Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

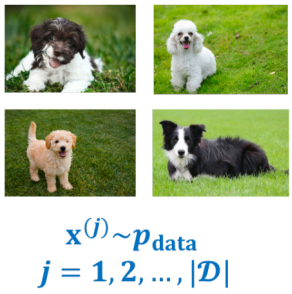
Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

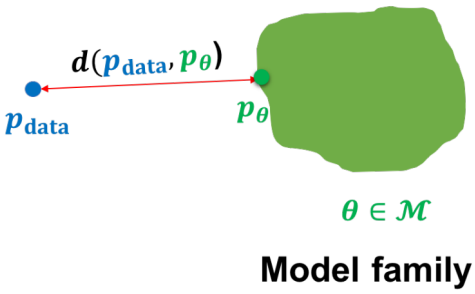
Thank you for your understanding and collaboration.

Content sourced from multiple references - see Reference section for details

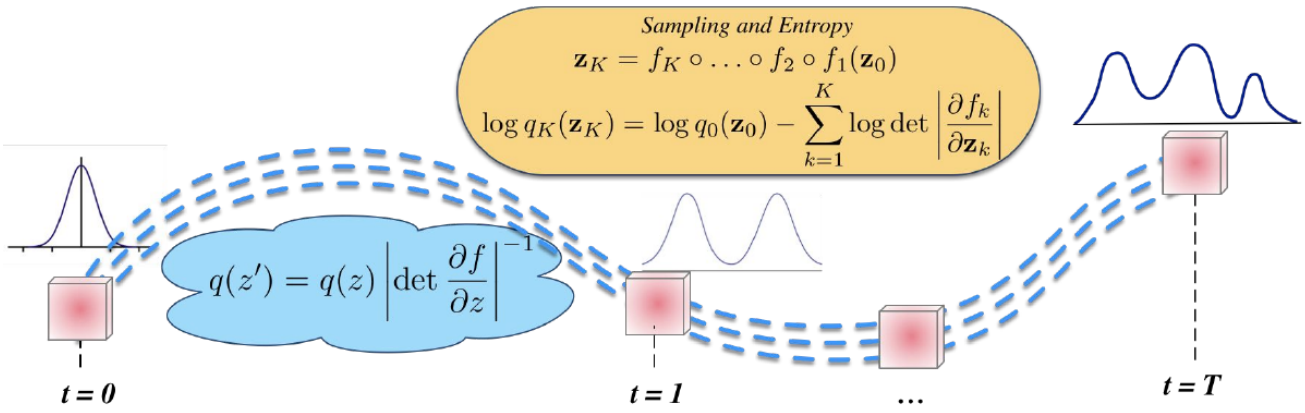
Normalizing Flow

1 Introduction





- Model families:
- **Autoregressive Models**:
  - $$p_\phi(\mathbf{x}) = \prod_{i=1}^{n} p_\phi(x_i | x_{<i})$$
  - Don't need to use variational inference.
  - Have direct access to the probability of the data (no need for encoder - decoder)
  - Provide tractable likelihoods but no direct mechanism. for learning features.
- **Variational Autoencoders**:
  - $$p_\phi(\mathbf{x}) = \int p_\phi(\mathbf{x}, \mathbf{z})\, d\mathbf{z}$$
  - You can define pretty flexible marginal distributions.
  - Can learn feature representations (via latent variables $\mathbf{z}$) but have intractable marginal likelihoods.
  - Can not evaluate the marginal probability $p_\phi(\mathbf{x})$ and need ELBO.
- **Key question**: Can we design a latent variable model with tractable likelihoods? **Yes! using Normalizing flows**



Distribution flows through a sequence of invertible transforms

2 Normalizing Flows

Normalizing flows are kind of latent variable model that learn a probability model by using a deep network to transform a simple distribution into a more complex one.

- They enable both sampling from the distribution and computing probabilities for new data without a need for variational inference.
- They are able to calculate the $p_\phi(\mathbf{x})$ directly which means we can use maximum likelihood for training them.

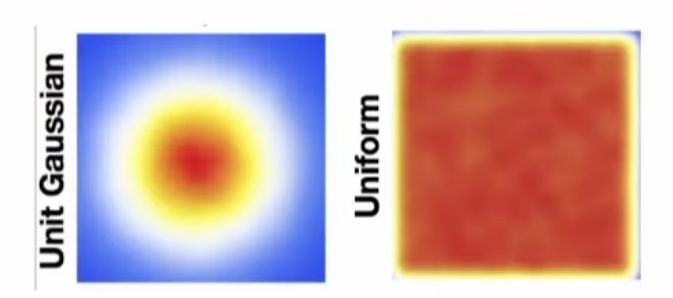
However, they require specialized architectures where each layer is invertible, allowing transformations to occur in both directions.

2.1 Main Idea

To have a model distribution $p_\phi(\mathbf{x})$ which is:

- Easy-to-evaluate, has closed form density (useful for training)
- Easy-to-sample - (for generation).

Many simple distributions satisfy the above properties e.g., Gaussian, uniform distributions.



However, data distributions could be much more complex (multi-modal) - similar when we used mixture-of-guassian models.

Key idea in flow models is to map simple distributions (easy to sample and evaluate densities) to complex distributions (learned via data) through an **invertible transformation**.

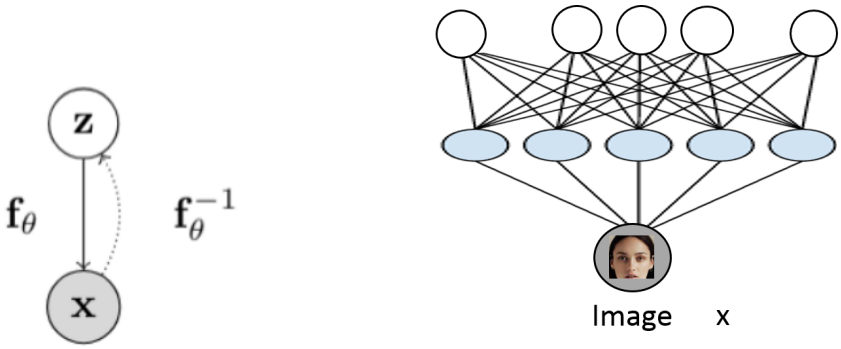


Figure 1: Deep latent variable models

This is essentially similar to VAE where:

- we have a latent variable $z \sim N(0, 1) = P(z)$
- We transform it via $p(x|z) = N(\mu_\phi(z), \Sigma_\phi(z))$ by passing z through some NNs.
- Even though $p(z)$ is simple, the $p_\phi(\mathbf{x})$ is very complex/flexible.
- The $p_\phi(\mathbf{x})$ was really expensive to compute as need iteration over all possible z that could have generated x.
- In VAE we essentially using encoder which trying to guess given a x which z is likely to produce it.

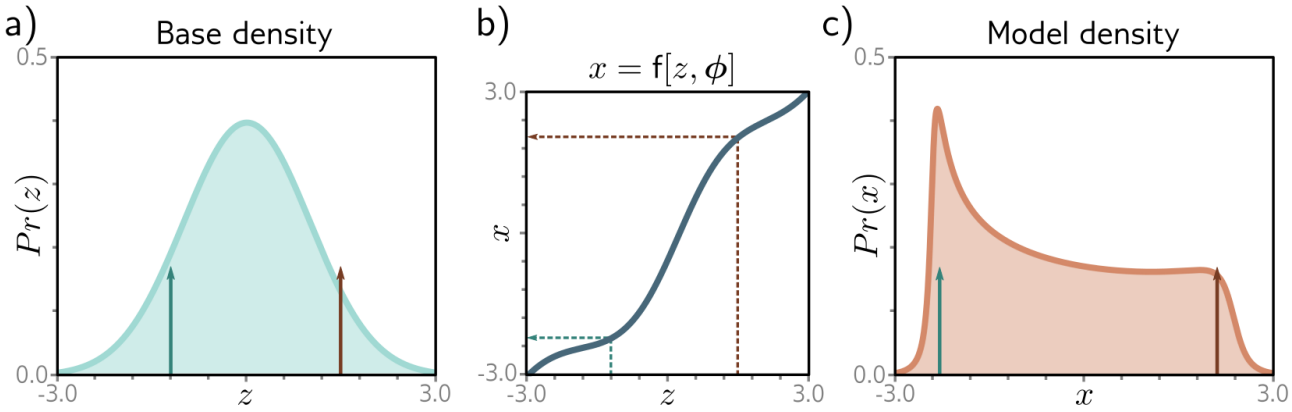
One way to avoid this process by design is to construct conditionals such that $p(x|z)$ can be easily inverted to compute $p(z|x)$. This can be achieved by defining $x = f_\phi(z)$ as a monotonic, deterministic and invertible function of z, ensuring that for every x, there is a unique corresponding z. Instead of passing z through neural networks to compute $\mu_\phi(z)$ and $\Sigma_\phi(z)$, we directly transform z using a single deterministic and invertible function.

This is the main ideas of **Flow models** which you can think of it as a VAE where the mapping between z to x is deterministic and invertible. although if we want the mapping to be invertible then z and x should have same dimension which is not the case in VAE.

3 1D Normalizing Flow

Normalizing flows are probabilistic generative models that fit a probability distribution to training data.

Consider modeling a 1D distribution $Pr(x)$. They start with a simple base distribution $Pr(z)$, typically a standard normal distribution, defined over a latent variable z. The transformation $x = f[z, \phi]$ is applied, where the parameters ϕ are chosen such that $Pr(x)$ matches the desired distribution. This process results in a new distribution for x.



To sample from this model:

- Draw values z from the base density (e.g., green and brown arrows in panel (a)).
- Pass these values through the transformation $f[z, \phi]$, as illustrated by dotted arrows in panel (b).
- Generate the corresponding values of x, shown as arrows in panel (c).

Generating new samples x is straightforward: draw z from the base distribution and compute $x = f[z, \phi]$.

3.1 Measuring Probability

Computing the probability of a data point x is more complex. Applying a function $f[z, \phi]$ to a random variable z with known density $Pr(z)$ affects the probability density.

