Taylor AD for Higher-Order Derivatives in JAX

VECTOR INSTITUTE
UNIVERSITY OF TORONTO

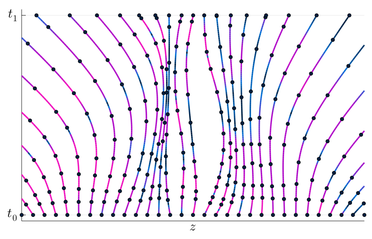
Jesse Bettencourt, Jacob Kelly, Matthew Johnson, and David Duvenaud

Motivation: Regularizing Neural ODEs

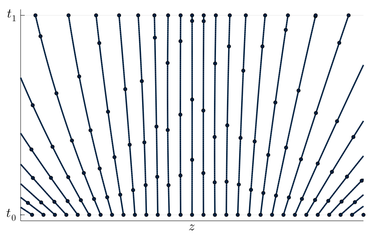
We want to learn ODEs that are easy to solve.
Optimize higher order properties of the ODE solution:

$$\nabla_\theta \int_0^T \left\| \frac{d^k z(t)}{dt^k} \right\|_2^2 dt$$

`grad odeint norm jet odeint nn`



Learned dynamics are unnecessarily complex and require many function evaluations to solve.



Dynamics with third derivative regularized require fewer evaluations to solve.

Kelly et al. "Learning Differential Equations that are Easy to Solve" arXiv preprint (2020).

https://github.com/jacobjinkelly/easy-neural-ode

TL;DR: Don't Nest First Order!

If you want higher order derivatives like

$$\frac{\partial^k f(x)}{\partial x^k} v^k$$

the naïve approach to nest first order derivatives

$$\texttt{deriv}(\cdots(\texttt{deriv}(f)))(x)(v)$$

may work with nice AD (like JAX)
but will scale exponentially in the order of differentiation

$$O(\exp(D))$$

because nesting first order does not share common work.

First-Order Derivatives

Given a composite function $f : \mathbb{R}^n \to \mathbb{R}^m$

$$f = (g \circ h)(x) = g(h(x))$$

input representing primal output

$$z_0 = h(x)$$

and first-order perturbation in direction v of $h(x)$

$$z_1 = \partial h(x)[v]$$

First-order AD computes primal output and first-order perturbation in direction v of $f(x)$

$$(y_0, y_1) = (f(x), \partial f(x)[v])$$

Where chain-rule gives

$$\partial f(x)[v] = \partial g(h(x)) * \partial h(x)[v] = \partial g(z_0)[z_1]$$

Implicit Jacobian $\in \mathbb{R}^{m \times n}$ vector $v \in \mathbb{R}^n$ product:

$$\partial f(\cdot)[v] = \frac{\partial f(\cdot)}{\partial \cdot}[v] \in \mathbb{R}^m$$

Higher-Order Derivatives

Inputs representing higher-order perturbations in direction v of $h(x)$

$$(z_0, \ldots, z_D) = (h(x), \ldots, \partial^D h(x)[v^D])$$

Higher-order perturbations in direction v of $f(x)$

$$(y_0, \ldots, y_D) = (f(x), \ldots, \partial^D f(x)[v^D])$$

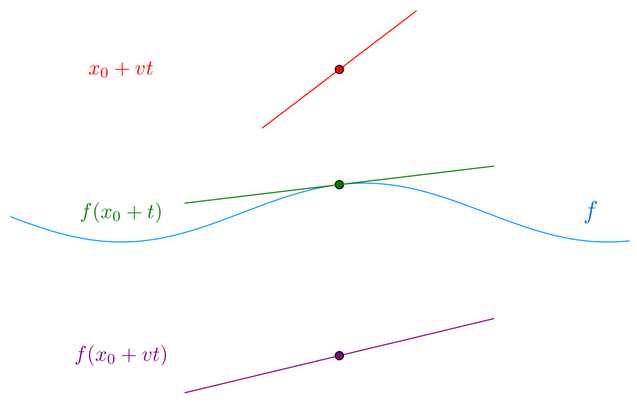
Implicit Jacobian $\in \mathbb{R}^{m \times n \times \cdots \times n}$ vector $v \in \mathbb{R}^n$ product:

$$\partial^k f(\cdot)[v^k] = \frac{\partial^k f(\cdot)}{\partial (\cdot)^k}[v] \cdots [v] \in \mathbb{R}^m$$

Higher derivatives have more complicated dependence on lower order perturbations than chain rule.
e.g. $k = 2$

$$y_2 = \partial^2 f(x)[v^2] = \underbrace{\partial g(z_0)[z_2]}_{\text{1st-order of } g \text{ on 2nd-order of } h} + \underbrace{\partial^2 g(z_0)[z_1^2]}_{\text{2nd-order of } g \text{ on 1st-order of } h}$$

First-order Automatic Differentiation



With primal value x_0 and first order perturbation,

$$v = \frac{\partial x(t)}{\partial t}$$

We have the expansion

$$x(t) = x_0 + vt$$

Given f and $\partial f(x_0)$, compute

$$f(x(t)) = y_0 + y_1 t$$

With coefficients

$$y_0 = f(x_0)$$
$$y_1 = \partial f(x_0)[v]$$

Jacobian-Vector Product `JAX` API

$$\texttt{jax.jvp}(f, x_0, v)$$

Taylor Polynomial Derivative Rules

Implemented by overloading primitives derivatives.
Interpreted as evaluating on polynomial inputs:
(Evaluating Derivatives. Griewank and Walther. 2008)

$$u = u_0 + u_1 t + \frac{1}{2!} u_2 t^2 + \cdots + \frac{1}{D!} u_D t^D \in \mathbb{R}^n$$
$$w = w_0 + w_1 t + \frac{1}{2!} w_2 t^2 + \cdots + \frac{1}{D!} w_D t^D \in \mathbb{R}^n$$

