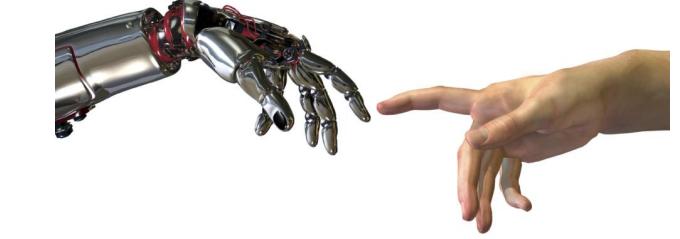
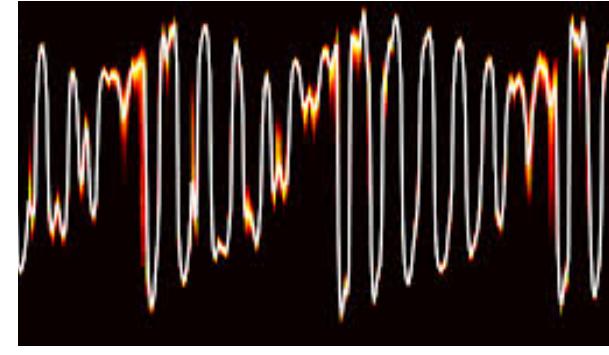
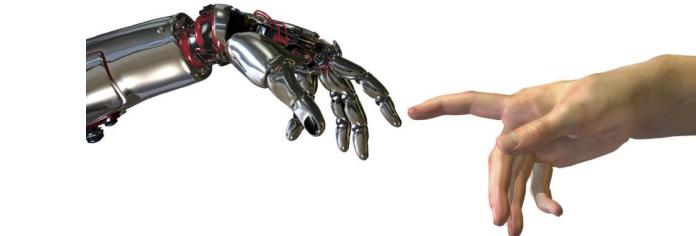
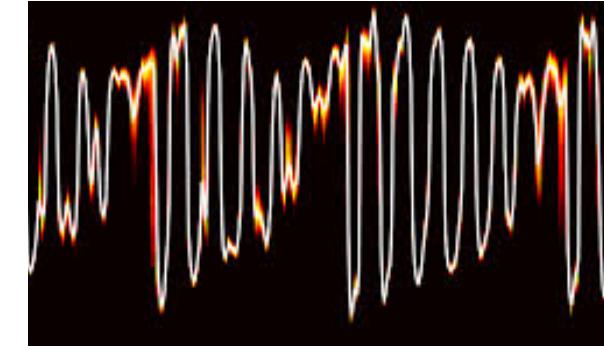


What is the secret behind deep learning?



What is the secret behind deep learning?

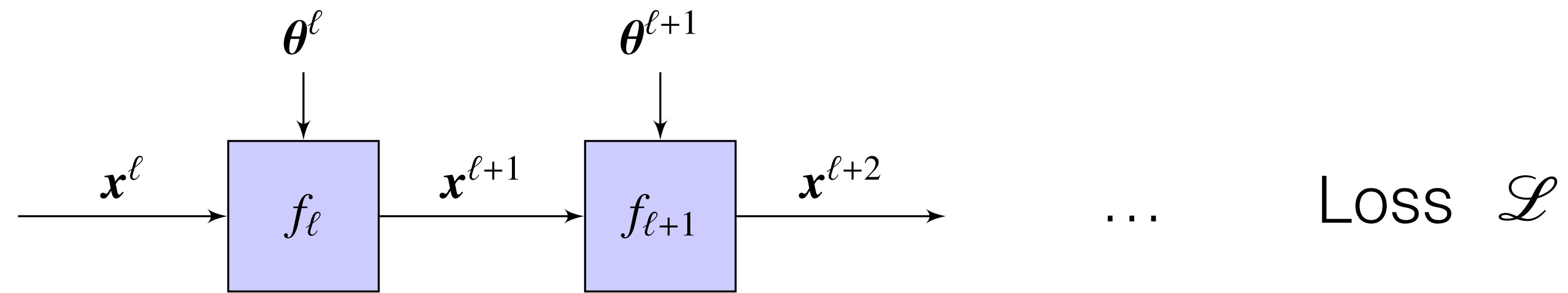


Key technology: differentiable programming

Core idea: learning representation

The engine of deep learning: Back-Propagation algorithm

computation graph



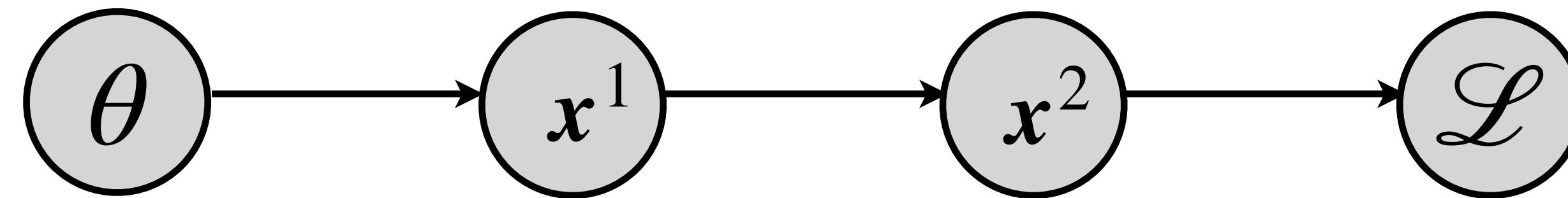
$$\frac{\partial \mathcal{L}}{\partial x^\ell} = \frac{\partial \mathcal{L}}{\partial x^{\ell+1}} \left(\frac{\partial x^{\ell+1}}{\partial x^\ell} \right)$$
$$\frac{\partial \mathcal{L}}{\partial \theta^\ell} = \frac{\partial \mathcal{L}}{\partial x^{\ell+1}} \left(\frac{\partial x^{\ell+1}}{\partial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

**Computes gradients efficiently & accurately
via reverse mode automatic differentiation**

Computation Graph

Forward: function evaluation

Chain graph



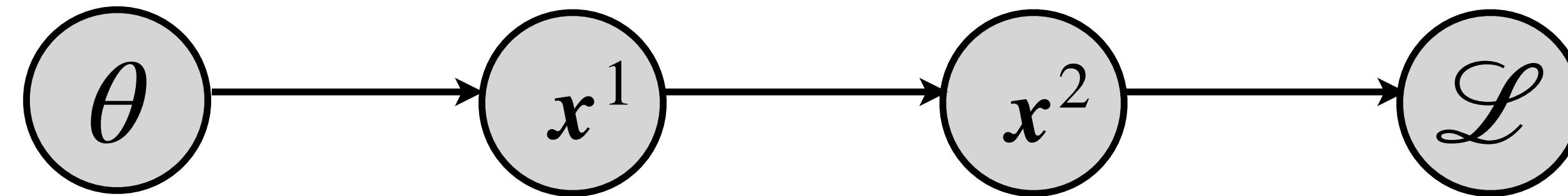
For backward, define “adjoint” $\bar{x} = \frac{\partial \mathcal{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Forward: function evaluation

Chain graph



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

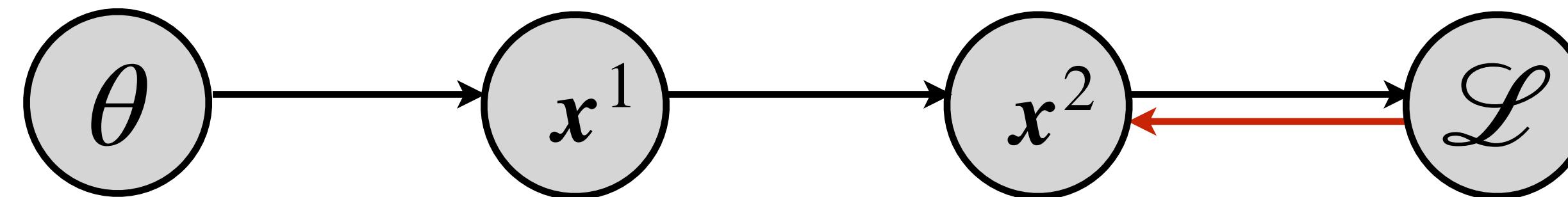
For backward, define “adjoint” $\bar{x} = \frac{\partial \mathcal{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Forward: function evaluation

Chain graph



$$\bar{x}^2 = \bar{\mathcal{L}} \frac{\partial \mathcal{L}}{\partial x^2} \quad \bar{\mathcal{L}} = 1$$

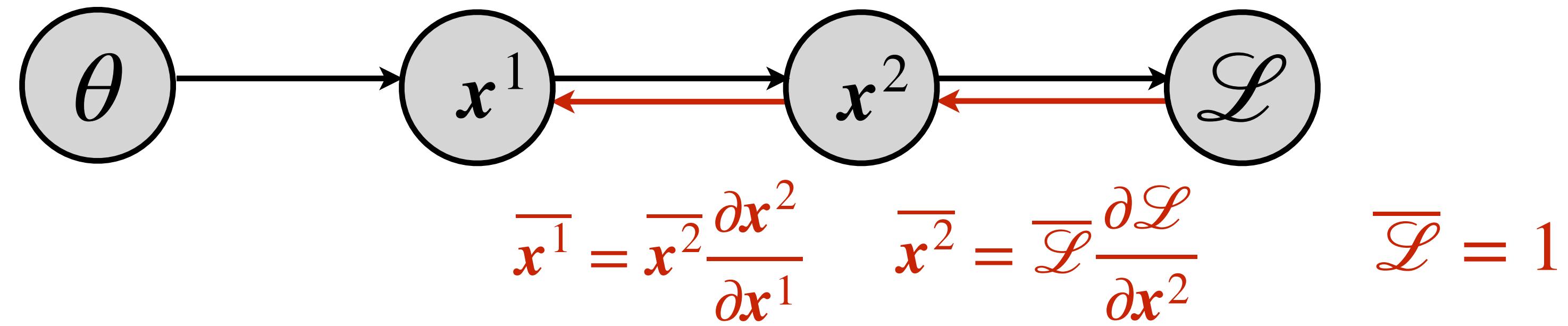
For backward, define “adjoint” $\bar{x} = \frac{\partial \mathcal{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Forward: function evaluation

Chain graph



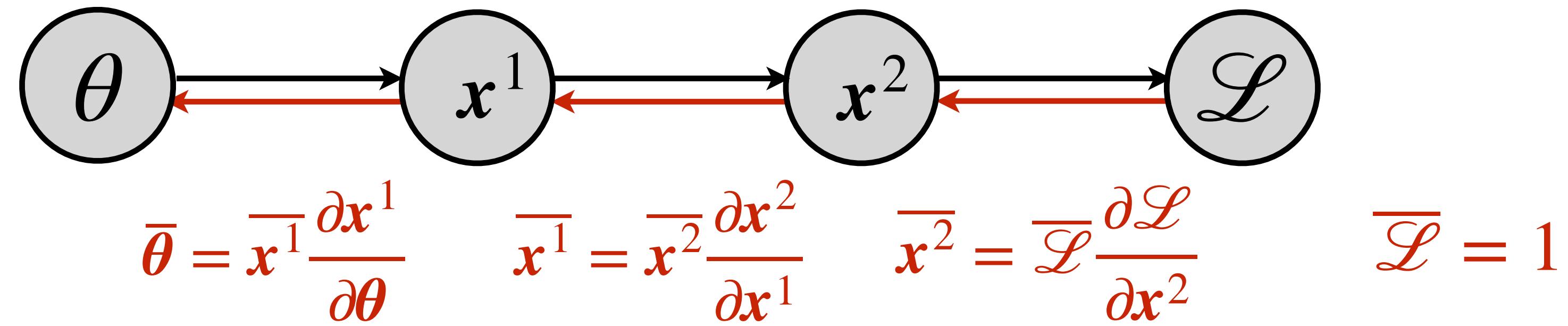
For backward, define “adjoint” $\bar{x} = \frac{\partial \mathcal{L}}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Forward: function evaluation

Chain graph

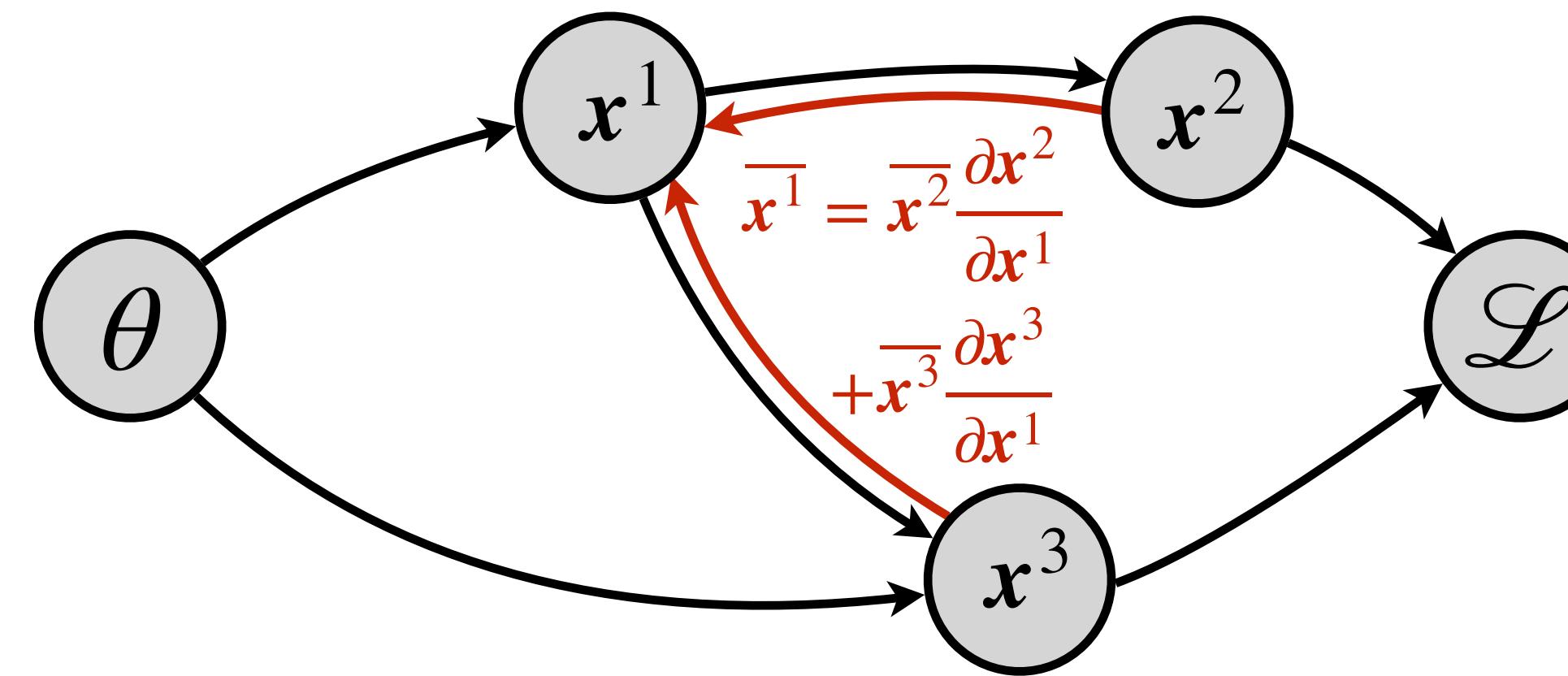


For backward, define “adjoint” $\bar{x} = \frac{\partial L}{\partial x}$

Backpropagates the adjoint via Vector-Jacobian Product

Computation Graph

Directed
acyclic graph

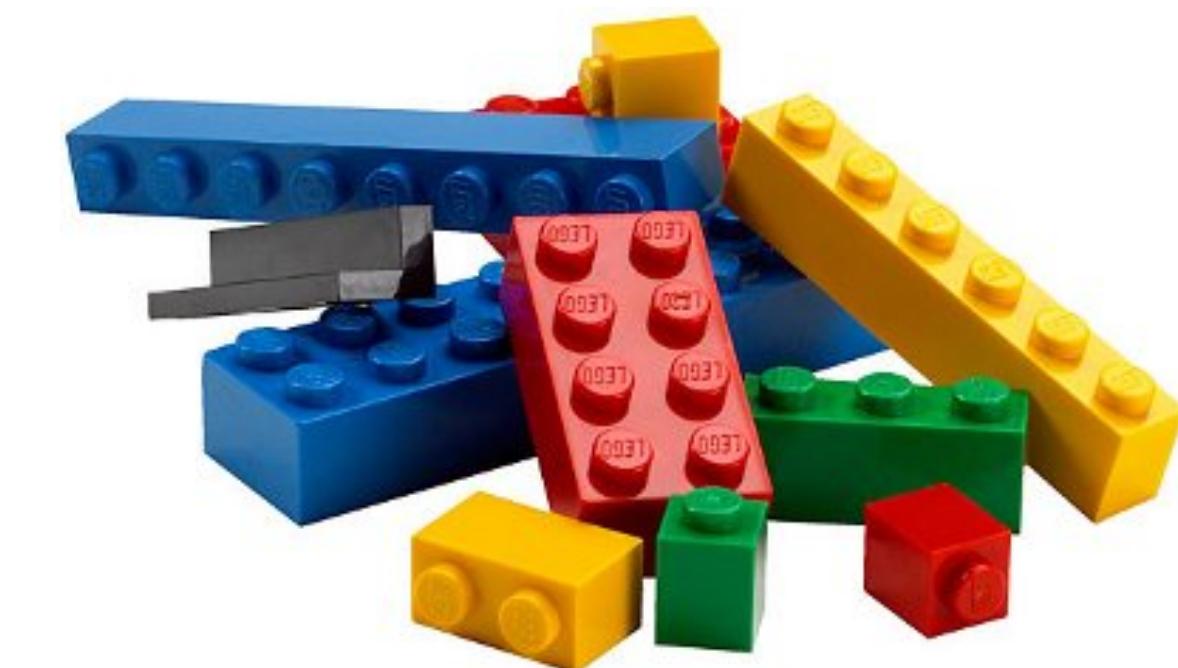


$$\bar{x}^i = \sum_{j: \text{child of } i} \bar{x}^j \frac{\partial x^j}{\partial x^i} \quad \text{with} \quad \bar{\mathcal{L}} = 1$$

Efficient computes gradient of arbitrary program

How to think about AD ?

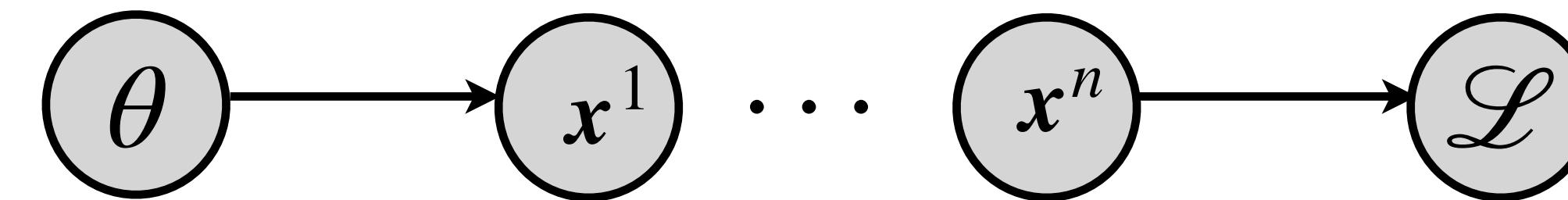
- AD is modular, and one can control its granularity
- Advantage of customized primitives
 - Reducing memory usage
 - Increasing numerical stability
 - Call to external libraries



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

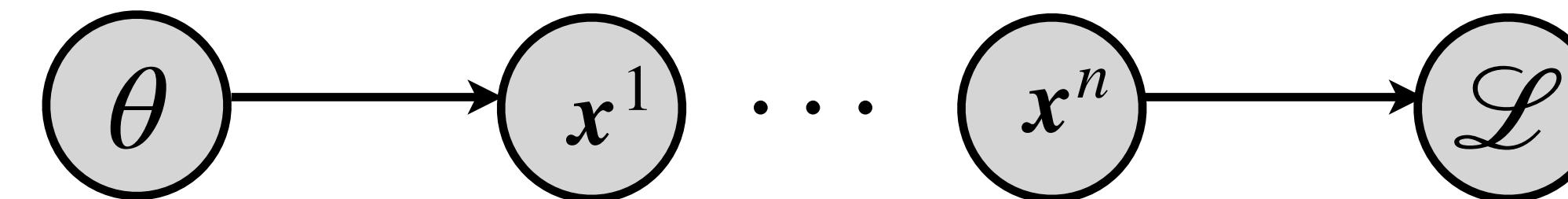


$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial x^n} \frac{\partial x^n}{\partial x^{n-1}} \cdots \frac{\partial x^2}{\partial x^1} \frac{\partial x^1}{\partial \theta}$$

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



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

Forward mode AD: Jacobian-Vector Product

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

Examples fo customized primitives

Forward

$$y = xW$$

$$y = \sigma(x)$$

...

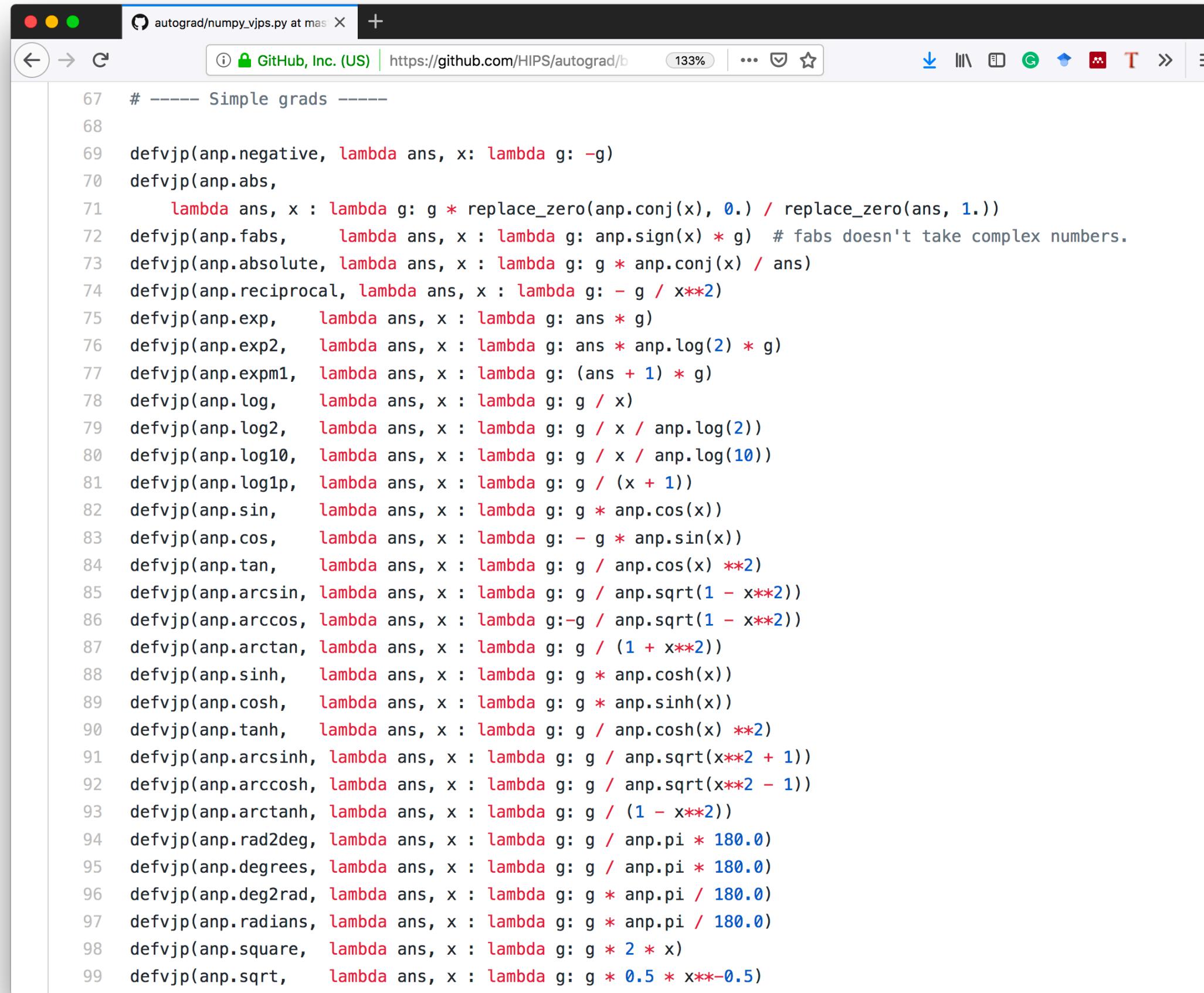
Backward

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

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

...

Examples for customized primitives



A screenshot of a web browser window displaying the file `numpy_vjps.py` from the `autograd` repository on GitHub. The code is a Python script containing numerous definitions of custom gradient functions for various mathematical operations. The browser interface shows the file path, a progress bar at 133%, and standard navigation and search tools.

```
# ----- 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.acos,     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

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

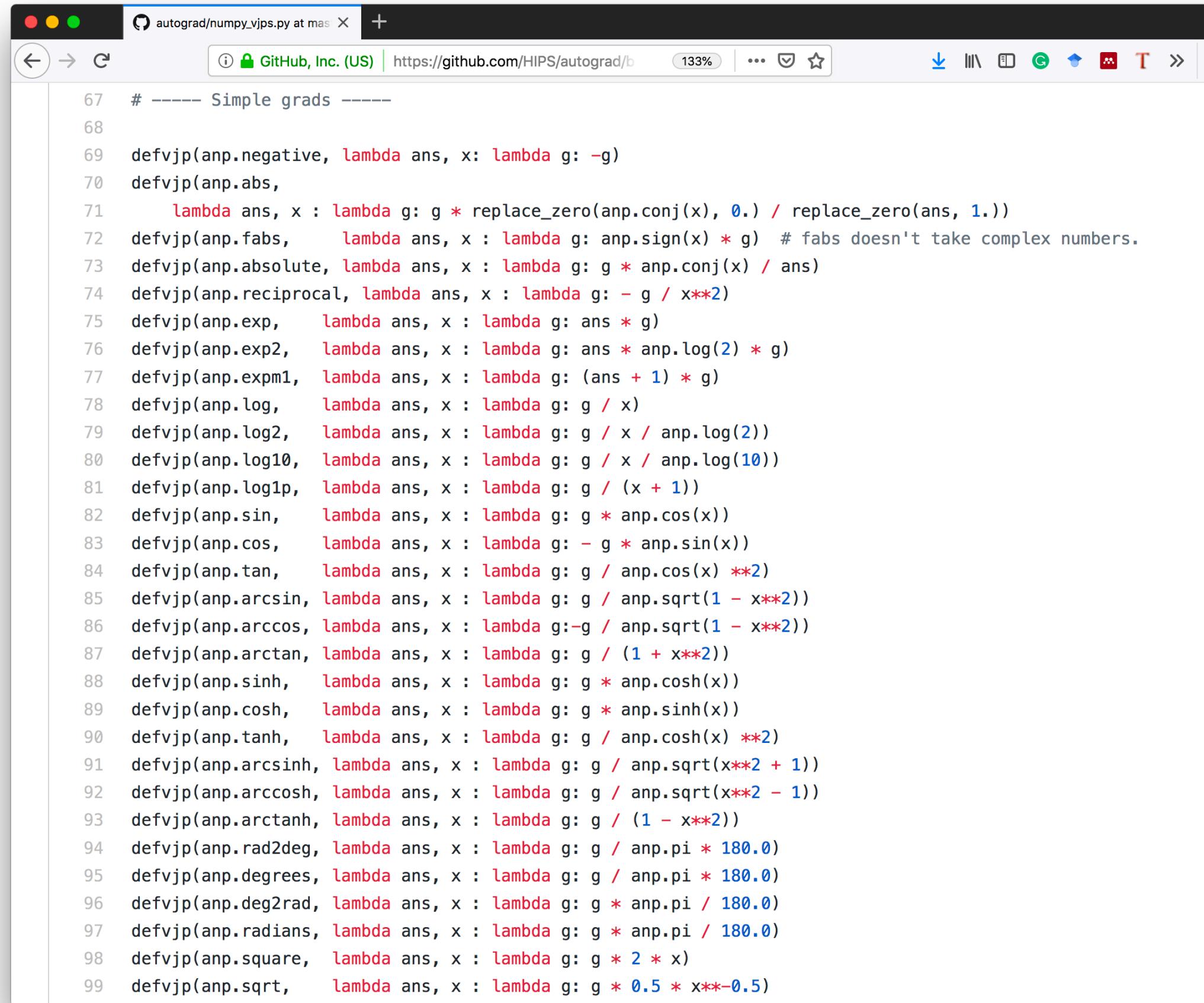
$$\bar{x} = \bar{y} \odot \sigma'(x)$$

...

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

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

Examples for customized primitives



A screenshot of a web browser displaying the file `numpy_vjps.py` from the GitHub repository `HIPS/autograd`. The code is a series of `defvjp` definitions for various NumPy functions, each mapping a function to a lambda function that computes its gradient. The browser interface shows the code in a monospaced font, with line numbers on the left and standard browser controls at the top.

