

# COMP547

## DEEP UNSUPERVISED LEARNING

Lecture #6 – Normalizing Flow Models



**KOÇ**  
**UNIVERSITY**

Aykut Erdem // Koç University // Spring 2022



# Previously on COMP547

- Motivation
- Simple generative models: histograms
- Parameterized distributions and maximum likelihood
- Autoregressive Models
  - Recurrent Neural Nets
  - Masking-based Models



# Our Goal Today

- How to fit a density model  $p_{\theta}(x)$  with continuous  $x \in \mathbb{R}^n$
- What do we want from this model?
  - Good fit to the training data (really, the underlying distribution!)
  - For new  $x$ , ability to evaluate  $p_{\theta}(x)$
  - Ability to sample from  $p_{\theta}(x)$
  - And, ideally, a latent representation that's meaningful

# Our Goal Today

- How to fit a density model  $p_{\theta}(x)$  with **continuous**  $x \in \mathbb{R}^n$
- What do we want from this model?
  - Good fit to the training data (really, the underlying distribution!)
  - For new  $x$ , ability to evaluate  $p_{\theta}(x)$
  - Ability to sample from  $p_{\theta}(x)$
  - And, ideally, **a latent representation** that's meaningful

**Differences from Autoregressive Models from last lecture**

# Lecture overview

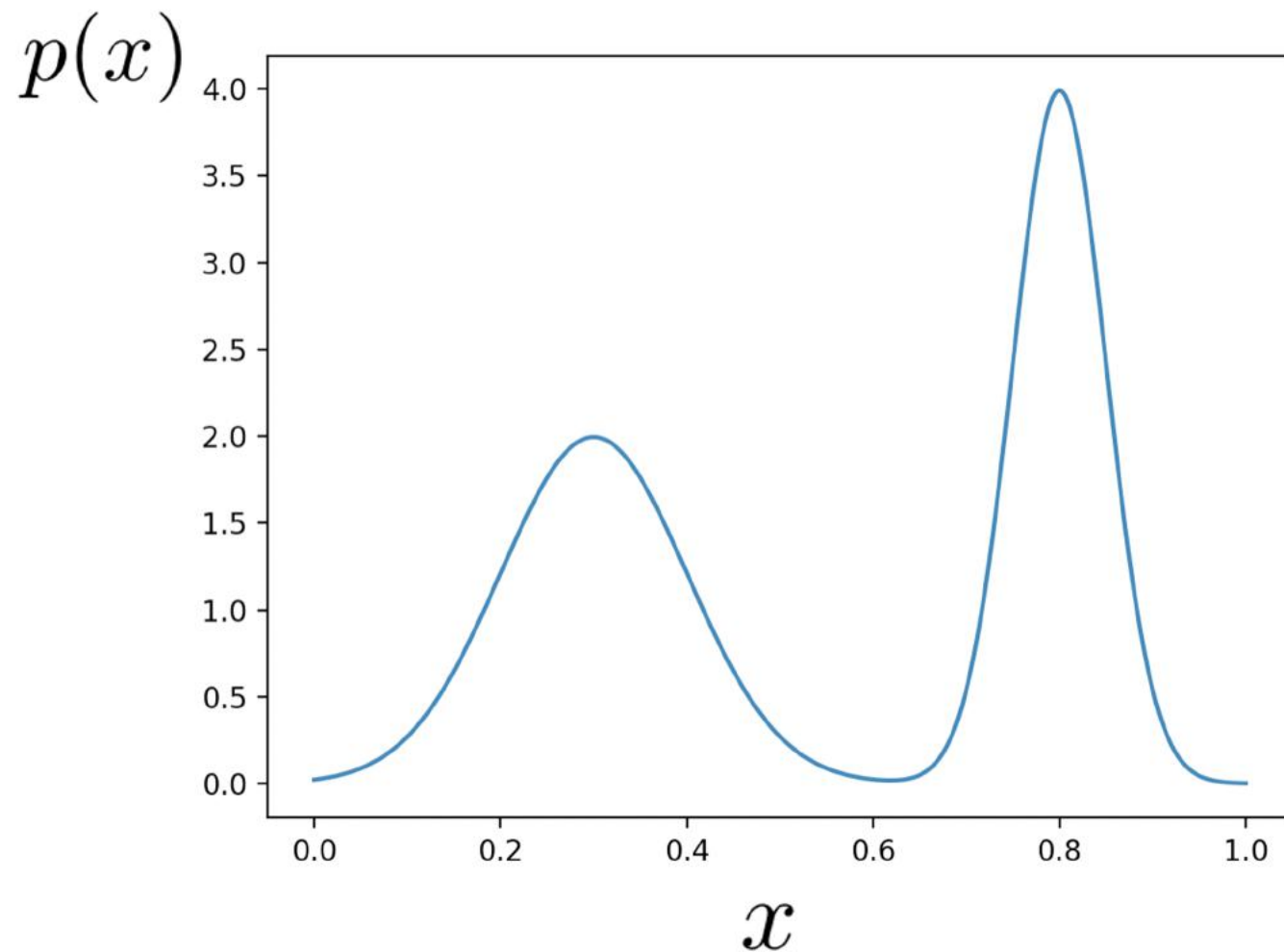
- Foundations of Flows (1-D)
- 2-D Flows
- N-D Flows
- Dequantization

**Disclaimer:** Much of the material and slides for this lecture were borrowed from  
—Pieter Abbeel, Peter Chen, Jonathan Ho, Aravind Srinivas' Berkeley CS294-158 class  
—Chin-Wei Huang slides on Normalizing Flows

# Lecture overview

- Foundations of Flows (1-D)
- 2-D Flows
- N-D Flows
- Dequantization

# Quick Refresher: Probability Density Models



$$P(x \in [a, b]) = \int_a^b p(x) dx$$

# How to fit a density model?

## Continuous data

0.22159854, 0.84525919, 0.09121633, 0.364252 , 0.30738086,  
0.32240615, 0.24371194, 0.22400792, 0.39181847, 0.16407012,  
0.84685229, 0.15944969, 0.79142357, 0.6505366 , 0.33123603,  
0.81409325, 0.74042126, 0.67950372, 0.74073271, 0.37091554,  
0.83476616, 0.38346571, 0.33561352, 0.74100048, 0.32061713,  
0.09172335, 0.39037131, 0.80496586, 0.80301971, 0.32048452,  
0.79428266, 0.6961708 , 0.20183965, 0.82621227, 0.367292 ,  
0.76095756, 0.10125199, 0.41495427, 0.85999877, 0.23004346,  
0.28881973, 0.41211802, 0.24764836, 0.72743029, 0.20749136,  
0.29877091, 0.75781455, 0.29219608, 0.79681589, 0.86823823,  
0.29936483, 0.02948181, 0.78528968, 0.84015573, 0.40391632,  
0.77816356, 0.75039186, 0.84709016, 0.76950307, 0.29772759,  
0.41163966, 0.24862007, 0.34249207, 0.74363912, 0.38303383, ...

## Maximum Likelihood:

$$\max_{\theta} \sum_i \log p_{\theta}(x^{(i)})$$

## Equivalently:

$$\min_{\theta} \mathbb{E}_x [-\log p_{\theta}(x)]$$

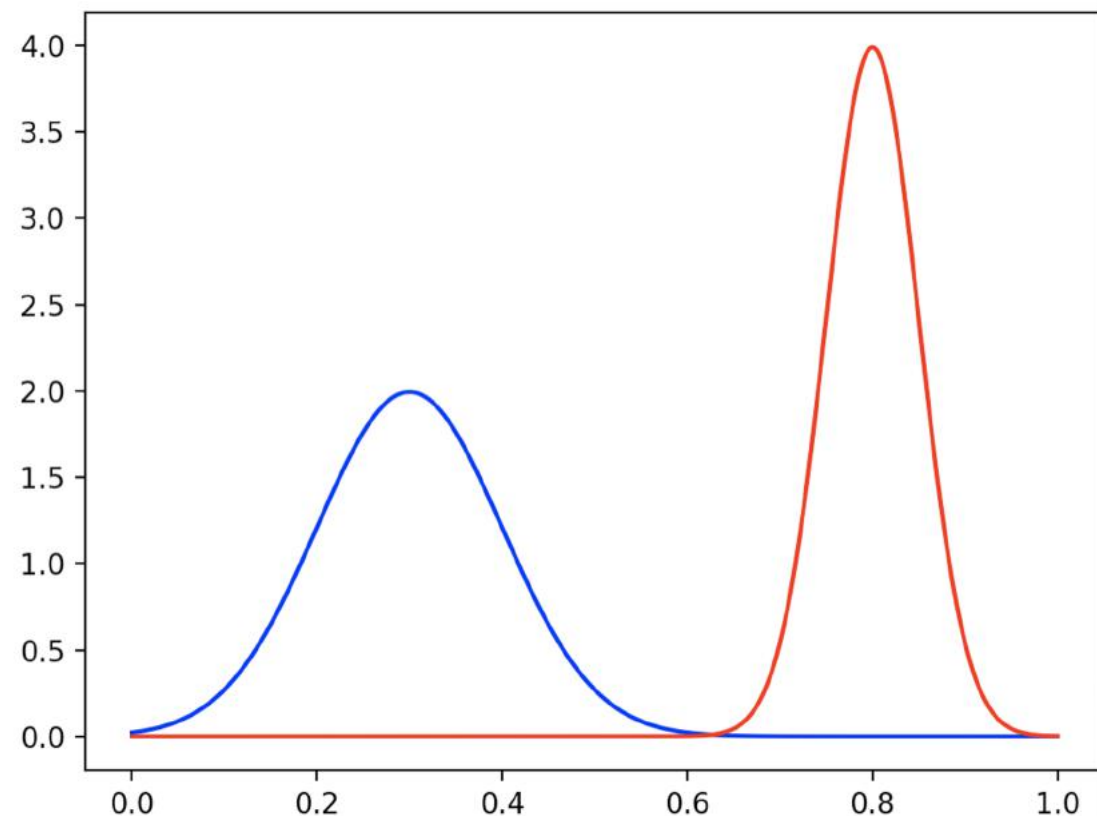


# Example Density Model: Mixtures of Gaussians

$$p_{\theta}(x) = \sum_{i=1}^k \pi_i \mathcal{N}(x; \mu_i, \sigma_i^2)$$

Parameters: means and variances of components, mixture weights

$$\theta = (\pi_1, \dots, \pi_k, \mu_1, \dots, \mu_k, \sigma_1, \dots, \sigma_k)$$



# Aside on Mixtures of Gaussians

Do mixtures of Gaussians work for high-dimensional data?

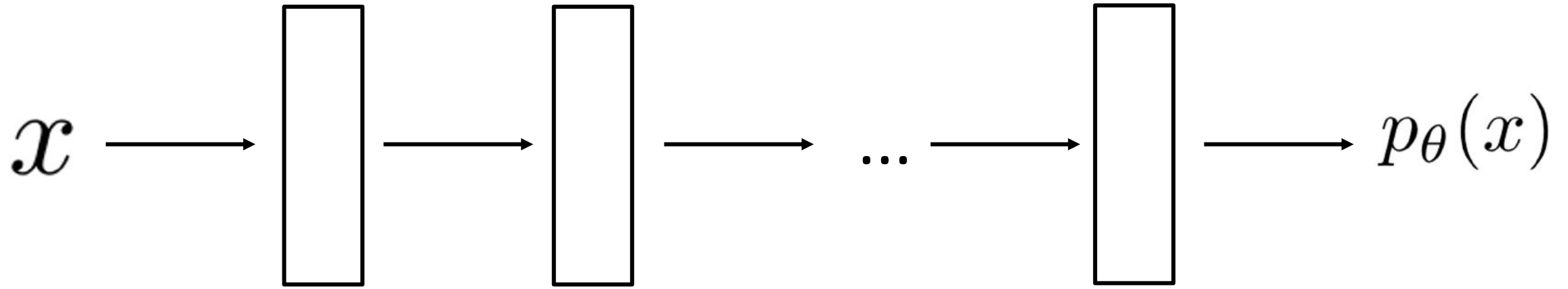
Not really. The sampling process is:

1. Pick a cluster center
2. Add Gaussian noise

Imagine this for modeling natural images! The only way a realistic image can be generated is if it is a cluster center, i.e. if it is already stored directly in the parameters.



# How to fit a general density model?

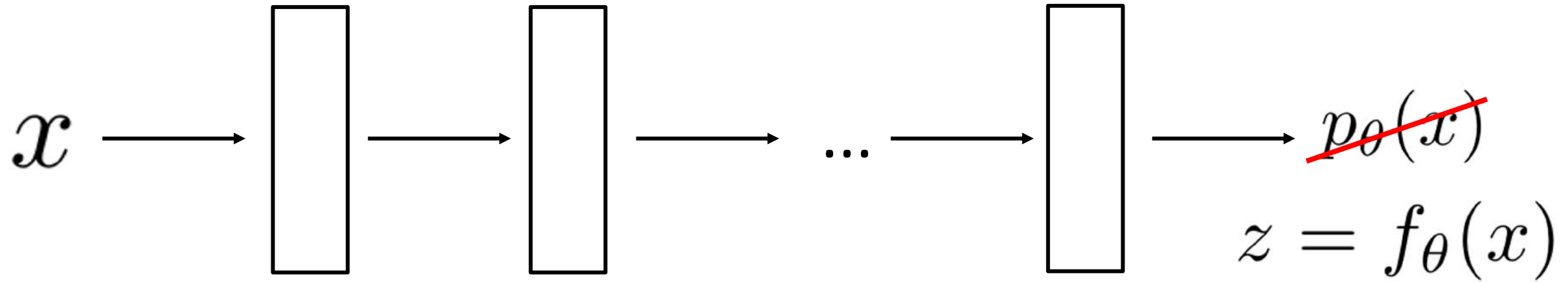


- How to ensure proper distribution?

$$\int_{-\infty}^{+\infty} p_\theta(x) dx = 1 \quad p_\theta(x) \geq 0 \quad \forall x$$

- How to sample? Easily achieved for discrete data, using softmax
- Latent representation? What about continuous data?

