Flow models

Artem Ryzhikov[1]

[1]National Research University Higher School of Economics

November 24, 2020

LAMBDA • HSE

Motivation

Generative models. Overview

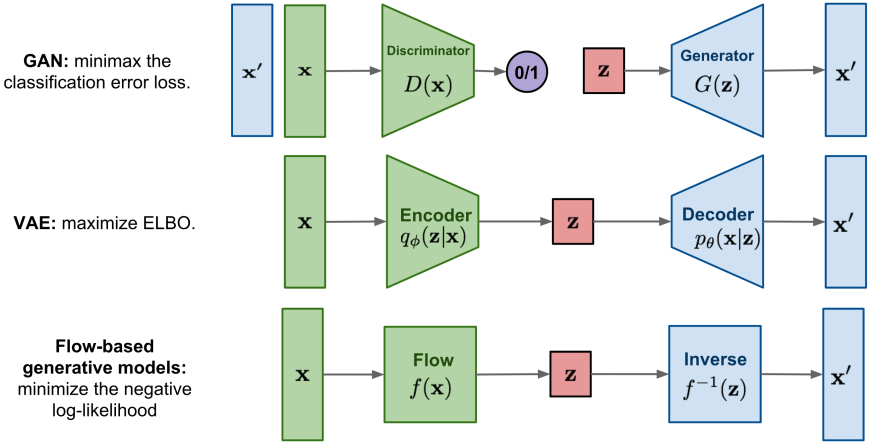


Figure: https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Normalizing flows

Change of variable theorem

Univariate case:

$$\int p(x)\,dx = \int \pi(z)\,dz = 1 \text{ ; Definition of probability distribution.}$$

$$p(x) = \pi(z)\left|\frac{dz}{dx}\right| = \pi(f^{-1}(x))\left|\frac{df^{-1}}{dx}\right| = \pi(f^{-1}(x))|(f^{-1})'(x)|$$

Multivariate case:

$$\mathbf{z} \sim \pi(\mathbf{z}), \mathbf{x} = f(\mathbf{z}), \mathbf{z} = f^{-1}(\mathbf{x})$$

$$p(\mathbf{x}) = \pi(\mathbf{z})\left|\det \frac{d\mathbf{z}}{d\mathbf{x}}\right| = \pi(f^{-1}(\mathbf{x}))\left|\det \frac{df^{-1}}{d\mathbf{x}}\right| = \frac{\pi(f^{-1}(\mathbf{x}))}{|\det J(x)|}$$

$z \sim \pi(z)$ is initial random variable, sampled from distribution $\pi(\cdot)$

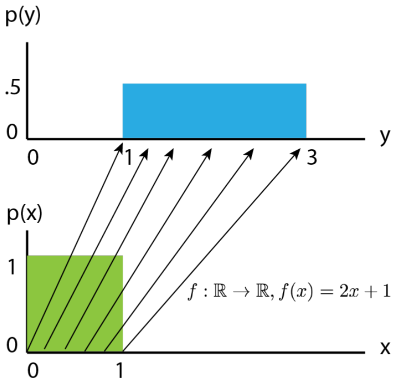
$x = f(z) \sim p(x)$ is transformed random variable, sampled from distribution $p(\cdot)$ ($f(\cdot)$ is transformation)

$J(\cdot)$ is *Jacobian*

Jacobian. Example

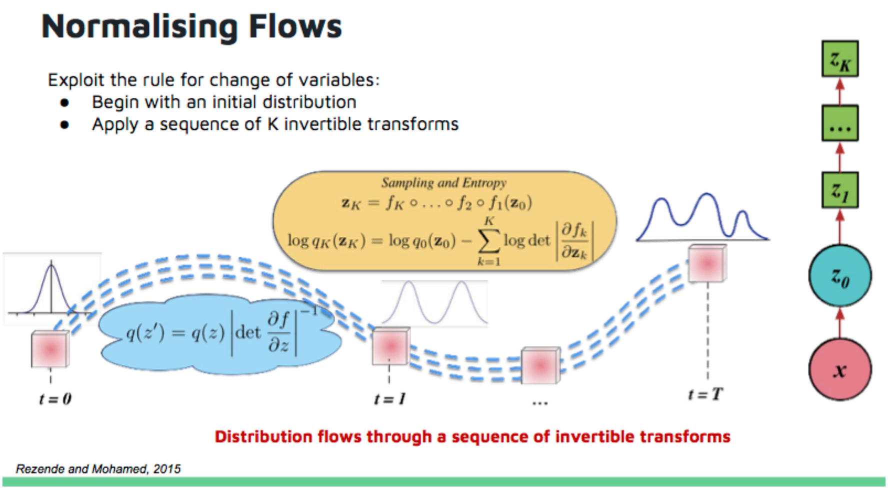
$$
\mathrm{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}
$$

$$
p(\mathbf{y}) = \frac{\pi(f^{-1}(\mathbf{y}))}{|\det J(y)|} = \frac{p(\mathbf{x})}{|\det J(y)|}
$$



