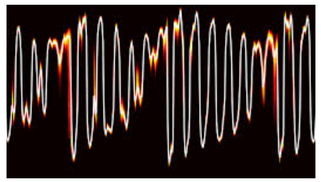
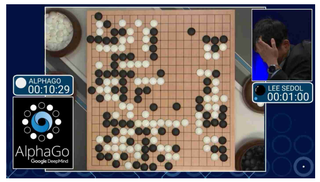
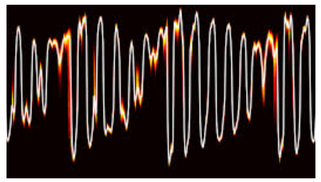
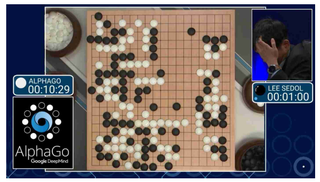
What is the secret behind deep learning ?





What is the secret behind deep learning ?

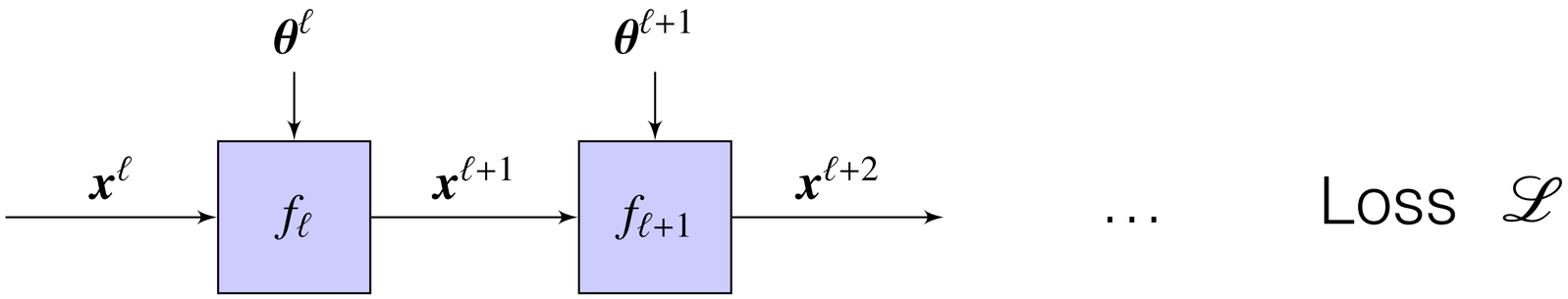




Key technology: differentiable programing

Core idea: leanring representation

The engine of deep learning: Back-Propagation algorithm

computation graph



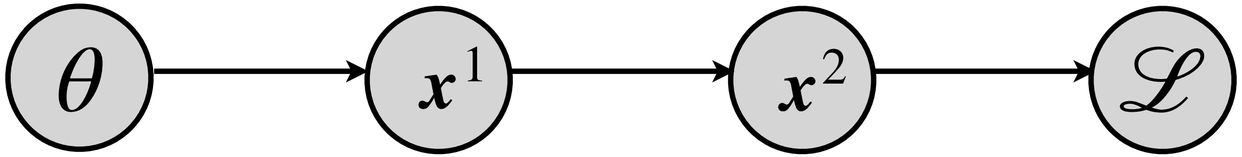
$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{x}^{\ell}} = \frac{\partial \mathscr{L}}{\partial \boldsymbol{x}^{\ell+1}} \left(\frac{\partial \boldsymbol{x}^{\ell+1}}{\partial \boldsymbol{x}^{\ell}} \right) \qquad \frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}^{\ell}} = \frac{\partial \mathscr{L}}{\partial \boldsymbol{x}^{\ell+1}} \left(\frac{\partial \boldsymbol{x}^{\ell+1}}{\partial \boldsymbol{\theta}^{\ell}} \right)$$

Computes gradients efficiently & accurately via reverse mode automatic differentiation

Computation Graph

Forward: function evaluation

Chain graph



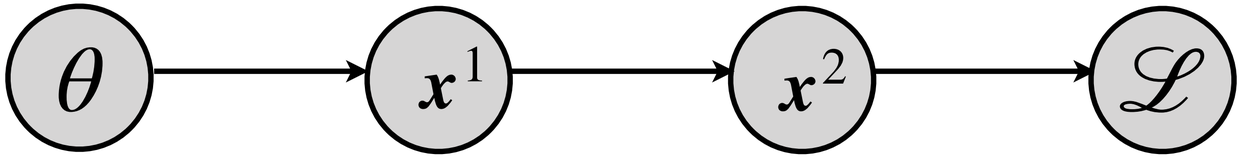
For backward, define "adjoint" $\bar{x} = \frac{\partial \mathscr{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Forward: function evaluation

Chain graph



$$\overline{\mathscr{L}} = 1$$

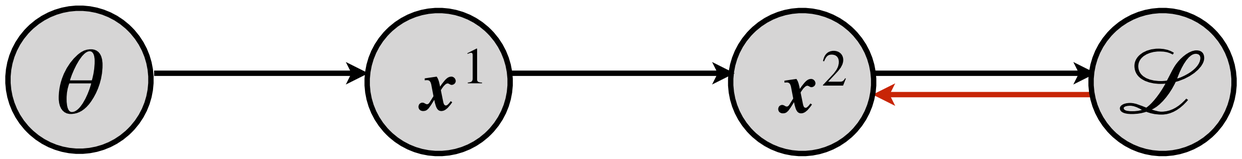
For backward, define "adjoint" $\bar{x} = \dfrac{\partial \mathscr{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Forward: function evaluation

Chain graph



$$\overline{x^2} = \overline{\mathscr{L}} \frac{\partial \mathscr{L}}{\partial x^2} \qquad \overline{\mathscr{L}} = 1$$

For backward, define "adjoint" $\bar{x} = \dfrac{\partial \mathscr{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Forward: function evaluation

Chain graph

$$\overline{x^1} = \overline{x^2}\frac{\partial x^2}{\partial x^1} \qquad \overline{x^2} = \overline{\mathscr{L}}\frac{\partial \mathscr{L}}{\partial x^2} \qquad \overline{\mathscr{L}} = 1$$

For backward, define “adjoint” $\bar{x} = \dfrac{\partial \mathscr{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Forward: function evaluation

Chain graph

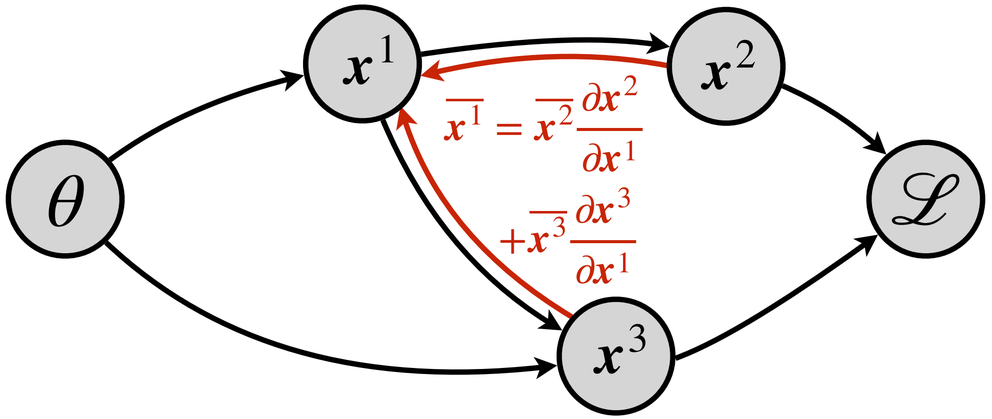
$$\bar{\theta} = \overline{x^1}\frac{\partial x^1}{\partial \theta} \qquad \overline{x^1} = \overline{x^2}\frac{\partial x^2}{\partial x^1} \qquad \overline{x^2} = \overline{\mathscr{L}}\frac{\partial \mathscr{L}}{\partial x^2} \qquad \overline{\mathscr{L}} = 1$$

For backward, define “adjoint” $\bar{x} = \dfrac{\partial \mathscr{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Directed
acyclic graph

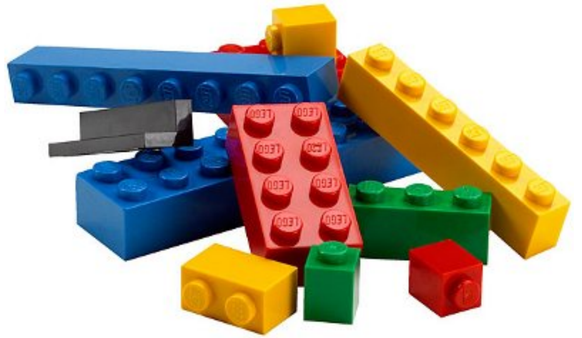


$$\overline{x^i} = \sum_{j:\,\text{child of } i} \overline{x^j}\,\frac{\partial x^j}{\partial x^i} \qquad \text{with} \quad \overline{\mathscr{L}} = 1$$

Efficient computes gradient of arbitrary program

How to think about AD ?

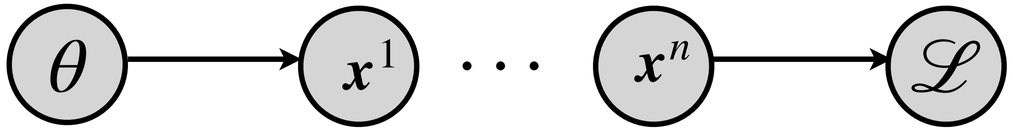
- AD is modular, and one can control its granularity
- Advantage of customized primitives
  - Reducing memory usage
  - Increasing numerical stability
  - Call to external libraries



Functional programing and differential geometry enthusiasm

https://colab.research.google.com/github/google/jax/blob/master/notebooks/autodiff_cookbook.ipynb

Reverse versus forward mode AD

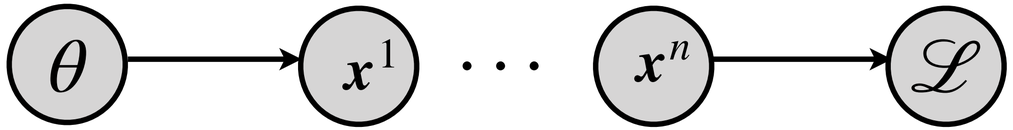


$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^n} \frac{\partial \boldsymbol{x}^n}{\partial \boldsymbol{x}^{n-1}} \cdots \frac{\partial \boldsymbol{x}^2}{\partial \boldsymbol{x}^1} \frac{\partial \boldsymbol{x}^1}{\partial \boldsymbol{\theta}}$$

Reverse mode AD: Vector-Jacobian product

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

Reverse versus forward mode AD



$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathscr{L}}{\partial \boldsymbol{x}^n} \frac{\partial \boldsymbol{x}^n}{\partial \boldsymbol{x}^{n-1}} \cdots \frac{\partial \boldsymbol{x}^2}{\partial \boldsymbol{x}^1} \frac{\partial \boldsymbol{x}^1}{\partial \boldsymbol{\theta}}$$

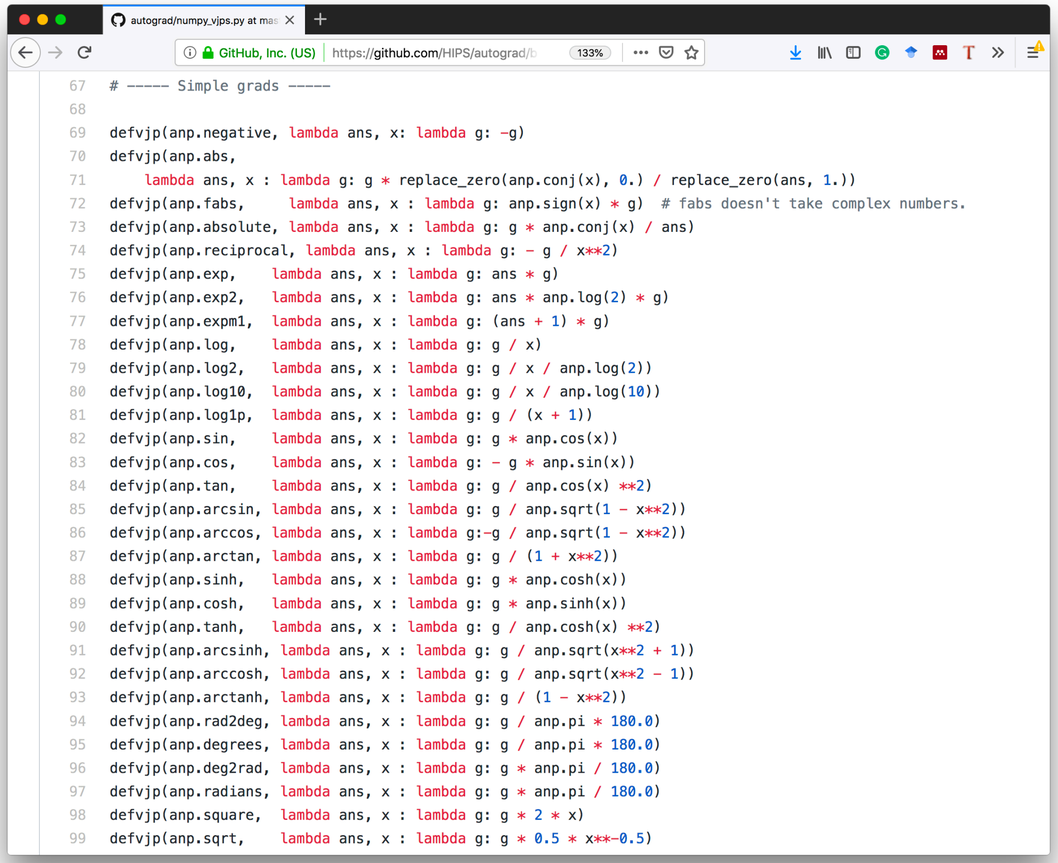
Forward mode AD: Jacobian-Vector Product

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

Examples of customized primitives

Forward	Backward
$y = xW$	$\bar{x} = \bar{y}W^T \quad \overline{W} = x^T\bar{y}$
$y = \sigma(x)$	$\bar{x} = \bar{y} \odot \sigma'(x)$
…	…

Examples of customized primitives




```
# ----- Simple grads -----

defvjp(anp.negative, lambda ans, x: lambda g: -g)
defvjp(anp.abs,
    lambda ans, x : lambda g: g * replace_zero(anp.conj(x), 0.) / replace_zero(ans, 1.))
defvjp(anp.fabs,     lambda ans, x : lambda g: anp.sign(x) * g)  # fabs doesn't take complex numbers.
defvjp(anp.absolute, lambda ans, x : lambda g: g * anp.conj(x) / ans)
defvjp(anp.reciprocal, lambda ans, x : lambda g: - g / x**2)
defvjp(anp.exp,    lambda ans, x : lambda g: ans * g)
defvjp(anp.exp2,   lambda ans, x : lambda g: ans * anp.log(2) * g)
defvjp(anp.expm1,  lambda ans, x : lambda g: (ans + 1) * g)
defvjp(anp.log,    lambda ans, x : lambda g: g / x)
defvjp(anp.log2,   lambda ans, x : lambda g: g / x / anp.log(2))
defvjp(anp.log10,  lambda ans, x : lambda g: g / x / anp.log(10))
defvjp(anp.log1p,  lambda ans, x : lambda g: g / (x + 1))
defvjp(anp.sin,    lambda ans, x : lambda g: g * anp.cos(x))
defvjp(anp.cos,    lambda ans, x : lambda g: - g * anp.sin(x))
defvjp(anp.tan,    lambda ans, x : lambda g: g / anp.cos(x) **2)
defvjp(anp.arcsin, lambda ans, x : lambda g: g / anp.sqrt(1 - x**2))
defvjp(anp.arccos, lambda ans, x : lambda g:-g / anp.sqrt(1 - x**2))
defvjp(anp.arctan, lambda ans, x : lambda g: g / (1 + x**2))
defvjp(anp.sinh,   lambda ans, x : lambda g: g * anp.cosh(x))
defvjp(anp.cosh,   lambda ans, x : lambda g: g * anp.sinh(x))
defvjp(anp.tanh,   lambda ans, x : lambda g: g / anp.cosh(x) **2)
defvjp(anp.arcsinh, lambda ans, x : lambda g: g / anp.sqrt(x**2 + 1))
defvjp(anp.arccosh, lambda ans, x : lambda g: g / anp.sqrt(x**2 - 1))
defvjp(anp.arctanh, lambda ans, x : lambda g: g / (1 - x**2))
defvjp(anp.rad2deg, lambda ans, x : lambda g: g / anp.pi * 180.0)
defvjp(anp.degrees, lambda ans, x : lambda g: g / anp.pi * 180.0)
defvjp(anp.deg2rad, lambda ans, x : lambda g: g * anp.pi / 180.0)
defvjp(anp.radians, lambda ans, x : lambda g: g * anp.pi / 180.0)
defvjp(anp.square,  lambda ans, x : lambda g: g * 2 * x)
defvjp(anp.sqrt,    lambda ans, x : lambda g: g * 0.5 * x**-0.5)
```

Backward

$$\bar{x} = \bar{y}W^T \quad \overline{W} = x^T\bar{y}$$

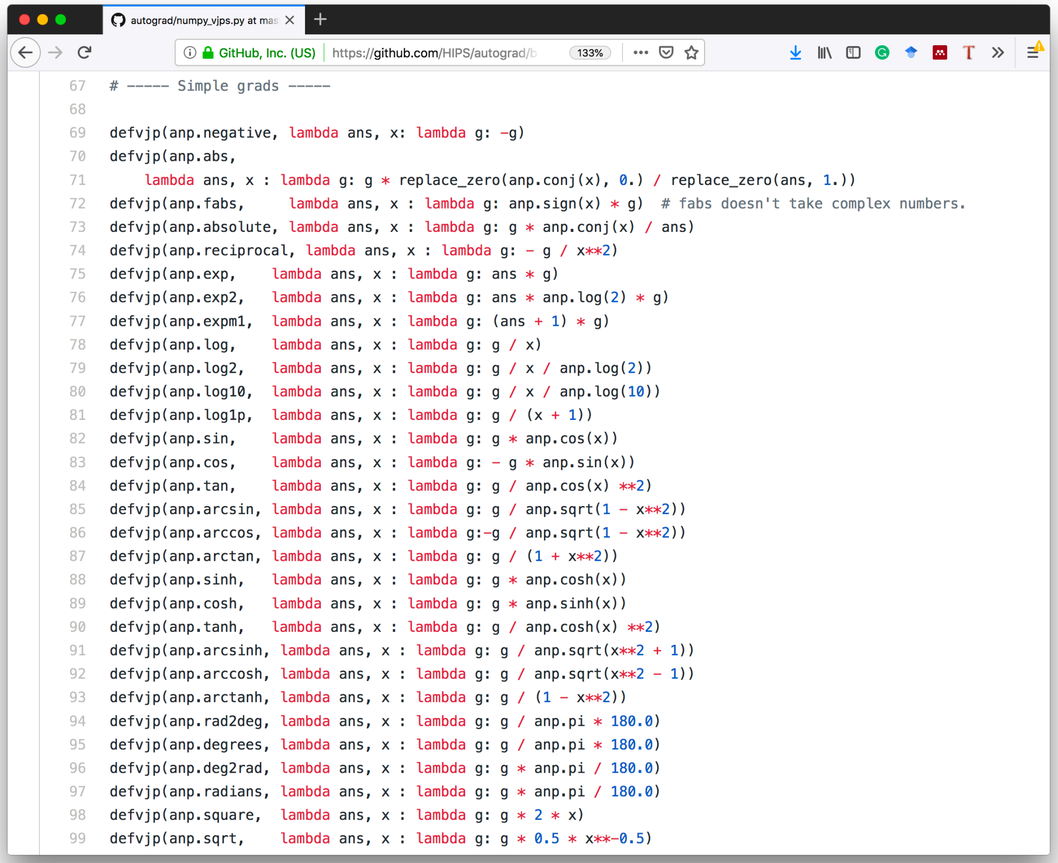
$$\bar{x} = \bar{y} \odot \sigma'(x)$$

...

