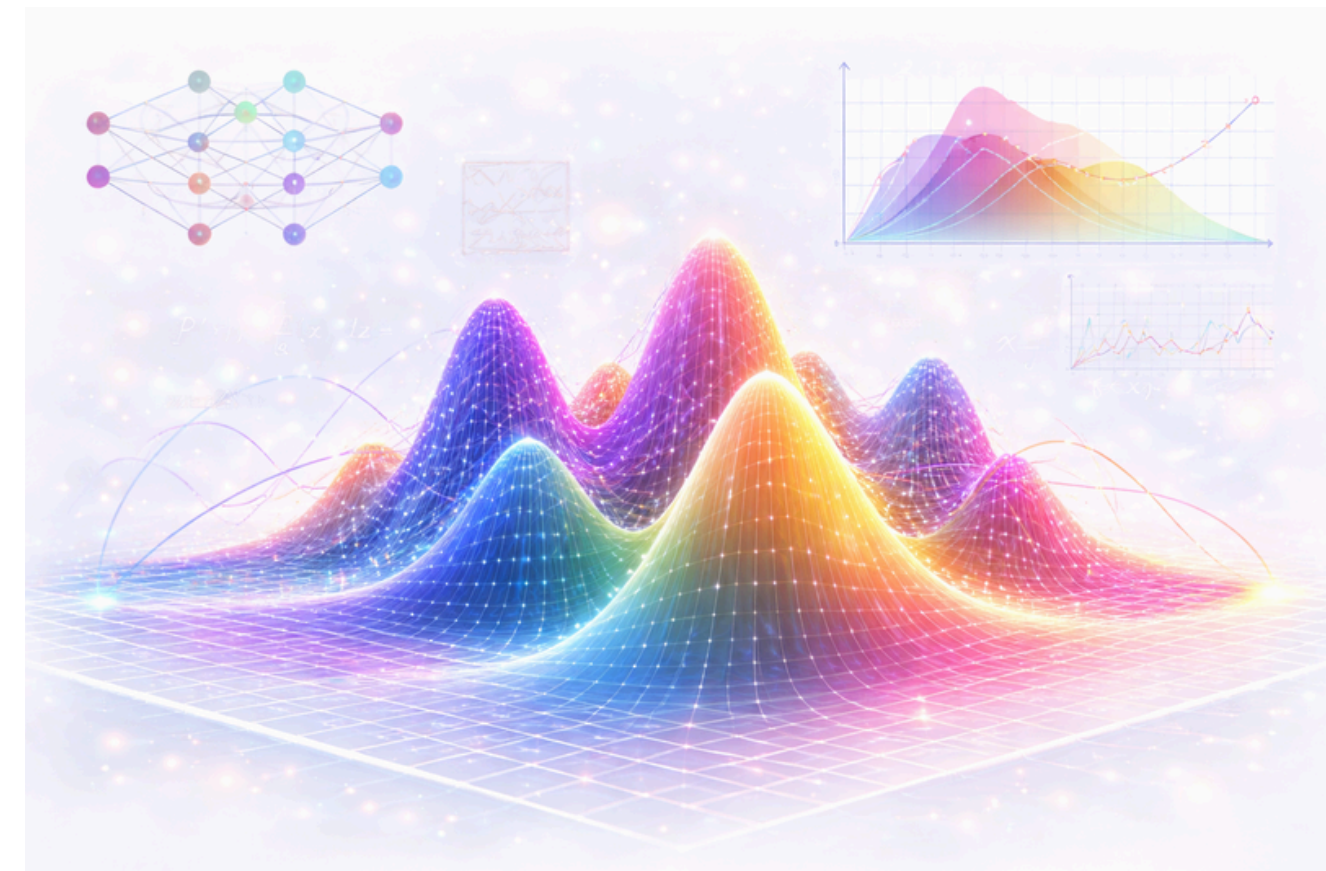


Generative Models

Session 3 - Variational Autoencoders & Normalising Flows



Part 1 – Variational Autoencoders

1. Latent Variable Models: Motivation
2. The ELBO and Variational Inference
3. Reparameterization Trick
4. Amortized Inference

Part 2 – Normalizing Flows

1. Change of Variables Formula
2. Flow-based Models: Principles
3. Coupling Layers: NICE & Real-NVP
4. Autoregressive Flows: MAF & IAF
5. Advanced Architectures: Glow, MintNet

