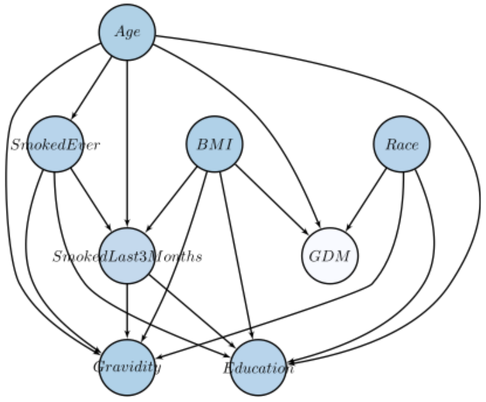
Normalizing flows

Materials adapted from:

- Normalizing Flows and Invertible Neural Networks, ECCV 2020 Tutorial
- Nordic Probabilistic AI School (ProbAI) 2022.

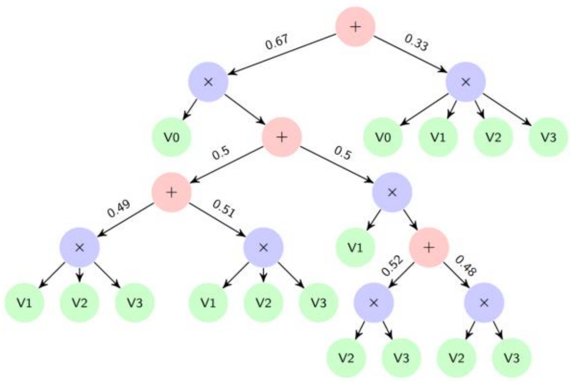
Joint distributions

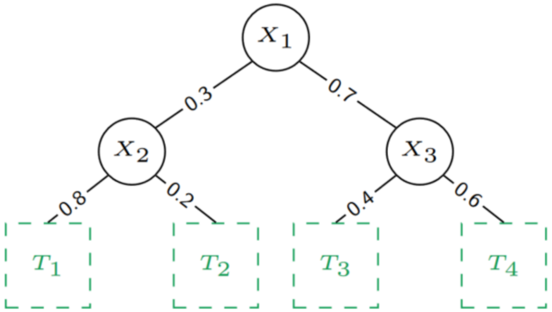
PGMs



Explainable

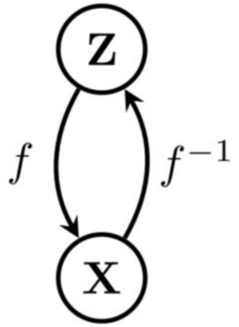
PCs





Tractable

Normalizing flows

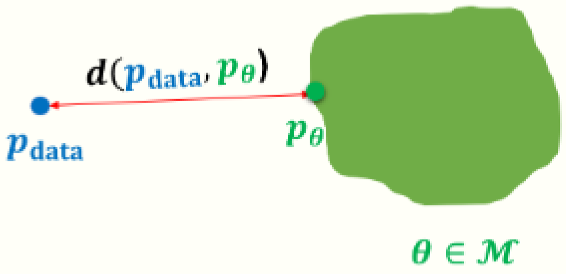


Expressive

Learning Probability Distributions

The curse of dimensionality & continuous spaces

$\mathbf{x}^{(j)} \sim p_{\text{data}}$
$j = 1, 2, \ldots, |\mathcal{D}|$



- Typical image resolution – 700 x 1400
- Each pixel has 3 channels - RGB
- Each channel can take values 0-255

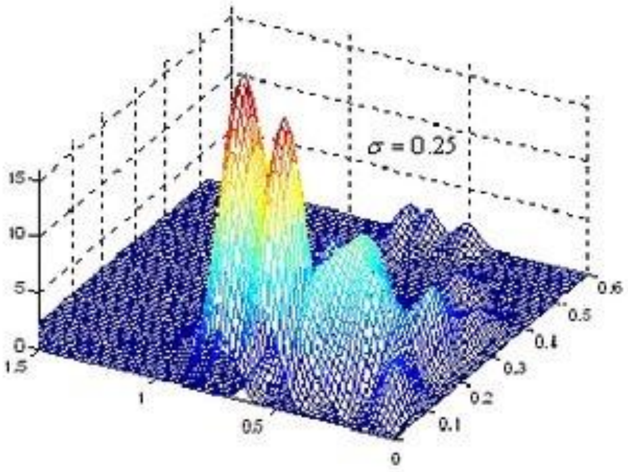
How many possible images?

$$256^{700\times1400\times3} \approx 10^{800000}$$

What if the data was continuous ?

- Infinite no. of values
- Complex shapes for probability densities

We have only a limited amount of data to fit the model



Give up ?

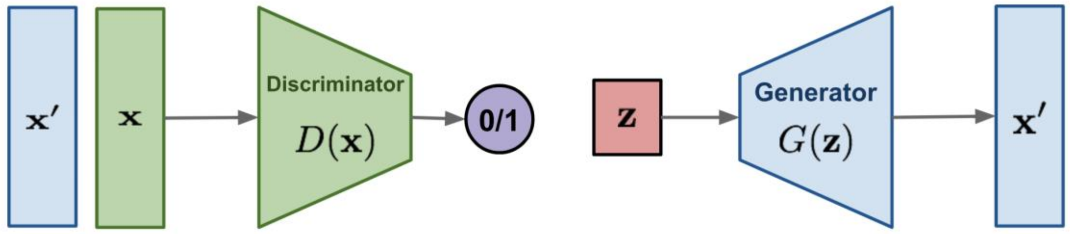
Not yet. Look at these images .. Can you say which of them are real ?

All of them are generated by a model. **None of these people exist in real life !**

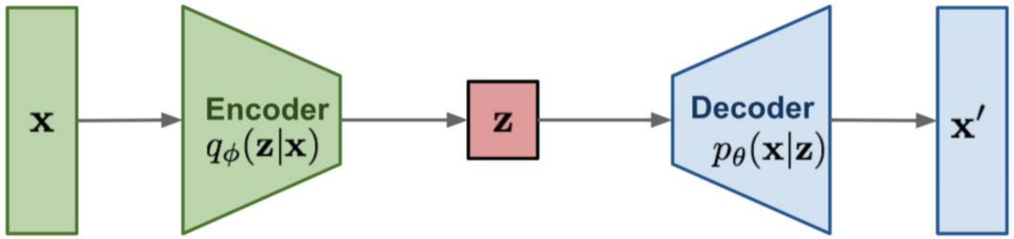
Deep Generative Models

Using deep neural networks to learn probability distributions

- Deep latent variable models
  - Assume a simple distribution over low dimensional latent factors of variation
  - Map to data space using neural networks



Generative Adversarial Networks (GANs)



Variational Auto Encoders (VAEs)

