Normalizing Flows

David Coufal
Institute of Computer Science
The Czech Academy of Sciences
david.coufal@cs.cas.cz

Vision for Robotics - FEL CTU
December 5, 2022

Introduction

- data $D = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$ comes from distribution P_D
  i.e., we assume that there exists a random variable D with values in $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$ such that $D \sim P_D$

- How to specify P_D on the basis of D?

Introduction

- specification of cdf is possible, but the most common approach is to specify a density $p_D : \mathbb{R}^d \to [0, \infty)$ of P_D

$$P_D(A) = \int_A p_D(\boldsymbol{x})\, d\boldsymbol{x} \qquad \text{for } A \in \mathcal{B}(\mathbb{R}^d)$$

- How to get the density from empirical data?

Introduction

- if $p_D \in \{p_\theta, \theta \in \Theta\}$ (a parametric set of densities) task reduces to estimate best parameter θ^* from data $D = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$ and set $p_D = p_{\theta^*}$

- maximum likelihood estimation

$$\theta_{\text{mle}} = \text{argmax}_\theta \, \mathbb{E}_{\boldsymbol{x} \sim P_D} \log p_\theta(\boldsymbol{x})$$

$$\theta^*_{\text{mle}} = \text{argmax}_\theta \, \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(\boldsymbol{x}_i)$$

Introduction

- maximum likelihood estimation

$$\theta_{\mathsf{mle}} = \mathsf{argmax}_{\theta}\, \mathbb{E}_{\boldsymbol{x}\sim P_D}\ \mathsf{log}\, p_{\theta}(\boldsymbol{x})$$

$$\theta^*_{\mathsf{mle}} = \mathsf{argmax}_{\theta}\, \frac{1}{n}\sum_{i=1}^{n} \mathsf{log}\, p_{\theta}(\boldsymbol{x}_i)$$

- optimization in terms of KL-divergence

$$\begin{aligned}\theta_{\mathsf{mle}} &= \mathsf{argmin}_{\theta}\, D_{\mathsf{KL}}(P_D(\boldsymbol{x})||P_{\theta}(\boldsymbol{x})) \\ &= \mathsf{argmin}_{\theta}\, \int p_D(\boldsymbol{x})\frac{p_D(\boldsymbol{x})}{p_{\theta}(\boldsymbol{x})}\, d\boldsymbol{x}\end{aligned}$$

MLE in terms of KL-divergence

- best approximation of P_D using P_θ
  - $\widehat{P}_D$ proxy for P_D, $\widehat{P}_D(d\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta_{\boldsymbol{x}_i}(d\boldsymbol{x})$ (Dirac m.)
  - P_θ - model distribution with density p_θ
- maximization MLE = minimization of $D_{\mathsf{KL}}(P_D || P_\theta)$

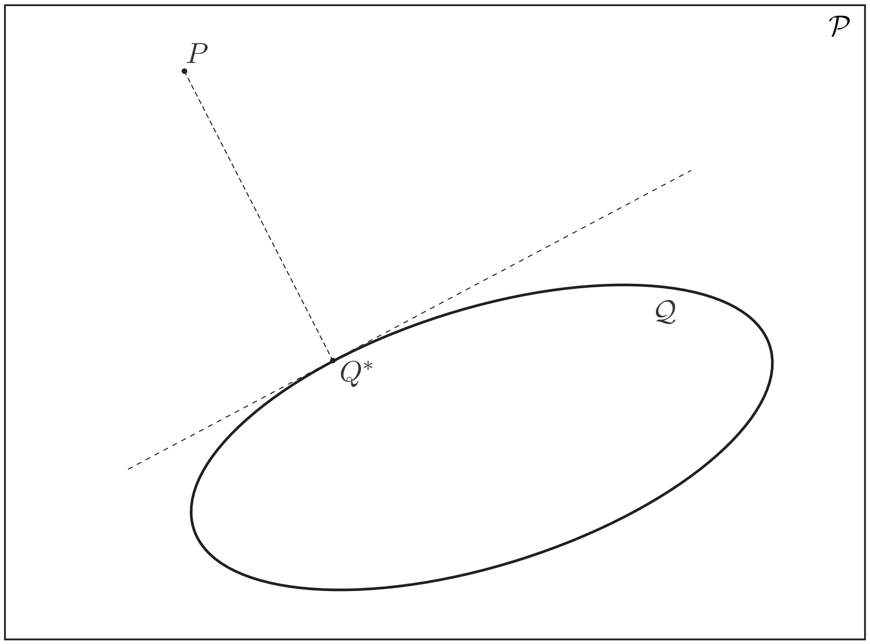
$$\begin{aligned}
D_{\mathsf{KL}}(P_D||P_\theta) &= \int \log \frac{dP_D}{dP_\theta}\, dP_D \;=\; \int \log \frac{p_D(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\, dP_D \\
&= \int \log p_D(\boldsymbol{x})\, dP_D - \int \log p_\theta(\boldsymbol{x})\, dP_D \\
&\approx -H[P_D] - \int \log p_\theta(\boldsymbol{x})\, d\widehat{P}_D \;\; (P_D \approx \widehat{P}_D) \\
&\propto -\int \log p_\theta(\boldsymbol{x})\, d\widehat{P}_D \;\; \text{(integration over Dirac)} \\
&\propto \underbrace{-\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(\boldsymbol{x}_i)}_{=\mathsf{MLE}}
\end{aligned}$$

Information projection

- let $P \in \mathcal{P}$ is fixed, and $\mathcal{Q} \subset \mathcal{P}$ (subset of prob. distributions)

$$Q^* = \text{argmin}_{Q \in \mathcal{Q}} D_{\mathsf{KL}}(P||Q),$$

Q^* is the closest distribution from subset of $\mathcal{Q}$ to P



Specification of $\mathcal{Q} \subset \mathcal{P}$

- via parametrized densities
  i.e., $\mathcal{Q} = \{p_\theta, \theta \in \Theta\}$, optimal parameter θ^*
  identified using MLE, which is a specific solution
  to the information projection problem based on densities

- p_θ^* is used to approximate the real denstity of P_D, i.e,

$$p_{\theta^*} \approx p_D = P_D \, dx$$

- How to sample from a given density/distribution?

Specification of $\mathcal{Q} \subset \mathcal{P}$

- via parametrized transformations
  X has some simple distribution which is easy to sample from and is transformed to a complex one using a deterministic function G
  e.g., let $X \sim N(0,1)$ then $X^2 \sim \chi^2(1)$ and $G(z) = z^2$

- $\mathcal{Q}$ is given by set of parametrized functions G_θ, $\theta \in \Theta$
  (neural networks parametrized via their weights)

- easy sampling from $G_\theta(X)$, sample $\boldsymbol{x} \sim X$ (easy)
  and then pass $\boldsymbol{x}$ through $G_\theta(X)$, i.e., compute $G_\theta(\boldsymbol{x})$

- How to solve the information projection problem
  that is based on transformations?

