[Paper review 32]

MADE : Masked Autoencoder for Distribution Estimation

(Germain, et al. 2015)

[Contents]

0. Overview

(by Coursera)

Autoregressive Flow

For some matrices, calculating a determinant is easy.

Ex) lower or upper triangular matrix

- the determinant is the product of the diagonal elements, of which there are D, \
  (meaning the determinant calculation scales linearly.)
- Hence, to attain a linear scaling of the determinant in the number of dimensions, it is enough to enforce that $\frac{\partial f_i}{\partial z_j} = 0$ whenever $j > i$. \
  (In other words, the component f_i depends only on $z_1, \ldots z_i$.)

Autoregressive models can be reinterpreted as normalising flows that fulfil this requirement.\
These are models that model the joint density $p(\mathbf{x})$ as the product of conditionals $\prod_i p(x_i \mid \mathbf{x}_{1:i-1})$.

example

For example, the conditionals could be parameterised as Gaussians:

$$\begin{aligned} p(x_i \mid \mathbf{x}_{1:i-1}) &= \mathcal{N}(x_i \mid \mu_i, \exp(\sigma_i)^2), \\ \text{where} \qquad \mu_i &= f_{\mu_i}(\mathbf{x}_{1:i-1}) \\ \text{and} \qquad \sigma_i &= f_{\sigma_i}(\mathbf{x}_{1:i-1}). \end{aligned}$$

Mean and standard deviations of each conditional distribution are "computed using (parameterised) functions of all previous variables"

The above can alternatively be written as:

$$x_i = \mu_i(\mathbf{x}_{1:i-1}) + \exp(\sigma_i(\mathbf{x}_{1:i-1}))z_i \qquad i = 1, \ldots, D$$

where $z_i \sim N(0,1)$ is sampled from a unit Gaussian.

This last equation shows how the autoregressive model can be viewed as a transformation f from the random variables $\mathbf{z} \in \mathbb{R}^D$ to the data $\mathbf{x} \in \mathbb{R}^D$.

This is an example of an *autoregressive* process where x_i depends only on the components of $\mathbf{z}$ that are lower than or equal to i but not any of the higher ones. The dependence on lower dimensions of $\mathbf{z}$ happens indirectly through the x_i dependence in the f_{μ_i} and f_{σ_i}.

Implementation

Implementation

- Masked Autoregressive Flow (MAF)
  - George Papamakarios, Theo Pavlakou, Iain Murray (2017). Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*, 2017.
- Inverse Autoregressive Flow (IAF)
  - Diederik Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, Max Welling (2016). Improved Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Processing Systems*, 2016.
- Real-NVP & NICE
  - Laurent Dinh, Jascha Sohl-Dickstein, Samy Bengio (2016). Density estimation using Real NVP.
  - Laurent Dinh, David Krueger, Yoshua Bengio (2014). NICE: Non-linear Independent Components Estimation.

1. Abstract

use NN to "estimate a distribution"

introduce simple modification for AutoEncoder(AE) NN

- key point : masks the AE parameters to respect "autoregressive constraints"
  - each input is reconstructed only from the previous inputs

with "autoregressive constraints", the outputs are "conditional probabilities"

Can also train a single NN that can decompose the joint pdf in multiple different ordering

2. Introduction

Distribution estimation : estimating $p(\mathbf{x})$ from a set of $\left\{\mathbf{x}^{(t)}\right\}_{t=1}^{T}$

Curse of dimensionality

- as the number of dimensions of input space x grows, the volume space exponentially increases
- to solve, various models have been proposed
  - ex) Autoregressive models

This paper focuses on Autoregressive models

- computing $p(x)$ for test data is tractable
- but, computational cost is high!

  ($O(D)$ times more than simple NN point predictor)

Contribution

- simple way of adapting AE N, making faster than existing alternatives
- use MASK to the weighted connections of the standard AE, to convert it to a "distribution estimator"

Key point : use a MASK!

- mask which are designed in a way that output is autoregressive!

