Advanced Course on Deep Generative Models

Lecture 7: Normalizing Flows

Adapted from Stefano Ermon, Volodymyr Kuleshov

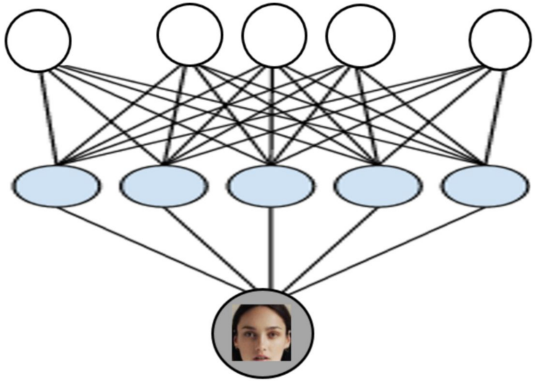
Dan Rosenbaum, CS Haifa

Today

- VAE recap

- Normalizing flow Background
  - Change of Variables Formula
  - Determinant, Jacobian
- Normalizing Flows Models
  - Representation and Learning
  - Composing Simple Transformations
  - Advanced Normalizing Flows

Latent variable models - deep learning approach



1. Don’t specify structure.
2. $p_\theta(x \mid z)$ is modeled via a high capacity deep neural network.
3. Need to model $p(x) = \int p(x \mid z)\, p(z)\, dz$

Image: Stefano Ermon

Variational bound

$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{y}) \frac{q(\mathbf{y}|\mathbf{x})}{q(\mathbf{y}|\mathbf{x})} \mathrm{d}\mathbf{y} = \mathbb{E}_{q(\mathbf{y}|\mathbf{x})} \left[\frac{p(\mathbf{x}, \mathbf{y})}{q(\mathbf{y}|\mathbf{x})} \right]$$

$$\log p(\mathbf{x}) = \log \mathbb{E}_{q(\mathbf{y}|\mathbf{x})} \left[\frac{p(\mathbf{x}, \mathbf{y})}{q(\mathbf{y}|\mathbf{x})} \right] \geq \mathbb{E}_{q(\mathbf{y}|\mathbf{x})} \left[\log \frac{p(\mathbf{x}, \mathbf{y})}{q(\mathbf{y}|\mathbf{x})} \right]$$

Variational bound, Evidence lower bound (ELBO)

Variational bound

$$\mathcal{L}_{\mathbf{x}}(q) = \mathbb{E}_{q(\mathbf{y}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{y})}{q(\mathbf{y}|\mathbf{x})}\right]$$

By manipulating the definition of the ELBO, we obtain the following equivalent forms

$$\begin{aligned}
\mathcal{L}_{\mathbf{x}}(q) &= \log p(\mathbf{x}) - \mathrm{KL}(q(\mathbf{y}|\mathbf{x})||p(\mathbf{y}|\mathbf{x})) && (13)\\
&= \mathbb{E}_{q(\mathbf{y}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{y}) - \mathrm{KL}(q(\mathbf{y}|\mathbf{x})||p(\mathbf{y})) && (14)\\
&= \mathbb{E}_{q(\mathbf{y}|\mathbf{x})} \log p(\mathbf{x},\mathbf{y}) + \mathcal{H}(q) && (15)
\end{aligned}$$

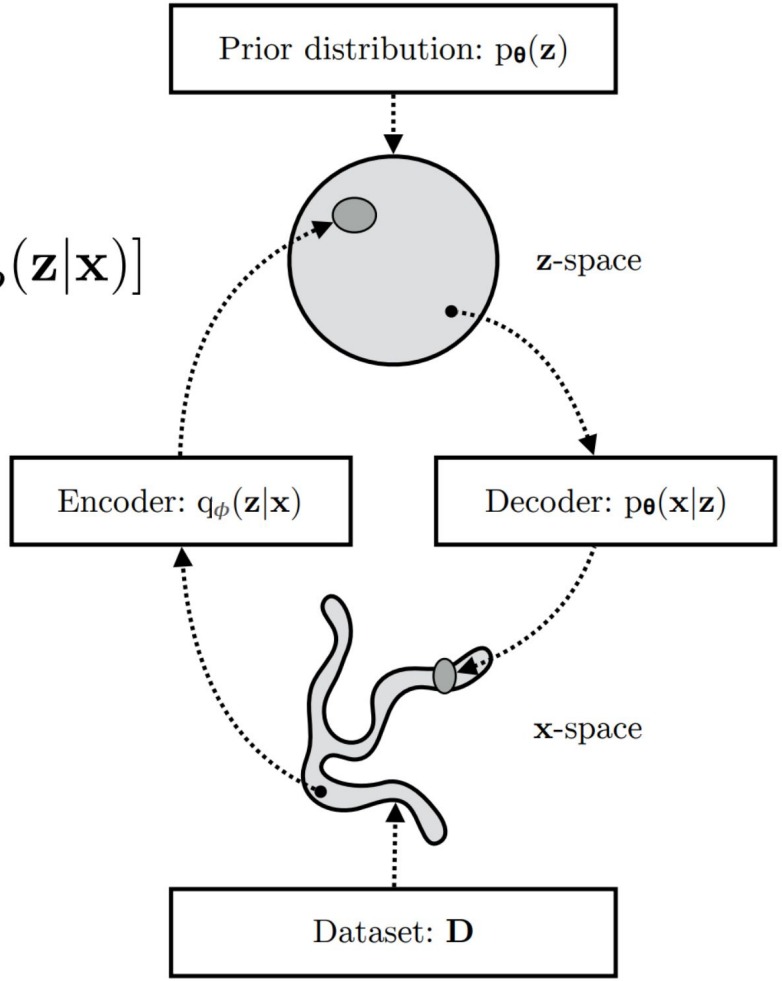
Variational AutoEncoder (VAE)

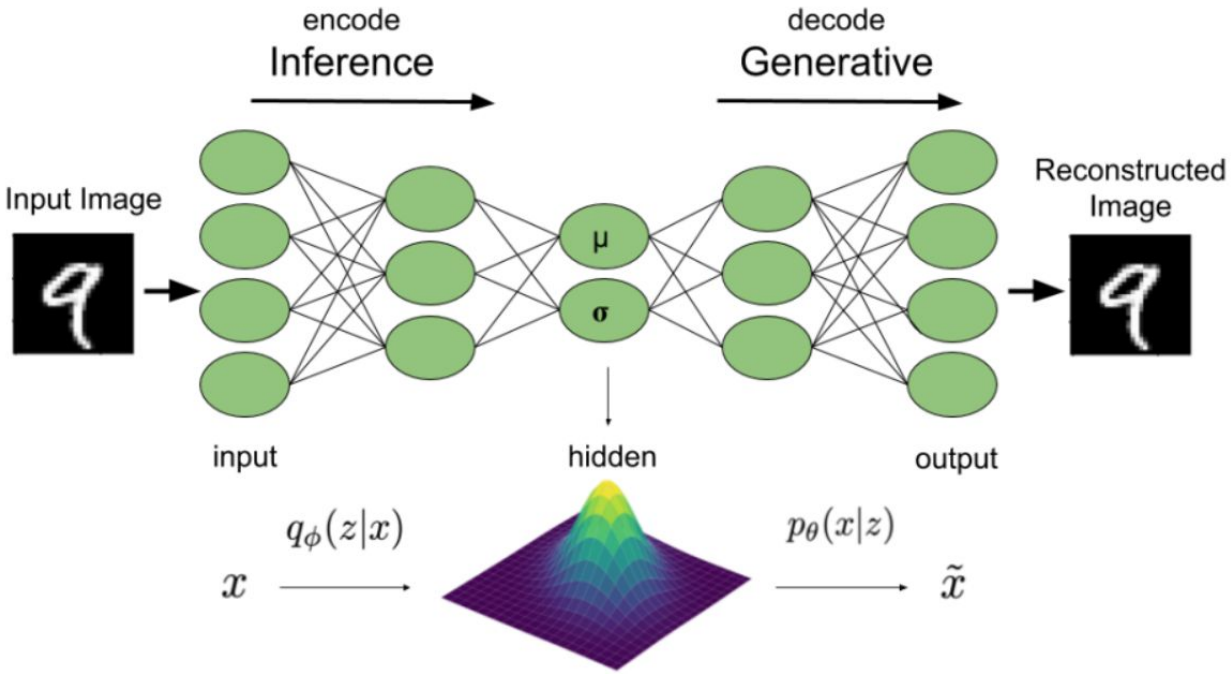
$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right]$$

Optimizing the ELBO w.r.t. $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ we get:

1. a better bound
2. a better model

Need to backpropagate through sampling → reparameterization trick.





$$E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z};\theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$$

Image: Stefano Ermon

KL between Gaussians

$$\int q_{\boldsymbol{\theta}}(\mathbf{z}) \log p(\mathbf{z})\, d\mathbf{z} = \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})\, d\mathbf{z}$$

$$= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^{J} (\mu_j^2 + \sigma_j^2)$$

