Lecture 1. Normalizing flows

Introduction to Bayesian Statistical Learning II

03.06.2024 Instructors: Alina Bazarova, Oleg Filatov

Brief recall on the Bayesian concepts

$$posterior = \frac{prior \times likelihood}{evidence}$$

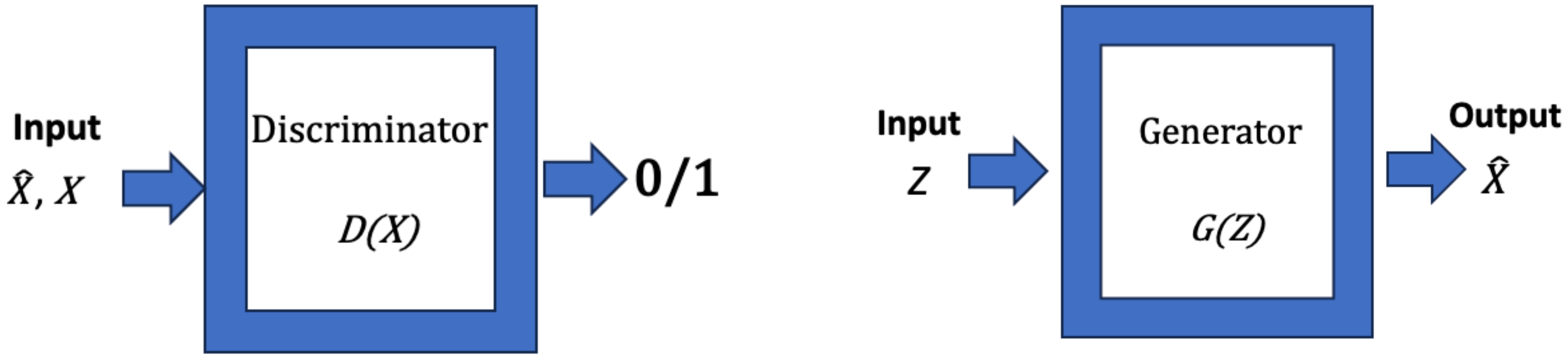
$$P(A|X) = \frac{P(A)P(X|A)}{P(X)}$$

Where A are the parameters and X is the data (discrete case)

$$p(\theta|x) = \frac{p(x)p(x|\theta)}{\int p(x)p(x|\theta)dx}$$

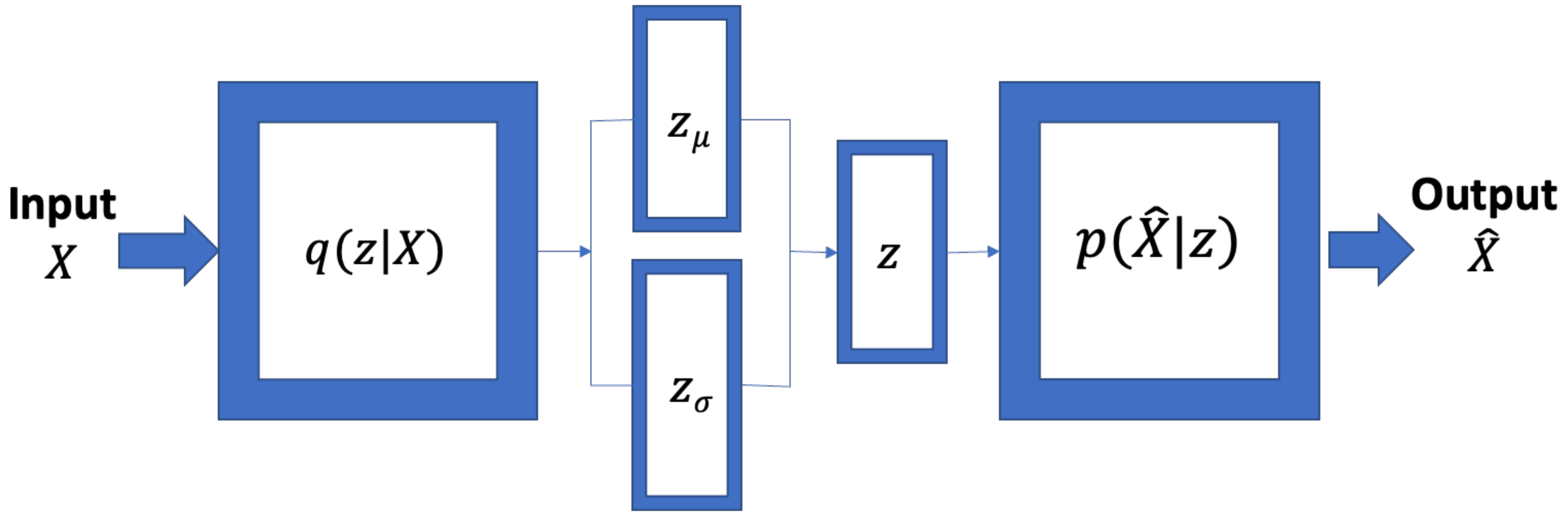
Where θ are the parameters, and $p(x)$ is a is a probability density function (continuous case)