# Flows: Main Idea

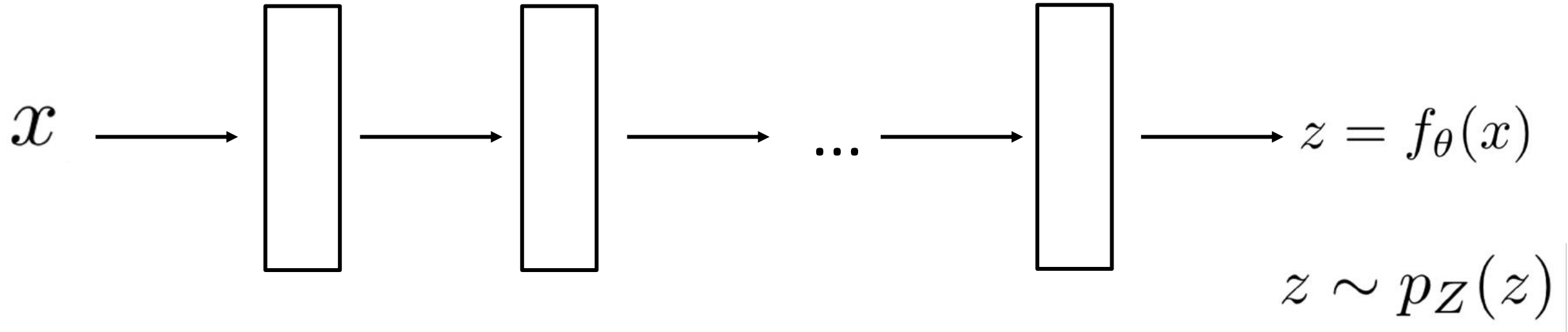


Generally:  $z \sim p_Z(z)$

Normalizing Flow:  $z \sim \mathcal{N}(0, 1)$

How to train? How to evaluate  $p_\theta(x)$ ? How to sample?

# Flows: Training



$$\max_{\theta} \sum_i \log p_{\theta}(x^{(i)})$$

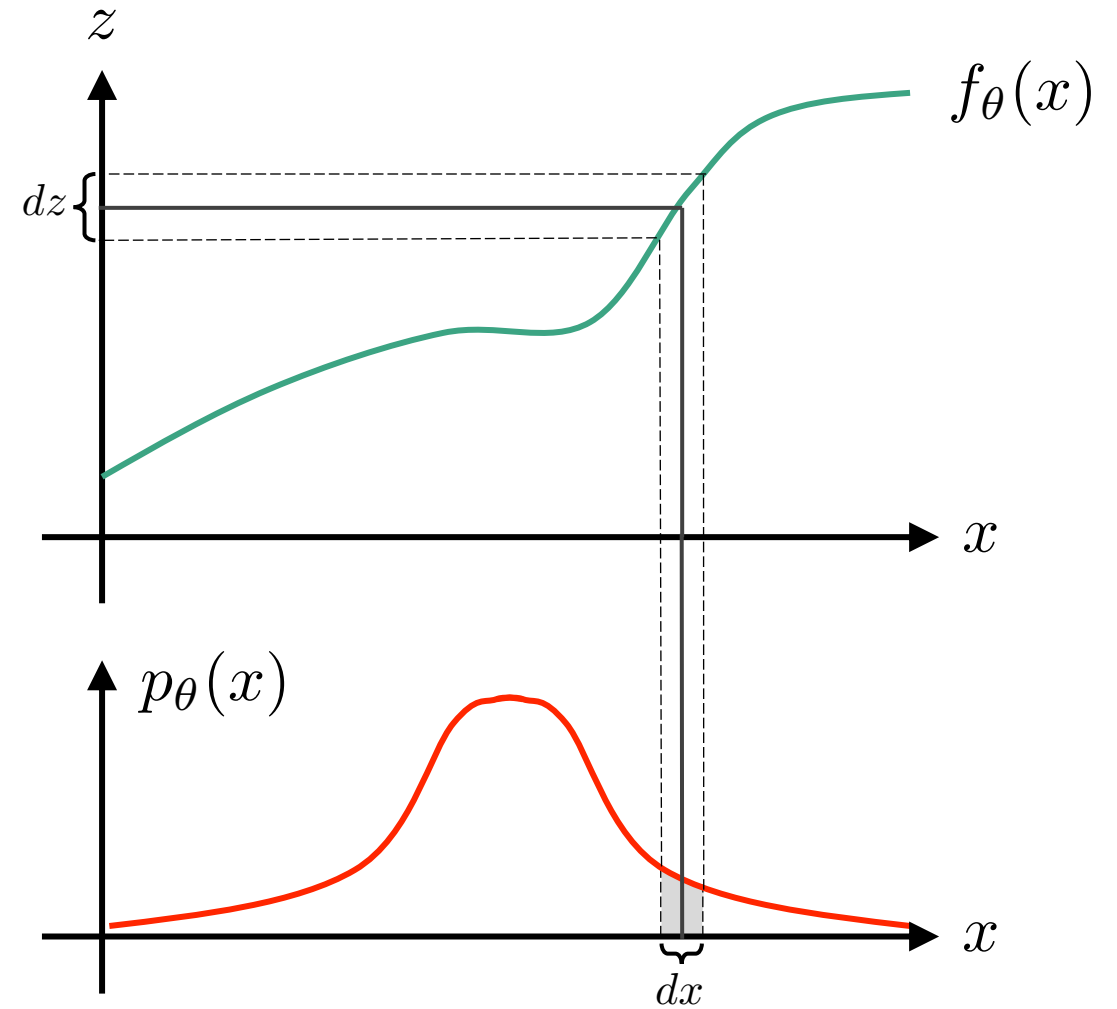


# Change of Variables

$$z = f_{\theta}(x)$$

$$p_{\theta}(x) dx = p(z) dz$$

$$p_{\theta}(x) = p(f_{\theta}(x)) \left| \frac{\partial f_{\theta}(x)}{\partial x} \right|$$

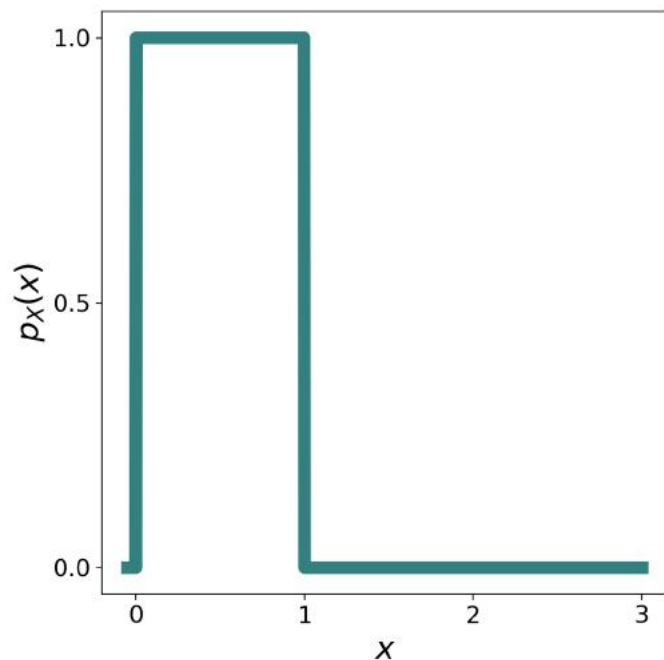


**Note:** requires  $f_{\theta}$  invertible & differentiable

# Change of Variable Density Needs to Be Normalized

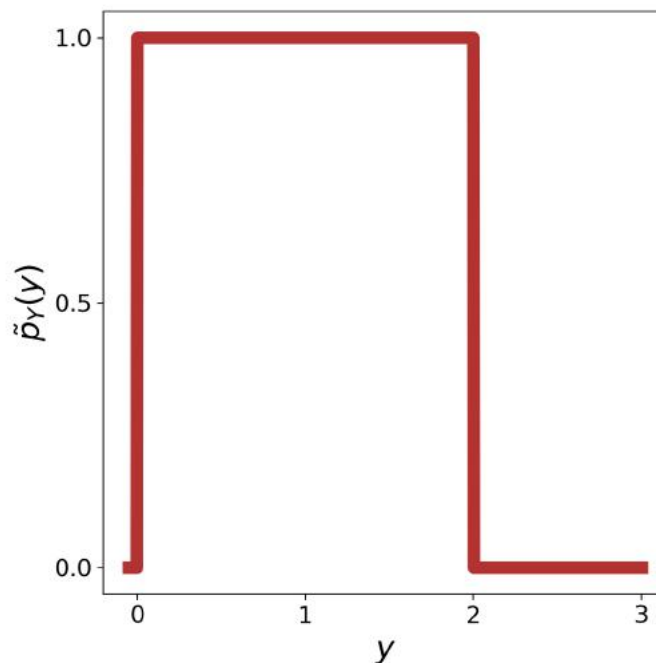
$$X \sim p_X$$

$$p_X(x) = \begin{cases} 1 & \text{for } 0 \leq x \leq 1 \\ 0 & \text{else} \end{cases}$$

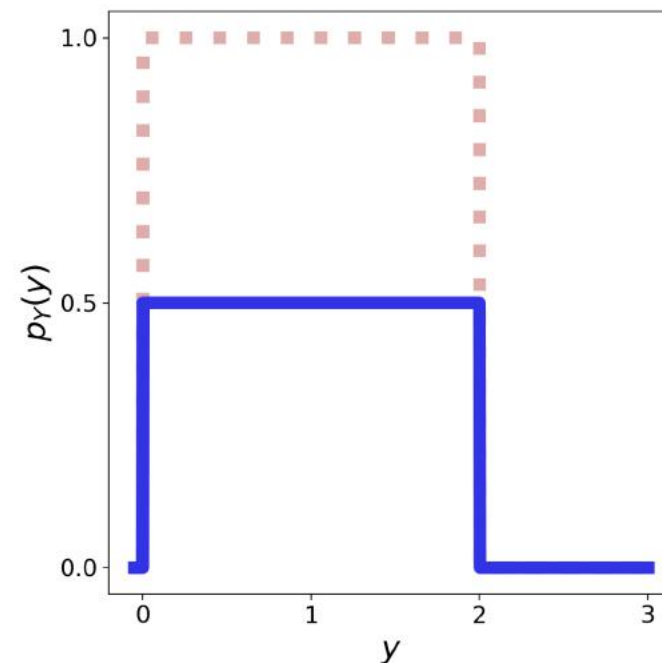


$$Y := 2X$$

$$\tilde{p}_Y(y) = p_X(y/2)$$



$$p_Y(y) = p_X(y/2)/2$$



# Flows: Training

$$\max_{\theta} \sum_i \log p_{\theta}(x^{(i)})$$

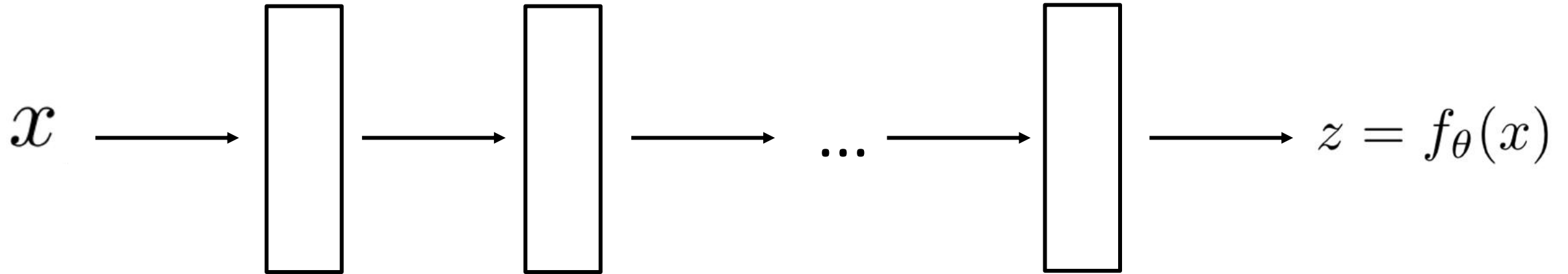
$$z^{(i)} = f_{\theta}(x^{(i)})$$

$$\begin{aligned} p_{\theta}(x^{(i)}) &= p_Z(z^{(i)}) \left| \frac{\partial z}{\partial x}(x^{(i)}) \right| \\ &= p_Z(f_{\theta}(x^{(i)})) \left| \frac{\partial f_{\theta}}{\partial x}(x^{(i)}) \right| \end{aligned}$$

$$\max_{\theta} \sum_i \log p_{\theta}(x^{(i)}) = \max_{\theta} \sum_i \log p_Z(f_{\theta}(x^{(i)})) + \log \left| \frac{\partial f_{\theta}}{\partial x}(x^{(i)}) \right|$$

→ assuming we have an expression for  $p_Z$ ,  
this can be optimized with Stochastic Gradient Descent

