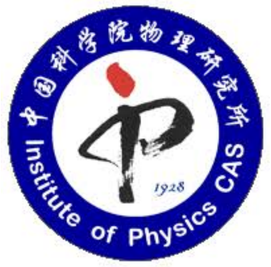
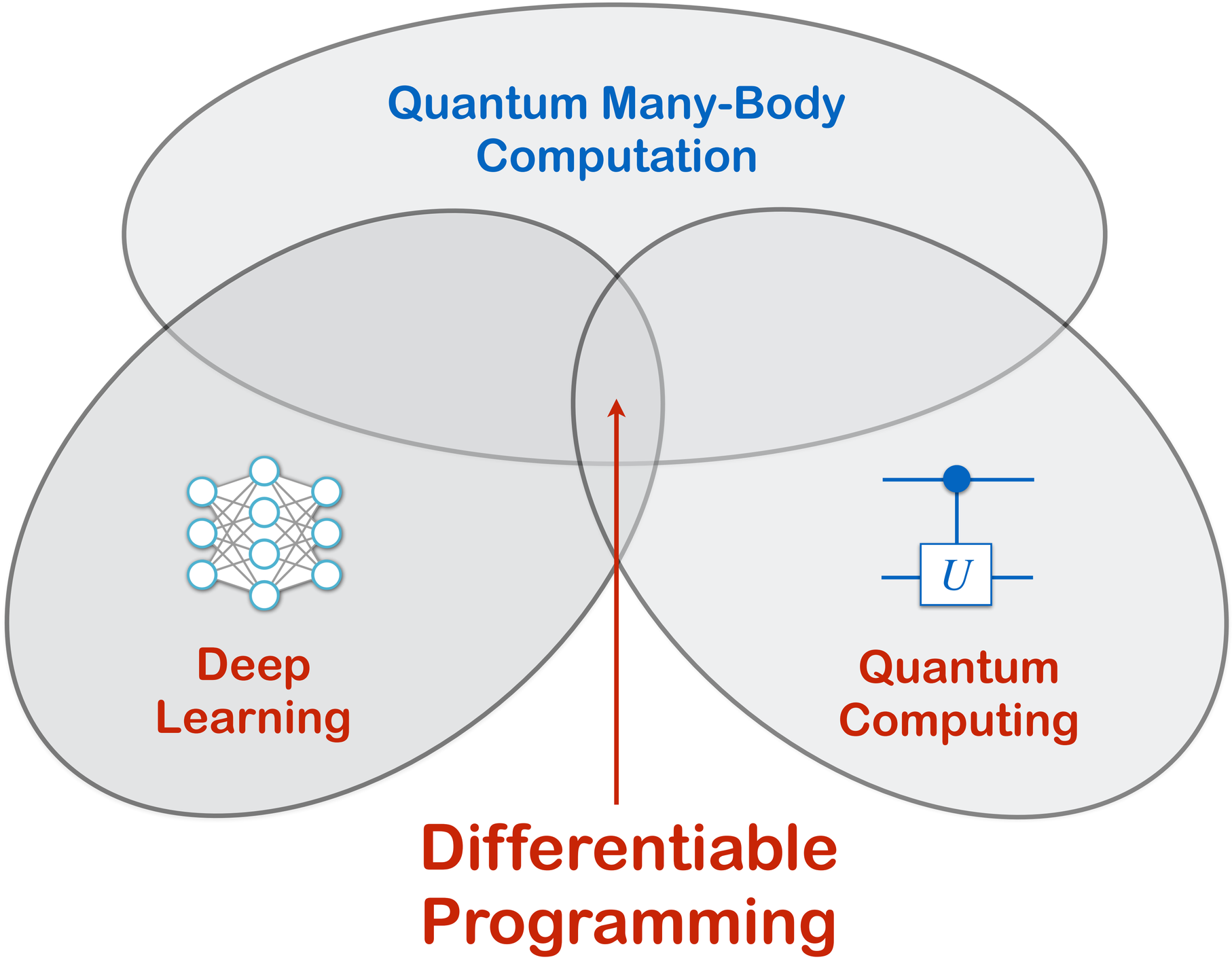
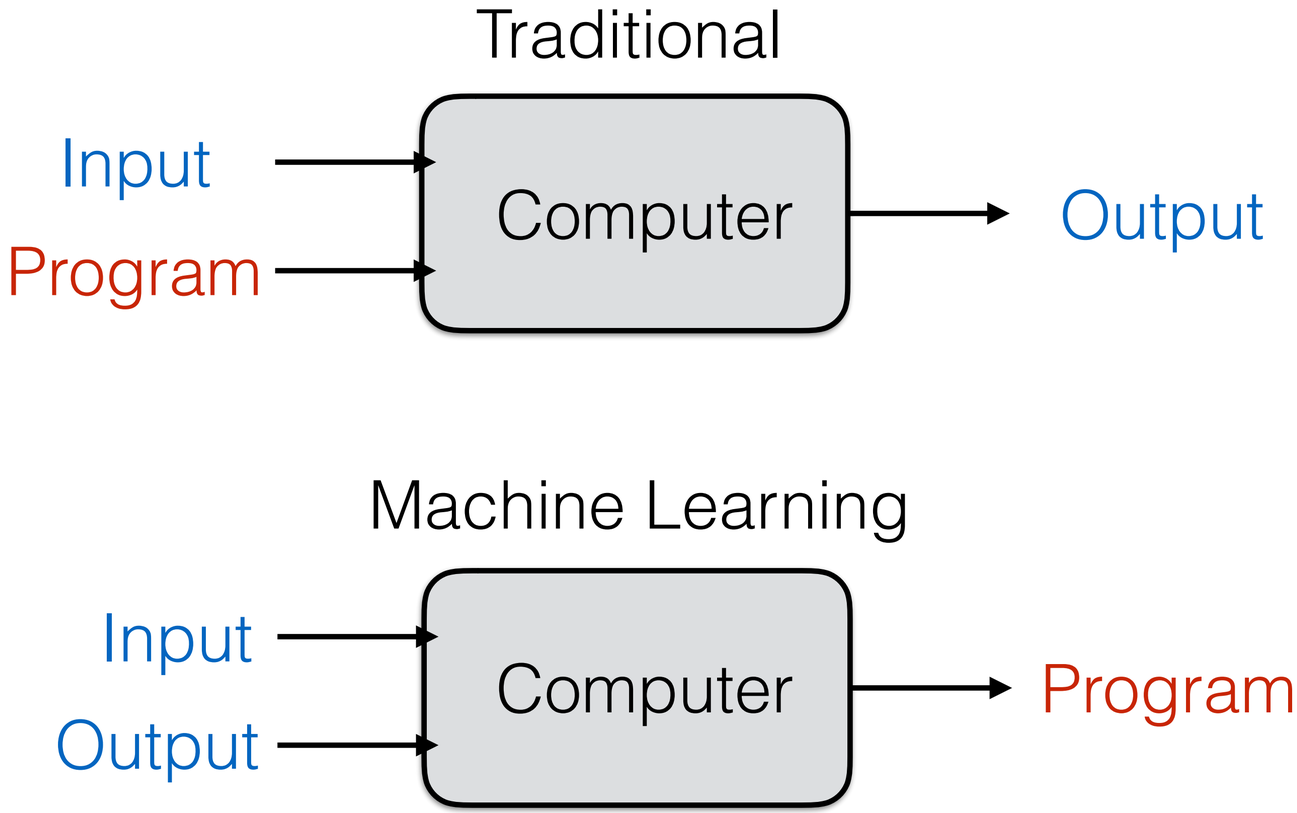
∂ifferentiate everything: A lesson from deep learning

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




Quantum Many-Body
Computation
Deep
Learning
U
Quantum
Computing
Differentiable
Programming

