Normalizing Flow Models

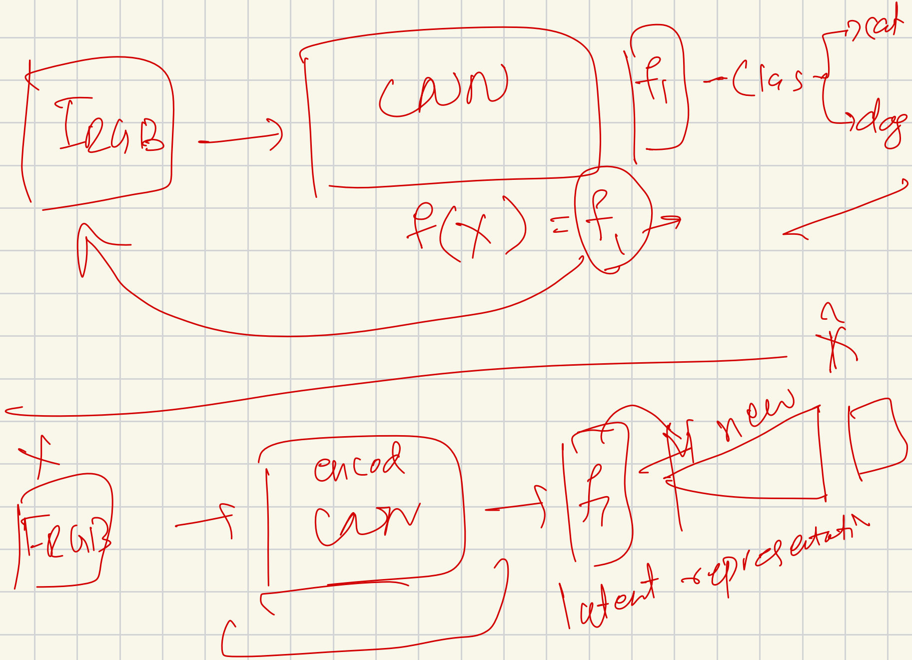
Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

June 2, 2025

Table of Contents

Table of Contents (cont.)



I RGB
CNN
f1
class
cat
dog
f(x) = f1
x̂
x
I RGB
encod
CNN
f1
new
latent representation

VAEs

Img → CNN → z → generate outputs

noise

$x \in \mathbb{R}^{d}, \quad z \in \mathbb{R}^{d'}$

$d' \ll d$

GAN

$$x \to \boxed{G} \xrightarrow{\hat{x}} \boxed{D} \to \text{real}$$

$$x \in \mathbb{R}^D \quad , \quad z \in \mathbb{R}^d$$

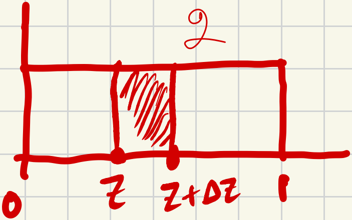
$$d \ll D$$

$$z \to \hat{x}$$

$$P_x(x)$$

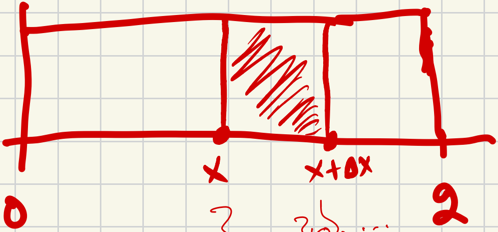
$f(z) = x = 2x$

4



$\underline{Z}$

1   1+2, ...



3   3+b, ...

$\underline{X}$

$$P_z(z) \cdot dz = P_x(x) \cdot dx$$

$$P_x(x) = P_z(z) \cdot \frac{dz}{dx}$$

$$P_x(x) = P_z(z) \cdot / \frac{dx}{dz}$$

↳ Image distribution.

Invertible

$$Z \longleftrightarrow X$$

$$X \longrightarrow Z$$

↳ samples (X) → to maximize the likelihood → (Z)

input

$$\boxed{X_n} \leftarrow \cdots \leftarrow X_2 \xleftarrow{f_2} X_1 \xleftarrow{f_1} X$$

↳ Z → pure white noise

$$\longrightarrow Z_n \xrightarrow{f_1} Z_2 \cdots\cdots Z_1 \longrightarrow \hat{X}$$

$$P_{X_1}(X_1)\cdot dx_1 = P_X(X)\cdot dx$$

$$P_{X_1}(X_1) = P_X(X)\cdot\frac{dx}{dx_1}$$

$$P_{X_2}(X_2) = P_{X_1}(X_1)\cdot\frac{dx_1}{dx_2}$$

$$P_{X_3}(X_3) = P_{X_2}(X_2)\,\frac{dx_2}{dx_3}$$

$$\vdots$$

$$P_{X_N}(X_N) = P_{X_{N-1}}(X_{N-1})\cdot\frac{dx_{N-1}}{dx_N}$$

$$P_{X_2}(X_2) = \underbrace{P_{X_1}(X_1)} \cdot \frac{dx_1}{dx_2}.$$

$$= \underbrace{P_X(X) \cdot \frac{dx}{dx_1} \cdot \frac{dx_1}{dx_2}}.$$

$$P_{X_3}(X_3) = P_X(X) \cdot \frac{dx}{dx_1} \cdot \frac{dX_1}{dX_2} \cdot \frac{dx_2}{dX_3}.$$

