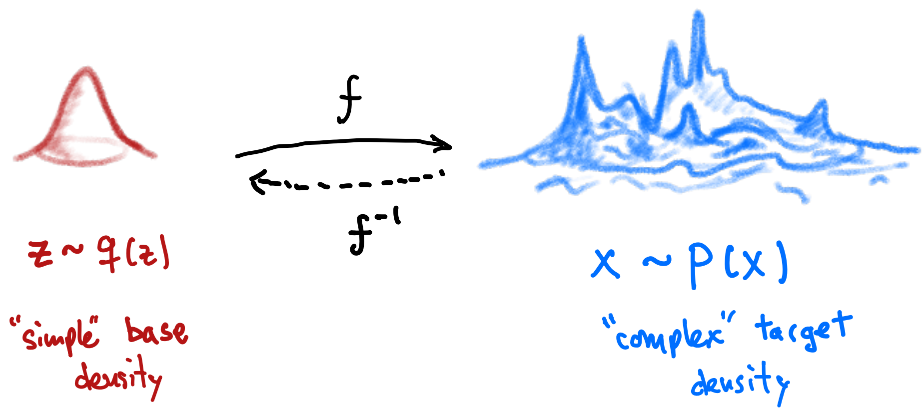
I. Normalizing flows (REF: Papamakarios, et al JMLR 1912.02762)

- core idea: $\boxed{\text{map a "simple" distribution into a "complex" one using the change-of-variables formula}}$ Rezende, Mohamed 2015' ICML

$x = f(z) \quad f: \mathbb{R}^d \to \mathbb{R}^d$

- f smooth and invertible map (bijection)

$z = f^{-1}(x)$



the map above induces a change in densities given by:

$$p(x) = q(z) \cdot \left|\det \frac{\partial f^{-1}(x)}{\partial x}\right| = q(z) \left|\det \frac{\partial f}{\partial z}\right|^{-1}$$

"pushforward of q by f"

jacobian matrix $(d \times d)$ $\quad J_{ij}(z) \equiv \dfrac{\partial f_i}{\partial z_j}$

- log-likelihood: $$\boxed{\log p(x) = \log q(z) - \underbrace{\log \left|\det \frac{\partial f}{\partial z}\right|}_{\text{volume correction}}}$$

- $q \sim N(0, 1)$ Gaussian then f is a "normalizing" map.
- Transformation deforms the simple density via volume expansions/contractions
- f would need to be fairly complicated if we want $p(x)$ to be an arbitrarily complex density...

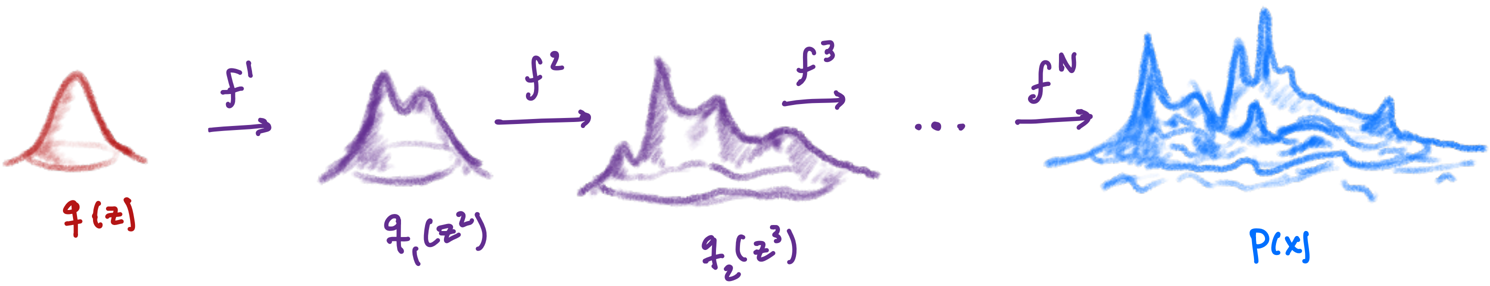
- Normalizing "flows": build complex map by composing many simple maps.

$$f = f^N \circ f^{N-1} \circ \dots \circ f^1 \qquad \text{"Flow of maps"}$$

$$z = \boxed{z^1} \xrightarrow{f^1} \boxed{z^2} \xrightarrow{f^2} \boxed{z^3} \xrightarrow{f^3} \cdots \xrightarrow{f^N} \boxed{z^N} = x$$

allows for more expressive "f".