```
67 # ----- Simple grads -----
68
69 defvjp(anp.negative, lambda ans, x: lambda g: -g)
70 defvjp(anp.abs,
71     lambda ans, x : lambda g: g * replace_zero(anp.conj(x), 0.) / replace_zero(ans, 1.))
72 defvjp(anp.fabs,      lambda ans, x : lambda g: anp.sign(x) * g) # fabs doesn't take complex numbers.
73 defvjp(anp.absolute, lambda ans, x : lambda g: g * anp.conj(x) / ans)
74 defvjp(anp.reciprocal, lambda ans, x : lambda g: - g / x**2)
75 defvjp(anp.exp,      lambda ans, x : lambda g: ans * g)
76 defvjp(anp.exp2,      lambda ans, x : lambda g: ans * anp.log(2) * g)
77 defvjp(anp.expm1,    lambda ans, x : lambda g: (ans + 1) * g)
78 defvjp(anp.log,      lambda ans, x : lambda g: g / x)
79 defvjp(anp.log2,      lambda ans, x : lambda g: g / x / anp.log(2))
80 defvjp(anp.log10,    lambda ans, x : lambda g: g / x / anp.log(10))
81 defvjp(anp.log1p,    lambda ans, x : lambda g: g / (x + 1))
82 defvjp(anp.sin,      lambda ans, x : lambda g: g * anp.cos(x))
83 defvjp(anp.cos,      lambda ans, x : lambda g: - g * anp.sin(x))
84 defvjp(anp.tan,      lambda ans, x : lambda g: g / anp.cos(x)**2)
85 defvjp(anp.arcsin,   lambda ans, x : lambda g: g / anp.sqrt(1 - x**2))
86 defvjp(anp.acccos,   lambda ans, x : lambda g:-g / anp.sqrt(1 - x**2))
87 defvjp(anp.arctan,   lambda ans, x : lambda g: g / (1 + x**2))
88 defvjp(anp.sinh,     lambda ans, x : lambda g: g * anp.cosh(x))
89 defvjp(anp.cosh,     lambda ans, x : lambda g: g * anp.sinh(x))
90 defvjp(anp.tanh,     lambda ans, x : lambda g: g / anp.cosh(x)**2)
91 defvjp(anp.arcsinh,  lambda ans, x : lambda g: g / anp.sqrt(x**2 + 1))
92 defvjp(anp.arccosh,  lambda ans, x : lambda g: g / anp.sqrt(x**2 - 1))
93 defvjp(anp.arctanh,  lambda ans, x : lambda g: g / (1 - x**2))
94 defvjp(anp.rad2deg,  lambda ans, x : lambda g: g / anp.pi * 180.0)
95 defvjp(anp.degrees,  lambda ans, x : lambda g: g / anp.pi * 180.0)
96 defvjp(anp.deg2rad,  lambda ans, x : lambda g: g * anp.pi / 180.0)
97 defvjp(anp.radians,  lambda ans, x : lambda g: g * anp.pi / 180.0)
