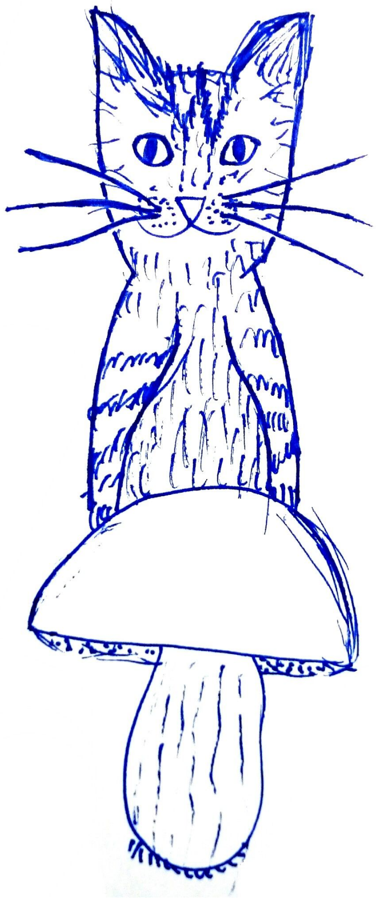
Automatic Differentiation and Other Building Blocks of Deep Learning

Adam Celarek

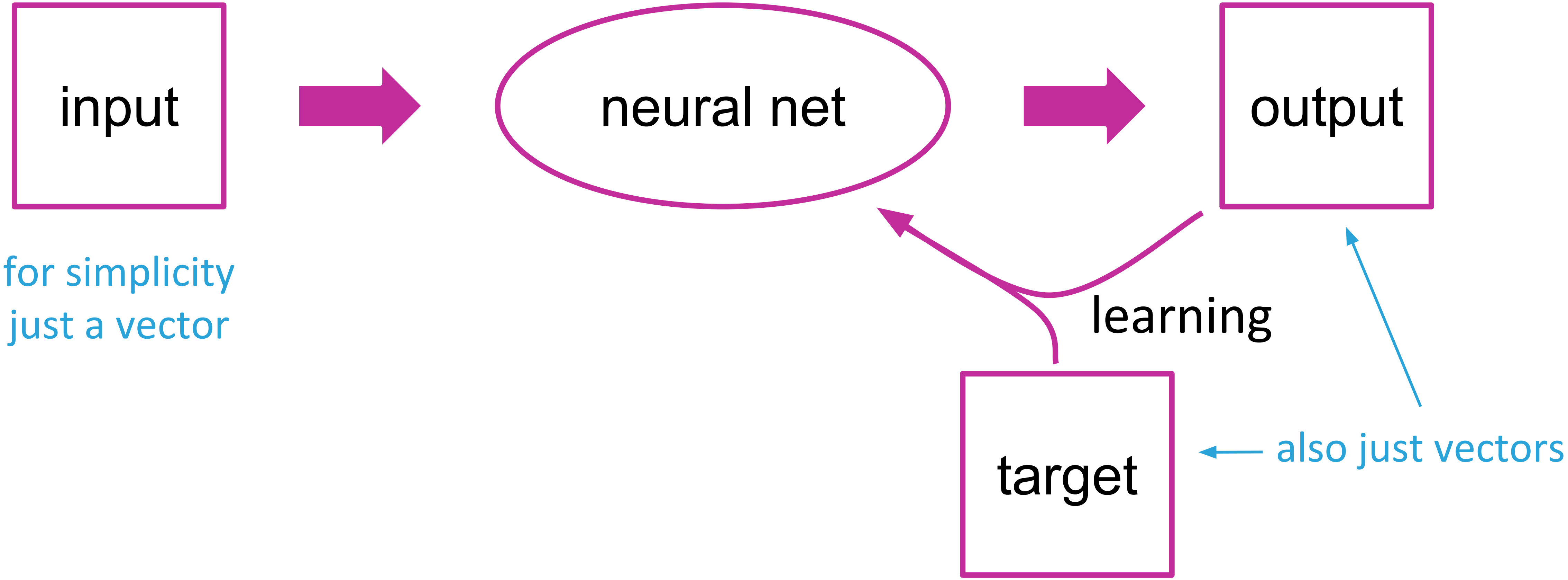
Research Unit of Computer Graphics
Institute of Visual Computing & Human-Centered Technology
TU Wien, Austria

Overview

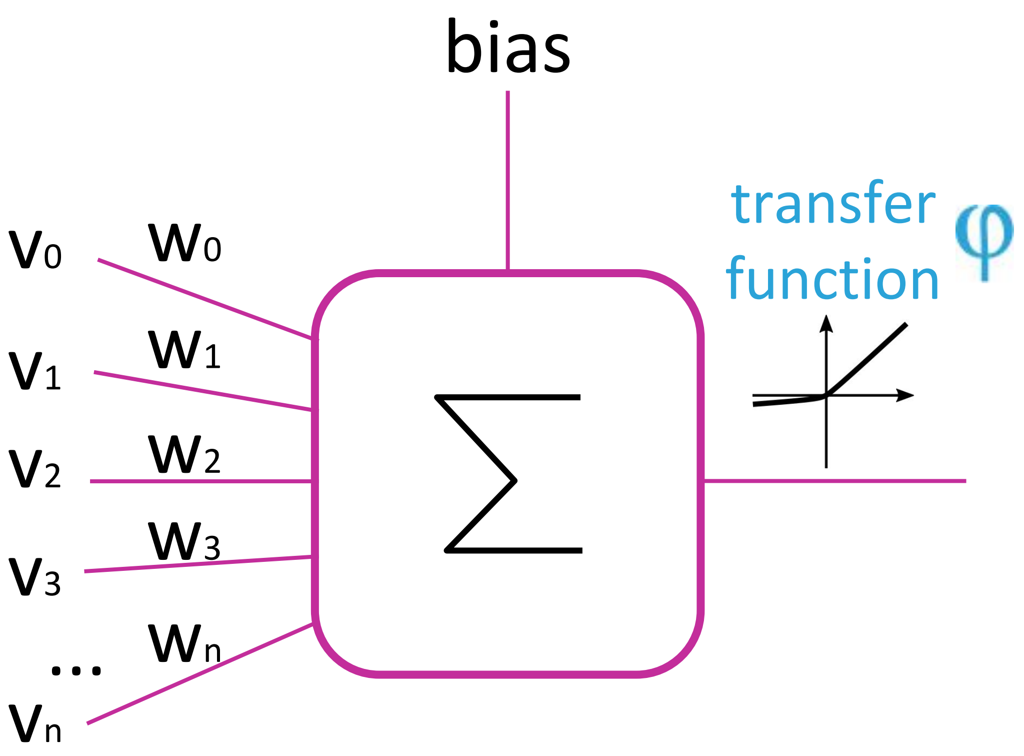
- Deep Neural Networks (DNN)
  - A Neuron and its activation function
  - Layers
- Learning
  - Cost function, Gradient descent
  - Automatic differentiation
- Implementation / Demo

