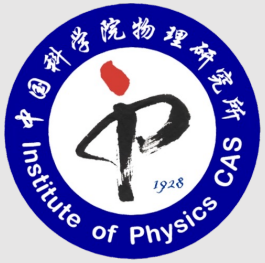
Flow model for computational physics

Qi Zhang (张琦)

Co-Supervisor: Lei Wang (王磊)

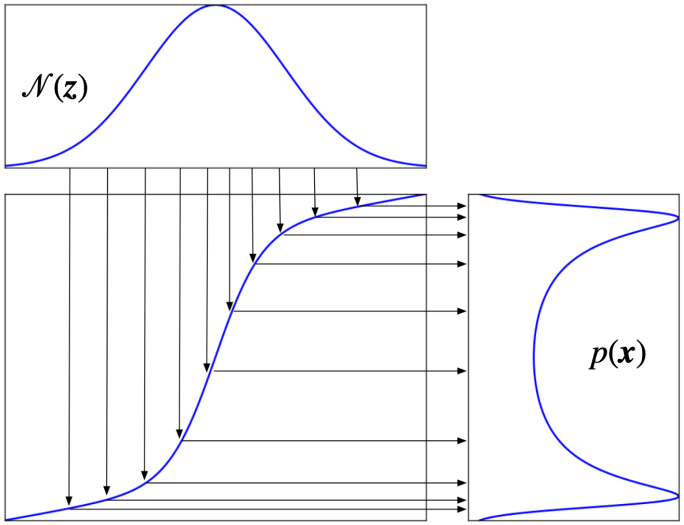
Institute of Physics, CAS



Outline

Normalizing flow

$$p(\boldsymbol{x}) = \mathcal{N}(z)\left|\det\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right)\right|$$



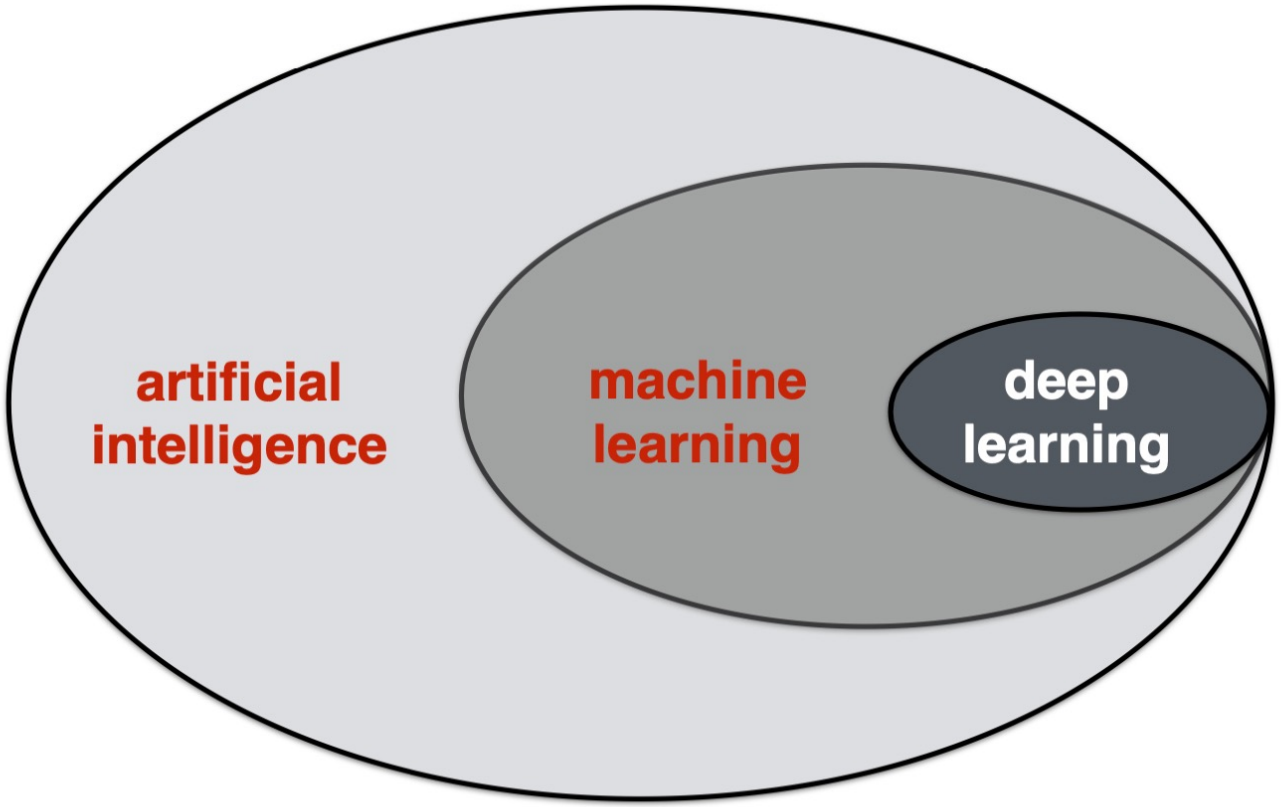
https://wangleiphy.github.io/lectures/PILtutorial.pdf
https://github.com/wangleiphy/ml4p

JAX

`grad`, `jit`, `vmap`, and `pmap`.

https://github.com/google/jax
https://jax.readthedocs.io/en/latest/index.html

Machine learning



Game-changing technology for human life and scientific research.

Machine learning

Discriminative learning

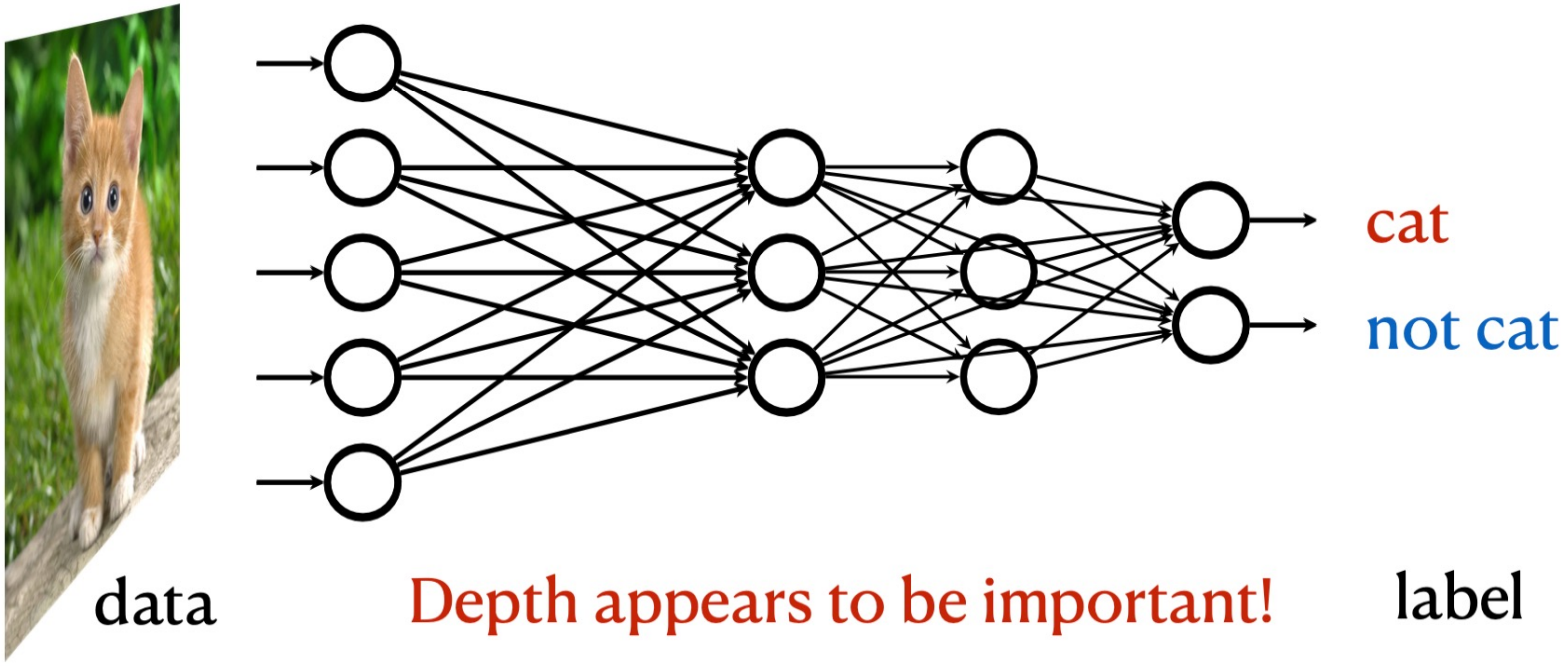
$$y = f(\boldsymbol{x})$$

or $p(y \mid \boldsymbol{x})$

Generative learning

$$p(\boldsymbol{x}, y)$$

Deep learning



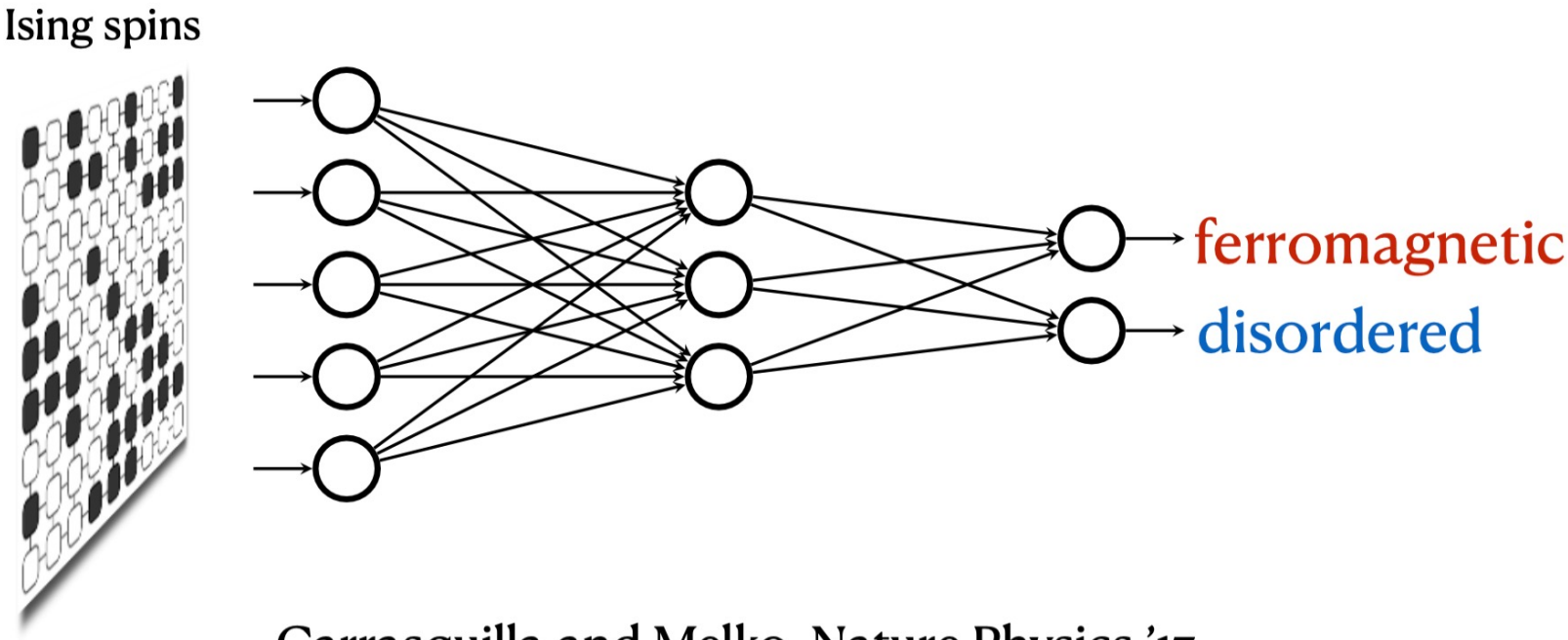
Q: Why does deep learning work?