$$\vdots$$

$$P_{X_N}(X_N) = P_X(X) \cdot \underbrace{\frac{dx}{dx_1} \cdot \frac{dx_1}{dx_2} \cdots \frac{dx_{N-1}}{dx_N}}$$

$$P_{X_N}(X_N) = P_X(x) \prod_{i=1}^{N} \left(\frac{dx_{i-1}}{dx_i} \right)$$

$$= P_X(x) \prod_{i=1}^{N} \left(\frac{dx_i}{dx_{i-1}} \right)^{-1}$$

$$= P_X(x) \prod_{i=1}^{N} \left(f'(x_{i-1}) \right)^{-1}$$

$$P_{X_N}(X_N) = P_X(x) \prod_{i=1}^{n} \left| \left(f'_{x_{i-1}} \right)^{-1} \right|$$

$$P_Z(z^{\to x_N}) = P_X(x) \prod_{i=1}^{N} \left| \left(f'_{x_{i-1}} \right)^{-1} \right|$$

$$\frac{dx_{i-1}}{dx} \leftarrow f'_{x_i}$$

$$| \qquad \leftarrow dx_i$$

$$f(x,y) = \begin{bmatrix} f_1(x,y) = x^2 + y \\ f_2(x,y) = \sin xy \end{bmatrix}.$$

$$J(x,y) = \begin{bmatrix} \dfrac{\partial f_1}{\partial x} & \dfrac{\partial f_1}{\partial y} \\[2ex] \dfrac{\partial f_2}{\partial x} & \dfrac{\partial f_2}{\partial y} \end{bmatrix}$$

$$\frac{\partial}{\partial x}(x^2 + y) = 2x$$

$$\frac{\partial}{\partial y} x^2 + y = 1$$

$$\frac{\partial}{\partial x} \sin xy = y\cos xy.$$

$$\frac{\partial}{\partial y} \sin xy = x\cos xy.$$

$$J(x,y) = \begin{bmatrix} 2x & 1 \\ y\cos xy & x\cos xy \end{bmatrix}.$$

$x \in \mathbb{R}^d$

same

$X \in \mathbb{R}^{d \times d}$ $\qquad\qquad$ $Z \in \mathbb{R}^{d \times d}$

$$\frac{d f(x_1)}{d x_1} = \begin{bmatrix} \frac{dz_1}{dx_1} & \frac{dz_2}{dx_1} & \cdots & \frac{dz_d}{dx_1} \\ \frac{dz_1}{dx_2} & \frac{dz_2}{dx_2} & & \frac{dz_d}{dx_2} \\ \frac{dz_1}{dx_3} & \vdots & & \vdots \\ \vdots & & & \\ \frac{dz_1}{dx_d} & \frac{dz_2}{dx_d} & & \frac{dz_d}{dx_d} \end{bmatrix}$$

Jacobian Matrix

$O(n^3) \to$

Normilizig flow → UPPER/LOWER TRIANGULR MATRICES

$$z \rightleftarrows x$$

$$P_{\cancel{X}_n}(X_N) = P_X(X) \prod_{i=1}^{N} \left| \left(f'_{i-1} \right)^{-1} \right|$$

$$= \log P_X(X) \prod_{i=1}^{N} \mathrm{Det}\, J(T_\theta(\cdot))$$

$$= P_Z(Z) \prod_{i=1}^{N} \mathrm{Det}\, J(T_\theta(\cdot))^{-1}$$

log likelihood:

$$\log P_N(X_N) = \log\left(P_X(X) \cdot \cancel{\log} \prod_{i=1}^{N} \underbrace{\mathrm{Det}\, J(T_\theta(\ldots))}\right)$$

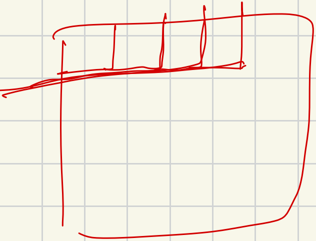
$$= \log P_X(X) + \log \prod_{i=1}^{N} \ldots$$

$$= \log P_X(X) + \sum_{i=1}^{N} \log \underbrace{\mathrm{Det}\, J(T_\theta)}$$

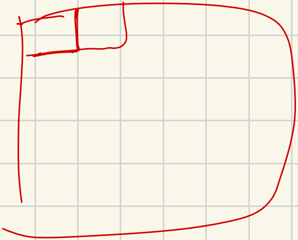
$$
\begin{bmatrix}
z_1 = f(x_1) & 0 & 0 & 0 & 0 & 0 \\
z_2 = f(x_1, x_2) & 0 & 0 & 0 & 0 & \\
z_3 = f(x_1, \underline{x_2, x_3}) & 0 & 0 & 0 & 0 & \\
\vdots & & & 0 & & \\
 & & & & - & \\
z_w = f(x_1, x_2, \dots, x_w) & & & & - &
\end{bmatrix}
$$

$$
P(z) = P_{x_1}(z_1) \cdot P_{x_1 x_2}(z_2 \mid z_1) \cdot P_{x_1 x_2 x_3}(z_3 \mid z_2, z_1
$$