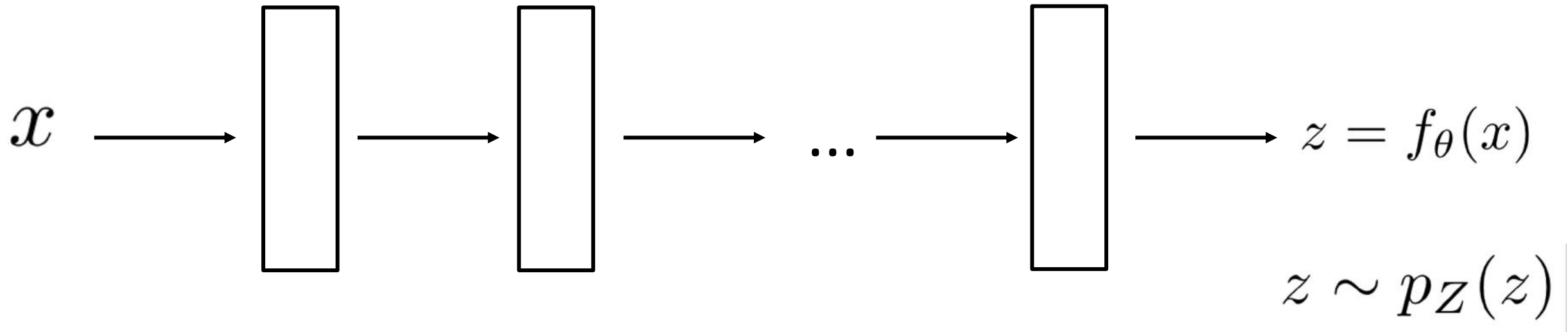
# Flows: Sampling



Step 1: sample  $z \sim p_Z(z)$

Step 2:  $x = f_{\theta}^{-1}(z)$

# What do we need to keep in mind for $f$ ?



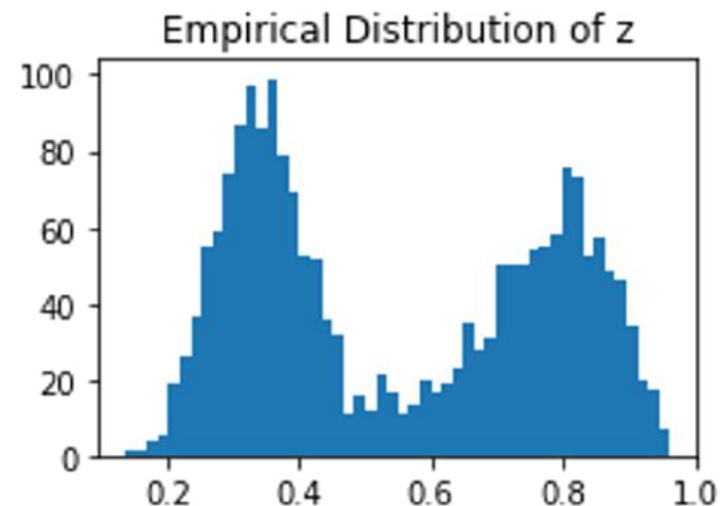
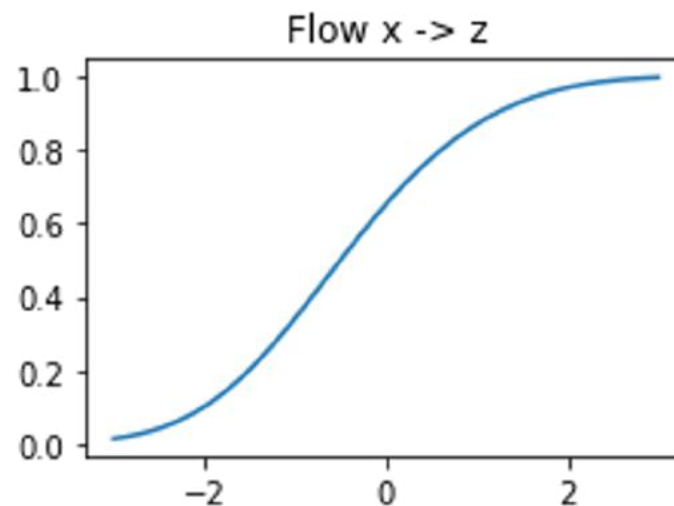
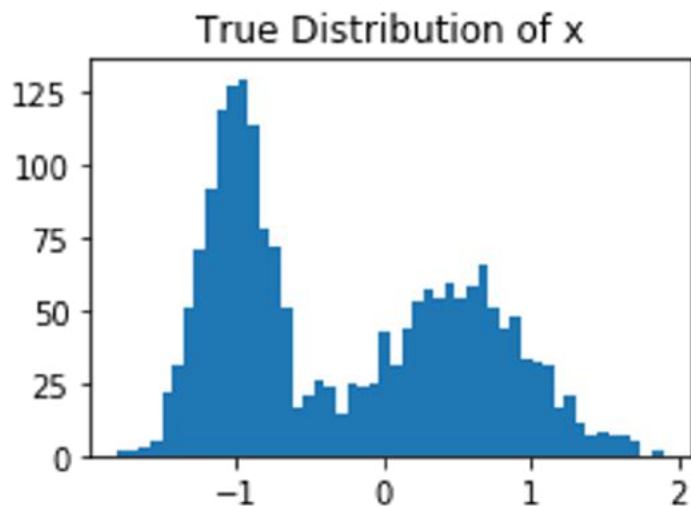
Recall, change of variable formula requires

- $f_{\theta}$  Invertible & differentiable

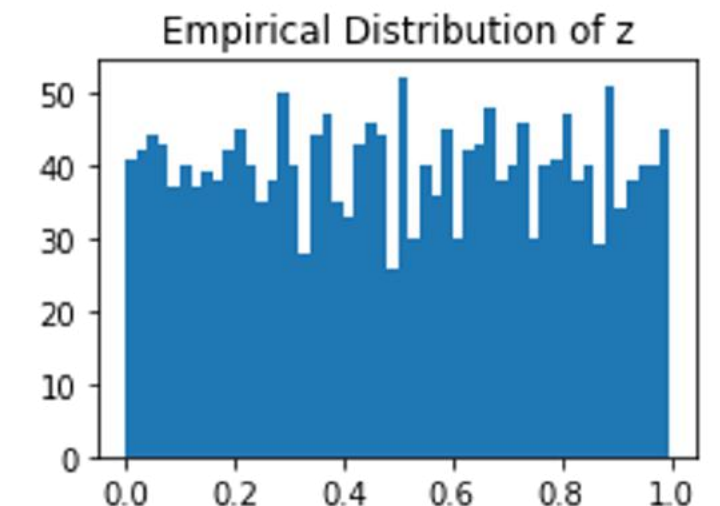
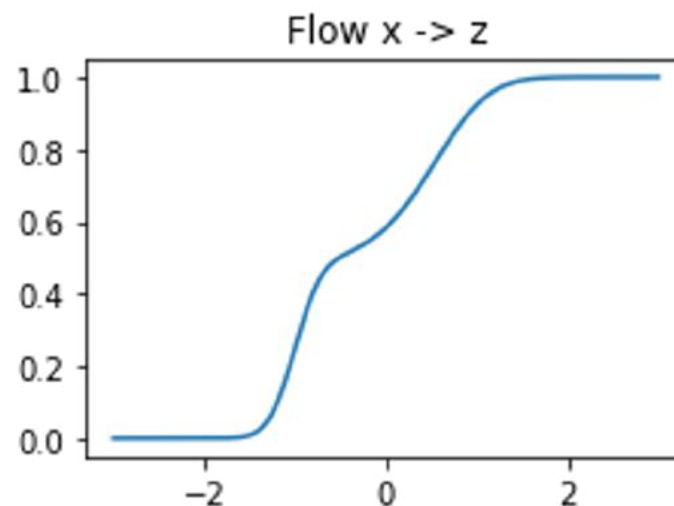
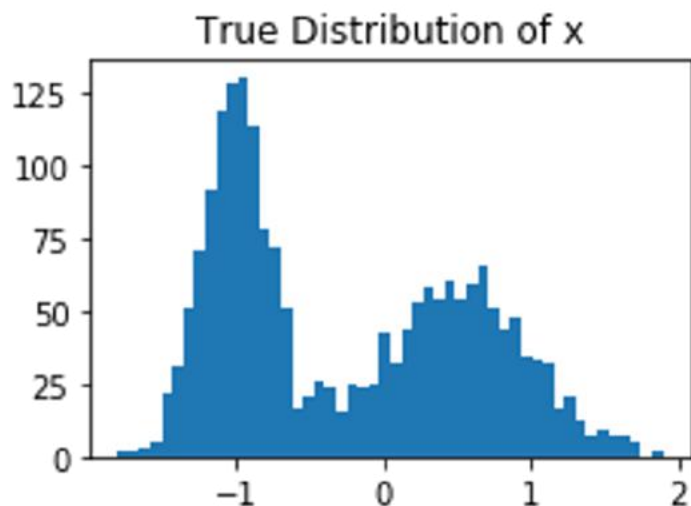


# Example: Flow to Uniform z

Before training



After training



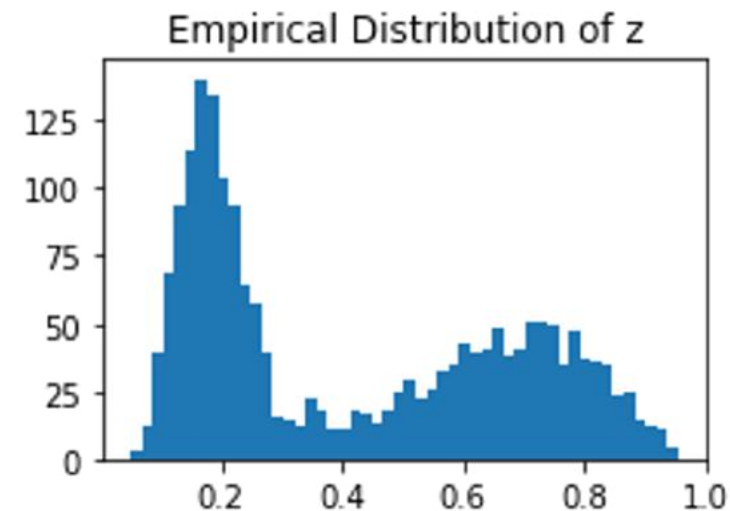
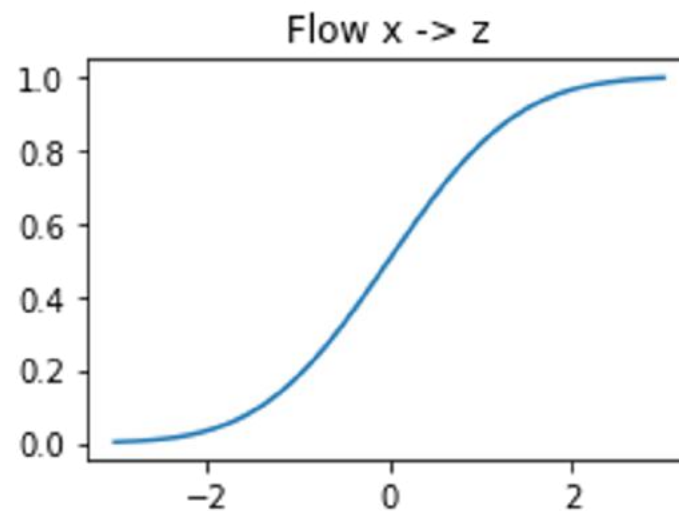
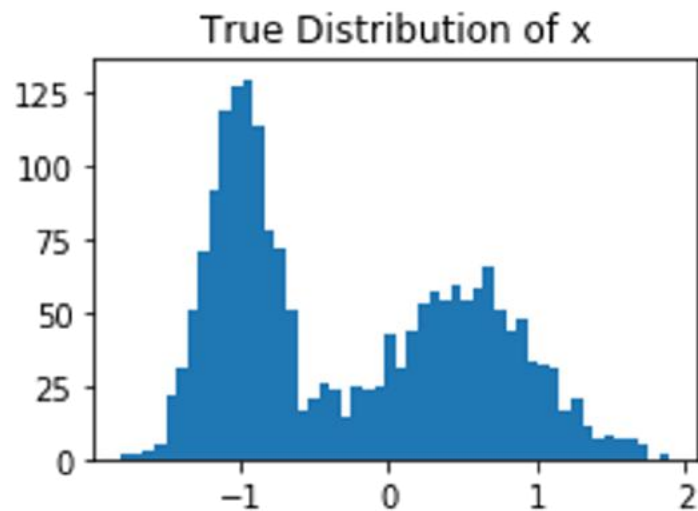
True distribution of x

Flow x  $\rightarrow$  z

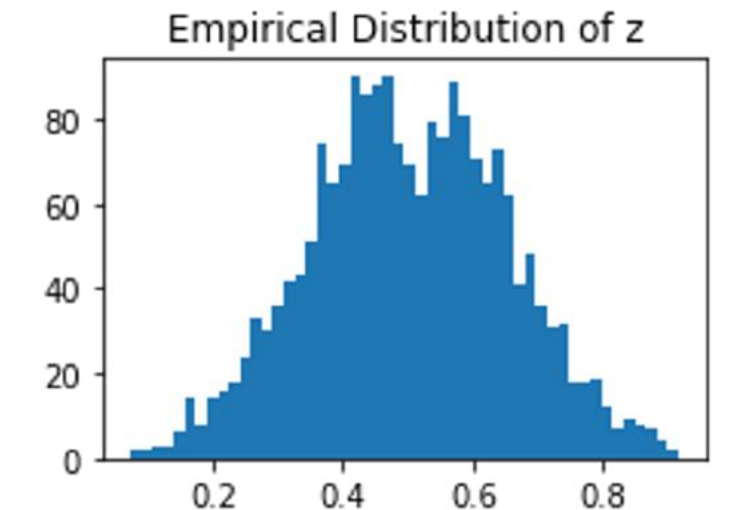
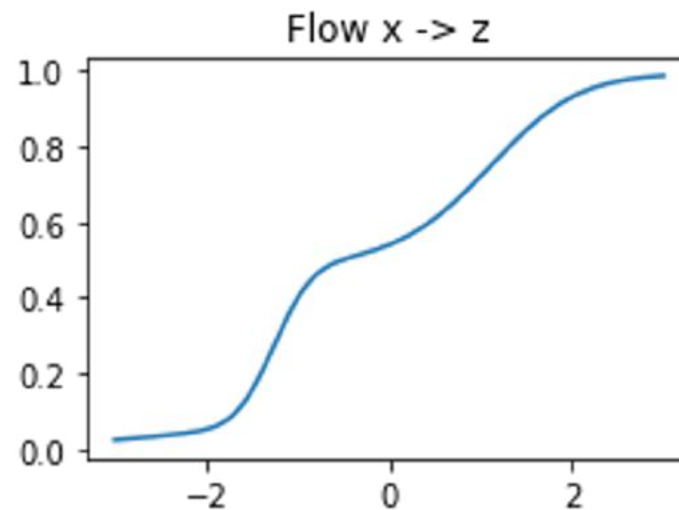
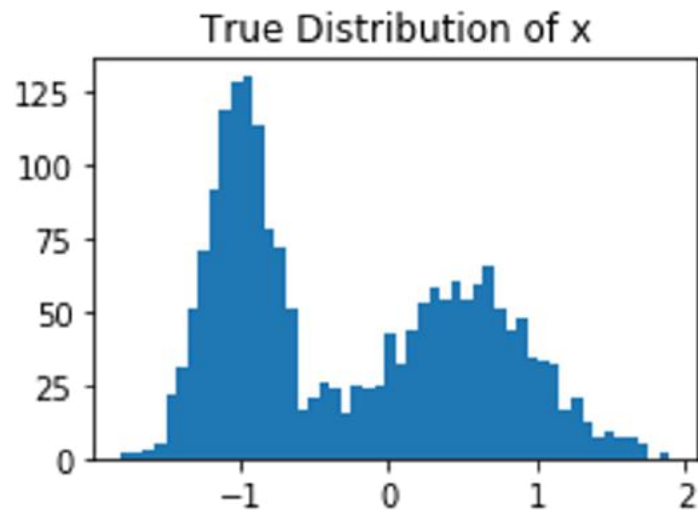
Empirical distribution of z

