Now let’s look at a fully-observable case

Autoregressive Generative Models

Simple and general idea

- Take the desired $P(\mathbf{x})$ (*e.g.,* with $\mathbf{x}$ a discrete random variable)
- Factorize it using chain rule over dimensions of $\mathbf{x}$
  - $P(\mathbf{x}) = P(\mathbf{x}^{(1)})P(\mathbf{x}^{(2)}|\mathbf{x}^{(1)})P(\mathbf{x}^{(3)}|\mathbf{x}^{(2)},\mathbf{x}^{(1)}) \dots P(\mathbf{x}^{(d)}|\mathbf{x}^{(d-1)},\dots,\mathbf{x}^{(1)})$
- Parametrize and learn each of the conditionals

In general, this still needs exponential number of parameters in d to model $P(\mathbf{x})$!

Simple and general idea

- Take the desired $P(\mathbf{x})$
- Factorize it using chain rule over dimensions of $\mathbf{x}$
  - $P(\mathbf{x}) = P(\mathbf{x}^{(1)})P(\mathbf{x}^{(2)}|\mathbf{x}^{(1)})P(\mathbf{x}^{(3)}|\mathbf{x}^{(2)}, \mathbf{x}^{(1)}) \dots P(\mathbf{x}^{(d)}|\mathbf{x}^{(d-1)}, \dots, \mathbf{x}^{(1)})$
- Parametrized and learn each of the conditionals
- For instance
  - imagine each $\mathbf{x}^{(i)}$ is a binary random variable
  - $P(\mathbf{x}^{(1)})$ needs 1 parameters, i.e. $P(\mathbf{x}^{(1)} = 0)$ and $P(\mathbf{x}^{(1)} = 1)$
  - $P(\mathbf{x}^{(2)}|\mathbf{x}^{(1)})$ needs 2 parameters, i.e. $P(\mathbf{x}^{(2)} = 0|\, \mathbf{x}^{(1)} = 0), P(\mathbf{x}^{(2)} = 0|\, \mathbf{x}^{(1)} = 1), P(\mathbf{x}^{(2)} = 1|\, \mathbf{x}^{(1)} = 0), P(\mathbf{x}^{(2)} = 1|\, \mathbf{x}^{(1)} = 1)$
  - …
  - $1 + 2 + 4 + \dots + 2^{d-1} = 2^d - 1$
  - Same goes for any discretization of a continuous random variable x (still exponential in d, with a change of base and a normalizing factor)

As usual, let’s make simplifying assumptions!

Autoregressive Generative Models

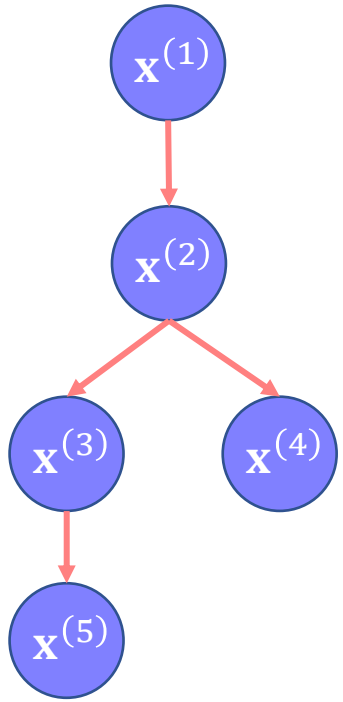
Simplification through independence assumption

$$P(\mathbf{x}) = P(\mathbf{x}^{(1)})P(\mathbf{x}^{(2)}|\mathbf{x}^{(1)})P(\mathbf{x}^{(3)}|\mathbf{x}^{(2)},\mathbf{x}^{(1)}) \dots P(\mathbf{x}^{(d)}|\mathbf{x}^{(d-1)},\dots,\mathbf{x}^{(1)})$$

$$= \prod_{i=1}^{d} P(\mathbf{x}^{(i)}|\mathbf{x}^{(<i)})$$

(directed acyclic graphical models) $$= \prod_{i=1}^{d} P\left(\mathbf{x}^{(i)}|\{\mathbf{x}^{(j)}\}_{j\in parents_i}\right)$$

(*e.g.*, Markov assumption) $$\approx \prod_{i=1}^{d} P(\mathbf{x}^{(i)}|\mathbf{x}^{(i-1)})$$



[Bengio&Bengio, "Modeling highdimensional discrete data with multi-layer neural networks", NIPS 1999]

[Frey et al. , "Does the wake-sleep algorithm learn good density estimators?", NIPS 1996]

[if interested take DD2420 Probabilistic Graphical Models]

Autoregressive Generative Models

Simplification through parameter reduction

- We want to model $P(\mathbf{x})$ through
  - $P(\mathbf{x}) = P(\mathbf{x}^{(1)})P(\mathbf{x}^{(2)}|\mathbf{x}^{(1)})P(\mathbf{x}^{(3)}|\mathbf{x}^{(2)},\mathbf{x}^{(1)}) \dots P(\mathbf{x}^{(d)}|\mathbf{x}^{(d-1)},\dots,\mathbf{x}^{(1)})$
- $P(\mathbf{x}) = P(\mathbf{x}^{(1)};\boldsymbol{\theta}_1)P(\mathbf{x}^{(2)}|\mathbf{x}^{(1)};\boldsymbol{\theta}_2)P(\mathbf{x}^{(3)}|\mathbf{x}^{(2)},\mathbf{x}^{(1)};\boldsymbol{\theta}_3) \dots P(\mathbf{x}^{(d)}|\mathbf{x}^{(d-1)},\dots,\mathbf{x}^{(1)};\boldsymbol{\theta}_d)$
- assume each of the factors $P(\mathbf{x}^{(i)}|\mathbf{x}^{(i-1)},\dots,\mathbf{x}^{(1)})$ is parametrized by $\boldsymbol{\theta}_i \in \mathbb{R}^i$
  - assume parameters increase as the conditionals become more complicated, but *only linearly*
  - example (log-linear / logistic regression for binary $\mathbf{x}$):
    - $P_{\boldsymbol{\theta}}(\mathbf{x}^{(1)} = 1) = \sigma\left(\boldsymbol{\theta}_1^{(1)}\right)$
    - $P_{\boldsymbol{\theta}}(\mathbf{x}^{(2)} = 1|\mathbf{x}^{(1)}) = \sigma\left(\boldsymbol{\theta}_2^{(1)} + \boldsymbol{\theta}_2^{(2)}\mathbf{x}^{(1)}\right)$
    - $P_{\boldsymbol{\theta}}(\mathbf{x}^{(3)} = 1|\mathbf{x}^{(1)},\mathbf{x}^{(2)}) = \sigma\left(\boldsymbol{\theta}_3^{(1)} + \boldsymbol{\theta}_3^{(2)}\mathbf{x}^{(1)} + \boldsymbol{\theta}_3^{(3)}\mathbf{x}^{(2)}\right)$

