Differential Programming

Miles Cranmer (Graduate Student, Princeton Astro)

Acknowledgements

- Baydin et al. (2018), Domke (2009), Olah (2015) were helpful guides in putting this together, as was the JAX documentation.

Differentiation

- Four categories of computing derivatives on computers:
  1. Hard-coding the analytical form (by hand)
  2. Numerical differentiation (approximate)
  3. Symbolic differentiation via computer algebra system (e.g., SymPy, Mathematica)
  4. **Automatic differentiation (AD)**

Misconceptions

Symbolic differentiation != Automatic differentiation

Symbolic differentiation can differentiate **expressions**, but
Automatic differentiation can differentiate **entire algorithms**!

Key Idea

- If you symbolically differentiated an algorithm, it would lead to an intractably large expression!

- So, what's the trick?
  - **You trace the original algorithm, carry along the numerical values for the derivatives, and repeatedly apply chain rule.**

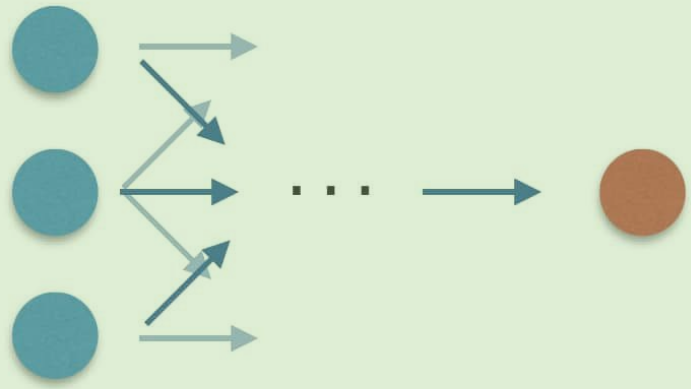
Forward/Reverse mode

- Forward:
  - Walk through the algorithm, and carry derivatives through, at the same time.
- Reverse:
  - First, evaluate the output, while recording the computational tree.
  - Then, walk back through the tree and compute derivatives.

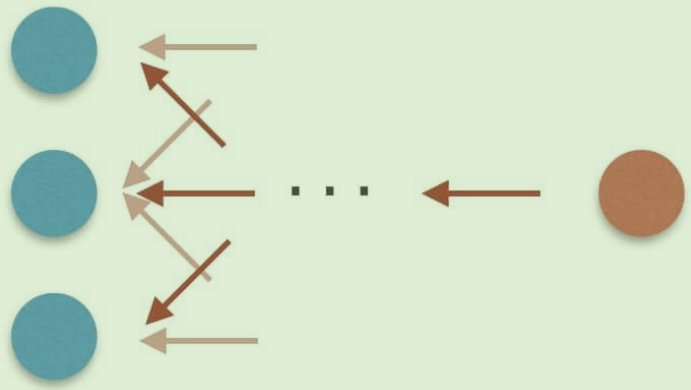
$f : \mathbb{R}^n \to \mathbb{R}^m$ **w.** $n >> m$

Forward 👎 Reverse 👍

Forward Pass



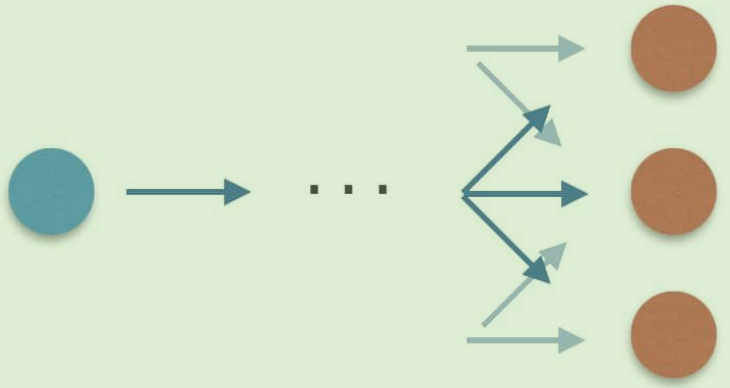
Backward Pass



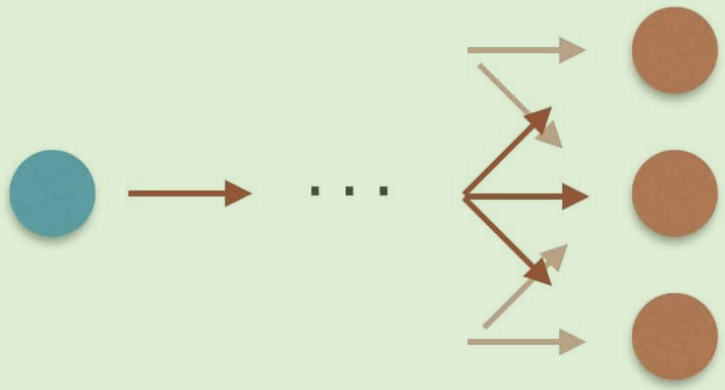
$f : \mathbb{R}^n \to \mathbb{R}^m$ **w.** $n << m$