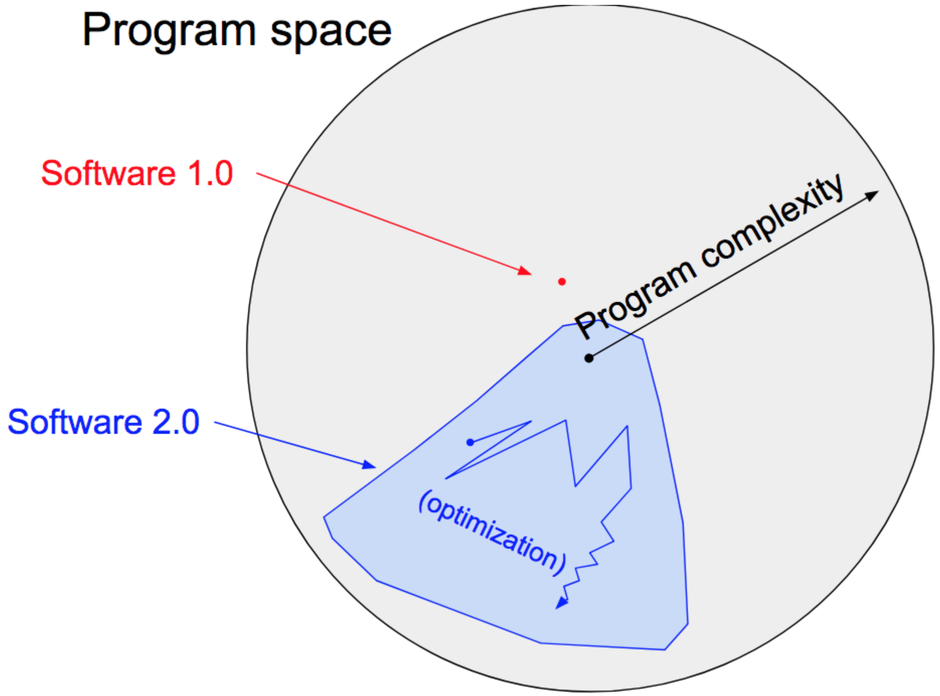
Differentiable Programming



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

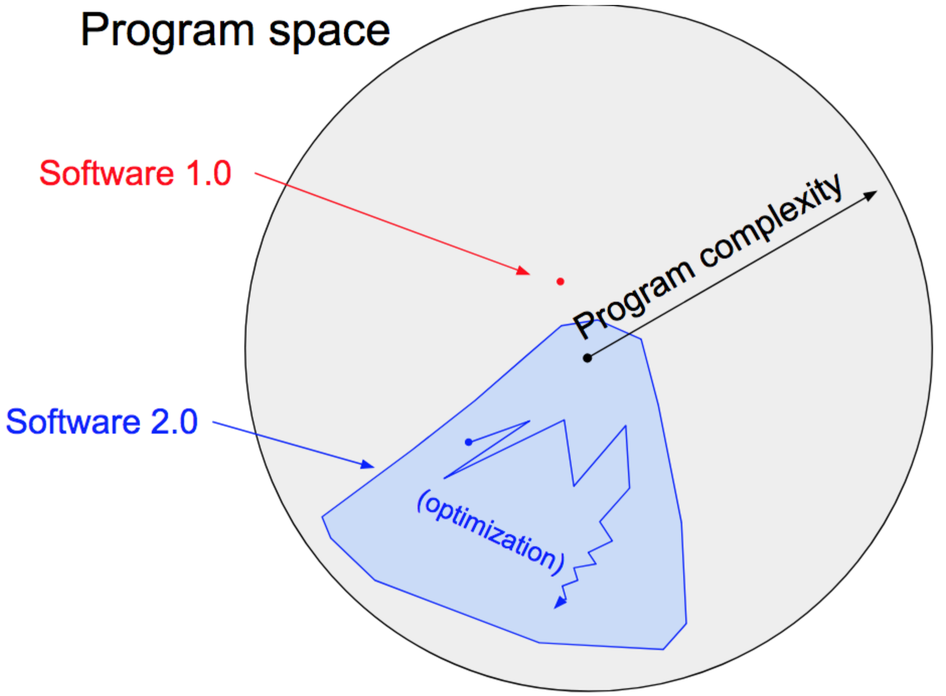
Benefits of Software 2.0

- 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

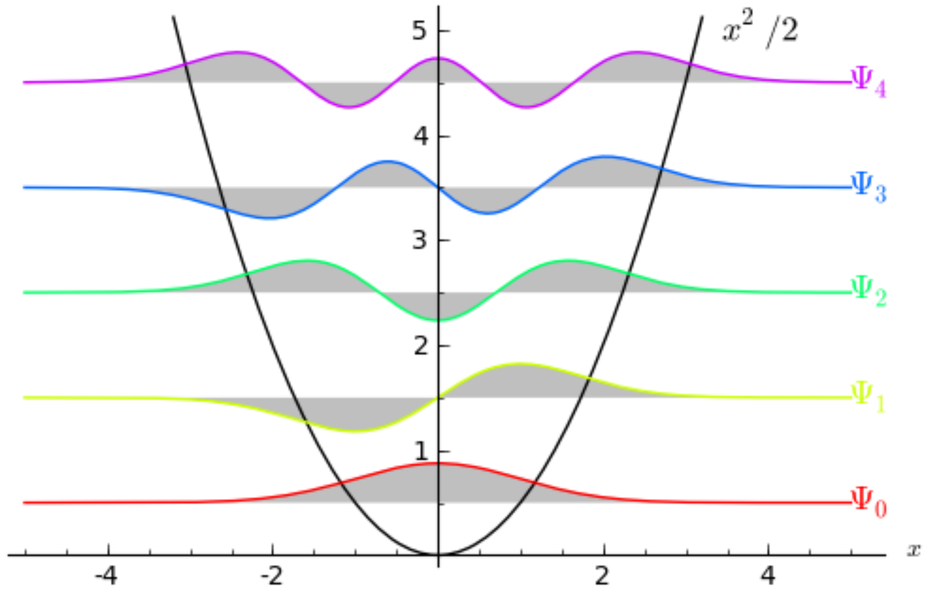


Writing software 2.0 by gradient search in the program space

Demo: Inverse Schrodinger Problem

Given ground state density, how to design the potential ?

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

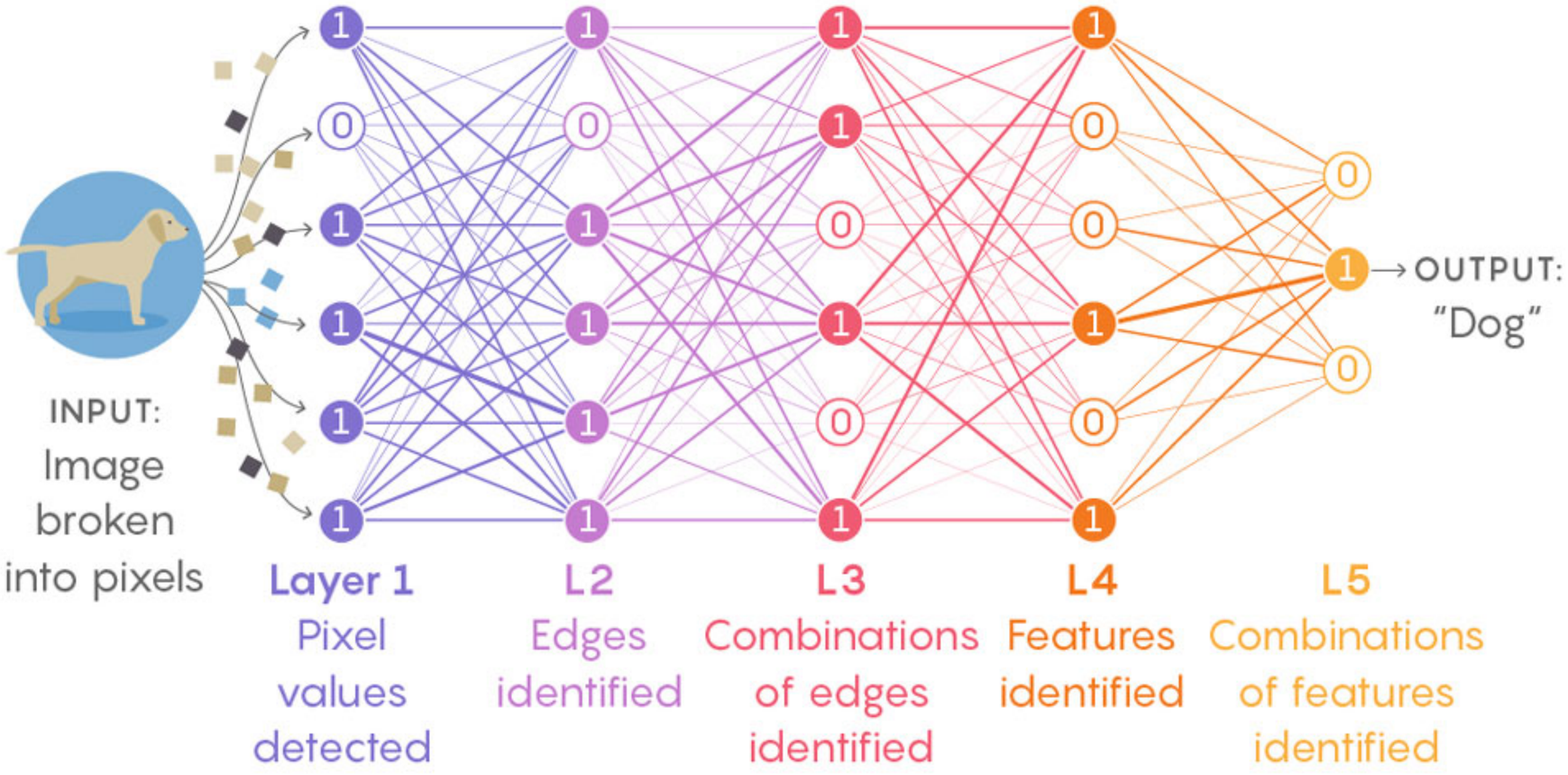


https://math.mit.edu/~stevenj/18.336/adjoint.pdf

https://github.com/QuantumBFS/SSSS/blob/master/1_deep_learning/schrodinger.py

What is under the hood ?

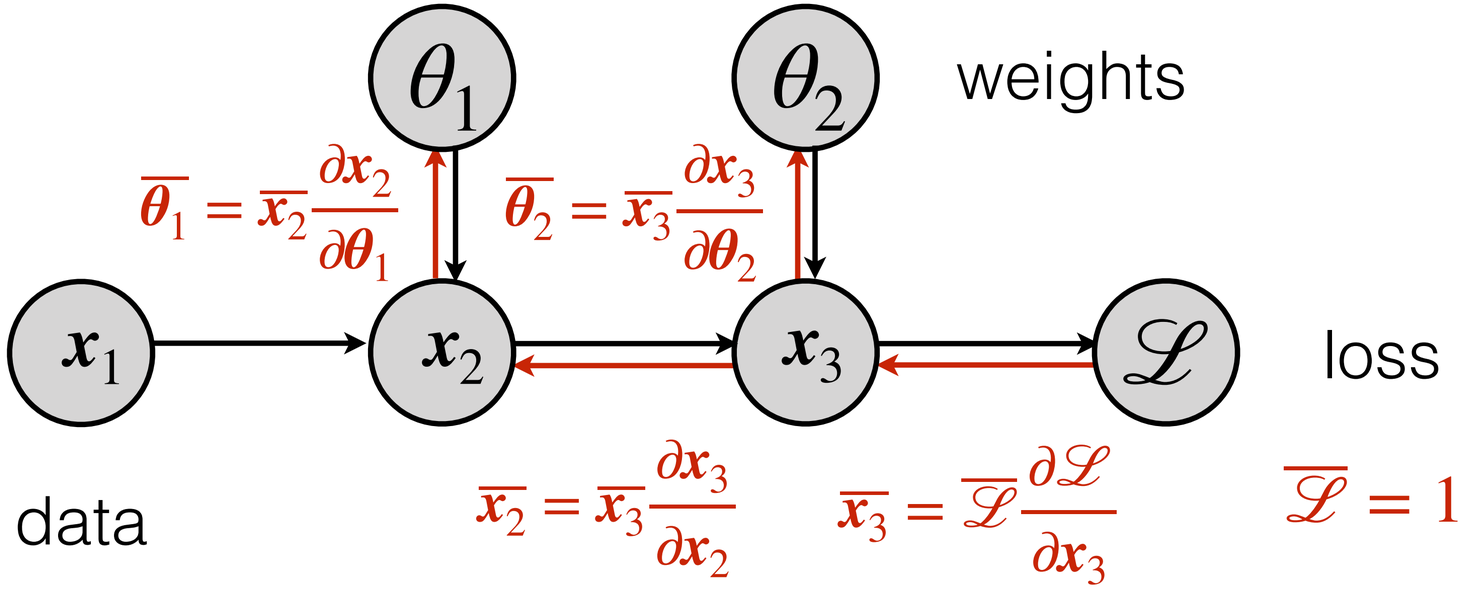
What is deep learning ?



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

Automatic differentiation on computation graph

"comb graph"

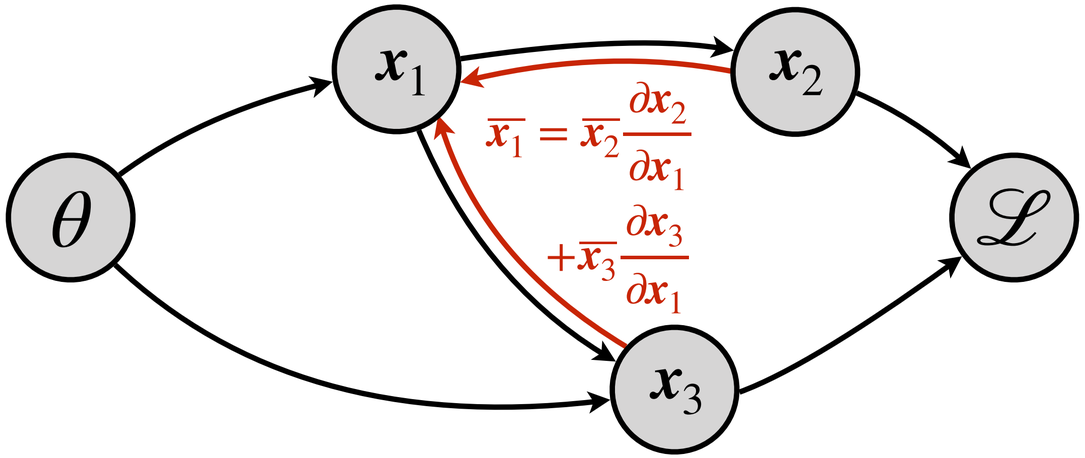


"adjoint variable" $\overline{x} = \dfrac{\partial \mathscr{L}}{\partial x}$

Pullback the adjoint through the graph

Automatic differentiation on computation graph

directed
acyclic graph

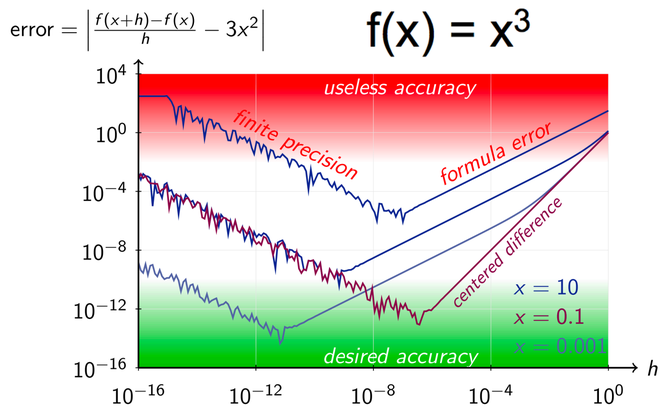


$$\overline{x_i} = \sum_{j:\,\text{child of } i} \overline{x_j}\,\frac{\partial x_j}{\partial x_i} \qquad \text{with} \quad \overline{\mathscr{L}} = 1$$

