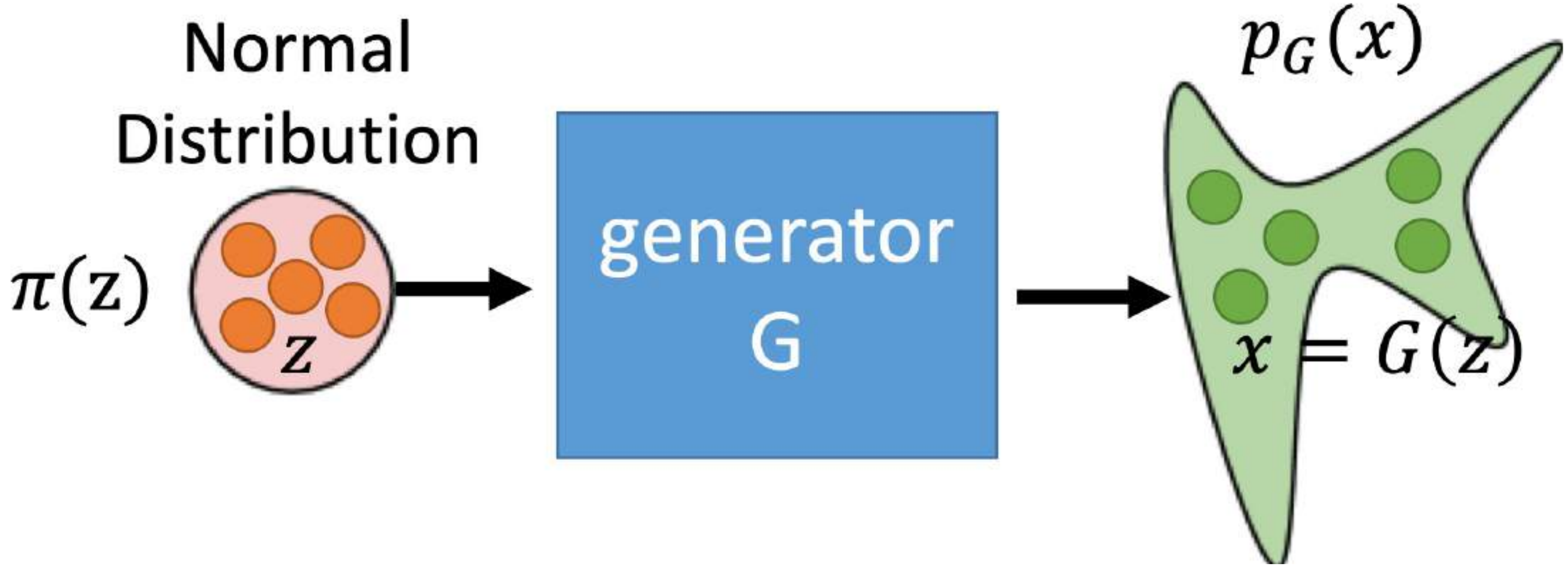
深度学习

Lecture 9 Normalizing Flow

Pang Tongyao, YMSC

Change Variables

Generator: transform a simple distribution to the data distribution



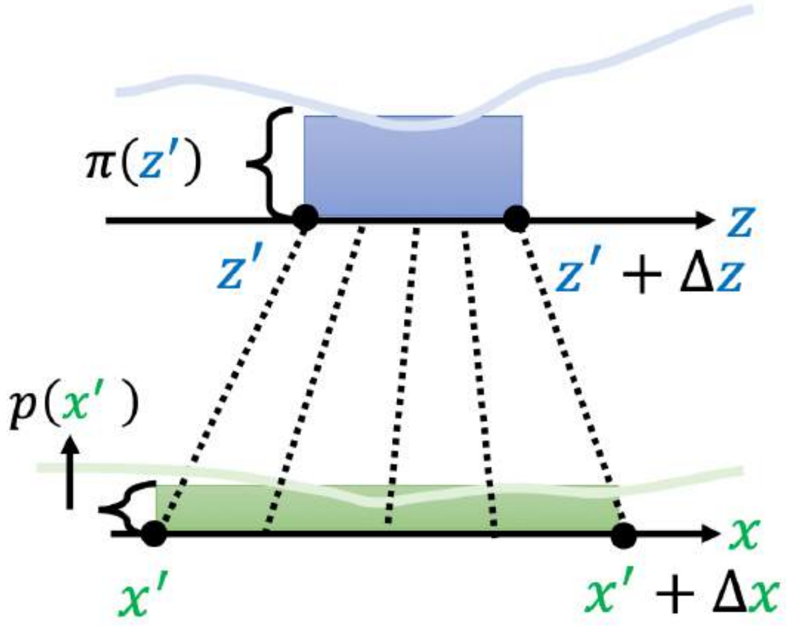
Generation: sample $z_0 \sim \pi(z), x = G(z_0)$

Change Variables

- For invertible neural networks, the change of variable formula:

$$p(x) = \pi(z)\,|\det(J_{G^{-1}}(x))\,|, z = G^{-1}(x)$$

$$J_{G^{-1}}: \text{Jacobian matrix of } G^{-1}$$



$$p(x')\Delta x = \pi(z')\Delta z$$

$$p(x') = \pi(z')\frac{\Delta z}{\Delta x}$$

$$p(x') = \pi(z')\left|\frac{dz}{dx}\right|$$

Image credit: Hungyi Lee, National Taiwan University

Normalizing Flows

- Loss function

$$\log p(x) = \log \pi(G^{-1}(x)) + \log|\det(J_{G^{-1}}(x))\ |$$

- Decompose G into the decomposition of many sub-net

$$z_K = f_{\theta^K} \circ f_{\theta^{K-1}} \circ \cdots \circ f_{\theta^1}(z_0)$$

Chain rule

$|\det(J_{G^{-1}})| = 1/|\det(J_G)|$

$$logp(z_K) = \log \pi(z_0) - \sum_{k=1}^{K} \log|\det(\frac{\partial f_{\theta^k}}{\partial z_k})\ |$$

