∂ifferentiable Scientific Computing

微分万物:深度学习的启示*

王 磊[1,2,†] 刘金国[3]

(1 中国科学院物理研究所 北京 100190)

(2 松山湖材料实验室 东莞 523808)

(3 哈佛大学物理系 剑桥 02138)

《物理》

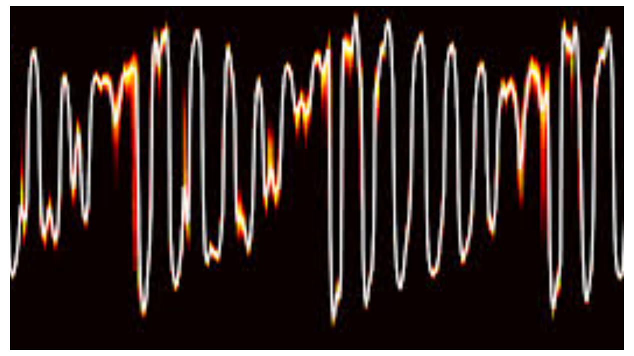
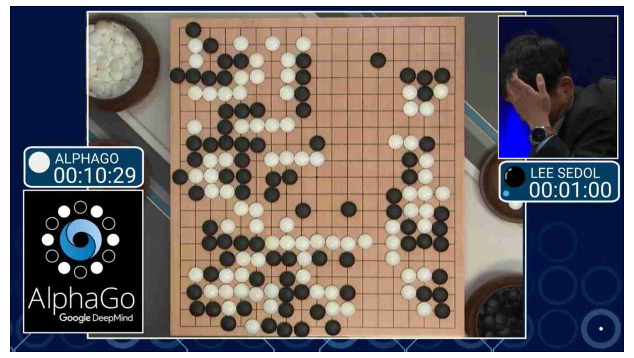
2021年2月

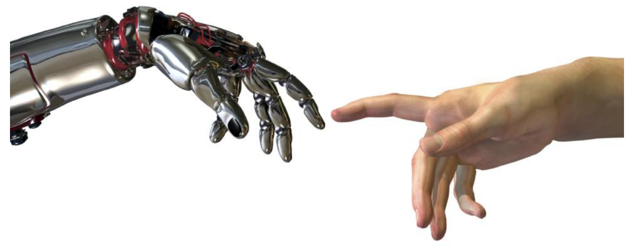
Lei Wang (王磊)

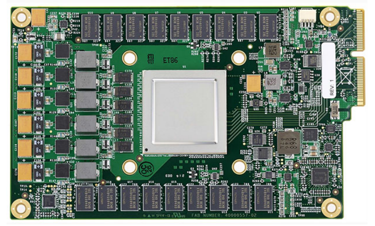
https://wangleiphy.github.io

Institute of Physics, CAS

Why deep learning ?



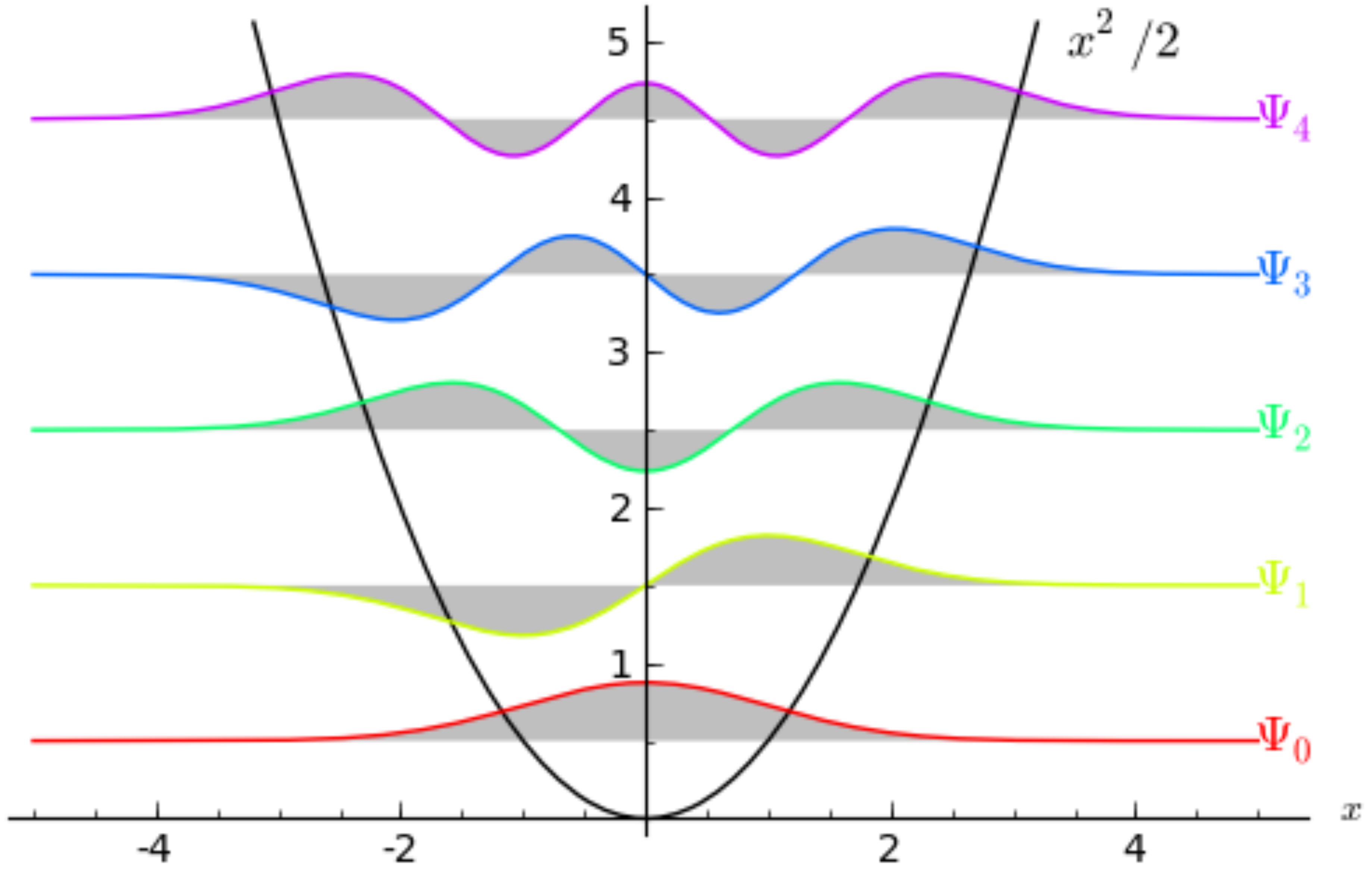



Game changing technology for scientific research

Demo: Inverse Schrodinger Problem

Given ground state density, how to design the potential ?

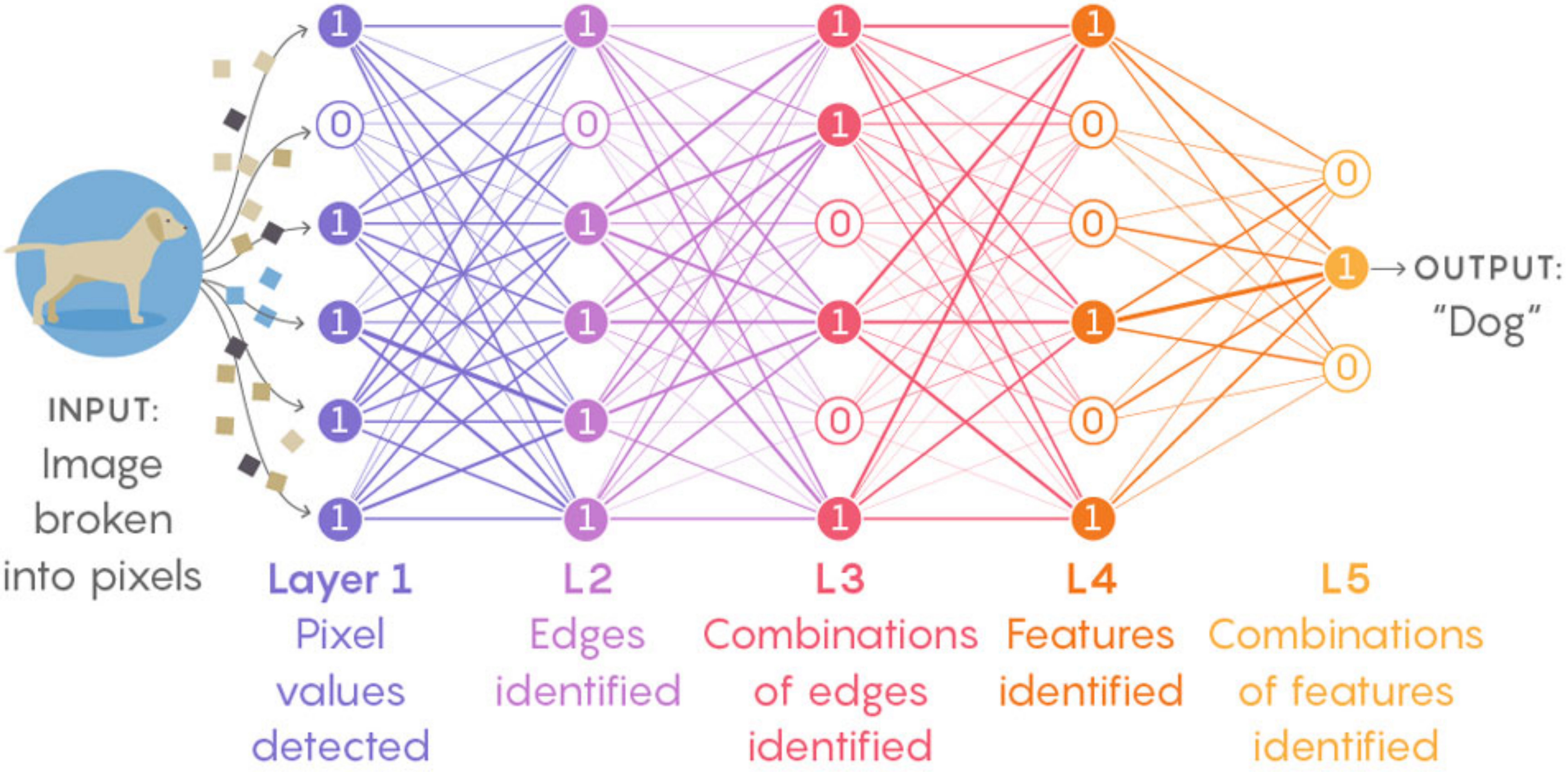
$$\left[-\frac{1}{2}\frac{\partial^2}{\partial x^2} + V(x)\right]\Psi(x) = E\Psi(x)$$



https://colab.research.google.com/drive/1e1NFA-E1Th7nN_9-DzQjAaglH6bwZtVU?usp=sharing

What is under the hood ?

What is deep learning ?



Composes differentiable components to a program e.g. a neural network, then optimizes it with gradients

Computing derivatives of a computer program

ANALYTICAL DIFFERENTIATION ON A DIGITAL COMPUTER

John F. Nolan

Massachusetts Institute of Technology (1953)

A Simple Automatic Derivative Evaluation Program

R. E. WENGERT

General Electric Company, Syracuse, New York*

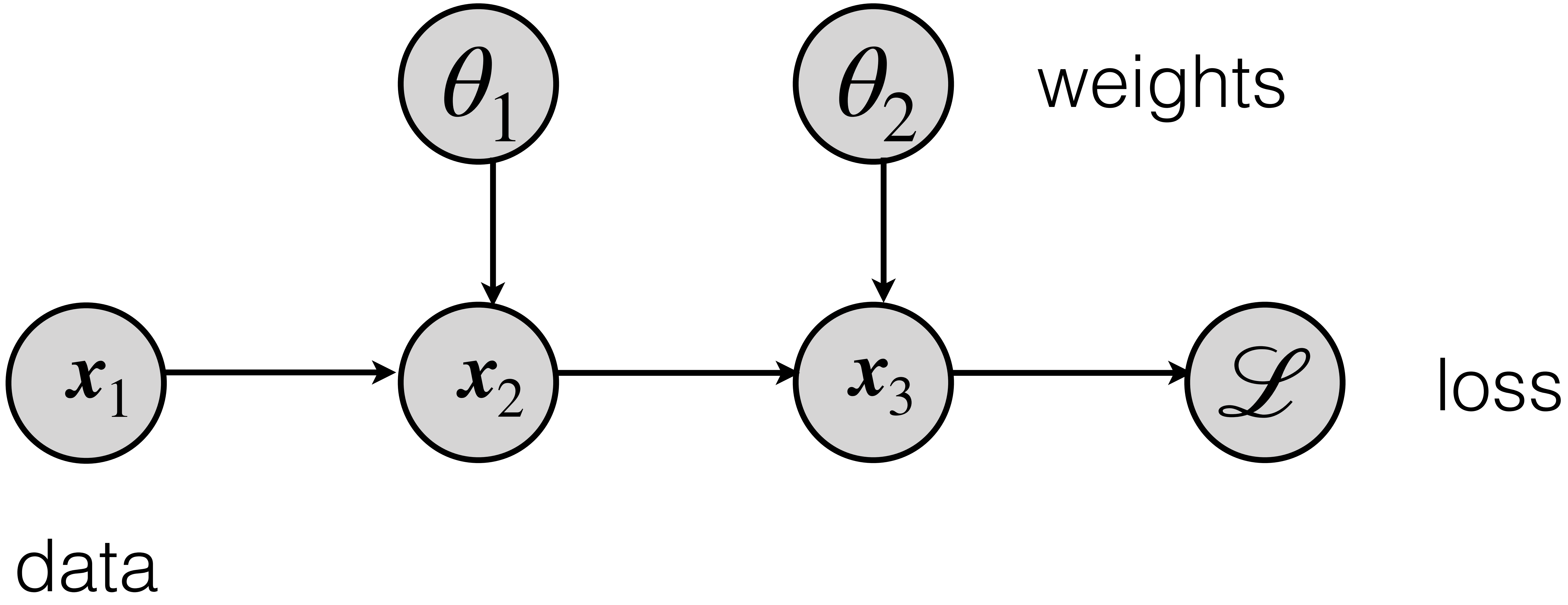
Dual number $x \to x + 1\epsilon \quad \epsilon^2 = 0$

$$f(x + 1\epsilon) = f(x) + f'(x)\epsilon$$

"forward mode" automatic differentiation

Reverse mode automatic differentiation

"comb graph"

