Normalising flows with applications

ML Masterclass

David Huk and Rigers Behluli

University of Warwick

May 22, 2023

Summary

Summary

Introduction

Task: model probability distribution from data

Existing methods:

- Direct analytical approaches
- Variational approaches
- Expectation maximization
- VAEs and GANs

VAEs and GANs do not allow for likelihood evaluation and may present some training issues (e.g. mode collapse, posterior collapse, vanishing gradients and training instability) and

$\rightarrow$ Normalizing Flows ([Kobyzev et al., 2020], review paper)

Introduction

Task: model probability distribution from data

Existing methods:

- Direct analytical approaches
- Variational approaches
- Expectation maximization
- VAEs and GANs

VAEs and GANs do not allow for likelihood evaluation and may present some training issues (e.g. mode collapse, posterior collapse, vanishing gradients and training instability) and

$\rightarrow$ Normalizing Flows ([Kobyzev et al., 2020], review paper)

Introduction

Task: model probability distribution from data

Existing methods:

- Direct analytical approaches
- Variational approaches
- Expectation maximization
- VAEs and GANs

VAEs and GANs do not allow for likelihood evaluation and may present some training issues (e.g. mode collapse, posterior collapse, vanishing gradients and training instability) and

$\rightarrow$ Normalizing Flows ([Kobyzev et al., 2020], review paper)

Introduction

Task: model probability distribution from data

Existing methods:

- Direct analytical approaches
- Variational approaches
- Expectation maximization
- VAEs and GANs

VAEs and GANs do not allow for likelihood evaluation and may present some training issues (e.g. mode collapse, posterior collapse, vanishing gradients and training instability) and

$\rightarrow$ Normalizing Flows ([Kobyzev et al., 2020], review paper)

Introduction

Task: model probability distribution from data

Existing methods:

- Direct analytical approaches
- Variational approaches
- Expectation maximization
- VAEs and GANs

VAEs and GANs do not allow for likelihood evaluation and may present some training issues (e.g. mode collapse, posterior collapse, vanishing gradients and training instability) and

→ Normalizing Flows ([Kobyzev et al., 2020], review paper)

Introduction

Task: model probability distribution from data

Existing methods:

- Direct analytical approaches
- Variational approaches
- Expectation maximization
- VAEs and GANs

VAEs and GANs do not allow for likelihood evaluation and may present some training issues (e.g. mode collapse, posterior collapse, vanishing gradients and training instability) and

$\rightarrow$ Normalizing Flows ([Kobyzev et al., 2020], review paper)

Introduction

Task: model probability distribution from data

Existing methods:

- Direct analytical approaches
- Variational approaches
- Expectation maximization
- VAEs and GANs

VAEs and GANs do not allow for likelihood evaluation and may present some training issues (e.g. mode collapse, posterior collapse, vanishing gradients and training instability) and

$\rightarrow$ Normalizing Flows ([Kobyzev et al., 2020], review paper)

Summary

Normalising flows basics

Let $\mathbf{Z} \in \mathbb{R}^D$ be a random variable with a known probability density function $p_{\mathbf{Z}} : \mathbb{R} \to \mathbb{R}$ and let $\mathbf{g}$ be an invertible function and $\mathbf{Y} = \mathbf{g}(\mathbf{Z})$.
Using the change of variables formula, one can compute:

$$p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{Z}}(f(\mathbf{y}))|\det \mathrm{D}\mathbf{g}(\mathbf{f}(\mathbf{y}))|^{-1},$$

where $\mathbf{f}$ is the inverse of $\mathbf{g}$ and $\det \mathrm{D}\mathbf{g}(\mathbf{z}) = \frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ is the Jacobian of g.

The function $\mathbf{g}$ transforms the the base density $p_{\mathbf{Z}}$ into a more complex density, defining also it's generative direction.

The inverse of $\mathbf{g}$ instead moves in the opposite direction, from a complicated density to a know one.

Normalising flows basics

Let $\mathbf{Z} \in \mathbb{R}^D$ be a random variable with a known probability density function $p_\mathbf{Z} : \mathbb{R} \to \mathbb{R}$ and let $\mathbf{g}$ be an invertible function and $\mathbf{Y} = \mathbf{g}(\mathbf{Z})$. Using the change of variables formula, one can compute:

$$p_\mathbf{Y}(\mathbf{y}) = p_\mathbf{Z}(f(\mathbf{y}))|\det \mathrm{D}\mathbf{g}(\mathbf{f}(\mathbf{y}))|^{-1},$$

where $\mathbf{f}$ is the inverse of $\mathbf{g}$ and $\det \mathrm{D}\mathbf{g}(\mathbf{z}) = \frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ is the Jacobian of g.

