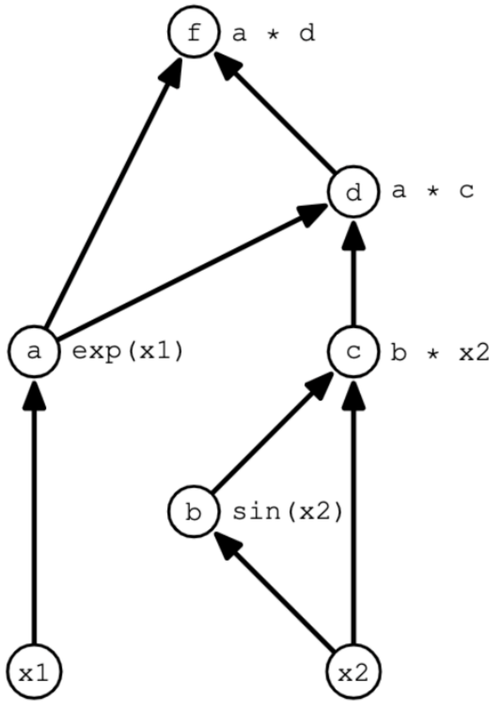
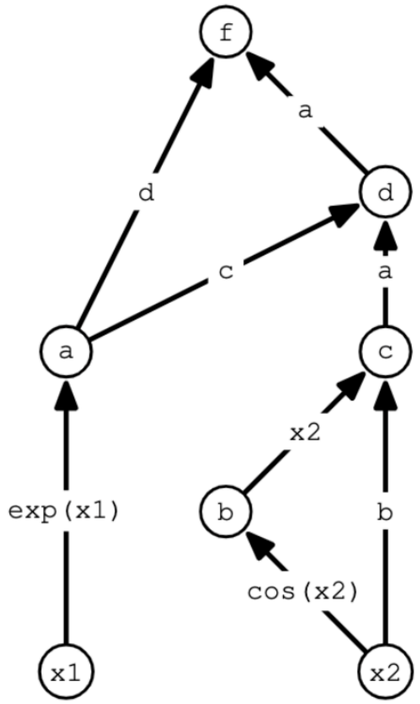
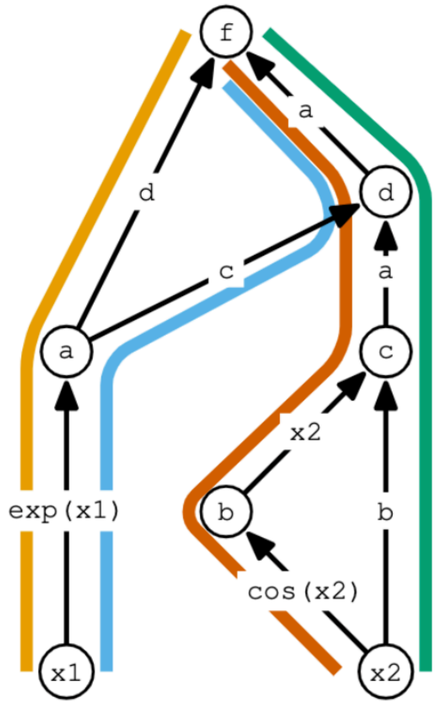
Automatic differentiation







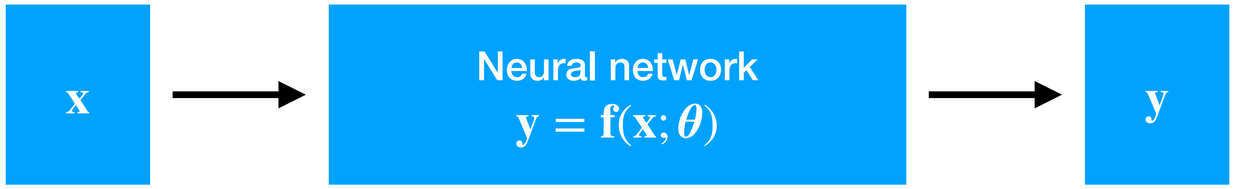
Cal Floyd
ML Journal club 5/9

Overview

- Basics of autodiff
  - Forward and reverse mode
  - Tradeoffs between the two
- Packages for differentiating through complex simulations
  - JAX MD, CFD
- Physical / chemical applications
  - Non-equilibrium protocols, avoiding kinetic traps, flows of complex fluids, bulk properties of glasses
- Alternative training methods
  - Equilibrium propagation, randomized AD, “neuroevolution”

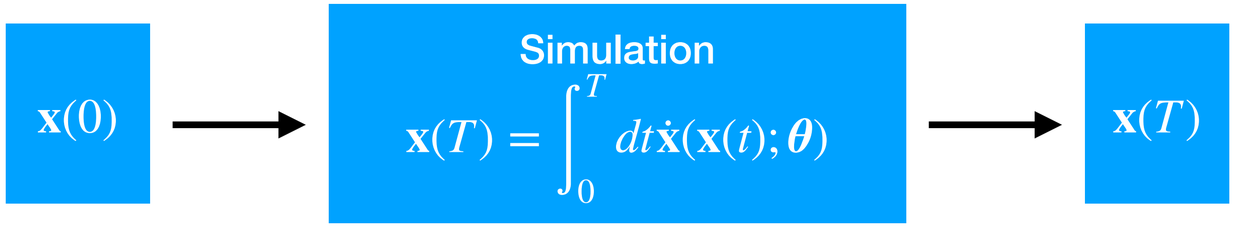
Differentiate everything

- The typical example of AD is back-propagation:



Backprop gives $\left.\frac{\partial \mathbf{y}}{\partial \boldsymbol{\theta}}\right|_{\mathbf{x}}$

- What if our computation is a time integration of some dynamics?



