Tractable Density Models and Normalizing Flows

Handout Notes - Generative and Deep Learning (GDL)

Davide Bacciu - University of Pisa

Notation. Observed random vectors are denoted by $\mathbf{X}$ and values by $\mathbf{x} \in \mathbb{R}^D$. Latent random vectors are denoted by $\mathbf{Z}$ and values by $\mathbf{z} \in \mathbb{R}^D$. A dataset is $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ with size $N = |\mathcal{D}|$. A base density is written $p(\mathbf{z})$, typically simple, such as $\mathcal{N}(\mathbf{0}, I)$. A flow transformation is $f : \mathbb{R}^D \to \mathbb{R}^D$, assumed invertible and differentiable, with inverse $g = f^{-1}$. For multi-step flows we write

$$\mathbf{z}_0 \xrightarrow{f_1} \mathbf{z}_1 \xrightarrow{f_2} \cdots \xrightarrow{f_K} \mathbf{z}_K = \mathbf{x}.$$

The Jacobian of a vector-valued map f at $\mathbf{u}$ is denoted by $J_f(\mathbf{u}) = \frac{\partial f(\mathbf{u})}{\partial \mathbf{u}}$. Model parameters are denoted by θ, with layer-specific parameters written θ_k when needed.

1 Why explicit likelihood models?

Generative models can be divided broadly into two families. Implicit models, such as GANs, learn a sampling mechanism without explicitly providing a tractable density. Explicit models instead aim to define or approximate a density

$$p_\theta(\mathbf{x})$$

that can be evaluated for data points.

Explicit likelihood models are attractive for several reasons. They provide a principled training objective through maximum likelihood, they support anomaly detection and density-based scoring, and they offer clearer statistical semantics. The price is that one must design a model whose density is either directly tractable or tractable up to a carefully controlled approximation.

Within explicit models, two important strategies are:

- **autoregressive factorization**, which writes the joint density through the chain rule;
- **change of variables**, which maps a simple base density into a complex data density through invertible transformations.

Normalizing flows belong to the second family.

2 Autoregressive density models as a point of departure

If all components of $\mathbf{x} = (x_1, \dots, x_D)$ are fully observed, the chain rule gives

$$p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i \mid x_{1:i-1}),$$

where $x_{1:i-1} = (x_1, \dots, x_{i-1})$. This is exact and needs no latent-variable approximation.

For text, this factorization is especially natural because token order is already given. For images, one may impose a scan order and model each pixel conditioned on preceding pixels. Historically, PixelRNN and related models implemented this idea by scanning the image and encoding previous pixels in recurrent or masked-convolutional states.

Autoregressive models are conceptually simple and likelihood-based, but they have an important weakness: generation is inherently sequential. To sample $\mathbf{x}$, one must generate one coordinate at a time:

$$x_1 \to x_2 \to \cdots \to x_D.$$

This makes sampling slow, especially for large images or long sequences.

Normalizing flows pursue a different compromise:

> *keep exact likelihood, but replace sequential conditional factorization with an invertible global transformation from a simple base density.*

3 From VAEs to invertible generative mappings

A variational autoencoder defines a latent-variable model with a stochastic encoder and decoder, but exact likelihood remains indirect because of latent marginalization. This raises a natural question:

> *Can we define a generative model that keeps the decoder direction, but gets rid of the approximate encoder by making the mapping exactly invertible?*

This is the starting point of normalizing flows. Suppose we can sample

$$\mathbf{z} \sim p(\mathbf{z}),$$

for a simple base density such as a standard Gaussian. Now let

$$\mathbf{x} = f(\mathbf{z}; \theta)$$

be an invertible, differentiable transformation. Then generation is easy:

1. sample $\mathbf{z}$ from the base density,
2. transform it forward through f to obtain $\mathbf{x}$.

If f is invertible, then we can also map an observed data point back into latent space:

$$\mathbf{z} = f^{-1}(\mathbf{x}; \theta).$$

This reverse map is the *normalizing* direction, because it converts a complicated data distribution into a simple latent one.

4 The intuition of normalizing flows

A normalizing flow learns a sequence of invertible transformations that gradually morph a simple density into a complex target density. If the base distribution is easy to sample and evaluate, and if each transformation is invertible with tractable Jacobian determinant, then the resulting model has two desirable properties:

- **sampling is easy**: sample in latent space, then push forward through the flow;
- **likelihood evaluation is exact**: pull a data point back to latent space and apply the change-of-variables formula.