GANs

- solution to the information projection problem
  JS-divergence minimalization
  via playing an adversial game between
  generator and discriminator

source: https://towardsdatascience.com/generative-adversarial-networks-learning-to-create-8b15709587c9

GANs

- GANs are learn adverisialy to minimize

$$D_{\mathsf{JSD}}(P_D || P_{G_\theta})$$

  by adjusting parameters θ of generator

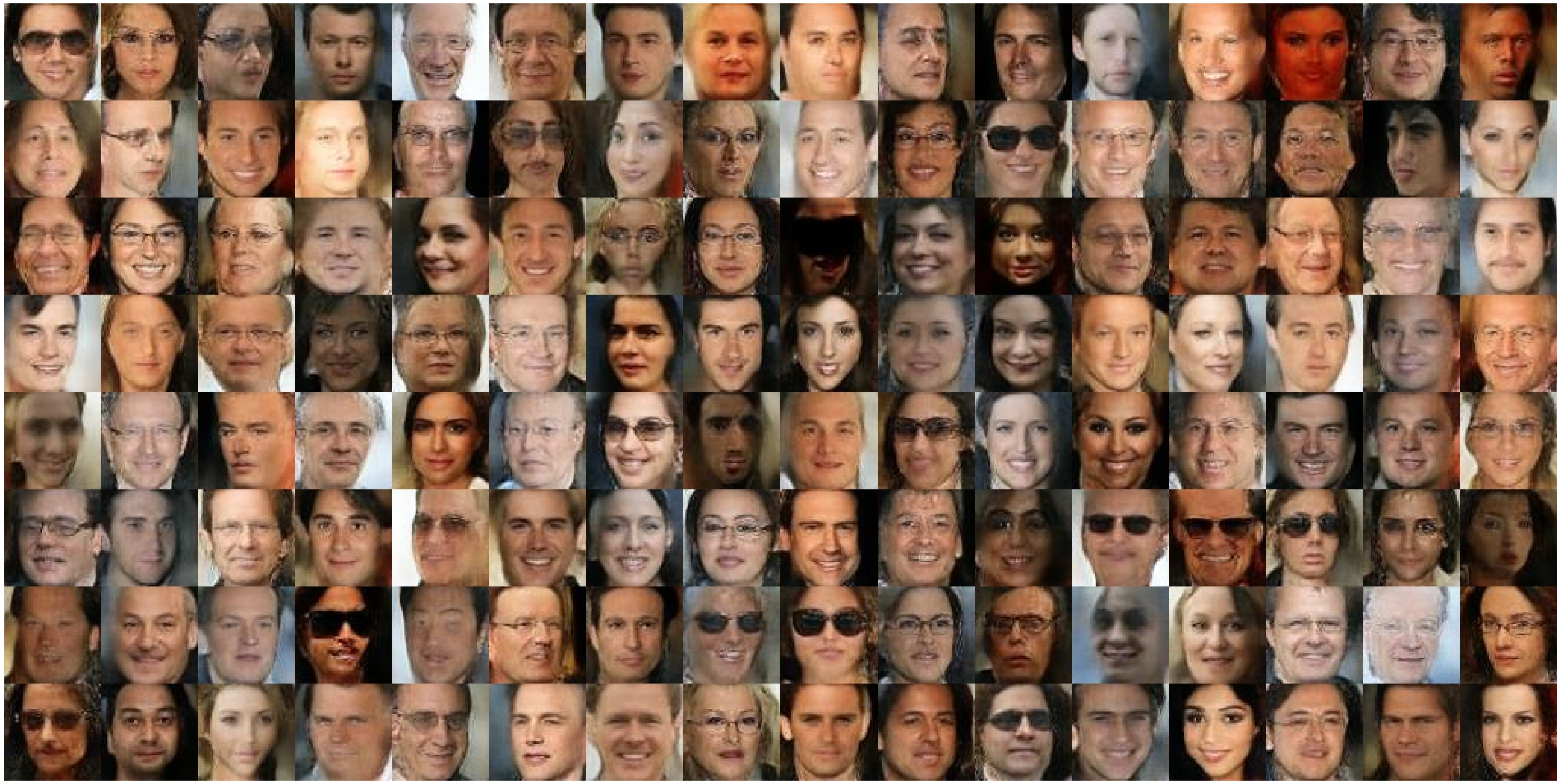
- setting properly adverisial learning is still more of an art than a strictly procedural matter

- there is no straithforward inverse procedure
  to find a latent $\boldsymbol{z}^*$ to the given $\boldsymbol{x}^*$ and directly evaluate

$$p_{G_\theta}(\boldsymbol{x}^*) = p_{G_\theta}(G_\theta(\boldsymbol{z}^*))$$

- or even better, to find a latent $\boldsymbol{z}_{real}$ to a given $\boldsymbol{x}_{real}$
  \- invertibility of the generator

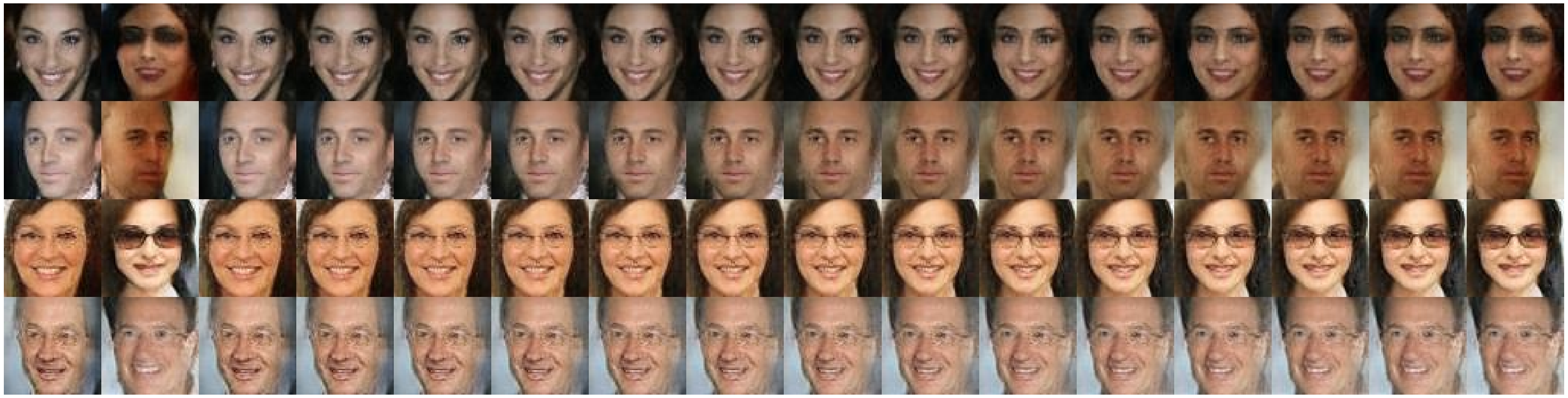
Conditional BEGAN



- each image has its latent $\boldsymbol{z} = (\boldsymbol{z}_{100}, \boldsymbol{c}_2)$
  $\boldsymbol{z}_{100} \in \mathbb{R}^{100}, z_i \sim \mathcal{N}(0,1)$ and $\boldsymbol{c}_2 \in \{-1,1\}^2$