$J(y) \equiv 2$. Final probability density is 2 times smaller

Normalizing flow



Normalizing flow. Problem

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\det M = \det \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \sum_{j_1 j_2 \dots j_n} (-1)^{\tau(j_1 j_2 \dots j_n)} a_{1j_1} a_{2j_2} \dots a_{nj_n}$$

Problem: hard to compute $\det J$ (it takes $O(n^3)$ time to compute, where n is dimensionality)

Solution: use a special family of transformations

Simple flows

- **Planar flow**:

$$f(z) = z + uh(w^T z + b)$$

$$|det\frac{\partial f}{\partial z}| = |1 + u^T h'(w^T z + b)w|$$

where $\mathbf{u}, \mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$ and h is an element-wise non-linearity such as tanh.

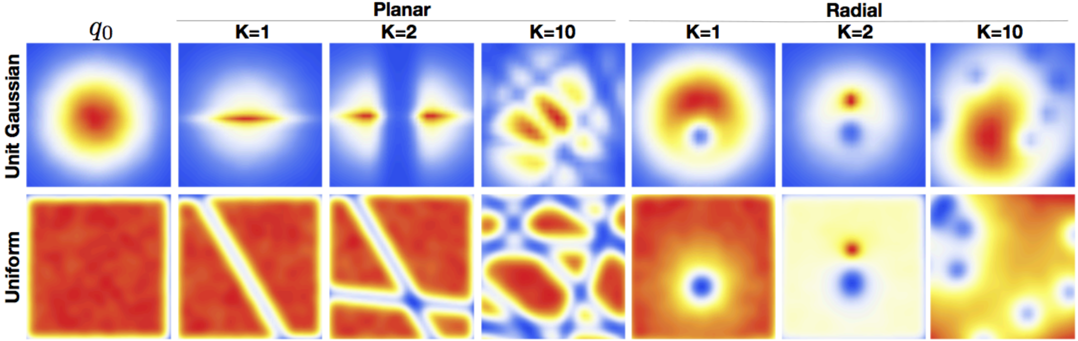
- **Radial flow**:

$$f(z) = z + \beta h(\alpha, r)(z - z_0)$$

$$\det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} = (1 + \beta h(\alpha, r) + \beta h'(\alpha, r)r)\,(1 + \beta h(\alpha, r))^{d-1}$$

where $\alpha \in \mathbb{R}^+$, $\beta \in \mathbb{R}$, $h(\alpha, r) = (\alpha + r)^{-1}$ and $r = ||\mathbf{z} - \mathbf{z}_0||$.

Simple flows



q_0 is initial (domain) distributuion. K denotes K'th transformed distribution

Autoregressive flows

Autoregressive flows. Real NVP

$$
\mathrm{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}
$$

Idea make the Jacobian triangular:

$$
y_i = f(z_{1:i})
$$

$$
detJ = \Pi_{i=1}^{d} J_{ii}
$$

Real NVP (Real Non-Volume Preserving NF, R-NVP)

$$
y_{1:k} = z_{1:k}
$$

$$
y_{k+1:d} = z_{k+1:d} \odot \sigma(z_{1:k}) + \mu(z_{1:k})
$$

$$
\frac{\partial y}{\partial z} = \Pi_{i=1}^{d-k} \sigma_i(z_{1:k})
$$

Problem: low generalization power

Autoregressive transformation. MAF

Idea: How to introduce complex dependencies between dimensions?

$$y_1 = \mu_1 + \sigma_1 z_1$$

$$y_i = \mu(y_{1:i-1}) + \sigma(y_{1:i-1}) z_i$$

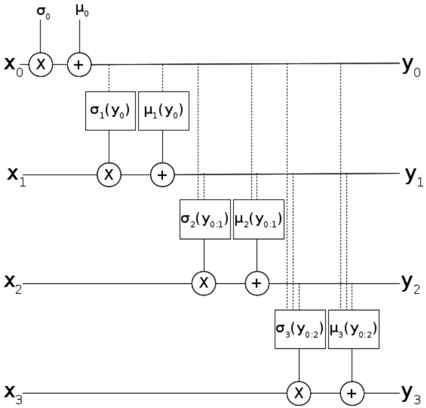
$$z_i = \frac{y_i - \mu(y_{1:i-1})}{\sigma(y_{1:i-1})}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ are arbitrary neural networks.

