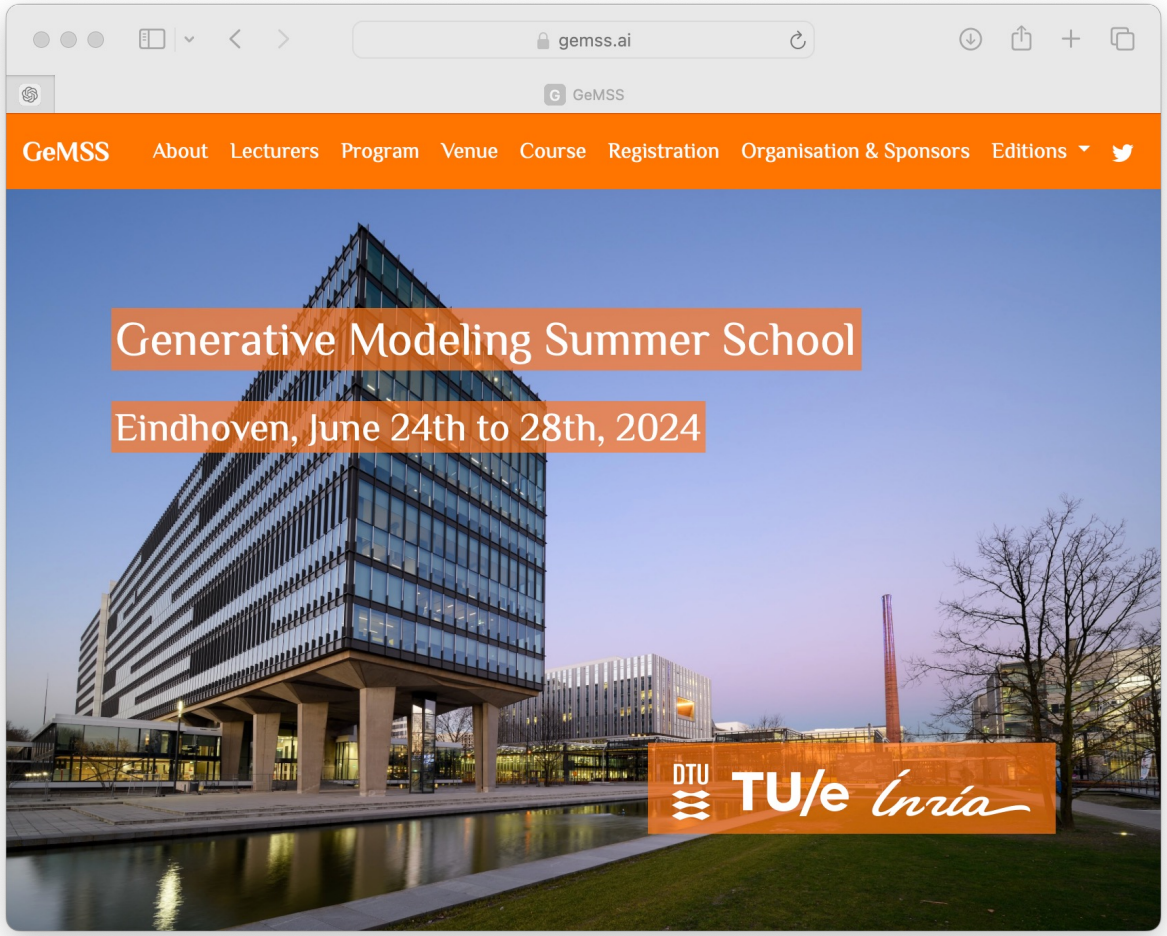
- Approximate likelihood
- Dimensionality reduction
- Can model model **discrete** data or **discrete nature** data

Performs a stochastic transformation of the prior



[1]Nielsen D, Jaini P, Hoogeboom E, Winther O, Welling M. SurVAE Flows: Surjections to Bridge the Gap between VAEs and Flows. NeurIPS, 2020.

Want to learn more about DGMs?



https://gemss.ai

I am looking for a postdoc working on DGMs for physics

Exercise 2: Flows

- Jupiter notebook with a partial Flow implementation
  - Runs in Google Colab
- Do exercise 1.2
  - Exercise 1.2: Implement a masked coupling layer
- Available at:

`https://github.com/frellsen/ProbAI-2024`

Thank you!