Normalizing Flows

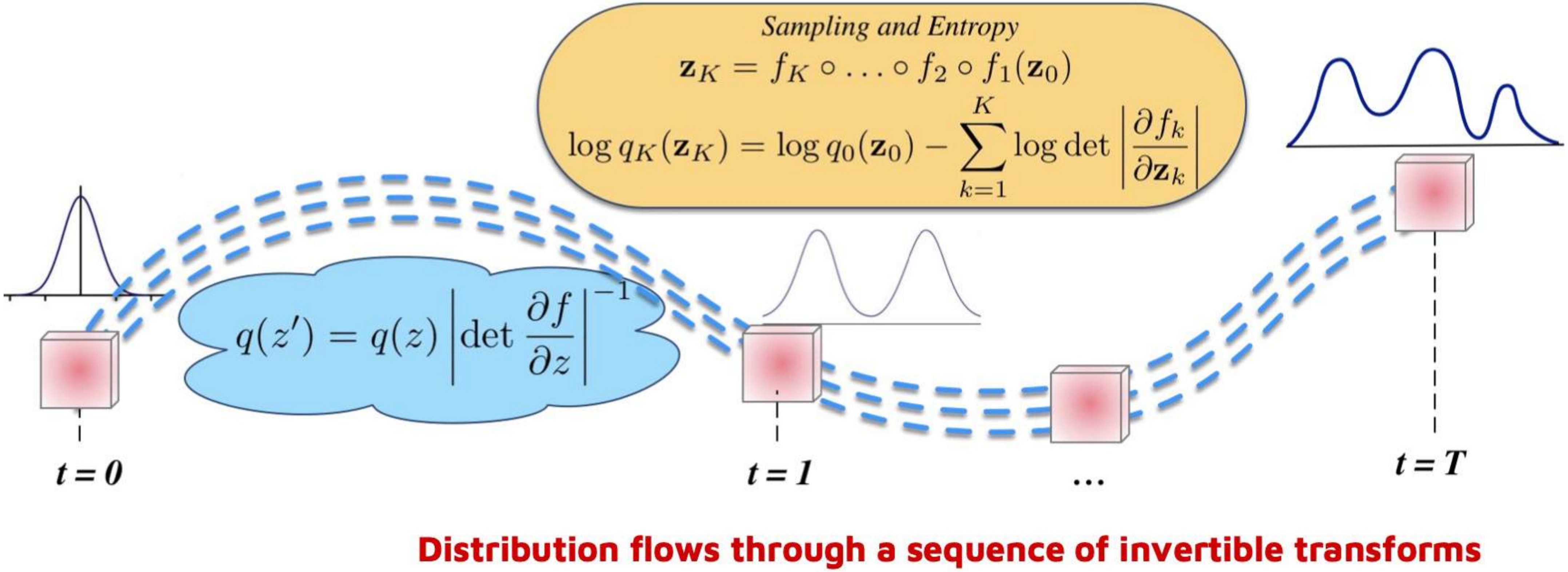


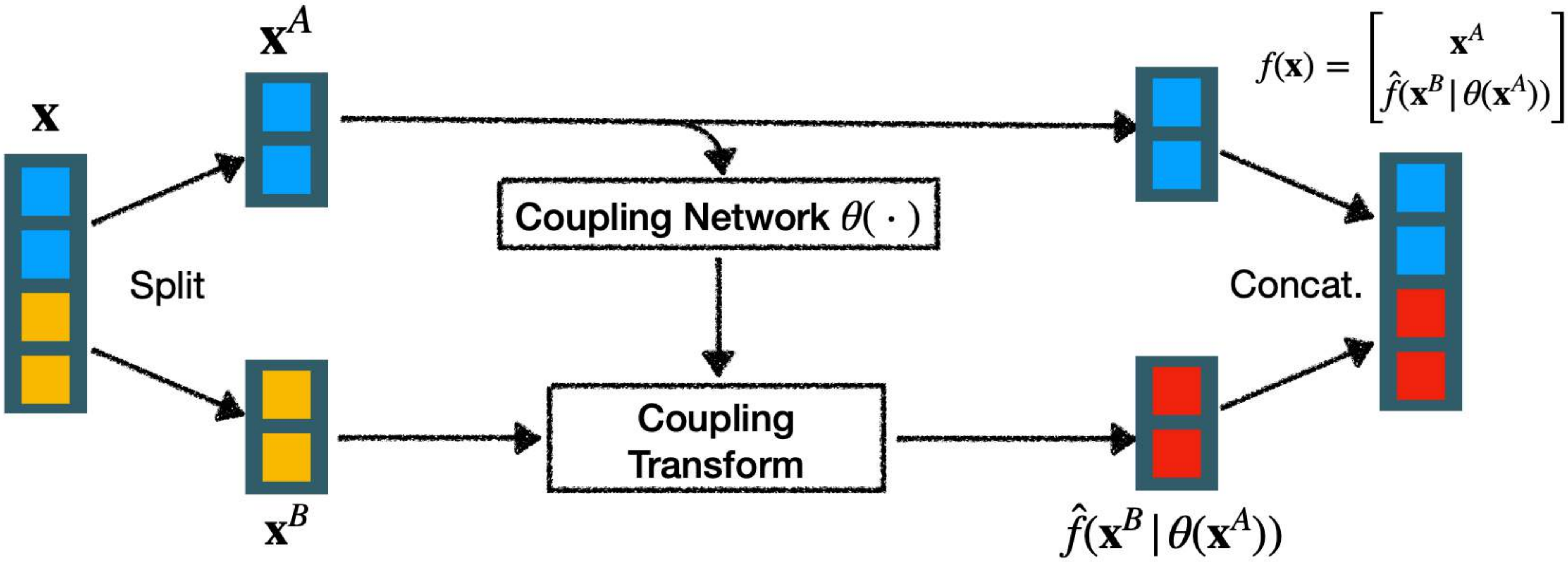
Image Credit: Rezende and Mohamed, DeepMind

Normalizing Flows

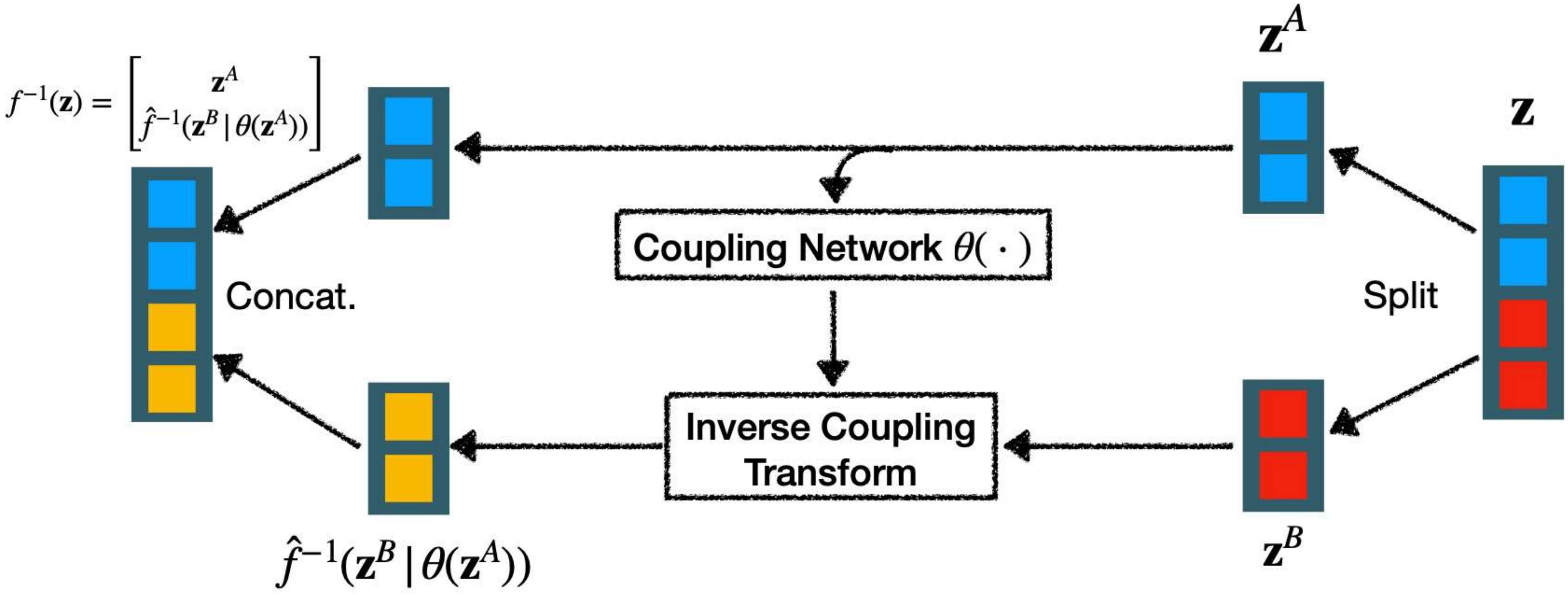
- Normalizing Flows f: invertible, differentiable, efficiently computable $\det|J_f|$.
- Non-linear Layer: e.g. ELU invertible and easy to compute inverse and determinant.
- Linear layer:

