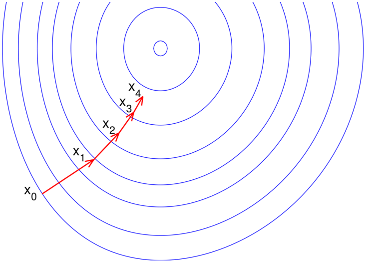
Basics of Numerical Optimization: Computing Derivatives

Ju Sun
Computer Science & Engineering
University of Minnesota, Twin Cities

February 25, 2020

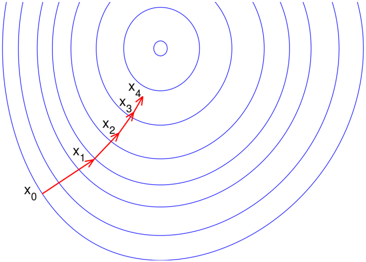
Derivatives for numerical optimization



Credit: aria42.com

- gradient descent
- Newton's method
- momentum methods
- quasi-Newton methods
- coordinate descent
- conjugate gradient methods
- trust-region methods
- etc

Derivatives for numerical optimization

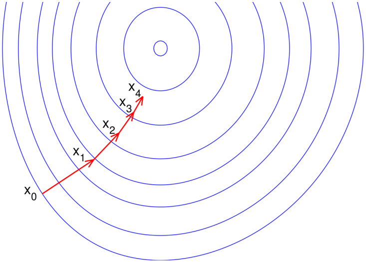


Credit: aria42.com

- gradient descent
- Newton's method
- momentum methods
- quasi-Newton methods
- coordinate descent
- conjugate gradient methods
- trust-region methods
- etc

- Almost all methods require low-order derivatives, i.e., gradient and/or Hessian, to proceed.

Derivatives for numerical optimization

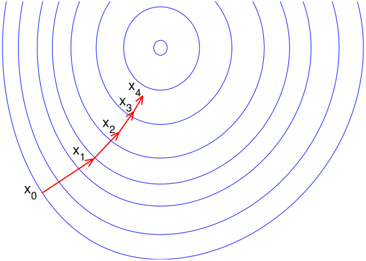


Credit: aria42.com

- gradient descent
- Newton's method
- momentum methods
- quasi-Newton methods
- coordinate descent
- conjugate gradient methods
- trust-region methods
- etc

- Almost all methods require low-order derivatives, i.e., gradient and/or Hessian, to proceed.
- Numerical derivatives (i.e., numbers), rather than analytic derivatives (i.e., math expressions), are needed

Derivatives for numerical optimization

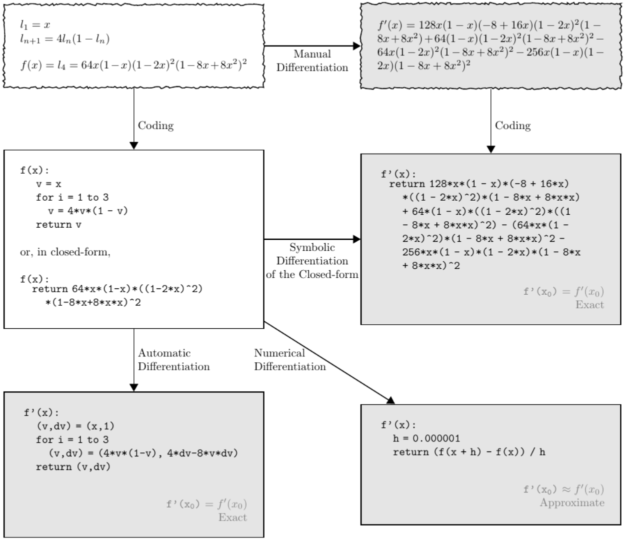


Credit: aria42.com

- gradient descent
- Newton's method
- momentum methods
- quasi-Newton methods
- coordinate descent
- conjugate gradient methods
- trust-region methods
- etc

- Almost all methods require low-order derivatives, i.e., gradient and/or Hessian, to proceed.
- Numerical derivatives (i.e., numbers), rather than analytic derivatives (i.e., math expressions), are needed

This lecture: how to compute the numerical derivatives



Credit: [Baydin et al., 2017]

Outline

Analytic derivatives

Idea: derive the analytic derivatives first, then make numerical substitution

Analytic derivatives

Idea: derive the analytic derivatives first, then make numerical substitution

To derive the analytic derivatives by hand:

- **Chain rule (vector version) method**
  Let $f : \mathbb{R}^m \to \mathbb{R}^n$ and $h : \mathbb{R}^n \to \mathbb{R}^k$, and f is differentiable at $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$ and h is differentiable at $\boldsymbol{y}$. Then, $h \circ f : \mathbb{R}^n \to \mathbb{R}^k$ is differentiable at $\boldsymbol{x}$, and

$$\boldsymbol{J}_{[h \circ f]}(\boldsymbol{x}) = \boldsymbol{J}_h(f(\boldsymbol{x}))\, \boldsymbol{J}_f(\boldsymbol{x}) .$$

  When $k = 1$,

$$\nabla [h \circ f](\boldsymbol{x}) = \boldsymbol{J}_f^\top(\boldsymbol{x})\, \nabla h(f(\boldsymbol{x})) .$$

Analytic derivatives

Idea: derive the analytic derivatives first, then make numerical substitution

To derive the analytic derivatives by hand:

- **Chain rule (vector version) method**
  Let $f : \mathbb{R}^m \to \mathbb{R}^n$ and $h : \mathbb{R}^n \to \mathbb{R}^k$, and f is differentiable at $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$ and h is differentiable at $\boldsymbol{y}$. Then, $h \circ f : \mathbb{R}^n \to \mathbb{R}^k$ is differentiable at $\boldsymbol{x}$, and

$$\boldsymbol{J}_{[h \circ f]}(\boldsymbol{x}) = \boldsymbol{J}_h(f(\boldsymbol{x}))\, \boldsymbol{J}_f(\boldsymbol{x}).$$

  When $k = 1$**,**

$$\nabla [h \circ f](\boldsymbol{x}) = \boldsymbol{J}_f^\top(\boldsymbol{x})\, \nabla h(f(\boldsymbol{x})).$$

- **Taylor expansion method**
  Expand the perturbed function $f(\boldsymbol{x} + \boldsymbol{\delta})$ and then match it against Taylor expansions to read off the gradient and/or Hessian:

$$f(\boldsymbol{x} + \boldsymbol{\delta}) \approx f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{\delta} \rangle$$

$$f(\boldsymbol{x} + \boldsymbol{\delta}) \approx f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{\delta} \rangle + \frac{1}{2} \langle \boldsymbol{\delta}, \nabla^2 f(\boldsymbol{x})\, \boldsymbol{\delta} \rangle$$

Derive chain rule by Taylor expansion (optional)

Start with $h(f(\boldsymbol{x}+\boldsymbol{\delta}))$, where $\boldsymbol{\delta}$ is always sufficiently small as we want

Derive chain rule by Taylor expansion (optional)

Start with $h(f(\boldsymbol{x}+\boldsymbol{\delta}))$, where $\boldsymbol{\delta}$ is always sufficiently small as we want

$$
\begin{aligned}
h(f(\boldsymbol{x}+\boldsymbol{\delta})) &= h\left(f(\boldsymbol{x}) + \underbrace{\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2)}_{\textbf{perturbation}}\right) \\
&= h(f(\boldsymbol{x})) + \boldsymbol{J}_h(f(\boldsymbol{x}))\left[\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2)\right] + \\
&\qquad \underbrace{o\left(\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2)\right)}_{o(\|\boldsymbol{\delta}\|_2)} \\
&= h(f(\boldsymbol{x})) + \underbrace{\boldsymbol{J}_h(f(\boldsymbol{x}))\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{\delta}}_{\textbf{linear term}} + o(\|\boldsymbol{\delta}\|_2),
\end{aligned}
$$

Derive chain rule by Taylor expansion (optional)

Start with $h(f(\boldsymbol{x}+\boldsymbol{\delta}))$, where $\boldsymbol{\delta}$ is always sufficiently small as we want

$$
\begin{aligned}
h(f(\boldsymbol{x}+\boldsymbol{\delta})) &= h\left(f(\boldsymbol{x}) + \underbrace{\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2)}_{\textbf{perturbation}}\right) \\
&= h(f(\boldsymbol{x})) + \boldsymbol{J}_h(f(\boldsymbol{x}))\left[\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2)\right] + \\
&\qquad \underbrace{o\left(\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|_2)\right)}_{o(\|\boldsymbol{\delta}\|_2)} \\
&= h(f(\boldsymbol{x})) + \underbrace{\boldsymbol{J}_h(f(\boldsymbol{x}))\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{\delta}}_{\textbf{linear term}} + o(\|\boldsymbol{\delta}\|_2),
\end{aligned}
$$

So,

$$
\boldsymbol{J}_{h\circ f(\boldsymbol{x})} = \boldsymbol{J}_h(f(\boldsymbol{x}))\boldsymbol{J}_f(\boldsymbol{x}).
$$

Taylor expansion method, again

Derive gradient of a three-layer linear neural network

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3} f\left(\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3\right) \doteq \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2$$

Taylor expansion method, again

Derive gradient of a three-layer linear neural network

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3} f(\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3) \doteq \sum_i \|\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\|_F^2$$

We can derive the partial gradients wrt $\boldsymbol{W}_i$'s separately. (Why?)

Taylor expansion method, again

Derive gradient of a three-layer linear neural network

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3} f\left(\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3\right) \doteq \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2$$

We can derive the partial gradients wrt $\boldsymbol{W}_i$'s separately. (Why?)

For example, for $\boldsymbol{W}_2$,

$$\begin{aligned}
& f\left(\boldsymbol{W}_1, \boldsymbol{W}_2 + \boldsymbol{\Delta}, \boldsymbol{W}_3\right) \\
= & \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \left(\boldsymbol{W}_2 + \boldsymbol{\Delta}\right) \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2 \\
= & \sum_i \left\|\left(\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right) - \boldsymbol{W}_3 \boldsymbol{\Delta} \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2 \\
= & \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2 - 2\left\langle \boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i, \boldsymbol{W}_3 \boldsymbol{\Delta} \boldsymbol{W}_1 \boldsymbol{x}_i \right\rangle + O(\|\boldsymbol{\Delta}\|_F^2) \\
= & \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2 \\
& \quad - 2 \sum_i \left\langle \boldsymbol{W}_3^{\intercal} \left(\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right) \left(\boldsymbol{W}_1 \boldsymbol{x}_i\right)^{\intercal}, \boldsymbol{\Delta} \right\rangle + O\left(\|\boldsymbol{\Delta}\|_F^2\right)
\end{aligned}$$

Taylor expansion method, again

Derive gradient of a three-layer linear neural network

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3} f\left(\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3\right) \doteq \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2$$

We can derive the partial gradients wrt $\boldsymbol{W}_i$'s separately. (Why?)

For example, for $\boldsymbol{W}_2$,

$$\begin{aligned}
& f\left(\boldsymbol{W}_1, \boldsymbol{W}_2 + \boldsymbol{\Delta}, \boldsymbol{W}_3\right) \\
= & \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \left(\boldsymbol{W}_2 + \boldsymbol{\Delta}\right) \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2 \\
= & \sum_i \left\|\left(\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right) - \boldsymbol{W}_3 \boldsymbol{\Delta} \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2 \\
= & \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2 - 2\left\langle \boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i, \boldsymbol{W}_3 \boldsymbol{\Delta} \boldsymbol{W}_1 \boldsymbol{x}_i\right\rangle + O(\|\boldsymbol{\Delta}\|_F^2) \\
= & \sum_i \left\|\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right\|_F^2 \\
& \quad - 2 \sum_i \left\langle \boldsymbol{W}_3^{\intercal} \left(\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right) \left(\boldsymbol{W}_1 \boldsymbol{x}_i\right)^{\intercal}, \boldsymbol{\Delta}\right\rangle + O\left(\|\boldsymbol{\Delta}\|_F^2\right)
\end{aligned}$$

So: $\nabla_{\boldsymbol{W}_2} f = -2 \sum_i \boldsymbol{W}_3^{\intercal} \left(\boldsymbol{y}_i - \boldsymbol{W}_3 \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}_i\right) \left(\boldsymbol{W}_1 \boldsymbol{x}_i\right)^{\intercal}$.