$$P(\mathbf{x}^{(i)} = 1|\mathbf{x}^{(1)},\dots,\mathbf{x}^{(i-1)};\boldsymbol{\theta}_i) = \sigma\left(\boldsymbol{\theta}_i^{(1)} + \sum_{j=2}^{i-1}\boldsymbol{\theta}_i^{(j)}\mathbf{x}^{(j-1)}\right)$$

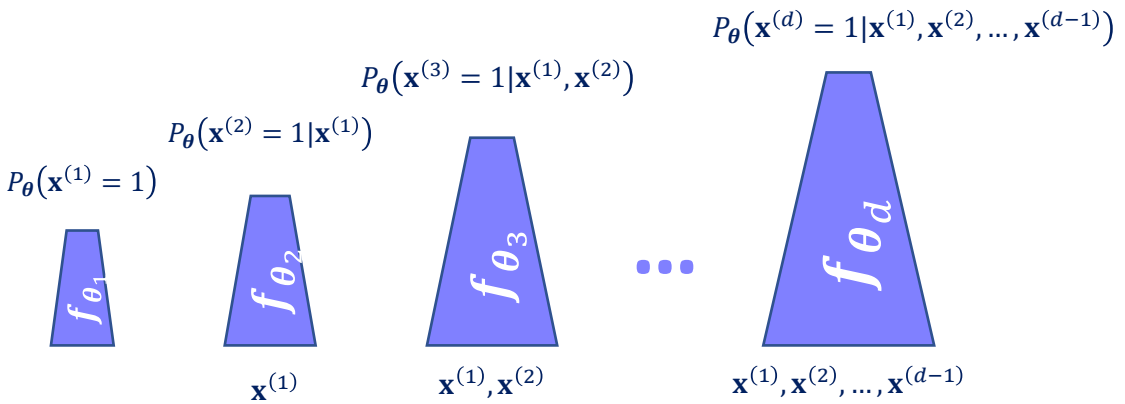
[Neal, "Connectionist learning of belief networks" AI 1992]

Log-linear functions are too restrictive though, what can we do?

Simplification through parameter reduction

- increase the representation power (neural networks)

- assume each of the factors $P\left(\mathbf{x}^{(i)}|\mathbf{x}^{(i-1)},\dots,\mathbf{x}^{(1)}\right)$ is parametrized by $\boldsymbol{\theta}_i \in \mathbb{R}^{p_i}$ of a neural network
  - parameters can remain constant or increase
  - example for binary $\mathbf{x}$:
    - $P_{\boldsymbol{\theta}}\left(\mathbf{x}^{(1)}=1\right) = \sigma\left(\boldsymbol{f}_{\boldsymbol{\theta}_1}\right)$
    - $P_{\boldsymbol{\theta}}\left(\mathbf{x}^{(2)}=1|\mathbf{x}^{(1)}\right) = \sigma\left(\boldsymbol{f}_{\boldsymbol{\theta}_2}\left(\mathbf{x}^{(1)}\right)\right)$
    - $P_{\boldsymbol{\theta}}\left(\mathbf{x}^{(3)}=1|\mathbf{x}^{(1)},\mathbf{x}^{(2)}\right) = \sigma\left(\boldsymbol{f}_{\boldsymbol{\theta}_3}\left(\mathbf{x}^{(1)},\mathbf{x}^{(2)}\right)\right)$
    - …



$$P(\mathbf{x}) = P\left(\mathbf{x}^{(1)}\right)P\left(\mathbf{x}^{(2)}|\mathbf{x}^{(1)}\right)P\left(\mathbf{x}^{(3)}|\mathbf{x}^{(2)},\mathbf{x}^{(1)}\right)\ldots P\left(\mathbf{x}^{(d)}|\mathbf{x}^{(d-1)},\ldots,\mathbf{x}^{(1)}\right)$$

Reflection/Discussion point

- How would one get a density estimate of a given $\mathbf{x}$; $P(\mathbf{x})$?
- How would one train the parameters of $P_\theta\left(\mathbf{x}^{(i)}|\mathbf{x}^{(<i)}\right)$?
- What would be the procedure to generate samples from $P(\mathbf{x})$using an autoregressive models?

5 minutes!

Autoregressive Generative Models

Generation and Density Estimation

- simple generation procedure

  This is sequential!

  - Sample $\hat{\mathbf{x}}^{(1)} \sim P\left(\mathbf{x}^{(1)}; \boldsymbol{\theta}_1\right)$
  - Sample $\hat{\mathbf{x}}^{(2)} \sim P\left(\mathbf{x}^{(2)} | \mathbf{x}^{(1)} = \hat{\mathbf{x}}^{(1)}; \boldsymbol{\theta}_2\right)$
  - …

- simple density estimation procedure

  - $P(\mathbf{x}) = P\left(\mathbf{x}^{(1)}; \boldsymbol{\theta}_1\right) P\left(\mathbf{x}^{(2)} | \mathbf{x}^{(1)}; \boldsymbol{\theta}_2\right) P\left(\mathbf{x}^{(3)} | \mathbf{x}^{(2)}, \mathbf{x}^{(1)}; \boldsymbol{\theta}_3\right) \dots P\left(\mathbf{x}^{(d)} \middle| \mathbf{x}^{(d-1)}, \dots, \mathbf{x}^{(1)}; \boldsymbol{\theta}_d\right)$

