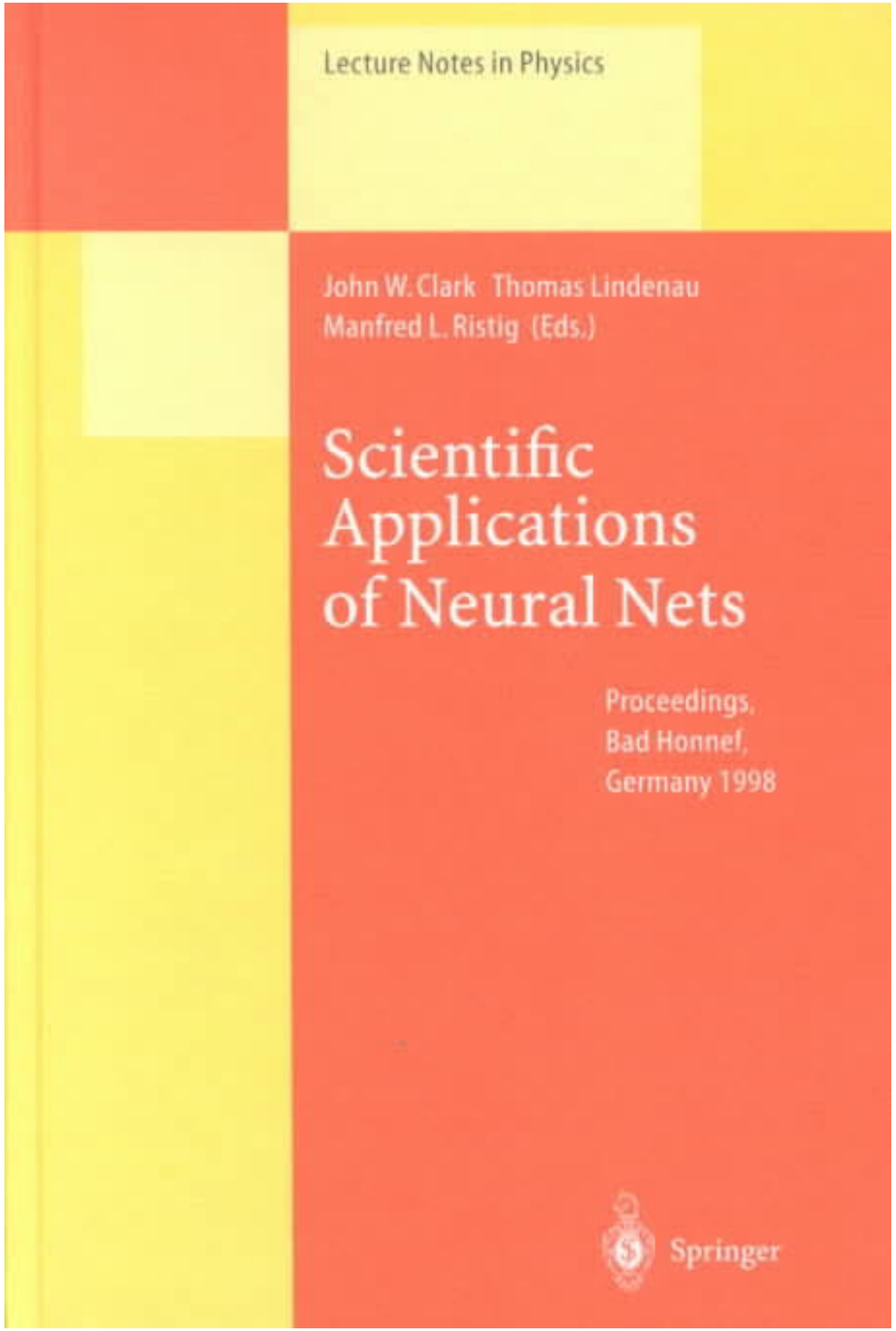
Secrets of deep learning

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



Reflection between this and last hype cycles

8 Doing Science With Neural Nets: Pride and Prejudice

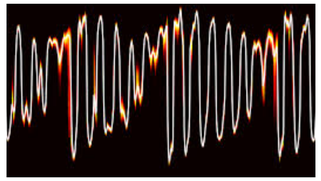
When neural networks re-emerged on the scene in the mid-80s as a new and glamorous computational paradigm, the initial reaction in some sectors of the scientific community was perhaps too enthusiastic and not sufficiently critical. There was a tendency on the part of practitioners to oversell the powers of neural-network or "connectionist" solutions relative to conventional techniques – where conventional techniques can include both traditional theory-rich modeling and established statistical methods. The last five years have seen a correction phase, as some of the practical limitations of neural-network approaches have become apparent, and as scientists have become better acquainted with the wide array of advanced statistical tools that are currently available.

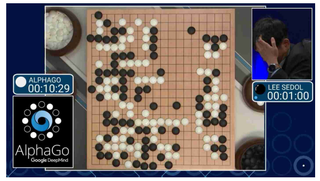
Why now, again ?

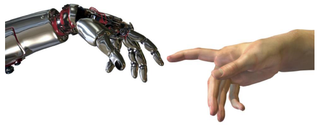
What has changed ?

What has not ?

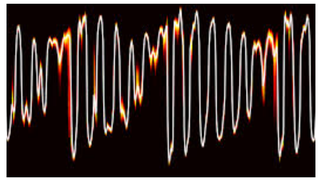
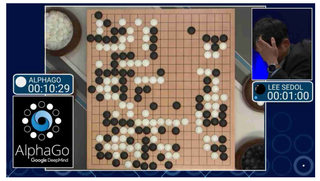
What is the secret behind deep learning ?

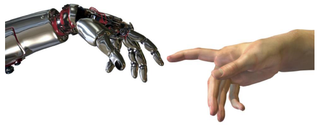






What is the secret behind deep learning ?

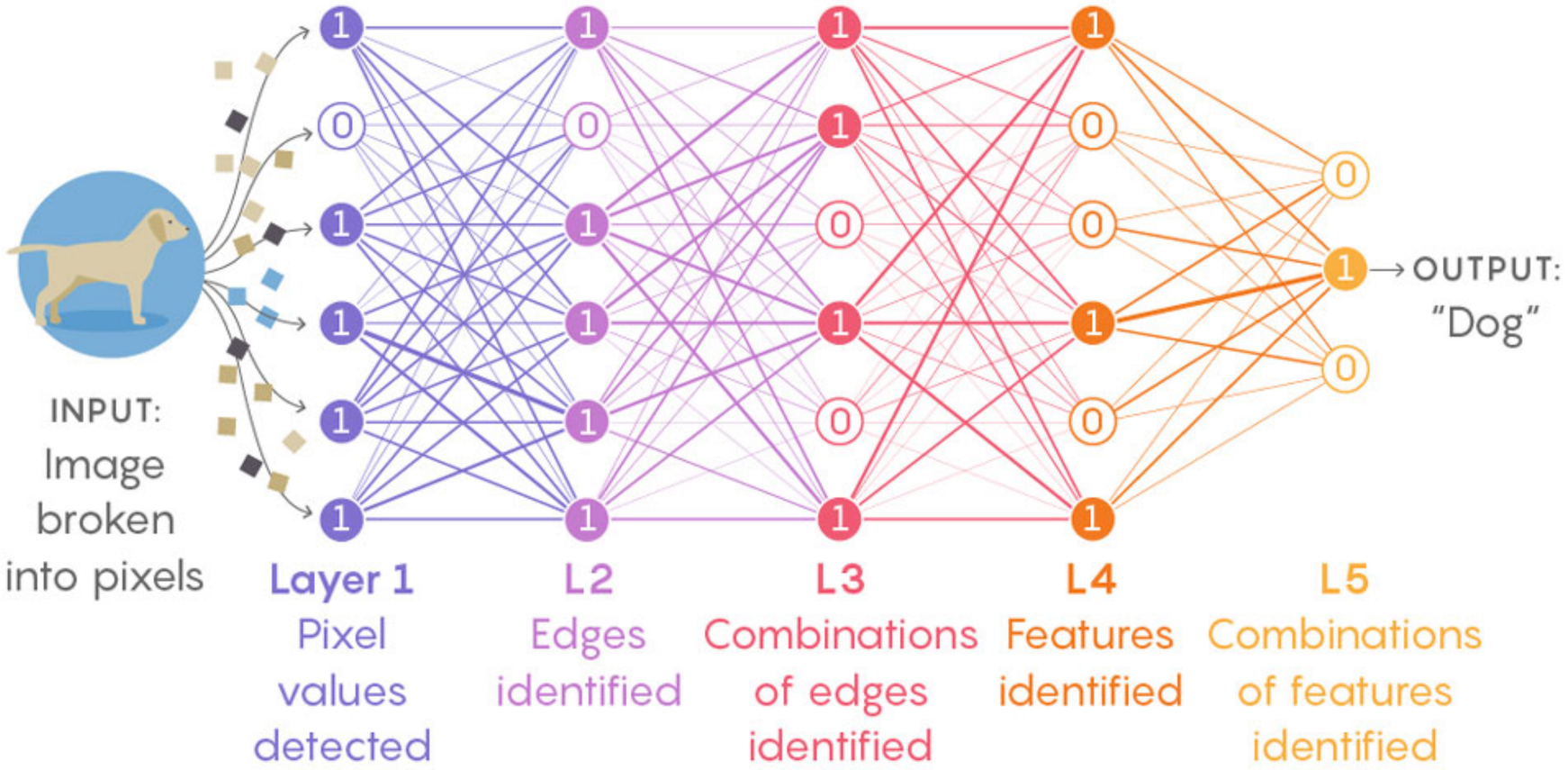





Key technology: differentiable programing

Core idea: learning representation

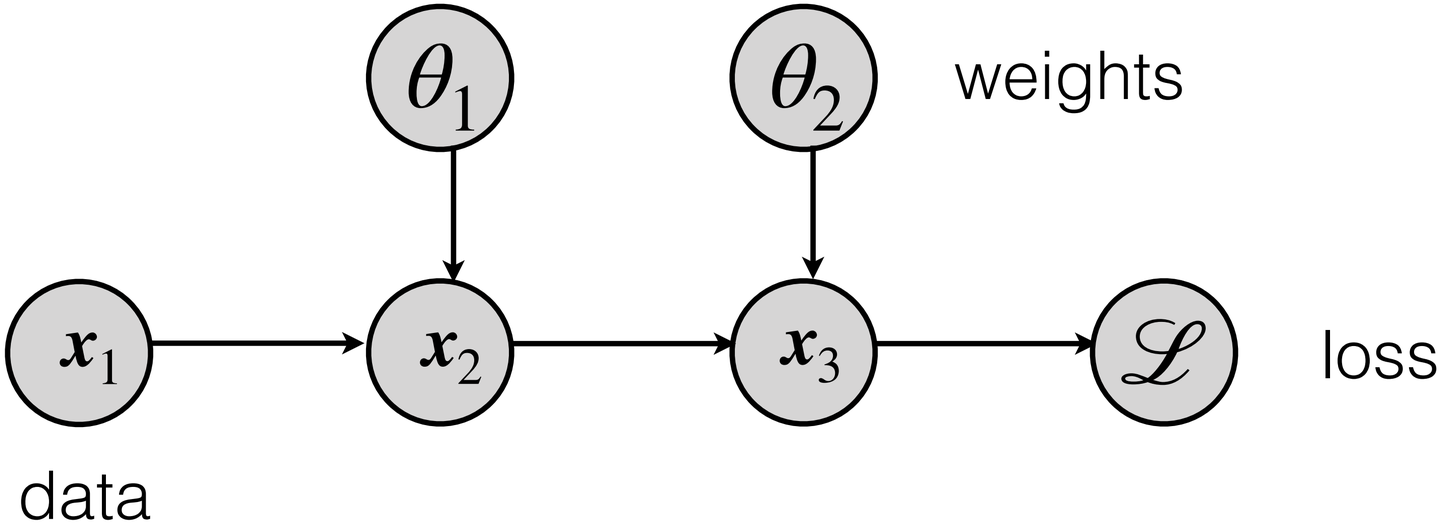
The engine of deep learning



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

Computation Graph

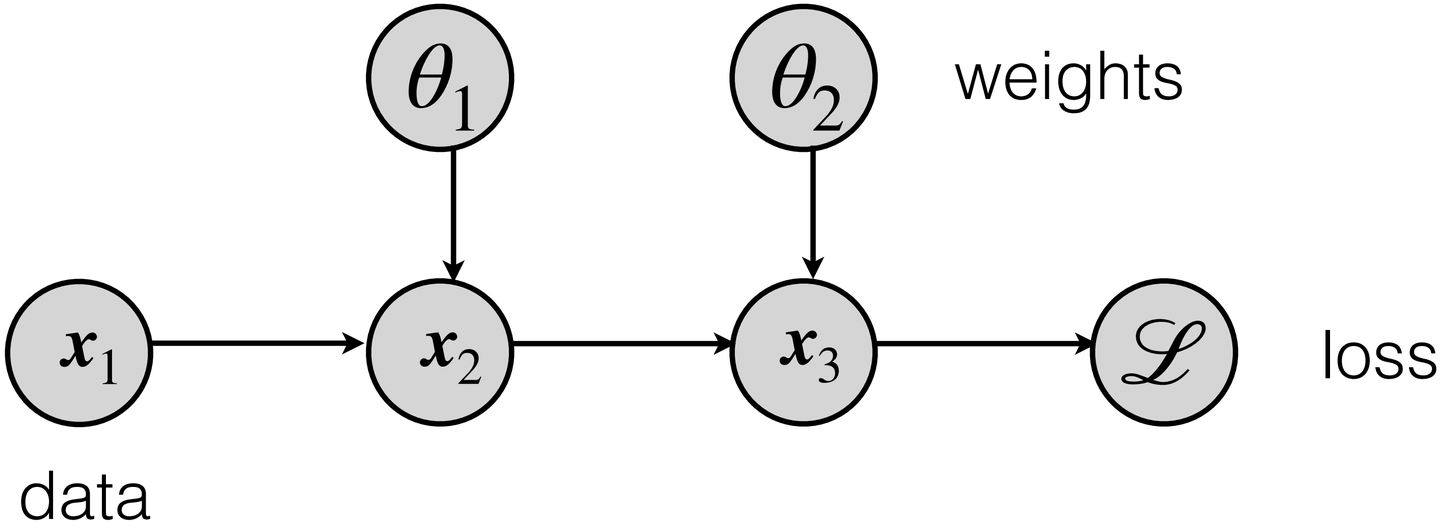
“comb" graph



Pullback the adjoint through the graph

Computation Graph

"comb" graph

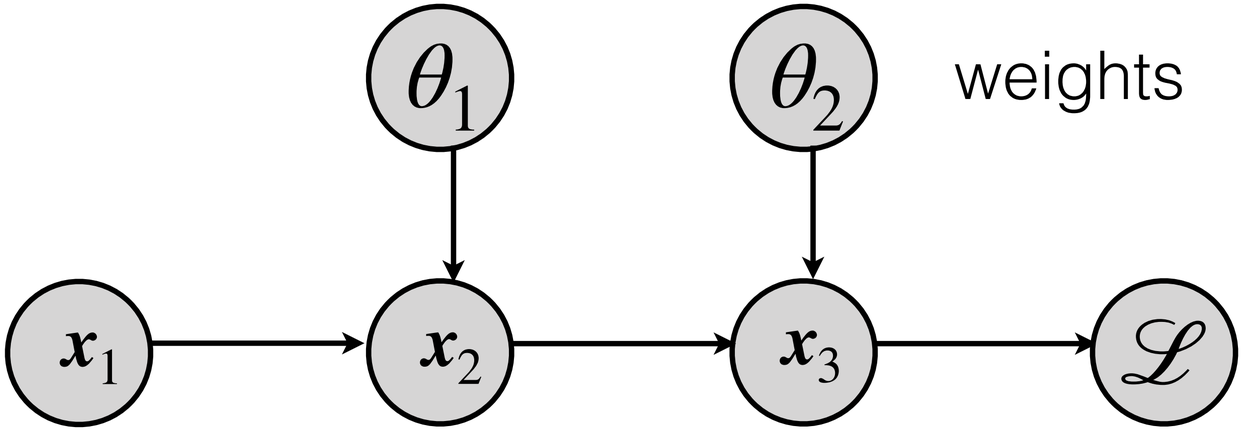


Define "adjoint" $\bar{x} = \dfrac{\partial \mathscr{L}}{\partial x}$

Pullback the adjoint through the graph

Computation Graph

"comb" graph



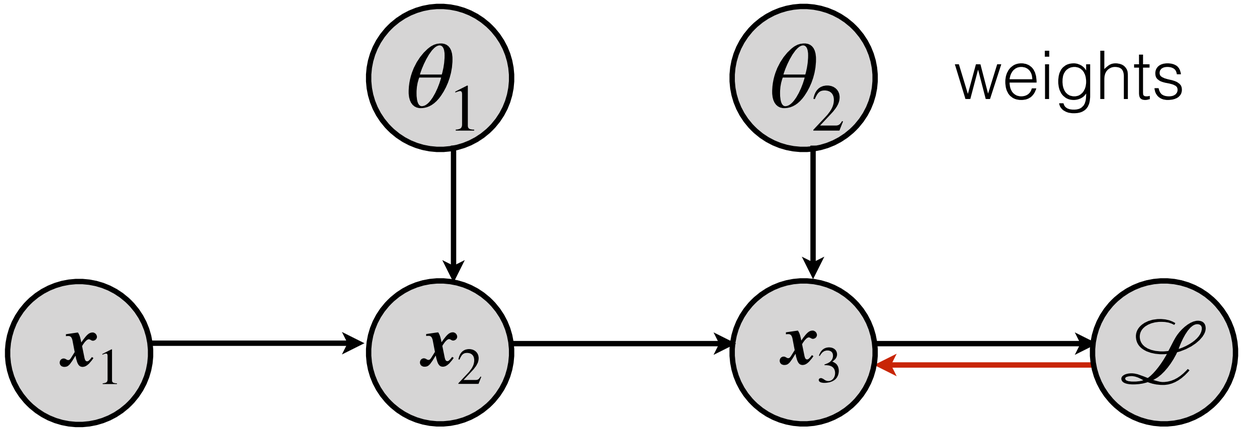
$\overline{\mathscr{L}} = 1$

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

Pullback the adjoint through the graph

Computation Graph

"comb" graph



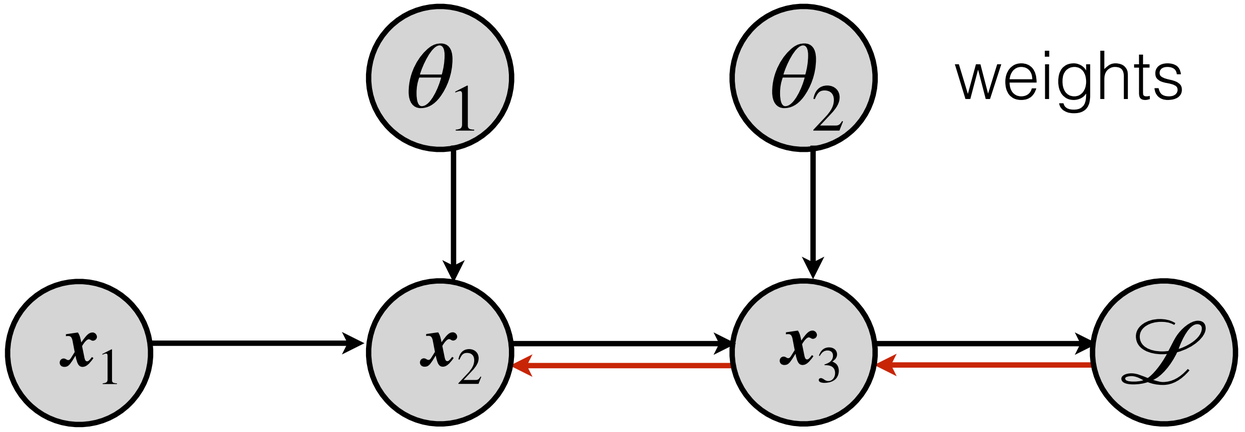
$$\overline{x_3} = \overline{\mathscr{L}} \frac{\partial \mathscr{L}}{\partial x_3} \qquad \overline{\mathscr{L}} = 1$$

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

Pullback the adjoint through the graph

Computation Graph

"comb" graph



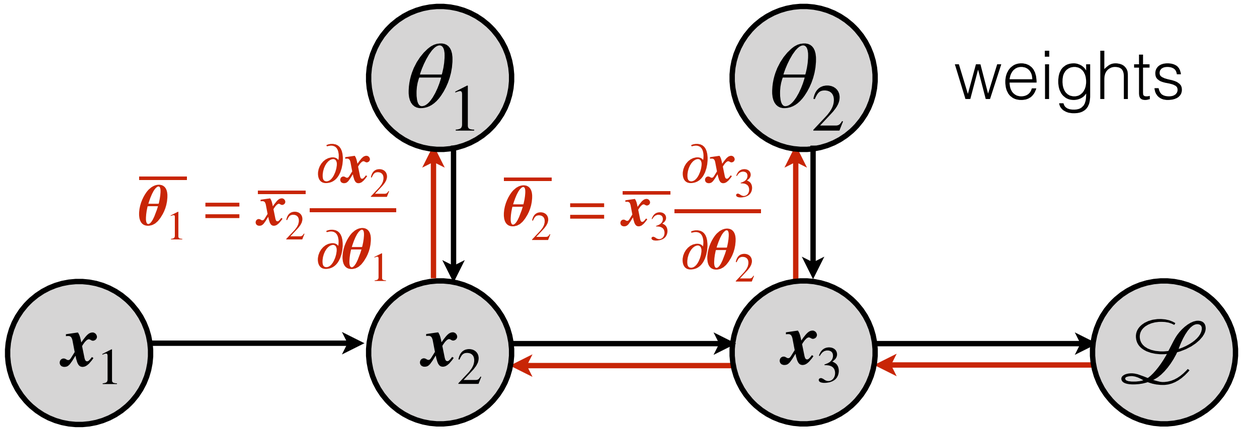
$$\overline{x_2} = \overline{x_3}\frac{\partial x_3}{\partial x_2} \qquad \overline{x_3} = \overline{\mathscr{L}}\frac{\partial \mathscr{L}}{\partial x_3} \qquad \overline{\mathscr{L}} = 1$$