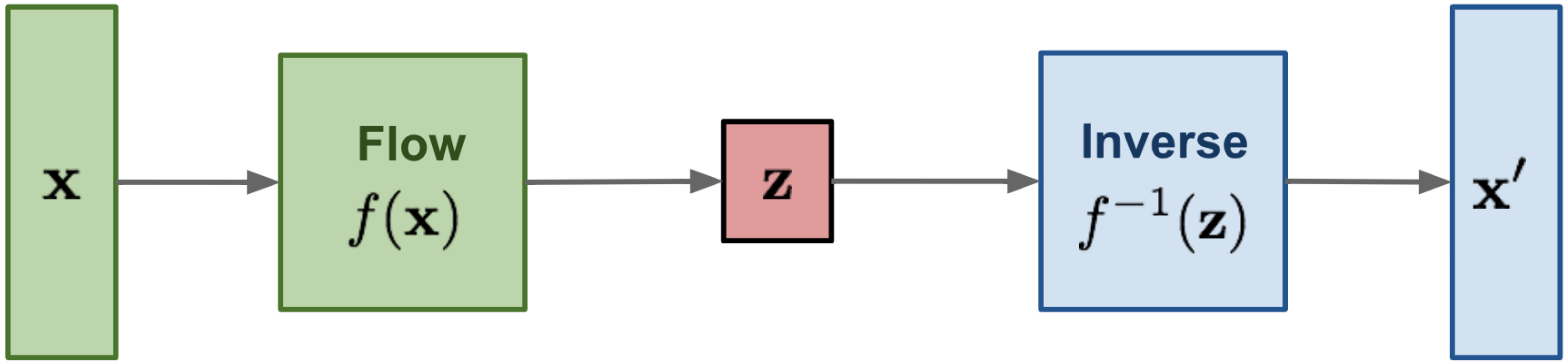
Expressive models, can generate high quality samples

But **cannot** perform exact inference over the modeled probability distribution

Normalizing flows

A new class of deep generative model

- Utilizes invertible transformations



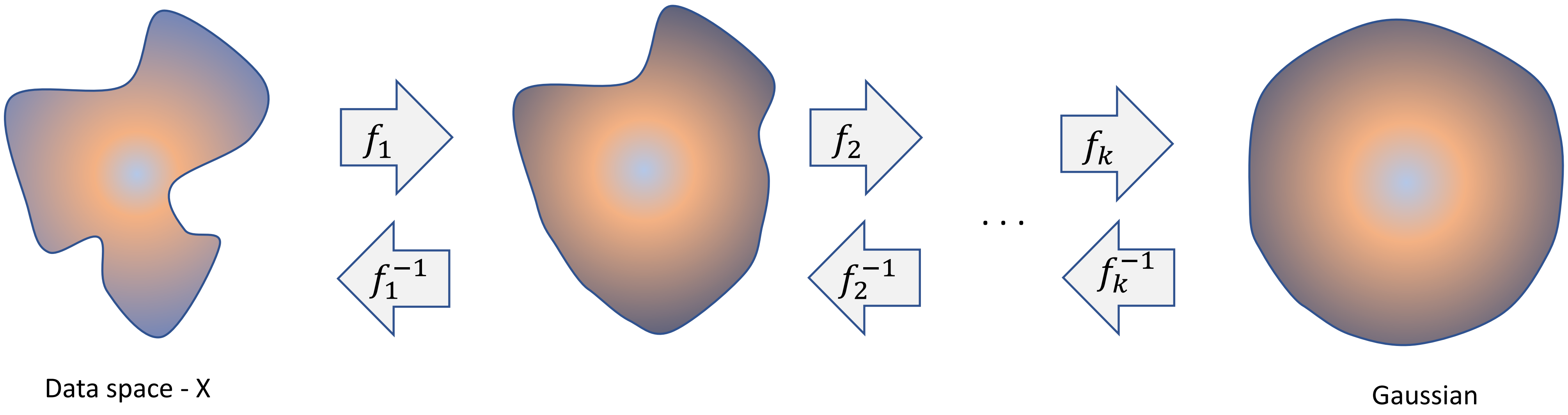
Why are they exciting?

- Expressive models utilizing flexible neural networks
- Can perform exact likelihood evaluation

Some cool results using normalizing flows

Almost as good as GANs / VAEs but better tractability

Normalizing Flows: Overview



Model complex probability distributions by

- flowing a simple distribution through a sequence of invertible transformations

Components

- Simple base distribution – typically a gaussian
- Invertible Transformation – parameterized using neural networks

Normalizing Flows: Formulation

Change of variables: Linking the probability densities in the two spaces

$$p_X(x) = \boxed{p_Z(f(x))}\boxed{\left|\det\left(J_f(x)\right)\right|}$$, where $$J_f(x) = \frac{\partial f(x)}{\partial x}$$

Density in the gaussian space

Volume correction term

Properties

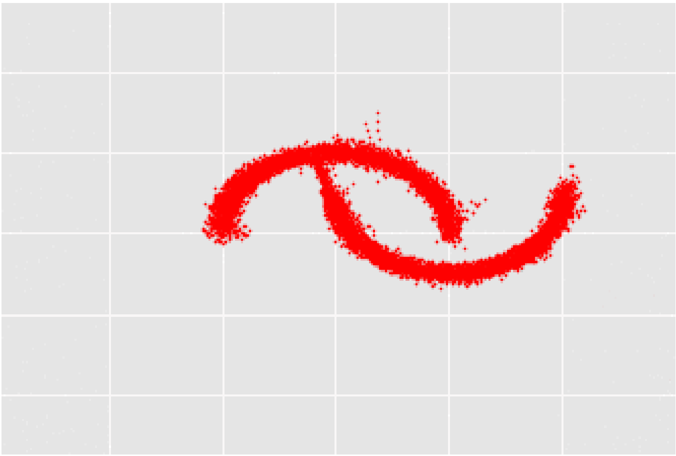
A flow f is a parametric function which is