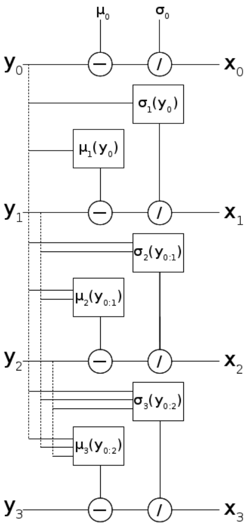
Problem: hard to sample (sampling is unscalable - y components must be sequentially processed)

Use case: Use Masked Autoregressive Flow (MAF) as VAE's prior (arXiv:1809.05861)

MAF



σ_0 μ_0
x_0 y_0
$\sigma_1(y_0)$ $\mu_1(y_0)$
x_1 y_1
$\sigma_2(y_{0:1})$ $\mu_2(y_{0:1})$
x_2 y_2
$\sigma_3(y_{0:2})$ $\mu_3(y_{0:2})$
x_3 y_3



μ_0 σ_0
y_0 x_0
$\sigma_1(y_0)$
$\mu_1(y_0)$
y_1 x_1
$\sigma_2(y_{0:1})$
$\mu_2(y_{0:1})$
y_2 x_2
$\sigma_3(y_{0:2})$
$\mu_3(y_{0:2})$
y_3 x_3

Inverse Autoregressive Flow (IAF)

Idea: Use reparametrization to make sampling easier

$$y_i = z_i \sigma(z_{1,i-1}) + \mu(z_{1:i-1})$$

$$z_{k-1,1} = \frac{z_{k,1} - \mu_{k,1}}{\sigma_{k,1}}$$

$$z_{k-1,i} = \frac{z_{k,i} - \mu_{k,i}(z_{k-1,1:i-1})}{\sigma_{k,i(z_{k-1,1:i-1})}}$$

Problem: hard to estimate likelihood. Hence, hard to train!

Advanced flow models

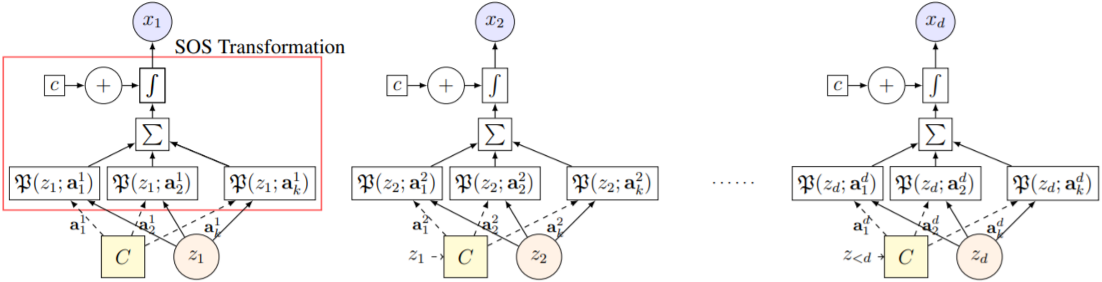
Sum-of-Squares polynomial flow (SOS)

Theorem 1 (inverse function) Let f be a strictly monotone continuous function on $[a, b]$ with f differentiable at $x_0 \in (a, b)$ and $f'(x_0) \neq 0$. Then f^{-1} exists and is continuous and strictly monotone.

Theorem 2 All positive univariate polynomials are the sum of squares of polynomials.

$$\mathfrak{P}_{2r+1}(z; \mathbf{a}) = c + \int_0^z \sum_{\kappa=1}^{k} \left(\sum_{l=0}^{r} a_{l,\kappa} u^l \right)^2 \mathrm{d}u$$

Sum-of-Squares polynomial flow (SOS)



$$\mathfrak{P}_{2r+1}(z;\mathbf{a}) = c + \int_0^z \sum_{\kappa=1}^{k} \left(\sum_{l=0}^{r} a_{l,\kappa} u^l \right)^2 du$$

arXiv:1905.02325

Residual Flows

Let $y = f(x) = x + g(x)$ and g is 1-Lipsitz $(|g'(\cdot)| < 1)$. Then

$$\log p(x) = \log p(f(x)) + \mathrm{tr}\left(\Sigma_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} [J_g(x)]^k\right)$$

where J_g is a Jacobian of g.

