[Paper review 48]

Normalizing Flows for Probabilistic Modeling and Inference

(Papamakarios, et al., 2019)

[Contents]

1. Abstract

NF : provide expressive distn, require 2 things

- (1) base distn
- (2) series of bijective transformation

Provide a perspective by describing flows through the lens of **probabilistic modeling and inference**

2. Introduction

How is the data generated (produced) ?

Build probability distn as NF!

- Section 2) formal & conceptual structure of NF
- Section 3) in detail for finite & infinitesimal variants
- Section 4) general perspective
- Section 5) extensions to structured domains & geometries
- Section 6) oft-encountered applications

3. Normalizing Flow

3-1. Definition & Basics

$\mathbf{x} = T(\mathbf{u}) \text{ where } \mathbf{u} \sim p_{\mathbf{u}}(\mathbf{u})$

- $p_{\mathbf{u}}(\mathbf{u})$: base distn
- params
  - ϕ : for transformation T
  - θ : for base distn $p_{\mathbf{u}}(\mathbf{u})$

- T : must be **invertible** & **differentiable**

  T^{-1} : must be **differentiable**

  $\rightarrow$ such T are called ***diffeomorphisms***

Change of variables

- $p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{u}}(\mathbf{u})|\det J_T(\mathbf{u})|^{-1} \quad \text{where} \quad \mathbf{u} = T^{-1}(\mathbf{x}).$
  (= $p_{\mathrm{x}}(\mathrm{x}) = p_{\mathrm{u}}\left(T^{-1}(\mathrm{x})\right)\left|\det J_{T^{-1}}(\mathrm{x})\right|$)

Jacobian $J_T(\mathbf{u})$

- is the $D \times D$ matrix of all partial derivatives of T

$$J_T(\mathbf{u}) = \begin{bmatrix} \frac{\partial T_1}{\partial \mathrm{u}_1} & \cdots & \frac{\partial T_1}{\partial \mathrm{u}_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_D}{\partial \mathrm{u}_1} & \cdots & \frac{\partial T_D}{\partial \mathrm{u}_D} \end{bmatrix}.$$

Typically, T : NN & $p_u(\mathbf{u})$: MVN

Absolute Jacobian determinant : $|\det J_T(\mathbf{u})|$

- quantifies the relative change of volume of a small neighbourhood around $\mathbf{u}$ due to T.

Property of "invertible" + "differentiable" transformation = **composable**

$$\begin{aligned} (T_2 \circ T_1)^{-1} &= T_1^{-1} \circ T_2^{-1} \\ \det J_{T_2 \circ T_1}(\mathbf{u}) &= \det J_{T_2}\left(T_1(\mathbf{u})\right) \cdot \det J_{T_1}(\mathbf{u}). \end{aligned}$$

"Normalizing" "Flow"

- (1) Flow : trajectory, that a collection of samples from $p_{\mathrm{u}}(\mathbf{u})$ follow

  by the sequence of transformations $T_1, \ldots, T_K$
- (2) Normalizing : inverse flow through $T_K^{-1}, \ldots, T_1^{-1}$ takes a collection of samples from $p_{\mathrm{x}}(\mathrm{x})$

  transforms it back (=normalize them) into a collection of samples from $p_{\mathrm{u}}(\mathbf{u})$

Flow-based model provides 2 operations

- **(1) SAMPLING from the model** (forward transformation)

  $\rightarrow \mathbf{x} = T(\mathbf{u}) \text{ where } \mathbf{u} \sim p_{\mathbf{u}}(\mathbf{u})$
- **(2) EVALUATING the model's density** (Inverse transformation & Jacobian determinant)

  $\rightarrow p_{\mathrm{x}}(\mathrm{x}) = p_{\mathrm{u}}\left(T^{-1}(\mathrm{x})\right)\left|\det J_{T^{-1}}(\mathrm{x})\right|$

3-2. Expressive power of flow-based models

How expressive is it?

$\rightarrow$ **universal representation is possible, under reasonable conditions on** $p_x(\mathbf{x})$

3-3. Using flows for modeling & inference

how to fit **flow-based model** $p_{\mathbf{x}}(\mathbf{x};\boldsymbol{\theta})$ to a **target distn** $p^*_{\mathrm{x}}(\mathrm{x})$?

- by minimizing some divergence/discrepancy (ex. KL-div)
- params : $\theta = \{\phi, \psi\}$

  where ϕ are the parameters of T and ψ are the parameters of $p_{\mathrm{u}}(\mathbf{u})$

3-3-1. Forward KL & MLE

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= D_{\mathrm{KL}}\left[p^*_{\mathrm{x}}(\mathrm{x}) \| p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta})\right] \\ &= -\mathbb{E}_{p^*_{\mathrm{x}}(\mathrm{x})}\left[\log p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta})\right] + \text{const.} \\ &= -\mathbb{E}_{p^*_{\mathrm{x}}(\mathrm{x})}\left[\log p_{\mathrm{u}}\left(T^{-1}(\mathrm{x};\phi);\psi\right) + \log\left|\det J_{T^{-1}}(\mathrm{x};\phi)\right|\right] + \text{const.}\end{aligned}$$

- suitable when we have **samples from the target distn**
- unsuitable when we cannot evaluate target density $p^*_{\mathrm{x}}(\mathrm{x})$

Using MC samples...

$$\mathcal{L}(\boldsymbol{\theta}) \approx -\frac{1}{N}\sum_{n=1}^{N} \log p_{\mathrm{u}}\left(T^{-1}\left(\mathbf{x}_n;\boldsymbol{\phi}\right);\boldsymbol{\psi}\right) + \log\left|\det J_{T^{-1}}\left(\mathbf{x}_n;\boldsymbol{\phi}\right)\right| + \text{ const.}.$$

Minimizing KL-div

= Fitting flow-based model to the samples $\{\mathbf{x}_n\}_{n=1}^N$ **by MLE**

How to optimize? **Iteratively with SGD**

$$\begin{aligned}\nabla_\phi \mathcal{L}(\boldsymbol{\theta}) &\approx -\frac{1}{N}\sum_{n=1}^{N} \nabla_\phi \log p_{\mathrm{u}}\left(T^{-1}\left(\mathbf{x}_n;\phi\right);\boldsymbol{\psi}\right) + \nabla_\phi \log\left|\det J_{T^{-1}}\left(\mathbf{x}_n;\phi\right)\right| \\ \nabla_\psi \mathcal{L}(\boldsymbol{\theta}) &\approx -\frac{1}{N}\sum_{n=1}^{N} \nabla_\psi \log p_{\mathrm{u}}\left(T^{-1}\left(\mathbf{x}_n;\phi\right);\boldsymbol{\psi}\right)\end{aligned}.$$

- $\nabla_\psi \mathcal{L}(\boldsymbol{\theta})$: can be done in closed-form if $p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})$ admits closed-form MLE

  (ex. Gaussian distn)

Fitting by **MLE**

- need to compute T^{-1}, Jacobian determinant, density $p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})$

  & different those three
- no need to compute T , or sample from $p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})$