# Example: Flow to Beta(5,5) z

Before training



After training



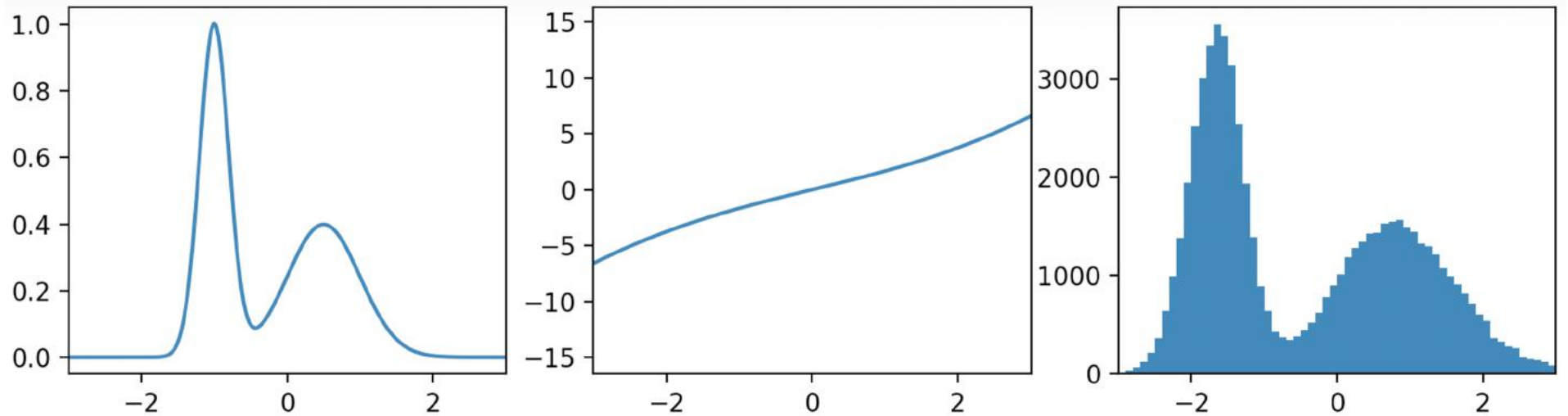
True distribution of x

Flow x  $\rightarrow$  z

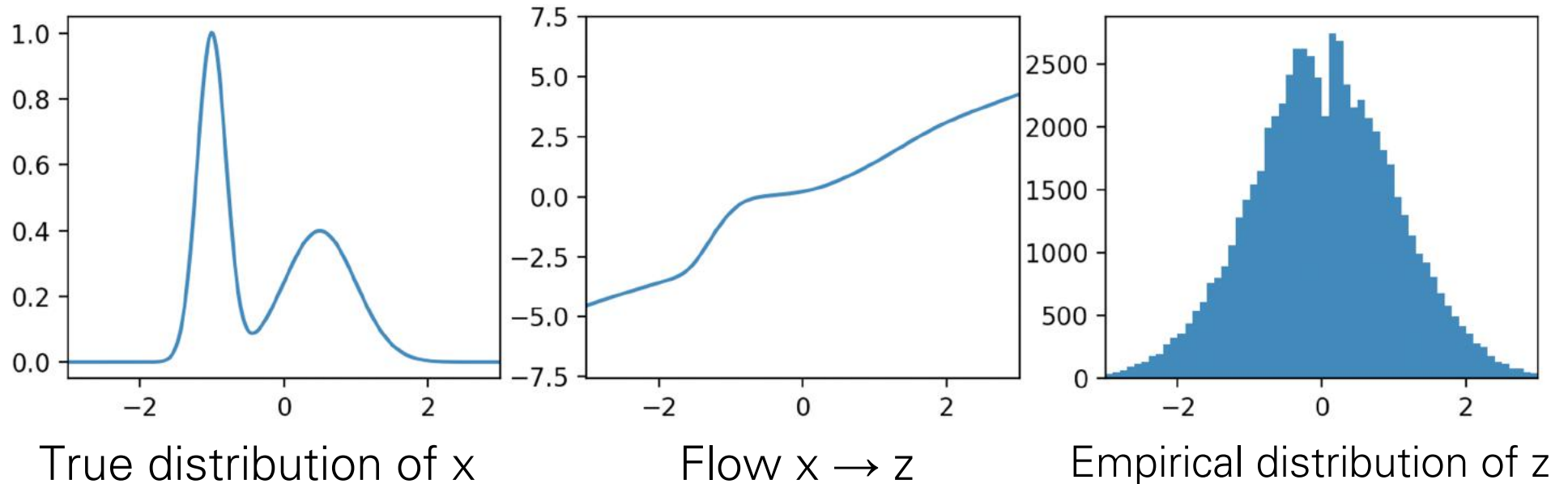
Empirical distribution of z

# Example: Flow to Gaussian $z$

Before training



After training



# Practical Parameterizations of Flows

**Requirement:** Invertible and Differentiable

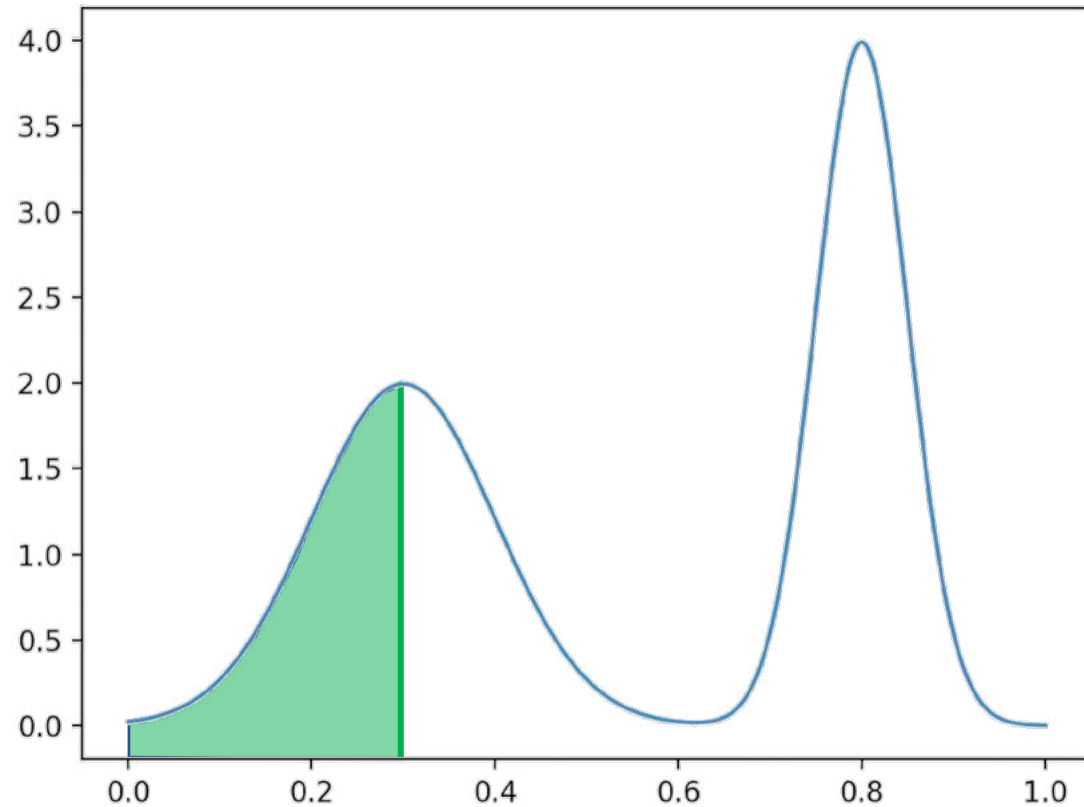
- Cumulative Density Functions
  - E.g. Gaussian mixture density, mixture of logistics
- Neural Net
  - If each layer flow, then sequencing of layers = flow
  - Each layer:
    - ReLU?
    - Sigmoid?
    - Tanh?

# How general are flows?

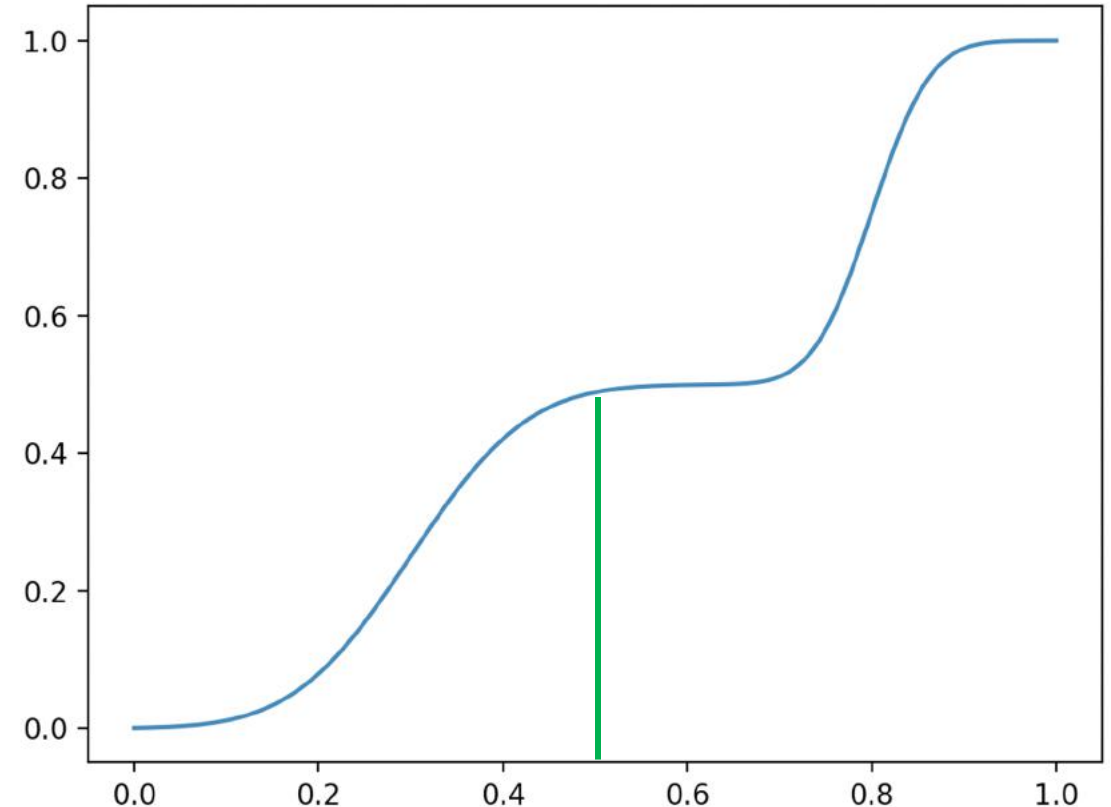
- Can every (smooth) distribution be represented by a (normalizing) flow? [considering 1-D for now]



# Refresher: Cumulative Density Function (CDF)



$p_{\theta}(x)$



$$f_{\theta}(x) = \int_{-\infty}^x p_{\theta}(t) dt$$

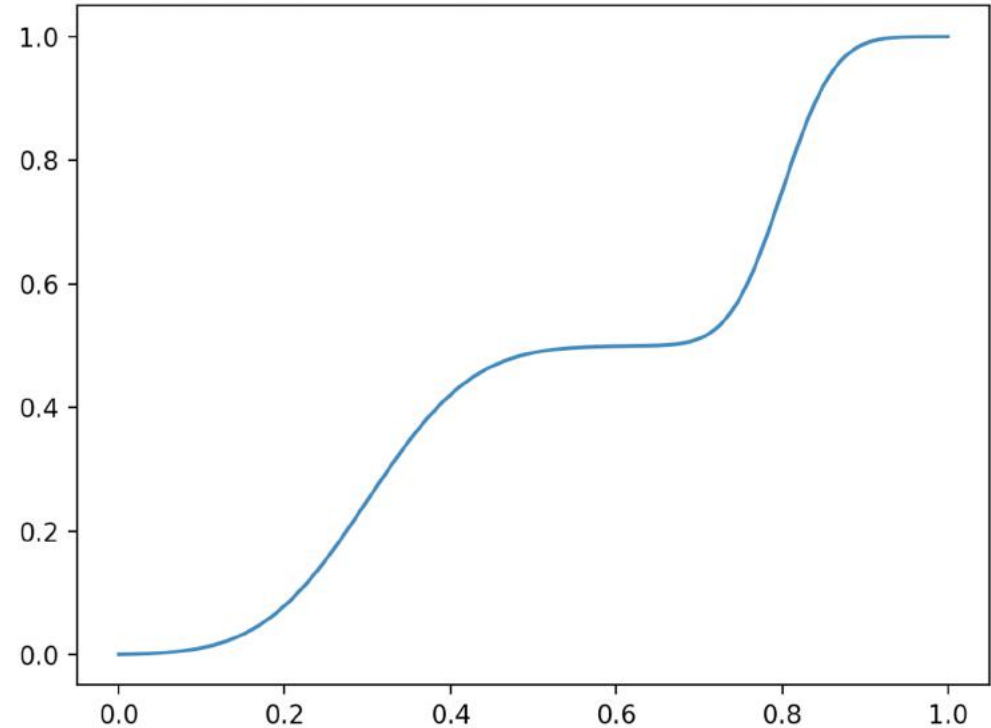
# Sampling via inverse CDF

Sampling from the model:

$$z \sim \text{Uniform}([0, 1])$$

$$x = f_{\theta}^{-1}(z)$$

The CDF is an invertible, differentiable map from data to  $[0, 1]$



$$f_{\theta}(x) = \int_{-\infty}^x p_{\theta}(t) dt$$

# How general are flows?

- CDF turns any density into uniform
- Inverse flow is flow

$$x \xrightarrow{\text{CDF}} u$$

$$z \xrightarrow{\text{CDF}} u$$

$$x \xrightarrow{\text{CDF}} u \xrightarrow{\text{CDF}} z$$

→ can turn any (smooth)  $p(x)$  into any (smooth)  $p(z)$

# Lecture overview

- Foundations of Flows (1-D)
- **2-D Flows**
- N-D Flows
- Dequantization

# 2-D Autoregressive Flow

$$x_1 \rightarrow z_1 = f_\theta(x_1)$$

$$x_2 \rightarrow z_2 = f_\phi(x_1, x_2)$$

$$\max_{\theta, \phi} \sum_i \log p_{z_1}(f_\theta(x_1)) + \log \left| \frac{dz_1}{dx_1} \right| + \log p_{z_2}(f_\phi(x_1, x_2)) + \log \left| \frac{dz_2}{dx_2} \right|$$

