Lecture 1. Normalizing flows

Introduction to Bayesian Statistical Learning II

03.06.2024 Instructors: Alina Bazarova, Oleg Filatov

Brief recall on the Bayesian concepts

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

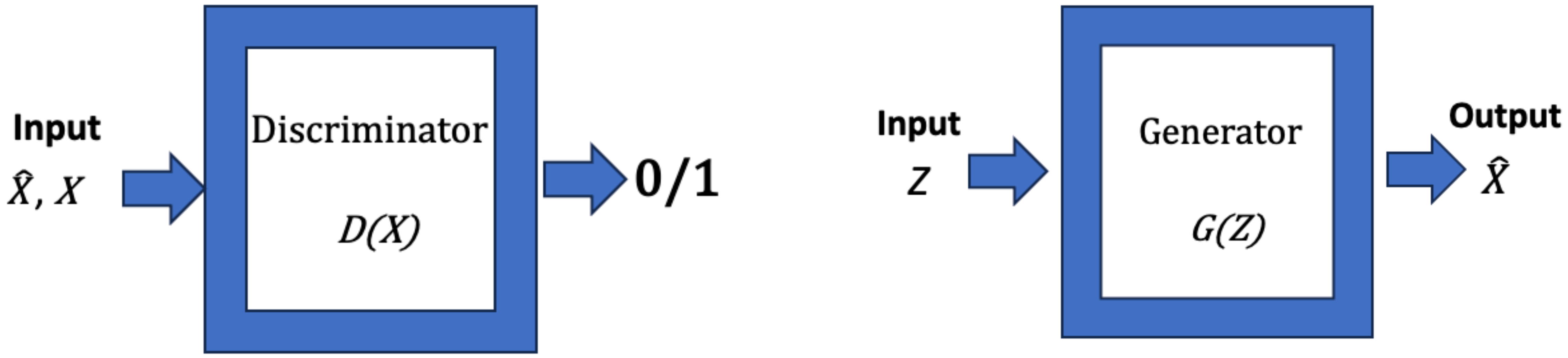
$$P(A|X) = \frac{P(A)P(X|A)}{P(X)}$$

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

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

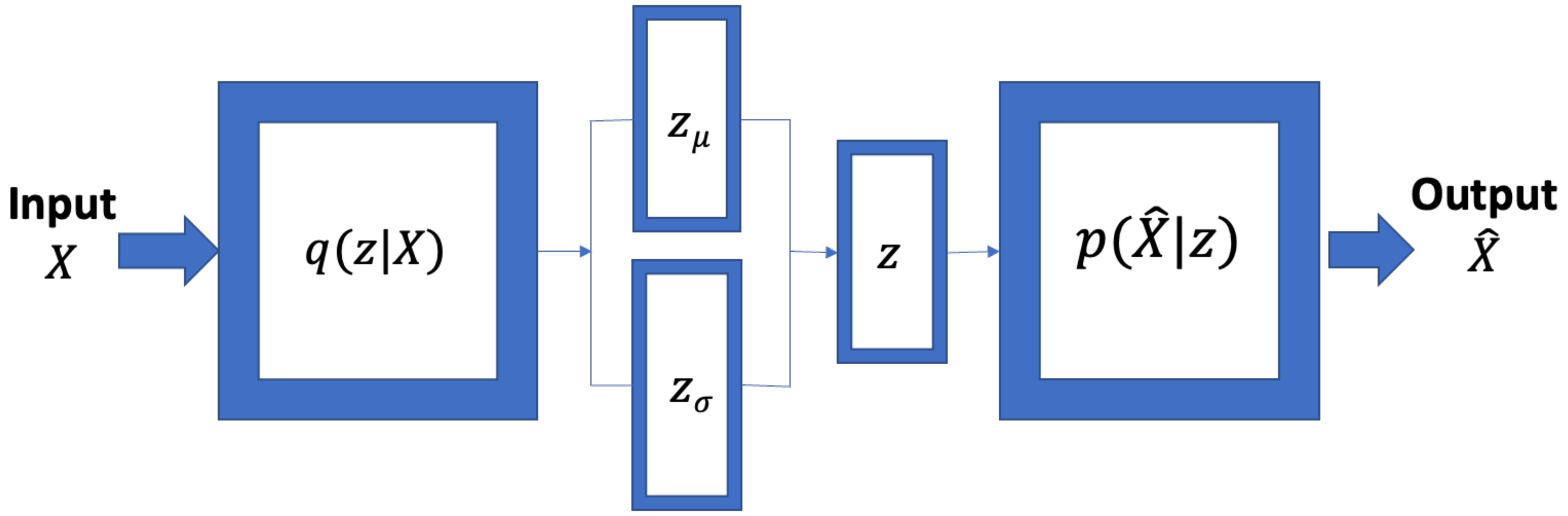
Where θ are the parameters, and $p(x)$ is a is a probability density function (continuous case)

