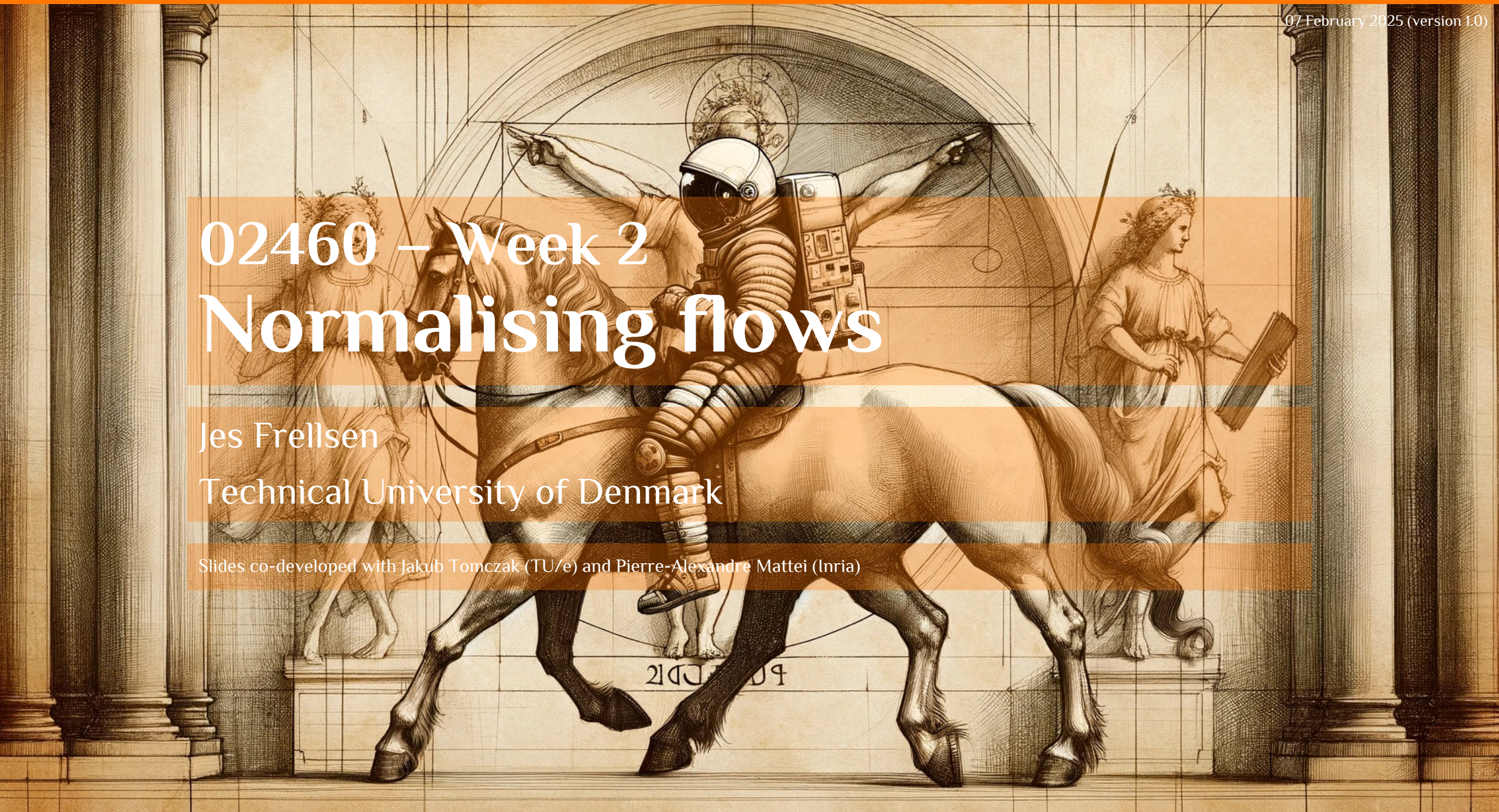
07 February 2025 (version 1.0)



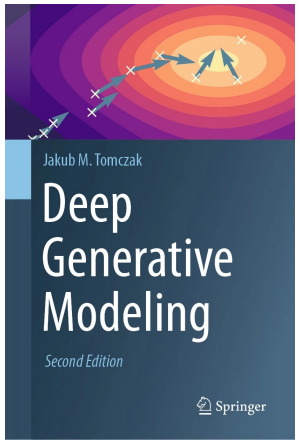
02460 – Week 2
Normalising flows

Jes Frellsen

Technical University of Denmark

Slides co-developed with Jakub Tomczak (TU/e) and Pierre-Alexandre Mattei (Inria)

Menu of the day



Module 1: Generative models

- Week 1: DLVMs
- Week 2: Flows
- Week 3: DDPMs
- Week 4: Mini-project 1

Lecture

- Intro to Flows
- Flow priors in VAEs
  - Chapter 3
  - Sec. 4.4.1.6 and intro of 4.4.2

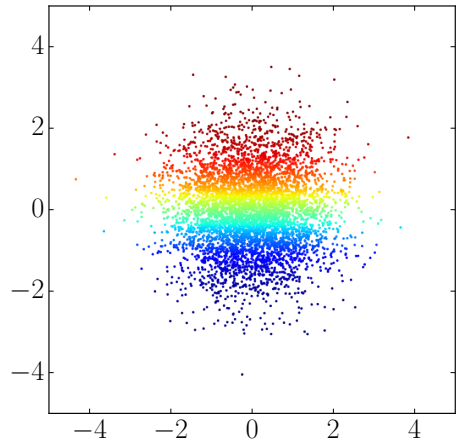
Exercises

- **Theoretical**: Invertible transformations and Jacobians
- **Programming**: Implement the masked coupling layer and flow prior for a VAE

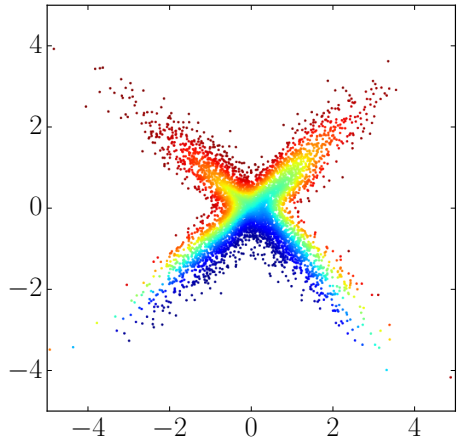
Flow-based models

How is that different from a VAE?