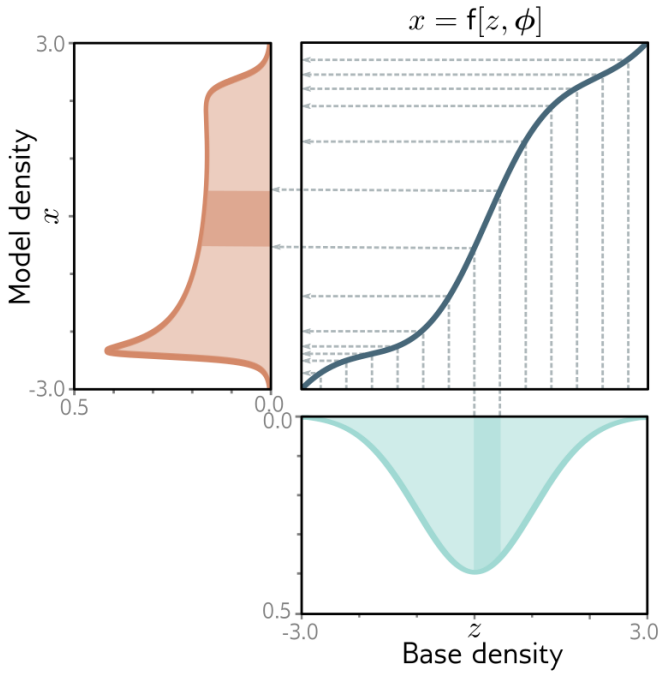
- It decreases in regions stretched by the function
- It increases in compressed regions

This is to ensure that the overall probability remains consistent and sums to one. The degree to which a function $f[z, \phi]$ stretches or compresses its input depends on the magnitude of its gradient. If a small change in the input causes a larger change in the output, the function stretches. Conversely, if a small change in the input causes a smaller change in the output, the function compresses.

The probability of data x under the transformed distribution is:

$$Pr(x|\phi) = \left| \frac{\partial f[z, \phi]}{\partial z} \right|^{-1} \cdot Pr(z),$$

where $z = f^{-1}[x, \phi]$ is the latent variable that generates x. The term $Pr(z)$ is the base probability of z, adjusted by the magnitude of the function's derivative. If the derivative is greater than one, the probability decreases; otherwise, it increases.



As shown in image above, The base density (cyan, bottom) passes through a function (blue curve, top right) to create the model density (orange, left). Consider dividing the base density into equal intervals (gray vertical lines). The probability mass between adjacent lines must remain the same after transformation. The cyan−shaded region passes through a part of the function where the gradient is larger than one, so this region is stretched. Consequently, the height of the orange−shaded region must be lower so that it retains the same area as the cyan−shaded region. In other places (e.g., $z = -2$), the gradient is less than one, and the model density increases relative to the base density.

3.2 Example

1. **Setup:**
   - Let Z be a uniform random variable, $Z \sim \mathcal{U}[0, 2]$, with a probability density $p_Z(z) = \frac{1}{2}$ for $z \in [0, 2]$.
   - Let $X = 4Z$, and denote the density of X as $p_X(x)$. We aim to compute $p_X(4)$.
2. **Naive Attempt:**
   - Suppose $p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1)$.
   - Since $p_Z(1) = \frac{1}{2}$, one might conclude $p_X(4) = \frac{1}{2}$, but this is **incorrect** because it does not account for the transformation.
3. **Correct Approach Using the Change of Variables Formula:**
   - The relationship $X = 4Z$ implies $Z = \frac{X}{4}$, so the inverse function is $h(X) = \frac{X}{4}$.
   - Using the change of variables formula for densities:
   $$p_X(x) = p_Z(h(x)) \cdot |h'(x)|$$
   where $h'(x)$ is the derivative of $h(x)$.
4. **Apply to This Case:**
   - Compute $h(X) = \frac{X}{4}$, so $h'(X) = \frac{1}{4}$.
   - Substitute into the formula:
   $$p_X(4) = p_Z\left(\frac{4}{4}\right) \cdot |h'(4)|$$
   $$p_X(4) = p_Z(1) \cdot \frac{1}{4}$$
   $$p_X(4) = \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}.$$

3.3 Example 2

- More interesting example: If $X = f(Z) = \exp(Z)$ and $Z \sim \mathcal{U}[0, 2]$, what is $p_X(x)$?
  - Note that $h(X) = \ln(X)$.
  - Using the change of variables formula:

$$p_X(x) = p_Z(\ln(x)) \cdot |h'(x)| = \frac{1}{2x}, \quad \text{for } x \in [\exp(0), \exp(2)].$$

- Note that the "shape" of $p_X(x)$ is different (more complex) compared to the prior $p_Z(z)$.

3.4 Change of Variables Formula (1D Case)

3.4.1 Key Idea

The change of variables formula is used to find the probability density function (PDF) of a transformed random variable. If $X = f(Z)$, where f is a monotone function with inverse $Z = f^{-1}(X) = h(X)$, the PDF of X is given by:

$$p_X(x) = p_Z(h(x)) \cdot |h'(x)|,$$

where $h'(x)$ is the derivative of the inverse function $h(x) = f^{-1}(x)$.

3.4.2 Steps to Derive the Formula

Step 1: Relating the cumulative distribution functions (CDFs) of X and Z:

$$F_X(x) = P[X \leq x] = P[f(Z) \leq x] = P[Z \leq h(x)] = F_Z(h(x)).$$

Step 2: Relating the PDFs by differentiating the CDF. Taking the derivative of $F_X(x)$ with respect to x:

$$p_X(x) = \frac{dF_X(x)}{dx} = \frac{dF_Z(h(x))}{dx}.$$

Using the chain rule:

$$p_X(x) = p_Z(h(x)) \cdot h'(x).$$

3.4.3 Derivative of the Inverse Function

To compute $h'(x)$, recall that f and f^{-1} are inverse functions, satisfying:

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

Differentiating $f(f^{-1}(x)) = x$ with respect to x using the chain rule:

$$\frac{d}{dx} f(f^{-1}(x)) = f'\left(f^{-1}(x)\right) \cdot \frac{d}{dx} f^{-1}(x).$$

Since the derivative of x is 1:

$$f'\left(f^{-1}(x)\right) \cdot h'(x) = 1.$$

This gives:

$$h'(x) = \frac{1}{f'\left(f^{-1}(x)\right)}.$$

3.4.4 Final Formula

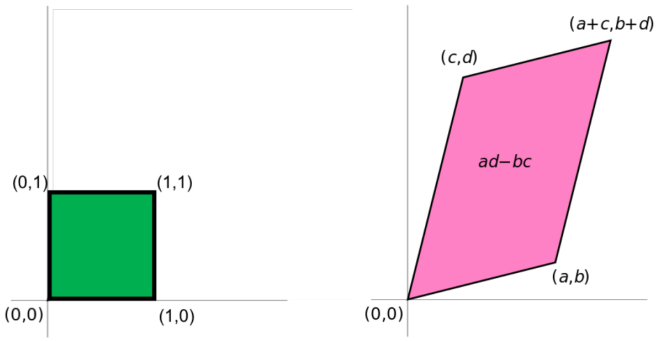
Let $z = h(x) = f^{-1}(x)$. Substituting $h'(x)$ into the PDF formula:

$$p_X(x) = p_Z(h(x)) \cdot h'(x) = p_Z(z) \cdot \frac{1}{f'(z)}.$$

3.5 Geometry Refresher

- Consider a uniform random vector Z defined in the unit hypercube $[0,1]^n$, which is the n-dimensional generalization of a square or cube.
- Let $X = AZ$, where A is a square invertible matrix, and $W = A^{-1}$ is its inverse. The question is: *How is X distributed?*
- Geometrically, the matrix A transforms the unit hypercube $[0,1]^n$ into a parallelotope, a generalized parallelogram in higher dimensions.
- Hypercubes and parallelotopes are higher-dimensional extensions of squares/cubes and parallelograms/parallelepipeds, respectively.

3.5.1 Illustration



- In 2D, the unit square $[0,1] \times [0,1]$ is mapped to a parallelogram by the matrix:

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

and

$$z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

- The vertices of the unit square $(0,0)$, $(1,0)$, $(0,1)$, and $(1,1)$ are transformed into:

$$(0,0), \quad (a,b), \quad (c,d), \quad (a+c, b+d).$$

- The area of the resulting parallelogram is given by $|\det(A)| = |ad - bc|$, which represents the volume scaling factor introduced by the transformation.

3.5.2 Matrix-Vector Multiplication

Given:

$$A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}, \quad Z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}, \quad X = AZ,$$

we compute:

$$X = AZ = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} az_1 + cz_2 \\ bz_1 + dz_2 \end{pmatrix}.$$

Thus:

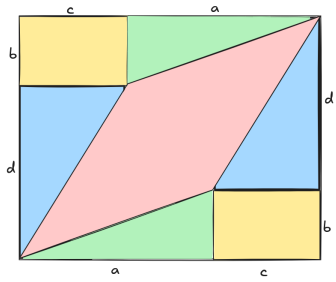
$$X_1 = az_1 + cz_2, \quad X_2 = bz_1 + dz_2,$$

where $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$ is the transformed vector.

3.6 Determinants and Volumes

The volume of the parallelogram is equivalent of absolute value of the determinant of the matrix A given by:

$$\text{Area} = |\det(A)| = |ad - bc|.$$



The determinant measures the scaling effect of the transformation, representing how the unit square is stretched or compressed.

Let $X = AZ$ for a square invertible matrix A, with inverse $W = A^{-1}$. X is uniformly distributed over the parallelotope of area $|\det(A)|$. Hence, we have:

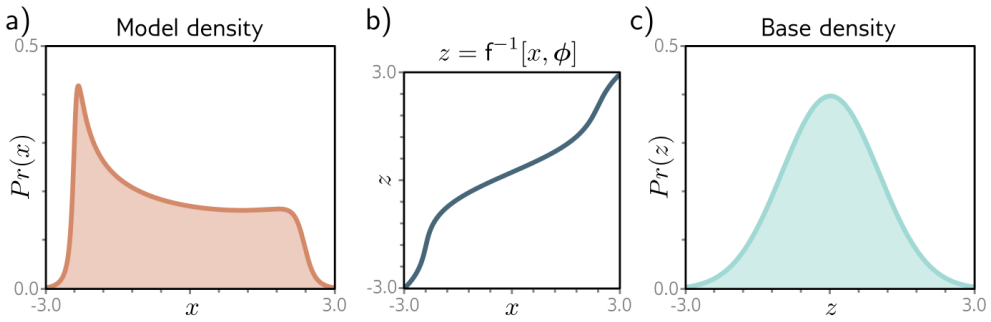
$$\begin{aligned} p_X(\mathbf{x}) &= p_Z(W\mathbf{x}) \cdot |\det(W)| \\ &= \frac{p_Z(W\mathbf{x})}{|\det(A)|} \end{aligned}$$

If $W = A^{-1}$ then $det(W) = \frac{1}{det(A)}$, This is similar to 1D case we saw earlier

3.7 Forward and Inverse Mappings

Normalizing flows use a forward mapping $x = f[z, \phi]$ to transform a base density (e.g., $Z \in$ standard normal) into a complex distribution. The inverse mapping $z = f^{-1}[x, \phi]$ is required to compute likelihoods, so $f[z, \phi]$ must be invertible.

