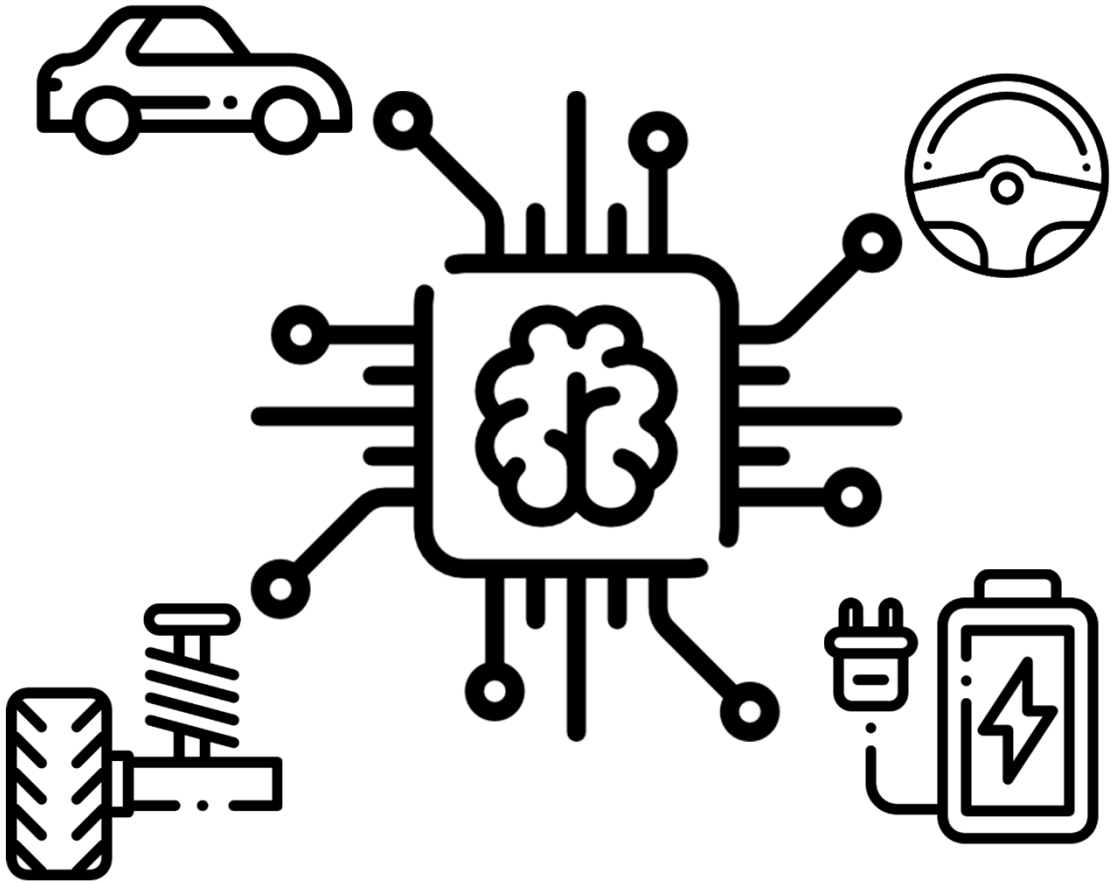
Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Artificial Intelligence in Automotive Technology

Johannes Betz / Prof. Dr.-Ing. Markus Lienkamp / Prof. Dr.-Ing. Boris Lohmann

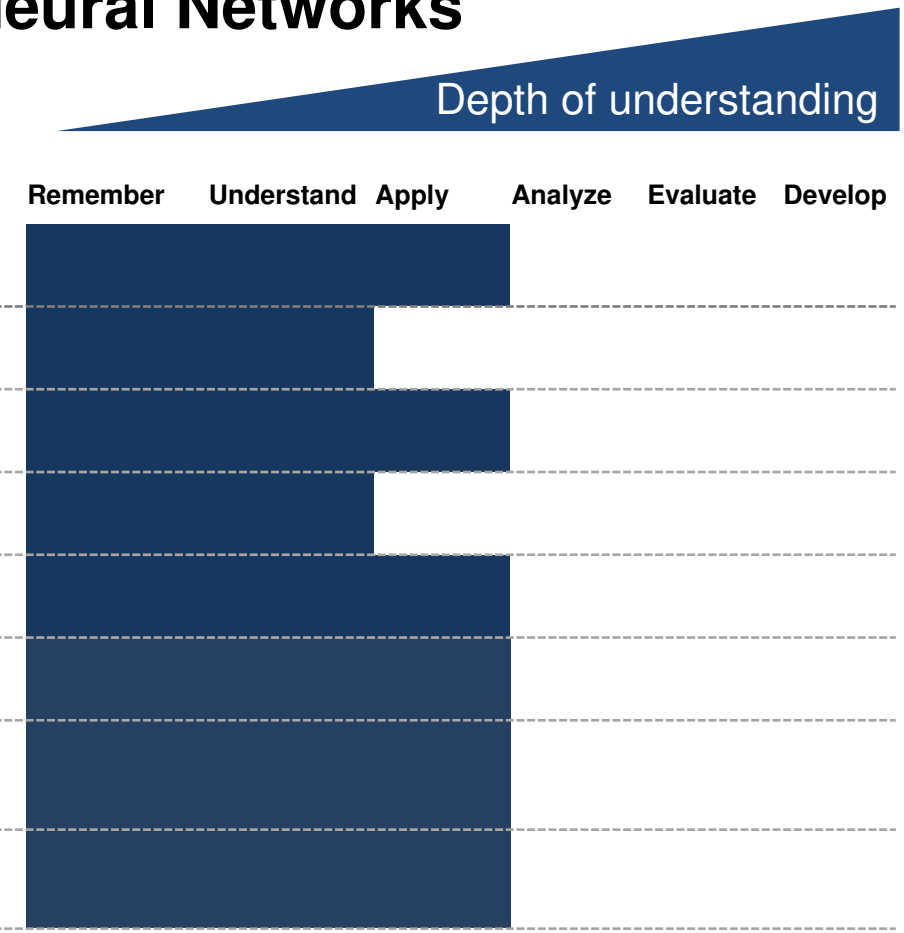


Lecture Overview

Overall Introduction for the Lecture 18.10.2018 – Betz Johannes	**6 Pathfinding: From British Museum to A*** 29.11.2018 – Lennart Adenaw	**11 Reinforcement Learning** 17.01.2019 – Christian Dengler
1 Introduction: Artificial Intelligence 18.10.2018 – Betz Johannes	**P6:** 29.11.2018 – Lennart Adenaw	**P11** 17.01.2019 – Christian Dengler
P1: 18.10.2018 – Betz Johannes	**7 Introduction: Artificial Neural Networks** 06.12.2018 – Lennart Adenaw	**12 AI-Development** 24.01.2019 – Johannes Betz
2 Perception 25.10.2018 – Betz Johannes	**P7** 06.12.2018 – Lennart Adenaw	**P12** 24.01.2019 – Johannes Betz
P2: 25.10.2018 – Betz Johannes	**8 Deep Neural Networks** 13.12.2018 – Jean-Michael Georg	**13 Free Discussion** 31.01.2019 – Betz/Adenaw
3 Supervised Learning: Regression 08.11.2018 – Alexander Wischnewski	**P8** 13.12.2018 – Jean-Michael Georg	
P3: 08.11.2018 – Alexander Wischnewski	**9 Convolutional Neural Networks** 20.12.2018 – Jean-Michael Georg	
4 Supervised Learning: Classification 15.11.2018 – Jan-Cedric Mertens	**P9** 20.12.2018 – Jean-Michael Georg	
P4: 15.11.2018 – Jan-Cedric Mertens	**10 Recurrent Neural Networks** 10.01.2019 – Christian Dengler	
5 Unsupervised Learning: Clustering 22.11.2018 – Jan-Cedric Mertens	**P10** 10.01.2019 – Christian Dengler	

Objectives for Lecture 8: Neural Networks

After the lecture you are able to…



… compute local gradients with a computational graph

… understand how backpropagation works in a neural network

… apply backpropagation algorithm to small neural networks to calculate weight changes and local gradients

… understand differences between activation functions and use cases

… compute correct dimensions for weight and biases matrices

… how to program fully connected neural networks

… initialize weights correctly

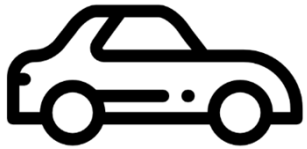
…. train a neural network

Deep Neural Networks

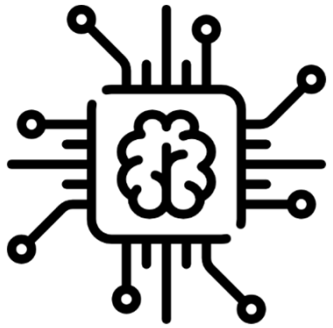
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

1. Chapter: Backpropagation
    - 1.1 Computational Graph
    - 1.2 Single Neuron
    - 1.3 Neural Chain
    - 1.4 Neural Network

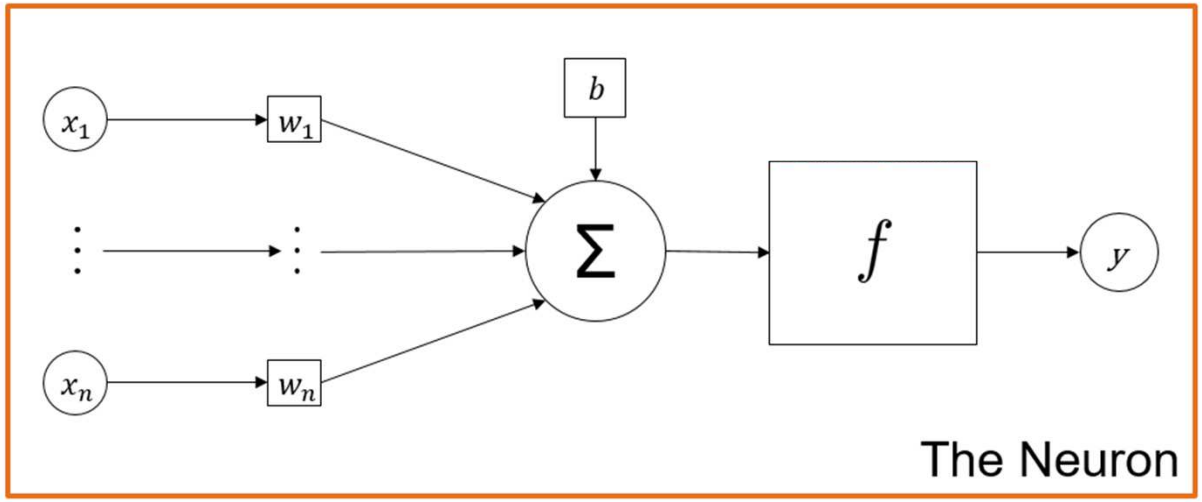


2. Chapter: Neuronal Networks
    - 2.1 Activation Functions
    - 2.2 Fully Connected Layer
    - 2.3 Batches
    - 2.3 Weight Inititalisation



3. Chapter: Overview

Repetition



$L = w^2$

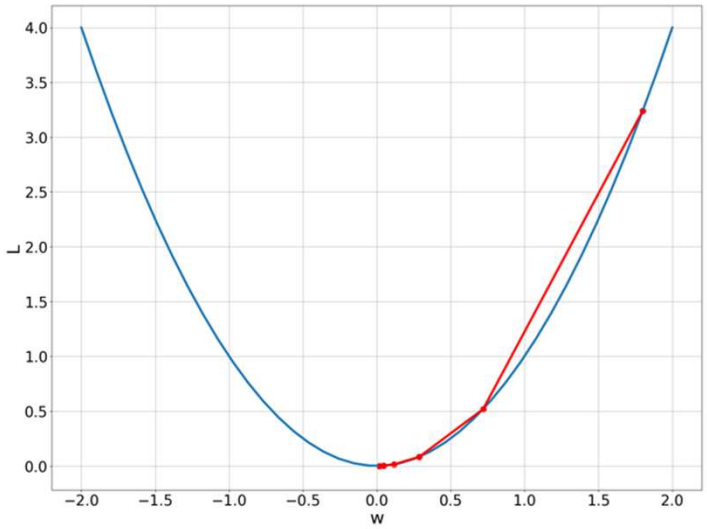
$\nabla L = \frac{dL}{dw} = 2 \cdot w$

$\alpha = 0.3$

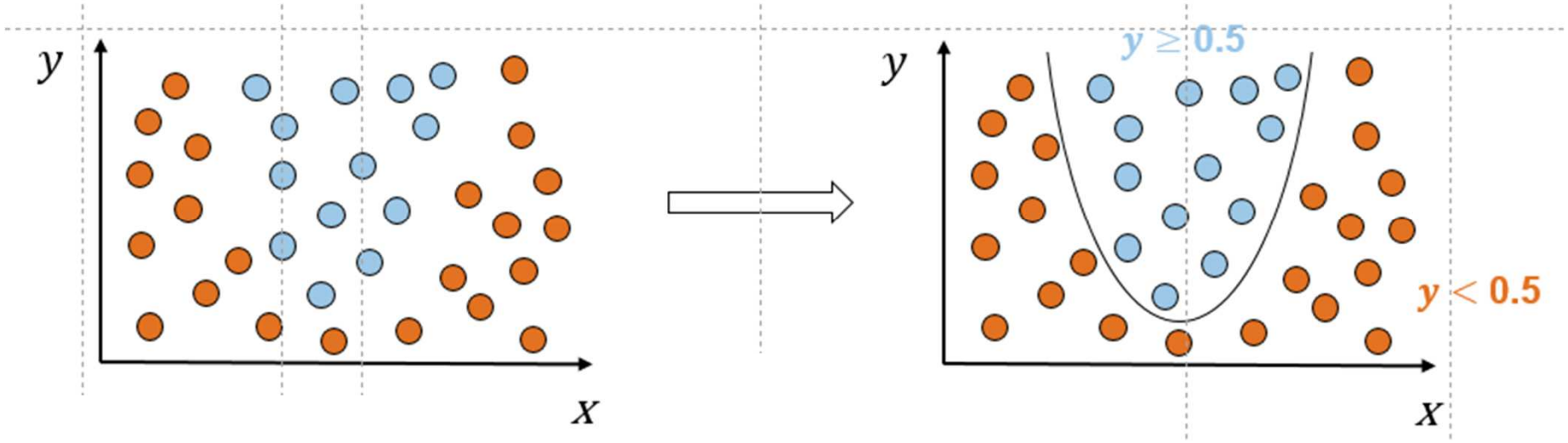
$w_0 = 1.8$

$\epsilon = 0.001$

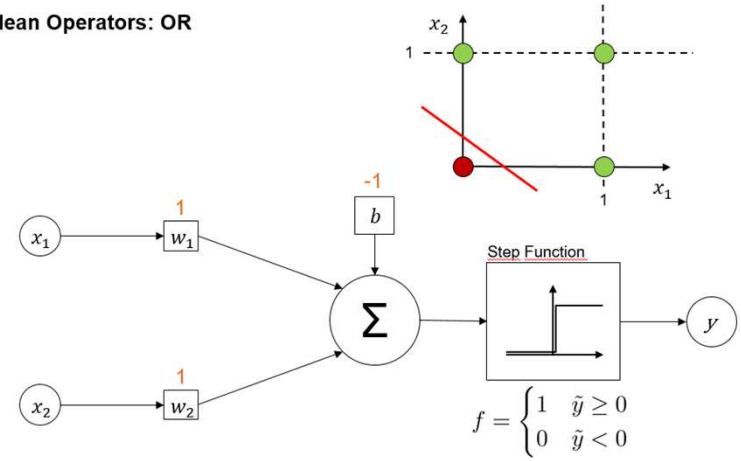
$N = 5$



Repetition

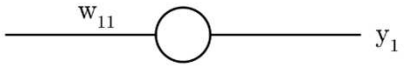


Boolean Operators: OR

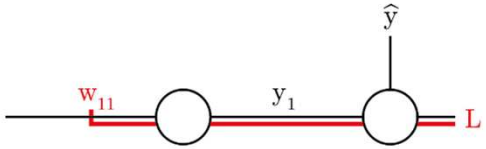


Last Week:

Single Neuron



1D Gradient Descent



$L = w^2$

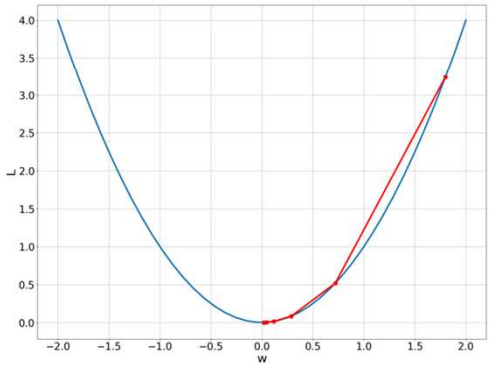
$\nabla L = \frac{dL}{dw} = 2 \cdot w$

$\alpha = 0.3$

$w_0 = 1.8$

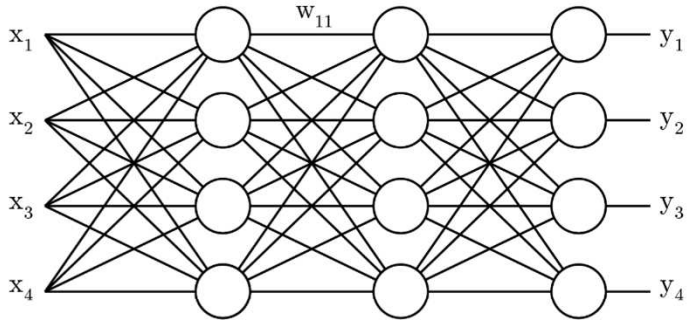
$\epsilon = 0.001$

$N = 5$

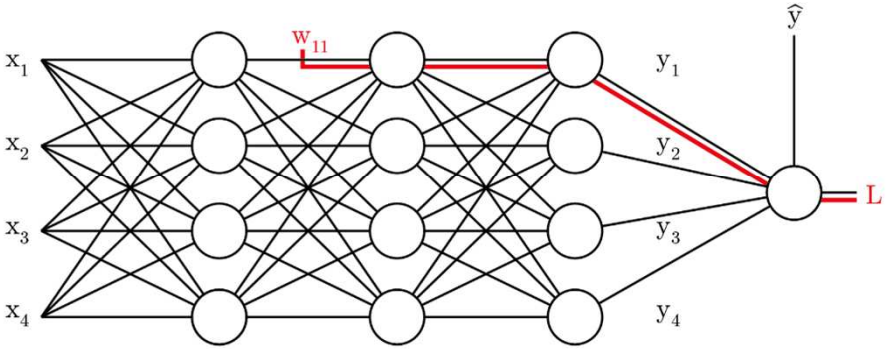


Today:

Neural Network



Backpropagtion

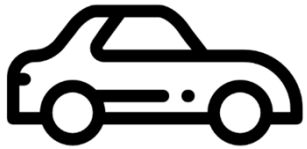


Deep Neural Networks

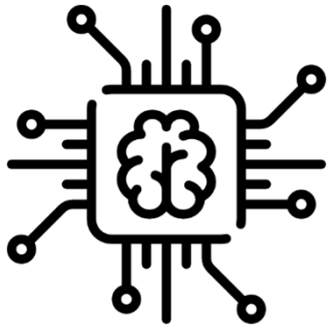
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

1. Chapter: Backpropagation
   - 1.1 Computational Graph
   - 1.2 Single Neuron
   - 1.3 Neural Chain
   - 1.4 Neural Network



2. Chapter: Neuronal Networks
   - 2.1 Activation Functions
   - 2.2 Fully Connected Layer
   - 2.3 Batches
   - 2.3 Weight Inititalisation



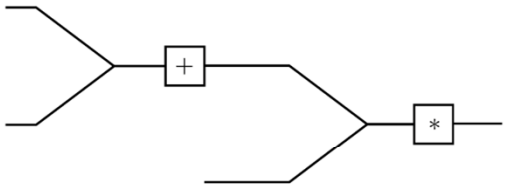
3. Chapter: Overview

Computational Graph

We need: $$\Delta w = -\alpha \frac{\partial L}{\partial w}$$

Options:
1. Analytical
2. Difference quotient
3. Computational graph

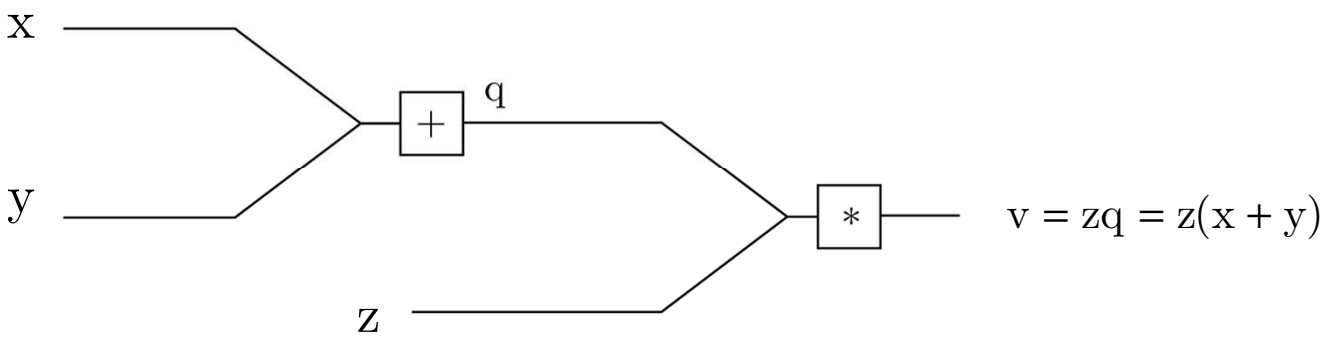
$$\lim_{h \to 0} \frac{L(w+h) - L(w)}{h}$$



$\Rightarrow$ Chain of derivitives $\Rightarrow$ Backpropagation

Computational Graph

Simple example

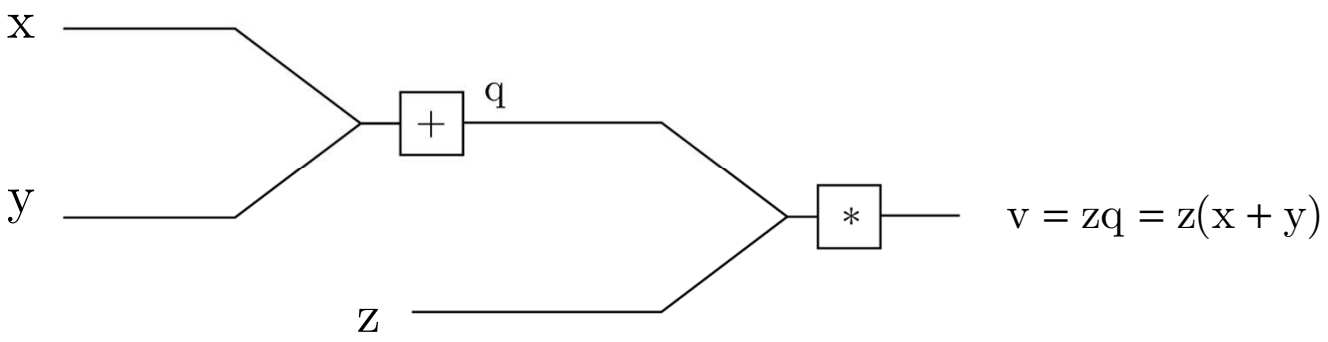


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} \qquad \frac{\partial v}{\partial q} \qquad \frac{\partial v}{\partial x} \qquad \frac{\partial v}{\partial y}$$

Computational Graph

Simple example

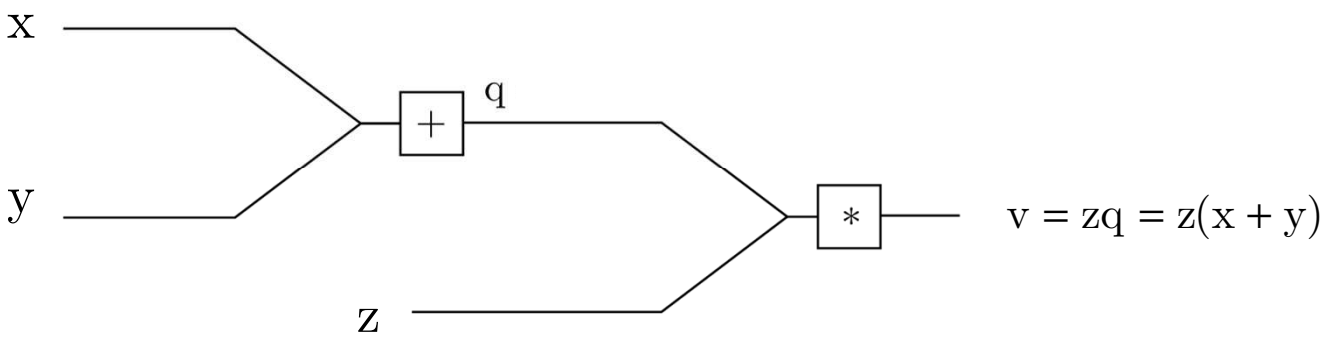


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} \qquad \frac{\partial v}{\partial x} \qquad \frac{\partial v}{\partial y}$$

Computational Graph

Simple example

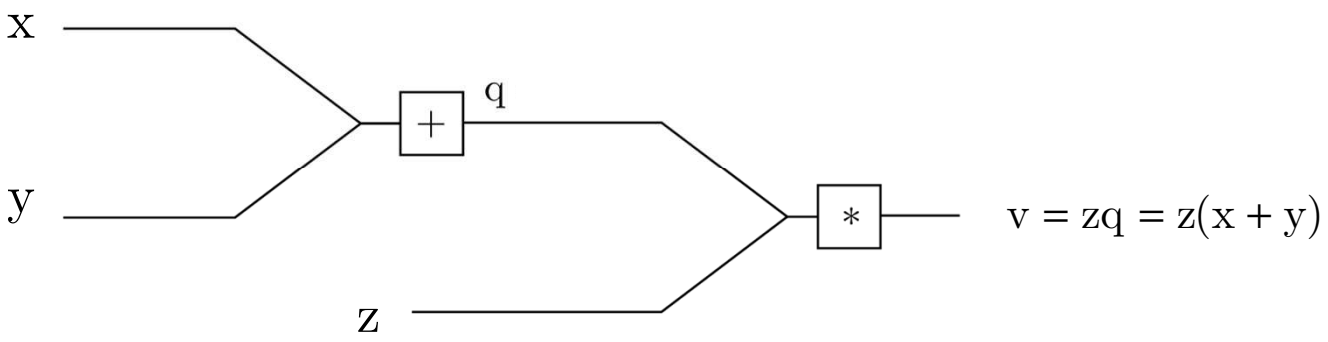


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} \qquad \frac{\partial v}{\partial y}$$

Computational Graph

Simple example

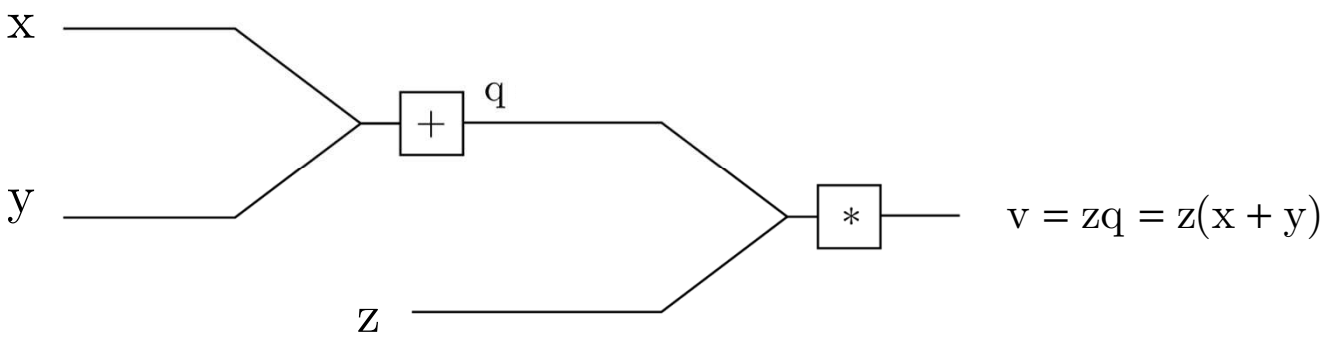


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 \qquad \frac{\partial v}{\partial y}$$