Symbolic differentiation

Idea: derive the analytic derivatives first, then make numerical substitution

To derive the analytic derivatives by software:

Differentiate Function

Find the derivative of the function `sin(x^2)`.

```
syms f(x)
f(x) = sin(x^2);
df = diff(f,x)
```

```
df(x) =
2*x*cos(x^2)
```

Find the value of the derivative at `x = 2`. Convert the value to double.

```
df2 = df(2)
```

```
df2 =
4*cos(4)
```

Symbolic differentiation

Idea: derive the analytic derivatives first, then make numerical substitution

To derive the analytic derivatives by software:

Differentiate Function

Find the derivative of the function `sin(x^2)`.

```
syms f(x)
f(x) = sin(x^2);
df = diff(f,x)
```

```
df(x) =
2*x*cos(x^2)
```

Find the value of the derivative at `x = 2`. Convert the value to double.

```
df2 = df(2)
```

```
df2 =
4*cos(4)
```

- Matlab (Symbolic Math Toolbox, `diff`)
- Python (SymPy, `diff`)
- Mathmatica (`D`)

Symbolic differentiation

Idea: derive the analytic derivatives first, then make numerical substitution

To derive the analytic derivatives by software:

Differentiate Function

Find the derivative of the function `sin(x^2)`.

```
syms f(x)
f(x) = sin(x^2);
df = diff(f,x)
```

```
df(x) =
2*x*cos(x^2)
```

Find the value of the derivative at `x = 2`. Convert the value to double.

```
df2 = df(2)
```

```
df2 =
4*cos(4)
```

- Matlab (Symbolic Math Toolbox, `diff`)
- Python (SymPy, `diff`)
- Mathmatica (`D`)

Effective for functions with few variables only

Outline

Analytic differentiation

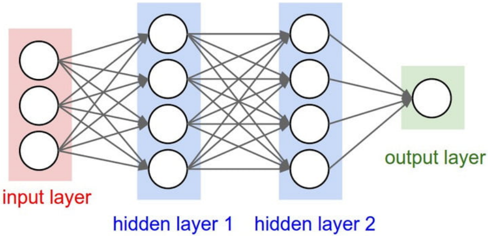
Finite-difference approximation

Automatic differentiation

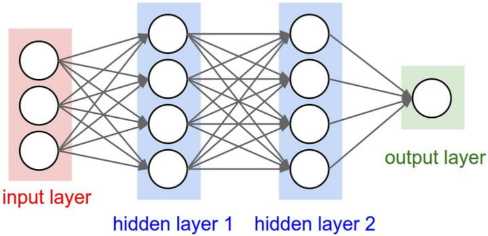
Differentiable programming

Suggested reading

Limitation of analytic differentiation



Limitation of analytic differentiation

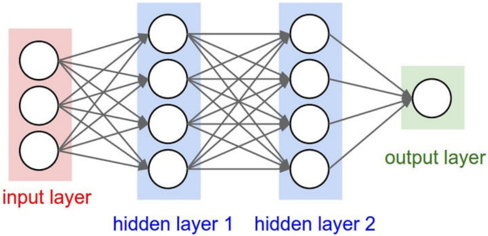


What is the gradient and/or Hessian of

$$f(\boldsymbol{W}) = \sum_i \|\boldsymbol{y}_i - \sigma(\boldsymbol{W}_k \sigma(\boldsymbol{W}_{k-1}\sigma \dots (\boldsymbol{W}_1 \boldsymbol{x}_i)))\|_F^2 ?$$

Applying the chain rule is boring and -prone. Performing Taylor expansion is also tedious

Limitation of analytic differentiation



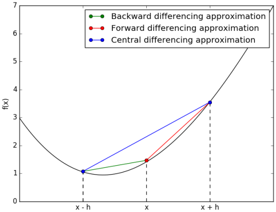
What is the gradient and/or Hessian of

$$f\left(\boldsymbol{W}\right) = \sum_i \left\| \boldsymbol{y}_i - \sigma\left(\boldsymbol{W}_k \sigma\left(\boldsymbol{W}_{k-1}\sigma \ldots \left(\boldsymbol{W}_1 \boldsymbol{x}_i\right)\right)\right)\right\|_F^2 ?$$

Applying the chain rule is boring and -prone. Performing Taylor expansion is also tedious

Lesson we learn from technology history: leave boring jobs to computers

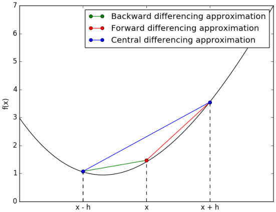
Approximate the gradient



(Credit: numex-blog.com)

$$f'(\boldsymbol{x}) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

Approximate the gradient



(Credit: numex-blog.com)

$f'(\boldsymbol{x}) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$

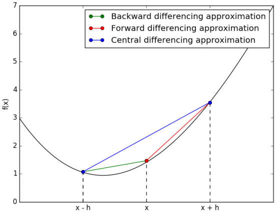
For $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x})}{\delta} \quad \text{(forward)}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x}) - f(\boldsymbol{x} - \delta \boldsymbol{e}_i)}{\delta} \quad \text{(backward)}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x} - \delta \boldsymbol{e}_i)}{2\delta} \quad \text{(central)}$$

Approximate the gradient



(Credit: numex-blog.com)

For $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x})}{\delta} \quad \text{(forward)}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x}) - f(\boldsymbol{x} - \delta \boldsymbol{e}_i)}{\delta} \quad \text{(backward)}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x} - \delta \boldsymbol{e}_i)}{2\delta} \quad \text{(central)}$$

$f'(\boldsymbol{x}) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$

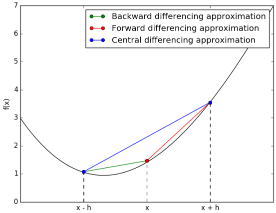
Similarly, to approximate the Jacobian for $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$:

$$\frac{\partial f_j}{\partial x_i} \approx \frac{f_j(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f_j(\boldsymbol{x})}{\delta} \quad \text{(one element each time)}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x})}{\delta} \quad \text{(one column each time)}$$

$$\boldsymbol{J}\boldsymbol{p} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{p}) - f(\boldsymbol{x})}{\delta} \quad \text{(directional)}$$

Approximate the gradient



(Credit: numex-blog.com)

For $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x})}{\delta} \quad \text{(forward)}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x}) - f(\boldsymbol{x} - \delta \boldsymbol{e}_i)}{\delta} \quad \text{(backward)}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x} - \delta \boldsymbol{e}_i)}{2\delta} \quad \text{(central)}$$

$f'(\boldsymbol{x}) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$

Similarly, to approximate the Jacobian for $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$:

$$\frac{\partial f_j}{\partial x_i} \approx \frac{f_j(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f_j(\boldsymbol{x})}{\delta} \quad \text{(one element each time)}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x})}{\delta} \quad \text{(one column each time)}$$

$$\boldsymbol{J}\boldsymbol{p} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{p}) - f(\boldsymbol{x})}{\delta} \quad \text{(directional)}$$

central themes can also be derived

Why central?

Stronger form of Taylor's theorems

- **1st order**: If $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable,
  $f(\boldsymbol{x} + \delta) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{\delta} \rangle + O\left(\|\boldsymbol{\delta}\|_2^2\right)$
- **2nd order**: If $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is three-times continuously differentiable,
  $f(\boldsymbol{x} + \delta) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{\delta} \rangle + \frac{1}{2} \left\langle \boldsymbol{\delta}, \nabla^2 f(\boldsymbol{x}) \boldsymbol{\delta} \right\rangle + O\left(\|\boldsymbol{\delta}\|_2^3\right)$

Why central?

Stronger form of Taylor's theorems

- **1st order**: If $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable,
  $f(\boldsymbol{x} + \delta) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{\delta} \rangle + O\left(\|\boldsymbol{\delta}\|_2^2\right)$
- **2nd order**: If $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is three-times continuously differentiable,
  $f(\boldsymbol{x} + \delta) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{\delta} \rangle + \frac{1}{2}\left\langle \boldsymbol{\delta}, \nabla^2 f(\boldsymbol{x}) \boldsymbol{\delta} \right\rangle + O\left(\|\boldsymbol{\delta}\|_2^3\right)$

Why the central theme is better?

- Forward: by 1st-order Taylor expansion
  $\frac{1}{\delta}\left(f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x})\right) = \frac{1}{\delta}\left(\delta \frac{\partial f}{\partial x_i} + O\left(\delta^2\right)\right) = \frac{\partial f}{\partial x_i} + O(\delta)$

Why central?

Stronger form of Taylor's theorems

- **1st order**: If $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, $f(\boldsymbol{x} + \delta) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{\delta} \rangle + O\left(\|\boldsymbol{\delta}\|_2^2\right)$
- **2nd order**: If $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is three-times continuously differentiable, $f(\boldsymbol{x} + \delta) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{\delta} \rangle + \frac{1}{2} \left\langle \boldsymbol{\delta}, \nabla^2 f(\boldsymbol{x}) \boldsymbol{\delta} \right\rangle + O\left(\|\boldsymbol{\delta}\|_2^3\right)$

Why the central theme is better?

- Forward: by 1st-order Taylor expansion $\frac{1}{\delta}\left(f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x})\right) = \frac{1}{\delta}\left(\delta \frac{\partial f}{\partial x_i} + O\left(\delta^2\right)\right) = \frac{\partial f}{\partial x_i} + O(\delta)$
- Central: by 2nd-order Taylor expansion $\frac{1}{\delta}\left(f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x} - \delta \boldsymbol{e}_i)\right) = \frac{1}{2\delta}\left(\delta \frac{\partial f}{\partial x_i} + \frac{1}{2}\delta^2 \frac{\partial^2 f}{\partial x_i^2} + \delta \frac{\partial f}{\partial x_i} - \frac{1}{2}\delta^2 \frac{\partial^2 f}{\partial x_i^2} + O\left(\delta^3\right)\right) = \frac{\partial f}{\partial x_i} + O(\delta^2)$

Approximate the Hessian

- Recall that for $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ that is 2nd-order differentiable, $\frac{\partial f}{\partial x_i}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$. So

$$\frac{\partial f^2}{\partial x_j \partial x_i}(\boldsymbol{x}) = \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}\right)(\boldsymbol{x}) \approx \frac{\left(\frac{\partial f}{\partial x_i}\right)(\boldsymbol{x} + \delta \boldsymbol{e}_j) - \left(\frac{\partial f}{\partial x_i}\right)(\boldsymbol{x})}{\delta}$$

Approximate the Hessian

- Recall that for $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ that is 2nd-order differentiable, $\frac{\partial f}{\partial x_i}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$. So

$$\frac{\partial f^2}{\partial x_j \partial x_i}(\boldsymbol{x}) = \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}\right)(\boldsymbol{x}) \approx \frac{\left(\frac{\partial f}{\partial x_i}\right)(\boldsymbol{x} + \delta \boldsymbol{e}_j) - \left(\frac{\partial f}{\partial x_i}\right)(\boldsymbol{x})}{\delta}$$

- We can also compute one row of Hessian each time by

$$\frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial \boldsymbol{x}}\right)(\boldsymbol{x}) \approx \frac{\left(\frac{\partial f}{\partial \boldsymbol{x}}\right)(\boldsymbol{x} + \delta \boldsymbol{e}_j) - \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)(\boldsymbol{x})}{\delta},$$

obtaining $\widehat{H}$, which might not be symmetric. Return $\frac{1}{2}\left(\widehat{\boldsymbol{H}} + \widehat{\boldsymbol{H}}^\intercal\right)$ instead

- Most times (e.g., in TRM, Newton-CG), only $\nabla^2 f(\boldsymbol{x})\boldsymbol{v}$ for certain $\boldsymbol{v}$'s needed: (see, e.g., Manopt https://www.manopt.org/)

$$\nabla^2 f(\boldsymbol{x})\boldsymbol{v} \approx \frac{\nabla f(\boldsymbol{x} + \delta \boldsymbol{v}) - f(\boldsymbol{x})}{\delta} \tag{1}$$

A few words

– Can be used for sanity check for correctness of analytic gradient

A few words

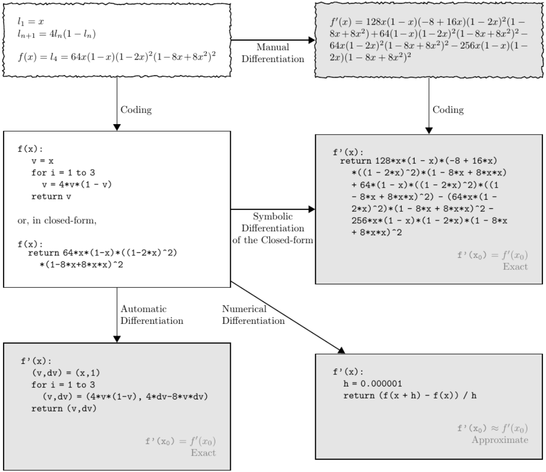
- Can be used for sanity check for correctness of analytic gradient
- Finite-difference approximation of higher (i.e., ≥ 2)-order derivatives combined with high-order iterative methods can be very efficient (e.g., Manopt https://www.manopt.org/tutorial.html#costdescription)

A few words

- Can be used for sanity check for correctness of analytic gradient
- Finite-difference approximation of higher (i.e., ≥ 2)-order derivatives combined with high-order iterative methods can be very efficient (e.g., Manopt https://www.manopt.org/tutorial.html#costdescription)
- Numerical stability can be an issue: truncation and round off s (finite δ; accurate evaluation of the nominators)

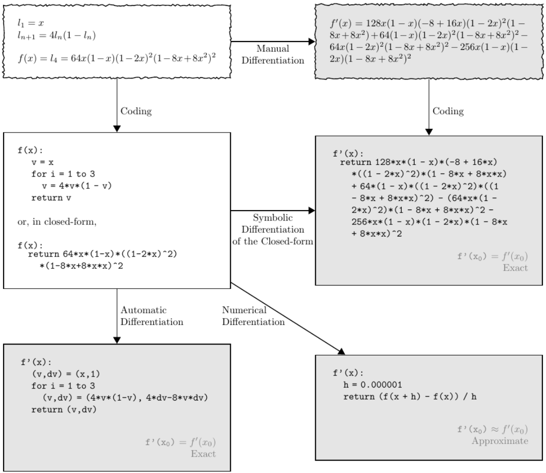
Outline

Four kinds of computing techniques



Credit: [Baydin et al., 2017]