Generative models



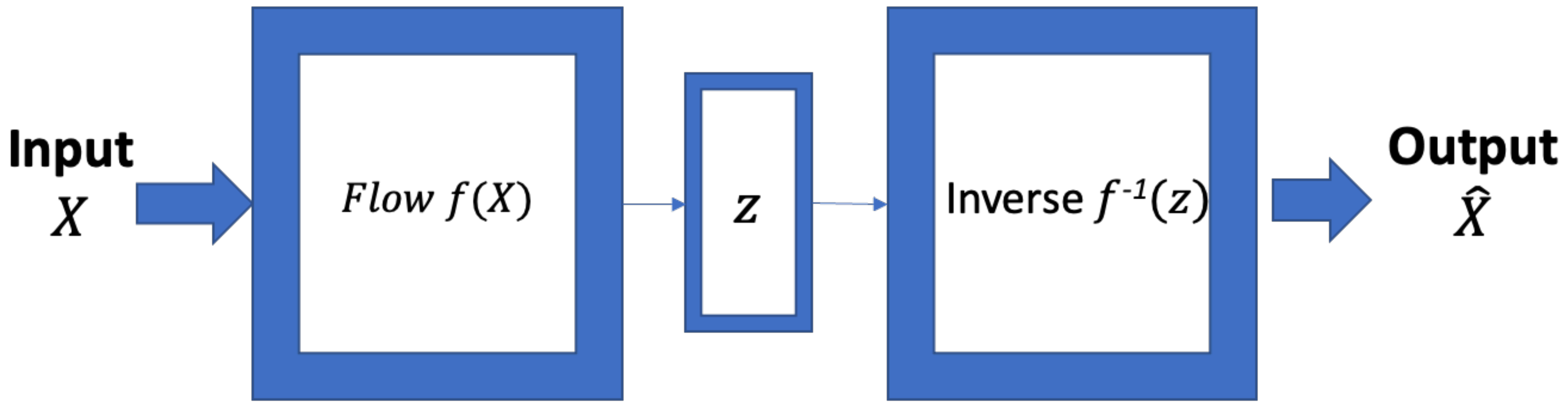
GAN: generator and discriminator trained together

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** which are easy to sample from 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 $f : \mathbb{R}^n \to \mathbb{R}^n$ 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 distribution**s 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 $f : \mathbb{R}^n \to \mathbb{R}^n$ 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
- $\frac{\partial f^{-1}(x)}{\partial x}$ is a Jacobian n x n matrix, where each (i, j) entry is $\frac{\partial f^{-1}(x)_i}{\partial x_j}$

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, $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^n$ 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