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

Pullback the adjoint through the graph

Computation Graph

"comb" graph



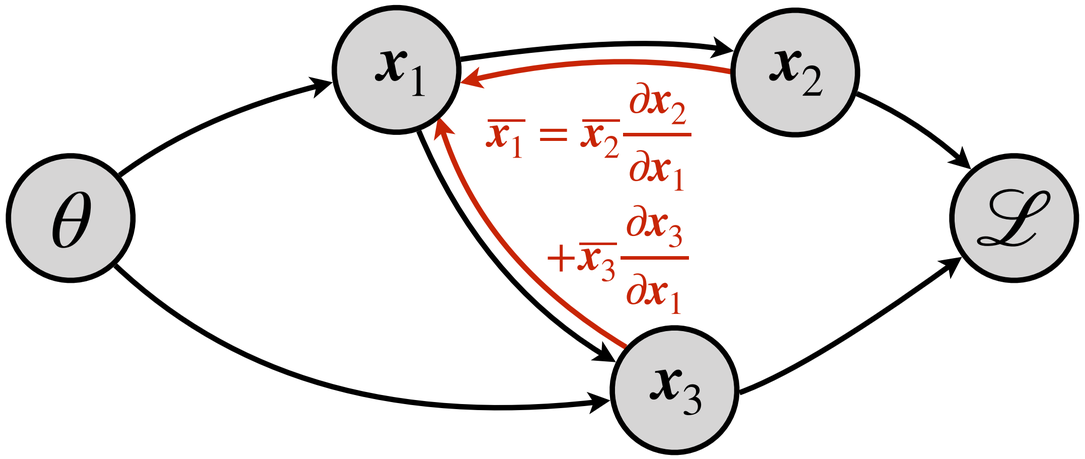
$$\overline{x_2} = \overline{x_3}\frac{\partial x_3}{\partial x_2} \qquad \overline{x_3} = \overline{\mathscr{L}}\frac{\partial \mathscr{L}}{\partial x_3} \qquad \overline{\mathscr{L}} = 1$$

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

Pullback the adjoint through the graph

Computation Graph

directed
acyclic graph

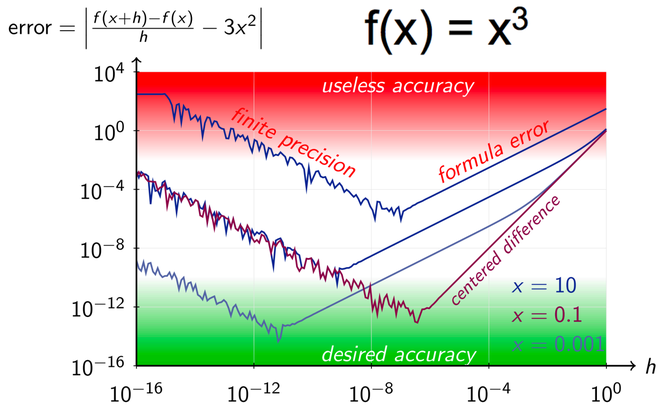


$$\overline{x_i} = \sum_{j:\,\text{child of } i} \overline{x_j}\,\frac{\partial x_j}{\partial x_i} \qquad \text{with} \quad \overline{\mathscr{L}} = 1$$

Message passing for the adjoint at each node

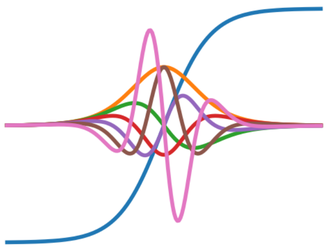
Advantages of automatic differentiation

- Accurate to the machine precision



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

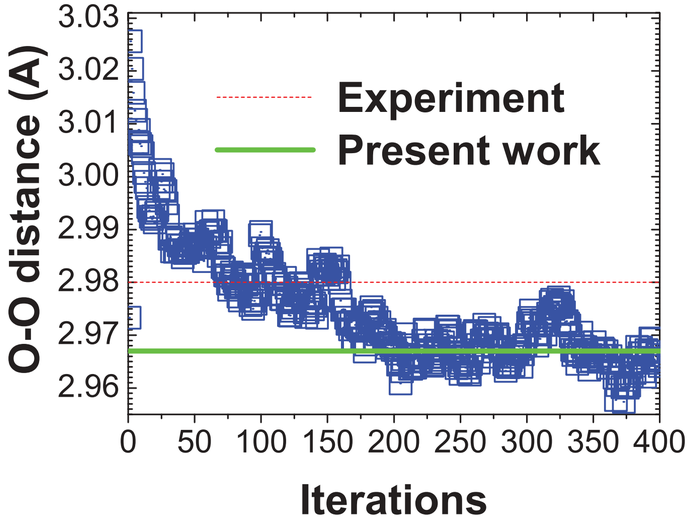
- Supports higher order gradients



```
>>> 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()
```

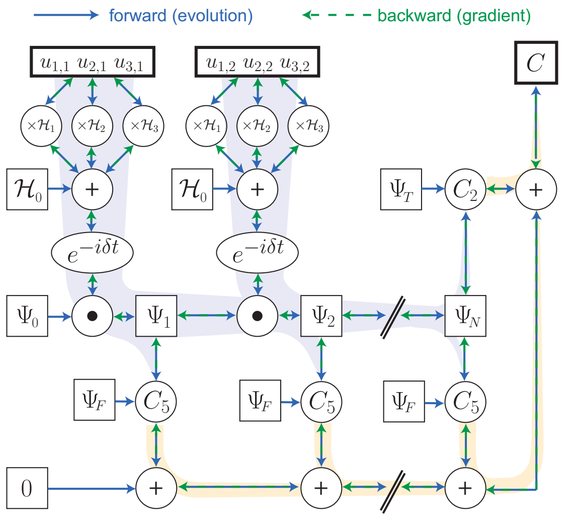
Applications of AD

Computing force



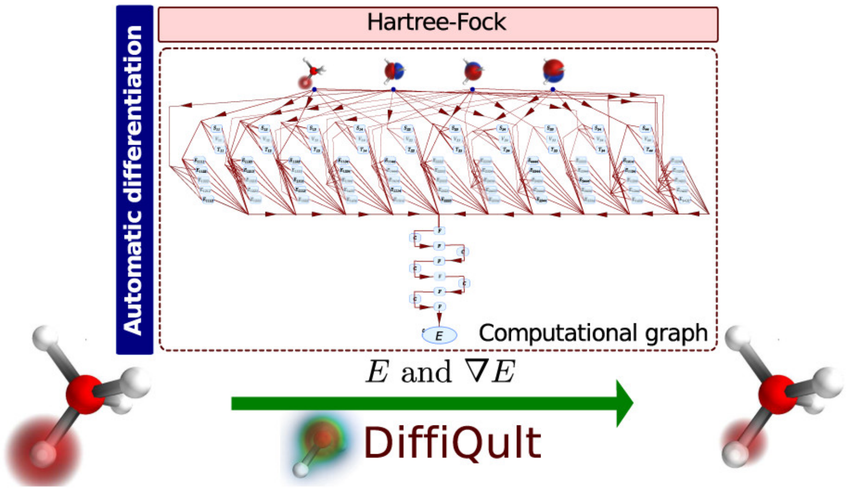
Sorella and Capriotti
J. Chem. Phys. ’10

Quantum optimal control



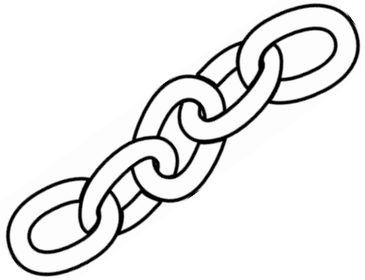
Leung et al
PRA ’17

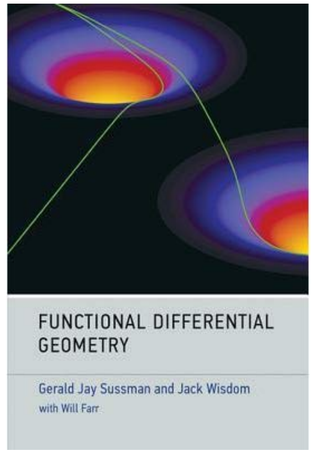
Variational Hartree-Fock



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

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

- Backtrace the computation graph $v_o (J)_{o \times i}$
- Needs to store intermediate results
- Efficient for graphs with large fan-in

Backpropagation = Reverse mode AD applied to neural networks

Reverse versus forward mode

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

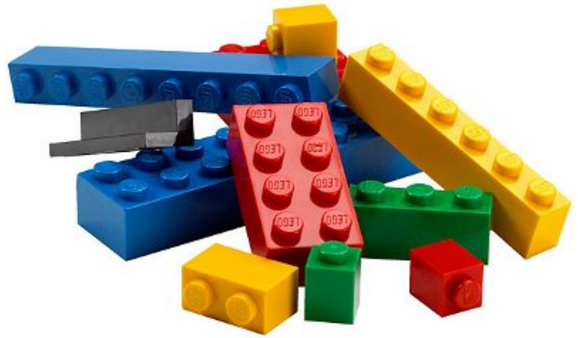
Forward mode AD: Jacobian-Vector Product of primitives

- 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)



