Advanced Flow Models

Volodymyr Kuleshov

Cornell Tech

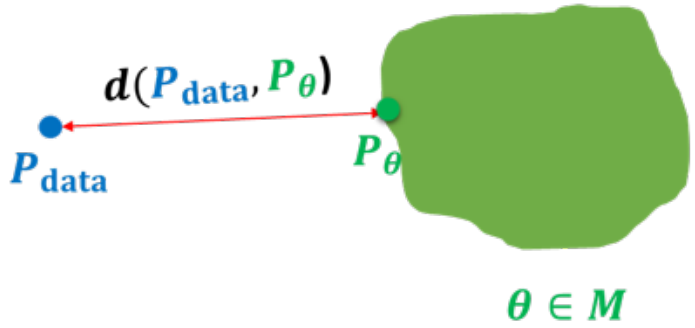
Lecture 8

Announcements

- Congratulations on completing Assignment 1!
- Assignment 2 will be out today and due in two weeks
- Project proposals are due in about one week in Gradescope
  - Come see me during office hours (or schedule time with me) to get feedback on your project ideas

Normalizing Flows: Motivation

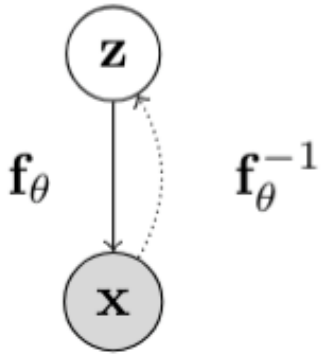
$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



- Model families:
  - Autoregressive Models: $p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i|\mathbf{x}_{<i})$
  - Variational Autoencoders: $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}$
- Autoregressive models provide tractable likelihoods but no direct mechanism for learning features
- Variational autoencoders can learn feature representations (via latent variables $\mathbf{z}$) but have intractable marginal likelihoods
- **Key question**: Can we design a latent variable model with tractable likelihoods? Yes! Use normalizing flows.

Normalizing Flow Models: Definition

In a **normalizing flow model**, the mapping between Z and X, given by $\mathbf{f}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$, is deterministic and invertible such that $X = \mathbf{f}_\theta(Z)$ and $Z = \mathbf{f}_\theta^{-1}(X)$



- We want to learn $p_X(\mathbf{x}; \theta)$ using the principle of maximum likelihood.
- Using change of variables, the marginal likelihood $p(\mathbf{x})$ is given by

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

Normalizing Flow Models: Constricuting f.

We need to construct a density transformation that is:

- Invertible, so that we can apply the change of variables formula.
- Expressive, so that we can learn complex distributions.
- Computationally tractable, so that we can optimize and evaluate it.
  - Computing likelihoods requires evaluting the determinant for an $n \times n$ Jacobian matrix, an expensive $O(n^3)$ operation!

Strategies:

1. Apply sequence of M **simple** invertible transformations with $\mathbf{x} \triangleq \mathbf{z}_M$

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

   Determininant of composition equals product of determinants:

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

2. Choose transformations for which the Jacobian matrix has special structure (e.g., it is diagonal).

Triangular Jacobian

Suppose we have the following vector-valued invertible mapping f:

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

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

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**.

Lecture Outline

1. Flows With Triangular Jacobians
   - Nonlinear Independent Components Estimation (Dinh et al. 2014)
   - Real NVP (Dinh et al. 2017)
2. Autoregressive Flows
   - Masked Autoregressive Flow (Papamakarios et al., 2017)
   - Inverse Autoregressive Flow (Kingma et al., 2016)
3. Probability Distillation and Parallel Wavenet

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: $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d})$