The function $\mathbf{g}$ transforms the the base density $p_\mathbf{Z}$ into a more complex density, defining also it's generative direction.

The inverse of $\mathbf{g}$ instead moves in the opposite direction, from a complicated density to a know one.

Normalising flows basics

Let $\mathbf{Z} \in \mathbb{R}^D$ be a random variable with a known probability density function $p_{\mathbf{Z}} : \mathbb{R} \to \mathbb{R}$ and let $\mathbf{g}$ be an invertible function and $\mathbf{Y} = \mathbf{g}(\mathbf{Z})$. Using the change of variables formula, one can compute:

$$p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{Z}}(f(\mathbf{y}))|\det \mathrm{D}\mathbf{g}(\mathbf{f}(\mathbf{y}))|^{-1},$$

where $\mathbf{f}$ is the inverse of $\mathbf{g}$ and $\det \mathrm{D}\mathbf{g}(\mathbf{z}) = \frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ is the Jacobian of g.

The function $\mathbf{g}$ transforms the the base density $p_{\mathbf{Z}}$ into a more complex density, defining also it's generative direction.

The inverse of $\mathbf{g}$ instead moves in the opposite direction, from a complicated density to a know one.

Normalising flows basics

Let $\mathbf{Z} \in \mathbb{R}^D$ be a random variable with a known probability density function $p_{\mathbf{Z}} : \mathbb{R} \to \mathbb{R}$ and let $\mathbf{g}$ be an invertible function and $\mathbf{Y} = \mathbf{g}(\mathbf{Z})$. Using the change of variables formula, one can compute:

$$p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{Z}}(f(\mathbf{y}))|\det \mathrm{D}\mathbf{g}(\mathbf{f}(\mathbf{y}))|^{-1},$$

where $\mathbf{f}$ is the inverse of $\mathbf{g}$ and $\det \mathrm{D}\mathbf{g}(\mathbf{z}) = \frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ is the Jacobian of g.

The function $\mathbf{g}$ transforms the the base density $p_{\mathbf{Z}}$ into a more complex density, defining also it's generative direction.

The inverse of $\mathbf{g}$ instead moves in the opposite direction, from a complicated density to a know one.

Summary

Application - Density estimation

- Data $y_1, \ldots, y_n \sim p_Y$
- Base noise $z \sim p_Z(.; \phi)$
- Flow model $y = \mathbf{g}_\theta(z)$ with inverse $\mathbf{f}_\theta$

$$\log p(y_1, \ldots, y_n \mid \theta, \phi) = \sum_{i=1}^{n} \log p_Y(y_i \mid \theta, \phi)$$
$$= \sum_{i=1}^{n} \underbrace{\log p_Z(\mathbf{f}(y_i \mid \theta) \mid \phi)}_{\text{ll under base measure}} + \underbrace{\log \left|\det D\mathbf{f}(y_i \mid \theta)\right|}_{\text{volume correction}}$$

During training, estimate θ, ϕ to maximise the above ll.
Computational cost depends on computing $\mathbf{f}$ and $\det D\mathbf{f}$ - the *normalising direction*.

Application - sampling

To sample from p_Y, it is enough to generate noise $z \sim p_Z$ and apply the flow $\mathbf{g}$.
Performance determined by the cost of applying $\mathbf{g}$ - the *generative direction*.

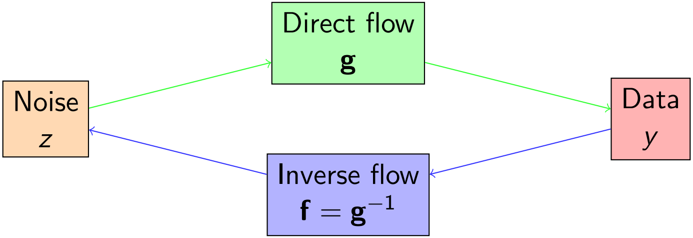


Figure: Generative direction (sampling) and Normalising direction (likelihood).

In general, want **invertible** and **expressive** $\mathbf{g}$ with **efficient evaluation** of $\mathbf{g}, \mathbf{f}$ and $\det D\mathbf{f}$. Different flows have different strengths.