Can autodiff give us $\left.\frac{\partial \mathbf{x}(T)}{\partial \boldsymbol{\theta}}\right|_{\mathbf{x}(0)}$?

Or $\left.\frac{\partial F[\mathbf{x}(t)]}{\partial \boldsymbol{\theta}}\right|_{\mathbf{x}(0)}$?

- What if our computation is an energy minimization?

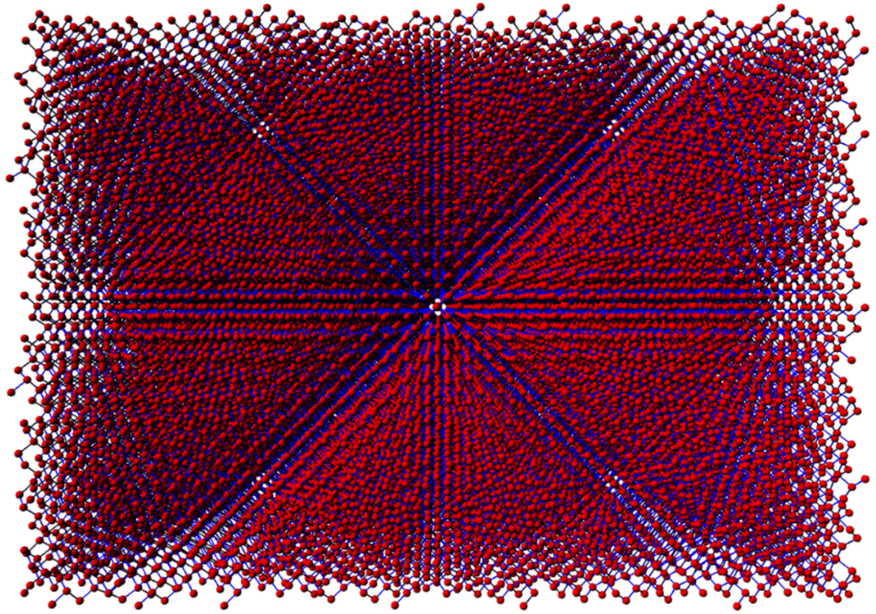
Can autodiff give us $\left.\frac{\partial F(\mathbf{x}^*)}{\partial \boldsymbol{\theta}}\right|_{\mathbf{x}(0)}$?

JAX, M.D.

JAX, M.D.
A Framework for Differentiable Physics

Samuel S. Schoenholz
Google Research: Brain Team
schsam@google.com

Ekin D. Cubuk
Google Research: Brain Team
cubuk@google.com



A large crystal simulation using NN~DFT potentials

- Advertised as a “bridge” between JAX and commonly used MD (many-body dynamics) algorithms
  - Avoid re-inventing the wheel for new physics applications
- Implements:
  - User defined energy functions -> forces via autograd
  - Spatial voxels and neighbor lists
  - Deep and graph neural networks
  - NVE / Nose-Hoover / Langevin / Brownian dynamics / descent dynamics / general update functions
  - Common interaction potentials (Morse / LJ / …)
- Reported applications in: approximating DFT trajectories via GNNs, particle packing, flocking simulations

Tuning glass properties

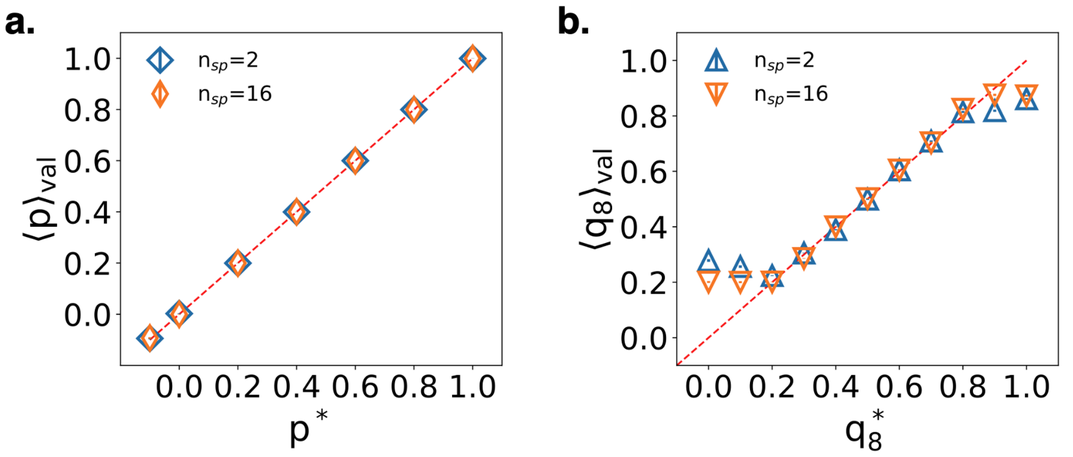
Designing athermal disordered solids with automatic differentiation

Mengjie ZU* and Carl GOODRICH†
Institute of Science and Technology Austria
(Dated: April 24, 2024)

$$V(r) = \begin{cases} \frac{k}{2}(r-\bar{\sigma})^2 - B, & r < \bar{\sigma} \\ B(e^{-2a(r-\bar{\sigma})} - 2e^{-a(r-\bar{\sigma})}), & r \geq \bar{\sigma} \end{cases}$$

terial properties. More precisely, we construct an objective function $\mathcal{L}$, e.g. $\mathcal{L} = (\nu - \nu^*)^2$, that indicates how far a property (e.g. ν) is from a target (e.g. ν^*), and use Automatic Differentiation (AD) [10–12] to calculate the gradient $\nabla_\theta \mathcal{L}$ of $\mathcal{L}$ with respect to the parameters $\theta = \{\sigma_\alpha\} \cup \{B_{\alpha\beta}\}$. $\nabla_\theta \mathcal{L}$ indicates how changes to the

