



Deep Learning for Language Analysis

Deep Learning Introduction – Neural Network Basics

Institut für Digital Humanities | Johanna Binnewitt, Valmir Etemi | 06.09.2019

Introduction

Perceptron

Multi Class Perceptron

Multi Layer Perceptron

Summary

Introduction

Perceptron

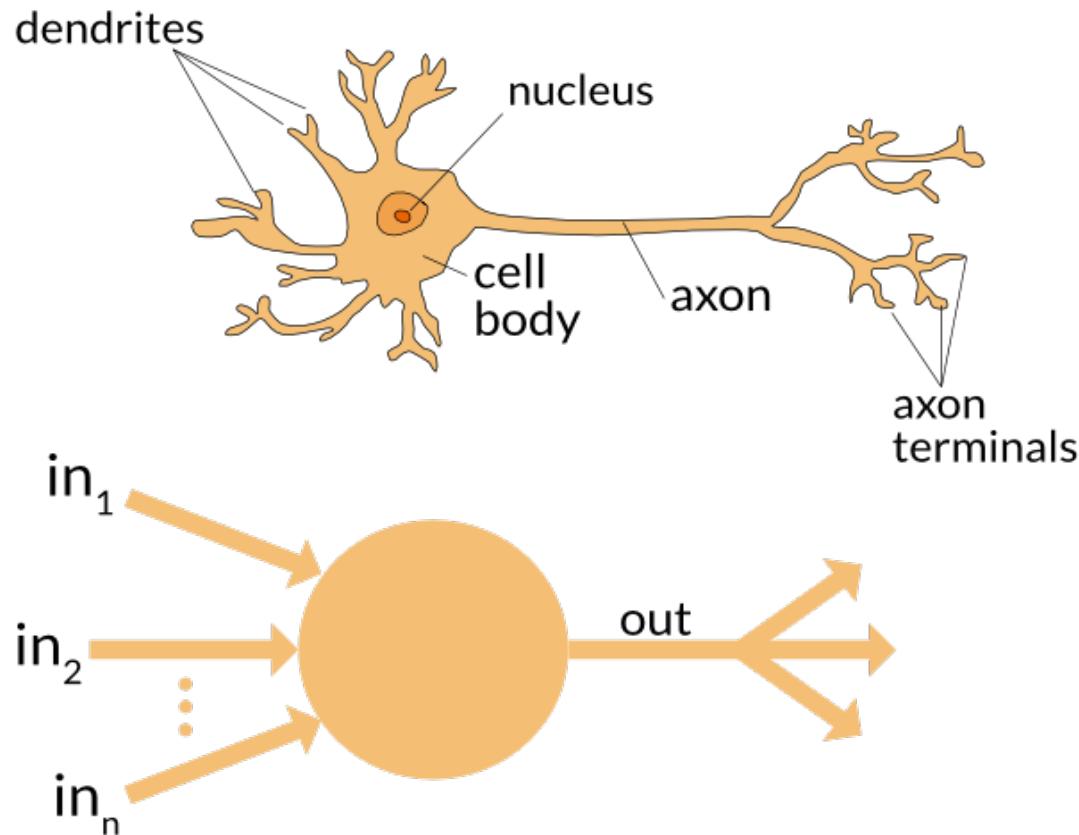
Multi Class Perceptron

Multi Layer Perceptron

Summary

Introduction

Biological Inspiration



Source: <https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7>

Introduction

Brief History

- > 1943 First Perceptron Model developed
- > 1960s AI was considered to replace all human activity
- > But too less data and low hardware stopped early progression
- > 1990s starting to grow again due to more available data, better hardware and more efficient networks

- > Is this only the next hype of AI?
- > We'll see...

