

Introduction to Backpropagation aka Reverse Mode of Algorithmic Differentiation

Andrea Walther and Sri Tadinada
Institut für Mathematik
Humboldt-Universität zu Berlin

Autumn School 2025
Artificial Intelligence in Natural Sciences

Amsterdam, October 31, 2025

Outline

- 1 Motivation and Conventions
- 2 Function Evaluation
- 3 Forward Mode of AD
- 4 Reverse Mode of AD = Backpropagation
- 5 A Few Additional Points
- 6 Conclusion

Where are Derivatives Needed?

- Optimization:

unbounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}$

bounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}, c(x) = 0, h(x) \leq 0$

Where are Derivatives Needed?

- Optimization:

unbounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}$

bounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}, \quad c(x) = 0, \quad h(x) \leq 0$

- Solution of nonlinear equation systems

$$F(x) = 0, \quad F : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

Newton method requires $F'(x) \in \mathbb{R}^{n \times n}$

Where are Derivatives Needed?

- Optimization:

unbounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}$

bounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}, \quad c(x) = 0, \quad h(x) \leq 0$

- Solution of nonlinear equation systems

$$F(x) = 0, \quad F : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

Newton method requires $F'(x) \in \mathbb{R}^{n \times n}$

- Simulation of complex system

- definition

- integration of differential equations using implicit methods

Where are Derivatives Needed?

- Optimization:

unbounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}$

bounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}, \quad c(x) = 0, \quad h(x) \leq 0$

- Solution of nonlinear equation systems

$$F(x) = 0, \quad F : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

Newton method requires $F'(x) \in \mathbb{R}^{n \times n}$

- Simulation of complex system

- definition

- integration of differential equations using implicit methods

- Sensitivity analysis

- Real-time control

Where are Derivatives Needed?

- Optimization:

unbounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}$

bounded: $\min f(x), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}, \quad c(x) = 0, \quad h(x) \leq 0$

- Solution of nonlinear equation systems

$$F(x) = 0, \quad F : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

Newton method requires $F'(x) \in \mathbb{R}^{n \times n}$

- Simulation of complex system

- definition

- integration of differential equations using implicit methods

- Sensitivity analysis

- Real-time control

- machine learning (ML), e.g., Stochastic Gradient Descent, Adam, . . .
target functions quite often nonsmooth!

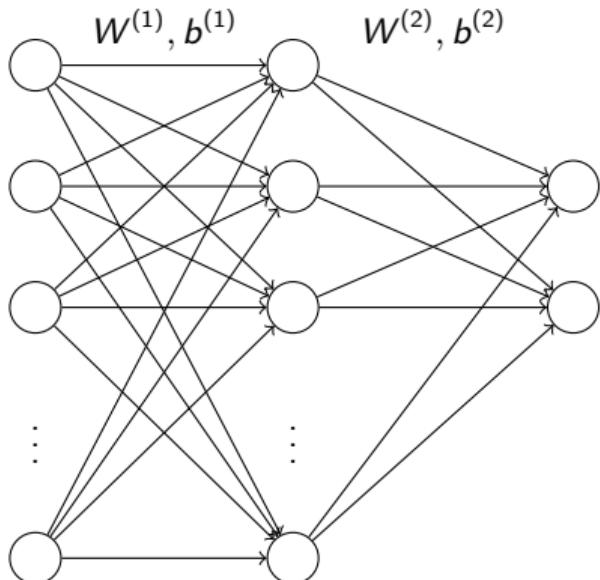
Training of a Deep Neural Network

Goal: Learn weights $W^{(i)}$ and biases $b^{(i)}$ such that the network approximates the true mapping.

Empirical Risk Minimization:

$$\min_{\{W^{(i)}, b^{(i)}\}} \frac{1}{M} \sum_{i=1}^M l(y(x_i), y_{\theta,i}^{NN})$$

- Parameters: $\theta = (W^{(i)}, b^{(i)})_{1 \leq i \leq k}$
- $y(x_i)$: value of true mapping for input x_i
- $y_{\theta,i}^{NN}$: output of the network for input x_i
- l : loss function (e.g. MSE, cross-entropy)



Stochastic Gradient Descent (SGD)

SGD Algorithm

for $t = 0, 1, \dots$

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} l_{\mathcal{I}}(y(x_{\mathcal{I}}), y_{\theta_t, \mathcal{I}}^{NN})$$

where:

- θ_t : parameter vector at step t
- η : learning rate = fixed (!) step size
- $l_{\mathcal{I}}$: part of the loss function for stochastic index/set \mathcal{I} of samples

Stochastic Gradient Descent (SGD)

SGD Algorithm

for $t = 0, 1, \dots$

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} l_{\mathcal{I}}(y(x_{\mathcal{I}}), y_{\theta_t, \mathcal{I}}^{NN})$$

where:

- θ_t : parameter vector at step t
- η : learning rate = fixed (!) step size
- $l_{\mathcal{I}}$: part of the loss function for stochastic index/set \mathcal{I} of samples

Very successfull for deep neural nets!

Stochastic Gradient Descent (SGD)

SGD Algorithm

for $t = 0, 1, \dots$

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} l_{\mathcal{I}}(y(x_{\mathcal{I}}), y_{\theta_t, \mathcal{I}}^{NN})$$

where:

- θ_t : parameter vector at step t
- η : learning rate = fixed (!) step size
- $l_{\mathcal{I}}$: part of the loss function for stochastic index/set \mathcal{I} of samples

Very successfull for deep neural nets!

Gradients essential!

My Point of View



Munich, March 2018

A (Quite Long) History of Differentiation

- Nasir ad-Din at-Tusi (1201-1274 Iran and Irak)
examined positive solvability of
 $c = f(x) = x^2(a - x)$
by maximizing $f(x)$ using derivative $f'(x)$.



A (Quite Long) History of Differentiation

- Nasir ad-Din at-Tusi (1201-1274 Iran and Irak)
examined positive solvability of
 $c = f(x) = x^2(a - x)$
by maximizing $f(x)$ using derivative $f'(x)$.
- Power series for trigonometric functions
and inverses as well as their derivatives
known in Kerala in late 1300's.

A (Quite Long) History of Differentiation

- Nasir ad-Din at-Tusi (1201-1274 Iran and Irak)
examined positive solvability of
 $c = f(x) = x^2(a - x)$
by maximizing $f(x)$ using derivative $f'(x)$.
- Power series for trigonometric functions
and inverses as well as their derivatives
known in Kerala in late 1300's.
- Hence: Definitely not just the Newton versus
Leibniz controversy in 1666-1717



Computing Derivatives

Given:

Description of functional relation as

- formula $y = F(x)$ \Rightarrow explicit expression $y' = F'(x)$
- computer program \Rightarrow ?

Computing Derivatives

Given:

Description of functional relation as

- formula $y = F(x)$ \Rightarrow explicit expression $y' = F'(x)$
- computer program \Rightarrow ?

Task:

Computation of derivatives taking

- requirements on exactness
- computational effort

into account

Finite Differences

Idea: Taylor expansion, $f : \mathbb{R} \rightarrow \mathbb{R}$ smooth then

$$\begin{aligned} f(x + h) &= f(x) + hf'(x) + h^2f''(x)/2 + h^3f'''(x)/6 + \dots \\ \Rightarrow f(x + h) &\approx f(x) + hf'(x) \\ \Rightarrow Df(x) &= \frac{f(x + h) - f(x)}{h} \end{aligned}$$

Finite Differences

Idea: Taylor expansion, $f : \mathbb{R} \rightarrow \mathbb{R}$ smooth then

$$\begin{aligned} f(x + h) &= f(x) + hf'(x) + h^2f''(x)/2 + h^3f'''(x)/6 + \dots \\ \Rightarrow f(x + h) &\approx f(x) + hf'(x) \\ \Rightarrow Df(x) &= \frac{f(x + h) - f(x)}{h} \end{aligned}$$

- simple derivative calculation (only function evaluations!)
- inexact derivatives
- computation cost often too high

$$F : \mathbb{R}^n \rightarrow \mathbb{R} \Rightarrow \text{OPS}(\nabla F(x)) \sim (n + 1)\text{OPS}(F(x))$$

Analytic Differentiation

- symbolic derivatives, e.g.,

$$f(x) = \exp(\sin(x^2)) \Rightarrow$$

$$f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$$

Analytic Differentiation

- symbolic derivatives, e.g.,

$$f(x) = \exp(\sin(x^2)) \Rightarrow$$

$$f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$$

- analytic optimality conditions for optimal control problems, i.e.,

$$\min J(x, u) \quad \text{such that} \quad x' = f(x, u) \quad + \quad \text{IC} ?$$

Analytic Differentiation

- symbolic derivatives, e.g.,

$$f(x) = \exp(\sin(x^2)) \Rightarrow$$

$$f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$$

- analytic optimality conditions for optimal control problems, i.e.,

$$\min J(x, u) \text{ such that } x' = f(x, u) + \text{IC} ?$$

derivative computation based on

- sensitivity equation
- adjoint equation

$$\lambda' = -f_x(x, u)^\top \lambda + \text{TC}$$

$\Rightarrow J'(y(u), u)$ based on continuous adjoint

Analytic Differentiation

- symbolic derivatives, e.g.,

$$f(x) = \exp(\sin(x^2)) \Rightarrow$$

$$f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$$

- analytic optimality conditions for optimal control problems, i.e.,

$$\min J(x, u) \text{ such that } x' = f(x, u) + \text{IC} ?$$

derivative computation based on

- sensitivity equation
- adjoint equation

$$\lambda' = -f_x(x, u)^\top \lambda + \text{TC}$$

$\Rightarrow J'(y(u), u)$ based on continuous adjoint

- legacy code (large number of lines) \Rightarrow closed form not available

Analytic Differentiation

- symbolic derivatives, e.g.,

$$f(x) = \exp(\sin(x^2)) \Rightarrow$$

$$f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$$

- analytic optimality conditions for optimal control problems, i.e.,

$$\min J(x, u) \text{ such that } x' = f(x, u) + \text{IC} ?$$

derivative computation based on

- sensitivity equation
- adjoint equation

$$\lambda' = -f_x(x, u)^\top \lambda + \text{TC}$$

$\Rightarrow J'(y(u), u)$ based on continuous adjoint

- legacy code (large number of lines) \Rightarrow closed form not available
consistent derivative information?!

