Deep Generative Models and a Probabilistic Programming Library

Jun Zhu

dcszj@mail.tsinghua.edu.cn

Department of Computer Science and Technology

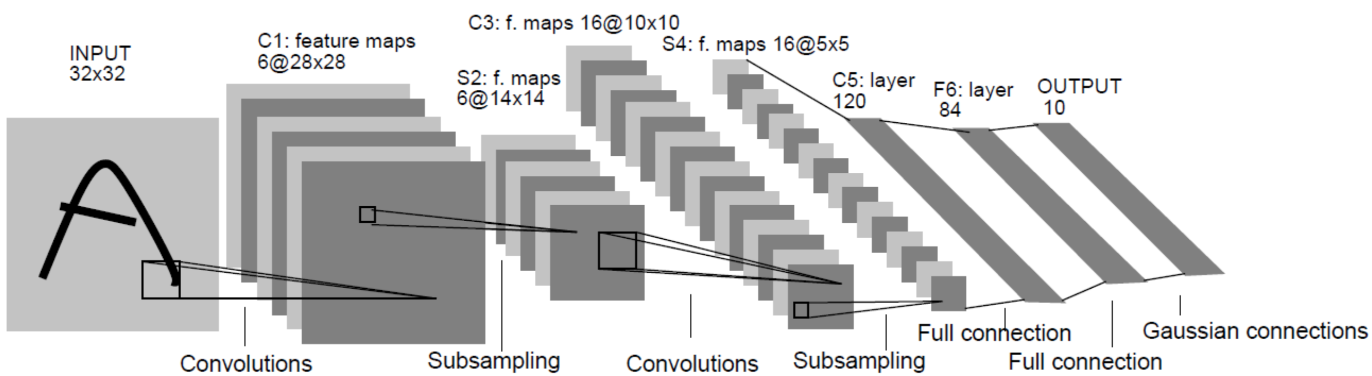
Tsinghua University

VALSE, Dalian, 2018

Discriminative (Deep) Learning

- Learn a (differentiable) function mapping from input to output

$$x \longrightarrow \boxed{f(x;\theta)} \longrightarrow y$$



  - Gradient back-propagation

Generative Modeling

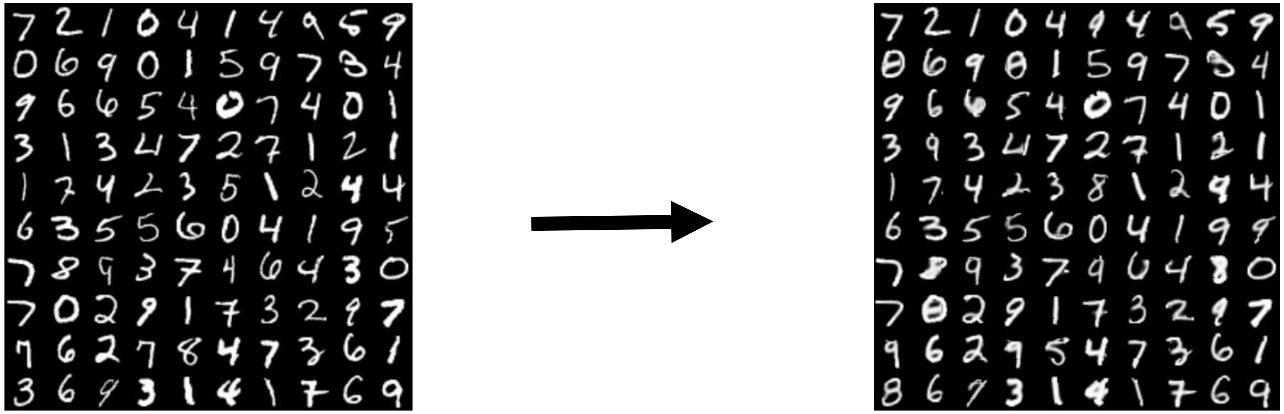
- Have training examples

$$\mathrm{x} \sim p_{\mathrm{data}}(\mathrm{x})$$

- Want a model that can draw samples:

$$\mathrm{x}' \sim p_{\mathrm{model}}(\mathrm{x})$$

  - where $p_{\mathrm{model}}(\mathrm{x}) \approx p_{\mathrm{data}}(\mathrm{x})$



$\mathrm{x} \sim p_{\mathrm{data}}(\mathrm{x})$ $\qquad$ $\mathrm{x}' \sim p_{\mathrm{model}}(\mathrm{x})$

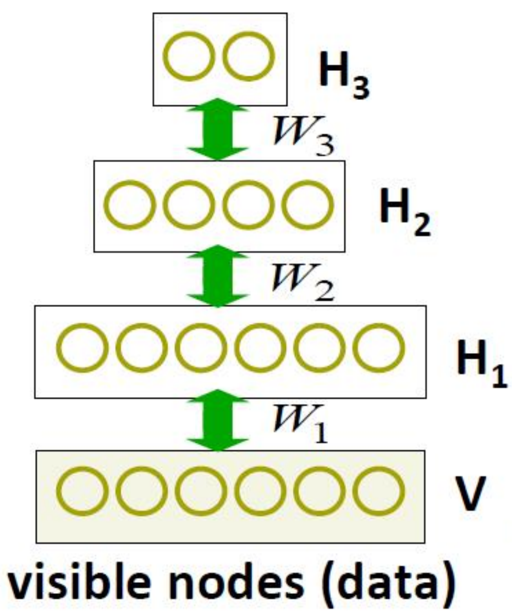
Why generative models?

- Leverage unlabeled datasets, which are often much larger than labeled ones
  - Unsupervised learning
    - e.g., clustering, density estimation, feature extraction, dimension reduction, data generation
  - Semi-supervised learning
    - e.g., classification, information extraction, learning-to-rank, network analysis, opinion mining

- Conditional generative models
  - Speech synthesis: Text ⇒ Speech
  - Machine Translation: French ⇒ English
  - Image captioning: Image ⇒ Text

Generative models are everywhere …

- (hierarchical) Language models for text
- Gaussian mixture models for clustering/density estimation
- Probabilistic PCA/FA/ICA for dimension reduction
- Probabilistic matrix factorization (PMF) for recommendation
- Deep belief networks (Hinton et al., 2016)



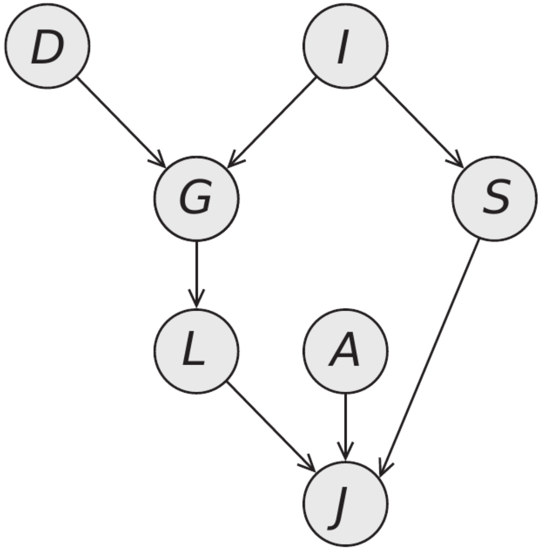
Two ways to build deep generative models

- **Traditional one**
  - Hierarchical Bayesian methods

- **More modern one**
  - Deep generative models

Hierarchical Bayesian Modeling

- Build a hierarchy throng distributions in analytical forms



$$P(D, I, G, S, L, A, J) = P(D)P(I)P(G|D, I)P(S|I) \cdot P(A)P(L|G)P(J|L, A, S)$$

Simple, Local Factors: a conditional probability distribution

Deep Generative Models

- **More flexible by using differential function mapping between random variables**

- If z is uniformly distributed over (0, 1), then $y = f(z)$ has the distribution

$$p(y) = p(z)\left|\frac{dz}{dy}\right|$$

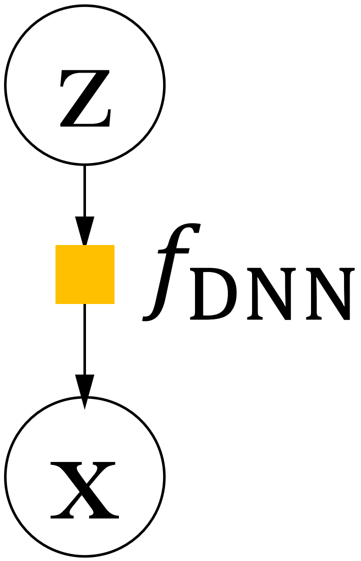
  - where $p(z) = 1$

- This trick is widely used to draw samples from exponential family distributions (e.g., Gaussian, Exponential)

Deep Generative Models

- **More flexible by using differential function mapping between random variables**