Introduction

Perceptron

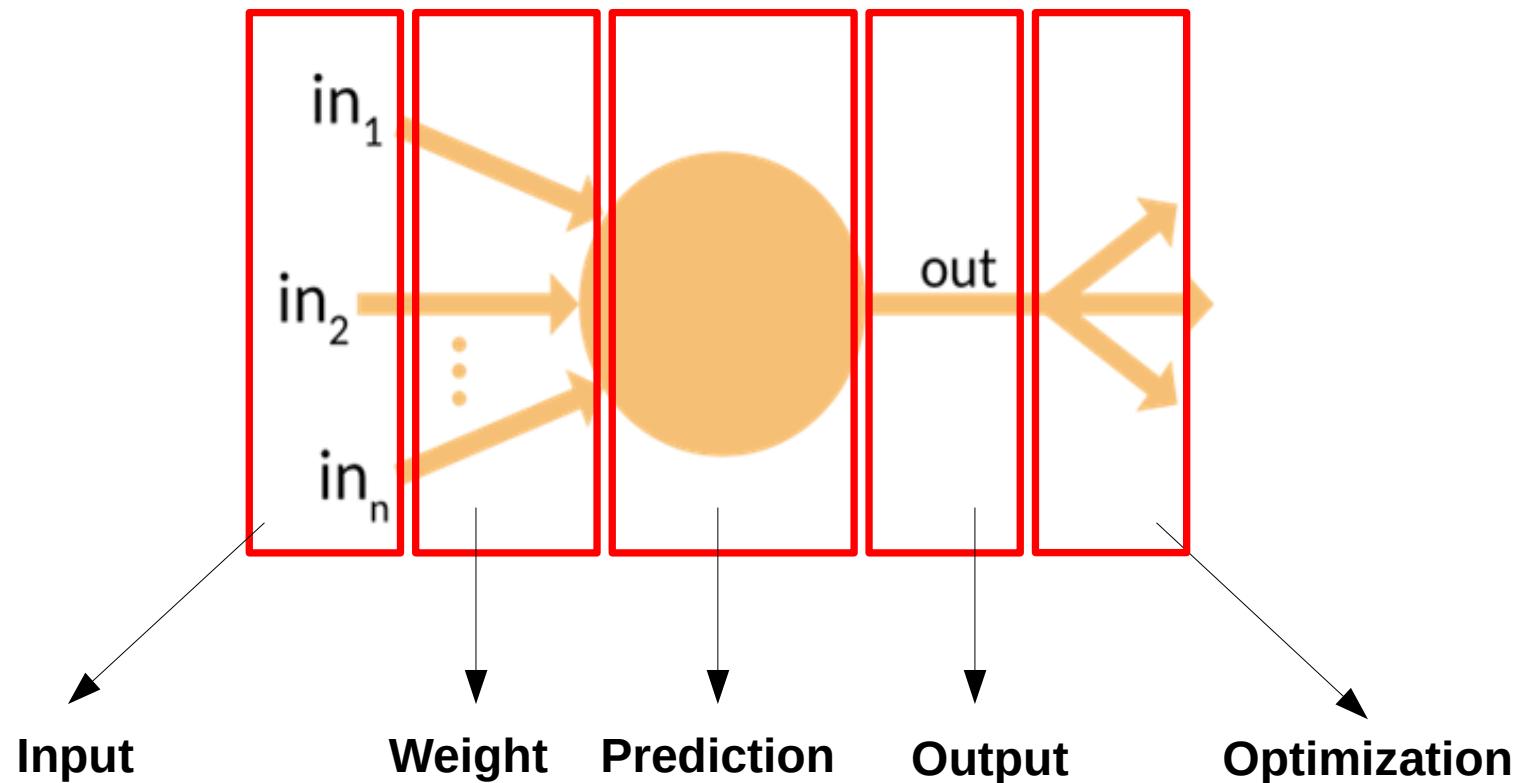
Multi Class Perceptron

Multi Layer Perceptron

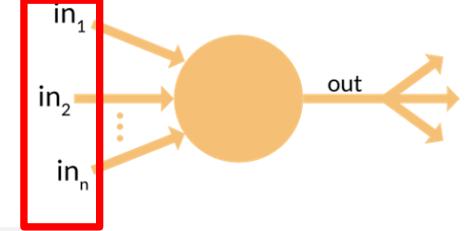
Summary

Perceptron

Components



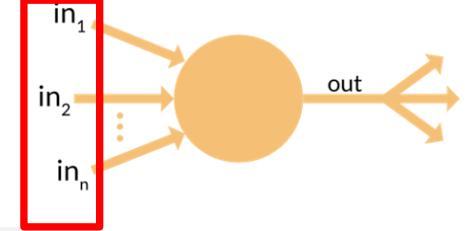
Perceptron



INPUT – General

- > Inputs must be numerical values
- > If data is not represented as numerical values they must be transformed into numbers
- > In order to train the Perceptron the class is needed for every data
- > Most of the time each entry is eventually represented as vector with different values

Perceptron



INPUT – Example

> Perceptron should learn if a patient has illness based on blood values, i.e. binary classification problem (sick or not sick)

> **Blood values of a patient:**

Input_1 = Amount of Iron

Input_2 = Amount of White Blood Cells

Input_3 = Amount of Sugar

Vector Components: { Input_1, Input_2, Input_3 }

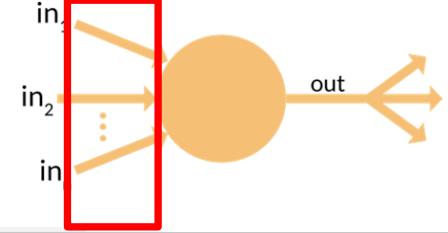
> **Train Data** could be as following:

Patient_1 = { 1, 0, 1 } => Class 1 (sick)

Patient_2 = { 1, 1, 1 } => Class 0 (not sick)

...

Perceptron

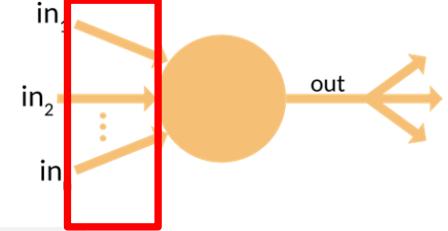


WEIGHTS – General

- > Each input type has its own weight
- > The weights are used to calculate the output class
- > They are also represented as vector
- > If weights lead to wrong class then they need to be adjusted
- > The adjustment will go on until the correct output class for a vector is calculated

- > But what are the intial values?
- > Most of the time semi-randomly generated

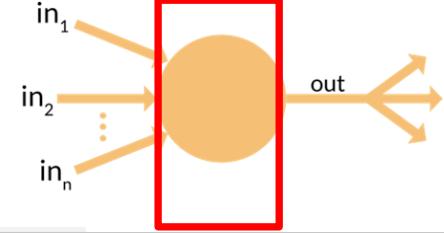
Perceptron



WEIGHTS – Example

- > Input types are blood values of patients (Input_1, Input_2, Input_3)
- > Each input type is connected to one weight
- > 3 input types => 3 different weights

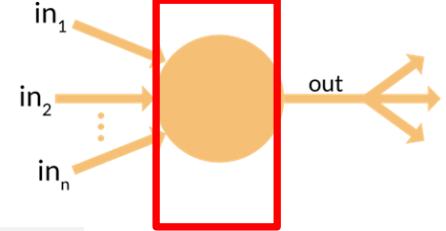
Perceptron



PREDICTION – General

- > The calculation is based on the dot product of the input vector and the weights vector
- > The result is something between 0 and 1

Perceptron



PREDICTION - Example

> Patient_1 has following blood values:

Input_1 = 2 (Amount of Iron)

Input_2 = 3 (Amount of White Blood Cells)