- we get a finite family of distributions q_n with increasing complexity.



log-likelihood:
$$\boxed{\log P(x) = \log q(z) - \sum_{n=1}^{N} \log \left| \det \frac{\partial f^n}{\partial z^n} \right|}$$

- Remarks:

i) only works for distributions with continuous random variables.

ii) Flow is "finite" $\{1, \dots, N\}$ "discrete time".

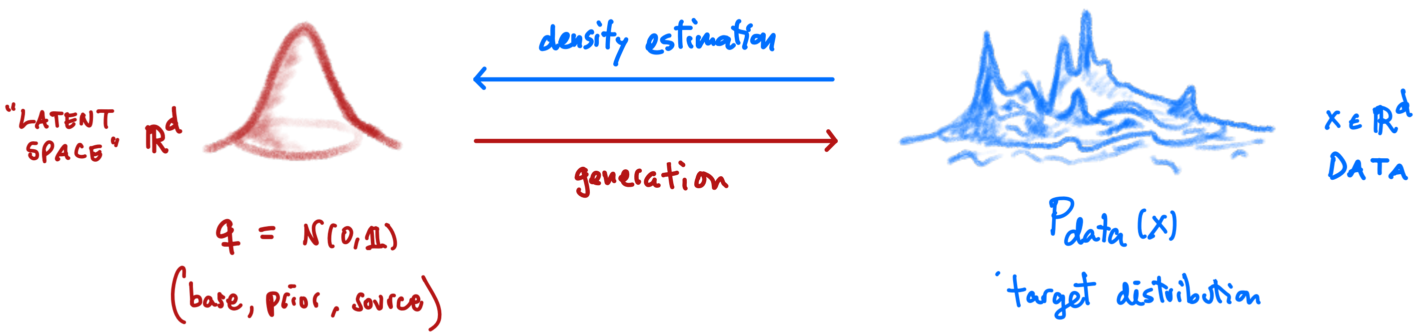
iii) All f^n maps need to be smooth, invertible (otherwise change of variable theorem breaks)

II. Normalizing flows as deep generative models

- learn a normalizing flow f_θ between a latent space $z \sim q(z) = \mathcal{N}(z \mid 0, \mathbb{1})$ and the data $x \sim P_{data}(x)$, where θ are learnable parameters.

$$\boxed{f_\theta = f_\theta^1 \circ \cdots \circ f_\theta^N}$$ parametrized by deep NN with "N" layers

once f_θ is learned $\Rightarrow$ $\begin{cases} \text{1. we can generate synthetic data! (generative model)} \\ \text{2. we have an approximation of } P_{data}\text{! (density estimation)} \end{cases}$



- Ancestral sampling: $\boxed{\begin{array}{l} i.\ z_0 \sim \mathcal{N}(0, \mathbb{1}) \\ ii.\ x_{NEW} = f_\theta(z_0) \end{array}}$

- (non-compressed) latent representation: $z = (f_\theta)^{-1}(x)$

- **Training**:

the goal is to aproximate the target distribution $P_{data}(x)$ with the "pushforward" $P_\theta(x)$ resulting from the NF model.