- Invertible
- Differentiable
- Has efficiently computable Jacobian determinant

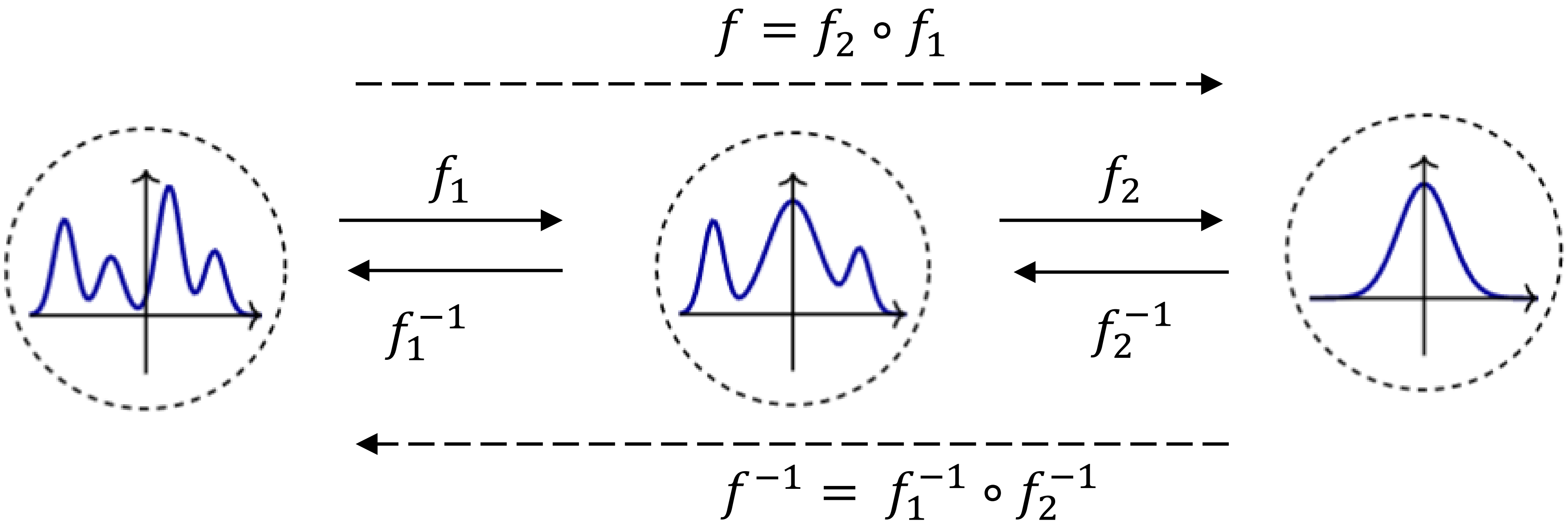
Designing such functions is the core research problem in the field of flows

Why are they called normalizing flows ?

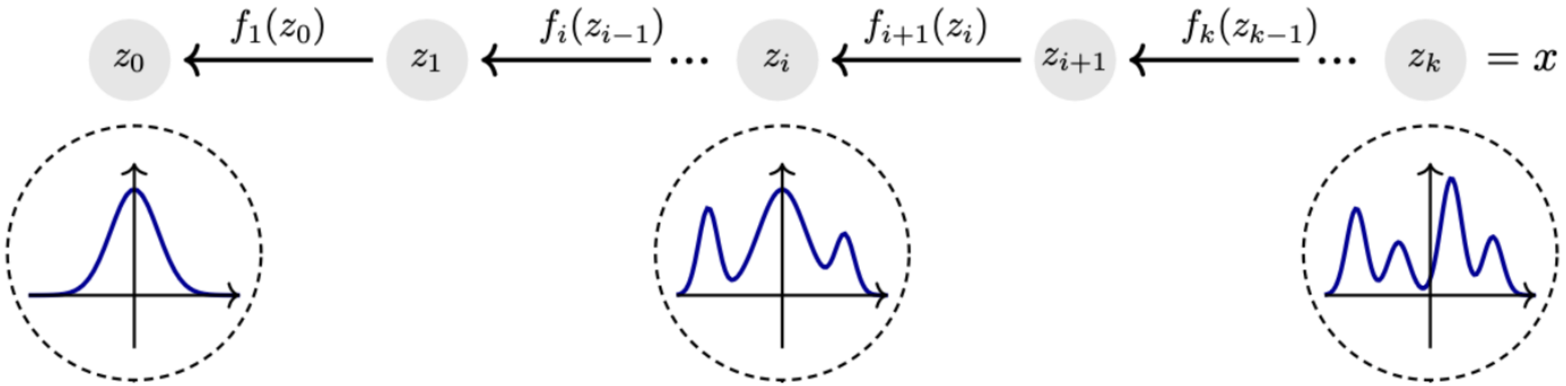
- Composing transformations
  - Invertible differentiable functions are closed under composition



- Can slowly '*flow*' a complex distribution step by step into a normal



How do you train them?



We can compute likelihood exactly

$$p_X(x) = p_Z(f(x))\,|\det(J_f(x))| \quad = p_Z(f_1 \circ f_2 \circ \ldots f_k(x))\,|\det(J_{f_1 \circ f_2 \circ \ldots f_k}(x))|$$

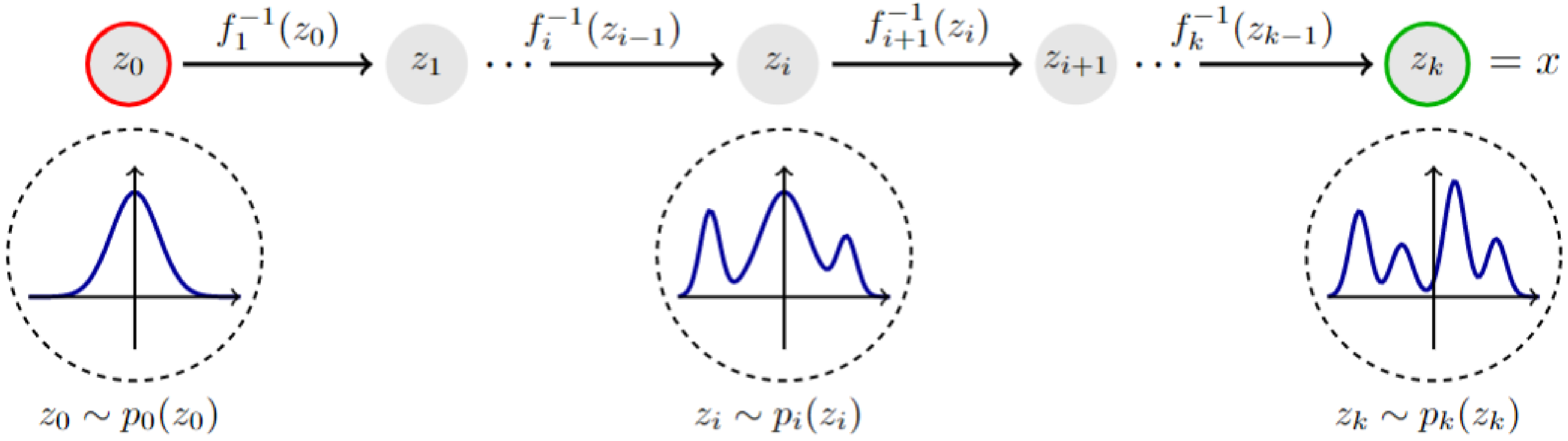
$$= p_Z(f_1 \circ f_2 \circ \ldots f_k(x)) \prod_i |\det(J_{f_i})|$$

Maximize the log-likelihood of the data points w.r.t to the parameters of the flow

$$\arg\max_\theta \sum_m \log p_X(x_m) = \arg\max_\theta \sum_m \log p_Z(f_\theta(x_m)) + \sum_m \sum_i \log\left|\det(J_{f_{\theta_i}})\right|$$

f is differentiable, solve using **gradient descent**

Tractable for what ?



- **Sampling**
  - Sample from gaussian and apply inverse transform
- **Density estimation**
  - Use change of variables formula to compute $p_X(x)$ exactly.