98 defvjp(anp.square,   lambda ans, x : lambda g: g * 2 * x)
99 defvjp(anp.sqrt,     lambda ans, x : lambda g: g * 0.5 * x**-0.5)
```

Backward

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

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

...

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 program, 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 Andrychowicz¹, Misha Denil¹, Sergio Gómez Colmenarejo¹, Matthew W. Hoffman¹, David Pfau¹, Tom Schaul¹, Brendan Shillingford^{1,2}, Nando de Freitas^{1,2,3}

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

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

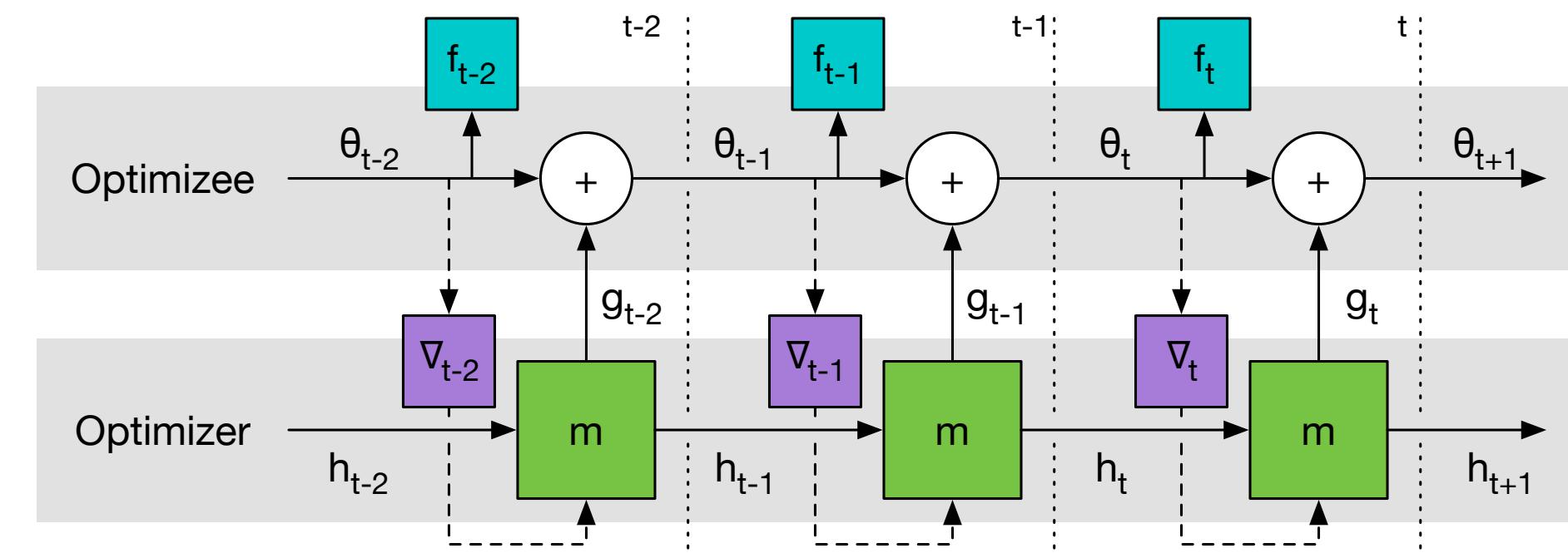
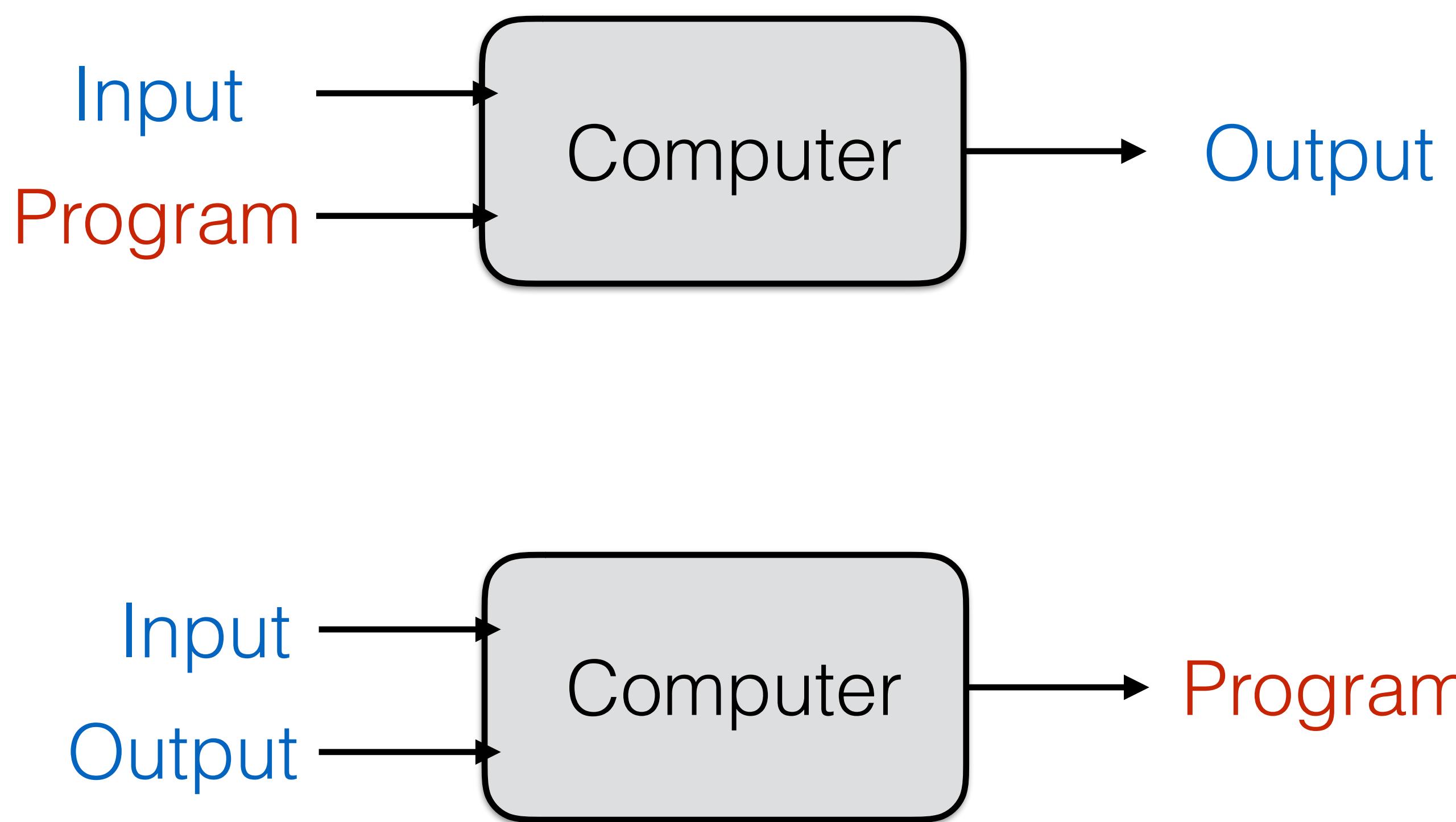


Figure 2: Computational graph used for computing the gradient of the optimizer.

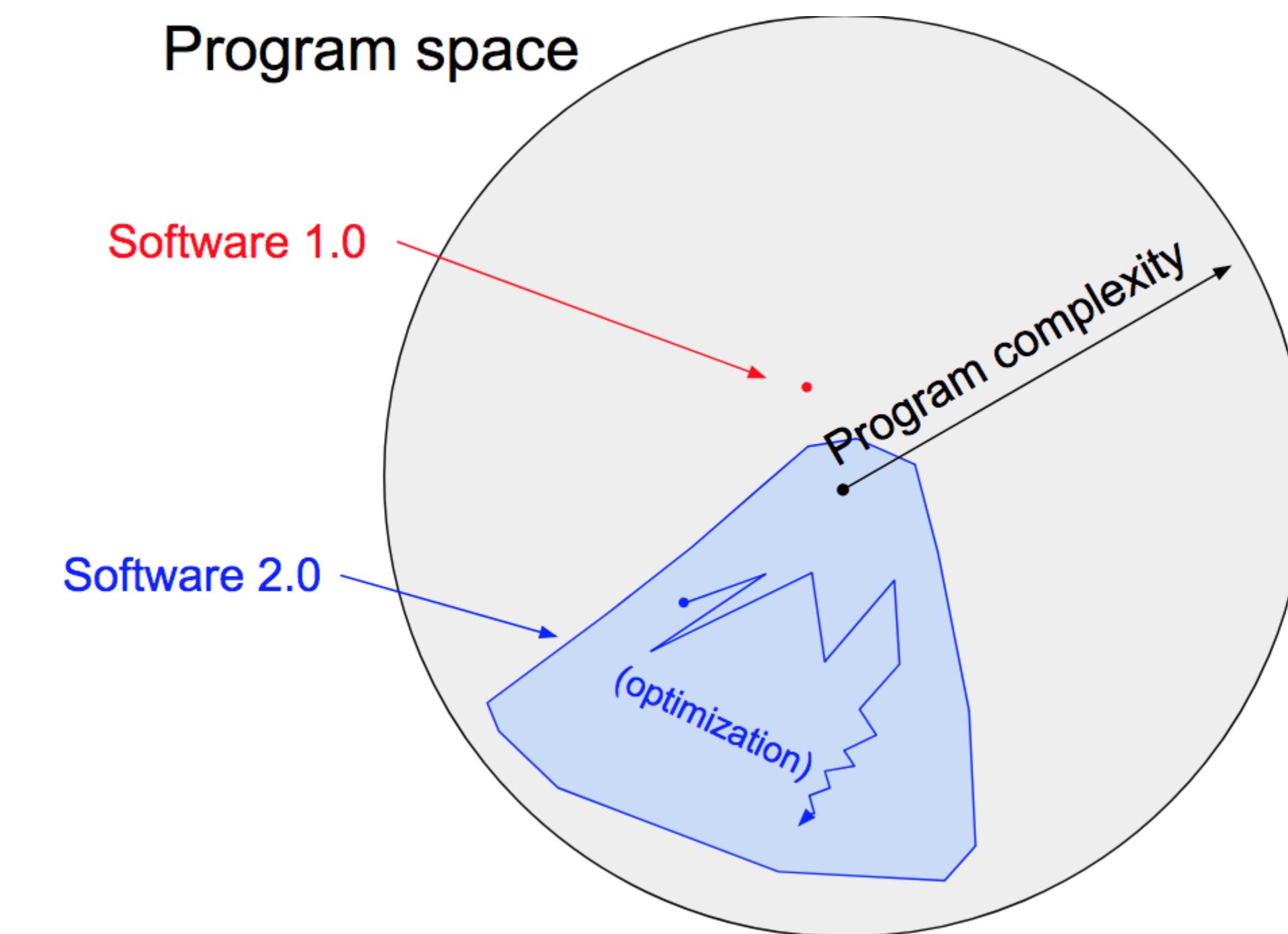
Differentiable Programming



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

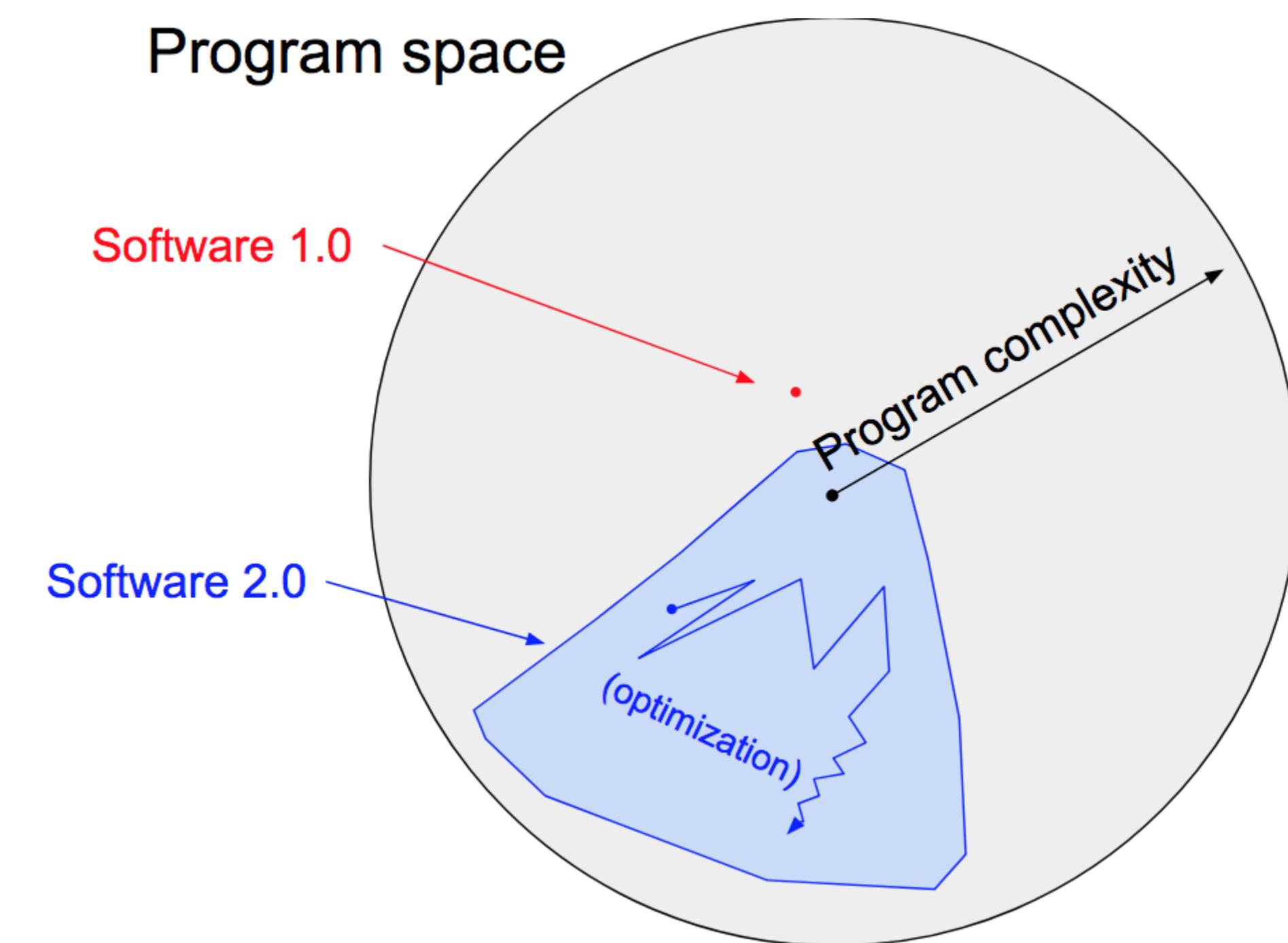
- Computationally homogeneous
- Simple to bake into silicon
- Constant running time
- Constant memory usage
- Highly portable & agile
- Modules can meld into an optimal whole
- **Better than humans**



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

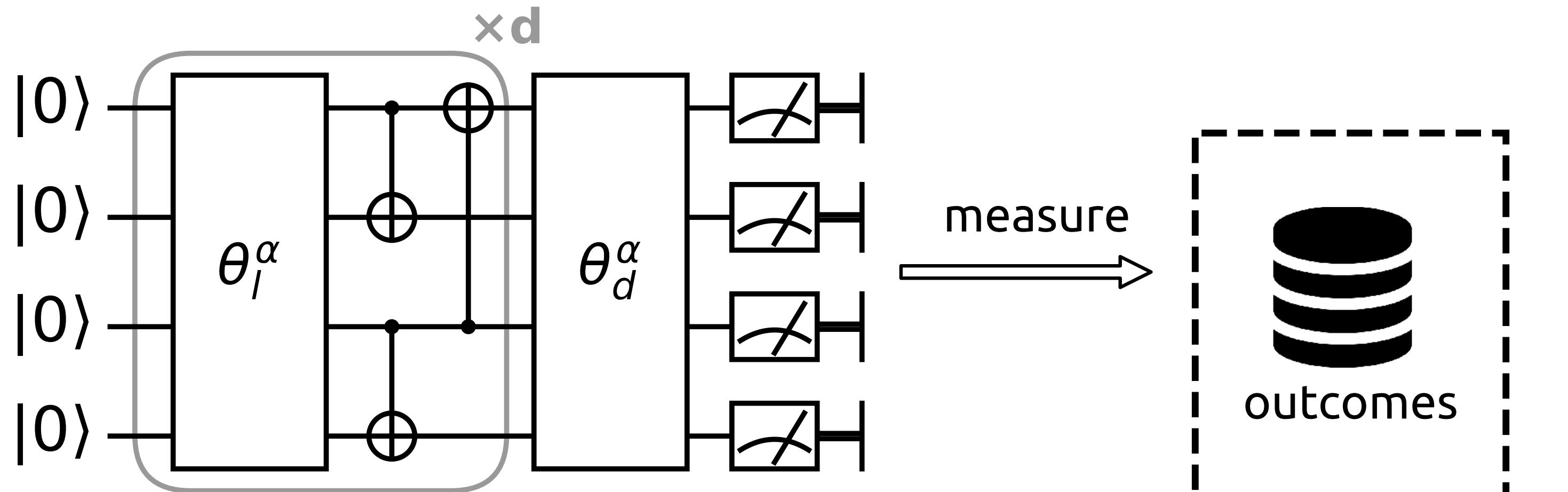
- Most linear algebra operations (Eigen, SVD!) are differentiable
- ODE integrators are differentiable with $O(1)$ memory
- Differentiable ray tracer and Differentiable fluid simulations
- Differentiable Monte Carlo/Tensor Network/Functional RG/
Dynamical Mean Field Theory/Density Functional Theory...

Differentiable programming is more than training neural networks

Differentiable Quantum Programming

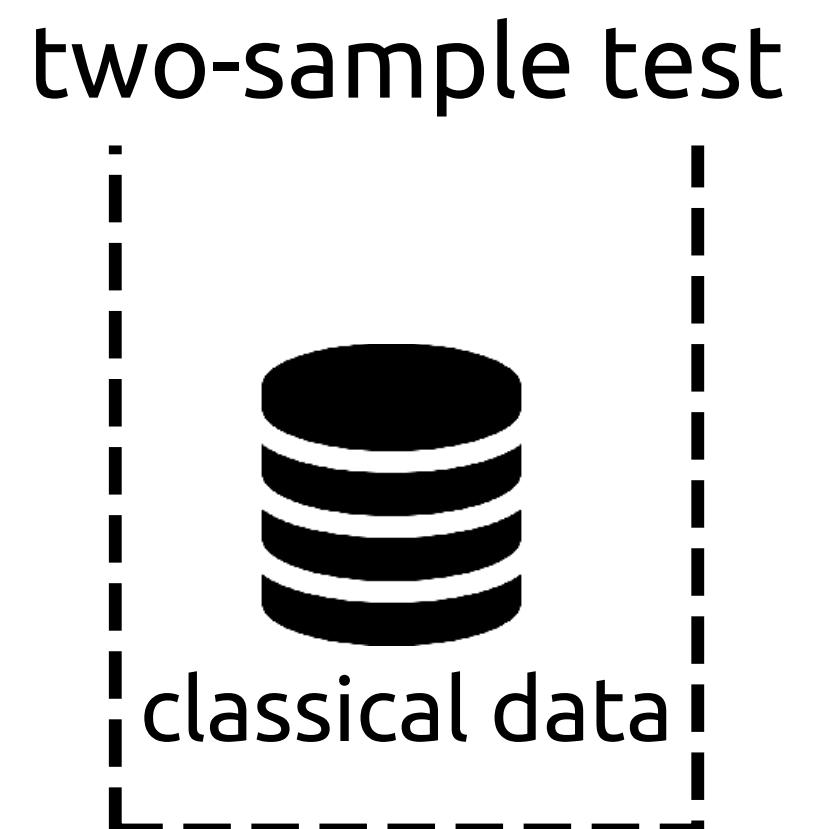
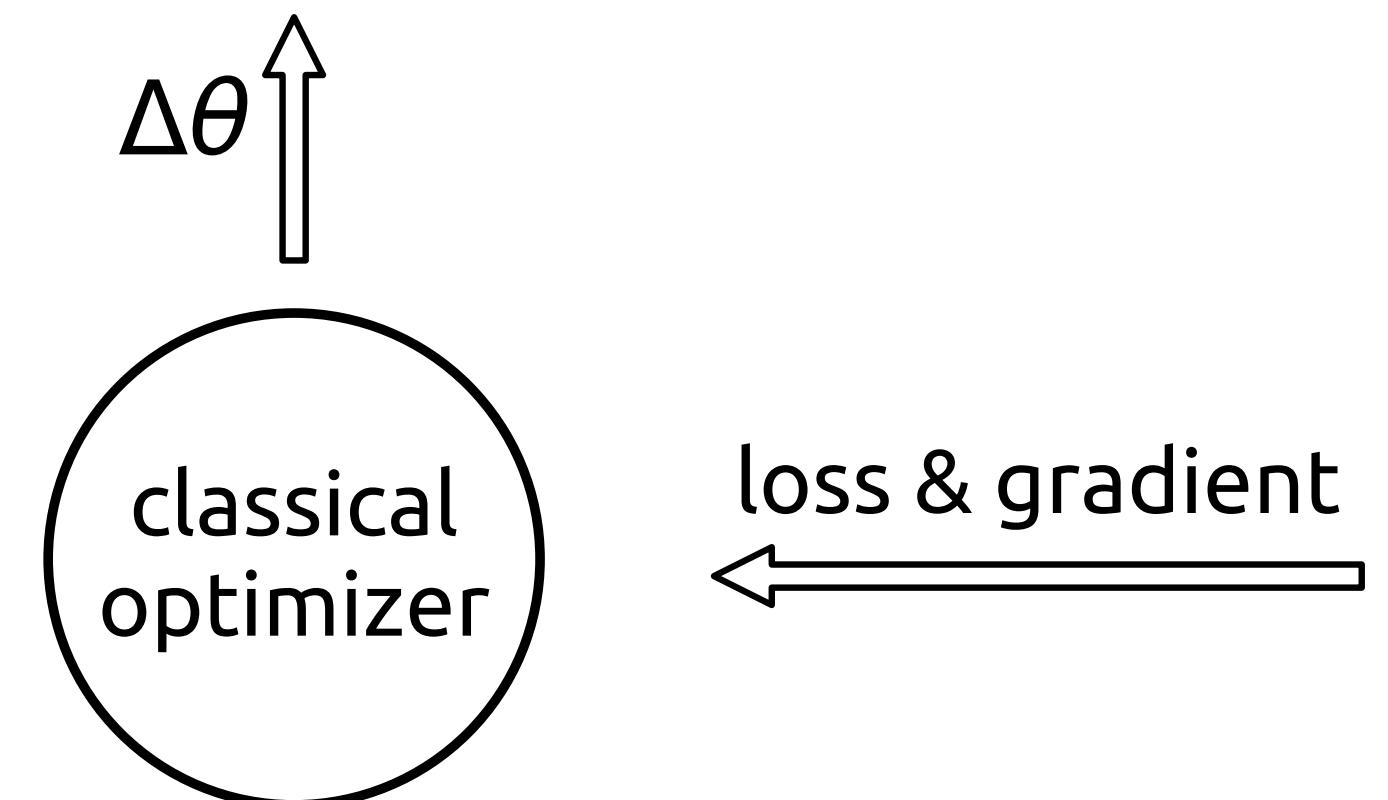
Short term:

What can we do with
circuits of limited depth ?



Long term:

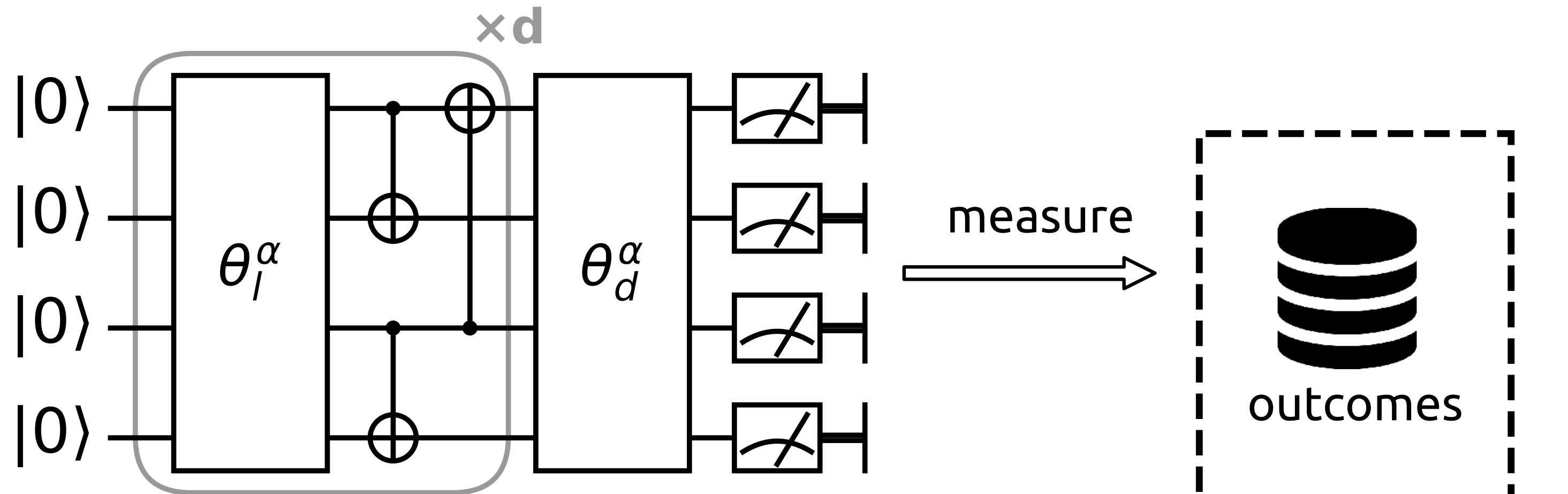
Are we really good at
programming quantum computers ?



Differentiable Quantum Programming

Short term:

What can we do with circuits of limited depth ?



Long term:

Are we really good at programming quantum computers ?

Quantum code

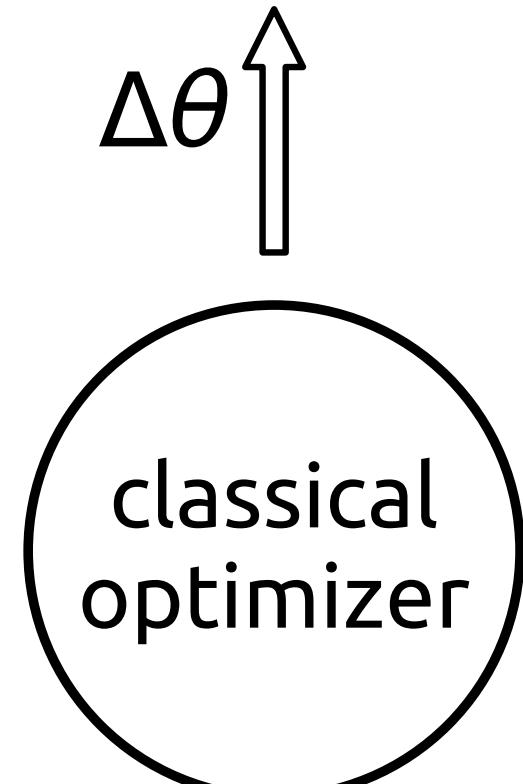


Andrej Karpathy ✅
@karpathy

Gradient descent can write code better than you. I'm sorry.



Following



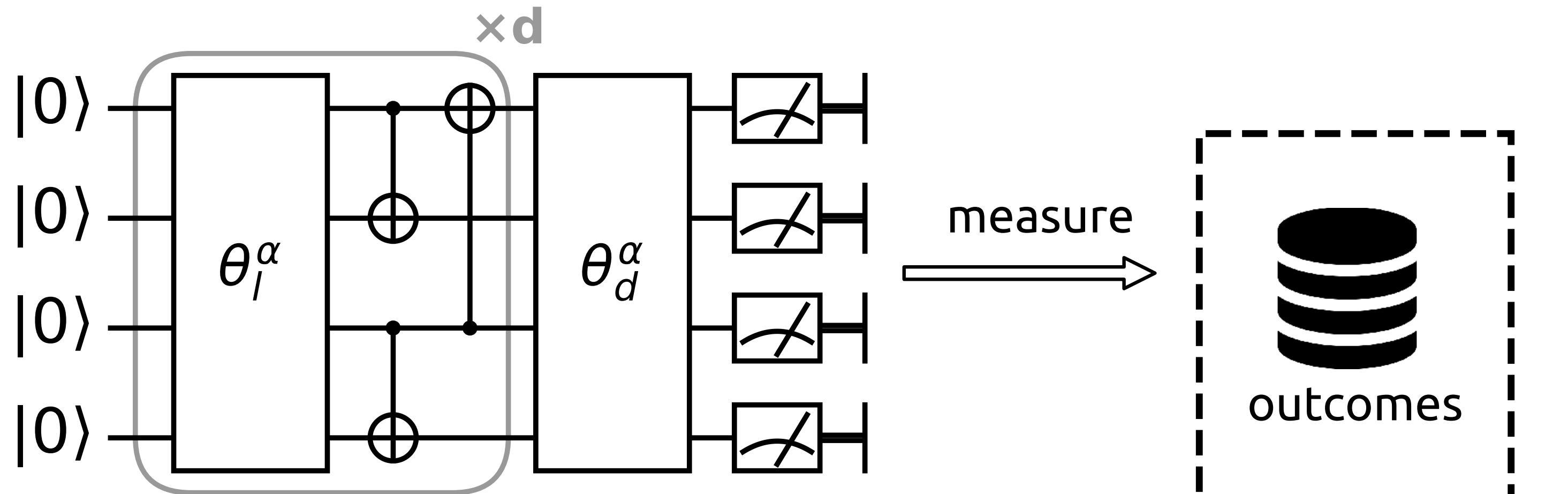
two-sample test



Differentiable Quantum Programming

Short term:

What can we do with circuits of limited depth ?



Long term:

Are we really good at programming quantum computers ?

Quantum code

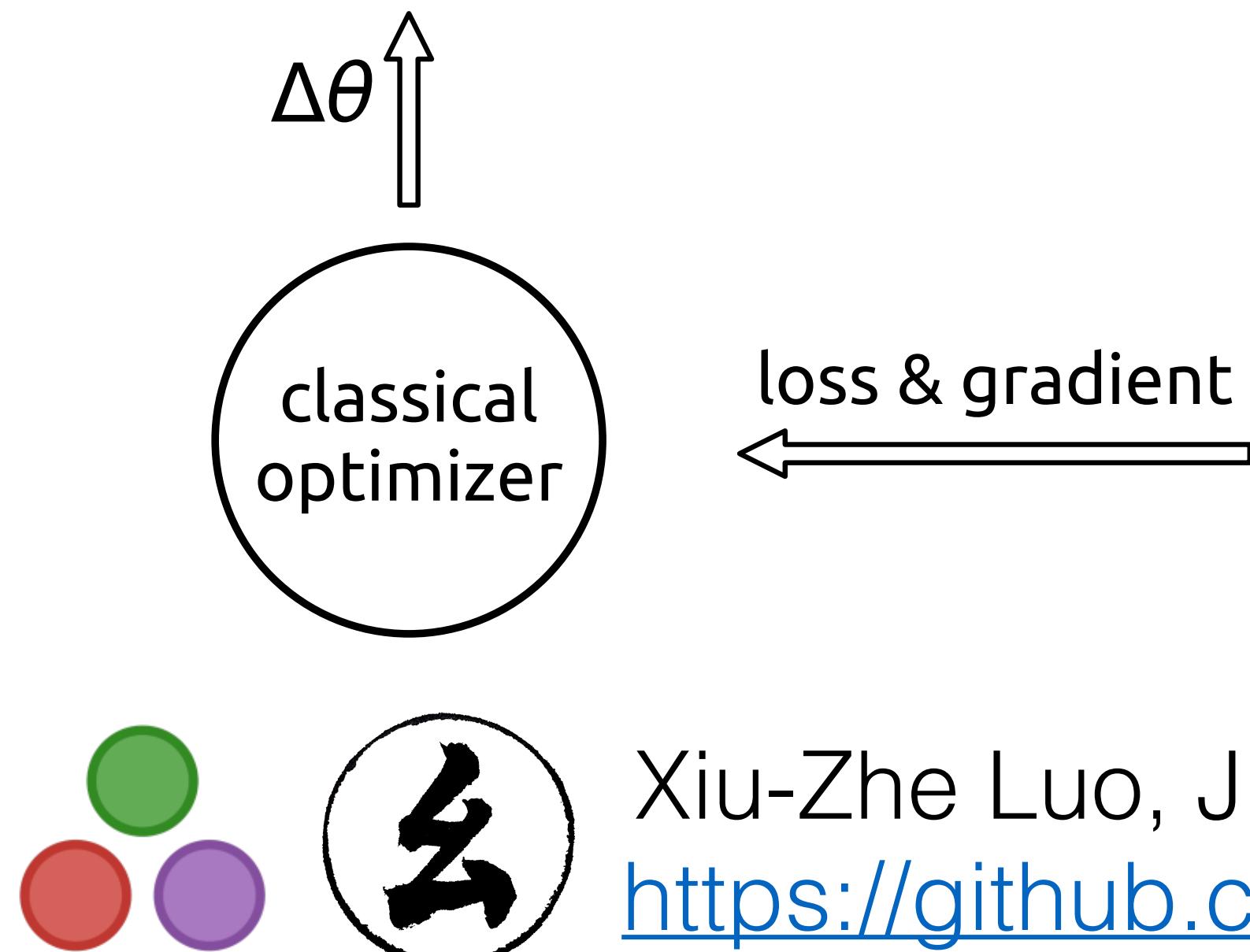


Andrej Karpathy ✅
@karpathy

Gradient descent can write code better than you. I'm sorry.



Following



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

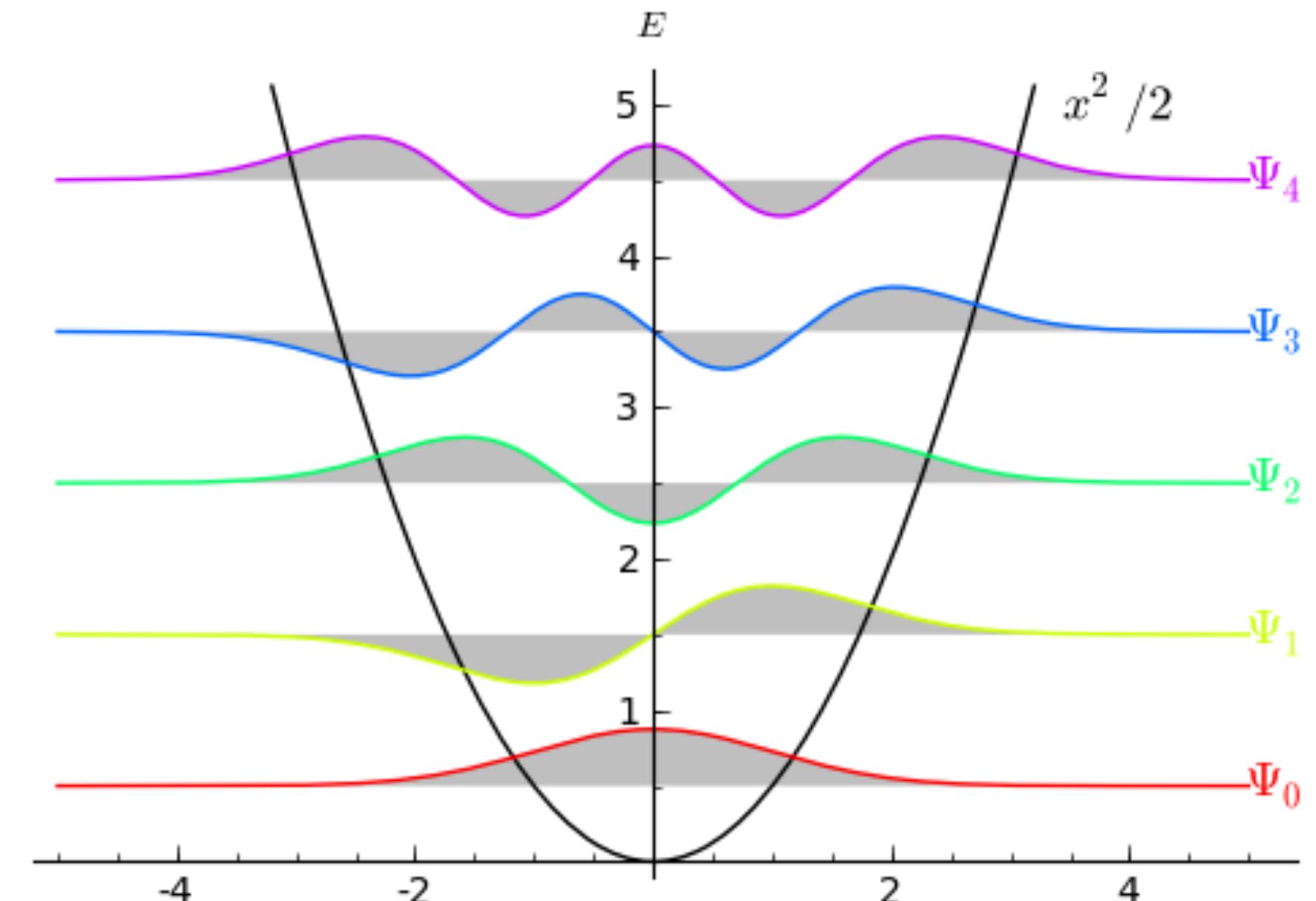
Reverse mode: How should I change H given
 $\partial\mathcal{L}/\partial\Psi$ and $\partial\mathcal{L}/\partial\Lambda$? Inverse
perturbation theory!

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

 PyTorch



theano


TensorFlow

 Keras

 flux

 Zygote

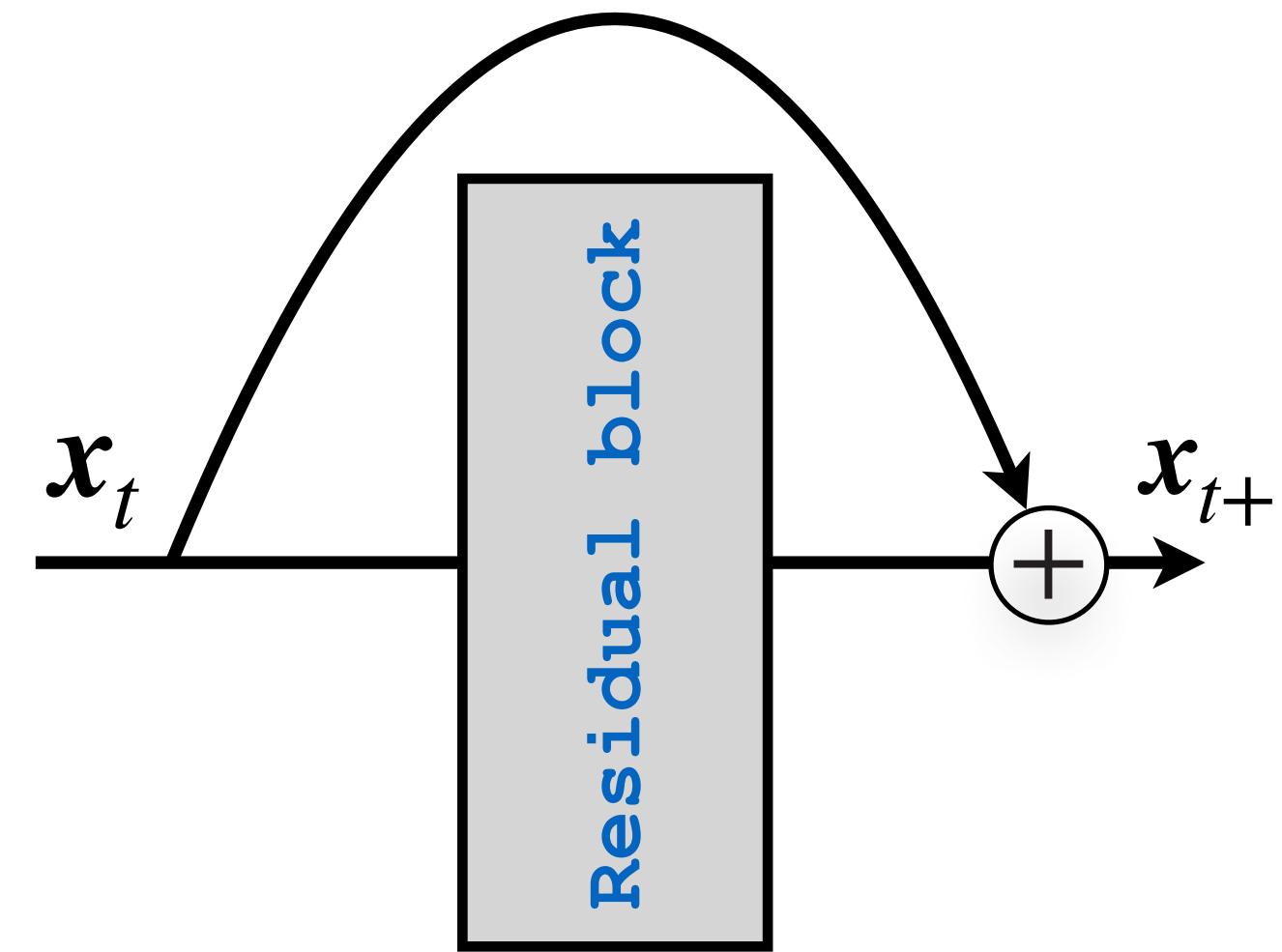
Current support for AD*

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

*as of May 2019

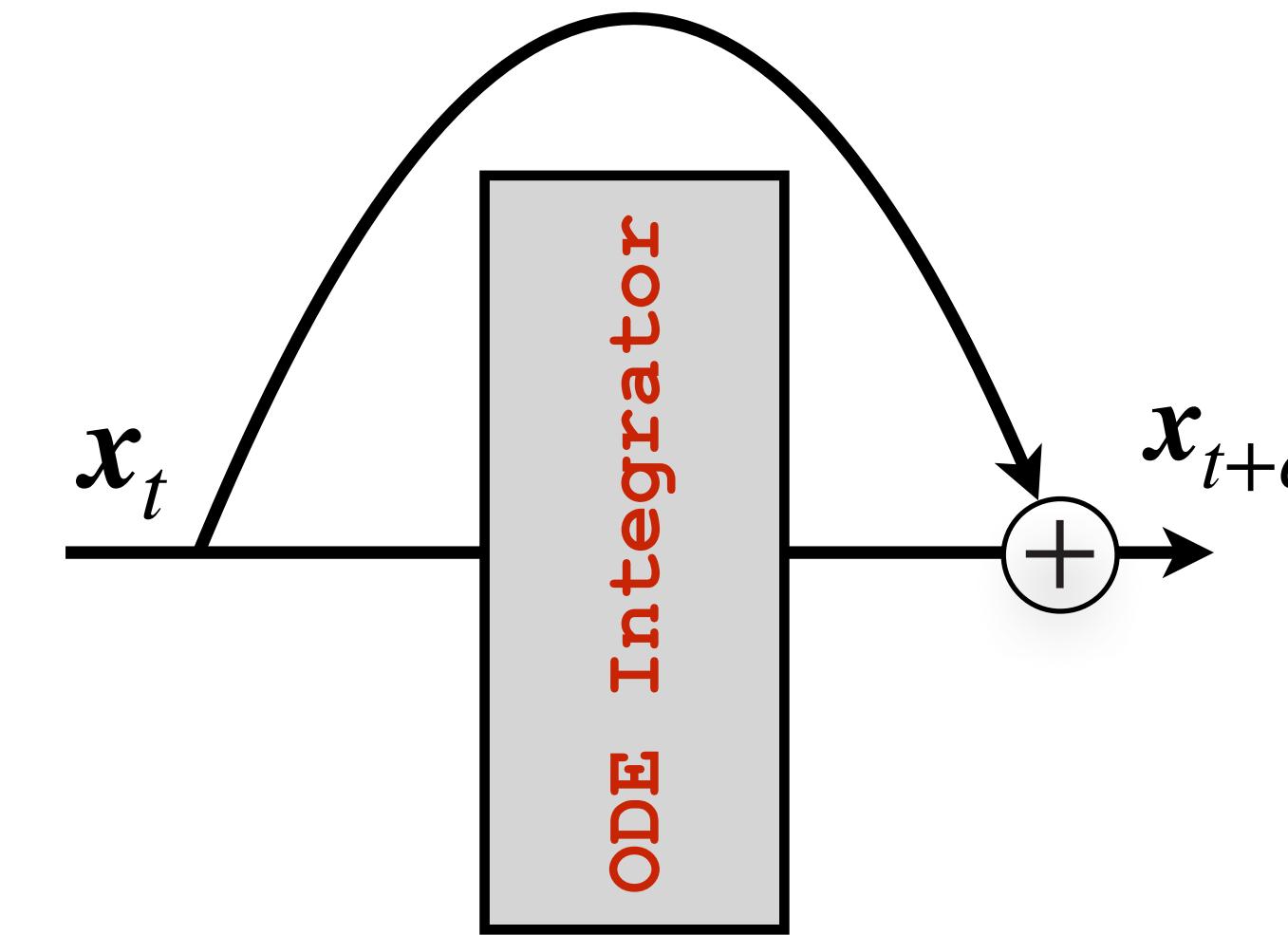
Neural Ordinary Differential Equations

Residual network



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

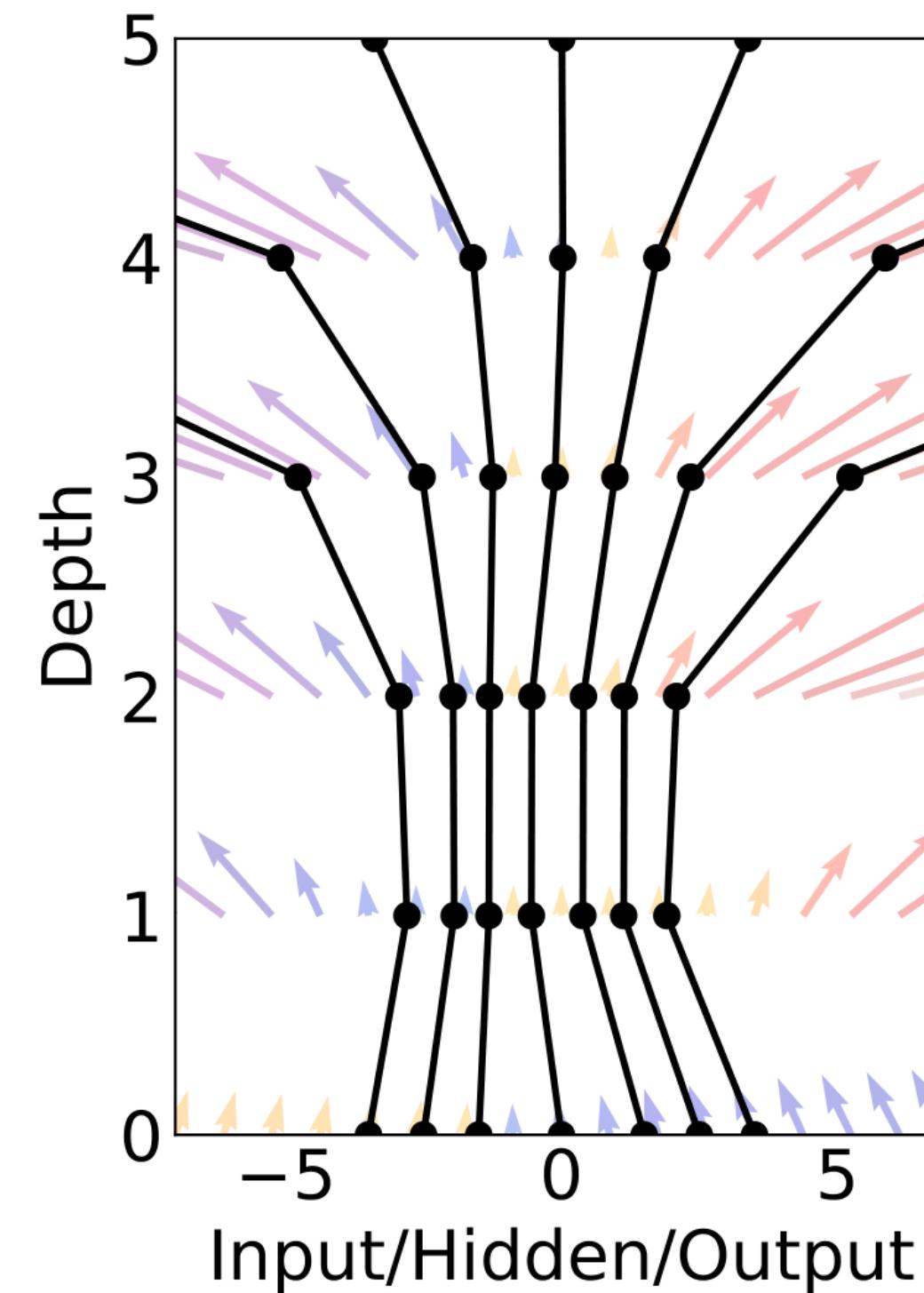
ODE integration



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

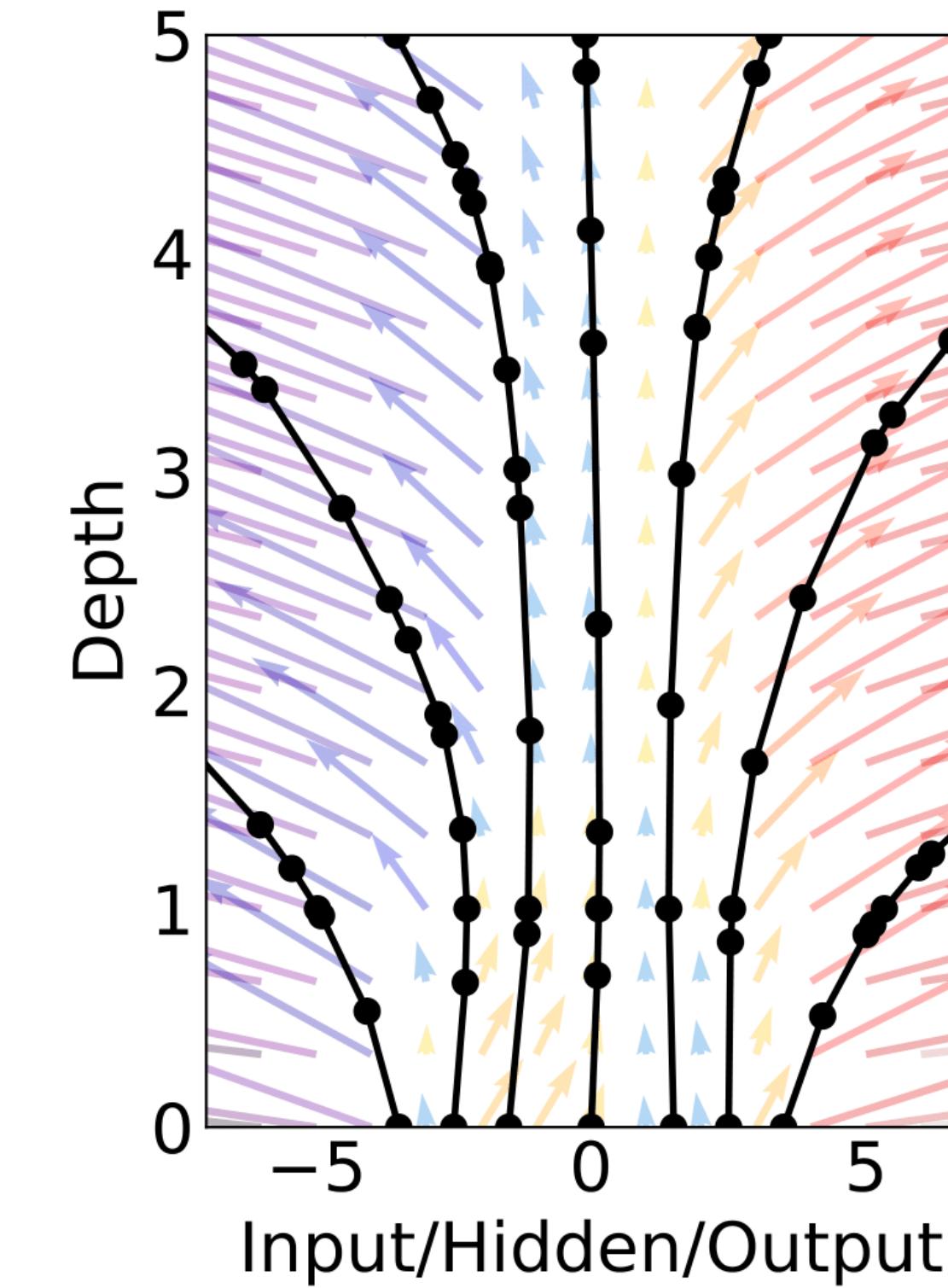
Neural Ordinary Differential Equations

Residual network



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

ODE integration



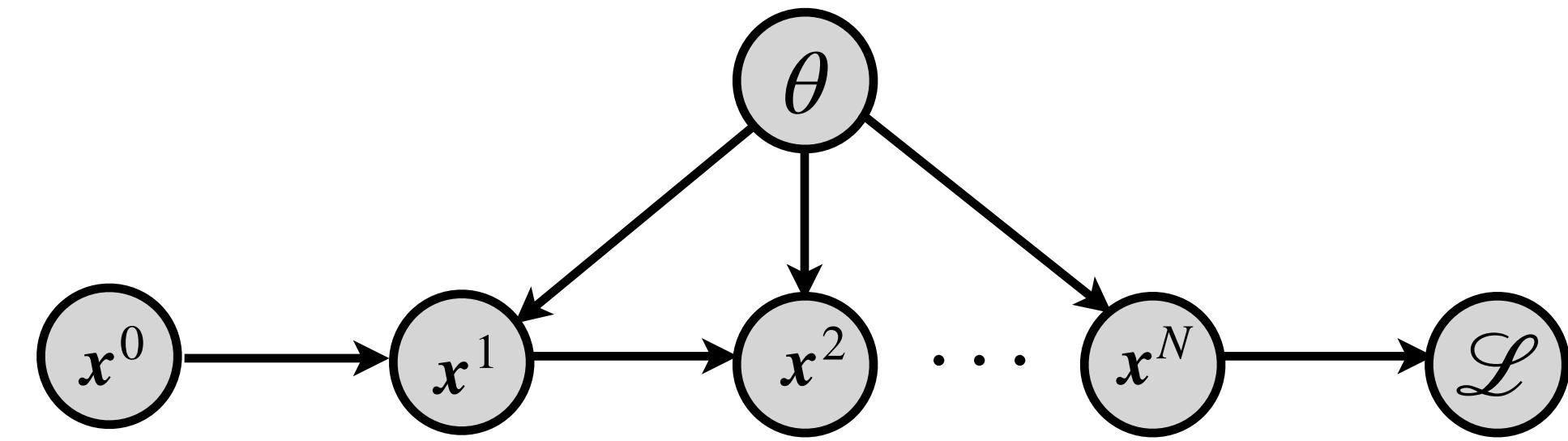
$$dx/dt = f(x)$$

Chen et al, 1806.07366 NIPS '18 Best paper award

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

Backpropagate through ODE solver

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



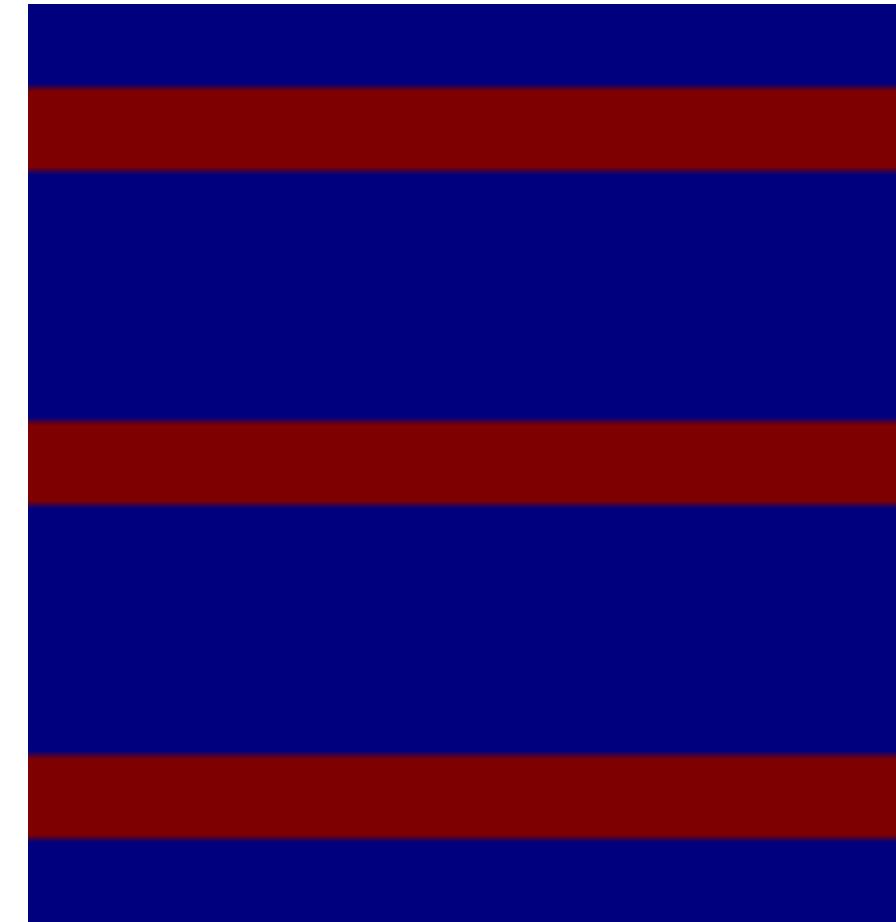
Adjoint $\bar{x}(t) = \frac{\partial \mathcal{L}}{\partial x(t)}$ satisfies another ODE

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

Gradient w.r.t. parameter