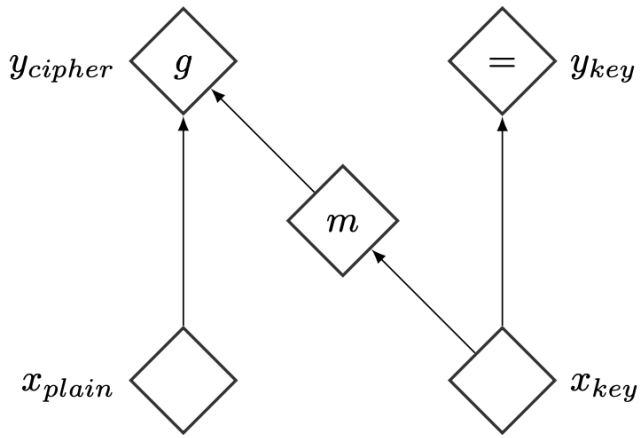


Figure 2: Computational graph of a coupling layer

- $m_\theta(\cdot)$ 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:

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

$$\det(J) = 1$$

- Observe that:
  - 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:

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

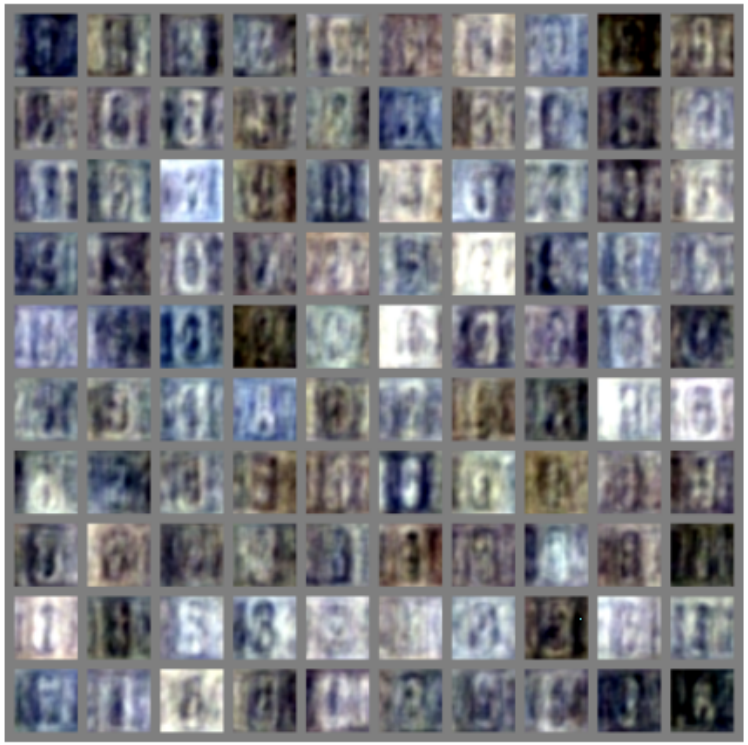
$$\det(J) = \prod_{i=1}^{n} s_i$$

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

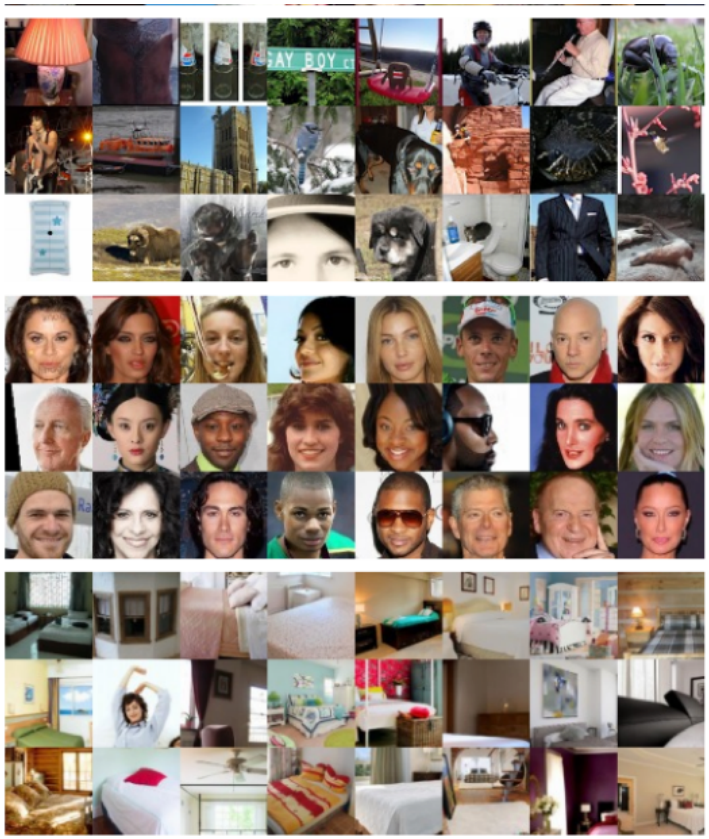
- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\theta(\mathbf{z}_{1:d})) + \mu_\theta(\mathbf{z}_{1:d})$
  - $\mu_\theta(\cdot)$ and $\alpha_\theta(\cdot)$ are both neural 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)
  - $\mathbf{z}_{d+1:n} = (\mathbf{x}_{d+1:n} - \mu_\theta(\mathbf{x}_{1:d})) \odot (\exp(-\alpha_\theta(\mathbf{x}_{1:d})))$
- Jacobian of forward mapping:

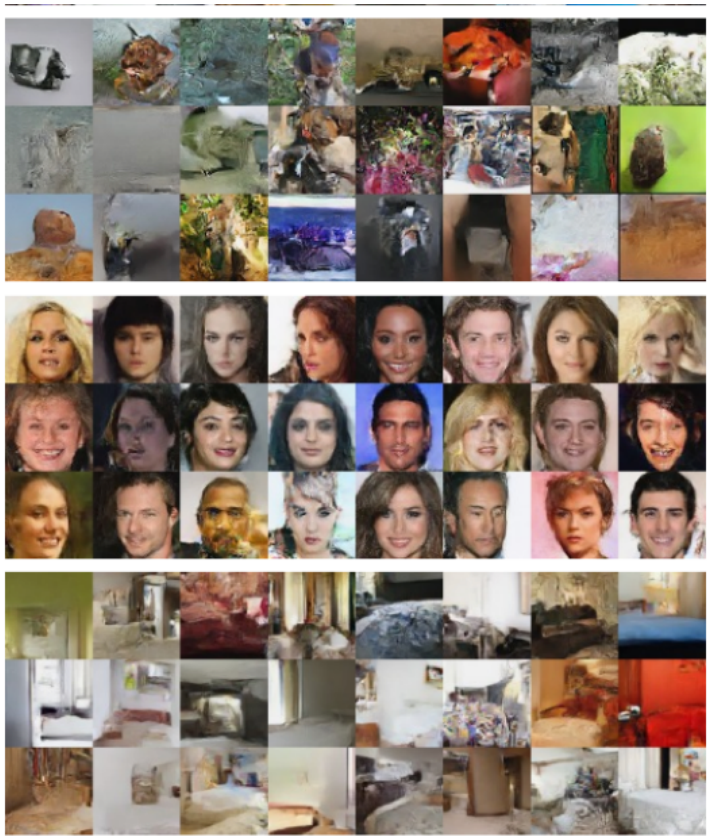
$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \text{diag}(\exp(\alpha_\theta(\mathbf{z}_{1:d}))) \end{pmatrix}$$

