## [ Paper review 47 ]

# Normalizing Flows : An Introduction and Review of Current Methods

### ( Kobyszev, et al., 2020 )

## [ Contents ]

# 1. Abstract

Normalizing Flows

- **generative models**, which produce **tractable** distn
- where both **(1) sampling** and **(2) density evaluation** can be efficient & exact

# 2. Introduction

Goal of statistics & ML : **model a probability distn**, given samples from that distn!

- example of unsupervised
- called **generative modeling**
- Applications : density estimation, outlier detection, prior construction, data summarization

### Generative Modeling

**(1) Variational approaches & EM :**

- introduce latent variables to explain observed data
- provide flexibility, but increase complexity of inference

**(2) Graphical models**

- model the conditional dependence between r.v

**(3) Generative neural appraoches**

- GANs, VAEs
- impressive performance results on hard tasks ( ex. learning distn of images )
- but neither allows exact evaluation of **probability density of new points** )

  ( + mode collapse, posterior collapse, vanishing gradients... )

**(4) Normalizing Flows (NF)**

- family of  generative models with **tractable distn**
- both **sampling & density evaluation** can be efficient & exact
- applications : image generation, noise modeling, video generation, audio generation, graph generation, RL....

Section 2 : NF & training NF

Section 3 : constructions for NF

Section 4 : datasets for testing NF & performance of different approaches

# 3. Background

NF was popularized in context of (1) VI & (2) density estimation

Normalizing Flow (NF)

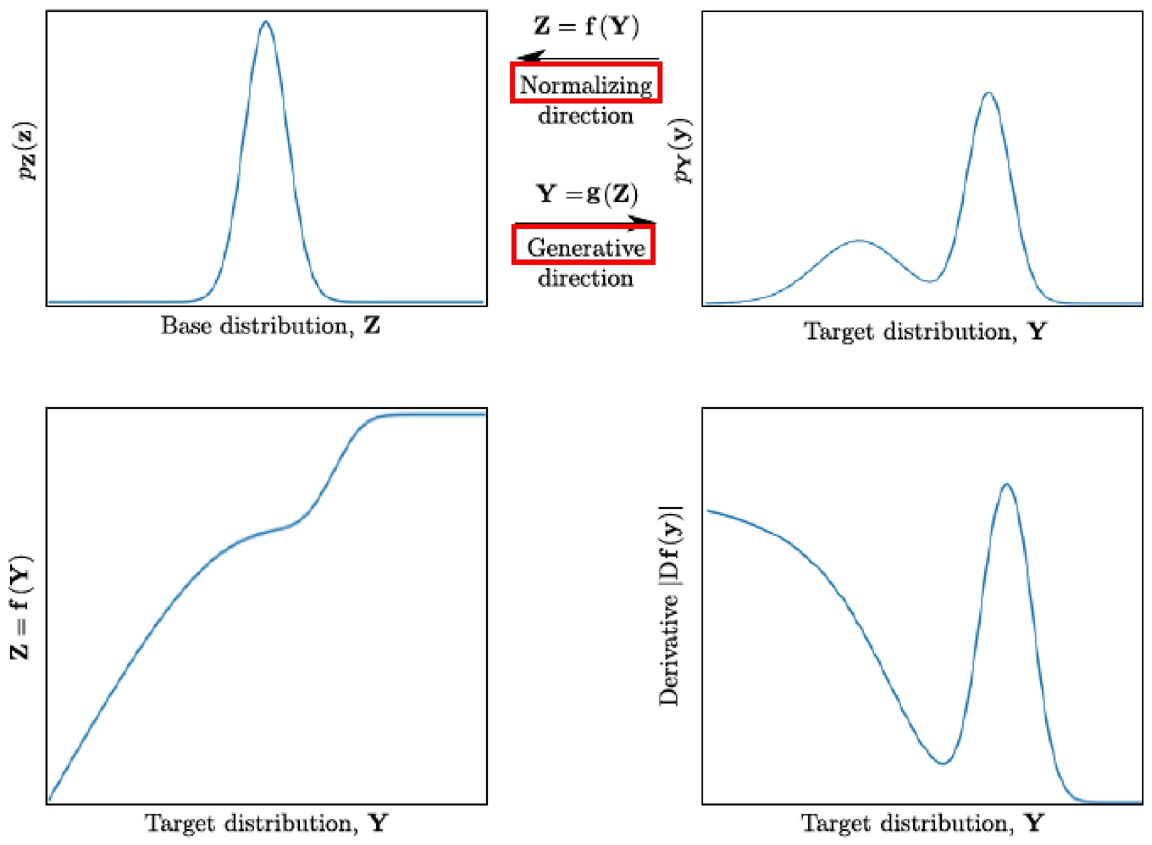
- simple $\rightarrow$ complex distn
- by sequence of **invertible** and **differentiable** mappings
- how to evaluate **density of sample**?