Generative models



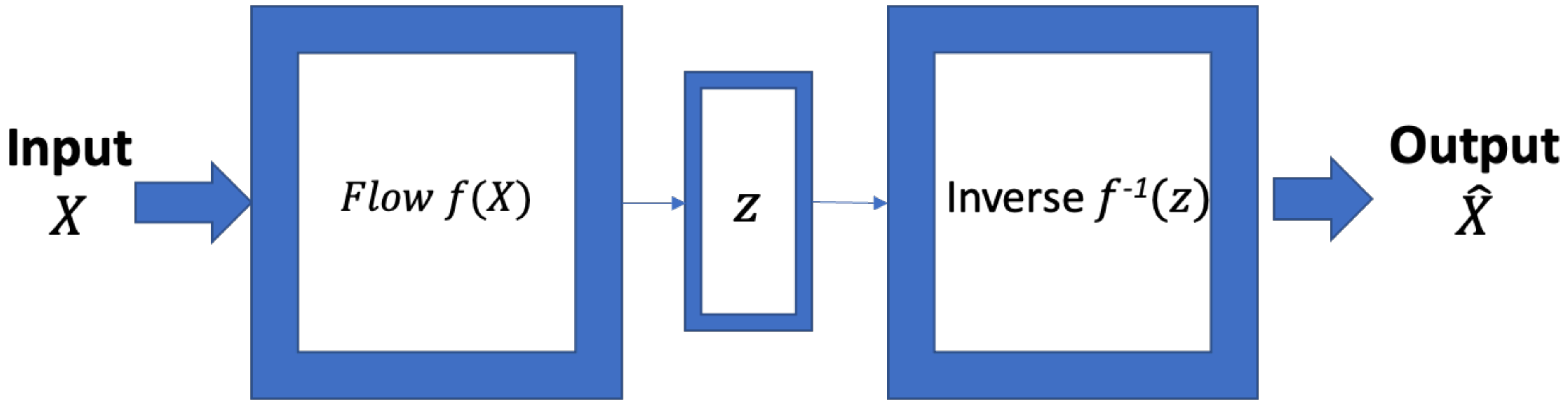
GAN: generator and discriminator trained together

No likelihood estimate



VAE: implicitly learns the distribution of the data

Latent space has a **lower** than input dimension



Normalizing flows: learns exact likelihood estimate, uses

A chain of invertible functions. Latent space has the **same**

dimension as input

What do normalizing flows have to do with Bayesian inference?

- Normalizing flows are capable of learning exact likelihood estimate, and therefore can be a powerful tool in approximate Bayesian methods such as Simulation Based Inference, especially in cases when likelihood is intractable

What do normalizing flows have to do with Bayesian inference?

- Normalizing flows are capable of learning **exact likelihood** estimate, and therefore can be a powerful tool in **approximate Bayesian** methods such as Simulation Based Inference, especially in cases when likelihood is **intractable**

- Normalizing flows represent a series of transformations of an initial simple distribution - can be viewed as our **prior beliefs** on the posterior distribution

More concrete…

Main idea: We wish to map **simple distributions** with easy to sample and evaluate densities to **complex ones** (which are learned via data)

Change of variables

Let Z and X be random variables, such that $X = f(Z),\ Z = f^{-1}(X)$, where $f : \mathbb{R}^n \rightarrow \mathbb{R}$

then $$p_X(x) = p_Z(f^{-1}(x))\,|\,det(\frac{\partial f^{-1}(x)}{\partial x})\,|$$ holds.

More concrete…

Main idea: We wish to map **simple distributions** with easy to sample and evaluate densities to **complex ones** (which are learned via data)

Change of variables

Let Z and X be random variables, such that $X = f(Z),\ Z = f^{-1}(X)$, where $f : \mathbb{R}^n \to \mathbb{R}$

then $$p_X(x) = p_Z(f^{-1}(x))\,|\,det(\frac{\partial f^{-1}(x)}{\partial x})\,|$$ holds.