- The forward mapping is also referred to as the **generative direction**, where the base density is typically a standard normal distribution.
- Conversely, the inverse mapping is called the **normalizing direction**, as it transforms the complex distribution over x into a simpler one (e.g., a normal distribution over z).



If the function is invertible, then it's possible to transform the model density back to the original base density. The probability of a point x under the model density depends partly on the probability of the equivalent point z under the base density

3.8 Learning

To learn the distribution, we optimize the parameters ϕ to maximize the likelihood of the training data $\{x_i\}_{i=1}^{I}$, or equivalently minimize the negative log-likelihood:

$$\begin{aligned}\hat{\phi} &= \arg\max_{\phi} \prod_{i=1}^{I} Pr(x_i|\phi), \text{(i.i.d. assumption)} \\ &= \arg\min_{\phi} \sum_{i=1}^{I} -\log Pr(x_i|\phi), \\ &= \arg\min_{\phi} \sum_{i=1}^{I} \left[\log\left[\left|\frac{\partial f[z_i,\phi]}{\partial z_i}\right|\right] - \log Pr(z_i)\right].\end{aligned}$$

4 General Case

The 1D example can be extended to multivariate distributions $Pr(\mathbf{x})$ by transforming a simpler base density $Pr(\mathbf{z})$ where $\mathbf{z} \in \mathbb{R}^D$ with a deep network $f[\mathbf{z}, \phi]$. The resulting variable $\mathbf{x} \in \mathbb{R}^D$ then follows a new distribution. To sample from this distribution:

- Draw $\mathbf{z}*$ from the base density $Pr(\mathbf{z})$.
- Pass $\mathbf{z}*$ through the deep network $f[\mathbf{z}, \phi]$ to compute $\mathbf{x}* = f[\mathbf{z}*, \phi]$.

The likelihood of $\mathbf{x}$ under this distribution is:

$$Pr(\mathbf{x}|\phi) = \left|\det \frac{\partial f[\mathbf{z},\phi]}{\partial \mathbf{z}}\right|^{-1} \cdot Pr(\mathbf{z}),$$

This formula explains how the probability density of the transformed variable $\mathbf{x}$ is computed from the base density $Pr(\mathbf{z})$ and the transformation $f[\mathbf{z}, \phi]$.

- **Latent Variable:** $\mathbf{z} = f^{-1}[\mathbf{x}, \phi]$ is the latent variable corresponding to $\mathbf{x}$. The inverse transformation f^{-1} allows us to map from $\mathbf{x}$ back to the base density space.
- For any invertible matrix A, $det(A^{-1}) = det(A)^{-1}$

- Both x and z should be continuous and have the same dimensions (unlike VAE).
- **Jacobian Determinant:** The term $\left|\det \frac{\partial f[\mathbf{z},\phi]}{\partial \mathbf{z}}\right|^{-1}$ is the inverse of the determinant of the Jacobian matrix. The Jacobian matrix J_f is defined as:

$$J_f(\mathbf{z}) = \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \frac{\partial f_1}{\partial z_2} & \cdots & \frac{\partial f_1}{\partial z_D} \\ \frac{\partial f_2}{\partial z_1} & \frac{\partial f_2}{\partial z_2} & \cdots & \frac{\partial f_2}{\partial z_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_D}{\partial z_1} & \frac{\partial f_D}{\partial z_2} & \cdots & \frac{\partial f_D}{\partial z_D} \end{bmatrix}.$$

  The determinant $\det(J_f)$ represents the local scaling factor of the transformation. Just as the absolute derivative measured the change of area at a point on a 1D function when the function was applied, the absolute determinant measures the change in volume at a point in the multivariate function.
- **Effect of the Inverse Determinant:** The inverse determinant is required because density is inversely proportional to volume. If the transformation stretches space (i.e., the determinant is large), the volume increases, and the density decreases. Conversely, if the transformation compresses space (i.e., the determinant is small), the volume decreases, and the density increases. This ensures that the total probability is conserved.

  Specifically, the probability density of $\mathbf{x}$ is scaled by the inverse of the determinant of the Jacobian:

$$Pr(\mathbf{x}) = Pr(\mathbf{z}) \cdot \left|\det(J_f(\mathbf{z}))\right|^{-1}.$$

- **Purpose:** This formulation ensures that the total probability is preserved during the transformation, maintaining consistency (i.e., the integral of the transformed density remains 1).

In summary, the formula combines the effects of the base density $Pr(\mathbf{z})$, the transformation $f[\mathbf{z},\phi]$, and the scaling factor $|\det(J_f)|^{-1}$, allowing us to compute the correct probability density $Pr(\mathbf{x}|\phi)$ for the transformed variable $\mathbf{x}$.

4.1 Two Dimensional Example

- Let Z_1 and Z_2 be continuous random variables with joint density p_{Z_1,Z_2}.
- Let $u = (u_1, u_2)$ be a transformation - Some sort of NNs.
- Let $v = (v_1, v_2)$ be the inverse transformation.
- Let $X_1 = u_1(Z_1, Z_2)$ and $X_2 = u_2(Z_1, Z_2)$. Then, $Z_1 = v_1(X_1, X_2)$ and $Z_2 = v_2(X_1, X_2)$.

For the inverse mapping:

$$p_{X_1,X_2}(x_1,x_2) = p_{Z_1,Z_2}\big(v_1(x_1,x_2), v_2(x_1,x_2)\big) \cdot \left|\det \begin{pmatrix} \frac{\partial v_1(x_1,x_2)}{\partial x_1} & \frac{\partial v_1(x_1,x_2)}{\partial x_2} \\ \frac{\partial v_2(x_1,x_2)}{\partial x_1} & \frac{\partial v_2(x_1,x_2)}{\partial x_2} \end{pmatrix}\right|$$

Start with the output variables x_1 and x_2, and compute their preimage (inverse) using the inverse transformation v. where:

- **First term:** Represents the original density p_{Z_1,Z_2} evaluated at the inverse mapping of x_1 and x_2.
- **Second term:** Corrects the density by accounting for the local stretching or compression of space under the transformation, as measured by the determinant of the Jacobian matrix of v.

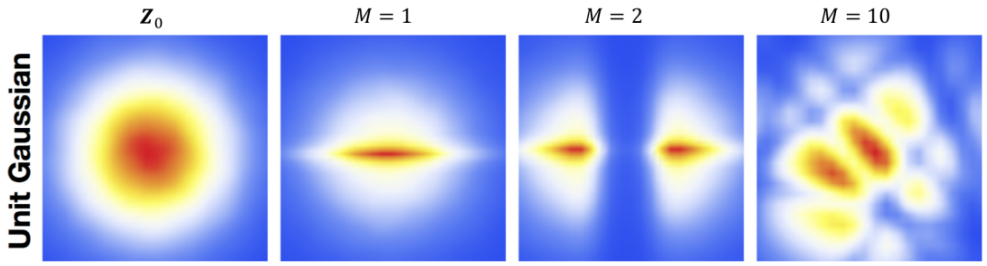
Similarly, for the forward mapping:

$$p_{Z_1,Z_2}(z_1,z_2) = p_{X_1,X_2}\big(u_1(z_1,z_2), u_2(z_1,z_2)\big) \cdot \left|\det \begin{pmatrix} \frac{\partial u_1(z_1,z_2)}{\partial z_1} & \frac{\partial u_1(z_1,z_2)}{\partial z_2} \\ \frac{\partial u_2(z_1,z_2)}{\partial z_1} & \frac{\partial u_2(z_1,z_2)}{\partial z_2} \end{pmatrix}\right|^{-1}$$

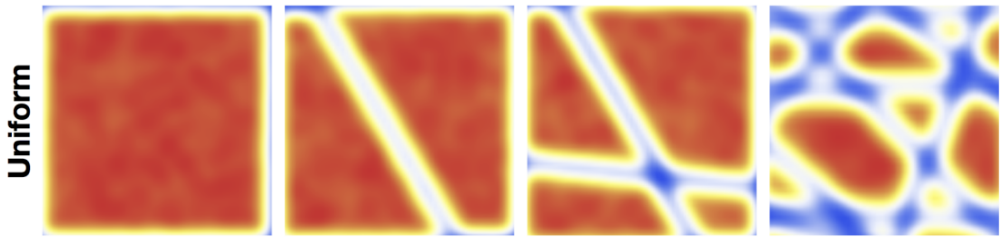
Start with the input variables z_1 and z_2, and map them to the output space using the forward transformation u. where:

- **First term:** Represents the original density p_{X_1,X_2} evaluated at the forward mapping of z_1 and z_2.
- **Second term:** Corrects the density by accounting for the local compression or stretching under the forward transformation, using the inverse of the determinant of the Jacobian matrix of u.

4.2 Forward Mapping with a Deep Neural Network



10 planar transformations (Planar flow) can transform simple distributions into a more complex one.



