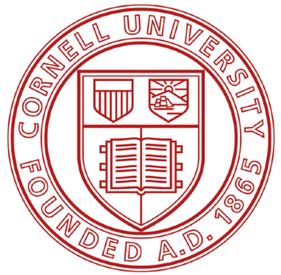
Lecture 6: Normalizing flows

CS 5788: Introduction to Generative Models

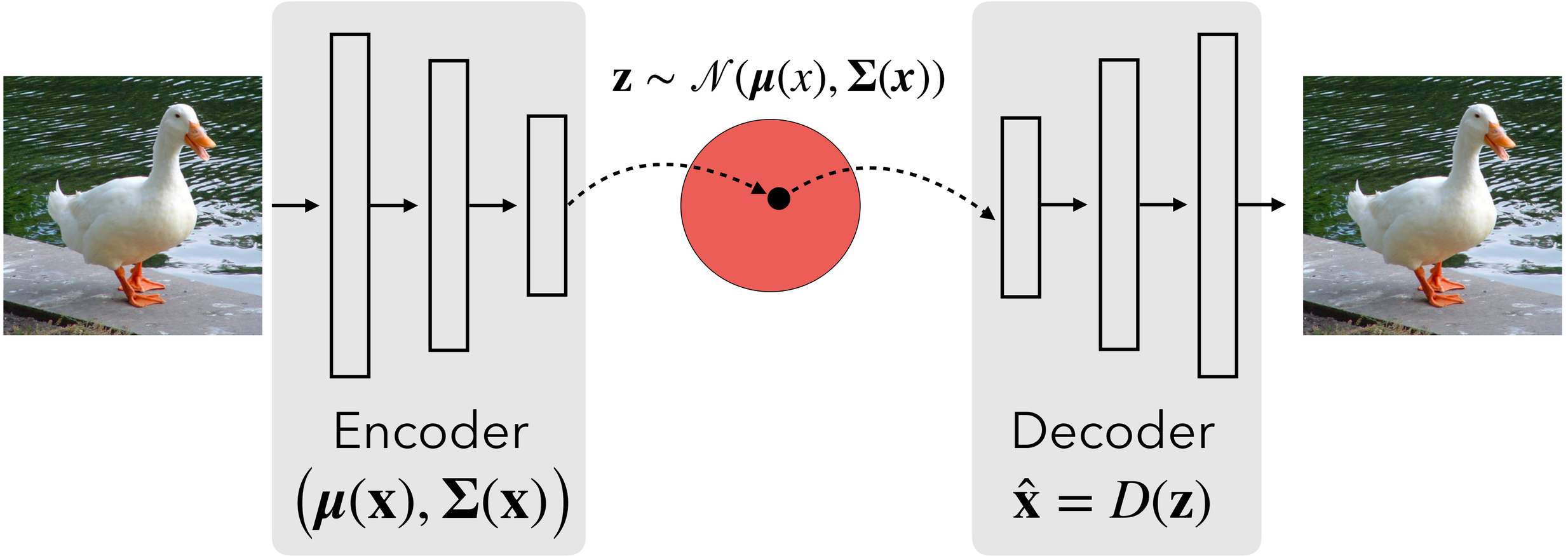


Many slides from Brubaker & Köthe and figures from Kobyzev, Prince, and Brubaker normalizing flow tutorial.

Today

- PS1 due next Tuesday
- Project info out in late February

Recall: Variational autoencoders



- Encoder net defines a distribution over $\mathbf{z}$: $q(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}\left(\boldsymbol{\mu}(x), \boldsymbol{\Sigma}(\boldsymbol{x})\right)$.
- Decoder net defines a distribution over $\mathbf{x}$: $p(\mathbf{x} \mid \mathbf{z}) = D(\mathbf{z})$.

Recall: training a VAE

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

$$\geq \int q(\mathbf{z} \mid \mathbf{x}) \log p(\mathbf{x} \mid \mathbf{z})\, d\mathbf{z} - \int q(\mathbf{z} \mid \mathbf{x}) \log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})}\, d\mathbf{z}$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{z})] - D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z}))$$

Reconstruction loss:

$$\mathscr{L}_R = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$ (up to scale factor)

Equivalent to log likelihood of $\mathcal{N}(\hat{\mathbf{x}}, I)$: i.e., how likely is $\mathbf{x}$ under a Gaussian with mean $\hat{\mathbf{x}}$?

KL divergence loss:

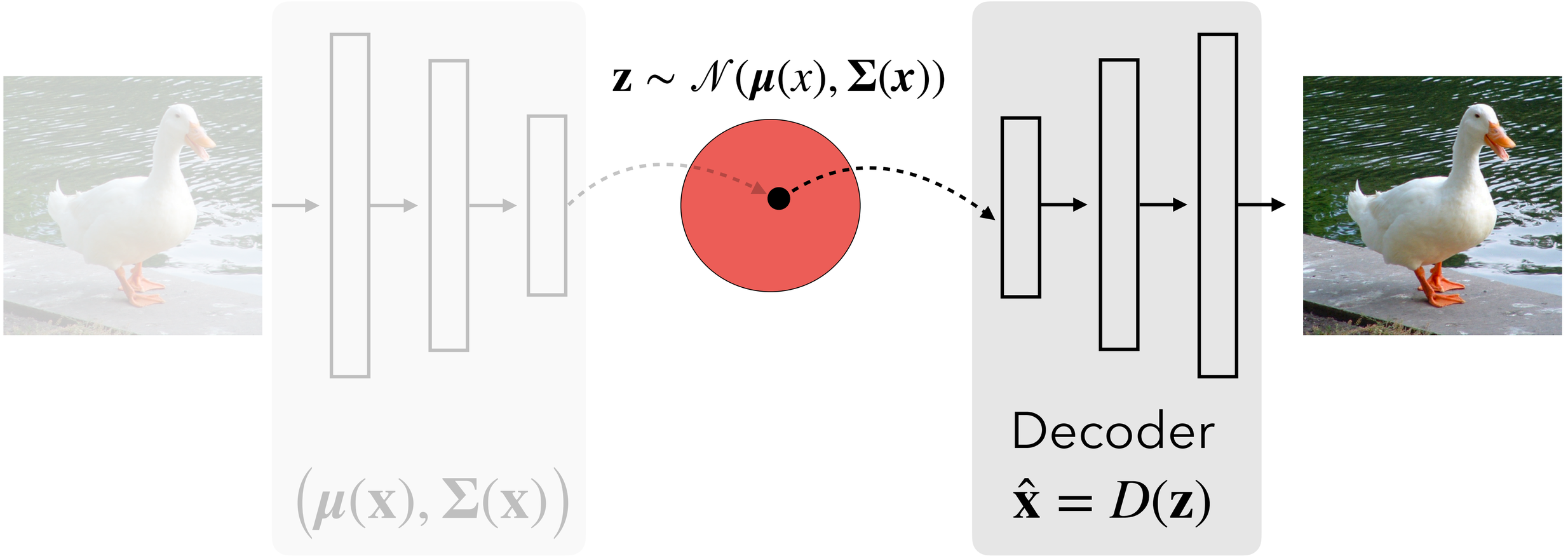
$$\mathscr{L}_P = \mathrm{D}_{KL}\left(q(\mathbf{z} \mid \mathbf{x}) \parallel \mathcal{N}(0, I)\right)$$

Make $\mathbf{z}$ to match our desired source distribution.

Potential problems with VAEs

- Can't directly estimate $\log p_\theta(\mathbf{x})$.
  - Instead, you are optimizing a lower bound.
- Requires you to capture all variation in low-dimensional $\mathbf{z}$.
- Need separate encoder/decoder networks.