	Inverse	Determinant
Full	$O(d^3)$	$O(d^3)$
Diagonal	$O(d)$	$O(d)$
Triangular	$O(d^2)$	$O(d)$
Block Diagonal	$O(c^3 d)$	$O(c^3 d)$
LU Factorized *[Kingma and Dhariwal 2018]*	$O(d^2)$	$O(d)$
Spatial Convolution *[Hoogeboom et al 2019; Karami et al., 2019]*	$O(d \log d)$	$O(d)$
1x1 Convolution *[Kingma and Dhariwal 2018]*	$O(c^3 + c^2 d)$	$O(c^3)$

Coupling Layers



Coupling Layers



Coupling Layers

- $f(x) = \begin{bmatrix} x^A \\ \hat{f}(x^B | \theta(x^A)) \end{bmatrix}$
- Jacobian

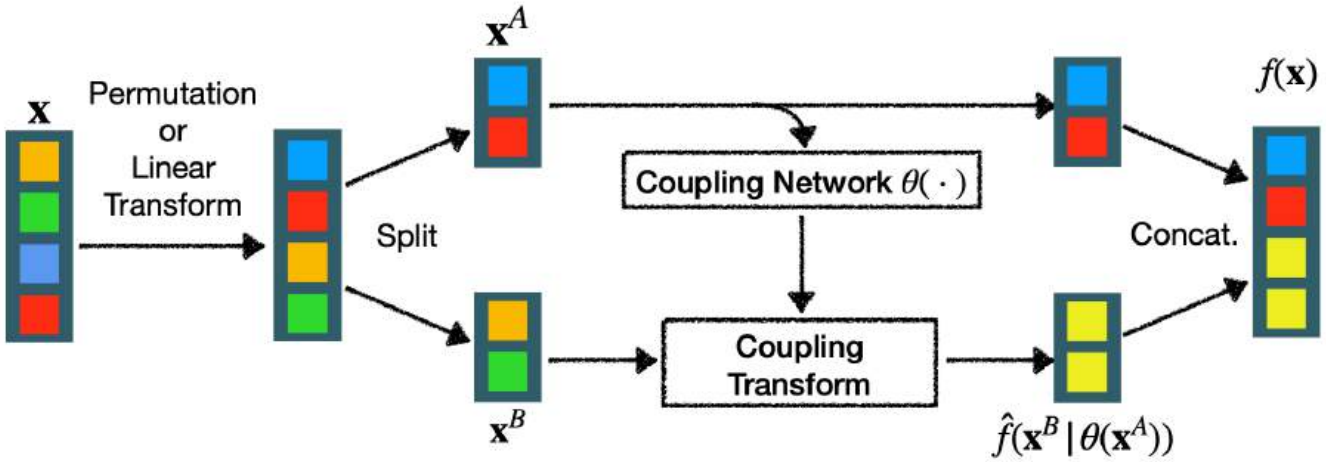
$$J_f = \begin{bmatrix} I & 0 \\ \partial \hat{f}(x^B | \theta(x^A)/\partial x^A & J_{\hat{f}} \end{bmatrix}$$

- Determinant

$$\det(J_f) = \det(J_{\hat{f}})$$

Coupling Layers

- $\theta(x^A)$ can be arbitrary network
- Splitting is important for the expressivity of the invertible net.



Coupling Transform

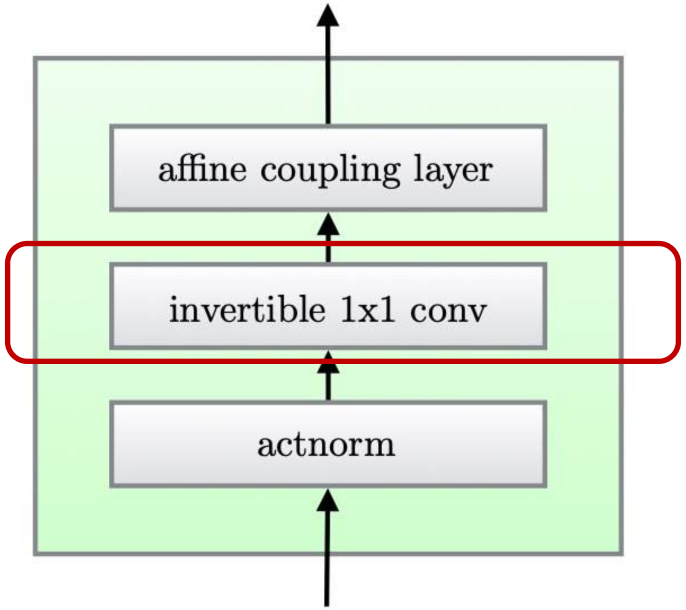
- Linear [NICE, Dinh et al. 2014]

$$\hat{f}(x|t) = x + t$$

- Affine [RealNVP, Dinh et al. 2016]

$$\hat{f}(x|s,t) = s \circ x + t$$

Glow



Input $\boldsymbol{h}$: $c \times h \times w$, $\boldsymbol{W}$: $c \times c$, Output: $c \times h \times w$

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

Compute $|\det(\boldsymbol{W})|$: O(c^3) (Vs. complexity of conv O(hwc^2))

Reparametrize **W** in LU decomposition

$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \mathrm{diag}(\mathbf{s}))$$

$$\log |\det(\mathbf{W})| = \mathrm{sum}(\log |\mathbf{s}|)$$

P(fixed): Premutation matrix

- **L:** A lower triangular matrix with diagonal elements equal to 1.
- **U:** An upper triangular matrix with diagonal elements equal to 0.