- $\boldsymbol{c}$ encodes man/woman, w/o glasses, image $= G_\theta(\boldsymbol{z})$

Conditional BEGAN



- linear approximation between two latents, z_1, z_2 (condition fixed)

$$z_t = z_1 + t/13 * (z_2 - z_1), \quad t = 0, \ldots, 13$$

- smooth transition

Conditional BEGAN

- different conditions for the same latent z_{100}
- properties manipulation
  FDA approval rate, https://insilico.com

Normalizing flows

- normalizing flows can be treated as invertible neural networks
- based on invertible differentiable bijections, which assures 1-to-1 correspondence, i.e., $\boldsymbol{z} \leftrightarrow \boldsymbol{x}$, and so invertibility
- exact evaluation of generative density

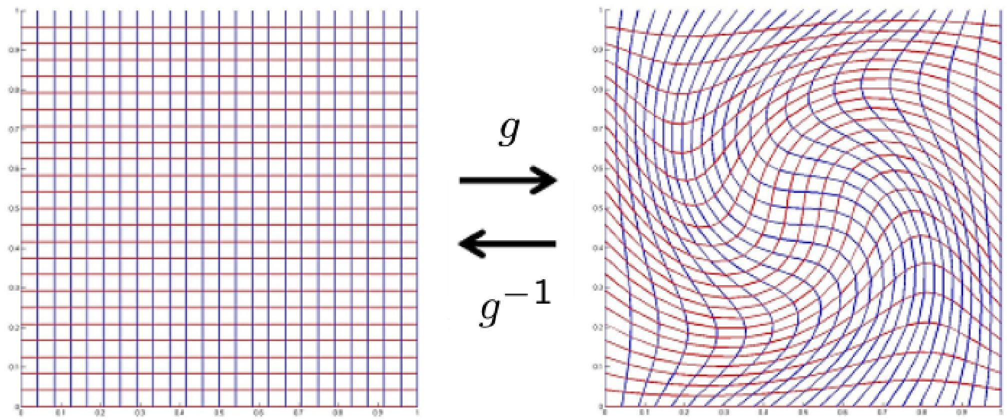
$$p_{G_\theta}(\boldsymbol{x}) = p_{G_\theta}(G_\theta(\boldsymbol{z}))$$

  which allows learning via maximum likelihood estimation

- a couple of tricks to make computation, learning and inversion procedure effective
- still, computationally more demanding than GANs
  less quality results

Diffeomorphism on $\mathbb{R}^d$

- a function $g : \mathbb{R}^d \to \mathbb{R}^d$ is called diffeomorphism if it is bijective, differentiable and has a differentiable inversion g^{-1}



source: https://arxiv.org/abs/1310.1710

- differentiable space deformation

Change of variable formula on $\mathbb{R}^d$

- distribution transformation under diffeomorphism

- let P_Z be a distribution on $\mathbb{R}^d$ with density $p_Z(\boldsymbol{z})$, g diffeomorphism on $\mathbb{R}^d$ and $\boldsymbol{x} = g(\boldsymbol{z})$, i.e., $\boldsymbol{z} = g^{-1}(\boldsymbol{x})$; then $\boldsymbol{x}$ has distribution P_X with density

$$p_X(\boldsymbol{x}) \;=\; p_Z(g^{-1}(\boldsymbol{x})) \cdot |\det(\mathsf{J}_{g^{-1}}(\boldsymbol{x}))|$$

- where $\mathsf{J}_{g^{-1}}$ is the Jacobian of g^{-1} (it is a $d \times d$ functional matrix) at point $\boldsymbol{x} \in \mathbb{R}^d$, $\det(\cdot)$ stands for determinant and $|\cdot|$ is the absolute value

Density transformation on $\mathbb{R}^d$

- $g : \mathbb{R}^d \to \mathbb{R}^d$ diffeomorphism with inversion $f = g^{-1}$

- let $\boldsymbol{x} = g(\boldsymbol{z})$, $\boldsymbol{z} \sim p_Z(\boldsymbol{z})$, then $\boldsymbol{x} \sim p_X(\boldsymbol{x})$ and

$$p_X(\boldsymbol{x}) \;=\; p_Z(f(\boldsymbol{x})) \cdot |\mathrm{det}(\mathsf{J}_f(\boldsymbol{x}))|$$

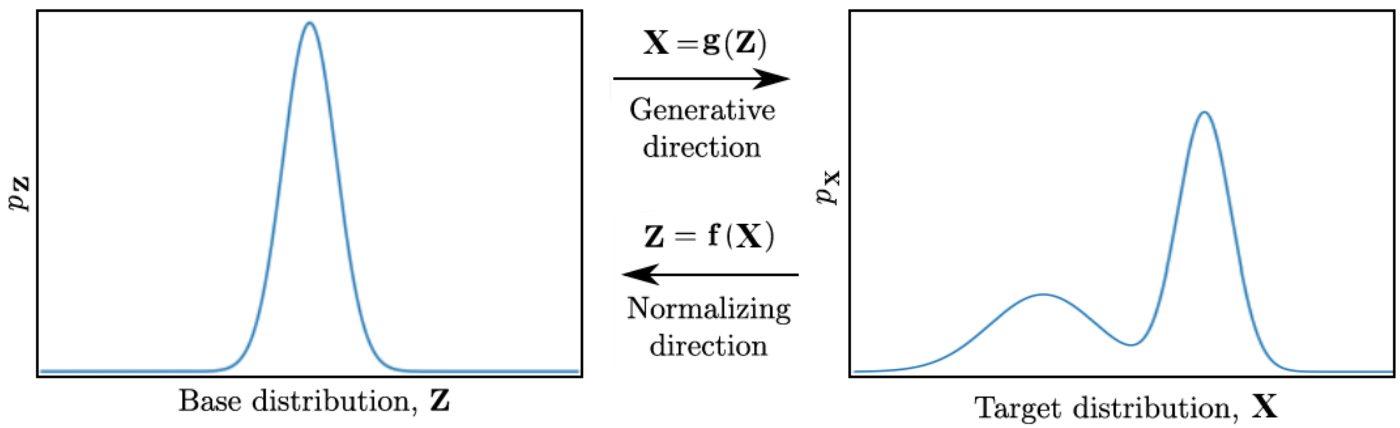
- J_f is the Jacobian of f, i.e., if $f = (f_1(\boldsymbol{x}), \ldots, f_d(\boldsymbol{x}))$, then

$$\mathsf{J}_f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\boldsymbol{x}) & \cdots & \frac{\partial f_d}{\partial x_d}(\boldsymbol{x}) \\ \vdots & \cdots & \vdots \\ \frac{\partial f_d}{\partial x_1}(\boldsymbol{x}) & \cdots & \frac{\partial f_d}{\partial x_d}(\boldsymbol{x}) \end{bmatrix}$$