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



$T(\mathbf{u})$



Generative process is

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

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

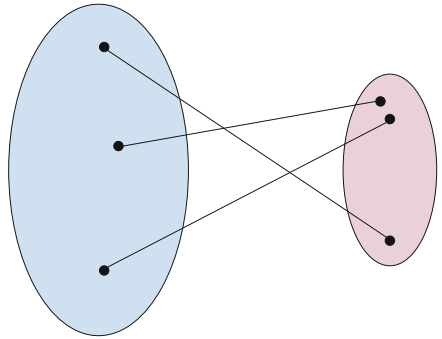
- If we want to calculate the density of $\mathbf{x}$, we need to use the **change-of-variable formular** for probability densities
- This requires f to **invertible** (bijection) and **differentiable**

VAE generative process

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

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

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

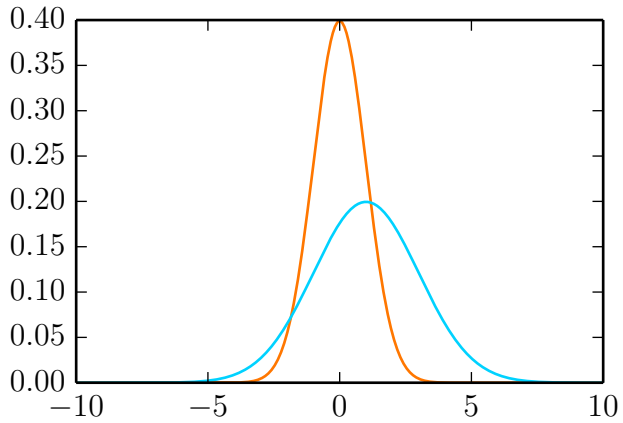


In VAEs $\dim \mathbf{z} < \dim \mathbf{x}$, hence f not bijective

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

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

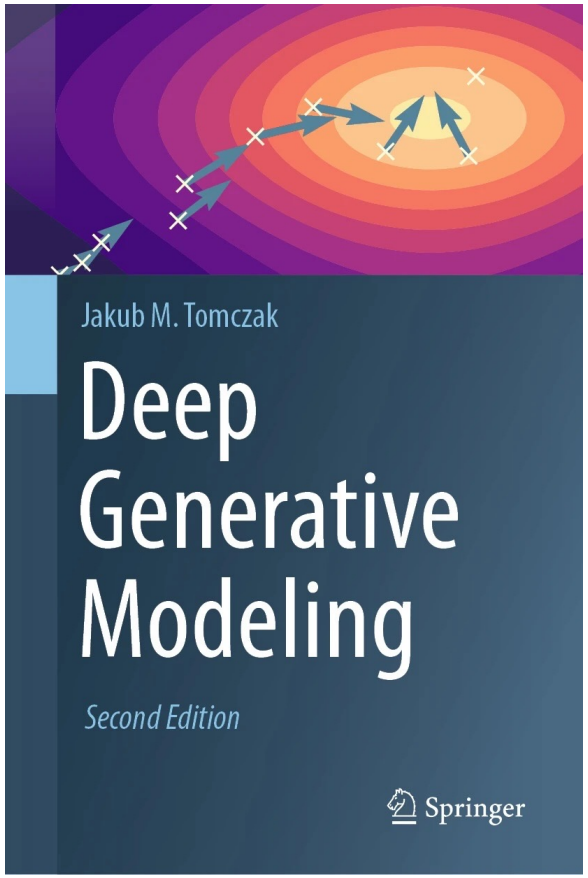
$$T^{-1}(x) = \frac{x-1}{2} \quad \text{and} \quad \left|\frac{d}{dx}T^{-1}(x)\right| = \frac{1}{2}$$

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

Lecture outline

- Introduction to **normalising flows**
- **Coupling layers** and **permutation layers** (RealNVP)
- **RealNVP** and **Glow**
- Flows as **VAE priors**

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"

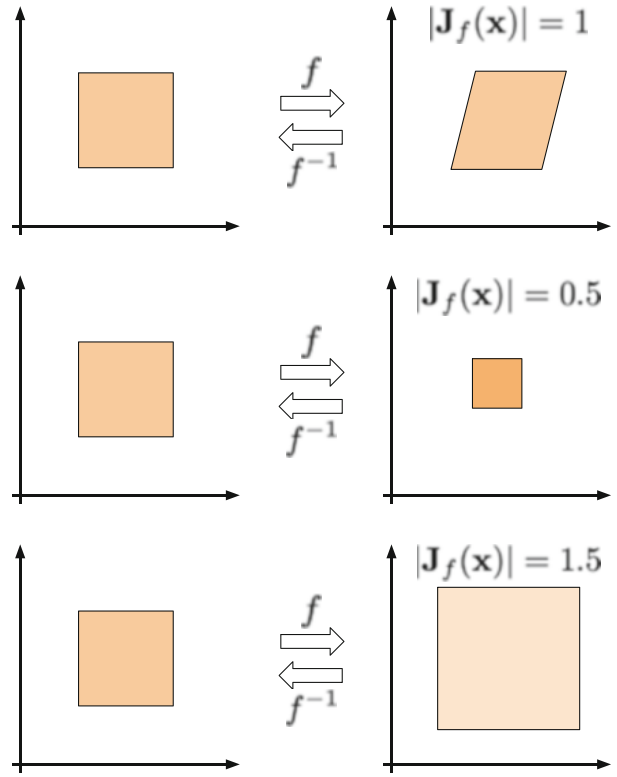


Figure from Tomczak JM. Deep generative modeling. Springer, 2022.

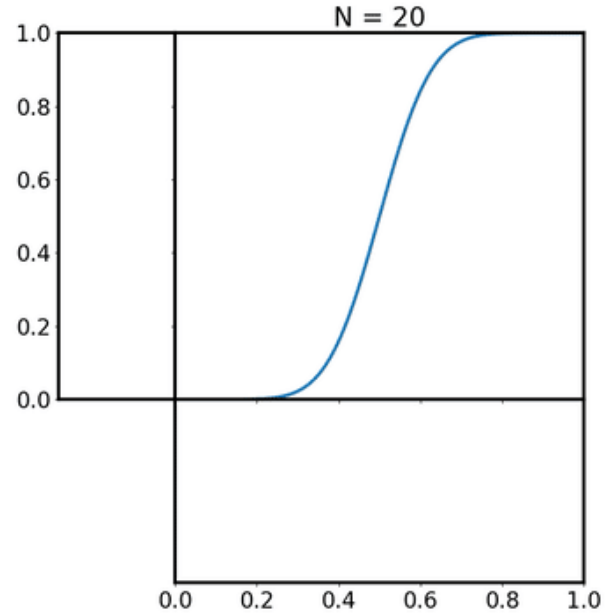
Another simple (well-known) example

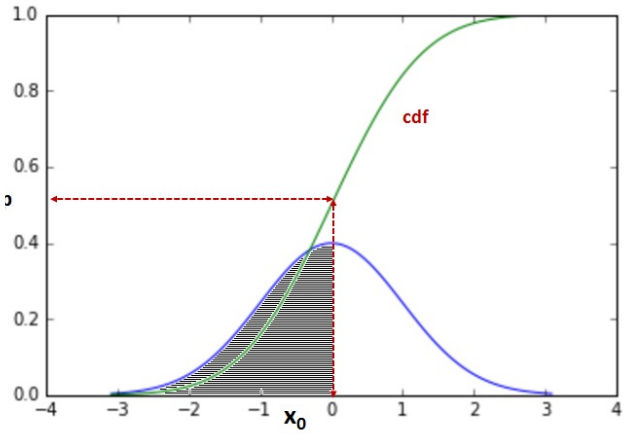
- Recall **inverse transform sampling:**
  - Procedure from for sampling from $p(x)$
  - $p(x)$ is **any* continuous univariate probability distribution**
  - Utilize the cumulative distribution function (CDF): $F(x) = \int_{-\infty}^{x} p(x')\,\mathrm{d}x'$
- We obtain samples x from $p(x)$ by the procedure