https://github.com/HIPS/autograd/blob/master/autograd/numpy/numpy_vjps.py

~200 functions to cover most of numpy in HIPS/autograd

Examples of customized primitives




```
# ----- Simple grads -----

defvjp(anp.negative, lambda ans, x: lambda g: -g)
defvjp(anp.abs,
    lambda ans, x : lambda g: g * replace_zero(anp.conj(x), 0.) / replace_zero(ans, 1.))
defvjp(anp.fabs,     lambda ans, x : lambda g: anp.sign(x) * g)  # fabs doesn't take complex numbers.
defvjp(anp.absolute, lambda ans, x : lambda g: g * anp.conj(x) / ans)
defvjp(anp.reciprocal, lambda ans, x : lambda g: - g / x**2)
defvjp(anp.exp,    lambda ans, x : lambda g: ans * g)
defvjp(anp.exp2,   lambda ans, x : lambda g: ans * anp.log(2) * g)
defvjp(anp.expm1,  lambda ans, x : lambda g: (ans + 1) * g)
defvjp(anp.log,    lambda ans, x : lambda g: g / x)
defvjp(anp.log2,   lambda ans, x : lambda g: g / x / anp.log(2))
defvjp(anp.log10,  lambda ans, x : lambda g: g / x / anp.log(10))
defvjp(anp.log1p,  lambda ans, x : lambda g: g / (x + 1))
defvjp(anp.sin,    lambda ans, x : lambda g: g * anp.cos(x))
defvjp(anp.cos,    lambda ans, x : lambda g: - g * anp.sin(x))
defvjp(anp.tan,    lambda ans, x : lambda g: g / anp.cos(x) **2)
defvjp(anp.arcsin, lambda ans, x : lambda g: g / anp.sqrt(1 - x**2))
defvjp(anp.arccos, lambda ans, x : lambda g:-g / anp.sqrt(1 - x**2))
defvjp(anp.arctan, lambda ans, x : lambda g: g / (1 + x**2))
defvjp(anp.sinh,   lambda ans, x : lambda g: g * anp.cosh(x))
defvjp(anp.cosh,   lambda ans, x : lambda g: g * anp.sinh(x))
defvjp(anp.tanh,   lambda ans, x : lambda g: g / anp.cosh(x) **2)
defvjp(anp.arcsinh, lambda ans, x : lambda g: g / anp.sqrt(x**2 + 1))
defvjp(anp.arccosh, lambda ans, x : lambda g: g / anp.sqrt(x**2 - 1))
defvjp(anp.arctanh, lambda ans, x : lambda g: g / (1 - x**2))
defvjp(anp.rad2deg, lambda ans, x : lambda g: g / anp.pi * 180.0)
defvjp(anp.degrees, lambda ans, x : lambda g: g / anp.pi * 180.0)
defvjp(anp.deg2rad, lambda ans, x : lambda g: g * anp.pi / 180.0)
defvjp(anp.radians, lambda ans, x : lambda g: g * anp.pi / 180.0)
defvjp(anp.square,  lambda ans, x : lambda g: g * 2 * x)
defvjp(anp.sqrt,    lambda ans, x : lambda g: g * 0.5 * x**-0.5)
```

Backward

$$\bar{x} = \bar{y}W^T \quad \overline{W} = x^T\bar{y}$$

$$\bar{x} = \bar{y} \odot \sigma'(x)$$

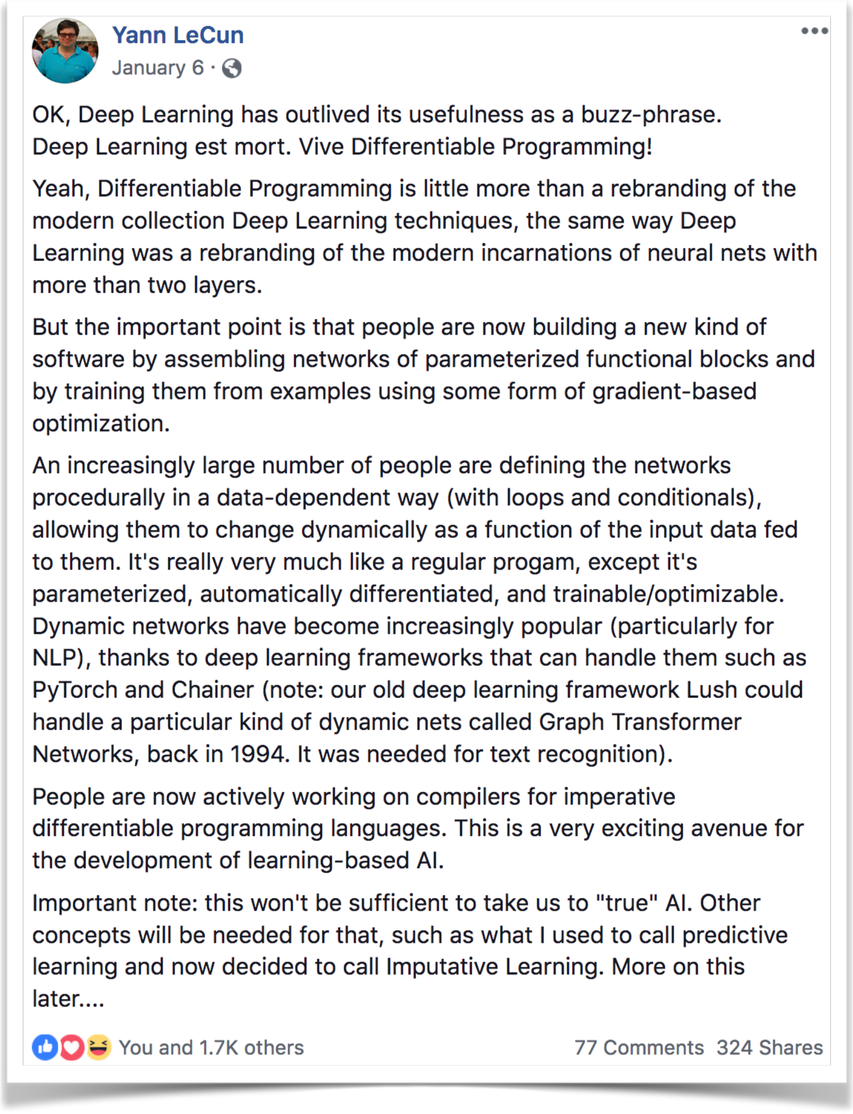
...

https://github.com/HIPS/autograd/blob/master/autograd/numpy/numpy_vjps.py

~200 functions to cover most of numpy in HIPS/autograd

Loop/Condition/Sort/Permutations are also differentiable

Differentiable Programming



Yann LeCun
January 6 ·

OK, Deep Learning has outlived its usefulness as a buzz-phrase.
Deep Learning est mort. Vive Differentiable Programming!

Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers.

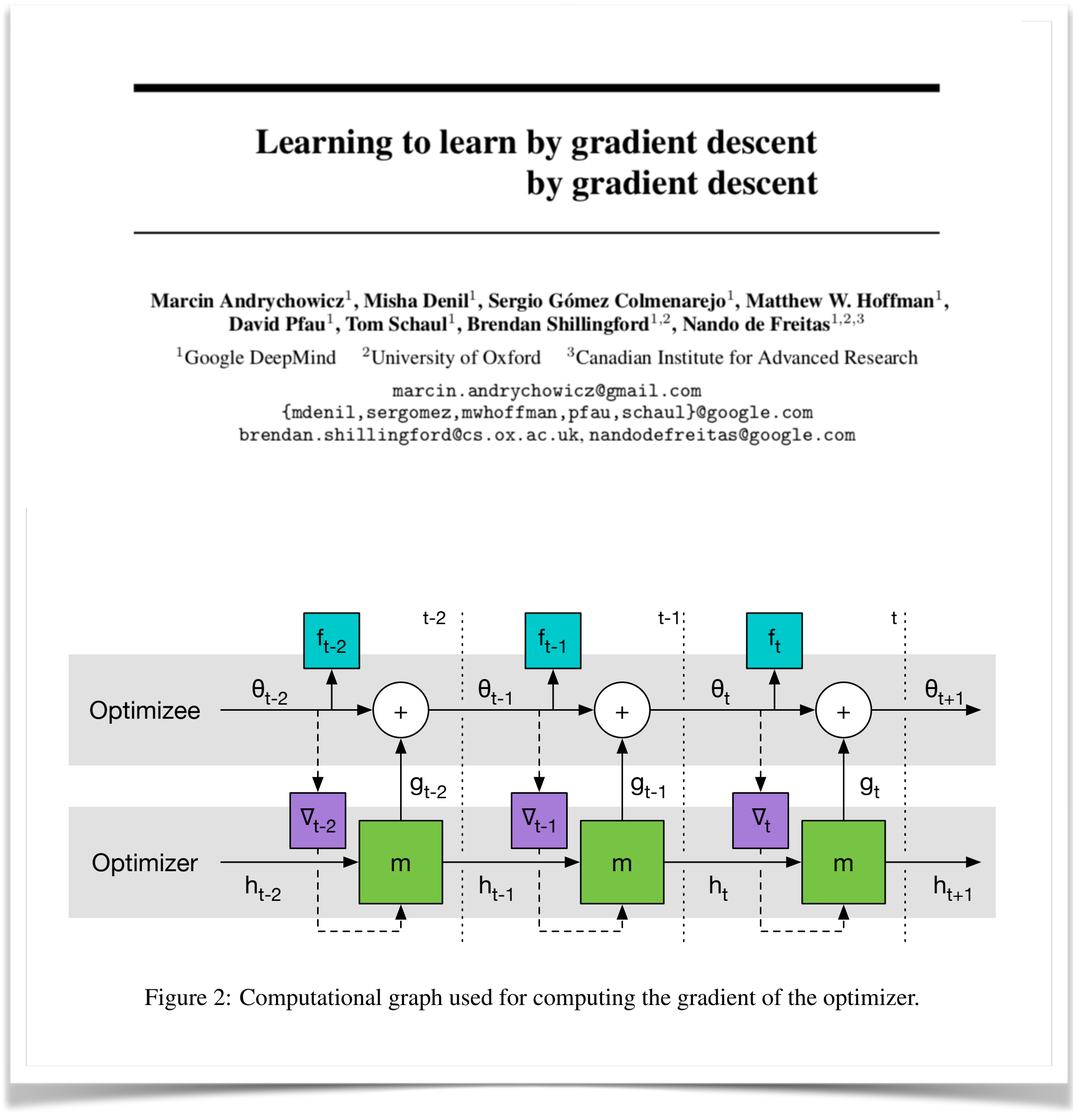
But the important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization.

An increasingly large number of people are defining the networks procedurally in a data-dependent way (with loops and conditionals), allowing them to change dynamically as a function of the input data fed to them. It's really very much like a regular progam, except it's parameterized, automatically differentiated, and trainable/optimizable. Dynamic networks have become increasingly popular (particularly for NLP), thanks to deep learning frameworks that can handle them such as PyTorch and Chainer (note: our old deep learning framework Lush could handle a particular kind of dynamic nets called Graph Transformer Networks, back in 1994. It was needed for text recognition).

People are now actively working on compilers for imperative differentiable programming languages. This is a very exciting avenue for the development of learning-based AI.

Important note: this won't be sufficient to take us to "true" AI. Other concepts will be needed for that, such as what I used to call predictive learning and now decided to call Imputative Learning. More on this later....

You and 1.7K others 77 Comments 324 Shares



Learning to learn by gradient descent by gradient descent

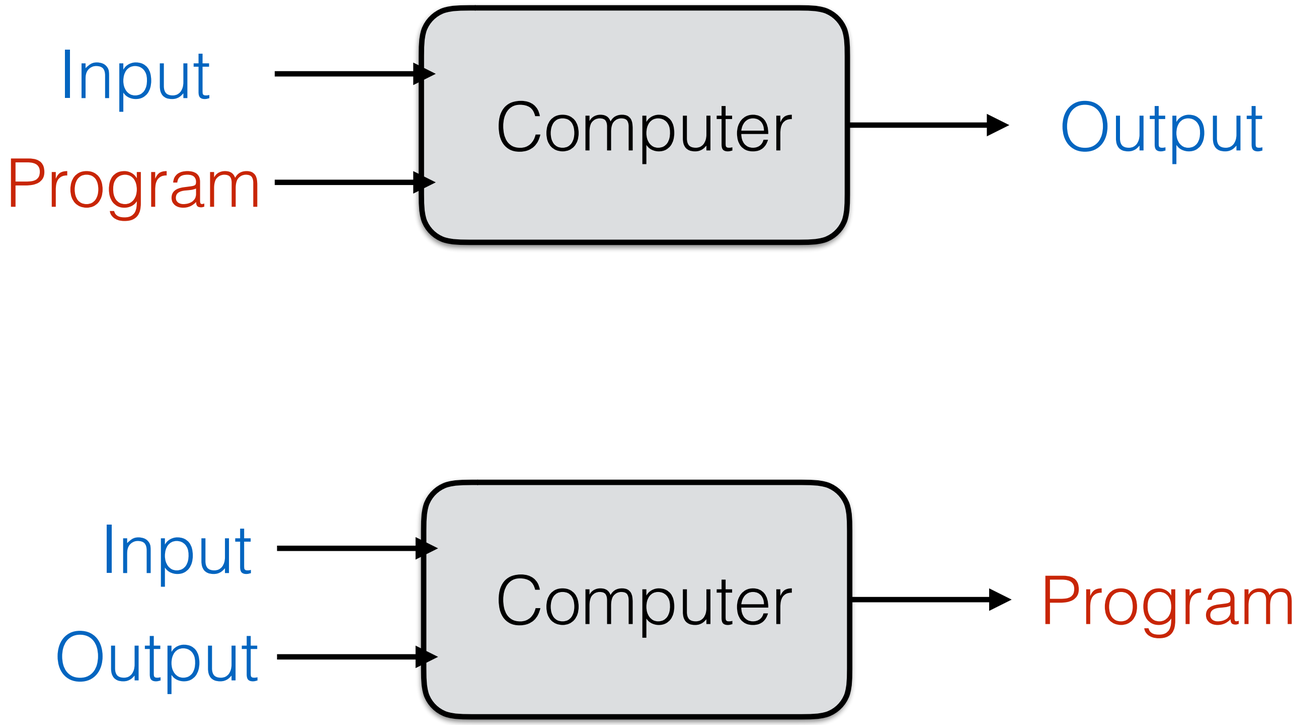
Marcin Andrychowicz[1], Misha Denil[1], Sergio Gómez Colmenarejo[1], Matthew W. Hoffman[1], David Pfau[1], Tom Schaul[1], Brendan Shillingford[1,2], Nando de Freitas[1,2,3]

[1]Google DeepMind [2]University of Oxford [3]Canadian Institute for Advanced Research

marcin.andrychowicz@gmail.com
{mdenil,sergomez,mwhoffman,pfau,schaul}@google.com
brendan.shillingford@cs.ox.ac.uk, nandodefreitas@google.com

Figure 2: Computational graph used for computing the gradient of the optimizer.

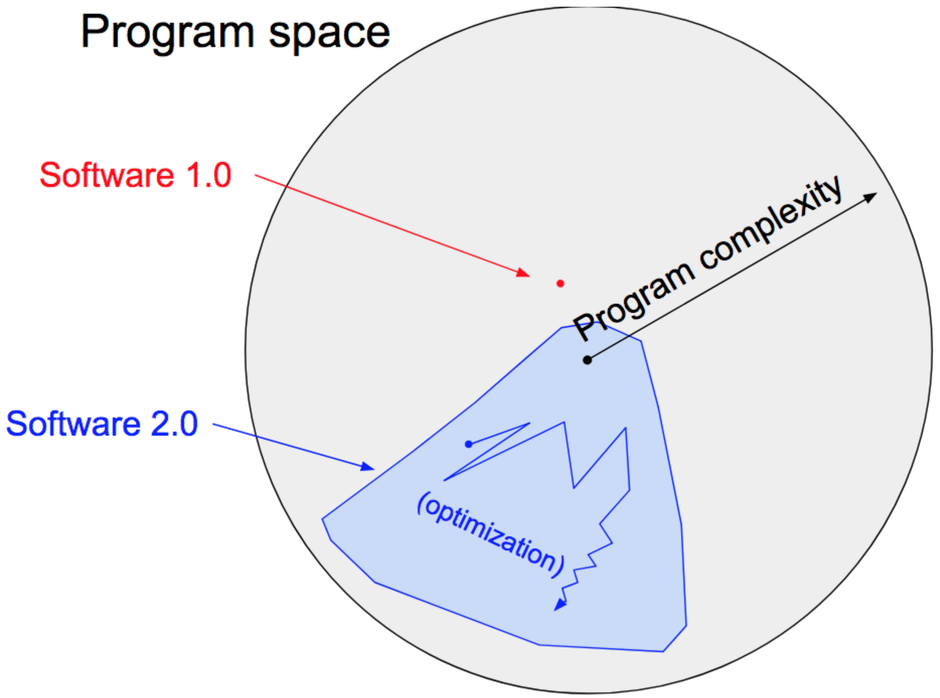
Differentiable Programming



Andrej Karpathy

Director of AI at Tesla. Previously Research Scientist at OpenAI and PhD student at Stanford. I like to train deep neural nets on large datasets.

https://medium.com/@karpathy/software-2-0-a64152b37c35



A new paradigm of programming computers

Software 2.0

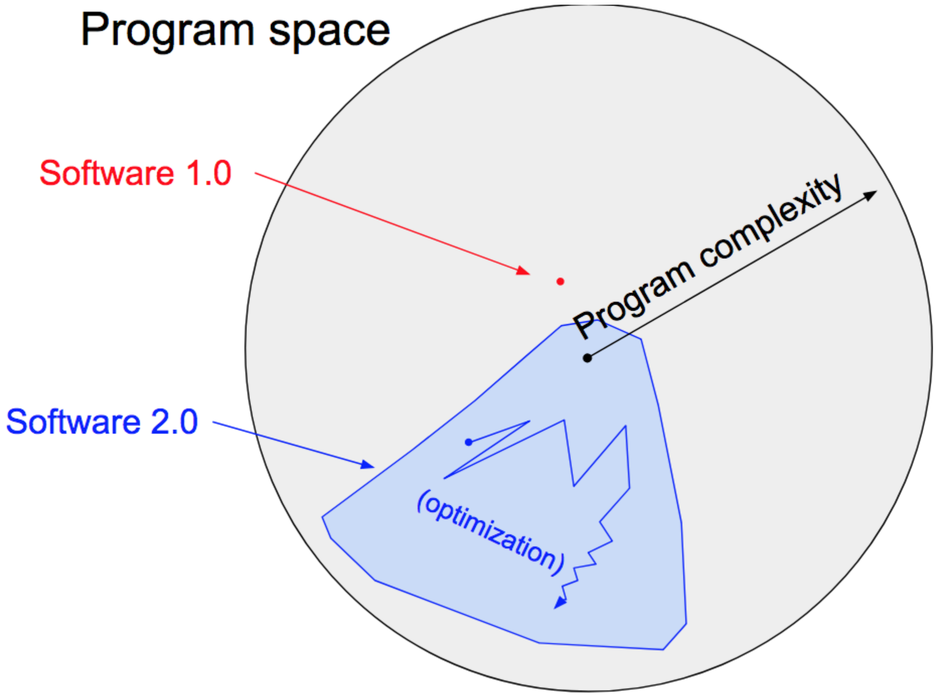
Benefits compared to 1.0

- Computationally homogeneous
- Simple to bake into silicon
- Constant running time
- Constant memory usage
- Highly portable & agile
- Modules can meld into an optimal whole
- **Better than humans**