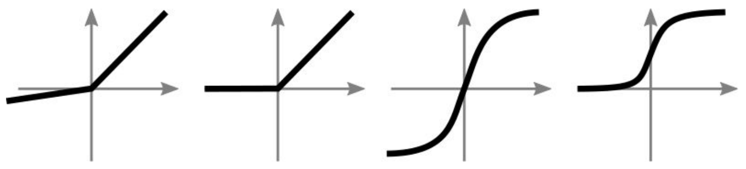


Deep Neural Networks



A Neuron and its activation function

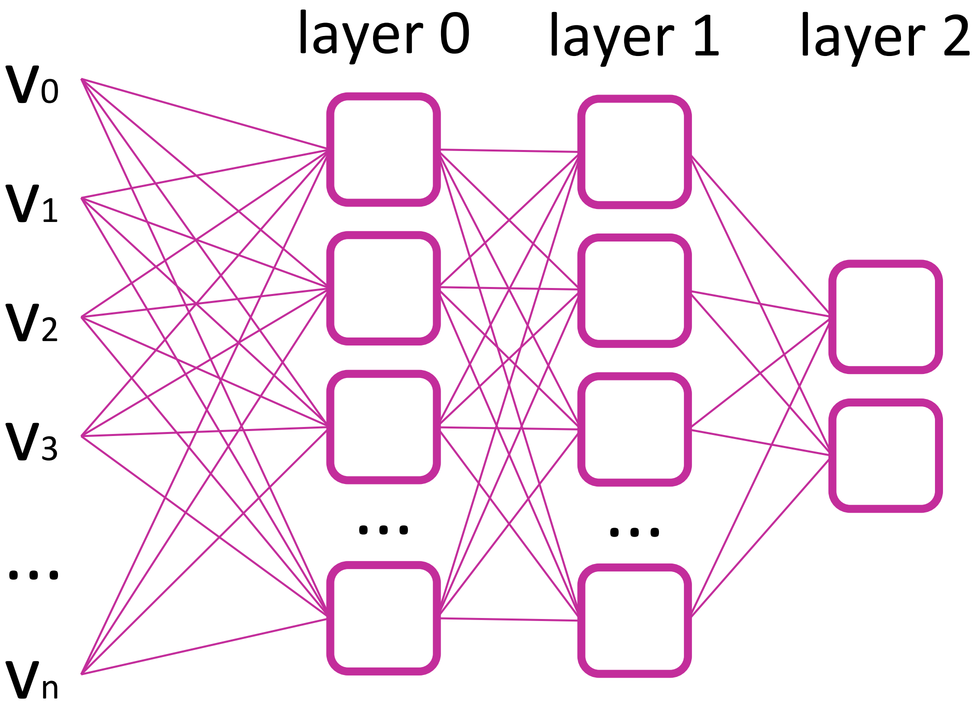




several options for transfer function

$$\mathrm{neuron}(\vec{v}) = \varphi(\vec{w}^T \vec{v} + b)$$

Layers



$$\mathrm{net}(\vec{v}) = \varphi_{\mathrm{out}}\left(W_2\varphi\left(W_1\varphi\left(W_0\vec{v}+\vec{b_0}\right)+\vec{b_1}\right)+\vec{b_2}\right)$$

Learning / Cost function, Gradient descent

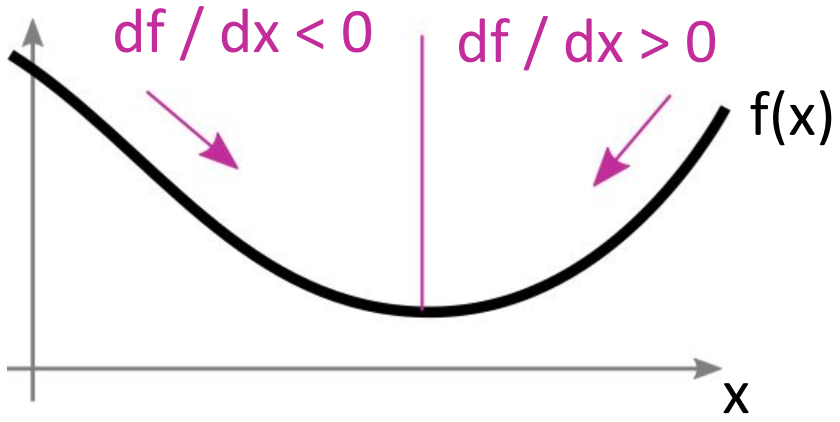
$$\mathrm{net}(\vec{v}, \underbrace{W_0, W_1, W_2, \vec{b}_0, \vec{b}_1, \vec{b}_2}_{\text{learn from examples}}) = \varphi_{\mathrm{out}}\left(W_2 \varphi(W_1 \varphi(W_0 \vec{v} + \vec{b}_0) + \vec{b}_1) + \vec{b}_2\right)$$

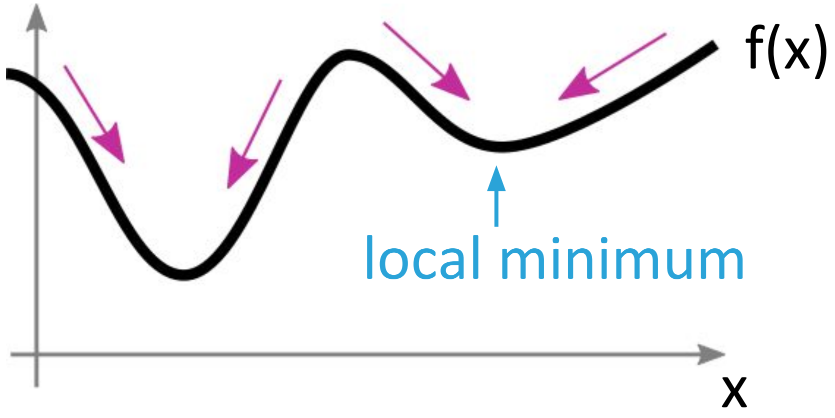
$$\underbrace{\begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \\ \dots \\ \vec{v}_M \end{pmatrix}}_{\text{input}} \longrightarrow \underbrace{\begin{pmatrix} \vec{t}_1 \\ \vec{t}_2 \\ \vec{t}_3 \\ \dots \\ \vec{t}_M \end{pmatrix}}_{\text{target}}$$