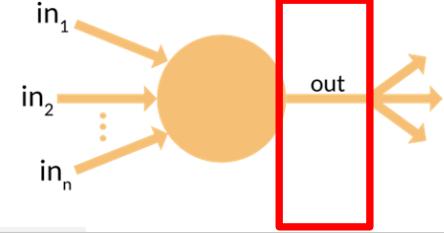
Input_3 = 2 (Amount of Sugar)

> **Vector** = { 2, 3, 2 }

> **Initial weights** = { 0.5, -0.2, 0.1 } (randomly generated)

> **Calculation** = $2 * 0.5 + 3 * (-0.2) + 2 * 0.1 = 0.6$

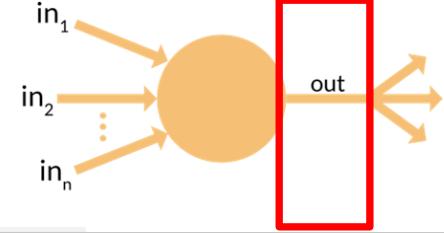
Perceptron



OUTPUT – General

- > The output is the result of the dot product and an activation function.
- > Determines the class
- > Activation functions can be super simple functions, e.g. thresholds or more complex ones, e.g. Sigmoid

Perceptron

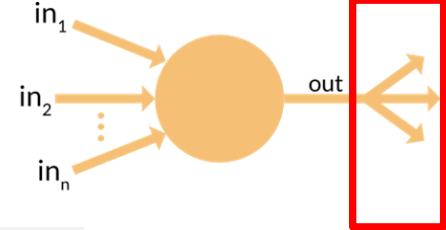


OUTPUT – Example

- > **Vector** = { 2, 3, 2 }
- > **Initial weights** = { 0.5, -0.2, 0.1 } (randomly generated)
- > **Calculation** = $2 * 0.5 + 3 * (-0.2) + 2 * 0.1 = 0.6$
- > **Threshold** = 0.5

- > **Class predicted is** = $0.6 > 0.5 \Rightarrow 1$ (patient is sick)

Perceptron



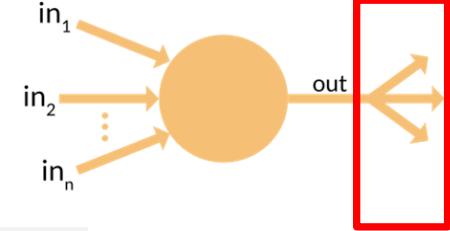
OPTIMIZATION - General

- > If calculation of prediction outputs a wrong class
- > Then the weights have to be adjusted in order to calculate the correct class

- > If the predicted class > actual class, then decrease weights
- > If the predicted class < actual class, then increase weights

- > Optimizer functions can also be simple, e.g. Hebbian Learn Rule or very complex

Perceptron



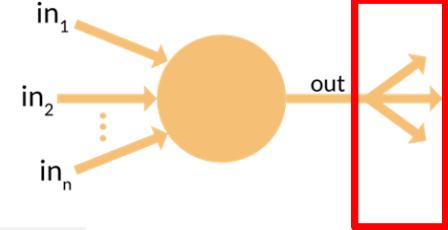
OPTIMIZATION - Training

> Hebbian Learn Rule:

*Weight Adjustment = Learning Rate * Input Value * Error*

- > Learning Rate is a Hyperparameter
- > The higher the rate the faster the perceptron adapts to new values but forgets faster already learned insights
- > Error is calculated as the following:
Error = Target Class – Predicted Class

Perceptron



OPTIMIZATION – Example

- > **Vector** = { 2, 3, 2 }
- > **Intial weights** = { 0.5, -0.2, 0.1 } (randomly generated)
- > **Calculation** = $2 * 0.5 + 3 * (-0.2) + 2 * 0.1 = 0.6$
- > **Threshold** = 0.5
- > **Class predicted is** = $0.6 > 0.5 \Rightarrow \text{,,1''}$ (patient is sick)
- > **Target class is** = „0“ (patient is not sick)
- > **Error** = 0 (Target) – 1 (Result) = -1
- > **Learning Rate** = 0.02

- > **Adjustment for Input_1 and Weight_1** = $0.02 * 2 * (-1) = -0.04$
- > **New Weight_1** = 0.5 (Old weight) – 0.04 (Adjustment) = 0.46

- > **Weights before:** { 0.5, -0.2, 0.1 }
- > **Weights after:** { 0.46, -0.26, 0.06 }

Introduction

Perceptron

Multi Class Perceptron

Multi Layer Perceptron

Summary



Multi Class Perceptron

Multi Class Problem

- > Previously only two classes: „0“ or „1“ (Sick – Not Sick, Spam – No Spam, etc.)

Example for Multi Class Problem:

- > Image Recognition
- > Each picture (dog, cat, mouse, etc.) equals to one class