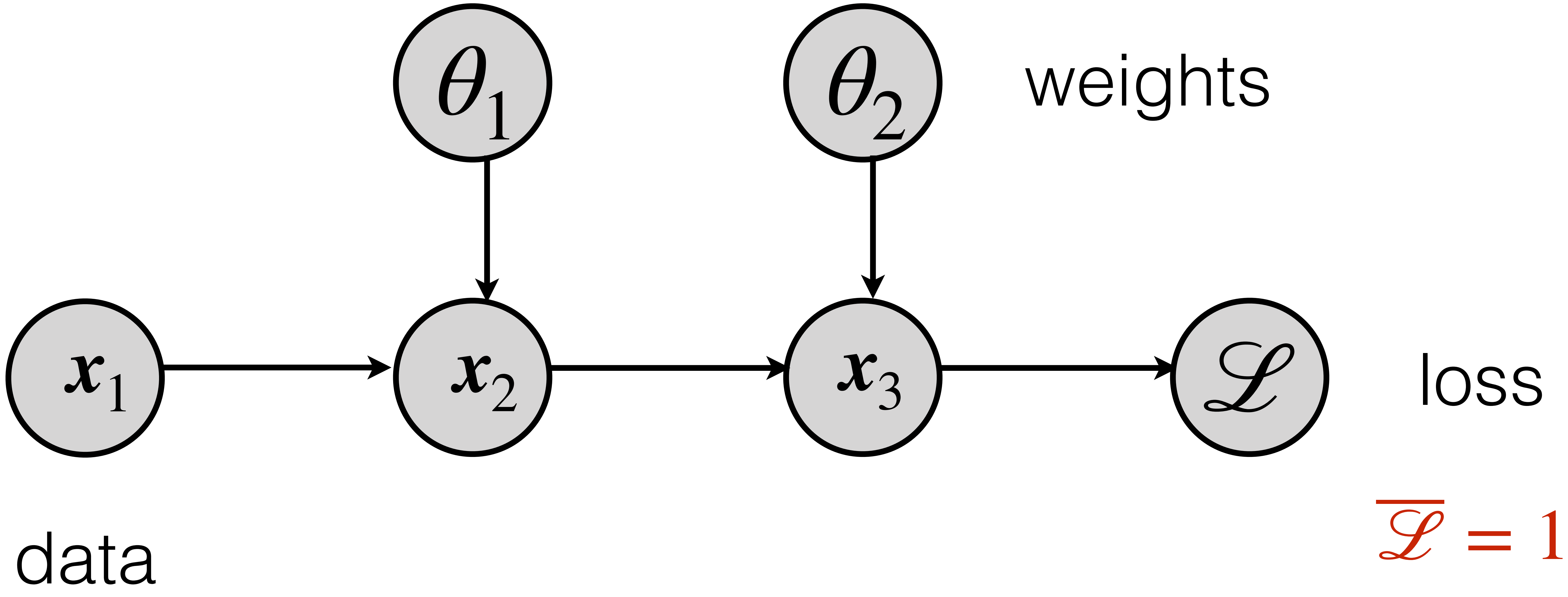


"adjoint variable" $\overline{\boldsymbol{x}} = \frac{\partial \mathscr{L}}{\partial \boldsymbol{x}}$

Pullback the adjoint through the computation graph

Reverse mode automatic differentiation

"comb graph"



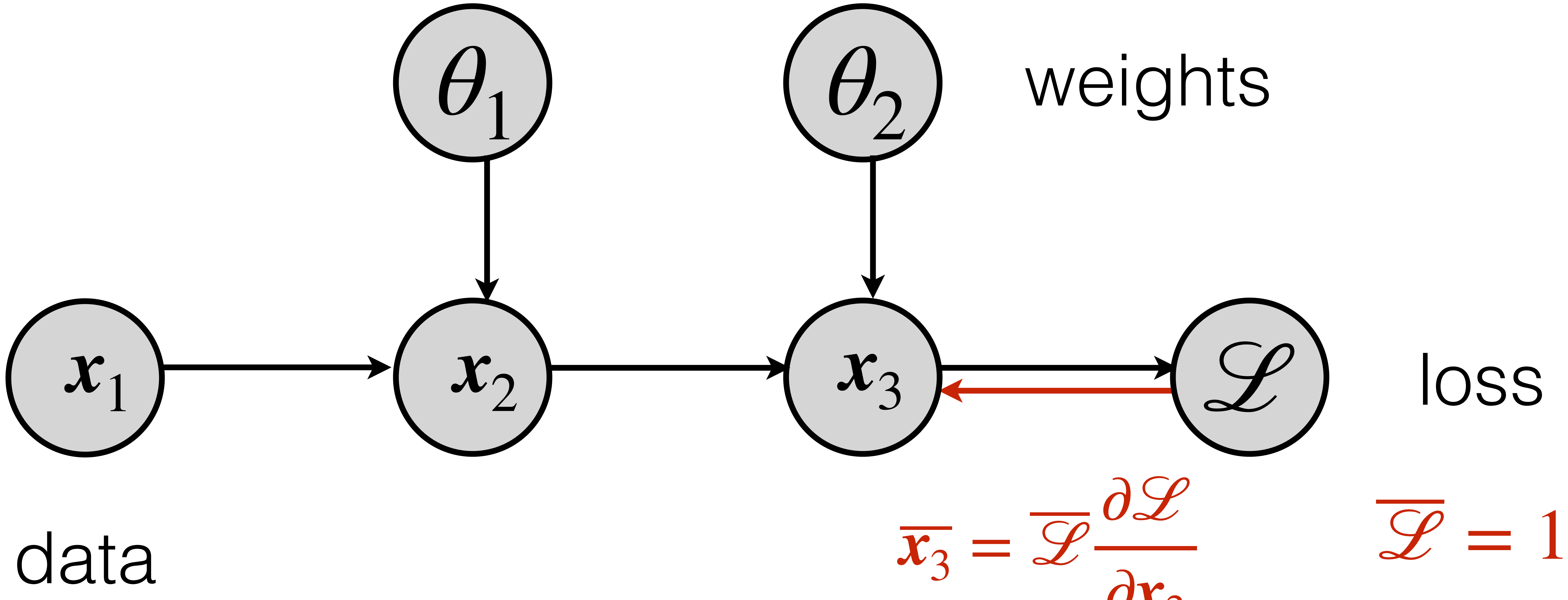
$\overline{\mathscr{L}} = 1$

"adjoint variable" $\bar{\boldsymbol{x}} = \frac{\partial \mathscr{L}}{\partial \boldsymbol{x}}$

Pullback the adjoint through the computation graph

Reverse mode automatic differentiation

“comb graph“



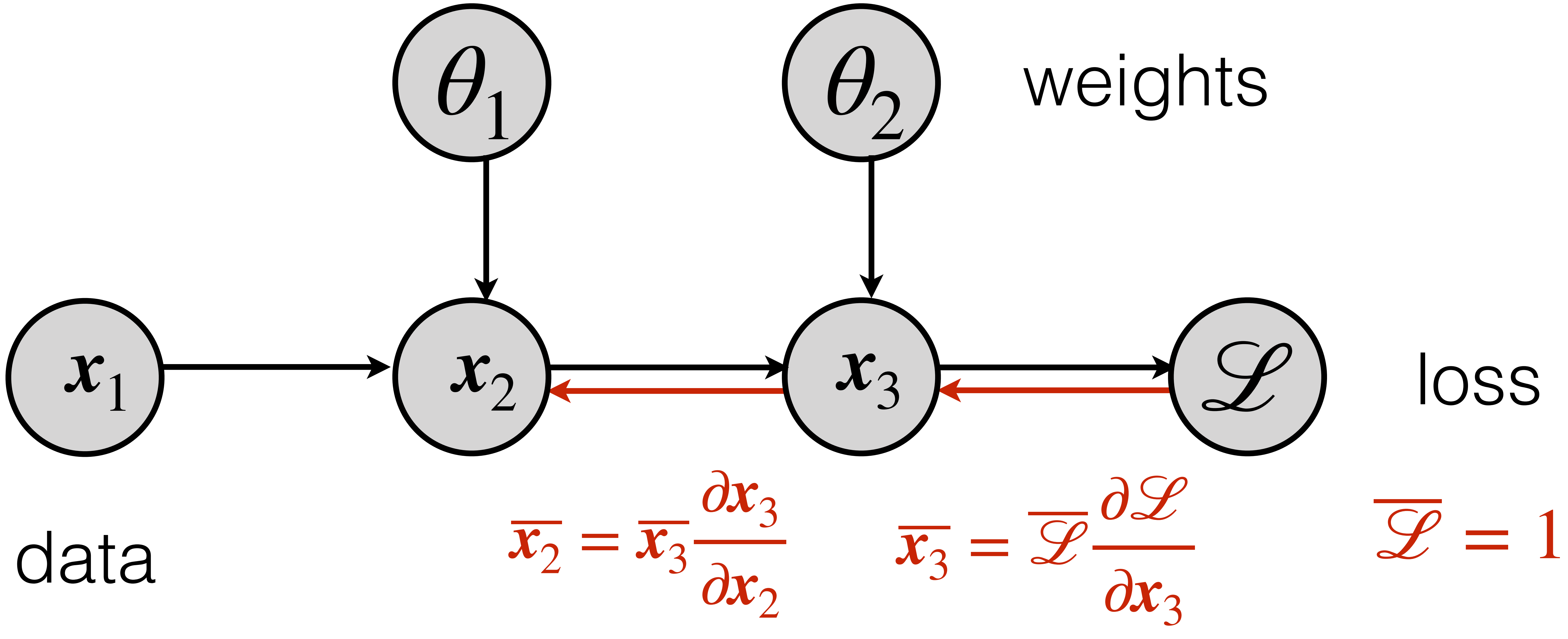
$\overline{x_3} = \overline{\mathscr{L}} \frac{\partial \mathscr{L}}{\partial x_3}$ $\overline{\mathscr{L}} = 1$

“adjoint variable” $\overline{x} = \frac{\partial \mathscr{L}}{\partial x}$

Pullback the adjoint through the computation graph

Reverse mode automatic differentiation

“comb graph“

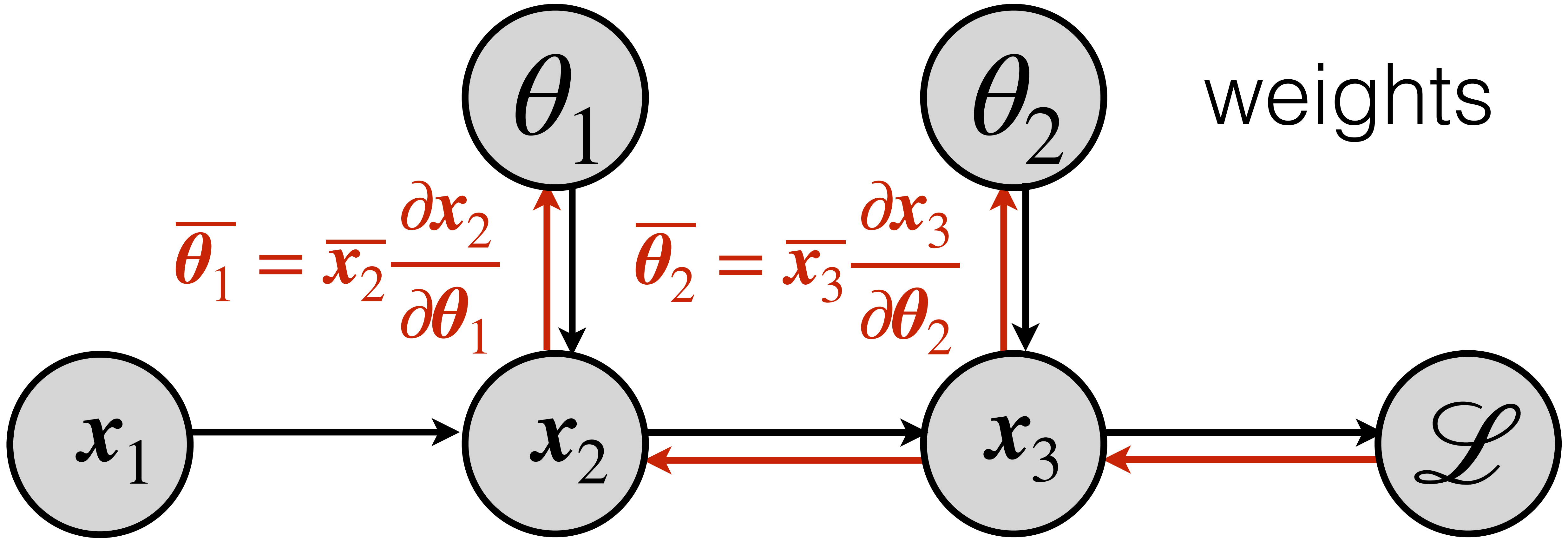


“adjoint variable” $\overline{x} = \frac{\partial \mathscr{L}}{\partial x}$

Pullback the adjoint through the computation graph

Reverse mode automatic differentiation

"comb graph"



$$\overline{\theta_1} = \overline{x_2}\frac{\partial x_2}{\partial \theta_1} \qquad \overline{\theta_2} = \overline{x_3}\frac{\partial x_3}{\partial \theta_2}$$

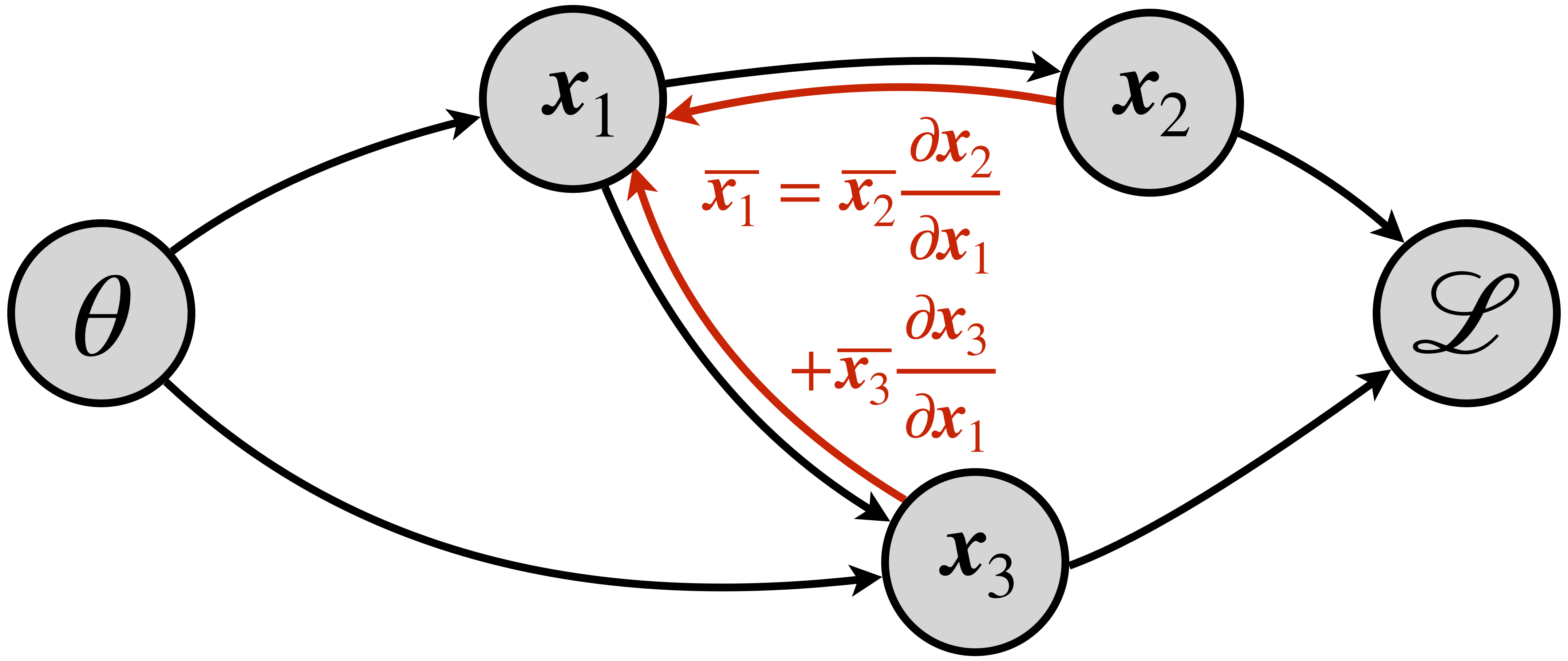
$$\overline{x_2} = \overline{x_3}\frac{\partial x_3}{\partial x_2} \qquad \overline{x_3} = \overline{\mathscr{L}}\frac{\partial \mathscr{L}}{\partial x_3} \qquad \overline{\mathscr{L}} = 1$$