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

- This is a flow with $p(x) = U(0,1)$ and $T(u) = F^{-1}(u)$!
- Assuming CDF is invariable, 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**

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

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

- 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

You will show this in an exercise

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

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

Composition is also invertible

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

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

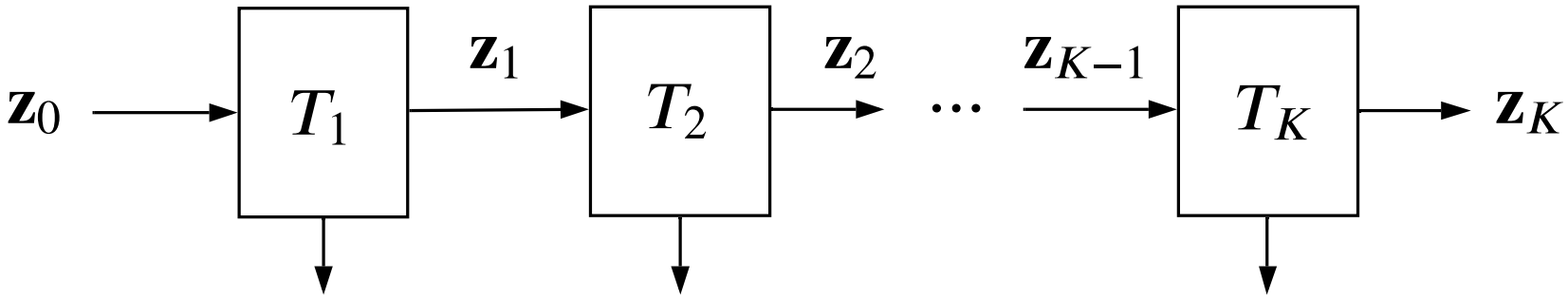
$$\mathbf{z}_{k-1} = T_k^{-1}(\mathbf{z}_k) \quad \text{for } k = K, \dots, 1$$

- ... and the log Jacobian-determinant

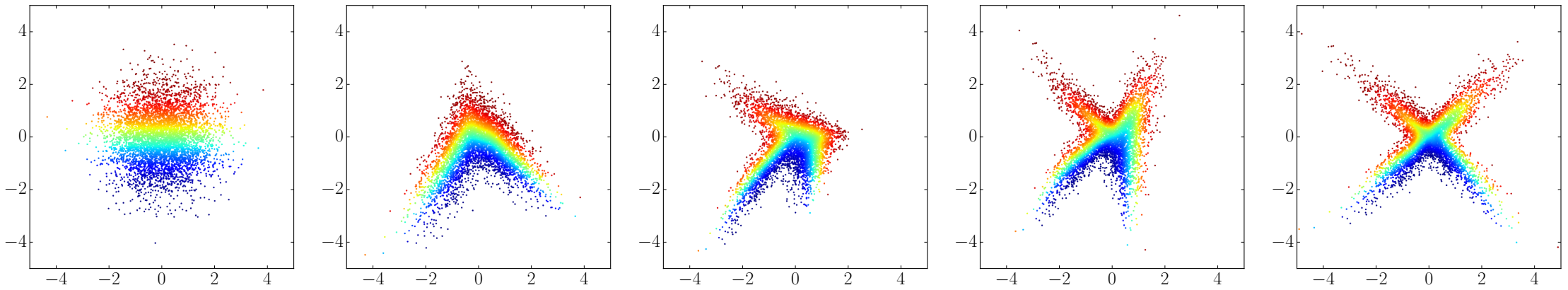
$$\log|\det J_T(\mathbf{x})| = \log\left|\prod_{k=1}^{K} \det J_{T_k}(\mathbf{x})\right| = \sum_{k=1}^{K} \log|\det J_{T_k}(\mathbf{x})|$$

Chain rule for Jacobian

Composable transformation



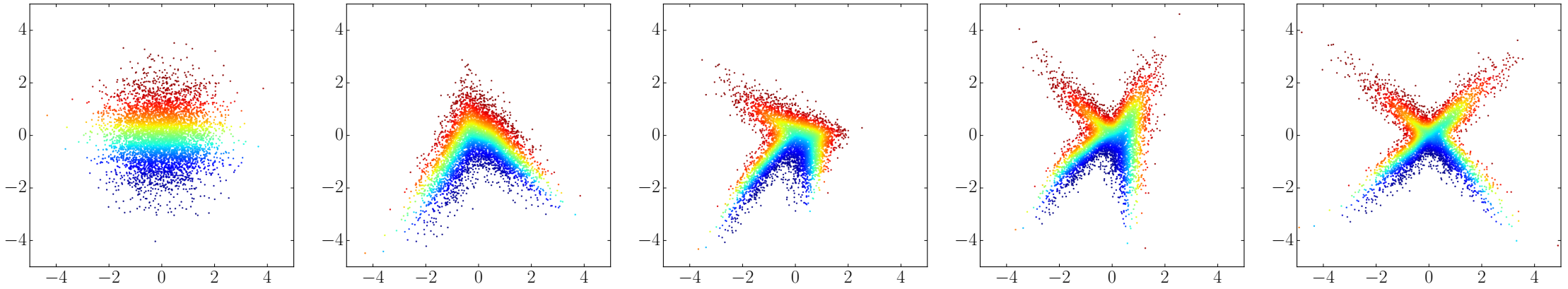
$$\log|\det J_{T_1}(\mathbf{z}_0)| + \log|\det J_{T_2}(\mathbf{z}_1)| + \cdots + \log|\det J_{T_K}(\mathbf{z}_{K-1})| = \log|\det J_T(\mathbf{z}_0)|$$



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.

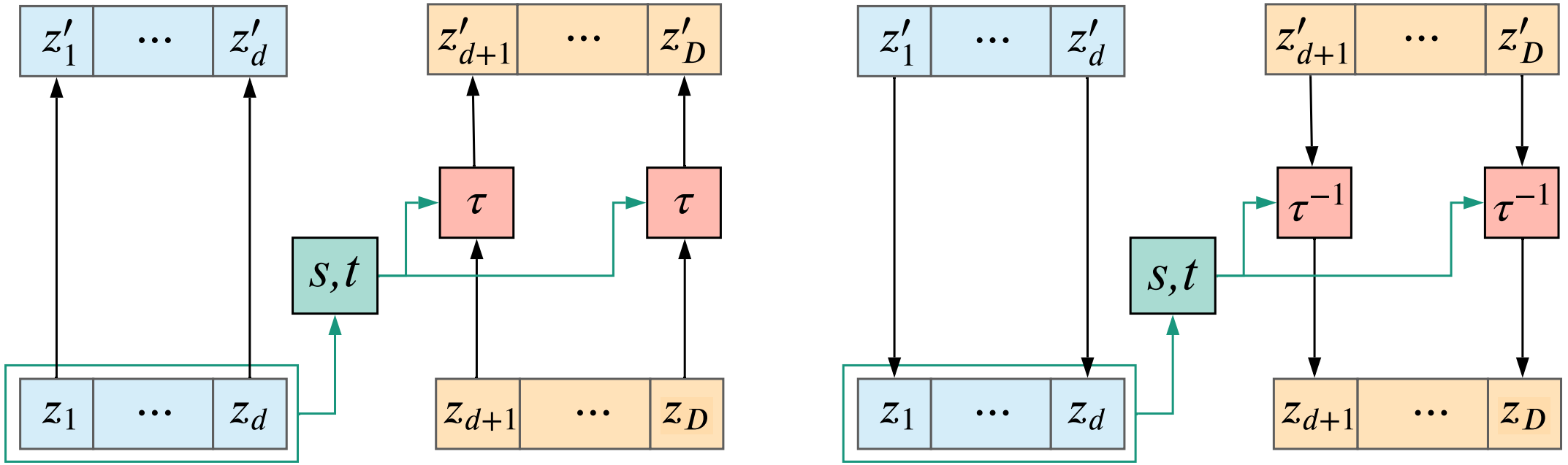
How to construct the transformations?

