[Paper review 33]

Improved Variational Inference with Inverse Autoregressive Flow

(Kingma, et al. 2016)

[Contents]

1. Abstract

NF (Normalizing Flow)

- strategy for flexible VI

"Inverse Autoregressive Flow (IAF)"

- new type of NF, which scales well to high-dimensional latent spaces
- consists of cahin of invertible transformations
  (each transformation is based on a "Autoregressive NN")

2. Introduction

SVI : scalable posterior inference, using stochastic gradient update

VAE : NN with inference network & generative network

IAF : scales well to high-dimensional latent space

IAF (Inverse Autoregressive Flow)

- lie in Gaussian autoregressive functions

(normally used for density estimation)

```
- input : variable with some specific ordering
- output : mean and std for each element
```

ex) RNN, MADE, PixelCNN, WaveNet

- such functions can....
  - be turned into "invertible" nonlinear transformations
  - with a "simple Jacobian determinant"

  $\rightarrow$ flexibliity + known determinant = Normalizing Flow

demonstrate this method by "improving the inference network" of deep VAE

3. Variational inference and Learning

notation

- x : observed variable
- z : latent variable
- $p(x, z)$: joint pdf $\rightarrow$ "generative model"

Maximize ELBO

- $\log p(\mathrm{x}) \geq \mathbb{E}_{q(\mathrm{z}|\mathrm{x})}[\log p(\mathrm{x}, \mathrm{z}) - \log q(\mathrm{z} \mid \mathrm{x})] = \mathcal{L}(\mathrm{x}; \boldsymbol{\theta}) \log p(\mathbf{x}) - D_{KL}(q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x}))$
- re-parameterization trick in $q(\mathbf{z} \mid \mathbf{x})$

Models with multiple latent variable

- factorize $q(\mathbf{z} \mid \mathbf{x})$

  (factorize into partial inference models with some ordering)
- $q(\mathbf{z}_a, \mathbf{z}_b \mid \mathbf{x}) = q(\mathbf{z}_a \mid \mathbf{x}) \, q(\mathbf{z}_b \mid \mathbf{z}_a, \mathbf{x})$

3-1. Requirements for computational tractability

have to efficiently optimize ELBO!

Computationally efficient to...

- 1) compute and differentiate $q(z \mid x)$
- 2) sample from it

  (since both these operations need to be performed for each datapoint in a minibatch at every iteration of optimization)

example) diagonal posterior

- $q(\mathbf{z} \mid \mathbf{x}) \sim \mathcal{N}\left(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}^2(\mathbf{x})\right)$,
- but not much flexible....

3-2. Normalizing Flow

NF

- in the context of SGVI (Stochastic Gradient Variational Inference)
- build flexible posterior distribution using "iterative procedure"

$\mathrm{z}_0 \sim q\left(\mathrm{z}_0 \mid \mathrm{x}\right), \quad \mathrm{z}_t=\mathrm{f}_t\left(\mathrm{z}_{t-1}, \mathrm{x}\right) \quad \forall t=1 \ldots T$

$\log q\left(\mathbf{z}_T \mid \mathbf{x}\right)=\log q\left(\mathbf{z}_0 \mid \mathbf{x}\right)-\sum_{t=1}^T \log \operatorname{det}\left|\frac{d \mathbf{z}_t}{d \mathbf{z}_{t-1}}\right|$

$\mathbf{f}_t\left(\mathbf{z}_{t-1}\right)=\mathbf{z}_{t-1}+\mathbf{u} h\left(\mathbf{w}^T \mathbf{z}_{t-1}+b\right)$

- $\mathbf{u} h\left(\mathbf{w}^T \mathbf{z}_{t-1}+b\right)$ can be interpreted as a MLP with "bottleneck hidden layer" with a single unit

  $\rightarrow$ problem : long chain of transform is needed in high-dimension

4. Inverse Autoregressive Transformations

for NF that scales well to high-dimensional space...