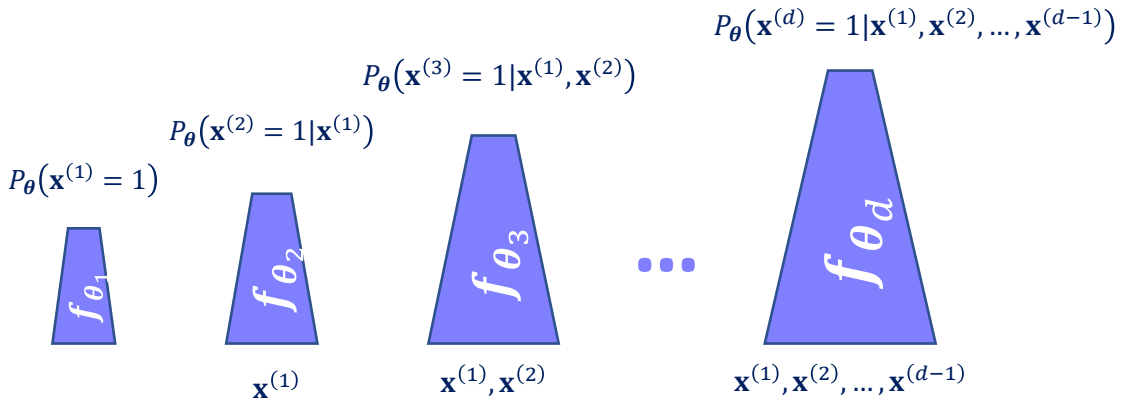
Maximum likelihood (ML) estimation of parameters

$$\max_{\boldsymbol{\theta}=\{\boldsymbol{\theta}_j\}_{j=1:d}} P_{\boldsymbol{\theta}}(\mathcal{D})$$

$$= \prod_{i=1}^{n} P_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

$$= \prod_{i=1}^{n} \prod_{j=1}^{d} P_{\boldsymbol{\theta}_j}(\mathbf{x}_i^{(j)} | \mathbf{x}_i^{(<j)})$$

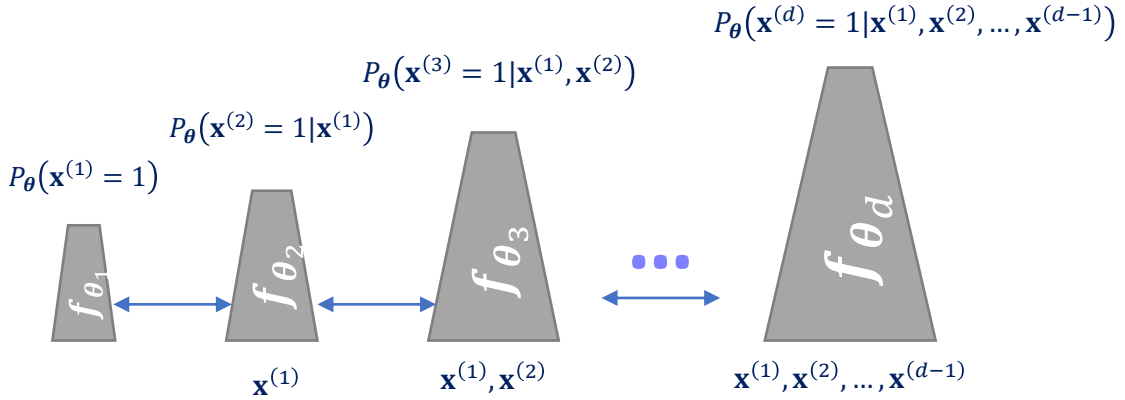
$$= \sum_{i=1}^{n} \sum_{j=1}^{d} \log P_{\boldsymbol{\theta}_j}(\mathbf{x}_i^{(j)} | \mathbf{x}_i^{(<j)})$$



However, having separate models for each dimension is in practice too expensive to train and too hard to train (incurs too many parameters and requires too-many samples)

Sharing parameters across networks

Specially the ones processing the same dimension



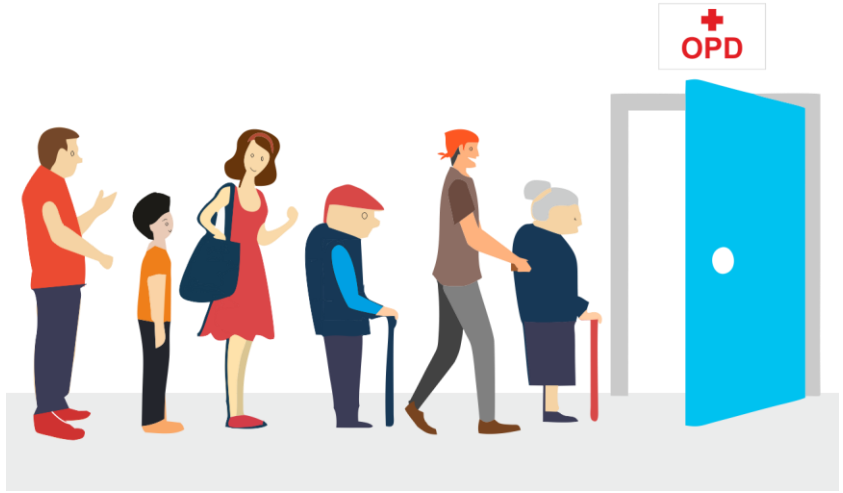
Classic methods:

[Larochelle&Murray, "The Neural Autoregressive Distribution Estimator", Aistats 2011] (NADE, sharing two-layer networks per dimension)
[Gregor&LeCun, "Learning Representations by Maximizing Compression.", arXiv 2011] (information theoretic perspective on RBM)
[Uria et al., "The real-valued neural autoregressive density-estimator", NIPS 2013] (RNADE, Recursive version)
[Uria et al., "A Deep and Tractable Density Estimator", ICML 2014] (ensemble of NADE's for different chair-rule orders)
[Uria, et al., "The Neural Autoregressive Distribution Estimation", JMLR 2016] (Journal Version)

Autoregressive – Order

Order matters!

- e.g. Ensemble of models with different orders (ENADE)



Results

- Quantitative density estimation evaluation
  - Log-likelihood of the test-set samples! (UCI 8 binary datasets)

Table 1. Average test-set log-likelihood per datapoint (in nats) of different models on eight binary datasets from the UCI repository. Baseline results were taken from Larochelle & Murray (2011).