The forward mapping $f[z, \phi]$ is defined by a neural network as a composition of layers $\{f_k[\cdot, \phi_k]\}$ with parameters ϕ_k:

$$x = f[z, \phi] = f_K\big[f_{K-1}\big[\dots f_2\big[f_1[z, \phi_1], \phi_2\big], \dots, \phi_{K-1}\big], \phi_K\big].$$

The inverse mapping (normalizing direction) is the composition of the inverse of each layer, applied in reverse:

$$z = f^{-1}[x, \phi] = f_1^{-1}\big[f_2^{-1}\big[\dots f_{K-1}^{-1}\big[f_K^{-1}[x, \phi_K], \phi_{K-1}\big], \dots, \phi_2\big], \phi_1\big].$$

The Jacobian of the forward mapping can be expressed as:

$$\frac{\partial f[z, \phi]}{\partial z} = \frac{\partial f_K[f_{K-1}, \phi_K]}{\partial f_{K-1}} \cdot \frac{\partial f_{K-1}[f_{K-2}, \phi_{K-1}]}{\partial f_{K-2}} \cdots \frac{\partial f_2[f_1, \phi_2]}{\partial f_1} \cdot \frac{\partial f_1[z, \phi_1]}{\partial z}.$$

The absolute determinant of the Jacobian of the inverse mapping is the reciprocal of the absolute determinant of the forward mapping:

$$\left|\det\left(\frac{\partial f[z, \phi]}{\partial z}\right)\right|^{-1}.$$

4.3 Training Objective for Normalizing Flows

Normalizing flows are trained using a dataset $\{x_i\}$ of I samples to maximize the likelihood or equivalently minimize the negative log-likelihood:

$$
\begin{aligned}
\hat{\phi} &= \arg\max_{\phi} \prod_{i=1}^{I} Pr(z_i) \cdot \left| \det \frac{\partial f[z_i, \phi]}{\partial z_i} \right|^{-1} \\
&= \arg\max_{\phi} \prod_{i=1}^{I} Pr(z_i) \cdot \frac{1}{\left| \det \frac{\partial f[z_i, \phi]}{\partial z_i} \right|} \\
&= \arg\max_{\phi} \sum_{i=1}^{I} \log \left(Pr(z_i) \cdot \frac{1}{\left| \det \frac{\partial f[z_i, \phi]}{\partial z_i} \right|} \right) \quad \text{(taking log)} \\
&= \arg\max_{\phi} \sum_{i=1}^{I} \left(\log Pr(z_i) + \log \left(\frac{1}{\left| \det \frac{\partial f[z_i, \phi]}{\partial z_i} \right|} \right) \right) \\
&= \arg\max_{\phi} \sum_{i=1}^{I} \left(\log Pr(z_i) - \log \left| \det \frac{\partial f[z_i, \phi]}{\partial z_i} \right| \right) \quad \text{(log property: } \log(a^{-1}) = -\log(a)\text{)} \\
&= \arg\min_{\phi} \sum_{i=1}^{I} \left[\log \left| \det \frac{\partial f[z_i, \phi]}{\partial z_i} \right| - \log Pr(z_i) \right].
\end{aligned}
$$

Here: - $z_i = f^{-1}[x_i, \phi]$ is the latent variable. - $Pr(z_i)$ is the base density, typically multivariate normal. - $\det \frac{\partial f[z_i, \phi]}{\partial z_i}$ is the Jacobian determinant of the inverse mapping.

5 Requirements for Flow Models and Network Layers

To make normalizing flows practical and efficient, the flow models and network layers must satisfy certain requirements:

5.1 Flow Models Requirements

- Use a simple prior $p_Z(z)$ that allows for efficient sampling and tractable likelihood evaluation. For example, an isotropic Gaussian is commonly used.
- Ensure invertible transformations with tractable evaluation:
  - Likelihood evaluation requires efficient computation of the mapping $x \mapsto z$.
  - Sampling requires efficient computation of the mapping $z \mapsto x$.
- Compute the determinant of Jacobian matrices efficiently:
  - The determinant of an $n \times n$ matrix requires $O(n^3)$ computation, which can be prohibitively expensive in a learning loop.
  - Key idea: Use transformations that produce Jacobian matrices with special structures. For instance, for triangular matrices, the determinant is simply the product of diagonal entries, reducing the computation to $O(n)$.

5.2 Network Layers Requirements

1. The network layers must collectively be sufficiently *expressive* to map a multivariate standard normal distribution to an arbitrary density.
2. Each layer must be *invertible*, defining a unique one-to-one mapping (bijection) from input to output. Without invertibility, the inverse mapping $z \mapsto x$ would be ambiguous.
3. Efficient computation of the *inverse* is essential, as this operation is performed during every likelihood evaluation.

4. Efficient computation of the determinant of the Jacobian is also necessary, whether for the forward or inverse mapping.

5.3 Triangular Jacobian

The Jacobian matrix J for a function $\mathbf{f}(\mathbf{z}) = (f_1(\mathbf{z}), f_2(\mathbf{z}), \ldots, f_n(\mathbf{z}))$ is given by:

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \cdots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}.$$

If $x_i = f_i(\mathbf{z})$ only depends on $z_{\leq i} = (z_1, \ldots, z_i)$, the Jacobian becomes lower triangular:

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & 0 & \cdots & 0 \\ \frac{\partial f_2}{\partial z_1} & \frac{\partial f_2}{\partial z_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \frac{\partial f_n}{\partial z_2} & \cdots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}.$$

For such triangular Jacobians, the determinant can be computed as:

$$\det(J) = \prod_{i=1}^{n} \frac{\partial f_i}{\partial z_i}.$$

This allows $O(n)$ computation of the determinant.

6 Invertible Network Layers

There are different types of invertible network layers or flows that are used in normalizing flows. These layers must balance expressiveness and computational efficiency. Below are the subsections covered:

- Linear Flows
- Elementwise Flows
- Coupling Flows
- Autoregressive Flows
- Inverse Autoregressive Flows
- Residual Flows
- Residual Flows and Contraction Mappings

6.1 Linear Flows

- Linear flows are transformations of the form:

  $$f[h] = \beta + \Omega h,$$

  where Ω is an invertible matrix, and β is a bias term.

- The Jacobian determinant is:

  $$\det(J(f)) = \det(\Omega),$$

  and both the determinant and inversion cost $O(D^3)$, where D is the dimensionality.

Different matrix structures can improve computational efficiency for inversion and determinant computation. Below are the key types:

- **Diagonal Matrices:**
  - Computational cost: $O(D)$ for inversion and determinant.
  - Limitation: No interaction between the elements of h.
- **Orthogonal Matrices:**
  - Efficient inversion.
  - Determinant is fixed (± 1).
  - Limitation: Cannot scale individual dimensions.
- **Triangular Matrices:**
  - Computational cost: $O(D^2)$ for inversion using back-substitution.
  - Determinant: Product of diagonal elements.
  - Advantage: Allows interactions between dimensions and maintains computational efficiency.

Linear flows cannot transform a normal distribution into arbitrary densities.

6.2 Elementwise Flows

Since linear flows are not sufficient, we use non-linear flows which apply a pointwise nonlinear function $f(\cdot, \phi)$ to each input dimension:

$$f[h] = \left[f(h_1, \phi), f(h_2, \phi), \ldots, f(h_D, \phi)\right]^T.$$

- The Jacobian $\frac{\partial \mathbf{f}[h]}{\partial h}$ is diagonal since the d-th input to $\mathbf{f}[h]$ only affects the d-th output, and its determinant is the product of the entries on the diagonal:

$$\det(J) = \prod_{d=1}^{D} \left| \frac{\partial f(h_d, \phi)}{\partial h_d} \right|.$$

- The function $f[\cdot, \phi]$ could be a fixed invertible nonlinearity like the leaky ReLU, in which case there are no parameters, or it may be any parameterized invertible one-to-one mapping.
- Efficient but does not mix input dimensions, so it is used in combination with other flows.

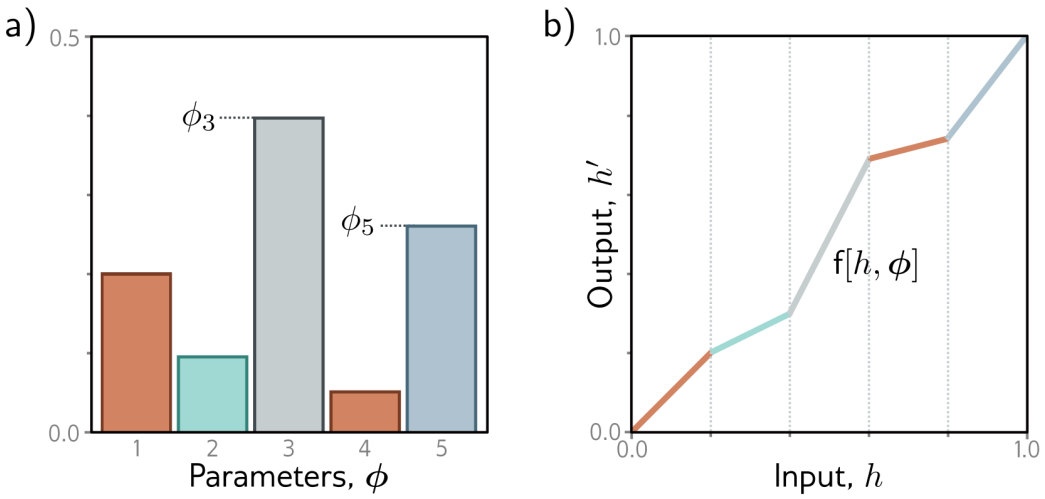
6.2.1 Example:

A simple example is a piecewise linear function with K regions, which maps $[0, 1]$ to $[0, 1]$ as:

$$f[h, \phi] = \left(\sum_{k=1}^{b-1} \phi_k\right) + (hK - b + 1)\phi_b$$

where the parameters $\phi_1, \phi_2, \ldots, \phi_K$ are positive (randomly initialized at first using uniform values) and sum to 1, and $b = \lfloor Kh \rfloor + 1$ is the index of the bin that contains h. The first term is the sum of all the preceding bins, and the second term represents the proportion of the way through the current bin that h lies.

This function is easy to invert, and its gradient can be calculated almost everywhere. There are many similar schemes for creating smooth functions, often using splines with parameters that ensure the function is monotonic and hence invertible.