And:

$$\int q_{\boldsymbol{\theta}}(\mathbf{z}) \log q_{\boldsymbol{\theta}}(\mathbf{z})\, d\mathbf{z} = \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)\, d\mathbf{z}$$

$$= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^{J} (1 + \log \sigma_j^2)$$

Therefore:

$$-D_{KL}((q_{\boldsymbol{\phi}}(\mathbf{z}) || p_{\boldsymbol{\theta}}(\mathbf{z})) = \int q_{\boldsymbol{\theta}}(\mathbf{z}) \left(\log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\boldsymbol{\theta}}(\mathbf{z})\right)\, d\mathbf{z}$$

$$= \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right)$$

Log-Likelihood for continuous data

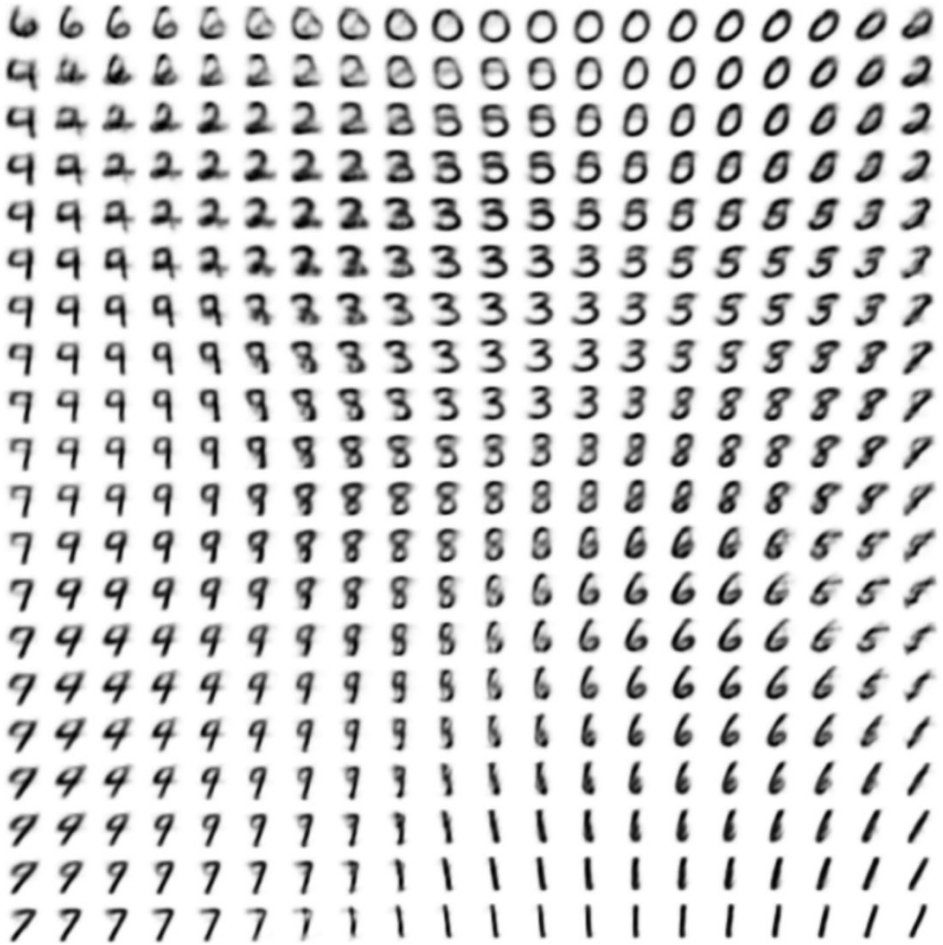
- In contrast to discrete modeling, the log-likelihood of continuous data can be tricky to interpret.
- What happens to the log-likelihood when one pixel is always black?
- Need to consider this in training and evaluation:
    - Training: limit the variance from below
    - Evaluation: Add uniform noise [0,1/256] to the values.

Results

(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 5: Random samples from learned generative models of MNIST for different dimensionalities of latent space.

2D latents



Results

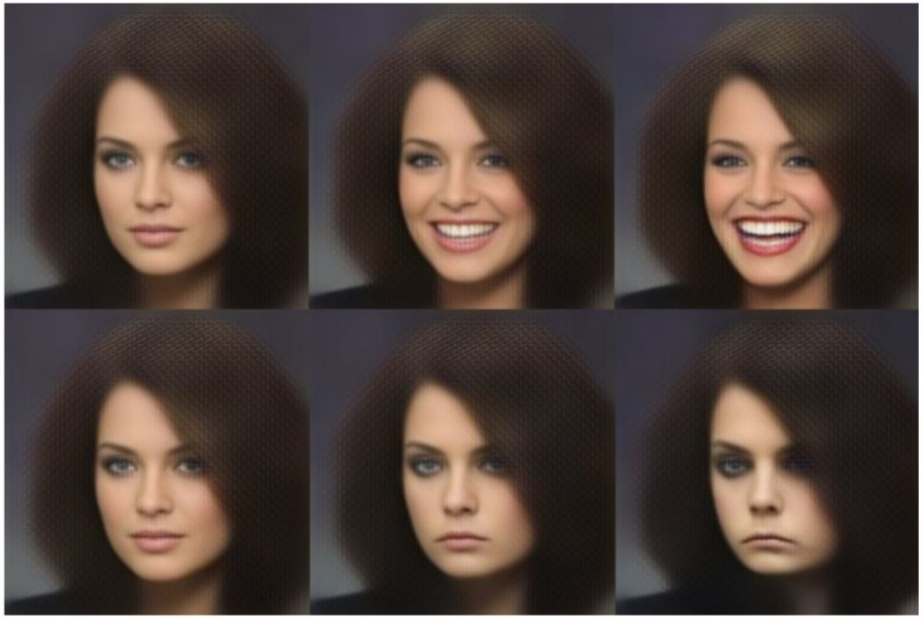
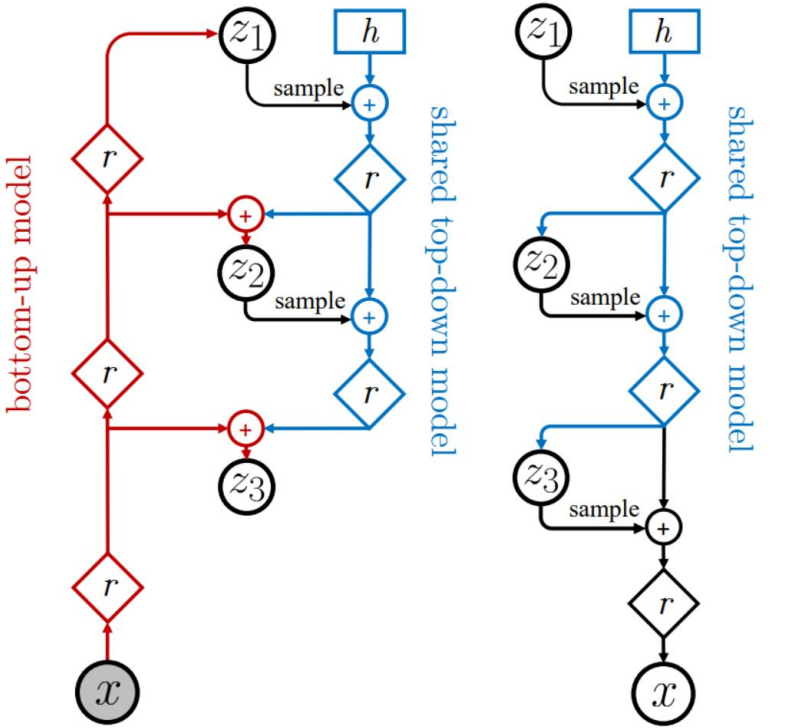


Figure 4.4: VAEs can be used for image resynthesis. In this example by White, 2016, an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness.

Extension - Hierarchical VAE

NVAE



(a) Bidirectional Encoder (b) Generative Model

Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. ◇ denotes residual neural networks, ⊕ denotes feature combination (e.g., concatenation), and [h] is a trainable parameter.

