INTRODUCTION TO DEEP LEARNING

PART I – INTRODUCTION AND BASICS

Auliya Fitri, Sai Vemuri, Sreerag Naveenachandran

Machine Learning Group
Institute of Data Science

DLR

Auliya Fitri

German Aerospace Center – Institute of Data Science

Data Analysis and Intelligence

Machine Learning Group

Research Interests:

- Machine Learning
- Explainable Artificial Intelligence
- Uncertainties in Neural Networks

Contact:

Phone: +49 3641 30960 165

Mail: auliya.fitri@dlr.de

Sai Karthikeya Vemuri

German Aerospace Center – Institute of Data Science

Data Analysis and Intelligence

Machine Learning Group

PhD Student: Computer Vision (FSU Jena)

Research Interests:

- Physics in Machine Learning
- Knowledge Integration

Contact:

Mail: sai.karthikeya.vemuri@uni-jena.de

Sreerag Vadakkemeppully Naveenachandran

German Aerospace Center – Institute of Data Science

Data Analysis and Intelligence

Machine Learning Group

Research Interests:

- Machine learning for engineering systems
- Anomaly detection in time series
- Diffusion models

Contact:

Phone: +49 3641 30960 206

Mail: Sreerag.Naveenachandran@dlr.de

Schedule

Date	Time	Activity
13.11.2025 Day 1	09:00 - 10:00	Introduction and basics
	10:00 - 10:30	Hands-on I
	10:30 - 10:45	Coffee break
	10:45 - 11:45	Advanced concept and Convolutional Neural Network
	11:45 - 12:15	Hands-on II
	12:15 - 12:30	Recap Day 1
14.11.2025 Day 2	09:00 - 10:00	Deep Generative Models
	10:00 - 10:30	Hands-on III
	10:30 - 10:45	Coffee break
	10:45 - 11:45	Transformers, LLM, and other interesting architectures
	11:45 - 12:15	Hands-on IV
	12:15 - 12:30	Code and knowledge sources + closing

Curriculum

I. Introduction and basics

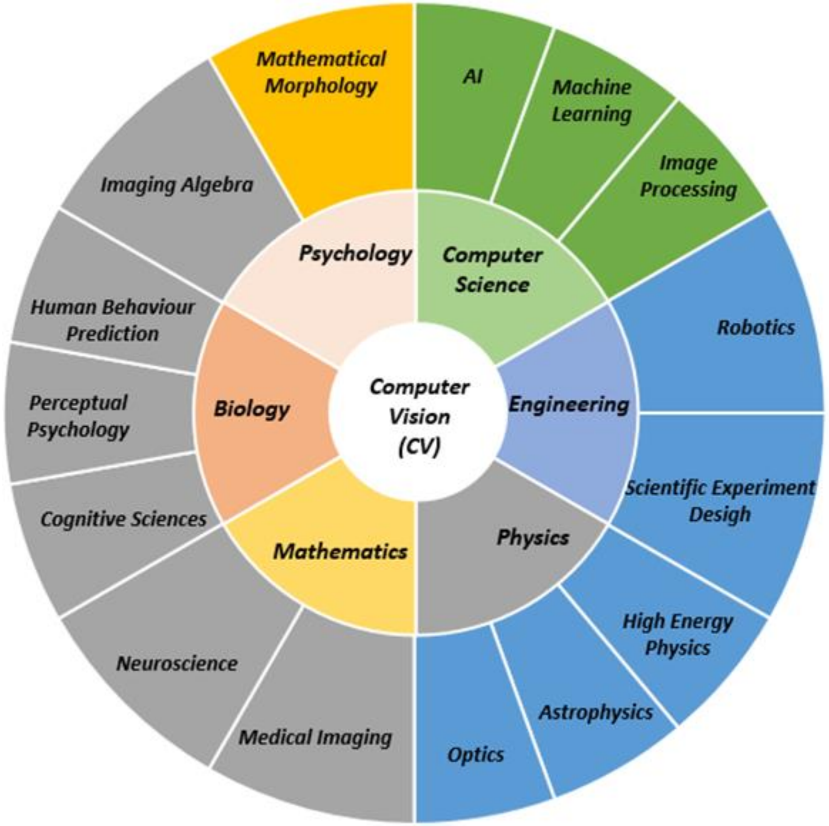
- Application examples
- Machine learning background
- Neural network concepts
- Training procedure

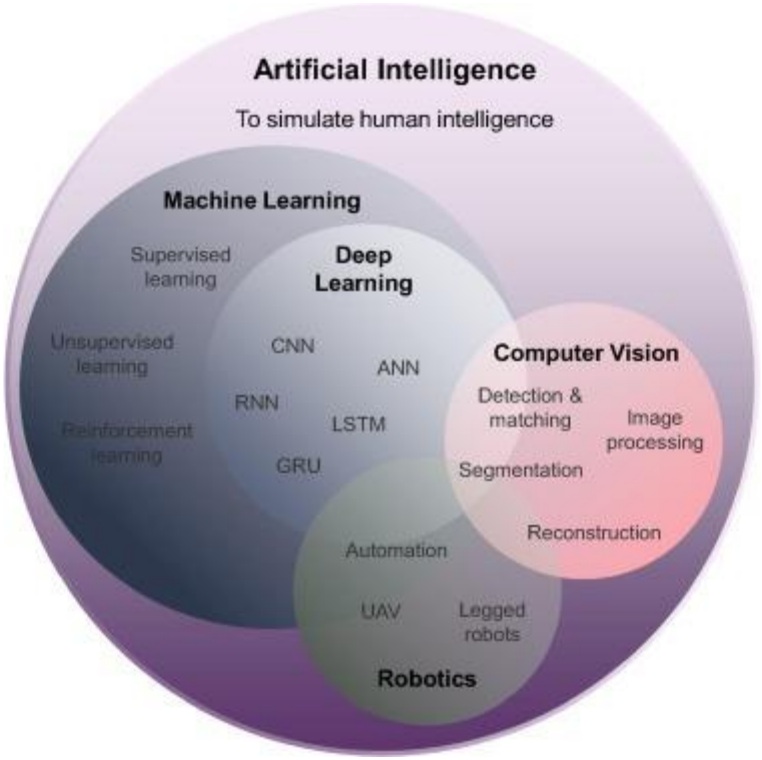
Inspired by lectures from Paris Saclay and MIT; images taken from these, if not noted otherwise

APPLICATION EXAMPLES

Application Examples
Computer Vision

Computer Vision is a interdisciplinary field with strong relations to AI and DL



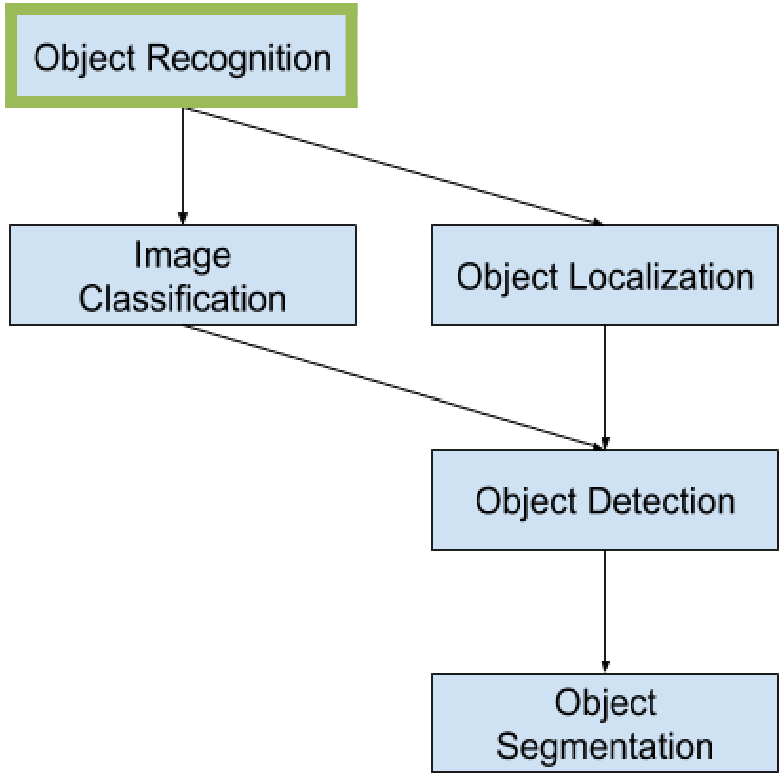


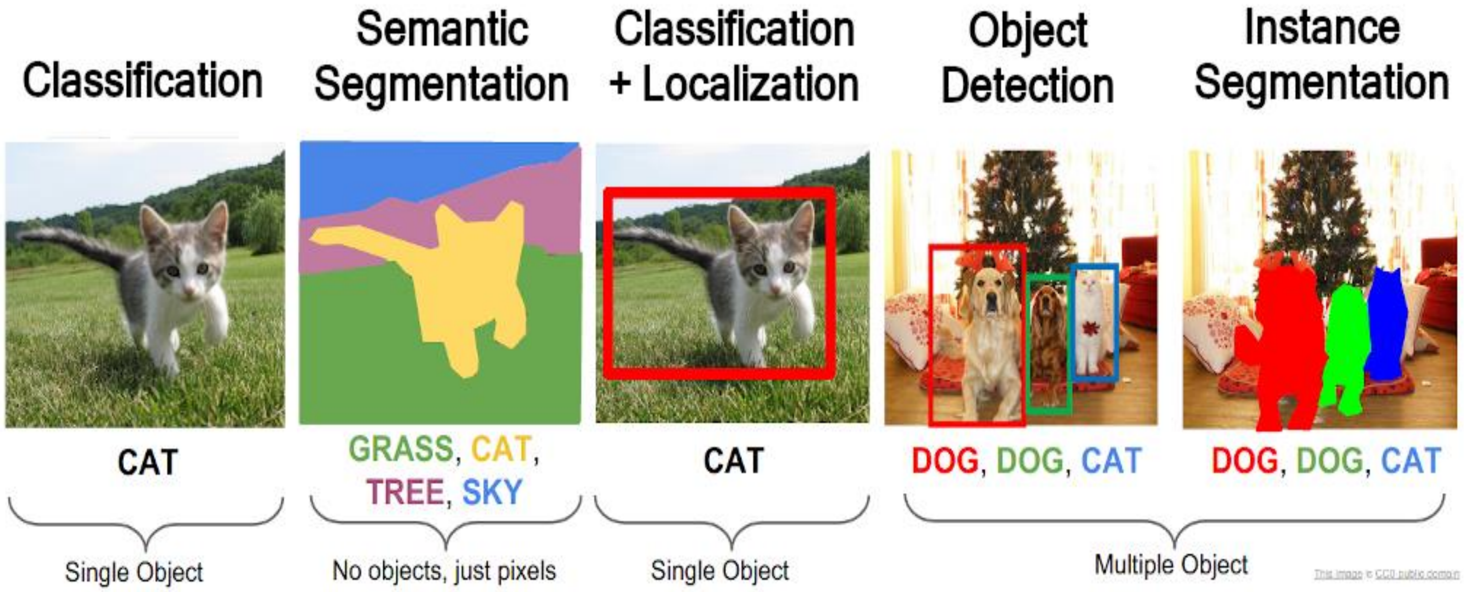
[Computer Vision and Deep Learning – SevenShineStudios (wordpress.com)]

[BIM, machine learning and computer vision techniques in underground construction: Current status and future perspectives - ScienceDirect]

Application Examples
Computer Vision - Object Recognition / Image Processing

"Object Recognition refers to a collection of related tasks for identifying objects in digital photographs."