Thus a flow is simultaneously a generator and a density model. This duality is one of its main strengths.

5 Change of variable in one dimension

The key mathematical tool is the change-of-variables formula. It is easiest to understand first in one dimension.

Let

$$z \sim p(z)$$

and define

$$x = f(z),$$

where f is invertible and differentiable. Probability mass must be conserved:

$$p(z)\,dz = p(x)\,dx.$$

Therefore,

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

Worked example — Linear one-dimensional change of variable

Let

$$z \sim \mathcal{N}(0,1)$$

and define the affine transformation

$$x = \mu + \sigma z, \qquad \sigma \neq 0.$$

The inverse is

$$z = \frac{x-\mu}{\sigma}.$$

By change of variables,

$$p(x) = p(z)\left|\frac{dz}{dx}\right| = p\left(\frac{x-\mu}{\sigma}\right)\frac{1}{|\sigma|}.$$

Since

$$p(z) = \frac{1}{\sqrt{2\pi}}e^{-z^2/2},$$

we obtain

$$p(x) = \frac{1}{|\sigma|\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right),$$

which is exactly the density of

$$x \sim \mathcal{N}(\mu, \sigma^2).$$

Thus the factor $\frac{1}{|\sigma|}$ accounts for the change in interval length induced by the transformation.

This example captures the essential idea: when a transformation stretches space, density decreases; when it compresses space, density increases.

6 Multidimensional change of variable

In multiple dimensions, interval length is replaced by volume. If

$$\mathbf{x} = f(\mathbf{z})$$

with $f : \mathbb{R}^D \to \mathbb{R}^D$ invertible and differentiable, then the density transforms as

$$p(\mathbf{x}) = p(\mathbf{z})\left|\det\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}\right|^{-1}, \qquad \mathbf{z} = f^{-1}(\mathbf{x}).$$

Equivalently,

$$p(\mathbf{x}) = p(f^{-1}(\mathbf{x})) \left| \det \frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right|.$$

The Jacobian determinant measures how the transformation changes local volume around a point. This is the multidimensional generalization of the scalar derivative.

Worked example — Why the Jacobian determinant appears

Consider a small region around $\mathbf{z}$. Under a differentiable map f, that region is approximately transformed linearly by the Jacobian

$$J_f(\mathbf{z}) = \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}.$$

A linear map changes volumes by a factor equal to the absolute value of its determinant. Hence

$$d\mathbf{x} = |\det J_f(\mathbf{z})| \, d\mathbf{z}.$$

Mass conservation then gives

$$p(\mathbf{x}) \, d\mathbf{x} = p(\mathbf{z}) \, d\mathbf{z},$$

so

$$p(\mathbf{x}) = p(\mathbf{z}) \, |\det J_f(\mathbf{z})|^{-1}.$$

This is the entire mathematical foundation of normalizing flows.

7 Compositions of invertible transformations

A practical flow is almost never a single transformation. Instead, it is a composition of simple invertible layers:

$$\mathbf{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0), \qquad \mathbf{x} = \mathbf{z}_K.$$

Because invertible differentiable maps are closed under composition, the whole flow remains invertible.

Applying the change-of-variables formula repeatedly yields

$$p(\mathbf{x}) = p(\mathbf{z}_0) \prod_{k=1}^{K} |\det J_{f_k}(\mathbf{z}_{k-1})|^{-1}.$$

Taking logs gives the numerically stable form used in learning:

$$\log p(\mathbf{x}) = \log p(\mathbf{z}_0) - \sum_{k=1}^{K} \log |\det J_{f_k}(\mathbf{z}_{k-1})|.$$

This expression is exact. Therefore maximum-likelihood learning is possible provided each layer satisfies three requirements:

- f_k is invertible;
- f_k^{-1} is computable efficiently;
- $\log |\det J_{f_k}|$ is tractable.

These requirements strongly shape flow architecture design.

8 Design desiderata for flow layers

A useful flow layer should balance expressivity and tractability. In practice, one wants:

- invertibility for all parameter values encountered during training;
- efficient forward computation for generation;
- efficient inverse computation for likelihood evaluation;
- easy computation of the log-determinant of the Jacobian;
- enough expressive power so that a stack of such layers can approximate complex densities.