- consider Gaussian version of autoregressive AE

  (ex. MADE, PixeCNN)

Notation

- $[\boldsymbol{\mu}(\mathbf{y}), \boldsymbol{\sigma}(\mathbf{y})]$: function of the vector y, to the vectors μ and σ
- $\left[\mu_i\left(\mathbf{y}_{1: i-1}\right), \sigma_i\left(\mathbf{y}_{1: i-1}\right)\right]$: predicted mean and standard deviation of the i -th element of y

  (Due to the autoregressive structure, Jacobian is lower triangular with zeros on the diagonal:

  $\partial\left[\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i\right] / \partial \mathbf{y}_j=[0,0]$ for $j \geq i$)
- $\epsilon \sim \mathcal{N}(0, \mathrm{I})$: sample from noise vector
- y
  - $y_0=\mu_0+\sigma_0 \odot \epsilon_0$
  - $y_i=\mu_i\left(\mathrm{y}_{1: i-1}\right)+\sigma_i\left(\mathrm{y}_{1: i-1}\right) \cdot \epsilon_i$ where $i>0$

Computation involved in this transformation : proportional to dimension D

- but inverse transformation is interesting in NF!

  $\epsilon_i=\frac{y_i-\mu_i\left(\mathbf{y}_{1: i-1}\right)}{\sigma_i\left(\mathbf{y}_{1: i-1}\right)}$

2 key observations

- 1) inverse transformation can be parallelized : $\boldsymbol{\epsilon} = (\mathbf{y} - \boldsymbol{\mu}(\mathbf{y}))/\boldsymbol{\sigma}(\mathbf{y})$

  (individual element ϵ_i do not depend on each other!)

- 2) inverse autoregressive operation has a simple Jacobian determinant

  due to the autoregressive structure ($\partial\left[\mu_i, \sigma_i\right] / \partial y_j = [0,0]$ for $j \geq i$)

  - $\partial \epsilon_i / \partial y_j = 0$ for $j > i$
  - $\partial \epsilon_i / \partial y_i = \sigma_i$ (simple diagonal)

  Thus, $\log \det\left|\frac{d\epsilon}{d\mathbf{y}}\right| = \sum_{i=1}^{D} - \log \sigma_i(\mathbf{y})$

5. Inverse Autoregressive Flow (IAF)

based on $\boldsymbol{\epsilon} = (\mathbf{y} - \boldsymbol{\mu}(\mathbf{y}))/\boldsymbol{\sigma}(\mathbf{y})$ (Inverse Autoregressive Transformations)

Algorithm 1: Pseudo-code of an approximate posterior with Inverse Autoregressive Flow (IAF)

Data:

$\mathbf{x}$: a datapoint, and optionally other conditioning information
$\boldsymbol{\theta}$: neural network parameters
$\texttt{EncoderNN}(\mathbf{x}; \boldsymbol{\theta})$: encoder neural network, with additional output $\mathbf{h}$
$\texttt{AutoregressiveNN}[*](\mathbf{z}, \mathbf{h}; \boldsymbol{\theta})$: autoregressive neural networks, with additional input $\mathbf{h}$
$\texttt{sum}(.)$: sum over vector elements
$\texttt{sigmoid}(.)$: element-wise sigmoid function

Result:

$\mathbf{z}$: a random sample from $q(\mathbf{z}|\mathbf{x})$, the approximate posterior distribution
l: the scalar value of $\log q(\mathbf{z}|\mathbf{x})$, evaluated at sample 'z'