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

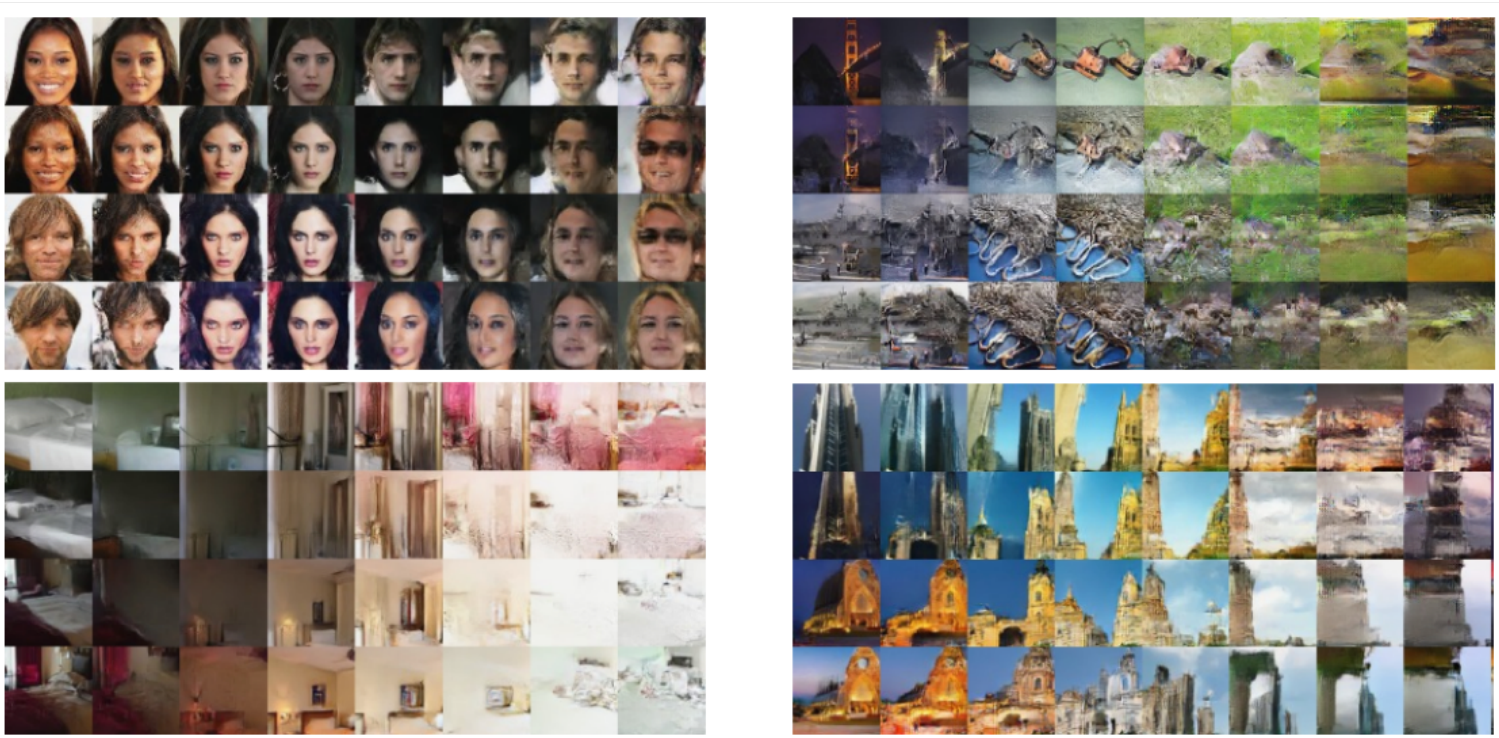
- **Non-volume preserving transformation** in general since determinant can be less than or greater than 1

Samples Generated via Real-NVP





Latent Space Interpolations via Real-NVP



Using four validation examples $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \mathbf{z}^{(3)}, \mathbf{z}^{(4)}$, define interpolated $\mathbf{z}$ as:

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

with manifold parameterized by ϕ and ϕ'.

Lecture Outline

1. Flows With Triangular Jacobians
   - Nonlinear Independent Components Estimation (Dinh et al. 2014)
   - Real NVP (Dinh et al. 2017)
2. Autoregressive Flows
   - Masked Autoregressive Flow (Papamakarios et al., 2017)
   - Inverse Autoregressive Flow (Kingma et al., 2016)
3. Probability Distillation and Parallel Wavenet

