Automatic Differentation

Charles Margossian

Given a program to evaluate

$$\begin{aligned} f &: \mathbb{R}^D \to \mathbb{R} \\ &: \mathbf{x} \to f(\mathbf{x}) \end{aligned}$$

want to evaluate the gradient $\nabla f(\mathbf{x})$ for a value of $\mathbf{x}$.

Given a program to evaluate

$$\begin{aligned} f &: \mathbb{R}^D \to \mathbb{R} \\ &: \mathbf{x} \to f(\mathbf{x}) \end{aligned}$$

want to evaluate the gradient $\nabla f(\mathbf{x})$ for a value of $\mathbf{x}$.

f can be:

- an objective function (optimization)
- a probability density (sampling)

Automatic differentiation is a framework for numerically evaluating derivatives of a program.

Automatic differentiation is a framework for numerically evaluating derivatives of a program.

Autodiff libraries:

- JAX
- TensorFlow
- Stan-math
- JuliaAD
- Pyro
- $\cdots$

Automatic differentiation is a framework for numerically evaluating derivatives of a program.

Autodiff libraries:

- JAX $\longrightarrow$ TensorFlow Probability, PyMC, FlowMC
- TensorFlow $\longrightarrow$ TensorFlow Probability
- Stan-math $\longrightarrow$ Stan
- JuliaAD $\longrightarrow$ Turing
- Pyro $\longrightarrow$ PyTorch
- $\cdots$

What came "before" autodiff?

- hand-coded gradients
- finite differentiation

$$\frac{\partial f(\mathbf{x})}{\partial x_1} \approx \frac{f(x_1 + \epsilon, x_2, \cdots, x_D) - f(x_1 - \epsilon, x_2, \cdots, x_D)}{2\epsilon}.$$

- symbolic differentiation

Outline:

- Forward and reverse mode automatic differentiation
- Higher-oder differentiation
- Example: adjoint-differentiated Laplace
- Implicit functions

Autodiff uses the chain rule.

Autodiff uses the chain rule.

Consider the decomposition of f,

$$f = f_L \circ f_{L-1} \circ \cdots f_2 \circ f_1(\mathbf{x})$$

Autodiff uses the chain rule.

Consider the decomposition of f,

$$f = f_L \circ f_{L-1} \circ \cdots f_2 \circ f_1(\mathbf{x})$$

The Jacobian is then

$$J = J_L \cdot J_{L-1} \cdot \cdots \cdot J_1.$$

Autodiff uses the chain rule.

Consider the decomposition of f,

$$f = f_L \circ f_{L-1} \circ \cdots f_2 \circ f_1(\mathbf{x})$$

The Jacobian is then

$$J = J_L \cdot J_{L-1} \cdot \cdots \cdot J_1.$$

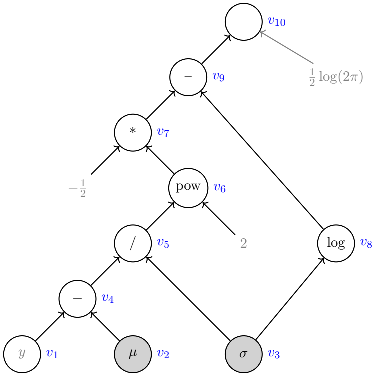
First-order intuition: an autodiff library associates to each operator

$$\texttt{f}_\ell : y \to f_\ell(y),$$

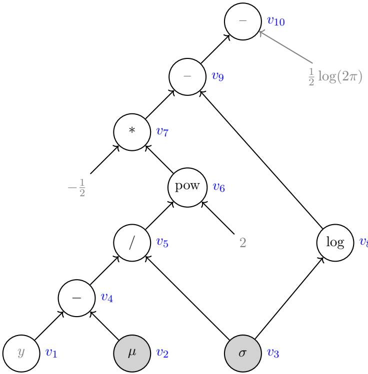
a corresponding differentiation operator

$$\texttt{df}_\ell : y \to J_\ell(y).$$

$$f(y, \mu, \sigma) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi)$$



$$f(y, \mu, \sigma) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi)$$

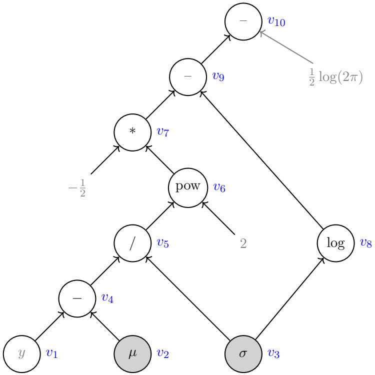


This can be written as a sequence of maps:

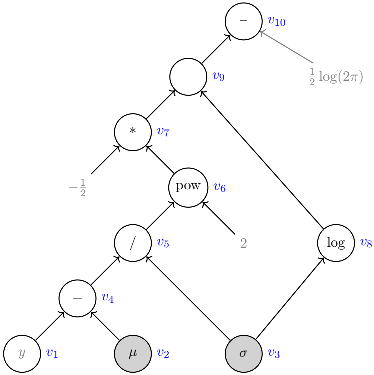
$$\begin{aligned}(y,\ \mu,\ \sigma) &\to (y-\mu,\ \sigma)\\ &\to \left(\frac{y-\mu}{\sigma},\ \sigma\right)\\ &\to \left(\frac{y-\mu}{\sigma},\ \log(\sigma)\right)\\ &\ldots\end{aligned}$$

In practice, operators (and difference operators) are only defined locally.

$$f(y, \mu, \sigma) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi)$$



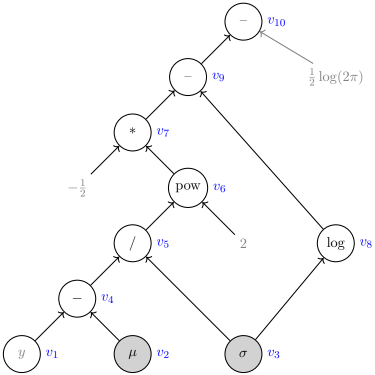
$$f(y, \mu, \sigma) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi)$$



Forward evaluation trace

$v_1 = y = 10$
$v_2 = \mu = 5$
$v_3 = \sigma = 2$
$v_4 = v_1 - v_2 = 5$
$v_5 = v_4 / v_3 = 2.5$
$v_6 = v_5^2 = 6.25$
$v_7 = -0.5 \times v_6 = 3.125$
$v_8 = \log(v_3) = \log(2)$
$v_9 = v_7 - v_8 = 3.125 - \log(2)$
$v_{10} = v_9 - 0.5\log(2\pi) = 3.125 - \log(4\pi)$

$$f(y,\mu,\sigma) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi)$$



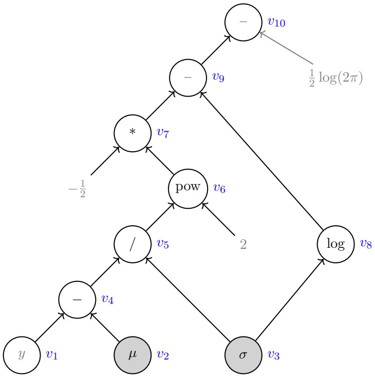
Forward evaluation trace

$v_1 = y = 10$
$v_2 = \mu = 5$
$v_3 = \sigma = 2$
$v_4 = v_1 - v_2 = 5$
$v_5 = v_4/v_3 = 2.5$
$v_6 = v_5^2 = 6.25$
$v_7 = -0.5 \times v_6 = 3.125$
$v_8 = \log(v_3) = \log(2)$
$v_9 = v_7 - v_8 = 3.125 - \log(2)$
$v_{10} = v_9 - 0.5\log(2\pi) = 3.125 - \log(4\pi)$