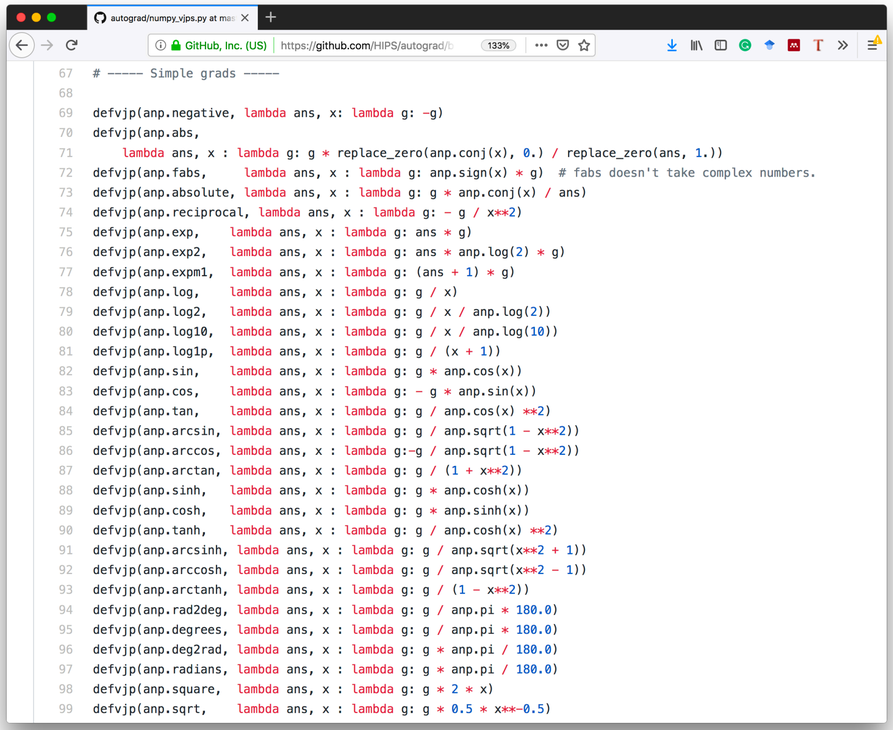
PENNYLANE

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,




```
# ----- 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 Scientific Programming

- Many 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 $\partial\mathscr{L}/\partial\Psi$ and $\partial\mathscr{L}/\partial\Lambda$? **Inverse perturbation theory!**

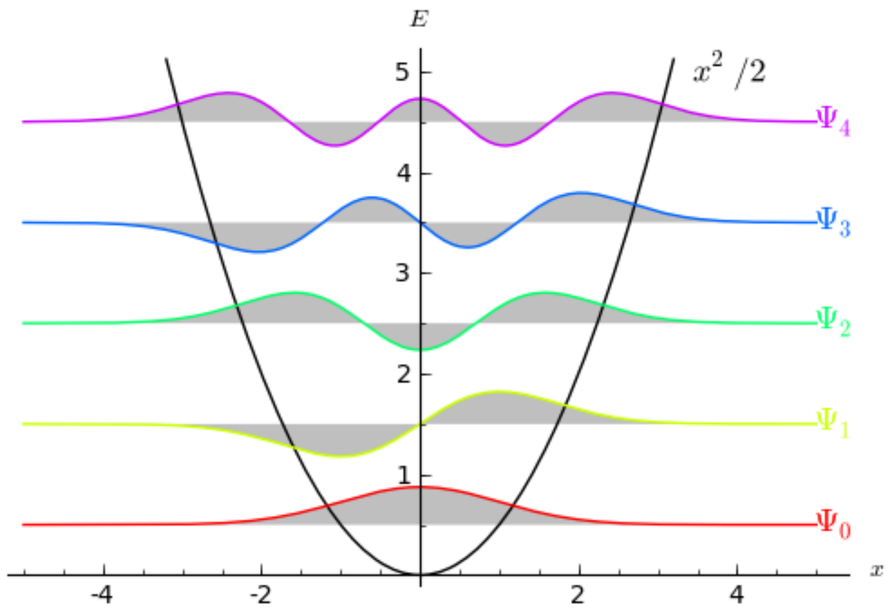
Hamiltonian engineering via differentiable programming

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

Demo: Inverse Schrodinger Solver

Given a target ground state, how to design the potential ?

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



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

Exercise

0. Understand what is under the hood.

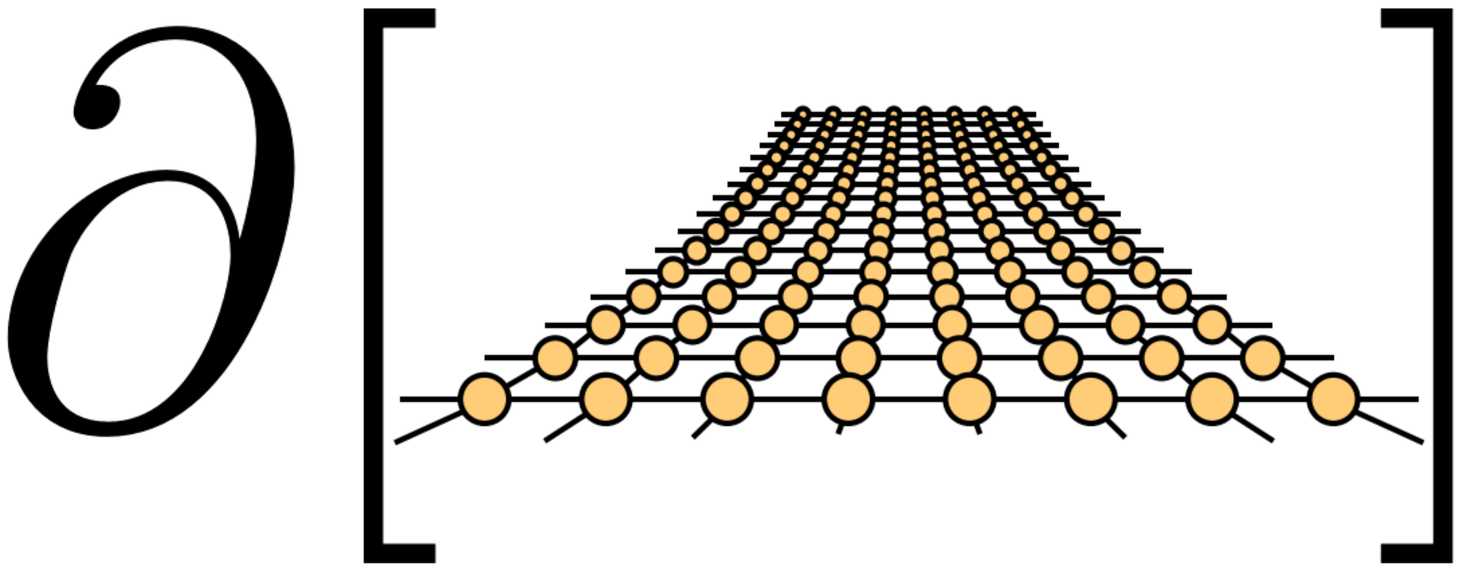
https://github.com/pytorch/pytorch/blob/master/tools/autograd/templates/Functions.cpp#L1700

1. What if we only solve the ground state (say using Power/Lanczos) ?

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

2. What about the case of degeneracy ?

Differentiable Programming Tensor Networks

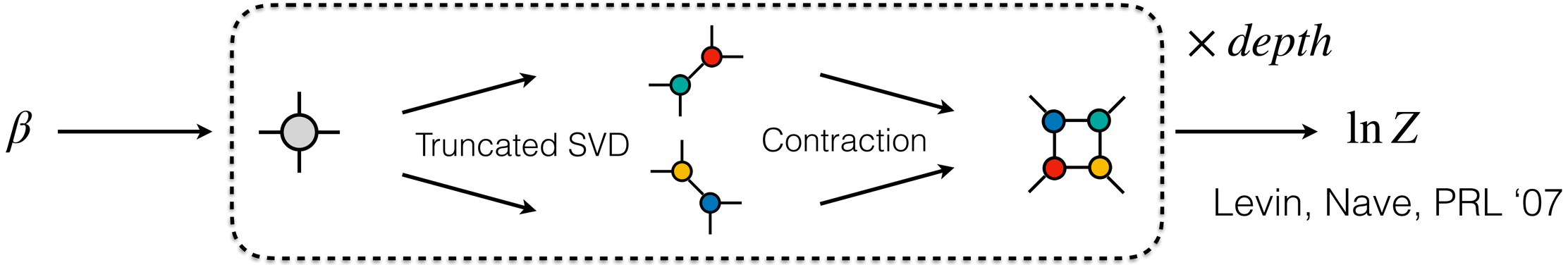


Liao, Liu, LW, Xiang, 1903.09650, PRX '19

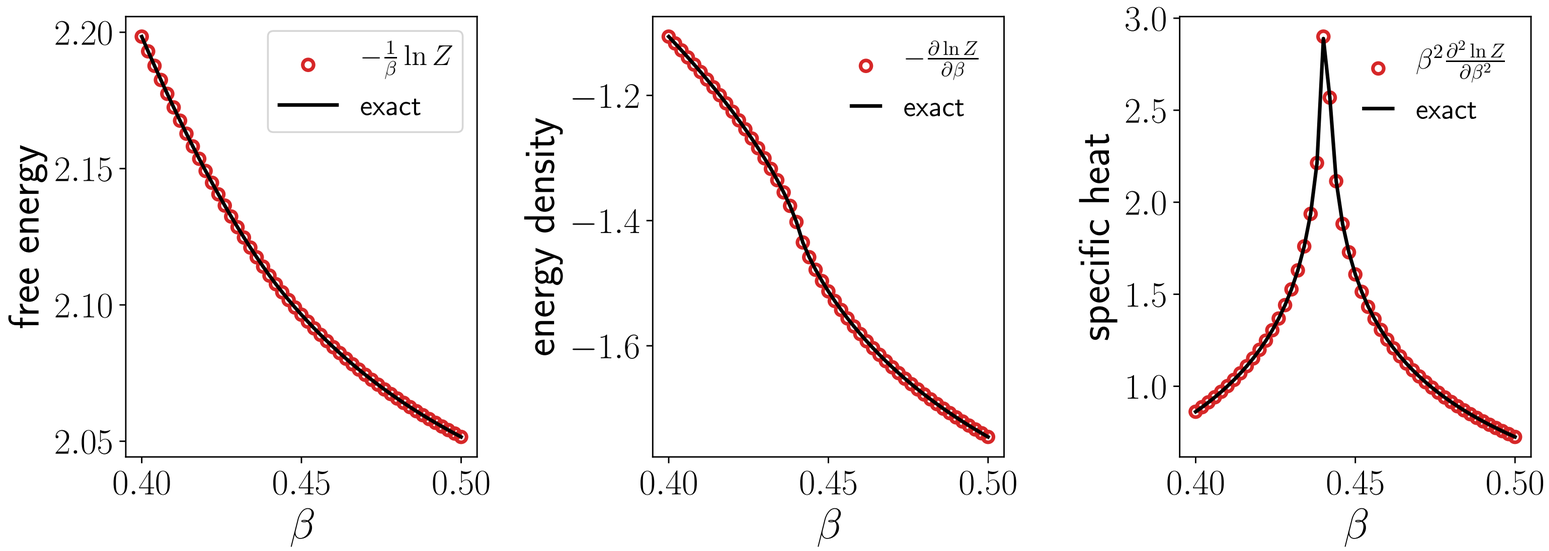
https://github.com/wangleiphy/tensorgrad

Differentiate through tensor RG for Ising

Computation graph

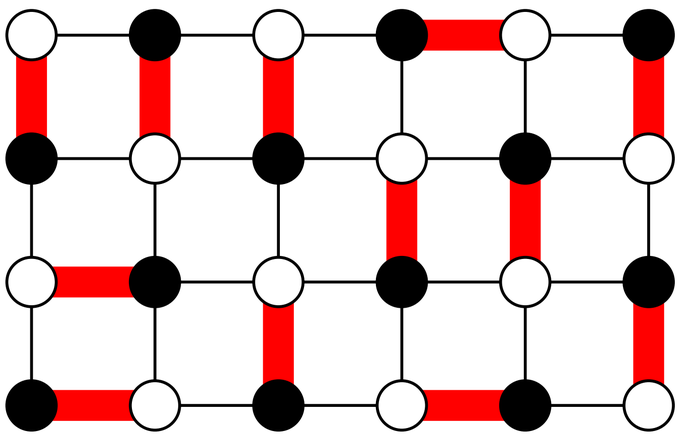


Levin, Nave, PRL '07



AD computes physical observables as high-order gradients

Demo: count # of dimer coverings



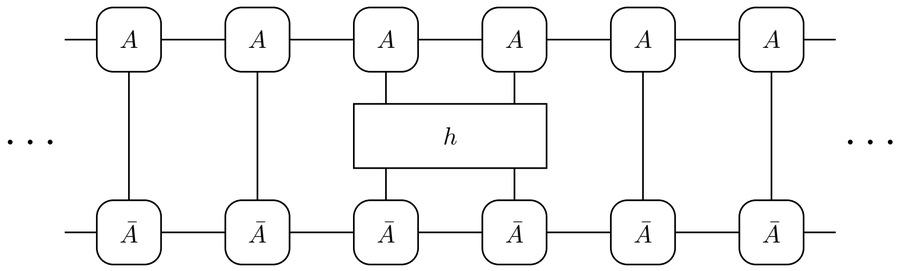
$$\lim_{n\to\infty} \frac{\ln(\#\text{ of coverings})}{n} ???$$

Kasteleyn, Fisher, Temperley, ‘61 Vanderstraeten, PRE, ‘18

Exercise

Automatic differentiation optimization of uniform MPS

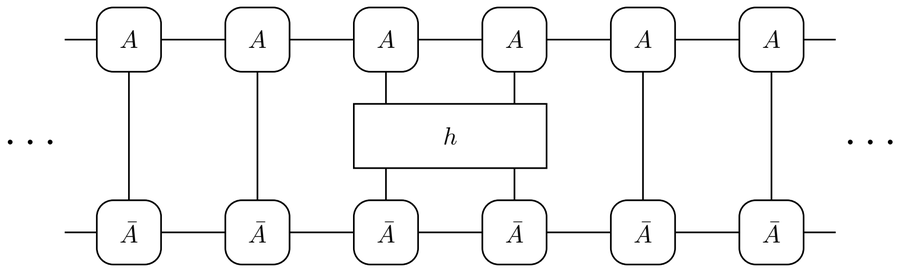
$$\min_{A}$$



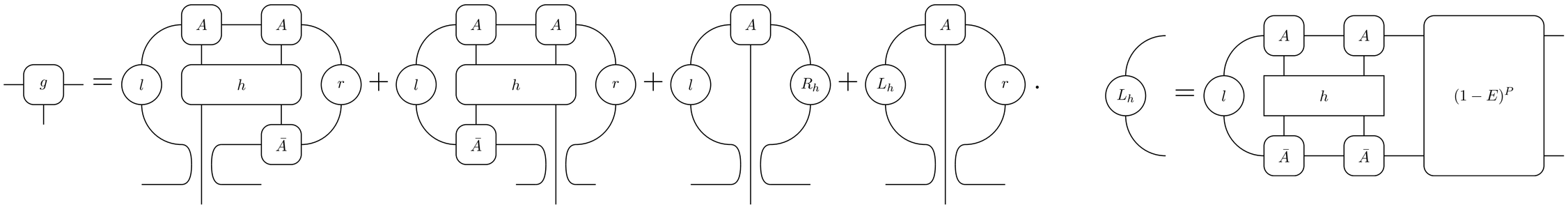
Exercise

Automatic differentiation optimization of uniform MPS

$$\min_{A}$$



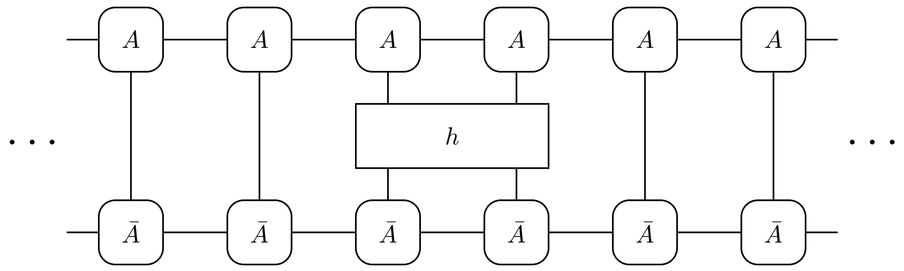
You will bypass these (VUMPS '18) …



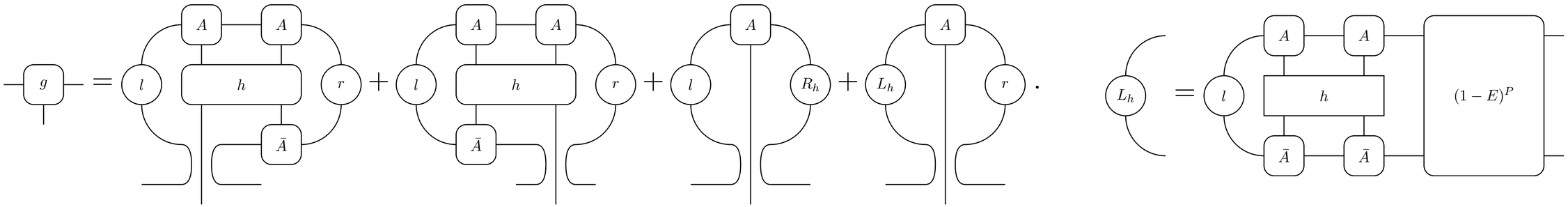
Exercise

Automatic differentiation optimization of uniform MPS

$$\min_{A}$$