Terminology

- g direction: generative or forward direction, from easy to a complex distribution

- $f = g^{-1}$ direction: flow or backward direction, from complex to an easy distribution - normalization of the complex distribution, it holds literally when Z has a normal distribution



source: https://arxiv.org/abs/1908.09257

Composite flow

- let $g_1, g_2, \ldots, g_K$ be a set of diffeomorphisms, then

$$g(\boldsymbol{z}) = g_K(g_{K-1}(\ldots(g_1(\boldsymbol{z})))) = g_K \circ g_{K-1} \circ \cdots \circ g_1$$

  is also a diffeorphism

- denoting $f_k = g_k^{-1}$, $k = 1, \ldots, K$ and $f = g^{-1}$
  then inverse of g writes as

$$g^{-1} = f(\boldsymbol{x}) = f_1(f_2(\ldots(f_K(\boldsymbol{x})))) = f_1 \circ f_2 \circ \cdots \circ f_K$$

- a composite flow is composed from simple flows

Jacobian of a composite flow

- composite flow

$$\boldsymbol{z} \underset{f_1}{\overset{g_1}{\rightleftarrows}} \boldsymbol{x}_1 \underset{f_2}{\overset{g_2}{\rightleftarrows}} \boldsymbol{x}_2 \longleftrightarrow \cdots \longleftrightarrow \boldsymbol{x}_{K-1} \underset{f_K}{\overset{g_K}{\rightleftarrows}} \boldsymbol{x}_K = \boldsymbol{x}$$

- if $f = f_1 \circ f_2 \circ \cdots \circ f_K$, then

$$\det(\mathsf{J}_f(\boldsymbol{x})) = \mathsf{J}_{f_1 \circ f_2 \circ \cdots \circ f_K}(\boldsymbol{x}) = \prod_{k=1}^{K} \det(\mathsf{J}_{f_k}(\boldsymbol{x}_k))$$

- the transformation formula has telescopic form

$$p_X(\boldsymbol{x}) \;=\; p_Z(f_1 \circ \cdots \circ f_K(\boldsymbol{x})) \cdot \prod_{k=1}^{K} |\det(\mathsf{J}_{f_k}(\boldsymbol{x}_k))|$$

Factorization of transformed density

- logarithm of transformed density

$$\log(p_X(\boldsymbol{x})) \;=\; \log(p_Z(f_1 \circ \cdots \circ f_K(\boldsymbol{x}))) + \sum_{k=1}^{K} \log(|\det(\mathrm{J}_{f_k}(\boldsymbol{x}_k))|)$$

- simple flows f_k are parametrized

$$\boldsymbol{z} = f_k(\boldsymbol{x}; \boldsymbol{\theta}_k)$$

- MLE optimization, $\mathcal{D} = \{\boldsymbol{x}^i\}_{i=1}^N$, w.r.t. $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k)$

$$\boldsymbol{\theta}^* = \max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left[\log(p_Z(f_1 \circ \ldots f_K(\boldsymbol{x}^i; \boldsymbol{\theta}))) + \sum_{k=1}^{K} \log(|\det(\mathrm{J}_{f_k}(\boldsymbol{x}_k^i; \boldsymbol{\theta}_k))|) \right]$$

Elementwise flow

- based on univariate differentiable bijections $h_i : \mathbb{R} \to \mathbb{R}$
- $g(\boldsymbol{z}) = (h_1(z_1), h_2(z_2), \ldots, h_d(z_d))$
- $f(\boldsymbol{x}) = (h_1^{-1}(x_1), h_2^{-1}(x_2), \ldots, h_d^{-1}(x_d))$
- Jacobian is diagonal matrix with entries

  $$\mathsf{J}_f(\boldsymbol{x}) = \mathsf{diag}(f(\boldsymbol{x})) = \mathsf{diag}((h_1^{-1}(x_1), h_2^{-1}(x_2), \ldots, h_d^{-1}(x_d)))$$

- determinant of J_f is product of its diagonal elements

  $$\mathsf{det}(\mathsf{J}_f(\boldsymbol{x})) = \prod_{i=1}^{d} \frac{\mathsf{d}h^{-1}}{\mathsf{d}x_i}(x_i)$$

Linear flow

- let $g(\boldsymbol{z}) = A\boldsymbol{z} + \boldsymbol{b}$ where A is an invertible matrix

- for inversion one has $f(\boldsymbol{x}) = A^{-1}(\boldsymbol{x} - \boldsymbol{b})$

- Jacobian is constant and equals to A^{-1} and therefore

$$\det(\mathrm{J}_f(\boldsymbol{x})) = \det(A^{-1}) = \det(A)^{-1}$$

- low expresibility, only linear transformations,
  a normal distribution transforms to a normal distribution

- generally, costly computation of J_f, it is $O(d^3)$

Coupling flow

- $\boldsymbol{x} \in \mathbb{R}^d$, split of $\boldsymbol{x} = (\boldsymbol{x}^D, \boldsymbol{x}^B)$, $\boldsymbol{x}^A \in \mathbb{R}^d$, $\boldsymbol{x}^B \in \mathbb{R}^{D-d}$
  let $h_\theta : \mathbb{R}^{D-d} \to \mathbb{R}^{D-d}$, $\theta \in \mathbb{R}^{D-d}$ be a parametrized bijection
  and Θ arbitrary function, $\Theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$

- coupling flow then reads as $f(\boldsymbol{x}) = (\boldsymbol{z}^A, \boldsymbol{z}^B)$, where

$$\begin{aligned} \boldsymbol{z}^A &= \boldsymbol{x}^A \\ \boldsymbol{z}^B &= h_\theta(\boldsymbol{x}^B) = h(\boldsymbol{x}^B; \theta = \Theta(\boldsymbol{x}^A)) \end{aligned}$$

  and h_θ is called a coupling function

- inverse $g(\boldsymbol{z}) = (\boldsymbol{x}^A, \boldsymbol{x}^B)$ then reads as

$$\begin{aligned} \boldsymbol{x}^A &= \boldsymbol{z}^A \\ \boldsymbol{x}^B &= h_\theta^{-1}(\boldsymbol{z}^B) = h^{-1}(\boldsymbol{z}^B; \theta = \Theta(\boldsymbol{z}^A)) \end{aligned}$$

Coupling flow - Jacobian

- standard coupling flow

$$\begin{aligned} \boldsymbol{z}^A &= \boldsymbol{x}^A \\ \boldsymbol{z}^B &= h_\theta(\boldsymbol{x}^B) = h(\boldsymbol{x}^B; \Theta(\boldsymbol{x}^A)) \end{aligned}$$

