Normalizing Flow Models
Part 2

Dmytro Urukov
24 Mar 2020
Recap: Normalizing Flows

Take a random variable $\mathbf{z}$ with distribution $q(\mathbf{z})$, apply some invertible mapping: $\mathbf{z}' = f(\mathbf{z})$
Recap: Change of variables rule

\[ p_X(x) = p_Z(z) \left| \frac{\partial z}{\partial x} \right| \]

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

\[ \max_{\theta} \log p_X(D; \theta) = \sum_{x \in D} \log p_Z (f^{-1}_\theta(x)) + \log \left| \det \left( \frac{\partial f^{-1}_\theta(x)}{\partial x} \right) \right| \]
Recap: triangular matrix

\[ x_1 = T_1(z_1) \]
\[ x_2 = T_2(z_1, z_2) \]
\[ x_3 = T_3(z_1, z_2, z_3) \]
\[ \vdots \]
\[ x_d = T_d(z_1, z_2, z_3, \ldots, z_d) \]

\[ T : \mathbb{R}^d \rightarrow \mathbb{R}^d \]

\[ \nabla_z T = \begin{bmatrix} \frac{\partial T_1}{\partial z_1} & 0 & \ldots & 0 \\
\frac{\partial T_2}{\partial z_1} & \frac{\partial T_2}{\partial z_2} & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial T_d}{\partial z_1} & \frac{\partial T_d}{\partial z_2} & \ldots & \frac{\partial T_d}{\partial z_d} \end{bmatrix} \]
Invertible transformations
NICE: Non-Linear Independent Components Estimation

\[ x_{1:d} = z_{1:d} \]

\[ x_{d+1:D} = g(z_{d+1:D}; m(z_{1:d})) \]
NICE: Additive coupling layers

\[
x_{1:d} = z_{1:d}
\]

\[
x_{d+1:D} = z_{d+1:D} + m(z_{1:d})
\]
NICE: Additive coupling layers

- **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
NICE: Scaling layers
NICE: Scaling layers

\[
\begin{bmatrix}
S_{1,1} & 0 & \cdots & \cdots & 0 \\
0 & S_{2,2} & 0 & \cdots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
\vdots & \vdots & \ddots & \ddots & 0 \\
0 & 0 & \cdots & \cdots & S_{D,D}
\end{bmatrix}
\begin{bmatrix}
z_1 \\
z_2 \\
\vdots \\
z_D
\end{bmatrix}
\]

Scaling matrix $S$
NICE: Scaling layers

- Forward mapping $z \mapsto x$:
  \[ x_i = s_i z_i \]
  where $s_i > 0$ is the scaling factor for the $i$-th dimension.

- Inverse mapping $x \mapsto z$:
  \[ z_i = \frac{x_i}{s_i} \]

- Jacobian of forward mapping:
  \[ J = \text{diag}(s) \]
  \[ \det(J) = \prod_{i=1}^{n} s_i \]
NICE: Generated samples

(a) Model trained on MNIST
(b) Model trained on TFD
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

*Density estimation using Real NVP (Dinh et al., 2017)*
Real-NVP: Examples
Autoregressive models as flow models

\[ q(x) = q_1(x_1) \cdot q_2(x_2 | x_1) \cdot \ldots \cdot q_d(x_d | x_{<d}) \]

choosing a conditional implicitly fixes a family of triangular maps

\[ x_j = T_j(z_j; \theta_j(z_{<j})) \]
Masked Autoregressive Flow (MAF)

- **Forward mapping from** \( z \mapsto x \):
  - 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

*Masked Autoregressive Flow for Density Estimation (Papamakarios et al., 2017)*
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)
Inverse Autoregressive Flow (IAF)

- Forward mapping from $z \mapsto x$ (parallel):
  - Sample $z_i \sim \mathcal{N}(0,1)$ for $i = 1, \ldots, n$
  - 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 $x \mapsto z$ (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 $z_1, z_2, \ldots, z_n$)

*Improving Variational Inference with Inverse Autoregressive Flow (Kingma et al., 2017)*
IAF is inverse of MAF
Parallel WaveNet

WaveNet Teacher

Linguistic features

Teacher Output

$P(x_i | x_{<i})$

Generated Samples

$x_i = g(z_i | z_{<i})$

WaveNet Student

Linguistic features

Student Output

$P(x_i | z_{<i})$

Input noise

$z_i$
Parallel WaveNet

- **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!**
References

- Normalizing Flow Models
- Primer on Normalizing Flows
- What are normalizing flows?
- Flow-based Deep Generative Models
Thank you!