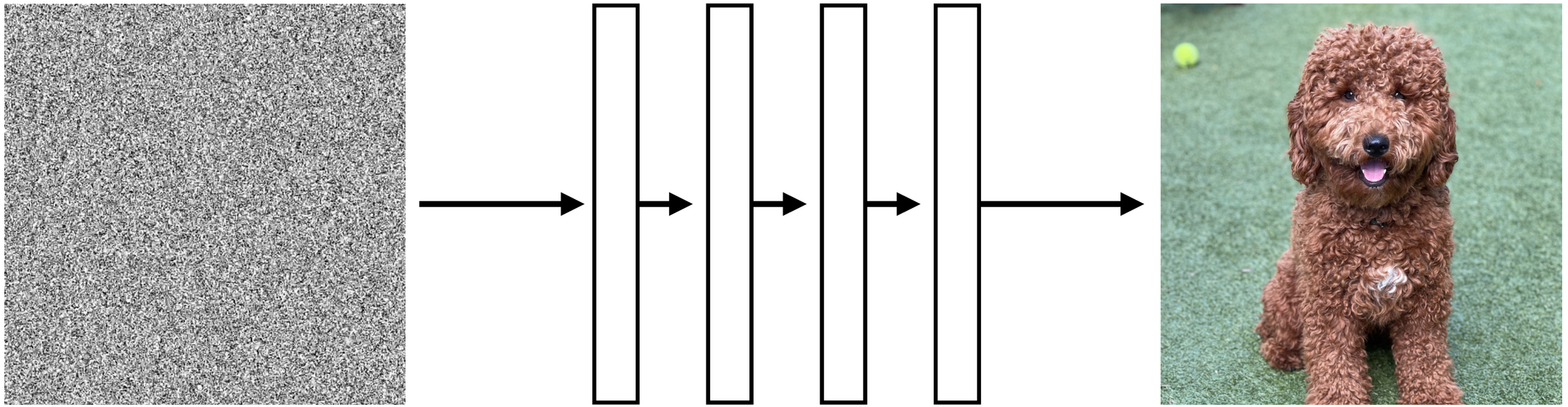
Recall: Variational autoencoders



Can we get rid of the encoder?

Invertible transformations

generative direction



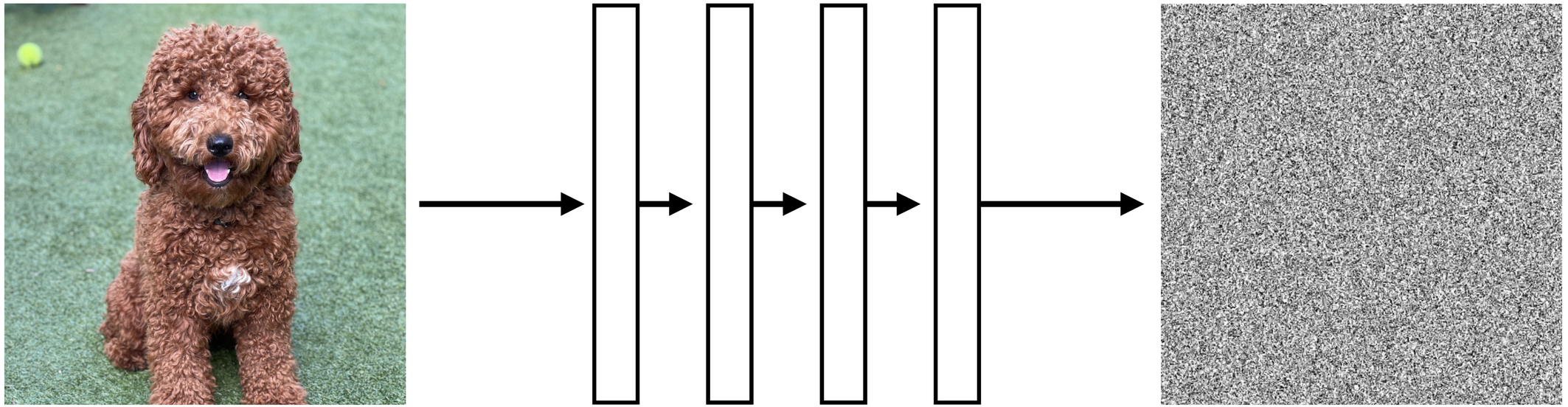
$\mathbf{Z}$

$\mathbf{X} = g(\mathbf{Z})$

$g : \mathbf{Z} \rightarrow \mathbf{X}$

Invertible transformations

normalizing direction (“flow”)



$\mathbf{X}$

$f \triangleq g^{-1}$

$\mathbf{Z} = f(\mathbf{X})$

$f : \mathbf{X} \rightarrow \mathbf{Z}$

Change of variables formula

Volume correction

If $\mathbf{X} = g(\mathbf{Z})$, then $p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}\left(g^{-1}(\mathbf{x})\right) \left|\det \mathrm{D}g^{-1}(\mathbf{x})\right|$

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}\left(f(\mathbf{x})\right) \left|\det \mathrm{D}f(\mathbf{x})\right| \text{ for } f \triangleq g^{-1}$$

where $\mathrm{D}f$ is the **Jacobian**, i.e., the matrix of partial derivatives:

$$\mathrm{D}f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}$$

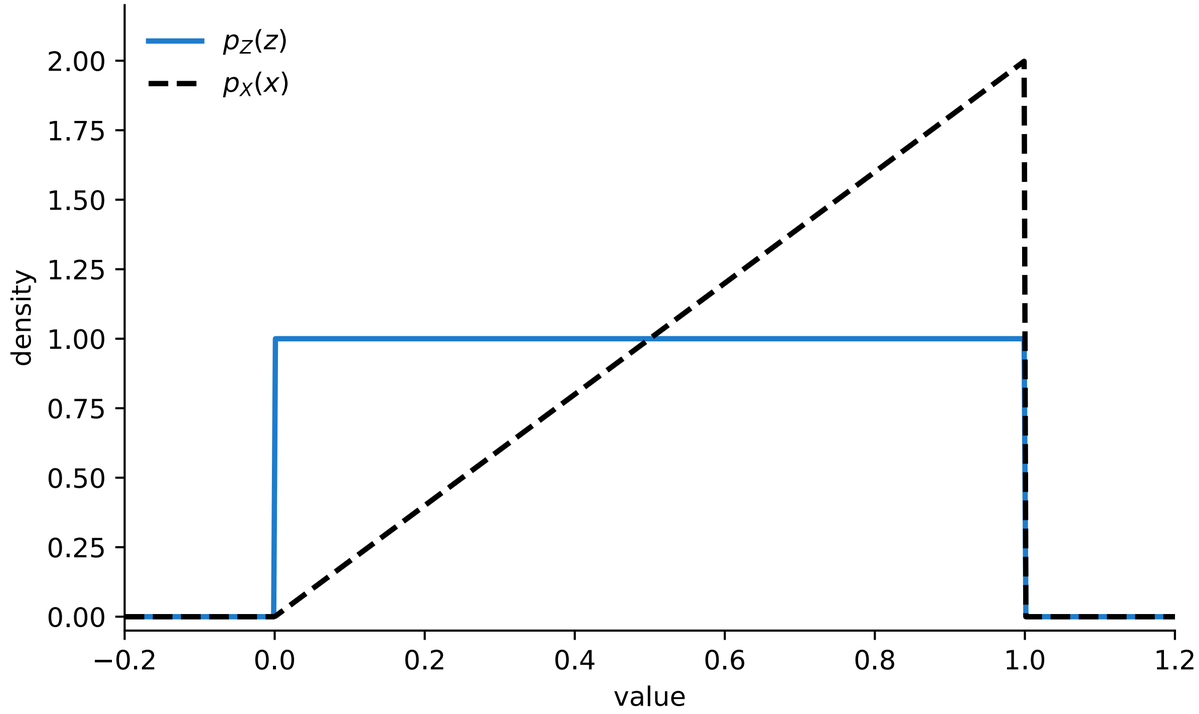
We say that $p_{\mathbf{X}}$ is the **pushforward** of $p_{\mathbf{Z}}$ by g.