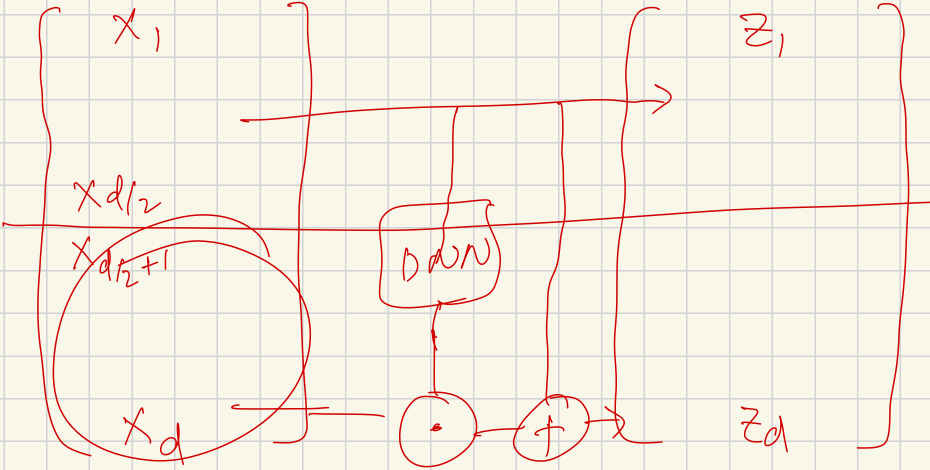
- - — — —



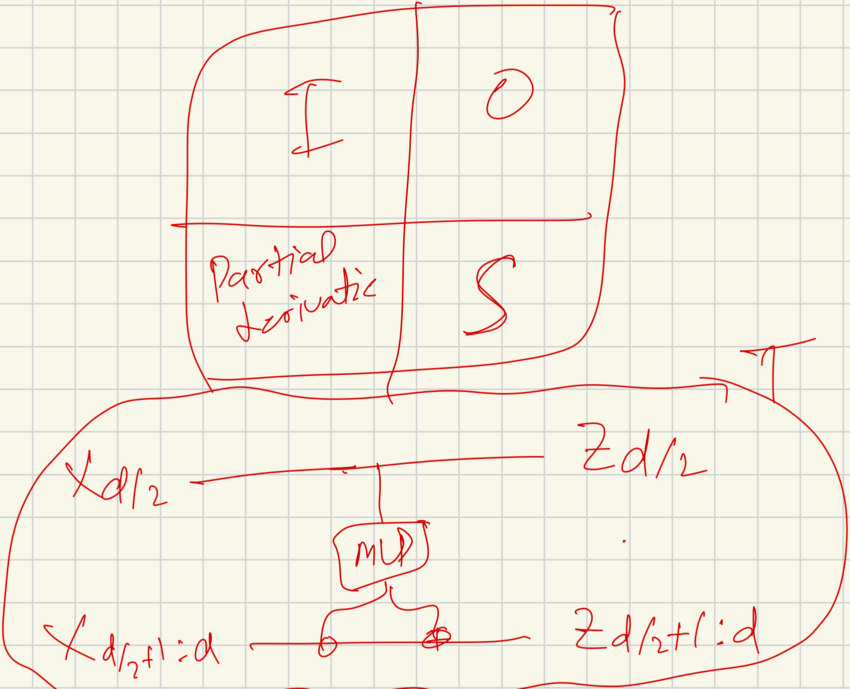
X

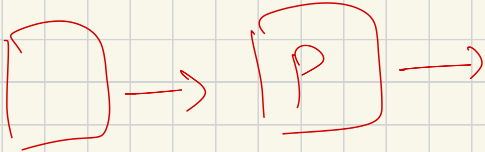


Z



x_1
$x_{d/2}$
$x_{d/2+1}$
x_d
DNN
$+$
z_1
z_d


I
0
Partial derivatic
S
T
$x_{d/2}$
$z_{d/2}$
MLP
$x_{d/2+1:d}$
$z_{d/2+1:d}$


P

$Z_{avg(old)}$ old

$Z - Z_{avg(old)} + Z_{avg(babies)}$

$Z_{avg(young)}$ young

$Z_{avg(babies)}$ babies

Table of Contents (cont.)

Normalizing Flow Models: Objectives

1. **Understand the concept of normalizing flows.**
   1.1 Learn how normalizing flows transform simple probability distributions into complex ones using invertible mappings.
2. **Explore the mathematical foundations.**
   2.1 Study the change of variables formula in probability.
   2.2 Analyze the role of the Jacobian matrix and its determinant in density estimation.
3. **Examine key architectures.**
   3.1 NICE (Non-linear Independent Components Estimation)
   3.2 RealNVP (Real-valued Non-Volume Preserving)
   3.3 Glow (Generative Flow with Invertible 1x1 Convolutions)
4. **Discuss practical applications and summarize key takeaways.**
   4.1 Review real-world applications of normalizing flows in generative modeling, density estimation, and more.
   4.2 Highlight the main insights and lessons from this topic.

Learning Outcomes: Normalizing Flow Models

After this lecture, you will be able to:

1. Explain the concept and purpose of normalizing flows in probabilistic modeling.
2. Describe how invertible mappings transform simple distributions into complex ones.
3. Apply the change of variables formula and compute the effect of the Jacobian determinant in density estimation.
4. Identify and compare key normalizing flow architectures such as NICE, RealNVP, and Glow.
5. Discuss practical applications of normalizing flows in generative modeling and density estimation.
6. Summarize the main insights and lessons related to normalizing flow models.

