I. Normalizing flows

REF: Papamakarios, et al JMLR 1912.02762

**Core Idea:** Map a "simple" distribution into a "complex" one using the change-of-variables formula

\[ X = f(z) \quad f : \mathbb{R}^d \to \mathbb{R}^d \]

- \( f \) smooth and invertible map (bijection)
  \[ z = f^{-1}(x) \]

The map above induces a **change in densities** given by:

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

- **Log-likelihood:**
  \[ \log P(x) = \log q(z) - \log \left| \det \frac{\partial f}{\partial z} \right| \]

- \( q \sim N(0, I) \) 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.

\[ f = f^N \circ f^{N-1} \circ \cdots \circ f \] "flow of maps"

\[ z = z^1 \xrightarrow{f^1} z^2 \xrightarrow{f^2} z^3 \xrightarrow{f^3} \cdots \xrightarrow{f^n} z^N = x \] allows for more expressive \( f \).

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

log-likelihood:

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

Remarks:

i) only works for distributions with continuous random variables

ii) flow is "finite" \( q_1, \ldots, q_N \) "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 $f_\theta$ between a latent space $z \sim q(z) = \mathcal{N}(z; 0, I)$ and the data $x \sim P_{data}(x)$, where $\theta$ are learnable parameters.

$$f_\theta = f_\theta^1 \circ \ldots \circ f_\theta^N$$ parametrized by deep NN with "N" layers

Once $f_\theta$ is learned $\Rightarrow$

1. We can generate synthetic data! (generative model)
2. We have an approximation of $P_{data}$! (density estimation)

- Ancestral sampling:
  - i. $z_0 \sim \mathcal{N}(0, I)$
  - ii. $x_{new} = f_\theta(z_0)$

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

the goal is to approximate the target distribution $P_{data}(x)$ with the "push-forward" $P_\theta(x)$ resulting from the NF model.

$$\Rightarrow \text{minimize the KL-divergence } D_{KL}(P_{data} \parallel P_\theta) = \int dx \ P_{data} \log \frac{P_{data}}{P_\theta}$$

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

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

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

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

Maximum Likelihood Estimation!

$$\hat{\theta} = \arg\max_\theta \sum_{i=1}^{D} \log P_\theta(x_i) \text{ by solving with gradient descent.}$$
Notice that NF allow for the exact computation of the likelihood of the data.

Computing $L$ requires computing the log-determinant of the Jacobian

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

Computational efficiency of the Jacobian is $O(d^3)$.

The main challenge is to reduce the complexity by modelling $f_\theta$ appropriately.

### III. NF in the wild:

$$ f_\theta = g^1_\theta \circ g^2_\theta \circ \ldots \circ g^N_\theta $$

$$ \left\{ \begin{array}{l} Z^n = g^n_\theta(Z^{n-1}) \\ Z^0 = \mathbf{z} \end{array} \right\} \quad \forall n = 1, \ldots, N $$

**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 $g^N_\theta$ and enough "depth" $N \sim O(10)$ layers.
4. Efficient calculation of the Jacobian complexity better than $O(d^3)$ is necessary! Ideally $O(d)$
I. Affine Flows: 
\[ g_\theta(z) = Az + b \quad \theta \in \mathbb{R}^{d \times d} \text{ invertible matrix} \]

- Not expressive enough, e.g. \( N(0, I) \xrightarrow{g} N(b, A^TA) \)
- Complexity of \( \Theta(d^3) \) for Jacobian.

\[ A = \text{diag}(A_1, \ldots, A_d) \Rightarrow \Theta(d) \]

- Affine maps can be used as a building block for more expressive flows.
  - Permutation layers (J=1) and normalization layer (diagonal A).

II. Planar Flows:
\[ g_\theta(z) = z + v \cdot h(W^Tz + b) \]

- Analytical Jacobian:
  \[ \text{Det}(J) = 1 + h'(W^Tz + b) \cdot W^Tv \]

- Complexity of \( \Theta(d) \) for the Jacobian
- The inverse \( g^{-1} \) exists for special values of parameters

\[ \Rightarrow \text{Difficult to compute inverse (not analytical)} \]
3) **Autoregressive flows**: (Kingma et al. 2016)

\[ x = (x_1, \ldots, x_d) \in \mathbb{R}^d \]

Each dimension is transformed conditioned on the previous dimension:

\[
\begin{align*}
    z'_1 &= g_1(z_1) \\
    z'_2 &= g_2(z_1, z_2) \\
    z'_3 &= g_3(z_1, z_2, z_3) \\
    &\vdots \\
    z'_d &= g_d(z_1, \ldots, z_d)
\end{align*}
\]

- **Masked Autoregressive flow**:

\[
\begin{bmatrix}
    z'_1 \\
    z'_2 \\
    \vdots \\
    z'_d
\end{bmatrix} = g\left(z_i, c^i_{\theta}(z_1, \ldots, z_{i-1})\right)
\]