Message passing for the adjoint at each node

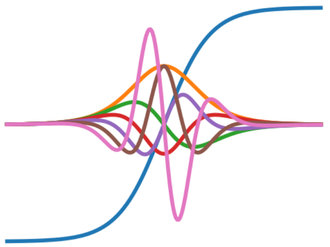
Advantages of automatic differentiation

- Accurate to the machine precision



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

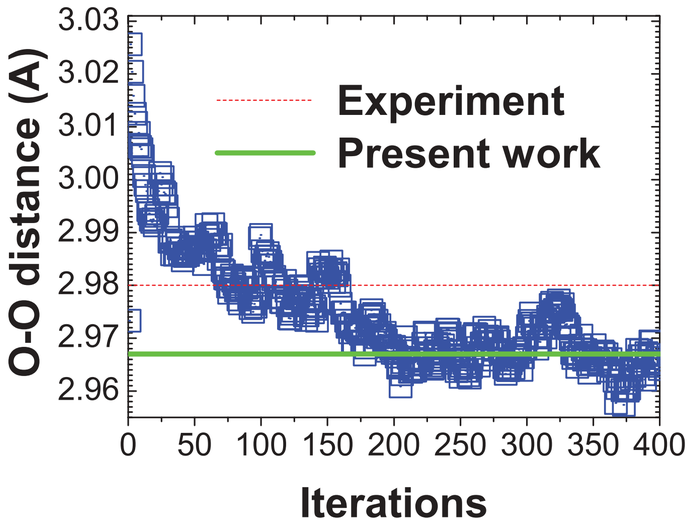
- Supports higher order gradients



```
>>> 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()
```

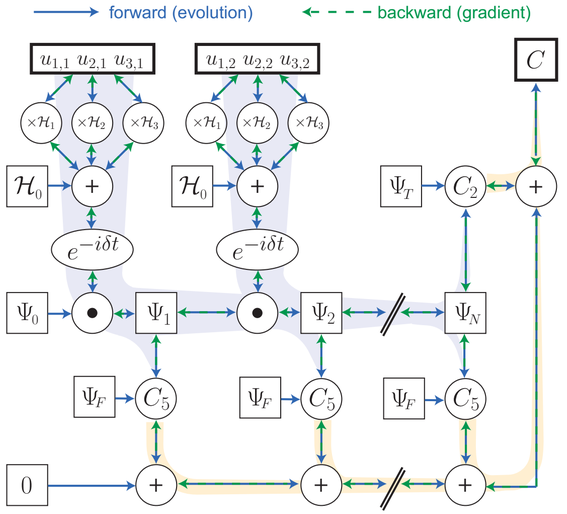
Applications of AD

Computing force



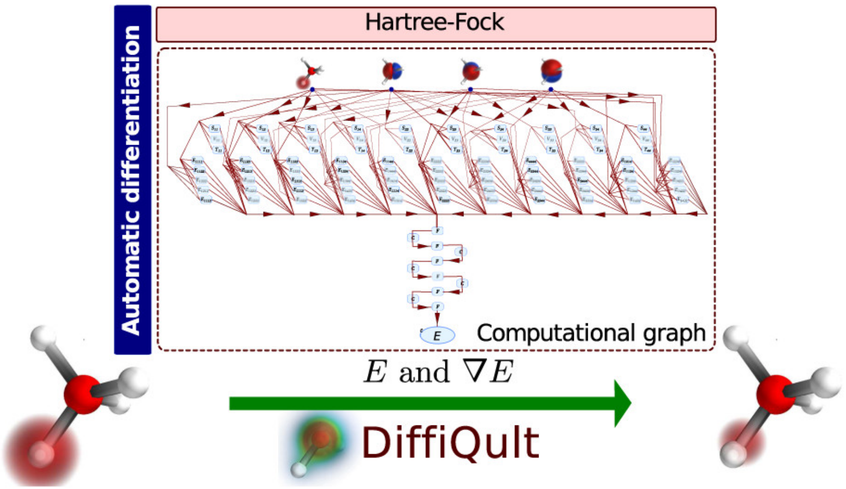
Sorella and Capriotti
J. Chem. Phys. ’10