Algorithmic Differentiation (AD)

aka Automatic Differentiation

= Differentiation of computer programs implementing $F : \mathbb{R}^n \mapsto \mathbb{R}^m$

Algorithmic Differentiation (AD)

aka Automatic Differentiation

= Differentiation of computer programs implementing $F : \mathbb{R}^n \mapsto \mathbb{R}^m$

Main Products:

- Quantitative dependence information (local):
 - Weighted and directed partial derivatives
 - Error and condition number estimates ...
 - Lipschitz constants, interval enclosures ...

Algorithmic Differentiation (AD)

aka Automatic Differentiation

= Differentiation of computer programs implementing $F : \mathbb{R}^n \mapsto \mathbb{R}^m$

Main Products:

- Quantitative dependence information (local):
 - Weighted and directed partial derivatives
 - Error and condition number estimates ...
 - Lipschitz constants, interval enclosures ...
- Qualitative dependence information (regional):
 - Sparsity structures, degrees of polynomials
 - Ranks, eigenvalue multiplicities ...

Algorithmic Differentiation (AD)

aka Automatic Differentiation

= Differentiation of computer programs implementing $F : \mathbb{R}^n \mapsto \mathbb{R}^m$

Main Products:

- Quantitative dependence information (local):
 - Weighted and directed partial derivatives
 - Error and condition number estimates ...
 - Lipschitz constants, interval enclosures ...
- Qualitative dependence information (regional):
 - Sparsity structures, degrees of polynomials
 - Ranks, eigenvalue multiplicities ...

Assumption: F differentiable at least in a neighbourhood of current argument x

Historical Development of AD

J. Nolan	1953	→	J. M. Thamès et al.	1975	→
L. M. Beda et al.	1959	→	D. D. Warner	1975	→
A. Gibbons	1960	→			
J. W. Hanson et al.	1962	→	J. Joss	1980	→
R. E. Wengert	1964	→			
R. D. Wilkins	1964	→	L. B. Rall	1980	→
G. Wanner	1965	→			
R. Bellman et al.	1965	→	R. Kalaba et al.	1983	→
Y. F. Chang	1967	→			
D. Barton et al.	1971	→			
R. E. Pugh	1972	→	L. C. W. Dixon et al.	1986	→
			...		

Historical Development of AD

J. Nolan	1953	→	J. M. Thamès et al.	1975	→
L. M. Beda et al.	1959	→	D. D. Warner	1975	→
A. Gibbons	1960	→	W. Miller	1975	←
J. W. Hanson et al.	1962	→	J. Joss	1980	→
R. E. Wengert	1964	→	G. Kedem	1980	←
R. D. Wilkins	1964	→	B. Speelpenning	1980	←
G. Wanner	1965	→	L. B. Rall	1980	→
R. Bellman et al.	1965	→	W. Baur, V. Strassen	1983	←
Y. F. Chang	1967	→	R. Kalaba et al.	1983	→
S. Linnainmaa	1970	←	M. Iri et al.	1984	←
D. Barton et al.	1971	→	K. W. Kim et al.	1984	←
G. M. Ostrowski	1971	←	J. W. Sawyer	1984	←
R. E. Pugh	1972	→	L. C. W. Dixon et al.	1986	→
W. Stacey	1973	←	...		
P. Werbos	1974	←			

Historical Development of AD

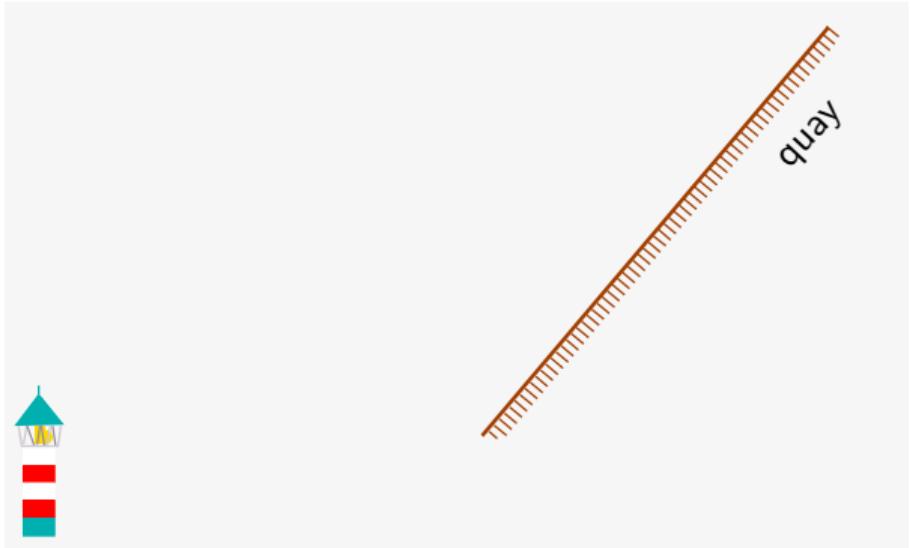
J. Nolan	1953	→	J. M. Thames et al.	1975	→
L. M. Beda et al.	1959	→	D. D. Warner	1975	→
A. Gibbons	1960	→	W. Miller	1975	←
J. W. Hanson et al.	1962	→	J. Joss	1980	→
R. E. Wengert	1964	→	G. Kedem	1980	←
R. D. Wilkins	1964	→	B. Speelpenning	1980	←
G. Wanner	1965	→	L. B. Rall	1980	→
R. Bellman et al.	1965	→	W. Baur, V. Strassen	1983	←
Y. F. Chang	1967	→	R. Kalaba et al.	1983	→
S. Linnainmaa	1970	←	M. Iri et al.	1984	←
D. Barton et al.	1971	→	K. W. Kim et al.	1984	←
G. M. Ostrowski	1971	←	J. W. Sawyer	1984	←
R. E. Pugh	1972	→	E. M. Orlow et al.	1985	↔
W. Stacey	1973	←	L. C. W. Dixon et al.	1986	→
P. Werbos	1974	←	...		

Historical Development of AD

J. Nolan	1953	→	J. M. Thamés et al.	1975	→
L. M. Beda et al.	1959	→	D. D. Warner	1975	→
A. Gibbons	1960	→	W. Miller	1975	←
J. W. Hanson et al.	1962	→	J. Joss	1980	→
R. E. Wengert	1964	→	G. Kedem	1980	←
R. D. Wilkins	1964	→	B. Speelpenning	1980	←
G. Wanner	1965	→	L. B. Rall	1980	→
R. Bellman et al.	1965	→	W. Baur, V. Strassen	1983	←
Y. F. Chang	1967	→	R. Kalaba et al.	1983	→
S. Linnainmaa	1970	←	M. Iri et al.	1984	←
D. Barton et al.	1971	→	K. W. Kim et al.	1984	←
G. M. Ostrowski	1971	←	J. W. Sawyer	1984	←
R. E. Pugh	1972	→	E. M. Orlow et al.	1985	↔
W. Stacey	1973	←	L. C. W. Dixon et al.	1986	→
P. Werbos	1974	←	...		

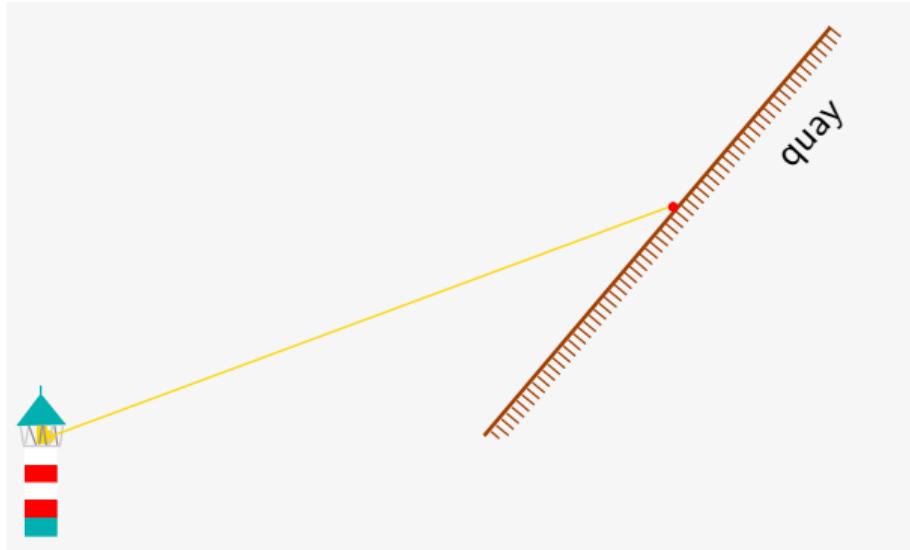
Rumelhart at al. (1986) made backpropagation famous for neural nets