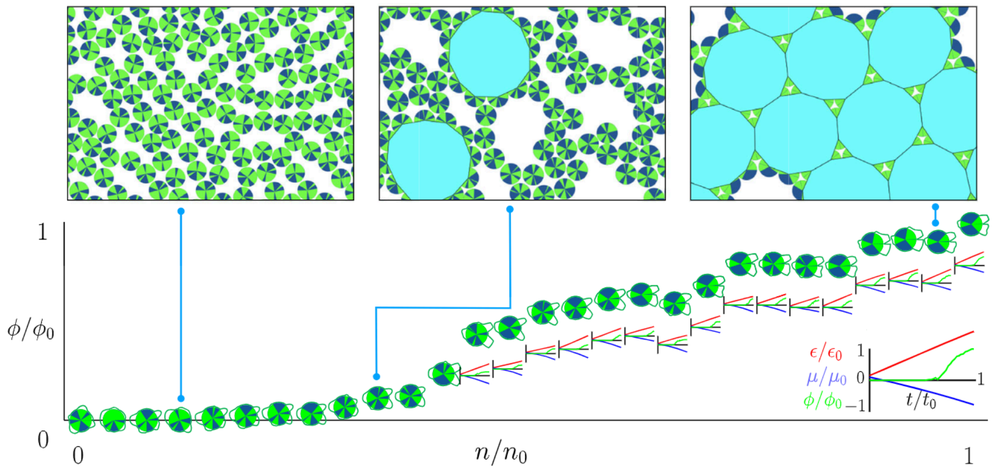
- In disordered spring networks, removing %1 of the right springs can push the Poisson ratio to its limiting values
- How can this work in multi-species particulate mixtures?
- Uses JAX-MD to **backprop through energy minimization** and linear response calculation (of Poisson ratio)
- For single configurations, can get $\nu \sim -1$, and for an ensemble of configurations can get $0.2 < \langle \nu \rangle < 0.7$
- Can train multi-object goals (order parameter and pressure)



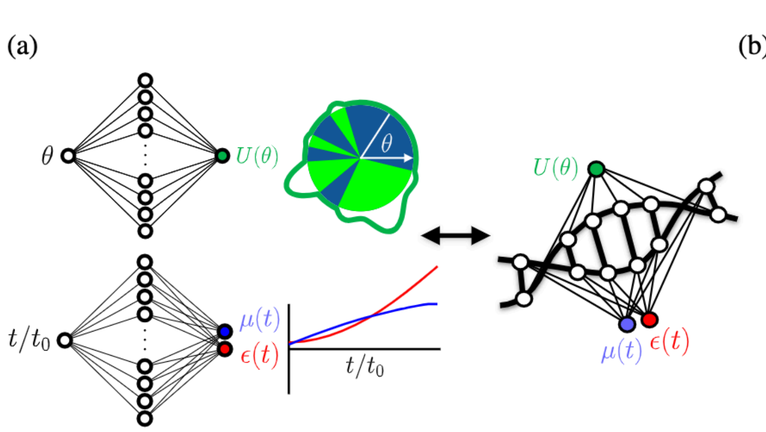
Alternative: “neuroevolution”

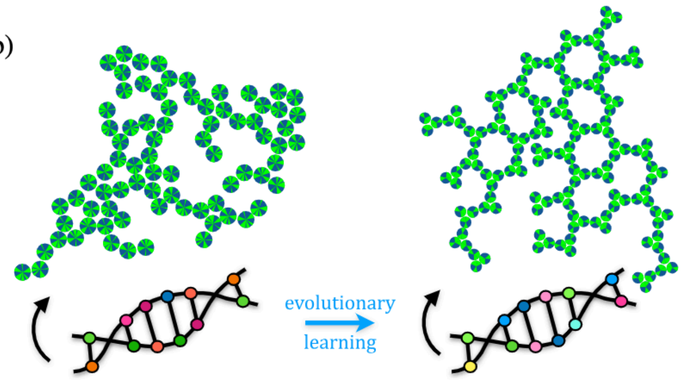
Neuroevolutionary Learning of Particles and Protocols for Self-Assembly

Stephen Whitelam and Isaac Tamblyn
Phys. Rev. Lett. **127**, 018003 – Published 29 June 2021



- Similar task: design “patchy” particles and a control protocol to achieve desired order parameter
- Parameterize the radial interaction and the “annealing protocol” as neutral networks
- No gradient descent: treat n.n. parameters as “genes” and do **evolutionary mutation** on them
  - Genetic algorithm on the network weights
- Works to anneal the particles into connected 12-gons
- Can also use novelty-based objectives to find new configs





Neuroevolution approximates gradients

ARTICLE

https://doi.org/10.1038/s41467-021-26568-2 OPEN

Check for updates

Correspondence between neuroevolution and gradient descent

Stephen Whitelam[1✉], Viktor Selin[2], Sang-Won Park[1] & Isaac Tamblyn[2,3,4✉]

- Neuroevolution is a Monte Carlo minimization of the cost
- For small Monte Carlo moves, it corresponds to Langevin dynamics on the cost function (already known…)

Neuroevolution. Now consider training the network by neuroevolution, defined by the following Monte Carlo protocol.