We continue on our quest for likelihood based generative model.

So far, we have studied two different type of generative model:

- **Autoregressive Models**: $p_\theta(x) = \prod_{i=1}^{N} p_\theta(x_i|x_{<i})$
  - Provide tractable likelihoods
  - But have no direct mechanism for learning features
  - Slow generation process
- **Variational Autoencoders**: $p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)$
  - Has intractable marginal likelihood
  - Can learn feature representation
  - We optimize the lower bound instead of maximizing the likelihood ... we don't know the gap between them

Question: Can we design a latent variable model with tractable likelihoods?

Answer: Normalizing Flow Models

- They combine the best of both worlds, allowing both feature learning and tractable marginal likelihood estimation.
- **Key Idea**: we wish to map simple distributions (easy to sample and evaluate densities) to complex ones (learned via data) using **change of variables**.

The change of variables formula describe how to evaluate densities of a random variable that is a deterministic transformation from another variable.

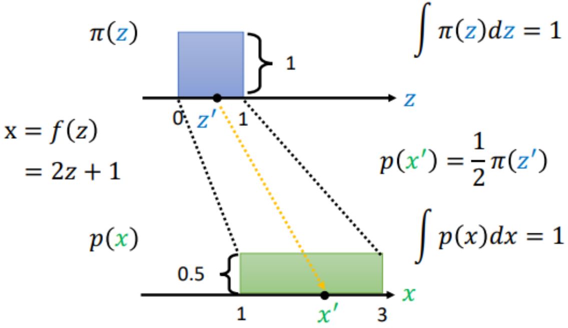


Figure 2: Transformation of a 1-D random variable to another random variable. Note that the areas of the blue and green rectangles are equal.

Change of Variables Theorem (cont.)

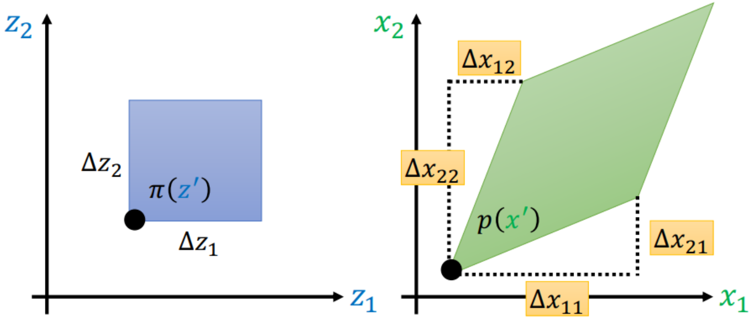


Figure 3: Transformation of a 2-D random variable to another random variable. Note that the areas of the blue and green rectangles are equal.

Change of Variables Theorem (cont.)

Theorem: Let Z and X be random variables related by a mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ such that $X = f(Z)$ and $Z = f^{-1}(X)$. Then,

$$p_X(x) = p_Z(f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

Some important points to note:

- x and z must be continuous random variables of the same dimension.
- $\frac{\partial f^{-1}(x)}{\partial x}$ is the $n \times n$ **Jacobian matrix**, where the entry at position (i, j) is $\frac{\partial [f^{-1}(x)]_i}{\partial x_j}$.
- For any invertible matrix A, $\det\left(A^{-1}\right) = [\det(A)]^{-1}$. Therefore, for $z = f^{-1}(x)$,

$$p_X(x) = p_Z(z) \left| \det \left(\frac{\partial f(z)}{\partial z} \right) \right|^{-1}$$

1) $$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

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

2) $$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} z_1 + z_2 \\ 2z_1 \end{bmatrix} = f\left(\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}\right)$$

$$\begin{bmatrix} x_2/2 \\ x_1 - x_2/2 \end{bmatrix} = f^{-1}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$$

3) $$J_f = \begin{array}{c} \text{input} \\ \begin{bmatrix} \partial x_1/\partial z_1 & \partial x_1/\partial z_2 \\ \partial x_2/\partial z_1 & \partial x_2/\partial z_2 \end{bmatrix} \end{array} \text{output}$$

$$J_{f^{-1}} = \begin{bmatrix} \partial z_1/\partial x_1 & \partial z_1/\partial x_2 \\ \partial z_2/\partial x_1 & \partial z_2/\partial x_2 \end{bmatrix}$$

4) $$J_f = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$$

$$J_{f^{-1}} = \begin{bmatrix} 0 & 1/2 \\ 1 & -1/2 \end{bmatrix}$$

$$J_f J_{f^{-1}} = I$$

19

Figure 4: Example of a 2D Jacobian matrix. See Appendix ?? for more information.

Matrix Determinant

The determinant of a **square matrix** is a **scalar** that provides information about the matrix.

- 2 X 2

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$det(A) = ad - bc$$

$$\boxed{\begin{aligned} det(A) &= 1/det(A^{-1}) \\ det(J_f) &= 1/det(J_{f^{-1}}) \end{aligned}}$$

- 3 x 3

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}$$

$$det(A) = a_1 a_5 a_9 + a_2 a_6 a_7 + a_3 a_4 a_8 - a_3 a_5 a_7 - a_2 a_4 a_9 - a_1 a_6 a_8$$

Figure 5: Example of matrix determinant.