- 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

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

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

- **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 an affine transformation of $\mathbf{z}_{>d}$, its easily invertible

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

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

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

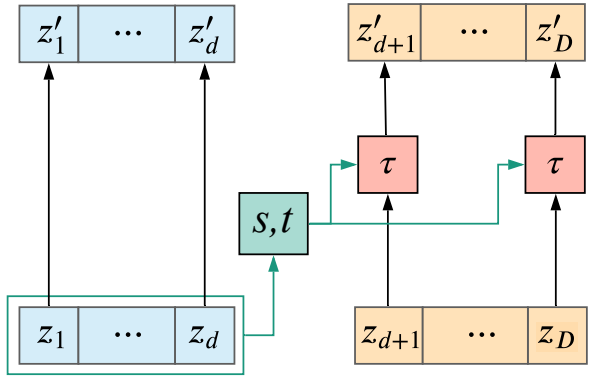
$$\mathbf{z}'_{>d} - t(\mathbf{z}_{\leq d}) = \exp\big(s(\mathbf{z}_{\leq d})\big) \odot \mathbf{z}_{>d}$$

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

[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



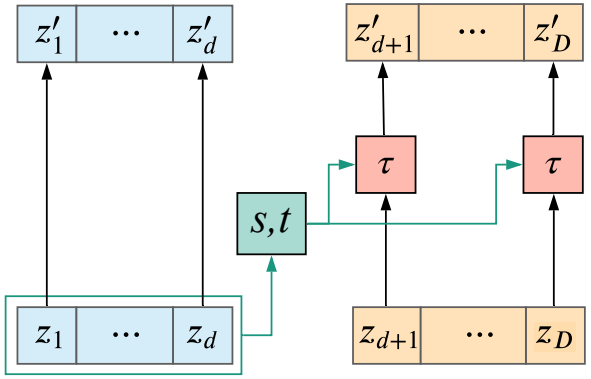
Forward

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

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

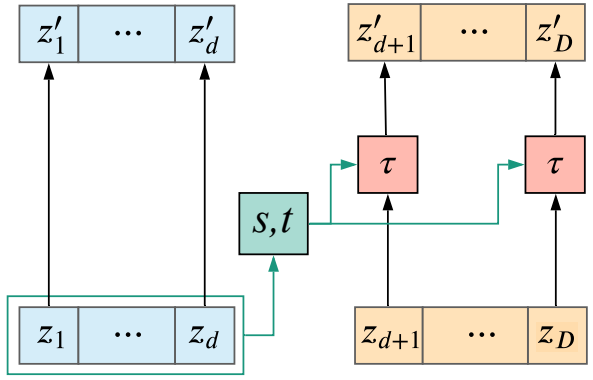
$$\frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{>d}} = \begin{bmatrix} \frac{\partial z'_1}{\partial z_{d+1}} & \cdots & \frac{\partial z'_1}{\partial z_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial z'_d}{\partial z_{d+1}} & \cdots & \frac{\partial z'_d}{\partial z_D} \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} = 0_{d \times (D-d)}$$

Affine coupling layers: the Jacobian (3)

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

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

Q: What is this block of the Jacobian?



Forward

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

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

Affine coupling layers: the Jacobian (3)

- This mean that

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

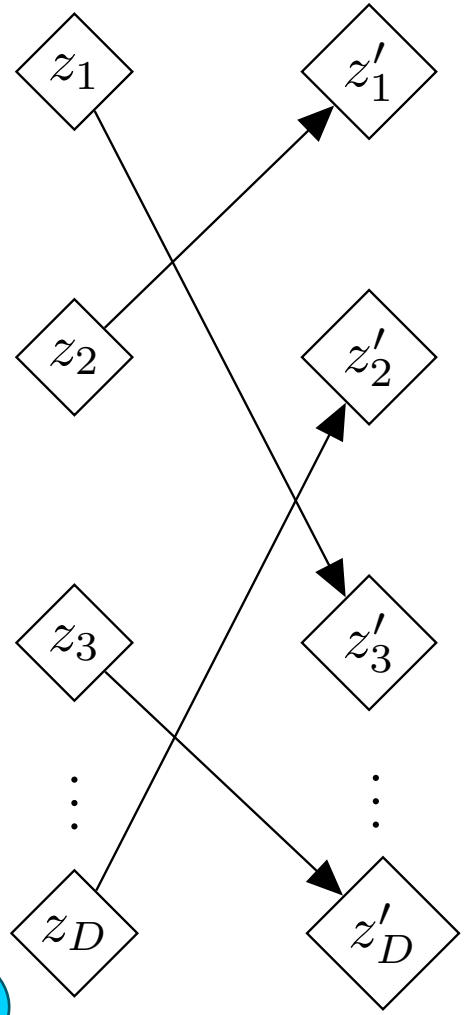
- Since $J_T(\mathbf{z})$ is **triangular**, we have

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

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., $|\det J_T(\mathbf{z})| = 1$



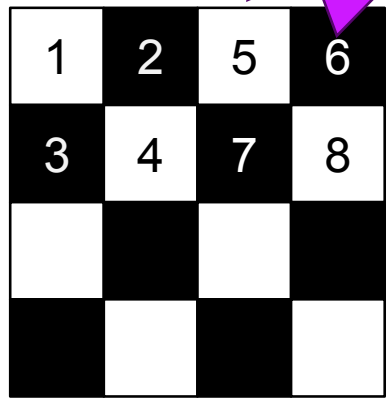
Q: What is $|\det J_T(\mathbf{u})|$ for a permutation?

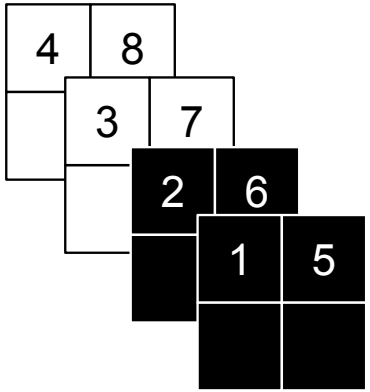
Hint: Recall, if we swap two rows (or columns) in a matrix, the determinant change sign.

RealNVP[1]

You will implement this today!