Four kinds of computing techniques



Credit: [Baydin et al., 2017]

Misnomer: should be **automatic numerical differentiation**

Forward mode in 1D

Consider a univariate function $f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1(x) : \mathbb{R} \to \mathbb{R}$. Write $y_0 = x$, $y_1 = f_1(x)$, $y_2 = f_2(y_1)$, $\ldots$, $y_k = f(y_{k-1})$, or in **computational graph** form:

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Forward mode in 1D

Consider a univariate function $f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1(x) : \mathbb{R} \to \mathbb{R}$. Write $y_0 = x$, $y_1 = f_1(x)$, $y_2 = f_2(y_1)$, ..., $y_k = f(y_{k-1})$, or in **computational graph** form:

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Chain rule: $\dfrac{df}{dx} = \dfrac{df}{dy_0} = \left(\dfrac{dy_k}{dy_{k-1}} \left(\dfrac{dy_{k-1}}{dy_{k-2}} \left(\cdots \left(\dfrac{dy_2}{dy_1} \left(\dfrac{dy_1}{dy_0} \right) \right) \right) \right) \right)$

Forward mode in 1D

Consider a univariate function $f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1(x) : \mathbb{R} \to \mathbb{R}$. Write $y_0 = x$, $y_1 = f_1(x)$, $y_2 = f_2(y_1)$, $\ldots$, $y_k = f(y_{k-1})$, or in **computational graph** form:

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Chain rule: $$\frac{df}{dx} = \frac{df}{dy_0} = \left(\frac{dy_k}{dy_{k-1}}\left(\frac{dy_{k-1}}{dy_{k-2}}\left(\cdots\left(\frac{dy_2}{dy_1}\left(\frac{dy_1}{dy_0}\right)\right)\right)\right)\right)$$

Compute $\left.\frac{df}{dx}\right|_{x_0}$ in one pass, from inner to outer most parenthesis:

Forward mode in 1D

Consider a univariate function $f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1 (x) : \mathbb{R} \to \mathbb{R}$. Write $y_0 = x$, $y_1 = f_1(x)$, $y_2 = f_2(y_1)$, $\ldots$, $y_k = f(y_{k-1})$, or in **computational graph** form:

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Chain rule: $$\frac{df}{dx} = \frac{df}{dy_0} = \left(\frac{dy_k}{dy_{k-1}} \left(\frac{dy_{k-1}}{dy_{k-2}} \left(\cdots \left(\frac{dy_2}{dy_1} \left(\frac{dy_1}{dy_0} \right) \right) \right) \right) \right)$$

Compute $\left.\frac{df}{dx}\right|_{x_0}$ in one pass, from inner to outer most parenthesis:

Input: x_0, initialization $\left.\frac{dy_0}{dy_0}\right|_{x_0} = 1$

  for $i = 1, \ldots, k$ **do**

    compute $y_i = f_i(y_{i-1})$

    compute $\left.\frac{dy_i}{dy_0}\right|_{x_0} = \left.\frac{dy_i}{dy_{i-1}}\right|_{y_{i-1}} \cdot \left.\frac{dy_{i-1}}{dy_0}\right|_{x_0} = f_i'(y_{i-1}) \left.\frac{dy_{i-1}}{dy_0}\right|_{x_0}$

  end for

Output: $\left.\frac{dy_k}{dy_0}\right|_{x_0}$

Reverse mode in 1D

Consider a univariate function $f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1 (x) : \mathbb{R} \to \mathbb{R}$. Write $y_0 = x$, $y_1 = f_1(x)$, $y_2 = f_2(y_1)$, $\ldots$, $y_k = f(y_{k-1})$, or in **computational graph** form:

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Reverse mode in 1D

Consider a univariate function $f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1 (x) : \mathbb{R} \to \mathbb{R}$. Write $y_0 = x$, $y_1 = f_1(x)$, $y_2 = f_2(y_1)$, $\ldots$, $y_k = f(y_{k-1})$, or in **computational graph** form:

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Chain rule: $$\frac{df}{dx} = \frac{df}{dy_0} = \left(\left(\left(\left(\left(\frac{dy_k}{dy_{k-1}}\right)\frac{dy_{k-1}}{dy_{k-2}}\right)\ldots\right)\frac{dy_2}{dy_1}\right)\frac{dy_1}{dy_0}\right)$$

Reverse mode in 1D

Consider a univariate function $f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1(x) : \mathbb{R} \to \mathbb{R}$. Write $y_0 = x$, $y_1 = f_1(x)$, $y_2 = f_2(y_1)$, $\ldots$, $y_k = f(y_{k-1})$, or in **computational graph** form:

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Chain rule: $$\frac{df}{dx} = \frac{df}{dy_0} = \left(\left(\left(\left(\left(\frac{dy_k}{dy_{k-1}}\right)\frac{dy_{k-1}}{dy_{k-2}}\right)\ldots\right)\frac{dy_2}{dy_1}\right)\frac{dy_1}{dy_0}\right)$$

Compute $\left.\frac{df}{dx}\right|_{x_0}$ in two passes, from inner to outer most parenthesis for the 2nd:

Reverse mode in 1D

Consider a univariate function $f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1(x) : \mathbb{R} \to \mathbb{R}$. Write $y_0 = x$, $y_1 = f_1(x)$, $y_2 = f_2(y_1)$, $\ldots$, $y_k = f(y_{k-1})$, or in **computational graph** form:

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Chain rule: $$\frac{df}{dx} = \frac{df}{dy_0} = \left(\left(\left(\left(\left(\frac{dy_k}{dy_{k-1}}\right)\frac{dy_{k-1}}{dy_{k-2}}\right)\ldots\right)\frac{dy_2}{dy_1}\right)\frac{dy_1}{dy_0}\right)$$

Compute $\left.\frac{df}{dx}\right|_{x_0}$ in two passes, from inner to outer most parenthesis for the 2nd:

Input: x_0, $\frac{dy_k}{dy_k} = 1$

for $i = 1, \ldots, k$ **do**

compute $y_i = f_i(y_{i-1})$

end for // forward pass

for $i = k-1, k-2, \ldots, 0$ **do**

compute $\left.\frac{dy_k}{dy_i}\right|_{y_i} = \left.\frac{dy_k}{dy_{i+1}}\right|_{y_{i+1}} \cdot \left.\frac{dy_{i+1}}{dy_i}\right|_{y_i} = f'_{i+1}(y_i)\left.\frac{dy_k}{dy_{i+1}}\right|_{y_{i+1}}$

end for // backward pass

Output: $\left.\frac{dy_k}{dy_0}\right|_{x_0}$

Forward vs reverse modes

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

Forward vs reverse modes

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

– **forward mode AD**: one forward pass, compute the intermediate variable and derivative values together
– **reverse mode AD**: one forward pass to compute the intermediate variable values, one backward pass to compute the intermediate derivatives

Forward vs reverse modes

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

- **forward mode AD**: one forward pass, compute the intermediate variable and derivative values together
- **reverse mode AD**: one forward pass to compute the intermediate variable values, one backward pass to compute the intermediate derivatives

Effectively, two different ways of grouping the multiplicative differential terms:

$$\frac{df}{dx} = \frac{df}{dy_0} = \left(\frac{dy_k}{dy_{k-1}} \left(\frac{dy_{k-1}}{dy_{k-2}} \left(\cdots \left(\frac{dy_2}{dy_1} \left(\frac{dy_1}{dy_0} \right) \right) \right) \right) \right)$$

i.e., starting from the root: $\frac{dy_0}{dy_0} \mapsto \frac{dy_1}{dy_0} \mapsto \frac{dy_2}{dy_0} \mapsto \cdots \mapsto \frac{dy_k}{dy_0}$

$$\frac{df}{dx} = \frac{df}{dy_0} = \left(\left(\left(\left(\left(\frac{dy_k}{dy_{k-1}} \right) \frac{dy_{k-1}}{dy_{k-2}} \right) \cdots \right) \frac{dy_2}{dy_1} \right) \frac{dy_1}{dy_0} \right)$$

i.e., starting from the leaf: $\frac{dy_k}{dy_k} \mapsto \frac{dy_k}{dy_{k-1}} \mapsto \frac{dy_k}{dy_{k-2}} \mapsto \cdots \mapsto \frac{dy_k}{dy_0}$

...mixed forward and reverse modes are indeed possible!

Forward vs reverse modes

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

- **forward mode AD**: one forward pass, compute the intermediate variable and derivative values together
- **reverse mode AD**: one forward pass to compute the intermediate variable values, one backward pass to compute the intermediate derivatives

Effectively, two different ways of grouping the multiplicative differential terms:

$$\frac{df}{dx} = \frac{df}{dy_0} = \left(\frac{dy_k}{dy_{k-1}}\left(\frac{dy_{k-1}}{dy_{k-2}}\left(\cdots\left(\frac{dy_2}{dy_1}\left(\frac{dy_1}{dy_0}\right)\right)\right)\right)\right)$$

i.e., starting from the root: $\frac{dy_0}{dy_0} \mapsto \frac{dy_1}{dy_0} \mapsto \frac{dy_2}{dy_0} \mapsto \cdots \mapsto \frac{dy_k}{dy_0}$

$$\frac{df}{dx} = \frac{df}{dy_0} = \left(\left(\left(\left(\left(\frac{dy_k}{dy_{k-1}}\right)\frac{dy_{k-1}}{dy_{k-2}}\right)\cdots\right)\frac{dy_2}{dy_1}\right)\frac{dy_1}{dy_0}\right)$$

i.e., starting from the leaf: $\frac{dy_k}{dy_k} \mapsto \frac{dy_k}{dy_{k-1}} \mapsto \frac{dy_k}{dy_{k-2}} \mapsto \cdots \mapsto \frac{dy_k}{dy_0}$

Forward vs reverse modes

$$y_0 \xrightarrow{f_1} y_1 \xrightarrow{f_2} y_2 \xrightarrow{f_3} \cdots \xrightarrow{f_k} y_k$$

- **forward mode AD**: one forward pass, compute the intermediate variable and derivative values together
- **reverse mode AD**: one forward pass to compute the intermediate variable values, one backward pass to compute the intermediate derivatives

Effectively, two different ways of grouping the multiplicative differential terms:

$$\frac{df}{dx} = \frac{df}{dy_0} = \left(\frac{dy_k}{dy_{k-1}}\left(\frac{dy_{k-1}}{dy_{k-2}}\left(\cdots\left(\frac{dy_2}{dy_1}\left(\frac{dy_1}{dy_0}\right)\right)\right)\right)\right)$$

i.e., starting from the root: $\frac{dy_0}{dy_0} \mapsto \frac{dy_1}{dy_0} \mapsto \frac{dy_2}{dy_0} \mapsto \cdots \mapsto \frac{dy_k}{dy_0}$

$$\frac{df}{dx} = \frac{df}{dy_0} = \left(\left(\left(\left(\left(\frac{dy_k}{dy_{k-1}}\right)\frac{dy_{k-1}}{dy_{k-2}}\right)\cdots\right)\frac{dy_2}{dy_1}\right)\frac{dy_1}{dy_0}\right)$$

i.e., starting from the leaf: $\frac{dy_k}{dy_k} \mapsto \frac{dy_k}{dy_{k-1}} \mapsto \frac{dy_k}{dy_{k-2}} \mapsto \cdots \mapsto \frac{dy_k}{dy_0}$

...mixed forward and reverse modes are indeed possible!

