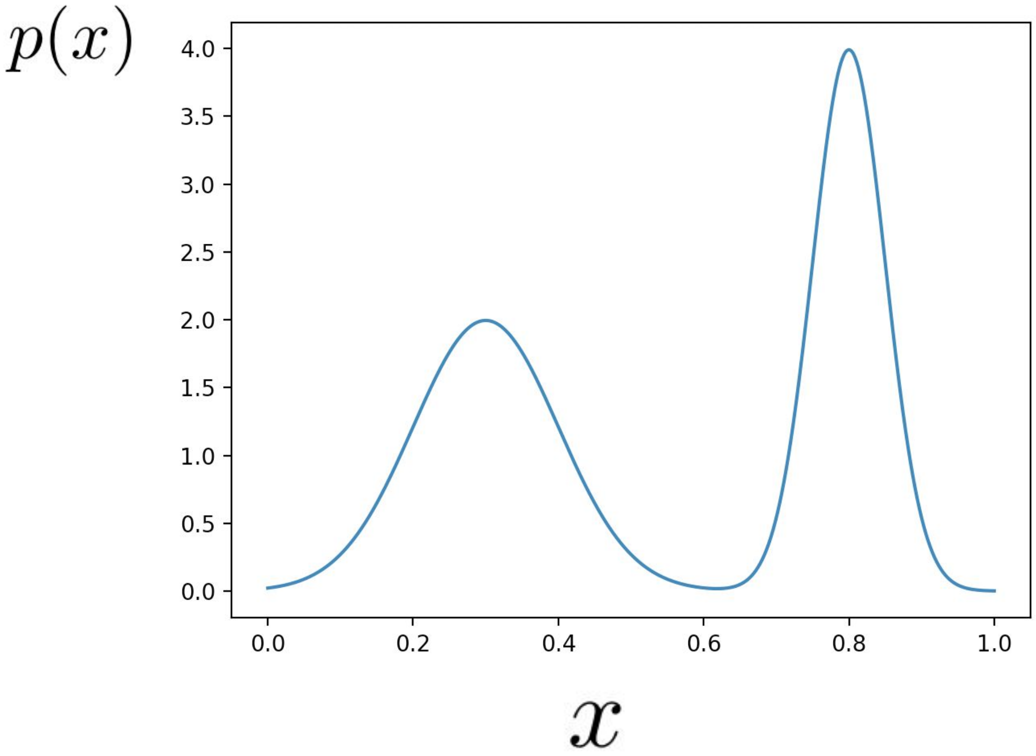
Normalising Flows

University of Victoria PHYS-555

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:

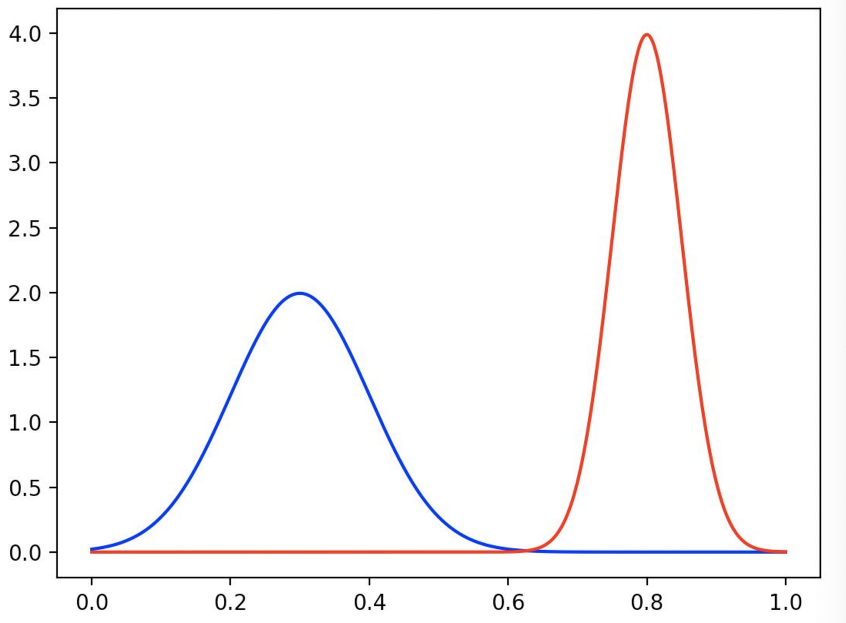
$$\max_\theta \sum_i \log p_\theta(x^{(i)})$$

Equivalently:

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

Example: Mixtures of Gaussians

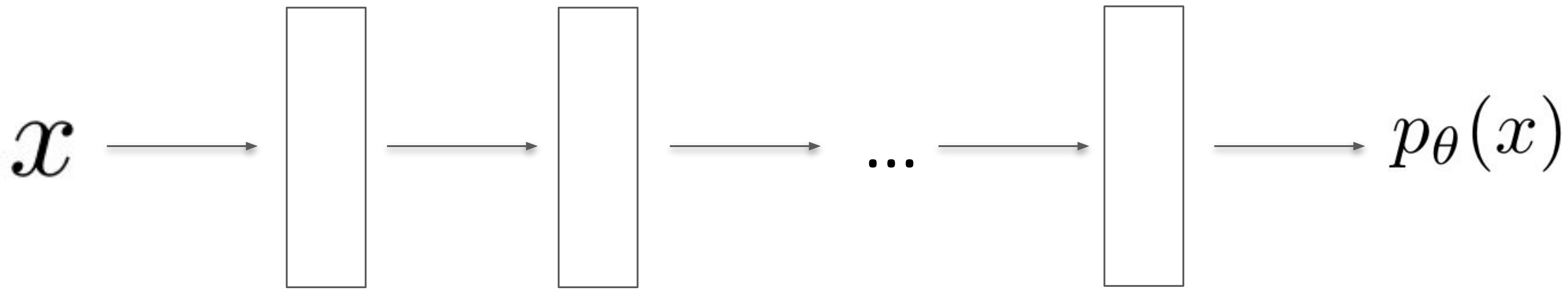
$$p_\theta(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x; \mu_i, \sigma_i^2)$$



Parameters: means and variances of components, mixture weights

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

How to fit a general density model?

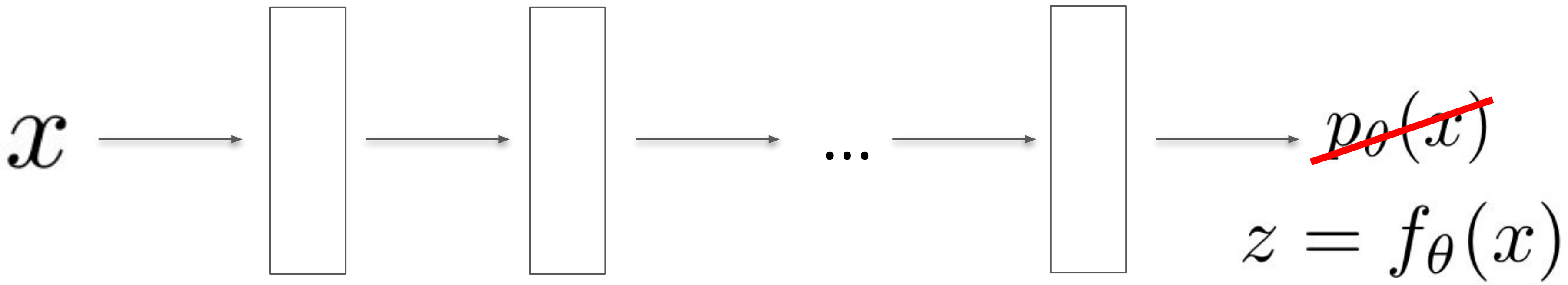


- How to ensure proper distribution, i.e:

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

- How to sample?
- Latent representation?

Flows: Main Idea

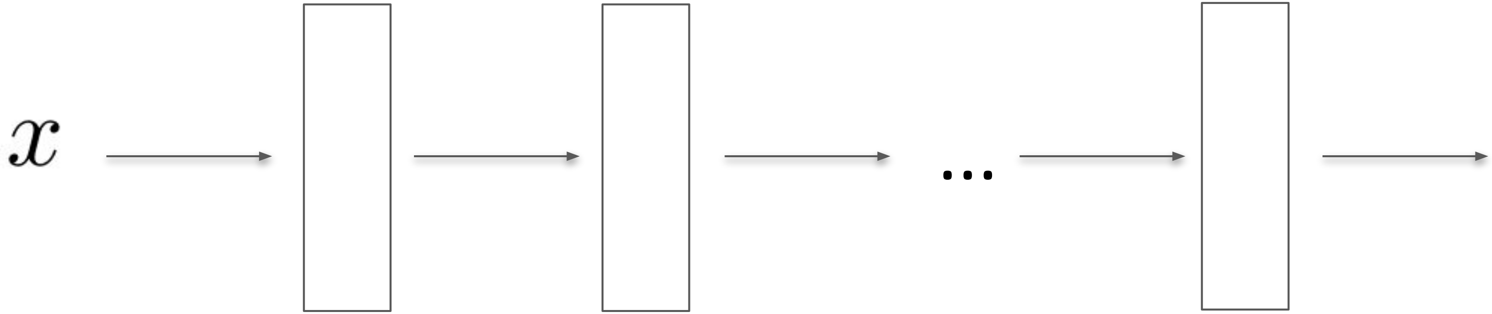


Generally: $z \sim p_Z(z)$

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

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

Flows: Training



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

$$z \sim p_Z(z)$$

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

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_θ invertible & differentiable

