Normalizing Flow Models

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

June 23, 2025

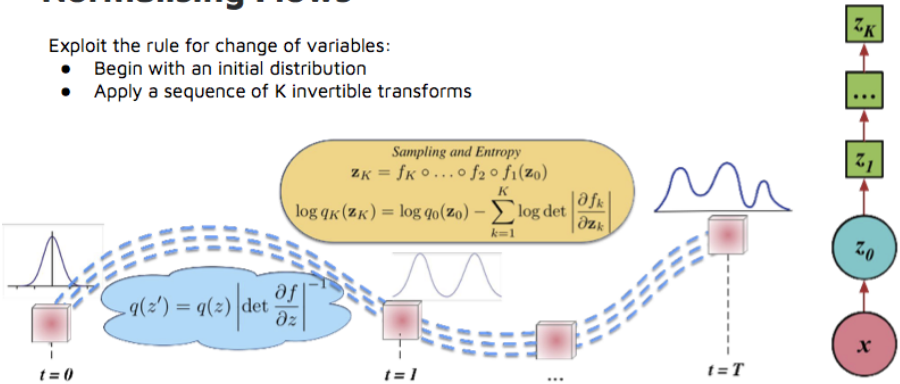
Table of Contents

Table of Contents (cont.)

Normalising Flows

Exploit the rule for change of variables:

- Begin with an initial distribution
- Apply a sequence of K invertible transforms



Distribution flows through a sequence of invertible transforms

Rezende and Mohamed, 2015

Motivation

- Probabilistic modeling is key to understanding data
- We want **flexible** models that can:
  - Generate data
  - Compute exact likelihoods
- Existing models trade off flexibility and tractability:
  - VAEs: Easy to train, blurry generations
  - GANs: Realistic samples, no likelihoods
  - Normalizing Flows: Both!

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.

Normalizing Flow Models: Objectives (cont.)

4.2 Highlight the main insights and lessons from this topic.

⇒ Learning Outcomes

By the end of this session, you will be able to:

1. Understand what Normalizing Flows are and why they matter
2. Apply change of variables in probability
3. Understand the role of Jacobians and determinants
4. Know how flow-based models like NICE, RealNVP, and Glow work
5. Evaluate benefits and limitations of normalizing flows

Normalizing Flow Models:

Introduction

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?

- **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**.

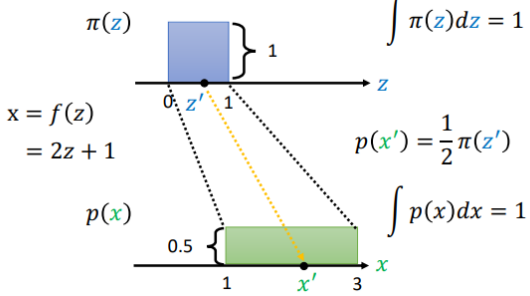
- A **normalizing flow** is a series of invertible transformations.
- It maps simple distributions (e.g., Gaussian) into complex ones.
- Let's say:
  - Sample z from a simple distribution.
  - Transform it to $x = f(z)$.
  - Want $p(x)$, i.e., how likely is x?

Normalizing Flow Models:

Change of Variables Theorem

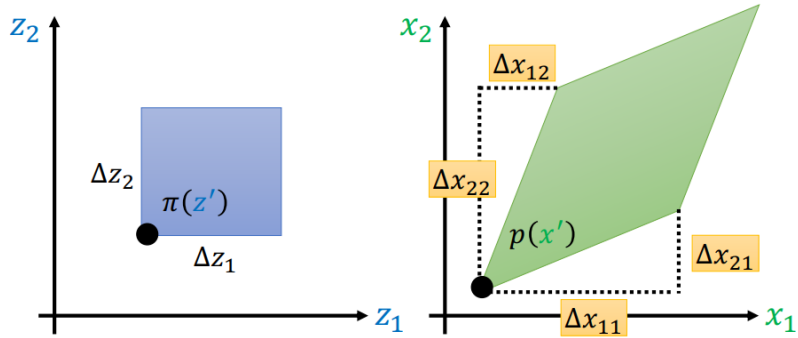
Change of Variables Theorem

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



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.)



