Modern Normalizing Flow Models

Volodymyr Kuleshov

Cornell Tech

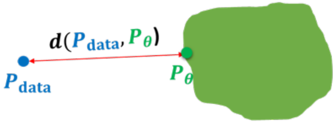
Lecture 8

Announcements

- Assignment 1 is due at midnight today!
  - If submitting late, please mark it as such.
- Submit Assignment 1 via Gradescope. The code is M45WYY.
  - Sign up for Gradescope.com with the code
  - Submit your assignment as a photo/pdf
- Assignment 2 will be out today and due in two weeks.
- Presentation slots are almost filled, but I can make space.
- Project instructions are on Piazza

Normalizing Flows: Motivation

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



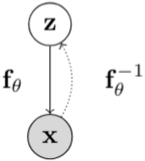
$\theta \in M$

Model family

- 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 Flows: Properties

1. Normalizing Flows Pros:
   - Exact marginal likelihood $p(x)$ is tractable to compute and optimize
   - Exact posterior inference $p(z|x)$ is tractable
2. 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.

Normalizing Flow Models: Constricting 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 complex tranformations so that the resulting Jacobian matrix has special structure.

Lecture Outline

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

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

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

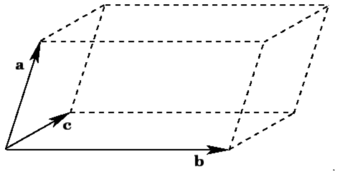
has lower triangular structure. Determinant can be computed in **linear time**.

Triangular Jacobian Can Be Computed in Linear Time

One intuition: Consider a square matrix M with square blocks A, B, C, D, and apply the following structure recursively

$$\det(M) = \det \begin{pmatrix} A & C \\ B & D \end{pmatrix} = AD - BC$$

Another intuition: Area of 2d parallelogram.



Strategy:

- Design transformation functions $t(z)$ such that

$$x_i = t_i(z_{<i}).$$

- Hint: this is starting to look like a normalizing flow!

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

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 $\mathbf{z} \mapsto \mathbf{x}$: defining $x \leftarrow \mathbf{f}(z)$
  - The first set of variables stays the same: $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
  - The second variables undergo an affine transformation: $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d})$

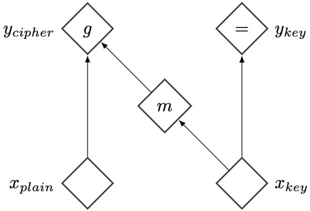


Figure 2: Computational graph of a coupling layer

- $m_\theta(\cdot)$ 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:

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

$$\det(J) = 1$$

- **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

- Additive coupling layers are composed together (with arbitrary partitions of variables in each layer)
- Final layer of NICE applies a rescaling transformation
- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:

$$x_i = s_i z_i$$

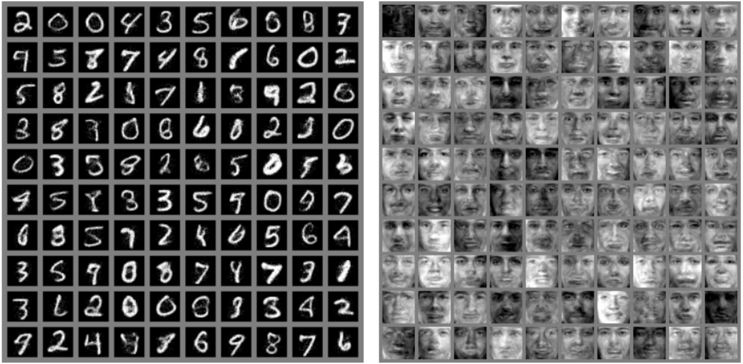
  where $s_i > 0$ is the scaling factor for the i-th dimension.
- Inverse mapping $\mathbf{x} \mapsto \mathbf{z}$:

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

- Jacobian of forward mapping:

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

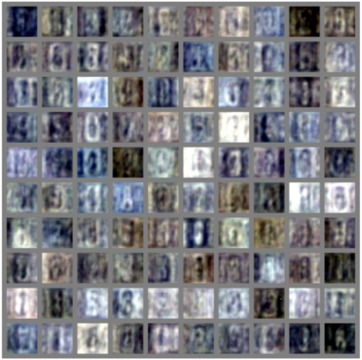
$$\det(J) = \prod_{i=1}^{n} s_i$$



(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

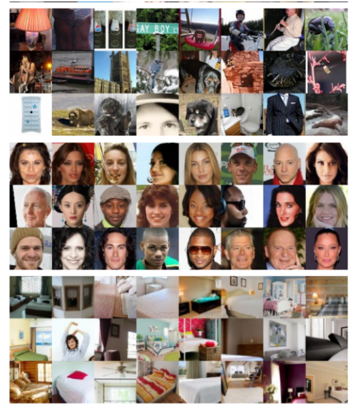
- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\theta(\mathbf{z}_{1:d})) + \mu_\theta(\mathbf{z}_{1:d})$
  - $\mu_\theta(\cdot)$ and $\alpha_\theta(\cdot)$ are both neural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