Quantum optimal control



Leung et al
PRA ’17

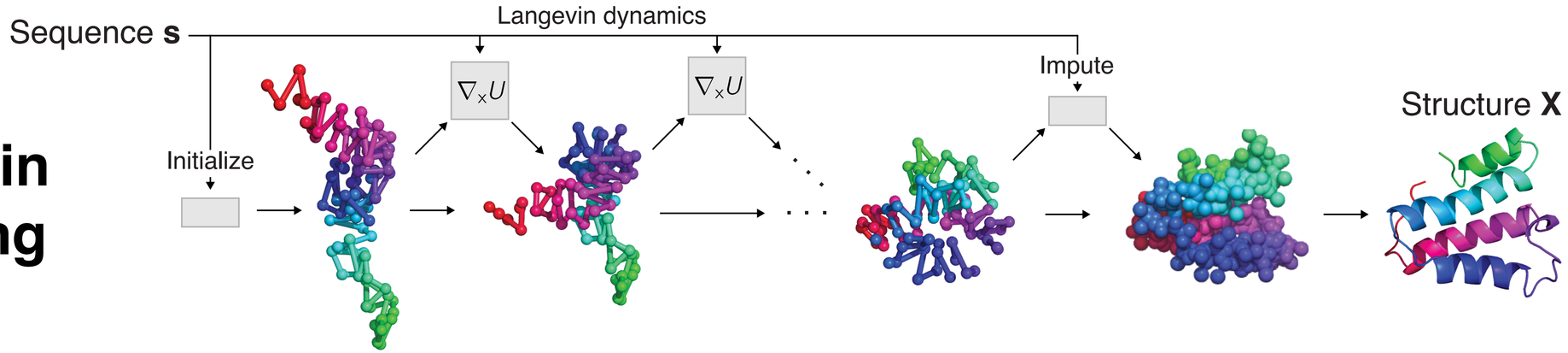
Variational Hartree-Fock



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

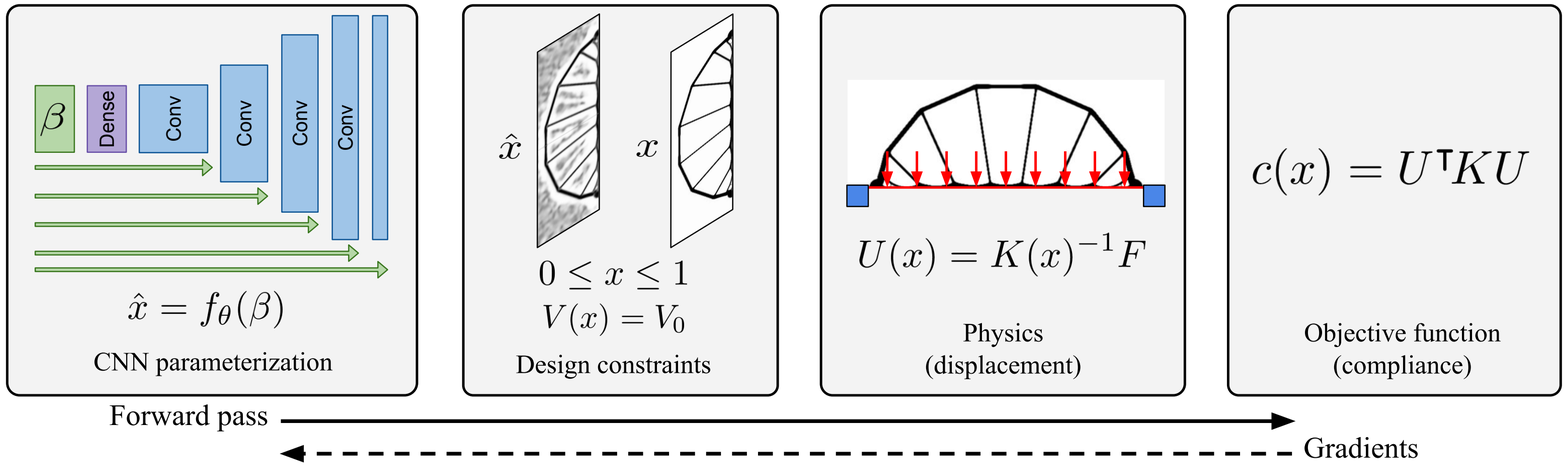
More Applications…

Protein folding

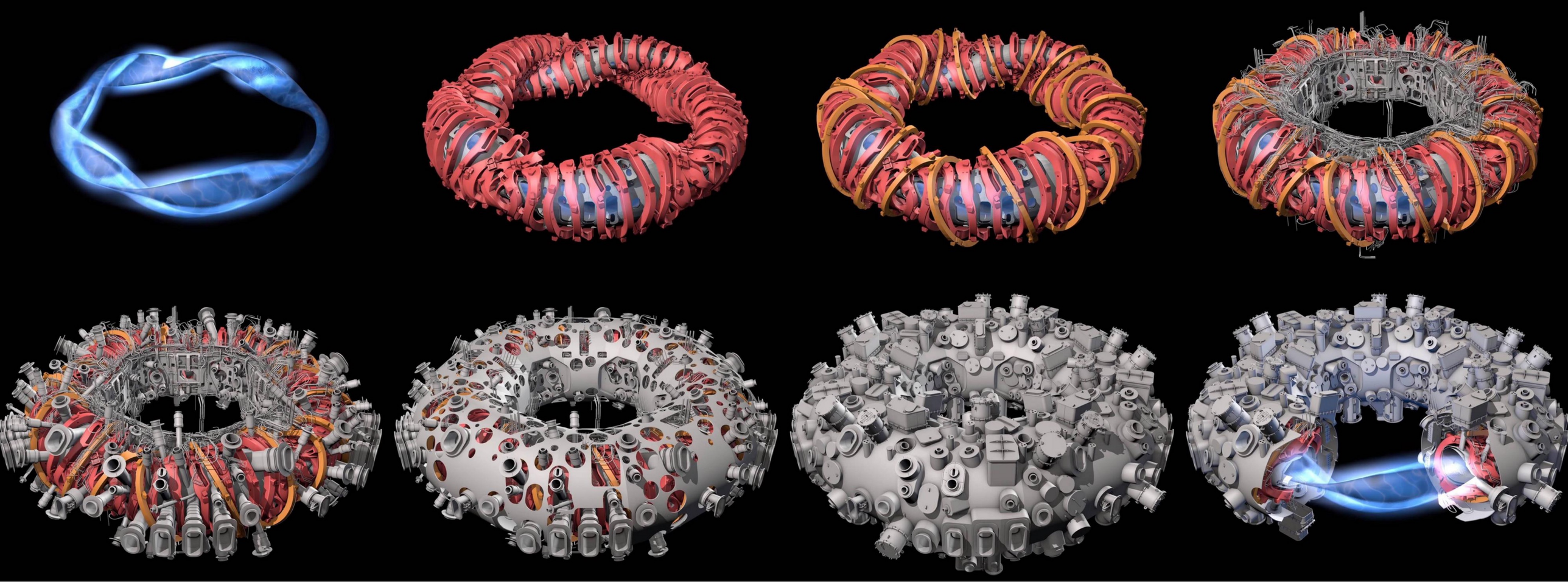


Ingraham et al ICLR ‘19

Structural optimization



Hoyer et al 1909.04240



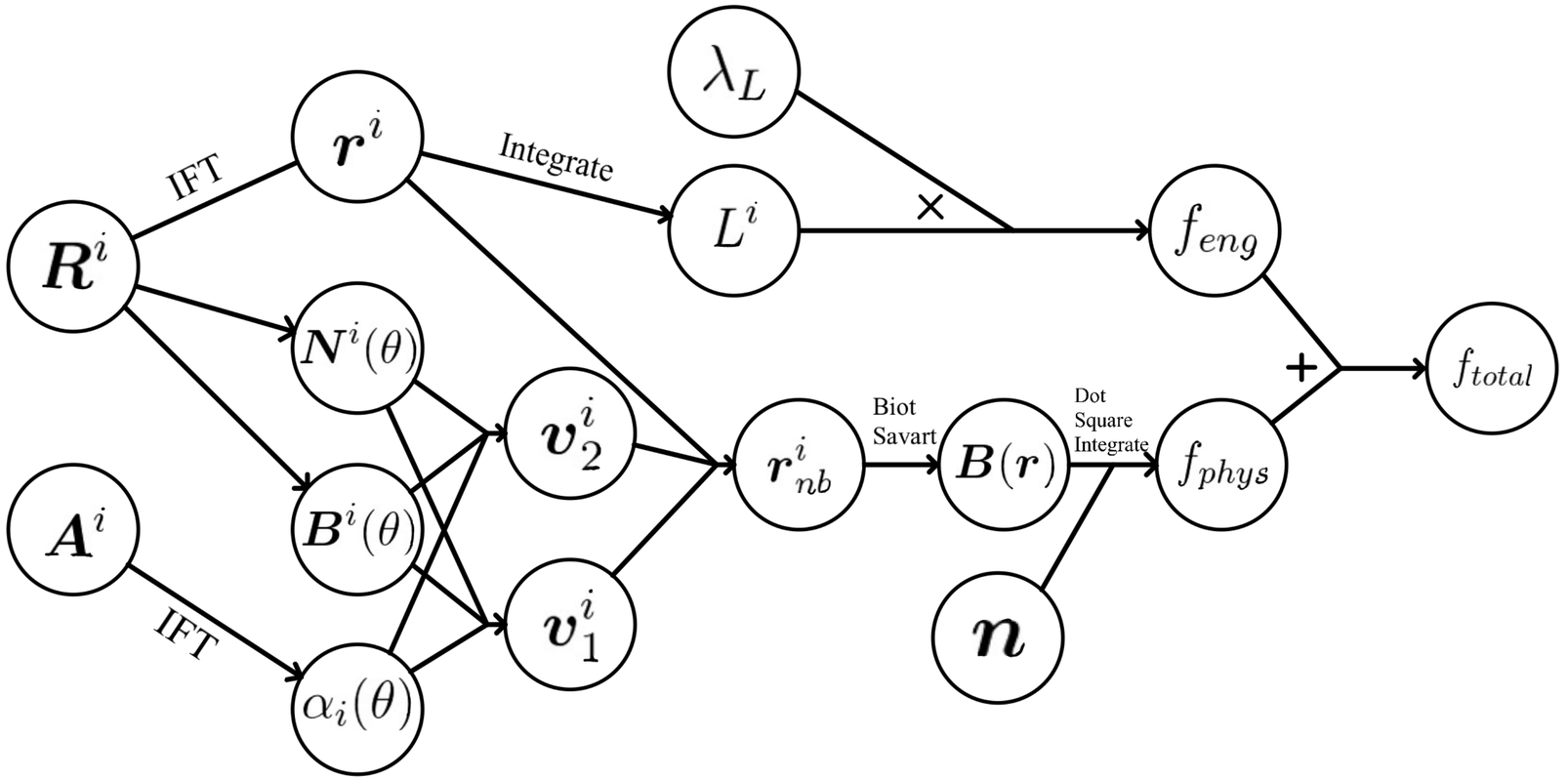
Coil design in fusion reactors (stellarator)

McGreivy et al 2009.00196

Computation graph

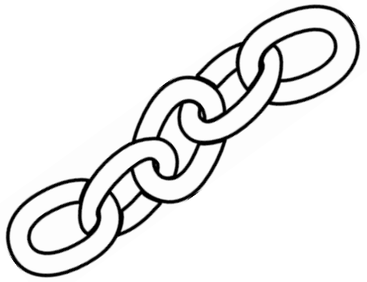
McGreivy et al 1909.04240

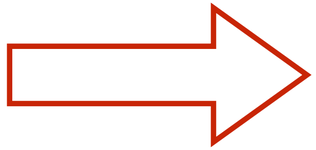
Coil parameters

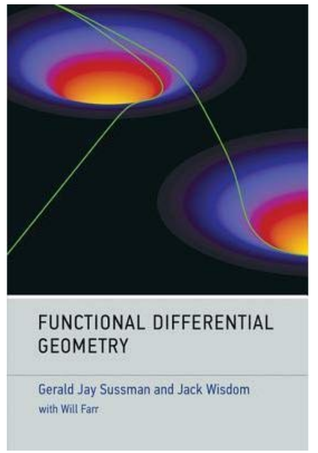


Total cost

Differentiable programming is more than training neural networks







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

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

Forward mode AD: Jacobian-Vector Product of primitives

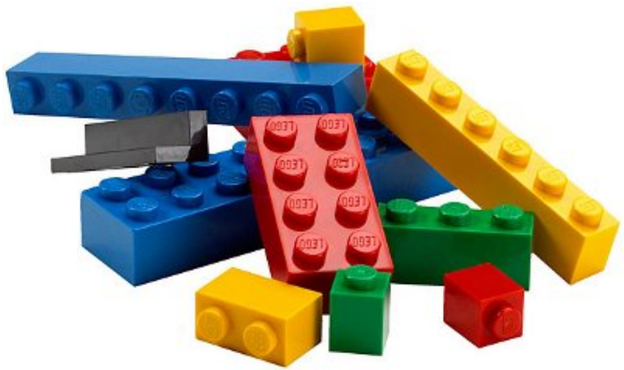
- Same order with the function evaluation
- No storage overhead
- Efficient for graph with large fan-out

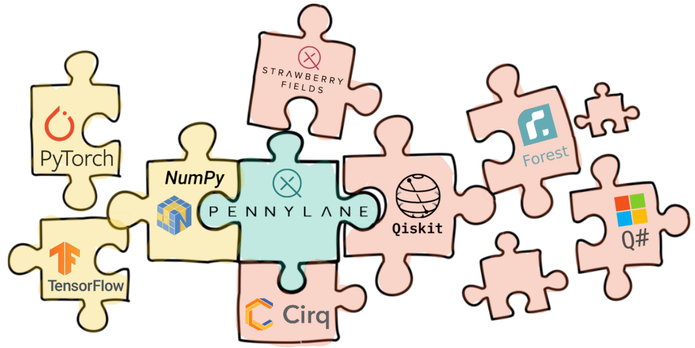
Less efficient for scalar output, but useful for higher-order derivatives

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)





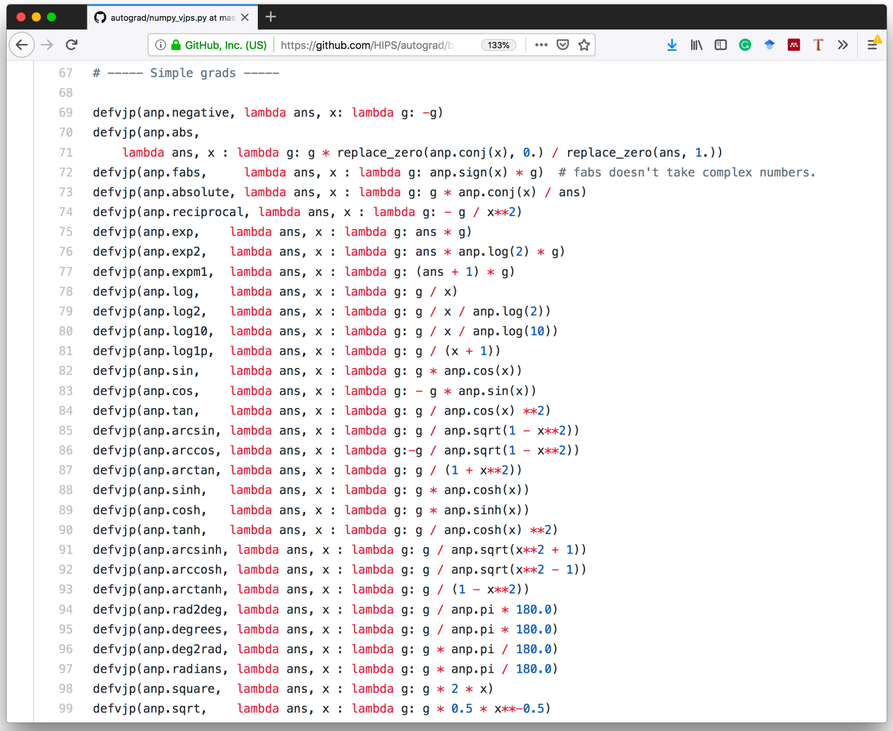
https://github.com/PennyLaneAI/pennylane

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

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

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 $\partial\mathscr{L}/\partial E$? **Inverse perturbation theory!**

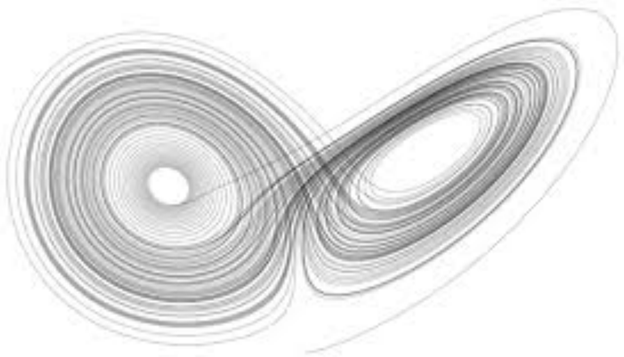
Hamiltonian engineering via differentiable programming