3-3-2. Reverse KL

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= D_{\mathrm{KL}}\left[p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta})||p^*_{\mathrm{x}}(\mathrm{x})\right] \\ &= \mathbb{E}_{p_{\mathrm{x}}(\mathrm{x};\theta)}\left[\log p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta}) - \log p^*_{\mathrm{x}}(\mathrm{x})\right] \\ &= \mathbb{E}_{p_{\mathrm{u}}(\mathrm{u};\psi)}\left[\log p_{\mathrm{u}}(\mathbf{u};\psi) - \log|\det J_T(\mathbf{u};\phi)| - \log p^*_{\mathrm{x}}(T(\mathbf{u};\phi))\right]\end{aligned} .$$

- suitable when we can evaluate target density $p^*_{\mathrm{x}}(\mathrm{x})$
- no need to **samples from the target distn**

Rewrite reverse KL

- $p^*_{\mathrm{x}}(\mathrm{x}) = \tilde{p}_{\mathrm{x}}(\mathrm{x})/C$ (where $\tilde{p}_{\mathrm{x}}(\mathrm{x})$ is unnormalized density)
- $\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathrm{u}}(\mathrm{u};\psi)}\left[\log p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi}) - \log|\det J_T(\mathbf{u};\boldsymbol{\phi})| - \log\tilde{p}_{\mathrm{x}}(T(\mathbf{u};\boldsymbol{\phi}))\right] + \text{ const.}$

Minimize reverse KL iteratively with SGD & Reparam Trick & MC estmation

- $\{\mathbf{u}_n\}_{n=1}^N$: samples from $p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})$
- minimize w.r.t ϕ :

  $\nabla_\phi \mathcal{L}(\boldsymbol{\theta}) \approx -\frac{1}{N}\sum_{n=1}^N \nabla_\phi \log|\det J_T\left(\mathbf{u}_n;\phi\right)| + \nabla_\phi \log\tilde{p}_{\mathrm{x}}\left(T\left(\mathbf{u}_n;\phi\right)\right).$
- minimize w.r.t ψ :

  use reparameterizion ... $\mathbf{u} = T'\left(\mathbf{u}';\boldsymbol{\psi}\right) \quad \text{where} \quad \mathbf{u}' \sim p_{\mathrm{u}'}\left(\mathbf{u}'\right)$

  but we can absorb T' into T & replace the base distn with $p_{\mathrm{u}'}\left(\mathbf{u}'\right)$

3-3-3. Duality between Forward & Reverse KL

Think of

- target $p^*_{\mathrm{x}}(\mathrm{x})$ as "base distribution"
- inverse flow as "inducing a distn $p^*_{\mathrm{u}}(\mathbf{u};\phi)$"

$p^*_{\mathrm{u}}(\mathbf{u};\phi) = p_{\mathrm{u}}(\mathbf{u};\psi)$ if and only if $p^*_{\mathrm{x}}(\mathrm{x}) = p_{\mathrm{x}}(\mathrm{x};\theta)$.... Thus (a) & (b) are equivalent!

- (a) fitting model $p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta})$ to target $p^*_{\mathrm{x}}(\mathrm{x})$
- (b) fitting induced distn $p^*_{\mathrm{u}}(\mathbf{u};\phi)$ to base $p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})$

Using change of variables... $D_{\mathrm{KL}}\left[p^*_{\mathrm{x}}(\mathrm{x})||p_{\mathrm{x}}(\mathrm{x};\theta)\right] = D_{\mathrm{KL}}\left[p^*_{\mathrm{u}}(\mathbf{u};\phi)||p_{\mathrm{u}}(\mathbf{u};\psi)\right]$.

$\rightarrow$ "fitting the model to the target using the **forward KL divergence**
= fitting the induced distribution $p^*_{\mathrm{u}}(\mathbf{u};\phi)$ to the base $p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})$ under the **reverse KL divergence**."

3-3-4. Alternative Divergences

not restricted to KL-divergence

- 1) f-divergence : use **density ratios** to compare models
- 2) Integral Probability Metrics (IPMs) : use **differences** for comparison

f -divergence	$D_f\left[p_{\mathrm{x}}^*(\mathrm{x}) \| p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta})\right]=\mathbb{E}_{p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta})}\left[f\left(\frac{p_{\mathrm{x}}^*(\mathrm{x})}{p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta})}\right)\right]$
IPM	$\delta_s\left[p_{\mathrm{x}}^*(\mathrm{x}) \| p_{\mathrm{x}}(\mathrm{x};\boldsymbol{\theta})\right]=\mathbb{E}_{p_{\mathrm{x}}^*(\mathrm{x})}[\boldsymbol{s}(\mathrm{x})]-\mathbb{E}_{p_{\mathrm{x}}(\mathrm{x};\theta)}[\boldsymbol{s}(\mathrm{x})]$

4. Constructing Flows Part 1 : Finitie Compositions

NF are **composable** : $T=T_K \circ \cdots \circ T_1$

- idea : use simple transformation as building blocks

Evaluation

- forward evaluation : $\mathbf{z}_k=T_k\left(\mathbf{z}_{k-1}\right) \quad$ for $k=1:K$
- inverse evaluation : $\mathbf{z}_{k-1}=T_k^{-1}\left(\mathbf{z}_k\right) \quad$ for $k=K:1$

Jacobian-determinant (in log domain) : $\log\left|J_T(\mathbf{z})\right|=\log\left|\prod_{k=1}^K J_{T_k}\left(\mathbf{z}_{k-1}\right)\right|=\sum_{k=1}^K \log\left|J_{T_k}\left(\mathbf{z}_{k-1}\right)\right|$

.

- increase depth $\rightarrow$ only $O(K)$ growth in computational complexity

Implement T_k or T_k^{-1} using NN with param ϕ_k = f_{ϕ_k}

(ensure network is invertible & tractable Jacobian determinant)

Ensuring f_{ϕ_k} is invertible $\neq$ Explicitly calculating its inverse

Tractable Jacobian determinant

- we can always find it ..... but cost $O(D^3)$ (with D inputs & D outputs)
  $\rightarrow$ intractable for large D
- should be at most $O(D)$
  will describe NN design that allow Jacobian determinant to be computed in linear time

4-1. Autoregressive Flow

under certain conditions, we can transform **any distn** $p_x(\mathbf{x})$ into a **uniform distn** using maps with **triangular Jacobian**

Autoregressive Flow

$z_i'=\tau\left(z_i ; h_i\right)$ where $h_i=c_i\left(\mathbf{z}_{<i}\right)$.