- coupling functions $h_\theta : \mathbb{R}^{D-d} \rightarrow \mathbb{R}^{D-d}$
  are applied to $\boldsymbol{x}_B$ elementwise

$$h(\cdot, \boldsymbol{\theta}) = (h_1(x_1^B, \theta_1),\ h_2(x_2^B, \theta_2),\ \ldots,\ h_{D-d}(x_{D-d}^B, \theta_d))$$

where each $h_i(\cdot, \theta_i)$ is a scalar differentiable bijection

Coupling flow - Jacobian

- then the Jacobian is a lower triangular matrix

$$\mathsf{J}_f = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial \boldsymbol{z}^B}{\partial \boldsymbol{x}^A} & \frac{\partial \boldsymbol{z}^B}{\partial \boldsymbol{x}^B} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial h(\boldsymbol{x}^B, \Theta(\boldsymbol{x}^A))}{\partial \boldsymbol{x}^A} & \frac{\partial h(\boldsymbol{x}^B, \Theta(\boldsymbol{x}^A))}{\partial \boldsymbol{x}^B} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial h(\boldsymbol{x}^B, \Theta(\boldsymbol{x}^A))}{\partial \boldsymbol{x}^A} & \mathrm{diag}(\partial h_i(\cdot, \theta_i)/\partial x_i^B) \end{bmatrix}$$

- determinant is then product of the diagonal elements of J_f

Coupling flow

- a concrete example

$$\begin{aligned} \boldsymbol{z}^{1:d} &= \boldsymbol{x}^{1:d} \\ \boldsymbol{z}^{d+1:D} &= \boldsymbol{x}^{d+1:D} \odot \exp(s_\theta(\boldsymbol{x}^{1:d})) + t_\theta(\boldsymbol{x}^{1:d}) \end{aligned}$$

where $s_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$, $t_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ are neural networks

- $\odot$ is the elementwise product, i.e,

$$\boldsymbol{x} \odot \boldsymbol{y} = (x_1 y_1, \ldots, x_d y_d)$$

- inverse reads as

$$\begin{aligned} \boldsymbol{x}^{1:d} &= \boldsymbol{z}^{1:d} \\ \boldsymbol{x}^{d+1:D} &= (\boldsymbol{z}^{d+1:D} - t_\theta(\boldsymbol{z}^{1:d})) \odot \exp(-s_\theta(\boldsymbol{z}^{1:d})) \end{aligned}$$

Coupling flow - expressibility

- going from layer to layer in a composite flow variables must be somehow permuted to allow for complex relation modelling

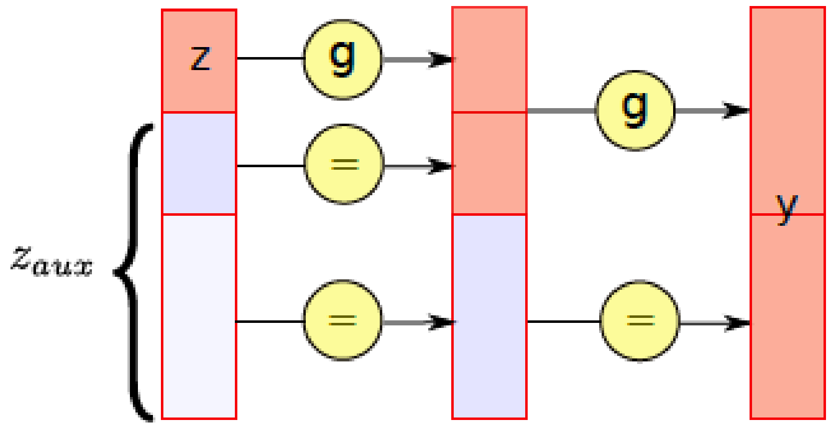
- standard approach is to apply random permutations when creating the flow and split dimensions in half

- more complex schema are possible, e.g., alternating pixels or blocks of channels, which is called masking

- computational complexity of Jacobian is $O(D)$

Coupling flow - multiscale architecture

- noise vector is introduced along length of the flow which decreases complexity of computations



source: https://arxiv.org/abs/1908.09257

Autoregressive flow

- autoregressive model of p-th order $AR(p)$ has form

$$
\begin{aligned}
X_t &= \sum_{i=1}^{p} \varphi_t X_{t-i} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0,1) \\
X_t &= h_t(\epsilon_t, \sum_{i=1}^{p} \varphi_t X_{t-i}) \\
X_t &= h_t(\epsilon_t, \Theta_t(\boldsymbol{X}_{t-1:t-p}))
\end{aligned}
$$

- in autoregressive flows the above schema is generalized

- h_t is a differentiable bijection a Θ_t is an arbitrary function typically represented by a neural network

Autoregressive flow

- let h_θ is parametrized differentiable bijection
  construct $g : \mathbb{R}^D \to \mathbb{R}^D$,

  $$(x_1, \ldots x_D) = \boldsymbol{x} = g(\boldsymbol{z})$$

  in autoregressive manner, i.e.,

  $$x_i = h(z_i; \Theta_i(\boldsymbol{x}_{1:i-1})),\ i = 1, \ldots, D$$

  with $\Theta_1 = \theta_1$ being a constant and Θ_i
  arbitrary functions defined on respective domains

- inverse $(z_1, \ldots z_D) = f(\boldsymbol{x})$, then reads as

  $$z_i = h^{-1}(x_i; \Theta_i(\boldsymbol{x}_{1:i-1})),\ i = 1, \ldots, D,\ \Theta_1 = \theta_1$$

  no autoregressive structure

Autoregressive flow

- Jacobian of f is a lower triangular matrix
- with determinant

$$\det(\mathsf{J}_f(\boldsymbol{x})) = \prod_{k=1}^{D} \frac{\partial h^{-1}(x_i; \Theta_i(\boldsymbol{x}_{1:i-1}))}{\partial x_i}$$

- example

$$x_i = z_i \cdot \exp(s_\theta(\boldsymbol{x}_{1:i-1})) + t_\theta(\boldsymbol{x}_{1:i-1}) \quad \text{and} \quad z_i \sim \mathcal{N}(0,1)$$

- tight connection to coupling flows

Masked autoregressive flow

- masking (MAF) allows for one-pass computation of $f(\boldsymbol{x})$ (fast evaluation of likelihood)

