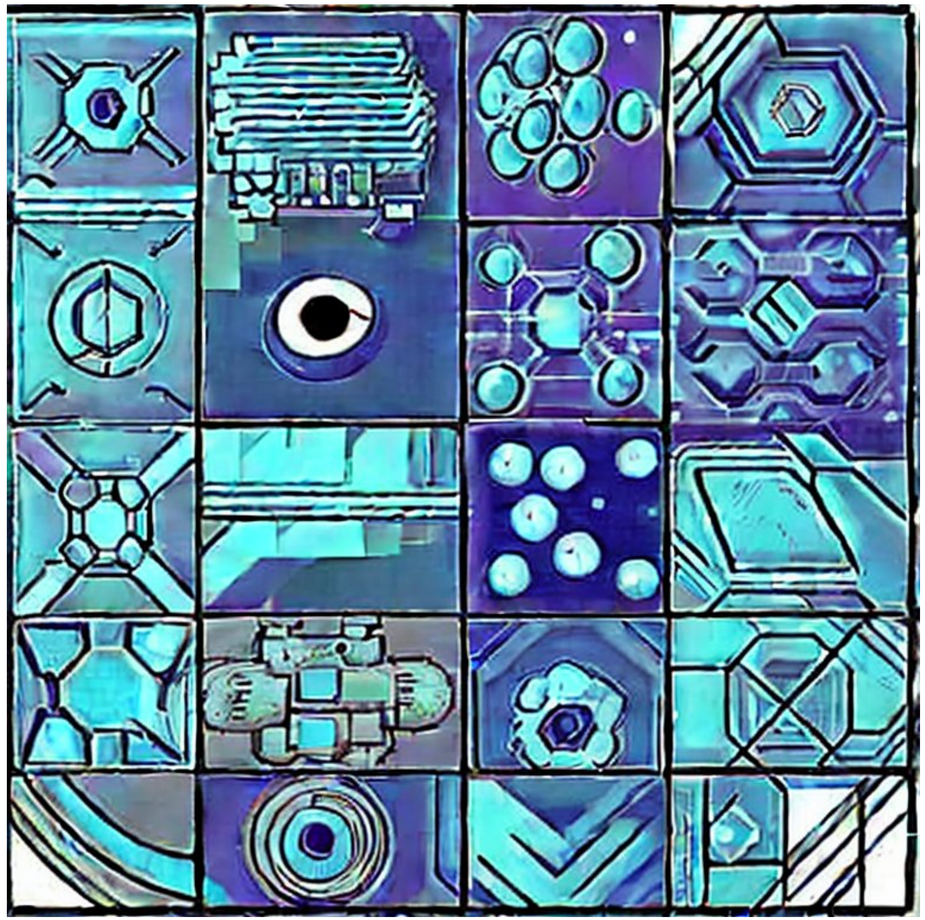


Machine learning for physicists

https://github.com/wangleiphy/ml4p

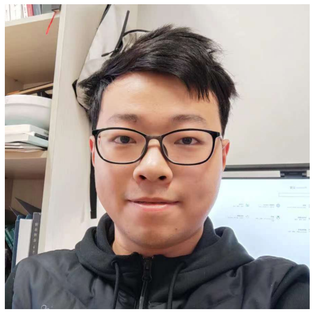
每周四上午10点

2.23	Overview
3.2	Machine learning practices
3.9	A hitchhiker’s guide to deep learning
3.16	Research projects hands-on
3.23	Symmetries in machine learning
3.30	Differentiable programming
4.6	Generative models-I
4.13	Generative models-II
4.20	Research projects presentation
4.27	AI for science: why now ?

课程微信群

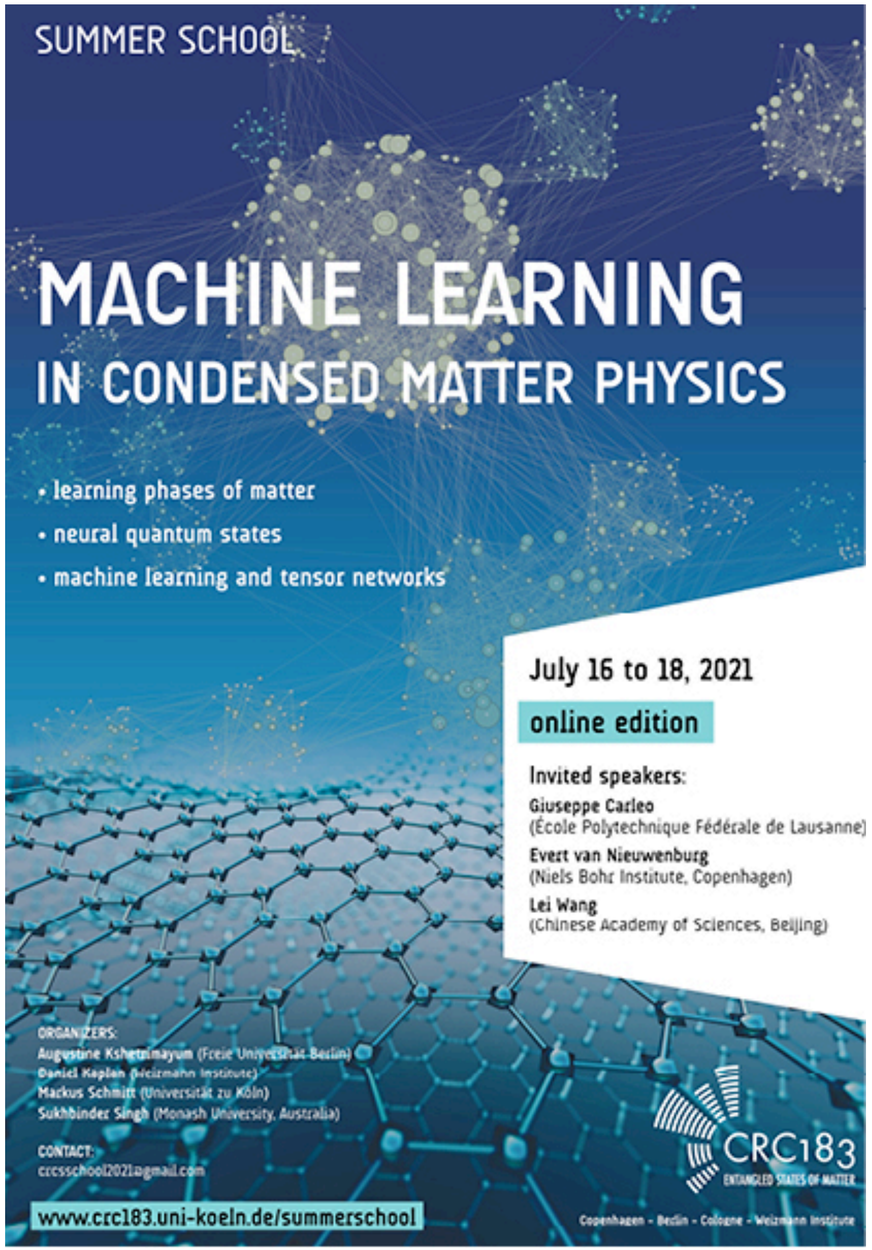
助教

李子航



李扬帆

考核方式： project + presentation （1学分）



Lei Wang 1.5h x 3

1. Scientific machine learning with and without data (overview talk)

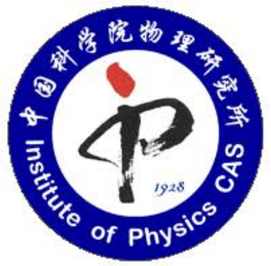
2. Generative models (mostly normalizing flows)

3. Differentiable programming (for tensor networks and quantum circuits)

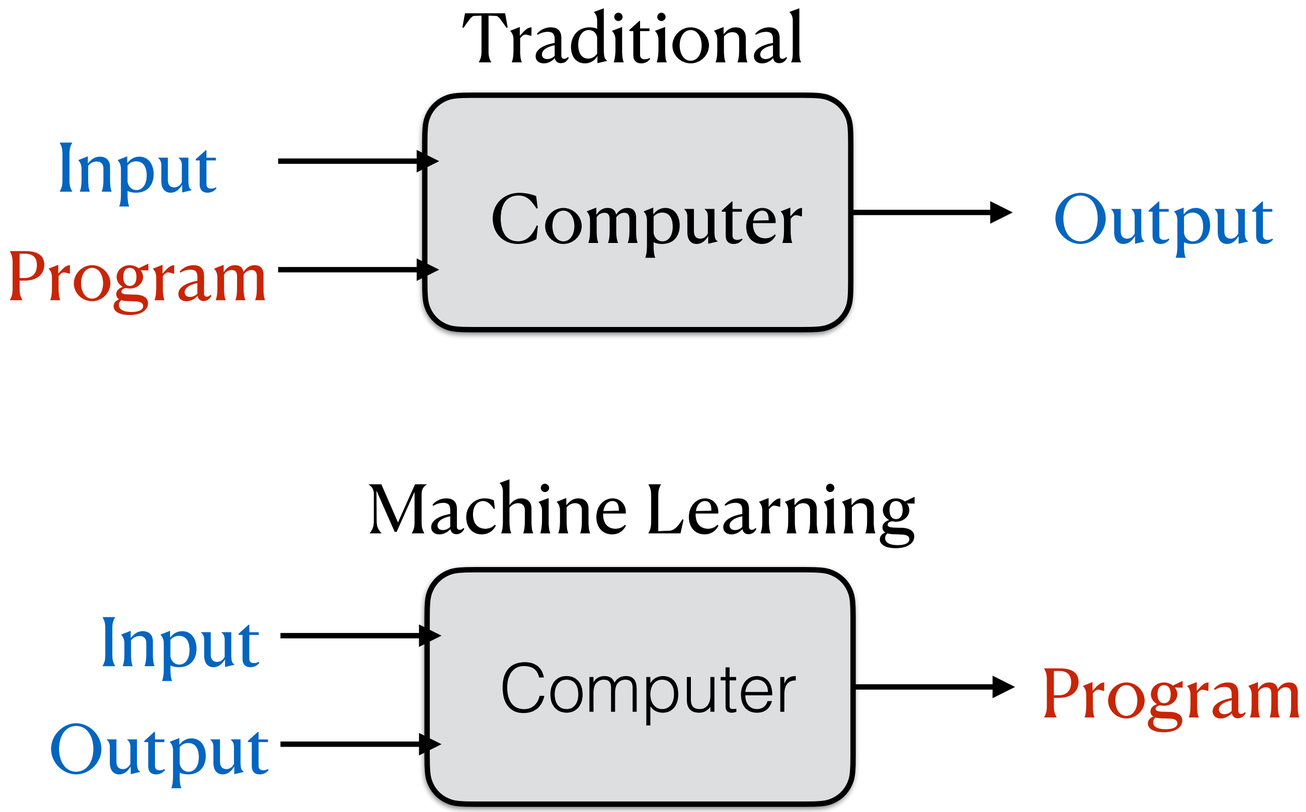
Recordings https://www.crc183.uni-koeln.de/summer-school-machine-learning/

Differentiable programming

Lei Wang (王磊)
https://wangleiphy.github.io
Institute of Physics, CAS



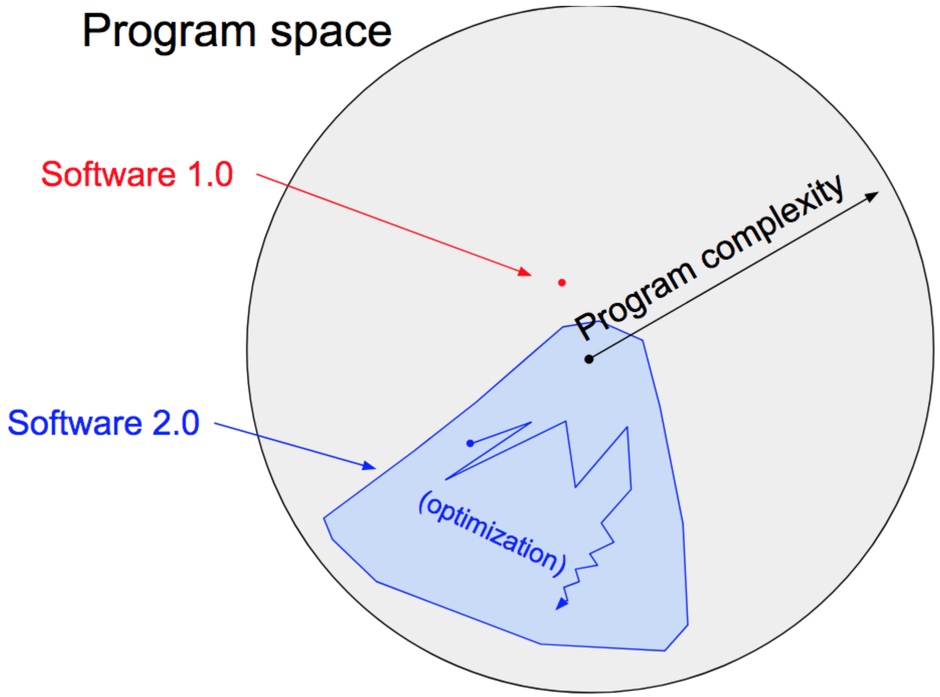
Differentiable Programming



Andrej Karpathy

OpenAI, Previously director of AI at Tesla, 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 gradient search in the program space

Differentiable Programming

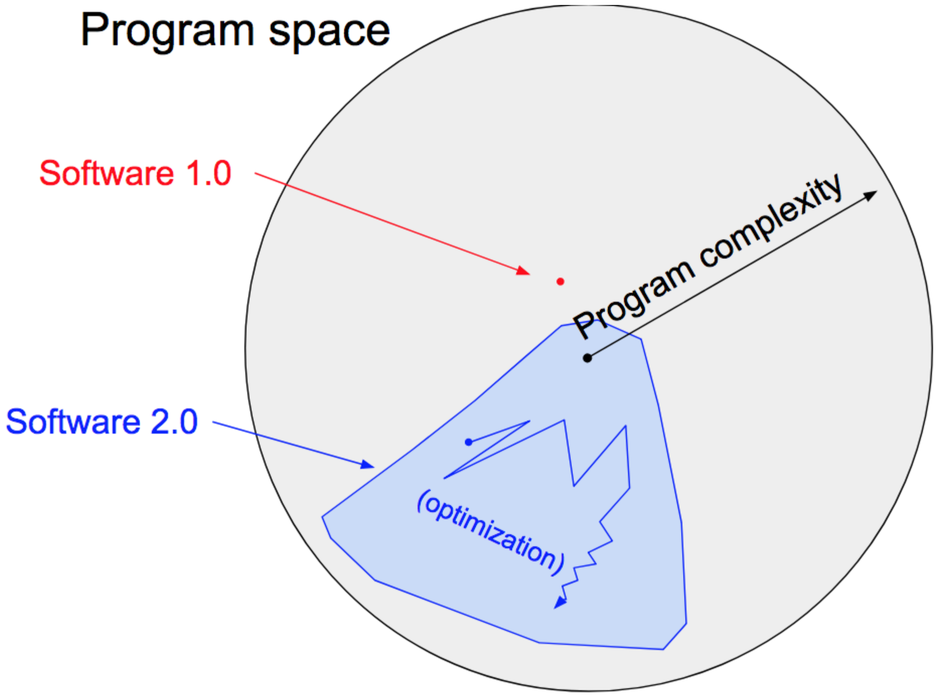
Benefits of Software 2.0

- 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

OpenAI, Previously director of AI at Tesla, 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

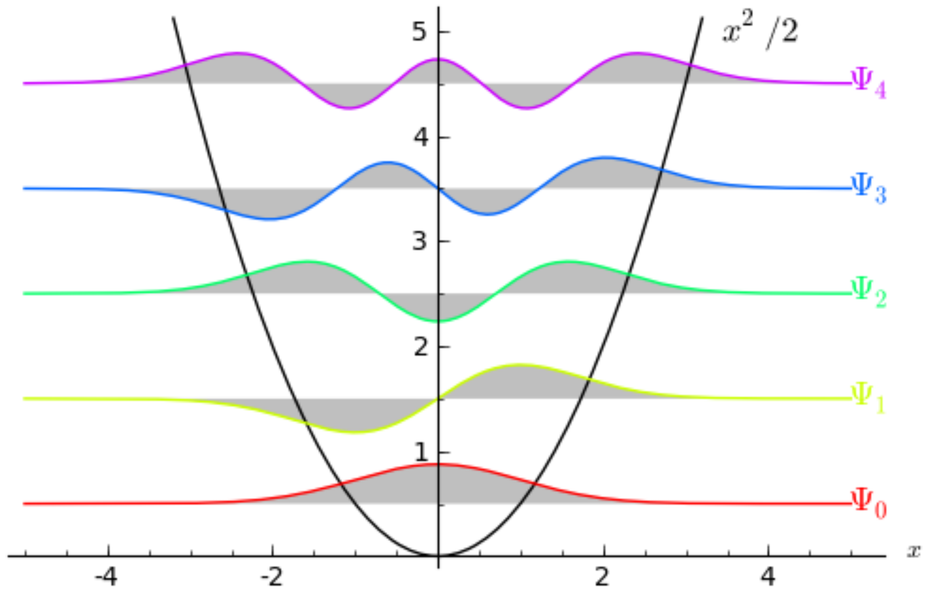


Writing software 2.0 by gradient search in the program space

Demo: Inverse Schrodinger Problem

Given ground state density, how to design the potential ?