Model	Adult	Connect4	DNA	Mushrooms	NIPS-0-12	Ocr-letters	RCV1	Web
MoBernoullis	−20.44	−23.41	−98.19	−14.46	−290.02	−40.56	−47.59	−30.16
RBM	−16.26	−22.66	−96.74	−15.15	−277.37	−43.05	−48.88	−29.38
FVSBN	**−13.17**	−12.39	**−83.64**	−10.27	−276.88	−39.30	−49.84	−29.35
NADE (fixed order)	−13.19	**−11.99**	−84.81	**−9.81**	**−273.08**	**−27.22**	**−46.66**	−28.39
NADE 1hl	−13.51	−13.04	−84.28	−10.06	−275.20	−29.05	−46.79	−28.30
NADE 2hl	−13.53	−12.99	−84.30	−10.05	−274.69	−28.92	−46.71	**−28.28**
NADE 3hl	−13.54	−13.08	−84.37	−10.10	−274.86	−28.89	−46.76	−28.29
EoNADE 1hl (2 ord)	−13.35	−12.81	−83.52	−9.88	−274.12	−28.36	−46.50	−28.11
EoNADE 1hl (16 ord)	−13.19	−12.58	**−82.31**	**−9.68**	**−272.38**	−27.31	**−46.12**	**−27.87**

[Uria et al., "A Deep and Tractable Density Estimator", ICML 2014]

Results

- Quantitative density estimation evaluation on images
  - Log-likelihood of the test-set samples (can be done on images, *e.g.*, MNIST)

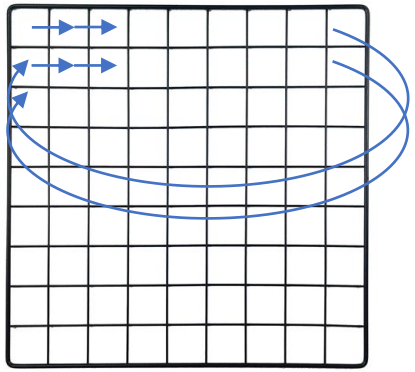


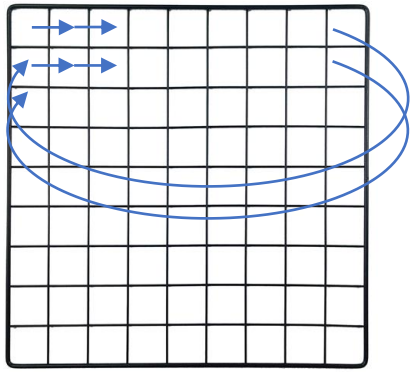
Table 2. Average test-set log-likelihood per datapoint of different models on 28×28 binarized images of digits taken from MNIST.

Model	Test LogL
MoBernoullis K=10	−168.95
MoBernoullis K=500	−137.64
RBM (500 h, 25 CD steps) approx.	−86.34
DBN 2hl approx.	**−84.55**
NADE 1hl (fixed order)	−88.86
NADE 1hl (fixed order, RLU, minibatch)	−88.33
NADE 1hl (fixed order, sigm, minibatch)	−88.35
NADE 1hl (no input masks)	−99.37
NADE 2hl (no input masks)	−95.33
NADE 1hl	−92.17
NADE 2hl	−89.17
NADE 3hl	−89.38
NADE 4hl	−89.60
EoNADE 1hl (2 orderings)	−90.69
EoNADE 1hl (128 orderings)	−87.71
EoNADE 2hl (2 orderings)	−87.96
EoNADE 2hl (128 orderings)	−85.10

[Uria et al., "A Deep and Tractable Density Estimator", ICML 2014]

Results

- Qualitative generation results
  - MNIST



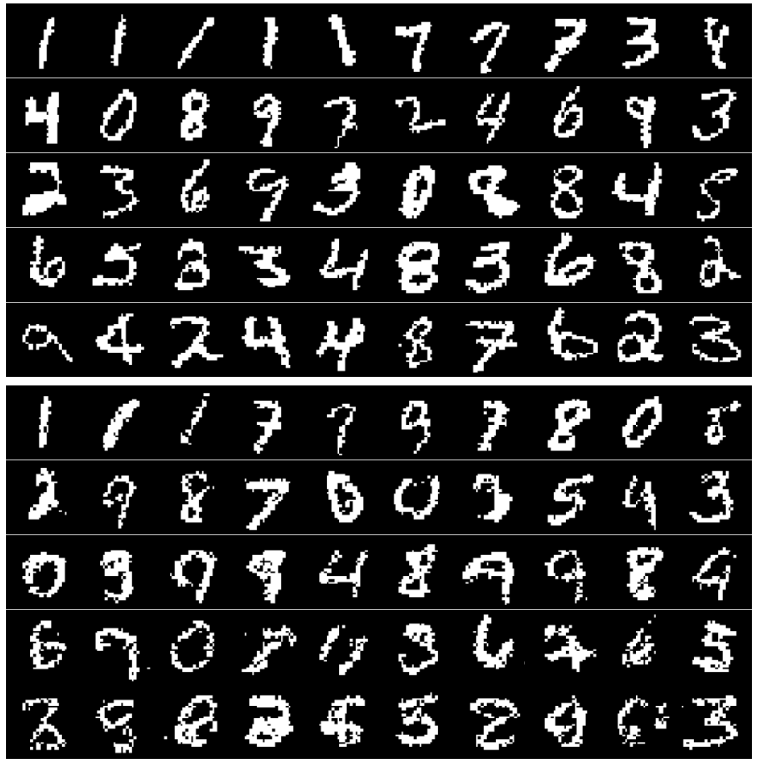
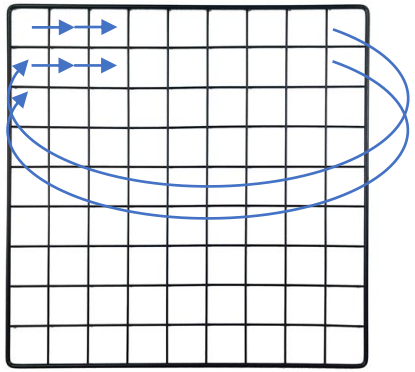


Figure 2. **Top:** 50 examples from binarized-MNIST ordered by decreasing likelihood under a 2-hidden-layer NADE. **Bottom:** 50 samples from a 2-hidden-layer NADE, also ordered by decreasing likelihood under the model.