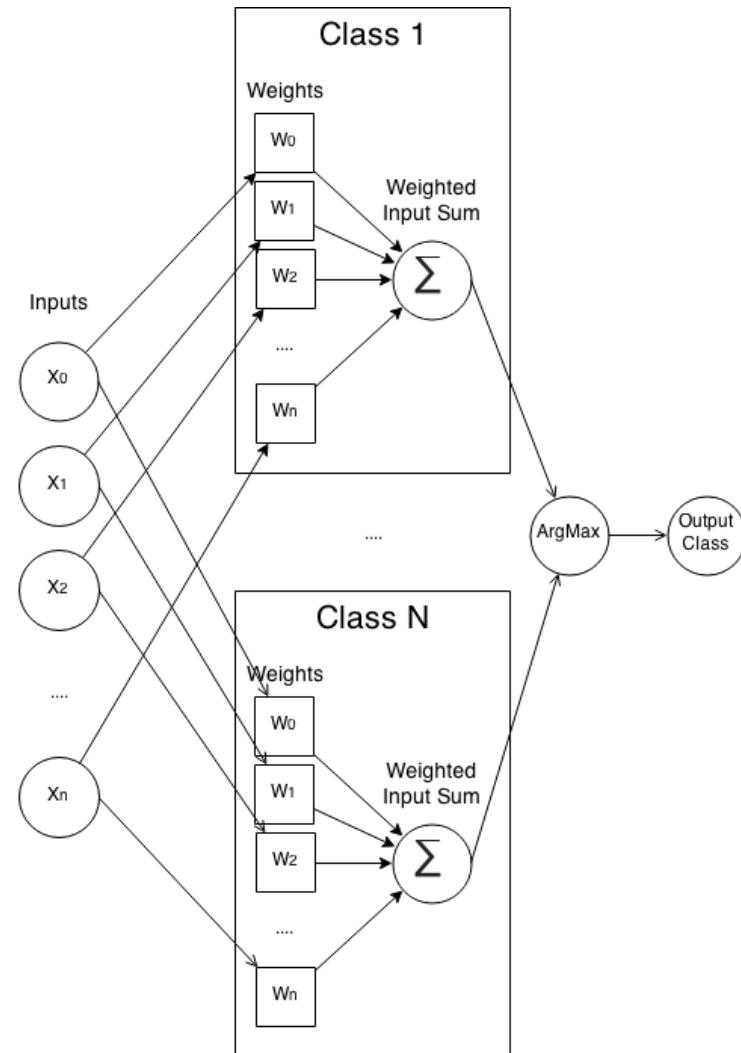
Multi Class Perceptron

From Binary to Multi?

- > Is it possible to use perceptrons for multiple classes?
- > If not, what could we change to make it work?

