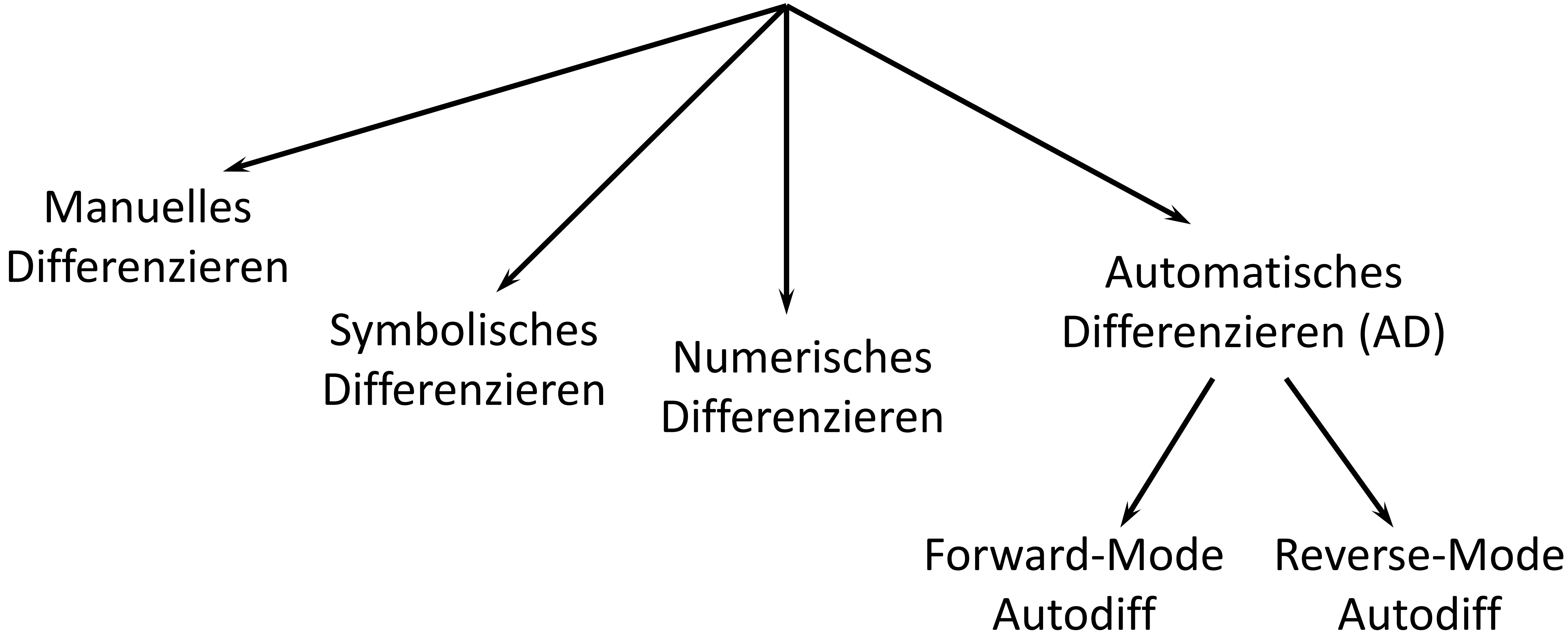
Automatisches Differenzieren

Prof. Dr. Jürgen Brauer
www.juergenbrauer.org

Ansätze zum Differenzieren



Manuelles Differenzieren

$$f(x_1, x_2) \coloneqq x_1 x_2 + \sin(x_1)$$

Ableitungsregeln in Leibniz Notation:

$$\frac{d(f+g)}{dx} = \frac{df}{dx} + \frac{dg}{dx}$$

$$\frac{d(fg)}{dx} = \frac{df}{dx} g + f \frac{dg}{dx}$$

$$\frac{d(c * f(x))}{dx} = c * \frac{df(x)}{dx}$$

$$\frac{d(f(g))}{dx} = \frac{df}{dg} \frac{dg}{dx}$$

$$\frac{d}{dx} \sin x = \cos x$$

➡ $$\frac{df}{dx_1} = x_2 + \cos(x_1) \qquad \frac{df}{dx_2} = x_1$$

☹ Fehleranfällig

Symbolisches Differenzieren

```python
# Use SymPy library - SymPy is a Python library for symbolic mathematics. It aims to
# become a full-featured computer algebra system (CAS) as Mathematica or Maple.
from sympy import *

x1 = Symbol('x1')
x2 = Symbol('x2')
f = x1*x2+sin(x1)
derivative1 = diff(f,x1)
derivative2 = diff(f,x2)
print("f(x1,x2) = "+str(f))
print("df/dx1   = "+str(derivative1))
print("df/dx2   = "+str(derivative2))
print("df/dx1 (1.0,2.0) = " + str(derivative1.evalf(subs={x1: 1.0, x2:2.0})))
print("df/dx2 (1.0,2.0) = " + str(derivative2.evalf(subs={x1: 1.0, x2:2.0}))
```

```
f(x1,x2) = x1*x2 + sin(x1)
df/dx1   = x2 + cos(x1)
df/dx2   = x1
df/dx1 (1.0,2.0) = 2.54030230586814
df/dx2 (1.0,2.0) = 1.00000000000000
```

Symbolisches Differenzieren (Vereinfachung)

```
# Use SymPy library
from sympy import *


x = Symbol('x')
y = Symbol('y')
f = (x+x*y)/x
print("f = "+str(f))
print("simplified = "+str(simplify(f)))
```

➡

```
f = (x*y + x)/x
simplified = y + 1
```

Symbolisches Differenzieren

```
import numpy as np

def f(x1,x2):
    result = 0
    if x1<x2:
        result = x1*x2+np.sin(x1)
    else:
        result = np.pi;
        result += x1*x2;
        for i in range(0,5):
            result += x1**2
    return result
```

→ Ableitung von f?

