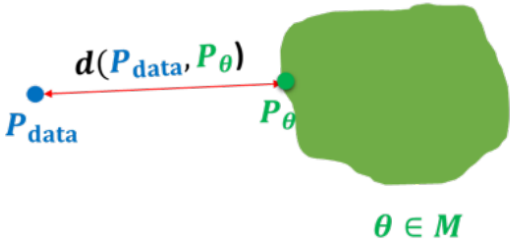
# CENG 796
# Deep Generative Models

## Normalizing Flows

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

# Recap of likelihood-based learning so far:

$x_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$




- 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!

Slide by Stefano Ermon, Aditya Grover

# Simple Prior to Complex Data Distributions

- Desirable properties of any model distribution:
  - Analytic density
  - Easy-to-sample
- Many simple distributions satisfy the above properties e.g., Gaussian, uniform distributions
- Unfortunately, data distributions could be much more complex (multi-modal)
- **Key idea**: Map simple distributions (easy to sample and evaluate densities) to complex distributions (learned via data) using **change of variables**.

# Change of Variables formula

- Let $Z$ be a uniform random variable $\mathcal{U}[0, 2]$ with density $p_Z$. What is $p_Z(1)$? $\frac{1}{2}$
- Let $X = 4Z$, and let $p_X$ be its density. What is $p_X(4)$?
- $p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1) = p_Z(1) = 1/2$ **No**
- Clearly, $X$ is uniform in $[0, 8]$, so $p_X(4) = 1/8$

# Change of Variables formula

- **Change of variables (1D case)**: If $X = f(Z)$ and $f(\cdot)$ 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 $Z \sim \mathcal{U}[0, 2]$, what is $p_X(4)$?
- Note that $h(X) = X/4$
- $p_X(4) = p_Z(1)h'(4) = 1/2 \times 1/4 = 1/8$

# Geometry: Determinants and volumes

- 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?
- Geometrically, the matrix $A$ maps the unit hypercube $[0, 1]^n$ to a parallelotope
- Hypercube and parallelotope are generalizations of square/cube and parallelogram/parallelopiped to higher dimensions

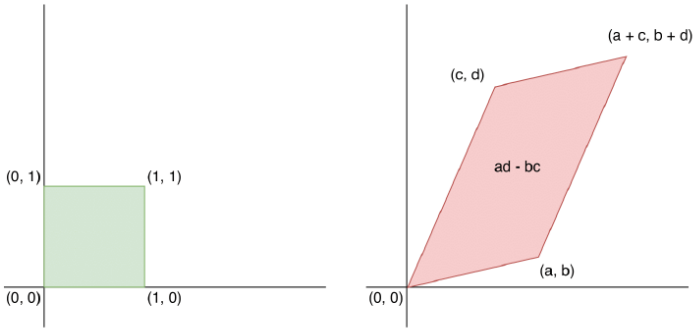



Figure: The matrix $A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ maps a unit square to a parallelogram

# Geometry: Determinants and volumes

- 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

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

# Generalized change of variables

- For linear transformations specified via $A$, change in volume is given by the determinant of $A$
- For non-linear transformations $\mathbf{f}(\cdot)$, the *linearized* change in volume is given by the determinant of the Jacobian of $\mathbf{f}(\cdot)$.
- **Change of variables (General case)**: The mapping between $Z$ and $X$, given by $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$, is invertible such that $X = \mathbf{f}(Z)$ and $Z = \mathbf{f}^{-1}(X)$.