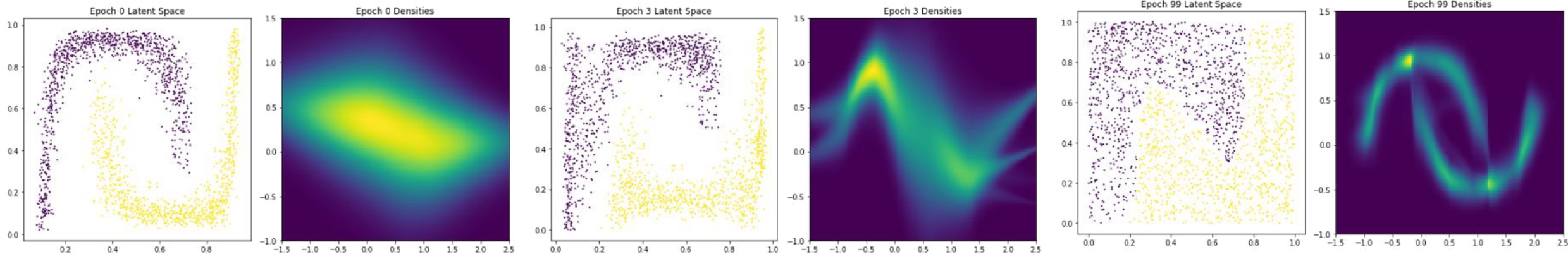
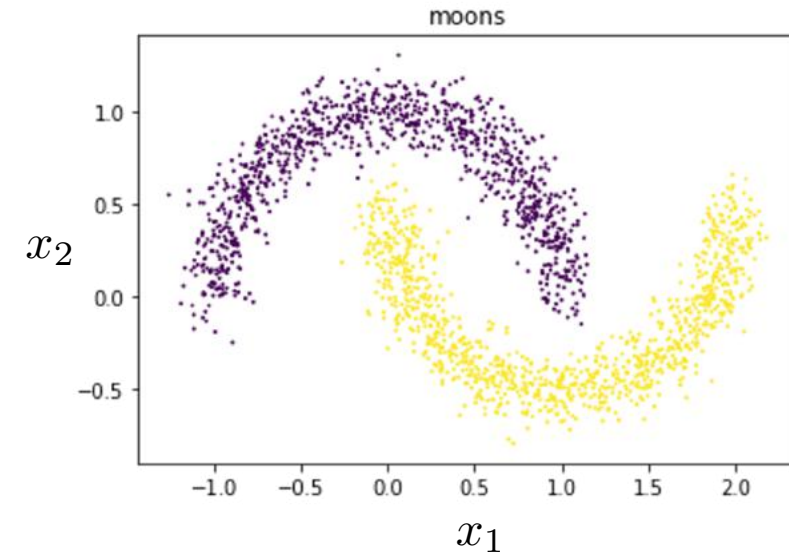
$$\frac{dz_1}{dx_1} = \frac{df_\theta(x_1)}{dx_1}, \quad \frac{dz_2}{dx_2} = \frac{df_\phi(x_1, x_2)}{dx_2}$$



# 2-D Autoregressive Flow: Two Moons

## Architecture:

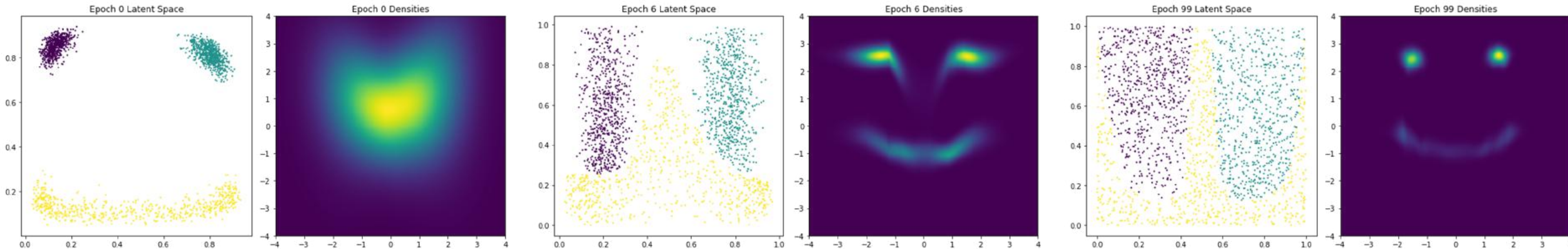
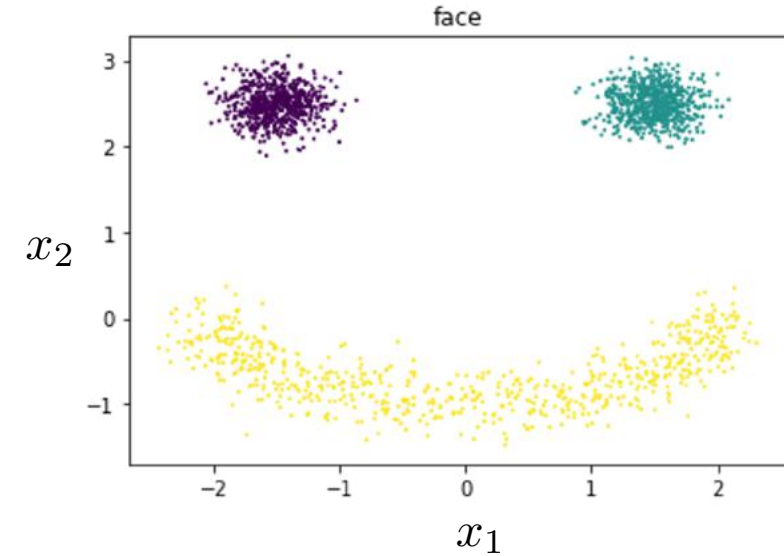
- Base distribution:  $\text{Uniform}[0,1]^2$
- $x_1$ : mixture of 5 Gaussians
- $x_2$ : mixture of 5 Gaussians, conditioned on  $x_1$



# 2-D Autoregressive Flow: Face

## Architecture:

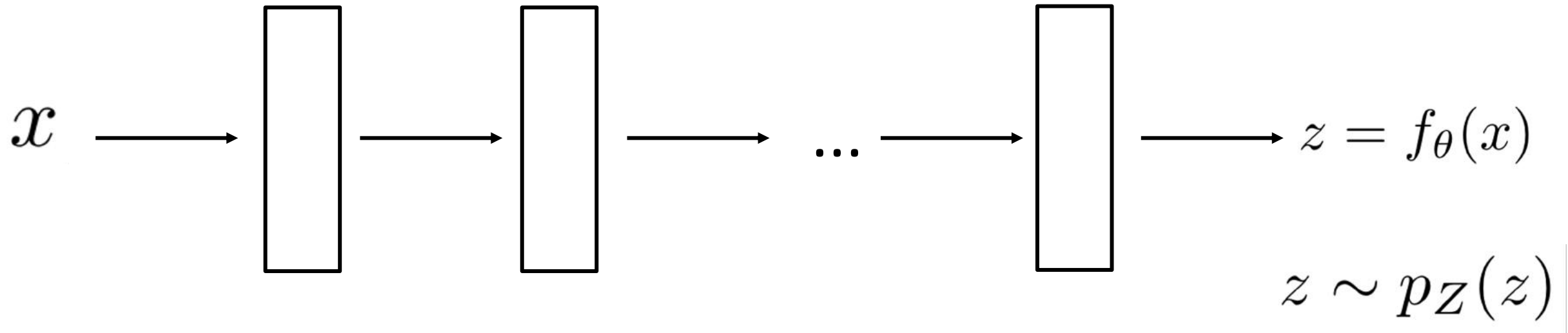
- Base distribution:  $\text{Uniform}[0,1]^2$
- $x_1$ : mixture of 5 Gaussians
- $x_2$ : mixture of 5 Gaussians, conditioned on  $x_1$



# Lecture overview

- Foundations of Flows (1-D)
- 2-D Flows
- **N-D Flows**
- Dequantization

# Recap: Normalizing Flows



- $f_{\theta}$  invertible & differentiable

Training objective:

$$\max_{\theta} \sum_i \log p_{\theta}(x^{(i)}) = \max_{\theta} \sum_i \log p_Z(f_{\theta}(x^{(i)})) + \log \left| \frac{\partial f_{\theta}}{\partial x}(x^{(i)}) \right|$$

# Lecture overview

- Foundations of Flows (1-D)
- 2-D Flows
- **N-D Flows**
  - **Autoregressive Flows and Inverse Autoregressive Flows**
  - RealNVP (like) architectures
  - Glow, Flow++, FFJORD
- Dequantization

# Autoregressive flows

- The sampling process of a Bayes net is a flow
  - If autoregressive, this flow is called an **autoregressive flow**

$$\begin{array}{lll} x_1 \sim p_\theta(x_1) & x_1 = f_\theta^{-1}(z_1) & z_1 = f_\theta(x_1) \\ x_2 \sim p_\theta(x_2|x_1) & x_2 = f_\theta^{-1}(z_2; x_1) & z_2 = f_\theta(x_1, x_2) \\ x_3 \sim p_\theta(x_3|x_1, x_2) & x_3 = f_\theta^{-1}(z_3; x_1, x_2) & z_3 = f_\theta(x_1, x_2, x_3) \end{array}$$

- Sampling is an **invertible** mapping from  $z$  to  $x$

# Autoregressive flows

- How to fit autoregressive flows?

- Map  $\mathbf{x}$  to  $\mathbf{z}$
- Fully parallelizable

$$p_{\theta}(\mathbf{x}) = p(f_{\theta}(\mathbf{x})) \left| \det \frac{\partial f_{\theta}(\mathbf{x})}{\partial \mathbf{x}} \right|$$

- Notice

- $\mathbf{x} \rightarrow \mathbf{z}$  has the same structure as the **log likelihood** computation of an autoregressive model
- $\mathbf{z} \rightarrow \mathbf{x}$  has the same structure as the **sampling** procedure of an autoregressive model

$$z_1 = f_{\theta}(x_1)$$

$$z_2 = f_{\theta}(x_2; x_1)$$

$$z_3 = f_{\theta}(x_3; x_1, x_2)$$

$$x_1 = f_{\theta}^{-1}(z_1)$$

$$x_2 = f_{\theta}^{-1}(z_2; x_1)$$

$$x_3 = f_{\theta}^{-1}(z_3; x_1, x_2)$$



# Inverse autoregressive flows

- The inverse of an autoregressive flow is also a flow, called the **inverse autoregressive flow (IAF)**
  - $\mathbf{x} \rightarrow \mathbf{z}$  has the same structure as the **sampling** in an autoregressive model
  - $\mathbf{z} \rightarrow \mathbf{x}$  has the same structure as **log likelihood** computation of an autoregressive model. So, **IAF sampling is fast**

$$z_1 = f_{\theta}^{-1}(x_1)$$

$$x_1 = f_{\theta}(z_1)$$

$$z_2 = f_{\theta}^{-1}(x_2; z_1)$$

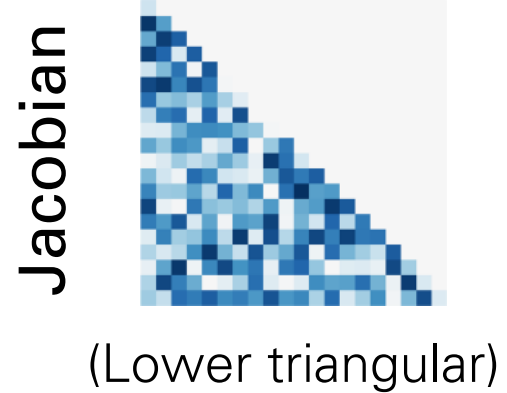
$$x_2 = f_{\theta}(z_2; z_1)$$

$$z_3 = f_{\theta}^{-1}(x_3; z_1, z_2)$$

$$x_3 = f_{\theta}(z_3; z_1, z_2)$$

# AF vs IAF

- Autoregressive flow
  - **Fast** evaluation of  $p(x)$  for arbitrary  $x$
  - **Slow** sampling
- Inverse autoregressive flow
  - **Slow** evaluation of  $p(x)$  for arbitrary  $x$ , so training directly by maximum likelihood is slow.
  - **Fast** sampling
  - **Fast** evaluation of  $p(x)$  if  $x$  is a sample
- There are models (Parallel WaveNet, IAF-VAE) that exploit IAF's fast sampling



# AF and IAF

Naively, both end up being as deep as the number of variables!

- E.g. 1MP image  $\rightarrow$  1M layers/sampling steps...