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

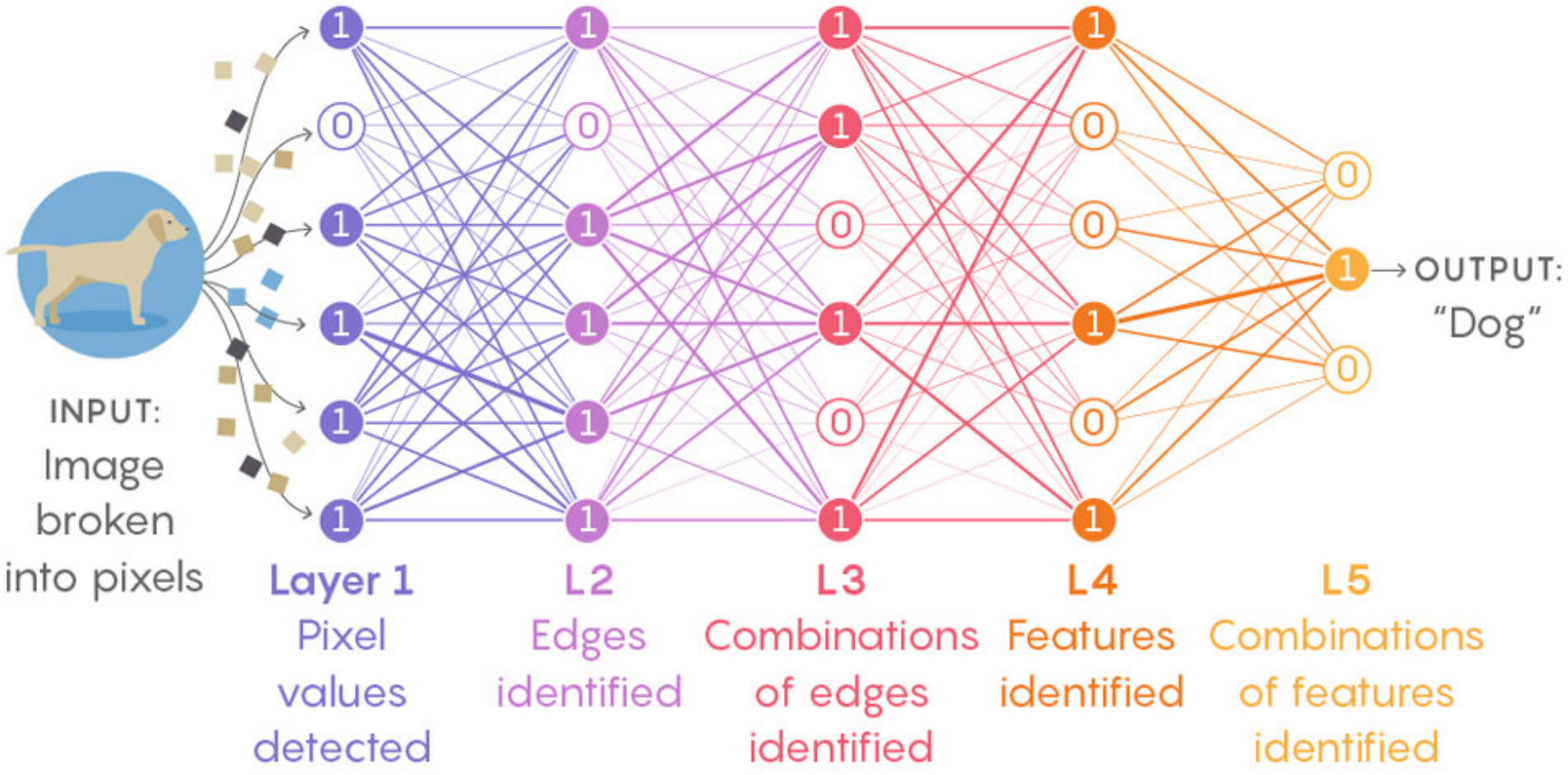


https://math.mit.edu/~stevenj/18.336/adjoint.pdf

https://github.com/wangleiphy/ml4p/blob/main/5_diffprogram/inverse_schrodinger.ipynb

What is under the hood ?

What is deep learning ?

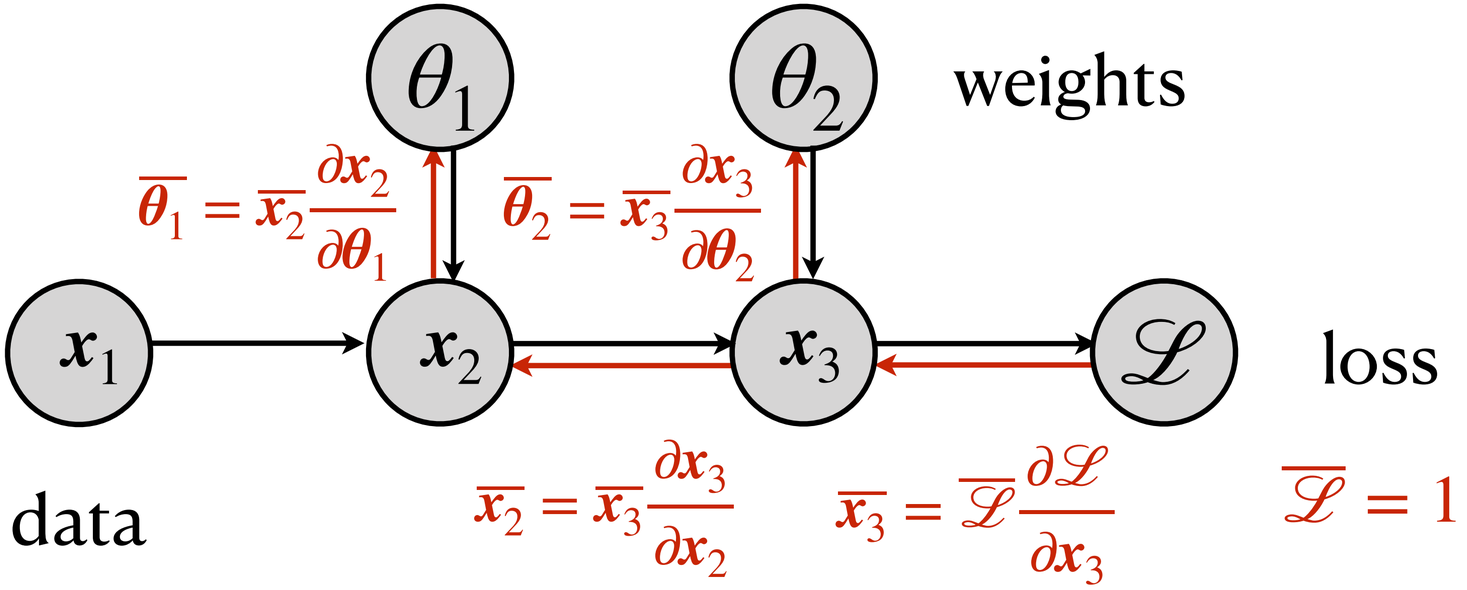


Composes differentiable components to a program

e.g. a neural network, then optimizes it with gradients

Automatic differentiation on computation graph

"comb graph"

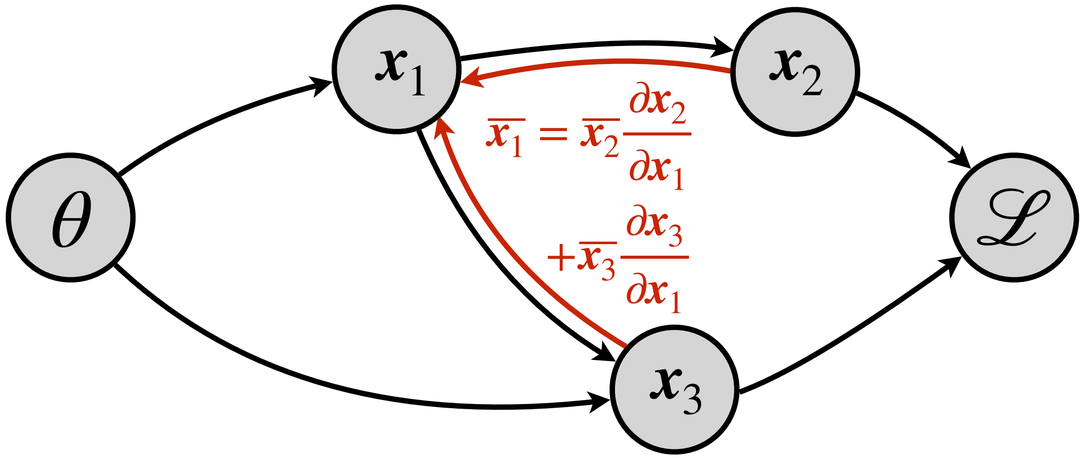


"adjoint variable" $\overline{x} = \dfrac{\partial \mathscr{L}}{\partial x}$

Pullback the adjoint through the graph

Automatic differentiation on computation graph

directed
acyclic graph

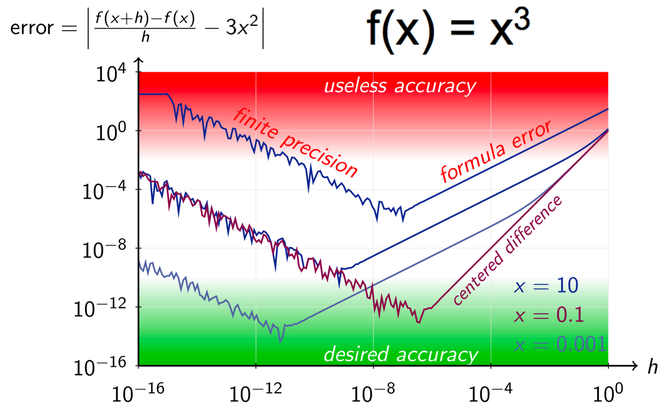


$$\overline{x_i} = \sum_{j\text{: child of } i} \overline{x_j} \frac{\partial x_j}{\partial x_i} \qquad \text{with} \quad \overline{\mathscr{L}} = 1$$

Message passing for the adjoint at each node

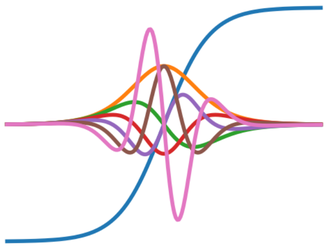
Advantages of automatic differentiation

- Accurate to the machine precision



- Same computational complexity as the function evaluation:
  Baur-Strassen theorem ’83

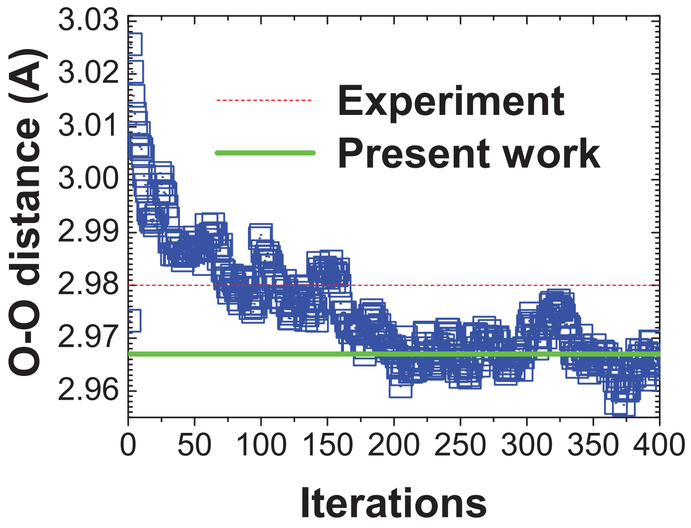
- Supports higher order gradients



```
>>> 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()
```

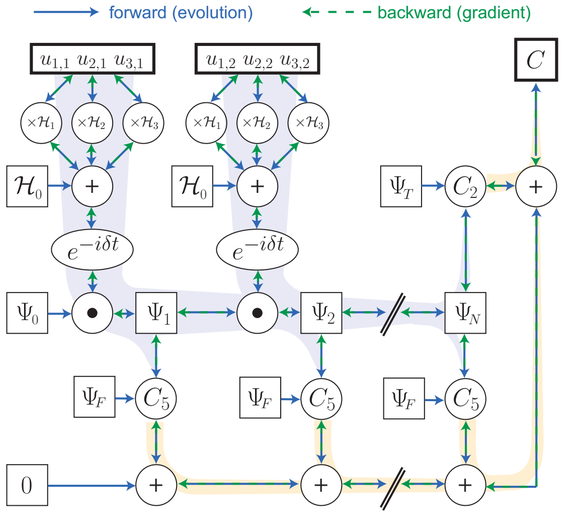
Applications of AD

Computing force



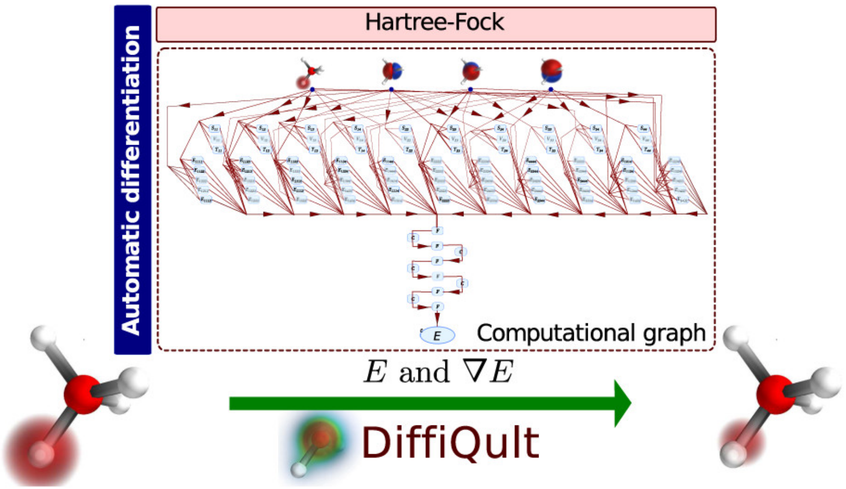
Sorella and Capriotti
J. Chem. Phys. '10

Quantum optimal control



Leung et al
PRA '17

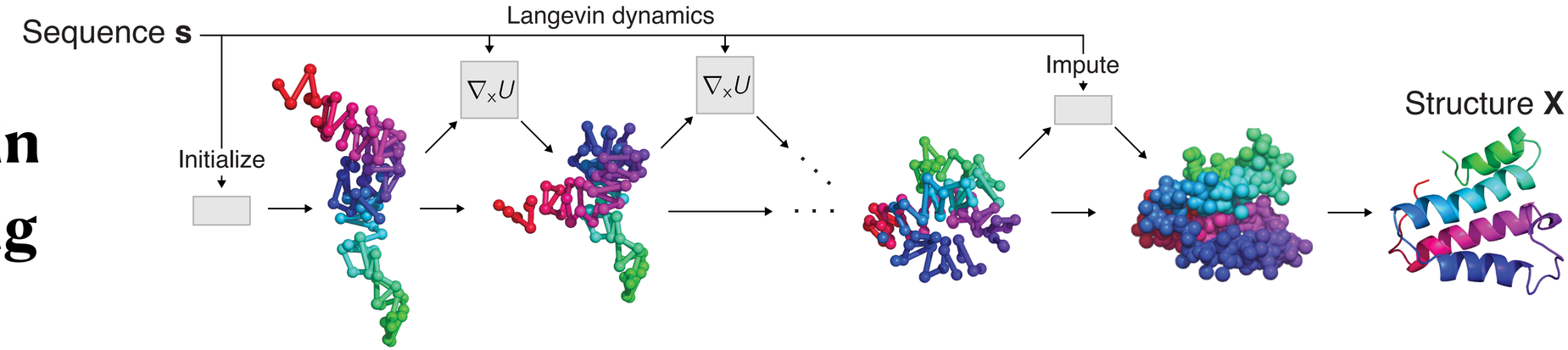
Variational Hartree-Fock



Tamayo-Mendoza et al
ACS Cent. Sci. '18

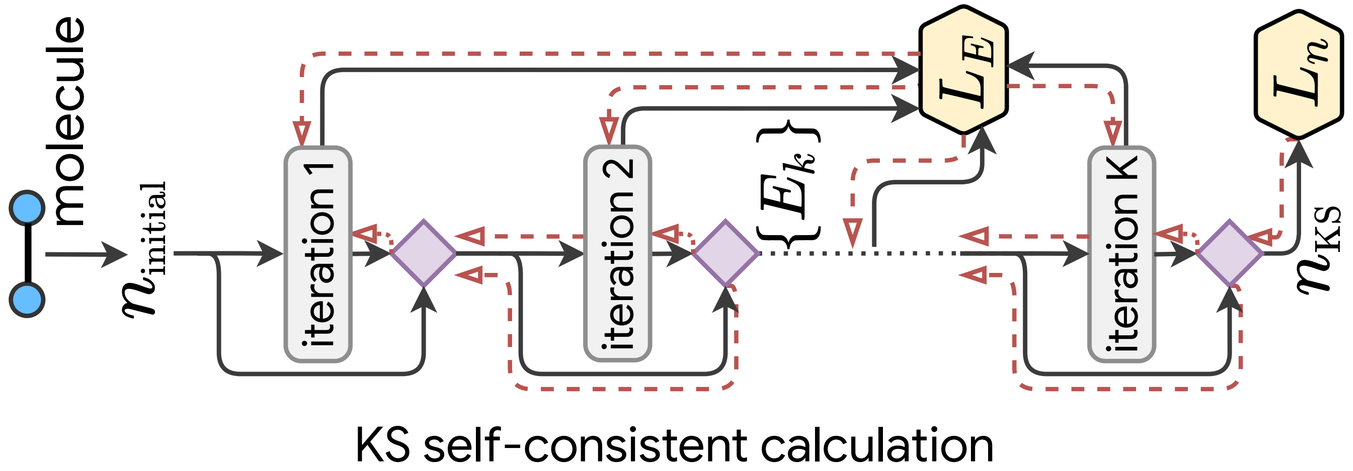
More Applications...