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

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

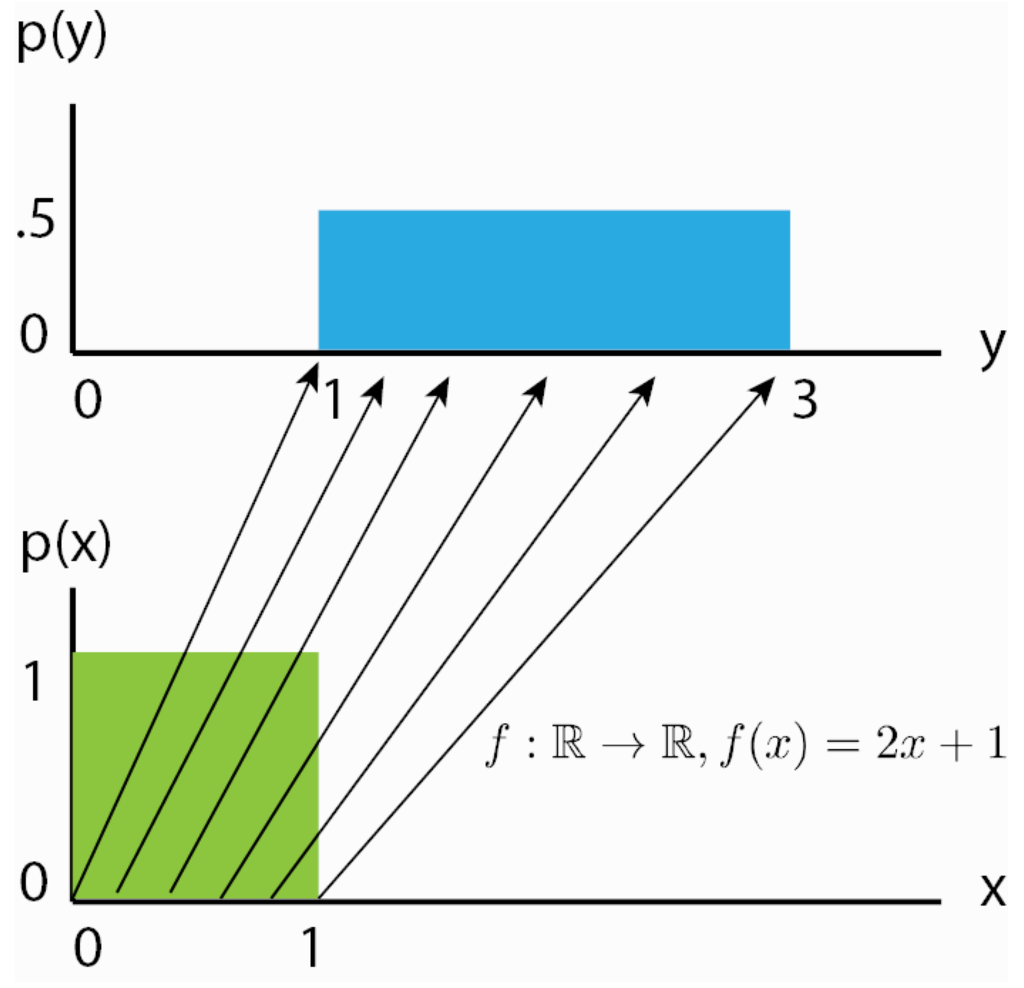


Figure credit Eric Jang https://blog.evjang.com/2018/01/nf1.html

<— Illustration: affine shift $p_X(x) = p_Z(f^{-1}(x))\,|\,det(\frac{\partial f^{-1}(x)}{\partial x})\,|$

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 x into two **disjoint subsets** x_1 and x_2

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

and m_θ is a **neural network**

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

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 x into two **disjoint subsets** x_1 and x_2

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

and m_θ is a **neural network**

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

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

Real Non-Volume Preserving (RealNVP)

$z_2 = \exp(s_\theta(x_1)) \odot x_2 + m_\theta(x_1)$ 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

Normalizing flow models. Examples

Masked Autoregressive Flow (MAF)

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

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

$$p(x_i \mid x_{1:i-1}) = \mathcal{N}(x_i \mid m_{i,\theta}, (\exp s_{i,\theta})^2), \; m_{i,\theta} = m_{i,\theta}(x_{1:i-1}), \; s_{i,\theta} = s_{i,\theta}(x_{1:i-1})$$

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

$$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | m_{i,\theta}, (\exp s_{i,\theta})^2),\ m_{i,\theta} = m_{i,\theta}(x_{1:i-1}),\ s_{i,\theta} = s_{i,\theta}(x_{1:i-1})$$