  (= each input dim is reconstructed only from dim preceding it)

Result : MADE = Masked Autoencoder Distribution Estimator

- "preserves the efficiency of a single pass through a regular AE"

3. Autoencoders

given a training set $\left\{\mathbf{x}^{(t)}\right\}_{t=1}^{T}$

AE : learns a hidden representation $h(x)$ of its input x

- encode) $\mathbf{h}(\mathbf{x})=\mathbf{g}(\mathbf{b}+\mathbf{W}\mathbf{x})$
- decode) $\widehat{\mathbf{x}}=\operatorname{sigm}(\mathbf{c}+\mathbf{V}\mathbf{h}(\mathbf{x}))$

loss function (= cross-entropy loss)

- $\ell(\mathrm{x})=\sum_{d=1}^{D}-x_{d} \log \widehat{x}_{d}-\left(1-x_{d}\right) \log \left(1-\widehat{x}_{d}\right)$
- optimize w.r.t $\{\mathbf{W}, \mathbf{V}, \mathbf{b}, \mathbf{c}\}$, using SGD

Advantage

- deep AE (insert more hidden layres) : flexibility

Disadvantage

- representations that in learns can be trivial

  (if hidden layer is as large as input .... can just "copy" the input dim)

  (loss function is not proper!)

  ```
  - $q(\mathbf{x})=\prod_{d} \widehat{x}_{d}^{x_{d}}\left(1-
  \widehat{x}_{d}\right)^{1-x_{d}}$
  - $\sum_{\mathbf{x}} q(\mathbf{x}) \neq 1$
  ```

4. Distribution Estimation as Autoregression

would like to be able to write $p(x)$ in a way that...

- could be computed based on the output of properly corrected AE

product rule

- $p(\mathrm{x})=\prod_{d=1}^{D} p\left(x_{d} \mid \mathrm{x}<d\right) \qquad \text{where } \mathrm{x}_{<d}=\left[x_{1}, \ldots, x_{d-1}\right]^{\top}$

Let

- $p\left(x_{d}=1 \mid \mathbf{x}_{<d}\right)=\hat{x}_{d}$,
- $p\left(x_{d}=0 \mid \mathrm{x}_{<d}\right)=1-\hat{x}_{d}$

  $\rightarrow$ autoregressive property

  (this provides a way to define AE that can be used for "distribution estimation")

Then, our loss function becomes

- before)

  $\ell(\mathrm{x})=\sum_{d=1}^{D}-x_{d} \log \widehat{x}_{d}-\left(1-x_{d}\right) \log \left(1-\widehat{x}_{d}\right)$

- after)

$$\begin{aligned}
-\log p(\mathrm{x}) &=\sum_{d=1}^{D}-\log p\left(x_{d} \mid \mathrm{x}<d\right) \\
&=\sum_{d=1}^{D}-x_{d} \log p\left(x_{d}=1 \mid \mathrm{x}_{<d}\right)-\left(1-x_{d}\right) \log p\left(x_{d}=0 \mid \mathrm{x}_{<d}\right) \\
&=\ell(\mathrm{x})
\end{aligned}$$

5. Masked Autoencoders

how to modify AE to satisfy autoregressive property?

- No computational path between $\hat{x}_d$ and $x_d, \ldots, x_D$
- by ZEROING connections

  (element-wise multiply each matrix by a binary mask matrix)

with Mask

- encoder) $\mathbf{h}(\mathbf{x})=\mathbf{g}\left(\mathbf{b}+\left(\mathbf{W} \odot \mathbf{M}^{\mathbf{W}}\right) \mathbf{x}\right)$
- decoder) $\hat{\mathbf{x}}=\operatorname{sigm}\left(\mathbf{c}+\left(\mathbf{V} \odot \mathbf{M}^{\mathbf{V}}\right) \mathbf{h}(\mathbf{x})\right)$
- it is left to the masks M^{W} and M^{V} to satisfy autoregressive property

5-1. Imposing autoregressive property

- 1) assign each unit (in hidden layer) an integer m (between 1 and $D-1$)