(a) MNIST ($t = 1.0$)

(b) CIFAR-10 ($t = 0.7$)

(c) CelebA 64 ($t = 0.6$)

(d) CelebA HQ ($t = 0.6$)

(e) FFHQ ($t = 0.5$)

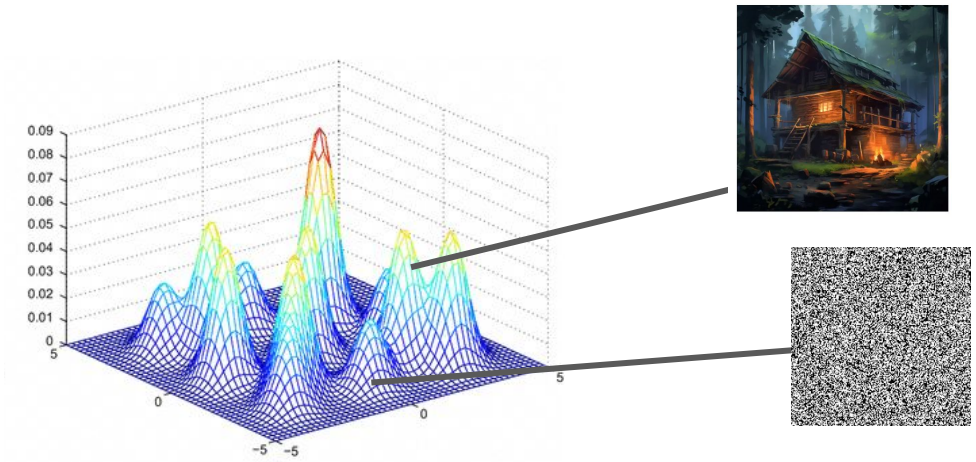
Generative models

Probabilistic model with high dimensional output.

- looking for the parameters θ such that p_θ is close to p_{data}

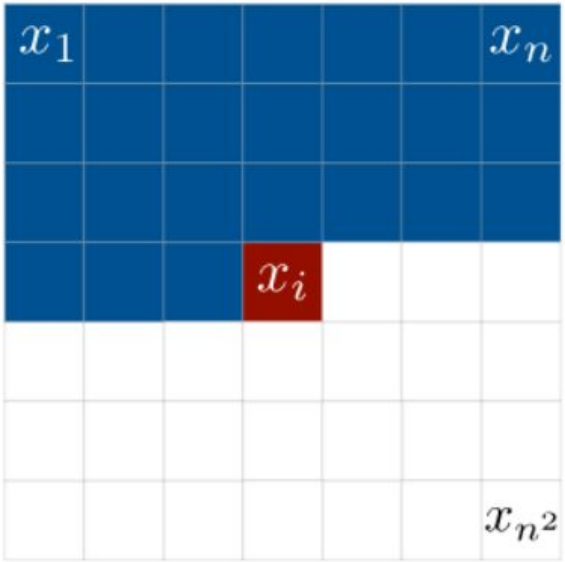
Usage:

- Generate data
- Representation learning
- Probabilistic inference



Recap - autoregressive models

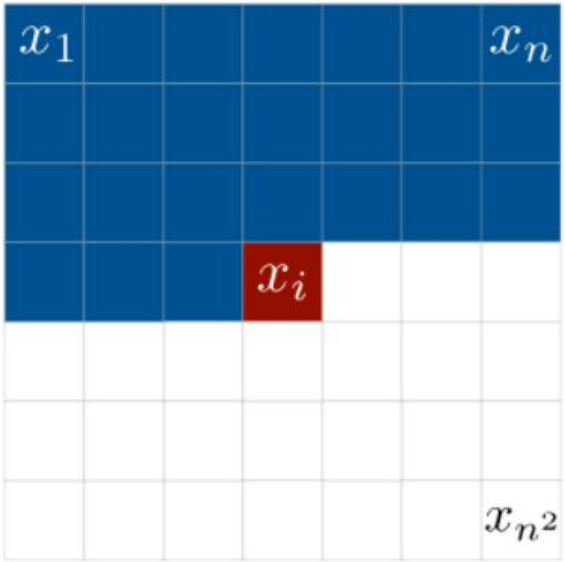
- Autoregressive models: $p_\theta(x) = \Pi\, p_\theta(x_i | x_{<i})$
- Probability distributions factorize into a product of factors
- We can efficiently represent **p** via conditional independence and/or neural parameterizations



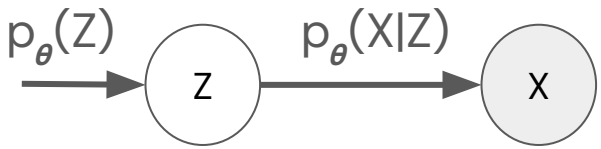
Recap - autoregressive models

- Pros:
  - Computationally tractable to evaluate likelihoods
  - Computationally tractable to train via maximum likelihood & gradient descent

- Cons:
  - Requires an arbitrary ordering of variables
  - Generation is sequential and usually slow
  - No natural learned representation



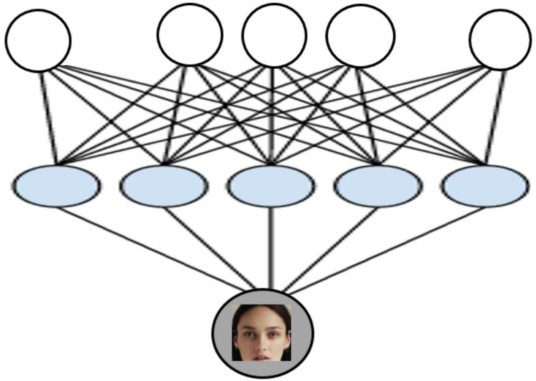
Recap - Variational Autoencoders



- Pros:
  - Naturally combine simple models into more flexible ones: $p(x) = \int p_\theta(x|z)p(z)dz$ with "simple" models $p_\theta(x|z)$, $p(z)$
  - Directed model permits efficient generation: $z \sim p(z)$, $x \sim p_\theta(x|z)$
- Cons:
  - Evaluating and training using exact log-likelihood is intractable
  - Fundamentally, the challenge is computing the posterior $p(z \mid x)$
  - Typically requires variational approximations

- **Key question:** Can we design a flexible latent variable model with tractable likelihoods? Yes! Use normalizing flows.

Latent variable models - deep learning approach



Need to model $p(x) = \int p(x \mid z)\, p(z)\, dz$

Normalizing flows: construct deterministic invertible mappings from **z** to **x.**

Simple to Complex Data Distributions

- We are looking for a latent variable model family with a tractable log-density ***log p(x)***.
- Many simple distributions have tractable densities:
  e.g., Gaussian, uniform, and mixtures.
- Unfortunately, these distributions are too simple to fit the data.

- **Key idea:** Map simple distributions (easy to sample and evaluate densities) to complex distributions (learned via data) using **invertible change of variables** transformations.

Change of Variables in One Dimension

- Let **Z** be a uniform random variable **U[0, 2]** with density p_Z .
- What is $p_Z(1)$?
  - 1/2
- Let **X = 4Z** , and let p_X be its density. What is $p_X(4)$?
  - $p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1) = p_Z(1) = 1/2$
- This is incorrect. Clearly, **X** is uniform in **[0, 8]**, so $p_X(4) = 1/8$
- Probability densities are not probability distributions (measures).
- Transformations expand the support of the distribution; we need to scale densities to preserve the volume of probability mass.

Change of Variables in One Dimension

Change of variables (1D case): If X = f(Z) and f(·) is monotone with inverse **Z = f^{-1}(X) = h(X)**, then:

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

- Previous example: If **X = 4Z** and **Z ~ U[0, 2]**, what is **$p_X(4)$**?
- Note that **h(X) = X/4**

$$p_X(4) = p_Z(1)\,h'(4) = 1/2 \times 1/4 = 1/8$$

- We have expanded the support of the distribution by **4**. Hence, we need to decrease the mass at each point by **4** to preserve the volume.

Change of Variables Formula (General Case)

Change of variables (General case): If the mapping between Z and X is given by **f:** $\mathbb{R}^n \rightarrow \mathbb{R}^n$, is invertible such that **X = f(Z)** and $\mathbf{Z = f^{-1}(X)}$, then:

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

- **Note 1:** **x** and **z** need to be continuous and have the same dimension.
- **Note 2:** For any invertible matrix **A**, $\mathbf{det(A^{-1}) = det(A)^{-1}}$. Therefore:

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

Change of Variables Formula (General Case)

Proof: For the case when **f** is monotonically increasing:

$$P_y(y) \triangleq P(Y \le y) = \int_{-\infty}^{y} p_y(\tilde{y})\, d\tilde{y}$$

$$P(Y \le y) = P(f(X) \le y) = P(X \in \{x \mid f(x) \le y\})$$

differentiating:

$$p_y(y) \triangleq \frac{\partial}{\partial y} P_y(y) = \frac{\partial}{\partial y} P_x\left(f^{-1}(y)\right) = \frac{\partial f^{-1}(y)}{\partial y} \frac{\partial}{\partial f^{-1}(y)} P_x\left(f^{-1}(y)\right)$$

$$= \frac{\partial f^{-1}(y)}{\partial y} p_x\left(f^{-1}(y)\right)$$

Change of Variables in One Dimension: Intuition

Change of variables (1D case): If X = f(Z) and f(·) is monotone with inverse **Z = f^{-1}(X) = h(X)**, then:

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

An integral is a sum of “infinitesimal rectangles” dz and dx. We adjust the “volume” of each dx around x because h changes it.