Example: change of variables formula

If $\mathbf{X} = g(\mathbf{Z})$, then $p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}\left(g^{-1}(\mathbf{x})\right) \left|\det \mathrm{D}g^{-1}(\mathbf{x})\right|$

$\mathbf{X} = \sqrt{\mathbf{Z}}$ where $\mathbf{Z} \sim U(0, 1)$, i.e., $p_{\mathbf{Z}}(z) = \begin{cases} 1, & 0 < z < 1, \\ 0, & \text{otherwise.} \end{cases}$

$$p_{\mathbf{X}}(x) = \begin{cases} 2x & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$



Function composition

Composition of N functions:

$$g = g_N \circ g_{N-1} \circ \cdots \circ g_1$$

$$g : \mathbf{Z} \to \mathbf{X}$$

Inverse:

$$f = f_1 \circ f_2 \circ \ldots \circ f_N$$

$$f : \mathbf{X} \to \mathbf{Z}$$

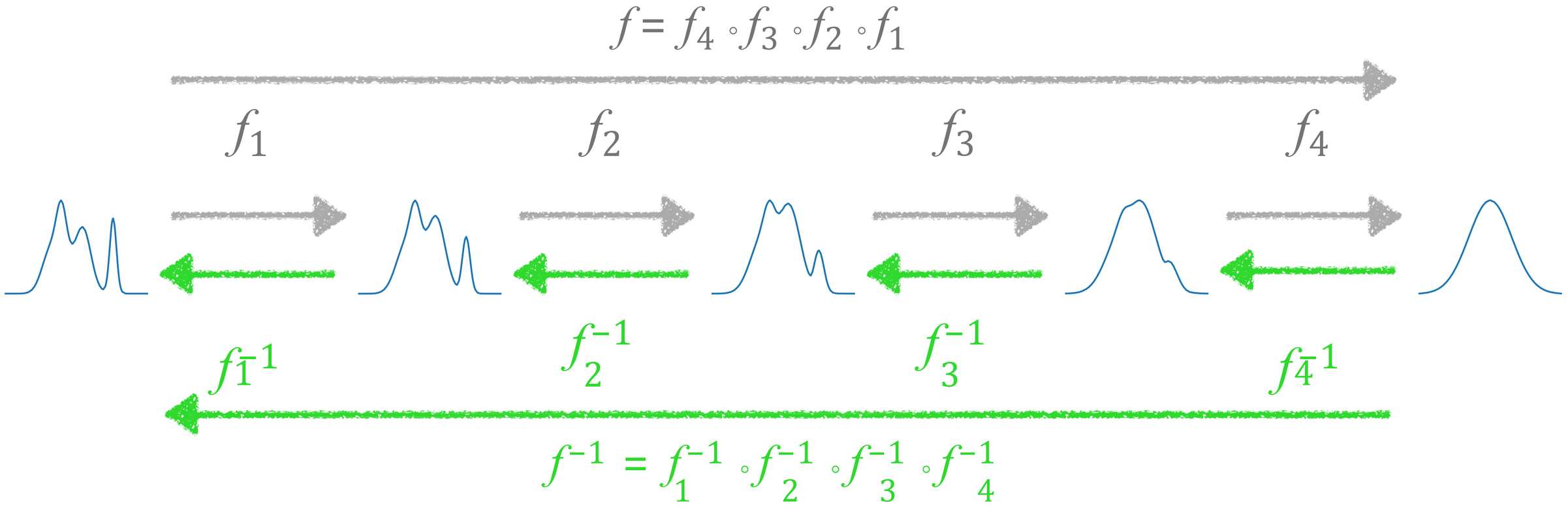
Determinant:

$$\det Df(\mathbf{x}) = \prod_{i=1}^{N} \det \mathrm{D}f_i(\mathbf{y}_i)$$

where $\mathbf{y}_i = g_i \circ g_{i-1} \circ \ldots g_1(\mathbf{z})$

$$= f_1 \circ f_2 \circ \ldots f_i(\mathbf{x})$$

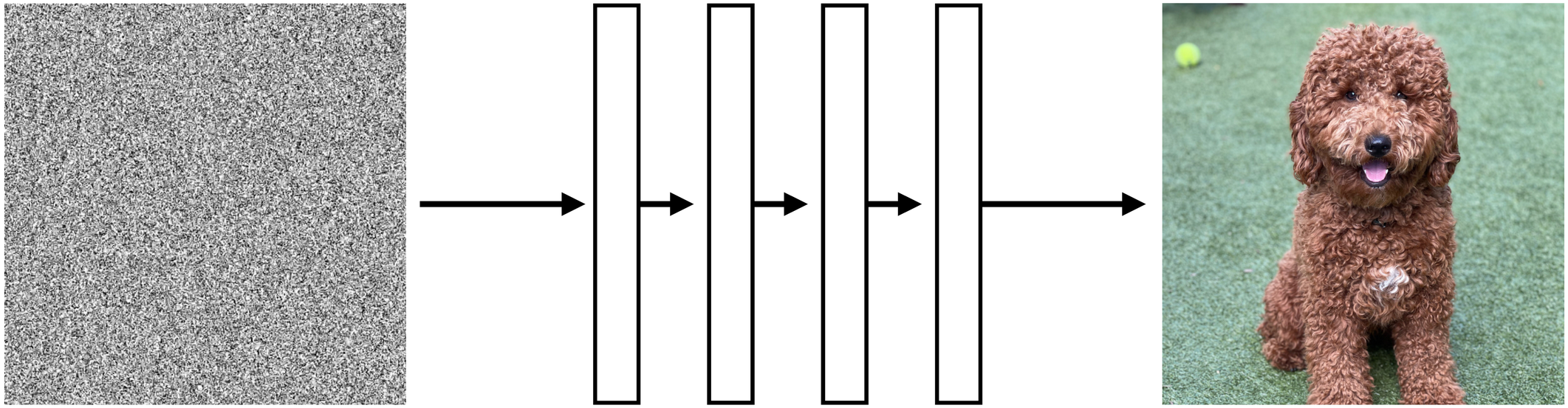
Composition of Flows



Source: M. Brubaker & Köthe

Building a deep generative model

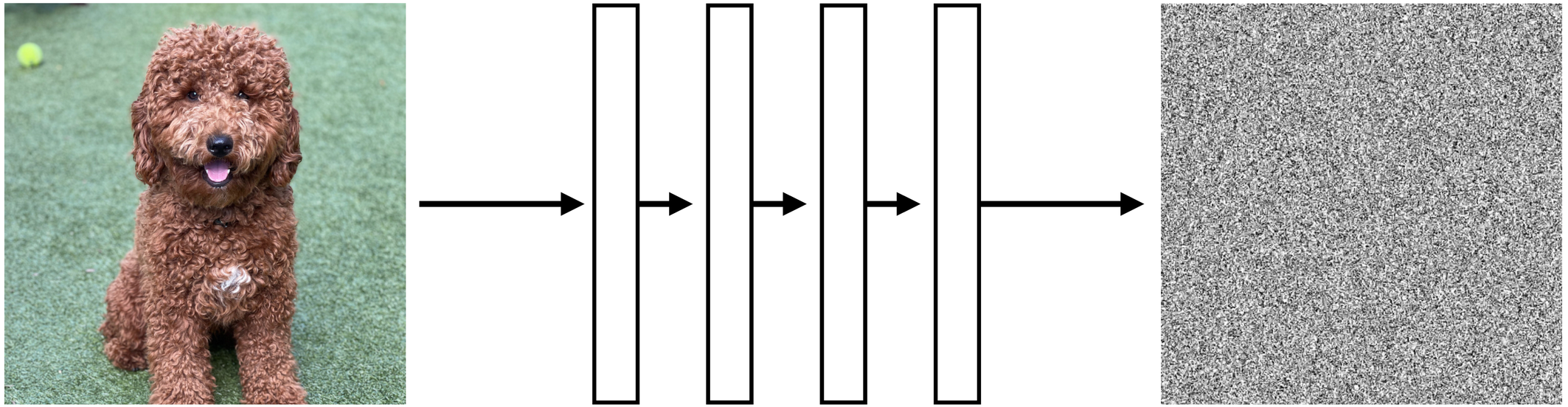
Normalizing flow model



$$g_N \circ g_{N-1} \circ \cdots \circ g_1(\mathbf{z})$$

Normalizing flow model

For estimating the density and learning:



$$\log p_\theta(\mathbf{X}) = \sum_{i=1}^{M} \log p_{\mathbf{Z}}(f(\mathbf{x}_i)) + \log \left| \det(\mathrm{D}f(\mathbf{x}_i)) \right|$$ where $\mathbf{x}_i$ is a training example.