Summary

Elementwise Flows

Normalizing flows should satisfy several conditions to be practical:

- be invertible,
- be sufficiently expressive to model the distribution of interest,
- be computationally efficient.

An example could be to use any bijective scalar function.
Let $h : \mathbb{R} \to \mathbb{R}$ be a scalar valued bijection. If $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, then

$$\mathbf{g}(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_D))^T$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the absolute values of the derivatives of h.

Very simple approach, but limiting.

Elementwise Flows

Normalizing flows should satisfy several conditions to be practical:

- be invertible,
- be sufficiently expressive to model the distribution of interest,
- be computationally efficient.

An example could be to use any bijective scalar function.
Let $h : \mathbb{R} \to \mathbb{R}$ be a scalar valued bijection. If $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, then

$$\mathbf{g}(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_D))^T$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the absolute values of the derivatives of h.

Very simple approach, but limiting.

Elementwise Flows

Normalizing flows should satisfy several conditions to be practical:

- be invertible,
- be sufficiently expressive to model the distribution of interest,
- be computationally efficient.

An example could be to use any bijective scalar function.
Let $h : \mathbb{R} \to \mathbb{R}$ be a scalar valued bijection. If $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, then

$$\mathbf{g}(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_D))^T$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the absolute values of the derivatives of h.

Very simple approach, but limiting.

Elementwise Flows

Normalizing flows should satisfy several conditions to be practical:

- be invertible,
- be sufficiently expressive to model the distribution of interest,
- be computationally efficient.

An example could be to use any bijective scalar function.
Let $h : \mathbb{R} \to \mathbb{R}$ be a scalar valued bijection. If $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, then

$$\mathbf{g}(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_D))^T$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the absolute values of the derivatives of h.

Very simple approach, but limiting.

Elementwise Flows

Normalizing flows should satisfy several conditions to be practical:

- be invertible,
- be sufficiently expressive to model the distribution of interest,
- be computationally efficient.

An example could be to use any bijective scalar function.
Let $h : \mathbb{R} \to \mathbb{R}$ be a scalar valued bijection. If $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, then

$$\mathbf{g}(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_D))^T$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the absolute values of the derivatives of h.

Very simple approach, but limiting.

Elementwise Flows

Normalizing flows should satisfy several conditions to be practical:

- be invertible,
- be sufficiently expressive to model the distribution of interest,
- be computationally efficient.

An example could be to use any bijective scalar function.
Let $h : \mathbb{R} \to \mathbb{R}$ be a scalar valued bijection. If $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, then

$$\mathbf{g}(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_D))^T$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the absolute values of the derivatives of h.

Very simple approach, but limiting.

Linear Flows

Linear mappings can express correlation between dimensions.

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

where A and b are parameters. If A is invertible, then the function is invertible too. Determinant of the Jacobian can be computed in $\mathcal{O}(D^3)$.

A can be:

- diagonal: close to elementwise flows,
- upper triangular: captures correlation between dimensions and can be computed in $\mathcal{O}(D^2)$,
- permutation and orthogonal matrices can be used for faster computations,
- factorization: can be computed in $\mathcal{O}(D)$

$$\mathbf{g}(\mathbf{x}) = \mathbf{PLU}\mathbf{x} + \mathbf{b}$$

Linear Flows

Linear mappings can express correlation between dimensions.

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

where A and b are parameters. If A is invertible, then the function is invertible too. Determinant of the Jacobian can be computed in $\mathcal{O}(D^3)$.

A can be:

- diagonal: close to elementwise flows,
- upper triangular: captures correlation between dimensions and can be computed in $\mathcal{O}(D^2)$,
- permutation and orthogonal matrices can be used for faster computations,
- factorization: can be computed in $\mathcal{O}(D)$

$$\mathbf{g}(\mathbf{x}) = \mathbf{PLU}\mathbf{x} + \mathbf{b}$$

Linear Flows

Linear mappings can express correlation between dimensions.

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

where A and b are parameters. If A is invertible, then the function is invertible too. Determinant of the Jacobian can be computed in $\mathcal{O}(D^3)$.

A can be:

- diagonal: close to elementwise flows,
- upper triangular: captures correlation between dimensions and can be computed in $\mathcal{O}(D^2)$,
- permutation and orthogonal matrices can be used for faster computations,
- factorization: can be computed in $\mathcal{O}(D)$

$$\mathbf{g}(\mathbf{x}) = \mathbf{PLU}\mathbf{x} + \mathbf{b}$$

Linear Flows

Linear mappings can express correlation between dimensions.

