

# Normalizing Flow Models

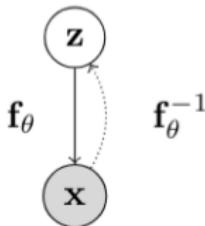
Stefano Ermon

Stanford University

Lecture 8

# Normalizing flow models

- Consider a directed, latent-variable model over observed variables  $X$  and latent variables  $Z$
- In a **normalizing flow model**, the mapping between  $Z$  and  $X$ , given by  $\mathbf{f}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$ , is deterministic and invertible such that  $X = \mathbf{f}_\theta(Z)$  and  $Z = \mathbf{f}_\theta^{-1}(X)$



- Using change of variables, the marginal likelihood  $p(x)$  is given by

$$p_X(\mathbf{x}; \theta) = p_Z(\mathbf{f}_\theta^{-1}(\mathbf{x})) \left| \det \left( \frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$


- Note:  $x, z$  need to be continuous and have the same dimension.

# A Flow of Transformations

**Normalizing:** Change of variables gives a normalized density after applying an invertible transformation

**Flow:** Invertible transformations can be composed with each other

$$\mathbf{z}_m = \mathbf{f}_{\theta}^m \circ \cdots \circ \mathbf{f}_{\theta}^1(\mathbf{z}_0) = \mathbf{f}_{\theta}^m(\mathbf{f}_{\theta}^{m-1}(\cdots(\mathbf{f}_{\theta}^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_{\theta}(\mathbf{z}_0)$$

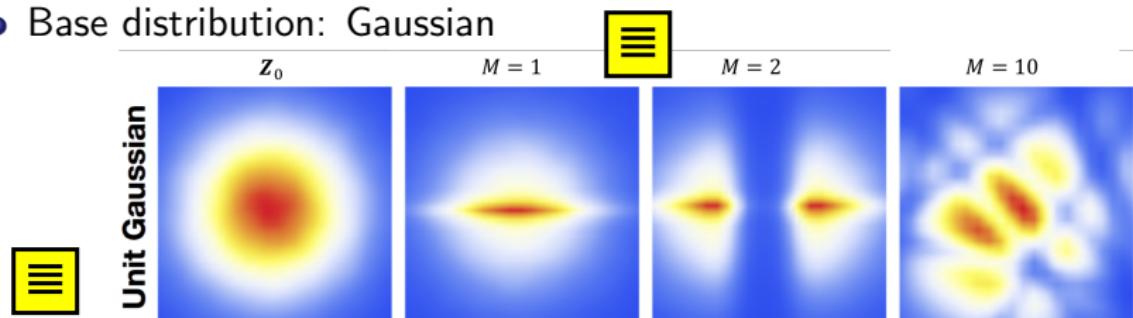

- Start with a simple distribution for  $\mathbf{z}_0$  (e.g., Gaussian)
- Apply a sequence of  $M$  invertible transformations to finally obtain  $\mathbf{x} = \mathbf{z}_M$
- By change of variables

$$p_{\mathbf{x}}(\mathbf{x}; \theta) = p_{\mathbf{z}}(\mathbf{f}_{\theta}^{-1}(\mathbf{x})) \prod_{m=1}^M \left| \det \left( \frac{\partial (\mathbf{f}_{\theta}^m)^{-1}(\mathbf{z}_m)}{\partial \mathbf{z}_m} \right) \right|$$

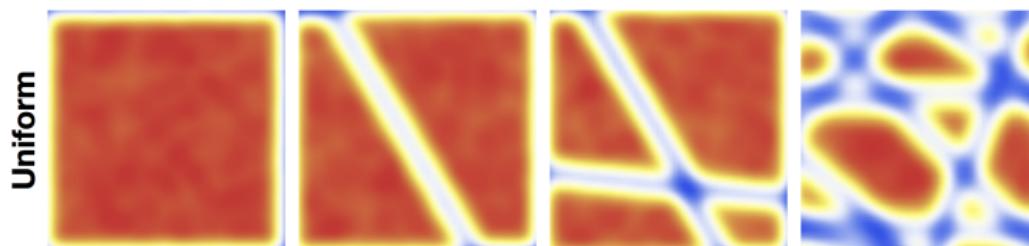
(Note: determinant of product equals product of determinants)

# Planar flows (Rezende & Mohamed, 2016)

- Base distribution: Gaussian



- Base distribution: Uniform



- 10 planar transformations can transform simple distributions into a more complex one