[Uria et al., "A Deep and Tractable Density Estimator", ICML 2014]

Results

- Qualitative data imputation
  - You can fill in the missing values in a very simple manner! (MNIST)



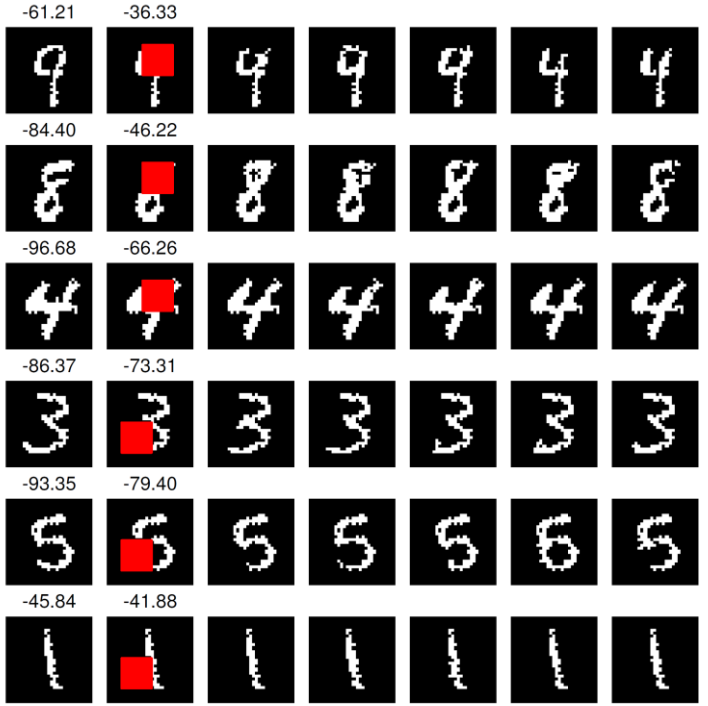


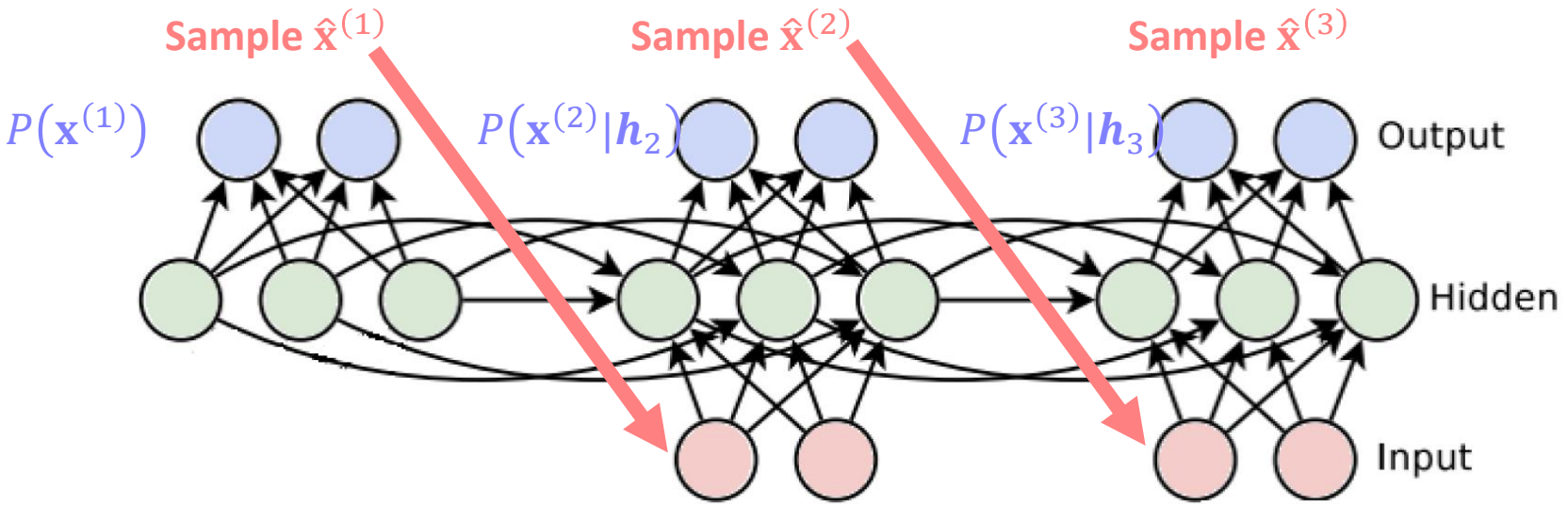
Figure 3. Example of marginalization and sampling. First column shows five examples from the test set of the MNIST dataset. The second column shows the density of these examples when a random 10 by 10 pixel region is marginalized. The right-most five columns show samples for the hollowed region. Both tasks can be done easily with a NADE where the pixels to marginalize are at the end of the ordering.

[Uria et al., "A Deep and Tractable Density Estimator", ICML 2014]

More modern deep autoregressive models exist too!

