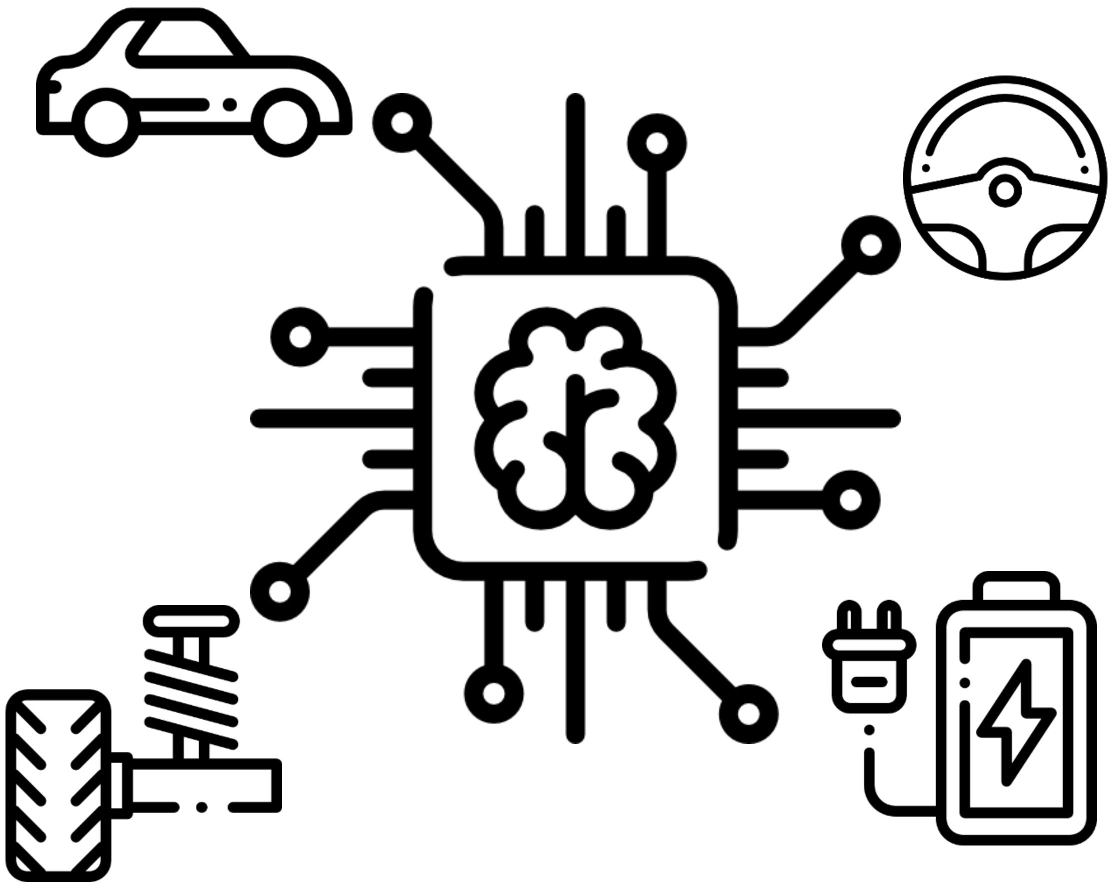
Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Artificial Intelligence in Automotive Technology

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



Lecture Overview

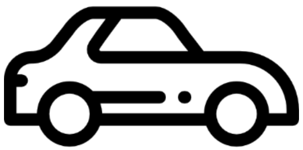
Einführung: Vorlesung 18.10.2018 – Betz Johannes	**6 Wegfindung: Von British Museum bis A*** 29.11.2018 – Lennart Adenaw	**11 Reinforcement Learning** 17.01.2019 – Christian Dengler
1 Einführung: Künstliche Intelligenz 18.10.2018 – Betz Johannes	**Ü6:** 29.11.2018 – Lennart Adenaw	**Ü11** 17.01.2019 – Christian Dengler
Ü1: 18.10.2018 – Betz Johannes	**7 Einführung: Neural Networks** 06.12.2018 – Lennart Adenaw	**12 AI-Development** 24.01.2019 – Johannes Betz
2 Grundlagen: Computer Vision 25.10.2018 – Betz Johannes	**Ü7** 06.12.2018 – Lennart Adenaw	**Ü12** 24.01.2019 – Johannes Betz
Ü2: 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	**Ü8** 13.12.2018 – Jean-Michael Georg	
Ü3: 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	**Ü9** 20.12.2018 – Jean-Michael Georg	
Ü4: 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	**Ü10** 10.01.2019 – Christian Dengler	

Introduction: Artificial Neural Networks

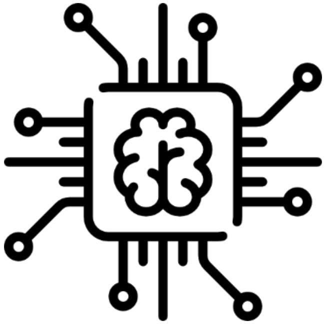
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Lennart Adenaw, M. Sc.)

Agenda

1. Chapter: Introduction



2. Chapter: Towards Artificial Neurons
   - 2.1 Linear Regression
   - 2.2 Gradient Descent
   - 2.3 The Neuron

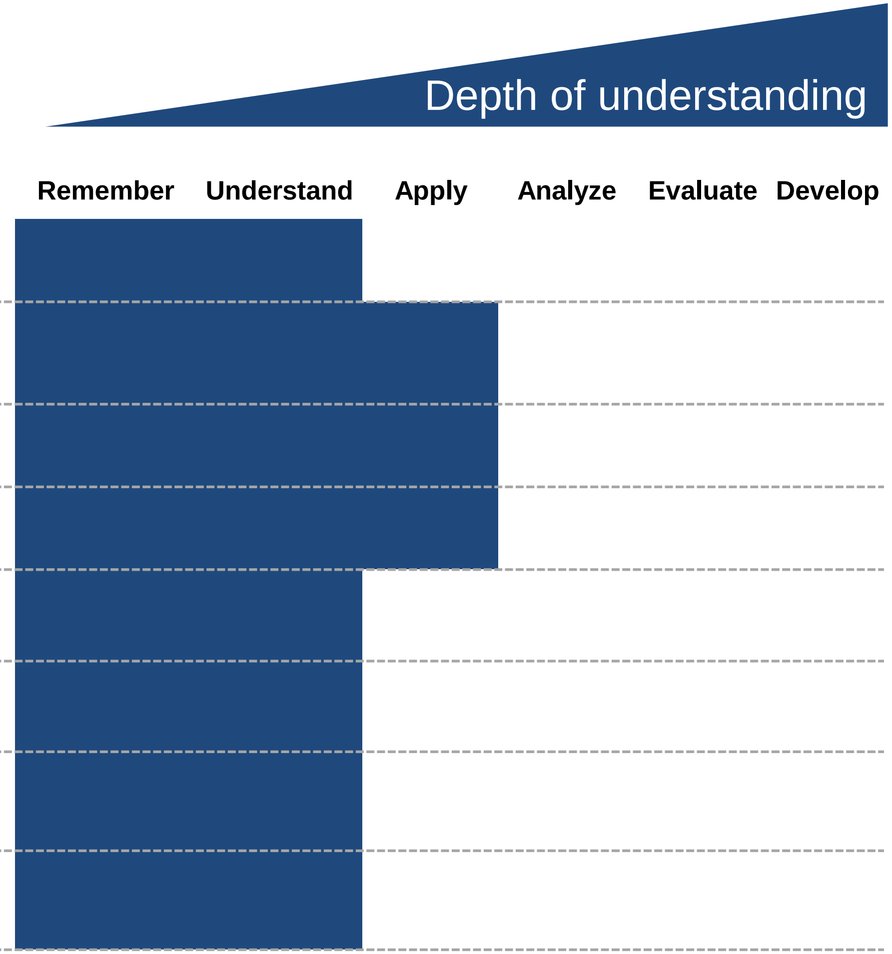


3. Chapter: Multilayer Networks
   - 3.1 Functional Completeness
   - 3.2 MNIST Example

4. Chapter: Summary

Objectives for Lecture 7: Introduction to Neural Nets

After the lecture you are able to…

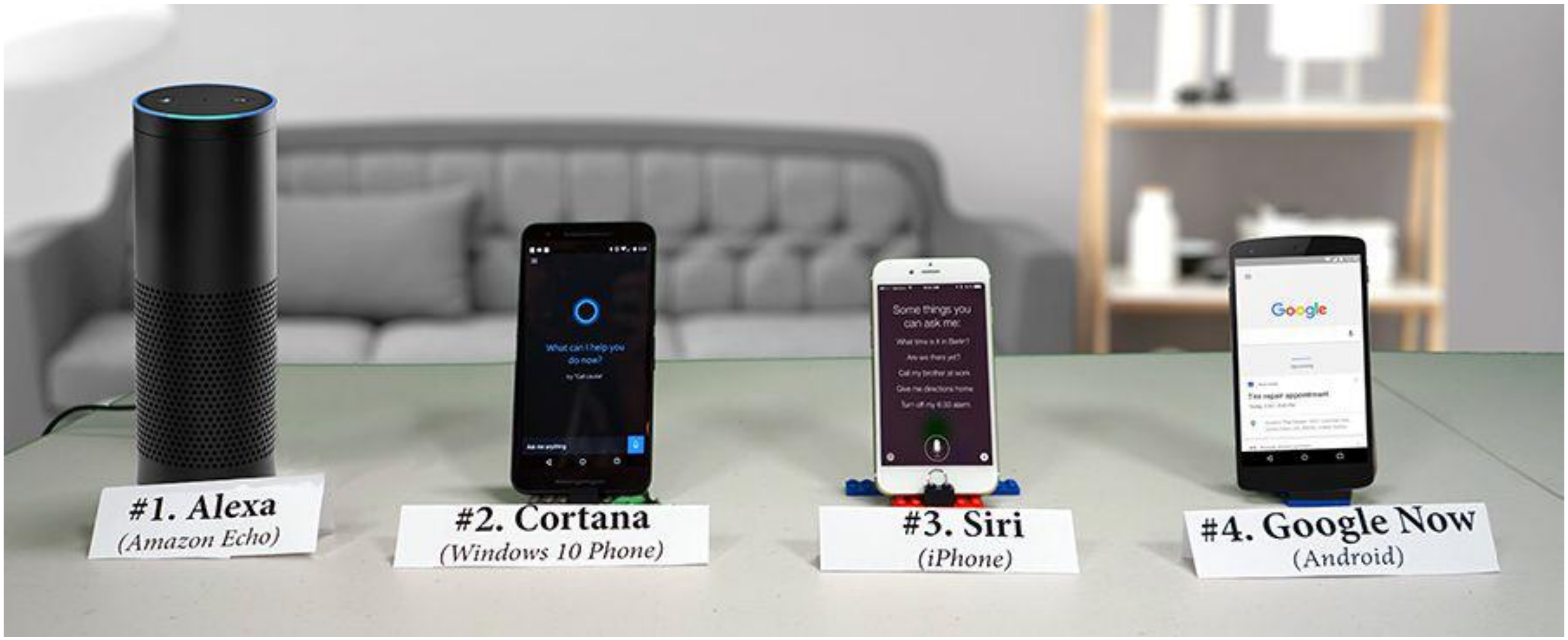


- … understand and explain what an artificial neuron is
- … draw graphical representations of artifical neurons
- … update the weights of a neuron using Gradient Descent
- … understand and solve simple regression and classification tasks using a single artificial neuron
- … understand how multiple artificial neurons form a neural network
- … explain functional completeness of simple neural networks
- … understand simple multi-layer architectures
- … remember and understand basic neural network vocabulary

Introduction
Neural Nets

Image to Text Translation

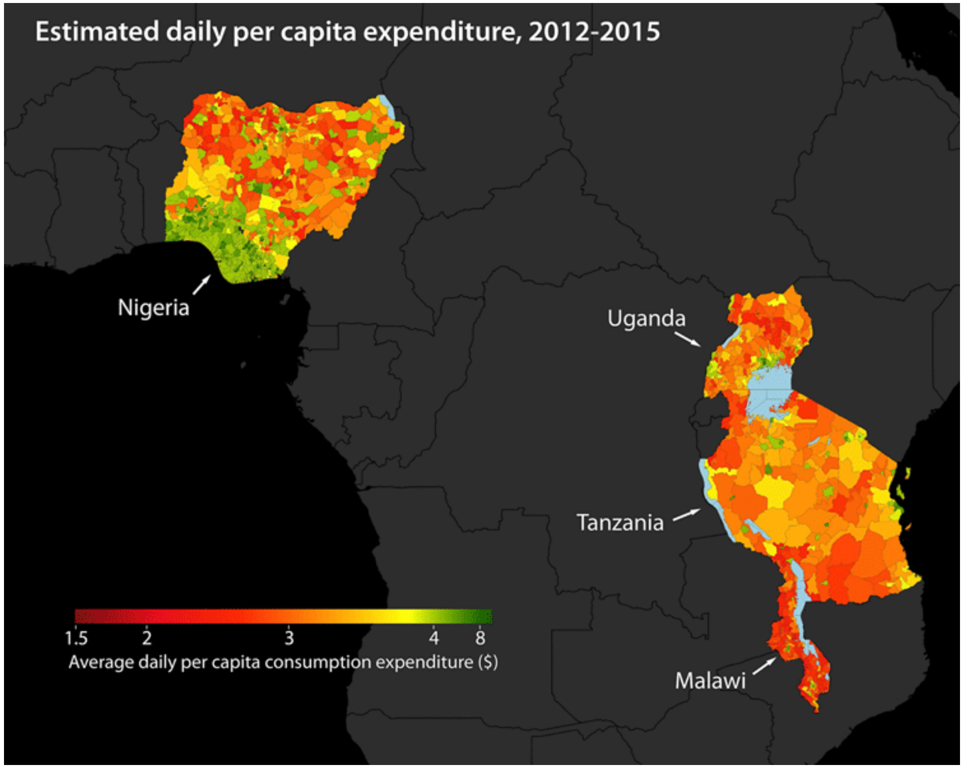
Introduction
Neural Nets



Speech Recognition
Speech Segmentation
Text-to-Speech

Introduction

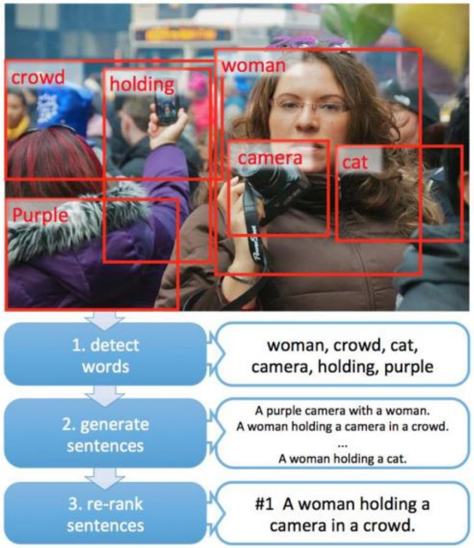
Neural Nets



Using Machine Learning to Map Poverty from Satellite Imagery

Introduction
Neural Nets

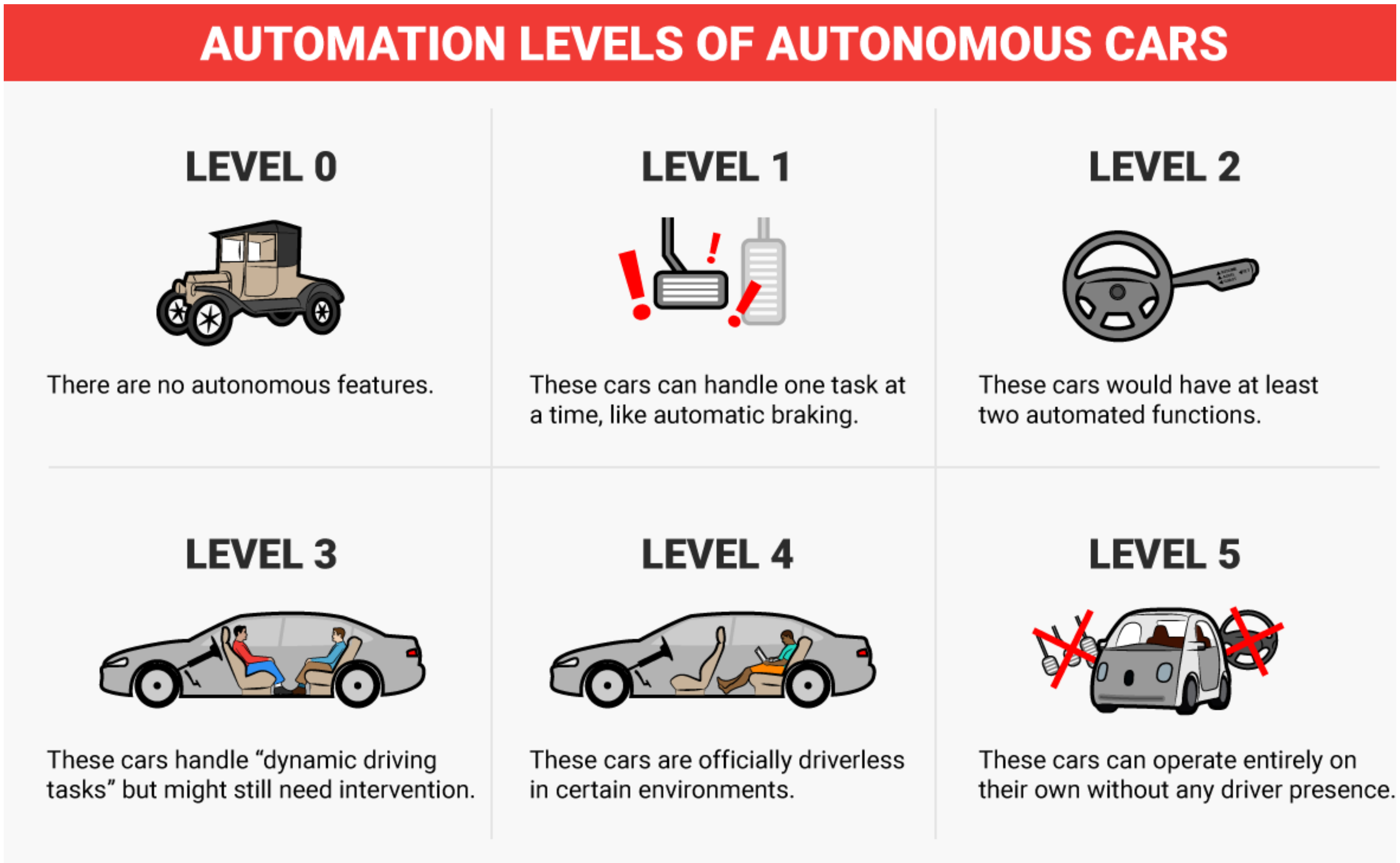
Image Colorization



Caption Generation

Artistic Style Transfer

Introduction
Neural Nets

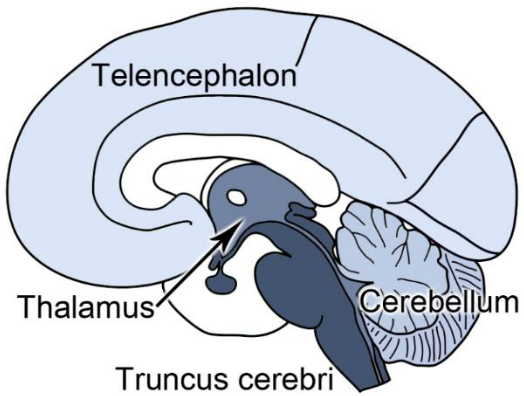


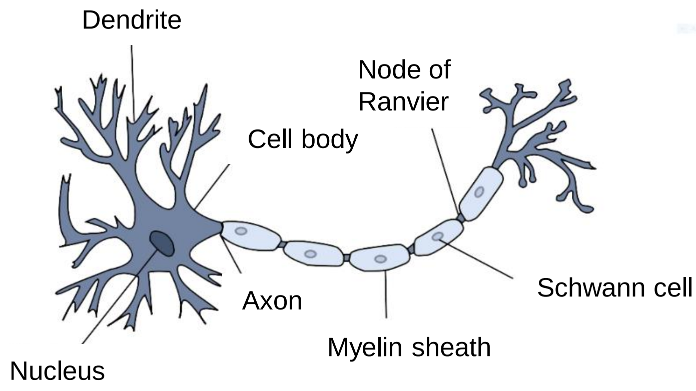
Introduction
Neural Nets

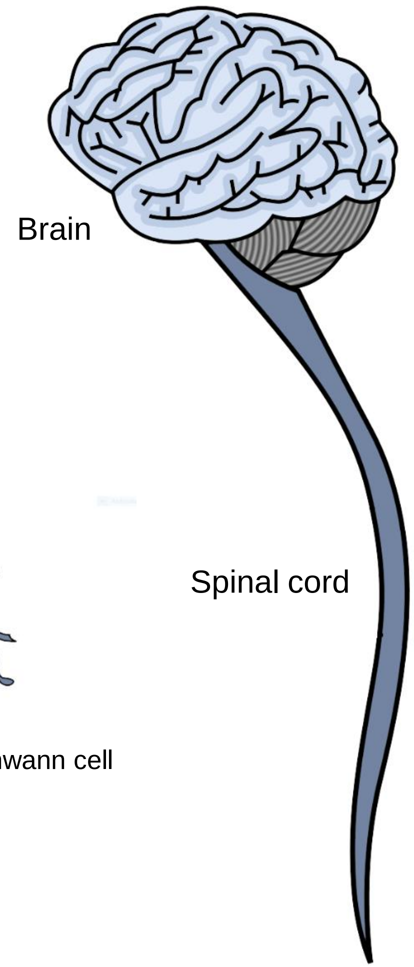
What is the similarity between these tasks?
How can one general approach fit them all?

Introduction

How to introduce Neural Nets



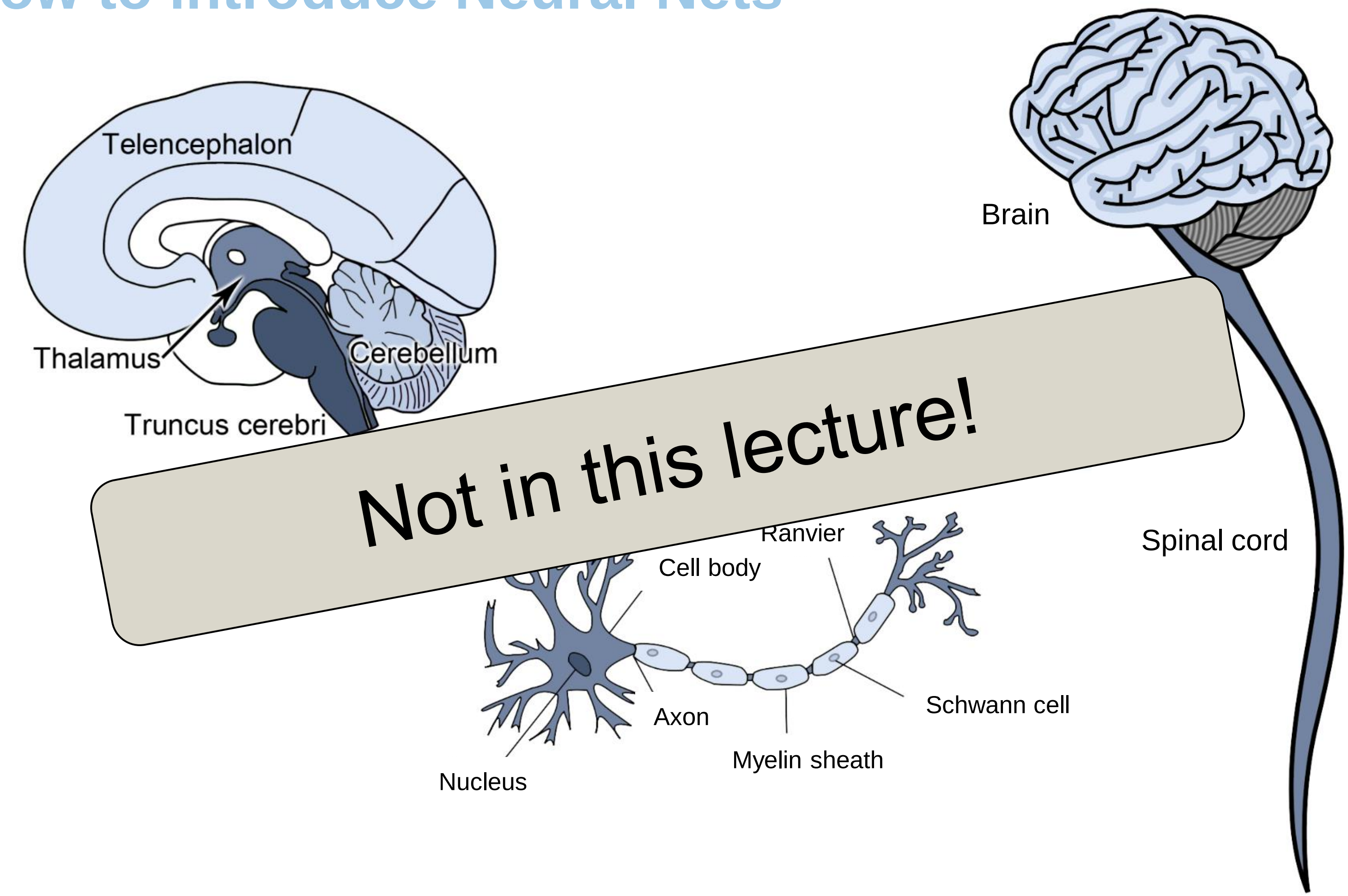




http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-1col-dkrieselcom.pdf

Introduction

How to introduce Neural Nets

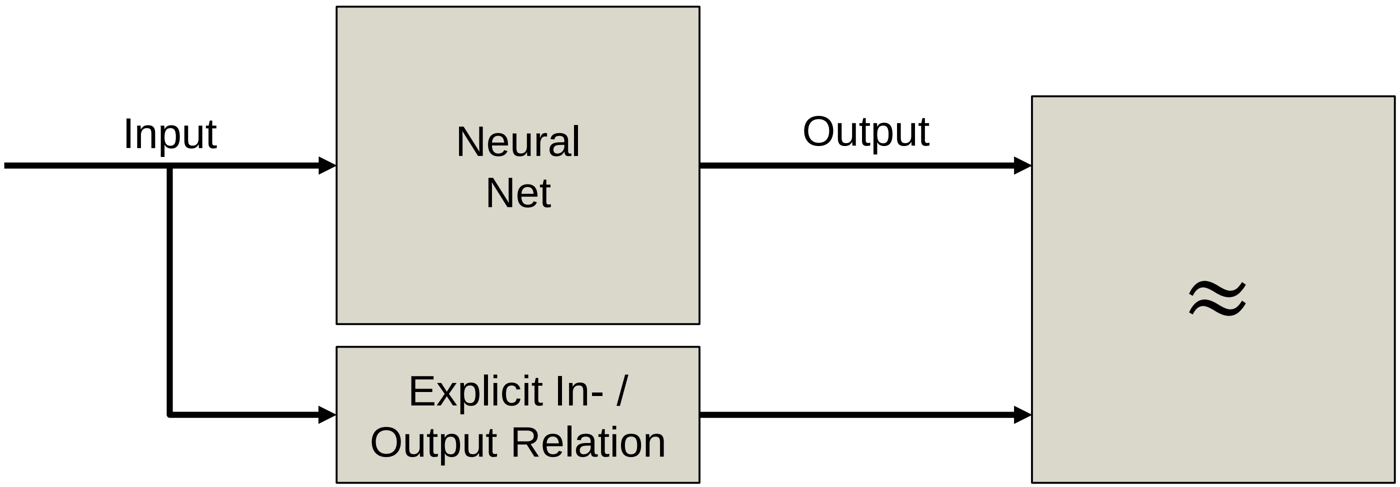


Not in this lecture!

