Deep Generative Models

Lecture 4

Roman Isachenko

Moscow Institute of Physics and Technology

2023, Autumn

Recap of previous lecture

$$\mathcal{L}(\phi, \boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \left[\log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) - \log \frac{q(\mathbf{z}|\mathbf{x}, \phi)}{p(\mathbf{z})} \right] \rightarrow \max_{\phi, \boldsymbol{\theta}}.$$

M-step: $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\phi, \boldsymbol{\theta})$, Monte Carlo estimation

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\phi, \boldsymbol{\theta}) = \int q(\mathbf{z}|\mathbf{x}, \phi) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) d\mathbf{z} \approx$$

$$\approx \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\mathbf{z}^*, \boldsymbol{\theta}), \quad \mathbf{z}^* \sim q(\mathbf{z}|\mathbf{x}, \phi).$$

E-step: $\nabla_{\phi} \mathcal{L}(\phi, \boldsymbol{\theta})$, reparametrization trick

$$\nabla_{\phi} \mathcal{L}(\phi, \boldsymbol{\theta}) = \int r(\epsilon) \nabla_{\phi} \log p(\mathbf{x}|g(\mathbf{x}, \epsilon, \phi), \boldsymbol{\theta}) d\epsilon - \nabla_{\phi} \mathrm{KL}$$

$$\approx \nabla_{\phi} \log p(\mathbf{x}|g(\mathbf{x}, \epsilon^*, \phi), \boldsymbol{\theta}) - \nabla_{\phi} \mathrm{KL}$$

Variational assumption

$$r(\epsilon) = \mathcal{N}(0, \mathbf{I}); \quad q(\mathbf{z}|\mathbf{x}, \phi) = \mathcal{N}(\boldsymbol{\mu}_{\phi}(\mathbf{x}), \boldsymbol{\sigma}^2_{\phi}(\mathbf{x})).$$

$$\mathbf{z} = g(\mathbf{x}, \epsilon, \phi) = \boldsymbol{\sigma}_{\phi}(\mathbf{x}) \cdot \epsilon + \boldsymbol{\mu}_{\phi}(\mathbf{x}).$$

Recap of previous lecture

Final EM-algorithm

- pick random sample $\mathbf{x}_i, i \sim U[1, n]$.
- compute the objective:

$$\boldsymbol{\epsilon}^* \sim r(\boldsymbol{\epsilon}); \quad \mathbf{z}^* = g(\mathbf{x}, \boldsymbol{\epsilon}^*, \boldsymbol{\phi});$$

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) \approx \log p(\mathbf{x}|\mathbf{z}^*, \boldsymbol{\theta}) - KL(q(\mathbf{z}^*|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z}^*)).$$

- compute a stochastic gradients w.r.t. $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\phi}} \log p(\mathbf{x}|g(\mathbf{x}, \boldsymbol{\epsilon}^*, \boldsymbol{\phi}), \boldsymbol{\theta}) - \nabla_{\boldsymbol{\phi}} \mathrm{KL}(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z}));$$
$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\mathbf{z}^*, \boldsymbol{\theta}).$$

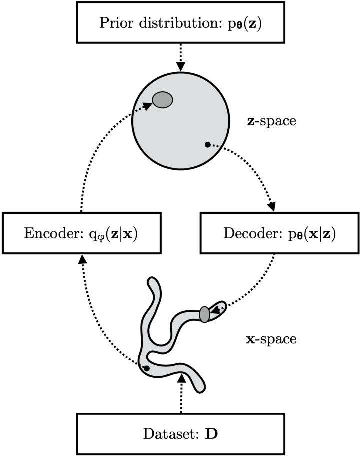
- update $\boldsymbol{\theta}, \boldsymbol{\phi}$ according to the selected optimization method (SGD, Adam, RMSProp):

$$\boldsymbol{\phi} := \boldsymbol{\phi} + \eta \nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}),$$
$$\boldsymbol{\theta} := \boldsymbol{\theta} + \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}).$$

Recap of previous lecture

Variational autoencoder (VAE)

- VAE learns stochastic mapping between $\mathbf{x}$-space, from $\pi(\mathbf{x})$, and a latent $\mathbf{z}$-space, with simple distribution.
- The generative model learns distribution $p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$, with a prior distribution $p(\mathbf{z})$, and a stochastic decoder $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$.
- The stochastic encoder $q(\mathbf{z}|\mathbf{x}, \phi)$ (inference model), approximates the true but intractable posterior $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$.