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, $\det(A^{-1}) = [\det(A)]^{-1}$. Therefore, for $z = f^{-1}(x)$,

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

Change of Variables Theorem (cont.)

We need:

- **Invertibility of** f: The function f must be invertible so that for every x there exists a unique $z = f^{-1}(x)$.
- **Jacobian of** f: The Jacobian matrix $\frac{\partial f(z)}{\partial z}$ must be computable and its determinant non-zero almost everywhere to ensure the change of variables formula is valid.

Normalizing Flow Models:

Jacobian (2D Case)

Jacobian (2D Case)

- The Jacobian is a matrix of partial derivatives:

$$J_{ij} = \frac{\partial f_i}{\partial z_j}$$

- It describes how each component of $f(z)$ changes with z.
- The Jacobian is needed to compute how volumes change after transformation.

Jacobian (2D Case) (cont.)

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) $J_f = \overbrace{\begin{bmatrix} \partial x_1/\partial z_1 & \partial x_1/\partial z_2 \\ \partial x_2/\partial z_1 & \partial x_2/\partial z_2 \end{bmatrix}}^{\text{input}}$ 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) $J_f = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$

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

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

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

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

- 2 X 2

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

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

- 3 x 3

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

$$det(A) = a_1a_5a_9 + a_2a_6a_7 + a_3a_4a_8 - a_3a_5a_7 - a_2a_4a_9 - a_1a_6a_8$$

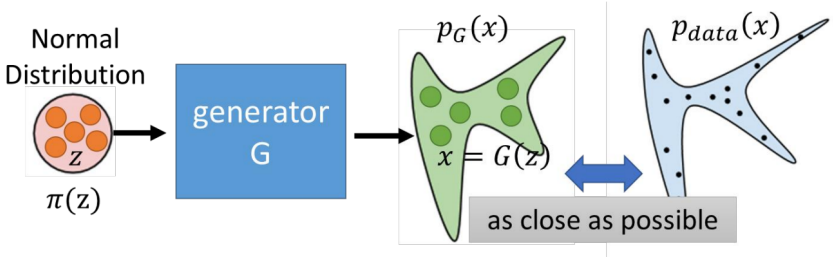
$$det(\mathrm{A}) = 1/det(A^{-1})$$

$$det(J_f) = 1/det(J_{f^{-1}})$$

.

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 $p_{\text{data}}(x)$.

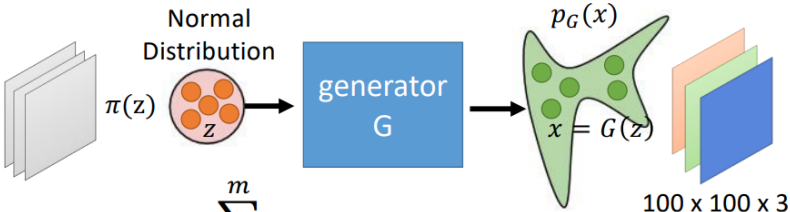


The goal of a generator network in a generative model

- 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.)



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

G has limitation

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

You can compute $det(J_G)$

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

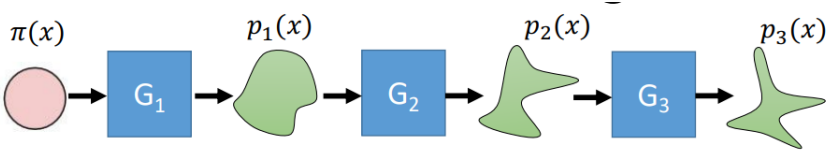
You know G^{-1}

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

Normalizing Flow Models (cont.)

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

Normalizing Flow Models (cont.)