# Learning and Inference

- Learning via **maximum likelihood** over the dataset  $\mathcal{D}$

$$\max_{\theta} \log p_X(\mathcal{D}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p_Z(\mathbf{f}_{\theta}^{-1}(\mathbf{x})) + \log \left| \det \left( \frac{\partial \mathbf{f}_{\theta}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

- **Exact likelihood evaluation** via inverse transformation  $\mathbf{x} \mapsto \mathbf{z}$  and change of variables formula
- **Sampling** via forward transformation  $\mathbf{z} \mapsto \mathbf{x}$

$$\mathbf{z} \sim p_Z(\mathbf{z}) \quad \mathbf{x} = \mathbf{f}_{\theta}(\mathbf{z})$$

- **Latent representations** inferred via inverse transformation (no inference network required!)

$$\mathbf{z} = \mathbf{f}_{\theta}^{-1}(\mathbf{x})$$

# Desiderata for flow models

- Simple prior  $p_Z(\mathbf{z})$  that allows for efficient sampling and tractable likelihood evaluation. E.g., isotropic Gaussian
- Invertible transformations with tractable evaluation:
  - Likelihood evaluation requires efficient evaluation of  $\mathbf{x} \mapsto \mathbf{z}$  mapping
  - Sampling requires efficient evaluation of  $\mathbf{z} \mapsto \mathbf{x}$  mapping
- Computing likelihoods also requires the evaluation of determinants of  $n \times n$  Jacobian matrices, where  $n$  is the data dimensionality
  - Computing the determinant for an  $n \times n$  matrix is  $O(n^3)$ : prohibitively expensive within a learning loop!
  - **Key idea:** Choose transformations so that the resulting Jacobian matrix has special structure. For example, the determinant of a triangular matrix is the product of the diagonal entries, i.e., an  $O(n)$  operation

# Triangular Jacobian

$$\mathbf{x} = (x_1, \dots, x_n) = \mathbf{f}(\mathbf{z}) = (f_1(\mathbf{z}), \dots, f_n(\mathbf{z}))$$

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial z_1} & \cdots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$$

Suppose  $x_i = f_i(\mathbf{z})$  only depends on  $\mathbf{z}_{\leq i}$ . Then

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & 0 \\ \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial z_1} & \cdots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$$



has lower triangular structure. Determinant can be computed in **linear time**. Similarly, the Jacobian is upper triangular if  $x_i$  only depends on  $\mathbf{z}_{\geq i}$

# Recap of normalizing flow models

So far

- Transform simple to complex distributions via sequence of invertible transformations
- Directed latent variable models with marginal likelihood given by the change of variables formula
- Triangular Jacobian permits efficient evaluation of log-likelihoods

Plan for today



- Invertible transformations with diagonal Jacobians (NICE, Real-NVP)
- Autoregressive Models as Normalizing Flow Models
- Invertible CNNs (MintNet)
- Gaussianization flows
- Case Study: Probability density distillation for efficient learning and inference in Parallel Wavenet

# Designing invertible transformations

- NICE or Nonlinear Independent Components Estimation (Dinh et al., 2014) composes two kinds of invertible transformations: additive coupling layers and rescaling layers
- Real-NVP (Dinh et al., 2017)
- Inverse Autoregressive Flow (Kingma et al., 2016)
- Masked Autoregressive Flow (Papamakarios et al., 2017)
- I-resnet (Behrmann et al, 2018)
- Glow (Kingma et al, 2018)
- MintNet (Song et al., 2019)
- And many more

# NICE - Additive coupling layers

Partition the variables  $\mathbf{z}$  into two disjoint subsets, say  $\mathbf{z}_{1:d}$  and  $\mathbf{z}_{d+1:n}$  for any  $1 \leq d < n$

- Forward mapping  $\mathbf{z} \mapsto \mathbf{x}$ :
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$  (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d})$  ( $m_\theta(\cdot)$  is a neural network with parameters  $\theta$ ,  $d$  input units, and  $n - d$  output units)
- Inverse mapping  $\mathbf{x} \mapsto \mathbf{z}$ :
  - $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$  (identity transformation)
  - $\mathbf{z}_{d+1:n} = \mathbf{x}_{d+1:n} - m_\theta(\mathbf{x}_{1:d})$
- Jacobian of forward mapping:

$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & I_{n-d} \end{pmatrix}$$

$$\det(J) = 1$$

- **Volume preserving transformation** since determinant is 1.