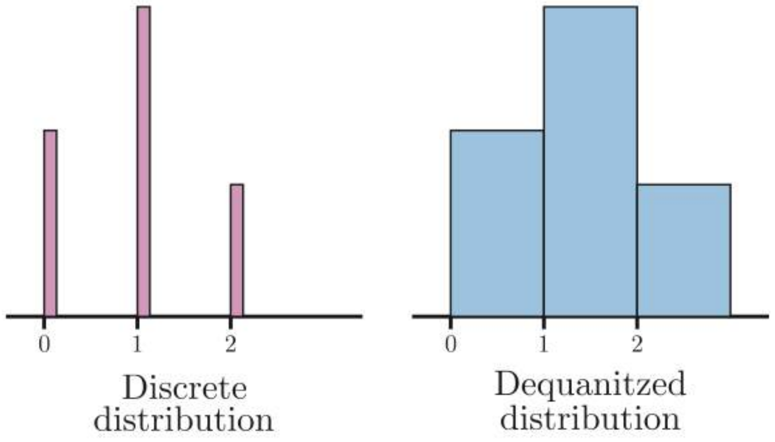
complexity: O(c)

"Glow: Generative Flow with Invertible 1×1 Convolutions"

Dequantize

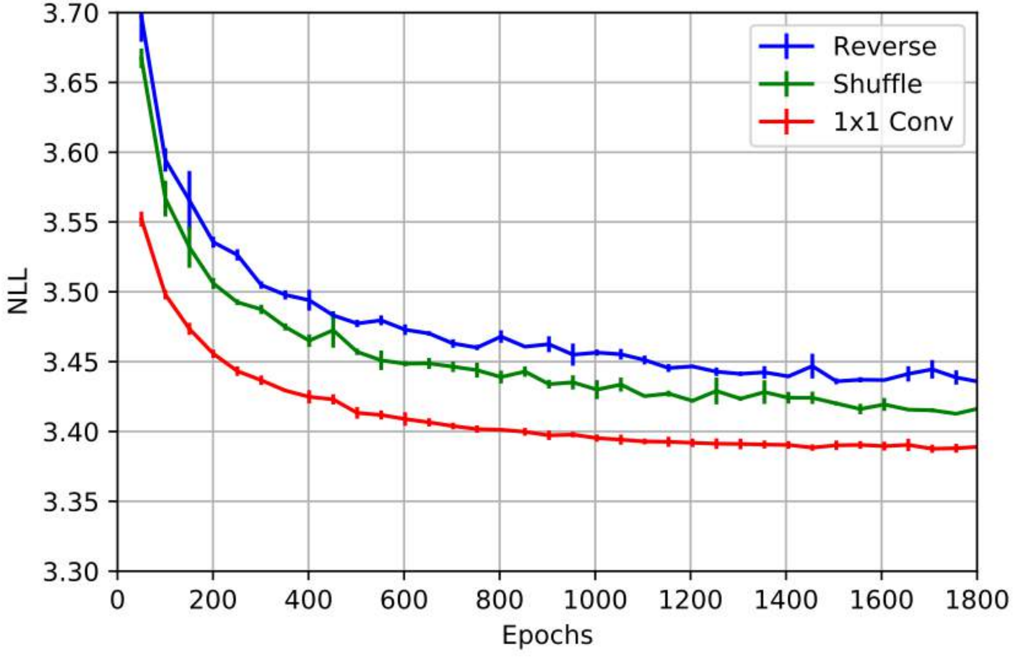
- Training data: image pixel values are discrete
- Training a continuous model with discrete data may cause singularity issues.
- dequantize data (add noise)

$$
\begin{aligned}
\mathrm{p}_d(y) &= \int p_{model}(y+u)p(u)du \\
&\approx \frac{1}{K}\sum_{k=1}^{K} p_{model}(y+u_k)
\end{aligned}
$$

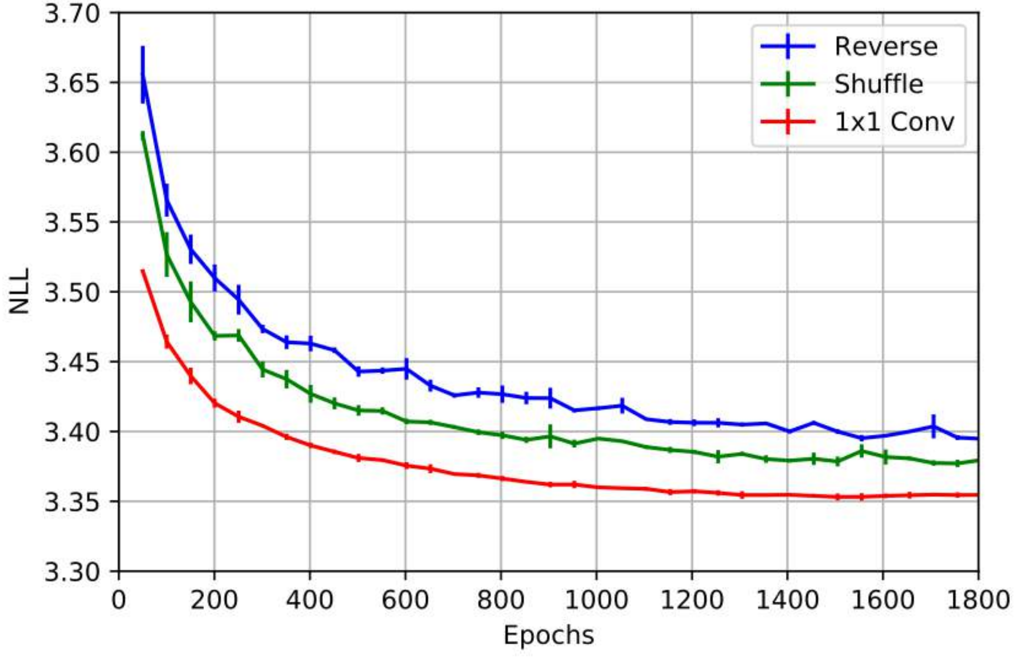


choose as uniform distribution

Glow



(a) Additive coupling.



(b) Affine coupling.