  by transforming BACK to the original sample,

  then compute the product of (1) & (2)

  - (1) density of the inverse-transformed sample under this distn
  - (2) **change in volume** ( = product of determinants of Jacobian )
- 1. **Density Evaluation** : ( as above )
  2. **Sampling** : by sampling from the initial density & apply transformation

## 3-1. Basics



change of variable formula :

$$\begin{aligned} p_{\mathbf{Y}}(\mathbf{y}) &= p_{\mathbf{Z}}(\mathbf{f}(\mathbf{y}))|\det \mathrm{Df}(\mathbf{y})| \\ &= p_{\mathbf{Z}}(\mathbf{f}(\mathbf{y}))|\det \mathrm{Dg}(\mathbf{f}(\mathbf{y}))|^{-1} . \end{aligned}$$

- $\mathbf{f}$ : inverse of $\mathbf{g}$
- $\mathrm{Df}(\mathbf{y}) = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ : Jacobian of $\mathbf{f}$
- $\mathrm{Dg}(\mathbf{z}) = \frac{\partial \mathbf{g}}{\partial z}$: Jacobian of $\mathbf{g}$
- $p_{\mathbf{Y}}(\mathbf{y})$: called **push forward** of density $p_{\mathbf{z}}$ by the function $\mathbf{g}$

If the transformation $\mathbf{g}$ can be arbitrary complex,
one can generate any distribution $p_{\mathbf{Y}}$ from any base distn $p_{\mathbf{Z}}$

Constructing arbitrary complicated non-linear invertible functions (bijections) can be difficult

Normalizing flow : bijections which are convenient to

- (1) compute

  (2) invert

  (3) calculate the determinant of their Jacobian
- (generative direction) $\mathrm{g} = \mathrm{g}_N \circ \mathrm{g}_{N-1} \circ \cdots \circ \mathrm{g}_1$

  (normalizing direction) $\mathbf{f} = \mathbf{f}_1 \circ \cdots \circ \mathbf{f}_{N-1} \circ \mathbf{f}_N$
- determinant of the Jacobian : $\det \mathrm{Df}(\mathbf{y}) = \prod_{i=1}^{N} \det \mathrm{Df}_i\left(\mathbf{x}_i\right)$
  - where $\mathrm{Df}_i(\mathbf{y}) = \frac{\partial \mathrm{f}_i}{\partial \mathrm{x}}$ ( = Jacobian of $f_i$ )
- intermediate flow : $\mathbf{x}_i = \mathbf{g}_i \circ \cdots \circ \mathbf{g}_1(\mathbf{z}) = \mathbf{f}_{i+1} \circ \cdots \circ \mathbf{f}_N(\mathbf{y})$

  ( $\mathbf{x}_N = \mathbf{y}$ )

## 3-2. Applications

### 3-2-1. Density Estimation & Sampling

Natural & most obvious use of NF = **Density Estimation**

Notation

- single flow : $\mathbf{g}$, parameterized by $\theta$
- base measure : $p_{\mathbf{z}}$ , parameterized by $\phi$
- observed data from complicated distn : $\mathcal{D}=\left\{\mathbf{y}^{(i)}\right\}_{i=1}^{M}$
- we can perform likelihood-based estimation of the params $\Theta=(\theta, \phi)$