Exercises



Variational Autoencoders

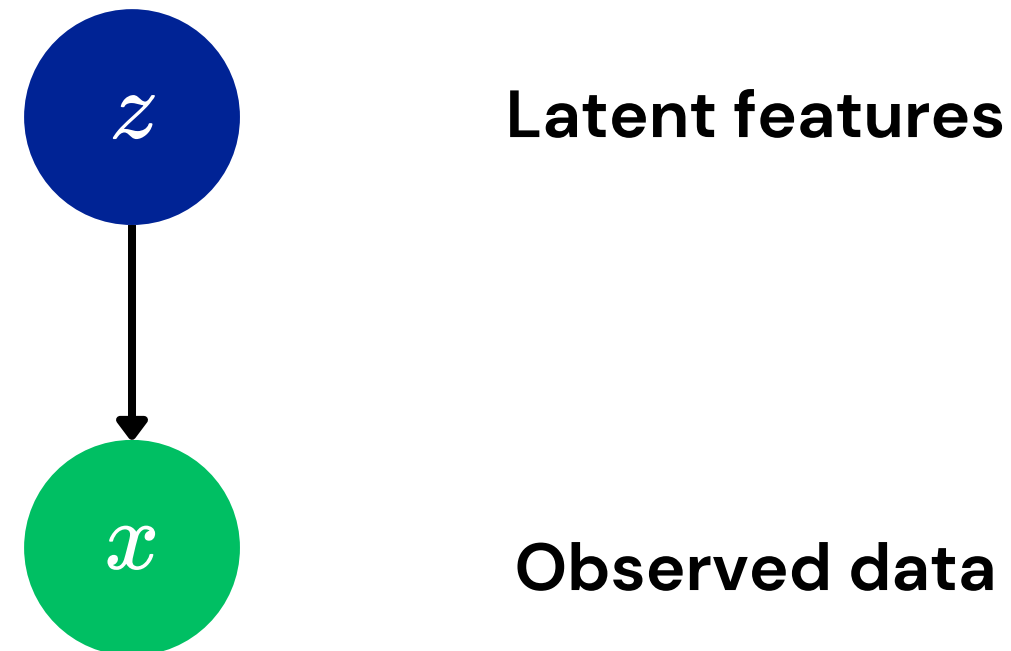
General Principle

Observation

- Images contain many factors of variation (e.g., pose, lighting, background, identity)
- These factors are not explicitly available (latent)

Key idea

- Model these factors using latent variables z
- Observed variables x (e.g., joint values of pixels forming an image)
- With z specified adequately, $p(x|z)$ could be way simpler than $p(x)$



Mixture of Gaussians (Shallow)

$$z \sim \text{Categorical}(1, \dots, K)$$

$$p(x|z = k) = \mathcal{N}(\mu_k, \Sigma_k)$$

Variational Autoencoder (Deep)

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

$$p_{\theta}(x|z = k) = \mathcal{N}(\mu_{\theta}(z), \Sigma_{\theta}(z))$$

where $\mu_{\theta}(z)$ and $\Sigma_{\theta}(z)$ are neural networks

Interpretation

- A mixture of an *infinite* number of Gaussians
- Even though $p_{\theta}(x|z)$ is simple, the marginal $p_{\theta}(x)$ is very complex and flexible

Problem Statement

- Given a dataset $\mathcal{D} = \{x^{(1)}, \dots, x^{(M)}\}$ we want to maximize

$$\log p_{\theta}(\mathcal{D}) = \sum_{x \in \mathcal{D}} \log p_{\theta}(x)$$

Intractability

- The marginal likelihood requires integrating over all latent variables

$$p_{\theta}(x) = \int p_{\theta}(x, z) dz = \int p_{\theta}(x|z)p_{\theta}(z) dz$$

- Discrete case - $z \in \{0, 1\}^{30}$ requires to evaluate $2^{30} \approx 10^9$ terms
- Continuous case - no closed form for the integral
- Computing gradients $\nabla_{\theta} \log p_{\theta}(x)$ is equally hard

Key Insight

- For any distribution $q(z)$ over the latent variables we have

$$\log p_{\theta}(x) = \log \int p_{\theta}(x, z) dz = \log \int \frac{q(z)}{q(z)} p_{\theta}(x, z) dz$$

Jensen's Inequality

- Since log is concave

$$\begin{aligned} \log p_{\theta}(x) &= \log \mathbb{E}_{q(z)} \left[\frac{p_{\theta}(x, z)}{q(z)} \right] \\ &\geq \mathbb{E}_{q(z)} \left[\log \frac{p_{\theta}(x, z)}{q(z)} \right] \\ &= \mathbb{E}_{q(z)} [\log p_{\theta}(x, z)] - \mathbb{E}_{q(z)} [\log q(z)] \\ &= \mathbb{E}_{q(z)} [\log p_{\theta}(x, z)] + H(q) \\ &:= \mathcal{L}(x; \theta, q) \end{aligned}$$

This is called the **Evidence Lower Bound (ELBO)**

Theorem

The gap between the log-likelihood and the ELBO is exactly the KL divergence

$$\log p_{\theta}(x) = \mathcal{L}(x; \theta, q) + D_{KL}(q(z) || p_{\theta}(z|x))$$

Proof (Sketch)

$$\begin{aligned} D_{KL}(q(z) || p_{\theta}(z|x)) &= \mathbb{E}_{q(z)} \left[\log \frac{q(z)}{p_{\theta}(z|x)} \right] \\ &= \mathbb{E}_{q(z)} [\log q(z) - \log p_{\theta}(z, x) + \log p_{\theta}(x)] \\ &= -\mathcal{L}(x; \theta, q) + \log p_{\theta}(x) \end{aligned}$$

Consequence

- The ELBO is tight when $q(z) = p_{\theta}(z|x)$ (posterior)
- Minimising the KL divergence is equivalent to maximizing the ELBO
 - $\log p_{\theta}(x)$ does not depend on the variational distribution $q(z)$

Problem

The true posterior $p_{\theta}(z|x)$ is intractable to compute

Variational Inference

Choose a tractable family of distributions $q_{\phi}(z)$ parametrized by ϕ

Example - Gaussian Distribution

$$q_{\phi}(z) = \mathcal{N}(\mu_{\phi}, \Sigma_{\phi})$$

Optimization Objective

Jointly optimize the ELBO over both θ and ϕ

$$\max_{\theta, \phi} \sum_{x \in \mathcal{D}} \mathcal{L}(x; \theta, \phi)$$

where

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{q(z)} [\log p_{\theta}(x, z) - \log q_{\phi}(z)]$$

Goal

Compute $\nabla_{\phi} \mathcal{L}(x; \theta, \phi)$ where

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{q(z)} [\log p_{\theta}(x, z) - \log q_{\phi}(z)]$$