Figure 5: **Qualitative samples.** Real (left) and random samples (right) from a model trained on 5bit 64×64 CelebA. The most visually appealing samples were picked out of 5 random batches.

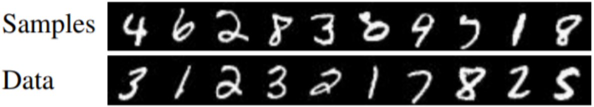
arXiv:1906.02735

Infinitesimal (Continuous) Flows. FFJORD

Residual Flows (previous slide): $\Delta x = g(x)$

Infinitesimal (Continuous) Flows: $\frac{d}{dt}x(t) = f(x(t), t; \theta)$

$$\log p(z(t_1)) = \log p(z(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\Big(\frac{\partial f}{\partial z(t)}\Big)$$



arXiv:1810.01367

Normalizing flows. Summary 1

Normalizing flow is generative model which ...

- ... fits bijection between simple (like gaussian or uniform) and complex (training data) distributions
- ... provides likelihood estimation for objects
- ... has consistent and clear training scheme (likelihood maximization)

... but which is ...

- ... still more limited than other generative models due to strict limitations on transformations (to ensure fast Jacobian computation)

Normalizing flows. Summary 2

	Simple Flows	R-NVP	MAF	IAF	Residual Flow	FFJORD	SOS
Generalization power	**Low**	**Good**	**Good+**	**Good+**	**High**	**High**	**High**
Likelihood estimation/Training	**Fast**	**Fast**	**Fast**	**Slow**	**Fast**	**Fast**	**Slow**
Sampling	**Fast**	**Fast**	**Slow**	**Fast**	**Fast**	**Fast**	**Fast**
Universality	**No**	**No**	**No**	**No**	**No**	**?**	**Yes**
Free-form Jacobian	**No**	**No**	**No**	**No**	**No**	**Yes**	**?**

* **Low** generalization power is caused by lack of correlation between components

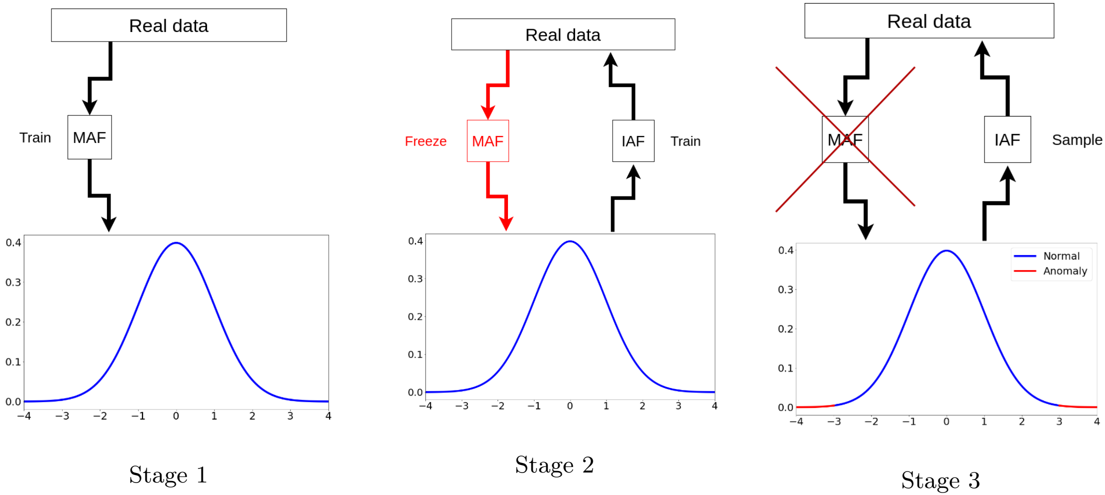
* **Slow** means relatively slow computational speed due to non-scalable transformation (all components in the corresponding mode can processed in sequential order only)

Universality means that any pdf can be fitted

Question: is it possible to make NF equally fast in both directions?

Answer: Yes! Probability distillation pdist

Probability distillation*



Thank you for your attention!

Artem Ryzhikov aryzhikov@hse.ru