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

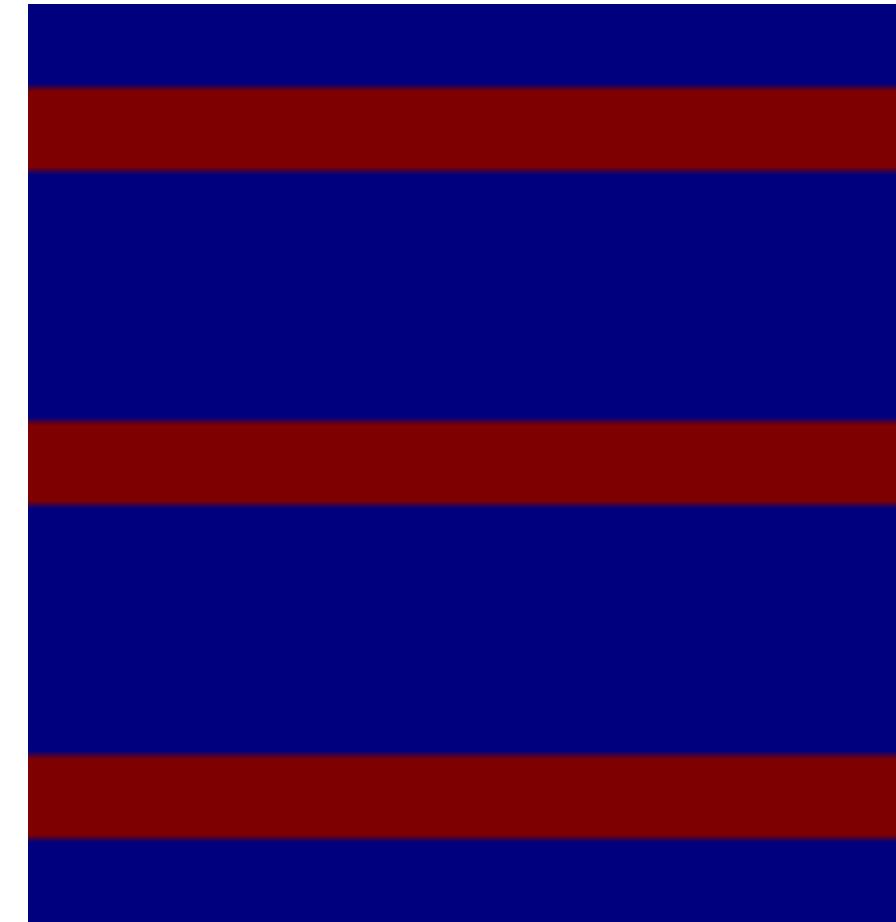
Why do we need Neural ODE ?



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 ?

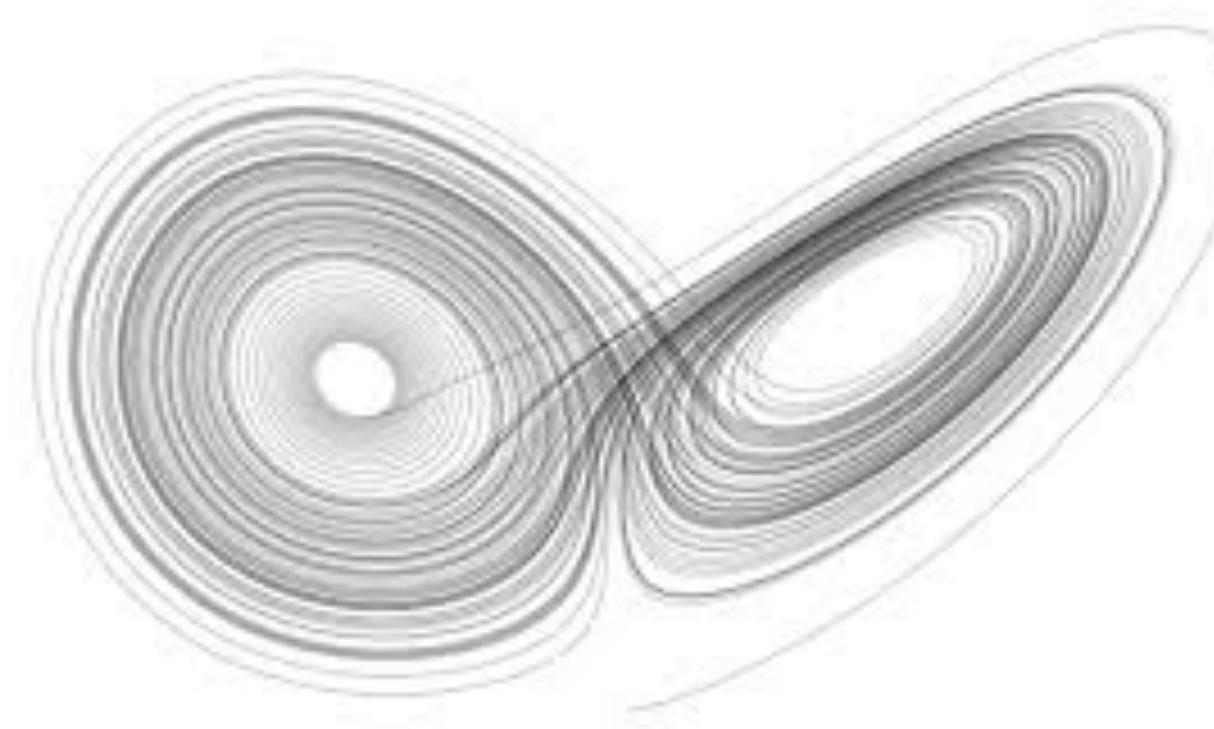


Backpropagating
through a fluid simulation

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

Applications

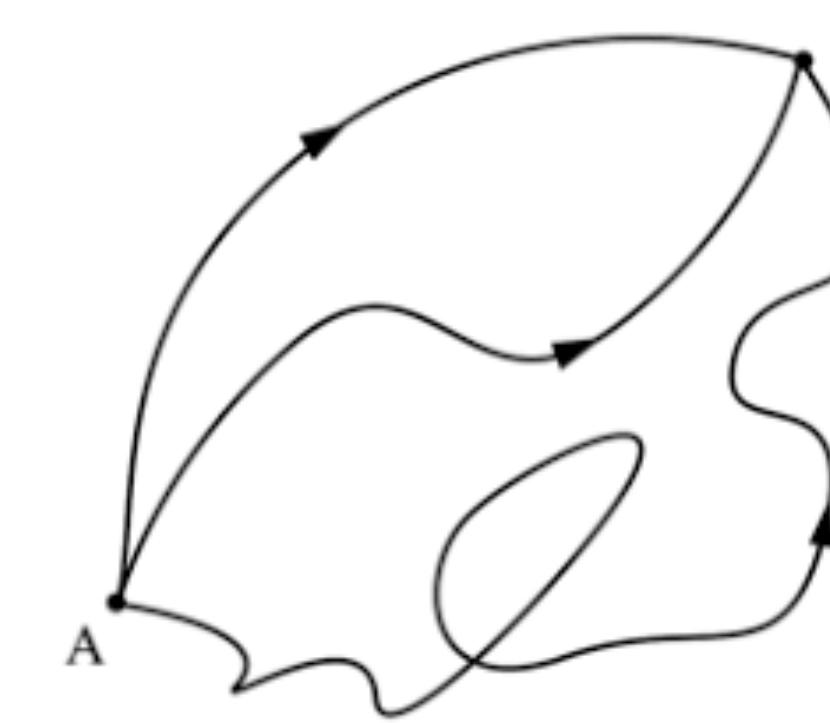
Dynamics systems



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

Classical and quantum control

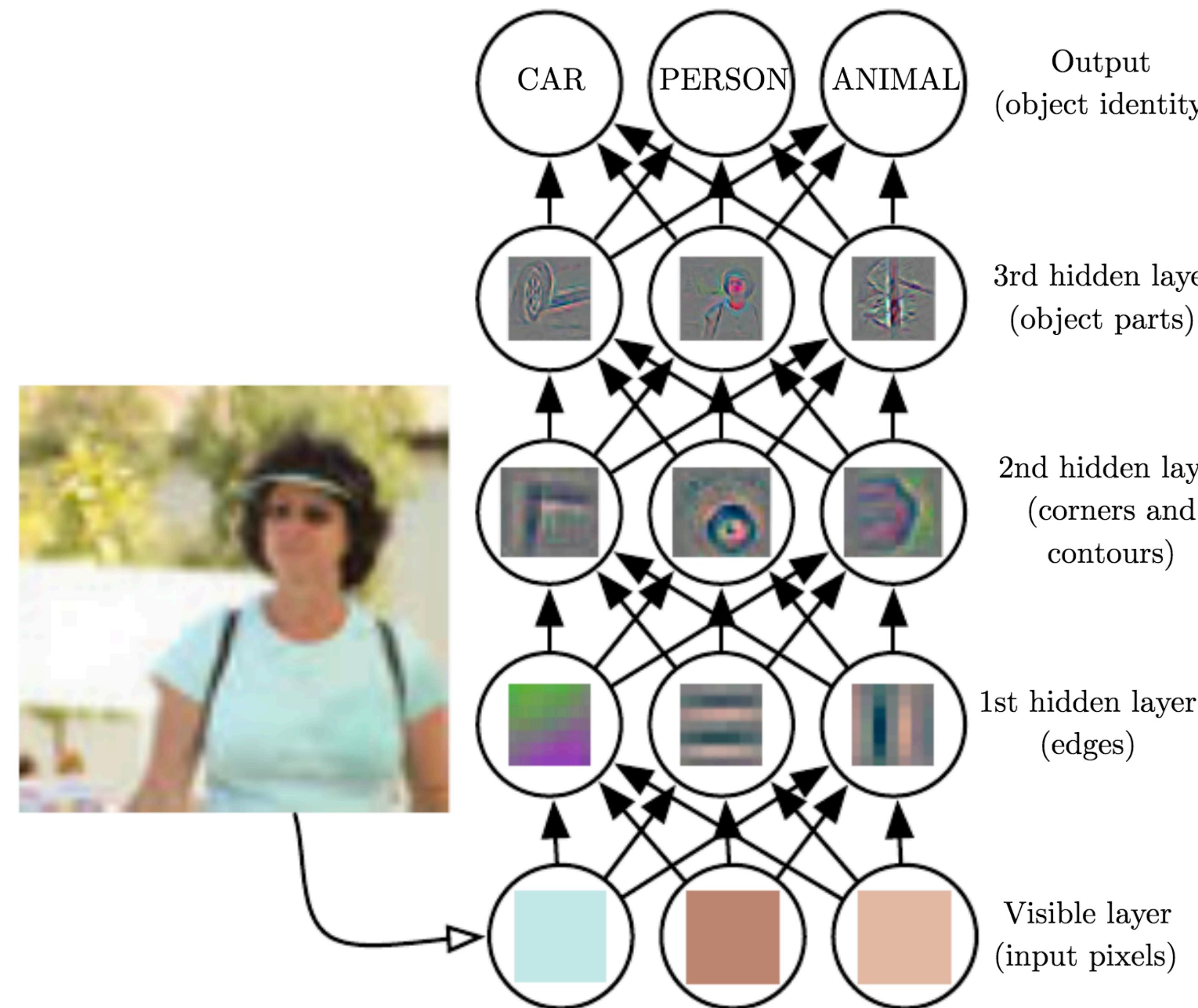
Principle of least actions



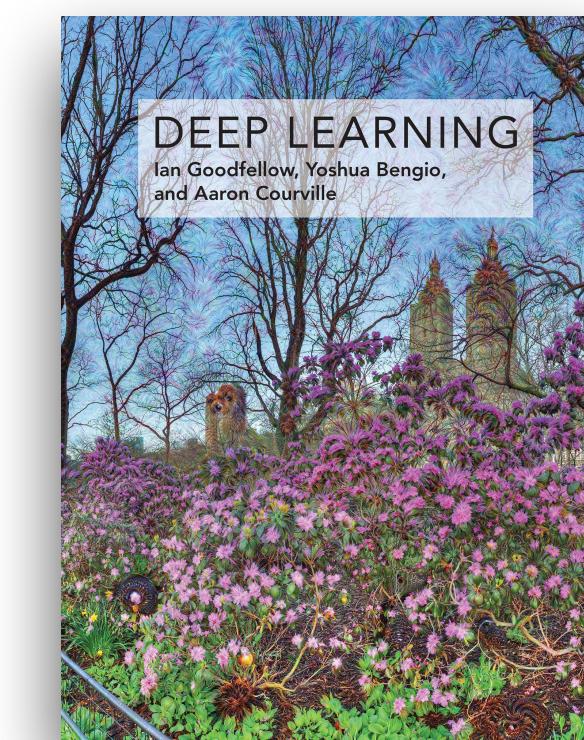
$$S = \int \mathcal{L}(q, \dot{q}, t) dt$$

Optics, (quantum) mechanics, field theory...

Learning Representation

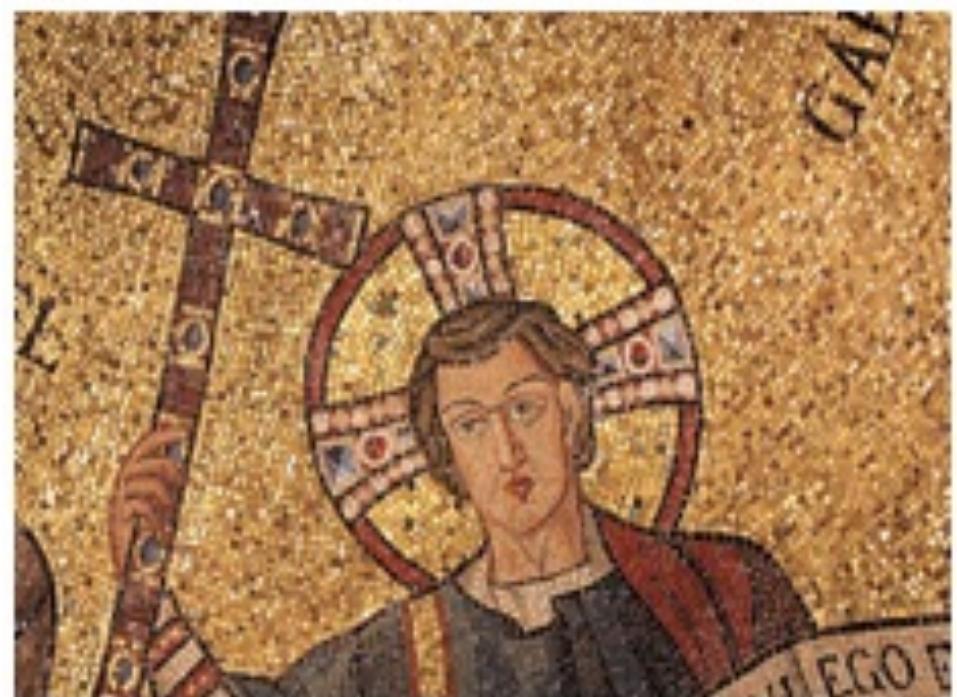


Page 6
Figure 1.2

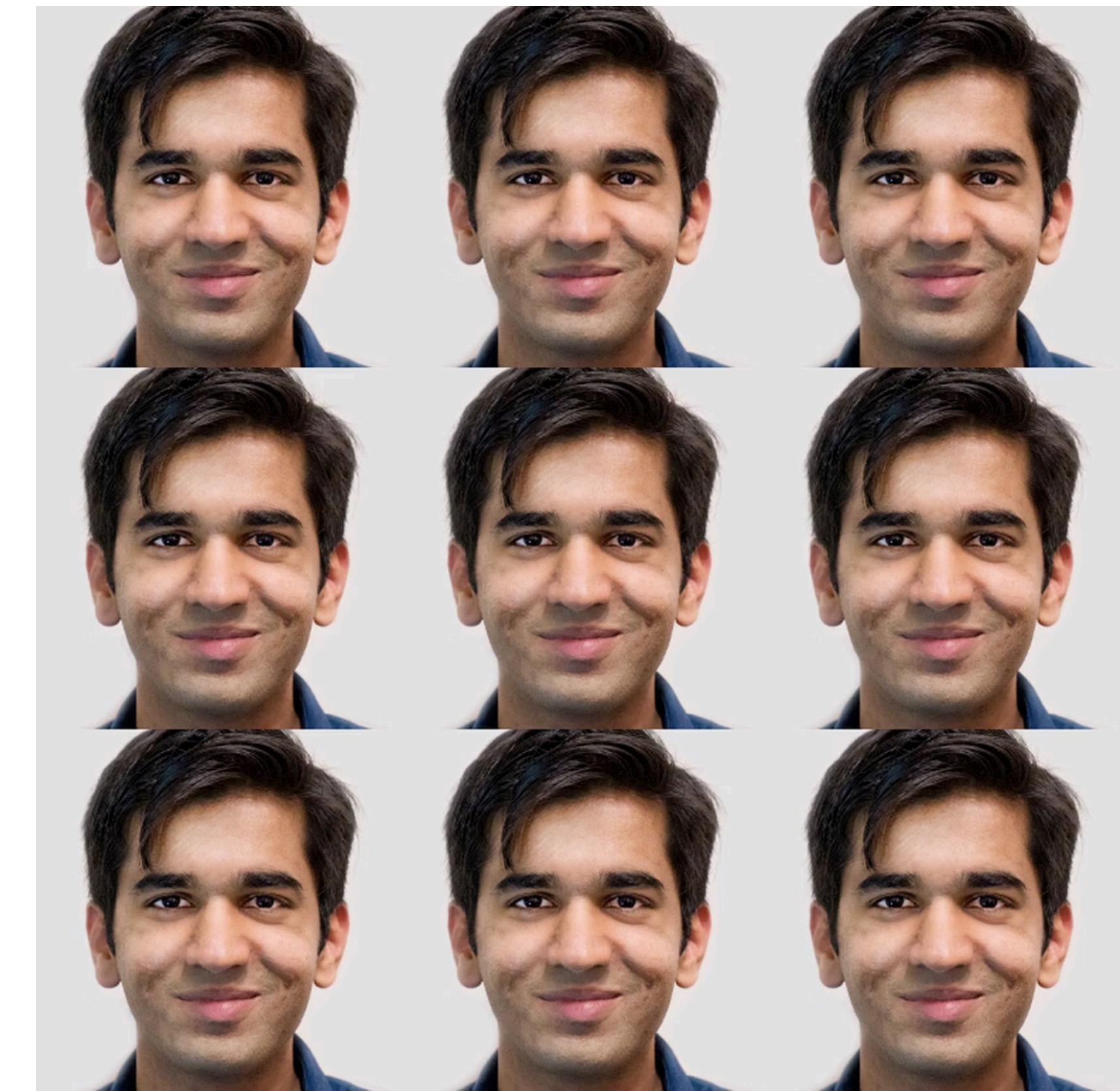


Magic of learning representation

Neural style transfer



Latent space interpolation



Gatys et al, 1508.06576

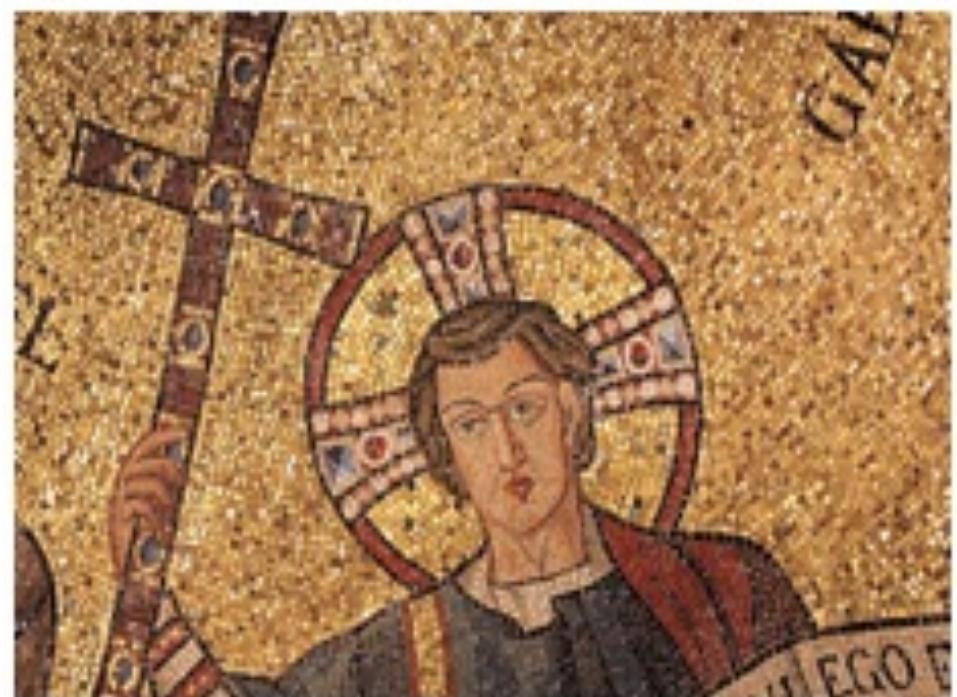


Glow 1807.03039

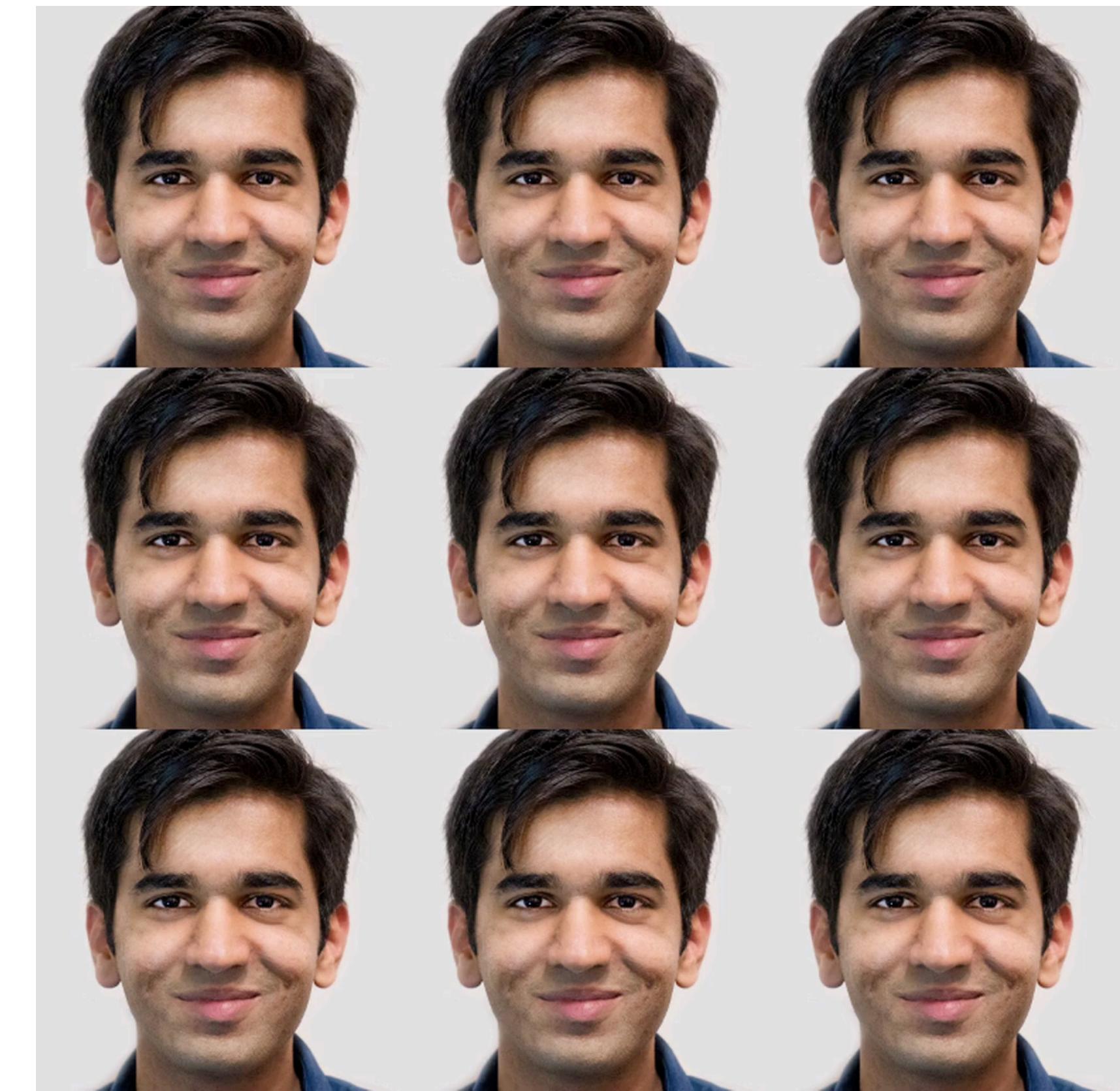
<https://blog.openai.com/glow/>

Magic of learning representation

Neural style transfer



Latent space interpolation



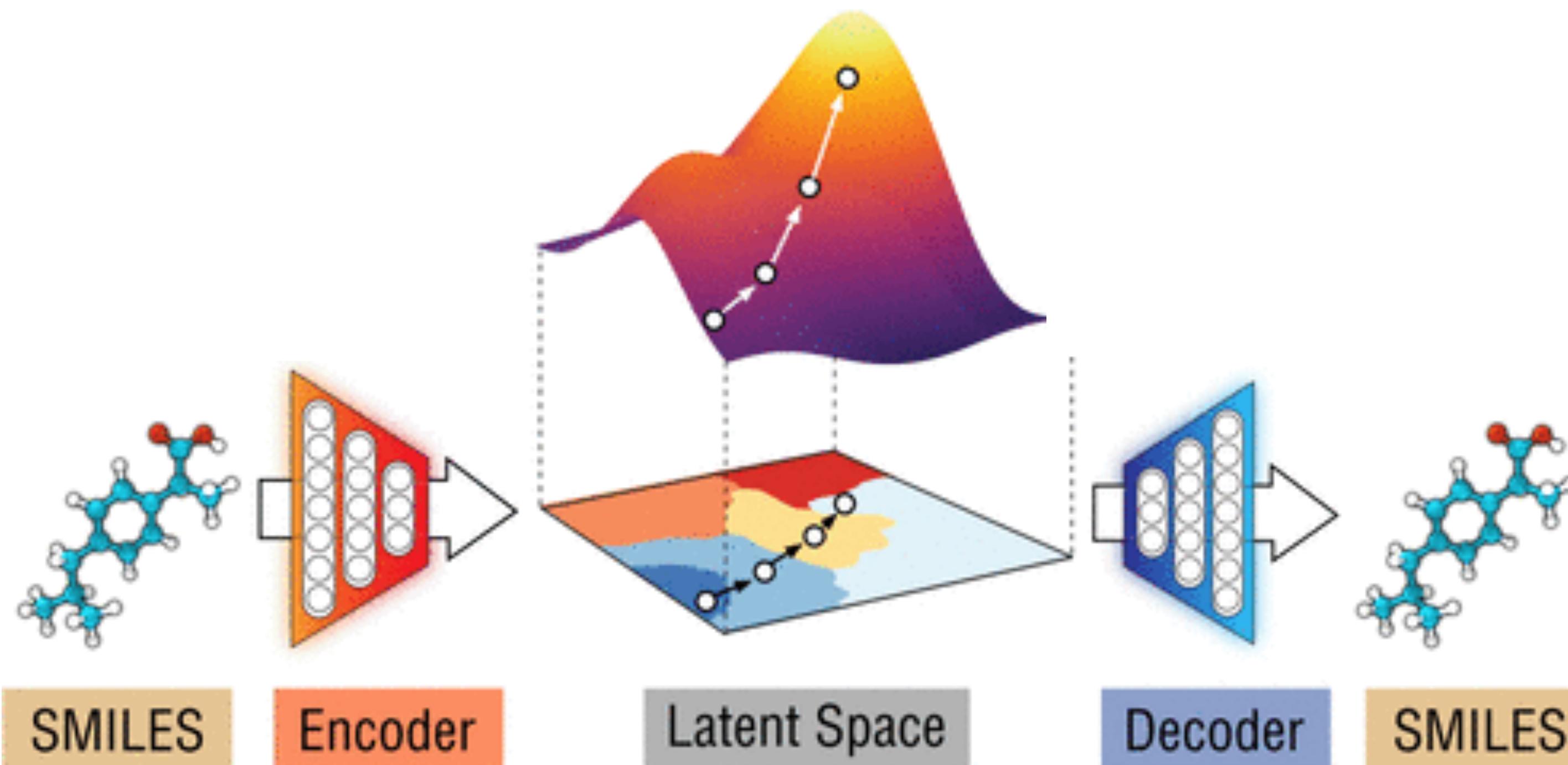
Gatys et al, 1508.06576



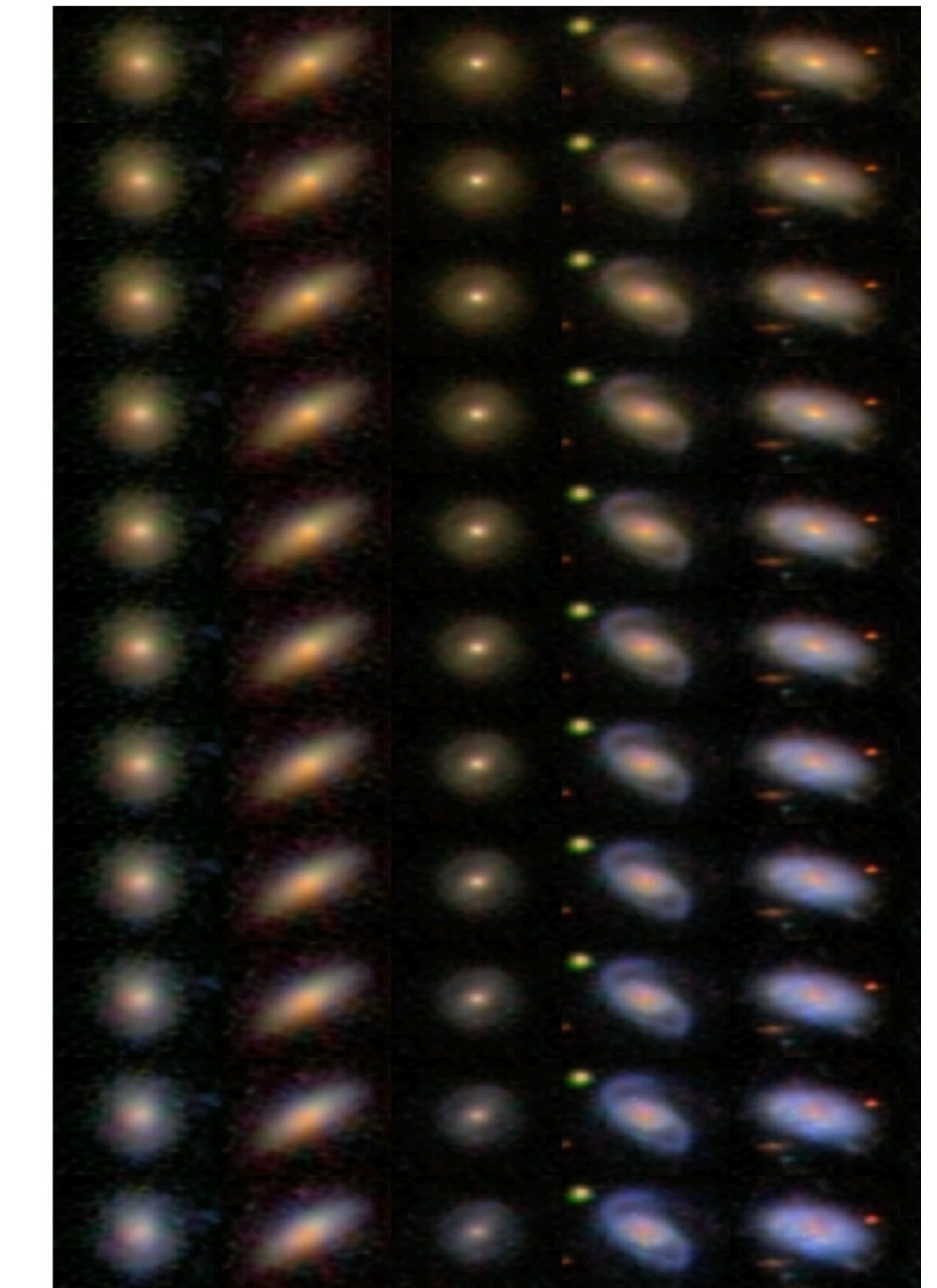
Glow 1807.03039

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

Learning representation for science

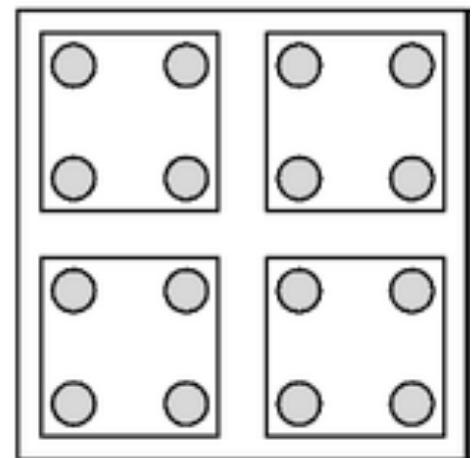


Automatic chemical design,
Gomez-Bombarelli et al, 1610.02415



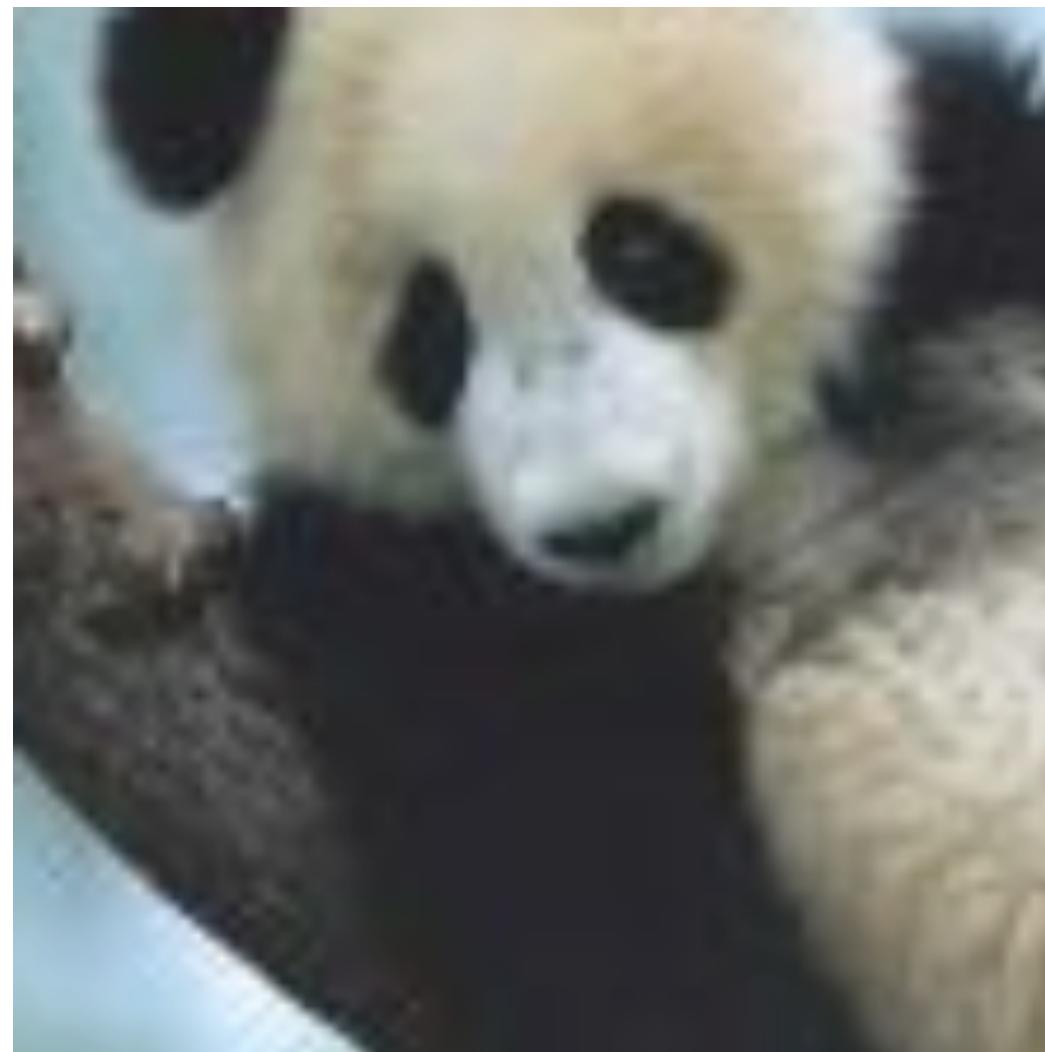
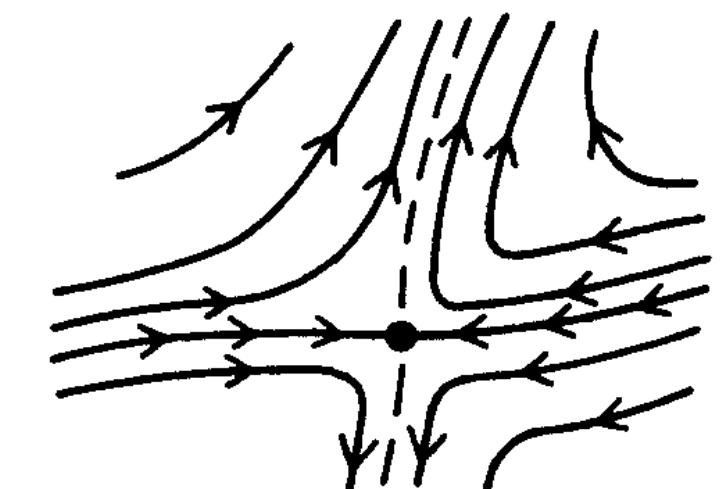
Galaxy evolution
Schawinski et al, 1812.01114

Deep Learning and Renormalization Group



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

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



+ .007 ×



=



熊猫

置信度58%

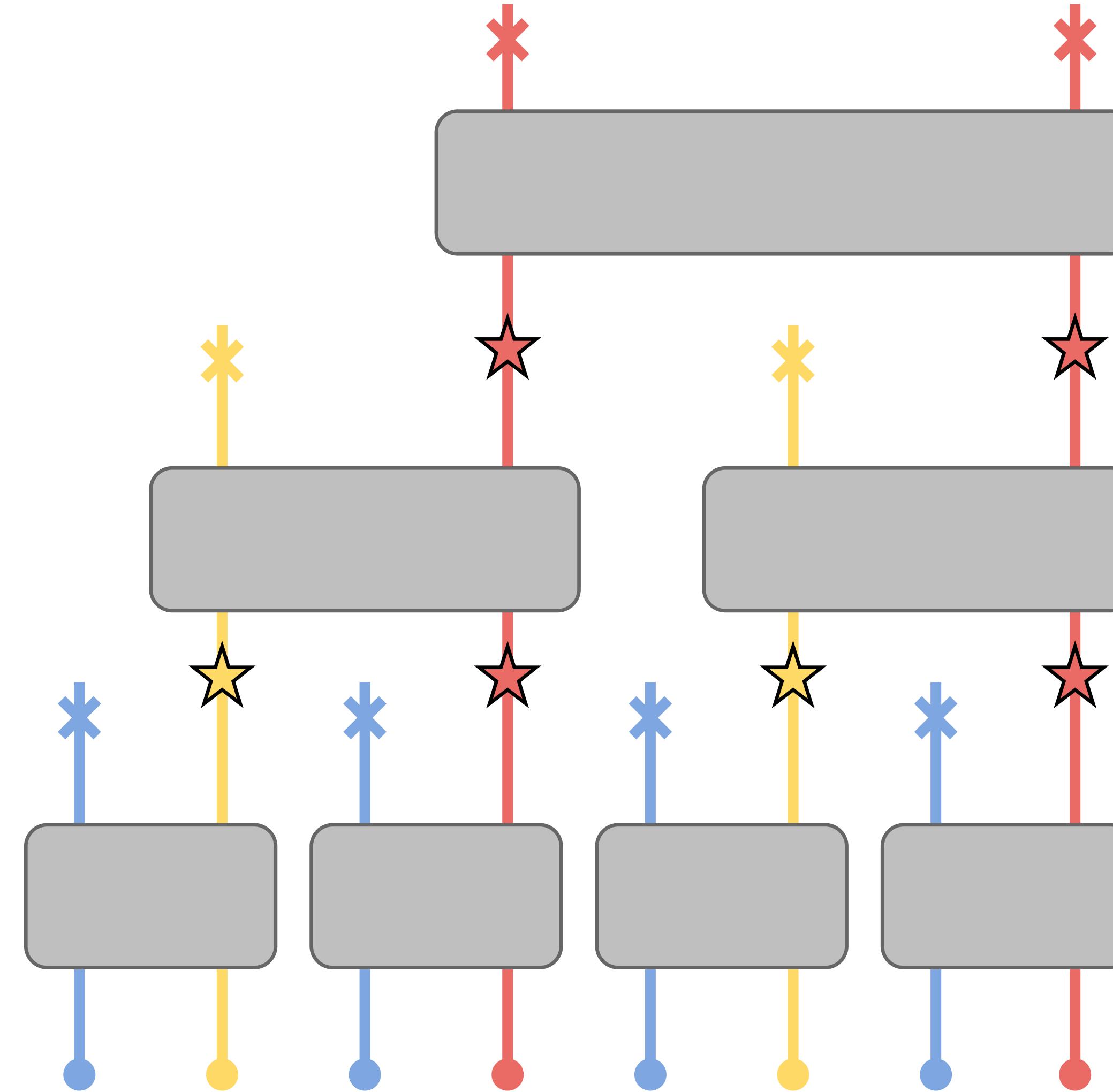
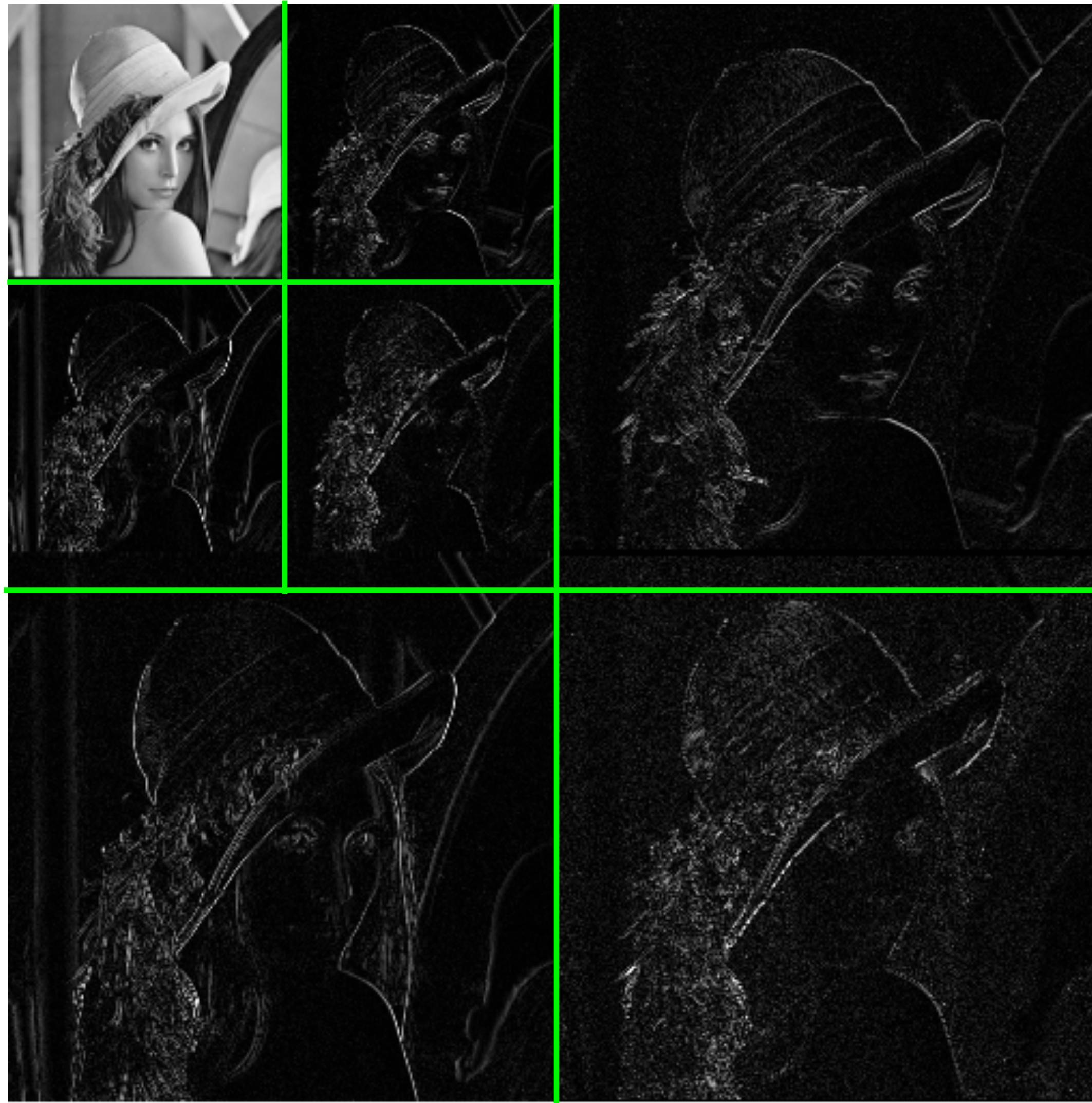
Goodfellow et al, 2014

长臂猿

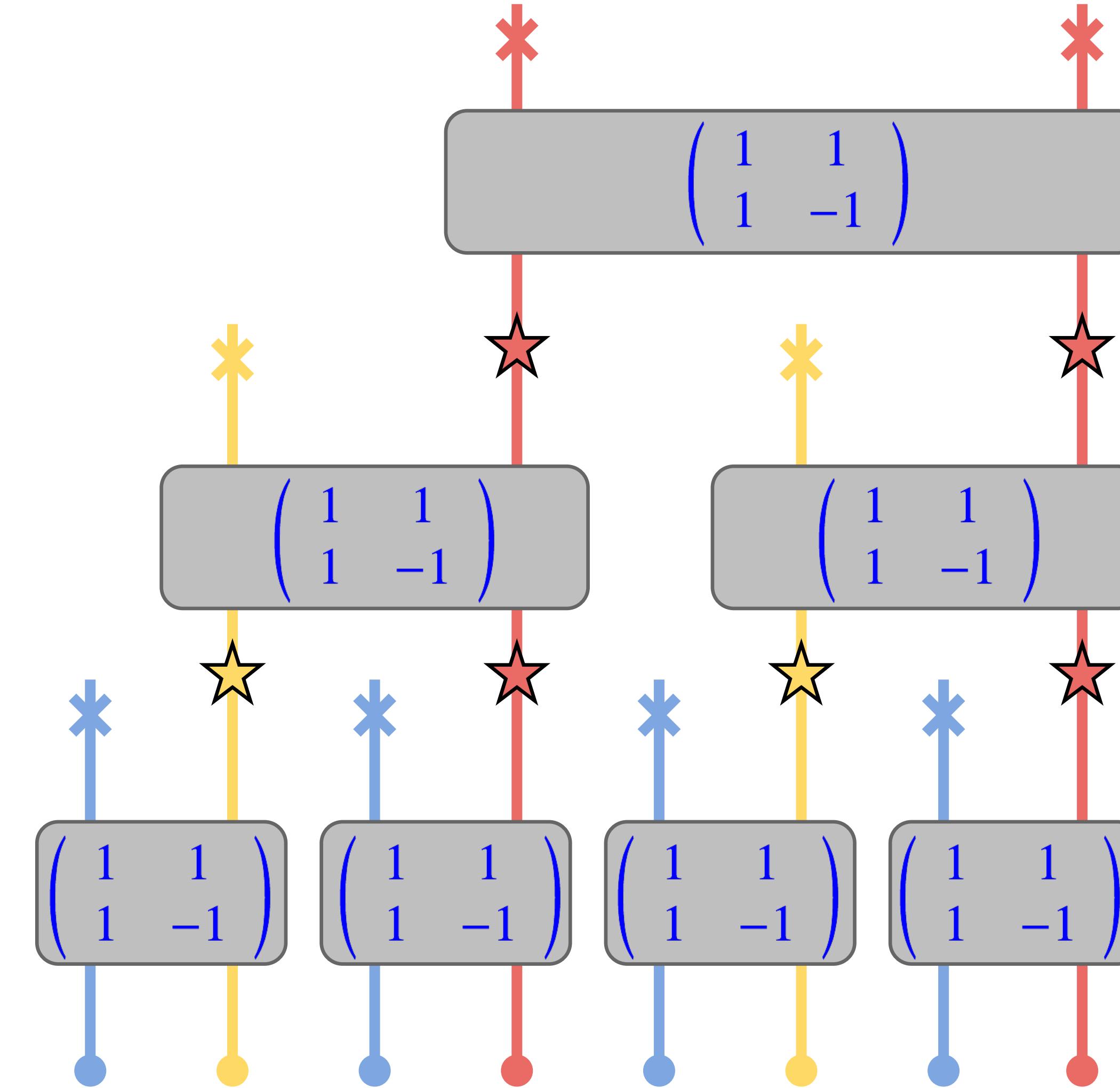
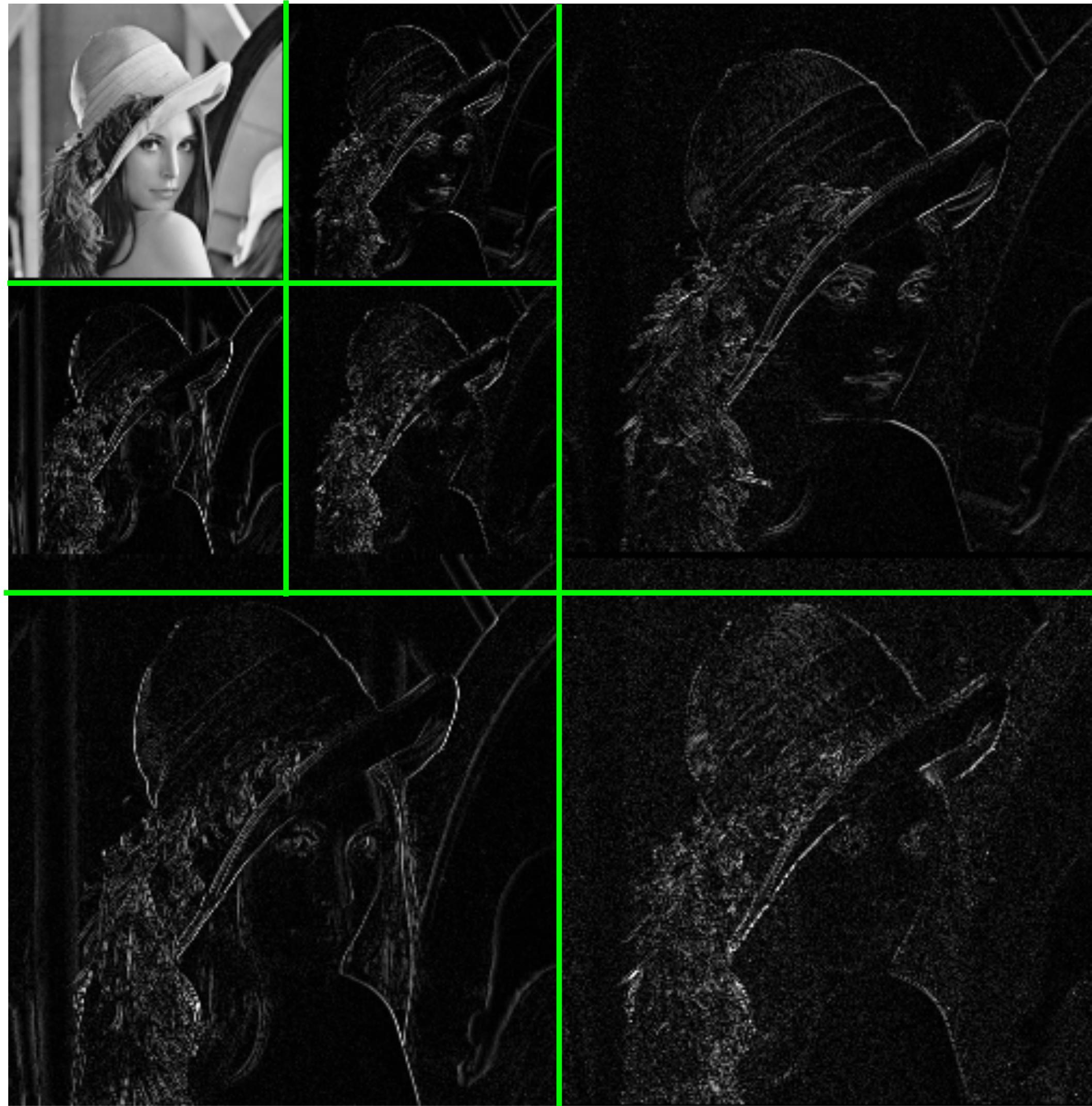
置信度99%

Vulnerability of deep learning, Kenway, 1803.06111 & 1803.10995