- 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



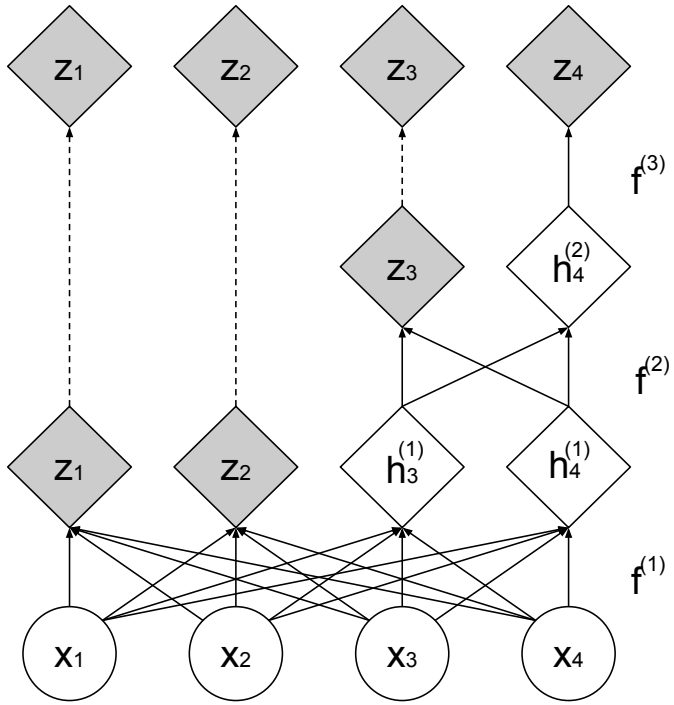


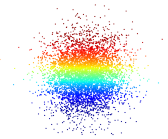
[1]Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP. ICLR, 2017. Figures from [1].

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) — details
  - In a **later layer** (coarser scale) — abstraction

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





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

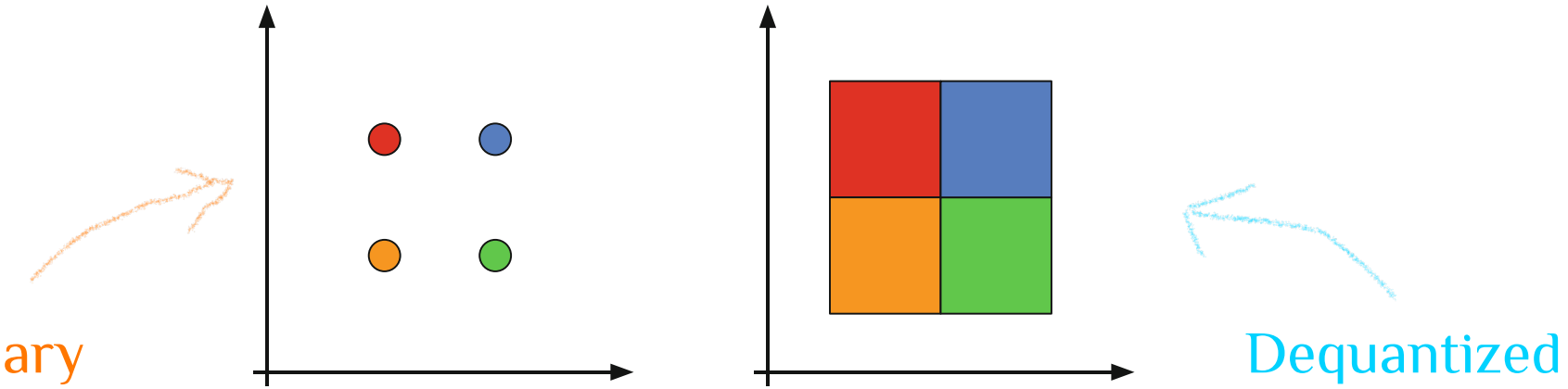
Dequantization

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 $\tilde{\mathbf{x}}$

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



Two binary variables

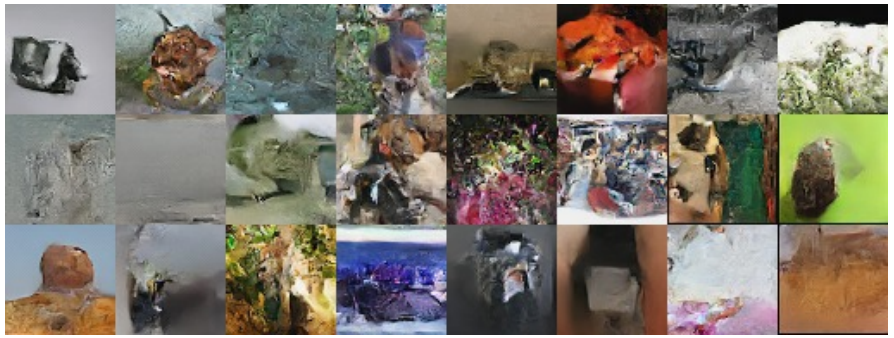
Dequantized density

RealNVP[1]

Data

Samples

Imagenet (64x64)



CelebA

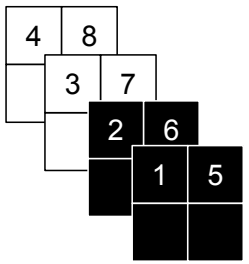
LSUN (bedroom)

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

Glow: Generative flow with 1×1 convolutions[1]

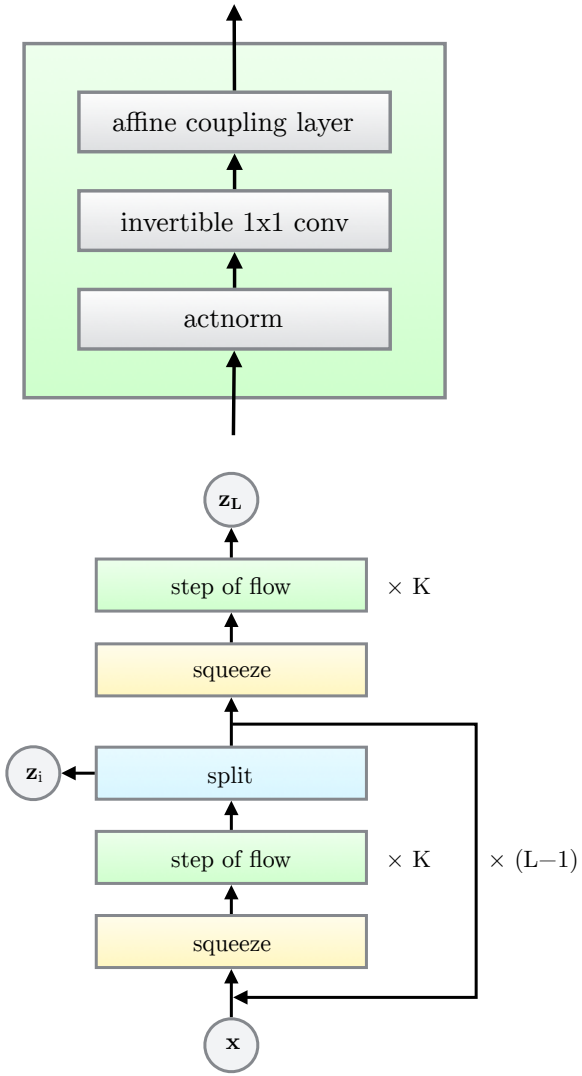
Glow: an evolution of RealNVP, where each flow step has:

- **Actnorm** (a simplified batch norm)
  - Per channel learned scale and bias layer
  - Initialized such that post-actnorm activations per-channel have zero mean and unit variance on a mini batch
- **Affine coupling layers**
  - Only split: channels dimension into two halves
- **Invertible 1 × 1 convolution**
  - A channel wise linear transformation
  - A "generalization" of a permutaion
  - Invertibility ensured by $W = PLU$ parametrisation



Use a multi scale architecture as RealNVP

- For CelebA: $L = 6$ and $K = 32$ (192 flow steps!)