# NICE - Rescaling layers

- Additive coupling layers are composed together (with arbitrary partitions of variables in each layer)
- Final layer of NICE applies a rescaling transformation
- Forward mapping  $\mathbf{z} \mapsto \mathbf{x}$ :

$$x_i = s_i z_i$$

where  $s_i > 0$  is the scaling factor for the  $i$ -th dimension.

- Inverse mapping  $\mathbf{x} \mapsto \mathbf{z}$ :

$$z_i = \frac{x_i}{s_i}$$

- Jacobian of forward mapping:

$$J = \text{diag}(\mathbf{s})$$

$$\det(J) = \prod_{i=1}^n s_i$$

# Samples generated via NICE



(a) Model trained on MNIST



(b) Model trained on TFD

# Samples generated via NICE



(c) Model trained on SVHN



(d) Model trained on CIFAR-10

# Real-NVP: Non-volume preserving extension of NICE

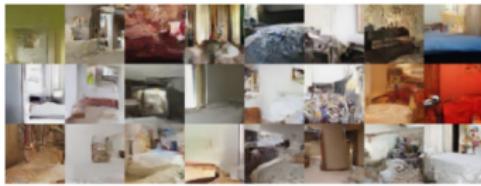
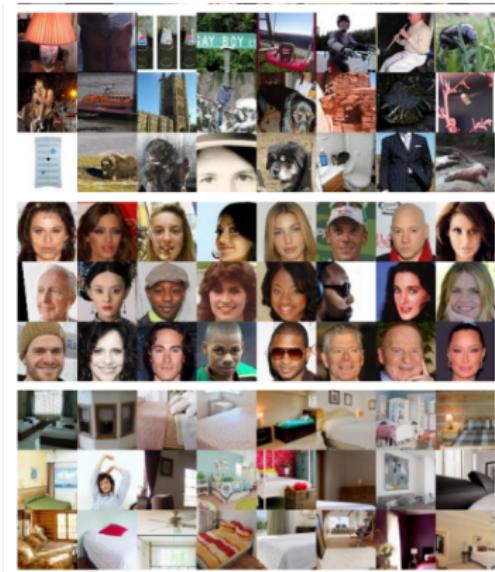
- Forward mapping  $\mathbf{z} \mapsto \mathbf{x}$ :
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$  (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\theta(\mathbf{z}_{1:d})) + \mu_\theta(\mathbf{z}_{1:d})$
  - $\mu_\theta(\cdot)$  and  $\alpha_\theta(\cdot)$  are both neural networks with parameters  $\theta$ ,  $d$  input units, and  $n - d$  output units [ $\odot$  denotes elementwise product]
- Inverse mapping  $\mathbf{x} \mapsto \mathbf{z}$ :
  - $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$  (identity transformation)
  - $\mathbf{z}_{d+1:n} = (\mathbf{x}_{d+1:n} - \mu_\theta(\mathbf{x}_{1:d})) \odot (\exp(-\alpha_\theta(\mathbf{x}_{1:d})))$
- Jacobian of forward mapping:

$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \text{diag}(\exp(\alpha_\theta(\mathbf{z}_{1:d}))) \end{pmatrix}$$

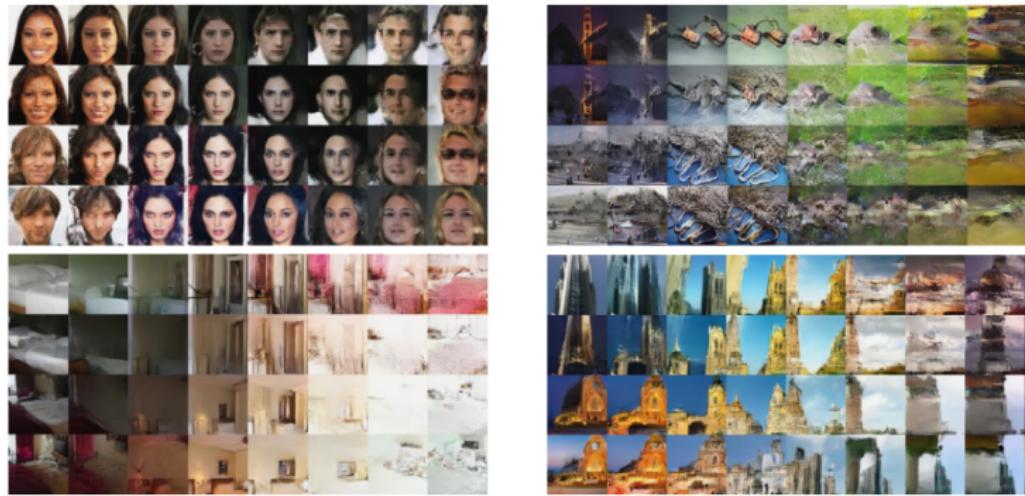
$$\det(J) = \prod_{i=d+1}^n \exp(\alpha_\theta(\mathbf{z}_{1:d})_i) = \exp \left( \sum_{i=d+1}^n \alpha_\theta(\mathbf{z}_{1:d})_i \right)$$