- DGMs learn a function transform with deep neural networks



Cause => Disease

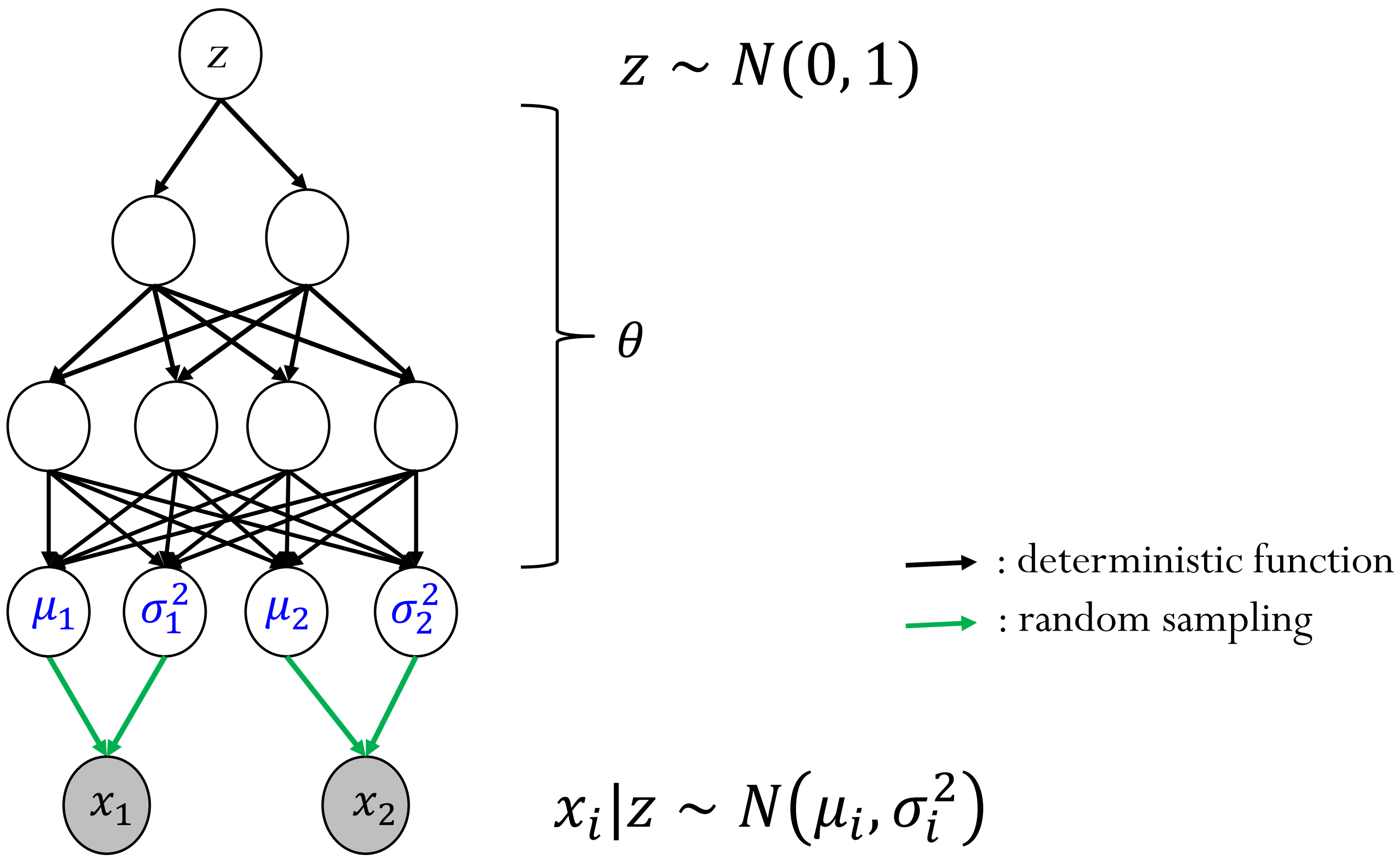
Topics => Docs

Objects => Images

Words => Phonemes

An example with MLP

- 1D latent variable z; 2D observation x
- **Idea**: NN + Gaussian (or Bernoulli) with a diagonal covariance

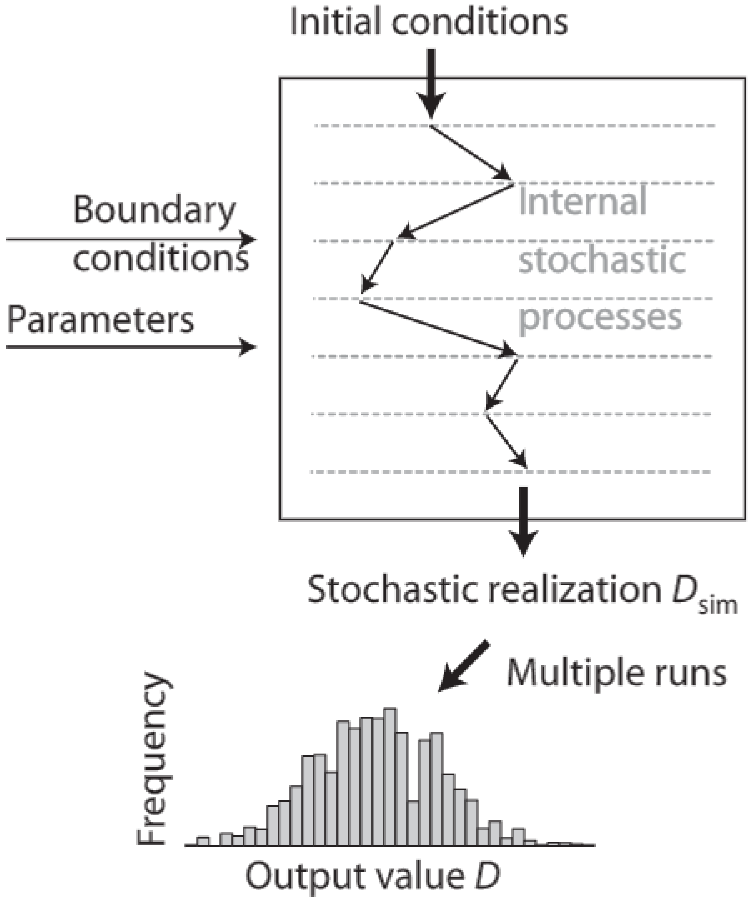


$z \sim N(0, 1)$

$x_i | z \sim N\left(\mu_i, \sigma_i^2\right)$

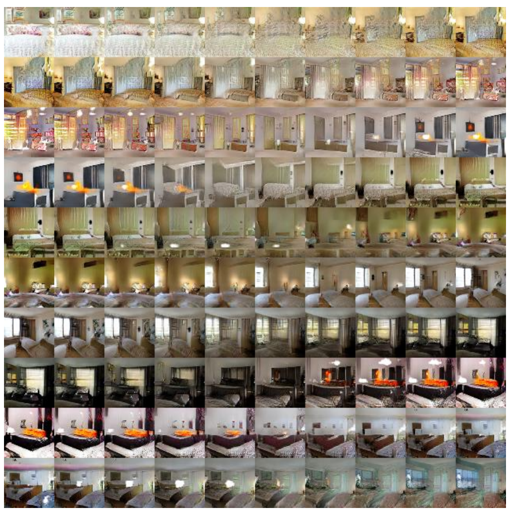
Implicit Deep Generative Models

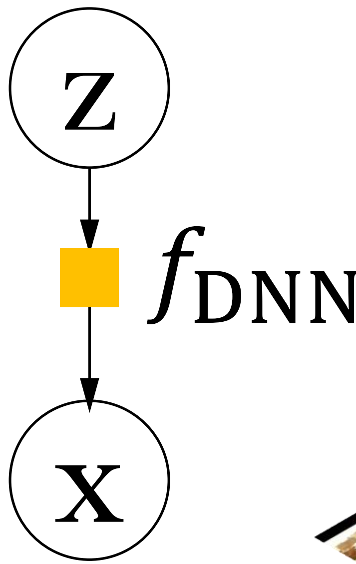
- Generate data with a stochastic process whose likelihood function is not explicitly specified (Hartig et al., 2011)

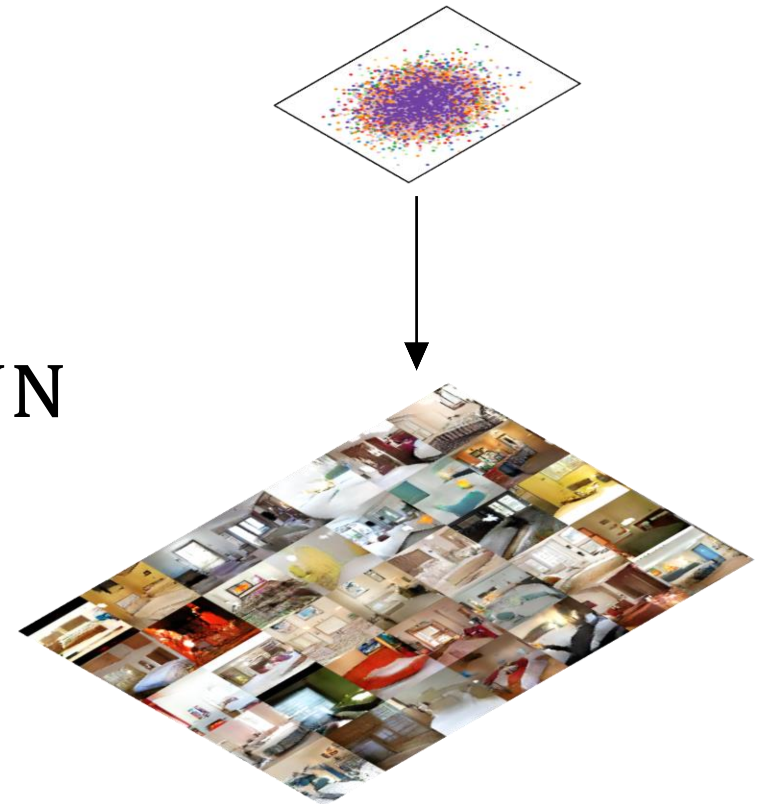


Deep Generative Models

[Image Generation:
Generative Adversarial Nets,
Goodfellow13 & Radford15]