1. Initialize the neural-network parameters $\mathbf{x}$ and calculate the loss function $U(\mathbf{x})$. Set time $t = 0$.
2. Propose a change (or "mutation") of each neural-network parameter by an independent Gaussian random number of zero mean and variance σ^2, so that
$$\mathbf{x} \to \mathbf{x} + \boldsymbol{\epsilon}, \tag{2}$$
where $\boldsymbol{\epsilon} = \{\epsilon_1, \ldots, \epsilon_N\}$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.
3. Accept the mutation with the Metropolis probability $\min\left(1, \mathrm{e}^{-\beta[U(\mathbf{x}+\boldsymbol{\epsilon})-U(\mathbf{x})]}\right)$, and otherwise reject it. In the latter case we return to the original neural network. The parameter β can be considered to be a reciprocal evolutionary temperature.
4. Increment time $t \to t+1$, and return to step 2.

$$\partial_t P(\mathbf{x},t) = \int \mathrm{d}\boldsymbol{\epsilon}\left[P(\mathbf{x}-\boldsymbol{\epsilon},t)W_{\boldsymbol{\epsilon}}(\mathbf{x}-\boldsymbol{\epsilon}) - P(\mathbf{x},t)W_{\boldsymbol{\epsilon}}(\mathbf{x})\right].$$

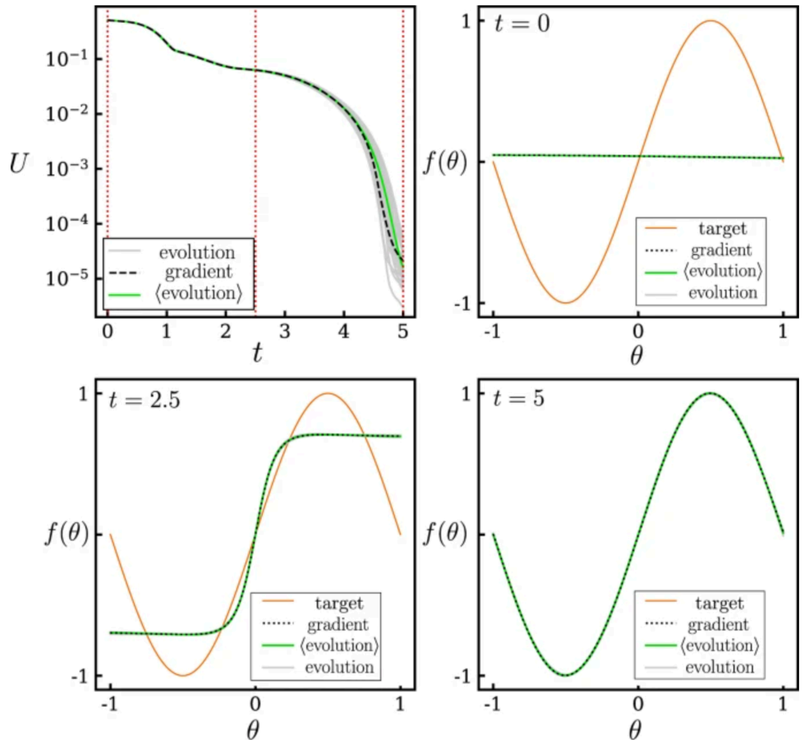
$$W_{\boldsymbol{\epsilon}}(\mathbf{x}) = p(\boldsymbol{\epsilon})\min\left(1, \mathrm{e}^{-\beta[U(\mathbf{x}+\boldsymbol{\epsilon})-U(\mathbf{x})]}\right)$$

$$P(\mathbf{x}-\boldsymbol{\epsilon},t) \approx \left(1 - \boldsymbol{\epsilon}\cdot\nabla + \frac{1}{2}(\boldsymbol{\epsilon}\cdot\nabla)^2\right)P(\mathbf{x},t)$$

$$W_{\boldsymbol{\epsilon}}(\mathbf{x}-\boldsymbol{\epsilon}) \approx \left(1 - \boldsymbol{\epsilon}\cdot\nabla + \frac{1}{2}(\boldsymbol{\epsilon}\cdot\nabla)^2\right)W_{\boldsymbol{\epsilon}}(\mathbf{x}),$$

$$\partial_t P(\mathbf{x},t) \approx -\int \mathrm{d}\boldsymbol{\epsilon}(\boldsymbol{\epsilon}\cdot\nabla)P(\mathbf{x},t)W_{\boldsymbol{\epsilon}}(\mathbf{x}) + \frac{1}{2}\int \mathrm{d}\boldsymbol{\epsilon}(\boldsymbol{\epsilon}\cdot\nabla)^2 P(\mathbf{x},t)W_{\boldsymbol{\epsilon}}(\mathbf{x}).$$

$$\partial_t P(\mathbf{x},t) \approx -\sum_{i=1}^{N}\frac{\partial}{\partial x_i}\left(A_i(\mathbf{x})P(\mathbf{x},t)\right) + \frac{1}{2}\sum_{i,j=1}^{N}\frac{\partial^2}{\partial x_i \partial x_j}\left(B_{ij}(\mathbf{x})P(\mathbf{x},t)\right),$$