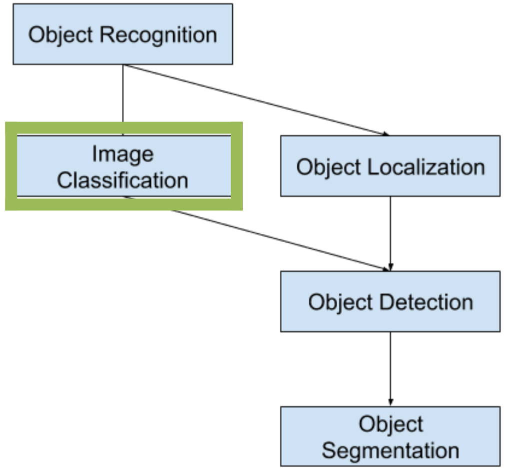


http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

A Gentle Introduction to Object Recognition With Deep Learning (machinelearningmastery.com) (2021)

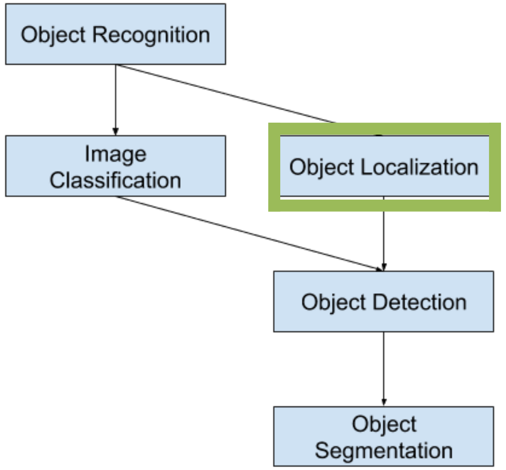
Application Examples
Computer Vision – Image Classification

"Image Classification predicts the type or class of an object in an image."



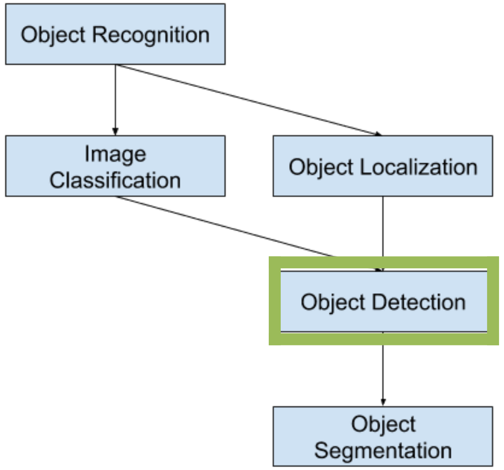
Application Examples
Computer Vision – Object Localization

"Object Localization locates the presence of objects in an image and indicate their location with a bounding box."



Application Examples
Computer Vision – Object Detection

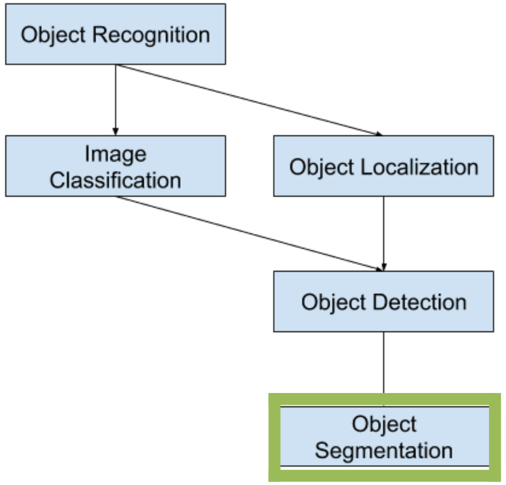
"Object Detection locates the presence of objects with a bounding box and types or classes of the located objects in an image."



Application Examples
Computer Vision – Semantic and Object Segmentation

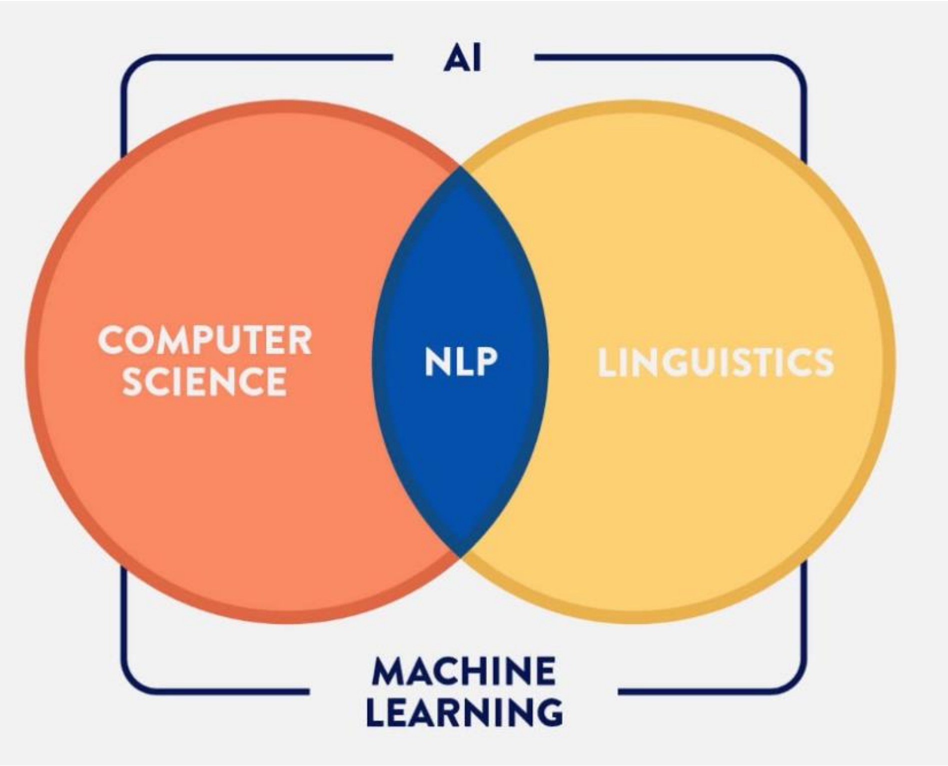
Semantic Segmentation highlights pixels but not objects

“Object Segmentation highlights the specific pixels of the object instead of a coarse bounding box.”



Application Examples
Natural Language Processing (NLP)

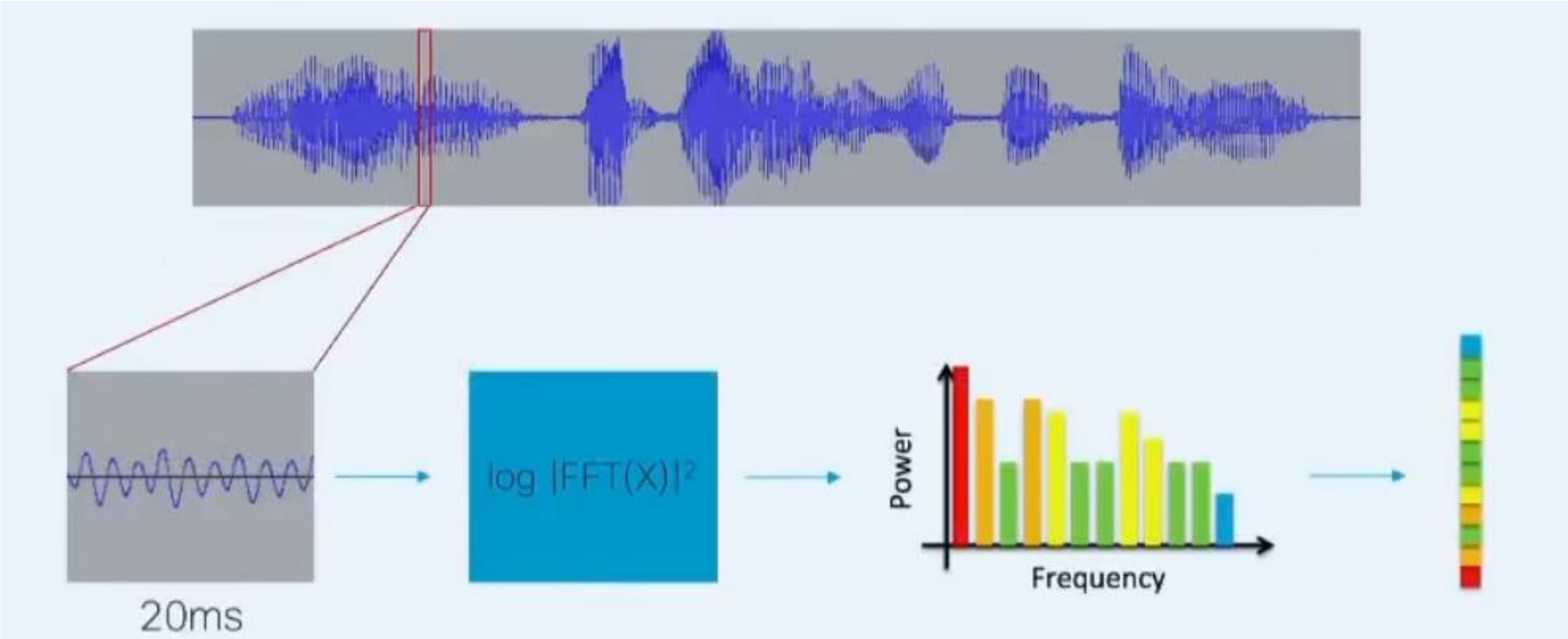
NLP is a interdisciplinary field and gives computers the ability to understand human language.



[Introduction to Natural Language Processing (NLP) | by Ibrahim Sharaf ElDen | Towards Data Science]

Application Examples
Speech Recognition

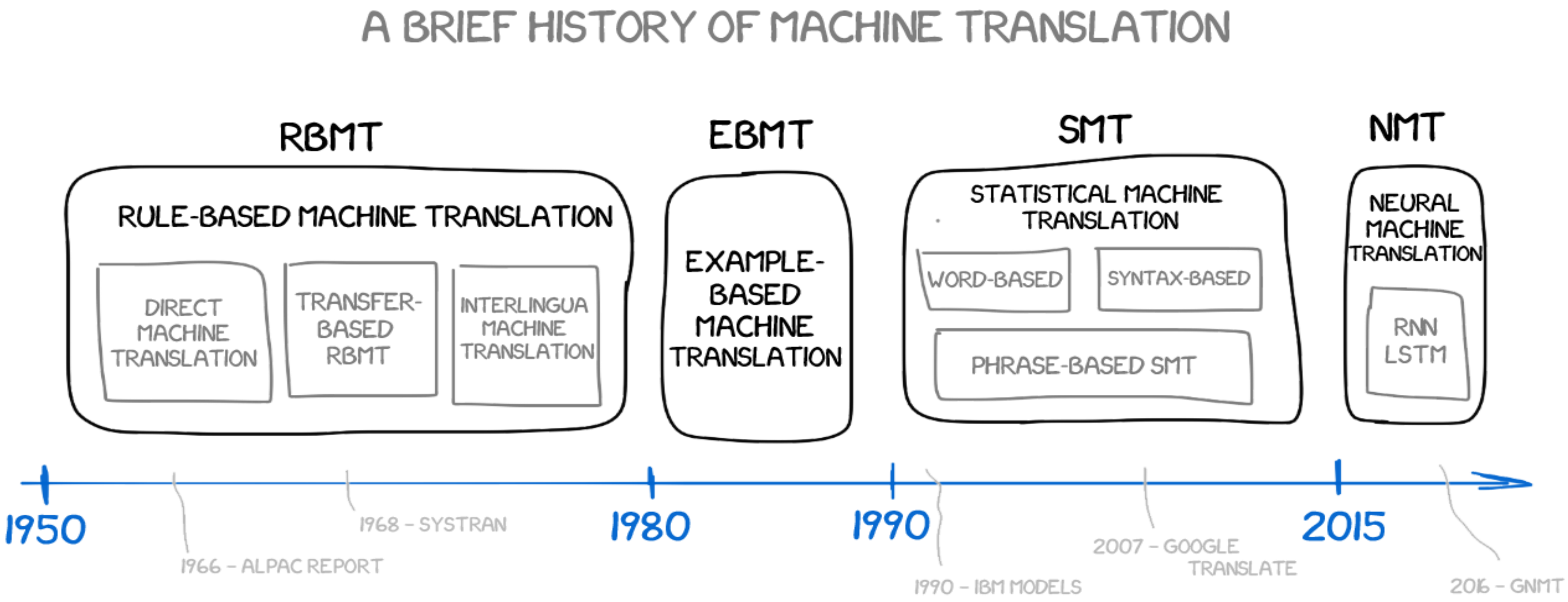
"The process of enabling a computer to identify and respond to the sounds produced in human speech."