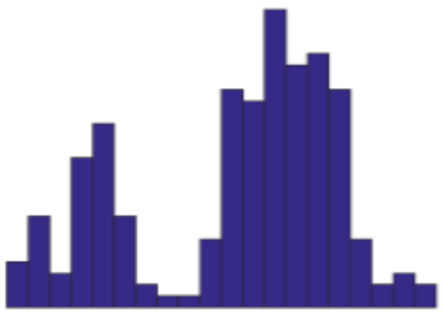




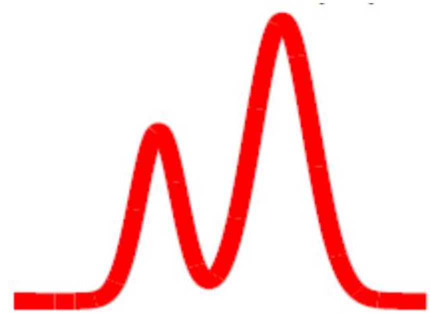
Learning Deep Generative Models

- **Given** a set D of unlabeled samples, learn the unknown parameters (or a distribution)

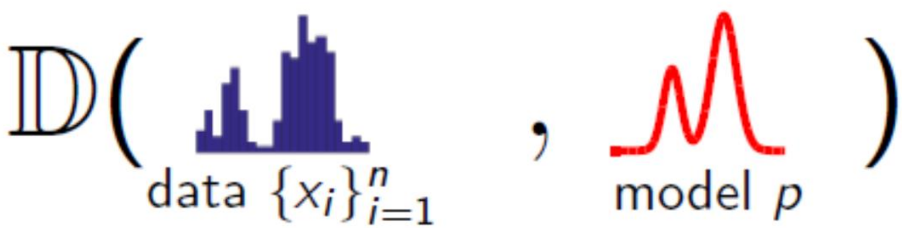
data $D=\{\mathrm{x_i}\}$



models $p(\mathbf{x}|\theta)$



- Find a model that minimizes



Learning Deep Generative Models

- Maximum likelihood estimation (MLE):

$$\hat{\theta} = \operatorname{argmax} p(D|\theta)$$

  - has an explicit likelihood model
- Minimax objective (e.g., GAN)
  - a two-player game to reach equilibrium
- Moment-matching:
  - draw samples from p: $\widehat{D} = \{\mathrm{y}_i\}_{i=1}^{M}, \text{where } \mathrm{y}_i \sim p(\mathrm{x}|\theta)$
  - Kernel MMD: $$\mathcal{L}_{MMD^2} = \left\| \frac{1}{N} \sum_{i=1}^{N} \phi(x_i) - \frac{1}{M} \sum_{j=1}^{M} \phi(y_j) \right\|_{\mathcal{H}}^2$$
    - rich enough to distinguish any two distributions in certain RKHS

Related Work at VALSE 2018

- Duplex GAN for Domain Adaptation (DGM)
  - Hu, Kan, Shan, Chen
- Multimodal Possion Gamma Belief Networks (DGM)
  - Wang, Chen, Zhou
- WHAI: Weibull Hybrid Autoencoding Inference for Deep Topic Models (DGM)
  - Zhang, Chen, Guo, Zhou
- The Assumed Parameter Filter (probabilistic programming)
  - Erol, Wu, Li, Russell
- …

Our work on DGMs

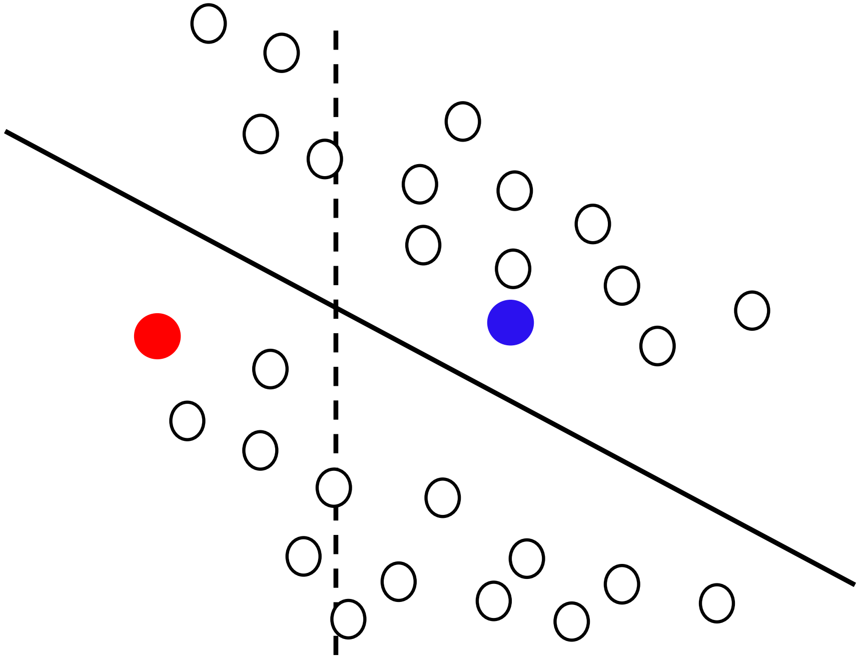
- **Learning algorithms and theories**
  - Max-margin variational auto-encoder (Li et al., NIPS 2015)
  - Learning to generate with memory (Li et al., ICML 2016)
  - Conditional moment-matching generative networks (Ren et al., NIPS 2016)
  - Population matching discrepancy (Chen et al., NIPS 2017)
  - Implicit variational inference (Shi et al., ICLR 2018)
- **Semi-supervised learning & Style transfer**
  - Max-margin variational auto-encoder for SSL (Li et al., PAMI 2017)
  - Triple generative adversarial networks (Li et al., NIPS 2017)
  - Structured generative adversarial networks (Deng et al., NIPS 2017)
  - Learning to write styled Chinese characters by reading a handful of examples (Sun et al., IJCAI 2018)
  - Smooth neighbors for SSL (Luo et al., CVPR 2018)
- **Programming library**
  - ZhuSuan (Shi et al., arXiv 2017)

Outline

- Deep Generative Models
- Semi-supervised Learning & Style transfer
- ZhuSuan: a Probabilistic Programming library
- Conclusions & QA

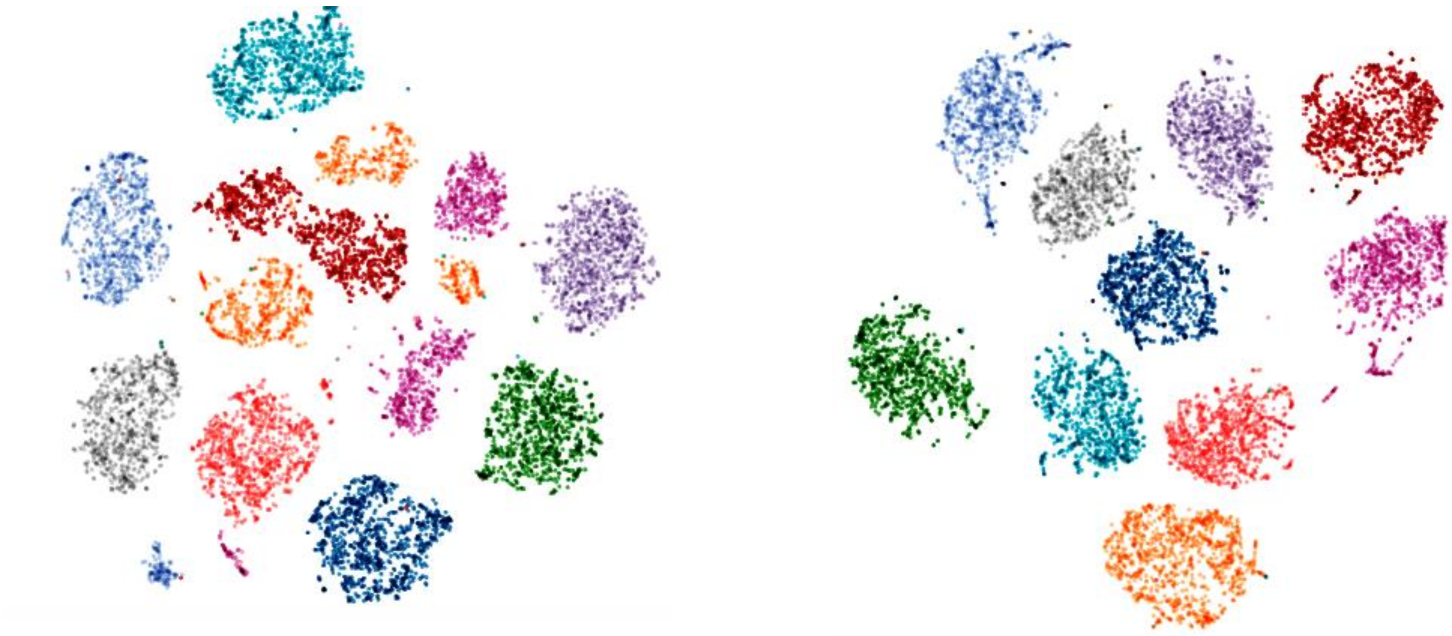
Semi-supervised Learning

- A toy example



Representation Matters

- t-SNE embedding of learned representations by different DGM models on CIFAR10



Triple Generative Adversarial Nets

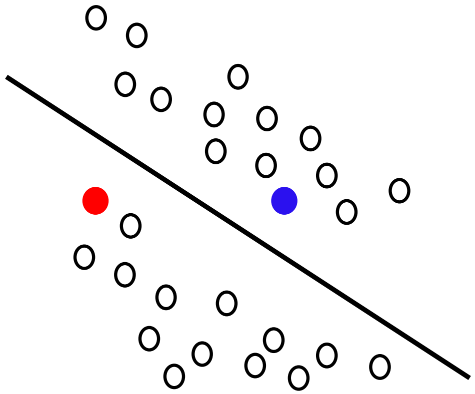
- A minimax game for semi-supervised learning
  - GAN is for unsupervised learning $p_{\text{model}}(\mathrm{x}) = p_{\text{data}}(\mathrm{x})$
  - We aim to learn the joint distribution $p_{\text{model}}(\mathrm{x}, \mathrm{y}) = p_{\text{data}}(\mathrm{x}, \mathrm{y})$