Computational Graph

Simple example

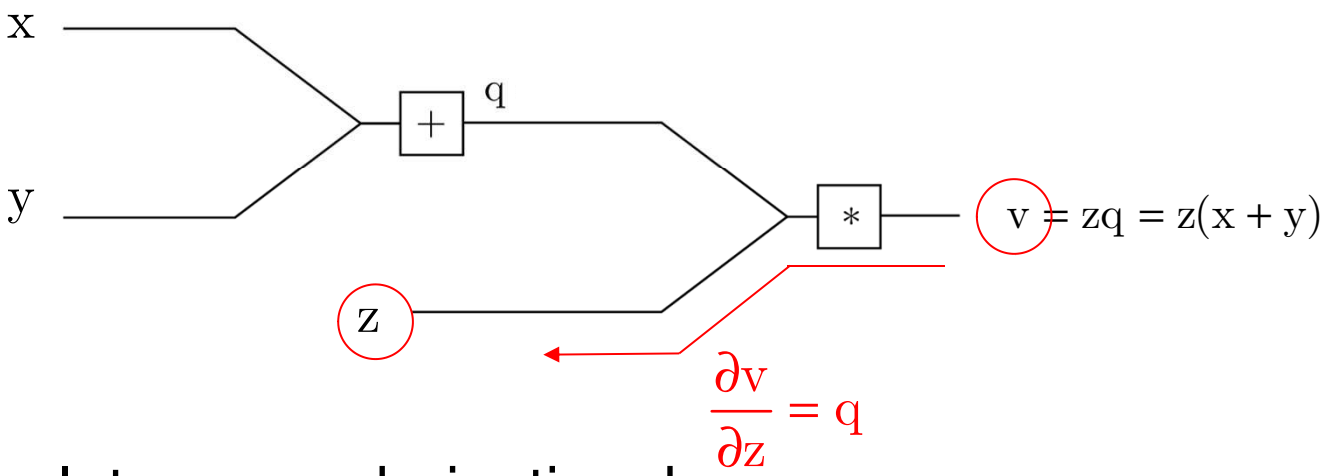


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 \qquad \frac{\partial v}{\partial y} = \frac{\partial v}{\partial q}\frac{\partial v}{\partial y} = z \cdot 1$$

Computational Graph

Simple example

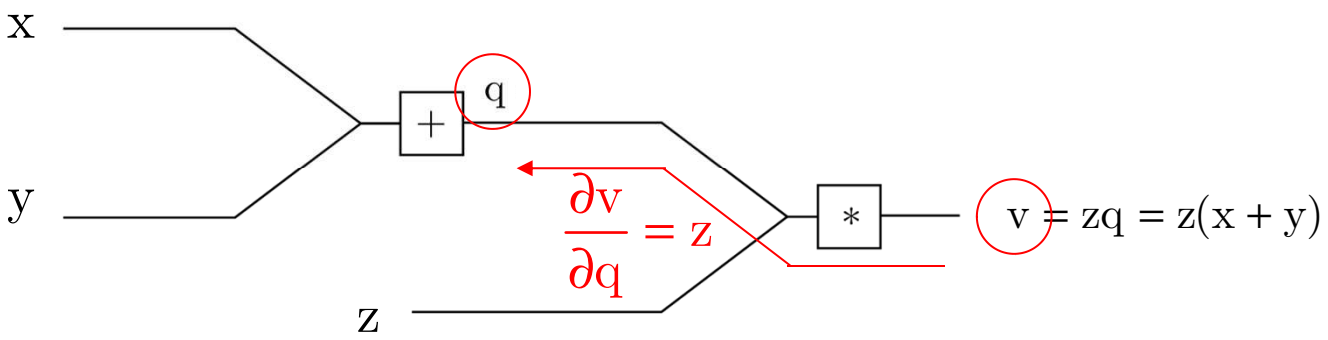


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 \qquad \frac{\partial v}{\partial y} = \frac{\partial v}{\partial q}\frac{\partial v}{\partial y} = z \cdot 1$$

Computational Graph

Simple example

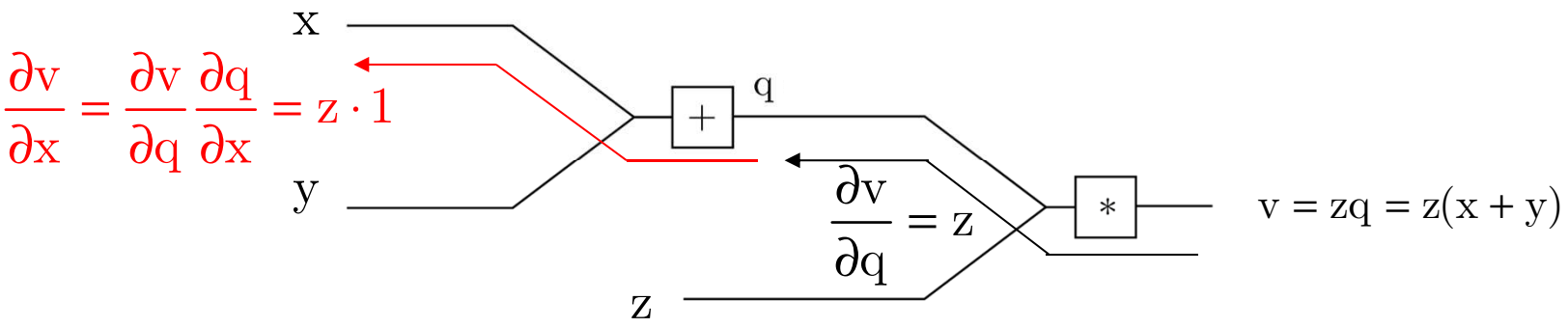


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 \qquad \frac{\partial v}{\partial y} = \frac{\partial v}{\partial q}\frac{\partial v}{\partial y} = z \cdot 1$$

Computational Graph

Simple example

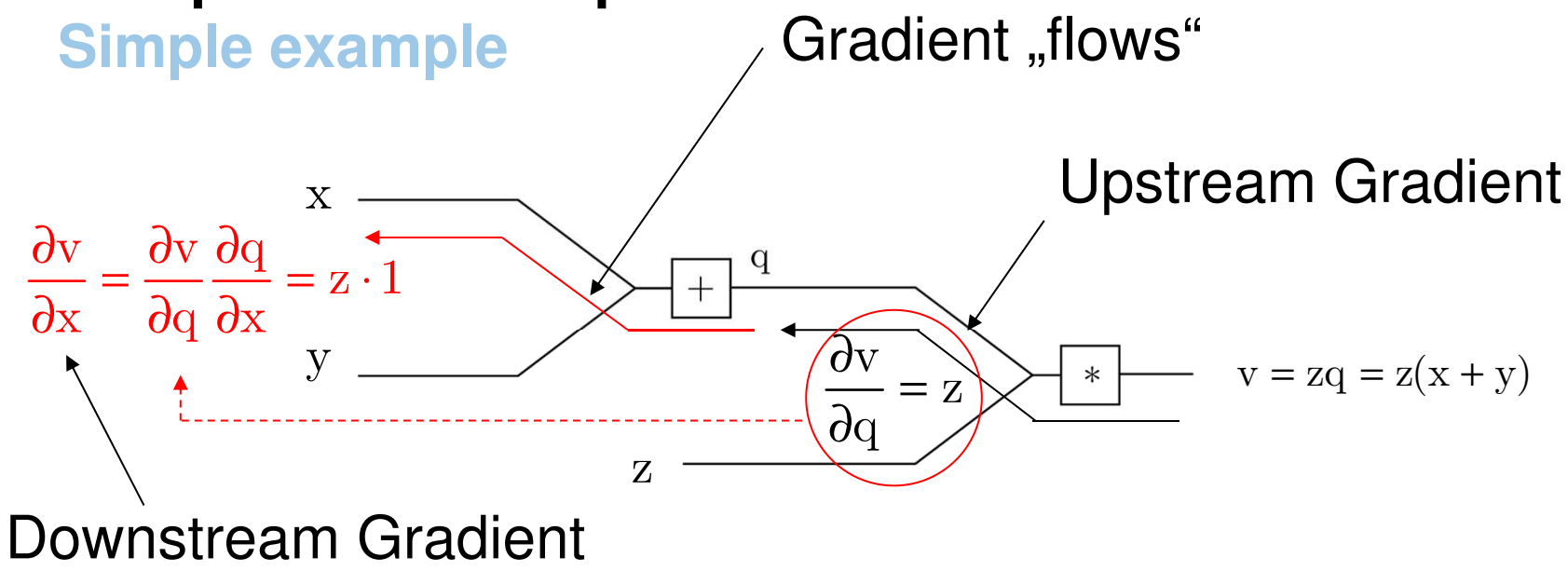


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 \qquad \frac{\partial v}{\partial y} = \frac{\partial v}{\partial q}\frac{\partial v}{\partial y} = z \cdot 1$$

Computational Graph

Simple example

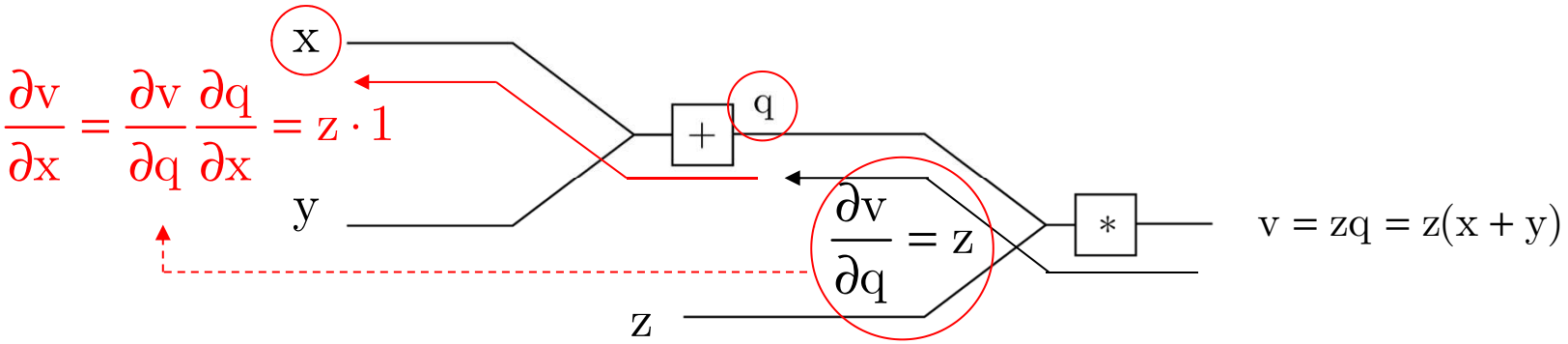


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 \qquad \frac{\partial v}{\partial y} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial y} = z \cdot 1$$

Computational Graph

Simple example

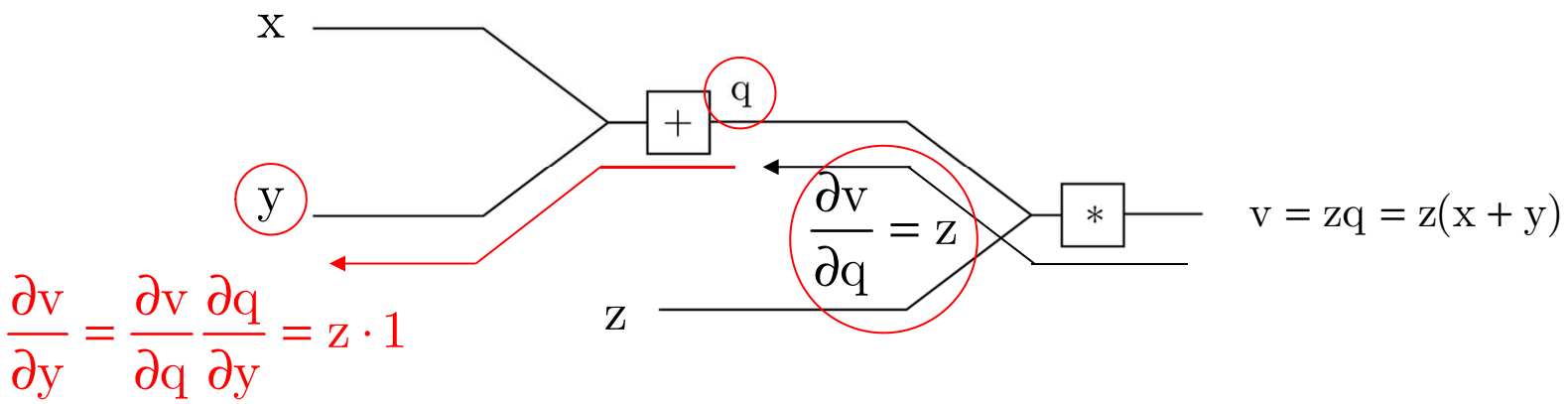


Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 \qquad \frac{\partial v}{\partial y} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial y} = z \cdot 1$$

Computational Graph

Simple example



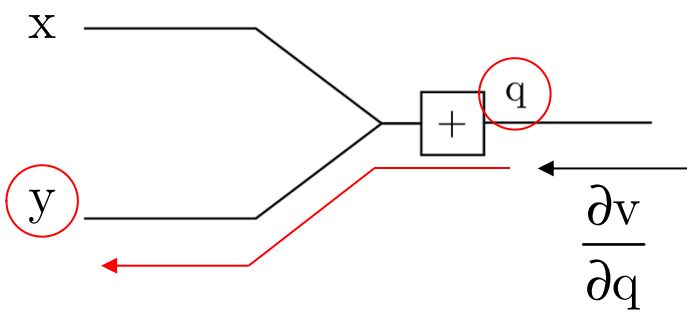
Lets calculate some derivatives!

$$\frac{\partial v}{\partial z} = q \qquad \frac{\partial v}{\partial q} = z \qquad \frac{\partial v}{\partial x} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 \qquad \frac{\partial v}{\partial y} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial y} = z \cdot 1$$

Computational Graph

Standard operations

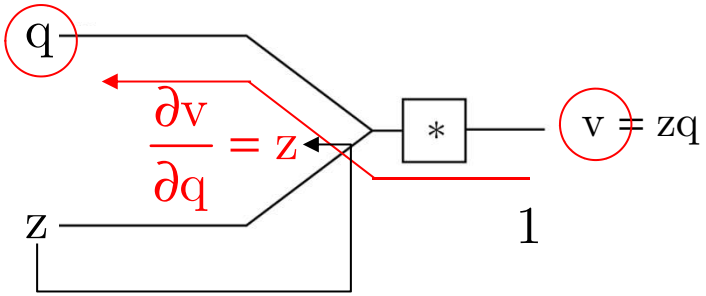
Addition:



$$\frac{\partial v}{\partial y} = \frac{\partial v}{\partial q}\frac{\partial q}{\partial y} = \frac{\partial v}{\partial q}$$

Downstream gradient remains the same

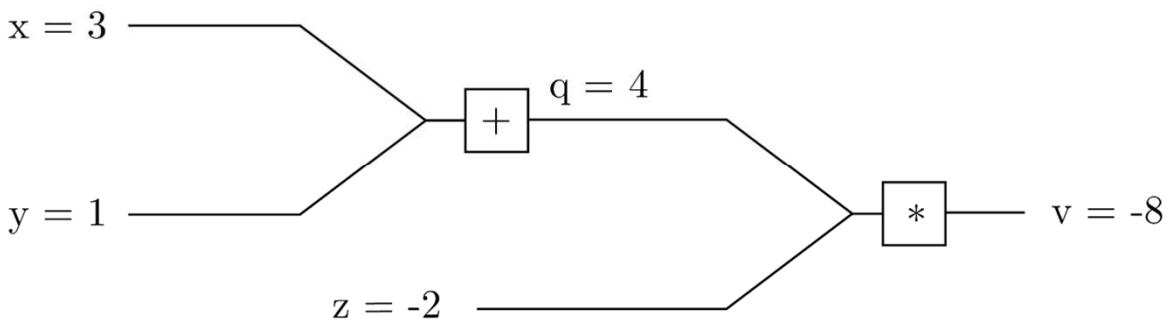
Multiplication



Downstream gradient switches to other factor

Computational Graph

Simple example, with numbers



$$\frac{\partial v}{\partial z} =$$

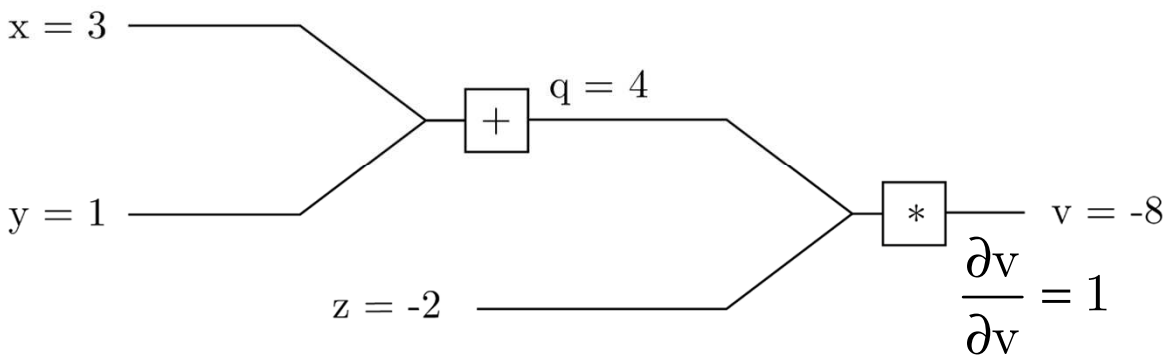
$$\frac{\partial v}{\partial q} =$$

$$\frac{\partial v}{\partial x} =$$

$$\frac{\partial v}{\partial y} =$$

Computational Graph

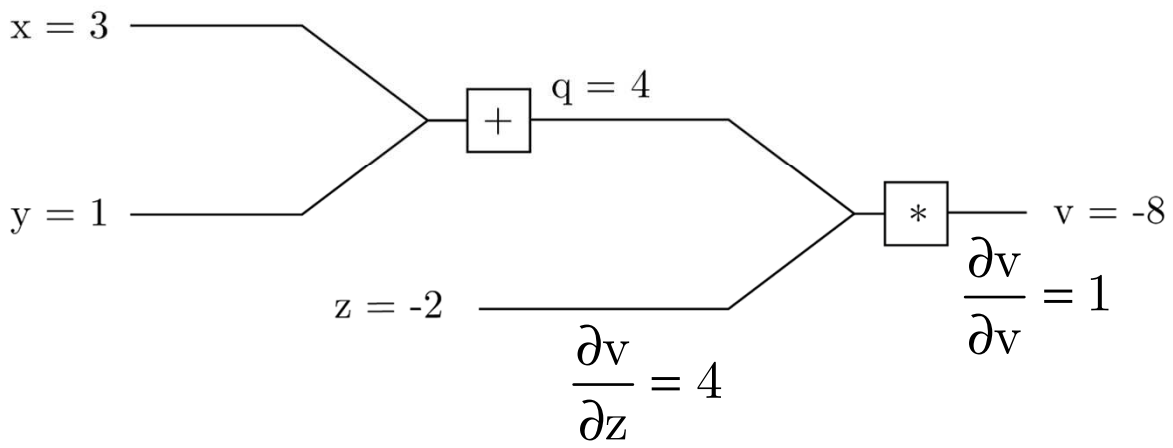
Simple example, with numbers



$\frac{\partial v}{\partial z} =$ $\frac{\partial v}{\partial q} =$ $\frac{\partial v}{\partial x} =$ $\frac{\partial v}{\partial y} =$

Computational Graph

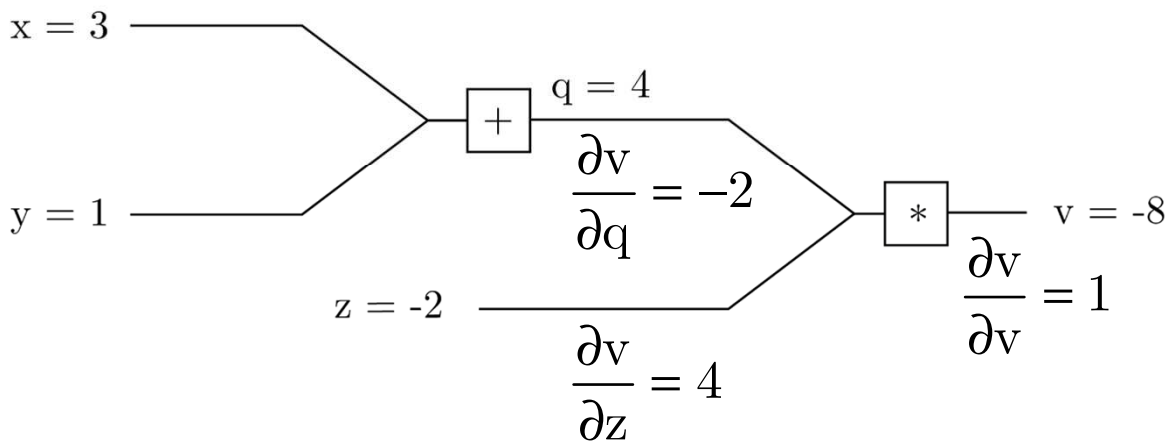
Simple example, with numbers



$$\frac{\partial v}{\partial z} = 1 \cdot 4 = 4 \qquad \frac{\partial v}{\partial q} = \qquad \frac{\partial v}{\partial x} = \qquad \frac{\partial v}{\partial y} =$$

Computational Graph

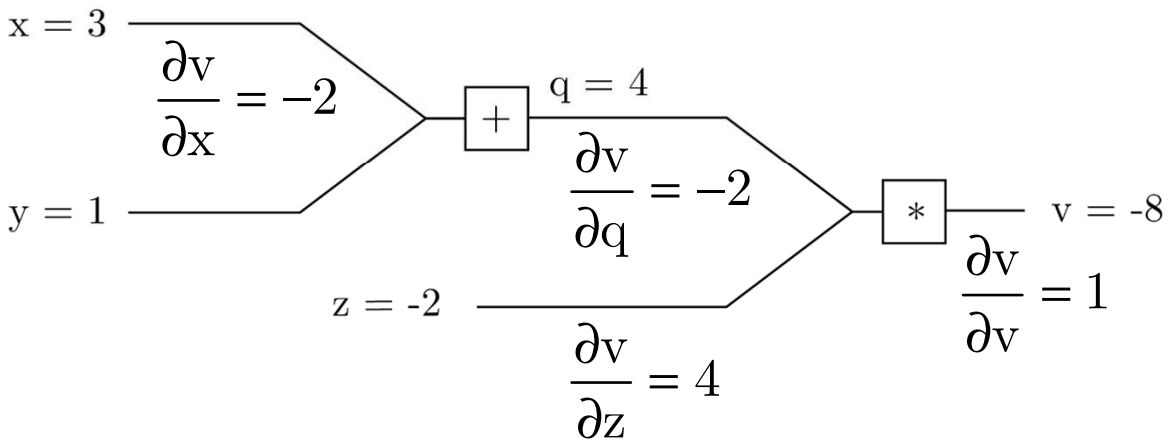
Simple example, with numbers



$$\frac{\partial v}{\partial z} = 1 \cdot 4 = 4 \qquad \frac{\partial v}{\partial q} = 1 \cdot -2 = -2 \qquad \frac{\partial v}{\partial x} = \qquad \frac{\partial v}{\partial y} =$$

Computational Graph

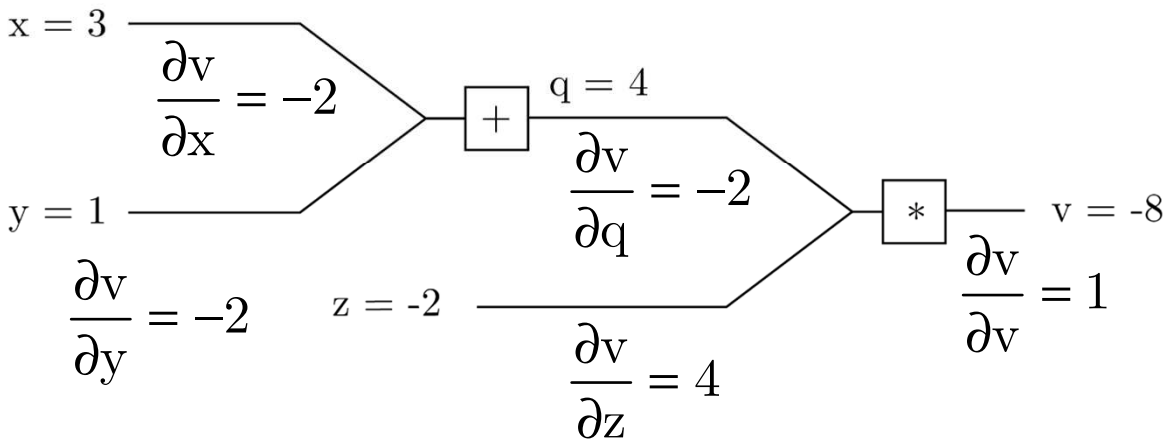
Simple example, with numbers



$$\frac{\partial v}{\partial z} = 1 \cdot 4 = 4 \qquad \frac{\partial v}{\partial q} = 1 \cdot -2 = -2 \qquad \frac{\partial v}{\partial x} = -2 \cdot 1 = -2 \qquad \frac{\partial v}{\partial y} =$$

Computational Graph

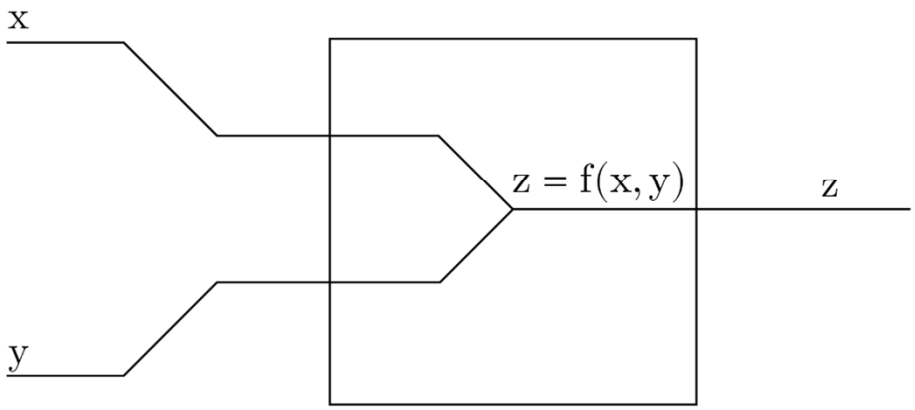
Simple example, with numbers



$$\frac{\partial v}{\partial z} = 1 \cdot 4 = 4 \qquad \frac{\partial v}{\partial q} = 1 \cdot -2 = -2 \qquad \frac{\partial v}{\partial x} = -2 \cdot 1 = -2 \qquad \frac{\partial v}{\partial y} = -2 \cdot 1 = -2$$