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

Differentiable ODE integrators

"Neural ODE" Chen et al, 1806.07366

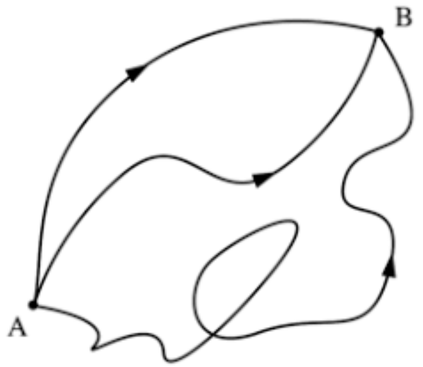
Dynamics systems



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

Classical and quantum control

Principle of least actions

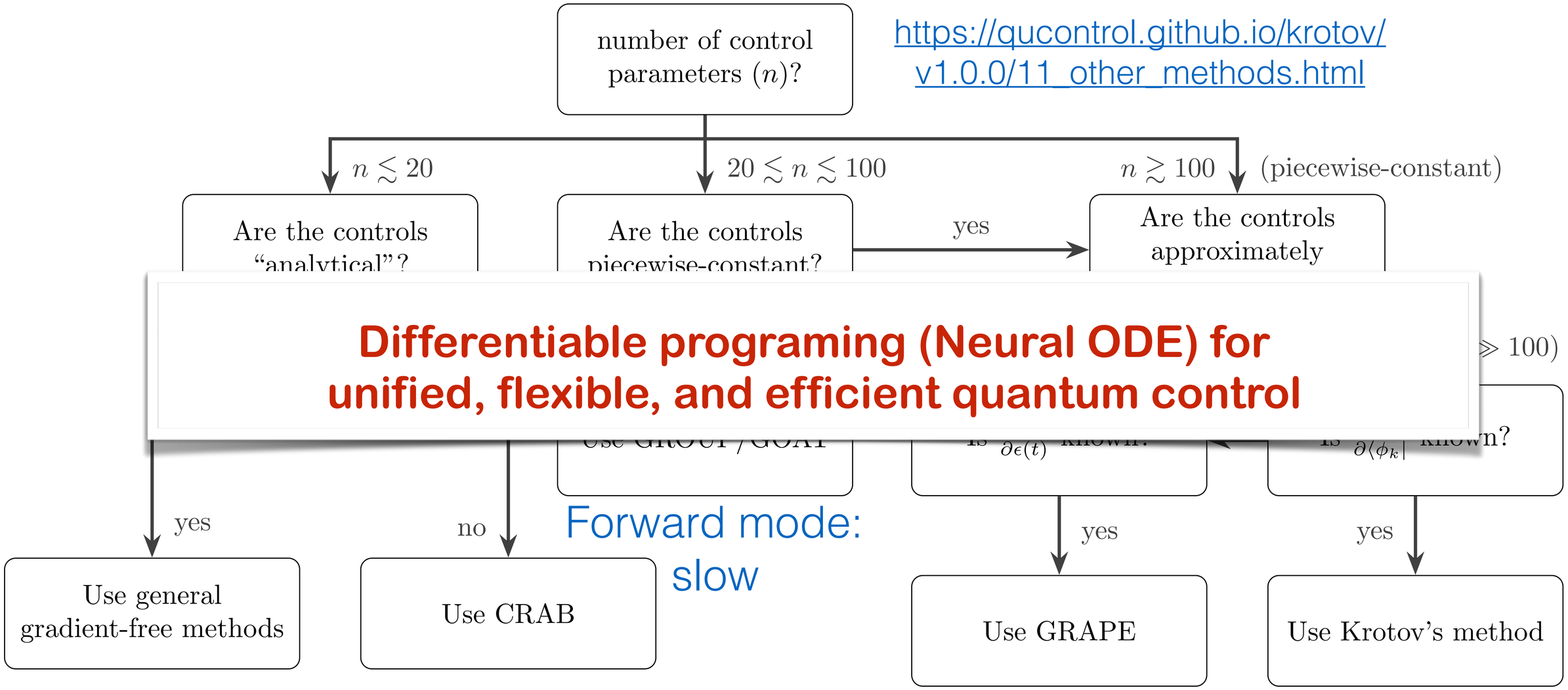


$$S = \int \mathscr{L}(q_\theta, \dot{q}_\theta, t)dt$$

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

Forward mode: slow

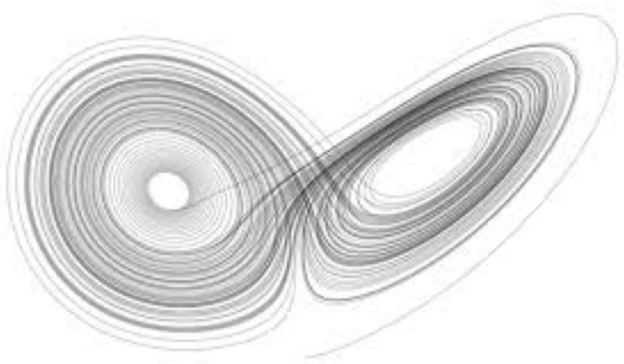
No gradient: not scalable

Reverse mode w/ discretize steps: piesewise-constant assumption

Differentiable ODE integrators

"Neural ODE" Chen et al, 1806.07366

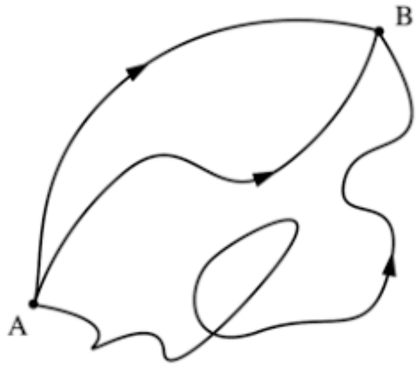
Dynamics systems



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

Classical and quantum control

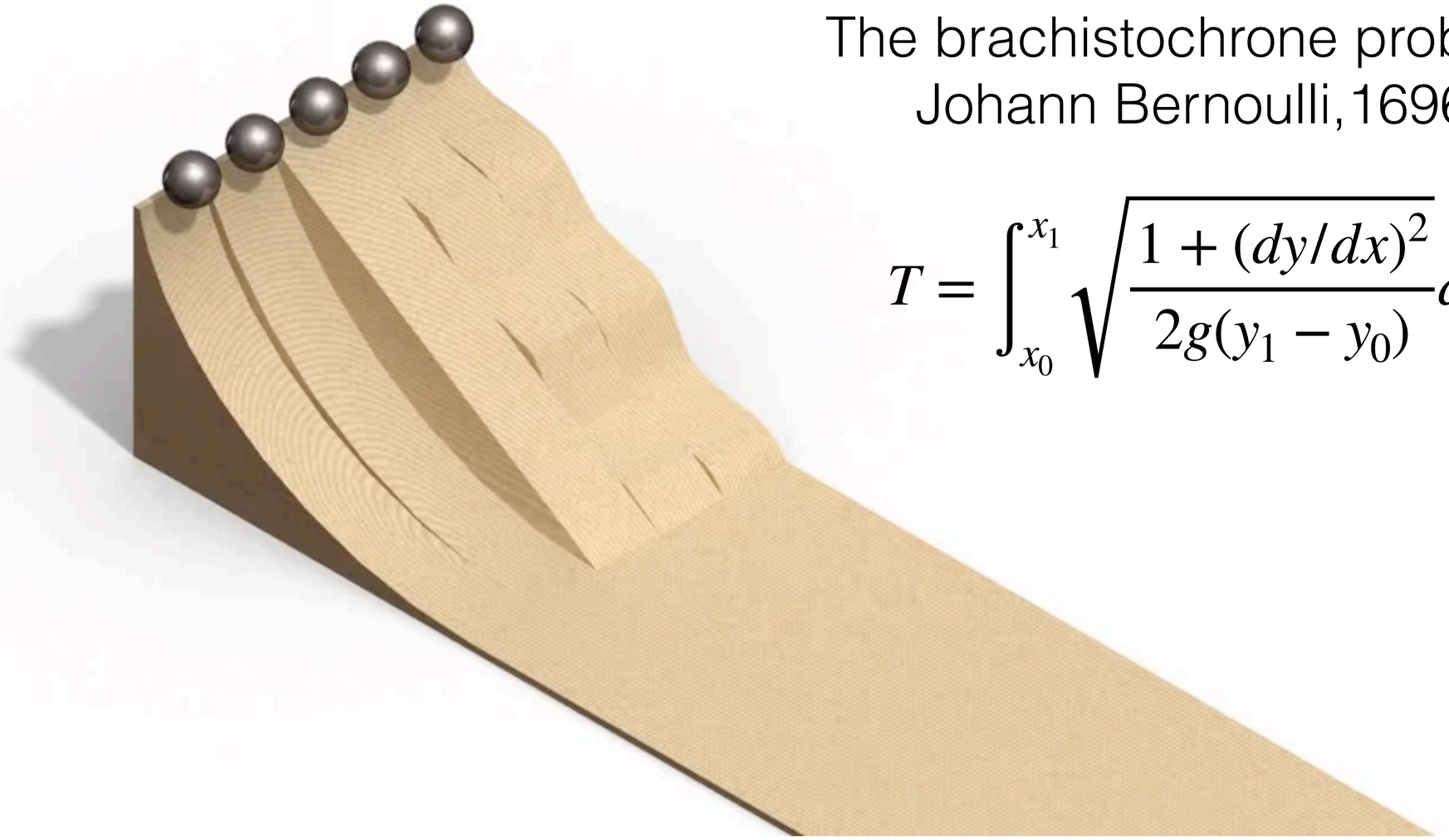
Principle of least actions



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

Optics, (quantum) mechanics, field theory…

Differentiable functional optimization