- A generator G is a network that maps a simple distribution (for example, a normal distribution) $\pi(z)$ to a complex data distribution $p_G(x)$, which aims to be as close as possible to the real data distribution $p_{\text{data}}(x)$.

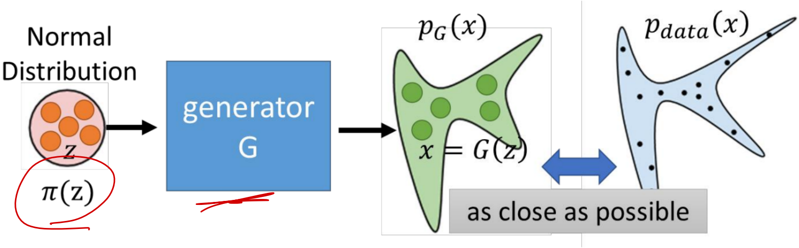
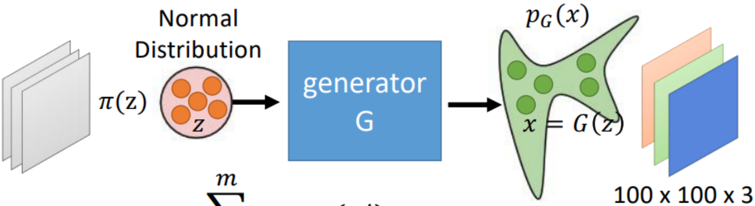


Figure 6: The goal of a generator network in a generative model

- Consider a directed latent-variable model with observed variables X and latent variables Z. In a normalizing flow model, the mapping between Z and X is deterministic and invertible: $G : \mathbb{R}^n \to \mathbb{R}^n$, such that $X = G(Z)$ and $Z = G^{-1}(X)$.
- By the change of variables formula, the marginal likelihood $p(x)$ is:

$$p_X(x;\theta) = p_Z(G_\theta^{-1}(x)) \left| \det \left(\frac{\partial G_\theta^{-1}(x)}{\partial x} \right) \right|$$

Normalizing Flow Models (cont.)



$$G^* = arg \max_G \sum_{i=1}^{m} log p_G(x^i)$$

G has limitation

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

$$z^i = G^{-1}(x^i)$$

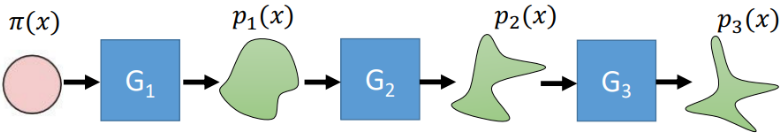
You can compute $det(J_G)$

You know G^{-1}

$$\boxed{log p_G(x^i) = log \pi\left(G^{-1}(x^i)\right) + log|det(J_{G^{-1}})|}$$

Normalizing Flow Models (cont.)

- The expressiveness of G is limited. To model more complex distributions, we need more expressive generators.



$$p_1(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right) \qquad z^i = G_1^{-1}\left(\cdots G_K^{-1}(x^i)\right)$$

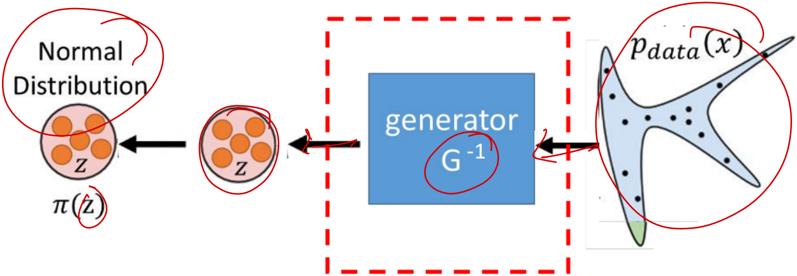
$$p_2(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)\left(\left|det\left(J_{G_2^{-1}}\right)\right|\right)$$

$$\vdots$$

$$p_K(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)\cdots\left(\left|det\left(J_{G_K^{-1}}\right)\right|\right)$$

$$log p_K(x^i) = log\pi(z^i) + \sum_{h=1}^{K} log\left|det\left(J_{G_K^{-1}}\right)\right|$$ Maximise

- In practice, we train G^{-1}, but use G for generation.



- **Normalizing**: The change of variables produces a normalized density after applying an invertible transformation.
- **Flow**: Invertible transformations can be composed to create more complex, expressive mappings.

- Learning is performed via maximum likelihood estimation over the dataset D:

$$\max_{\theta} \log p(D;\theta) = \sum_{x \in D} \left[\log \pi(G_\theta^{-1}(x)) + \log \left| \det \left(\frac{\partial G_\theta^{-1}(x)}{\partial x} \right) \right| \right]$$

- Exact likelihood evaluation is achieved using the inverse transformation and the change of variables formula.
- Sampling is performed via the forward transformation $G_\theta : Z \to X$:

$$z \sim \pi(z), \quad x = G_\theta(z)$$

- Latent representations are inferred via the inverse transformation (no inference network required):

$$z = G_\theta^{-1}(x)$$

Requirements for Flow Models

- A simple prior $\pi(z)$ that allows for efficient sampling and tractable likelihood evaluation (e.g., Gaussian).
- Invertible transformations.
- Computing likelihoods also requires evaluating the determinants of $n \times n$ Jacobian matrices, where n is the data dimensionality.
  - Computing the determinant of an $n \times n$ matrix is $O(n^3)$, which is prohibitively expensive within a learning loop.
  - Key idea: Choose transformations so that the resulting Jacobian matrix has a special structure. For example, the determinant of a triangular matrix is the product of its diagonal entries, making it an $O(n)$ operation.