Kingma D. P., Welling M. An introduction to variational autoencoders, 2019

Recap of previous lecture

VAE objective

$$\log p(\mathbf{x}|\boldsymbol{\theta}) \geq \mathcal{L}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \log \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \phi)} \rightarrow \max_{q,\boldsymbol{\theta}}$$

IWAE objective

$$\mathcal{L}_K(q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z}_1,\dots,\mathbf{z}_K \sim q(\mathbf{z}|\mathbf{x},\phi)} \log \left(\frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \phi)} \right) \rightarrow \max_{\phi,\boldsymbol{\theta}}.$$

Theorem

1. $\log p(\mathbf{x}|\boldsymbol{\theta}) \geq \mathcal{L}_K(q, \boldsymbol{\theta}) \geq \mathcal{L}_M(q, \boldsymbol{\theta}) \geq \mathcal{L}(q, \boldsymbol{\theta}), \quad$ for $K \geq M$;
2. $\log p(\mathbf{x}|\boldsymbol{\theta}) = \lim_{K\to\infty} \mathcal{L}_K(q, \boldsymbol{\theta})$ if $\frac{p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x},\phi)}$ is bounded.

- IWAE makes the variational bound tighter and extends the class of variational distributions.
- Gradient signal becomes really small, training is complicated.
- IWAE is a standard quality measure for VAE models.

Burda Y., Grosse R., Salakhutdinov R. Importance Weighted Autoencoders, 2015

Outline

1. Normalizing flows (NF)
2. Forward and Reverse KL for NF
3. NF examples
   - Linear flows
   - Gaussian autoregressive flows

Outline

1. Normalizing flows (NF)

2. Forward and Reverse KL for NF

3. NF examples
   - Linear flows
   - Gaussian autoregressive flows

Likelihood-based models so far...

Autoregressive models

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{j=1}^{m} p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta})$$

- tractable likelihood,
- no inferred latent factors.

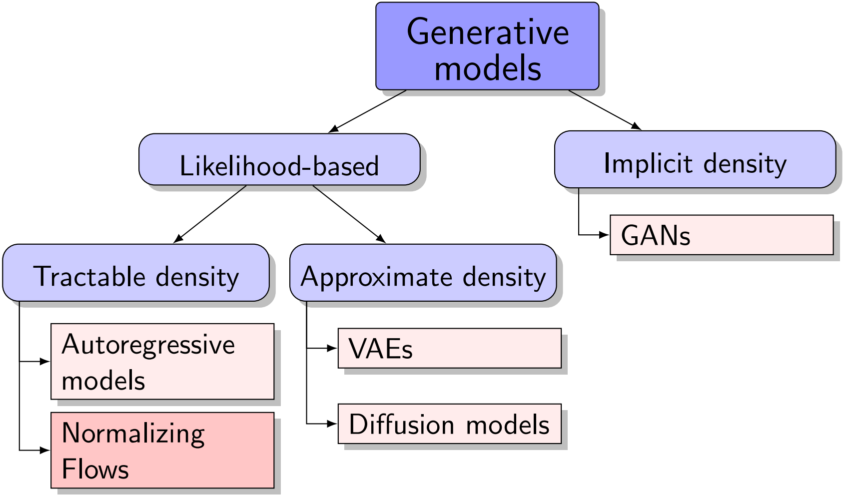
Latent variable models

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) d\mathbf{z}$$

- latent feature representation,
- intractable likelihood.

How to build model with latent variables and tractable likelihood?