☹ Funktioniert nicht, wenn das Modell als Programm vorliegt

Symbolisches Differenzieren (mit menschlicher Hilfe)

```
import numpy as np
from sympy import *

def f(x1,x2):
    result = 0
    if x1<=x2: # x1*x2 + sin(x1)
        result = x1*x2+np.sin(x1)
    else: # pi + x1*x2 + 5*x1^2
        result = np.pi;
        result += x1*x2;
        for i in range(0,5):
            result += x1**2
    return result
```

Symbolisches Differenzieren (mit menschlicher Hilfe)

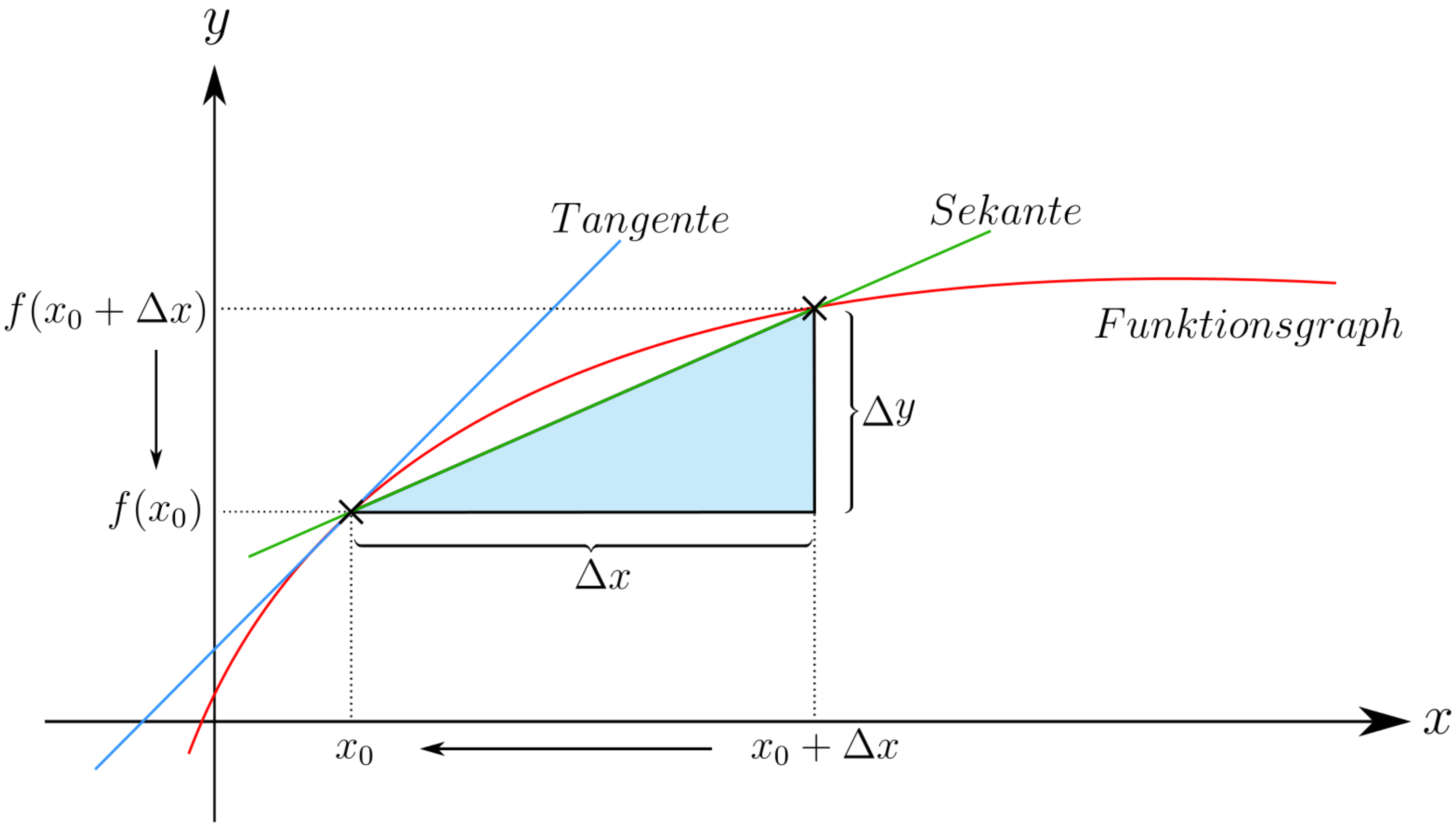
```
x1 = Symbol('x1')
x2 = Symbol('x2')
f1 = x1*x2+sin(x1)
f2 = pi + x1*x2 + 5*x1**2
derivative1 = diff(f2,x1)
derivative2 = diff(f2,x2)
print("df2/dx1 = "+str(derivative1))
print("df2/dx2 = "+str(derivative2))
print("df2/dx1 (2.0,1.0) = " + str(derivative1.evalf(subs={x1: 2.0, x2:1.0})))
print("df2/dx2 (2.0,1.0) = " + str(derivative2.evalf(subs={x1: 2.0, x2:1.0})))
```

➡

```
df2/dx1 = 10*x1 + x2
df2/dx2 = x1
df2/dx1 (2.0,1.0) = 21.0000000000000
df2/dx2 (2.0,1.0) = 2.00000000000000
```

Numerisches Differenzieren

$$f'(x_0) = \lim_{\Delta x \to 0} \left(\frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \right)$$



Numerisches Differenzieren

$f: \mathbb{R} \to \mathbb{R}$

$$f'(x_0) \approx \left(\frac{f(x_0 + h) - f(x_0)}{h}\right)$$

$f: \mathbb{R}^n \to \mathbb{R}$

$$with\ h \ll 1$$

$$\frac{d\mathrm{f}(x_0)}{dx_i} \approx \left(\frac{f(x_0 + h * e_i) - f(x_0)}{h}\right)$$

$$with\ e_i = (0, \ldots 0{,}1{,}0, \ldots, 0)^T$$

i-1 i i+1

Numerisches Differenzieren (Python Beispiel)

```python
import numpy as np

def f(x1,x2):
    result = 0
    if x1<=x2: # x1*x2 + sin(x1)
        result = x1*x2+np.sin(x1)
    else: # pi + x1*x2 + 5*x1^2
        result = np.pi;
        result += x1*x2;
        for i in range(0,5):
            result += x1**2
    return result

def dfdx1(x1,x2):
    h = 0.000001
    return ( f(x1+h,x2) - f(x1,x2) ) / h

print("dfdx1(2,1) = " + str(dfdx1(2,1)))
```

➡ dfdx1(2,1) = 21.000005002491662

☹ Das ist nicht genau

10*x1 + x2 = 21.0!