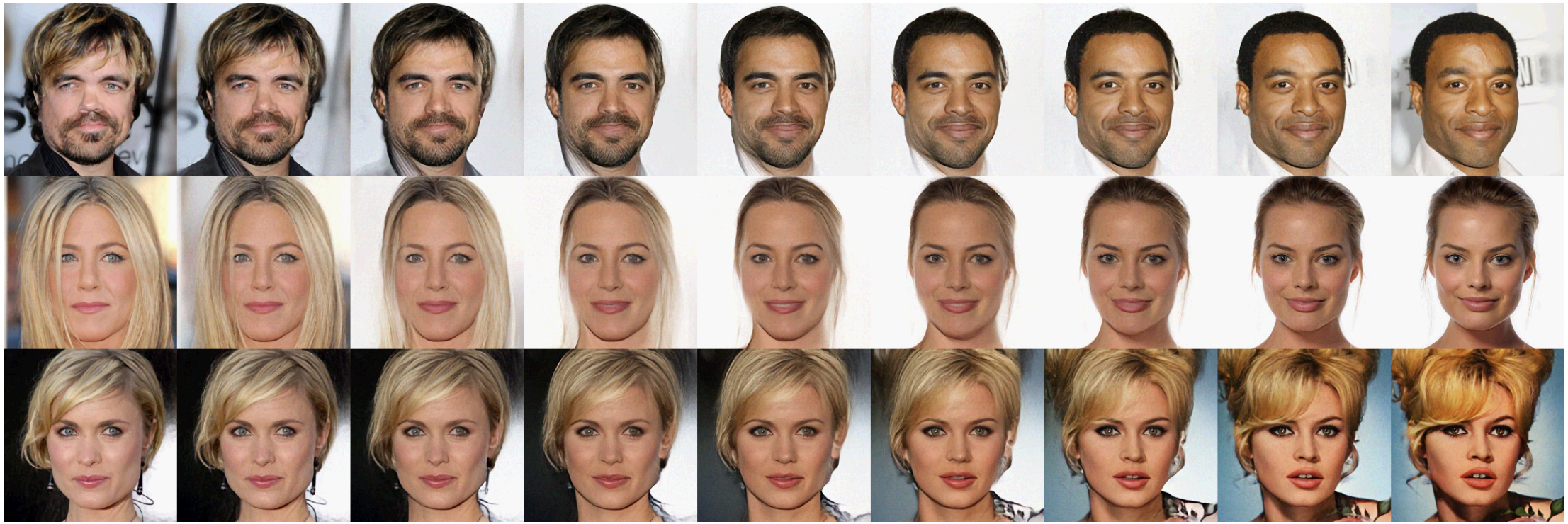
[1] Kingma DP, Dhariwal P. Glow: generative flow with invertible 1× 1 convolutions. NeurIPS, 2018.

Glow: samples from model on CelebA HQ[1]

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

Glow: Linear interpolation in "latent space" between real images[1]

Q: Why the quotation marks?



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

Classes of flows (1)[1]

- **Linear flows**

$$\mathbf{z}' = W\mathbf{z}$$

  - Main challenge **parametrizing W** so it's **invertible**
  - **Permutation** is a special case

- **Autoregressive flows**

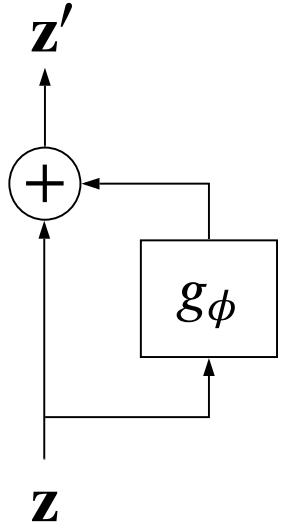
$$z_i' = \tau(z_i; \boldsymbol{h}_i) \quad \text{where} \quad \boldsymbol{h}_i = c_i(z_{<i})$$

  where **τ is strictly monotonic** (inevitable) and **c_i is any neural net**

  - **Jacobian** is **triangular** and **tractable**
  - **Affine coupling layers** is a special case

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

Classes of flows (2)[1]



- **Residual flows**

$$\mathbf{z}' = \mathbf{z} + g_\phi(\mathbf{z})$$

Invertible if $g_\phi(\mathbf{z})$ is **contractive**, i.e.,

$$\exists L < 1, \forall z, \tilde{z} \in \mathbb{R}^d : \delta\big(F(z), F(\tilde{z})\big) \leq L\delta(z, \tilde{z})$$

where δ is a **distance function**

- A contractive map brings inputs closer together
- **Contractive**: **Lipschitz continuous**, with the additional constraint $L < 1$
  - **Composition** of K Lipschitz functions **are Lipschitz** with $L = \prod_{k=1}^{K} L_k$
  - Standard **nonlinearities are Lipschitz** with $L \leq 1$, e.g., logistic sigmoid and ReLU
  - **Linear layer** can be made **contrastive** using **spectral normalization**
- **The Jacobian is difficult** (autodiff or power series approximation)

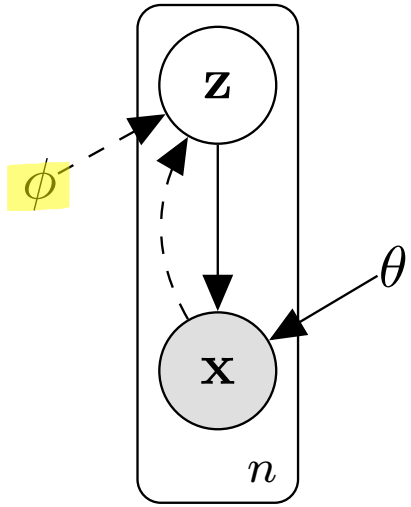
[1]Papamakarios G, Nalisnick E, Rezende DJ, Mohamed S, Lakshminarayanan B. Normalizing Flows for Probabilistic Modeling and Inference. JMLR, 2021.

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.