This is the main architectural tension in normalizing flows. A fully general invertible neural network would have a dense Jacobian with expensive determinant and inverse. So scalable flow models use specially structured layers whose Jacobians are diagonal, triangular, or otherwise easy to handle.

9 Simple invertible layers

Before turning to modern coupling and autoregressive flows, it is useful to classify elementary invertible layers.

9.1 Affine linear layers

An affine transformation

$$f(\mathbf{z}) = \mathbf{b} + W\mathbf{z}$$

is invertible if and only if W is full rank. Then

$$J_f(\mathbf{z}) = W, \qquad \log|\det J_f| = \log|\det W|.$$

This is exact, but for unrestricted dense W both inversion and determinant computation can be costly.

9.2 Pointwise invertible nonlinearities

If

$$f(\mathbf{z}) = (f_1(z_1), \ldots, f_D(z_D))$$

with each scalar f_i invertible, then the Jacobian is diagonal:

$$J_f(\mathbf{z}) = \operatorname{diag}(f_1'(z_1), \ldots, f_D'(z_D)).$$

Hence

$$\log|\det J_f(\mathbf{z})| = \sum_{i=1}^{D} \log|f_i'(z_i)|.$$

This is efficient, but pointwise maps alone cannot create dependencies across dimensions.

9.3 Permutations

A permutation of coordinates is invertible and volume-preserving:

$$|\det J| = 1.$$

It does not change density volume, but it is useful between other flow layers because it mixes which variables interact.

These simple layers are rarely sufficient on their own, but they are the building blocks from which practical flows are assembled.

10 Coupling flows: the core idea

Coupling flows achieve a remarkable trade-off: they allow expressive nonlinear transformations while keeping inversion and Jacobian determinants easy.

Partition the input into two blocks:

$$\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2).$$

A coupling layer leaves one block unchanged and transforms the other block using parameters predicted from the unchanged part.

In its most general form,

$$\mathbf{z}_1' = \mathbf{z}_1, \qquad \mathbf{z}_2' = f\big(\mathbf{z}_2; \theta(\mathbf{z}_1)\big),$$

where $\theta(\mathbf{z}_1)$ is produced by a neural network. The unchanged block acts as conditioning information for transforming the other block.

Because $\mathbf{z}_1$ is copied directly, inversion is straightforward:

$$\mathbf{z}_1 = \mathbf{z}_1', \qquad \mathbf{z}_2 = f^{-1}\big(\mathbf{z}_2'; \theta(\mathbf{z}_1')\big).$$

11 NICE: additive coupling

The first influential coupling flow was NICE. Its additive coupling layer is

$$\mathbf{z}_1' = \mathbf{z}_1, \qquad \mathbf{z}_2' = \mathbf{z}_2 + t(\mathbf{z}_1),$$

where $t(\cdot)$ is a neural network.

The inverse is immediate:

$$\mathbf{z}_1 = \mathbf{z}_1', \qquad \mathbf{z}_2 = \mathbf{z}_2' - t(\mathbf{z}_1').$$

Worked example — Jacobian of an additive coupling layer

Consider

$$\mathbf{z}_1' = \mathbf{z}_1, \qquad \mathbf{z}_2' = \mathbf{z}_2 + t(\mathbf{z}_1).$$

Its Jacobian has block form

$$J = \begin{bmatrix} I & 0 \\ \frac{\partial t(\mathbf{z}_1)}{\partial \mathbf{z}_1} & I \end{bmatrix}.$$

This matrix is triangular, so its determinant is the product of diagonal-block determinants:

$$\det J = \det(I)\det(I) = 1.$$

Therefore

$$\log|\det J| = 0.$$

So an additive coupling layer is volume-preserving. Its likelihood contribution is trivial, and all density change comes only from other layers in the flow.

Additive coupling is computationally attractive, but it is somewhat limited because it cannot rescale volume locally. This motivates affine coupling.

12 RealNVP: affine coupling

RealNVP extends additive coupling by including both scale and shift:

$$\mathbf{z}_1' = \mathbf{z}_1, \qquad \mathbf{z}_2' = \exp(s(\mathbf{z}_1)) \odot \mathbf{z}_2 + t(\mathbf{z}_1).$$

Here $s(\cdot)$ and $t(\cdot)$ are neural networks, and the exponential ensures invertible positive scaling.