"adjoint variable" $\overline{x} = \frac{\partial \mathscr{L}}{\partial x}$

Pullback the adjoint through the computation graph

Reverse mode automatic differentiation

directed acyclic graph

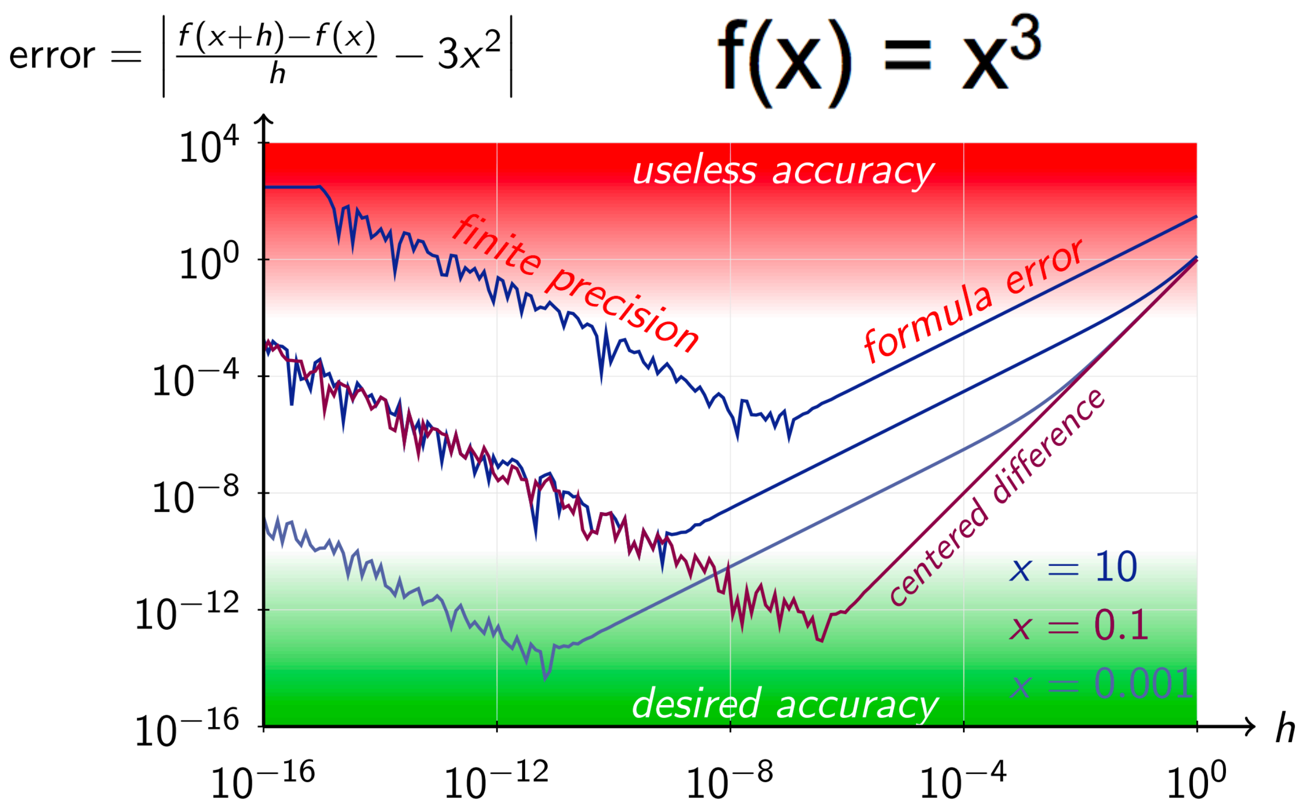


$$\overline{x_i} = \sum_{j:\,\text{child of}\,i} \overline{x_j}\,\frac{\partial x_j}{\partial x_i} \qquad \text{with} \quad \overline{\mathscr{L}} = 1$$

Message passing for the adjoint at each node

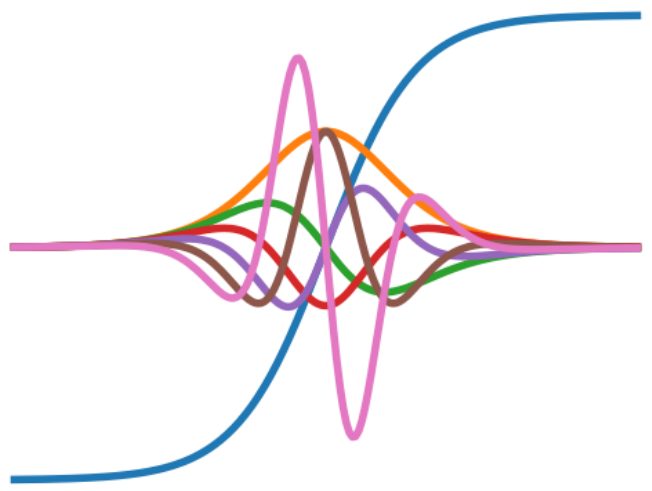
Advantages of automatic differentiation

- Accurate to the machine precision



- Reverse mode has the same computational complexity as the function evaluation: Baur-Strassen theorem ’83

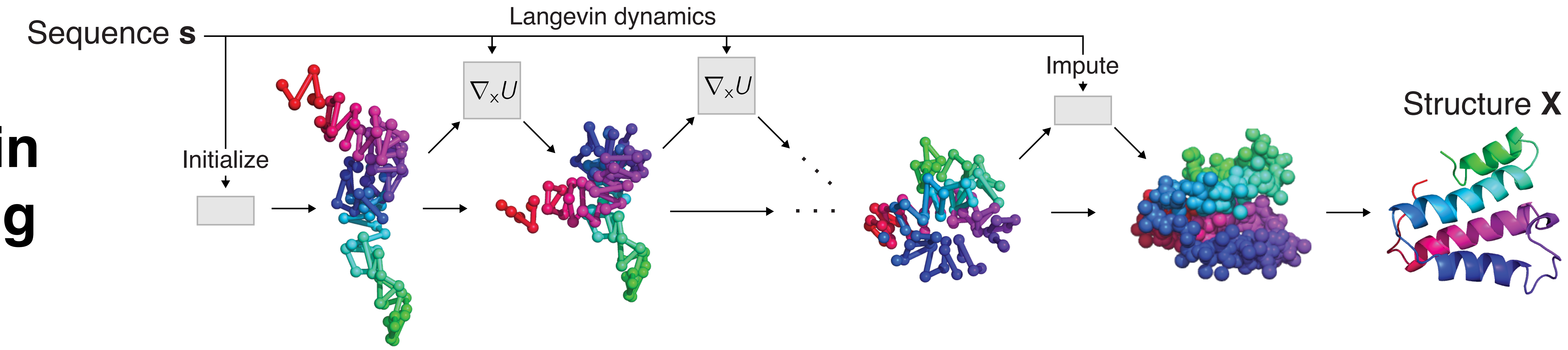
- Supports higher order gradients



```
>>> from autograd import elementwise_grad as egrad  # for functions that vectorize over inputs
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-7, 7, 200)
>>> plt.plot(x, tanh(x),
...          x, egrad(tanh)(x),                                     # first  derivative
...          x, egrad(egrad(tanh))(x),                              # second derivative
...          x, egrad(egrad(egrad(tanh)))(x),                       # third  derivative
...          x, egrad(egrad(egrad(egrad(tanh))))(x),                # fourth derivative
...          x, egrad(egrad(egrad(egrad(egrad(tanh)))))(x),         # fifth  derivative
...          x, egrad(egrad(egrad(egrad(egrad(egrad(tanh))))))(x))  # sixth  derivative
>>> plt.show()
```

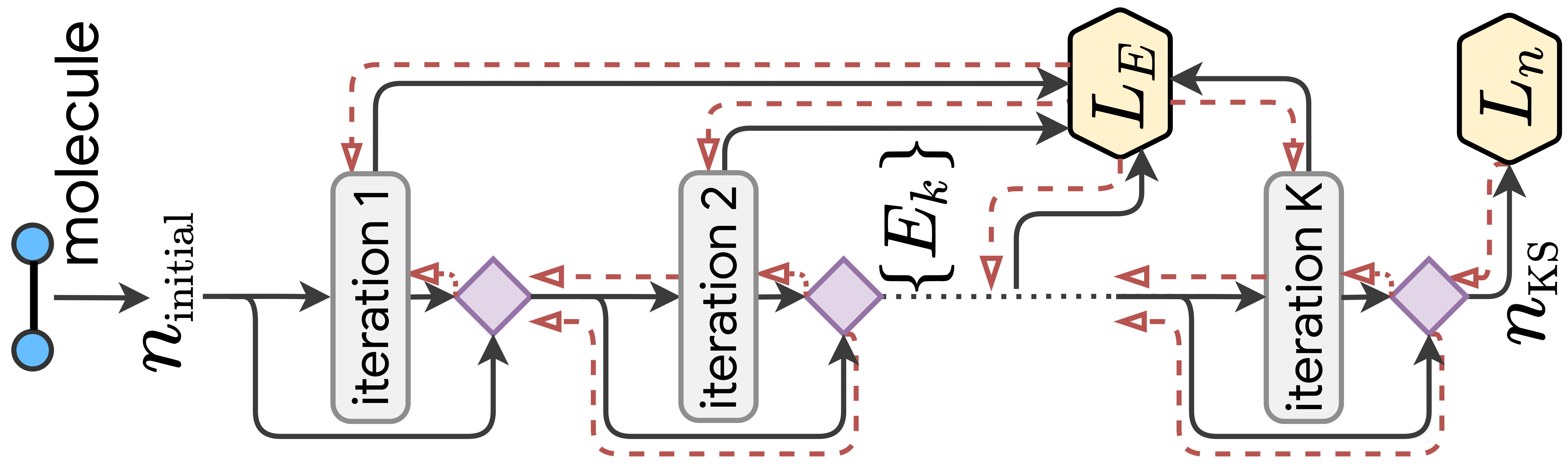
Differentiable simulations

Protein folding



Ingraham et al
ICLR '19

Learning density functionals



KS self-consistent calculation

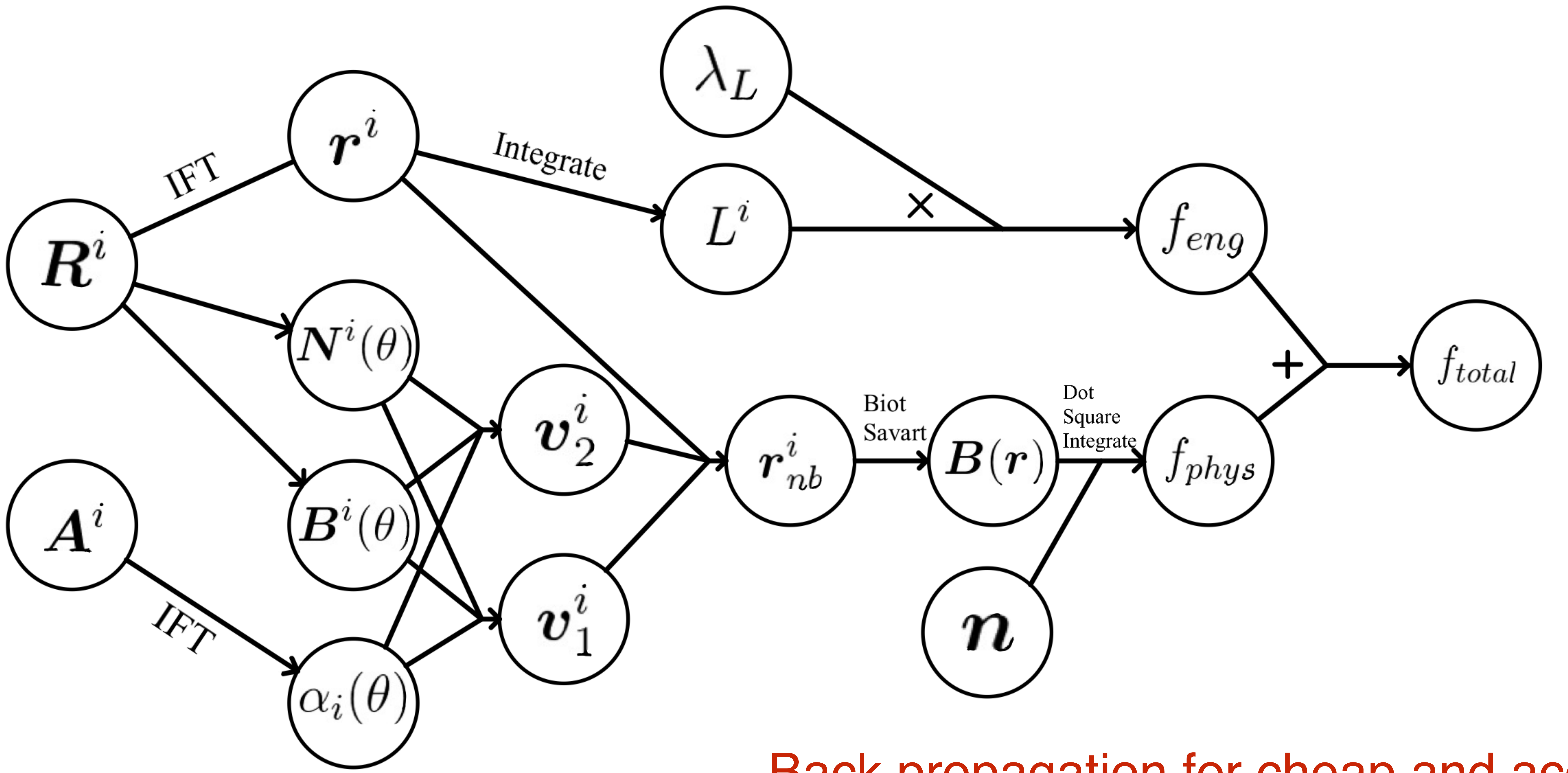
Li et al
PRL '21

Kasim, Vinko, PRL '21
Dick et al, 2106.04481

Coil design in fusion reactors (stellarator)