$\Rightarrow$ minimize the KL-divergence $\mathcal{D}_{KL}(P_{data} \| P_\theta) = \int dx \, P_{data} \log\left(\frac{P_{data}}{P_\theta}\right)$

$$\mathcal{L} = \mathcal{D}_{KL}(P_{data} \| P_\theta) = -\mathbb{E}_{P_{data}}(\log P_\theta) + \cancel{H(P_{data})}$$

DATA ENTROPY
$(H(p) = \mathbb{E}_p \log p)$
IRRELEVANT when minimizing
independent of "θ".

$$\Rightarrow \quad \mathcal{L} = -\mathbb{E}_{P_{data}}(\log P_\theta)$$

given the data sample $\{x_i\}_{i=1,\dots,D}$ we can approximate the integral via Monte-carlo:

$$\boxed{\mathcal{L} = -\frac{1}{D}\sum_{i=1}^{D} \log P_\theta(x_i)}$$

MAXIMUM LIKELIHOOD ESTIMATION!

$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{D} \log P_\theta(x_i)$ by solving with gradient descent.

- Notice that NF allow for the exact computation of the likelihood of the data.

- computing $\mathcal{L}$ requires computing the log-determinant of the jacobian

$$\mathcal{L} = -\frac{1}{D} \sum_{i=1}^{D} \left\{ \log q(f_\theta(x_i)) - \log \left| \mathrm{Det} \frac{\partial f_\theta}{\partial x_i} \right| \right\}$$

computational efficiency of the Jacobian is $\mathcal{O}(d^3)$.

main challenge is to reduce the complexity by modelling f_θ appropriately.

III. NF in the wild:

$$f_\theta = g^1_{\theta_1} \circ g^2_{\theta_2} \circ \cdots \circ g^N_{\theta_N} \quad , \quad \begin{cases} z_n = g_{\theta_n}(z_{n-1}) \\ z_{n-1} = g^{-1}_{\theta_n}(z_n) \end{cases} \quad \forall n = 1, \dots, N$$

Requirements:

1) The base distribution should be easy to sample: Gaussian, Uniform...

2) $(g^n_\theta)^{-1}$ needs to be easy to compute!

3) sufficiently expressive g^n_θ and enough "depth" $N \sim \mathcal{O}(10)$ layers.

4) efficient calculation of the Jacobian
complexity better than $\mathcal{O}(d^3)$ is necessary! ideally $\mathcal{O}(d)$

1) Affine flows: $g_\theta(z) = Az + b$ $\begin{cases} A \in \mathbb{R}^{d \times d} \text{ invertible matrix} \\ b \in \mathbb{R}^d \end{cases}$

- Not expressive enough, e.g. $\mathcal{N}(0, \mathbb{1}) \xrightarrow[g]{} \mathcal{N}(b, A^T A)$
- complexity of $\mathcal{O}(d^3)$ for Jacobian.

$A = \text{diag}(A_1, \dots, A_d) \Rightarrow \mathcal{O}(d)$ but then no correlations between dimensions.

- Affine maps can be used as a building block for more expressive flows.

$\Rightarrow$ Permutation layers $(J = 1)$ and normalization layer (diagonal A).

2) Planar flows:

$$g_\theta(z) = z + v \cdot h(w^T z + b)$$

$\begin{cases} v, w \in \mathbb{R}^D \\ b \in \mathbb{R} \\ h(\cdot) \text{ non-linear activation function} \end{cases}$

- Analytical Jacobian:

$$\text{Det}(J) = 1 + h'(w^T z + b) \cdot w^T v$$

- complexity of $\mathcal{O}(d)$ for the Jacobian
- the inverse g^{-1} exists for special values of parameters

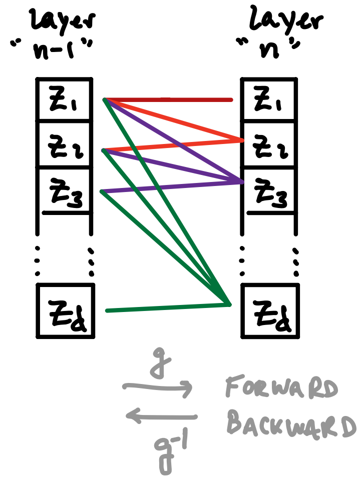
$\Rightarrow$ DIFFICULT TO COMPUTE INVERSE (NOT ANALYTICAL)

3) Autoregressive Flows: (Kingma, et al, 2016)