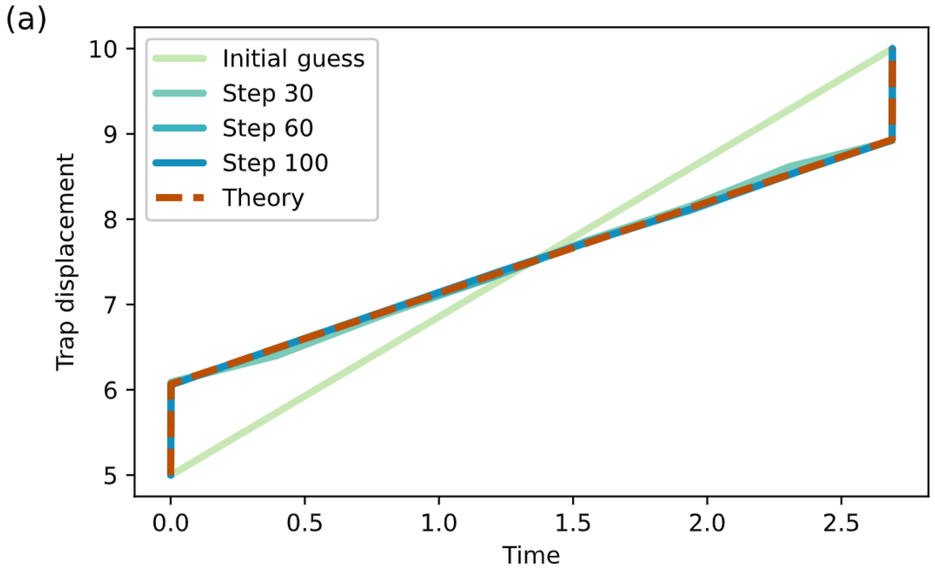
Minimally dissipative non-eq dynamics

Open Access

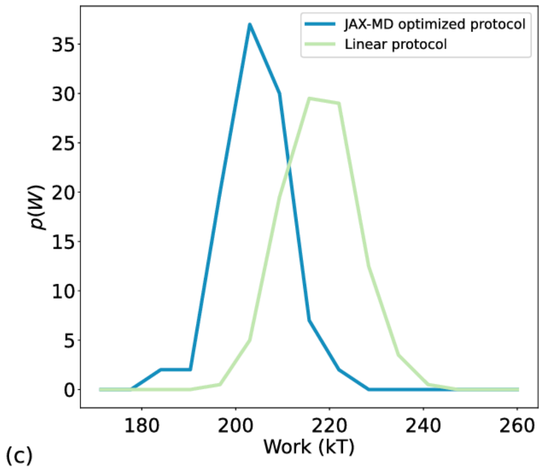
Optimal Control of Nonequilibrium Systems through Automatic Differentiation

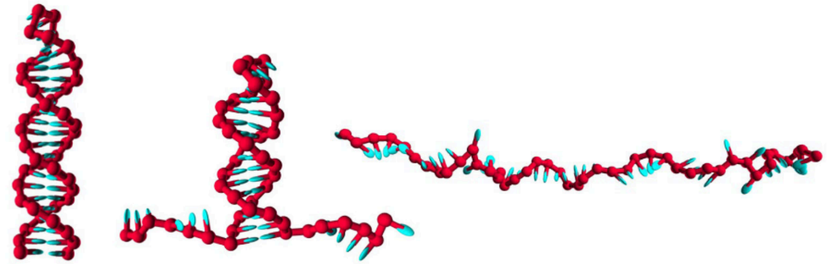
Megan C. Engel, Jamie A. Smith, and Michael P. Brenner
Phys. Rev. X **13**, 041032 – Published 16 November 2023

- Optimal (minimally dissipative) control of non-equilibrium dynamics received much attention starting from ~2010
  - Analytical solutions are known to canonical problems (particle in a trap, applied field on Ising model)
- Autodiff finds the same solutions and can work far from equilibrium
- Sensitive to how the protocols are parameterized (piecewise linear, Chebyshev, …)



Particle in harmonic trap





Did not actually backprop through this simulation

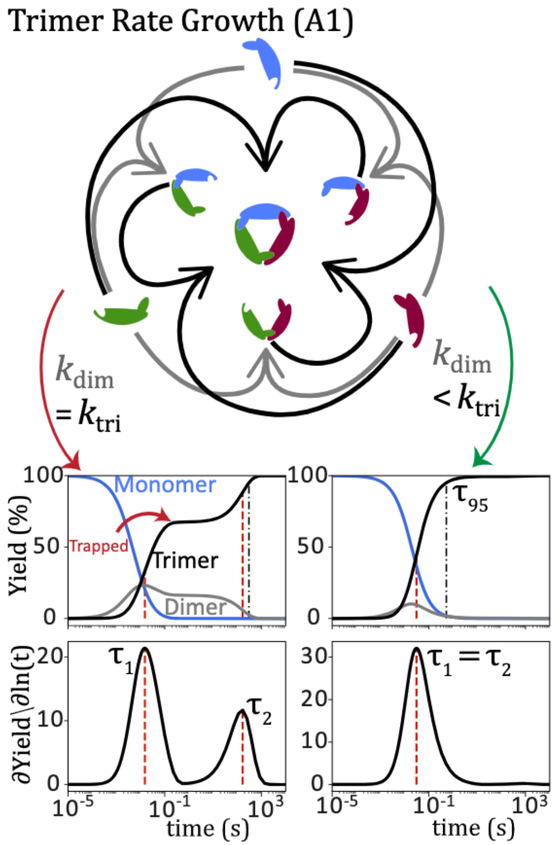
Avoiding kinetic traps

PNAS RESEARCH ARTICLE | BIOPHYSICS AND COMPUTATIONAL BIOLOGY

Discovering optimal kinetic pathways for self-assembly using automatic differentiation