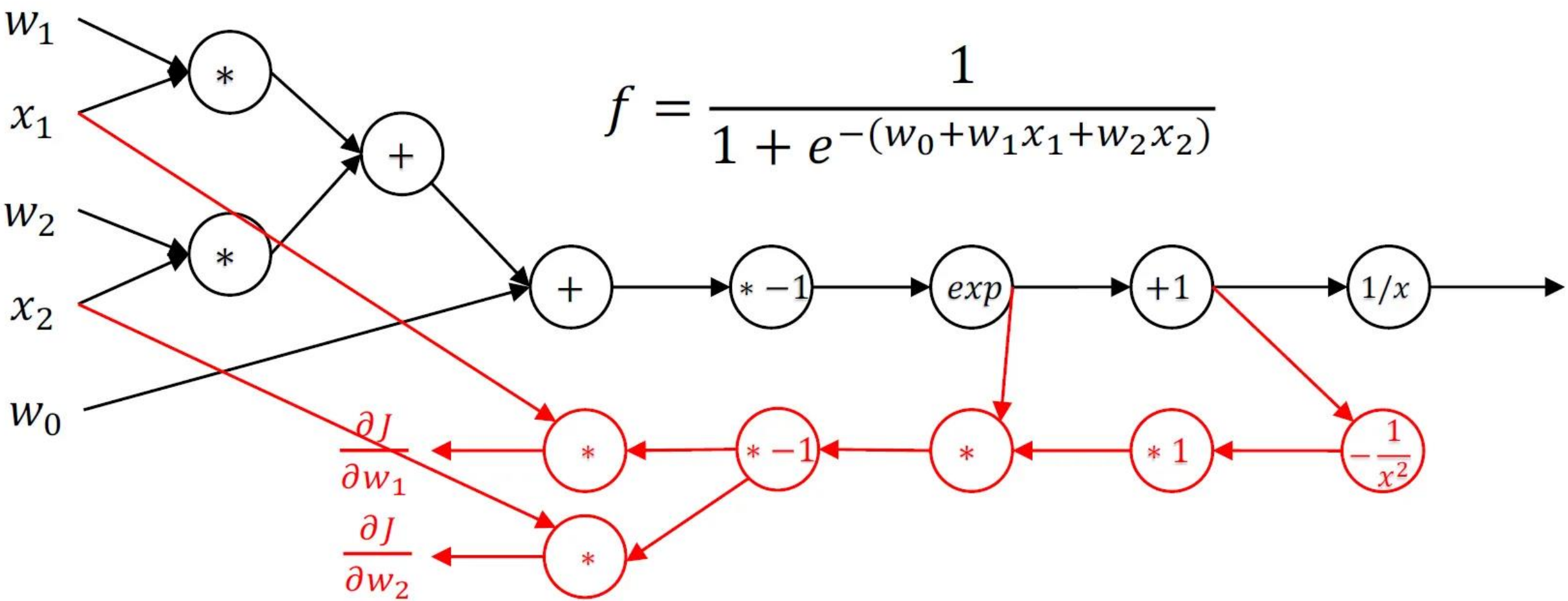
Forward 👍 Reverse 👎

Primal Trace



Derivative Trace



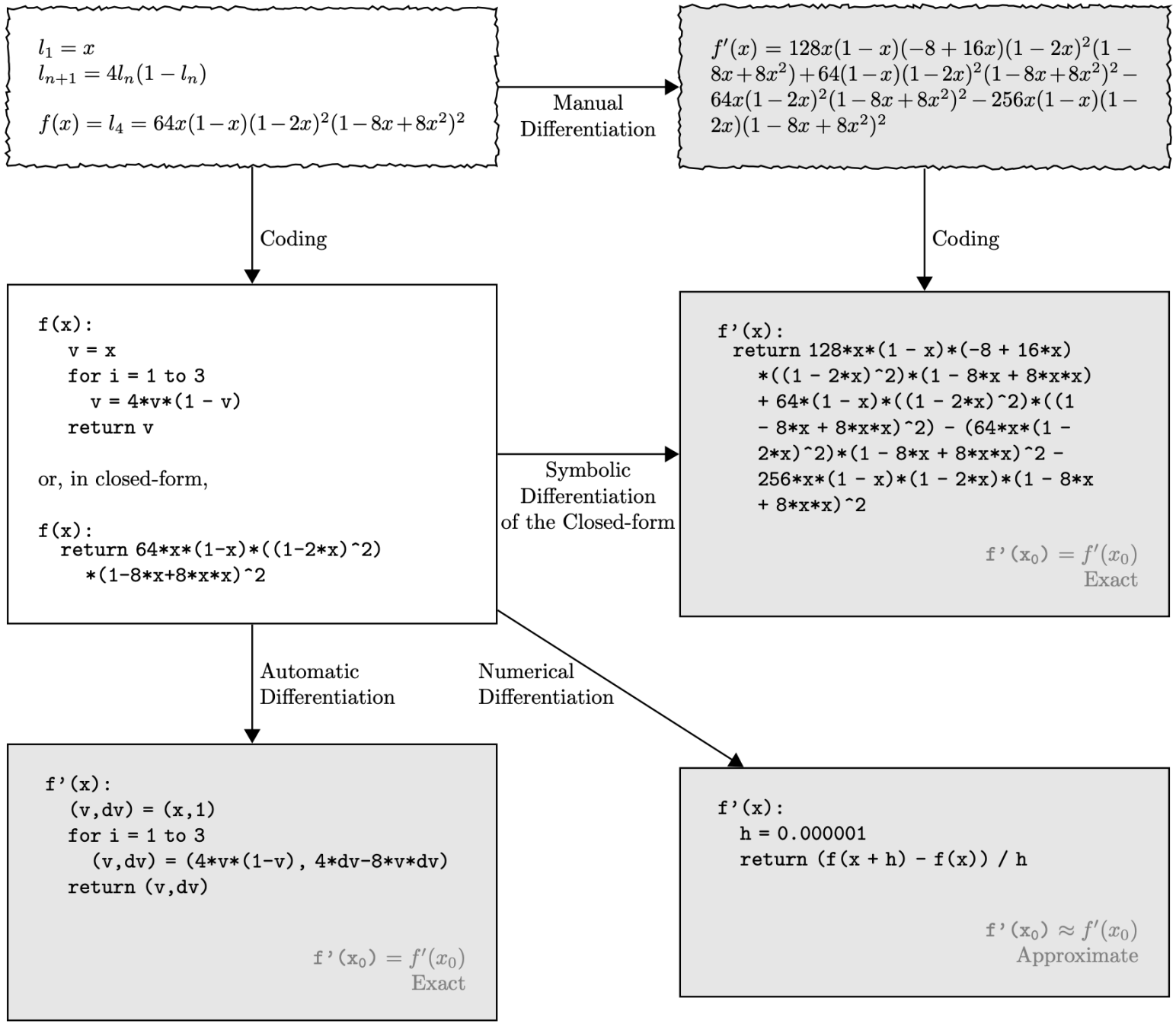

w_1
x_1
w_2
x_2
w_0
$*$
$+$
$*$
$+$
$* -1$
exp
$+1$
$1/x$
$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$
$\frac{\partial J}{\partial w_1}$
$\frac{\partial J}{\partial w_2}$
$*$
$* -1$
$*$
$* 1$
$-\frac{1}{x^2}$
$*$

Key Idea

So, we do not have the closed-form analytic formula for the derivative.

But:

- We can compute the derivative **exactly** at any point.
- We don't have to write extra code to do this.
- We can compute derivatives through massively complicated algorithms and pipelines!



Baydin et al. (2018)

Applications

- Optimization in high-dimensional spaces
- Hamiltonian MCMC
- Likelihood-free inference
- Machine learning
- **ML with inductive biases**
  - (learning internal components of a handwritten pipeline)

Differential programming

Writing code in a framework that supports autodiff.

All the code you write now has derivatives for free!

The revelation:

if you don't know what to put for some part of the algorithm, just **learn it.**

This is differential programming.