Wavelet transformation for Lena and Ising

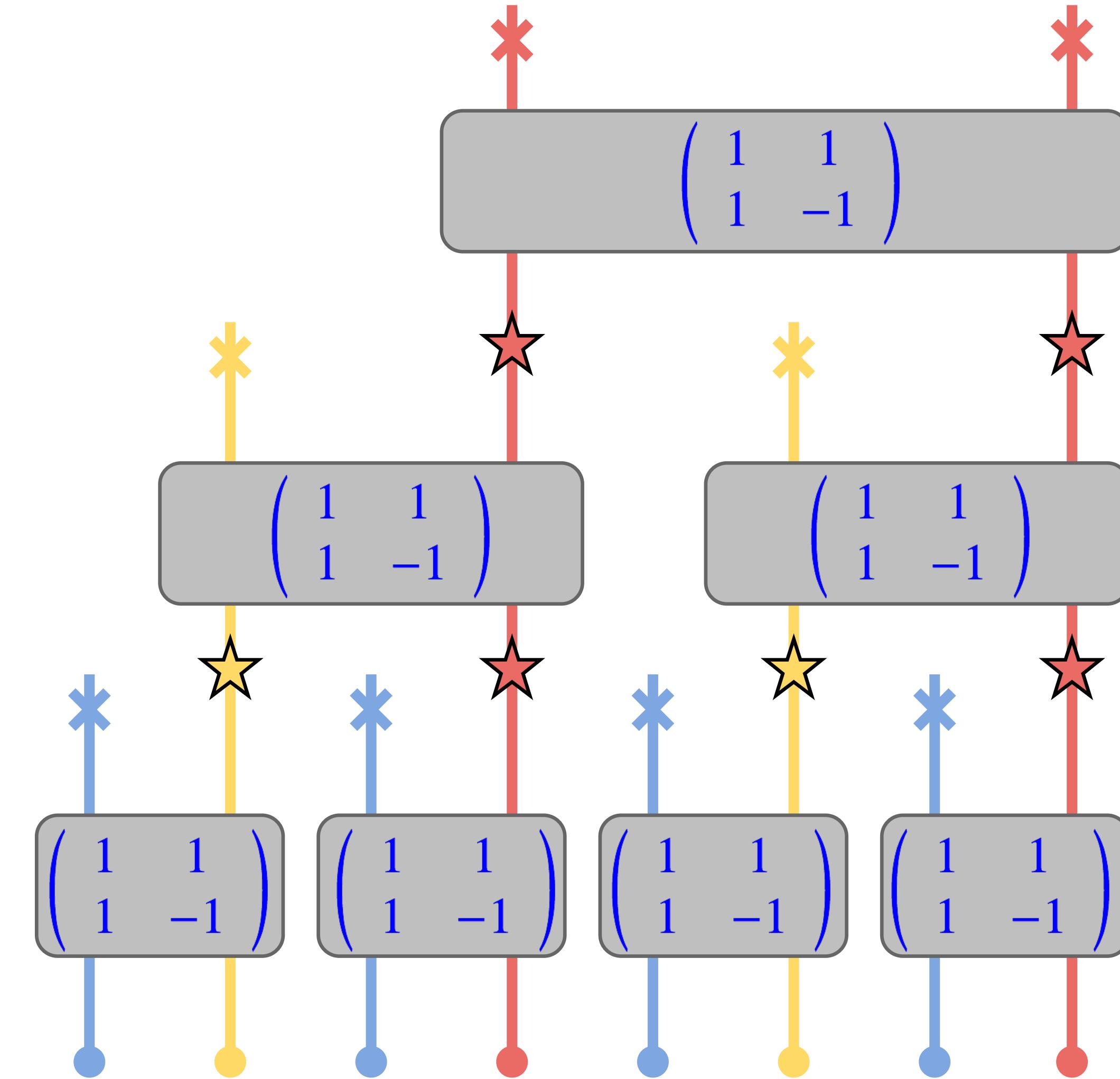
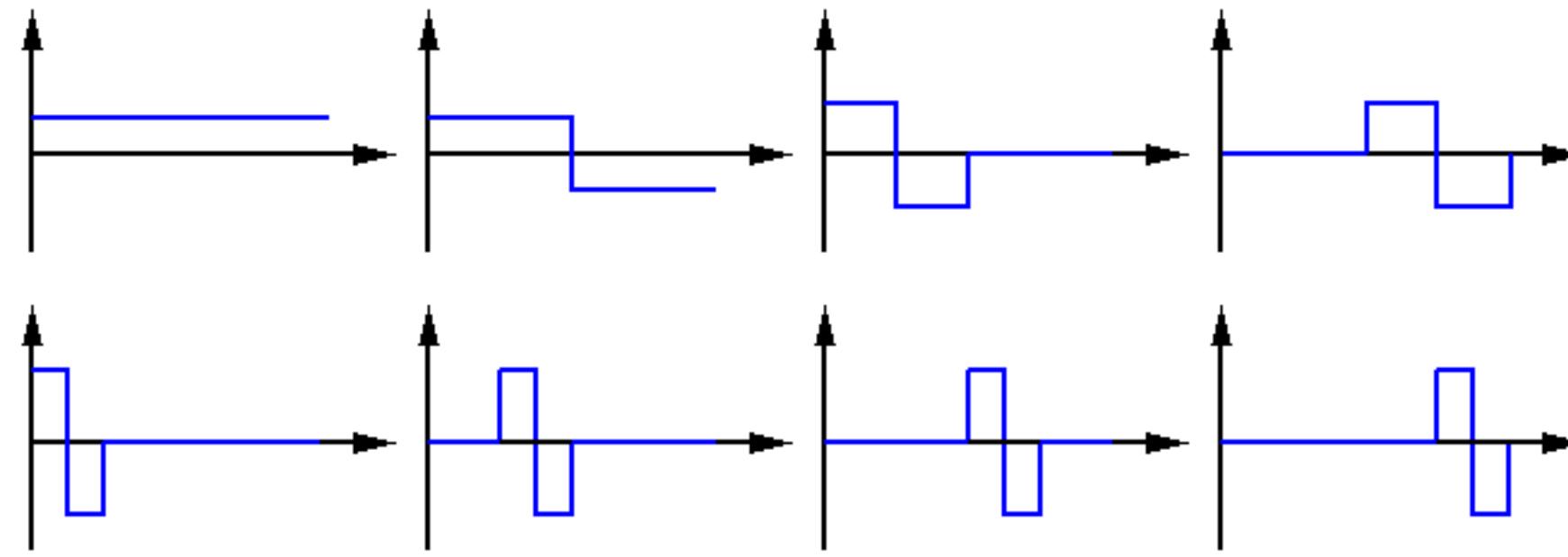


Wavelet transformation for Lena and Ising



Wavelet transformation for Lena and Ising

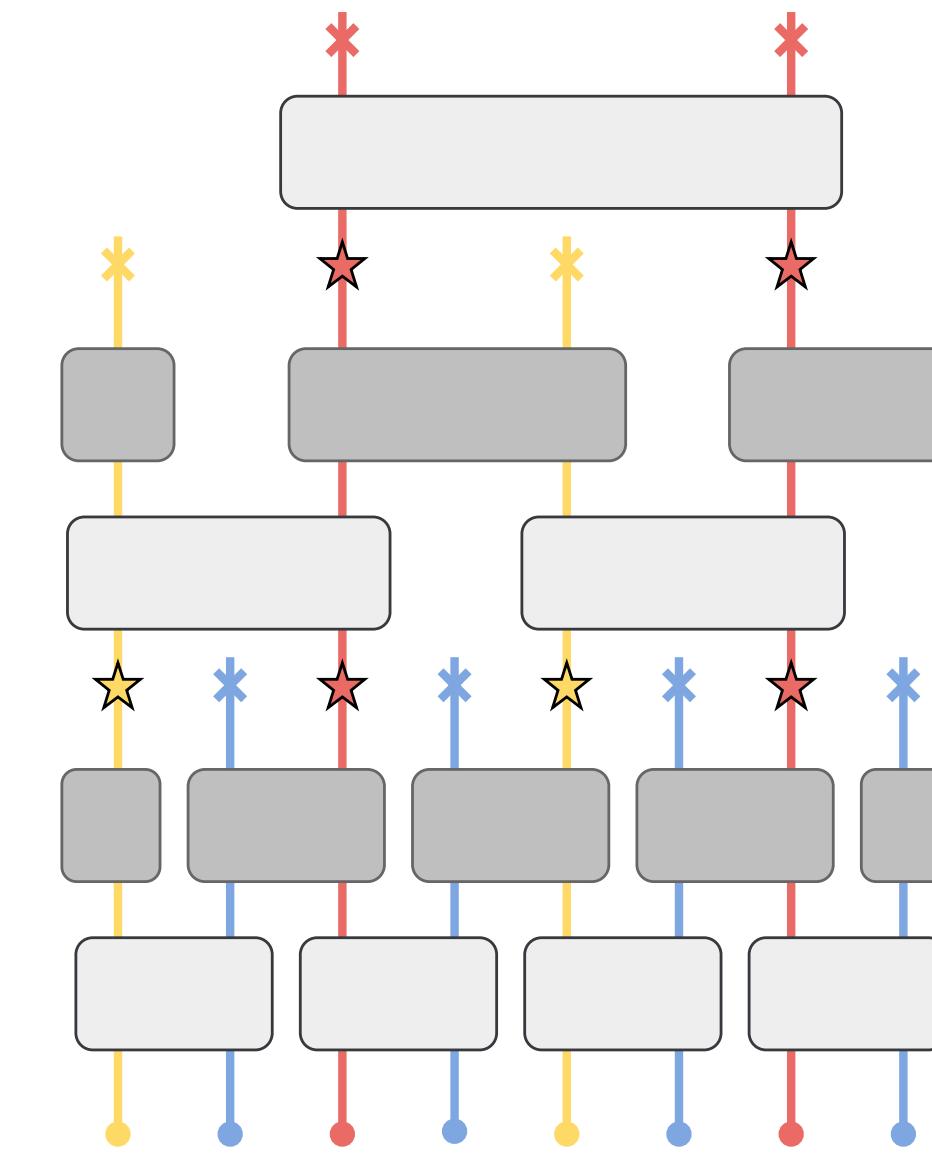
$$H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$



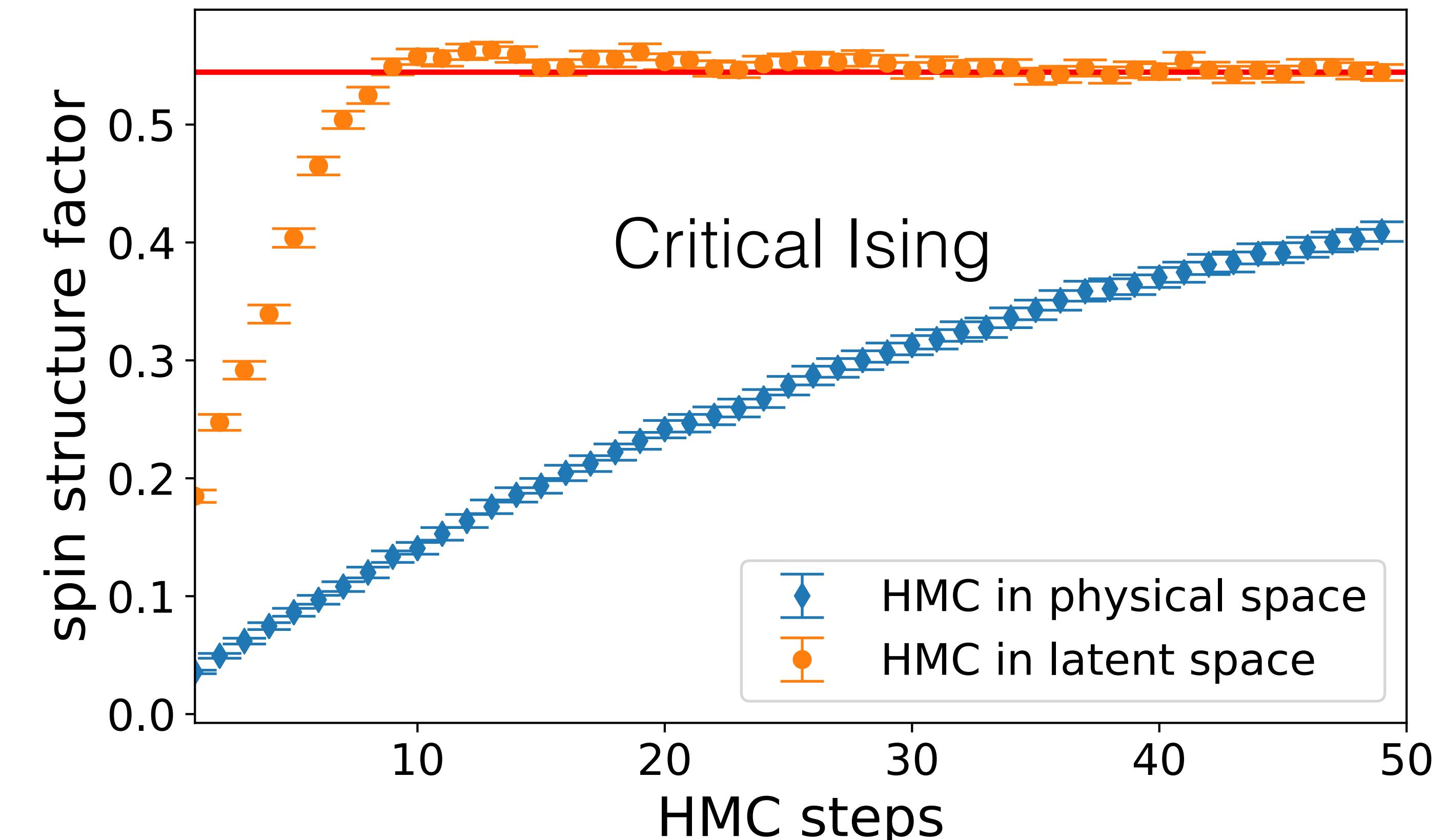
Latent space Hybrid MC

Latent space energy function

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



Physical energy function $E(x)$



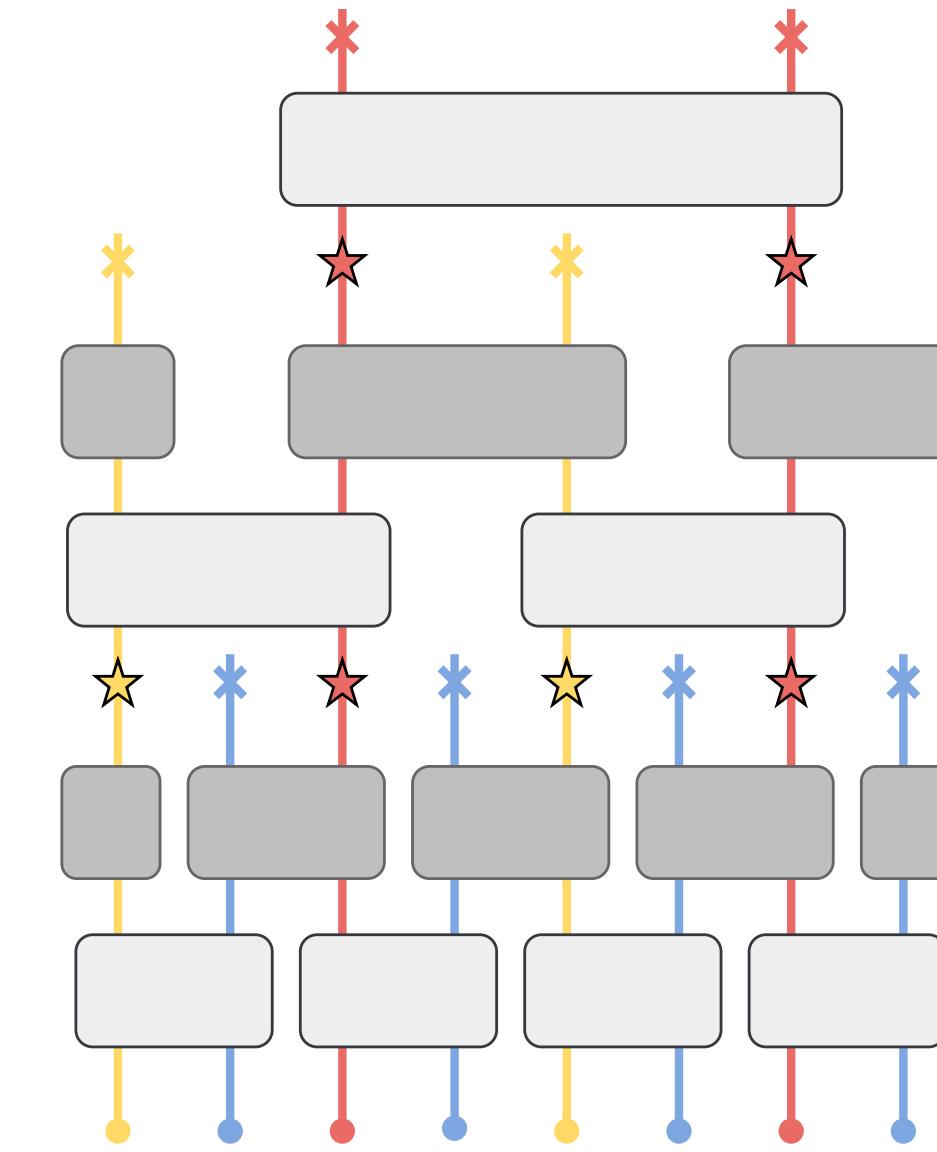
<https://github.com/li012589/NeuralRG>

HMC thermalizes faster in the latent space

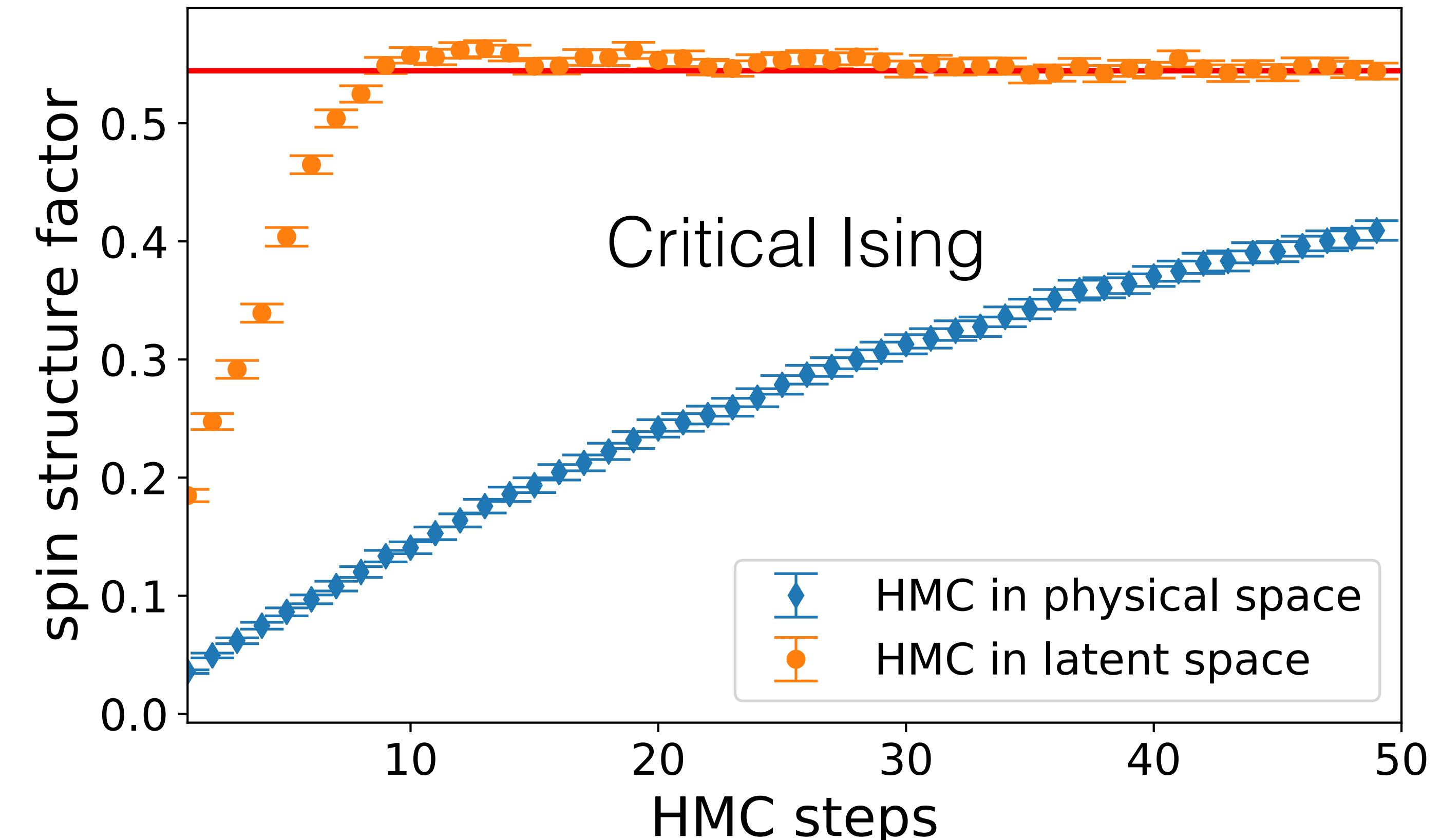
Latent space Hybrid MC

Latent space energy function

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



Physical energy function $E(x)$

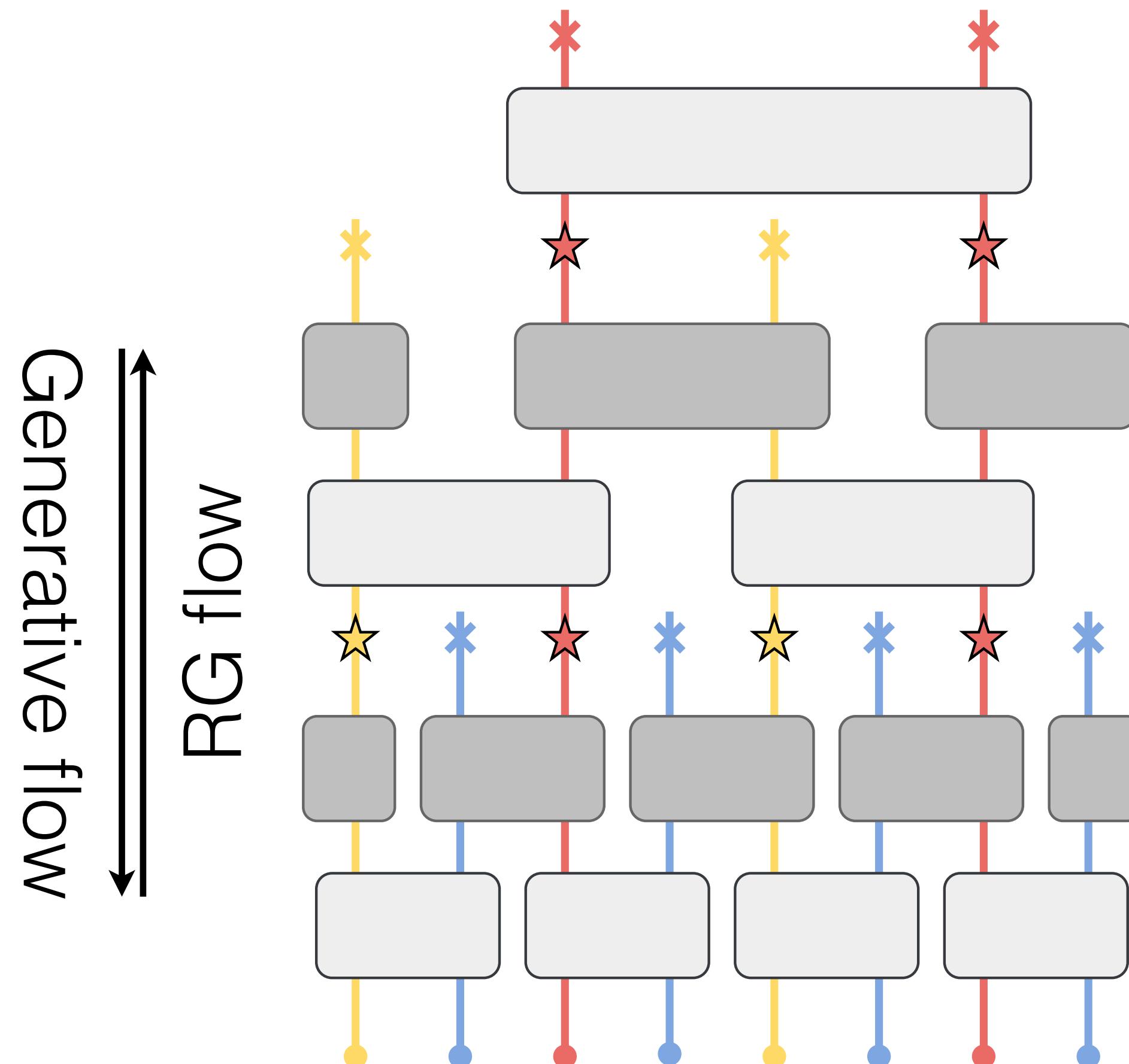


<https://github.com/li012589/NeuralRG>

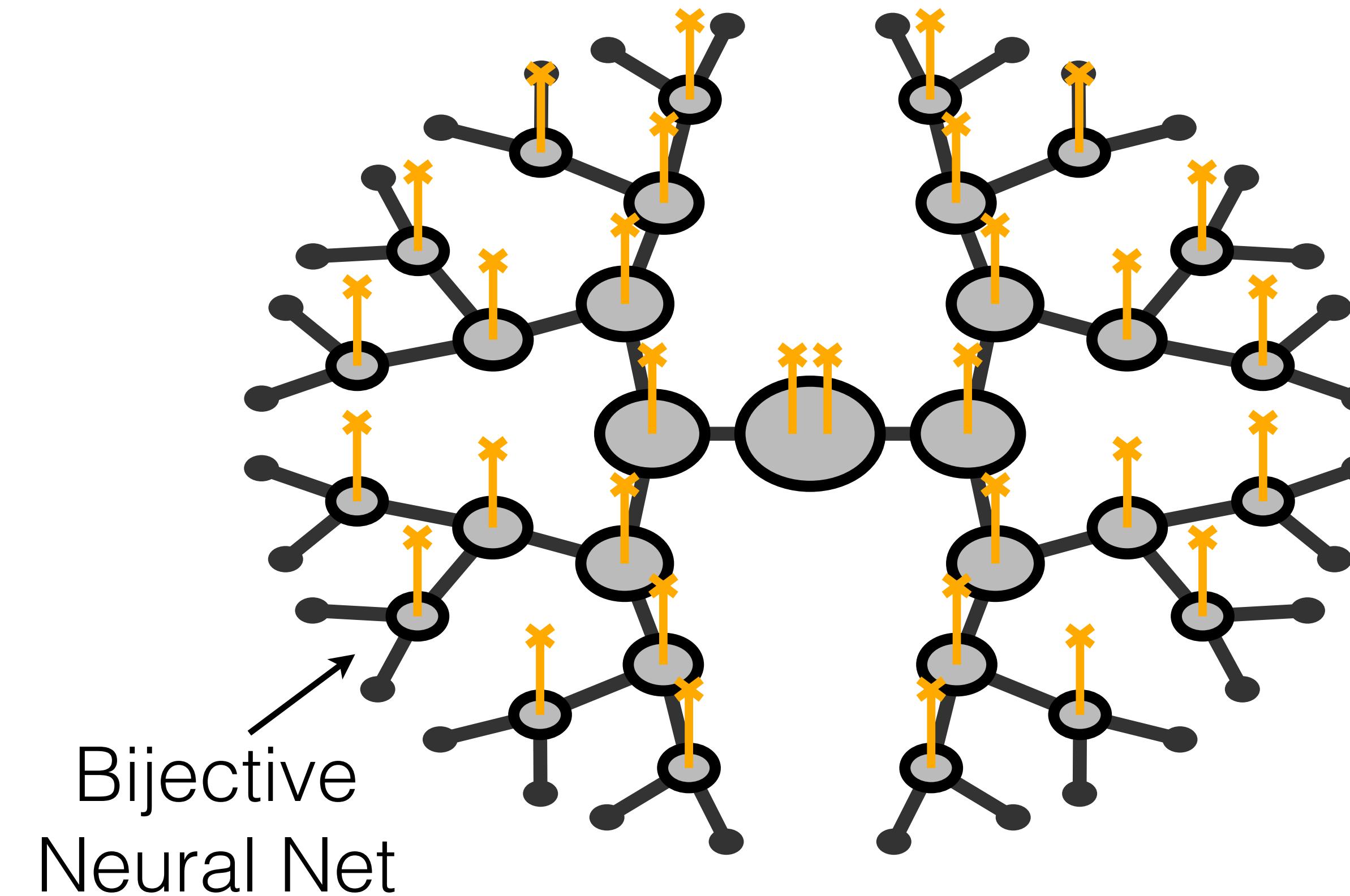
HMC thermalizes faster in the latent space

Neural Renormalization Group Flow

Normalizing flow with multiscale network structures



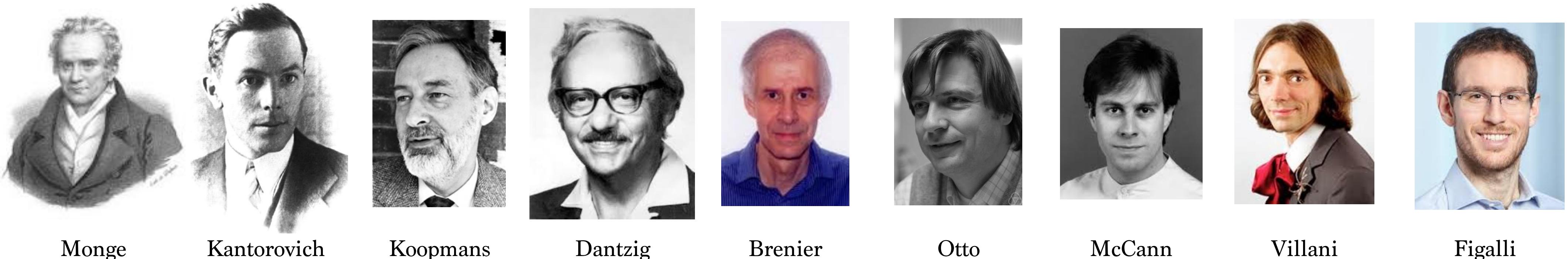
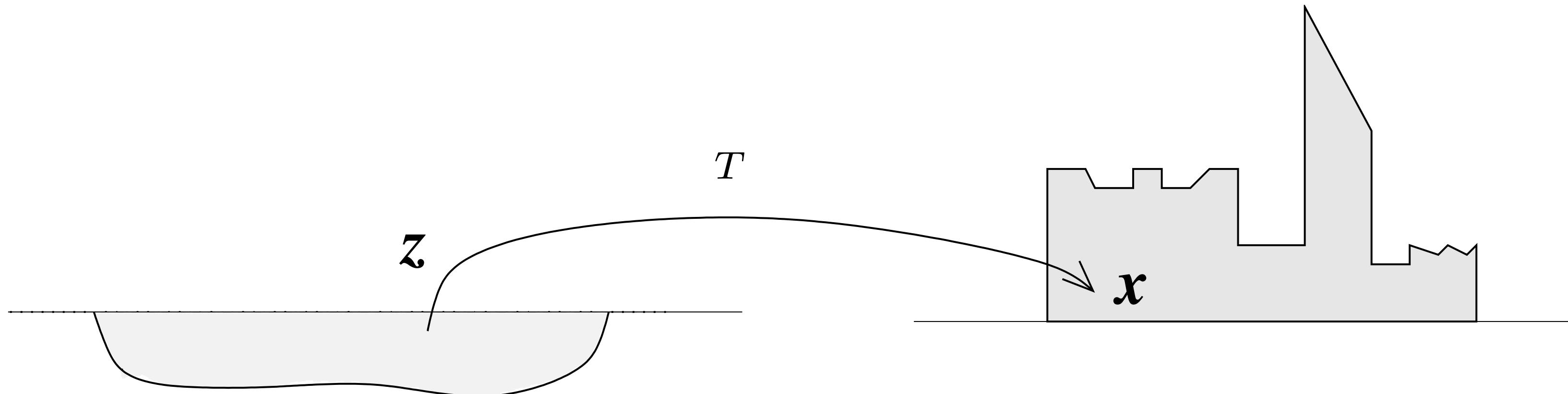
Swingle 0905.1317, Qi 1309.6282 and more



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 ?



Nobel Prize in Economics '75

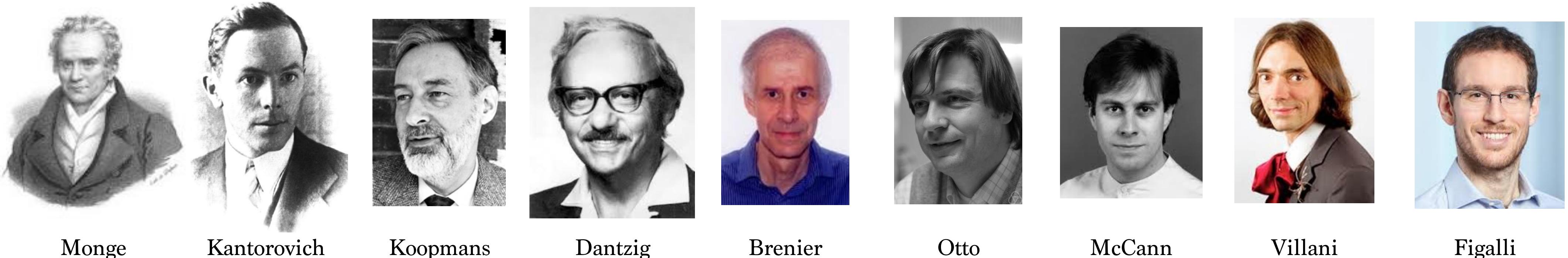
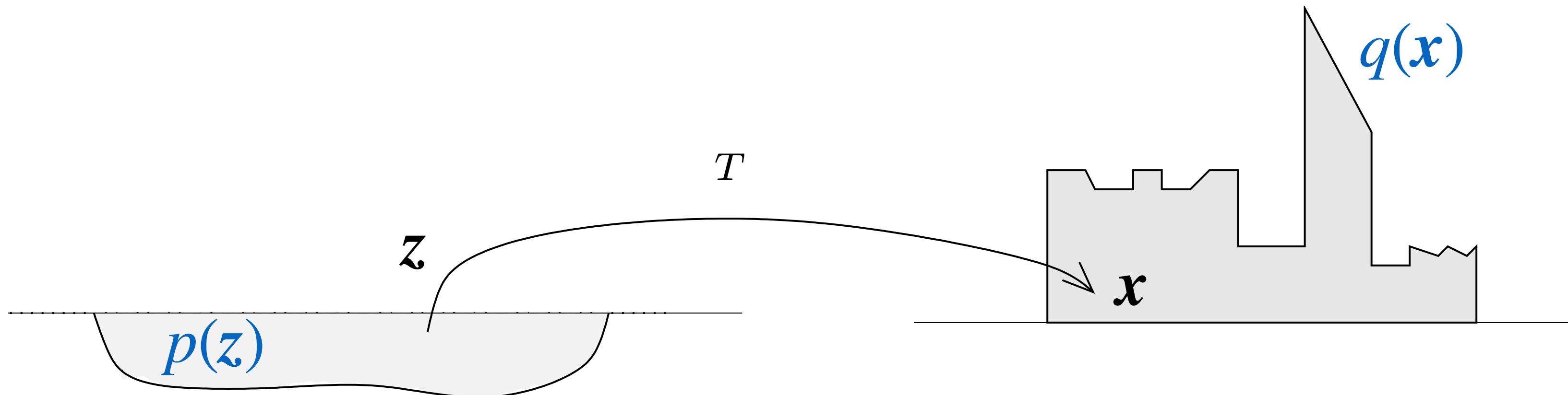
Fields Medal '10

Fields Medal '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

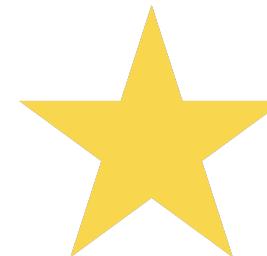
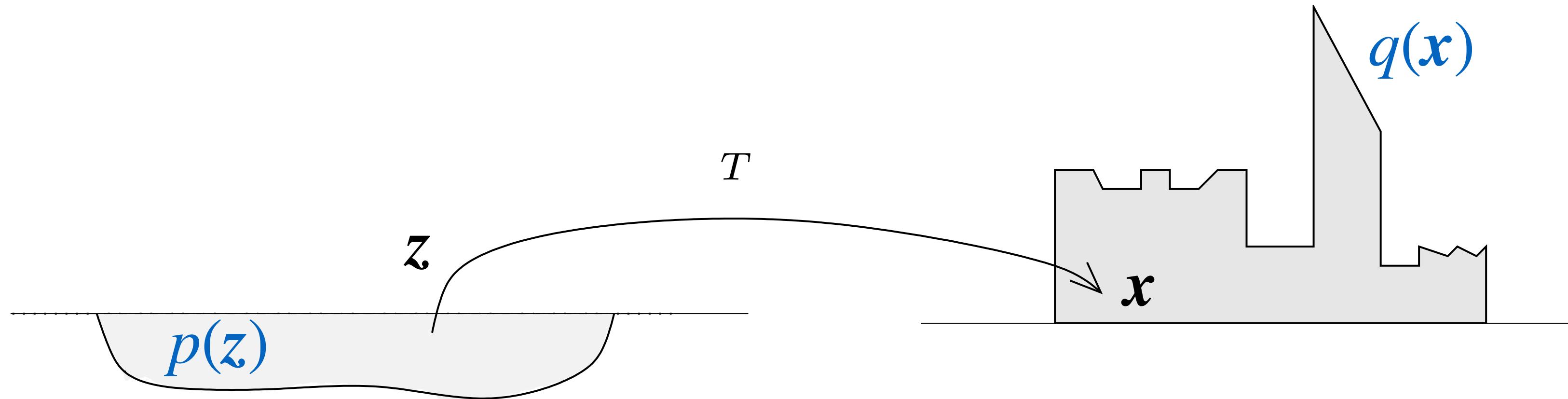
Fields Medal '10

Fields Medal '18

from Cuturi, Solomon NISP 2017 tutorial

Optimal Transport Theory

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



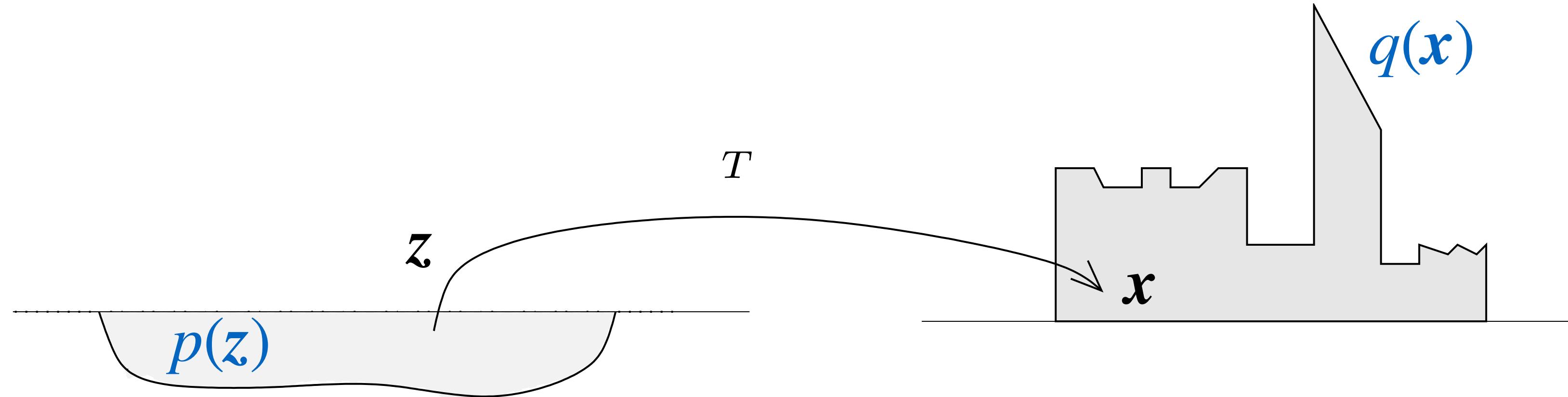
Brenier theorem (1991)

Under reasonable conditions
the optimal map is

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

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](#)

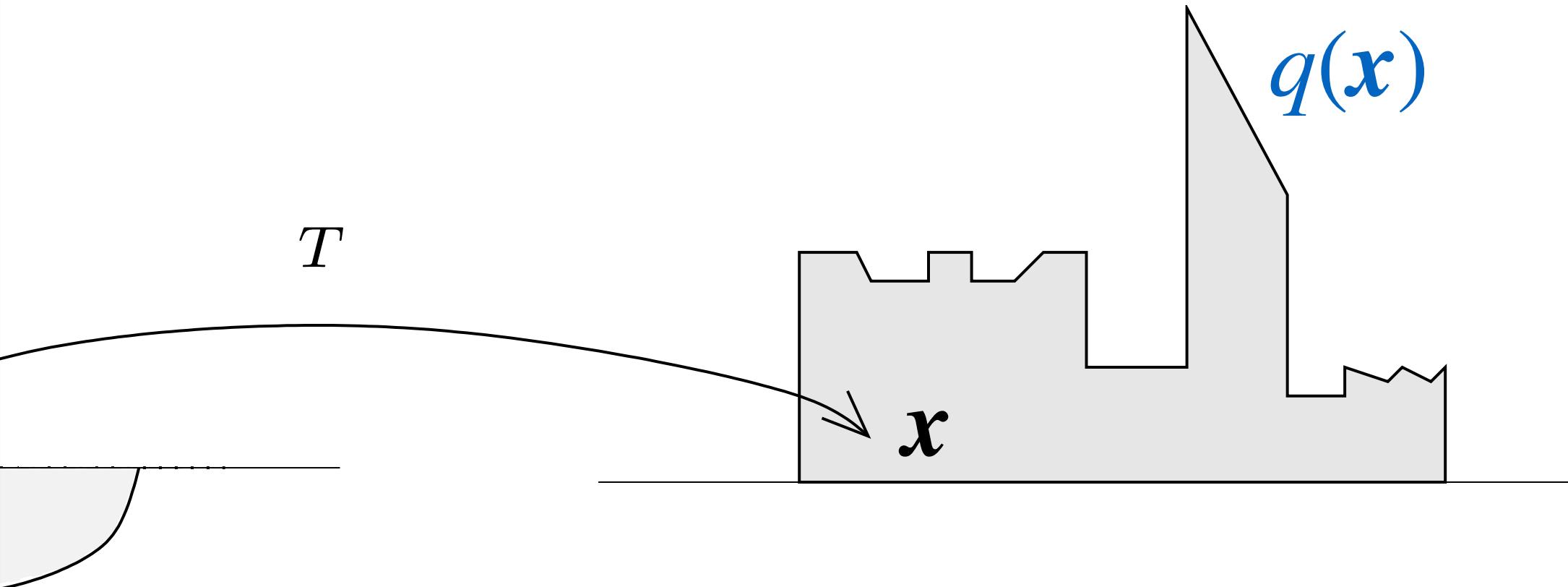


Brenier theorem (1991)

Monge-Ampère Equation