Requirements for Flow Models (cont.)

- We need a fast and simple prior (e.g., Gaussian).
- The big question: How do we obtain invertible transformations and efficiently compute the determinants of Jacobians?
- **One solution**: Use coupling layers to design invertible functions and efficiently calculate the determinant of the Jacobian!

Coupling layers

Concept:

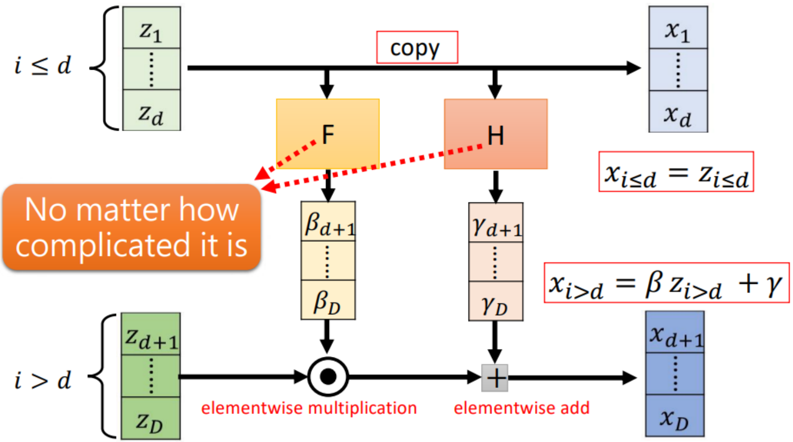
- Split the input $\mathbf{x}$ into two parts: $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]$.
- Apply a transformation to one part conditioned on the other:

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{x}_1 \\ \mathbf{y}_2 &= \mathbf{x}_2 \odot \exp\left(s(\mathbf{x}_1)\right) + t(\mathbf{x}_1) \end{aligned}$$

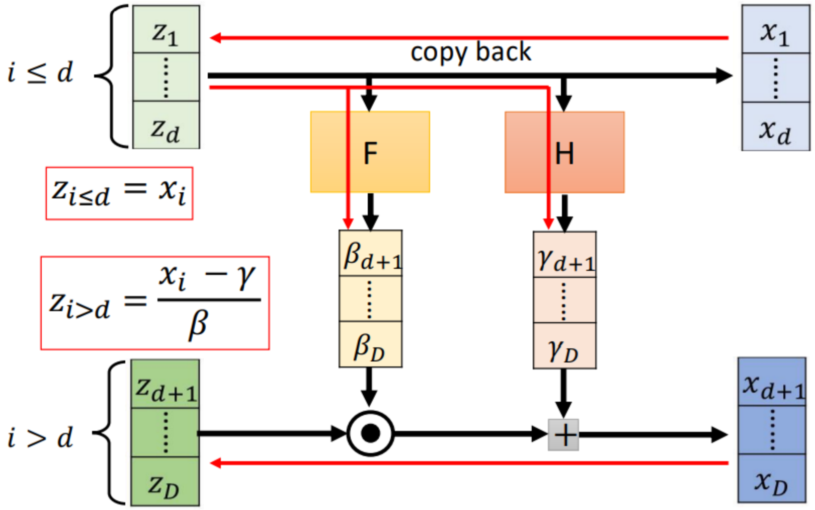
where s and t are scale and translation functions, respectively.

Advantages:

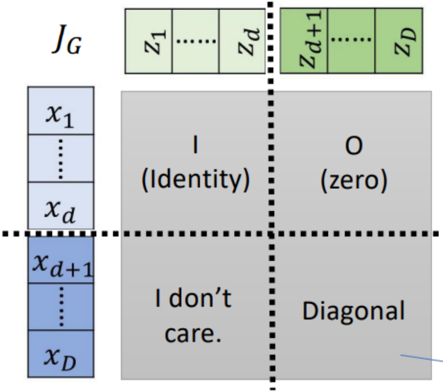
- Invertibility is straightforward.
- The Jacobian is triangular, making determinant computation efficient.
- Facilitates the design of complex, yet tractable, transformations.



$i \leq d$
z_1
z_d
copy
F
H
x_1
x_d
$x_{i \leq d} = z_{i \leq d}$
No matter how complicated it is
β_{d+1}
β_D
γ_{d+1}
γ_D
$x_{i>d} = \beta\, z_{i>d} + \gamma$
$i > d$
z_{d+1}
z_D
elementwise multiplication
elementwise add
x_{d+1}
x_D


$i \le d$
z_1
z_d
copy back
x_1
x_d
F
H
$z_{i \le d} = x_i$
β_{d+1}
β_D
γ_{d+1}
γ_D
$z_{i>d} = \frac{x_i - \gamma}{\beta}$
$i > d$
z_{d+1}
z_D
x_{d+1}
x_D

Coupling layers (cont.)