Numerisches Differenzieren (verschiedene h)

```
def dfdx1(x1,x2,h):
    return ( f(x1+h,x2) - f(x1,x2) ) / h

h = 1.0
for i in range(1,20):
    h = h/10.0
    print("h = %e " % h + " -->
dxdx1(2,1,h) = " + str(dfdx1(2,1,h)))
```

h = 1.000000e-01 --> dxdx1(2,1,h) = 21.50000000000002
h = 1.000000e-02 --> dxdx1(2,1,h) = 21.049999999999258
h = 1.000000e-03 --> dxdx1(2,1,h) = 21.004999999995277
h = 1.000000e-04 --> dxdx1(2,1,h) = 21.000499999992428
h = 1.000000e-05 --> dxdx1(2,1,h) = 21.000050000097303
h = 1.000000e-06 --> dxdx1(2,1,h) = 21.00000500249166
h = 1.000000e-07 --> dxdx1(2,1,h) = 21.000000423043726
h = 1.000000e-08 --> dxdx1(2,1,h) = 20.999999961190948
h = 1.000000e-09 --> dxdx1(2,1,h) = 21.000001737547784
h = 1.000000e-10 --> dxdx1(2,1,h) = 20.99998397397939
h = 1.000000e-11 --> dxdx1(2,1,h) = 21.000090555389754
h = 1.000000e-12 --> dxdx1(2,1,h) = 21.000090555389757
h = 1.000000e-13 --> dxdx1(2,1,h) = 20.925483568134947
h = 1.000000e-14 --> dxdx1(2,1,h) = 21.671553440683052
h = 1.000000e-15 --> dxdx1(2,1,h) = 17.763568394002505
h = 1.000000e-16 --> dxdx1(2,1,h) = 0.0
h = 1.000000e-17 --> dxdx1(2,1,h) = 0.0
h = 1.000000e-18 --> dxdx1(2,1,h) = 0.0
h = 1.000000e-19 --> dxdx1(2,1,h) = 0.0

Numerisches Differenzieren (Hauptproblem)

$f: \mathbb{R}^2 \to \mathbb{R}$

```
def dfdx1(x1,x2,h):
    return ( f(x1+h,x2) - f(x1,x2) ) / h

def dfdx2(x1,x2,h):
    return ( f(x1,x2+h) - f(x1,x2) ) / h
```

2+1
Modell-Evaluierungen
notwendig

$f: \mathbb{R}^n \to \mathbb{R}$

n+1
Modell-Evaluierungen
notwendig

Ansätze zum Differenzieren

- Manuelles Differenzieren
  ☹ Exakt, aber fehleranfällig
- Symbolisches Differenzieren
  ☹ Exakt, aber funktioniert nicht, wenn Modell als Programm vorliegt
- Numerisches Differenzieren
  ☹ Nur eine Approximation
  ☹ Hoher Rechenaufwand
- Automatisches Differenzieren (AD)
  - Forward-Mode Autodiff
  - Reverse-Mode Autodiff

Automatisches Differenzieren (AD)

nutzt aus: jede Berechnung / jedes Modell besteht aus einer Folge von
elementaren Rechenoperationen (+,-,/,*, …) und/oder
elementaren Funktionsaufrufen (sin, cos, exp, log, …)

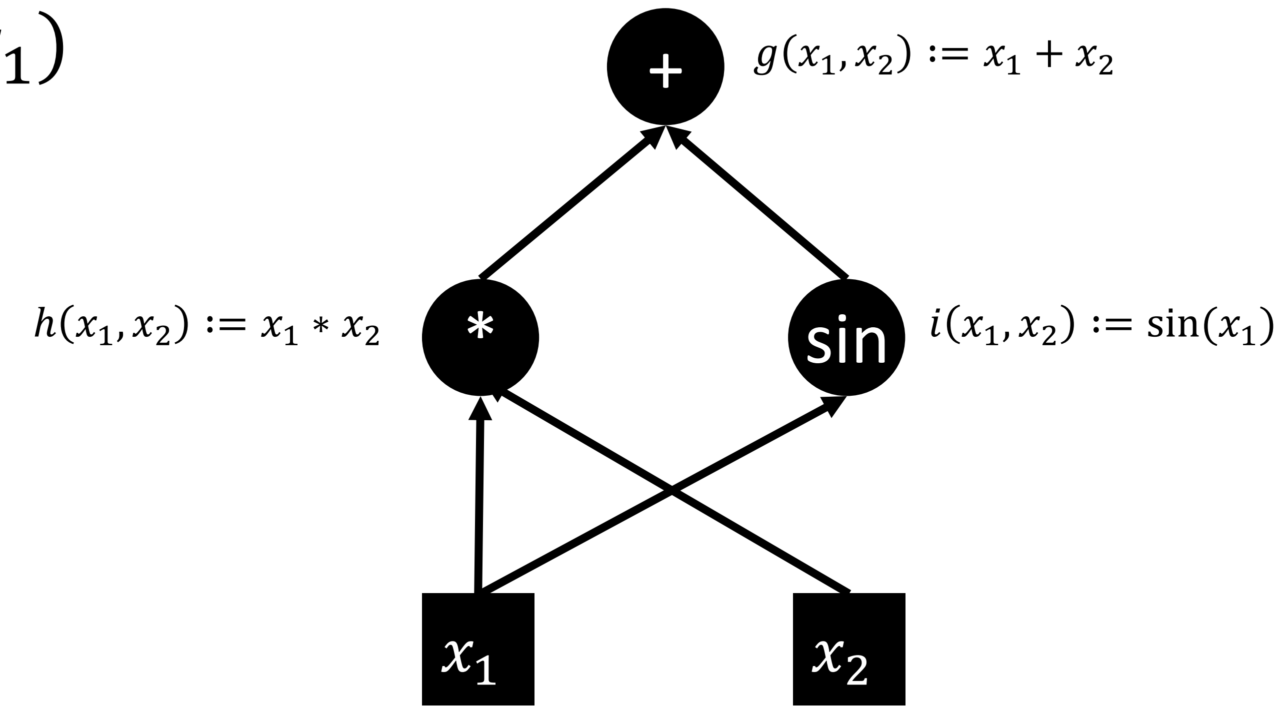
$$y \coloneqq f(x_1, x_2) \coloneqq x_1 * x_2 + \sin(x_1)$$