Data Likelihood :

$$
\begin{aligned}
\log p(\mathcal{D} \mid \Theta) &=\sum_{i=1}^{M} \log p_{\mathbf{Y}}\left(\mathbf{y}^{(i)} \mid \Theta\right) \\
&=\sum_{i=1}^{M} \log p_{\mathbf{Z}}\left(\mathbf{f}\left(\mathbf{y}^{(i)} \mid \theta\right) \mid \phi\right)+\log \mid \det \mathrm{Df}\left(\mathbf{y}^{(i)} \mid \theta\right)
\end{aligned}.
$$

During training,

- param of the flow ( = $\theta$ )
- param of base distn ( = $\phi$ )

are adjusted to maximize the log-likelihood

#### (1) Density Estimation

- (1) computing $\mathbf{f}$ ( = normalizing direction )
- (2) log determinant

  $\rightarrow$ efficiency of these are important!

#### (2) Sampling

- (1) evaluating the inverse $\mathbf{g}$ ( = generative direction )

  $\rightarrow$ performance is determined by the cost of generative direction

### 3-2-2. Variational Inference

Notation

- latent variable model : $p(\mathbf{x})=\int p(\mathbf{x}, \mathbf{y}) d \mathbf{y}$
- posterior distn : $p(\mathbf{y} \mid \mathbf{x})$ ... usually intractable
- approximate posterior : $q(\mathbf{y} \mid \mathbf{x}, \theta)$

By minimizing KL-div ( = $D_{KL}(q(\mathbf{y} \mid \mathbf{x}, \theta) \| p(\mathbf{y} \mid \mathbf{x}))$ )

( = By maximizing ELBO ( = $\mathcal{L}(\theta)=\mathbb{E}_{q(\mathbf{y} \mid \mathbf{x}, \theta)}[\log (p(\mathbf{y}, \mathbf{x}))-\log (q(\mathbf{y} \mid \mathbf{x}, \theta))]$ ))

Optimization can be done with gradient descent!

- need to compute $\nabla_\theta \mathbb{E}_{q(\mathbf{y} \mid \mathbf{x}, \theta)}[h(\mathbf{y})]$, which is not straight forward
- Reparameterize $q(\mathbf{y} \mid \mathbf{x}, \theta)=p_{\mathbf{Y}}(\mathbf{y} \mid \theta)$ with Normalizing Flows

  $\rightarrow \mathbb{E}_{p_{\mathbf{Y}}(\mathbf{y} \mid \theta)}[h(\mathbf{y})]=\mathbb{E}_{p_{\mathbf{Z}}(\mathbf{z})}[h(\mathbf{g}(\mathbf{z} \mid \theta))]$

# 4. Methods

for NF to be practical...

- (1) invertible
- (2) sufficiently expressive
- (3) computationally efficient

  ( both in terms of computing $\mathbf{f}$ and $\mathbf{g}$ & determinant of Jacobian )

Will introduce different types of flows

- (4-1) Elementwise Flows
- (4-2) Linear Flows
- (4-3) Planar and Radial Flows
- (4-4) Coupling and Autoregressive Flows
- (4-5) Residual Flows
- (4-6) Infinitesimal (Continuous) Flows

## 4-1. Elementwise Flows

Bijective scalar function ($h: \mathbb{R} \rightarrow \mathbb{R}$ )

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

- bijection whose inverse simply requires computing $h^{-1}$
- Jacobian : product of the absolute values of derivatives of $h$
- $h$ : activation function

  ( ReLU : not bijective )

  ( Leaky ReLU : bijective )

  ( spline-based activation functions have also been considered )

## 4-2. Linear Flows

Elementwise operation can not express **correlation** between dimensions, but **Linear mappings** can!