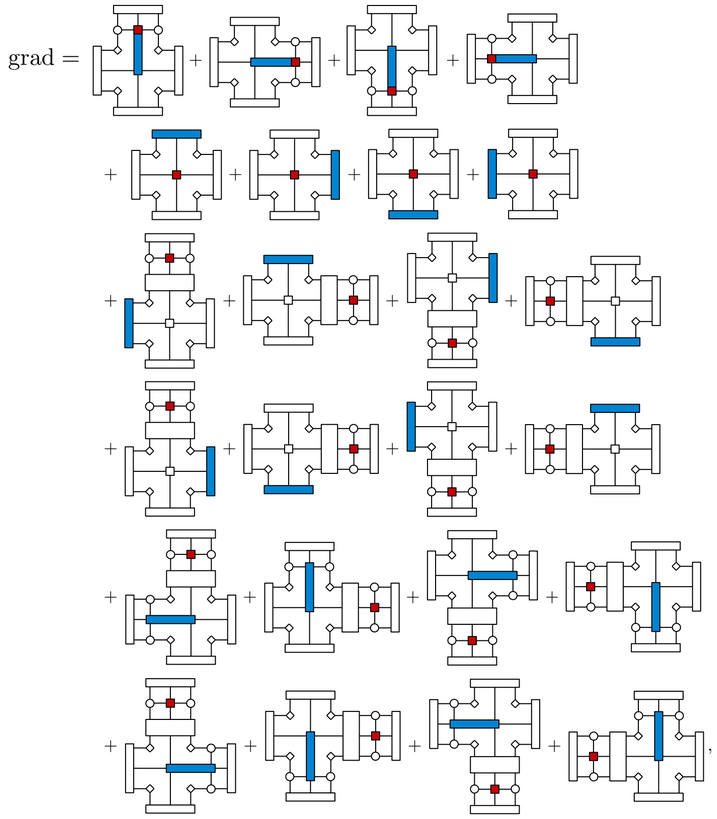
You will bypass these (VUMPS ’18) …



… but VUMPS would still be better, in this case

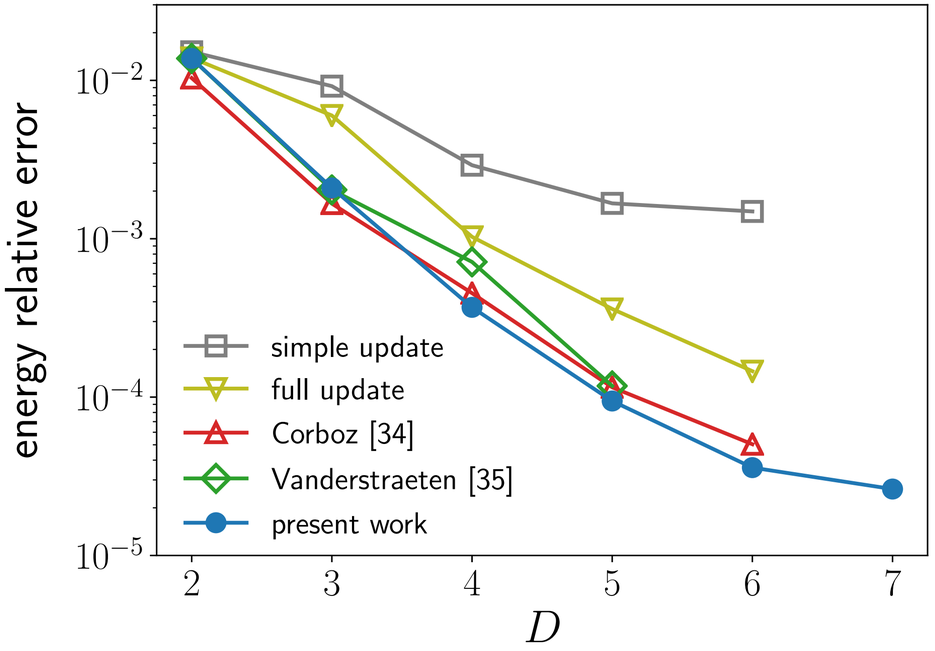
Quantum Heisenberg model in 2D

before…



now, w/ AD

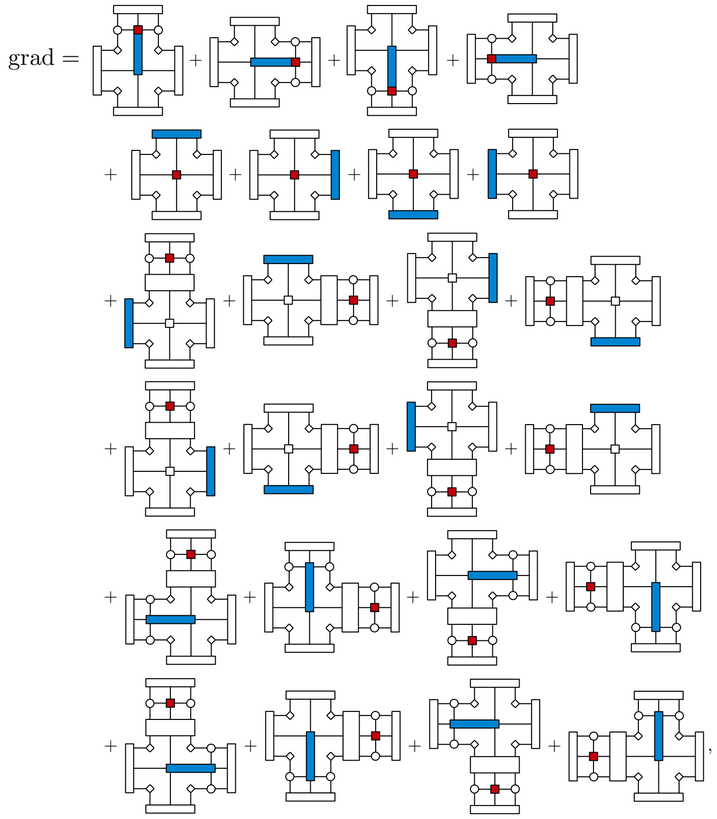
Liao, Liu, LW, Xiang, PRX '19



Best variational energy to date

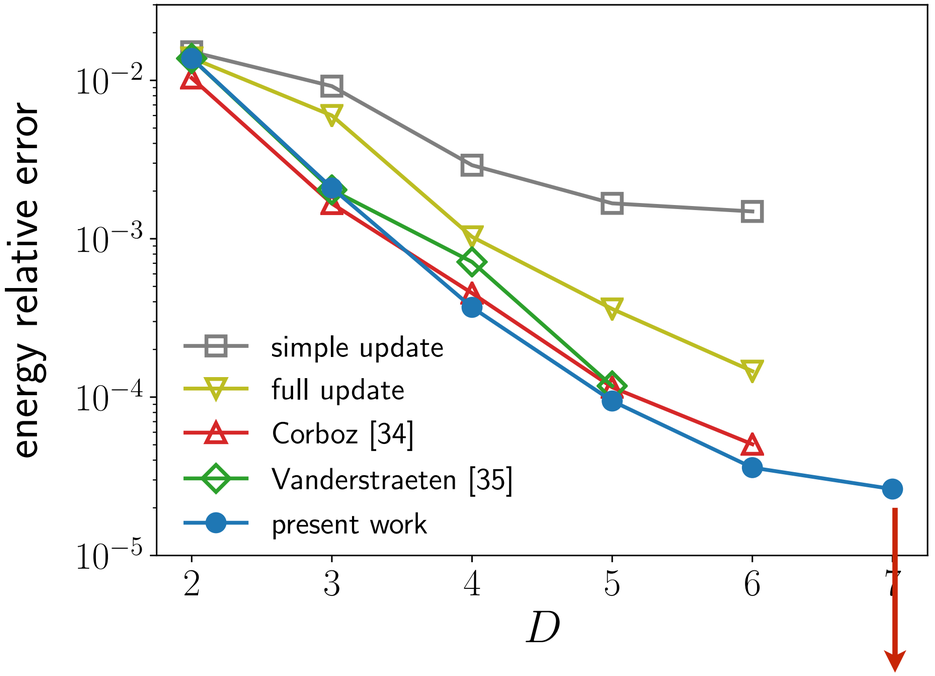
Quantum Heisenberg model in 2D

before…



now, w/ AD

Liao, Liu, LW, Xiang, PRX '19

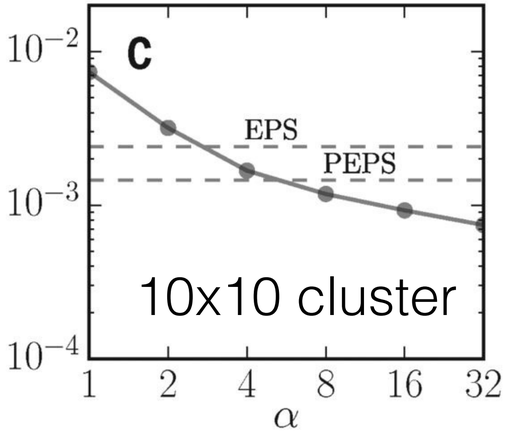


Best variational energy to date

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

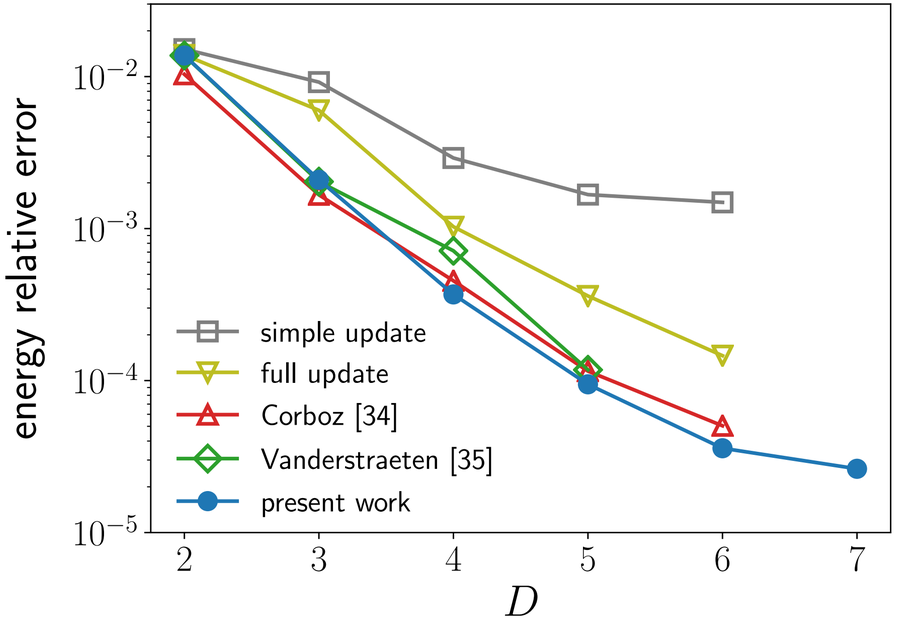
Quantum Heisenberg model in 2D

Finite size
Neural network



Carleo & Troyer, Science '17

Infinite size
Tensor network



Liao, Liu, LW, Xiang, PRX '19

This is leading to solutions of challenging physical problems: frustrated magnets, fermions, thermodynamics …

Nuts and Bolts

- Numerical stable backward through SVD

$$A = UDV^T \qquad \bar{A} \xleftarrow{?} \bar{U}, \bar{D}, \bar{V}$$

- Reduce memory via checkpointing or exploiting RG fixed point property

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

Liao, Liu, LW, Xiang, 1903.09650

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

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

Addition Multiplication Inverse Norm Determinant

Matrix polynomial Matrix exponential

Eigensovler Cholesky SVD QR

https://people.maths.ox.ac.uk/gilesm/files/NA-08-01.pdf
https://j-towns.github.io/papers/svd-derivative.pdf
Auto-Differentiating Linear Algebra, 1710.08717

Differentiable programming tools

HIPS/autograd

theano

flux

PyTorch

TensorFlow

Zygote

JAX

Keras

Zygote: A Differentiable Programming System to Bridge Machine Learning and Scientific Computing

Mike Innes
Julia Computing, Inc.

Alan Edelman
MIT

Keno Fischer
Julia Computing, Inc.

Chris Rackauckas
MIT

Elliot Saba
Julia Computing, Inc.

Viral B Shah
Julia Computing, Inc.

Will Tebbutt
Invenia Labs

Deep learning

Reinforcement learning

Computer vision

arXiv:1907.07587

Financial derivatives

Differential equations

Quantum circuits

Current support for AD*

*as of August 2019