$$y = g(h(x_1, x_2), i(x_1, x_2))$$

$$h(x_1, x_2) \coloneqq x_1 * x_2$$

$$i(x_1, x_2) \coloneqq \sin(x_1)$$

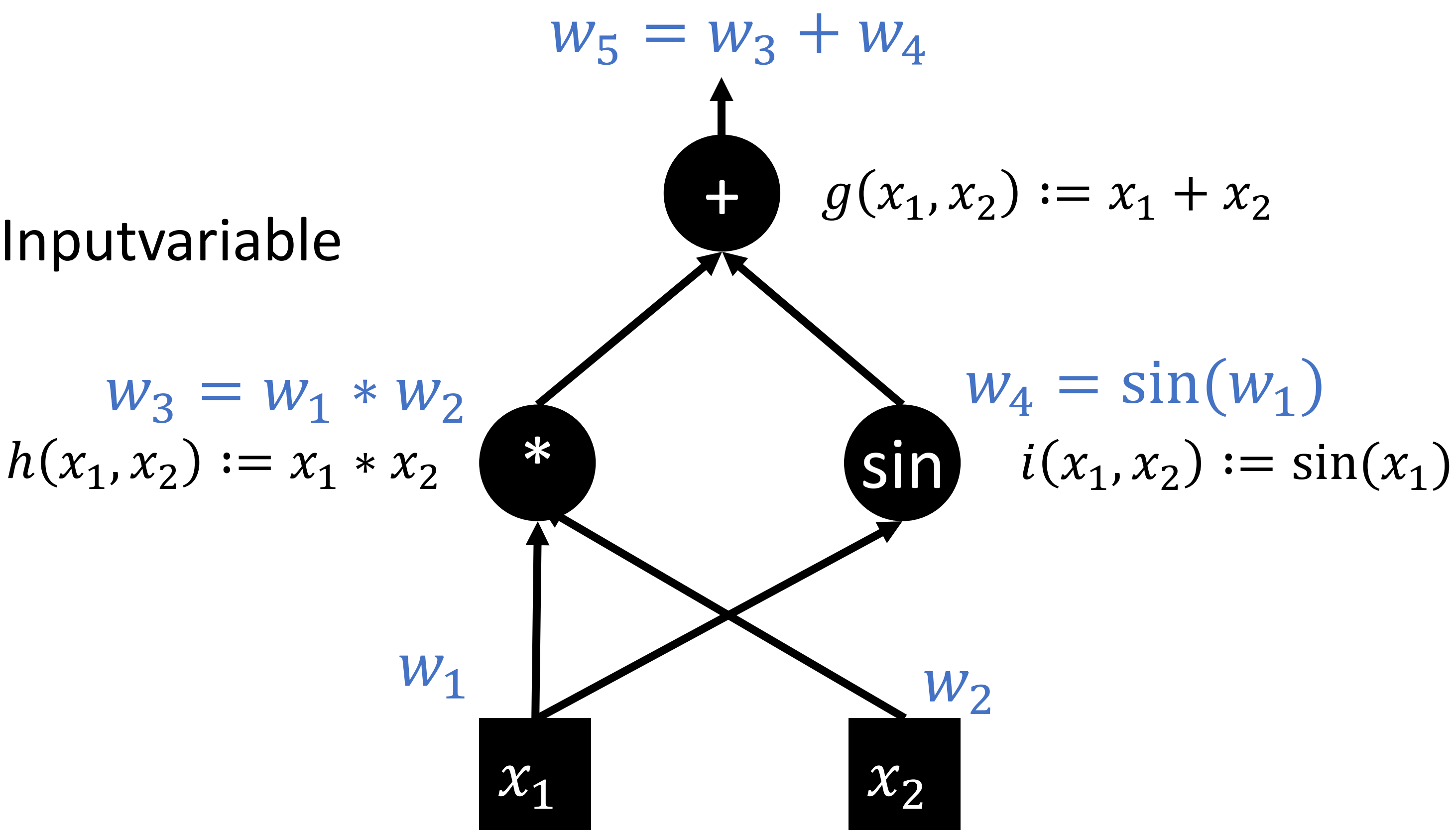
$$g(x_1, x_2) \coloneqq x_1 + x_2$$



Forward-Mode Autodiff

$$\dot{w}_i := \frac{dw_i}{dx}$$

Ableitung bzgl. Inputvariable



Forward-Mode Autodiff

$$\frac{dg}{dx_1} = \frac{dw_5}{dw_1}?$$

$\dot{w}_5 := \dot{w}_3 + \dot{w}_4$

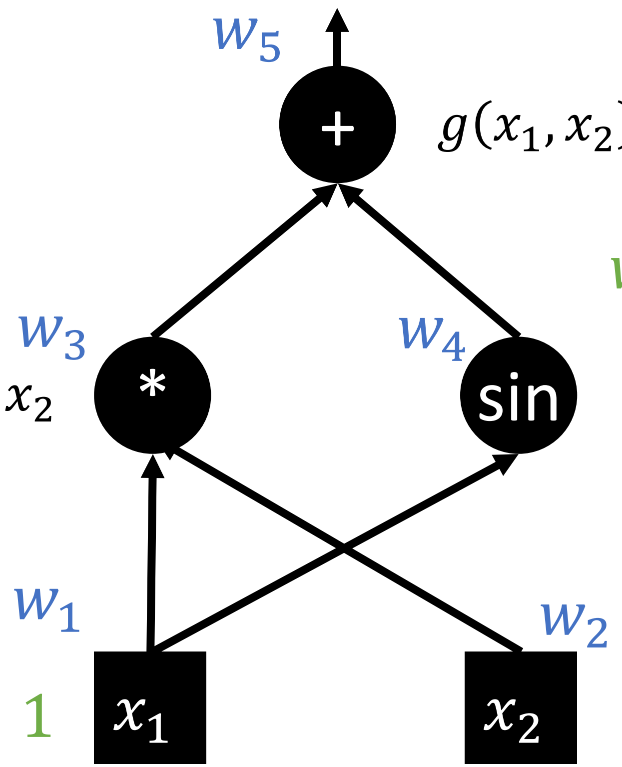
$g(x_1, x_2) := x_1 + x_2$

$\dot{w}_3 := \dot{w}_1 * w_2 + w_1 * \dot{w}_2$