$$\int p_Z(z)dz = \int p_Z(z)\frac{dx}{dx}dz = \int p_Z(h(x))\left|\frac{dz}{dx}\right| dx$$

$$= \int p_Z(h(x))\left|h'(x)\right| dx$$

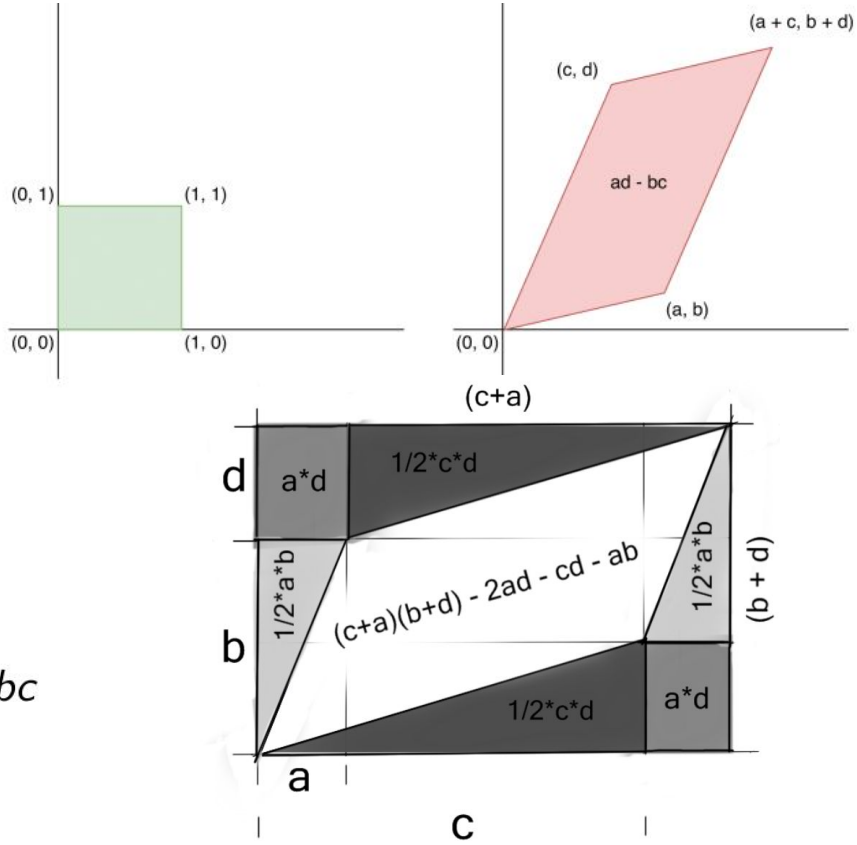
Review: Determinants and Volumes (in 2D)

- Matrix $A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ maps a unit square to a parallelogram, e.g.:

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

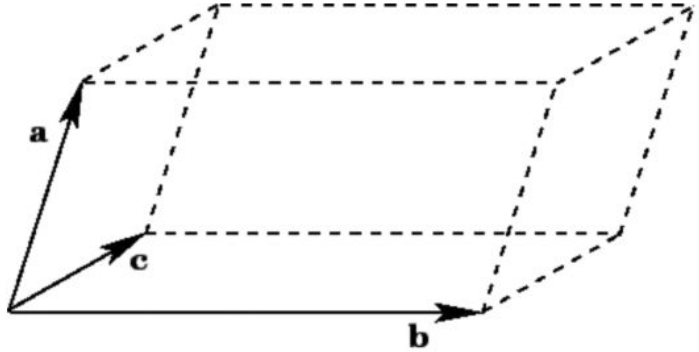
- The volume of the parallelotope is equal to the determinant of A

$$\det(A) = \det \begin{pmatrix} a & c \\ b & d \end{pmatrix} = ad - bc$$



Review: Determinants and Volumes (in 3D)

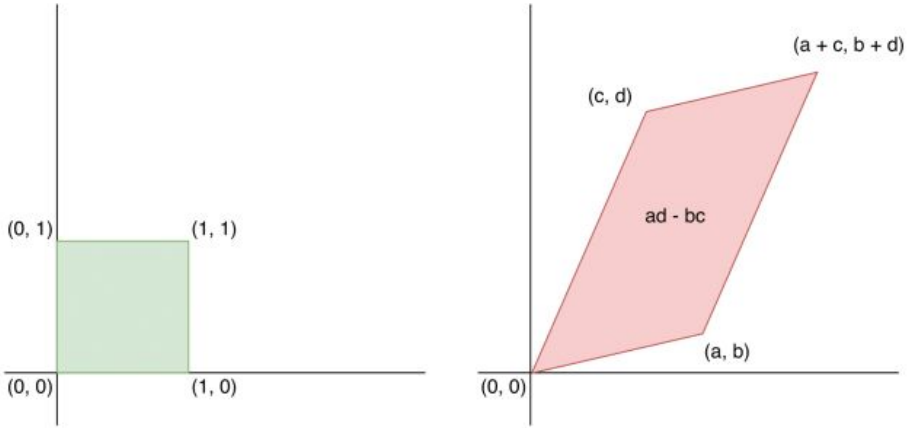
The volume formula still holds in 3D.



Note that if two vectors are colinear, we get a plane, which has volume zero in 3D. The determinant is zero and the matrix is singular.

Review: Determinants and Volumes (in n-D)

- In general, the matrix A maps the unit hypercube $[0, 1]^n$ to a parallelotope
- Hypercube and parallelotope are generalizations of square/cube and parallelogram/parallelepiped to higher dimensions
- Determinant det(A) still gives volume of the n-D shape.



Change of Variables Formula (Linear Case in n-D)

- Let **Z** be a uniform random vector in $[0, 1]^n$
- Let **X = AZ** for a square invertible matrix **A**, with inverse **W = A^{-1}** . How is **X** distributed?
- The volume of the parallelotope is equal to the determinant of the transformation **A**:

$$\det(A) = \det\begin{pmatrix} a & c \\ b & d \end{pmatrix} = ad - bc$$

- X is uniformly distributed over the parallelotope. Hence, we have:

$$\begin{aligned} p_X(\mathbf{x}) &= p_Z\left(W\mathbf{x}\right)\left|\det(W)\right| \\ &= p_Z\left(W\mathbf{x}\right) / \left|\det(A)\right| \end{aligned}$$

Change of Variables Formula (General Case)

- For linear transformations specified via **A**, change in volume is given by the determinant of **A**

- For non-linear transformations **f(·)**, the linearized change in volume is given by the determinant of the **Jacobian** of **f(·)**.

Review: The Jacobian

Consider a vector valued function $f : \mathbb{R}^n \to \mathbb{R}^m$, with:

- $\mathbf{x} = (x_1, \cdots, x_n)$
- $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_m(\mathbf{x}))$

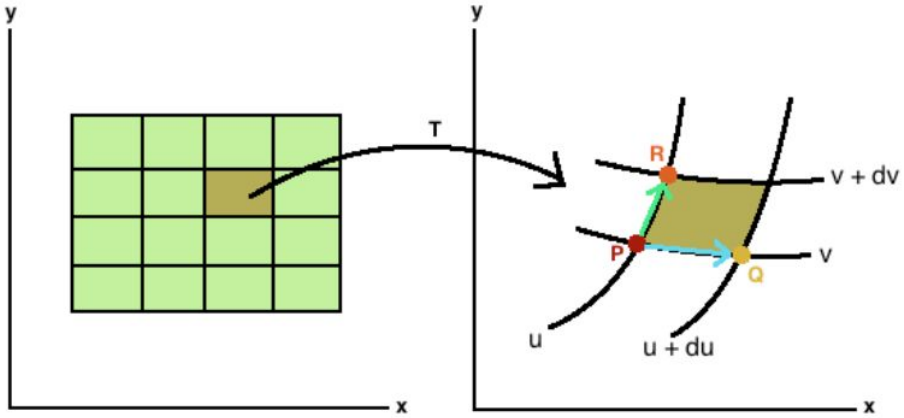
The Jacobian is defined as:

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

This generalizes the gradient to multi-variate functions.

Change of Variables Formula (General Case): Intuition

- We are interested in mapping a small volume between **(v , u)** and **(v + dv , u + du)**.
- For sufficiently small **du, dv** , the function can be linearized, and becomes the linear mapping specified by the Jacobian.