"Glow: Generative Flow with Invertible 1×1 Convolutions"

Continuous-time Normalizing Flows

- Neural ODE

$$\frac{dz(t)}{dt} = f(z(t), t, \theta)$$

forward: $z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)$ (discretization=ResNet)

reverse: $z(t_0) = z(t_1) + \int_{t_1}^{t_0} f(z(t), t, \theta)$

- Flow map of an ODE is always invertible.
- Instantaneous Change of Variables

$$\frac{dlog\, p(z(t))}{dt} = -Tr\left(\frac{df}{dz(t)}\right) = -\mathrm{div}\,(f)$$

"Neural Ordinary Differential Equations"

Continuous-time Normalizing Flows

- Loss

$$L(z(t)) = -\log p\big(z(t)\big) = -\log p\big(z(0)\big) + \int_0^t Tr\left(\frac{df}{dz(t)}\right)$$

$$\underbrace{\begin{bmatrix} \mathbf{z}_0 \\ \log p(\mathbf{x}) - \log p_{z_0}(\mathbf{z}_0) \end{bmatrix}}_{\text{solutions}} = \underbrace{\int_{t_1}^{t_0} \begin{bmatrix} f(\mathbf{z}(t), t; \theta) \\ -\operatorname{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) \end{bmatrix} dt}_{\text{dynamics}}, \qquad \underbrace{\begin{bmatrix} \mathbf{z}(t_1) \\ \log p(\mathbf{x}) - \log p(\mathbf{z}(t_1)) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}}_{\text{initial values}}$$

- Dynamics of adjoint $\mathbf{a}(t) = {}^{\partial L}/_{\partial \mathbf{z}(t)}$

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \left(\frac{\partial L}{\partial \mathbf{z}(t)}\right)^T \frac{\partial f(\mathbf{z}(t), t; \theta)}{\partial \theta} dt.$$

Continuous-time Normalizing Flows

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient ${\partial L}/{\partial \mathbf{z}(t_1)}$

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$ ▷ Define initial augmented state

def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta$): ▷ Define dynamics on augmented state

  return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial \theta}]$ ▷ Compute vector-Jacobian products

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

FFJORD

- Neural ODE: compute $\frac{df}{dz(t)}$ is expansive(approximately d forward pass)

$$L(z(t) = -\log p\big(z(t)\big) = -\log p\big(z(0)\big) + \int_0^t Tr\ \left(\frac{df}{dz(t)}\right)$$

- Use Monte-Carlo trace estimate to approximate $Tr\ \left(\frac{df}{dz(t)}\right)$

$$\mathrm{Tr}(A) = E_{p(\boldsymbol{\epsilon})}[\boldsymbol{\epsilon}^T A \boldsymbol{\epsilon}].$$

$$\begin{aligned}
\log p(\mathbf{z}(t_1)) &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt \\
&= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\epsilon}\right] dt \\
&= \log p(\mathbf{z}(t_0)) - \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\int_{t_0}^{t_1} \boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\epsilon} dt\right]
\end{aligned}$$

Vector Jacobian (directional derivative): $\boldsymbol{\epsilon}^T \frac{df}{d\boldsymbol{z}(t)}$

one forward pass

“FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models”

FFJORD

Algorithm 1 Unbiased stochastic log-density estimation using the FFJORD model

Require: dynamics f_θ, start time t_0, stop time t_1, minibatch of samples $\mathbf{x}$.

$\boldsymbol{\epsilon} \leftarrow$ sample_unit_variance($\mathbf{x}$.shape) ▷ Sample $\boldsymbol{\epsilon}$ outside of the integral

function $f_{aug}([\mathbf{z}_t, \log p_t], t)$: ▷ Augment f with log-density dynamics.

$f_t \leftarrow f_\theta(\mathbf{z}(t), t)$ ▷ Evaluate dynamics

$g \leftarrow \boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}}\Big|_{\mathbf{z}(t)}$ ▷ Compute vector-Jacobian product with automatic differentiation

$\widetilde{\mathrm{Tr}} = \mathrm{matrix\_multiply}(g, \boldsymbol{\epsilon})$ ▷ Unbiased estimate of $\mathrm{Tr}(\frac{\partial f}{\partial \mathbf{z}})$ with $\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}} \boldsymbol{\epsilon}$

return $[f_t, -\widetilde{\mathrm{Tr}}]$ ▷ Concatenate dynamics of state and log-density

end function

$[\mathbf{z}, \Delta_{logp}] \leftarrow \mathrm{odeint}(f_{aug}, [\mathbf{x}, \vec{0}], t_0, t_1)$ ▷ Solve the ODE, ie. $\int_{t_0}^{t_1} f_{aug}([\mathbf{z}(t), \log p(\mathbf{z}(t))], t)\ dt$

$\log \hat{p}(\mathbf{x}) \leftarrow \log p_{\mathbf{z}_0}(\mathbf{z})$ - Δ_{logp} ▷ Add change in log-density

return $\log \hat{p}(\mathbf{x})$

Glow has trouble modeling the areas of low probability.

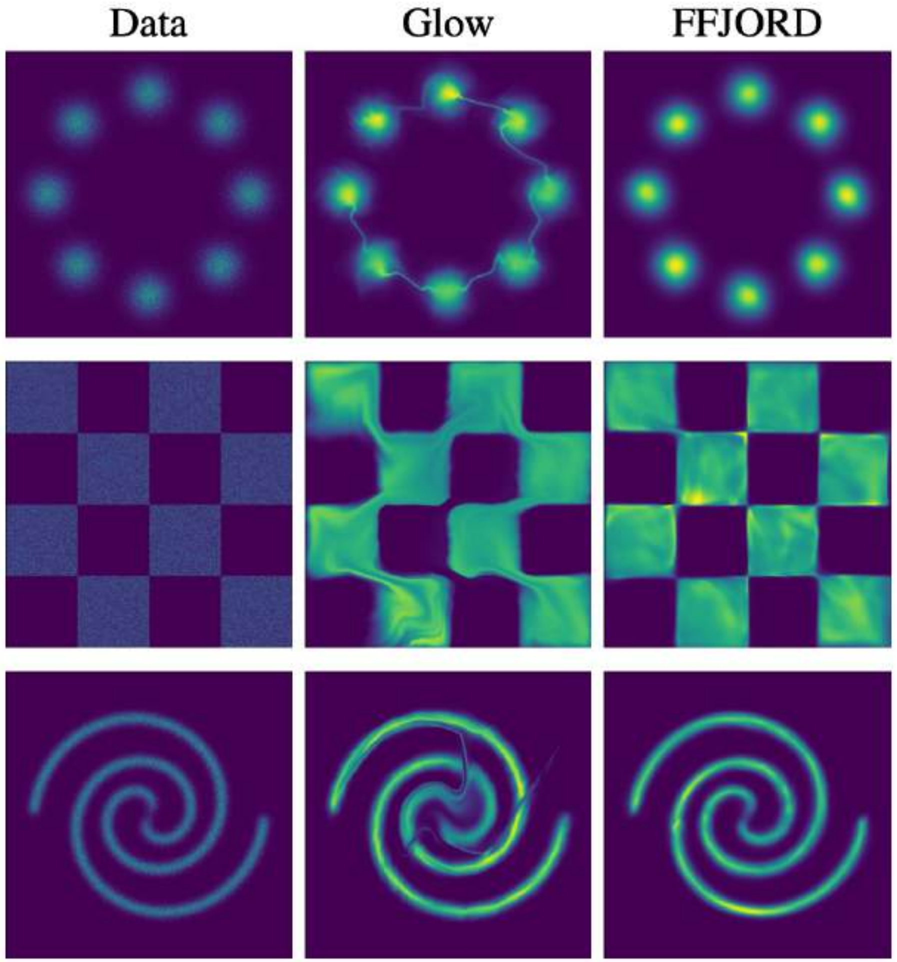


Figure 2: Comparison of trained FFJORD and Glow models on 2-dimensional distributions including multi-modal and discontinuous densities.

