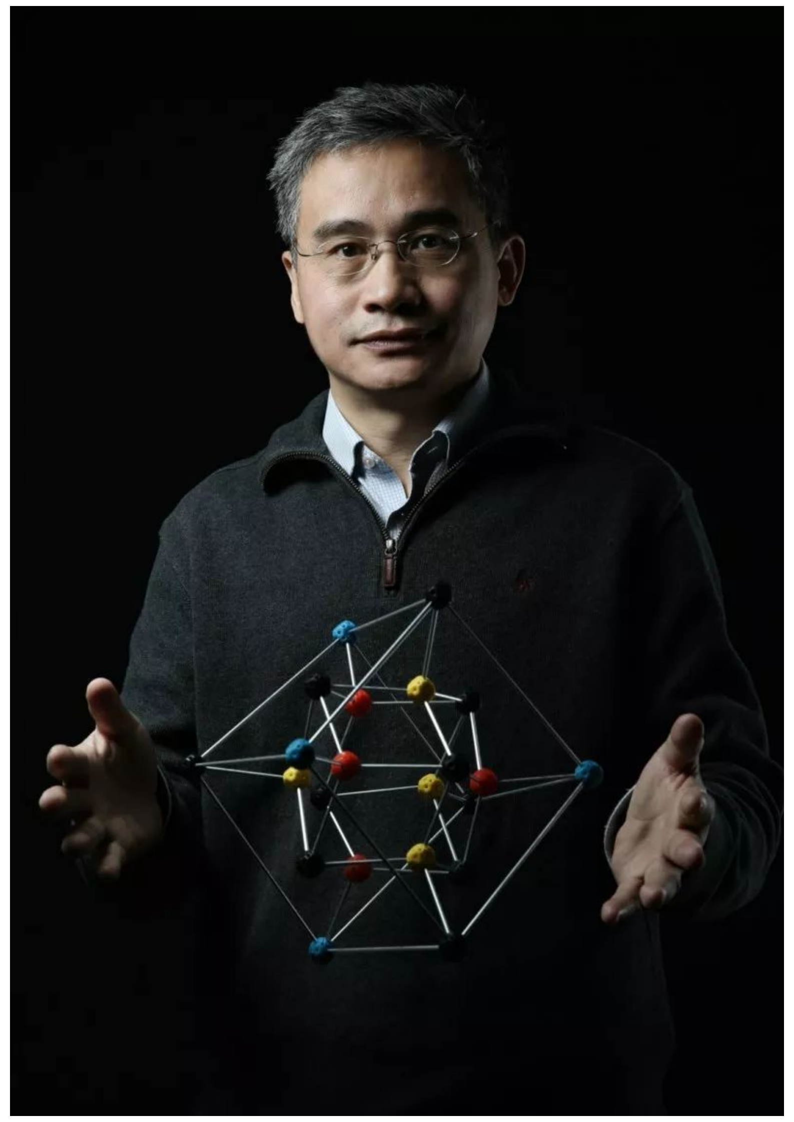
∂ifferentiable programming tensor networks and quantum circuits

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

Collaborators

Hai-Jun Liao

Jin-Guo Liu
(on the market)

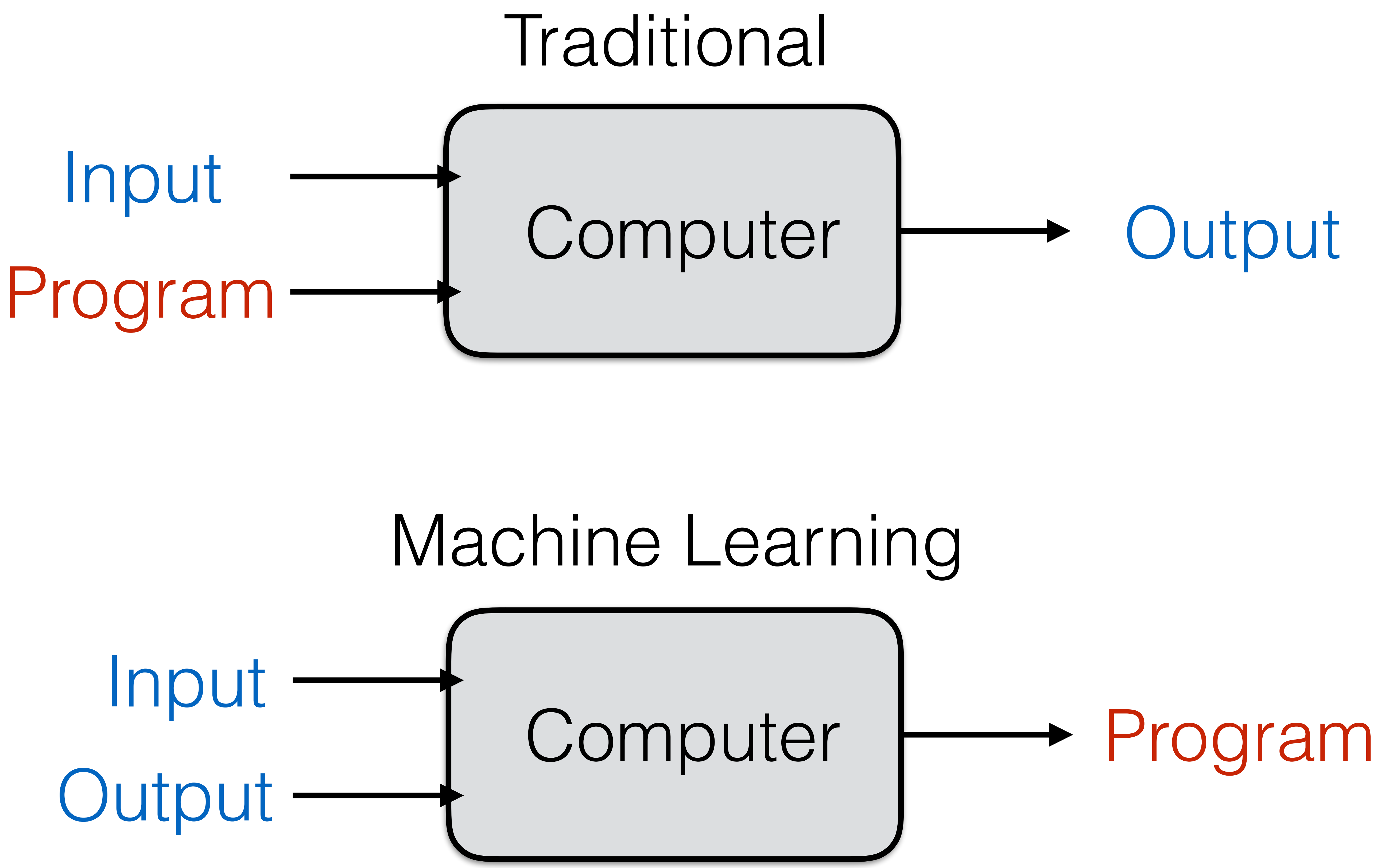


Tao Xiang

1903.09650

https://github.com/wangleiphy/tensorgrad

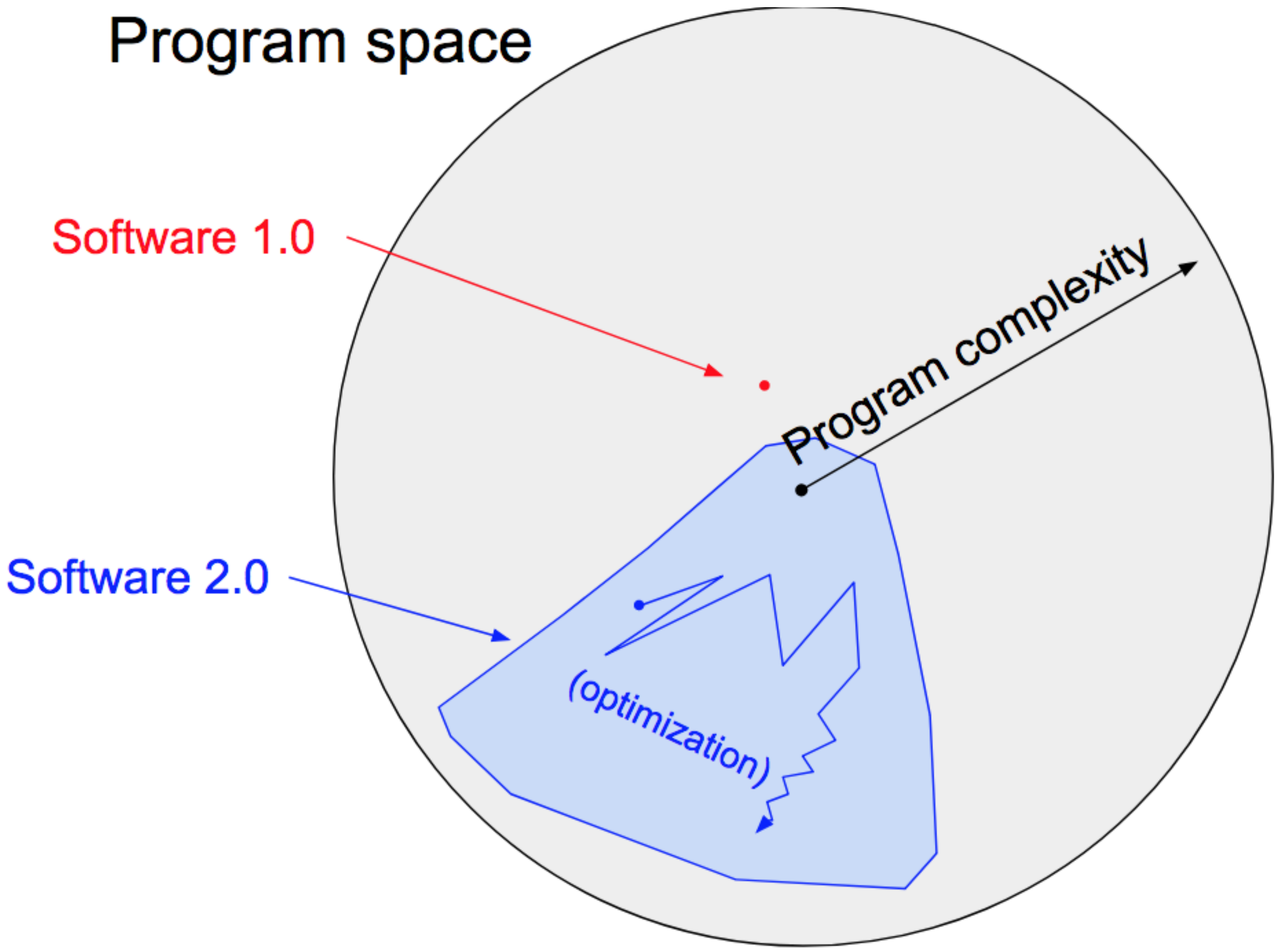
Differentiable Programming



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



Writing software 2.0 by gradient search in the program space

Differentiable Programming

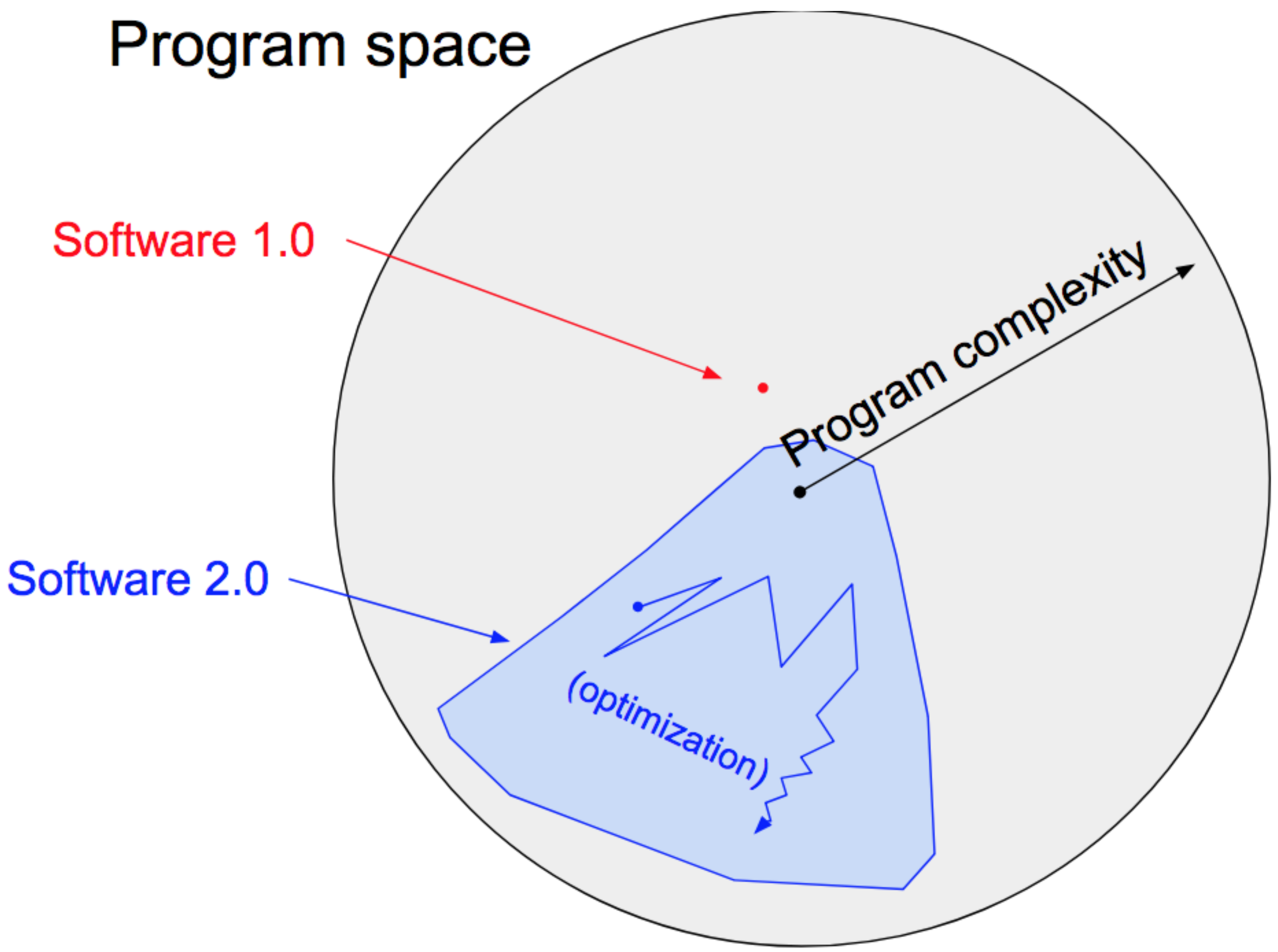
Benefits of Software 2.0

- 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

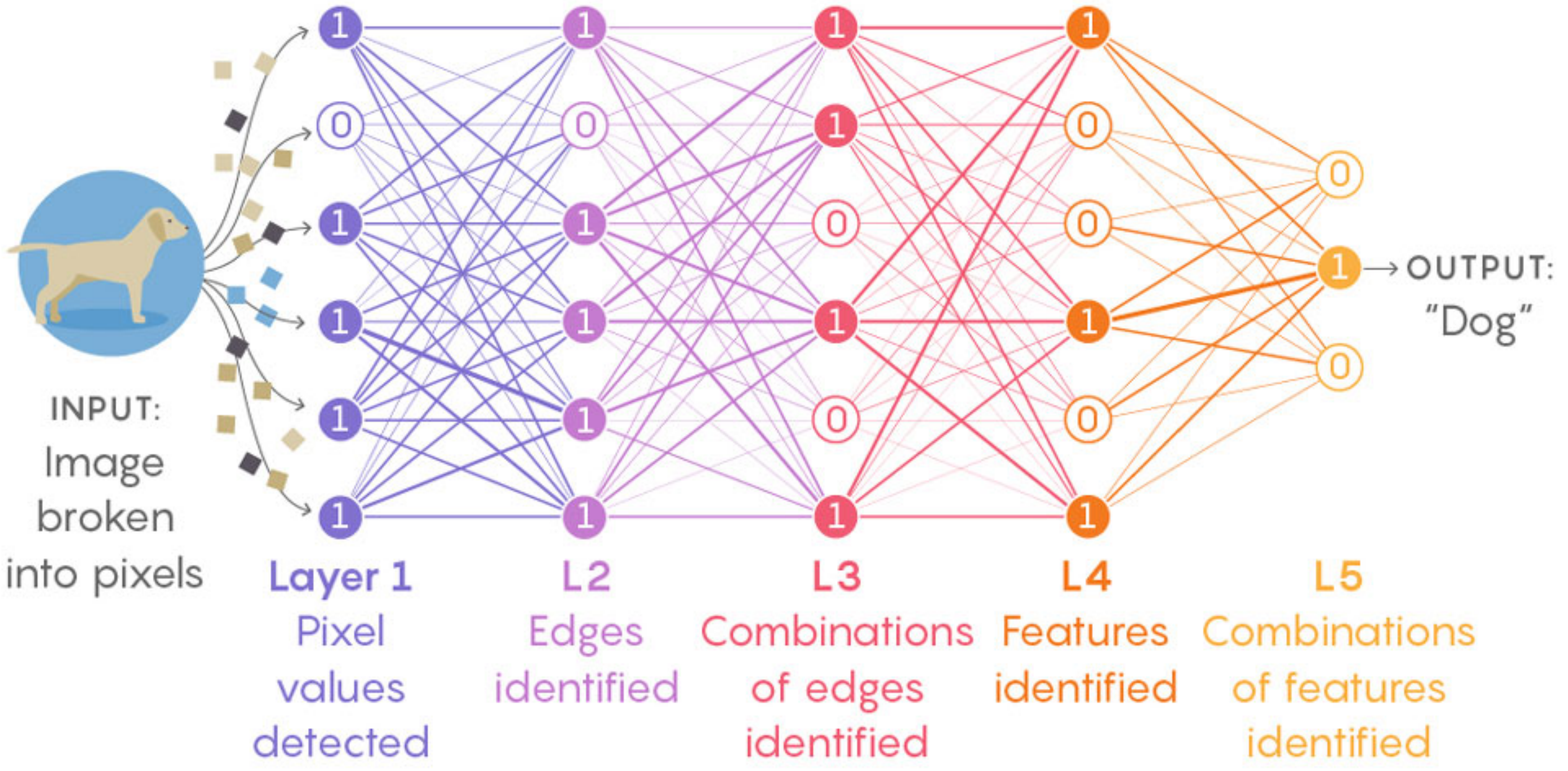
Director of AI at Tesla. Previously Research Scientist at OpenAI and PhD student at Stanford. I like to train deep neural nets on large datasets.

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



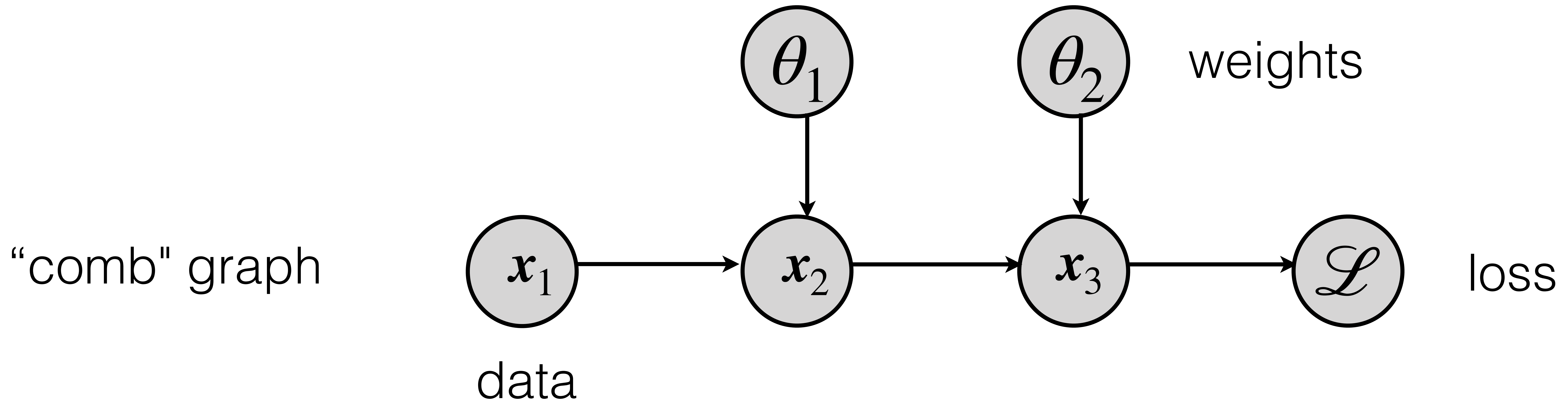
Writing software 2.0 by gradient search in the program space

The engine of deep learning



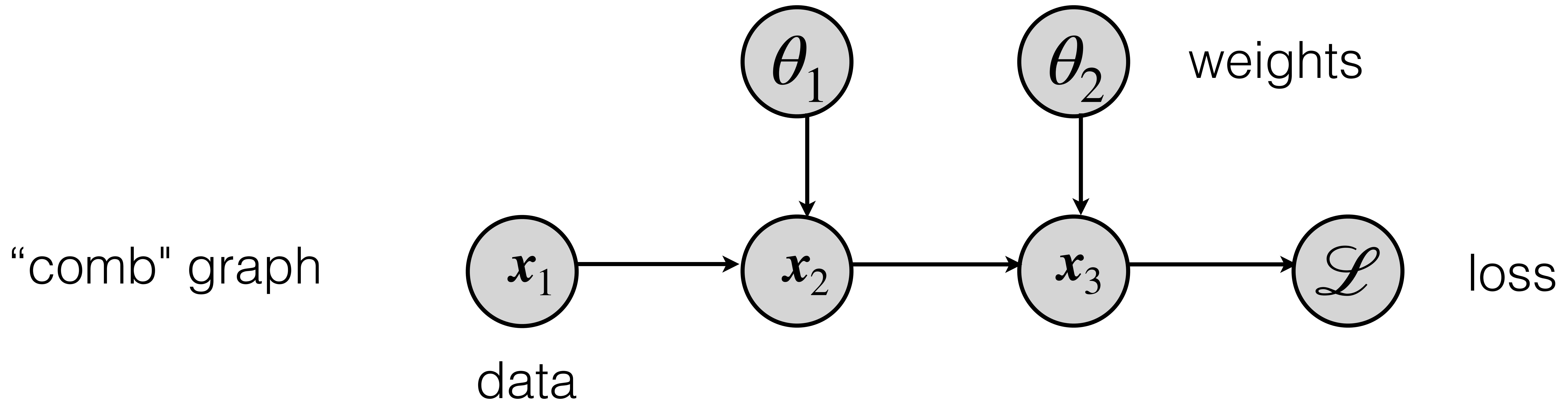
Compose differentiable components to a program e.g. a neural network, then optimize with gradient

Computation Graph



Pullback the adjoint through the graph

Computation Graph



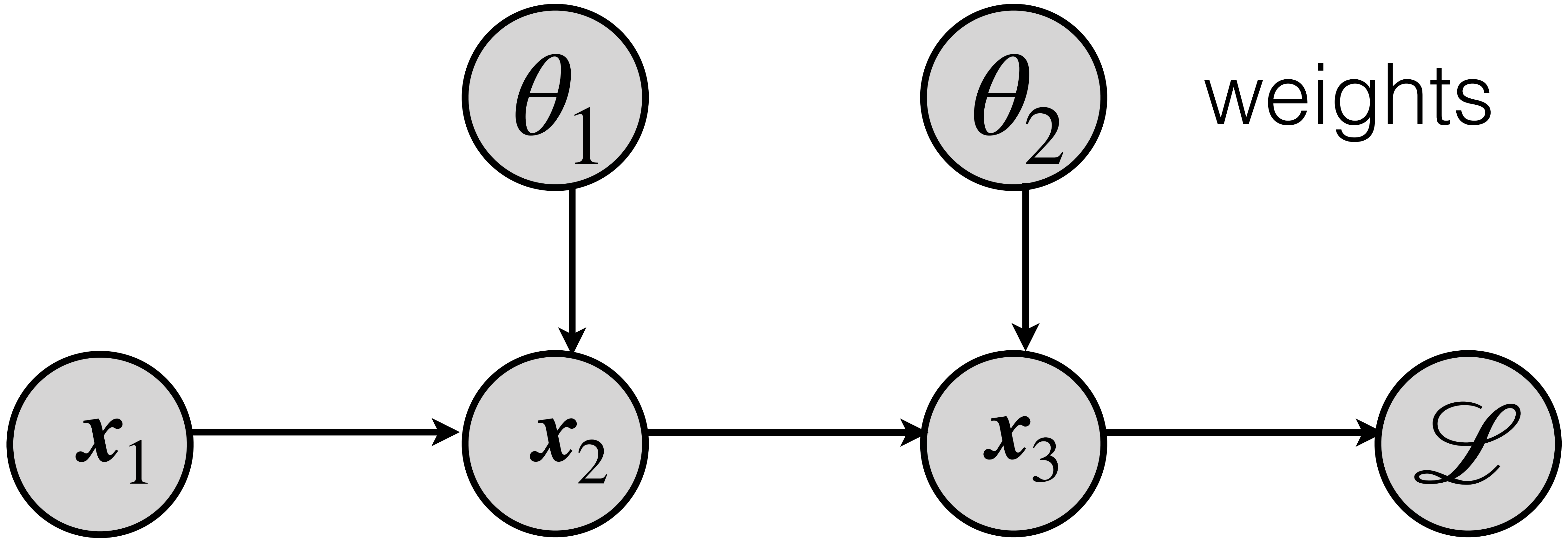
Define "adjoint" $\bar{\boldsymbol{x}} = \dfrac{\partial \mathscr{L}}{\partial \boldsymbol{x}}$

Pullback the adjoint through the graph

Computation Graph

weights

"comb" graph



loss

data

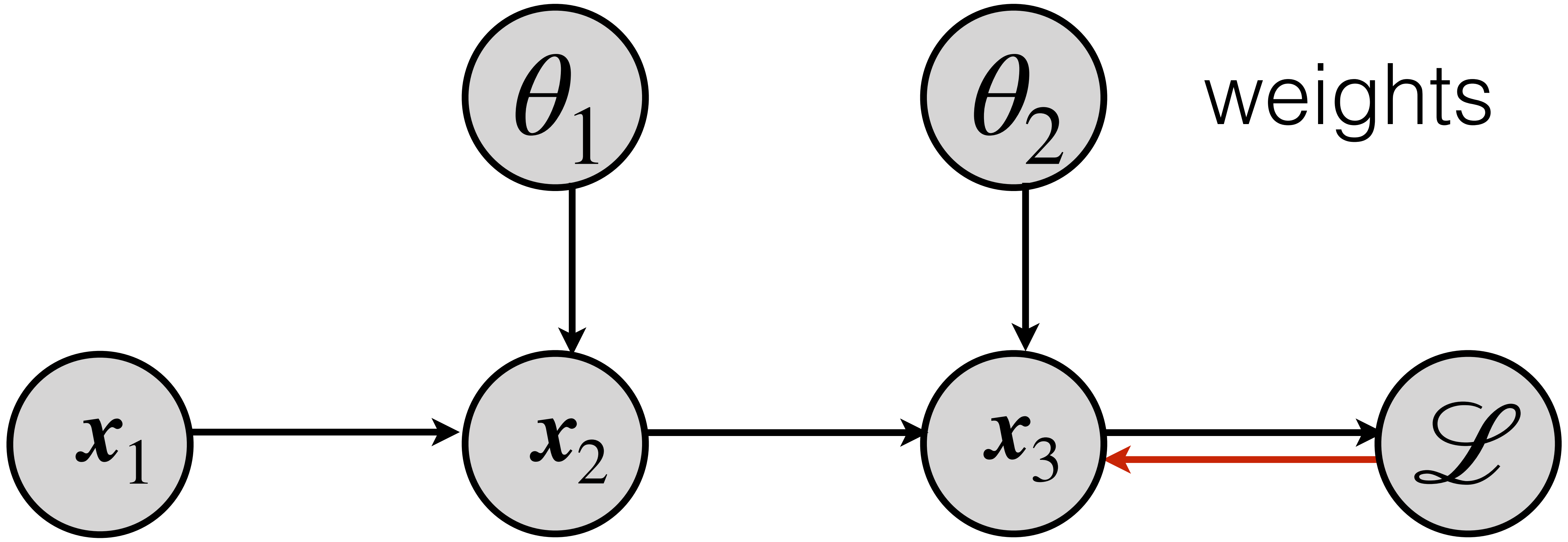
$\overline{\mathscr{L}} = 1$

Define "adjoint" $\bar{\boldsymbol{x}} = \dfrac{\partial \mathscr{L}}{\partial \boldsymbol{x}}$