Protein folding



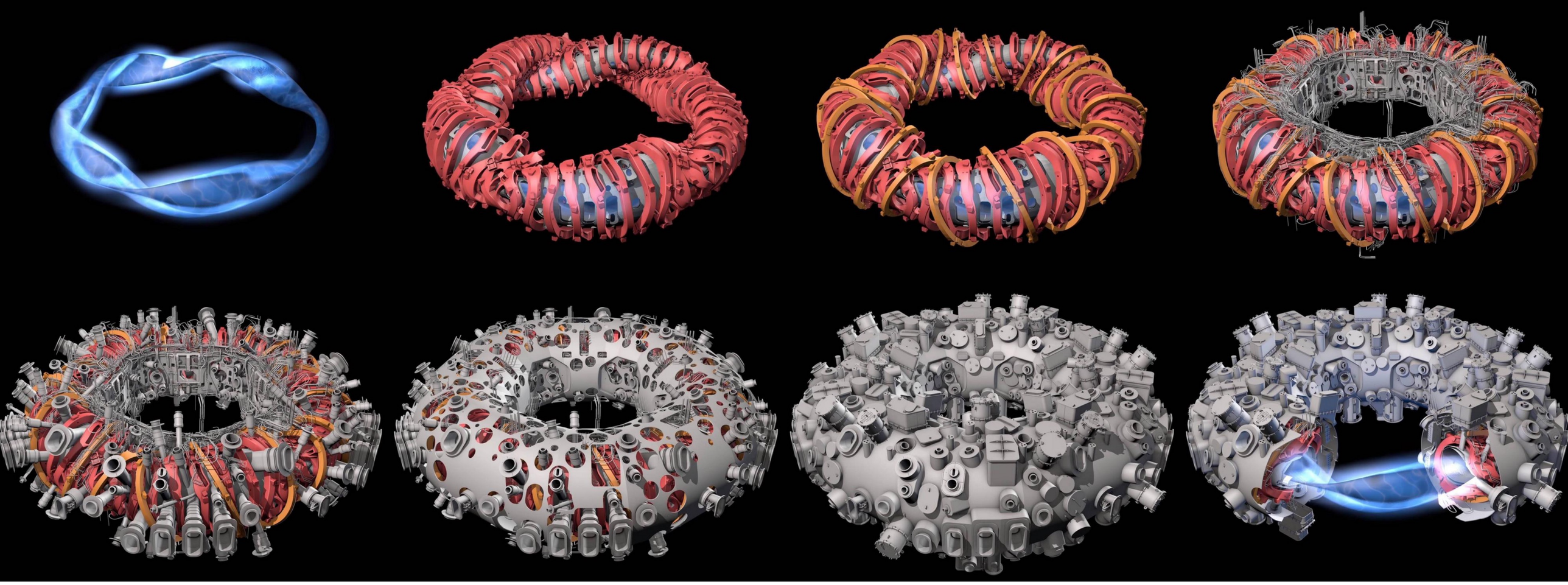
Ingraham et al
ICLR '19

Learning density functionals



Li et al
PRL '21

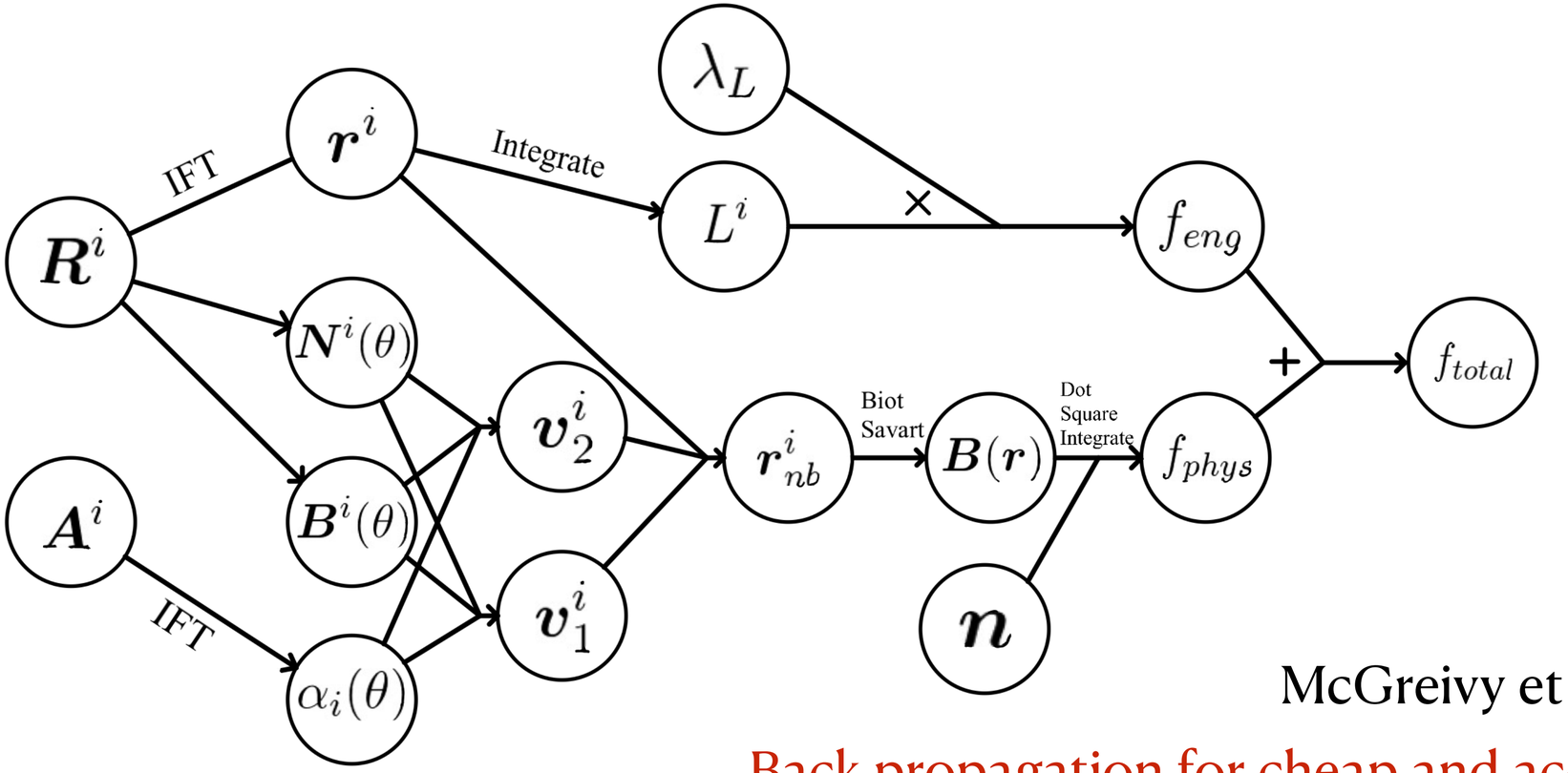
Kasim, Vinko, 2102.04229
Dick et al, 2106.04481



Coil design in fusion reactors (stellarator)

Differentiable stellarator design

Coil parameters



Total cost

McGreivy et al 2009.00196

Back propagation for cheap and accurate gradient

Differentiable programming is broader than training neural networks

微分万物:深度学习的启示*

王 磊[1,2,†] 刘金国[3]

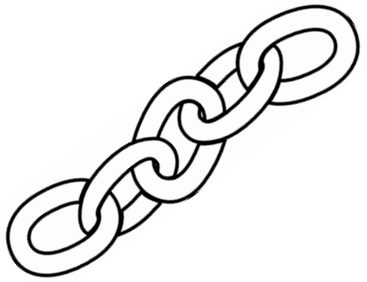
(1 中国科学院物理研究所 北京 100190)

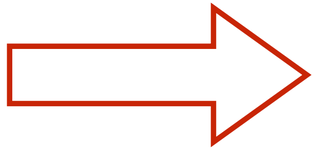
(2 松山湖材料实验室 东莞 523808)

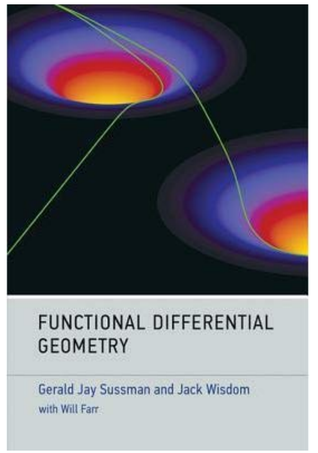
(3 哈佛大学物理系 剑桥 02138)

《物理》

2021年2月







Black magic box

Chain rule

≈

Functional differential geometry

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

Differentiating a general computer program (rather than neural networks) calls for deeper understanding of the technique

Reverse versus forward mode

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

Reverse mode AD: Vector-Jacobian Product of primitives

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

Backpropagation = Reverse mode AD applied to neural networks

Reverse versus forward mode

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

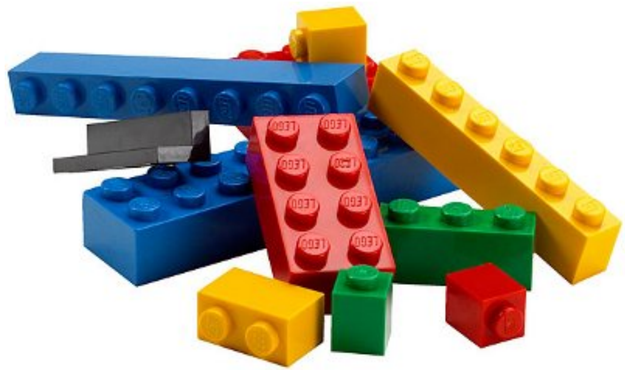
Forward mode AD: Jacobian-Vector Product of primitives

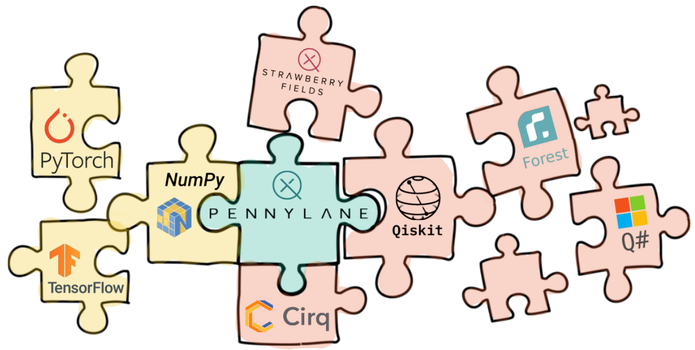
- Same order with the function evaluation
- No storage overhead
- Efficient for graph with large fan-out

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

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 external libraries written agnostically to AD
    (or, even a quantum processor)





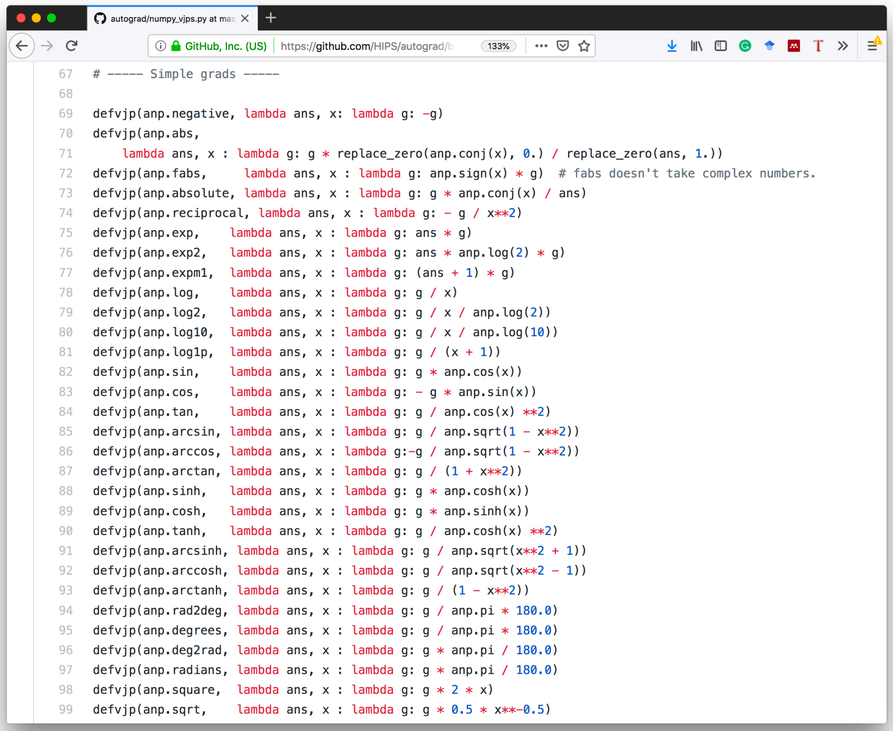
https://github.com/PennyLaneAI/pennylane