Change weights and biases in a way that the input is optimally mapped to the target
=> Minimise cost function using gradient descent.

compute gradient (derivative) of f(x) and go in opposite direction

$$x_{n+1} = x_n - \alpha * (d\, f(x_n) / d\, x_n)$$





Learning / Cost function, Gradient descent

$$\begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \\ \dots \\ \vec{v}_M \end{pmatrix} \longrightarrow \begin{pmatrix} \vec{t}_1 \\ \vec{t}_2 \\ \vec{t}_3 \\ \dots \\ \vec{t}_M \end{pmatrix}$$

Minimise cost function using gradient descent.

$$\mathrm{cost}(W_0, W_1, W_2, \vec{b}_0, \vec{b}_1, \vec{b}_2) = \frac{1}{M}\sum_m^M \underbrace{\left(\mathrm{net}(\vec{v}_m, \dots) - \vec{t}_m\right)^2}_{\text{more generally}}$$

$$\mathrm{loss}\left(\mathrm{net}(\vec{v}_m, \dots), \vec{t}_m\right)$$

we want to minimise that function =>
compute gradients for $W_0, W_1, W_2, \vec{b}_0, \vec{b}_1, \vec{b}_2$
and go into opposite direction =>
gradient descend

Learning / Cost function, Gradient descent

$$\text{cost}(W_0, W_1, W_2, \vec{b_0}, \vec{b_1}, \vec{b_2}) = \frac{1}{M}\sum_{m}^{M} \text{loss}\left(\text{net}(\vec{v}_m, \ldots), \vec{t_m}\right)$$

we want to compute the gradients / derivatives of a complicated function with many parameters (can be up to gigabytes of floats).

Differentiation methods:

- symbolic
- numeric
- automatic

$f(x, y) = (x + y) * (y + 3)$

$x = 1, y = 2$

$f(x, y) = ?$

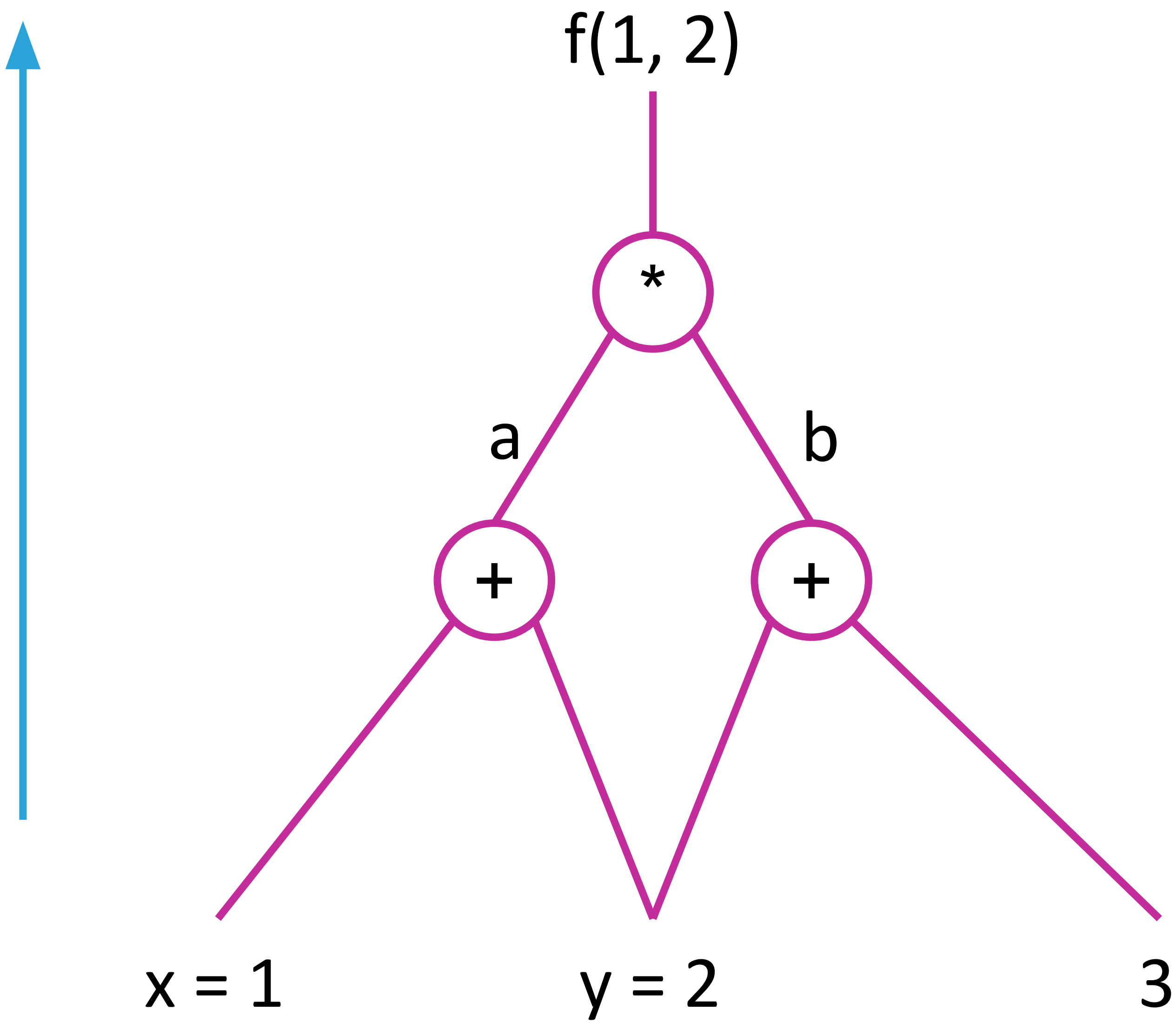
$df / dx = ?$

$df / dy = ?$

Automatic Differentiation / A complicated computation

$f(x, y) = (x + y) * (y + 3)$
$x = 1, y = 2$
$a = x + y$
$b = y + 3$
$f(x, y) = a * b$