Problem

The expectation itself depends on ϕ so we cannot simply move the gradient instead the expectation

Naive Monte Carlo (does not work)

$$\mathcal{L}(x; \theta, \phi) \approx \frac{1}{K} \sum_{k=1}^K [\log p_{\theta}(x, z^{(k)}) - \log q_{\phi}(z^{(k)})]$$

$$\mathcal{L}(x; \theta, \phi) \not\approx \frac{1}{K} \sum_{k=1}^K \nabla_{\phi} [\log p_{\theta}(x, z^{(k)}) - \log q_{\phi}(z^{(k)})]$$

where $z^{(k)} \sim q_{\phi}(z)$

Key idea

Express $z \sim q_\phi(z)$ as a deterministic function of ϕ and a random variable ϵ that does not depend on ϕ

For Gaussian $q_\phi(z) = \mathcal{N}(\mu, \sigma^2)$

Two equivalent sampling procedures

- Sample $z \sim q_\phi(z)$
- Sample $\epsilon \sim \mathcal{N}(0, I)$ and set $z = \mu + \sigma \odot \epsilon = g(\epsilon; \phi)$

Gradient computation

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi(z)}[r(z)] &= \nabla_\phi \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)}[r(g(\epsilon; \phi))] \\ &= \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)}[\nabla_\phi r(g(\epsilon; \phi))] \\ &\approx \frac{1}{K} \sum_{k=1}^K [\nabla_\phi r(g(\epsilon^{(k)}; \phi))]\end{aligned}$$

Now we can use backpropagation!

Current Approach

For each data point $x^{(i)}$, we have separate variational parameters ϕ_i

$$\max_{\theta, \phi_1, \dots, \phi_M} \sum_{i=1}^M \mathcal{L}(x^{(i)}; \theta, \phi_i)$$

Problems

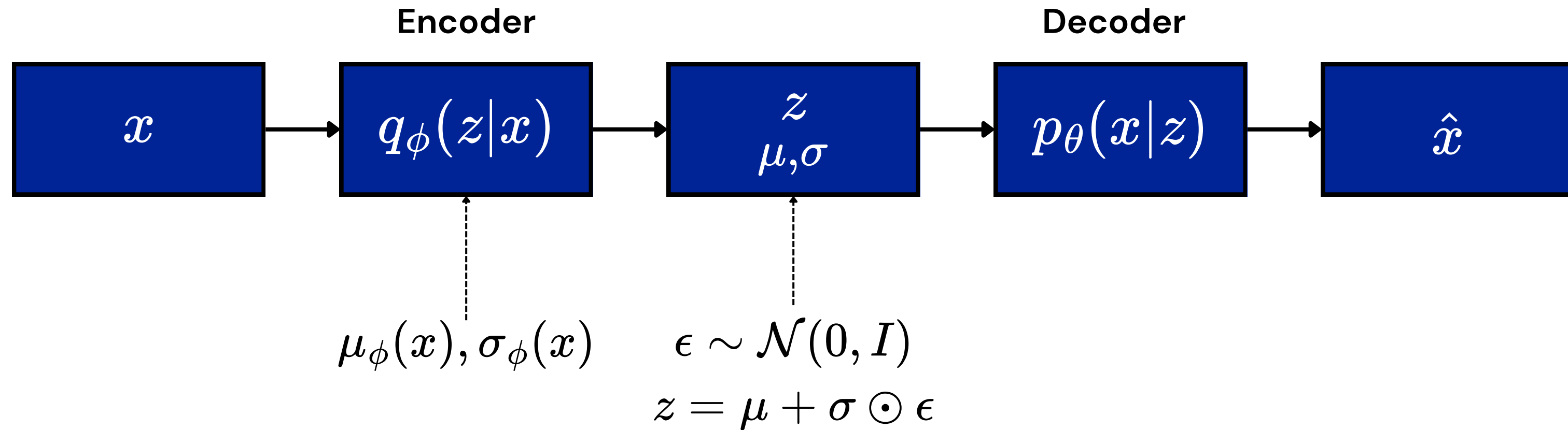
- Need to store M sets of parameters (one per data point)
- For each new data point, need to run optimization from scratch
- Does not scale to large datasets

Amortized Inference

Learn a single function $f_\lambda : \mathcal{X} \rightarrow \phi$ that maps data points to variational parameters

$$\phi_i = f_\lambda(x^{(i)})$$

In VAE notation: $q_\phi(z|x)$ where $\phi = \lambda$ are the parameters of the encoder network

**Training Objective (ELBO)**

$$\begin{aligned}\mathcal{L}(x; \theta, \phi) &= \mathbb{E}_{q(z)}[\log p_\theta(x, z)] - D_{KL}(q(z|x) || p(z)) \\ &= \text{Reconstruction term} - \text{Regularization term}\end{aligned}$$

Generative Model (Decoder)

$$p(z) = \mathcal{N}(0, I)$$
$$p(x|z) = \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z))$$

Inference Model (Encoder)

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$$

Training

- Maximize ELBO using stochastic gradient descent
- Reparameterization trick enables backpropagation through stochastic layers

Generation

$$z \sim \mathcal{N}(0, I) \quad x \sim p_\theta(x|z)$$

Inference

$$z \sim q_\phi(z|x)$$

Goal

Prove that the ELBO can be written in the following form

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x) || p(z))$$

Start form

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z|x)]$$

Hint

Use the factorization $p_\theta(x, z) = p_\theta(x|z)p_\theta(z)$ and the definition of KL divergence