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

- Note 1: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension. For example, if $\mathbf{x} \in \mathbb{R}^n$ then $\mathbf{z} \in \mathbb{R}^n$
- Note 2: For any invertible matrix $A$, $det(A^{-1}) = det(A)^{-1}$

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

Slide by Stefano Ermon, Aditya Grover

# Two Dimensional Example

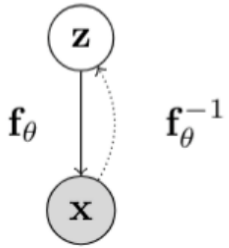
- Let $Z_1$ and $Z_2$ be continuous random variables with joint density $p_{Z_1,Z_2}$.
- Let $u = (u_1, u_2)$ be a transformation
- Let $v = (v_1, v_2)$ be the inverse transformation
- Let $X_1 = u_1(Z_1, Z_2)$ and $X_2 = u_2(Z_1, Z_2)$ Then, $Z_1 = v_1(X_1, X_2)$ and $Z_2 = v_2(X_1, X_2)$

$$
\begin{aligned}
& p_{X_1,X_2}(x_1, x_2) \\
&= p_{Z_1,Z_2}(v_1(x_1, x_2), v_2(x_1, x_2)) \left| \det \begin{pmatrix} \frac{\partial v_1(x_1,x_2)}{\partial x_1} & \frac{\partial v_1(x_1,x_2)}{\partial x_2} \\ \frac{\partial v_2(x_1,x_2)}{\partial x_1} & \frac{\partial v_2(x_1,x_2)}{\partial x_2} \end{pmatrix} \right| \text{(inverse)} \\
&= p_{Z_1,Z_2}(z_1, z_2) \left| \det \begin{pmatrix} \frac{\partial u_1(z_1,z_2)}{\partial z_1} & \frac{\partial u_1(z_1,z_2)}{\partial z_2} \\ \frac{\partial u_2(z_1,z_2)}{\partial z_1} & \frac{\partial u_2(z_1,z_2)}{\partial z_2} \end{pmatrix} \right|^{-1} \text{(forward)}
\end{aligned}
$$

Slide by Stefano Ermon, Aditya Grover

# Normalizing flow models

- Consider a directed, latent-variable model over observed variables $X$ and latent variables $Z$
- 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 $Z = \mathbf{f}_\theta^{-1}(X)$




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

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

- Note: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension.

Slide by Stefano Ermon, Aditya Grover

# A Flow of Transformations

**Normalizing:** Change of variables gives a normalized density after applying an invertible transformation

**Flow:** Invertible transformations can be composed with each other

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

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

Slide by Stefano Ermon, Aditya Grover

# Planar flows (Rezende & Mohamed, 2016)

- Planar flow. Invertible transformation

$$\mathbf{x} = \mathbf{f}_\theta(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$$

  parameterized by $\theta = (\mathbf{w}, \mathbf{u}, b)$ where $h(\cdot)$ is a non-linearity
- Absolute value of the determinant of the Jacobian is given by

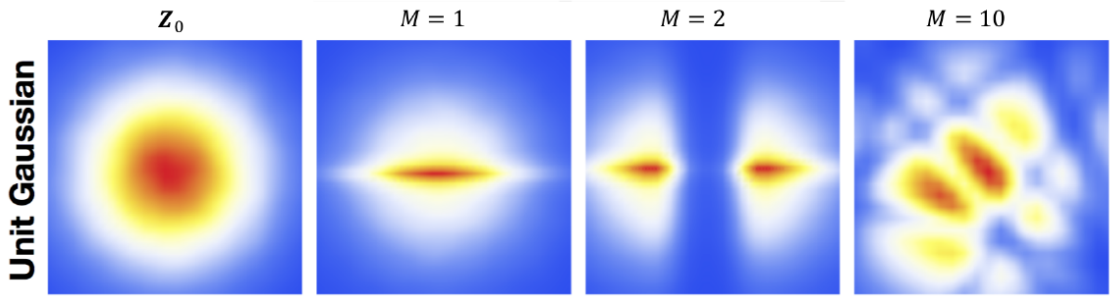
$$\begin{aligned}\left|\det \frac{\partial \mathbf{f}_\theta(\mathbf{z})}{\partial \mathbf{z}}\right| &= \left|\det(I + h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{u}\mathbf{w}^T)\right| \\ &= \left|1 + h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{u}^T\mathbf{w}\right| \\ &\text{(matrix determinant lemma)}\end{aligned}$$