Can do parameter sharing as in Autoregressive Models from previous lecture [e.g. RNN, masking]

# Lecture overview

- Foundations of Flows (1-D)
- 2-D Flows
- **N-D Flows**
  - Autoregressive Flows and Inverse Autoregressive Flows
  - **RealNVP (like) architectures**
  - Glow, Flow++, FFJORD
- Dequantization

# Change of MANY variables

For  $z \sim p(z)$ , sampling process  $f^{-1}$  linearly transforms a small cube  $dz$  to a small parallelepiped  $dx$ . Probability is conserved:

$$p(x) = p(z) \frac{\text{vol}(dz)}{\text{vol}(dx)} = p(z) \left| \det \frac{dz}{dx} \right|$$

**Intuition:**  $x$  is likely if it maps to a “large” region in  $z$  space

# Flow models: training

Change-of-variables formula lets us compute the density over  $\mathbf{x}$ :

$$p_{\theta}(\mathbf{x}) = p(f_{\theta}(\mathbf{x})) \left| \det \frac{\partial f_{\theta}(\mathbf{x})}{\partial \mathbf{x}} \right|$$

Train with maximum likelihood:

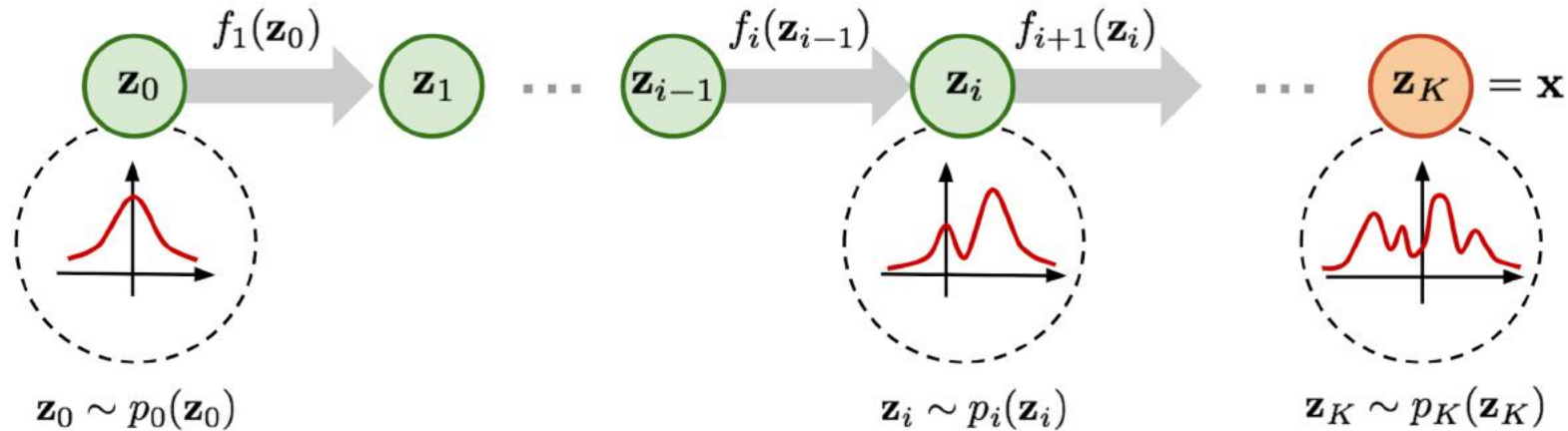
$$\arg \min_{\theta} \mathbb{E}_{\mathbf{x}} [-\log p_{\theta}(\mathbf{x})] = \mathbb{E}_{\mathbf{x}} \left[ -\log p(f_{\theta}(\mathbf{x})) - \log \det \left| \frac{\partial f_{\theta}(\mathbf{x})}{\partial \mathbf{x}} \right| \right]$$

**New key requirement:** the Jacobian determinant must be easy to calculate and differentiate!

# Chaining Invertible Mappings

$$f = f_S \circ \cdots \circ f_2 \circ f_1$$

$$f(x) = f_S(\cdots f_2(f_1(x)))$$



$$\frac{\partial f(x)}{\partial x} = \frac{f_S(x_{S-1})}{\partial x_{S-1}} \cdots \frac{f_2(x_1)}{\partial x_1} \frac{f_1(x_0)}{\partial x_0} \quad \begin{array}{l} x_s = f_s(x_{s-1}) \\ x_0 = x \end{array}$$

Chain rule

$$\det \left( \frac{\partial f(x)}{\partial x} \right) = \det \left( \frac{f_S(x_{S-1})}{\partial x_{S-1}} \right) \cdots \det \left( \frac{f_2(x_1)}{\partial x_1} \right) \det \left( \frac{f_1(x_0)}{\partial x_0} \right)$$

Determinant of  
matrix product



# Constructing flows: composition

- Flows can be composed

$$x \rightarrow f_1 \rightarrow f_2 \rightarrow \dots f_k \rightarrow z$$

$$z = f_k \circ \dots \circ f_1(x)$$

$$x = f_1^{-1} \circ \dots \circ f_k^{-1}(z)$$

$$\log p_\theta(x) = \log p_\theta(z) + \sum_{i=1}^k \log \left| \det \frac{\partial f_i}{\partial f_{i-1}} \right|$$

- Easy way to increase expressiveness

# Affine flows

- Another name for affine flow: multivariate Gaussian.
  - Parameters: an invertible matrix  $A$  and a vector  $b$
  - $f(x) = A^{-1}(x - b)$
- Sampling:  $x = Az + b$ , where  $z \sim \mathcal{N}(0, I)$   $x \sim \mathcal{N}(b, AA^T)$
- Log likelihood is expensive when dimension is large.
  - The Jacobian of  $f$  is  $A^{-1}$
  - Log likelihood involves calculating  $\det(A)$

# Elementwise flows

$$f_{\theta}((x_1, \dots, x_d)) = (f_{\theta}(x_1), \dots, f_{\theta}(x_d))$$

- Lots of freedom in elementwise flow
  - Can use elementwise affine functions or CDF flows.
- The Jacobian is diagonal, so the determinant is easy to evaluate.

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \text{diag}(f'_{\theta}(x_1), \dots, f'_{\theta}(x_d))$$

$$\det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \prod_{i=1}^d f'_{\theta}(x_i)$$

# NICE/RealNVP

Affine coupling layer

- Split variables in half:  $\mathbf{x}_{1:d/2}$ ,  $\mathbf{x}_{d/2+1:d}$

$$\mathbf{z}_{1:d/2} = \mathbf{x}_{1:d/2}$$

$$\mathbf{z}_{d/2:d} = \mathbf{x}_{d/2:d} \cdot \exp(s_{\theta}(\mathbf{x}_{1:d/2})) + t_{\theta}(\mathbf{x}_{1:d/2})$$

- Invertible! Note that  $s_{\theta}$  and  $t_{\theta}$  can be arbitrary neural nets with **no restrictions**.
  - Think of them as **data-parameterized elementwise flows**.

# NICE/RealNVP

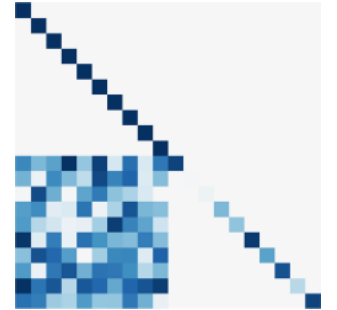
- It also has a tractable Jacobian determinant

$$\mathbf{z}_{1:d/2} = \mathbf{x}_{1:d/2}$$

$$\mathbf{z}_{d/2:d} = \mathbf{x}_{d/2:d} \cdot s_{\theta}(\mathbf{x}_{1:d/2}) + t_{\theta}(\mathbf{x}_{1:d/2})$$

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{bmatrix} I & 0 \\ \frac{\partial \mathbf{z}_{d/2:d}}{\partial \mathbf{x}_{1:d/2}} & \text{diag}(s_{\theta}(\mathbf{x}_{1:d/2})) \end{bmatrix}$$

Jacobian



(Lower triangular  
+ structured)

- The Jacobian is triangular, so its determinant is the product of diagonal entries.

$$\det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \prod_{k=1}^d s_{\theta}(\mathbf{x}_{1:d/2})_k$$



# RealNVP

- Takeaway: coupling layers allow unrestricted neural nets to be used in flows, while preserving invertibility and tractability



[Dinh et al. Density estimation using Real NVP. ICLR 2017]

# RealNVP Architecture

Input  $x$ :  $32 \times 32 \times c$  image

- Layer 1: (Checkerboard  $\times 3$ , channel squeeze, channel  $\times 3$ )
  - Split result to get  $x_1$ :  $16 \times 16 \times 2c$  and  $z_1$ :  $16 \times 16 \times 2c$  (fine-grained latents)
- Layer 2: (Checkerboard  $\times 3$ , channel squeeze, channel  $\times 3$ )
  - Split result to get  $x_2$ :  $8 \times 8 \times 4c$  and  $z_2$ :  $8 \times 8 \times 4c$  (coarser latents)
- Layer 3: (Checkerboard  $\times 3$ , channel squeeze, channel  $\times 3$ )
  - Get  $z_3$ :  $4 \times 4 \times 16c$  (latents for highest-level details)

Can be better??



# RealNVP: How to partition variables?

Partitioning can be implemented using a binary mask  $b$ , and using the functional form for  $y$

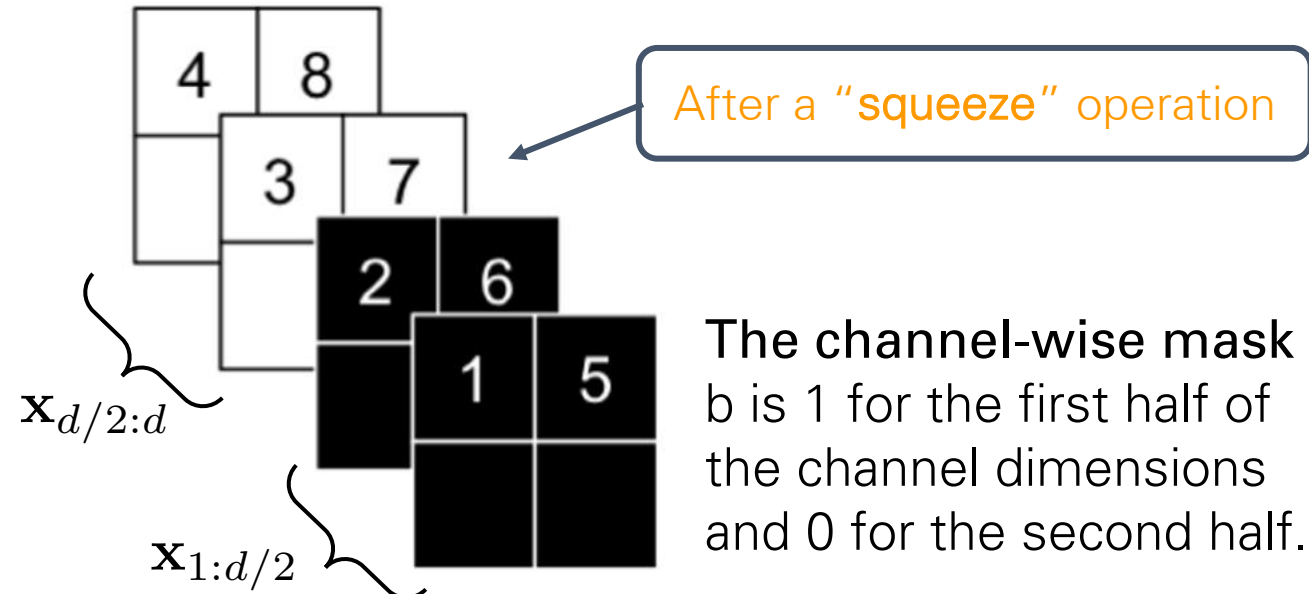
$$f(x) = b \odot x + (1 - b) \odot (x \odot \exp(s_-(b \odot x)) + m(b \odot x))$$

# RealNVP: How to partition variables?

Partitioning can be implemented using a binary mask  $b$ , and using the functional form for  $y$

$$f(x) = b \odot x + (1 - b) \odot (x \odot \exp(s_-(b \odot x)) + m(b \odot x))$$

The **spatial checkerboard pattern mask** has value 1 where the sum of spatial coordinates is odd, and 0 otherwise.

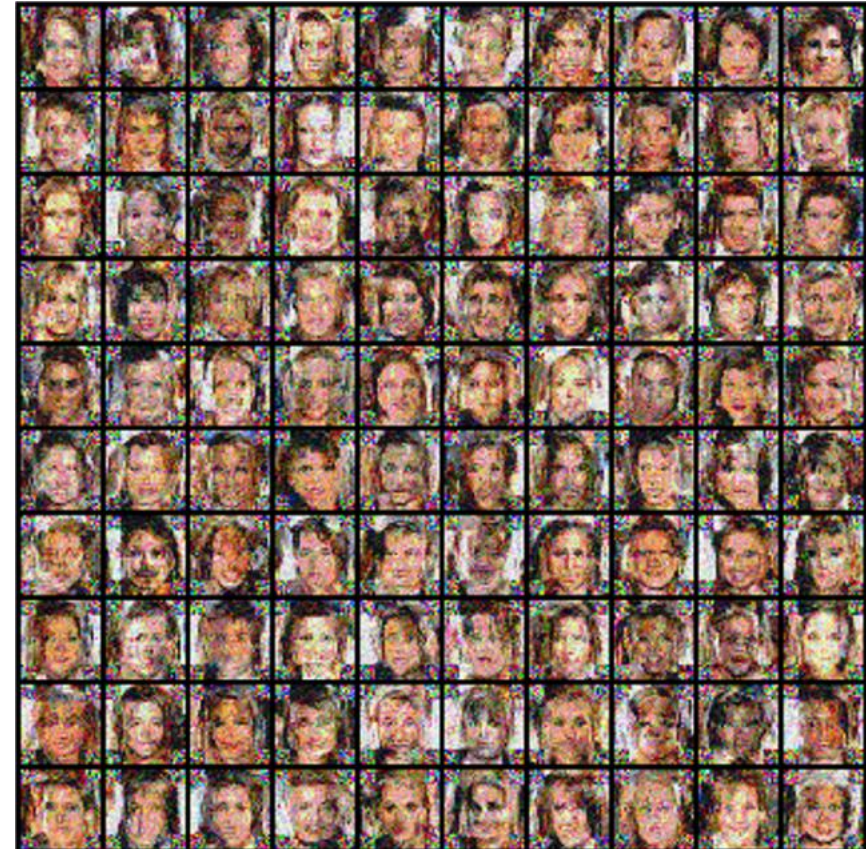


# Good vs Bad Partitioning

Checkerboard  $\times 4$ ; channel squeeze;  
channel  $\times 3$ ; channel unsqueeze;  
checkerboard  $\times 3$



(Mask top half; mask bottom  
half; mask left half; mask right  
half)  $\times 2$



# Lecture overview

- Foundations of Flows (1-D)
- 2-D Flows
- **N-D Flows**
  - Autoregressive Flows and Inverse Autoregressive Flows
  - RealNVP (like) architectures
  - **Glow, Flow++, FFJORD**
- Dequantization

# Choice of coupling transformation

- A Bayes net defines coupling dependency, but what invertible transformation  $f$  to use is a design question

$$\mathbf{x}_i = f_{\theta}(\mathbf{z}_i; \text{parent}(\mathbf{x}_i))$$

- Affine transformation is the most commonly used one (NICE, RealNVP, IAF-VAE, ...)