Time

5-10 minutes

Starting from the ELBO

$$\begin{aligned}\mathcal{L}(x; \theta, \phi) &= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z) - \log q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z) + \log p(z) - \log q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \mathbb{E}_{q_\phi(z|x)} [\log p(z) - \log q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{q_\phi(z|x)}{p(z)} \right] \\ &= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x) || p(z))\end{aligned}$$

This decomposition shows

- **Reconstruction term** - encourages $x \approx \hat{x}$
- **Regularization term** - encourages latent code to match the prior



Normalizing Flows

Tractable Density Estimation

Model	Pros	Cons
Autoregressive	<ul style="list-style-type: none">• Tractable Likelihood• Easy to train	<ul style="list-style-type: none">• Sequential Generation• No direct feature learning
VAE	<ul style="list-style-type: none">• Learn latent features• Parallel Generation	<ul style="list-style-type: none">• Intractable Likelihood• Approximate Inference

Key question

Can we design a latent variable model with **tractable** likelihoods?

Answer: Yes! Using *Normalizing Flows*

Desirable Properties

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

Simple Distributions

Gaussian, uniform distributions satisfy the above properties

But...

Real data distributions are complex (multi-modal, have structure)

Key idea

Map simple distributions (easy to sample and evaluate) to complex distributions through an invertible transformation

$$z \sim p_Z(z) \xrightarrow{f_\theta} x = f_\theta(z)$$

For a continuous random variable

- Cumulative density function (CDF) $F_X(a) = P(X \leq a)$
- Probability density function (pdf) $p_X(a) = F'_X(a) = \frac{dF_X(a)}{da}$

Common Parametric Densities

- Gaussian $X \sim \mathcal{N}(\mu, \sigma^2)$

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

- Uniform $X \sim \mathcal{U}(a, b)$

$$p_X(x) = \frac{1}{b - a} \mathbb{I}_{[a \leq x \leq b]}$$

For Random Vectors

Joint pdf $p_X(x)$ for $X \in \mathbb{R}^n$

Setup

Let $Z \sim \mathcal{U}(0, 2)$ with density $p_Z(z) = \frac{1}{2}$

Question

Let $X = 4Z$. What is $p_X(4)$?

Wrong Answer

$$p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1) = p_Z(1) = \frac{1}{2}$$

Correct Answer

Clearly, $X \sim \mathcal{U}(0, 8)$, so $p_X(4) = \frac{1}{8}$

Issue

We cannot simply substitute! We need the **change of variables** formula

Theorem

- If $X = f(Z)$ where $f(\cdot)$ is monotone with inverse $Z = f^{-1}(X) = h(X)$, then:

$$p_X(x) = p_Z(h(x)) |h'(x)|$$

Verification on the previous example

$$X = f(Z) = 4Z \text{ where } Z \sim \mathcal{U}[0, 2]$$

$$\text{Inverse: } h(X) = X/4$$

$$\begin{aligned} p_X(4) &= p_Z(h(4)) |h'(4)| \\ &= p_Z(1) \cdot \left| \frac{1}{4} \right| \\ &= \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8} \quad \checkmark \end{aligned}$$

Proof for Monotonically Increasing f

Start with the CDF

$$\begin{aligned}F_X(x) &= P(X \leq x) \\&= P(f(Z) \leq x) \\&= P(Z \leq f^{-1}(x)) \quad (\text{since } f \text{ is increasing}) \\&= P(Z \leq h(x)) \\&= F_Z(h(x))\end{aligned}$$

Taking derivatives with respect to x

$$\begin{aligned}p_X(x) &= \frac{dF_X(x)}{dx} \\&= \frac{dF_Z(h(x))}{dx} \\&= p_Z(h(x)) \cdot h'(x) \quad (\text{chain rule})\end{aligned}$$

What if it is monotonically decreasing?

Another Expression

Recall that for $h(x) = f^{-1}(x)$

$$h'(x) = [f^{-1}]'(x) = \frac{1}{f'(f^{-1}(x))} = \frac{1}{f'(z)}$$

where $z = h(x) = f^{-1}(x)$

Therefore, we can also write

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

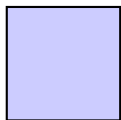
Two equivalent forms

- Inverse direction $p_X(x) = p_Z(f^{-1}(x)) |[f^{-1}]'(x)|$
- Forward direction $p_X(x) = \frac{p_Z(z)}{|f'(z)|}$ where $z = f^{-1}(x)$

Linear Transformations

Let $Z \sim \mathcal{U}[0, 1]^n$ be uniform in the unit hypercube

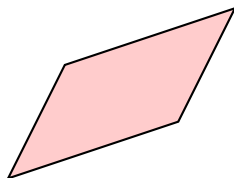
Let $X = AZ$ for an invertible matrix A



Unit square

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

→



Parallelogram

Key Fact

Volume of parallelogram = $|\det(A)|$

Change of Variables: Linear Case

For Linear Transformations

Let $X = AZ$ with $W = A^{-1}$

X is uniformly distributed over a parallelotope of volume $|\det(A)|$

Density

$$p_X(x) = \frac{p_Z(Wx)}{|\det(A)|} = p_Z(Wx) \cdot |\det(W)|$$

since $\det(W) = \det(A^{-1}) = \frac{1}{\det(A)}$

Analogy with 1D Case

- 1D: $p_X(x) = p_Z(z)/|f'(z)|$
- nD (linear): $p_X(x) = p_Z(z)/|\det(A)|$

Change of Variables: General Case

Theorem (General Case)

Let $X = f(Z)$ where $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is invertible

Then:

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

Equivalently:

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

where $z = f^{-1}(x)$ and $J_f(z) = \frac{\partial f(z)}{\partial z}$ is the **Jacobian matrix**

Key Insight

- For linear f : Jacobian = constant matrix A
- For nonlinear f : Jacobian depends on z (linearization)

The Jacobian Matrix

Definition

For $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, $f(z) = (f_1(z), \dots, f_n(z))$:

$$J_f(z) = \frac{\partial f(z)}{\partial z} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \dots & \frac{\partial f_1}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \dots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$$

Important Properties

- 1 $\det(J_{f^{-1}}(x)) = \frac{1}{\det(J_f(z))}$ where $z = f^{-1}(x)$
- 2 For a general $n \times n$ matrix: $\det(J)$ costs $O(n^3)$ to compute
- 3 **Key design principle:** Choose f so that $\det(J_f)$ is cheap!

Computational Challenge

Computing $\det(J_f(z))$ for arbitrary f is prohibitively expensive in a learning loop!

Normalizing Flow Models

Definition

A directed latent variable model where:

- Observed variables: X
- Latent variables: Z
- Mapping $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is **deterministic and invertible**

Model

$$X = f_\theta(Z), \quad Z = f_\theta^{-1}(X)$$

Marginal Likelihood

Using change of variables:

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

Key Difference from VAE

- VAE: x, z can have different dimensions, stochastic decoder
- Flow: $x, z \in \mathbb{R}^n$, deterministic and invertible

A Flow of Transformations

Composing Transformations

Apply a sequence of M invertible transformations:

$$z_m = f_\theta^m \circ \dots \circ f_\theta^1(z_0) = f_\theta^m(f_\theta^{m-1}(\dots(f_\theta^1(z_0))))$$

Let $f_\theta = f_\theta^M \circ \dots \circ f_\theta^1$ be the composition

Why "Normalizing Flow"?

- **Normalizing**: Change of variables gives a normalized density
- **Flow**: Invertible transformations flow from z_0 to $x = z_M$

Marginal Likelihood

By change of variables:

$$p_X(x; \theta) = p_Z(f_\theta^{-1}(x)) \prod_{m=1}^M \left| \det \left(\frac{\partial (f_\theta^m)^{-1}(z_m)}{\partial z_m} \right) \right|$$

since $\det(AB) = \det(A)\det(B)$

Training via Maximum Likelihood

Over dataset \mathcal{D} :

$$\max_{\theta} \log p_{\mathbf{x}}(\mathcal{D}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \left[\log p_{\mathbf{z}}(f_{\theta}^{-1}(\mathbf{x})) + \log \left| \det \left(\frac{\partial f_{\theta}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \right]$$

Exact Likelihood Evaluation

Via inverse transformation $\mathbf{x} \mapsto \mathbf{z}$ and change of variables

Sampling

Via forward transformation $\mathbf{z} \mapsto \mathbf{x}$:

$$\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z}), \quad \mathbf{x} = f_{\theta}(\mathbf{z})$$

Inference

Latent representations via inverse transformation (no inference network!):

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

Property	Pros	Cons
Likelihood	Intractable (need ELBO)	Exact (change of variables)
Dimensionality	\mathcal{X}, \mathcal{Z} can differ	$\mathcal{X}, \mathcal{Z} \in \mathbb{R}^n$
Mapping	Stochastic	Deterministic
Inference	Learned encoder $q_\phi(z x)$	Direct $z = f^{-1}(x)$
Generation	Sample \mathcal{Z} , decode stochastically	Sample $z, x = f(z)$
Training	Maximize ELBO	Maximize exact likelihood

Key Advantage of Flows

- Exact likelihood evaluation
- No approximation gap

Key Challenge

- Designing invertible with tractable Jacobian determinant

Requirements

1 Simple prior $p_Z(z)$

- Easy to sample from (e.g., $\mathcal{N}(0, I)$)
- Easy to evaluate density

2 Invertible transformations

- Forward pass $z \rightarrow x$ for sampling
- Inverse pass $x \rightarrow z$ for likelihood evaluation

3 Tractable Jacobian determinant

- General $n \times n$ matrix: $\det(J) = O(n^3)$ (too expensive!)
- Need special structure for $O(n)$ computation

Key Idea

Choose transformations so that the Jacobian has **special structure**

Triangular Jacobian

Lower Triangular Jacobian

If $x_i = f_i(z)$ depends only on $z_{\leq i} = (z_1, \dots, z_i)$:

$$J_f = \frac{\partial f}{\partial z} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & 0 & \dots & 0 \\ \frac{\partial f_2}{\partial z_1} & \frac{\partial f_2}{\partial z_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \frac{\partial f_n}{\partial z_2} & \dots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$$

Determinant

For a triangular matrix:

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

Complexity: $O(n)$ instead of $O(n^3)$!

Similarly, upper triangular if x_i depends only on $z_{\geq i}$

NICE: Additive Coupling Layers

NICE = Nonlinear Independent Components Estimation (Dinh et al., 2014)

Idea

Partition variables $z = (z_{1:d}, z_{d+1:n})$ into two disjoint subsets

Forward Transformation $z \rightarrow x$:

$$\begin{aligned}x_{1:d} &= z_{1:d} \quad (\text{identity}) \\x_{d+1:n} &= z_{d+1:n} + m_{\theta}(z_{1:d})\end{aligned}$$

where $m_{\theta} : \mathbb{R}^d \rightarrow \mathbb{R}^{n-d}$ is a neural network

Inverse Transformation $x \rightarrow z$:

$$\begin{aligned}z_{1:d} &= x_{1:d} \\z_{d+1:n} &= x_{d+1:n} - m_{\theta}(x_{1:d})\end{aligned}$$

Key: Inverse is trivial to compute!