  ($m(k)$ = maximum number of input units to which it can be connected)

  ($m(k) \neq 1, m(k) \neq D$)

- 2) [MASK] matrix masking the connections between "input & hidden units"

  constraints on the maximum number of inputs to each hidden unit are encoded in it!

  $$M_{k, d}^{\mathrm{W}}=1_{m(k) \geq d}=\begin{cases}1 & \text{if } m(k) \geq d \\ 0 & \text{otherwise}\end{cases}$$

- 3) [MASK] matrix masking the connections between "hidden & output" units

  $$M_{d, k}^{\mathrm{V}}=1_{d>m(k)}=\begin{cases}1 & \text{if } d>m(k) \\ 0 & \text{otherwise}\end{cases}$$

Notation & Meaning

- M^{V} and M^{W} : network's connectivity
- matrix product $\mathbf{M}^{\mathrm{V}, \mathrm{W}}=\mathrm{M}^{\mathrm{V}} \mathrm{M}^{\mathrm{W}}$: connectivity between the input and the output layer
- $M_{d^{\prime}, d}^{\mathrm{V}, \mathrm{W}}$: number of network paths between output unit $\hat{x}_{d^{\prime}}$ and input unit x_d.

with the above, we can show that the "autoregressive property" is made!

proof)

- need to show that $\mathrm{M}^{\mathrm{V}, \mathrm{W}}$ is strictly lower diagonal ($M_{d^{\prime}, d}^{\mathrm{V}, \mathrm{W}}$ is 0 if $d^{\prime} \leq d$)
- $M_{d^{\prime}, d}^{\mathrm{V}, \mathrm{W}}=\sum_{k=1}^K M_{d^{\prime}, k}^{\mathrm{V}} M_{k, d}^{\mathrm{W}}=\sum_{k=1}^K 1_{d^{\prime}>m(k)} 1_{m(k) \geq d}$
  - If $d^{\prime} \leq d$, then there are no values for $m(k)$

- $\therefore M_{d',d}^{\mathrm{V,W}}$ =0

when constructing masks,

- only requires an assignment of $m(k)$ to each hidden unit! SIMPLE!
- set $m(k)$ by sampling from Uniform discrete dist'n ($1 \sim D-1$), independently for each K hidden units

5-2. Deep MADE

generalizes to deep NN ($L>1$ hidden layers)

Notation

- W^{1} : first hidden layer matrix
- W^{2} : second hidden layer matrix
- K^{l} : number of hidden units in layer l
- $m^{l}(k)$: maximum number of connected inputs of the k^{th} unit in the l^{th} layer

Mask

- $M_{k',k}^{\mathbf{W}^{l}}=1_{m^{l}(k') \geq m^{l-1}(k)}=\begin{cases}1 & \text{if } m^{l}(k') \geq m^{l-1}(k) \\ 0 & \text{otherwise}\end{cases}$
- $M_{d,k}^{\mathrm{V}}=1_{d>m^{L}(k)}=\begin{cases}1 & \text{if } d>m^{L}(k) \\ 0 & \text{otherwise}\end{cases}$

5-3. Order-agnostic training

interested in a modeling the conditionals, associated with "an arbitrary ordering of input's dim"

training an AR model on ALL ordering can be beneficial

$\rightarrow$ order-agnostic training

order-agnostic training

- can be achieved by "sampling an ordering" before each (stochastic/minibatch gradient) update
- 2 advantages
  - 1) missing values in some input vectors can be imputed efficiently
  - 2) ensemble of AR models can be constructed on the fly (??)

ordering

- $\mathbf{m}^{0}=\left[m^{0}(1), \ldots, m^{0}(D)\right]$.