The “Hello-World”-Example of AD



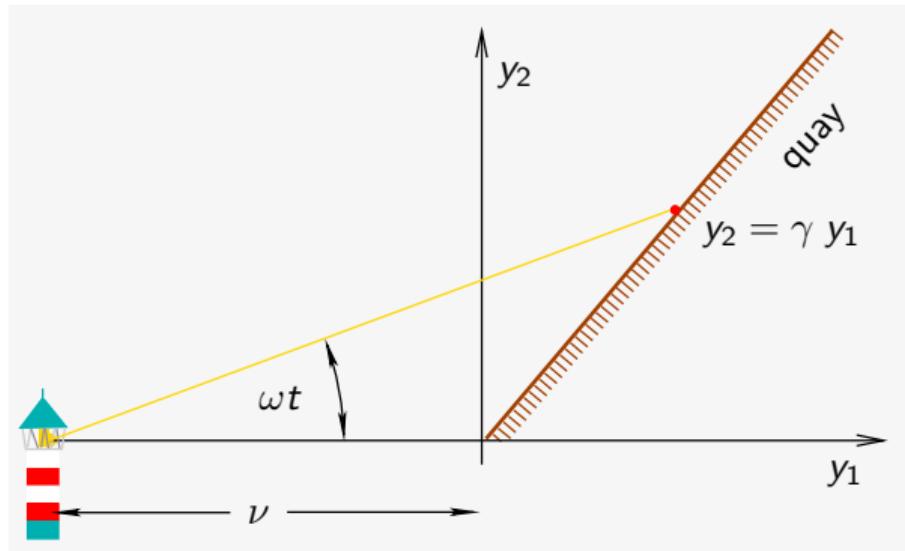
Lighthouse

The “Hello-World”-Example of AD



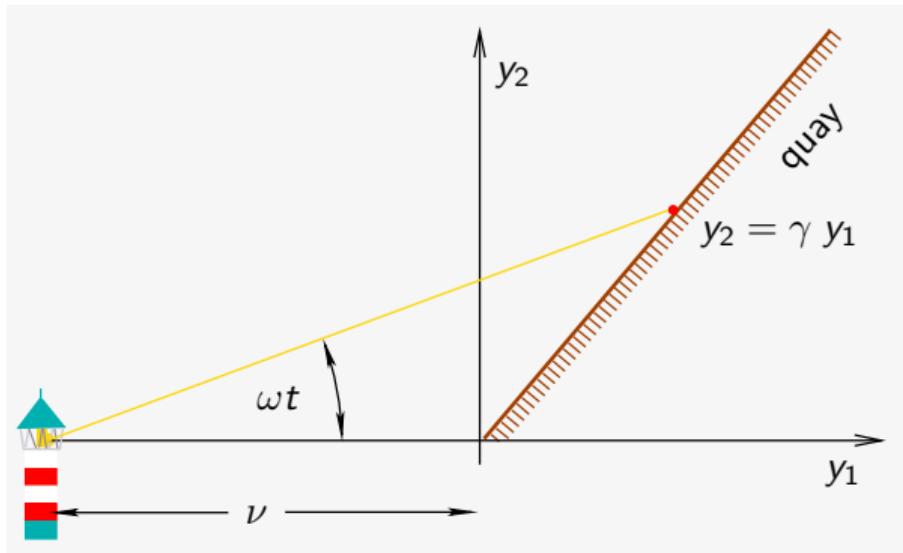
Lighthouse

The “Hello-World”-Example of AD



Lighthouse

The “Hello-World”-Example of AD



Lighthouse

$$y_1 = \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)}$$

and

$$y_2 = \frac{\gamma \nu \tan(\omega t)}{\gamma - \tan(\omega t)}$$

Evaluation Procedure (Lighthouse)

$$y_1 = \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)}$$

\implies

$$y_2 = \frac{\gamma \nu \tan(\omega t)}{\gamma - \tan(\omega t)}$$

$v_{-3} = x_1 = \nu$	
$v_{-2} = x_2 = \gamma$	
$v_{-1} = x_3 = \omega$	
$v_0 = x_4 = t$	
<hr/>	
$v_1 = v_{-1} * v_0 \equiv \varphi_1(v_{-1}, v_0)$	
$v_2 = \tan(v_1) \equiv \varphi_2(v_1)$	
$v_3 = v_{-2} - v_2 \equiv \varphi_3(v_{-2}, v_2)$	
$v_4 = v_{-3} * v_2 \equiv \varphi_4(v_{-3}, v_2)$	
$v_5 = v_4 / v_3 \equiv \varphi_5(v_4, v_3)$	
<hr/>	
$y_1 = v_5$	
$y_2 = v_6$	

Function Evaluation in ML

Typical function evaluation (deep neural net):

Propagation of one data point:

$$\begin{aligned} x = x^{(1)} &\rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} & \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)}) \\ &\rightarrow \tilde{x}^{(2)} = W^{(2)}x^{(2)} + b^{(2)} & \rightarrow x^{(3)} = \rho(\tilde{x}^{(2)}) \rightarrow \dots \\ &\rightarrow y = W^{(k)}x^{(k)} + b^{(k)} \end{aligned}$$

Function Evaluation in ML

Typical function evaluation (deep neural net):

Propagation of one data point:

$$\begin{aligned} x = x^{(1)} &\rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} & \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)}) \\ &\rightarrow \tilde{x}^{(2)} = W^{(2)}x^{(2)} + b^{(2)} & \rightarrow x^{(3)} = \rho(\tilde{x}^{(2)}) \rightarrow \dots \\ &\rightarrow y = W^{(k)}x^{(k)} + b^{(k)} \end{aligned}$$

Empirical risk, loss function, ...

$$f(x_{1 \leq i \leq M}) = \frac{1}{M} \sum_{i=1}^M l(y_i(x_i), y_{\theta,i}^{NN})$$

Function Evaluation in ML

Typical function evaluation (deep neural net):

Propagation of one data point:

$$\begin{aligned} x = x^{(1)} &\rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} & \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)}) \\ &\rightarrow \tilde{x}^{(2)} = W^{(2)}x^{(2)} + b^{(2)} & \rightarrow x^{(3)} = \rho(\tilde{x}^{(2)}) \rightarrow \dots \\ &\rightarrow y = W^{(k)}x^{(k)} + b^{(k)} \end{aligned}$$

Empirical risk, loss function, ...

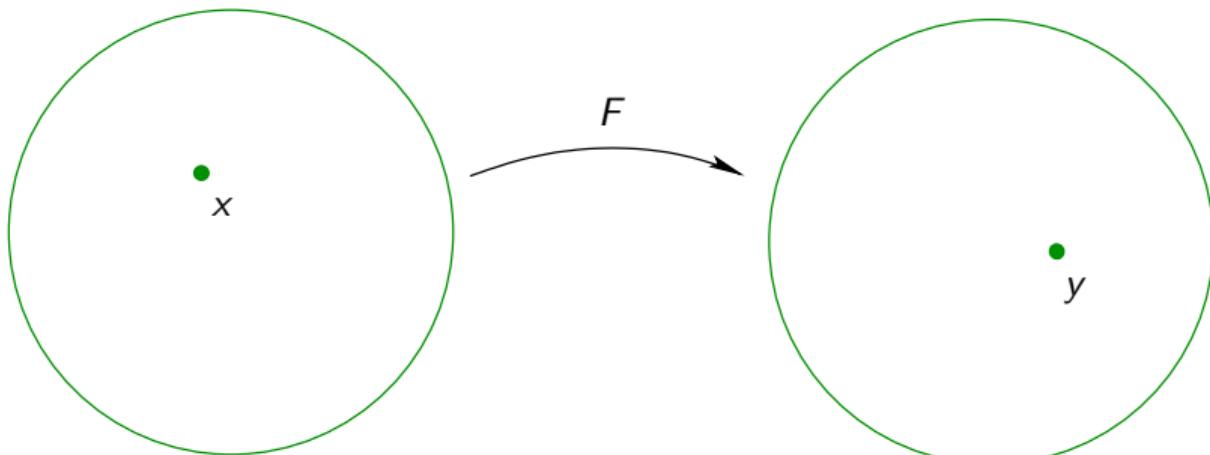
$$f(x_{1 \leq i \leq M}) = \frac{1}{M} \sum_{i=1}^M l(y_i(x_i), y_{\theta,i}^{NN})$$

Stochastic gradient descent requires

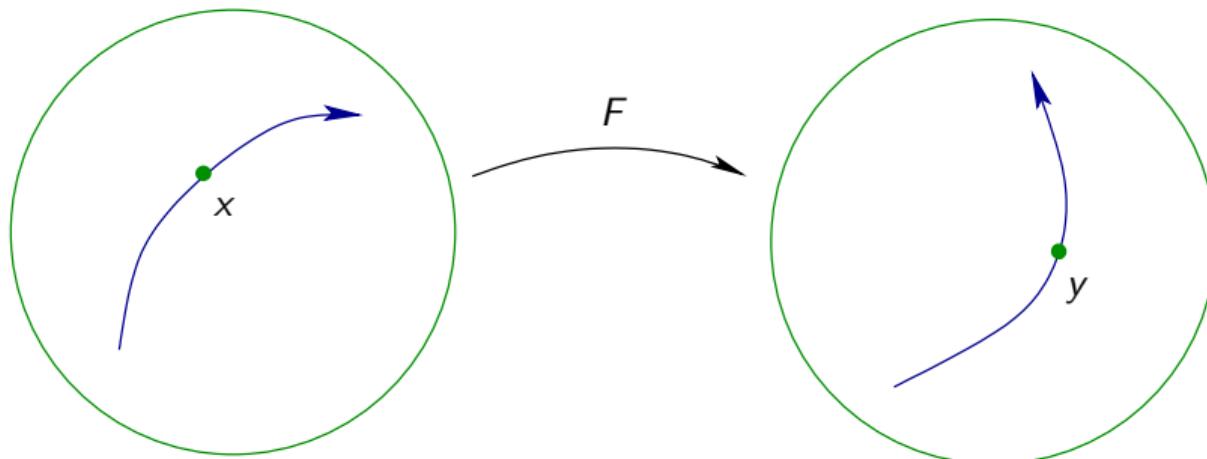
$$\nabla_{W^1, b^1, \dots, W^k, b^k} l(y_i(x_i), y_{\theta,i}^{NN})$$

for $i \in \{1, \dots, M\}$

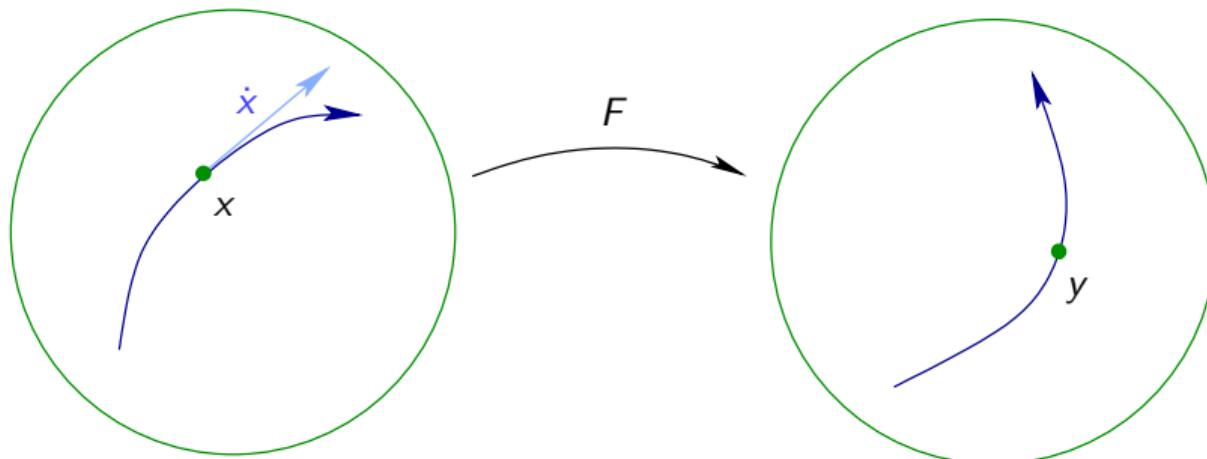
Forward mode AD = Tangents/Sensitivities