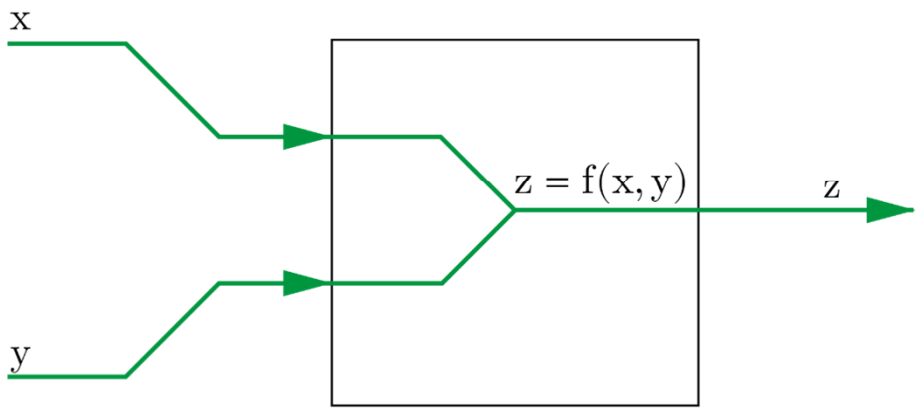
Computational Graph

Abstract function



Computational Graph

Abstract function

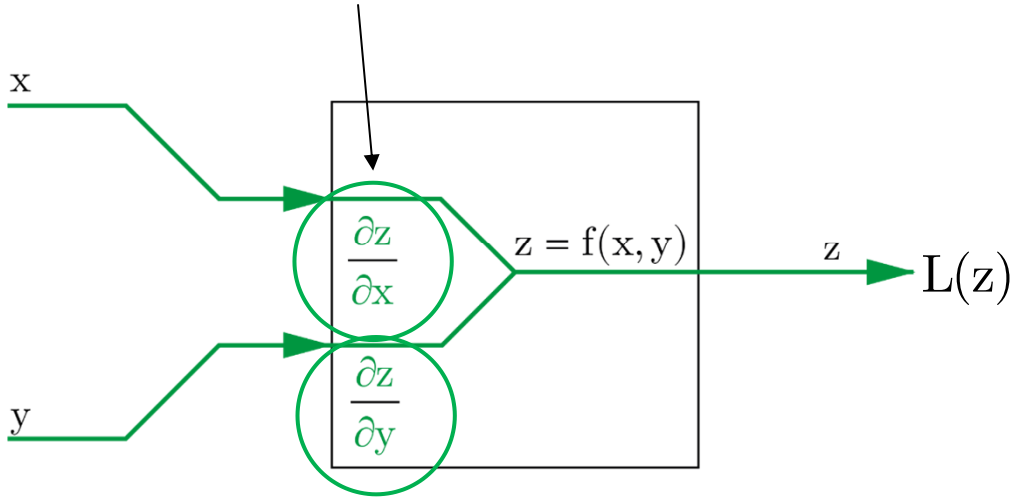


Forward path

Computational Graph

Abstract function

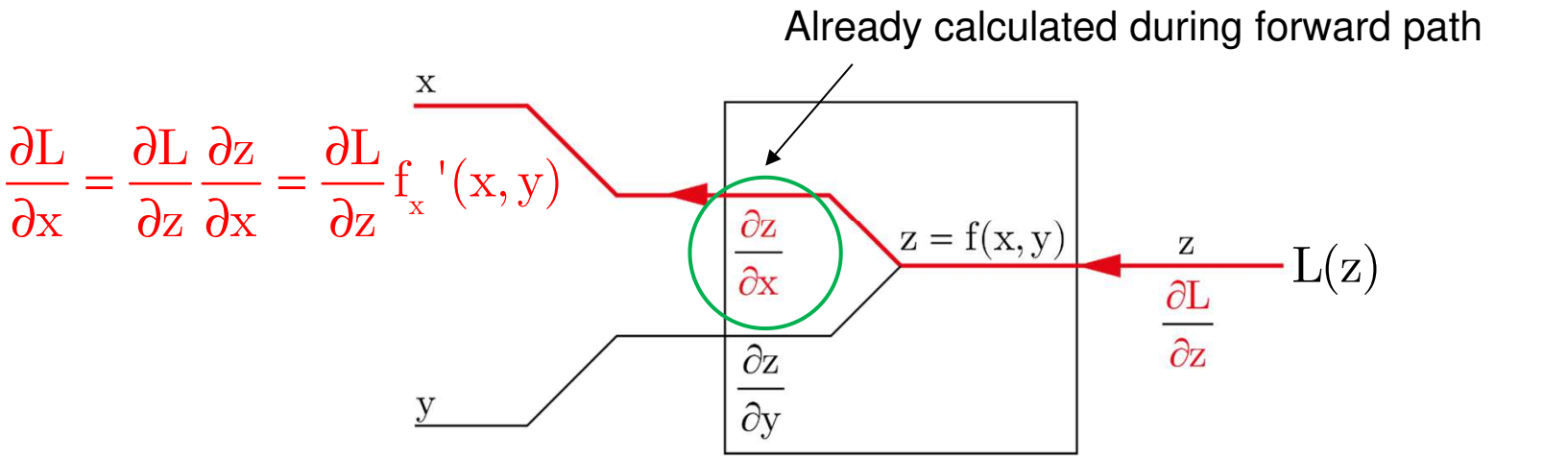
Saving local gradients during forward path



Forward path

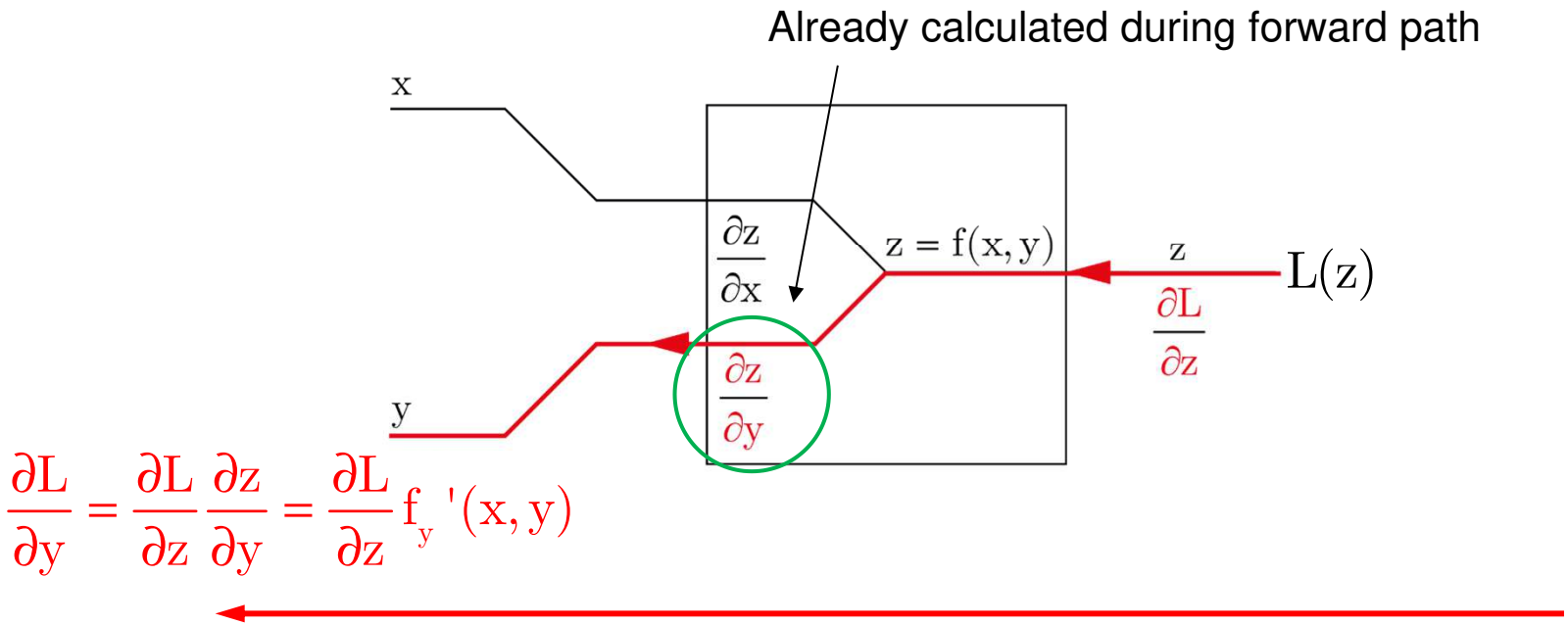
Computational Graph

Abstract function



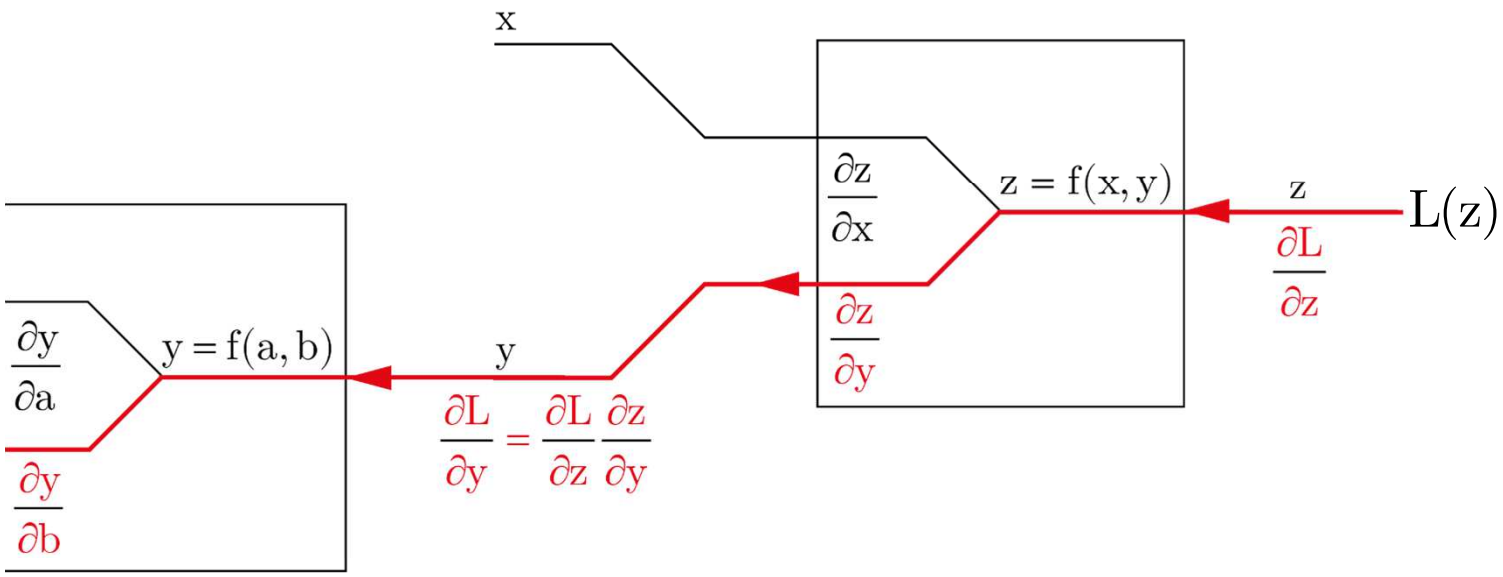
Computational Graph

Abstract function



Computational Graph

Abstract function



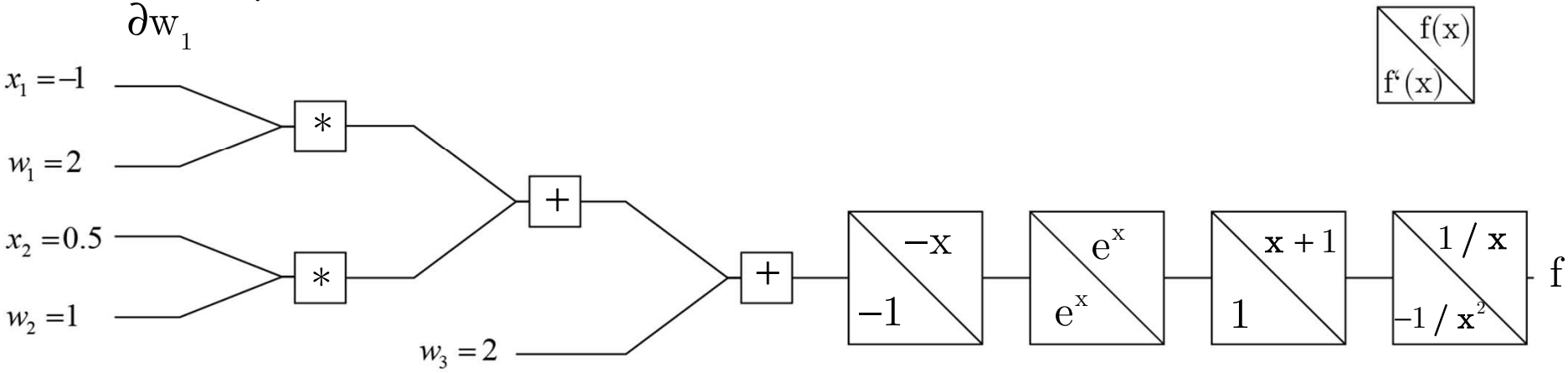
Computational Graph

Complex Example

$$\frac{\partial f}{\partial x_1} = ?$$

$$\frac{\partial f}{\partial w_1} = ?$$

$$f(x_1, x_2) = \frac{1}{1 + e^{-(x_1 w_1 + x_2 w_2 + w_3)}}$$



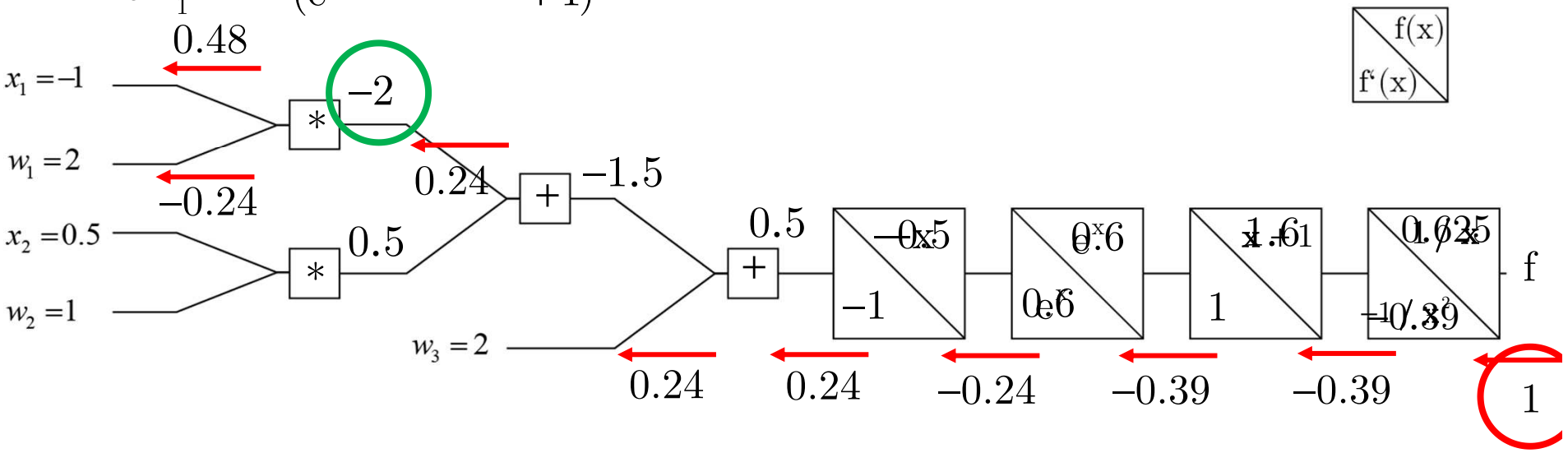
Computational Graph

Complex Example

$$\frac{\partial f}{\partial x_1} = w_1 \frac{e^{-w_1x_1-w_2x_2-w_3}}{(e^{-w_1x_1-w_2x_2-w_3}+1)^2} = 0.470074$$

$$\frac{\partial f}{\partial w_1} = x_1 \frac{e^{-w_1x_1-w_2x_2-w_3}}{(e^{-w_1x_1-w_2x_2-w_3}+1)^2} = -0.235004$$

$$f(x_1, x_2) = \frac{1}{1+e^{-(x_1w_1+x_2w_2+w_3)}}$$



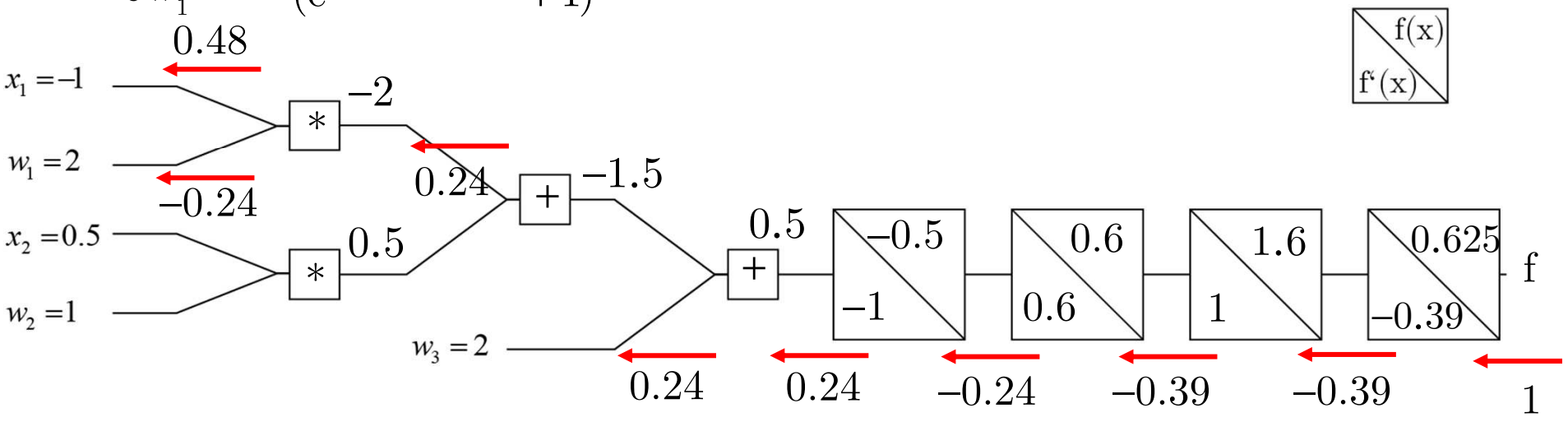
Computational Graph

Complex Example

$$\frac{\partial f}{\partial x_1} = w_1 \frac{e^{-w_1x_1-w_2x_2-w_3}}{(e^{-w_1x_1-w_2x_2-w_3}+1)^2} = 0.470074$$

$$\frac{\partial f}{\partial w_1} = x_1 \frac{e^{-w_1x_1-w_2x_2-w_3}}{(e^{-w_1x_1-w_2x_2-w_3}+1)^2} = -0.235004$$

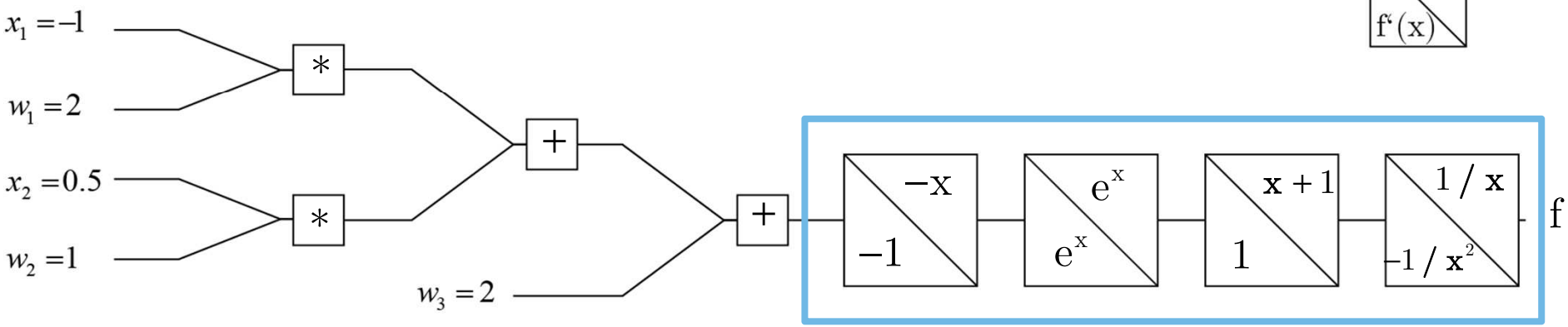
$$f(x_1, x_2) = \frac{1}{1+e^{-(x_1w_1+x_2w_2+w_3)}}$$