Differentiable stellarator design

Coil parameters



Total cost

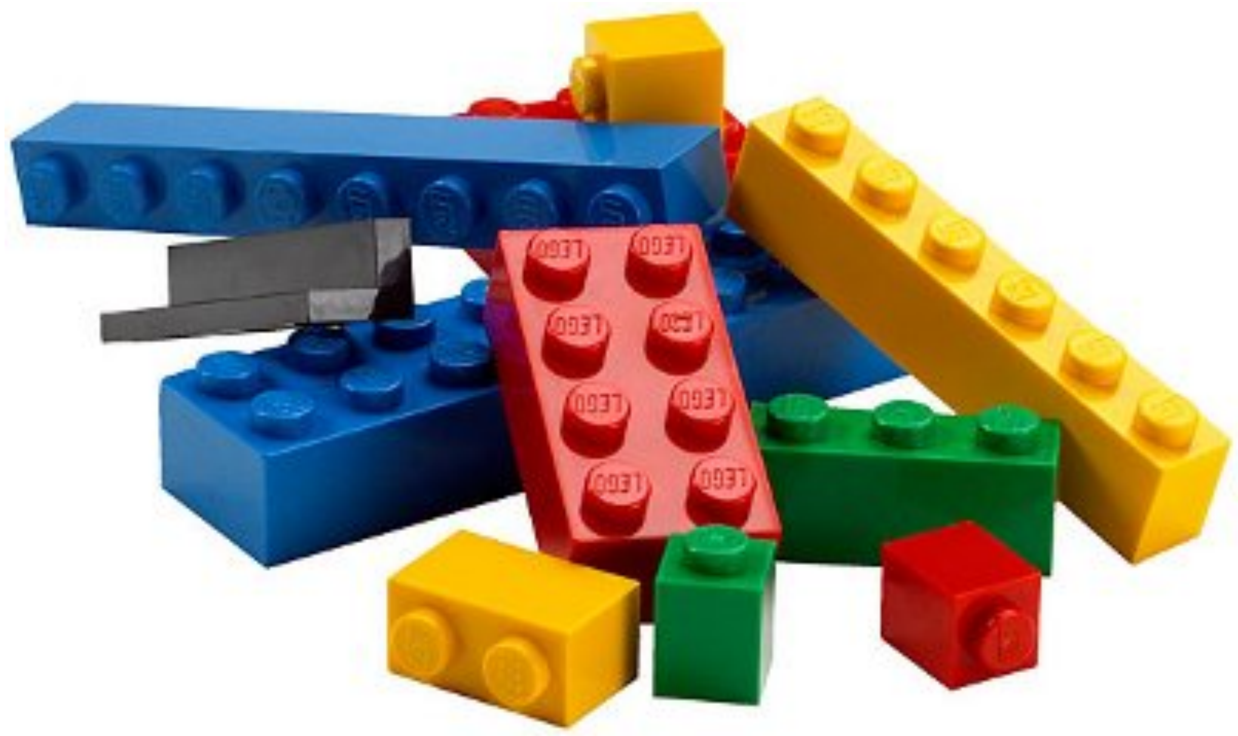
Back propagation for cheap and accurate gradient

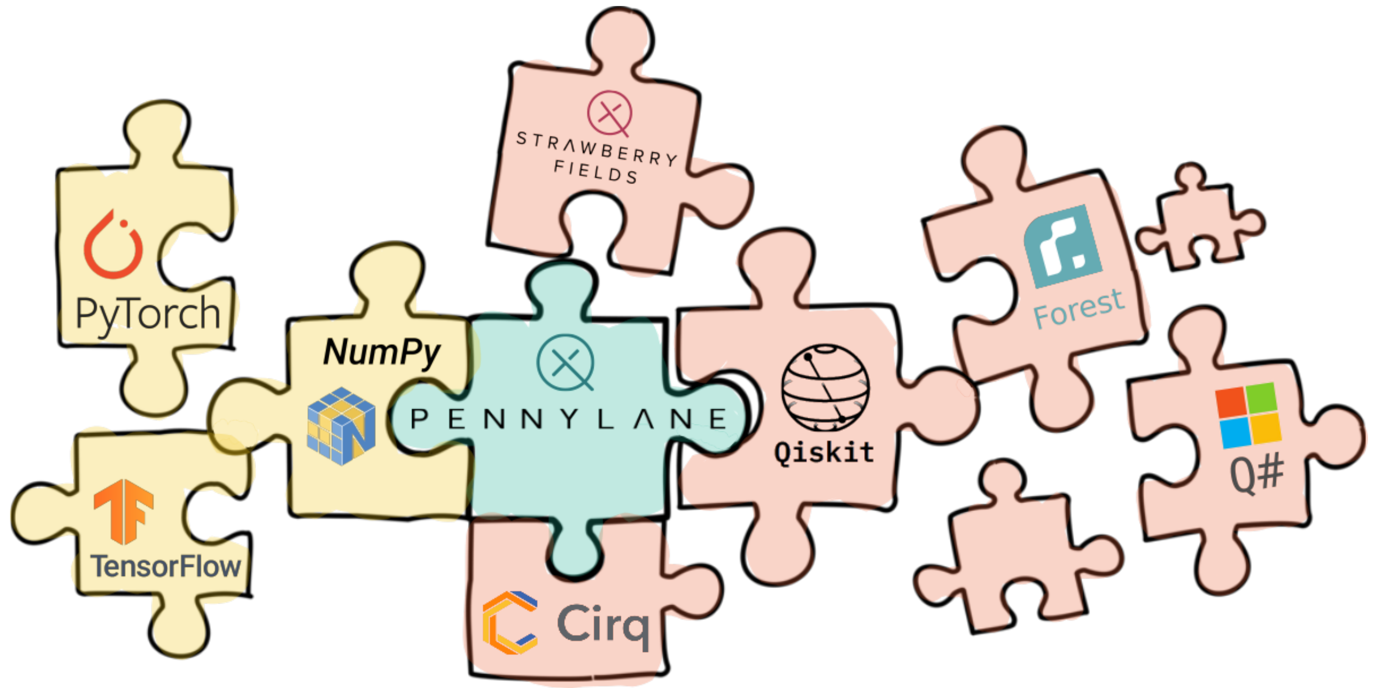
McGreivy et al 2009.00196

Differentiable programming is broader than training neural networks

How to think about AD ?

- AD is modular, and one can control its granularity
- Benefits of writing customized primitives
  - Reducing memory usage
  - Increasing numerical stability
  - Call to external libraries written agnostically to AD
    (or, even a quantum processor)





https://github.com/PennyLaneAI/pennylane

Forward versus reverse modes

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

Forward mode
"Propagate the perturbation"

Same direction as the function evaluation

Same complexity as numerical finite difference

Reverse mode
"Pull back the adjoint"

Backtrace the computation graph

Needs to store intermediate results

Backpropagation = Reverse mode AD applied to neural networks

Examples of primitives

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

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

Operators	+, −, *, /, (−), **, %, <, <=, ==, !=, >=, >
Basic math functions	exp, log, square, sqrt, sin, cos, tan, sinh, cosh, tanh, sinc, abs, fabs, logaddexp, logaddexp2, absolute, reciprocal, exp2, expm1, log2, log10, log1p, arcsin, arccos, arctan, arcsinh, arccosh, arctanh, rad2deg, degrees, deg2rad, radians
Complex numbers	real, imag, conj, angle, fft, fftshift, ifftshift, real_if_close
Array reductions	sum, mean, prod, var, std, max, min, amax, amin
Array reshaping	reshape, ravel, squeeze, diag, roll, array_split, split, vsplit, hsplit, dsplit, expand_dims, flipud, fliplr, rot90, swapaxes, rollaxis, transpose, atleast_1d, atleast_2d, atleast_3d
Linear algebra	dot, tensordot, einsum, cross, trace, outer, det, slogdet, inv, norm, eigh, cholesky, sqrtm, solve_triangular
Other array operations	cumsum, clip, maximum, minimum, sort, msort, partition, concatenate, diagonal, truncate_pad, tile, full, triu, tril, where, diff, nan_to_num, vstack, hstack
Probability functions	t.pdf, t.cdf, t.logpdf, t.logcdf, multivariate_normal.logpdf, multivariate_normal.pdf, multivariate_normal.entropy, norm.pdf, norm.cdf, norm.logpdf, norm.logcdf,

autograd/numpy_vjps.py at mas

GitHub, Inc. (US) https://github.com/HIPS/autograd/b 133%

```python
# ----- 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)
```

Loop/Condition/Sort/Permutations are also differentiable

http://videolectures.net/deeplearning2017_johnson_automatic_differentiation/

Differentiable programming tools

HIPS/autograd

theano

Zygote

PyTorch

TensorFlow

SciML

Keras

NiLang

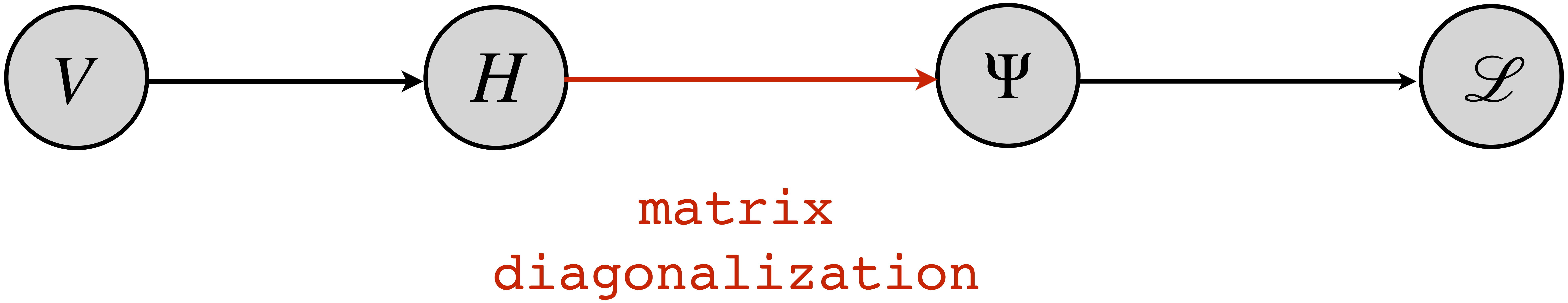
Differentiable Scientific Computing

- Many scientific computations (FFT, 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/Hartree-Fock/Coupled Cluster/Gutzwiller/Molecular Dynamics…

Differentiate through domain-specific computational processes to solve learning, control, optimization and inverse problems

Differentiable Eigensolver

Inverse Schrodinger Problem



Useful for inverse Kohn-Sham problem, Jensen & Wasserman '17

Differentiable Eigensolver

$$H\Psi = \Psi E$$

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

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

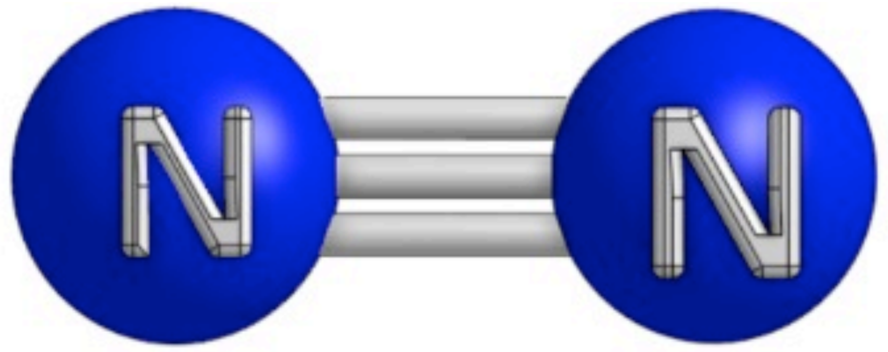
Hamiltonian engineering via differentiable programming

https://github.com/wangleiphy/DL4CSRC/tree/master/2-ising See also Fujita et al, PRB '18

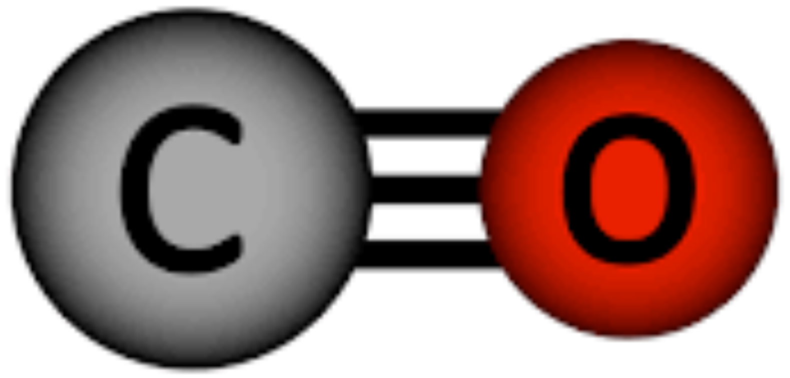
Differentiable Quantum Chemistry

nuclear charge

$$E(Z+\epsilon) = E(Z) + E'(Z)\epsilon + \frac{1}{2}E''(Z)\epsilon^2 + \ldots$$



$Z = (7,7)$



$Z = (6,8)$

Quantum Alchemy 2109.11238

AD for SCF: Steiger et al, Future Generation Computer Systems ’05, Tamayo-Mendoza et al ACS Cent. Sci. ’18
AD for coupled cluster 2011.11690   AD for VMC: Sorella and Capriotti J. Chem. Phys. ’10
Codes: https://github.com/diffqc/dqc https://github.com/CCQC/Quax https://github.com/fishjojo/pyscfad

Differentiable density functional theory

$$Q_n = \left.\frac{d^n E}{d\lambda^n}\right|_{\lambda \to 0}$$

type of perturbation λ	order n	physical property Q
displacements of atoms	1	atomic force
$\delta\mathbf{R}$	2	force constants
	≥ 3	anharmonic force constants
homogeneous strain η	1	stress
	2	elastic constants
	≥ 3	higher order elastic constants
homogeneous electric field $\mathbf{E}$	1	dipole moment
	2	polarizability
$\delta\mathbf{R} + \eta$	2+1	Grüneisen parameter
$\delta\mathbf{R} + \mathbf{E}$	1+2	Raman scattering cross section

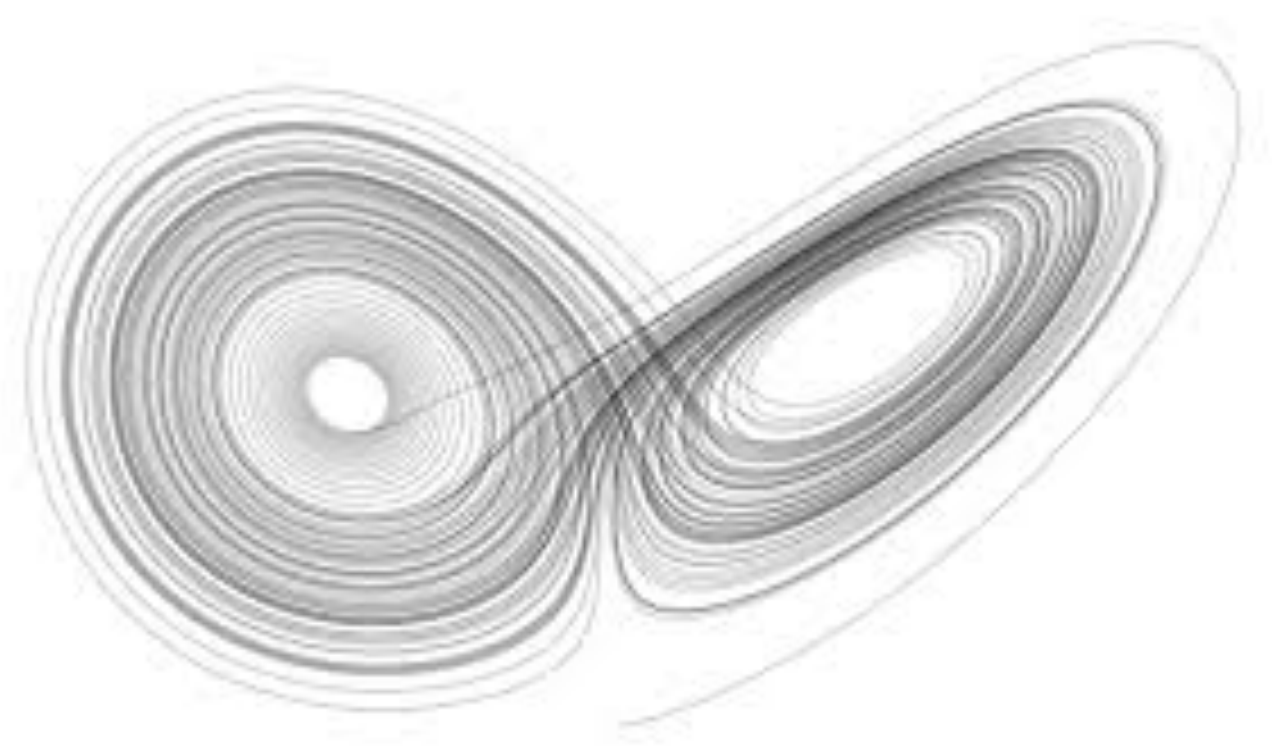
Baroni et al, RMP 2001

Differentiable DFT for a unified, flexible, and (very likely) more efficient framework