$x_i = z_i \exp s_{i,\theta} + m_{i,\theta},\ z_i \sim \mathcal{N}(0,1)$, where $m_{i,\theta}$ and $s_{i,\theta}$ are neural networks

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

$$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | m_{i,\theta}, (\exp s_{i,\theta})^2),\ m_{i,\theta} = m_{i,\theta}(x_{1:i-1}),\ s_{i,\theta} = s_{i,\theta}(x_{1:i-1})$$

$x_i = z_i \exp s_{i,\theta} + m_{i,\theta},\ z_i \sim \mathcal{N}(0,1)$, where $m_{i,\theta}$ and $s_{i,\theta}$ are neural networks

Inverse: $z_i = (x_i - m_{i,\theta})/\exp s_i$ **—>** no need to compute $m_{i,\theta}^{-1}$ and $s_{i,\theta}^{-1}$

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

$$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | m_{i,\theta}, (\exp s_{i,\theta})^2),\ m_{i,\theta} = m_{i,\theta}(x_{1:i-1}),\ s_{i,\theta} = s_{i,\theta}(x_{1:i-1})$$

$x_i = z_i \exp s_{i,\theta} + m_{i,\theta},\ z_i \sim \mathcal{N}(0,1)$, where $m_{i,\theta}$ and $s_{i,\theta}$ are neural networks

Inverse: $z_i = (x_i - m_{i,\theta})/\exp s_i$ **—>** no need to compute $m_{i,\theta}^{-1}$ and $s_{i,\theta}^{-1}$

The **generation** procedure is **slow,** since for x_i we need to compute all the previous $x_{1:i-1}$

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

$$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | m_{i,\theta}, (\exp s_{i,\theta})^2), \; m_{i,\theta} = m_{i,\theta}(x_{1:i-1}), \; s_{i,\theta} = s_{i,\theta}(x_{1:i-1})$$

$x_i = z_i \exp s_{i,\theta} + m_{i,\theta}, \; z_i \sim \mathcal{N}(0,1)$, where $m_{i,\theta}$ and $s_{i,\theta}$ are neural networks

Inverse: $z_i = (x_i - m_{i,\theta})/\exp s_{i,\theta}$ **—>** no need to compute $m_{i,\theta}^{-1}$ and $s_{i,\theta}^{-1}$

The **generation** procedure is **slow,** since for x_i we need to compute all the previous $x_{1:i-1}$

However, once x is computed, the density estimation can be significantly **speeded up** with **MADE!**