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

where A and b are parameters. If A is invertible, then the function is invertible too. Determinant of the Jacobian can be computed in $\mathcal{O}(D^3)$.

A can be:

- diagonal: close to elementwise flows,
- upper triangular: captures correlation between dimensions and can be computed in $\mathcal{O}(D^2)$,
- permutation and orthogonal matrices can be used for faster computations,
- factorization: can be computed in $\mathcal{O}(D)$

$$\mathbf{g}(\mathbf{x}) = \mathbf{PLU}\mathbf{x} + \mathbf{b}$$

Linear Flows

Linear mappings can express correlation between dimensions.

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

where A and b are parameters. If A is invertible, then the function is invertible too. Determinant of the Jacobian can be computed in $\mathcal{O}(D^3)$.

A can be:

- diagonal: close to elementwise flows,
- upper triangular: captures correlation between dimensions and can be computed in $\mathcal{O}(D^2)$,
- permutation and orthogonal matrices can be used for faster computations,
- factorization: can be computed in $\mathcal{O}(D)$

$$\mathbf{g}(\mathbf{x}) = \mathbf{PLU}\mathbf{x} + \mathbf{b}$$

Coupling Flows (1)

Introduced by [Dinh et al., 2014], the idea is to only transform subsets of data at a time.

- Split input $\mathbf{x} \in R^D$ into $\mathbf{x}^A \in \mathbb{R}^d$ and $\mathbf{x}^B \in \mathbb{R}^{D-d}$.
- *Coupling function* is a bijection $\mathbf{h}(., \theta) : R^d \to R^d$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Define the *coupling flow* $\mathbf{g} : \mathbb{R}^D \to R^D$ as:

$$\mathbf{y}^A = \mathbf{h}(\mathbf{x}^A, \theta = \Theta(\mathbf{x}^B))$$
$$\mathbf{y}^B = \mathbf{x}^B$$

Coupling Flows (1)

Introduced by [Dinh et al., 2014], the idea is to only transform subsets of data at a time.

- Split input $\mathbf{x} \in R^D$ into $\mathbf{x}^A \in \mathbb{R}^d$ and $\mathbf{x}^B \in \mathbb{R}^{D-d}$.
- *Coupling function* is a bijection $\mathbf{h}(., \theta) : R^d \to R^d$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Define the *coupling flow* $\mathbf{g} : \mathbb{R}^D \to R^D$ as:

$$\mathbf{y}^A = \mathbf{h}(\mathbf{x}^A, \theta = \Theta(\mathbf{x}^B))$$
$$\mathbf{y}^B = \mathbf{x}^B$$

Coupling Flows (1)

Introduced by [Dinh et al., 2014], the idea is to only transform subsets of data at a time.

- Split input $\mathbf{x} \in R^D$ into $\mathbf{x}^A \in \mathbb{R}^d$ and $\mathbf{x}^B \in \mathbb{R}^{D-d}$.
- *Coupling function* is a bijection $\mathbf{h}(., \theta) : R^d \to R^d$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Define the *coupling flow* $\mathbf{g} : \mathbb{R}^D \to R^D$ as:

$$
\begin{aligned}
\mathbf{y}^A &= \mathbf{h}(\mathbf{x}^A, \theta = \Theta(\mathbf{x}^B)) \\
\mathbf{y}^B &= \mathbf{x}^B
\end{aligned}
$$

Coupling Flows (1)

Introduced by [Dinh et al., 2014], the idea is to only transform subsets of data at a time.

- Split input $\mathbf{x} \in R^D$ into $\mathbf{x}^A \in \mathbb{R}^d$ and $\mathbf{x}^B \in \mathbb{R}^{D-d}$.
- *Coupling function* is a bijection $\mathbf{h}(., \theta) : R^d \to R^d$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Define the *coupling flow* $\mathbf{g} : \mathbb{R}^D \to R^D$ as:

$$\mathbf{y}^A = \mathbf{h}(\mathbf{x}^A, \theta = \Theta(\mathbf{x}^B))$$
$$\mathbf{y}^B = \mathbf{x}^B$$

Coupling Flows (1)

Introduced by [Dinh et al., 2014], the idea is to only transform subsets of data at a time.