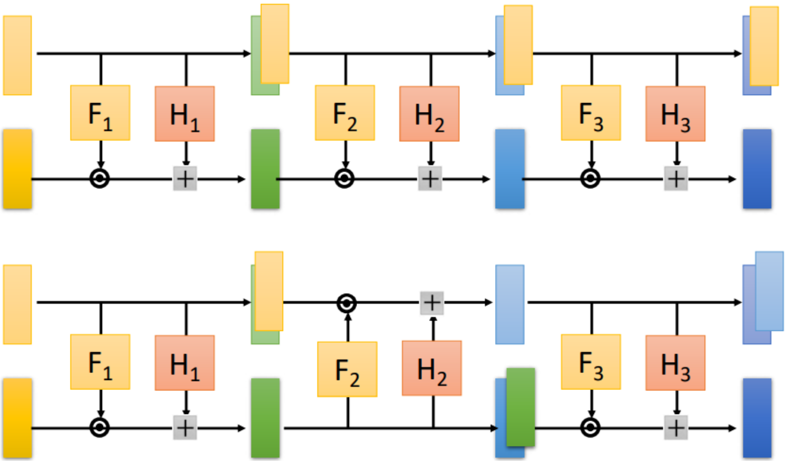
$$det(J_G)$$

$$= \frac{\partial x_{d+1}}{\partial z_{d+1}} \frac{\partial x_{d+2}}{\partial z_{d+2}} \cdots \frac{\partial x_D}{\partial z_D}$$

$$= \beta_{d+1} \beta_{d+2} \cdots \beta_D$$

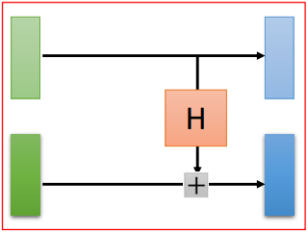
$$x_{i>d} = \beta z_{i>d} + \gamma$$

KAUST ACADEMY أكاديمية كاوست



NICE: Nonlinear Independent Components Estimation

- **Overview:** Introduced by Dinh et al. (2014), NICE employs additive coupling layers for constructing invertible transformations.
- **Additive Coupling Layers:**
  - Partition the variables z into two disjoint subsets:
  - $x_{1:d} = z_{1:d}$
  - $x_{d+1:n} = z_{d+1:n} + H(z_{1:d})$, where H is a neural network.



- The Jacobian determinant of this transformation is 1, which simplifies density computation.

NICE: Nonlinear Independent Components Estimation (cont.)

- Multiple additive coupling layers are composed together, with different partitions in each layer.
- **Limitation:** The transformation is volume-preserving and cannot model changes in volume, which limits expressiveness.
- **Enhancement:** A final rescaling layer is applied to introduce volume changes:

$$x = s \odot z$$

where s is a scaling factor.

NICE - Results

Figure 7: NICE generated samples when trained on the MNIST digits dataset.

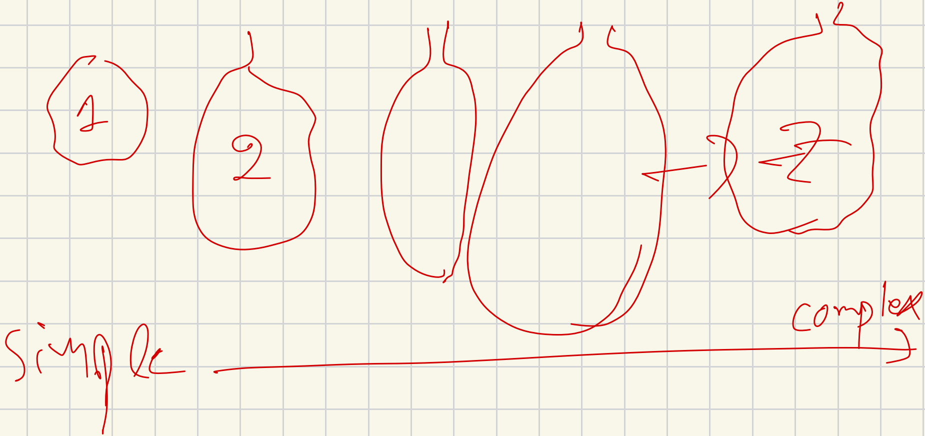
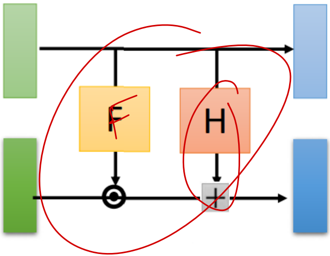

1
2
simple
complex

Figure 8: NICE generated samples when trained on the CIFAR-10 dataset.

RealNVP: Real-valued Non-Volume Preserving

- Enhancements over NICE:
  - Introduces scaling in coupling layers:
    - Partition the variables z into two disjoint subsets.
    - $x_{1:d} = z_{1:d}$
    - $x_{d+1:n} = z_{d+1:n} \odot \exp\left(F(z_{1:d})\right) + H(z_{1:d})$, where F and H are neural networks.



  - Allows modeling of volume changes, increasing flexibility.