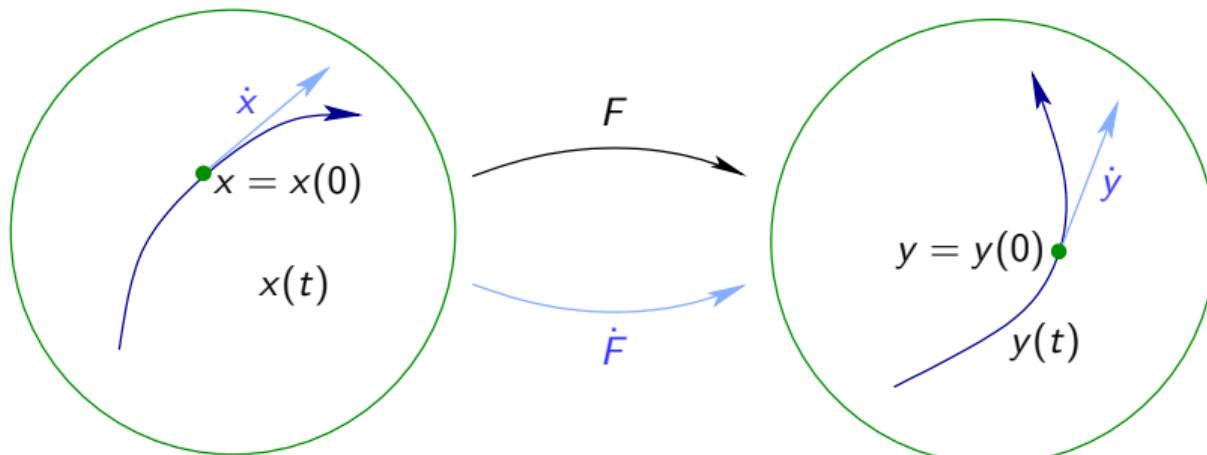
Forward mode AD = Tangents/Sensitivities



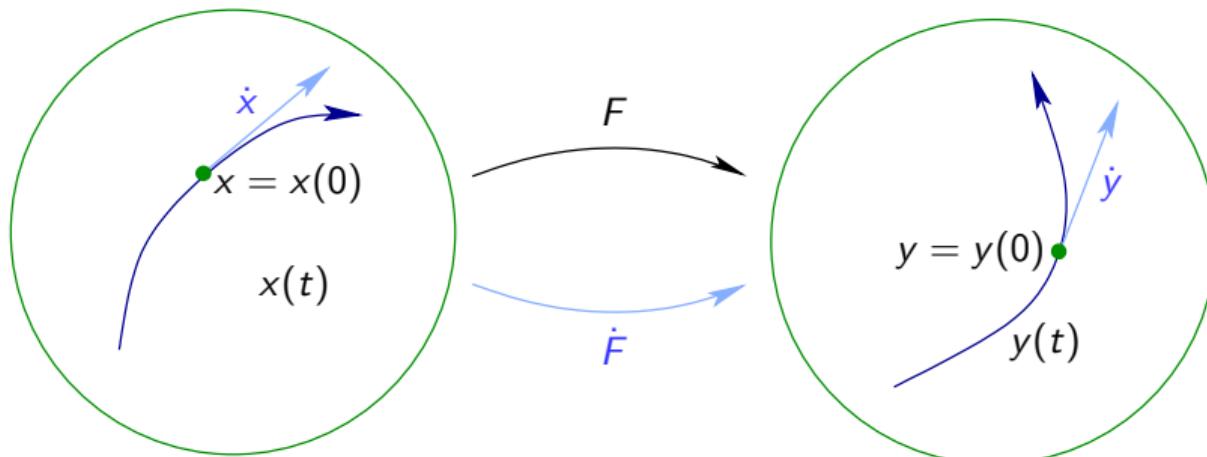
Forward mode AD = Tangents/Sensitivities



Forward mode AD = Tangents/Sensitivities



Forward mode AD = Tangents/Sensitivities



$$\dot{y}(t) = \frac{\partial}{\partial t} F(x(t)) = F'(x(t)) \dot{x}(t) \equiv \dot{F}(x, \dot{x})$$

Forward Mode (Lighthouse)

$$v_{-3} = x_1 = \nu$$

$$v_{-2} = x_2 = \gamma$$

$$v_{-1} = x_3 = \omega$$

$$v_0 = x_4 = t$$

$$v_1 = v_{-1} * v_0$$

$$v_2 = \tan(v_1)$$

$$v_3 = v_{-2} - v_2$$

$$v_4 = v_{-3} * v_2$$

$$v_5 = v_4 / v_3$$

$$v_6 = v_5 * v_{-2}$$

$$y_1 = v_5$$

$$y_2 = v_6$$

Forward Mode (Lighthouse)

$$\begin{array}{rcl}
 v_{-3} & = & x_1 = \nu \\
 v_{-2} & = & x_2 = \gamma \\
 v_{-1} & = & x_3 = \omega \\
 v_0 & = & x_4 = t
 \end{array}
 \quad
 \begin{array}{rcl}
 \dot{v}_{-3} & = & \dot{x}_1 \\
 \dot{v}_{-2} & = & \dot{x}_2 \\
 \dot{v}_{-1} & = & \dot{x}_3 \\
 \dot{v}_0 & = & \dot{x}_4
 \end{array}$$

$$v_1 = v_{-1} * v_0$$

$$v_2 = \tan(v_1)$$

$$v_3 = v_{-2} - v_2$$

$$v_4 = v_{-3} * v_2$$

$$v_5 = v_4 / v_3$$

$$v_6 = v_5 * v_{-2}$$

$$y_1 = v_5$$

$$y_2 = v_6$$

Forward Mode (Lighthouse)

v_{-3}	$=$	$x_1 = \nu$	\dot{v}_{-3}	$=$	\dot{x}_1
v_{-2}	$=$	$x_2 = \gamma$	\dot{v}_{-2}	$=$	\dot{x}_2
v_{-1}	$=$	$x_3 = \omega$	\dot{v}_{-1}	$=$	\dot{x}_3
v_0	$=$	$x_4 = t$	\dot{v}_0	$=$	\dot{x}_4
<hr/>					
v_1	$=$	$v_{-1} * v_0$	\dot{v}_1	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$			
v_3	$=$	$v_{-2} - v_2$			
v_4	$=$	$v_{-3} * v_2$			
v_5	$=$	v_4 / v_3			
v_6	$=$	$v_5 * v_{-2}$			
<hr/>					
y_1	$=$	v_5			
y_2	$=$	v_6			

Forward Mode (Lighthouse)

v_{-3}	$=$	$x_1 = \nu$	\dot{v}_{-3}	$=$	\dot{x}_1
v_{-2}	$=$	$x_2 = \gamma$	\dot{v}_{-2}	$=$	\dot{x}_2
v_{-1}	$=$	$x_3 = \omega$	\dot{v}_{-1}	$=$	\dot{x}_3
v_0	$=$	$x_4 = t$	\dot{v}_0	$=$	\dot{x}_4
<hr/>					
v_1	$=$	$v_{-1} * v_0$	\dot{v}_1	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$	\dot{v}_2	$=$	$\dot{v}_1 / \cos(v_1)^2$
v_3	$=$	$v_{-2} - v_2$			
v_4	$=$	$v_{-3} * v_2$			
v_5	$=$	v_4 / v_3			
v_6	$=$	$v_5 * v_{-2}$			
<hr/>					
y_1	$=$	v_5			
y_2	$=$	v_6			

Forward Mode (Lighthouse)

v_{-3}	$=$	$x_1 = \nu$	\dot{v}_{-3}	$=$	\dot{x}_1
v_{-2}	$=$	$x_2 = \gamma$	\dot{v}_{-2}	$=$	\dot{x}_2
v_{-1}	$=$	$x_3 = \omega$	\dot{v}_{-1}	$=$	\dot{x}_3
v_0	$=$	$x_4 = t$	\dot{v}_0	$=$	\dot{x}_4
<hr/>					
v_1	$=$	$v_{-1} * v_0$	\dot{v}_1	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$	\dot{v}_2	$=$	$\dot{v}_1 / \cos(v_1)^2$
v_3	$=$	$v_{-2} - v_2$	\dot{v}_3	$=$	$\dot{v}_{-2} - \dot{v}_2$
v_4	$=$	$v_{-3} * v_2$			
v_5	$=$	v_4 / v_3			
v_6	$=$	$v_5 * v_{-2}$			
<hr/>					
y_1	$=$	v_5			
y_2	$=$	v_6			

Forward Mode (Lighthouse)

v_{-3}	$=$	$x_1 = \nu$	\dot{v}_{-3}	$=$	\dot{x}_1
v_{-2}	$=$	$x_2 = \gamma$	\dot{v}_{-2}	$=$	\dot{x}_2
v_{-1}	$=$	$x_3 = \omega$	\dot{v}_{-1}	$=$	\dot{x}_3
v_0	$=$	$x_4 = t$	\dot{v}_0	$=$	\dot{x}_4
<hr/>					
v_1	$=$	$v_{-1} * v_0$	\dot{v}_1	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$	\dot{v}_2	$=$	$\dot{v}_1 / \cos(v_1)^2$
v_3	$=$	$v_{-2} - v_2$	\dot{v}_3	$=$	$\dot{v}_{-2} - \dot{v}_2$
v_4	$=$	$v_{-3} * v_2$	\dot{v}_4	$=$	$\dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2$
v_5	$=$	v_4 / v_3			
v_6	$=$	$v_5 * v_{-2}$			
<hr/>					
y_1	$=$	v_5			
y_2	$=$	v_6			