Autoregressive Models as Flow Models

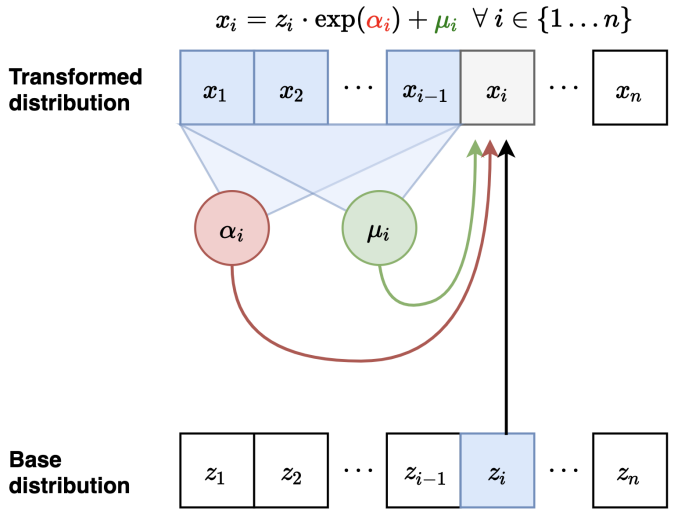
- Consider a Gaussian autoregressive model:

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

  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 $i = 1$.
- 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 (MAF)

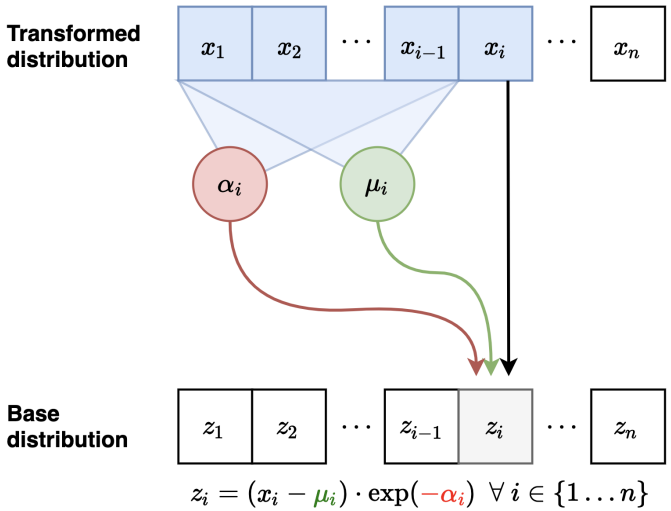
A Masked Autoregressive Flow (MAF) is a normalizing flow model in which the transformation $\mathbf{f} : Z \to X$ implements this process:



- 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 $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
- Computing the forward mapping (i.e. sampling) is sequential and slow: $O(n)$ time (it's an autoregressive model)

Figure adapted from Eric Jang's blog

Masked Autoregressive Flow (MAF)

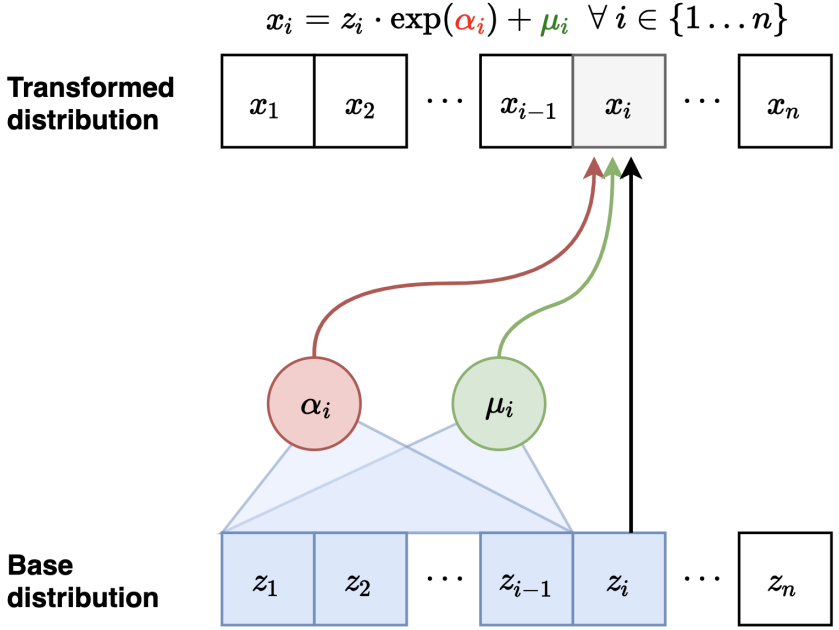


- Inverse mapping from $\mathbf{x} \mapsto \mathbf{z}$:
  - Compute all μ_i, α_i (can be done in parallel)
  - Let $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$ (scale and shift)
  - Let $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$
  - Let $z_3 = (x_3 - \mu_3)/\exp(\alpha_3)$ ...
- Jacobian is lower diagonal; determinant is computed efficiently
- Inverse mapping (i.e., likelihood evaluation) is easy and parallelizable