Chain rule in computational graphs

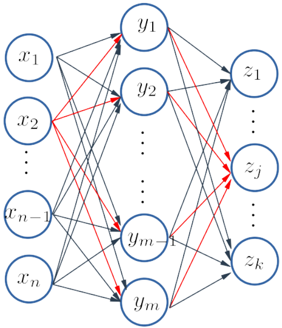
Chain rule Let $f : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^k$, and f is differentiable at $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$ and h is differentiable at $\boldsymbol{y}$. Then, $h \circ f : \mathbb{R}^n \to \mathbb{R}^k$ is differentiable at $\boldsymbol{x}$, and (write $\boldsymbol{z} = h(\boldsymbol{y})$)

$$\boldsymbol{J}_{[h \circ f]}(\boldsymbol{x}) = \boldsymbol{J}_h(f(\boldsymbol{x}))\, \boldsymbol{J}_f(\boldsymbol{x}), \quad \text{or} \; \frac{\partial z_j}{\partial x_i} = \sum_{\ell=1}^{m} \frac{\partial z_j}{\partial y_\ell} \frac{\partial y_\ell}{\partial x_i} \; \forall\, i, j$$

Chain rule in computational graphs

Chain rule Let $f : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^k$, and f is differentiable at $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$ and h is differentiable at $\boldsymbol{y}$. Then, $h \circ f : \mathbb{R}^n \to \mathbb{R}^k$ is differentiable at $\boldsymbol{x}$, and (write $\boldsymbol{z} = h(\boldsymbol{y})$)

$$\boldsymbol{J}_{[h \circ f]}(\boldsymbol{x}) = \boldsymbol{J}_h(f(\boldsymbol{x}))\, \boldsymbol{J}_f(\boldsymbol{x}), \quad \text{or } \frac{\partial z_j}{\partial x_i} = \sum_{\ell=1}^{m} \frac{\partial z_j}{\partial y_\ell} \frac{\partial y_\ell}{\partial x_i} \;\forall\, i, j$$

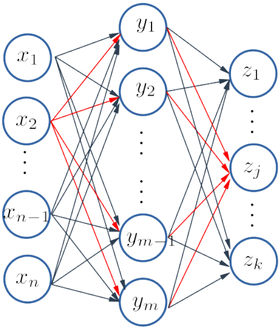


NB: this is a computational graph, not a NN

Chain rule in computational graphs

Chain rule Let $f : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^k$, and f is differentiable at $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$ and h is differentiable at $\boldsymbol{y}$. Then, $h \circ f : \mathbb{R}^n \to \mathbb{R}^k$ is differentiable at $\boldsymbol{x}$, and (write $\boldsymbol{z} = h(\boldsymbol{y})$)

$$\boldsymbol{J}_{[h \circ f]}(\boldsymbol{x}) = \boldsymbol{J}_h(f(\boldsymbol{x})) \boldsymbol{J}_f(\boldsymbol{x}), \quad \text{or } \frac{\partial z_j}{\partial x_i} = \sum_{\ell=1}^{m} \frac{\partial z_j}{\partial y_\ell} \frac{\partial y_\ell}{\partial x_i} \ \forall\, i, j$$



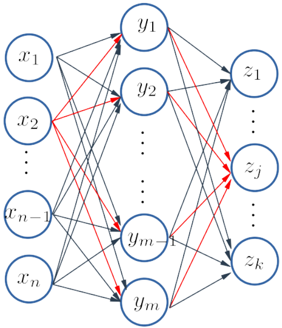
- Each node is a variable, as a function of all incoming variables

NB: this is a computational graph, not a NN

Chain rule in computational graphs

Chain rule Let $f : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^k$, and f is differentiable at $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$ and h is differentiable at $\boldsymbol{y}$. Then, $h \circ f : \mathbb{R}^n \to \mathbb{R}^k$ is differentiable at $\boldsymbol{x}$, and (write $\boldsymbol{z} = h(\boldsymbol{y})$)

$$\boldsymbol{J}_{[h \circ f]}(\boldsymbol{x}) = \boldsymbol{J}_h(f(\boldsymbol{x}))\, \boldsymbol{J}_f(\boldsymbol{x}), \quad \text{or } \frac{\partial z_j}{\partial x_i} = \sum_{\ell=1}^{m} \frac{\partial z_j}{\partial y_\ell} \frac{\partial y_\ell}{\partial x_i} \ \forall\, i, j$$



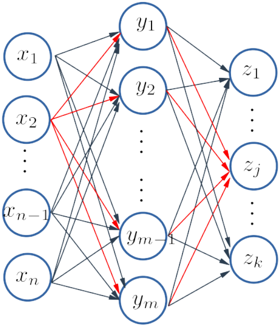
- Each node is a variable, as a function of all incoming variables
- If node B a descent of node A, $\frac{\partial B}{\partial A}$ is the rate of change in B wrt change in A

NB: this is a computational graph, not a NN

Chain rule in computational graphs

Chain rule Let $f : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^k$, and f is differentiable at $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$ and h is differentiable at $\boldsymbol{y}$. Then, $h \circ f : \mathbb{R}^n \to \mathbb{R}^k$ is differentiable at $\boldsymbol{x}$, and (write $\boldsymbol{z} = h(\boldsymbol{y})$)

$$\boldsymbol{J}_{[h \circ f]}(\boldsymbol{x}) = \boldsymbol{J}_h(f(\boldsymbol{x}))\, \boldsymbol{J}_f(\boldsymbol{x}), \quad \text{or} \ \frac{\partial z_j}{\partial x_i} = \sum_{\ell=1}^{m} \frac{\partial z_j}{\partial y_\ell} \frac{\partial y_\ell}{\partial x_i} \ \forall\, i, j$$



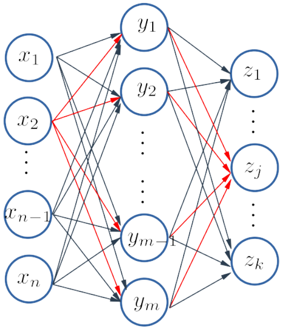
- Each node is a variable, as a function of all incoming variables
- If node B a descent of node A, $\frac{\partial B}{\partial A}$ is the rate of change in B wrt change in A
- Traveling along a path, rates of changes should be multiplied

NB: this is a computational graph, not a NN

Chain rule in computational graphs

Chain rule Let $f : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^k$, and f is differentiable at $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$ and h is differentiable at $\boldsymbol{y}$. Then, $h \circ f : \mathbb{R}^n \to \mathbb{R}^k$ is differentiable at $\boldsymbol{x}$, and (write $\boldsymbol{z} = h(\boldsymbol{y})$)

$$\boldsymbol{J}_{[h \circ f]}(\boldsymbol{x}) = \boldsymbol{J}_h(f(\boldsymbol{x}))\, \boldsymbol{J}_f(\boldsymbol{x}), \quad \text{or } \frac{\partial z_j}{\partial x_i} = \sum_{\ell=1}^{m} \frac{\partial z_j}{\partial y_\ell} \frac{\partial y_\ell}{\partial x_i} \;\forall\, i, j$$

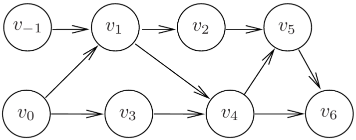


NB: this is a computational graph, not a NN

- Each node is a variable, as a function of all incoming variables
- If node B a descent of node A, $\frac{\partial B}{\partial A}$ is the rate of change in B wrt change in A
- Traveling along a path, rates of changes should be multiplied
- Chain rule: summing up rates over all connecting paths! (e.g., x_2 to z_j as shown)

A multivariate example — forward mode

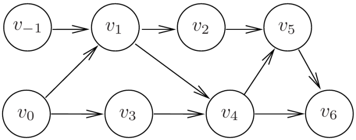
$$y = \left(\sin \frac{x_1}{x_2} + \frac{x_1}{x_2} - e^{x_2}\right)\left(\frac{x_1}{x_2} - e^{x_2}\right)$$



v_{-1}	$=$	x_1	$=$	1.5000		
v_0	$=$	x_2	$=$	0.5000		
v_1	$=$	v_{-1}/v_0	$=$	$1.5000/0.5000$	$=$	3.0000
v_2	$=$	$\sin(v_1)$	$=$	$\sin(3.0000)$	$=$	0.1411
v_3	$=$	$\exp(v_0)$	$=$	$\exp(0.5000)$	$=$	1.6487
v_4	$=$	$v_1 - v_3$	$=$	$3.0000 - 1.6487$	$=$	1.3513
v_5	$=$	$v_2 + v_4$	$=$	$0.1411 + 1.3513$	$=$	1.4924
v_6	$=$	$v_5 * v_4$	$=$	$1.4924 * 1.3513$	$=$	2.0167
y	$=$	v_6	$=$	2.0167		

A multivariate example — forward mode

$$y = \left(\sin\frac{x_1}{x_2} + \frac{x_1}{x_2} - e^{x_2}\right)\left(\frac{x_1}{x_2} - e^{x_2}\right)$$



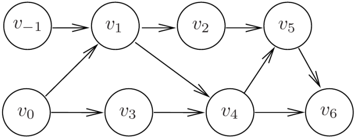
v_{-1}	$=$	x_1	$=$	1.5000		
v_0	$=$	x_2	$=$	0.5000		
v_1	$=$	v_{-1}/v_0	$=$	$1.5000/0.5000$	$=$	3.0000
v_2	$=$	$\sin(v_1)$	$=$	$\sin(3.0000)$	$=$	0.1411
v_3	$=$	$\exp(v_0)$	$=$	$\exp(0.5000)$	$=$	1.6487
v_4	$=$	$v_1 - v_3$	$=$	$3.0000 - 1.6487$	$=$	1.3513
v_5	$=$	$v_2 + v_4$	$=$	$0.1411 + 1.3513$	$=$	1.4924
v_6	$=$	$v_5 * v_4$	$=$	$1.4924 * 1.3513$	$=$	2.0167
y	$=$	v_6	$=$	2.0167		

v_{-1}	$= x_1$	$= 1.5000$	
$\dot v_{-1}$	$= \dot x_1$	$= 1.0000$	
v_0	$= x_2$	$= 0.5000$	
$\dot v_0$	$= \dot x_2$	$= 0.0000$	
v_1	$= v_{-1}/v_0$	$= 1.5000/0.5000$	$= 3.0000$
$\dot v_1$	$= (\dot v_{-1} - v_1 * \dot v_0)/v_0$	$= 1.0000/0.5000$	$= 2.0000$
v_2	$= \sin(v_1)$	$= \sin(3.0000)$	$= 0.1411$
$\dot v_2$	$= \cos(v_1) * \dot v_1$	$= -0.9900 * 2.0000$	$= -1.9800$
v_3	$= \exp(v_0)$	$= \exp(0.5000)$	$= 1.6487$
$\dot v_3$	$= v_3 * \dot v_0$	$= 1.6487 * 0.0000$	$= 0.0000$
v_4	$= v_1 - v_3$	$= 3.0000 - 1.6487$	$= 1.3513$
$\dot v_4$	$= \dot v_1 - \dot v_3$	$= 2.0000 - 0.0000$	$= 2.0000$
v_5	$= v_2 + v_4$	$= 0.1411 + 1.3513$	$= 1.4924$
$\dot v_5$	$= \dot v_2 + \dot v_4$	$= -1.9800 + 2.0000$	$= 0.0200$
v_6	$= v_5 * v_4$	$= 1.4924 * 1.3513$	$= 2.0167$
$\dot v_6$	$= \dot v_5 * v_4 + v_5 * \dot v_4$	$= 0.0200 * 1.3513 + 1.4924 * 2.0000$	$= 3.0118$
y	$= v_6$	$= 2.0100$	
$\dot y$	$= \dot v_6$	$= 3.0110$	

– interested in $\frac{\partial}{\partial x_1}$; for each variable v_i, write $\dot v_i \doteq \frac{\partial v_i}{\partial x_1}$

A multivariate example — forward mode

$$y = \left(\sin\frac{x_1}{x_2} + \frac{x_1}{x_2} - e^{x_2}\right)\left(\frac{x_1}{x_2} - e^{x_2}\right)$$



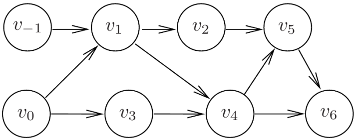
v_{-1}	$=$	x_1	$=$	1.5000		
v_0	$=$	x_2	$=$	0.5000		
v_1	$=$	v_{-1}/v_0	$=$	$1.5000/0.5000$	$=$	3.0000
v_2	$=$	$\sin(v_1)$	$=$	$\sin(3.0000)$	$=$	0.1411
v_3	$=$	$\exp(v_0)$	$=$	$\exp(0.5000)$	$=$	1.6487
v_4	$=$	$v_1 - v_3$	$=$	$3.0000 - 1.6487$	$=$	1.3513
v_5	$=$	$v_2 + v_4$	$=$	$0.1411 + 1.3513$	$=$	1.4924
v_6	$=$	$v_5 * v_4$	$=$	$1.4924 * 1.3513$	$=$	2.0167
y	$=$	v_6	$=$	2.0167		

$v_{-1} = x_1$	$= 1.5000$		
$\dot{v}_{-1} = \dot{x}_1$	$= 1.0000$		
$v_0 = x_2$	$= 0.5000$		
$\dot{v}_0 = \dot{x}_2$	$= 0.0000$		
$v_1 = v_{-1}/v_0$	$= 1.5000/0.5000$	$=$	3.0000
$\dot{v}_1 = (\dot{v}_{-1} - v_1 * \dot{v}_0)/v_0$	$= 1.0000/0.5000$	$=$	2.0000
$v_2 = \sin(v_1)$	$= \sin(3.0000)$	$=$	0.1411
$\dot{v}_2 = \cos(v_1) * \dot{v}_1$	$= -0.9900 * 2.0000$	$=$	-1.9800
$v_3 = \exp(v_0)$	$= \exp(0.5000)$	$=$	1.6487
$\dot{v}_3 = v_3 * \dot{v}_0$	$= 1.6487 * 0.0000$	$=$	0.0000
$v_4 = v_1 - v_3$	$= 3.0000 - 1.6487$	$=$	1.3513
$\dot{v}_4 = \dot{v}_1 - \dot{v}_3$	$= 2.0000 - 0.0000$	$=$	2.0000
$v_5 = v_2 + v_4$	$= 0.1411 + 1.3513$	$=$	1.4924
$\dot{v}_5 = \dot{v}_2 + \dot{v}_4$	$= -1.9800 + 2.0000$	$=$	0.0200
$v_6 = v_5 * v_4$	$= 1.4924 * 1.3513$	$=$	2.0167
$\dot{v}_6 = \dot{v}_5 * v_4 + v_5 * \dot{v}_4$	$= 0.0200 * 1.3513 + 1.4924 * 2.0000$	$=$	3.0118
$y = v_6$	$= 2.0100$		
$\dot{y} = \dot{v}_6$	$= 3.0110$		