Exact likelihood estimation (unlike VAE, which uses lower bound).

What’s the catch?

Let’s look at the change-of-variables formula again:

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}\left(g^{-1}(\mathbf{x})\right) \left|\det \mathrm{D}g^{-1}(\mathbf{x})\right|$$

Network architecture?

$$\log p_\theta(\mathbf{X}) = \sum_{i=1}^{M} \log p_{\mathbf{Z}}(f(\mathbf{x}_i)) + \log \left| \det(\mathrm{D}f(\mathbf{x}_i)) \right|$$

The change-of-variables formula places constraints on network:

- Each layer g_i must be invertible
- Must be able to compute $\log \det Df_i$
- Latent $\mathbf{z}$ needs to be same dimension as $\mathbf{x}$
- All of these must be efficient to compute!

Pointwise nonlinearities

$$f(\mathbf{y}) = \begin{bmatrix} g(y_1) \\ g(y_2) \\ \cdots \\ g(y_d) \end{bmatrix}$$

if g is invertible, so is f

Pointwise nonlinearities

$$f(\mathbf{y}) = \begin{bmatrix} g(y_1) \\ g(y_2) \\ \dots \\ g(y_d) \end{bmatrix}$$

Jacobian:

$$D = \begin{bmatrix} g'(y_1) & 0 & 0 & 0 \\ 0 & g'(y_2) & 0 & 0 \\ 0 & 0 & g'(y_3) & 0 \\ 0 & 0 & 0 & \dots \end{bmatrix}$$

$$\det D = \prod_{i=1}^{N} g'(y_i)$$

(Jacobian of f^{-1} is similar, but using f for simplicity)

Linear layers

$$f(\mathbf{y}) = A\mathbf{y} + \mathbf{b}$$

$$f : \mathbb{R}^n \to \mathbb{R}^n$$

Inverse? $f^{-1}(\mathbf{y}) = A^{-1}(\mathbf{y} - \mathbf{b})$

but A needs to be full rank.

Determinant? $\det \mathrm{D}f(\mathbf{x}) = \det A$

…but expensive! $\mathcal{O}(N^3)$

Linear layers: special cases

$$g(\mathbf{y}) = D\mathbf{y} + \mathbf{b}$$

$$g : \mathbb{R}^n \to \mathbb{R}^n$$

Diagonal matrix:

$$D = \mathrm{diag}(a, b, c, \ldots) = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & \ldots \end{bmatrix}$$

$$\det(D) = a \times b \times c \times \ldots$$

computable in $O(n)$ time.

Linear layers: special cases

$$g(\mathbf{y}) = L\mathbf{y} + \mathbf{b}$$

$$g : \mathbb{R}^n \to \mathbb{R}^n$$

Triangular matrix:

$$L = \begin{bmatrix} a & 0 & 0 & 0 \\ b & c & 0 & 0 \\ d & e & f & 0 \\ g & h & i & j \end{bmatrix}$$

- Determinant is product of diagonal.
- Invertible in $O(n^2)$ time.

Still, limited expressiveness…

Linear layers: permutation matrix

$$g(\mathbf{y}) = P\mathbf{y} + \mathbf{b}$$

$$g : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

Permutation matrix:

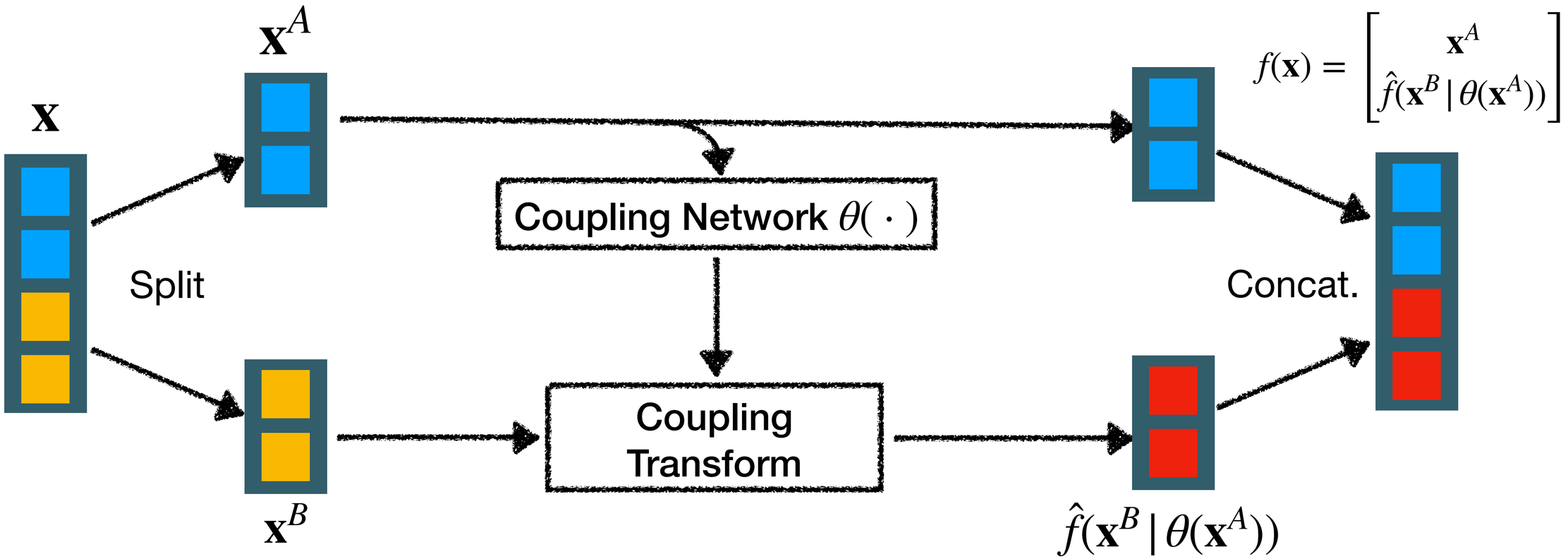
What does this matrix do?

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \det(P) = 1$$

Can chain together with lower triangular matrix to improve expressiveness.