✓ Evidential	✗ Marginal	✗ Conditional	✗ MAP

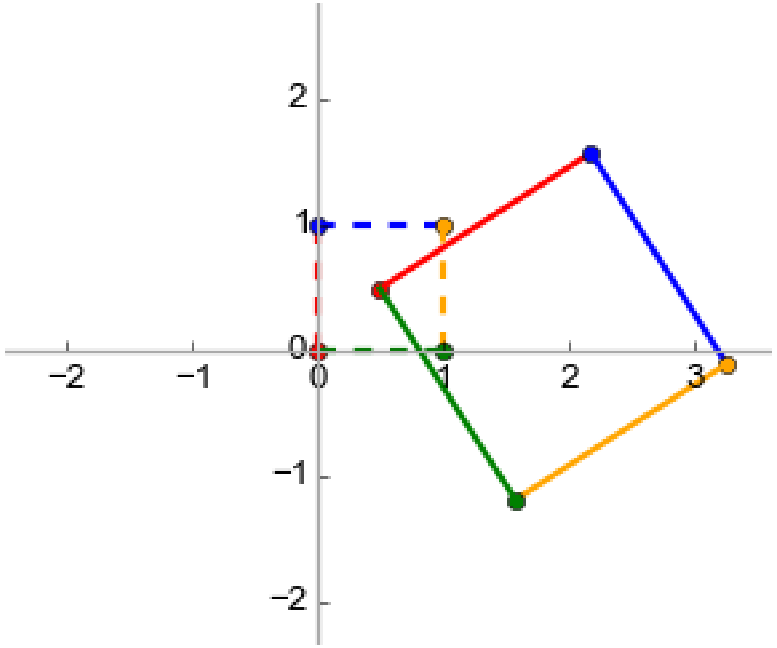
Parameterizing flow transformations

Linear Flows

We can define a flow using a linear transformation if the matrix is invertible

$$f(\boldsymbol{x}) = A\boldsymbol{x} + b$$

- Inverse: $f^{-1}(\boldsymbol{z}) = A^{-1}(\boldsymbol{z} - b)$
- Jacobian is same as A. Thus, $\det J_f = \det A$



Caveats

- Will have to ensure invertibility during learning
- Affine gaussians are gaussians! Not quite expressive.
- Determinant computation is expensive $O(d^3)$

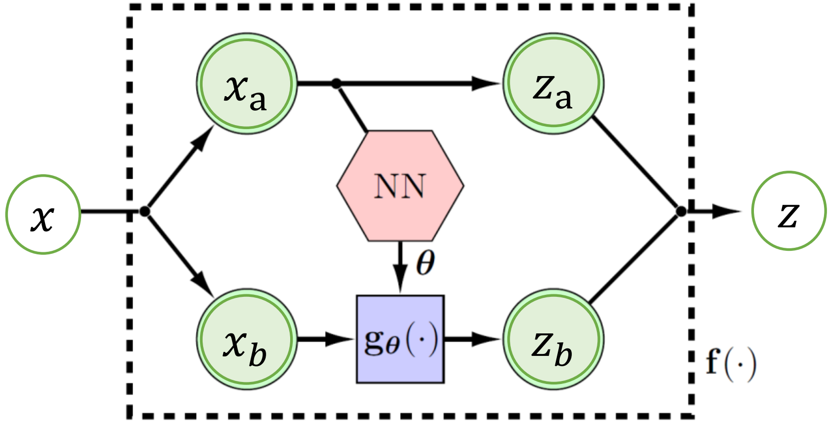
Coupling Flows

A general approach to building more expressive Non-linear flows

Partition the input into two disjoint sets $\boldsymbol{x} = (\boldsymbol{x_A}, \boldsymbol{x_B})$

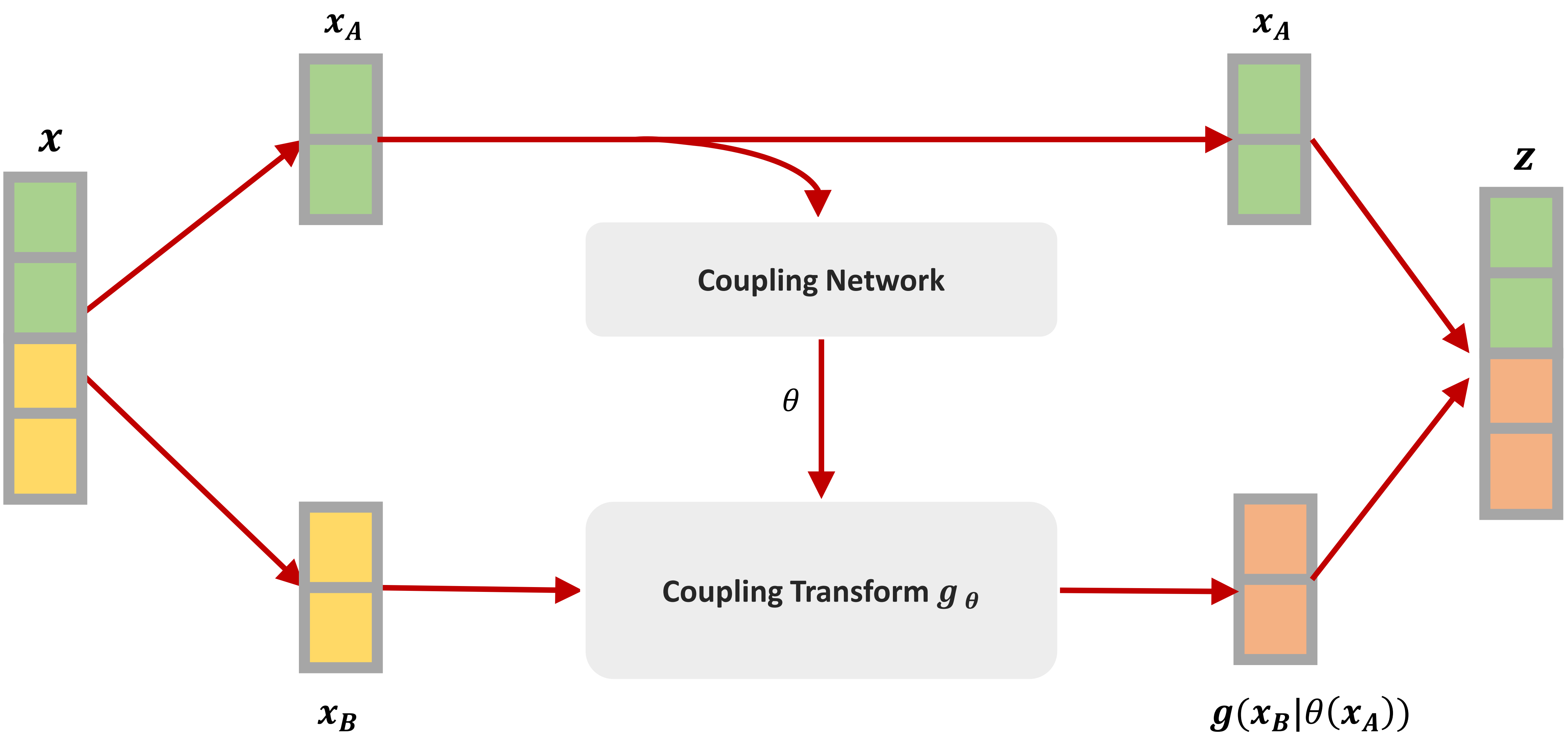
Define $f(\boldsymbol{x}) = (\boldsymbol{x_A}, g(\boldsymbol{x_B}|\theta(\boldsymbol{x_A}))$