Generative models zoo



Normalizing flows prerequisites

Jacobian matrix

Let $f : \mathbb{R}^m \to \mathbb{R}^m$ be a differentiable function.

$$\mathbf{z} = f(\mathbf{x}), \quad \mathbf{J} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \dots & \frac{\partial z_1}{\partial x_m} \\ \dots & \dots & \dots \\ \frac{\partial z_m}{\partial x_1} & \dots & \frac{\partial z_m}{\partial x_m} \end{pmatrix} \in \mathbb{R}^{m \times m}$$

Change of variable theorem (CoV)

Let $\mathbf{x}$ be a random variable with density function $p(\mathbf{x})$ and $f : \mathbb{R}^m \to \mathbb{R}^m$ is a differentiable, **invertible** function (diffeomorphism). If $\mathbf{z} = f(\mathbf{x})$, $\mathbf{x} = f^{-1}(\mathbf{z}) = g(\mathbf{z})$, then

$$p(\mathbf{x}) = p(\mathbf{z})|\det(\mathbf{J}_f)| = p(\mathbf{z}) \left| \det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| = p(f(\mathbf{x})) \left| \det \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

$$p(\mathbf{z}) = p(\mathbf{x})|\det(\mathbf{J}_g)| = p(\mathbf{x}) \left| \det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right) \right| = p(g(\mathbf{z})) \left| \det \left(\frac{\partial g(\mathbf{z})}{\partial \mathbf{z}} \right) \right|.$$

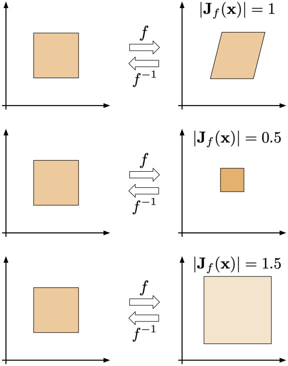
Jacobian determinant

Inverse function theorem

If function f is invertible and Jacobian matrix is continuous and non-singular, then

$$\mathbf{J}_f = \mathbf{J}_{g^{-1}} = \mathbf{J}_g^{-1}; \quad |\det(\mathbf{J}_f)| = \frac{1}{|\det(\mathbf{J}_g)|}.$$

- $\mathbf{x}$ and $\mathbf{z}$ have the same dimensionality ($\mathbb{R}^m$).
- $f(\mathbf{x}, \boldsymbol{\theta})$ could be parametric function.
- Determinant of Jacobian matrix $\mathbf{J} = \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}}$ shows how the volume changes under the transformation.



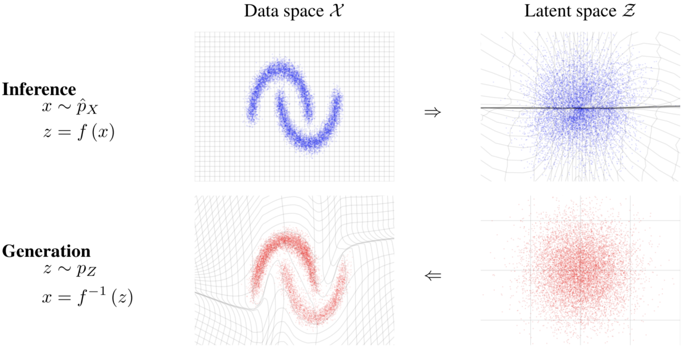
https://jmtomczak.github.io/blog/3/3_flows.html

Fitting normalizing flows

MLE problem

$$p(\mathbf{x}|\boldsymbol{\theta}) = p(\mathbf{z}) \left| \det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| = p(f(\mathbf{x}, \boldsymbol{\theta})) \left| \det \left(\frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}} \right) \right|$$

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log |\det(\mathbf{J}_f)| \to \max_{\boldsymbol{\theta}}$$