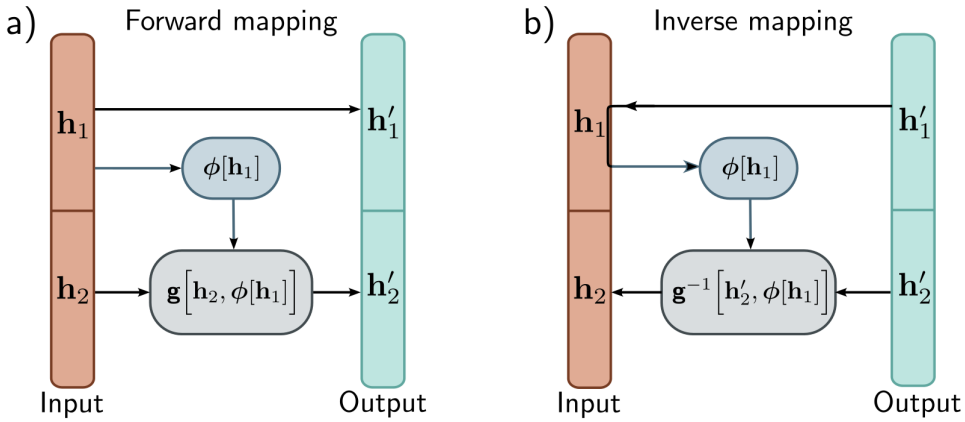


An invertible piecewise linear mapping $h' = f[h, \phi]$ can be created by dividing the input domain $h \in [0, 1]$ into K equally sized regions (here $K = 5$). Each region has a slope with parameter ϕ_k.

- a) If these parameters are positive and sum to one, then
- b) The function will be invertible and map to the output domain $h' \in [0, 1]$.

Elementwise flows are nonlinear but do not mix input dimensions, so they cannot create correlations between variables. When alternated with linear flows (which do mix dimensions), more complex transformations can be modeled. However, in practice, elementwise flows are used as components of more complex layers like coupling flows.

6.3 Coupling Flows



- Input is split into two parts: $h = [h_1^T, h_2^T]^T$.
- The transformation/flow is:

$$h'_1 = h_1, \quad h'_2 = g(h_2, \phi(h_1)),$$

  where $g(\cdot, \phi)$ is an invertible function (here an elementwise flow), and $\phi(h_1)$ is computed from h_1 and is a nonlinear function which is normally defined by a NNs and doesn't need to be invertible.
- original values can be recovered as:

$$h_1 = h'_1, \quad h_2 = g^{-1}(h'_2, \phi(h_1)),$$

- If g is elementwise flow, the Jacobian is triangular with the identity matrix in the top-left quadrant:

$$J = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{\partial g_3}{\partial h_3} & 0 \\ 0 & 0 & 0 & \frac{\partial g_4}{\partial h_4} \end{pmatrix}$$

  for above example:

$$J = \begin{pmatrix} 1 & 0 \\ 0 & \frac{\partial g[h_2, \phi(h_1)]}{\partial h_2} \end{pmatrix}.$$

- Jacobinas determinant is:

$$\det(J) = \prod \text{diagonal terms.}$$

- The inverse and Jacobian can be computed efficiently, but this approach only transforms the second half of the parameters in a way that depends on the first half.
- To create a more general transformation, the elements of h are randomly shuffled using permutation matrices between layers, ensuring that every variable is ultimately transformed by every other.
- In practice, these permutation matrices are difficult to learn. As a result:
  - They are initialized randomly.
  - They are then frozen during training.
- For structured data like images:
  - The channels are divided into two halves h_1 and h_2.
  - The two halves are permuted between layers using 1×1 convolutions.

6.3.1 Permutation Matrices

A permutation matrix has exactly one non-zero entry in each row and column, and all of these entries take the value one. It is a special case of an orthogonal matrix, so its inverse is its own transpose, and its determinant is always one. As the name suggests, it has the effect of permuting the entries of a vector. For example:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} b \\ c \\ a \end{pmatrix}.$$

6.3.2 NICE

The NICE (Nonlinear Independent Components Estimation) technique belongs to the category of Coupling Flows.

Additive Coupling Layer $\mathbf{z} \mapsto \mathbf{x}$: Partition the variables $\mathbf{z}$ into two disjoint subsets, say $\mathbf{z}_{1:d}$ and $\mathbf{z}_{d+1:n}$ for any $1 \leq d < n$.

- $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
- $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\phi(\mathbf{z}_{1:d})$, where $m_\phi(\cdot)$ is a neural network with parameters ϕ, d input units, and $n - d$ output units.

Inverse Mapping $\mathbf{x} \mapsto \mathbf{z}$:

- $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$ (identity transformation)
- $\mathbf{z}_{d+1:n} = \mathbf{x}_{d+1:n} - m_\phi(\mathbf{x}_{1:d})$

Jacobian of Forward Mapping:

$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & I_{n-d} \end{pmatrix}.$$

$$\det(J) = 1$$

Rescaling Transformation (Final Layer) $\mathbf{x} \mapsto \mathbf{z}$:

- Additive coupling layers are composed together (with arbitrary partitions of variables in each layer) - order doesn't matter.
- The final layer of NICE applies a rescaling transformation (simple element-wise):

  $$x_i = s_i z_i,$$

  where $s_i > 0$ is the scaling factor for the i-th dimension.

Inverse Mapping for Scaling Transformation:

$$z_i = \frac{x_i}{s_i}.$$

Jacobian of Forward Mapping (Scaling Transformation):

$$J = \text{diag}(s),$$

where $\text{diag}(s)$ is a diagonal matrix with s_i as diagonal elements.

Determinant of Jacobian:

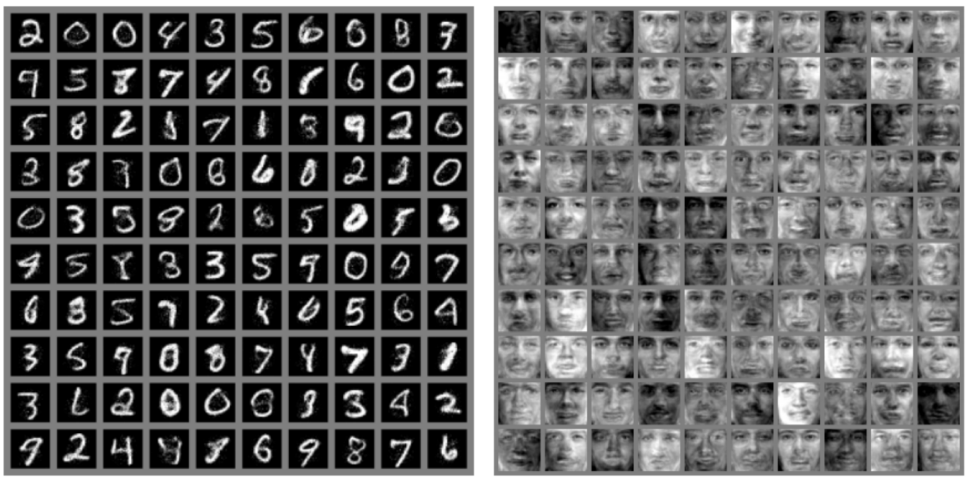
$$\det(J) = \prod_{i=1}^{n} s_i.$$

Which One Applies When?

- **Additive coupling:** Describes the main transformation applied layer-by-layer in NICE.
- **Rescaling layer:** Applied at the final layer of NICE to allow for scaling of the transformed variables.

Volume Preserving Transformation:

- Since the determinant of the Jacobian for the coupling layers is 1, these transformations are volume-preserving.
- The scaling transformation adds the ability to expand or contract volumes, as determined by s_i.



(a) Model trained on MNIST (b) Model trained on TFD

Real-NVP: Non-Volume Preserving Extension of NICE

Forward Mapping $\mathbf{z} \mapsto \mathbf{x}$:

- $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
- $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\phi(\mathbf{z}_{1:d})) + \mu_\phi(\mathbf{z}_{1:d})$
  - $\mu_\phi(\cdot)$ and $\alpha_\phi(\cdot)$ are both neural networks with parameters ϕ.
  - d input units and $n - d$ output units.
  - $\odot$: elementwise product.

Inverse Mapping $\mathbf{x} \mapsto \mathbf{z}$:

- $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$ (identity transformation)
- $\mathbf{z}_{d+1:n} = (\mathbf{x}_{d+1:n} - \mu_\phi(\mathbf{x}_{1:d})) \odot \exp(-\alpha_\phi(\mathbf{x}_{1:d}))$

Jacobian of Forward Mapping:

$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \operatorname{diag}(\exp(\alpha_\phi(\mathbf{z}_{1:d}))) \end{pmatrix}.$$

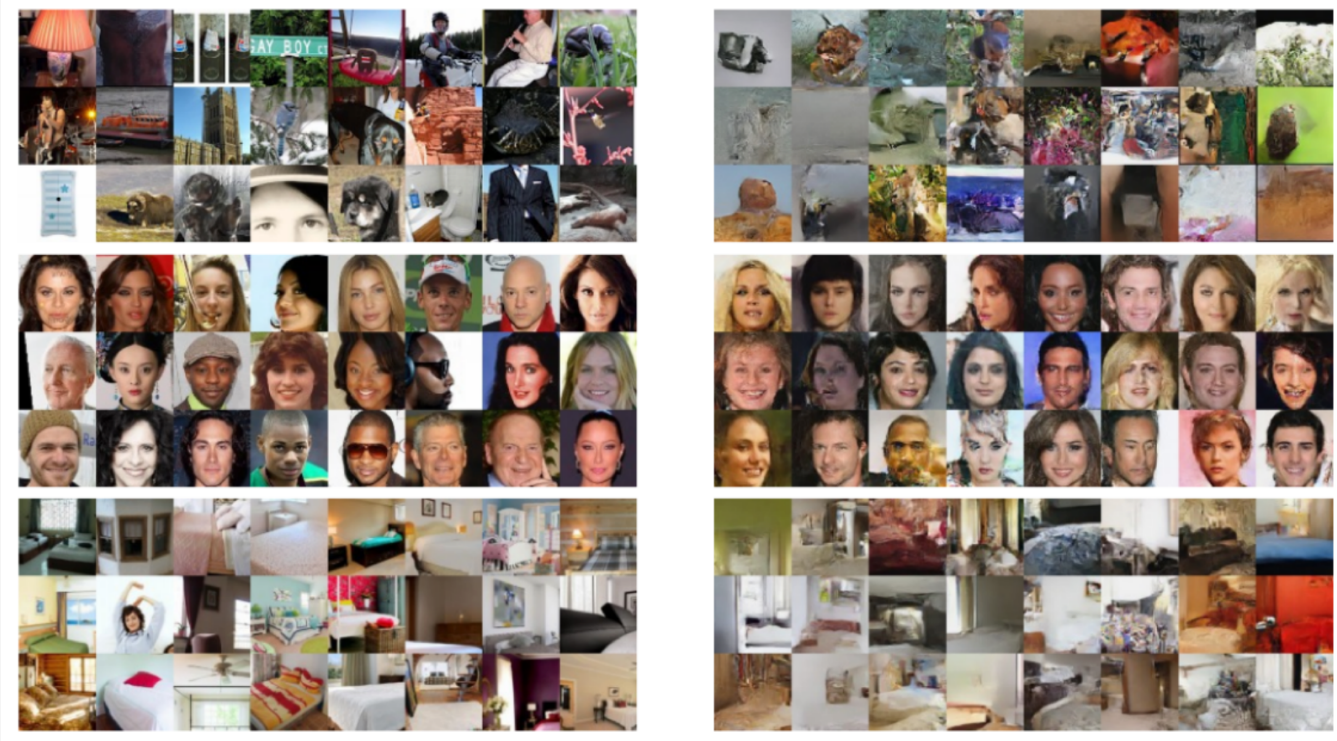
Determinant of Jacobian:

$$\det(J) = \prod_{i=d+1}^{n} \exp(\alpha_\phi(\mathbf{z}_i)) = \exp\left(\sum_{i=d+1}^{n} \alpha_\phi(\mathbf{z}_i)\right).$$