- τ : transformer

    - strictly monotonic function of z_i (thus invertible)
    - parameterized by h_i
    - specifies how the flow acts on z_i to output z_i'
- c_i : i-th conditioner
    - determines the parameters of the transformer
    - DOES NOT need to be a bijection

The above is **invertible** for any choice of τ and c_i, as long as "transformer is invertible"

$\mathrm{z}_i = \tau^{-1}\left(\mathrm{z}_i'; \boldsymbol{h}_i\right) \quad \text{where} \quad \boldsymbol{h}_i = c_i\left(\mathbf{z}_{<i}\right).$

- (forward : $z_1 \rightarrow z_K$) can be done in parallel
- (inverse : $z_k \rightarrow z_1$) all $\mathbf{z}_{<i}$ need to be computed before $\mathbf{z_i}$

Triangular Jacobian $\rightarrow$ tractable ($\mathcal{O}(D)$ time)

$$J_{f_\phi}(\mathbf{z}) = \begin{bmatrix} \frac{\partial \tau}{\partial z_1}(\mathrm{z}_1; \boldsymbol{h}_1) & & 0 \\ & \ddots & \\ \mathbf{L}(\mathbf{z}) & & \frac{\partial \tau}{\partial \mathbf{z}_D}(\mathrm{z}_D; \boldsymbol{h}_D) \end{bmatrix}.$$

$\log\left|\det J_{f_\phi}(\mathbf{z})\right| = \log\left|\prod_{i=1}^{D} \frac{\partial \tau}{\partial \mathrm{z}_i}(\mathrm{z}_i; \boldsymbol{h}_i)\right| = \sum_{i=1}^{D} \log \mid \frac{\partial \tau}{\partial \mathrm{z}_i}(\mathrm{z}_i; \boldsymbol{h}_i)$

Autoregressive flows are **Universal approximators** , provided that the transformer & conditioner are flexible enough!

Alternative : conditioner c_i take in $\mathbf{z}'_{<i}$ instead of $\mathbf{z}_{<i}$

(mathematically equivalent)

4-1-1. Implementing the Transformer

What to choose as a transformer?

(a) Affine autoregressive flows

(= **location-scale** transformation)

flow : $\tau\left(\mathrm{z}_i; \boldsymbol{h}_i\right) = \alpha_i \mathrm{z}_i + \beta_i \quad \text{where} \quad \boldsymbol{h}_i = \{\alpha_i, \beta_i\}.$

Jacobian : $\log\left|\det J_{f_\phi}(\mathbf{z})\right| = \sum_{i=1}^{D} \log|\alpha_i| = \sum_{i=1}^{D} \tilde{\alpha}_i.$

Pros & Cons

- Pros) simplicity & analytical tractability
- Cons) expressivity is limited

(ex. let $\mathbf{z}$: Gaussian $\rightarrow$ output is also Gaussian ....Thus need to stack multiple affine AF layers)

Widely used

- NICE, Real NVP, IAF, MAF, Glow

(b) Non-Affine neural transformers

conic combinations & **compositions** of monotonic functions are also monotonic

- Conic combination: $\tau(\mathrm{z}) = \sum_{k=1}^{K} w_k \tau_k(\mathrm{z})$, where $w_k > 0$ for all k.
- Composition: $\tau(\mathrm{z}) = \tau_K \circ \cdots \circ \tau_1(\mathrm{z})$

Non-Affine neural transformers can be constructed...

"using **conic combination of monotonically increasing activation functon** $\sigma(\cdot)$"

$\tau\left(\mathrm{z}_i; \boldsymbol{h}_i\right) = w_{i0} + \sum_{k=1}^{K} w_{ik} \sigma\left(\alpha_{ik} \mathrm{z}_i + \beta_{ik}\right) \quad \text{where} \quad \boldsymbol{h}_i = \{w_{i0}, \ldots, w_{iK}, \alpha_{ik}, \beta_{ik}\}.$

Jacobian determinant

- analytically obtainable (but more commonly computed by **back-propagation**)

Drawback

- (in general) **can not be inverted analytically**
  (can be inverted only iteratively... e.g. **bijection search**)

Variants : **NAF, B-NAF, Flow++**

(c) Integration-based transformers

The integral of some positive function = monotonically increasing function

ex) $\tau\left(\mathrm{z}_i; \boldsymbol{h}_i\right) = \int_0^{\mathrm{z}_i} g\left(\mathrm{z}; \boldsymbol{\alpha}_i\right) d\mathrm{z} + \beta_i \quad \text{where} \quad \boldsymbol{h}_i = \{\boldsymbol{\alpha}_i, \beta_i\}.$

- $g\left(\cdot; \boldsymbol{\alpha}_i\right)$: any positive-valued NN
- derivative of transformer = $g\left(\mathbf{z_i}; \boldsymbol{\alpha}_i\right)$
- $g\left(\cdot; \boldsymbol{\alpha}_i\right)$ to be a positive polynomial of degree $2L$ (then integral will be $2L+1$ in $\mathbf{z_i}$)
  $\rightarrow$ **Sum-of-squares polynomial transformer**
  $\tau\left(\mathrm{z}_i; \boldsymbol{h}_i\right) = \int_0^{\mathrm{z}_i} \sum_{k=1}^{K} \left(\sum_{\ell=0}^{L} \alpha_{ik\ell} \mathrm{z}^{\ell}\right)^2 d\mathrm{z} + \beta_i.$

Affine transformer : $L = 0$ of above

$\int_0^{z_i} \sum_{k=1}^K \left(\alpha_{ik0} z^0\right)^2 dz + \beta_i = \left(\sum_{k=1}^K \alpha_{ik0}^2\right) z \Big|_0^{z_i} + \beta_i = \alpha_i z_i + \beta_i.$

(for large enough L) can approximate arbitrarily well any monotonic increasing function

(d) Neural spline flows

Non-affine transformers...don't have analytic inverse

But since all transformers are monotonic, can be inverted by **bijections search**

- $\tau\left(\mathrm{z}_{iA}; \boldsymbol{h}_i\right) < \mathrm{z}_i' < \tau\left(\mathrm{z}_{iB}; \boldsymbol{h}_i\right)$.
- keep halving, such that solution $\mathbf{z_i}$ is always contained
- Bijection search computes $\mathbf{z_i}$ with accuracy ϵ in $\mathcal{O}\left(\log \frac{1}{\epsilon}\right)$

  $\rightarrow$ trade-off btw accuracy & computation

Overcome this trade-off with **monotonic spline**

(= piecewise function consisting of K segments , which are easy to invert)

- given $K+1$ input locations $\mathbf{z}_{i0}, \ldots, \mathbf{z}_{iK}$
- transformer $\tau\left(\mathrm{z}_i; \boldsymbol{h}_i\right)$: simple monotonic function in each interval $\left[\mathrm{z}_{i(k-1)}, \mathrm{z}_{ik}\right]$
- parameters $\boldsymbol{h}_i$:
  - input locations $z_{i0}, \ldots, z_{iK}$
  - corresponding output locations $z_{i0}', \ldots, z_{iK}'$
  - derivatives (i.e. slopes) at $\mathrm{z}_{i0}, \ldots, \mathrm{z}_{iK}$

Spline-based transformers

- distinguished by the type of spline they use
- fast to invert as to evaluate, while maintaining exact analytical invertibility
- Evaluating or inverting
  - (step 1) locate right segment ($\mathcal{O}(\log K)$ using binary search)
  - (step 2) evaluate / invert that segment (= analytically tractable)

4-1-2. Implementing the Conditioner

Conditioner : $c_i\left(\mathbf{z}_{<i}\right)$

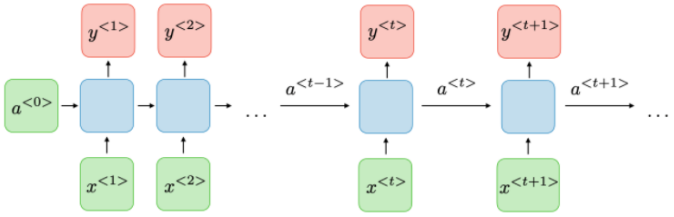
- can be any function (NN ok)
- each $c_i\left(\mathbf{z}_{<i}\right)$ having its separate NN $\rightarrow$ scale poorly with dim D

  (cost of storing, learning params of D independent networks...)
- should SHARE params! how?
  - 1) Recurrent Autoregressive Flows
  - 2) Masked Autoregressive Flows
  - 3) Coupling Layer

(a) Recurrent Autoregressive Flows

conditioner = RNN

$$\boldsymbol{h}_i = c\left(\boldsymbol{s}_i\right) \text{ where } \begin{array}{l} s_1 = \text{ initial state} \\ s_i = \mathrm{RNN}(\mathrm{z}_{i-1}, s_{i-1}) \text{ for } i > 1 \end{array}.$$



share params across conditional distributions of **autoregressive models**!