Example of primitives

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

https://github.com/HIPS/antograd/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,




```
# ----- 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/

An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiation

https://people.maths.ox.ac.uk/gilesm/files/NA-08-01.pdf

Mike Giles
Oxford University Computing Laboratory, Parks Road, Oxford, U.K.

Addition Multiplication Inverse Norm Determinant

Matrix polynomial Matrix exponential Eigensovler Cholesky SVD

AD for complex-valued SVD, 1909.02659 AD for dominant eigensolver, 2001.04121 AD of QR, 2009.10071

amazon Auto-Differentiating Linear Algebra, 1710.08717 Stable higher-order AD for Log-Det, 1909.02487

Differentiable programming tools

HIPS/autograd

Differentiation tensor operations on hardware accelerators

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

matrix

diagonalization

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

Differentiable Eigensolver

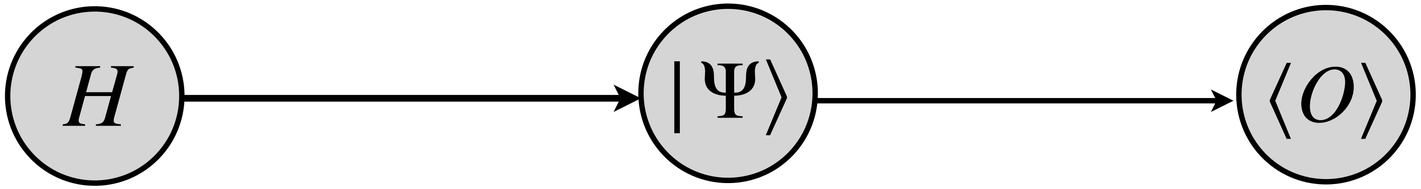
$$H\Psi = \Psi E$$

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



Automatic differentiation

where $F_{i,j} = (d_j - d_i)^{-1}$ for $i \neq j$, and zero otherwise. Hence, the forward mode sensitivity equations are

$$\begin{aligned} \dot{D} &= I \circ (U^{-1}\dot{A}\,U), \\ \dot{U} &= U\left(F \circ (U^{-1}\dot{A}\,U)\right). \end{aligned}$$

In reverse mode, using the identity $\mathrm{Tr}(A\,(B \circ C)) = \mathrm{Tr}((A \circ B^T)\,C)$, we get

$$\begin{aligned} \mathrm{Tr}\left(\overline{D}^T \mathrm{d}D + \overline{U}^T \mathrm{d}U\right) &= \mathrm{Tr}\left(\overline{D}^T U^{-1}\mathrm{d}A\,U\right) + \mathrm{Tr}\left(\overline{U}^T U\left(F \circ (U^{-1}\mathrm{d}A\,U)\right)\right) \\ &= \mathrm{Tr}\left(\overline{D}^T U^{-1}\mathrm{d}A\,U\right) + \mathrm{Tr}\left(\left((\overline{U}^T U) \circ F^T\right)\,U^{-1}\mathrm{d}A\,U\right) \\ &= \mathrm{Tr}\left(U\left(\overline{D}^T + (\overline{U}^T U) \circ F^T\right)\,U^{-1}\mathrm{d}A\right) \end{aligned}$$

and so

$$\overline{A} = U^{-T}\left(\overline{D} + F \circ (U^T\,\overline{U})\right)\,U^T.$$

https://people.maths.ox.ac.uk/gilesm/files/NA-08-01.pdf

same thing

Linear response theory

In an independent-particle approximation the states are determined by the hamiltonian $\hat{H}_{\mathrm{eff}}$ in the effective Schrödinger equation (3.36). The change in the individual independent-particle orbitals, $\Delta\psi_i(\mathbf{r})$ to first order in perturbation theory, can be written in terms of a sum over the spectrum of the unperturbed hamiltonian $\hat{H}^0_{\mathrm{eff}}$ as [11, 265, 266],

$$\Delta\psi_i(\mathbf{r}) = \sum_{j\neq i} \psi_j(\mathbf{r}) \frac{\langle\psi_j|\Delta\hat{H}_{\mathrm{eff}}|\psi_i\rangle}{\varepsilon_i - \varepsilon_j}, \tag{3.61}$$

where the sum is over all the states of the system, occupied and empty, with the exception of the state being considered. Similarly, the change in the expectation value of an operator $\hat{O}$ in the perturbed ground state to lowest order in $\Delta\hat{H}_{\mathrm{eff}}$ can be written