$g(x)=A x+b.$

- $\mathbf{A} \in \mathbb{R}^{D \times D}$ and $\mathbf{b} \in \mathbb{R}^{D}$
- if $\mathbf{A}$ is invertible, function is also invertible

Linear flows are limited in their expressiveness

- if Gaussian based distn $\rightarrow$ Gaussian

  ( ex. $p_{\mathbf{Z}}(\mathbf{z})=\mathcal{N}(\mathbf{z}, \mu, \Sigma) \rightarrow p_{\mathbf{Y}}=\mathcal{N}\left(\mathbf{y}, \mathbf{A} \mu+\mathbf{b}, \mathbf{A}^{T} \Sigma \mathbf{A}\right)$ )
- if Exp family based distn $\rightarrow$ Exp family

Determinant of Jacobian : $\det(\mathbf{A})$

$\rightarrow$ computed in $\mathcal{O}\left(D^{3}\right)$...expensive

How to make it more practical?

## 4-2-1. Diagonal

Its inverse can be computed in **linear time**

**determinant = product of diagonal entries**

but this is just elementwise transformation....

## 4-2-2. Triangular

More expressive

**determinant = product of diagonal entries**

( As long as all the diagonals are non-zero, it is non-singular )

Inversion : inexpensive ( costing $\mathcal{O}\left(D^{2}\right)$ )

Combine Triangular matrix!

- $K$ triangular matrices $\mathbf{T}_{i}$
- $K$ dim probability vector $\omega$

  $\rightarrow$ General expression : weight is like weighted sum! $\mathbf{y}=\left(\sum_{i=1}^{K} \omega_{i} \mathbf{T}_{i}\right) \mathbf{z}$
- Determinant : 1
- if some are upper triangle / some are lower triangle.... it costs $\mathcal{O}\left(D^{3}\right)$ for inversion

## 4-2-3. Permutation & Orthogonal

Expressiveness of triangular transformation is sensitive to **ordering of dimension**

$\rightarrow$ use **"Permutation matrix"** to easily reorder! ( = determinant of 1 )

More general alternative : **Orthogonal transformation**

- inverse & absolute determinant is both trivial $\rightarrow$ efficient
- (Linear Algebra) *"Any orthogonal matrix can be written as a production of reflections"*

- parameterize reflection matrix $H$ ( in $\mathbb{R}^D$ ) by..
  - fix a non-zero vector $\mathbf{v} \in \mathbb{R}^D$
  - define $H = 1 - \frac{2}{\|\mathbf{v}\|^2}\mathbf{v}\mathbf{v}^T$

## 4-2-4. Factorizations

Instead of limiting the form of $\mathbf{A}$,

use **LU  factorziation** : $\mathbf{g}(x) = \mathbf{PLUx} + \mathbf{b}$

- $\mathbf{L}$ : lower triangular ( with 1 on the diagonal )
- $\mathbf{U}$ : upper triangular ( with non-zero diagonal )
- $\mathbf{P}$ : permutation matrix

(a) determinant : **product of the diagonal of $\mathbf{U}$** :   $\rightarrow$ can be computed in $O(D)$

(b) inverse : computed using two passes of backward substitution  $\rightarrow$ can be computed in $O(D^2)$

BUT, $\mathbf{P}$ can not be easily optimized.... So randomly generate initially & fix it

- but fixing it limits the flexibility....thus propose **QR decomposition**

## 4-2-5. Convolution

Convolutions :

- (pros) easy to compute
- (cons) inverse & determinant are non-obvious

(1) $1 \times 1$ **convolutions**

- just full linear transformation, which is applied only across channels

(2) 1D convolutions (**ConvFlow**)

- (Zheng et al. 2018)
- exploited the triangular structure
- efficiently compute the determinant

(1) $d \times d$ **convolutions**

- more general solution
- by stacking together masked autoregressive convolutions ( = Emerging Conv )
- by exploiting the Fourier domain representation to efficiently compute inverse & determinant ( = Periodic Conv )

# 4-3. Planar and Radial Flows

Relatively simple, but inverses aren't easily computed :(

### 4-3-1. Planar Flows

$\mathrm{g}(\mathrm{x})=\mathrm{x}+\mathrm{u} h\left(\mathrm{w}^{T} \mathrm{x}+b\right)$.