Differentiable ODE integrators

"Neural ODE" Chen et al, 1806.07366

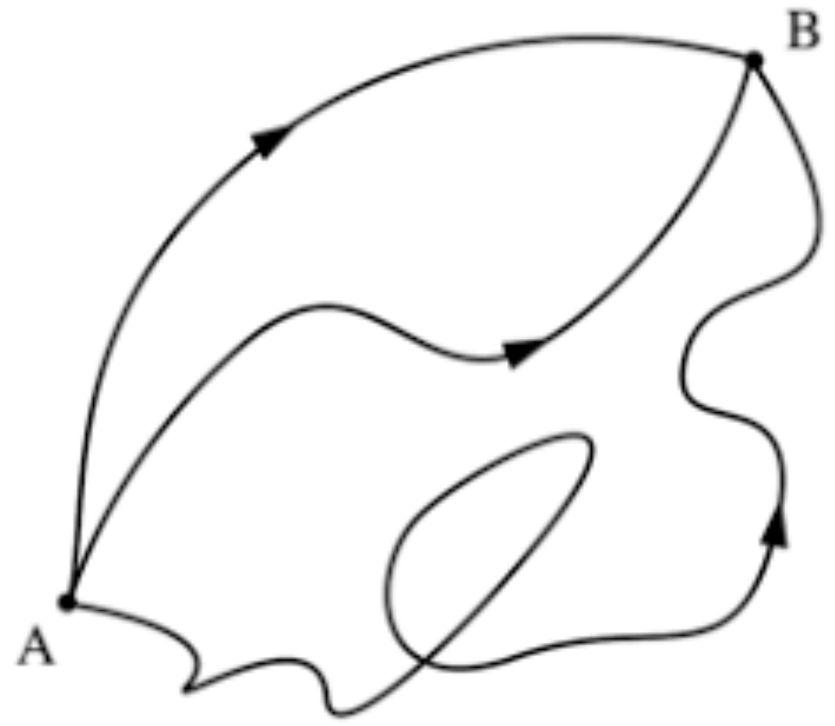
Dynamics systems



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

Classical and quantum control

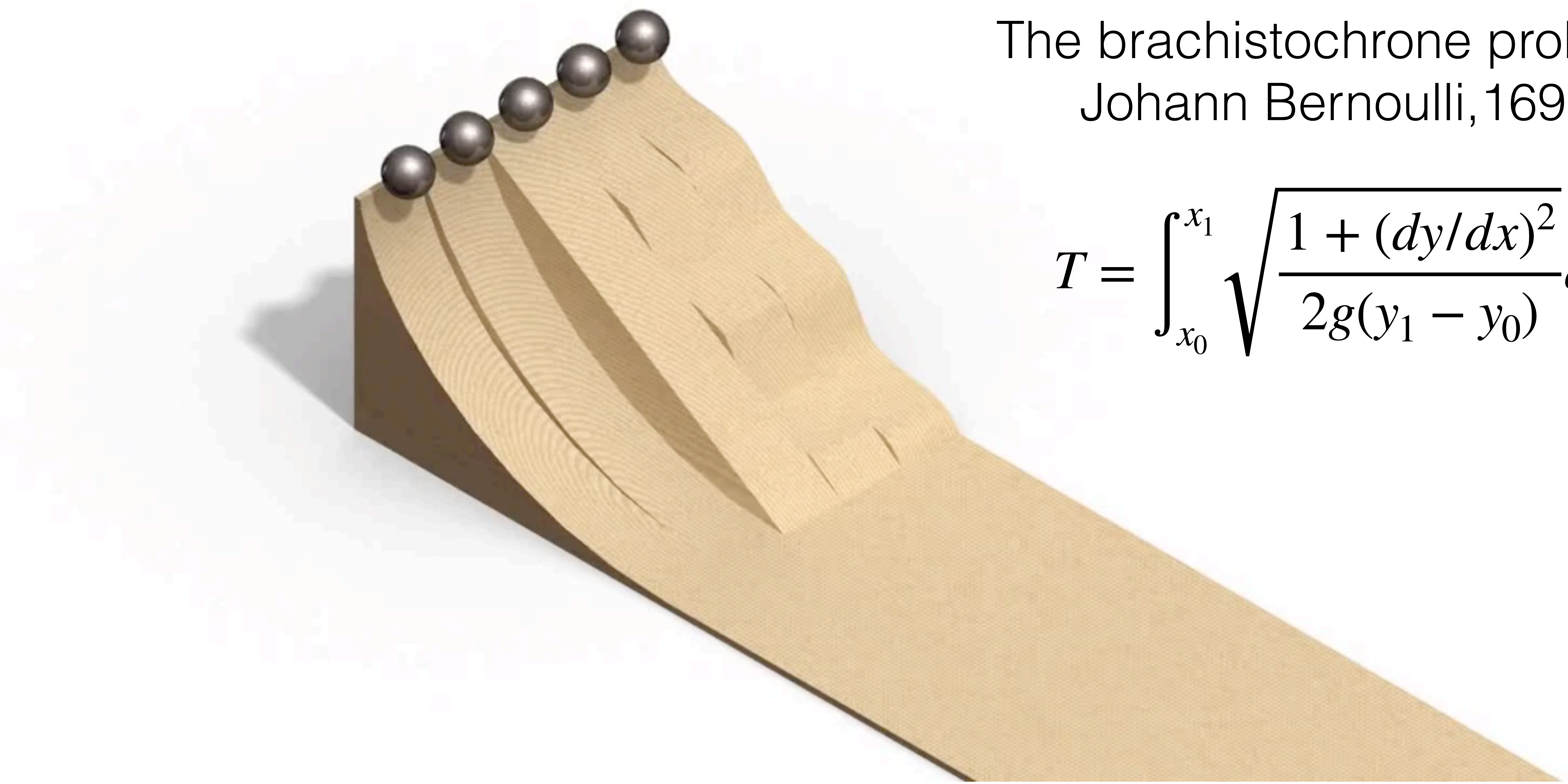
Principle of least actions



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

Optics, (quantum) mechanics, field theory…

Differentiable functional optimization

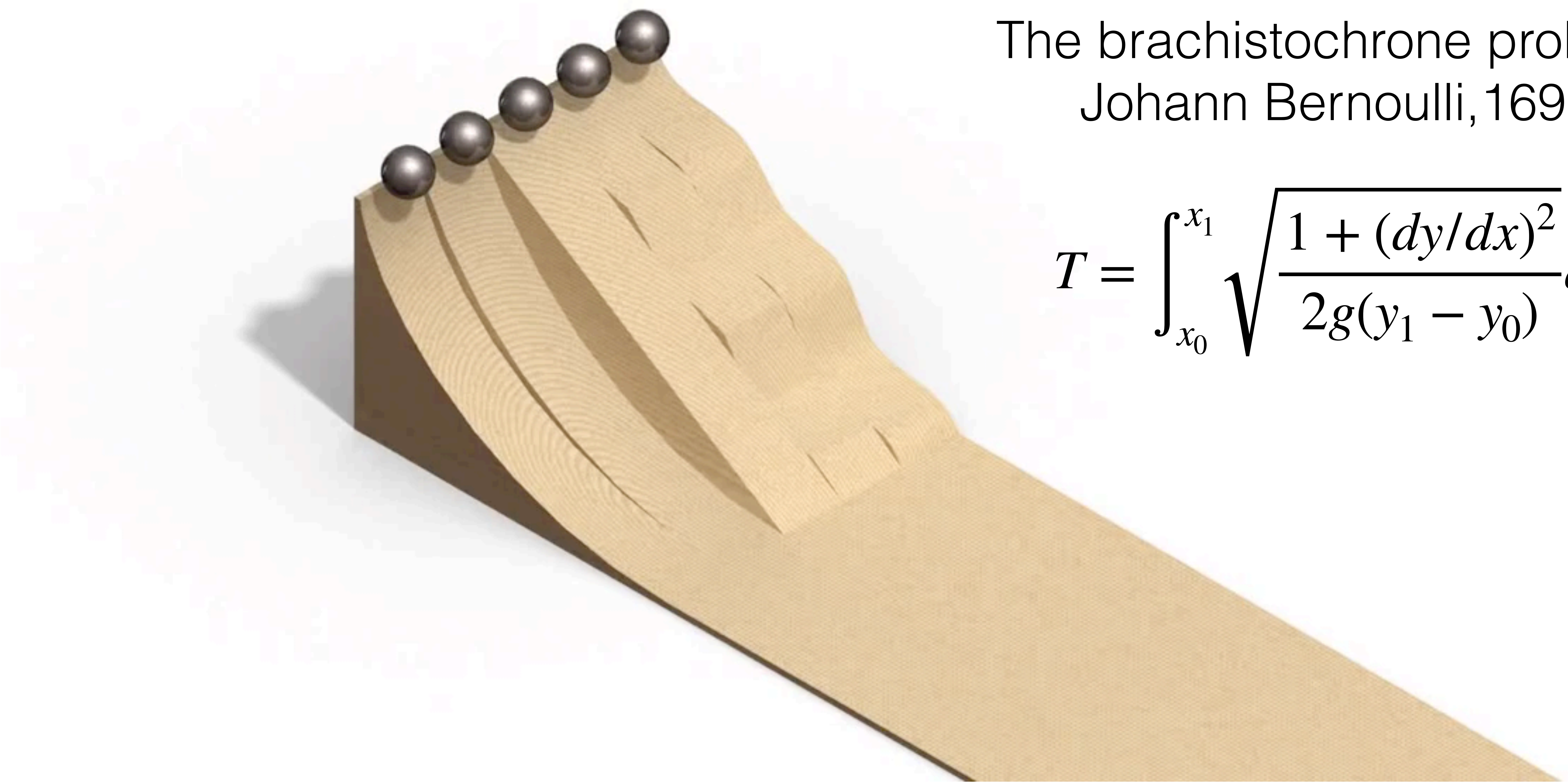


The brachistochrone problem
Johann Bernoulli, 1696

$$T = \int_{x_0}^{x_1} \sqrt{\frac{1 + (dy/dx)^2}{2g(y_1 - y_0)}} dx$$

https://github.com/QuantumBFS/SSSS/tree/master/1_deep_learning/brachistochrone

Differentiable functional optimization



The brachistochrone problem
Johann Bernoulli, 1696

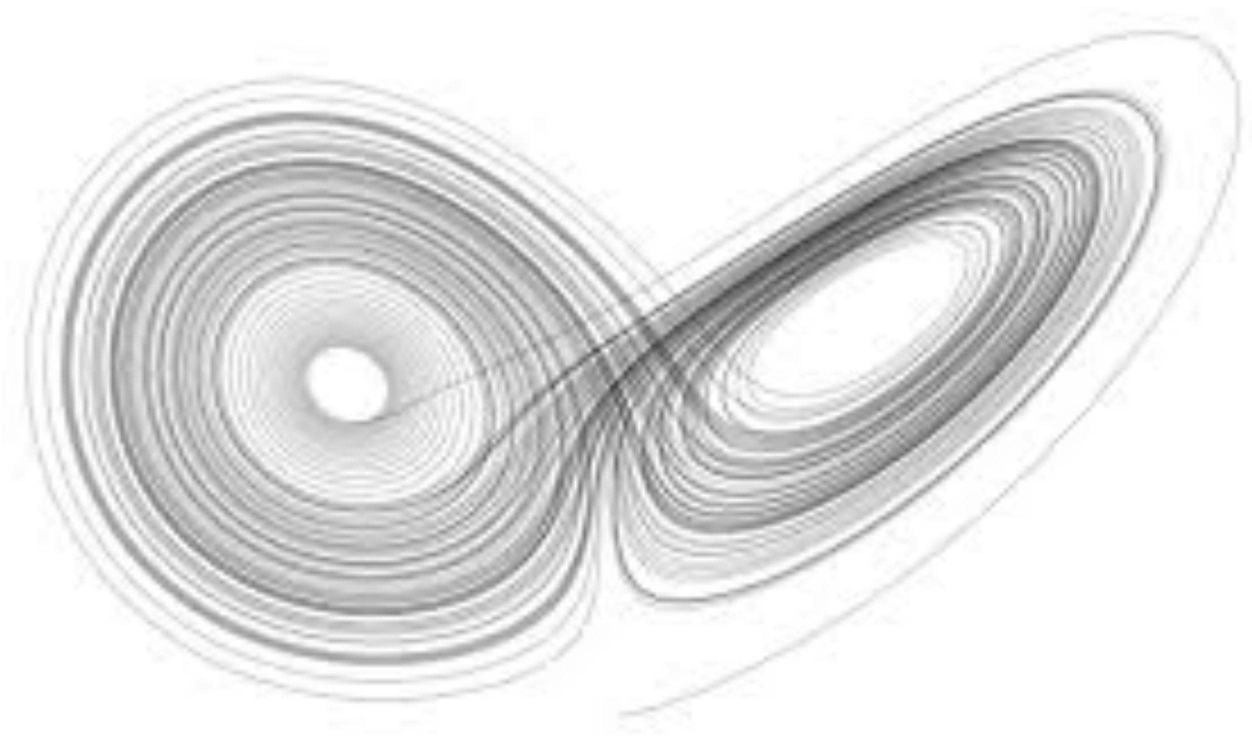
$$T = \int_{x_0}^{x_1} \sqrt{\frac{1 + (dy/dx)^2}{2g(y_1 - y_0)}} dx$$