$$\begin{aligned} \Delta\langle\hat{O}\rangle &= \sum_{i=1}^{\mathrm{occ}} \langle\psi_i + \delta\psi_i|\hat{O}|\psi_i + \delta\psi_i\rangle \\ &= \sum_{i=1}^{\mathrm{occ}} \sum_{j}^{\mathrm{empty}} \frac{\langle\psi_i|\hat{O}|\psi_j\rangle\langle\psi_j|\Delta\hat{H}_{\mathrm{eff}}|\psi_i\rangle}{\varepsilon_i - \varepsilon_j} + \mathrm{c.c.} \end{aligned} \tag{3.62}$$

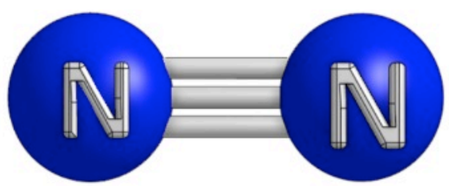
In (3.62) the sum over j is restricted to conduction states only, which follows from the fact that the contributions of pairs of occupied states i, j and j, i cancel in (3.62) (Exercise 3.21). Expressions Eqs. (3.61) and (3.62) are the basic equations upon which is built the theory of response functions (App. D) and methods for calculating static (Ch. 19) and dynamic responses (Ch. 20) in materials.

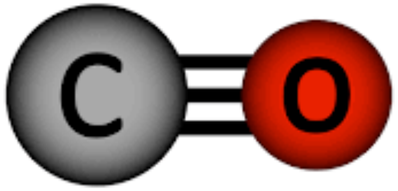
Richard Martin, Electronic structure

Differentiable Quantum Chemistry

nuclear charges → quantum chemistry program → energy

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





$Z = (7,7)$ $Z = (6,8)$

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

Density functional perturbation 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

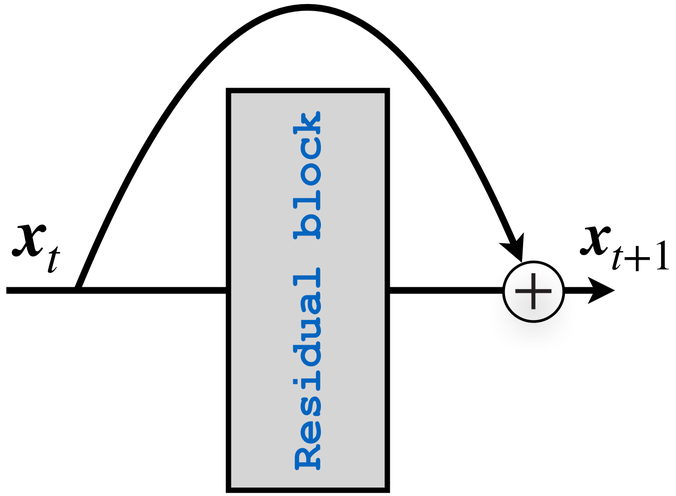
Differentiable DFT for a

unified, flexible, and (very likely) more efficient framework

https://github.com/mfherbst/juliacon_dft_workshop/blob/master/7_Properties_automatic_differentation.ipynb

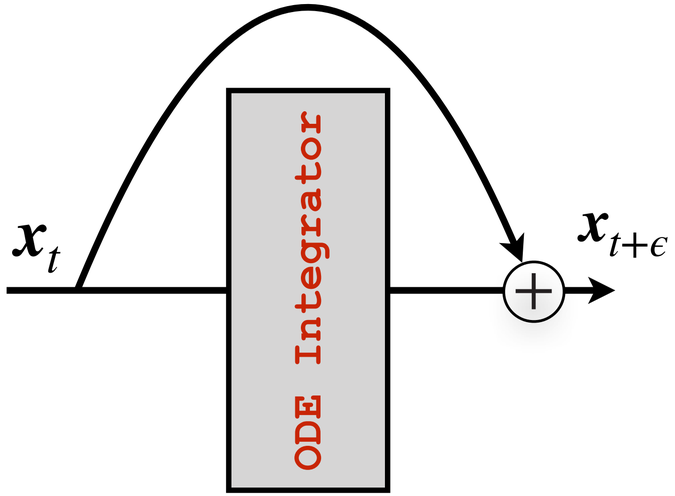
Neural Ordinary Differential Equations

Residual network



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

ODE integration



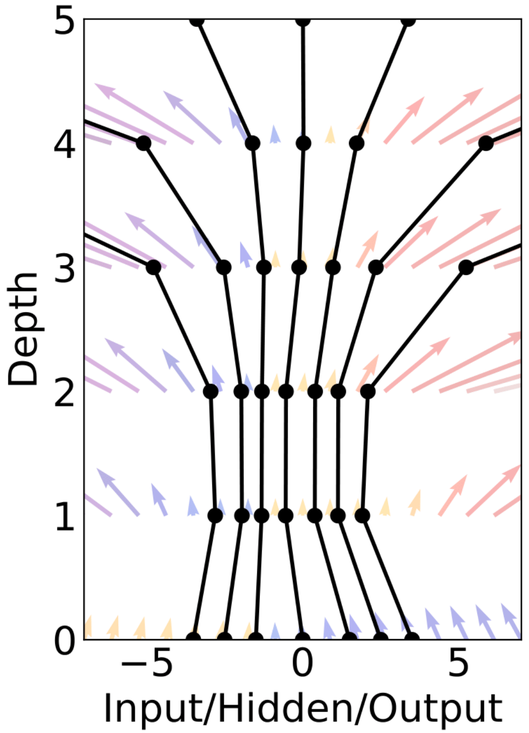
$$dx/dt = f(x)$$

Chen et al, 1806.07366 NIPS '18 Best paper award

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

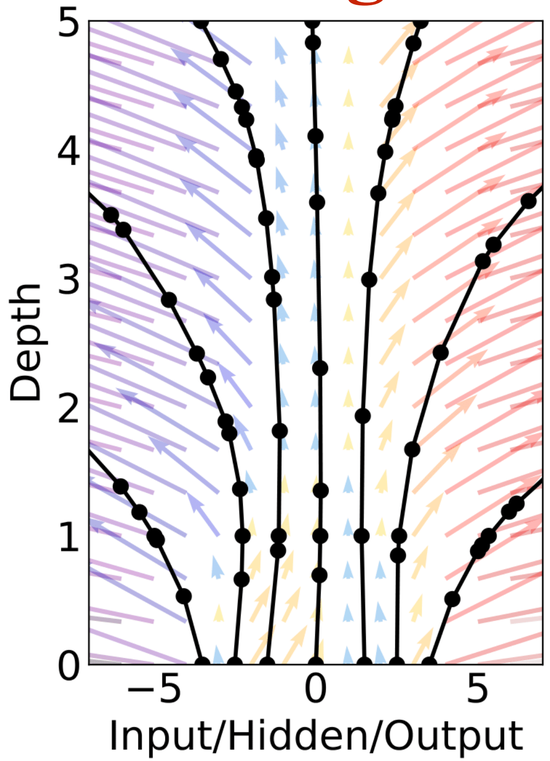
Neural Ordinary Differential Equations

Residual network



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

ODE integration



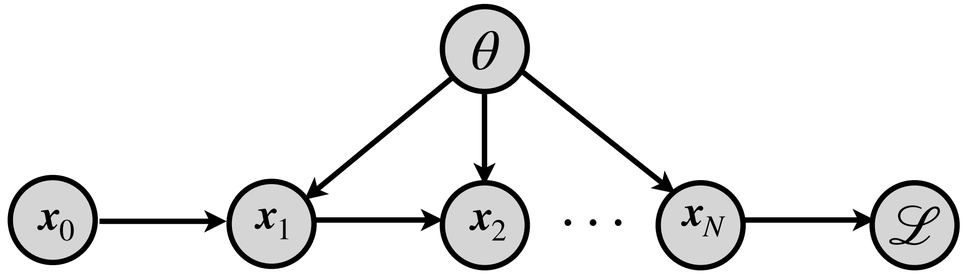
$$d\boldsymbol{x}/dt = f(\boldsymbol{x})$$

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

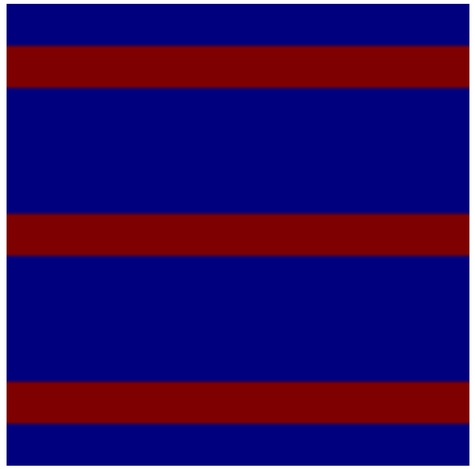
$$\frac{d\bar{\boldsymbol{x}}(t)}{dt} = -\bar{\boldsymbol{x}}(t)\frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta}, t)}{\partial \boldsymbol{x}}$$

Gradient w.r.t. parameter

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

Exercise:
Derive this!

Why do we need Neural ODE ?

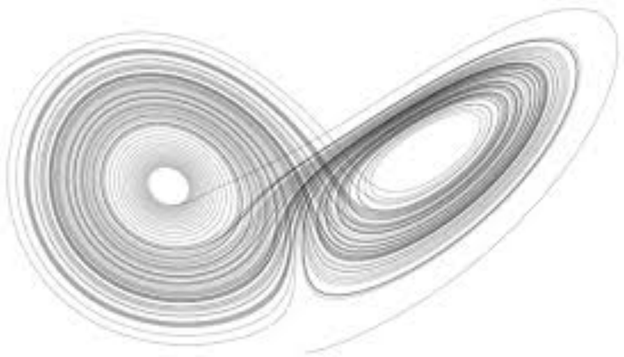


Backpropagating through
a fluid simulation

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

Differentiable ODE integrators

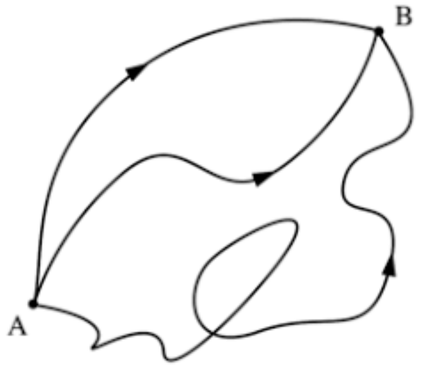
Dynamics systems



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

Classical and quantum control

Principle of least actions



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