Normalizing flow models. Examples

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

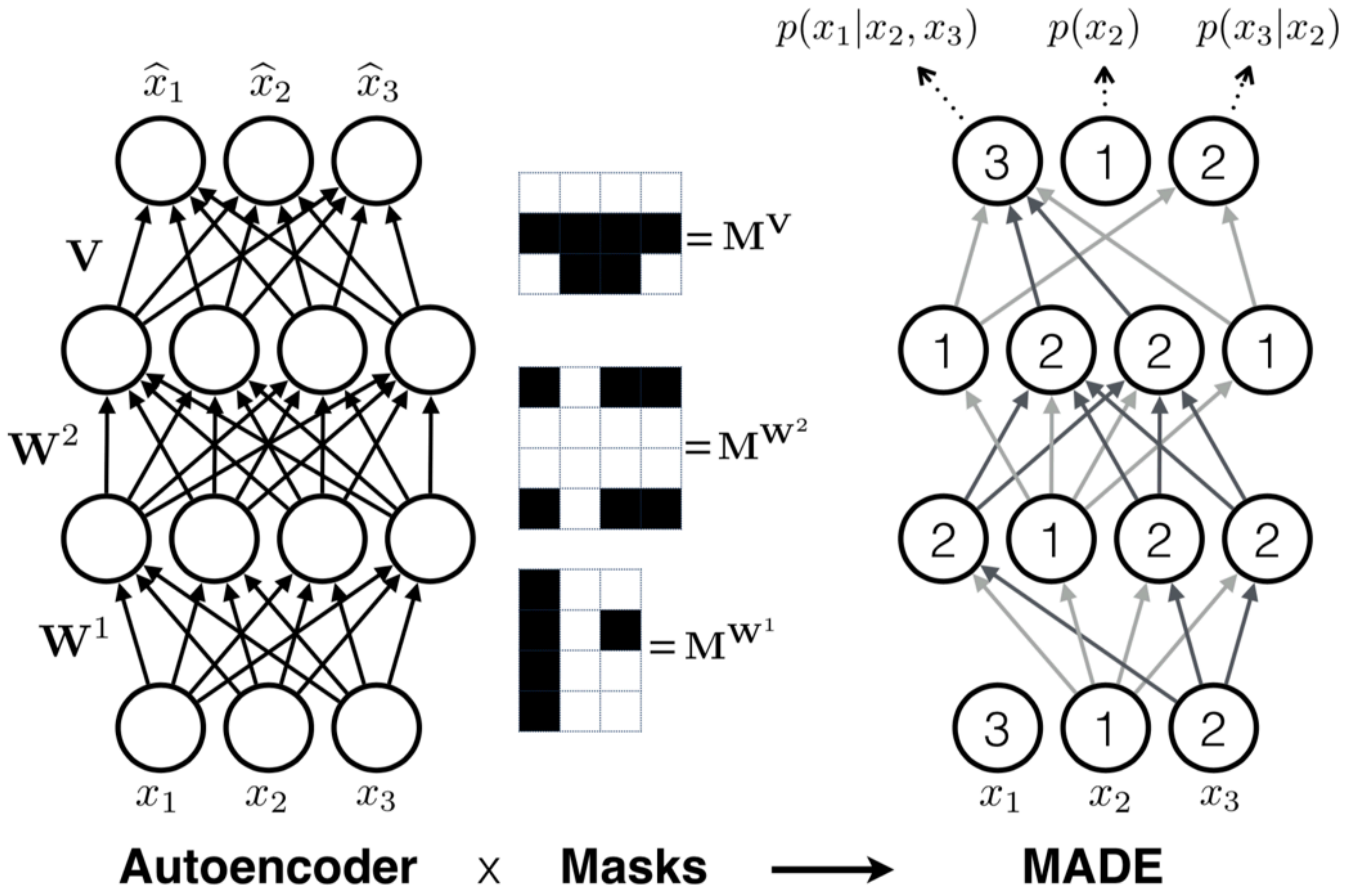
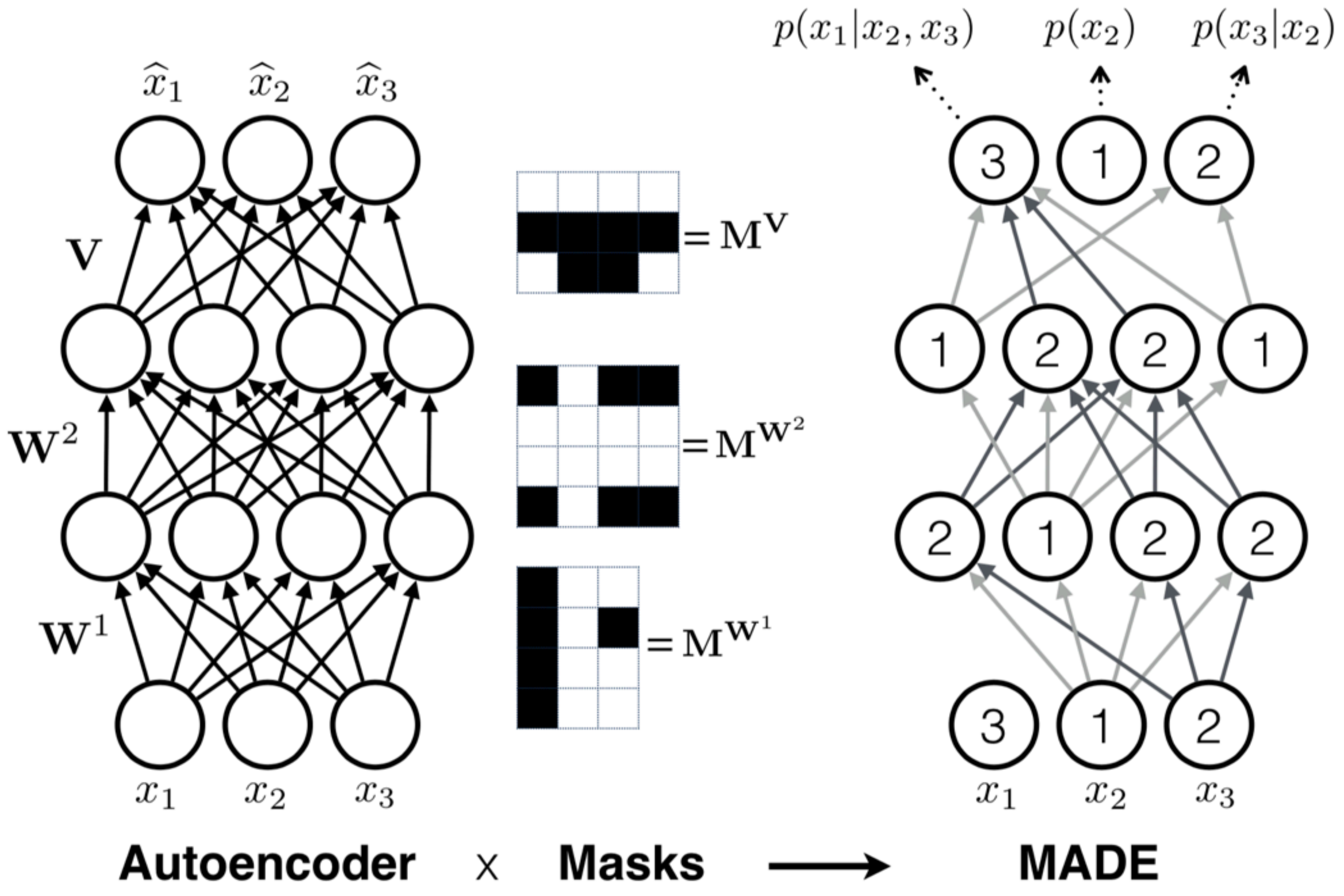