Universal Function Approximator

A: Law of physics: symmetry, locality, composability...

Cybenko 1989
Lin, Tegmark, Rolnick, 1608.08225
Hornik, Stinchcombe, White 1989

Deep learning

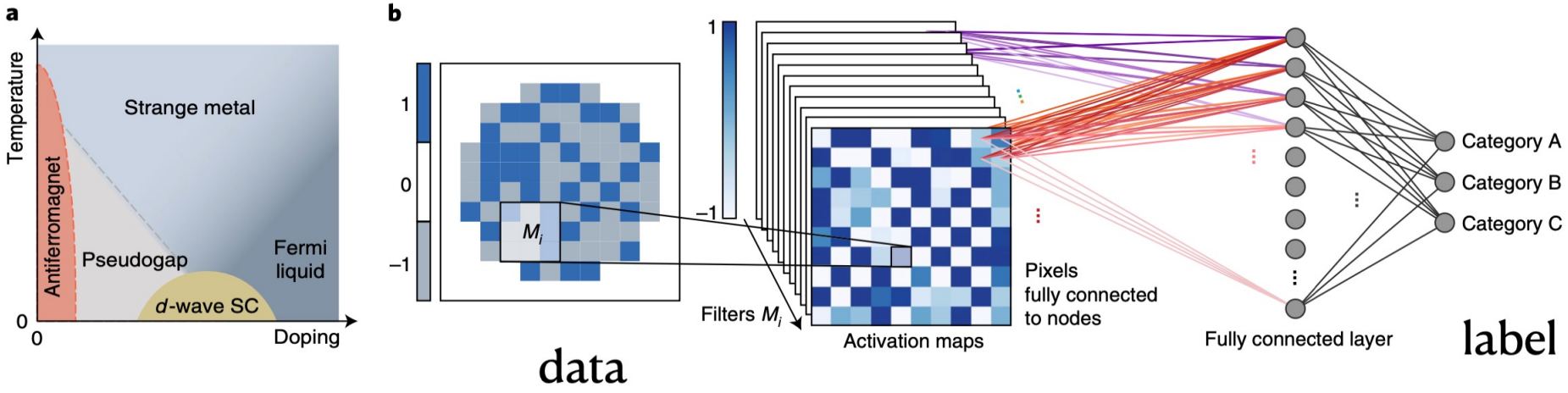


Carrasquilla and Melko, Nature Physics '17

+actual experimental data (ultracold atoms/STM/...)

Discovering Phase Transitions with Unsupervised Learning, LW, PRB, 16'

Deep learning



Bohrdt et al, Nature Physics 2019

Generative Pre-Training

Using ChatGPT: https://chatgpt.com

ChatGPT 4o

How to solve 2D Ising model?

The 2D Ising model is a mathematical model used in statistical mechanics to describe ferromagnetism in materials. Solving the 2D Ising model involves finding the partition function, magnetization, and other thermodynamic properties. Here's an overview of the steps involved in solving the 2D Ising model:

1. Model Definition

The 2D Ising model consists of a grid (lattice) of spins, where each spin can be in one of two states: up (+1) or down (-1). The Hamiltonian (energy function) of the system is given by:

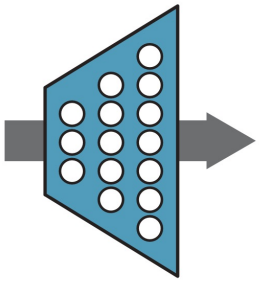
$$H = -J \sum_{\langle i,j \rangle} s_i s_j - h \sum_i s_i$$

where:

- s_i is the spin at site i.
- J is the interaction energy between neighboring spins.
- $\langle i,j \rangle$ denotes the sum over nearest-neighbor pairs.
- h is an external magnetic field.

Generative arts

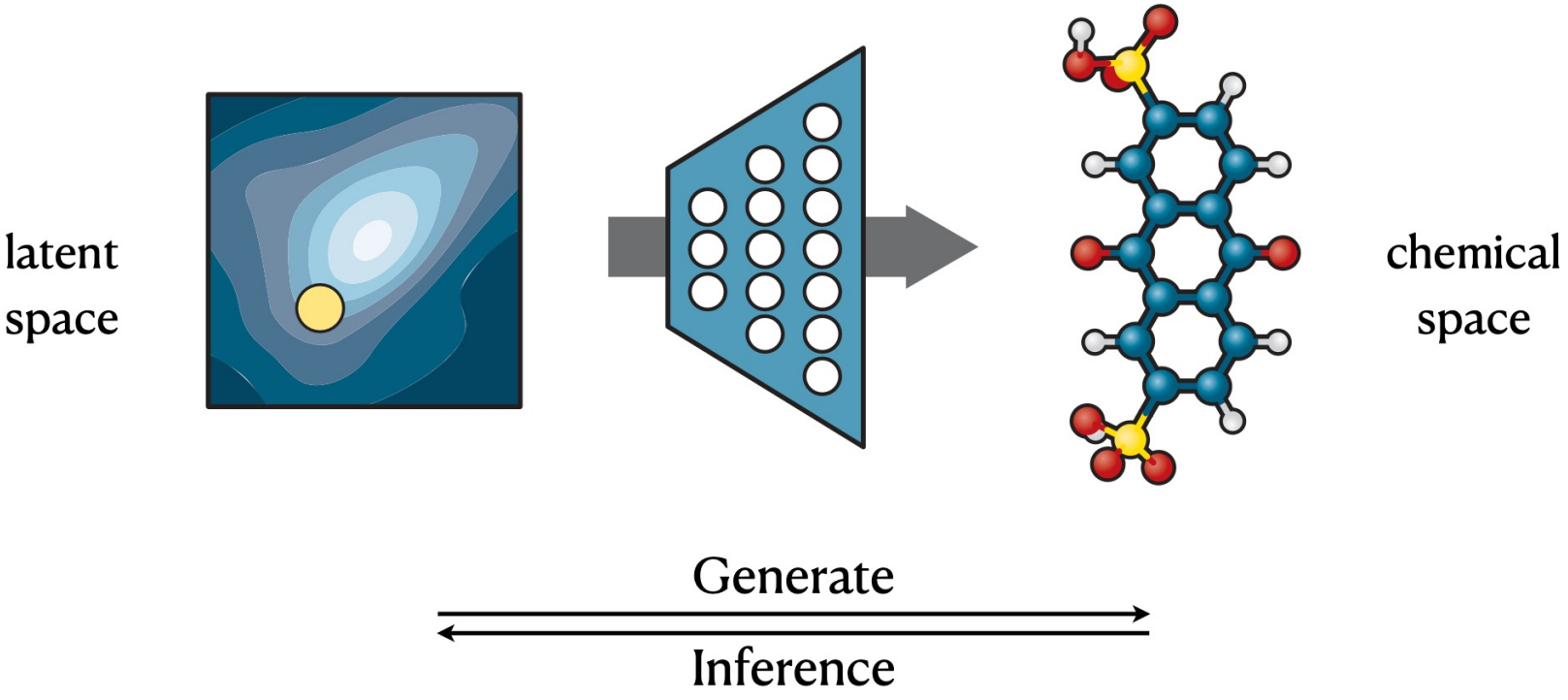
Gaussian Noise



Generative Network

$432,500
25 October 2018
Christie's New York

Generative molecules

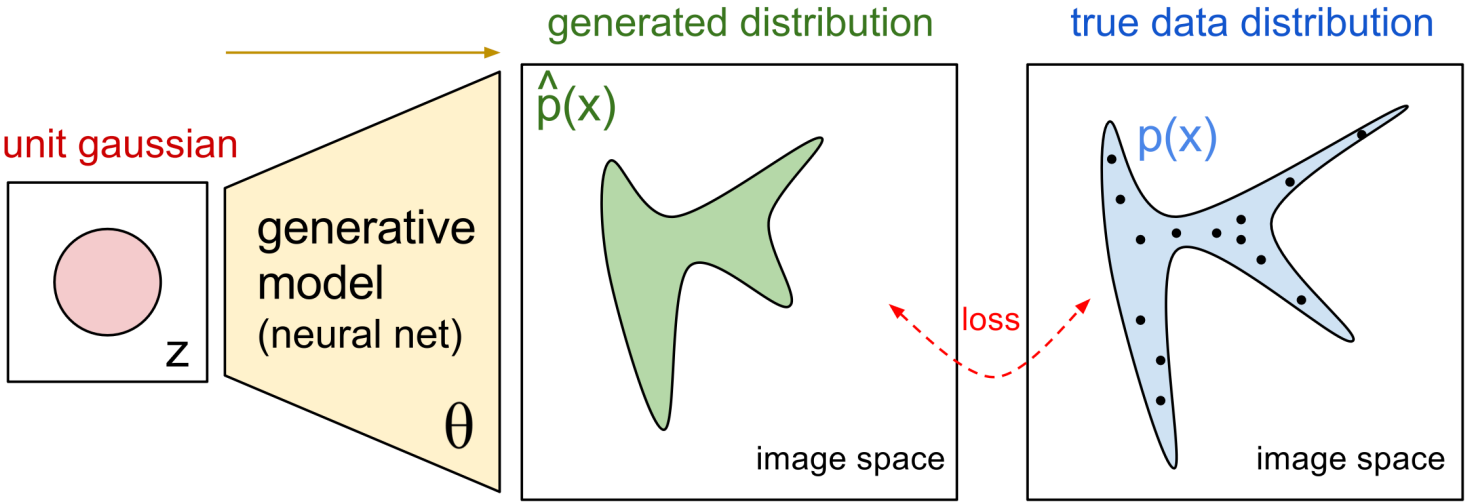


Review: “Inverse molecular design using machine learning”, Sanchez-Lengeling & Aspuru-Guzik, Science ’18

Probabilistic Generative Modeling

$$p(\boldsymbol{x})$$

How to express, learn, and sample from a high-dimensional probability distribution ?



https://blog.openai.com/generative-models/

References

Lecture Note: https://wangleiphy.github.io/lectures/PILtutorial.pdf

Generative Models for Physicists

Lei Wang*

Institute of Physics, Chinese Academy of Sciences
Beijing 100190, China

November 5, 2018

Abstract

Generative models generate unseen samples according to a learned joint probability distribution in the high-dimensional space. They find wide applications in density estimation, variational inference, representation learning and more. Deep generative models and associated techniques (such as differentiable programing and representation learning) are cutting-edge technologies physicists can learn from deep learning.

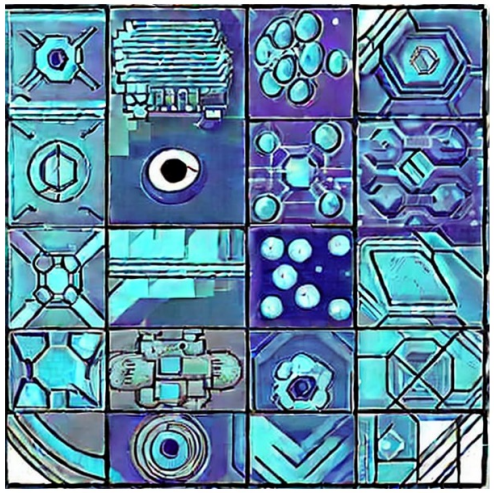
This note introduces the concept and principles of generative modeling, together with applications of modern generative models (autoregressive models, normalizing flows, variational autoencoders etc) as well as the old ones (Boltzmann machines) to physics problems. As a bonus, this note puts some emphasize on physics-inspired generative models which take insights from statistical, quantum, and fluid mechanics.