- x and z are continuous and of the same dimension
- $\dfrac{\partial f^{-1}(x)}{\partial x}$ is a Jacobian n x n matrix, where each (i, j) entry is $\dfrac{\partial f^{-1}(x)_i}{\partial x_j}$

Normalizing flow models

Latent variables Z and observed variables X, $f_\theta : \mathbb{R}^n \to \mathbb{R}^n$ is a mapping between X and Z, which is deterministic and invertible.

Normalizing flow models

Latent variables Z and observed variables X, $f_\theta : \mathbb{R}^n \to \mathbb{R}^n$ is a mapping between X and Z, which is deterministic and invertible.

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

$$p_X(x;\theta) = p_Z(f_\theta^{-1}(x))\,|\,det(\frac{\partial f_\theta^{-1}(x)}{\partial x})\,|$$

Key requirements:

1. f_θ is invertible
2. x and z have the same dimension
3. Jacobian computation has to be efficient

Normalizing flow models. Examples

Planar flow

$x = f_\theta(z) = z + uh(w^T z + b)$, where u, w, b are trainable parameters

$|det(\frac{\partial f_\theta(z)}{\partial z})| = |1 + h'(w^T z + b)u^T w|$ **NB:** $h'(w^T z + b)u^T w \geq -1$, h is invertible

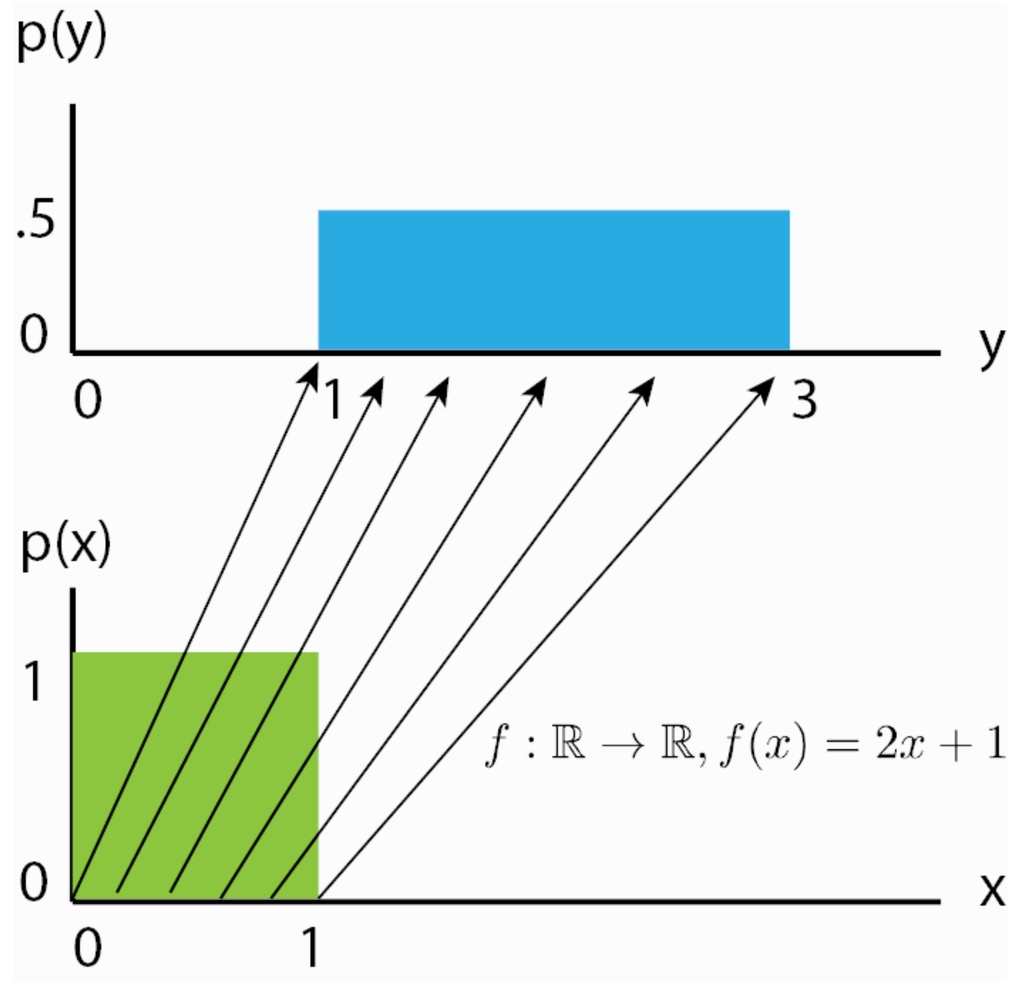


Figure credit Eric Jang https://blog.evjang.com/2018/01/nf1.html

<— Illustration: affine shift

Transforming $U[0,1]$ distribution using $f(x) = 2x + 1$

NB: we are dealing with probability density functions,

hence the transformed volume has to **integrate to 1**!