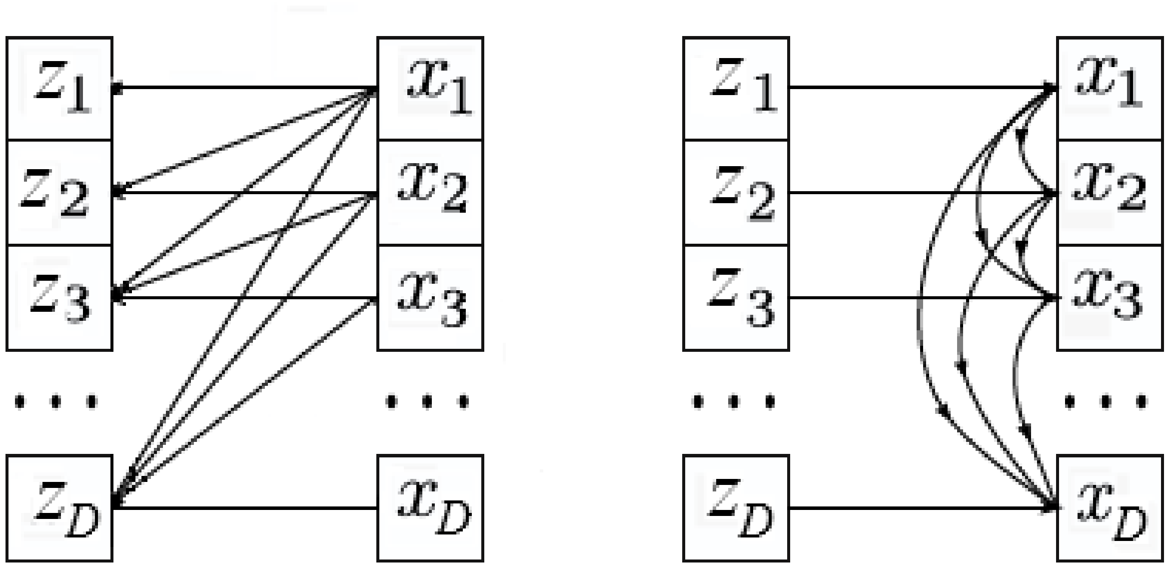
$$z_i = h^{-1}(x_i; \Theta_i(\boldsymbol{x}_{1:i-1})) \quad \text{(parallel via masking)}$$

- however sampling (generative direction), i.e., computing $g(\boldsymbol{z})$, is inherently sequential (slow)

$$x_i = h(z_i; \Theta_i(\boldsymbol{x}_{1:i-1})) \quad \text{(sequential)}$$

Autoregresive flow

- masked autoregresive flows (MAF)
  - fast likelihood, slow sampling



- inverse autoregresive flows (IAF)
  - fast sampling, slow likelihood

Conditional autoregresive flow

- natural extension to conditional version, by augmenting input with class information

- for a training point $\{\boldsymbol{x}, \boldsymbol{c}\}$, we incorporate $\boldsymbol{c}$ into the θ parameter to get conditional density

$$p_X(\boldsymbol{x}|\boldsymbol{c}) = p_Z(f(\boldsymbol{x}|\boldsymbol{c})) \cdot |\det(\mathsf{J}_f(\boldsymbol{x}|\boldsymbol{c}))|$$

$$z_i = h^{-1}(x_i; \Theta_i(\boldsymbol{x}_{1:i-1}, \boldsymbol{c})),\ i = 1, \ldots, D$$

- conditional sampling

$$x_i|\boldsymbol{c} = h(z_i; \Theta_i(\boldsymbol{x}_{1:i-1}, \boldsymbol{c})),\ i = 1, \ldots, D$$

NICE (2014)

- L. Dinh, D. Krueger, Y. Bengio:
  NICE: Non-linear Independent Component Estimation
  https://arxiv.org/abs/1410.8516

$$h^{(1)}_{I_1} = x_{I_1}$$
$$h^{(1)}_{I_2} = x_{I_2} + m^{(1)}(x_{I_1})$$

$$h^{(2)}_{I_2} = h^{(1)}_{I_2}$$
$$h^{(2)}_{I_1} = h^{(1)}_{I_1} + m^{(2)}(x_{I_2})$$

$$h^{(3)}_{I_1} = h^{(2)}_{I_1}$$
$$h^{(3)}_{I_2} = h^{(2)}_{I_2} + m^{(3)}(x_{I_1})$$

$$h^{(4)}_{I_2} = h^{(3)}_{I_2}$$
$$h^{(4)}_{I_1} = h^{(3)}_{I_1} + m^{(4)}(x_{I_2})$$

$$h = \exp(s) \odot h^{(4)}$$

The coupling functions $m^{(1)}, m^{(2)}, m^{(3)}$ and $m^{(4)}$ used for the coupling layers are all deep rectified networks with linear output units. We use the same network architecture for each coupling function: five hidden layers of 1000 units for MNIST, four of 5000 for TFD, and four of 2000 for SVHN and CIFAR-10.

NICE (2014)

- four standard ML datasets

  MNIST - Handwritten digit dataset - 28x28 (grayscale)
  TFD - Toronto Faces Dataset - 32x32 (grayscale)
  SVHN - The Street View House Numbers - 32x32 RGB
  CIFAR-10 - 32x32 RGB images in 10 classes

- numerical results

Dataset	MNIST	TFD	SVHN	CIFAR-10
# dimensions	784	2304	3072	3072
Preprocessing	None	Approx. whitening	ZCA	ZCA
# hidden layers	5	4	4	4
# hidden units	1000	5000	2000	2000
Prior	logistic	gaussian	logistic	logistic
Log-likelihood	1980.50	5514.71	11496.55	5371.78

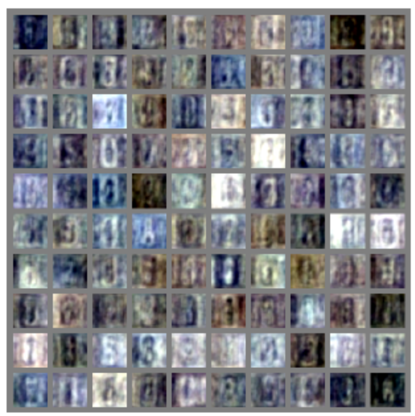
Figure 3: Architecture and results. # hidden units refer to the number of units per hidden layer.

NICE (2014)

- sampling

(a) Model trained on MNIST

(b) Model trained on TFD



(c) Model trained on SVHN

(d) Model trained on CIFAR-10

Figure 5: Unbiased samples from a trained NICE model. We sample $h \sim p_H(h)$ and we output $x = f^{-1}(h)$.

Real NVP (ICLR 2017)

- L. Dinh, J. Sohl-Dickstein, S. Bengio:
  Density Estimation Using Real NVP
  https://arxiv.org/abs/1605.08803

but which depends on the remainder of the input vector in a complex way. We refer to each of these simple bijections as an *affine coupling layer*. Given a D dimensional input x and $d < D$, the output y of an affine coupling layer follows the equations

$$y_{1:d} = x_{1:d} \tag{4}$$

$$y_{d+1:D} = x_{d+1:D} \odot \exp\big(s(x_{1:d})\big) + t(x_{1:d}), \tag{5}$$

where s and t stand for scale and translation, and are functions from $R^d \mapsto R^{D-d}$, and $\odot$ is the Hadamard product or element-wise product (see Figure 2(a)).