Dinh L., Sohl-Dickstein J., Bengio S. Density estimation using Real NVP, 2016

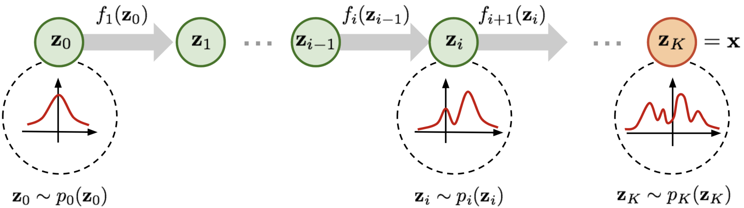
Composition of normalizing flows

Theorem

Diffeomorphisms are **composable** (If $\{f_k\}_{k=1}^K$ satisfy conditions of the change of variable theorem, then $\mathbf{z} = f(\mathbf{x}) = f_K \circ \cdots \circ f_1(\mathbf{x})$ also satisfies it).

$$p(\mathbf{x}) = p(f(\mathbf{x})) \left|\det\left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\right)\right| = p(f(\mathbf{x})) \left|\det\left(\frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \cdots \frac{\partial \mathbf{f}_1}{\partial \mathbf{x}}\right)\right| =$$

$$= p(f(\mathbf{x})) \prod_{k=1}^{K} \left|\det\left(\frac{\partial \mathbf{f}_k}{\partial \mathbf{f}_{k-1}}\right)\right| = p(f(\mathbf{x})) \prod_{k=1}^{K} |\det(\mathbf{J}_{f_k})|$$



https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Normalizing flows (NF)

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log|\det(\mathbf{J}_f)|$$

Definition

Normalizing flow is a *differentiable, invertible* mapping from data $\mathbf{x}$ to the noise $\mathbf{z}$.

- **Normalizing** means that the inverse NF takes samples from $\pi(\mathbf{x})$ and normalizes them into samples from the density $p(\mathbf{z})$.
- **Flow** refers to the trajectory followed by samples from $p(\mathbf{z})$ as they are transformed by the sequence of transformations

$$\mathbf{z} = f_K \circ \dots \circ f_1(\mathbf{x}); \quad \mathbf{x} = f_1^{-1} \circ \dots \circ f_K^{-1}(\mathbf{z}) = g_1 \circ \dots \circ g_K(\mathbf{z})$$

Log likelihood

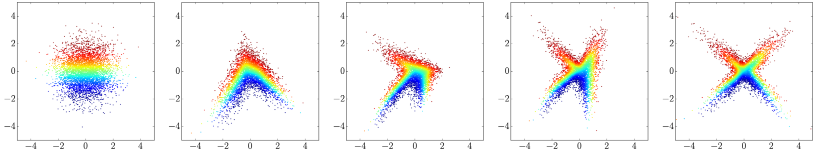
$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f_K \circ \dots \circ f_1(\mathbf{x})) + \sum_{k=1}^{K} \log|\det(\mathbf{J}_{f_k})|,$$

where $\mathbf{J}_{f_k} = \frac{\partial \mathbf{f}_k}{\partial \mathbf{f}_{k-1}}$.

Note: Here we consider only **continuous** random variables.

Normalizing flows

Example of a 4-step NF



NF log likelihood

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log|\det(\mathbf{J}_f)|$$

What is the complexity of the determinant computation?

What we need:

- efficient computation of the Jacobian matrix $\mathbf{J}_f = \frac{\partial f(\mathbf{x},\boldsymbol{\theta})}{\partial \mathbf{x}}$;
- efficient inversion of $f(\mathbf{x}, \boldsymbol{\theta})$;
- loss function to minimize.