Image source https://arxiv.org/abs/1502.03509

Normalizing flow models. Examples

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



MADE **masks** out connections in the network

Each hidden node is assigned with a **random** "connectivity integer" , which determines the mask

Image source https://arxiv.org/abs/1502.03509

Normalizing flow models. Examples

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

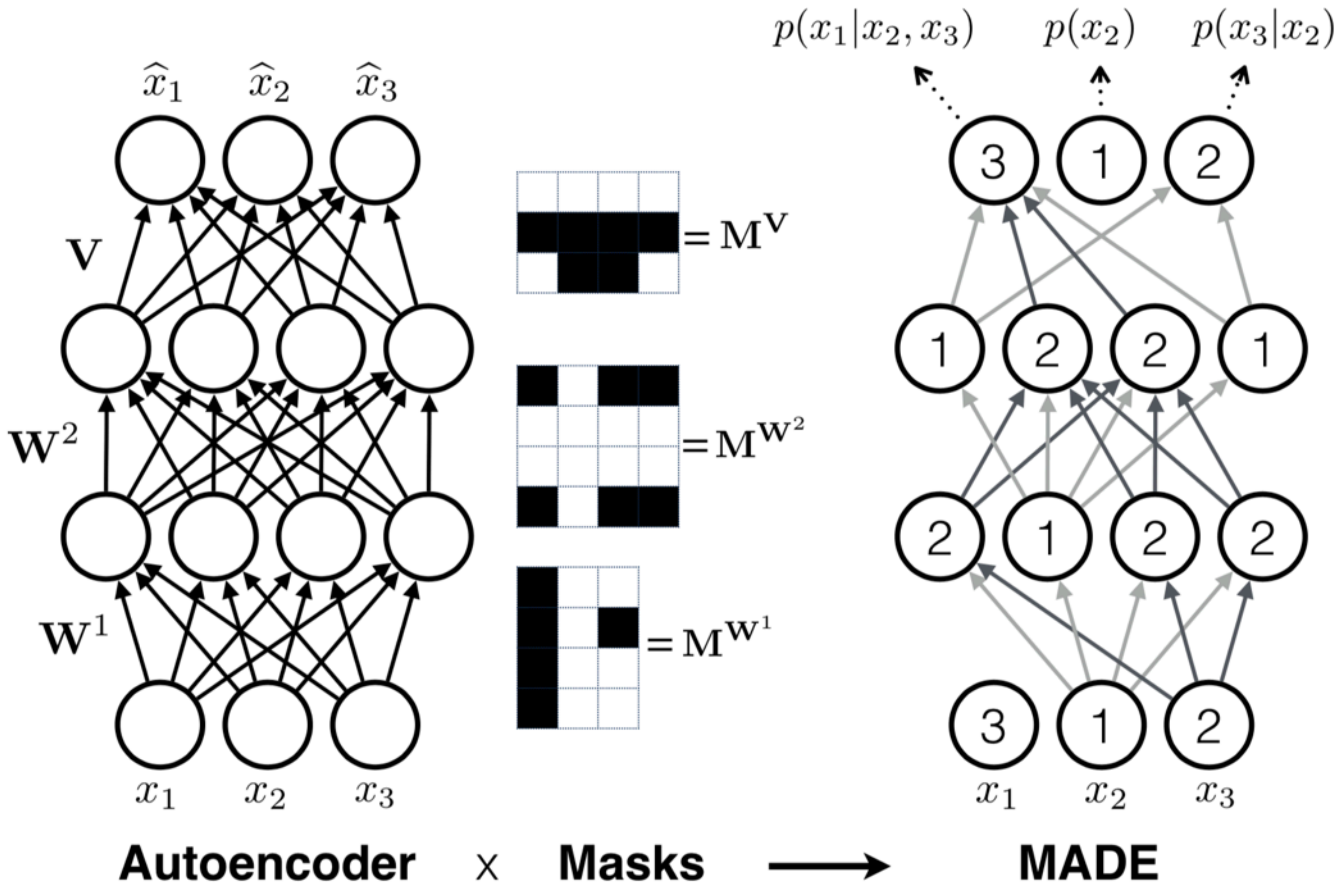


Image source https://arxiv.org/abs/1502.03509

MADE **masks** out connections in the network

Each hidden node is assigned with a **random** "connectivity integer" , which determines the mask

- Order agnostic (shuffle input dimensions)
- Connectivity agnostic (resample connectivity integers)

Normalizing flow models. Examples

Inverse Autoregressive Flow (IAF)

MAF transformation $x_i = z_i \exp s_{i,\theta} + m_{i,\theta},\ z_i \sim \mathcal{N}(0,1)$ x_i depends on $x_{1:i-1}$

Inverse $z_i = \dfrac{1}{\exp s_{i,\theta}} x_i - \dfrac{m_{i,\theta}}{\exp s_{i,\theta}}$ where $m_{i,\theta}$ and $s_{i,\theta}$ depend only on $x_{1:i-1}$

Normalizing flow models. Examples

Inverse Autoregressive Flow (IAF)

MAF transformation $x_i = z_i \exp s_{i,\theta} + m_{i,\theta},\ z_i \sim \mathcal{N}(0,1)$

Inverse $z_i = \dfrac{1}{\exp s_{i,\theta}} x_i - \dfrac{m_{i,\theta}}{\exp s_{i,\theta}}$ where $m_{i,\theta}$ and $s_{i,\theta}$ depend only on $x_{1:i-1}$

IAF: $\tilde{x}_i = \tilde{z}_i \exp \tilde{s}_{i,\theta} + \tilde{m}_{i,\theta}$, where $\tilde{m}_{i,\theta} = \tilde{m}_{i,\theta}(\tilde{z}_{1:i-1}),\ \tilde{s}_{i,\theta} = \tilde{s}_{i,\theta}(\tilde{z}_{1:i-1})$

Normalizing flow models. Examples

Inverse Autoregressive Flow (IAF)