- Split input $\mathbf{x} \in R^D$ into $\mathbf{x}^A \in \mathbb{R}^d$ and $\mathbf{x}^B \in \mathbb{R}^{D-d}$.
- *Coupling function* is a bijection $\mathbf{h}(., \theta) : R^d \to R^d$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Define the *coupling flow* $\mathbf{g} : \mathbb{R}^D \to R^D$ as:

$$\mathbf{y}^A = \mathbf{h}(\mathbf{x}^A, \theta = \Theta(\mathbf{x}^B))$$
$$\mathbf{y}^B = \mathbf{x}^B$$

Coupling Flows (2)

The Jacobian of $\mathbf{g}$ is a block triangular matrix where diagonal blocks are $D\mathbf{h}$ and the identity. So the determinant of the Jacobian is $\det D\mathbf{h}$.

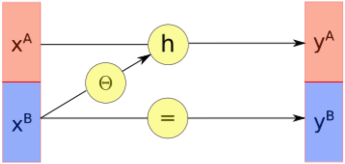


Figure: A single coupling flow illustration from [Kobyzev et al., 2020].

The power of coupling flows is in the ability to have very complex models for $\Theta(\mathbf{x}^B)$ (which do not need to be invertible), such as neural networks.

Coupling Flows (2)

The Jacobian of $\mathbf{g}$ is a block triangular matrix where diagonal blocks are $D\mathbf{h}$ and the identity. So the determinant of the Jacobian is $\det D\mathbf{h}$.

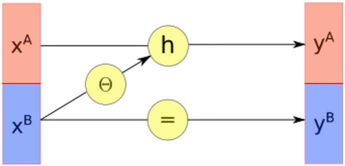


Figure: A single coupling flow illustration from [Kobyzev et al., 2020].

The power of coupling flows is in the ability to have very complex models for $\Theta(\mathbf{x}^B)$ (which do not need to be invertible), such as neural networks.

Autoregressive Flows (1)

Similar to coupling flows, but with each entry of outputs $\mathbf{y} = \mathbf{g}(\mathbf{x})$ conditioned on the previous entries of the input.

- *Coupling function* is a bijection $h(., \theta) : R \to R$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Then the AR flow is given by:

$$y_t = h\left(x_t; \Theta_t\left(\mathbf{x}_{1:t-1}\right)\right), \quad t \in \{1, \dots, D\}.$$

The Jacobian of $\mathbf{g}$ is triangular with inputs y_t depending only on $x_{1:t}$, so the determinant is a product of its diagonal entries.
Furthermore, $\mathbf{g}$ can be parrallelised easily due to a dependence on available subsets of $\mathbf{x}$ only.

Autoregressive Flows (1)

Similar to coupling flows, but with each entry of outputs $\mathbf{y} = \mathbf{g}(\mathbf{x})$ conditioned on the previous entries of the input.

- *Coupling function* is a bijection $h(., \theta) : R \to R$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Then the AR flow is given by:

$$y_t = h\left(x_t; \Theta_t\left(\mathbf{x}_{1:t-1}\right)\right), \quad t \in \{1, \ldots, D\}.$$

The Jacobian of $\mathbf{g}$ is triangular with inputs y_t depending only on $x_{1:t}$, so the determinant is a product of its diagonal entries.
Furthermore, $\mathbf{g}$ can be parrallelised easily due to a dependence on available subsets of $\mathbf{x}$ only.

Autoregressive Flows (1)

Similar to coupling flows, but with each entry of outputs $\mathbf{y} = \mathbf{g}(\mathbf{x})$ conditioned on the previous entries of the input.

- *Coupling function* is a bijection $h(., \theta) : R \rightarrow R$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Then the AR flow is given by:

$$y_t = h\left(x_t; \Theta_t\left(\mathbf{x}_{1:t-1}\right)\right), \quad t \in \{1, \dots, D\}.$$

The Jacobian of $\mathbf{g}$ is triangular with inputs y_t depending only on $x_{1:t}$, so the determinant is a product of its diagonal entries.
Furthermore, $\mathbf{g}$ can be parrallelised easily due to a dependence on available subsets of $\mathbf{x}$ only.

Autoregressive Flows (1)

Similar to coupling flows, but with each entry of outputs $\mathbf{y} = \mathbf{g}(\mathbf{x})$ conditioned on the previous entries of the input.

- *Coupling function* is a bijection $h(., \theta) : R \to R$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Then the AR flow is given by:

$$y_t = h\left(x_t; \Theta_t\left(\mathbf{x}_{1:t-1}\right)\right), \quad t \in \{1, \ldots, D\}.$$

The Jacobian of $\mathbf{g}$ is triangular with inputs y_t depending only on $x_{1:t}$, so the determinant is a product of its diagonal entries.
Furthermore, $\mathbf{g}$ can be parrallelised easily due to a dependence on available subsets of $\mathbf{x}$ only.