MADE (Masked Autoregressive dist. estimation) Germain et al 2015,

is 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! \( J = \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} \)

\[
\log |\text{Det } J| = \log \prod_{i=1}^{d} \left| \frac{\partial \hat{g}(z_i, c^i_{\theta})}{\partial z_i} \right| = \sum_{i=1}^{d} \left| \frac{\partial \hat{g}(z_i, c^i_{\theta})}{\partial z_i} \right| \quad \mathcal{O}(d) \text{ complexity!}
\]

- Notice that by construction these models are sensitive to the order of \( x = (x_1, \ldots, x_d) \) ! 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

\[
Z'_\lambda = g(Z_\lambda, C_\theta^\lambda) = \exp\left\{\alpha^\lambda_\theta(Z_1, \ldots, Z_{\lambda-1})\right\} Z_\lambda + \mu^\lambda_\theta(Z_1, \ldots, Z_{\lambda-1})
\]

  \[\text{scaling}\quad \text{translation}\]

- $\alpha^\lambda_\theta, \mu^\lambda_\theta \rightarrow \text{neural nets}$

- $\exp(\alpha^\lambda_\theta)$ guarantees that scale $\neq 0 \forall \theta \Rightarrow \text{map is invertible}$

\[
\begin{cases}
\text{inverse: } Z_\lambda = g^{-1}(Z'_\lambda, C_\theta^\lambda) = \exp\{-\alpha^\lambda_\theta\} \cdot [Z'_\lambda - \mu^\lambda_\theta] \\
\text{Jacobian: } \log|\text{Det}(J_g)| = \sum_{\lambda=1}^d \alpha^\lambda_\theta \quad \text{NO NEED TO INVERT THE NN!}
\end{cases}
\]

- The forward pass, $Z \overset{g}{\rightarrow} Z'$ is fast since each dimension can be parallelized (one pass).

- The backward pass, $Z' \overset{g^{-1}}{\rightarrow} Z$, slower by factor $\Theta(d)$.

  \[
  Z_\lambda = g^{-1}(Z'_\lambda, C_\theta^\lambda) \quad \text{the inverse } Z_\lambda \text{ requires computing } Z_1 \ldots Z_{\lambda-1} \text{ beforehand!}
  \]

\[
\Rightarrow \text{Density estimation is fast}
\]

\[
\Rightarrow \text{Sampling slow}
\]
Inverse Autoregressive flow (IAF):

\[
\begin{align*}
    z'_1 &= g_1(z_1) \\
    z'_2 &= g_2(z'_1, z_2) \\
    z'_3 &= g_3(z'_1, z'_2, z_3) \\
        & \vdots \quad \vdots \\
    z'_d &= g_d(z_1, \ldots, z_d)
\end{align*}
\]

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

\[
    z'_{\text{IAF}} = g(z, \text{C}_0(z', \ldots, z'_{n-1}))
\]

\[ \Rightarrow \text{Density estimation is slow} \]
\[ \Rightarrow \text{Sampling fast} \]

Density distillation method (teacher-student training)

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

\[
\begin{align*}
    \text{MAF} & \leftarrow \text{"teacher" already trained} \ldots \\
    \text{IAF} & \leftarrow \text{"student" trained on output of teacher}
\end{align*}
\]

\[
\begin{align*}
    \text{X} & \rightarrow \text{Z} & \text{IAF} & \rightarrow \text{X}' \\
\text{data} & & \text{data} & \\
\end{align*}
\]

\[
\begin{align*}
    \text{Z} & \rightarrow \text{IAF} & \rightarrow \text{MAF} \\
\end{align*}
\]

Train IAF with

\[
\mathcal{L}_\text{MSE} = \mathcal{L}_X + \mathcal{L}_Z 
\]

IAF distilled from MAF's
used for calorimeter fast simulation.
4) **Coupling flows** (Dinh et al. 2015, 2017) REALNVP

Autoregressive flows have an asymmetry in computational time

$\Rightarrow O(d)$ 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:

$z = (z_A, z_B) \in \mathbb{R}^{d-k}$

\[
\begin{align*}
  z_A' &= z_A \quad \text{i.e. identity transform}, \\
  z_B' &= g(z_B, c_0(z_A))
\end{align*}
\]

In components:

\[
\begin{align*}
  z_i' &= z_i \quad \text{for } i \leq k \\
  z_i' &= g(z_i, c_0(z_1, \ldots, z_k)) \quad \text{for } i > k
\end{align*}
\]

- Inverse flow is straightforward:

\[
\begin{align*}
  z_i &= z_i' \quad (i \leq k) \\
  z_i &= g^{-1}(z_i', c_0(z_1, \ldots, z_k)) \quad (i > k)
\end{align*}
\]
Notice that, unlike MAF, here computing the inverse does not require iterating ⇒ forward and backward passes leave the same complexity! (one pass)

- Jacobian is triangular:

\[
J = \begin{pmatrix}
1 & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & 1
\end{pmatrix} \Rightarrow \log |\det J| = \sum_{i=k}^{d-k} \left| \frac{\partial g(z_i, c_i)}{\partial z_i} \right|
\]

\((d-k) \times k\) matrix

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