Figure adapted from Eric Jang's blog

Inverse Autoregressive Flow (IAF)

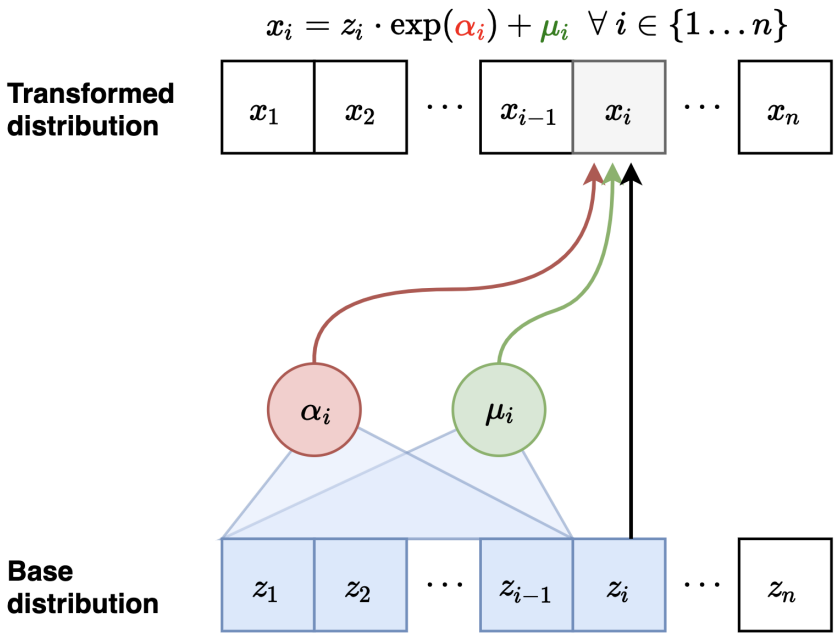
An Inverse Autoregressive Flow (IAF) is a model in which the transformation $\mathbf{f} : X \to Z$ is **autoregressive in** z (while an MAF is autoregressive in x):



- 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 $x_2 = \exp(\alpha_2) z_2 + \mu_2$ ...

Figure adapted from Eric Jang's blog

Inverse Autoregressive Flow (IAF)



- 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 $z_1, z_2, \dots, z_n$)