Forward Mode (Lighthouse)

v_{-3}	$=$	$x_1 = \nu$	\dot{v}_{-3}	$=$	\dot{x}_1
v_{-2}	$=$	$x_2 = \gamma$	\dot{v}_{-2}	$=$	\dot{x}_2
v_{-1}	$=$	$x_3 = \omega$	\dot{v}_{-1}	$=$	\dot{x}_3
v_0	$=$	$x_4 = t$	\dot{v}_0	$=$	\dot{x}_4
<hr/>					
v_1	$=$	$v_{-1} * v_0$	\dot{v}_1	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$	\dot{v}_2	$=$	$\dot{v}_1 / \cos(v_1)^2$
v_3	$=$	$v_{-2} - v_2$	\dot{v}_3	$=$	$\dot{v}_{-2} - \dot{v}_2$
v_4	$=$	$v_{-3} * v_2$	\dot{v}_4	$=$	$\dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2$
v_5	$=$	v_4 / v_3	\dot{v}_5	$=$	$(\dot{v}_4 - \dot{v}_3 * v_5) * (1/v_3)$
v_6	$=$	$v_5 * v_{-2}$			
<hr/>					
y_1	$=$	v_5			
y_2	$=$	v_6			

Forward Mode (Lighthouse)

v_{-3}	$=$	$x_1 = \nu$	\dot{v}_{-3}	$=$	\dot{x}_1
v_{-2}	$=$	$x_2 = \gamma$	\dot{v}_{-2}	$=$	\dot{x}_2
v_{-1}	$=$	$x_3 = \omega$	\dot{v}_{-1}	$=$	\dot{x}_3
v_0	$=$	$x_4 = t$	\dot{v}_0	$=$	\dot{x}_4
<hr/>					
v_1	$=$	$v_{-1} * v_0$	\dot{v}_1	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$	\dot{v}_2	$=$	$\dot{v}_1 / \cos(v_1)^2$
v_3	$=$	$v_{-2} - v_2$	\dot{v}_3	$=$	$\dot{v}_{-2} - \dot{v}_2$
v_4	$=$	$v_{-3} * v_2$	\dot{v}_4	$=$	$\dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2$
v_5	$=$	v_4 / v_3	\dot{v}_5	$=$	$(\dot{v}_4 - \dot{v}_3 * v_5) * (1/v_3)$
v_6	$=$	$v_5 * v_{-2}$	\dot{v}_6	$=$	$\dot{v}_5 * v_{-2} + v_5 * \dot{v}_{-2}$
<hr/>					
y_1	$=$	v_5			
y_2	$=$	v_6			

Forward Mode (Lighthouse)

v_{-3}	$=$	$x_1 = \nu$	\dot{v}_{-3}	$=$	\dot{x}_1
v_{-2}	$=$	$x_2 = \gamma$	\dot{v}_{-2}	$=$	\dot{x}_2
v_{-1}	$=$	$x_3 = \omega$	\dot{v}_{-1}	$=$	\dot{x}_3
v_0	$=$	$x_4 = t$	\dot{v}_0	$=$	\dot{x}_4
<hr/>					
v_1	$=$	$v_{-1} * v_0$	\dot{v}_1	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$	\dot{v}_2	$=$	$\dot{v}_1 / \cos(v_1)^2$
v_3	$=$	$v_{-2} - v_2$	\dot{v}_3	$=$	$\dot{v}_{-2} - \dot{v}_2$
v_4	$=$	$v_{-3} * v_2$	\dot{v}_4	$=$	$\dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2$
v_5	$=$	v_4 / v_3	\dot{v}_5	$=$	$(\dot{v}_4 - \dot{v}_3 * v_5) * (1/v_3)$
v_6	$=$	$v_5 * v_{-2}$	\dot{v}_6	$=$	$\dot{v}_5 * v_{-2} + v_5 * \dot{v}_{-2}$
<hr/>					
y_1	$=$	v_5	\dot{y}_1	$=$	\dot{v}_5
y_2	$=$	v_6	\dot{y}_2	$=$	\dot{v}_6

Complexity (Forward Mode)

tang	c	\pm	*	ψ
MOVES	$1 + 1$	$3 + 3$	$3 + 3$	$2 + 2$
ADDS	0	$1 + 1$	$0 + 1$	$0 + 0$
MULTS	0	0	$1 + 2$	$0 + 1$
NLOPS	0	0	0	$1 + 1$

Complexity (Forward Mode)

tang	c	\pm	*	ψ
MOVES	$1 + 1$	$3 + 3$	$3 + 3$	$2 + 2$
ADDS	0	$1 + 1$	$0 + 1$	$0 + 0$
MULTS	0	0	$1 + 2$	$0 + 1$
NLOPS	0	0	0	$1 + 1$



$$\text{OPS}(\mathcal{F}'(x)\dot{x}) \leq c \text{ OPS}(F(x))$$

with $c \in [2, 5/2]$ platform dependent

Forward Mode AD for ML

Typical function evaluation (deep neural net):

$$x = x^{(1)} \rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)})$$

Attention: Optimization variables W and $b \Rightarrow \dot{W}$ and \dot{b} !

Forward Mode AD for ML

Typical function evaluation (deep neural net):

$$x = x^{(1)} \rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)})$$

Attention: Optimization variables W and $b \Rightarrow \dot{W}$ and \dot{b} !

$$\begin{aligned} x = x^{(1)} \rightarrow \tilde{x}^{(1)} &= W^{(1)}x^{(1)} + b^{(1)} & \rightarrow x^{(2)} &= \rho(\tilde{x}^{(1)}) \\ \dot{\tilde{x}}^{(1)} &= \dot{W}^{(1)}x^{(1)} + \dot{b}^{(1)} & \rightarrow \dot{x}^{(2)} &= \rho'(\tilde{x}^{(1)})\dot{\tilde{x}}^{(1)} \end{aligned}$$

Forward Mode AD for ML

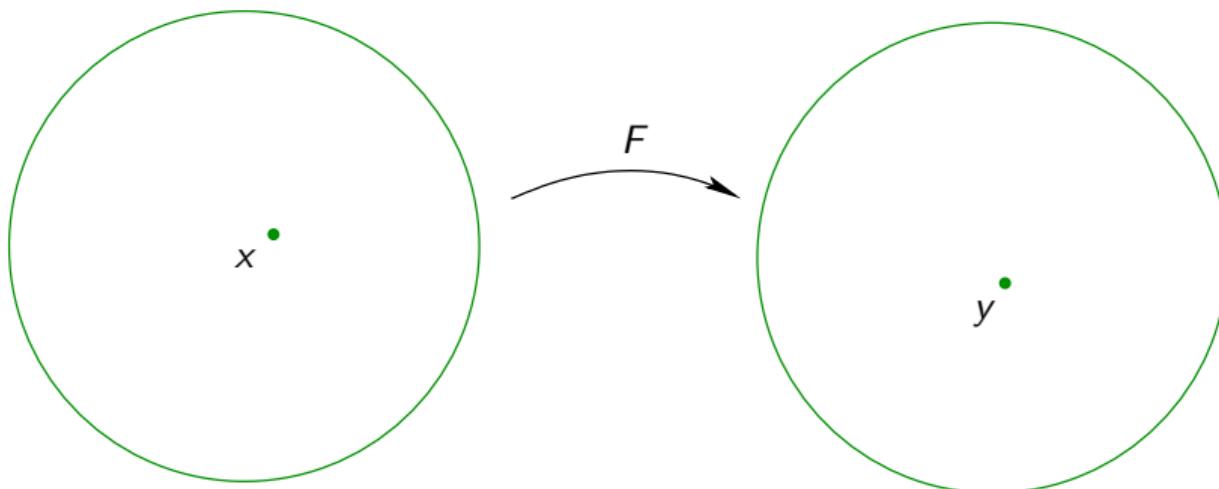
Typical function evaluation (deep neural net):

$$x = x^{(1)} \rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)})$$

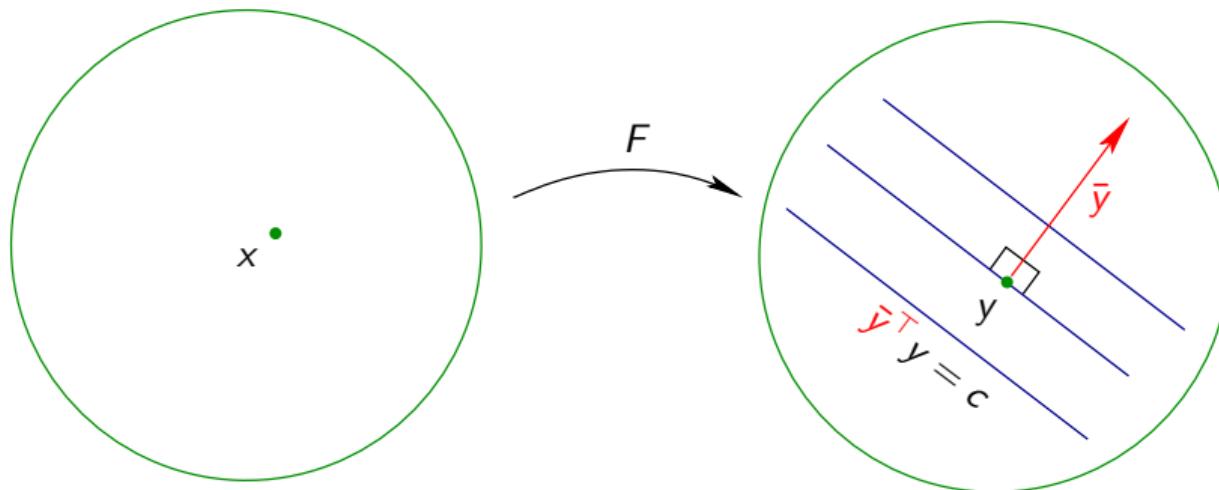
Attention: Optimization variables W and $b \Rightarrow \dot{W}$ and \dot{b} !

$$\begin{aligned} x = x^{(1)} &\rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} & \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)}) \\ \dot{\tilde{x}}^{(1)} &= \dot{W}^{(1)}x^{(1)} + \dot{b}^{(1)} & \rightarrow \dot{x}^{(2)} = \rho'(\tilde{x}^{(1)})\dot{\tilde{x}}^{(1)} \\ \rightarrow \tilde{x}^{(2)} &= W^{(2)}x^{(2)} + b^{(2)} & \rightarrow x^{(3)} = \rho(\tilde{x}^{(2)}) \\ \dot{\tilde{x}}^{(2)} &= \dot{W}^{(2)}x^{(2)} + W^{(2)}\dot{x}^{(2)} + \dot{b}^{(2)} & \rightarrow \dot{x}^{(3)} = \rho'(\tilde{x}^{(3)})\dot{\tilde{x}}^{(3)} \\ \rightarrow \cdots & & \\ \rightarrow y &= W^{(k)}x^{(k)} + b^{(k)} & \\ \rightarrow \dot{y} &= \dot{W}^{(k)}x^{(k)} + W^{(k)}\dot{x}^{(k)} + \dot{b}^{(k)} & \end{aligned}$$