Packages available

- Python
  - JAX, PyTorch, TensorFlow
- Julia:
  - (built-in)
- C++:
  - “autodiff”
  - https://github.com/autodiff/autodiff

PyTorch (DL focused, dynamic)

- Pure Python is allowed!
- Very good for differential programming, due to great Deep Learning functionality.
  - Very easy to plug in a neural network into your algorithm when you don't know what it should be!
- Drawbacks: not compiled.

```
x = torch.randn(100, 10)
x.requires_grad = True
x = x.cuda()
torch.autograd.grad(f(x), x)
```

JAX

- Compiled!
- Very generic: no emphasis on ML
- Numpy syntax: can do “from jax import numpy as np”
- Can vectorize any operation with “vmap”
- GPUs utilized automatically

```
key = random.PRNGKey(0)
x = random.normal(key, (100, 5))
df = grad(f)
df(x)
```

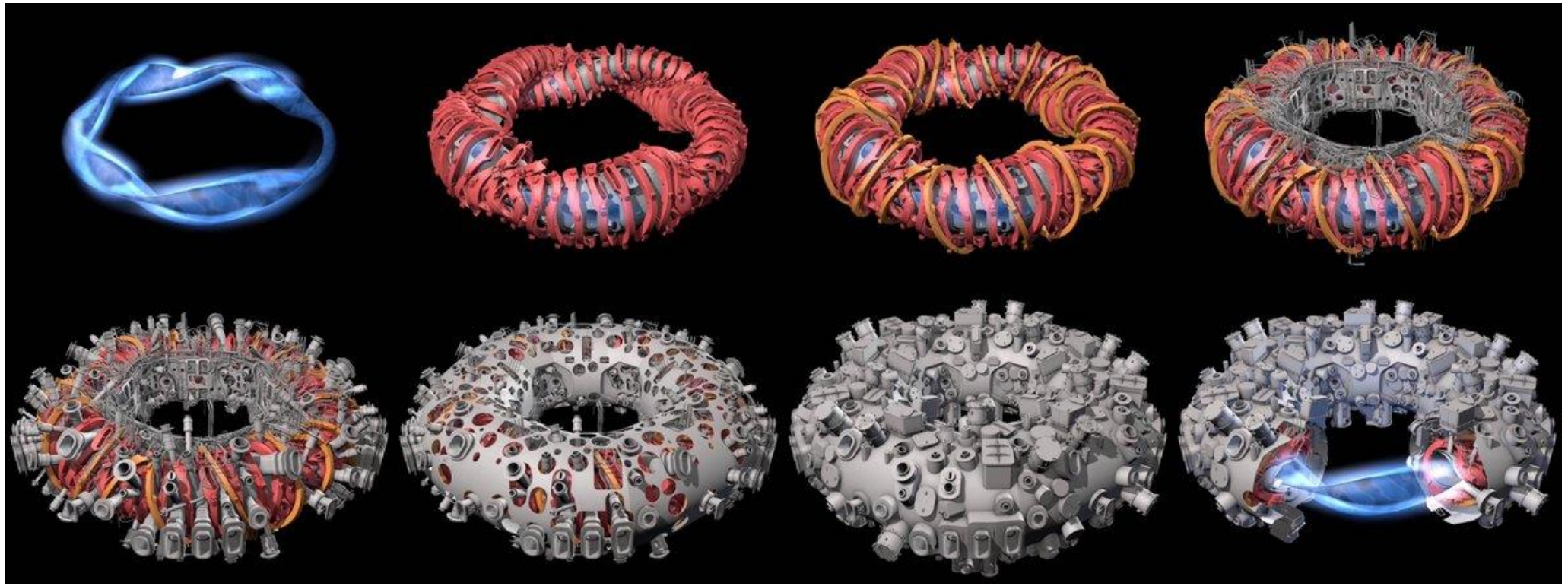
Julia

- Compiled.
- Autodiff works on base language! No need for new library.
- Weaknesses:
  - GPU support seems a bit more work than Torch/JAX.
  - Deep Learning (Flux.jl) not nearly as large a community as Python

```
x = randn(Float32, (100, 5))
ReverseDiff.gradient(f, x)
```