$[\boldsymbol{\mu}, \boldsymbol{\sigma}, \mathbf{h}] \leftarrow \texttt{EncoderNN}(\mathbf{x}; \boldsymbol{\theta})$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$
$\mathbf{z} \leftarrow \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} + \boldsymbol{\mu}$
$l \leftarrow -\texttt{sum}(\log \boldsymbol{\sigma} + \frac{1}{2}\boldsymbol{\epsilon}^2 + \frac{1}{2}\log(2\pi))$
for $t \leftarrow 1$ **to** T **do**
  $[\mathbf{m}, \mathbf{s}] \leftarrow \texttt{AutoregressiveNN}[t](\mathbf{z}, \mathbf{h}; \boldsymbol{\theta})$
  $\boldsymbol{\sigma} \leftarrow \texttt{sigmoid}(\mathbf{s})$
  $\mathbf{z} \leftarrow \boldsymbol{\sigma} \odot \mathbf{z} + (1 - \boldsymbol{\sigma}) \odot \mathbf{m}$
  $l \leftarrow l - \texttt{sum}(\log \boldsymbol{\sigma})$
end

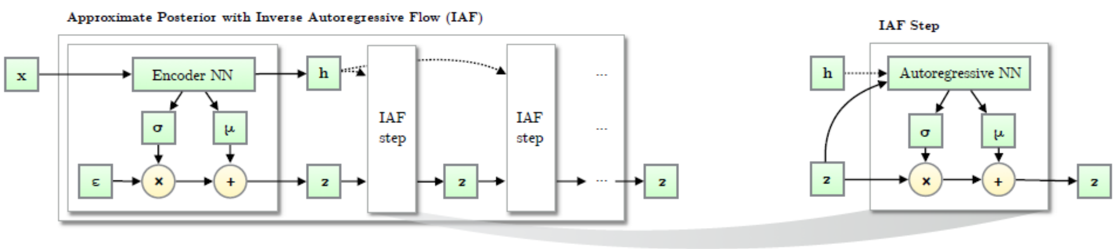

Figure 2: Like other normalizing flows, drawing samples from an approximate posterior with Inverse Autoregressive Flow (IAF) consists of an initial sample $\mathbf{z}$ drawn from a simple distribution, such as a Gaussian with diagonal covariance, followed by a chain of nonlinear invertible transformations of $\mathbf{z}$, each with a simple Jacobian determinants.

chain of T : $\mathbf{z}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1}$

autoregressive w.r.t. z_{t-1}

- Jacobians $\frac{d\mu_t}{d\mathbf{z}_{t-1}}$ and $\frac{d\sigma_t}{d\mathbf{z}_{t-1}}$ are triangular with zeros on the diagonal.

  Thus, $\frac{d\mathbf{z}_t}{d\mathbf{z}_{t-1}}$ is triangular with σ_t on the diagonal, with determinant $\prod_{i=1}^{D} \sigma_{t,i}$

$$\log q\left(\mathbf{z}_T \mid \mathbf{x}\right) = -\sum_{i=1}^{D}\left(\frac{1}{2}\epsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T}\log\sigma_{t,i}\right)$$

Autoregressive Neural Network

$$\begin{aligned} [\mathbf{m}_t, \mathbf{s}_t] &\leftarrow \text{AutoregressiveNN}[t](\mathbf{z}_t, \mathbf{h}; \boldsymbol{\theta}) \\ \boldsymbol{\sigma}_t &= \operatorname{sigmoid}(\mathbf{s}_t) \\ \mathbf{z}_t &= \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1} + (1-\boldsymbol{\sigma}_t) \odot \mathbf{m}_t \end{aligned}$$

Autoregressive NN form a rich family of nonlinear transformations for IAF!

6. Summary

(by Coursera)

Inverse Autoregressive Flow (IAF)

The inverse autoregressive flow reverses the dependencies to make the forward pass parallelisable, but the inverse pass sequential.

It uses the same equations:

$$x_i = \mu_i + \exp(\sigma_i) z_i \qquad i = 1, \ldots, D$$

but has the scale and shift functions depend on the z_i instead of the x_i:

$$\mu_i = f_{\mu_i}(z_1, \ldots, z_{i-1}) \qquad \sigma_i = f_{\sigma_i}(z_1, \ldots, z_{i-1}).$$

Note that now the forward equation (determining $\mathbf{x}$ from $\mathbf{z}$) can be parallelised, but the reverse transformations require determining z_1, followed by z_2, etc. and must hence be solved in sequence.