The latest version of the note is at http://wangleiphy.github.io/. Please send comments, suggestions and corrections to the email address in below.

CONTENTS

References

Machine learning for physicists: https://github.com/wangleiphy/ml4p



1. Overview
   - Scientific machine learning with and without data
2. Machine learning practices
   - Hello world example: Hand-written digits recognition
   - Programming frameworks, hardware, and workflow
3. A hitchhiker's guide to deep learning
   - The four pillars: data, model, loss function, and optimization
   - Deep learning primitives: CNN, GNN, and transformer
4. Symmetries in machine learning
   - Invariant and equivariant neural networks
   - DeepMD, Euclidean equivariant GNN, Tensor field networks
   - Permutation symmetry and quantum wavefunctions
5. Differentiable programming
   - The engine of deep learning: automatic differentiation on computation graphs
   - Differentiable DFT/MD/Tensor networks/..., and why they are useful
6. Generative models-I
   - A dictionary of generative models and statistical physics
   - Boltzmann machines
   - Autoregressive models
   - Variational autoencoders
7. Generative models-II
   - Normalizing flows
   - Diffusions models
   - Applications of generative models to many-body problems
   - The Universe as a generative model
8. Wrap up
   - AI for science: why now?

Title image generated by stable diffusion with the prompt: "a tile image for the course on 'Machine learning for physicists', eye-catching, artist style with sci-fi feeling". (Yes, "tile" instead of "title" :P)

Generative models for physicists

Generative modeling	Statistical physics
Negative log-likelihood	Energy function
Score function	Force
Latent variables	Collective variables/coarse graining/renormalization group
Partition function	Free energy calculation
Sample diversity	Enhanced sampling

Generative models for physicists

Generative modeling

Known: samples

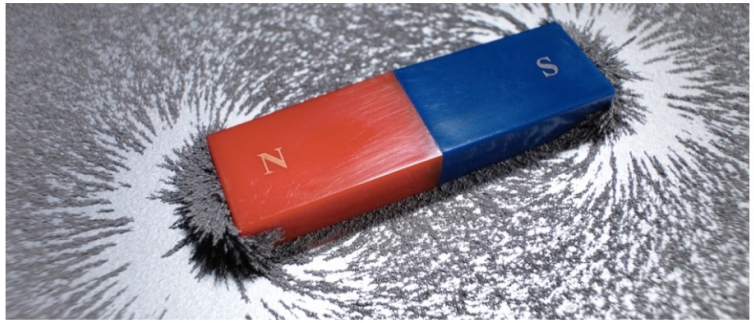
Unknown: generating distribution

Maximum likelihood estimation

"learn from data"

$$\mathscr{L} = -\mathbb{E}_{\boldsymbol{x}\sim\text{dataset}}\left[\ln p(\boldsymbol{x})\right]$$

Statistical physics



Known: energy function

Unknown: partition function, samples

Variational free energy

"learn from Hamiltonian"

$$F = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[k_B T \ln p(\boldsymbol{x}) + H(\boldsymbol{x})\right]$$

Kullback-Leibler divergence

How to measure one probability distribution diverges from a reference distribution?

Kullback-Leibler divergence!

$$\mathrm{KL}(p||q) = \sum_{x} p(\boldsymbol{x}) \ln \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}.$$

$$\mathbb{KL}(\pi \parallel p) \equiv \int d\boldsymbol{x} \pi(\boldsymbol{x}) \left[\ln \pi(\boldsymbol{x}) - \ln p(\boldsymbol{x}) \right]$$

$$\mathbb{KL}(\pi \parallel p) \geq 0$$

$$\mathbb{KL}(\pi \parallel p) = 0 \iff \pi(\boldsymbol{x}) = p(\boldsymbol{x})$$

$$\mathbb{KL}(\pi \parallel p) \neq \mathbb{KL}(p \parallel \pi)$$

Forward KL or Reverse KL?

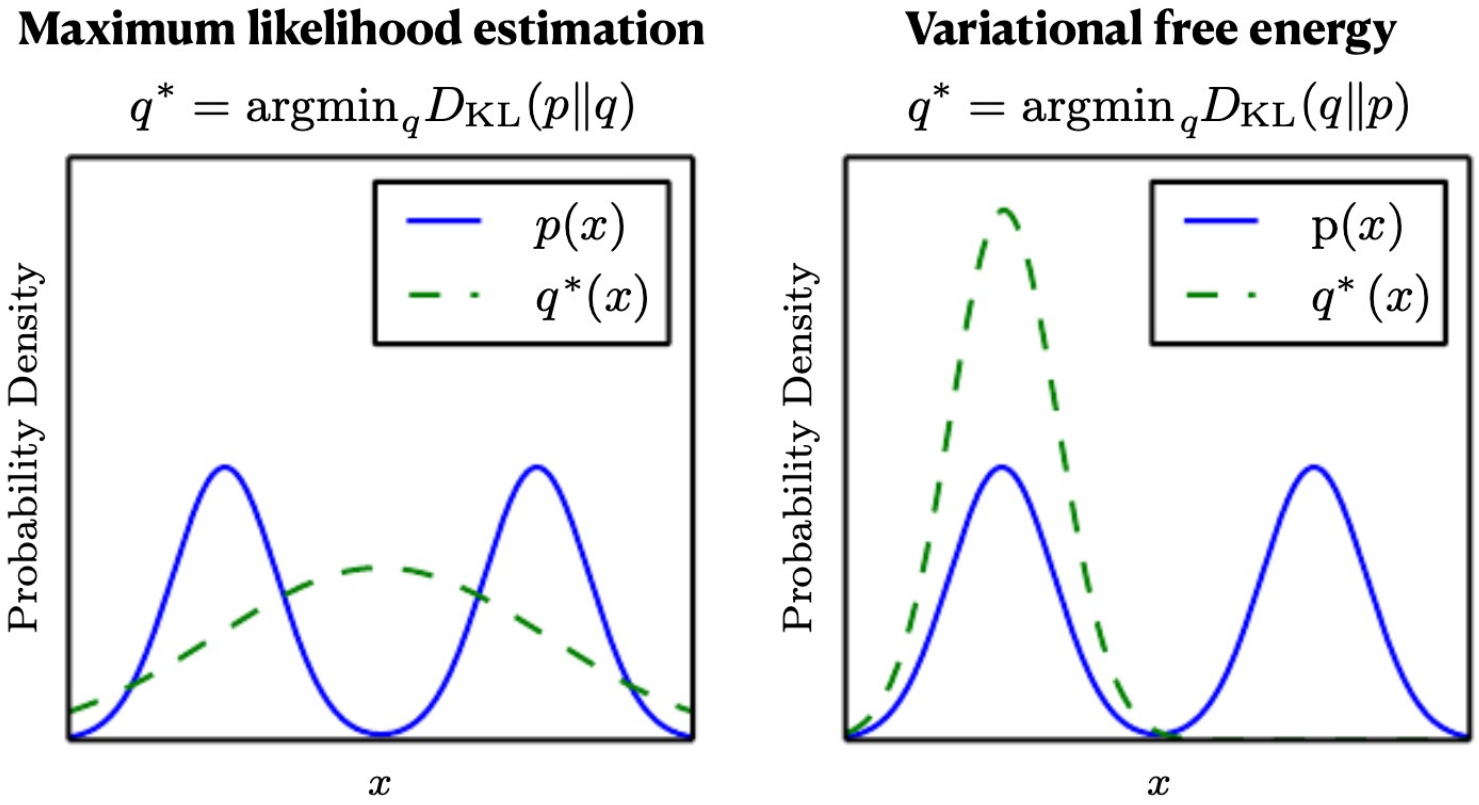


Fig. 3.6, Goodfellow, Bengio, Courville, http://www.deeplearningbook.org/

$$\text{KL}(q||p) = \text{const.} - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim q(\boldsymbol{x})}[\ln p(\boldsymbol{x}, \boldsymbol{\theta})] \qquad \text{KL}(p||q) = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\boldsymbol{x})}[\ln p(\boldsymbol{x}) - \ln q(\boldsymbol{x})]$$

Training approaches

Density estimation

"learn from data"

$$\mathscr{L} = -\mathbb{E}_{\boldsymbol{x}\sim\text{dataset}}\left[\ln p(\boldsymbol{x})\right]$$

$$\mathbb{KL}(\pi||p) = \underbrace{\sum_{\boldsymbol{x}} \pi \ln \pi - \sum_{\boldsymbol{x}} \pi \ln p}_{\mathscr{L}}$$

Sample from dataset in the physical space

Variational calculation

"learn from Hamiltonian"

$$\mathscr{L} = \int d\boldsymbol{x}\, p(\boldsymbol{x})\left[\ln p(\boldsymbol{x}) + \beta H(\boldsymbol{x})\right]$$

$$\mathscr{L} + \ln Z = \mathbb{KL}\left(p||\frac{e^{-\beta H}}{Z}\right) \geq 0$$

Sample in the latent space

Learn from Hamiltonian

Target distribution: $\pi(\boldsymbol{x}) \propto e^{-H/k_B T}$ Boltzmann distribution!

$$\min_\theta \mathbb{KL}(p_\theta \,\|\, \pi) \iff \min_\theta \left\{ \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})} \left[H(\boldsymbol{x}) + k_B T \ln p_\theta(\boldsymbol{x}) \right] \right\}$$

model (↑ p_θ) target (↑ π) Variational free energy

$$F = E - TS$$

energy entropy

F is a cost function given by Nature

Deep variational free energy approach

Use deep generative models as the variational density

$$F[p] = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\boldsymbol{x})} \left[H(\boldsymbol{x}) + k_B T \ln p(\boldsymbol{x}) \right]$$

energy ↓ entropy ↓ 😄

Li and LW, PRL '18
Wu, LW, Zhang, PRL '19

with normalizing flow & autoregressive models

✓ Tractable entropy

✓ Direct sampling

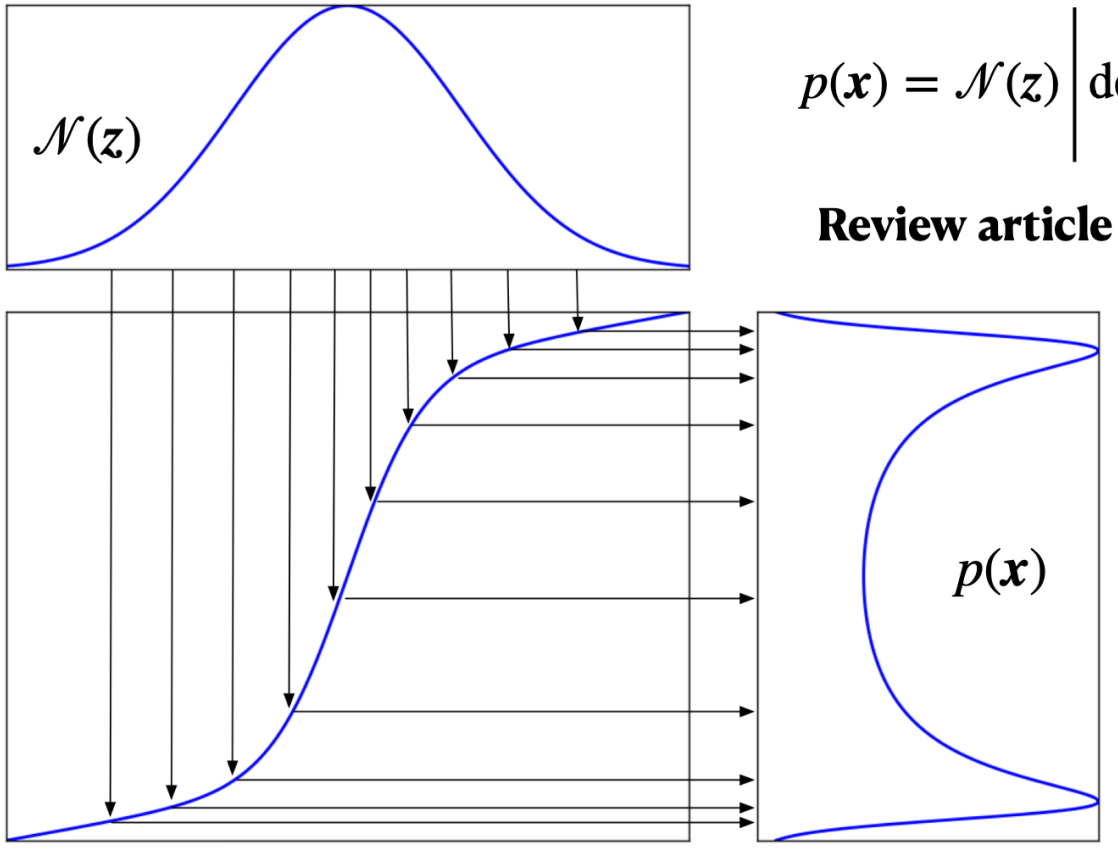
✓ Turning a sampling problem to an optimization problem better leverages the deep learning engine:

Normalizing flow

Base density

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}) \left| \det \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} \right) \right|$$

Review article 1912.02762



"neural net" with 1 neuron

Target density

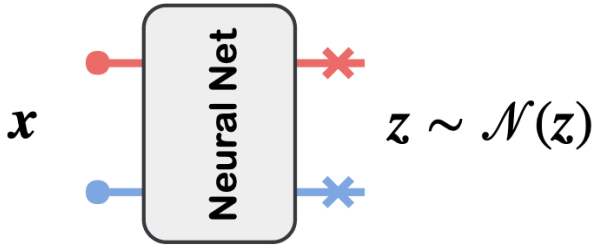
Normalizing flow

Change of variables $x \leftrightarrow z$ with deep neural nets

$$p(x) = \mathcal{N}(z) \left| \det \left(\frac{\partial z}{\partial x} \right) \right|$$

Review article 1912.02762
Tutorial https://iclr.cc/virtual_2020/speaker_4.html

composable, differentiable, and invertible mapping between manifolds



Learn probability transformations with normalizing flows

Got this name in Tabak & Vanden-Eijnden, Commun. Math. Sci. '10

Monte Carlo Gradient Estimators

$$F[p] = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\boldsymbol{x})} \left[H(\boldsymbol{x}) + k_B T \ln p(\boldsymbol{x}) \right]$$

energy ↓ entropy ↓ 😄

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}) \left| \det \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} \right) \right|$$

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}} \left[f(\boldsymbol{x}) \right]$$

Review: 1906.10652

Reinforcement learning
Variational inference
Variational Monte Carlo
Variational quantum algorithms
…

Score function estimator (REINFORCE)

$$\nabla_\theta \mathbb{E}_{\boldsymbol{x}\sim p_\theta} \left[f(\boldsymbol{x}) \right] = \mathbb{E}_{\boldsymbol{x}\sim p_\theta} \left[f(\boldsymbol{x}) \nabla_\theta \ln p_\theta(\boldsymbol{x}) \right]$$

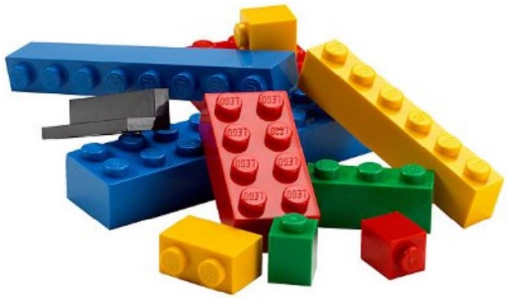
Pathwise estimator (Reparametrization trick) $\boldsymbol{x} = g_\theta(\boldsymbol{z})$

$$\nabla_\theta \mathbb{E}_{\boldsymbol{x}\sim p_\theta} \left[f(\boldsymbol{x}) \right] = \mathbb{E}_{\boldsymbol{z}\sim \mathcal{N}(\boldsymbol{z})} \left[\nabla_\theta f(g_\theta(\boldsymbol{z})) \right]$$

Choose the one with the lowest variance

Design principles

Composability

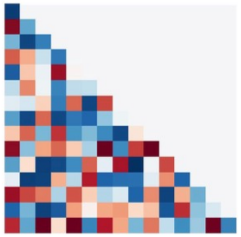


$$z = \mathcal{T}(x)$$

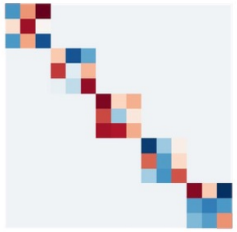
$$\mathcal{T} = \mathcal{T}_1 \circ \mathcal{T}_2 \circ \mathcal{T}_3 \circ \cdots$$

Balanced efficiency & inductive bias

$$\left|\det\left(\frac{\partial z}{\partial x}\right)\right|$$



Autoregressive



Neural RG

$$\frac{\partial \rho(x,t)}{\partial t} + \nabla \cdot \left[\rho(x,t)v\right] = 0$$

Continuous flow

Example of a building block

Forward

arbitrary neural nets

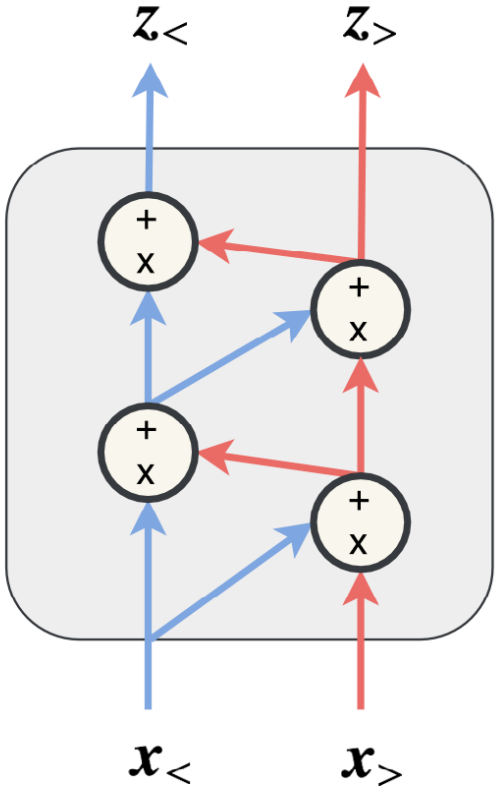
$$\begin{cases} \boldsymbol{x}_< = \boldsymbol{z}_< \\ \boldsymbol{x}_> = \boldsymbol{z}_> \odot e^{s(\boldsymbol{z}_<)} + t(\boldsymbol{z}_<) \end{cases}$$

Inverse

$$\begin{cases} \boldsymbol{z}_< = \boldsymbol{x}_< \\ \boldsymbol{z}_> = (\boldsymbol{x}_> - t(\boldsymbol{x}_<)) \odot e^{-s(\boldsymbol{x}_<)} \end{cases}$$

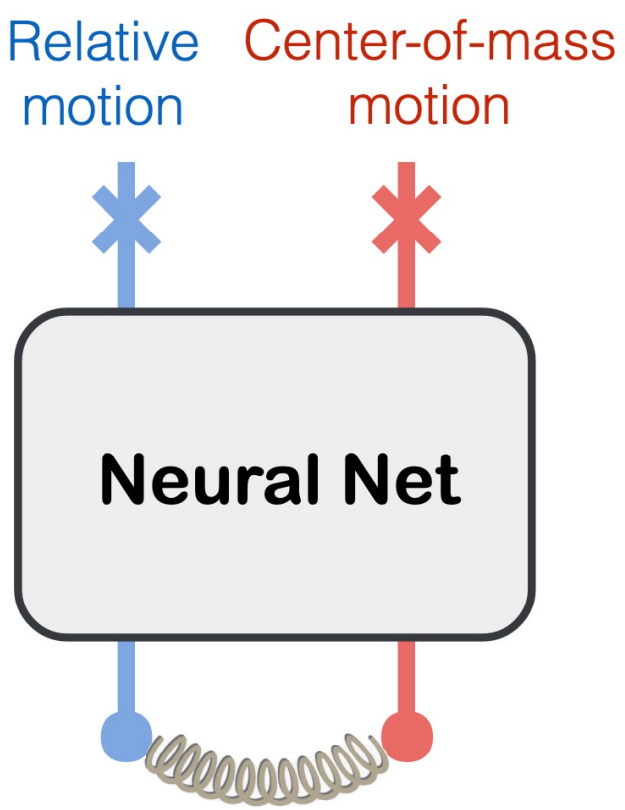
Log-Abs-Jacobian-Det

$$\ln \left| \det \left(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}} \right) \right| = \sum_i [s(\boldsymbol{z}_<)]_i$$

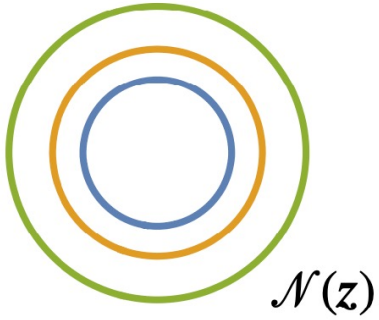


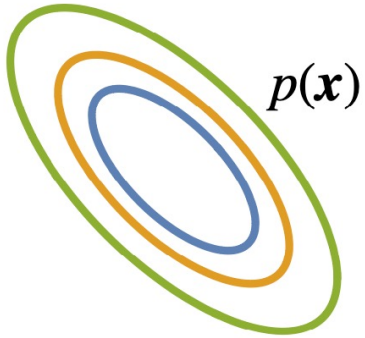
Real NVP, Dinh et al,1605.08803

How it can be useful in physics?