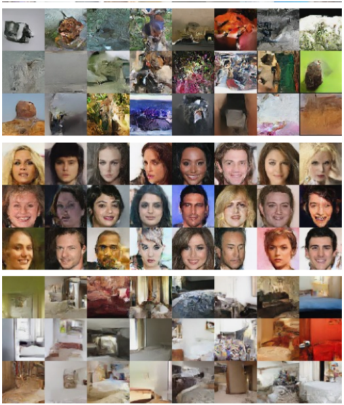
$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \operatorname{diag}(\exp(\alpha_\theta(\mathbf{z}_{1:d}))) \end{pmatrix}$$

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

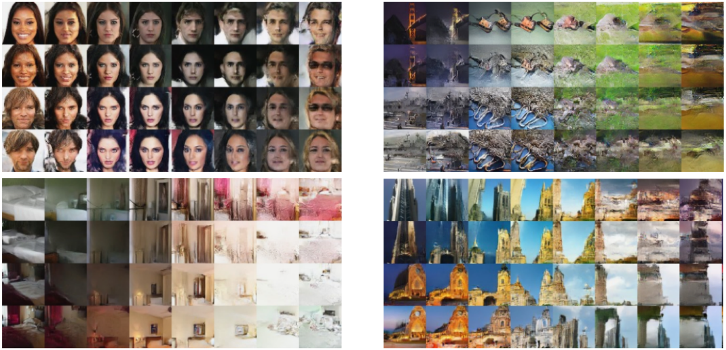
- **Non-volume preserving transformation** in general since determinant can be less than or greater than 1

Samples generated via Real-NVP





Latent space interpolations via Real-NVP



Using 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 ϕ and ϕ'.

Lecture Outline

1. Recap and Motivation for Normalizing Flows
   - Triangular Jacobians
2. Nonlinear Independent Components Estimation (Dinh et al. 2014)
3. Real NVP (Dinh et al. 2017)
4. Masked Autoregressive Flow (Papamakarios et al., 2017)
5. Inverse Autoregressive Flow (Kingma et al., 2016)
6. 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 $p(x_i \mid \mathbf{x}_{<i}) = \mathcal{N}(\mu_i(x_1, \cdots, x_{i-1}), \exp(\alpha_i(x_1, \cdots, x_{i-1}))^2)$.

- 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 $x_3 = \exp(\alpha_3)z_3 + \mu_3$. ...

Autoregressive models as flow models

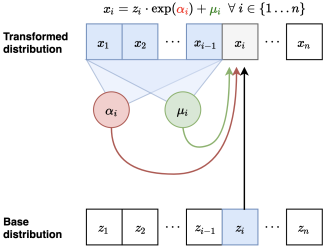
- Consider a Gaussian autoregressive model:

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

  such that $p(x_i \mid \mathbf{x}_{<i}) = \mathcal{N}(\mu_i(x_1, \cdots, x_{i-1}), \exp(\alpha_i(x_1, \cdots, x_{i-1}))^2)$.
- 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 $\mu_2(x_1), \alpha_2(x_1)$ etc.
- **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 $\mu_i(\cdot), \alpha_i(\cdot)$)
  - Can be used as flow layers
  - Independent of ordering!
  - Can be composed

Masked Autoregressive Flow (MAF)

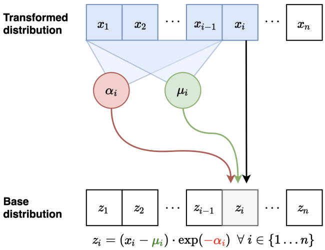
Masked Autoregressive Flow (MAF) is a bijective normalizing flow transformation $\mathbf{f} : X \to Z$ that implements this intuition:



- 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

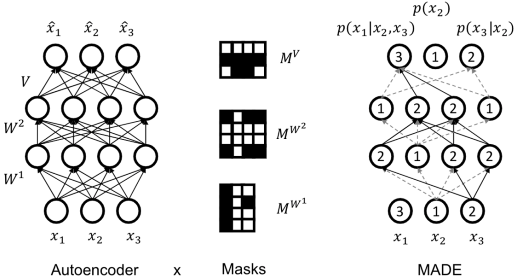
Masked Autoregressive Flow (MAF)