Forward derivative trace

$\dot{v}_1 = 0$
$\dot{v}_2 = 1$
$\dot{v}_3 = 0$
$\dot{v}_4 = \frac{\partial v_4}{\partial v_1}\dot{v}_1 + \frac{\partial v_4}{\partial v_2}\dot{v}_2 = 0 + (-1) \times 1 = -1$
$\dot{v}_5 = \frac{\partial v_5}{\partial v_4}\dot{v}_4 + \frac{\partial v_5}{\partial v_3}\dot{v}_3 = \frac{1}{v_3} \times (-1) = -0.5$
$\dot{v}_6 = \frac{\partial v_6}{\partial v_5}\dot{v}_5 = 2v_5 \times (-0.5) = -2.5$
$\dot{v}_7 = \frac{\partial v_7}{\partial v_6}\dot{v}_6 = -0.5 \times (-2.5) = 1.25$
$\dot{v}_8 = \frac{\partial v_8}{\partial v_3}\dot{v}_3 = 0$
$\dot{v}_9 = \frac{\partial v_9}{\partial v_7}\dot{v}_7 + \frac{\partial v_9}{\partial v_8}\dot{v}_8 = 1 \times 1.25 + 0 = 1.25$
$\dot{v}_{10} = \frac{\partial v_{10}}{\partial v_9}\dot{v}_9 = 1.25$

$$f(y, \mu, \sigma) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi)$$



Forward evaluation trace

$v_1 = y = 10$
$v_2 = \mu = 5$
$v_3 = \sigma = 2$
$v_4 = v_1 - v_2 = 5$
$v_5 = v_4 / v_3 = 2.5$
$v_6 = v_5^2 = 6.25$
$v_7 = -0.5 \times v_6 = 3.125$
$v_8 = \log(v_3) = \log(2)$
$v_9 = v_7 - v_8 = 3.125 - \log(2)$
$v_{10} = v_9 - 0.5\log(2\pi) = 3.125 - \log(4\pi)$

Reverse adjoint trace

$\bar{v}_{10} = 1$
$\bar{v}_9 = \frac{\partial v_{10}}{\partial v_9}\bar{v}_{10} = 1 \times 1 = 1$
$\bar{v}_8 = \frac{\partial v_9}{\partial v_8}\bar{v}_9 = (-1) \times 1 = -1$
$\bar{v}_7 = \frac{\partial v_9}{\partial v_7}\bar{v}_9 = 1 \times 1 = 1$
$\bar{v}_6 = \frac{\partial v_7}{\partial v_6}\bar{v}_7 = (-0.5) \times 1 = -0.5$
$\bar{v}_5 = \frac{\partial v_6}{\partial v_5}\bar{v}_6 = 2v_5 \times \bar{v}_6 = -2.5$
$\bar{v}_4 = \frac{\partial v_5}{\partial v_4}\bar{v}_5 = \frac{1}{v_3} \times (-2.5) = -1.25$
$\bar{v}_3 = \frac{\partial v_5}{\partial v_3}\bar{v}_5 + \frac{\partial v_8}{\partial v_3}\bar{v}_8 = 2.625$
$\bar{v}_2 = \frac{\partial v_4}{\partial v_2}\dot{v}_4 = (-1) \times (-1.25) = 1.25$

The differentiation operator $\texttt{df}_\ell$ does <u>not</u> compute J_ℓ, rather it returns a directional derivative.

The differentiation operator $\texttt{df}_\ell$ does <u>not</u> compute J_ℓ, rather it returns a directional derivative.

In *forward* mode, it computes

$$J_\ell \cdot u_{\ell-1}.$$

The differentiation operator $\texttt{df}_\ell$ does <u>not</u> compute J_ℓ, rather it returns a directional derivative.

In *forward* mode, it computes

$$J_\ell \cdot u_{\ell-1}.$$

Given an initial tangent, u_0, a forward sweep returns

$$\begin{aligned} J \cdot u_0 &= J_L \cdot u_{L-1} \\ &= J_L \cdot J_{L-1} \cdot u_{L-2} \\ &= \cdots \\ &= J_L \cdot J_{L-1} \cdots J_1 \cdot u_0. \end{aligned}$$

The differentiation operator $\texttt{df}_\ell$ does <u>not</u> compute J_ℓ, rather it returns a directional derivative.