Jacobian Matrix

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

Determinant

Lower triangular with ones on diagonal:

$$\det(J) = 1$$

Volume Preserving

NICE transformations preserve volume ($|\det(J)| = 1$)

To change volume, NICE adds a final **rescaling layer**:

$$x_i = s_i z_i$$

where $s_i > 0$ are learnable scaling factors

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

Real-NVP = Real-valued Non-Volume Preserving (Dinh et al., 2017)

Extension of NICE

Add scaling in addition to translation:

Forward Transformation $z \rightarrow x$:

$$\begin{aligned}x_{1:d} &= z_{1:d} \\x_{d+1:n} &= z_{d+1:n} \odot \exp(\alpha_\theta(z_{1:d})) + \mu_\theta(z_{1:d})\end{aligned}$$

where $\mu_\theta, \alpha_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^{n-d}$ are neural networks

Inverse Transformation $x \rightarrow z$:

$$\begin{aligned}z_{1:d} &= x_{1:d} \\z_{d+1:n} &= (x_{d+1:n} - \mu_\theta(x_{1:d})) \odot \exp(-\alpha_\theta(x_{1:d}))\end{aligned}$$

Jacobian Matrix

$$J = \frac{\partial x}{\partial z} = \begin{pmatrix} I_d & 0 \\ \frac{\partial x_{d+1:n}}{\partial z_{1:d}} & \text{diag}(\exp(\alpha_\theta(z_{1:d}))) \end{pmatrix}$$

Determinant

Lower triangular:

$$\det(J) = \prod_{i=d+1}^n \exp(\alpha_\theta(z_{1:d})_i) = \exp\left(\sum_{i=d+1}^n \alpha_\theta(z_{1:d})_i\right)$$

Log-Determinant

$$\log |\det(J)| = \sum_{i=d+1}^n \alpha_\theta(z_{1:d})_i$$

Complexity: $O(n)$

Coupling Layers: Key Properties

Advantages

- 1 **Efficient inverse:** Trivial to compute $z = f^{-1}(x)$
- 2 **Tractable Jacobian:** $O(n)$ determinant computation
- 3 **Flexible:** $m_\theta, \mu_\theta, \alpha_\theta$ can be arbitrary neural networks
- 4 **Expressive:** Stack multiple coupling layers with different partitions

Partitioning Strategies

- Alternate between first d and last $n - d$ dimensions
- For images: checkerboard pattern or channel-wise split

Multi-Scale Architecture

Real-NVP uses a multi-scale architecture:

- Progressively squeeze spatial dimensions
- Split off half the channels to latent z
- Continue with remaining channels

Gaussian Autoregressive Model

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

where $p(x_i | x_{<i}) = \mathcal{N}(\mu_i(x_{1:i-1}), \exp(\alpha_i(x_{1:i-1}))^2)$

Sampling Procedure

Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \dots, n$

$$x_1 = \exp(\alpha_1)z_1 + \mu_1$$

$$x_2 = \exp(\alpha_2(x_1))z_2 + \mu_2(x_1)$$

$$x_3 = \exp(\alpha_3(x_1, x_2))z_3 + \mu_3(x_1, x_2)$$

⋮

Flow Interpretation

Transform samples from $z \sim \mathcal{N}(0, 1)$ to x via invertible transformations!

Masked Autoregressive Flow (MAF)

MAF (Papamakarios et al., 2017)

Forward $z \rightarrow x$ (Sequential):

$$x_1 = \exp(\alpha_1)z_1 + \mu_1$$

$$x_2 = \exp(\alpha_2(x_1))z_2 + \mu_2(x_1) \quad (\text{compute } \mu_2, \alpha_2)$$

$$x_3 = \exp(\alpha_3(x_1, x_2))z_3 + \mu_3(x_1, x_2) \quad (\text{compute } \mu_3, \alpha_3)$$

\vdots

Time: $O(n)$ (sequential)

Inverse $x \rightarrow z$ (Parallel):

Compute all μ_i, α_i in parallel using MADE

$$z_1 = (x_1 - \mu_1) / \exp(\alpha_1)$$

$$z_2 = (x_2 - \mu_2) / \exp(\alpha_2)$$

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

\vdots

Time: $O(1)$ (parallel on GPU)

Inverse Autoregressive Flow (IAF)

IAF (Kingma et al., 2016)

Forward $z \rightarrow x$ (Parallel):

Sample $z_i \sim \mathcal{N}(0, 1)$ for all i

Compute all $\mu_i(z_{<i}), \alpha_i(z_{<i})$ in parallel

$$x_1 = \exp(\alpha_1)z_1 + \mu_1$$

$$x_2 = \exp(\alpha_2(z_1))z_2 + \mu_2(z_1)$$

$$x_3 = \exp(\alpha_3(z_1, z_2))z_3 + \mu_3(z_1, z_2)$$

\vdots

Time: $O(1)$ (parallel)

Inverse $x \rightarrow z$ (Sequential):

$$z_1 = (x_1 - \mu_1) / \exp(\alpha_1) \quad (\text{compute } \mu_2, \alpha_2)$$

$$z_2 = (x_2 - \mu_2(z_1)) / \exp(\alpha_2(z_1)) \quad (\text{compute } \mu_3, \alpha_3)$$

\vdots

MAF vs IAF: Computational Tradeoffs

Operation	MAF	IAF
Sampling ($z \rightarrow x$)	Sequential $O(n)$	Parallel $O(1)$
Likelihood ($x \rightarrow z$)	Parallel $O(1)$	Sequential $O(n)$
Training	Fast (MLE)	Slow
Generation	Slow	Fast
Best for	Density estimation	Real-time generation

Relationship

IAF is the *inverse* of MAF!