http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-1col-dkrieselcom.pdf

Introduction
Universal Approximation Theorem

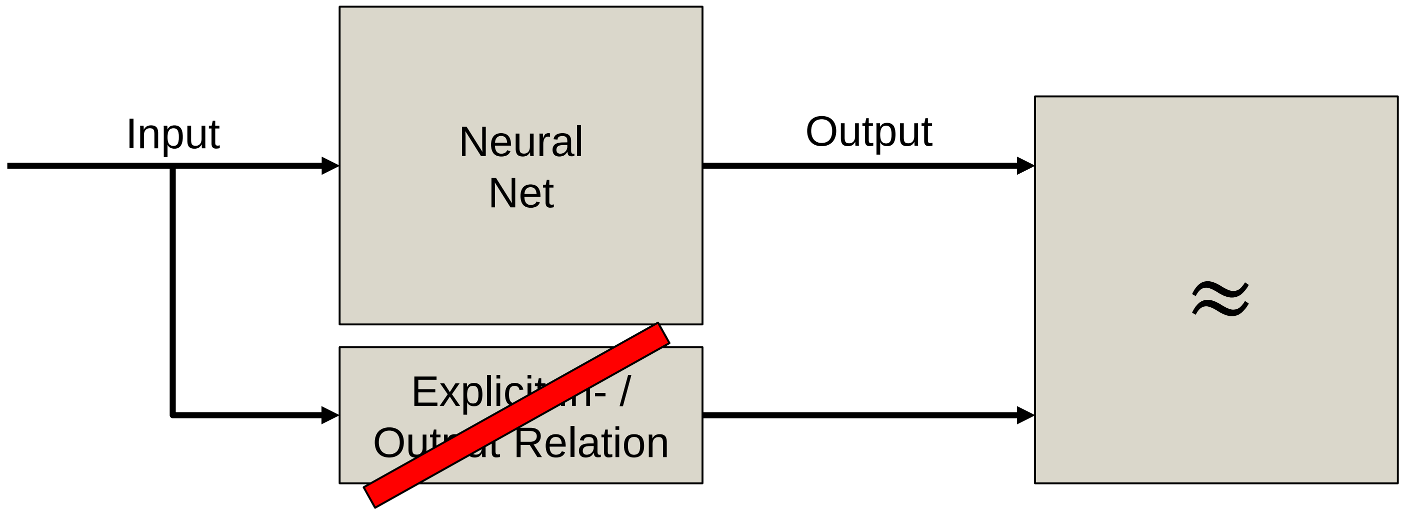
Neural Nets are Universal Approximators:



Introduction
Universal Approximation Theorem

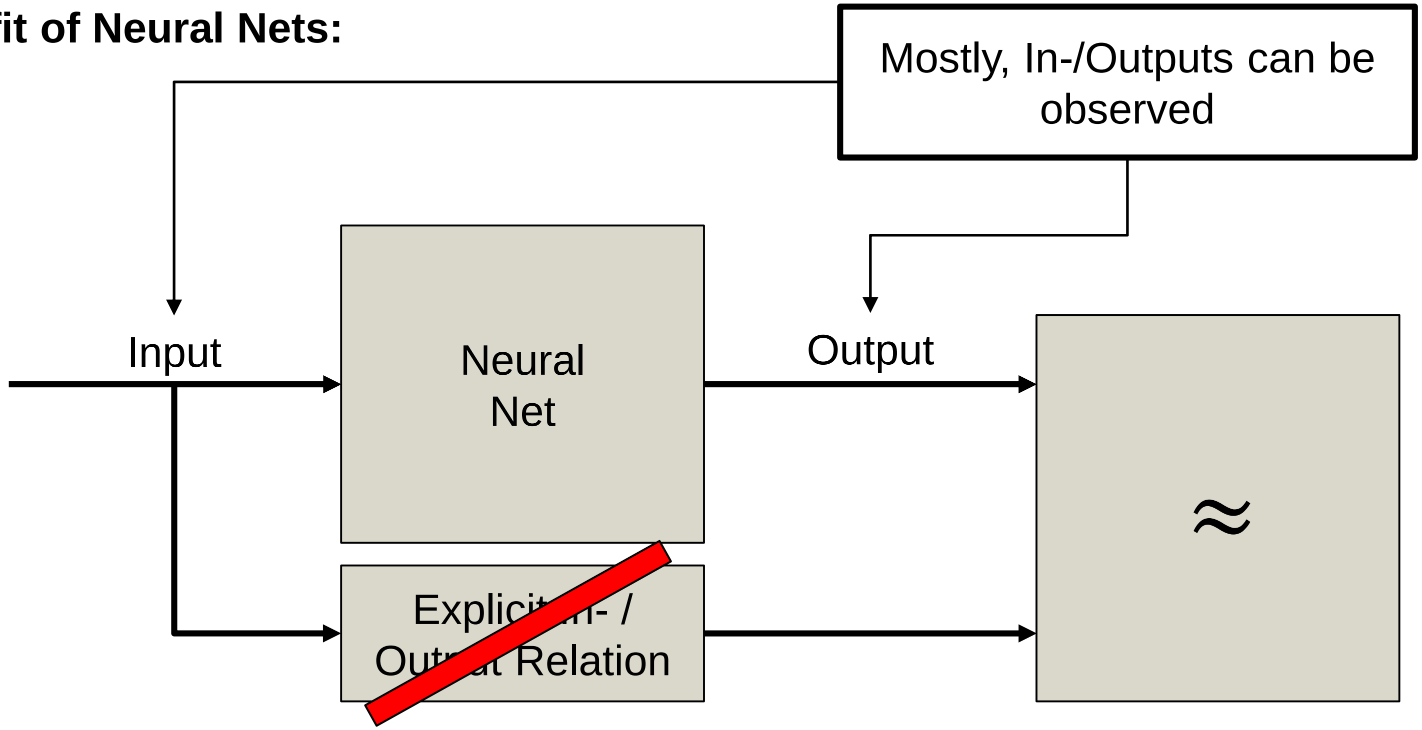
Benefit of Neural Nets:

Often, no explicit In- / Output Relation is known!



Introduction
Universal Approximation Theorem

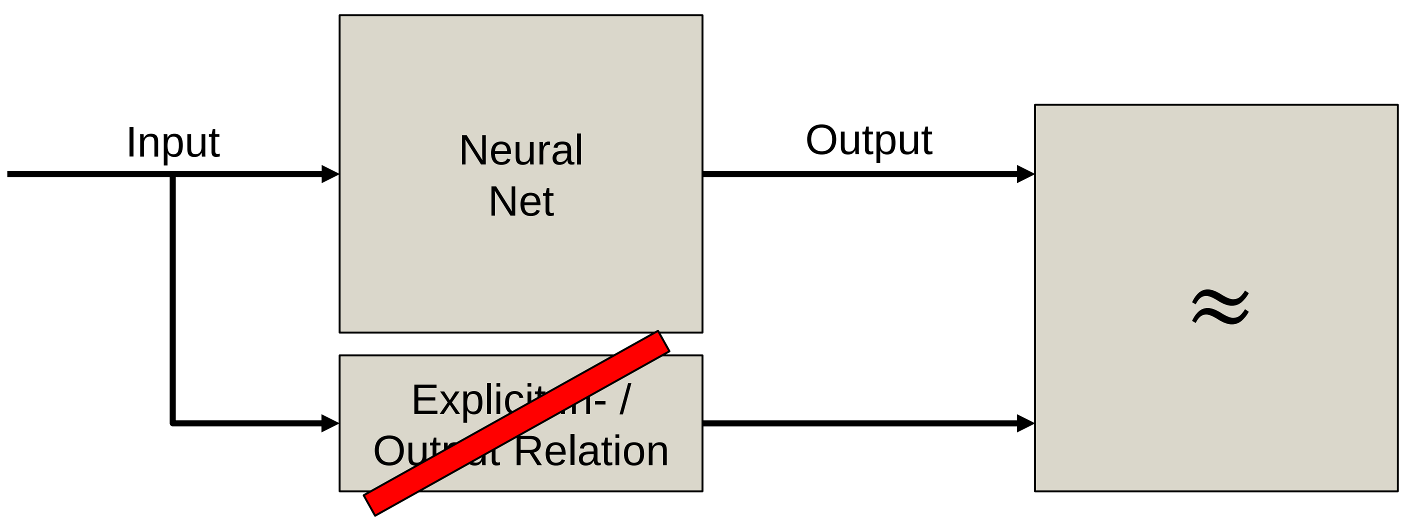
Benefit of Neural Nets:



Introduction
Universal Approximation Theorem

Benefit of Neural Nets:

Neural Nets can learn from observations



⇨ (Supervised) Machine Learning

Additional Slides

The Universal Approximation Theorem for ANN shows that even relatively simple ANN can appoximate any analytic function with arbitrary accuracy. Even though the Universal Approximation Theorem does not proof that the necessary parameters and architectures of the desired ANN can be found easily or by the means of finite time or input data, countless practical examples prove the applicability to real life problems.

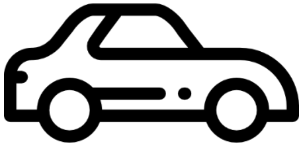
Since ANN are trained only by input data and expected outputs, they do not require full analytic knowledge of the physical domains being appoximated, thus making them powerful tools when analytical solutions are unknown, too complex or not real-time ready as long as inputs and outputs of the physical system to be modeled can be observed – or in case of outputs, generated manually.

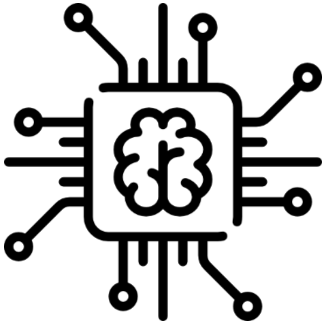
That being said, analytical solutions still ought to be sought whenever possible because they offer extended insights into the modeled domain as well as a potentially higher numeric accuracy.

Introduction: Artificial Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Lennart Adenaw, M. Sc.)

Agenda





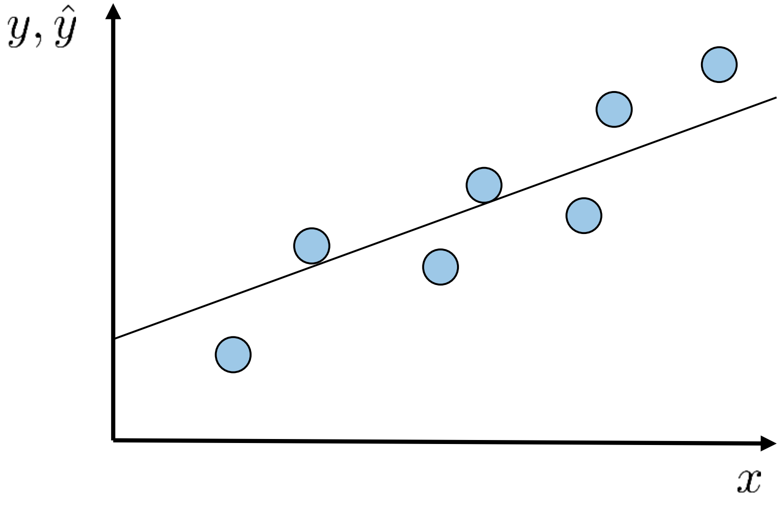
Towards Artificial Neurons

Linear Regression

The Simplest Approximation:

$$y = f(\vec{w} \cdot \vec{x}, b) = \sum_i w_i x_i + b = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

$\vec{x}$:	*Input Vector*
$\vec{w}$:	*Weight Vector*
b :	*Bias*
y :	*Output*
$\hat{y}$:	*Training Data*



Additional Slides

In order to derive the necessary ideas and maths for the understanding of neural networks, which are proven to be „universal approximators“, we start by taking a closer look at the simplest form of mathematical approximation – linear regression without basis functions.

Linear regression can be used to define and introduce a number of important concepts in the context of deep learning like Weights, Biases, Loss Function, Forward Pass and Gradient Descent.

From linear regression one can then derive more complex forms of regression by introducing non-linearities into the corresponding models. From this line of thought, the basic model of a Neuron as it is used in ANN can be derived.

Additional Slides

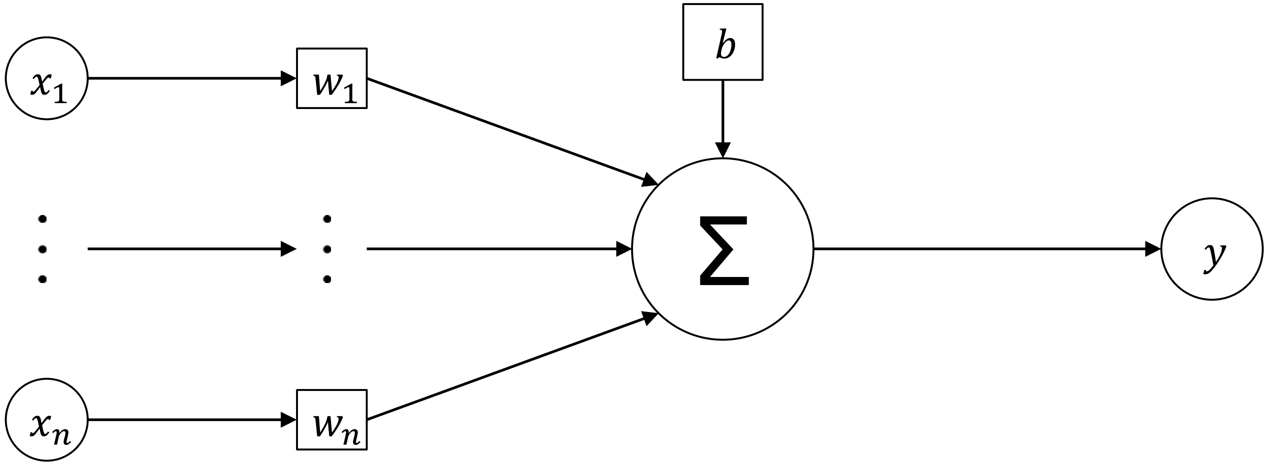
The idea of linear regression is to find Weights $\vec{w}$ and Bias b, such that the resulting function $y = f(\vec{w} \cdot \vec{x}, b) = \sum_i w_i x_i + b$ approximates a data set with inputs $\vec{x} \in X$ and Outputs $\hat{y} \in \hat{Y}$ as accurately as possible.

Towards Artificial Neurons

Linear Regression

Graphical Representation:

$$y = f(\vec{w} \cdot \vec{x}, b) = \sum_i w_i x_i + b = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

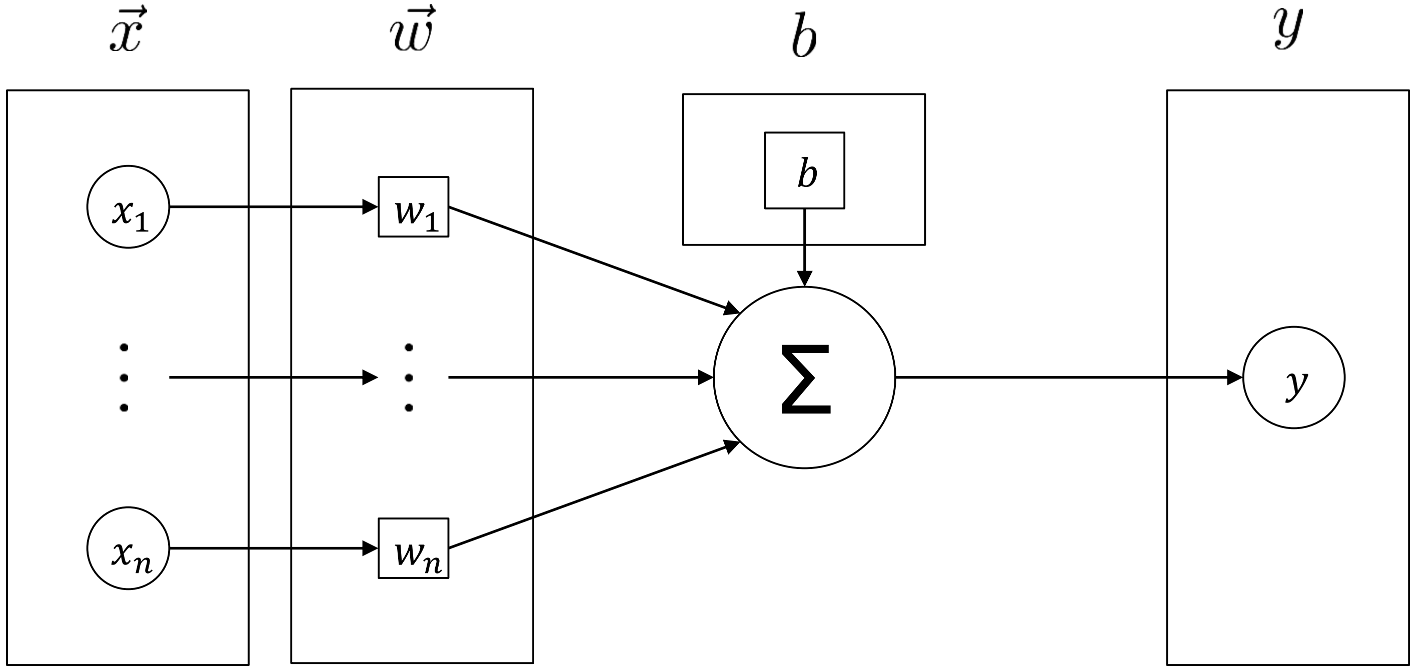


Towards Artificial Neurons

Linear Regression

Graphical Representation:

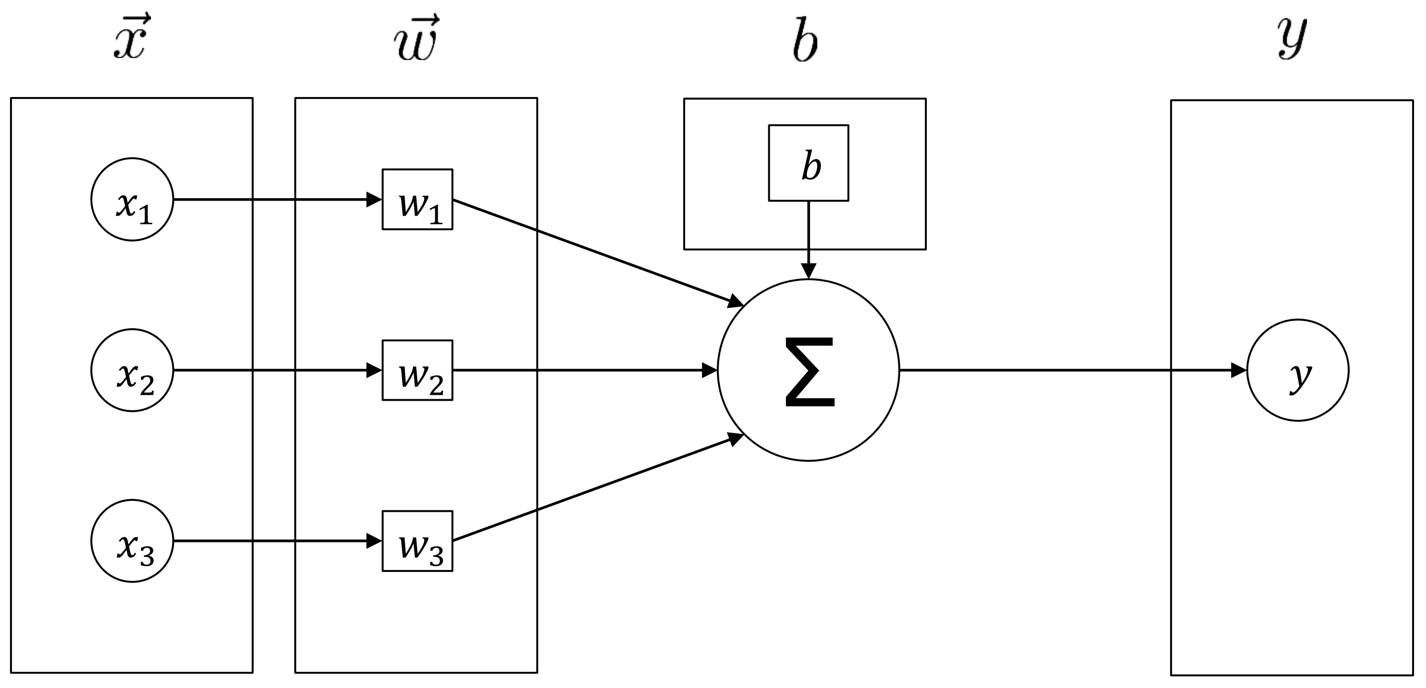
$$y = f(\vec{w} \cdot \vec{x}, b) = \sum_i w_i x_i + b = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$



Towards Artificial Neurons

Linear Regression

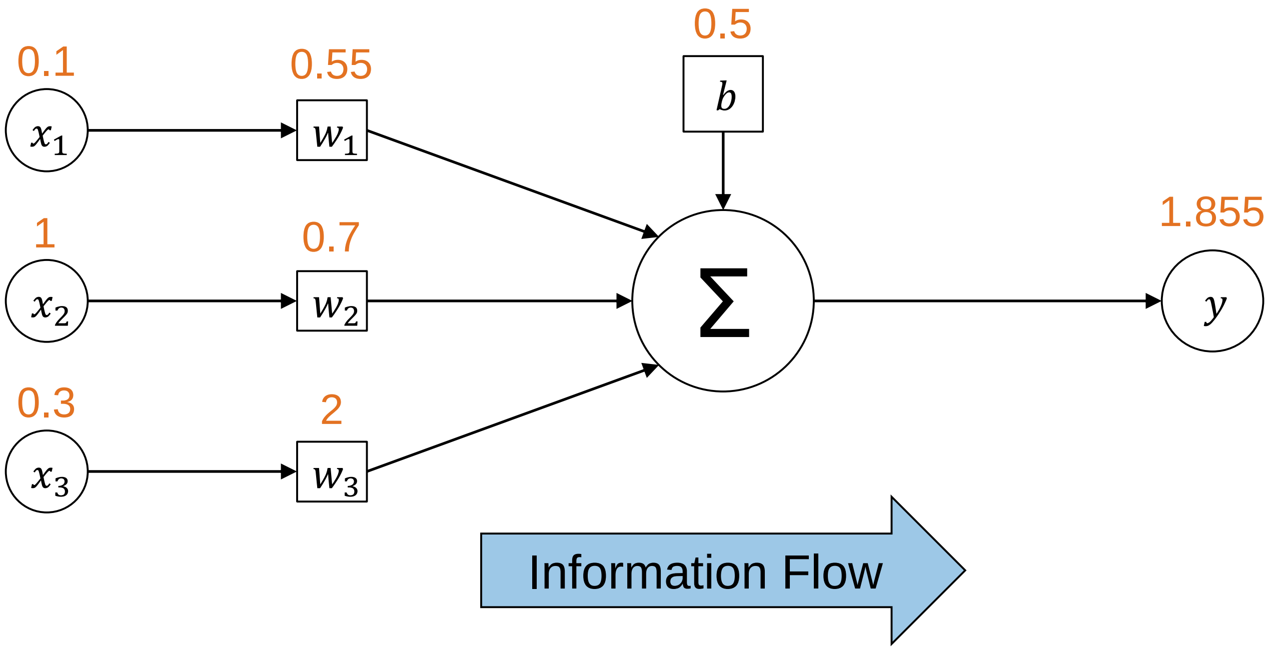
Graphical Representation – 3 Inputs Example:



Towards Artificial Neurons

Linear Regression

Forward Pass:



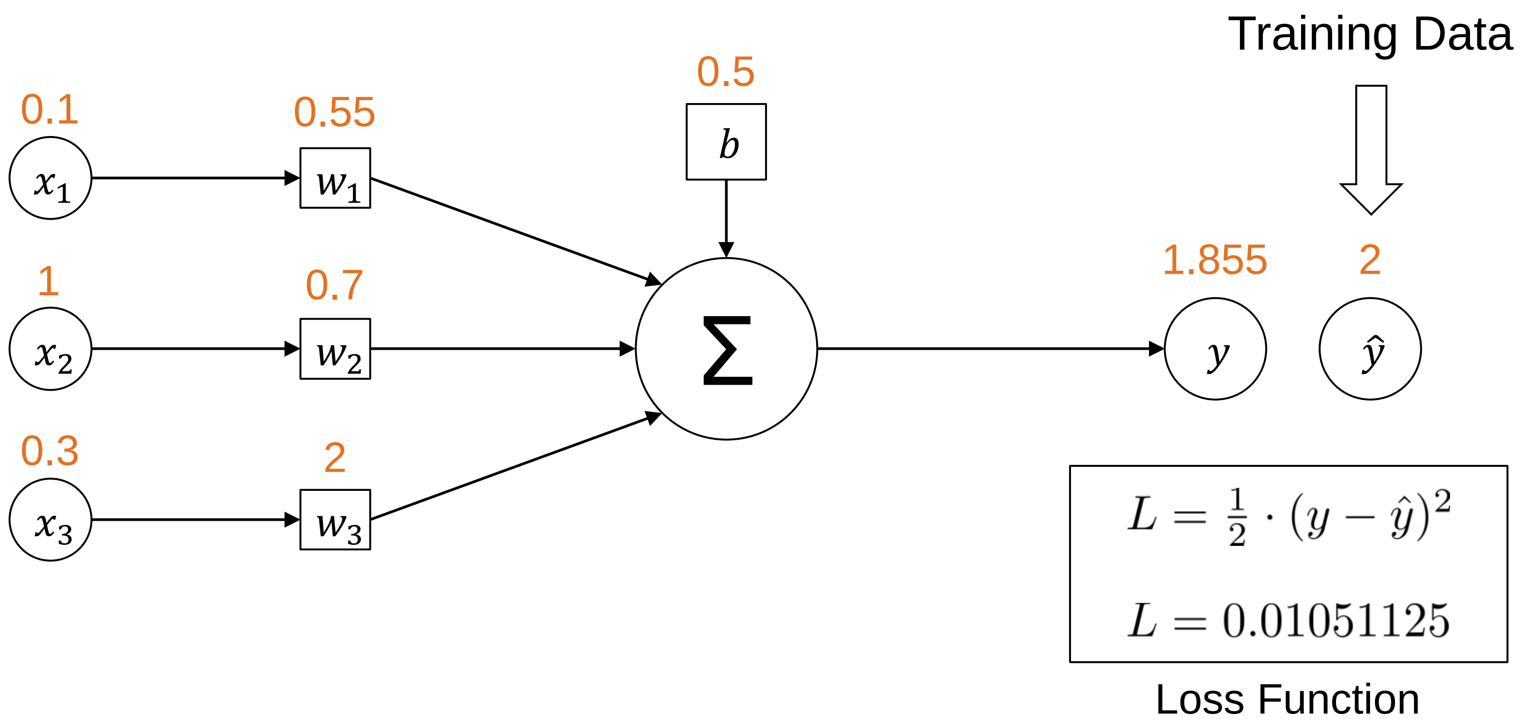
Additional Slides

Since this is an introduction to neural networks, neural network vocabulary is employed. Hence, the neural network term „Forward Pass“ is used, although it is not a term used in linear regression, because it refers to the corresponding process in an artificial neuron.