In *forward* mode, it computes

$$J_\ell \cdot u_{\ell-1}.$$

Given an initial tangent, u_0, a forward sweep returns

$$\begin{aligned} J \cdot u_0 &= J_L \cdot u_{L-1} \\ &= J_L \cdot J_{L-1} \cdot u_{L-2} \\ &= \cdots \\ &= J_L \cdot J_{L-1} \cdots J_1 \cdot u_0. \end{aligned}$$

In our example, computing J (the gradient) requires two forward sweeps with

- $u_0 = (0, 1, 0)$
- $u_0 = (0, 0, 1)$.

In *reverse* mode, the differentiation operator returns

$$J_\ell^\dagger \cdot w_\ell.$$

In *reverse* mode, the differentiation operator returns

$$J_\ell^\dagger \cdot w_\ell.$$

Given an initial cotangent, w_L, a reverse sweep returns

$$\begin{aligned} J_1^\dagger \cdot w_1 &= J_1^\dagger \cdot J_2^\dagger \cdot w_2 \\ &= \cdots \\ &= J_1^\dagger \cdot J_2^\dagger \cdots J_L^\dagger \cdot w_L. \end{aligned}$$

In *reverse* mode, the differentiation operator returns

$$J_\ell^\dagger \cdot w_\ell.$$

Given an initial cotangent, w_L, a reverse sweep returns

$$\begin{aligned} J_1^\dagger \cdot w_1 &= J_1^\dagger \cdot J_2^\dagger \cdot w_2 \\ &= \cdots \\ &= J_1^\dagger \cdot J_2^\dagger \cdots J_L^\dagger \cdot w_L. \end{aligned}$$

In our example, computing J (the gradient) requires one reverse sweep with

- $w_L = (1)$

How many sweeps of autodiff would we need to compute a full Jacobian of

$$f : \mathbb{R}^N \to \mathbb{R}^M?$$

Outline:

- Forward and reverse mode automatic differentiation
- Higher-oder differentiation
- Example: adjoint-differentiated Laplace
- Implicit functions

Suppose we want to compute a Hessian-vector product,

$$H \cdot v = \nabla^2 f \cdot v.$$

How many sweeps of autodiff would we need?

Suppose we want to compute a Hessian-vector product,

$$H \cdot v = \nabla^2 f \cdot v.$$

How many sweeps of autodiff would we need?

Need to find the right decomposition:

$$H \cdot v = \underbrace{[\nabla(\underbrace{\nabla f \cdot v}_{\text{forward}})]^\dagger}_{\text{reverse}} \cdot \mathbf{1}\,.$$

Suppose now we want to compute a diagonal Hessian.

Suppose now we want to compute a diagonal Hessian.

Use:

$$\mathrm{diag}(H) = \underbrace{[\nabla(\underbrace{\nabla f \cdot u}_{\text{forward}})]^\dagger \cdot \mathbf{1}}_{\text{reverse}},$$

where

$$u = (1, 1, \cdots, 1).$$

Suppose now we want to compute a diagonal Hessian.

Use:

$$\mathrm{diag}(H) = \underbrace{[\nabla(\underbrace{\nabla f \cdot u}_{\text{forward}})]^{\dagger} \cdot \mathbf{1}}_{\text{reverse}},$$

where

$$u = (1, 1, \cdots, 1).$$

What if we had a block-diagonal Hessian?

Outline:

- Forward and reverse mode automatic differentiation
- Higher-oder differentiation
- Example: adjoint-differentiated Laplace
- Implicit functions

Example: embedded Laplace approximation

Consider the Gaussian process

$$\begin{aligned} \phi &\sim \pi(\phi) \\ \boldsymbol{\theta} \mid \phi &\sim \text{normal}(0, K(\phi)) \\ y_i \mid \phi, \boldsymbol{\theta} &\sim \pi(y_i \mid \theta_i, \phi). \end{aligned}$$

Example: embedded Laplace approximation

Consider the Gaussian process

$$\begin{aligned} \phi &\sim \pi(\phi) \\ \boldsymbol{\theta} \mid \phi &\sim \text{normal}(0, K(\phi)) \\ y_i \mid \phi, \boldsymbol{\theta} &\sim \pi(y_i \mid \theta_i, \phi). \end{aligned}$$