- MAF inverse = IAF forward
- MAF forward = IAF inverse

Question: Can we get the best of both worlds?

Parallel WaveNet: Best of Both Worlds

Idea (van den Oord et al., 2018)

Use both MAF and IAF together!

Two-Stage Training

1 Teacher Model (MAF)

- Train via MLE (fast likelihood evaluation)
- Learns the target distribution

2 Student Model (IAF)

- Cannot evaluate likelihood of data efficiently
- *But* can evaluate likelihood of its own samples (cache z)
- Train to match teacher via **probability density distillation**

Distillation Objective

Minimize KL divergence between student s and teacher t :

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

All terms can be efficiently computed!

Parallel WaveNet: Algorithm

Training

- 1: **Phase 1:** Train teacher model (MAF) via maximum likelihood

$$\max_{\theta_t} \sum_{x \in \mathcal{D}} \log p_{\theta_t}(x)$$

- 2: **Phase 2:** Train student model (IAF) via distillation
- 3: **for** iteration = 1, ..., N **do**
- 4: Sample $x \sim p_{\theta_s}$ (fast, parallel sampling)
- 5: Compute $\log p_{\theta_s}(x)$ (use cached z)
- 6: Compute $\log p_{\theta_t}(x)$ (fast likelihood)
- 7: Update θ_s to minimize $D_{KL}(p_{\theta_s} \| p_{\theta_t})$
- 8: **end for**

Test Time

Use student model for fast parallel generation (1000× speedup over WaveNet!)

Key Contributions

- 1 **Actnorm**: Activation normalization
 - Data-dependent initialization
 - Stabilizes training
- 2 **Invertible 1×1 convolutions**
 - Generalization of channel permutation
 - Learnable mixing of channels
 - $\det(W)$ computed via LU decomposition
- 3 **Affine coupling layers**
 - Similar to Real-NVP
 - More expressive conditioner networks

Results

- High-quality image synthesis (CelebA-HQ 256×256)
- Smooth interpolation in latent space
- Exact likelihood evaluation

MintNet (Song et al., 2019)

MintNet = Masked Invertible Neural Networks

Key Idea

Use masked convolutions (like PixelCNN) to build invertible CNNs

Properties

- 1 **Autoregressive structure**: Masking enforces ordering
- 2 **Triangular Jacobian**: Determinant is product of diagonal elements
- 3 **Invertibility**: Ensure diagonal elements are strictly positive

Advantages

- Leverage powerful CNN architectures
- Efficient determinant computation $O(n)$
- Can be inverted using fixed-point iteration

Applications

Density estimation on images (MNIST, CIFAR-10, ImageNet 32×32)

Core Principle

Transform data into Gaussian through iterative Gaussianization

Key Insight

For 1D random variable X with CDF F_X :

- $U = F_X(X)$ is uniform
- $Z = \Phi^{-1}(U) = \Phi^{-1}(F_X(X))$ is Gaussian

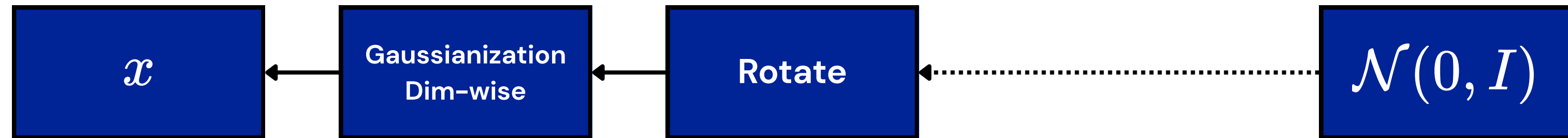
where Φ is the standard normal CDF

Two-Step Procedure

- 1 **Dimension-wise Gaussianization:** Apply inverse CDF transform to each dimension
 - Jacobian is diagonal (tractable!)
 - Each marginal becomes Gaussian
- 2 **Rotation:** Apply orthogonal matrix
 - $\mathcal{N}(0, I)$ is rotationally invariant
 - Jacobian is orthogonal ($|\det| = 1$, tractable!)

Stacking Layers

Repeat: Gaussianization \rightarrow Rotation \rightarrow Gaussianization \rightarrow Rotation \rightarrow ...



Advantages

- Principled approach (based on copula theory)
- All Jacobians are tractable
- Works well for tabular data

Implementation

- Learn monotonic transformations for dimension-wise step
- Parameterize rotation with Householder matrices

Exercise 2: Jacobian Computation for Coupling Layer

Goal: Compute the Jacobian determinant for the Real-NVP coupling layer

Given

Real-NVP transformation with partition at dimension d :

$$\begin{aligned}x_{1:d} &= z_{1:d} \\x_{d+1:n} &= z_{d+1:n} \odot \exp(\alpha_{\theta}(z_{1:d})) + \mu_{\theta}(z_{1:d})\end{aligned}$$

Tasks

- 1 Write out the Jacobian matrix $J = \frac{\partial x}{\partial z}$ explicitly
- 2 Show that J is lower triangular
- 3 Compute $\det(J)$
- 4 Express $\log |\det(J)|$ in a form suitable for gradient computation

Time: 10-15 minutes

Solution: Exercise 2 (Part 1)