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

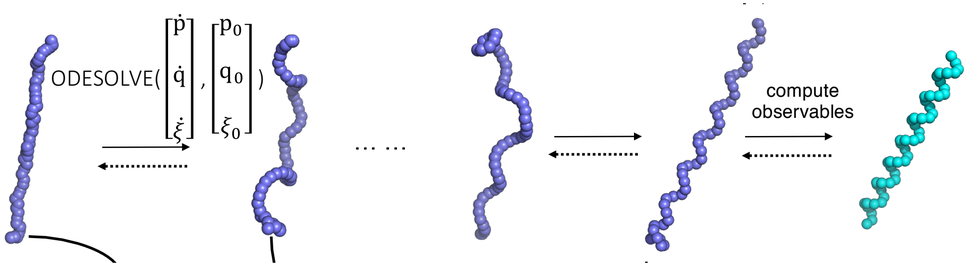
Differentiable Molecular Simulations for Control and Learning

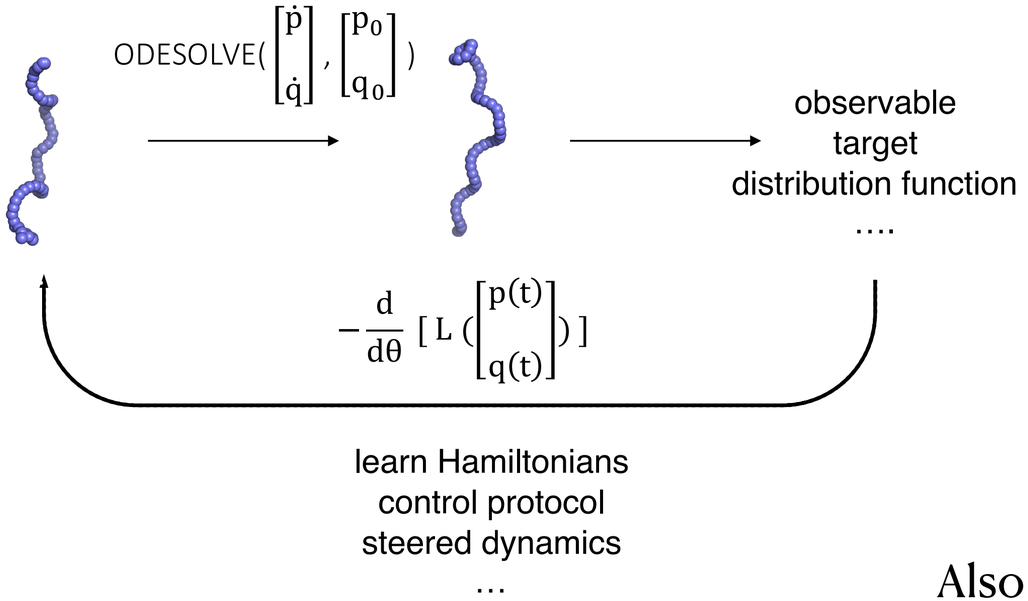
Wujie Wang[1], **Simon Axelrod**[1, 2], **and Rafael Gómez-Bombarelli**[1]

[1] Department of Material Science and Engineering,
Massachusetts Institute of Technology,
Cambridge, MA 02139, USA
{wwj, rafagb}@mit.edu
[2] Department of Chemistry and Chemical Biology,
Harvard University,
Cambridge, MA 02138, USA
{simonaxelrod}@g.harvard.edu

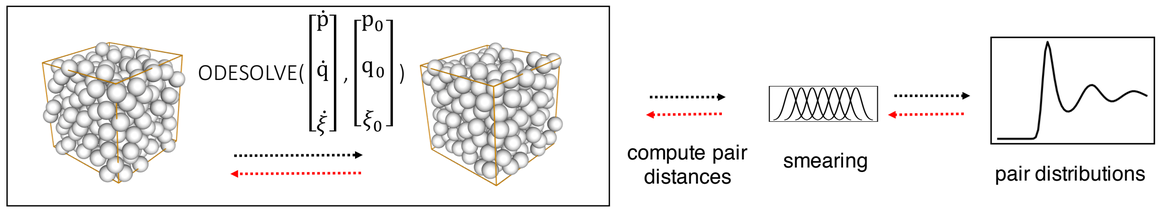
arXiv:2003.00868

Control: drive polymer to helix chain





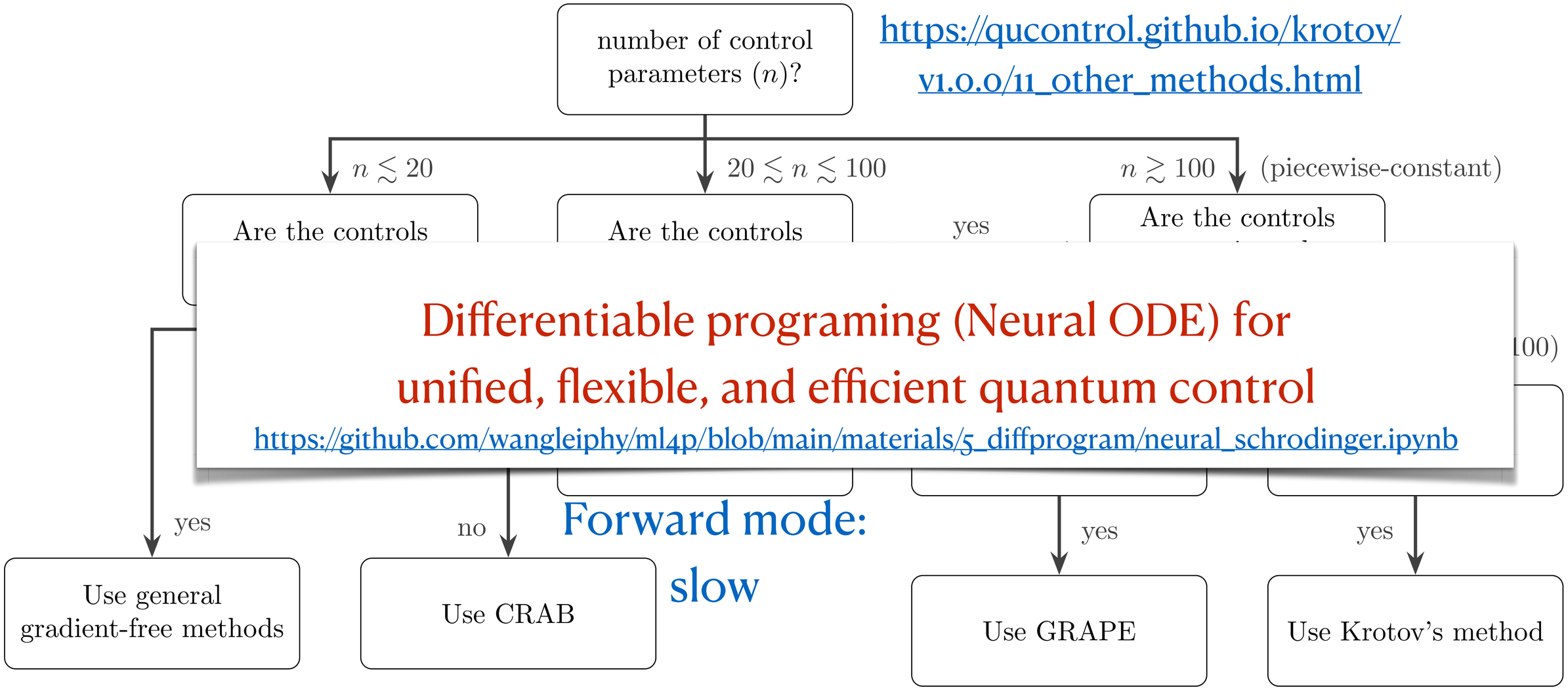
Learning:determine potential parameter from observed pair density



Also see Jax MD, Schoenholz, et al 1912.04232 and Goodrich et al, 2010.15175

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://github.com/wangleiphy/ml4p/blob/main/materials/5_diffprogram/neural_schrodinger.ipynb

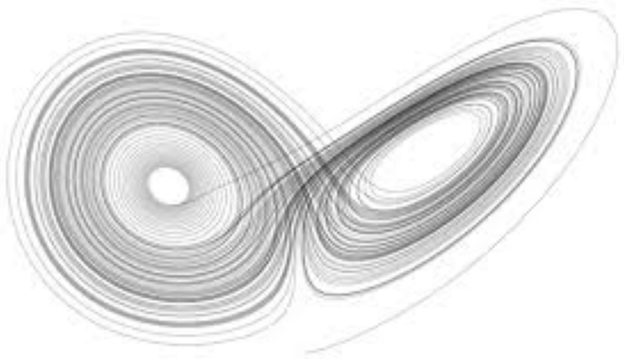
Forward mode: slow

No gradient: not scalable

Reverse mode w/ discretize steps: piesewise-constant assumption

Differentiable ODE integrators

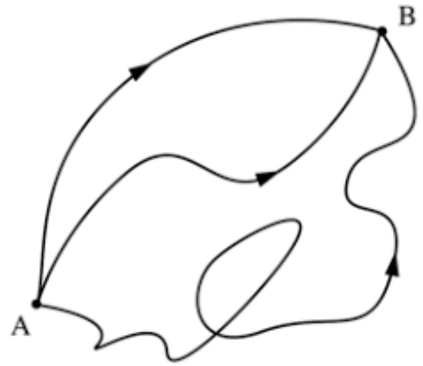
Dynamics systems



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

Classical and quantum control

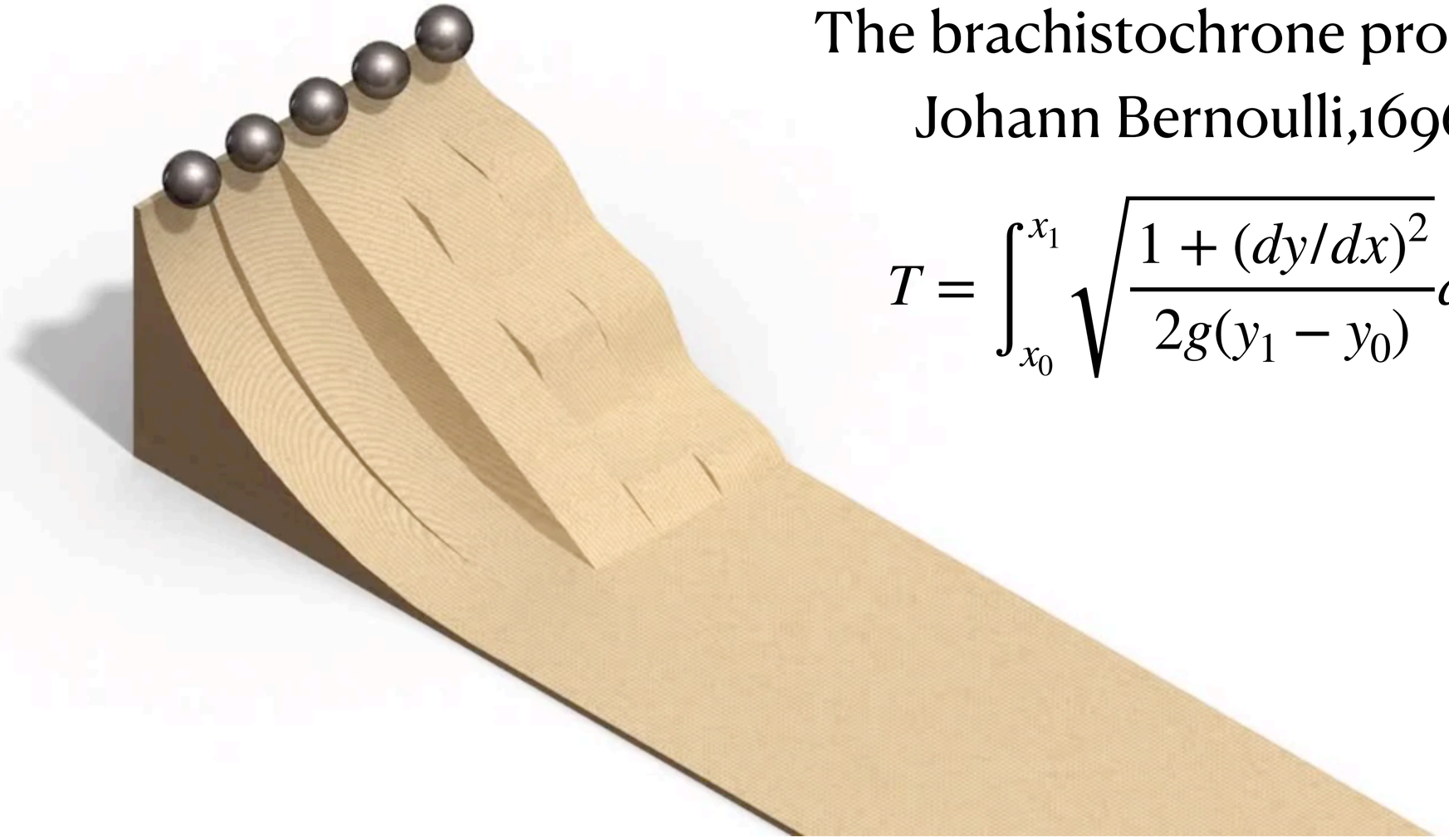
Principle of least actions



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

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

Differentiable functional optimization



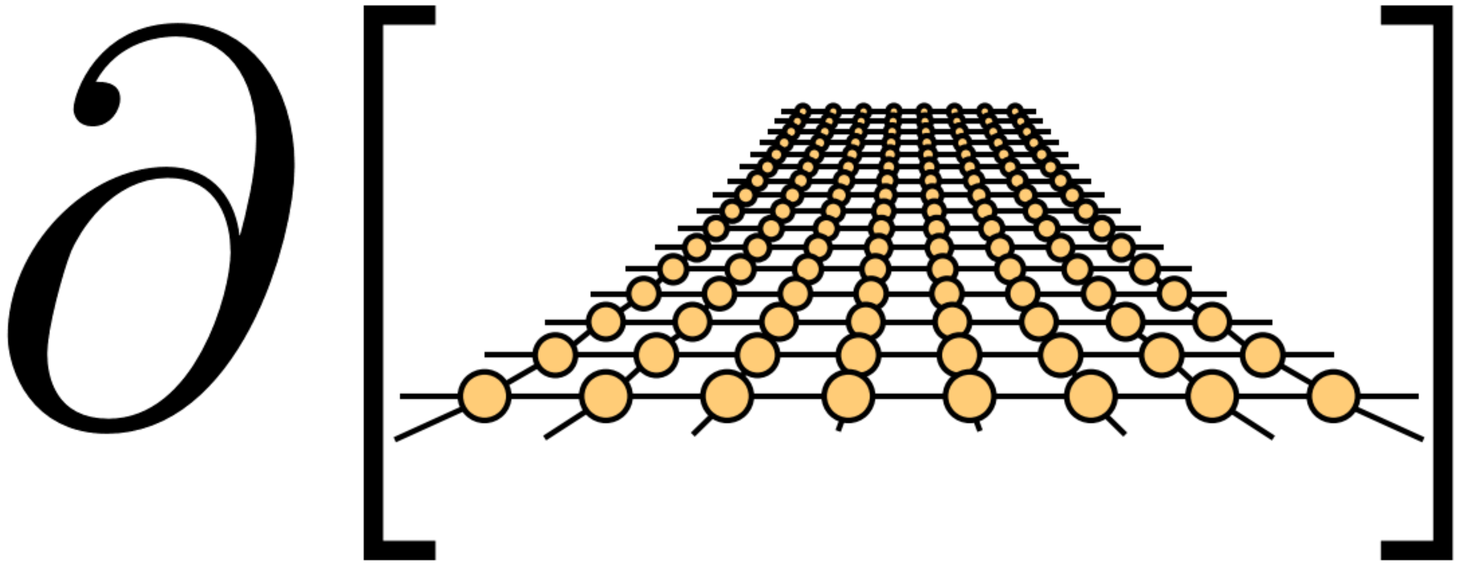
The brachistochrone problem

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

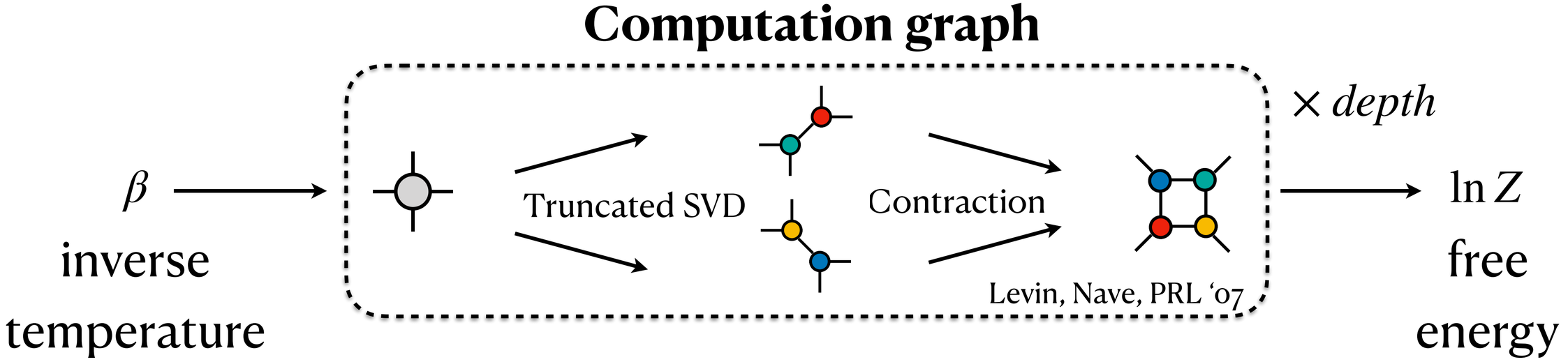
Differentiable Programming Tensor Networks

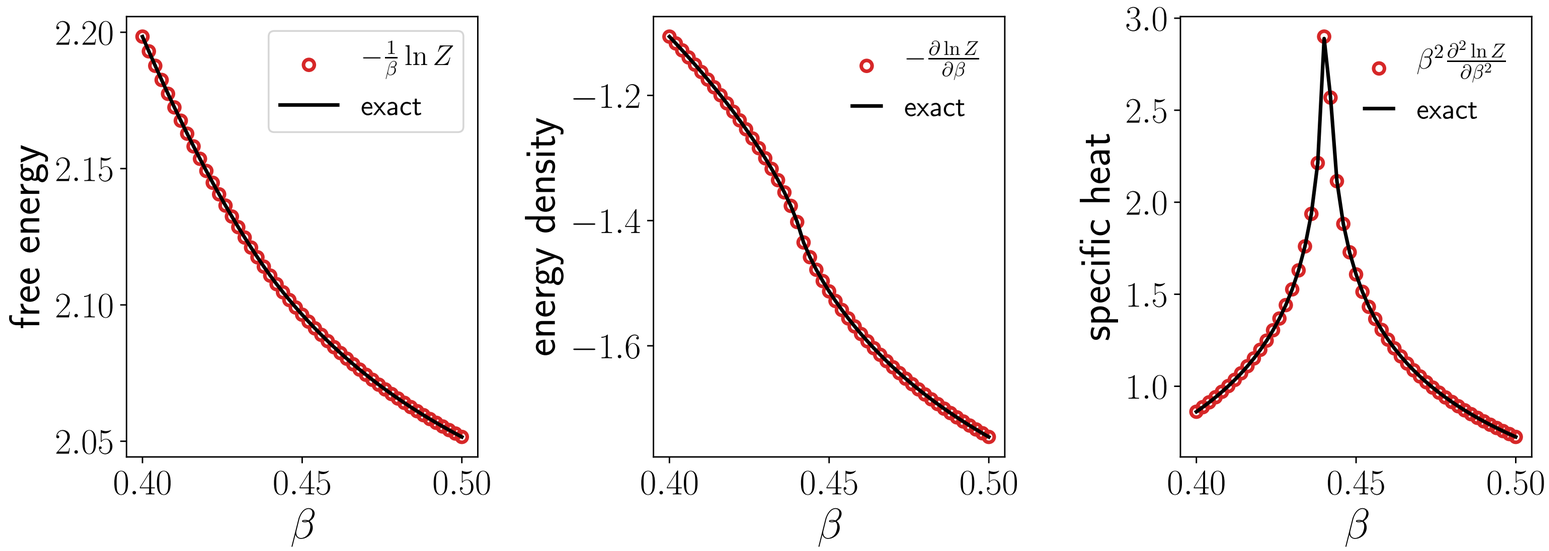


Liao, Liu, LW, Xiang, 1903.09650, PRX '19

https://github.com/wangleiphy/tensorgrad

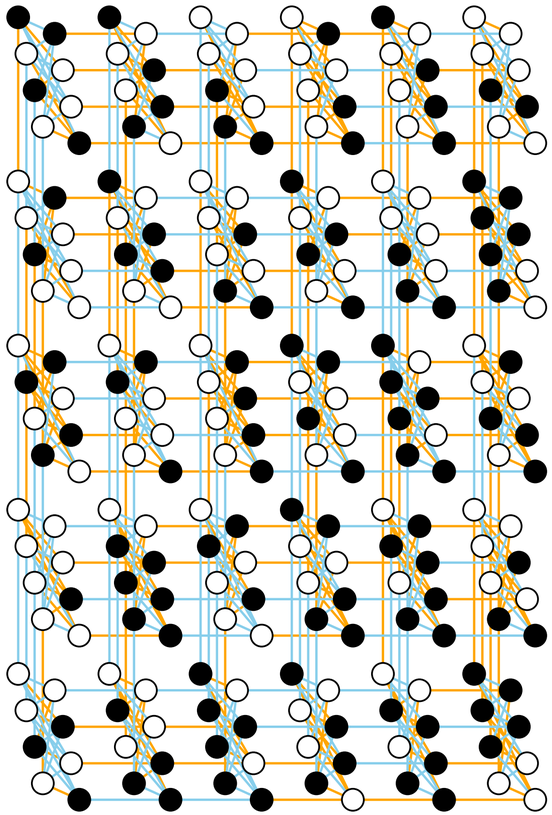
Differentiate through tensor renormalization group

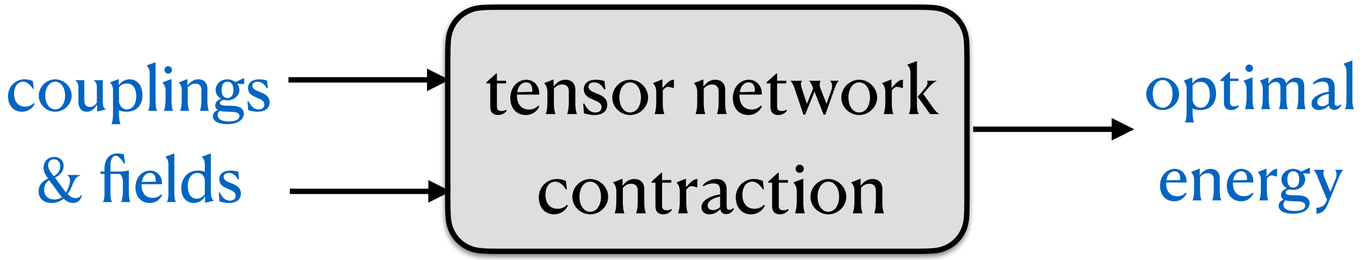




Compute physical observables as gradient of tensor network contraction

Differentiable spin glass solver





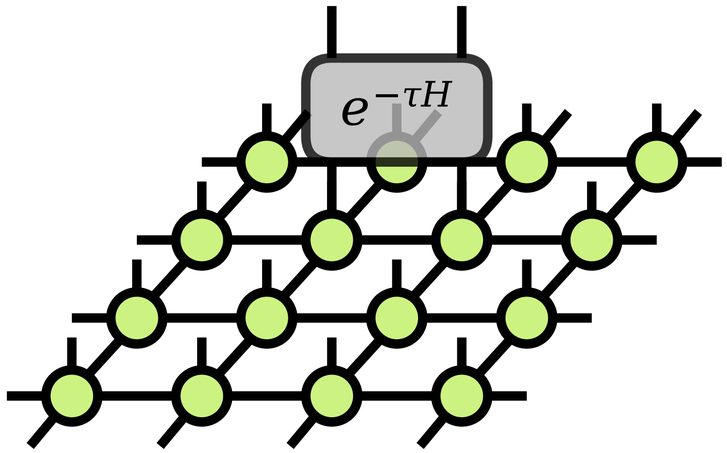
$$\text{optimal configuration} = \frac{\partial[\text{optimal energy}]}{\partial[\text{field}]}$$

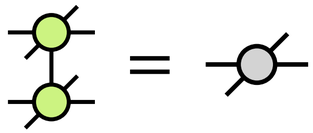
Liu, LW, Zhang, 2008.06888, PRL '21

https://github.com/TensorBFS/TropicalTensors.jl

Tensor network quantum states

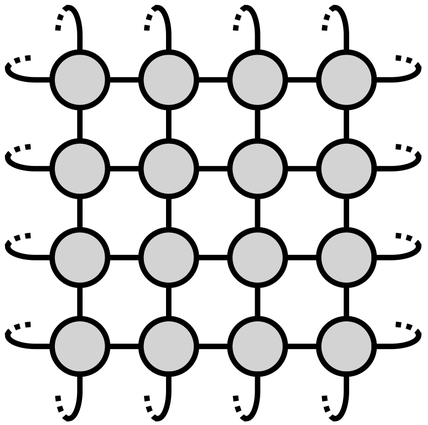
Optimization





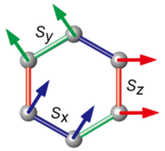
- Trotterized imaginary-time projection
- Update schemes: “simple”, “full” “cluster”, “faster full”...

Contraction

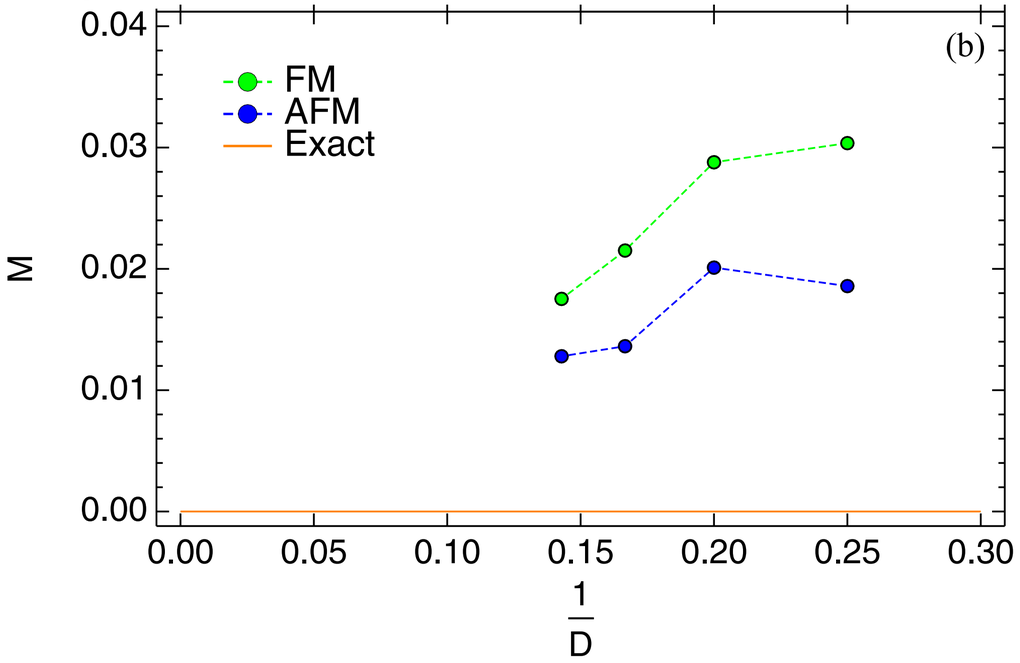


- #P hard in general
- Approximated schemes: TRG, Boundary MPS, Corner transfer matrix RG

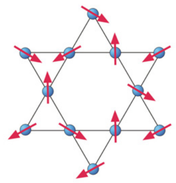
Expressibility v.s. Optimization: an eternal problem



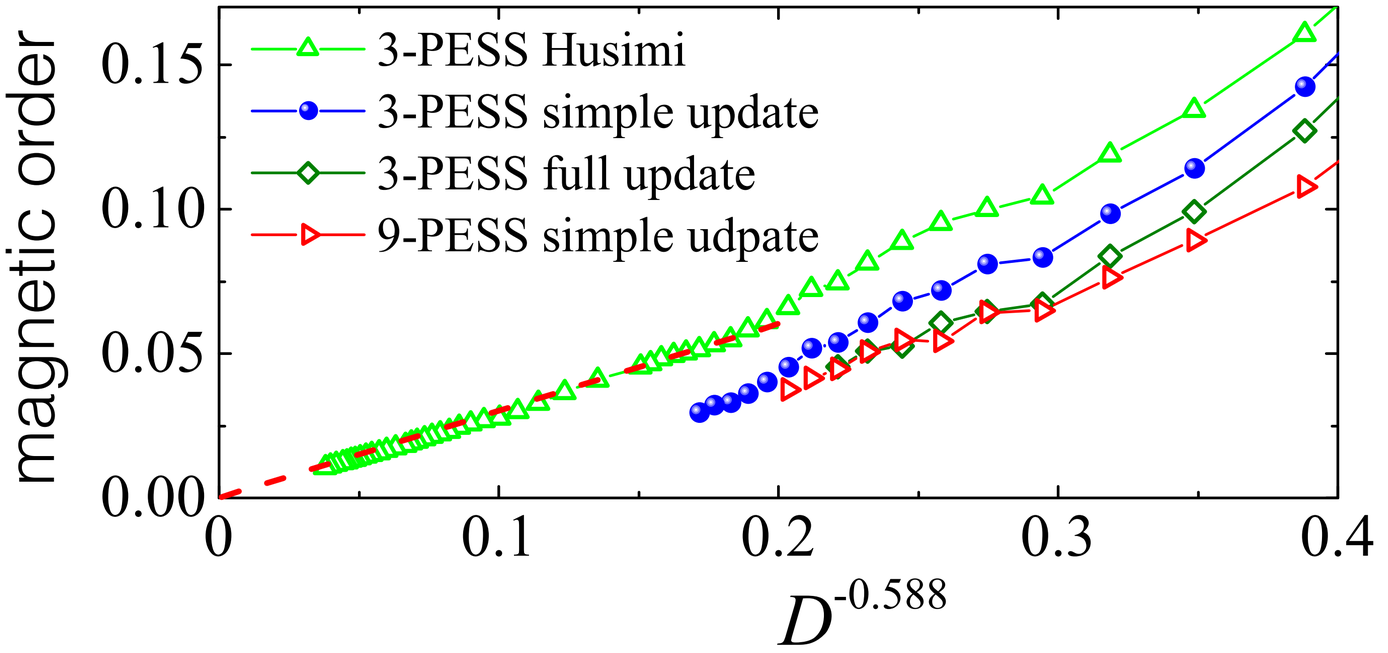
Kitaev Honeycomb model



Osorio, Corboz, Troyer, PRB '14



Kagome Heisenberg model