Change of Variables Formula (General Case)

Change of variables (General case): If the mapping between Z and X is given by **f:** $\mathbb{R}^n \rightarrow \mathbb{R}^n$, is invertible such that **X = f(Z)** and **Z =** $\mathbf{f}^{-1}$**(X)**, then:

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

- **Note 1: x** and **z** need to be continuous and have the same dimension.
- **Note 2:** For any invertible matrix **A**, **det(**$\mathbf{A}^{-1}$**) = det(A)**$^{-1}$. Therefore:

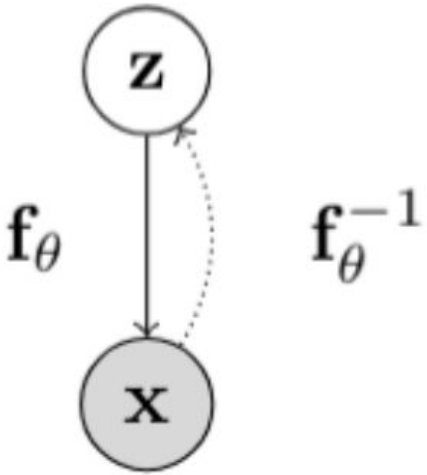
$$p_X(\mathbf{x}) = p_Z(\mathbf{z})\left|\det\left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}}\right)\right|^{-1}$$

Normalizing Flows

Normalizing Flow Models: Representation

- A normalizing flow model is a directed, latent-variable model over observed variables **X** and latent variables **Z**

- The mapping between **Z** and **X** , given by $\mathbf{f_\theta}: \mathbb{R}^n \rightarrow \mathbb{R}^n$, Is deterministic and invertible such that $\mathbf{X = f_\theta(Z)}$ and $\mathbf{Z = f_\theta^{-1}(X)}$

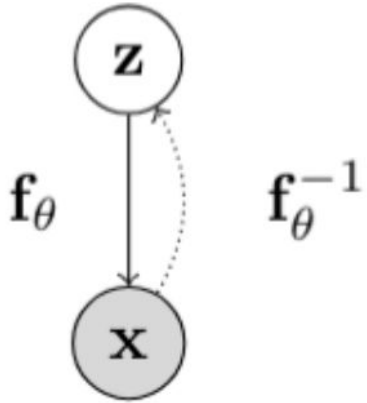


Normalizing Flow Models: Learning

- We want to learn $\mathbf{p_X(x; \theta)}$ using the principle of maximum likelihood.

- Using change of variables, the marginal likelihood $\mathbf{p_X(x; \theta)}$ is given by:

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



- **Note 1:** Unlike in VAEs, we compute the marginal likelihood exactly!
- **Note 2**: **x, z** need to be continuous and have the same dimension.

Normalizing Flow Models: Constructing f

We need to construct a density transformation that is:

1. Invertible, so that we can apply the change of variables formula.
2. Expressive, so that we can learn complex distributions.
3. Computationally tractable, so that we can optimize and evaluate it.

One approach is to start with a simple distribution (e.g. Gaussian) and then compose multiple simple transformation.

Constructing f by composing simple transformations

- Start with a simple distribution for $\mathbf{z}_0$ (e.g., Gaussian)
- Apply sequence of M **simple** invertible transformations with $\mathbf{x} \triangleq \mathbf{z}_M$

$$\mathbf{z}_m := \mathbf{f}_\theta^m \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^m(\mathbf{f}_\theta^{m-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0)$$

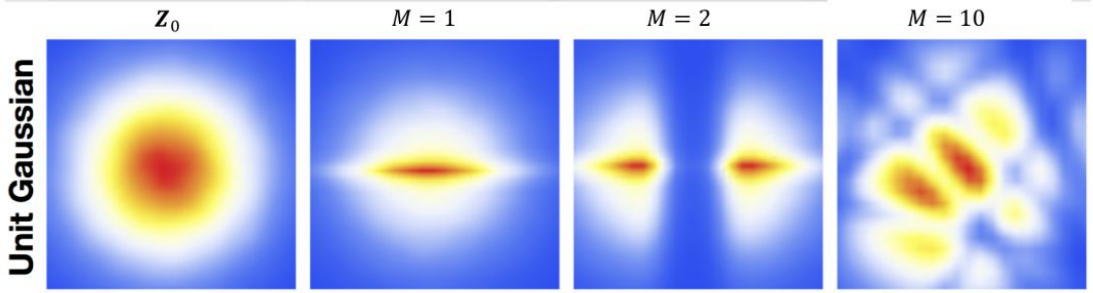
- By change of variables

$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right) \prod_{m=1}^{M} \left| \det \left(\frac{\partial (\mathbf{f}_\theta^m)^{-1}(\mathbf{z}_m)}{\partial \mathbf{z}_m} \right) \right|$$

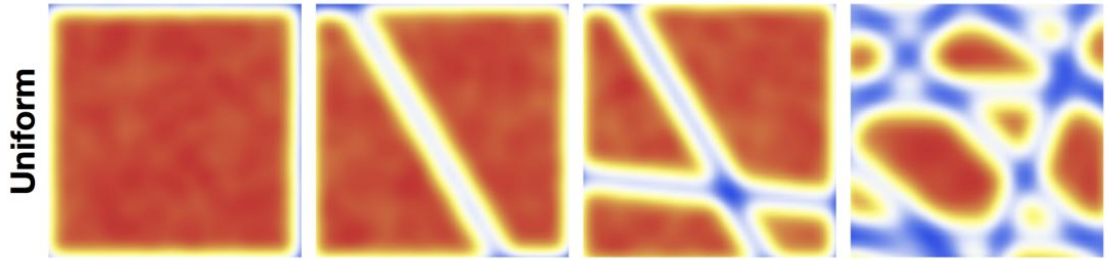
(Note: determininant of composition equals product of determinants)

Example: Planar Flows

- Base distribution: Gaussian



- Base distribution: Uniform



Normalizing Flows: Recap

Normalizing: Change of variables gives a normalized density after applying an invertible transformation.
Flow: The function f makes the probability mass smoothly flow from a simple distribution over the space to one that is complex.

- Transformations need to be invertible, hence $\dim(X) = \dim(Z)$.
- Complex transformations can be composed from simple ones:

$$\mathbf{z}_m := \mathbf{f}_\theta^m \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^m(\mathbf{f}_\theta^{m-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0)$$

Normalizing Flows: Recap

- **Exact likelihood evaluation** via inverse tranformation $\mathbf{x} \mapsto \mathbf{z}$ and change of variables formula

- Learning via **maximum likelihood** over the dataset $\mathcal{D}$

$$\max_{\theta} \log p_X(\mathcal{D}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right) + \log \left| \det \left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

Normalizing Flows: Recap

- **Sampling** via forward transformation $\mathbf{z} \mapsto \mathbf{x}$

$$\mathbf{z} \sim p_Z(\mathbf{z}) \quad \mathbf{x} = \mathbf{f}_\theta(\mathbf{z})$$

- **Latent representations** inferred via inverse transformation (no inference network required!)

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

Challenges in Building Flow Models

- Complex, invertible transformations with tractable evaluation:
  - Likelihood evaluation requires efficient evaluation of $\mathbf{x} \mapsto \mathbf{z}$ mapping
  - Sampling requires efficient evaluation of $\mathbf{z} \mapsto \mathbf{x}$ mapping
- Computing likelihoods also requires the evaluation of determinants of $n \times n$ Jacobian matrices, where n is the data dimensionality
  - Computing the determinant for an $n \times n$ matrix is $O(n^3)$: prohibitively expensive within a learning loop!

Key idea: Choose tranformations so that the resulting Jacobian matrix has special structure. For example, the determinant of a triangular matrix is the product of the diagonal entries, i.e., an $O(n)$ operation

Triangular Jacobian

$$\mathbf{x} = (x_1, \cdots, x_n) = \mathbf{f}(\mathbf{z}) = (f_1(\mathbf{z}), \cdots, f_n(\mathbf{z}))$$

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

Suppose $x_i = f_i(\mathbf{z})$ only depends on $\mathbf{z}_{\leq i}$. Then

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & 0 \\ \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial z_1} & \cdots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$$

has lower triangular structure. Determinant is computed in **linear time**.

Nonlinear Independent Components Estimation (NICE)

Nonlinear Independent Components Estimation (NICE; Dinh et al., 2014) is a flow-based model, where the transformation

$$x \leftarrow \mathbf{f}_\theta^m \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^m(\mathbf{f}_\theta^{m-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0)$$

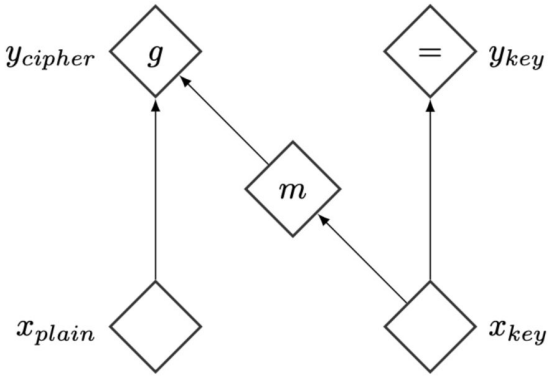
is made of a composition of two types of layers:

1. Additive coupling layers
2. Rescaling layers

NICE: Additive Coupling Layers

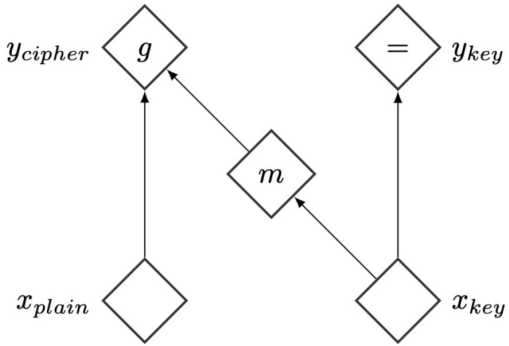
An additive coupling layer has the following structure:

- First, we partition the variables $\mathbf{z}$ into two disjoint subsets, say $\mathbf{z}_{1:d}$ and $\mathbf{z}_{d+1:n}$ for any $1 \leq d < n$
- We define the forward mapping $\mathbf{z} \mapsto \mathbf{x}$ as follows:
  - The first set of variables stays the same: $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$
  - The second variables undergo an affine transformation: $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d})$



- $m_\theta(\cdot)$ is a DNN with params θ, d input units, and $n - d$ output units

NICE: Additive Coupling Layers



Is this invertible?
Yes!

- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d})$ ($m_\theta(\cdot)$ is a neural network with parameters θ, d input units, and $n - d$ output units)
- Inverse mapping $\mathbf{x} \mapsto \mathbf{z}$: defining $z \leftarrow \mathbf{f}^{-1}(x)$
  - The first d dimensions are unchanged: $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$ (identity transformation)
  - The other dimensions are simply shifted (using the fact that the first dimensions are unchanged): $\mathbf{z}_{d+1:n} = \mathbf{x}_{d+1:n} - m_\theta(\mathbf{x}_{1:d})$

NICE: Additive Coupling Layers

Is the Jacobian tractable? Yes!

- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d})$ ($m_\theta(\cdot)$ is a neural network with parameters θ, d input units, and $n - d$ output units)
- Jacobian of forward mapping:

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

- Observe that:

$$\det(J) = 1$$

  - We have a **volume preserving transformation** since determinant is 1.
  - Inverse mapping can be computed for any m.
  - Determinant is independent of m_θ, hence we can use any function!

NICE: Rescaling Layers

Rescaling layers in NICE are defined as follows:

- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:

$$x_i = s_i z_i$$

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

- Inverse mapping $\mathbf{x} \mapsto \mathbf{z}$:

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

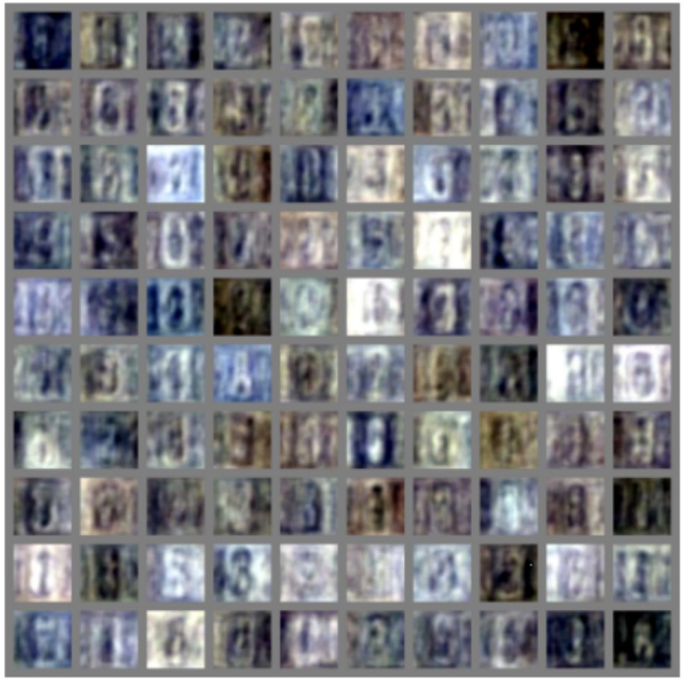
- Jacobian of forward mapping: $J = \text{diag}(\mathbf{s})$ $\det(J) = \prod_{i=1}^{n} s_i$

Samples Generated via NICE

(a) Model trained on MNIST

(b) Model trained on TFD

Samples Generated via NICE



(c) Model trained on SVHN

(d) Model trained on CIFAR-10

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)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\theta(\mathbf{z}_{1:d})) + \mu_\theta(\mathbf{z}_{1:d})$
  - $\mu_\theta(\cdot)$ and $\alpha_\theta(\cdot)$ are both neural networks with parameters θ, 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)
  - $\mathbf{z}_{d+1:n} = (\mathbf{x}_{d+1:n} - \mu_\theta(\mathbf{x}_{1:d})) \odot (\exp(-\alpha_\theta(\mathbf{x}_{1:d})))$

Real-NVP: Non-Volume Preserving Extension of NICE

- Jacobian of forward mapping:

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

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

- **Non-volume preserving transformation** in general since determinant can be less than or greater than 1

Samples Generated via Real-NVP

Latent Space Interpolations via Real-NVP

Connection between Autoregressive and Flow Models

Autoregressive Models as Flow Models

- Consider a Gaussian autoregressive model:

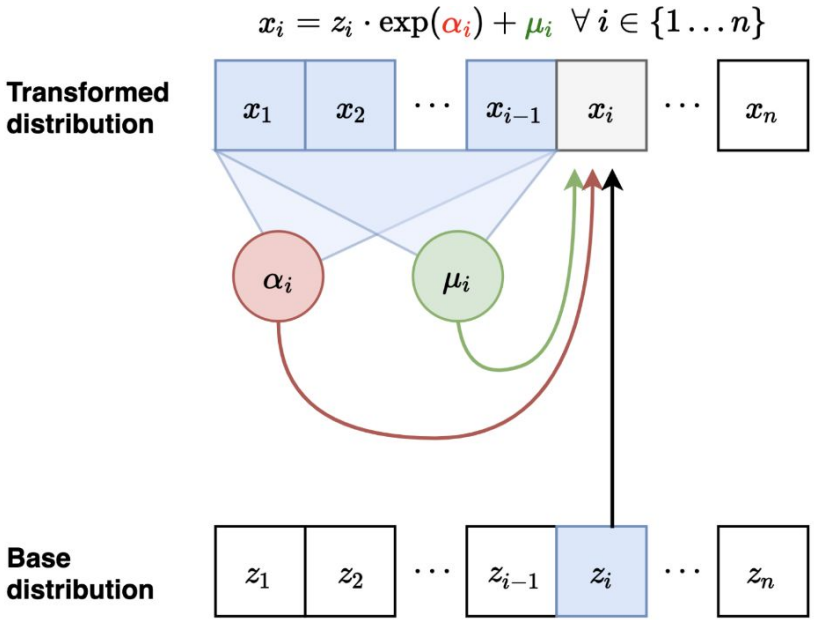
$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|\mathbf{x}_{<i})$$

  such that $p(x_i \mid \mathbf{x}_{<i}) = \mathcal{N}(\mu_i(x_1, \cdots, x_{i-1}), \exp(\alpha_i(x_1, \cdots, x_{i-1}))^2)$.
- $\mu_i(\cdot)$ and $\alpha_i(\cdot)$ are neural networks for $i > 1$ and constants for $i = 1$.

Autoregressive Models as Flow Models

- Consider a sampler for this model:
  - Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \cdots, n$
  - 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 $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
  - Let $x_3 = \exp(\alpha_3) z_3 + \mu_3$. ...
- This defines an invertible transformation from z to x. Hence, this type of autoregressive model can be **interpreted as a flow**!

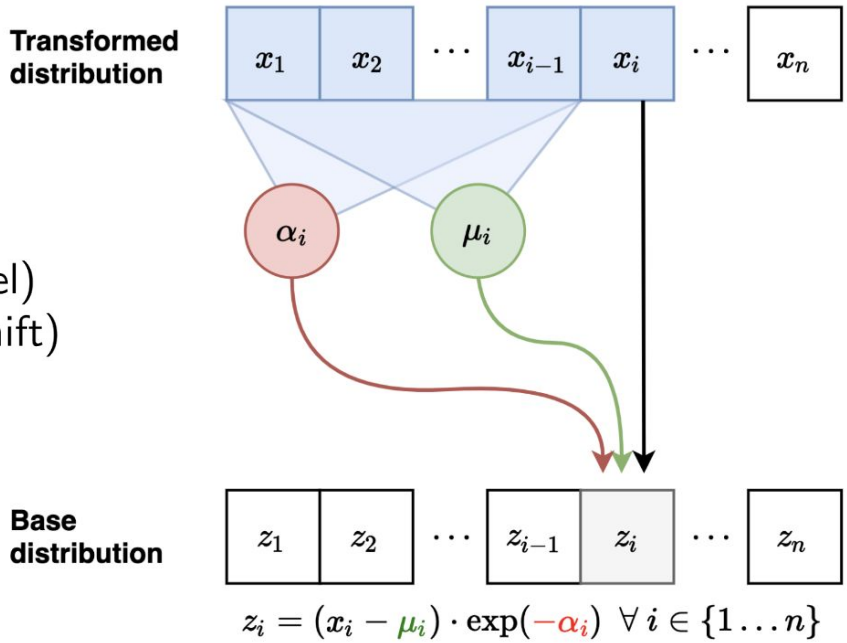
Masked Autoregressive Flow



- Computing the forward mapping (i.e. sampling) is sequential and slow: $O(n)$ time (it's an autoregressive model)

Masked Autoregressive Flow

- Inverse mapping from $\mathbf{x} \mapsto \mathbf{z}$:
  - Compute all μ_i, α_i (can be done in parallel)
  - Let $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$ (scale and shift)
  - Let $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$
  - Let $z_3 = (x_3 - \mu_3)/\exp(\alpha_3)$ ...