Pullback the adjoint through the graph

Computation Graph

"comb" graph



weights

loss

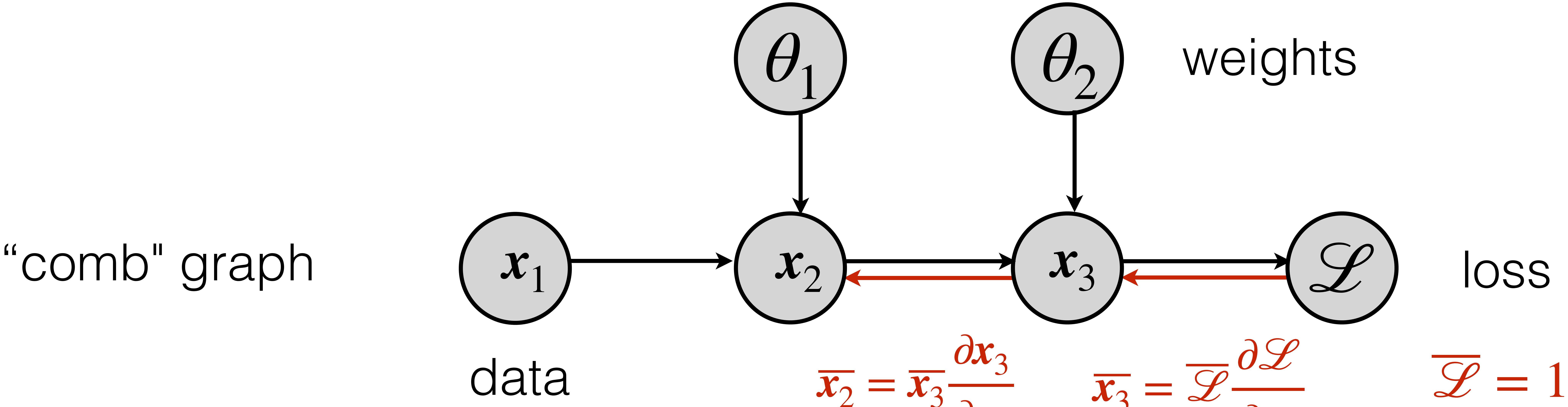
data

$$\overline{\boldsymbol{x}_3} = \overline{\mathscr{L}} \frac{\partial \mathscr{L}}{\partial \boldsymbol{x}_3} \qquad \overline{\mathscr{L}} = 1$$

Define "adjoint" $\bar{\boldsymbol{x}} = \dfrac{\partial \mathscr{L}}{\partial \boldsymbol{x}}$

Pullback the adjoint through the graph

Computation Graph

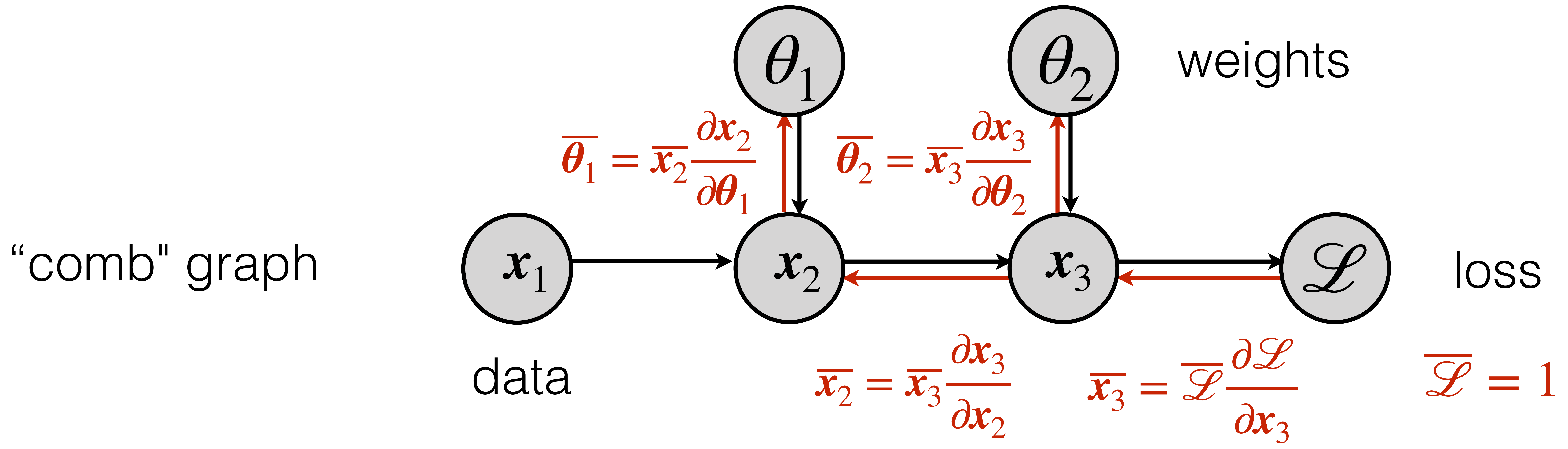


$$\overline{x_2} = \overline{x_3}\frac{\partial x_3}{\partial x_2} \qquad \overline{x_3} = \overline{\mathscr{L}}\frac{\partial \mathscr{L}}{\partial x_3} \qquad \overline{\mathscr{L}} = 1$$

Define "adjoint" $\bar{x} = \dfrac{\partial \mathscr{L}}{\partial x}$

Pullback the adjoint through the graph

Computation Graph

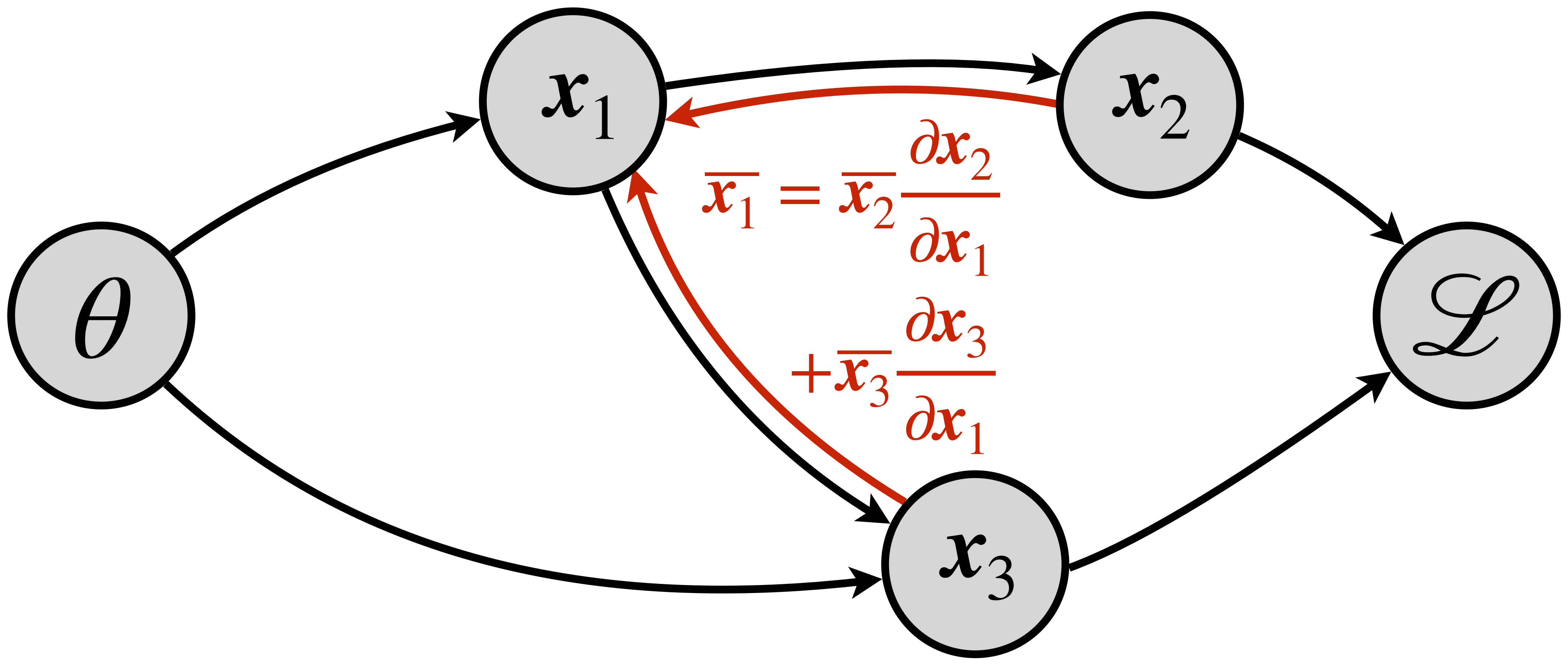


Define "adjoint" $\bar{\boldsymbol{x}} = \frac{\partial \mathscr{L}}{\partial \boldsymbol{x}}$

Pullback the adjoint through the graph

Computation Graph

directed
acyclic graph

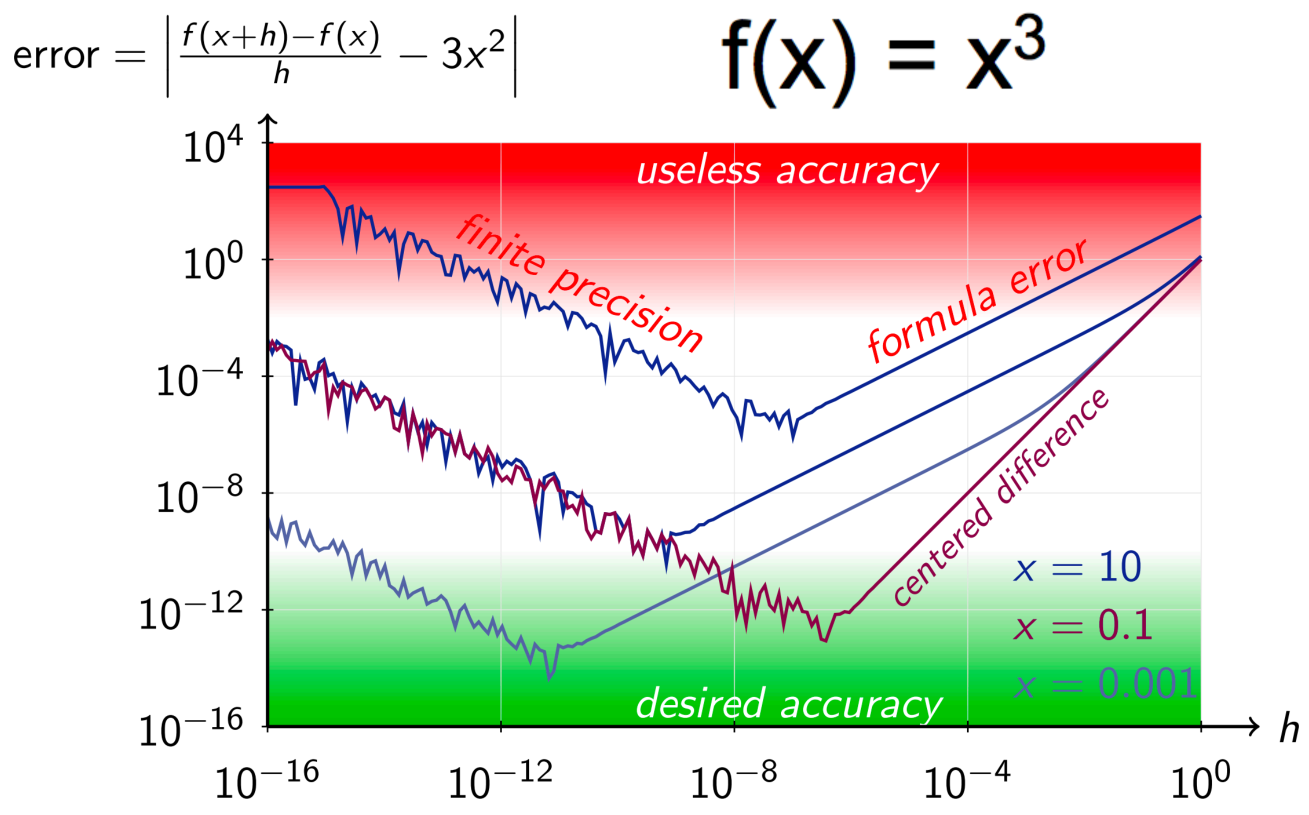


$$\overline{\boldsymbol{x}_i} = \sum_{j:\,\text{child of } i} \overline{\boldsymbol{x}_j}\,\frac{\partial \boldsymbol{x}_j}{\partial \boldsymbol{x}_i} \qquad \text{with} \quad \overline{\mathscr{L}} = 1$$

Message passing for the adjoint at each node

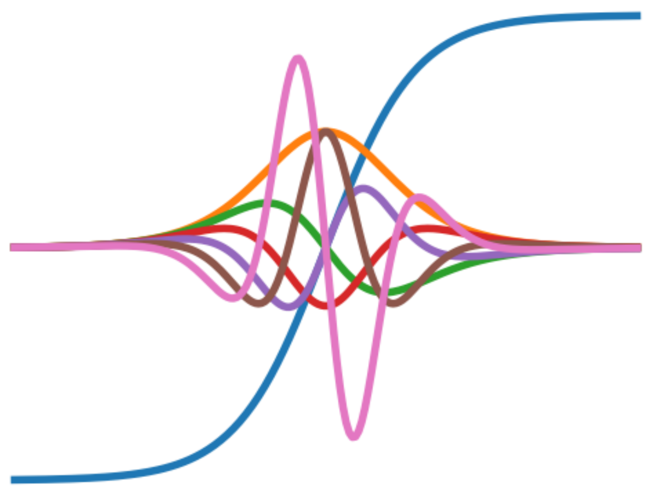
Advantages of automatic differentiation

- Accurate to the machine precision



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

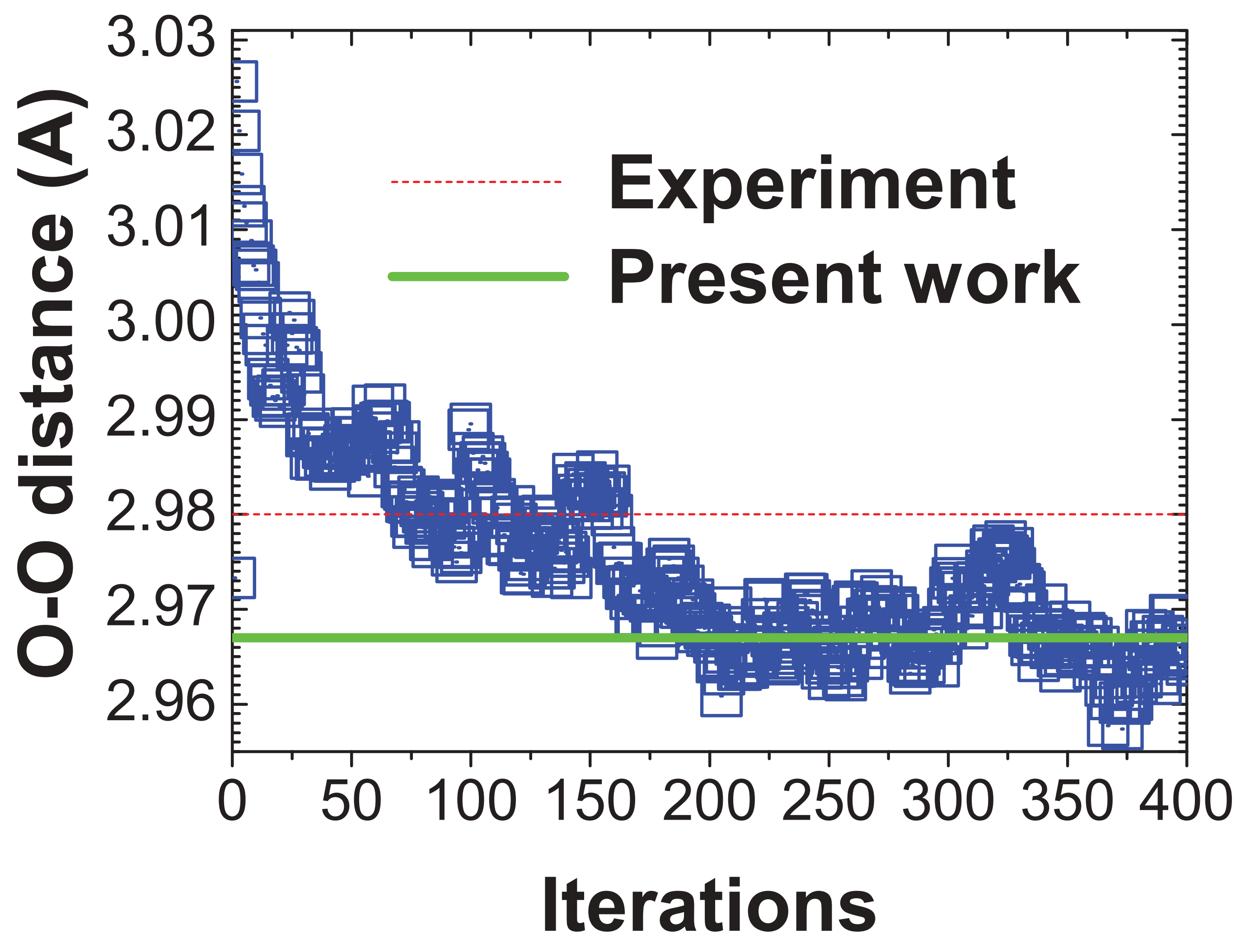
- Supports higher order gradients



```
>>> 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()
```

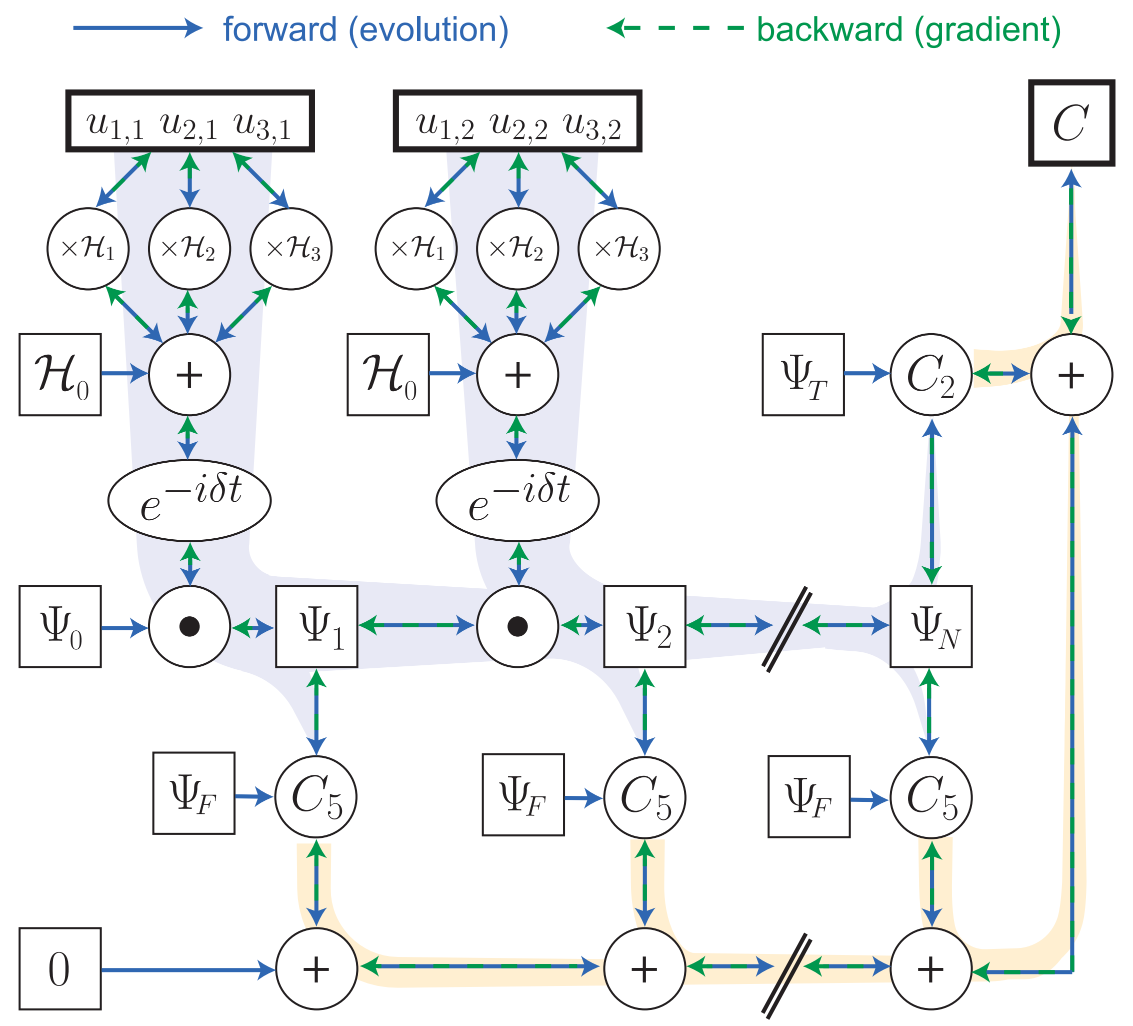
Applications of AD

Computing force



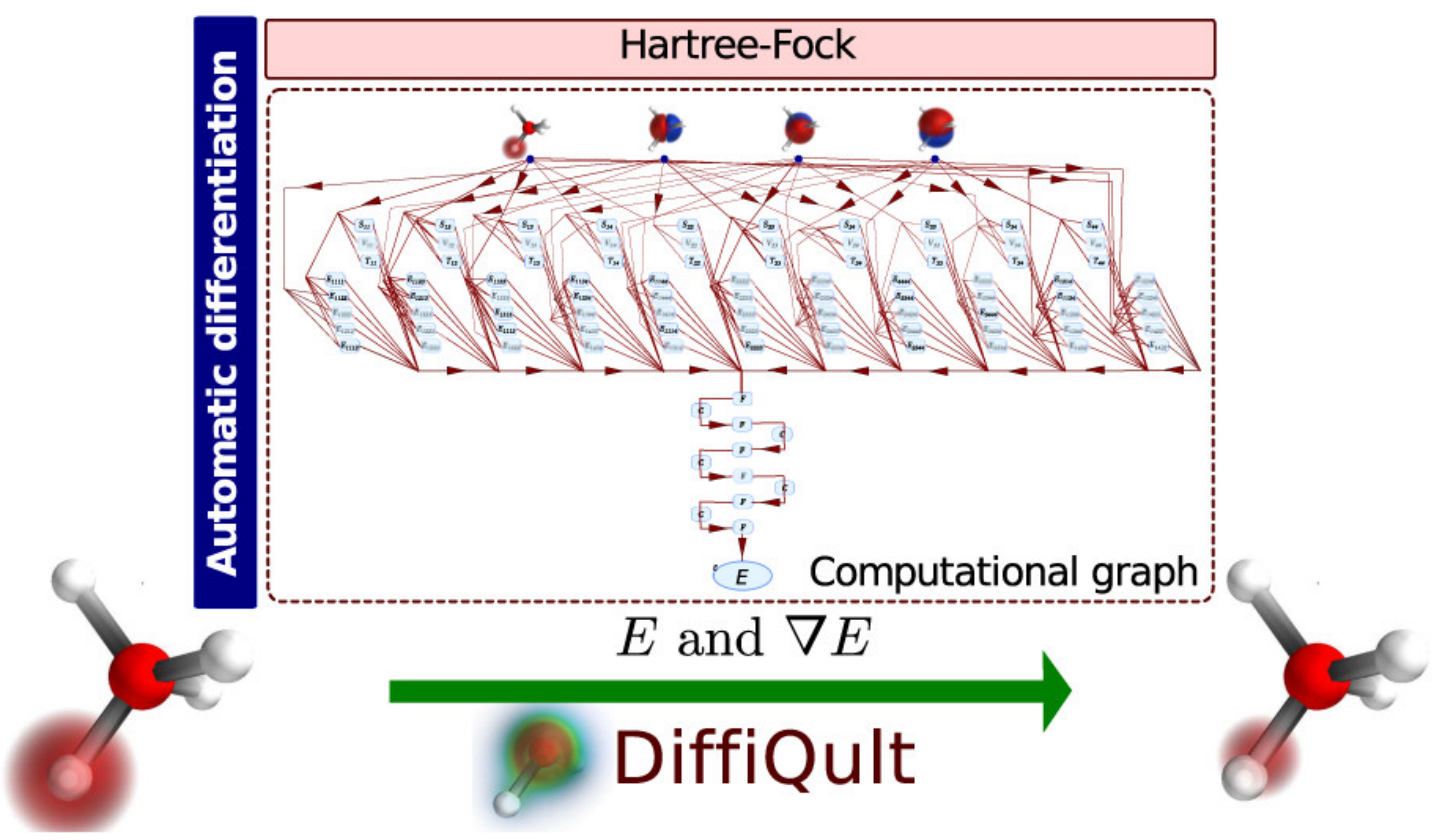
Sorella and Capriotti
J. Chem. Phys. '10