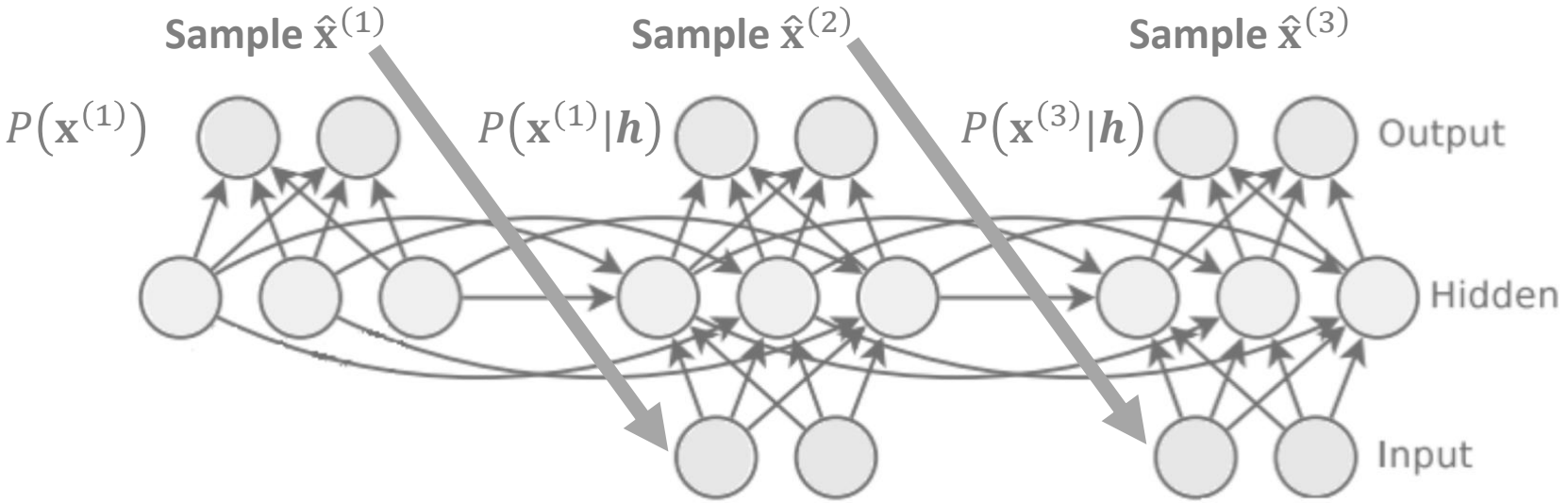
Autoregressive Generative Models

RNN can be used as an autoregressive generator!



Autoregressive - RNN

RNN can be used as an autoregressive generator!



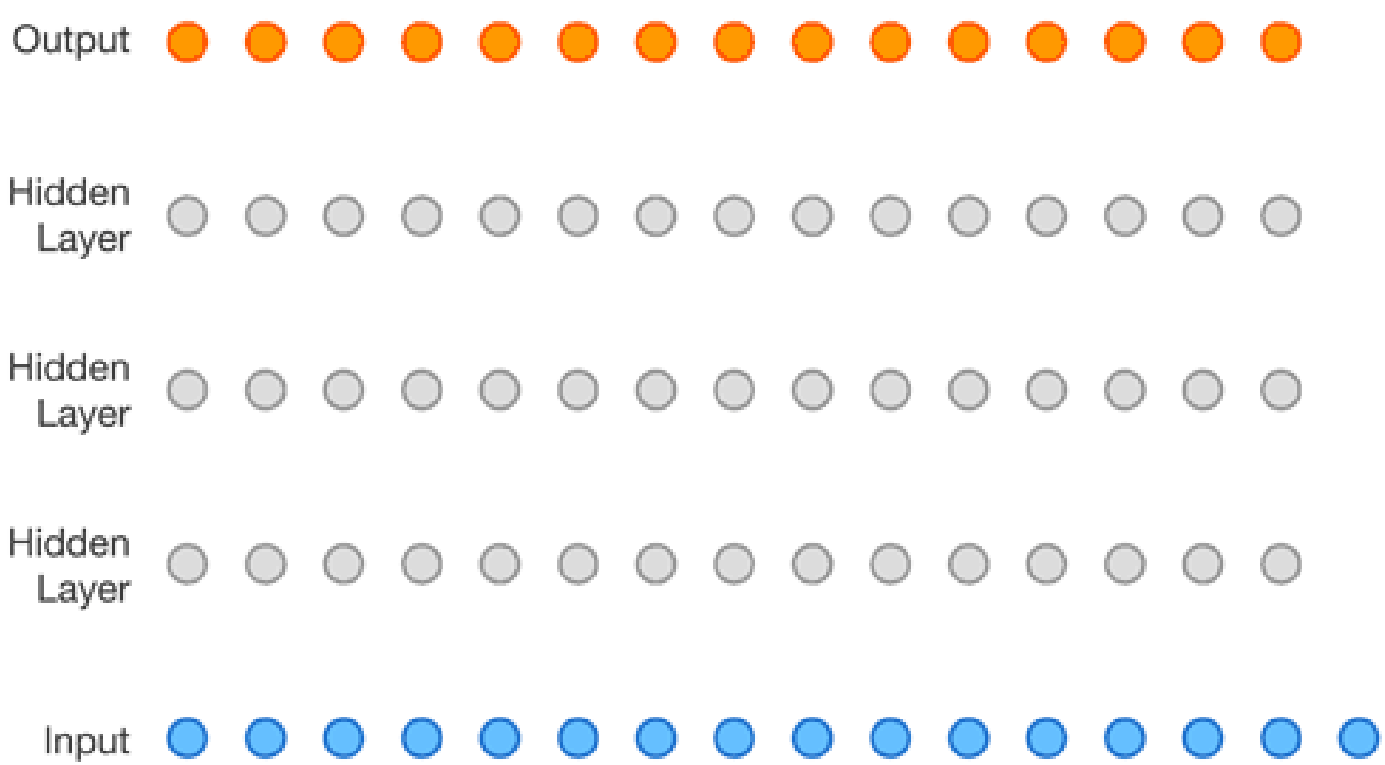
Pros

- ✓ Sequence of arbitrary length
- ✓ Can cast any generation process and increase the modeling capacity as we like

Cons

- ❖ Slow training due to sequential likelihood evaluation
- ❖ Vanishing/exploding gradients (but one can use LSTM/GRU)

Autoregressive deep models



[source: DeepMind: https://deepmind.com/blog/wavenet-generative-model-raw-audio/]

[Oord et al. "WaveNet: A Generative Model for Raw Audio", arXiv 2016]

Autoregressive - Applications

- WaveNet
  - https://www.deepmind.com/blog/wavenet-a-generative-model-for-raw-audio