- **Non-volume preserving transformation** in general since determinant can be less than or greater than 1

# Samples generated via Real-NVP



# Latent space interpolations via Real-NVP



Using with four validation examples  $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \mathbf{z}^{(3)}, \mathbf{z}^{(4)}$ , define interpolated  $\mathbf{z}$  as:

$$\mathbf{z} = \cos\phi(\mathbf{z}^{(1)}\cos\phi' + \mathbf{z}^{(2)}\sin\phi') + \sin\phi(\mathbf{z}^{(3)}\cos\phi' + \mathbf{z}^{(4)}\sin\phi')$$

with manifold parameterized by  $\phi$  and  $\phi'$ .

# Continuous Autoregressive models as flow models

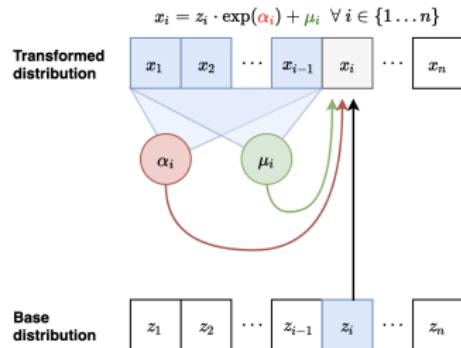
- Consider a Gaussian autoregressive model:

$$p(\mathbf{x}) = \prod_{i=1}^n p(x_i | \mathbf{x}_{<i})$$

such that  $p(x_i | \mathbf{x}_{<i}) = \mathcal{N}(\mu_i(x_1, \dots, x_{i-1}), \exp(\alpha_i(x_1, \dots, x_{i-1}))^2)$ . Here,  $\mu_i(\cdot)$  and  $\alpha_i(\cdot)$  are neural networks for  $i > 1$  and constants for  $i = 1$ .

- Sampler for this model:
  - Sample  $z_i \sim \mathcal{N}(0, 1)$  for  $i = 1, \dots, n$
  - Let  $x_1 = \exp(\alpha_1)z_1 + \mu_1$ . Compute  $\mu_2(x_1), \alpha_2(x_1)$
  - Let  $x_2 = \exp(\alpha_2)z_2 + \mu_2$ . Compute  $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
  - Let  $x_3 = \exp(\alpha_3)z_3 + \mu_3$ . ...
- Flow interpretation:** transforms samples from the standard Gaussian  $(z_1, z_2, \dots, z_n)$  to those generated from the model  $(x_1, x_2, \dots, x_n)$  via invertible transformations (parameterized by  $\mu_i(\cdot), \alpha_i(\cdot)$ )

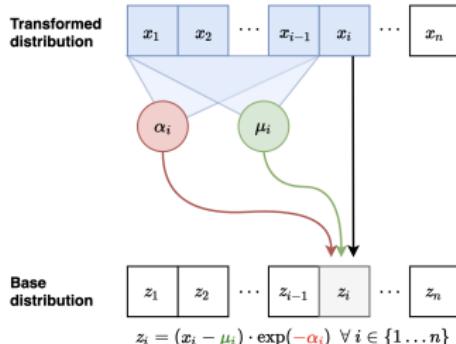
# Masked Autoregressive Flow (MAF)



- Forward mapping from  $\mathbf{z} \mapsto \mathbf{x}$ :
  - Let  $x_1 = \exp(\alpha_1)z_1 + \mu_1$ . Compute  $\mu_2(x_1), \alpha_2(x_1)$
  - Let  $x_2 = \exp(\alpha_2)z_2 + \mu_2$ . Compute  $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
- Sampling is sequential and slow (like autoregressive):  $O(n)$  time