$f(x, y) = ?$
$df / dx = ?$
$df / dy = ?$



Automatic Differentiation / A complicated computation

$f(x, y) = (x + y) * (y + 3)$

$x = 1, y = 2$

$a = x + y$

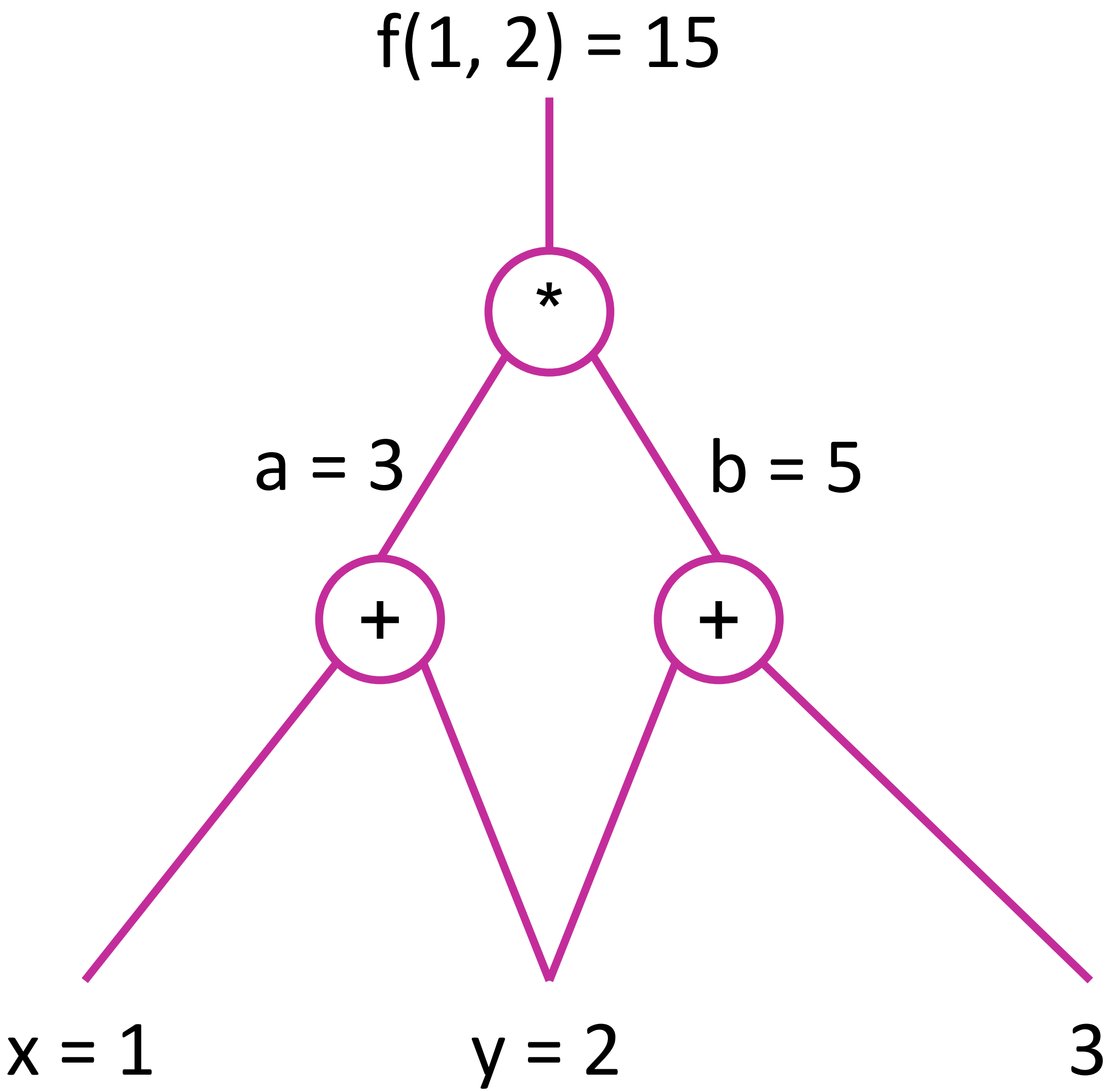
$b = y + 3$

$f(x, y) = a * b$

$f(x, y) = 15$

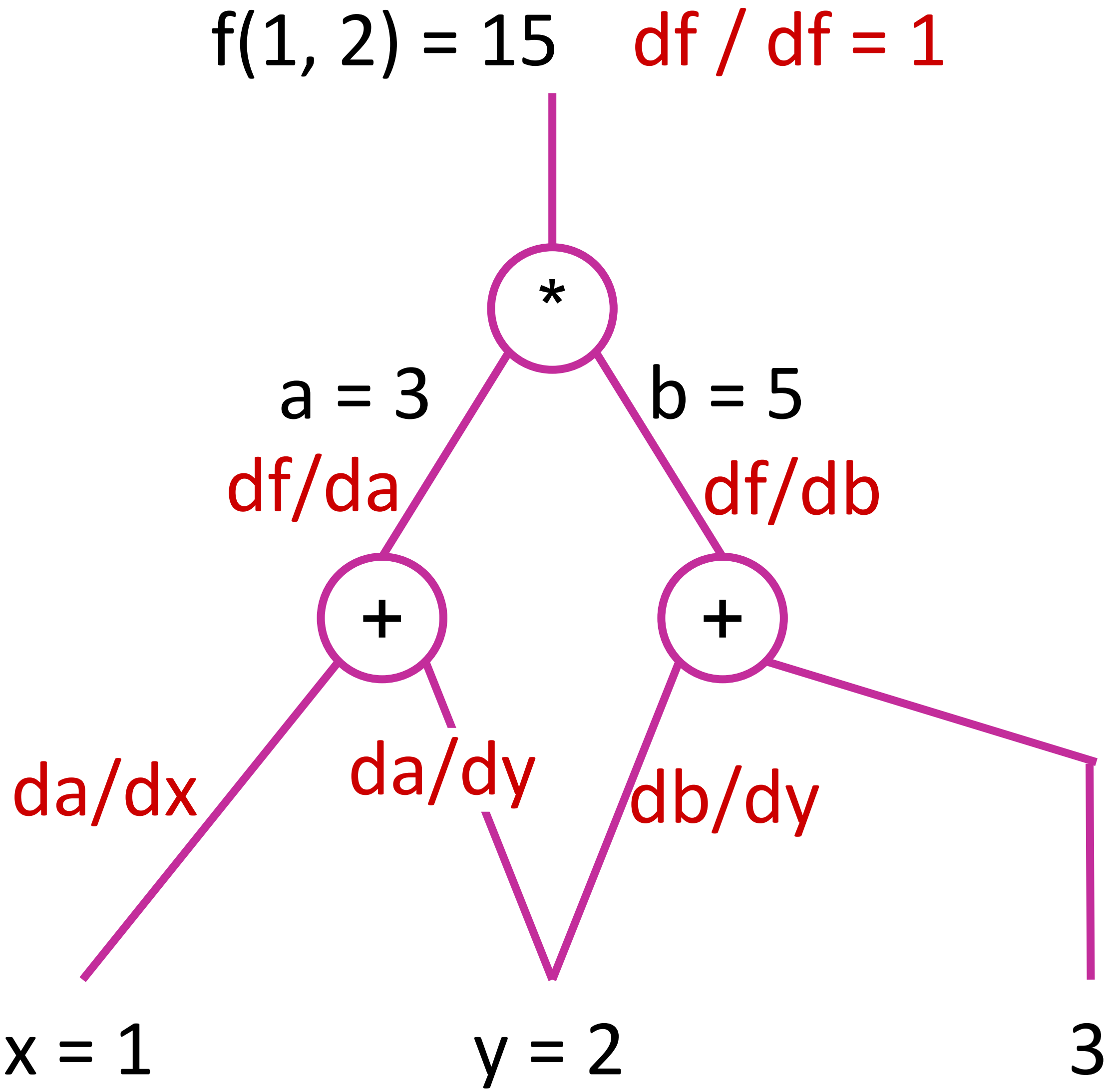
$df / dx = ?$

$df / dy = ?$



f(x, y) = (x + y) * (y + 3)
x = 1, y = 2
a = x + y
b = y + 3
f(x, y) = a * b

f(x, y) = 15
df / dx = ?
df / dy = ?