Deep Learning for Speech Recognition | by ODSC - Open Data Science | Medium

Application Examples
NLP – Language Translation / Machine Translation

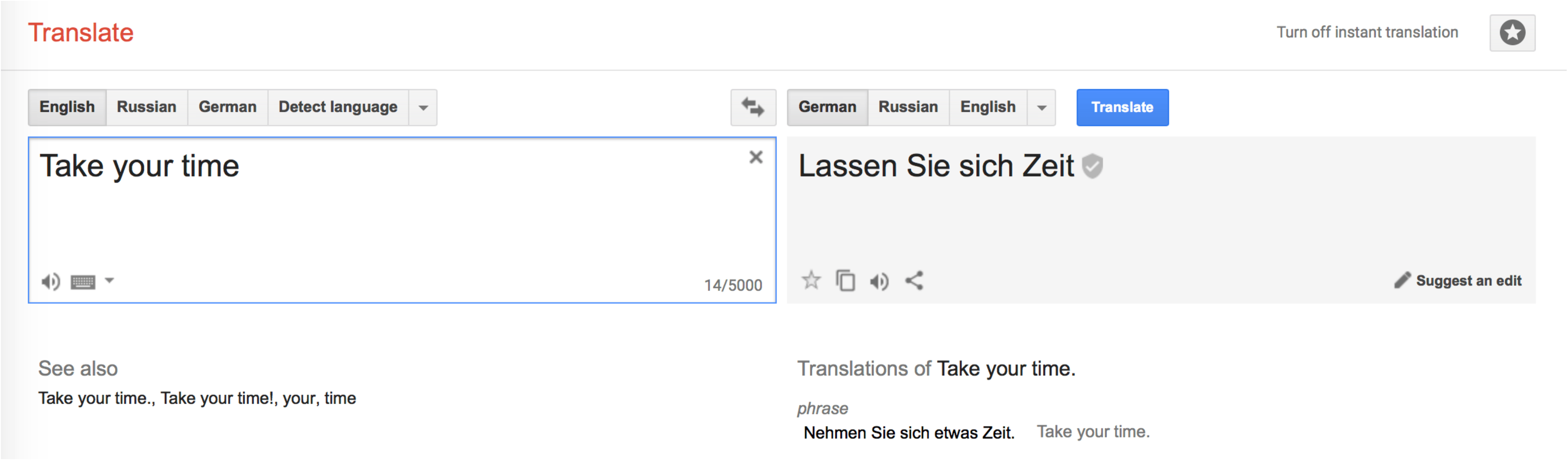
"Machine translation is the process of using artificial intelligence (AI) to automatically translate content from one language (the source) to another (the target) without any human input."



[Machine Translation :: From the Cold War to Deep Learning :: vas3k.com]

Application Examples
NLP – Language Translation / Machine Translation

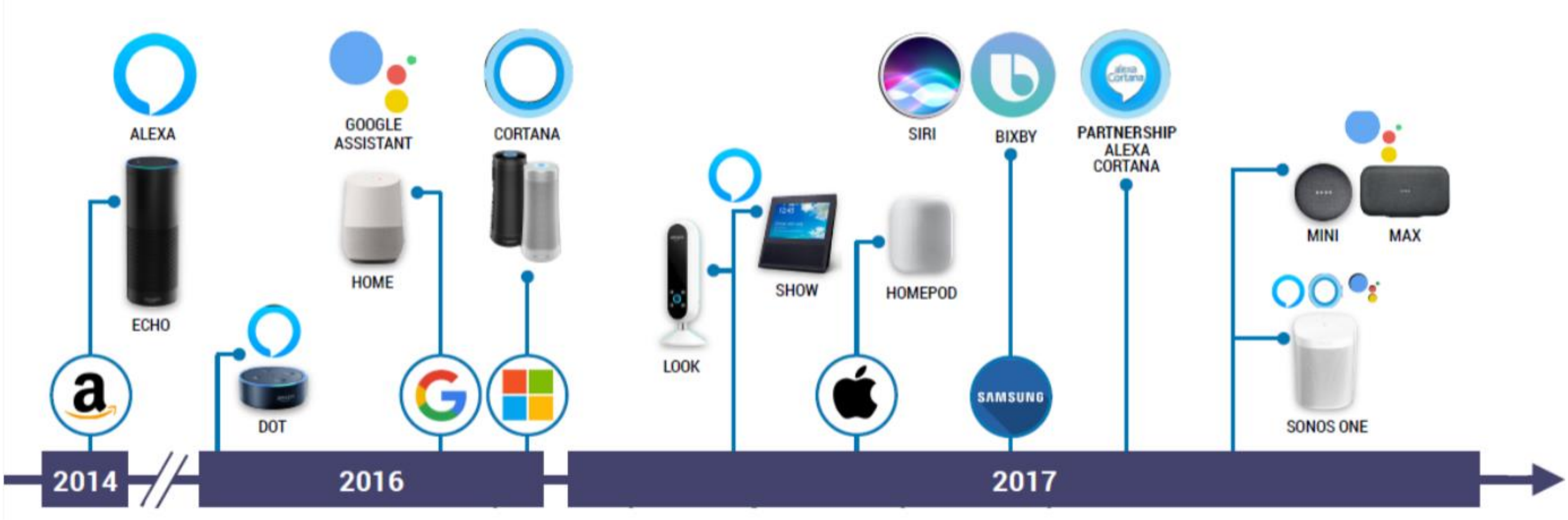
Google Translate



Machine Translation :: From the Cold War to Deep Learning :: vas3k.com

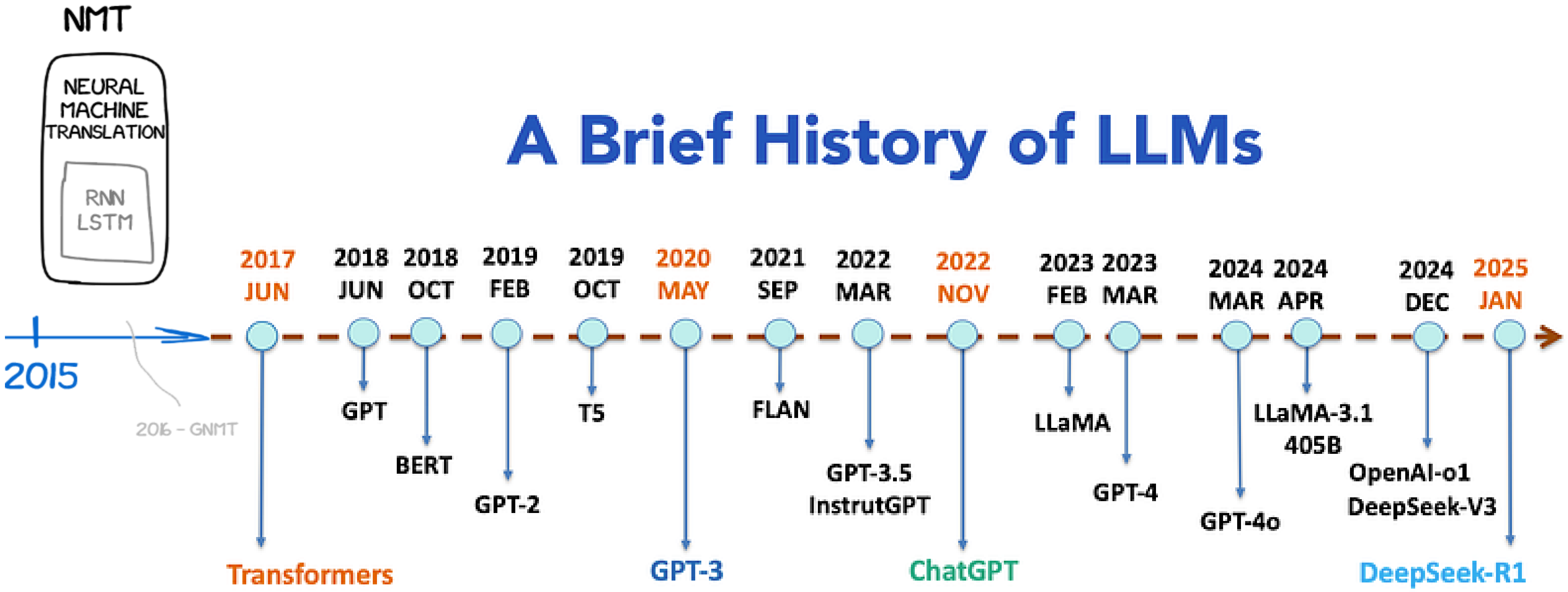
Application Examples
NLP – Virtual & Voice Assistants

“A virtual assistant is an application that understands voice commands and completes tasks for a user”



[Haben Sie schon Ihren persönlichen Voice Assistant? (pidas.com)](pidas.com)

Application Examples
NLP – Large Language Models (LLMs)



[A brief history of LLMs (medium.com)]

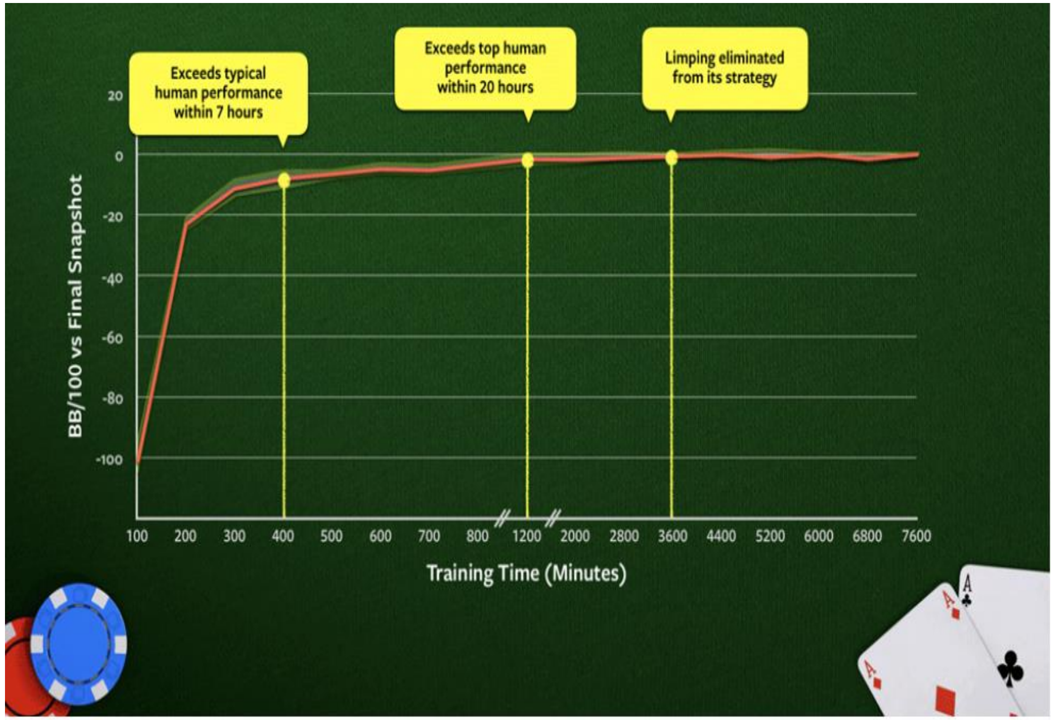
Application Examples
Entertainment – games & bots

AlphaZero (deepmind) is a self-taught computer program for very high complexities. It learns with a complete game information approach solely based on game rules