Let's revisit the VAE and ELBO



likelihood prior

- Recall the model $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})$
- Learn model by optimizing ELBO wrt. θ, ϕ

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

tractable objective

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})]$$

variational distribution

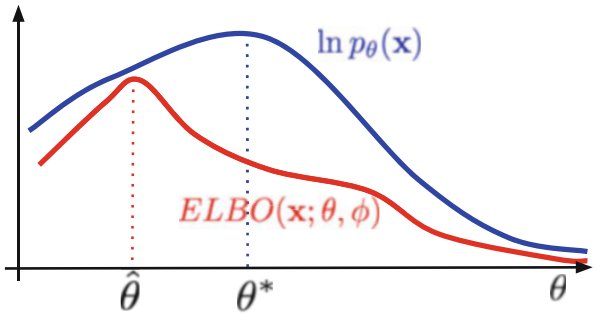
$$= \log p_\theta(\mathbf{x}) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x}))$$

gap

- Gradients are calculated from the reparameterized ELBO

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0},\mathbf{I}_M)}[\log p_\theta(\mathbf{x}|\boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon})] - \mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})]$$

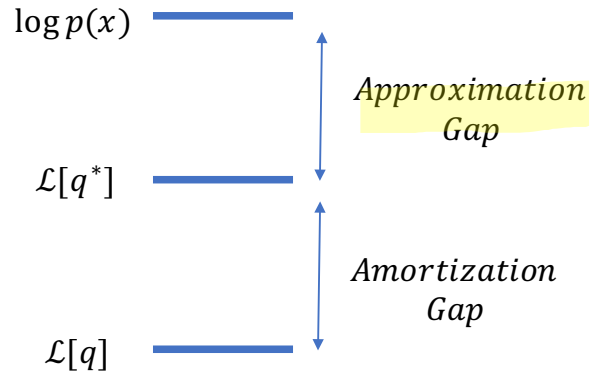
Inference suboptimality in VAEs[1]



- We can write the ELBO $\mathcal{L}$ as

$$\log p_\theta(\mathbf{x}) = \mathcal{L}(\theta,\phi) + \overbrace{\mathrm{KL}\big(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(z|x)\big)}^{\geq 0}$$

- The KL-term is the gap between ELBO and true log-likelihood
  - Approximation gap:
    tight when the true posterior belongs to the variational family
  - Amortisation gap:
    tight when encoder has large capacity
- We can tighten approximation gap with a more expressive variational family!



We can use a flow as the **variational distribution** in a VAE

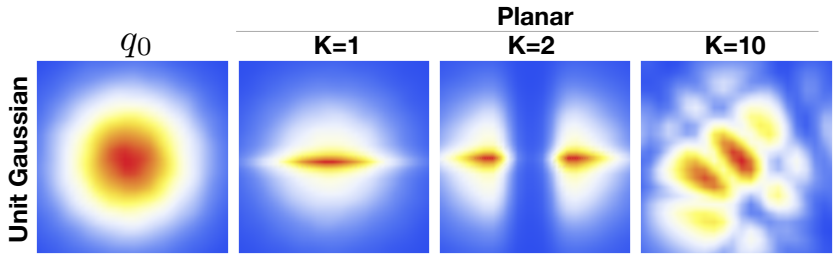
Figures from 1,2
[1]Cremer C, Li X, Duvenaud D. Inference suboptimality in variational autoencoders. ICML, 2018.
[2]Tomczak JM. Deep generative modeling. Springer, 2022.

Variational inference with flows[1]

- Rezende and Mohamed (2015)[1] use a **residual flow**, called planar

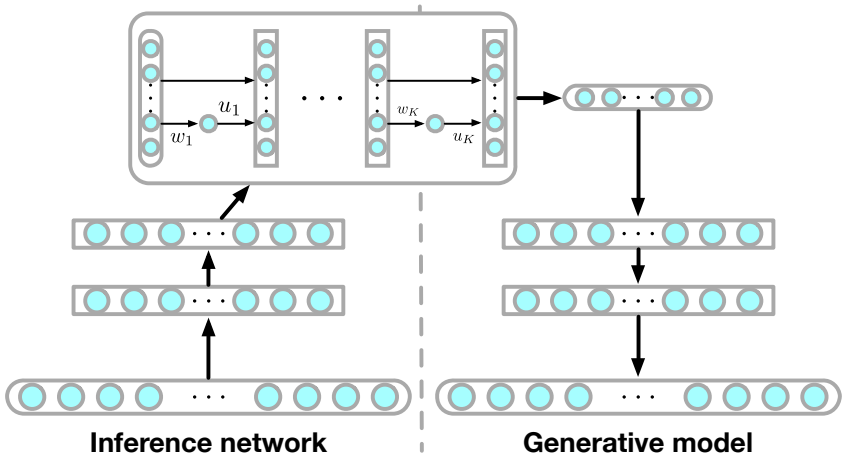
$$T_k(\mathbf{u}) = \mathbf{u} + \mathbf{v}\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{u} + b\right)$$

  - σ is an activation function and invertible if $\mathbf{w}^{\mathrm{T}}\mathbf{v} > -\left(\sup_x \sigma(x)\right)^{-1}$



Test set ELBO on binarized MNSIT using k transformation

Model	$-\ln p(\mathbf{x})$
DLGM diagonal covariance	$\leq$ 89.9
DLGM+NF (k = 10)	$\leq$ 87.5
DLGM+NF (k = 20)	$\leq$ 86.5
DLGM+NF (k = 40)	$\leq$ 85.7
DLGM+NF (k = 80)	$\leq$ 85.1



Figures from 1

[1]Rezende D, Mohamed S. Variational inference with normalizing flows. ICML, 2015.

Recall: Improving the prior of a VAE

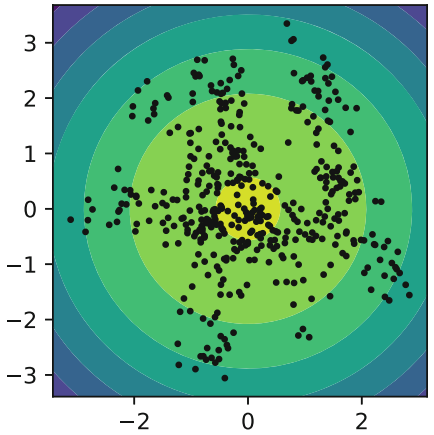
$$\mathcal{L}(\theta,\phi) = \overbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}^{\text{reconstruction term}} - \overbrace{\mathrm{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\lambda(\mathbf{z})\right]}^{\text{regularisation term}}$$

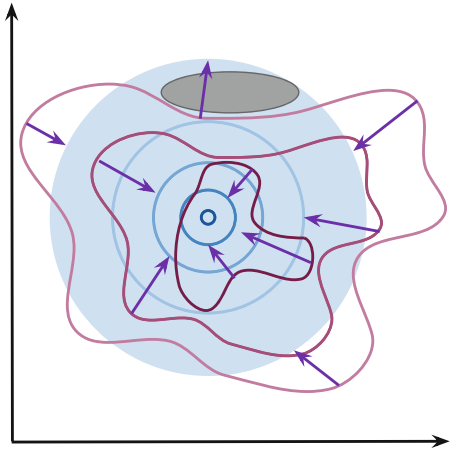
- We can rewrite the **regularization** term as

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

  where $q_\phi(\mathbf{z}) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[q_\phi(\mathbf{z}|\mathbf{x})\right] \approx \frac{1}{N}\sum_n q_\phi(\mathbf{z}|\mathbf{x}_n)$ 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.