- interested in $\frac{\partial}{\partial x_1}$; for each variable v_i, write $\dot{v}_i \doteq \frac{\partial v_i}{\partial x_1}$
- for each node, sum up partials over all incoming edges, e.g., $\dot{v}_4 = \frac{\partial v_4}{\partial v_1}\dot{v}_1 + \frac{\partial v_4}{\partial v_3}\dot{v}_3$

A multivariate example — forward mode

$$y = \left(\sin\frac{x_1}{x_2} + \frac{x_1}{x_2} - e^{x_2}\right)\left(\frac{x_1}{x_2} - e^{x_2}\right)$$



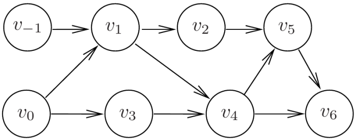
v_{-1}	$=$	x_1	$=$	1.5000		
v_0	$=$	x_2	$=$	0.5000		
v_1	$=$	v_{-1}/v_0	$=$	1.5000/0.5000	$=$	3.0000
v_2	$=$	$\sin(v_1)$	$=$	$\sin(3.0000)$	$=$	0.1411
v_3	$=$	$\exp(v_0)$	$=$	$\exp(0.5000)$	$=$	1.6487
v_4	$=$	$v_1 - v_3$	$=$	$3.0000 - 1.6487$	$=$	1.3513
v_5	$=$	$v_2 + v_4$	$=$	$0.1411 + 1.3513$	$=$	1.4924
v_6	$=$	$v_5 * v_4$	$=$	$1.4924 * 1.3513$	$=$	2.0167
y	$=$	v_6	$=$	2.0167		

$v_{-1} = x_1$	$= 1.5000$		
$\dot{v}_{-1} = \dot{x}_1$	$= 1.0000$		
$v_0 = x_2$	$= 0.5000$		
$\dot{v}_0 = \dot{x}_2$	$= 0.0000$		
$v_1 = v_{-1}/v_0$	$= 1.5000/0.5000$	$=$	3.0000
$\dot{v}_1 = (\dot{v}_{-1} - v_1 * \dot{v}_0)/v_0$	$= 1.0000/0.5000$	$=$	2.0000
$v_2 = \sin(v_1)$	$= \sin(3.0000)$	$=$	0.1411
$\dot{v}_2 = \cos(v_1) * \dot{v}_1$	$= -0.9900 * 2.0000$	$=$	-1.9800
$v_3 = \exp(v_0)$	$= \exp(0.5000)$	$=$	1.6487
$\dot{v}_3 = v_3 * \dot{v}_0$	$= 1.6487 * 0.0000$	$=$	0.0000
$v_4 = v_1 - v_3$	$= 3.0000 - 1.6487$	$=$	1.3513
$\dot{v}_4 = \dot{v}_1 - \dot{v}_3$	$= 2.0000 - 0.0000$	$=$	2.0000
$v_5 = v_2 + v_4$	$= 0.1411 + 1.3513$	$=$	1.4924
$\dot{v}_5 = \dot{v}_2 + \dot{v}_4$	$= -1.9800 + 2.0000$	$=$	0.0200
$v_6 = v_5 * v_4$	$= 1.4924 * 1.3513$	$=$	2.0167
$\dot{v}_6 = \dot{v}_5 * v_4 + v_5 * \dot{v}_4$	$= 0.0200 * 1.3513 + 1.4924 * 2.0000$	$=$	3.0118
$y = v_6$	$= 2.0100$		
$\dot{y} = \dot{v}_6$	$= 3.0110$		

- interested in $\frac{\partial}{\partial x_1}$; for each variable v_i, write $\dot{v}_i \doteq \frac{\partial v_i}{\partial x_1}$
- for each node, sum up partials over all incoming edges, e.g., $\dot{v}_4 = \frac{\partial v_4}{\partial v_1}\dot{v}_1 + \frac{\partial v_4}{\partial v_3}\dot{v}_3$
- complexity:

A multivariate example — forward mode

$$y = \left(\sin\frac{x_1}{x_2} + \frac{x_1}{x_2} - e^{x_2}\right)\left(\frac{x_1}{x_2} - e^{x_2}\right)$$



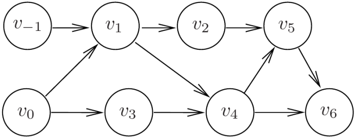
v_{-1}	$=$	x_1	$=$	1.5000		
v_0	$=$	x_2	$=$	0.5000		
v_1	$=$	v_{-1}/v_0	$=$	$1.5000/0.5000$	$=$	3.0000
v_2	$=$	$\sin(v_1)$	$=$	$\sin(3.0000)$	$=$	0.1411
v_3	$=$	$\exp(v_0)$	$=$	$\exp(0.5000)$	$=$	1.6487
v_4	$=$	$v_1 - v_3$	$=$	$3.0000 - 1.6487$	$=$	1.3513
v_5	$=$	$v_2 + v_4$	$=$	$0.1411 + 1.3513$	$=$	1.4924
v_6	$=$	$v_5 * v_4$	$=$	$1.4924 * 1.3513$	$=$	2.0167
y	$=$	v_6	$=$	2.0167		

$v_{-1} = x_1$	$= 1.5000$		
$\dot v_{-1} = \dot x_1$	$= 1.0000$		
$v_0 = x_2$	$= 0.5000$		
$\dot v_0 = \dot x_2$	$= 0.0000$		
$v_1 = v_{-1}/v_0$	$= 1.5000/0.5000$	$=$	3.0000
$\dot v_1 = (\dot v_{-1} - v_1 * \dot v_0)/v_0$	$= 1.0000/0.5000$	$=$	2.0000
$v_2 = \sin(v_1)$	$= \sin(3.0000)$	$=$	0.1411
$\dot v_2 = \cos(v_1) * \dot v_1$	$= -0.9900 * 2.0000$	$=$	-1.9800
$v_3 = \exp(v_0)$	$= \exp(0.5000)$	$=$	1.6487
$\dot v_3 = v_3 * \dot v_0$	$= 1.6487 * 0.0000$	$=$	0.0000
$v_4 = v_1 - v_3$	$= 3.0000 - 1.6487$	$=$	1.3513
$\dot v_4 = \dot v_1 - \dot v_3$	$= 2.0000 - 0.0000$	$=$	2.0000
$v_5 = v_2 + v_4$	$= 0.1411 + 1.3513$	$=$	1.4924
$\dot v_5 = \dot v_2 + \dot v_4$	$= -1.9800 + 2.0000$	$=$	0.0200
$v_6 = v_5 * v_4$	$= 1.4924 * 1.3513$	$=$	2.0167
$\dot v_6 = \dot v_5 * v_4 + v_5 * \dot v_4$	$= 0.0200 * 1.3513 + 1.4924 * 2.0000$	$=$	3.0118
$y = v_6$	$= 2.0100$		
$\dot y = \dot v_6$	$= 3.0110$		

- interested in $\frac{\partial}{\partial x_1}$; for each variable v_i, write $\dot v_i \doteq \frac{\partial v_i}{\partial x_1}$
- for each node, sum up partials over all incoming edges, e.g., $\dot v_4 = \frac{\partial v_4}{\partial v_1}\dot v_1 + \frac{\partial v_4}{\partial v_3}\dot v_3$
- complexity: $O(\#\text{edges} + \#\text{nodes})$

A multivariate example — forward mode

$$y = \left(\sin \frac{x_1}{x_2} + \frac{x_1}{x_2} - e^{x_2}\right)\left(\frac{x_1}{x_2} - e^{x_2}\right)$$

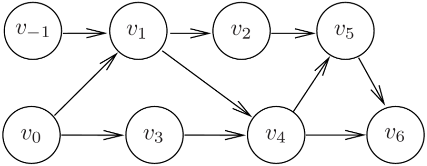


v_{-1}	$=$	x_1	$=$	1.5000		
v_0	$=$	x_2	$=$	0.5000		
v_1	$=$	v_{-1}/v_0	$=$	$1.5000/0.5000$	$=$	3.0000
v_2	$=$	$\sin(v_1)$	$=$	$\sin(3.0000)$	$=$	0.1411
v_3	$=$	$\exp(v_0)$	$=$	$\exp(0.5000)$	$=$	1.6487
v_4	$=$	$v_1 - v_3$	$=$	$3.0000 - 1.6487$	$=$	1.3513
v_5	$=$	$v_2 + v_4$	$=$	$0.1411 + 1.3513$	$=$	1.4924
v_6	$=$	$v_5 * v_4$	$=$	$1.4924 * 1.3513$	$=$	2.0167
y	$=$	v_6	$=$	2.0167		

v_{-1}	$= x_1$	$= 1.5000$	
$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1.0000$	
v_0	$= x_2$	$= 0.5000$	
$\dot{v}_0$	$= \dot{x}_2$	$= 0.0000$	
v_1	$= v_{-1}/v_0$	$= 1.5000/0.5000$	$= 3.0000$
$\dot{v}_1$	$= (\dot{v}_{-1} - v_1 * \dot{v}_0)/v_0$	$= 1.0000/0.5000$	$= 2.0000$
v_2	$= \sin(v_1)$	$= \sin(3.0000)$	$= 0.1411$
$\dot{v}_2$	$= \cos(v_1) * \dot{v}_1$	$= -0.9900 * 2.0000$	$= -1.9800$
v_3	$= \exp(v_0)$	$= \exp(0.5000)$	$= 1.6487$
$\dot{v}_3$	$= v_3 * \dot{v}_0$	$= 1.6487 * 0.0000$	$= 0.0000$
v_4	$= v_1 - v_3$	$= 3.0000 - 1.6487$	$= 1.3513$
$\dot{v}_4$	$= \dot{v}_1 - \dot{v}_3$	$= 2.0000 - 0.0000$	$= 2.0000$
v_5	$= v_2 + v_4$	$= 0.1411 + 1.3513$	$= 1.4924$
$\dot{v}_5$	$= \dot{v}_2 + \dot{v}_4$	$= -1.9800 + 2.0000$	$= 0.0200$
v_6	$= v_5 * v_4$	$= 1.4924 * 1.3513$	$= 2.0167$
$\dot{v}_6$	$= \dot{v}_5 * v_4 + v_5 * \dot{v}_4$	$= 0.0200 * 1.3513 + 1.4924 * 2.0000$	$= 3.0118$
y	$= v_6$	$= 2.0100$	
$\dot{y}$	$= \dot{v}_6$	$= 3.0110$	

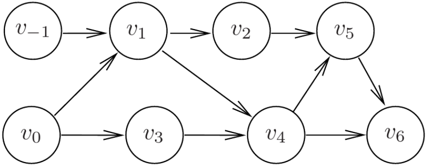
- interested in $\frac{\partial}{\partial x_1}$; for each variable v_i, write $\dot{v}_i \doteq \frac{\partial v_i}{\partial x_1}$
- for each node, sum up partials over all incoming edges, e.g., $\dot{v}_4 = \frac{\partial v_4}{\partial v_1}\dot{v}_1 + \frac{\partial v_4}{\partial v_3}\dot{v}_3$
- complexity: $O\,(\#\text{edges} + \#\text{nodes})$
- for $f : \mathbb{R}^n \to \mathbb{R}^m$, make n forward passes: $O\,(n\,(\#\text{edges} + \#\text{nodes}))$

A multivariate example — reverse mode



$$
\begin{aligned}
v_{-1} &= x_1 = 1.5000\\
v_0 &= x_2 = 0.5000\\
v_1 &= v_{-1}/v_0 = 1.5000/0.5000 = 3.0000\\
v_2 &= \sin(v_1) = \sin(3.0000) = 0.1411\\
v_3 &= \exp(v_0) = \exp(0.5000) = 1.6487\\
v_4 &= v_1 - v_3 = 3.0000 - 1.6487 = 1.3513\\
v_5 &= v_2 + v_4 = 0.1411 + 1.3513 = 1.4924\\
v_6 &= v_5 * v_4 = 1.4924 * 1.3513 = 2.0167\\
y &= v_6 = 2.0167\\
\bar{v}_6 &= \bar{y} = 1.0000\\
\bar{v}_5 &= \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513\\
\bar{v}_4 &= \bar{v}_6 * v_5 = 1.0000 * 1.4924 = 1.4924\\
\bar{v}_4 &= \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437\\
\bar{v}_2 &= \bar{v}_5 = 1.3513\\
\bar{v}_3 &= -\bar{v}_4 = -2.8437\\
\bar{v}_1 &= \bar{v}_4 = 2.8437\\
\bar{v}_0 &= \bar{v}_3 * v_3 = -2.8437 * 1.6487 = -4.6884\\
\bar{v}_1 &= \bar{v}_1 + \bar{v}_2 * \cos(v_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059\\
\bar{v}_0 &= \bar{v}_0 - \bar{v}_1 * v_1/v_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239\\
\bar{v}_{-1} &= \bar{v}_1/v_0 = 1.5059/0.5000 = 3.0118\\
\bar{x}_2 &= \bar{v}_0 = -13.7239\\
\bar{x}_1 &= \bar{v}_{-1} = 3.0118
\end{aligned}
$$