https://github.com/QuantumBFS/SSSS/tree/master/1_deep_learning/brachistochrone

Differentiable ODE integrators

"Neural ODE" Chen et al, 1806.07366

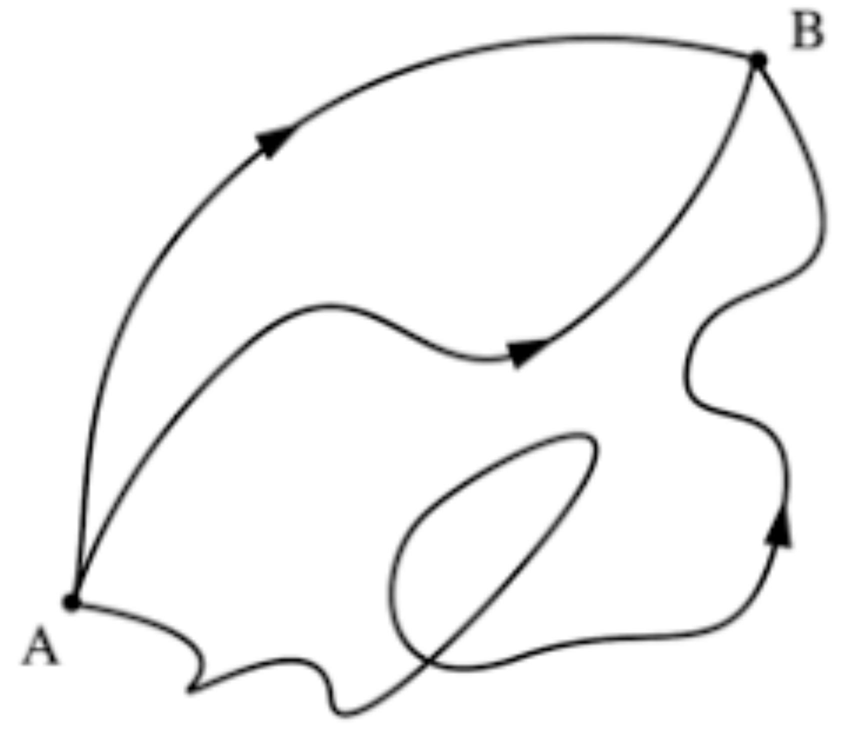
Dynamics systems



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

Classical and quantum control

Principle of least actions

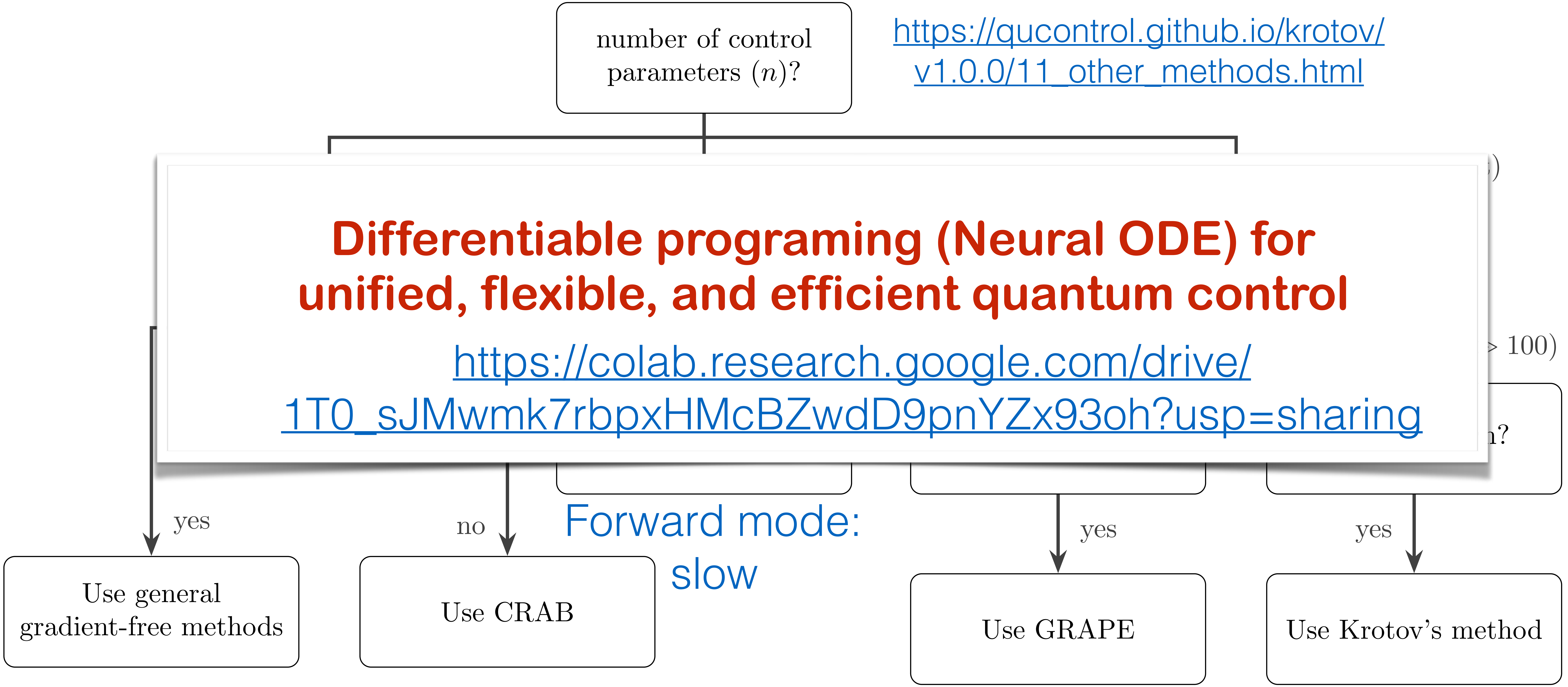


$$S = \int \mathscr{L}(q_\theta, \dot{q}_\theta, t)dt$$

Optics, (quantum) mechanics, field theory…

Quantum optimal control $i\frac{dU}{dt} = HU$

https://qucontrol.github.io/krotov/v1.0.0/11_other_methods.html



Differentiable programing (Neural ODE) for unified, flexible, and efficient quantum control

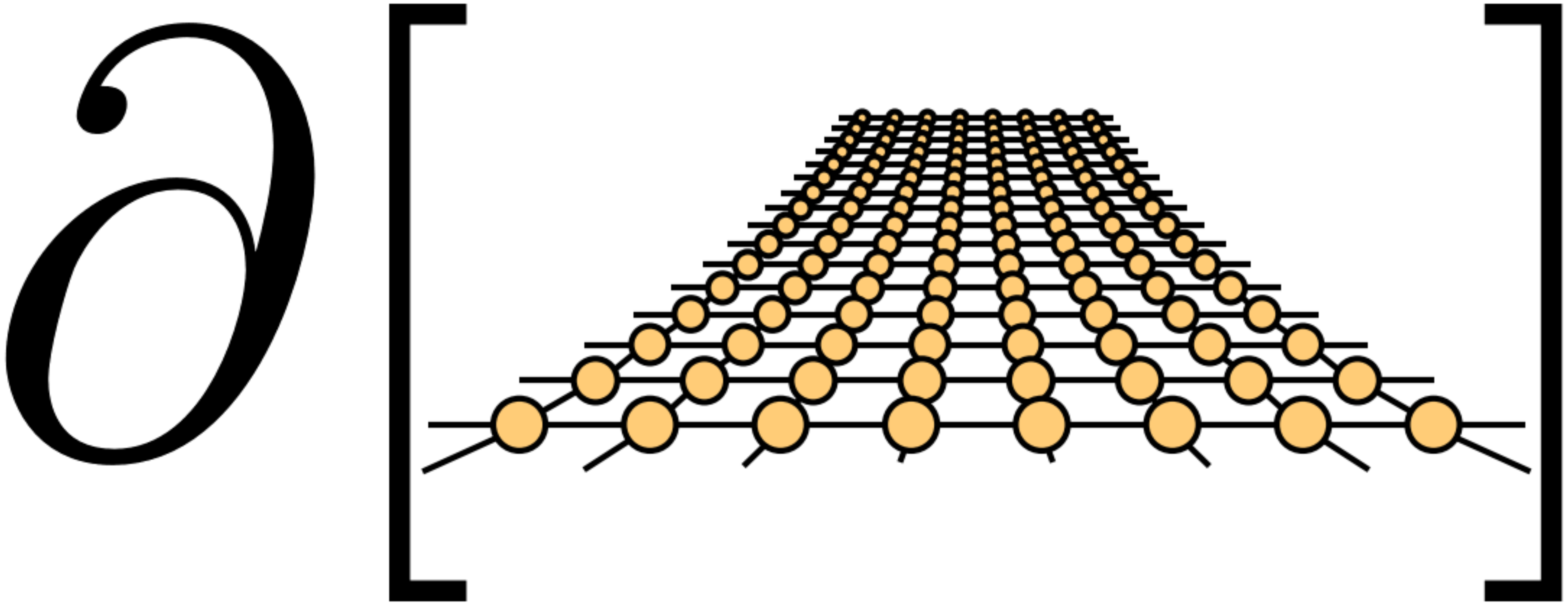
https://colab.research.google.com/drive/1T0_sJMwmk7rbpxHMcBZwdD9pnYZx93oh?usp=sharing

Forward mode: slow

No gradient: not scalable

Reverse mode w/ discretize steps: piesewise-constant assumption

Differentiable Programming Tensor Networks

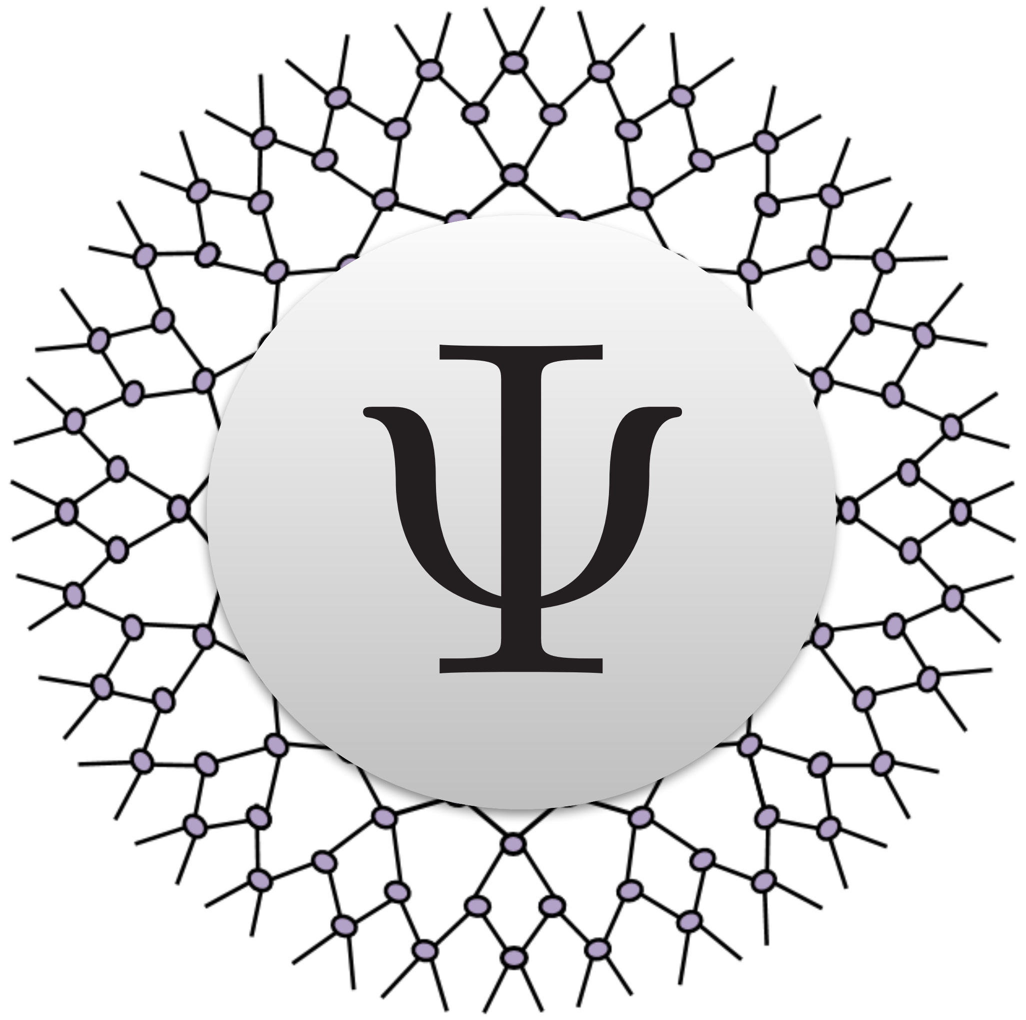


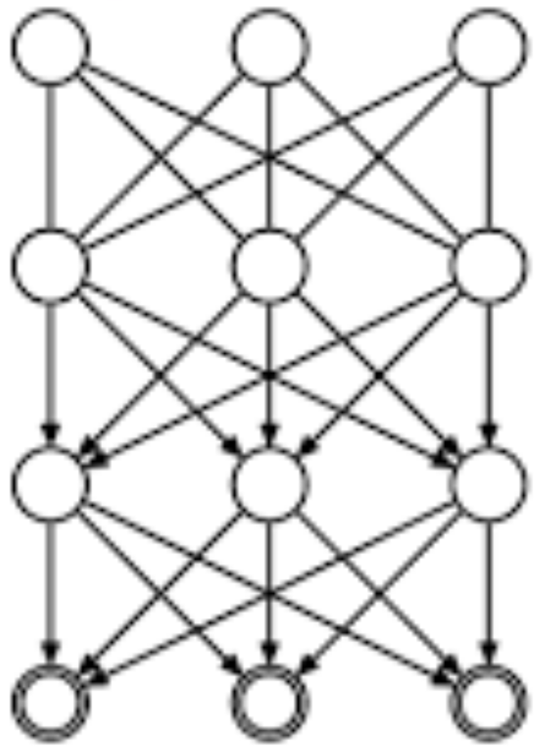
Liao, Liu, LW, Xiang, 1903.09650, PRX ‘19

https://github.com/wangleiphy/tensorgrad

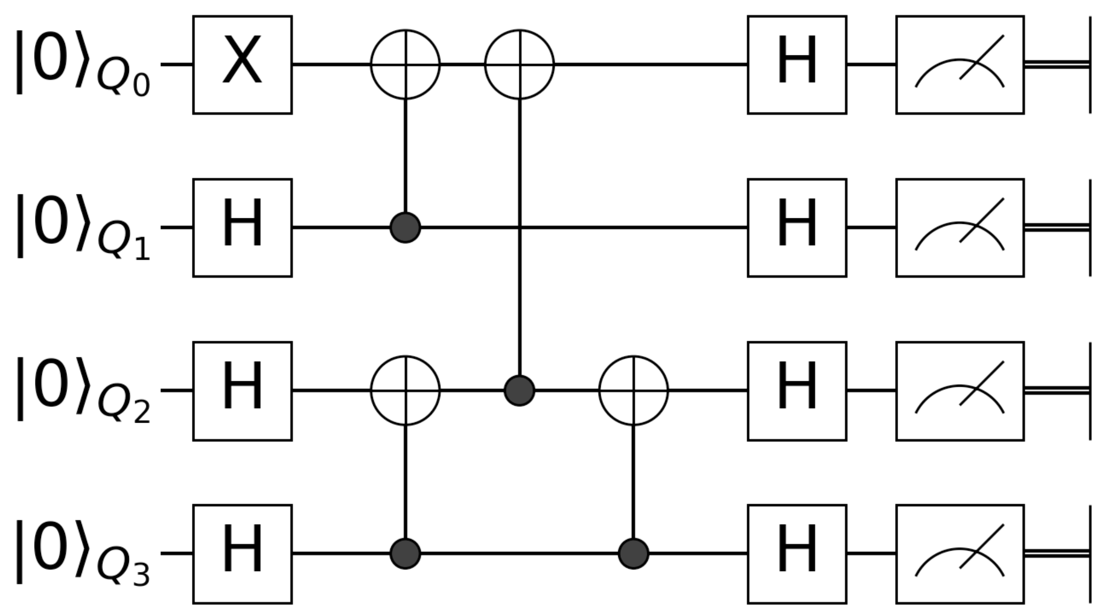
“Tensor network is 21 century’s matrix”