g is an invertible transformation whose parameters depend on x_A

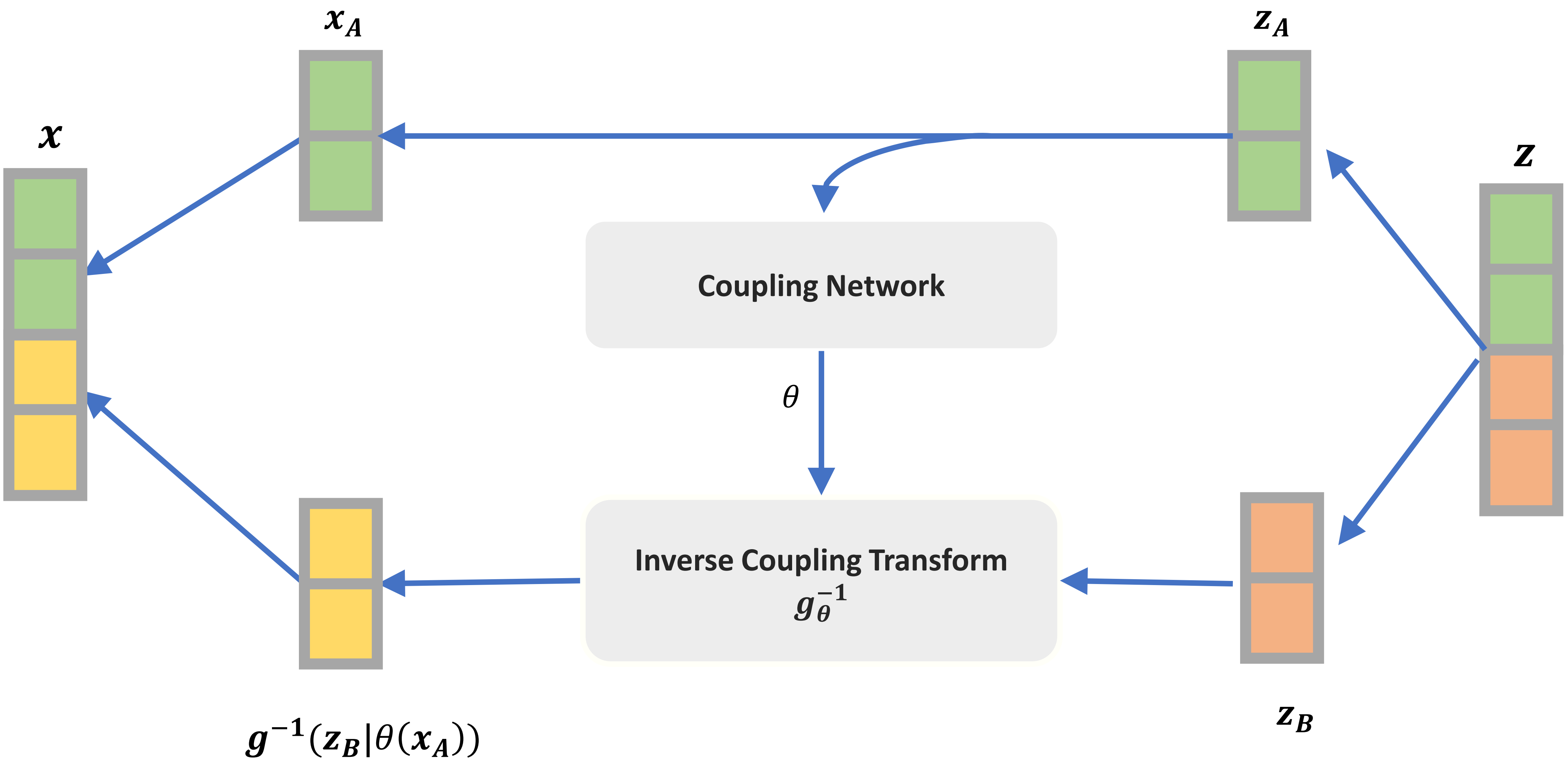


- Will use this as a case study demonstrate the key properties
  - invertibility
  - efficiently Jacobian determinant
- Also implement in hands on part

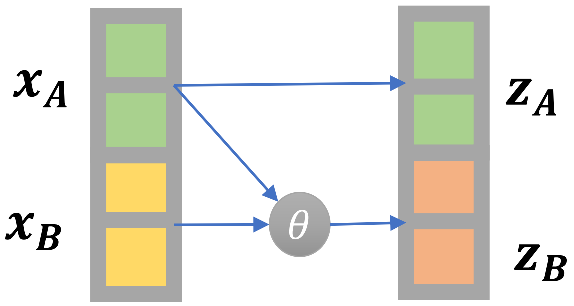
Coupling Flow: The forward transformation



Coupling Flow: The inverse transformation



Coupling Flow: Jacobian



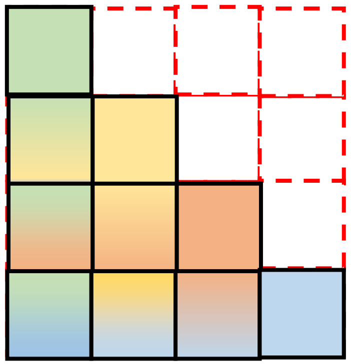
$$J_f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial z_A}{\partial x_A} & \frac{\partial z_A}{\partial x_B} \\ \frac{\partial z_B}{\partial x_A} & \frac{\partial z_B}{\partial x_B} \end{bmatrix} = \begin{bmatrix} \mathbb{I} & 0 \\ \boxed{\frac{\partial g(x_B|\theta(x_A))}{\partial x_A}} & J_g \end{bmatrix}$$

We don't need this for determinant!