Multi Class Perceptron

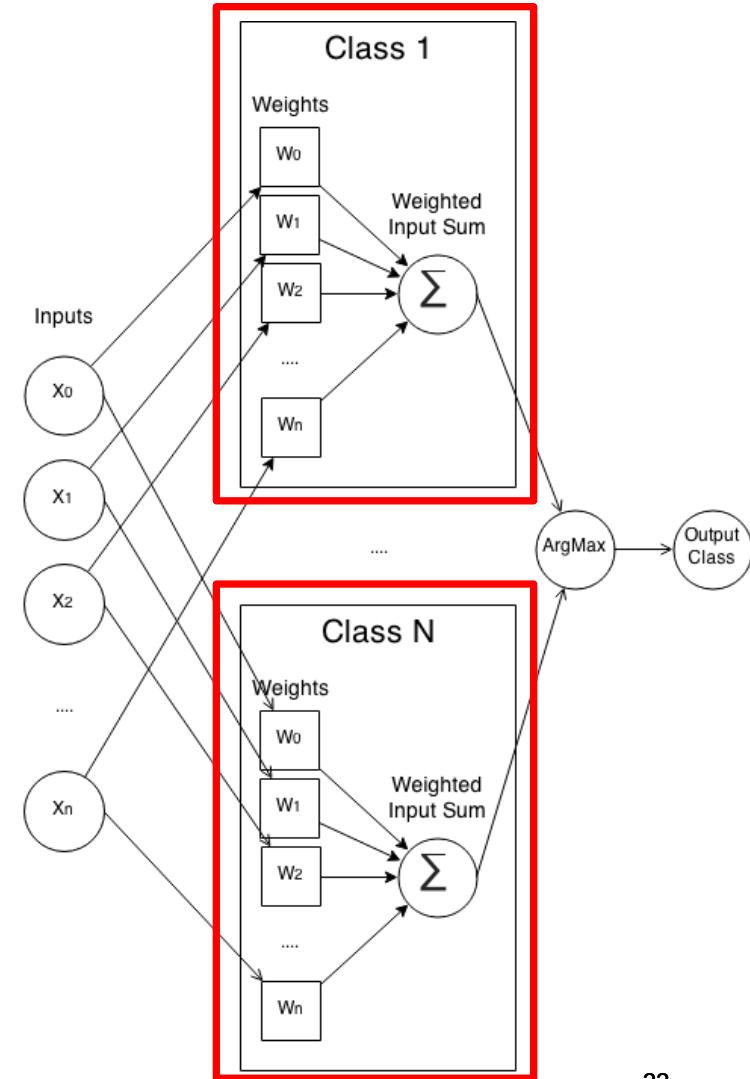
General



Multi Class Perceptron

CALCULATION

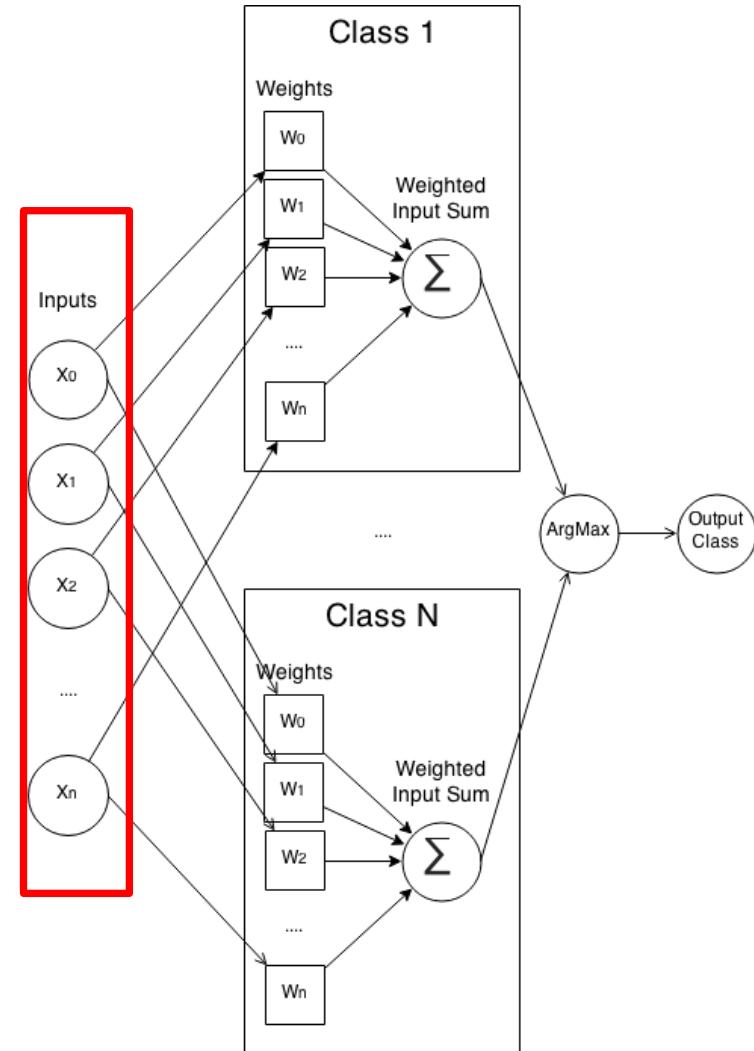
- > Each class is one perceptron and will be trained separately
- > That is, each class/perceptron has its own set of weights to train
- > Calculation remains the same



Multi Class Perceptron

INPUT

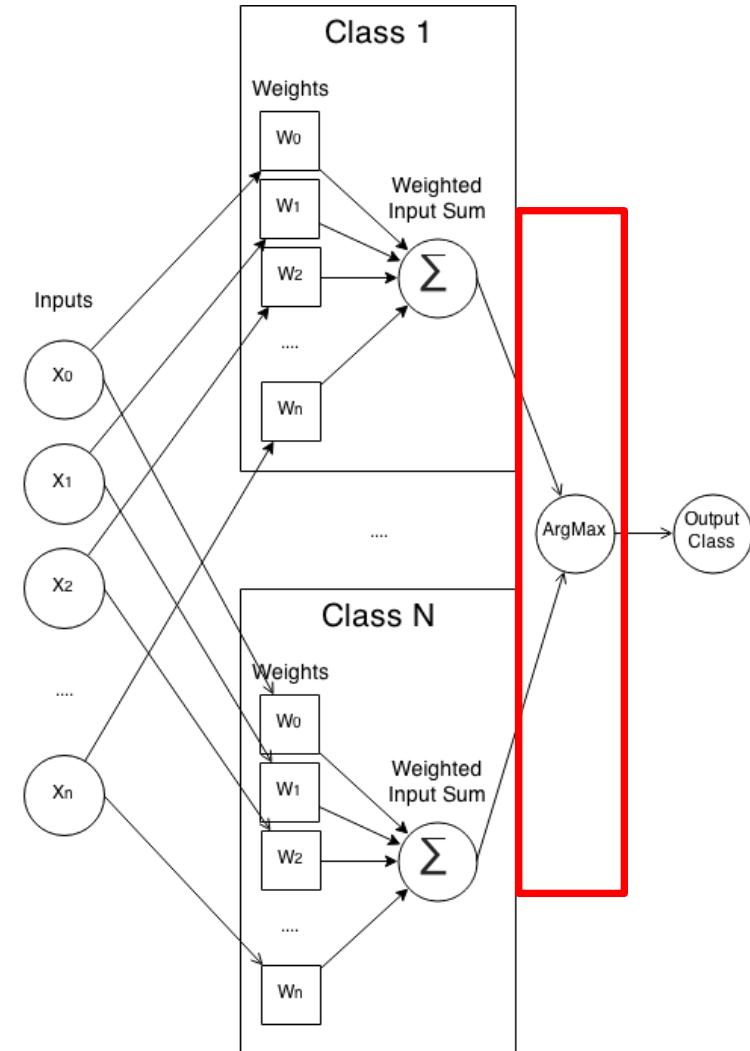
- > The whole input will be trained with each class weight set



Multi Class Perceptron

PREDICTION

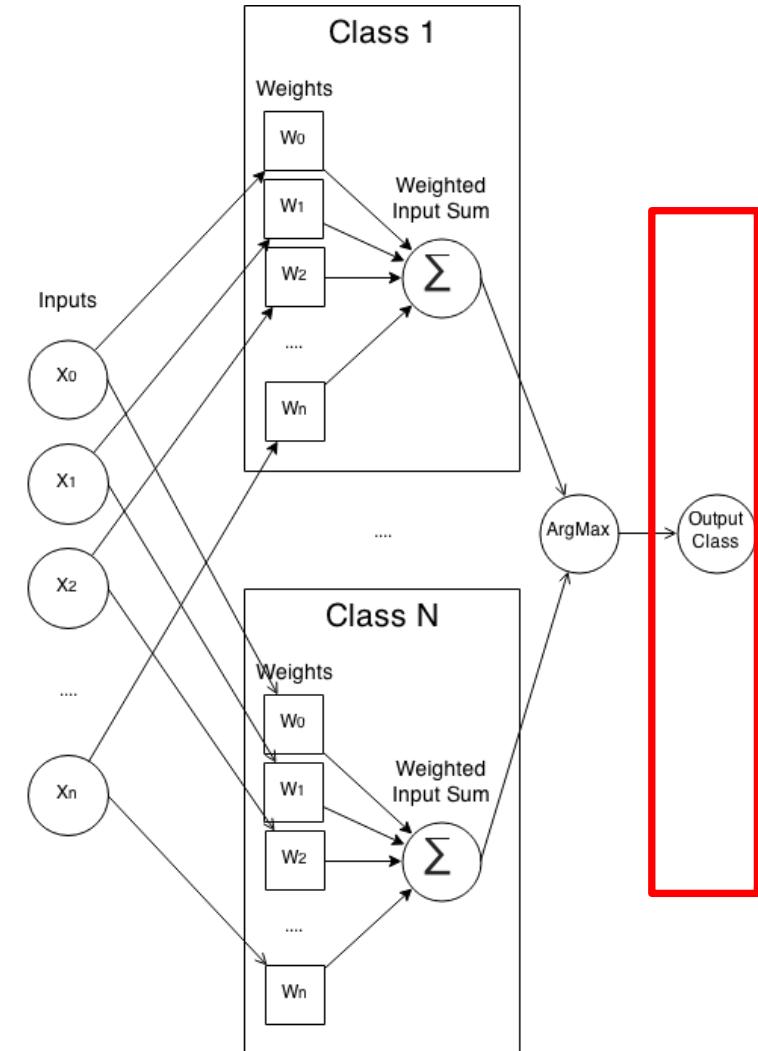
- > Results of each class weight calculation will be compared
- > The class with the highest calculation is the class of prediction (ArgMax Function)