- A simple insight
  - factorization form with conditionals

A classifier

$$
\begin{aligned}
p(x, y) &= p(x)p(y|x) \\
&= p(y)p(x|y)
\end{aligned}
$$

A class-conditional generator

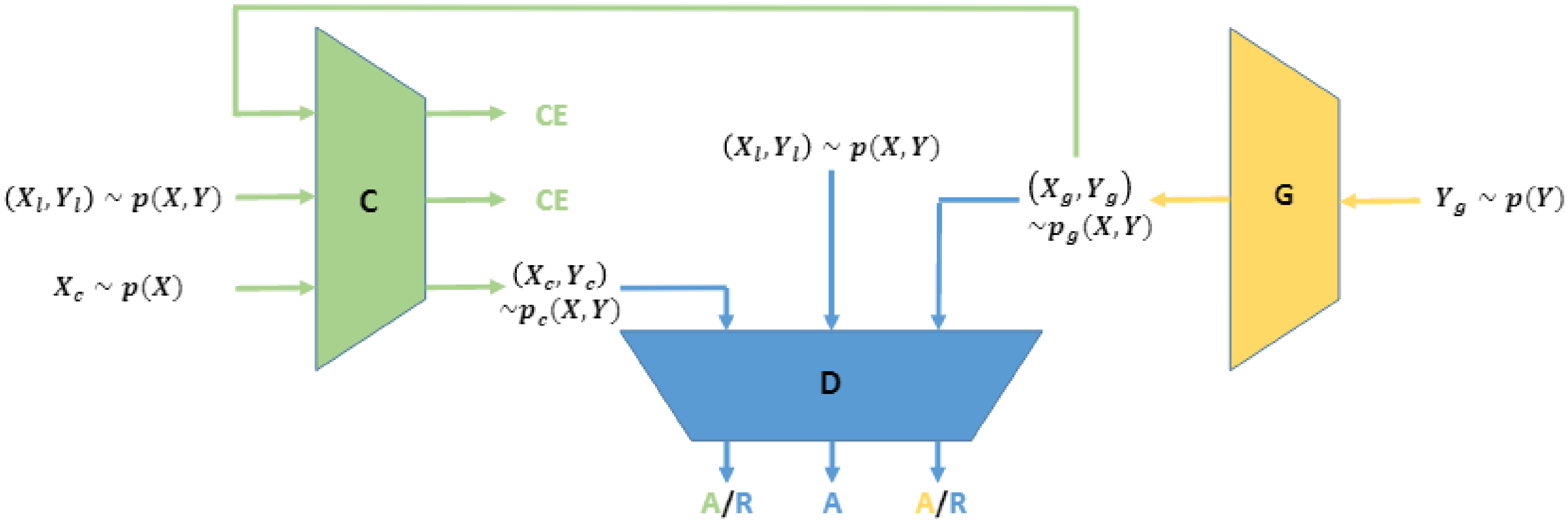


- We need three players
  - Two generators to generate (x,y)
  - A discriminator to distinguish fake (x,y)

[Li et al., NIPS 2017]

Triple-GAN

- The network architecture



- Both C and G are generators
- D is the discriminator
- CE: cross-entropy loss for learning classifier

[Li et al., NIPS 2017]

A minimax game

- The optimization problem

$$\min_{C,G}\max_{D} U(C, G, D) = E_p[\log D(x, y)] + \alpha E_{p_c}[\log(1 - D(x, y))] + (1 - \alpha) E_{p_g}[\log(1 - D(x, y))]$$

  - The hyper-parameter *a* is often set at 1/2

  - The standard supervised loss can be incorporated

$$\min_{C,G}\max_{D} \tilde{U}(C, G, D) = U(C, G, D) + E_p[-\log p_c(y|x)]$$

[Li et al., NIPS 2017]

Major theoretical results

Theorem

The equilibrium of $\tilde{U}(C, G, D)$ *is achieved if and only if* $p(x, y) = p_g(x, y) = p_c(x, y)$ *with* $D^*_{C,G}(x, y) = \frac{1}{2}$ *and the optimum value is* $-\log 4$.

Lemma

For any fixed C and G, the optimal discriminator D is:

$$D^*_{C,G}(x, y) = \frac{p(x, y)}{p(x, y) + p_\alpha(x, y)},$$

where $p_\alpha(x, y) := (1 - \alpha)p_g(x, y) + \alpha p_c(x, y)$.

Some Practical Tricks for SSL

- Pseudo discriminative loss: using $(x, y) \sim p_g(x, y)$ as labeled data to train C
  - Explicit loss, equivalent to $KL(p_g(x, y) || p_c(x, y))$
  - Complementary to the implicit regularization by D
- Collapsing to the empirical distribution $p(x, y)$
  - Sample $(x, y) \sim p_c(x, y)$ as true data for D
  - Biased solution: target shifting towards $p_c(x, y)$
- Unlabeled data loss on C
  - Confidence (Springenberg [2015])
  - Consistence (Laine and Aila [2016])

Some Results

- Semi-supervised classification

Table 1: Error rates (%) on partially labeled MNIST, SHVN and CIFAR10 datasets. The results with † are trained with more than 500,000 extra unlabeled data on SVHN.

Algorithm	MNIST $n = 100$	SVHN $n = 1000$	CIFAR10 $n = 4000$
M1+M2 [11]	3.33 ($\pm$0.14)	36.02 ($\pm$0.10)	
VAT [18]	2.33		24.63
Ladder [23]	1.06 ($\pm$0.37)		20.40 ($\pm$0.47)
Conv-Ladder [23]	**0.89** ($\pm$0.50)		
ADGM [17]	0.96 ($\pm$0.02)	22.86 †	
SDGM [17]	1.32 ($\pm$0.07)	16.61($\pm$0.24)†	
MMCVA [15]	1.24 ($\pm$0.54)	**4.95** ($\pm$0.18) †	
CatGAN [26]	1.39 ($\pm$0.28)		19.58 ($\pm$0.58)
Improved-GAN [25]	0.93 ($\pm$0.07)	8.11 ($\pm$1.3)	18.63 ($\pm$2.32)
ALI [5]		7.3	18.3
Triple-GAN **(ours)**	**0.91** ($\pm$0.58)	**5.77**($\pm$0.17)	**16.99** ($\pm$0.36)

Some Results

- Class-conditional generation

(d) Dog (e) Horse (f) Ship

Some Results

- Disentangle class and style

Figure: Same y for each row. Same z for each column.

Some Results

- Latent space interpolation on MNIST

Some Results

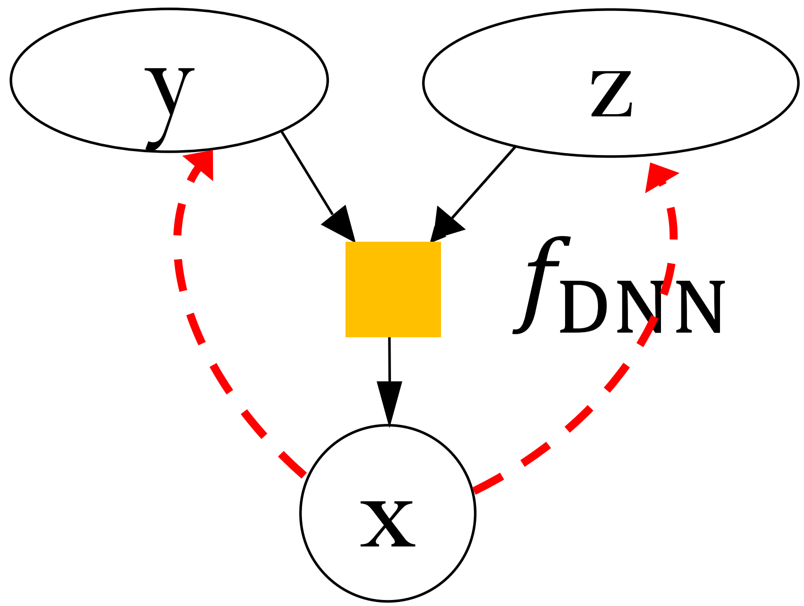
- Latent space interpolation on SVHN

Some Results

- Latent space interpolation on CIFAR10

Structured GAN

- Structural extensions to Triple-GAN
  - Learn joint distribution $p(\mathrm{x}, \mathrm{y}, \mathrm{z})$
  - Ability to infer the posterior $p(\mathrm{y}, \mathrm{z} | \mathrm{x})$, $p(\mathrm{y} | \mathrm{x})$, $p(\mathrm{z} | \mathrm{x})$
  - Structural bias to disentangle latent factors y and z by information regularization on $p(\mathrm{y} | \mathrm{x})$ and $p(\mathrm{x} | \mathrm{x})$



[Deng et al., NIPS 2017]