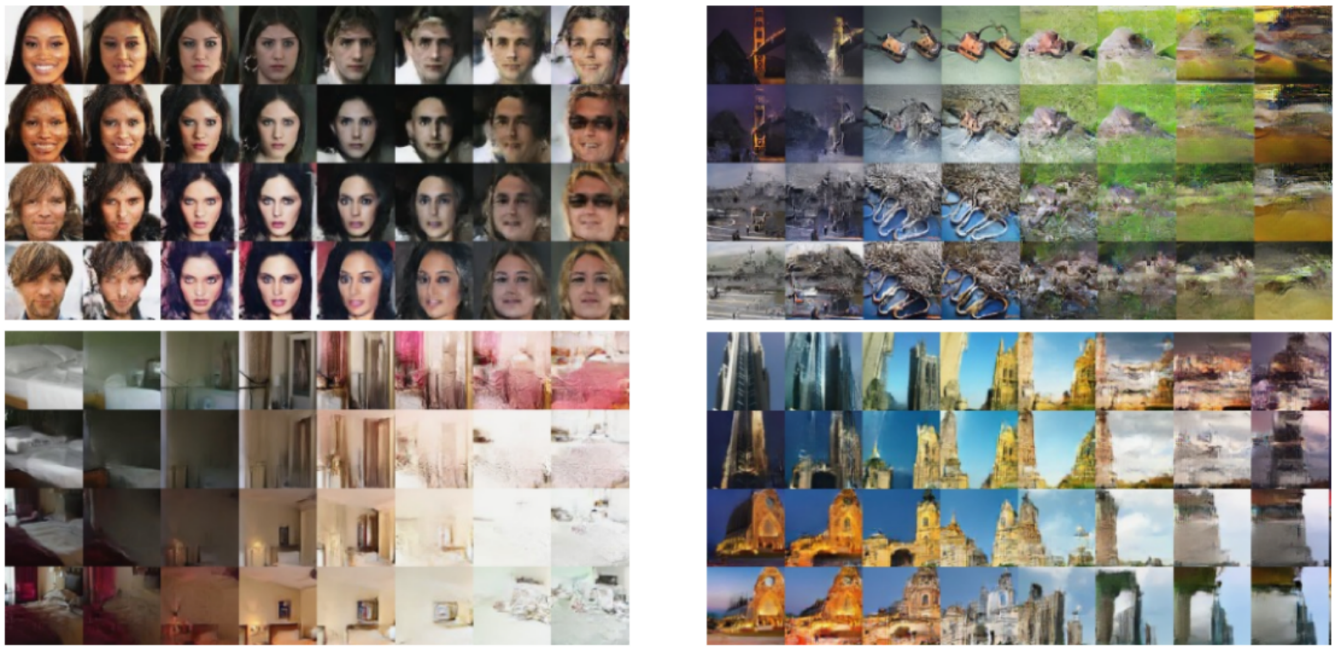
Properties:

- Real-NVP introduces a **non-volume preserving transformation**.
- The determinant can be less than or greater than 1 depending on the scaling factors α_ϕ.

Samples Generated via Real-NVP The following are example samples generated using Real-NVP:



Interpolation in Latent Space with Real-NVP



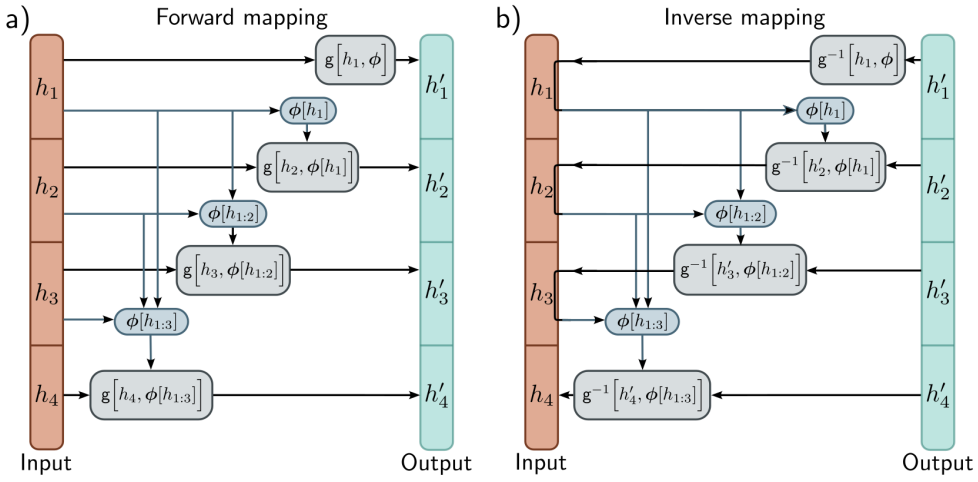
Given four latent vectors $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \mathbf{z}^{(3)}, \mathbf{z}^{(4)}$ obtained from the mappings $\mathbf{x} \mapsto \mathbf{z}$, we can define an interpolated latent vector $\mathbf{z}$ as:

$$\mathbf{z} = \cos(\phi) \left(\mathbf{z}^{(1)} \cos(\phi') + \mathbf{z}^{(2)} \sin(\phi') \right) + \sin(\phi) \left(\mathbf{z}^{(3)} \cos(\phi') + \mathbf{z}^{(4)} \sin(\phi') \right),$$

where:

- ϕ and ϕ' are interpolation parameters.
- This equation combines the four latent vectors using weighted contributions from each.

6.4 Autoregressive Flows



- Generalize coupling flows by transforming each input dimension sequentially:

$$h'_d = g(h_d, \phi(h_{1:d-1})),$$

where $g(\cdot, \phi)$ is invertible, and $\phi(h_{1:d-1})$ depends on previous dimensions.

- Forward pass is efficient and parallelizable with masking, but inversion requires sequential computation.
- Autoregressive flows are universal approximators for probability distributions.

6.5 Continuous Autoregressive Models as Flow Models

- Consider a Gaussian autoregressive model:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|\mathbf{x}_{<i})$$

such that

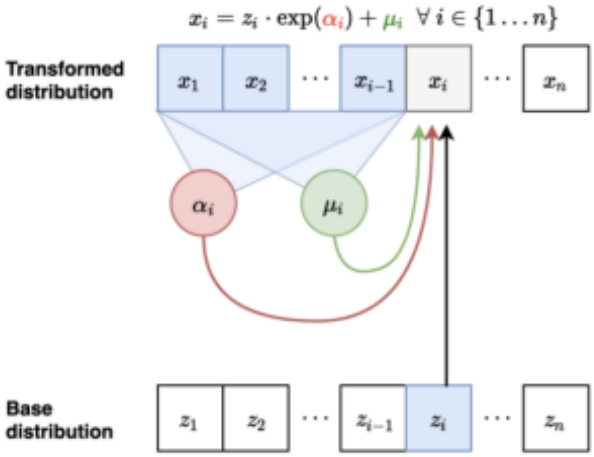
$$p(x_i|\mathbf{x}_{<i}) = \mathcal{N}(\mu_i(x_1, \cdots, x_{i-1}), \exp(\alpha_i(x_1, \cdots, x_{i-1}))^2).$$

Here, $\mu_i(\cdot)$ and $\alpha_i(\cdot)$ are neural networks for $i > 1$ and constants for $i = 1$.

- Sampler for this model:
  1. Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \cdots, n$.
  2. Let $x_1 = \exp(\alpha_1)z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$.
  3. Let $x_2 = \exp(\alpha_2)z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$.
  4. Let $x_3 = \exp(\alpha_3)z_3 + \mu_3$. $\cdots$
- **Flow interpretation**: Transforms samples from the standard Gaussian $(z_1, z_2, \cdots, z_n)$ to those generated from the model $(x_1, x_2, \cdots, x_n)$ via invertible transformations (parameterized by $\mu_i(\cdot), \alpha_i(\cdot)$).

- Forward mapping from $\mathbf{z} \mapsto \mathbf{x}$:
  - Let $x_1 = \exp(\alpha_1)z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$.
  - Let $x_2 = \exp(\alpha_2)z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$.
- Sampling is sequential and slow (like autoregressive): $\mathcal{O}(n)$ time.

$$x_i = z_i \cdot \exp(\alpha_i) + \mu_i \quad \forall i \in \{1, \ldots, n\}$$

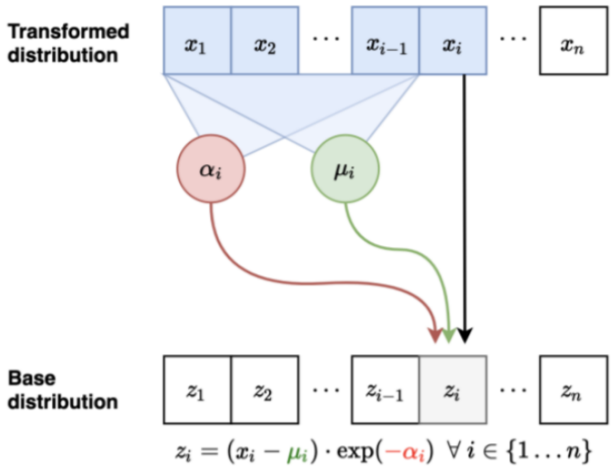


- Inverse mapping from $\mathbf{x} \mapsto \mathbf{z}$:
  - Compute all μ_i, α_i (can be done in parallel using, e.g., MADE).
  - Let $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$ (scale and shift).
  - Let $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$.