The „Backward Pass“ which is part of the „Backpropagation“ idea will be introduced the next lecture.

Towards Artificial Neurons
Linear Regression

Loss Function:



Towards Artificial Neurons
Linear Regression

Optimization Problem:

$$minimize_{\vec{w},b}(L(y,\hat{y}))$$

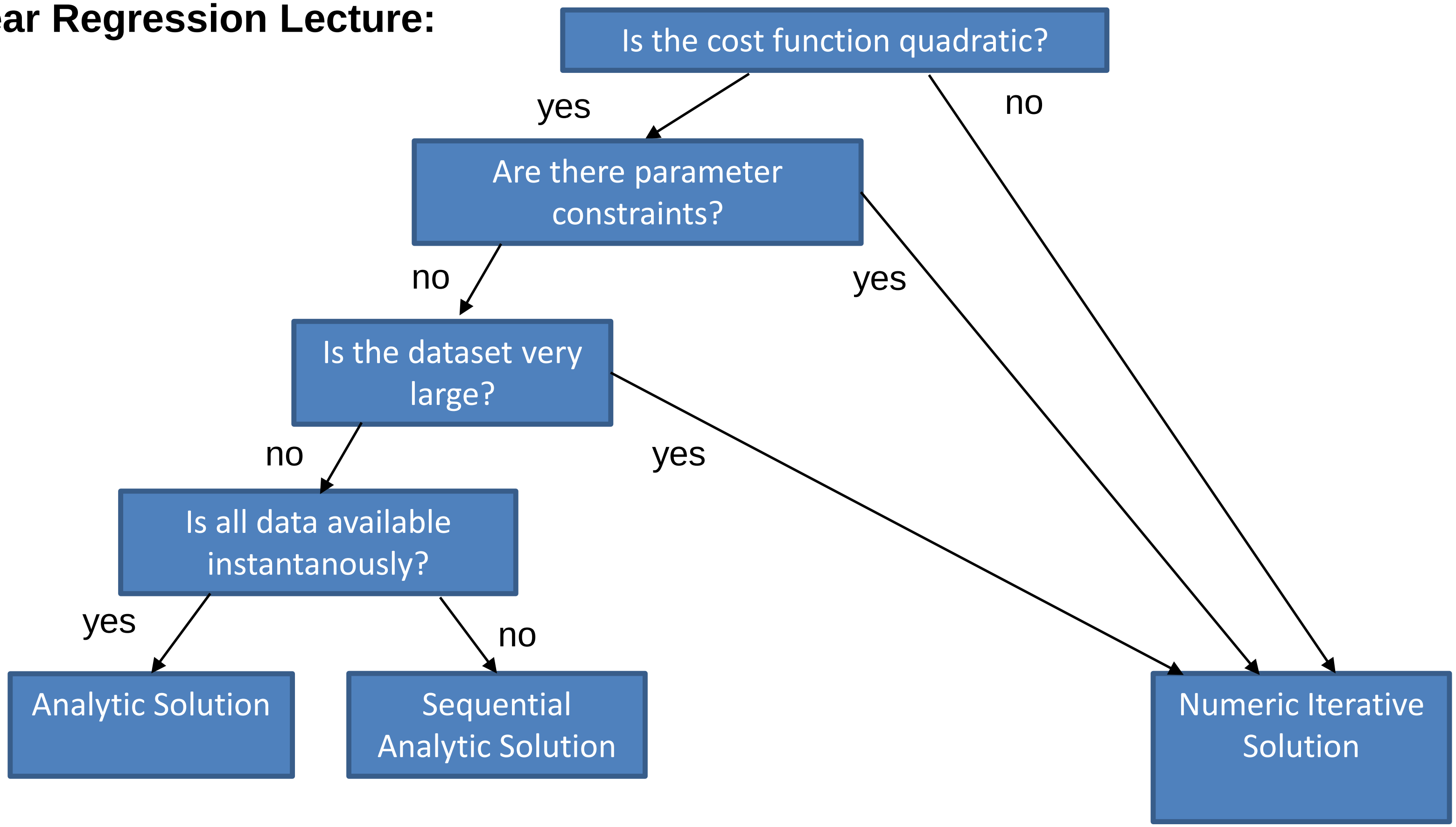
$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$

$$y = f(\vec{w},\vec{x}) = \sum_i (w_i \cdot x_i + b)$$

Towards Artificial Neurons

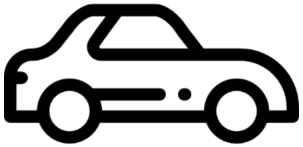
Linear Regression

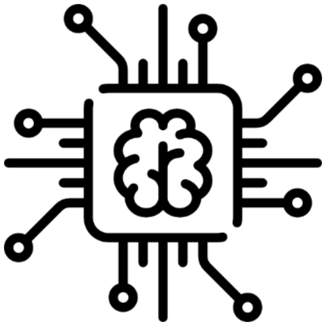
Linear Regression Lecture:



Introduction: Artificial Neural Networks
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Lennart Adenaw, M. Sc.)

Agenda





Towards Artificial Neurons

Gradient Descent

Approach:

- Gradient defines steepest ascent ($-\nabla L$)

- Update weights by a step in the opposite direction (i.e. steepest descent, length α)

- Stop when optimization criterium is met (e.g. loss threshold ϵ) or after N iterations

$$\nabla L = \begin{pmatrix} \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} \\ \vdots \\ \frac{\delta L}{\delta w_n} \\ \frac{\delta L}{\delta b} \end{pmatrix}$$

$$w_{\vec{new}} = w_{\vec{old}} - \alpha \cdot \nabla L$$

Towards Artificial Neurons
Gradient Descent

Well behaved:

$$\vec{w_{new}} = \vec{w_{old}} - \alpha \cdot \nabla L$$

$$L = w^2$$

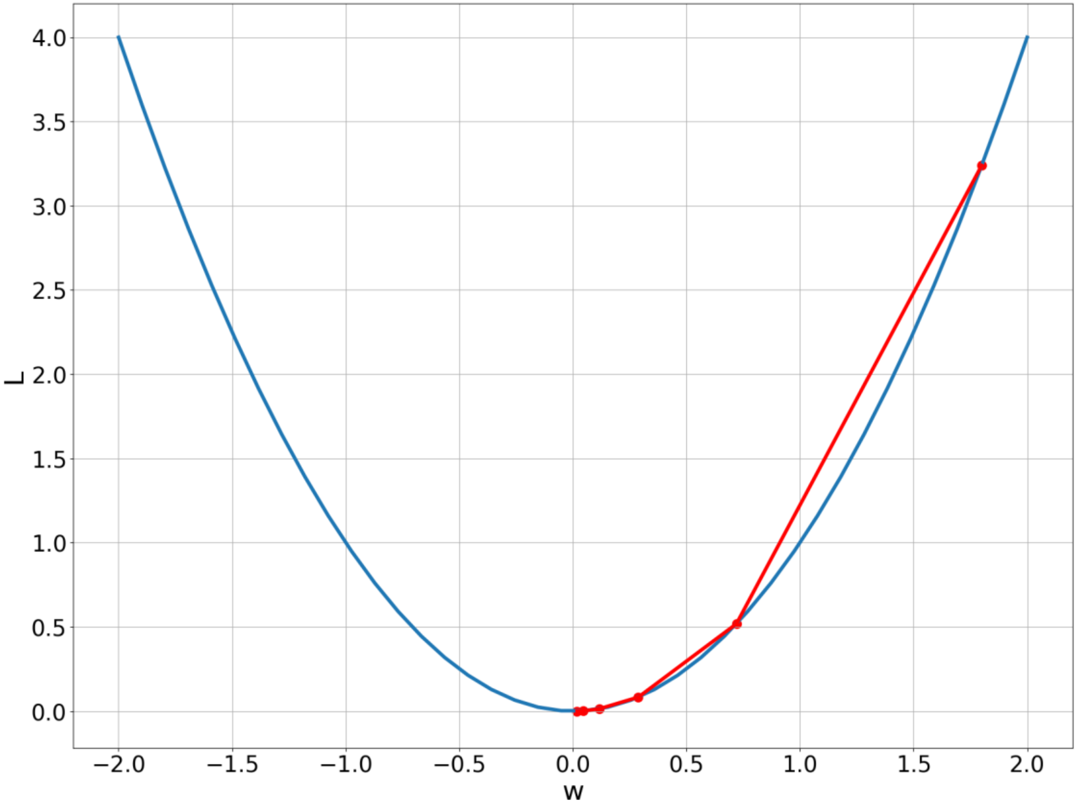
$$\nabla L = \frac{dL}{dw} = 2 \cdot w$$

$$\alpha = 0.3$$

$$w_0 = 1.8$$

$$\epsilon = 0.001$$

$$N = 5$$

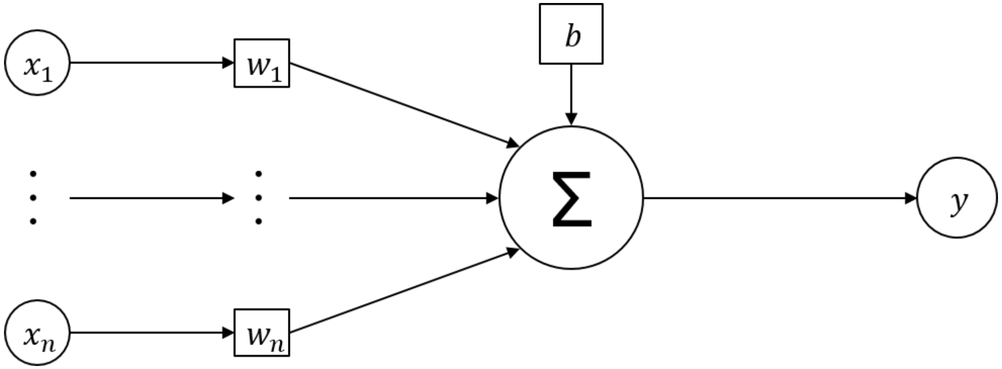


Towards Artificial Neurons

Gradient Descent

Finding the Gradient:

$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$



$$\nabla L = \begin{pmatrix} \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} \\ \vdots \\ \frac{\delta L}{\delta w_n} \\ \frac{\delta L}{\delta b} \end{pmatrix}$$

$$\frac{\delta L}{\delta w_i} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta w_i} \qquad \frac{\delta L}{\delta b} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta b}$$

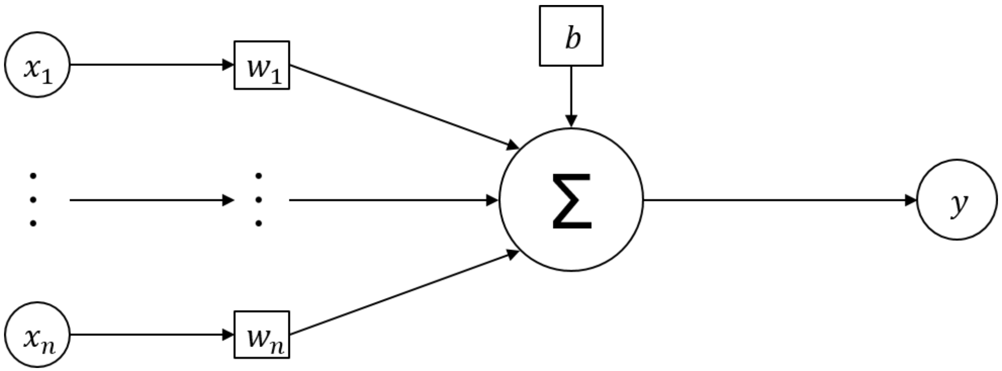
Towards Artificial Neurons

Gradient Descent

Finding the Gradient:

$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$

$$\nabla L = \begin{pmatrix} \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} \\ \vdots \\ \frac{\delta L}{\delta w_n} \\ \frac{\delta L}{\delta b} \end{pmatrix}$$



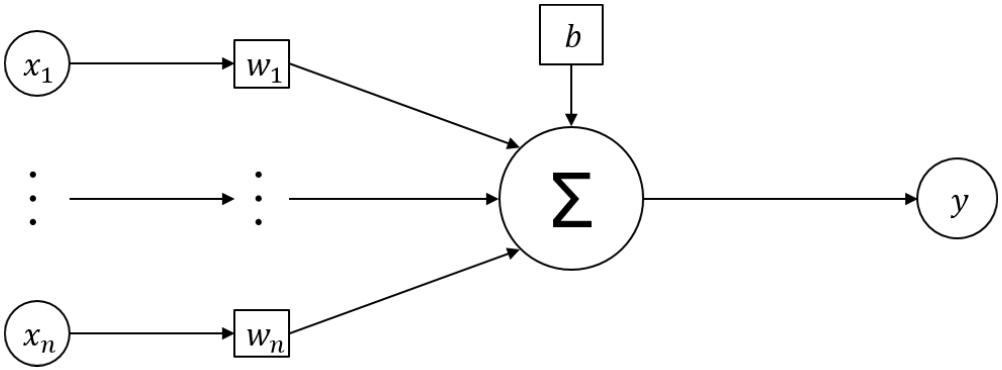
$$\frac{\delta L}{\delta w_i} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta w_i}$$

$$\frac{\delta L}{\delta b} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta b}$$

Towards Artificial Neurons

Gradient Descent

Finding the Gradient:



$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$

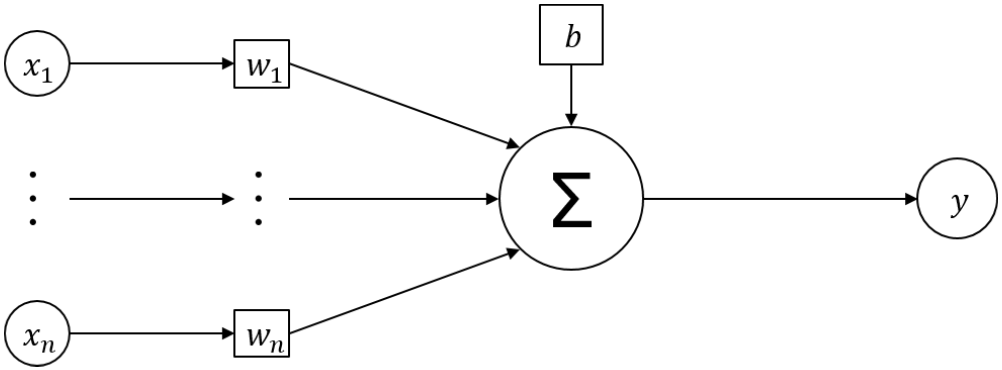
$$\nabla L = \begin{pmatrix} \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} \\ \vdots \\ \frac{\delta L}{\delta w_n} \\ \frac{\delta L}{\delta b} \end{pmatrix}$$

$$\frac{\delta L}{\delta w_i} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta w_i} \qquad \frac{\delta L}{\delta b} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta b}$$

$$\frac{dL}{dy} = \frac{d}{dy} \frac{1}{2} \cdot (y - \hat{y})^2 = y - \hat{y} = \Delta y$$

Towards Artificial Neurons
Gradient Descent

Finding the Gradient:



$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$

$$\nabla L = \begin{pmatrix} \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} \\ \vdots \\ \frac{\delta L}{\delta w_n} \\ \frac{\delta L}{\delta b} \end{pmatrix}$$

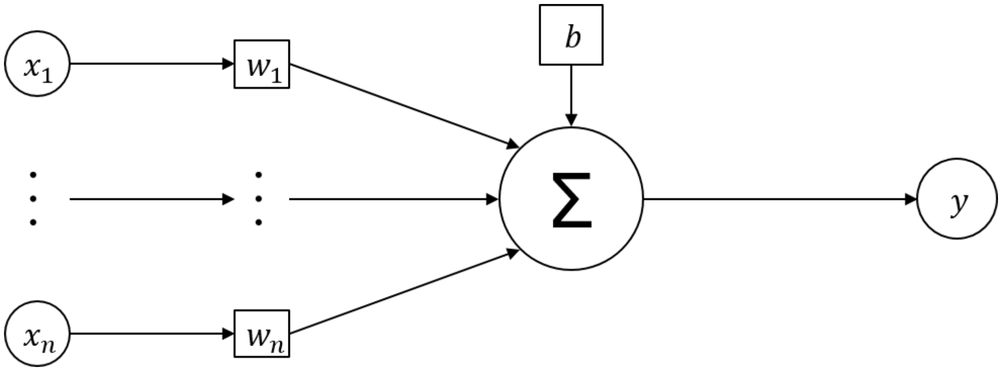
$$\frac{\delta L}{\delta w_i} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta w_i} = \Delta y \cdot x_i$$

$$\frac{\delta y}{\delta w_i} = \frac{\delta}{\delta w_i} \sum_i w_i \cdot x_i + b = x_i$$

Towards Artificial Neurons

Gradient Descent

Finding the Gradient:



$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$

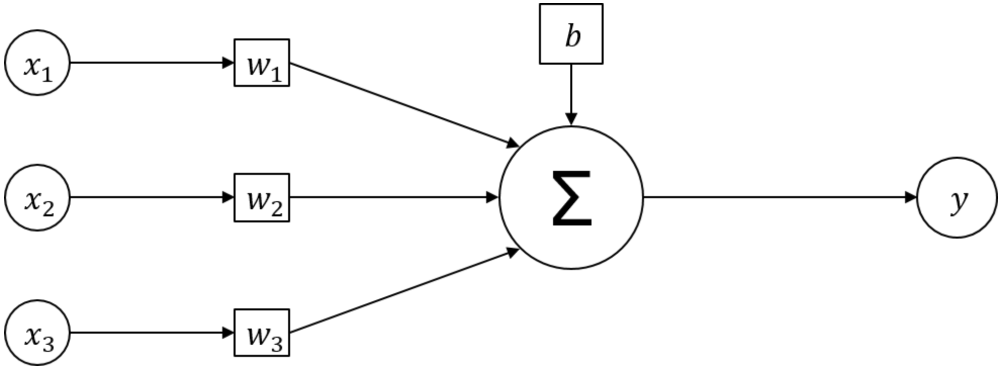
$$\nabla L = \begin{pmatrix} \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} \\ \vdots \\ \frac{\delta L}{\delta w_n} \\ \frac{\delta L}{\delta b} \end{pmatrix}$$

$$\frac{\delta L}{\delta b} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta b} = \frac{dL}{dy} \cdot \frac{\delta y}{\delta w_i} = \Delta y$$

$$\frac{\delta y}{\delta b} = \frac{\delta}{\delta b} \cdot \sum_i w_i \cdot x_i + b = 1$$

Towards Artificial Neurons
Gradient Descent

Numerical Example:

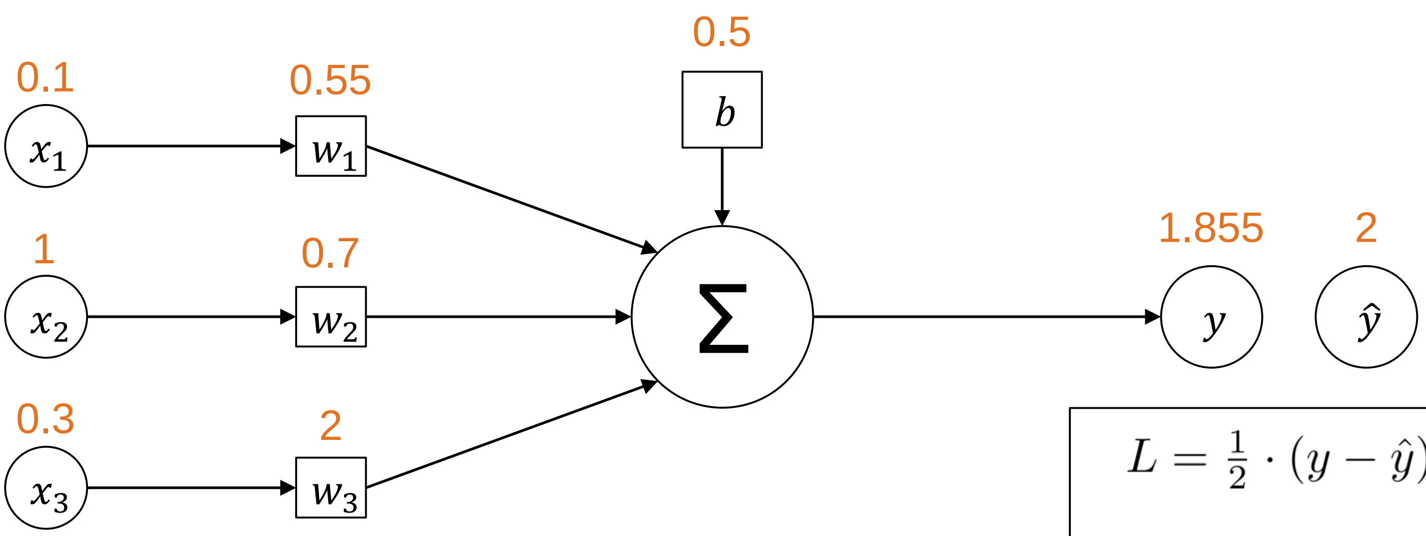


$$\nabla L = \begin{pmatrix} \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} \\ \frac{\delta L}{\delta w_3} \\ \frac{\delta L}{\delta b} \end{pmatrix} = \begin{pmatrix} \Delta y \cdot x_1 \\ \Delta y \cdot x_2 \\ \Delta y \cdot x_3 \\ \Delta y \end{pmatrix}$$

Towards Artificial Neurons

Gradient Descent

$$\nabla L = \begin{pmatrix} \Delta y \cdot x_1 \\ \Delta y \cdot x_2 \\ \Delta y \cdot x_3 \\ \Delta y \end{pmatrix} = - \begin{pmatrix} 0.145 \cdot 0.1 \\ 0.145 \cdot 1 \\ 0.145 \cdot 0.3 \\ 0.145 \end{pmatrix} = - \begin{pmatrix} 0.0145 \\ 0.145 \\ 0.0435 \\ 0.145 \end{pmatrix}$$