Structured GAN

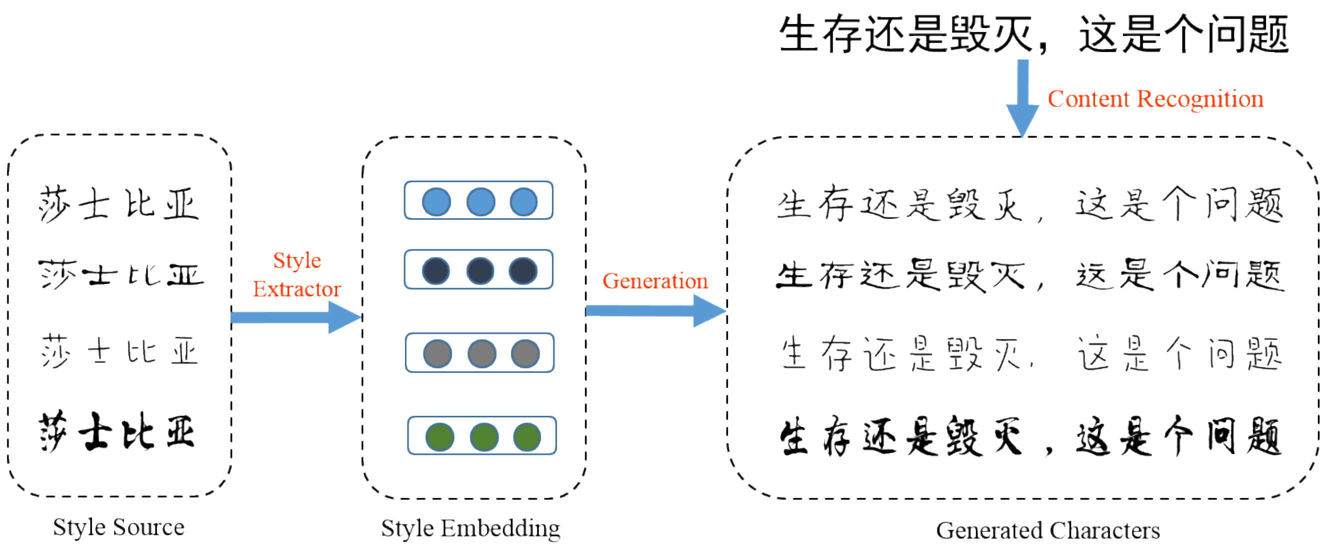
- Produces even better results than Triple-GAN

Method	MNIST			SVHN	CIFAR-10
	$n=20$	$n=50$	$n=100$	$n=1000$	$n=4000$
Ladder [22]	-	-	**0.89(±0.50)**	-	20.40(±0.47)
VAE [12]	-	-	3.33(±0.14)	36.02(±0.10)	-
CatGAN [28]	-	-	1.39(±0.28)	-	19.58(±0.58)
ALI [5]	-	-	-	7.3	18.3
ImprovedGAN [27]	16.77(±4.52)	2.21(±1.36)	0.93 (±0.07)	8.11(±1.3)	18.63(±2.32)
TripleGAN [15]	5.40(±6.53)	1.59(±0.69)	0.92(±0.58)	5.83(±0.20)	18.82(±0.32)
SGAN	**4.0(±4.14)**	**1.29(±0.47)**	**0.89(±0.11)**	**5.73(±0.12)**	**17.26(±0.69)**

- inception score： 6.91(±0.07)
  - TripleGAN： 5.08(±0.09)
  - Improved-GAN： 3.87(±0.03)

Style Transfer for Chinese Characters

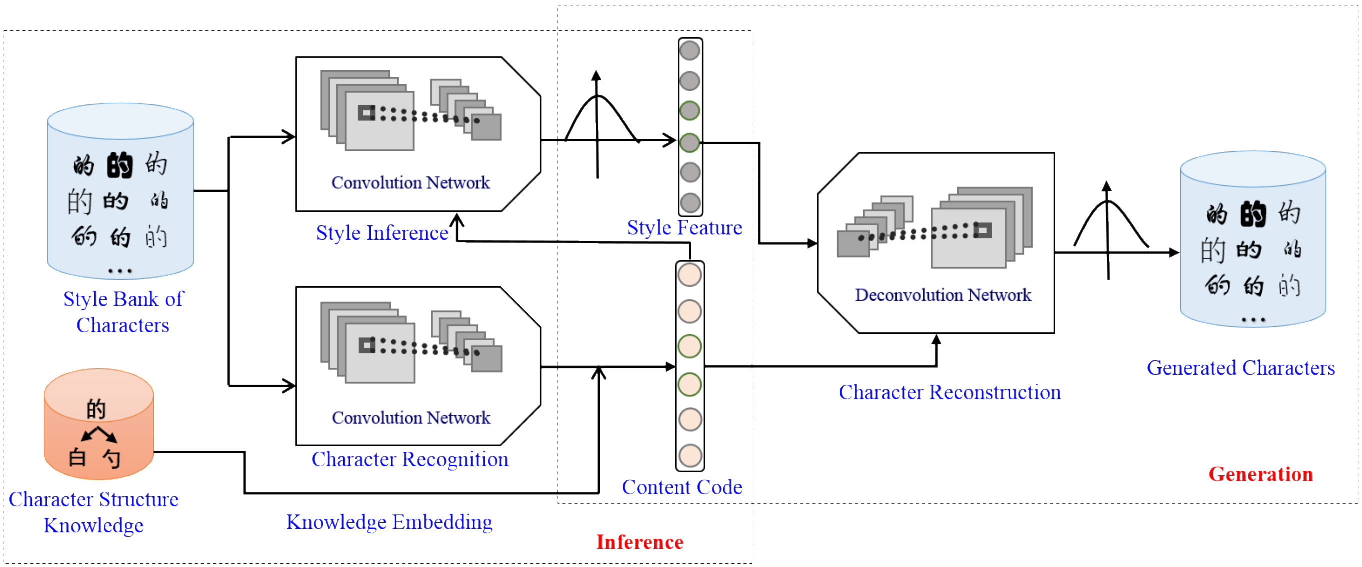
- Build a generative model for Chinese characters generation and writing style transfer tasks

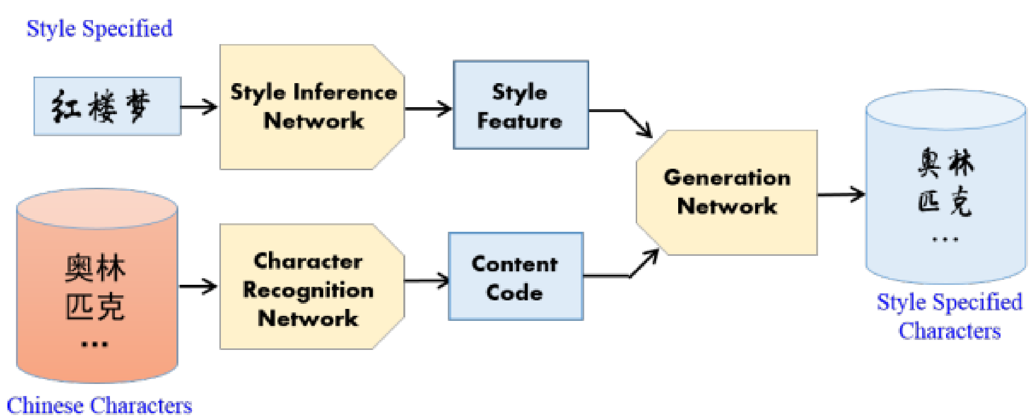


One-shot Generalization　　　　Disentanglement

[Sun et al., IJCAI 2018]

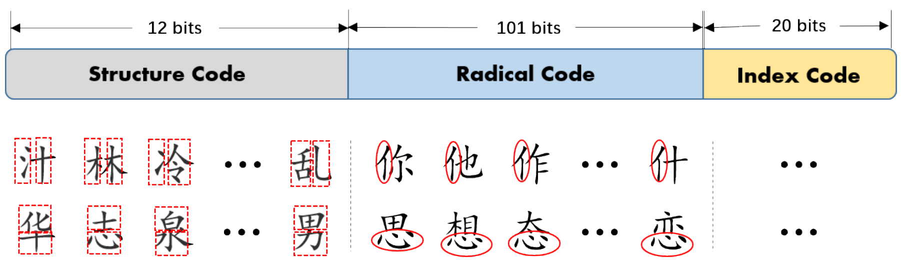
Style-Transfer VAE



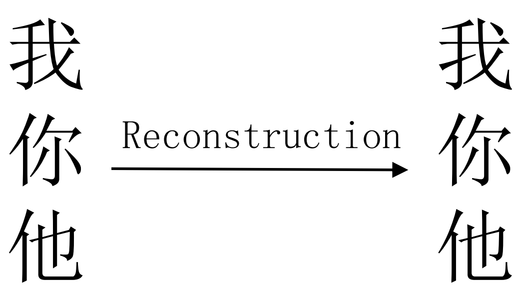


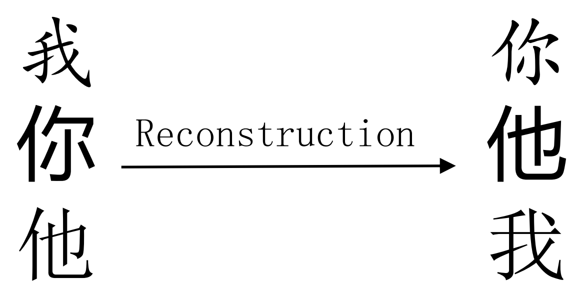
Structural Information

- Content Code (Knowledge)
  - Instead one-hot label to identify unique Chinese characters with human knowledge.
  - Reuse the structure information sharing in all Chinese characters.