Multi Class Perceptron

TRAINING

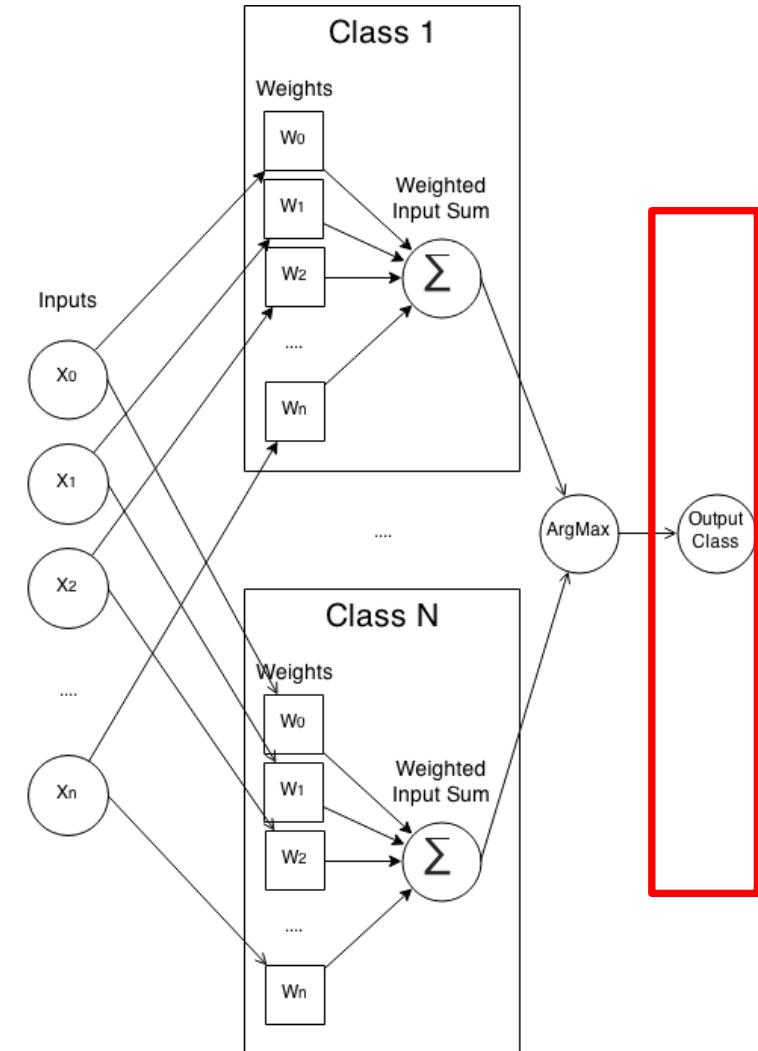
- But how do we train now?



Multi Class Perceptron

TRAINING

- > If the prediction is wrong, the weights of the predicted class have to be decreased for the particular input
- > Weights of correct (but not predicted) class have to be adjusted upwards



Introduction

Perceptron

Multi Class Perceptron

Multi Layer Perceptron

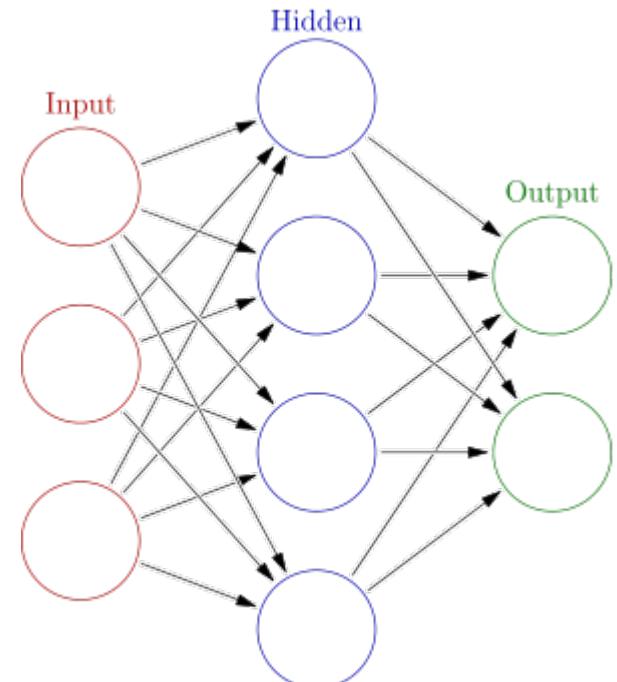
Summary

Multi Layer Perceptrons

Structure of Multi Layer Perceptrons (aka Neural Networks)