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

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

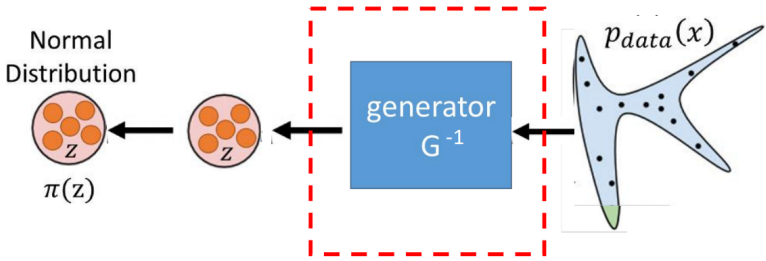
$$\vdots$$

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

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

Normalizing Flow Models (cont.)

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



Normalizing Flow Models (cont.)

- **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.

Normalizing Flow Models:

Learning and Inference

- 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):

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

Normalizing Flow Models:

Requirements

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!

Normalizing Flow Models:

Coupling layers

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:

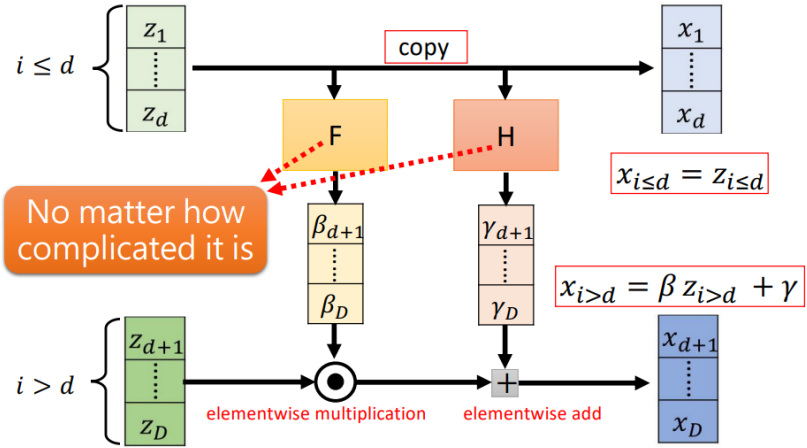
$$\mathbf{y}_1 = \mathbf{x}_1$$
$$\mathbf{y}_2 = \mathbf{x}_2 \odot \exp\left(s(\mathbf{x}_1)\right) + t(\mathbf{x}_1)$$

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

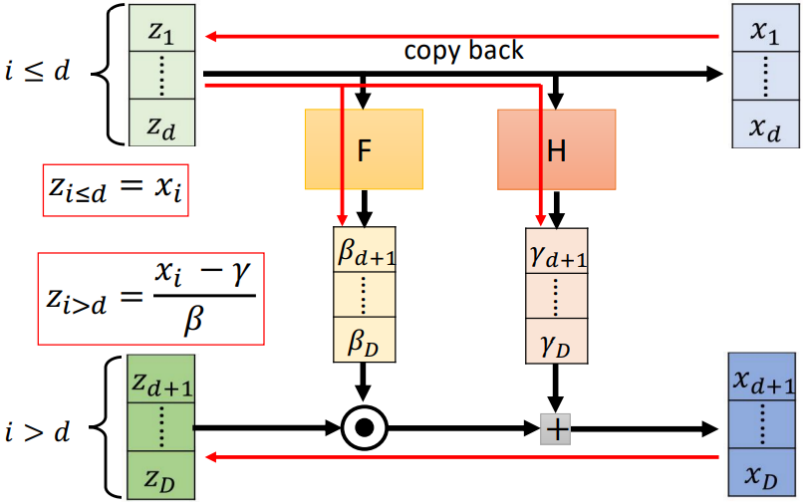
Advantages:

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

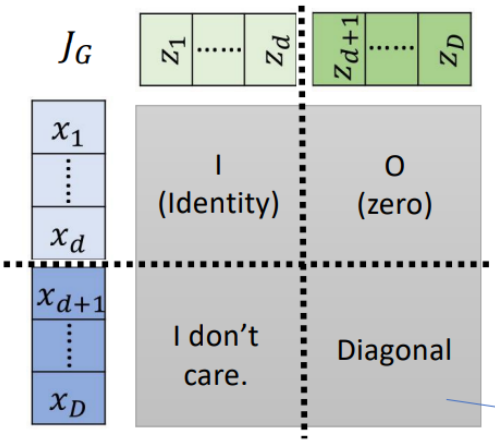
Coupling layers (cont.)



Coupling layers (cont.)



Coupling layers (cont.)



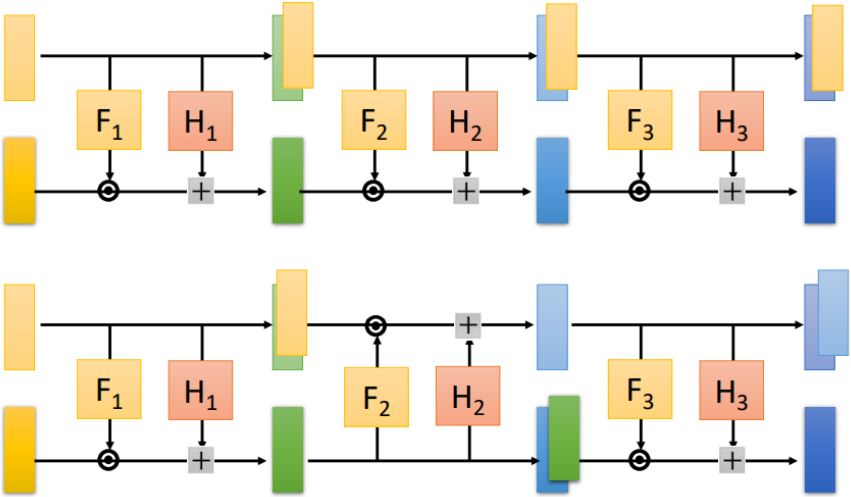
$$det(J_G)$$

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

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

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

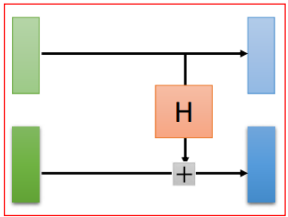
Coupling layers (cont.)



Normalizing Flow Models:

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.



NICE (cont.)

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

NICE (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

NICE generated samples when trained on the MNIST digits dataset.

NICE - Results (cont.)

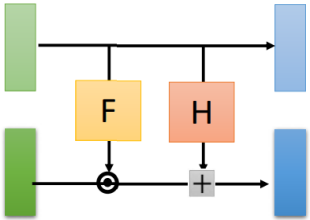
NICE generated samples when trained on the CIFAR-10 dataset.

Normalizing Flow Models:

Real NVP - Real-valued Non-Volume Preserving

RealNVP: Real-valued Non-Volume Preserving

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



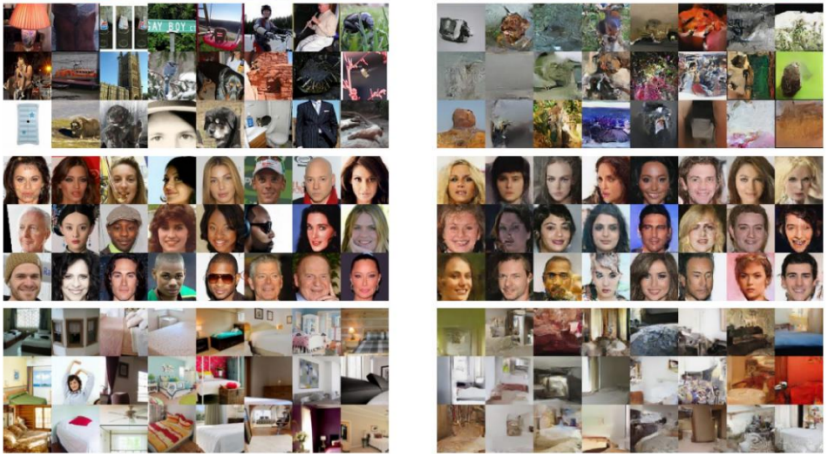
Real NVP (cont.)