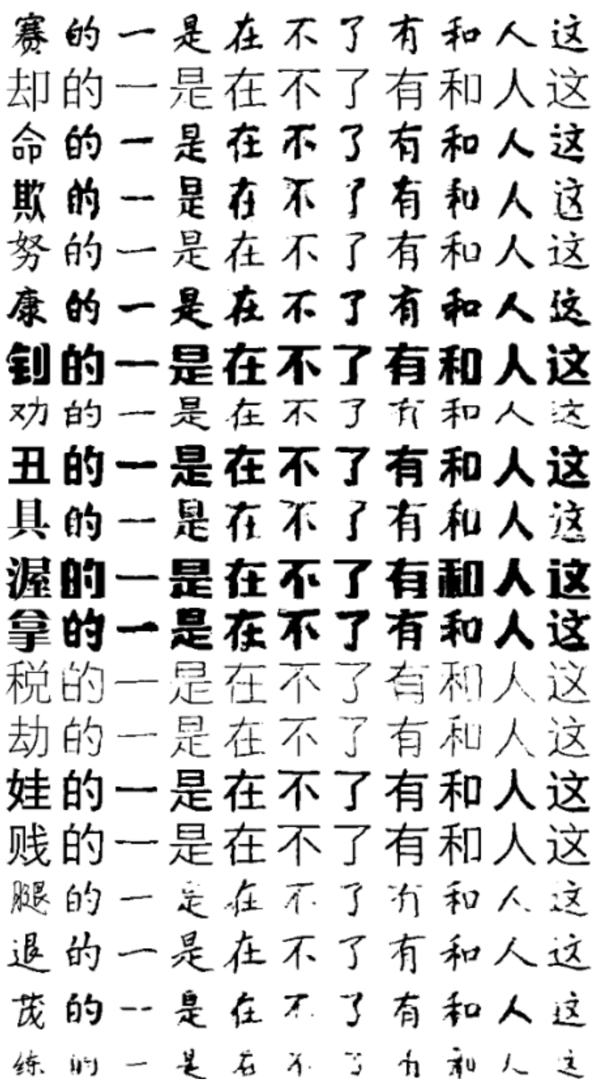


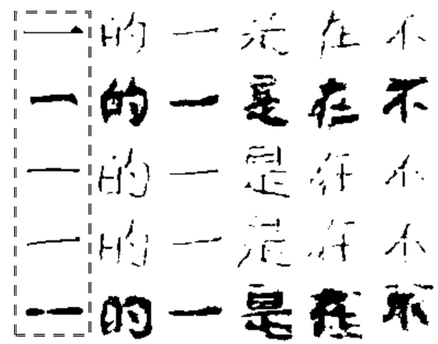
- Pairwise Training:





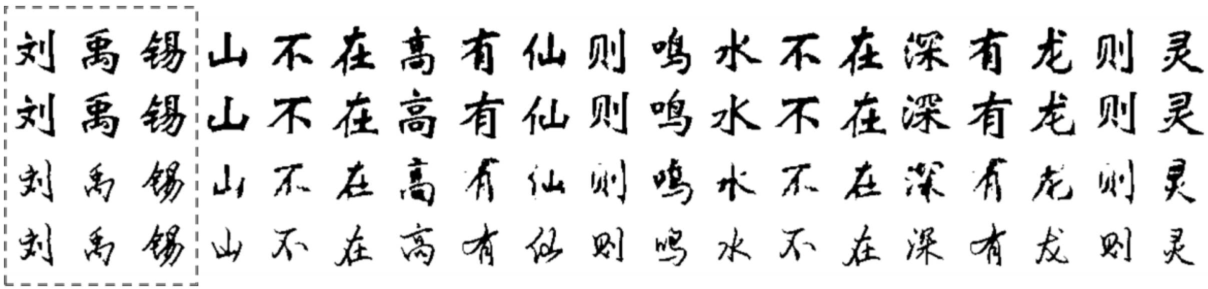
One-Shot Generation

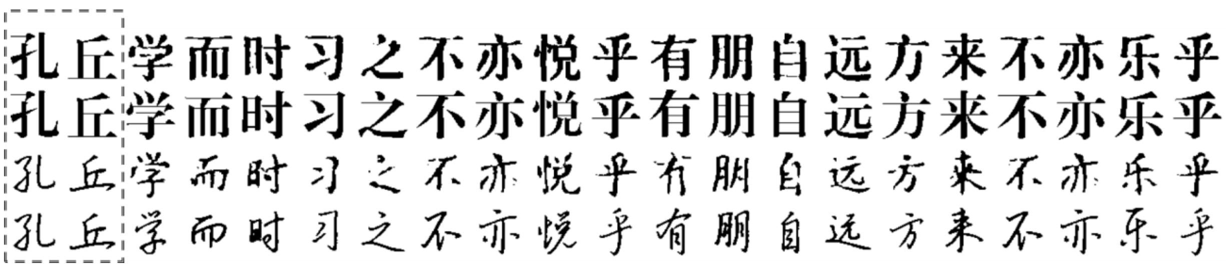


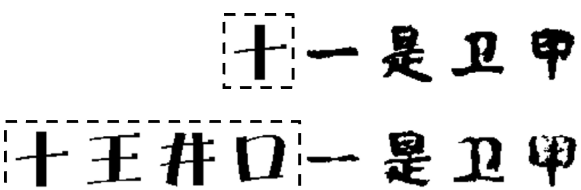


Too simple characters can't provide sufficient style information.

Low-Shot Generation







Sometimes low-shot can capture more detailed style information.

Style Generation

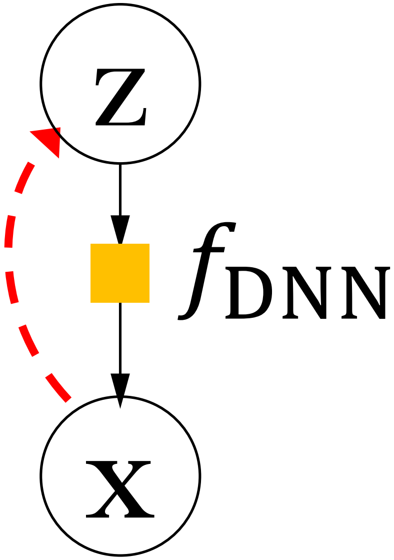
- Interpolation

Outline

- Deep Generative Models
- Semi-supervised Learning
- ZhuSuan: a Probabilistic Programming library
- Conclusions & QA

Bayesian inference

$$p(\mathrm{z}|\mathrm{x}) = \frac{p(\mathrm{x},\mathrm{z})}{p(\mathrm{x})}$$



Given Disease, what is the Cause?

Given Object, what are the Components?

Given Docs, what are the Topics?

Find Cause of Disease

Extract Topics from Docs

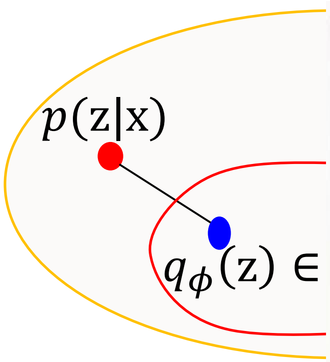
Identify Objects from Images

Recognize Words in Speeches

Inference in Old Days

Variational Inference

(Too much math!!!)



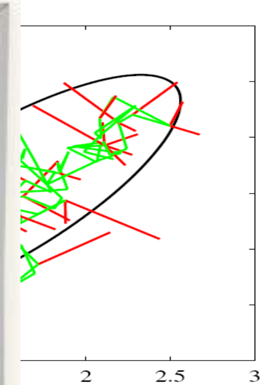
$\min_{\phi} \mathrm{KL}(q$

Varia

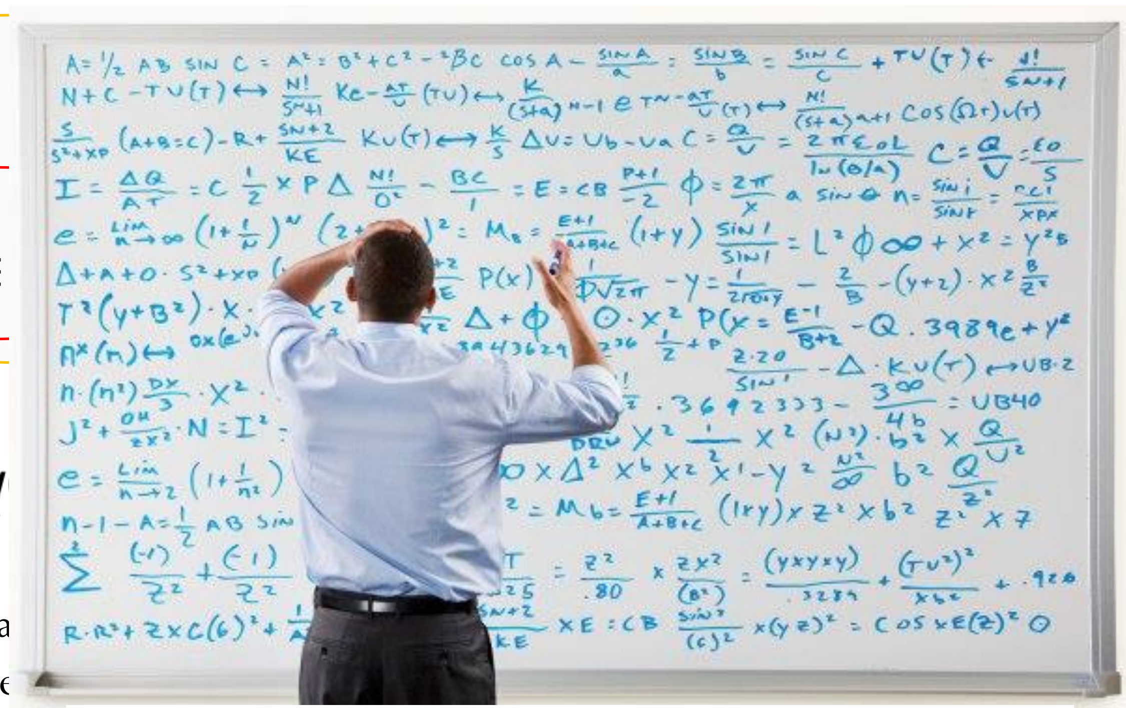
poste

MCMC

(Many dynamics!!!)