Quantum optimal control



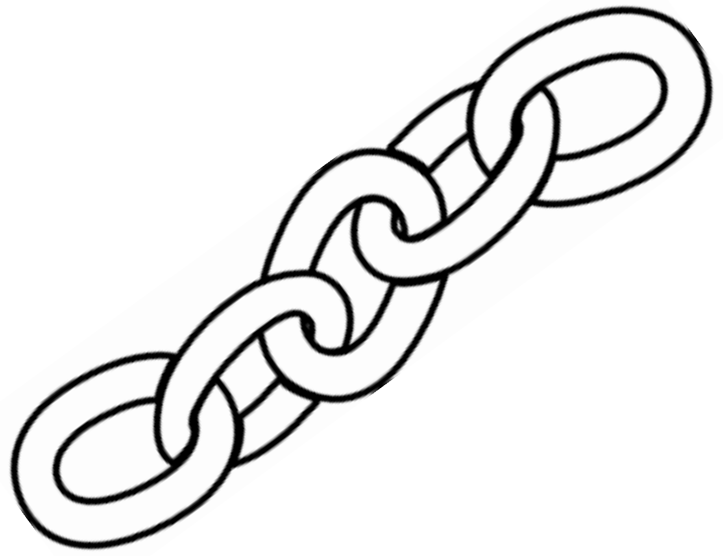
Leung et al
PRA '17

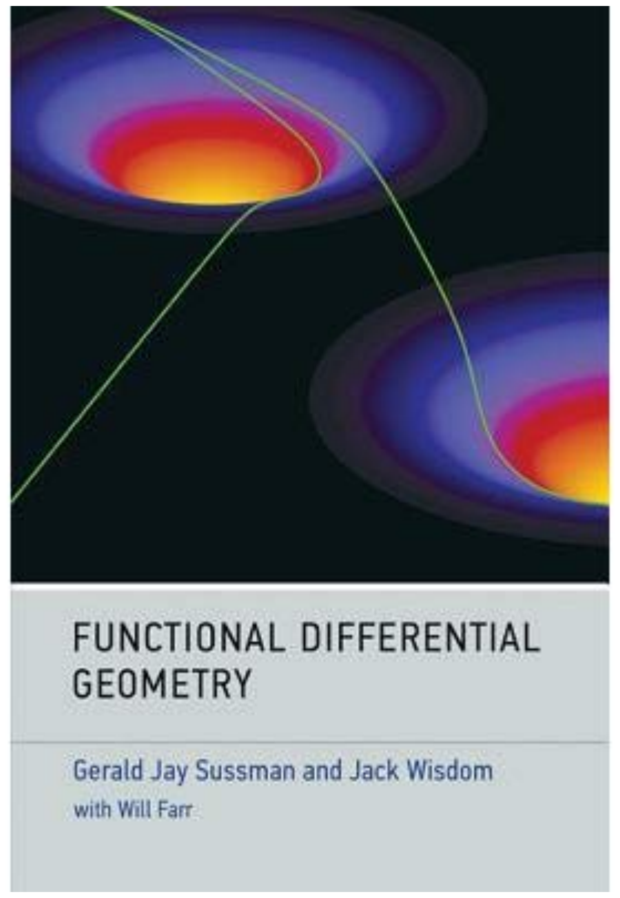
Variational Hartree-Fock



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

Understandings of AD





Black magic box

Chain rule

Functional differential geometry

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

Reverse versus forward mode

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

Reverse mode AD: Vector-Jacobian Product of primitives

$$v_o \, (J)_{o \times i}$$

- 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

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

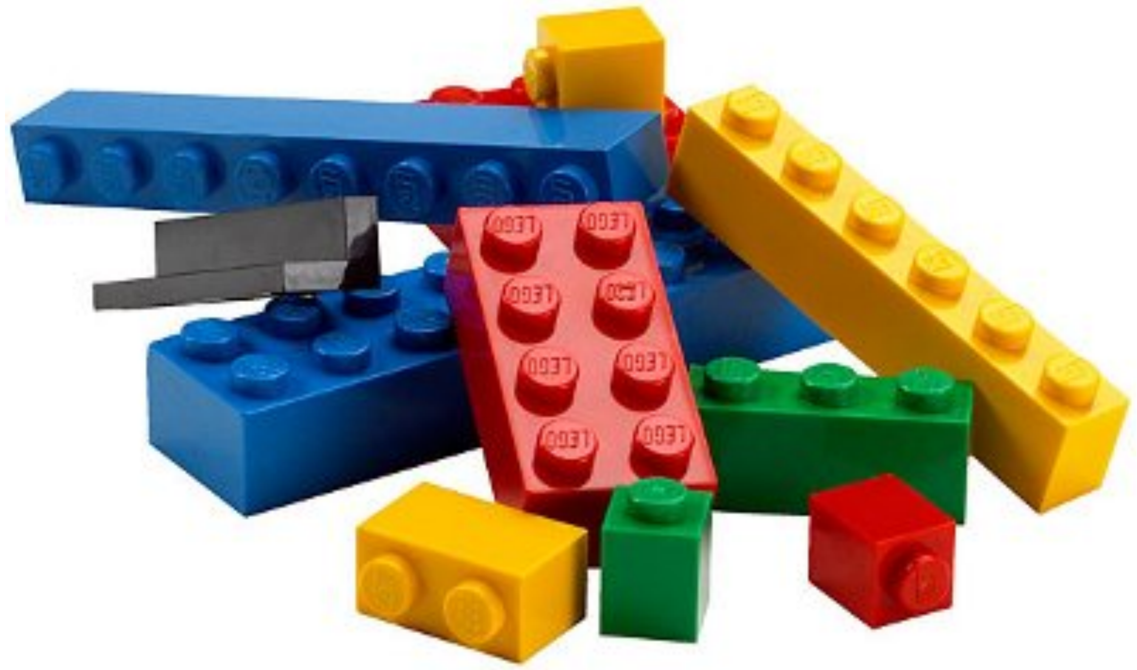
Forward mode AD: Jacobian-Vector Product of primitives

- Same order with the function evaluation $(J)_{o\times i} v_i$
- 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 to external libraries written agnostically to AD
    (or, even a quantum processor)



PENNYLANE

Example of the 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

Differentiable programming tools

HIPS/autograd

theano

flux

PyTorch

TensorFlow

Zygote

Keras

Current support for AD*

	linalg	complex	GPU	mixed-mode
PyTorch	✓	✗	✓	✗
TensorFlow	✓	🐞	✓	✗
Autograd	✓	🐞	✗	✗
Jax	✓	🐞	✓	✓
Flux.jl/Zygote.jl	✗	✓	✓	✓

*as of July 2019

Jin-Guo's blog post https://giggleliu.github.io/2019/04/02/einsumbp.html

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/Hartree-Fock/Coupled Cluster/Gutzwiller/ Molecular Dynamics…

Differentiable programming is more than training neural networks

Differentiable Eigensolver

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

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

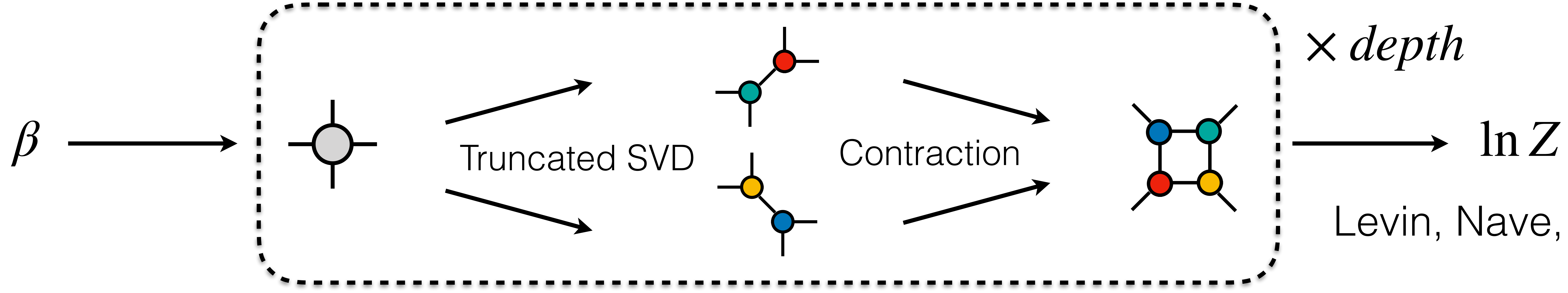
Reverse mode: How should I change H given $\partial\mathcal{L}/\partial\Psi$ and $\partial\mathcal{L}/\partial\Lambda$? **Inverse perturbation theory!**

Hamiltonian engineering via differentiable programming

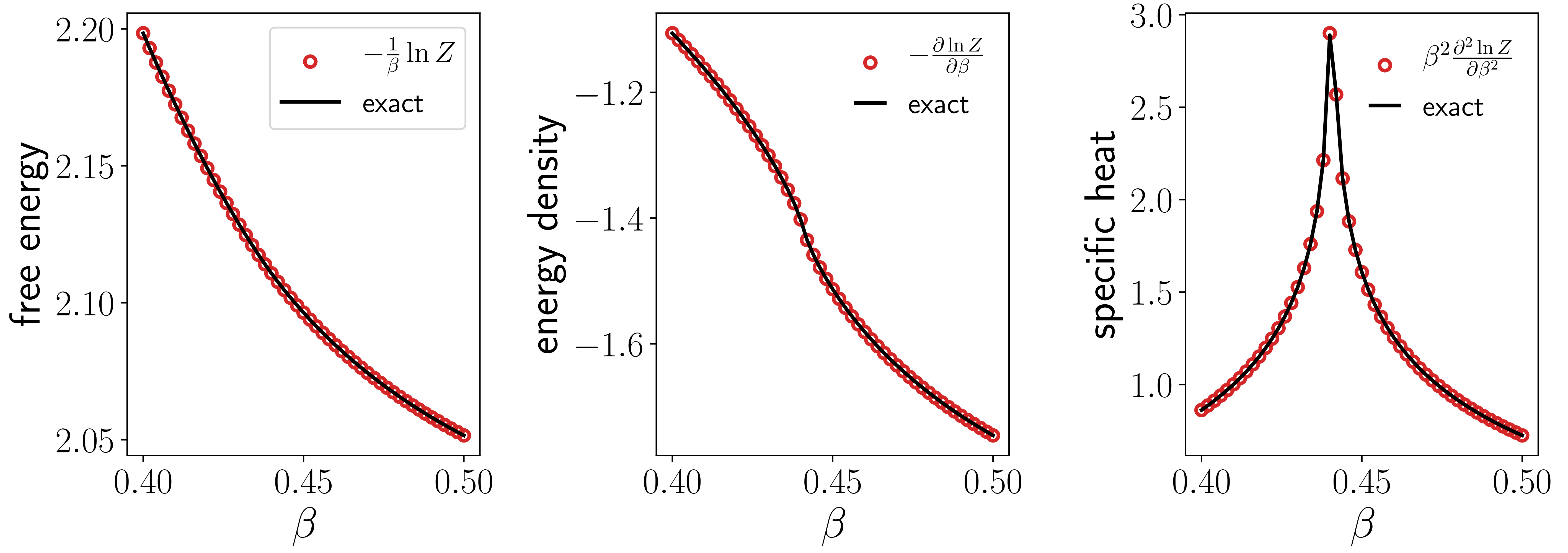
https://github.com/wangleiphy/DL4CSRC/tree/master/2-ising See also Fujita et al, PRB '18

Differentiate through TRG for Ising

Computation graph



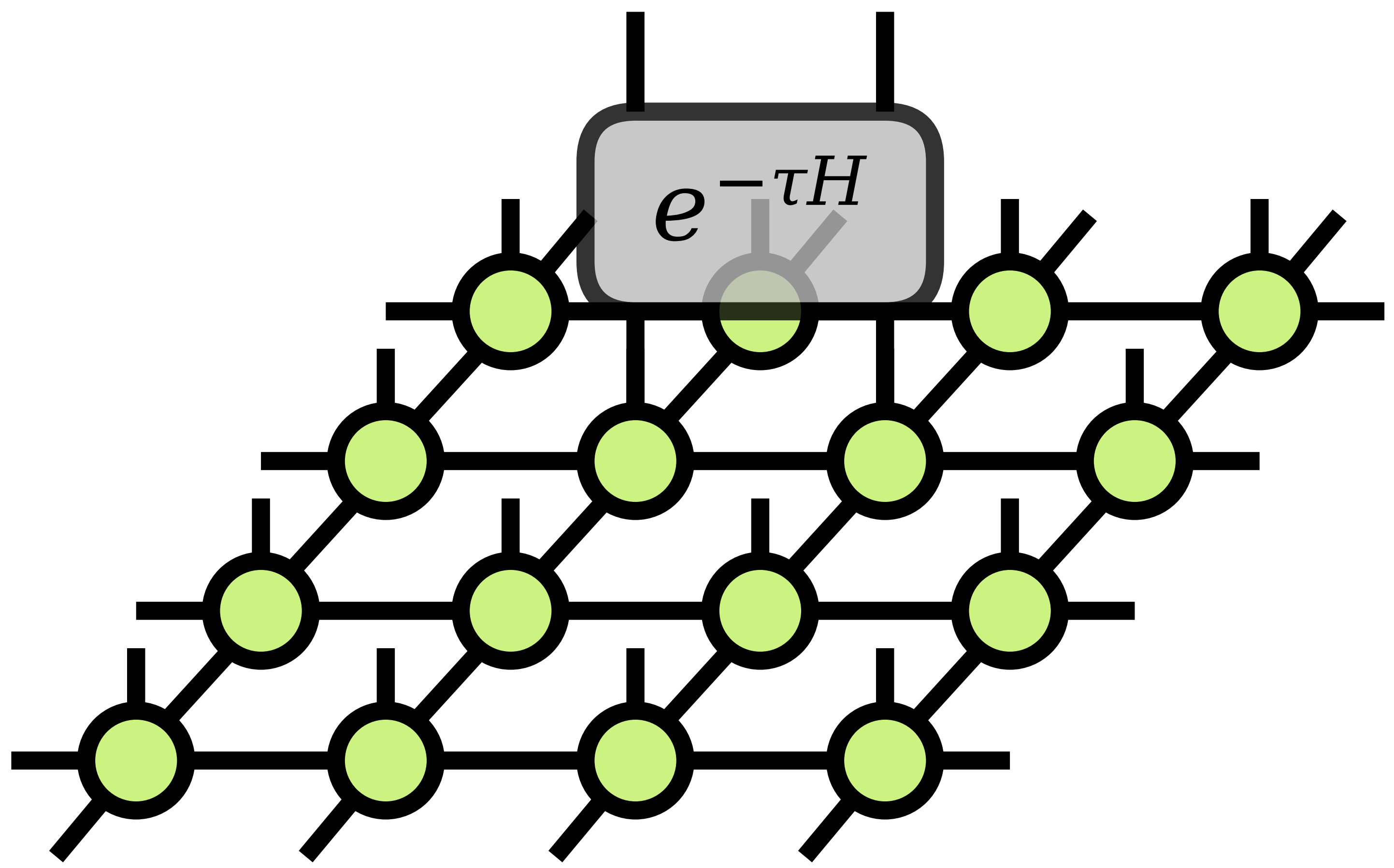
Levin, Nave, PRL '07

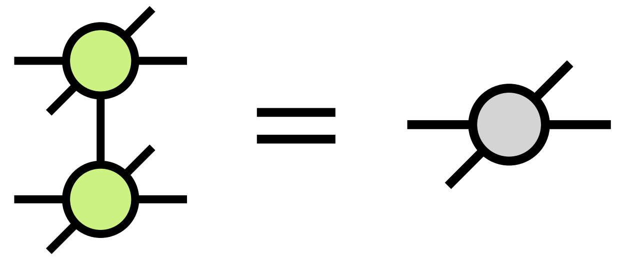


AD computes physical observables as high-order gradients

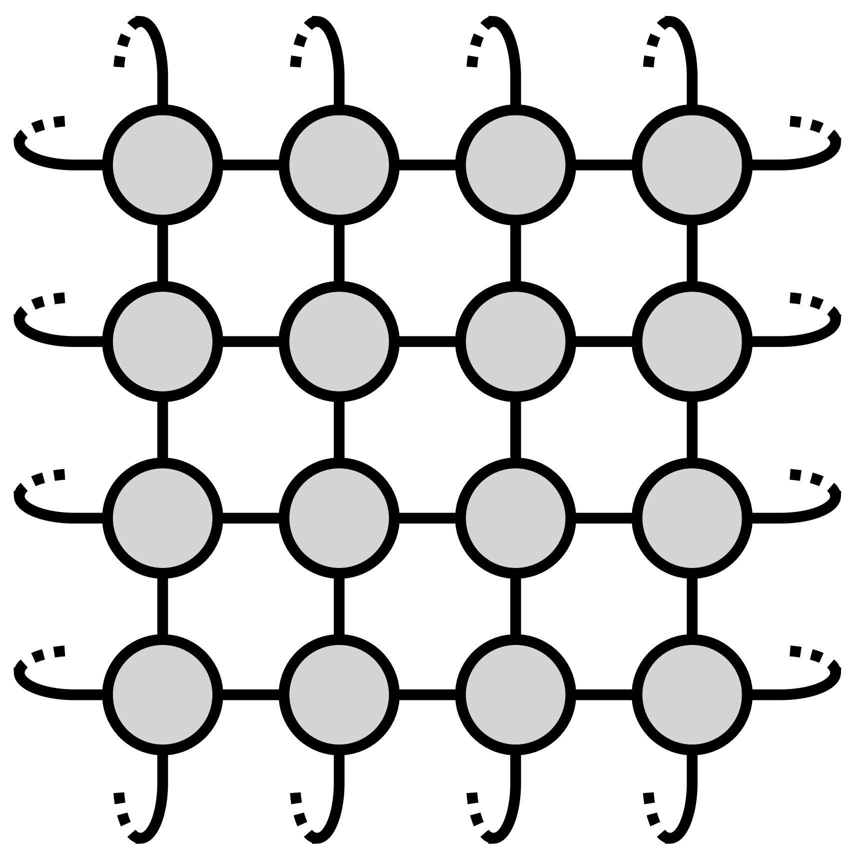
Tensor network quantum states

Optimization





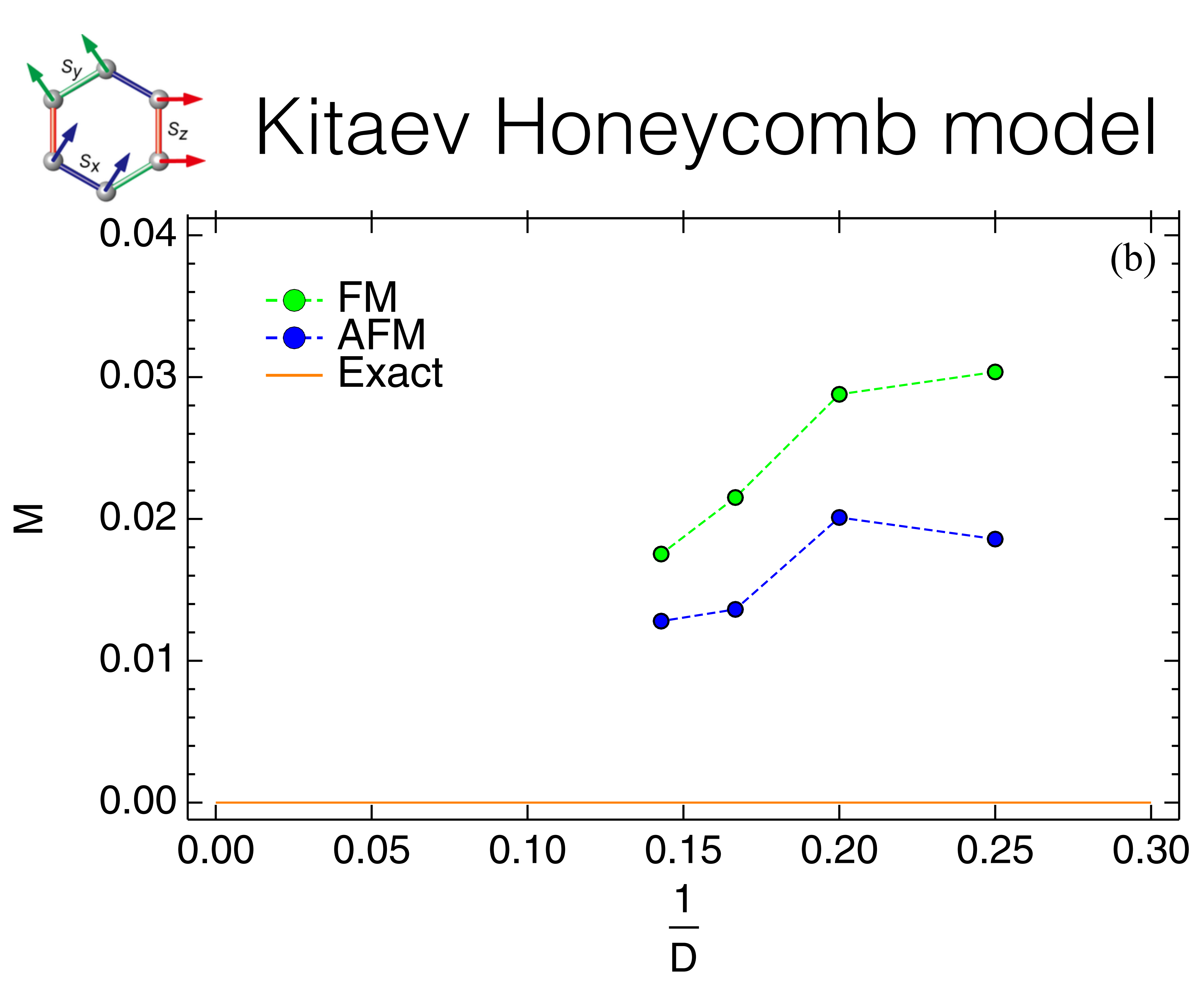
Contraction



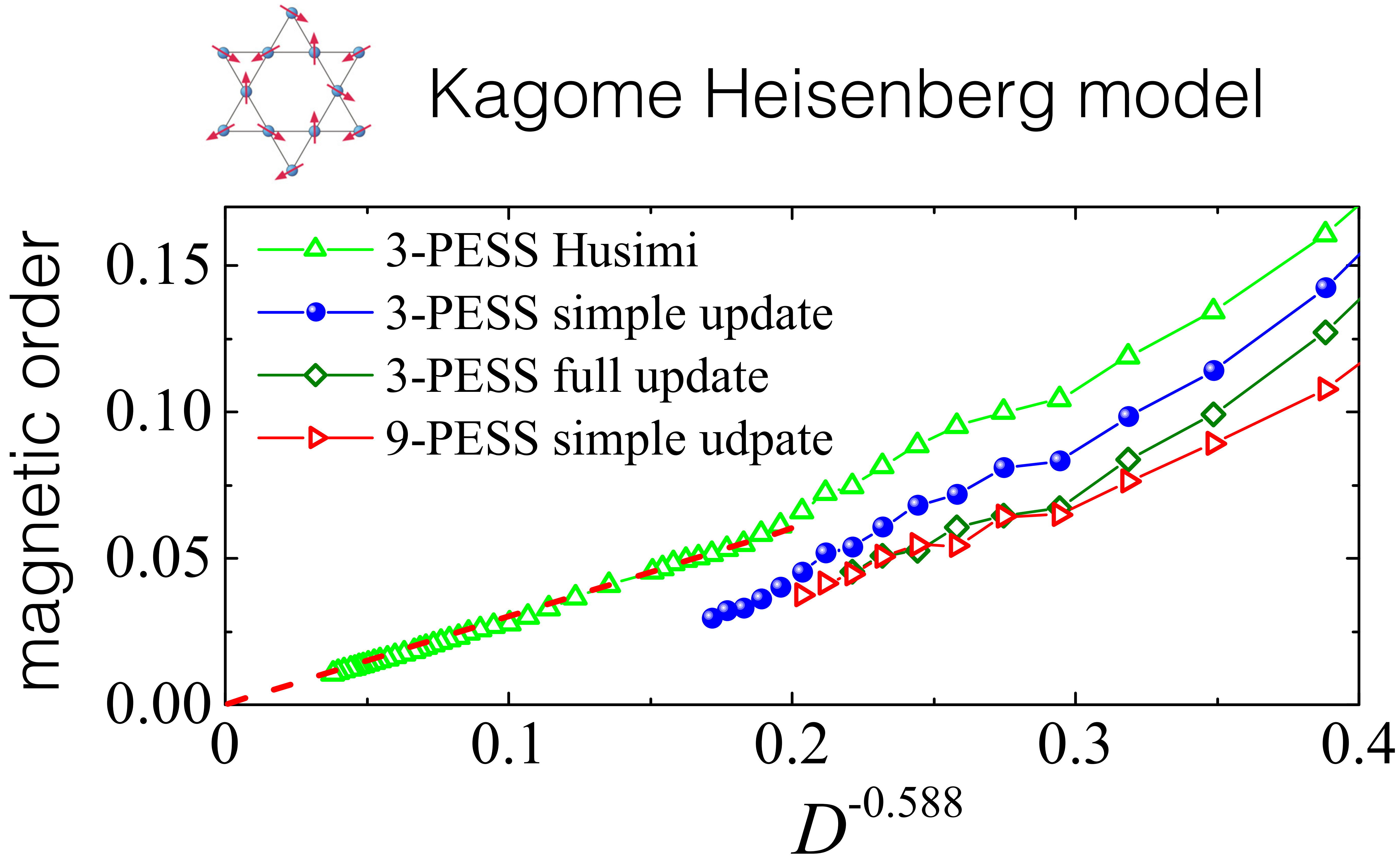
- Trotterized imaginary-time projection
- Update schemes: "simple", "full" "cluster", "faster full"…