- Need to restrict parameters and non-linearity for the mapping to be invertible. For example, $h = tanh()$ and $h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{u}^T\mathbf{w} \geq -1$

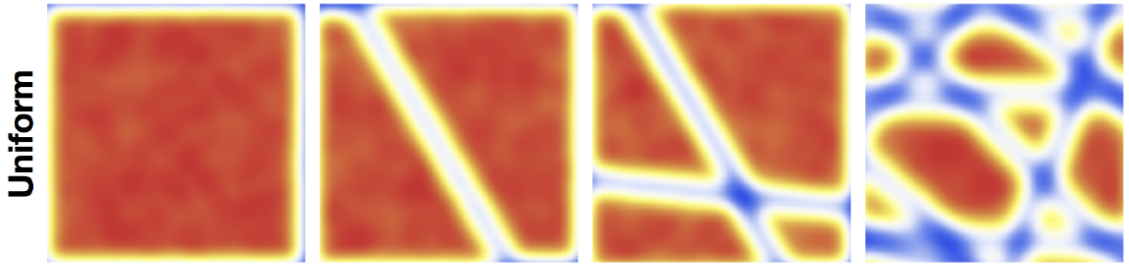
Slide by Stefano Ermon, Aditya Grover

# Planar flows (Rezende & Mohamed, 2016)

- Base distribution: Gaussian




- Base distribution: Uniform




- 10 planar transformations can transform simple distributions into a more complex one

Slide by Stefano Ermon, Aditya Grover

# Learning and Inference

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

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

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

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

Slide by Stefano Ermon, Aditya Grover

# Desiderata for flow models

- Simple prior $p_Z(\mathbf{z})$ that allows for efficient sampling and tractable likelihood evaluation. E.g., isotropic Gaussian
- 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 can be computed in **linear time**. Similarly, the Jacobian is upper triangular if $x_i$ only depends on $\mathbf{z}_{\geq i}$

**Next lecture:** Designing invertible transformations!

Slide by Stefano Ermon, Aditya Grover

# Part II

# Recap of normalizing flow models

So far

- Transform simple to complex distributions via sequence of invertible transformations
- Directed latent variable models with marginal likelihood given by the change of variables formula
- Triangular Jacobian permits efficient evaluation of log-likelihoods

Plan for today

- Invertible transformations with diagonal Jacobians (NICE, Real-NVP)
- Autoregressive Models as Normalizing Flow Models
- Case Study: Probability density distillation for efficient learning and inference in Parallel Wavenet

# Designing invertible transformations

- NICE or Nonlinear Independent Components Estimation (Dinh et al., 2014) composes two kinds of invertible transformations: additive coupling layers and rescaling layers
- Real-NVP (Dinh et al., 2017)
- Inverse Autoregressive Flow (Kingma et al., 2016)
- Masked Autoregressive Flow (Papamakarios et al., 2017)

# NICE - Additive coupling layers

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

- 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 $\theta$, $d$ input units, and $n - d$ output units)
- Inverse mapping $\mathbf{x} \mapsto \mathbf{z}$:
  - $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$ (identity transformation)
  - $\mathbf{z}_{d+1:n} = \mathbf{x}_{d+1:n} - m_\theta(\mathbf{x}_{1:d})$
- Jacobian of forward mapping:

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

$$\det(J) = 1$$

- **Volume preserving transformation** since determinant is 1.

# NICE - Rescaling layers

- Additive coupling layers are composed together (with arbitrary partitions of variables in each layer)
- Final layer of NICE applies a rescaling transformation
- 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:

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

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

Slide by Stefano Ermon, Aditya Grover

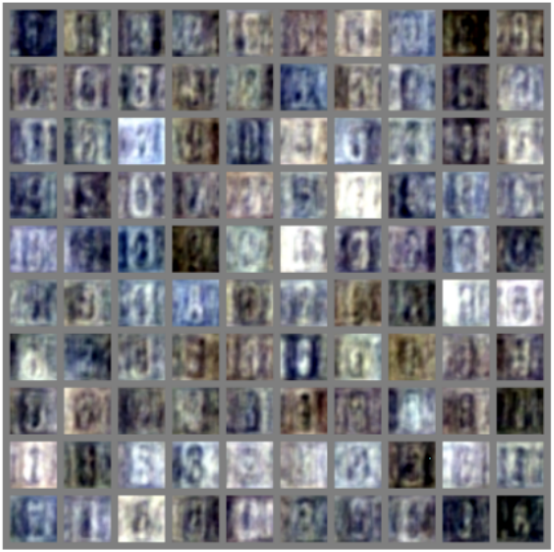
# Samples generated via NICE

(a) Model trained on MNIST

(b) Model trained on TFD

Slide by Stefano Ermon, Aditya Grover

# Samples generated via NICE



(c) Model trained on SVHN

(d) Model trained on CIFAR-10

Slide by Stefano Ermon, Aditya Grover

# 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 $\theta$, $d$ 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:

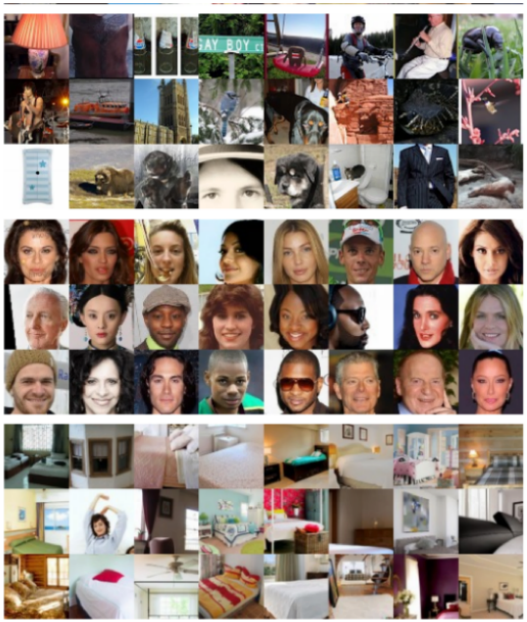
$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \text{diag}(\exp(\alpha_\theta(\mathbf{z}_{1:d}))) \end{pmatrix}$$

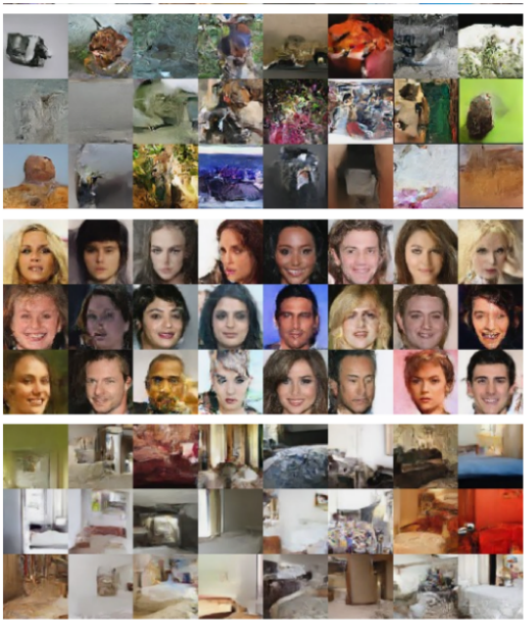
$$\det(J) = \prod_{i=d+1}^{n} \exp(\alpha_\theta(\mathbf{z}_{1:d})_i) = \exp\left(\sum_{i=d+1}^{n} \alpha_\theta(\mathbf{z}_{1:d})_i\right)$$