MAF transformation $x_i = z_i \exp s_{i,\theta} + m_{i,\theta},\ z_i \sim \mathcal{N}(0,1)$

Inverse $z_i = \dfrac{1}{\exp s_{i,\theta}} x_i - \dfrac{m_{i,\theta}}{\exp s_{i,\theta}}$ where $m_{i,\theta}$ and $s_{i,\theta}$ depend only on $x_{1:i-1}$

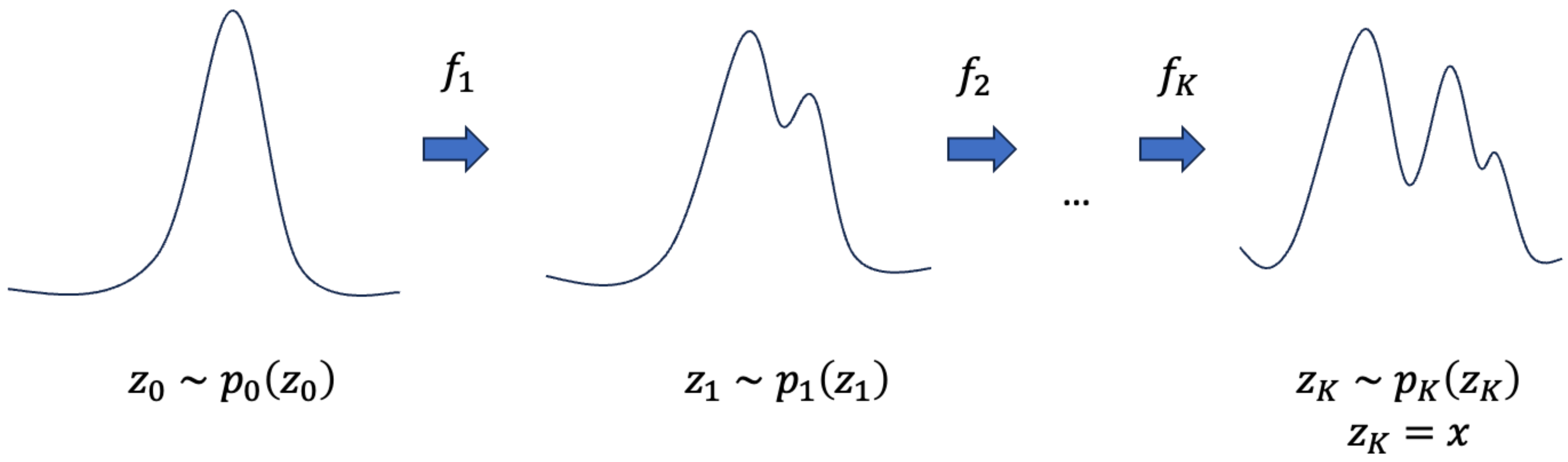
IAF: $\tilde{x}_i = \tilde{z}_i \exp \tilde{s}_{i,\theta} + \tilde{m}_{i,\theta}$, where $\tilde{m}_{i,\theta} = \tilde{m}_{i,\theta}(\tilde{z}_{1:i-1}),\ \tilde{s}_{i,\theta} = \tilde{s}_{i,\theta}(\tilde{z}_{1:i-1})$

Data generation: fast with MADE!

Density estimation: slow, as requires computation of $\tilde{z}_{1:i-1}$ since $\tilde{x}_i \sim p(\tilde{x}_i \,|\, \tilde{z}_{1:i-1})$

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: $-\log p_z(f_1^{-1} \circ f_2^{-1} \circ \ldots \circ f_K^{-1}(x)) - \sum_i \log \det | \frac{df_i^{-1}(z_i)}{dz_i} |$

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 **continuous** 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

$$p(x) = \int p(x+u)du =$$

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

$$p(x) = \int p(x+u)du = \int \frac{q(u|x)}{q(u|x)} p(x+u)du =$$

Dequantization

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

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

$$p(x) = \int p(x+u)du = \int \frac{q(u|x)}{q(u|x)} p(x+u)du = E_{u \sim q(u|x)} \frac{p(x+u)}{q(u|x)} =$$

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

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