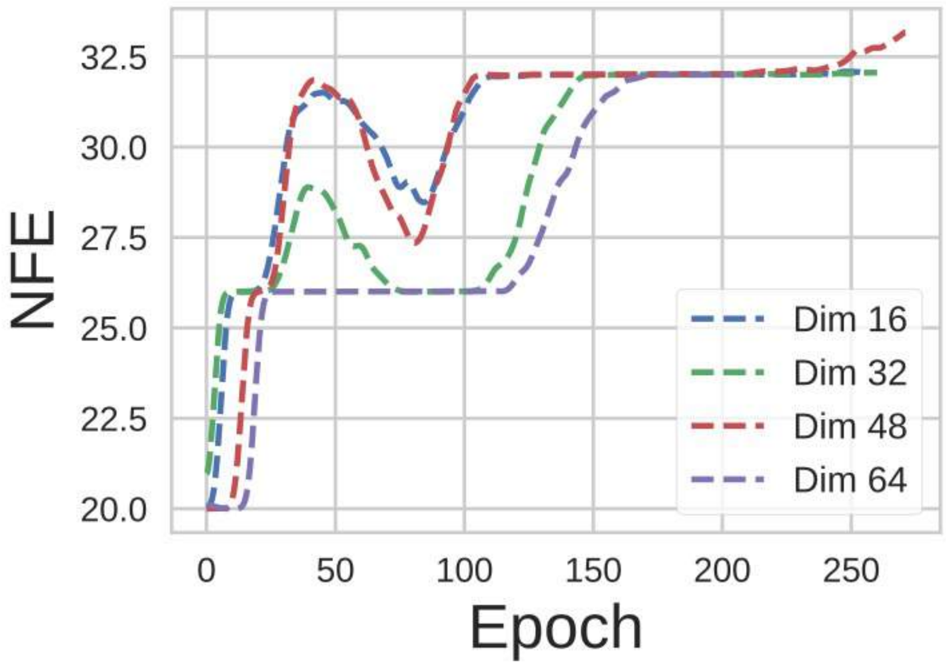


Figure 5: Number of function evaluates used by the adaptive ODE solver (NFE) is approximately independent of data-dimension.

iResNet

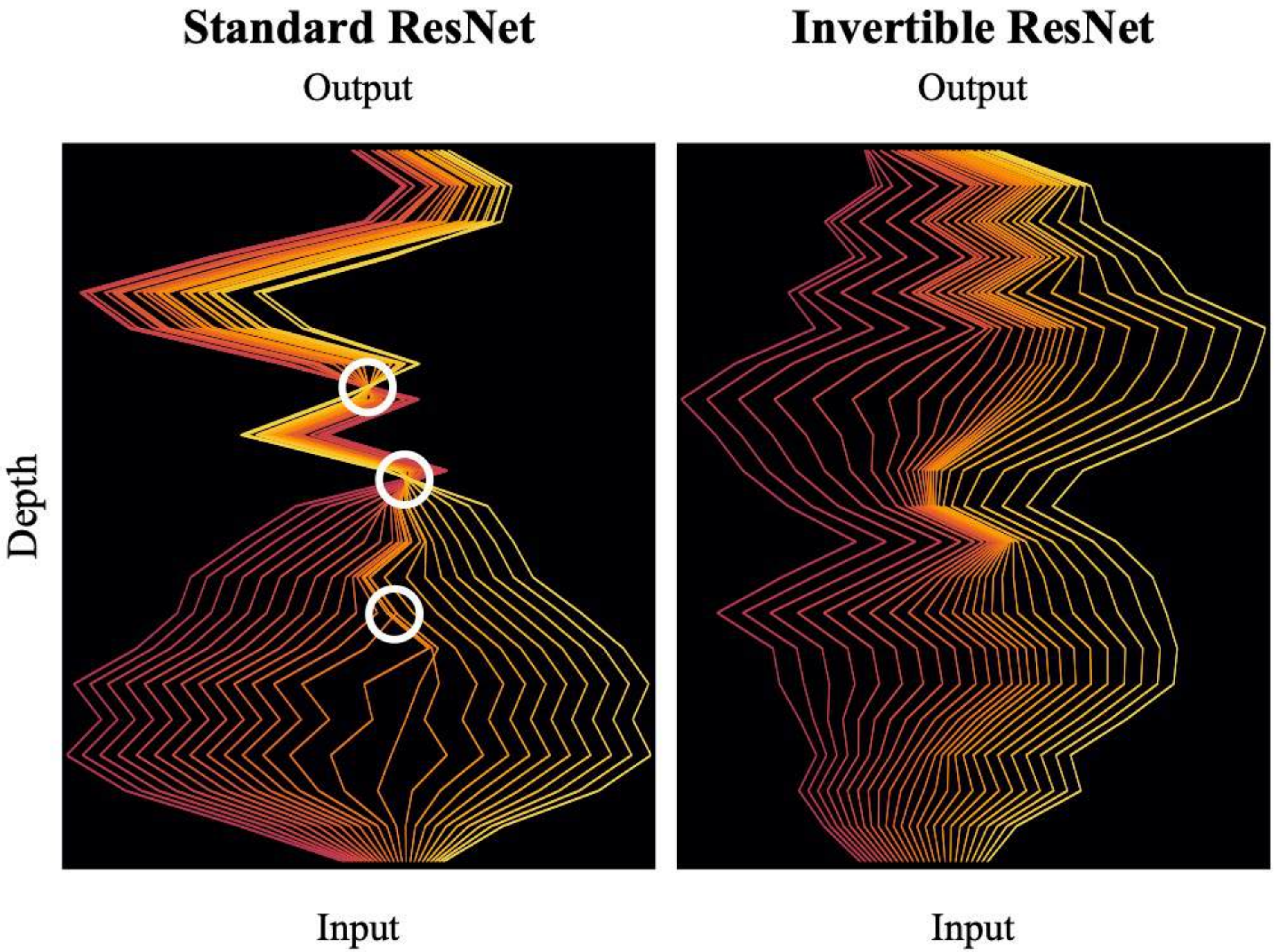
- ResNet: Euler discretization of neural ODE, not invertible in general.
- How to make it invertible?

Theorem 1 (Sufficient condition for invertible ResNets). *Let $F_\theta : \mathbb{R}^d \to \mathbb{R}^d$ with $F_\theta = (F_\theta^1 \circ \ldots \circ F_\theta^T)$ denote a ResNet with blocks $F_\theta^t = I + g_{\theta_t}$. Then, the ResNet F_θ is invertible if*

$$\mathrm{Lip}(g_{\theta_t}) < 1, \textit{ for all } t = 1, \ldots, T,$$

where $\mathrm{Lip}(g_{\theta_t})$ *is the Lipschitz-constant of* g_{θ_t}.