  ($m^{0}(d)$: position of the original d^{th} dimension of x in the product of conditionals)

- random ordering can be obtained by "randomly permuting" the ordered vector $[1, \ldots, D]$

5-4. Connectivity-agnostic training

in addition to choosing an ordering (in 5-3),

- also have to choose each hidden unit's connectivity constraint, $m^l(k)$
  (= agnostic of the connectivity pattern generated by these constraints)

By resampling the connectivity of hidden units for every update,

$\rightarrow$ each hidden unit will have a constantly changing number of incoming inputs during training

6. Algorithm Summary

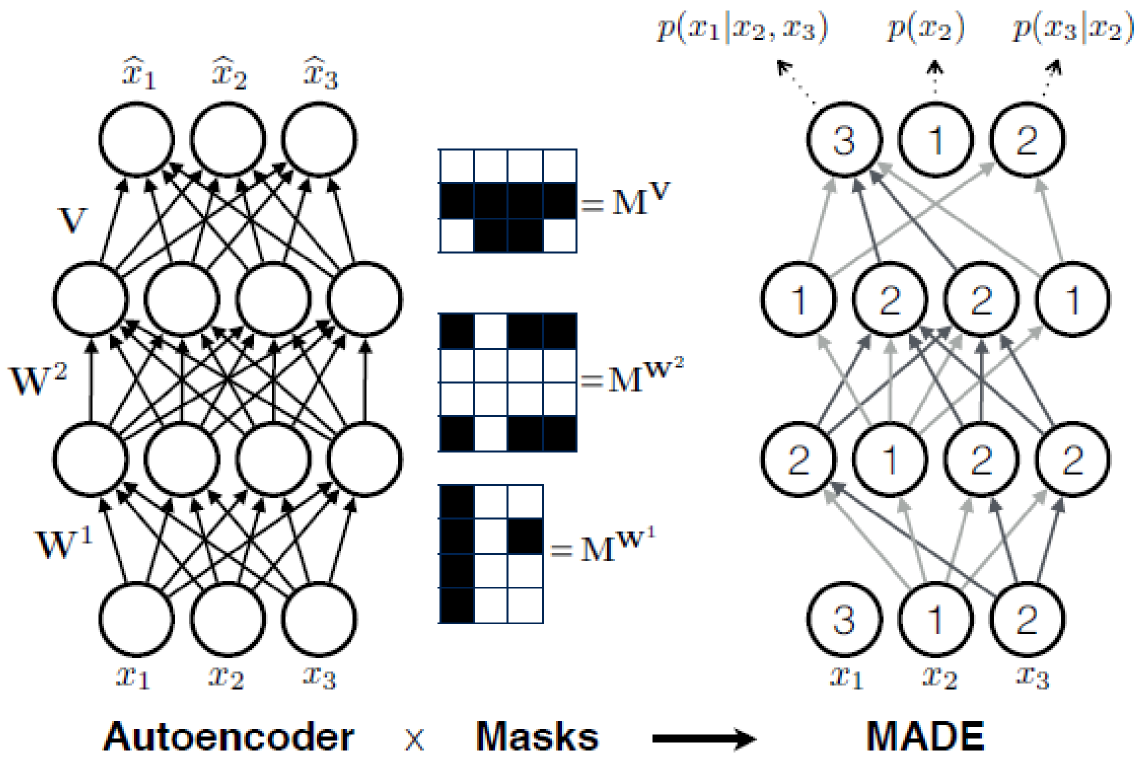

Figure 1. **Left: Conventional three hidden layer autoencoder.** Input in the bottom is passed through fully connected layers and point-wise nonlinearities. In the final top layer, a reconstruction specified as a probability distribution over inputs is produced. As this distribution depends on the input itself, a standard autoencoder cannot predict or sample new data. **Right: MADE.** The network has the same structure as the autoencoder, but a set of connections is removed such that each input unit is only predicted from the previous ones, using multiplicative binary masks ($\mathbf{M}^{\mathbf{W}^1}, \mathbf{M}^{\mathbf{W}^2}, \mathbf{M}^{\mathbf{V}}$). In this example, the ordering of the input is changed from 1,2,3 to 3,1,2. This change is explained in section 4.2, but is not necessary for understanding the basic principle. The numbers in the hidden units indicate the maximum number of inputs on which the k^{th} unit of layer l depends. The masks are constructed based on these numbers (see Equations 12 and 13). These masks ensure that MADE satisfies the autoregressive property, allowing it to form a probabilistic model, in this example $p(\mathbf{x}) = p(x_2)\,p(x_3|x_2)\,p(x_1|x_2,x_3)$. Connections in light gray correspond to paths that depend only on 1 input, while the dark gray connections depend on 2 inputs.