$f(x, y) = (x + y) * (y + 3)$

$x = 1, y = 2$

$a = x + y$

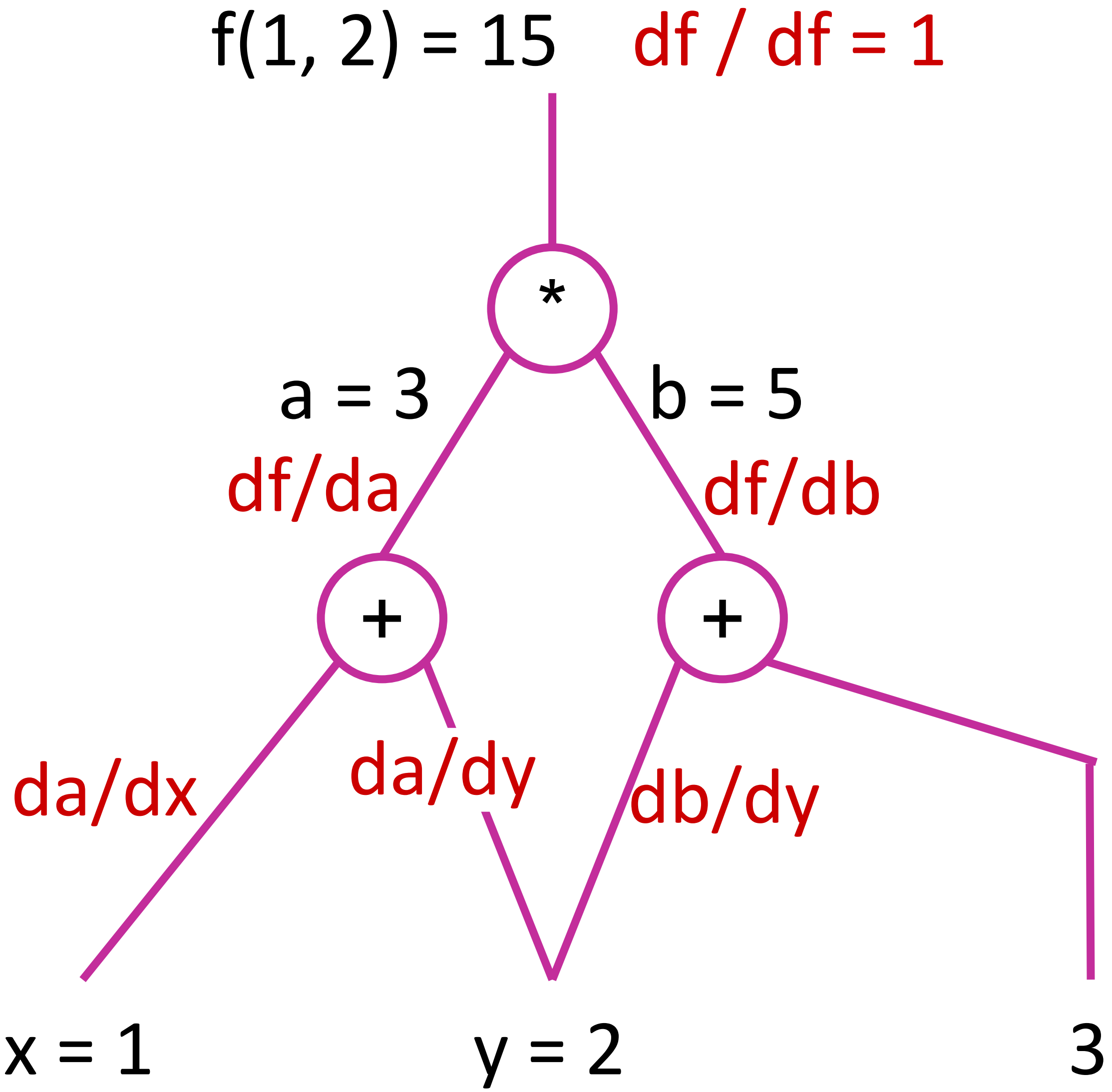
$b = y + 3$

$f(x, y) = a * b$

$f(x, y) = 15$

$$\frac{df}{dx} = \frac{df}{da}\frac{da}{dx}$$

$$\frac{df}{dy} = \frac{df}{da}\frac{da}{dy} + \frac{df}{db}\frac{db}{dy}$$



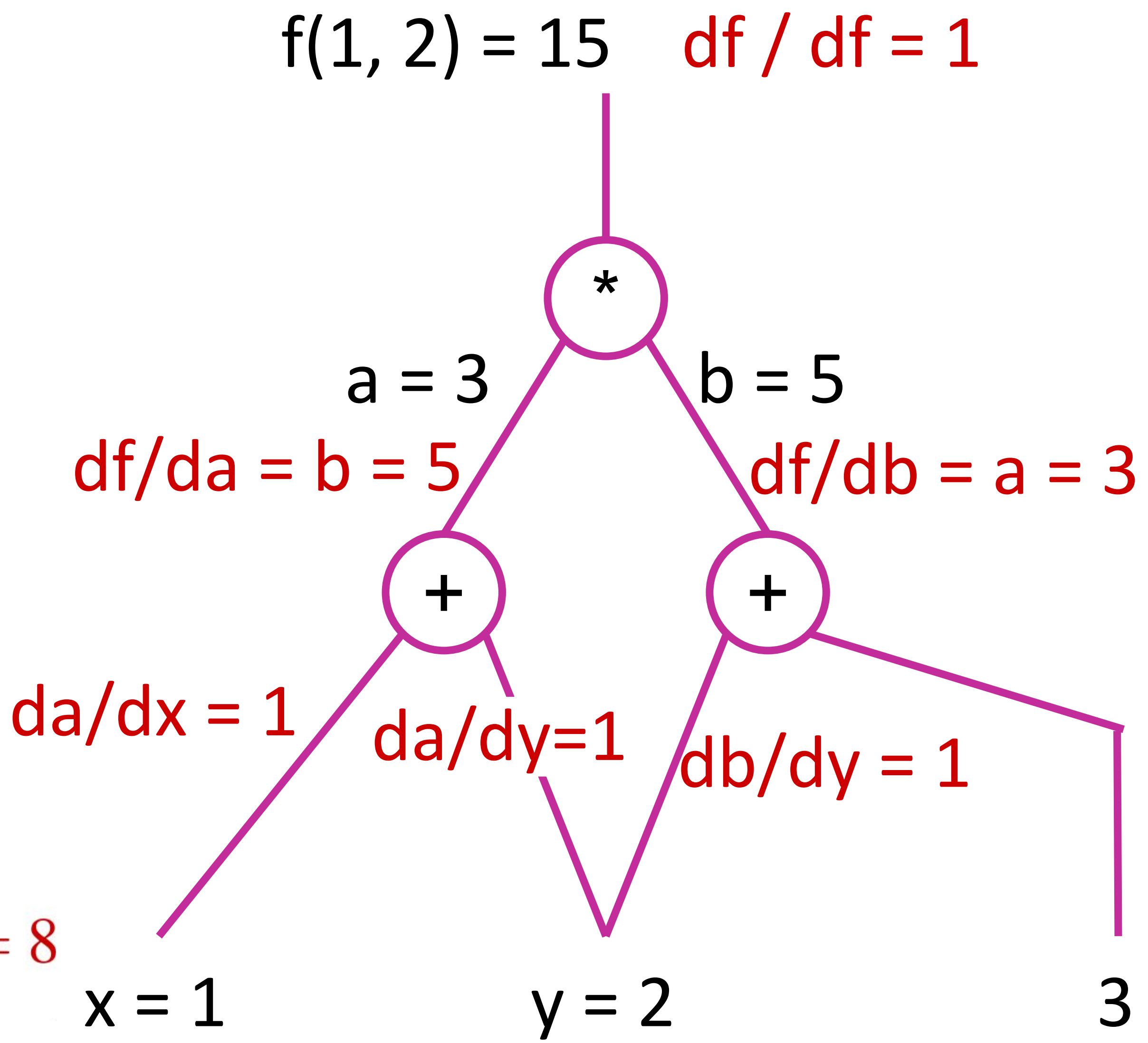
Automatic Differentiation / A complicated computation

f(x, y) = (x + y) * (y + 3)
x = 1, y = 2
a = x + y
b = y + 3
f(x, y) = a * b

f(x, y) = 15

$$\frac{df}{dx} = \frac{df}{da}\frac{da}{dx} = 5 * 1$$

$$\frac{df}{dy} = \frac{df}{da}\frac{da}{dy} + \frac{df}{db}\frac{db}{dy} = 5 * 1 + 3 * 1 = 8$$