Papamakarios G. et al. Normalizing flows for probabilistic modeling and inference, 2019

Outline

Forward KL vs Reverse KL

Forward KL $\equiv$ MLE

$$
\begin{aligned}
KL(\pi || p) &= \int \pi(\mathbf{x}) \log \frac{\pi(\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})} d\mathbf{x} \\
&= -\mathbb{E}_{\pi(\mathbf{x})} \log p(\mathbf{x}|\boldsymbol{\theta}) + \text{const} \to \min_{\boldsymbol{\theta}}
\end{aligned}
$$

Forward KL for NF model

$$
\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log |\det(\mathbf{J}_f)|
$$

$$
KL(\pi || p) = -\mathbb{E}_{\pi(\mathbf{x})} [\log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log |\det(\mathbf{J}_f)|] + \text{const}
$$

- We need to be able to compute $f(\mathbf{x}, \boldsymbol{\theta})$ and its Jacobian.
- We need to be able to compute the density $p(\mathbf{z})$.
- We don't need to think about computing the function $g(\mathbf{z}, \boldsymbol{\theta}) = f^{-1}(\mathbf{z}, \boldsymbol{\theta})$ until we want to sample from the NF.

Forward KL vs Reverse KL

Reverse KL

$$
\begin{aligned}
KL(p||\pi) &= \int p(\mathbf{x}|\boldsymbol{\theta}) \log \frac{p(\mathbf{x}|\boldsymbol{\theta})}{\pi(\mathbf{x})} d\mathbf{x} \\
&= \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})} \left[\log p(\mathbf{x}|\boldsymbol{\theta}) - \log \pi(\mathbf{x}) \right] \rightarrow \min_{\boldsymbol{\theta}}
\end{aligned}
$$

Reverse KL for NF model (LOTUS trick)

$$
\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(\mathbf{z}) + \log |\det(\mathbf{J}_f)| = \log p(\mathbf{z}) - \log |\det(\mathbf{J}_g)|
$$

$$
KL(p||\pi) = \mathbb{E}_{p(\mathbf{z})} \left[\log p(\mathbf{z}) - \log |\det(\mathbf{J}_g)| - \log \pi(g(\mathbf{z}, \boldsymbol{\theta})) \right]
$$

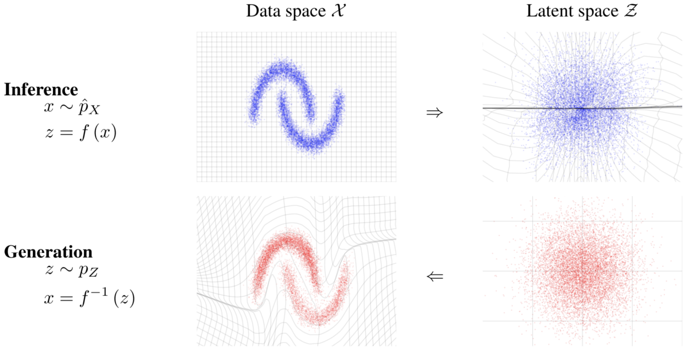
- We need to be able to compute $g(\mathbf{z}, \boldsymbol{\theta})$ and its Jacobian.
- We need to be able to sample from the density $p(\mathbf{z})$ (do not need to evaluate it) and to evaluate(!) $\pi(\mathbf{x})$.
- We don't need to think about computing the function $f(\mathbf{x}, \boldsymbol{\theta})$.

Normalizing flows KL duality

Theorem

Fitting NF model $p(\mathbf{x}|\boldsymbol{\theta})$ to the target distribution $\pi(\mathbf{x})$ using forward KL (MLE) is equivalent to fitting the induced distribution $p(\mathbf{z}|\boldsymbol{\theta})$ to the base $p(\mathbf{z})$ using reverse KL:

$$\arg\min_{\boldsymbol{\theta}} KL(\pi(\mathbf{x})||p(\mathbf{x}|\boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}} KL(p(\mathbf{z}|\boldsymbol{\theta})||p(\mathbf{z})).$$



Papamakarios G. et al. Normalizing flows for probabilistic modeling and inference, 2019

Normalizing flows KL duality

Theorem

$$\arg\min_{\boldsymbol{\theta}} KL(\pi(\mathbf{x})||p(\mathbf{x}|\boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}} KL(p(\mathbf{z}|\boldsymbol{\theta})||p(\mathbf{z})).$$