- Jacobian is lower diagonal; determinant is computed efficiently
- Inverse mapping (i.e., likelihood evaluation) is easy and parallelizable

Inverse Autoregressive Flow

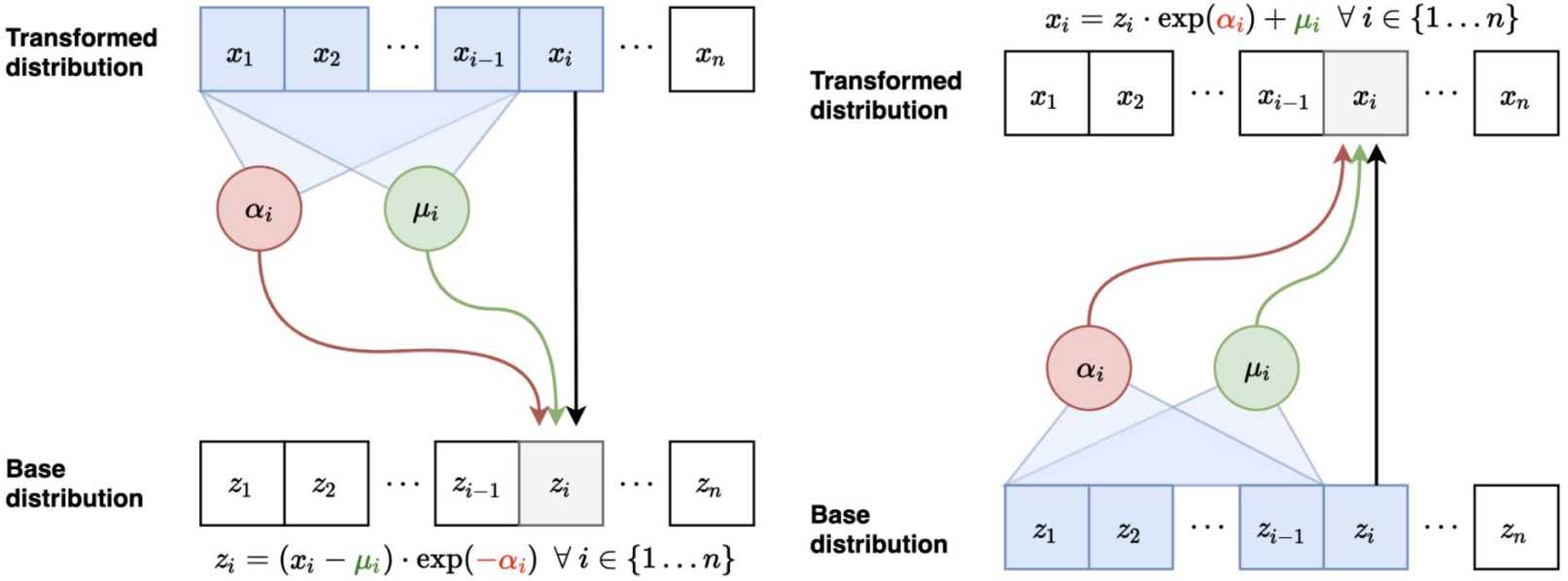
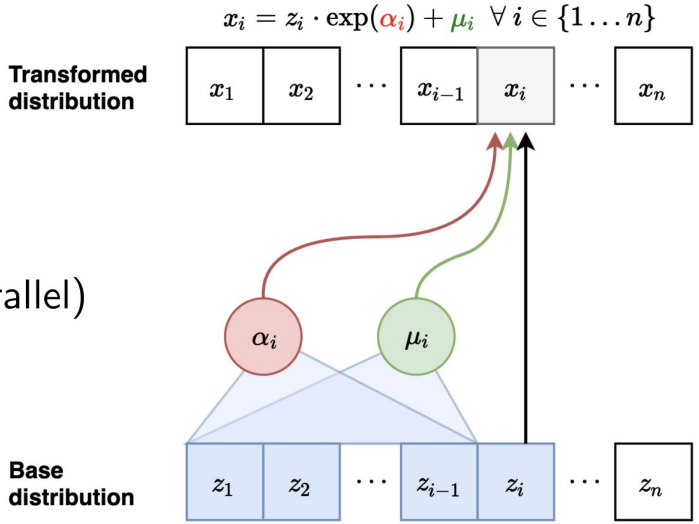


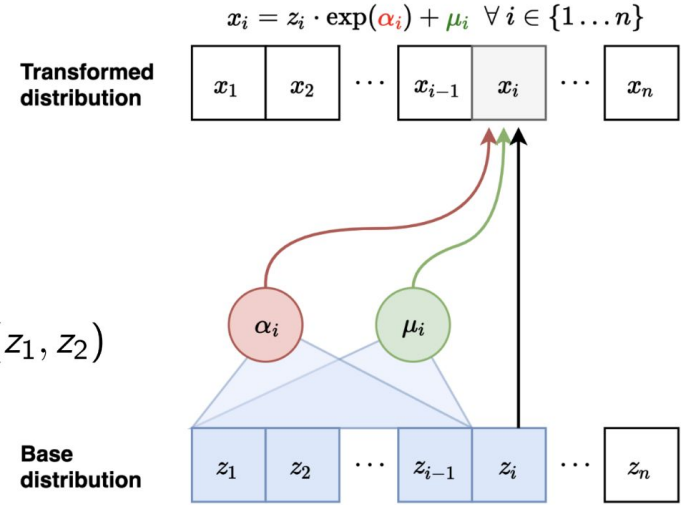
Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

Inverse Autoregressive Flow

- Forward mapping from $\mathbf{z} \mapsto \mathbf{x}$:
  - Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \cdots, n$
  - Compute all $\mu_i(z_{<i}), \alpha_i(z_{<i})$ (can be done in parallel)
  - Let $x_1 = \exp(\alpha_1)z_1 + \mu_1$
  - Let $x_2 = \exp(\alpha_2)z_2 + \mu_2$ ...