Automatic Differentiation / A complicated computation

f(x, y) = (x + y) * (y + 3)
x = 1, y = 2
a = x + y
b = y + 3
f(x, y) = a * b

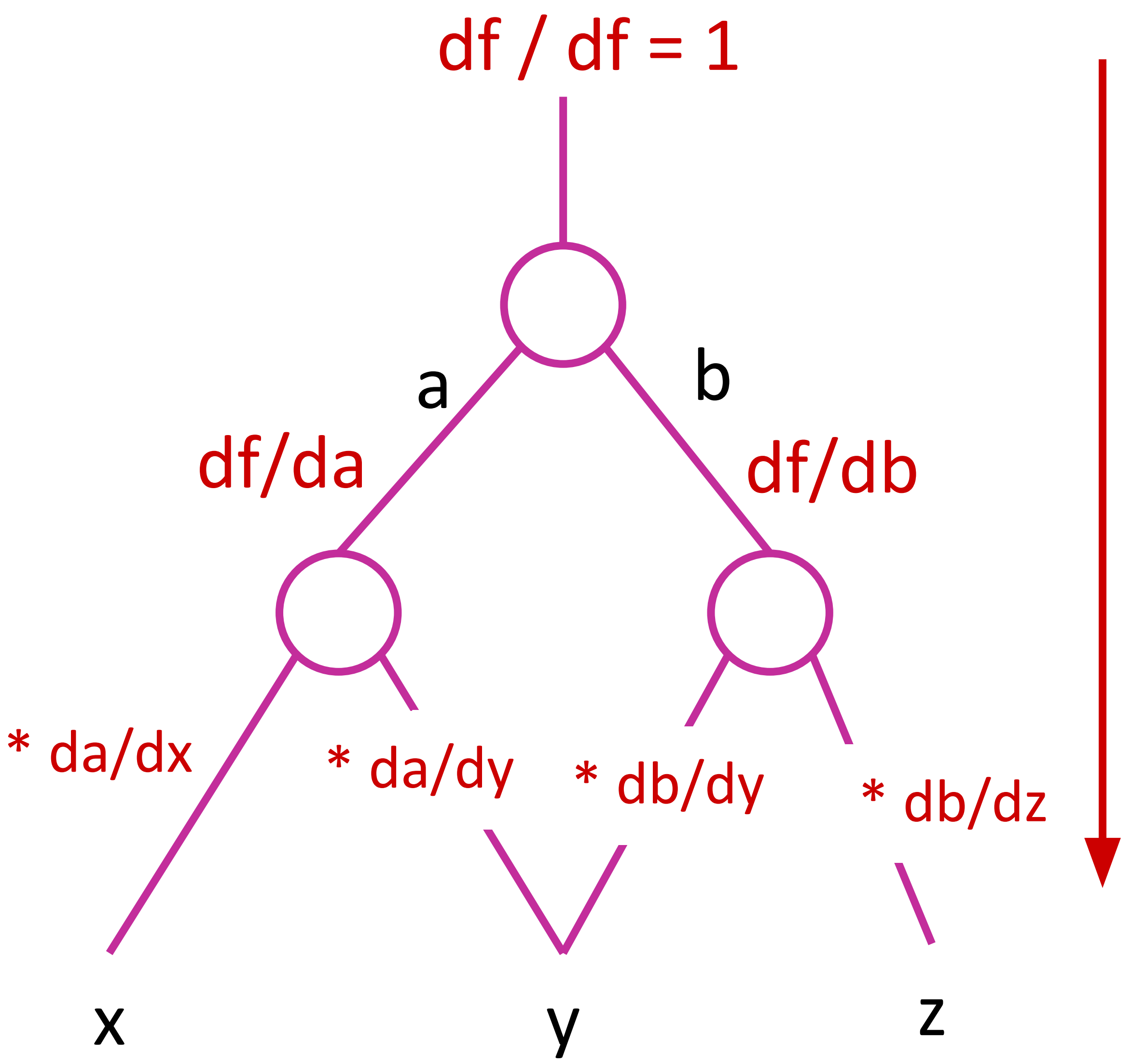
f(x, y) = 15

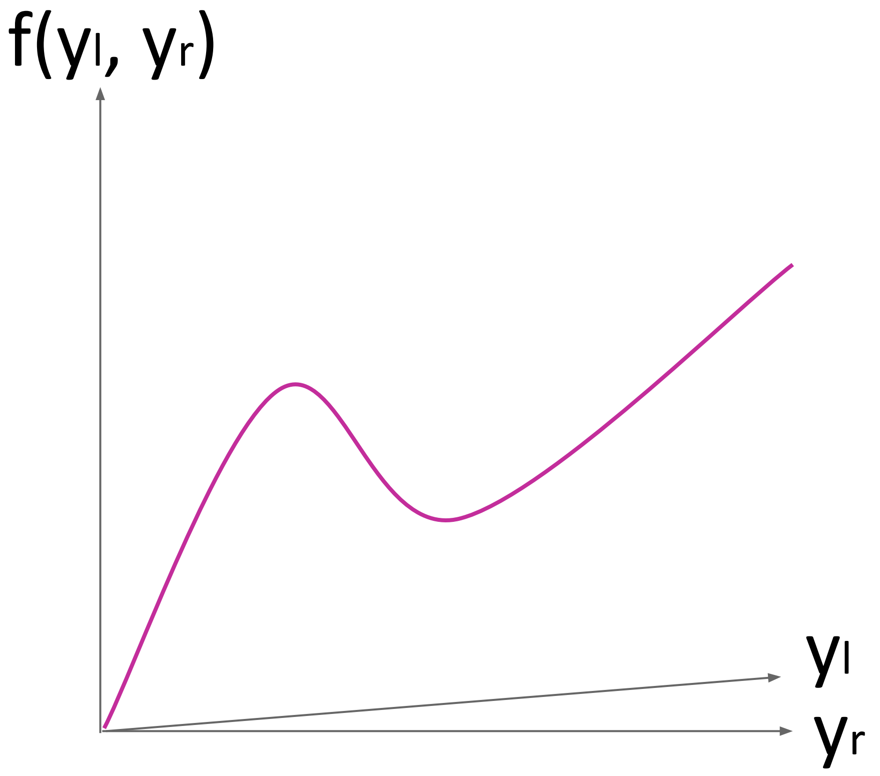
$$\frac{df}{dx} = \frac{df}{da}\frac{da}{dx} = 5 * 1$$

$$\frac{df}{dy} = \frac{df}{da}\frac{da}{dy} + \frac{df}{db}\frac{db}{dy} = 5 * 1 + 3 * 1 = 8$$

check:
f(x,y) = xy + 3x + y^2 + 3y
df/dx = y + 3 = 5
df/dy = x + 2y + 3 = 1 + 4 + 3 = 8

AD / Why are we summing in the leaves





AD / A complicated implementation