$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$

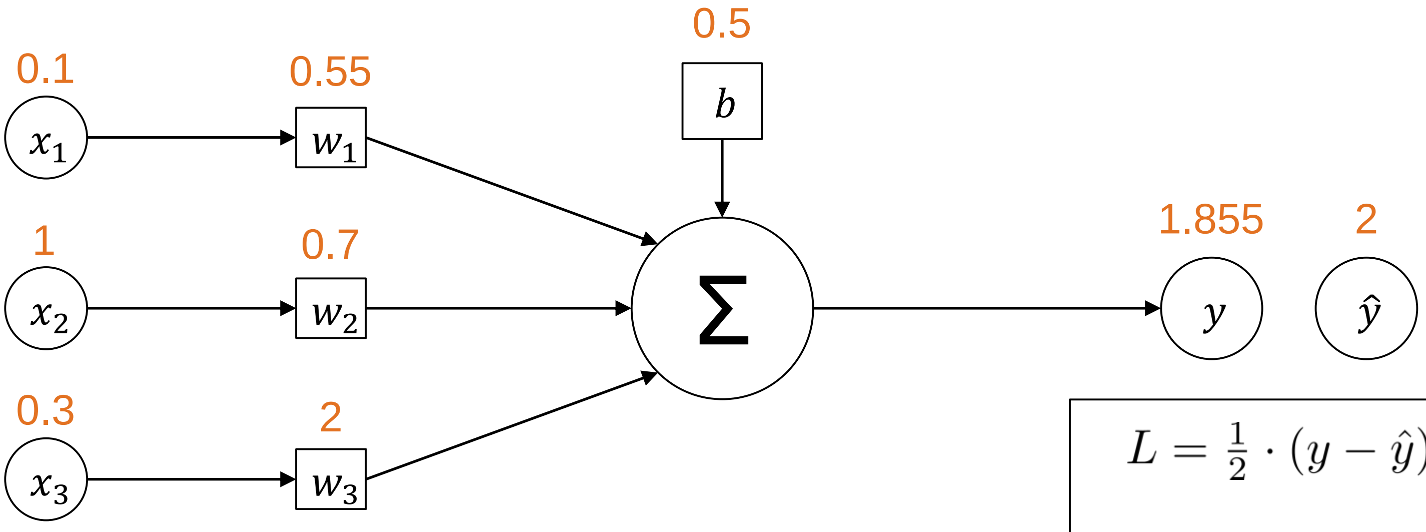
$$L = 0.01051125$$

Numerical Example – One Iteration

Towards Artificial Neurons

Gradient Descent

$$\nabla L = -\begin{pmatrix} 0.0145 \\ 0.145 \\ 0.0435 \\ 0.145 \end{pmatrix} \quad \alpha = 0.5, \vec{w_{new}} = \vec{w_{old}} - 0.5 \cdot \nabla L$$



$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$

$$L = 0.01051125$$

Numerical Example – One Iteration

Towards Artificial Neurons

Gradient Descent

$$\nabla L = -\begin{pmatrix} 0.0145 \\ 0.145 \\ 0.0435 \\ 0.145 \end{pmatrix} \quad \alpha = 0.5, \vec{w_{new}} = \vec{w_{old}} - 0.5 \cdot \nabla L$$

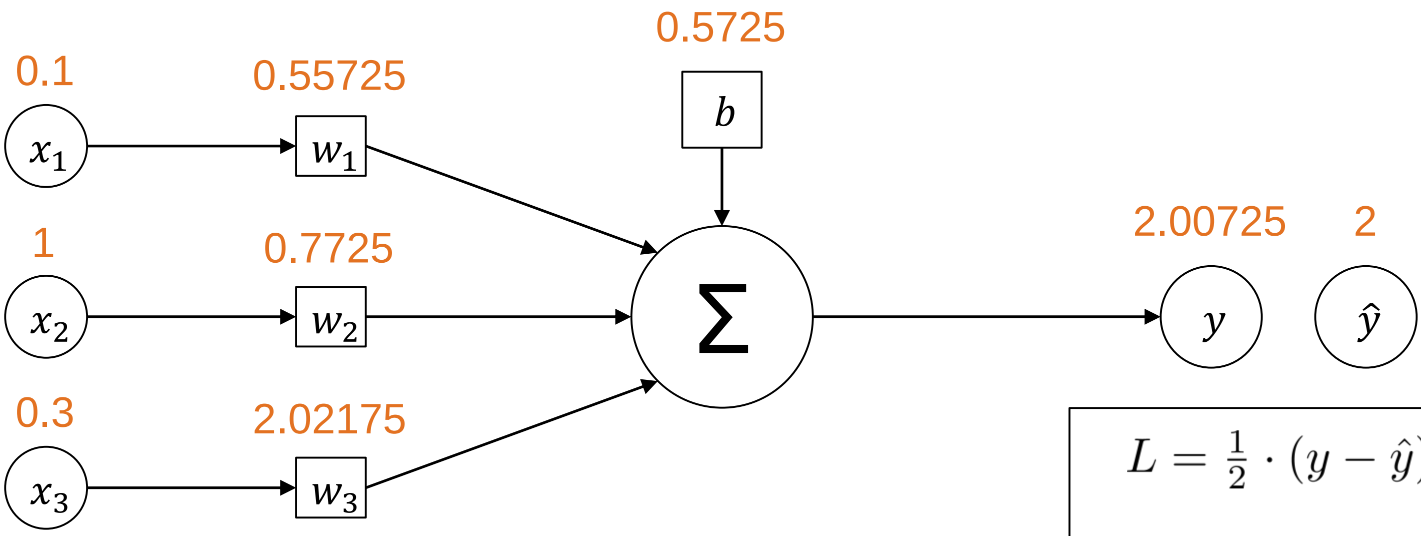
$$\vec{w_{new}} = \vec{w_{old}} - 0.5 \cdot \nabla L$$

$$\begin{pmatrix} 0.55 \\ 0.7 \\ 2 \\ 0.5 \end{pmatrix} + 0.5 \cdot \begin{pmatrix} 0.0145 \\ 0.145 \\ 0.0435 \\ 0.145 \end{pmatrix} = \begin{pmatrix} 0.55725 \\ 0.7725 \\ 2.02175 \\ 0.5725 \end{pmatrix}$$

Numerical Example – One Iteration

Towards Artificial Neurons
Gradient Descent

$$\vec{w_{new}} = \begin{pmatrix} 0.55725 \\ 0.7725 \\ 2.02175 \\ 0.5725 \end{pmatrix}$$



$$L = \frac{1}{2} \cdot (y - \hat{y})^2$$

$$L = 0.00002628125$$

Numerical Example – One Iteration

Towards Artificial Neurons

Gradient Descent

Stuck in Local Minimum:

$$\vec{w_{new}} = \vec{w_{old}} - \alpha \cdot \nabla L$$

$$L = w^4 - w^3 - 7w^2 + w + 20$$

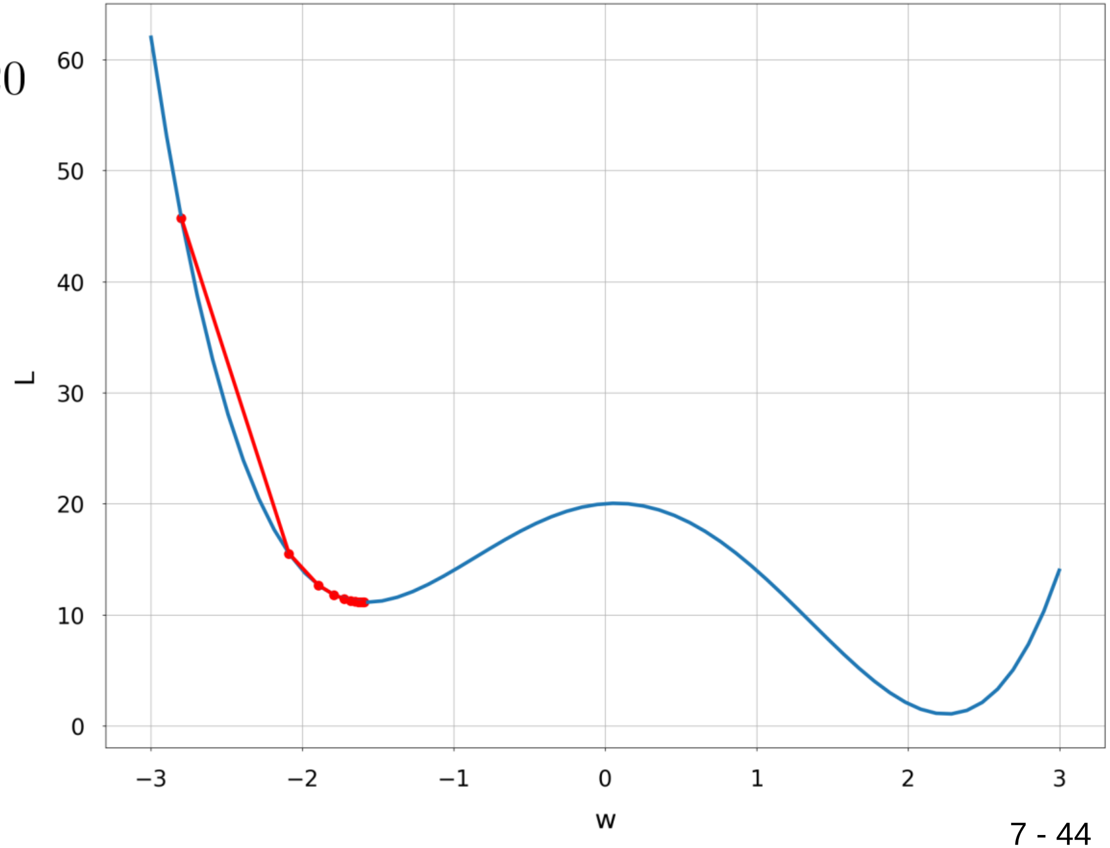
$$\nabla L = 4w^3 - 3w^2 - 14w + 1$$

$$\alpha = 0.01$$

$$w_0 = -2.8$$

$$\epsilon = 0.001$$

$$N = 10$$



Towards Artificial Neurons
Gradient Descent

Vanishing Gradient:

$$\vec{w_{new}} = \vec{w_{old}} - \alpha \cdot \nabla L$$

$$L = w^3$$

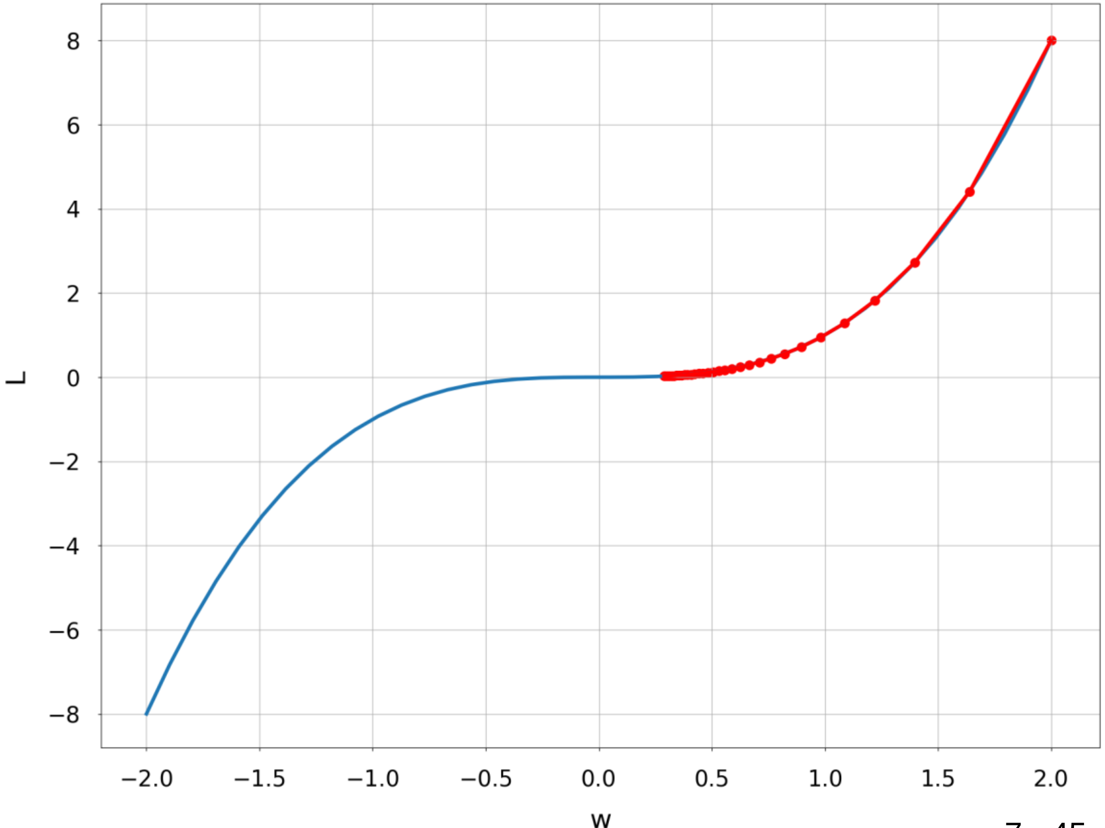
$$\nabla L = \frac{dL}{dw} = 3w^2$$

$$\alpha = 0.03$$

$$w_0 = 2$$

$$\epsilon = 0.001$$

$$N = 30$$



Towards Artificial Neurons

Gradient Descent

Oscillating:

$$w_{\vec{new}} = w_{\vec{old}} - \alpha \cdot \nabla L$$

$$L = w^2$$

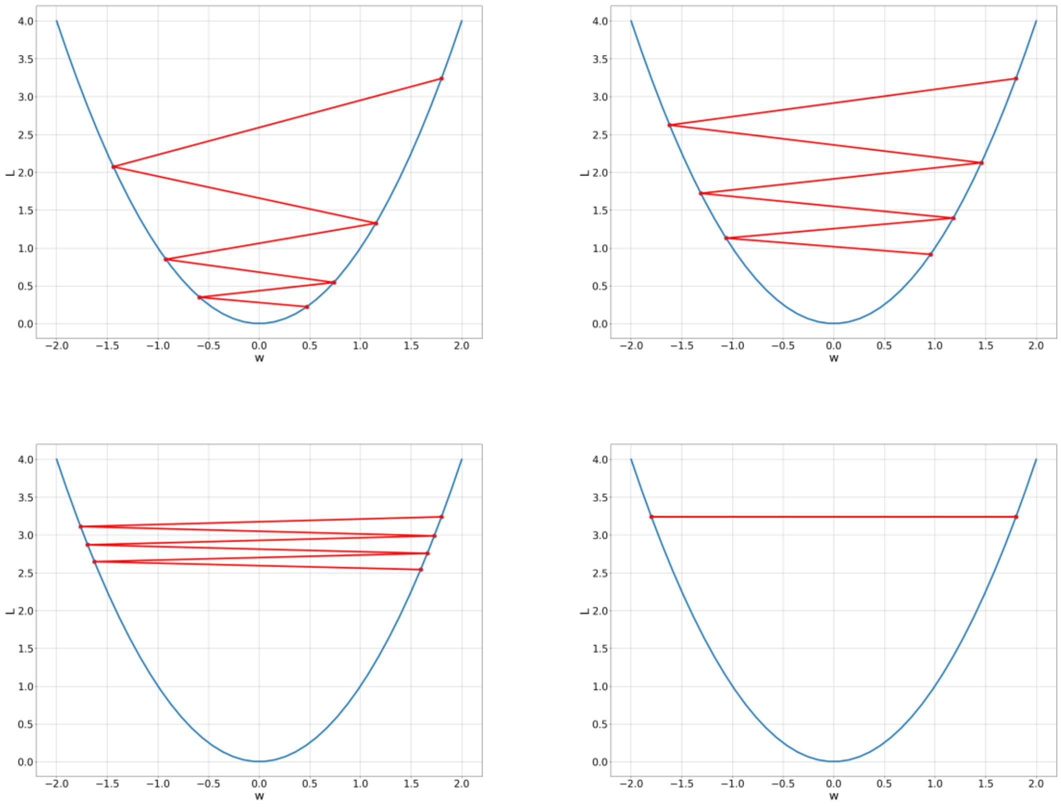
$$\nabla L = \frac{dL}{dw} = 2 \cdot w$$

$$\alpha = [0.9, 0.95, 0.99, 1]$$

$$w_0 = 1.8$$

$$\epsilon = 0.001$$

$$N = 5$$



Towards Artificial Neurons

Gradient Descent

Jumping out of Minima:

$$\vec{w_{new}} = \vec{w_{old}} - \alpha \cdot \nabla L$$

$$L = w^4 - w^3 - 7w^2 + w + 20$$

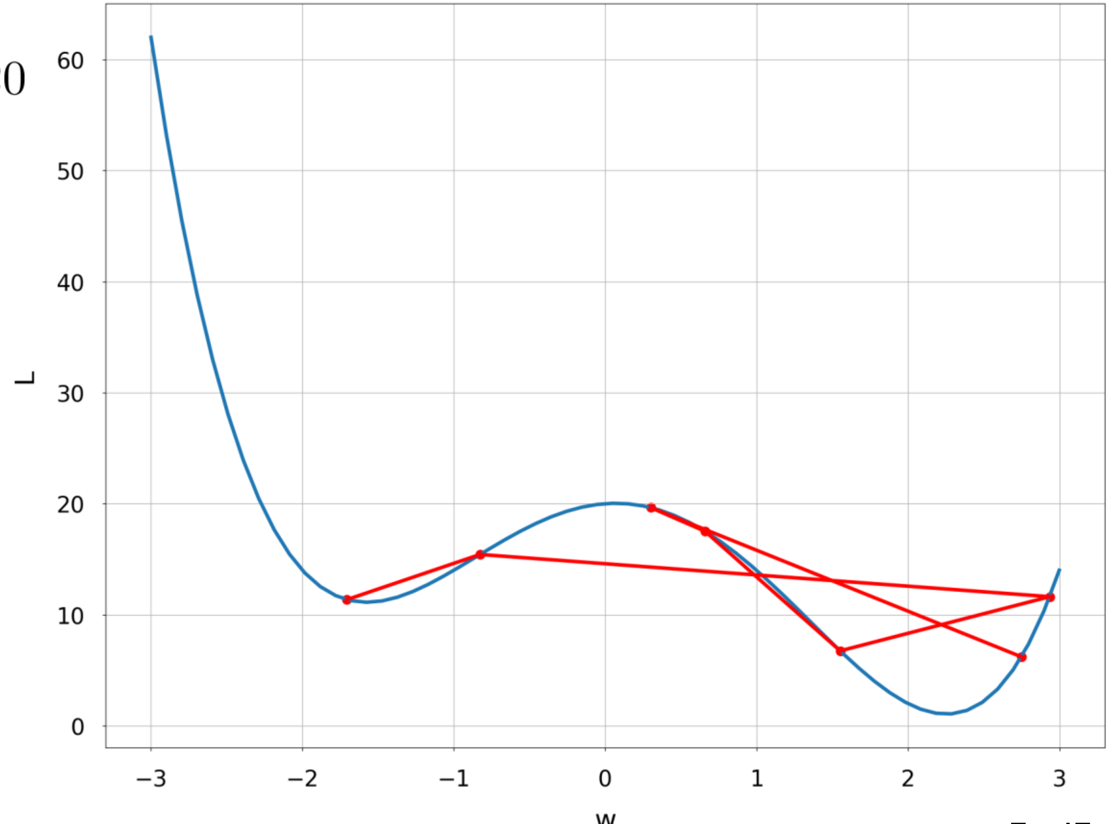
$$\nabla L = 4w^3 - 3w^2 - 14w + 1$$

$$\alpha = 0.1065$$

$$w_0 = 2.75$$

$$\epsilon = 0.001$$

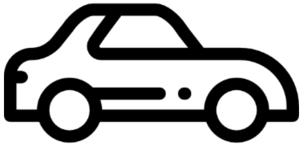
$$N = 5$$

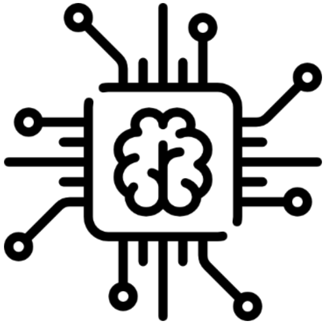


Introduction: Artificial Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Lennart Adenaw, M. Sc.)

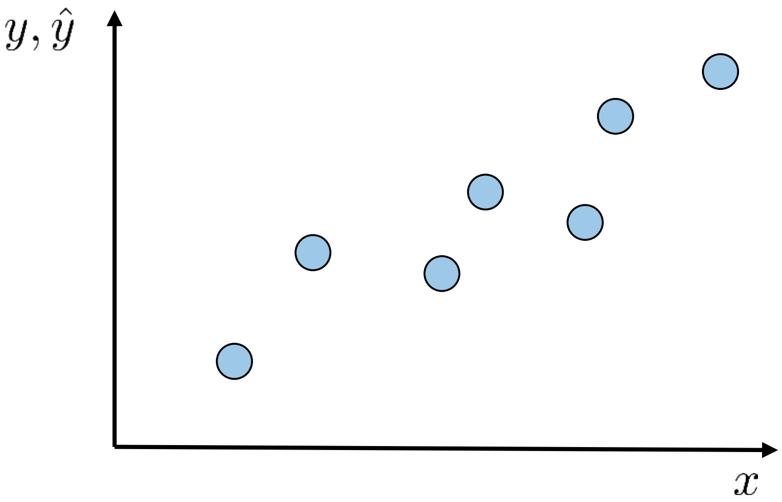
Agenda



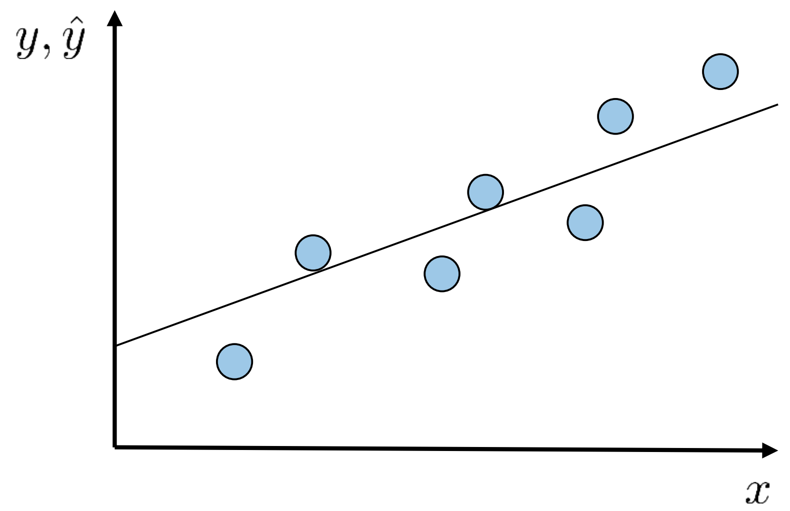


Towards Artificial Neurons

Wrap Up:



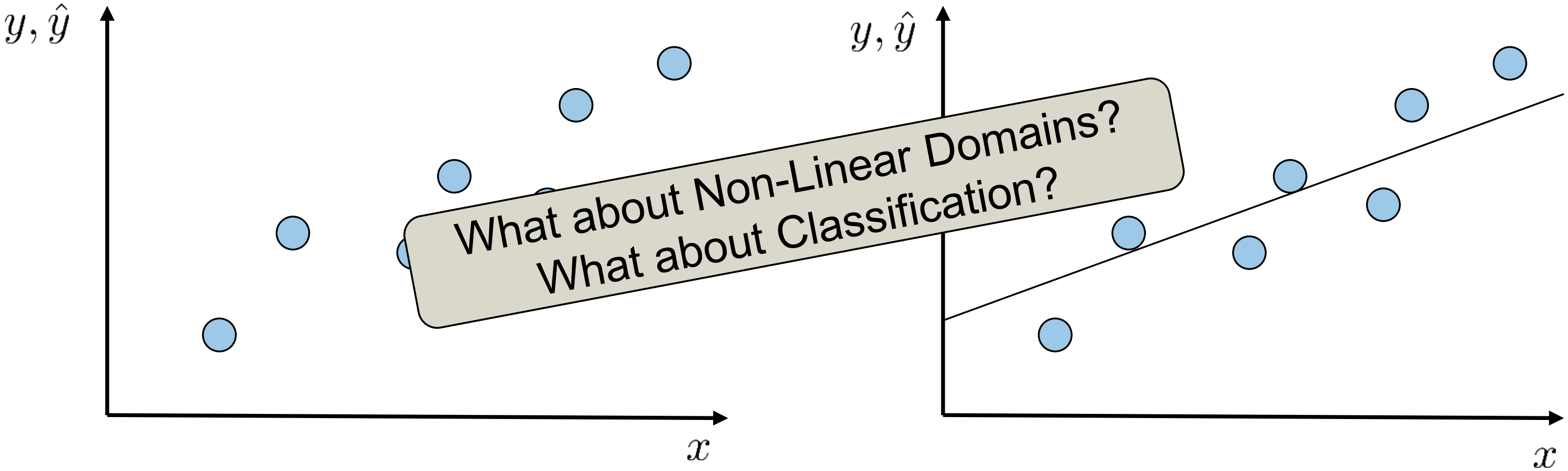
Input Data
Specific Observations of the Domain



Abstraction of the Domain
Regression
Prediction

Towards Artificial Neurons

Wrap Up:



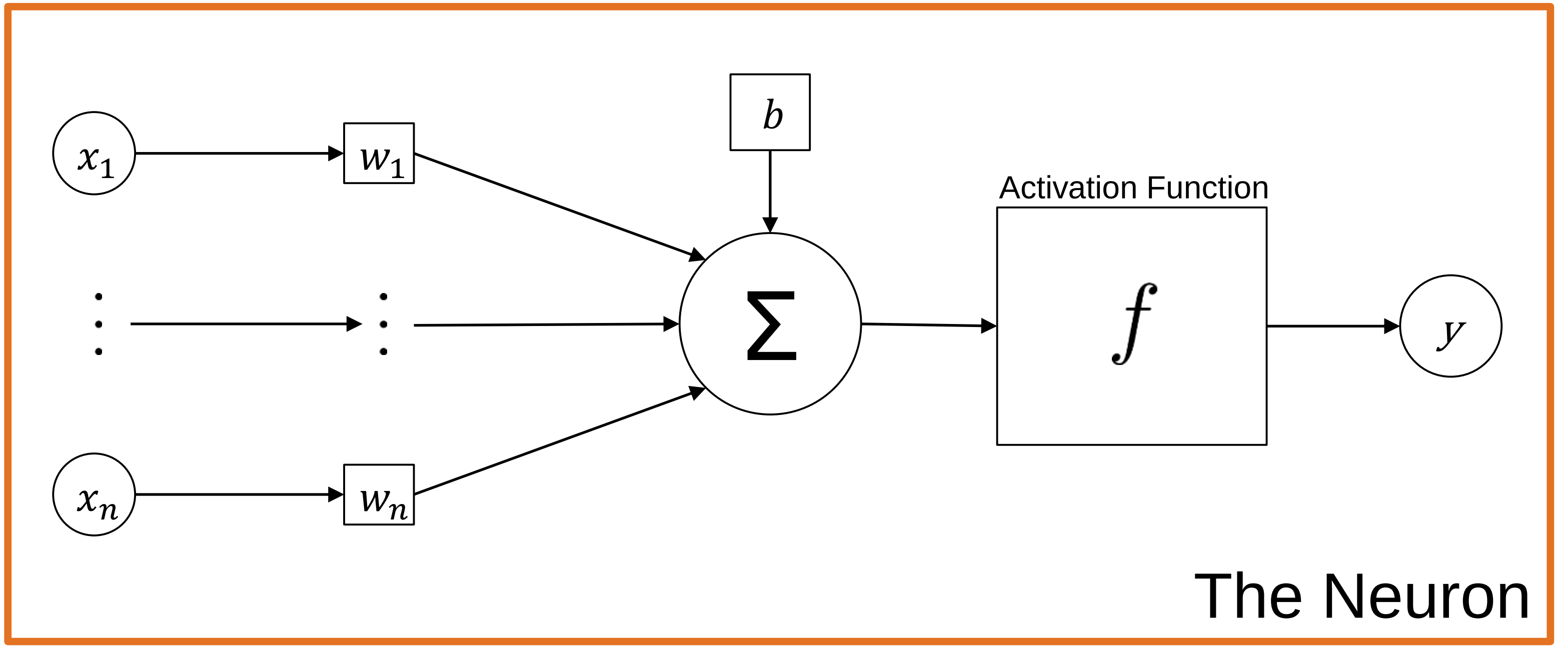
Input Data
Specific Observations of the Domain

Abstraction of the Domain
Regression
Prediction

Towards Artificial Neurons

The Neuron

Introduction of an Activation Function:

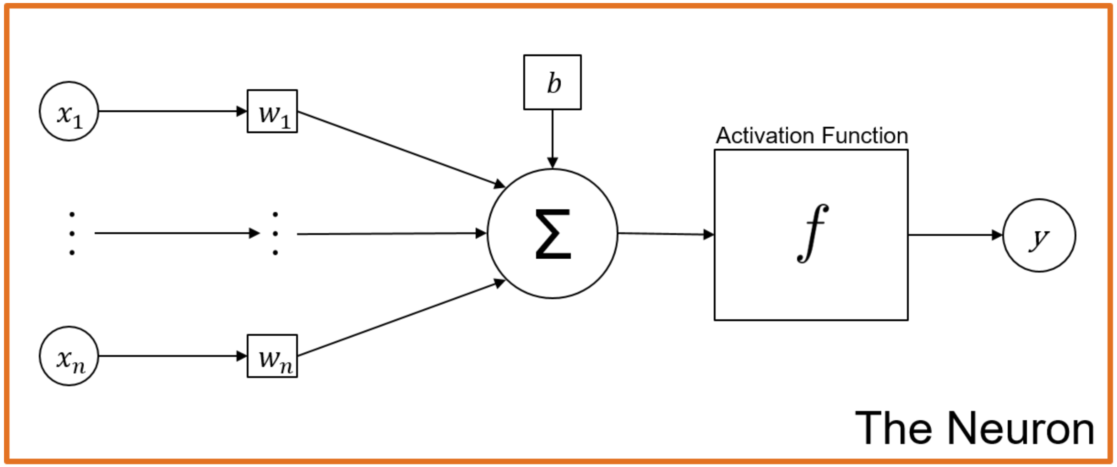


Towards Artificial Neurons

The Neuron

Properties:

- One Output
- One or many Inputs
- Bias b, Weights w
- Activation Function f
- $y = f(\sum_i w_i \cdot x_i + b)$



Additional Slides

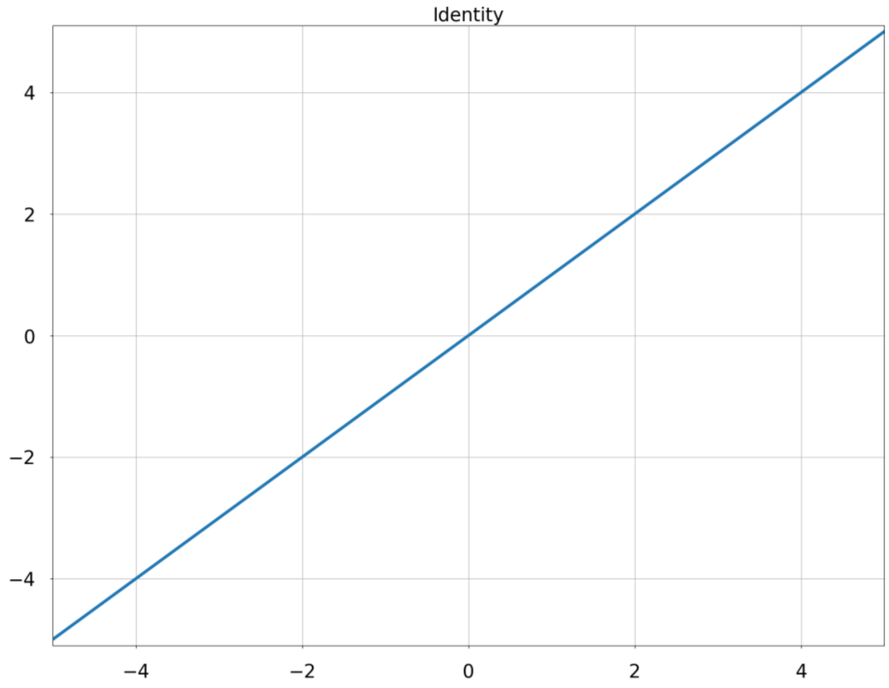
In order to enable approximation of non-linear domains, the Linear Regression model is augmented by a non-linear activation function. The resulting model is called an Artificial Neuron and serves as the base component of an ANN.
Aritificial Neurons can be wired together to form arbitrarily large structures in which arrays of Neurons of the same hierarchy level are called „Layers“.

Each ANN has an „Input Layer“ where the data is fed into the network, a number of „Hidden Layers“ made up of artificial Neurons and an „Output Layer“ which contains the results of the forward pass.

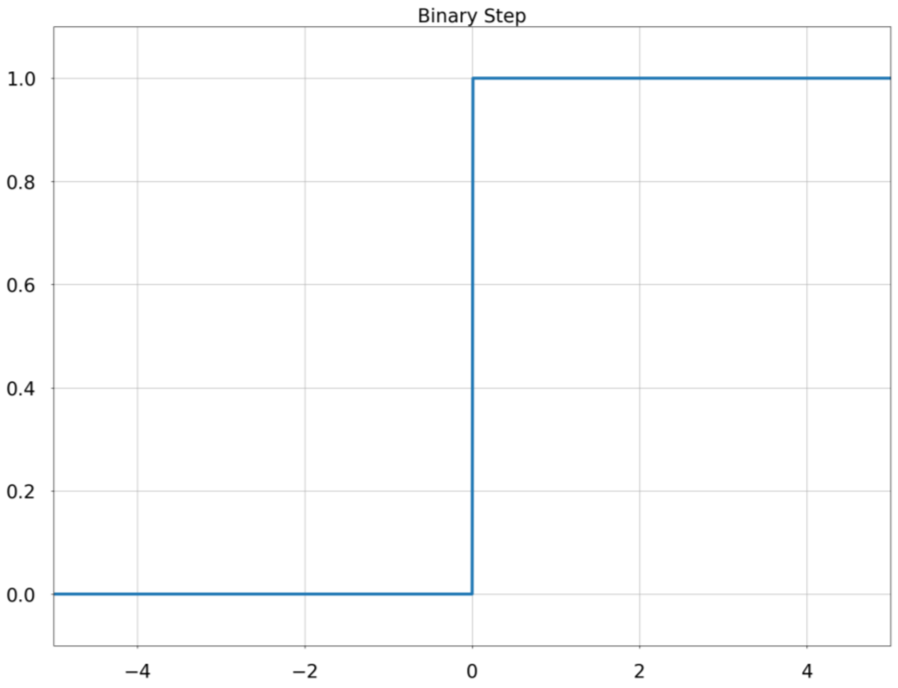
Towards Artificial Neurons

The Neuron

Linear Regression



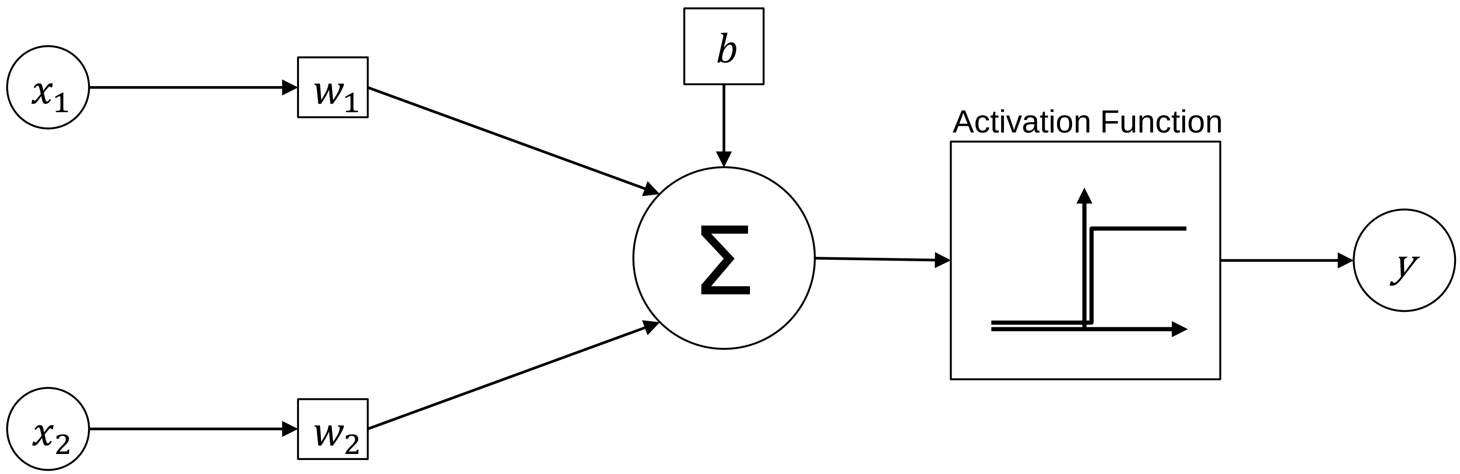
Binary Classification



Towards Artificial Neurons

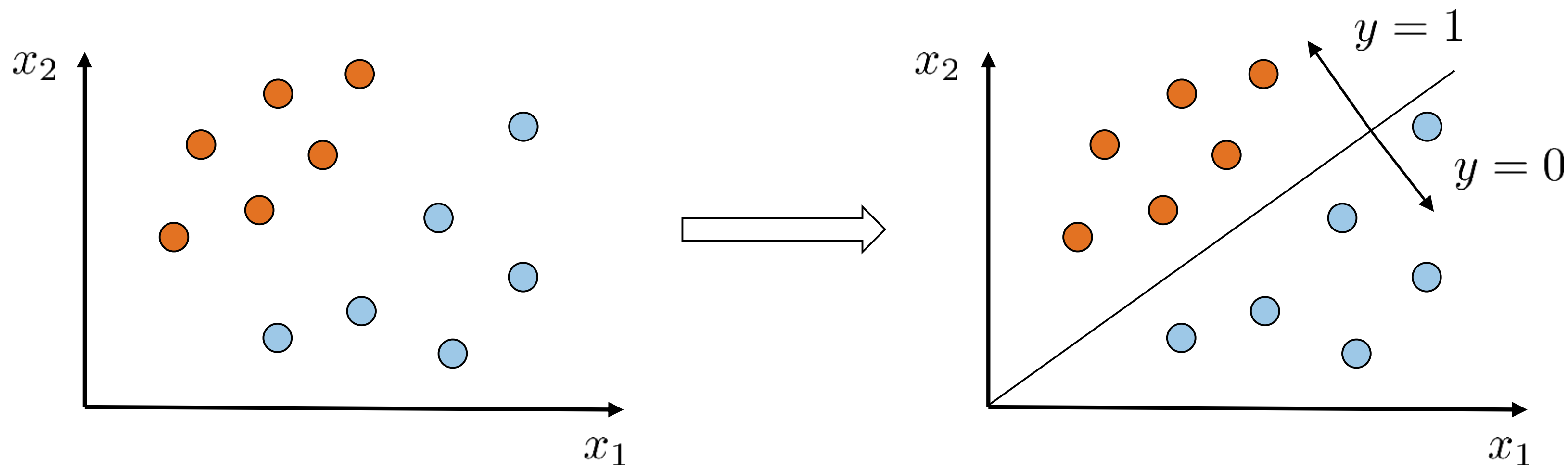
The Neuron

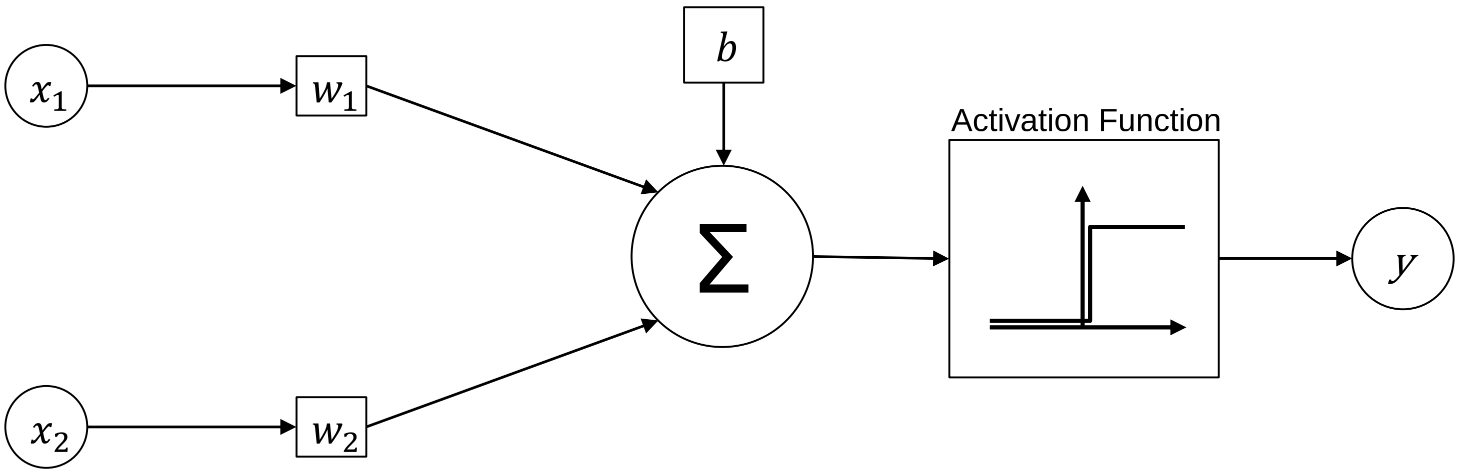
Suppose the following Setup:



Towards Artificial Neurons

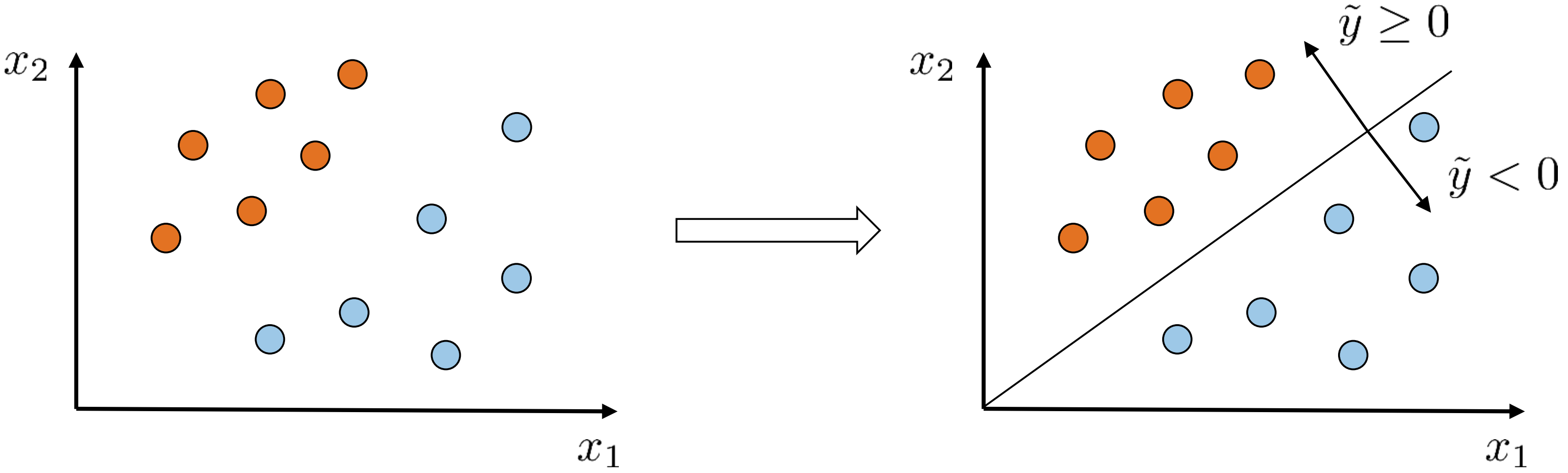
The Neuron

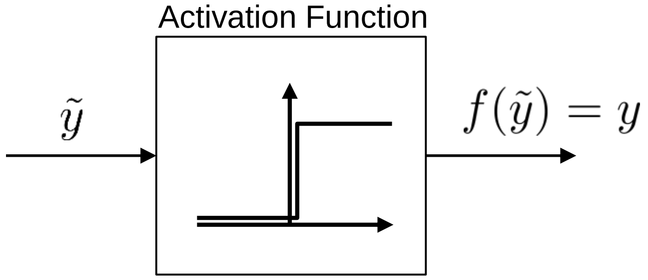




Towards Artificial Neurons

The Neuron

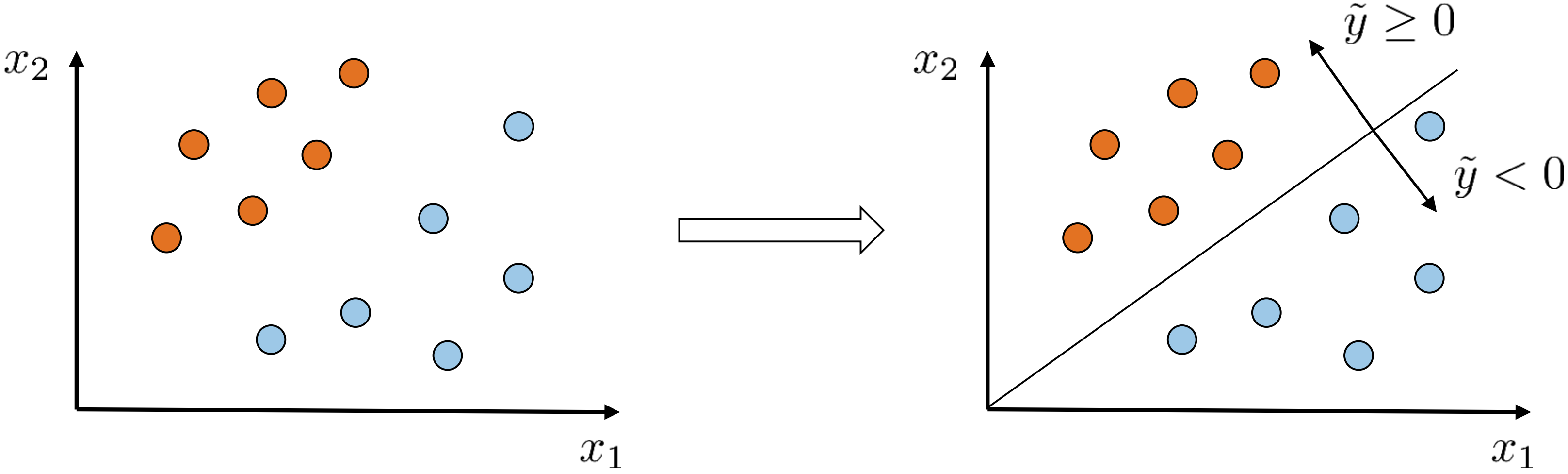




$$f = \begin{cases} 1 & \tilde{y} \geq 0 \\ 0 & \tilde{y} < 0 \end{cases}$$

Step Function

The Neuron



Activation Fu

$$\frac{\mathrm{d}f}{\mathrm{d}\tilde{y}} = \begin{cases} 0 & \tilde{y} > 0 \\ \infty & \tilde{y} = 0 \\ 0 & \tilde{y} < 0 \end{cases}$$

$$f = \begin{cases} 1 & \tilde{y} \geq 0 \\ 0 & \tilde{y} < 0 \end{cases}$$

Step Function

Additional Slides

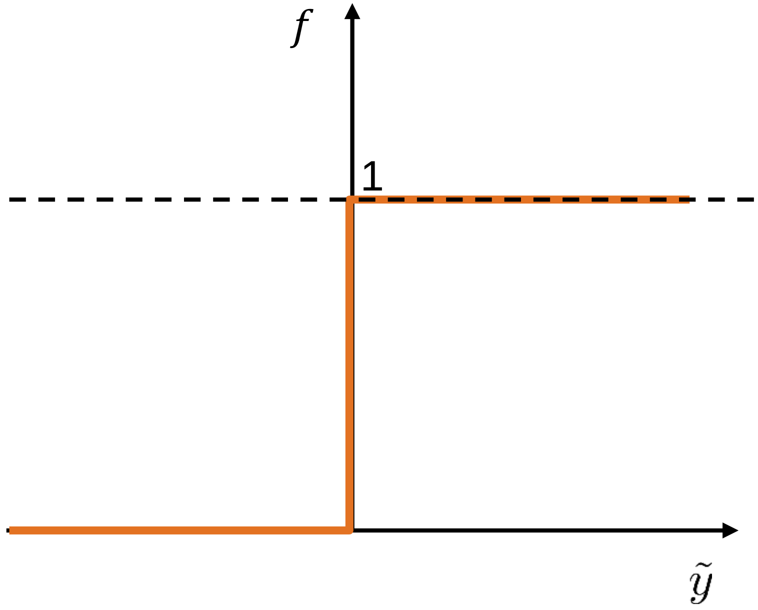
To derive an Artificial Neuron from the Linear Regression model, an activation function was added. In order to perform a simple binary classification task with only one artificial neuron, a „Threshold“ or „Step Function“ is used as the activation function. The idea behind this is to generate an output which is either 1 or 0 depending on the class an input is coming from.

The problem with the binary step is that gradient descent fails to converge against a reasonable set of weights, because the derivative of the loss function will be either 0 or very large through the introduction of a binary step activation function.

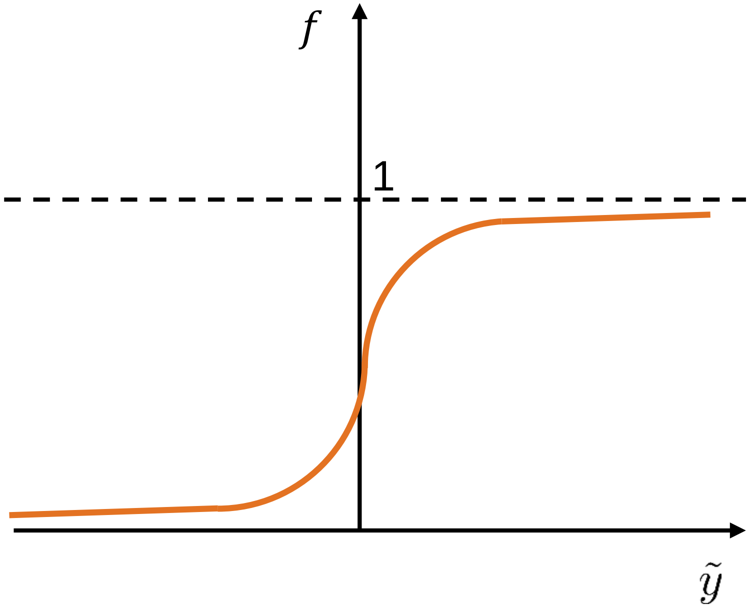
Towards Artificial Neurons

The Neuron

Step Function



Sigmoid Function



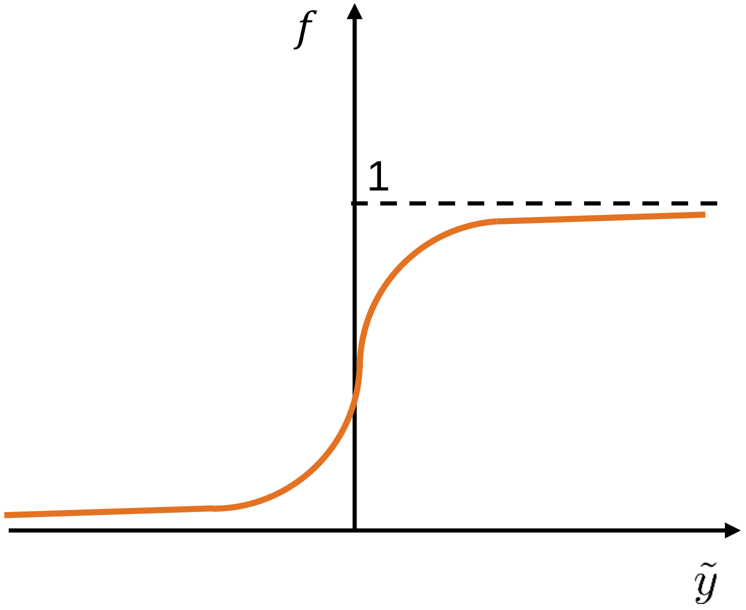
Towards Artificial Neurons
The Neuron

$$f = \frac{1}{1+e^{-\tilde{y}}} = \frac{e^{\tilde{y}}}{1+e^{\tilde{y}}}$$

$$\frac{df}{d\tilde{y}} = f(1-f)$$

Continuously Differentiable!