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

Slide by Stefano Ermon, Aditya Grover

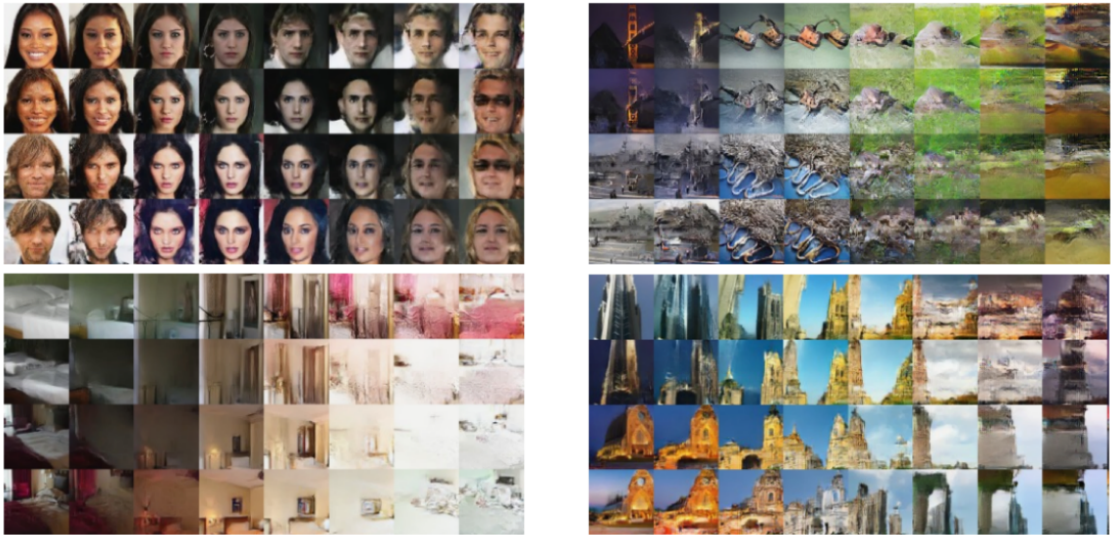
# Samples generated via Real-NVP





Slide by Stefano Ermon, Aditya Grover

# Latent space interpolations via Real-NVP



Using with 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 $\phi$ and $\phi'$.

Slide by Stefano Ermon, Aditya Grover

# Autoregressive models as flow models

- Consider a Gausian autoregressive model:

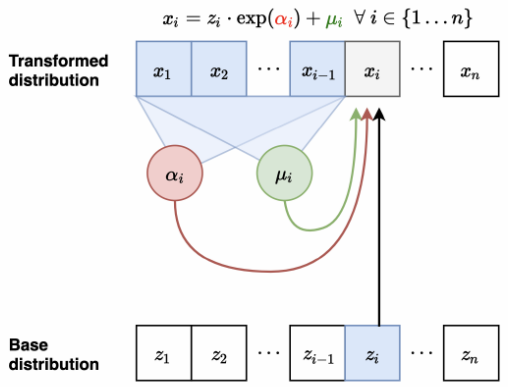
$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|\mathbf{x}_{<i})$$

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

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

- 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 $x_3 = \exp(\alpha_3)z_3 + \mu_3$. ...
- **Flow interpretation:** transforms samples from the standard Gaussian $(z_1, z_2, \ldots, z_n)$ to those generated from the model $(x_1, x_2, \ldots, x_n)$ via invertible transformations (parameterized by $\mu_i(\cdot), \alpha_i(\cdot)$)

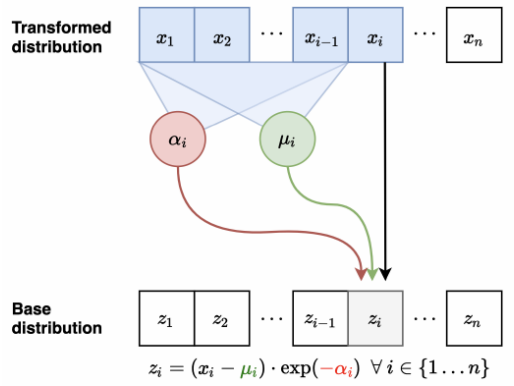
# Masked Autoregressive Flow (MAF)