θ can be an arbitrarily complex (non-invertible) model like a neural network

Block diagonal jacobian, determinant computation is efficient

$O(d)$

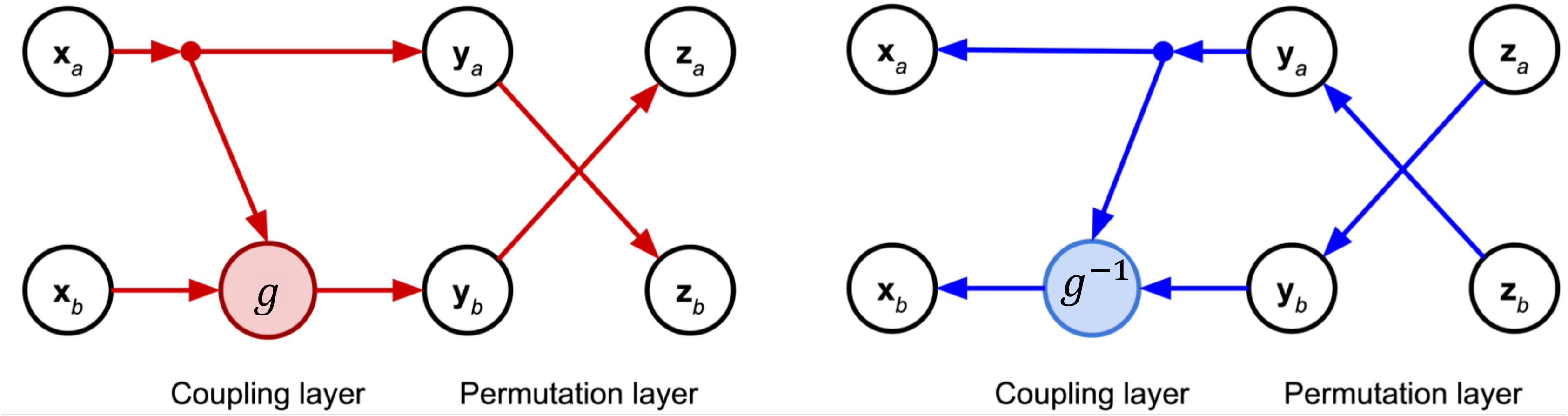


Coupling Transforms

Just Compose ? Needs permutations between flow layers to utilize the entire space

- Permutation is an invertible linear operation
    - Has unit determinant

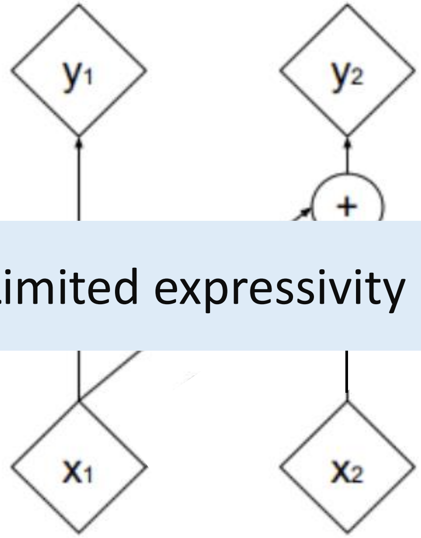
$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{P}^{\mathrm{T}}\text{ matrix}} \cdot \underbrace{\begin{bmatrix} 4 & 6 & 8 & 1 \\ 5 & 3 & 1 & 0 \\ 7 & 21 & 0 & 9 \\ 3 & -4 & 1 & 3 \end{bmatrix}}_{\mathbf{A}\text{ matrix}} = \underbrace{\begin{bmatrix} 3 & -4 & 1 & 3 \\ 4 & 6 & 8 & 1 \\ 5 & 3 & 1 & 0 \\ 7 & 21 & 0 & 9 \end{bmatrix}}_{\mathbf{P}^{\mathrm{T}}\cdot\mathbf{A}\text{ matrix}}$$



Coupling Transforms

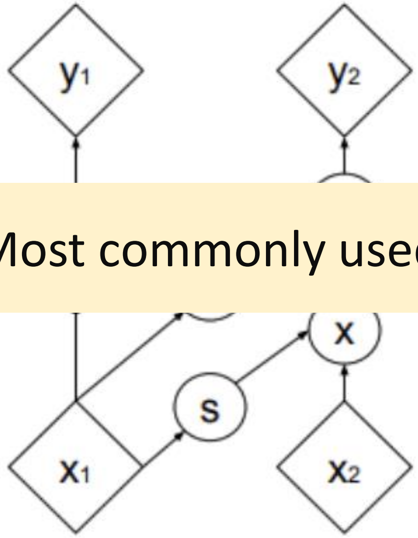
Defining Coupling transforms g .. **Simple** invertible transformations suffice

- Additive [NICE-Dinh et al., 2014]
  - $g(x|t) \quad = x + t$
  - $g^{-1}(z|t) = z - t$



Limited expressivity

- Affine [RealNVP-Dinh et al., 2017]
  - $g(x|s,t) \quad = \exp(s) \odot x + t$
  - $g^{-1}(z|s,t) = \exp(s)^{-1} \odot (z - t)$