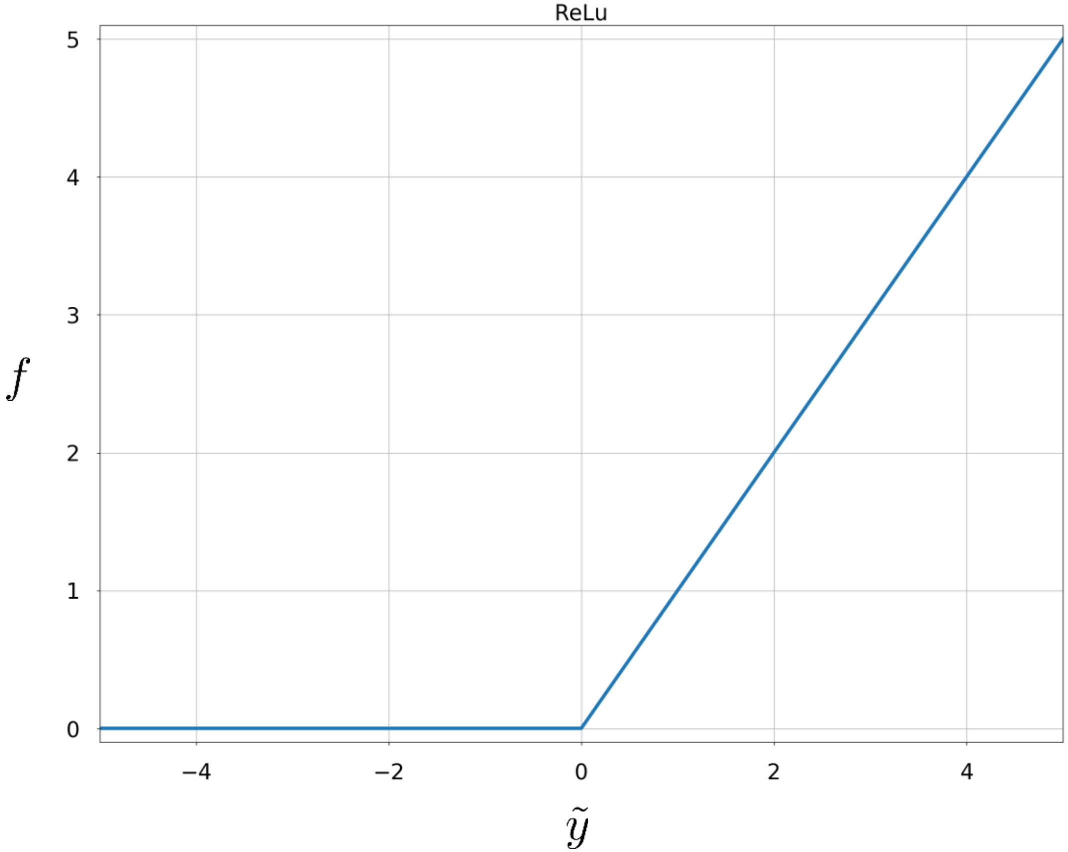
Sigmoid Function



Towards Artificial Neurons

The Neuron

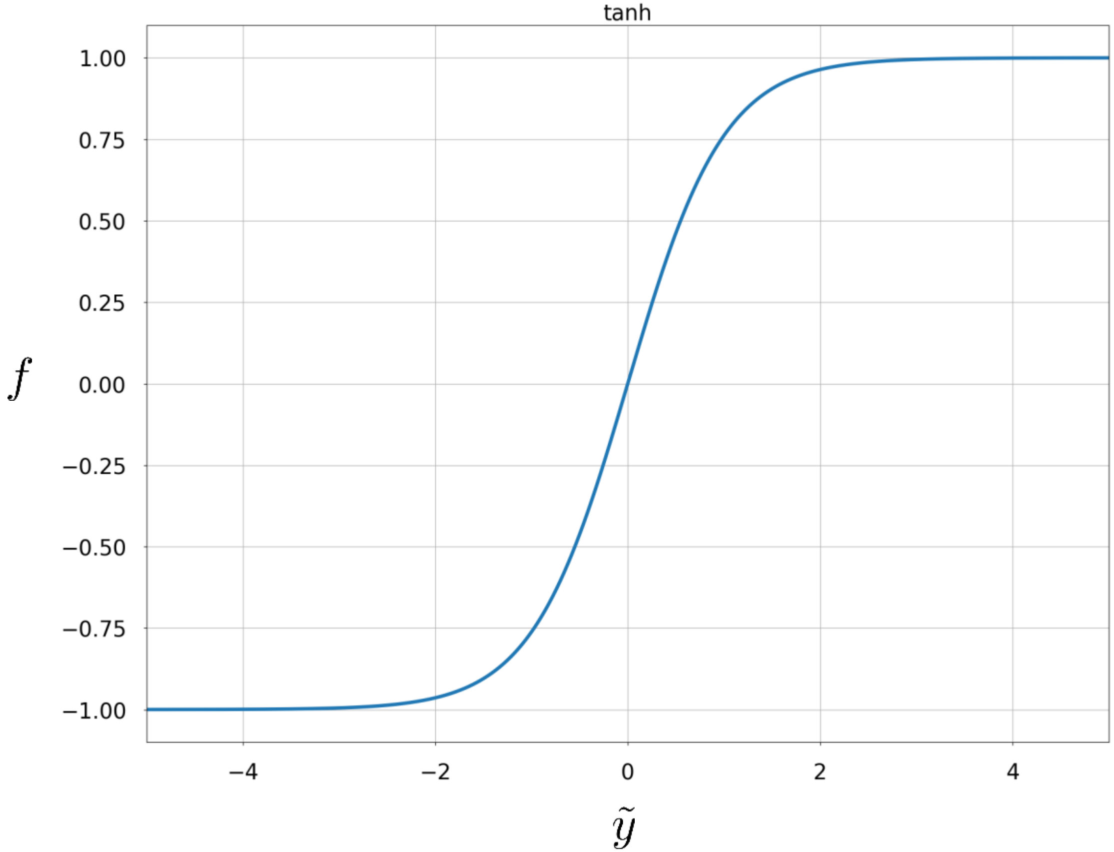
$$f = \begin{cases} \tilde{y} & \tilde{y} \geq 0 \\ 0 & \tilde{y} < 0 \end{cases}$$



Towards Artificial Neurons

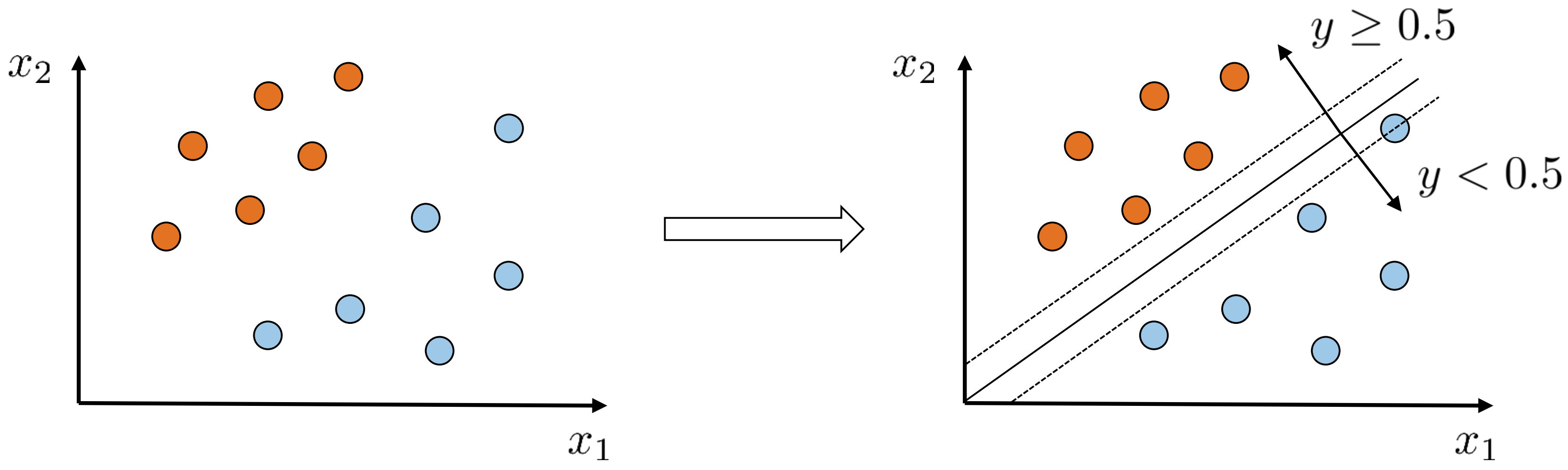
The Neuron

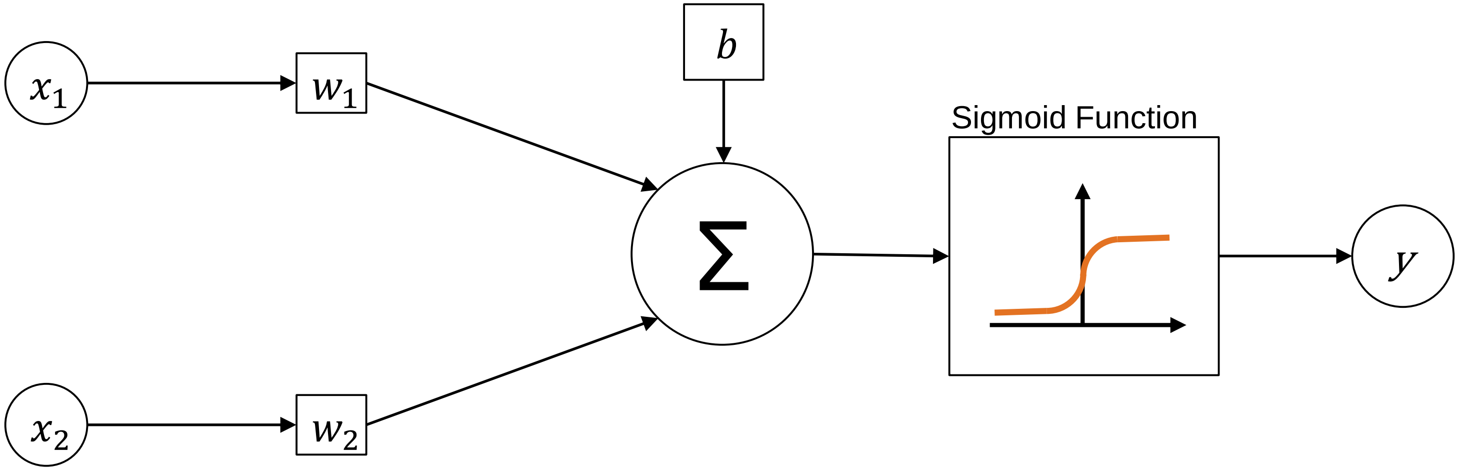
$$f = \tanh\left(\tilde{y}\right)$$



Towards Artificial Neurons

The Neuron

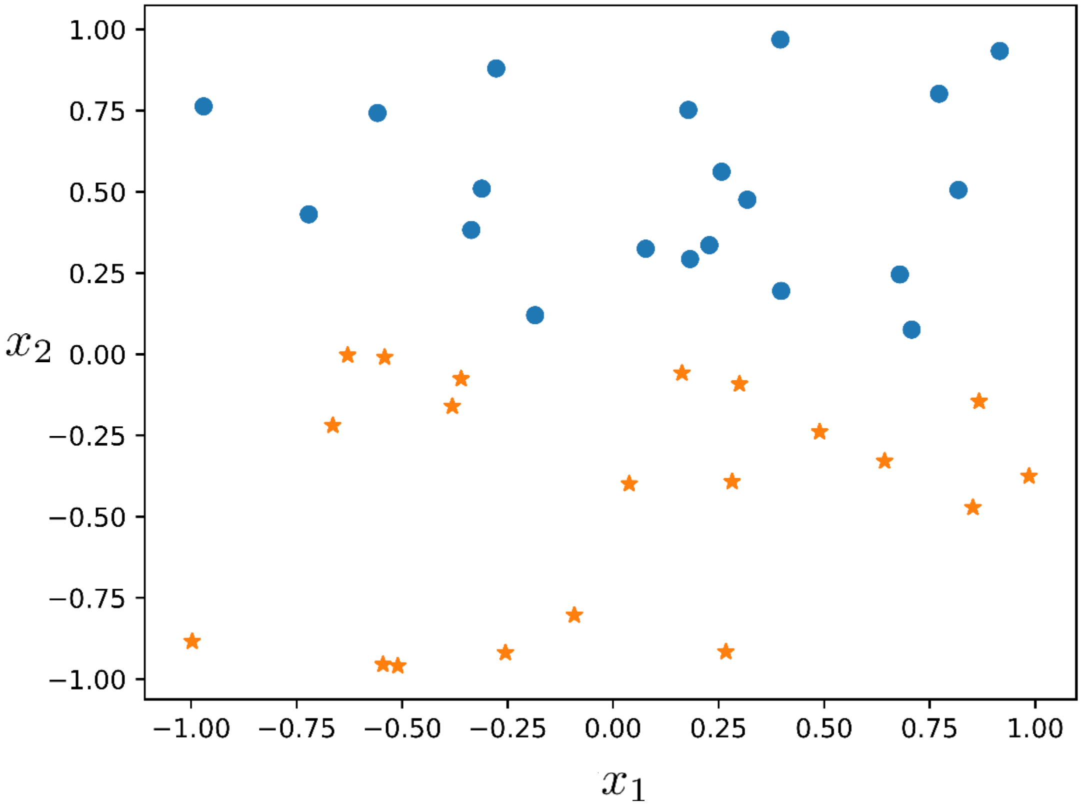




Towards Artificial Neurons

The Neuron

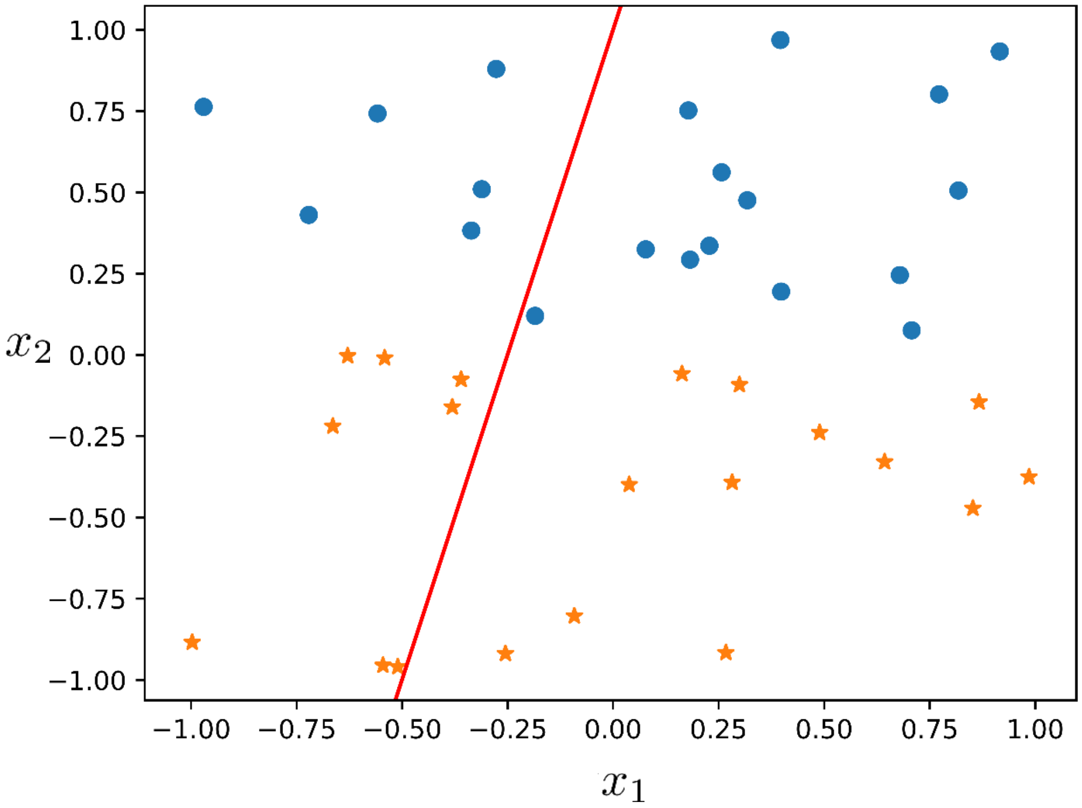
Input Data:



Towards Artificial Neurons

The Neuron

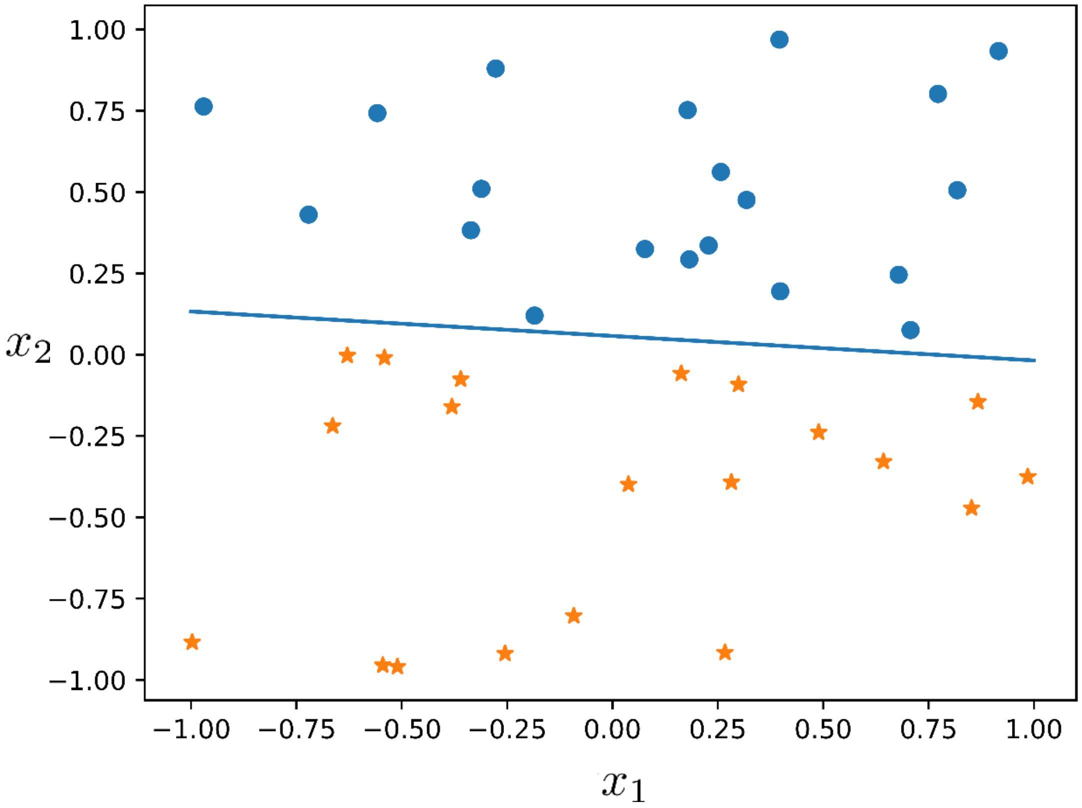
Initialization:



Towards Artificial Neurons

The Neuron

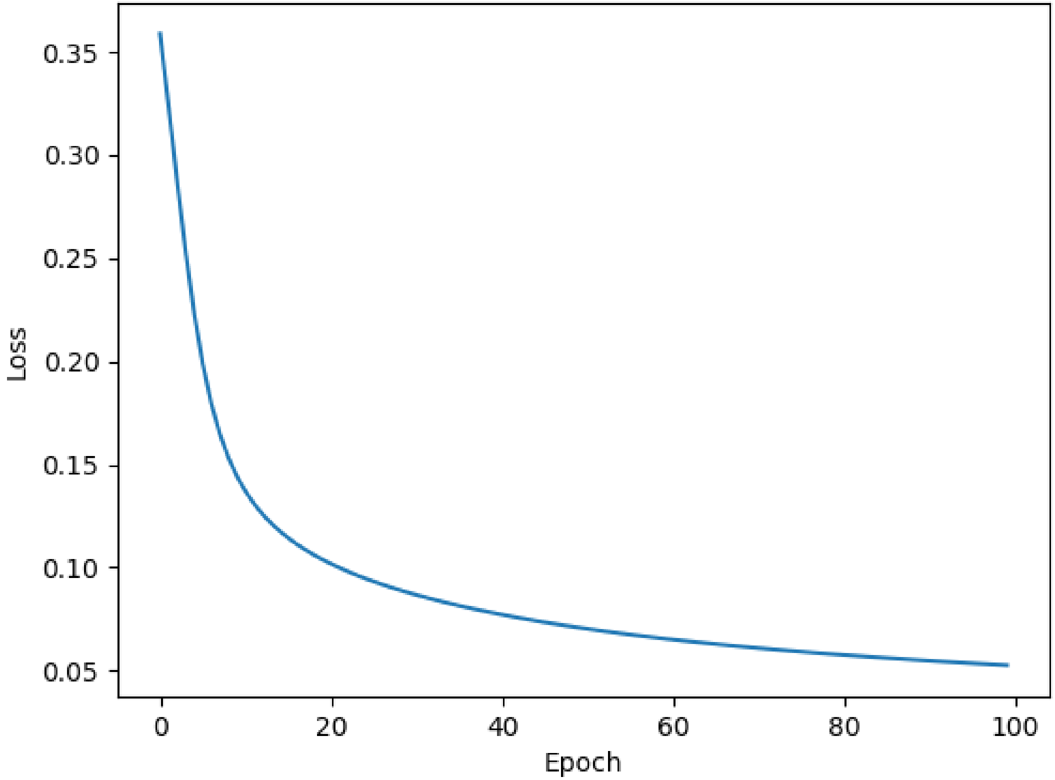
After Training:



Towards Artificial Neurons

The Neuron

Loss History:

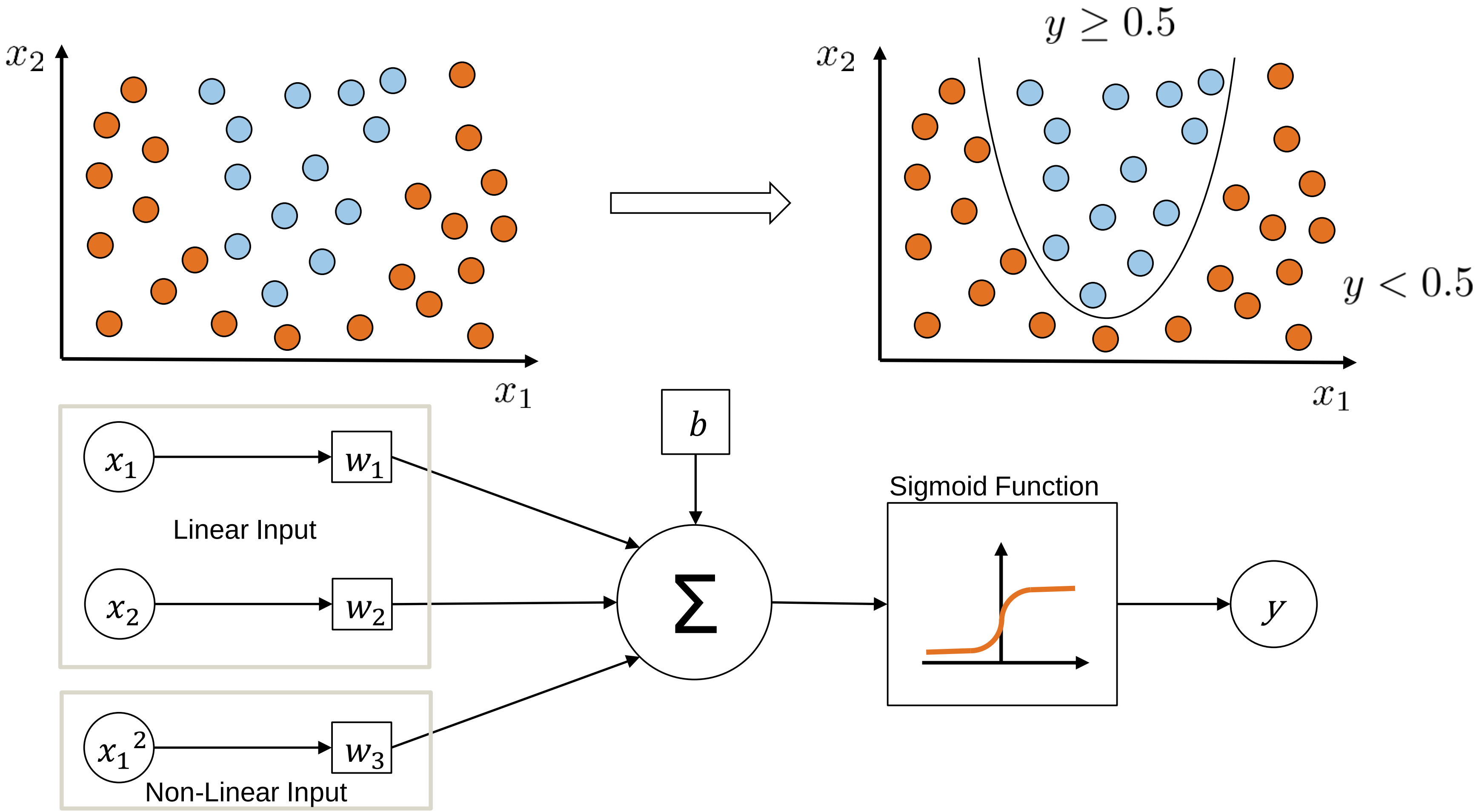


Epoch:

An epoch has passed when all training vectors have been used once to update the weights.