Compute marginal distribution

$$\pi(\phi \mid \mathbf{y}) = \int_\Theta \pi(\phi, \boldsymbol{\theta} \mid \mathbf{y}) \mathrm{d}\boldsymbol{\theta},$$

and the gradient

$$\nabla_\phi \log \pi(\phi \mid \mathbf{y}).$$

From Rasmussen and Williams (2006),

Algorithm

input: $\mathbf{y}$, ϕ, $\pi(y \mid \theta, \phi)$
saved input from Newton solver: θ^*, K, $W^{\frac{1}{2}}$, L, a
$Z = \frac{1}{2} a^T \theta^* + \log \pi(\mathbf{y} \mid \theta^*, \phi) - \sum \log(\text{diag}(L))$
$R = W^{\frac{1}{2}} L^T \setminus (L \setminus W^{\frac{1}{2}})$
$C = L \setminus (W^{\frac{1}{2}} K)$
$s_2 = -\frac{1}{2} \text{diag}(\text{diag}(K) - \text{diag}(C^T C)) \nabla_\theta^3 \log \pi(\mathbf{y} \mid \theta^*, \phi)$
for $j = 1 \ \dots \ \dim(\phi)$
  $K' = \partial K / \partial \phi_j$
  $s_1 = \frac{1}{2} a^T K' a - \frac{1}{2} \text{tr}(R K')$
  $b = K' \nabla_\theta \log \pi(\mathbf{y} \mid \theta, \phi)$
  $s_3 = b - K R b$
  $\frac{\partial}{\partial \phi_j} \pi(\mathbf{y} \mid \phi) = s_1 + s_2^T s_3$
end for
return $\nabla_\phi \log \pi_{\mathcal{G}}(\mathbf{y} \mid \phi)$

From Rasmussen and Williams (2006),

Algorithm

input: $\mathbf{y}$, ϕ, $\pi(y \mid \theta, \phi)$
saved input from Newton solver: θ^*, K, $W^{\frac{1}{2}}$, L, a
$Z = \frac{1}{2} a^T \theta^* + \log \pi(\mathbf{y} \mid \theta^*, \phi) - \sum \log(\text{diag}(L))$
$R = W^{\frac{1}{2}} L^T \setminus (L \setminus W^{\frac{1}{2}})$
$C = L \setminus (W^{\frac{1}{2}} K)$
$s_2 = -\frac{1}{2} \text{diag}(\text{diag}(K) - \text{diag}(C^T C)) \nabla_\theta^3 \log \pi(\mathbf{y} \mid \theta^*, \phi)$
for $j = 1 \ \dots \ \dim(\phi)$
$\quad K' = \partial K / \partial \phi_j$
$\quad s_1 = \frac{1}{2} a^T K' a - \frac{1}{2} \text{tr}(R K')$
$\quad b = K' \nabla_\theta \log \pi(\mathbf{y} \mid \theta, \phi)$
$\quad s_3 = b - K R b$
$\quad \frac{\partial}{\partial \phi_j} \pi(\mathbf{y} \mid \phi) = s_1 + s_2^T s_3$
end for
return $\nabla_\phi \log \pi_{\mathcal{G}}(\mathbf{y} \mid \phi)$

A first-order read of the chain rule prescribes using autodiff to compute

$$K' = \partial dK/\partial\phi,$$

and plugging it back in the Algorithm.

A first-order read of the chain rule prescribes using autodiff to compute

$$K' = \partial dK / \partial \phi,$$

and plugging it back in the Algorithm.

For $\phi \in \mathbb{R}^D$ and $K \in \mathbb{R}^{N \times N}$, need

- D forward mode sweeps
- $\mathcal{O}(N^2)$ reverse mode sweeps.

A first-order read of the chain rule prescribes using autodiff to compute

$$K' = \partial dK / \partial \phi,$$

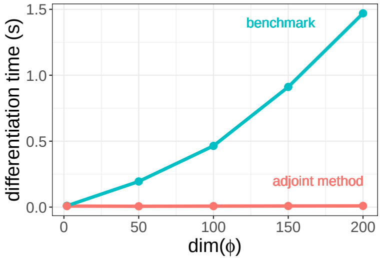
and plugging it back in the Algorithm.