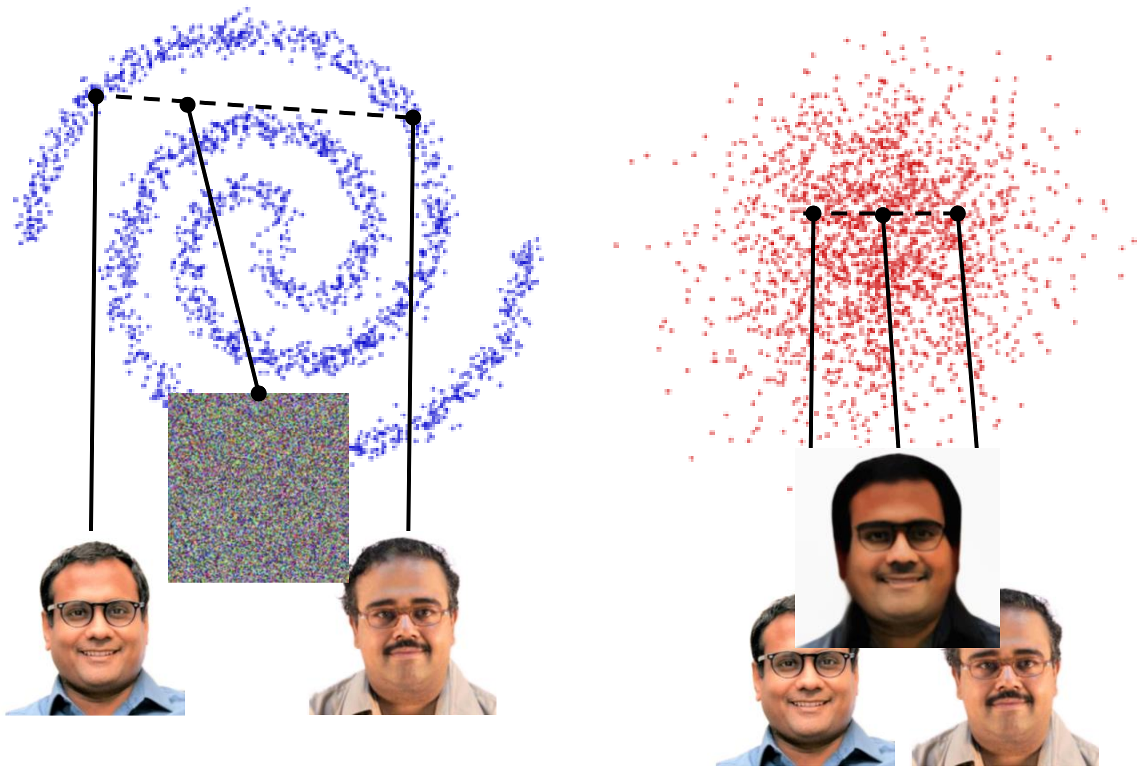


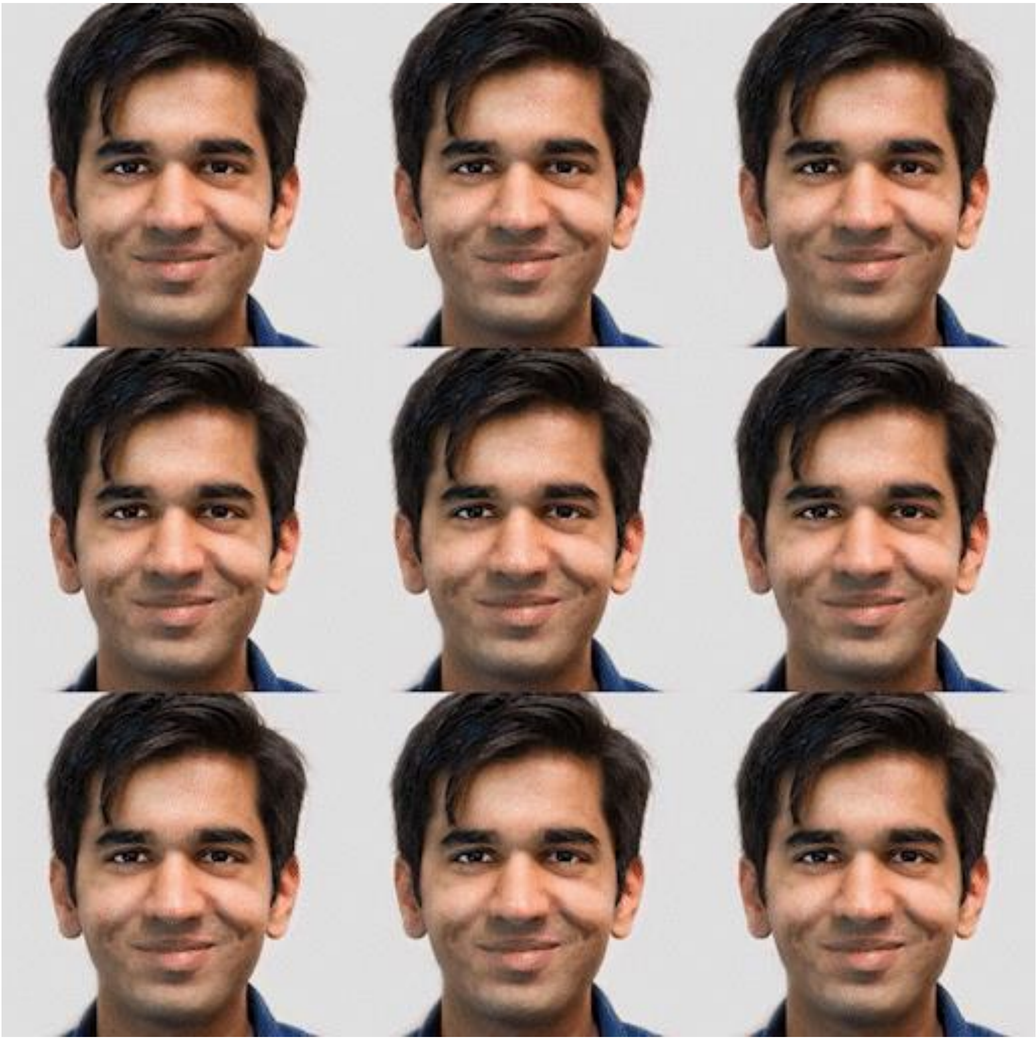
Most commonly used

Can have more complex coupling transform formulations .. MLP, splines, etc. [Kingma et al., 2016, Huang et al., 2018; de Cao et al., 2019; Durkan et al., 2019]

Utilizing the latent space of the flow

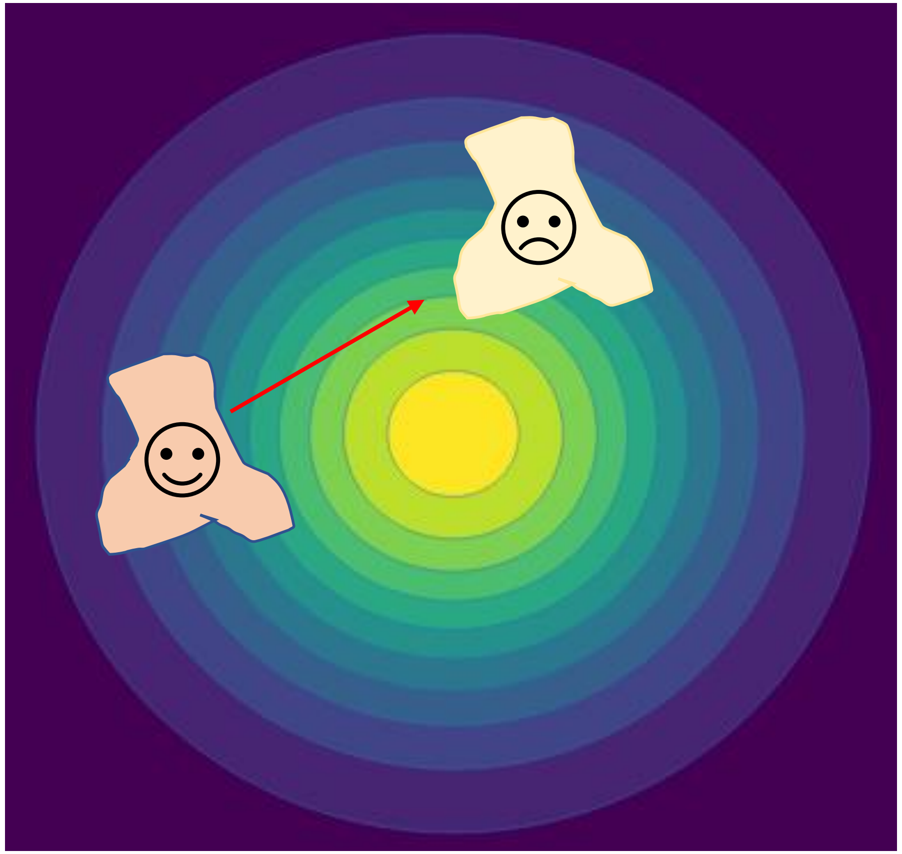
Can Interpolate between points in the latent space