Figure adapted from Eric Jang's blog

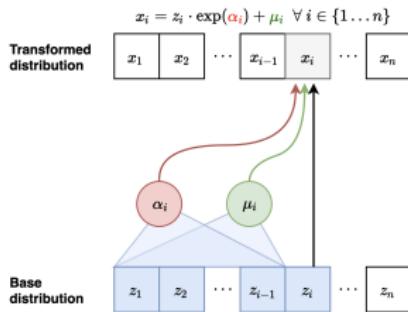
# Masked Autoregressive Flow (MAF)



- Inverse mapping from  $\mathbf{x} \mapsto \mathbf{z}$ :
  - Compute **all**  $\mu_i, \alpha_i$  (can be done in parallel using e.g., MADE)
  - Let  $z_1 = (x_1 - \mu_1) / \exp(\alpha_1)$  (scale and shift)
  - Let  $z_2 = (x_2 - \mu_2) / \exp(\alpha_2)$
  - Let  $z_3 = (x_3 - \mu_3) / \exp(\alpha_3) \dots$
- Jacobian is lower diagonal, hence efficient determinant computation
- Likelihood evaluation is easy and parallelizable (like MADE)
- Layers with different variable orderings can be stacked

Figure adapted from Eric Jang's blog

# Inverse Autoregressive Flow (IAF)



- Forward mapping from  $\mathbf{z} \mapsto \mathbf{x}$  (parallel):
  - Sample  $z_i \sim \mathcal{N}(0, 1)$  for  $i = 1, \dots, n$
  - Compute all  $\mu_i, \alpha_i$  (can be done in parallel)
  - Let  $x_1 = \exp(\alpha_1)z_1 + \mu_1$
  - Let  $x_2 = \exp(\alpha_2)z_2 + \mu_2 \dots$
- Inverse mapping from  $\mathbf{x} \mapsto \mathbf{z}$  (sequential):
  - Let  $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$ . Compute  $\mu_2(z_1), \alpha_2(z_1)$
  - Let  $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$ . Compute  $\mu_3(z_1, z_2), \alpha_3(z_1, z_2)$
- Fast to sample from, slow to evaluate likelihoods of data points (train)
- Note: Fast to evaluate likelihoods of a generated point (cache  $z_1, z_2, \dots, z_n$ )

Figure adapted from Eric Jang's blog

# IAF is inverse of MAF

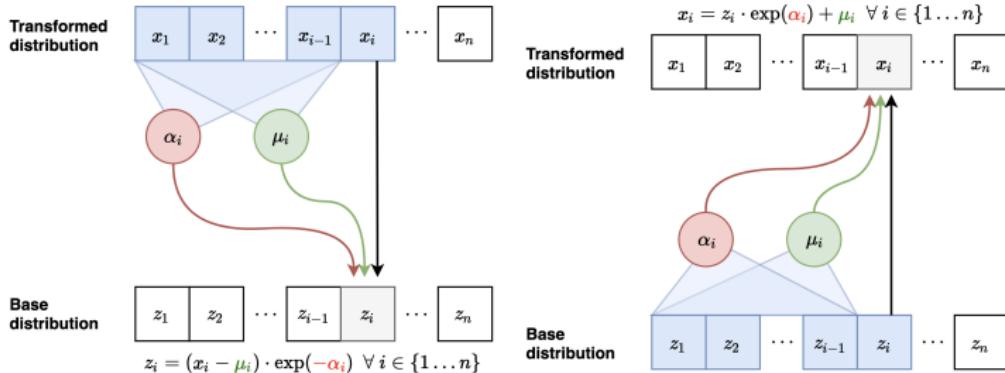


Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

- Interchanging  $z$  and  $x$  in the inverse transformation of MAF gives the forward transformation of IAF
- Similarly, forward transformation of MAF is inverse transformation of IAF

Figure adapted from Eric Jang's blog

# IAF vs. MAF

- Computational tradeoffs
  - MAF: Fast likelihood evaluation, slow sampling
  - IAF: Fast sampling, slow likelihood evaluation
- MAF more suited for training based on MLE, density estimation
- IAF more suited for real-time generation
- Can we get the best of both worlds?