Flows: Training

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

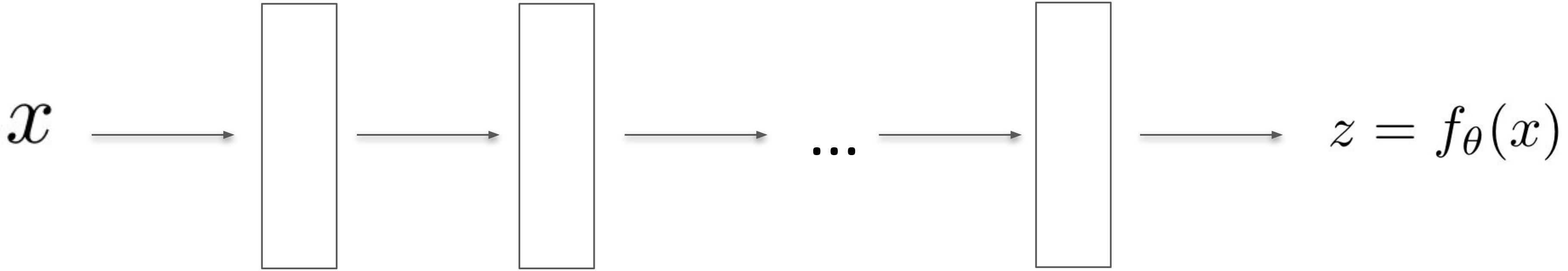
$$\max_\theta \sum_i \log p_\theta(x^{(i)})$$

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

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

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

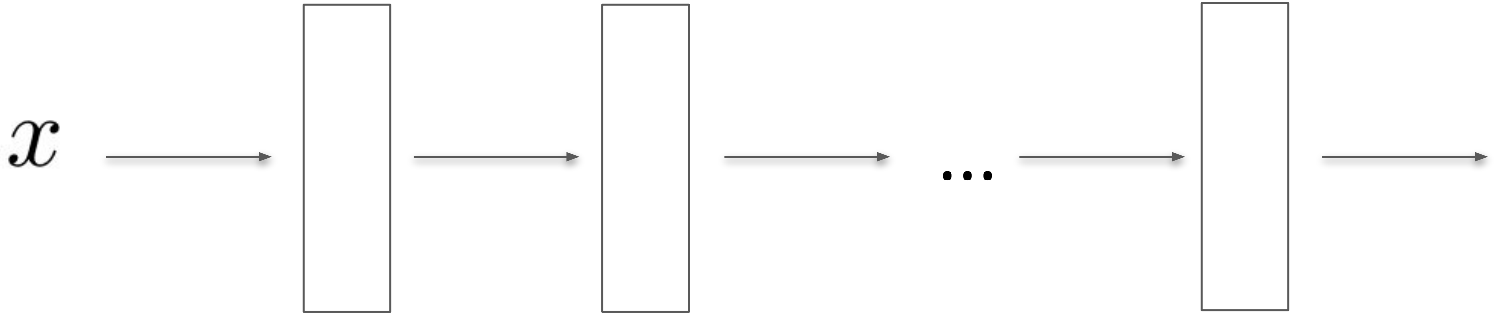
Flows: Sampling



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?



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

$$z \sim p_Z(z)$$

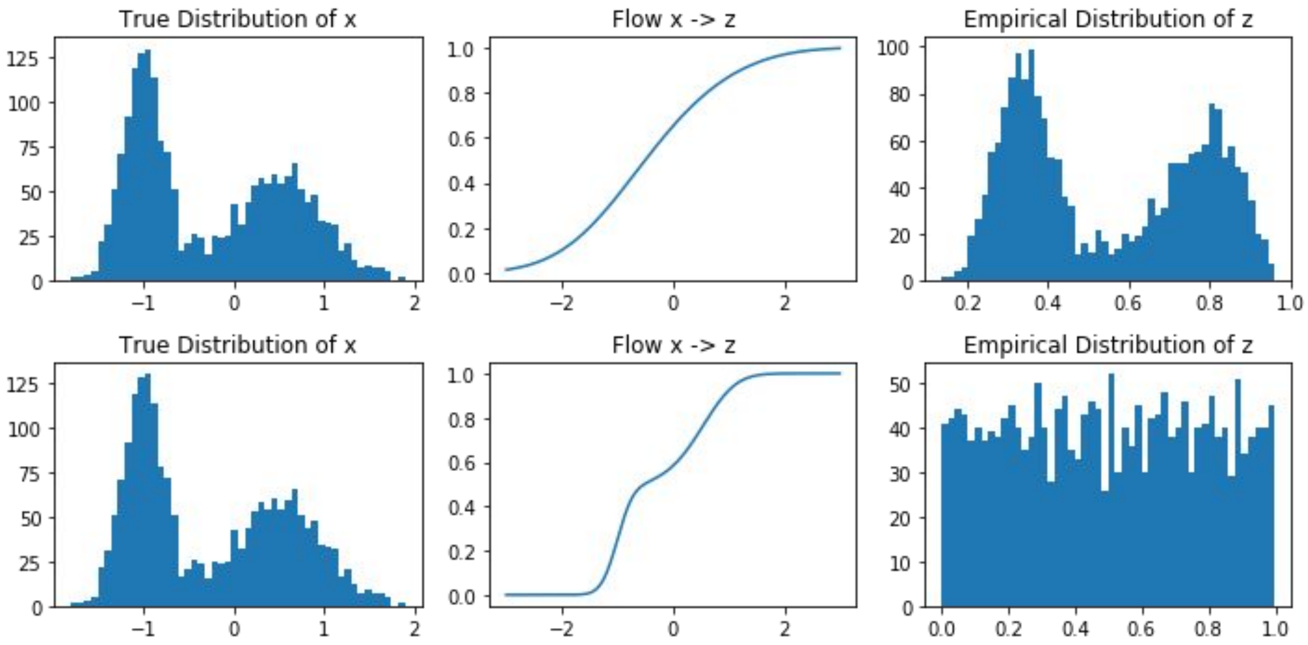
Recall, change of variable formula requires