Coupled harmonic oscillator





Continuous normalizing flows

$$\ln p(\boldsymbol{x}) = \ln \mathcal{N}(\boldsymbol{z}) - \ln \left| \det \left(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}} \right) \right|$$

Consider infinitesimal change-of-variables Chen et al 1806.07366

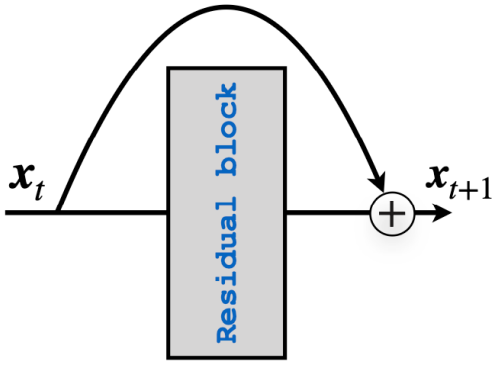
$$\boldsymbol{x} = \boldsymbol{z} + \varepsilon \boldsymbol{v} \qquad \ln p(\boldsymbol{x}) - \ln \mathcal{N}(\boldsymbol{z}) = - \ln \left| \det \left(1 + \varepsilon \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{z}} \right) \right|$$

$\varepsilon \to 0$

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v} \qquad \frac{d \ln \rho(\boldsymbol{x}, t)}{dt} = - \nabla \cdot \boldsymbol{v}$$

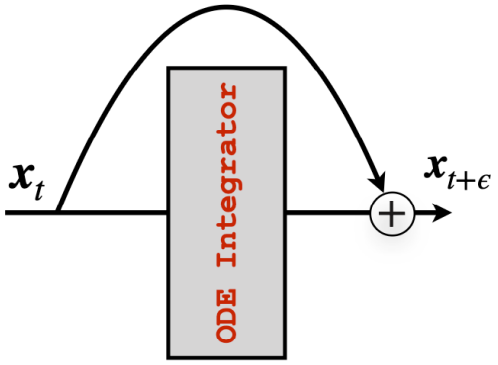
Neural Ordinary Differential Equations

Residual network



$$x_{t+1} = x_t + f(x_t)$$

ODE integration



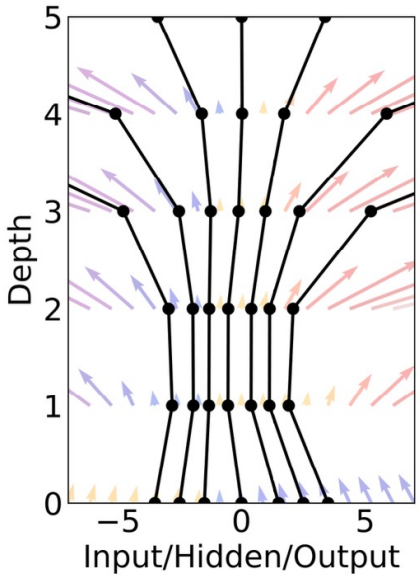
$$dx/dt = f(x)$$

Chen et al, 1806.07366

Harbor el al 1705.03341
Lu et al 1710.10121,
E Commun. Math. Stat 17'…

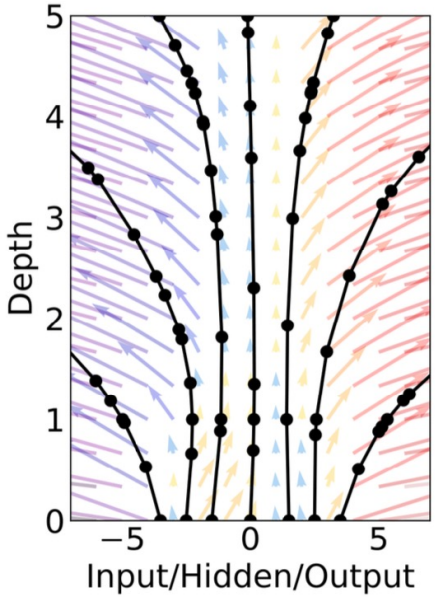
Neural Ordinary Differential Equations

Residual network



$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + f(\boldsymbol{x}_t)$$

ODE integration



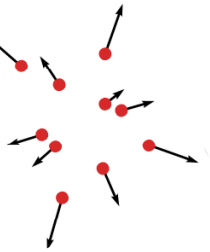
$$d\boldsymbol{x}/dt = f(\boldsymbol{x})$$

Chen et al, 1806.07366

Harbor el al 1705.03341
Lu et al 1710.10121,
E Commun. Math. Stat 17'…

Fluid physics behind flows

$$\frac{dx}{dt} = v$$

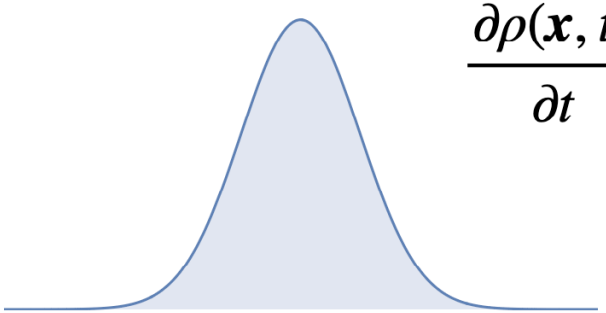


Zhang, E, LW 1809.10188
wangleiphy/MongeAmpereFlow

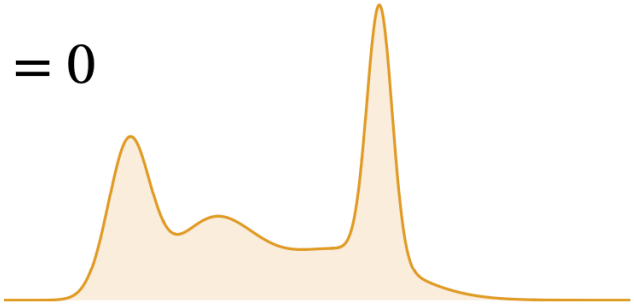
$$\frac{d \ln \rho(x,t)}{dt} = -\nabla \cdot v \qquad \frac{d}{dt} = \frac{\partial}{\partial t} + v \cdot \nabla$$

"material derivative"

$$\frac{\partial \rho(x,t)}{\partial t} + \nabla \cdot \left[\rho(x,t) v\right] = 0$$



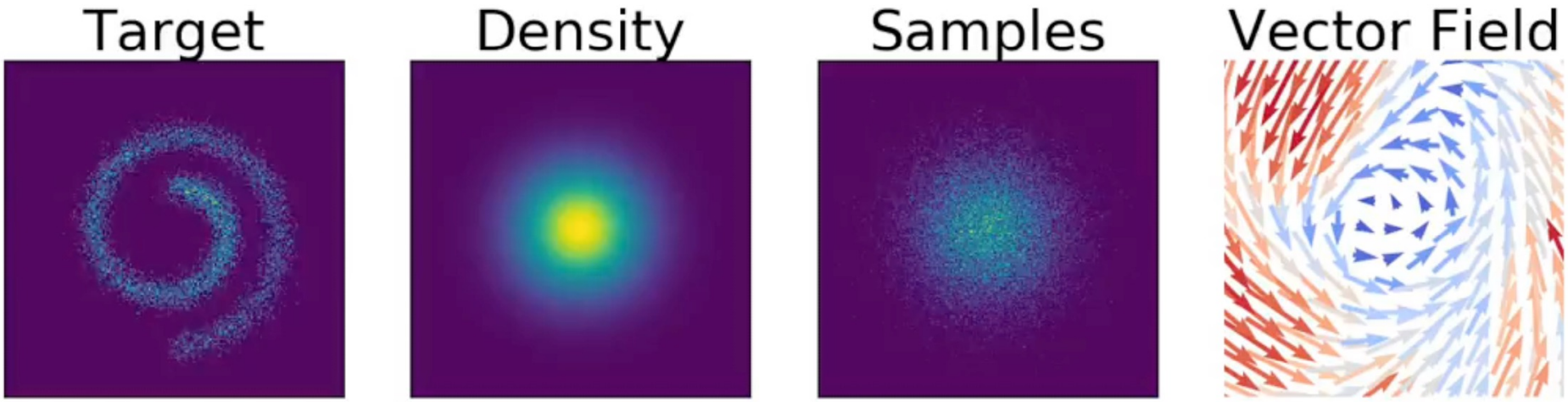
Simple density

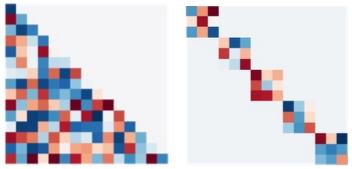


Complex density

Neural Ordinary Differential Equations

Chen et al, 1806.07366, Grathwohl et al 1810.01367

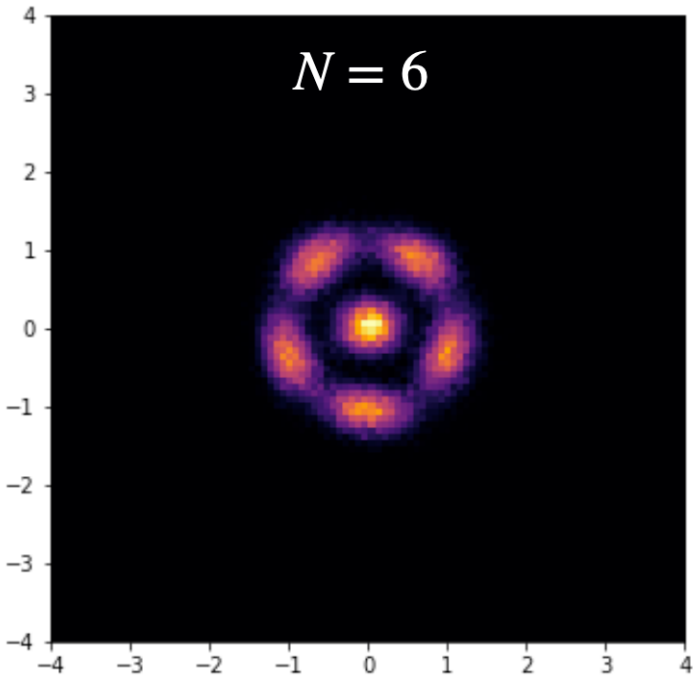


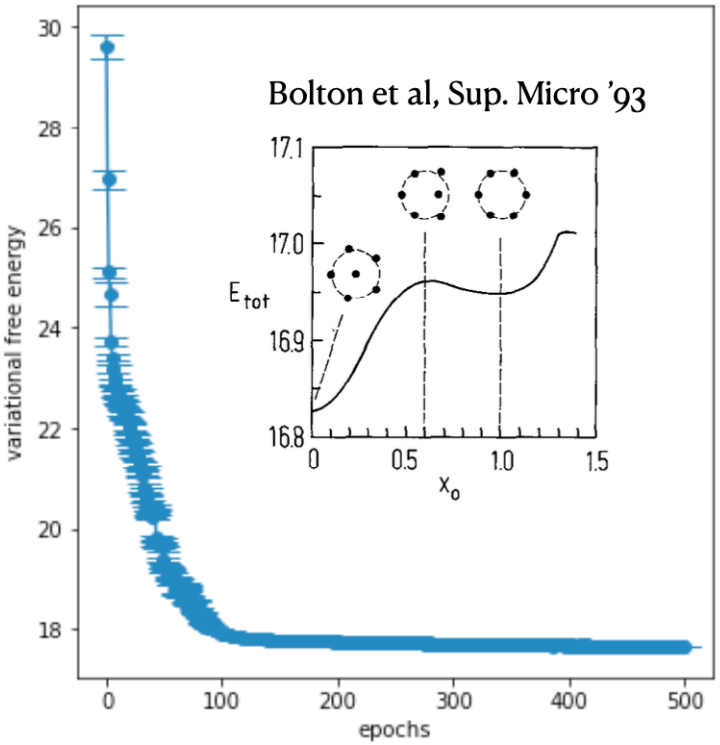


Continuous normalizing flow have no structural constraints on the transformation Jacobian

Demo: Classical Coulomb gas

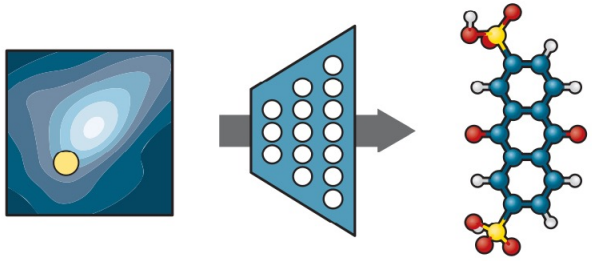
$$H = \sum_{i<j} \frac{1}{|\boldsymbol{x}_i - \boldsymbol{x}_j|} + \sum_i^N \boldsymbol{x}_i^2$$