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

Expressibility v.s. Optimization: an eternal problem



Osorio, Corboz, Troyer, PRB '14

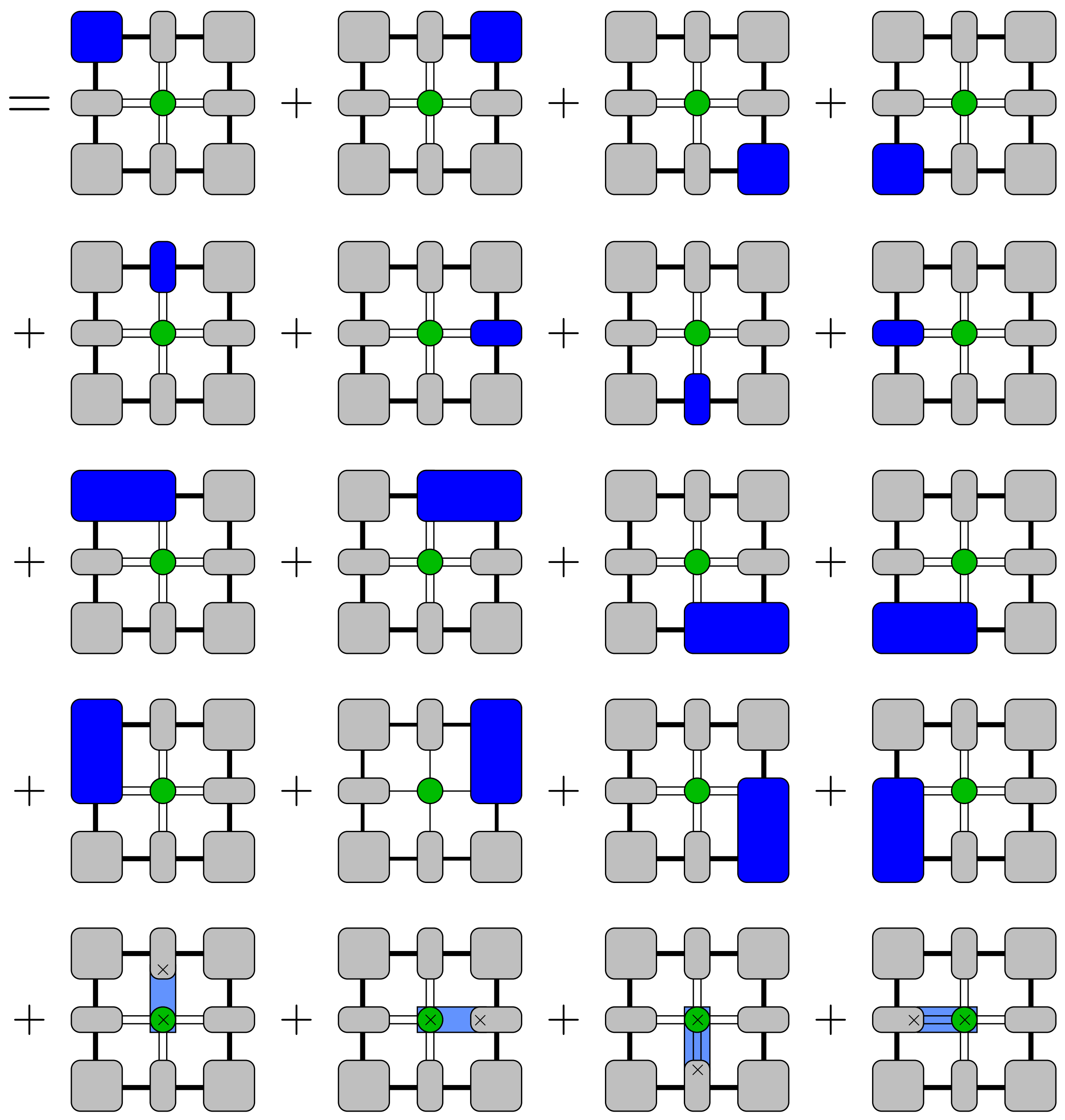


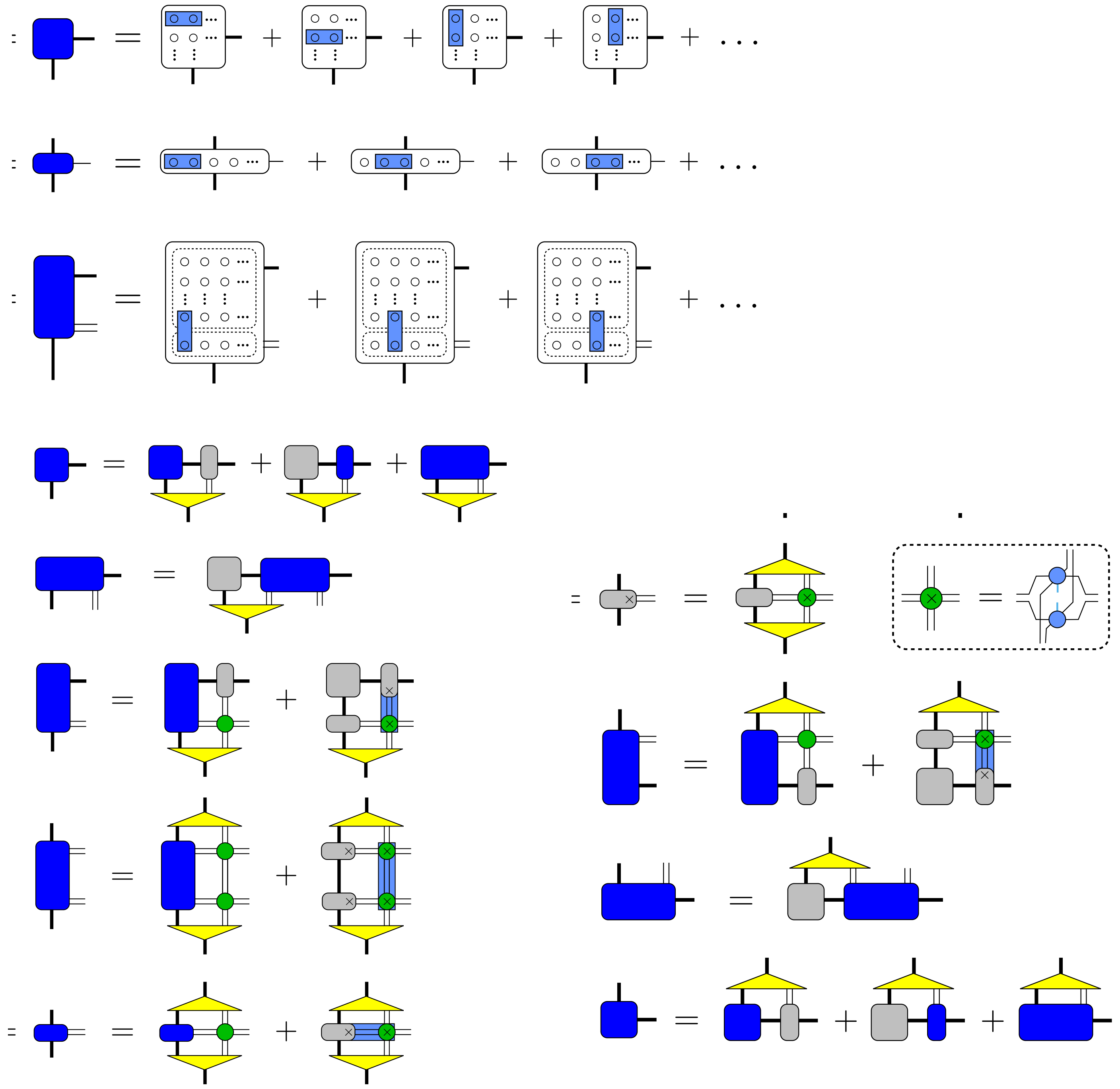
Liao et al, PRL '17

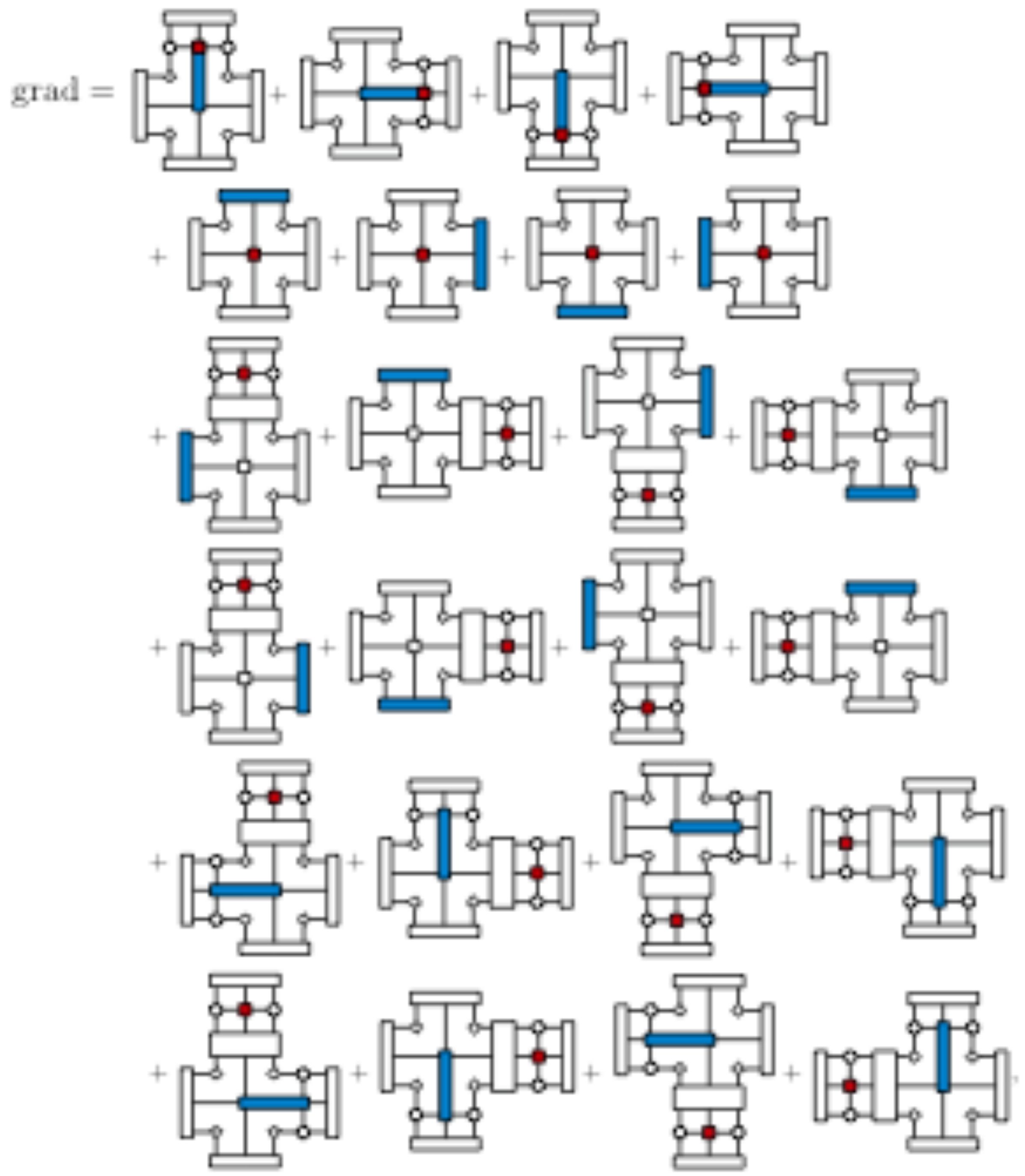
Typically, one finds ordered states at small D and tries hard to push up the bond dimensions

Variational optimization infinite tensor networks

$$\mathscr{L}_{\boldsymbol{\theta}} = \langle \Psi_{\boldsymbol{\theta}} | \hat{H} | \Psi_{\boldsymbol{\theta}} \rangle / \langle \Psi_{\boldsymbol{\theta}} | \Psi_{\boldsymbol{\theta}} \rangle$$







Corboz, PRB '16 Vanderstraeten et al, PRB '16

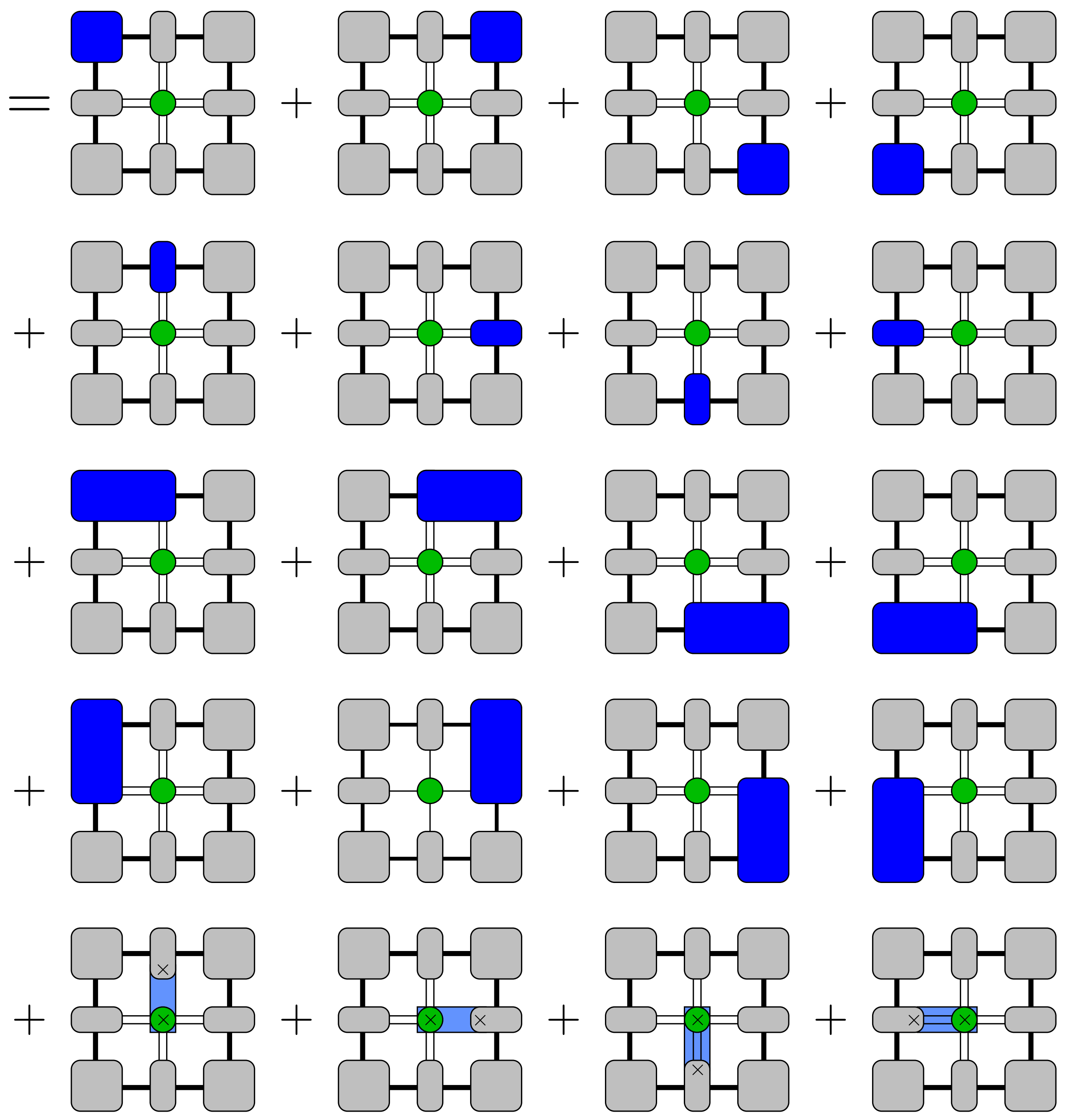
Variational optimization with gradient indeed help!

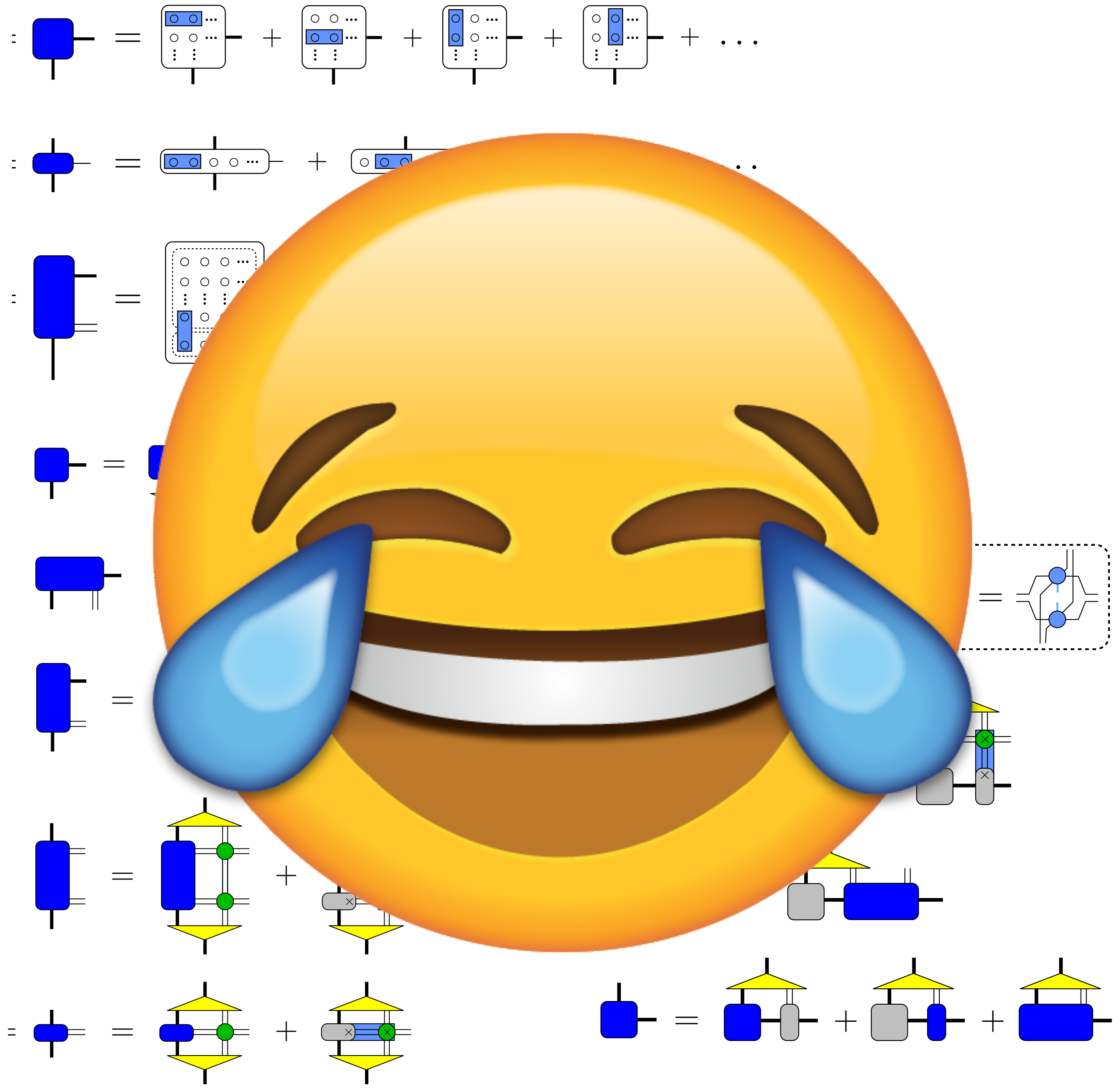
However, manually deriving gradients is cumbersome

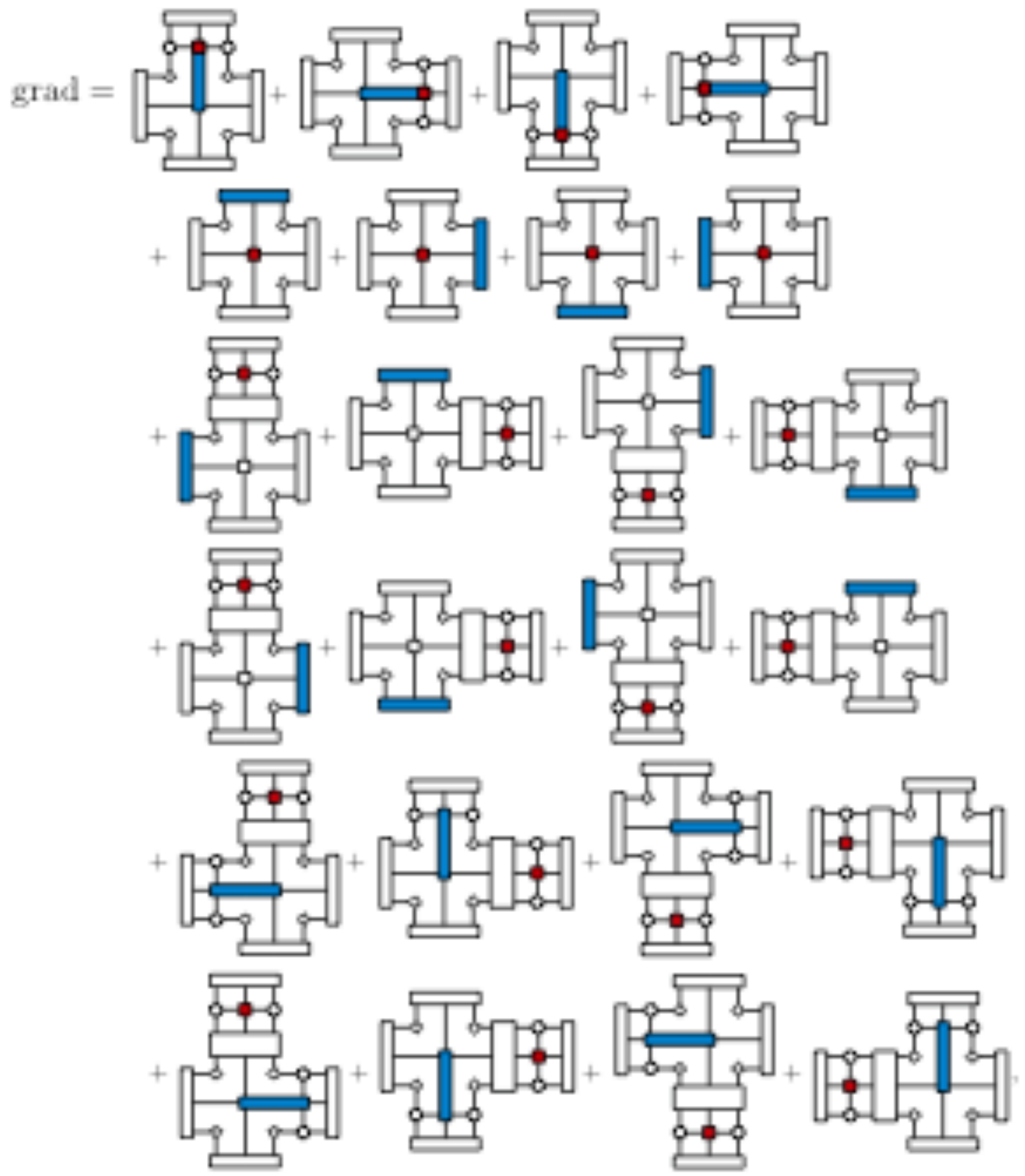
Corboz et al, PRX '18
Rader et al, PRX '18

Variational optimization infinite tensor networks

$$\mathscr{L}_{\theta} = \langle \Psi_{\theta} | \hat{H} | \Psi_{\theta} \rangle / \langle \Psi_{\theta} | \Psi_{\theta} \rangle$$







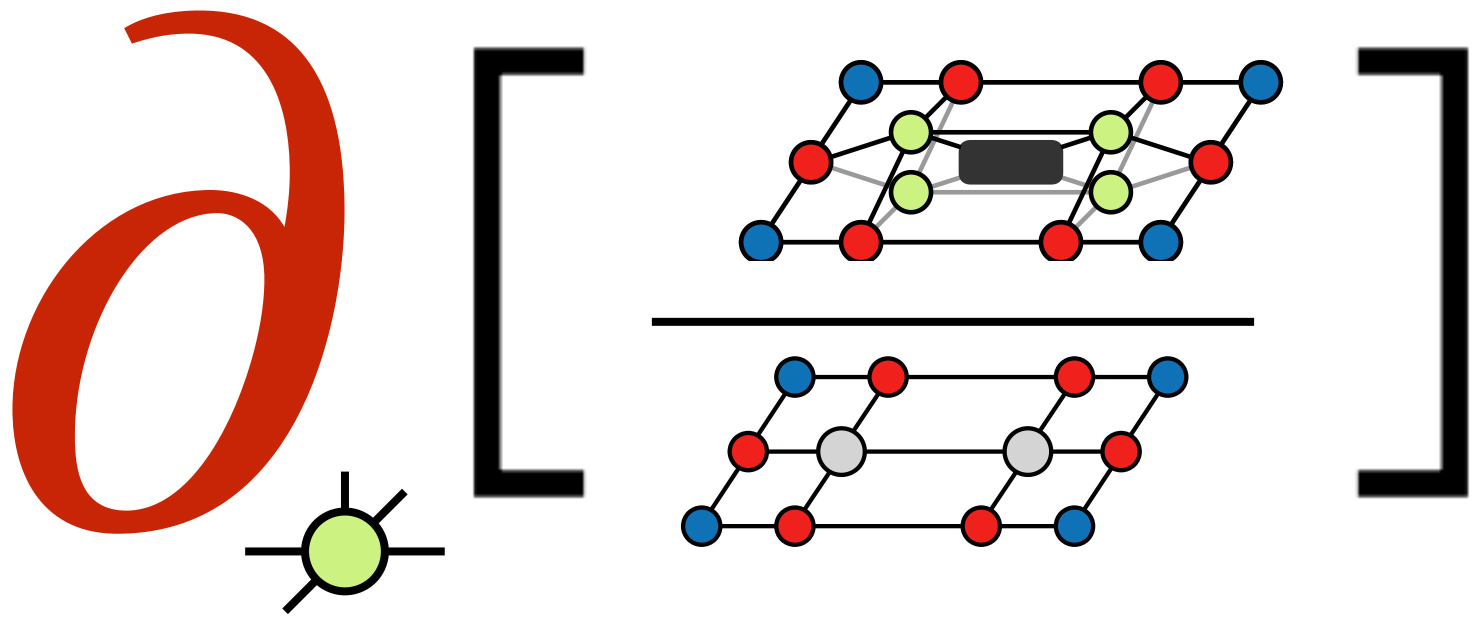
Corboz, PRB ’16 Vanderstraeten et al, PRB ’16

Variational optimization with gradient indeed help!
However, manually deriving gradients is cumbersome

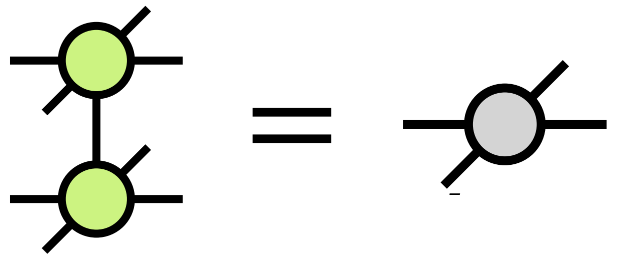
Corboz et al, PRX ’18
Rader et al, PRX ‘18