Normalizing flow models. Examples

Nonlinear Independent Components Estimation (NICE)

Partitions z into two **disjoint subsets** z_1 and z_2

Forward mapping: $x_1 = z_1, x_2 = z_2 + m_\theta(z_1)$, where the first one is an **identity mapping**,

and m_θ is a **neural network**

Reverse mapping: $z_1 = x_1,\ z_2 = x_2 - m_\theta(x_1)$

The **Jacobian** of the forward mapping is **lower-triangular**, determinant is equal to 1 (volume preserving transform).

Normalizing flow models. Examples

Nonlinear Independent Components Estimation (NICE)

Partitions z into two **disjoint subsets** z_1 and z_2

Forward mapping: $x_1 = z_1, x_2 = z_2 + m_\theta(z_1)$, where the first one is an **identity mapping**,

and m_θ is a **neural network**

Reverse mapping: $z_1 = x_1,\ z_2 = x_2 - m_\theta(x_1)$

The **Jacobian** of the forward mapping is **lower-triangular**, determinant is equal to 1 (volume preserving transform).

Real Non-Volume Preserving (RealNVP)

$x_2 = \exp(s_\theta(z_1)) \odot z_2 + m_\theta(z_1)$ Will look closer in the jupyter notebook!

Jacobian is a product of the **scaling factors**!

Normalizing flow models. Examples

Masked Autoregressive Flow (MAF)

$$p(x) = \prod_i p(x_i | x_{1:i-1})$$

Target density is modelled as a product of one-dimensional densities, depending only on the previous values

$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | \mu_i, (\exp s_i)^2),\ \mu_i = \mu_i(x_{1:i-1}),\ s_i = s_i(x_{1:i-1})$, where $\mu_i()$ and $s_i()$ are neural networks

$x_i = z_i \exp s_i + \mu_i,\ z_i \sim \mathcal{N}(0,1)$ **Inverse:** $z_i = (x_i - \mu_i)/\exp s_i$ **—>** no need to compute μ_i^{-1} and s_i^{-1}

Normalizing flow models. Examples

Masked Autoregressive Flow (MAF)

$$p(x) = \prod_i p(x_i | x_{1:i-1})$$

Target density is modelled as a product of one-dimensional densities, depending only on the previous values

$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | \mu_i, (\exp s_i)^2),\ \mu_i = \mu_i(x_{1:i-1}),\ s_i = s_i(x_{1:i-1})$, where $\mu_i()$ and $s_i()$ are neural networks

$x_i = z_i \exp s_i + \mu_i,\ z_i \sim \mathcal{N}(0,1)$ **Inverse:** $z_i = (x_i - \mu_i)/\exp s_i$ **—>** no need to compute μ_i^{-1} and s_i^{-1}

Masked Autoencoder for Distribution Estimation (MADE): allows to speed up MAF!

All conditional likelihoods $p(x_1), p(x_2 | x_1), \ldots, p(x_D | x_{1:D-1})$ are estimated in a single pass of D threads

Normalizing flow models. Examples

Masked Autoregressive Flow (MAF)

$$p(x) = \prod_i p(x_i | x_{1:i-1})$$ Target density is modelled as a product of one-dimensional densities, depending only on the previous values

$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | \mu_i, (\exp s_i)^2),\ \mu_i = \mu_i(x_{1:i-1}),\ s_i = s_i(x_{1:i-1})$, where $\mu_i()$ and $s_i()$ are neural networks

$x_i = z_i \exp s_i + \mu_i,\ z_i \sim \mathcal{N}(0,1)$ **Inverse:** $z_i = (x_i - \mu_i)/\exp s_i$ **—>** no need to compute μ_i^{-1} and s_i^{-1}

Masked Autoencoder for Distribution Estimation (MADE): allows to speed up MAF!