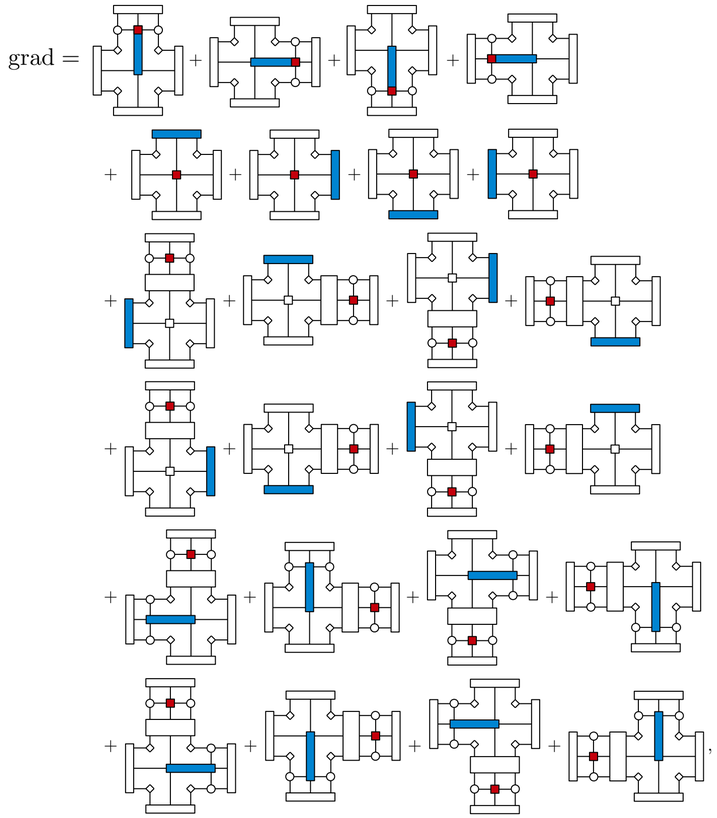
Liao et al, PRL '17

One typically finds ordered states

and tries hard to push up the bond dimensions

Differentiable tensor optimization

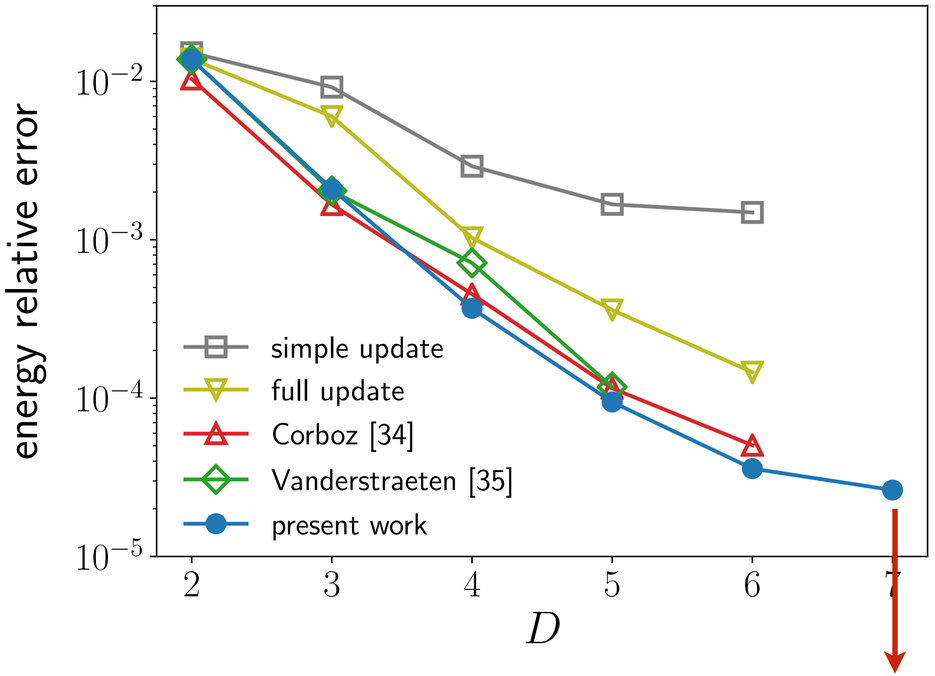
before...



Vanderstraeten et al, PRB ‘16

now, w/ differentiable programming

Liao, Liu, LW, Xiang, PRX ‘19



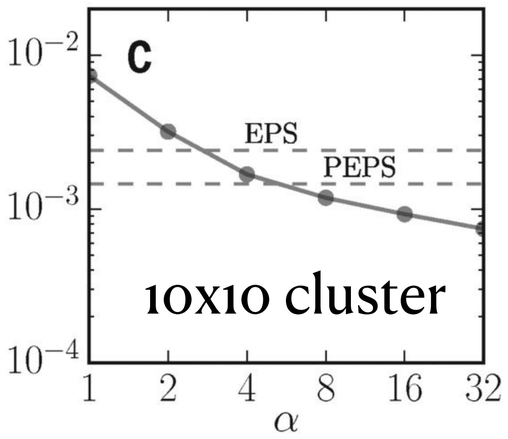
Best variational energy

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

Differentiable tensor optimization

Finite size

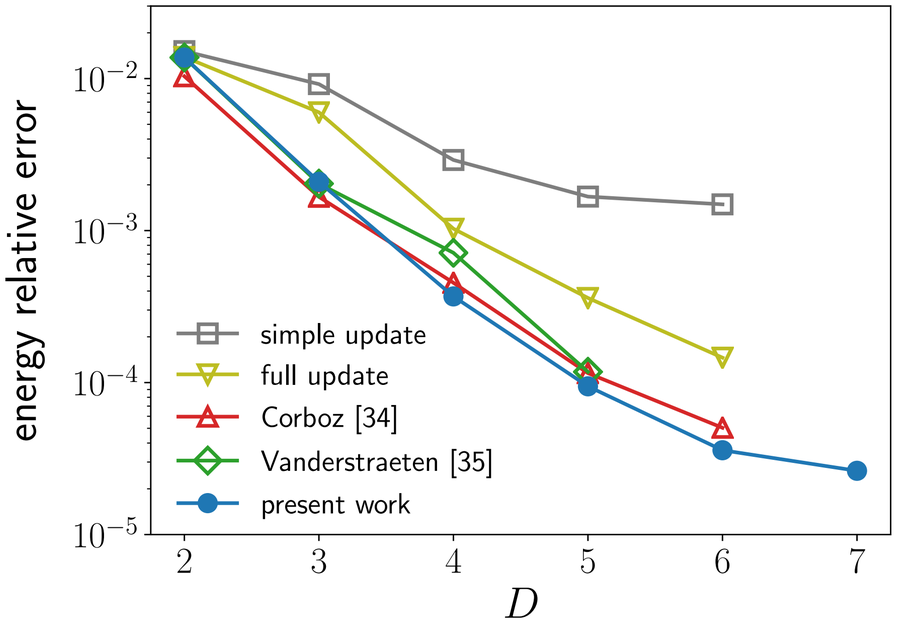
Neural network



Carleo & Troyer, Science ‘17

Infinite size

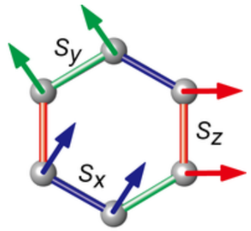
Tensor network



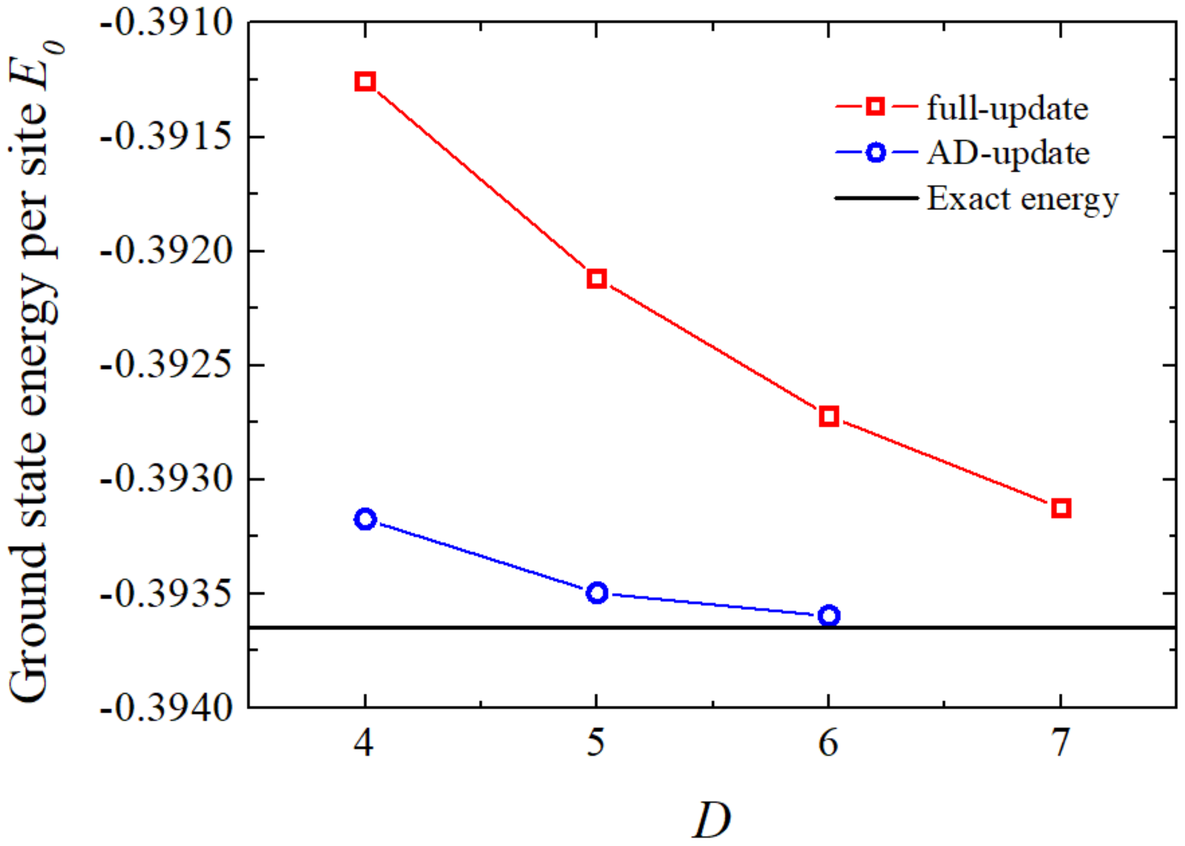
Liao, Liu, LW, Xiang, PRX ‘19

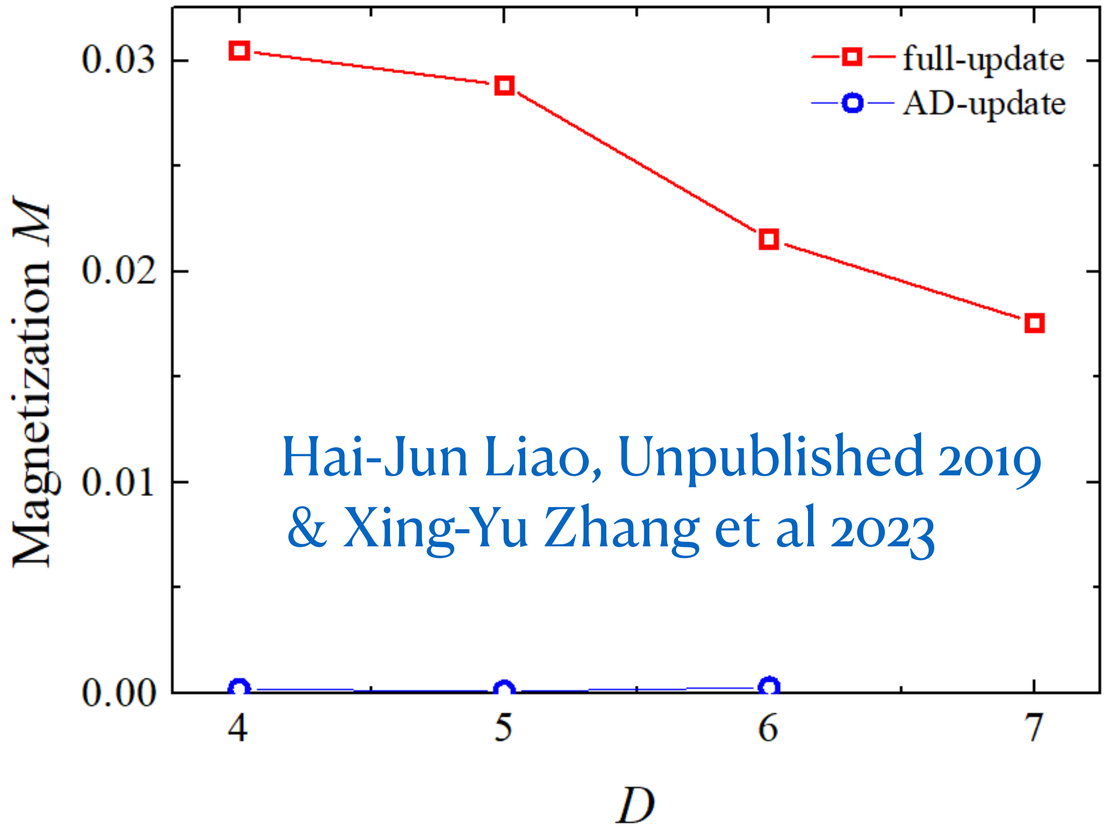
Further progress for challenging physical problems:
frustrated magnets, fermions, thermodynamics …

Chen et al, ‘19
Xie et al, ’20
Tang et al ’20



Kitaev honeycomb model

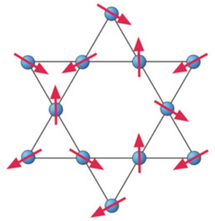




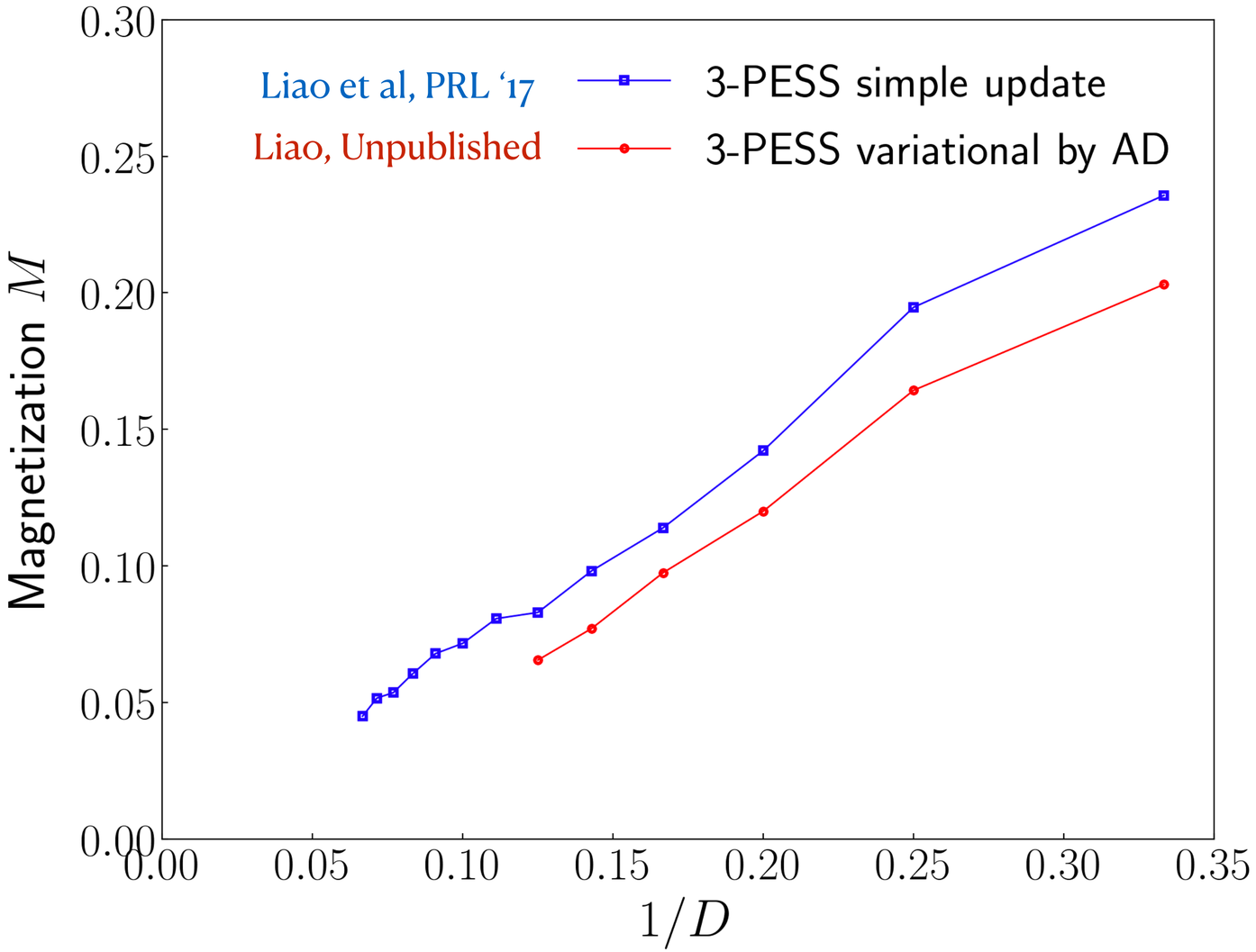
Reaches lower energy with fewer variational parameters

And substantially reduced magnetic order

c.f. analytically constructed iPEPS, Lee et al, PRL '19

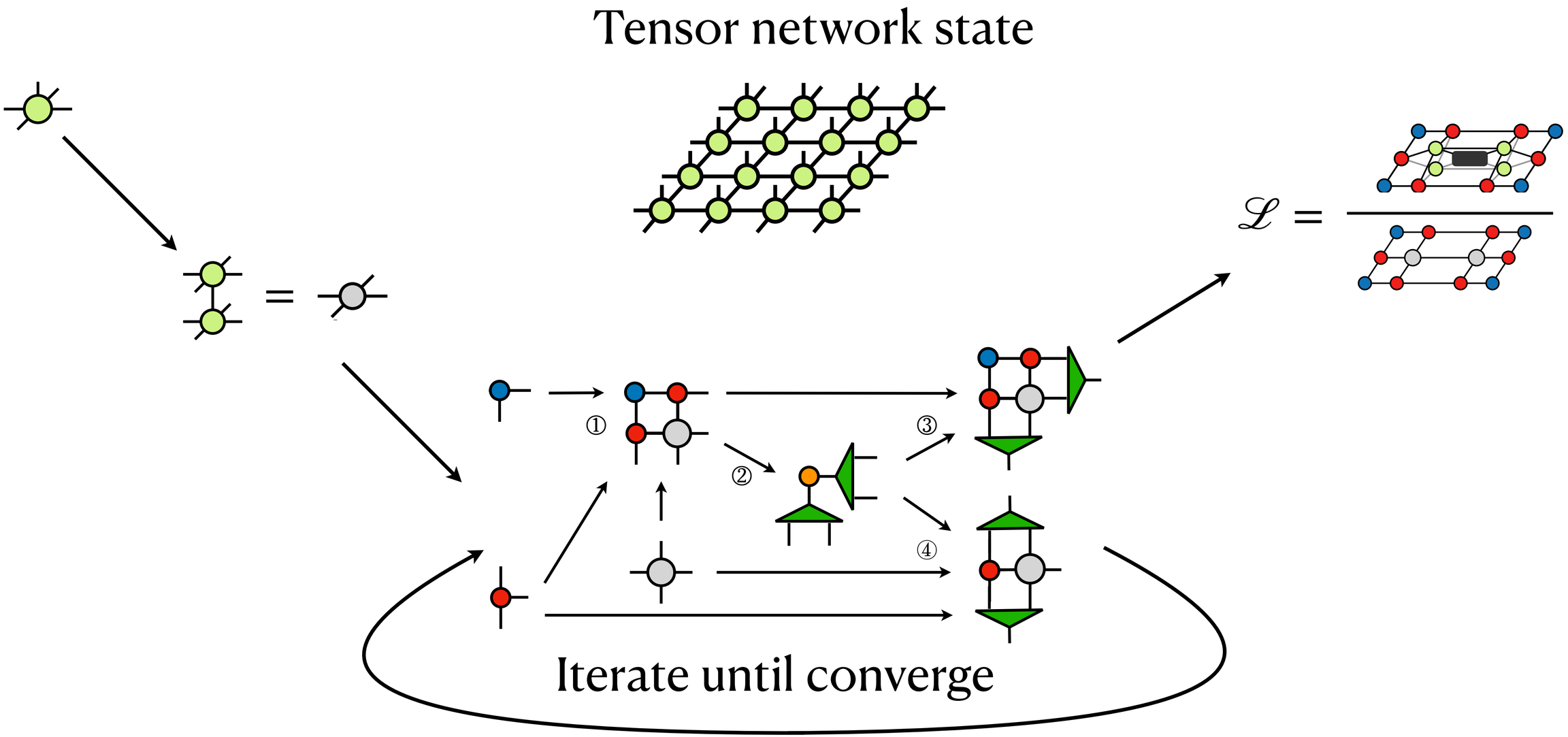


Kagome Heisenberg model



Lower energy, reduced magnetic order

Computation graph



Nuts and Bolts

- Numerical stable backward through SVD

$$A \to UDV^T \qquad \overline{A} \xleftarrow{?} \overline{U}, \overline{D}, \overline{V}$$

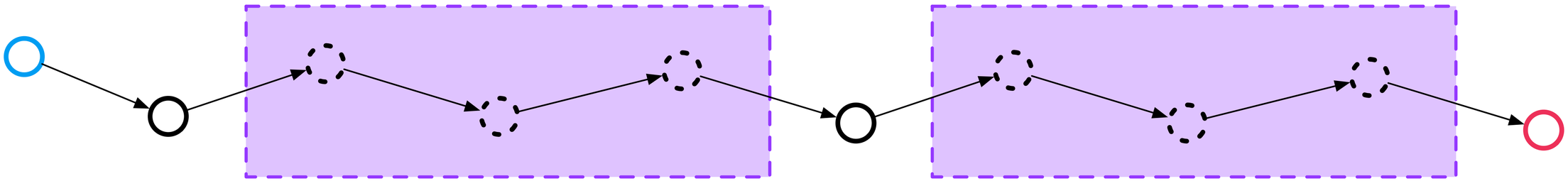
- Reduce memory via checkpointing or exploiting RG fixed point property

$$T_{i+1} = f(T_i, \theta) \xrightarrow{\textbf{Iterate}} T^* = f(T^*, \theta) \qquad \overline{\theta} = \overline{T^*} \left[1 - \frac{\partial f}{\partial T^*} \right]^{-1} \frac{\partial f}{\partial \theta}$$

Liao, Liu, LW, Xiang, PRX '19

Checkpoint

Trade memory with recomputation



Can also be regarded as layers with customized gradients

Implicit differentiation

$$F(x^\star(\theta), \theta) = 0\,.$$

$$\Downarrow$$

$$\partial_1 F(x^\star(\theta), \theta)\partial x^\star(\theta) + \partial_2 F(x^\star(\theta), \theta) = 0\,.$$

$$\Downarrow$$

$$\underbrace{-\partial_1 F(x^\star(\theta), \theta)}_{A \in \mathbb{R}^{d\times d}} \underbrace{\partial x^\star(\theta)}_{J \in \mathbb{R}^{d\times n}} = \underbrace{\partial_2 F(x^\star(\theta), \theta)}_{B \in \mathbb{R}^{d\times n}}\,.$$

$$x^\star(\theta) = \underset{x\in\mathbb{R}^{k\times 2}}{\operatorname{argmin}} f(x, \theta) \coloneqq \sum_{i,j} U(x_{i,j}, \theta),$$

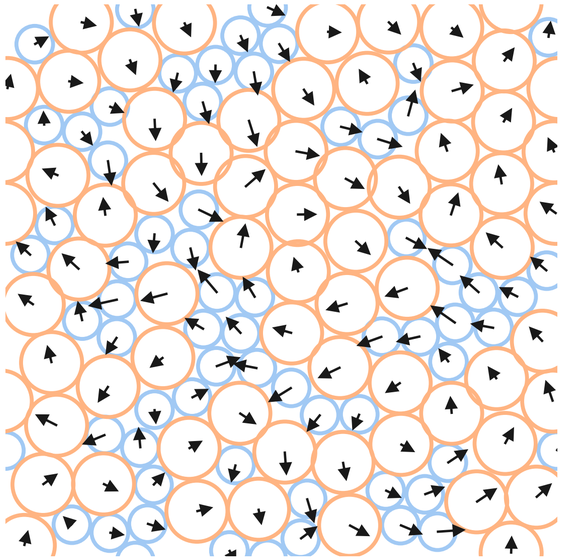
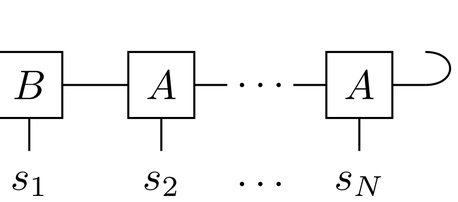


Figure 5: Particle positions and position sensitivity vectors, with respect to increasing the diameter of the blue particles.

Blondel et al, 2105.15183

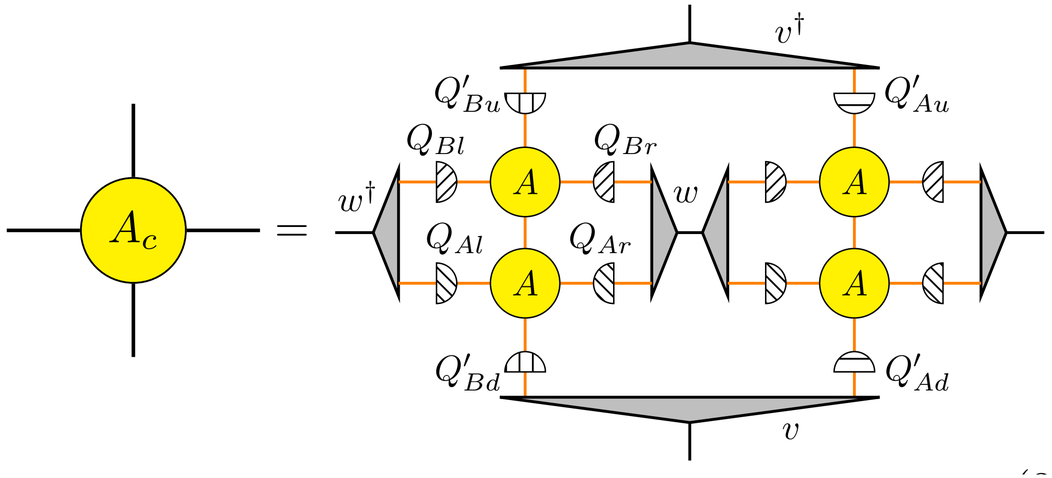
More applications of differentiable tensor networks

$$|\Phi_k(B)\rangle = \sum_{j=0}^{N-1} \mathrm{e}^{-ikj} \hat{T}^j$$



$$= \left. \frac{\partial |G(\lambda)\rangle}{\partial \lambda} \right|_{\lambda=0}$$