Automatic differentiation to the rescue

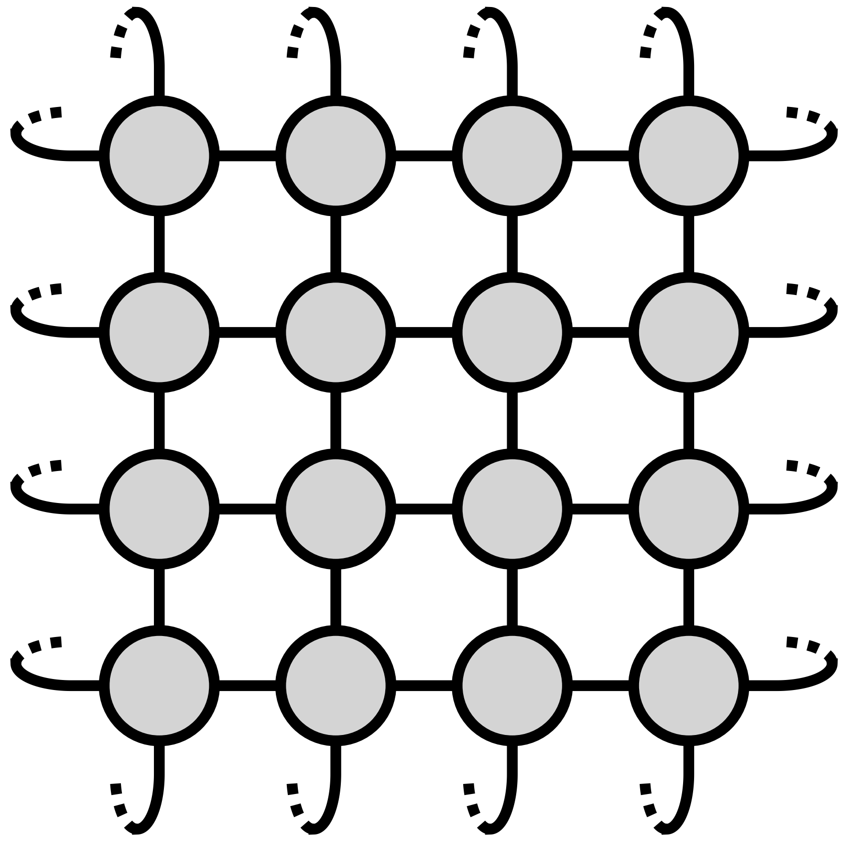
Optimization



conjugate-gradient, quasi-Newton, etc



Contraction

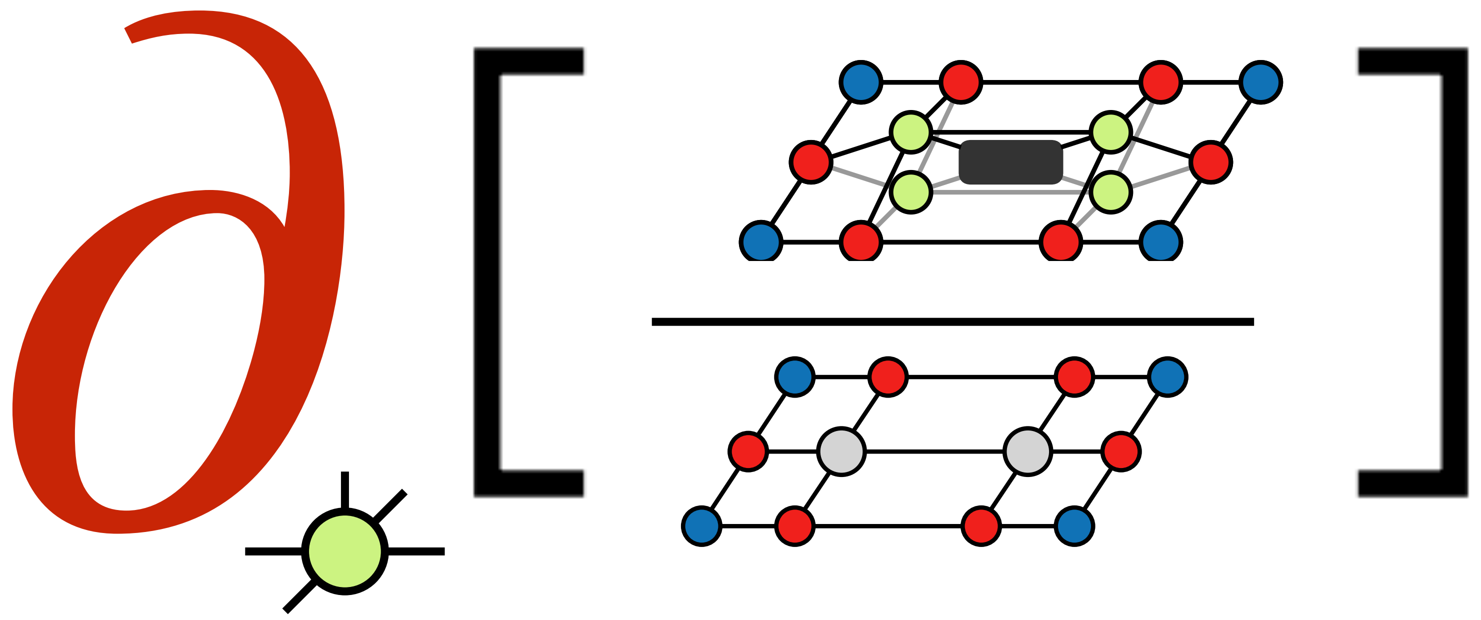


Any lattice, any Hamiltonian, any contraction scheme

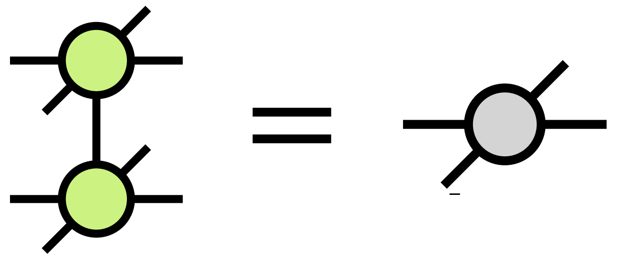
Human only cares about tensor contraction
Differentiable programing takes care of the optimization

Automatic differentiation to the rescue

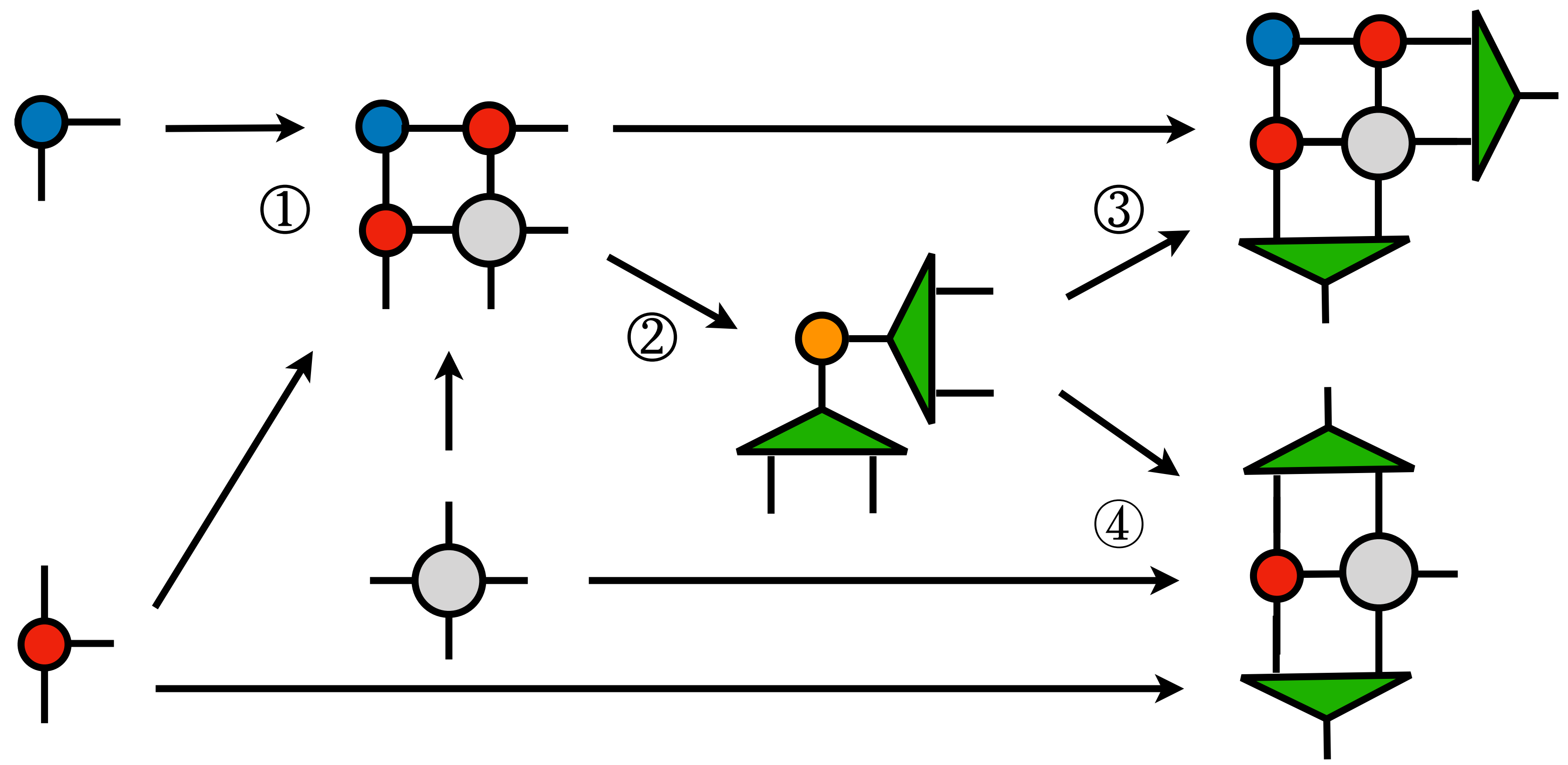
Optimization



conjugate-gradient, quasi-Newton, etc



Contraction



CTMRG, Nishino, Okunishi, JPSJ, '95

Any lattice, any Hamiltonian, any contraction scheme

Human only cares about tensor contraction
Differentiable programing takes care of the optimization

Nuts and Bolts

- Numerical stable backward through SVD

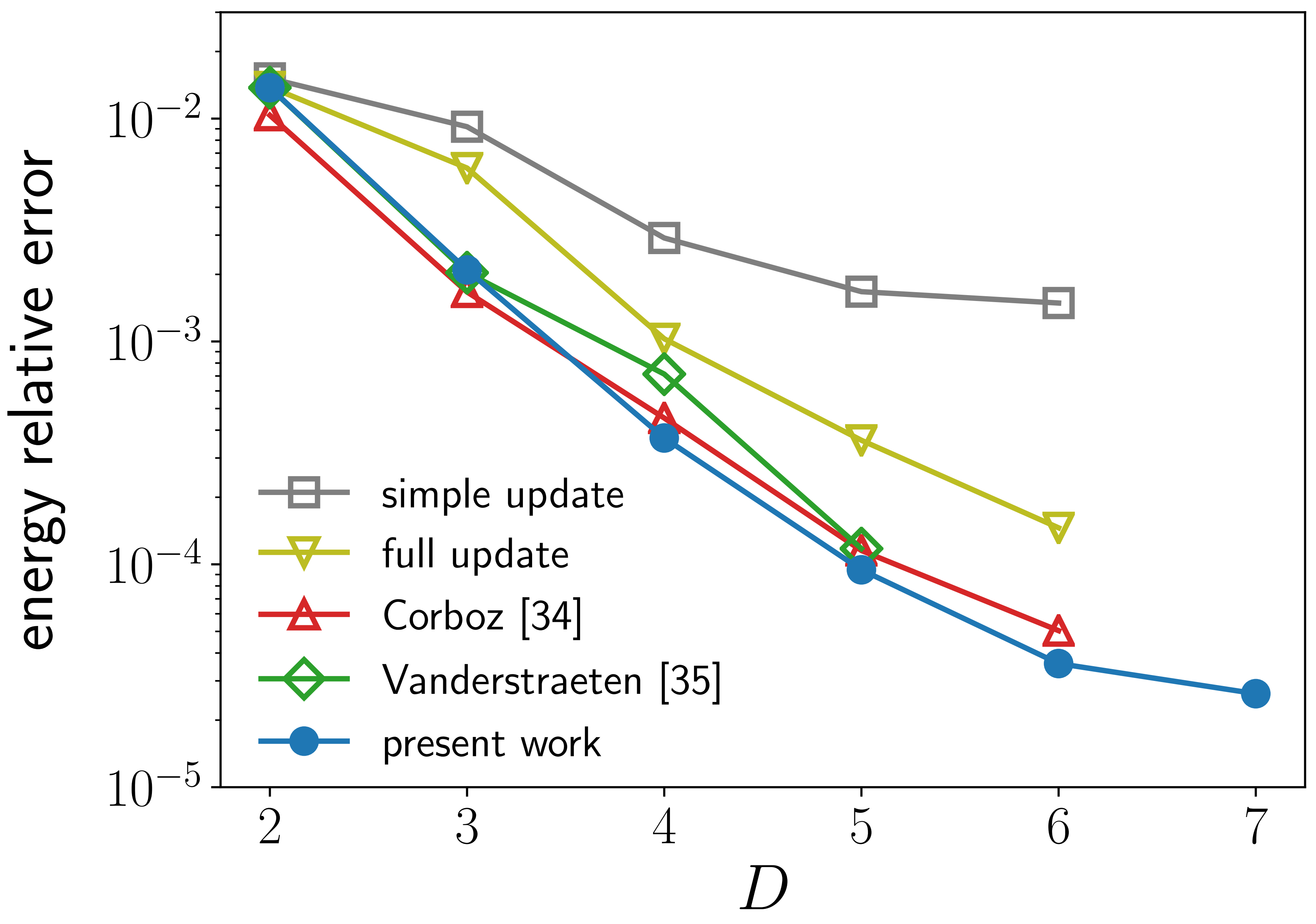
$$A = UDV^T \qquad \overline{A} \overset{?}{\longleftarrow} \overline{U}, \overline{D}, \overline{V}$$

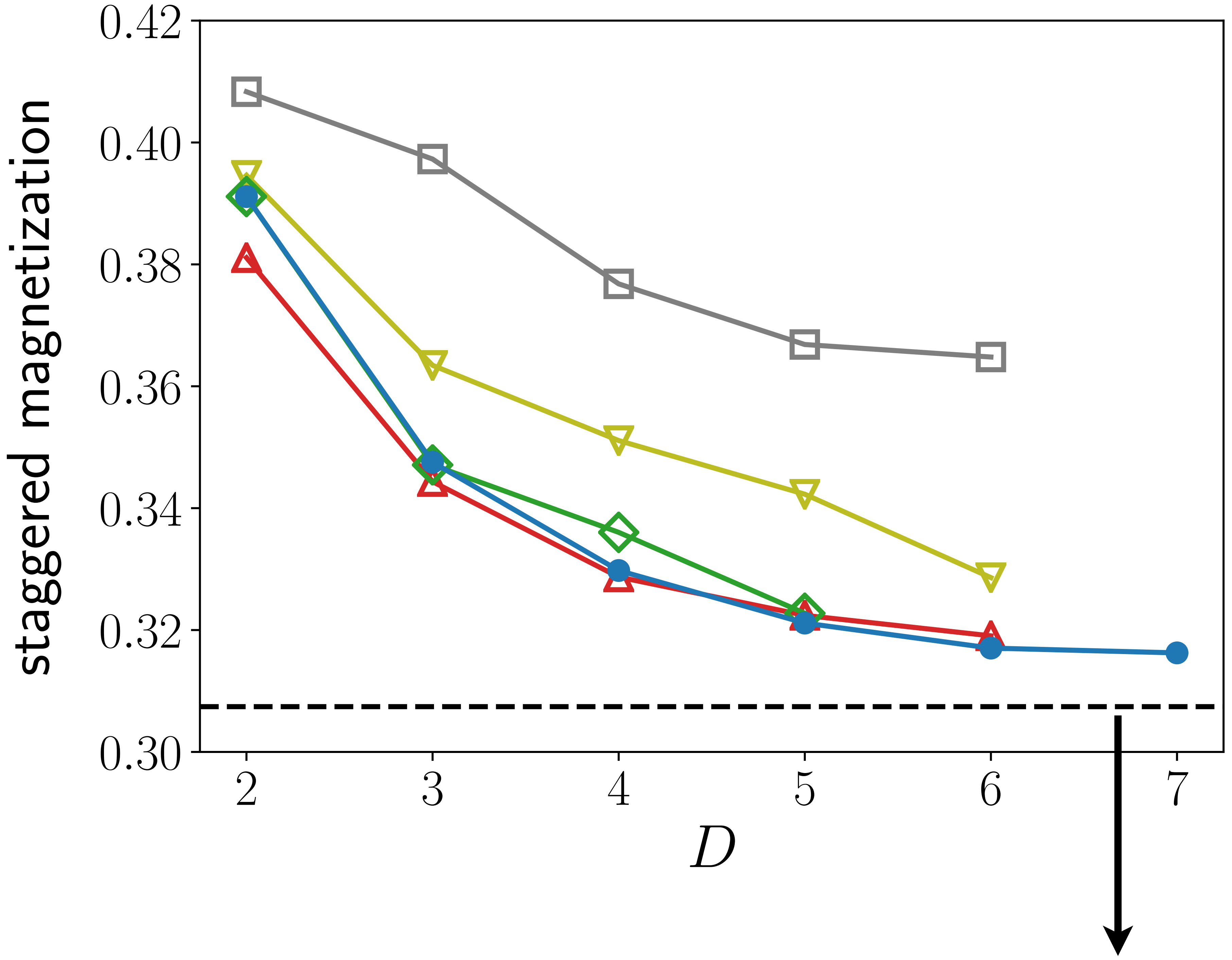
- Reduce memory via checkpointing or exploiting RG fixed point property