Generate AI for Science

1. How to Build a GPT-3 for Science — Scientific language model

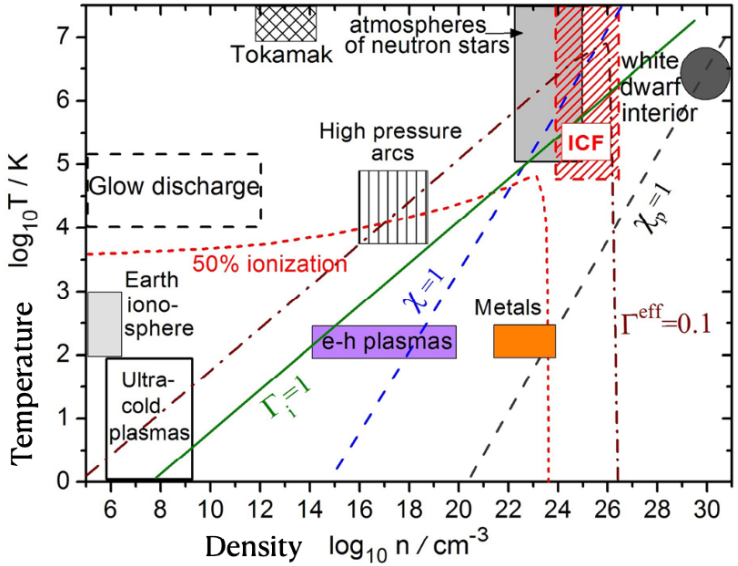
2.  — Matter inverse design

3. $F = E - TS$ — Nature's cost function

Ab-initio study of quantum matters at T>0

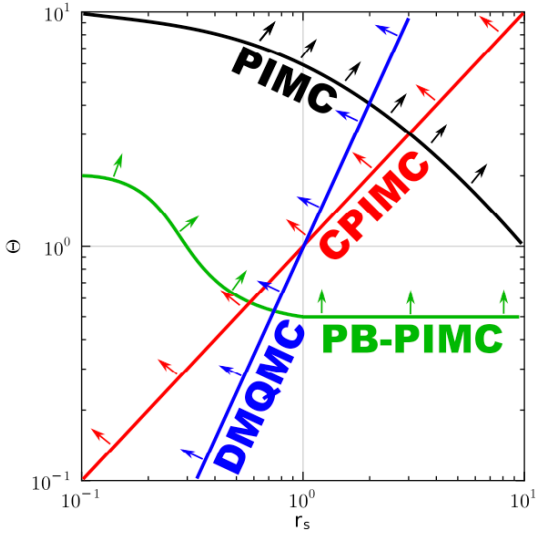
$$H = -\sum_i \frac{\hbar^2}{2m_e}\nabla_i^2 - \sum_I \frac{\hbar^2}{2m_I}\nabla_I^2 - \sum_{I,i} \frac{Z_I e^2}{|R_I - r_i|} + \frac{1}{2}\sum_{i\neq j} \frac{e^2}{|r_i - r_j|} + \frac{1}{2}\sum_{I\neq J} \frac{Z_I Z_J e^2}{|R_I - R_J|}$$

$$Z = \mathrm{Tr}(e^{-H/k_B T})$$



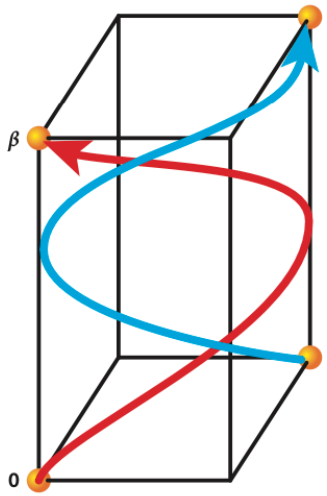
Bonitz et al, Phys. Plasmas '20

Application range of quantum Monte Carlo



Dornheim et al, Phys. Plasmas '17

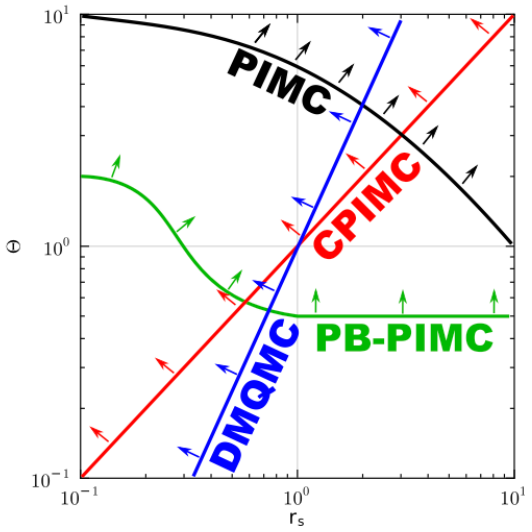
How to solve quantum many-body systems?



Quantum-to-classical mapping via path integral, then we are done

$$Z = \int d\tau\, d\boldsymbol{x} \cdots$$

However, the sign problem strikes again: the "weights" may not be positive 😂



Classical to Quantum

Classical world

Probability density p

Kullback-Leibler divergence

$$\mathbb{KL}\left(p\,||\,q\right)$$

Variational free-energy

$$F = \int d\boldsymbol{x}\ \left[\frac{1}{\beta}p(\boldsymbol{x})\ln p(\boldsymbol{x}) + p(\boldsymbol{x})H(\boldsymbol{x})\right]$$

Quantum world

Density matrix ρ

Quantum relative entropy

$$S\left(\rho\,||\,\sigma\right)$$

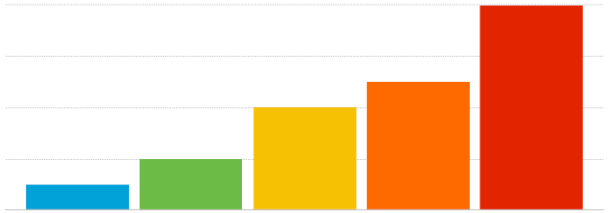
Variational free-energy

$$F = \frac{1}{\beta}\mathrm{Tr}(\rho\ln\rho) + \mathrm{Tr}(\rho H)$$

Density matrix

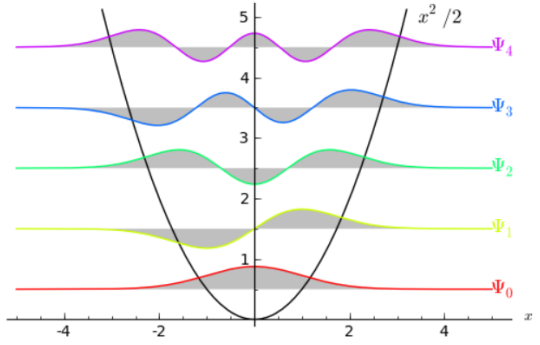
$$\rho = \sum_n p_n \,|\Psi_n\rangle\langle\Psi_n|$$

Classical probability $0 < p_n < 1$



$$\sum_n p_n = 1$$

Quantum states $\Psi_n(x) = \langle x \,|\, \Psi_n \rangle$



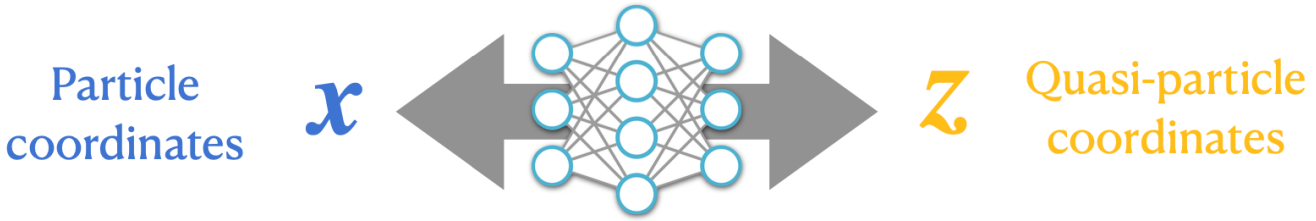
$$\langle \Psi_m \,|\, \Psi_n \rangle = \delta_{mn}$$

How to represent them ??

Use TWO deep generative models !!

Square root of normalizing flow

$$\sqrt{\text{Normalizing flow}}$$



$$\Psi_n(x) = \Phi_n(z) \cdot \left| \det \left(\frac{\partial z}{\partial x} \right) \right|^{\frac{1}{2}}$$

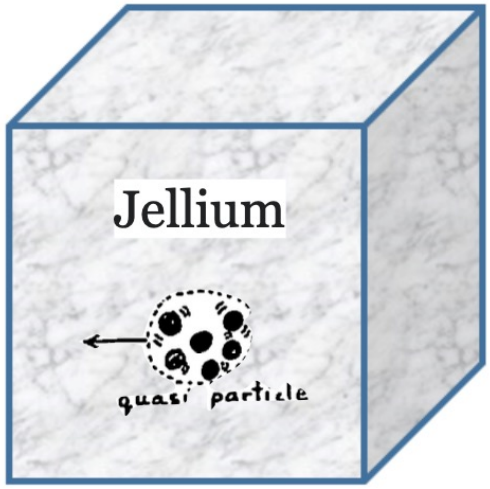
Target states | Base states | Jacobian of the flow

The flow implements a *learnable* many-body unitary transformation

hence the name "neural canonical transformation" a classical generalization of Li, Dong, Zhang, LW, PRX '20

Applications

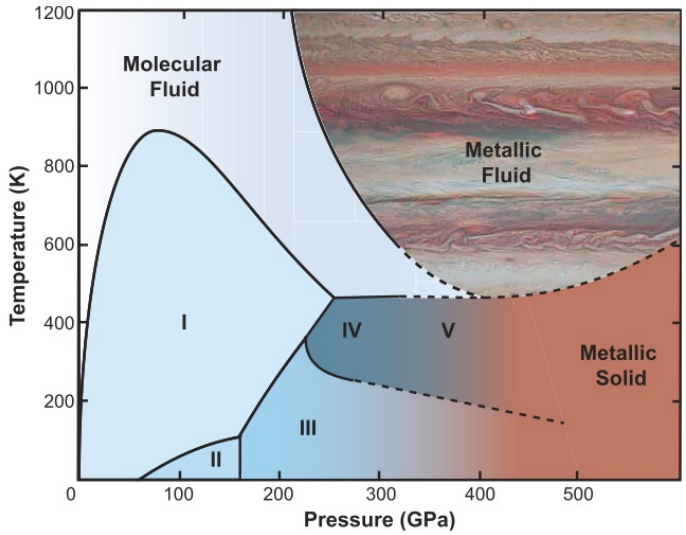
Uniform electron gas



Xie, Zhang, LW, 2201.03156

Dense hydrogen

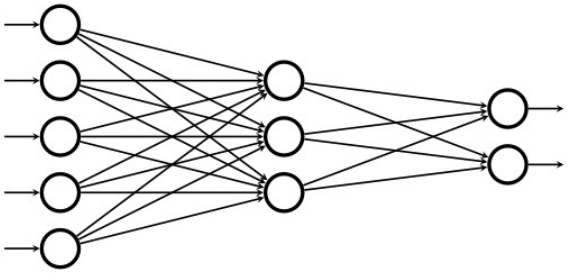
Gregoryanz et al, Matter Radiat. Extremes, 2020