Transport Theory

How to transport earth with optimal cost ?



Under reasonable conditions
the optimal map is

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

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



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



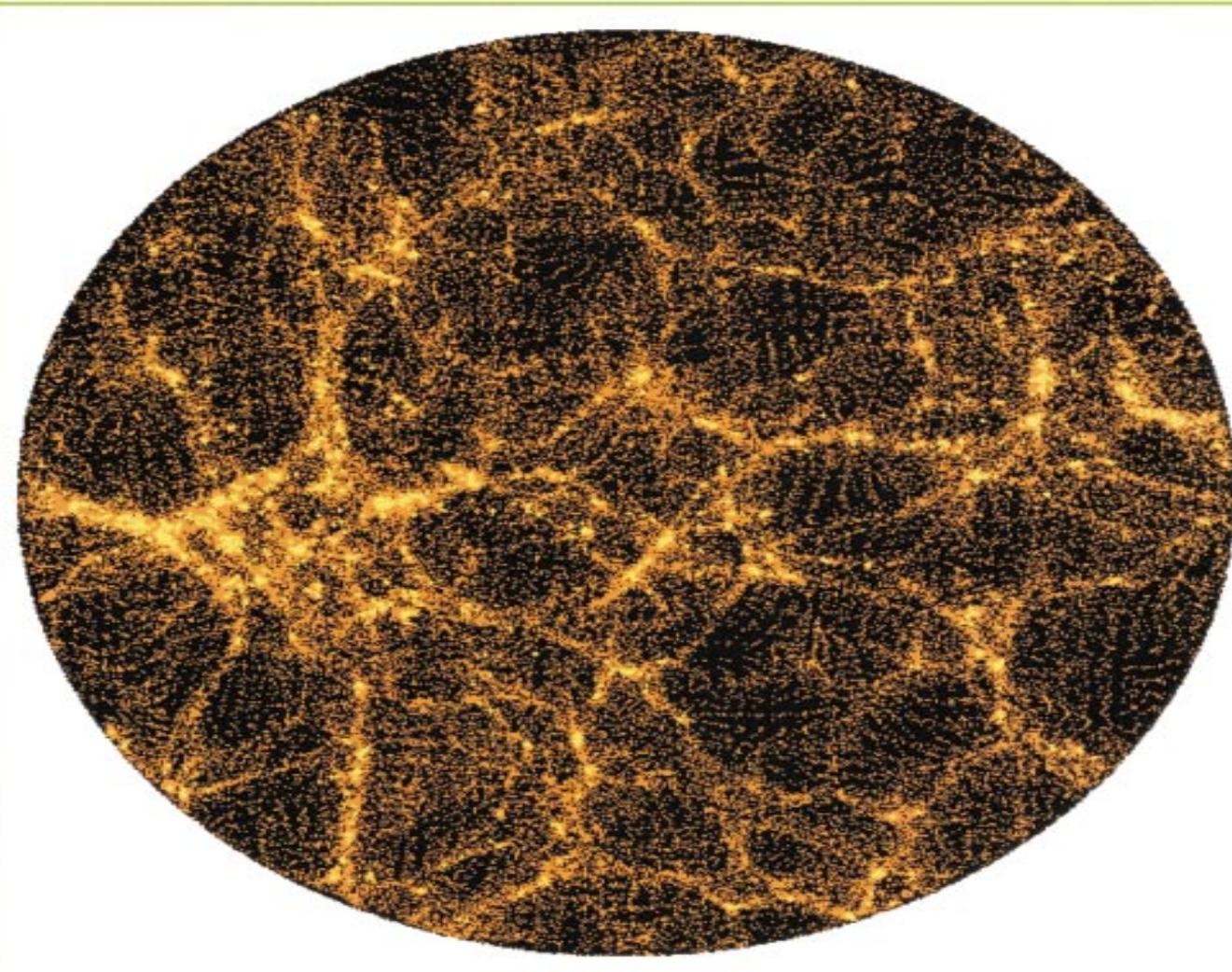
Brenier theorem (1991)

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§*



Under reasonable conditions
the optimal map is

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

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

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: $q = \nabla_x \Theta(x)$, 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 $\rho(x)$ the final one. Mass conservation implies $\rho_0 d^3 q = \rho(x) d^3 x$. 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 equation²⁰ for the unknown potential Θ :

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

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 problem²¹ (see also refs 22 and 23).

A geometric perspective to deep learning

19:33 4G

X 老顾谈几何 ...

最优传输映射 自映射 $T : \Omega \rightarrow \Omega$ 的传输代价
(Transportation Cost) 定义为

$$E(T) := \frac{1}{2} \int_{\Omega} |x - T(x)|^2 d\mu(x).$$

在所有保持测度的自映射中, 传输代价最小者被称为是最优传输映射 (Optimal Mass Transportation Map), 亦即:

$$T^* = \operatorname{argmin}_{T^* \mu=\nu} E(T),$$

最优传输映射的传输代价被称为是概率测度 μ 和概率测度 ν 之间的 Wasserstein 距离, 记为 $d_W(\mu, \nu)$ 。

在这种情形下, Brenier 证明存在一个凸函数 $u : \Omega \rightarrow \mathbb{R}$, 其梯度映射

$$T : x \mapsto \nabla u(x)$$

就是唯一的最优传输映射。这个凸函数被称为是 Brenier 势能函数 (Brenier potential)。

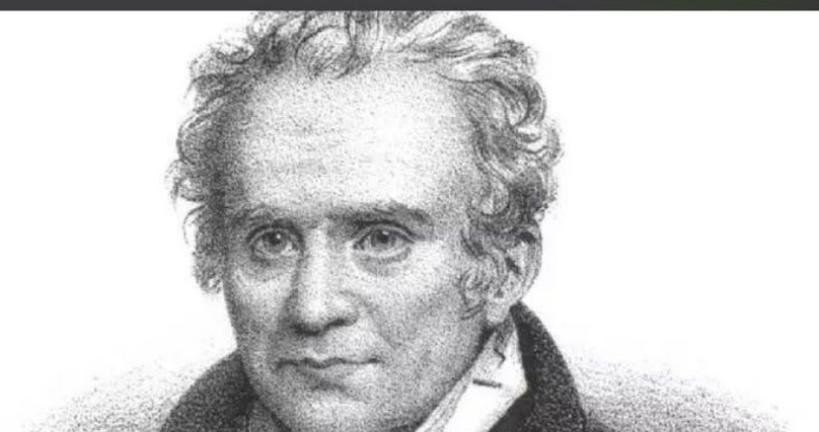
由 Jacobian 方程, 我们得到 Brenier 势满足蒙日-安培方程, 梯度映射的雅克比矩阵是 Brenier 势能函数的海森矩阵 (Hessian Matrix),

$$\det \left(\frac{\partial^2 u}{\partial x_i \partial x_j} \right) (\mathbf{x}) = \frac{\mu(\mathbf{x})}{\nu \circ \nabla(\mathbf{x})}.$$

蒙日-安培方程解的存在性、唯一性等价于经典的凸几何中的亚历山大定理 (Alexandrov Theorem)。

19:30 4G

< WeChat (2) 老顾谈几何



菲尔兹奖青睐的领域: 最优传输和蒙日-安培方程

最优传输理论是数学历史上美学价值和实用价值相结合的一个典范! 它在工程和医疗领域得到广泛应用, 例如图形学中的参数化、视觉中的曲面注册、大数据中的几何分类、网络中的特征提取。特别是最优传输理论为深度学习的生成模型奠定了坚实的理论基础。

