DSci | Study & Research

Data Science Journey

Mathematics

Domain Knowledge — Computer Science

4 years 2 months have passed

since the start of the journey on 3 January 2020.

Inhaltsverzeichnis

Teil I

Artificial Intelligence

Kapitel 1

Artifical Neural Network

1 Objective

Reference: Watching Neural Networks Learn
An (ANN) is a *non-linear mathematical tool* intended to simulate the human brain process by using simple units called artificial neurons arranged in structures known as layers.

An ANN is a *universal function approximator*. They are machines that create functions.

This non-linear mathematical tool can also be understood as a *universal function approximator*. The *Universal Approximation Theorem* states that ANN with infinite neurons can approximate any function given the right weights. How to achieve this or how to come closer should be the objective of this segment. For example, the layer architecture, the activation function, and preparing the data are relevant topics to consider for finding the global minimum.

Different ways to look at it Many mathmatical field are in interplay when it comes to a ANN.

Linear Algebra is mostly used when it comes to computing information though the network. Graph Theory can also be used to to the same thing, but is mostly considered when it comes to analyzing the structure of the neural network.

Linear Algebra ANNs can be represented using matrices and vectors, where each neuron's input and output can be described as linear combinations of the inputs with corresponding weights. Matrix operations such as multiplication, addition, and transposition are fundamental in understanding how information flows through the network during forward and backward propagation.

Graph Theory ANNs can be represented as directed graphs, where neurons correspond to nodes and connections between neurons correspond to edges. Graph theory concepts are useful for analyzing the structure of neural networks, including connectivity patterns and network architectures.

Given a certain structure of the network *optimisation* with main tool from *calculus*: *gradient descent* is used to find the parameter that reduces the loss function.[1]

[1]While calculus and optimization are related and often used together in the context of machine learning and neural networks, they are distinct mathematical concepts with different focuses.

Optimization ANN are trained by optimizing a loss function with respect to the network's parameters. Optimization theory provides a framework for selecting appropriate optimization algorithms, tuning hyperparameters, and understanding convergence properties. While calculus provides the mathematical framework for computing gradients and understanding the behavior of optimization algorithms, optimization theory encompasses a wider range of techniques beyond calculus, such as convex optimization, linear programming, and heuristic algorithms.

Calculus (Optimization) Calculus is essential for optimizing the parameters (weights and biases) of ANN through techniques like gradient descent and its variants. Understanding derivatives is crucial for computing gradients, which indicate how the loss function changes with respect to changes in the network's parameters. Calculus helps in finding the direction and rate of change that minimizes or maximizes a function, which is crucial for training neural networks to minimize the loss function.

The field of *probability and statistics* as well as *information theory* are used for multiple aspects of ANNs. For example the former can be used to preparing or preprocessing of data, model evaluation and bringing into context the uncertainty in predictions. The later will also be used for finding the features in the dataset as well as analysing the information flow inside a ANN.

Probability and Statistics ANNs can be viewed as probabilistic models, especially in the context of Bayesian neural networks, where uncertainty is explicitly modeled. Statistical techniques are used for data preprocessing, model evaluation, and uncertainty estimation in predictions.

Information Theory Information theory concepts, such as entropy and mutual information, can be used to analyze the information flow within neural networks. Compression techniques inspired by information theory, such as autoencoders, are used for feature extraction and dimensionality reduction.

The field of *functional analysis* provides the tools to analyze the natrue of the universal function approximator, called ANN.

Functional Analysis ANNs can be viewed as function approximators, and functional analysis provides tools for analyzing the properties of these approximations. Understanding function spaces and convergence properties is relevant when studying the expressive power and generalization abilities of neural networks.

Landscape of different Architectures There are different kinds of ANN, for example: convolutional, recursive. This segment will focus on *feed-forward* networks. A Feed-Forward is usually a non-linear regression or classification with a sigmoidal activation and a multilayer perceptron.