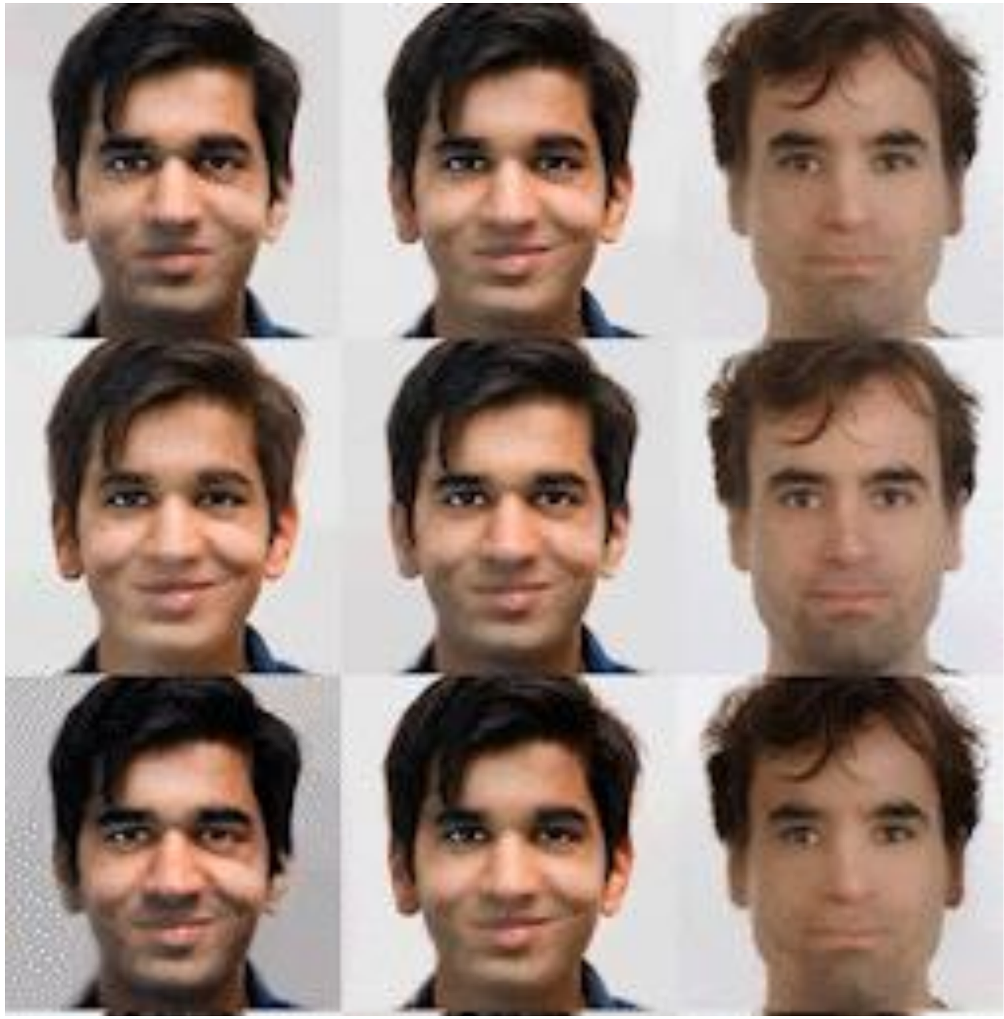




Utilizing the latent space of the flow

Manipulate semantic features by moving in certain directions of the latent space





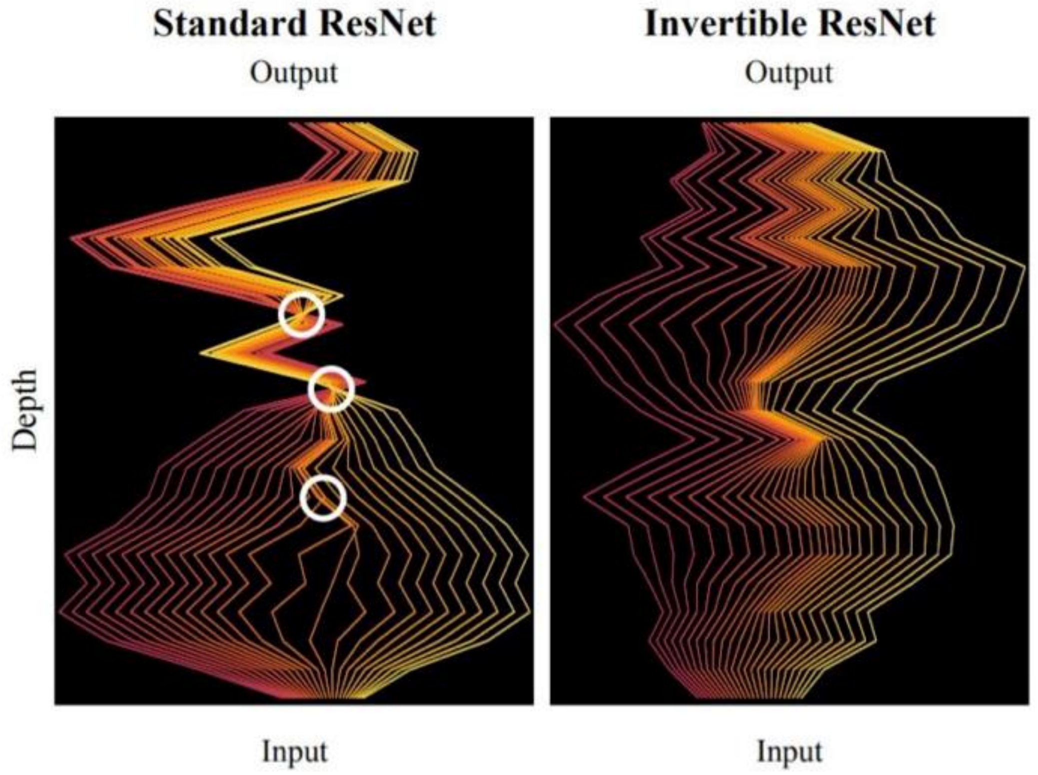
Normalizing Flows
Hands-On Demo

https://bit.ly/tpm-day2-flows

Conclusion

- Other approaches
  - Discrete time invertible transforms
    - Contractive Flows [Behrmann et al. 2019, Perugachi-Diaz et. Al 2020, Chen et al.]
  - Continuous time invertible transforms
    - Neural ODE flows [Grathwohl et al. 2018]

- Limitations and takeaways
  - Tractability compared to PCs
  - Dimensionality preserving
  - Topological restrictions of diffeomorphisms



Some Resources on Flows

Review Paper

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2021). **Normalizing flows for probabilistic modeling and inference**. *The Journal of Machine Learning Research*,

Tutorial

Normalizing Flows and Invertible Neural Networks in Computer Vision, CVPR 2021 by Marcus A. Brubaker, Ullrich Köthe

Github Repo

janosh / **awesome-normalizing-flows** Public

Thank You!