$\dot{w}_4 := \cos(w_1) * \dot{w}_1$

$h(x_1, x_2) := x_1 * x_2$

$i(x_1, x_2) := \sin(x_1)$

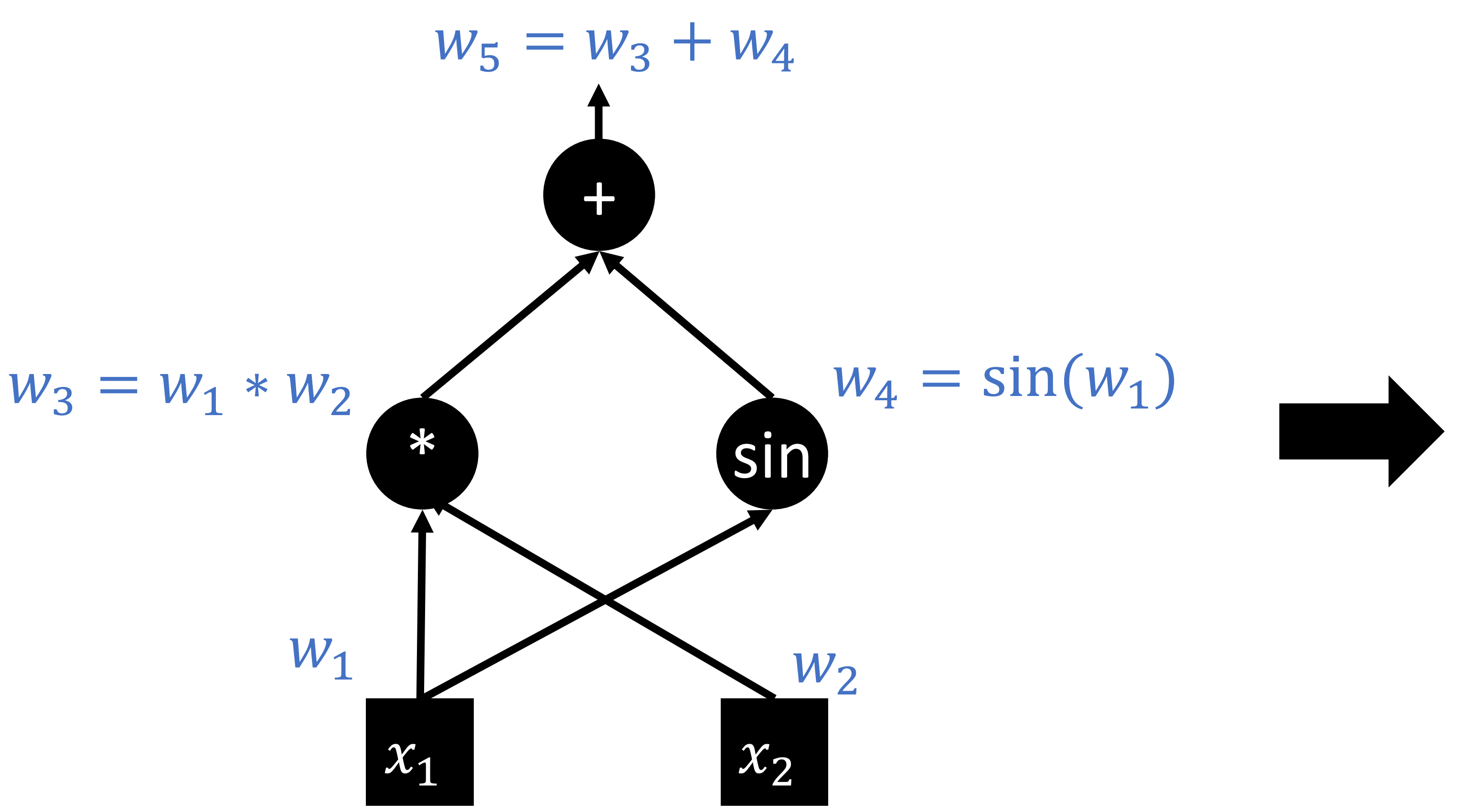


$\dot{w}_1 := 1$

$\dot{w}_2 := 0$

"In forward accumulation AD, one first fixes the *independent variable (here: w_1, w_2)* to which differentiation is performed and computes the derivative of each sub-expression recursively. In a pen-and-paper calculation, one can do so by repeatedly substituting the derivative of the *inner* functions in the chain rule"