Generating function for
tensor diagrammatic summation
Tu et al, 2101.03935



Scaling dimension from
tensor RG fixed point
Lyu et al, 2102.08136

Common primitives of Tensor RG, DFT, and scientific computing

Eigensolver/SVD	AD of complex-valued SVD, Wan and Zhang, 1909.02659 Degenerated eigenvalues: https://github.com/google/jax/issues/669
Dominant or truncated Eigensolver/SVD	https://math.mit.edu/~stevenj/18.336/adjoint.pdf https://buwantaiji.github.io/2020/01/AD-of-truncated-SVD/ Xie, Liu, LW, PRB '20
Fixed point iteration	http://implicit-layers-tutorial.org/implicit_functions/
ODE	https://github.com/google/jax/issues/1927

Gradients are Not All You Need

Luke Metz* **C. Daniel Freeman*** **Samuel S. Schoenholz**
Google Research, Brain Team
{lmetz, cdfreeman, schsam}@google.com

Tal Kachman
Radboud University
Donders Institute for Brain, Cognition and Behaviour
tal.kachman@donders.ru.nl

Abstract

Differentiable programming techniques are widely used in the community and are responsible for the machine learning renaissance of the past several decades. While these methods are powerful, they have limits. In this short report, we discuss a common chaos based failure mode which appears in a variety of differentiable circumstances, ranging from recurrent neural networks and numerical physics simulation to training learned optimizers. We trace this failure to the spectrum of the Jacobian of the system under study, and provide criteria for when a practitioner might expect this failure to spoil their differentiation based optimization algorithms.

Metz et al, 2111.05803

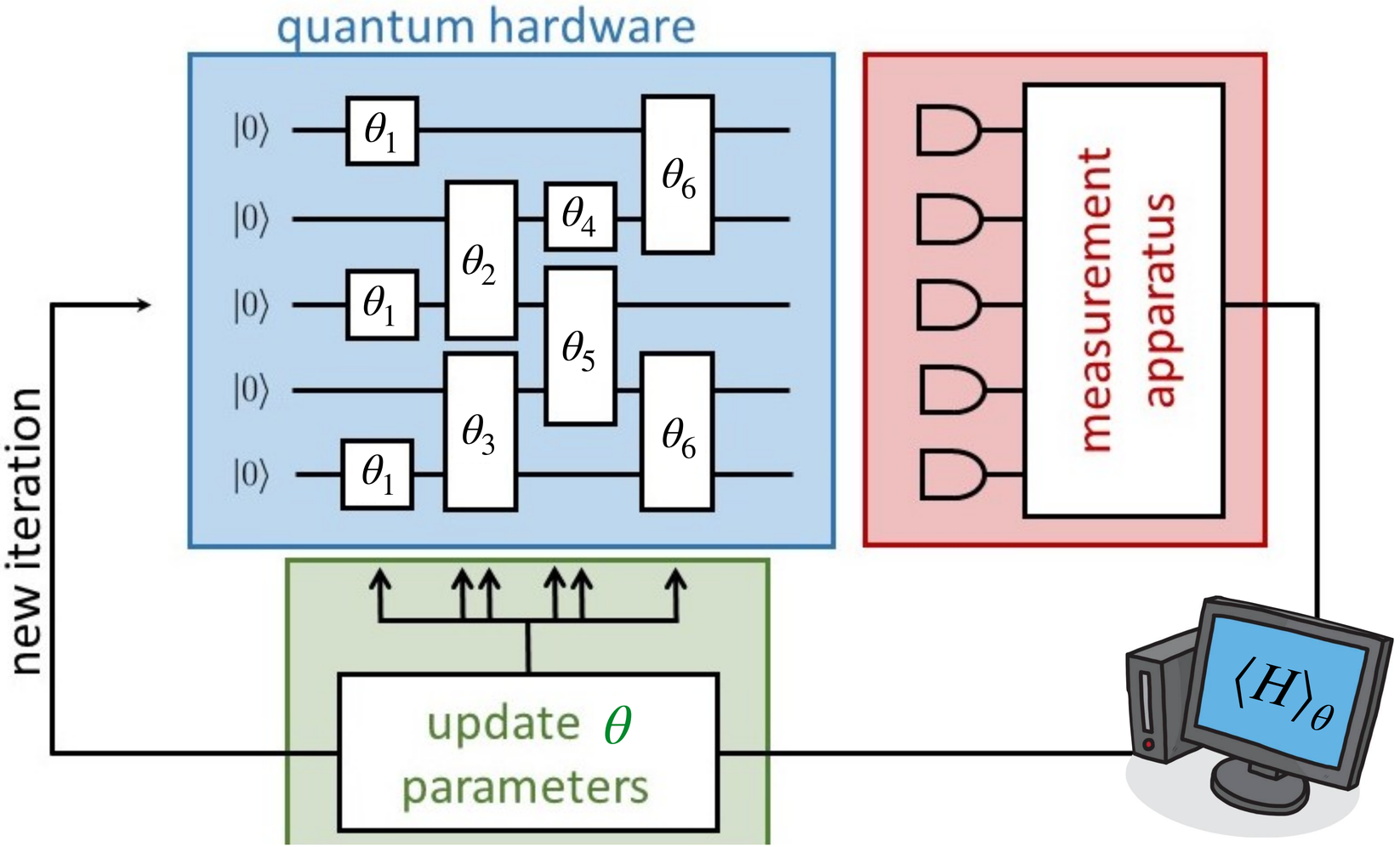
- Pick well behaved systems
- Truncated backpropogation
- Gradient clipping
- Back to blackbox method

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

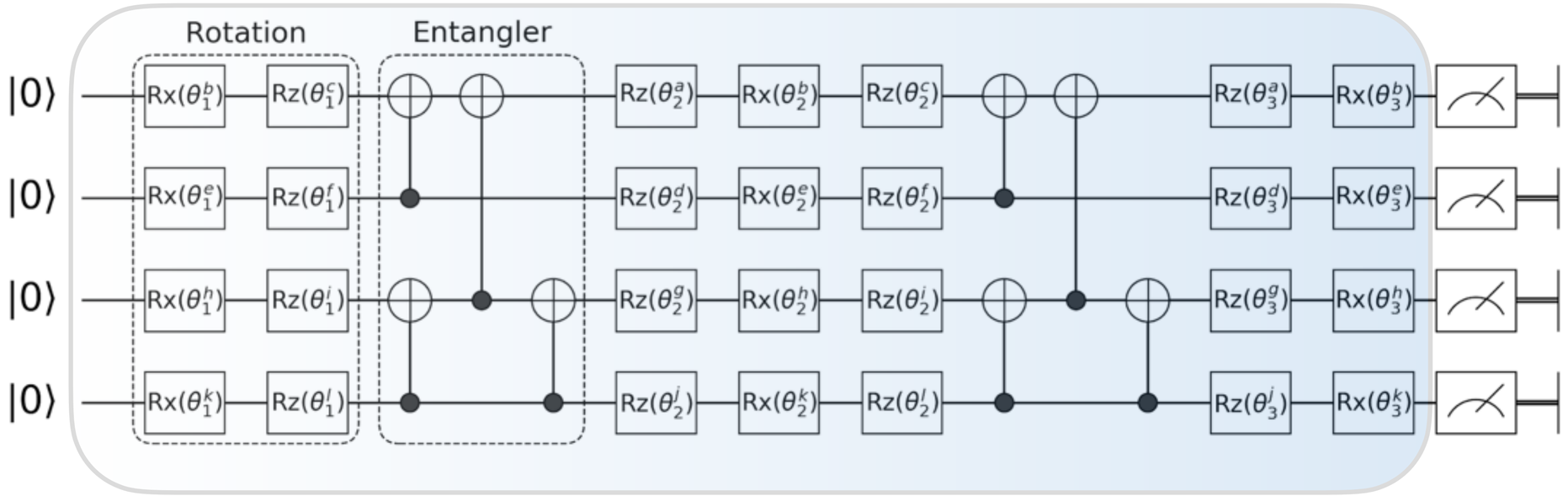


Peruzzo et al, Nat. Comm. '13

Quantum circuit as a variational ansatz

Differentiable quantum circuits

compute gradient in classical simulations



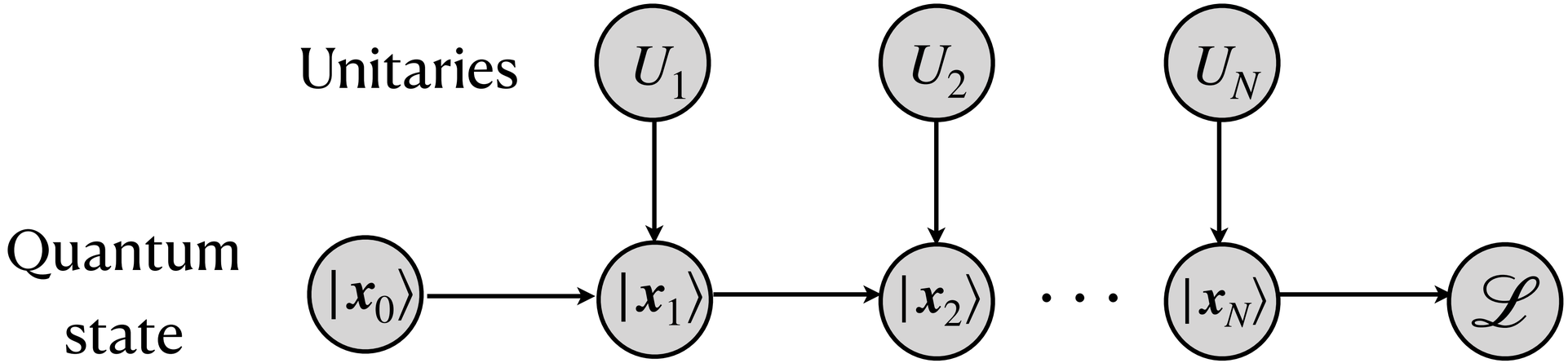
PENNYLANE

TensorFlow Quantum

Unfortunately, forward mode is too slow

Reverse mode is too 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

Reversible AD for variational quantum circuits*

forward

$U|x\rangle \rightarrow |y\rangle$

backward

"uncompute" $|x\rangle \leftarrow U^\dagger |y\rangle$

adjoint for mat-vec multiply

$\overline{|x\rangle} \leftarrow U^\dagger \overline{|y\rangle}$

$\overline{U} \leftarrow \overline{|y\rangle}\langle x|$

All are in-place operations without caching

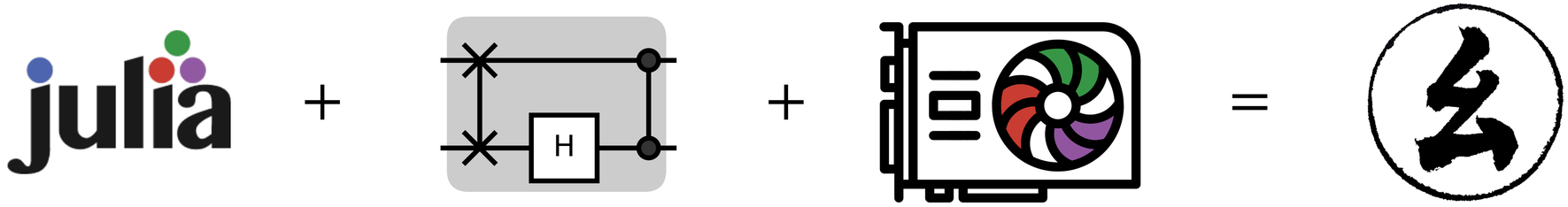
*GRAPE type algorithm on the level of circuits

```
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



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

Features:

- Reversible AD engine for quantum circuits
- Batch parallelization with GPU acceleration
- Quantum block intermediate representation

Luo, Liu, Zhang and LW, 1912.10877

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