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

Liao, Liu, LW, Xiang, 1903.09650

Square lattice Heisenberg model





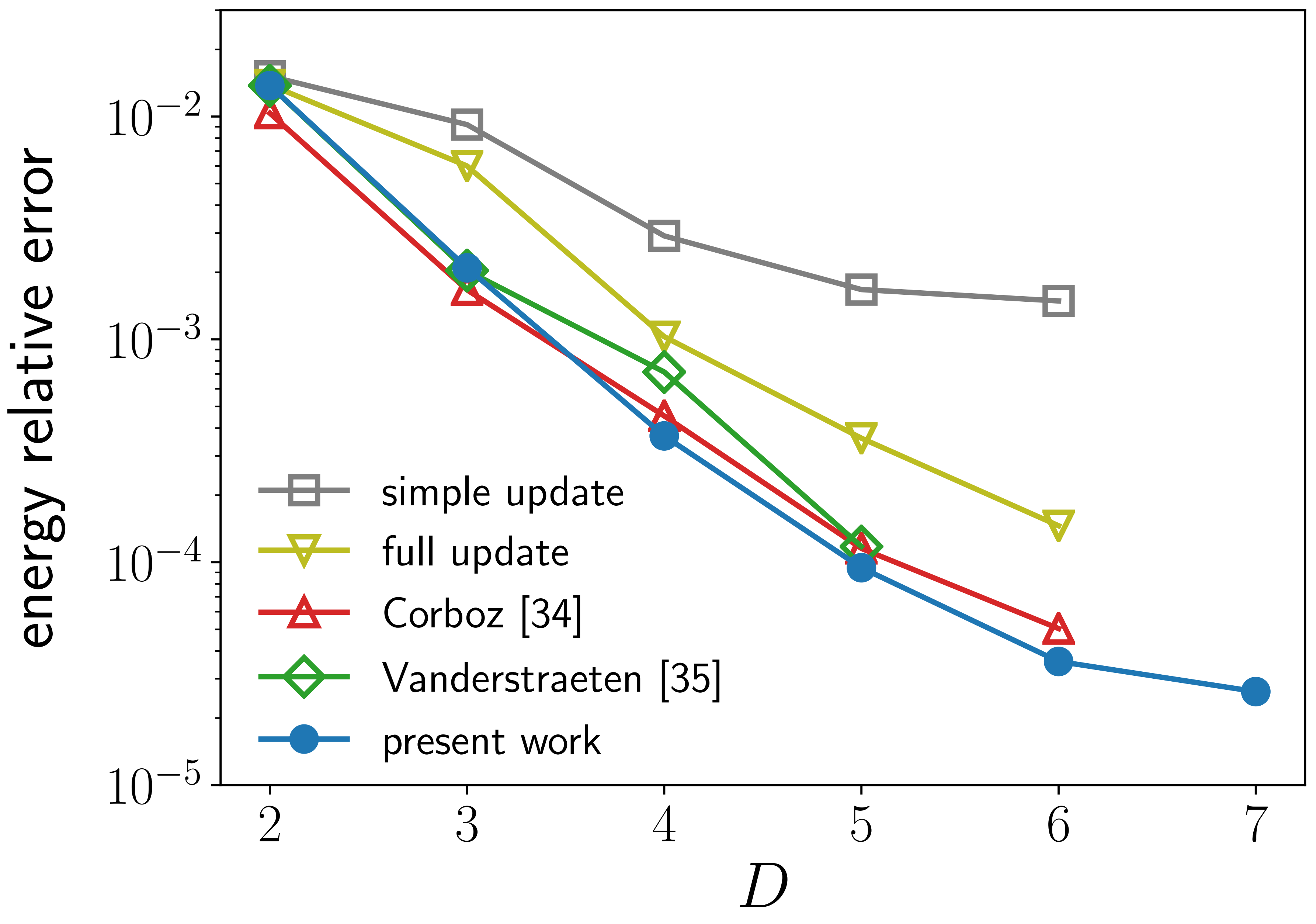
Liao, Liu, LW, Xiang, 1903.09650

Extrapolated QMC, Sandvik '10

Square lattice Heisenberg model

Infinite size

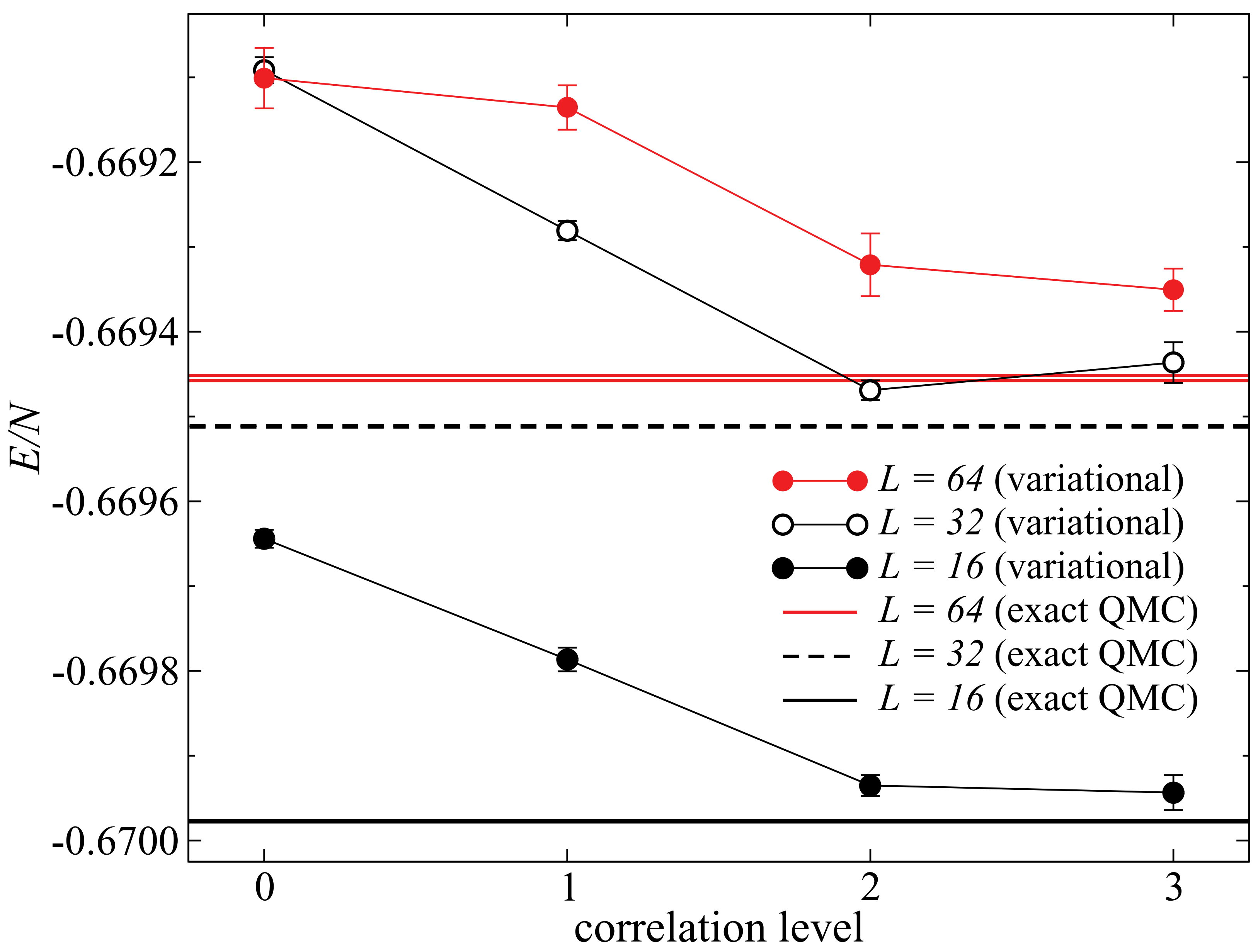
AD optimized iPEPS



Liao, Liu, LW, Xiang, 1903.09650

Finite size

VMC of an RVB-type state



1×10^{-4} @L=64

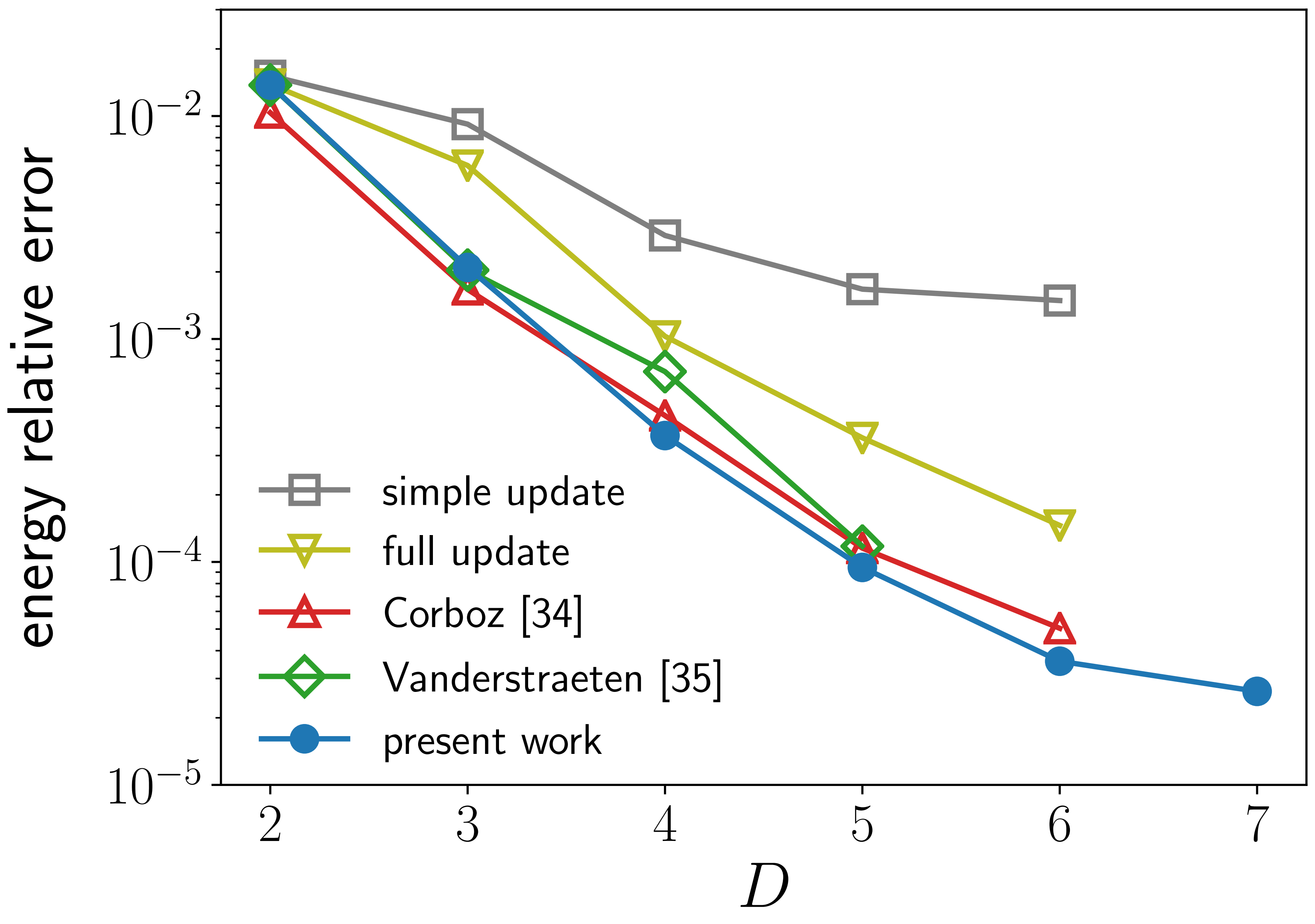
4×10^{-5} @L=16

Lin, Tang, Sandvik, PRB '12

Square lattice Heisenberg model

Infinite size

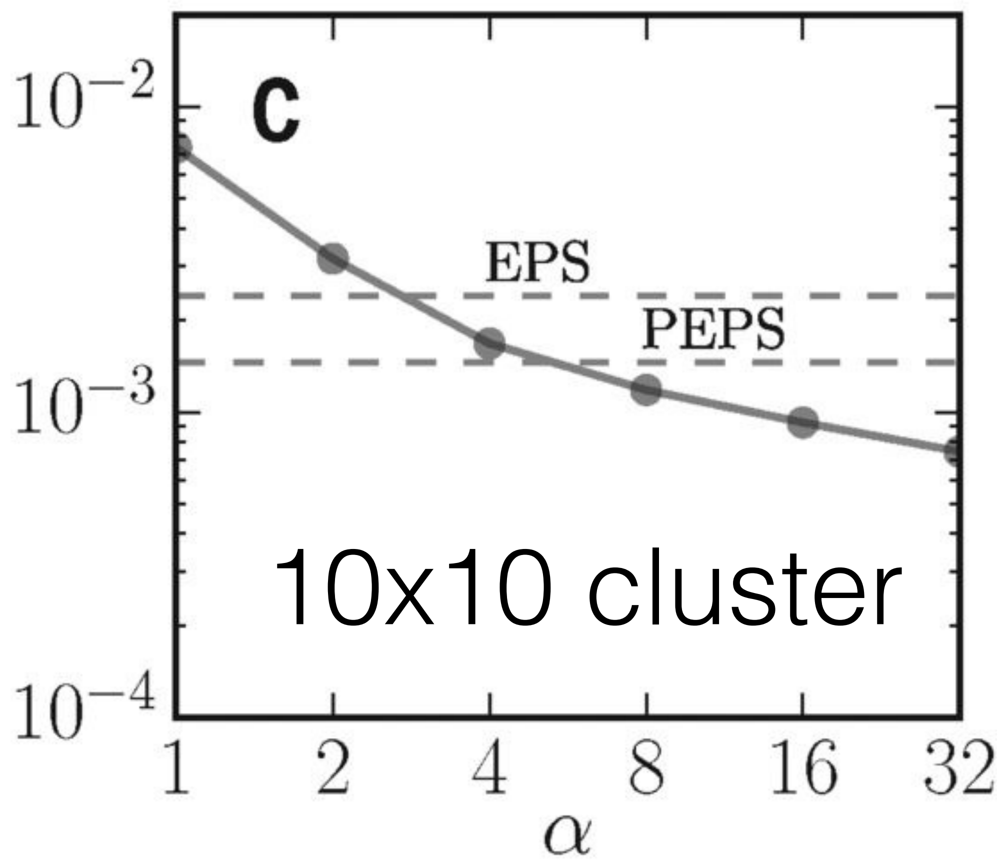
AD optimized iPEPS



Liao, Liu, LW, Xiang, 1903.09650

Finite size

VMC optimized RBM

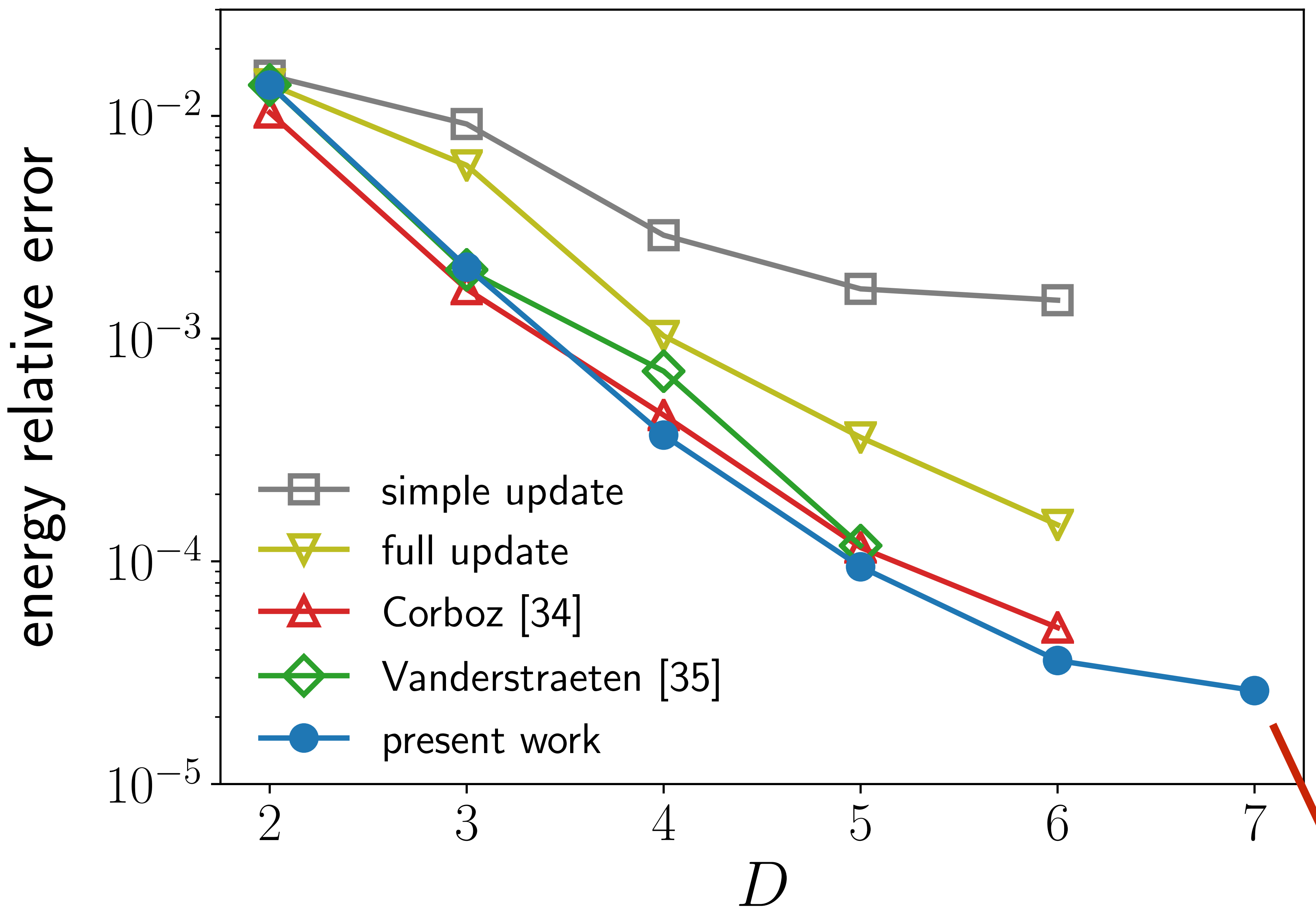


Carleo & Troyer, Science ‘17

Square lattice Heisenberg model

Infinite size

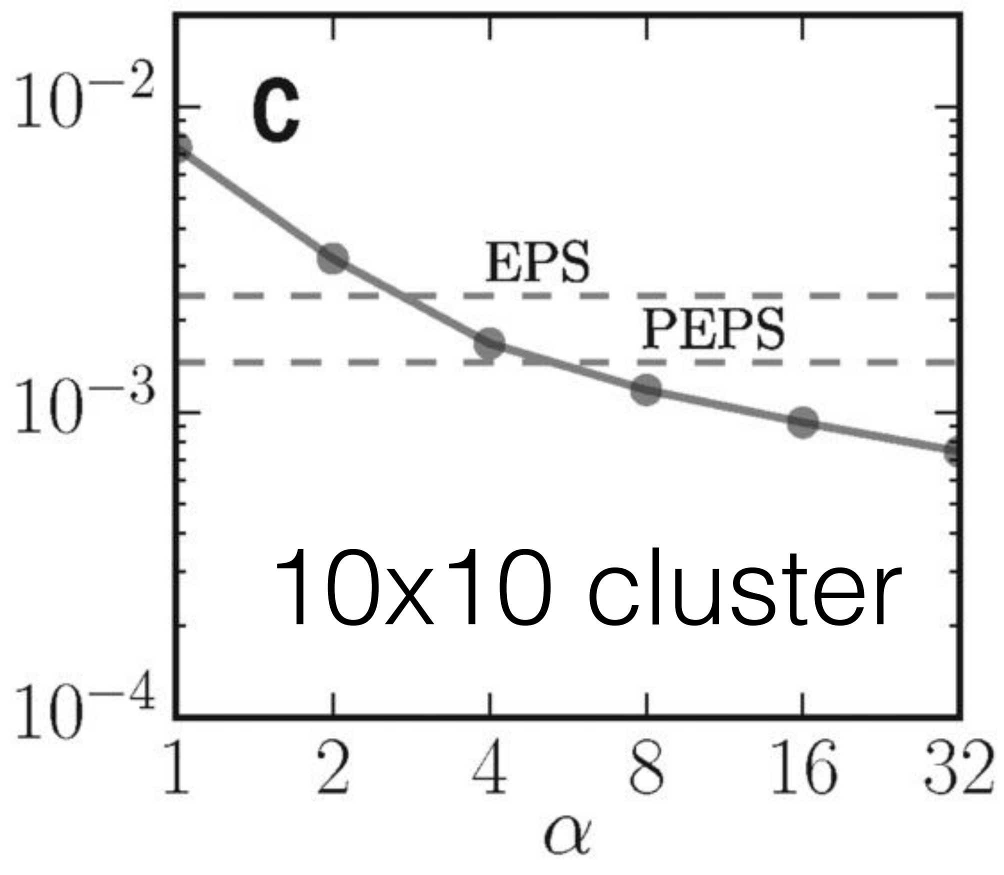
AD optimized iPEPS



Liao, Liu, LW, Xiang, 1903.09650

Finite size

VMC optimized RBM

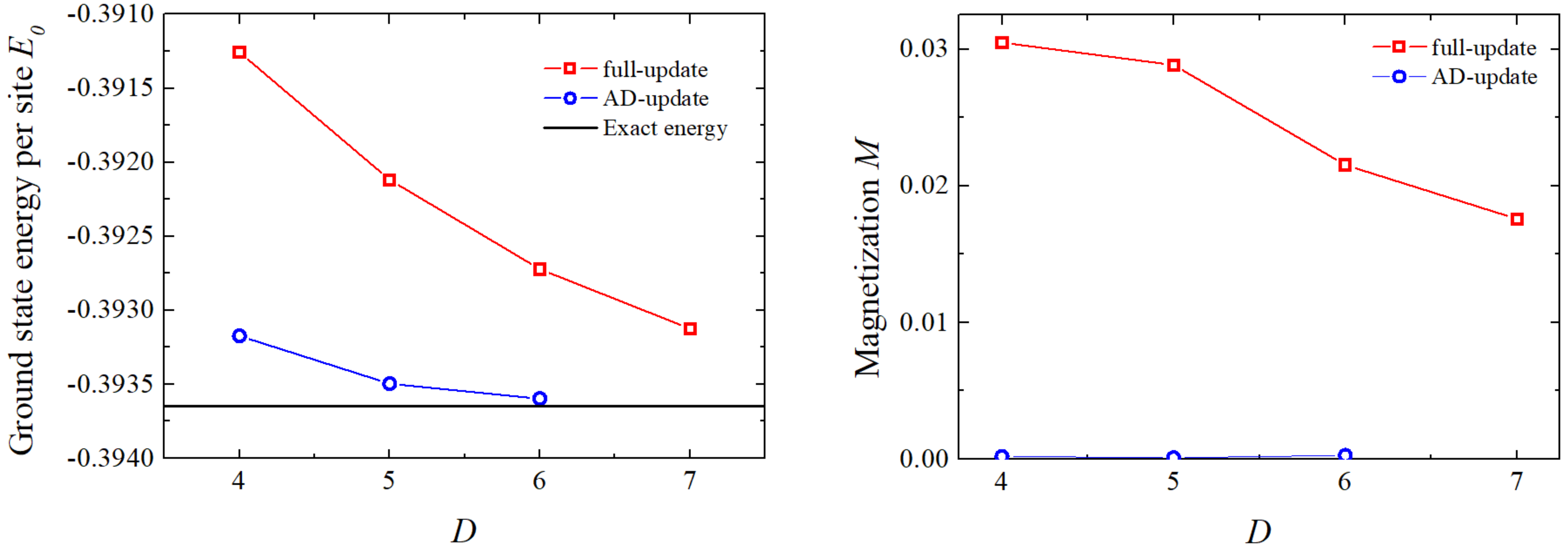


Carleo & Troyer, Science '17

Lowest variational energy for infinite system

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

Kitaev honeycomb model



Reaches lower energy even at smaller bond dimensions with substantially reduced magnetic order

c.f. analytically constructed iPEPS, Lee et al, 1901.05786

The morals

- iPEPS with small bond dimensions are more expressive than we thought. We just did not optimize them hard enough

- Differentiable programming tensor networks has a bright future: variational contraction, gauge fixing, fermions…

Google summer of code
Andreas Peter mentored by Jin-Guo Liu

https://github.com/under-Peter/TensorNetworkAD.jl

The morals

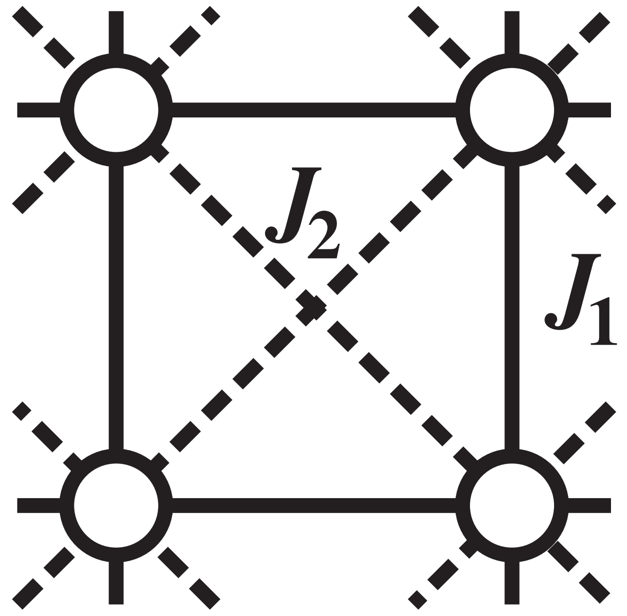
- iPEPS with small bond dimensions are more expressive than we thought. We just did not optimize them hard enough

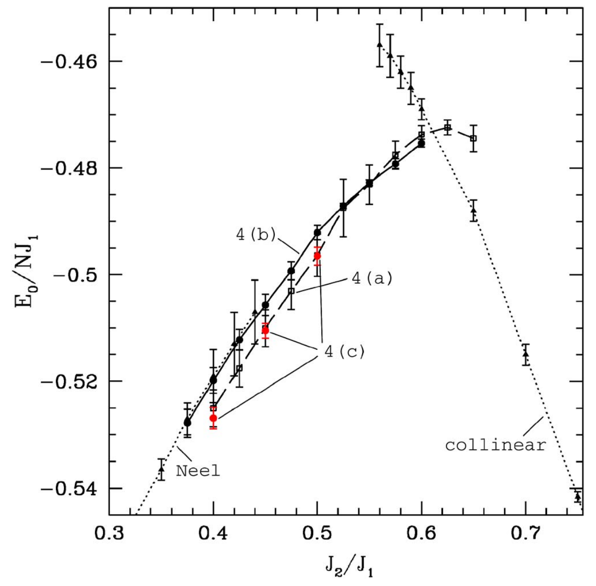
- Differentiable programming tensor networks has a bright future: variational contraction, gauge fixing, fermions…

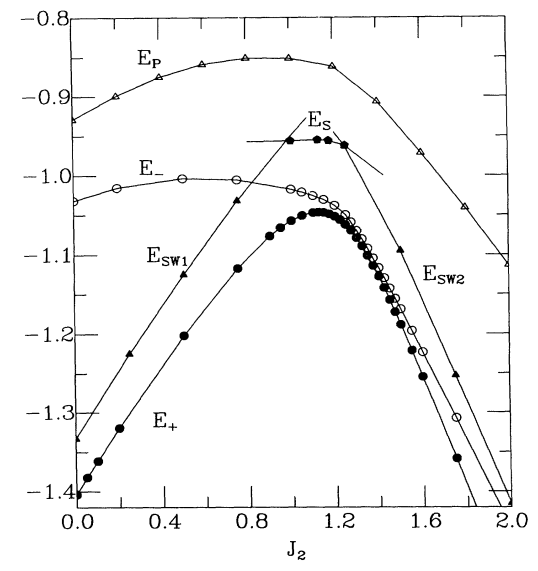
Google summer of code
Andreas Peter mentored by Jin-Guo Liu
https://github.com/under-Peter/TensorNetworkAD.jl

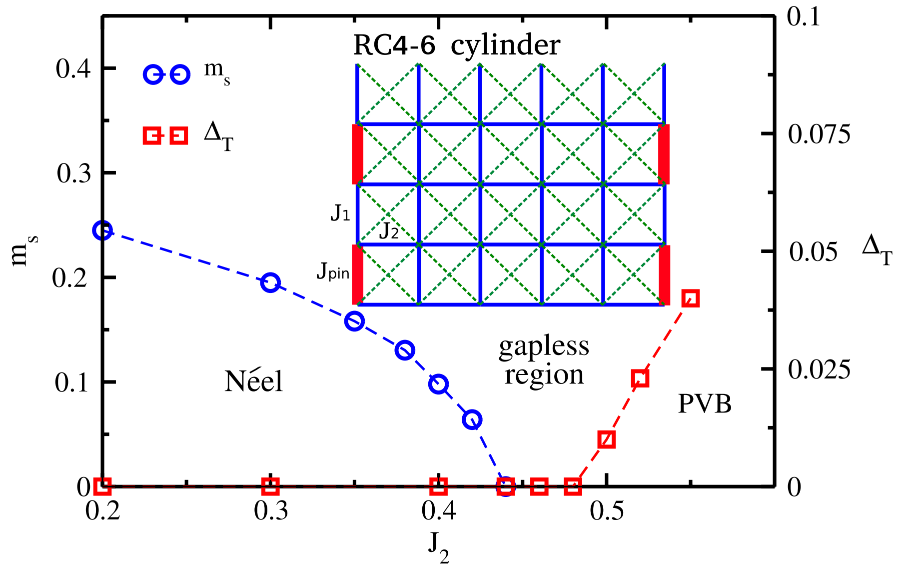
- BTW, the difficulty of optimizing neural network quantum states with VMC: stochastic optimization with correlated samples and poor gradient estimator (potentially can also be fixed by ML).

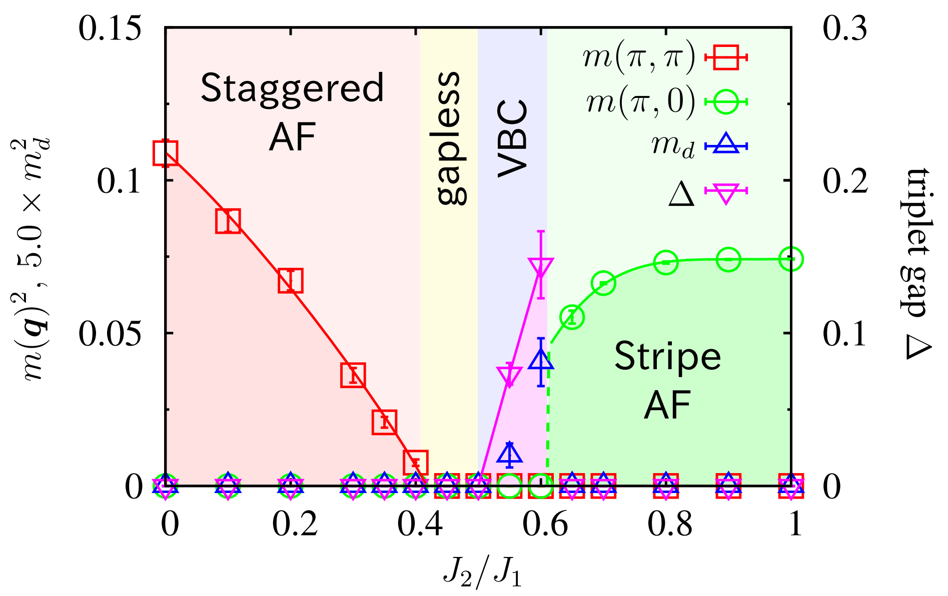
Ground state phase diagram of the J1-J2 model

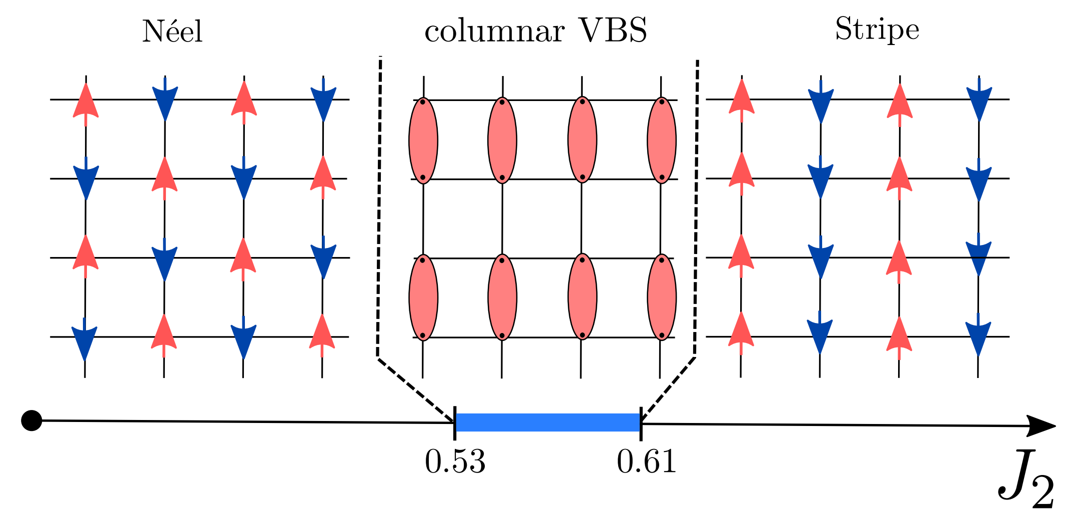












Exact Diagonalization
Diagotto et al
PRL '89

Series Expansion
Sirker et al
PRB '06

DMRG
Gong et al
PRL '14

VMC
Morita et al
JPSJ '15

iPEPS
Haghshenas et al
PRB '19

+ many many others

Ground state phase diagram of the J1-J2 model

?

J_2/J_1

Ground state phase diagram of the J1-J2 model

IOP, CAS

Hai-Jun Liao

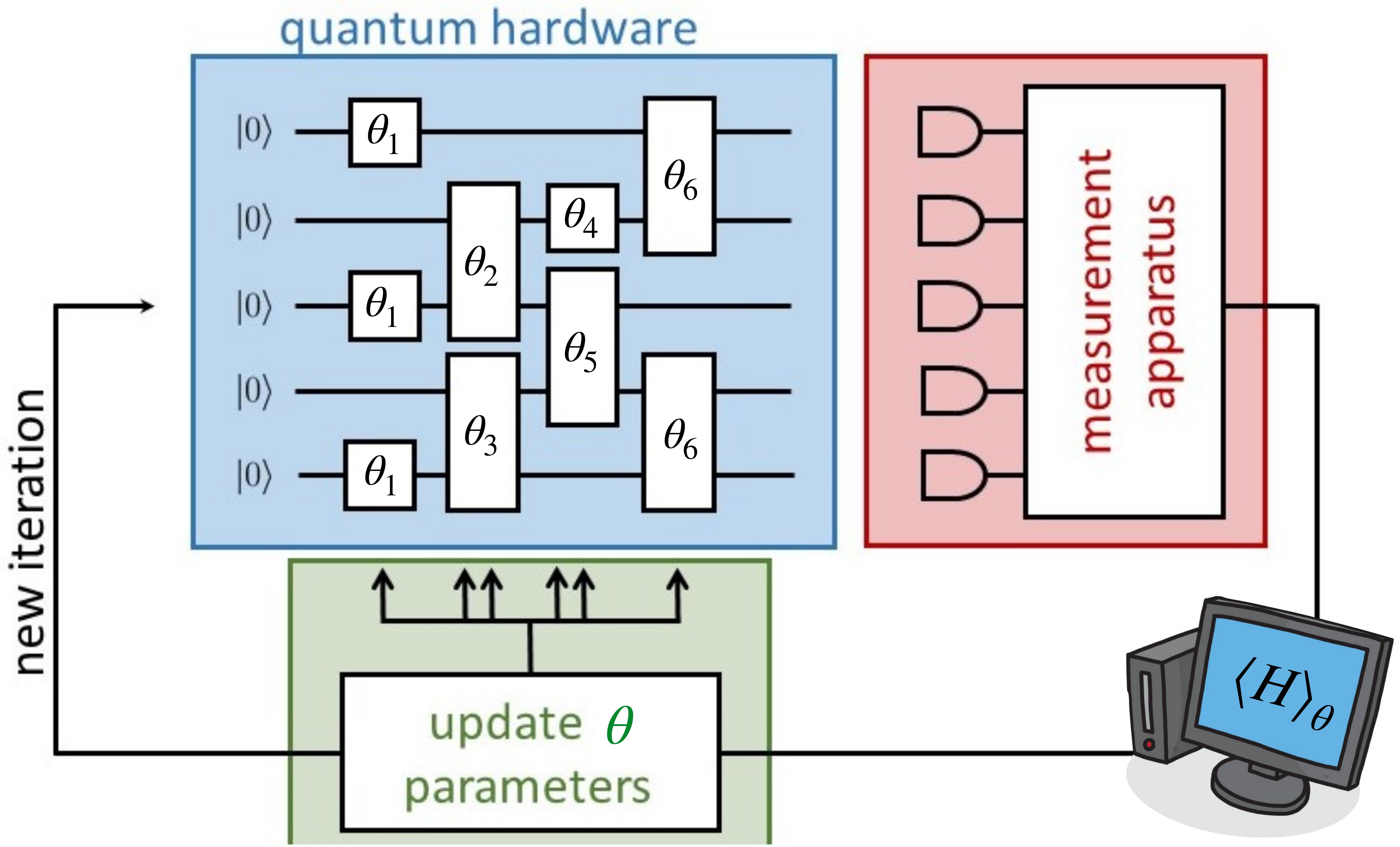
Please stay tuned !