- f_θ Invertible & differentiable

Example: Flow to Uniform z

Before training

After training

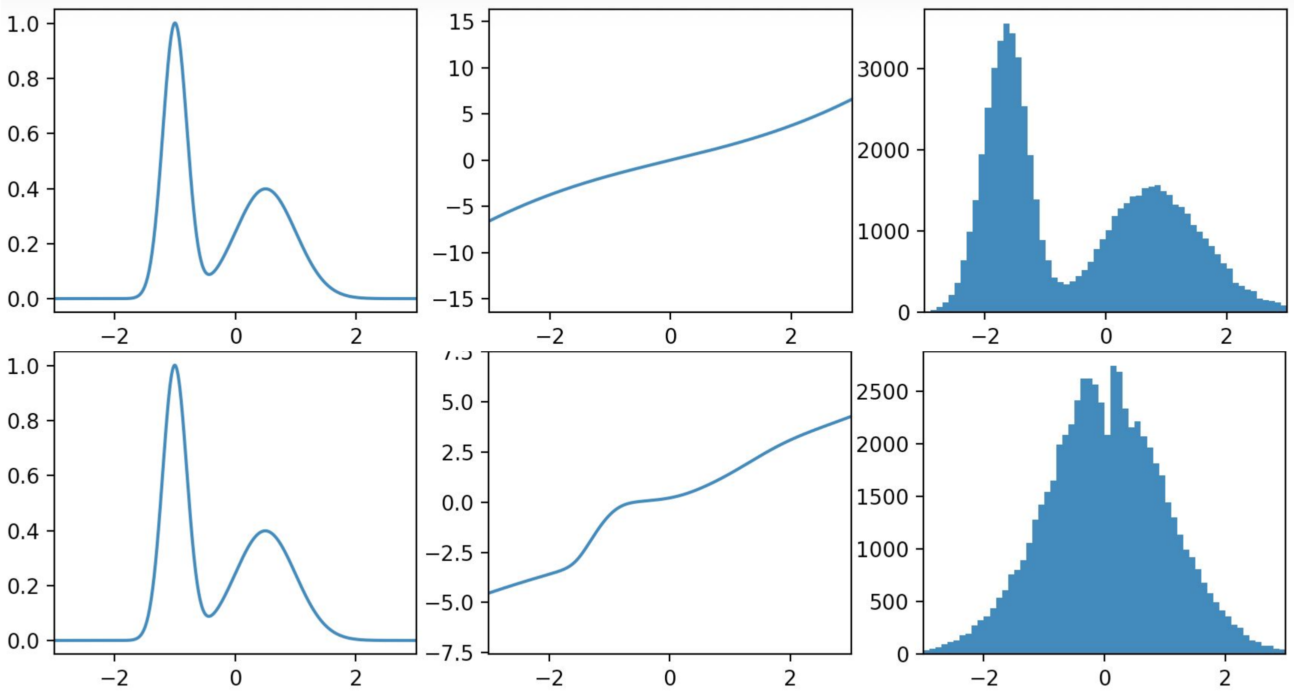


True distribution of x　　Flow x → z　　Empirical distribution of z

Example: Flow to Gaussian z

Before training

After training



True distribution of x

Flow x ⟶ z

Empirical distribution of z

Practical Parameterizations of Flows

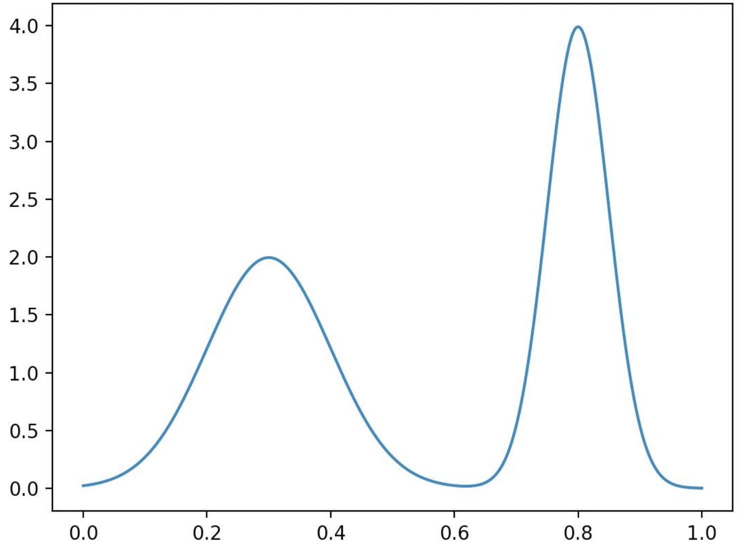
Requirement: Invertible and Differentiable

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

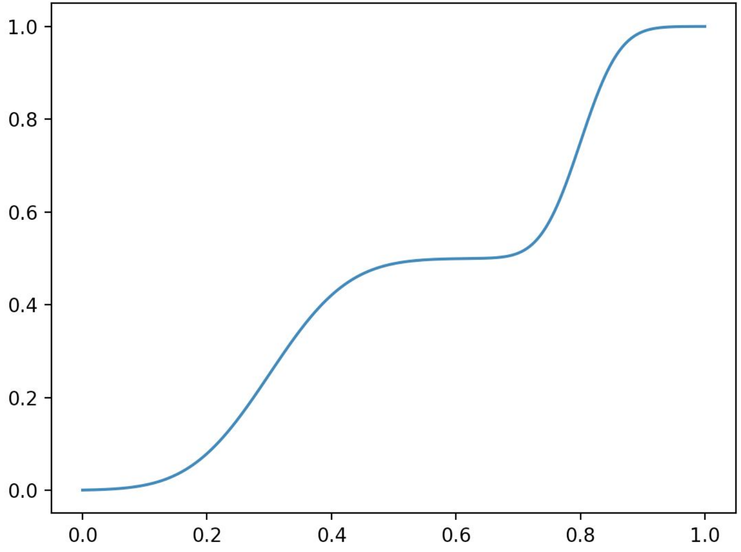