- 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): $O(n)$ time

Figure adapted from Eric Jang's blog

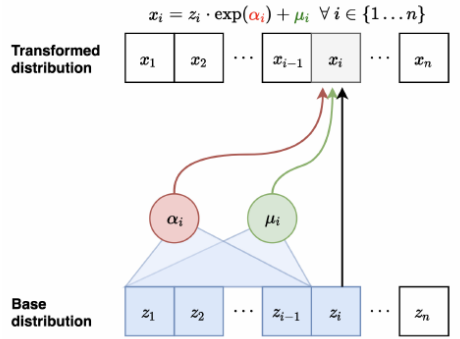
# Masked Autoregressive Flow (MAF)




- Inverse mapping from $\mathbf{x} \mapsto \mathbf{z}$:
  - Compute all $\mu_i, \alpha_i$ (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 $z_3 = (x_3 - \mu_3) / \exp(\alpha_3)$ ...
- Jacobian is lower diagonal, hence determinant can be computed efficiently
- Likelihood evaluation is easy and parallelizable (like MADE)

Figure adapted from Eric Jang's blog

# 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 $\mu_i, \alpha_i$ (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 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 $z_1, z_2, \dots, z_n$)

Figure adapted from Eric Jang's blog

Slide by Stefano Ermon, Aditya Grover

# IAF is inverse of MAF

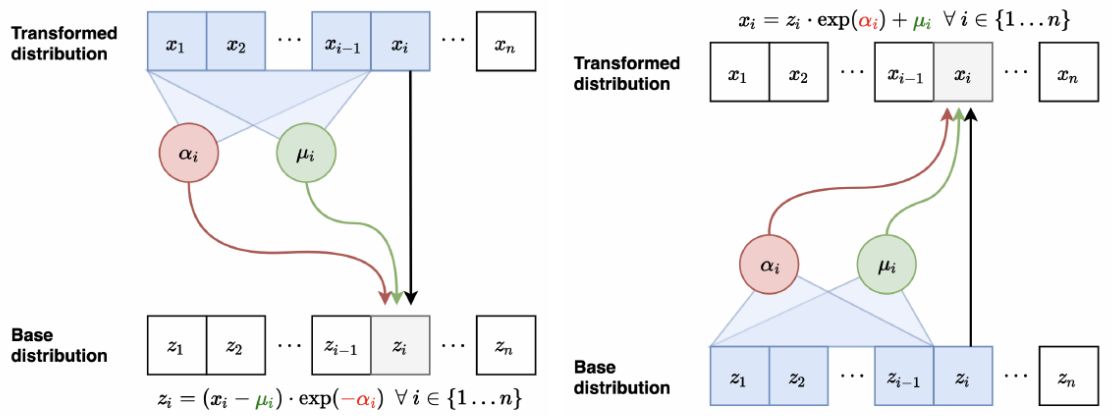



Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

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

Figure adapted from Eric Jang's blog

# IAF vs. MAF

- 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?

# Parallel Wavenet

- Two part training with a teacher and student model
- Teacher is parameterized by MAF. Teacher can be efficiently trained via MLE
- Once teacher is trained, initialize a student model parameterized by IAF. Student model cannot efficiently evaluate density for external datapoints but allows for efficient sampling
- **Key observation**: IAF can also efficiently evaluate densities of its own generations (via caching the noise variates $z_1, z_2, \ldots, z_n$)

# Parallel Wavenet

- **Probability density distillation**: Student distribution is trained to minimize the KL divergence between student ($s$) and 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 (IAF)
  - Density of $\mathbf{x}$ assigned by student model
  - Density of $\mathbf{x}$ assigned by teacher model (MAF)
- All operations above can be implemented efficiently

Slide by Stefano Ermon, Aditya Grover

# Parallel Wavenet: Overall algorithm

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

Slide by Stefano Ermon, Aditya Grover

# Summary of Normalizing Flow Models

- Transform simple distributions into more complex distributions via change of variables
- Jacobian of transformations should have tractable determinant for efficient learning and density estimation
- Computational tradeoffs in evaluating forward and inverse transformations