$x = (x_1, \dots, x_d) \in \mathbb{R}^d$

Each dimension is transformed conditioned on the previous dimension:

$$z' = g(z): \begin{cases} z'_1 = g_1(z_1) \\ z'_2 = g_2(z_1, z_2) \\ z'_3 = g_3(z_1, z_2, z_3) \\ \vdots \qquad \vdots \\ z'_d = g_d(z_1, \dots, z_d) \end{cases}$$



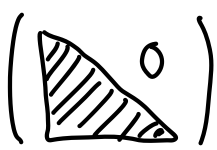
- **Masked Autoregressive flow:**

$$\boxed{z'_i = g(z_i, C^i_\theta(z_1, \dots, z_{i-1}))}$$

conditioner Neural Network

MADE (Masked Autoencoder dist. estimation) Germain et al 2015'

↳ special architecture that enforces autoregressive structure with "binary mask" matrices.

- As long as $g(\cdot)$ is invertible the flow is invertible $z_i = g^{-1}(z'_i, C^i_\theta)$
- These flows have triangular Jacobian matrices! $J =$ 

$$\log|\mathrm{Det}\, J| = \log \prod_{i=1}^{d} \left| \frac{\partial g(z_i, c^i_\theta)}{\partial z_i} \right| = \sum_{i=1}^{d} \left| \frac{\partial g(z_i, c^i_\theta)}{\partial z_i} \right|$$

$\mathcal{O}(d)$ complexity!

- Notice that by construction these models are sensitive to the order of $x = \{x_1, \dots, x_d\}$! the flow requires permutation of the entries between layers in order to increase expressivity

- Affine transformations:

- g can be any invertible transformation: e.g. Affine map

$$z_i' = g(z_i, c_\theta^i) = \underbrace{\exp\{\alpha_\theta^i(z_1 \cdots, z_{i-1})\}}_{\text{scaling}} z_i + \underbrace{\mu_\theta^i(z_1, \dots, z_{i-1})}_{\text{translation}}$$

- $\alpha_\theta^i, \mu_\theta^i \rightarrow$ Neural nets

- $\exp(\alpha_\theta^i)$ guarantees that scale $\neq 0 \ \forall \theta \Rightarrow$ map is invertible

$$\begin{cases} \text{inverse: } z_i = g^{-1}(z_i', c_\theta^i) = \exp\{-\alpha_\theta^i\} \cdot [z_i' - \mu_\theta^i] \\ \text{Jacobian: } \log|\mathrm{Det}(J_g)| = \sum_{i=1}^{d} \alpha_\theta^i \end{cases}$$

NO NEED TO INVERT THE NN!

- the forward pass, $z \xrightarrow{g} z'$ is fast since each dimension can be parallelized (one pass).
- the backward pass, $z' \xrightarrow{g^{-1}} z$, slower by factor $O(d)$.

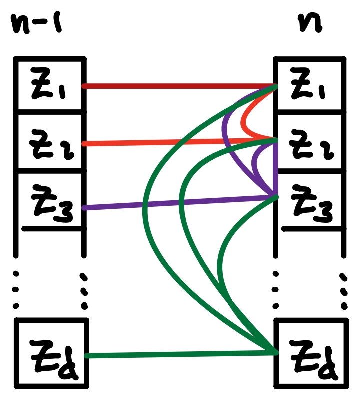
$$z_i = g^{-1}(z_i', c_\theta^i)$$

the inverse z_i requires computing $z_1 \cdots z_{i-1}$ beforehand!

$$\boxed{\begin{array}{l} \Rightarrow \text{Density estimation is } \underline{\text{fast}} \\ \Rightarrow \text{Sampling } \underline{\text{slow}} \end{array}}$$

• Inverse Autoregressive flow (IAF):

$$z' = g(z) : \begin{cases} z'_1 = g_1(z_1) \\ z'_2 = g_2(z'_1, z_2) \\ z'_3 = g_3(z'_1, z'_2, z_3) \\ \vdots \qquad \vdots \\ z'_d = g_d(z_1, \dots, z_d) \end{cases}$$