“Invertible Residual Networks”


Standard ResNet
Output
Invertible ResNet
Output
Depth
Input
Input

iResNet

- Compute the inverse

Algorithm 1. Inverse of i-ResNet layer via fixed-point iteration.

Input: output from residual layer y, contractive residual block g, number of fixed-point iterations n
Init: $x^0 := y$
for $i = 0, \ldots, n$ **do**
  $x^{i+1} := y - g(x^i)$
end for

- Convergence: the smaller Lip(g) is, the faster the convergence is.

$$\|x - x^n\|_2 \leq \frac{\mathrm{Lip}(g)^n}{1 - \mathrm{Lip}(g)} \|x^1 - x^0\|_2.$$

iResNet

- For contractive activation functions(ReLU, ELU,tanh) and linear mappings, i. $e.\ g = W_i \phi(W_{i-1}x)$

$$\mathrm{Lip}(g) < 1, \quad \text{if } \|W_i\|_2 < 1,$$

- spectral normalization: power iteration to estimate spectral norm $\tilde{\sigma}_i$

$$\tilde{W}_i = \begin{cases} cW_i/\tilde{\sigma}_i, & \text{if } c/\tilde{\sigma}_i < 1 \\ W_i, & \text{else} \end{cases}$$

iResNet

- change of variable

$$x = F(z) = z + g(z)$$
$$\log p(x) = \log p(z) + \log|\det(J_F)|$$

- For iResNet, $F = I + g \; is \; close \; to \; I, thus$

$$\log|\det(J_F)| = \log\det(J_F) = tr\,(\log J_F)$$

power series approximate $tr\,(\log J_F)$

$$tr\,(\log J_F) = tr\,(\log(I + J_g)) = \sum_{k=1}^{\infty}(-1)^{k+1}\,\frac{tr(J_g^k)}{k}$$

Monte-Carlo trace estimate approximates $tr(J_g^k)$

$$\mathrm{Tr}(A) = E_{p(\epsilon)}[\epsilon^T A \epsilon].$$

Algorithm 2. Forward pass of an invertible ResNets with Lipschitz constraint and log-determinant approximation, SN denotes spectral normalization based on (2).

Input: data point x, network F, residual block g, number of power series terms n

for Each residual block **do**

  Lip constraint: $\hat{W}_j := \mathrm{SN}(W_j, x)$ for linear Layer W_j.

  Draw v from $\mathcal{N}(0, I)$

  $w^T := v^T$

  $\ln \det := 0$

  for $k = 1$ **to** n **do**

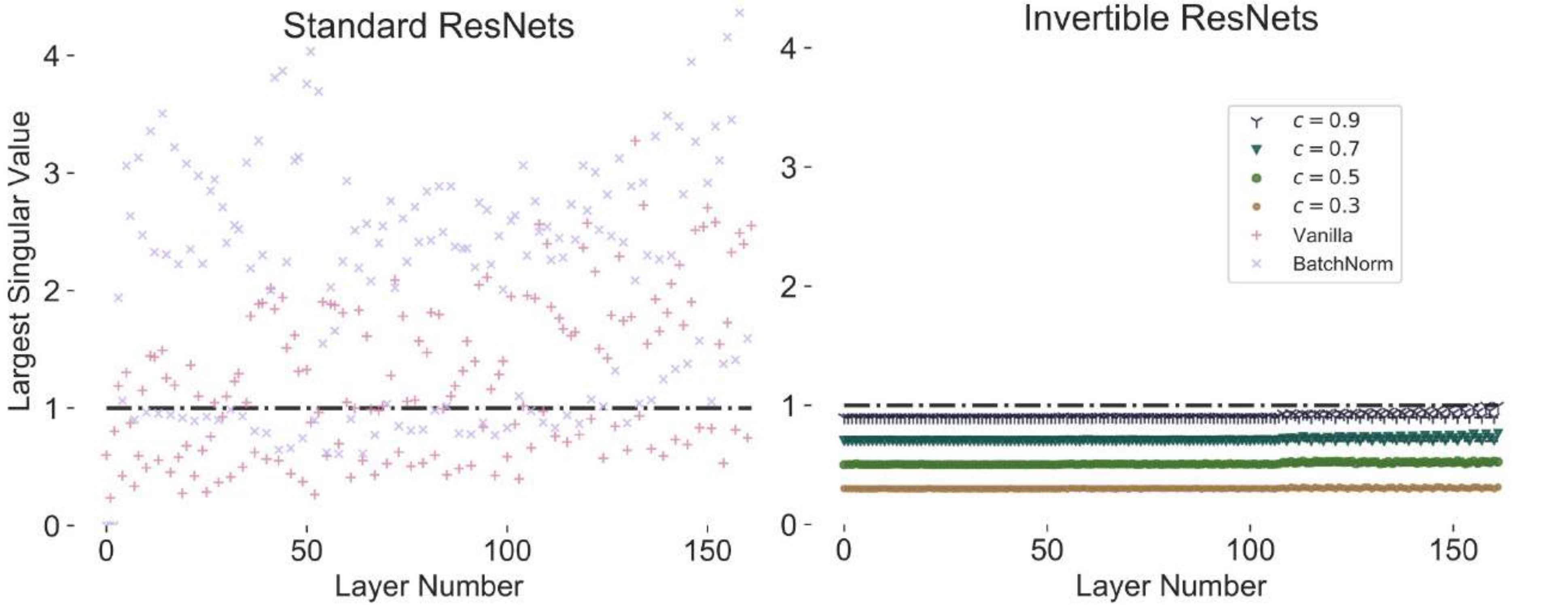
    $w^T := w^T J_g$ (vector-Jacobian product)