Figure adapted from Eric Jang's blog

IAF is Transpose of MAF

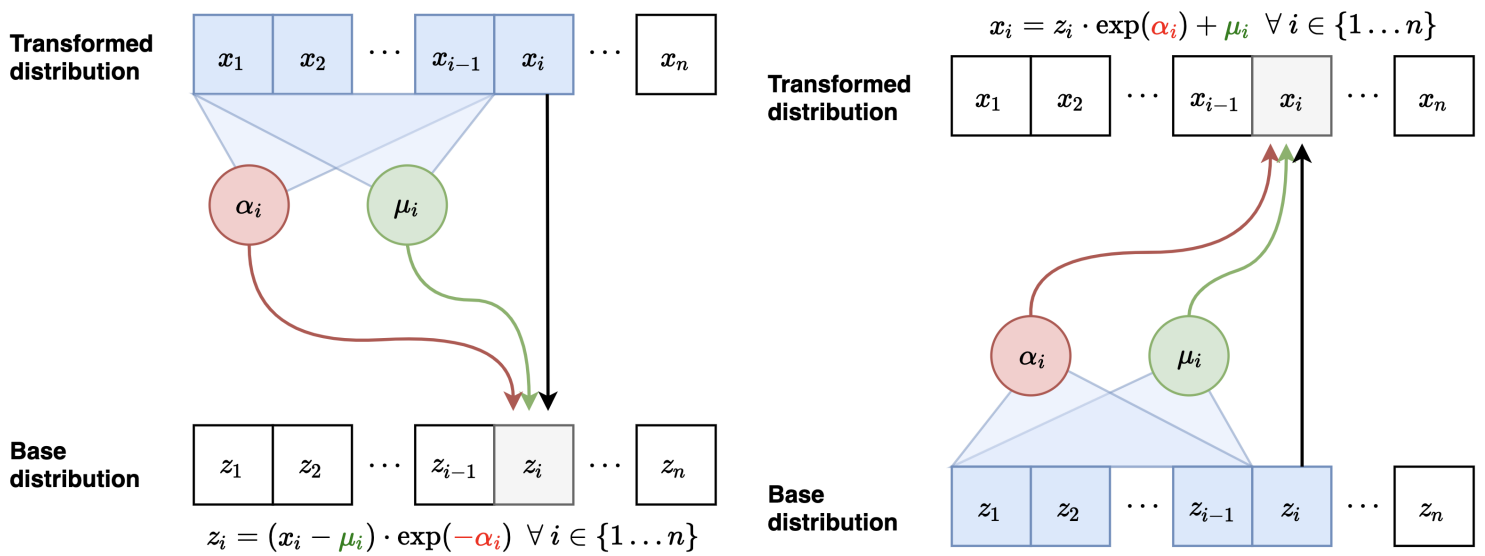


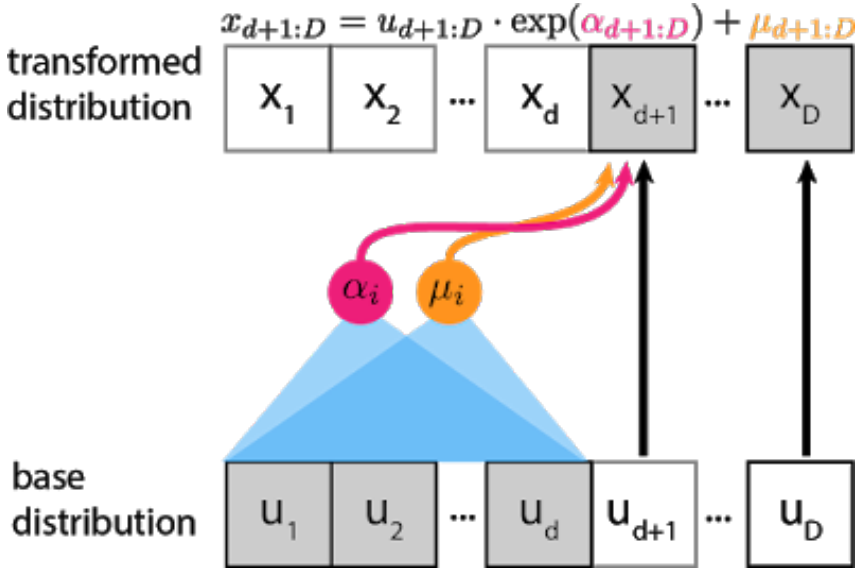
Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

- Interchanging **z** and **x** in the inverse transformation of MAF gives the forward transformation of IAF
- Similarly, swapping **z** and **x** in the forward transform of MAF yields the inverse transform of IAF

Figure adapted from Eric Jang's blog

NICE and Real NVP as IAF

- Note that NICE and Real NVP are special cases of the IAF framework.



- But scale and shift statistics can be computed in a single pass because they are a function of the partition that is not being transformed.
- Therefore sampling and posterior inference is fast.