Go (game) - Wikipedia

Application Examples
Entertainment – games & bots

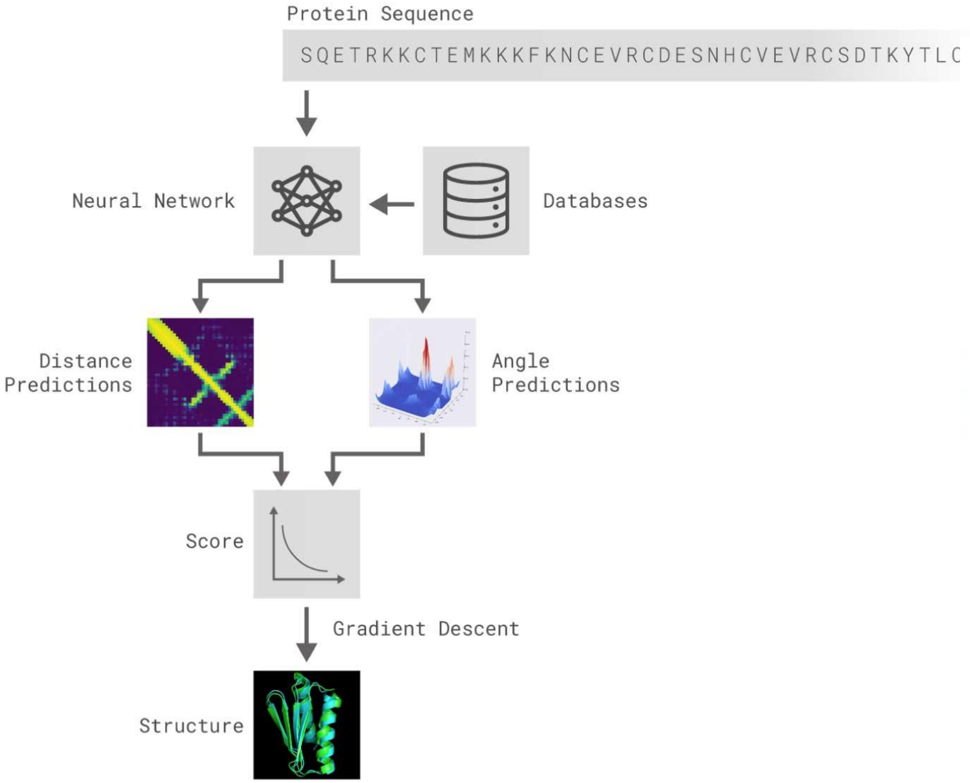
Pluribus (facebook) is an AI for multiplayer poker that beats 6 pros at once.
It reaches top performances after 20h training and learns with an incomplete game information approach.

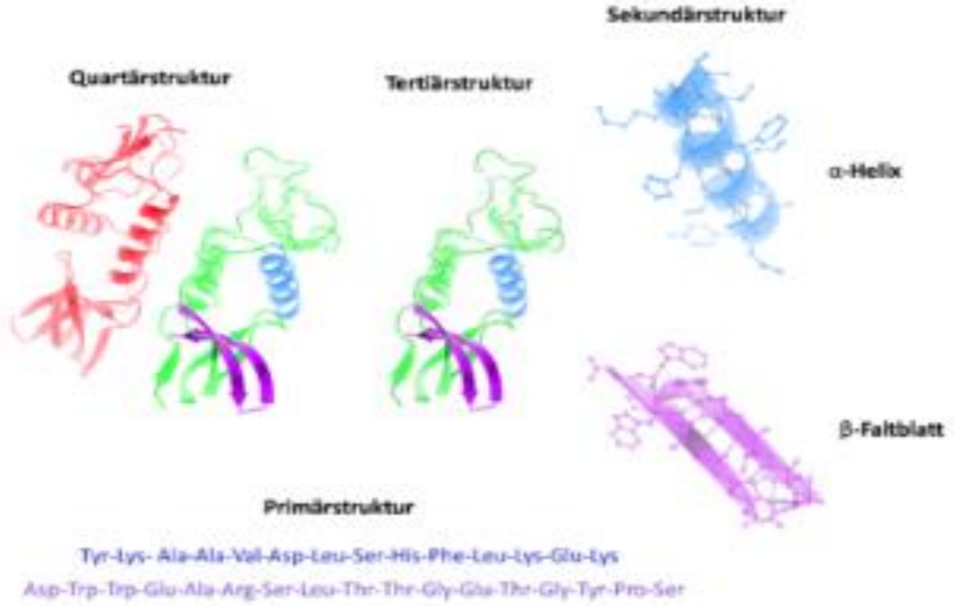


Superhuman AI for multiplayer poker (science.org)
Facebook, Carnegie Mellon build first AI that beats pros in 6-player poker

Application Examples
AlphaFold by DeepMind (Winner of chemistry Nobel Prize 2024)

AlphaFold predicts protein structure based on the amino acid sequence of the protein.