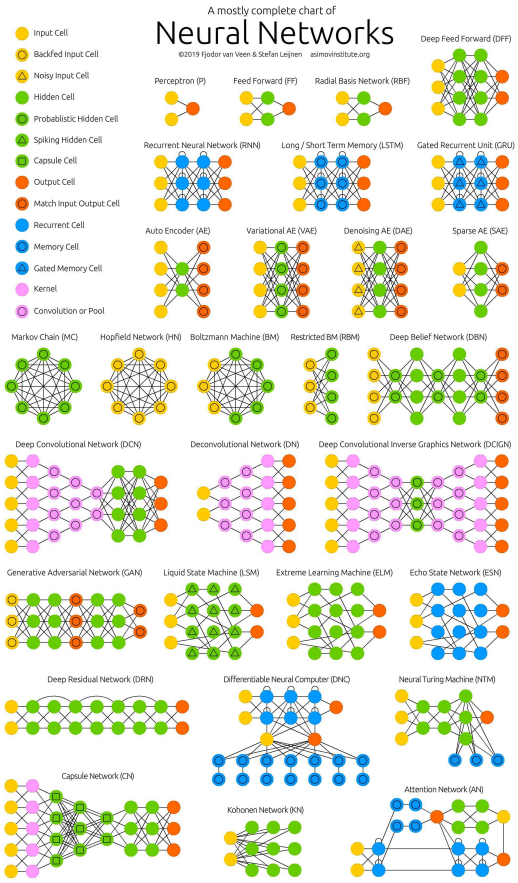

Abbildung 1.1: Types of artificial neural networks.

2 Multilayer Perceptrons (MLP)

Mathematical Foundations Reference, siehe auskommentierte Link und dem Buch Andriy, 2018[93]

ANN: Nonlinear Mathematical Tools as Universal Function Approximators

Each node combines an *affine linear function* and a nonlinear *activation function*.

ANNs are nonlinear mathematical tools[2], intended to mimic human brain processes by stacking simple neurons together. The simplest unit of an ANN is called a *Neuron*. They are structured in *layers*. The learning process for humans is called training, while for ANNs it is referred to as training.
A layer can be represented as a *parameterized function*. From the perspective of a directed graph, each node can be represented as an *affine linear function*. The combination of each layer can be understood as a new *affine linear function* [for a given neuron].
The nonlinear quality of an ANN arises from the *activation function*.

Transformer Function Given the input x^t[3]. These are processed in a *neuron* through *weights* $w_i, i = 1, \ldots, n$, producing z, which then produces the final output y through some

[2]However, there is a possibility to design it so that it can be a linear function.

[3]The initial input of the layer can be understood as *features*, which can be represented as a vector $x^t = (x_1, \ldots, x_n)$

function ϕ, $\phi(z) = y$.
The output z is the result of processing the $n - x_i$ weightily:

$$z = (w_1, \ldots, w_n) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + b \tag{2.1}$$

$$= \sum_{i=1}^{n} x_i \cdot w_i + b = w^t x + b, \tag{2.2}$$

where b is called *bias*. This should give the system more flexibility by imitating a human filter. This *affine function* can be changed into a linear function, where b is set as w_0 and $x_0 = 1$:

$$z = (w_0, \ldots, w_n) \begin{pmatrix} x_0 = 1 \\ \vdots \\ x_n \end{pmatrix} \tag{2.3}$$

$$= \sum_{i=0}^{n} x_i \cdot w_i = w^t x. \tag{2.4}$$

To avoid the notation of the transpose, it can be represented by $w \cdot x$.
However, depending on the needs, these processes can be viewed as affine or linear functions. In either case, processing the input x_i resulting in z is called a **transfer function**. Depending on the needs different transformers can be used. The most popular or most used is the sum operator $\sum$:

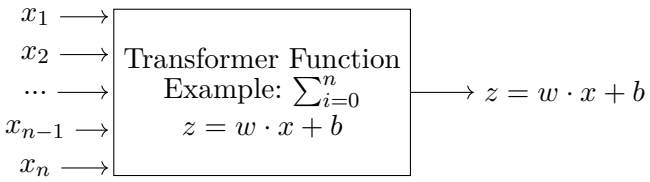

Abbildung 1.2: One perceptron, aka neuron

Activation Function The function $y = \phi(z)$ is known as the *activation function* and it is this function that is resonsible for the non-linearity of the process. The function can be "freely" selected, if it is fullyfilling certain criteria.

Each neuron can be regared as a mathematical function that is changing the *transformation function* and the *activation* with the inputs togehter: $y = \phi(z) = \phi(wx + b)$:

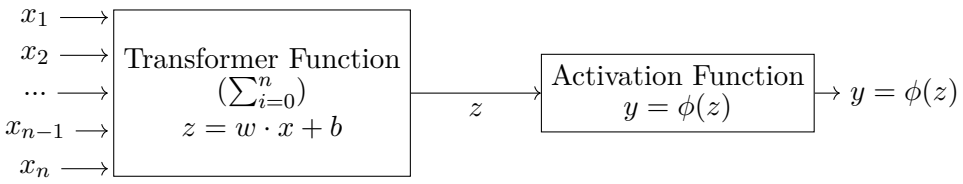

Abbildung 1.3: One perceptron, aka neuron

Feed-Forward As discussed before, the entirety of ANNs can be understood as mathematical functions with an input x and an output y.

$$f_{NN}(x) = y \tag{2.5}$$

To combine each neuron, the output of each neuron is sent to all neurons in the next layer. This forwarding of information is called *feed-forward*. From the perspective of a function, this is called *verschachteln*. Each neuron in each layer (hidden layer) receives the output of the previous neurons. This neuron is then called a Multilayer Layer Perceptrons (MLP) .