Forward-Mode Autodiff




```python
import numpy as np

def f(x1,x2):

    w1 = x1
    w2 = x2
    w3 = w1*w2
    w4 = np.sin(w1)
    w5 = w3 + w4
    return w5
```

Forward-Mode Autodiff

```
import numpy as np

def f(x1,x2):

    w1 = x1
    dw1 = 1
    w2 = x2
    dw2 = 0
    w3 = w1*w2
    dw3 = dw1 * w2 + w1 * dw2
    w4 = np.sin(w1)
    dw4 = np.cos(w1) * dw1
    w5 = w3 + w4
    dw5 = dw3 + dw4
    return w5, dw5


val, deriv = f(1,2)
print("df/dx1 (1,2)=" + str(deriv))
print("df/dx1 (1,2)=" + str(2 + np.cos(1)))
# df/dx1 = x2 + cos(x1)
```

➡

```
df/dx1 (1,2)=2.5403023058681398
df/dx1 (1,2)=2.5403023058681398
```

Forward-Mode Autodiff

$$\frac{dg}{dx_1} = \frac{dw_5}{dw_1}$$

✓

$$\frac{dg}{dx_2} = \frac{dw_5}{dw_2}$$

?

Forward-Mode Autodiff

```
import numpy as np

def f(x1,x2):

    w1 = x1
    dw1 = 0
    w2 = x2
    dw2 = 1
    w3 = w1*w2
    dw3 = dw1 * w2 + w1 * dw2
    w4 = np.sin(w1)
    dw4 = np.cos(w1) * dw1
    w5 = w3 + w4
    dw5 = dw3 + dw4
    return w5, dw5

val, deriv = f(1,2)
print("df/dx2 (1,2)=" + str(deriv)) # df/dx2 = x1
```

➡

```
df/dx2 (1,2)=1.0
```

Problem der Forward-Mode Autodiff

$f: \mathbb{R}^n \to \mathbb{R}$ ➡ n Durchläufe mit unterschiedlichen Seed Values benötigt

$f: \mathbb{R}^n \to \mathbb{R}^m$ ➡ nur sinnvoll, wenn $n \ll m$ ist

Schleifen und Forward-Mode Autodiff

```python
import numpy as np

def f(x1,x2):
    result = 0
    if x1<=x2: # x1*x2 + sin(x1)
        result = x1*x2+np.sin(x1)
    else: # pi + x1*x2 + 5*x1^2
        result = np.pi;
        result += x1*x2;
        for i in range(0,5):
            result += x1**2
    return result
```

Schleifen und Forward-Mode Autodiff

```python
import numpy as np

def f(x1,x2):

    w1 = x1
    w2 = x2
    dw1 = 1
    dw2 = 0
    result = 0
    if x1<x2:
        # f(x1,x2) = x1*x2 + sin(x1)
        w3 = w1 * w2
        dw3 = dw1 * w2 + w1 * dw2
        w4 = np.sin(w1)
        dw4 = np.cos(w1) * dw1
        w5 = w3 + w4
        dw5 = dw3 + dw4
        return w5, dw5
    else:
```

```python
        # f(x1,x2) = pi + x1*x2 + 5*x1^2
        # df/dx1 = x2 + 10*x1
        w3 = np.pi
        dw3 = 0
        w4 = w1*w2
        dw4 = dw1*w2 + w1*dw2
        w5 = w3+w4
        dw5 = dw3 + dw4
        w7 = w5
        dw7 = dw5
        for i in range(0,5):
            w6 = w1**2
            dw6 = 2*w1
            w7 = w7 + w6
            dw7 = dw7 + dw6
        return w7, dw7

val, deriv = f(2,1)
print("dfdx1(2,1) = " + str(deriv))
```

➡ dfdx1(2,1) = 21

Ansätze zum Differenzieren

Manuelles Differenzieren

☹

Exakt, aber fehleranfällig

Symbolisches Differenzieren

☹

Exakt, aber funktioniert nicht, wenn Modell als Programm vorliegt

Numerisches Differenzieren

☹

Nur eine Approximation

☹

Hoher Rechenaufwand

Automatisches Differenzieren (AD)

- **Forward-Mode Autodiff**

  ☹ Exakt, aber rechenaufwändig, für $f: \mathbb{R}^n \to \mathbb{R}^m$ falls $n \gg m$

- **Reverse-Mode Autodiff**

Automatisches Differenzieren

$$y(x) = c\Big(b\big(a(x)\big)\Big) = c\big(b(a(w_0))\big) = c\big(b(w_1)\big) = c(w_2) = w_3$$

$$\frac{dy}{dx} = \underbrace{\overbrace{\frac{dw_3}{dw_2}}^{3.}}_{1.} \underbrace{\overbrace{\frac{dw_2}{dw_1}}^{2.}}_{2.} \underbrace{\overbrace{\frac{dw_1}{dw_0}}^{1.}}_{3.}$$