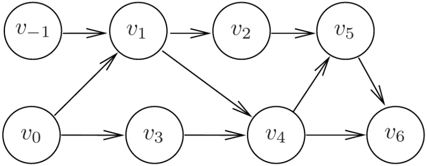
A multivariate example — reverse mode



$$
\begin{aligned}
&v_{-1} = x_1 = 1.5000\\
&\quad v_0 = x_2 = 0.5000\\
&\quad\quad v_1 = v_{-1}/v_0 = 1.5000/0.5000 = 3.0000\\
&\quad\quad\quad v_2 = \sin(v_1) = \sin(3.0000) = 0.1411\\
&\quad\quad\quad\quad v_3 = \exp(v_0) = \exp(0.5000) = 1.6487\\
&\quad\quad\quad\quad\quad v_4 = v_1 - v_3 = 3.0000 - 1.6487 = 1.3513\\
&\quad\quad\quad\quad\quad\quad v_5 = v_2 + v_4 = 0.1411 + 1.3513 = 1.4924\\
&\quad\quad\quad\quad\quad\quad\quad v_6 = v_5 * v_4 = 1.4924 * 1.3513 = 2.0167\\
&\quad\quad\quad\quad\quad\quad\quad\quad y = v_6 = 2.0167\\
&\quad\quad\quad\quad\quad\quad\quad\quad \bar{v}_6 = \bar{y} = 1.0000\\
&\quad\quad\quad\quad\quad\quad\quad \bar{v}_5 = \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513\\
&\quad\quad\quad\quad\quad\quad\quad \bar{v}_4 = \bar{v}_6 * v_5 = 1.0000 * 1.4924 = 1.4924\\
&\quad\quad\quad\quad\quad\quad \bar{v}_4 = \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437\\
&\quad\quad\quad\quad\quad\quad \bar{v}_2 = \bar{v}_5 = 1.3513\\
&\quad\quad\quad\quad\quad \bar{v}_3 = -\bar{v}_4 = -2.8437\\
&\quad\quad\quad\quad\quad \bar{v}_1 = \bar{v}_4 = 2.8437\\
&\quad\quad\quad\quad \bar{v}_0 = \bar{v}_3 * v_3 = -2.8437 * 1.6487 = -4.6884\\
&\quad\quad\quad \bar{v}_1 = \bar{v}_1 + \bar{v}_2 * \cos(v_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059\\
&\quad\quad \bar{v}_0 = \bar{v}_0 - \bar{v}_1 * v_1/v_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239\\
&\quad\quad \bar{v}_{-1} = \bar{v}_1/v_0 = 1.5059/0.5000 = 3.0118\\
&\quad \bar{x}_2 = \bar{v}_0 = -13.7239\\
&\bar{x}_1 = \bar{v}_{-1} = 3.0118
\end{aligned}
$$

– interested in $\frac{\partial y}{\partial}$; for each variable v_i, write $\overline{v}_i \doteq \frac{\partial y}{\partial v_i}$ (called **adjoint variable**)

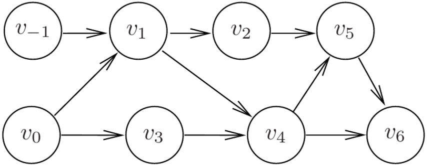
A multivariate example — reverse mode



$$
\begin{aligned}
v_{-1} &= x_1 = 1.5000\\
v_0 &= x_2 = 0.5000\\
v_1 &= v_{-1}/v_0 = 1.5000/0.5000 = 3.0000\\
v_2 &= \sin(v_1) = \sin(3.0000) = 0.1411\\
v_3 &= \exp(v_0) = \exp(0.5000) = 1.6487\\
v_4 &= v_1 - v_3 = 3.0000 - 1.6487 = 1.3513\\
v_5 &= v_2 + v_4 = 0.1411 + 1.3513 = 1.4924\\
v_6 &= v_5 * v_4 = 1.4924 * 1.3513 = 2.0167\\
y &= v_6 = 2.0167\\
\bar{v}_6 &= \bar{y} = 1.0000\\
\bar{v}_5 &= \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513\\
\bar{v}_4 &= \bar{v}_6 * v_5 = 1.0000 * 1.4924 = 1.4924\\
\bar{v}_4 &= \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437\\
\bar{v}_2 &= \bar{v}_5 = 1.3513\\
\bar{v}_3 &= -\bar{v}_4 = -2.8437\\
\bar{v}_1 &= \bar{v}_4 = 2.8437\\
\bar{v}_0 &= \bar{v}_3 * v_3 = -2.8437 * 1.6487 = -4.6884\\
\bar{v}_1 &= \bar{v}_1 + \bar{v}_2 * \cos(v_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059\\
\bar{v}_0 &= \bar{v}_0 - \bar{v}_1 * v_1/v_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239\\
\bar{v}_{-1} &= \bar{v}_1/v_0 = 1.5059/0.5000 = 3.0118\\
\bar{x}_2 &= \bar{v}_0 = -13.7239\\
\bar{x}_1 &= \bar{v}_{-1} = 3.0118
\end{aligned}
$$

- interested in $\frac{\partial y}{\partial}$; for each variable v_i, write $\overline{v}_i \doteq \frac{\partial y}{\partial v_i}$ (called **adjoint variable**)
- for each node, sum up partials over all outgoing edges, e.g., $\overline{v}_4 = \frac{\partial v_5}{\partial v_4}\overline{v}_5 + \frac{\partial v_6}{\partial v_4}\overline{v}_6$

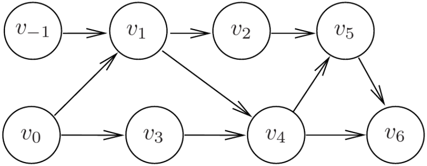
A multivariate example — reverse mode



$$
\begin{aligned}
v_{-1} &= x_1 = 1.5000\\
v_0 &= x_2 = 0.5000\\
v_1 &= v_{-1}/v_0 = 1.5000/0.5000 = 3.0000\\
v_2 &= \sin(v_1) = \sin(3.0000) = 0.1411\\
v_3 &= \exp(v_0) = \exp(0.5000) = 1.6487\\
v_4 &= v_1 - v_3 = 3.0000 - 1.6487 = 1.3513\\
v_5 &= v_2 + v_4 = 0.1411 + 1.3513 = 1.4924\\
v_6 &= v_5 * v_4 = 1.4924 * 1.3513 = 2.0167\\
y &= v_6 = 2.0167\\
\bar{v}_6 &= \bar{y} = 1.0000\\
\bar{v}_5 &= \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513\\
\bar{v}_4 &= \bar{v}_6 * v_5 = 1.0000 * 1.4924 = 1.4924\\
\bar{v}_4 &= \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437\\
\bar{v}_2 &= \bar{v}_5 = 1.3513\\
\bar{v}_3 &= -\bar{v}_4 = -2.8437\\
\bar{v}_1 &= \bar{v}_4 = 2.8437\\
\bar{v}_0 &= \bar{v}_3 * v_3 = -2.8437 * 1.6487 = -4.6884\\
\bar{v}_1 &= \bar{v}_1 + \bar{v}_2 * \cos(v_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059\\
\bar{v}_0 &= \bar{v}_0 - \bar{v}_1 * v_1/v_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239\\
\bar{v}_{-1} &= \bar{v}_1/v_0 = 1.5059/0.5000 = 3.0118\\
\bar{x}_2 &= \bar{v}_0 = -13.7239\\
\bar{x}_1 &= \bar{v}_{-1} = 3.0118
\end{aligned}
$$

- interested in $\frac{\partial y}{\partial}$; for each variable v_i, write $\overline{v}_i \doteq \frac{\partial y}{\partial v_i}$ (called **adjoint variable**)
- for each node, sum up partials over all outgoing edges, e.g., $\overline{v}_4 = \frac{\partial v_5}{\partial v_4}\overline{v}_5 + \frac{\partial v_6}{\partial v_4}\overline{v}_6$
- complexity:

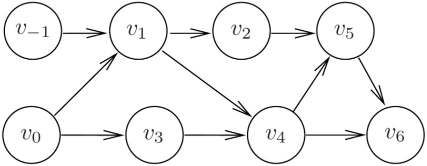
A multivariate example — reverse mode



$$
\begin{aligned}
v_{-1} &= x_1 = 1.5000\\
v_0 &= x_2 = 0.5000\\
v_1 &= v_{-1}/v_0 = 1.5000/0.5000 = 3.0000\\
v_2 &= \sin(v_1) = \sin(3.0000) = 0.1411\\
v_3 &= \exp(v_0) = \exp(0.5000) = 1.6487\\
v_4 &= v_1 - v_3 = 3.0000 - 1.6487 = 1.3513\\
v_5 &= v_2 + v_4 = 0.1411 + 1.3513 = 1.4924\\
v_6 &= v_5 * v_4 = 1.4924 * 1.3513 = 2.0167\\
y &= v_6 = 2.0167\\
\bar{v}_6 &= \bar{y} = 1.0000\\
\bar{v}_5 &= \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513\\
\bar{v}_4 &= \bar{v}_6 * v_5 = 1.0000 * 1.4924 = 1.4924\\
\bar{v}_4 &= \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437\\
\bar{v}_2 &= \bar{v}_5 = 1.3513\\
\bar{v}_3 &= -\bar{v}_4 = -2.8437\\
\bar{v}_1 &= \bar{v}_4 = 2.8437\\
\bar{v}_0 &= \bar{v}_3 * v_3 = -2.8437 * 1.6487 = -4.6884\\
\bar{v}_1 &= \bar{v}_1 + \bar{v}_2 * \cos(v_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059\\
\bar{v}_0 &= \bar{v}_0 - \bar{v}_1 * v_1/v_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239\\
\bar{v}_{-1} &= \bar{v}_1/v_0 = 1.5059/0.5000 = 3.0118\\
\bar{x}_2 &= \bar{v}_0 = -13.7239\\
\bar{x}_1 &= \bar{v}_{-1} = 3.0118
\end{aligned}
$$

- interested in $\frac{\partial y}{\partial}$; for each variable v_i, write $\overline{v}_i \doteq \frac{\partial y}{\partial v_i}$ (called **adjoint variable**)
- for each node, sum up partials over all outgoing edges, e.g., $\overline{v}_4 = \frac{\partial v_5}{\partial v_4}\overline{v}_5 + \frac{\partial v_6}{\partial v_4}\overline{v}_6$
- complexity: O (#edges + #nodes)

A multivariate example — reverse mode



$$
\begin{aligned}
&v_{-1} = x_1 = 1.5000\\
&v_0 = x_2 = 0.5000\\
&v_1 = v_{-1}/v_0 = 1.5000/0.5000 = 3.0000\\
&v_2 = \sin(v_1) = \sin(3.0000) = 0.1411\\
&v_3 = \exp(v_0) = \exp(0.5000) = 1.6487\\
&v_4 = v_1 - v_3 = 3.0000 - 1.6487 = 1.3513\\
&v_5 = v_2 + v_4 = 0.1411 + 1.3513 = 1.4924\\
&v_6 = v_5 * v_4 = 1.4924 * 1.3513 = 2.0167\\
&y = v_6 = 2.0167\\
&\bar{v}_6 = \bar{y} = 1.0000\\
&\bar{v}_5 = \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513\\
&\bar{v}_4 = \bar{v}_6 * v_5 = 1.0000 * 1.4924 = 1.4924\\
&\bar{v}_4 = \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437\\
&\bar{v}_2 = \bar{v}_5 = 1.3513\\
&\bar{v}_3 = -\bar{v}_4 = -2.8437\\
&\bar{v}_1 = \bar{v}_4 = 2.8437\\
&\bar{v}_0 = \bar{v}_3 * v_3 = -2.8437 * 1.6487 = -4.6884\\
&\bar{v}_1 = \bar{v}_1 + \bar{v}_2 * \cos(v_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059\\
&\bar{v}_0 = \bar{v}_0 - \bar{v}_1 * v_1/v_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239\\
&\bar{v}_{-1} = \bar{v}_1/v_0 = 1.5059/0.5000 = 3.0118\\
&\bar{x}_2 = \bar{v}_0 = -13.7239\\
&\bar{x}_1 = \bar{v}_{-1} = 3.0118
\end{aligned}
$$

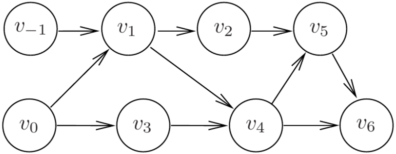
- interested in $\frac{\partial y}{\partial}$; for each variable v_i, write $\overline{v}_i \doteq \frac{\partial y}{\partial v_i}$ (called **adjoint variable**)
- for each node, sum up partials over all outgoing edges, e.g., $\overline{v}_4 = \frac{\partial v_5}{\partial v_4}\overline{v}_5 + \frac{\partial v_6}{\partial v_4}\overline{v}_6$
- complexity: $O\,(\#\text{edges} + \#\text{nodes})$
- for $f : \mathbb{R}^n \to \mathbb{R}^m$, make n forward passes: $O\,(m\,(\#\text{edges} + \#\text{nodes}))$

example from Ch 1 of [Griewank and Walther, 2008]

Forward vs. reverse modes

For general function $f : \mathbb{R}^n \to \mathbb{R}^m$, suppose there is no loop in the computational graph, i.e., **acyclic graph**.