Towards Artificial Neurons

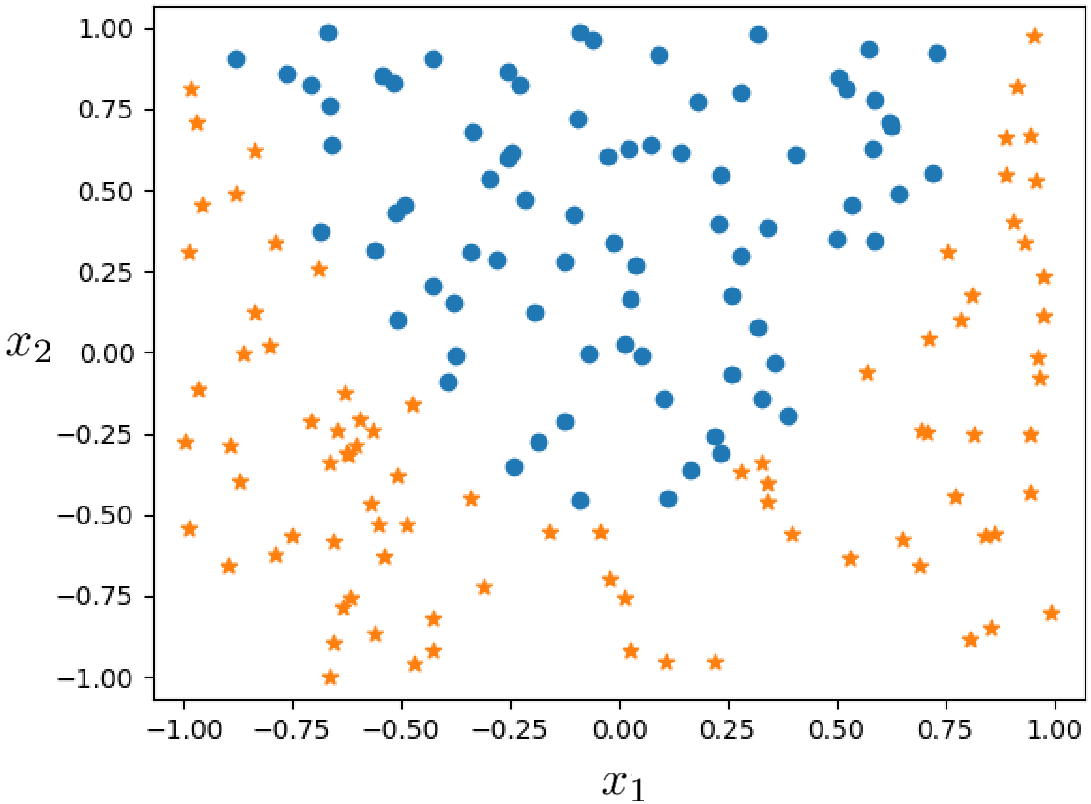
The Neuron



Towards Artificial Neurons

The Neuron

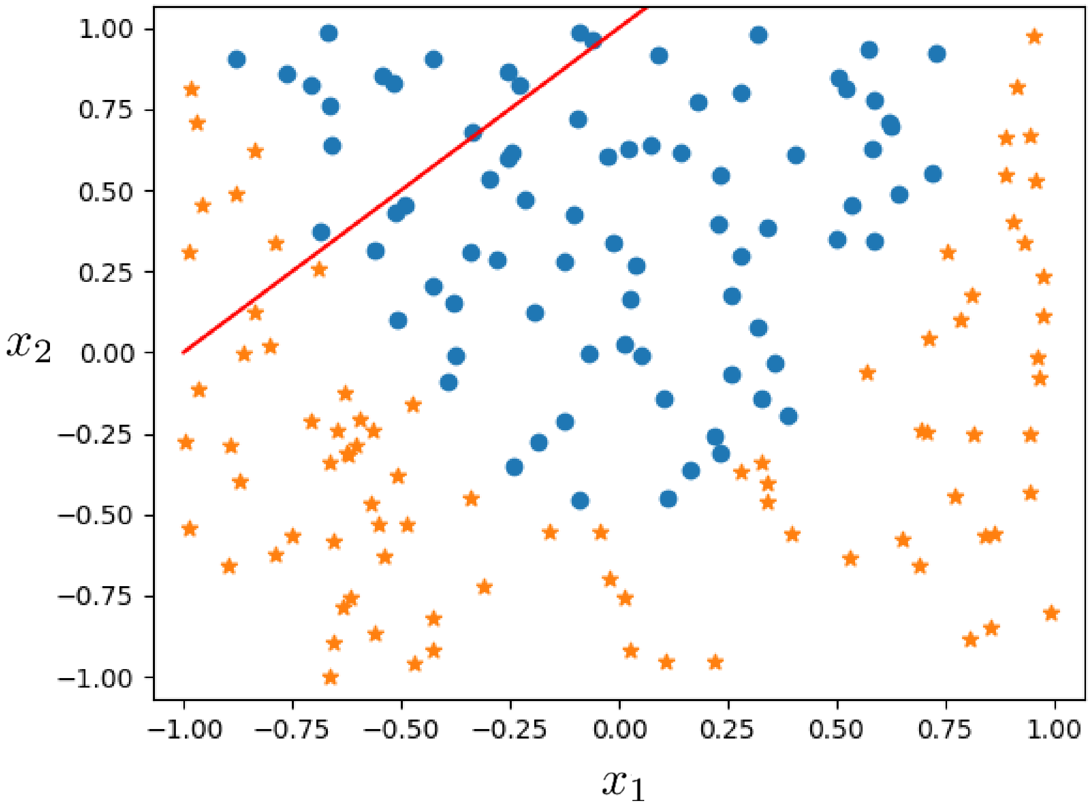
Input Data:



Towards Artificial Neurons

The Neuron

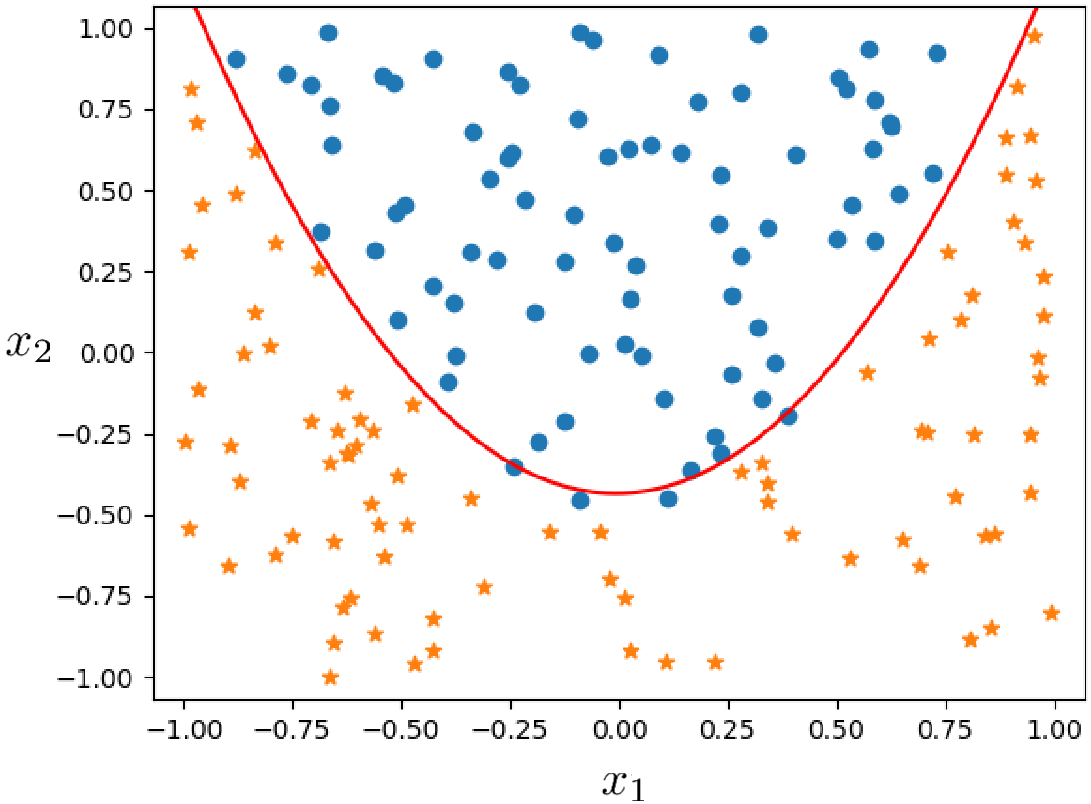
Initialization:



Towards Artificial Neurons

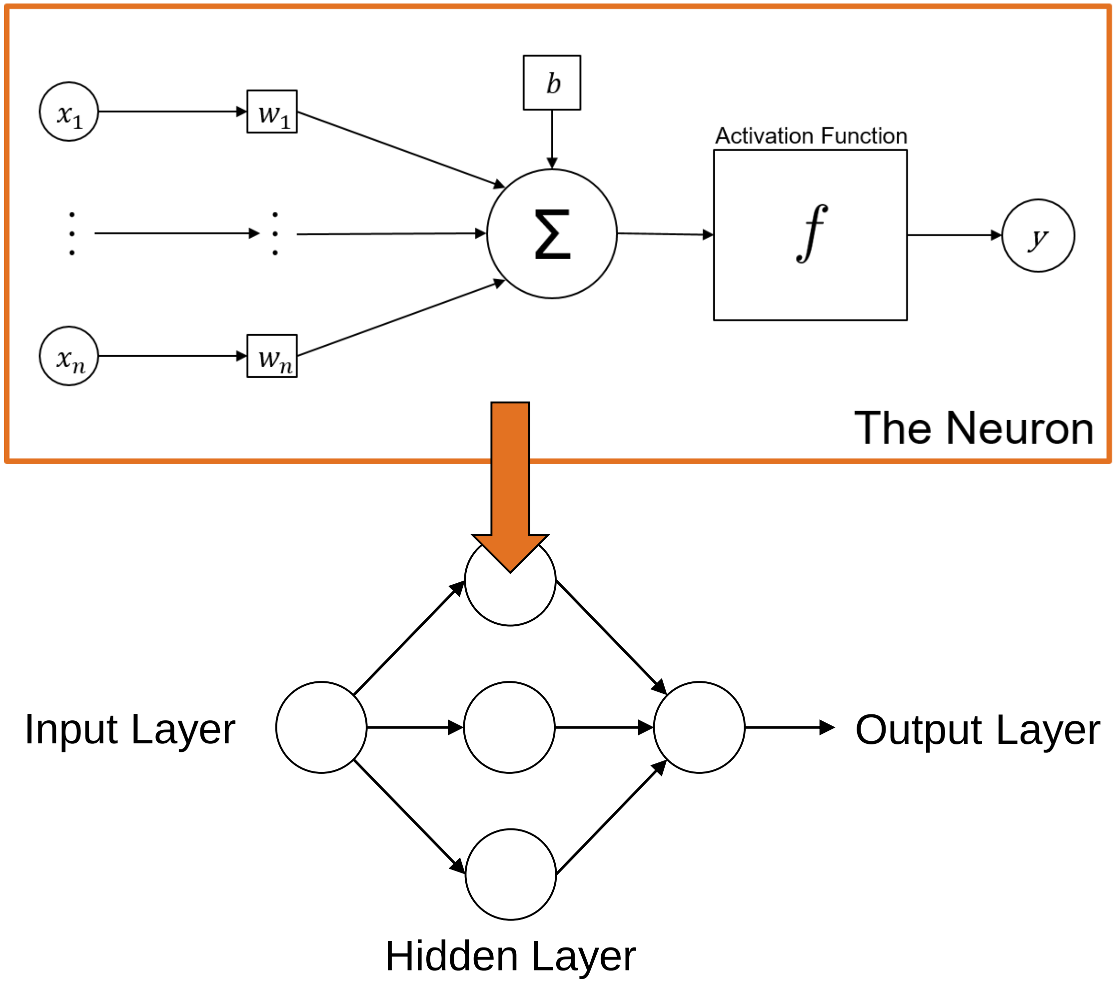
The Neuron

After Training :



Towards Artificial Neurons

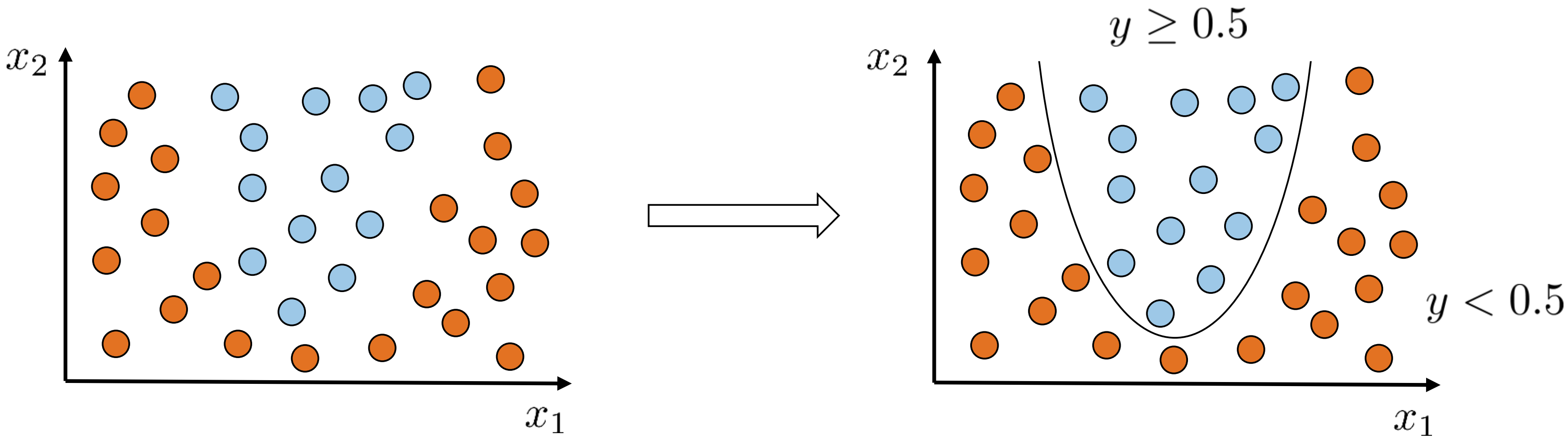
The Neuron

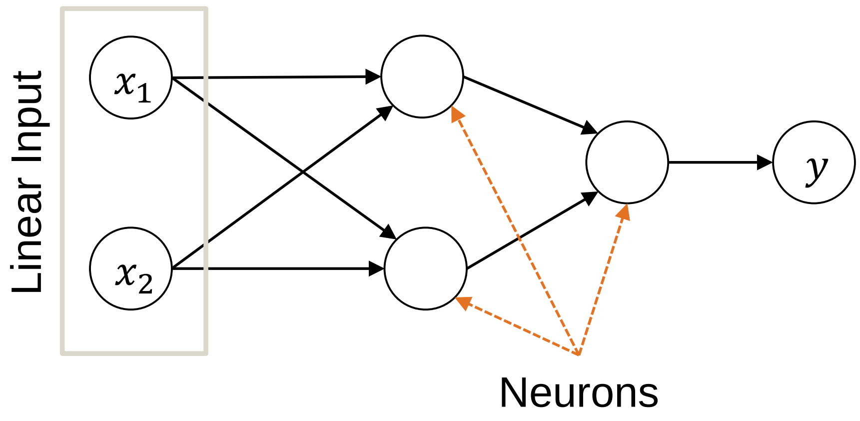


TUM

Towards Artificial Neurons

The Neuron





Net Properties:

Loss Function: Mean Squared Error

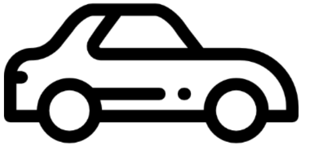
Activation Function: Sigmoid / Linear

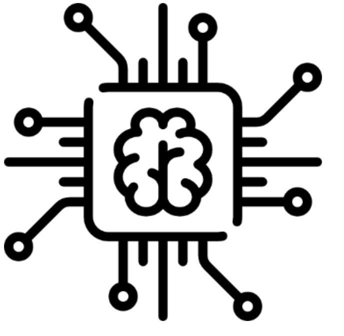
Optimizer: Gradient Descent

Introduction: Artificial Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Lennart Adenaw, M. Sc.)

Agenda

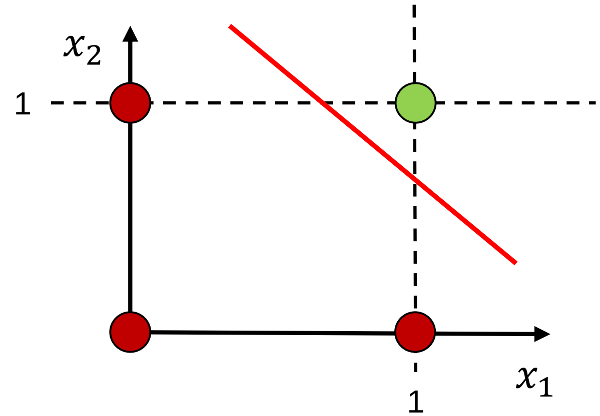


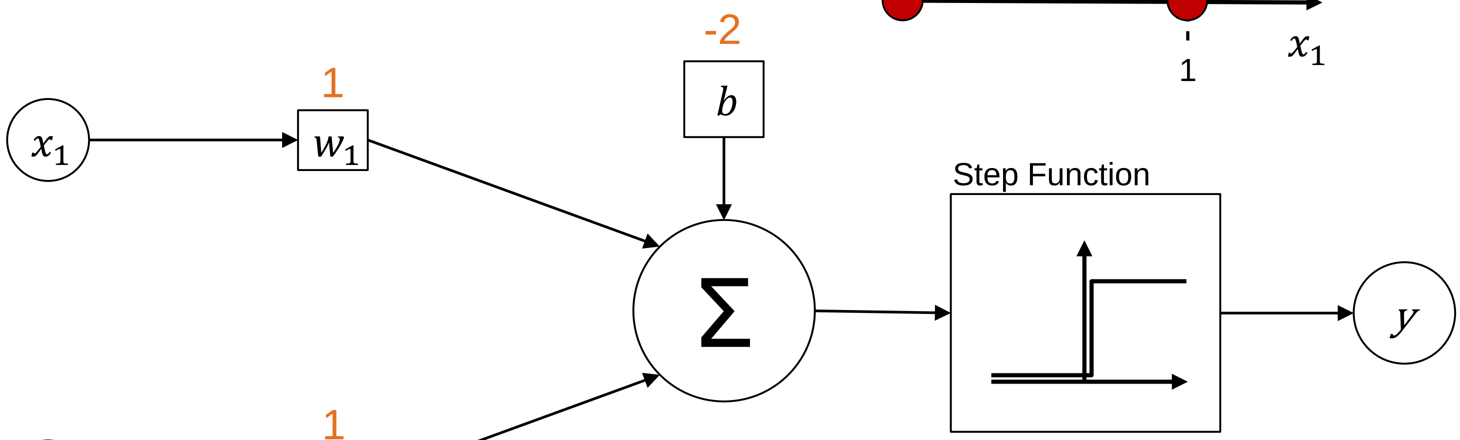


Multilayer Networks

Functional Completeness

Boolean Operators: AND



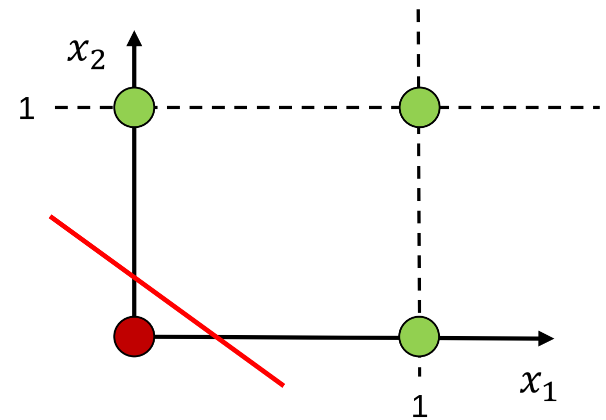


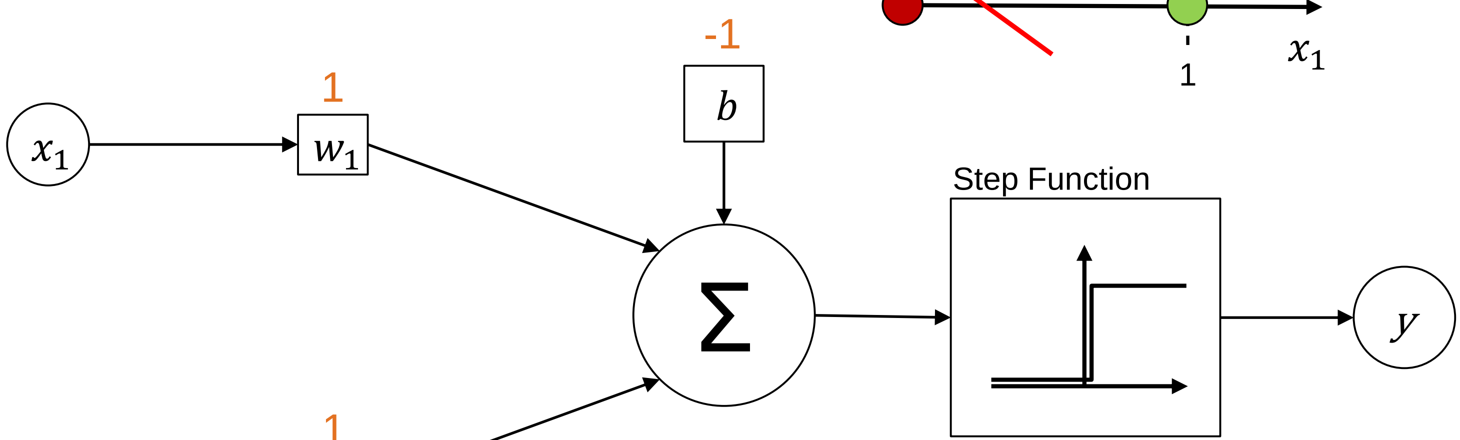
$$f = \begin{cases} 1 & \tilde{y} \geq 0 \\ 0 & \tilde{y} < 0 \end{cases}$$