Andrej Karpathy

Director of AI at Tesla. Previously Research Scientist at OpenAI and PhD student at Stanford. I like to train deep neural nets on large datasets.

https://medium.com/@karpathy/software-2-0-a64152b37c35



Writing software 2.0 by searching in the program space

Differentiable Scientific Programming

- Most linear algebra operations (Eigen, SVD!) are differentiable
- ODE integrators are differentiable with O(1) memory
- Differentiable ray tracer and Differentiable fluid simulations
- Differentiable Monte Carlo/Tensor Network/Functional RG/ Dynamical Mean Field Theory/Density Functional Theory…

Differentiable programming is more than training neural networks

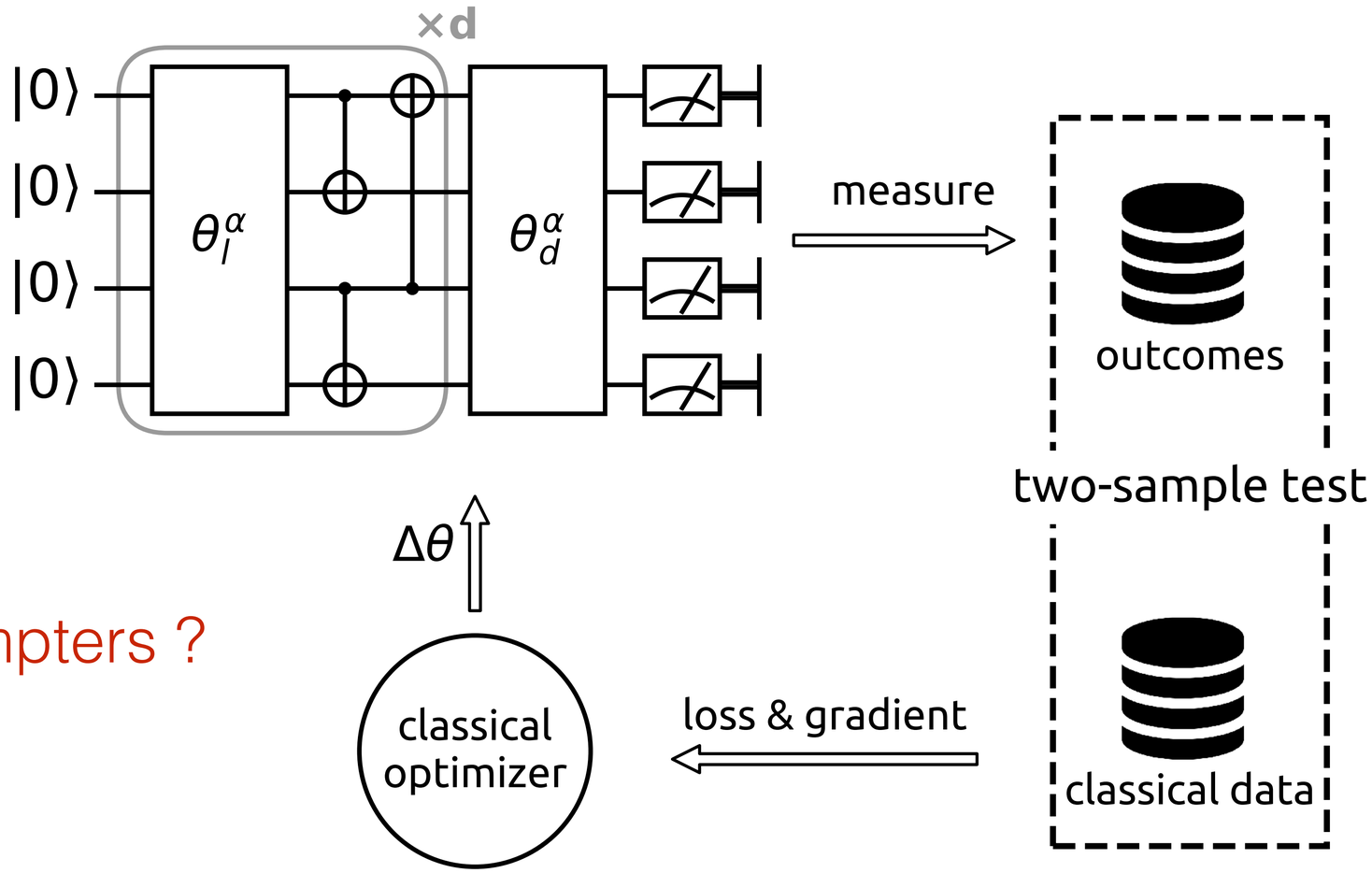
Differentiable Quantum Programming

Short term:

What can we do with circuits of limited depth ?

Long term:

Are we really good at programing quantum compters ?



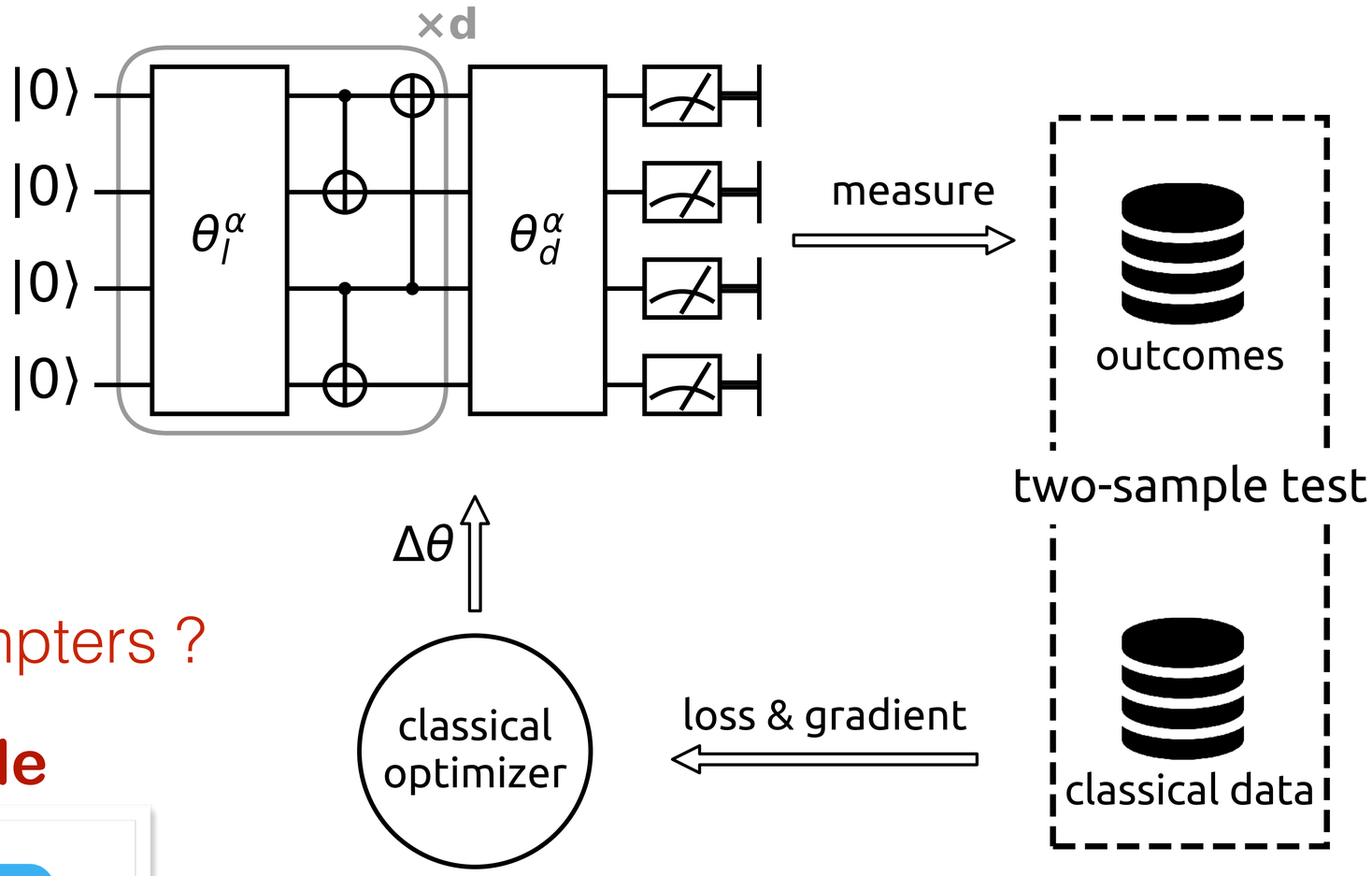
Differentiable Quantum Programming

Short term:

What can we do with circuits of limited depth ?

Long term:

Are we really good at programing quantum compters ?



Quantum code

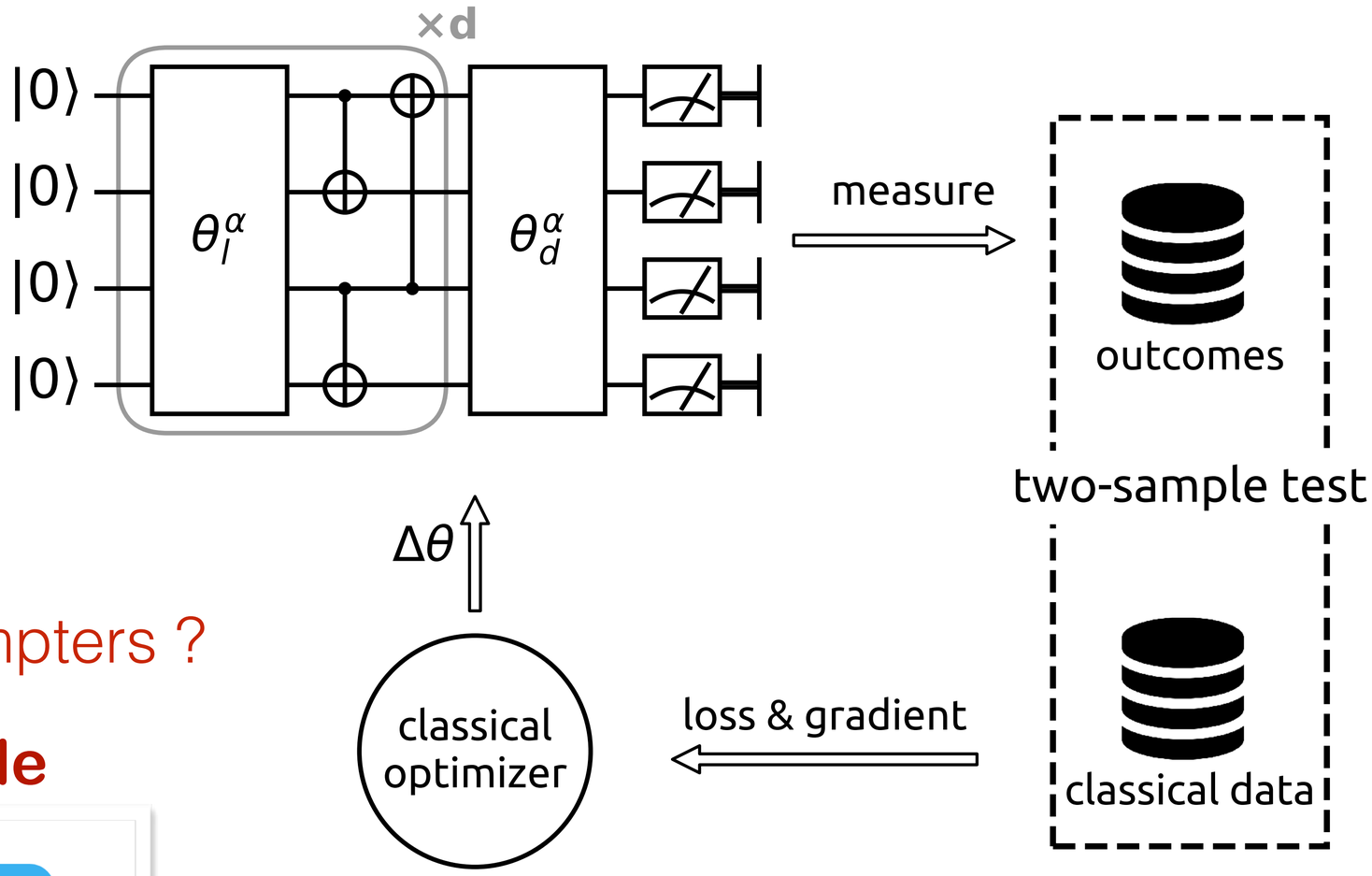
Differentiable Quantum Programming

Short term:

What can we do with circuits of limited depth ?

Long term:

Are we really good at programing quantum compters ?



Quantum code

Xiu-Zhe Luo, Jin-Guo Liu

https://github.com/QuantumBFS/Yao.jl/

Differentiable Eigensolver

$$H\Psi = \Psi\Lambda$$

Forward mode: What happen if $H = H + dH$? Perturbation theory

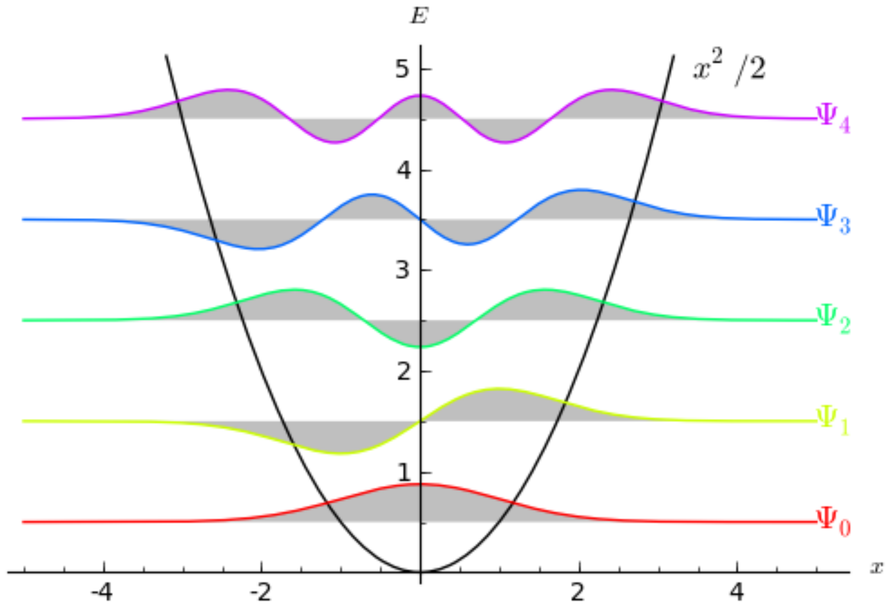
Reverse mode: How should I change H given $\partial\mathscr{L}/\partial\Psi$ and $\partial\mathscr{L}/\partial\Lambda$? **Inverse perturbation theory!**

Hamiltonian engineering via differentiable programming

Demo: Inverse Schrodinger Solver

Given a target ground state, how to design the potential ?

$$\left[-\frac{1}{2}\frac{\partial^2}{\partial x^2} + V(x)\right]\Psi(x) = \Lambda\Psi(x)$$



Deep learning tools

HIPS/autograd

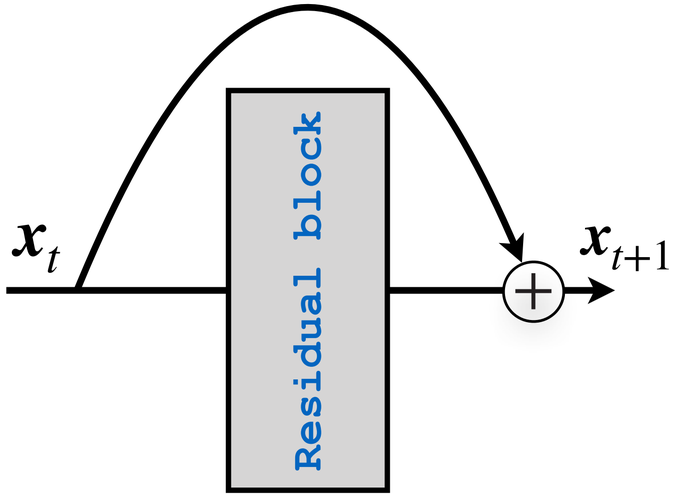
Current support for AD*

	Linear Algebra	Complex	GPU	Mixed-mode
PyTorch	✓	✗	✓	✗
TensorFlow	✓	✓	✓	✗
Autograd	✓	✓	✗	✗
Jax	✓	✓	✓	✓
Flux.jl/Zygote.jl	✗	✓	✓	✓

*as of May 2019

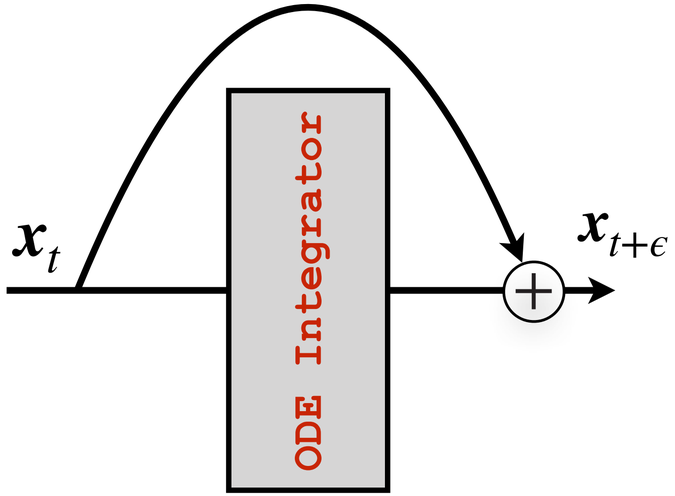
Neural Ordinary Differential Equations

Residual network



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

ODE integration



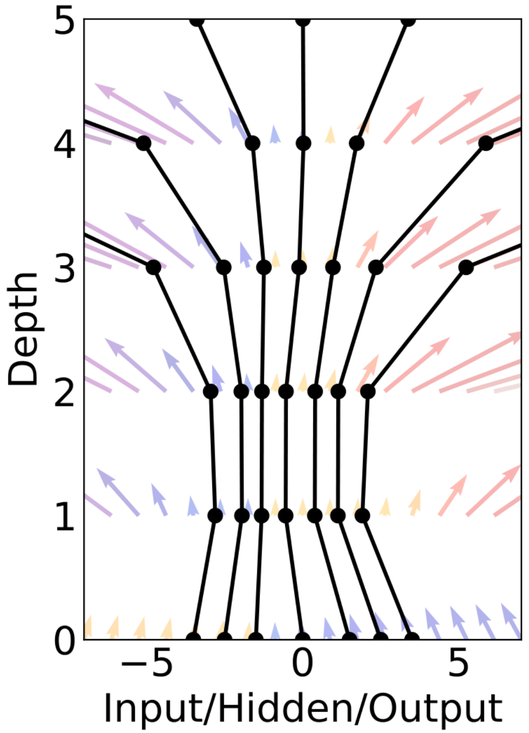
$$dx/dt = f(x)$$

Chen et al, 1806.07366 NIPS '18 Best paper award

cf Harbor el al 1705.03341
Lu et al 1710.10121, E 17'…

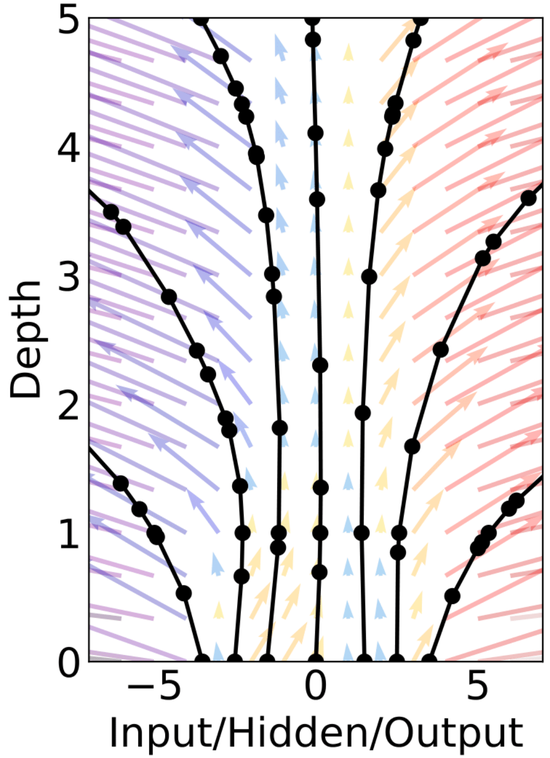
Neural Ordinary Differential Equations

Residual network



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

ODE integration



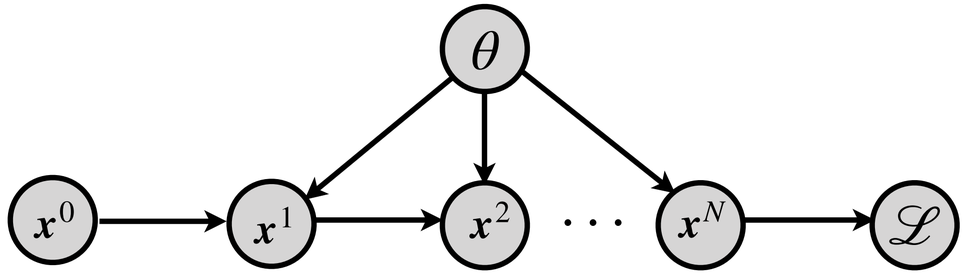
$$d\boldsymbol{x}/dt = f(\boldsymbol{x})$$