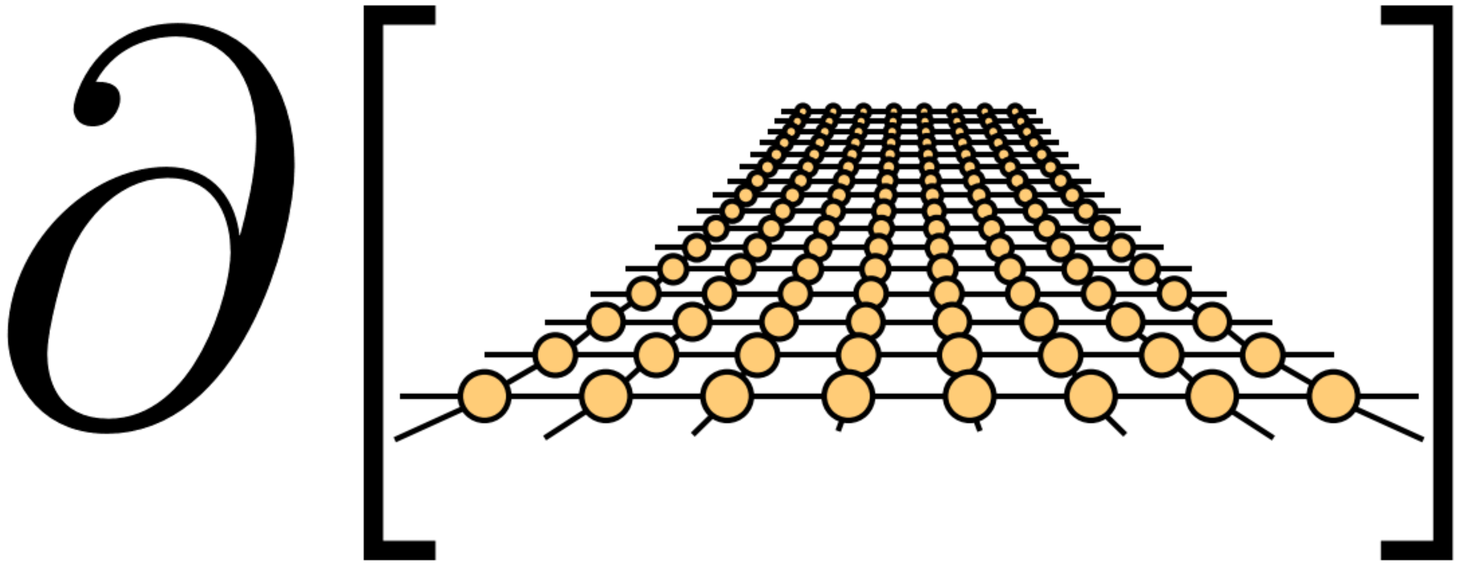


The brachistochrone problem
Johann Bernoulli, 1696

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

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

Differentiable Programming Tensor Networks

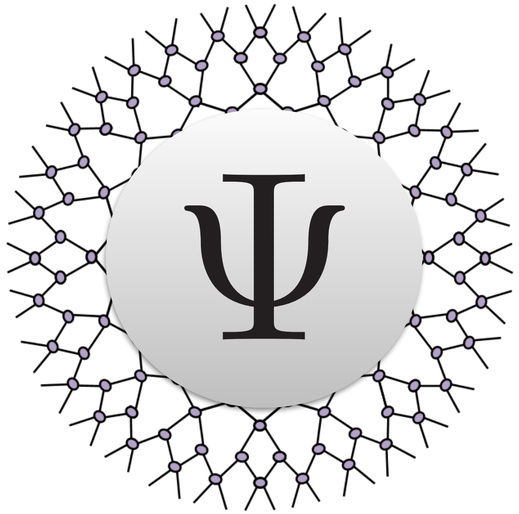


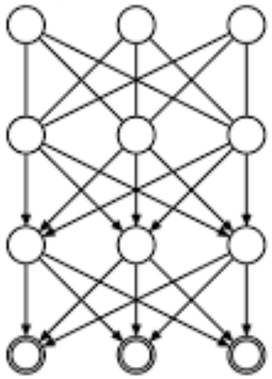
Liao, Liu, LW, Xiang, 1903.09650, PRX ‘19

https://github.com/wangleiphy/tensorgrad

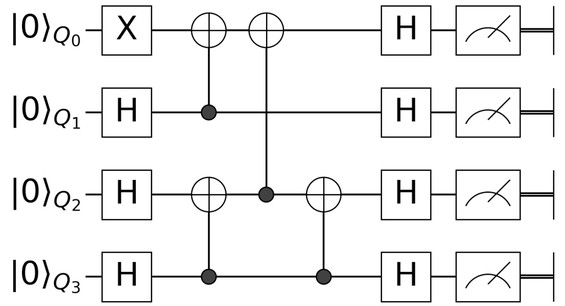
“Tensor network is 21 century’s matrix”

—Mario Szegedy



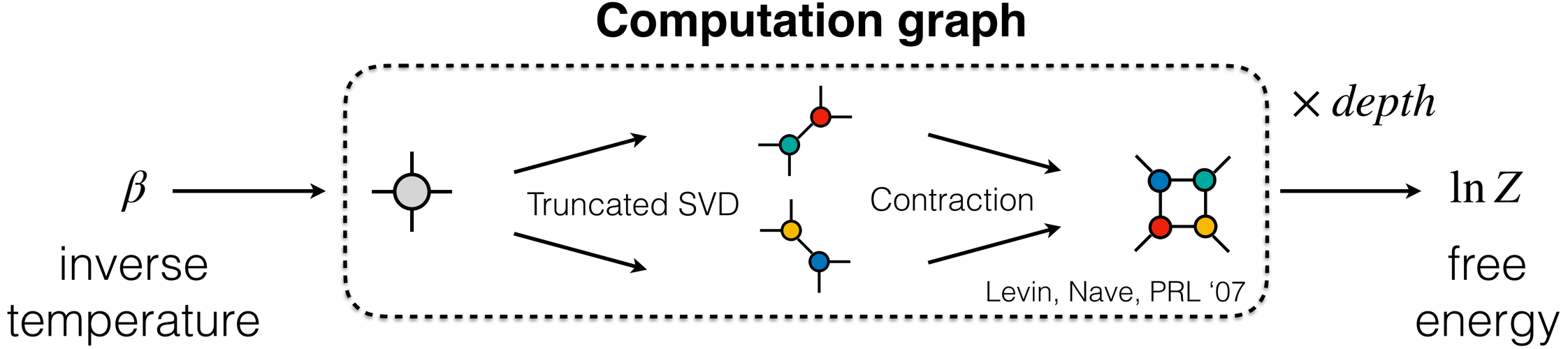


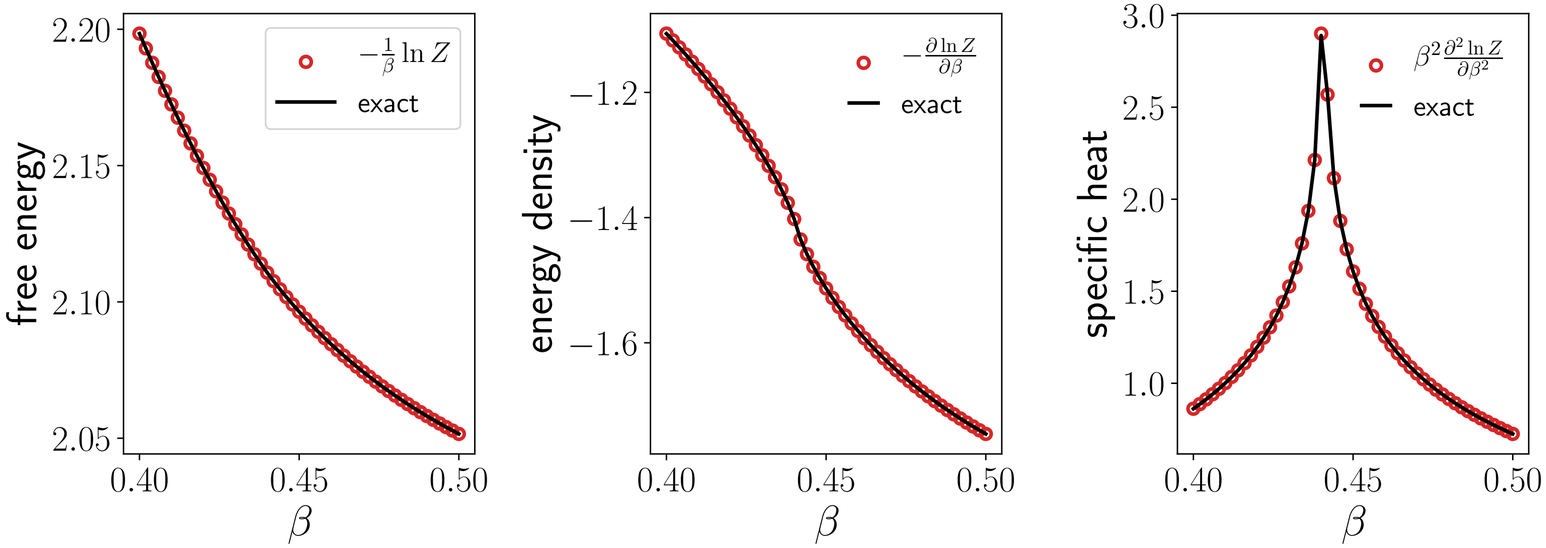
Neural networks and Probabilistic graphical models



Quantum circuit architecture, parametrization, and simulation

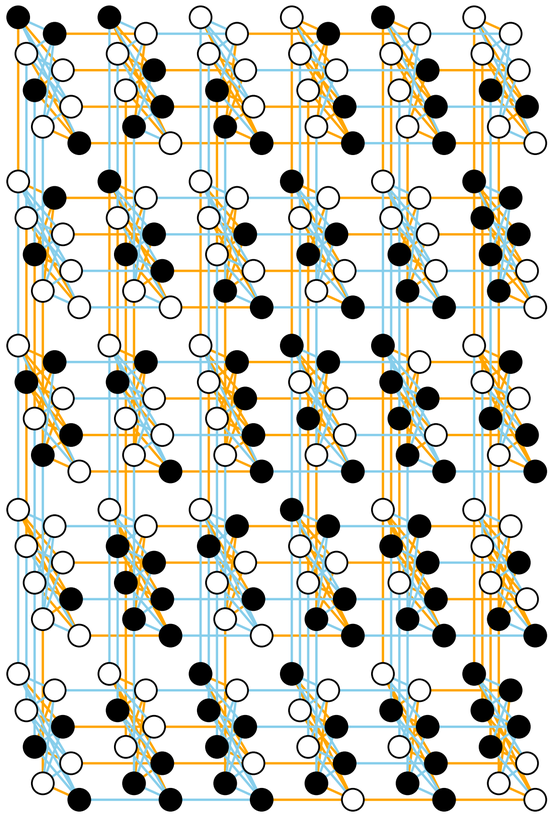
Differentiate through tensor renormalization group





Compute physical observables as gradient of tensor network contraction

Differentiable spin glass solver



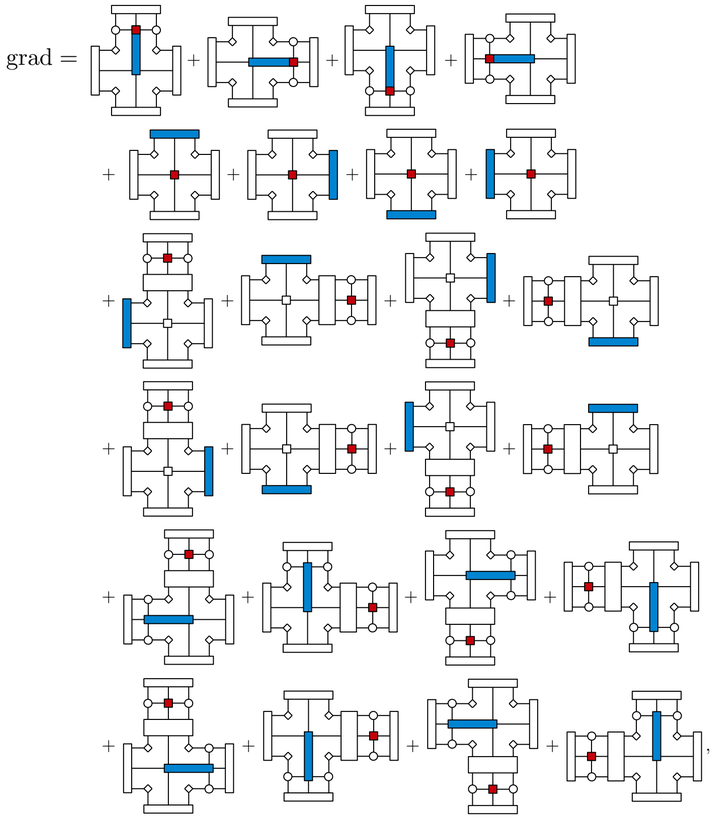
$$\text{optimal configuration} = \frac{\partial[\text{optimal energy}]}{\partial[\text{field}]}$$