# Parallel Wavenet

- Two part training with a teacher and student model
- Teacher is parameterized by MAF. Teacher can be efficiently trained via MLE
- Once teacher is trained, initialize a student model parameterized by IAF. Student model cannot efficiently evaluate density for external datapoints but allows for efficient sampling
- **Key observation:** IAF can also efficiently evaluate densities of its own generations (via caching the noise variates  $z_1, z_2, \dots, z_n$ )

- **Probability density distillation:** Student distribution is trained to minimize the KL divergence between student ( $s$ ) and teacher ( $t$ )

$$D_{\text{KL}}(s, t) = E_{\mathbf{x} \sim s}[\log s(\mathbf{x}) - \log t(\mathbf{x})]$$

- Evaluating and optimizing Monte Carlo estimates of this objective requires:
  - Samples  $\mathbf{x}$  from student model (IAF)
  - Density of  $\mathbf{x}$  assigned by student model
  - Density of  $\mathbf{x}$  assigned by teacher model (MAF)
- All operations above can be implemented efficiently

# Parallel Wavenet: Overall algorithm

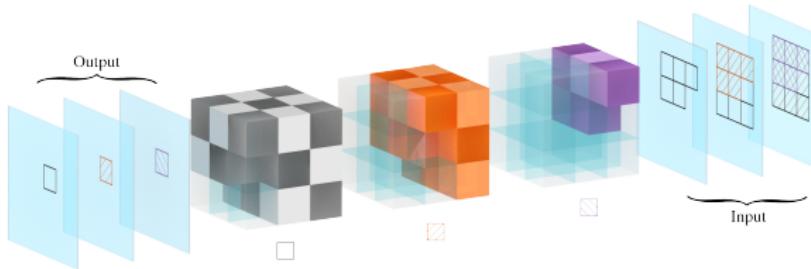
- Training
  - Step 1: Train teacher model (MAF) via MLE
  - Step 2: Train student model (IAF) to minimize KL divergence with teacher
- Test-time: Use student model for testing
- Improves sampling efficiency over original Wavenet (vanilla autoregressive model) by 1000x!

## MintNet (Song et al., 2019)

- MintNet: Building invertible neural networks with masked convolutions.
- A regular convolutional neural network is powerful, but it is not invertible and its Jacobian determinant is expensive.
- We can instead use masked convolutions like in autoregressive models to enforce ordering (like PixelCNN)
- Because of the ordering, the Jacobian matrix is triangular and the determinant is efficient to compute.
- If all the diagonal elements of the Jacobian matrix are (strictly) positive, the transformation is invertible.

# MintNet (Song et al., 2019)

- Illustration of a masked convolution with 3 filters and kernel size  $3 \times 3$ .



- Solid checkerboard cubes inside each filter represent unmasked weights, while the transparent blue blocks represent the weights that have been masked out.
- The receptive field of each filter on the input feature maps is indicated by regions shaded with the pattern (the colored square) below the corresponding filter.

# MintNet (Song et al., 2019)

- Uncurated samples on MNIST, CIFAR-10, and ImageNet 32x32 datasets

6 4 7 7 0 6 1 8  
1 3 3 3 1 1 1 6  
8 1 7 6 2 0 1 9  
4 9 6 9 3 4 0 0  
4 7 7 0 4 1 1 2  
0 6 2 9 3 3 8 9  
5 9 6 0 0 3 4 5  
1 2 2 6 1 2 3 7

