Normalizing Flow Models

Stefano Ermon, Aditya Grover

Stanford University

Lecture 8

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 θ, 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:

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

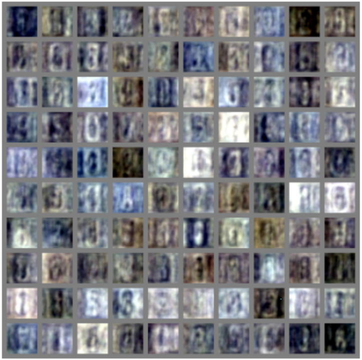
$$\det(J) = \prod_{i=1}^{n} s_i$$

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

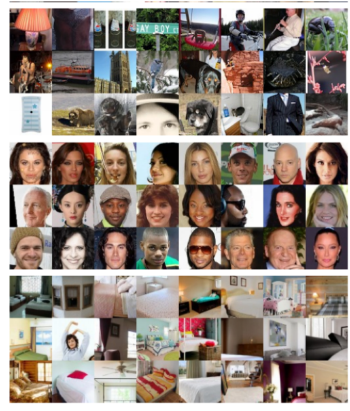
- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\theta(\mathbf{z}_{1:d})) + \mu_\theta(\mathbf{z}_{1:d})$
  - $\mu_\theta(\cdot)$ and $\alpha_\theta(\cdot)$ are both neural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

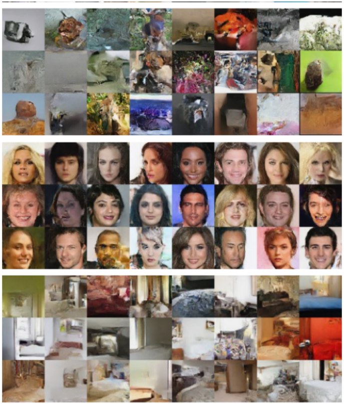
$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \mathrm{diag}(\exp(\alpha_\theta(\mathbf{z}_{1:d}))) \end{pmatrix}$$

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

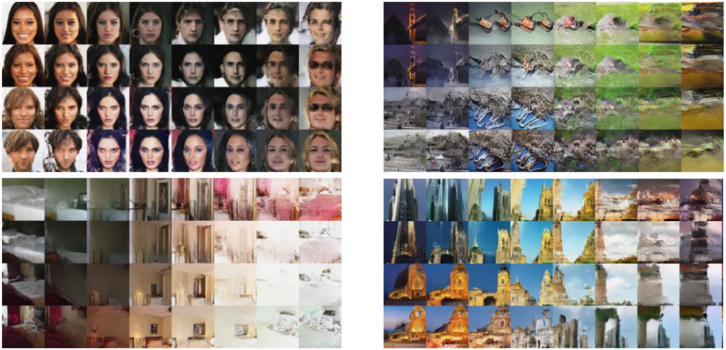
- **Non-volume preserving transformation** in general since determinant can be less than or greater than 1

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'.

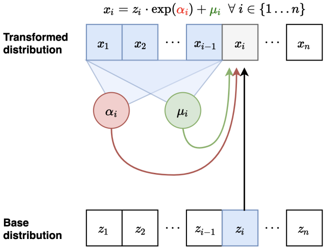
Autoregressive models as flow models

- 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 $\mu_i(\cdot), \alpha_i(\cdot)$)

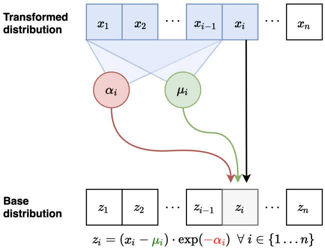
Masked Autoregressive Flow (MAF)



- 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

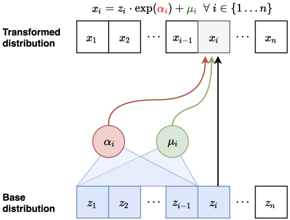
Masked Autoregressive Flow (MAF)



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

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 μ_i, α_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 $z_1, z_2, \ldots, z_n$)

Figure adapted from Eric Jang's blog

IAF is inverse of MAF

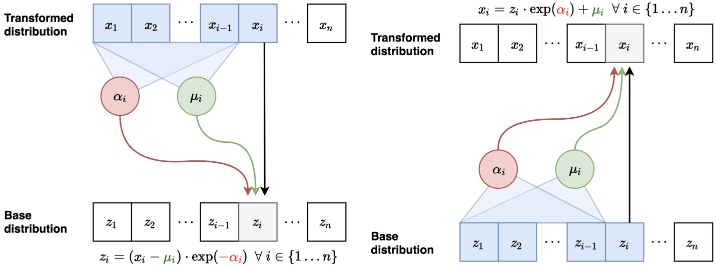


Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

- 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?

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

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
- Jacobian of transformations should have tractable determinant for efficient learning and density estimation
- Computational tradeoffs in evaluating forward and inverse transformations