- (params) $\mathbf{u}, \mathbf{w} \in \mathbb{R}^{D}$ and $b \in \mathbb{R}$
- $h: \mathbb{R} \rightarrow \mathbb{R}$ : smooth non-linearity

Jacobian determinant :

$$\begin{aligned} \operatorname{det}\left(\frac{\partial \mathrm{g}}{\partial \mathrm{x}}\right) &=\operatorname{det}\left(1_{D}+\mathbf{u} h^{\prime}\left(\mathbf{w}^{T} \mathbf{x}+b\right) \mathbf{w}^{T}\right) \\ &=1+h^{\prime}\left(\mathbf{w}^{T} \mathbf{x}+b\right) \mathbf{u}^{T} \mathbf{w} \end{aligned}.$$

- (a) Jacobian : can be computed in $\mathcal{O}(D)$
- (b) Inverse : may not exist...but possible for certain choices of $h(\cdot)$

Term $\mathrm{u} h\left(\mathrm{w}^{T} \mathrm{x}+b\right)$

- interpreted as MLP with a bottleneck hidden layer with a single unit
- bottleneck= "need to stack many planar flows"  to get expressive result

### ( + Sylvester flows )

$\mathrm{g}(\mathrm{x})=\mathrm{x}+\mathrm{U} \mathrm{h}\left(\mathrm{W}^{T} \mathrm{x}+\mathrm{b}\right)$.

- where $\mathbf{U}$ and $\mathbf{W}$ are $D \times M$ matrices & $\mathbf{b} \in \mathbb{R}^{M}$
- $\mathbf{h}: \mathbb{R}^{M} \rightarrow \mathbb{R}^{M}$: smooth non-linearity
- $M \leq D$ : hyperparameter to choose ( = like dimension of hidden layer )

Jacobian determinant :

$$\begin{aligned} \operatorname{det}\left(\frac{\partial \mathrm{g}}{\partial \mathrm{x}}\right) &=\operatorname{det}\left(1_{D}+\mathbf{U} \operatorname{diag}\left(\mathbf{h}^{\prime}\left(\mathbf{W}^{T} \mathbf{x}+b\right)\right) \mathbf{W}^{T}\right) \\ &=\operatorname{det}\left(1_{M}+\operatorname{diag}\left(\mathbf{h}^{\prime}\left(\mathbf{W}^{T} \mathbf{x}+b\right)\right) \mathbf{W} \mathbf{U}^{T}\right) \end{aligned}.$$

### 4-3-2. Radial Flows

Modify the distn around a specific point

$g(x)=x+\frac{\beta}{\alpha+\left\|x-x_{0}\right\|}\left(x-x_{0}\right)$.

- where $\mathrm{x}_{0} \in \mathbb{R}^{D}$ is the point around which the dist distorted

(1) Jacobian determinant : relatively efficiently

(2) Inverse : cannot be given in closed form ....

## 4-4. Coupling and Autoregressive Flows

## 4-4-1. Coupling Flows

disjoint partition : $\left(\mathrm{x}^A, \mathrm{x}^B\right) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$

function g : $\mathbb{R}^D \rightarrow \mathbb{R}^D$ :

$$\begin{aligned} \mathbf{y}^A & =\mathbf{h}\left(\mathbf{x}^A ; \Theta\left(\mathbf{x}^B\right)\right) \\ \mathbf{y}^B & =\mathbf{x}^B \end{aligned}.$$

- $\Theta\left(\mathrm{x}^B\right)$ : "conditioner"
- $\mathbf{h}$ : "coupling function"
- $\mathbf{g}$ : "coupling flow"

$\mathbf{g}$ is invertible $\leftrightarrow$ $\mathbf{h}$ is invertible & has inverse

$$\begin{aligned} \mathbf{x}^A & =\mathbf{h}^{-1}\left(\mathbf{y}^A ; \Theta\left(\mathbf{x}^B\right)\right) \\ \mathbf{x}^B & =\mathbf{y}^B \end{aligned}.$$