3.3 Properties

The Jacobian of this transformation is

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x^T_{1:d}} & \text{diag}\left(\exp\left[s\left(x_{1:d}\right)\right]\right) \end{bmatrix}, \tag{6}$$

Real NVP (ICLR 2017)

- masked convolutions

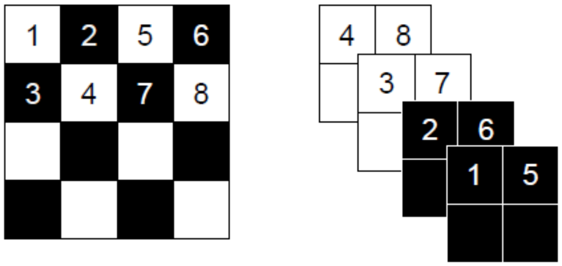


Figure 3: Masking schemes for affine coupling layers. On the left, a spatial checkerboard pattern mask. On the right, a channel-wise masking. The squeezing operation reduces the $4 \times 4 \times 1$ tensor (on the left) into a $2 \times 2 \times 4$ tensor (on the right). Before the squeezing operation, a checkerboard pattern is used for coupling layers while a channel-wise masking pattern is used afterward.

(see Figure 2(b)),

$$\begin{cases} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp\big(s(x_{1:d})\big) + t(x_{1:d}) \end{cases} \tag{7}$$

$$\Leftrightarrow \begin{cases} x_{1:d} &= y_{1:d} \\ x_{d+1:D} &= \big(y_{d+1:D} - t(y_{1:d})\big) \odot \exp\big(-s(y_{1:d})\big), \end{cases} \tag{8}$$

meaning that sampling is as efficient as inference for this model. Note again that computing the inverse of the coupling layer does not require computing the inverse of s or t, so these functions can be arbitrarily complex and difficult to invert.

3.4 Masked convolution

Partitioning can be implemented using a binary mask b, and using the functional form for y,

$$y = b \odot x + (1 - b) \odot \Big(x \odot \exp\big(s(b \odot x)\big) + t(b \odot x)\Big). \tag{9}$$

Real NVP (ICLR 2017)

- results on CelebA

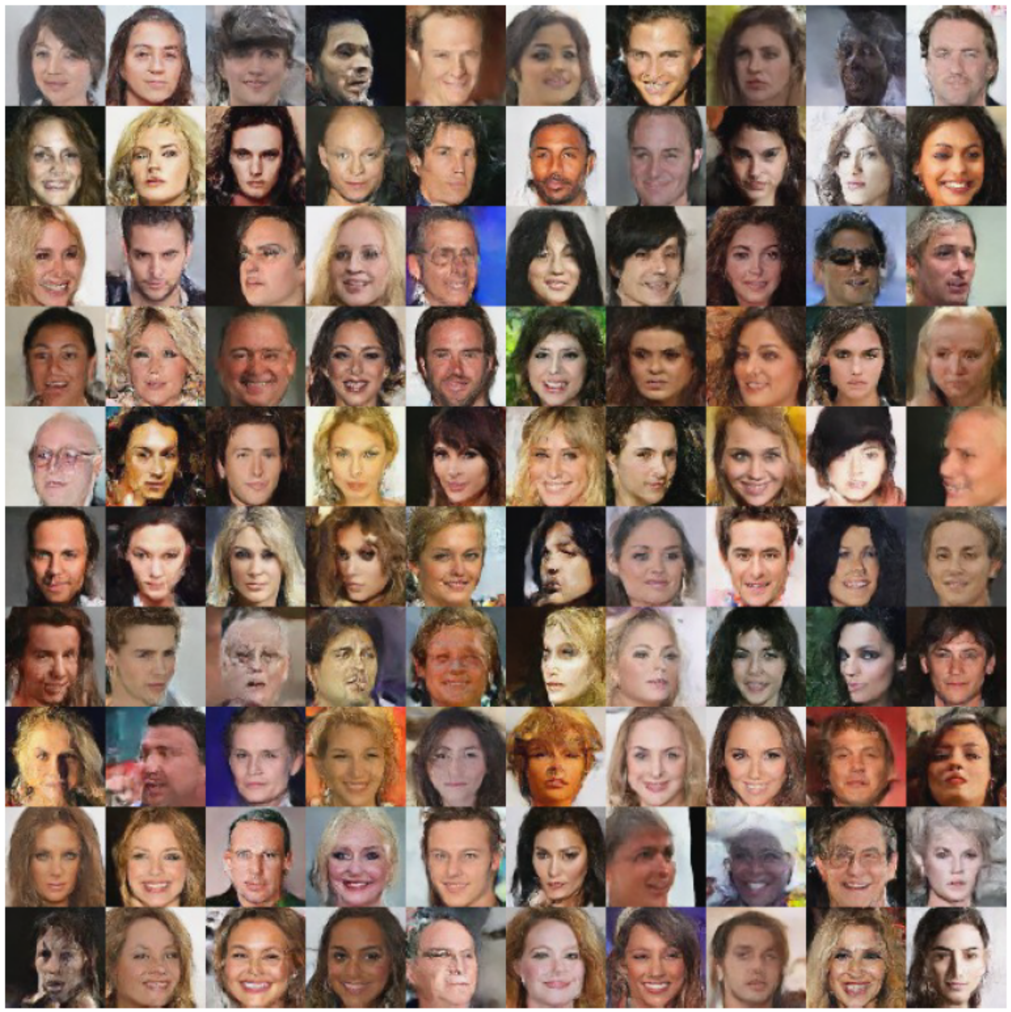
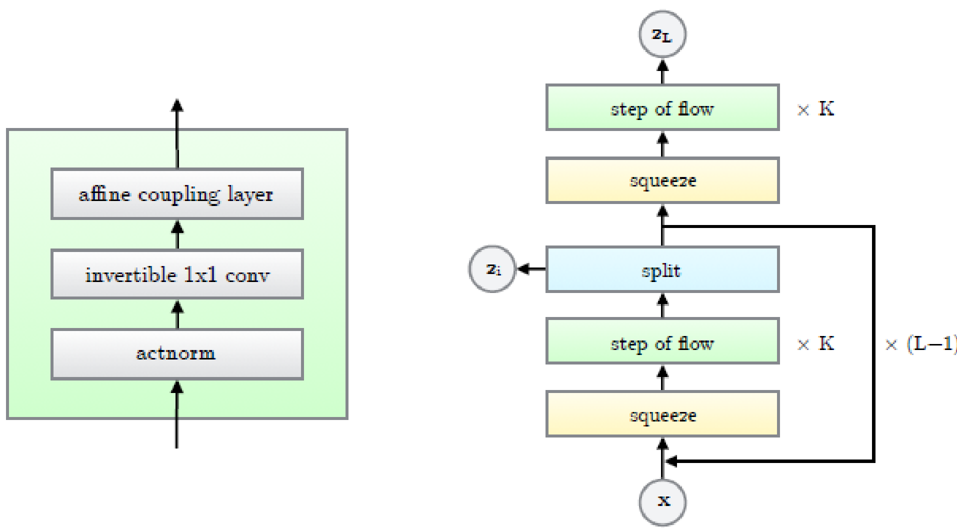


Figure 8: Samples from a model trained on *CelebA*.