(a) MNIST



(b) CIFAR-10



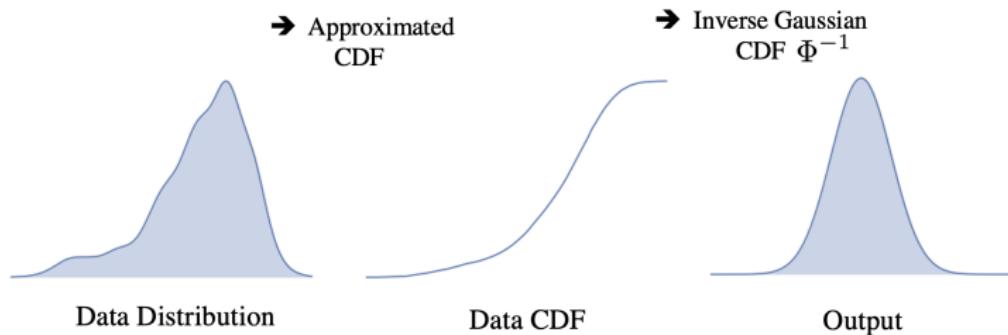
(c) ImageNet-32×32

# Gaussianization Flows (Meng et al., 2020)

- Let  $X = f_\theta(Z)$  be a flow model with Gaussian prior  $Z \sim \mathcal{N}(0, I) = p_Z$ , and let  $\tilde{X} \sim p_{\text{data}}$  be a random vector distributed according to the true data distribution.
- Flow models are trained with maximum likelihood to minimize the KL divergence  $D_{\text{KL}}(p_{\text{data}} \parallel p_\theta(x)) = D_{\text{KL}}(p_{\tilde{X}} \parallel p_X)$ . Gaussian samples transformed through  $f_\theta$  should be distributed as the data.
- It can be shown that  $D_{\text{KL}}(p_{\tilde{X}} \parallel p_X) = D_{\text{KL}}\left(p_{f_\theta^{-1}(\tilde{X})} \parallel p_{f_\theta^{-1}(X)}\right) = D_{\text{KL}}\left(p_{f_\theta^{-1}(\tilde{X})} \parallel p_Z\right)$ . Data samples transformed through  $f_\theta^{-1}$  should be distributed as Gaussian
- How can we achieve this?

# Gaussianization Flows (Meng et al., 2020)

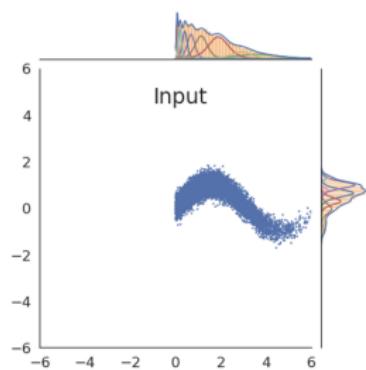
- Let's start with a 1D example. Let the data  $\tilde{X}$  have density  $p_{data}$  and cumulative density function (CDF)  $F_{data}(a) = \int_{-\infty}^a p_{data}$ .
- Inverse CDF trick:** If  $F_{data}$  is known, we can sample from  $p_{data}$  via  $\tilde{X} = F_{data}^{-1}(U)$  where  $U \in [0, 1]$  is a uniform random variable.



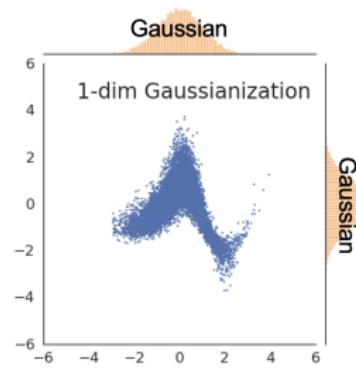
- This means that  $U = F_{data}(\tilde{X})$  is uniform. We can transform  $U$  into a Gaussian using the inverse CDF trick:  $\Phi^{-1}(U) = \Phi^{-1}(F_{data}(\tilde{X}))$ .
- The invertible transformation  $\Phi^{-1} \circ F_{data}$  Gaussianizes the data!

# Gaussianization Flows (Meng et al., 2020)

- Step 1: Dimension-wise Gaussianization (Jacobian is a diagonal matrix and is tractable)



Input data

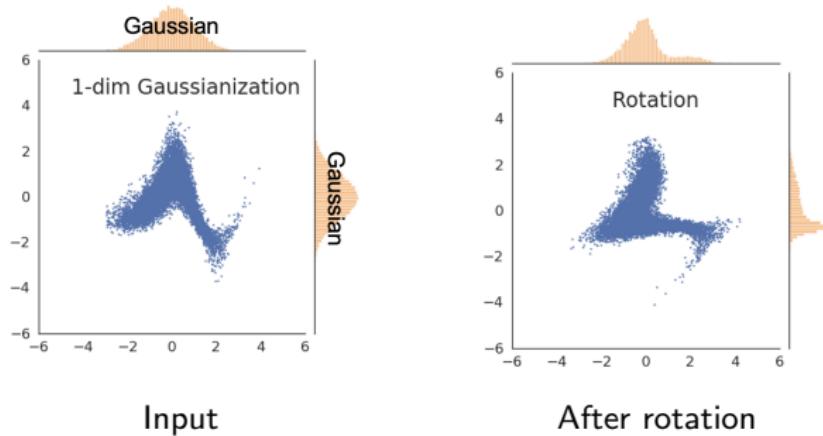


Dimension-wise Gaussianization

Note: Even though each dimension is marginally Gaussian, they are **not** jointly Gaussian. Aside: Approximating this with a Gaussian prior is a shallow flow model known as a copula model (Sklar, 1959).