Computational Graph

Complex Example

$$f(x_1, x_2) = \frac{1}{1 + e^{-(x_1 w_1 + x_2 w_2 + w_3)}}$$

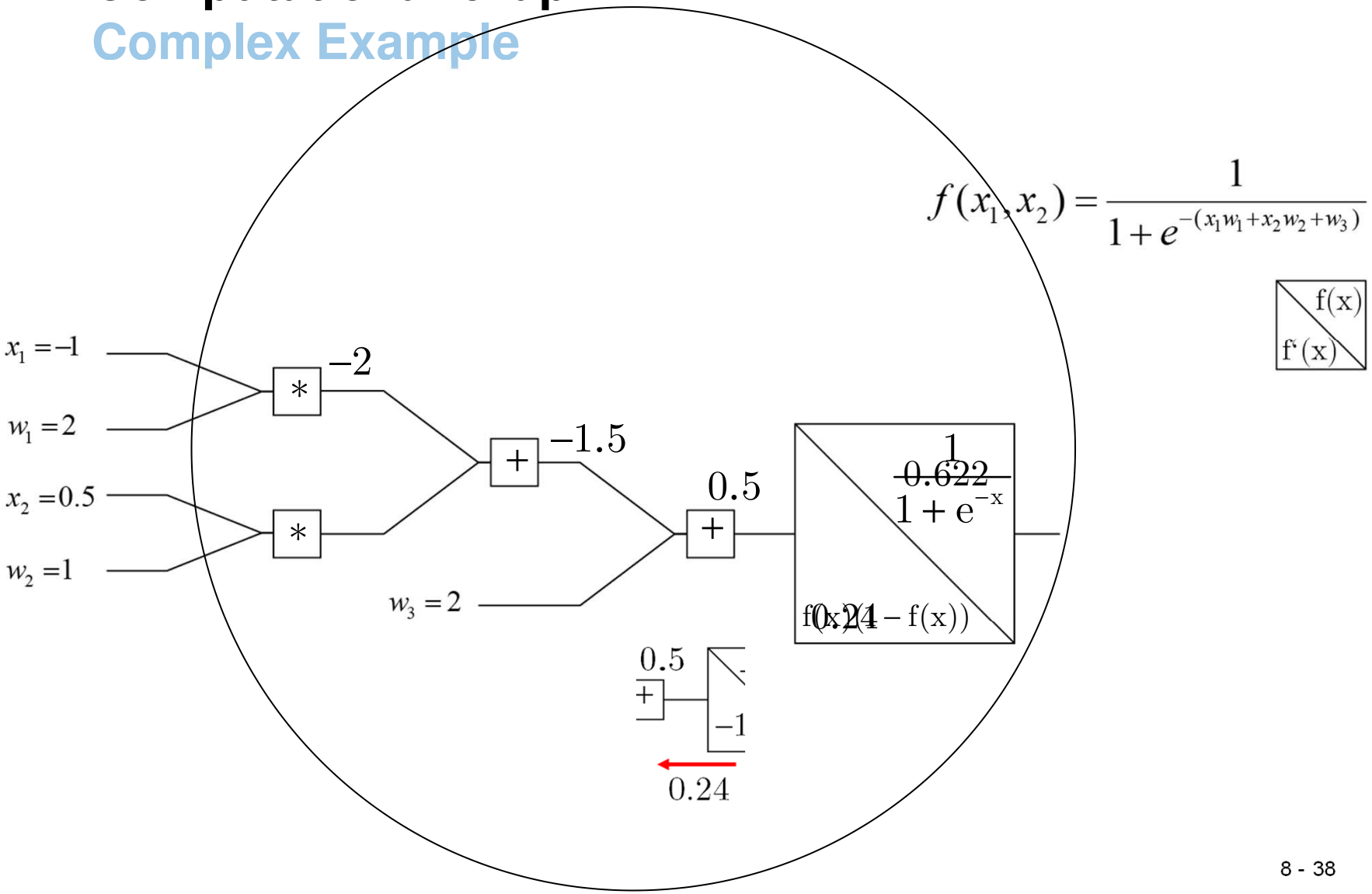


$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

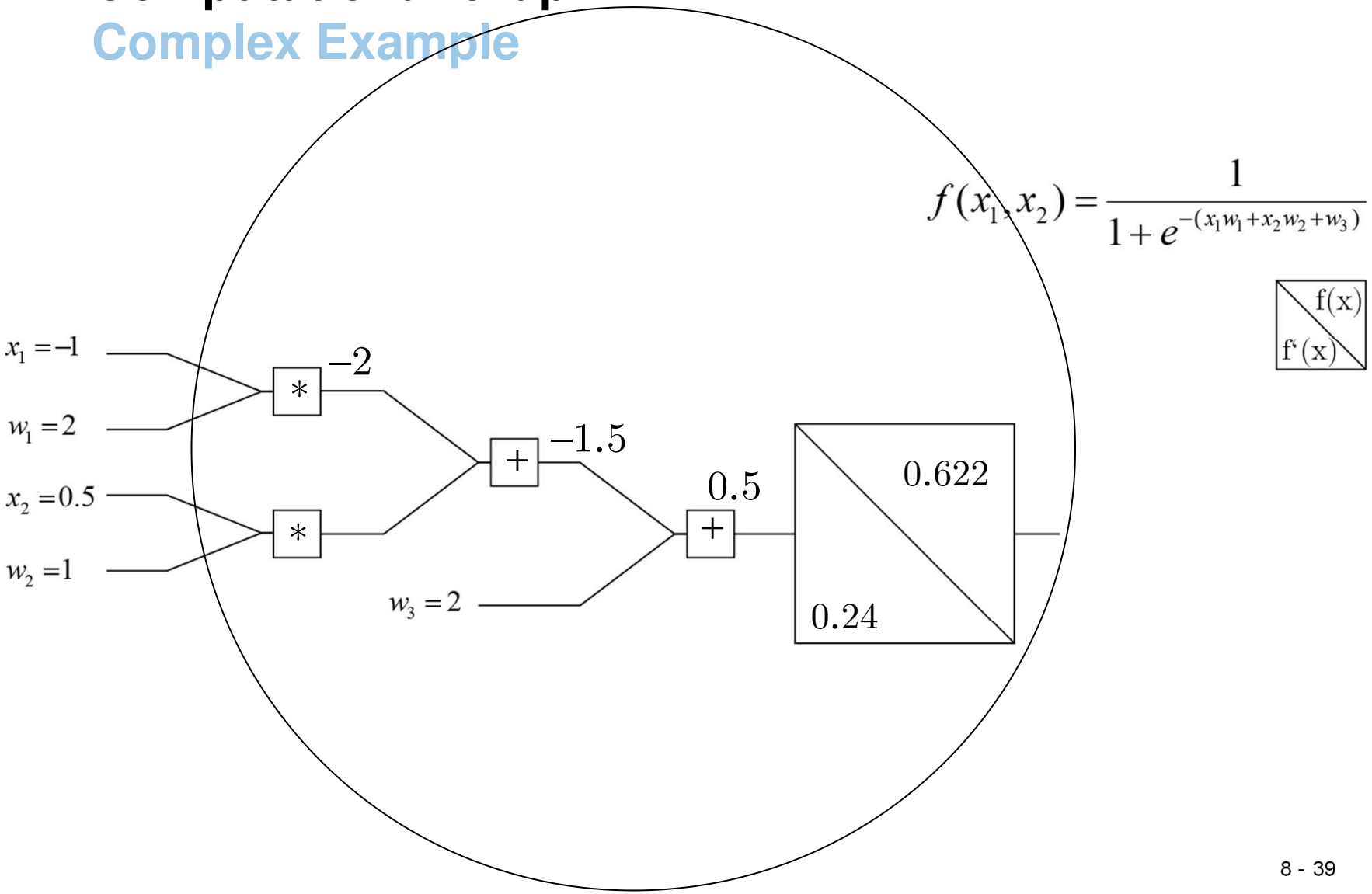
Computational Graph

Complex Example



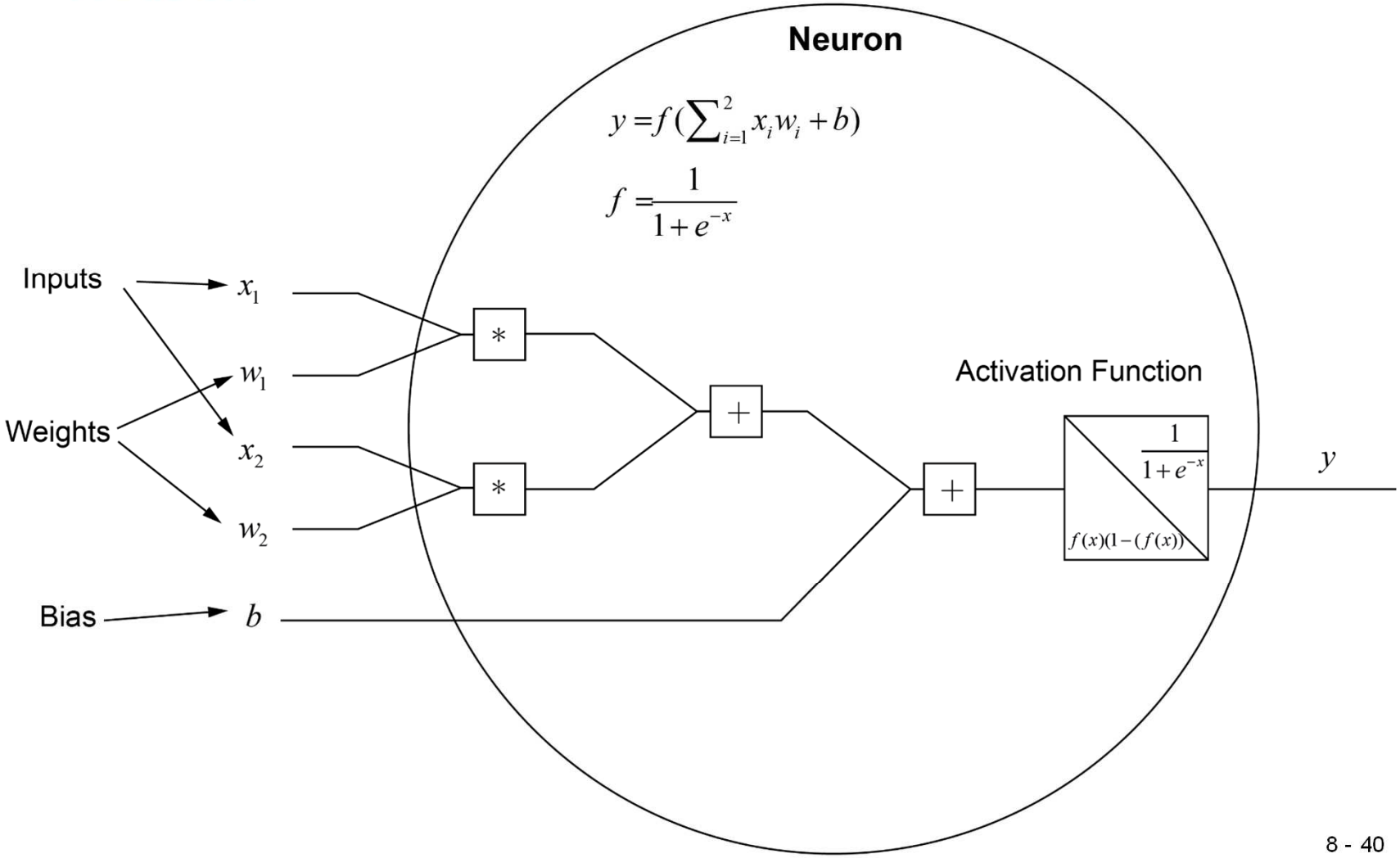
Computational Graph

Complex Example



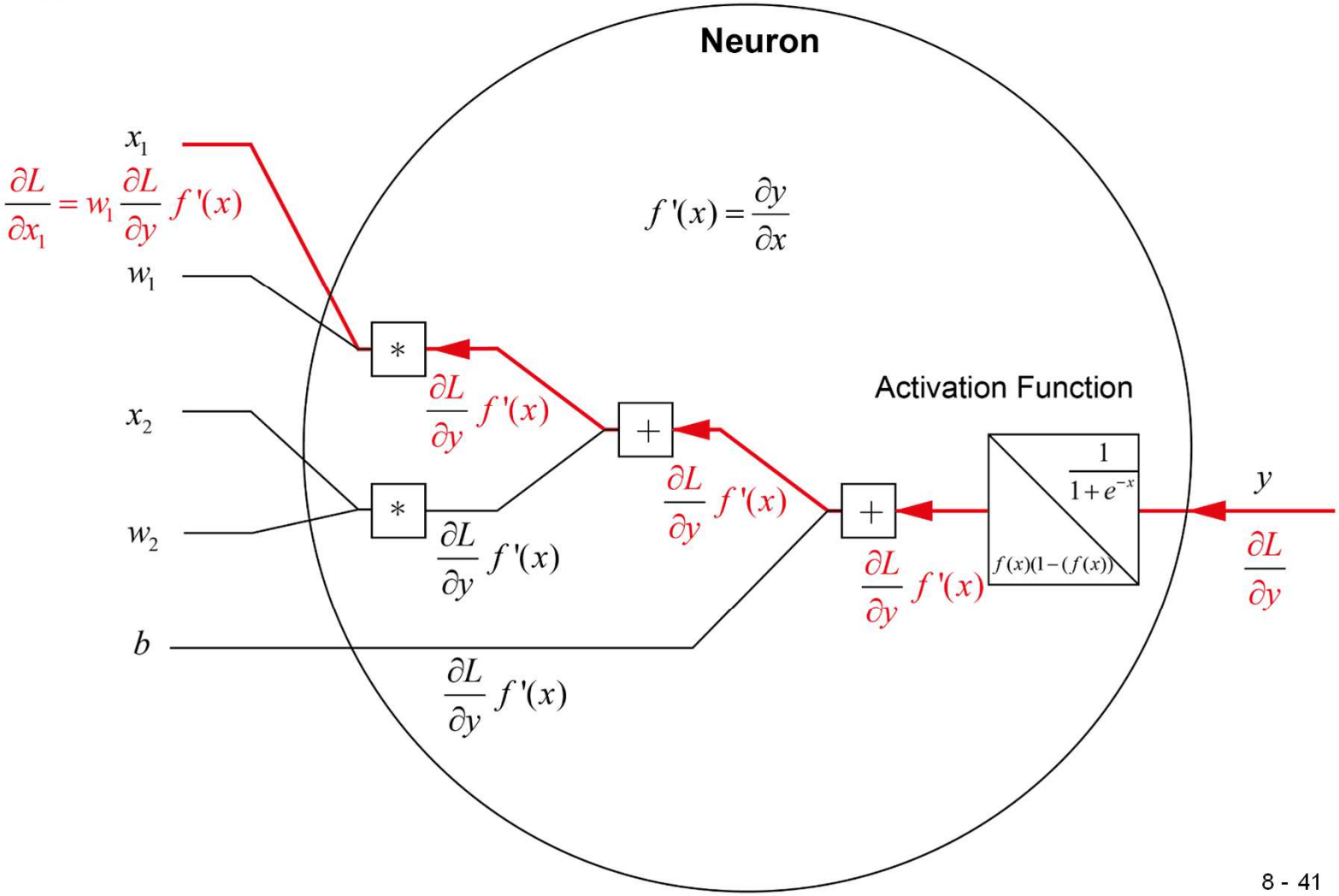
Computational Graph

Neuron



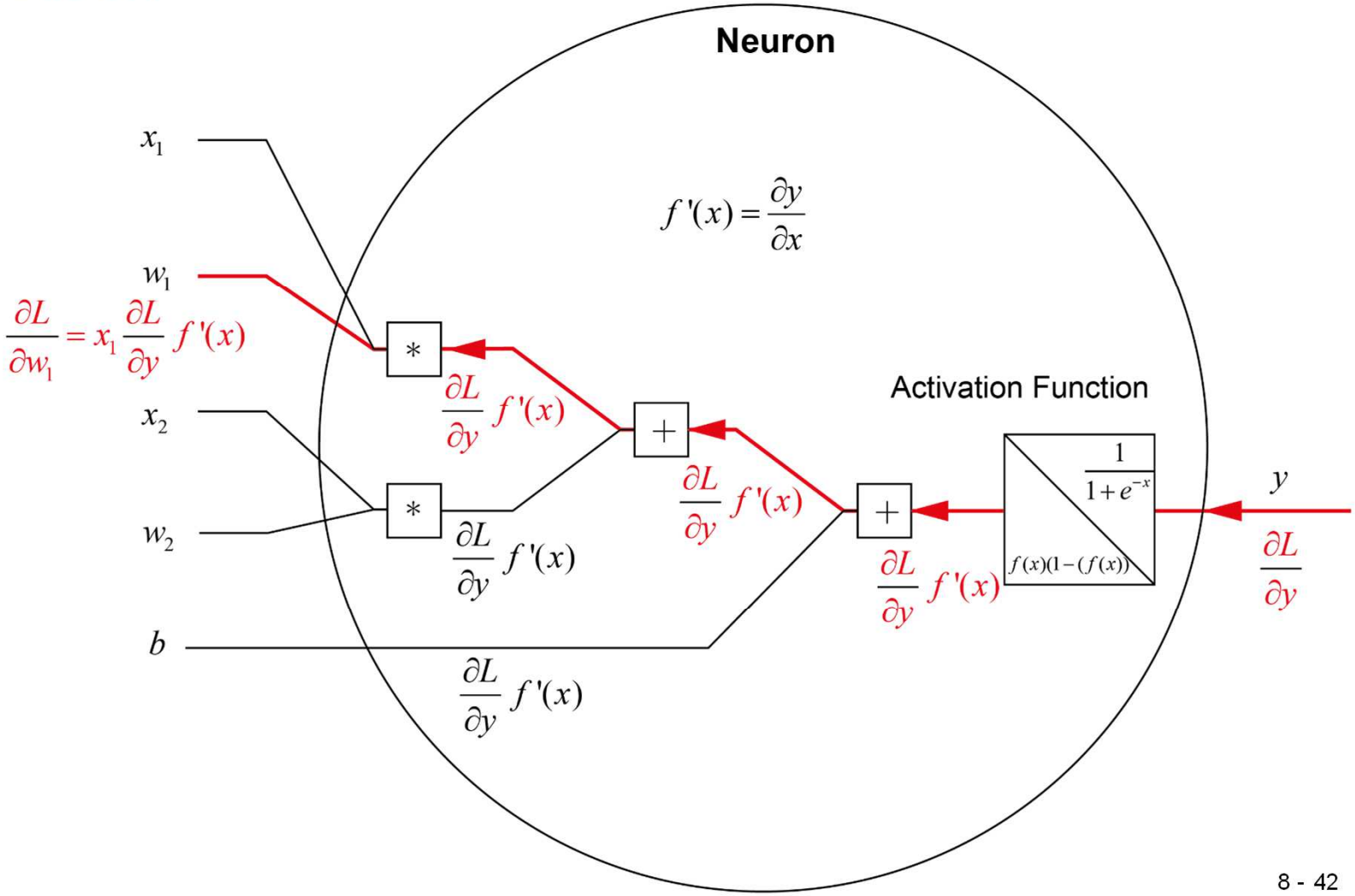
Computational Graph

Neuron



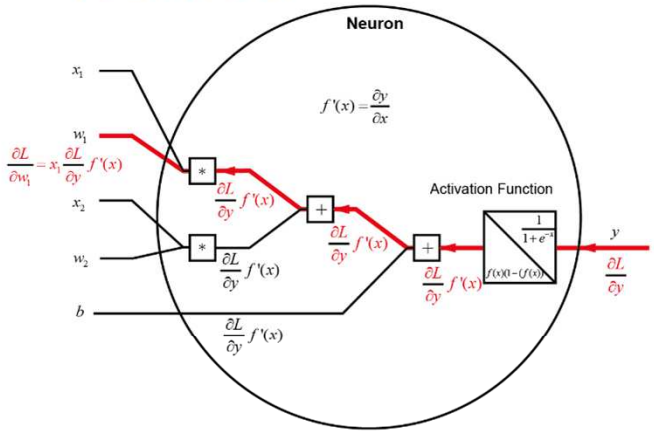
Computational Graph

Neuron

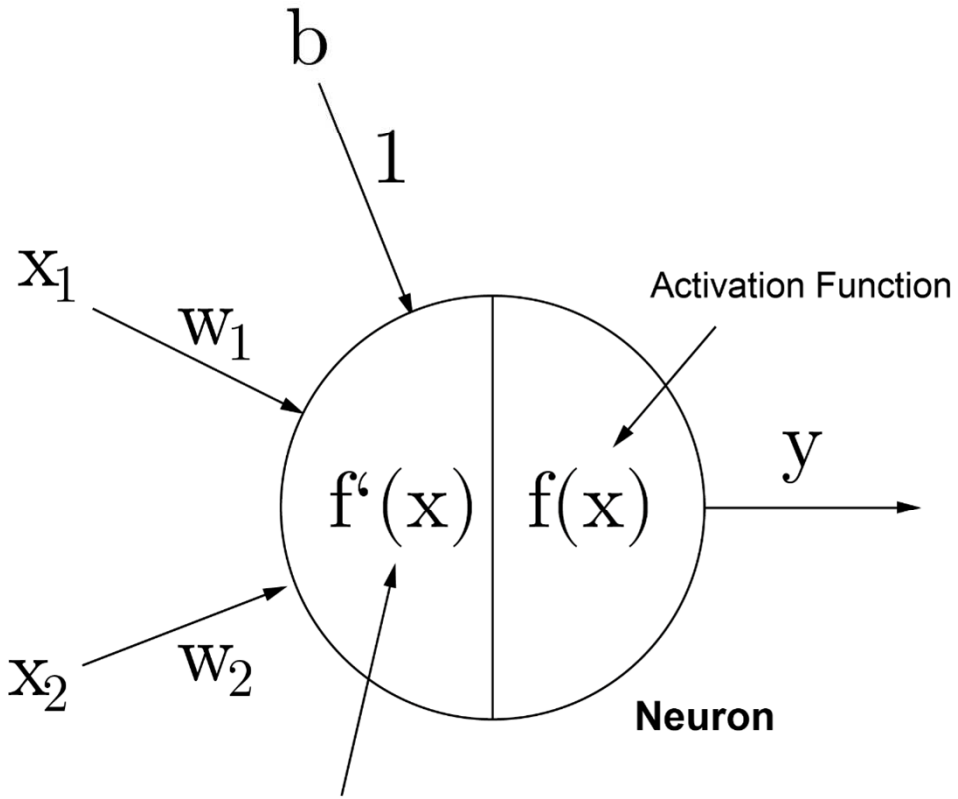


Computational Graph

Neuron

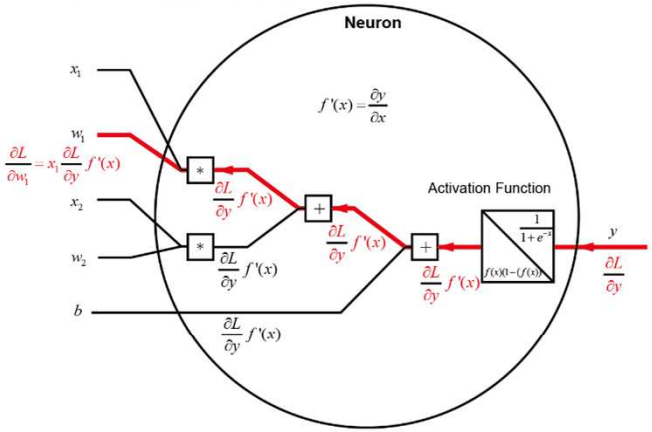


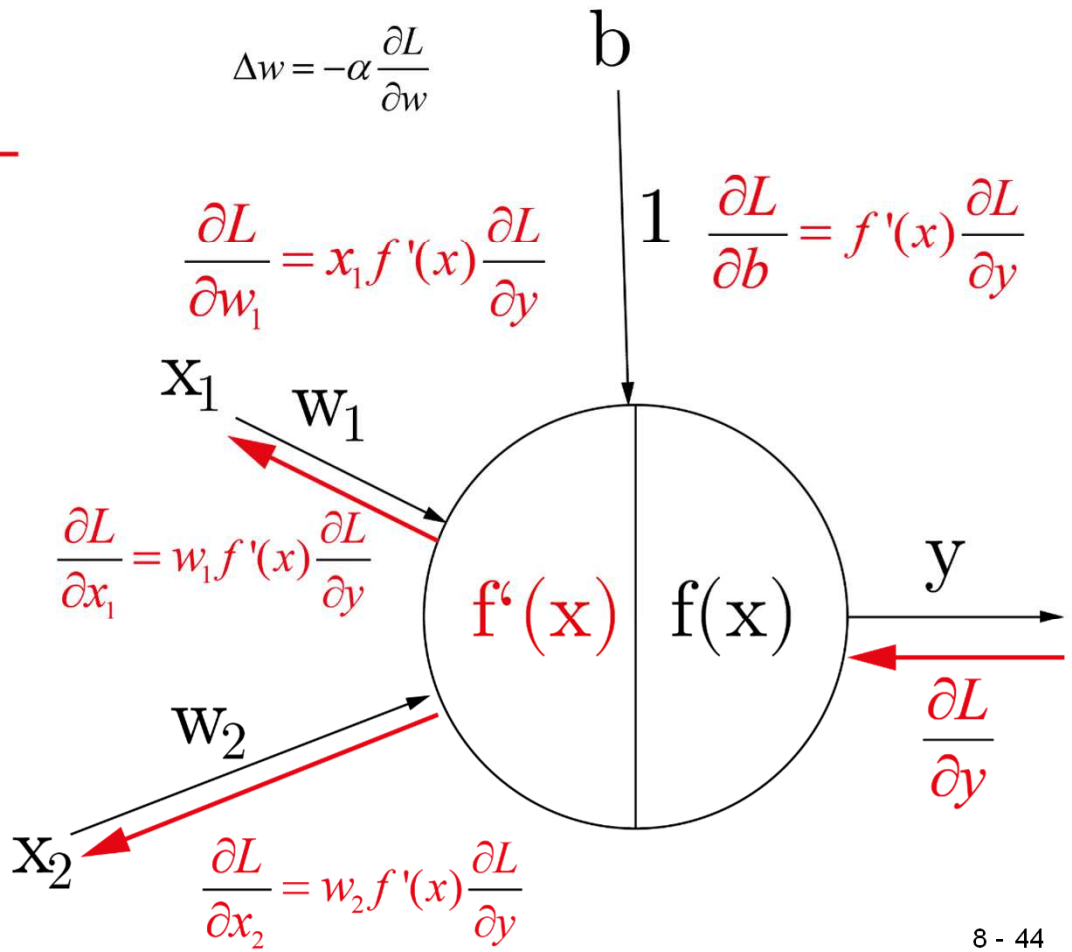
$$y = f\left(\sum_{i=1}^{2} x_i w_i + b\right)$$



Computational Graph

Neuron



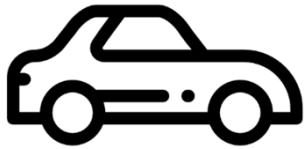


Deep Neural Networks

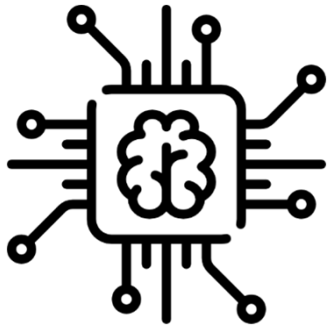
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

1. Chapter: Backpropagation
    - 1.1 Computational Graph
    - 1.2 Single Neuron
    - 1.3 Neural Chain
    - 1.4 Neural Network



2. Chapter: Neuronal Networks
    - 2.1 Activation Functions
    - 2.2 Fully Connected Layer
    - 2.3 Batches
    - 2.3 Weight Inititalisation



3. Chapter: Overview

Backpropagation

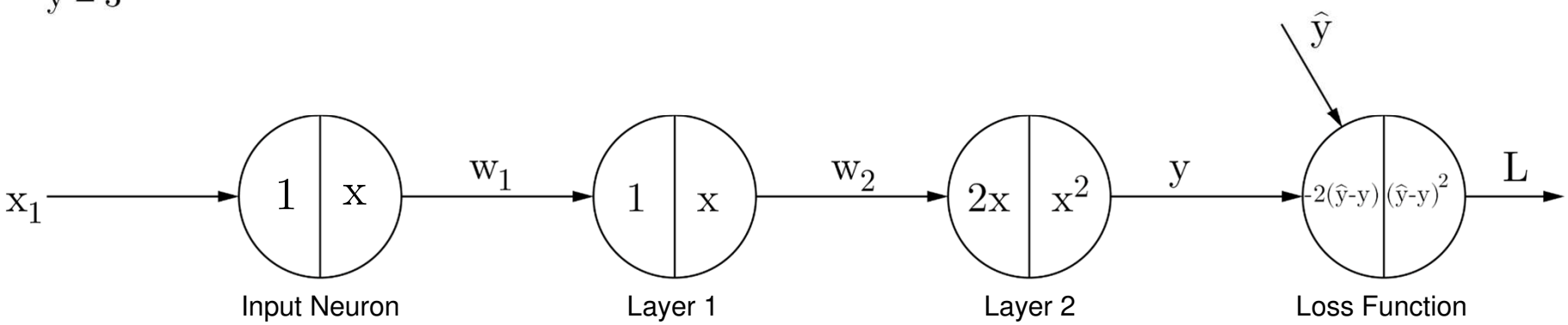
Neural Chain

$x_1 = 3$

$w_1 = 0.5$

$w_2 = -1$

$\hat{y} = 3$



Backpropagation

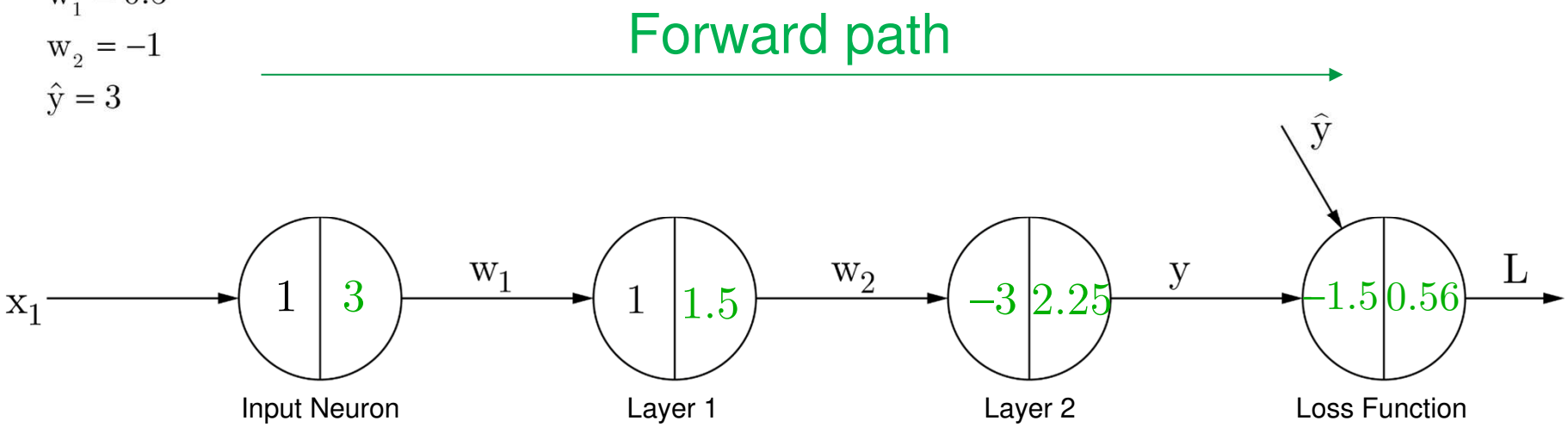
Neural Chain

$x_1 = 3$

$w_1 = 0.5$

$w_2 = -1$

$\hat{y} = 3$



Backpropagation

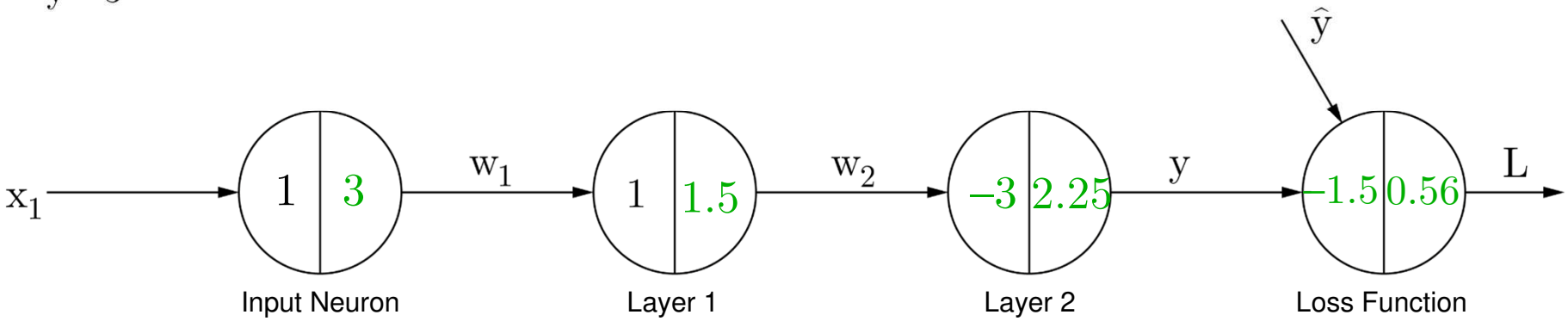
Neural Chain

$x_1 = 3$

$w_1 = 0.5$

$w_2 = -1$

$\hat{y} = 3$

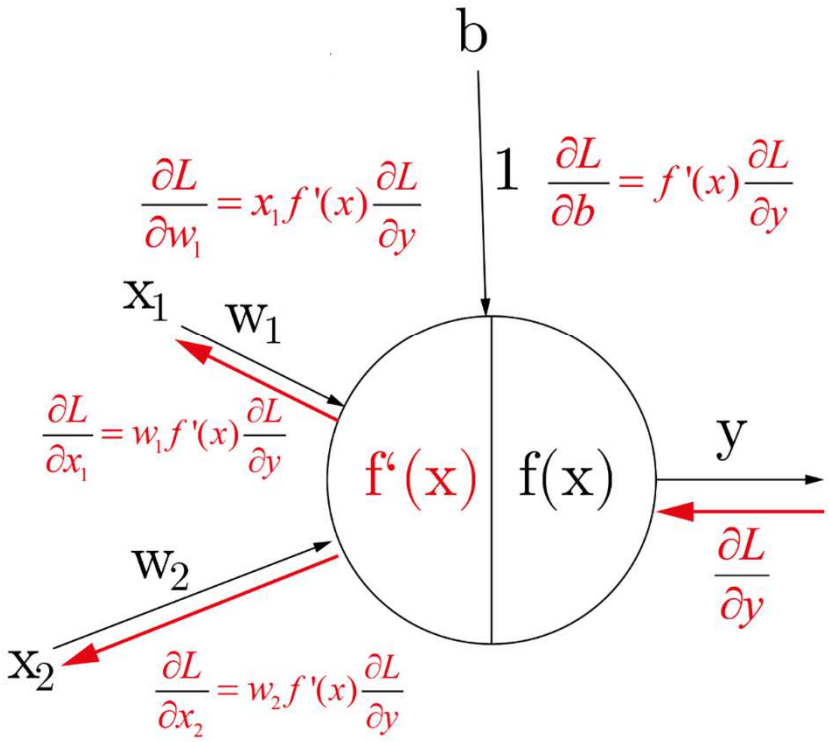


Backward path

$$\Delta w_1 = -\alpha \frac{\partial L}{\partial w_1}, \Delta w_2 = -\alpha \frac{\partial L}{\partial w_2}$$