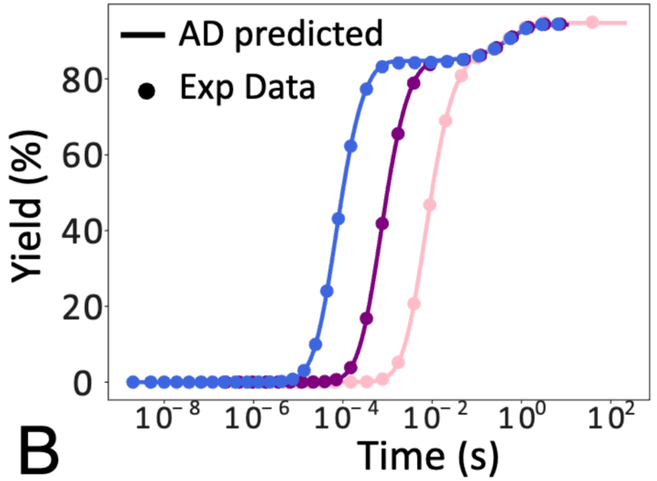
Adip Jhaveri[a,1], Spencer Loggia[a,1], Yian Qian[a], and Margaret E. Johnson[a,2]

Edited by Chris Jarzynski, University of Maryland, College Park, MD; received February 22, 2024; accepted April 3, 2024



- Achieving high yield requires avoiding kinetic traps
- How can the kinetic rates be tuned to avoid this?
- Method: numerically integrate the mass-action ODEs and measure yield at t_{stop}
  - Backprop through time to obtain $\partial_{k_i}\mathscr{L}$
  - Update k_i with regularization to ensure same K_{eq}
- Can also be used to fit parameters to experimental trajectories!

$$SAE = \sum_{j=1}^{N_C} \sum_{t=t_{min}}^{t_{max}} \left| Y_{true}(t_i, C_j) - Y_{pred}(t_i, C_j) \right|$$



JAX, CFD

RESEARCH ARTICLE | APPLIED MATHEMATICS

Machine learning–accelerated computational fluid dynamics

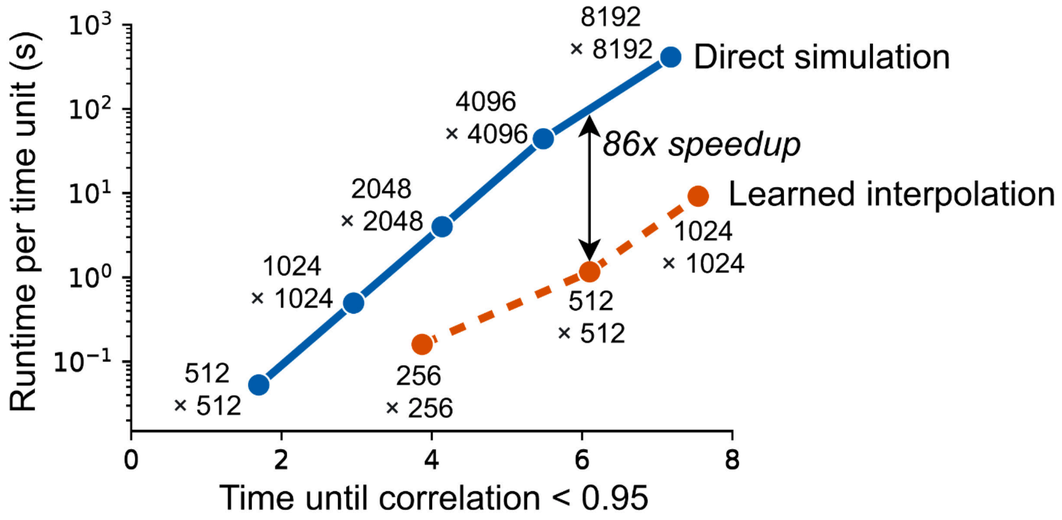
Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, +2, and Stephan Hoyer Authors Info & Affiliations

Edited by Andrea L. Bertozzi, University of California, Los Angeles, CA, and approved March 25, 2021 (received for review January 29, 2021)

May 18, 2021 | 118 (21) e2101784118 | https://doi.org/10.1073/pnas.2101784118

93,114 | 154 PDF/EPUB

710 citations since 2021!