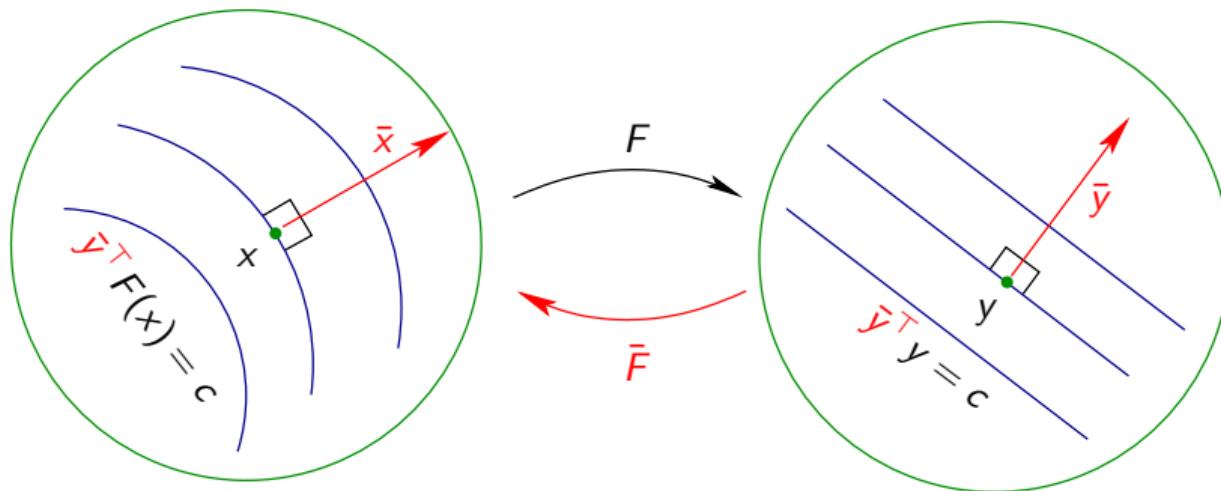
Reverse Mode AD = Discrete Adjoints



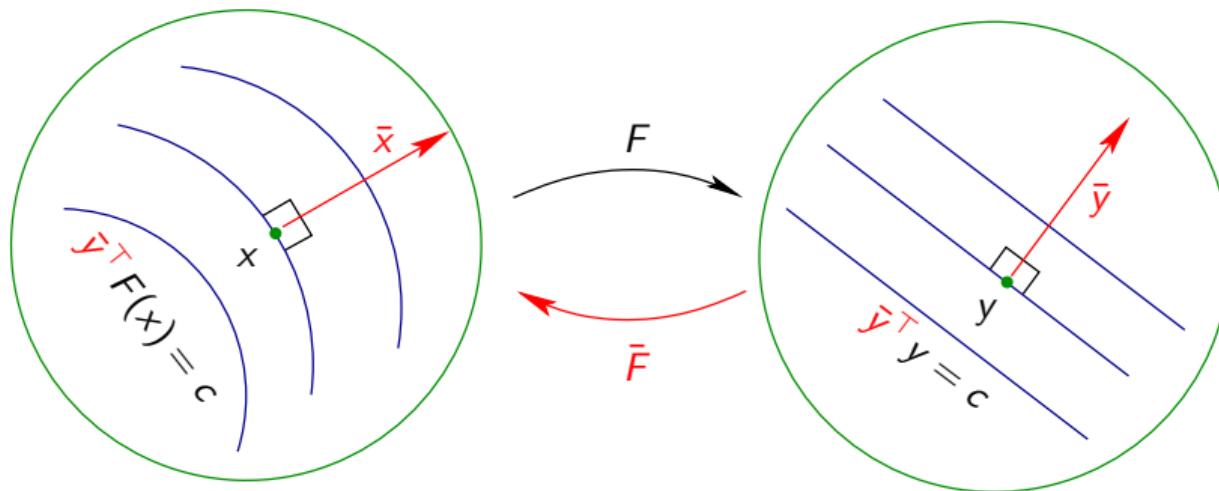
Reverse Mode AD = Discrete Adjoints



Reverse Mode AD = Discrete Adjoints



Reverse Mode AD = Discrete Adjoints



$$\bar{x} \equiv \bar{y}^\top F'(x) = \nabla_x \langle \bar{y}^\top F(x) \rangle \equiv \bar{F}(x, \bar{y})$$

Reverse Mode (Lighthouse)

```

 $v_{-3} = x_1; \quad v_{-2} = x_2; \quad v_{-1} = x_3; \quad v_0 = x_4;$ 
 $v_1 = v_{-1} * v_0;$ 
 $v_2 = \tan(v_1);$ 
 $v_3 = v_{-2} - v_2;$ 
 $v_4 = v_{-3} * v_2;$ 
 $v_5 = v_4 / v_3;$ 
 $v_6 = v_5 * v_{-2};$ 


---


 $y_1 = v_5; \quad y_2 = v_6;$ 
 $\bar{v}_5 = \bar{y}_1; \quad \bar{v}_6 = \bar{y}_2;$ 
 $\bar{v}_5 += \bar{v}_6 * v_{-2}; \quad \bar{v}_{-2} += \bar{v}_6 * v_5;$ 
 $\bar{v}_4 += \bar{v}_5 / v_3; \quad \bar{v}_3 -= \bar{v}_5 * v_5 / v_3;$ 
 $\bar{v}_{-3} += \bar{v}_4 * v_2; \quad \bar{v}_2 += \bar{v}_4 * v_{-3};$ 
 $\bar{v}_{-2} += \bar{v}_3; \quad \bar{v}_2 -= \bar{v}_3;$ 
 $\bar{v}_1 += \bar{v}_2 / \cos^2(v_1);$ 
 $\bar{v}_{-1} += \bar{v}_1 * v_0; \quad \bar{v}_0 += \bar{v}_1 * v_{-1};$ 
 $\bar{x}_4 = \bar{v}_0; \quad \bar{x}_3 = \bar{v}_{-1}; \quad \bar{x}_2 = \bar{v}_{-2}; \quad \bar{x}_1 = \bar{v}_{-3};$ 

```

Complexity (Reverse Mode)

grad	c	\pm	*	ψ
MOVES	$1 + 1$	$3 + 6$	$3 + 8$	$2 + 5$
ADDS	0	$1 + 2$	$0 + 2$	$0 + 1$
MULTS	0	0	$1 + 2$	$0 + 1$
NLOPS	0	0	0	$1 + 1$



$$\text{OPS}(\bar{y}^\top F'(x)) \leq c \text{ OPS}(F(x)), \text{ MEM}(\bar{y}^\top F'(x)) \sim \text{OPS}(F(x))$$

with $c \in [3, 4]$ platform dependent

Complexity (Reverse Mode)

grad	c	\pm	*	ψ
MOVES	$1 + 1$	$3 + 6$	$3 + 8$	$2 + 5$
ADDS	0	$1 + 2$	$0 + 2$	$0 + 1$
MULTS	0	0	$1 + 2$	$0 + 1$
NLOPS	0	0	0	$1 + 1$



$$\text{OPS}(\bar{y}^\top F'(x)) \leq c \text{ OPS}(F(x)), \text{ MEM}(\bar{y}^\top F'(x)) \sim \text{OPS}(F(x))$$

with $c \in [3, 4]$ platform dependent

Remarks:

- Cost for gradient calculation independent of n
- Memory requirement may cause problem! \Rightarrow Checkpointing

Reverse Mode AD for ML

= a.k.a. backpropagation

Typical function evaluation (deep neural net):

$$\begin{aligned}x &= x^{(1)} \rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)}) \\&\rightarrow \tilde{x}^{(2)} = W^{(2)}x^{(2)} + b^{(2)} \rightarrow x^{(3)} = \rho(\tilde{x}^{(2)}) \\&\rightarrow \dots \\&\rightarrow y = W^{(k)}x^{(k)} + b^{(k)}\end{aligned}$$

Reverse Mode AD for ML

= a.k.a. backpropagation

Typical function evaluation (deep neural net):

$$\begin{aligned}
 x = x^{(1)} &\rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} & \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)}) \\
 &\rightarrow \tilde{x}^{(2)} = W^{(2)}x^{(2)} + b^{(2)} & \rightarrow x^{(3)} = \rho(\tilde{x}^{(2)}) \\
 &\rightarrow \dots \\
 &\rightarrow y = W^{(k)}x^{(k)} + b^{(k)}
 \end{aligned}$$

With $\bar{y} = 1$ one obtains

$$\bar{W}^{(k)} = [x^{(k)}], \quad \bar{x}^{(k)} = W^{(k)}, \quad \bar{b}^{(k)} = \mathbb{1}$$

Reverse Mode AD for ML

= a.k.a. backpropagation

Typical function evaluation (deep neural net):

$$\begin{aligned}
 x = x^{(1)} &\rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} & \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)}) \\
 &\rightarrow \tilde{x}^{(2)} = W^{(2)}x^{(2)} + b^{(2)} & \rightarrow x^{(3)} = \rho(\tilde{x}^{(2)}) \\
 &\rightarrow \dots \\
 &\rightarrow y = W^{(k)}x^{(k)} + b^{(k)}
 \end{aligned}$$

With $\bar{y} = 1$ one obtains

$$\bar{W}^{(k)} = [x^{(k)}], \quad \bar{x}^{(k)} = W^{(k)}, \quad \bar{b}^{(k)} = \mathbb{1}$$

\dots

$$\begin{aligned}
 \tilde{\bar{x}}^{(2)} &= \rho'(x^{(2)}) * \bar{x}^{(3)}, & \bar{W}^{(2)} &= x^{(2)} * \tilde{\bar{x}}^{(2)}, \bar{x}^{(2)} &= W^{(2)} * \tilde{\bar{x}}^{(2)}, & \bar{b}^{(2)} &= \tilde{\bar{x}}^{(2)} \\
 \tilde{\bar{x}}^{(1)} &= \rho'(x^{(1)}) * \bar{x}^{(2)}, & \bar{W}^{(1)} &= x^{(1)} * \tilde{\bar{x}}^{(1)}, \bar{x}^{(1)} &= W^{(1)} * \tilde{\bar{x}}^{(1)}, & \bar{b}^{(1)} &= \tilde{\bar{x}}^{(1)}
 \end{aligned}$$