Chen et al, 1806.07366 NIPS ’18 Best paper award

cf Harbor el al 1705.03341
Lu et al 1710.10121, E 17’…

Backpropagate through ODE solver

$$\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x}, \boldsymbol{\theta}, t)$$



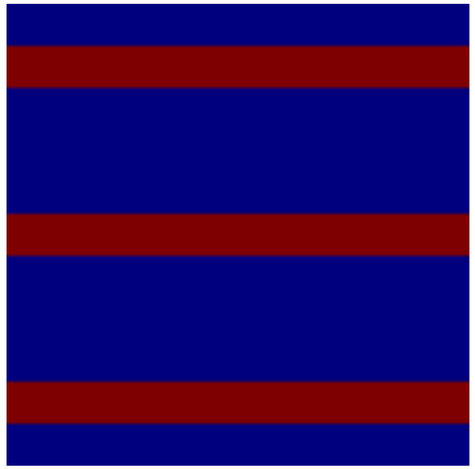
Adjoint $\bar{\boldsymbol{x}}(t) = \dfrac{\partial \mathscr{L}}{\partial \boldsymbol{x}(t)}$ satisfies another ODE

$$\frac{d\bar{\boldsymbol{x}}(t)}{dt} = -\bar{\boldsymbol{x}}(t)\frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta}, t)}{\partial \boldsymbol{x}}$$

Gradient w.r.t. parameter

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}} = \int_0^T dt\, \bar{\boldsymbol{x}}(t)\frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta}, t)}{\partial \boldsymbol{\theta}}$$

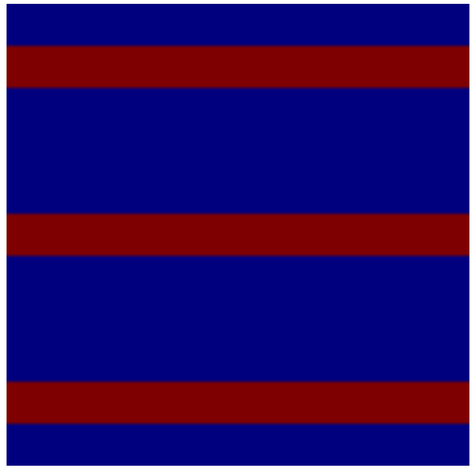
Why do we need Neural ODE ?



Backpropagating
through a fluid simulation

- Neural ODE has constant memory usage
- Works for black box ODE integrator
- Works with adaptive steps and implicit schemes

Why do we need Neural ODE ?

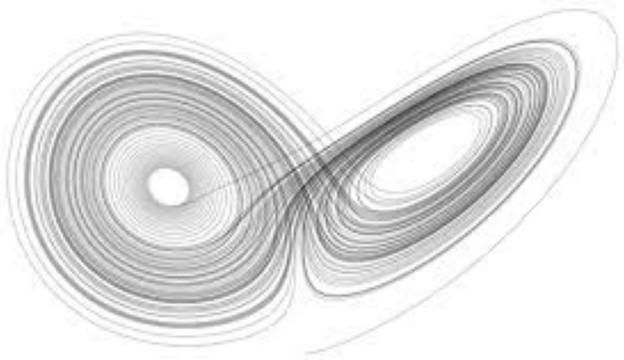


Backpropagating
through a fluid simulation

- Neural ODE has constant memory usage
- Works for black box ODE integrator
- Works with adaptive steps and implicit schemes

Applications

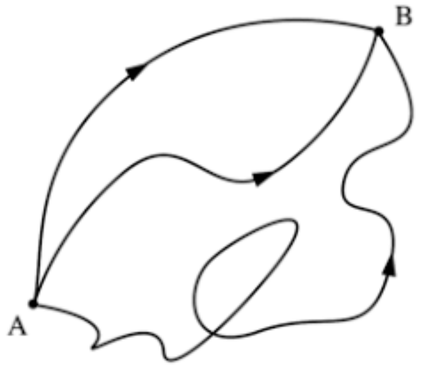
Dynamics systems



$$\frac{dx}{dt} = f(x,t)$$

Classical and quantum control

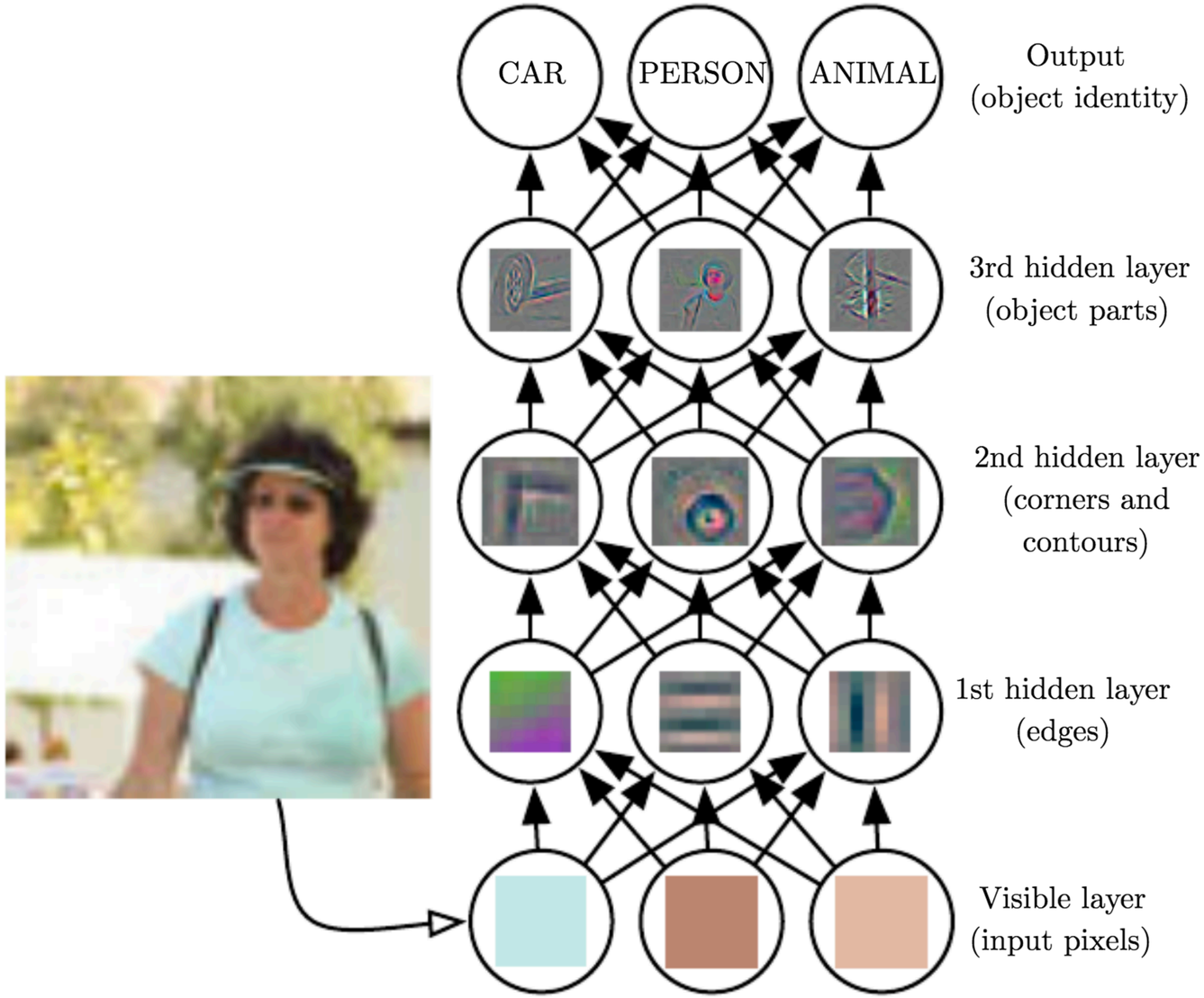
Principle of least actions



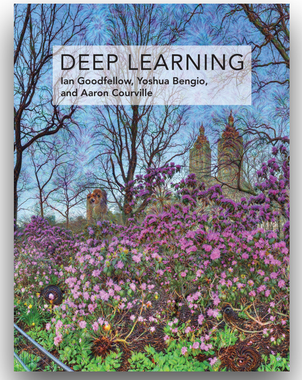
$$S = \int \mathscr{L}(q, \dot{q}, t)dt$$

Optics, (quantum) mechanics, field theory…

Learning Representation



Page 6
Figure 1.2



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

Magic of learning representation

Neural style transfer

Gatys et al, 1508.06576

Latent space interpolation

Glow 1807.03039

https://blog.openai.com/glow/

Magic of learning representation

Neural style transfer

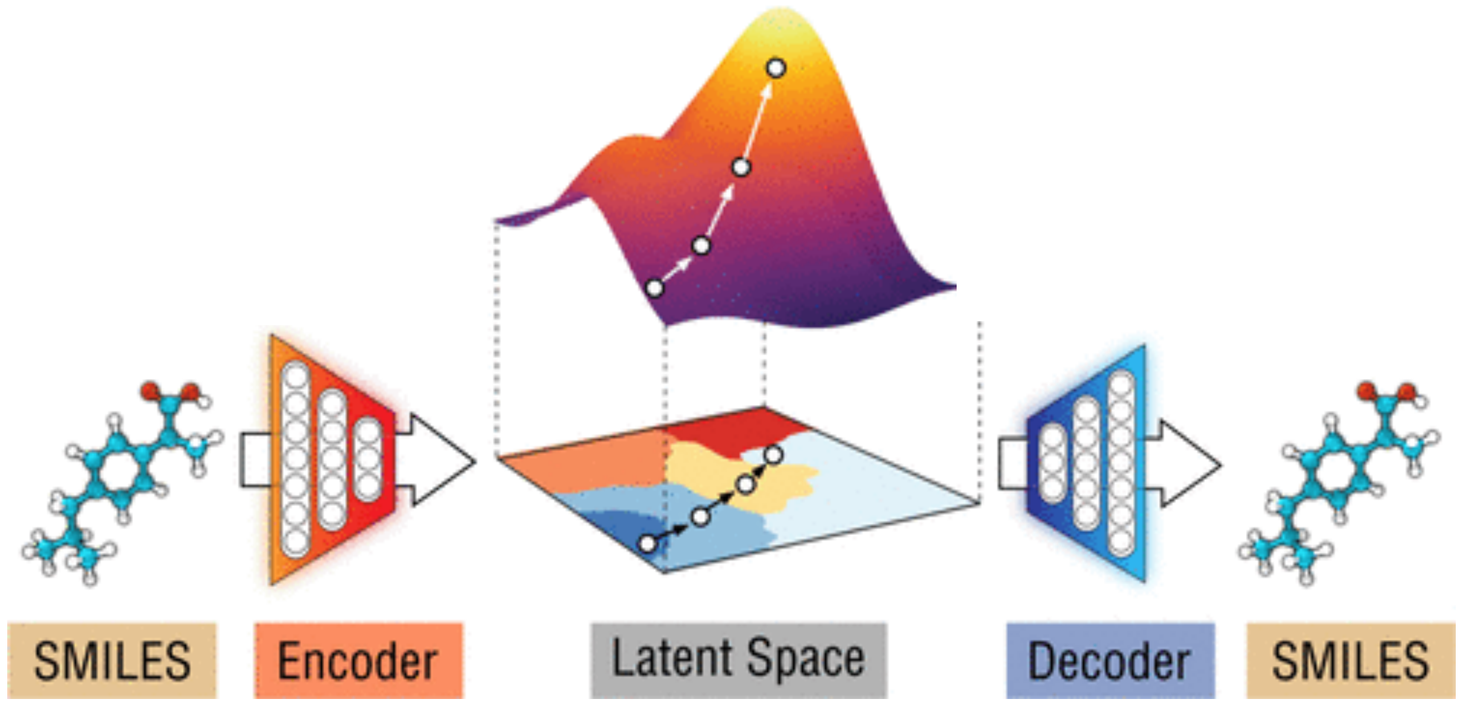
Gatys et al, 1508.06576

Latent space interpolation

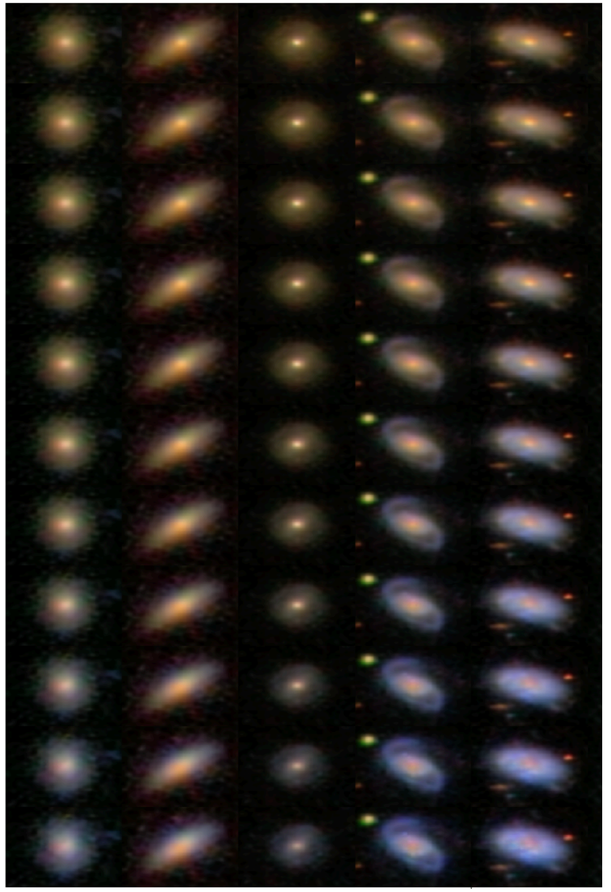
Glow 1807.03039

https://blog.openai.com/glow/

Learning representation for science

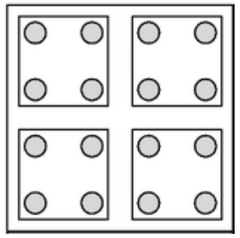


Automatic chemical design,
Gomez-Bombarelli et al,1610.02415



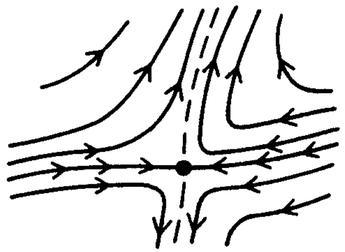
Galaxy evolution
Schawinski et al, 1812.01114