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

Google summer of code

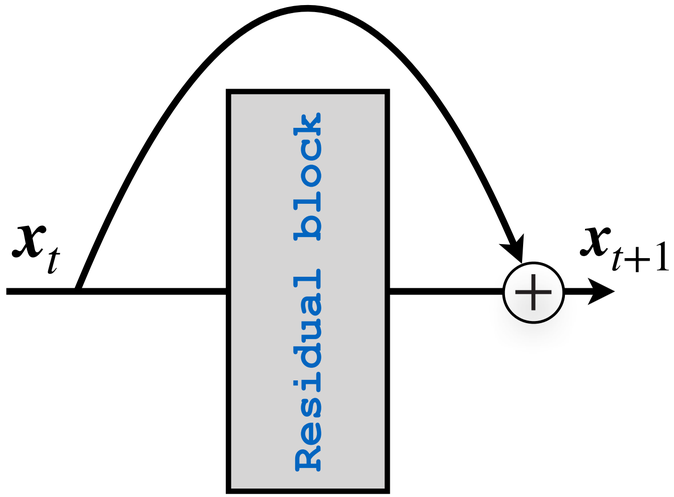
Andreas Peter mentored by Jin-Guo Liu

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

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

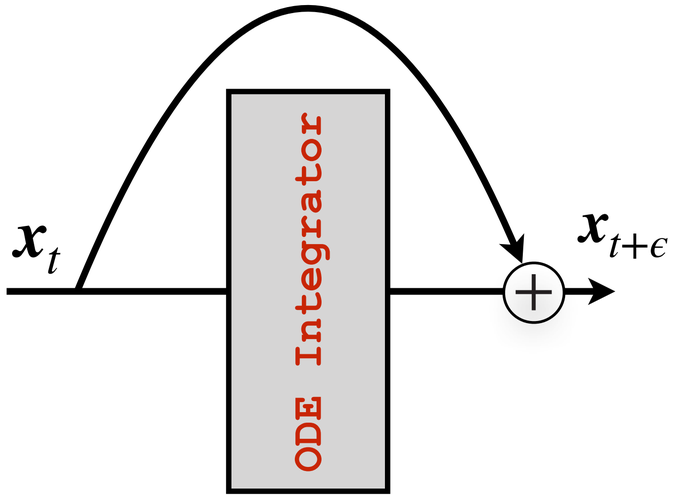
Neural Ordinary Differential Equations

Residual network



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

ODE integration



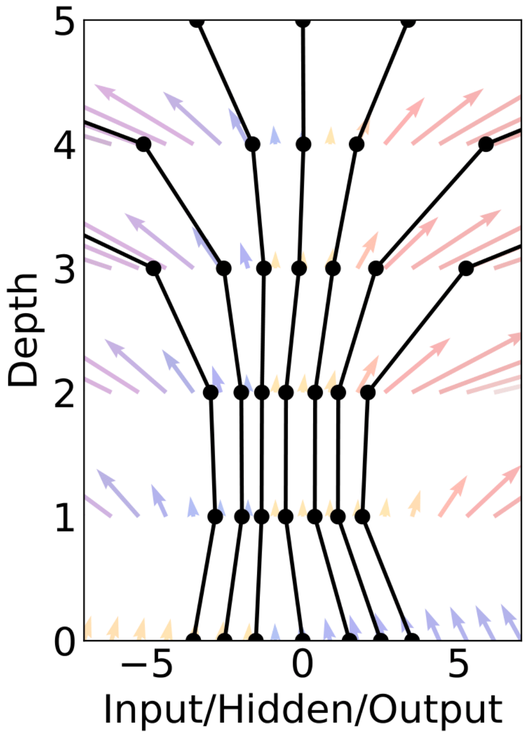
$$dx/dt = f(x)$$

Chen et al, 1806.07366 NIPS '18 Best paper award

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

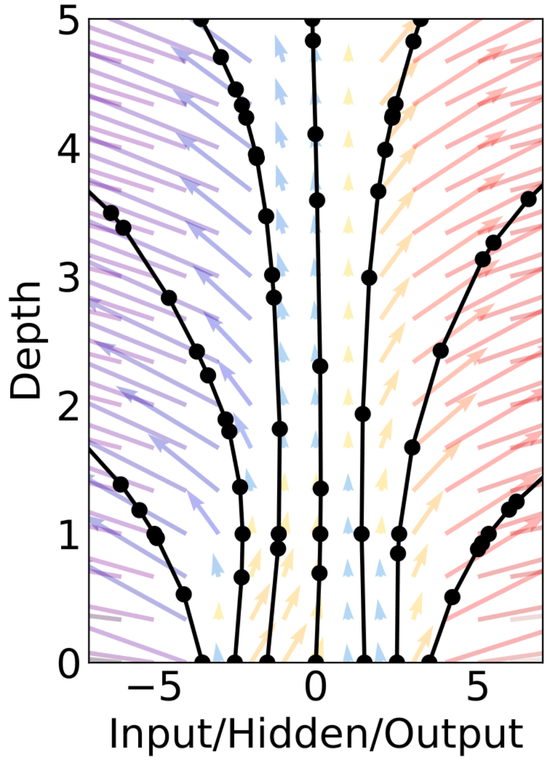
Neural Ordinary Differential Equations

Residual network



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

ODE integration



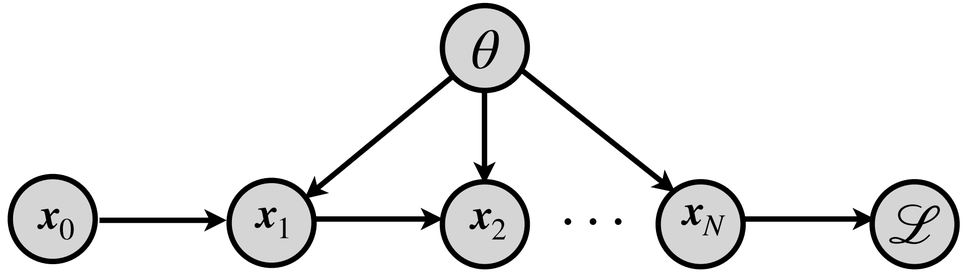
$$d\boldsymbol{x}/dt = f(\boldsymbol{x})$$

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

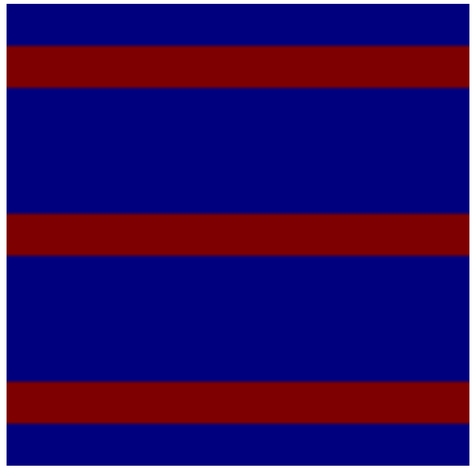
$$\frac{d\bar{\boldsymbol{x}}(t)}{dt} = -\bar{\boldsymbol{x}}(t)\frac{\partial f(\boldsymbol{x}, \boldsymbol{\theta}, t)}{\partial \boldsymbol{x}}$$

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!

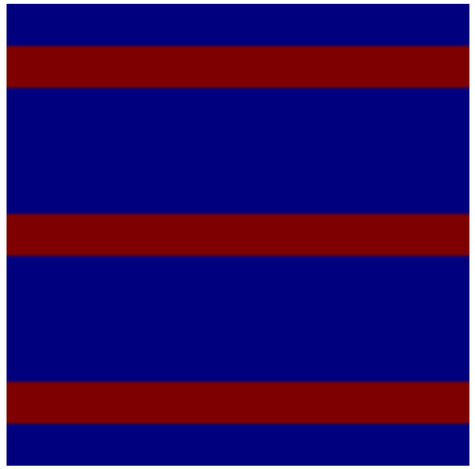
Why do we need Neural ODE ?



Backpropagating
through a fluid simulation

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

Why do we need Neural ODE ?

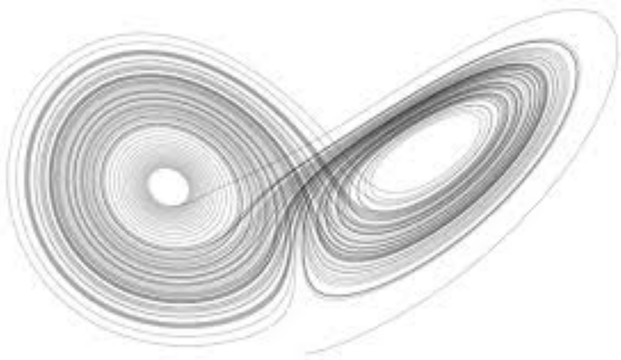


Backpropagating
through a fluid simulation

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

Applications

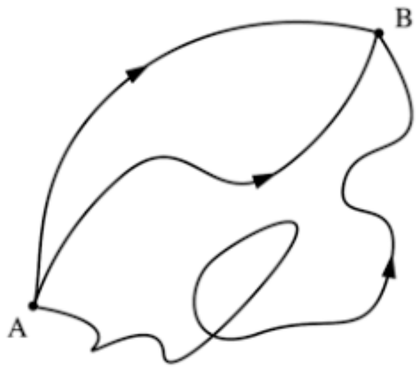
Dynamics systems



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

Classical and quantum control

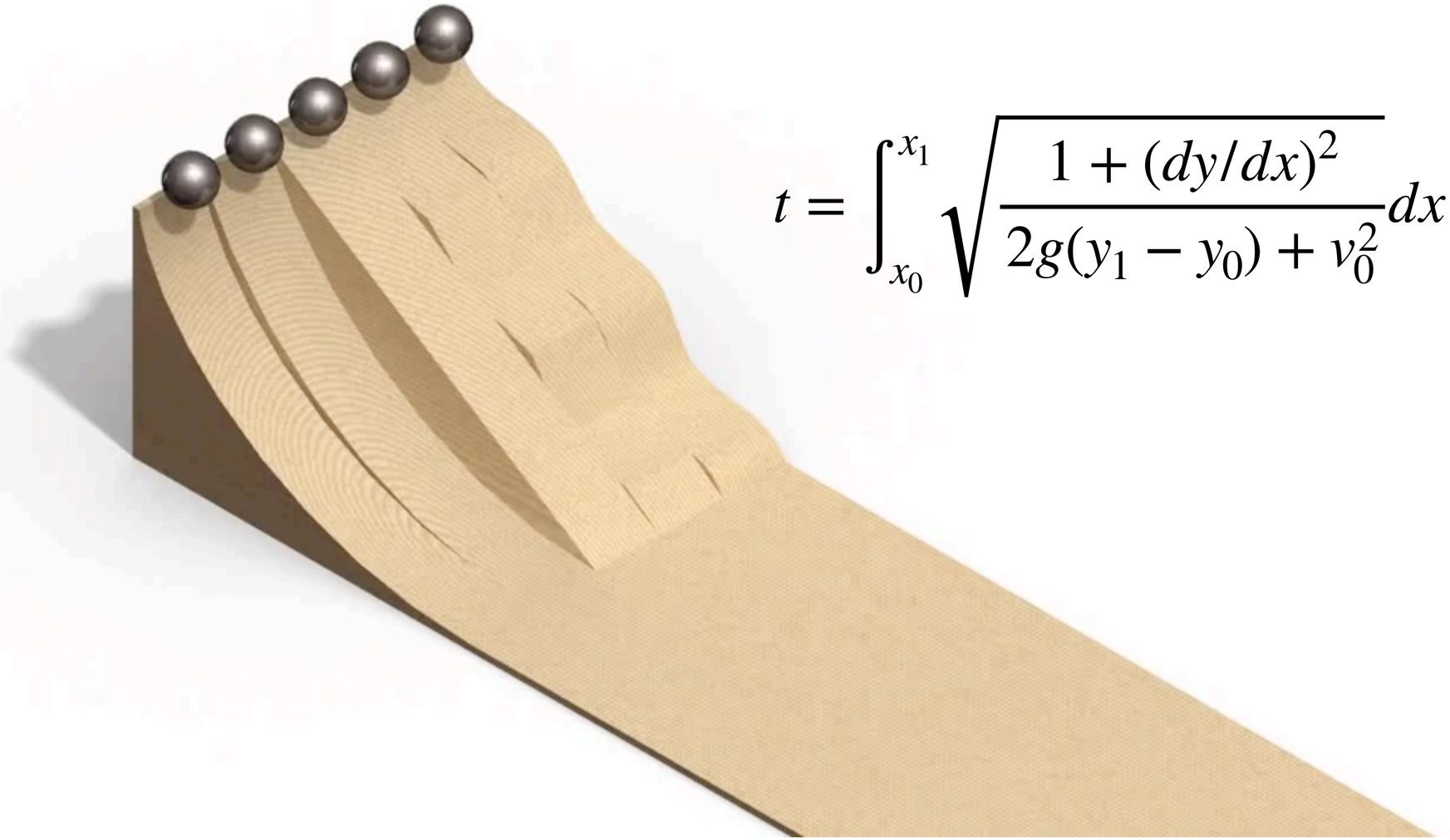
Principle of least actions



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

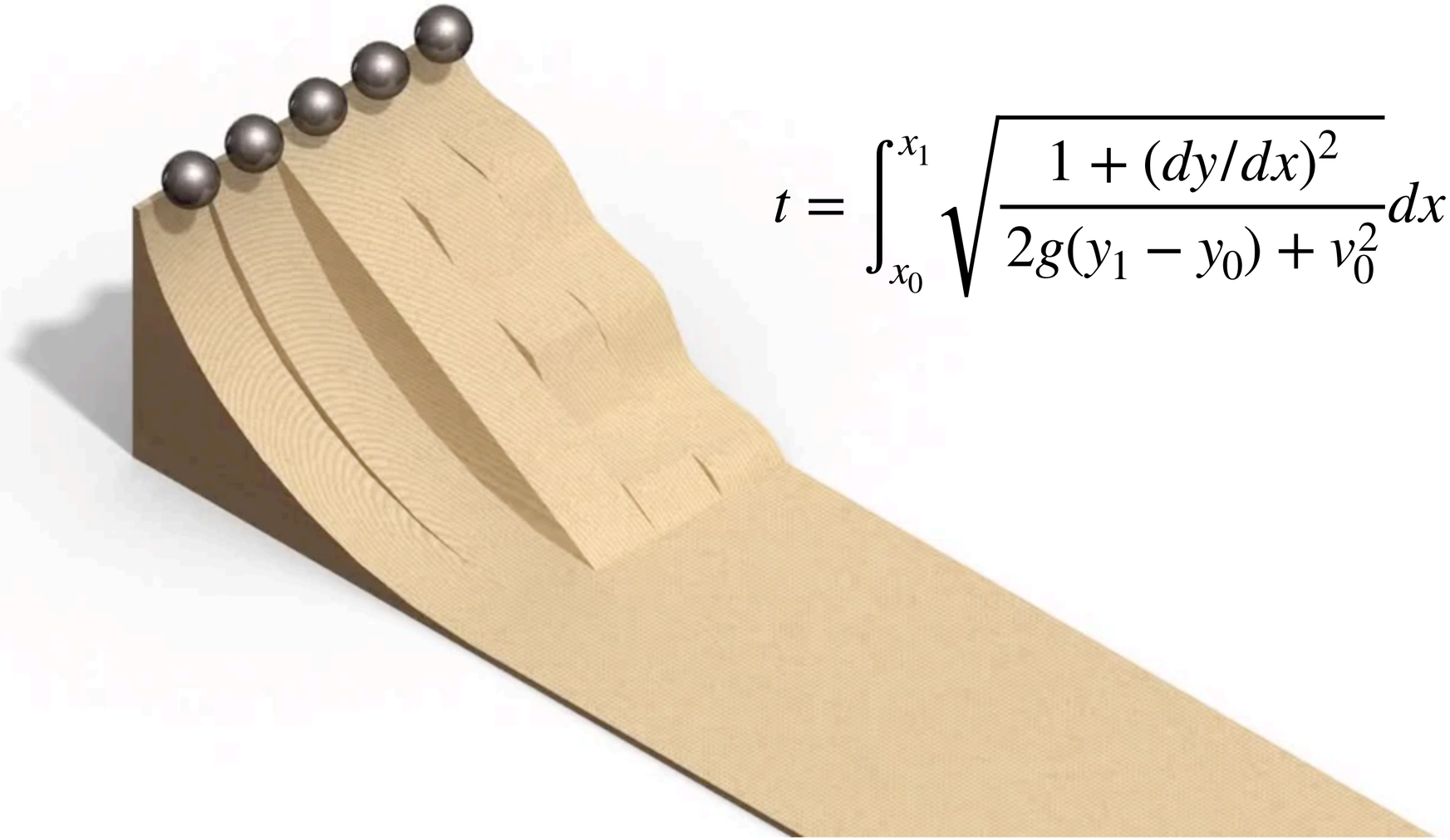
Optics, (quantum) mechanics, field theory…

Demo: Brachistochrone problem



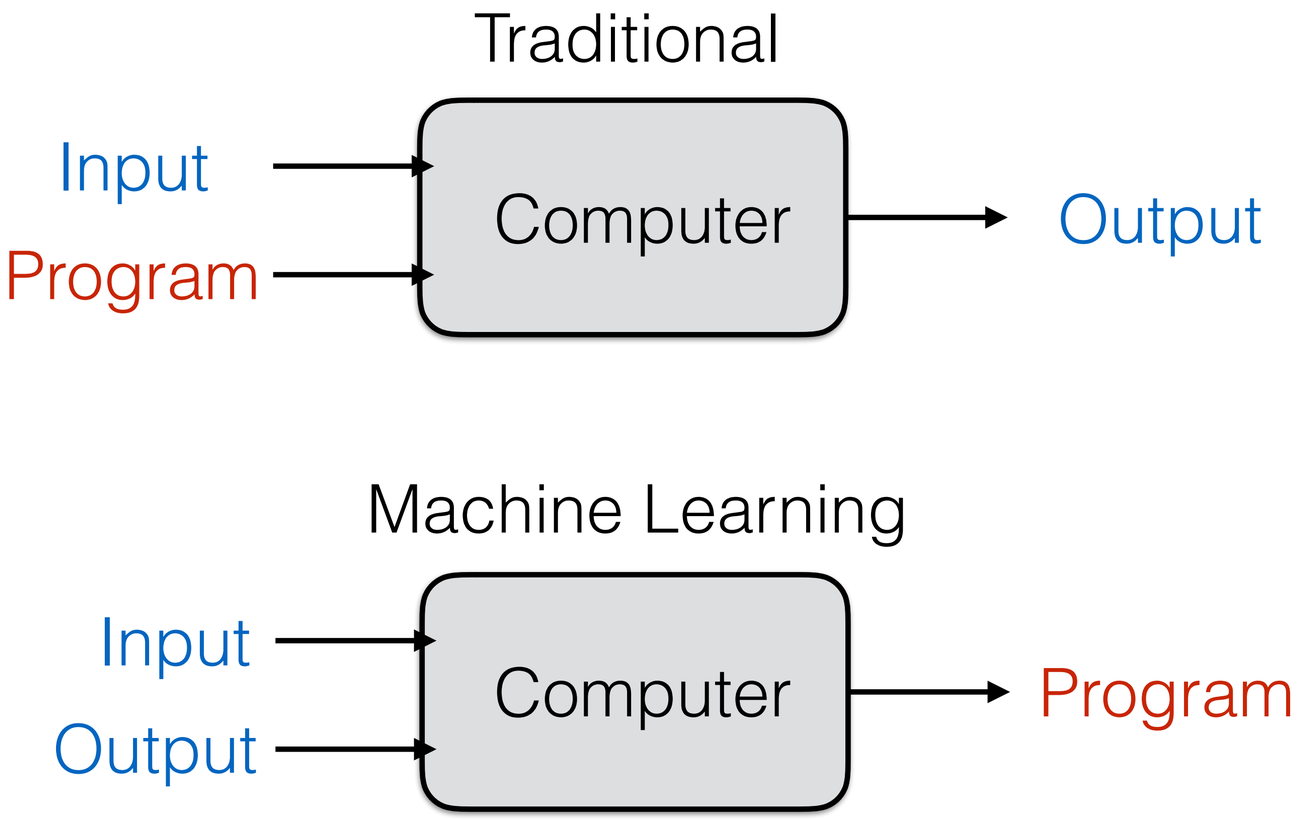
$$t = \int_{x_0}^{x_1} \sqrt{\frac{1 + (dy/dx)^2}{2g(y_1 - y_0) + v_0^2}}\, dx$$

Demo: Brachistochrone problem



$$t = \int_{x_0}^{x_1} \sqrt{\frac{1 + (dy/dx)^2}{2g(y_1 - y_0) + v_0^2}}\, dx$$