Downside?

- parallel computation into **sequential** one
- involves $O(D)$ steps...slow for high-dim data

(b) Maksed Autoregressive Flows

"Share params + avoid sequential computation of RNN by **MAKSING** "

by Masking...

- single feedforward! (takes in $\mathbf{z}$, outputs the entire sequence $\left(\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_D}\right)$)
- only requirement : **autoregressive structure**

  (output $\mathbf{h_i}$ can not depend on inputs $\mathbf{z}_{\geq i}$)
- remove connections from input z_i to outputs $\left(\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_i}\right)$

  (by multiplying each weight matrix elementwise, with binary matrix of same size)
- will have same architecture as original NN
- evaluate efficiently using GPU
- ex) **MADE** (Masked Autoencoder for Distribution Estimation), CNN, Self-attention...

MADE (Masked Autoencoder for Distribution Estimation)

- assign a "degree" between 1~D for each input/hidden/output node
- mask-out the weights between subsequent layers

2 main advantages

- 1) Efficient to evaluate (one NN pass & can be computed parallel)
- 2) Universal approximators

Disadvantage

- Not as efficient to **invert** as to evaluate

  ($\because$ params $\mathbf{h_i}$ are needed to obtain $\tau^{-1}\left(\mathbf{z}_i^{\prime} ; \boldsymbol{h}_i\right)$, thus can not be computed until $\left(z_i, \ldots z_{i-1}\right)$ is obtained)

Pseudocode

$$
\begin{array}{l}
\text{Initialize } z \text{ to an arbitrary value} \\
\text{for } i=1: D \\
\left(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_D\right)=c(\mathbf{z}) \\
\mathbf{z}_i=\tau^{-1}\left(\mathbf{z}_i^{\prime} ; \boldsymbol{h}_i\right)
\end{array}
$$

- inverse : D times more expansive than evaluating forward transformation

Examples of masking to implementing **autoregressive flows**

- IAF, MAF, NAF, B-NAF, MintNet, MaCow

Examples of masking to implementing **non-flow based autoregressive models**

- MADE, PixelCNN, WaveNet

(c) Coupling Layers

Masked Autoregressive Flows : computational asymmetry

(either (1) sampling or (2) density evaluation will be D times slower .... IAF vs MAF)

$\rightarrow$ for both to be fast, different conditioner is needed! ... **COUPLING LAYER**

Coupling layer

- choose index d (commonly, $D/2$)
- design the conditioner such that
  - params $\left(\mathbf{h_1}, \ldots . \mathbf{h_d}\right)$ are constnats
  - params $\left(\mathbf{h_{d+1}}, \ldots . \mathbf{h_D}\right)$ are functions of $\mathbf{z}_{<\mathbf{d}}$ (ex. NN)
- (coupling layer) splits into 2 parts

  (fully autoregressive flow) splits into D parts

  (intermediate) split into K parts

  $\rightarrow$ inverting the transformation will be $O(K)$ times more expensive than evaluating
- [forward]

$$
\begin{aligned}
\mathbf{z}_{\leq d}^{\prime} &= \mathbf{z} \leq d \\
\left(\boldsymbol{h}_{d+1}, \ldots, \boldsymbol{h}_D\right) &= \mathrm{NN}\left(\mathbf{z}_{\leq d}\right) \\
\mathbf{z}_i^{\prime} &= \tau\left(\mathbf{z}_i ; \boldsymbol{h}_i\right) \text{ for } i>d .
\end{aligned}
$$

- [inverse]

$$
\begin{aligned}
\mathbf{z}_{\leq d} &= \mathbf{z}_{\leq d}^{\prime} \\
\left(\boldsymbol{h}_{d+1}, \ldots, \boldsymbol{h}_D\right) &= \mathrm{NN}\left(\mathbf{z}_{\leq d}\right) \\
\mathbf{z}_i &= \tau^{-1}\left(\mathbf{z}_i^{\prime} ; \boldsymbol{h}_i\right) \text{ for } i>d .
\end{aligned}
$$

- Jacobian :

$J_{f_\phi} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{A} & \mathbf{D} \end{bmatrix}$

  - I : $d \times d$ matrix
  - $\mathbf{0}$: $d \times (D-d)$ matrix
  - $\mathbf{A}$: $(D-d) \times d$ matrix
  - $\mathbf{D}$: $(D-d) \times (D-d)$ matrix
  - thus, jacobian : product of diagonals of $\mathbf{D}$ = $\tau\left(\cdot ; h_{d+1}\right), \ldots, \tau\left(\cdot ; h_D\right)$
- but due to efficiency...reduced expressive power

  single coupling layer $\neq$ universal approximator

  $\rightarrow$ need to compose multiple composing layers!

  (when composing, elements of $\mathbf{z}$ need to be permuted! so that all dim have chance to be transformed)
- D coupling layers is an universal approximator

  (set index d of the i-th coupling layer = $i-1$)
- Summary
  - coupling layers = most popular method for flow-based models

    (∵ allow both density evaluation & sampling to be done in single NN pass)
  - widely used in generative models of high-dim data

    (NICE, Real NVP, Glow, WaveGlow, FloWaveNet, Flow++)

4-1-3. Relationship with Autoregressive Models

Alongside NF, **autoregressive models** are another popular model for high-dim distribution

$p_{\mathrm{x}}(\mathrm{x}) = \prod_{i=1}^{D} p_{\mathrm{x}}\left(\mathrm{x}_i \mid \mathrm{x}_{<i}\right)$.