The inverse is

$$\mathbf{z}_1 = \mathbf{z}_1', \qquad \mathbf{z}_2 = \big(\mathbf{z}_2' - t(\mathbf{z}_1')\big) \odot \exp(-s(\mathbf{z}_1')).$$

Because scaling is diagonal in the transformed block, the Jacobian remains triangular and easy to handle.

Worked example — Log-determinant of an affine coupling layer

For

$$\mathbf{z}_1' = \mathbf{z}_1, \qquad \mathbf{z}_2' = \exp(s(\mathbf{z}_1)) \odot \mathbf{z}_2 + t(\mathbf{z}_1),$$

the Jacobian is

$$J = \begin{bmatrix} I & 0 \\ * & \operatorname{diag}(\exp(s(\mathbf{z}_1))) \end{bmatrix},$$

where $*$ denotes terms that do not affect the determinant. Hence

$$\det J = \det(I) \cdot \det\big(\operatorname{diag}(\exp(s(\mathbf{z}_1)))\big) = \prod_j \exp(s_j(\mathbf{z}_1)).$$

Therefore

$$\log|\det J| = \sum_j s_j(\mathbf{z}_1).$$

So affine coupling allows nontrivial local volume change while preserving tractability.

This is the main reason affine coupling became one of the standard building blocks of image flows.

13 Why coupling layers must be stacked

A single coupling layer transforms only part of the variables directly. If the same partition were used repeatedly, some coordinates would remain insufficiently mixed. Therefore practical flow architectures alternate partitions and include permutations or learned channel-mixing transformations between layers.

In image models, the partition is often implemented by masks:

- checkerboard masks at one resolution,
- channel-wise masks after reshaping operations.

This ensures that, across layers, every coordinate both conditions others and is itself transformed.

14 Multiscale flows, masking, and squeezing

Image flows must cope with high-dimensional tensors. RealNVP introduced a multiscale architecture that combines coupling layers with shape-rearrangement operations.

14.1 Masking

A binary mask selects which subset of variables is left unchanged and which subset is transformed. For images, checkerboard masks are natural at pixel level, while channel masks are natural after reshaping.

14.2 Squeezing

A squeeze operation trades spatial resolution for channel depth. For example, an input of shape

$$s \times s \times c$$

can be rearranged into

$$\frac{s}{2} \times \frac{s}{2} \times 4c.$$

This does not lose information; it only reorganizes it. The benefit is that subsequent coupling layers can operate over channels while still indirectly modeling local spatial dependencies.

14.3 Multiscale factorization

At certain depths, part of the representation is factored out and modeled directly by the base distribution, while the remaining part continues through deeper flow layers. This reduces computational cost and encourages hierarchical structure.

15 Glow and invertible 1×1 convolutions

Glow improves the mixing between channels by replacing fixed permutations with learned invertible 1×1 convolutions. At each spatial position, the channel vector is transformed linearly:

$$\mathbf{y}_{h,w} = W\,\mathbf{x}_{h,w},$$

where $W \in \mathbb{R}^{c \times c}$ is invertible.

This acts like a learned channel permutation and rotation. Because the same W is applied at every spatial position, the Jacobian determinant is

$$\log|\det J| = HW \log|\det W|,$$

where H and W here denote spatial height and width.

Glow parameterizes W using an LU decomposition so that determinant computation remains efficient. This is a good example of flow design philosophy: use expressive invertible transformations, but only in forms whose log-determinants remain cheap.

16 Autoregressive flows

Coupling flows transform one block conditioned on another block. Autoregressive flows push this idea further by treating each dimension as its own block.

A general autoregressive transformation has the form

$$x_i = f_i(z_i; \theta_i(z_{1:i-1})), \qquad i = 1, \ldots, D.$$

Each coordinate is transformed using parameters that depend on previous coordinates. The resulting Jacobian is triangular, because x_i depends only on $z_1, \ldots, z_i$. Therefore the determinant is again easy to compute.

This construction yields more expressive dependencies than simple two-block coupling, but it introduces an asymmetry: one direction is parallelizable and the other is sequential.