Autoregressive Flows (1)

Similar to coupling flows, but with each entry of outputs $\mathbf{y} = \mathbf{g}(\mathbf{x})$ conditioned on the previous entries of the input.

- *Coupling function* is a bijection $h(., \theta) : R \to R$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Then the AR flow is given by:

$$y_t = h\left(x_t; \Theta_t\left(\mathbf{x}_{1:t-1}\right)\right), \quad t \in \{1, \ldots, D\}.$$

The Jacobian of $\mathbf{g}$ is triangular with inputs y_t depending only on $x_{1:t}$, so the determinant is a product of its diagonal entries.

Furthermore, $\mathbf{g}$ can be parrallelised easily due to a dependence on available subsets of $\mathbf{x}$ only.

Autoregressive Flows (1)

Similar to coupling flows, but with each entry of outputs $\mathbf{y} = \mathbf{g}(\mathbf{x})$ conditioned on the previous entries of the input.

- *Coupling function* is a bijection $h(., \theta) : R \to R$.
- *Conditioner* $\Theta(.)$ used to define θ given inputs.

Then the AR flow is given by:

$$y_t = h\left(x_t; \Theta_t\left(\mathbf{x}_{1:t-1}\right)\right), \quad t \in \{1, \ldots, D\}.$$

The Jacobian of $\mathbf{g}$ is triangular with inputs y_t depending only on $x_{1:t}$, so the determinant is a product of its diagonal entries.
Furthermore, $\mathbf{g}$ can be parrallelised easily due to a dependence on available subsets of $\mathbf{x}$ only.

Autoregressive Flows (2)

However, the inverse flow **f** is more complicated and is found recursively, making it hard to implement efficiently.

An alternative is proposed by [Kingma et al., 2016] under the name of *inverse autoregressive flows* (IAF), which outputs terms from **y** conditioned on previous entries of **y**:

$$y_t = h(x_t; \Theta_t(\mathbf{y}_{1:t-1}))$$

While most flows are modeled in the normalising direction to ensure efficient evaluations of **f**, the IAF does the opposite and models the generative direction.

- Classical AR flows have efficient density estimation.
- IAFs have efficient sampling.

Autoregressive Flows (2)

However, the inverse flow **f** is more complicated and is found recursively, making it hard to implement efficiently.
An alternative is proposed by [Kingma et al., 2016] under the name of *inverse autoregressive flows* (IAF), which outputs terms from **y** conditioned on previous entries of **y**:

$$y_t = h(x_t; \Theta_t(\mathbf{y}_{1:t-1}))$$

While most flows are modeled in the normalising direction to ensure efficient evaluations of **f**, the IAF does the opposite and models the generative direction.

- Classical AR flows have efficient density estimation.
- IAFs have efficient sampling.

Autoregressive Flows (2)

However, the inverse flow $\mathbf{f}$ is more complicated and is found recursively, making it hard to implement efficiently.
An alternative is proposed by [Kingma et al., 2016] under the name of *inverse autoregressive flows* (IAF), which outputs terms from $\mathbf{y}$ conditioned on previous entries of $\mathbf{y}$:

$$y_t = h(x_t; \Theta_t(\mathbf{y}_{1:t-1}))$$

While most flows are modeled in the normalising direction to ensure efficient evaluations of $\mathbf{f}$, the IAF does the opposite and models the generative direction.

- Classical AR flows have efficient density estimation.
- IAFs have efficient sampling.

Autoregressive Flows (2)

However, the inverse flow $\mathbf{f}$ is more complicated and is found recursively, making it hard to implement efficiently.
An alternative is proposed by [Kingma et al., 2016] under the name of *inverse autoregressive flows* (IAF), which outputs terms from $\mathbf{y}$ conditioned on previous entries of $\mathbf{y}$:

$$y_t = h(x_t; \Theta_t(\mathbf{y}_{1:t-1}))$$

While most flows are modeled in the normalising direction to ensure efficient evaluations of $\mathbf{f}$, the IAF does the opposite and models the generative direction.

- Classical AR flows have efficient density estimation.
- IAFs have efficient sampling.