– Let $z_3 = (x_3 - \mu_3)/\exp(\alpha_3)$ $\cdots$

- Jacobian is lower diagonal, hence efficient determinant computation.
- Likelihood evaluation is easy and parallelizable (like MADE).
- Layers with different variable orderings can be stacked.

$$z_i = (x_i - \mu_i) \cdot \exp(-\alpha_i) \quad \forall\, i \in \{1, \ldots, n\}$$



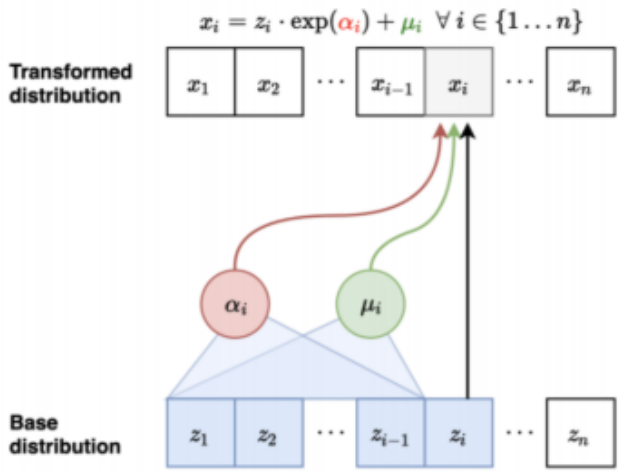
6.6 Inverse Autoregressive Flows

- Defined in the normalizing (inverse) direction for efficient likelihood computation:

$$h'_d = g(h_d, \phi(h_{1:d-1})).$$

- Sampling in the generative direction is sequential and slow.
- Combines masked autoregressive flows (for training) and inverse flows (for sampling).

6.6.1 Inverse Autoregressive Flow (IAF)



- **Forward mapping from $\mathbf{z} \mapsto \mathbf{x}$ (parallel):**

- Sample $z_i \sim \mathcal{N}(0,1)$ for $i = 1, \cdots, n$.
- Compute all μ_i, α_i (can be done in parallel).
- Let $x_1 = \exp(\alpha_1) z_1 + \mu_1$.
- Let $x_2 = \exp(\alpha_2) z_2 + \mu_2 \cdots$

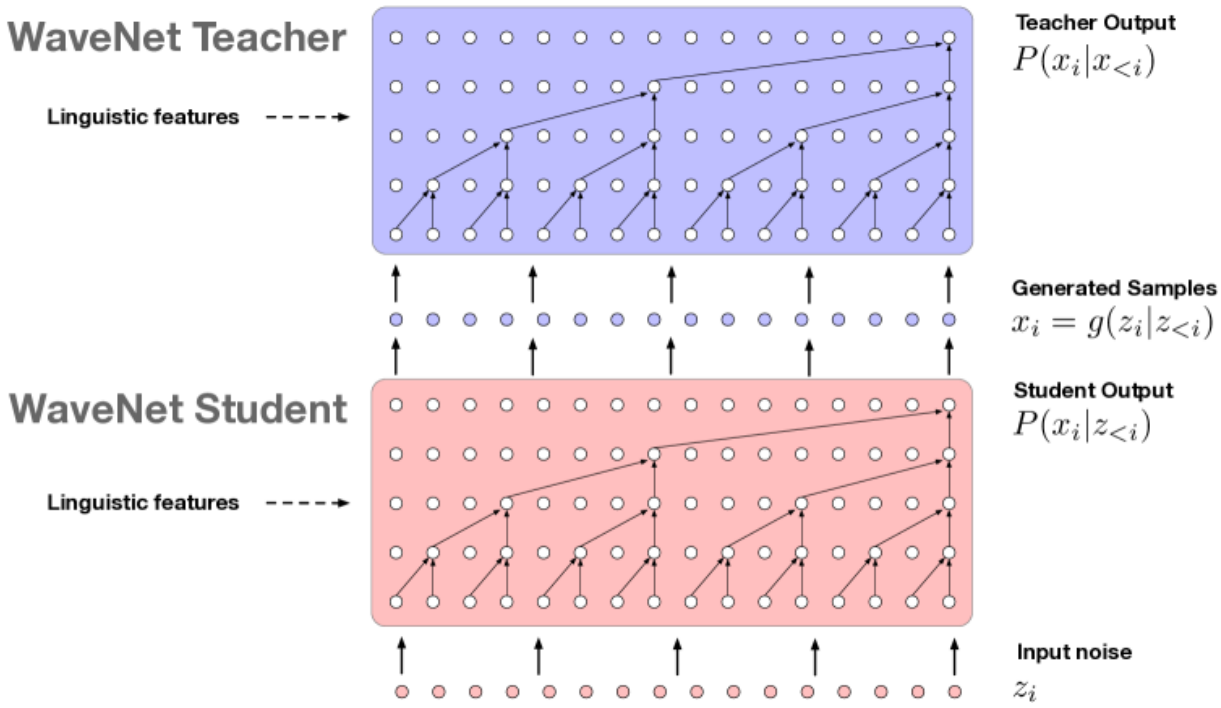
- **Inverse mapping from $\mathbf{x} \mapsto \mathbf{z}$ (sequential):**
  - Let $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$. Compute $\mu_2(z_1), \alpha_2(z_1)$.
  - Let $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$. Compute $\mu_3(z_1, z_2), \alpha_3(z_1, z_2)$.
- Fast to sample from, slow to evaluate likelihoods of data points (train).
- **Note:** Fast to evaluate likelihoods of a generated point (cache $z_1, z_2, \cdots, z_n$).

$$x_i = z_i \cdot \exp(\alpha_i) + \mu_i \quad \forall\, i \in \{1, \ldots, n\}$$

Computational tradeoffs:

- **MAF:** Fast likelihood evaluation, slow sampling.
- **IAF:** Fast sampling, slow likelihood evaluation.
- MAF is more suited for training based on MLE (Maximum Likelihood Estimation) and density estimation.
- IAF is more suited for real-time generation.
- Can we get the best of both worlds? yes, Parallel-Wavenet

6.6.2 Parallel Wavenet



- **Two-part training with a teacher and student model:**
  - Teacher is parameterized by MAF and can be efficiently trained via Maximum Likelihood Estimation (MLE).

  - Once the teacher is trained, a student model is initialized and parameterized by IAF.
  - Student model allows for efficient sampling but cannot efficiently evaluate density for external data points.
- **Key observation:** IAF can efficiently evaluate densities of its own generations by caching the noise variables $(u_1, u_2, \ldots, u_n)$.

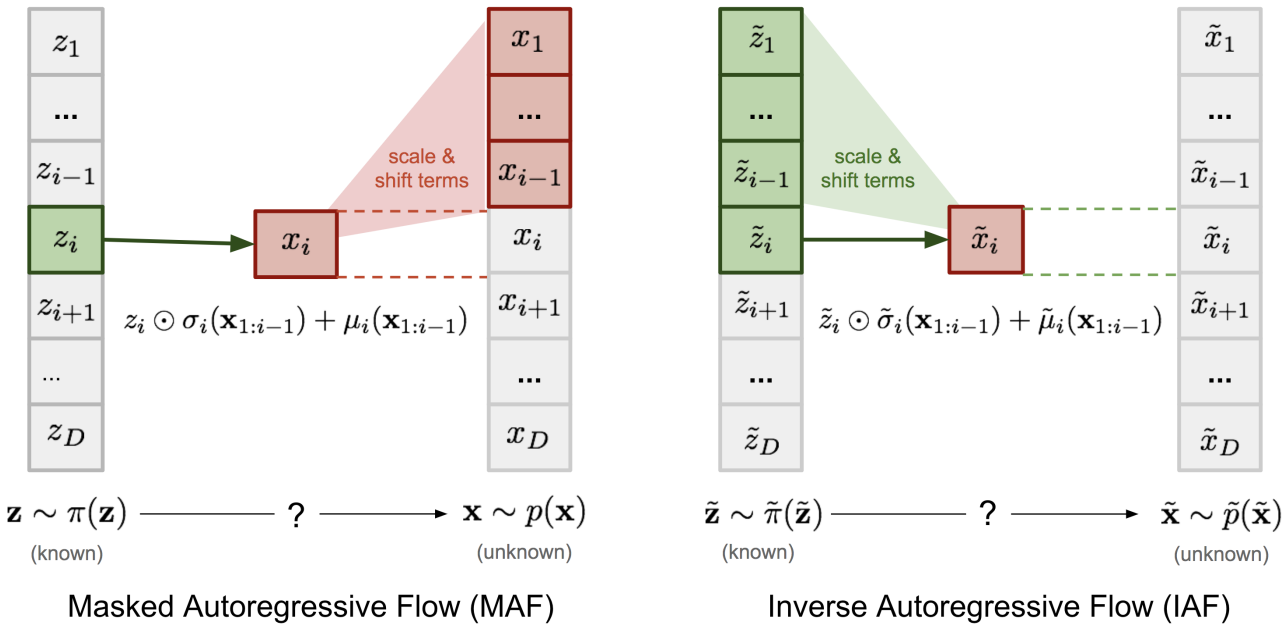
Probability Density Distillation

- The student distribution is trained to minimize the KL divergence between the student (s) and teacher (t):

$$D_{\mathrm{KL}}(s, t) = \mathbb{E}_{\mathbf{x}\sim s}[\log s(\mathbf{x}) - \log t(\mathbf{x})].$$

- Monte Carlo estimates for this objective require:
  - Samples $\mathbf{x}$ from the student model (IAF).
  - Density of $\mathbf{x}$ assigned by the student model.
  - Density of $\mathbf{x}$ assigned by the teacher model (MAF).
- All operations above can be implemented efficiently.