Figure from Eric Jang's blog

IAF vs. MAF

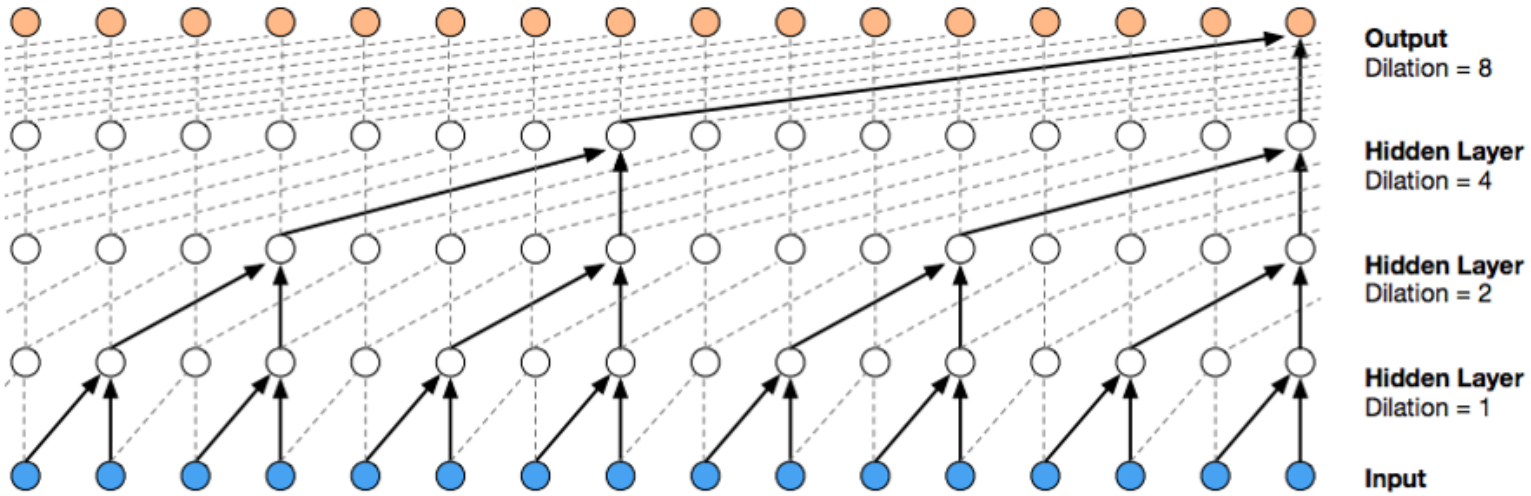
- Both IAF and MAF are expressive models, with different computational tradeoffs
  - MAF: Fast likelihood evaluation, slow sampling
  - IAF: Fast sampling, slow likelihood evaluation
- MAF more suited for training based on MLE, density estimation
- IAF more suited for real-time generation
- Can we get the best of both worlds?

Lecture Outline

1. Flows With Triangular Jacobians
   - Nonlinear Independent Components Estimation (Dinh et al. 2014)
   - Real NVP (Dinh et al. 2017)
2. Autoregressive Flows
   - Masked Autoregressive Flow (Papamakarios et al., 2017)
   - Inverse Autoregressive Flow (Kingma et al., 2016)
3. Probability Distillation and Parallel Wavenet

Recall: WaveNet (van den Oord et al., 2016)

WaveNet is a state of the art generative model for audio.



WaveNet is an autoregressive model, therefore generating from this model is slow, especially for audio sampled at tens of thousands of Hz.

Model Distillation

In **probability density distillation**, a student distribution is trained to minimize the KL divergence between the student (s) and the teacher (t)

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

- Evaluating and optimizing Monte Carlo estimates of this objective requires:
  - Samples $\mathbf{x}$ from student model (e.g., IAF)
  - Density of $\mathbf{x}$ assigned by student model
  - Density of $\mathbf{x}$ assigned by teacher model (e.g., MAF)

Accelerating Wavenet

Fast WaveNet: Two-part training with a teacher and student model

1. Train a WaveNet teacher model efficiently via MLE
2. Fit a student model parameterized by IAF
   - Student IAF model allows for efficient sampling.
   - Student IAF can also evaluate density of its samples! (via caching)
   - Teacher can also quickly evaluate the density of samples from the student, making distillation possible
3. At inference-time we us the student IAF model to generate audio. This improves WaveNet speed by 1000x!

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
  - The 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 auto-regressive structure that influences speed of forward and inverse sampling.