Glow (2018)

- D. P. Kingma, P. Dhariwal :
  Glow: Generative Flow with Invertible 1x1 Convolutions
  https://arxiv.org/abs/1807.03039



(a) One step of our flow. (b) Multi-scale architecture (Dinh et al., 2016).

Figure 2: We propose a generative flow where each step (left) consists of an *actnorm* step, followed by an invertible 1×1 convolution, followed by an affine transformation (Dinh et al., 2014). This flow is combined with a multi-scale architecture (right). See Section 3 and Table 1.

Glow (2018)

- 1×1 convolutions

3.2 Invertible 1×1 convolution

(Dinh et al., 2014, 2016) proposed a flow containing the equivalent of a permutation that reverses the ordering of the channels. We propose to replace this fixed permutation with a (learned) invertible 1×1 convolution, where the weight matrix is initialized as a random rotation matrix. Note that a 1×1 convolution with equal number of input and output channels is a generalization of a permutation operation.

The log-determinant of an invertible 1×1 convolution of a $h \times w \times c$ tensor $\mathbf{h}$ with $c \times c$ weight matrix $\mathbf{W}$ is straightforward to compute:

$$\log \left| \det \left(\frac{d\, \texttt{conv2D}(\mathbf{h}; \mathbf{W})}{d\, \mathbf{h}} \right) \right| = h \cdot w \cdot \log |\det(\mathbf{W})| \tag{9}$$

The cost of computing or differentiating $\det(\mathbf{W})$ is $\mathcal{O}(c^3)$, which is often comparable to the cost computing $\texttt{conv2D}(\mathbf{h}; \mathbf{W})$ which is $\mathcal{O}(h \cdot w \cdot c^2)$. We initialize the weights $\mathbf{W}$ as a random rotation matrix, having a log-determinant of 0; after one SGD step these values start to diverge from 0.

LU Decomposition. This cost of computing $\det(\mathbf{W})$ can be reduced from $\mathcal{O}(c^3)$ to $\mathcal{O}(c)$ by parameterizing $\mathbf{W}$ directly in its LU decomposition:

$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \text{diag}(\mathbf{s})) \tag{10}$$

where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is a lower triangular matrix with ones on the diagonal, $\mathbf{U}$ is an upper triangular matrix with zeros on the diagonal, and $\mathbf{s}$ is a vector. The log-determinant is then simply:

$$\log |\det(\mathbf{W})| = \texttt{sum}(\log |\mathbf{s}|) \tag{11}$$

Glow (2018)

- samples (learning - 40 GPU for a week)

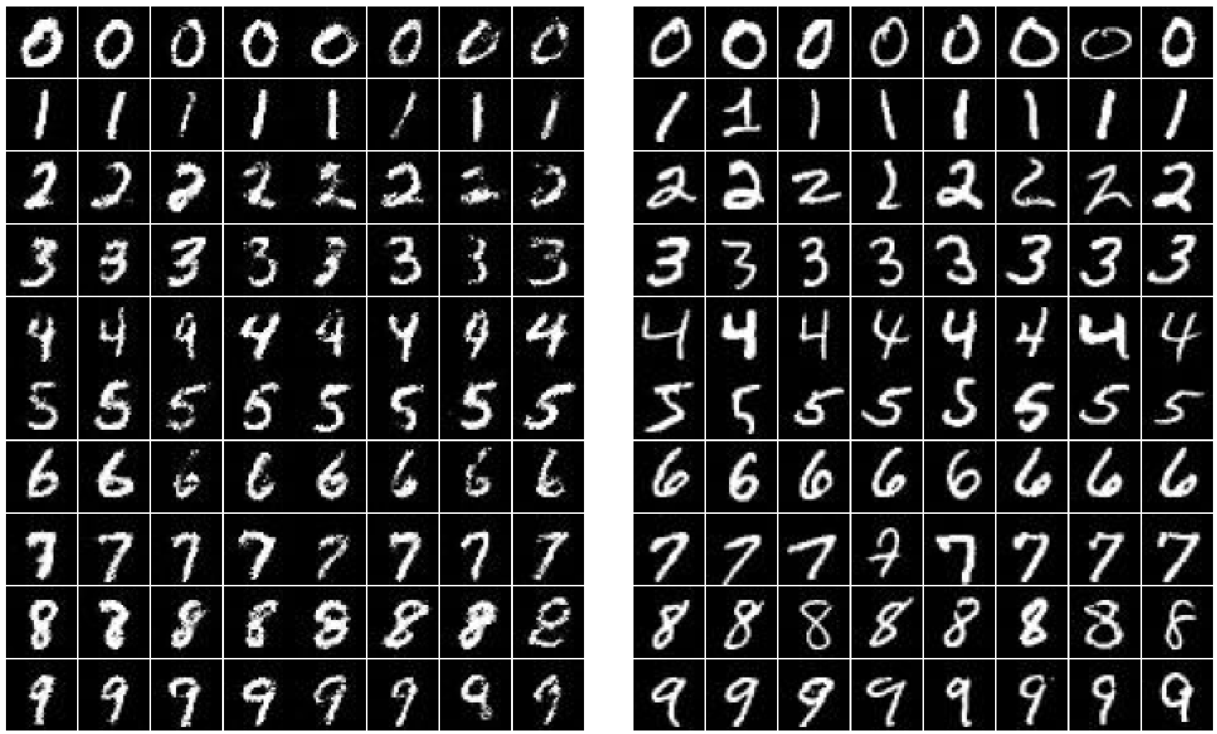
Table 2: Best results in bits per dimension of our model compared to RealNVP.

Model	CIFAR-10	ImageNet 32x32	ImageNet 64x64	LSUN (bedroom)	LSUN (tower)	LSUN (church outdoor)
RealNVP	3.49	4.28	3.98	2.72	2.81	3.08
Glow	**3.35**	**4.09**	**3.81**	**2.38**	**2.46**	**2.67**

Figure 4: Random samples from the model, with temperature 0.7

Masked Autoregressive Flows (2017)

- P. Papamakarios, Theo Pavlakou, Iain Murray: *Masked Autoregressive Flow for Density Estimation* https://arxiv.org/abs/1705.07057



(a) Generated images (b) Real images

Masked Autoregressive Flows (2017)

- conditional CIFAR

(a) Generated images (b) Real images

Other flows

- residual and planar flows (no closed form inversion)
- residual flows (iResNet)
- continuous flows - ODE, SDE (FFJORD, Diffusion flows)

Review article

- I. Kobyzev, S. J. D. Prince, M. A. Brubaker: *Normalizing Flows: An Introduction and Review of Current Methods* (2020) https://ieeexplore.ieee.org/document/9089305