How general are flows?

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

Cumulative Density Function (CDF)



$$p_\theta(x)$$



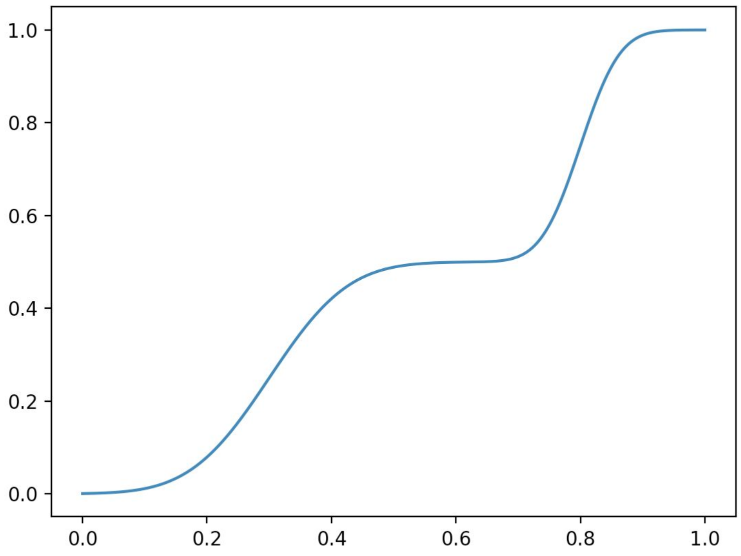
$$f_\theta(x) = \int_{-\infty}^{x} p_\theta(t)\, dt$$

Sampling via inverse CDF

Sampling from the model:

$$z \sim \text{Uniform}([0, 1])$$
$$x = f_\theta^{-1}(z)$$

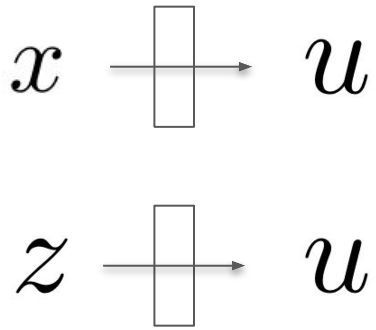
The CDF is an invertible, differentiable map from data to [0, 1]



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

How general are flows?