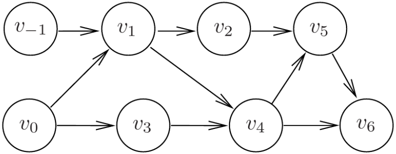
Define E: set of edges ; V: set of nodes



Forward vs. reverse modes

For general function $f : \mathbb{R}^n \to \mathbb{R}^m$, suppose there is no loop in the computational graph, i.e., **acyclic graph**.

Define E: set of edges ; V: set of nodes



	forward mode	**reverse mode**
start from	roots	leaves
end with	leaves	roots
invariants	$\dot{v}_i \doteq \frac{\partial v_i}{\partial x}$ (x—root of interest)	$\overline{v}_i \doteq \frac{\partial y}{\partial v_i}$ (y—leaf of interest)
rule	sum over incoming edges	sum over outgoing edges
complexity	$O(n\,\lvert E\rvert + n\,\lvert V\rvert)$	$O(m\,\lvert E\rvert + m\,\lvert V\rvert)$
better when	$m \gg n$	$n \gg m$

Directional derivatives

Consider $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$. Let v_s's be the variables in its computational graph. Particularly, $v_{n-1} = x_1, v_{n-2} = x_2, \ldots, v_0 = x_n$. $D_p(\cdot)$ means directional derivative wrt $\boldsymbol{p}$. In practical implementations,

Directional derivatives

Consider $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$. Let v_s's be the variables in its computational graph. Particularly, $v_{n-1} = x_1, v_{n-2} = x_2, \ldots, v_0 = x_n$. $D_p(\cdot)$ means directional derivative wrt $\boldsymbol{p}$. In practical implementations,

forward mode: compute $\boldsymbol{J}_f \boldsymbol{p}$, i.e., Jacobian-vector product

Directional derivatives

Consider $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$. Let v_s's be the variables in its computational graph. Particularly, $v_{n-1} = x_1, v_{n-2} = x_2, \dots, v_0 = x_n$. $D_p(\cdot)$ means directional derivative wrt $\boldsymbol{p}$. In practical implementations,

forward mode: compute $\boldsymbol{J}_f \boldsymbol{p}$, i.e., Jacobian-vector product

- Why? (1) Columns of $\boldsymbol{J}_f$ can be obtained by setting $\boldsymbol{p} = \boldsymbol{e}_1, \dots, \boldsymbol{e}_n$. (2) When $\boldsymbol{J}_f$ has special structures (e.g., sparsity), save computation by judicious choices of $\boldsymbol{p}$'s (3) Problem may only need $\boldsymbol{J}_f \boldsymbol{p}$ for a specific $\boldsymbol{p}$, not $\boldsymbol{J}_f$ itself—save computation again

Directional derivatives

Consider $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$. Let v_s's be the variables in its computational graph. Particularly, $v_{n-1} = x_1, v_{n-2} = x_2, \dots, v_0 = x_n$. $D_p(\cdot)$ means directional derivative wrt $\boldsymbol{p}$. In practical implementations,

forward mode: compute $\boldsymbol{J}_f \boldsymbol{p}$, i.e., Jacobian-vector product

- Why? (1) Columns of $\boldsymbol{J}_f$ can be obtained by setting $\boldsymbol{p} = \boldsymbol{e}_1, \dots, \boldsymbol{e}_n$. (2) When $\boldsymbol{J}_f$ has special structures (e.g., sparsity), save computation by judicious choices of $\boldsymbol{p}$'s (3) Problem may only need $\boldsymbol{J}_f \boldsymbol{p}$ for a specific $\boldsymbol{p}$, not $\boldsymbol{J}_f$ itself—save computation again
- How? (1) initialize $D_{\boldsymbol{p}} v_{n-1} = p_1, \dots, D_{\boldsymbol{p}} v_0 = p_n$. (2) apply chain rule:

$$\nabla_{\boldsymbol{x}} v_i = \sum_{j:\text{incoming}} \frac{\partial v_i}{\partial v_j} \nabla_{\boldsymbol{x}} v_j \Longrightarrow D_{\boldsymbol{p}} v_i = \sum_{j:\text{incoming}} \frac{\partial v_i}{\partial v_j} D_{\boldsymbol{p}} v_j$$

Directional derivatives

Consider $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$. Let v_s's be the variables in its computational graph. Particularly, $v_{n-1} = x_1, v_{n-2} = x_2, \dots, v_0 = x_n$. $D_p(\cdot)$ means directional derivative wrt $\boldsymbol{p}$. In practical implementations,

forward mode: compute $\boldsymbol{J}_f \boldsymbol{p}$, i.e., Jacobian-vector product

- Why? (1) Columns of $\boldsymbol{J}_f$ can be obtained by setting $\boldsymbol{p} = \boldsymbol{e}_1, \dots, \boldsymbol{e}_n$. (2) When $\boldsymbol{J}_f$ has special structures (e.g., sparsity), save computation by judicious choices of $\boldsymbol{p}$'s (3) Problem may only need $\boldsymbol{J}_f \boldsymbol{p}$ for a specific $\boldsymbol{p}$, not $\boldsymbol{J}_f$ itself—save computation again
- How? (1) initialize $D_{\boldsymbol{p}} v_{n-1} = p_1, \dots, D_{\boldsymbol{p}} v_0 = p_n$. (2) apply chain rule:

$$\nabla_{\boldsymbol{x}} v_i = \sum_{j:\text{incoming}} \frac{\partial v_i}{\partial v_j} \nabla_{\boldsymbol{x}} v_j \Longrightarrow D_{\boldsymbol{p}} v_i = \sum_{j:\text{incoming}} \frac{\partial v_i}{\partial v_j} D_{\boldsymbol{p}} v_j$$

reverse mode: compute $\boldsymbol{J}_f^\intercal \boldsymbol{q} = \nabla_{\boldsymbol{x}} (f^\intercal \boldsymbol{q})$, i.e., Jacobian-trans-vector product

Directional derivatives

Consider $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$. Let v_s's be the variables in its computational graph. Particularly, $v_{n-1} = x_1, v_{n-2} = x_2, \ldots, v_0 = x_n$. $D_p(\cdot)$ means directional derivative wrt $\boldsymbol{p}$. In practical implementations,

forward mode: compute $\boldsymbol{J}_f \boldsymbol{p}$, i.e., Jacobian-vector product

- Why? (1) Columns of $\boldsymbol{J}_f$ can be obtained by setting $\boldsymbol{p} = \boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$. (2) When $\boldsymbol{J}_f$ has special structures (e.g., sparsity), save computation by judicious choices of $\boldsymbol{p}$'s (3) Problem may only need $\boldsymbol{J}_f \boldsymbol{p}$ for a specific $\boldsymbol{p}$, not $\boldsymbol{J}_f$ itself—save computation again
- How? (1) initialize $D_{\boldsymbol{p}} v_{n-1} = p_1, \ldots, D_{\boldsymbol{p}} v_0 = p_n$. (2) apply chain rule:

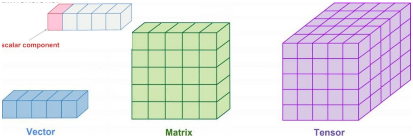
$$\nabla_{\boldsymbol{x}} v_i = \sum_{j:\text{incoming}} \frac{\partial v_i}{\partial v_j} \nabla_{\boldsymbol{x}} v_j \Longrightarrow D_{\boldsymbol{p}} v_i = \sum_{j:\text{incoming}} \frac{\partial v_i}{\partial v_j} D_{\boldsymbol{p}} v_j$$

reverse mode: compute $\boldsymbol{J}_f^\intercal \boldsymbol{q} = \nabla_{\boldsymbol{x}} (f^\intercal \boldsymbol{q})$, i.e., Jacobian-trans-vector product

- Why? Similar to the above
- How? Track $\frac{d}{dv_i}(f^\intercal \boldsymbol{q})$: $\frac{d}{dv_i}(f^\intercal \boldsymbol{q}) = \sum_{k:\text{outgoing}} \frac{\partial v_k}{\partial v_i} \frac{d}{dv_k}(f^\intercal \boldsymbol{q})$

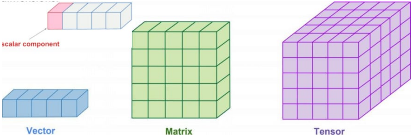
Tensor abstraction

Tensors: multi-dimensional arrays



Tensor abstraction

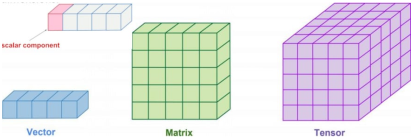
Tensors: multi-dimensional arrays



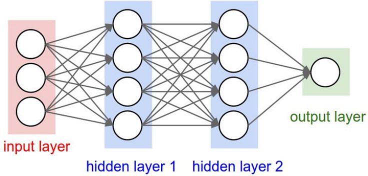
Each node in the computational graph can be a tensor (scalar, vector, matrix, 3-D tensor, ...)

Tensor abstraction

Tensors: multi-dimensional arrays



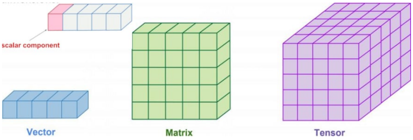
Each node in the computational graph can be a tensor (scalar, vector, matrix, 3-D tensor, ...)



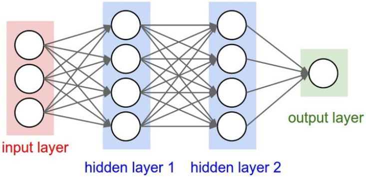
$$f(\boldsymbol{W}) = \|\boldsymbol{Y} - \sigma(\boldsymbol{W}_k \sigma(\boldsymbol{W}_{k-1} \sigma \dots (\boldsymbol{W}_1 \boldsymbol{X})))\|_F^2$$

Tensor abstraction

Tensors: multi-dimensional arrays

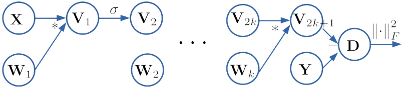


Each node in the computational graph can be a tensor (scalar, vector, matrix, 3-D tensor, ...)

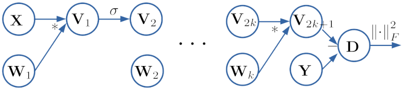


$$f(\boldsymbol{W}) = \|\boldsymbol{Y} - \sigma(\boldsymbol{W}_k \sigma(\boldsymbol{W}_{k-1}\sigma \ldots (\boldsymbol{W}_1 \boldsymbol{X})))\|_F^2$$

computational graph for DNN

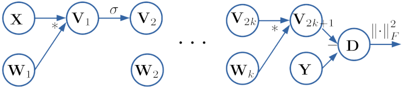


Tensor abstraction



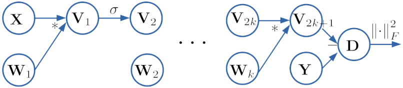
– Abstract out low-level details; operations are often simple $e.g., *, \sigma$ so partials are simple

Tensor abstraction



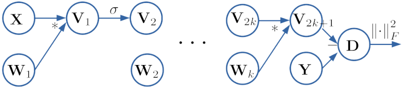
- Abstract out low-level details; operations are often simple $e.g., *, \sigma$ so partials are simple
- Tensor (i.e., vector) chain rules apply, often via tensor-free computation

Tensor abstraction



- Abstract out low-level details; operations are often simple $e.g., *, \sigma$ so partials are simple
- Tensor (i.e., vector) chain rules apply, often via tensor-free computation
- Basis of implementation for: Tensorflow, Pytorch, Jax, etc
  Jax: https://github.com/google/jax

Tensor abstraction



– Abstract out low-level details; operations are often simple $e.g., *, \sigma$ so partials are simple

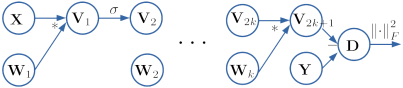
– Tensor (i.e., vector) chain rules apply, often via tensor-free computation

– Basis of implementation for: Tensorflow, Pytorch, Jax, etc
Jax: https://github.com/google/jax

Good to know:

– In practice, graphs are built automatically by software

Tensor abstraction



– Abstract out low-level details; operations are often simple $e.g., *, \sigma$ so partials are simple

– Tensor (i.e., vector) chain rules apply, often via tensor-free computation

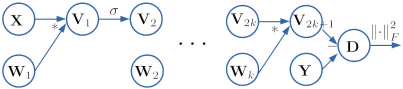
– Basis of implementation for: Tensorflow, Pytorch, Jax, etc
  Jax: https://github.com/google/jax

Good to know:

– In practice, graphs are built automatically by software

– Higher-order derivatives can also be done, particularly Hessian-vector product $\nabla^2 f(\boldsymbol{x})\,\boldsymbol{v}$ (Check out Jax!)