ENGLISH

Google HMM

Google RNN/LSTM

Google WaveNet

MANDARIN

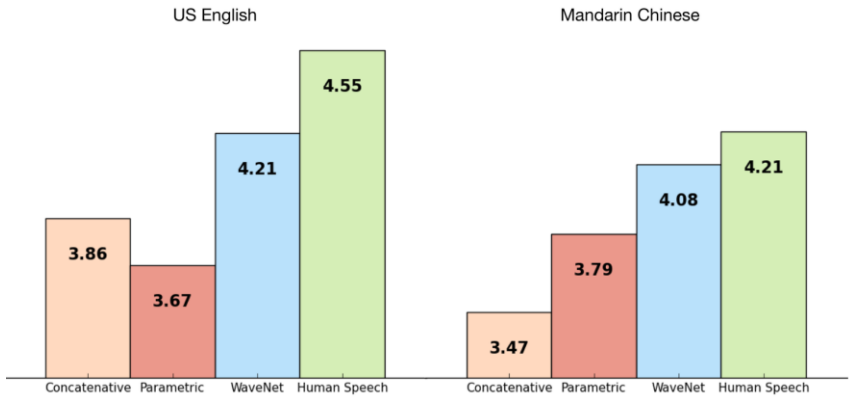
Google HMM

Google RNN/LSTM

Google WaveNet

MUSIC

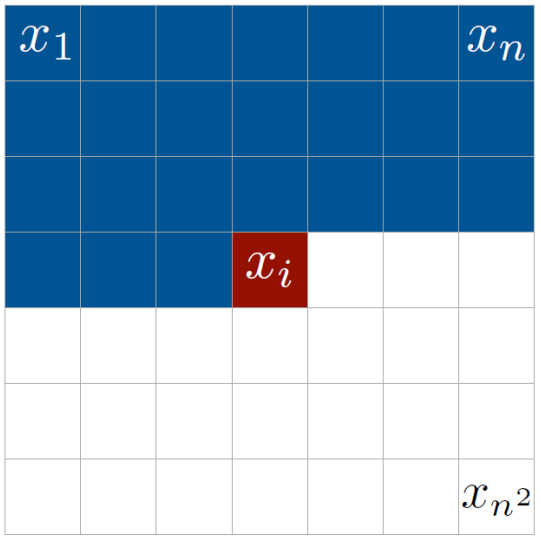
Google WaveNet



[Oord et al. "WaveNet: A Generative Model for Raw Audio", arXiv 2016]

Modern image models

$$P(\mathbf{x}) = \prod_{i=1}^{n^2} P(\mathbf{x}^{(i)} | \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(i-1)})$$

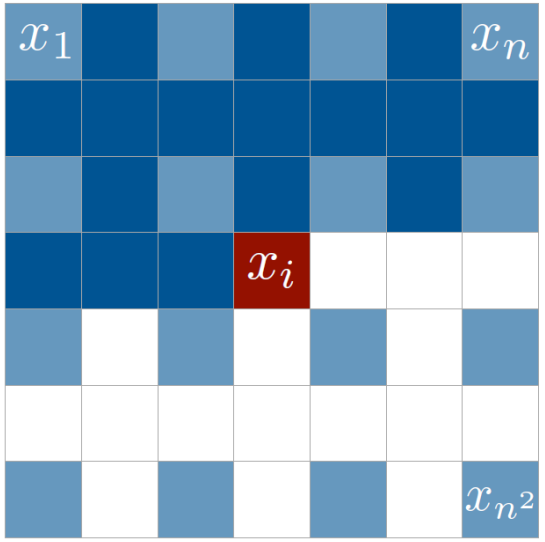


[Oord, Kalchbrenner, Kavukcuoglu, "Pixel Recurrent Neural Networks ", JMLR 2016]

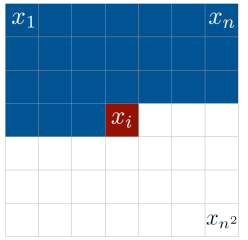
Modern image models

$$P(\mathbf{x}) = \prod_{i=1}^{n^2} P\left(\mathbf{x}^{(i)} \middle| \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(i-1)}, \mathbf{x}'\right)$$

$\mathbf{x} \in \mathbb{R}^{n^2}$, $\mathbf{x}' \in \mathbb{R}^{\frac{n^2}{4}}$



Multi-scale context



Context

[Oord, Kalchbrenner, Kavukcuoglu, "Pixel Recurrent Neural Networks ", JMLR 2016]

Modern image models

- As mentioned, RNN for Autoregressive generation is slow to train

- PixelCNN
  - You have to also apply chain rule on the 3 RGB channels
  - No pooling is used in CNN, so resolution is preserved
  - At each layer the receptive field is bounded
  - Use masks to follow the necessary conditional independence

  - Efficient Training
  - Slow generation
  - Quality not as high as PixelRNN (due to incomplete dependence)

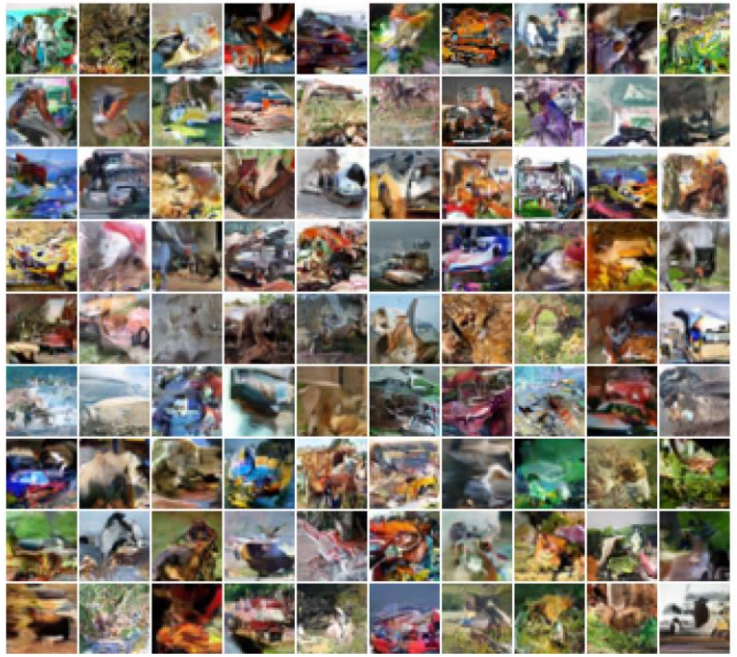
[Oord, Kalchbrenner, Kavukcuoglu, “Pixel Recurrent Neural Networks “, JMLR 2016]