—Mario Szegedy





Neural networks and Probabilistic graphical models

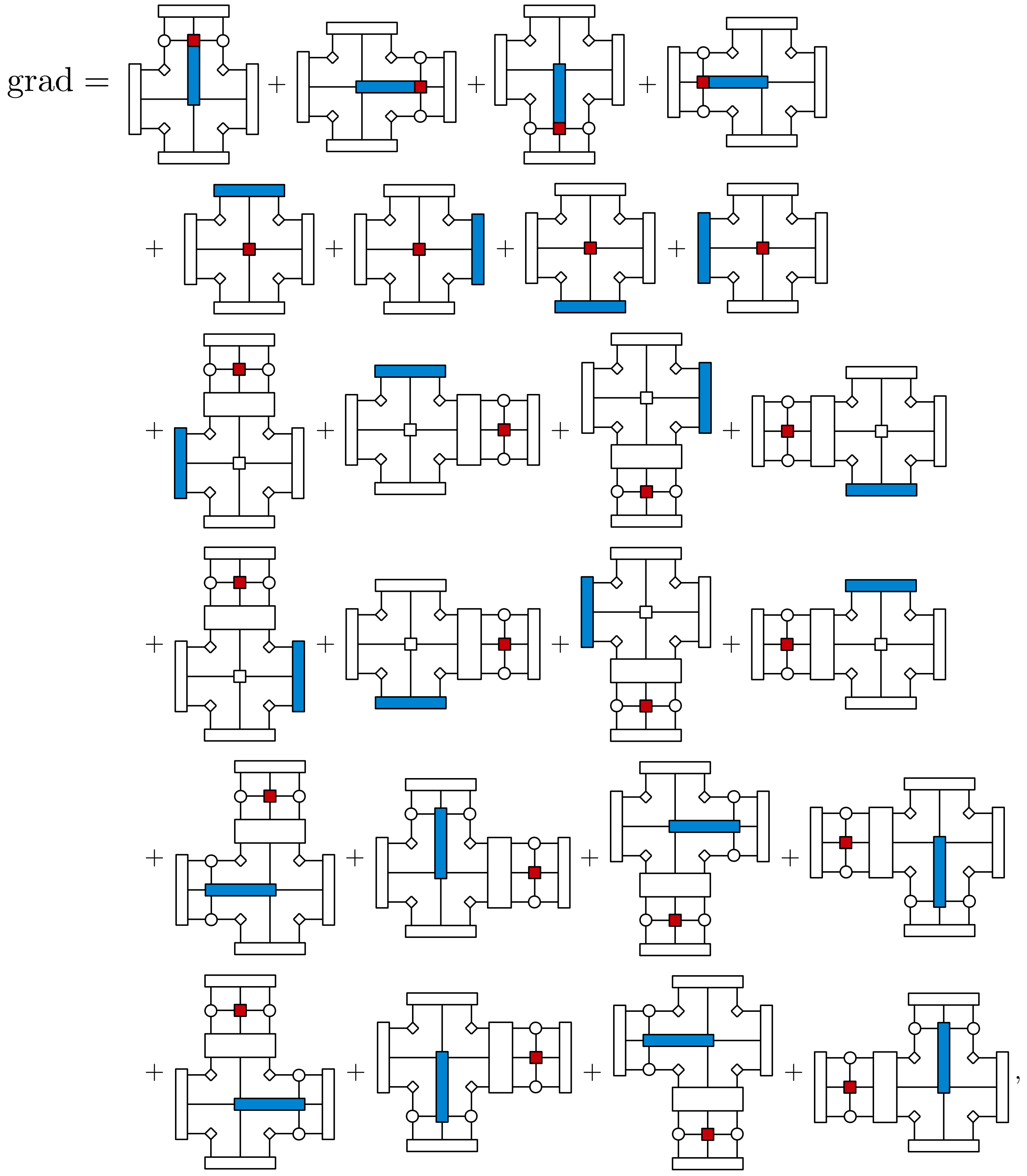


Quantum circuit architecture, parametrization, and simulation

Differentiable tensor network optimization

Finding ground state of a quantum magnet

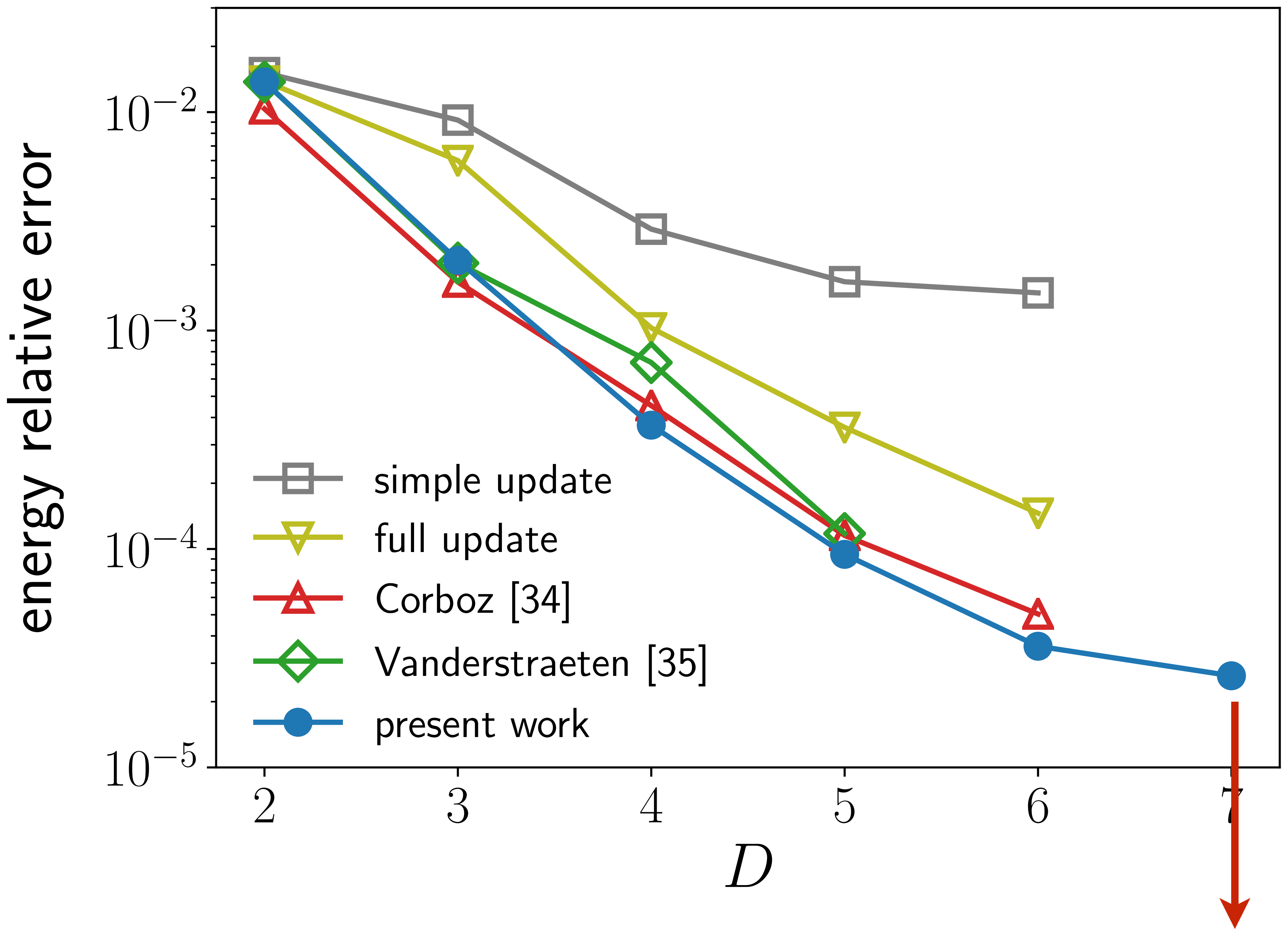
before…



Vanderstraeten et al, PRB '16

now, w/ differentiable programming

Liao, Liu, LW, Xiang, PRX '19

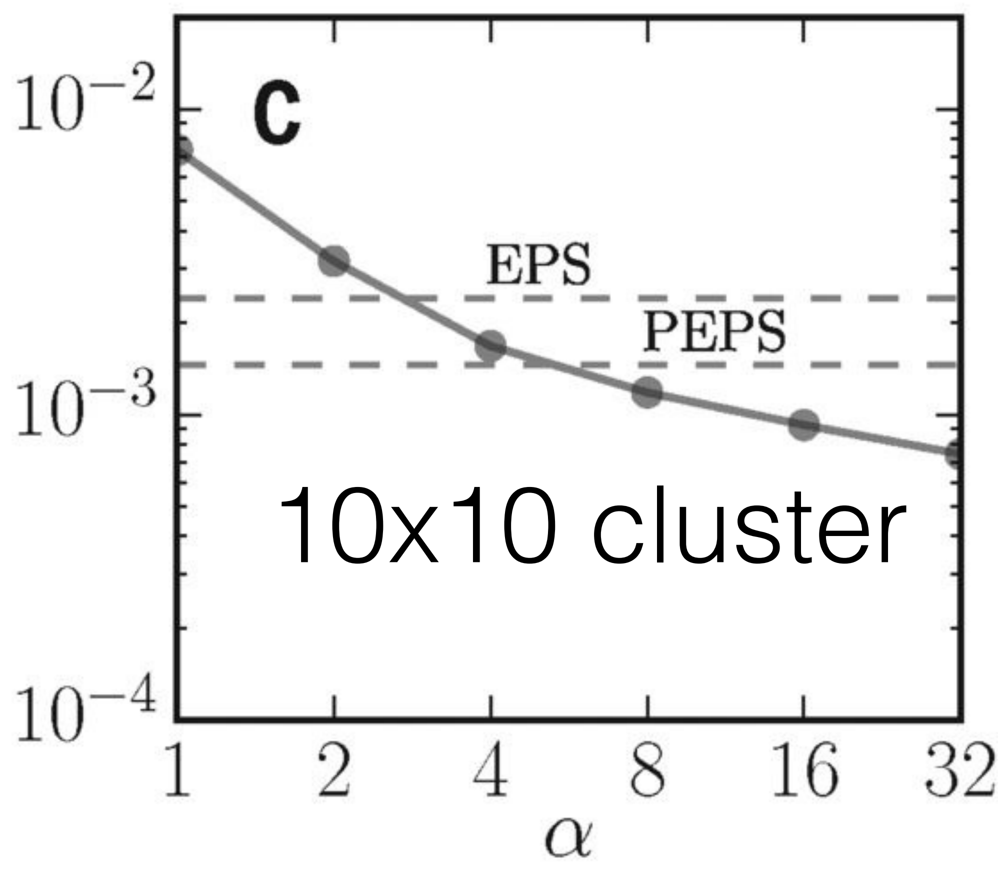


Lowest variational energy

https://github.com/wangleiphy/tensorgrad 1 GPU (Nvidia P100) week

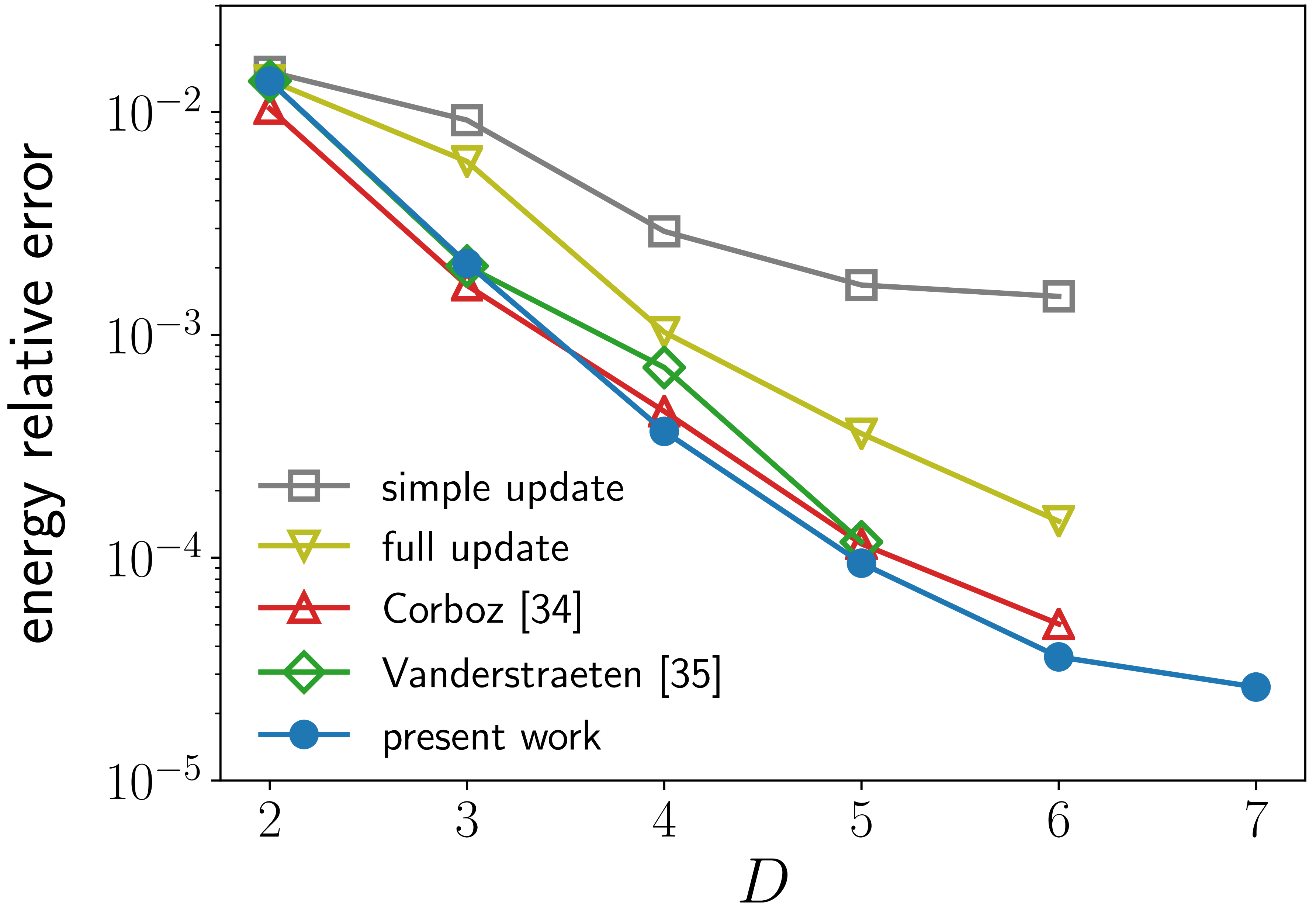
Differentiable tensor network optimization

Finite size
Neural network



Carleo & Troyer, Science '17

Infinite size
Tensor network



Liao, Liu, LW, Xiang, PRX '19

Further progress for challenging physical problems: frustrated magnets, fermions, thermodynamics …