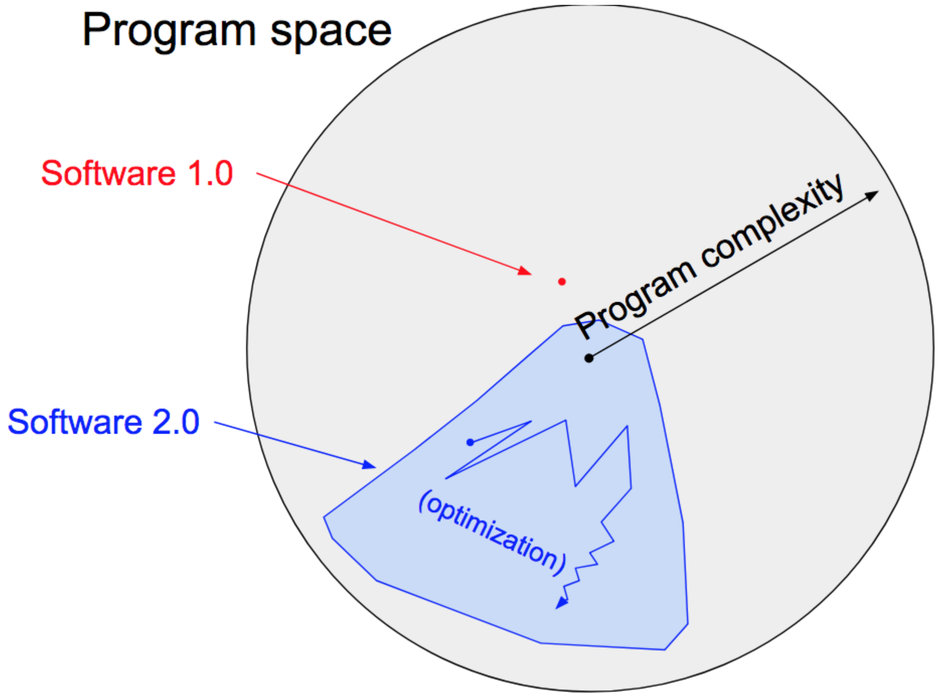
Differentiable Programming



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

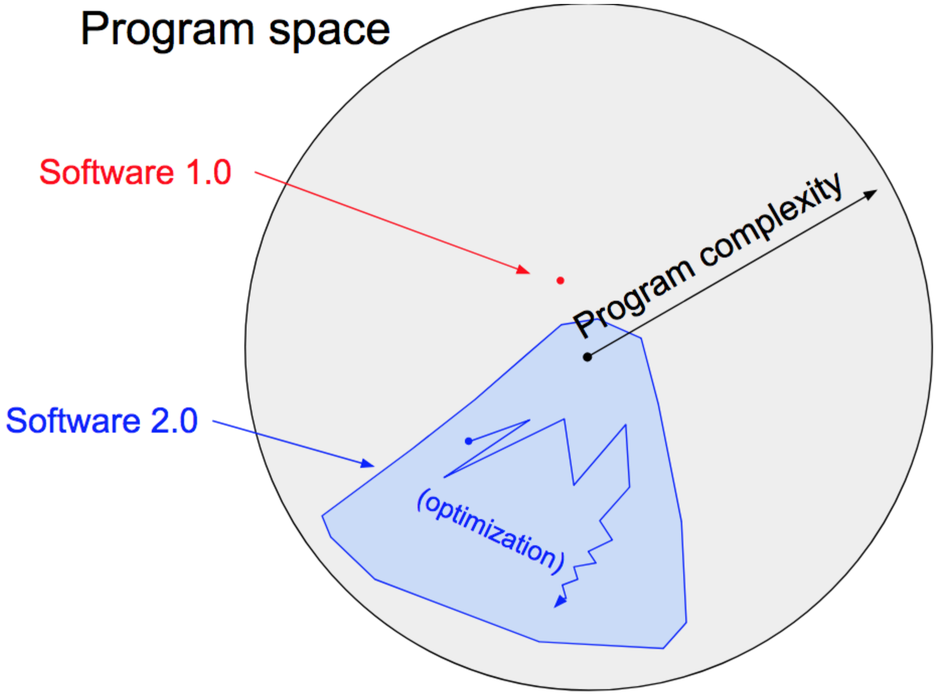
Benefits of Software 2.0

- 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

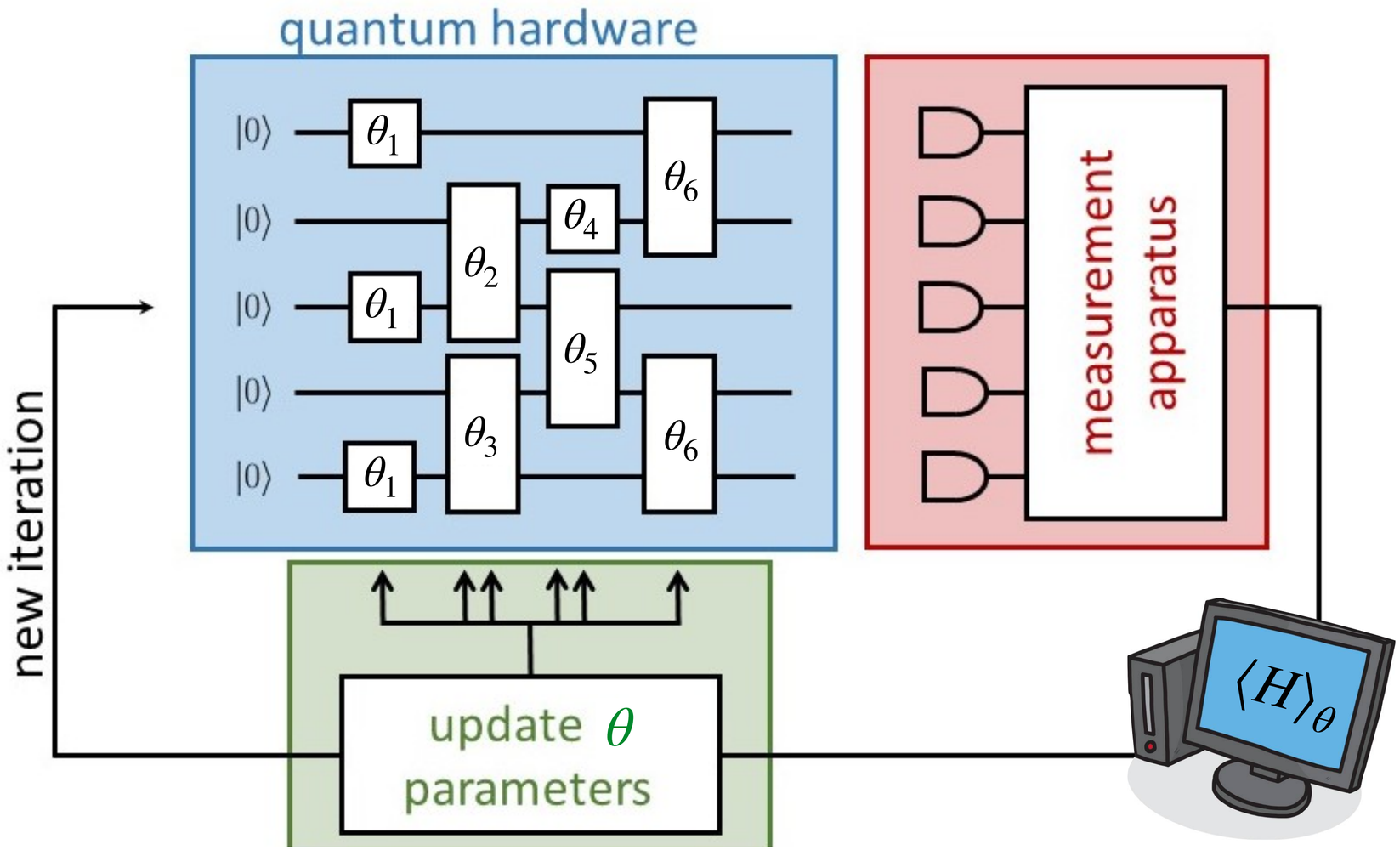


Writing software 2.0 by gradient search in the program space

Differentiable Programming Quantum Circuits

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

Variational quantum eigensolver

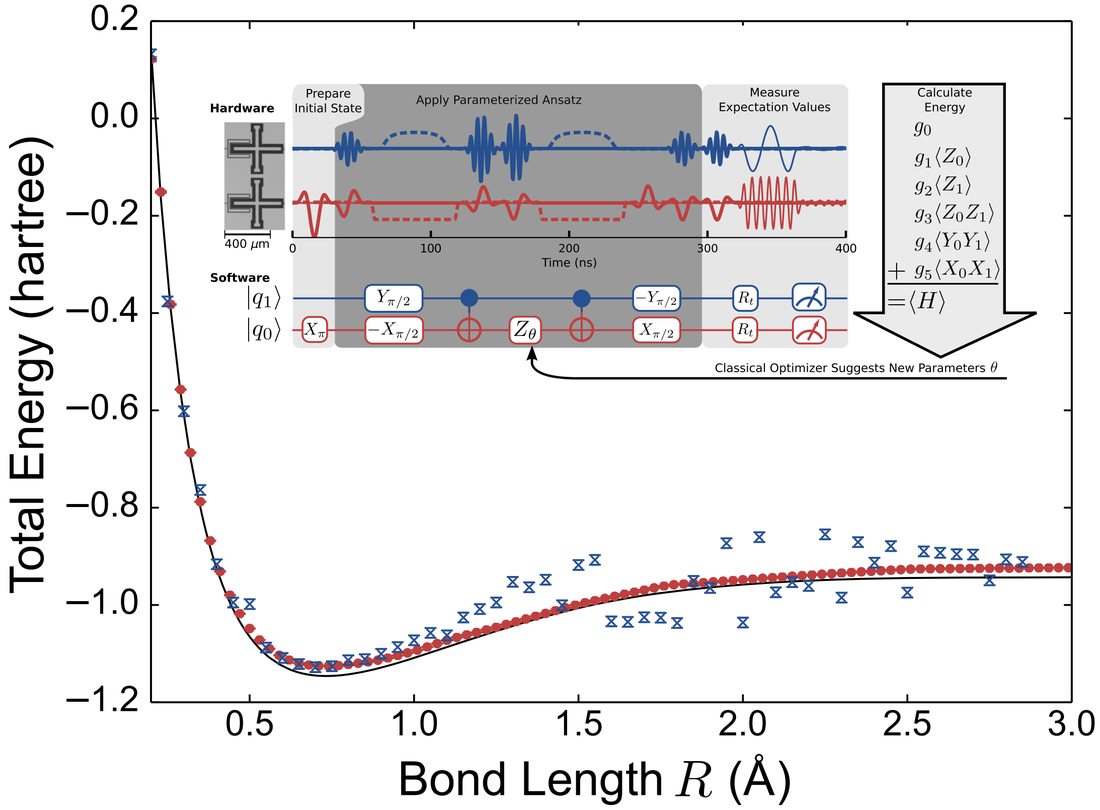


Quantum circuit as a variational ansatz

Peruzzo et al, Nat. Comm. ’13

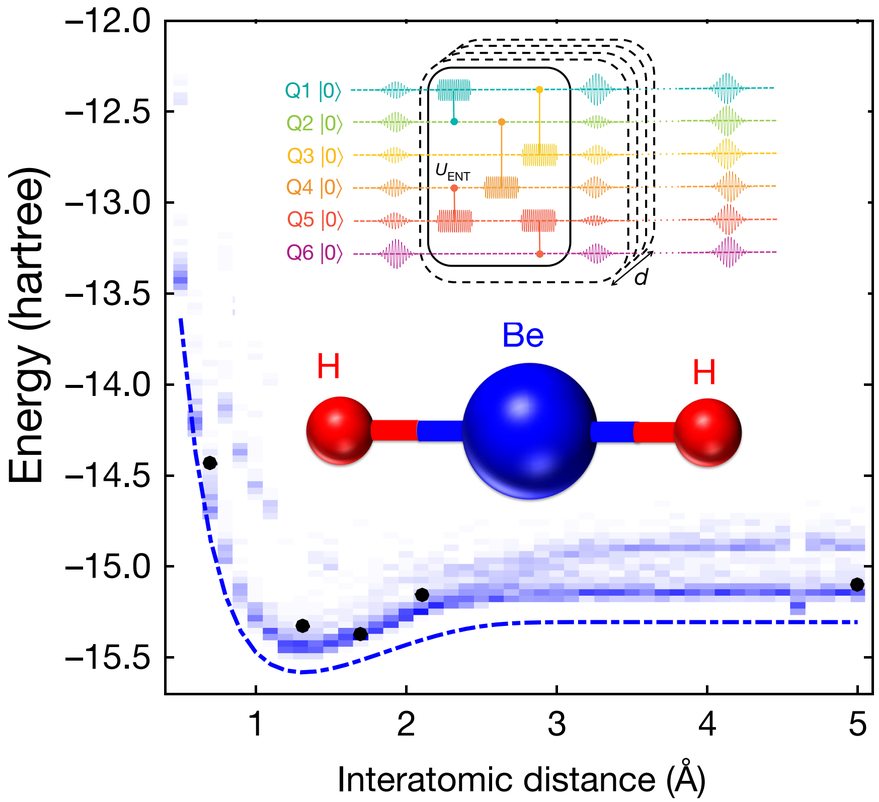
VQE on actual quantum devices

H_2 molecule with 2 qubits



Google PRX '16

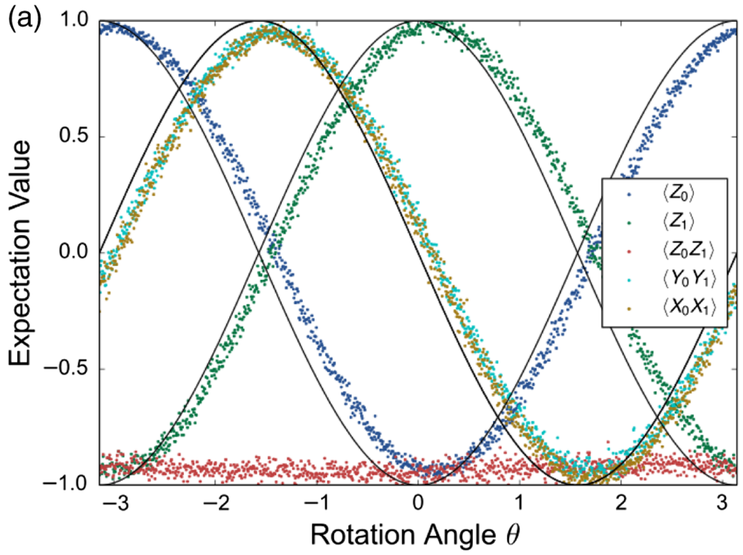
BeH_2 molecule with 6 qubits



IBM Nature '17

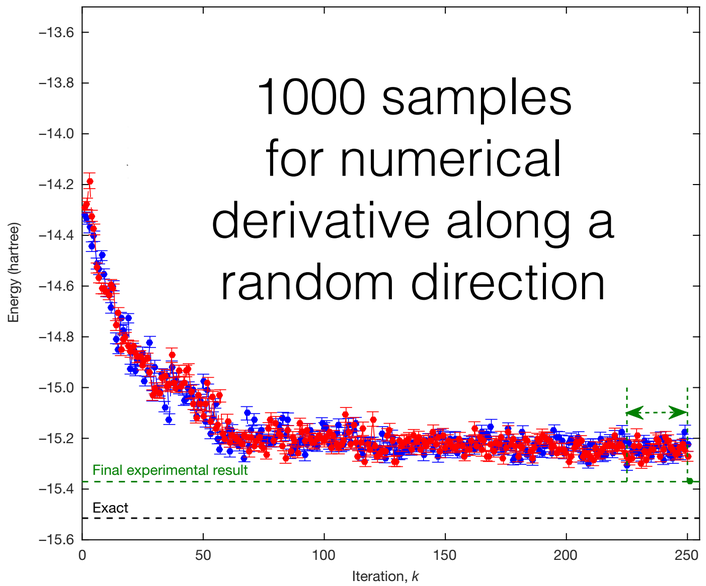
Optimize the quantum circuit

Scan 1000 values of the single variational parameter



Google PRX '16

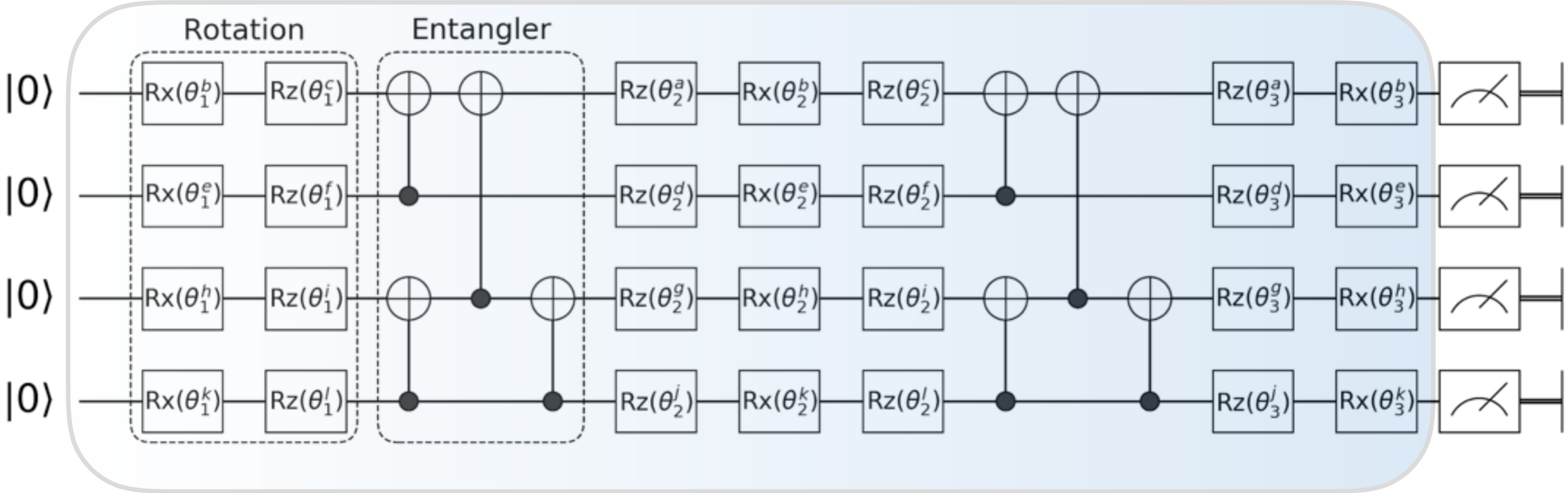
Stochastic gradient descend with numerical derivative



IBM Nature '17

These optimization schemes do not scale to higher dimensions

Differentiable quantum circuits



Parametrized gate of the form

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

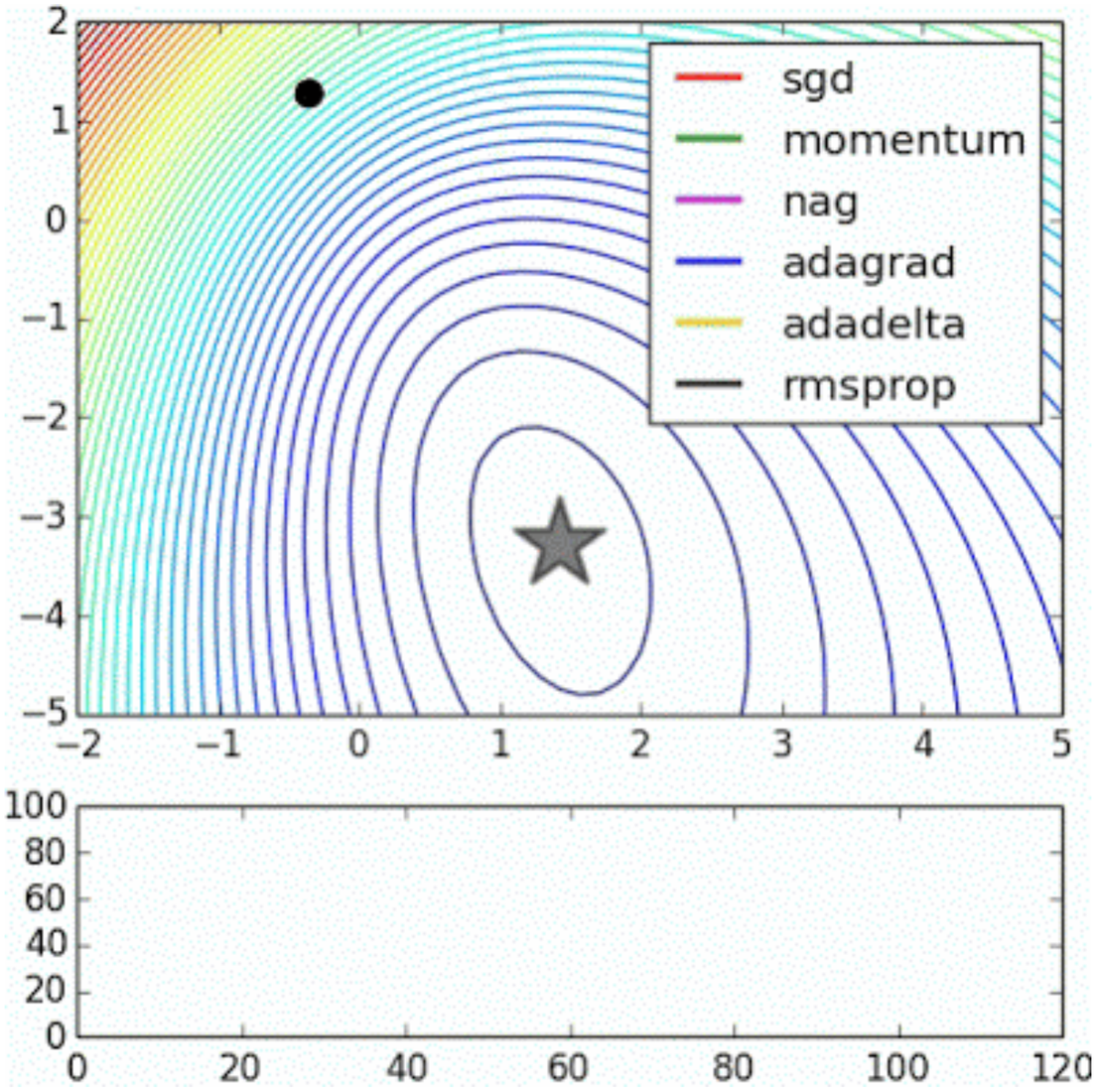
eg, X, Y, Z, CNOT, SWAP…

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

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

Unbiased gradient estimator measured on quantum circuits

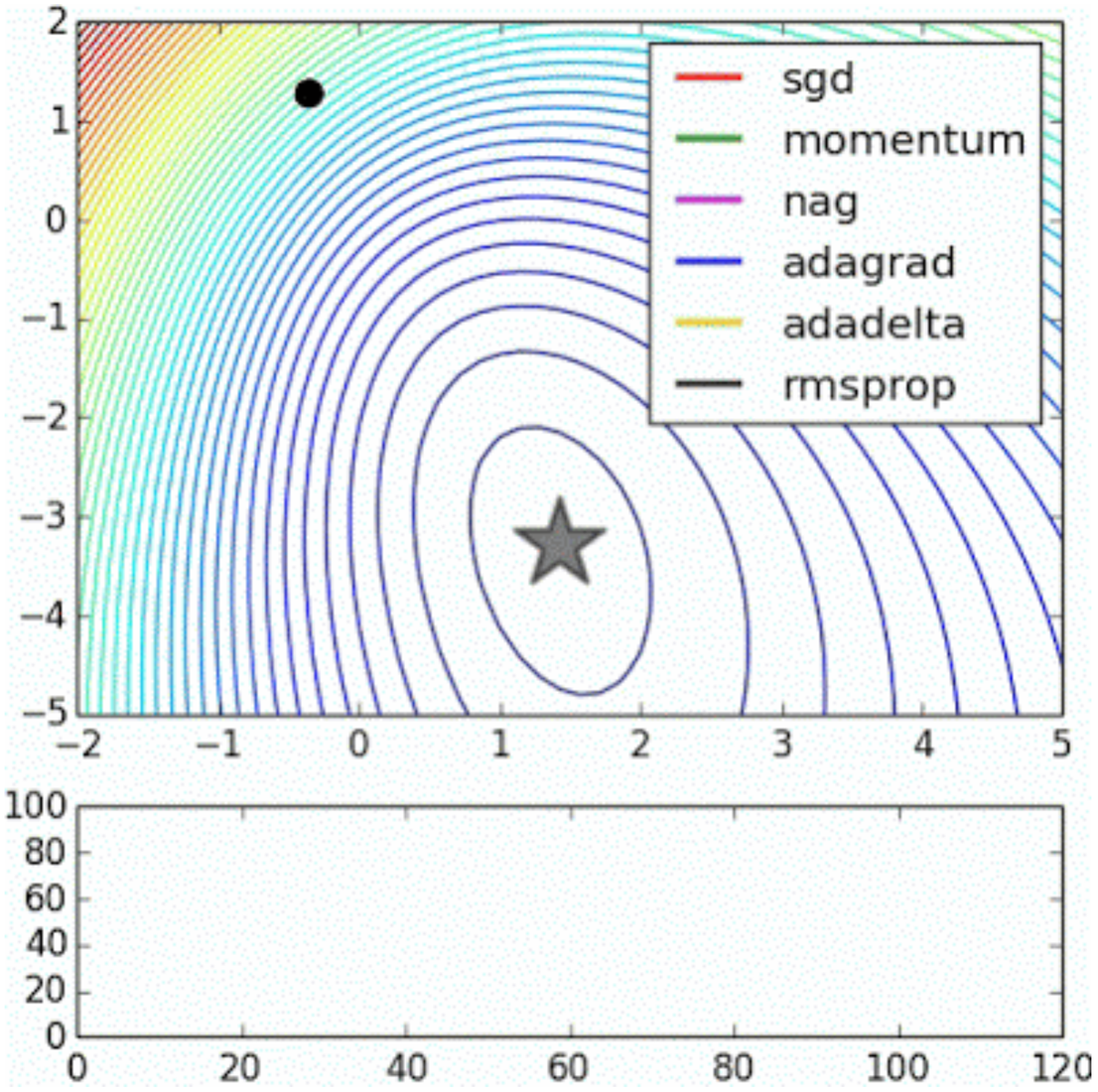
Optimization with noisy gradients



Ruder, 1609.04747

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

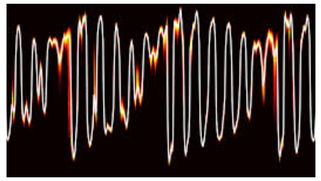
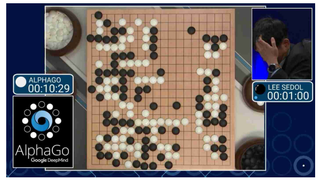
Optimization with noisy gradients

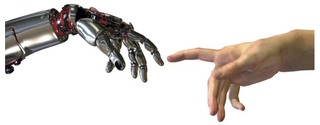


Ruder, 1609.04747

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

What is the secret behind deep learning ?

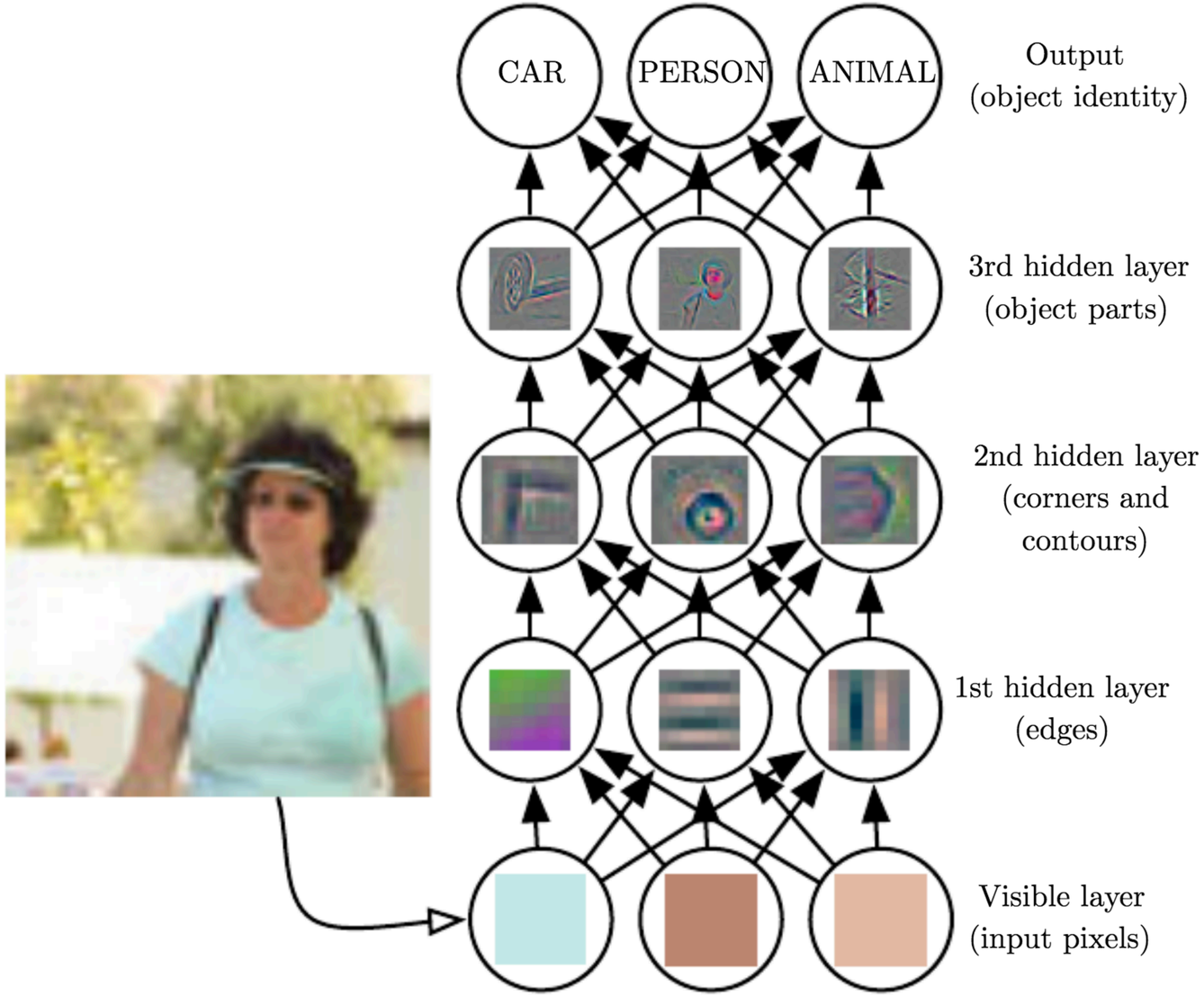





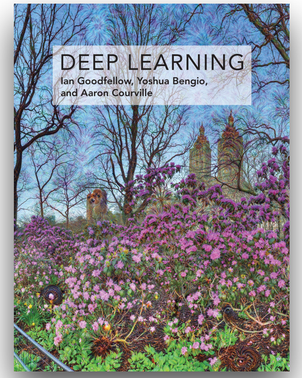
Key technology: differentiable programing

Core idea: learning representation

Learning Representation



Page 6
Figure 1.2