Liu, LW, Zhang, 2008.06888

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

Differentiable iPEPS optimization

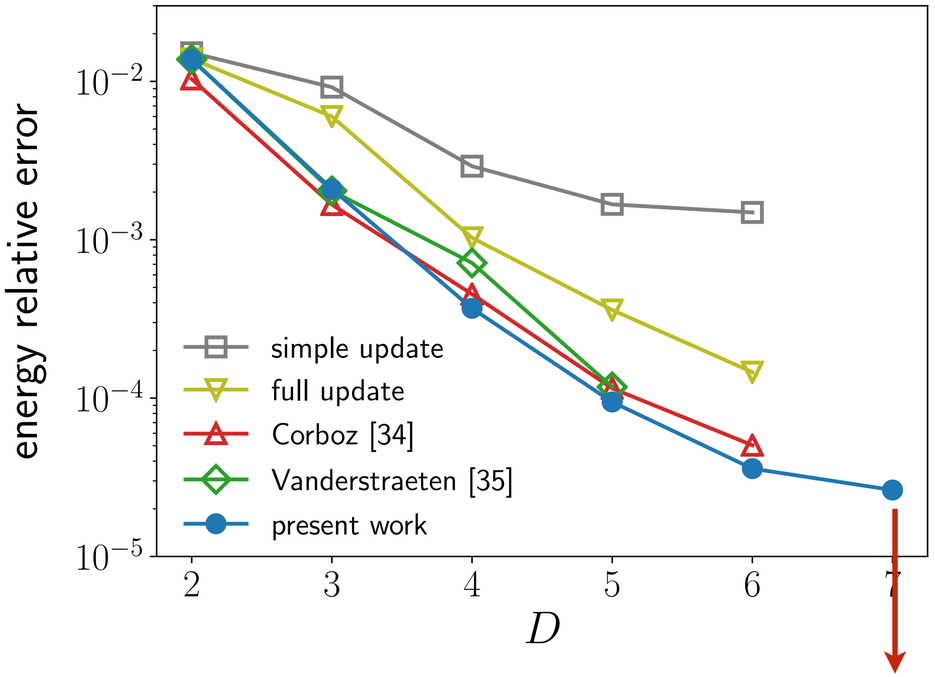
before…



Vanderstraeten et al, PRB ‘16

now, w/ differentiable programming

Liao, Liu, LW, Xiang, PRX ‘19



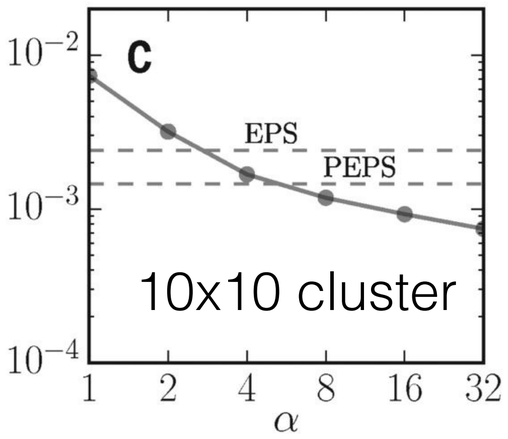
Best variational energy to date

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

Differentiable iPEPS optimization

Finite size

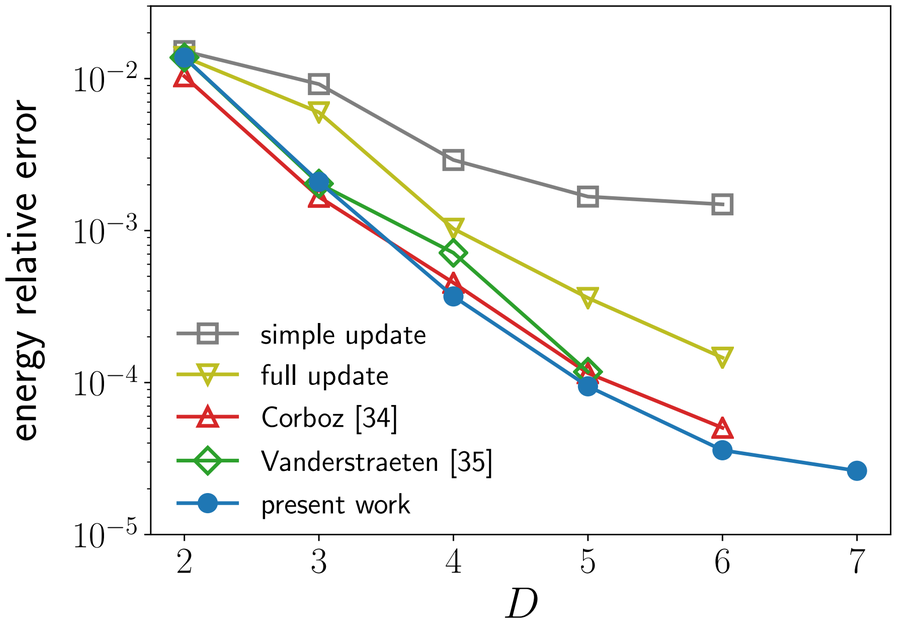
Neural network



Carleo & Troyer, Science ‘17

Infinite size

Tensor network



Liao, Liu, LW, Xiang, PRX ‘19

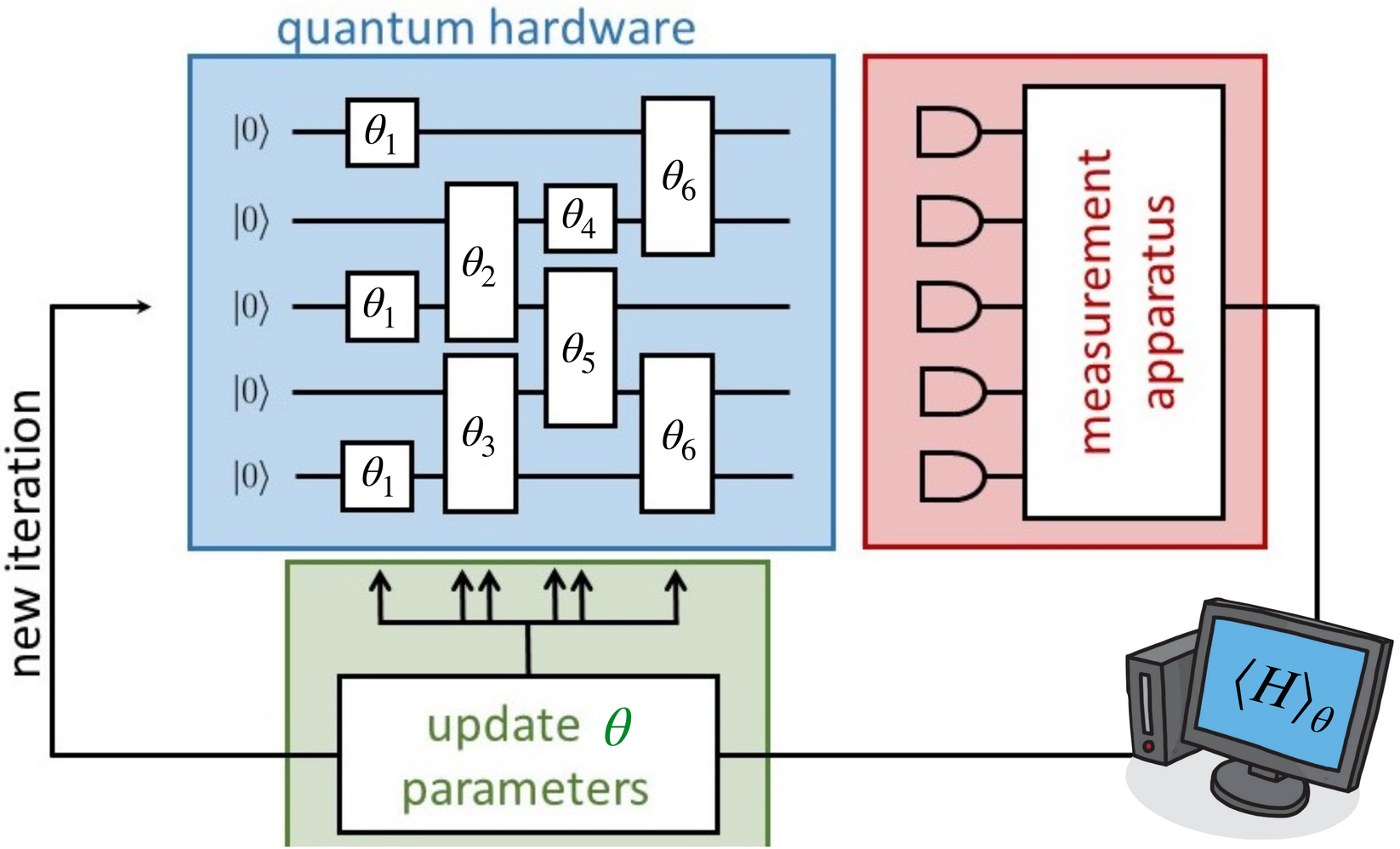
Further progress for challenging physical problems: frustrated magnets, fermions, thermodynamics …