$$\mathbf{x}_i = \mathbf{z}_i \cdot \mathbf{a}_{\theta}(\text{parent}(\mathbf{x}_i)) + \mathbf{b}_{\theta}(\text{parent}(\mathbf{x}_i))$$

- More complex, nonlinear transformations -> better performance
  - CDFs and inverse CDFs for Mixture of Gaussians or Logistics (Flow++)
  - Piecewise linear/quadratic functions (Neural Importance Sampling)



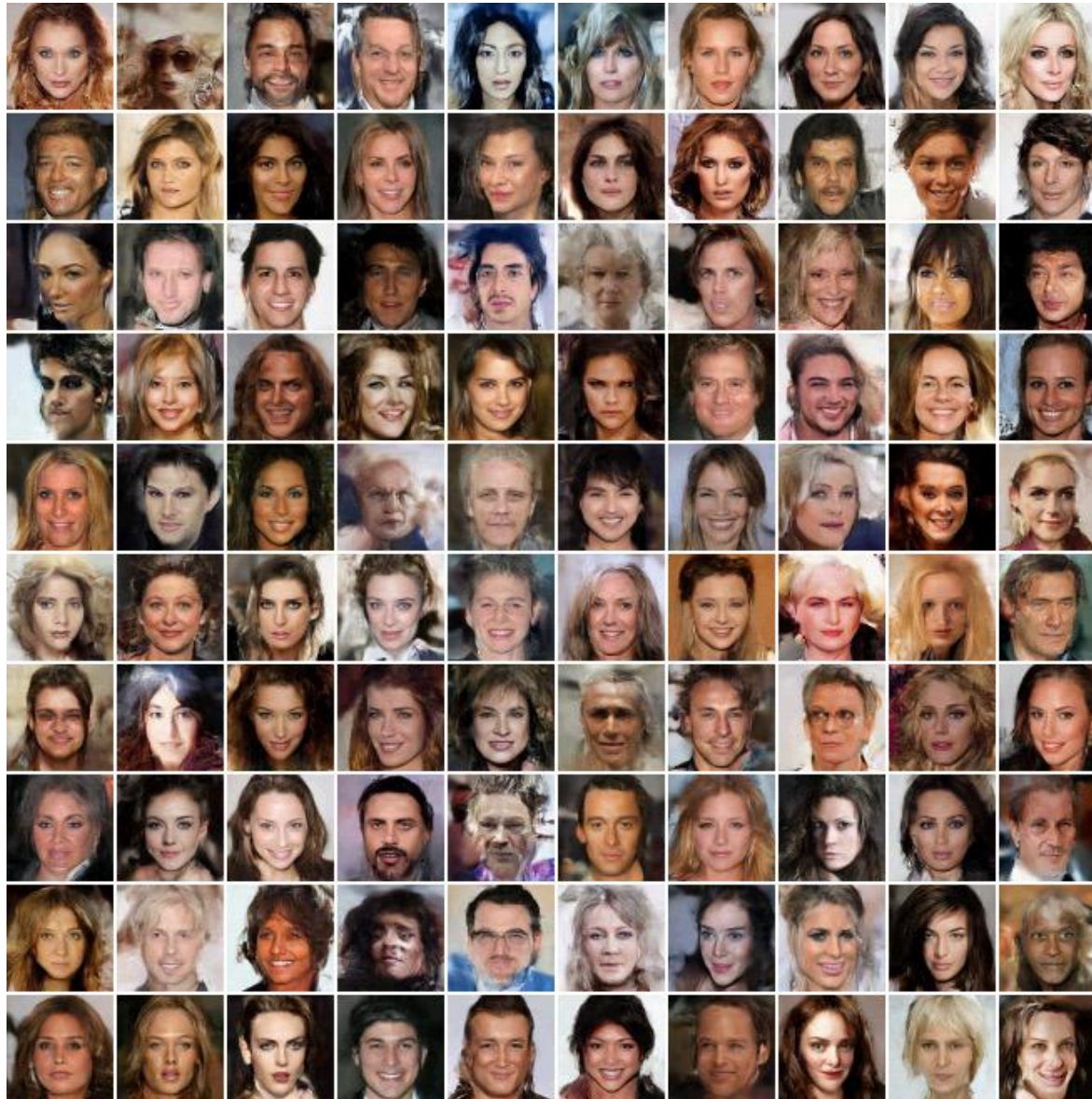
# NN architecture also matters

- Flow++ = MoL transformation + self-attention in NN
  - Bayes net (coupling dependency), transformation function class, NN architecture all play a role in a flow's performance.

*Table 2. CIFAR10 ablation results after 400 epochs of training.  
Models not converged for the purposes of ablation study.*

<b>Ablation</b>	<b>bits/dim</b>	<b>parameters</b>
uniform dequantization	3.292	32.3M
affine coupling	3.200	32.0M
no self-attention	3.193	31.4M
<b>Flow++ (not converged for ablation)</b>	<b>3.165</b>	<b>31.4M</b>

# Flow++



Samples from Flow++  
trained on 64x64 CelebA



# Other classes of flows

- Glow ([link](#))
  - Replacing permutation with  $1 \times 1$  convolution (soft permutation)
  - Large-scale training
- Continuous time flows (FFJORD)
  - Allows for unrestricted architectures. Invertibility and fast log probability computation guaranteed.





# Architectural Taxonomy

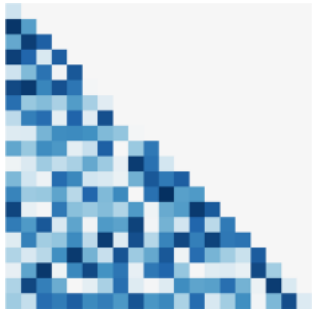
## Sparse connection

$$f(\boldsymbol{x})_t = g(\boldsymbol{x}_{1:t})$$

### 1. Autoregressive

IAF/MAF/NAF  
SOS polynomial  
UMNN

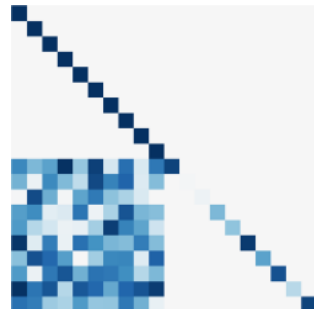
Jacobian



(Lower triangular)

### 2. Block coupling

NICE/RealNVP/Glow  
Cubic Spline Flow  
Neural Spline Flow



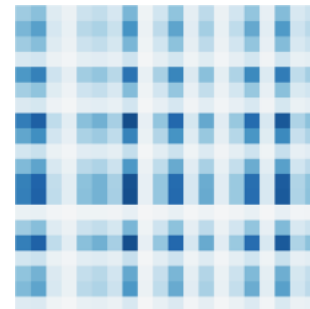
(Lower triangular +  
structured)

## Residual Connection

$$f(\boldsymbol{x}) = \boldsymbol{x} + g(\boldsymbol{x})$$

### 3. Det identity

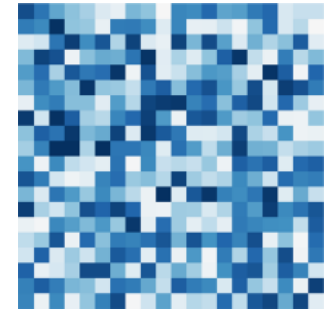
Planar/Sylvester  
flows  
Radial flow



(Low rank)

### 4. Stochastic estimation

Residual  
Flow  
FFJORD

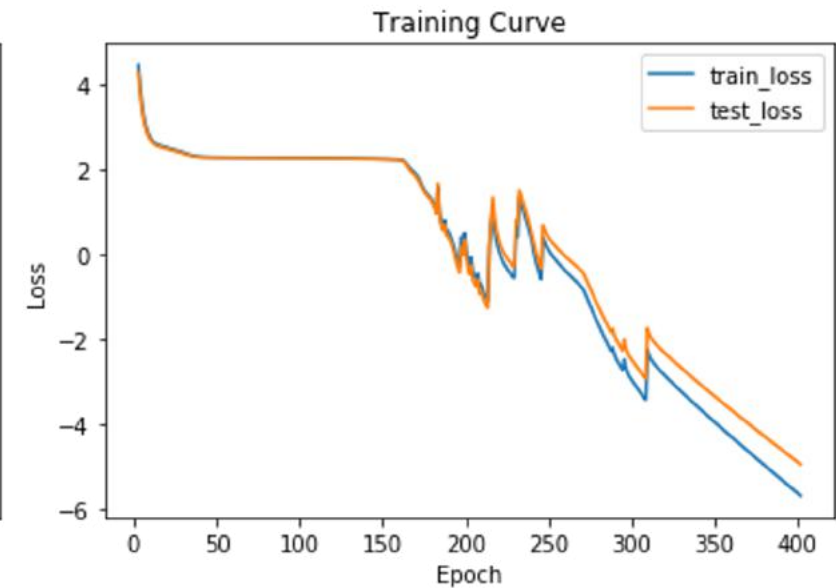
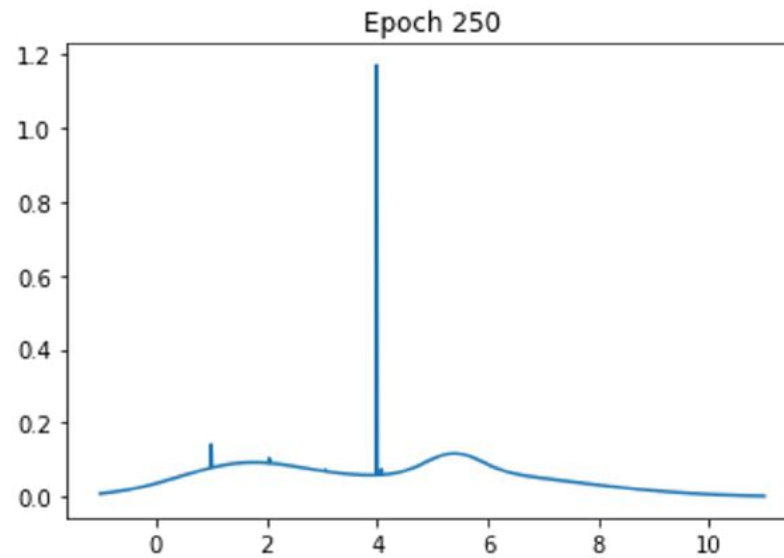
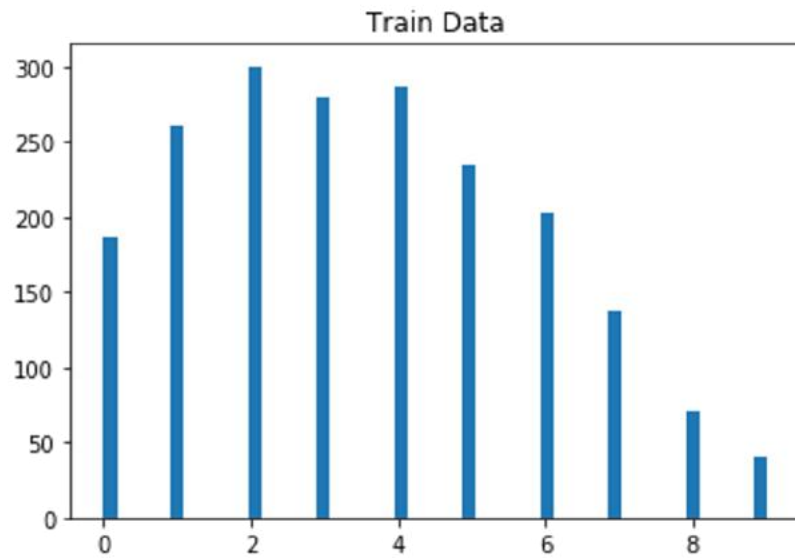


(Arbitrary)

# Lecture overview

- Foundations of Flows (1-D)
- 2-D Flows
- N-D Flows
- Dequantization

# Flow on Discrete Data Without Dequantization...



# Continuous flows for discrete data

- A problem arises when fitting continuous density models to discrete data: degeneracy
  - When the data are 3-bit pixel values,  $\mathbf{x} \in \{0, 1, 2, \dots, 255\}$
  - What density does a model assign to values between bins like 0.4, 0.42...?
- Correct semantics: we want the integral of probability density within a discrete interval to approximate discrete probability mass

$$P_{\text{model}}(\mathbf{x}) := \int_{[0,1)^D} p_{\text{model}}(\mathbf{x} + \mathbf{u}) d\mathbf{u}$$