Kettenregel

3. 2. 1. ➡ Forward-Mode Autodiff

1. 2. 3. ➡ Reverse-Mode Autodiff

Reverse-Mode Autodiff

$$\frac{dg}{dx_1} = \frac{dw_5}{dw_1}?$$

$$\frac{dg}{dx_2} = \frac{dw_5}{dw_2}?$$

$$\bar{w}_i := \frac{dy}{dw_i}$$

Ableitung einer Output-variable bzgl. eines Knotens w_i

$$\bar{w}_5 := 1$$

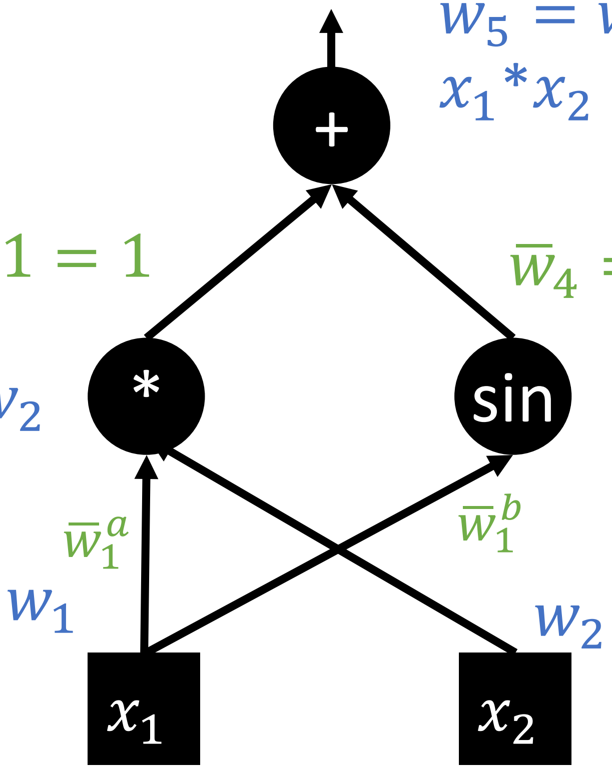
$$w_5 = w_3 + w_4 = x_1 * x_2 + \sin(x_1)$$

$$\bar{w}_3 = \bar{w}_5 \frac{dw_5}{dw_3} = 1 * 1 = 1$$

$$\bar{w}_4 = \bar{w}_5 \frac{dw_5}{dw_4} = 1 \;* 1$$

$$w_3 = w_1 * w_2$$

$$w_4 = \sin(w_1)$$



$$\bar{w}_1^a = \bar{w}_3 \frac{dw_3}{dw_1} = 1 \;* w_2$$

$$\bar{w}_1^b = \bar{w}_4 \frac{dw_4}{dw_1} = 1 \;* \cos(w_1)$$

$$\bar{w}_2 = \bar{w}_3 \frac{dw_3}{dw_2} = w_1$$

$$\bar{w}_1 = \bar{w}_1^a + \bar{w}_1^b = w_2 + \cos(w_1)$$

"In reverse accumulation AD, one first fixes the *dependent variable (here: w_5)* to be differentiated and computes the derivative *with respect to* each sub-expression recursively. In a pen-and-paper calculation, one can perform the equivalent by repeatedly substituting the derivative of the *outer* functions in the chain rule"

Reverse-Mode Autodiff: Direkte Berechnung immer möglich?

$\bar{w}_1 = \bar{w}_1^a + \bar{w}_1^b = w_2 + \cos(w_1)$

$\bar{w}_2 = \bar{w}_3 \frac{dw_3}{dw_2} = w_1$

⬇

$$\frac{df}{dx_1}(1,2) = 2 + \cos(1) = 2.5403023058681398$$

$$\frac{df}{dx_2}(1,2) = 1$$

Reverse-Mode Autodiff (Funktionsvariante)

$$\frac{dg}{dx_1} = \frac{dw_5}{dw_1}?$$

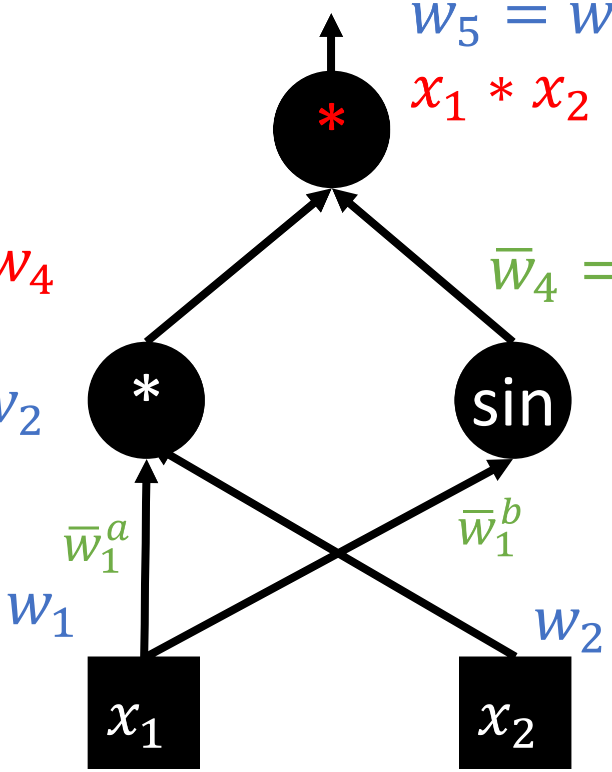
$$\frac{dg}{dx_2} = \frac{dw_5}{dw_2}?$$

$$\bar{w}_i := \frac{dy}{dw_i}$$

Ableitung einer Output-variable bzgl. eines Knotens w_i

$$\bar{w}_5 := 1$$

$$w_5 = w_3 * w_4 = x_1 * x_2 * \sin(x_1)$$



$$\bar{w}_3 = \bar{w}_5 \frac{dw_5}{dw_3} = 1 * w_4$$

$$\bar{w}_4 = \bar{w}_5 \frac{dw_5}{dw_4} = 1 * w_3$$

$$w_3 = w_1 * w_2$$

$$w_4 = \sin(w_1)$$

$$\bar{w}_1^a = \bar{w}_3 \frac{dw_3}{dw_1} = w_4 * w_2$$

$$\bar{w}_1^b = \bar{w}_4 \frac{dw_4}{dw_1} = w_3 * \cos(w_1)$$

$$\bar{w}_2 = \bar{w}_3 \frac{dw_3}{dw_2} = w_4 * w_1$$

$$\bar{w}_1 = \bar{w}_1^a + \bar{w}_1^b = w_4 * w_2 + w_3 * \cos(w_1)$$

“In reverse accumulation AD, one first fixes the *dependent variable (here: w_5)* to be differentiated and computes the derivative *with respect to* each sub-expression recursively. In a pen-and-paper calculation, one can perform the equivalent by repeatedly substituting the derivative of the *outer* functions in the chain rule”