Overall Algorithm



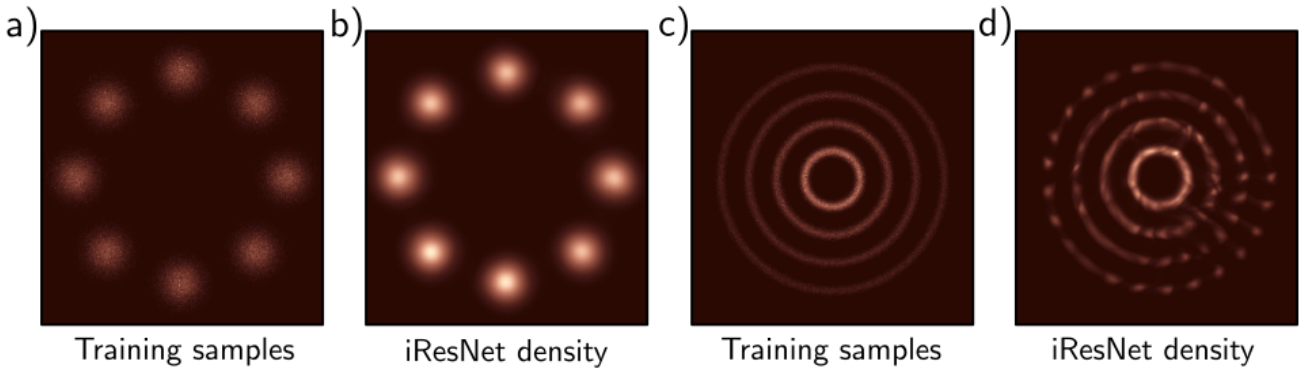
- **Training:**
  - Step 1: Train teacher model (MAF) via MLE.
  - Step 2: Train student model (IAF) to minimize KL divergence with the teacher.
- **Test-time:** Use the student model for testing.
- **Outcome:** Improves sampling efficiency over the original Wavenet (vanilla autoregressive model) by 1000×.

7 Applications of Normalizing Flows

7.1 Modeling Densities

Normalizing flows enable the exact computation of log-likelihoods for new samples, unlike other generative models such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and diffusion models, which only provide approximations or lower bounds on the log-likelihood.

An example of density modeling using i-ResNet shows its ability to model toy 2D distributions, highlighting the effectiveness of normalizing flows in density estimation.

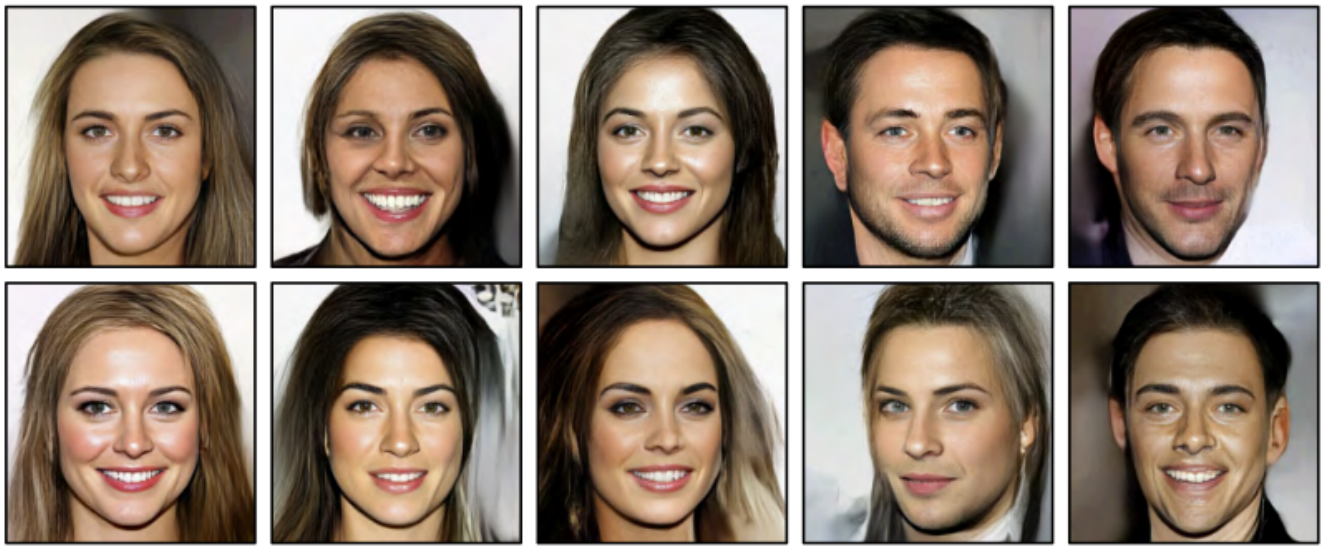


Key applications in density modeling include:

- Density Estimation: Normalizing flows can precisely model probability distributions, making them useful for accurate density estimation.
- Anomaly Detection: By training a normalizing flow model on a dataset, the probability of new samples can be evaluated. Samples with low probabilities are flagged as anomalies. However, care must be taken as high-probability outliers that do not belong to the "typical set" may still exist.

7.2 Synthesis (GLOW Model)

The GLOW model is a specific application of normalizing flows for generating high-fidelity images. This model incorporates several key techniques to ensure image generation is both efficient and high-quality.

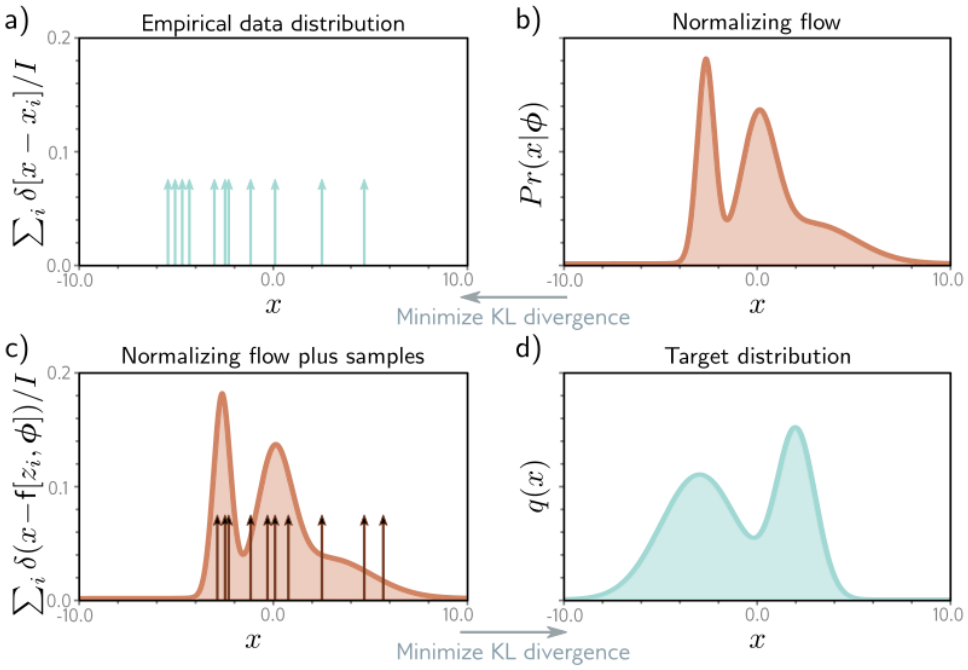


- Input: The input is an image tensor of size $256 \times 256 \times 3$ representing an RGB image.
- Architecture:
  - Coupling Layers: The image channels are split into two halves. One half undergoes an affine transformation, with the parameters computed using a convolutional neural network (CNN) on the other half.
  - 1x1 Convolutions: These are parameterized using LU decomposition, enabling channel-wise mixing.
  - Downsampling: The resolution is periodically reduced by merging each 2×2 patch into a single position with four times as many channels.
  - Multi-scale Flow: Certain channels are periodically removed and stored in a latent vector $\mathbf{z}$, enabling the multi-scale nature of the flow.
- Dequantization: Images are discrete due to the quantization of RGB values. To prevent the log-likelihood from growing unbounded, noise is added to the inputs during
- Sampling: During sampling, the GLOW model samples from the base density raised to a positive power. This ensures that samples are closer to the center of the distribution rather than the tails, a process analogous to the "truncation trick" used in GANs.

While GLOW produces high-quality samples, GANs and diffusion models generally achieve better image fidelity. This is possibly due to the invertibility constraint on normalizing flows or the relatively lower research effort devoted to these models. Image interpolation using GLOW demonstrates the ability to generate smooth transformations between two real images.

7.3 Approximating Other Density Models

Normalizing flows can be used to generate samples that approximate existing densities that are easy to evaluate but difficult to sample from.



- **Teacher-Student Setup:**
  - The target density $q(x)$ is referred to as the *teacher*.
  - The density modeled by the normalizing flow $P(x \mid \varphi)$ is the *student*.
- **Training Approach:**
  - Generate samples $x_i = f(z_i, \varphi)$ from the student model. Since these samples are generated by the model, the latent variables z_i are known.
  - Compute the likelihood of these samples using the student model without needing to invert the transformation.
  - Use a loss function based on the reverse Kullback-Leibler (KL) divergence to train the student model to match the likelihood of the teacher model:

$$\hat{\varphi} = \arg\min_{\varphi} \mathrm{KL}\left(\frac{1}{I}\sum_{i=1}^{I}\delta(x - f(z_i, \varphi)) \middle\| q(x)\right)$$

    Here, $\delta(x)$ is the Dirac delta function, and the goal is to minimize the divergence between the likelihoods of the student and teacher.
- **Comparison with Traditional Normalizing Flows:**
  - Traditional normalizing flows aim to build a density model $P(x \mid \varphi)$ for data samples x_i drawn from an unknown distribution.
  - The standard approach relies on maximum likelihood estimation and minimizes the forward KL divergence:

$$\hat{\varphi} = \arg\min_{\varphi} \mathrm{KL}\left(\frac{1}{I}\sum_{i=1}^{I}\delta(x - x_i) \middle\| P(x \mid \varphi)\right)$$

  - This approach allows normalizing flows to be used in contexts like posterior modeling in Variational Autoencoders (VAEs).

References

[1] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[2] Stefano Ermon. *CS236*.

[3] Rezende and Mohamed 2016. *Normalizing Flows*