Backpropagation

Neural Chain



Backpropagation

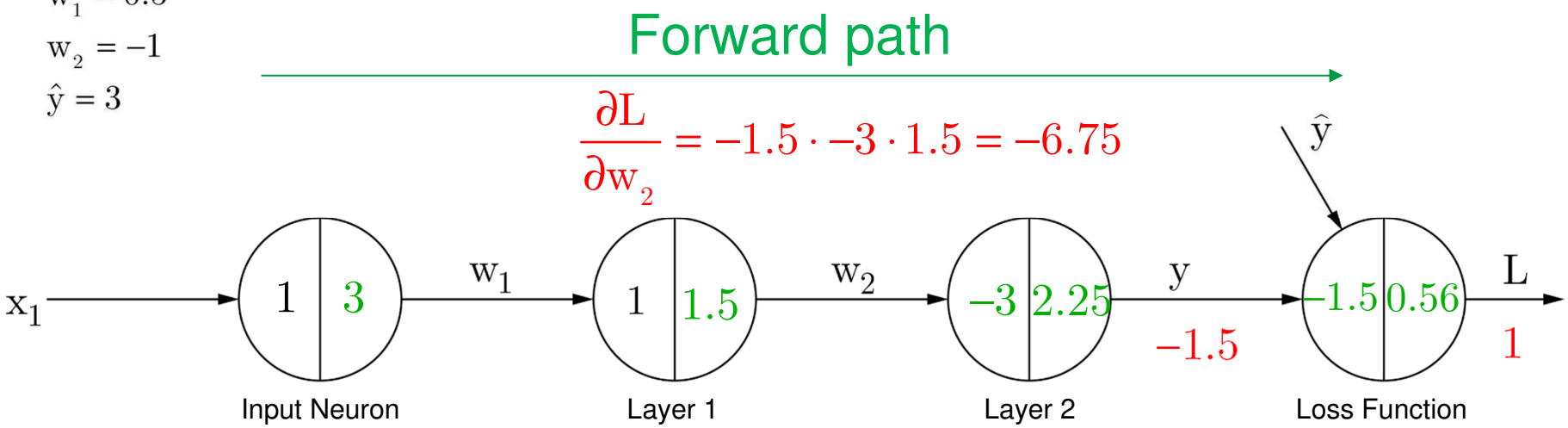
Neural Chain

$x_1 = 3$

$w_1 = 0.5$

$w_2 = -1$

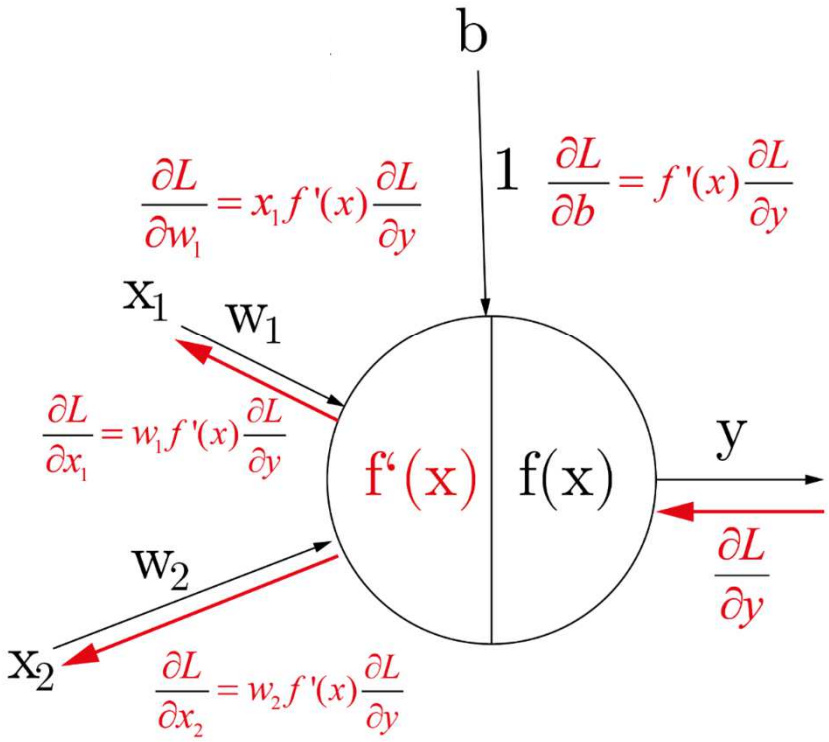
$\hat{y} = 3$



$$\Delta w_1 = -\alpha \frac{\partial L}{\partial w_1}, \Delta w_2 = -\alpha \frac{\partial L}{\partial w_2}$$

Backpropagation

Neural Chain



Backpropagation

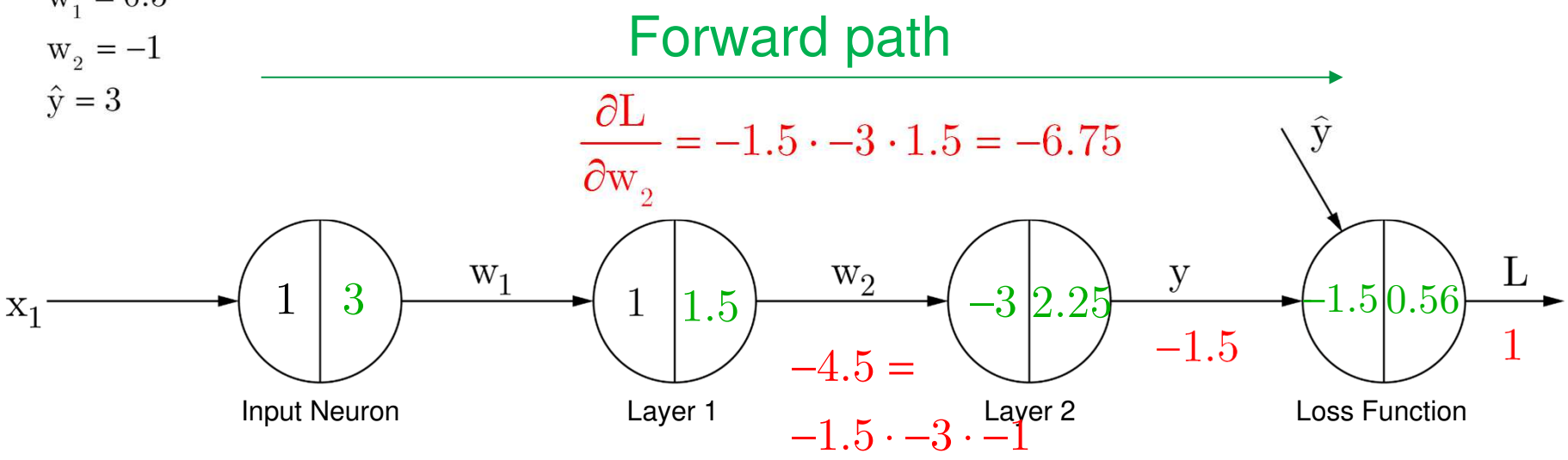
Neural Chain

$x_1 = 3$

$w_1 = 0.5$

$w_2 = -1$

$\hat{y} = 3$



$$\Delta w_1 = -\alpha \frac{\partial L}{\partial w_1}, \Delta w_2 = -\alpha \frac{\partial L}{\partial w_2}$$

Backpropagation

Neural Chain

$x_1 = 3$

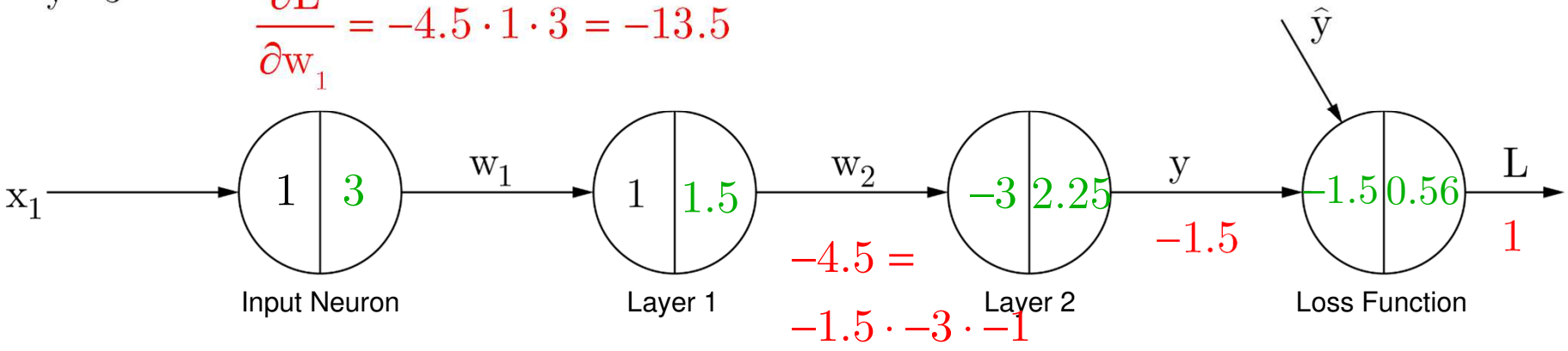
$w_1 = 0.5$

$w_2 = -1$

$\hat{y} = 3$

Forward path

$$\frac{\partial L}{\partial w_1} = -4.5 \cdot 1 \cdot 3 = -13.5$$



$$-4.5 = -1.5 \cdot -3 \cdot -1$$

Backward path

$$\Delta w_1 = -\alpha \frac{\partial L}{\partial w_1}, \Delta w_2 = -\alpha \frac{\partial L}{\partial w_2}$$

Backpropagation

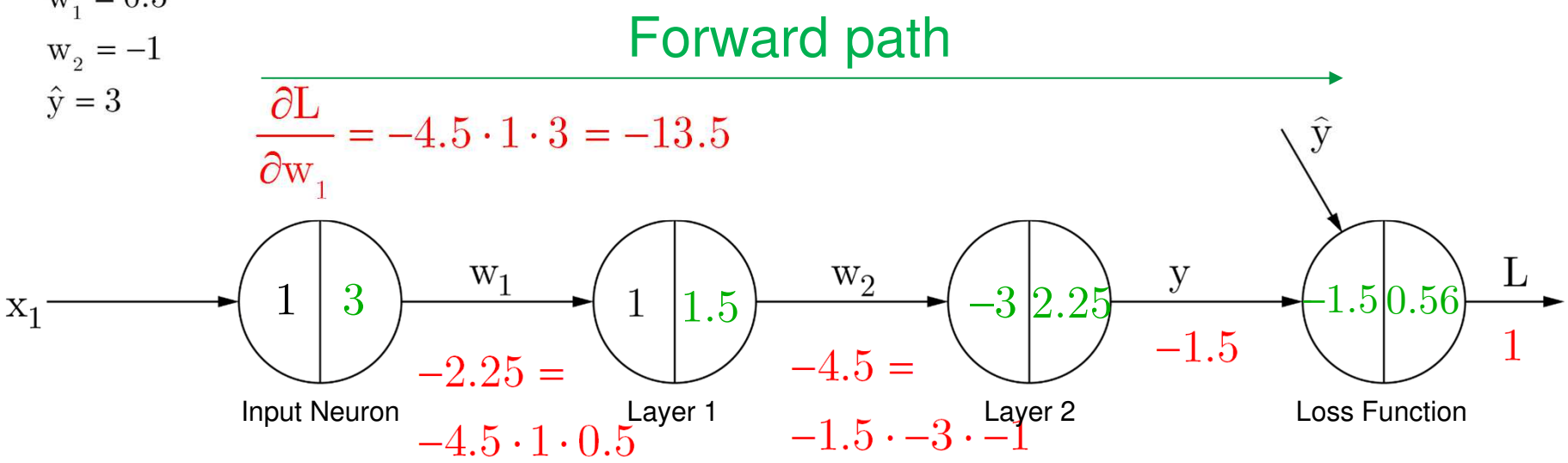
Neural Chain

$x_1 = 3$

$w_1 = 0.5$

$w_2 = -1$

$\hat{y} = 3$

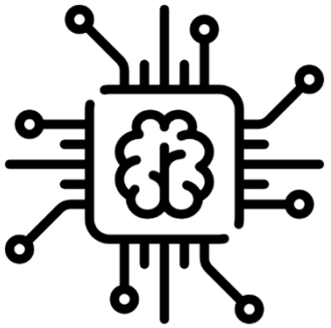


$$\Delta w_1 = -\alpha \frac{\partial L}{\partial w_1}, \Delta w_2 = -\alpha \frac{\partial L}{\partial w_2}$$

Deep Neural Networks

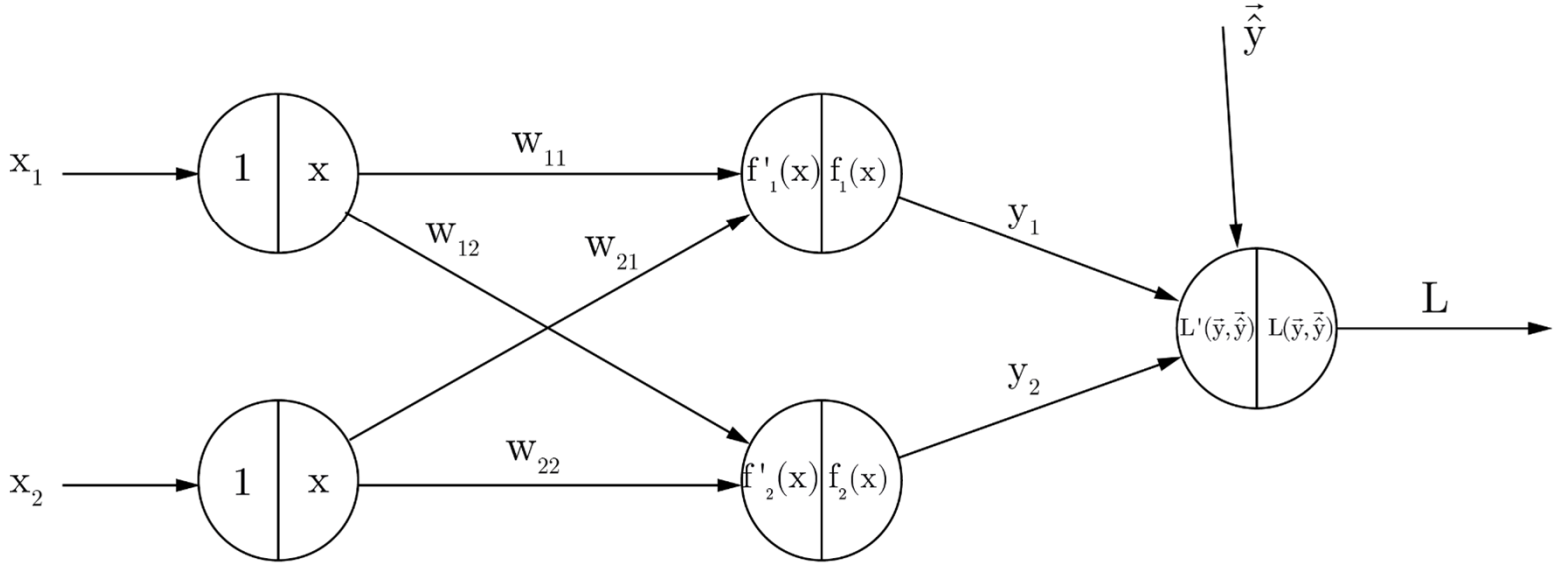
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda



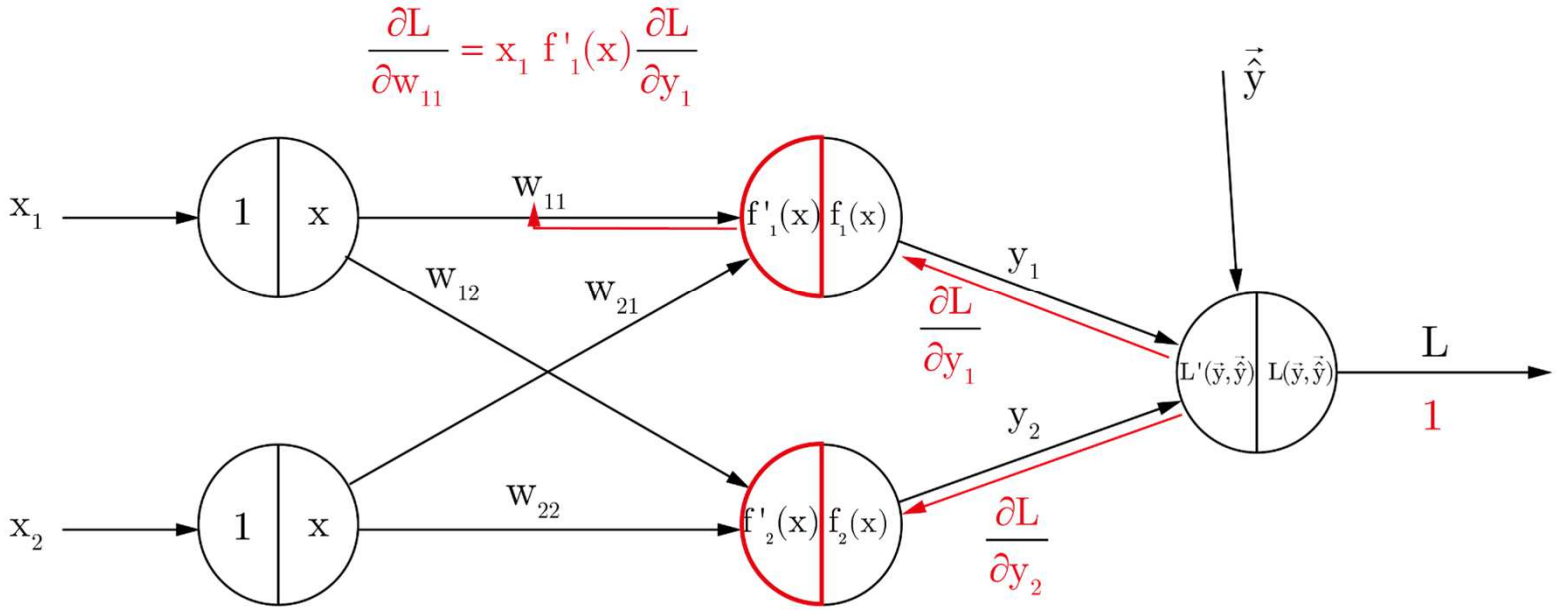
Backpropagation

Neural Network



Backpropagation

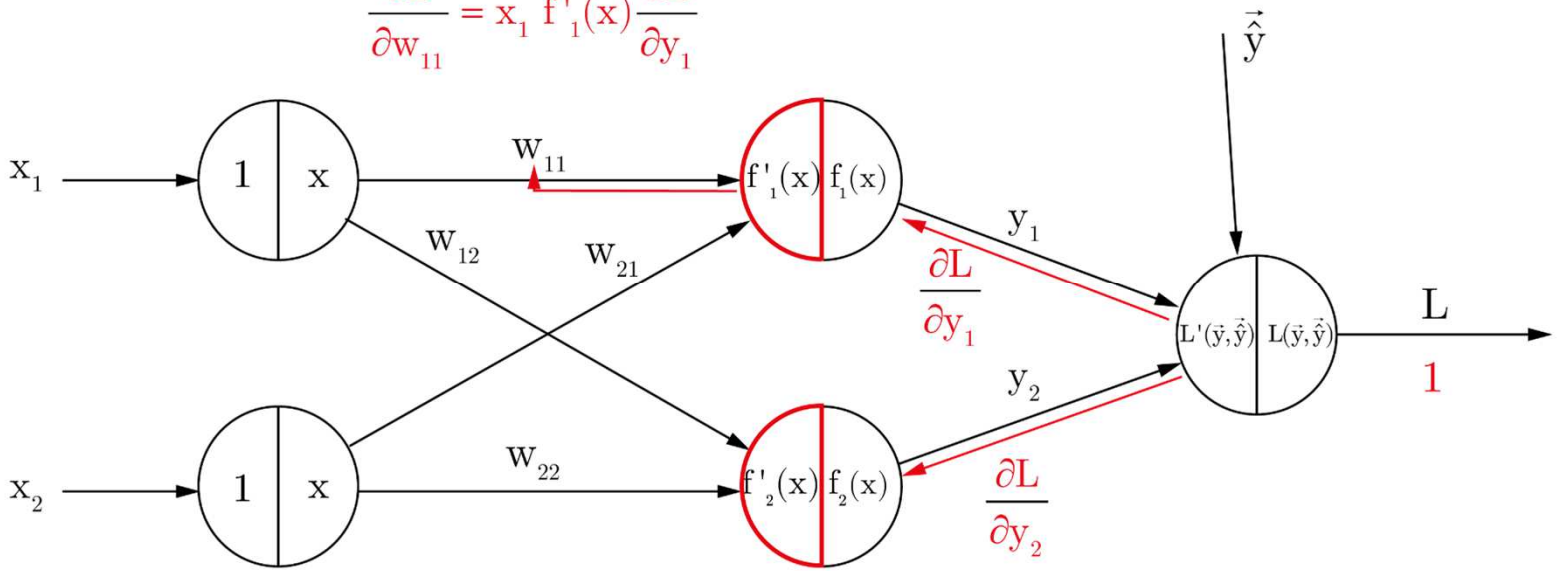
Neural Network



Backpropagation

Neural Network

$$\frac{\partial L}{\partial w_{11}} = x_1 \, f'_1(x) \frac{\partial L}{\partial y_1}$$



$$\frac{\partial L}{\partial w_{11}} = x_1 \, f'_1(x) \frac{\partial L}{\partial y_1} \qquad \frac{\partial L}{\partial w_{12}} = x_1 \, f'_2(x) \frac{\partial L}{\partial y_2} \qquad \frac{\partial L}{\partial w_{22}} = x_2 \, f'_2(x) \frac{\partial L}{\partial y_2} \qquad \frac{\partial L}{\partial w_{21}} = x_2 \, f'_1(x) \frac{\partial L}{\partial y_1}$$

Backpropagation

Neural Network

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

$$W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$$

$$\Rightarrow \Delta W = -\alpha \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} \end{pmatrix}$$

$$\Rightarrow \Delta W = -\alpha \begin{pmatrix} x_1 \frac{\partial L}{\partial y_1} f'_1 & x_1 \frac{\partial L}{\partial y_2} f'_2 \\ x_2 \frac{\partial L}{\partial y_1} f'_1 & x_2 \frac{\partial L}{\partial y_2} f'_2 \end{pmatrix}$$

Hadamard product

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \odot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1a & 2b \\ 3c & 4d \end{pmatrix}$$

$$\Rightarrow \Delta W = -\alpha \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \left[\begin{pmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \end{pmatrix} \odot \begin{pmatrix} f'_1 \\ f'_2 \end{pmatrix} \right]^T$$

Backpropagation

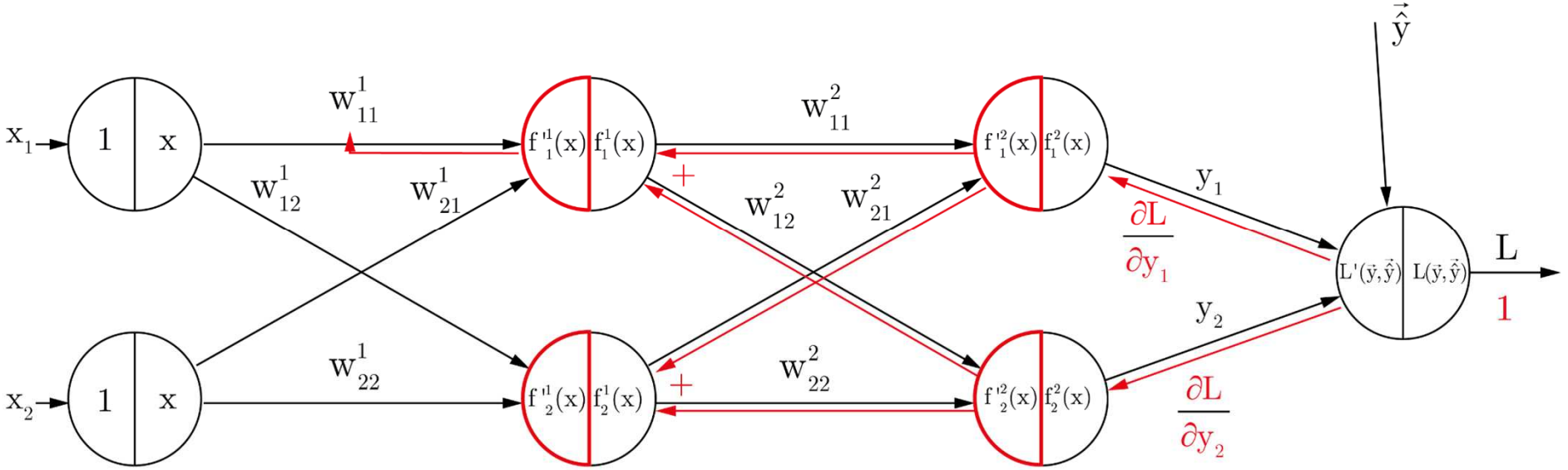
Neural Network

$$\Rightarrow \Delta W = -\alpha \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \left[\begin{pmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \end{pmatrix} \odot \begin{pmatrix} f'_1 \\ f'_2 \end{pmatrix} \right]^T = -\alpha \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \left[\begin{pmatrix} \frac{\partial L}{\partial y_1} f'_1 \\ \frac{\partial L}{\partial y_2} f'_2 \end{pmatrix} \right]^T$$

$$= -\alpha \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \begin{pmatrix} \frac{\partial L}{\partial y_1} f'_1 & \frac{\partial L}{\partial y_2} f'_2 \end{pmatrix} = -\alpha \begin{pmatrix} x_1 \frac{\partial L}{\partial y_1} f'_1 & x_1 \frac{\partial L}{\partial y_2} f'_2 \\ x_2 \frac{\partial L}{\partial y_1} f'_1 & x_2 \frac{\partial L}{\partial y_2} f'_2 \end{pmatrix}$$

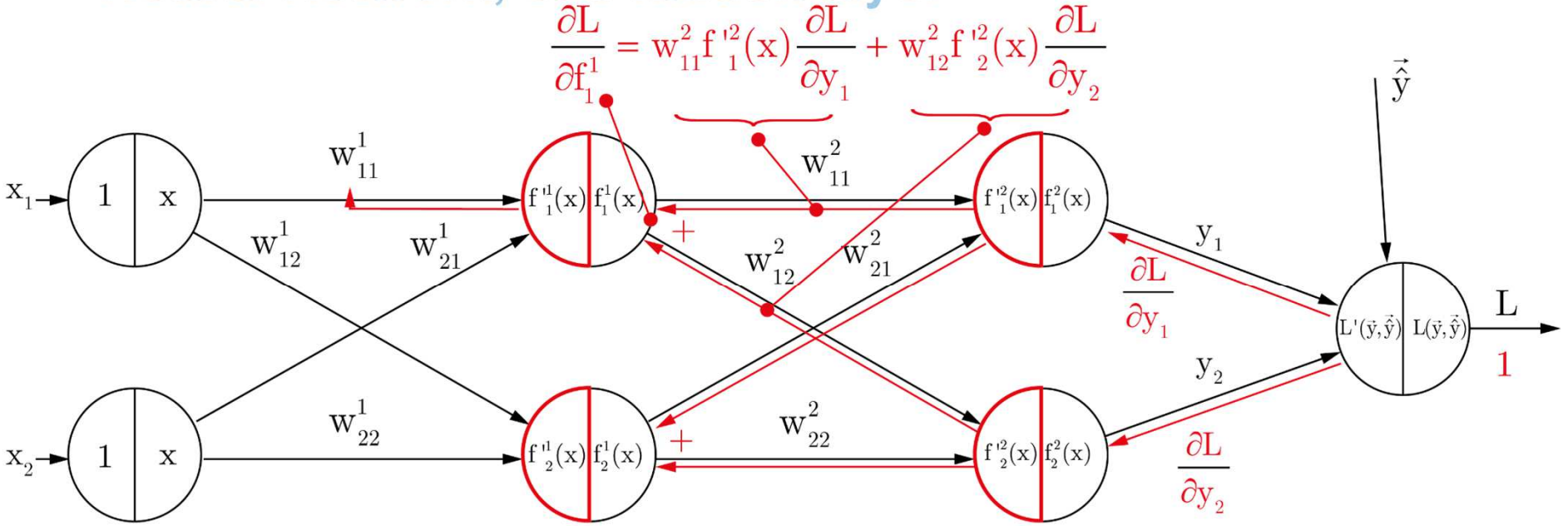
Backpropagation

Neural Network, one hidden layer



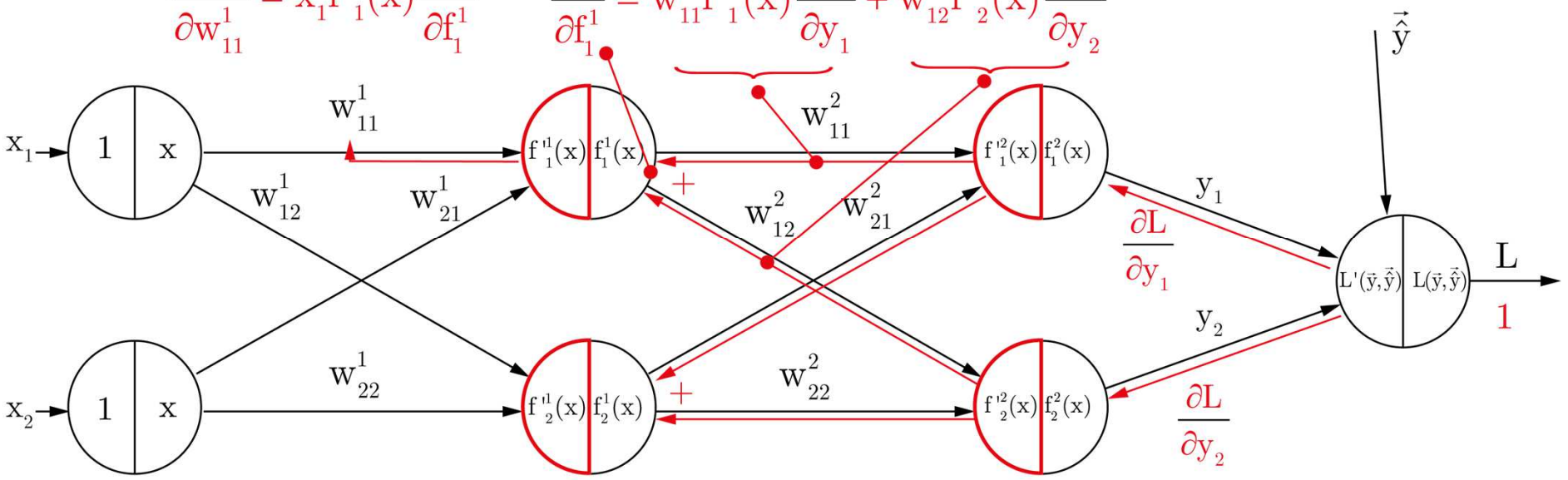
Backpropagation

Neural Network, one hidden layer



Backpropagation

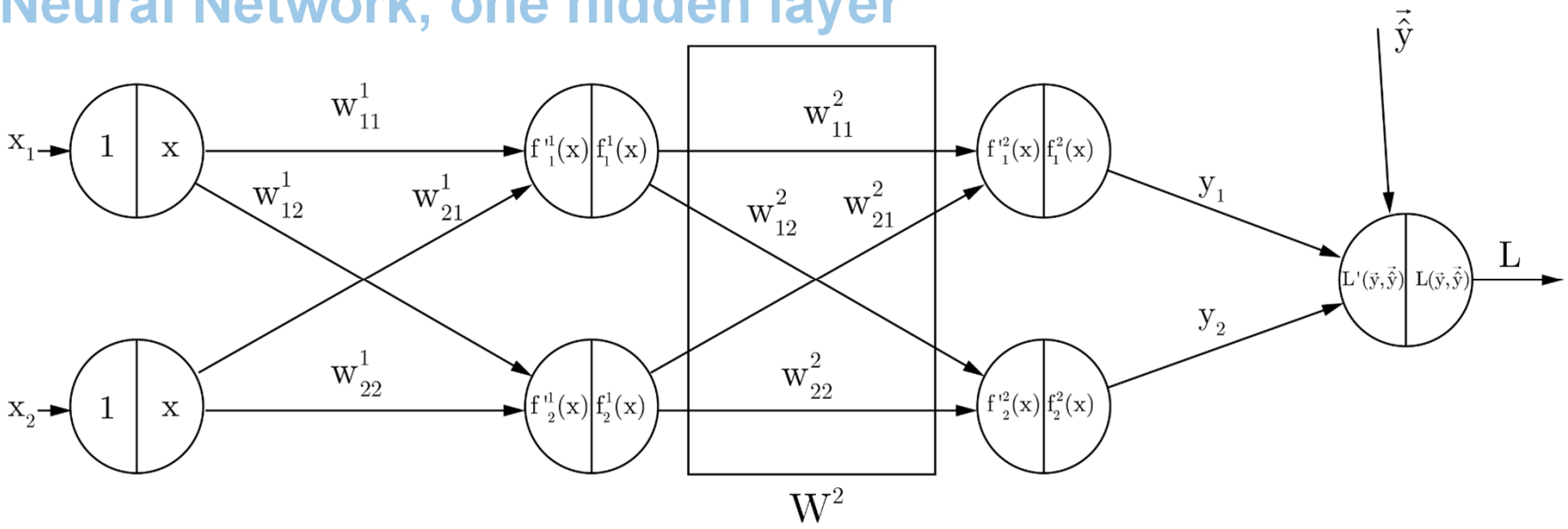
Neural Network, one hidden layer



$$\frac{\partial L}{\partial w_{11}^1} = x_1 f'^1_1(x) \frac{\partial L}{\partial f_1^1}, \qquad \frac{\partial L}{\partial w_{12}^1} = x_1 f'^1_2(x) \frac{\partial L}{\partial f_2^1}, \qquad \frac{\partial L}{\partial w_{21}^1} = x_2 f'^1_1(x) \frac{\partial L}{\partial f_1^1}, \qquad \frac{\partial L}{\partial w_{22}^1} = x_2 f'^1_2(x) \frac{\partial L}{\partial f_2^1}$$