Oct 10, 2018 11:35



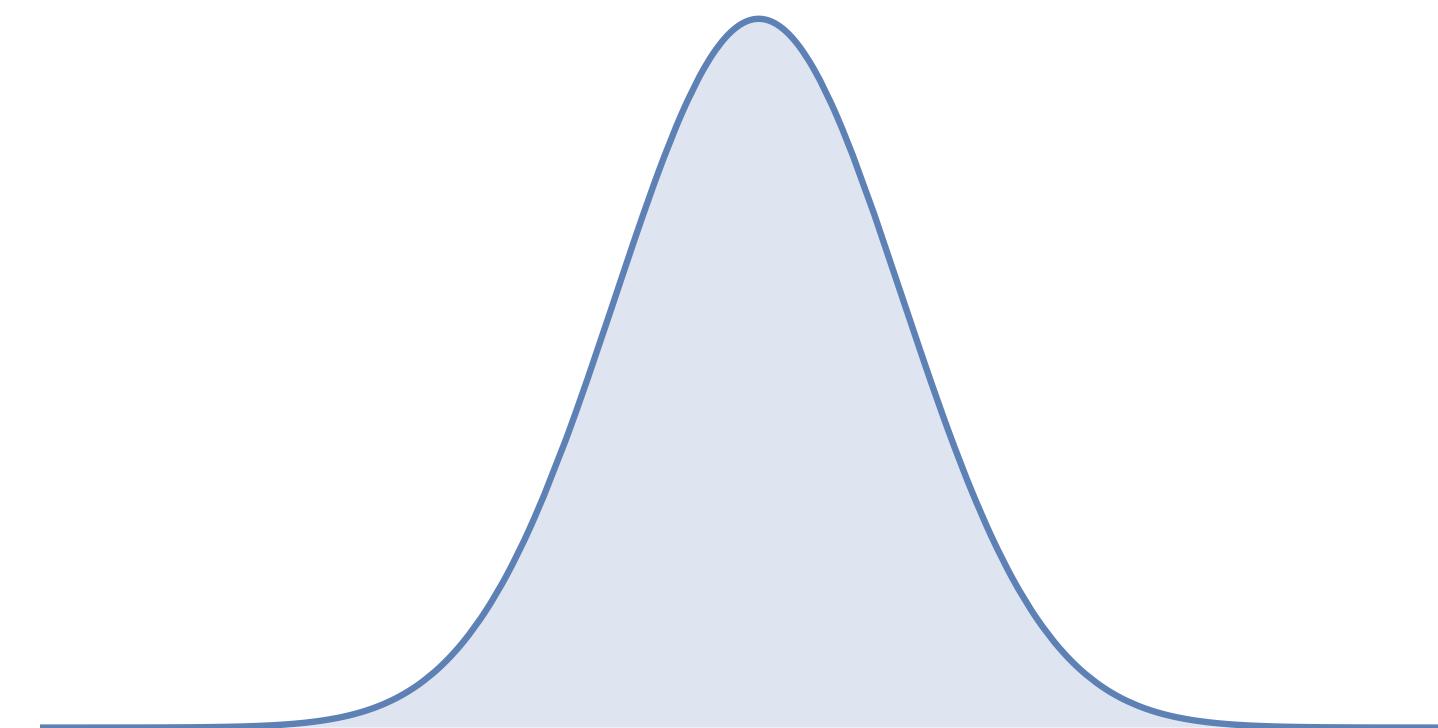
在宇宙中不易被风吹散

清华计算机系理论组的老一辈师长虽然已经退休, 但是他们传播的知识由弟子们传承, 他们对

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



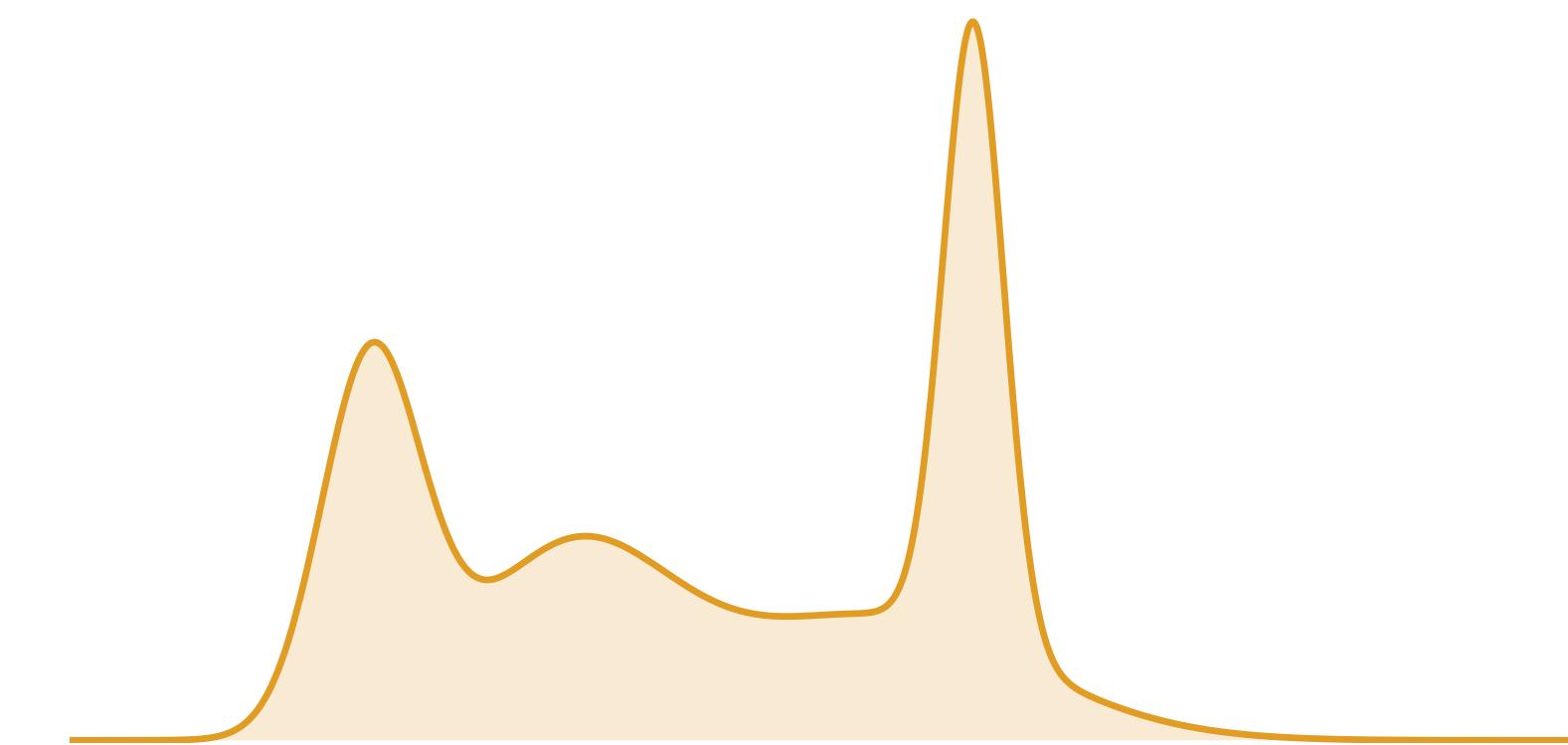
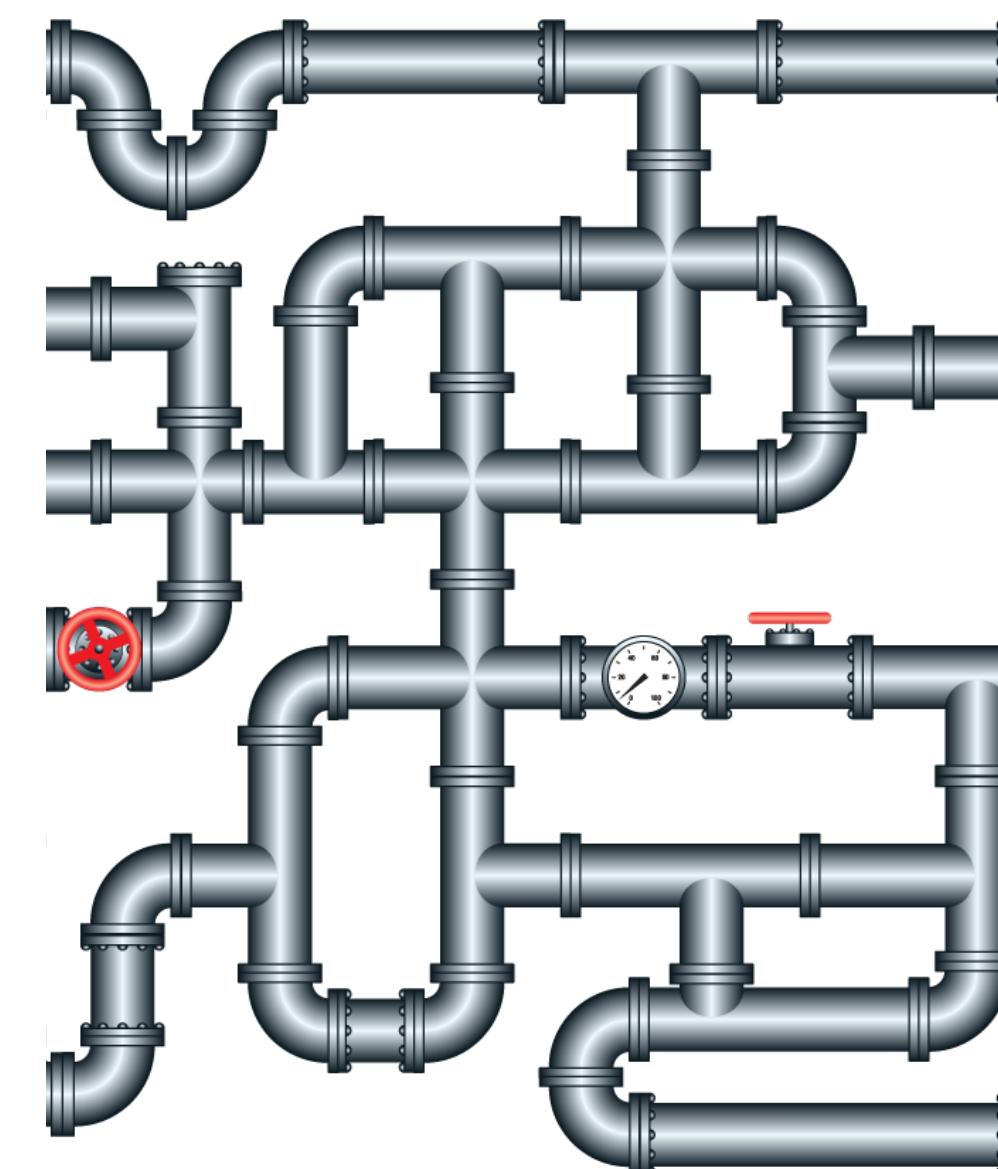
Simple density

Continuous-time limit

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

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

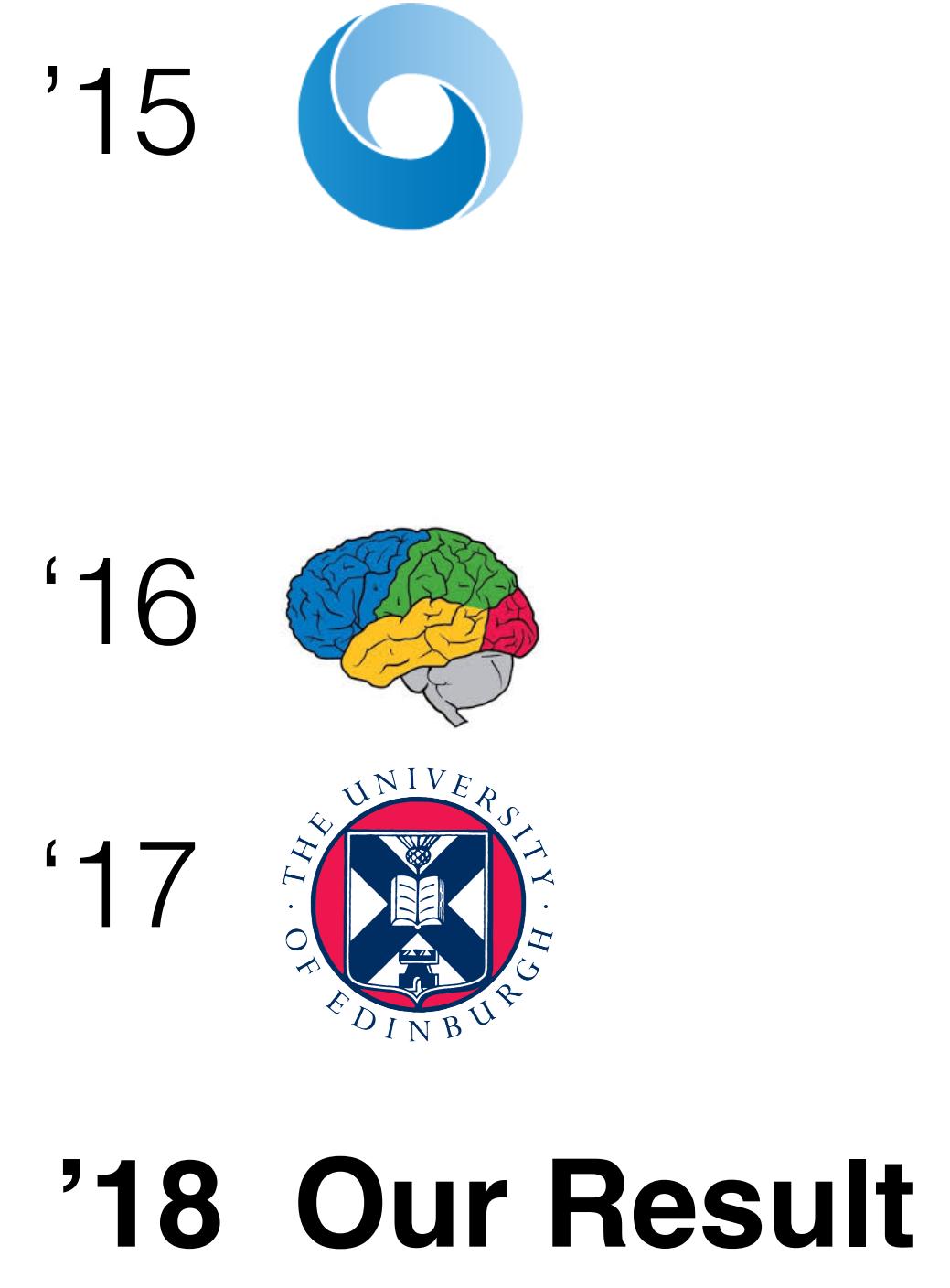
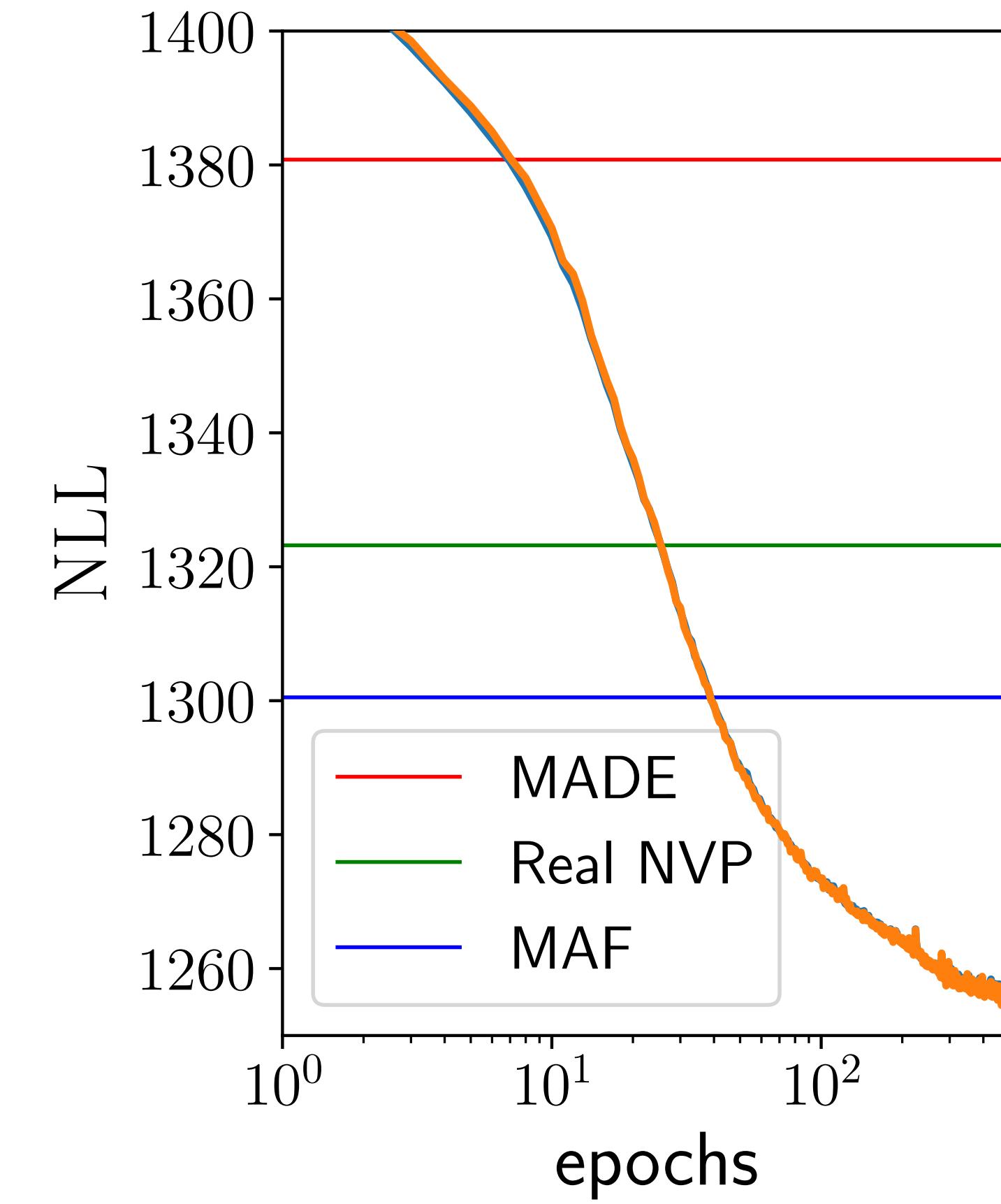
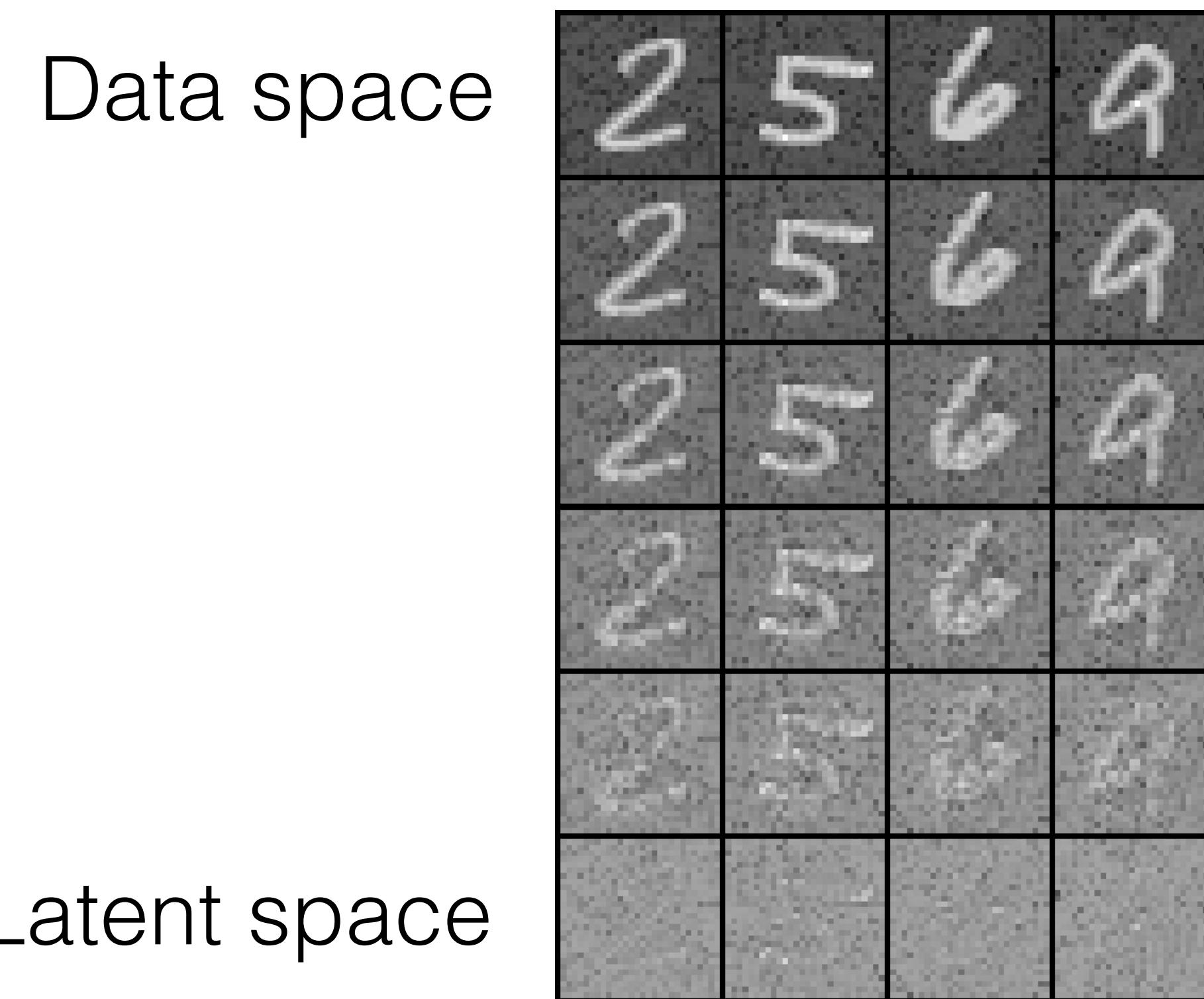
Continuity equation of
compressible fluids



Complex density

Density estimation of hand-written digits

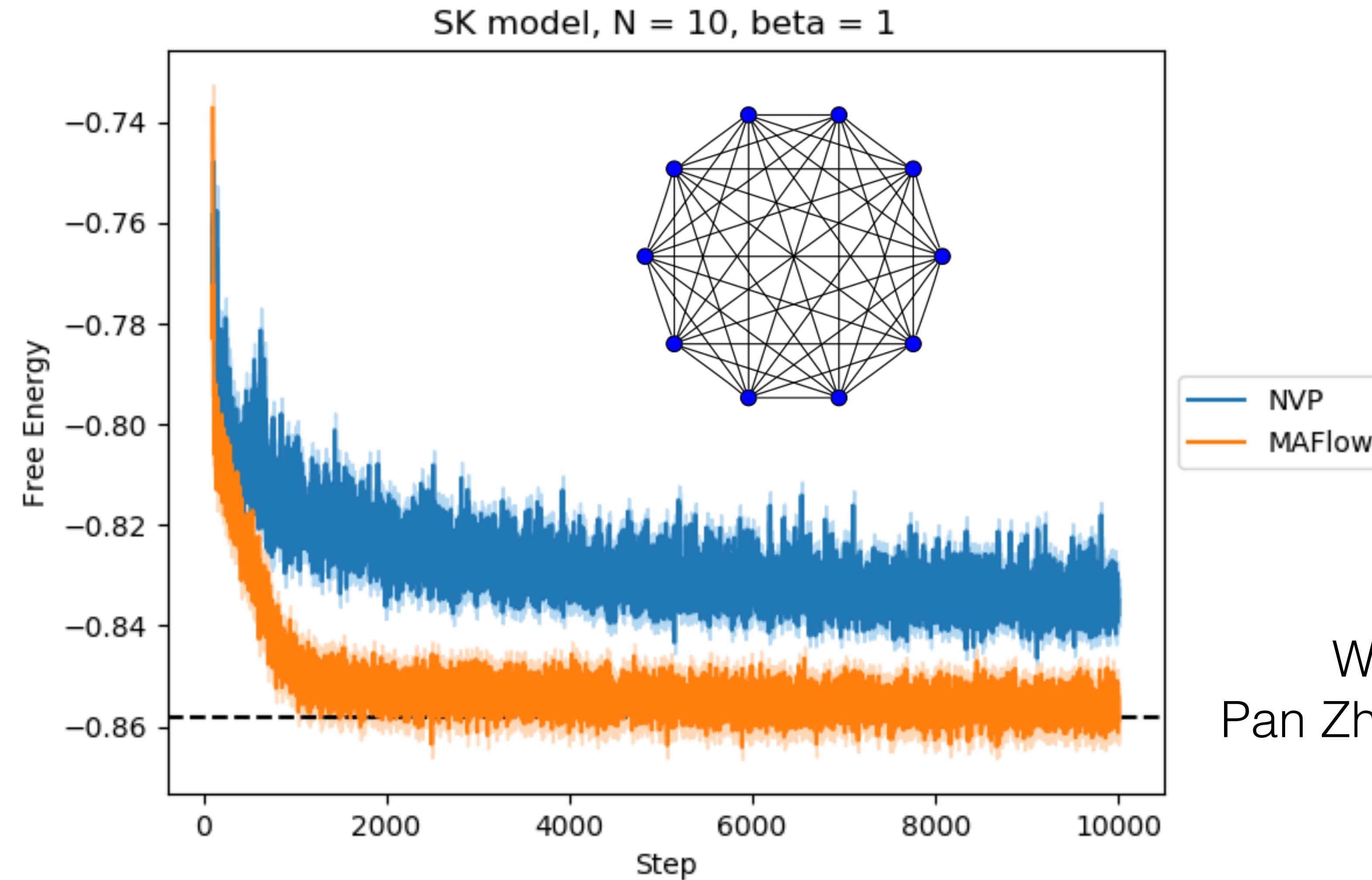
A standard benchmark for generative models, lower is better



1

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

Variational study of Sherrington-Kirkpatrick spin glasses

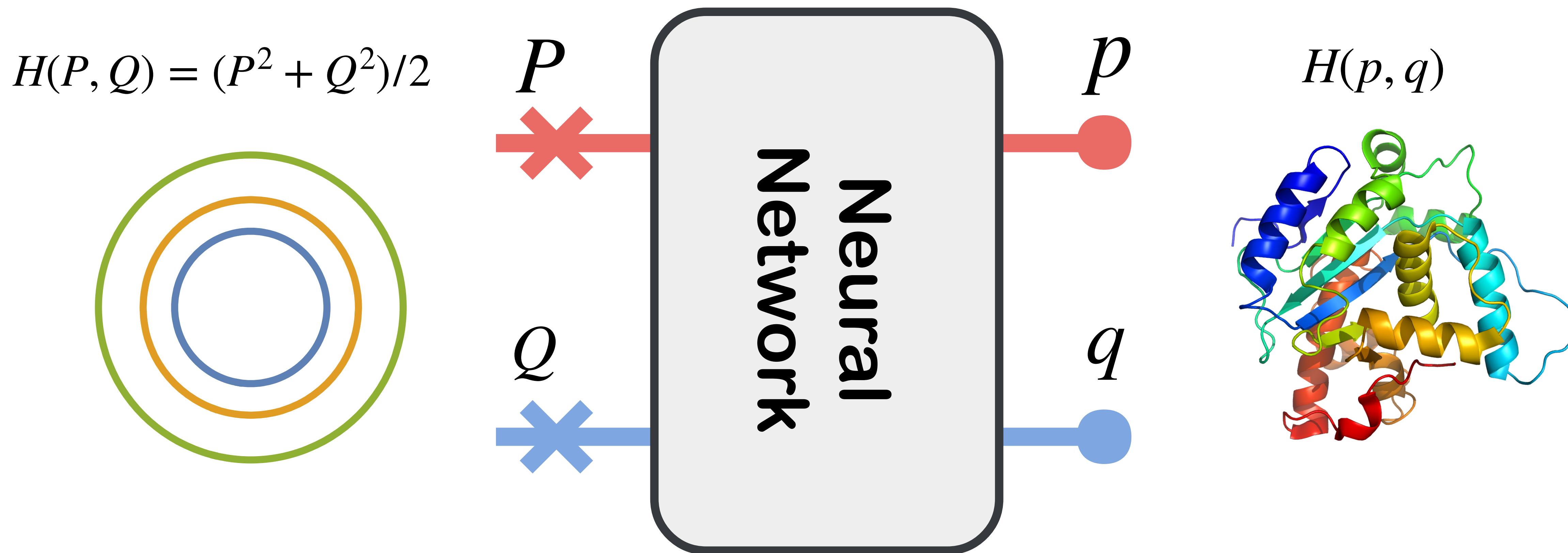


②

Better variational energy than previous network structure

Neural Canonical Transformations

Incompressible symplectic flow in phase space



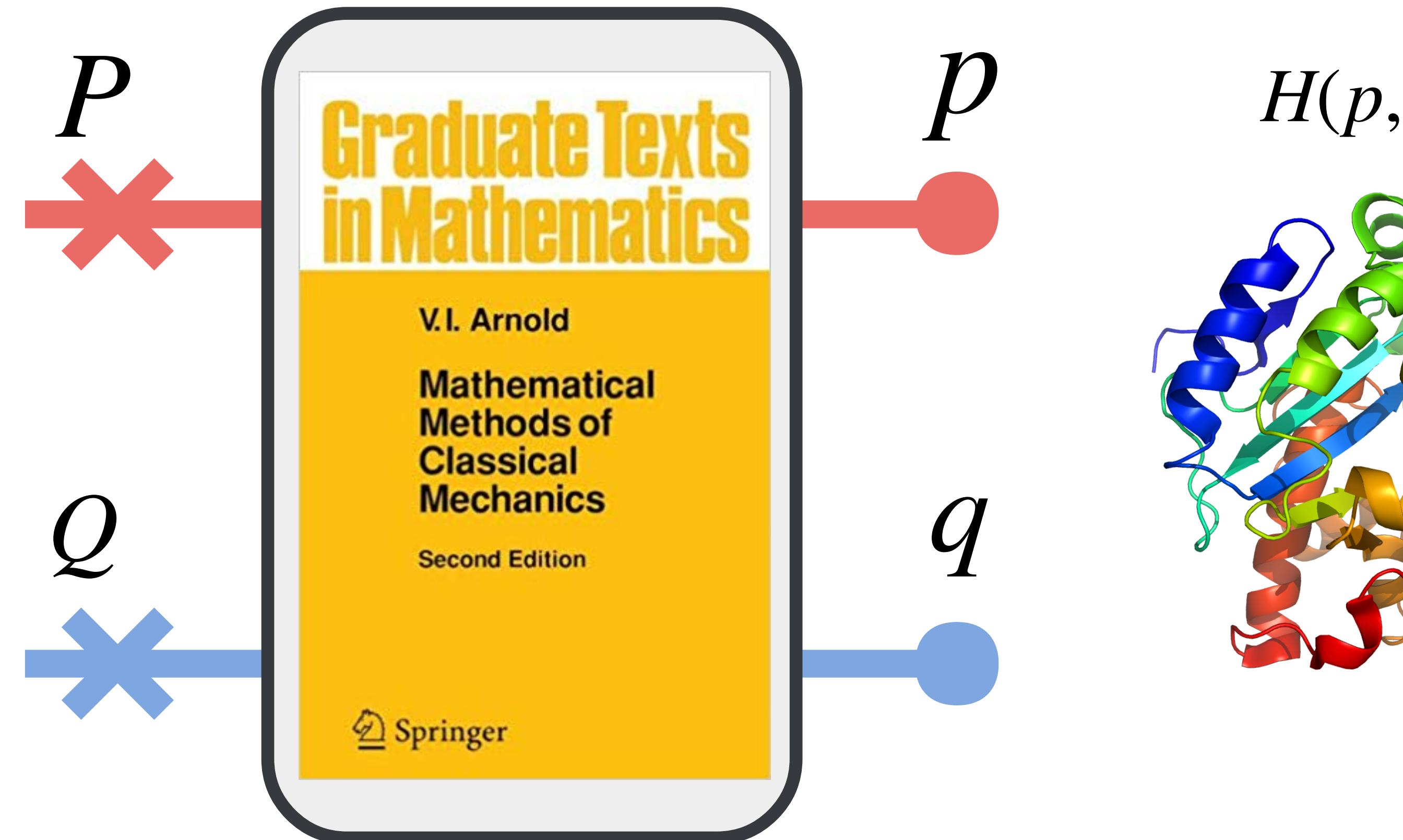
③

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

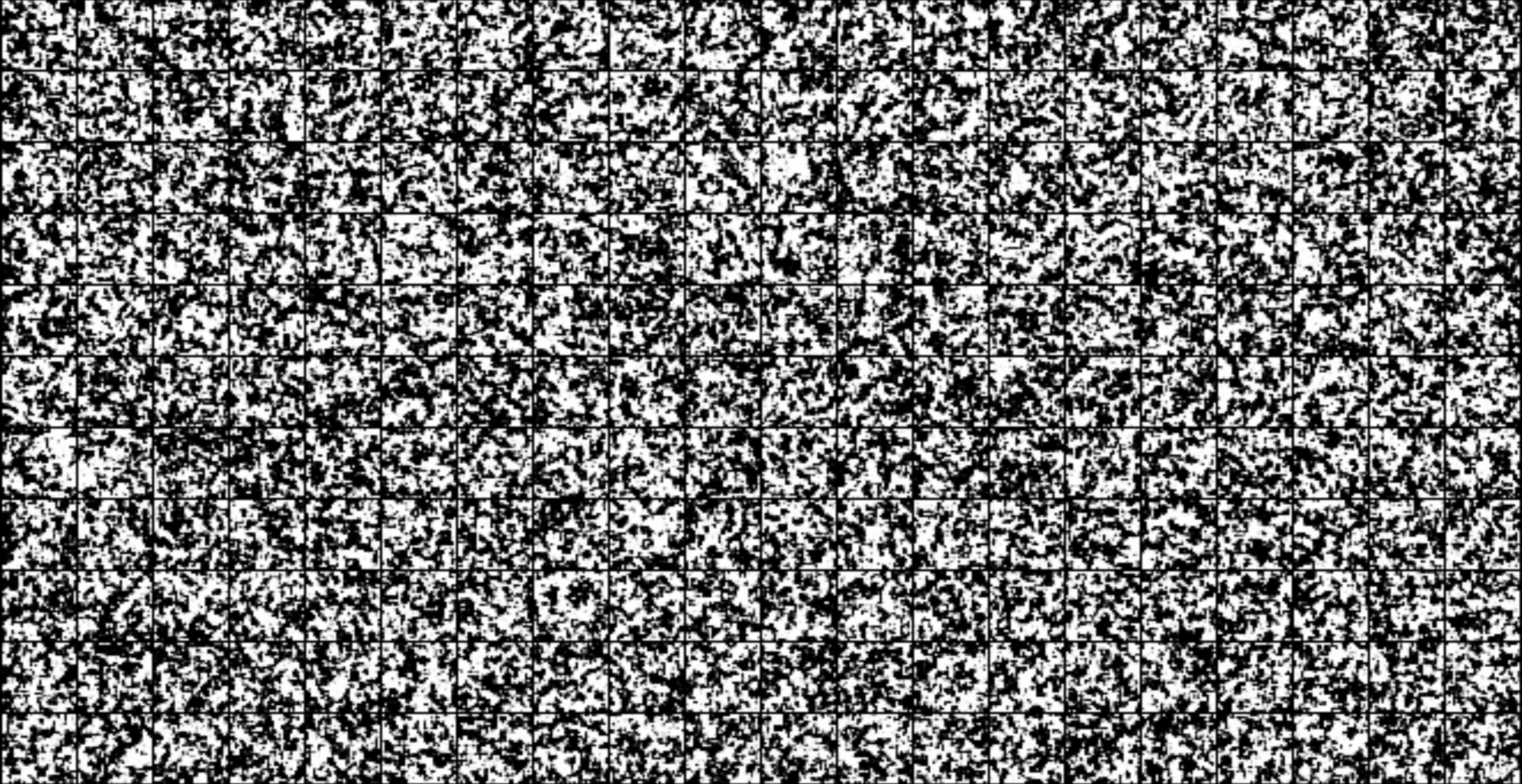
Neural Canonical Transformations

Incompressible symplectic flow in phase space

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

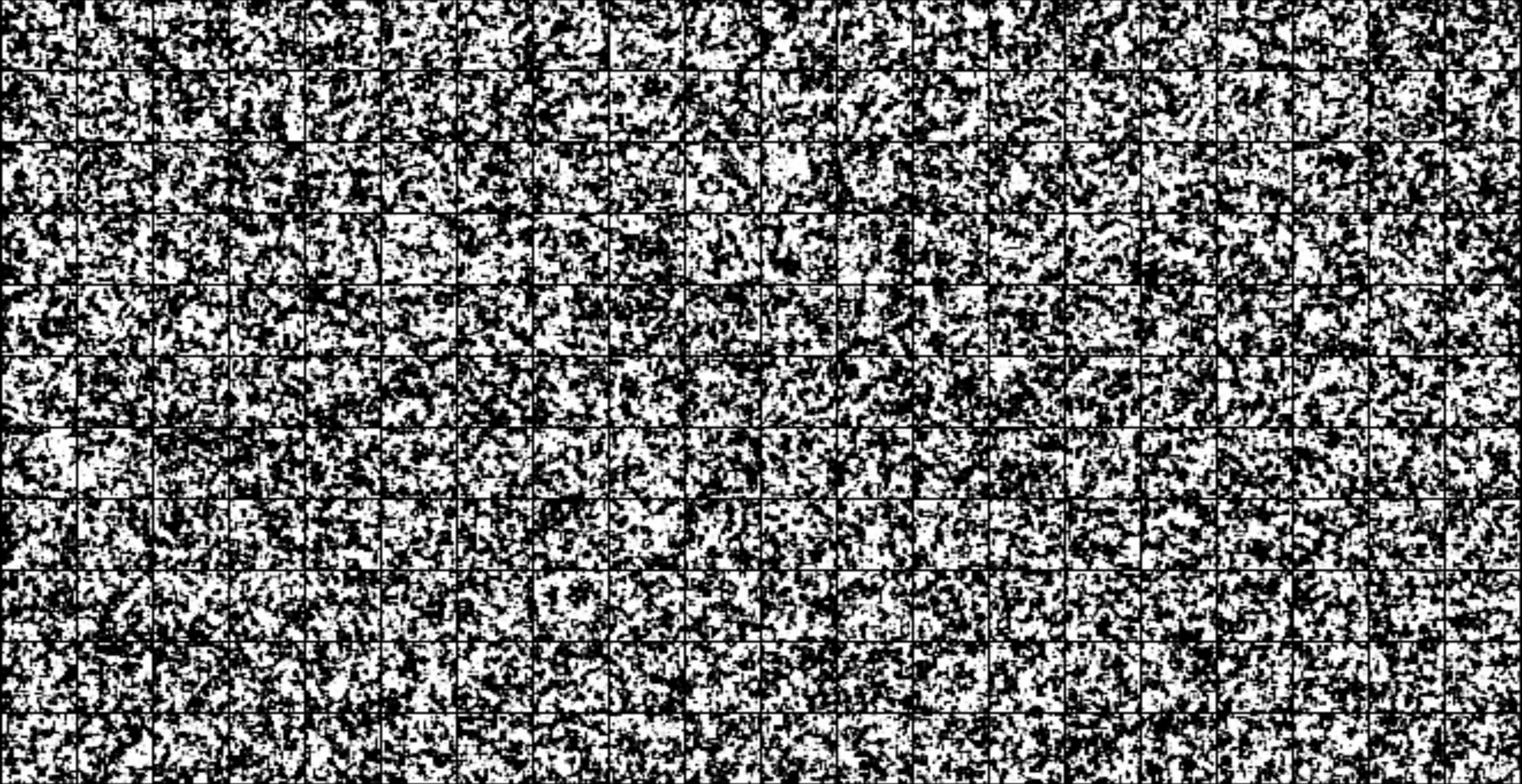


③ Identifying mutually independent collective modes for molecular simulations (MD, PIMD), and effective field theory



④

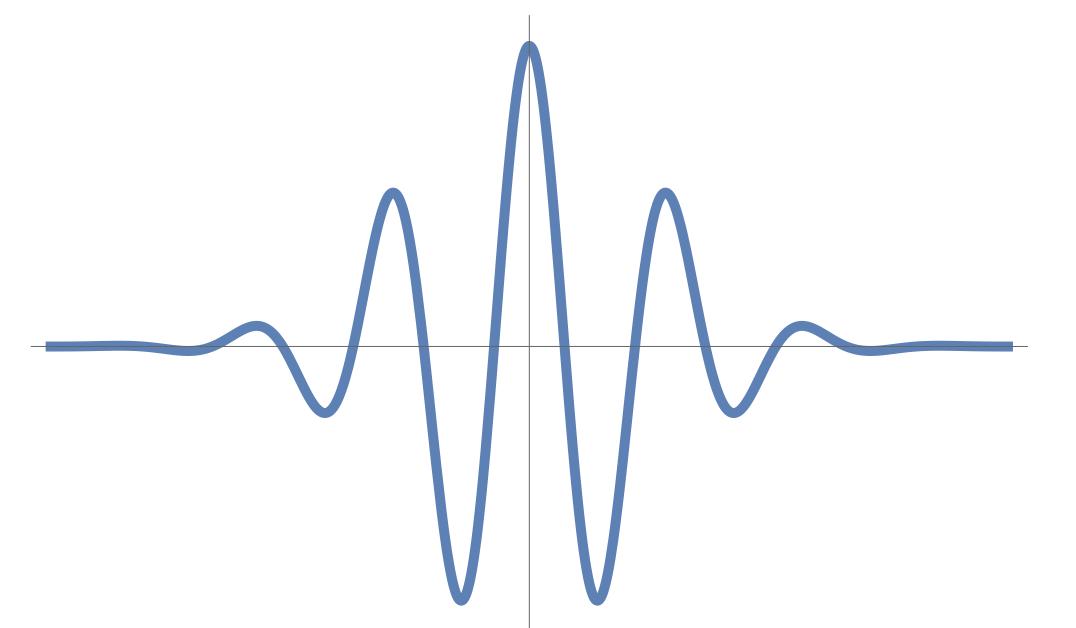
Direct sample magnetic domains respecting physical symmetry



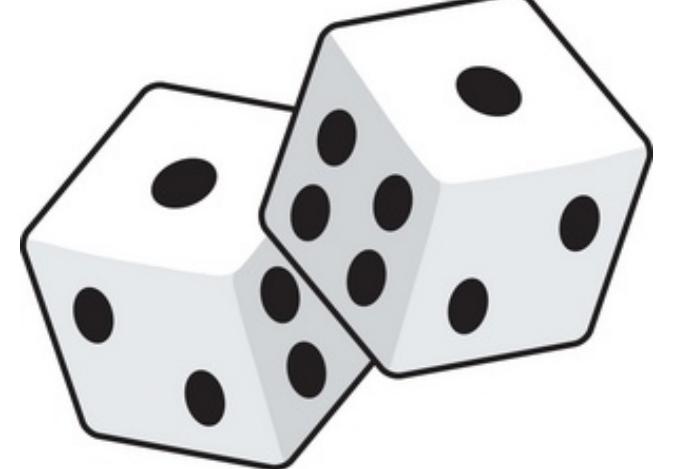
④

Direct sample magnetic domains respecting physical symmetry

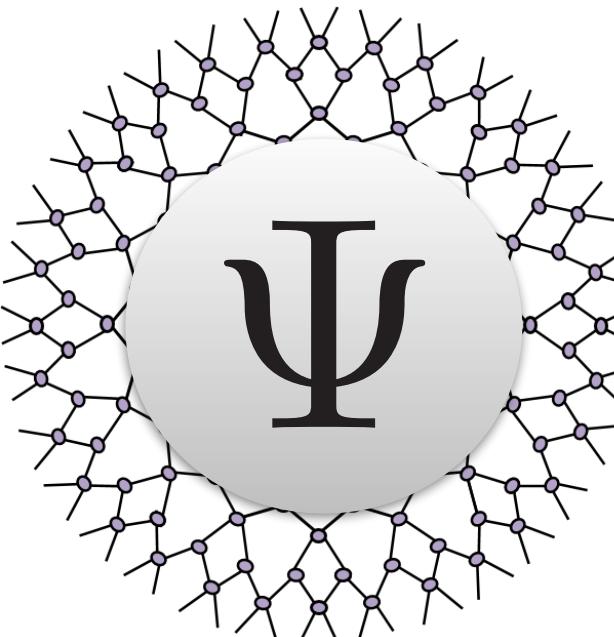
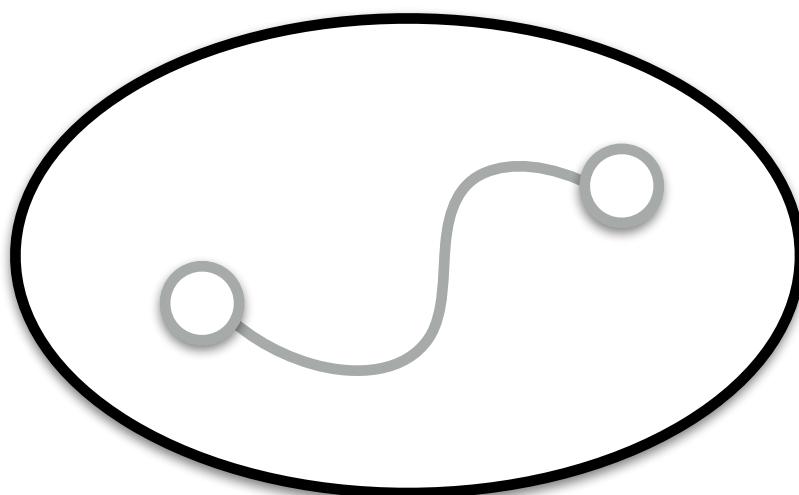
Wavelets



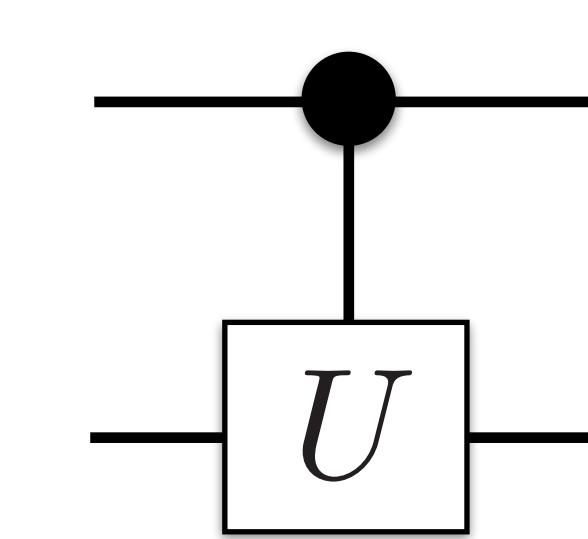
Monte Carlo



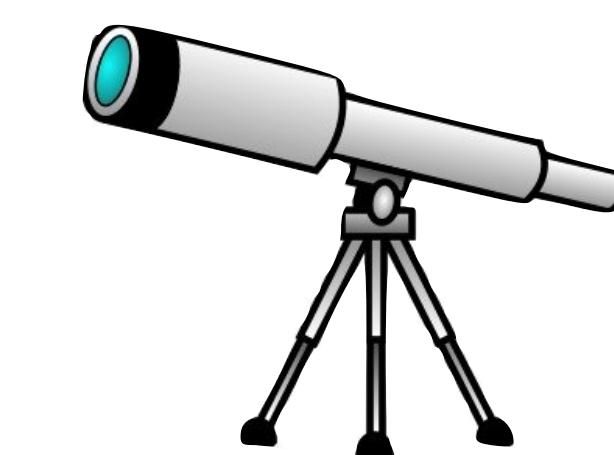
Variational
Inference



Tensor Networks



Quantum Circuits



Holographic RG

Thank You!