Algorithm 1 Computation of $p(\mathbf{x})$ and learning gradients for MADE with order and connectivity sampling. D is the size of the input, L the number of hidden layers and K the number of hidden units.

```
**Input:** training observation vector $\mathbf{x}$
**Output:** $p(\mathbf{x})$ and gradients of $-\log p(\mathbf{x})$ on parameters

# Sampling $\mathbf{m}^l$ vectors
$\mathbf{m}^0 \leftarrow \text{shuffle}([1, \ldots, D])$
**for** $l$ from 1 to $L$ **do**
  **for** $k$ from 1 to $K^l$ **do**
    $m^l(k) \leftarrow \text{Uniform}([\min_{k'} m^{l-1}(k'), \ldots, D-1])$
  **end for**
**end for**

# Constructing masks for each layer
**for** $l$ from 1 to $L$ **do**
  $\mathbf{M}^{\mathbf{W}^l} \leftarrow \mathbf{1}_{\mathbf{m}^l \geq \mathbf{m}^{l-1}}$
**end for**
$\mathbf{M}^{\mathbf{V}} \leftarrow \mathbf{1}_{\mathbf{m}^0 > \mathbf{m}^L}$
```

```
# Computing $p(\mathbf{x})$
$\mathbf{h}^0(\mathbf{x}) \leftarrow \mathbf{x}$
for $l$ from 1 to $L$ do
    $\mathbf{h}^l(\mathbf{x}) \leftarrow \mathbf{g}(\mathbf{b}^l + (\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l})\mathbf{h}^{l-1}(\mathbf{x}))$
end for
$\widehat{\mathbf{x}} \leftarrow \mathrm{sigm}(\mathbf{c} + (\mathbf{V} \odot \mathbf{M}^{\mathbf{V}})\mathbf{h}^L(\mathbf{x}))$
$p(\mathbf{x}) \leftarrow \exp\left(\sum_{d=1}^{D} x_d \log \widehat{x}_d + (1-x_d)\log(1-\widehat{x}_d)\right)$

# Computing gradients of $-\log p(\mathbf{x})$
$\mathrm{tmp} \leftarrow \widehat{\mathbf{x}} - \mathbf{x}$
$\delta\mathbf{c} \leftarrow \mathrm{tmp}$
$\delta\mathbf{V} \leftarrow \left(\mathrm{tmp}\, \mathbf{h}^L(\mathbf{x})^\top\right) \odot \mathbf{M}^{\mathbf{V}}$
$\mathrm{tmp} \leftarrow (\mathrm{tmp}^\top(\mathbf{V} \odot \mathbf{M}^{\mathbf{V}}))^\top$
for $l$ from $L$ to 1 do
    $\mathrm{tmp} \leftarrow \mathrm{tmp} \odot \mathbf{g}'(\mathbf{b}^l + (\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l})\mathbf{h}^{l-1}(\mathbf{x}))$
    $\delta\mathbf{b}^l \leftarrow \mathrm{tmp}$
    $\delta\mathbf{W}^l \leftarrow \left(\mathrm{tmp}\, \mathbf{h}^{l-1}(\mathbf{x})^\top\right) \odot \mathbf{M}^{\mathbf{W}^l}$
    $\mathrm{tmp} \leftarrow (\mathrm{tmp}^\top(\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l}))^\top$
end for
return $p(\mathbf{x}), \delta\mathbf{b}^1, \ldots, \delta\mathbf{b}^L, \delta\mathbf{W}^1, \ldots, \delta\mathbf{W}^L, \delta\mathbf{c}, \delta\mathbf{V}$
```