Real NVP (cont.)

- **Benefits:**
  - Efficient computation of the Jacobian determinant due to the triangular structure.
  - Supports exact likelihood estimation and sampling.
- **Implementation Details:**
  - Alternating the roles of $z_{1:d}$ and $z_{d+1:n}$ across layers ensures all dimensions are transformed.
- Coupling layers are composed together (with arbitrary partitions of variables in each layer).

Real NVP - Results

Figure 9: Real NVP generated samples

Glow: Generative Flow

Innovations:

Introduced by Kingma and Dhariwal (2018), Glow incorporates:

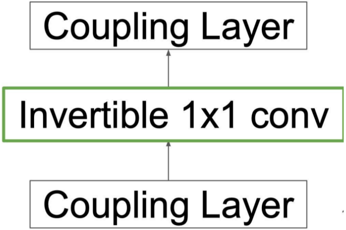
- **Invertible** 1×1 **convolutions** for channel mixing.
- **ActNorm layers** for data-dependent normalization.
- **Affine coupling layers** similar to RealNVP.

Advantages:

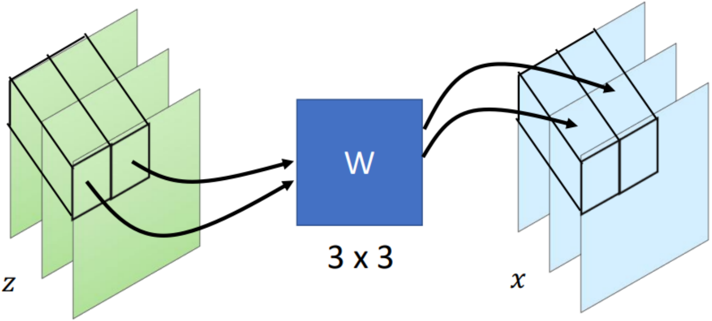
- Improved expressiveness and stability.
- Enhanced performance in image generation tasks.

Key Components:

- **ActNorm:** Applies per-channel affine transformation initialized with data statistics.
- **Invertible 1×1 Convolution:** Generalizes permutation operations, allowing learned channel mixing.
- **Affine Coupling Layers:** As in RealNVP, but integrated with the above components for greater flexibility.



Glow: Generative Flow (cont.)



W can shuffle the channels.

If W is invertible, it is easy to compute W^{-1}.

$$\begin{bmatrix}3\\1\\2\end{bmatrix} = \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix}$$

Figure 10: Glow generated samples

Glow - Results (cont.)

Figure 11: Linear interpolation in latent space between real images with Glow

Normalizing Flow Models - Summary

KAUST ACADEMY أكاديمية كاوست

Key Takeaways:

- Normalizing flows provide a powerful framework for modeling complex distributions with exact likelihoods.
- They transform simple base distributions into complex data distributions using invertible mappings.
- The change of variables formula and the Jacobian determinant are central to their mathematical foundation.
- Architectures such as NICE, RealNVP, and Glow illustrate the evolution and increasing sophistication of flow-based models.

Considerations:

- There are trade-offs between model expressiveness and computational efficiency.
- Designing transformations with tractable Jacobians is crucial for efficient learning and density estimation.

References

Reference Slides

- Fei-Fei Li, "Generative Deep Learning" (CS231)
- Hao Dong, "Deep Generative Models"
- Hung-Yi Lee, "Machine Learning"
- Murtaza Taj, "Deep Learning" (CS437)

Jacobian Matrix (2D Case)

Intuition Behind the Jacobian: In multivariable calculus, the Jacobian matrix describes how a transformation function changes space locally. Think of it as a local linear approximation to a nonlinear transformation. In 2D, it tells us how small changes in the input (x, y) affect the output (u, v).

Formal Definition: Let

$$f(x, y) = (u(x, y), v(x, y))$$

be a transformation from $\mathbb{R}^2 \to \mathbb{R}^2$. The Jacobian matrix is:

$$J_f(x, y) = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix}$$

Each entry measures how one output dimension changes with respect to one input dimension.

Geometric Interpretation:

The determinant of the Jacobian, $|\det J_f(x, y)|$, tells us how much area is stretched or compressed by the transformation at a specific point.

- $|\det J| > 1$: area expands.
- $|\det J| < 1$: area contracts.
- $|\det J| = 0$: local collapse (not invertible).

Example (Rotation + Scaling): Let

$$f(x, y) = (2x + y,\ x + 3y)$$

Then the Jacobian is:

$$J_f(x, y) = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

Determinant:

$$\det J = 2 \cdot 3 - 1 \cdot 1 = 6 - 1 = 5$$

Interpretation: Locally, this transformation scales area by a factor of 5.

In normalizing flows, we use:

$$p_X(x) = p_Z(f^{-1}(x)) \cdot \left| \det \left(\frac{\partial f^{-1}}{\partial x} \right) \right|$$

To compute the exact density after transformation, we must evaluate the Jacobian determinant. Hence, choosing transformations where the Jacobian (or its determinant) is easy to compute is critical (e.g., triangular matrices in coupling layers).

Visual Aid: For an interactive visualization of how the Jacobian affects area in 2D transformations, visit the Wolfram Demonstrations Project:

▸ 2D Jacobian Visualization

Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.