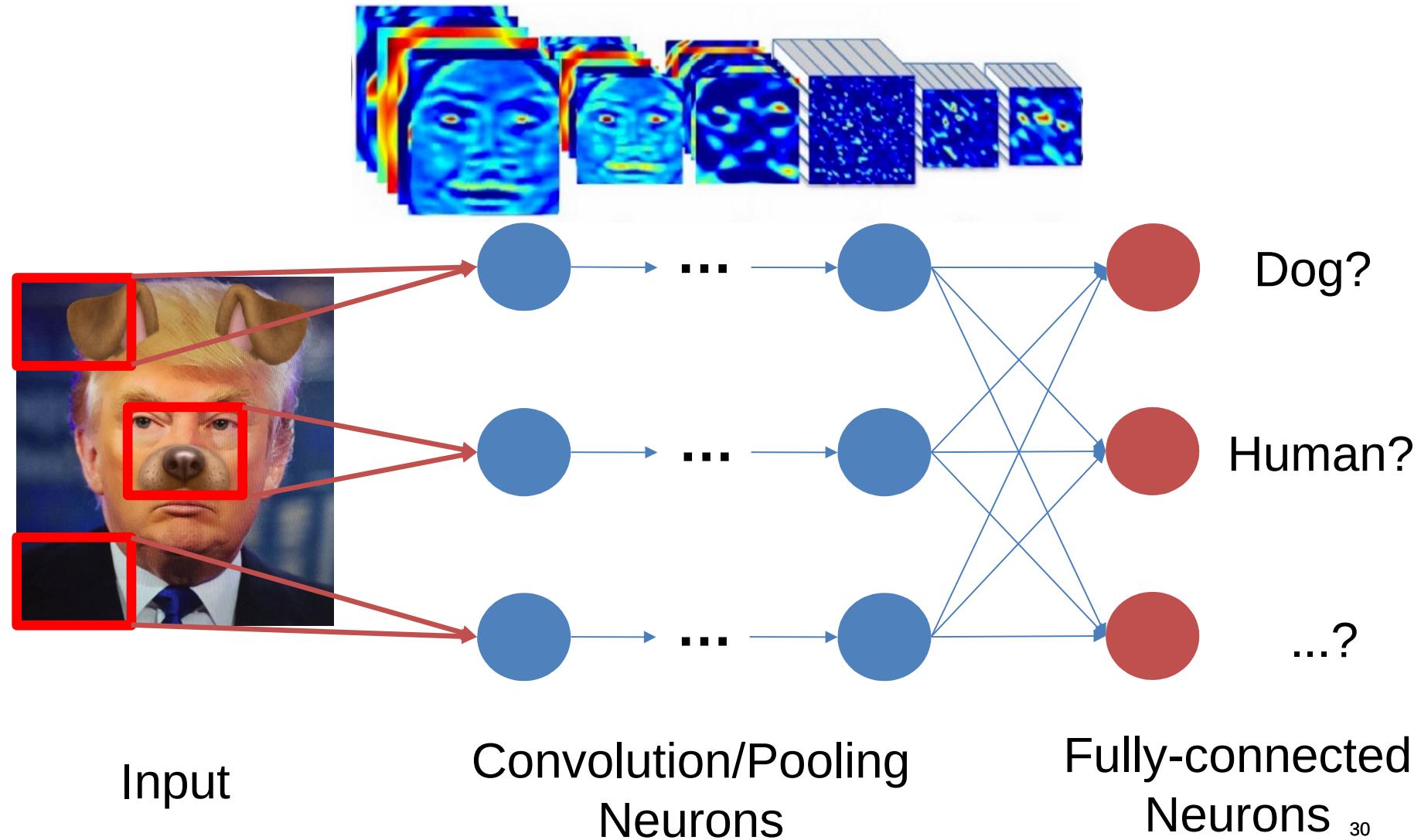
- > Multi Class Perceptrons have one hidden layer where all the calculations is done

- > For more complex problems neural networks with several layers have been developed (and nowadays called „deep“)