# Gaussianization Flows (Meng et al., 2020)

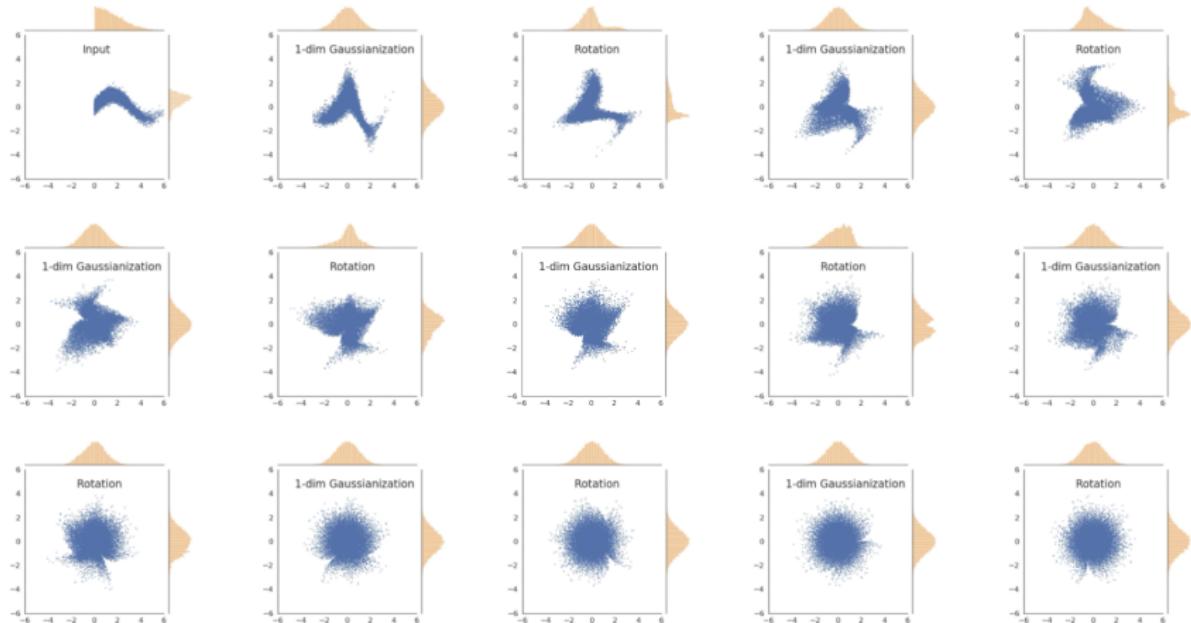
- Step 2: apply a rotation matrix to the transformed data (Jacobian is an orthogonal matrix and is tractable)



- Note:  $\mathcal{N}(\mathbf{0}, \mathbf{I})$  is rotationally invariant

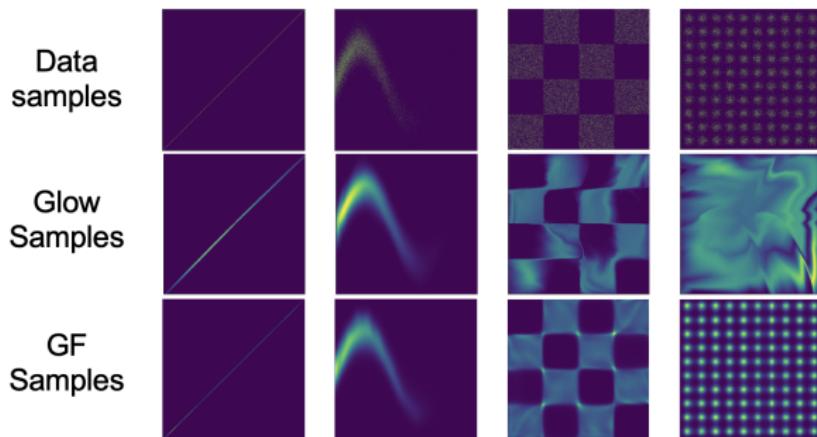
# Gaussianization Flows (Meng et al., 2020)

- Gaussianization flow: repeat Step 1 and Step 2 (stacking learnable Gaussian copula). Transform data into a normal distribution.



# Experiments: Density Estimation

- Density estimation on 2D datasets



# Summary of Normalizing Flow Models

- Transform simple distributions into more complex distributions via change of variables
- Jacobian of transformations should have tractable determinant for efficient learning and density estimation
- Computational tradeoffs in evaluating forward and inverse transformations