Deep Learning and Renormalization Group

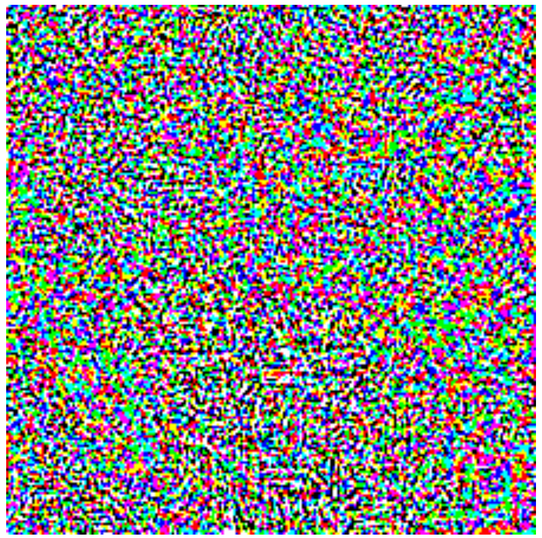


't Hooft, Gross, Wilczek, Kadanoff, Wilson, Fisher…

Bény, Mehta, Schwab, Lin, Tegmark, You, Qi …



$+ .007 \times$



$=$

熊猫

置信度58%

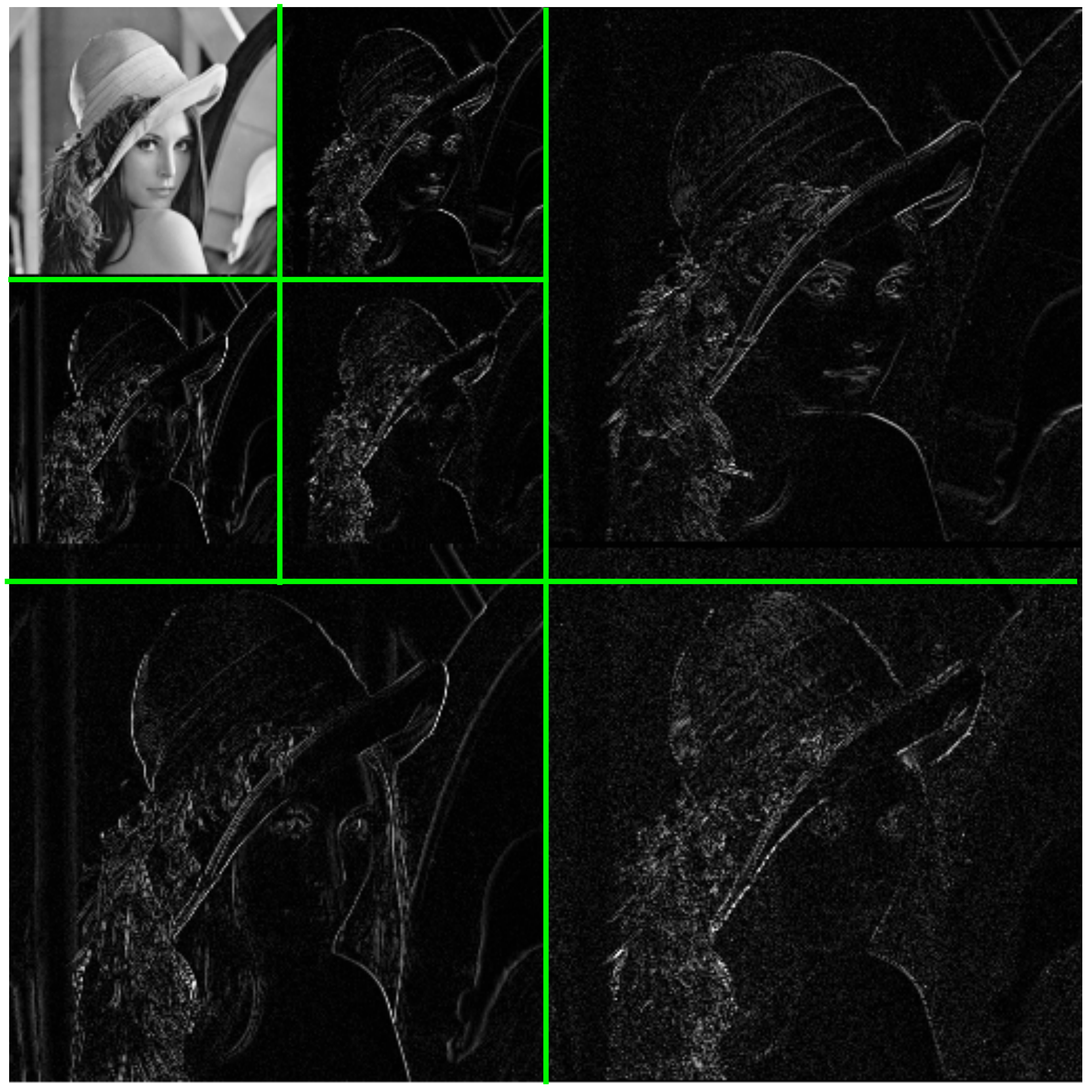
Goodfellow et al, 2014

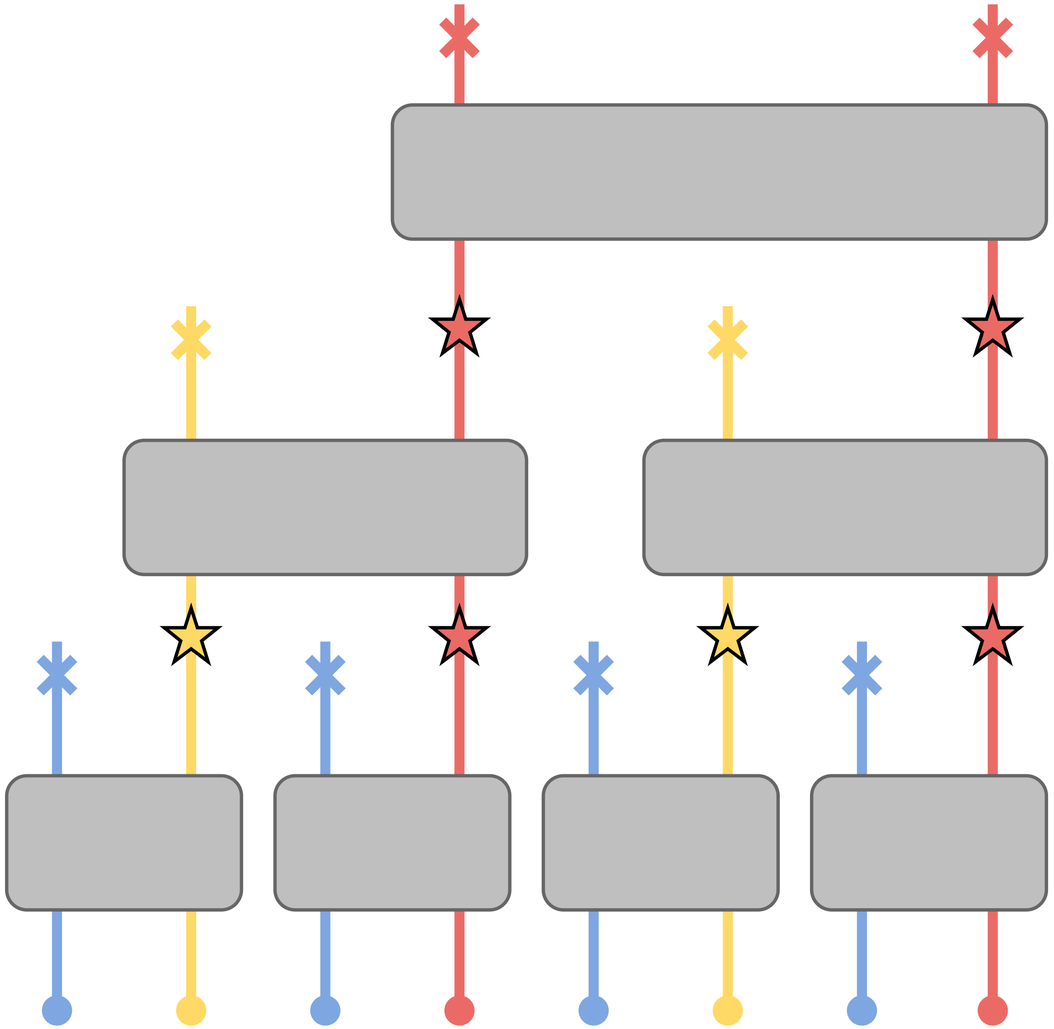
长臂猿

置信度99%

Vulnerability of deep learning, Kenway, 1803.06111 & 1803.10995

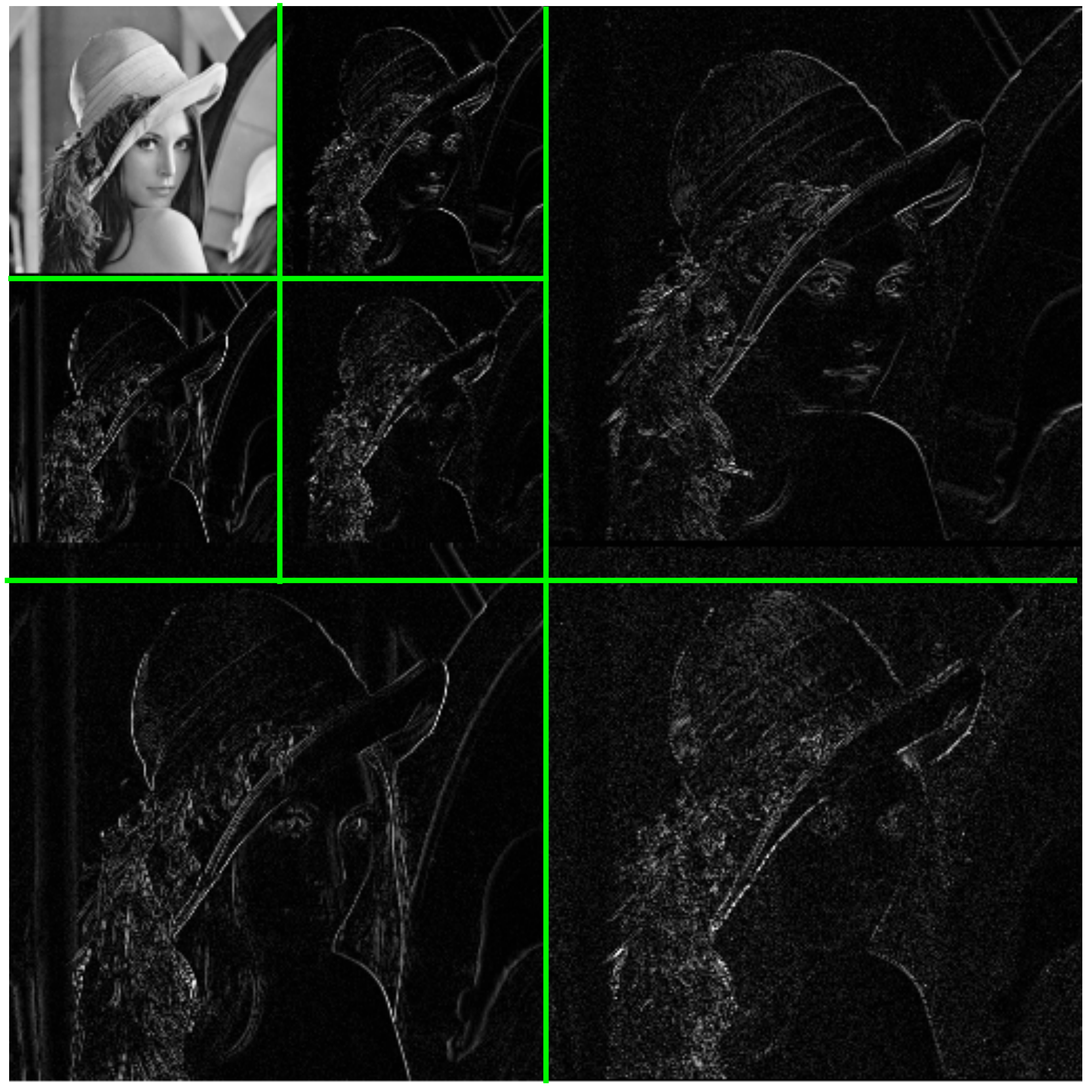
Wavelet transformation for Lena and Ising

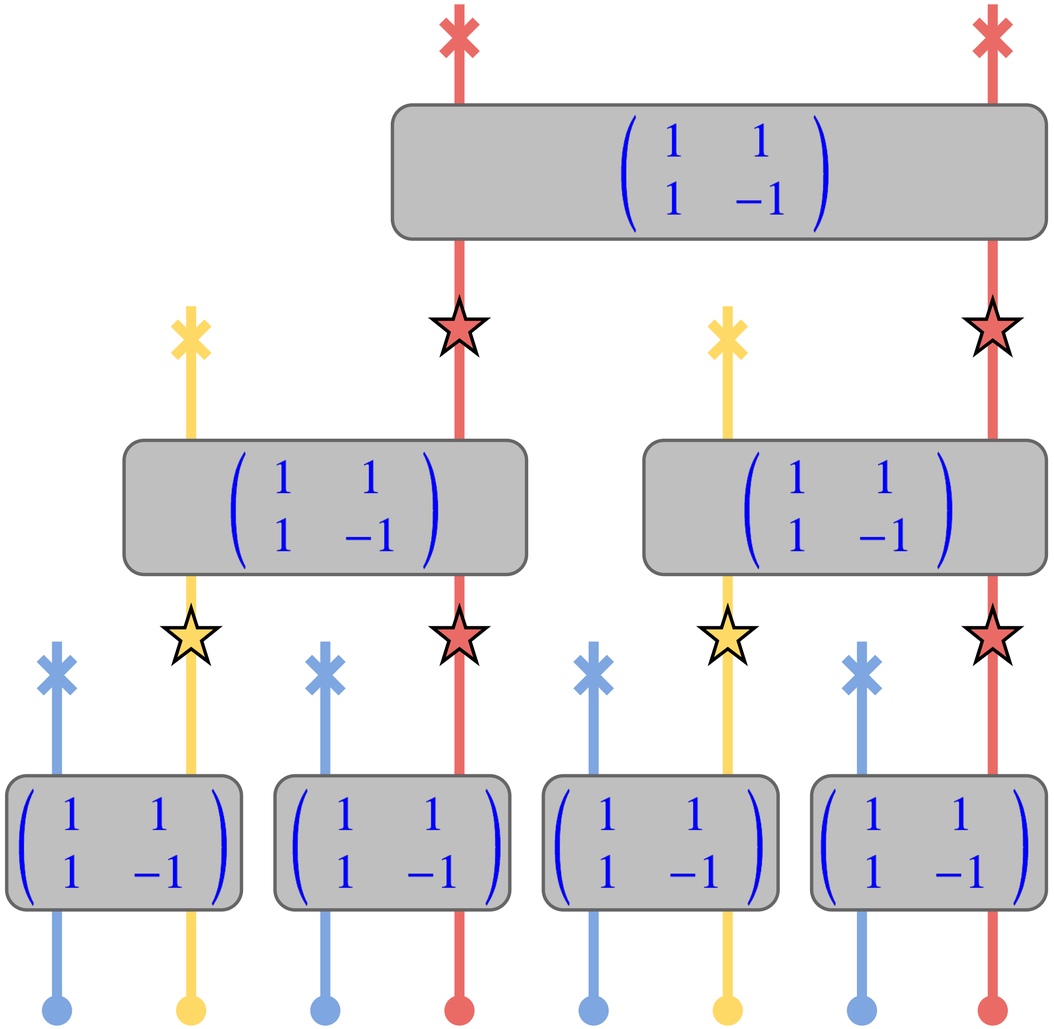




Guy, Wavelets & RG1999+ White, Evenbly, Qi, Wavelets, MERA, and holographic mapping 2013+

Wavelet transformation for Lena and Ising

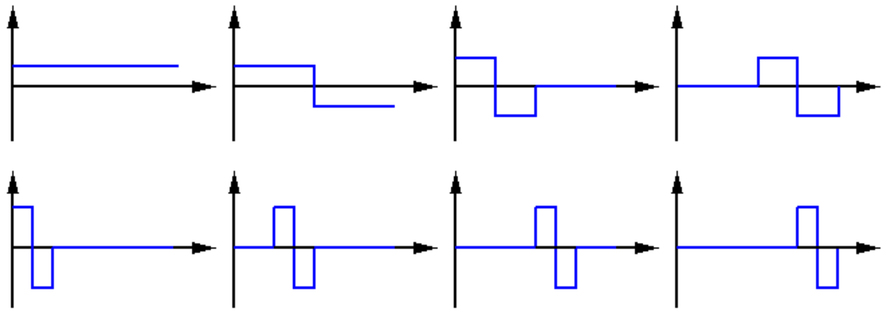


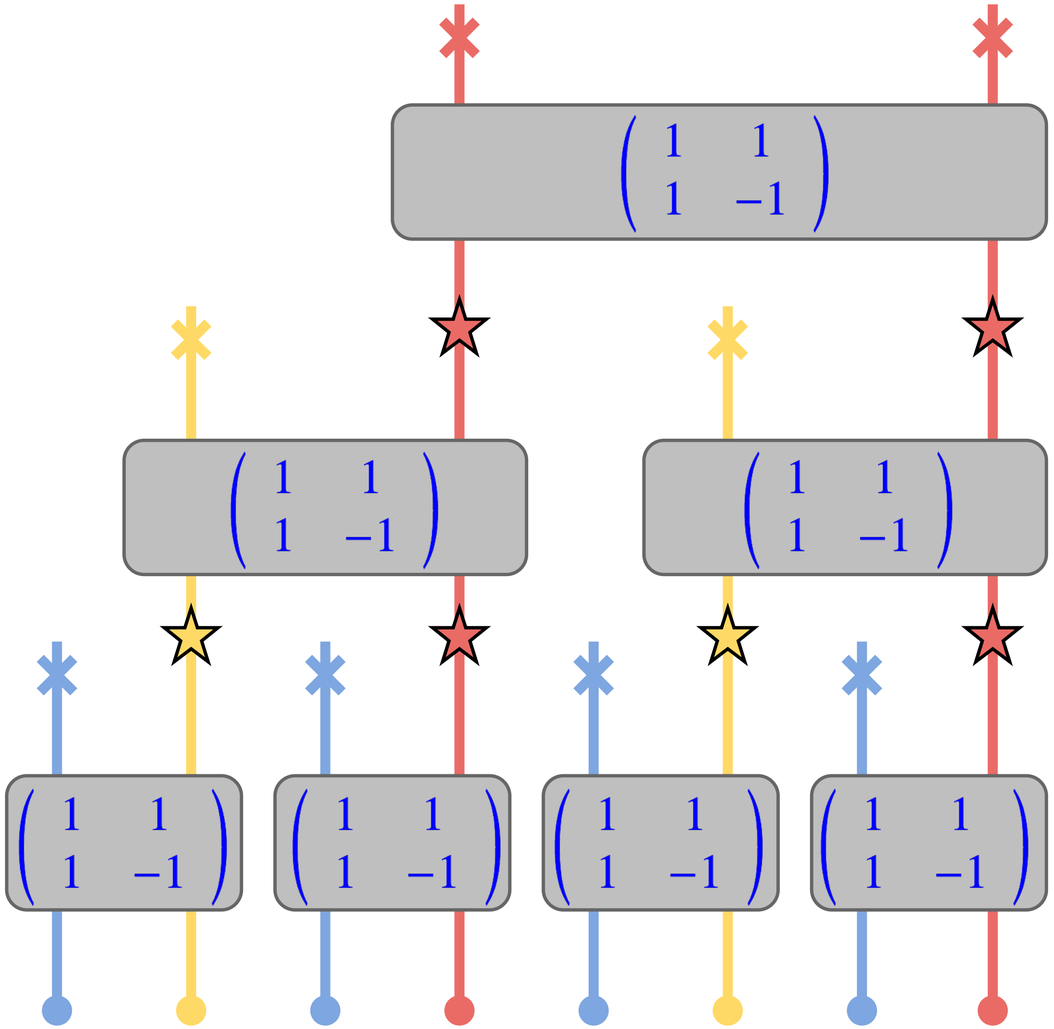


Guy, Wavelets & RG1999+ White, Evenbly, Qi, Wavelets, MERA, and holographic mapping 2013+

Wavelet transformation for Lena and Ising

$$H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$



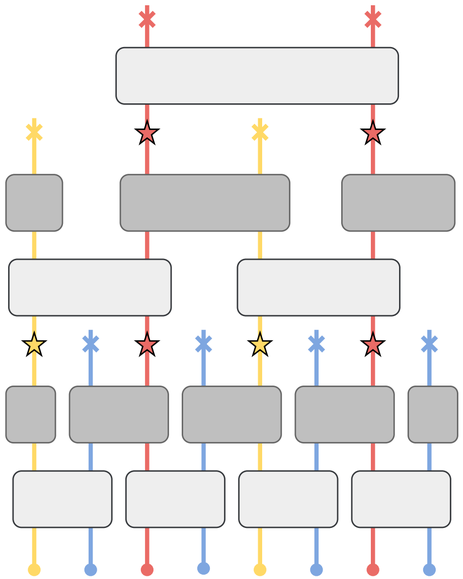


Guy, Wavelets & RG1999+ White, Evenbly, Qi, Wavelets, MERA, and holographic mapping 2013+