17 Masked autoregressive flow

Masked Autoregressive Flow (MAF) uses an autoregressive Gaussian transformation:

$$x_i = \mu_i + z_i \exp(s_i),$$

where

$$\mu_i = \mu_i(x_{1:i-1}), \qquad s_i = s_i(x_{1:i-1}).$$

Equivalently, the inverse is

$$z_i = (x_i - \mu_i) \exp(-s_i).$$

Because each z_i depends on previously reconstructed x_j, density evaluation is efficient in the inverse direction, while sampling is sequential in the forward direction.

Worked example — Triangular Jacobian in an autoregressive flow

Let

$$x_i = \mu_i(x_{1:i-1}) + z_i \exp(s_i(x_{1:i-1})).$$

Then the Jacobian $\frac{\partial \mathbf{x}}{\partial \mathbf{z}}$ is lower triangular, because x_i depends only on $z_1, \dots, z_i$. Its diagonal entries are

$$\frac{\partial x_i}{\partial z_i} = \exp(s_i).$$

Therefore

$$\det \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \prod_{i=1}^{D} \exp(s_i), \qquad \log \left| \det \frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right| = \sum_{i=1}^{D} s_i.$$

So the likelihood remains tractable even though the transformation is highly expressive.

This illustrates a recurring theme: flow layers are engineered so that their Jacobians are triangular or close enough to triangular that the determinant is cheap.

18 A practical asymmetry: parallel versus sequential directions

Autoregressive flows expose an important computational trade-off. For MAF:

- density evaluation is efficient, because the inverse map can be computed using a masked autoregressive network in parallel;
- sampling is slow, because coordinates must be generated sequentially.

For inverse autoregressive constructions, this trade-off is reversed.

Thus there is no universally best flow design. Some architectures are optimized for fast density estimation, others for fast sampling, and others for image-specific inductive biases.

19 Residual flows

So far, scalable flows achieved tractability by strongly constraining Jacobian structure. Residual flows ask whether one can move closer to full-rank, expressive Jacobians while remaining invertible.

Consider a residual layer

$$f(\mathbf{z}) = \mathbf{z} + h_\theta(\mathbf{z}).$$

If h_θ is Lipschitz with constant $L < 1$, then f is invertible. The intuition is that the residual update is small enough that the layer remains a perturbation of the identity.

The inverse generally has no closed form, but it can be recovered by fixed-point iteration:

$$\mathbf{z}^{(m+1)} = \mathbf{z}' - h_\theta(\mathbf{z}^{(m)}),$$

which converges under the contraction condition.

Worked example — Why a contractive residual map is invertible

Suppose

$$f(\mathbf{z}) = \mathbf{z} + h_\theta(\mathbf{z}),$$

and assume

$$\|h_\theta(\mathbf{u}) - h_\theta(\mathbf{v})\| \leq L\|\mathbf{u} - \mathbf{v}\|, \qquad L < 1.$$

To invert f, given $\mathbf{z}'$, define

$$T(\mathbf{z}) = \mathbf{z}' - h_\theta(\mathbf{z}).$$

Then

$$\|T(\mathbf{u}) - T(\mathbf{v})\| = \|h_\theta(\mathbf{v}) - h_\theta(\mathbf{u})\| \leq L\|\mathbf{u} - \mathbf{v}\|.$$

So T is a contraction. By the Banach fixed-point theorem, T has a unique fixed point $\mathbf{z}^*$, and that fixed point satisfies

$$\mathbf{z}^* = \mathbf{z}' - h_\theta(\mathbf{z}^*) \quad \Longleftrightarrow \quad \mathbf{z}' = \mathbf{z}^* + h_\theta(\mathbf{z}^*) = f(\mathbf{z}^*).$$

Hence the inverse exists and is unique.

This greatly expands the class of usable transformations, but introduces extra numerical complexity.

20 Likelihood for residual flows

For a residual flow

$$f(\mathbf{z}) = \mathbf{z} + h_\theta(\mathbf{z}),$$

the log-determinant term becomes

$$\log|\det J_f(\mathbf{z})| = \log\det(I + J_{h_\theta}(\mathbf{z})).$$

This is generally expensive to compute exactly. A common strategy is to use the series expansion

$$\log\det(I + A) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k}\mathrm{tr}(A^k),$$

valid under suitable spectral conditions. Applying this with $A = J_{h_\theta}(\mathbf{z})$ gives an expression in terms of traces, which can then be approximated efficiently using stochastic trace estimators such as Hutchinson's method.

This is a good example of the flow design principle evolving: earlier models used exactly tractable Jacobians by construction; residual flows accept more difficult Jacobians but approximate them carefully enough to remain practical.