Backpropagation

Neural Network, one hidden layer



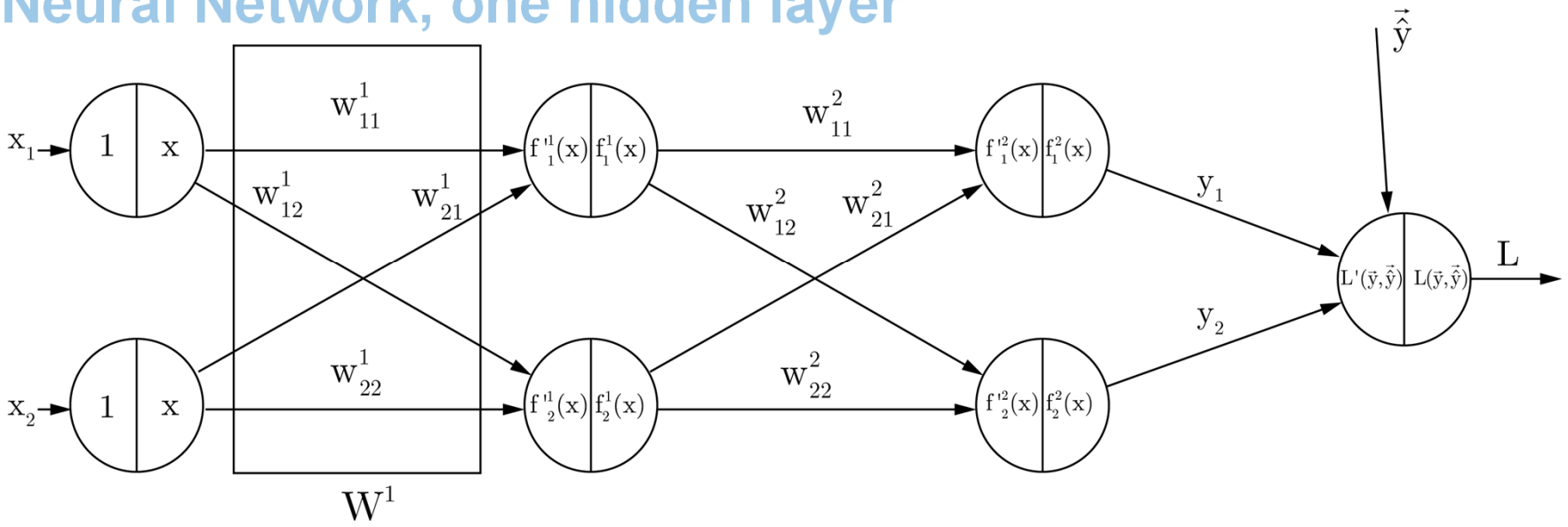
$$\Rightarrow \Delta W^2 = -\alpha \begin{pmatrix} f_1^1 \\ f_2^1 \end{pmatrix} \left[\begin{pmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \end{pmatrix} \odot \begin{pmatrix} f'^2_1 \\ f'^2_2 \end{pmatrix} \right]^T$$

Forward path

Backward path

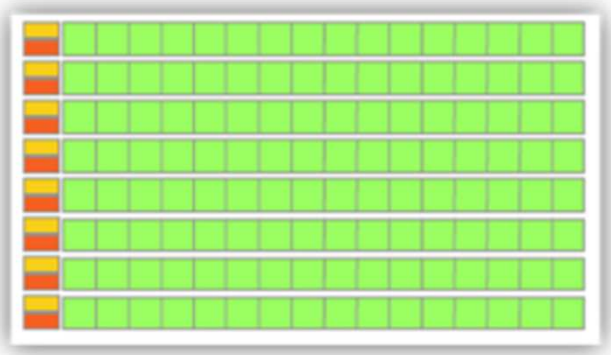
Backpropagation

Neural Network, one hidden layer



$$\Rightarrow \Delta W^1 = -\alpha \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \left[\begin{pmatrix} \frac{\partial L}{\partial f^1_1} \\ \frac{\partial L}{\partial f^1_2} \end{pmatrix} \odot \begin{pmatrix} f'^1_1 \\ f'^1_2 \end{pmatrix} \right]^T = -\alpha \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \left[\begin{pmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \end{pmatrix} \left[\begin{pmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \end{pmatrix} \odot \begin{pmatrix} f'^2_1 \\ f'^2_2 \end{pmatrix} \right]^T \right] \odot \begin{pmatrix} f'^1_1 \\ f'^1_2 \end{pmatrix} \Bigg]^T$$

CPU vs GPU



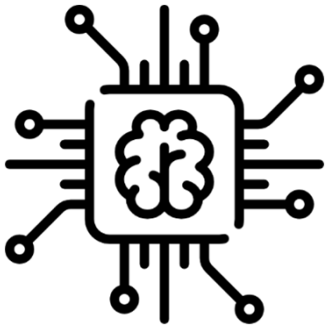
- Low computedensity
- Complex logic control
- Large Cache
- Low latencytollerance

- High computedensity
- Many calculation per memory acces
- Optimizer for parallel computing
- Low latency tolerance

Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

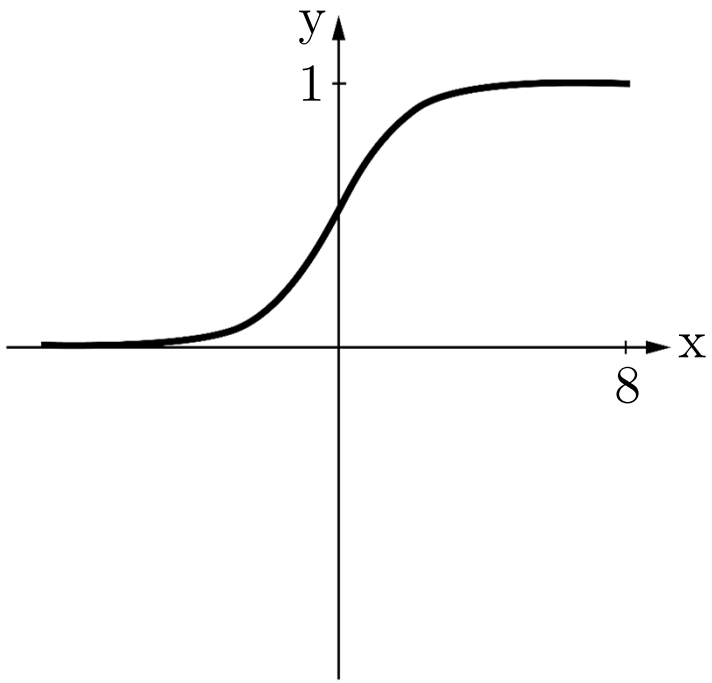


Activation Functions

Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$



Two main problems:

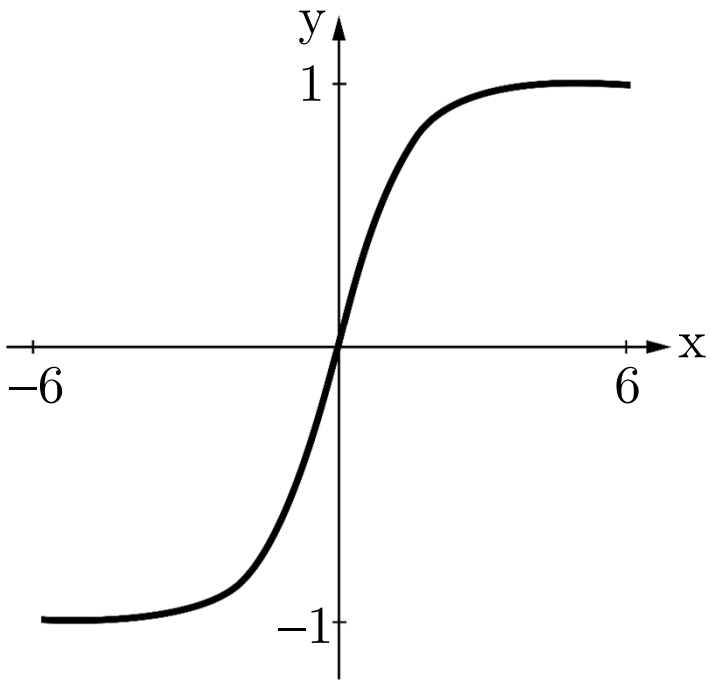
- Causes vanishing gradient: Gradient nearly zero for very large or small x, kills gradient and network stops learning
- Output isn‘t zero centered: Always all gradients positive or all negativ, inefficient weight updates

Activation Functions

Tangens hyperbolicus (tanh)

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

$$f'(x) = \frac{4}{(e^{x} + e^{-x})^{2}}$$



Better than sigmoid:

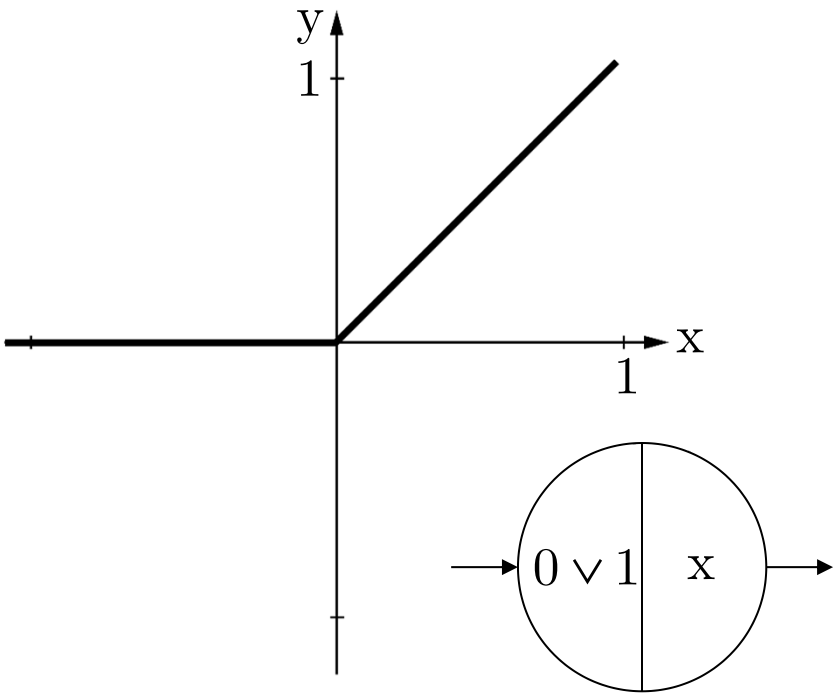
- Output is zero centered
- But still causes vanishing gradient

Activation Functions

Rectified Linear Unit (ReLU)

$$f(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$



Most common activation functio:

- Computationally efficient
- Converges very fast
- Does not activate all neurons at the same time

Problem:

- Gradient is zero for x<0 and can cause vanishing gradient -> dead relus may happen
- Not zero centered

Usage:

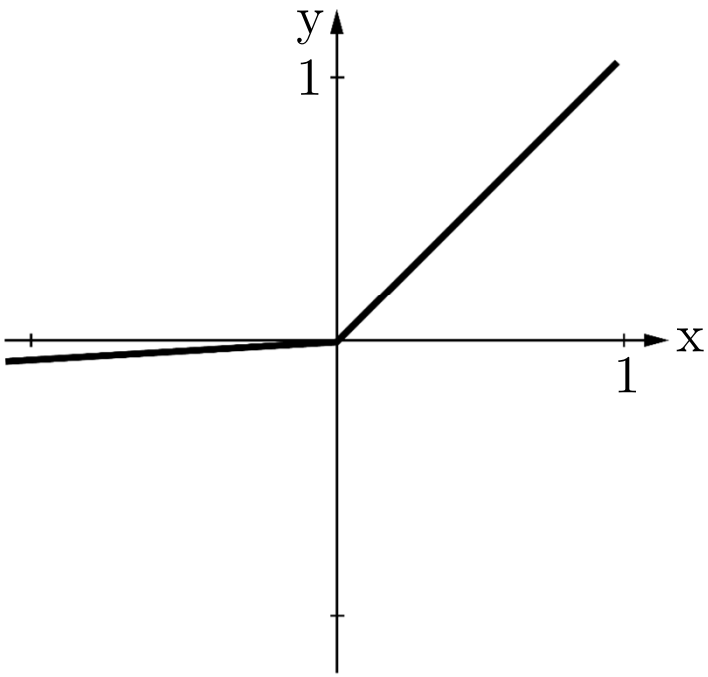
- Mostly used in hidden layers
- Positive bias at init to get active ReLU

Activation Functions

Leaky ReLU

$$f(x) = \begin{cases} x, x \geq 0 \\ ax, x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, x \geq 0 \\ a, x < 0 \end{cases}$$



Improves on ReLU:

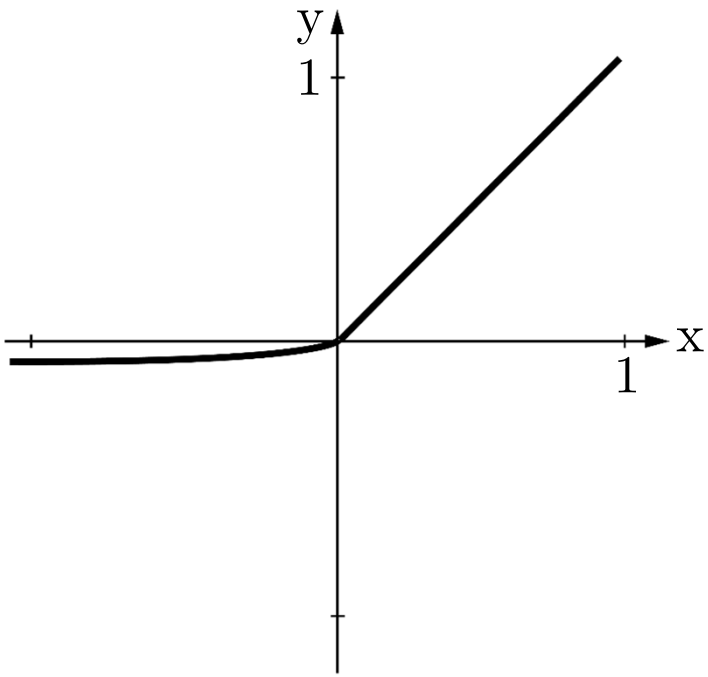
- Removes zero part of ReLU by adding a small slope. More stable then relu, but adds another paramter
- Computationally efficient
- Converges very fast
- Doesn‘t die
- Parameter a can also be learned by the network

Activation Functions

Exponential Linea Unit (ELU)

$$f(x) = \begin{cases} x, x \geq 0 \\ a(e^x - 1), x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, x \geq 0 \\ ae^x, x < 0 \end{cases}$$



- Benefits of ReLU and Leaky ReLU
- Computation requires e^x

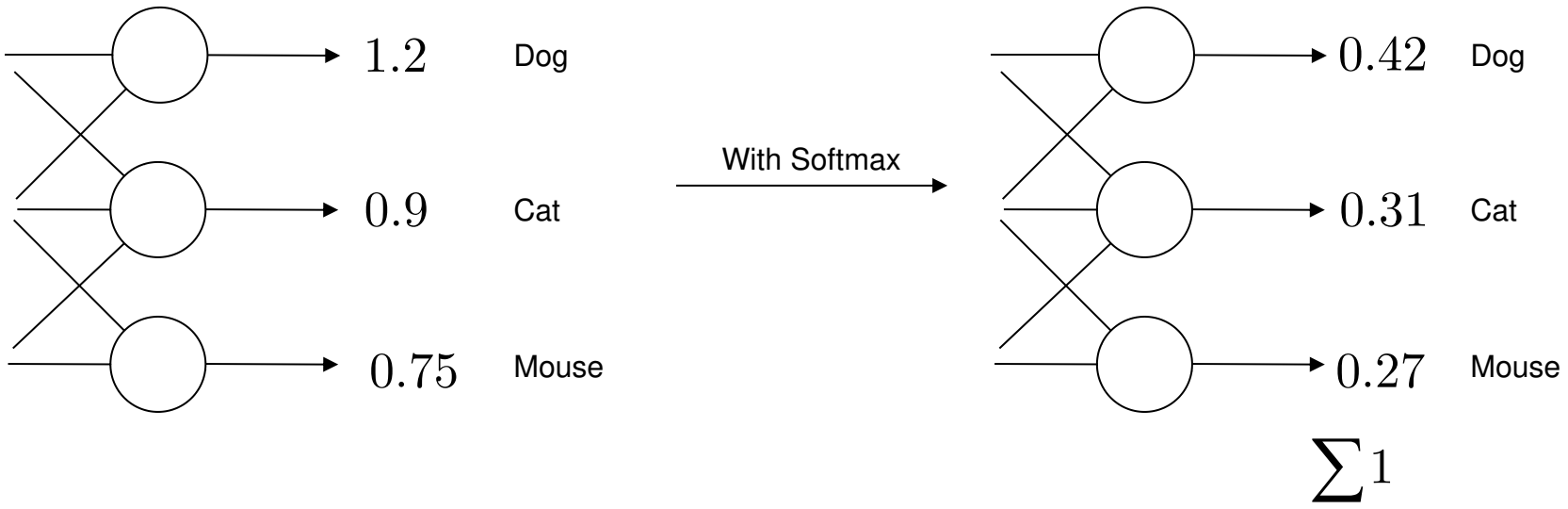
Activation Functions

Softmax

$$f(x) = \frac{e^{y_i}}{\sum_{k=1}^{K} e^{y_k}}$$

- Type of Sigmoid, handy for classification problems.
- Divides by the sum of all outputs, allows for percentage representation

Classifier:



Activation Functions

Rule of thumb

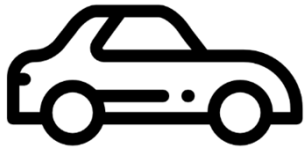
- Sigmoid / Softmax for classifiers
- Sigmoid, tanh sometimes avoided due to vanishing gradient
- ReLU mostly used today, but should only be used in hidden layer
- Start with ReLU if you don‘t get optimal results go for LeakyReLU od ELU

Deep Neural Networks

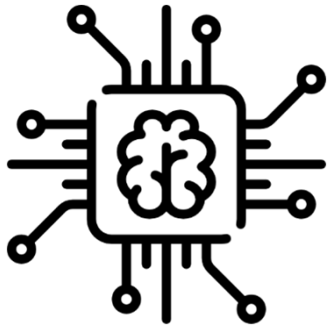
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

1. Chapter: Backpropagation
    - 1.1 Computational Graph
    - 1.2 Single Neuron
    - 1.3 Neural Chain
    - 1.4 Neural Network



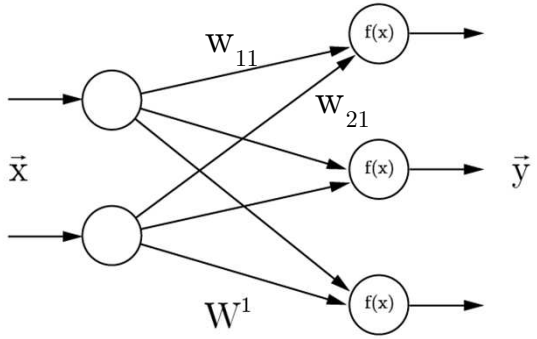
2. Chapter: Neuronal Networks
    - 2.1 Activation Functions
    - 2.2 Fully Connected Layer
    - 2.3 Batches
    - 2.3 Weight Inititalisation



3. Chapter: Overview

Fully Connected Layer

Dimensions



$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} f(x_1 w_{11} + x_2 w_{21} + b_1) \\ f(x_1 w_{12} + x_2 w_{22} + b_2) \\ f(x_1 w_{13} + x_2 w_{23} + b_3) \end{pmatrix} = f\begin{pmatrix} x_1 w_{11} + x_2 w_{21} + b_1 \\ x_1 w_{12} + x_2 w_{22} + b_2 \\ x_1 w_{13} + x_2 w_{23} + b_3 \end{pmatrix}$$

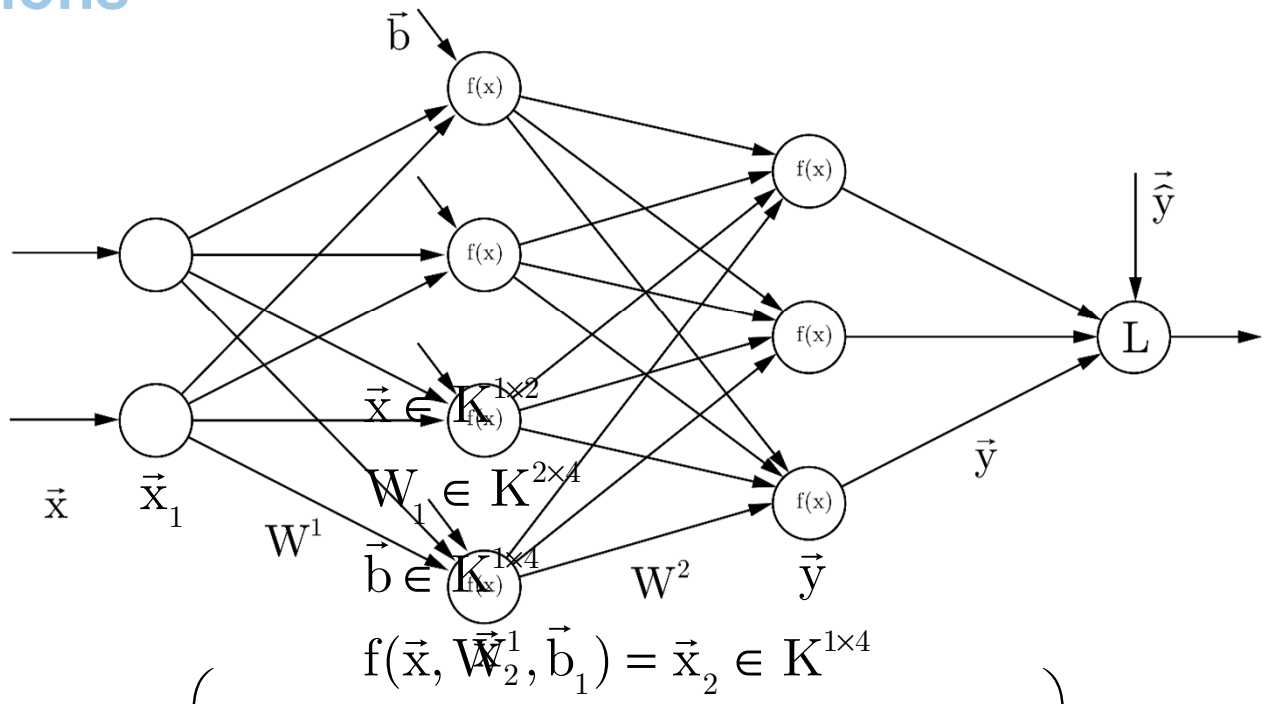
$$= f\left(\begin{pmatrix} x_1 w_{11} + x_2 w_{21} \\ x_1 w_{12} + x_2 w_{22} \\ x_1 w_{13} + x_2 w_{23} \end{pmatrix} + \vec{b}\right) = f\left(\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{23} & w_{33} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \vec{b}\right) = f\left(W^1\vec{x} + \vec{b}\right)$$

Alternativ:

$$\vec{y}^T = \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix} = f\left(W^1\vec{x} + \vec{b}\right)^T = f\left(\vec{x}^T W^{1T} + \vec{b}^T\right)$$

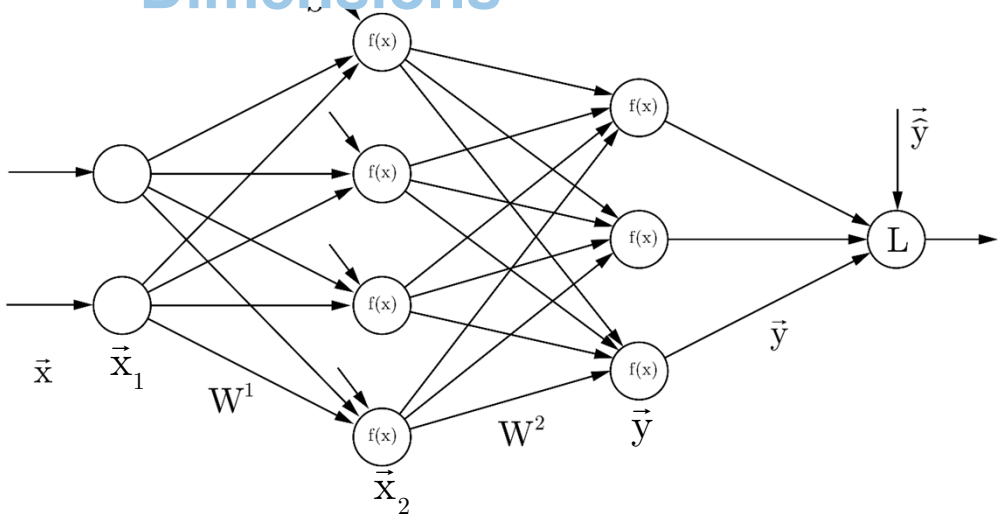
Fully Connected Layer

Dimensions



Fully Connected Layer

Dimensions



$$\vec{x} \in K^{1\times 2}$$

$$W_1 \in K^{2\times 4}$$

$$\vec{b} \in K^{1\times 4}$$

$$f(\vec{x}, W^1, \vec{b}_1) = \vec{x}_2 \in K^{1\times 4}$$

$$\vec{x}_2 = f\left(\underbrace{\begin{pmatrix} x_1 & x_2 \end{pmatrix}}_{\vec{x}} \cdot \underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{pmatrix}}_{W^1} + \vec{b}\right)$$