Proof

- $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta})$, $\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})$;
- $\mathbf{x} \sim \pi(\mathbf{x})$, $\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta})$, $\mathbf{z} \sim p(\mathbf{z}|\boldsymbol{\theta})$;

$$\log p(\mathbf{z}|\boldsymbol{\theta}) = \log \pi(g(\mathbf{z}, \boldsymbol{\theta})) + \log|\det(\mathbf{J}_g)|;$$
$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log|\det(\mathbf{J}_f)|.$$

$$KL\left(p(\mathbf{z}|\boldsymbol{\theta})||p(\mathbf{z})\right) = \mathbb{E}_{p(\mathbf{z}|\boldsymbol{\theta})}\left[\log p(\mathbf{z}|\boldsymbol{\theta}) - \log p(\mathbf{z})\right] =$$
$$= \mathbb{E}_{p(\mathbf{z}|\boldsymbol{\theta})}\left[\log \pi(g(\mathbf{z}, \boldsymbol{\theta})) + \log|\det(\mathbf{J}_g)| - \log p(\mathbf{z})\right] =$$
$$= \mathbb{E}_{\pi(\mathbf{x})}\left[\log \pi(\mathbf{x}) - \log|\det(\mathbf{J}_f)| - \log p(f(\mathbf{x}, \boldsymbol{\theta}))\right] =$$
$$= \mathbb{E}_{\pi(\mathbf{x})}\left[\log \pi(\mathbf{x}) - \log p(\mathbf{x}|\boldsymbol{\theta})\right] = KL(\pi(\mathbf{x})||p(\mathbf{x}|\boldsymbol{\theta})).$$

Papamakarios G., Pavlakou T., Murray I. Masked Autoregressive Flow for Density Estimation, 2017

Outline

1. Normalizing flows (NF)

2. Forward and Reverse KL for NF

3. NF examples
   - Linear flows
   - Gaussian autoregressive flows

Outline

Jacobian structure

Normalizing flows log-likelihood

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log \left| \det \left(\frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}} \right) \right|$$

The main challenge is a determinant of the Jacobian matrix.

What is the $det(\mathbf{J})$ in the following cases?

Consider a linear layer $\mathbf{z} = \mathbf{W}\mathbf{x}$, $\mathbf{W} \in \mathbb{R}^{m \times m}$.

1. Let $\mathbf{z}$ be a permutation of $\mathbf{x}$.
2. Let z_j depend only on x_j.

$$\log \left| \det \left(\frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}} \right) \right| = \log \left| \prod_{j=1}^{m} \frac{\partial f_j(x_j, \boldsymbol{\theta})}{\partial x_j} \right| = \sum_{j=1}^{m} \log \left| \frac{\partial f_j(x_j, \boldsymbol{\theta})}{\partial x_j} \right|.$$

3. Let z_j depend only on $\mathbf{x}_{1:j}$ (autoregressive dependency).

Linear normalizing flows

$$\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{W}\mathbf{x}, \quad \mathbf{W} \in \mathbb{R}^{m \times m}, \quad \boldsymbol{\theta} = \mathbf{W}, \quad \mathbf{J}_f = \mathbf{W}^T$$

In general, we need $O(m^3)$ to invert matrix.

Invertibility

- Diagonal matrix $O(m)$.
- Triangular matrix $O(m^2)$.
- It is impossible to parametrize all invertible matrices.

Invertible 1x1 conv

$\mathbf{W} \in \mathbb{R}^{c \times c}$ - kernel of 1x1 convolution with c input and c output channels. The computational complexity of computing or differentiating $\det(\mathbf{W})$ is $O(c^3)$. Cost to compute $\det(\mathbf{W})$ is $O(c^3)$. It should be invertible.