Goodfellow, Bengio, Courville, http://www.deeplearningbook.org/

Magic of learning representation

Neural style transfer

Gatys et al, 1508.06576

Latent space interpolation

Glow 1807.03039

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

Magic of learning representation

Neural style transfer

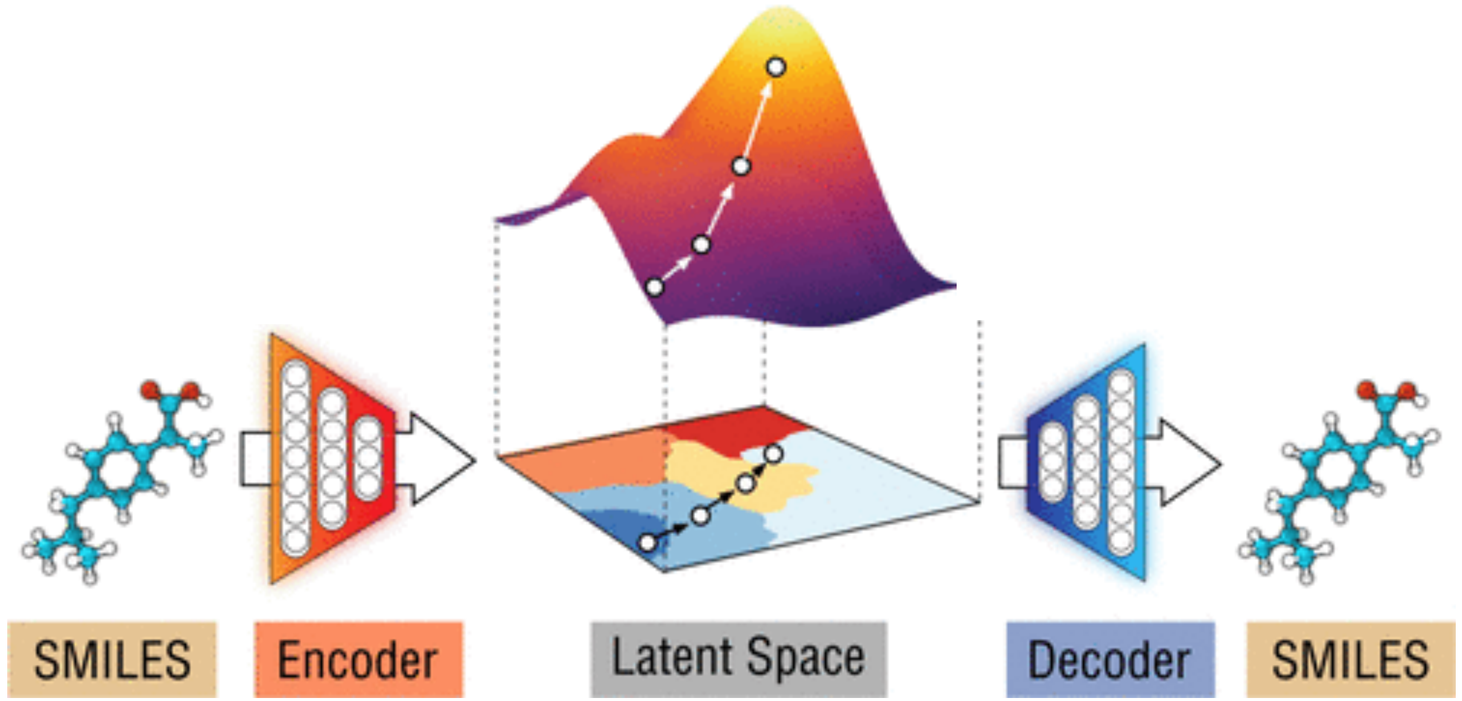
Gatys et al, 1508.06576

Latent space interpolation

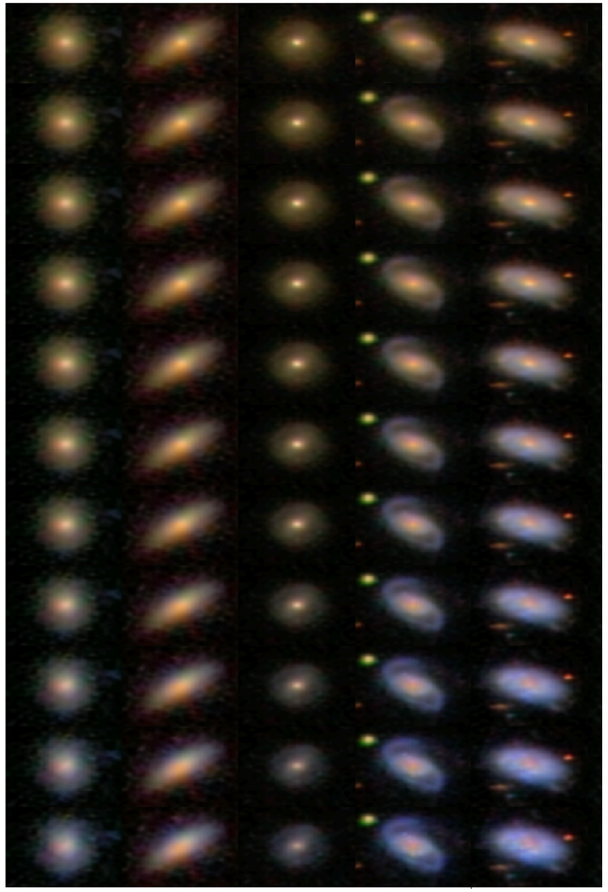
Glow 1807.03039

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

Learning representation for science

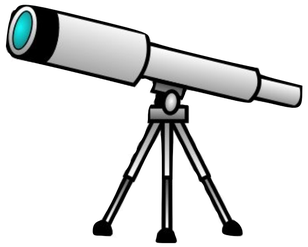


Automatic chemical design
Gomez-Bombarelli et al,1610.02415



Galaxy evolution
Schawinski et al, 1812.01114

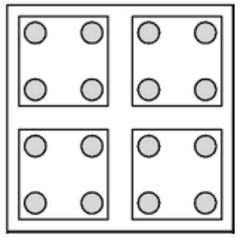
Learning representation and CMP



Renormalization group

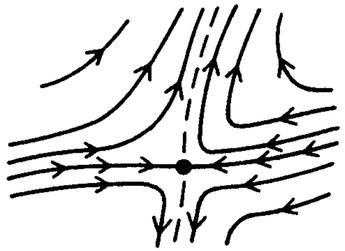
Monte Carlo update

Deep Learning and Renormalization Group

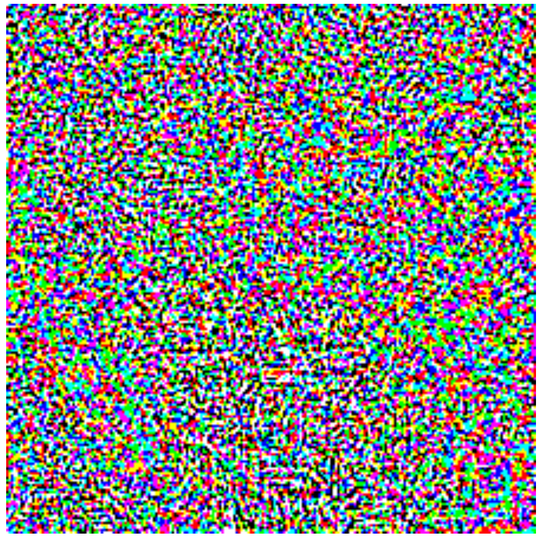


't Hooft, Gross, Wilczek, Kadanoff, Wilson, Fisher…

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



$+ .007 \times$



$=$

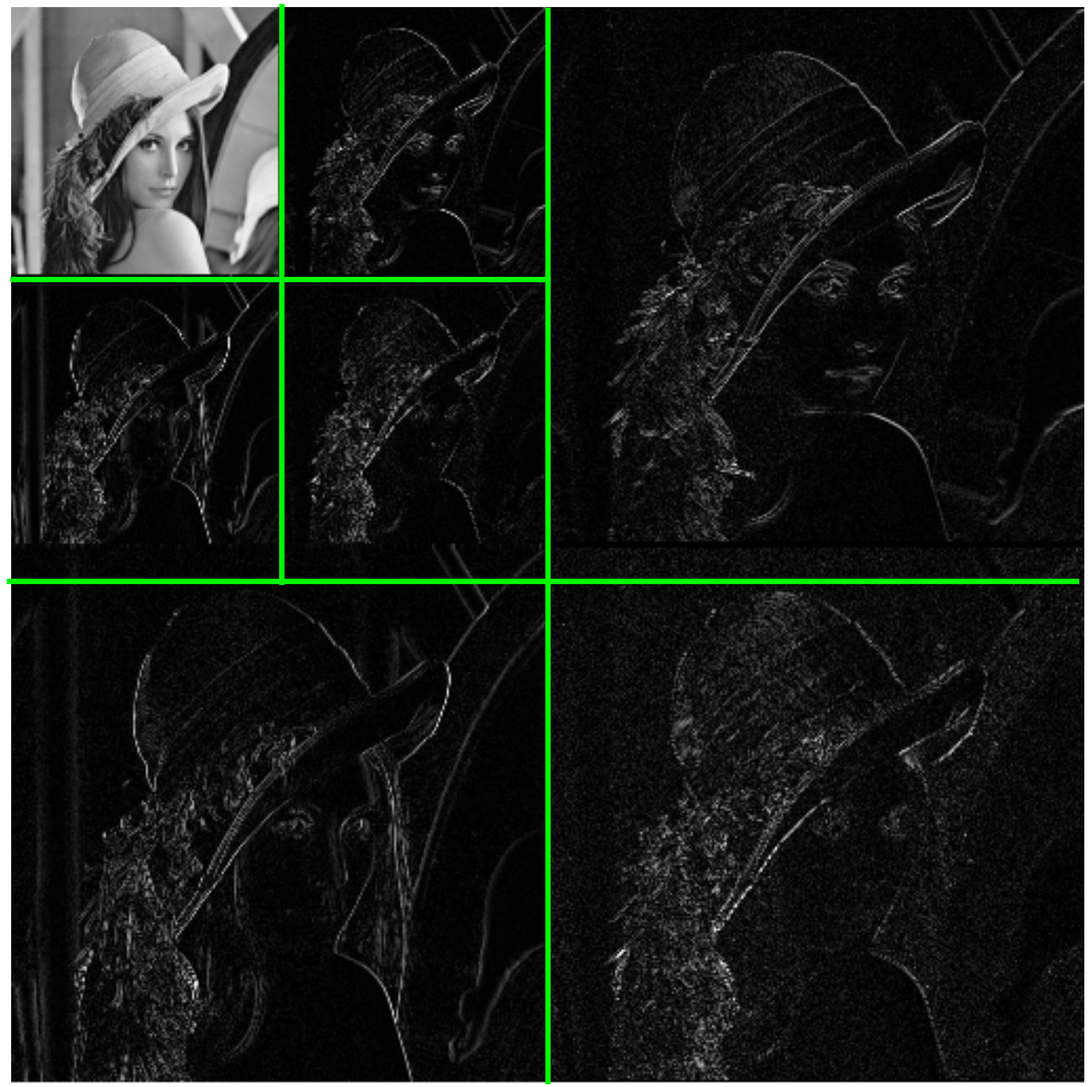
Panda
Confidence 58%

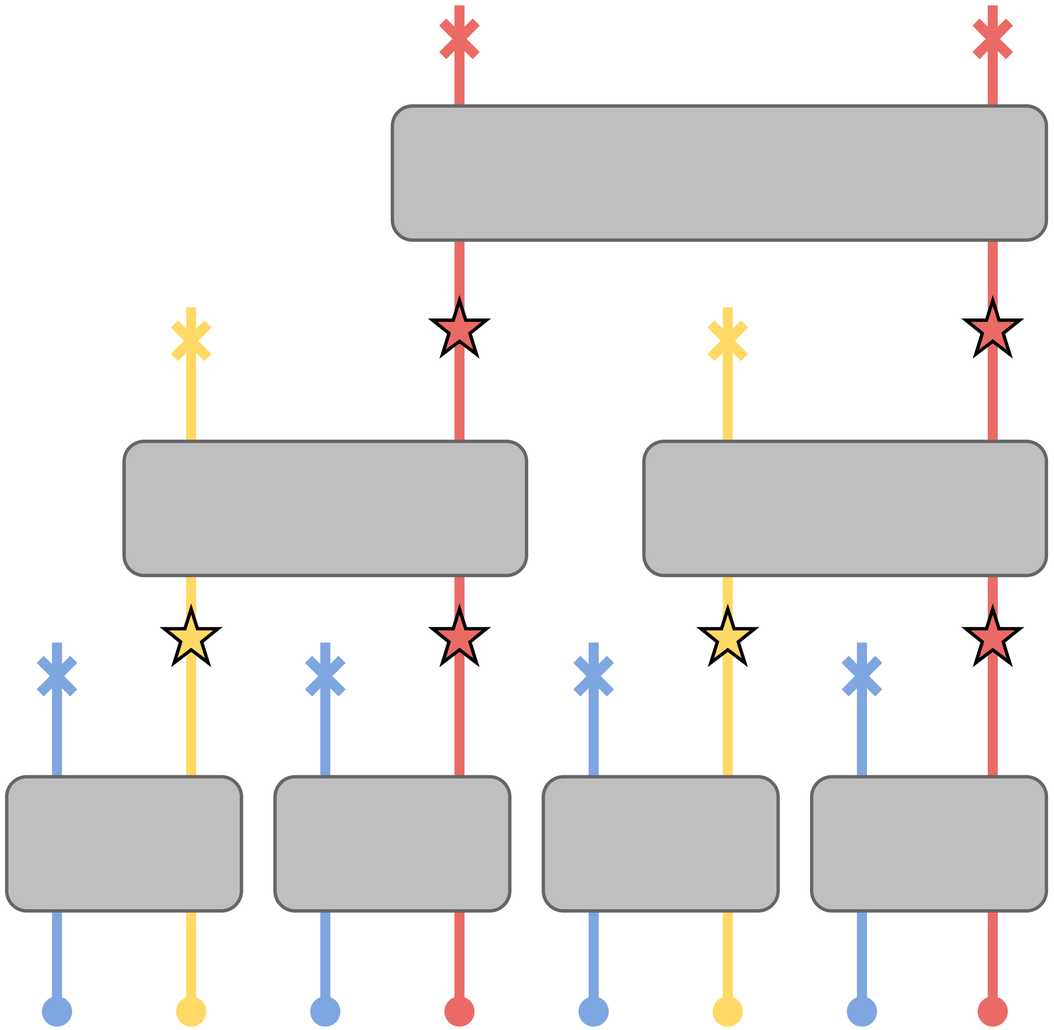
Goodfellow et al, 2014

Gibbon
Confidence 99%

Vulnerability of deep learning, Kenway, 1803.06111 & 1803.10995

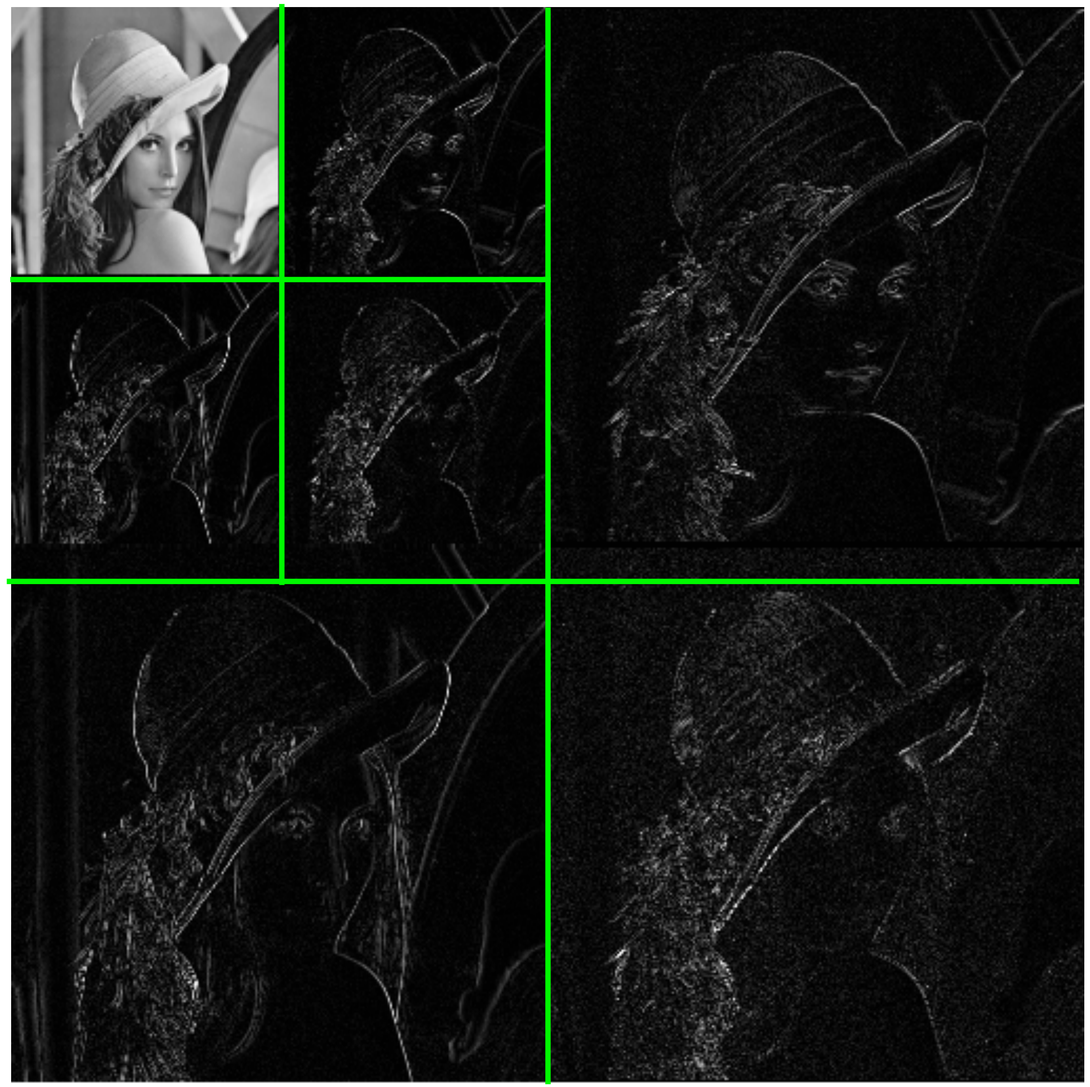
Wavelet transformation for Lena and Ising

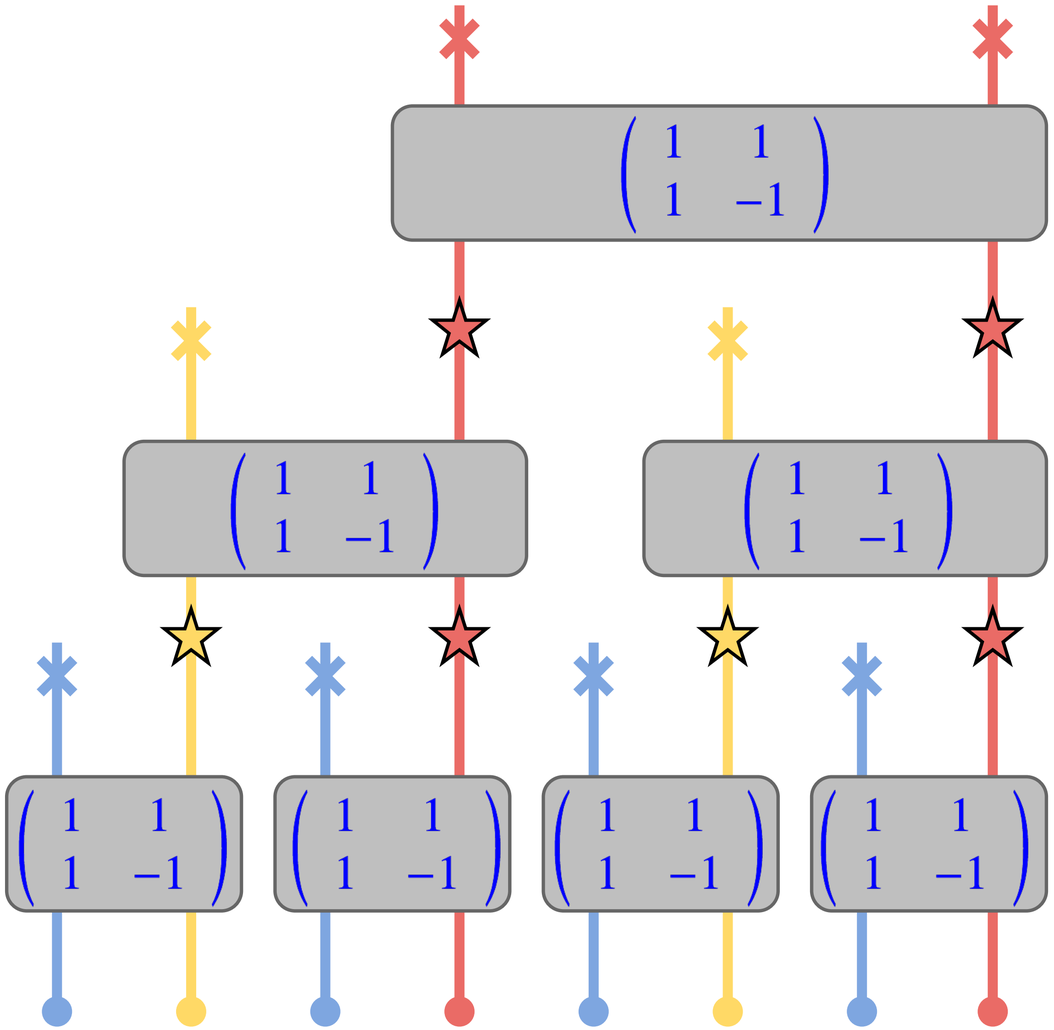




Guy, Wavelets & RG1999+ White, Evenbly, Qi, Wavelets, MERA, and holographic mapping 2013+

Wavelet transformation for Lena and Ising

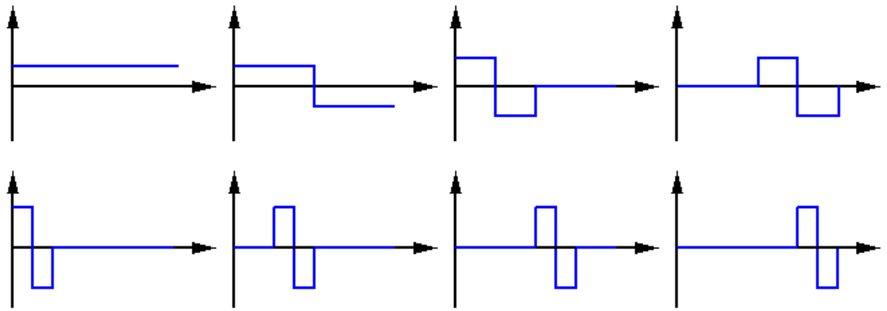


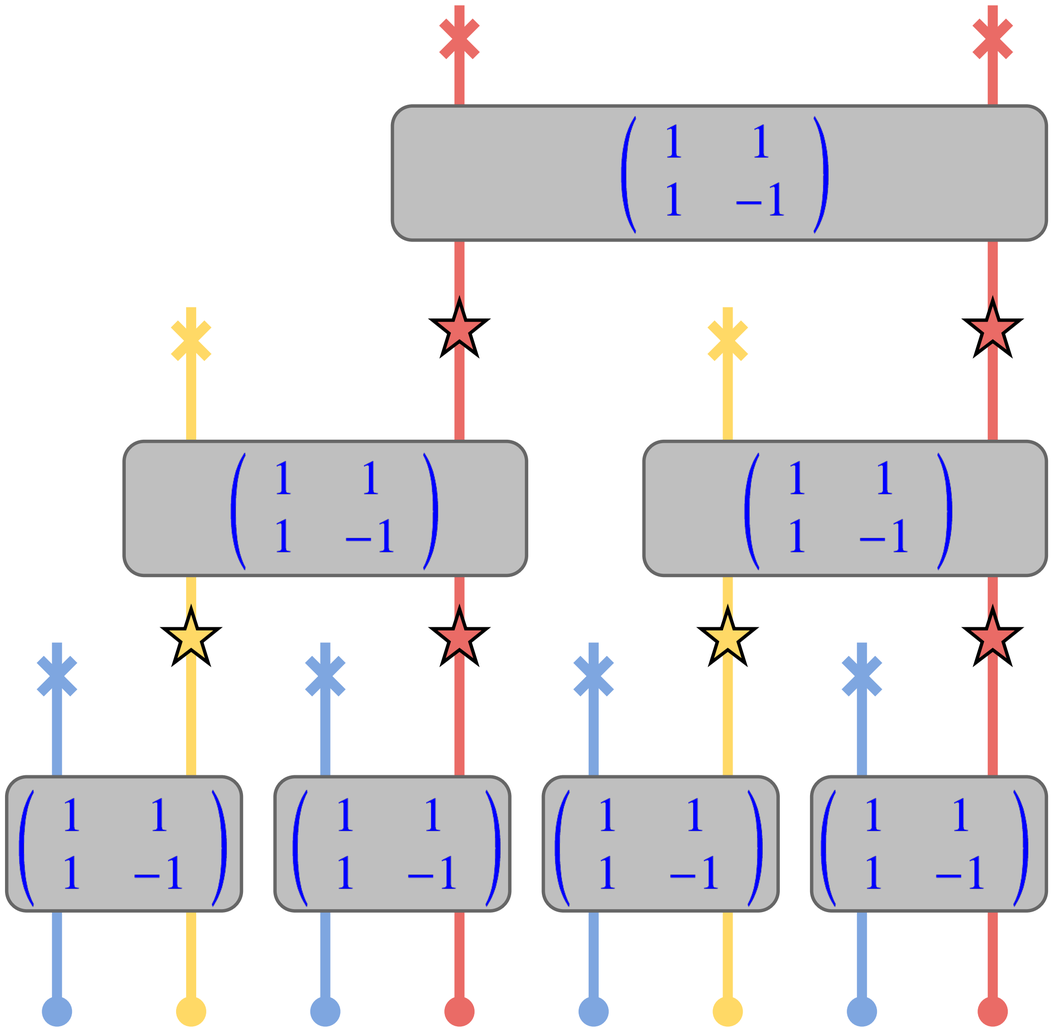


Guy, Wavelets & RG1999+ White, Evenbly, Qi, Wavelets, MERA, and holographic mapping 2013+

Wavelet transformation for Lena and Ising

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



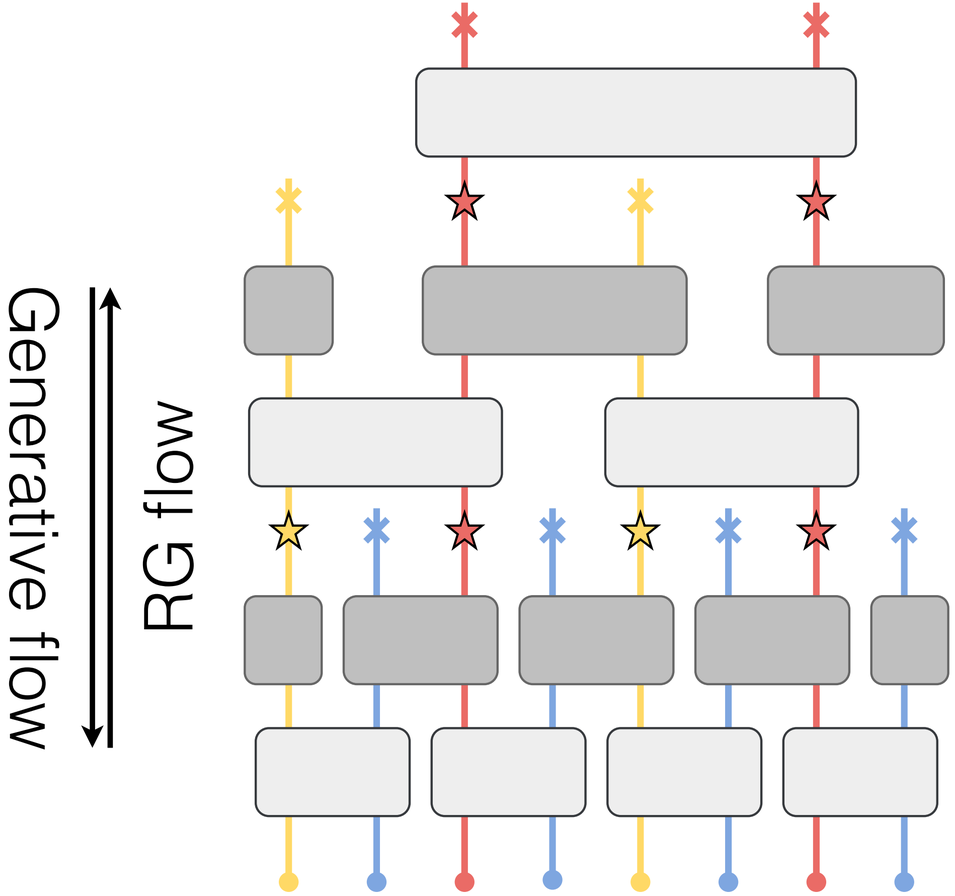


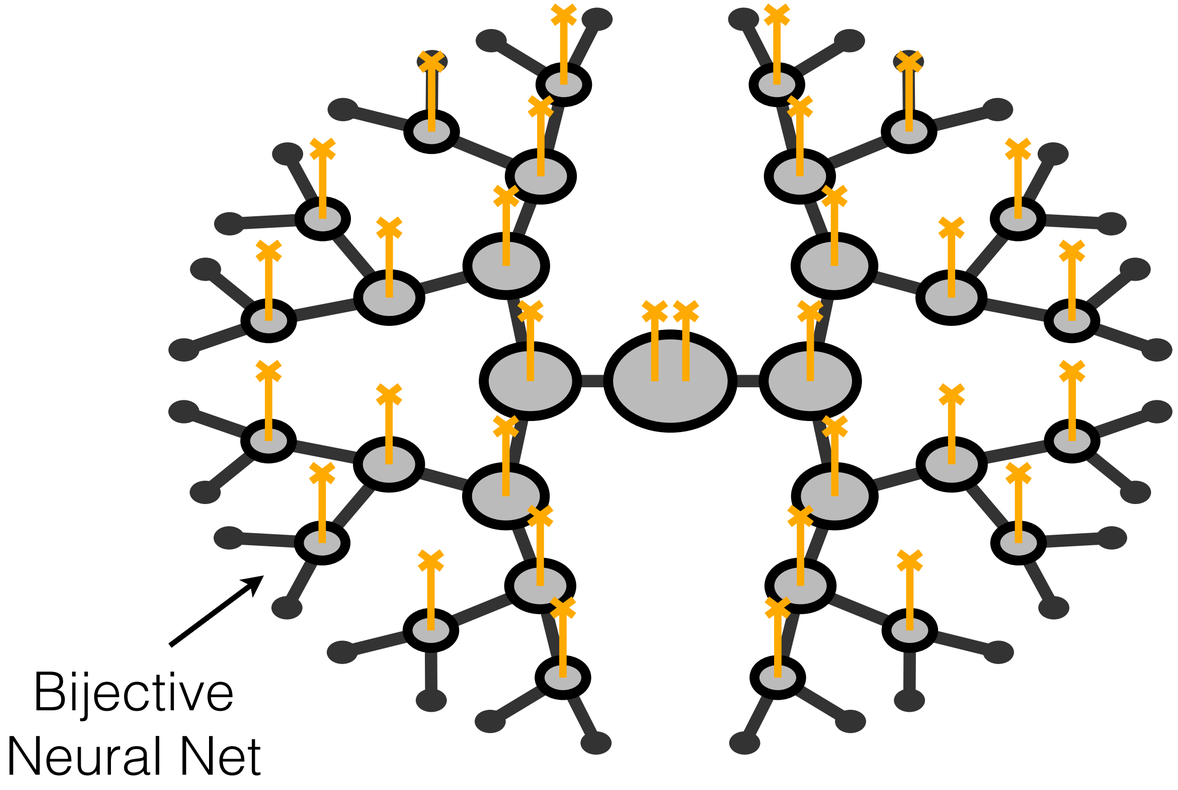
Guy, Wavelets & RG1999+ White, Evenbly, Qi, Wavelets, MERA, and holographic mapping 2013+

Neural Renormalization Group Flow

Li and LW, PRL '18

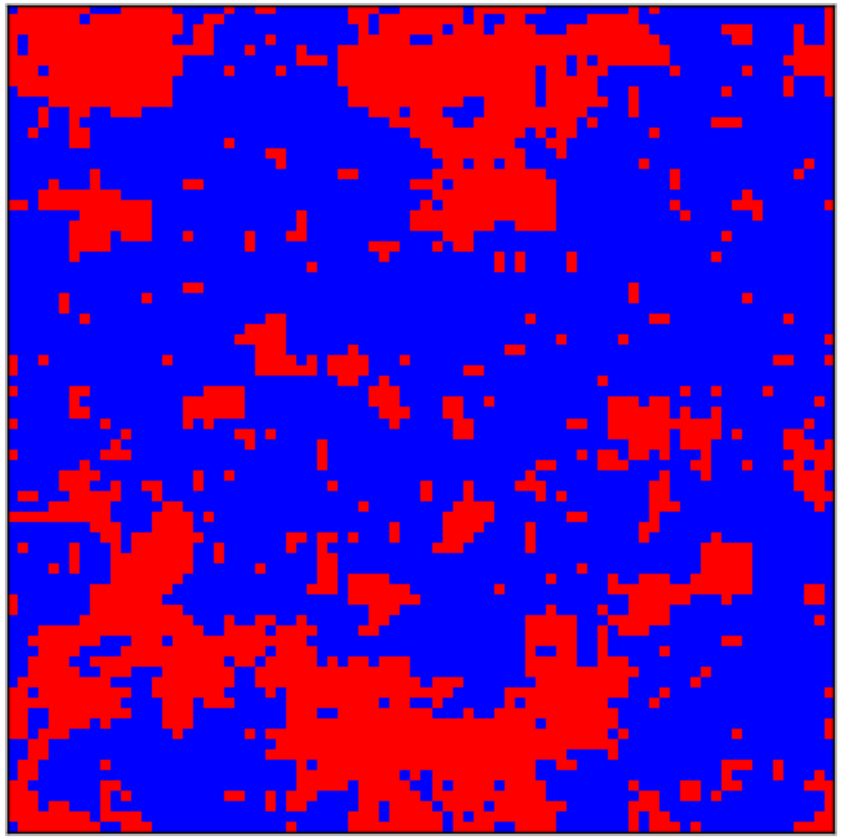
Hu, Li, LW, You, 1903.00804

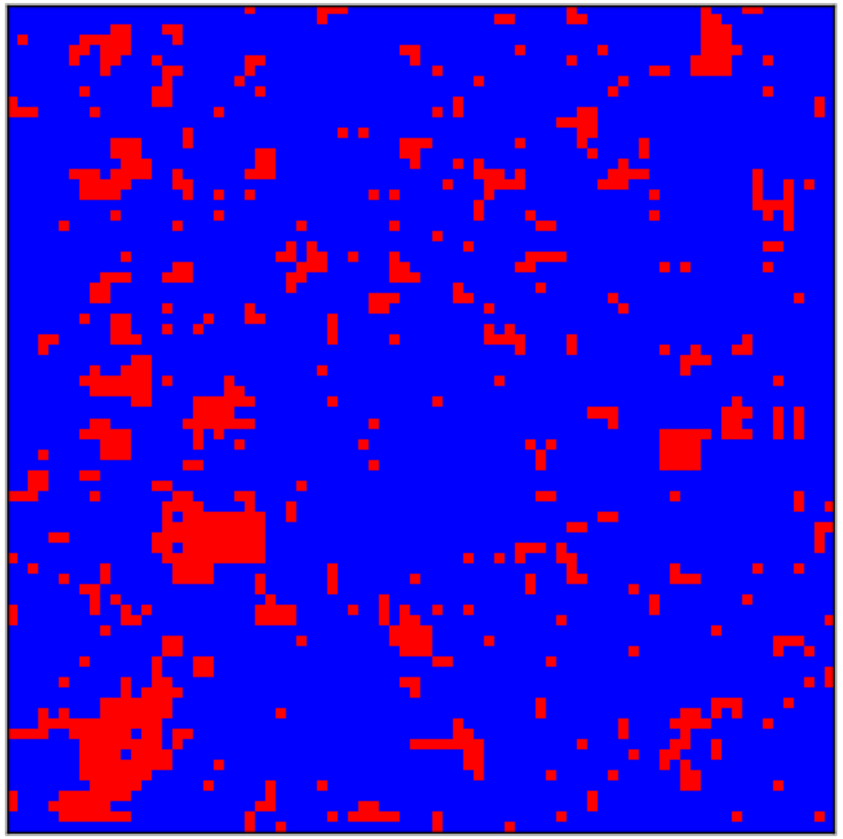