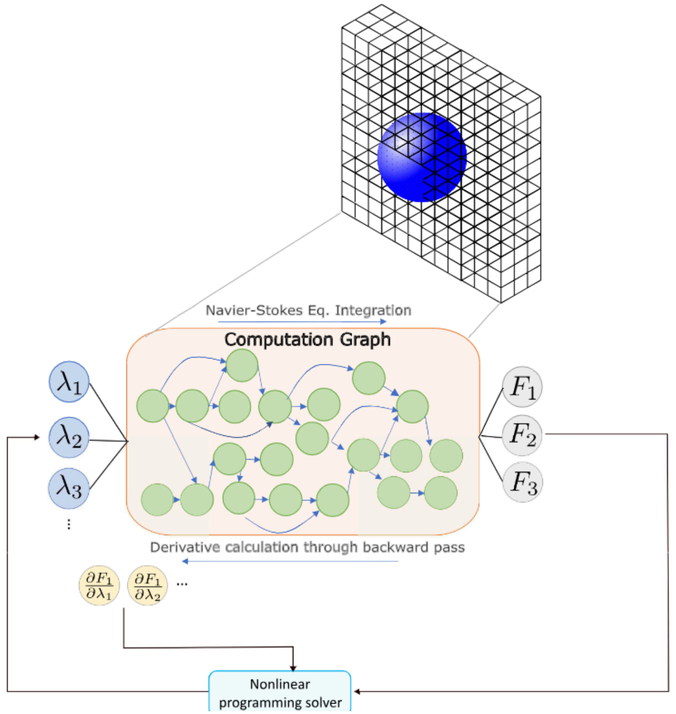


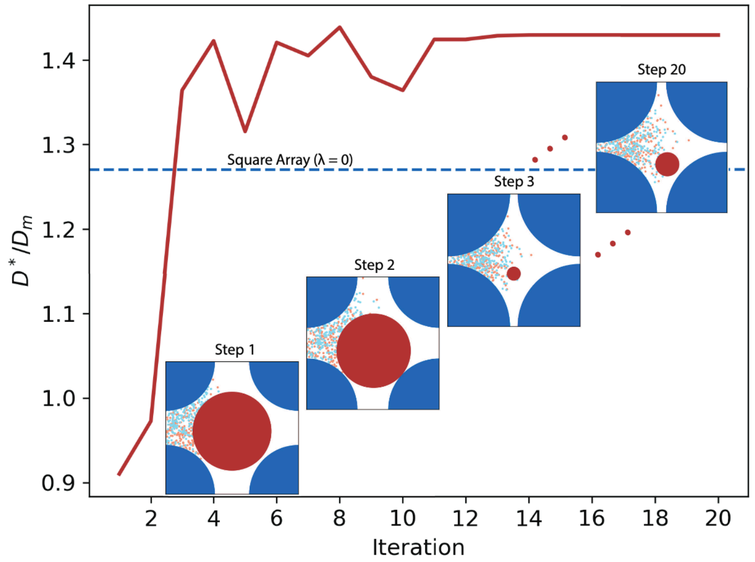
- Uses ML to approximate a full CFD simulation
- Speed up ~85x for incompressible Navier-Stokes
  - Helpful for climate modeling and airplane design (pay attention, Boeing)
- One option is use ML to “learn” the Navier-Stokes equation
  - Generalizes poorly and doesn’t enforce constraints (incompressibility)
- This approach uses ML to **apply corrections** to a cheap, coarse simulation
- Necessary gradients computed using autograd in JAX,
  - It’s possible to backprop through the simulation to do optimizations

Fluid-structure interactions

Control of flow behavior in complex fluids using automatic differentiation

Mohammed G. Alhashim * [a,c], Kaylie Hausknecht [b], and Michael P. Brenner[†a,b]





Change the red obstacle to optimize mixing

- Extensions to JAX-CFD allow treating fluid-structure interactions
- The immersed boundary method is used and its computational graph (along with Navier-Stokes) is differentiated
- Applied to
  - Optimize topology of porous medium
  - Optimize tracker dispersion in porous medium
  - Optimize mixing in a journal bearing
  - Optimize propulsion efficiency of a swimmer

Randomized AD

Published as a conference paper at ICLR 2021

RANDOMIZED AUTOMATIC DIFFERENTIATION

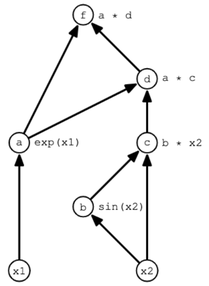
Deniz Oktay[1], **Nick McGreivy**[2], **Joshua Aduol**[1], **Alex Beatson**[1], **Ryan P. Adams**[1]
Princeton University
Princeton, NJ
{doktay,mcgreivy,jaduol,abeatson,rpa}@princeton.edu

- Most training expense comes from computing the gradient, but we take it over a **stochastic sample** of data
- We don't need high precision gradient estimates for this inherently noisy computation!
- Options:
  - Sample of subset of the computational paths during the gradient calculation
  - Inject random matrices $E[\mathbf{P}] = \mathbf{I}$ which are sparse, low rank, etc., in-between layers of AD
  - Similar to randomized linear algebra - large potential memory savings!

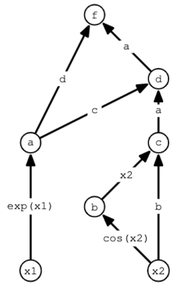
```
from math import sin, exp

def f(x1, x2):
  a = exp(x1)
  b = sin(x2)
  c = b * x2
  d = a * c
  return a * d
```

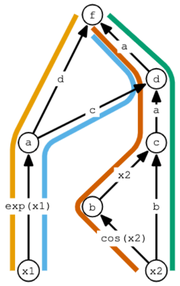
(a) Differentiable Python function



(b) Primal graph



(c) Linearized graph

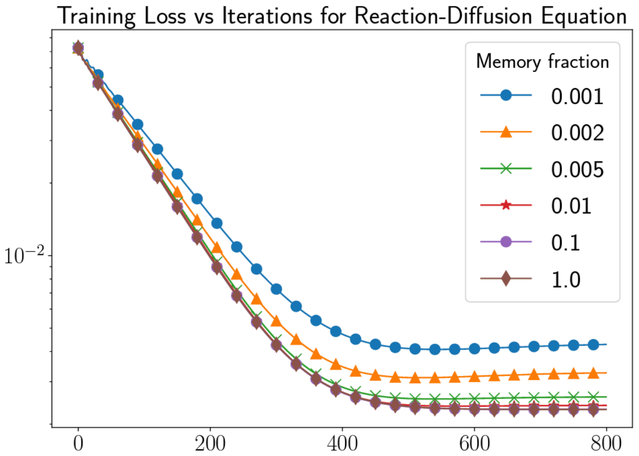


(d) Bauer paths

$$\frac{\partial y_j}{\partial \theta_i} = \mathcal{J}_\theta[f]_{j,i} = \sum_{[i \to j]} \prod_{(k,l) \in [i \to j]} \frac{\partial z_l}{\partial z_k}$$

$$\frac{\partial y}{\partial \theta} = \frac{\partial y}{\partial C}\frac{\partial C}{\partial B}\frac{\partial B}{\partial A}\frac{\partial A}{\partial \theta} \longrightarrow \frac{\partial y}{\partial C}P_2\frac{\partial C}{\partial B}P_2\frac{\partial B}{\partial A}P_1\frac{\partial A}{\partial \theta}$$