- model each conditional as $p_{\mathrm{x}}\left(\mathrm{x}_i \mid \mathrm{x}_{<i}\right) = p_{\mathrm{x}}\left(\mathrm{x}_i ; \boldsymbol{h}_i\right), \quad \text{where} \quad \boldsymbol{h}_i = c_i\left(\mathrm{x}_{<i}\right)$
- ex) $p_{\mathrm{x}}\left(\mathrm{x}_i ; \boldsymbol{h}_i\right)$: Gaussian, parameterized by its mean & var
- $c_i\left(\mathrm{x}_{<i}\right)$: analogous to **conditioners** of autoregresssive flow

  (= typically implemented with NN)
- can also be used for discrete/mixed data

Autoregressive models of continuous variables = **Autoregressive flows** with single layer

let $\tau\left(\mathrm{x}_i ; \boldsymbol{h}_i\right)$ = cumulative distribution of $p_{\mathrm{x}}\left(\mathrm{x}_i ; \boldsymbol{h}_i\right)$

$\rightarrow \tau\left(\mathrm{x}_i ; \boldsymbol{h}_i\right) = \int_{-\infty}^{\mathrm{x}_i} p_{\mathrm{x}}\left(\mathrm{x}_i^{\prime} ; \boldsymbol{h}_i\right) d \mathrm{x}_i^{\prime}$

and the vector $\mathbf{u} = \left(\mathrm{u}_1, \ldots, \mathrm{u}_D\right)$

$\rightarrow \mathrm{u}_i = \tau\left(\mathrm{x}_i ; \boldsymbol{h}_i\right) \quad \text{where} \quad \boldsymbol{h}_i = c_i\left(\mathrm{x}_{<i}\right)$

is always distributed uniformly in $(0,1)^D$.

"same as autoregressive flow" with $\mathbf{z} = \mathbf{x}$ and $\mathbf{z}^{\prime} = \mathbf{u}$.

Log probability :

$$\log p_{\mathrm{x}}(\mathrm{x}) = \log \prod_{i=1}^{D} \mathrm{Uniform}(\tau\left(\mathrm{x}_i; \boldsymbol{h}_i\right); 0, 1) + \log \prod_{i=1}^{D} p_{\mathrm{x}}\left(\mathrm{x}_i; \boldsymbol{h}_i\right) = \sum_{i=1}^{D} \log p_{\mathrm{X}}\left(\mathrm{x}_i \mid \mathrm{x}_{<i}\right).$$

IAF : $\mathrm{z}_i = \tau^{-1}\left(\mathrm{u}_i; \boldsymbol{h}_i\right) \quad \text{where} \quad \boldsymbol{h}_i = c_i\left(\mathbf{x}_{<i}\right)$

- corresponds exactly to sampling from autoregressive model (inverse transform sampling)

Transformer : not limited to being the inverse CDF

ex) $p_{\mathrm{x}}\left(\mathrm{x}_i; \boldsymbol{h}_i\right) = \mathcal{N}\left(\mathrm{x}_i; \mu_i, \sigma_i^2\right) \quad \text{where} \quad \boldsymbol{h}_i = \{\mu_i, \sigma_i\}$

- reparam : $\mathrm{x}_i = \sigma_i \mathrm{u}_i + \mu_i \quad \text{where} \quad \mathrm{u}_i \sim \mathcal{N}(0, 1)$
- thus, entire autoregressive model = affine autoregressive flow

Conclusion

- 1) AF = extnending AM for continuous variables
- 2) Benefits of viewing AM as flows?
  - (a) provides a framework for their composition $\rightarrow$ enhance flexibility
  - (b) gives us freedom in specifying the base distn (base distn can be learend)
  - (c) compose AM with other types of flows (ex. non-autoregressive flows)

4-2. Linear Flows

Autoregressive flows : restrict $z_i^{'}$ to depend only on inputs $\mathbf{z}_{\neq i}$

- dependent on the **order of input var**
- in the limit of capacity, no limit in flexibility, but in practice , it isn't
- So use **permutation**!

Permutation

- (1) easily invertible transformation

  (2) absolute Jacobian determinant = 1
- general idea of permutation = "Linear Flow"

  $\mathbf{z}' = \mathbf{W}\mathbf{z}$
  - where $\mathbf{W}$ is a $D \times D$ invertible matrix that parameterizes the transformation
  - Jacobian determinant = det $\mathbf{W}$
- special case of linear flow, where $\mathbf{W}$ is a **permutation matrix**
- Commonly... alternate "invertible linear transformation" & "autoregressive/coupling layers"

Parameterize and learn matrix $\mathbf{W}$?

- problem 1) $\mathbf{W}$ is not guaranteed to be invertible

  problem 2) solving $\mathbf{W}z = z'$ takes $O(D^3)$

problem 3) det $\mathbf{W}$ takes $O(D^3)$ in general

- goal 1) $\mathbf{W}$ that is invertible (& inversion cost $O(D^2)$)

  goal 2) computing its determinant costs $O(D)$

4-3. Residual Flows

invertible transformation of the from : $\mathbf{z}' = \mathbf{z} + g_\phi(\mathbf{z})$

- g_ϕ : NN that outputs D dim translation vector
- ϕ : NN parameter

2 general approaches

- 1) contractive maps
- 2) matrix determinant lemma

4-3-1. Contractive RF

RF is not always invertible, but invertible when g_ϕ **can be made contractive w.r.t some distance function**

- $\delta\left(F\left(\mathbf{z}_A\right), F\left(\mathbf{z}_B\right)\right) \leq L \delta\left(\mathbf{z}_A, \mathbf{z}_B\right)$.
- meaning) contractive map brings any 2 inputs close together, by at least a factor L

Banach fixed-point theorem

- contractive map has exactly one fixed point $\mathbf{z}_* = F\left(\mathbf{z}_*\right)$
- $\mathbf{z}_{k+1} = F\left(\mathbf{z}_k\right)$

Residual transformation : $\mathbf{z}' = f_\phi(\mathbf{z}) = \mathbf{z} + g_\phi(\mathbf{z})$

- $F(\hat{\mathbf{z}}) = \mathbf{z}' - g_\phi(\hat{\mathbf{z}})$
- If g_ϕ is contractive with Lipschitz constant L, then F is also ~.
- by *Banach fixed-point theorem*,
  - there exists a unique $\mathbf{z}_*$ such that $\mathbf{z}_* = \mathbf{z}' - g_\phi\left(\mathbf{z}_*\right)$
  - starting from arbitrary input $\mathbf{z}_0$, iteratively apply F as follows:

    $\mathbf{z}_{k+1} = \mathbf{z}' - g_\phi\left(\mathbf{z}_k\right) \quad \text{for } k \geq 0$
  - guarantees that the procedure above converges to $\mathbf{z}_* = f_\phi^{-1}\left(\mathbf{z}'\right)$
  - rate of convergence : $\delta\left(\mathbf{z}_k, \mathbf{z}_*\right) \leq \frac{L^k}{1-L} \delta\left(\mathbf{z}_0, \mathbf{z}_1\right)$

    smaller L, faster $\mathbf{z_k}$ converges to $\mathbf{z}_*$

    (trading off flexibility for efficiency)

Challenge of **contractive RF**

how to make contractive, without impinging upon its flexibility?