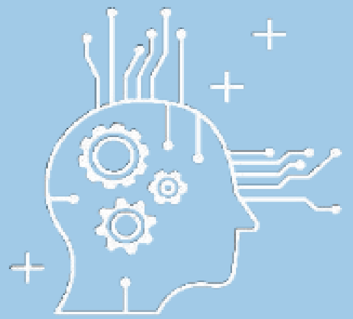
AlphaFold – Wikipedia

MACHINE LEARNING BACKGROUND

Machine learning background

Artificial Intelligence

Any technique that enables computers to mimic human behavior

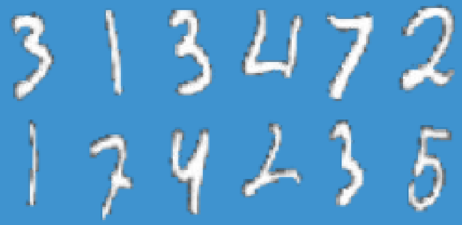


Machine Learning

Ability to learn without explicitly being programmed

Deep Learning

Extract patterns from data using neural networks

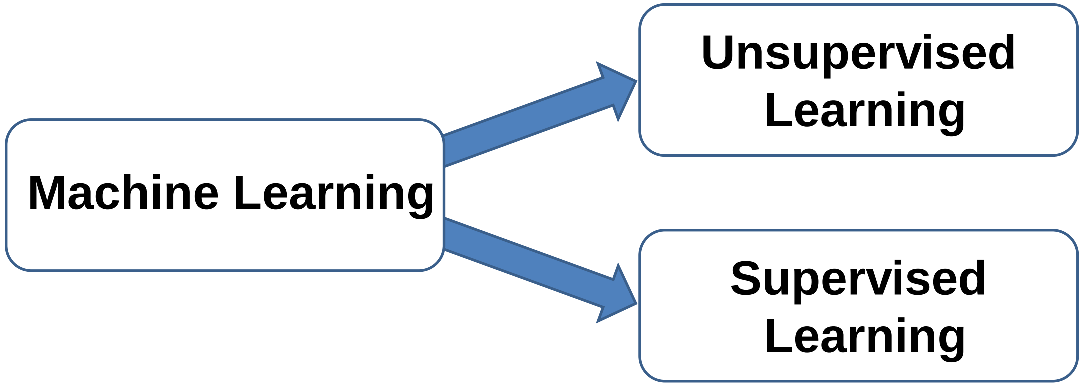


Machine learning background

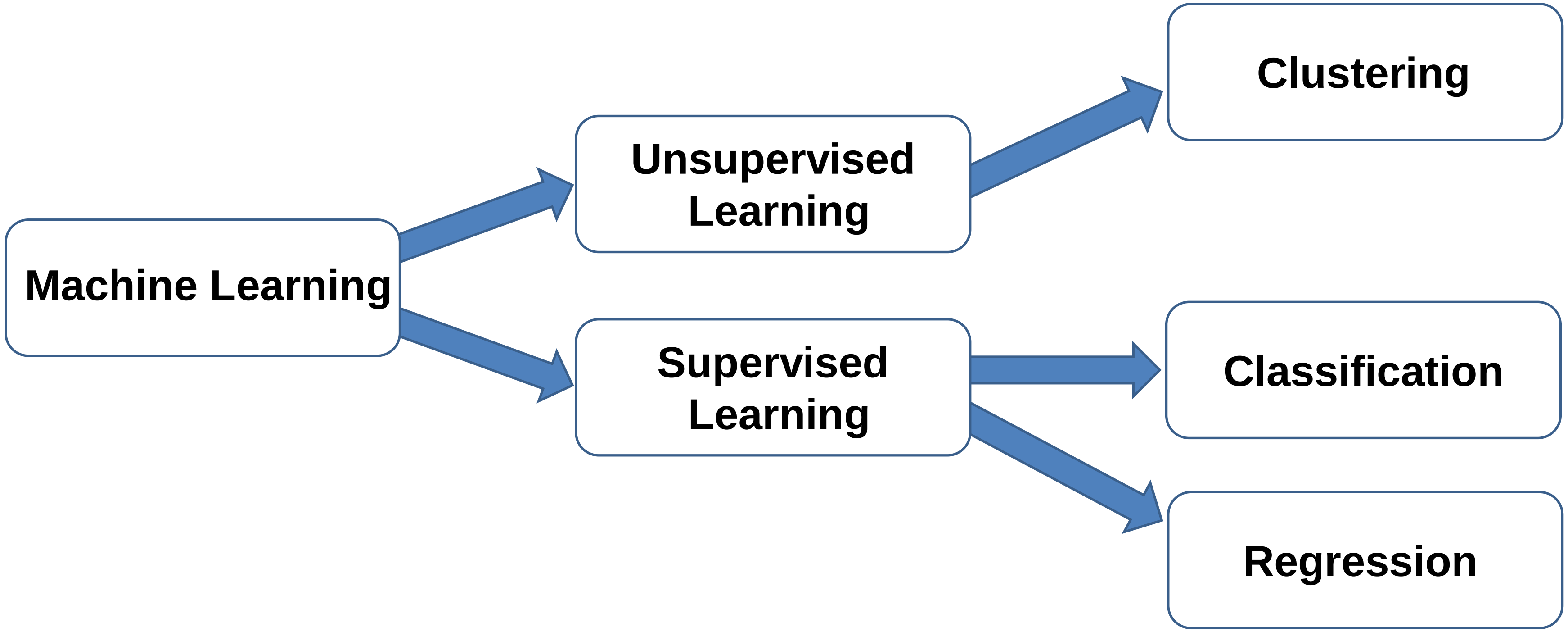
Machine Learning Technics

Machine Learning

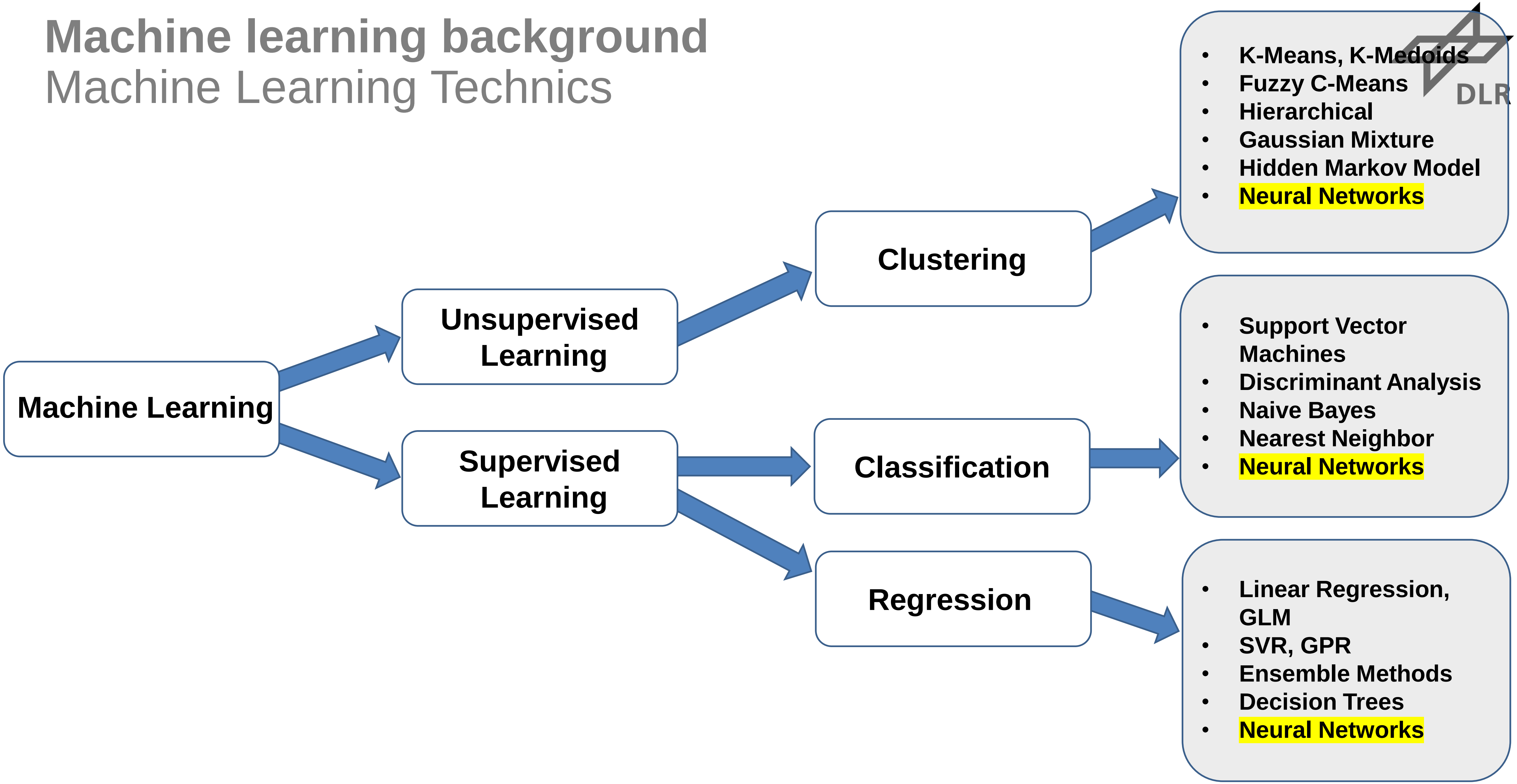
Machine learning background
Machine Learning Technics



Machine learning background
Machine Learning Technics

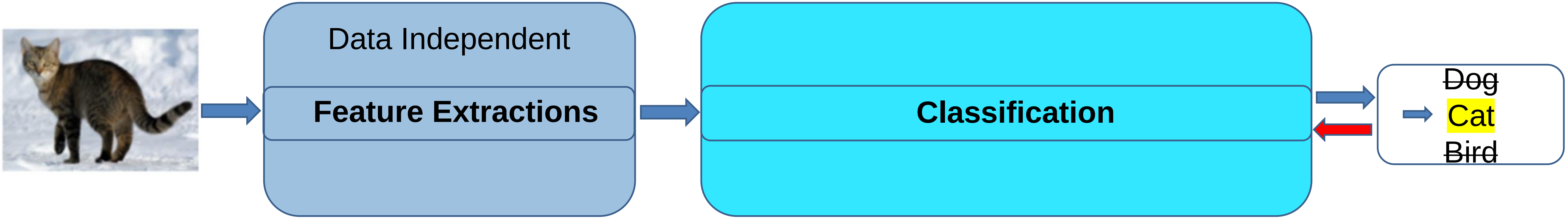


Machine learning background
Machine Learning Technics

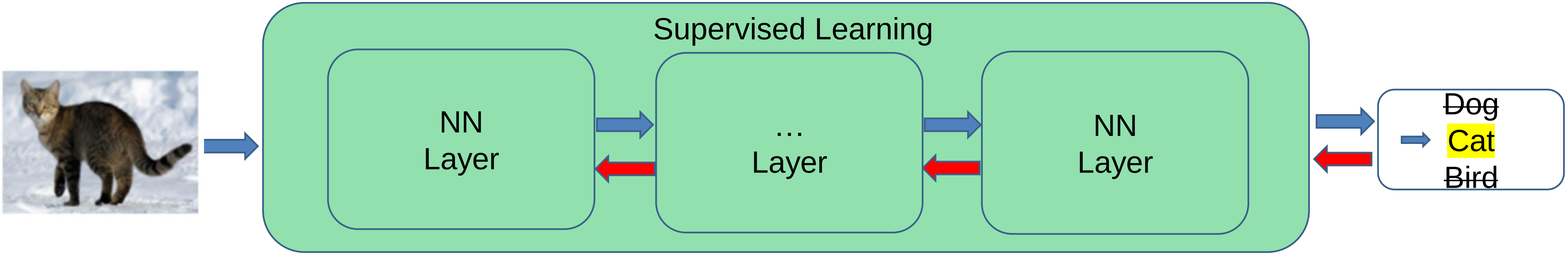


Machine learning background

“Traditional“ ML System



Deep Learning System



Machine learning background
Deep learning system - Why deep learning now?

1. Big Data

+DL superiority with increasing big data sets
+Collection & storage
+relevant for multiple disciplines

+decreasing error rate

2. Hardware

+Parallelization (GPU/TPU)

+Cloud-based DL systems (e.g. AWS, MS-Azure)

+reduce computing Power and memory use

3. Software

+improved techniques from growing research
+optimization SW&HW
+new models and toolboxes

PyTorch TensorFlow

- 1952 – Stochastic Gradient Descent
- 1958 – Perceptron -learnable weights
- …
- 1986 – Back-propagation Multi-Layer Perceptron
- 1995 – Deep Con-volutional NN Digit Recogn.
- 1997 – “Long Short-Term memory” (LSTM)- paper
- 2006 – Deep Belief Network- RBM-Layers
- 2008 – GPU Revolution in DL begins
- 2011 – ReLU activation function
- 2014 – Birth of GAN
- 2016 – TPU (Tensor Processing Unit) by Google
- 2018 – Turing Award in Deep learning
- 2020 – DeepSpeed (MS)

NEURAL NETWORK CONCEPTS

Neural network concepts

Building the network
- Input layer
- Hidden layers
- Output layer

Forward propagate input
- Propagate input through neurons
- Apply non-linearities

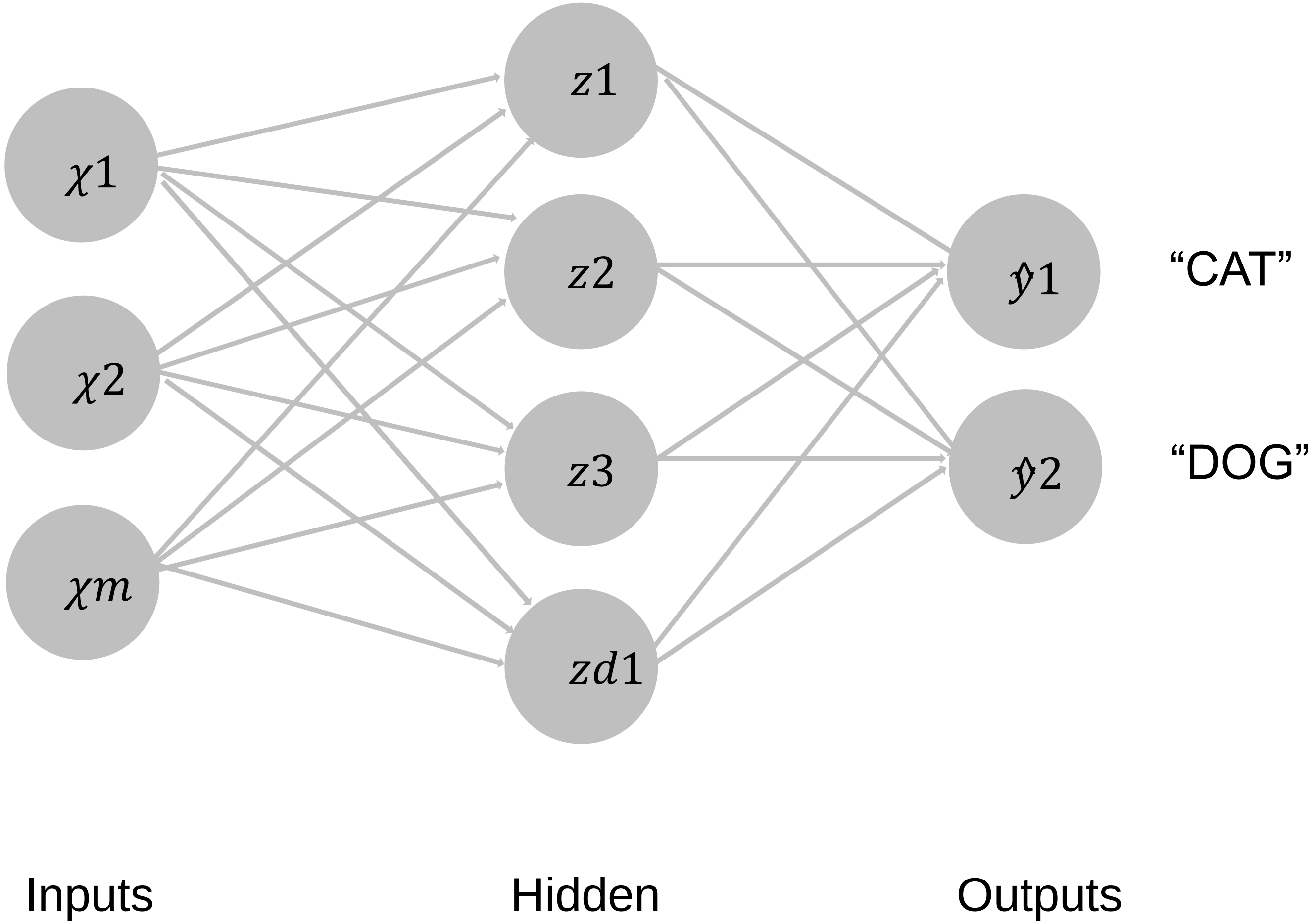
Evaluate performance
- Compute the loss of the prediction

Back propagate
- Update weights according to the optimization algorithm

Improving the network
- Regularization,
- Advanced architectures

Building the network

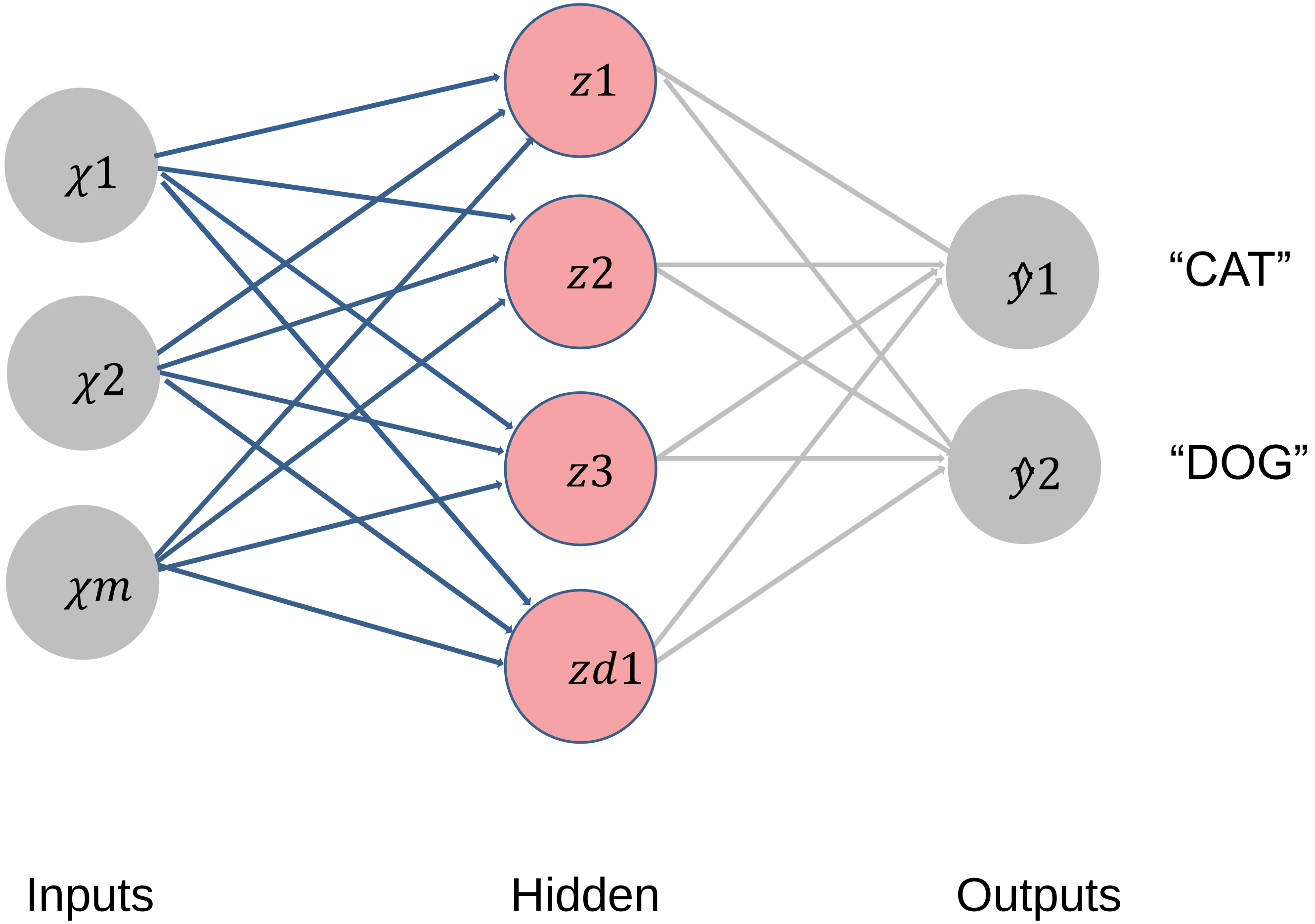
$$\begin{bmatrix} \chi 1 \\ \chi 2 \\ \chi m \end{bmatrix}$$