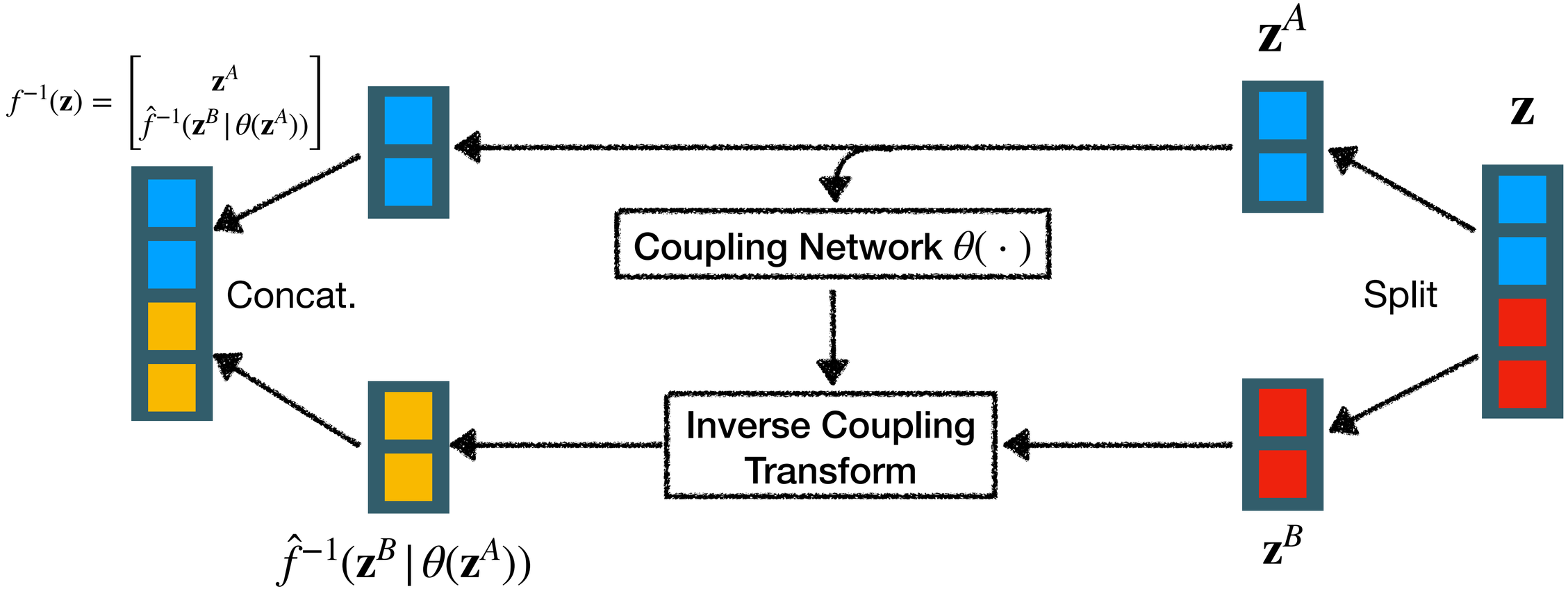
Other useful special cases

Coupling flows



Source: M. Brubaker & Köthe *[Figure adapted from Jason Yu]*

Coupling flows: inverse



Source: M. Brubaker & Köthe *[Figure adapted from Jason Yu]*

Coupling flows

$$f(\mathbf{x}) = \begin{bmatrix} \mathbf{x}^A \\ \hat{f}\left(\mathbf{x}^B \mid \theta\left(\mathbf{x}^A\right)\right) \end{bmatrix} \qquad \mathrm{D}f(\mathbf{x}) = \begin{bmatrix} \end{bmatrix}$$

Coupling flows

$$f(\mathbf{x}) = \begin{bmatrix} \mathbf{x}^A \\ \hat{f}\left(\mathbf{x}^B \mid \theta\left(\mathbf{x}^A\right)\right) \end{bmatrix} \quad \mathrm{D}f(\mathbf{x}) = \begin{bmatrix} I & \mathbf{0} \\ \frac{\partial}{\partial \mathbf{x}^A} \hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) & \mathrm{D}\,\hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) \end{bmatrix}$$

Coupling flows

$$\mathrm{D}f(\mathbf{x}) = \begin{bmatrix} I & \mathbf{0} \\ \frac{\partial}{\partial \mathbf{x}^A} \hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) & \mathrm{D}\,\hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) \end{bmatrix}$$

$$\det \mathrm{D}f(\mathbf{x}) = ?$$

Useful property:

$$\det \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} = \det(A) \det(D)$$

Coupling flows

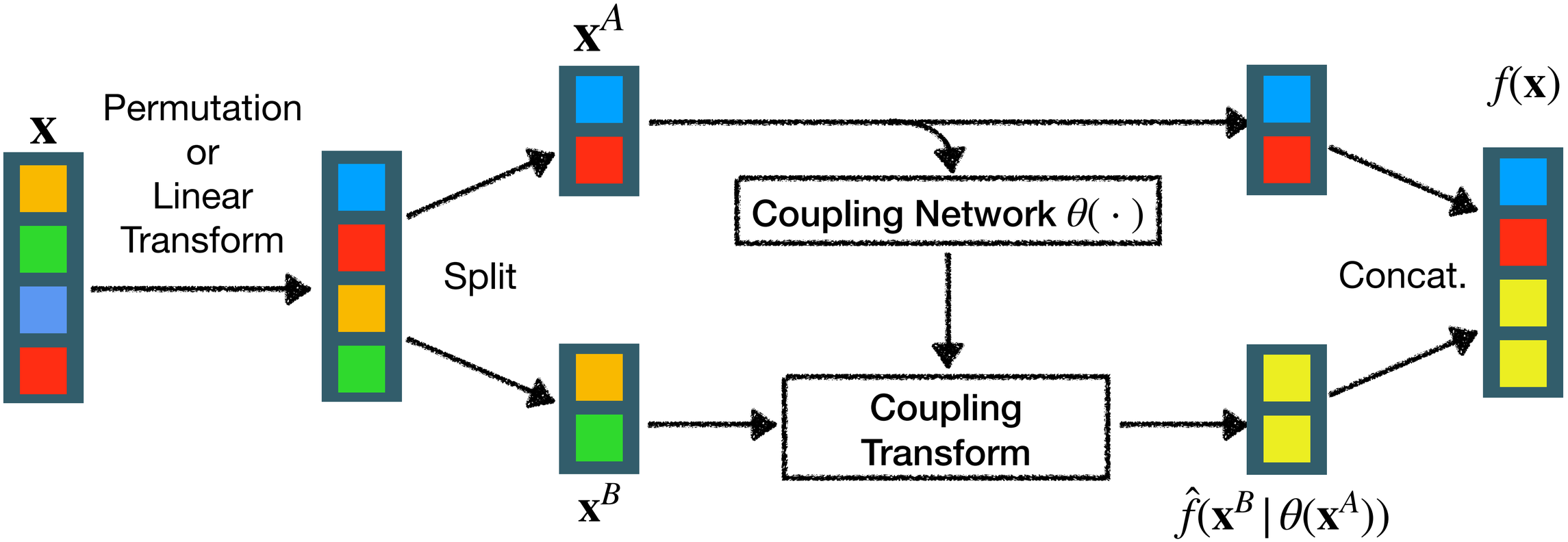
$$\mathrm{D}f(\mathbf{x}) = \begin{bmatrix} I & \mathbf{0} \\ \frac{\partial}{\partial \mathbf{x}^A} \hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) & \mathrm{D}\,\hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) \end{bmatrix}$$

$$\det \mathrm{D}f(\mathbf{x}) = \det\left(\mathrm{D}\,\hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right)\right)$$

$\theta(\mathbf{x}^A)$ is arbitrary!

(e.g., MLP, CNN, etc.)

Coupling flows



Can increase expressivity with other operations.

Source: M. Brubaker & Köthe *[Figure adapted from Jason Yu]*

Coupling transforms

Lots of possible choices.

- Additive [“NICE”, Dinh et al 2014]

$$\hat{f}(\mathbf{x} \mid \mathbf{t}) = \mathbf{x} + \mathbf{t}$$

(where $\mathbf{t}$ is a function that maps $\mathbf{x}^A$ to a vector the same size as $\mathbf{x}^B$).