Multilayer Networks

Functional Completeness

Boolean Operators: OR

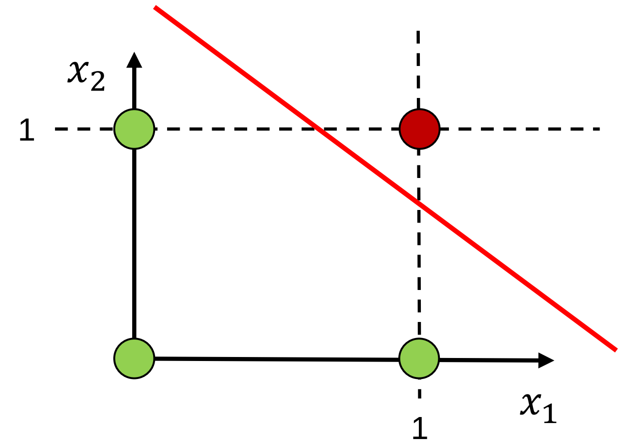


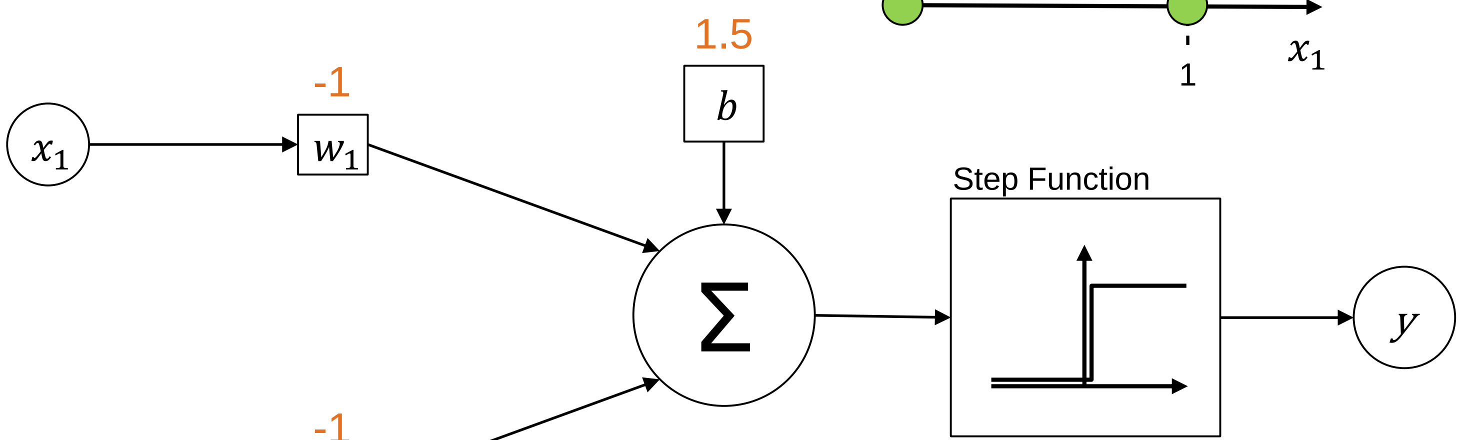


$$f = \begin{cases} 1 & \tilde{y} \geq 0 \\ 0 & \tilde{y} < 0 \end{cases}$$

Multilayer Networks
Functional Completeness

Boolean Operators: NAND

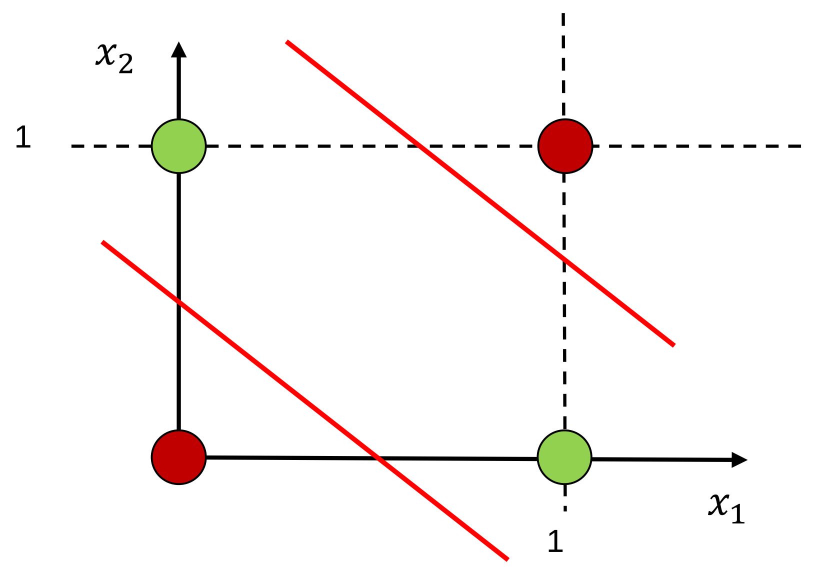




$$f = \begin{cases} 1 & \tilde{y} \geq 0 \\ 0 & \tilde{y} < 0 \end{cases}$$

Multilayer Networks
Functional Completeness

Boolean Operators: XOR

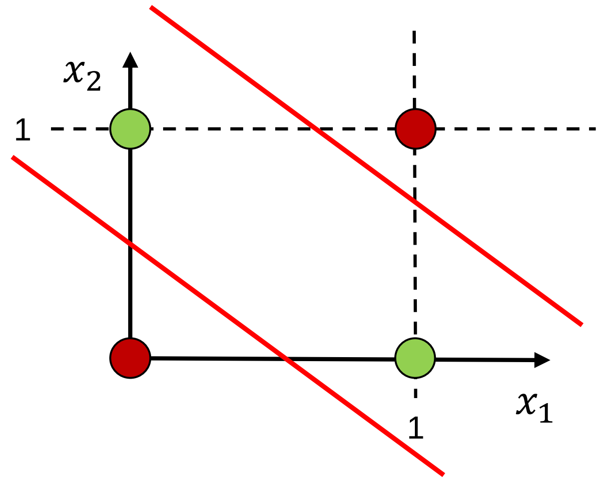


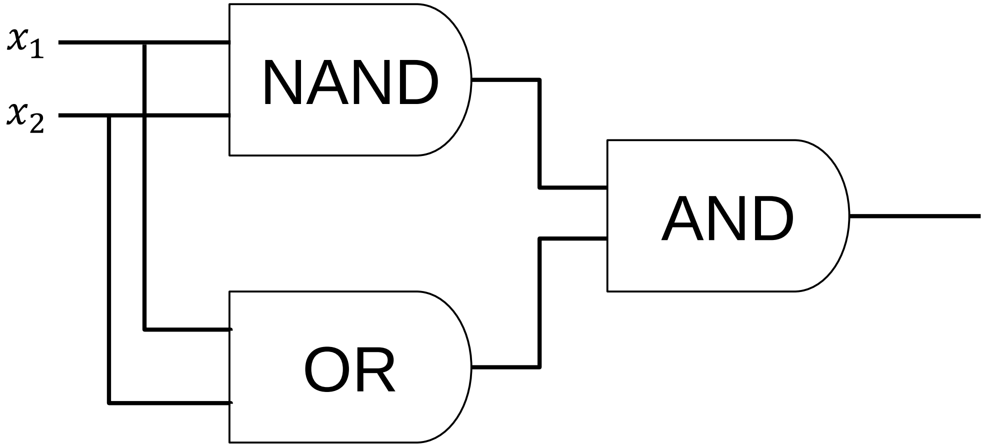
No linear separability

Multilayer Networks

Functional Completeness

Boolean Operators: XOR

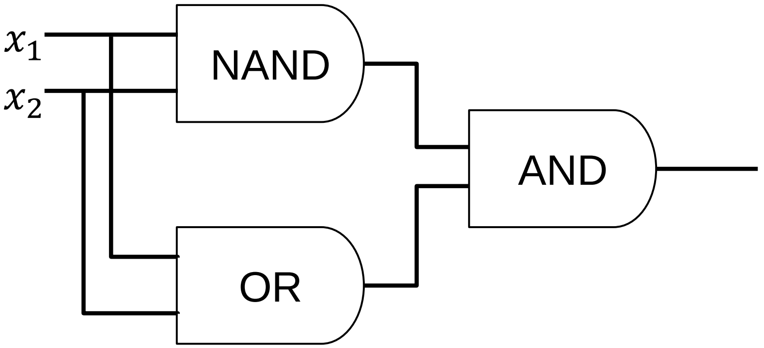


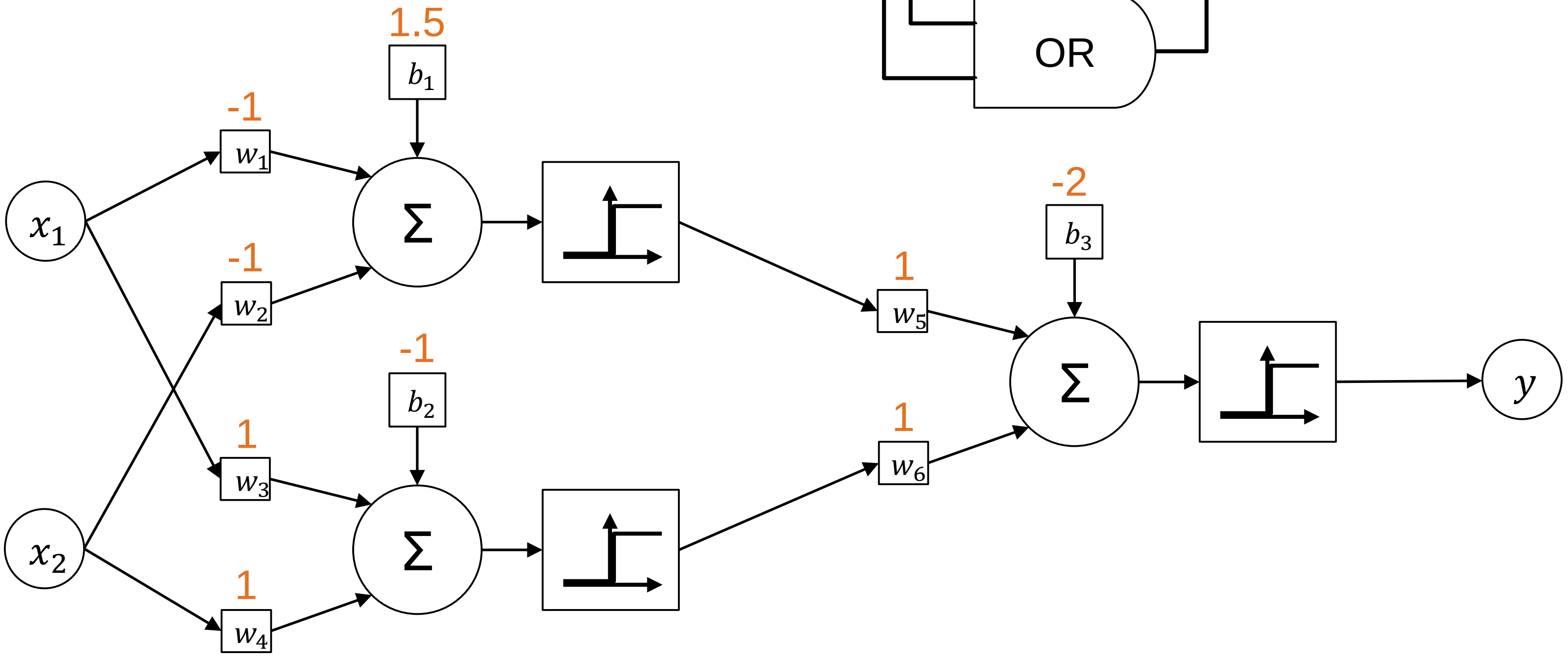


Multilayer Networks

Functional Completeness

Boolean Operators: XOR





Introduction: Artificial Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Lennart Adenaw, M. Sc.)

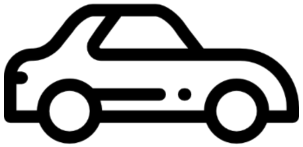
Agenda

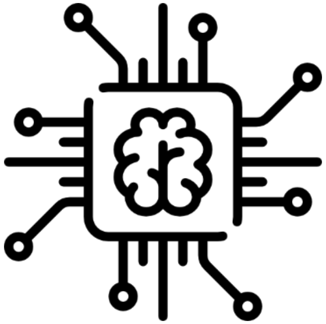
1. Chapter: Introduction

2. Chapter: Towards Artificial Neurons
    - 2.1 Linear Regression
    - 2.2 Gradient Descent
    - 2.3 The Neuron

3. Chapter: Multilayer Networks
    - 3.1 Functional Completeness
    - **3.2 MNIST Example**

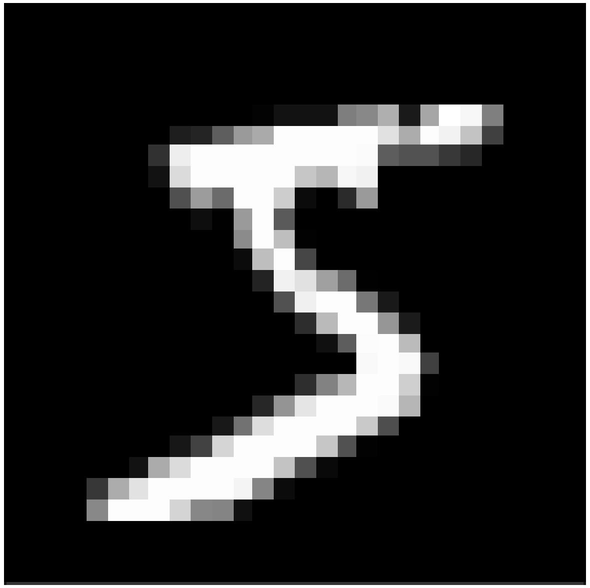
4. Chapter: Summary



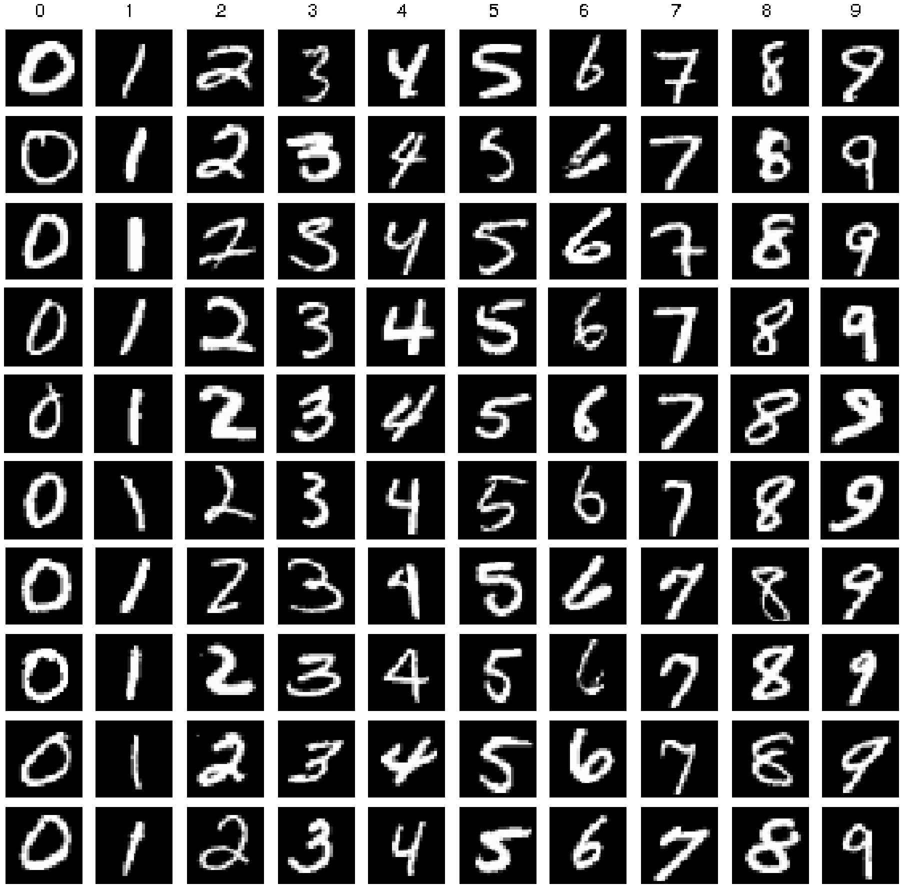


Multilayer Networks
MNIST Example

28x28 Grayscale



http://conx.readthedocs.io/en/latest/_images/MNIST_6_0.png



https://www.researchgate.net/publication/306056875_An_analysis_of_image_storage_systems_for_scalable_training_of_deep_neural_networks/figures?lo=1

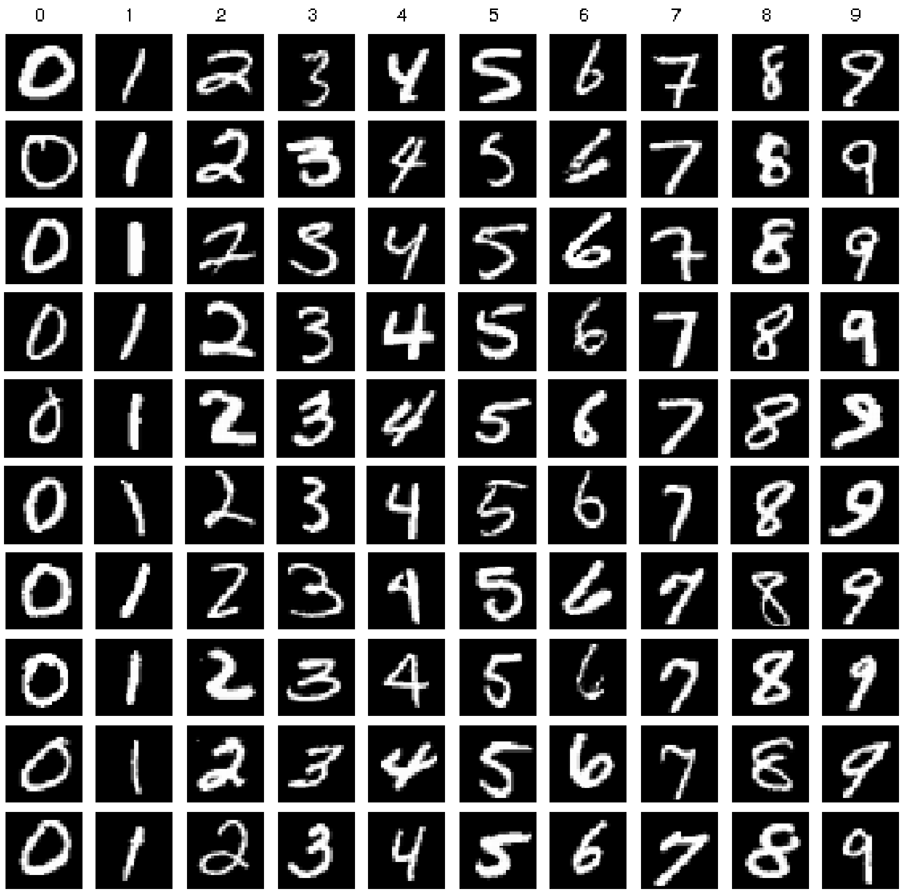
Multilayer Networks
MNIST Example

Properties:

- 60.000 handwritten numbers
- 28x28 pixels
- 0 to 255 grayscale
- Numbers 0 to 9

Task

Train a classifier that can identify the handwritten numbers

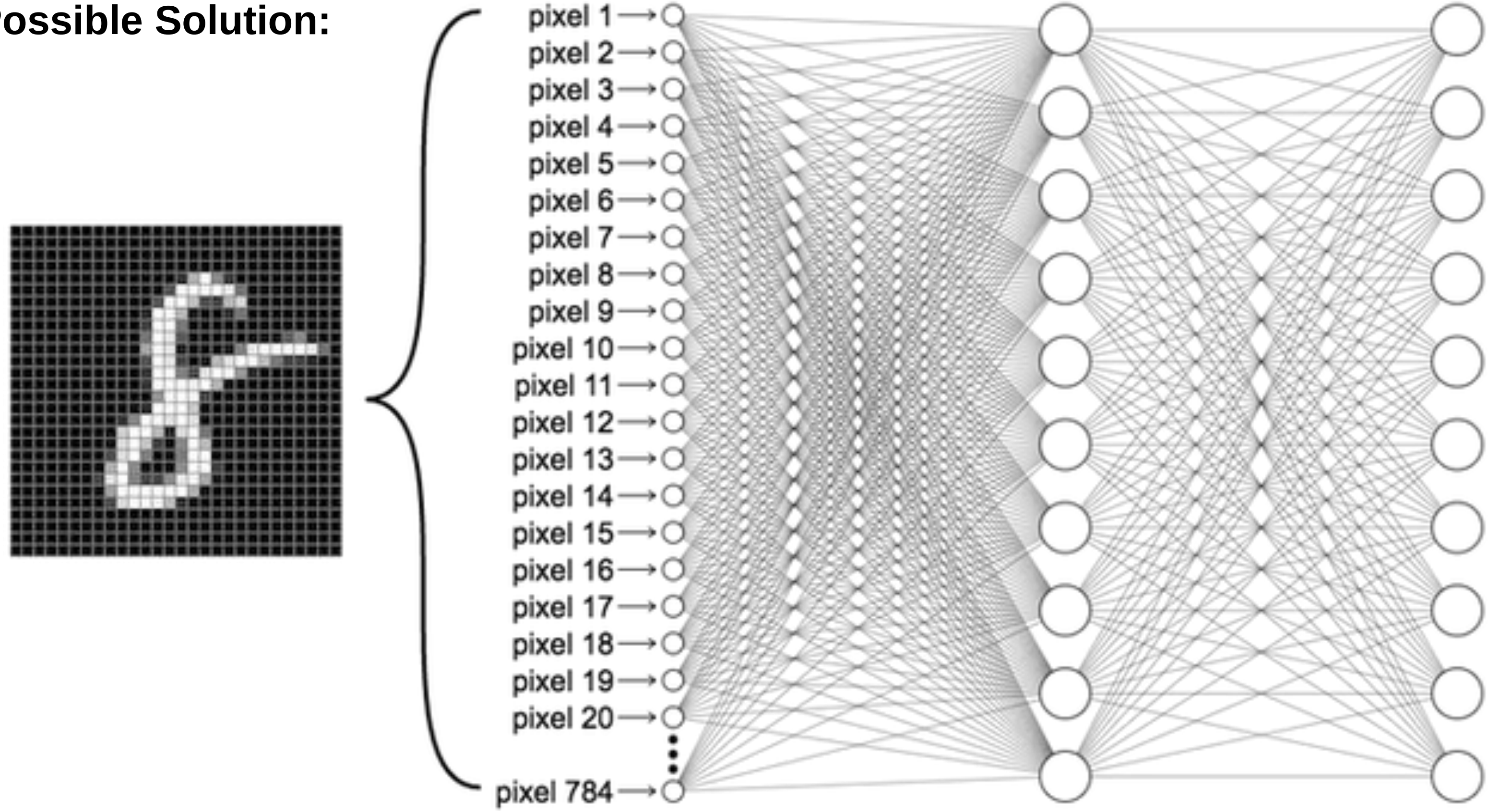


https://www.researchgate.net/publication/306056875_An_analysis_of_image_storage_systems_for_scalable_training_of_deep_neural_networks/figures?lo=1

Multilayer Networks

MNIST Example

Possible Solution:

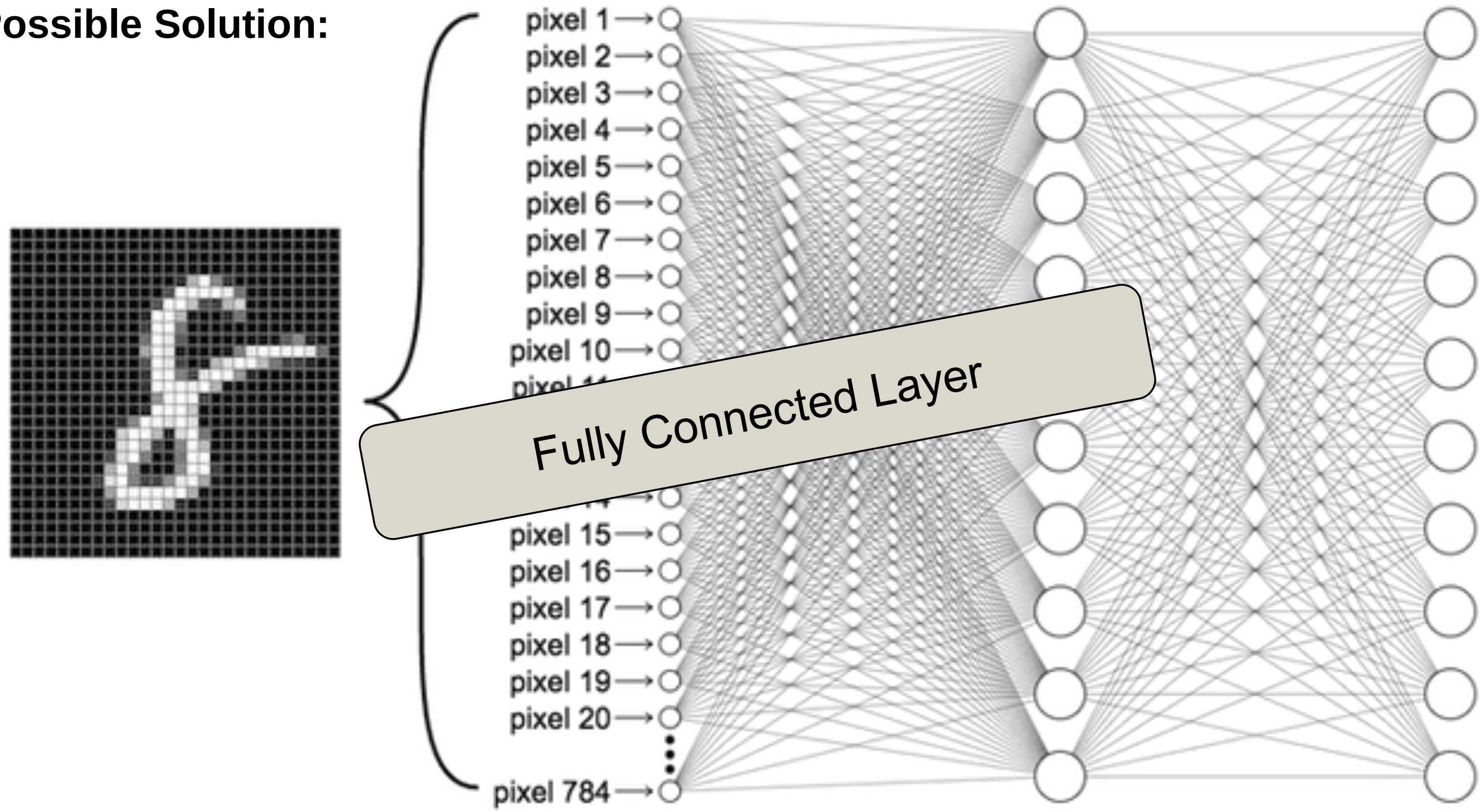


https://achintavarna.wordpress.com/2017/11/17/keras-tutorial-for-beginners-a-simple-neural-network-to-identify-numbers-mnist-data/

Multilayer Networks

MNIST Example

Possible Solution:

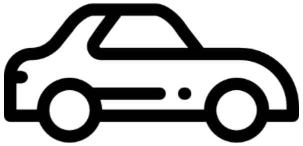


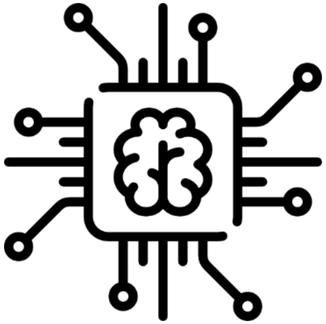
https://achintavarna.wordpress.com/2017/11/17/keras-tutorial-for-beginners-a-simple-neural-network-to-identify-numbers-mnist-data/

Introduction: Artificial Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Lennart Adenaw, M. Sc.)

Agenda





Summary

What we learned today:

Neural Networks are mathematical tools that can approximate any mathematical function

Gradient Descent is an approach suitable for weight adjustments

A single Neuron can perform Linear Regression and Binary Classification

Non-Linear, Multiple Classification and Regression is best performed by Neural Networks

Vocabulary and Ideas