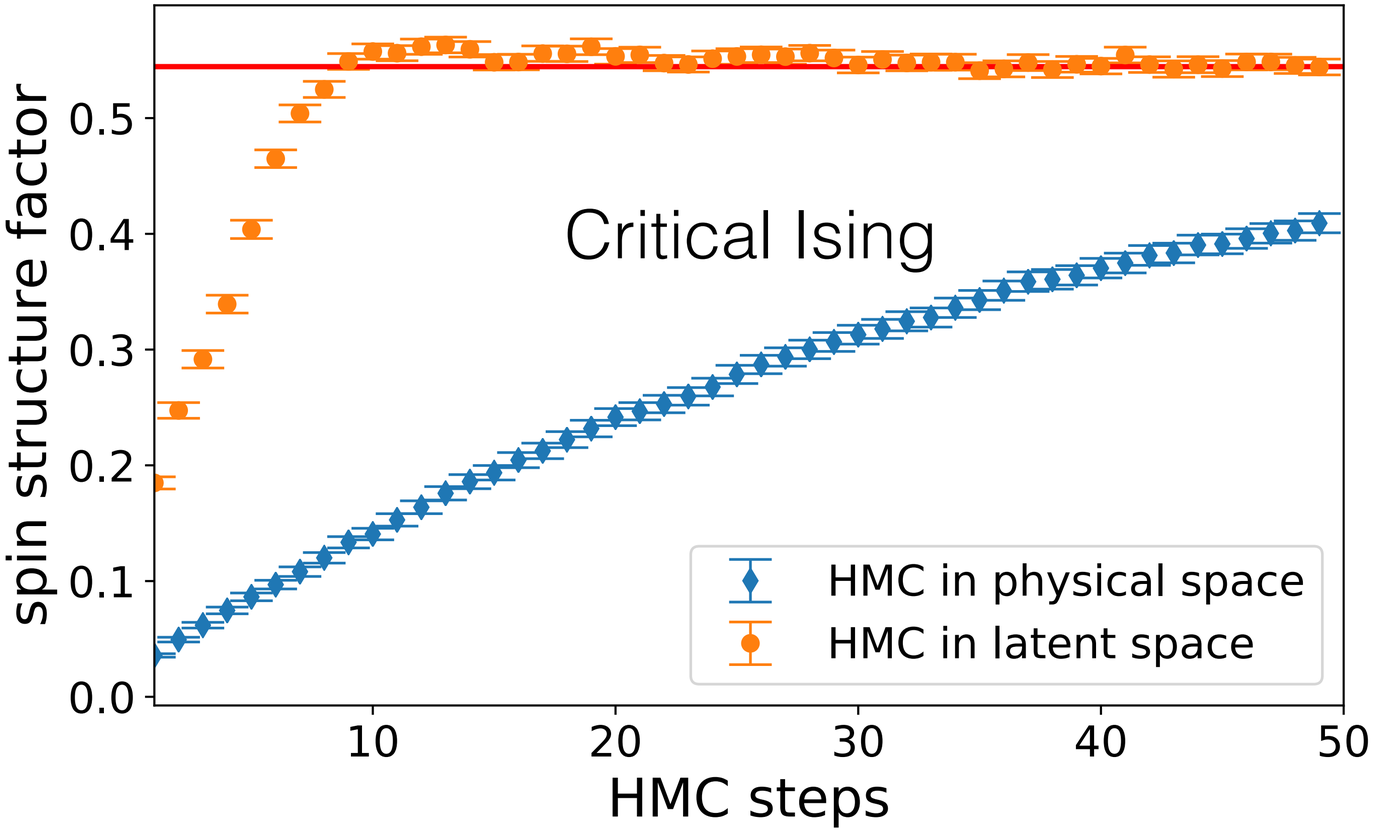
Latent space Hybrid MC

Latent space energy function

$$E_{\text{eff}}(z) = E(g(z)) + \ln q(g(z)) - \ln p(z)$$



Physical energy function $E(x)$



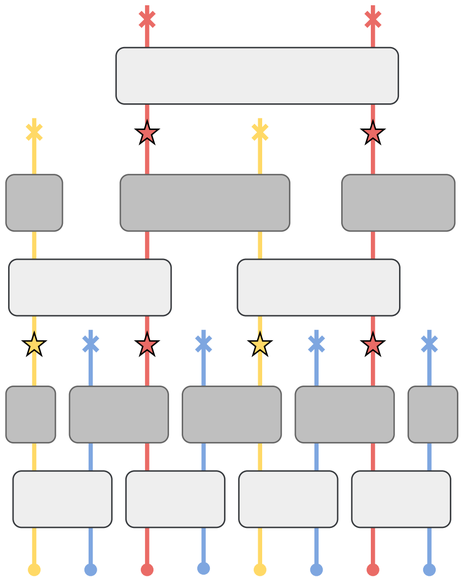
https://github.com/li012589/NeuralRG

HMC thermalizes faster in the latent space

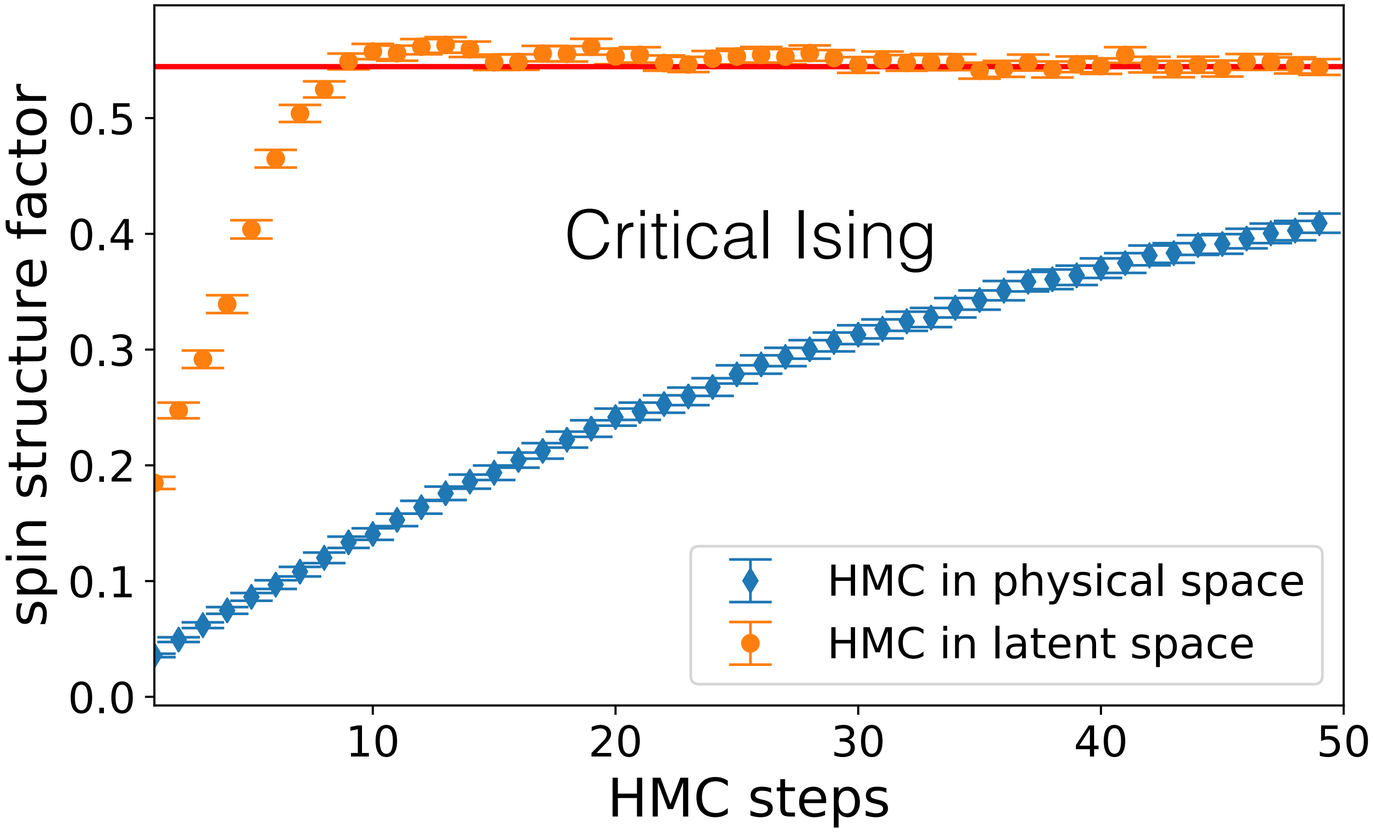
Latent space Hybrid MC

Latent space energy function

$$E_{\text{eff}}(z) = E(g(z)) + \ln q(g(z)) - \ln p(z)$$



Physical energy function $E(x)$



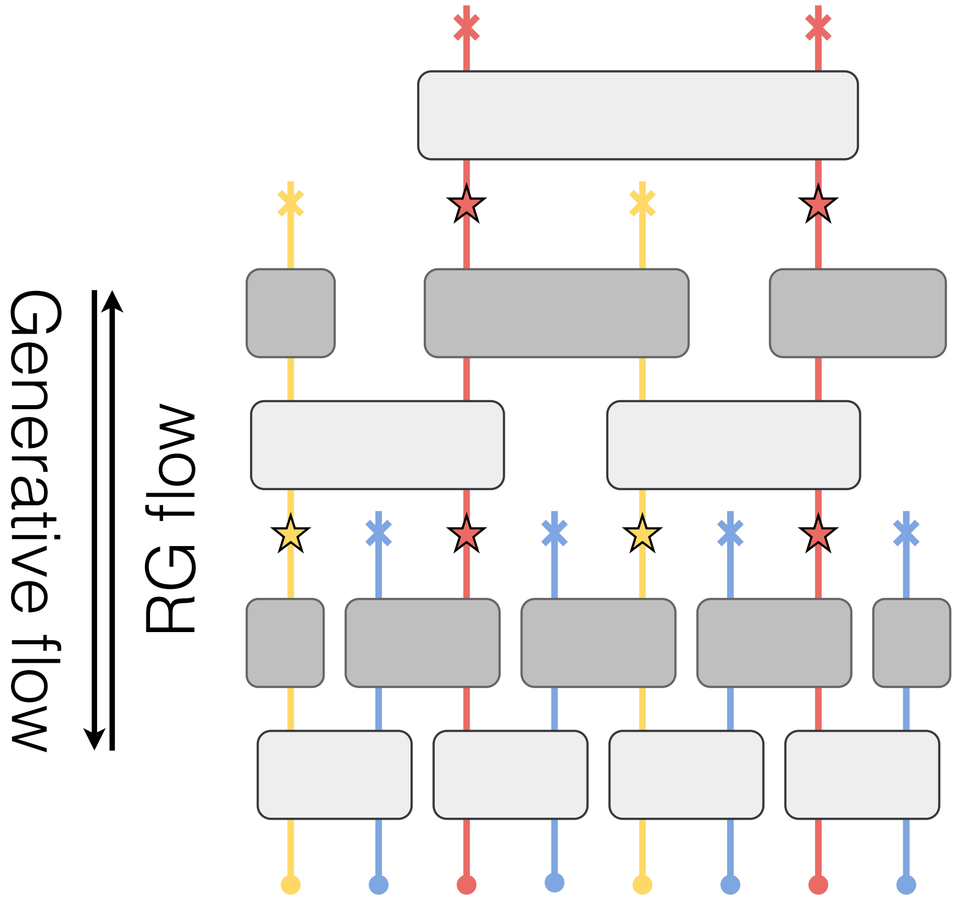
https://github.com/li012589/NeuralRG

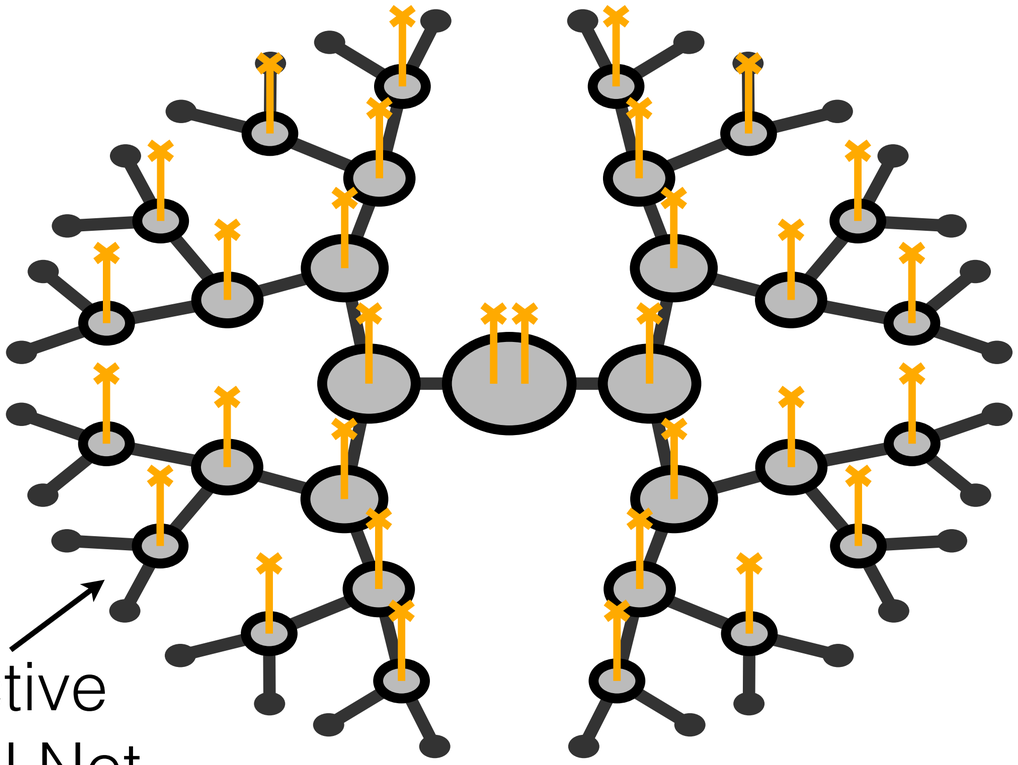
HMC thermalizes faster in the latent space

Neural Renormalization Group Flow

Normalizing flow with multiscale network structures

Swingle 0905.1317, Qi 1309.6282 and more



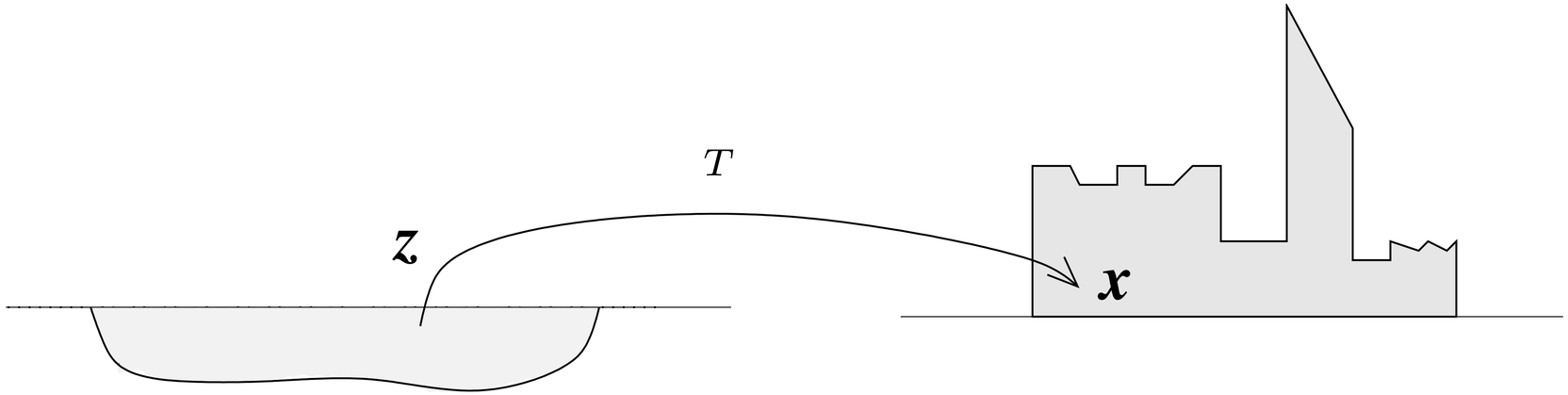


Bijective
Neural Net

Nonlinear & adaptive generalizations of wavelets
And, a fresh approach to holographic duality

Optimal Transport Theory

Monge problem (1781): How to transport earth with optimal cost ?



Monge | Kantorovich | Koopmans | Dantzig | Brenier | Otto | McCann | Villani | Figalli

Nobel Prize in Economics ’75

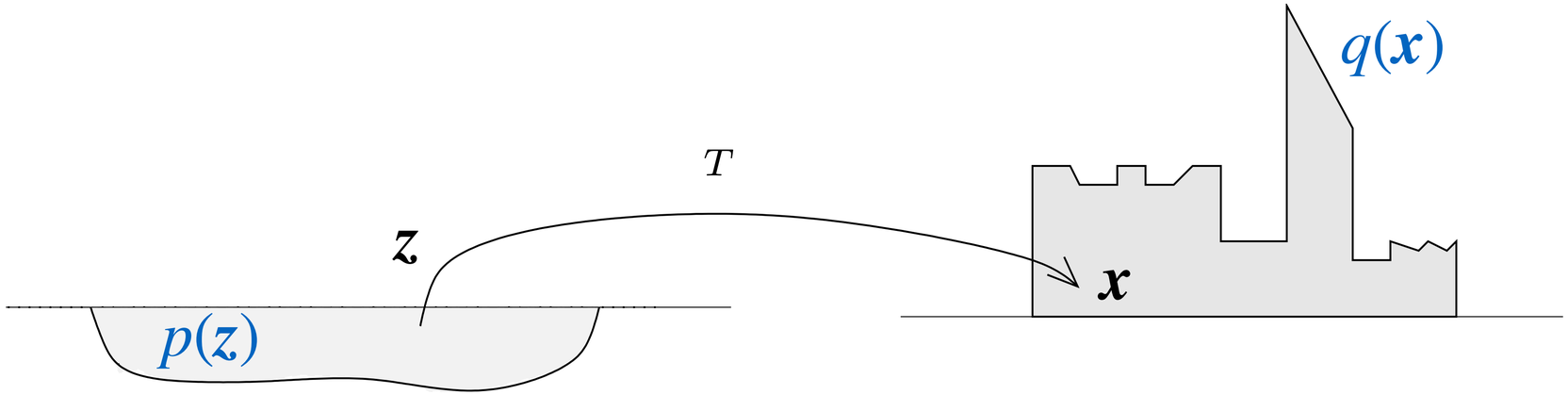
Fields Metal ’10

Fields Metal ’18

from Cuturi, Solomon NISP 2017 tutorial

Optimal Transport Theory

Monge problem (1781): How to transport earth with optimal cost ?



Monge | Kantorovich | Koopmans | Dantzig | Brenier | Otto | McCann | Villani | Figalli

Nobel Prize in Economics ’75

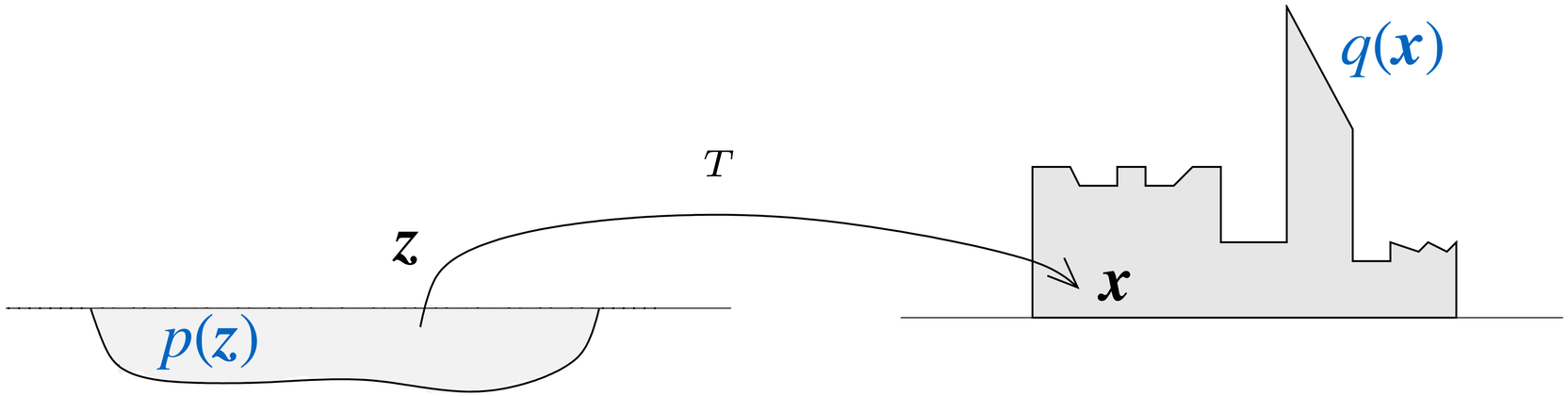
Fields Metal ’10

Fields Metal ’18

from Cuturi, Solomon NISP 2017 tutorial

Optimal Transport Theory

Monge problem (1781): How to transport earth with optimal cost ?