For $\phi \in \mathbb{R}^D$ and $K \in \mathbb{R}^{N \times N}$, need

- D forward mode sweeps
- $\mathcal{O}(N^2)$ reverse mode sweeps.

With the right cotangent matrix Ω need one reverse mode sweep,

$$\left[\frac{\partial K}{\partial \phi}\right]^\dagger \cdot \Omega.$$



Adjoint method benchmarked against forward mode K'

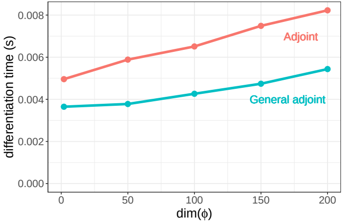
M et al. (2020), *NeurIPS*

Can apply the same principles to handle $\nabla_\theta^{(n)} \log \pi(\mathbf{y} \mid \theta, \phi)$.

Can apply the same principles to handle $\nabla_\theta^{(n)} \log \pi(\mathbf{y} \mid \theta, \phi)$.

If y_n only depends on θ_n, then all higher-oder objects are diagonal.

For non-diagonal Hessians++, want to contract Jacobians.



Adjoint with analytical plug-in for likelihood and general adjoint

M (2022), *PhD thesis*

Outline:

- Forward and reverse mode automatic differentiation
- Higher-oder differentiation
- Example: adjoint-differentiated Laplace
- Implicit functions

Suppose the operator $\mathtt{f}_\ell$ returns an implicit function.

- algebraic equation,
$$g(f_\ell, \theta) = 0$$
- ordinary differential equation,
$$g(\dot{f}_\ell, f_\ell, \theta, t) = 0$$
- approximate marginalization,
$$f_\ell = \log \int_\Theta \pi(\mathbf{y} \mid \theta, \phi) \mathrm{d}\theta.$$

Suppose the operator $\texttt{f}_\ell$ returns an implicit function.

- algebraic equation,
$$g(f_\ell, \theta) = 0$$
- ordinary differential equation,
$$g(\dot{f}_\ell, f_\ell, \theta, t) = 0$$
- approximate marginalization,
$$f_\ell = \log \int_\Theta \pi(\mathbf{y} \mid \theta, \phi) \mathrm{d}\theta.$$

How do we construct the differentiation operator $\texttt{df}_\ell$?

Example: algebraic equation

$$g(f_\ell, \theta) = 0$$

Example: algebraic equation

$$g(f_\ell, \theta) = 0$$

Trace method: *run autodiff through the elementary steps taken by a numerical integrator, e.g. Newton steps.*

Example: algebraic equation

$$g(f_\ell, \theta) = 0$$

Trace method: *run autodiff through the elementary steps taken by a numerical integrator, e.g. Newton steps.*

Implicit function theorem approach:

$$\frac{\mathrm{d}}{\mathrm{d}\theta} g(f_\ell, \theta) = \frac{\partial g}{\partial \theta} + \frac{\mathrm{d}g}{\mathrm{d}f_\ell} \frac{\partial f_\ell}{\partial \theta} = 0$$

Example: algebraic equation

$$g(f_\ell, \theta) = 0$$

Trace method: *run autodiff through the elementary steps taken by a numerical integrator, e.g. Newton steps.*

Implicit function theorem approach:

$$\frac{\mathrm{d}}{\mathrm{d}\theta} g(f_\ell, \theta) = \frac{\partial g}{\partial \theta} + \frac{\mathrm{d}g}{\mathrm{d}f_\ell} \frac{\partial f_\ell}{\partial \theta} = 0$$

Then

$$\frac{\partial f_\ell}{\partial \theta} = - \left[\frac{\mathrm{d}g}{\mathrm{d}f_\ell} \right]^{-1} \frac{\partial g}{\partial \theta}.$$

This expression can then be contracted against u or w.