Flow-based prior

- Let's write the ELBO (again) for a learnable prior flow-based prior $p_\lambda(\mathbf{z}) = p_z(\mathbf{z}) = p_u(\mathbf{u})|\det J_T(\mathbf{u})|^{-1}$ and $\mathbf{u} = T^{-1}(\mathbf{z})$:

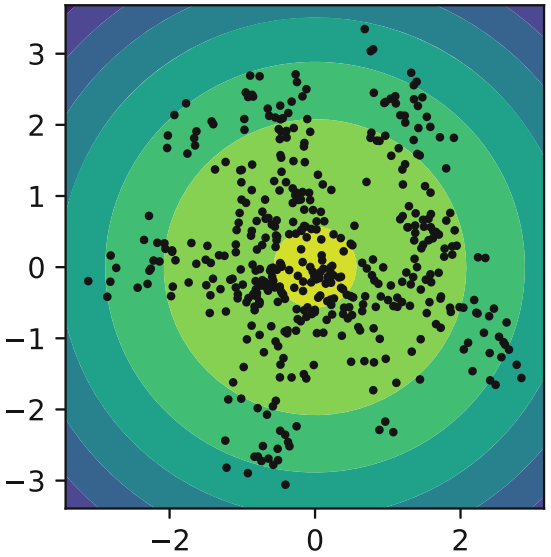
$$
\begin{aligned}
\mathcal{L} &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_{\theta,\lambda}(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\lambda(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log\left(p_u\left(T^{-1}(\mathbf{z})\right)\left|\det J_T\left(T^{-1}(\mathbf{z})\right)\right|^{-1}\right) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) - \log\left|\det J_T\left(T^{-1}(\mathbf{z})\right)\right|\right] + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_u\left(T^{-1}(\mathbf{z})\right)}{q_\phi(\mathbf{z}|\mathbf{x})}\right]
\end{aligned}
$$

- We recover the usual ELBO, we one additional term $-\mathrm{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}) \,\|\, p_u\left(T^{-1}(\mathbf{z})\right)\right)$
- Gradients are straight forward

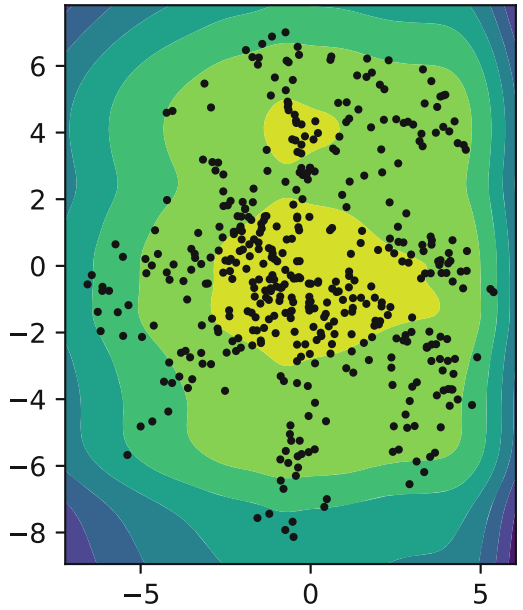
Flow-based prior on digits dataset[1]

Examples from Jakub's book

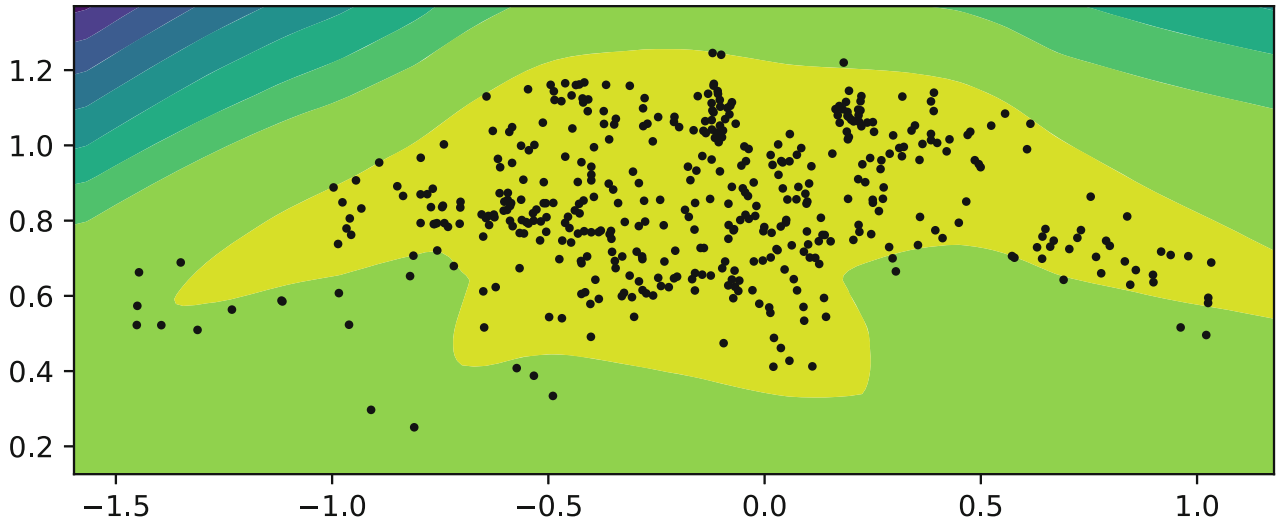
Standard Gaussian prior



MoG prior



RealNVP prior



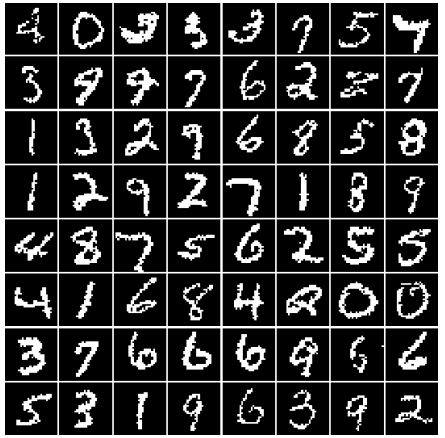
[1]Tomczak JM. Deep generative modeling. Springer, 2022.

To learn prior or variational posterior?

- Flow **prior** and flow **var. posterior** have the same computational cost
- You have a more **expressive model with a flow prior**
  - Therefore, Chen et al. (2017)[1] **argues for using a flow prior**
- Empirical validation[1], using
  - Inverse autoregressive flow (IAF) **variational posterior**
  - Autoregressive flow (AF) **prior**

Table 1: Statically Binarized MNIST

Model	NLL Test
Normalizing flows (Rezende & Mohamed, 2015)	85.10
DRAW (Gregor et al., 2015)	< 80.97
Discrete VAE (Rolfe, 2016)	81.01
PixelRNN (van den Oord et al., 2016a)	79.20
IAF VAE (Kingma et al., 2016)	79.88
AF VAE	79.30
VLAE	**79.03**



(b) Samples from VLAE

[1]Chen X, Kingma DP, Salimans T, Duan Y, Dhariwal P, Schulman J, Sutskever I, Abbeel P. Variational Lossy Autoencoder. ICLR, 2017.

Flows vs VAEs[1]

Flows

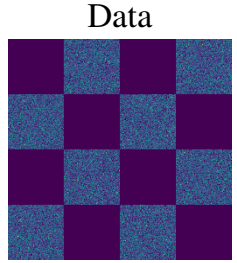
- Exact likelihood
- No **dimensionality reduction**
- Don't model **discrete** data or **discrete nature** data (well)

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.

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**

Thank you!