- composition of K Lipschitz-cont func $F_1, \dots F_K$ = Lipschitz cont with constant equal to $\prod_{k=1}^{K} L_k$
  - where L_k is the Lipschitz constant of F_k
  - make each layer Lipschtiz continuous with $L_k \leq 1$ & at leas one $L_k < 1$
  - ex) sigmoid, tanh, ReLU : Lipschitz continuous with a constant ≤ 1
- linear layer can be made **contractive**, by dividing them with operator norm
  - ex) Spectral normalization : use euclidean norm

Drawback of **contractive RF**

- no known general, efficient procedure for computing Jacobian .... costs $O(D^3)$
- nonetheless, able to obtain unbiased estimate of log absolute Jacobian

  (by power series)

  $\log\left|\det J_{f_\phi}(\mathbf{z})\right| = \log\left|\det\left(\mathbf{I} + J_{g_\phi}(\mathbf{z})\right)\right| = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \operatorname{Tr}\left\{J_{g_\phi}^k(\mathbf{z})\right\}$
  - $J_{g_\phi}^k(\mathbf{z})$: k -th power of the Jacobian of g_ϕ evaluated at $\mathbf{z}$

  (by Hutchinson trace estimator) efficiently estimate Trace

  $\operatorname{Tr}\left\{J_{g_\phi}^k(\mathbf{z})\right\} \approx \mathbf{v}^\top J_{g_\phi}^k(\mathbf{z}) \mathbf{v}$
  - $\mathbf{v}$: D-dim random vector with zero mean & unit cov
  - $\mathbf{v}^\top J_{g_\phi}^k(\mathbf{z})$ can be computed with k backprop

  (by Russian-roulette estimator)

  infinite sum can be estimated as finite sum of re-weighted terms

contractive RF (vs AF)

- dense Jacobian ... allows all input variables to affect all output variables

  $\rightarrow$ thus can be very flexible
- but exact density evaluation is computationally expensive &

  sampling is done iteratively

4-3-2. RF based on the matrix determinant lemma

Matrix determinant lemma (= MD lemma)

$\det\left(\mathbf{A} + \mathbf{V}\mathbf{W}^\top\right) = \det\left(\mathbf{I} + \mathbf{W}^\top \mathbf{A}^{-1} \mathbf{V}\right) \det \mathbf{A}$

- left) $\mathcal{O}\left(D^3 + D^2 M\right)$
- right) $\mathcal{O}\left(M^3 + D M^2\right)$

(a) Planar Flow

g_ϕ is a one-layer NN with a single hidden unit:

$\mathbf{z}' = \mathbf{z} + \mathbf{v}\sigma\left(\mathbf{w}^\top \mathbf{z} + b\right)$

- $\mathbf{v} \in \mathbb{R}^D, \mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$, and σ is a differentiable activation function

Jacobian :

- $J_{f_\phi}(\mathbf{z}) = \mathbf{I} + \sigma'\left(\mathbf{w}^\top \mathbf{z} + b\right)\mathbf{v}\mathbf{w}^\top$.
- (using MD lemma) $\det J_{f_\phi}(\mathbf{z}) = 1 + \sigma'\left(\mathbf{w}^\top \mathbf{z} + b\right)\mathbf{w}^\top \mathbf{v}$

Conditions of Planar Flow to be invertible

- 1) σ' : positive & bounded above
- 2) $\mathbf{w}^\top \mathbf{v} > -\frac{1}{\sup_x \sigma'(x)}$

  $\rightarrow$ ensures $\det J_{f_\phi}(\mathbf{z})$ is always non-zero

(b) Sylvester Flow

extend Planar Flow to M hidden units

$\mathbf{z}' = \mathbf{z} + \mathbf{V}\sigma\left(\mathbf{W}^\top \mathbf{z} + \mathbf{b}\right)$.

- $\mathbf{V} \in \mathbb{R}^{D\times M}, \mathbf{W} \in \mathbb{R}^{D\times M}$ and $\mathbf{b} \in \mathbb{R}^M$, and activation function σ to be elementwise

Jacobian

- $J_{f_\phi}(\mathbf{z}) = \mathbf{I} + \mathbf{V}\mathbf{S}(\mathbf{z})\mathbf{W}^\top$
- (by MD lemma) $\det J_{f_\phi}(\mathbf{z}) = \det\left(\mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^\top \mathbf{V}\right)$

QR decomposition

- $\mathbf{V} = \mathbf{Q}\mathbf{U}$ and $\mathbf{W} = \mathbf{Q}\mathbf{L}$
  - $\mathbf{Q}$: $D \times M$ matrix....whose columns are orthonormal ($\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$)
  - $\mathbf{U}$: $M \times M$ upper triangle
  - $\mathbf{L}$: $M \times M$ lower triangle
- Thus, Jacobian determinant :

  $\det J_{f_\phi}(\mathbf{z}) = \det\left(\mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{L}^\top \mathbf{U}\right) = \prod_{i=1}^{D}\left(1 + S_{ii}(\mathbf{z})L_{ii}U_{ii}\right)$

(c) Radial flow

$\mathbf{z}' = \mathbf{z} + \frac{\beta}{\alpha + r(\mathbf{z})}(\mathbf{z} - \mathbf{z}_0) \quad \text{where} \quad r(\mathbf{z}) = \|\mathbf{z} - \mathbf{z}_0\|$

- $\alpha \in (0, +\infty), \beta \in \mathbb{R}$ and $\mathbf{z}_0 \in \mathbb{R}^D$, and $\|\cdot\|$ is the norm
- (meaning) contraction/expansion radially with center $\mathbf{z_0}$

Jacobian :

- $J_{f_\phi}(\mathbf{z}) = \left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)\mathbf{I} - \frac{\beta}{r(\mathbf{z})(\alpha + r(\mathbf{z}))^2}(\mathbf{z} - \mathbf{z}_0)(\mathbf{z} - \mathbf{z}_0)^\top$.
- (using MD lemma) $\det J_{f_\phi}(\mathbf{z}) = \left(1 + \frac{\alpha\beta}{(\alpha + r(\mathbf{z}))^2}\right)\left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)^{D-1}$

  be computed in $\mathcal{O}(D)$

The radial flow is not invertible for all β...condition?

- "$\beta > -\alpha$ " .....ensures that $\det J_{f_\phi}(\mathbf{z})$ is always non-zero

Summary of planar, Sylvester, radial flows

- $O(D)$ Jacobian determinant
- can be made invertible by suitably restricting their params

  (but no analytical way to compute inverse)
- $\therefore$ mostly used to **approximate posterior** for VAE (not in generative model)

4-4. Practical considerations when combining transformations

compose many flows as possible?

Glow

- 320 sub-transformations & 40 GPUs
- challenging computation....

2 techniques to solve?