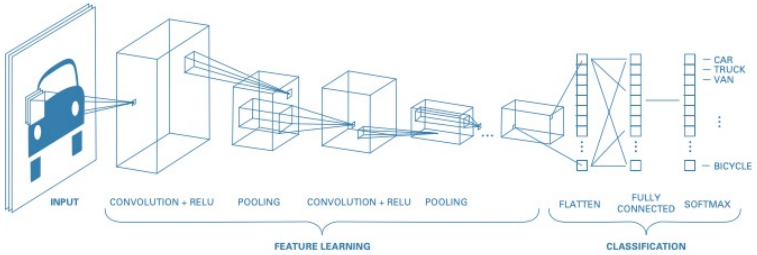
Xie, Li, Wang, Zhang, LW, 2209.06095

Models

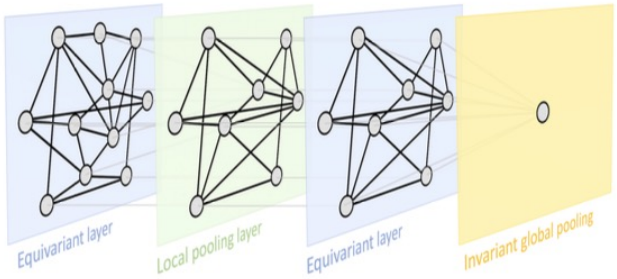
Multi-layer perceptron



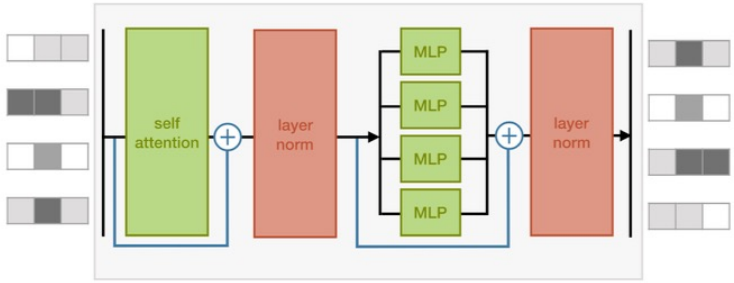
Convolutional neural network



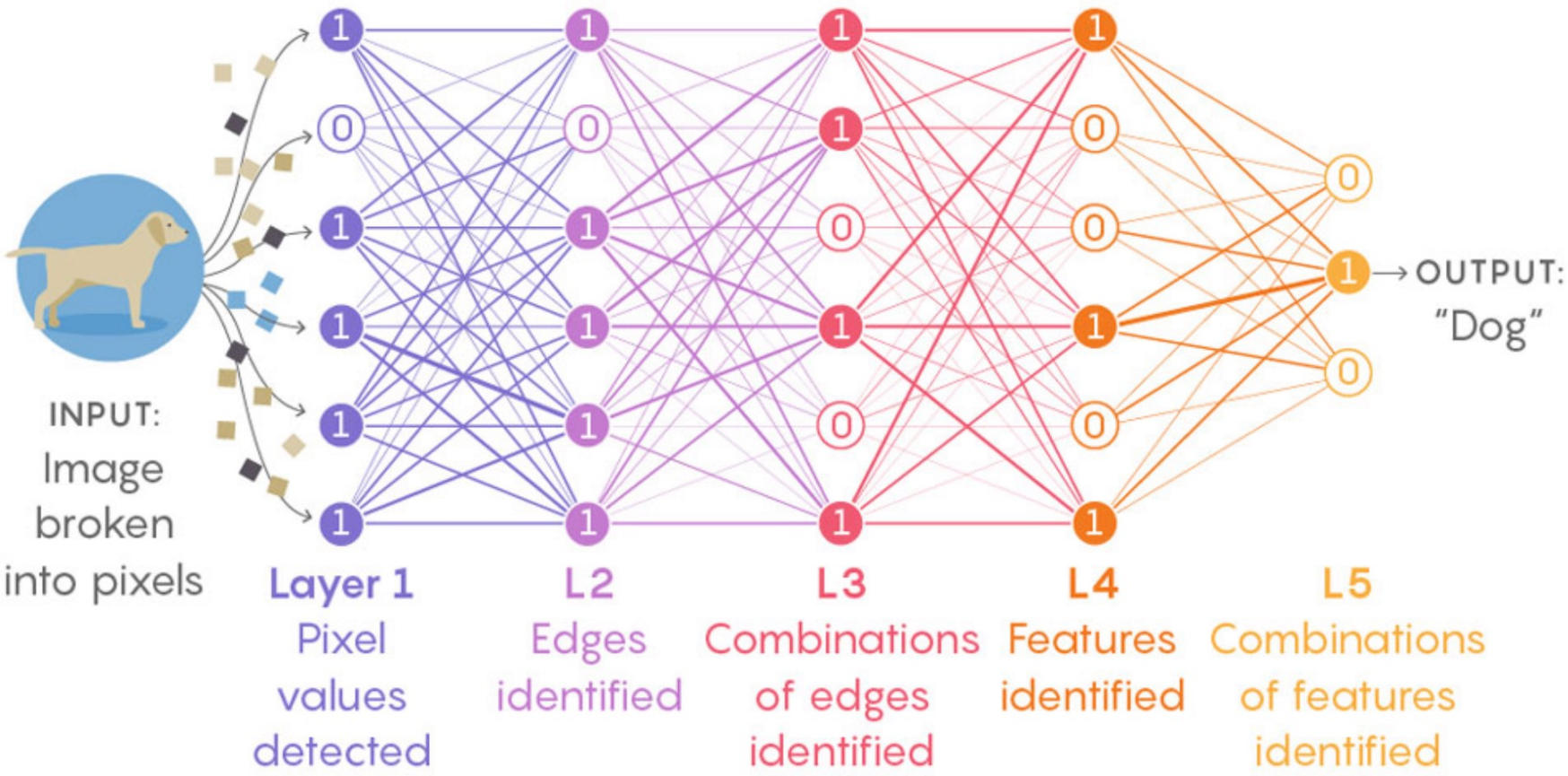
Graph neural network



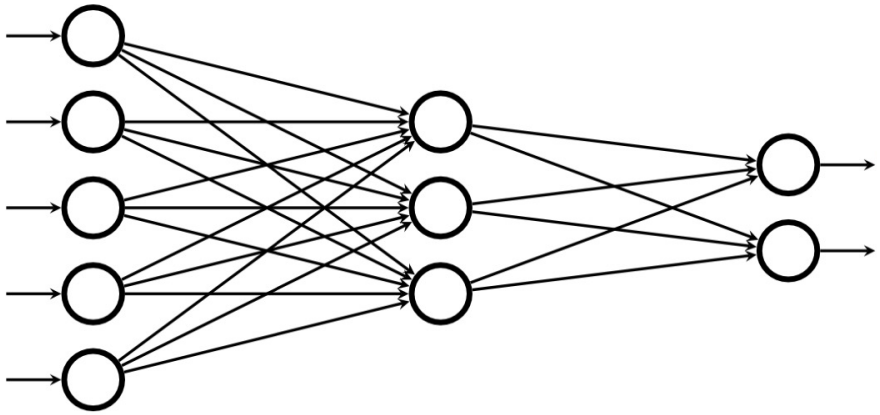
Transformer



Multiple-layer perceptrons



Multiple-layer perceptrons

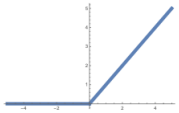


$$\boldsymbol{y} = \sigma(W\boldsymbol{x} + b)$$

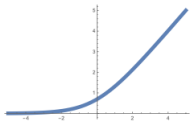
$$\boldsymbol{x} \in \mathbb{R}^{F_x} \qquad W \in \mathbb{R}^{F_y \times F_x} \qquad b \in \mathbb{R}^{F_y}$$

σ: elementwise activation

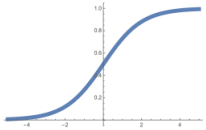
ReLU = $\max(0,x)$



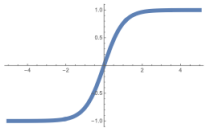
Softplus = $\ln(1 + e^x)$



Sigmoid = $1/(1 + e^{-x})$



tanh

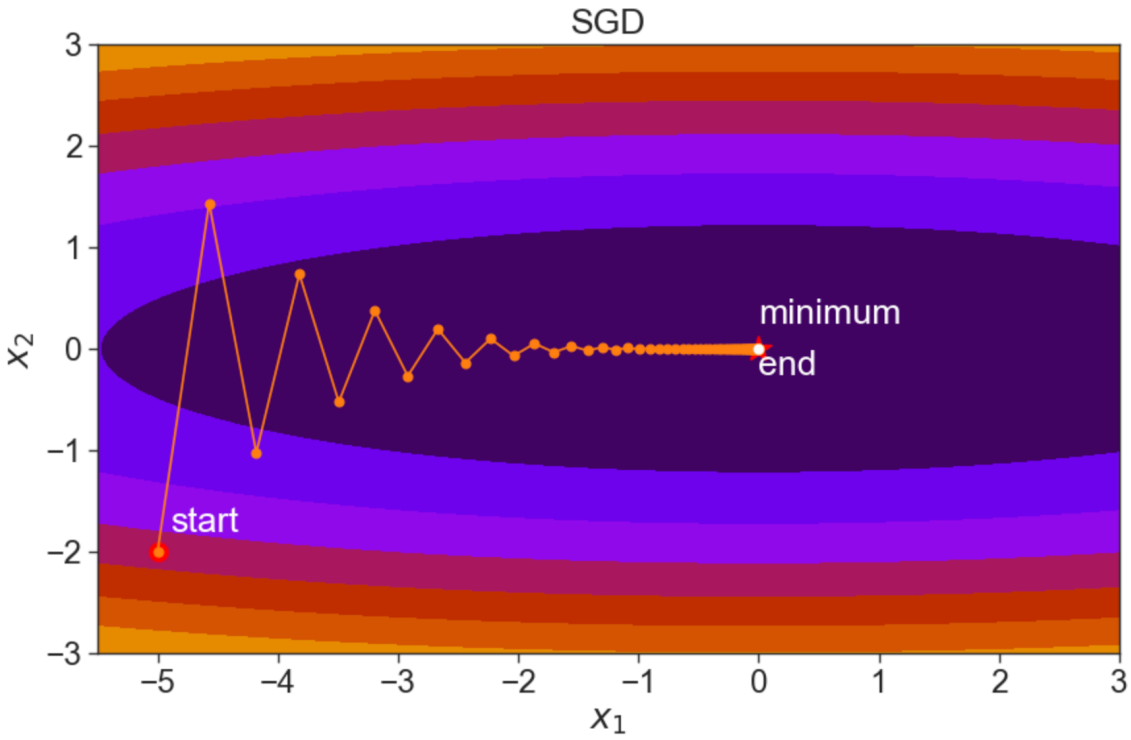


Optimization

Stochastic gradient descent

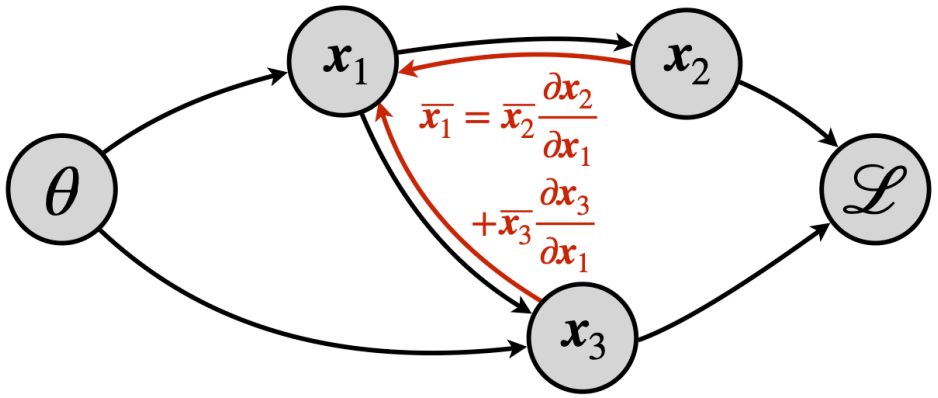
$$\theta \leftarrow \theta - \eta \frac{\partial \mathscr{L}}{\partial \theta}$$

$$\mathscr{L} = \mathop{\mathbb{E}}_{x} \left[\ell(x) \right]$$



Automatic differentiation

directed
acyclic graph



$$\overline{x_i} = \sum_{j:\,\text{child of } i} \overline{x_j}\frac{\partial x_j}{\partial x_i} \qquad \text{with} \quad \overline{\mathscr{L}} = 1$$