Jacobian of $\mathbf{g}$ : block triangular matrix

$\rightarrow$ determinant of the Jacobian = determinant of $D\mathbf{h}$

Most coupling layers are applied to $\mathrm{x}^A$ element-wise

$$\mathbf{h}\left(\mathbf{x}^A ; \theta\right)=\left(h_1\left(x_1^A ; \theta_1\right), \ldots, h_d\left(x_d^A ; \theta_d\right)\right)$$

- triangular transformation

Conditioner $\Theta\left(\mathrm{x}^B\right)$

- has to be complex ( ex. NN ) for flow to be complex
- sometimes, can be constant! ( = "multi-scale flow" )

  $\rightarrow$ gradually introduce new dimension in the generative direction

  $\rightarrow$ reduces the computation of transforming high dim distn & can capture the multi-scale structure

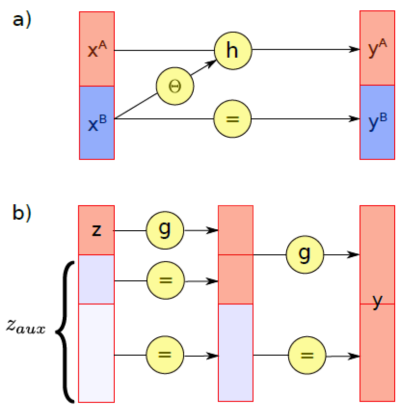


Fig. 3. Coupling architecture. a) A single coupling flow described in Equation (15). A coupling function $\mathbf{h}$ is applied to one part of the space, while its parameters depend on the other part. b) Two subsequent multi-scale flows in the generative direction. A flow is applied to a relatively low dimensional vector $\mathbf{z}$; its parameters no longer depend on the rest part $\mathbf{z}_{aux}$. Then new dimensions are gradually introduced to the distribution.

Then, how to partition $\mathbf{x}$ ?

- simple : (1) permutation $\rightarrow$ (2) split into half
- others : "masked flows" ( Real NVP )

  "1x1 conv" ( Glow )

## 4-4-2. Autoregressive Flows

Non-linear generalizations of multiplications by **triangular** matrix

$y_t = h\left(x_t; \Theta_t\left(\mathrm{x}_{1:t-1}\right)\right)$

- $\Theta_t(\cdot)$ : conditioner
- conditioned on the previous entries of the input

(1) Jacobian :

- triangular $\rightarrow$ efficiently in one-pass!
- $\det(\mathrm{Dg}) = \prod_{t=1}^{D} \frac{\partial y_t}{\partial x_t}$.

(2) Inverse

- (given then inverse of $h$) inverse of $\mathbf{g}$ can be found **with recursion**
- $x_1 = h^{-1}\left(y_1; \Theta_1\right)$.

  $x_t = h^{-1}\left(y_t; \Theta_t\left(\mathrm{x}_{1:t-1}\right)\right)$
- sequential...difficult to make it efficiently & can not be parallelized

## ( + Inverse Autoregressive Flow (IAF) )

- outputs each entry of $\mathbf{y}$, conditioned the previous entries of $\mathbf{y}$
- $y_t = h\left(x_t; \theta_t\left(\mathbf{y}_{1:t-1}\right)\right)$
- Computation : sequential & expensive

  Inverse : relatively efficient
- IAF = ***flow in the generative direction***

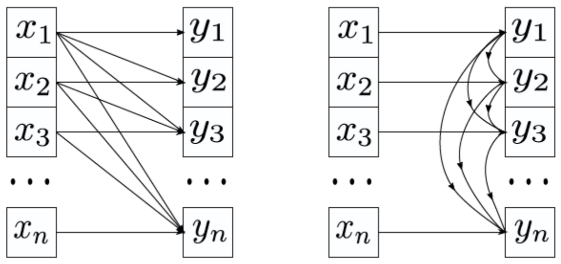


Fig. 4. Autoregressive flows. On the left, is the direct autoregressive flow given in Equation (18). Each output depends on the current and previous inputs and so this operation can be easily parallelized. On the right, is the inverse autoregressive flow from Equation (20). Each output depends on the current input and the previous outputs and so computation is inherently sequential and cannot be parallelized.