$\nabla_q[p^\top G^{-1} p]\mathrm{d}t + \mathcal{N}(0, 2CG\mathrm{d}t)$

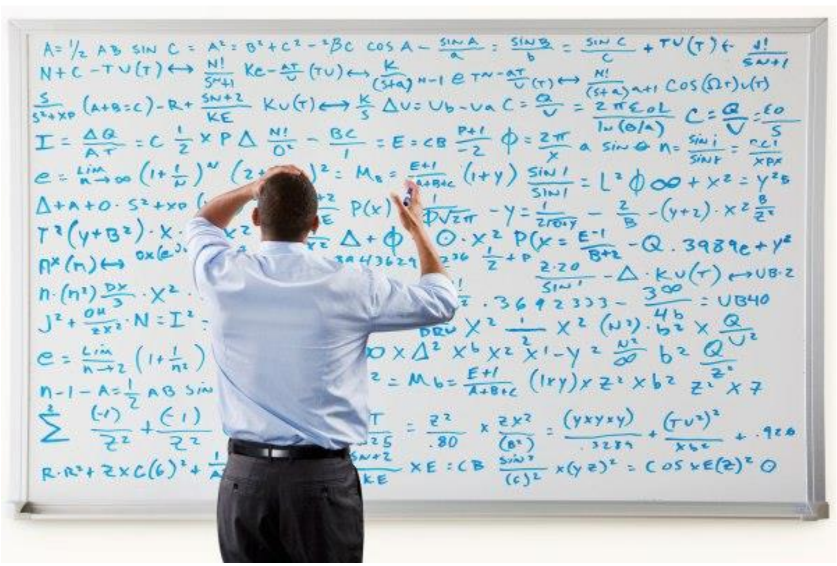


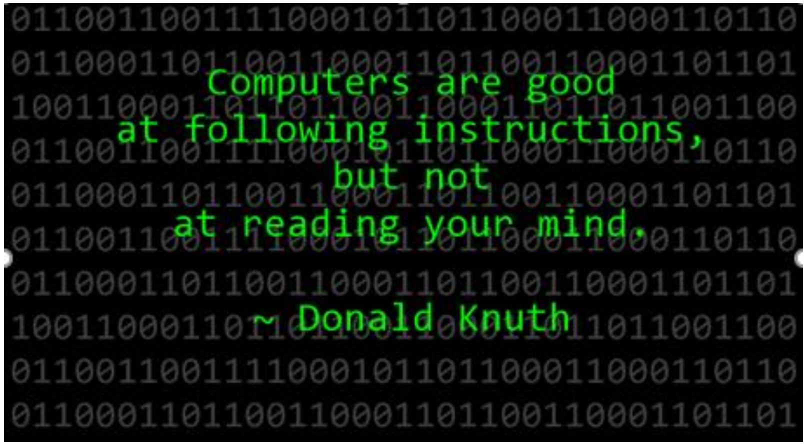
Extremely painful for a single model!

ZHUSUAN

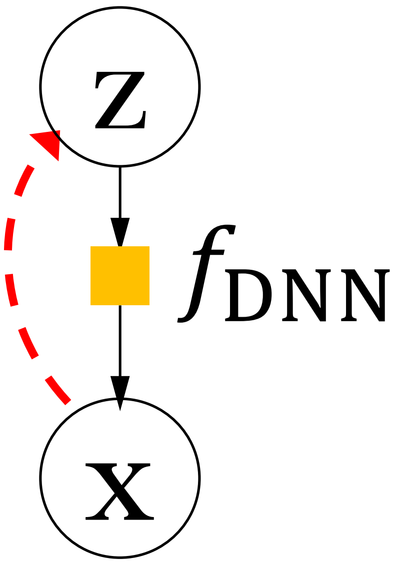
Inference in ZhuSuan

Turn painful math deviations into Easy and Intuitive (Probabilistic) Programming





ZhuSuan



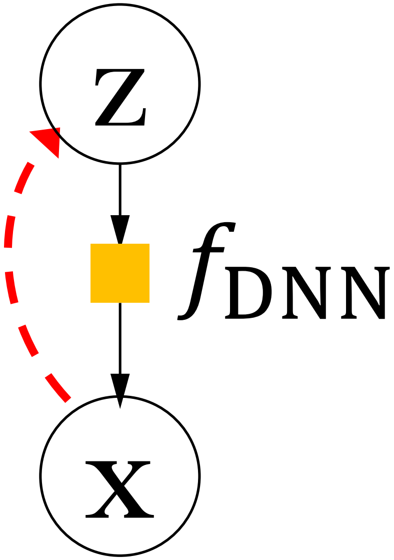
ZhuSuan is a python library for generative models, built upon TensorFlow.

Unlike existing DL libraries, which are mainly for supervised tasks, ZhuSuan is featured for:

- its deep root into Bayesian Inference
- supporting various kinds of generative models: traditional hierarchical Bayesian models & recent deep generative models.

ZhuSuan

ZHUSUAN



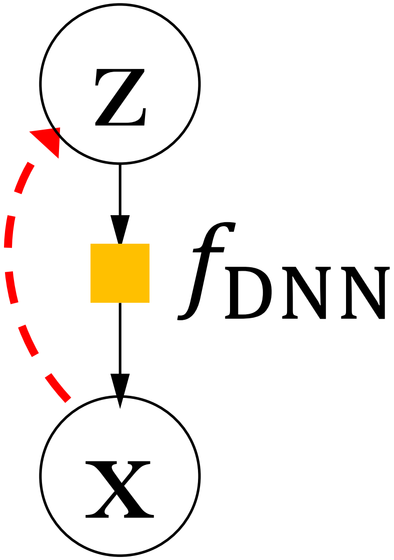
With ZhuSuan, users can enjoy

- powerful fitting and multi-GPU training of deep learning

- while at the same time they can use generative models to
  - ✓ model the complex world
  - ✓ exploit unlabeled data
  - ✓ deal with uncertainty by performing principled Bayesian inference
  - ✓ generate new samples

Model Primitives: BayesianNet

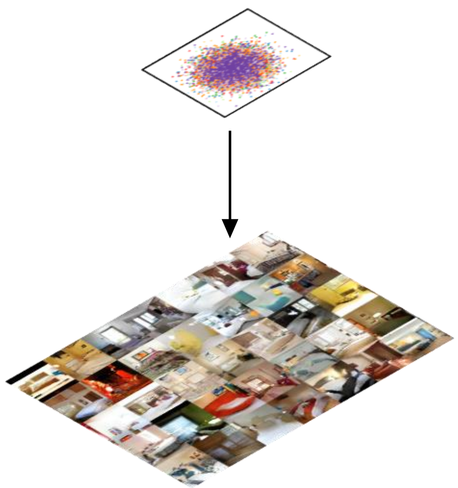
- A DAG representing a Bayesian Network
- Two types of nodes:
  - Deterministic nodes: Can be composed of any Tensorflow operations.
  - Stochastic nodes: Use StochasticTensor's from ZhuSuan's library.
- Start a BayesianNet environment



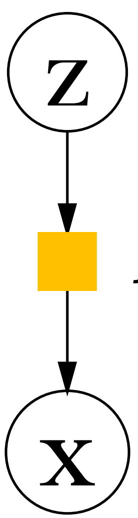

```
import zhusuan as zs
with zs.BayesianNet() as model:
    # build the model
```

Example: Variational Autoencoders

ZHUSUAN



$$z \sim \mathrm{N}(z|0, I)$$
$$x_{logits} = f_{NN}(z)$$
$$x \sim \mathrm{Bernoulli}(x|\mathrm{sigmoid}(x_{logits}))$$



f_{DNN}: $2L$ MLP

```
import tensorflow as tf
from tensorflow.contrib import layers
import zhusuan as zs

with zs.BayesianNet() as model:
    z_mean = tf.zeros([n, n_z])
    z_logstd = tf.zeros([n, n_z])
    z = zs.Normal('z', z_mean, z_logstd)
    h = layers.fully_connected(z, 500)
    x_logits = layers.fully_connected(
        h, n_x, activation_fn=None)
    x = zs.Bernoulli('x', x_logits)
```

Variational Inference in ZhuSuan

ZHUSUAN

```
with zs.BayesianNet() as variational:
    # build variational ...
qz_samples, log_qz = variational.query(
    'z', outputs=True, local_log_prob=True)
```

- Build variational posterior as BayesianNet

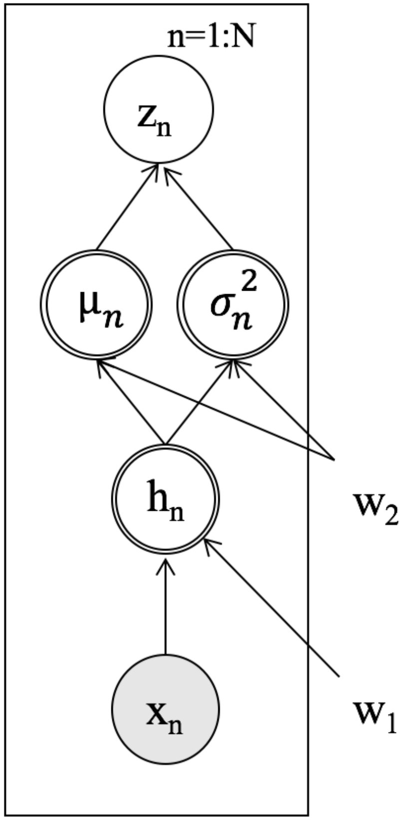
- Call variational objectives

```
lower_bound = zs.sgvb(log_joint, observed={'x': x},
                      latent={'z': [qz_samples, log_qz]})
```

```
optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
run_op = optimizer.minimize(-lower_bound)
With tf.Session() as sess:
    for iter in range(iters):
        sess.run(run_op)
```

- Run gradient descent!