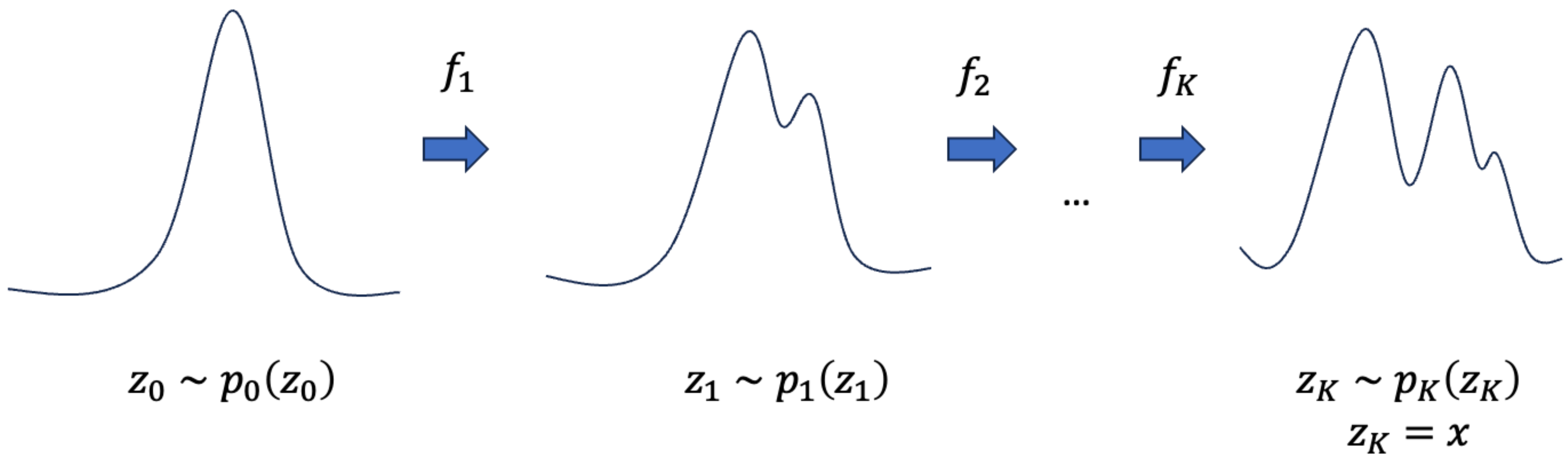
All conditional likelihoods $p(x_1), p(x_2 | x_1), \ldots, p(x_D | x_{1:D-1})$ are estimated in a single pass of D threads

Inverse Autoregressive Flow (IAF): $\mu_i = \mu_i(z_{1:i-1}),\ s_i = s_i(z_{1:i-1})$

somewhat similar to RealNVP

Training Normalising flows

In reality we apply a chain of transformations $f_1, \ldots f_K$ to the prior density $p(z)$



Loss <—negative log-likelihood: $-\log p_z(f_K \circ f_{K-1} \circ \ldots \circ f_1(z)) - \sum_i \log \det | \frac{df_i(z_i)}{dz_i} |$

With respect to function (bijector) parameters

Dequantization

- Normalizing flows operate on continuous distributions
- Most of the data we work with have discrete nature
- Hence dequantization: **converting** discrete data into **continuou**s by **adding small amount of noise**

Dequantization

- Normalizing flows operate on continuous distributions
- Most of the data we work with have discrete nature
- Hence dequantization: **converting** discrete data into **continuou**s by **adding small amount of noise**

1. $v = x + u,\ u \sim U[0,1]$, where x is the original discrete input

$$p(x) = \int p(x+u)du = \int \frac{q(u|x)}{q(u|x)} p(x+u)du = E_{u \sim q(u|x)} \frac{p(x+u)}{q(u|x)} = E_{u \sim U[0,1]^d} p(x+u)$$

Dequantization

- Normalizing flows operate on continuous distributions
- Most of the data we work with have discrete nature
- Hence dequantization: **converting** discrete data into **continuou**s by **adding small amount of noise**

1. $v = x + u,\ u \sim U[0,1]$, where x is the original discrete input

$$p(x) = \int p(x+u)du = \int \frac{q(u|x)}{q(u|x)} p(x+u)du = E_{u \sim q(u|x)} \frac{p(x+u)}{q(u|x)} = E_{u \sim U[0,1]^d} p(x+u)$$

Problems: uniform distribution has sharp boarders —> difficult to convert to normal

2. Solution: **Variational** dequantization. In the above formula use learnable distribution $q_\theta(u|x)$,

modelled via an additional normalizing flow

Jupyter notebook