Reverse Mode AD for ML

= a.k.a. backpropagation

Typical function evaluation (deep neural net):

$$\begin{aligned}
 x = x^{(1)} &\rightarrow \tilde{x}^{(1)} = W^{(1)}x^{(1)} + b^{(1)} & \rightarrow x^{(2)} = \rho(\tilde{x}^{(1)}) \\
 &\rightarrow \tilde{x}^{(2)} = W^{(2)}x^{(2)} + b^{(2)} & \rightarrow x^{(3)} = \rho(\tilde{x}^{(2)}) \\
 &\rightarrow \dots \\
 &\rightarrow y = W^{(k)}x^{(k)} + b^{(k)}
 \end{aligned}$$

With $\bar{y} = 1$ one obtains

$$\bar{W}^{(k)} = [x^{(k)}], \quad \bar{x}^{(k)} = W^{(k)}, \quad \bar{b}^{(k)} = \mathbb{1}$$

\dots

$$\begin{aligned}
 \tilde{\bar{x}}^{(2)} &= \rho'(x^{(2)}) * \bar{x}^{(3)}, & \bar{W}^{(2)} &= x^{(2)} * \tilde{\bar{x}}^{(2)}, \bar{x}^{(2)} &= W^{(2)} * \tilde{\bar{x}}^{(2)}, & \bar{b}^{(2)} &= \tilde{\bar{x}}^{(2)} \\
 \tilde{\bar{x}}^{(1)} &= \rho'(x^{(1)}) * \bar{x}^{(2)}, & \bar{W}^{(1)} &= x^{(1)} * \tilde{\bar{x}}^{(1)}, \bar{x}^{(1)} &= W^{(1)} * \tilde{\bar{x}}^{(1)}, & \bar{b}^{(1)} &= \tilde{\bar{x}}^{(1)}
 \end{aligned}$$

very simple to implement!

Numerical Stability

Usually:

No problems with accuracy of computed derivatives

Numerical Stability

Usually:

No problems with accuracy of computed derivatives but:

Example by John Reid shows:

Blow up of derivatives for cross-country mode of AD possible!

Numerical Stability

Usually:

No problems with accuracy of computed derivatives but:

Example by John Reid shows:

Blow up of derivatives for cross-country mode of AD possible!

Therefore:

Formal error analysis for forward and reverse mode of AD yielding:

Condition of derivative calculation = condition of function evaluation

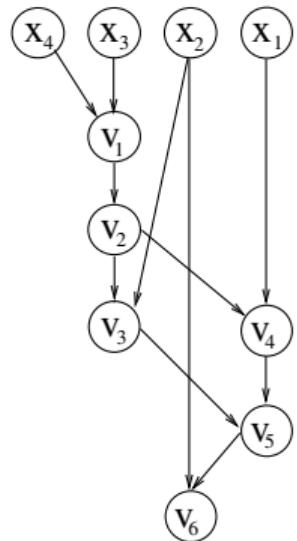
(A. Griewank, K. Kulshreshtha, A. Walther, Computing, 2012)

Berlin Mathematics Research Center

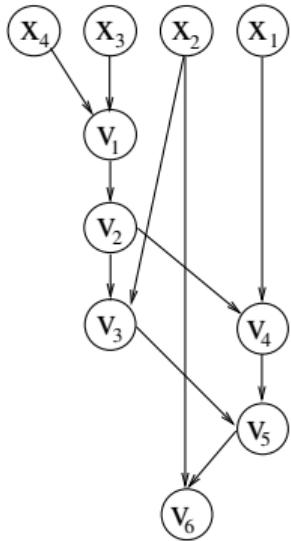
Automatic Differentiation by OverLoading in C++

- ADOL-C version 2.7, available at COIN-OR since 2009
open source (GPL or ECL)
- based on operator overloading, trace as internal representation
- general-purpose AD tool with focus on functionalities
- interfaces to ColPack (Purdue University) and Ipopt (COIN-OR)
- current developments
 - exploitation of fixed-point structure for second-order derivatives
 - generalized derivatives for nonsmooth functions

Model Generation with ADOL-C



Model Generation with ADOL-C



```

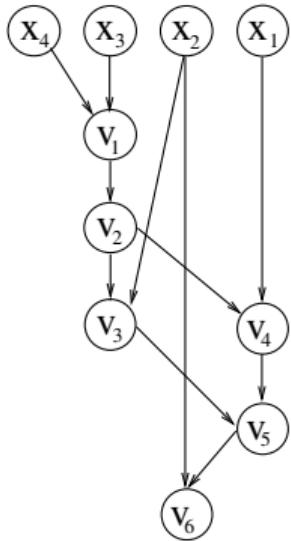
double x1, x2, x3, x4;
double v1, v2, v3, v4, v5, v6;
double y1, y2;
  
```

$$\begin{aligned}x1 &= 3.7; \quad x2 = 0.7; \\x3 &= 0.5; \quad x4 = 0.5;\end{aligned}$$

$$\begin{aligned}v1 &= x3 * x4; \\v2 &= \tan(v1); \\v3 &= x2 - v2; \\v4 &= x1 * v2; \\v5 &= v4 / v3; \\v6 &= v5 * x2\end{aligned}$$

$$y1 = v5; \quad y2 = v6;$$

Model Generation with ADOL-C



```

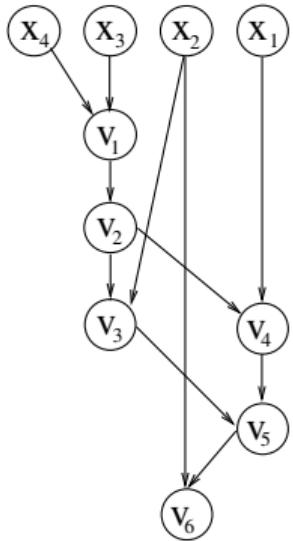
adouble x1, x2, x3, x4;
adouble v1, v2, v3, v4, v5, v6;
double y1, y2;

trace_on(tag);
x1 <= 3.7; x2 <= 0.7;
x3 <= 0.5; x4 <= 0.5;

v1 = x3*x4;
v2 = tan(v1);
v3 = x2-v2;
v4 = x1*v2;
v5 = v4/v3;
v6 = v5*x2

v5 >= y1; v6 >= y2;
trace_off();
  
```

Model Generation with ADOL-C



```

adouble x1, x2, x3, x4;
adouble v1, v2, v3, v4, v5, v6;
double y1, y2;

trace_on(tag);
x1 <= 3.7; x2 <= 0.7;
x3 <= 0.5; x4 <= 0.5;
v1 = x3*x4;
v2 = tan(v1);
v3 = x2-v2;
v4 = x1*v2;
v5 = v4/v3;
v6 = v5*x2
v5 >= y1; v6 >= y2;
trace_off();
  
```

ADOL-C trace

trace_on, tag
<=, x1, 3.7
<=, x2, 0.7
<=, x3, 0.5
<=, x4, 0.5
*, x3, x4, v1
tan, v1, v2
-, x2, v2, v3
*, x1, v2, v4
/, v4, v3, v5
*, v5, x2, v6
>=, v5, y1
>=, v6, y2
trace_off

Easy-to-use Routines

int gradient(**tag**,**n**,**x[n]**,**g[n]**):

tag	= tape number
n	= # indeps
x[n]	= values of indeps
g[n]	= $\nabla f(x)$

Easy-to-use Routines

`int gradient(tag,n,x[n],g[n]):`

<code>tag</code>	= tape number
<code>n</code>	= # indeps
<code>x[n]</code>	= values of indeps
<code>g[n]</code>	= $\nabla f(x)$

`int jacobian(tag,m,n,x[n],J[m][n]):`

<code>tag</code>	= tape number
<code>m</code>	= # deps
<code>n</code>	= # indeps
<code>x[n]</code>	= values of indeps
<code>J[m][n]</code>	= $F'(x)$

Easy-to-use Routines

`int gradient(tag,n,x[n],g[n]):`

<code>tag</code>	= tape number
<code>n</code>	= # indeps
<code>x[n]</code>	= values of indeps
<code>g[n]</code>	= $\nabla f(x)$

`int jacobian(tag,m,n,x[n],J[m][n]):`

<code>tag</code>	= tape number
<code>m</code>	= # deps
<code>n</code>	= # indeps
<code>x[n]</code>	= values of indeps
<code>J[m][n]</code>	= $F'(x)$

`int hessian(tag,n,x[n],H[n][n])`

<code>tag</code>	= tape number
<code>n</code>	= # indeps
<code>x[n]</code>	= values of indeps
<code>H[n][n]</code>	= $\nabla^2 f(x)$

Lighthouse Example: Real Code

```
void f(double *x, double*y)
{
    // intermediates
    double v[2];
    v[0] = tan(x[2]*x[3]);
    v[1] = x[0]*v[0];
    y[0] = v[1]/(x[1]-v[0]);
    y[1] = x[1]*y[0];
}
```

Lighthouse Example: Real Code for Forward Mode

```
void d1_f(double* x, double* d1_x, double* y, double* d1_y)
//$ad indep x d1_x
//$ad dep y d1_y
{
    double v[2];           double d1_v[2];
    double w1_0 = 0;        double d1_w1_0 = 0;
    ...
    double w1_5 = 0;        double d1_w1_5 = 0;

    d1_w1_0 = d1_x[2];     w1_0 = x[2];
    d1_w1_1 = d1_x[3];     w1_1 = x[3];
    d1_w1_2 = w1_1*d1_w1_0 + w1_0*d1_w1_1;
    w1_2 = w1_0*w1_1;
    d1_w1_3 = 1/(cos(w1_2)*cos(w1_2)) * d1_w1_2;
    w1_3 = tan(w1_2);
    ...
                                using dcc (U. Naumann, RWTH Aachen)
```