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

Differentiable Programming Quantum Circuits

neural networks — graphical models — tensor networks — quantum circuits

Variational quantum algorithms

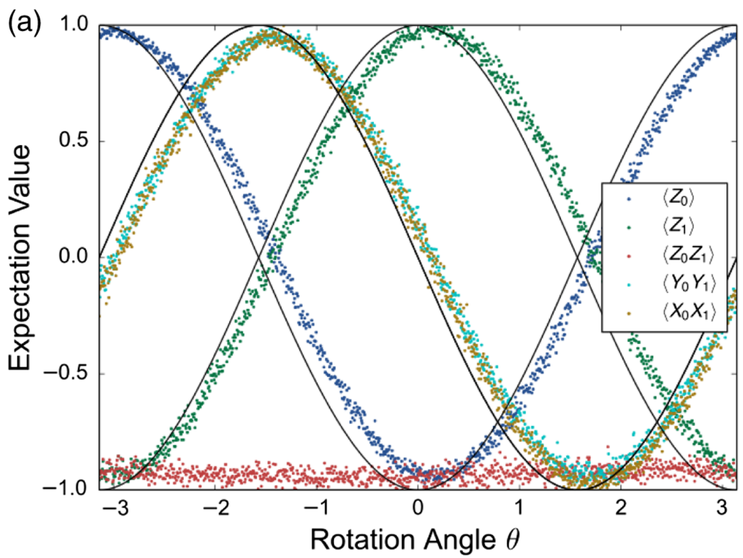


Peruzzo et al, Nat. Comm. '13

Quantum circuit as a variational ansatz

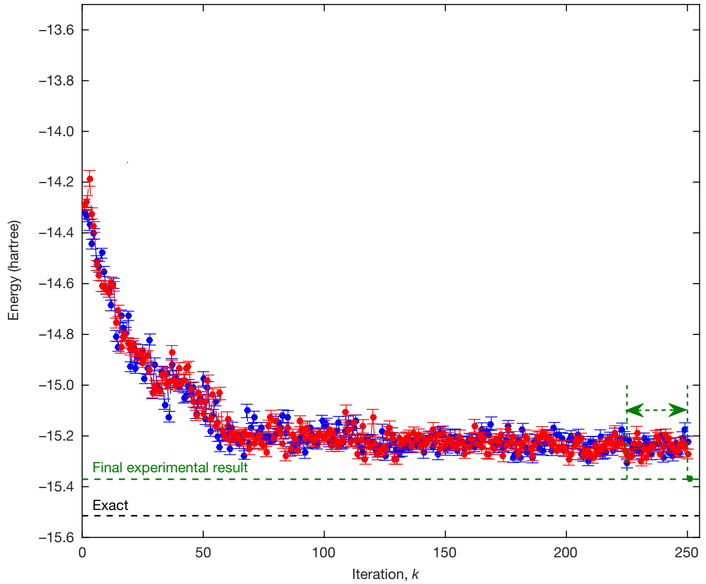
Optimize variational quantum circuits

Scan the single variational parameter



Google PRX ‘16

Stochastic perturbation of 30 variational parameters

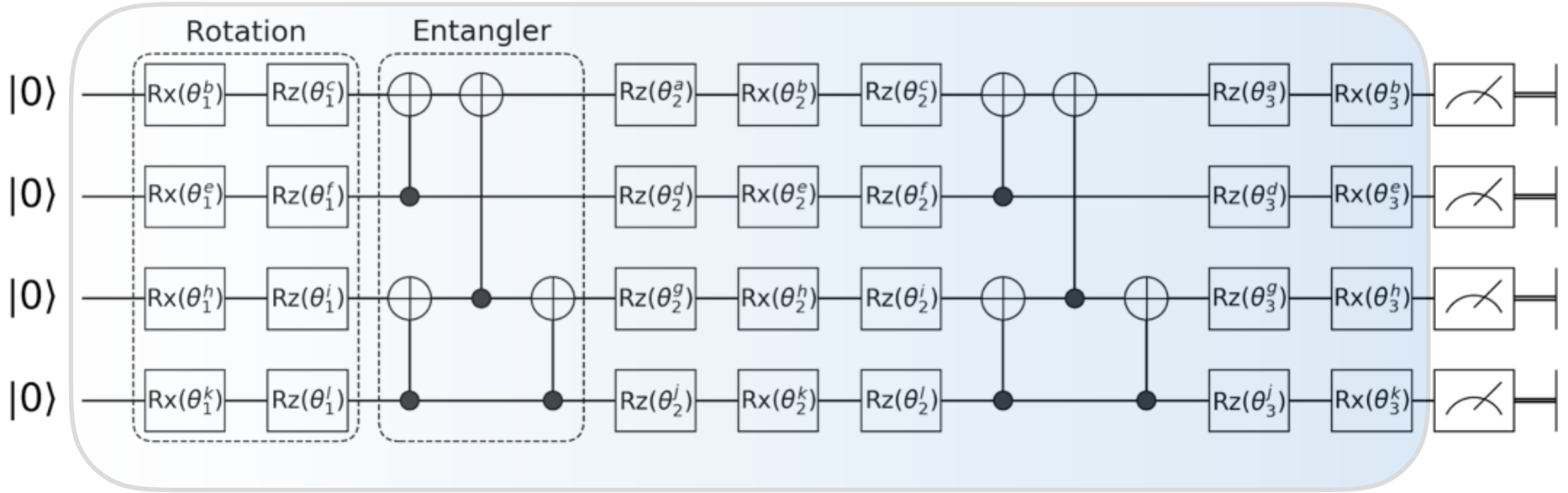


IBM Nature ‘17

Optimization with analytical gradient is essential for higher dimensions

Differentiable[1] quantum circuits

measure gradient on real device



Parametrized gate of the form

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

e.g., X, Y, Z, CNOT, SWAP…

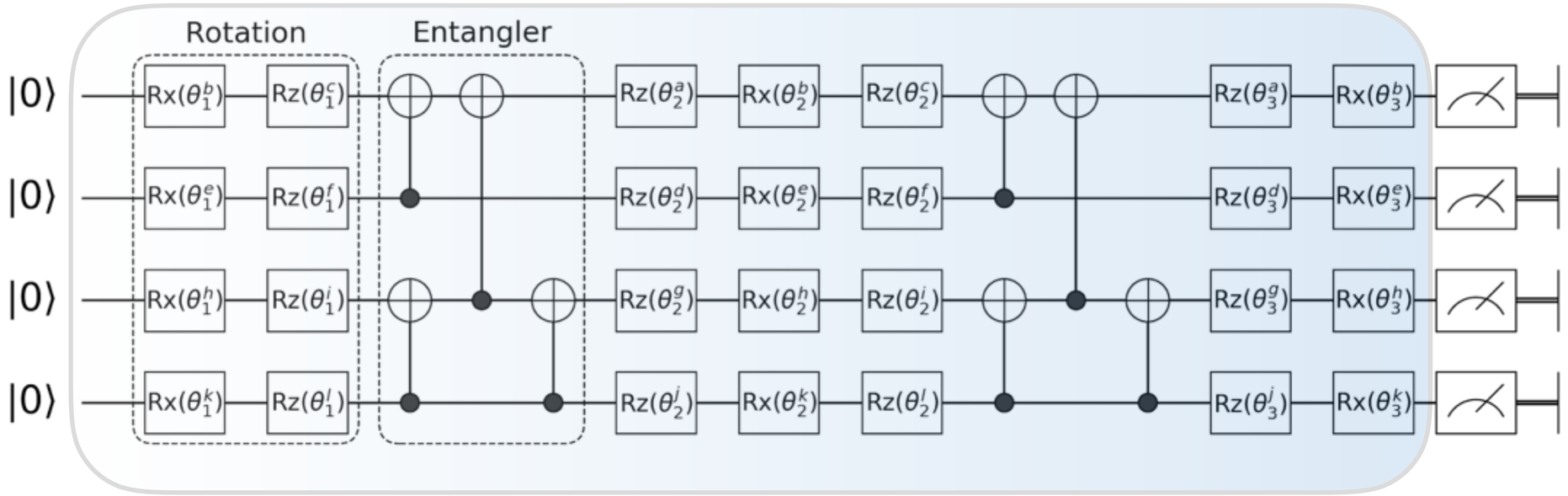
Li et al, PRL ’17, Mitarai et al, PRA ’18
Schuld et al, PRA ’19, Crooks, ’19…

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

Same complexity as forward mode automatic differentiation

Differentiable^2 quantum circuits

compute gradient in classical simulations

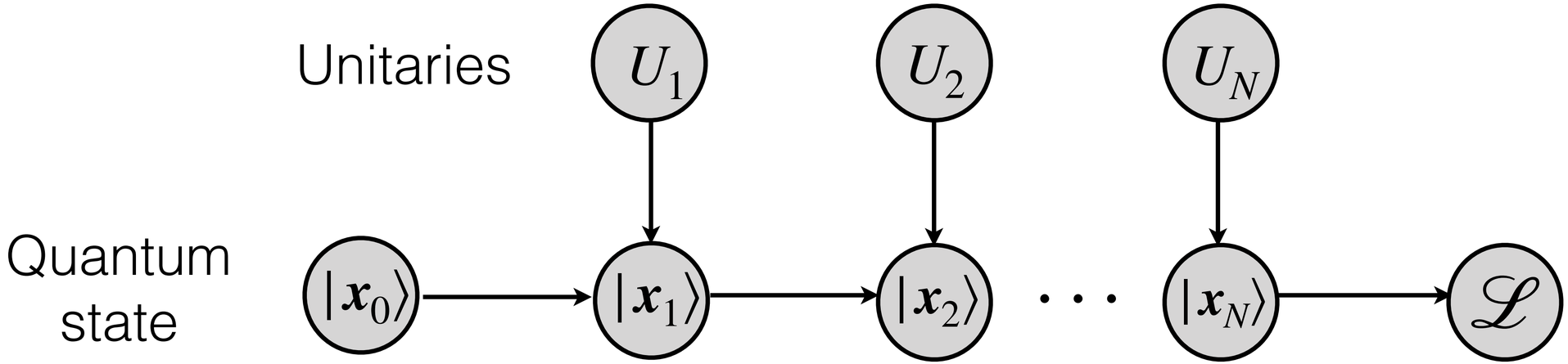


PENNYLANE

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

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

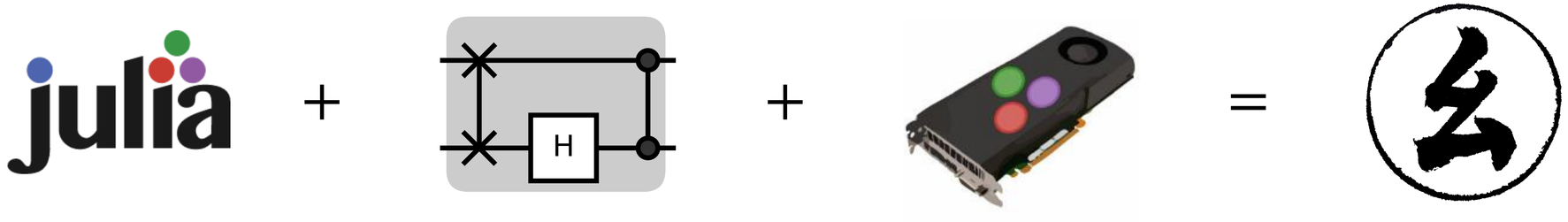
*GRAPE type algorithm on the level of circuits

```
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



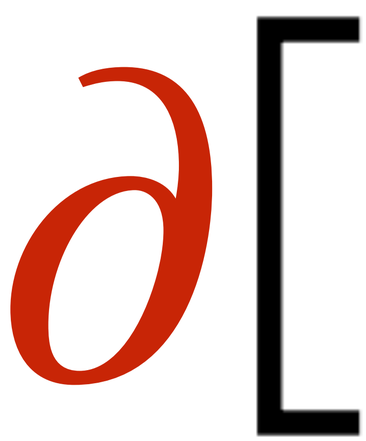
Xiu-Zhe Luo (IOP, CAS → Waterloo & PI)
Jin-Guo Liu (IOP, CAS → QuEra Computing & Harvard)

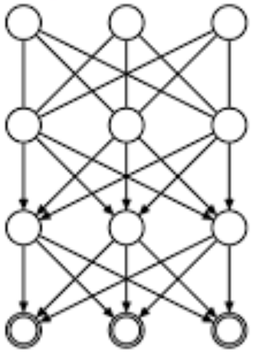
Features:

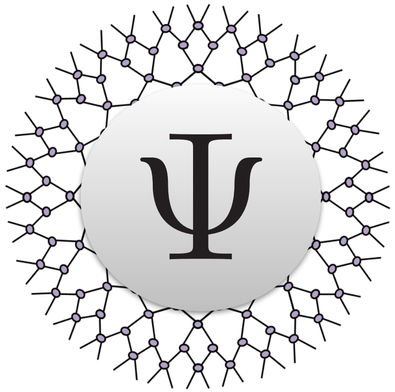
- Reversible AD engine for quantum circuits
- Batch parallelization with GPU acceleration
- Quantum block intermediate representation

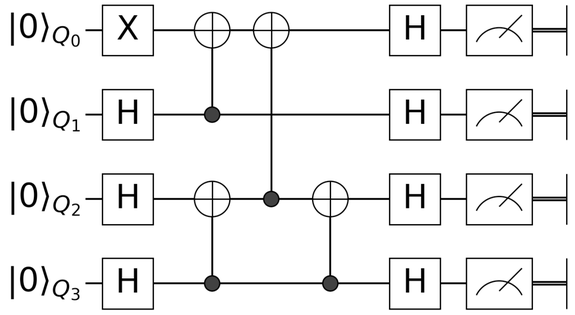
Luo, Liu, Zhang and LW, 1912.10877

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



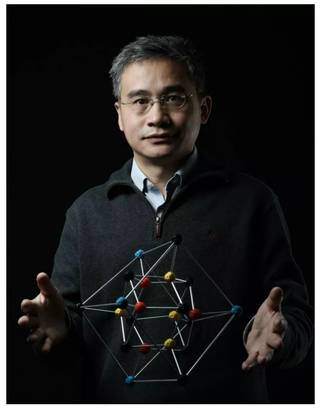






Thank you!

Hai-Jun Liao
IOP CAS



Tao Xiang
IOP CAS

Pan Zhang
ITP CAS

Jin-Guo Liu,
QuEra & Harvard

Xiu-Zhe Luo
Waterloo & PI