Inject random sparse matrices to save memory

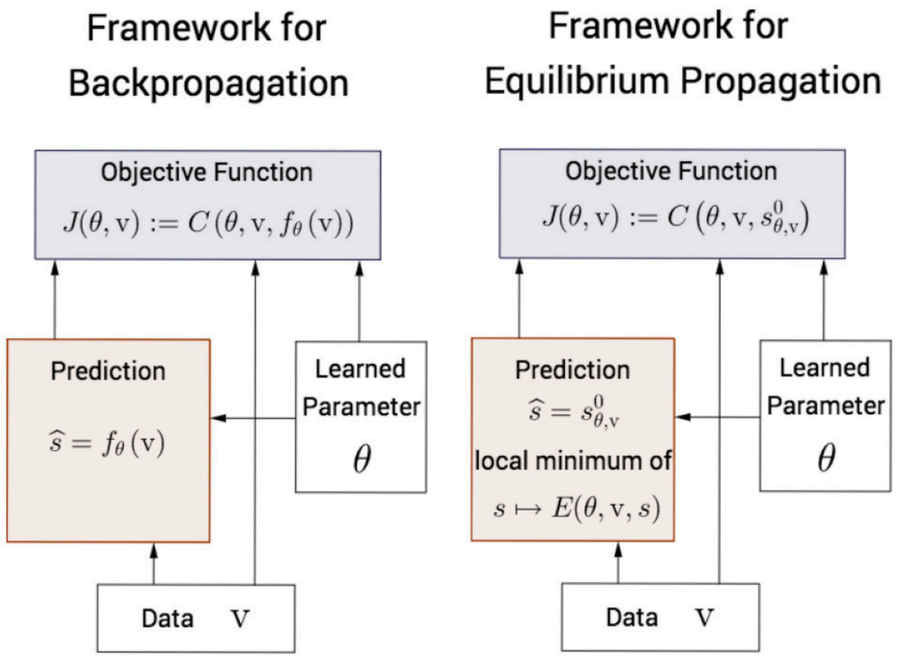


Equilibrium propagation

Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation

Benjamin Scellier and Yoshua Bengio †*

Département d'Informatique et de Recherche Opérationnelle, Montreal Institute for Learning Algorithms, Université de Montréal, Montreal, QC, Canada



- Training algorithm which avoids backpropagation of errors
- Requires a dynamical quantity $\mathscr{E}$ which the system extremizes at steady-state

1. Present an input pattern and let the system reach steady-state $\mathbf{p}^F$
2. Collect $\partial_{\boldsymbol{\theta}} \mathscr{E}[\mathbf{p}^F; \boldsymbol{\theta}]$
3. Nudge system toward desired output and let reach $\mathbf{p}^C$
4. Collect $\partial_{\boldsymbol{\theta}} \mathscr{E}[\mathbf{p}^C; \boldsymbol{\theta}]$
5. Update as $\Delta\boldsymbol{\theta} \sim \partial_{\boldsymbol{\theta}} \mathscr{E}[\mathbf{p}^F; \boldsymbol{\theta}] - \partial_{\boldsymbol{\theta}} \mathscr{E}[\mathbf{p}^C; \boldsymbol{\theta}]$

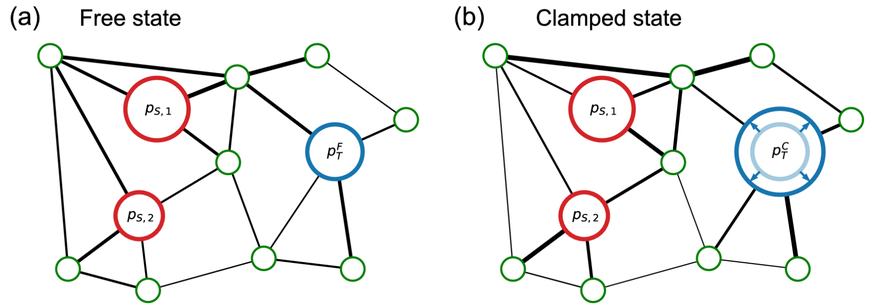
Equilibrium propagation

Open Access

Supervised Learning in Physical Networks: From Machine Learning to Learning Machines

Menachem Stern, Daniel Hexner, Jason W. Rocks, and Andrea J. Liu
Phys. Rev. X **11**, 021045 – Published 28 May 2021

- Equilibrium propagation can be effectively employed to train physical system
  - Networks of springs, resistors, hydraulics
- Can make a spring network do classification tasks (and have been physically implemented)



$$\mathcal{P} = \frac{1}{2}\sum_j k_j \Delta p_j^2,$$

$$\dot{k}_j = \alpha\eta^{-1}\frac{\partial}{\partial k_j}\{\mathcal{P}^F - \mathcal{P}^C\}$$

$$= \frac{1}{2}\alpha\eta^{-1}\{[\Delta p_j^F]^2 - [\Delta p_j^C]^2\},$$