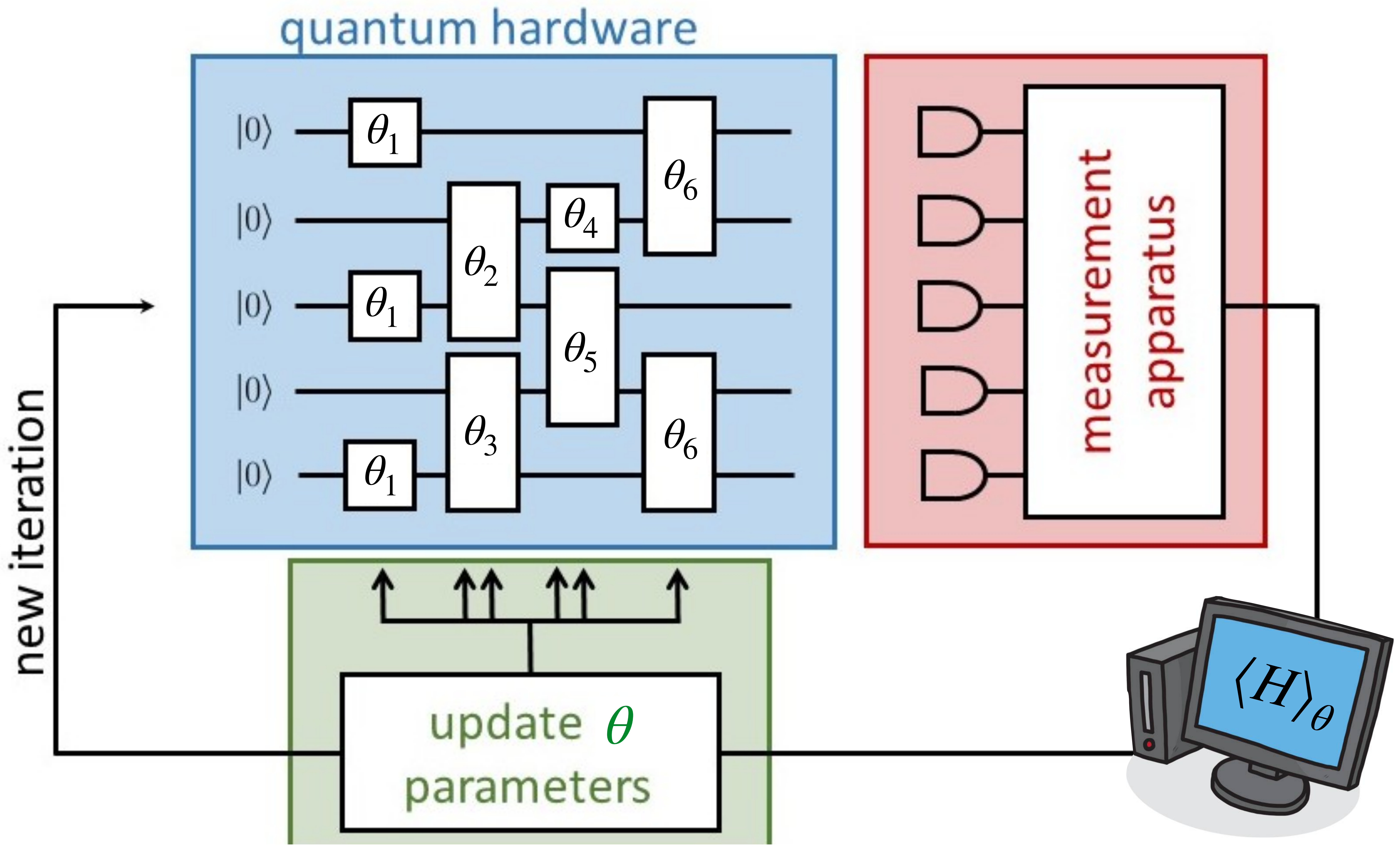
Chen et al, '19
Xie et al, '20
Tang et al '20
…

Differentiable Programming Quantum Circuits

Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design

Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, Lei Wang, 1912.10877, Quantum ‘20

Variational quantum algorithms

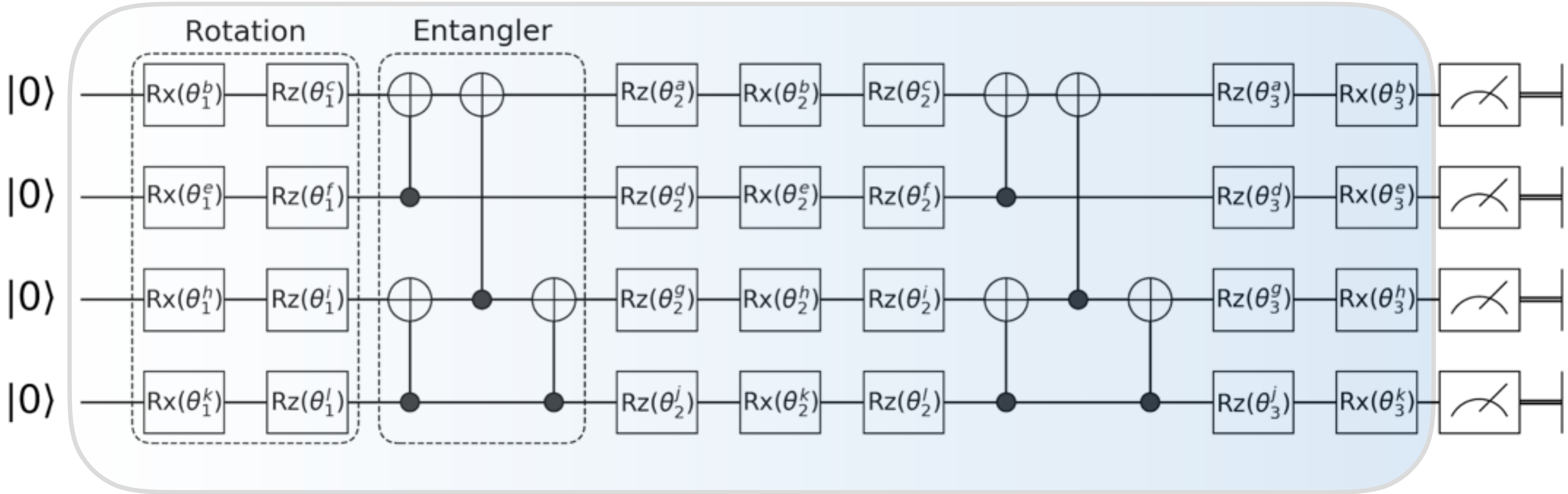


Peruzzo et al,
Nat. Comm. '13

Quantum circuit as a variational ansatz or a machine learning model

Differentiable quantum circuits

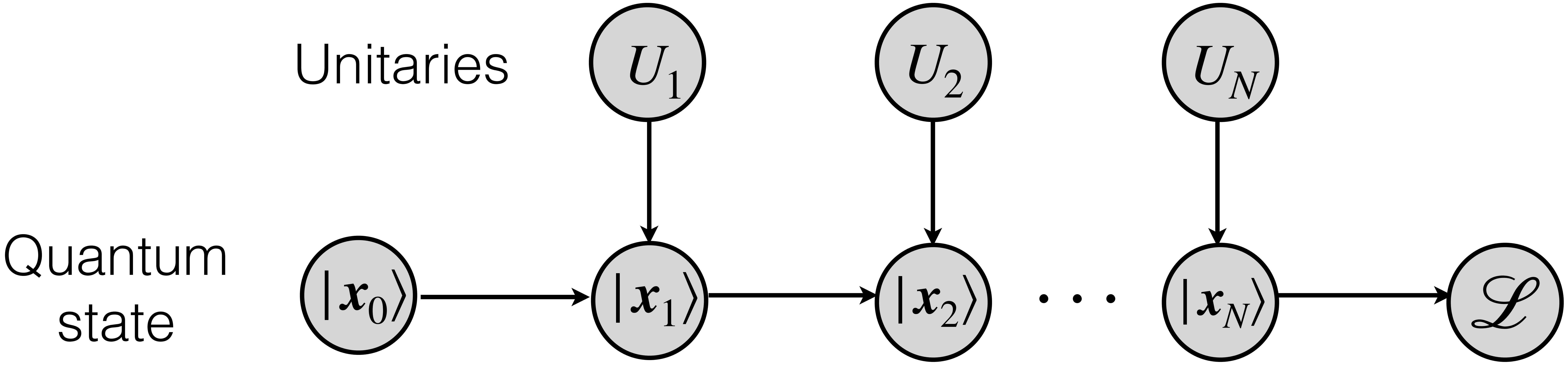
compute gradient in classical simulations



PENNYLANE   TensorFlow Quantum

Unfortunately, forward mode is slow
Reverse mode is memory consuming

Quantum circuit computation graph



The same "comb graph" as the feedforward neural network, except that quantum computing is reversible

O(1) memory AD for reversible neural nets Gomez et al, 1707.04585 Chen et al, 1806.07366

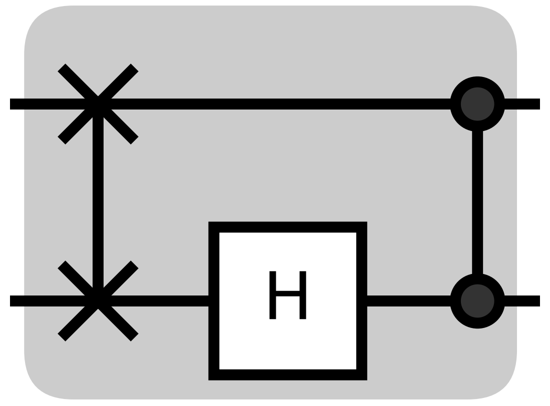
```julia
julia> using Yao, YaoExtensions

julia> n = 10; depth = 10000;

julia> circuit = dispatch!(
    variational_circuit(n, depth),
    :random);

julia> gatecount(circuit)
Dict{Type{#s54} where #s54 <:
    AbstractBlock,Int64} with 3 entries:
  RotationGate{1,Float64,ZGate} => 200000
  RotationGate{1,Float64,XGate} => 100010
  ControlBlock{10,XGate,1,1}    => 100000

julia> nparameters(circuit)
300010

julia> h = heisenberg(n);

julia> for i = 1:100
    _, grad = expect'(h, zero_state(n)=>
                            circuit)
    dispatch!(-, circuit, 1e-3 * grad)
    println("Step $i, energy = $(expect(
            h, zero_state(n)=>circuit))")
        end
```

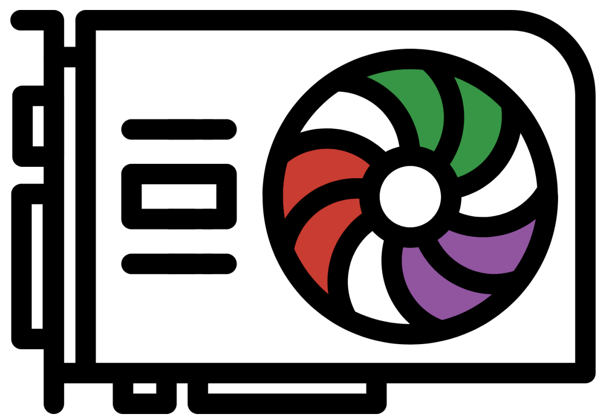
Train a 10,000 layer, 300,000 parameter circuit on a laptop

https://yaoquantum.org/

Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design

https://yaoquantum.org/

julia + 

+  =

Xiu-Zhe Roger Luo (IOP, CAS → Waterloo & PI)
Jin-Guo Liu (IOP, CAS → Harvard & QuEra)

Features:

- Differentiable programming quantum circuits
- Batch parallelization with GPU acceleration
- Quantum block intermediate representation

Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design

https://yaoquantum.org/

Announcing the Winner of the 2020 Wittek Quantum Prize for Open Source Software

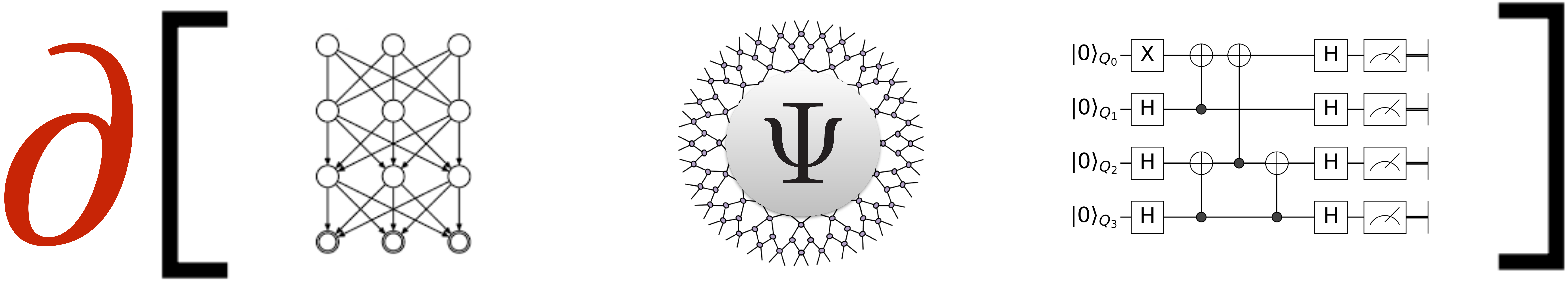
Tomas Babej Following
Feb 1 · 3 min read

Roger Luo is the 2020 Winner of the Wittek Quantum Prize for Open Source Software for his work on Yao.jl and several other projects. The prize has been awarded by the Quantum Open Source Foundation (QOSF) in collaboration with Unitary Fund, reviewing over 50 candidatures from a worldwide community.

PI)

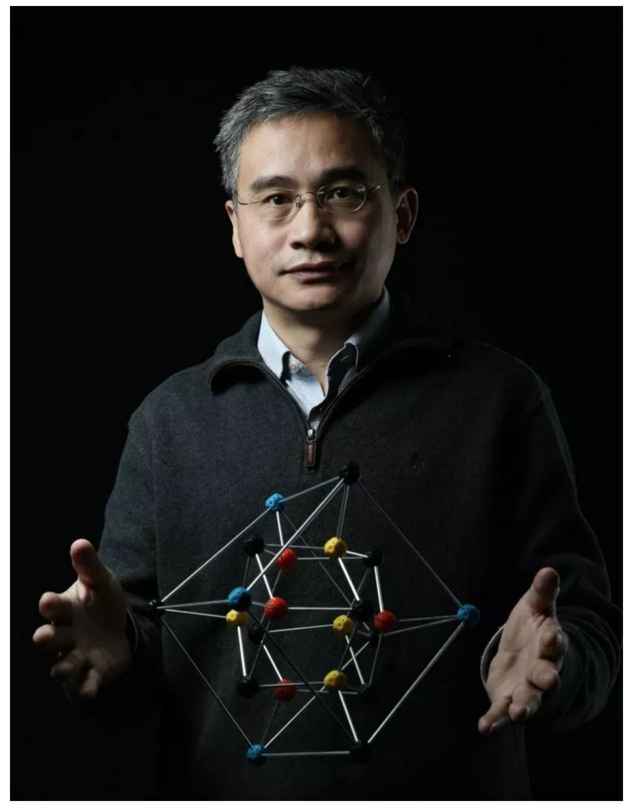
Features

- Differentiable programming quantum circuits
- Batch parallelization with GPU acceleration
- Quantum block intermediate representation

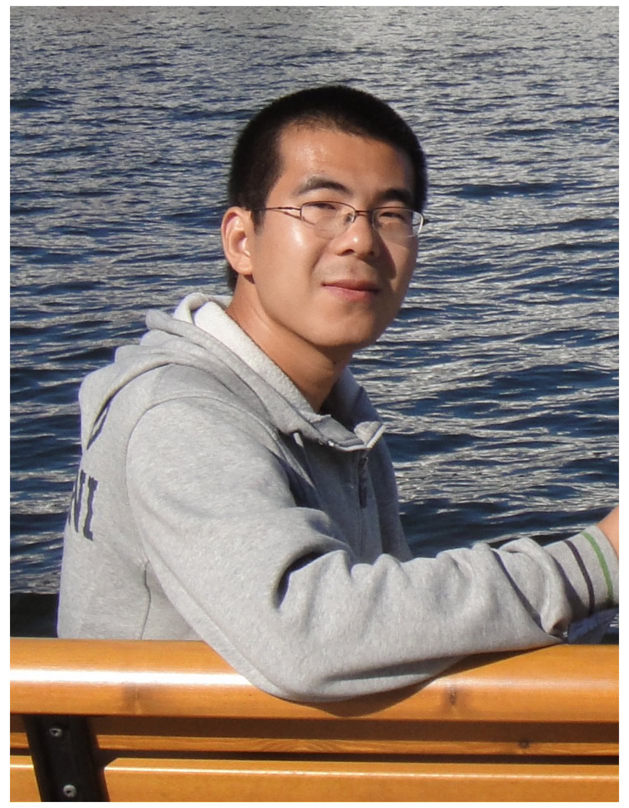


Thank you!

Hai-Jun Liao
IOP CAS



Tao Xiang
IOP CAS



Pan Zhang
ITP CAS

Jin-Guo Liu,
QuEra & Harvard

Xiu-Zhe Luo
Waterloo & PI