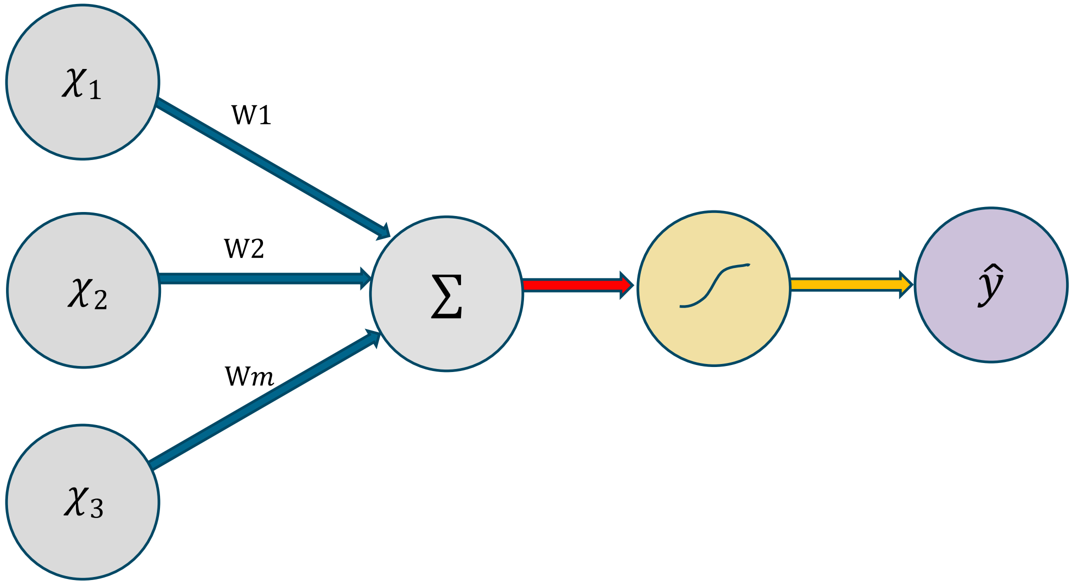
Forward Propagate Input

$$\begin{bmatrix} \chi 1 \\ \chi 2 \\ \chi m \end{bmatrix}$$

Calculating weighted sum
Add non-linearity
Make prediction



Forward Propagate Input
A single neuron (perceptron)



Inputs Weights Sum Non-Linearity Output

Output

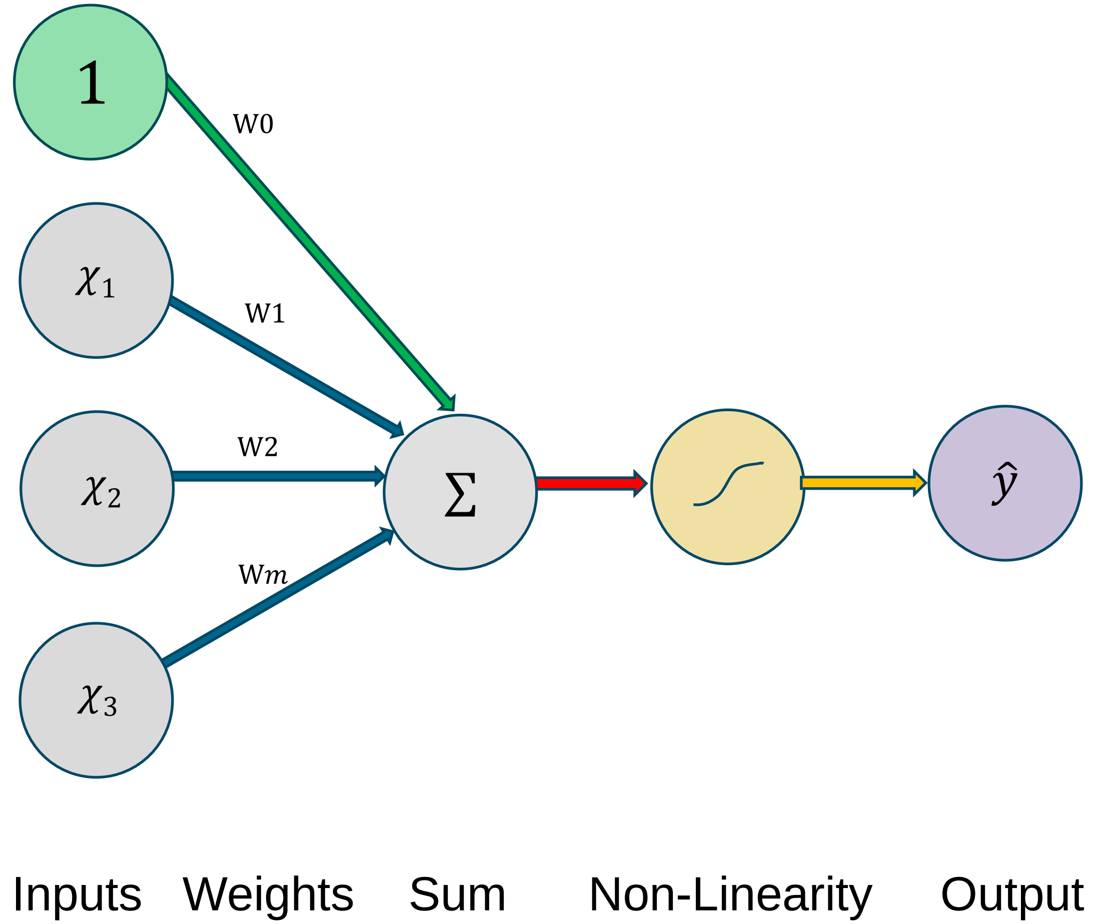
Linear combination of inputs

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i\, w_i\right)$$

Non-linear activation function

Forward Propagate Input

A single neuron (perceptron)

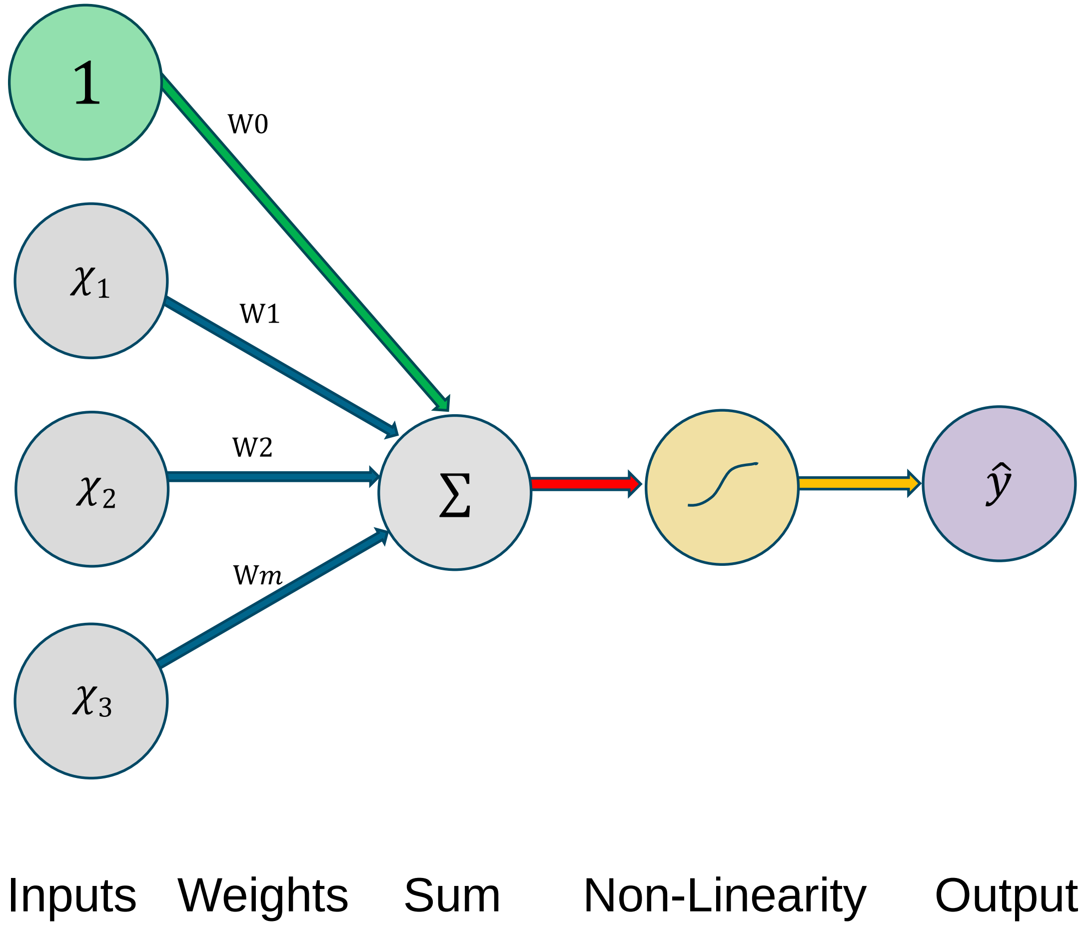


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Bias

Forward Propagate Input
A single neuron (perceptron)

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$



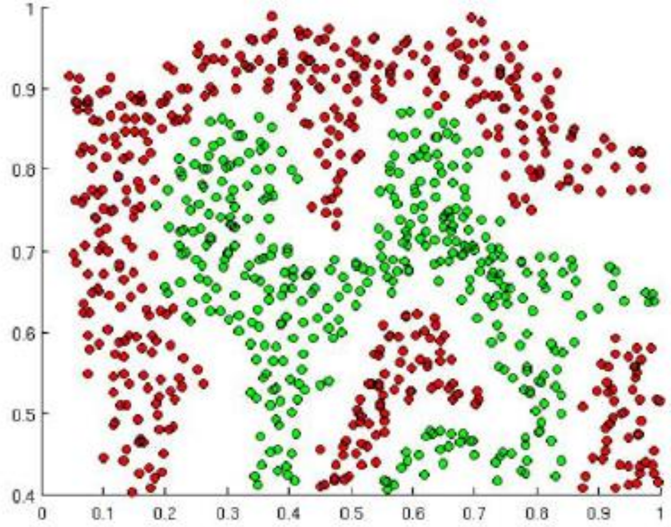
Vector/ Matrix operations

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

$$\boldsymbol{X} = \begin{bmatrix} X_1 \\ \vdots \\ Xm \end{bmatrix} \qquad \boldsymbol{W} = \begin{bmatrix} W1 \\ \vdots \\ Wm \end{bmatrix}$$