Definition I.1: Multilayer Perceptron

A MLP is a function $f_l : \mathbb{R}^n \to \mathbb{R}$. It is called an $n - L - m$ perceptron, with n inputs, L hidden layers, and m outputs.

$$y = f^{(l)}(\vec{x}) = f_{l-1}\left(f_{l-2}\left(\dots\left(f_0(\vec{x})\right)\right)\right). \tag{2.6}$$

Where $f^{(i)}(\vec{z}) \overset{def}{=} \phi_l\left([W]^{(l)}\vec{z} + b^{(l)}\right)$.

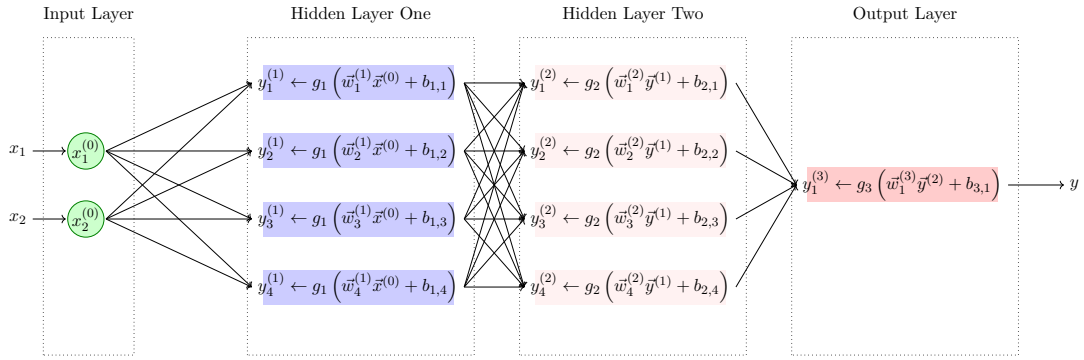

Abbildung 1.4: MLP with two hidden layer, one input vector and one output layer

In this example, the output of the MLP $y_1^{(1)}$ is calculated by the activation function g_1 for the hidden layer 1. The weight vector $\vec{w}_1^{(0)}$ has dimension 2, as it receives two inputs $x_1^{(1)}$ and $x_2^{(1)}$, along with the scalar bias $b_1^{(1)}$. In the next layer, the dimension of each weight vector is 4, as each MLP receives inputs $y_1^{(1)}$, $y_2^{(1)}$, $y_3^{(1)}$, and $y_4^{(1)}$ from the previous layer.

3 Curve fitting

Idea The ANN fittes it's internal parameter to that degree, that the training data given the parameter of the ANN produce an output that "fit's" the required output.

For example, the training data output or target values $\vec{y}$ are representing a sinus curve. The ANN produces linear output.

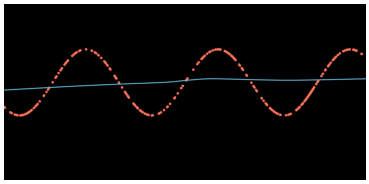

Abbildung 1.5: Curve fitting inital start

After some iteration of weights by tthe process *backpropagation* (General term *gradient descent*) the ANN reduces the loss value. The ANN "fits" more the target values.

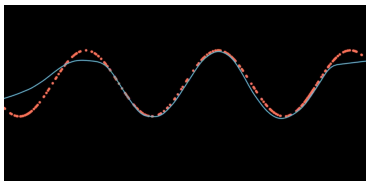

Abbildung 1.6: Curve fitting after some iteration

This process is generalisable and therefore not specific to any function, this is derived from the Universal approximation theorem.

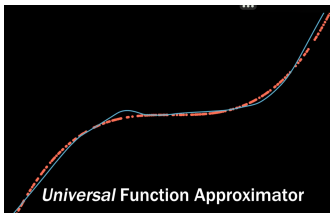

Abbildung 1.7: A Neural Net as universal function approximator $f(x) \approx NN(x)$

3.1 Gradient Descent

3.1.1 Optimization Using Gradient Descent

We now consider the problem of solving for the *minimum* of a real-values function

$$\min_x f(x). \tag{3.1}$$

We assume

- $f : \mathbb{R}^4 \to R$ captures the machine learning problem at hand,
- f is differentiable,

- we are unable to analytically find the solution in closed form.