    $\ln \det := \ln \det + (-1)^{k+1} w^T v / k$

  end for

end for

iResNet



Running Time

Glow	i-ResNet	i-ResNet SN	i-ResNet SN LogDet
0.72 sec	0.31 sec	0.57 sec	1.88 sec

iResNet

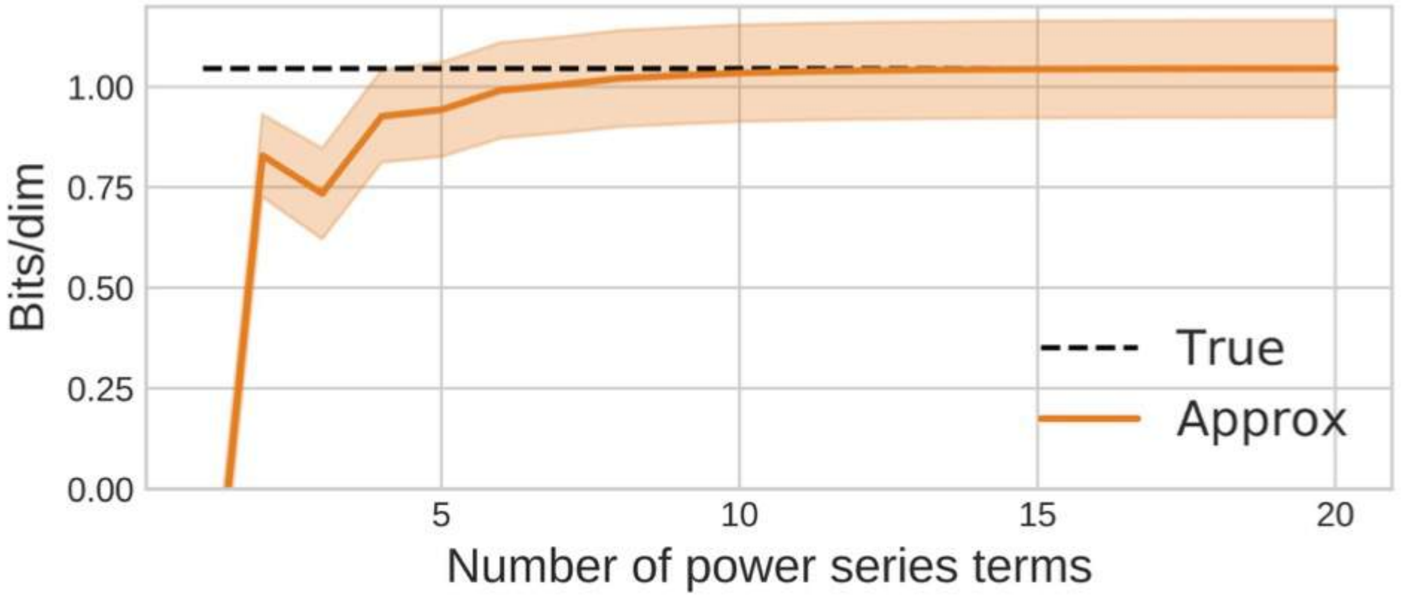
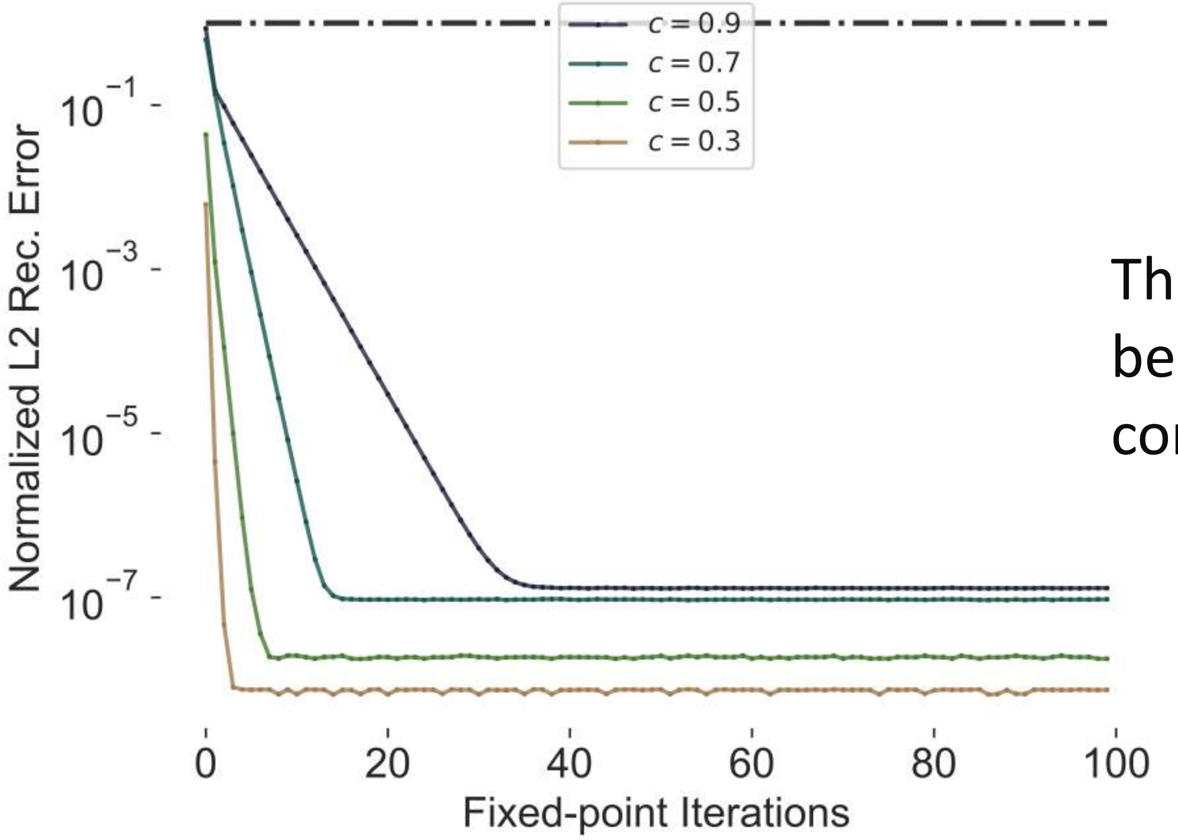


Figure 4. Bias and standard deviation of our log-determinant estimator as the number of power series terms increases. Variance is due to the stochastic trace estimator.

iResNet



The smaller c is, the smaller Lip(g) becomes, and the faster the convergence.

iResNet

CIFAR10 samples.

MNIST samples.

Comparison

Method	MNIST	CIFAR10
NICE (Dinh et al., 2014)	4.36	4.48†
MADE (Germain et al., 2015)	2.04	5.67
MAF (Papamakarios et al., 2017)	1.89	4.31
Real NVP (Dinh et al., 2017)	1.06	3.49
Glow (Kingma & Dhariwal, 2018)	1.05	3.35
FFJORD (Grathwohl et al., 2019)	0.99	3.40
i-ResNet	1.06	3.45

Table 4. MNIST and CIFAR10 bits/dim results. † Uses ZCA pre-processing making results not directly comparable.

Summary

Invertible Residual Networks

Method	ResNet	NICE/ i-RevNet	Real-NVP	Glow	FFJORD	i-ResNet
Free-form	✓	✗	✗	✗	✓	✓
Analytic Forward	✓	✓	✓	✓	✗	✓
Analytic Inverse	N/A	✓	✓	✗	✗	✗
Non-volume Preserving	N/A	✗	✓	✓	✓	✓
Exact Likelihood	N/A	✓	✓	✓	✗	✗
Unbiased Stochastic Log-Det Estimator	N/A	N/A	N/A	N/A	✓	✗

Table 1. Comparing i-ResNet and ResNets to NICE (Dinh et al., 2014), Real-NVP (Dinh et al., 2017), Glow (Kingma & Dhariwal, 2018) and FFJORD (Grathwohl et al., 2019). Non-volume preserving refers to the ability to allow for contraction and expansions and exact likelihood to compute the change of variables (3) exactly. The unbiased estimator refers to a stochastic approximation of the log-determinant, see section 3.2.