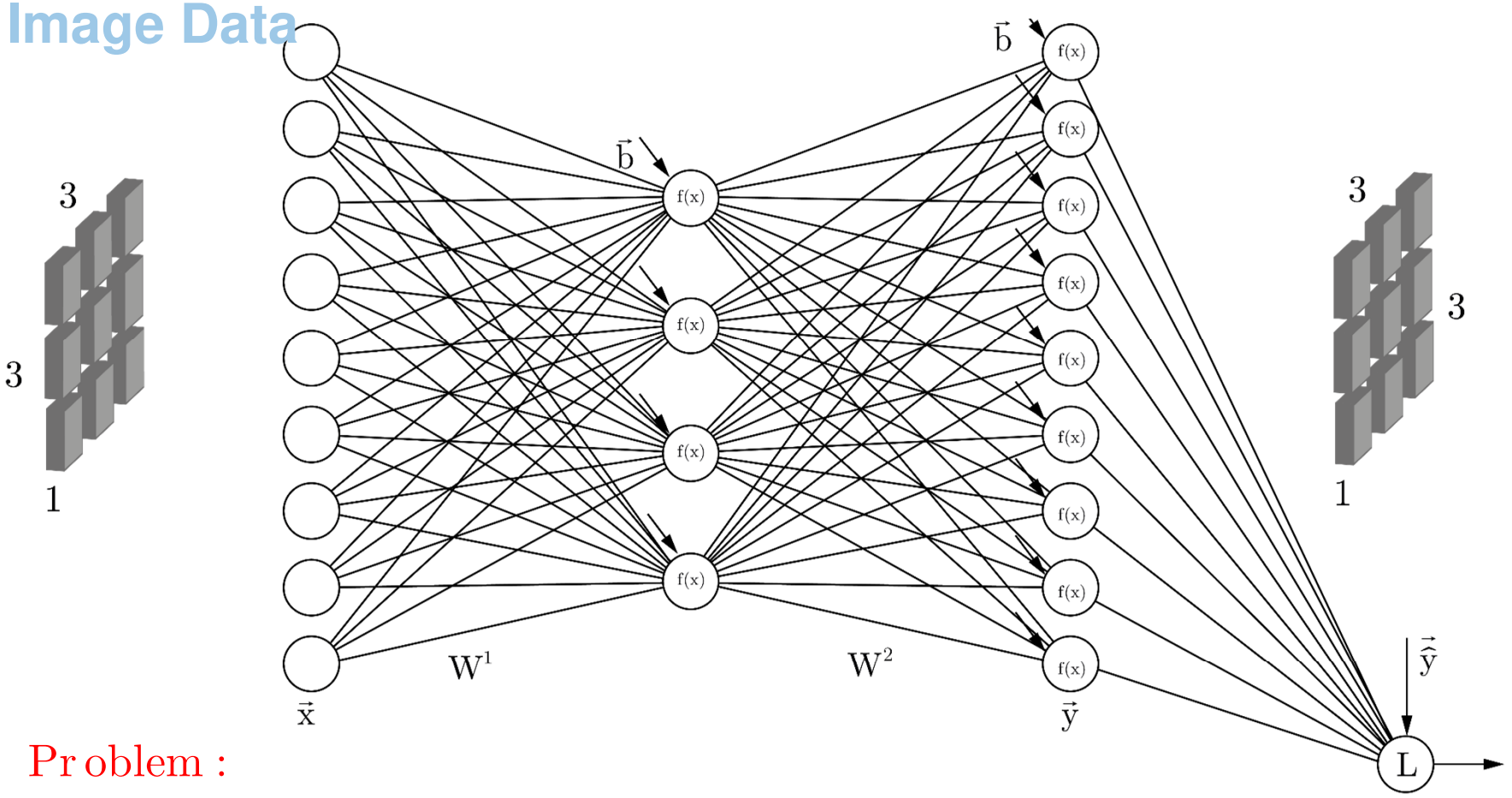
Here :

$$\vec{x} \cdot W^1$$

$$(1\times 2)\cdot(2\times 4) = (1\times 4)$$

Fully Connected Layer

Image Data

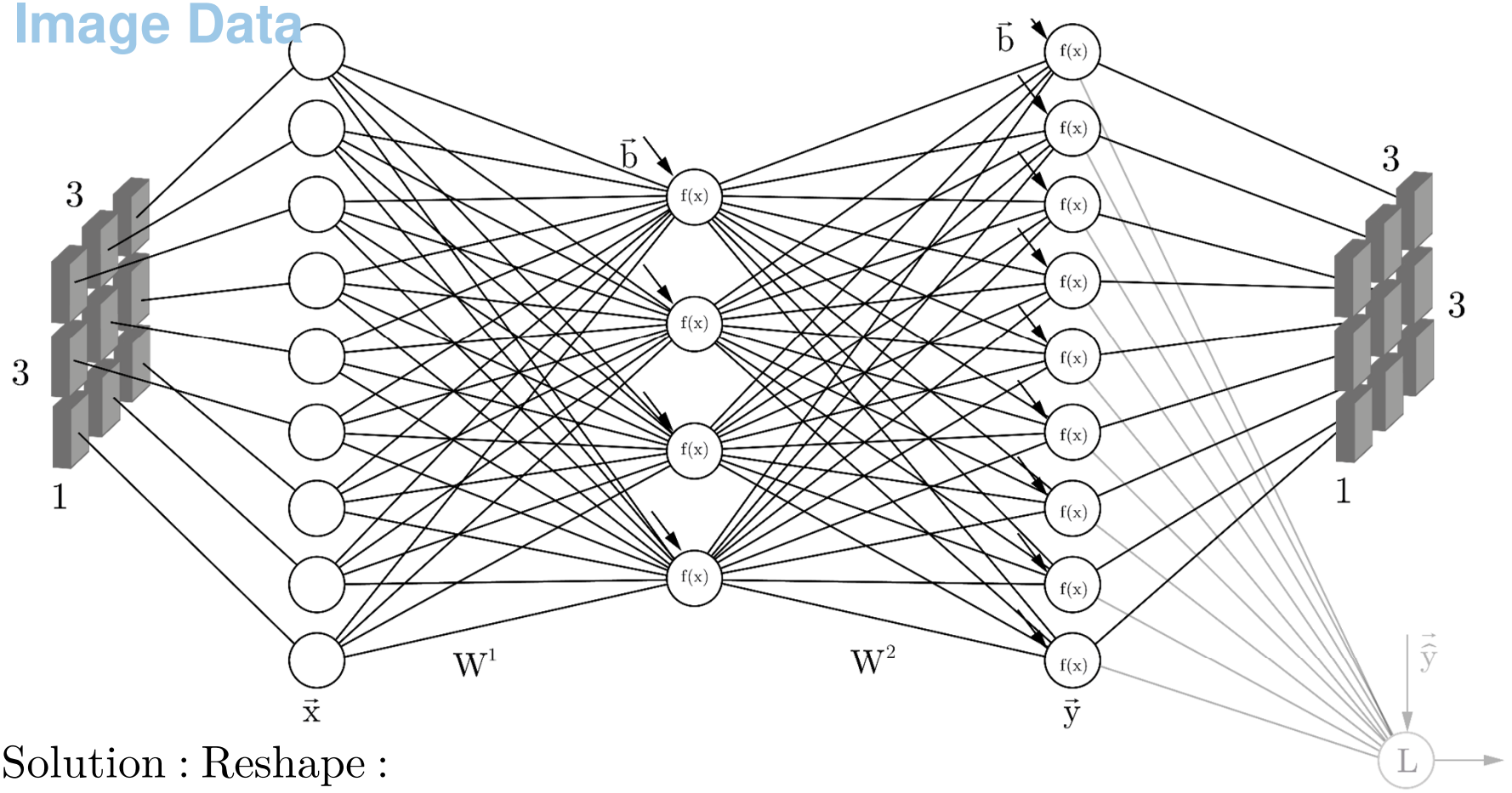


Problem :

$(1 \times 3 \times 3) \cdot (9 \times 4)$

Fully Connected Layer

Image Data



Solution : Reshape :

$(1 \times 3 \times 3) \rightarrow (1 \times 9)$

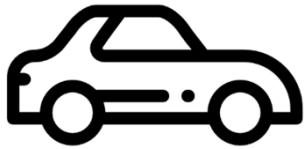
$\Rightarrow (1 \times 9) \cdot (9 \times 4)$

Deep Neural Networks

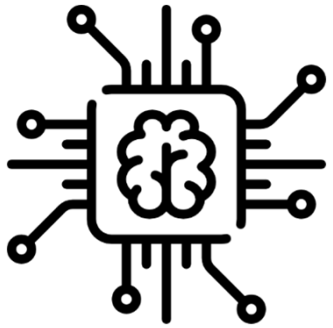
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

1. Chapter: Backpropagation
    - 1.1 Computational Graph
    - 1.2 Single Neuron
    - 1.3 Neural Chain
    - 1.4 Neural Network

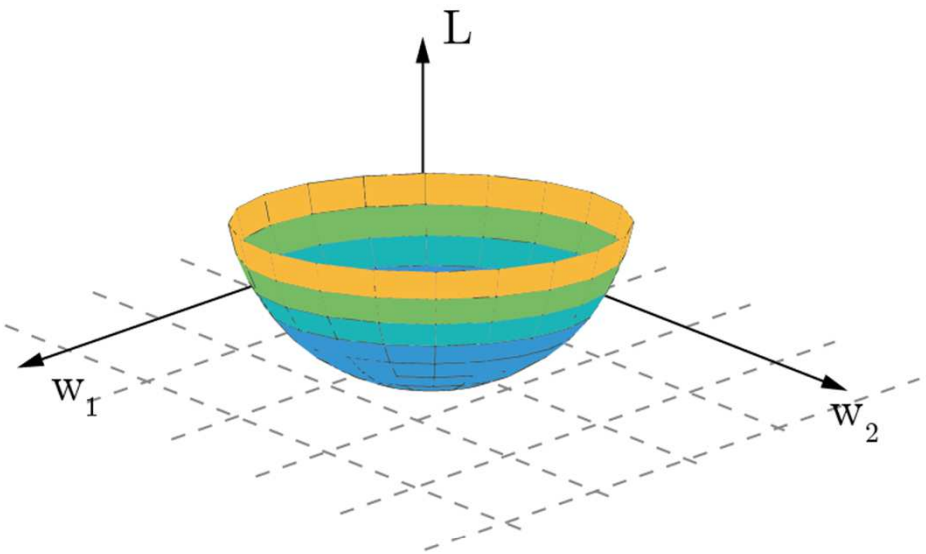


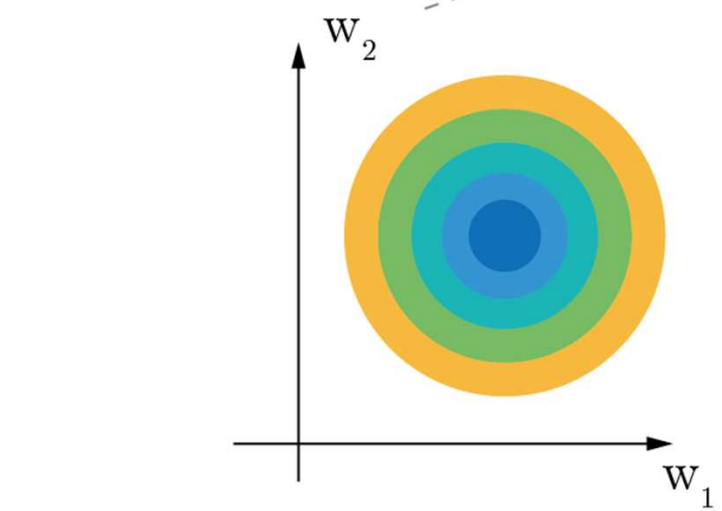
2. Chapter: Neuronal Networks
    - 2.1 Activation Functions
    - 2.2 Fully Connected Layer
    - 2.3 Batches
    - 2.3 Weight Inititalisation

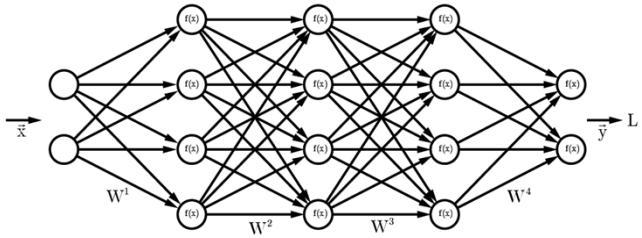


3. Chapter: Overview

Gradient Descent





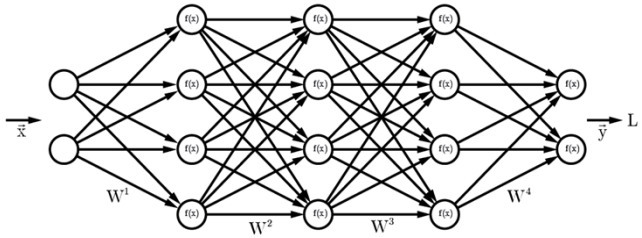


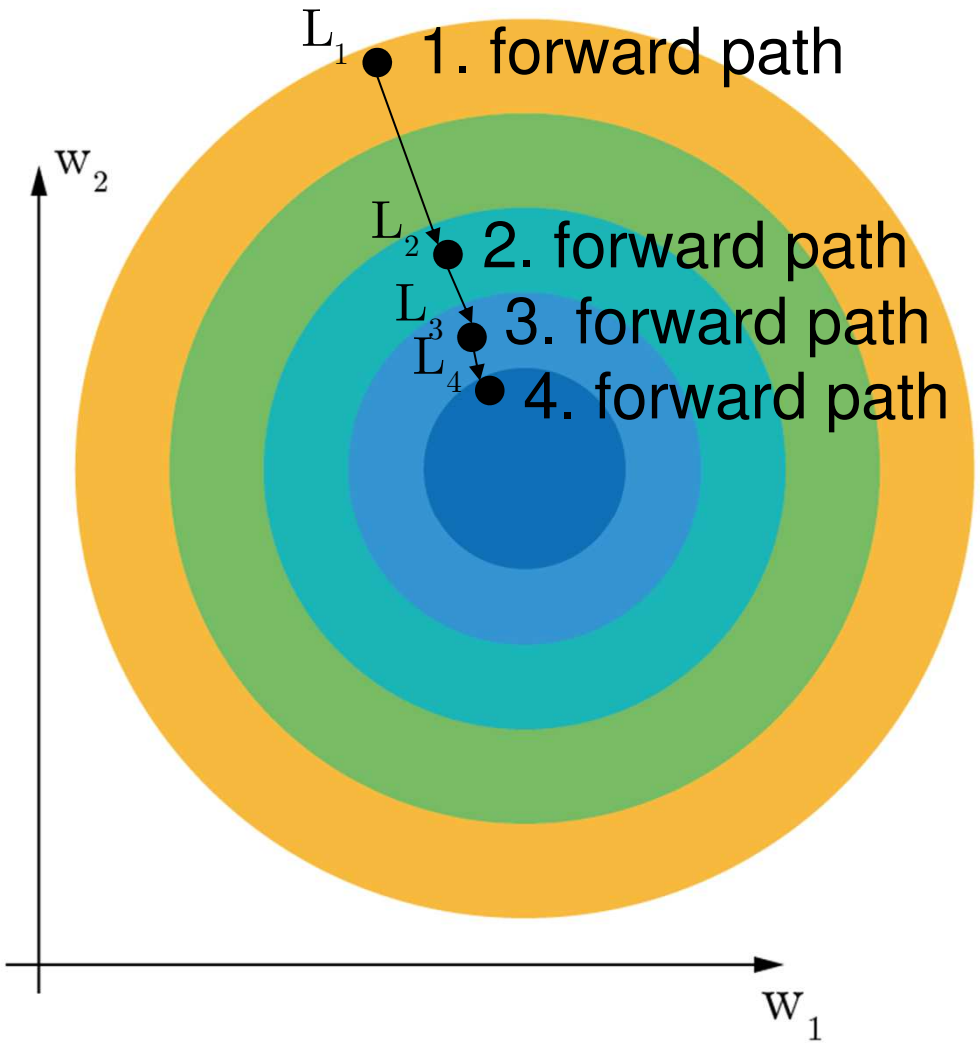
$$L(X) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Training:

for n < max_epochs:
- Forward path
- Backward path
- Update weights

Gradient Descent





$$L(X) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

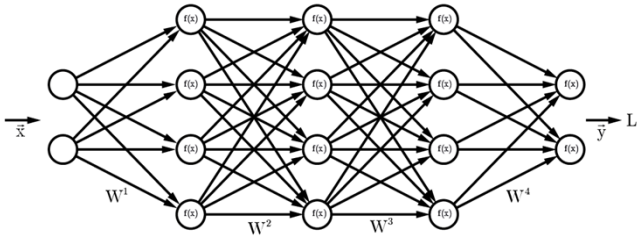
All the data

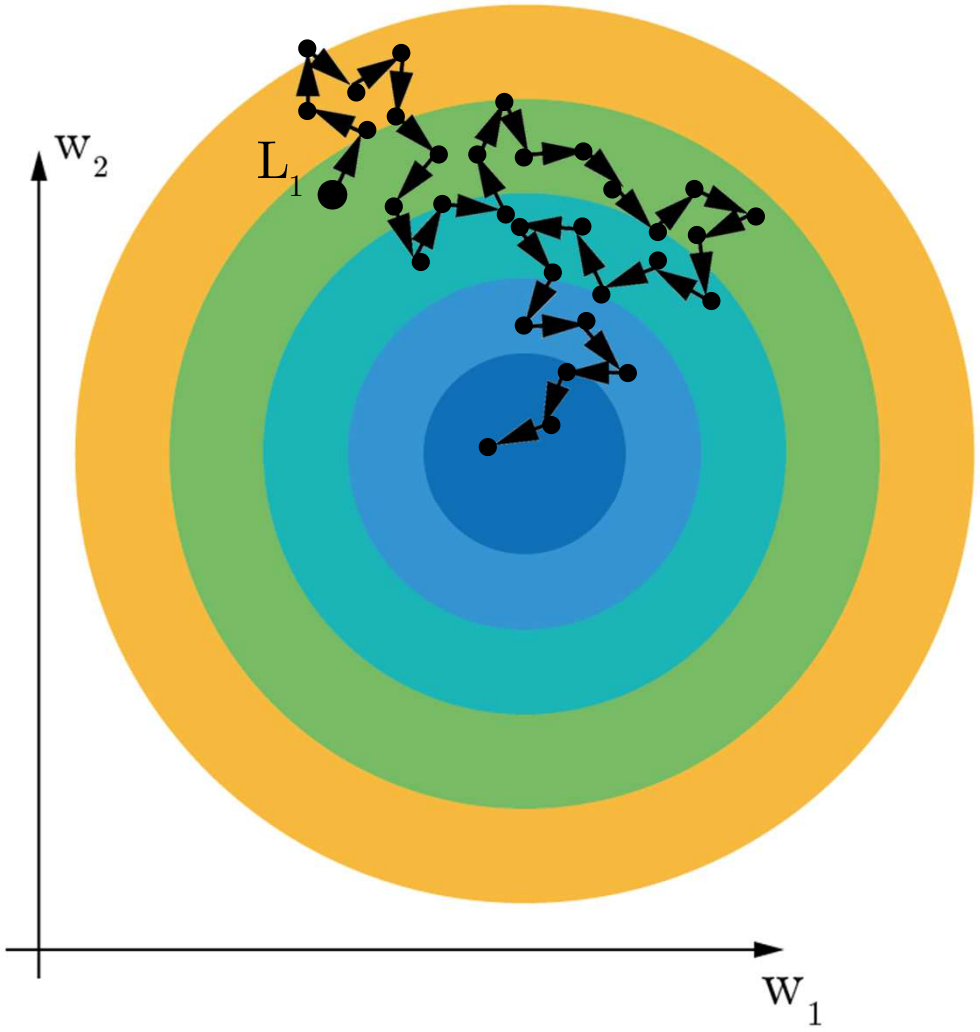
$$-\alpha \nabla L_W$$

3x compute time

Stochastic Gradient Decent

Alternative to Gradient Decent





Only use one random datapoint at a time

$\Rightarrow$ Reduces compute time per optimisation step

$\Rightarrow$ But finding local minimums takes longer

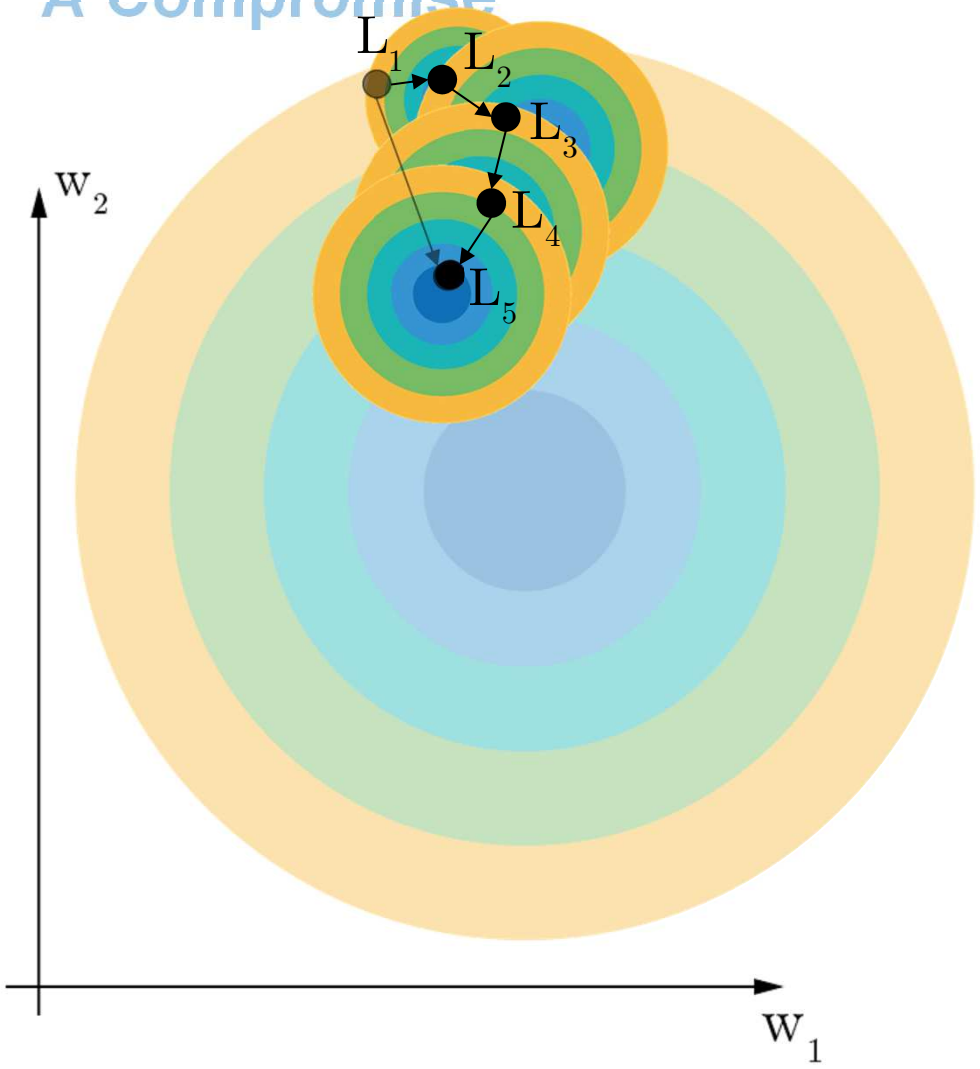
$$L(X) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

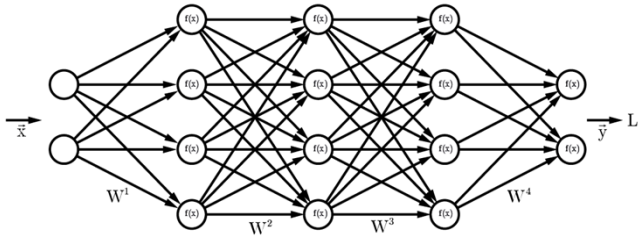
$$n = 1$$

$$\Rightarrow L(x) = (y - \hat{y})^2$$

Batch Gradient Descent

A Compromise





$$L(X) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

Split dataset in batches,

=> Trying to minimize global loss function with local cost functions

Dimensions with batch gradient descent

$$\vec{x}_2 = f\left(\underbrace{\begin{pmatrix} x_1 & x_2 \end{pmatrix}}_{\vec{x}} \cdot \underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{pmatrix}}_{W^1} + \vec{b} \right)$$

$$\vec{x} \in K^{1\times 2}$$
$$W_1 \in K^{2\times 4}$$
$$\vec{b} \in K^{1\times 4}$$

For batchsize n:

$$\begin{matrix} \vec{x}_{2,1} \\ \vec{x}_{2,2} \\ \vdots \\ \vec{x}_{2,n} \end{matrix} = f\left(\underbrace{\begin{matrix} \begin{pmatrix} x_1 & x_2 \end{pmatrix}_1 \\ \begin{pmatrix} x_1 & x_2 \end{pmatrix}_2 \\ \vdots \\ \begin{pmatrix} x_1 & x_2 \end{pmatrix}_n \end{matrix}}_{\vec{x}} \cdot \underbrace{\begin{matrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{pmatrix}_1 \\ \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{pmatrix}_2 \\ \vdots \\ \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{pmatrix}_n \end{matrix}}_{W^1} + \begin{matrix} \vec{b}_1 \\ \vec{b}_2 \\ \vdots \\ \vec{b}_n \end{matrix} \right)$$

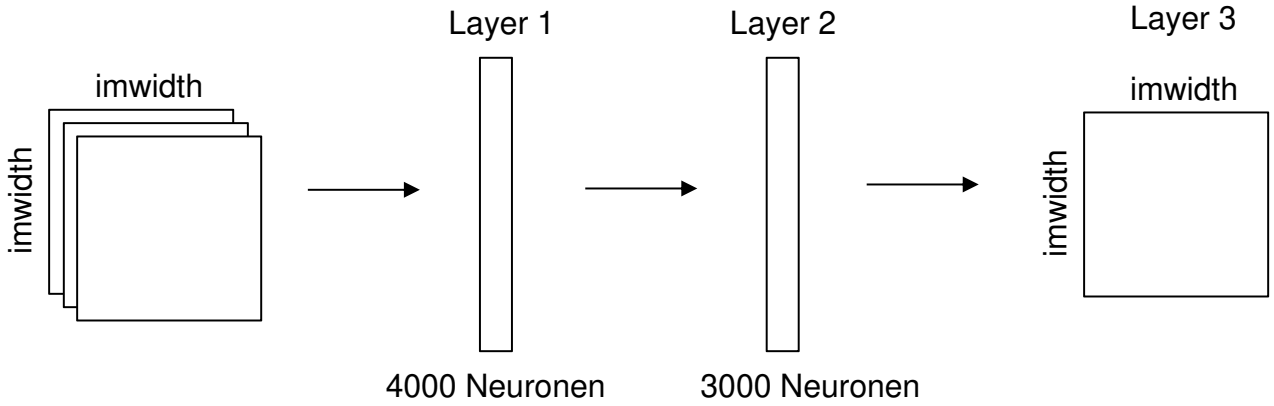
$$\vec{x} \in K^{n\times 1\times 2}$$
$$W_1 \in K^{n\times 2\times 4}$$
$$\vec{b} \in K^{n\times 1\times 4}$$

3D Matrix multiplcation
-> handled by tensorflow

Stochastic vs. Batch

- Stochastic training can miss local minimums because of the randomness of each input
- Takes longer in general
- Mini batches finds minimum quickly but may take more computational power