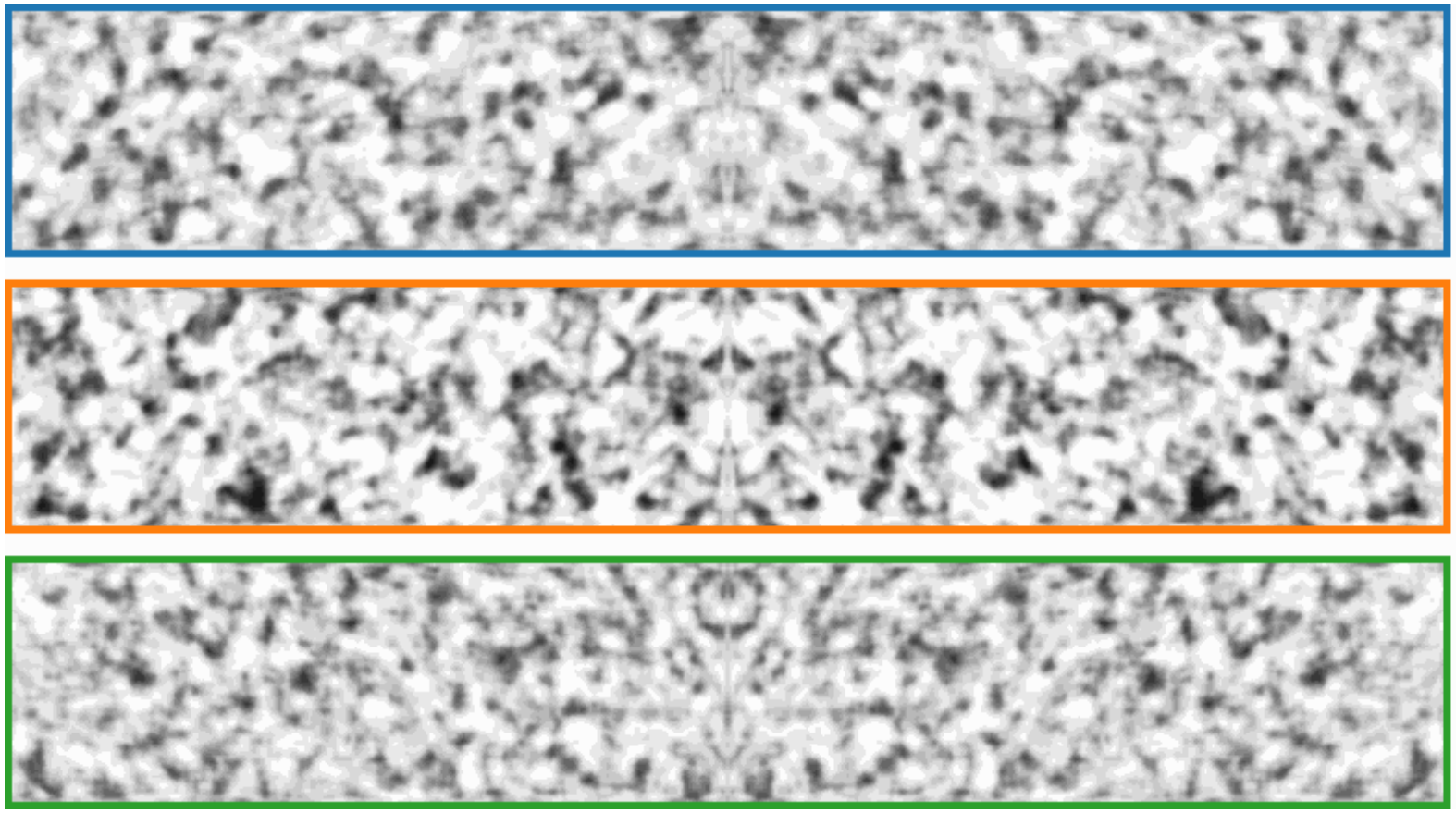
Applications

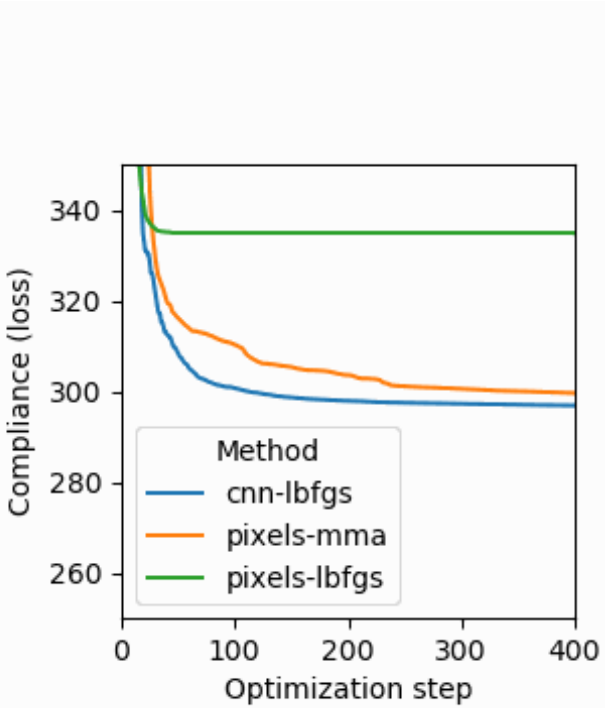
Stellerator (plasma fusion) – here, they optimize the coil topology with autodiff! (McGreivy et al. 2020)