Example: Variational Autoencoders



Structured variational posterior as a BayesianNet ALSO.

```
import tensorflow as tf
from tensorflow.contrib import layers
import zhusuan as zs

with zs.BayesianNet() as variational:
    x = tf.placeholder([None, n_x], tf.float32)
    h = layers.fully_connected(x, 500)
    z_mean = layers.fully_connected(
        h, n_z, activation_fn=None)
    z_logstd = layers.fully_connected(
        h, n_z, activation_fn=None)
    z = zs.Normal('z', z_mean, z_logstd)
```

Variational Inference Algorithms

ZhuSuan supports a broad class of variational objectives, ranging from widely used evidence lower bounds to recent state-of-arts.

Works for continuous latent variables

- `zs.sgvb`: Stochastic gradient variational Bayes.
- `zs.iwae`: Importance weighted lower bounds.

Works for both continuous and discrete latent variables

- `zs.nvil`: Variance reduced score function estimator/REINFORCE.
- `zs.vimco`: Variance reduced multi-sample score function estimator.

* This is like optimization algorithms (SGD, momentum, Adam, etc.) in deep learning software. Users need not dive into the technical details of these algorithms because ZhuSuan provides easy-to-use APIs for users to directly try on their generative models.

HMC like a TensorFlow optimizer

ZHUSUAN

```
# like creating the variable to optimize over.
z = tf.Variable(0.)
```

- Create the variable to store samples

```
# like optimizer = tf.train.AdamOptimizer(…)
hmc = zs.HMC(step_size=1e-3, n_leapfrogs=10)
```

- Initialize HMC

```
# like optimize_op = optimizer.minimize(…)
sample_op, hmc_info = hmc.sample(
  log_joint, observed={'x': x}, latent={'z': z})
```

- Call sample() method to return a sample operation

```
with tf.Session() as sess:
    for iter in range(iters):
        # like sess.run(optimize_op)
        _ = sess.run(sample_op)
```

- Run the sample operation like an optimizer!

Applications: ZhuSuan as a Research Platform

ZhuSuan is featured for both Bayesian Statistics and Deep Learning.
State-of-the-Art models can be found in ZhuSuan's examples.

- Bayesian Logistic Regression
- Bayesian Neural Nets for Multivariate Regression
- (Convolutional) Variational Autoencoders (VAE)
- Semi-supervised learning for images with VAEs
- Deep Sigmoid Belief Networks
- Generative Adversarial Networks (GAN)
- Gaussian processes (GPs)
- Topic Models
- More to come …

ZhuSuan: GitHub Page

ZHUSUAN

thu-ml / **zhusuan**

Watch 116 | Star 1,202 | Fork 233

Code | Issues 5 | Pull requests 4 | Projects 0 | Insights

Contributions & Stars Welcome!

Branch: master ▾ zhusuan / docs / **index.rst**

Jiaxin Change README to markdown for displaying the logo.

1 contributor

86 lines (63 sloc) | 2.49 KB

Welcome to ZhuSuan

ZhuSuan is a python probabilistic programming library for
advantages of Bayesian methods and deep learning. ZhuSu
which are mainly designed for deterministic neural network
primitives and algorithms for building probabilistic models
algorithms include:

- Variational inference with programmable variational pc (SGVB, REINFORCE, VIMCO, etc.).
- Importance sampling for learning and evaluating mode
- Hamiltonian Monte Carlo (HMC) with parallel chains, and optional automatic parameter tuning.

README.md

ZhuSuan

ZhuSuan is a python library for **Generative Models**, built upon Tensorflow. Unlike existing deep learning libraries, which are mainly designed for supervised tasks, ZhuSuan is featured for its deep root into Bayesian Inference, thus supporting various kinds of generative models: both the traditional **hierarchical Bayesian models** and recent **deep generative models**.

With ZhuSuan, users can enjoy powerful fitting and multi-GPU training of deep learning, while at the same time they can use generative models to model the complex world, exploit unlabeled data and deal with uncertainty by performing principled Bayesian inference.

Supported Inference

(Stochastic) Variational Inference (VI & SVI)

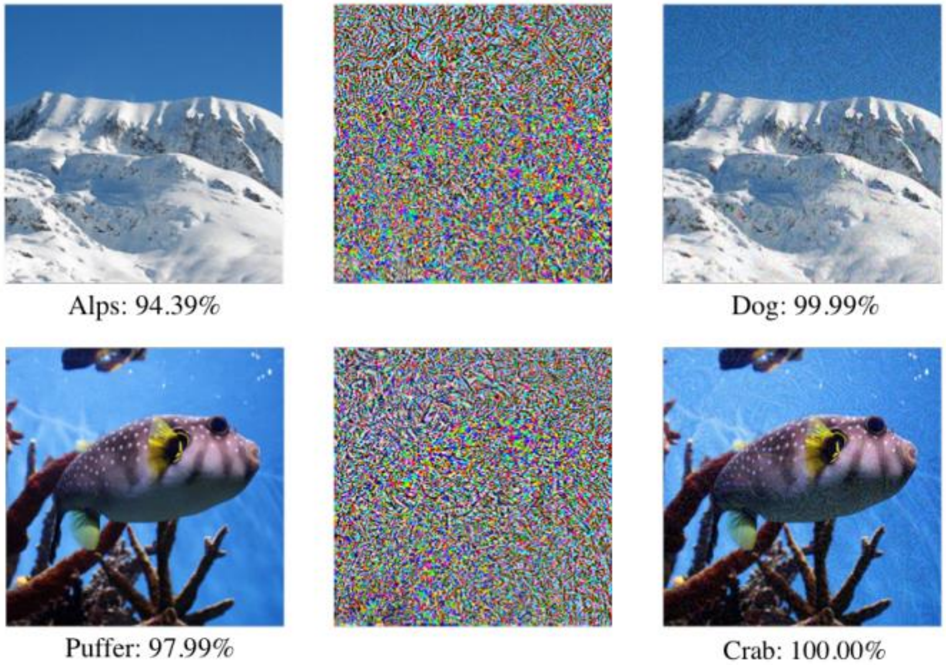
- Kinds of variational posteriors we support:
  - **Mean-field** posterior: Fully-factorized.
  - **Structured** posterior: With user specified dependencies.
- Variational objectives we support:
  - **SGVB**: Stochastic gradient variational Bayes
  - **IWAE**: Importance weighted objectives
  - **NVIL**: Score function estimator with variance reduction
  - **VIMCO**: Multi-sample score function estimator with variance

github.com/thu-ml/zhusuan

Summary

- Deep Generative models are powerful tools
  - hierarchical Bayesian models
  - deep generative models
  - semi-supervised learning & style transfer
- ZhuSuan provides a Python programming library
  - deep root into Bayesian inference
  - support deep generative models

DL still long way to go …



- Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, et al., Boosting Adversarail Attacks with Momentum, CVPR 2018.
- F. Liao, M. Liang, Y. Dong, T. Pang, J. Zhu, X. Hu. Defense against Adversarial Attack using High-level Representation Guided Denoiser, CVPR 2018.
- NIPS 2017 Adversarial Attack & Defense, first-places in all three tasks.

What VALSE taught me?

ACML 2018
November 14 - 16
Beijing, China
Asian Conference on Machine Learning

Main » Home

HOME

CONFERENCE

- Speakers
- Call For Tutorials
- Call For Workshops
- Award
- Committees

FOR AUTHORS

- Call For Papers
- Paper Submission

FOR PARTICIPANTS

The 10th Asian Conference on Machine Learning, Beijing

Beijing Jiaotong University
November 14 - 16, 2018

ACML 2018

Welcome to the 10th Asian Conference on Machine Learning (ACML 2018). The conference will take place on November 14 - 16, 2018 at Beijing Jiaotong University, Beijing, China. The conference aims to provide a leading international forum for researchers in machine learning and related fields to share their new ideas, progresses and achievements. Submissions from regions other than the Asia-Pacific are also highly encouraged.

Thanks!

ZHUSUAN

ZhuSuan: A Library for Bayesian Deep Learning. J. Shi, J. Chen, J. Zhu, S. Sun, Y. Luo, Y. Gu, Y. Zhou. arXiv preprint, arXiv:1709.05870 , 2017

Online Documents: http://zhusuan.readthedocs.io/