Addition
Primitive: $v = u + cw$
Derivatives: $v_k = u_k + cw_k$

Multiplication
Primitive: $v = u * w$
Derivatives: $v_k = \sum_{j=0}^{k} u_j w_{k-j}$

$v_0 = u_0 * w_0$
$v_1 = u_0 * w_1 + u_1 * w_0$ (familiar product rule)
$v_2 = u_0 * w_2 + 2 * u_1 * w_1 + u_2 * w_0$

Division
Primitive: $v = u/w$
Derivatives: $v_k = \frac{1}{w_0}\left(u_k - \sum_{j=0}^{k-1} v_j w_{k-j}\right)$

Exp
Primitive: $v = \exp(u)$
Derivatives: $v_k = \frac{1}{k}\sum_{j=1}^{k} j u_j v_{k-j}$

Sin
Primitive: $s = \sin(u)$
Derivatives: $s_k = \frac{1}{k}\sum_{j=1}^{k} j u_j c_{k-j}$

Cos
Primitive: $c = \cos(u)$
Derivatives: $c_k = \frac{1}{k}\sum_{j=1}^{k} -j u_j s_{k-j}$

Linear and Bi-Linear Jets

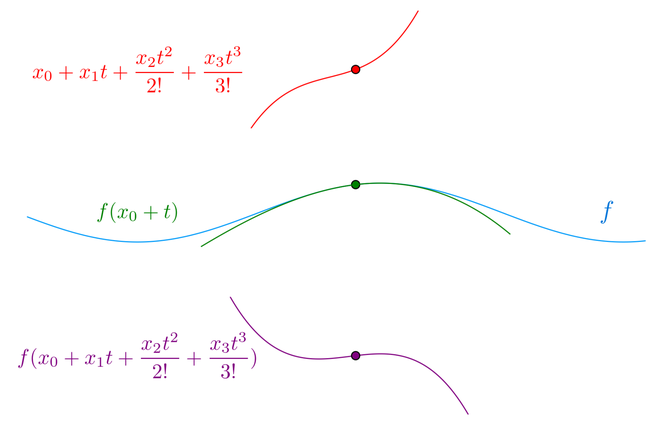
Addition is a special case of **Linear** $v_k = f(u_k, w_k)$

```
def linear_jet(f, primals, series, **params):
  y_0 = f(*primals, **params)
  y_s = [f(*coeffs, **params) for coeffs in series]
  return y_0, y_s
```

Multiply of **Bi-Linear** $v_k = \sum_{j=0}^{k} f(u_j, w_{k-j})$

```
def bilinear_jet(f, primals, series, **params):
  y_0 = f(*primals, **params)
  u, w = zip(primals + series)
  y_s = conv(f,u,w,0,k, **params)
  return y_0, y_s
```

Higher-order Automatic Differentiation



Primal value x_0 and higher order perturbations,

$$x_k = \frac{\partial^k x(t)}{\partial t^k}$$

We have the expansion

$$x(t) = x_0 + x_1 t + \frac{x_2 t^2}{2!} + \cdots + \frac{x_D t^D}{D!}$$

Given f and $\partial^k f(x_0)$, compute

$$y(t) = y_0 + y_1 t + \frac{y_2 t^2}{2!} + \cdots + \frac{y_D t^D}{D!}$$

With coefficients

$$y_0 = f(x_0)$$
$$y_1 = \partial f(x_0)[x_1]$$
$$y_2 = \partial f(x_0)[x_2] + \partial^2 f(x_0)[x_1^2]$$
$$y_3 = \partial f(x_0)[x_3] + 3\partial^2 f(x_0)[x_1 x_2] + \partial^3 f(x_0)[x_1^3]$$

Taylor-Mode `JAX` API

$$\texttt{jax.jet}(f, x_0, [v, 0, \ldots, 0])$$

or more generally

$$\texttt{jax.jet}(f, x_0, [x_1, x_2, \ldots, x_D])$$

Performance Taylor v.s. Nesting

Naïve Nesting First Order: $O(\exp D)$
Taylor Mode: $O(D \log D)$

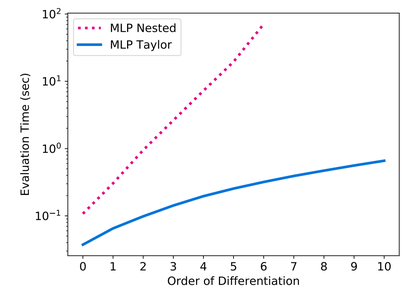


Figure: Scaling for higher derivatives of 2-layer MLP.

Relationship Between odeint and Taylor

ODEs are specified by first derivative

$$\frac{dz(t)}{dt} = f_\theta(z(t), t)$$

A nice recursive relationship in the Taylor expansion $z(t)$

$$z(t) \approx z_0 + \frac{dz(t)}{dt} t + \cdots \approx z_0 + f_\theta(z(t), t)t + \cdots$$

`odeint` as a primitive can use this
Primitive: $z = \texttt{odeint}(f, z_0)$
Derivatives: $z\_k = \texttt{jet}(f, z\_0, (z\_1, \ldots, z\_k - 1))$

Taylor AD in Julia?

`TaylorSeries.jl` is excellent. However, highly-mutating and scalarized, difficult to compose with reverse mode gradients.
`TaylorIntegration.jl` implements jets of ODE solutions.
`PyCall.jl` works great with JAX, and will be getting even easier!