# Continuous flows for discrete data

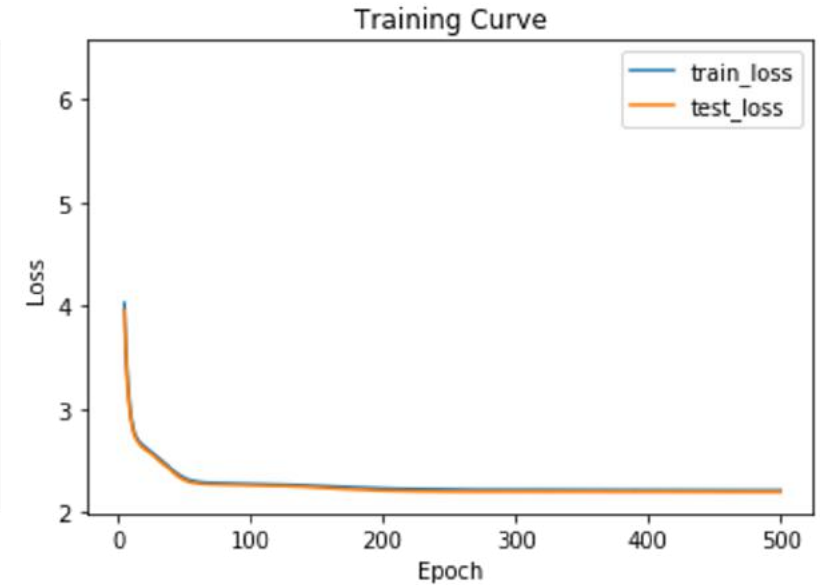
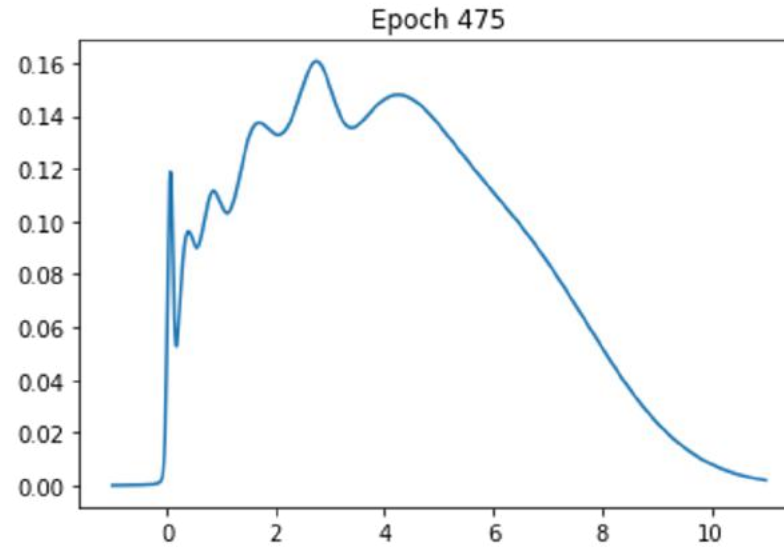
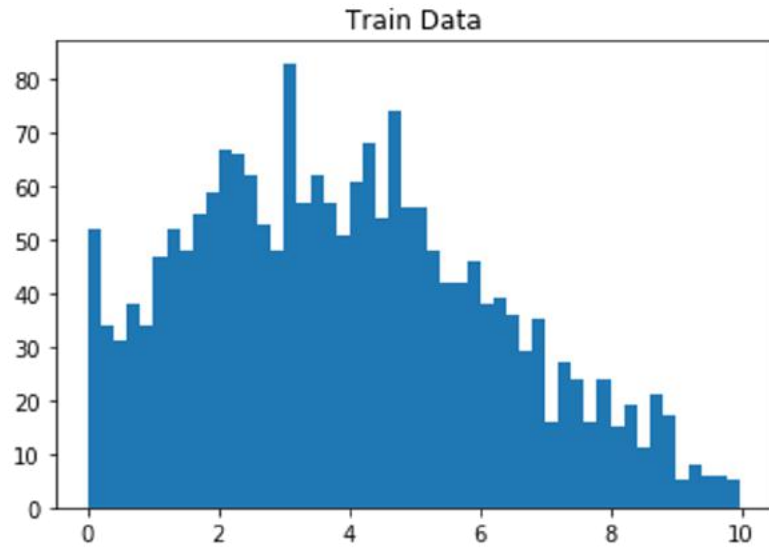
- Solution: **Dequantization**. Add noise to data.

$$\mathbf{x} \in \{0, 1, 2, \dots, 255\}$$

- We draw noise  $\mathbf{u}$  uniformly from  $[0, 1)^D$

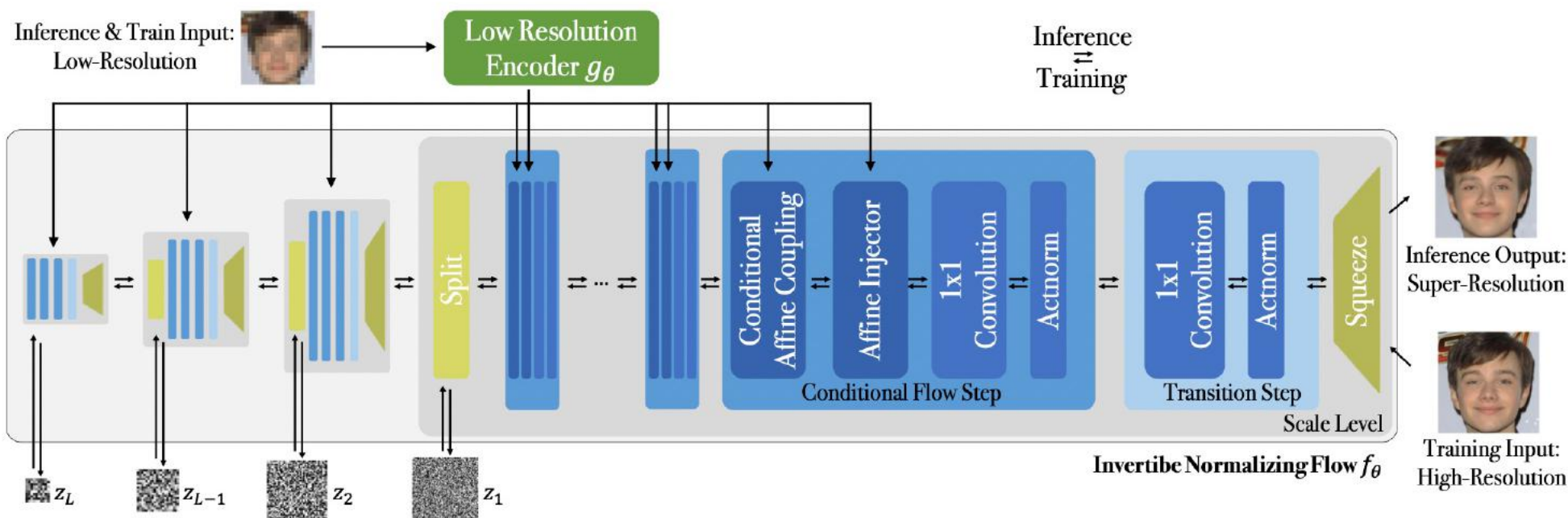
$$\begin{aligned}\mathbb{E}_{\mathbf{y} \sim p_{\text{data}}} [\log p_{\text{model}}(\mathbf{y})] &= \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \int_{[0,1)^D} \log p_{\text{model}}(\mathbf{x} + \mathbf{u}) d\mathbf{u} \\ &\leq \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \int_{[0,1)^D} p_{\text{model}}(\mathbf{x} + \mathbf{u}) d\mathbf{u} \\ &= \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [\log P_{\text{model}}(\mathbf{x})]\end{aligned}$$

# Flow on Discrete Data With Dequantization



# Applications: Super-Resolution

- SRFlow
  - A normalizing flow based super-resolution method, allowing diversity
  - Outperforms state-of-the-art GAN-based approaches



# Application: Text Synthesis

- Language Flow
  - non-autoregressive and autoregressive flow-based models

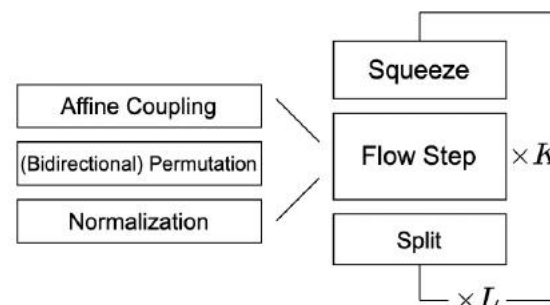


Figure 2: Non-autoregressive Language Flow model with Multi-Scale architecture.

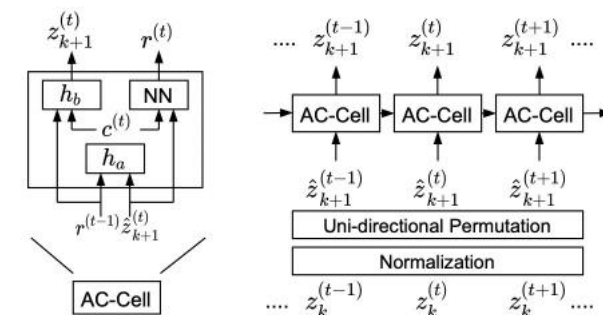


Figure 3: Autoregressive Language Generative Flow model. The whole autoregressive flow model contains multiple K steps. This figure illustrates one flow step from  $z_k$  to  $z_{k+1}$ .

Non-Autoregressive Samples	Autoregressive Samples
what does house way when when that little he when the even? what did richard know when he she else there the what does nelson going when he she when he what that to? what did richard know when he she else there the what does nelson going when he she when he what that to?	what does wilson probably do after drawing? what did jamie want after charlie forget her immediately what is brian aware what did caleb say after he went out? what does phoebe think?

Table 1: Data samples generated by our flow models. We sample from a Gaussian distribution and generate questions by our non-autoregressive or autoregressive flow decoders. Models are trained on TVQA questions.



# Applications: Audio Synthesis

- FloWaveNet
  - A flow-based generative model for raw audio synthesis
  - Efficiently samples raw audio in real-time

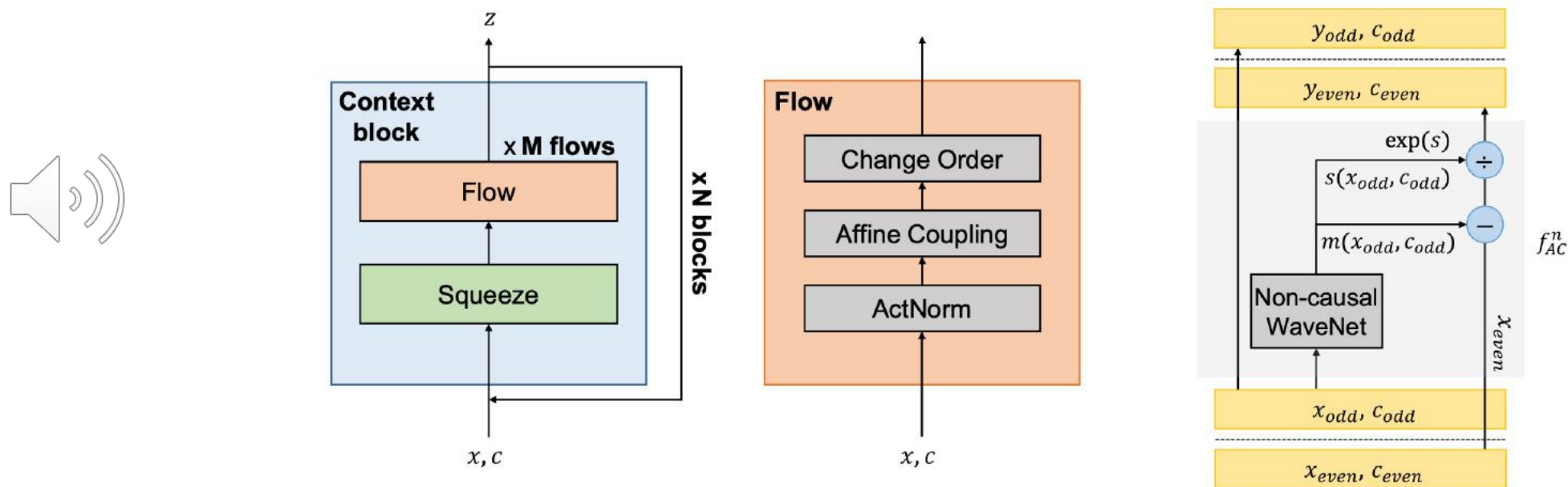
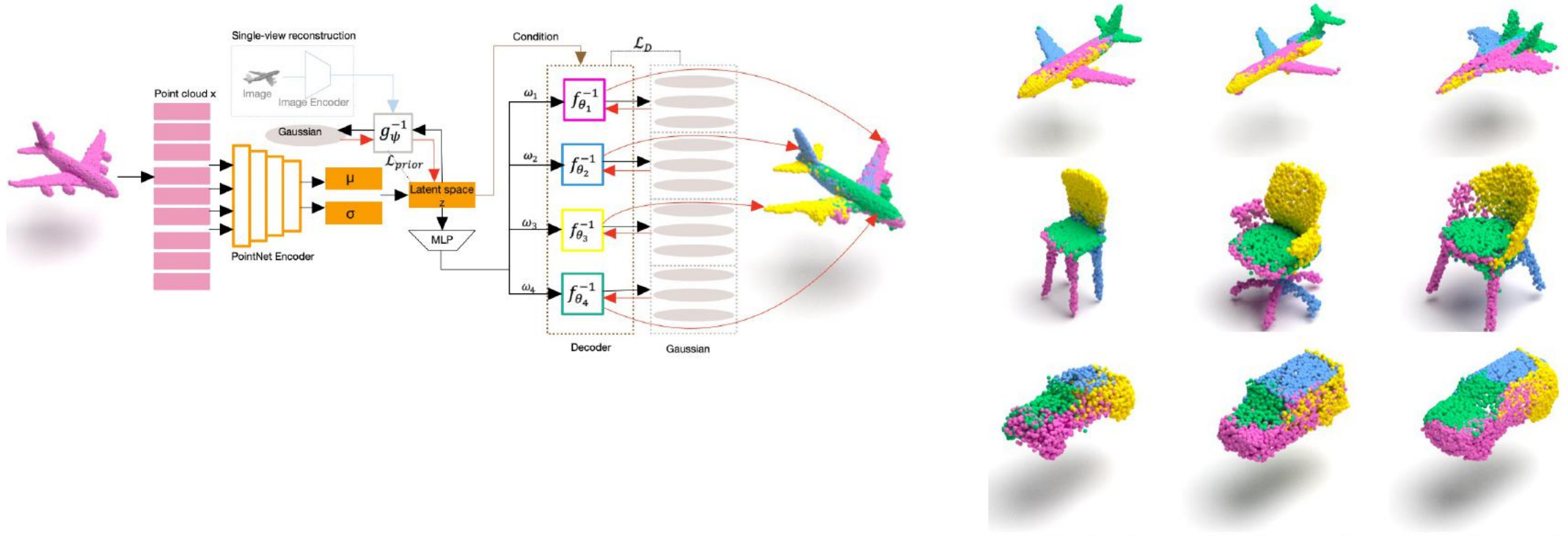


Figure 1. Schematic diagram of FloWaveNet. Left: an entire forward pass of the FloWaveNet consisting of N context blocks. Middle: an abstract diagram of the flow operation. Right: a detailed version of the affine coupling operation.

# Applications: Point Cloud Generation

- Mixture of Normalizing Flows for modeling 3D point clouds
- Each mixture component learns to specialize in a distinct subregion in an unsupervised fashion.



# Future directions

- The ultimate goal: a likelihood-based model with
  - fast sampling
  - fast inference
  - fast training
  - good samples
  - good compression
- Flows seem to let us achieve some of these criteria.
- But how exactly do we design and compose flows for great performance? That's an open question.

# **Next lecture: Variational Autoencoders**