Forward Propagate Input
Activation functions

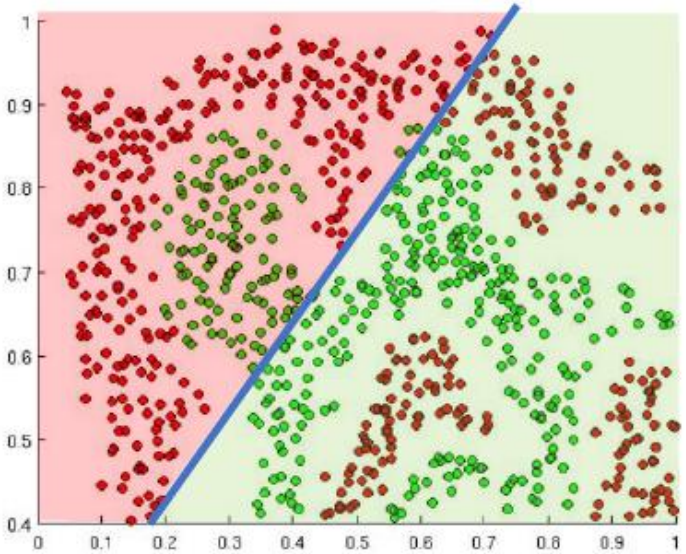
The purpose of activation functions is to ***introduce non-linearities*** *into the network*



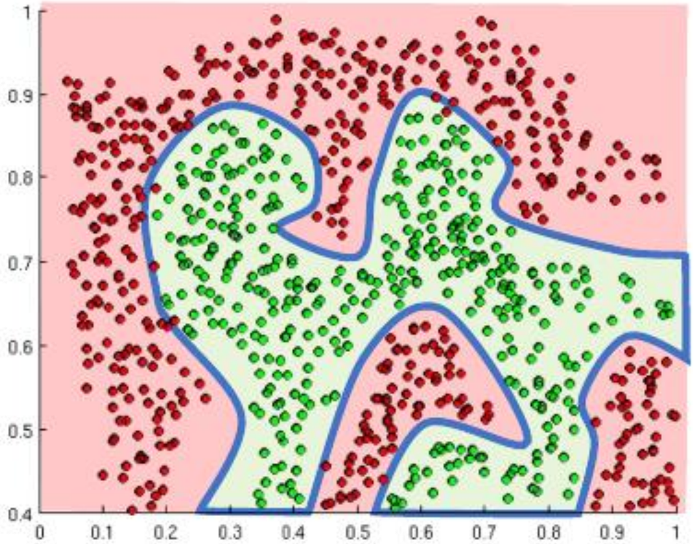
What if we wanted to build a Neural Network to distinguish green vs red points?

Forward Propagate Input
Activation functions

The *purpose of activation functions is to* ***introduce non-linearities*** *into the network*



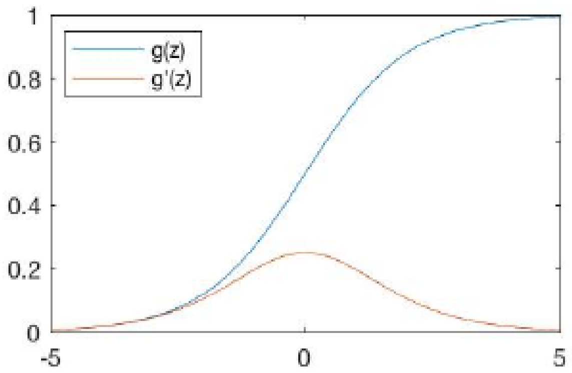
Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Forward Propagate Input
Non-linear Activation functions

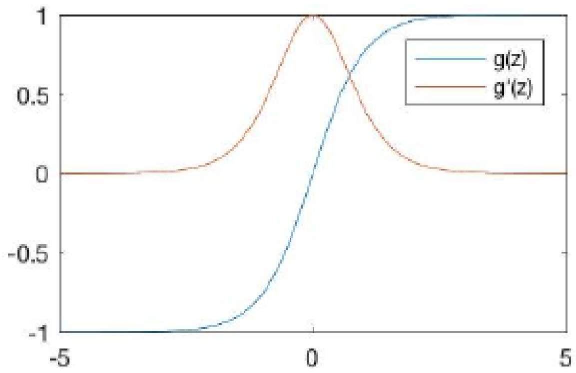
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

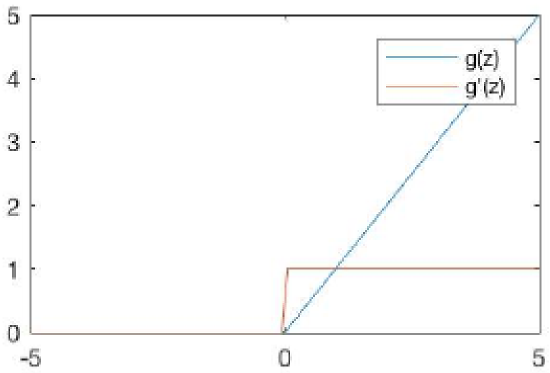
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)

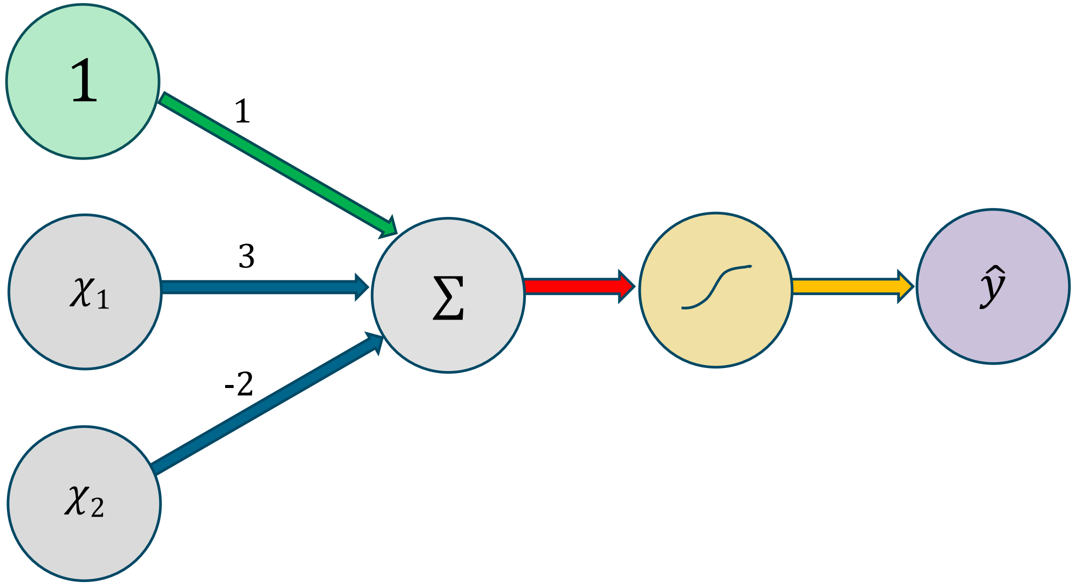


$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Forward Propagate Input

A single neuron (perceptron) - Example



$$\hat{y} = g\ (w_0 + \boldsymbol{X}^T\ \boldsymbol{W})$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g\underbrace{(1 + 3x_1 - 2x_2)}_{\text{This is just a line in 2D}}$$

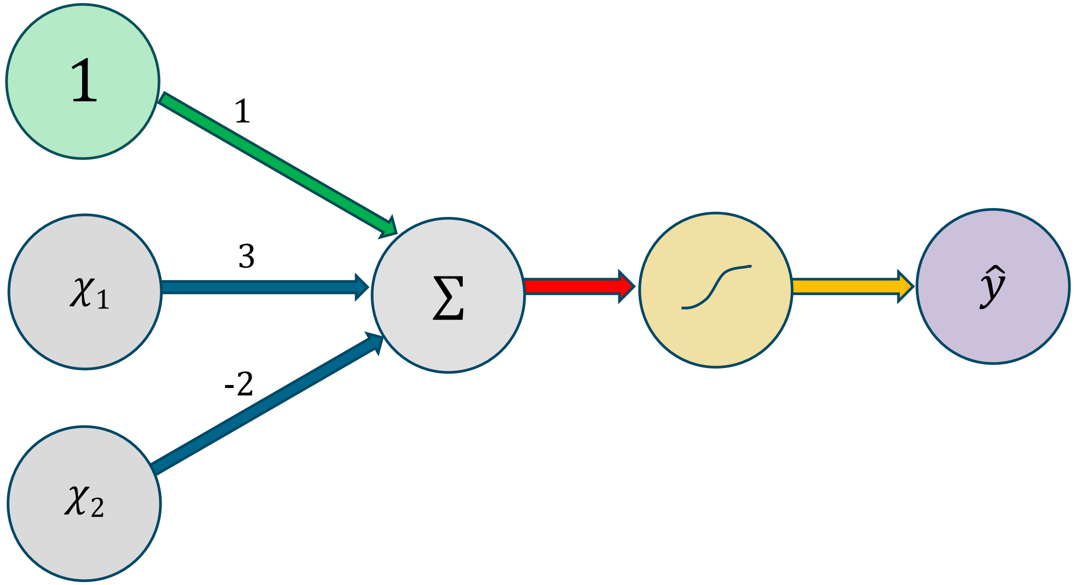
$\boldsymbol{w_0}$ =1

$W = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

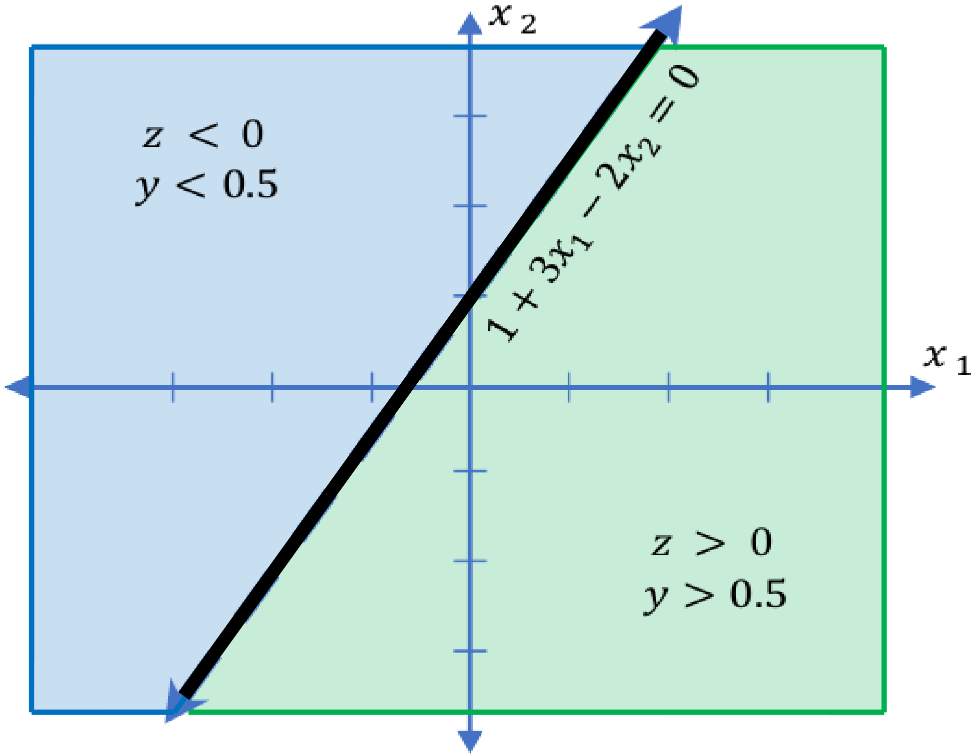
Forward Propagate Input
A single neuron (perceptron) - Example