Quick look at a 140 line python implementation

A full DNN implementation (C++ demo)

Longer look at the C++ DNN implementation

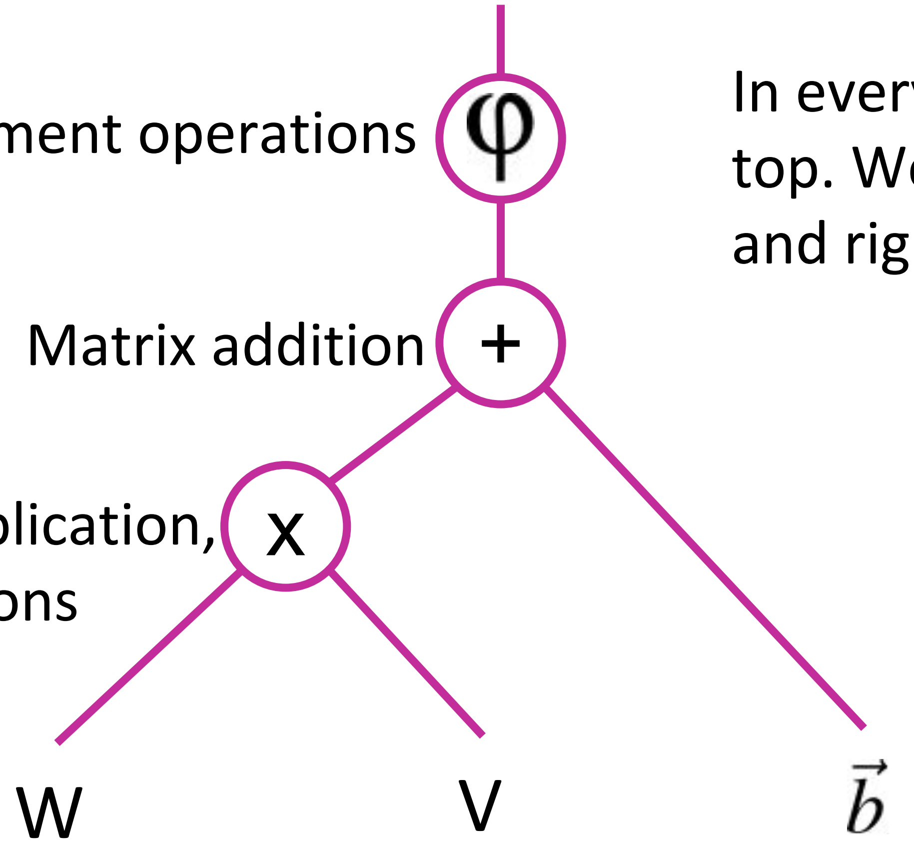
AD with matrices

$$\operatorname{net}(\vec{v}, W_0, W_1, W_2, b_0, b_1, b_2) = \varphi_{\text{out}}\Big(W_2\varphi\big(W_1\varphi(W_0\vec{v} + b_0) + b_1\big) + b_2\Big)$$

per element operations

Matrix addition

Matrix multiplication,
new dimensions



In every node there is a gradient coming from the top. We need to compute the derivative wrt left and right, multiply and send it on.