Lighthouse Example: Real Code for Forward Mode

```
void d1_f(double* x, double* d1_x, double* y, double* d1_y)
//$ad indep x d1_x
//$ad dep y d1_y
{
    double v[2];           double d1_v[2];
    double w1_0 = 0;        double d1_w1_0 = 0;
    ...
    double w1_5 = 0;        double d1_w1_5 = 0;

    d1_w1_0 = d1_x[2];     w1_0 = x[2];
    d1_w1_1 = d1_x[3];     w1_1 = x[3];
    d1_w1_2 = w1_1*d1_w1_0 + w1_0*d1_w1_1;
    w1_2 = w1_0*w1_1;
    d1_w1_3 = 1/(cos(w1_2)*cos(w1_2)) * d1_w1_2;
    w1_3 = tan(w1_2);
    ...
                                using dcc (U. Naumann, RWTH Aachen)
```

Lighthouse Example: Real Code for Forward Mode

```
void d1_f(double* x, double* d1_x, double* y, double* d1_y)
//$ad indep x d1_x
//$ad dep y d1_y
{
    double v[2];           double d1_v[2];
    double w1_0 = 0;        double d1_w1_0 = 0;
    ...
    double w1_5 = 0;        double d1_w1_5 = 0;

    d1_w1_0 = d1_x[2];     w1_0 = x[2];
    d1_w1_1 = d1_x[3];     w1_1 = x[3];
    d1_w1_2 = w1_1*d1_w1_0 + w1_0*d1_w1_1;
    w1_2 = w1_0*w1_1;
    d1_w1_3 = 1/(cos(w1_2)*cos(w1_2)) * d1_w1_2;
    w1_3 = tan(w1_2);
    ...
                                using dcc (U. Naumann, RWTH Aachen)
```

Lighthouse Example: Real Code for Reverse Mode

```

void b1.f(int& bmode_1, double* x, double* b1_x, double* y, double* b1_y)
//$ad indep x b1_x b1_y
//$ad dep y b1_x
{ double v[2];      double b1_v[2];
  double w1_0 = 0;    double b1_w1_0 = 0;    ...
  double w1_5 = 0;    double b1_w1_5 = 0;
  int save_cs_c = 0;  save_cs_c = cs_c;
  if (bmode_1==1) { // augmented forward section
    cs[cs_c] = 0;    cs_c = cs_c+1;
    fds[fds_c] = v[0];  fds_c = fds_c+1;  v[0] = tan(x[2]*x[3]);
    ...
    fds[fds_c] = y[1];  fds_c = fds_c+1;  y[1] = x[1]*y[0];
    while (cs_c>save_cs_c) { // reverse section
      cs_c = cs_c-1;
      if (cs[cs_c]==0) {
        fds_c = fds_c-1;  y[1] = fds[fds_c];
        w1_0 = x[1];      w1_1 = y[0];      w1_2 = w1_0*w1_1;
        b1_w1_2 = b1_y[1];  b1_y[1] = 0; // adjoint assignment
        b1_w1_0 = w1_1*b1_w1_2;  b1_w1_1 = w1_0*b1_w1_2;  ...
      }
    }
  }
}

```

Lighthouse Example: Real Code for Reverse Mode

```

void b1.f(int& bmode_1, double* x, double* b1_x, double* y, double* b1_y)
//$ad indep x b1_x b1_y
//$ad dep y b1_x
{ double v[2];      double b1_v[2];
  double w1_0 = 0;    double b1_w1_0 = 0;    ...
  double w1_5 = 0;    double b1_w1_5 = 0;
  int save_cs_c = 0;  save_cs_c = cs_c;
  if (bmode_1==1) { // augmented forward section
    cs[cs_c] = 0;    cs_c = cs_c+1;
    fds[fds_c] = v[0];  fds_c = fds_c+1;  v[0] = tan(x[2]*x[3]);
    ...
    fds[fds_c] = y[1];  fds_c = fds_c+1;  y[1] = x[1]*y[0];
    while (cs_c>save_cs_c) { // reverse section
      cs_c = cs_c-1;
      if (cs[cs_c]==0) {
        fds_c = fds_c-1;  y[1] = fds[fds_c];
        w1_0 = x[1];      w1_1 = y[0];      w1_2 = w1_0*w1_1;
        b1_w1_2 = b1_y[1];  b1_y[1] = 0; // adjoint assignment
        b1_w1_0 = w1_1*b1_w1_2;  b1_w1_1 = w1_0*b1_w1_2;  ...
      }
    }
  }
}

```

Lighthouse Example: Real Code for Reverse Mode

```

void b1.f(int& bmode_1, double* x, double* b1_x, double* y, double* b1_y)
//$ad indep x b1_x b1_y
//$ad dep y b1_x
{ double v[2];      double b1_v[2];
  double w1_0 = 0;    double b1_w1_0 = 0;    ...
  double w1_5 = 0;    double b1_w1_5 = 0;
  int save_cs_c = 0;  save_cs_c = cs_c;
  if (bmode_1==1) { // augmented forward section
    cs[cs_c] = 0;    cs_c = cs_c+1;
    fds[fds_c] = v[0];  fds_c = fds_c+1;  v[0] = tan(x[2]*x[3]);
    ...
    fds[fds_c] = y[1];  fds_c = fds_c+1;  y[1] = x[1]*y[0];
  while (cs_c>save_cs_c) { // reverse section
    cs_c = cs_c-1;
    if (cs[cs_c]==0) {
      fds_c = fds_c-1;  y[1] = fds[fds_c];
      w1_0 = x[1];      w1_1 = y[0];      w1_2 = w1_0*w1_1;
      b1_w1_2 = b1_y[1];  b1_y[1] = 0; // adjoint assignment
      b1_w1_0 = w1_1*b1_w1_2;  b1_w1_1 = w1_0*b1_w1_2;  ...
    }
  }
}

```

Lighthouse Example: Real Code for Reverse Mode

```

void b1.f(int& bmode_1, double* x, double* b1_x, double* y, double* b1_y)
//$ad indep x b1_x b1_y
//$ad dep y b1_x
{ double v[2];      double b1_v[2];
  double w1_0 = 0;    double b1_w1_0 = 0;    ...
  double w1_5 = 0;    double b1_w1_5 = 0;
  int save_cs_c = 0;  save_cs_c = cs_c;
  if (bmode_1==1) { // augmented forward section
    cs[cs_c] = 0;    cs_c = cs_c+1;
    fds[fds_c] = v[0];  fds_c = fds_c+1;  v[0] = tan(x[2]*x[3]);
    ...
    fds[fds_c] = y[1];  fds_c = fds_c+1;  y[1] = x[1]*y[0];
  while (cs_c>save_cs_c) { // reverse section
    cs_c = cs_c-1;
    if (cs[cs_c]==0) {
      fds_c = fds_c-1;    y[1] = fds[fds_c];
      w1_0 = x[1];        w1_1 = y[0];    w1_2 = w1_0*w1_1;
      b1_w1_2 = b1_y[1];  b1_y[1] = 0; // adjoint assignment
      b1_w1_0 = w1_1*b1_w1_2;  b1_w1_1 = w1_0*b1_w1_2;    ...
      ...
      using again dcc
    }
  }
}

```

Conclusions Part I



- Differentiation of computer programmes with working accuracy
(Griewank, Kulshreshtha, Walther 2012)

Conclusions Part I

- Differentiation of computer programmes with working accuracy
(Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F)$, $c \in [2, 5/2]$

Conclusions Part I

- Differentiation of computer programmes with working accuracy
(Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F)$, $c \in [2, 5/2]$
= discrete analogon to sensitivity equation

Conclusions Part I

- Differentiation of computer programmes with working accuracy
(Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F)$, $c \in [2, 5/2]$
Reverse mode: $\text{OPS}(\bar{y}^T F'(x)) \leq c \text{OPS}(F)$, $c \in [3, 4]$
 $\text{MEM}(\bar{y}^T F'(x)) \sim \text{OPS}(F)$,

Conclusions Part I

- Differentiation of computer programmes with working accuracy
(Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F)$, $c \in [2, 5/2]$
Reverse mode: $\text{OPS}(\bar{y}^T F'(x)) \leq c \text{OPS}(F)$, $c \in [3, 4]$
 $\text{MEM}(\bar{y}^T F'(x)) \sim \text{OPS}(F)$,
= discrete analogon to adjoint equation

Conclusions Part I

- Differentiation of computer programmes with working accuracy
(Griewank, Kulshreshtha, Walther 2012)
 - Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F)$, $c \in [2, 5/2]$
Reverse mode: $\text{OPS}(\bar{y}^T F'(x)) \leq c \text{OPS}(F)$, $c \in [3, 4]$
 $\text{MEM}(\bar{y}^T F'(x)) \sim \text{OPS}(F)$,
- ⇒ Gradients are cheap \sim Function costs!!

Conclusions Part I

- Differentiation of computer programmes with working accuracy
(Griewank, Kulshreshtha, Walther 2012)
 - Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F)$, $c \in [2, 5/2]$
Reverse mode: $\text{OPS}(\bar{y}^T F'(x)) \leq c \text{OPS}(F)$, $c \in [3, 4]$
 $\text{MEM}(\bar{y}^T F'(x)) \sim \text{OPS}(F)$,
- ⇒ Gradients are cheap \sim Function costs!!

Conclusions Part I

- Differentiation of computer programmes with working accuracy (Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F)$, $c \in [2, 5/2]$
Reverse mode: $\text{OPS}(\bar{y}^T F'(x)) \leq c \text{OPS}(F)$, $c \in [3, 4]$
 $\text{MEM}(\bar{y}^T F'(x)) \sim \text{OPS}(F)$,

 \implies Gradients are cheap \sim Function costs!!

- Structure exploitation indispensable
- many AD tools, e.g. ADOL-C
- AD in real-life applications, e.g., backpropagation for ML

Conclusions Part I

- Differentiation of computer programmes with working accuracy (Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\text{OPS}(F'(x)\dot{x}) \leq c \text{OPS}(F)$, $c \in [2, 5/2]$
Reverse mode: $\text{OPS}(\bar{y}^T F'(x)) \leq c \text{OPS}(F)$, $c \in [3, 4]$
 $\text{MEM}(\bar{y}^T F'(x)) \sim \text{OPS}(F)$,

 \implies Gradients are cheap \sim Function costs!!

- Structure exploitation indispensable
- many AD tools, e.g. ADOL-C
- AD in real-life applications, e.g., backpropagation for ML

(Griewank, Walther 2008), (Naumann 2012), www.autodiff.org