IAF vs MAF

- IAF ) **fast sampling**
- MAF ) **fast density estimation**

## 4-4-3. Universality

( = *can learn any target* , given sufficient capacity & data )

To claim that a class of AF is universal, it is enough to show that ***a family of coupling cuntions $h$ used in the calss is dense in the set of all "monotone" functions in the pointwise convergence topology***

## 4-4-4. Coupling Functions

Coupling Flows & Autoregressive flows

- have similar functional form
- both have coupling functions as building blocks
- (Coupling Flows) coupling functions are typically "scalar" coupling functions

  (Autoregressive Flows) $d = 1$ and they are also scalar valued

Will deal with coupling functions below

- (a) Affine coupling
- (b) Nonlinear squared flow
- (c) Continuous mixture CDFs
- (d) Splines
- (e) Neural autoregressive flow
- (f) Sum-of-Squares polynomial flow
- (g) Piecewise-bijective coupling

**(a) Affine coupling**

NICE proposed 2 coupling functions

- (1) **Additive** coupling function : $h(x;\theta)=x+\theta, \quad \theta\in\mathbb{R}$
- (2) **Affine** coupling function : $h(x;\theta)=\theta_1 x+\theta_2, \quad \theta_1\neq 0, \theta_2\in\mathbb{R}$

**Affine** coupling function

- used in NICE, RealNVP, Glow, IAF
- simple & efficient
- but, expressiveness is limited $\rightarrow$ Thus, must be stacked

**(b) Nonlinear squared flow**

$h(x;\theta)=ax+b+\frac{c}{1+(dx+h)^2}$

- under some constraints, it is invertible & inverse is analytically computable
- good at learning multi-modal distn

**(c) Continuous mixture CDFs**

Propose **Flow++**

- more expressive coupling function
- layer : (linear transformation + monotone function to $x$ )

  $h(x;\theta)=\theta_1 F\left(x,\theta_3\right)+\theta_2$

  - $F(x,\pi,\mu,\mathrm{s})$ : CDF of a mixture of $K$ logistics, postcomposed with an inverse sigmoid:

    $F(x,\pi,\mu,\mathbf{s})=\sigma^{-1}\left(\sum_{j=1}^K \pi_j \sigma\left(\frac{x-\mu_j}{s_j}\right)\right).$
  - derivative of transformation w.r.t $x$ : in terms of PDF of logistic mixture
  - computation : not expensive

**(d) Splines**

- piecewise-polynomial ( $K+1$ points, $(x_i,y_i)_{i=0}^K$ called **knots** )
- Piecewise-linear
  - linear splines for $h:[0,1]\rightarrow[0,1]$
  - divide domain into $K$ equal bins
  - $h(x;\theta)=\alpha\theta_b+\sum_{k=1}^{b-1}\theta_k.$
    - $\theta\in\mathbb{R}^K$: probability vector
    - $b=\lfloor Kx\rfloor$: bin that contains $x$
    - $\alpha=Kx-b$ : position of $x$ in bin $b$
  - invertible ( if all $\theta_k>0$ ) & derivative : $\frac{\partial h}{\partial x}=\theta_b K$
- Piecewise-quadratic
- Cubic Splines
  - monotone cubic splines for a coupling function
  - do not restrict the domain to unit interval!
  - $h(\cdot;\theta)=\sigma^{-1}(\hat{h}(\sigma(\cdot);\theta))$

where $\hat{h}(\cdot;\theta) : [0,1] \rightarrow [0,1]$
  - computation of derivative is easy! ( $\because$ piecewise-quadratic )
  - can be trained by gradient descent
- Rational quadratic splines

**(e) Neural autoregressive flow (NAF)**

coupling function $h(\cdot;\theta)$ : DNN

- not invertible, but can be bijective

2 forms of NN

- (1) NAF-DSF ( Deep Sigmoidal coupling function )
  - single hidden layer of sigmoid units
- (2) NAF-DDSF ( Deep Dense Sigmoidal coupling function )
  - more general ( does not have this bottle neck as NAF-DSF )
  - universal flow