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

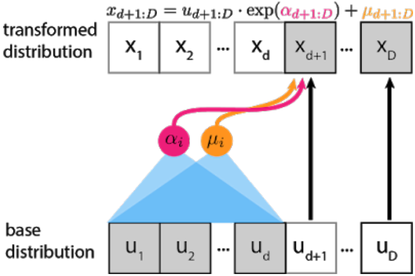
MADE: Masked Autoencoder for Distribution Estimation



1. **Challenge**: Compute the μ_i in parallel and reuse weights.
2. **Solution**: use masks to disallow certain paths (Germain et al., 2015). Suppose ordering is x_2, x_3, x_1.
   1. The unit producing the parameters for $p(x_2)$ is not allowed to depend on any input. Unit for $p(x_3|x_2)$ only on x_2. And so on...
   2. For each unit in a hidden layer, pick an integer i in $[1, n-1]$. Unit made to depend on the first i inputs in ordering.
   3. Add mask to preserve this invariant: connect to all units in previous layer with smaller or equal assigned number (strictly $<$ in final layer)

NICE and Real NVP as MAF

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

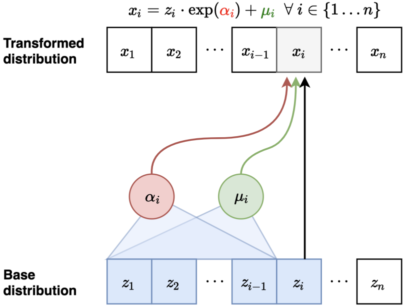


- But scale and shift statistics can be computed in a single pass.
- Therefore sampling and posterior inference is fast and a MADE-style approach is not needed.

Figure from Eric Jang's blog

Inverse Autoregressive Flow (IAF)

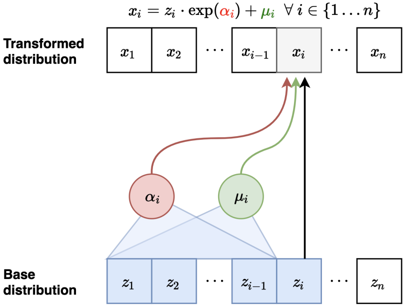
Inverse Autoregressive Flow (IAF) is a bijective normalizing flow transformation $\mathbf{f} : X \rightarrow Z$ that implements the opposite sampling approach:



- Forward mapping from $\mathbf{z} \mapsto \mathbf{x}$ (parallel):
  - Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \cdots, n$
  - Compute all $\mu_i(z_{<i}), \alpha_i(z_{<i})$ (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 $\mu_3(z_1, z_2), \alpha_3(z_1, z_2)$
- 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

IAF is inverse of MAF

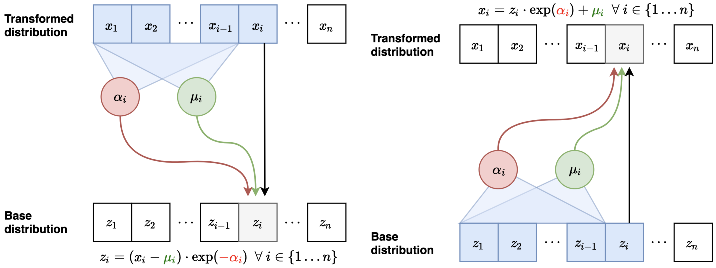


Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

- 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

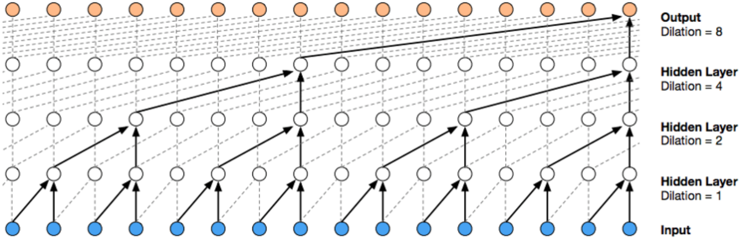
Figure adapted from Eric Jang's blog

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?

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

State of the art model for speech:



Dilated convolutions increase the receptive field: kernel only touches the signal at every 2^d entries.

Accelerating Wavenet

- **Challenge:** How to make sampling fast?
- **Solution:** 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 $z_1, z_2, \ldots, z_n$)

Model Distillation

- **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

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!

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.