Codeexample Tensorflow:



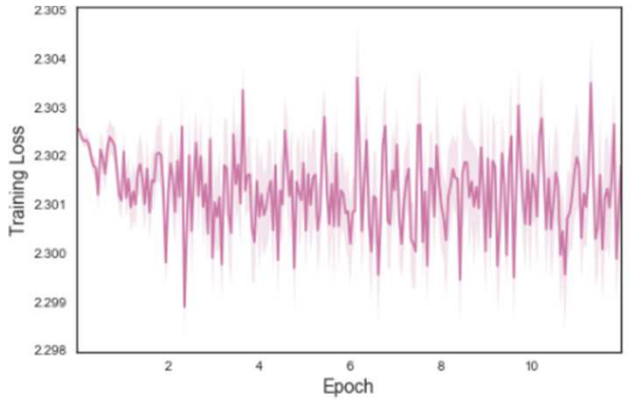

```
with tf.name_scope("Network"):
    with tf.name_scope("Layer1"):
        w1 = tf.Variable(tf.random_normal([batchsize,imwidth*imwidth*3,4000],mean = 0.0, stddev = 0.01))
        b1 = tf.Variable(tf.random_normal([batchsize,1,4000],mean = 0.0, stddev = 0.01))
        a1 = tf.matmul(x,w1)
        l1 = tf.nn.relu(tf.add(a1,b1))
    with tf.name_scope("Layer2"):
        w2 = tf.Variable(tf.random_normal([batchsize,4000,3000],mean = 0.0, stddev = 0.01))
        b2 = tf.Variable(tf.random_normal([batchsize,1,3000],mean = 0.0, stddev = 0.01))
        a2 = tf.matmul(l1,w2)
        l2 = tf.nn.relu(tf.add(a2,b2))
    with tf.name_scope("Layer3"):
        w3 = tf.Variable(tf.random_normal([batchsize,3000,imwidth*imwidth],mean = 0.0, stddev = 0.01))
        b3 = tf.Variable(tf.random_normal([batchsize,1,imwidth*imwidth],mean = 0.0, stddev = 0.01))
        a3 = tf.matmul(l2,w3)
        y_out = tf.sigmoid(tf.add(a3,b3))
```

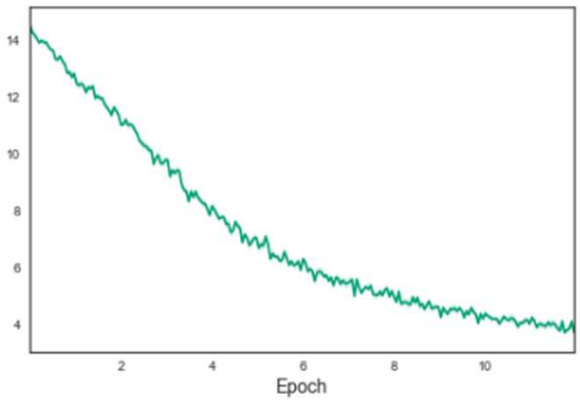
Weight Initialisation

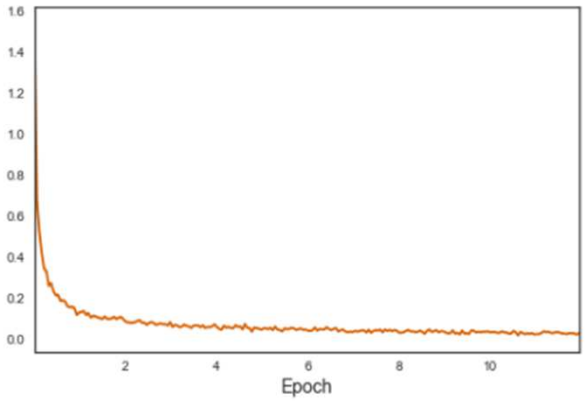
$$W_1 = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}$$

- **Init as 0**
- **Normal Distribution**
- **(0, $\sigma = 0.4$)**
- **Normal Distribution (0, $\sigma \sim \sqrt{\frac{2}{n_i}}$)**

n_i: Number of Neurons in previous layer







- **z.B.:**

$$W_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$W_1 = \begin{pmatrix} 0.01 & -0.50 & 0.14 \\ -0.10 & 0.11 & -0.06 \\ -0.33 & 0.47 & 0.32 \end{pmatrix}$$

$$W_1 = \begin{pmatrix} 1.20 & -1.50 & 1.4 \\ -1.2 & 0.97 & -0.6 \\ 0.63 & -0.47 & 0.52 \end{pmatrix}$$

$n_i = 2$
=> $\sigma = 1$

https://intoli.com/blog/neural-network-initialization/

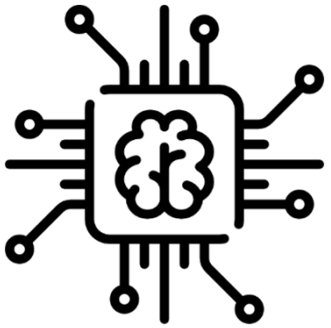
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

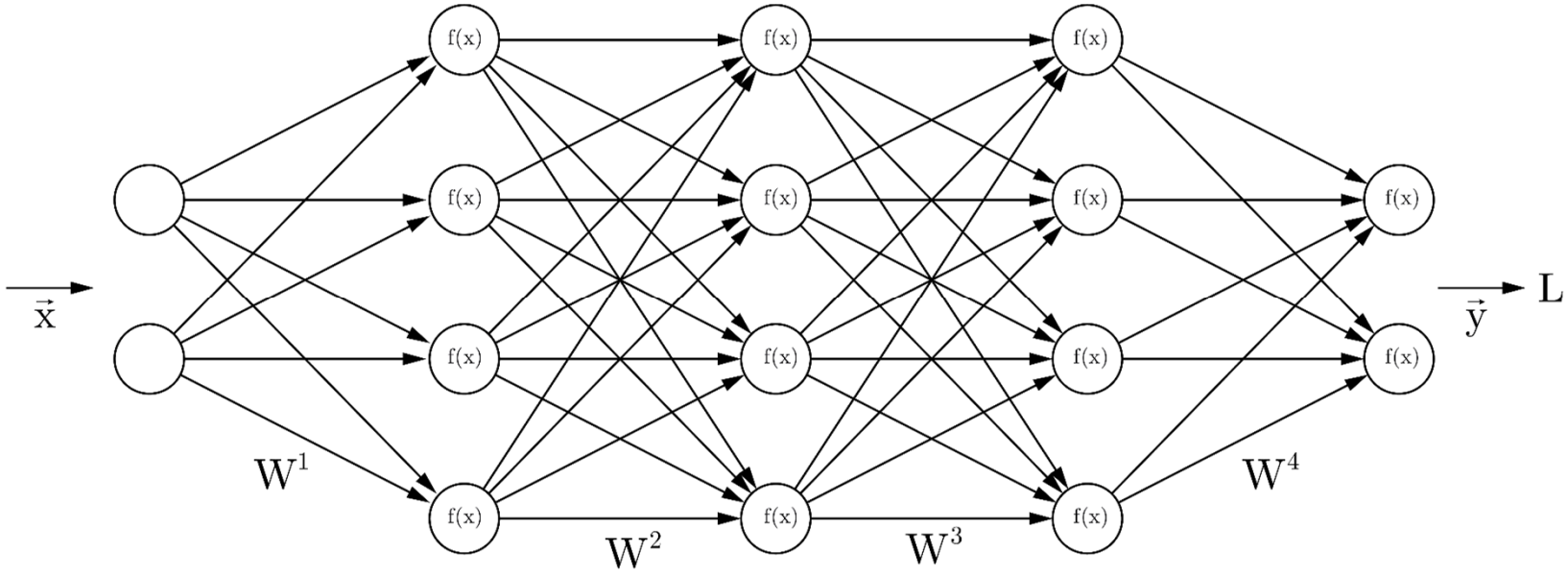
1. Chapter: Backpropagation
    - 1.1 Computational Graph
    - 1.2 Single Neuron
    - 1.3 Neural Chain
    - 1.4 Neural Network

2. Chapter: Neuronal Networks
    - 2.1 Activation Functions
    - 2.2 Fully Connected Layer
    - 2.3 Batches
    - 2.3 Weight Inititalisation



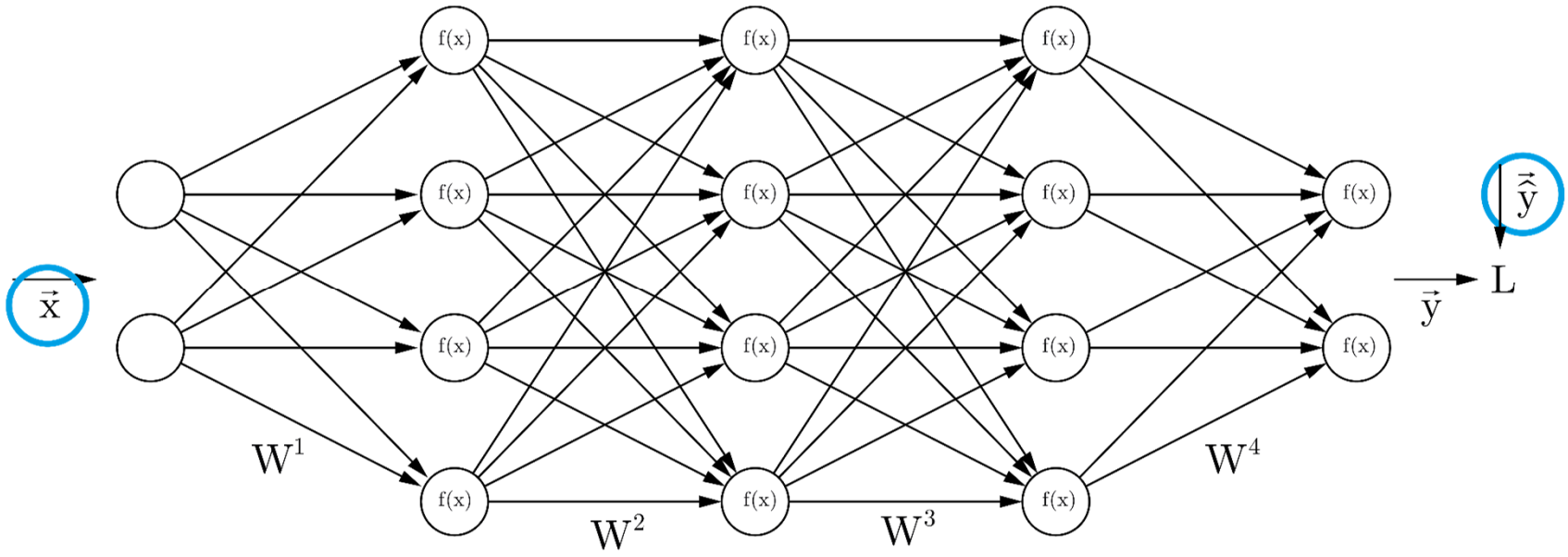
➡ 3. Chapter: Overview

Training a Neural Network



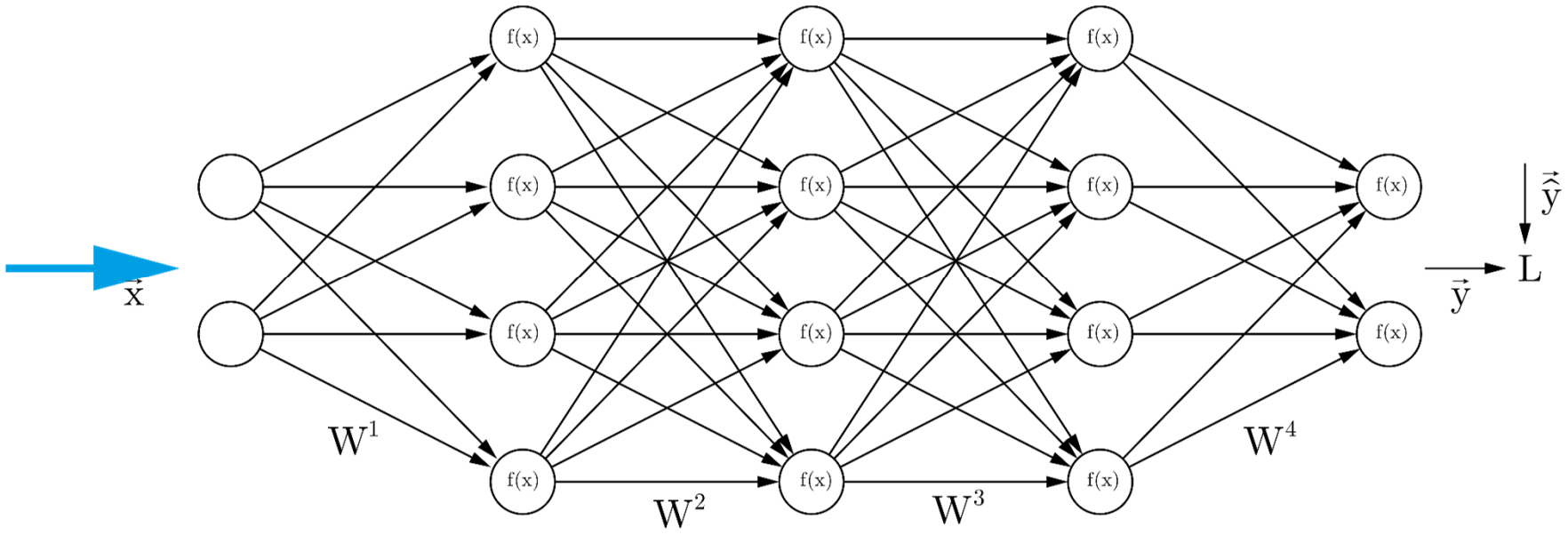
1. Preparing input data
2. Prepare datapipeline to get the data into the network
3. Define the network
4. Initialize all variables
5. Train the network
6. Supervise the training

Training a Neural Network



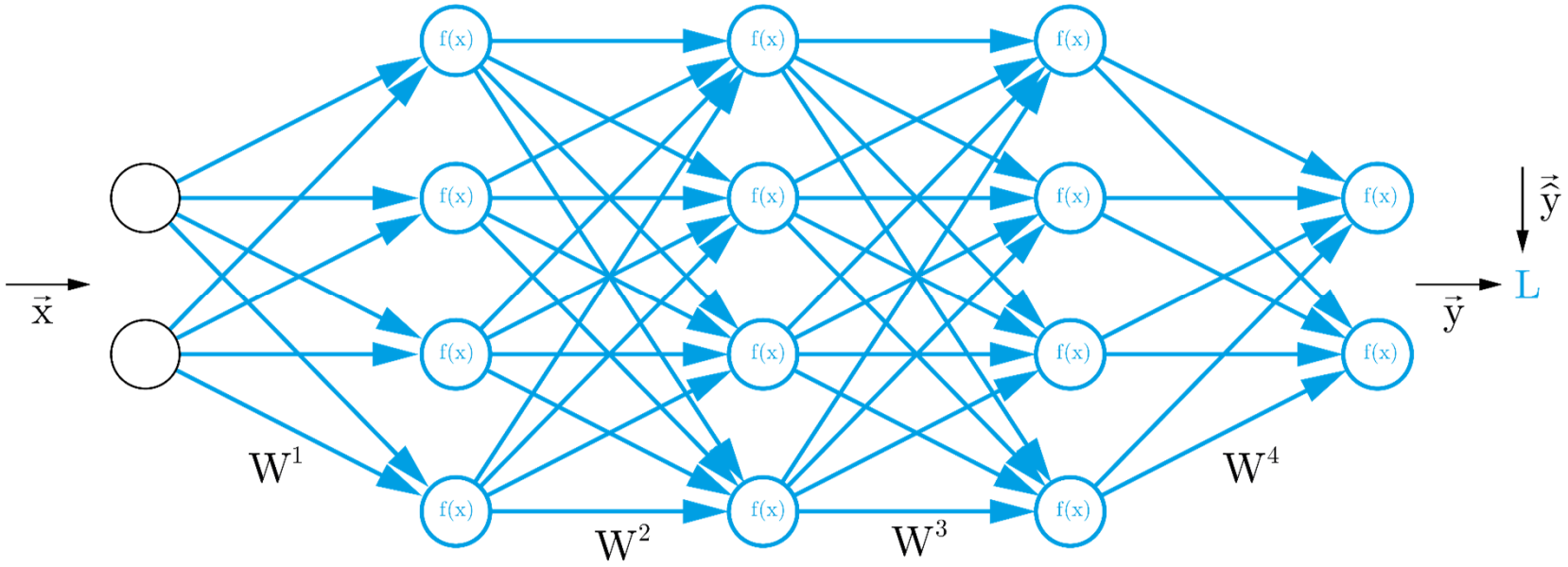
1. **Preparing input data**
2. Prepare datapipeline to get the data into the network
3. Define the network
4. Initialize all variables
5. Train the network
6. Supervise the training

Training a Neural Network



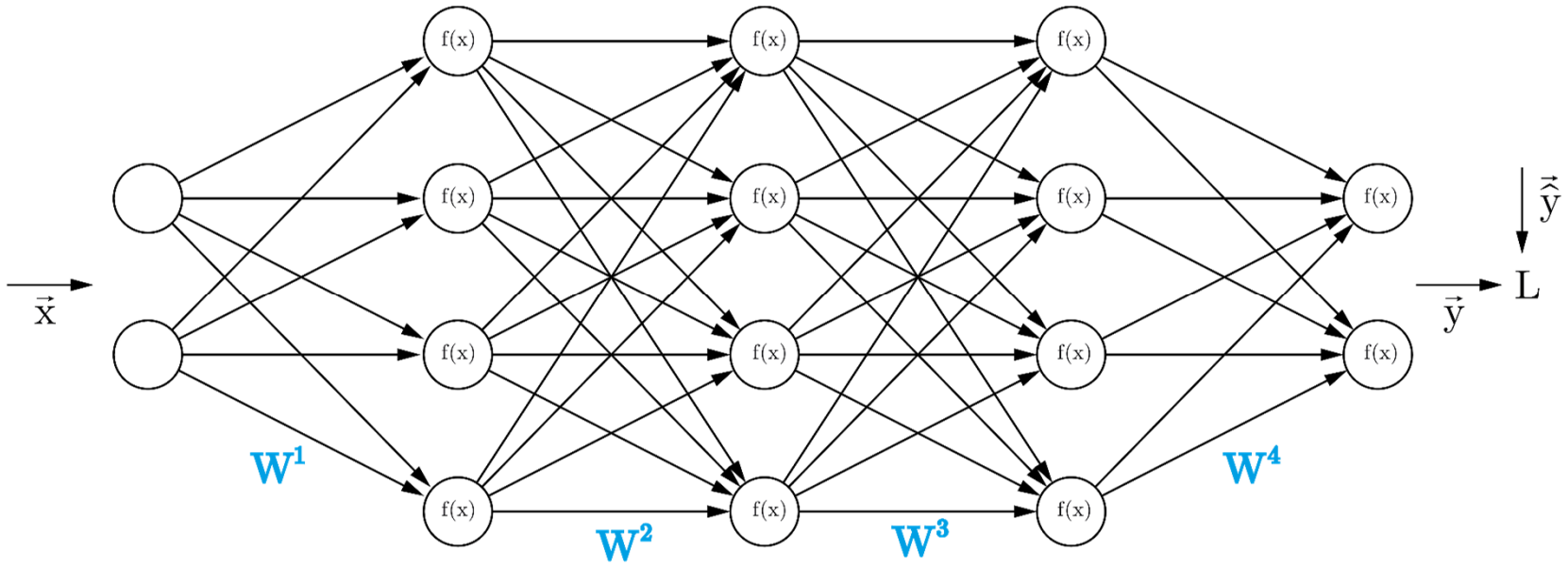
1. Preparing input data
2. **Prepare datapipeline to get the data into the network**
3. Define the network
4. Initialize all variables
5. Train the network
6. Supervise the training

Training a Neural Network



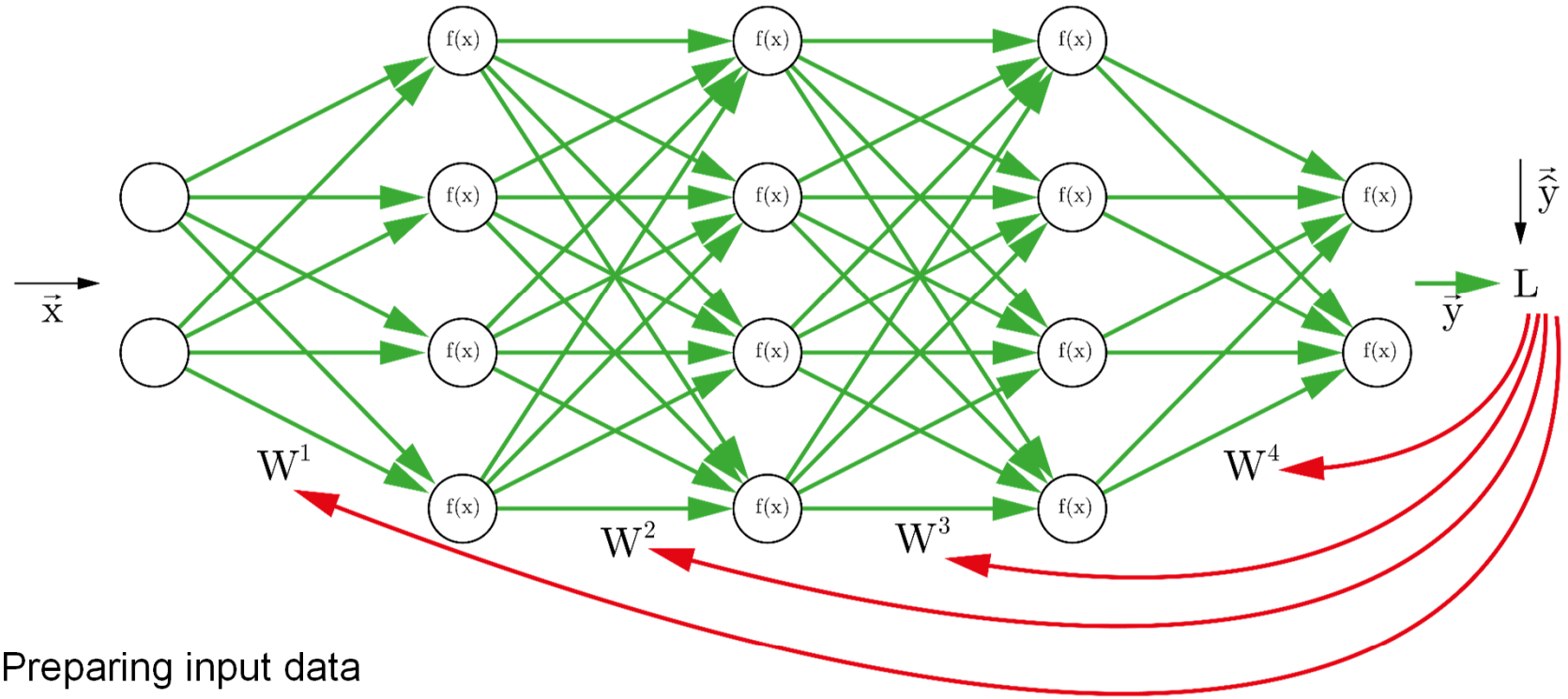
1. Preparing input data
2. Prepare datapipeline to get the data into the network
3. **Define the network**
4. Initialize all variables
5. Train the network
6. Supervise the training

Training a Neural Network



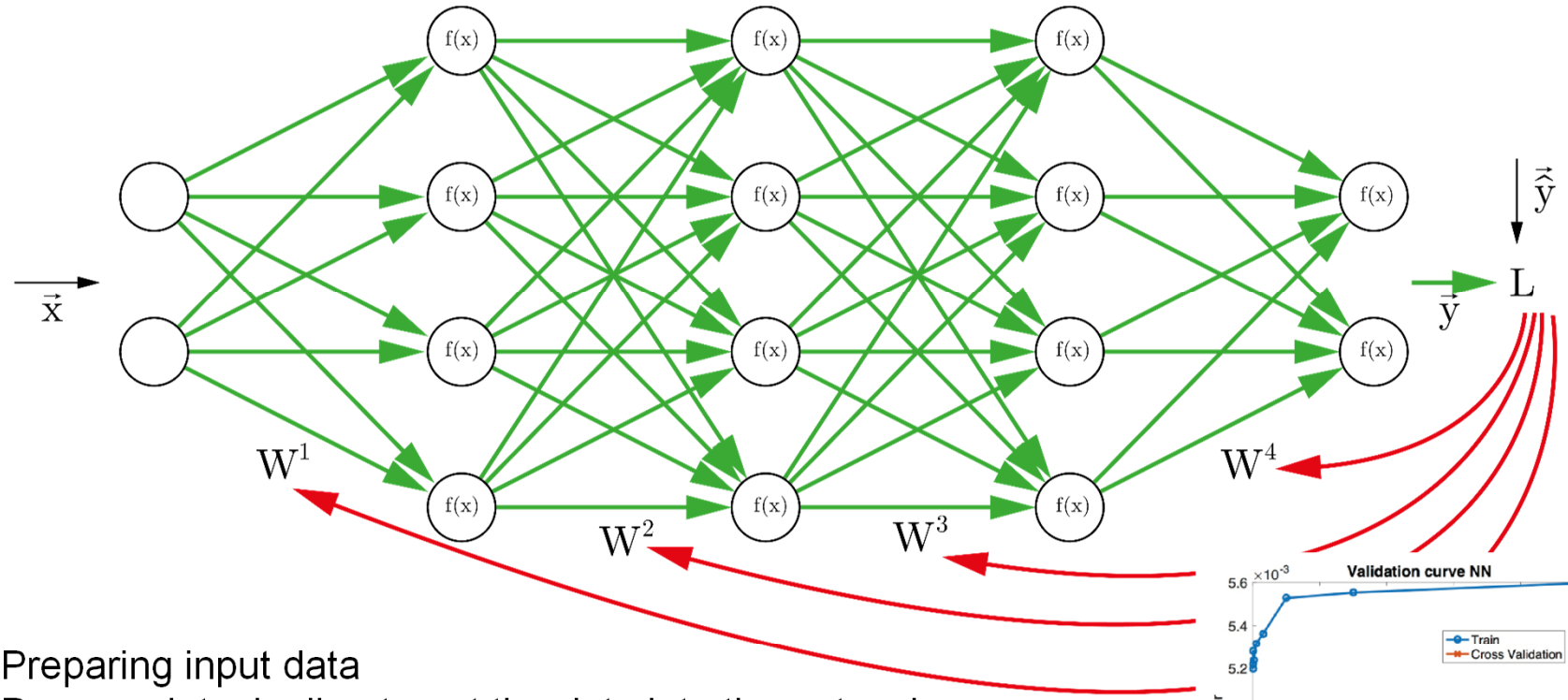
1. Preparing input data
2. Prepare datapipeline to get the data into the network
3. Define the network
4. **Initialize all variables**
5. Train the network
6. Supervise the training

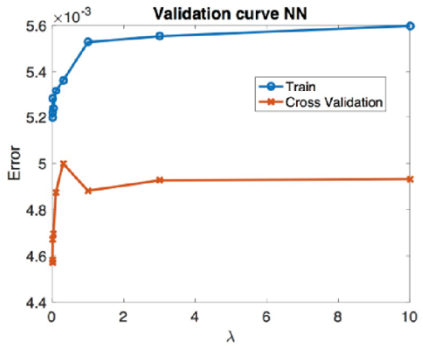
Training a Neural Network



1. Preparing input data
2. Prepare datapipeline to get the data into the network
3. Define the network
4. Initialize all variables
5. **Train the network**
6. Supervise the training

Training a Neural Network





1. Preparing input data
2. Prepare datapipeline to get the data into the network
3. Define the network
4. Initialize all variables
5. Train the network
6. **Supervise the training**

Summary

What we learned today:

Using computational graphs to calculate local gradients in any mathematical function
Backpropagating local gradients to calculate weight updates
Most common activation functions and their use cases
Use of batches for training neural networks
Initialize weights correctly
Steps to take in training a neural network

End

Kommentarfolie

*Dies ist eine Kommentarfolie.

Wir nutzen diese Folien, um zusätzeliche Inhalte zu erläutern (Beispielsweise größere Rechnungen, Zusatztexte in langer Prosa etc.)

Diese Folie wird NICHT in der Vorlesung gezeigt, sondern auskommentiert

Diese Folie kommt in das Skript für die Studenten und ist Zusatzinformation.*