J_2/J_1

Differentiable Programming Quantum Circuits

Neural Nets — Probabilistic Graphical Models — Tensor Nets — Quantum Circuits

Variational quantum eigensolver

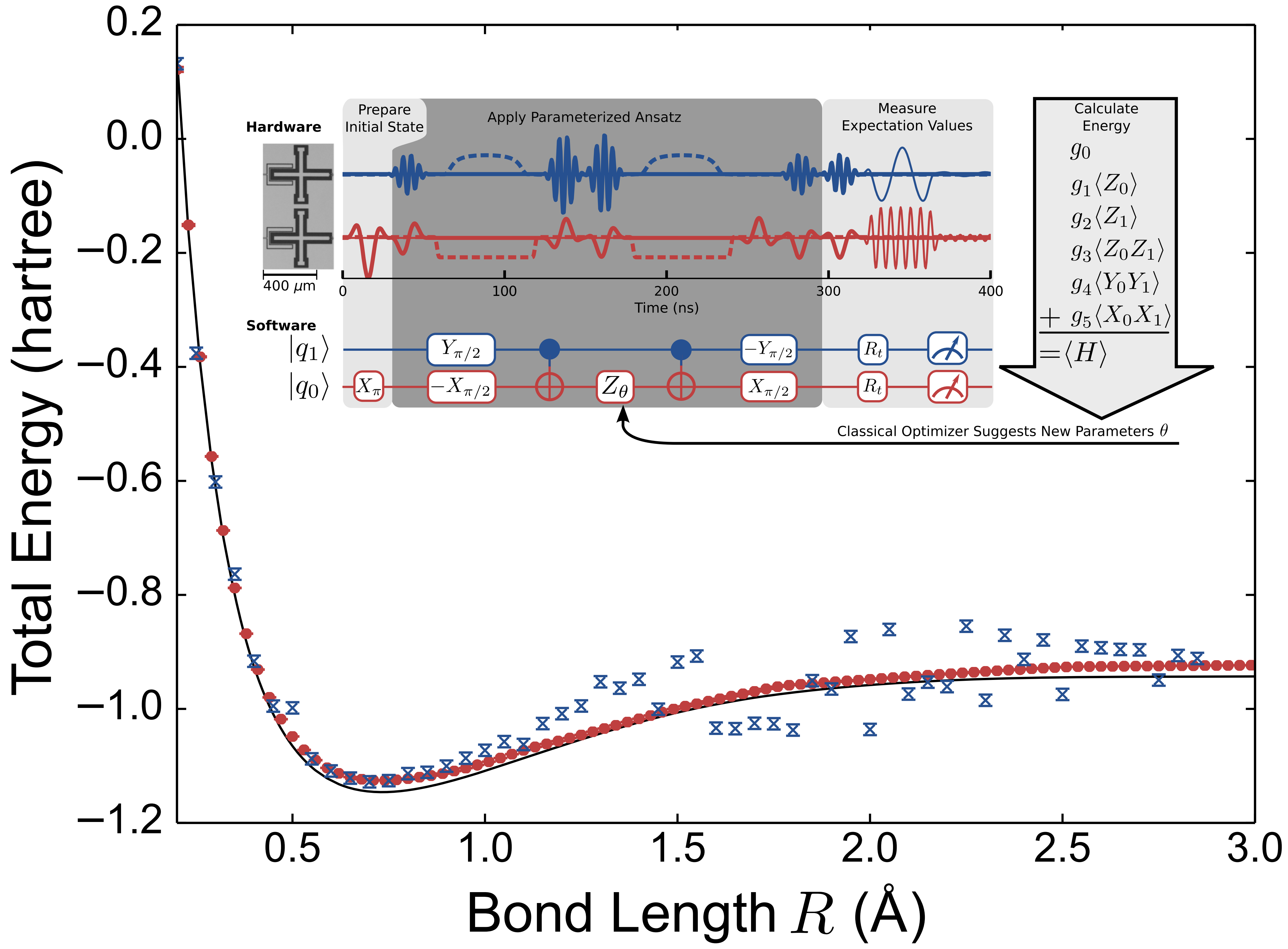


Quantum circuit as a variational ansatz

Peruzzo et al,
Nat. Comm. ’13

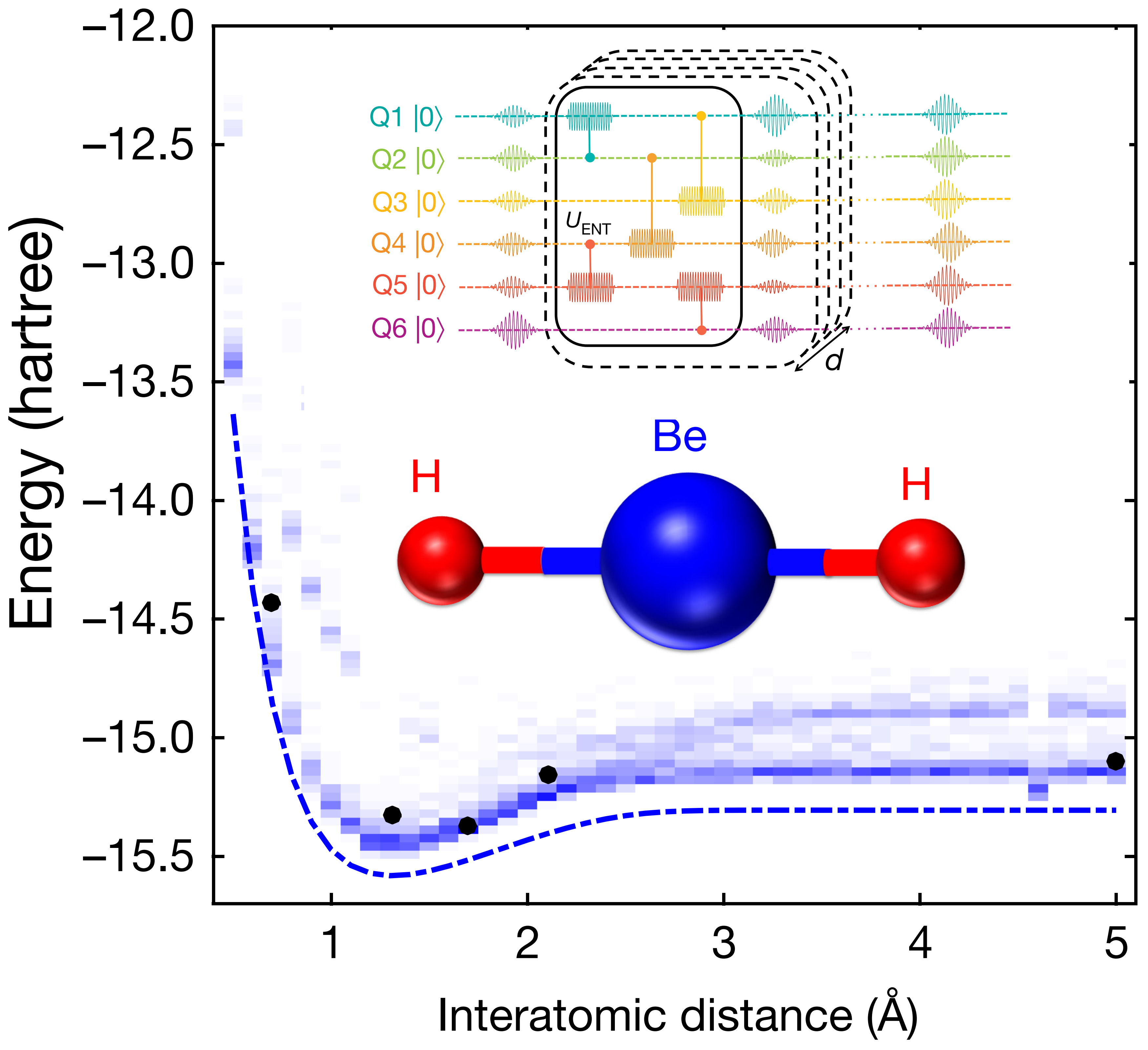
VQE on actual quantum devices

H_2 molecule with 2 qubits



Google PRX '16

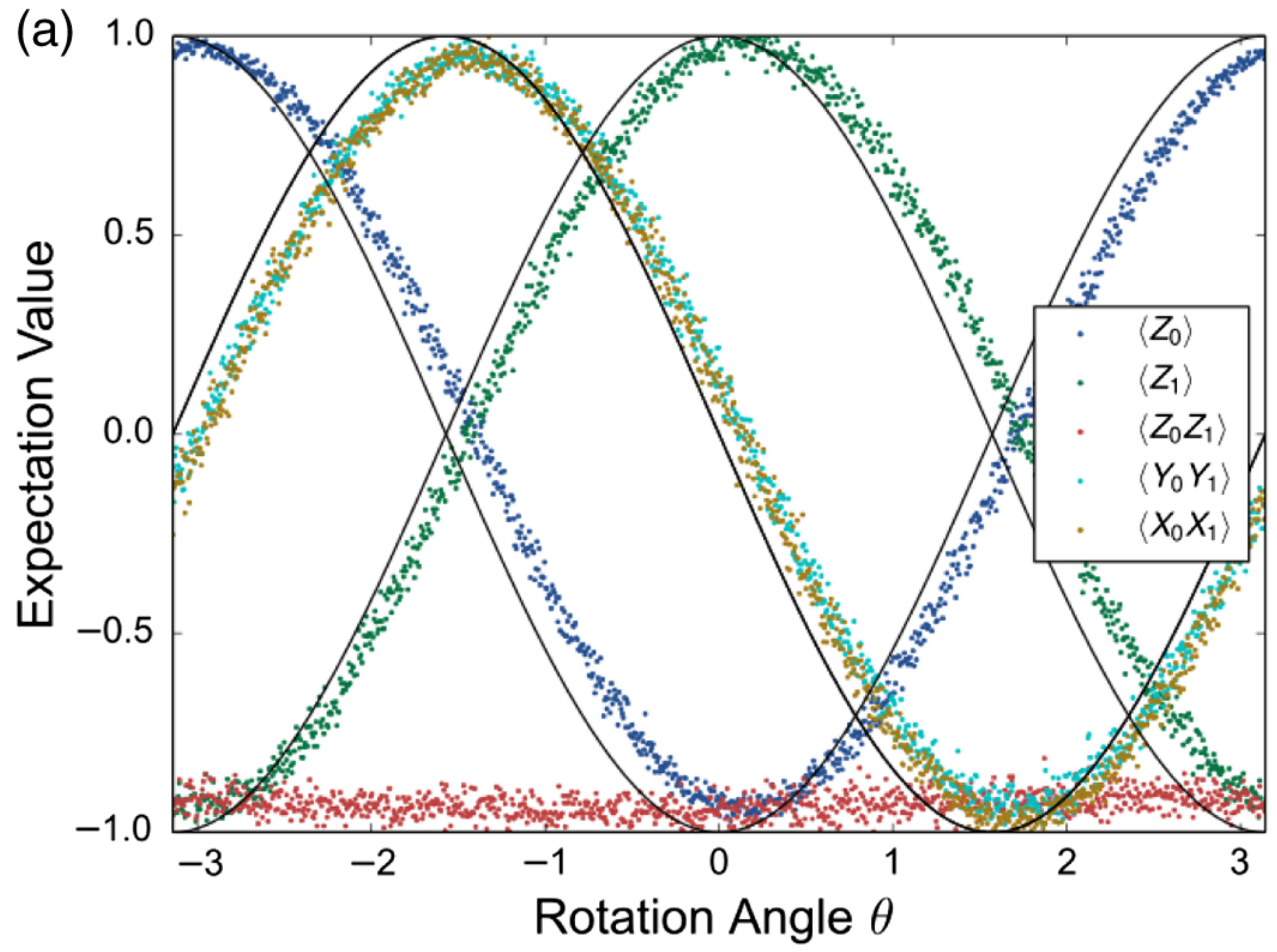
BeH_2 molecule with 6 qubits



IBM Nature '17

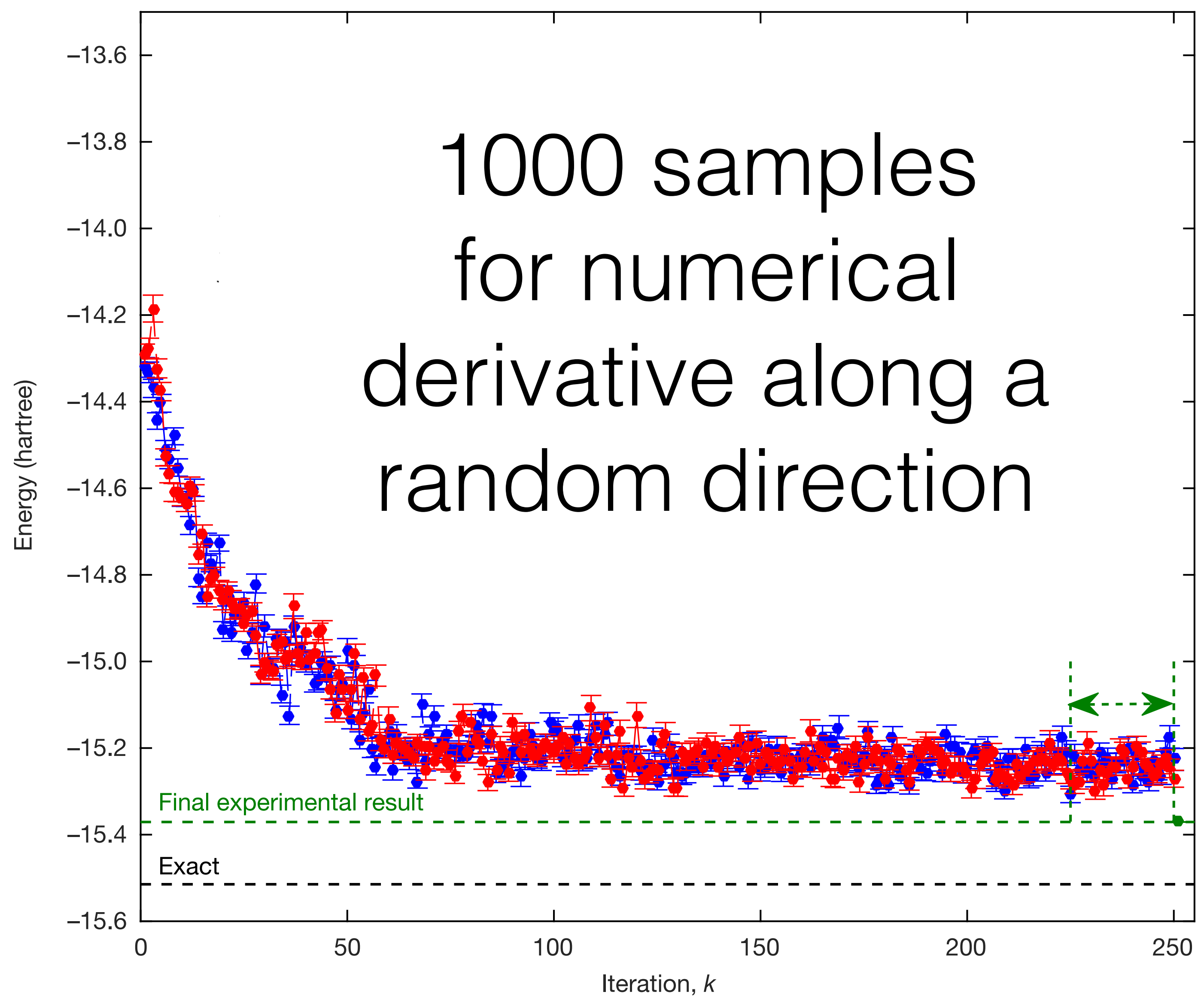
Optimize the quantum circuit

Scan 1000 values of the single variational parameter



Google PRX '16

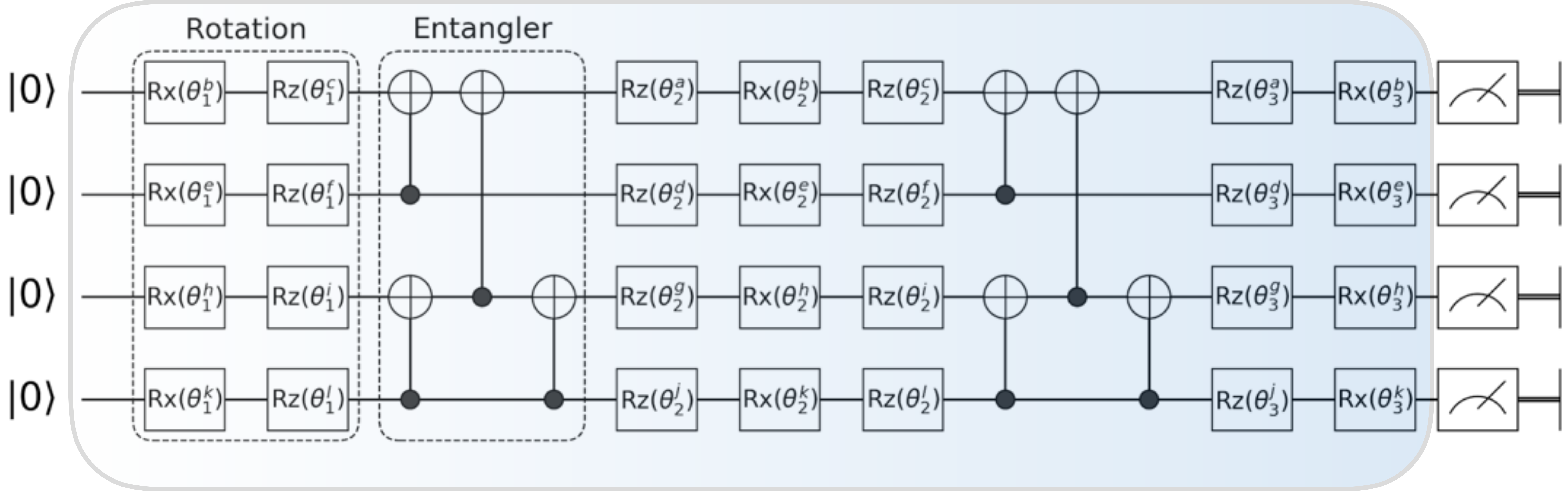
Stochastic gradient descend with numerical derivative



IBM Nature '17

These optimization schemes do not scale to higher dimensions

Differentiable quantum circuits



Parametrized gate of the form

$$e^{-\frac{i\theta}{2}\Sigma} \quad \text{with} \quad \Sigma^2 = 1$$

eg, X, Y, Z, CNOT, SWAP…

Li et al, PRL '17, Mitarai et al, PRA '18
Schuld et al, PRA '19, Nakanishi et al '19

$$\boxed{\nabla \langle H \rangle_\theta = \left(\langle H \rangle_{\theta+\pi/2} - \langle H \rangle_{\theta-\pi/2} \right) / 2}$$

Unbiased gradient estimator measured on quantum circuits

Monte Carlo Gradient Estimation in Machine Learning

Shakir Mohamed* SHAKIR@GOOGLE.COM
Mihaela Rosca* MIHAELACR@GOOGLE.COM
Michael Figurnov* MFIGURNOV@GOOGLE.COM
Andriy Mnih* AMNIH@GOOGLE.COM
**Equal contributions; DeepMind, London*

59 pages survey, three types of gradient estimators

1906.10652

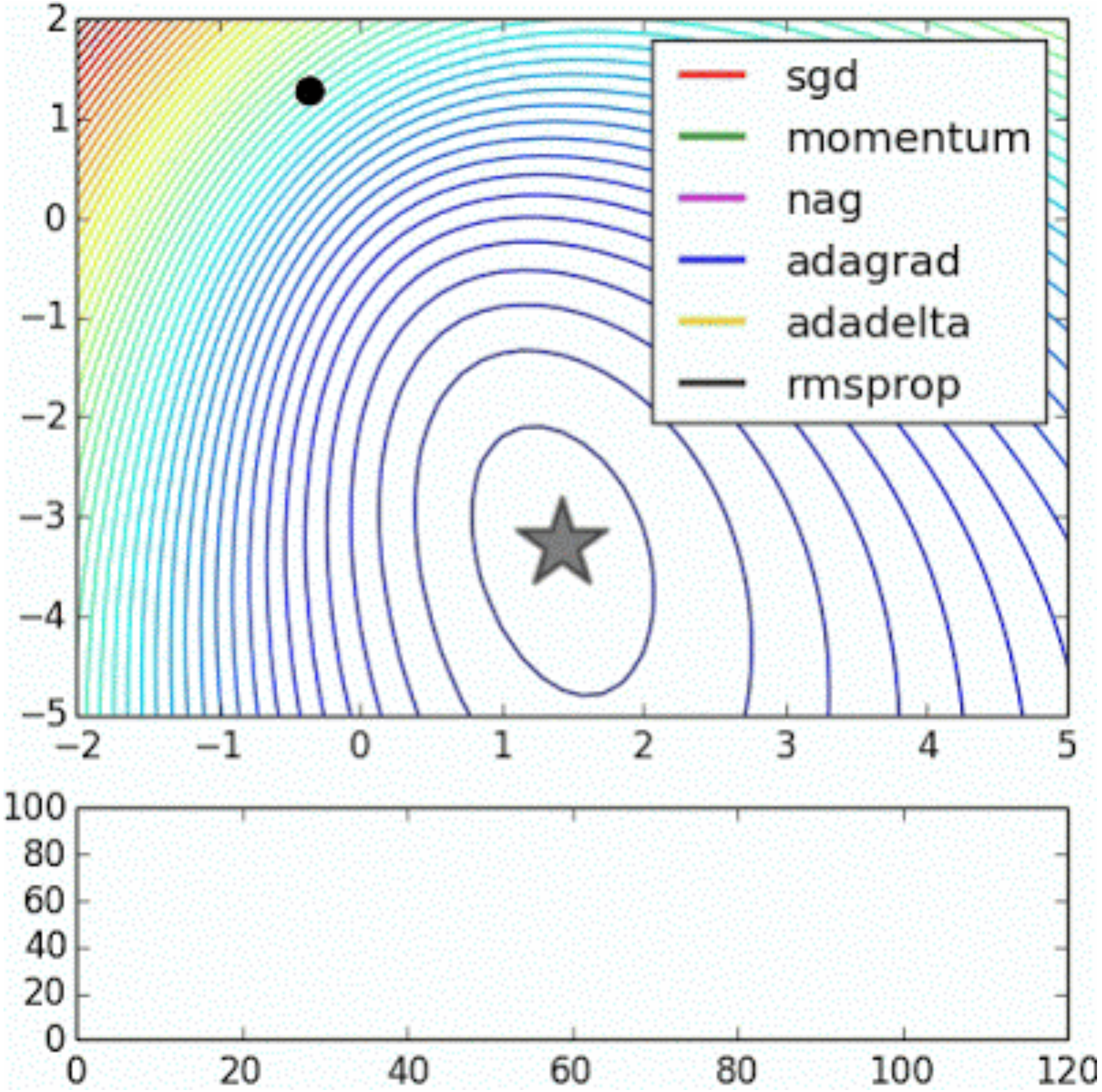
10.1 Guidance in Choosing Gradient Estimators

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

With so many competing approaches, we offer our rules of thumb in choosing an estimator, which follow the intuition we developed throughout the paper:

- If our estimation problem involves continuous functions and measures that are continuous in the domain, then using the pathwise estimator is a good default. It is relatively easy to implement and a default implementation, one without other variance reduction, will typically have variance that is low enough so as not to interfere with the optimisation.
- If the cost function is not differentiable or a black-box function then the score-function or the measure-valued gradients are available. If the number of parameters is low, then the measure-valued gradient will typically have lower variance and would be preferred. But if we have a high-dimensional parameter set, then the score function estimator should be used.
- If we have no control over the number of times we can evaluate a black-box cost function, effectively only allowing a single evaluation of it, then the score function is the only estimator