Imposing positivity of weights on flow $\rightarrow$ makes training harder & require more codntion

Thus, introduce **UMNN(Unconstrained Monotonic NN)**

- strictly pos/neg function with NN & integrate it numerically
- require less params than NAF
- scalable for high-dim dataset

**(f) Sum-of-Squares polynomial flow**

$h(\cdot;\theta)$ : strictly increasing polynomial

- universal flow

coupling function :

$h(x;\theta) = c + \int_0^x \sum_{k=1}^K \left(\sum_{l=0}^L a_{kl} u^l\right)^2 du.$

- need to integrate Sum of Squares

Easier to train than NAF ( $\because$ no restrictions on the params )

for $L = 0$ : SOS = affine coupling function

**(g) Piecewise-bijective coupling**

- coupling function need not be bijective ( just **Piecewise-bijective** )
- **Each part of the base distn**, recieves contributions from **each subset of data domain**
- notation

- target $y$
- base $z$
- lookup function $\phi : \mathbb{R} \rightarrow [K] = \{1, \ldots, K\}$

  ( such that $\phi(y) = k$ if $y \in A_k$ )
- density on a target space : $p_{Z,[K]}(z, k) = p_{[K] \mid Z}(k \mid z) p_Z(z)$

- for each point $z$, can find its pre-image by sampling from $p_{[K] \mid Z}$ ( = gating network )
- $p_Y(y) = p_{Z,[K]}(h(y), \phi(y)) |\mathrm{D}h(y)|$

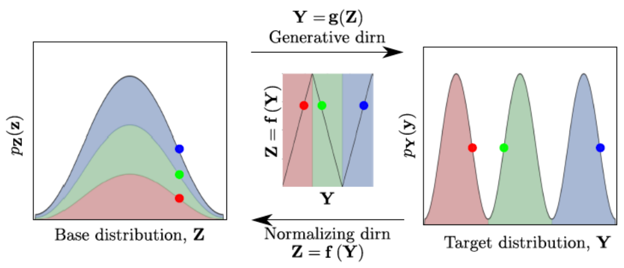

$\mathbf{Y} = \mathbf{g}(\mathbf{Z})$
Generative dirn
$p_Z(\mathbf{z})$
$\mathbf{Z} = \mathbf{f}(\mathbf{Y})$
$p_Y(\mathbf{y})$
$\mathbf{Y}$
Base distribution, $\mathbf{Z}$
Normalizing dirn
$\mathbf{Z} = \mathbf{f}(\mathbf{Y})$
Target distribution, $\mathbf{Y}$

Fig. 5. Piecewise bijective coupling. The target domain (right) is divided into disjoint sections (colors) and each mapped by a monotone function (center) to the base distribution (left). For inverting the function, one samples a component of the base distribution using a gating network.

## 4-5. Residual Flows

$\mathrm{g}(\mathrm{x}) = \mathrm{x} + F(\mathrm{x})$

- $\mathrm{g}(\mathrm{x})$ : Residual connection
- $F(\cdot)$ : Residual block

Building a reversible network based on residual connections : **RevNets & iRevNets**

- disjoint partition $\mathbf{x} = (\mathbf{x}^A, \mathbf{x}^B)$..... $\mathbb{R}^D = \mathbb{R}^d \times \mathbb{R}^{D-d}$

  $$\begin{aligned} \mathbf{y}^A &= \mathbf{x}^A + F(\mathbf{x}^B) \\ \mathbf{y}^B &= \mathbf{x}^B + G(\mathbf{y}^A), \end{aligned}$$

  where $F : \mathbb{R}^{D-d} \rightarrow \mathbb{R}^d$ and $G : \mathbb{R}^d \rightarrow \mathbb{R}^{D-d}$
- Invertible

  ***A residual connection is invertible, if the Lipschitz constant of the residual block is Lip(F) < 1.***

  - no closed form for the inverse
  - but can be found numerically
- Computing Jacobian is inefficient

## 4-6. Infinitesimal (Continuous) Flows