Kingma D. P., Dhariwal P. Glow: Generative Flow with Invertible 1x1 Convolutions, 2018

Linear normalizing flows

$$\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{W}\mathbf{x}, \quad \mathbf{W} \in \mathbb{R}^{m \times m}, \quad \boldsymbol{\theta} = \mathbf{W}, \quad \mathbf{J}_f = \mathbf{W}^T$$

Matrix decompositions

- **LU-decomposition**

  $$\mathbf{W} = \mathbf{P}\mathbf{L}\mathbf{U},$$

  where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is lower triangular with positive diagonal, $\mathbf{U}$ is upper triangular with positive diagonal.
- **QR-decomposition**

  $$\mathbf{W} = \mathbf{Q}\mathbf{R},$$

  where $\mathbf{Q}$ is an orthogonal matrix, $\mathbf{R}$ is an upper triangular matrix with positive diagonal.

Decomposition should be done only once in the beggining. Next, we fit decomposed matrices ($\mathbf{P}/\mathbf{L}/\mathbf{U}$ or $\mathbf{Q}/\mathbf{R}$).

Kingma D. P., Dhariwal P. Glow: Generative Flow with Invertible 1x1 Convolutions, 2018

Hoogeboom E., et al. Emerging convolutions for generative normalizing flows, 2019

Outline

1. Normalizing flows (NF)
2. Forward and Reverse KL for NF
3. NF examples
   - Linear flows
   - Gaussian autoregressive flows

Gaussian autoregressive model

Consider an autoregressive model

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{j=1}^{m} p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta}), \quad p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta}) = \mathcal{N}\left(\mu_j(\mathbf{x}_{1:j-1}), \sigma_j^2(\mathbf{x}_{1:j-1})\right).$$

Sampling: reparametrization trick

$$x_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot z_j + \mu_j(\mathbf{x}_{1:j-1}), \quad z_j \sim \mathcal{N}(0, 1).$$

Inverse transform

$$z_j = (x_j - \mu_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\sigma_j(\mathbf{x}_{1:j-1})}.$$

- We have an **invertible** and **differentiable** transformation from $p(\mathbf{z})$ to $p(\mathbf{x}|\boldsymbol{\theta})$.
- It is an autoregressive (AR) NF with the base distribution $p(\mathbf{z}) = \mathcal{N}(0, 1)$!
- Jacobian of such transformation is triangular!

Kingma D. P. et al. Improving Variational Inference with Inverse Autoregressive Flow, 2016

Gaussian autoregressive NF

$$\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta}) \quad \Rightarrow \quad x_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot z_j + \mu_j(\mathbf{x}_{1:j-1}).$$

$$\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta}) \quad \Rightarrow \quad z_j = (x_j - \mu_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\sigma_j(\mathbf{x}_{1:j-1})}.$$

Generation function $g(\mathbf{z}, \boldsymbol{\theta})$ is **sequential**.
Inference function $f(\mathbf{x}, \boldsymbol{\theta})$ is **not sequential**.

Forward KL for NF

$$KL(\pi || p) = -\mathbb{E}_{\pi(\mathbf{x})} \left[\log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log |\det(\mathbf{J}_f)|\right] + \text{const}$$

- We need to be able to compute $f(\mathbf{x}, \boldsymbol{\theta})$ and its Jacobian.
- We need to be able to compute the density $p(\mathbf{z})$.
- We don't need to think about computing the function $g(\mathbf{z}, \boldsymbol{\theta}) = f^{-1}(\mathbf{z}, \boldsymbol{\theta})$ until we want to sample from the model.

Papamakarios G., Pavlakou T., Murray I. Masked Autoregressive Flow for Density Estimation, 2017

Summary

- Change of variable theorem allows to get the density function of the random variable under the invertible transformation.
- Normalizing flows transform a simple base distribution to a complex one via a sequence of invertible transformations with tractable Jacobian.
- Normalizing flows have a tractable likelihood that is given by the change of variable theorem.
- We fit normalizing flows using forward or reverse KL minimization.
- Linear NF try to parametrize set of invertible matrices via matrix decompositions.
- Gaussian autoregressive NF is an autoregressive model with triangular Jacobian. It has fast inference function and slow generation function. Forward KL is a natural loss function.