Example: algebraic equation

$$g(f_\ell, \theta) = 0$$

Trace method: *run autodiff through the elementary steps taken by a numerical integrator, e.g. Newton steps.*

Implicit function theorem approach:

$$\frac{\mathrm{d}}{\mathrm{d}\theta} g(f_\ell, \theta) = \frac{\partial g}{\partial \theta} + \frac{\mathrm{d}g}{\mathrm{d}f_\ell} \frac{\partial f_\ell}{\partial \theta} = 0$$

Then

$$\frac{\partial f_\ell}{\partial \theta} = -\left[\frac{\mathrm{d}g}{\mathrm{d}f_\ell}\right]^{-1} \frac{\partial g}{\partial \theta}.$$

This expression can then be contracted against u or w.

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Trace method: *run autodiff through the elementary steps taken by a numerical integrator, e.g. Euler steps.*

Implicit function theorem approach:

$$\frac{\mathrm{d}f_\ell}{\mathrm{d}\theta}(T) = -\left[\frac{\delta g}{\delta f_\ell}\right]^{-1} \frac{\delta g}{\delta \theta},$$

where δ is a Fréchet derivative, which cannot be represented in finite memory.

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = \dot{f}_\ell - h(f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = \dot{f}_\ell - h(f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Forward method:

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\Big(\dot{f}_\ell - h(f_\ell, \theta, t)\Big) = \frac{\mathrm{d}\dot{f}_\ell}{\mathrm{d}\theta} - \frac{\partial h}{\partial \theta} - \frac{\mathrm{d}h}{\mathrm{d}f_\ell}\frac{\partial f_\ell}{\partial \theta} = 0,$$

which is an ODE solved by $\partial f_\ell/\partial\theta$.

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = \dot{f}_\ell - h(f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Forward method:

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\Big(\dot{f}_\ell - h(f_\ell, \theta, t)\Big) = \frac{\mathrm{d}\dot{f}_\ell}{\mathrm{d}\theta} - \frac{\partial h}{\partial \theta} - \frac{\mathrm{d}h}{\mathrm{d}f_\ell}\frac{\partial f_\ell}{\partial \theta} = 0,$$

which is an ODE solved by $\partial f_\ell / \partial \theta$.

For $D = \dim(\theta)$ and $N = \dim(f_\ell)$, the augmented ODE has $N + ND$ states.

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = \dot{f}_\ell - h(f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = \dot{f}_\ell - h(f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Adjoint method: Construct an ODE solved by

$$\lambda(T) = \left[\frac{\partial f_\ell}{\partial \theta}(T)\right]^\dagger \cdot w$$

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = \dot{f}_\ell - h(f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Adjoint method: Construct an ODE solved by

$$\lambda(T) = \left[\frac{\partial f_\ell}{\partial \theta}(T)\right]^\dagger \cdot w$$

Solving for λ requires solving an augmented ODE backwards in time with $N + D$ states.

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = \dot{f}_\ell - h(f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Adjoint method: Construct an ODE solved by

$$\lambda(T) = \left[\frac{\partial f_\ell}{\partial \theta}(T)\right]^\dagger \cdot w$$

Solving for λ requires solving an augmented ODE backwards in time with $N + D$ states.

Required to scale for large D's, e.g. Neural ODEs (Chen, Rubanova, Bettencourt and Duvenaud, 2018, *NeurIPS*)

Example: ordinary differential equation

$$g(\dot{f}_\ell, f_\ell, \theta, t) = \dot{f}_\ell - h(f_\ell, \theta, t) = 0$$

Want to differentiate $f_\ell(T)$ with respect to θ.

Adjoint method: Construct an ODE solved by

$$\lambda(T) = \left[\frac{\partial f_\ell}{\partial \theta}(T)\right]^\dagger \cdot w$$

Solving for λ requires solving an augmented ODE backwards in time with $N + D$ states.

Required to scale for large D's, e.g. Neural ODEs (Chen, Rubanova, Bettencourt and Duvenaud, 2018, *NeurIPS*)

No clear winner to differentiate ODEs (Ma et al 2018, *arXiv:1812.01892*).

What we covered:

- Autodiff is more than the chain rule!
- vector-Jacobian contractions
- Higher-order autodiff
- Implicit functions

What we didn't cover:

- Computational impelmentation
- Memory management
- Exploitation of SIMD

References

- M 2019, *WIRES*
- Baydin et al 2018, *JMLR*
- M and Betancourt 2022, *arXiv:2112.14217*