Message passing for the adjoint at each node

Reverse mode AD

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathscr{L}}{\partial \boldsymbol{x}_n}\frac{\partial \boldsymbol{x}_n}{\partial \boldsymbol{x}_{n-1}}\cdots\frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{x}_1}\frac{\partial \boldsymbol{x}_1}{\partial \boldsymbol{\theta}}$$

Reverse mode AD: Vector-Jacobian Product of primitives

- Backtrace the computation graph
- Needs to store intermediate results
- Efficient for graphs with large fan-in

Backpropagation = Reverse mode AD applied to neural networks

Forward mode AD

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}} = \underleftarrow{\frac{\partial \mathscr{L}}{\partial \boldsymbol{x}_n}\frac{\partial \boldsymbol{x}_n}{\partial \boldsymbol{x}_{n-1}}\cdots\frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{x}_1}\frac{\partial \boldsymbol{x}_1}{\partial \boldsymbol{\theta}}}$$

Forward mode AD: Jacobian-Vector Product of primitives

- Same order with the function evaluation
- No storage overhead
- Efficient for graph with large fan-out

Less efficient for scalar output, but useful for higher-order derivatives

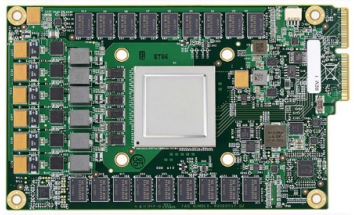
Differentiable programming tools

HIPS/autograd

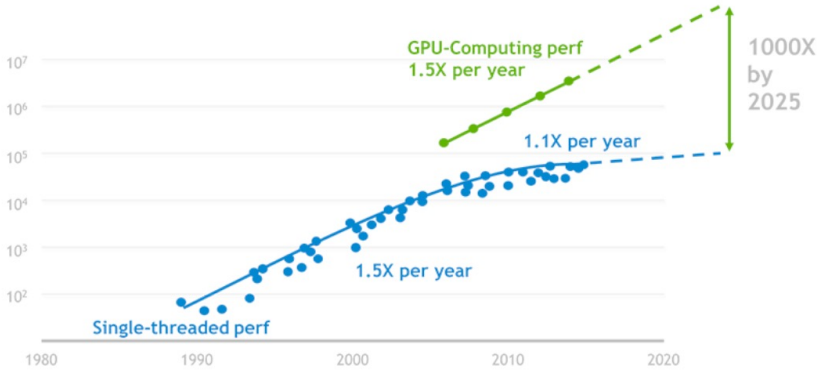
Hardware accelerators

for massive homogeneous computations (e.g. matrix multiplication)

GPU



TPU



The Hardware Lottery

Sara Hooker

Google Research, Brain Team
shooker@google.com

JAX

- NumPy and SciPy compatible
- Automatic differentiation (**grad**)
- Just-in-time compilation (**jit**)
- Automatic vectorization (**vmap**)
- Code transformations are composable
- Actively developed by Google
- Gaining a lot of popularity among ML and science researchers

Monte Carlo sampling

算法 1 Metropolis-Hastings sampling algorithm

Input: Initial state $\boldsymbol{x}_{\text{init}}$, total steps N.

Output: Final state $\boldsymbol{x}_{\text{final}}$.

$\boldsymbol{x} = \boldsymbol{x}_{\text{init}}$

for $i = 1, \cdots, N$ **do**

  Sample new state $\boldsymbol{x}'$ from the proposal probability $g(\boldsymbol{x} \to \boldsymbol{x}')$

  $u \sim \text{Uniform}(0, 1)$

  if $u < \min\left(1, \frac{p(\boldsymbol{x}')g(\boldsymbol{x}' \to \boldsymbol{x})}{p(\boldsymbol{x})g(\boldsymbol{x} \to \boldsymbol{x}')}\right)$ **then**

    $\boldsymbol{x} = \boldsymbol{x}'$ ▷ accept new sample

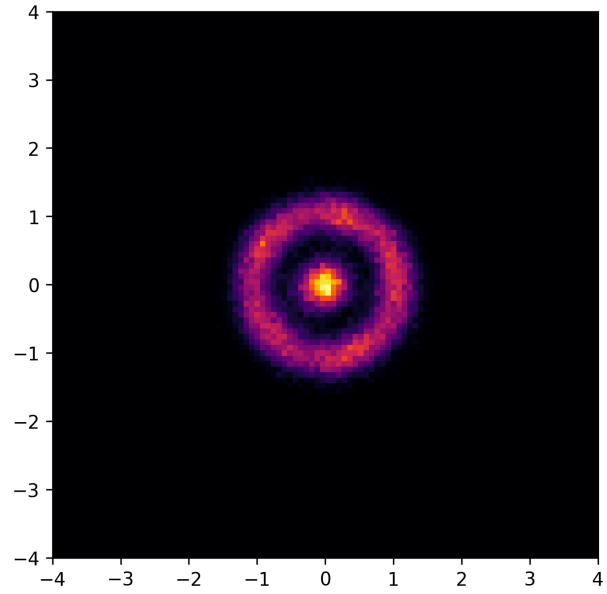
  end if

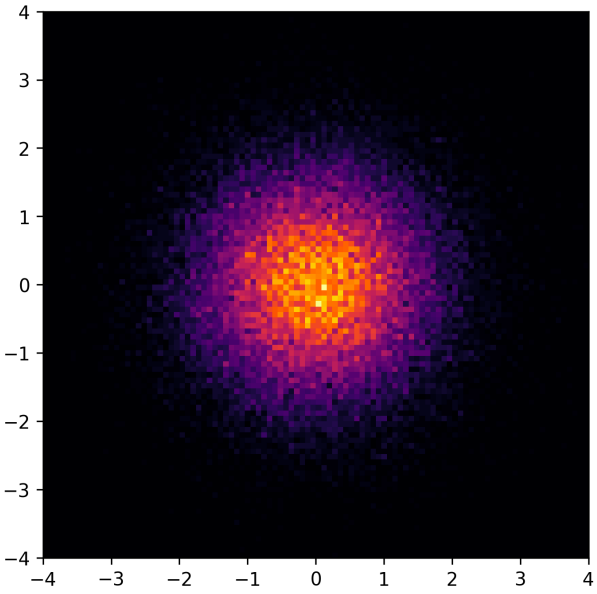
end for

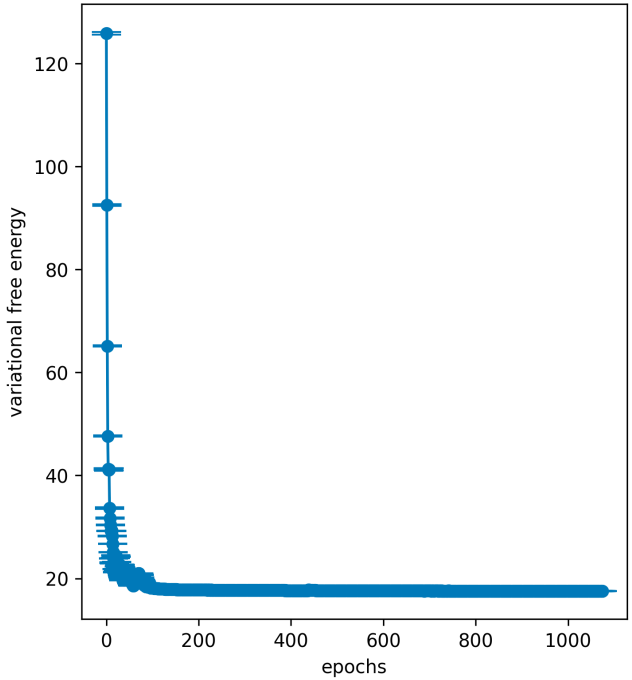
return $\boldsymbol{x}$

Some questions

epoch: 1074 F: 17.534191 (0.001373) E: 17.377776 (0.003067) S: -1.564144 (0.027247)

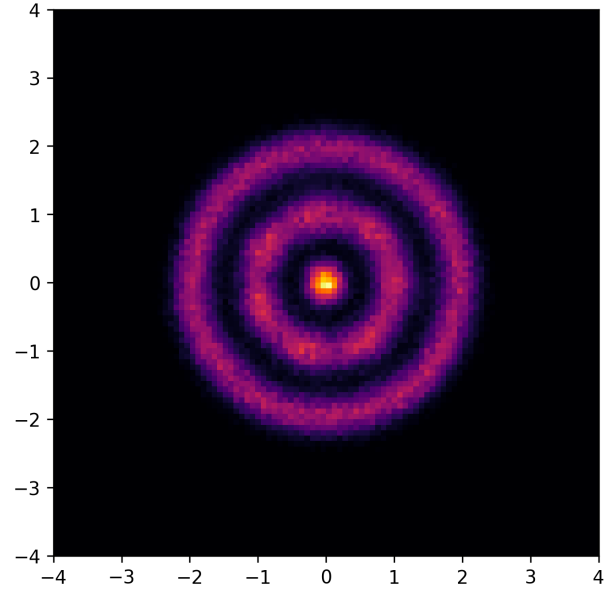


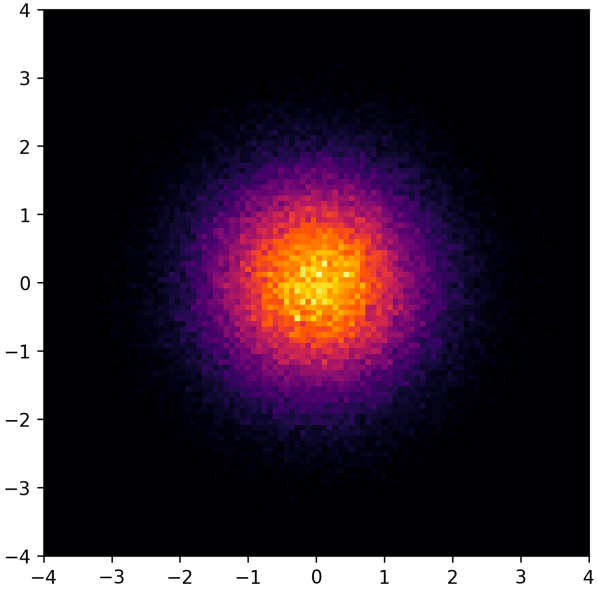


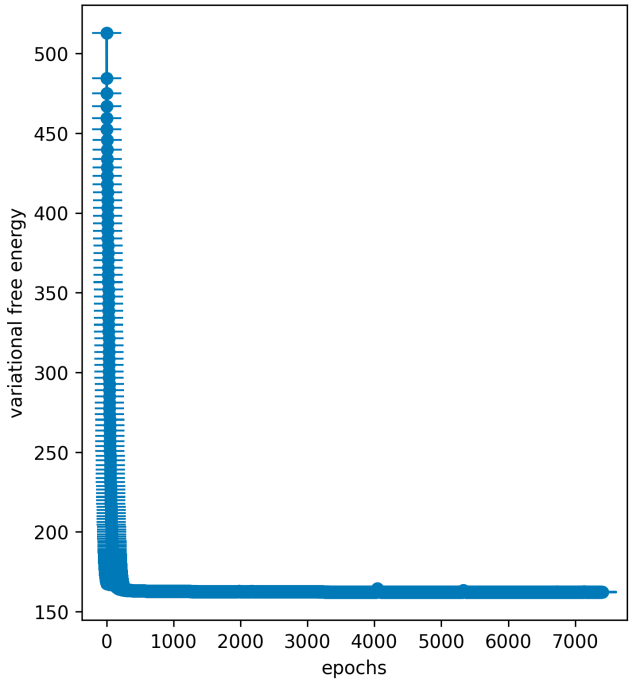


Some questions

epoch: 7404 F: 162.236320 (0.003012) E: 160.854909 (0.005717) S: -13.814109 (0.047445)







Thanks for your attention!