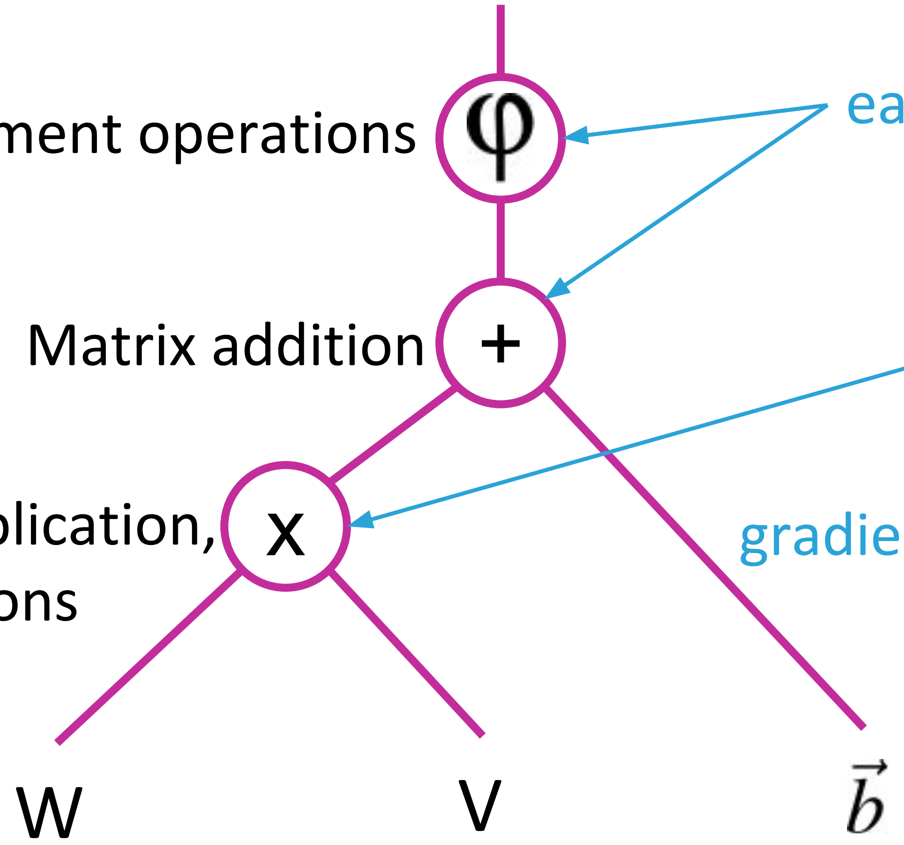
AD with matrices

$$\mathrm{net}(\vec{v}, W_0, W_1, W_2, b_0, b_1, b_2) = \varphi_{\mathrm{out}}\Big(W_2\varphi\big(W_1\varphi(W_0\vec{v} + b_0) + b_1\big) + b_2\Big)$$

per element operations

Matrix addition

Matrix multiplication, new dimensions



easy

W x V = O

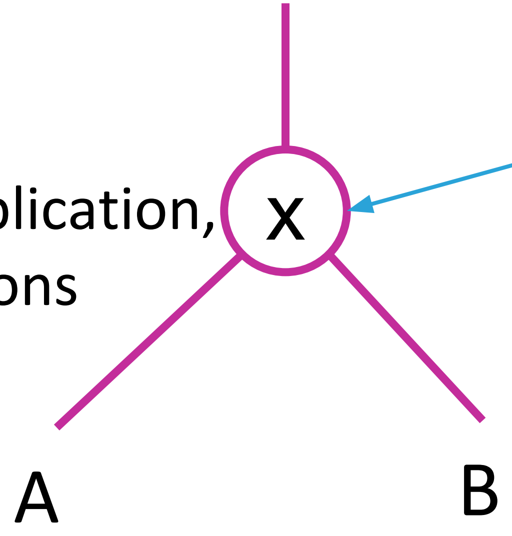
(m*n) x (n*o) = (m*o)

gradient going to the left, right, coming from top

Problems during implementation

AD with matrices

Matrix multiplication,
new dimensions



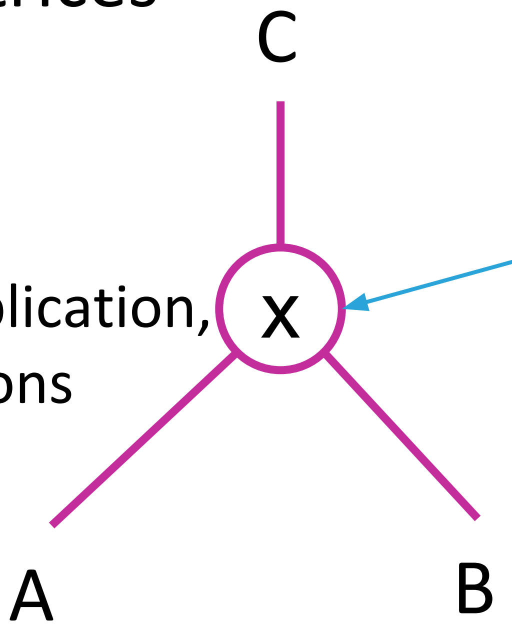
A x B = C

(m*n) x (n*o) = (m*o)

gradient going to the left, right, coming from top

Problems during implementation

AD with matrices



Matrix multiplication, new dimensions

A x B = C

(m*n) x (n*o) = (m*o)

gradient going to the left, right, coming from top

$$\frac{d\Phi}{dC}\frac{dC}{dA} = \frac{d\Phi}{dC}B^T$$

$$\frac{d\Phi}{dC}\frac{dC}{dB} = A^T\frac{d\Phi}{dC}$$

do you see what's wrong with that?

$$f(x,y) = \frac{e^x}{e^x + e^y}$$

Problems during implementation

- AD with matrices
- Numerical stability
- Vanishing gradients
- Exploding gradients
- Dying neurons
- How to initialise the weights and biases

Summary and questions

- DNNs are just a bunch of matrices and transfer functions in the right order
- They use gradient descent for learning, some-thing a school kid could do
- However, there are some challenges during implementation
- github.com/cg-tuwien/deep_learning_demo

Questions?