Tensor abstraction

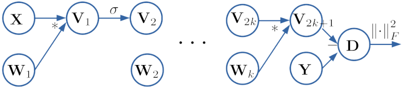


- Abstract out low-level details; operations are often simple $e.g., *, \sigma$ so partials are simple
- Tensor (i.e., vector) chain rules apply, often via tensor-free computation
- Basis of implementation for: Tensorflow, Pytorch, Jax, etc
  Jax: https://github.com/google/jax

Good to know:

- In practice, graphs are built automatically by software
- Higher-order derivatives can also be done, particularly Hessian-vector product $\nabla^2 f(\boldsymbol{x})\,\boldsymbol{v}$ (Check out Jax!)
- Auto-diff in Tensorflow and Pytorch are specialized to DNNs and focus on 1st order, Jax (in Python) is full fledged and also supports GPU

Tensor abstraction



- Abstract out low-level details; operations are often simple $e.g., *, \sigma$ so partials are simple
- Tensor (i.e., vector) chain rules apply, often via tensor-free computation
- Basis of implementation for: Tensorflow, Pytorch, Jax, etc
  Jax: https://github.com/google/jax

Good to know:

- In practice, graphs are built automatically by software
- Higher-order derivatives can also be done, particularly Hessian-vector product $\nabla^2 f(\boldsymbol{x}) \boldsymbol{v}$ (Check out Jax!)
- Auto-diff in Tensorflow and Pytorch are specialized to DNNs and focus on 1st order, Jax (in Python) is full fledged and also supports GPU
- General resources for autodiff: http://www.autodiff.org/, [Griewank and Walther, 2008]

Autodiff in Pytorch

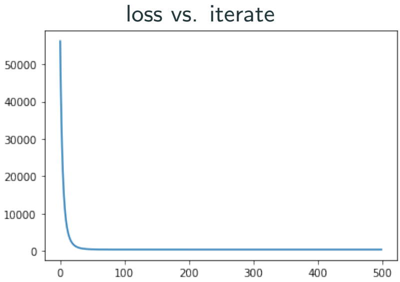
Solve least squares $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$ with $\nabla f(\boldsymbol{x}) = -\boldsymbol{A}^\intercal(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x})$

```python
import torch
import matplotlib.pyplot as plt

dtype = torch.float
device = torch.device("cpu")

n, p = 500, 100

A = torch.randn(n, p, device=device, dtype=dtype)
y = torch.randn(n, device=device, dtype=dtype)

x = torch.randn(p, device=device, dtype=dtype, requires_grad=True)

step_size = 1e-4

num_step = 500
loss_vec = torch.zeros(500, device=device, dtype=dtype)

for t in range(500):
  pred = torch.matmul(A, x)
  loss = torch.pow(torch.norm(y - pred), 2)

  loss_vec[t] = loss.item()

  # one line for computing the gradient
  loss.backward()

  # updates
  with torch.no_grad():
    x -= step_size*x.grad

    # zero the gradient after updating
    x.grad.zero_()

plt.plot(loss_vec.numpy())
```

Autodiff in Pytorch

Solve least squares $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$ with $\nabla f(\boldsymbol{x}) = -\boldsymbol{A}^\intercal(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x})$

```
import torch
import matplotlib.pyplot as plt

dtype = torch.float
device = torch.device("cpu")

n, p = 500, 100

A = torch.randn(n, p, device=device, dtype=dtype)
y = torch.randn(n, device=device, dtype=dtype)

x = torch.randn(p, device=device, dtype=dtype, requires_grad=True)

step_size = 1e-4

num_step = 500
loss_vec = torch.zeros(500, device=device, dtype=dtype)

for t in range(500):
  pred = torch.matmul(A, x)
  loss = torch.pow(torch.norm(y - pred), 2)

  loss_vec[t] = loss.item()

  # one line for computing the gradient
  loss.backward()

  # updates
  with torch.no_grad():
    x -= step_size*x.grad

    # zero the gradient after updating
    x.grad.zero_()

plt.plot(loss_vec.numpy())
```



Autodiff in Pytorch

Train a shallow neural network

$$f(\boldsymbol{W}) = \sum_i \|\boldsymbol{y}_i - \boldsymbol{W}_2 \sigma(\boldsymbol{W}_1 \boldsymbol{x}_i)\|_2^2$$

where $\sigma(z) = \max(z, 0)$, i.e., ReLU

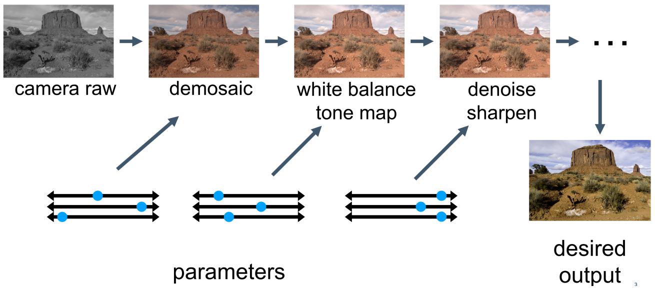
https://pytorch.org/tutorials/beginner/pytorch_with_examples.html

- `torch.mm`
- `torch.clamp`
- `torch.no_grad()`

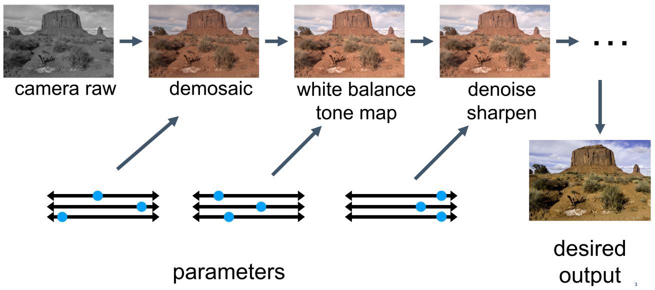
Back propagation is reverse mode auto-differentiation!

Outline

Example: image enhancement

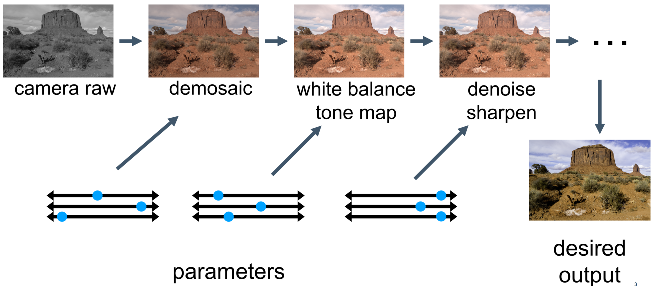


Example: image enhancement



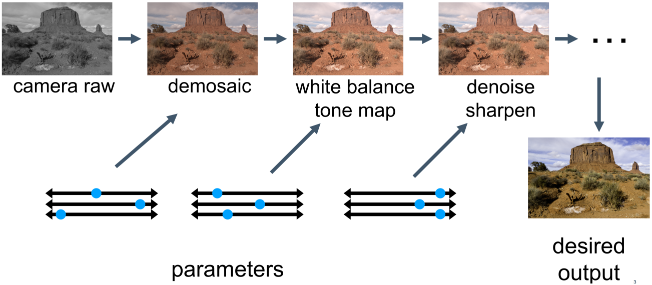
- Each stage applies a parameterized function to the image, i.e., $q_{\boldsymbol{w}_k} \circ \cdots \circ h_{\boldsymbol{w}_3} \circ g_{\boldsymbol{w}_2} \circ f_{\boldsymbol{w}_1}(\boldsymbol{X})$ ($\boldsymbol{X}$ is the camera raw)

Example: image enhancement



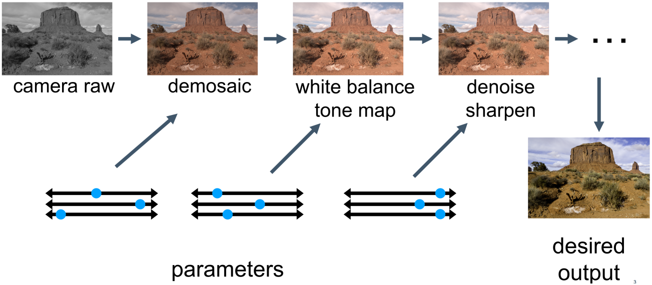
– Each stage applies a parameterized function to the image, i.e., $q_{\boldsymbol{w}_k} \circ \cdots \circ h_{\boldsymbol{w}_3} \circ g_{\boldsymbol{w}_2} \circ f_{\boldsymbol{w}_1}(\boldsymbol{X})$ ($\boldsymbol{X}$ is the camera raw)
– The parameterized functions may or may not be DNNs

Example: image enhancement



- Each stage applies a parameterized function to the image, i.e., $q_{\boldsymbol{w}_k} \circ \cdots \circ h_{\boldsymbol{w}_3} \circ g_{\boldsymbol{w}_2} \circ f_{\boldsymbol{w}_1}(\boldsymbol{X})$ ($\boldsymbol{X}$ is the camera raw)
- The parameterized functions may or may not be DNNs
- Each function may be analytic, or simply a chunk of codes dependent on the parameters

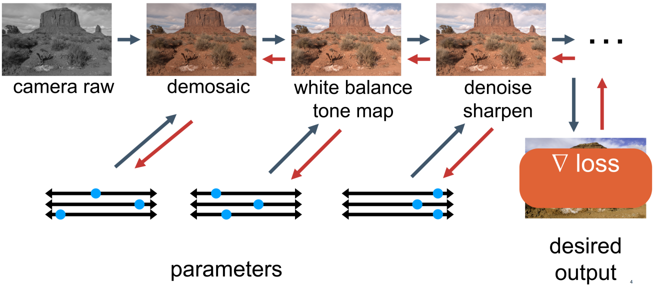
Example: image enhancement



- Each stage applies a parameterized function to the image, i.e., $q_{\boldsymbol{w}_k} \circ \cdots \circ h_{\boldsymbol{w}_3} \circ g_{\boldsymbol{w}_2} \circ f_{\boldsymbol{w}_1}(\boldsymbol{X})$ ($\boldsymbol{X}$ is the camera raw)
- The parameterized functions may or may not be DNNs
- Each function may be analytic, or simply a chunk of codes dependent on the parameters
- $\boldsymbol{w}_i$'s are the trainable parameters

Credit: https://people.csail.mit.edu/tzumao/gradient_halide/

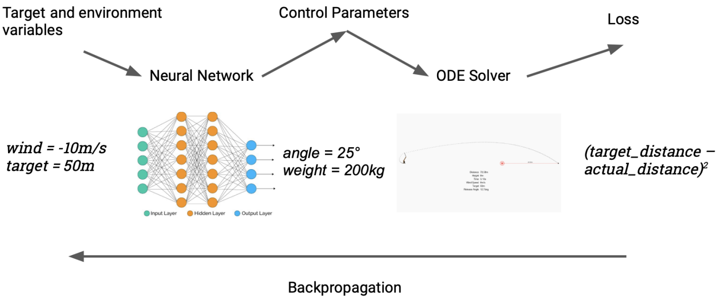
Example: image enhancement



- the trainable parameters are learned by gradient descent based on auto-differentiation
- This is generalization of training DNNs with the classic feedforward structure to training general parameterized functions, using derivative-based methods

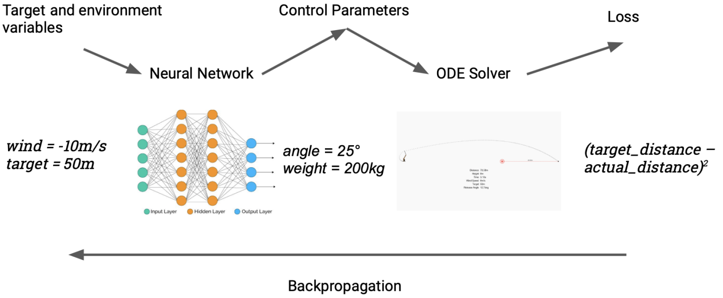
Credit: https://people.csail.mit.edu/tzumao/gradient_halide/

Example: control a trebuchet



https://fluxml.ai/2019/03/05/dp-vs-rl.html

Example: control a trebuchet



https://fluxml.ai/2019/03/05/dp-vs-rl.html

- Given wind speed and target distance, the DNN predicts the **angle of release** and **mass of counterweight**
- Given the angle of release and mass of counterweight as initial conditions, the ODE solver calculates the actual distance by iterative methods
- We perform auto-differentiation through the iterative ODE solver and the DNN

Differential programming

Interesting resources

- Notable implementations: Swift for Tensorflow https://www.tensorflow.org/swift, and Zygote in Julia https://github.com/FluxML/Zygote.jl
- Flux: machine learning package based on Zygote https://fluxml.ai/
- Taichi: differentiable programming language tailored to 3D computer graphics https://github.com/taichi-dev/taichi

Outline

Autodiff in DNNs

- http://neuralnetworksanddeeplearning.com/chap2.html
- https://colah.github.io/posts/2015-08-Backprop/

Differentiable programming

- https://en.wikipedia.org/wiki/Differentiable_programming
- https://fluxml.ai/2019/02/07/what-is-differentiable-programming.html
- https://fluxml.ai/2019/03/05/dp-vs-rl.html

References i

[Baydin et al., 2017] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). **Automatic differentiation in machine learning: a survey.** *The Journal of Machine Learning Research*, 18(1):5595–5637.

[Griewank and Walther, 2008] Griewank, A. and Walther, A. (2008). **Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.** Society for Industrial and Applied Mathematics.