- Affine [“RealNVP”, Dinh et al 2016], where $\odot$ is elementwise product

$$\hat{f}(\mathbf{x} \mid \mathbf{s}, \mathbf{t}) = \mathbf{x} \odot \exp(\mathbf{s}) + \mathbf{t}$$

Adapted from M. Brubaker & Köthe

Coupling flows (affine model)

$$\hat{f}(\mathbf{x} \mid \mathbf{s}, \mathbf{t}) = \mathbf{x} \odot \exp(\mathbf{s}) + \mathbf{t}$$

Jacobian:

$$\mathrm{D}f(\mathbf{x}) = \begin{bmatrix} I & \mathbf{0} \\ \frac{\partial}{\partial \mathbf{x}^A} \hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) & \mathrm{D}\,\hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) \end{bmatrix}$$

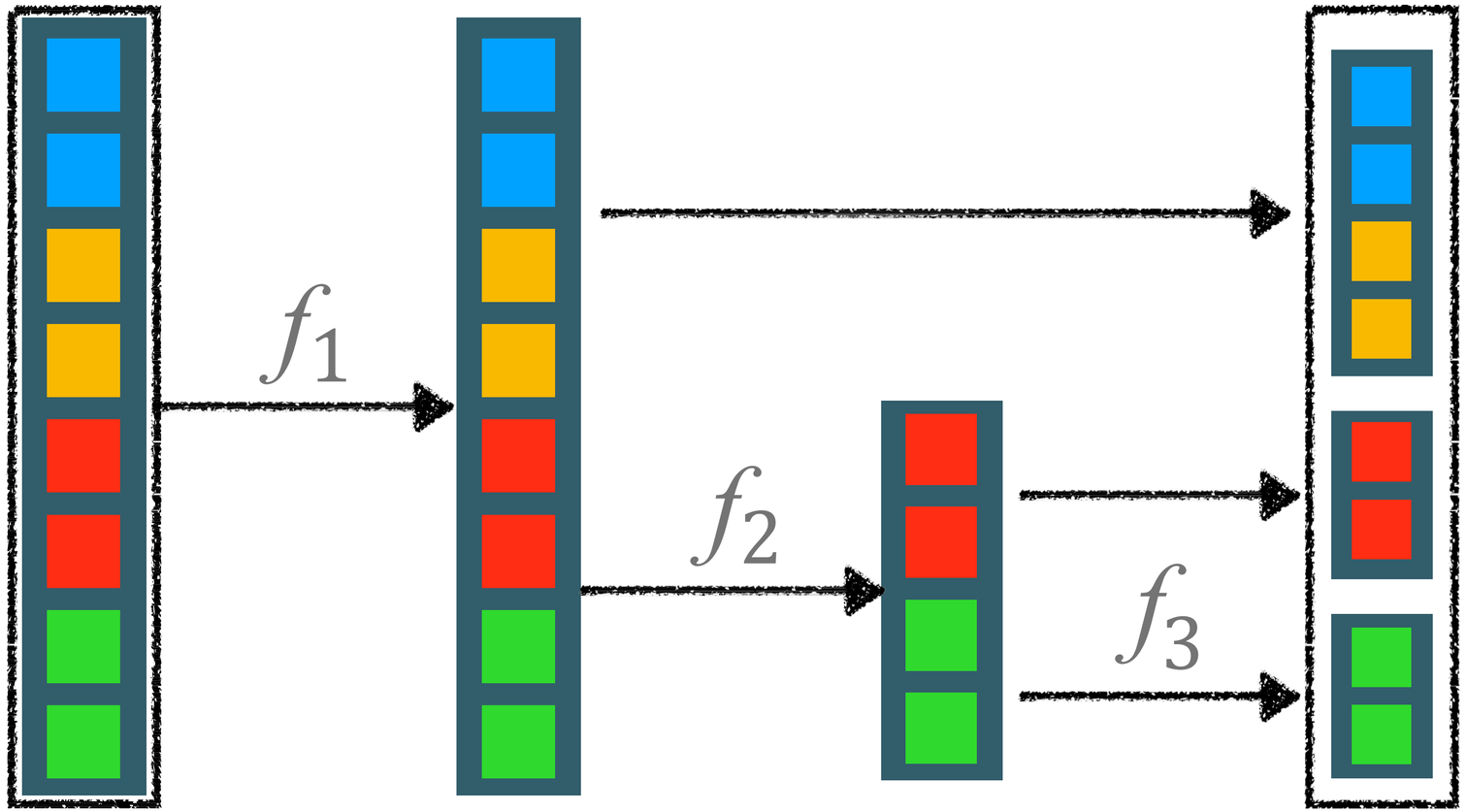
$$\mathrm{D}f(\mathbf{x}) = \begin{bmatrix} I & \mathbf{0} \\ \frac{\partial}{\partial \mathbf{x}^A} \hat{f}\left(\mathbf{x}^B \mid \theta(\mathbf{x}^A)\right) & \mathrm{diag}\left(\exp(\mathbf{s}_1), \ldots, \exp(\mathbf{s}_K)\right) \end{bmatrix}$$

Log determinant:

$$\det(\mathrm{D}f(\mathbf{x}))) = \exp\left(\sum_{i=1}^{K} \mathbf{s}_i\right)$$