21 Continuous normalizing flows

A residual flow can be seen as a discretized dynamical system:

$$\mathbf{z}_{t+\delta} = \mathbf{z}_t + \delta\, h_t(\mathbf{z}_t).$$

Letting $\delta \to 0$ leads to an ordinary differential equation:

$$\frac{d\mathbf{z}_t}{dt} = h_t(\mathbf{z}_t).$$

This is the basis of a continuous normalizing flow (CNF), also called a neural ODE flow.

Instead of composing finitely many discrete layers, one evolves the latent variable continuously in time from $t = 0$ to $t = T$. The corresponding change in log-density satisfies

$$\frac{d}{dt} \log p(\mathbf{z}_t) = -\mathrm{tr}\left(\frac{\partial h_t(\mathbf{z}_t)}{\partial \mathbf{z}_t}\right).$$

Integrating from 0 to T yields

$$\log p(\mathbf{x}) = \log p(\mathbf{z}_0) - \int_0^T \mathrm{tr}\left(\frac{\partial h_t(\mathbf{z}_t)}{\partial \mathbf{z}_t}\right) dt.$$

Worked example — From residual updates to a continuous flow

Start from the scaled residual update

$$\mathbf{z}_{t+\delta} = \mathbf{z}_t + \delta\, h_t(\mathbf{z}_t).$$

Rearrange:

$$\frac{\mathbf{z}_{t+\delta} - \mathbf{z}_t}{\delta} = h_t(\mathbf{z}_t).$$

In the limit $\delta \to 0$, this becomes the differential equation

$$\frac{d\mathbf{z}_t}{dt} = h_t(\mathbf{z}_t).$$

Thus a deep stack of small residual updates can be interpreted as an ODE trajectory in latent space. The associated density evolution replaces repeated Jacobian determinants by an integral of Jacobian traces.

CNFs are elegant and expressive, but they can be computationally expensive because both training and sampling require numerical ODE integration.

22 Dequantization

A pragmatic issue arises when flow models are applied to data that are discrete in practice, such as image pixels in $\{0, \ldots, 255\}$. The change-of-variables formula assumes continuous densities. A common solution is *dequantization*.

In uniform dequantization, one adds random noise:

$$\mathbf{x} = \mathbf{x}' + \mathbf{u}, \qquad \mathbf{u} \sim \mathrm{Uniform}([-0.5, 0.5]^D),$$

where $\mathbf{x}'$ is the original discrete-valued observation. This turns the data into a continuous random vector and avoids degenerate density behavior on a discrete grid.

Although this may look like a technical detail, it is essential for making continuous flow models well defined on image data.

23 Strengths and limitations of normalizing flows

Normalizing flows occupy a distinctive position among generative models.

23.1 Strengths

- exact and tractable log-likelihood,
- exact latent inference through the inverse map,
- straightforward sampling from the base density,
- smooth, structured latent spaces,
- strong theoretical flexibility: with sufficient capacity, flows can approximate very rich target densities.

23.2 Limitations

- architecture must remain invertible throughout learning,
- Jacobian determinants or trace terms can still be costly,
- exact likelihood evaluation may be much cheaper than in many other models, but not necessarily cheap in absolute terms,
- some flow designs generate slowly, others evaluate slowly,
- for images, flow sample quality has historically been strong but often less visually sharp than the best adversarial or diffusion models.

Thus flows are especially attractive when exact density, invertibility, and latent inference matter.

24 Conceptual summary

Normalizing flows provide an elegant answer to the problem of tractable generative modeling:

> *start from a simple density, transform it by a sequence of invertible maps, and use change of variables to obtain an exact likelihood on data space.*

The main ideas are:

- explicit likelihood is obtained through the Jacobian determinant of an invertible transformation;
- multi-step flows compose simple bijections into expressive global maps;
- practical flow layers are designed so that inversion and log-determinants are tractable;
- coupling flows achieve this with triangular Jacobians;
- autoregressive flows generalize the same idea coordinate by coordinate;
- residual and continuous flows relax Jacobian structure further, at the cost of more sophisticated approximation or ODE computation.

More broadly, flows sit at a particularly attractive point in the design space of generative deep learning: they retain exact density modeling like classical probabilistic methods, but they implement it with flexible neural transformations like modern deep architectures.