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



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

2-D Autoregressive Flow

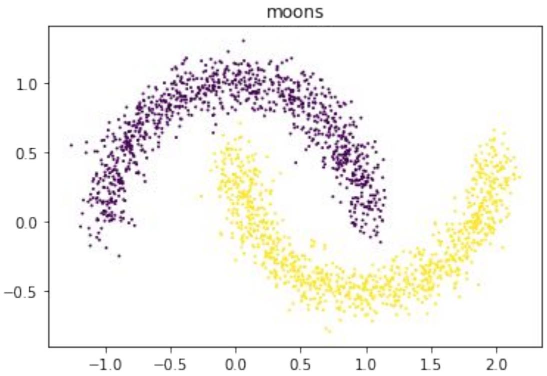
$$x_1 \to z_1 = f_\theta(x_1)$$

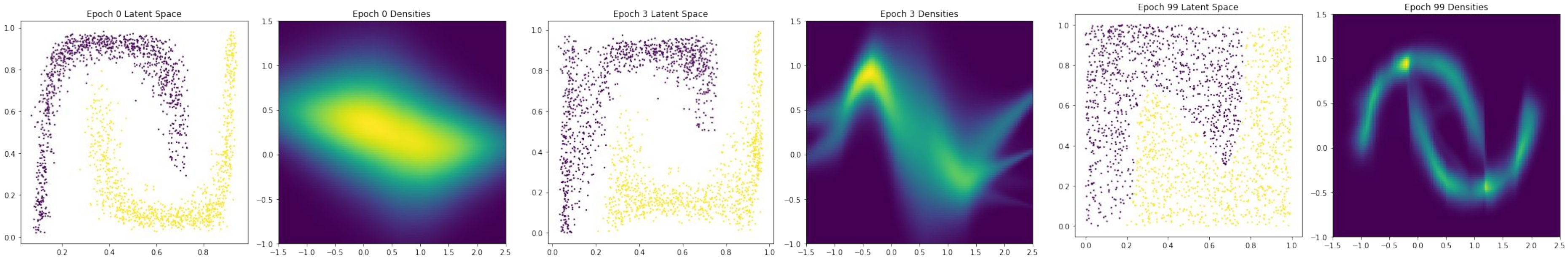
$$x_2 \to z_2 = f_\theta(x_1, x_2)$$

2-D Autoregressive Flow: Two Moons

Architecture:

- Base distribution: Uniform[0,1]^2
- x1: mixture of 5 Gaussians
- x2: mixture of 5 Gaussians, conditioned on x1

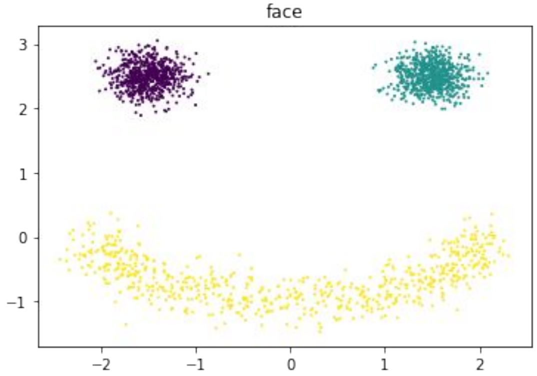


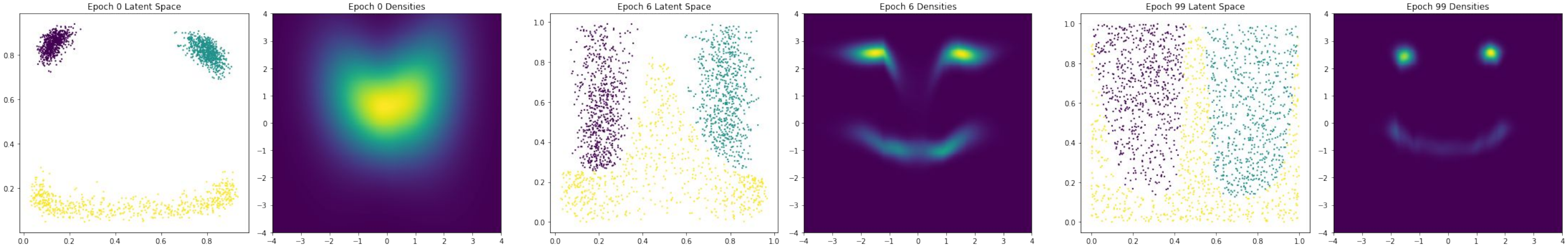


2-D Autoregressive Flow: Face

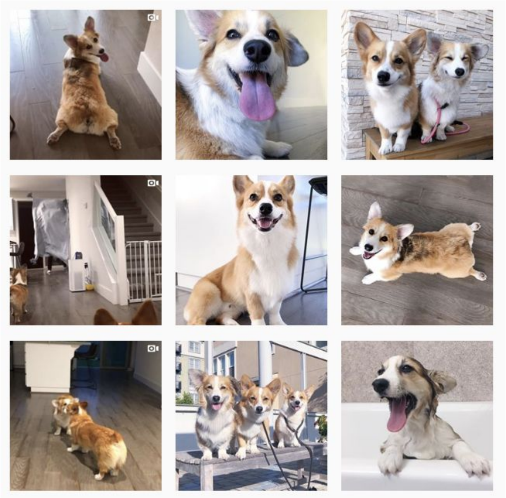
Architecture:

- Base distribution: Uniform[0,1]^2
- x1: mixture of 5 Gaussians
- x2: mixture of 5 Gaussians, conditioned on x1



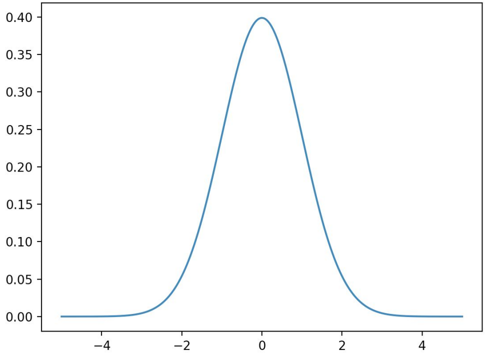


High-dimensional data



f (inference)

f^{-1} (sampling)



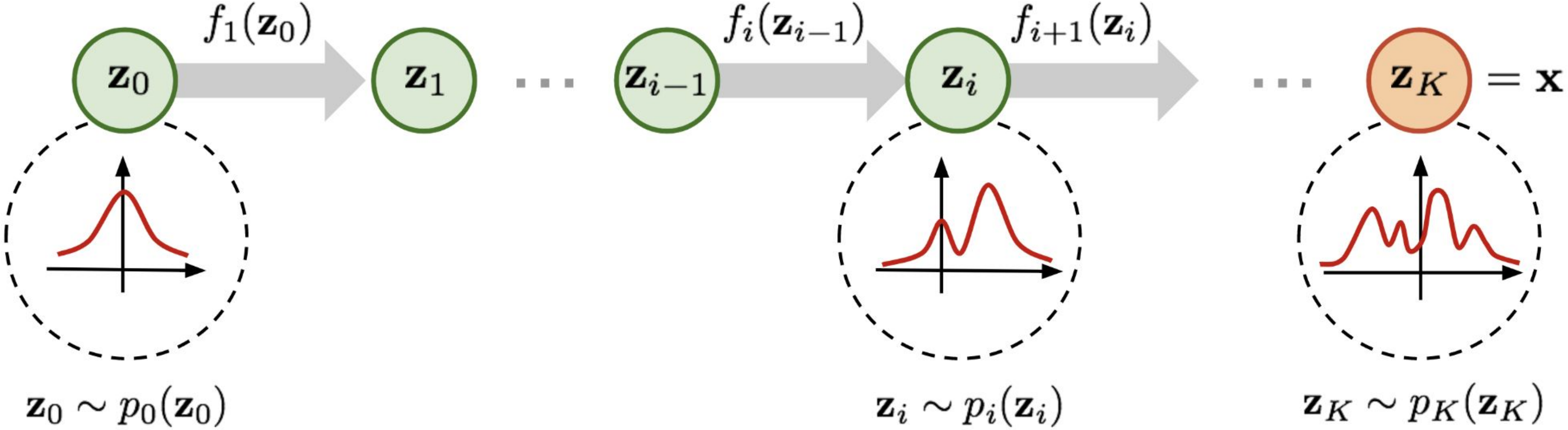
x and z must have the same dimension

Flow-models desiderata

1. Easy to compute prior p(**z**) (i.e. isotropic Gaussian) for fast sampling

2. Invertible transformation:
    a. needs fast evaluation from **x** → **z**
    b. needs fast sampling from **z** →**x**

3. Needs determinant of the NxN Jacobian matrix ~ $O(N^3)$
    → architecture doing a special structure of the Jacobian matrix



$f_1(\mathbf{z}_0)$
$f_i(\mathbf{z}_{i-1})$
$f_{i+1}(\mathbf{z}_i)$
$\mathbf{z}_0$
$\mathbf{z}_1$
$\mathbf{z}_{i-1}$
$\mathbf{z}_i$
$\mathbf{z}_K = \mathbf{x}$
$\mathbf{z}_0 \sim p_0(\mathbf{z}_0)$
$\mathbf{z}_i \sim p_i(\mathbf{z}_i)$
$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$

Change of MANY variables

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

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

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 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:

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

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

Constructing flows: composition

- **Flows can be composed**

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

- Easy way to increase expressiveness

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

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

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

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 $z \sim N(0, I)$
- Log likelihood is expensive when dimension is large.
    - The Jacobian of f is A^{-1}
    - Log likelihood involves calculating det(A)

Elementwise flows

$$f_\theta((x_1, \ldots, x_d)) = (f_\theta(x_1), \ldots, f_\theta(x_d))$$

- 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.