Reverse-Mode Autodiff (Vorwärts)

$$\bar{w}_5 := 1$$

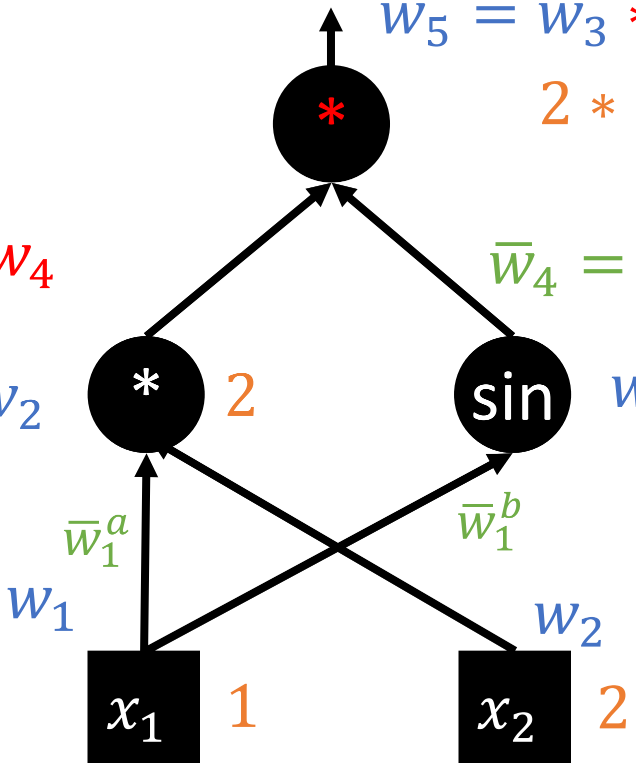
$$w_5 = w_3 * w_4 \qquad 2 * 0.841 = 1.682$$

$$\bar{w}_3 = \bar{w}_5 \frac{dw_5}{dw_3} = 1 * w_4$$

$$\bar{w}_4 = \bar{w}_5 \frac{dw_5}{dw_4} = 1 * w_3$$

$$w_3 = w_1 * w_2$$

$$w_4 = \sin(w_1) \qquad 0.841$$



$$\bar{w}_1^a = \bar{w}_3 \frac{dw_3}{dw_1} = w_4 * w_2$$

$$\bar{w}_2 = \bar{w}_3 \frac{dw_3}{dw_2} = w_4 * w_1$$

$$\bar{w}_1^b = \bar{w}_4 \frac{dw_4}{dw_1} = w_3 * \cos(w_1)$$

$$\bar{w}_1 = \bar{w}_1^a + \bar{w}_1^b = w_4 * w_2 + w_3 * \cos(w_1)$$

Reverse-Mode Autodiff (Rückwärts)

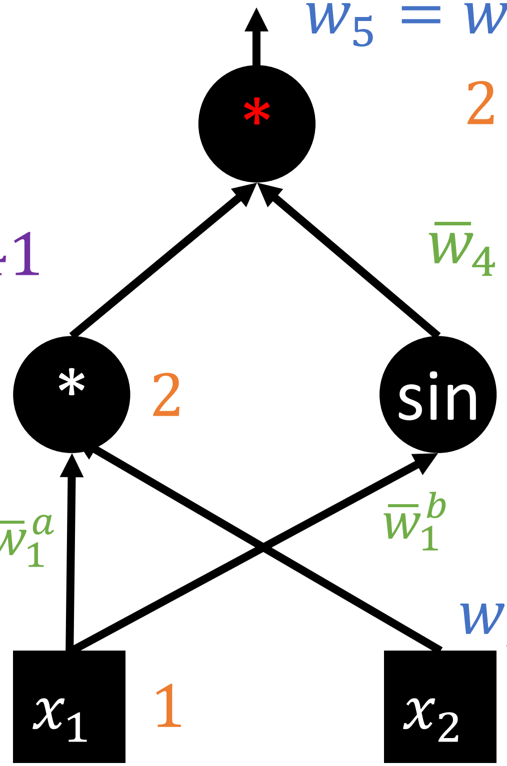
$\bar{w}_5 := 1$

$w_5 = w_3 * w_4$

$2 * 0.841 = 1.68$

$\bar{w}_3 = \bar{w}_5 \dfrac{dw_5}{dw_3} = 1 * w_4 = 0.841$

$\bar{w}_4 = \bar{w}_5 \dfrac{dw_5}{dw_4} = 1 * w_3 = 2$



$w_3 = w_1 * w_2$ 2

$w_4 = \sin(w_1)$ 0.84

w_1

w_2

x_1 1

x_2 2

$\bar{w}_1^a = \bar{w}_3 \dfrac{dw_3}{dw_1} = w_4 * w_2 = 0.84 * 2$

$\bar{w}_2 = \bar{w}_3 \dfrac{dw_3}{dw_2} = w_4 * w_1 = 0.84 * 1$

$\bar{w}_1^b = \bar{w}_4 \dfrac{dw_4}{dw_1} = w_3 * \cos(w_1) = 2 * \cos(1) = 1.08$

$\bar{w}_1 = \bar{w}_1^a + \bar{w}_1^b = w_4 * w_2 + w_3 * \cos(w_1) = 1.68 + 1.08 = 2.76$

Reverse-Mode Autodiff (Überprüfung der AutoDiff-Ableitungsergebnisse)

$$f(x_1, x_2) := x_1 * \ x_2 * \sin(x_1)$$

$$\frac{df}{dx_1} = x_2 * (1 * \sin(x_1) + x_1 * \cos(x_1))$$

$$\frac{df}{dx_1}(1{,}2) = 2 * (0.84 + 1 * 0.54) = 2.76$$

$$\frac{df}{dx_2} = x_1 * \sin(x_1)$$

$$\frac{df}{dx_2}(1{,}2) = 1 * \sin(1) = 0.84$$