Multi Layer Perceptrons

Example for Deep Learning – Convolutional Neural Network



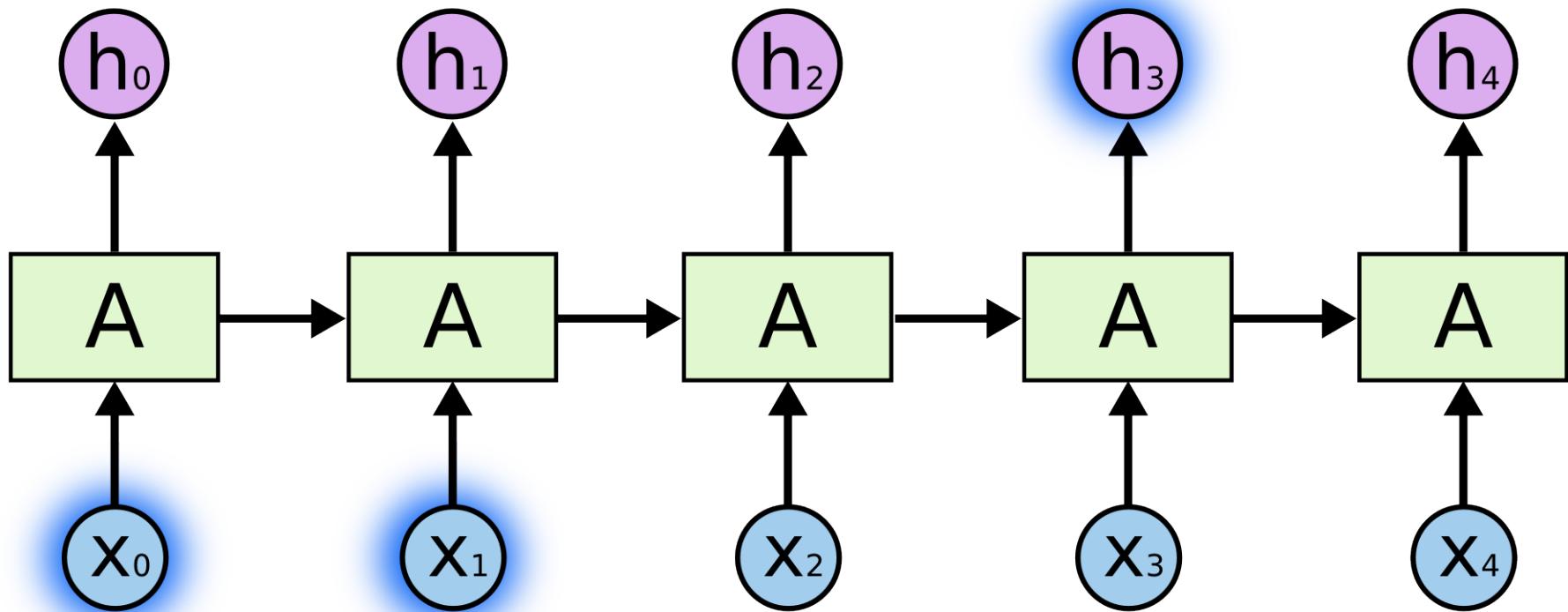
Multi Layer Perceptrons

Recurrent Neural Network

- > Recurrent Neural Network (RNN) as model
- > RNNs have a „memory“
- > Takes not only input into calculation for each neuron but also the result of previous neurons
- > Much more like the human brain (reading, speaking, etc.)

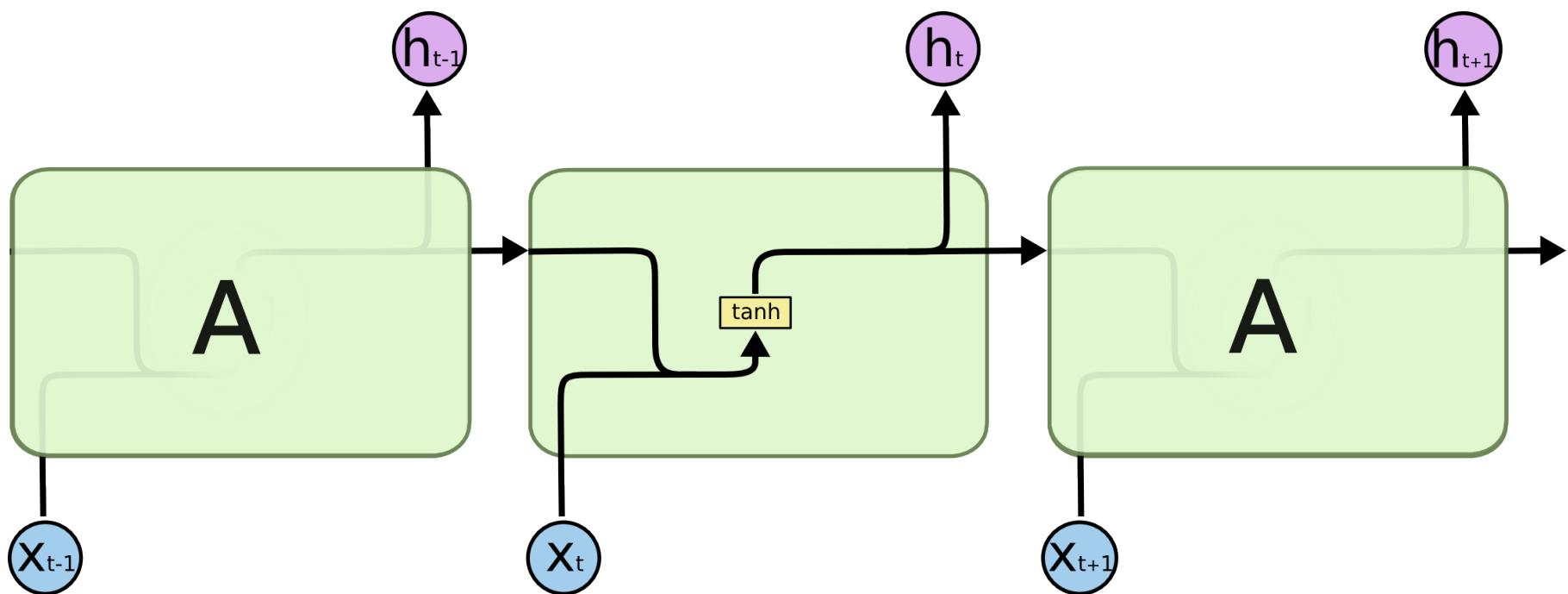
Multi Layer Perceptrons

Recurrent Neural Network – Remember previous



Multi Layer Perceptrons

Recurrent Neural Network – Internal Calculation



Introduction

Perceptron

Multi Class Perceptron

Multi Layer Perceptron

Summary

Neural Network Basics

Summary and terms

- > **Input:** Numerical vectors or matrices
- > **Activation:** Can be a threshold or other functions to determine if Neuron fires or not („0“ or „1“)
- > **Learning rate:** Determines how fast an algorithm should adjust to new data. A high learning rate adjusts better and faster to new data but also discards learned insights
- > **Optimizer:** Function to optimize model (i.e. weights)
- > **Epochs:** An epoch is a whole iteration of than one iteration of all data to have an optimal model for the given problemall input data. Algorithms need more
- > **Batches:** Sometimes data is too big to put fully into RAM. Then it needs to be processed into data slices (aka batches).
- > **Overfitting:** If model adapts perfectly on training data but not on new unseen data, then we talk of overfitting. It is like remebering by heart
- > **Regularization:** Goal is to use as less variables for a model as possible to prevent overfitting, e.g. by dropping out a fixed number of neurons