Autoregressive Flows (3)

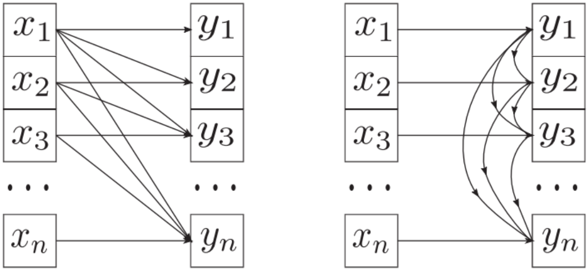
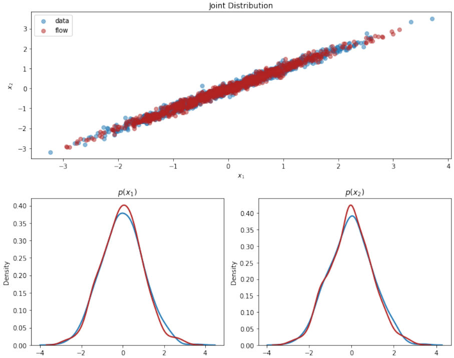


Figure: Left: classical Autoregressive Flows. Right: Inverse Autoregressive FLows.

Summary

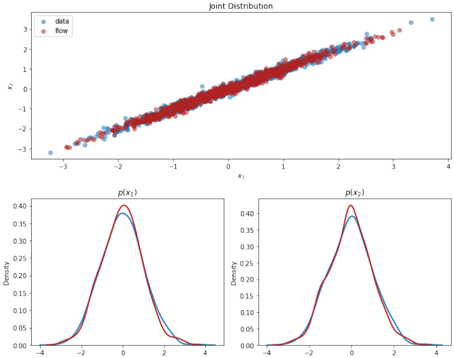
Learning a multivariate Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density if $\mathcal{N}_2\left(\mathbf{0}, \begin{bmatrix} 1 & 0.99 \\ 0.99 & 1 \end{bmatrix}\right)$.
- Use a coupling flow with affine h as $\begin{cases} y_1 = & h(x_1;\theta) = \theta_1 x_1 + \theta_2 \\ y_2 = & x_2 \end{cases}$



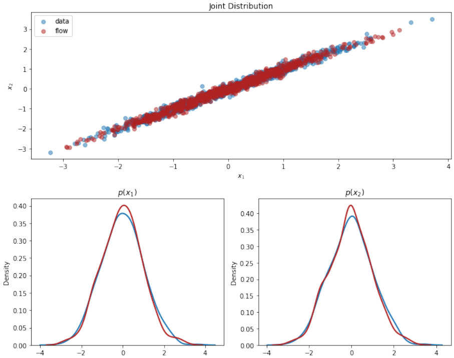
Learning a multivariate Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density if $\mathcal{N}_2\left(\mathbf{0}, \begin{bmatrix} 1 & 0.99 \\ 0.99 & 1 \end{bmatrix}\right)$.
- Use a coupling flow with affine h as $\begin{cases} y_1 = & h(x_1; \theta) = \theta_1 x_1 + \theta_2 \\ y_2 = & x_2 \end{cases}$



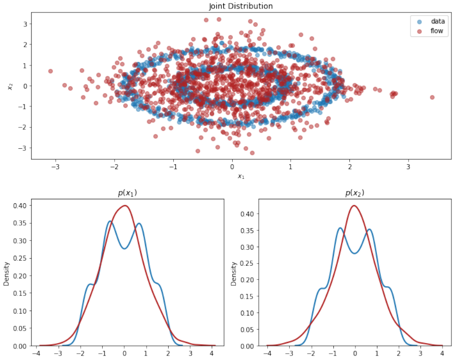
Learning a multivariate Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density if $\mathcal{N}_2\left(\mathbf{0}, \begin{bmatrix} 1 & 0.99 \\ 0.99 & 1 \end{bmatrix}\right)$.
- Use a coupling flow with affine h as $\begin{cases} y_1 = & h(x_1;\theta) = \theta_1 x_1 + \theta_2 \\ y_2 = & x_2 \end{cases}$



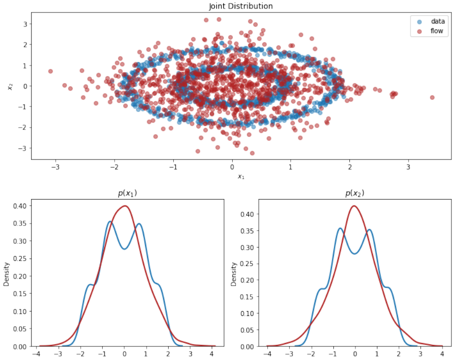
Learning a non-Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density is uniform on two circles.
- Use a coupling flow with affine h as $\begin{cases} y_1 = & h(x_1; \boldsymbol{\theta}) = \theta_1 x_1 + \theta_2 \\ y_2 = & x_2 \end{cases}$