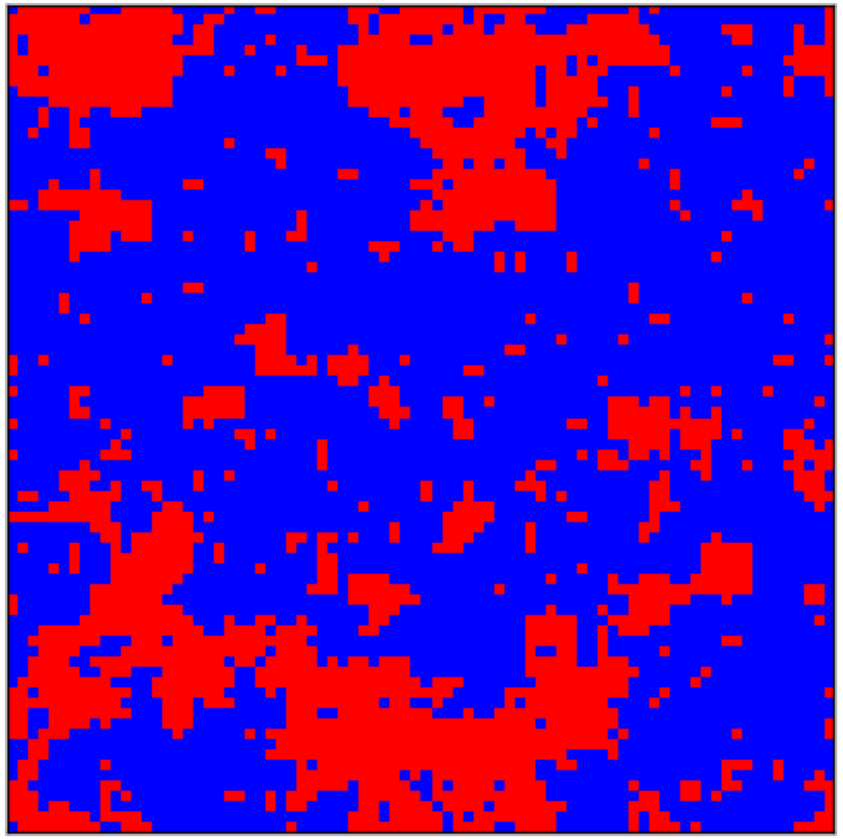
Multiscale invertible network for RG and holographic duality

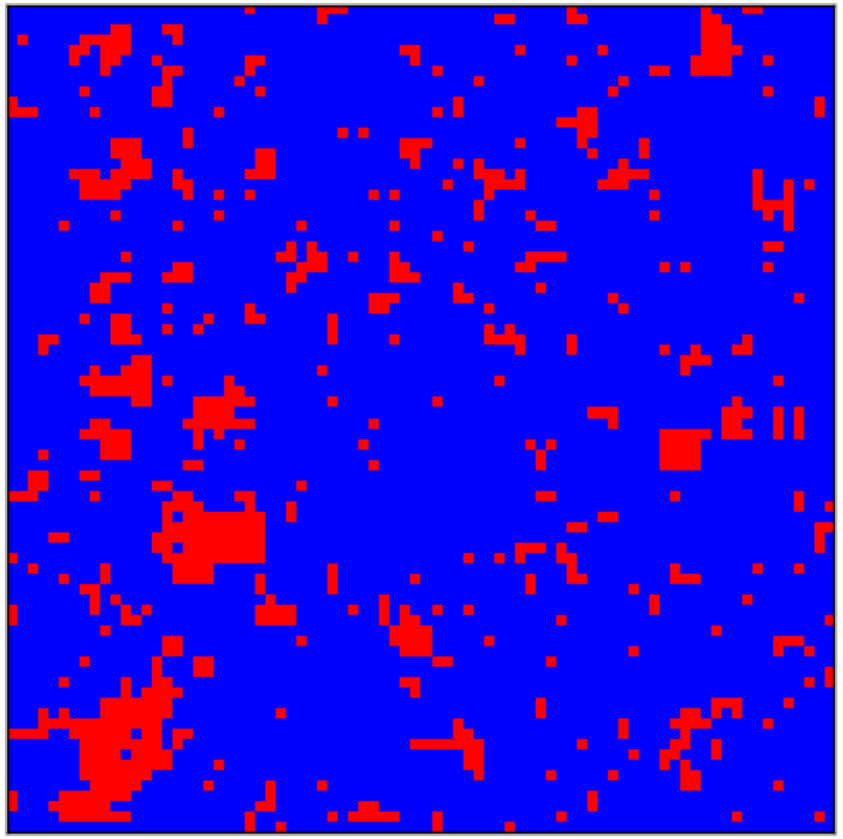
Local vs Cluster Monte Carlo updates



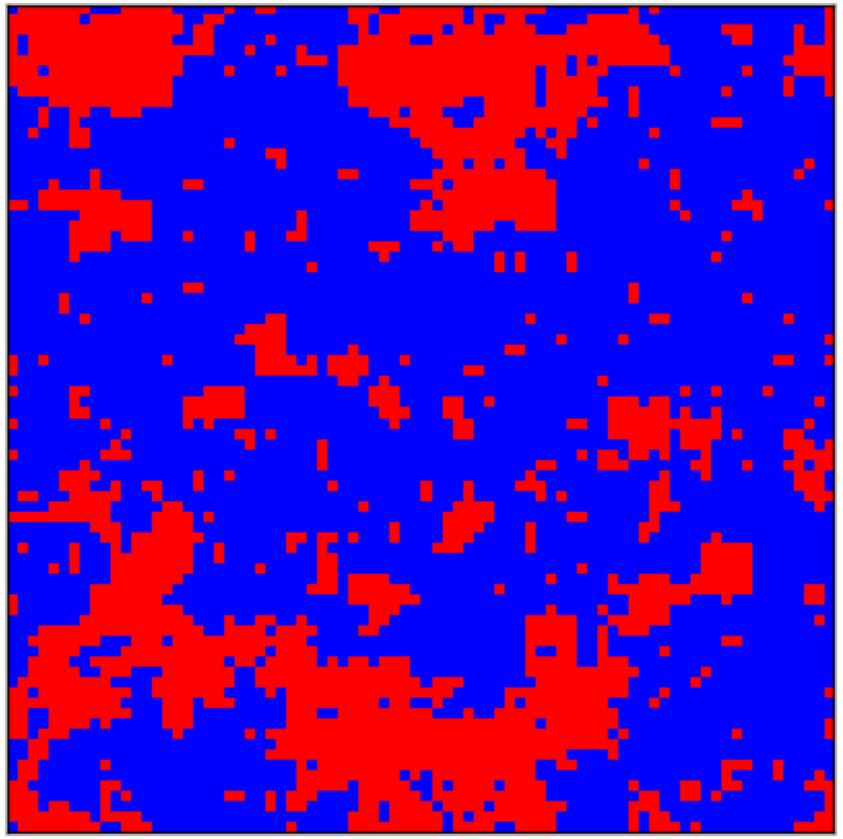


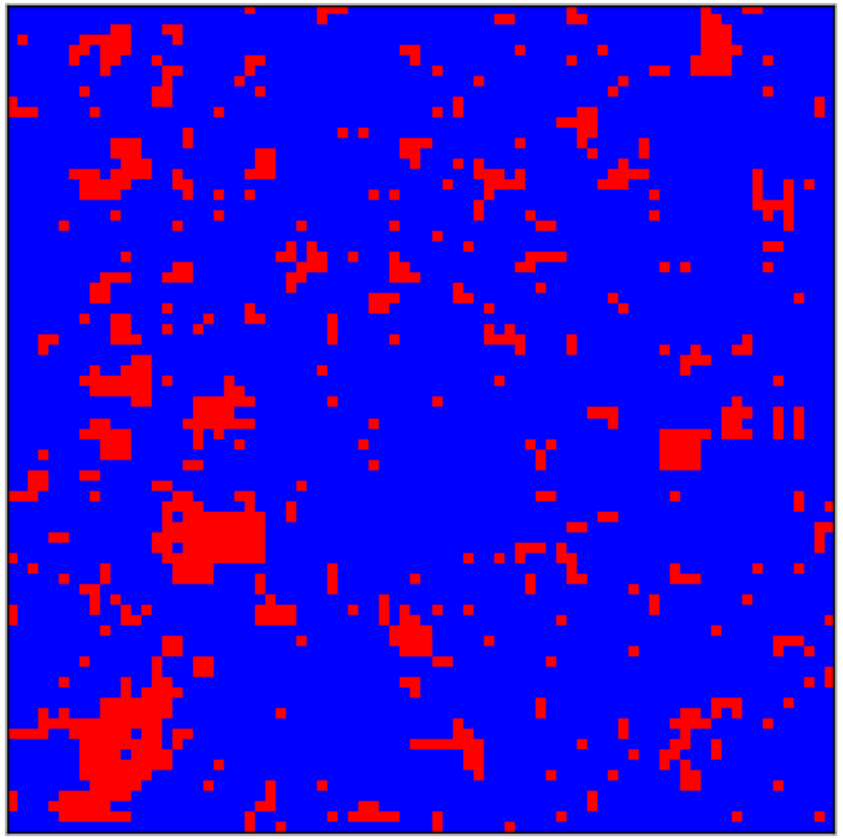
Local vs Cluster Monte Carlo updates



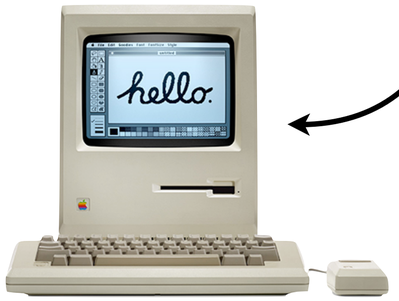


Local vs Cluster Monte Carlo updates

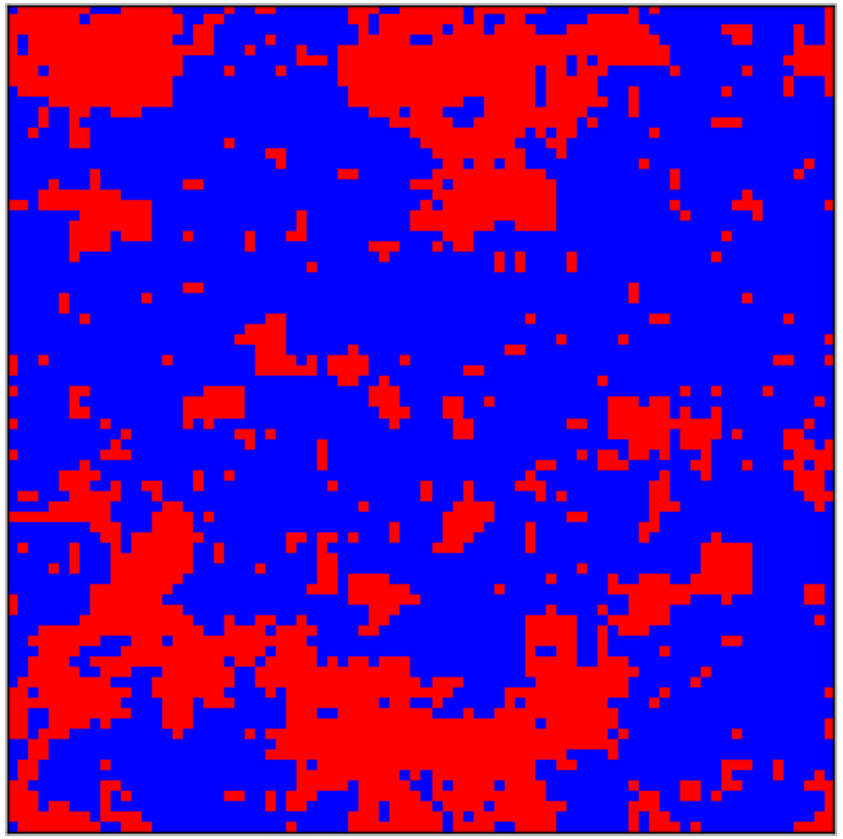


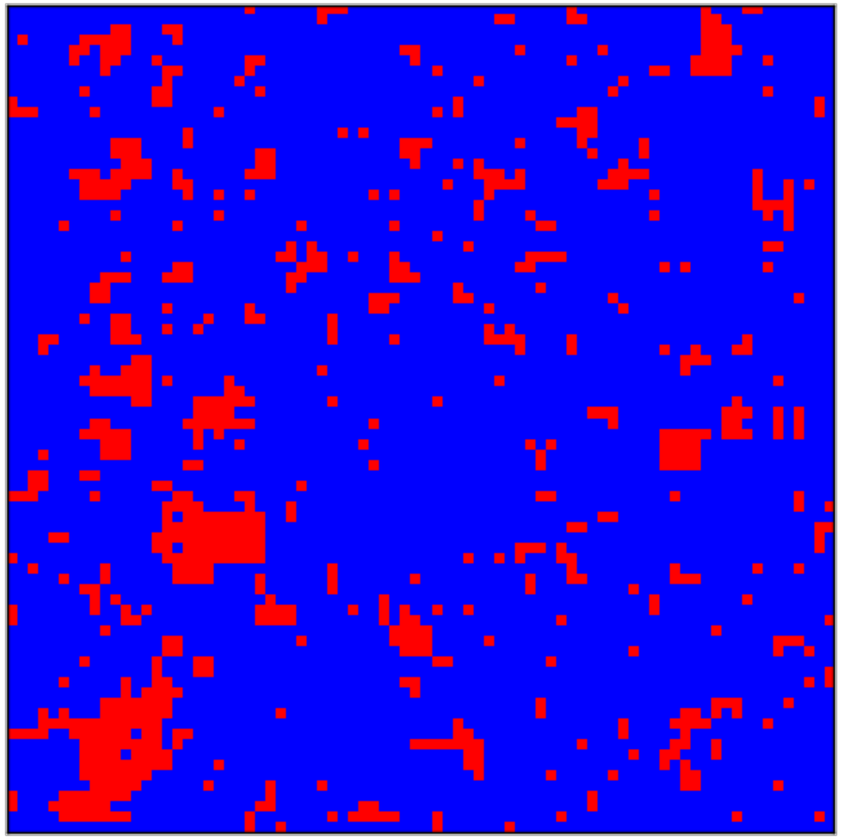


is slower than



Local vs Cluster Monte Carlo updates

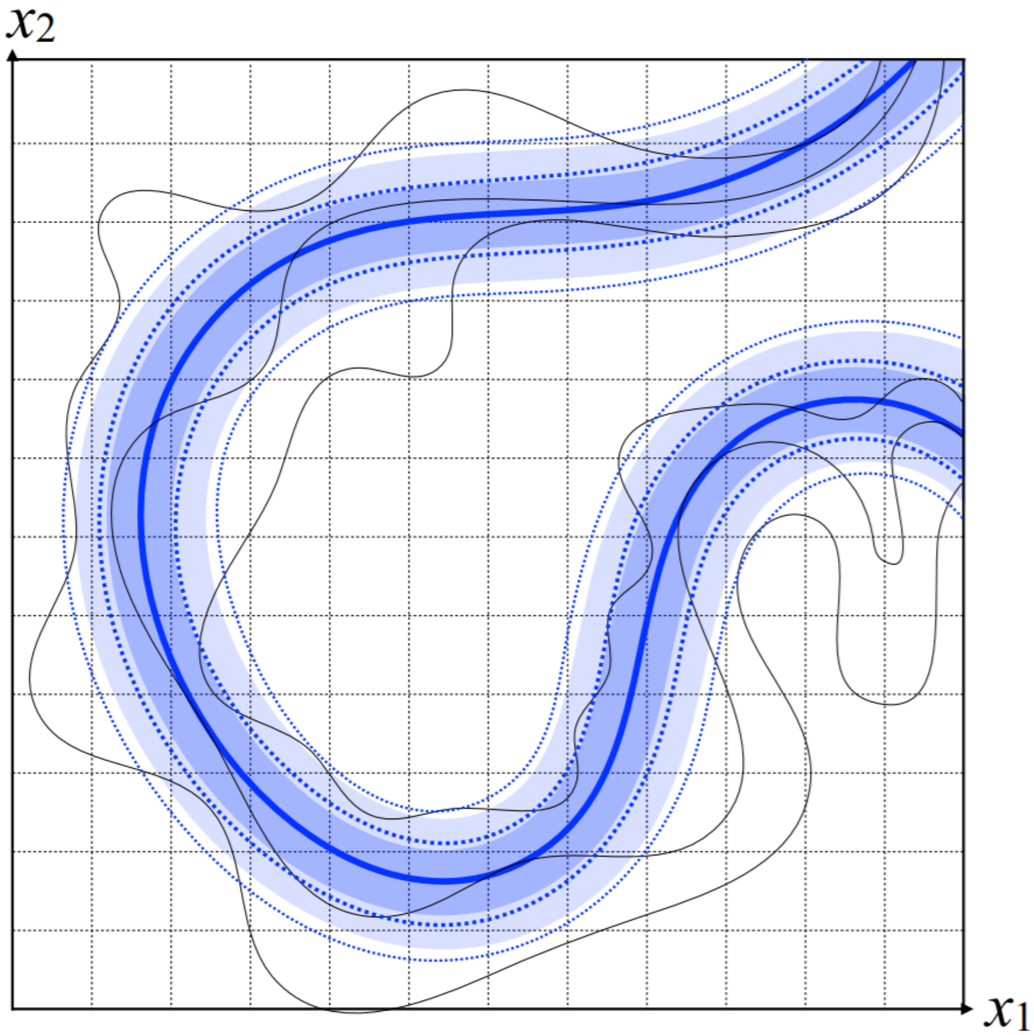




Algorithmic innovation outperforms Moore's law!

Physics happens on a manifold

Monte Carlo sampling of the physical d.o.f. can be difficult since they live in a nontrivial manifold

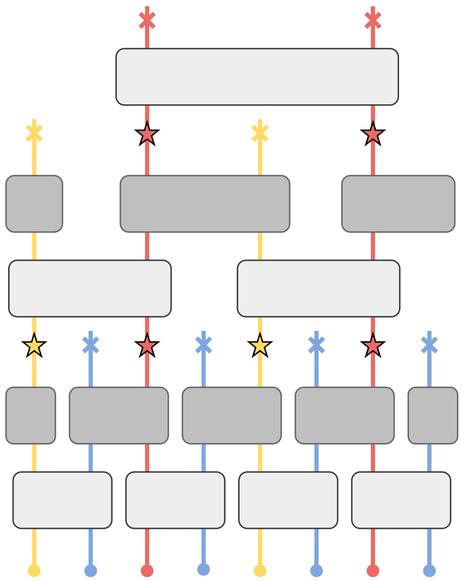


Representation learning unfolds the manifold

Latent space Hybrid MC

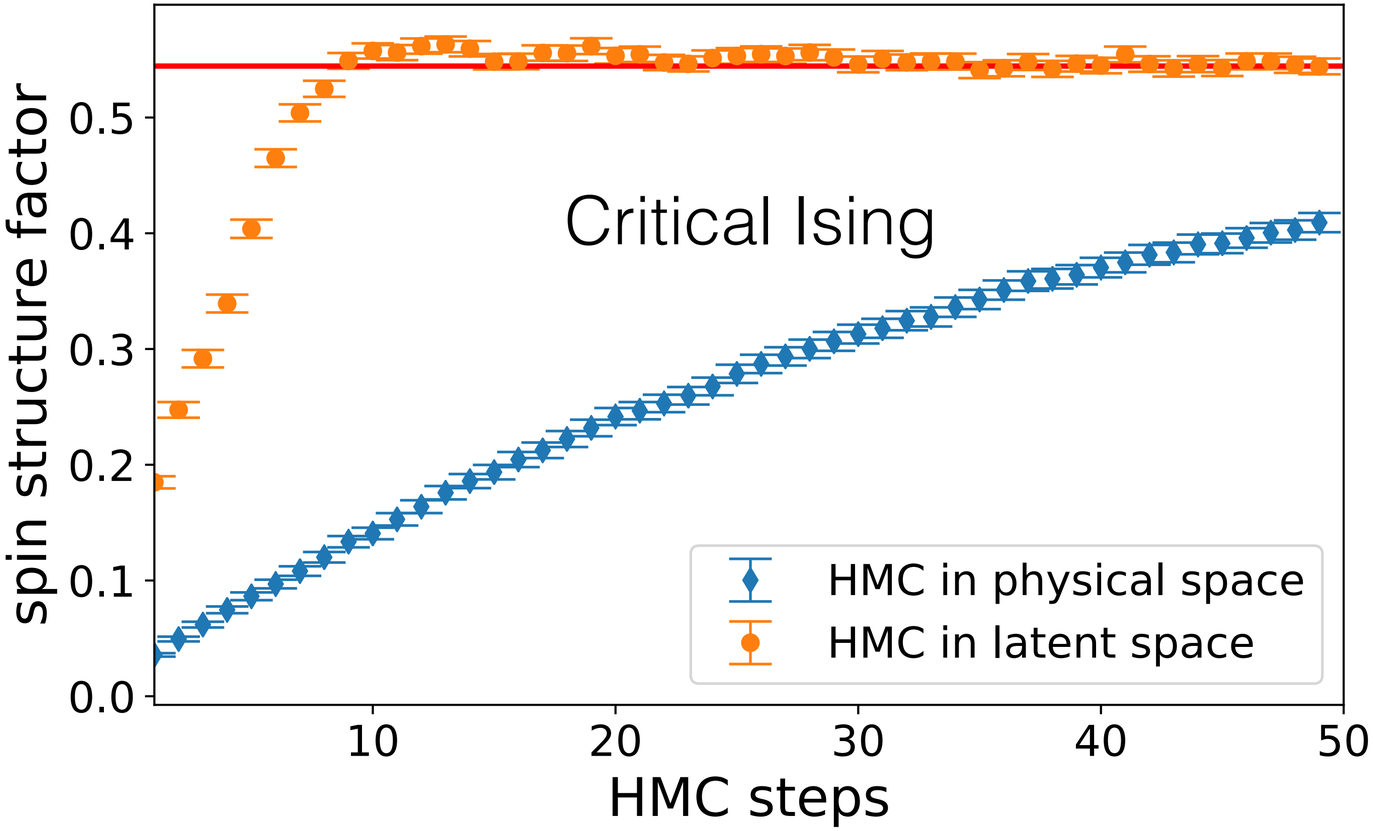
Latent space energy function

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



Physical energy function $E(x)$

https://github.com/li012589/NeuralRG

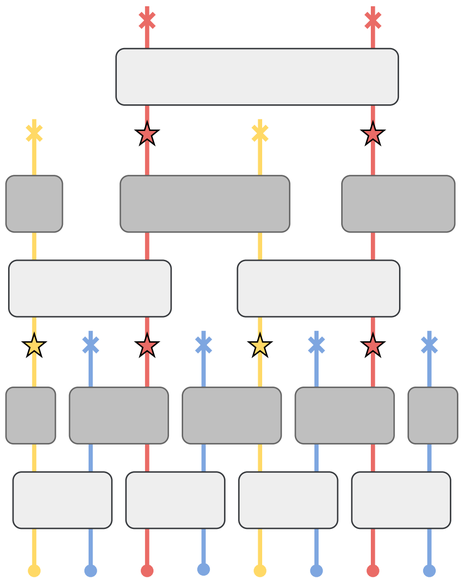


HMC thermalizes faster in the latent space Li and LW, PRL '18

Latent space Hybrid MC

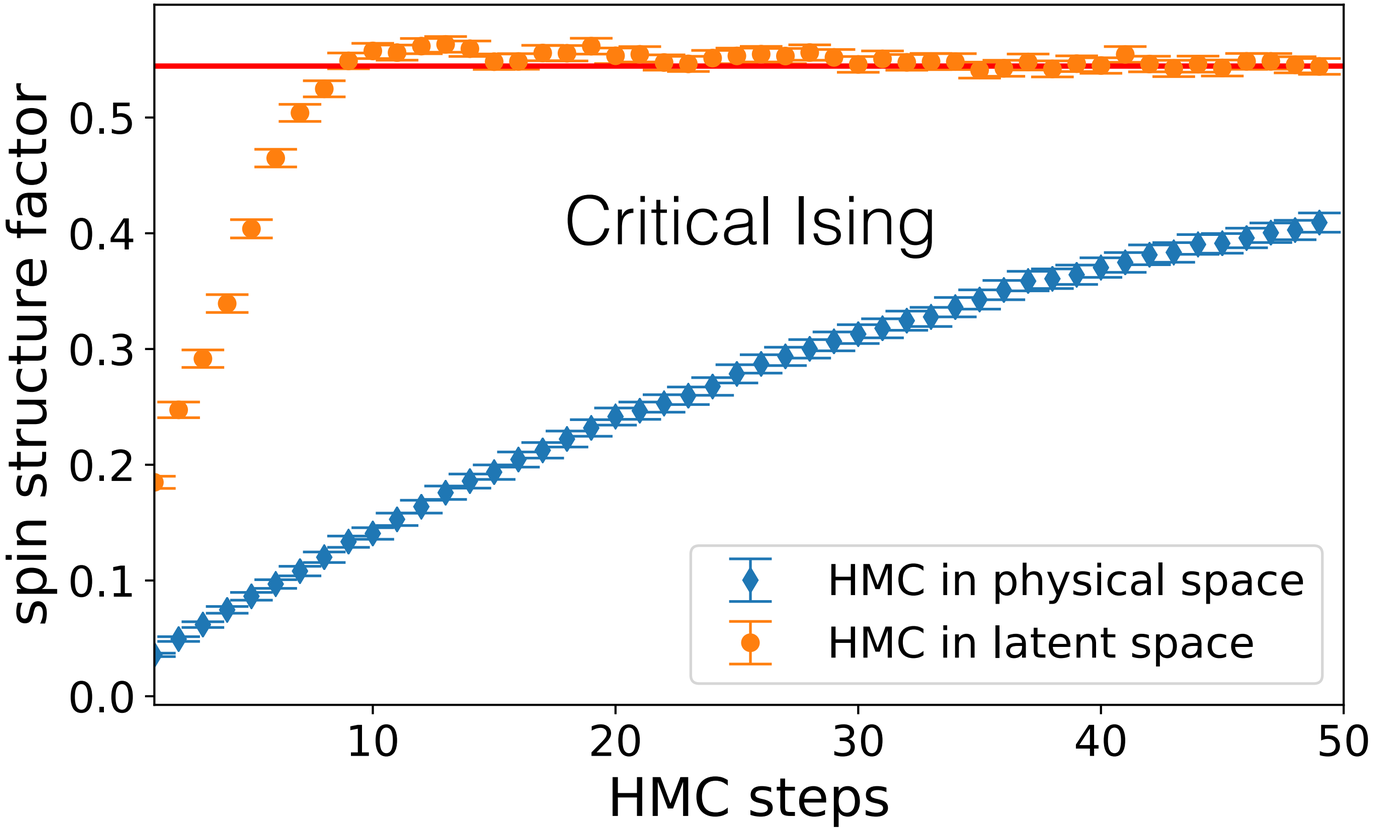
Latent space energy function

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



Physical energy function $E(x)$

https://github.com/li012589/NeuralRG

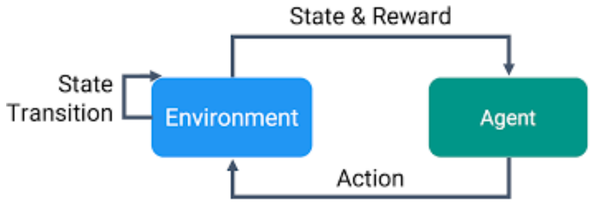
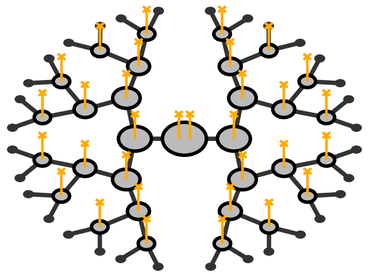


HMC thermalizes faster in the latent space Li and LW, PRL '18

Remarks on accelerated MC sampling

1. Cheap surrogate function for MC weight Neal 96' Jun. S Liu 01' A recommender engine for MC updates when the surrogate is a generative model: Huang, LW, 1610.02746, Liu, Qi, Meng, Fu, 1610.03137

2. Reinforcement learning the transition kernel: Song et al 1706.07561, Levy et al 1711.09268, Cusumano-Towner et al 1801.03612, Bojesen 1808.09095, Zhao et al 1903.04698



3. Performs MC in the variationally learned disentangled representation Adaptive generalization of Wavelet MC, Ismail '03 and spatial warping, Zhu el al '02, Minary et al '08