- 1) stabilize the optimization
- 2) ease with computational demands

(a) Batch Normalization

stabilize & improve NN

composition of 2 affine transformation

- 1) scale & translation params (by batch statistics)
- 2) free params α (scale) & β (translation)

$\mathrm{BN}(\mathbf{z}) = \alpha \odot \frac{\mathrm{z} - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon}} + \beta, \quad \mathrm{BN}^{-1}(\mathbf{z}') = \hat{\mu} + \frac{\mathrm{z}' - \beta}{\alpha} \odot \sqrt{\hat{\sigma}^2 + \epsilon}.$

easy to compute Jacobian determinant (due to acting element-wise)

$\det J_{\mathrm{BN}}(\mathbf{z}) = \prod_{i=1}^{D} \frac{\alpha_i}{\sqrt{\hat{\sigma}_i^2 + \epsilon_i}}$

Inserted between transformations! ($T_k \circ \mathrm{BN} \circ T_{k-1}$)

ex) Glow

- employs a variant called **activation normalization**
- doesn't use batch statistics $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$.
  (instead, before training, flow once & α and β are set so that batch has zero & unit var)
  (= data-dependent initialization)
  (α and β are optimized as model params)
- preferable when training with small-mini batches
  (∵ batch norm's statistics can be noisy)

(b) Multi-scale architecture

$\mathbf{x}$ and $\mathbf{u}$ should have same dim & T_k must preserve dimensionality

Multi-scale architecture :

- when going from $\mathbf{x}$ to $\mathbf{u}$, some sub-dim of $\mathbf{z}_k$ are clamped!
  (& no additional transformation)
- just like **skip-connection**
- these help optimization!
- encode more global & semantic info in the dimensions

5. Constructing Flows part 2 : Continuous-Time Transformations

(until now) DISCRETE one-step transformation

(now) CONTINUOUS time, by parameterizing **flow's infinitesimal dynamics**

$\rightarrow$ construct flow by defining **ODE** (Ordinary Differential Equation)
(describes flow's evolution over time...called "continuous time")

5-1. Definition

$\mathbf{z}_t$: flow's state at time t

- time t runs continuously from t_0 to t_1
- $\mathbf{z}_{t_0} = \mathbf{u}$ and $\mathbf{z}_{t_1} = \mathbf{x}$
- "parameterize the time derivative of $\mathbf{z_t}$" with NN g_ϕ

$\frac{d\mathbf{z}_t}{dt} = g_\phi\left(t, \mathbf{z}_t\right).$

ODE : $\frac{d\mathbf{z}_t}{dt} = g_\phi\left(t, \mathbf{z}_t\right)$

- requirement of g_ϕ : uniformly Lipschitiz continuous in $\mathbf{z_t}$ & continuous in $\mathbf{t}$

  (many NN meet those requirements)
- do not need invertibility & tractability of Jacobian determinant

to compute transformation ($\mathrm{x} = T(\mathbf{u})$)

- need to run dynamics forward in time

  (= need to do integration)
- $\mathbf{x} = \mathbf{z}_{t_1} = \mathbf{u} + \int_{t=t_0}^{t_1} g_\phi\left(t, \mathbf{z}_t\right) dt.$

inverse transform T^{-1}

- $\mathbf{u} = \mathbf{z}_{t_0} = \mathbf{x} + \int_{t=t_1}^{t_0} g_\phi\left(t, \mathbf{z}_t\right) dt = \mathbf{x} - \int_{t=t_0}^{t_1} g_\phi\left(t, \mathbf{z}_t\right) dt$
- **Unlike discrete compositions (in before sections)**
  continuous-time flows have the same computational complexity in each direction

change in log density

- $\frac{d \log p(\mathbf{z}_t)}{dt} = -\operatorname{Tr}\left\{J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right)\right\}.$
  - $\operatorname{Tr}\{\cdot\}$: trace operator
  - $J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right)$: Jacobian of $g_\phi(t,\cdot)$ at $\mathbf{z}_t$
- but trace operatoor requiers $O(D)$ back-prop!

  use "Hutchinson's trace estimator"

  $\operatorname{Tr}\left\{J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right)\right\} \approx \mathbf{v}^\top J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right) \mathbf{v}$
  - $\mathbf{v}$: any D-dim random vector with zero mean & unit covariance
  - $\mathbf{v}^\top J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right)$: computed in a single back-prop pass

    which D times more efficient!

Integrating the derivative of $\log p(\mathbf{z_t})$

- $\log p_{\mathrm{x}}(\mathrm{x}) = \log p_{\mathrm{u}}(\mathbf{u}) - \int_{t=t_0}^{t_1} \operatorname{Tr}\left\{J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right)\right\} dt$
- forward transform & log density can be done simultaneously

  $\begin{bmatrix} \mathrm{x} \\ \log p_{\mathrm{x}}(\mathrm{x}) \end{bmatrix} = \begin{bmatrix} \mathrm{u} \\ \log p_{\mathrm{u}}(\mathbf{u}) \end{bmatrix} + \int_{t=t_0}^{t_1} \begin{bmatrix} g_\phi\left(t, \mathbf{z}_t\right) \\ -\operatorname{Tr}\left\{J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right)\right\} \end{bmatrix} dt.$
- computation is not analytically feasible for general $g_\phi \rightarrow$ use numerical integration

5-2. Solving & Optimizing continuous-time flow

(1) Euler's method

(2) Adjoint method

5-2-1. Euler's method and equivalence to RF

Simplest numerical technique

$\mathbf{z}_{t+\epsilon} \approx f_\phi\left(\mathbf{z}_t\right)=\mathbf{z}_t+\epsilon g_\phi\left(t, \mathbf{z}_t\right)$

- ϕ can be optimized with gradients computed via back-prop
- looks like $\mathbf{z}'=\mathbf{z}+g_\phi(\mathbf{z})$ (= contractive residual flow)

Taylor series expansion

log absolute Jacobian determinant of f_ϕ as follows:

like $\log\left|\det J_{f_\phi}(\mathbf{z})\right|=\log\left|\det\left(\mathbf{I}+J_{g_\phi}(\mathbf{z})\right)\right|=\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \operatorname{Tr}\left\{J_{g_\phi}^k(\mathbf{z})\right\}$

$\log\left|\det J_{f_\phi}\left(\mathbf{z}_t\right)\right|=\sum_{k=1}^{\infty} \frac{(-1)^{k+1} \epsilon^k}{k} \operatorname{Tr}\left\{J_{g_\phi(t,)}^k\left(\mathbf{z}_t\right)\right\}=\epsilon \operatorname{Tr}\left\{J_{g_\phi(t,)}\left(\mathbf{z}_t\right)\right\}+\mathcal{O}\left(\epsilon^2\right).$

- $\log p\left(\mathbf{z}_{t+\epsilon}\right)=\log p\left(\mathbf{z}_t\right)-\epsilon \operatorname{Tr}\left\{J_{g_\phi(t,)}\left(\mathbf{z}_t\right)\right\}+\mathcal{O}\left(\epsilon^2\right)$
- $\frac{\log p\left(\mathbf{z}_{t+\epsilon}\right)-\log p\left(\mathbf{z}_t\right)}{\epsilon}=-\operatorname{Tr}\left\{J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right)\right\}+\mathcal{O}(\epsilon)$
- becomes $\frac{d \log p\left(\mathbf{z}_t\right)}{d t}=-\operatorname{Tr}\left\{J_{g_\phi(t,\cdot)}\left(\mathbf{z}_t\right)\right\}$ as $\epsilon \rightarrow 0$

5-2-2. The Adjoint Method

target $\mathcal{L}(\mathbf{x} ; \phi)$

gradient of $\partial \mathcal{L} / \partial \mathbf{z}_t$

$\frac{d}{d t}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{z}_t}\right)=-\left(\frac{\partial \mathcal{L}}{\partial \mathbf{z}_t}\right)^{\top} \frac{\partial g_\phi\left(t, \mathbf{z}_t\right)}{\partial \mathbf{z}_t}$