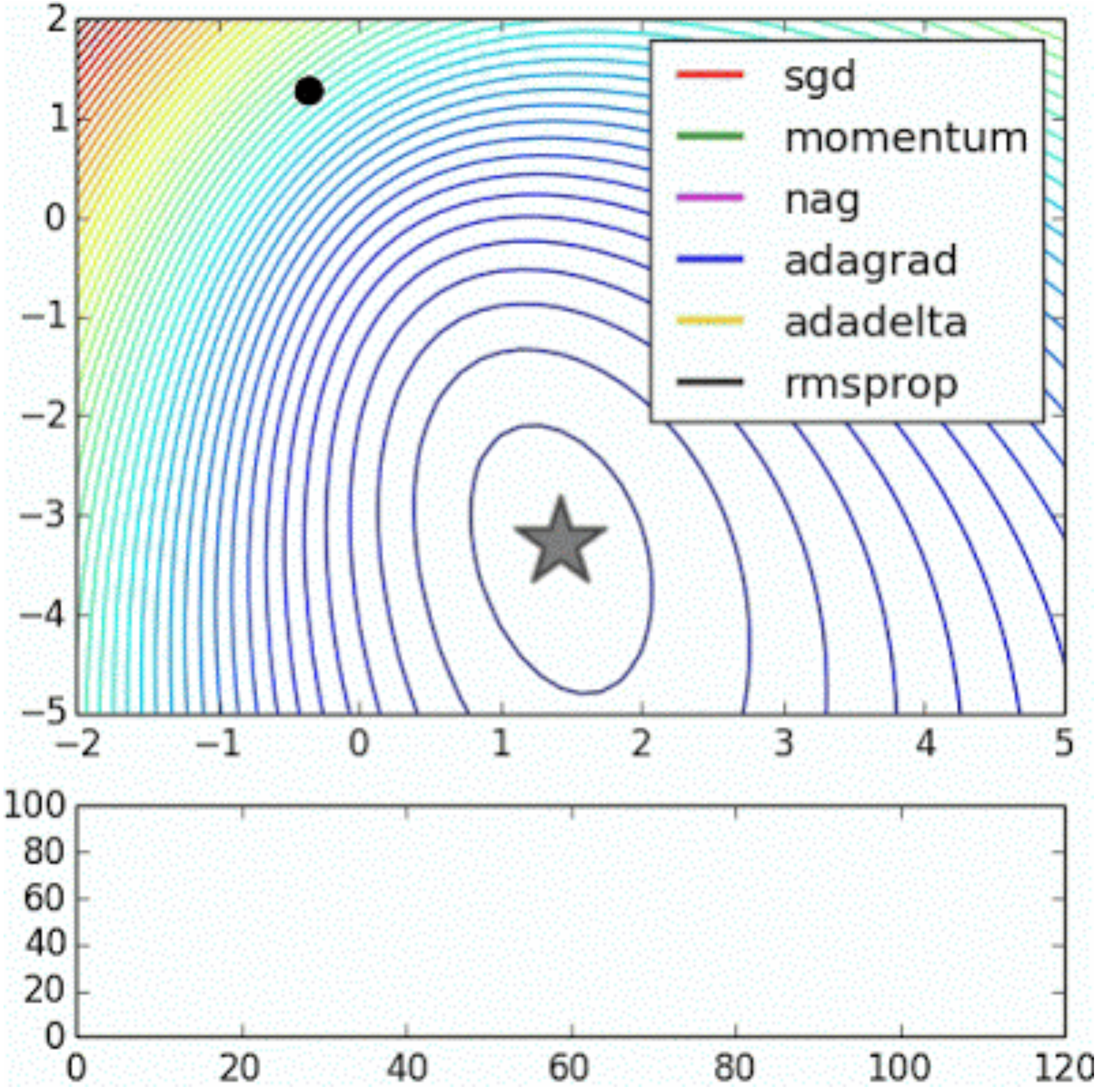
Optimization with noisy gradient



Ruder, 1609.04747

VQE encounters the "same type" of stochastic optimization in deep learning

Optimization with noisy gradient

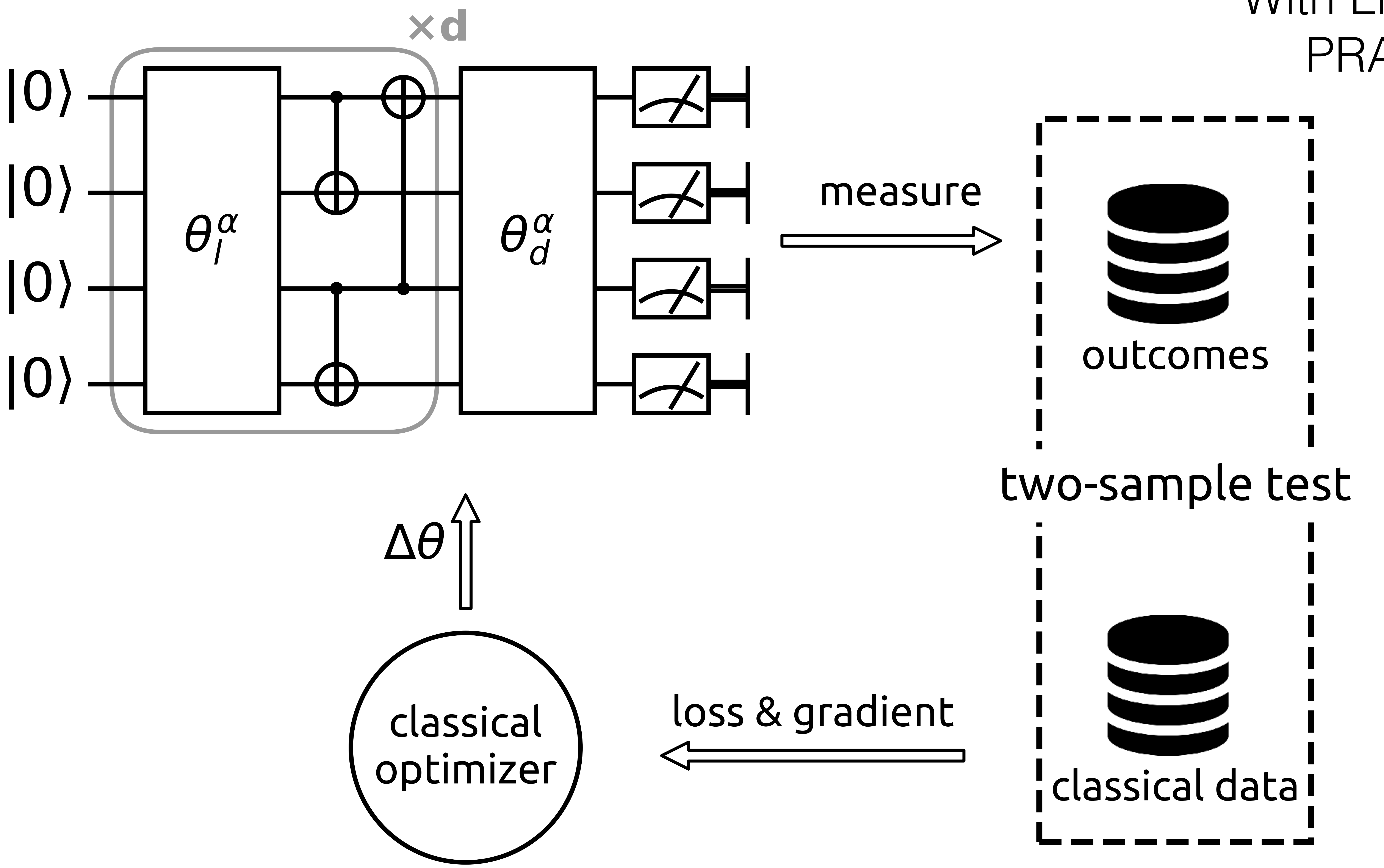


Ruder, 1609.04747

VQE encounters the “same type” of stochastic optimization in deep learning

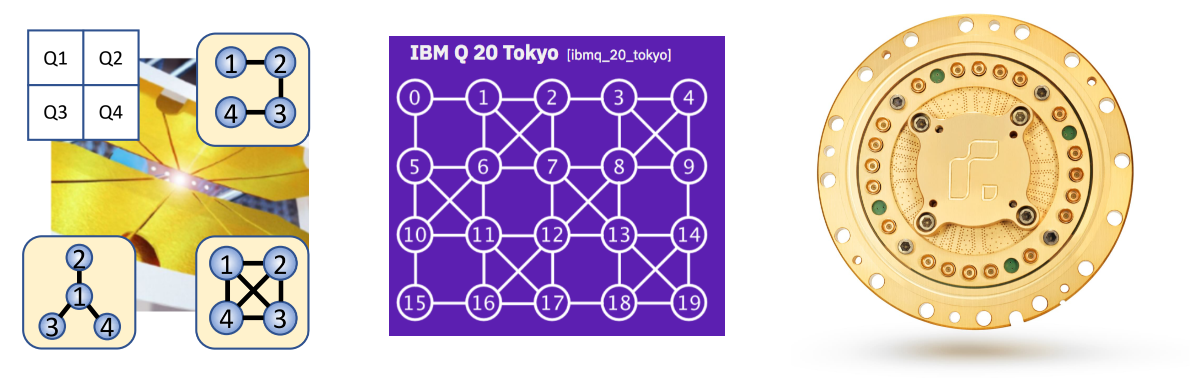
Quantum Circuit Born Machine

With Liu, Zeng, Wu, Hu
PRA '18, PRA '19



Experiments:

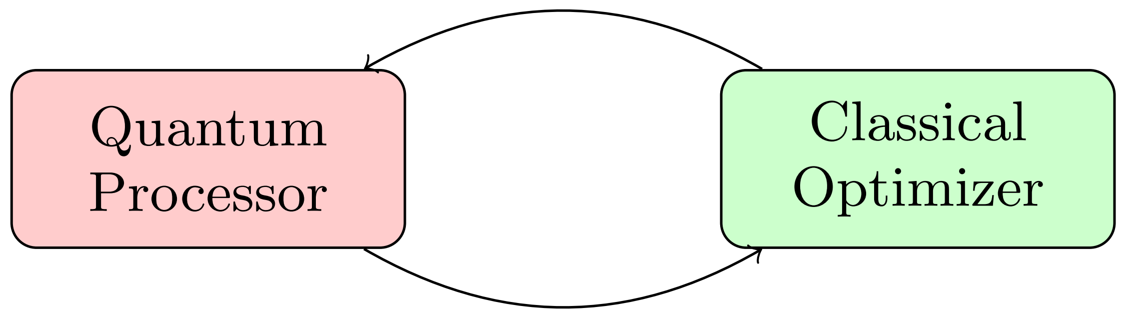
1801.07686
1812.08862
1811.09905
1901.08047
1904.02214



Train quantum circuits as probabilistic generative models with implicit density

Strong expressibility due to quantum sampling complexity

Differentiable Quantum Programming



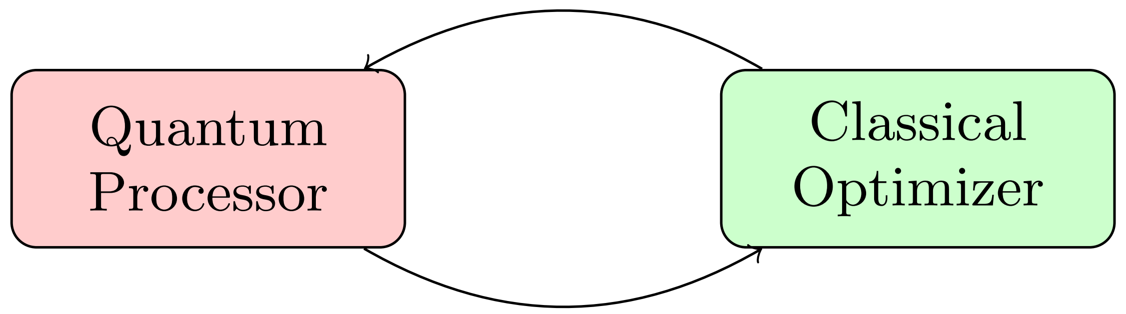
It is a paradigm beyond quantum-classical hybrid

- Variational quantum eigensovler (VQE)
- Quantum circuit Born machine (QCBM)
- Quantum approximate optimization algorithm (QAOA)
- Quantum pattern recognition

…

Quantum circuit classifier	Farhi, Neven, 1802.06002 Havlicek et al, 1804.11326
TNS inspired circuit architecture	Huggins, Patel, Whaley, Stoudenmire, 1803.11537
VQE with fewer qubits	Liu, Zhang, Wan, LW, 1902.02663
Quantum generative model	Gao, Zhang, Duan, 1711.02038
Quantum adversarial training	Dallaire-Demers, Lloyd, Benedetti 1804.08641,1804.09139, 1806.00463

Differentiable Quantum Programming



It is a paradigm beyond quantum-classical hybrid

- Variational quantum eigensovler (VQE)
- Quantum circuit Born machine (QCBM)
- Quantum approximate optimization algorithm (QAOA)
- Quantum pattern recognition

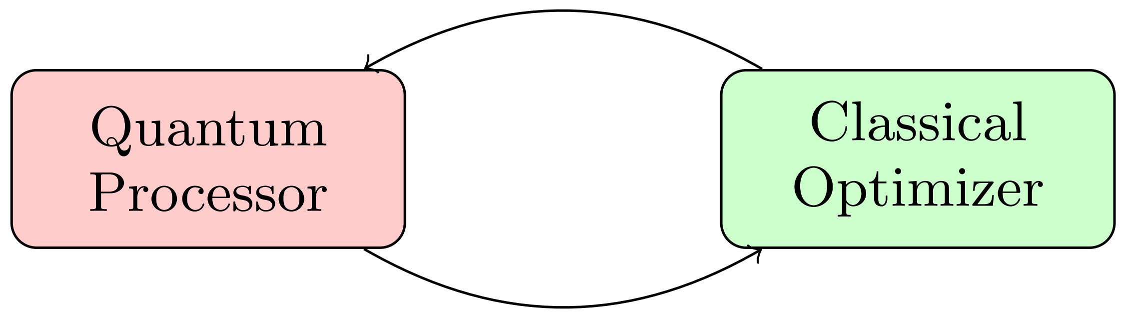
Near term：
What can we do with noisy circuits of limited depth ?

Long term:
Are we really good at programing quantum computers ?

…

Quantum circuit classifier	Farhi, Neven, 1802.06002 Havlicek et al, 1804.11326
TNS inspired circuit architecture	Huggins, Patel, Whaley, Stoudenmire, 1803.11537
VQE with fewer qubits	Liu, Zhang, Wan, LW, 1902.02663
Quantum generative model	Gao, Zhang, Duan, 1711.02038
Quantum adversarial training	Dallaire-Demers, Lloyd, Benedetti 1804.08641,1804.09139, 1806.00463

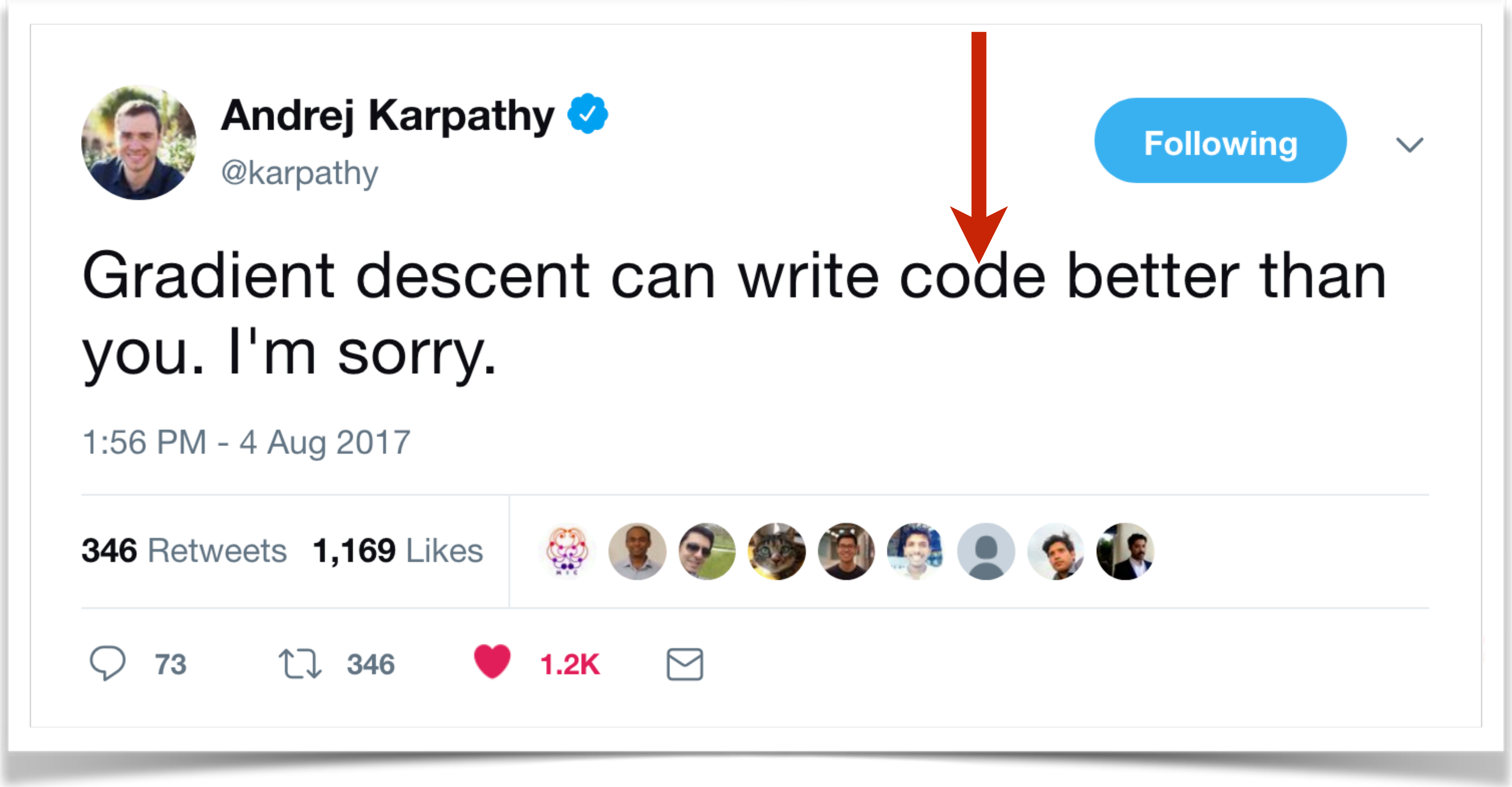
Differentiable Quantum Programming



It is a paradigm beyond quantum-classical hybrid

- Variational quantum eigensovler (VQE)
- Quantum circuit Born machine (QCBM)
- Quantum approximate optimization algorithm (QAOA)
- Quantum pattern recognition

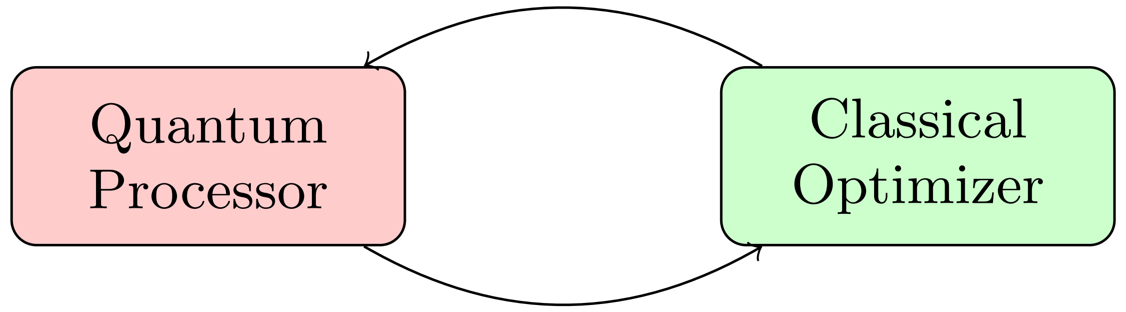
Quantum code



…

Quantum circuit classifier	Farhi, Neven, 1802.06002 Havlicek et al, 1804.11326
TNS inspired circuit architecture	Huggins, Patel, Whaley, Stoudenmire, 1803.11537
VQE with fewer qubits	Liu, Zhang, Wan, LW, 1902.02663
Quantum generative model	Gao, Zhang, Duan, 1711.02038
Quantum adversarial training	Dallaire-Demers, Lloyd, Benedetti 1804.08641,1804.09139, 1806.00463

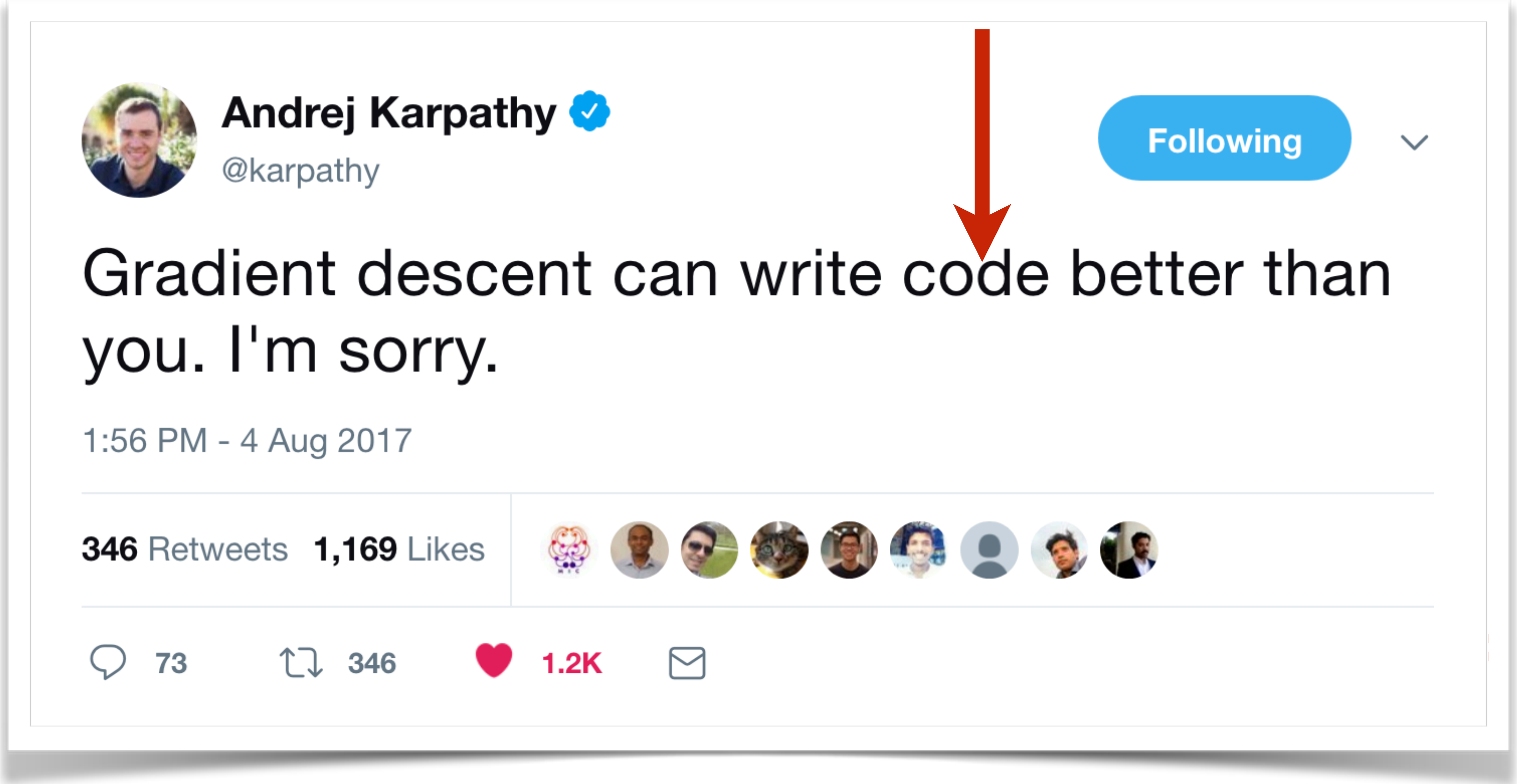
Differentiable Quantum Programming



It is a paradigm beyond quantum-classical hybrid

- Variational quantum eigensovler (VQE)
- Quantum circuit Born machine (QCBM)
- Quantum approximate optimization algorithm (QAOA)
- Quantum pattern recognition

Quantum code



…

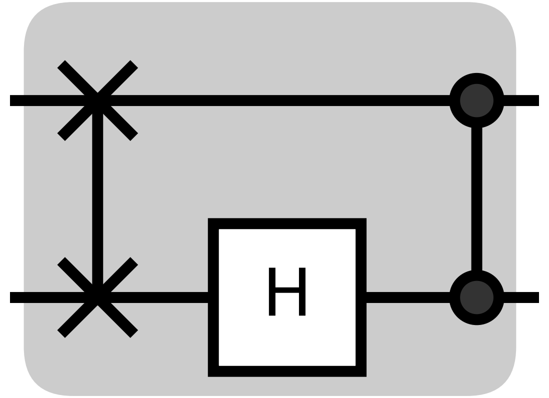
Quantum circuit
TNS inspired circ
VQE with fewer q
Quantum genera
Quantum advers

Quantum Software 2.0

1806.00463

Be prepared for Quantum Software 2.0

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

julia + 

+ 
= 幺

Xiu-Zhe Luo (Waterloo & PI) **Jin-Guo Liu (IOP, CAS)**

Features:

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

Thank You!

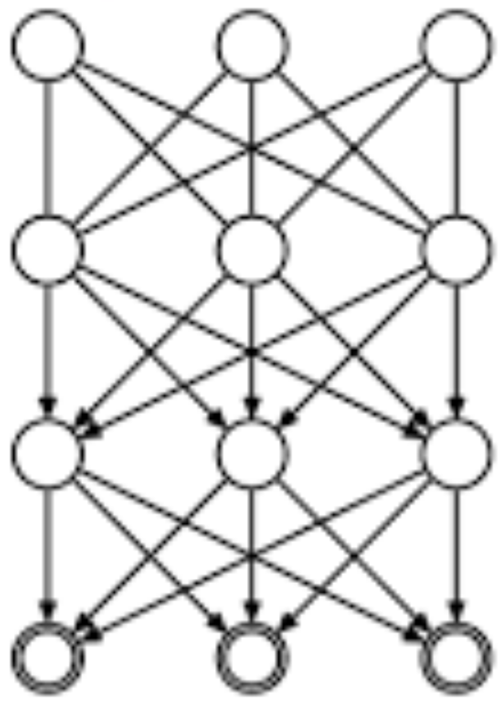
Differentiable Programming Tensor Networks

Hai-Jun Liao, Jin-Guo Liu, LW, Tao Xiang, 1903.09650, PRX in press

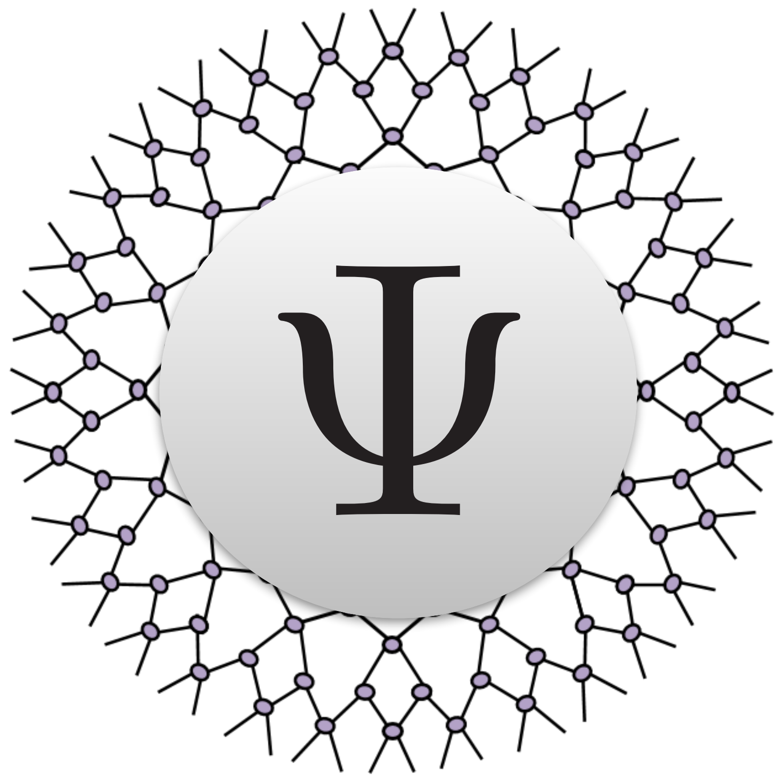
Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design

Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, LW, up coming

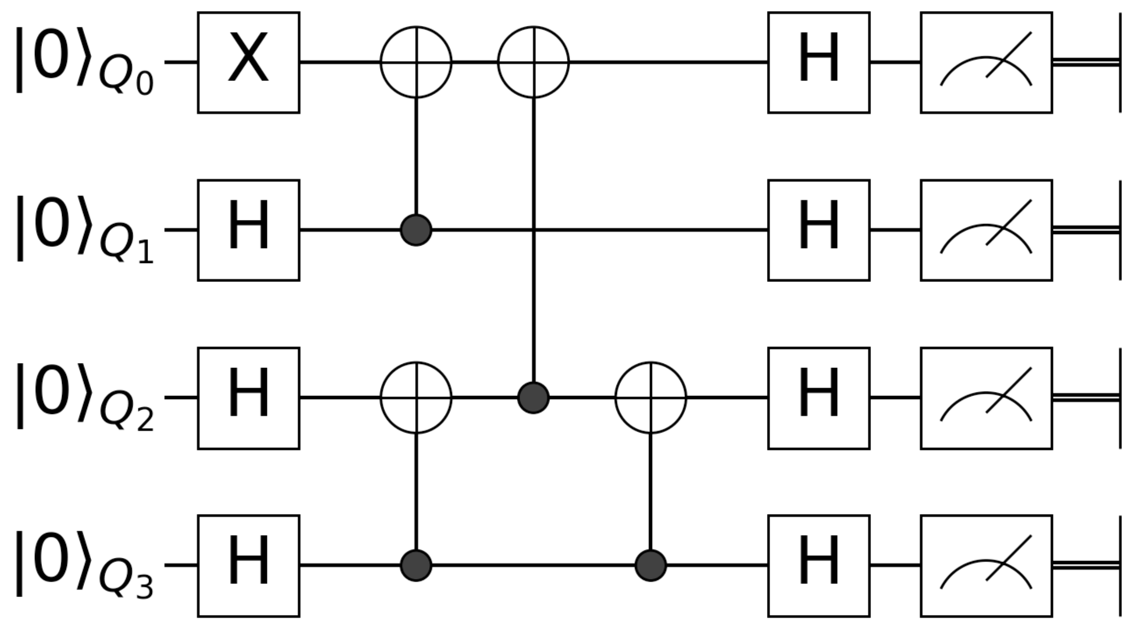
$\partial\Big[$



Neural Networks



Tensor Networks



Quantum Circuits

$\Big]$