• IAF is similar to MAF with $z \leftrightarrow z'$, $g \leftrightarrow g^{-1}$.

$$z'_i = g(z_i, C^i_\theta(z'_1, \dots, z'_{i-1}))$$

$\Rightarrow$ Density estimation is slow

$\Rightarrow$ Sampling fast

• Density distillation method (teacher-student training)

For fast sampling, we can train IAF to approximate a pre-trained MAF

$\begin{cases} \text{MAF} \leftarrow \text{"teacher" already trained...} \\ \text{IAF} \leftarrow \text{"student" trained on output of teacher} \end{cases}$

$$\begin{cases} \underset{\text{data}}{x} \xrightarrow{\text{MAF}} \underset{\text{latent}}{z} \xrightarrow{\text{IAF}} \underset{\text{data}}{x'} \\ z \xrightarrow{\text{IAF}} x \xrightarrow{\text{MAF}} z' \end{cases}$$

$\Leftarrow$ train IAF with

$$\mathcal{L}_{MSE} = \| x - x' \|^2 + \| z - z' \|$$

IAF distilled from MAF!
used for calorimeter fast simulation

4) Coupling flows [Dinh et al. 2015, 2017] REALNVP

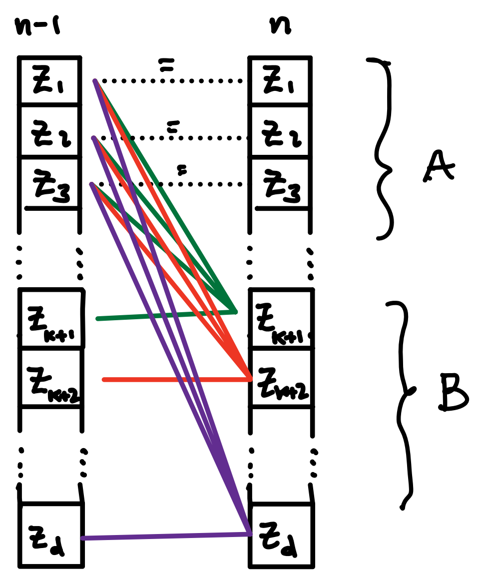
Autoregressive flows have an asymmetry in computational time

$\Rightarrow$ $\mathcal{O}(d)$ times slower in sampling (MAF) or evaluating the density (IAF).

- this can be solved by changing the AF setup in the following way:

$\Rightarrow$ partition the dimensions into two sets A and B:

$$z = (z_A, z_B) \; : \; \begin{cases} z_A \in \mathbb{R}^K \\ z_B \in \mathbb{R}^{d-K} \end{cases}$$



$$\boxed{\begin{cases} z'_A = z_A & \text{i.e. identity transform.} \\ z'_B = g(z_B, c_\theta(z_A)) \end{cases}}$$

in components: $\begin{cases} z'_i = z_i & \text{for } i \leq K \\ z'_i = g(z_i, c^i_\theta(z_1, \dots, z_K)) & \text{for } i > K \end{cases}$

- inverse flow is straightforward:

$$\begin{cases} z_i = z'_i & (i \leq K) \\ z_i = g^{-1}(z'_i, c^i_\theta(z_1, \dots, z_K)) & (i > K) \end{cases}$$

Notice that, unlike MAF, here computing the inverse does not require iterating $\Rightarrow$ forward and backward passes have the same complexity! (one pass)

- Jacobian is triangular:

$$J = \left(\begin{array}{c:c} \mathbb{1} & 0 \\ \hdashline \underbrace{\blacksquare}_{(d-k)\times k \text{ matrix}} & \text{diag} \end{array}\right) \Rightarrow \log|\text{Det } J_g| = \sum_{i=k}^{d-k} \left| \frac{\partial g(z_i, c_\theta^i)}{\partial z_i} \right|$$

- A single coupling flow layer will allways leaves unchanged data components (the "A" part). We therefore need to stack several layers and permute in between to transform all the data.