- $\mathbf{z_t}$: flow's intermediate state

gradient w.r.t ϕ

$\frac{\partial \mathcal{L}(\mathrm{x} ; \phi)}{\partial \phi}=\int_{t=t_1}^{t_0} \frac{\partial \mathcal{L}}{\partial \mathbf{z}_t} \frac{\partial g_\phi\left(t, \mathbf{z}_t\right)}{\partial \phi} d t$

6. Applications

2 primitive operations :

- (1) density calculation
- (2) sampling

application to

- 1) probabilistic
- 2) inference
- 3) supervised learning
- 4) reinforcement learning

6-1. Probabilistic Modeling

assume finite number of draws $\mathbf{x}$ from unknown generative process : $p_X^*(\mathbf{x})$

- goal) make a good approximation to $p_{\mathrm{x}}^*(\mathbf{x})$
- popular method : MLE (use forward KL div)

$$
\begin{aligned}
D_{\mathrm{KL}}\left[p_{\mathrm{x}}^*(\mathrm{x}) \| p_{\mathrm{x}}(\mathrm{x} ; \theta)\right] &= -\mathbb{E}_{p_{\mathrm{x}}^*(\mathrm{x})}\left[\log p_{\mathrm{x}}(\mathrm{x} ; \theta)\right]+\text{const} \\
&\approx -\frac{1}{N} \sum_{n=1}^N \log p_{\mathrm{x}}\left(\mathrm{x}_n ; \theta\right)+\text{const} \\
&= -\frac{1}{N} \sum_{n=1}^N \log p_{\mathrm{u}}\left(T^{-1}\left(\mathrm{x}_n ; \phi\right) ; \psi\right)+\log \left|J_{T^{-1}}\left(\mathrm{x}_n ; \phi\right)\right|+\text{const}.
\end{aligned}
$$

With the resulting model $p_{\mathrm{x}}(\mathrm{x} ; \theta)$

- 1) density estimation
- 2) generation

(a) Density Estimation

use model to calculate densities

1) low-dim cases ...NF can represent skewed, multi-modal densities

2) use density function to perform one-class classification

3) detect rotations and corruptions of images

4) composition allows for better density estimation (MADE, Real NVP)

(b) Generation

sampling from model (use as it is sampled from $p_x^*(\mathbf{x})$)

image, video, etc...

6-2. Inference

modeling params to infer **unknown quantities** within a model

most common setting : $\int \pi(\eta) d\eta$

summarize the use of flows for

- 1) sampling
- 2) variational inference
- 3) likelihood-free inference

6-2-1. Importance and Rejection sampling

Importance Sampling (IS)

- auxiliary distn $q(\eta)$

  $\int \pi(\eta)d\eta = \int q(\eta)\frac{\pi(\eta)}{q(\eta)}d\eta = \mathbb{E}_{q(\eta)}\left[\frac{\pi(\eta)}{q(\eta)}\right] \approx \frac{1}{S}\sum_{s=1}^{S}\frac{\pi(\hat{\eta}_s)}{q(\hat{\eta}_s)}.$

  - $q(\eta)$ is a user-specified density function
  - $\hat{\eta}_s$ is a sample from $q(\eta)$
- choice of $q(\eta)$
  - if it doesn't contain $\pi(\eta)$'s, estimate becomes impractical
  - if it is too broad : inefficiency
- Importance of using expressive proposals $q(\eta)$!

  $\rightarrow$ **employ NFs**
- Requires both sampling & density evaluation to be tractable for many flows

2 alternatives to optimize flow's params

- 1) minimize KL div
- 2) minimize the variance of IS estimator

Rejection sampling (RS)

- aims to draw samples from $p(\boldsymbol{\eta}) = \pi(\boldsymbol{\eta})/Z$
- use Real NVP to parameterize proposal distn

  (since coupling layers allow for fast density evaluation & sampling)

6-2-2. Reparameterizing models for MCMC

NF can be integrated into MCMC

$\rightarrow$ by using the flow to **reparam the target distn**

(can effectively smooth away pathologies, by allowing MCMC to be run on the simpler & better-behaved base density)

Metropolis-Hastings ratio to the reparameterized model

$r\left(\hat{\mathbf{u}}_*; \hat{\mathbf{u}}_t\right) = \frac{p_{\mathrm{u}}(\hat{\mathbf{u}}_*)}{p_{\mathrm{u}}(\hat{\mathbf{u}}_t)} = \frac{\pi(T(\hat{\mathbf{u}}_*;\phi))|\det J_T(\hat{\mathbf{u}}_*;\phi)|}{\pi(T(\hat{\mathbf{u}}_t;\phi))|\det J_T(\hat{\mathbf{u}}_t;\phi)|}.$

- $\hat{\mathbf{u}}_*$: proposed value
- $\hat{\mathbf{u}}_t$: current value

6-2-3. Variational Inference

use NF to fit **distn over latent variables**

"flows can usefully serve as posterior approximations"

Use a (trained) flow-based model

$p(\eta \mid \mathrm{x}) \approx q(\eta ; \phi)=q_{\mathrm{u}}(\mathbf{u})\left|\det J_T(\mathbf{u} ; \phi)\right|^{-1}$

- $q_{\mathrm{u}}(\mathbf{u})$: base distn
- $T(\cdot ; \phi)$: transformation

By maximizing ELBO

$$
\begin{aligned}
\log p(\mathbf{x}) & \geq \mathbb{E}_{q(\eta ; \phi)}[\log p(\mathbf{x}, \eta)]-\mathbb{E}_{q(\eta ; \phi)}[\log q(\eta ; \phi)] \\
& =\mathbb{E}_{q_{\mathbf{u}}(\mathbf{u})}[\log p(\mathbf{x}, T(\mathbf{u} ; \phi))]-\mathbb{E}_{q_{\mathbf{u}}(\mathbf{u})}\left[\log q_{\mathbf{u}}(\mathbf{u})\right]+\mathbb{E}_{q_{\mathbf{u}}(\mathbf{u})}\left[\log \left|\det J_T(\mathbf{u} ; \phi)\right|\right] \\
& =\mathbb{E}_{q_{\mathbf{u}}(\mathbf{u})}[\log p(\mathbf{x}, T(\mathbf{u} ; \phi))]+\mathbb{H}\left[q_{\mathbf{u}}(\mathbf{u})\right]+\mathbb{E}_{q_{\mathbf{u}}(\mathbf{u})}\left[\log \left|\det J_T(\mathbf{u} ; \phi)\right|\right]
\end{aligned}
$$

- $\mathbb{H}\left[q_{\mathrm{u}}(\mathbf{u})\right]$: differential entropy (constant w.r.t ϕ)
- with MC estimation...

  $$
  \begin{aligned}
  \mathbb{E}_{q_{\mathrm{u}}(\mathbf{u})}[\log p(\mathbf{x}, T(\mathbf{u} ; \phi))] & \approx \frac{1}{S} \sum_{s=1}^S \log p\left(\mathbf{x}, T\left(\hat{u}_s ; \phi\right)\right) \\
  \mathbb{E}_{q_{\mathrm{u}}(\mathbf{u})}\left[\log \left|\det J_T(\mathbf{u} ; \phi)\right|\right] & \approx \frac{1}{S} \sum_{s=1}^S \log \left|\det J_T\left(\hat{u}_s ; \phi\right)\right|
  \end{aligned}
  $$