Inverse Autoregressive Flow

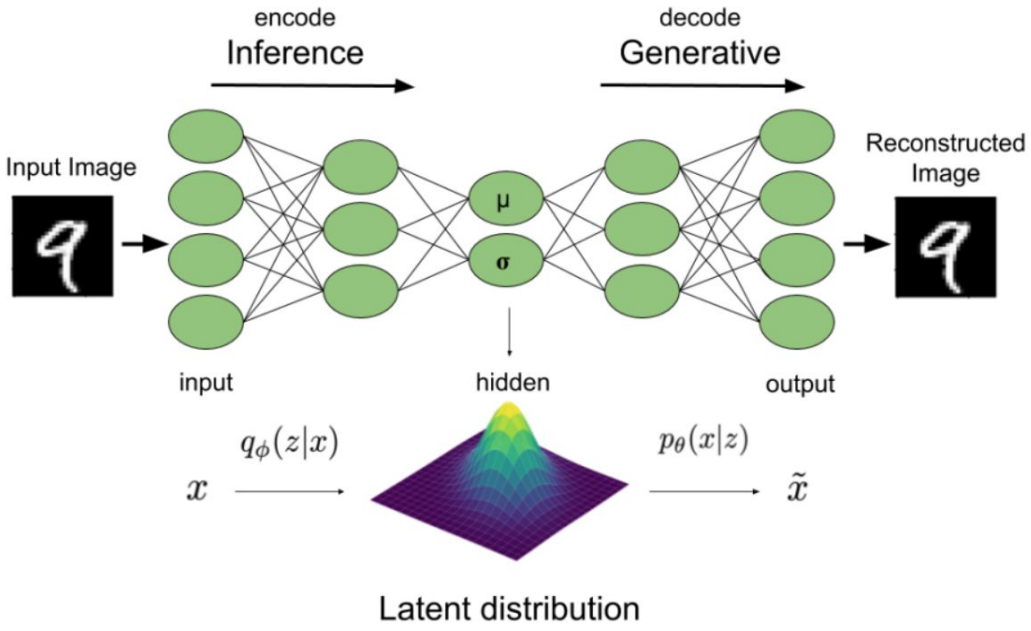


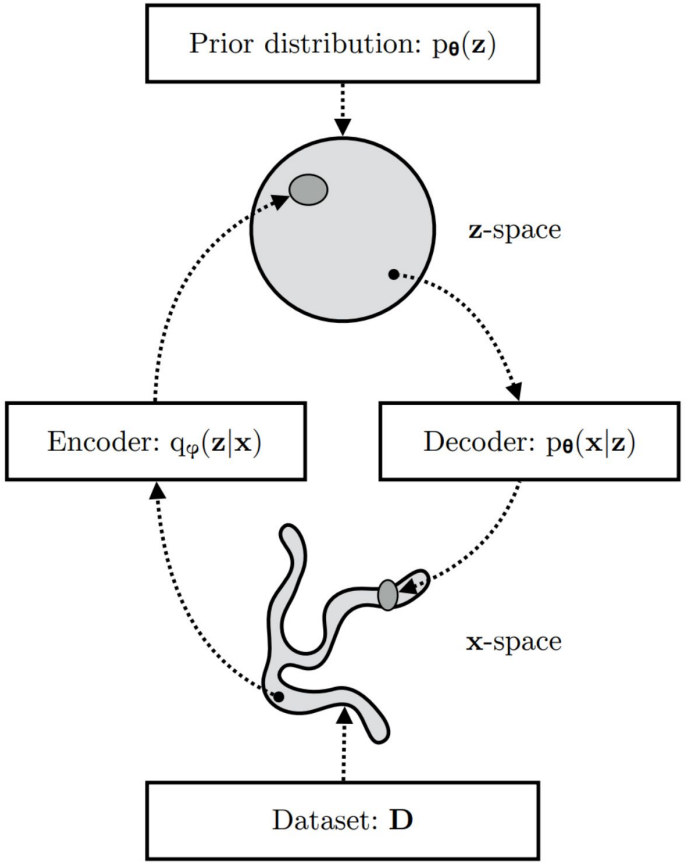
- Inverse mapping from $\mathbf{x} \mapsto \mathbf{z}$ (sequential):
  - Let $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$. Compute $\mu_2(z_1), \alpha_2(z_1)$
  - Let $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$. Compute $\mu_3(z_1, z_2), \alpha_3(z_1, z_2)$

- It's fast to sample from an IAF, but slow to evaluate likelihoods and train
- Note: Fast to evaluate likelihoods of a generated point (cache $z_1, z_2, \dots, z_n$)

Where would this be useful?

VAE

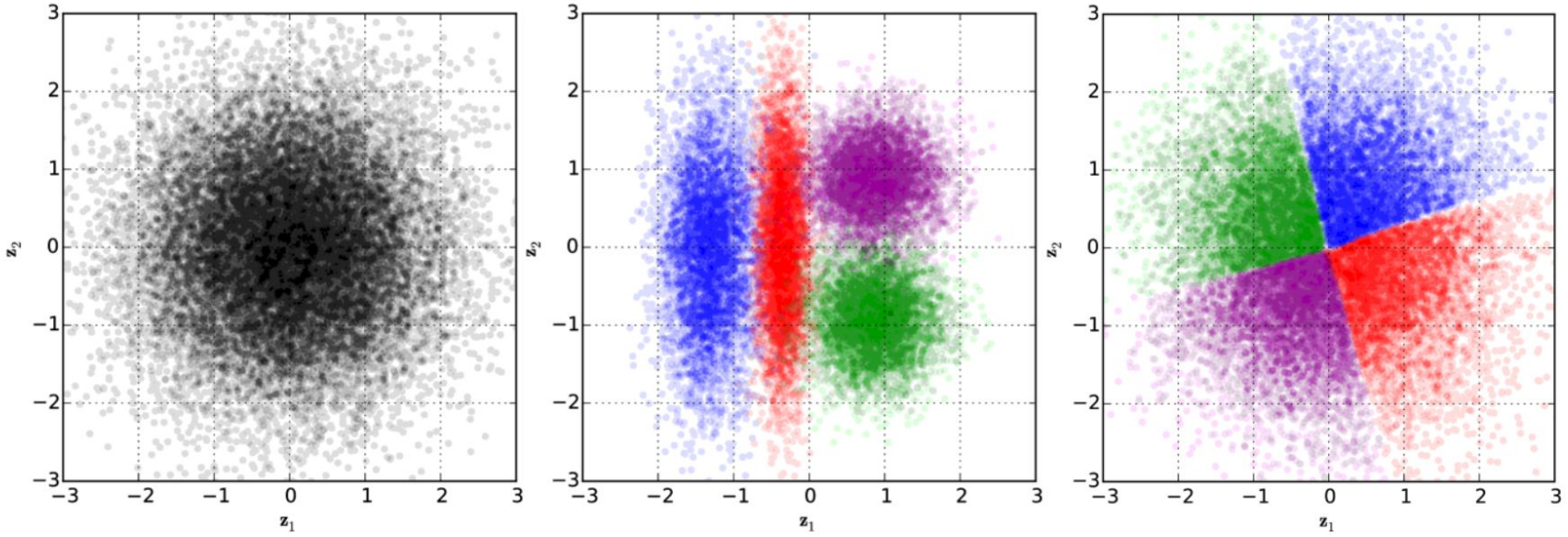




$$E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z};\theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$$

Modeling the posterior

Kingma et al. 2017



(a) Prior distribution (b) Posteriors in standard VAE (c) Posteriors in VAE with IAF

Table 2: Our results with ResNet VAEs on CIFAR-10 images, compared to earlier results, in *average number of bits per data dimension* on the test set. The number for convolutional DRAW is an upper bound, while the ResNet VAE log-likelihood was estimated using importance sampling.

Method	bits/dim $\leq$
Results with tractable likelihood models:	
Uniform distribution (van den Oord et al., 2016b)	8.00
Multivariate Gaussian (van den Oord et al., 2016b)	4.70
NICE (Dinh et al., 2014)	4.48
Deep GMMs (van den Oord and Schrauwen, 2014)	4.00
Real NVP (Dinh et al., 2016)	3.49
PixelRNN (van den Oord et al., 2016b)	**3.00**
Gated PixelCNN (van den Oord et al., 2016c)	**3.03**
Results with variationally trained latent-variable models:	
Deep Diffusion (Sohl-Dickstein et al., 2015)	5.40
Convolutional DRAW (Gregor et al., 2016)	3.58
ResNet VAE with IAF (Ours)	**3.11**

Negative log-likelihood

bit/dimension for all datasets except for MNIST which is in nats.

Method	MNIST 28×28	CIFAR-10 32×32	ImageNet 32×32	CelebA 64×64	CelebA HQ 256×256	FFHQ 256×256
NVAE w/o flow	**78.01**	2.93	-	2.04	-	0.71
NVAE w/ flow	78.19	**2.91**	3.92	**2.03**	**0.70**	**0.69**
VAE Models with an Unconditional Decoder						
BIVA [36]	78.41	3.08	3.96	2.48	-	-
IAF-VAE [4]	79.10	3.11	-	-	-	-
DVAE++ [20]	78.49	3.38	-	-	-	-
Conv Draw [42]	-	3.58	4.40	-	-	-
Flow Models without any Autoregressive Components in the Generative Model						
VFlow [59]	-	2.98	-	-	-	-
ANF [60]	-	3.05	3.92	-	0.72	-
Flow++ [61]	-	3.08	**3.86**	-	-	-
Residual flow [50]	-	3.28	4.01	-	0.99	-
GLOW [62]	-	3.35	4.09	-	1.03	-
Real NVP [63]	-	3.49	4.28	3.02	-	-
VAE and Flow Models with Autoregressive Components in the Generative Model						
δ-VAE [25]	-	2.83	3.77	-	-	-
PixelVAE++ [35]	78.00	2.90	-	-	-	-
VampPrior [64]	78.45	-	-	-	-	-
MAE [65]	77.98	2.95	-	-	-	-
Lossy VAE [66]	78.53	2.95	-	-	-	-
MaCow [67]	-	3.16	-	-	0.67	-
Autoregressive Models						
SPN [68]	-	-	3.85	-	0.61	-
PixelSNAIL [34]	-	2.85	3.80	-	-	-
Image Transformer [69]	-	2.90	3.77	-	-	-
PixelCNN++ [70]	-	2.92	-	-	-	-
PixelRNN [41]	-	3.00	3.86	-	-	-
Gated PixelCNN [71]	-	3.03	3.83	-	-	-

Summary of Normalizing Flow Models

- Transform simple distributions into more complex distributions via change of variables
- Normalizing Flows Pros:
  - Exact marginal likelihood p(x) is tractable to compute and optimize
  - Exact posterior inference p(z|x) is tractable
- Normalizing Flows Cons:
  - Only works for continuous variables
  - Dimensionality of z and x must be the same (can pose computational challenges).
  - Places important constraints on what model family we can use.
- Strategies for constructing flows
  - Composition of simple bijections
  - Triangular Jacobian
  - Can be interpreted as model with a certain autoregressive structure that influences speed of forward and inverse sampling.