Handling images

Multiscale flows

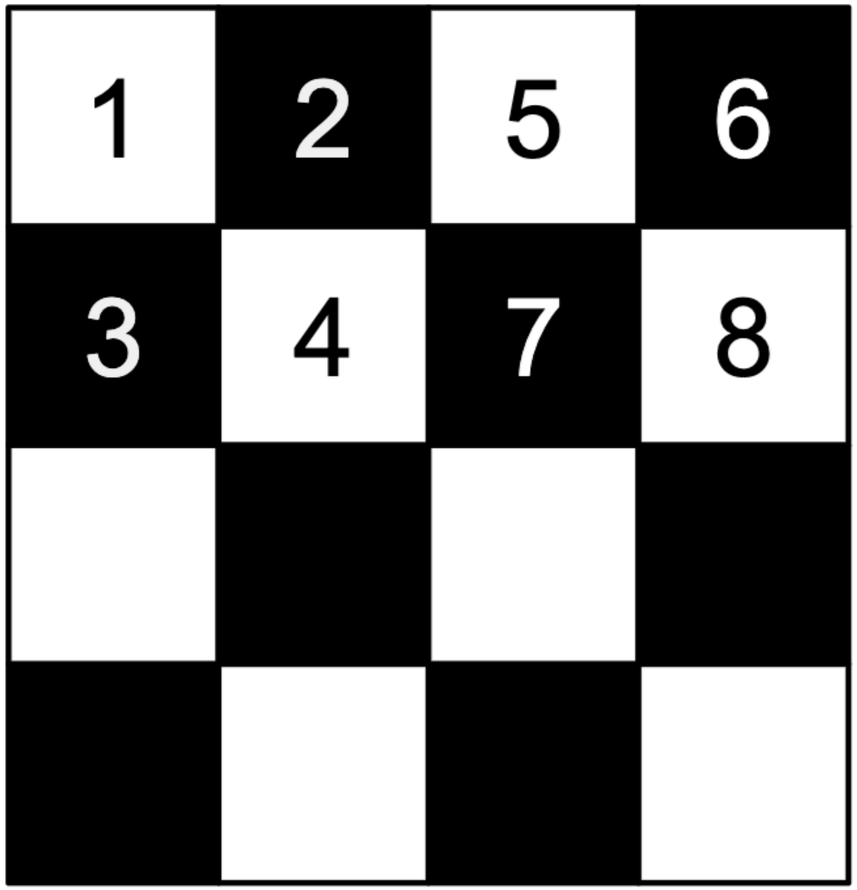


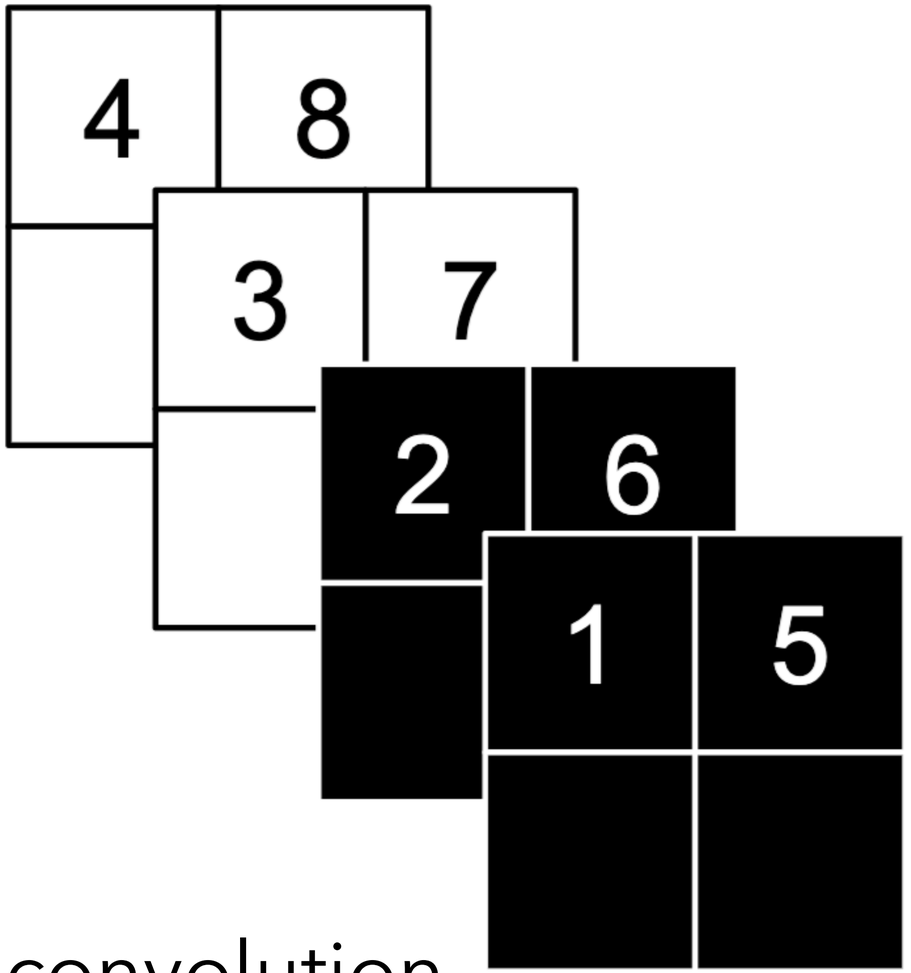
Normalizing flows force same dimension. But we can just stop using subsets of dimensions as we go!

Adapted from M. Brubaker & Köthe

[Dinh et al 2016]

Masked convolution

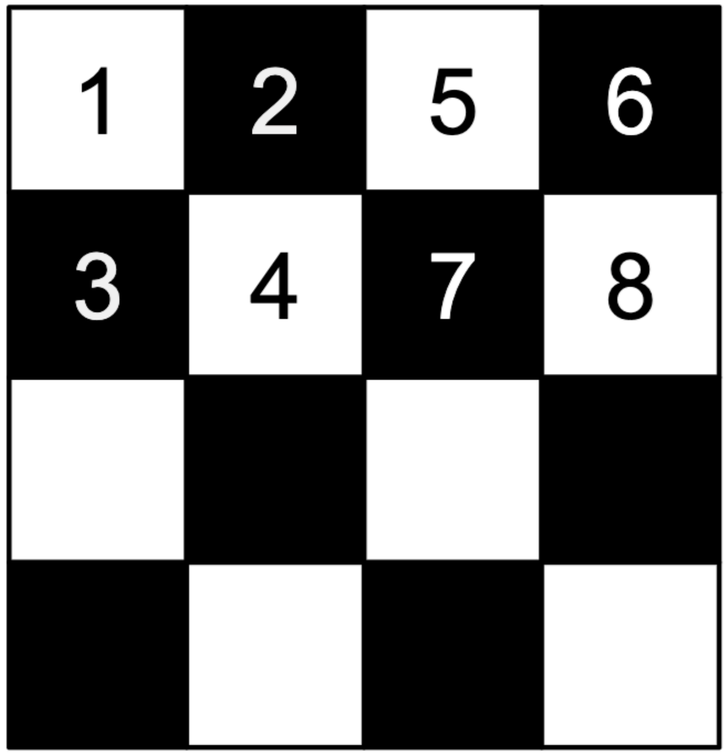




Mask features before convolution

Source: [Dinh et al., “RealNVP”]

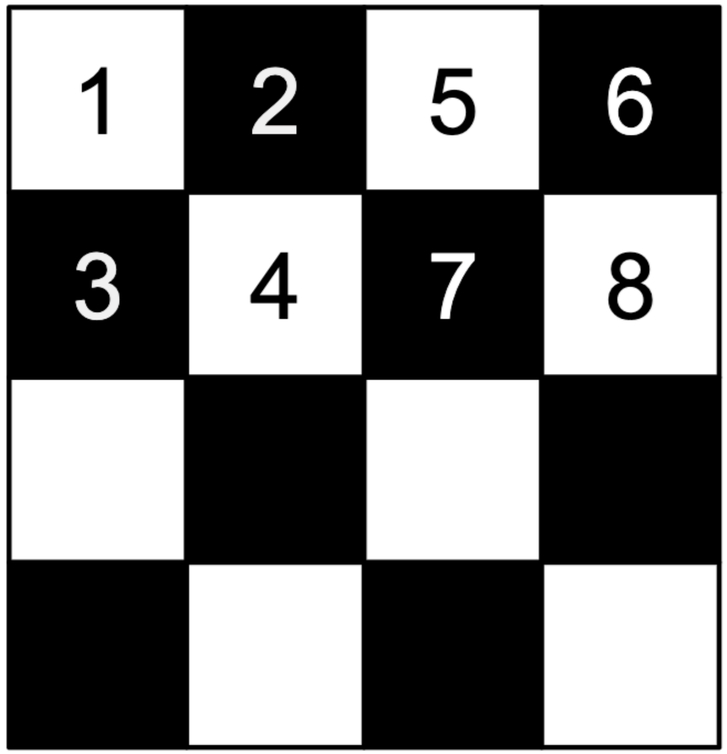
Masked convolution



$$\mathbf{y} = \mathbf{b} \odot \mathbf{x} + (\mathbf{1} - \mathbf{b}) \odot \left(\mathbf{x} \odot \exp\left(s(\mathbf{b} \odot \mathbf{x})\right) + t(\mathbf{b} \odot \mathbf{x}) \right)$$

Source: [Dinh et al., “RealNVP”]

Masked convolution

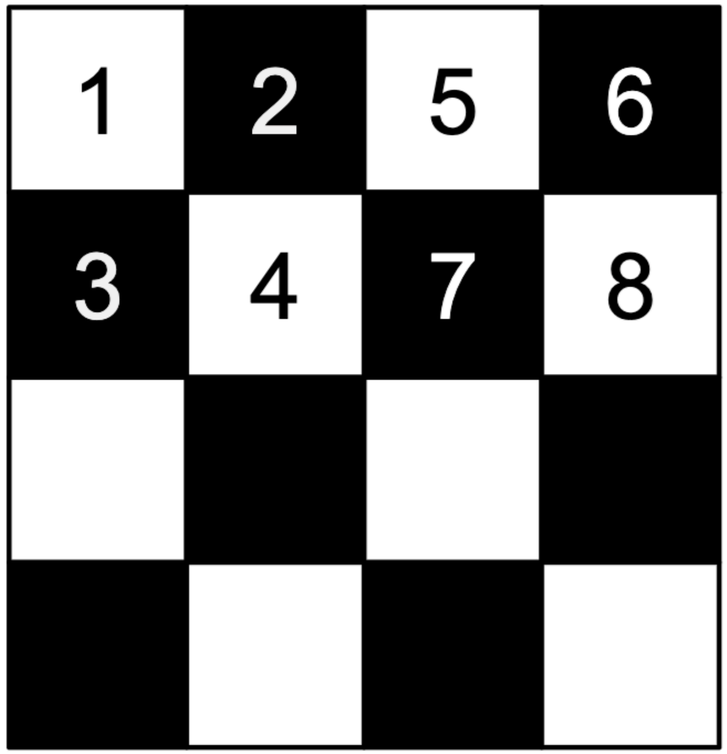


binary mask

$$\mathbf{y} = \mathbf{b} \odot \mathbf{x} + (\mathbf{1} - \mathbf{b}) \odot \left(\mathbf{x} \odot \exp\left(s(\mathbf{b} \odot \mathbf{x}) \right) + t(\mathbf{b} \odot \mathbf{x}) \right)$$

Source: [Dinh et al., “RealNVP”]