The *gradient descent* takes steps a proportionaly (γ) to the negative of the gradient of the function ∇f at the current point x_0:

$$\nabla x_1 = x_0 - \gamma((\nabla f)(x_0))^T. \tag{3.2}$$

At the point x_0 the function $f(x_0)$ degresses fastes if on moves from x_0 in the direction of the negative gradient $-\gamma((\nabla f)(x_0))^T$ of f at x_0. From this follows: For a small *step-size*[4] $\gamma \geq 0$ $f(x_1) \geq f(x_0)$.

The Simple Gradient Descent Algorithm To find the local optimum f_{x_*} we start with a inial guess x_0 of the parameter we wish to optimize. Then we iterate according to

$$\nabla x_{i+1} = x_i - \gamma_i((\nabla f)(x_i))^T. \tag{3.3}$$

For a suitable step-size γ_i, the sequence $f(x_0) \leq f(x_1) \leq ...$ converges to a *local* minimum.

Graphical Gradient Descent Only if the step-size γ is a big, that the $f(x_{i+1}$ overshots the local minimum, the functional value $y = f(x_i)$ converges to the local minimum.

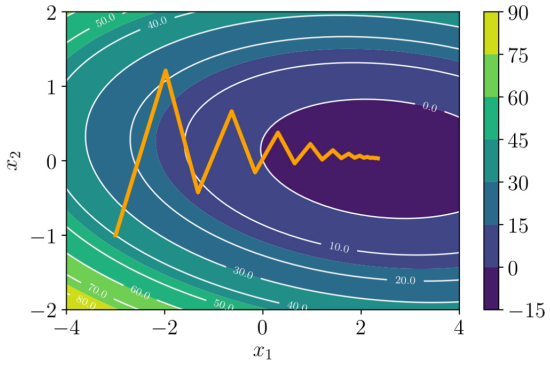

Abbildung 1.8: Graph shows the gradient descent on a two dimensional quadratic surface.

The zig-zag of the graph

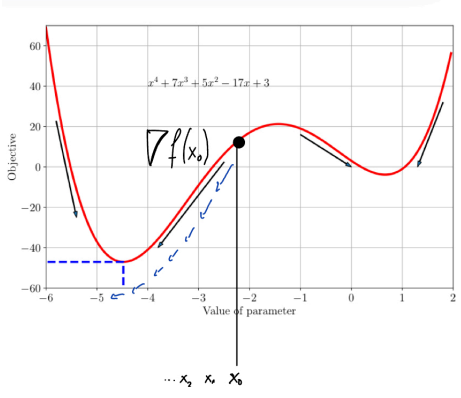

Abbildung 1.9: Example of a one dimentional function. Negative gradient descent is indicated by the arrows.

[4]In Azure ML is called *learning-rate*

3.1.2 Solving a Linear Equation System

In Form Ax=b

Mathematical Fields	Featurization	Data Processing	Model Architecture	Optimization Parameter	Evaluation	General Assessment
Linear Algebra			ANN represented as matrices and vectors; Input and Output linear combination			
Graph Theory			ANN represented as directed graphs; Analyzing the structure of the network			
Information Theory	Compression techniques: autoencoder; dimensionality reduction					
Optimization (e.g., Calculus)				Techniques like gradient descent and weights and biases		
Probability Theory (Stochastic)		Techniques for data processing steps	In case of a BNN		Model evaluation; uncertainty estimation in predictions	
Functional Analysis						Understanding the expressive power/general ability

Tabelle 1.1: Workflow Diagram of Mathematical Perspectives in ANN

Teil II

Anhang

14

Anhang A

Abkürzungsverzeichnis

Symbole

.ipynb Jupyter Notebook file format which in interoperable accross many platforms. The name also referes to the user-facing web interface called Jupyter Notebook.. *Glossar:* Jupyter Notebook (Web Interface)
.json JavaScript Object Notation. *Glossar:* JSON

A

ANN Artificial Neural Network. 4–6, 8, 9

E

EC2 Amazon Elastic Computing Cloud. *Glossar:* Amazon Elastic Computing Cloud

H

HTTP Hypertext Tranfer Protokoll. *Glossar:* HTTP

J

JSON JavaScript Object Notation. *Glossar:* JSON

M

MLP Multilayer Layer Perceptrons. 8

N

Na Not available. *Glossar:* Na (R)
NaN Not a Number. *Glossar:* NaN

O

O(OKR) Objective form the OKR logic. *Glossar:* Objective (OKR)
ODBC Open Database Connectivity (Connection). *Glossar:*

S

SQL Structured Query Language. *Glossar:* SQL

U

URL Uniform Resource Locator. *Glossar:* URL

Anhang B

Glossar