- 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



Real NVP generated samples

Normalizing Flow Models:

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.

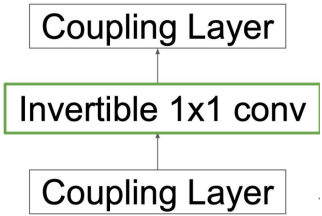
Advantages:

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

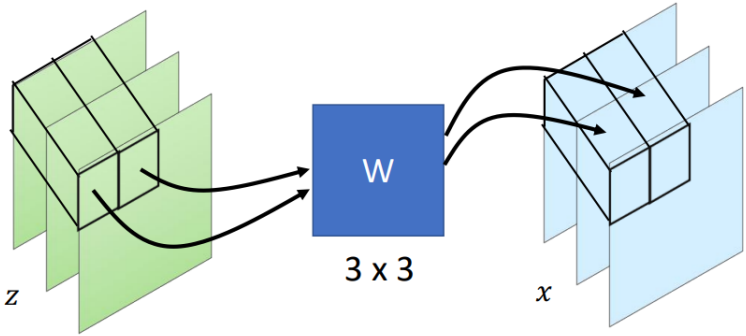
Glow (cont.)

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 (cont.)



W can shuffle the channels.

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

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

Glow - Results

Glow generated samples

Glow - Results (cont.)

Linear interpolation in latent space between real images with Glow

Normalizing Flow Models:

Summary

Normalizing Flow Models - Summary

What Did We Learn?

- Normalizing flows help us build flexible models for complex data.
- We start with a simple distribution and use invertible steps to make it match real data.
- The math uses the change of variables and the Jacobian determinant.
- Models like NICE, RealNVP, and Glow show how these ideas have improved over time.

Things to Keep in Mind:

- More flexible models can be harder and slower to train.
- We need to design steps that are both powerful and easy to compute.

Normalizing Flow Models:

Limitations

Limitations

- Need invertible functions (limits architecture choices)
- Jacobian computation still costly in general
- Not as expressive as autoregressive models in some cases
- Large models can be slow

Normalizing Flow Models:

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)

References (cont.)

[1] Dinh, L., Krueger, D., & Bengio, Y. (2014). NICE: Non-linear Independent Components Estimation. *arXiv preprint arXiv:1410.8516*.

[2] Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*.

[3] Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *Advances in Neural Information Processing Systems*, 31.

[4] Rezende, D. J., & Mohamed, S. (2015). Variational inference with normalizing flows. *International Conference on Machine Learning (ICML)*, 1530-1538.

[5] Weng, L. (2018). From GAN to WGAN. https://lilianweng.github.io/posts/2018-08-12-gan/

References (cont.)

[6] Kumar, A., Kim, S., & Poole, B. (2023). Maximum Likelihood Training of Score-Based Diffusion Models. *Advances in Neural Information Processing Systems*, 36.

[7] Hoogeboom, E., et al. (2022). Equivariant Discrete Normalizing Flows. *International Conference on Learning Representations (ICLR)*.

[8] Kim, S., et al. (2023). Flow Matching for Generative Modeling. *International Conference on Machine Learning (ICML)*.

[9] Lippe, P., Hoogeboom, E., & Welling, M. (2023). Flow Matching Models for Learning Score-Based Diffusions. *arXiv preprint arXiv:2306.00941*.

[10] Zhang, Y., et al. (2024). Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Normalizing Flow Models:

Appendix

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:

Appendix (cont.)

▸ 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.