Masked convolution

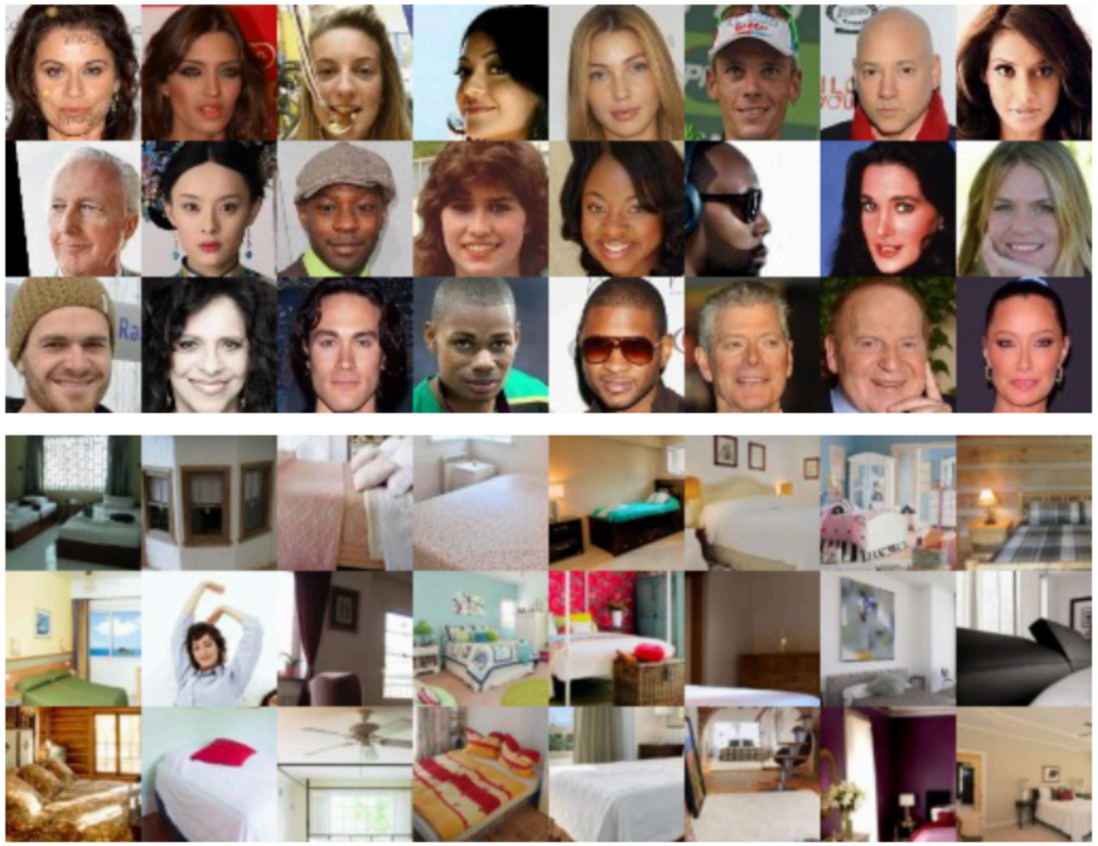


convolutional nets

$$\mathbf{y} = \mathbf{b} \odot \mathbf{x} + (\mathbf{1} - \mathbf{b}) \odot \left(\mathbf{x} \odot \exp\big(s(\mathbf{b} \odot \mathbf{x})\big) + t(\mathbf{b} \odot \mathbf{x}) \right)$$

Source: [Dinh et al., “RealNVP”]

Generated images

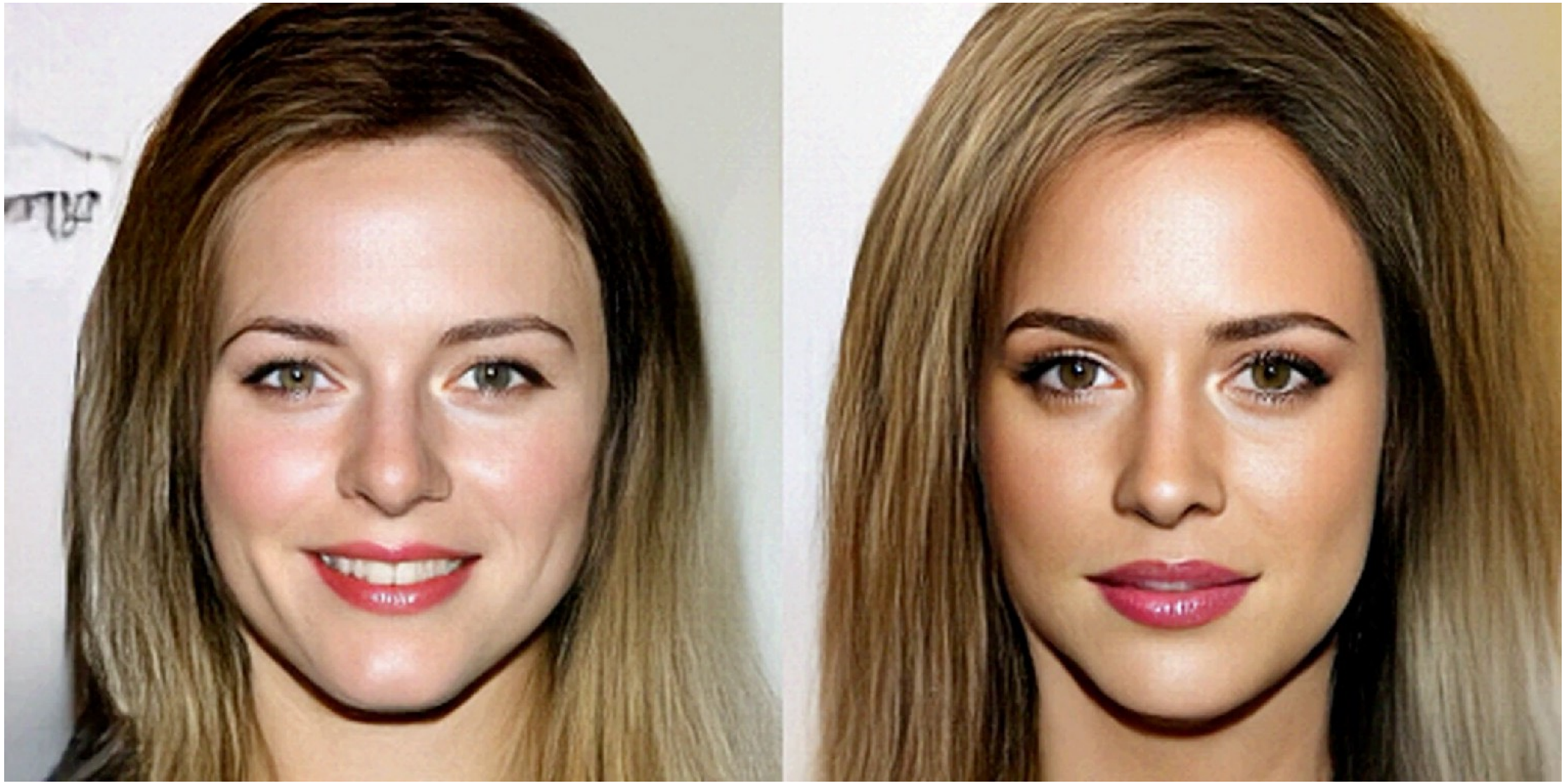


Real

Generated

Source: [Dinh et al., “RealNVP”]

Generated images



Improved model [Kingma and Dhariwal, “GLOW”, NeurIPS 2018]

Source: M. Brubaker & Köthe

Normalizing flows vs. VAEs

- Normalizing flows give exact log likelihood, while VAEs allow you to approximately estimate a lower bound.
- Architecture for VAE is much less constrained.
  - Normalizing flow layers must be invertible
  - Must also be efficient to compute determinant of Jacobian
- VAEs typically reduce dimensionality in encoder, whereas normalizing flow must preserve input size throughout.

Next class: Generative adversarial networks