Autoregressive - PixelCNN

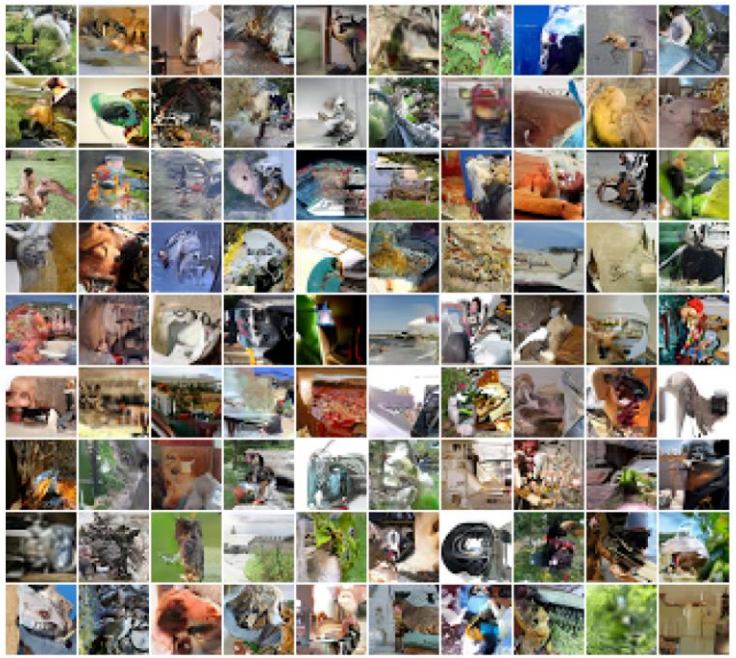
Results

- qualitative results for generative modeling

CIFAR-10



ImageNet



[Oord, Kalchbrenner, Kavukcuoglu, “Pixel Recurrent Neural Networks “, JMLR 2016]

Autoregressive - PixelCNN

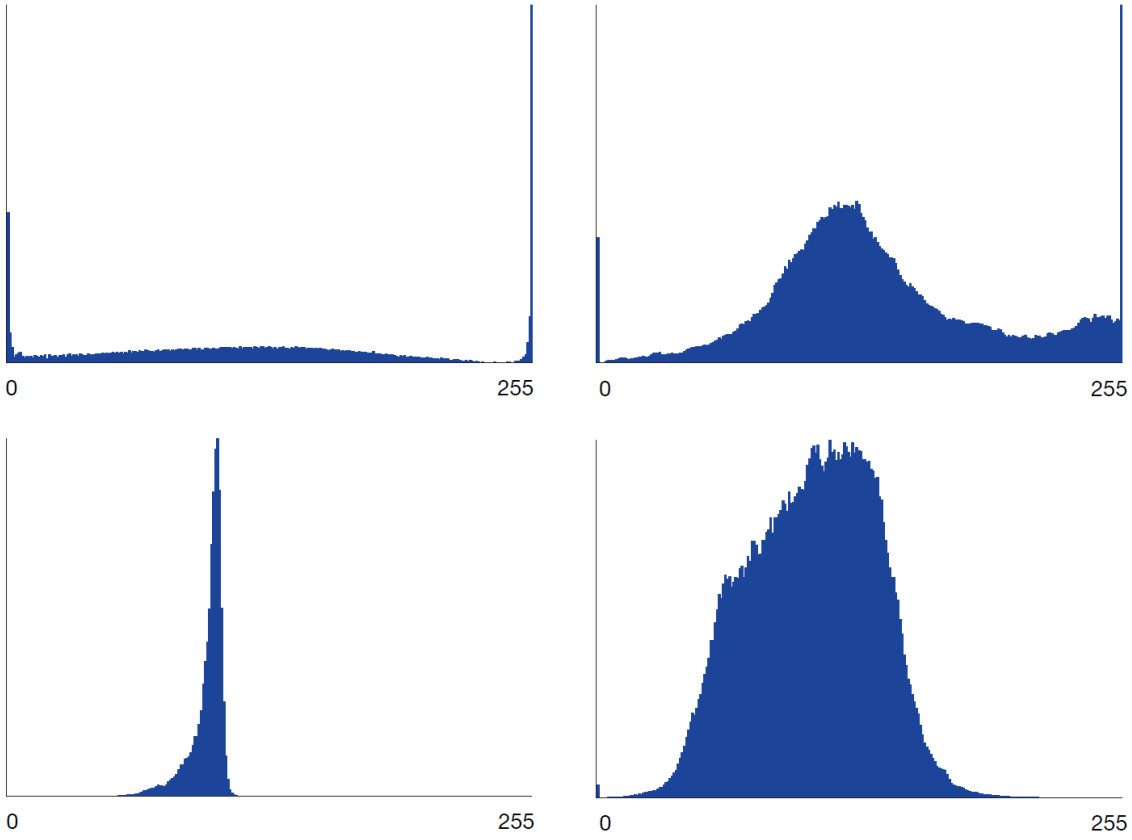
Results

- data imputation (ImageNet 32x32)

Autoregressive - PixelCNN

Results

- conditional distribution of different pixels using discrete variables and softmax (instead of continuous parametric distributions)



[Oord, Kalchbrenner, Kavukcuoglu, "Pixel Recurrent Neural Networks ", JMLR 2016]

Autoregressive – PixelCNN

- Gated PixelCNN
  - [Oord, et al. “Conditional Image Generation with PixelCNN Decoders”, NIPS 2016]
    - different activation function: similar to LSTM
    - stacks of convolutions: to increase the effective receptive field to all previous pixels
- PixelCNN++
  - [Salimans, Karpathy, Chen, Kingma, “PIXELCNN++: IMPROVING THE PIXELCNN WITH DISCRETIZED LOGISTIC MIXTURE LIKELIHOOD AND OTHER MODIFICATIONS”, ICLR 2017]
    - Better loss function
    - Better architecture
    - Dropout
- PixelGAN
  - [Makhzani, Frey, “PixelGAN Autoencoders”, NIPS 2017]
    - PixelCNN as generator and GAN-like training for the encoder (recognition) network
- PixelVAE
  - [Gulrajani,…, Courville. “PixelVAE: A Latent Variable Model for Natural Images”, ICLR 2017]

Summary

- Simple generation process
- Exact and simple density estimation
- Very good for data imputation
- No encoding
- Slow training, sample generation, and density estimation (due to the sequential nature)