Applications

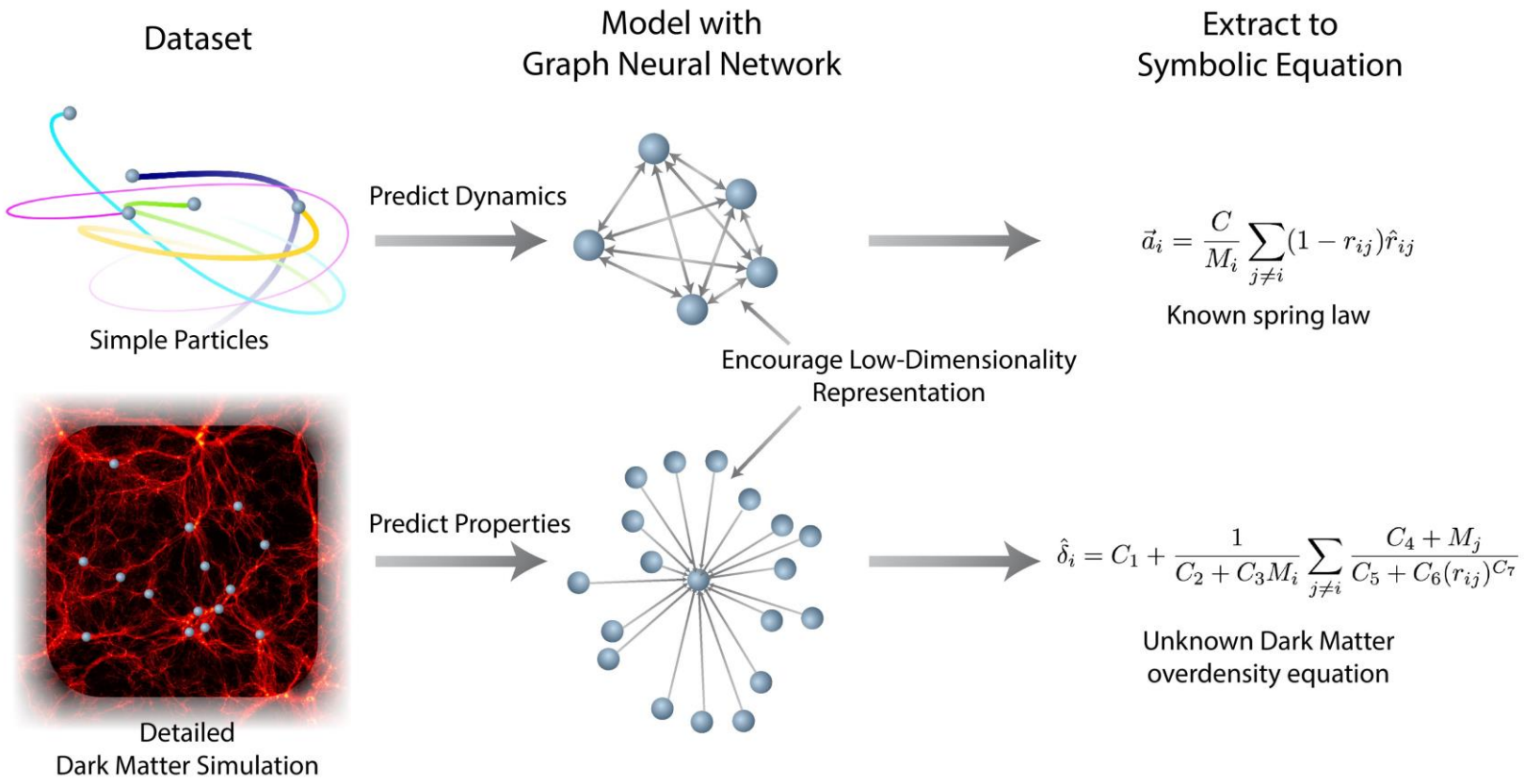
Bridge optimization in raw density space:





Applications

- (Future talk on this) – autodiff + symbolic regression to discover force laws, Hamiltonians, new dark matter equation:



Questions?

Additional References

- https://towardsdatascience.com/forward-mode-automatic-differentiation-dual-numbers-8f47351064bf