Task 1: Jacobian Matrix

The transformation is:

$$\begin{pmatrix} x_{1:d} \\ x_{d+1:n} \end{pmatrix} = \begin{pmatrix} z_{1:d} \\ z_{d+1:n} \odot \exp(\alpha_\theta(z_{1:d})) + \mu_\theta(z_{1:d}) \end{pmatrix}$$

The Jacobian is:

$$J = \frac{\partial x}{\partial z} = \begin{pmatrix} \frac{\partial x_{1:d}}{\partial z_{1:d}} & \frac{\partial x_{1:d}}{\partial z_{d+1:n}} \\ \frac{\partial x_{d+1:n}}{\partial z_{1:d}} & \frac{\partial x_{d+1:n}}{\partial z_{d+1:n}} \end{pmatrix}$$

Computing each block:

$$\begin{aligned} \frac{\partial x_{1:d}}{\partial z_{1:d}} &= I_d \\ \frac{\partial x_{1:d}}{\partial z_{d+1:n}} &= 0 \\ \frac{\partial x_{d+1:n}}{\partial z_{d+1:n}} &= \text{diag}(\exp(\alpha_\theta(z_{1:d}))) \end{aligned}$$

Solution: Exercise 2 (Part 2)

Task 2: Lower Triangular Structure

$$J = \begin{pmatrix} I_d & 0 \\ \frac{\partial x_{d+1:n}}{\partial z_{1:d}} & \text{diag}(\exp(\alpha_\theta(z_{1:d}))) \end{pmatrix}$$

This is lower triangular because:

- Upper-right block is zero: $\frac{\partial x_{1:d}}{\partial z_{d+1:n}} = 0$
- Upper-left is identity: $\frac{\partial x_{1:d}}{\partial z_{1:d}} = I_d$
- Lower-right is diagonal: $\frac{\partial x_{d+1:n}}{\partial z_{d+1:n}} = \text{diag}(\exp(\alpha_\theta(z_{1:d})))$

Task 3: Determinant

For a lower triangular matrix, $\det(J)$ = product of diagonal elements:

$$\det(J) = 1 \cdot 1 \cdot \dots \cdot 1 \cdot \prod_{i=d+1}^n \exp(\alpha_\theta(z_{1:d})_i) = \prod_{i=d+1}^n \exp(\alpha_\theta(z_{1:d})_i)$$

Solution: Exercise 2 (Part 3)

Task 4: Log-Determinant

$$\begin{aligned}\log |\det(J)| &= \log \left| \prod_{i=d+1}^n \exp(\alpha_{\theta}(z_{1:d})_i) \right| \\ &= \log \left(\prod_{i=d+1}^n \exp(\alpha_{\theta}(z_{1:d})_i) \right) \\ &= \sum_{i=d+1}^n \log \exp(\alpha_{\theta}(z_{1:d})_i) \\ &= \sum_{i=d+1}^n \alpha_{\theta}(z_{1:d})_i \\ &= \mathbf{1}^T \alpha_{\theta}(z_{1:d})\end{aligned}$$

where $\mathbf{1}$ is the vector of ones.

Gradient Computation

This form is ideal for backpropagation:

Summary: Normalizing Flows

Key Principles

- 1 Map simple distributions to complex ones via invertible transformations
- 2 Use change of variables formula for exact likelihood
- 3 Design transformations with tractable Jacobian determinants

Architecture Families

- **Coupling Layers** (NICE, Real-NVP, Glow)
 - Fast inverse, tractable Jacobian
 - Multi-scale architectures for images
- **Autoregressive Flows** (MAF, IAF)
 - Trade-off between sampling and likelihood evaluation
 - Can be combined via distillation (Parallel WaveNet)
- **Continuous Transformations** (MintNet, Gaussianization)
 - Invertible CNNs, principled Gaussianization

Comparison: VAE vs Normalizing Flows

Property	VAE	Normalizing Flow
Likelihood	Intractable (ELBO)	Exact (change of variables)
Latent dim	Flexible ($z \in \mathbb{R}^m, x \in \mathbb{R}^n$)	Same as data ($z, x \in \mathbb{R}^n$)
Mapping	Stochastic	Deterministic
Inference	Amortized encoder	Direct inversion
Training	ELBO (lower bound)	Exact log-likelihood
Generation	Fast	Fast (forward pass)
Challenges	Posterior collapse	Architectural constraints

When to use which?

- **VAE**: Need dimension reduction, representation learning
- **Flows**: Need exact likelihoods, lossless compression

Recent Advances

- **Continuous Normalizing Flows** (Neural ODEs)
 - Infinitesimal transformations
 - Free-form Jacobians
- **Score-based Generative Models**
 - Connection to flows via probability flow ODE
 - State-of-the-art image generation
- **Diffusion Models**
 - Can be viewed as latent variable models
 - Hierarchical VAE interpretation

Open Questions

- Scaling flows to very high-dimensional data
- Bridging the gap between flows and diffusion models
- Theoretical understanding of expressivity

Foundational Papers

- NICE: Dinh et al., 2014
- Real-NVP: Dinh et al., 2017
- MAF: Papamakarios et al., 2017
- IAF: Kingma et al., 2016
- Glow: Kingma & Dhariwal, 2018
- Parallel WaveNet: van den Oord et al., 2018

Tutorials & Reviews

- Papamakarios et al., "Normalizing Flows for Probabilistic Modeling and Inference", 2021
- Kobyzev et al., "Normalizing Flows: An Introduction and Review of Current Methods", 2020

Implementations

- nflows library (PyTorch)
- distrax (JAX)

Thank you!

Questions?