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

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

NICE / RealNVP

Affine coupling layer

- Split variables in half: $x_{1:d/2}$, $x_{d/2+1:d}$

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

- Invertible! Note that s_θ and t_θ 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

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

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

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

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

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

RealNVP

Coupling layers allow unrestricted neural nets to be used in flows, while preserving invertibility and tractability

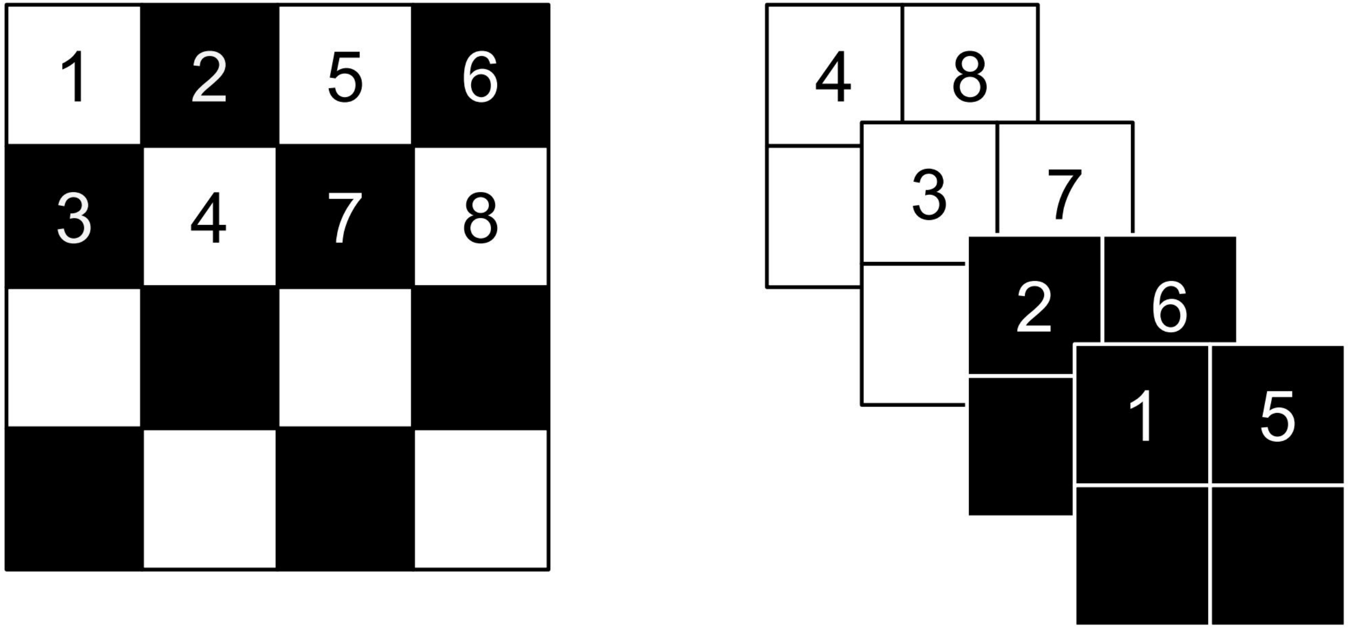
[Dinh et al. Density estimation using Real NVP. ICLR 2017]

RealNVP Architecture

Input x: 32x32xc image

- Layer 1: (Checkerboard x3, channel squeeze, channel x3)
  - Split result to get x1: 16x16x2c and z1: 16x16x2c (fine-grained latents)
- Layer 2: (Checkerboard x3, channel squeeze, channel x3)
  - Split result to get x2: 8x8x4c and z2: 8x8x4c (coarser latents)
- Layer 3: (Checkerboard x3, channel squeeze, channel x3)
  - Get z3: 4x4x16c (latents for highest-level details)

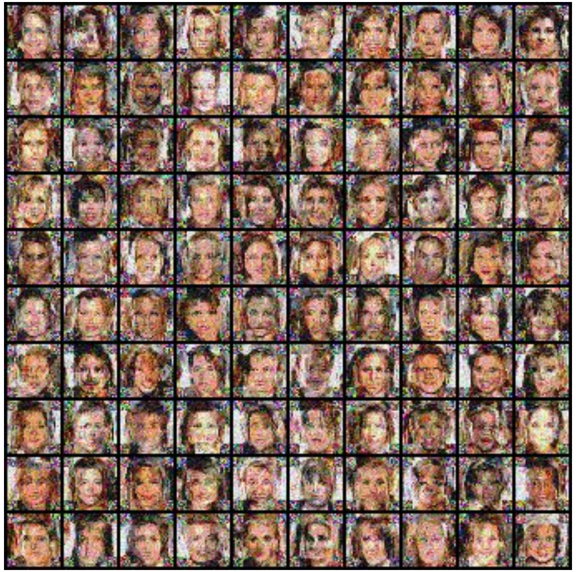
RealNVP: Partitioning variables



Good vs Bad Partitioning

Checkerboard x4; channel squeeze; channel x3; channel unsqueeze; checkerboard x3

(Mask top half; mask bottom half; mask left half; mask right half) x2



Choice of coupling transformation

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

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

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

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

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

Other classes of flows

- Glow (link)
    - Invertible 1x1 convolutions
    - Large-scale training

- Continuous time flows (FFJORD)
    - Allows for unrestricted architectures. Invertibility and fast log probability computation guaranteed.