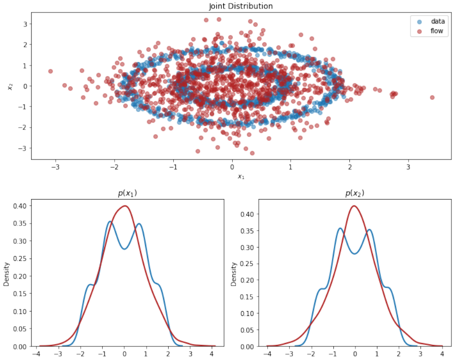
Learning a non-Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density is uniform on two circles.
- Use a coupling flow with affine h as $\begin{cases} y_1 = & h(x_1; \boldsymbol{\theta}) = \theta_1 x_1 + \theta_2 \\ y_2 = & x_2 \end{cases}$



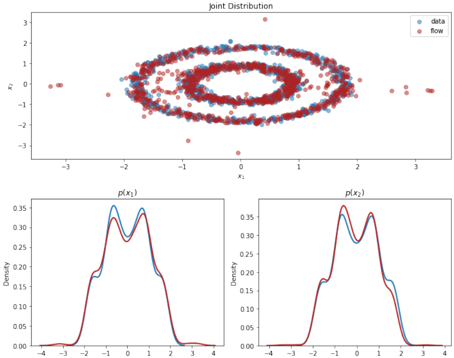
Learning a non-Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density is uniform on two circles.
- Use a coupling flow with affine h as $\begin{cases} y_1 = & h(x_1; \boldsymbol{\theta}) = \theta_1 x_1 + \theta_2 \\ y_2 = & x_2 \end{cases}$



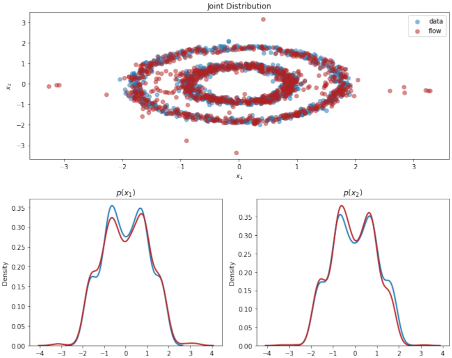
Learning a multivariate Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density is uniform on two circles.
- Use a coupling flow with spline h as $\begin{cases} y_1 = & h(x_1; \Theta(x_2)) = s_\theta(x_1; x_2) \\ y_2 = & x_2 \end{cases}$



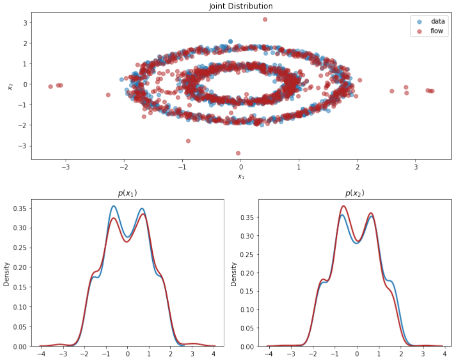
Learning a multivariate Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density is uniform on two circles.
- Use a coupling flow with spline h as $\begin{cases} y_1 = & h(x_1; \Theta(x_2)) = s_\theta(x_1; x_2) \\ y_2 = & x_2 \end{cases}$



Learning a multivariate Gaussian

- Base noise is $\mathcal{N}_2(\mathbf{0}, \mathbb{I}_2)$.
- Target density is uniform on two circles.
- Use a coupling flow with spline h as $\begin{cases} y_1 = & h(x_1; \Theta(x_2)) = s_\theta(x_1; x_2) \\ y_2 = & x_2 \end{cases}$



Summary

References

Dinh, L., Krueger, D., and Bengio, Y. (2014).
Nice: Non-linear independent components estimation.
arXiv preprint arXiv:1410.8516.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016).
Improved variational inference with inverse autoregressive flow.
Advances in neural information processing systems, 29.

Kobyzev, I., Prince, S. J., and Brubaker, M. A. (2020).
Normalizing flows: An introduction and review of current methods.
IEEE transactions on pattern analysis and machine intelligence, 43(11):3964–3979.

Thank you!

David Huk and Rigers Behluli