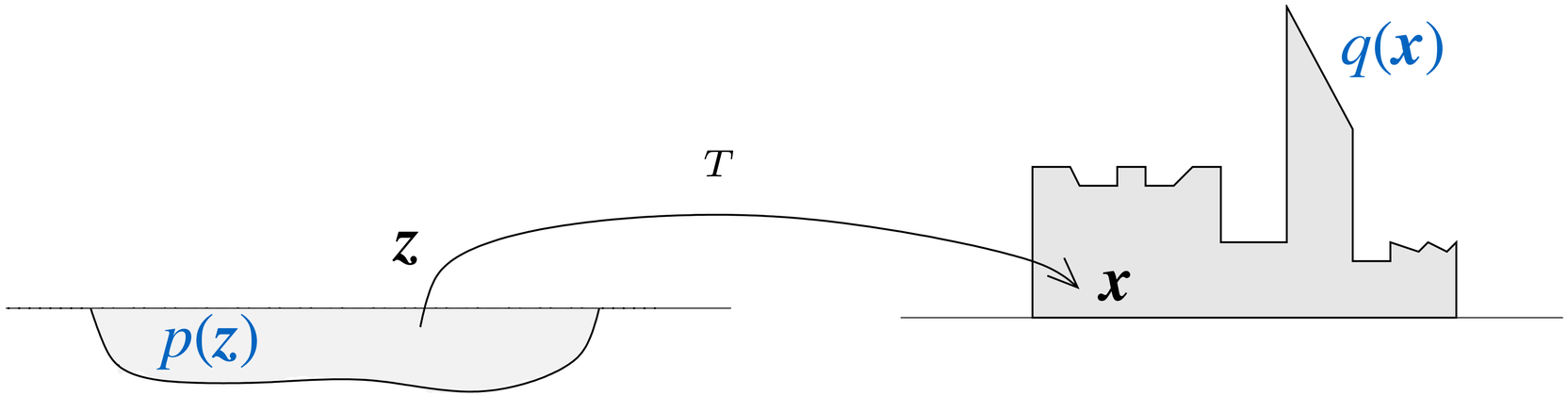
★ Brenier theorem (1991)

Under reasonable conditions
the optimal map is

$$\boldsymbol{z} \mapsto \boldsymbol{x} = \nabla u(\boldsymbol{z})$$

Optimal Transport Theory

Monge problem (1781): How to transport earth with optimal cost ?



★ Brenier theorem (1991)

Under reasonable conditions
the optimal map is

$$z \mapsto x = \nabla u(z)$$

Monge-Ampère Equation

$$\frac{p(z)}{q(\nabla u(z))} = \det\left(\frac{\partial^2 u}{\partial z_i \partial z_j}\right)$$

Shing-Tung Yau

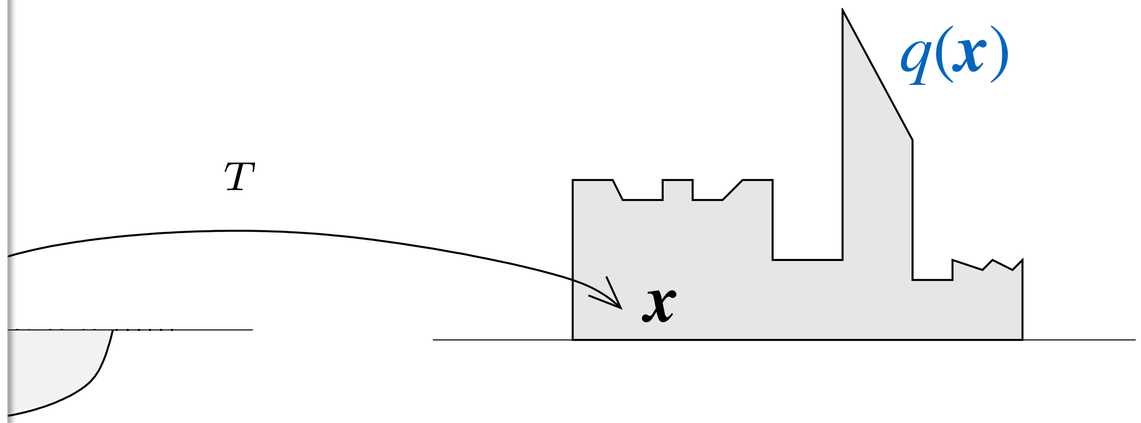
丘成桐

Fields Metal ‘82

Made contributions in differential equations, also to the Calabi conjecture in algebraic geometry, to the positive mass conjecture of general relativity theory, and to real and complex Monge-Ampère equation

Transport Theory

How to transport earth with optimal cost ?



★ Brenier theorem (1991)

Under reasonable conditions
the optimal map is

$$z \mapsto x = \nabla u(z)$$

Monge-Ampère Equation

$$\frac{p(z)}{q(\nabla u(z))} = \det\left(\frac{\partial^2 u}{\partial z_i \partial z_j}\right)$$

Shing-Tung Yau

丘成桐

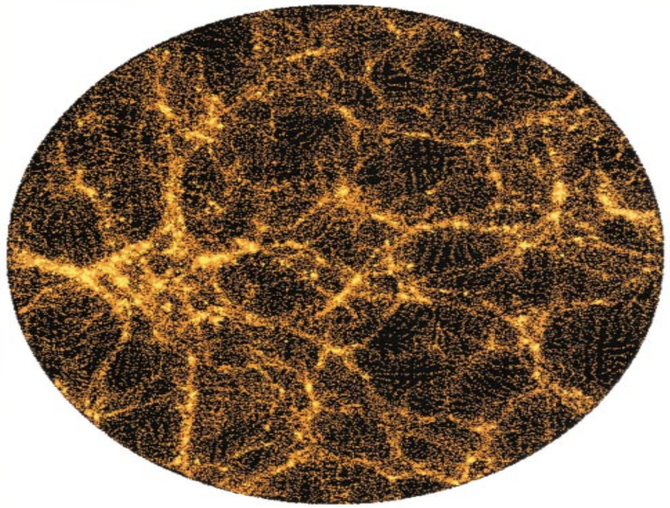
Fields Metal ‘82

Made contributions in differential equations, also to the Calabi conjecture in algebraic geometry, to the positive mass conjecture of general relativity theory, and to real and complex Monge-Ampère equation

letters to nature

A reconstruction of the initial conditions of the Universe by optimal mass transportation

Uriel Frisch*, Sabino Matarrese†, Roya Mohayaee‡* & Andrei Sobolevski§*



viscosity (discovered by Maxwell) does not operate, so that a non-collisional mechanism involving a small-scale gravitational instability must be invoked.

Our reconstruction hypothesis implies that the initial positions can be obtained from the present ones by another gradient map: $\mathbf{q} = \nabla_{\mathbf{x}}\Theta(\mathbf{x})$, where Θ is a convex potential related to Φ by a Legendre–Fenchel transform (see Methods). We denote by ρ_0 the initial mass density (which can be treated as uniform) and by $\rho(\mathbf{x})$ the final one. Mass conservation implies $\rho_0 \, \mathrm{d}^3q = \rho(\mathbf{x}) \, \mathrm{d}^3x$. Thus, the ratio of final to initial density is the jacobian of the inverse lagrangian map. This can be written as the following Monge–Ampère equation[20] for the unknown potential Θ:

$$\det(\nabla_{x_i}\nabla_{x_j}\Theta(\mathbf{x})) = \rho(\mathbf{x})/\rho_0 \qquad (1)$$

where ‘det’ stands for determinant.

We emphasize that no information about the dynamics of matter other than the reconstruction hypothesis is needed for our method, whose degree of success depends crucially on how well this hypothesis is satisfied. Exact reconstruction is obtained, for example, for the Zel’dovich approximation (before particle trajectories cross) and for adhesion-model dynamics (at arbitrary times).

We note that our Monge–Ampère equation for self-gravitating matter may be viewed as a nonlinear generalization of a Poisson equation (used for reconstruction in ref. 4), to which it reduces if particles have moved very little from their initial positions.

It has been discovered recently that the map generated by the solution to the Monge–Ampère equation (1) is the (unique) solution to an optimization problem[21] (see also refs 22 and 23).

★ Brenier theorem (1991)

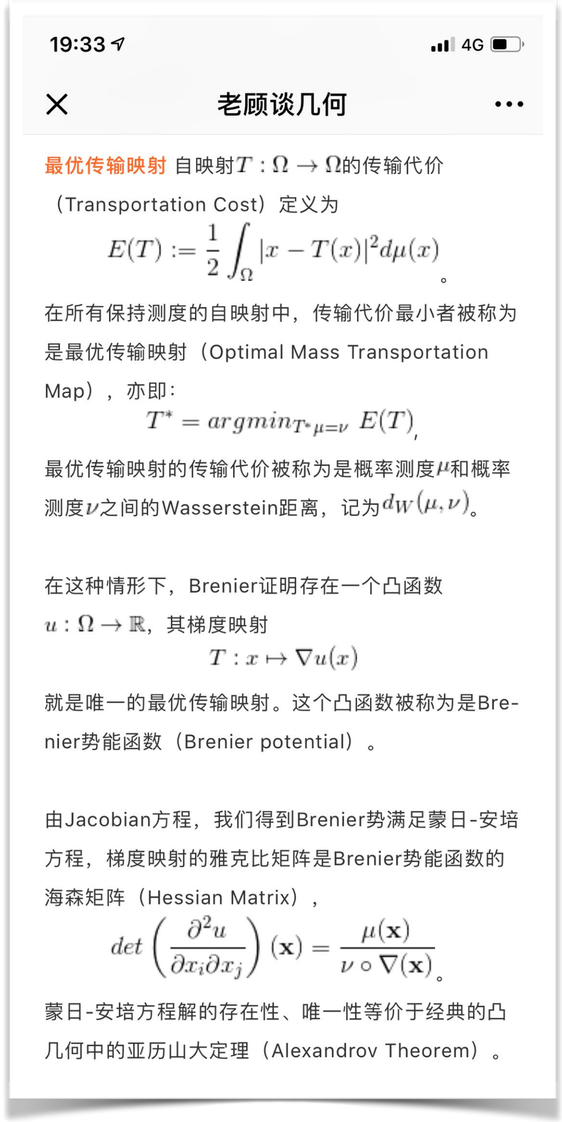
Under reasonable conditions the optimal map is

$$z \mapsto x = \nabla u(z)$$

Monge-Ampère Equation

$$\frac{p(z)}{q(\nabla u(z))} = \det\left(\frac{\partial^2 u}{\partial z_i \partial z_j}\right)$$

A geometric perspective to deep learning



Lei, Su, Cui, Yau, Gu 1710.05488

DL as a fluid control problem

$$\frac{p(z)}{q(\nabla u(z))} = \det\left(\frac{\partial^2 u}{\partial z_i \partial z_j}\right)$$

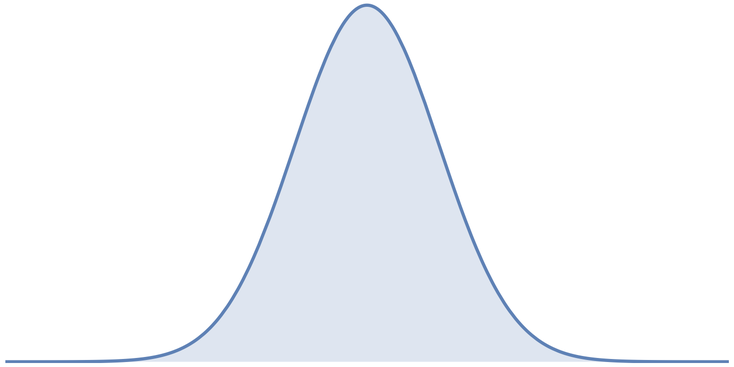
Monge-Ampère equation
optimal transport theory

Continuous-time limit

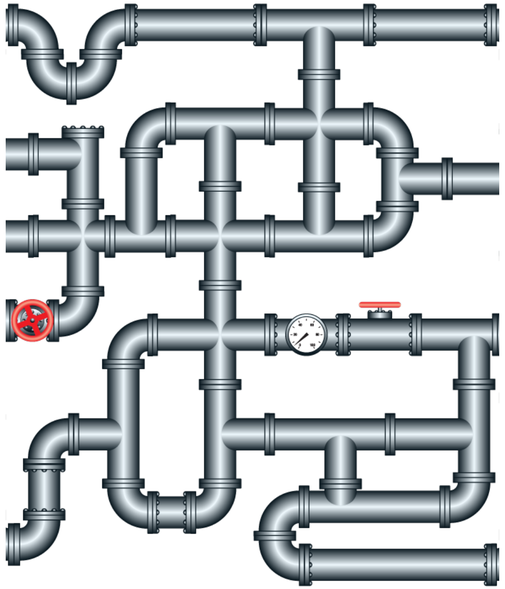
$$u(z) = |z|^2/2 + \epsilon\varphi(z)$$

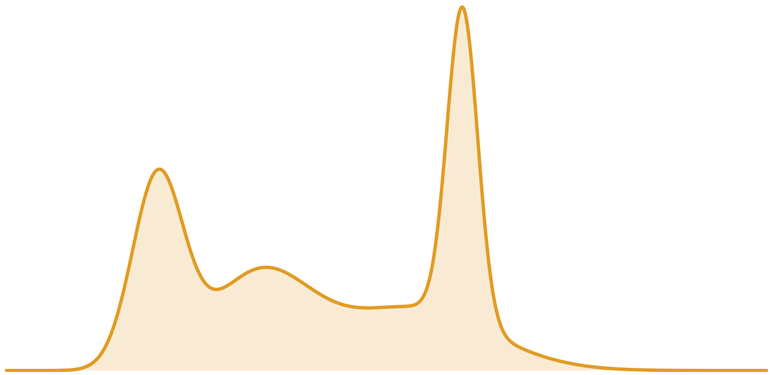
$$\frac{\partial p(\boldsymbol{x}, t)}{\partial t} + \nabla \cdot \left[p(\boldsymbol{x}, t)\, \nabla \varphi\right] = 0$$

Continuity equation of
compressible fluids



Simple density





Complex density

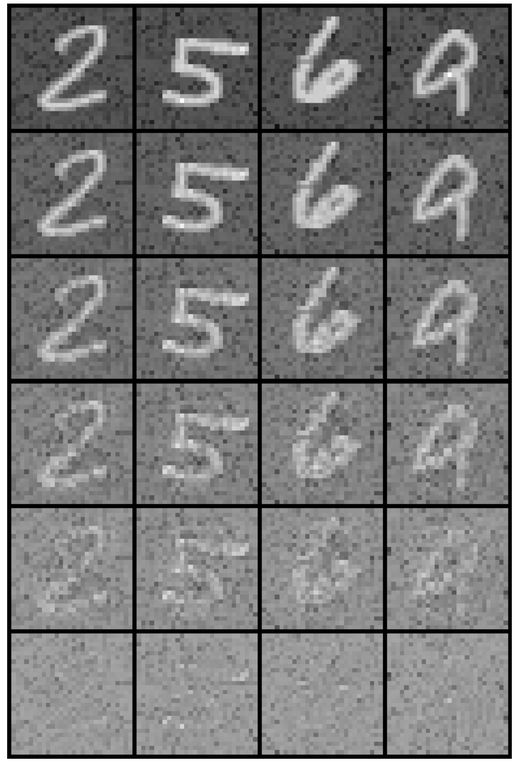
Zhang, E, LW, 1809.10188
c.f. Neural ODE, 1806.07366

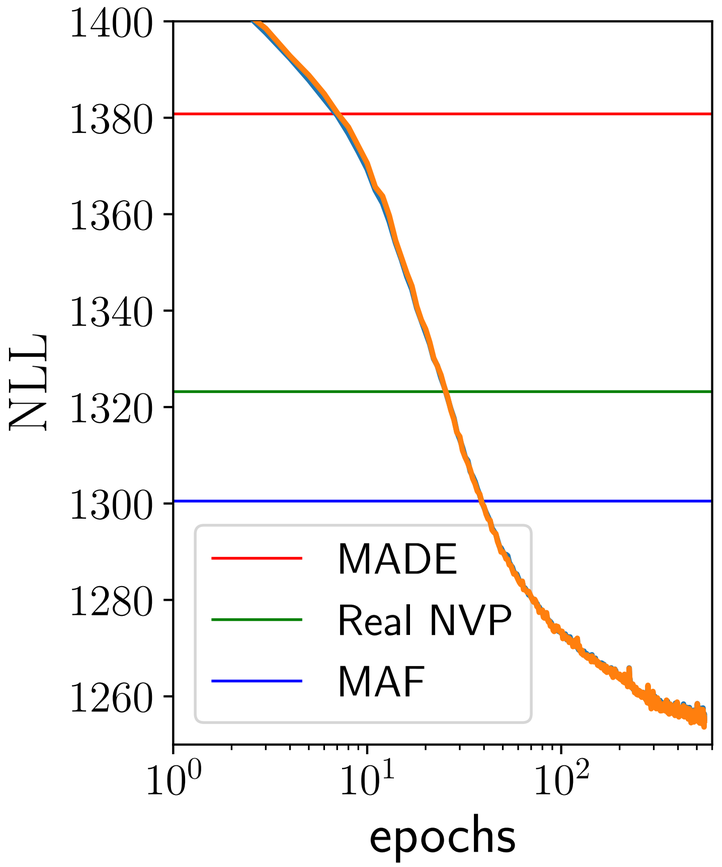
Density estimation of hand-written digits

A standard benchmark for generative models, lower is better

Data space

Latent space





’15

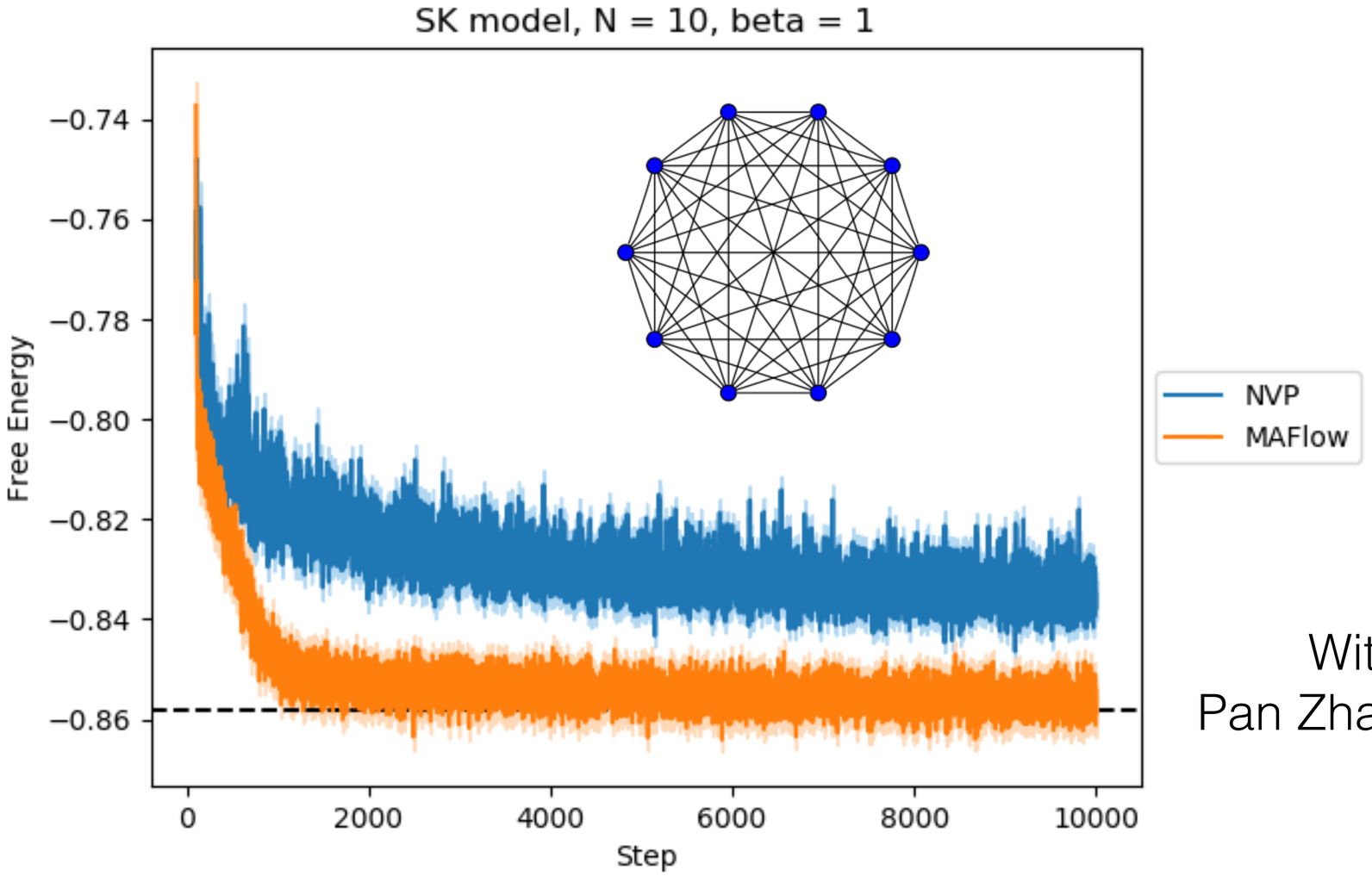
‘16

‘17

’18 Our Result

1 **State-of-the-art performance in unstructured density estimation**

Variational study of Sherrington-Kirkpatrick spin glasses



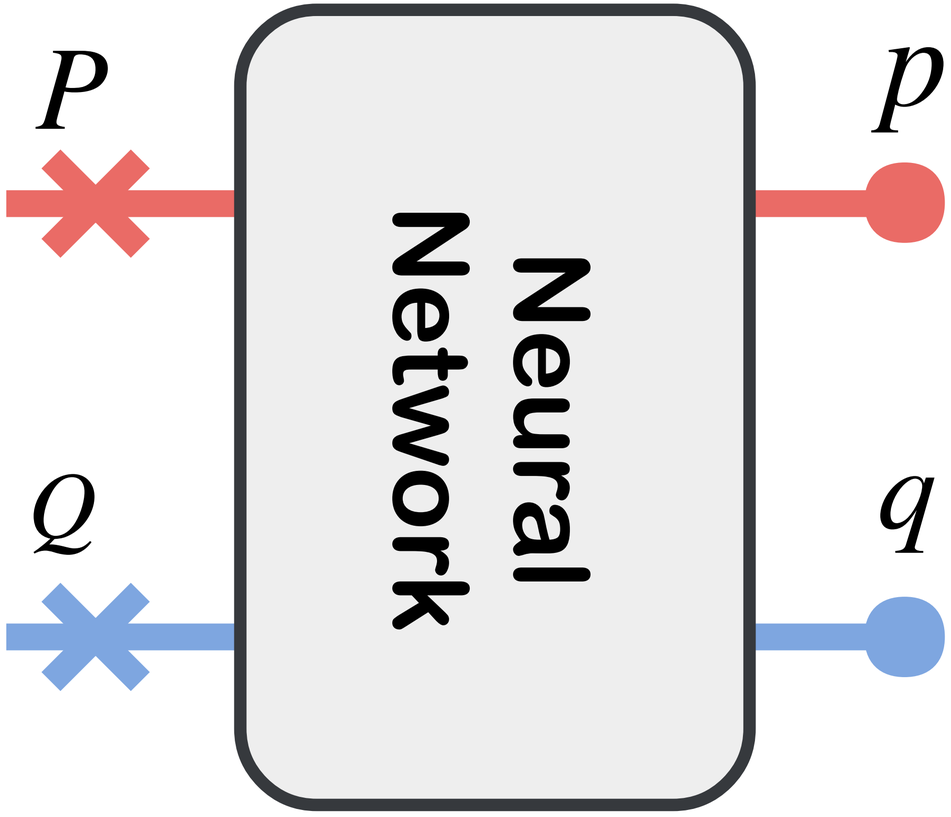
With Dian Wu,
Pan Zhang unpublished

② **Better variational energy than previous network structure**

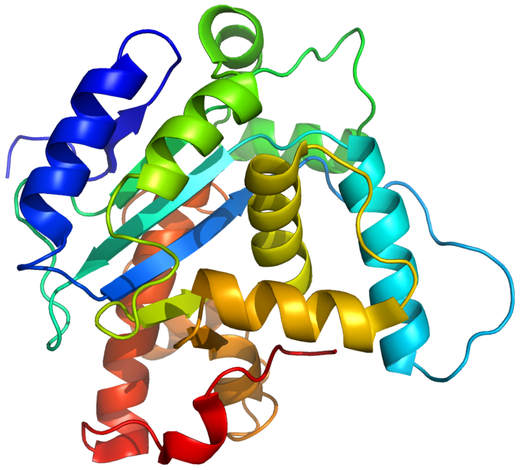
Neural Canonical Transformations

Incompressible symplectic flow in phase space

$H(P, Q) = (P^2 + Q^2)/2$



$H(p, q)$



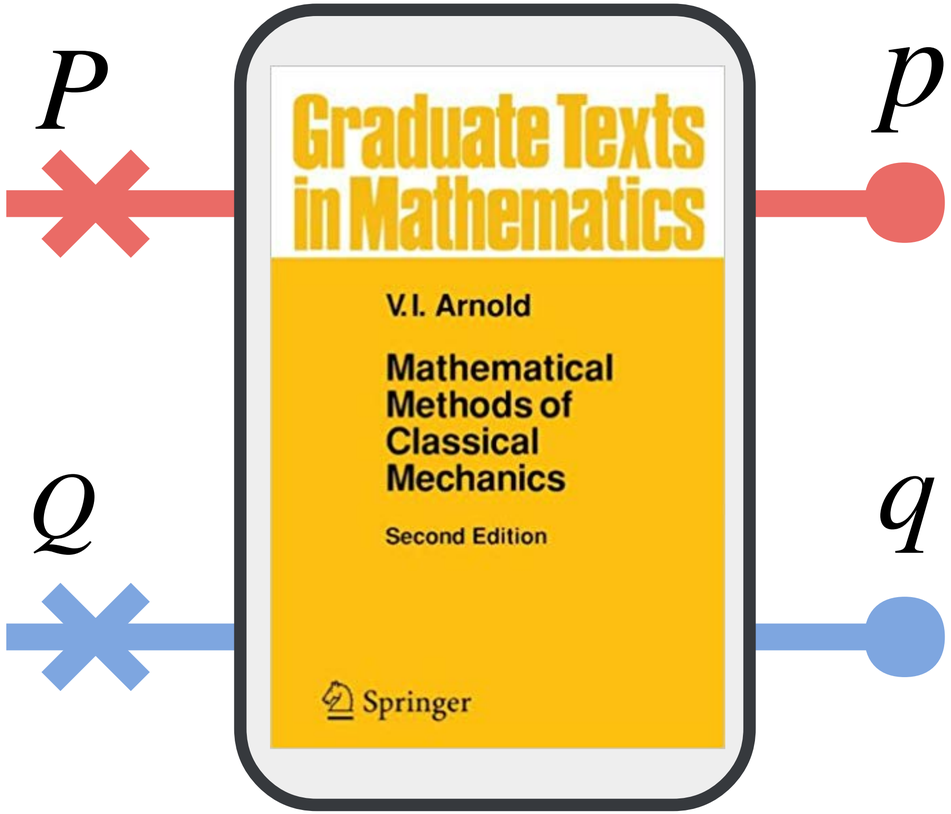
3

Identifying mutually independent collective modes for molecular simulations (MD, PIMD), and effective field theory

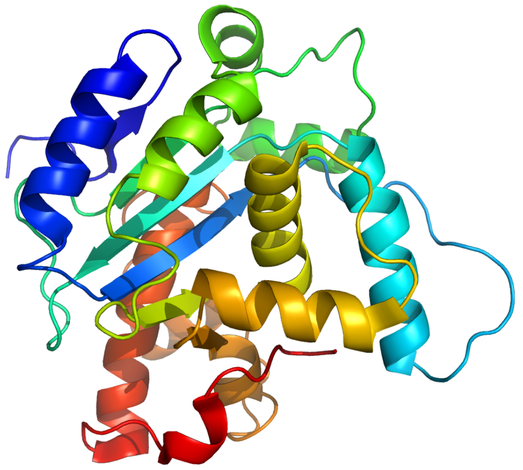
Neural Canonical Transformations

Incompressible symplectic flow in phase space

$$H(P,Q) = (P^2 + Q^2)/2$$

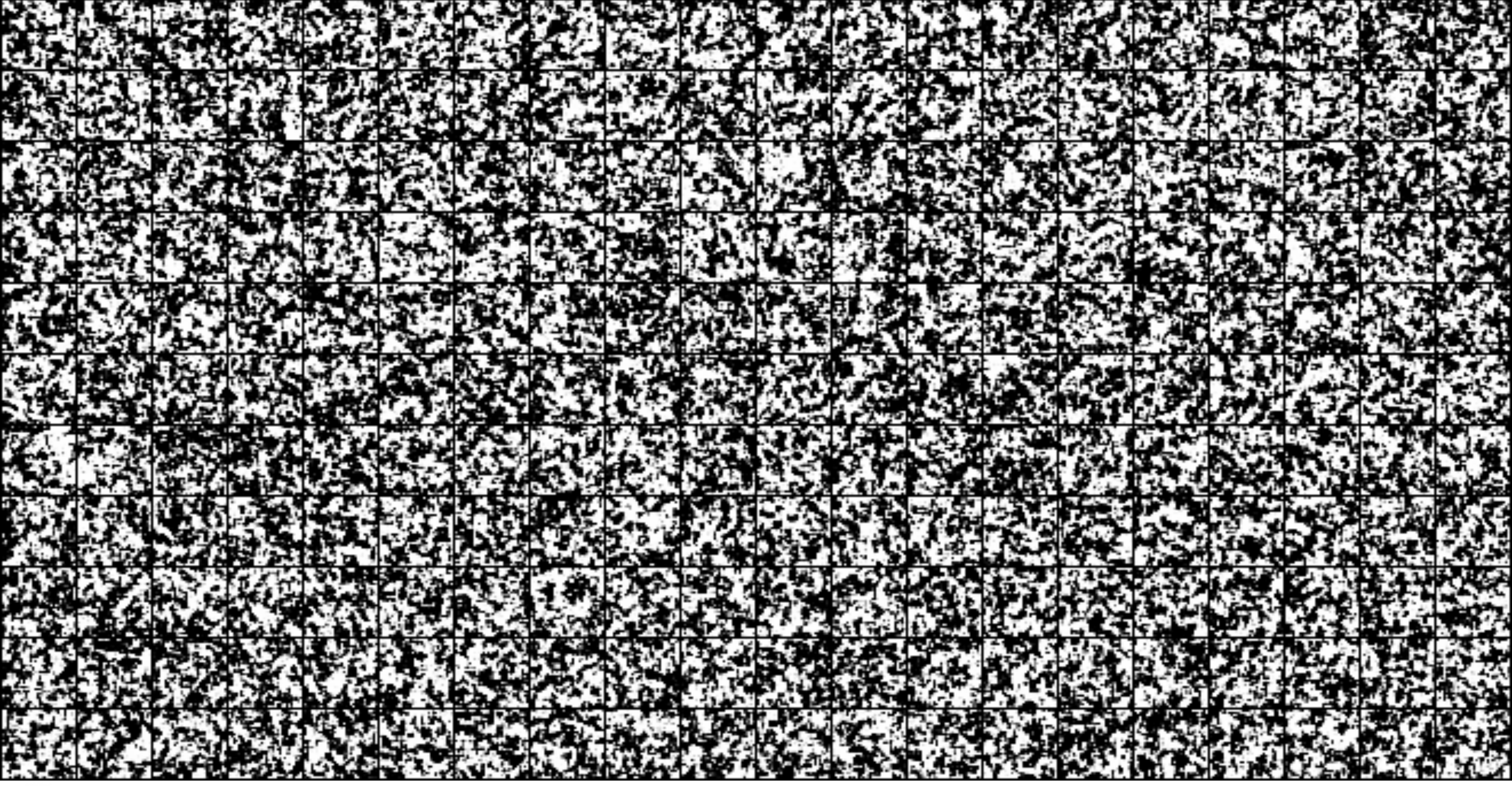


$$H(p,q)$$

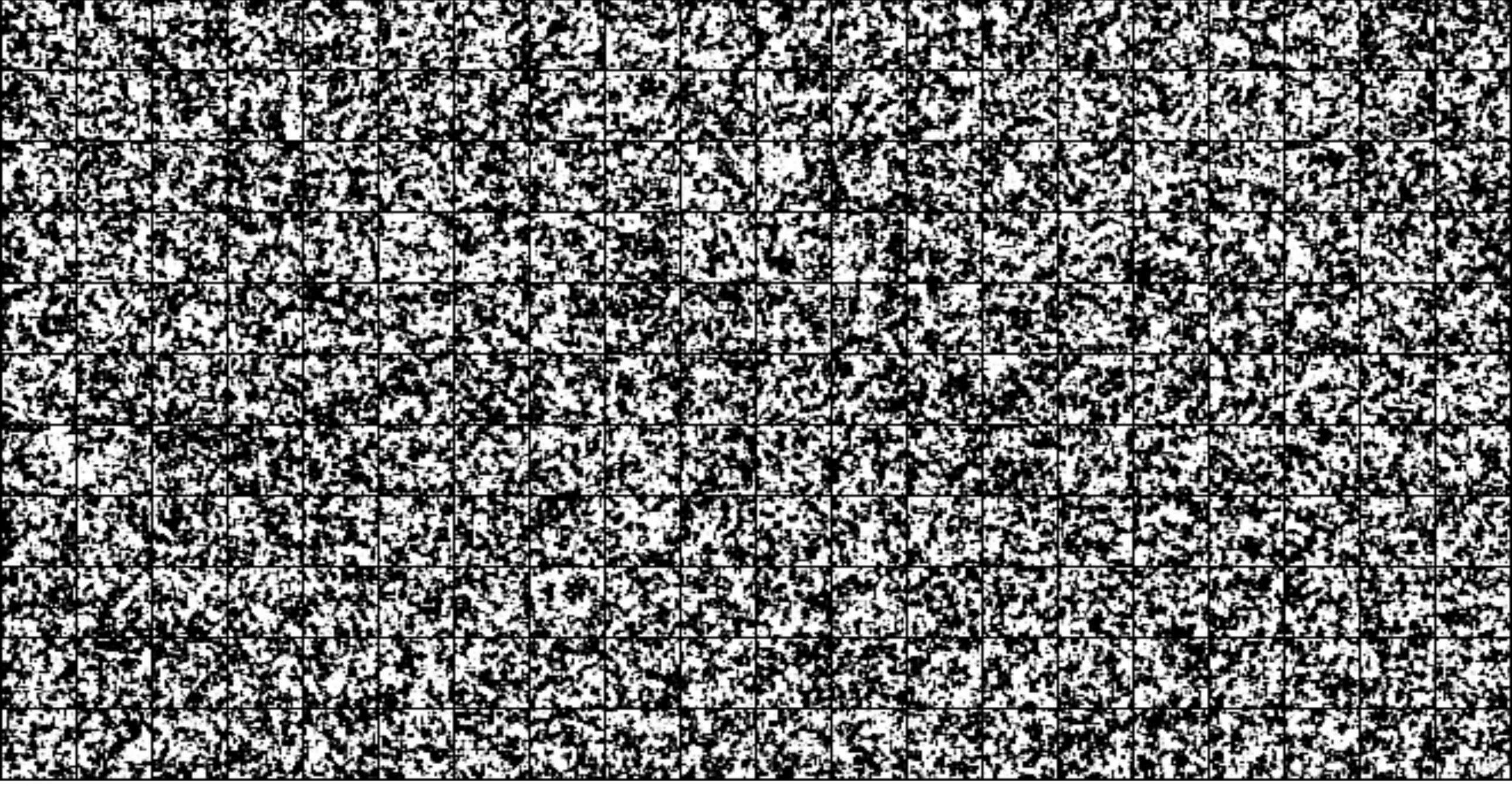


3

Identifying mutually independent collective modes for molecular simulations (MD, PIMD), and effective field theory

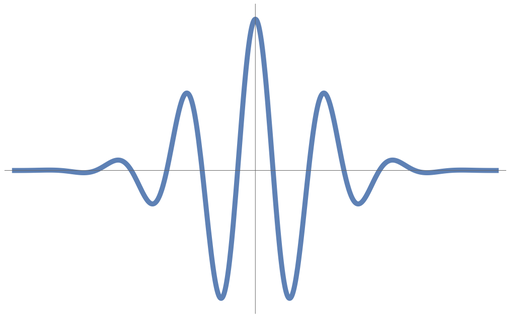


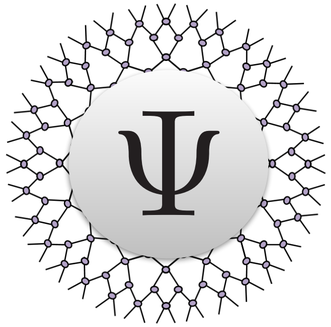
4 **Direct sample magnetic domains respecting physical symmetry**



4 **Direct sample magnetic domains respecting physical symmetry**

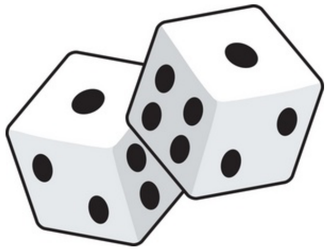
Wavelets

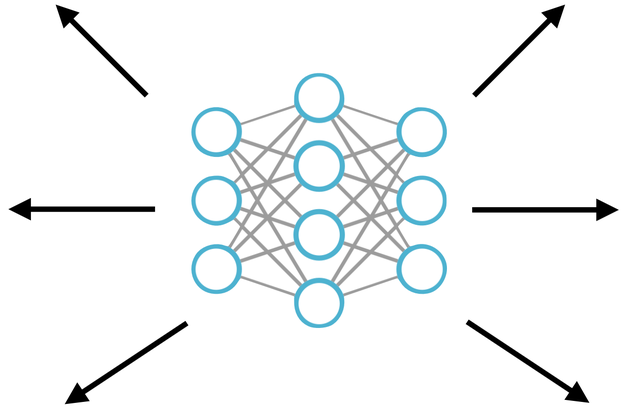


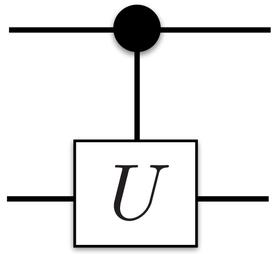


Tensor Networks

Monte Carlo

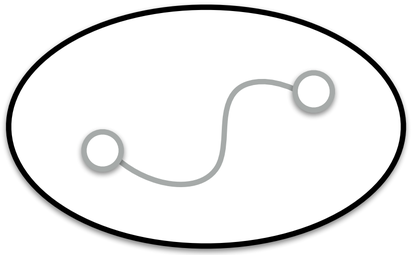


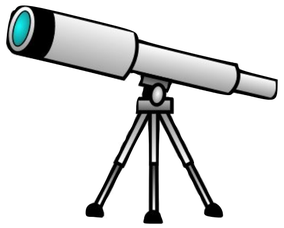




Quantum Circuits

Variational Inference





Holographic RG

Thank You!