$w_0 = 1$

$W = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

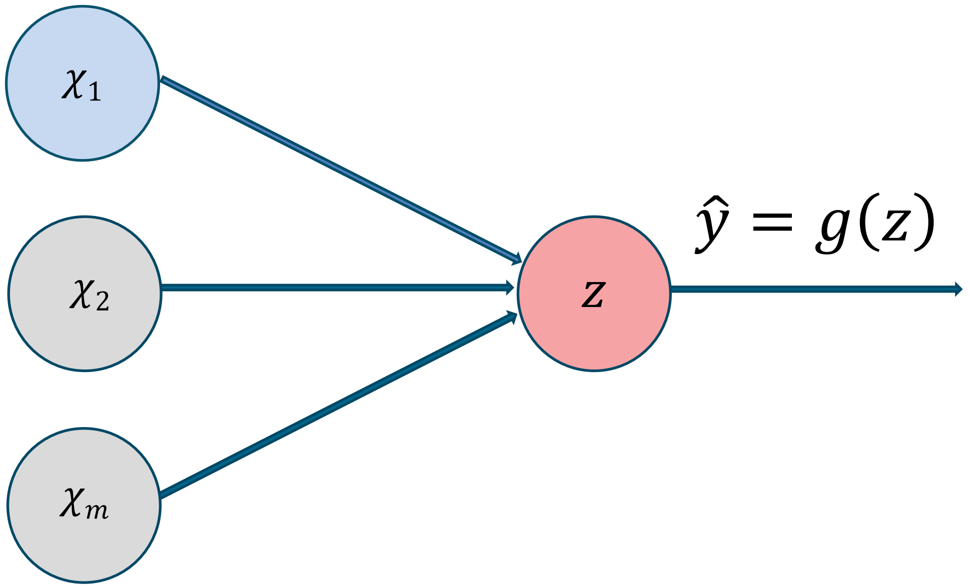


$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



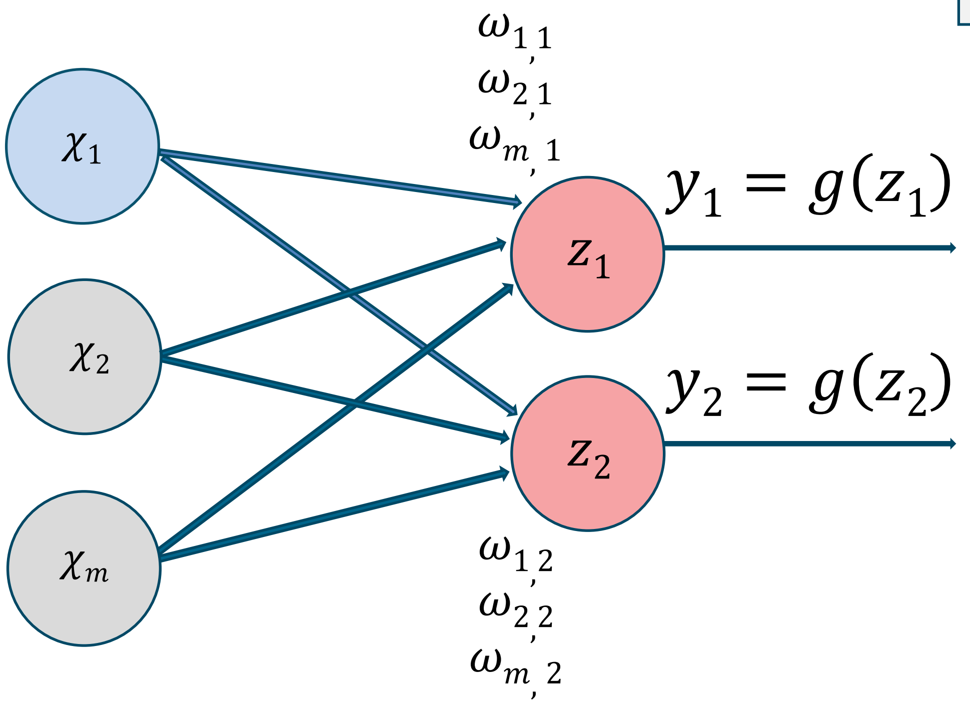
Forward Propagate Input
Simplified neuron

$$z = \left(w_0 + \sum_{j=1}^{m} x_j w_j \right)$$



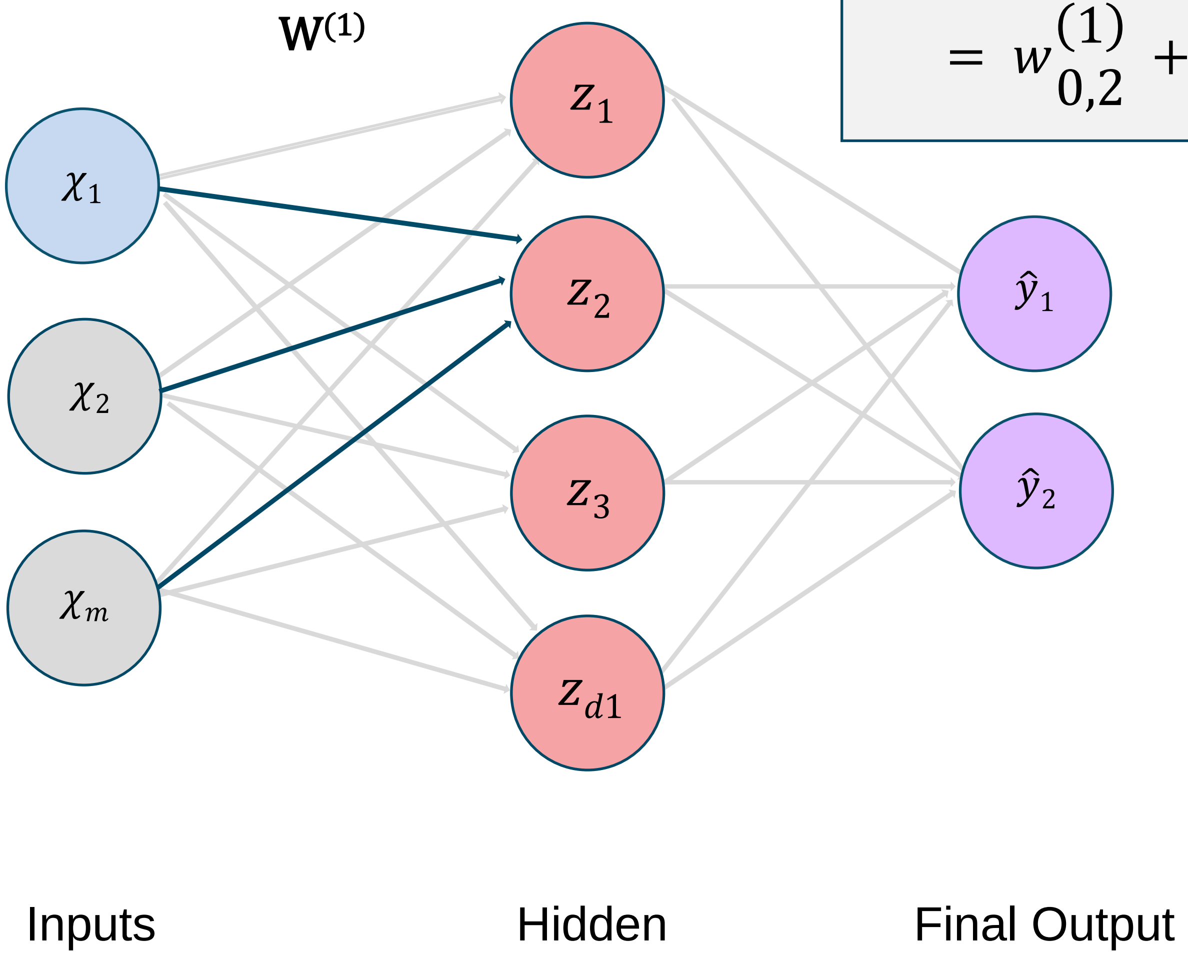
Forward Propagate Input
Multi-output neuron

$$z_i = \left(w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i} \right)$$



Forward Propagate Input
A single-layer neural network

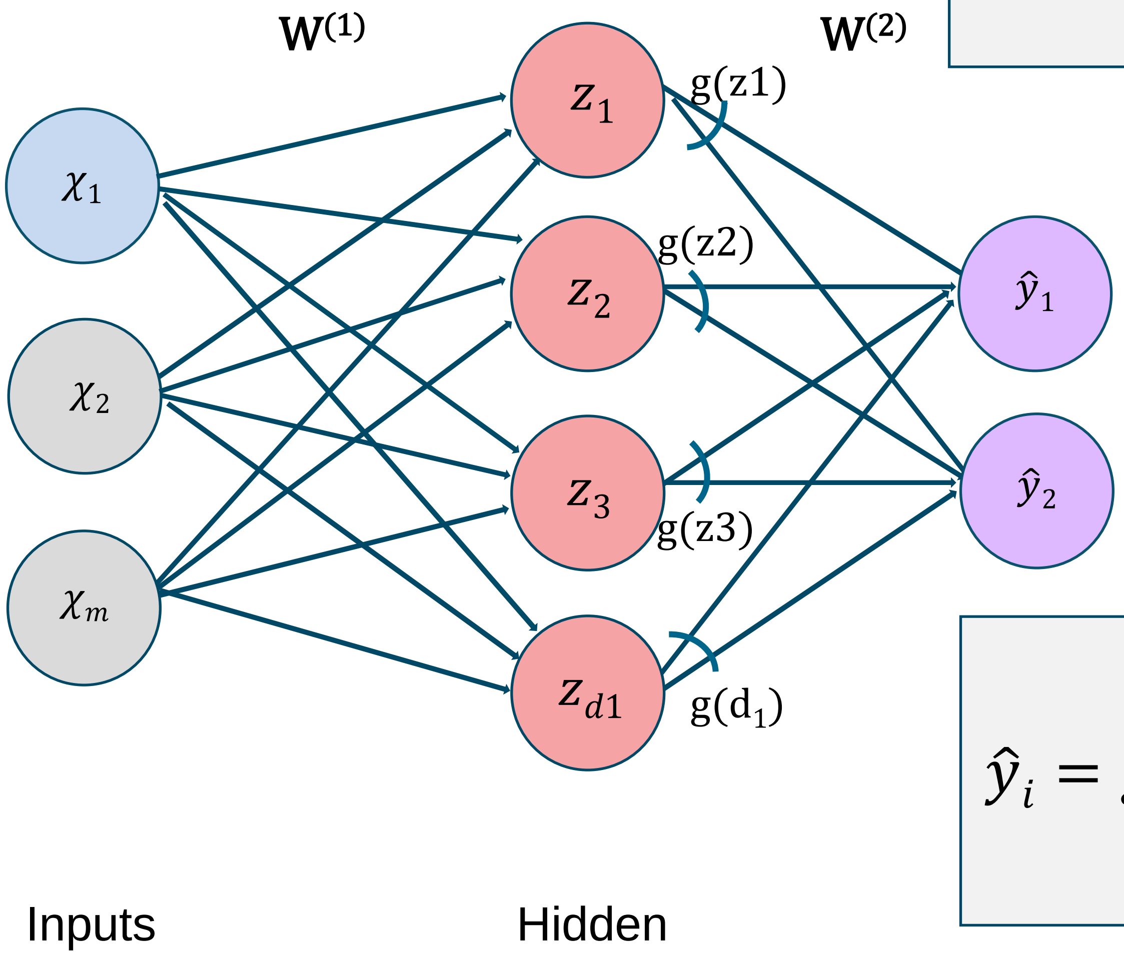
$$z_2 = \left(w^{(1)}_{0,2} + \sum_{j=1}^{m} x_j \left(w^{(1)}_{j,2} \right) \right)$$
$$= w^{(1)}_{0,2} + \underbrace{x_1 w^{(1)}_{1,2} + x_2 w^{(1)}_{2,2} + x_m w^{(1)}_{m,2}}$$



Forward Propagate Input

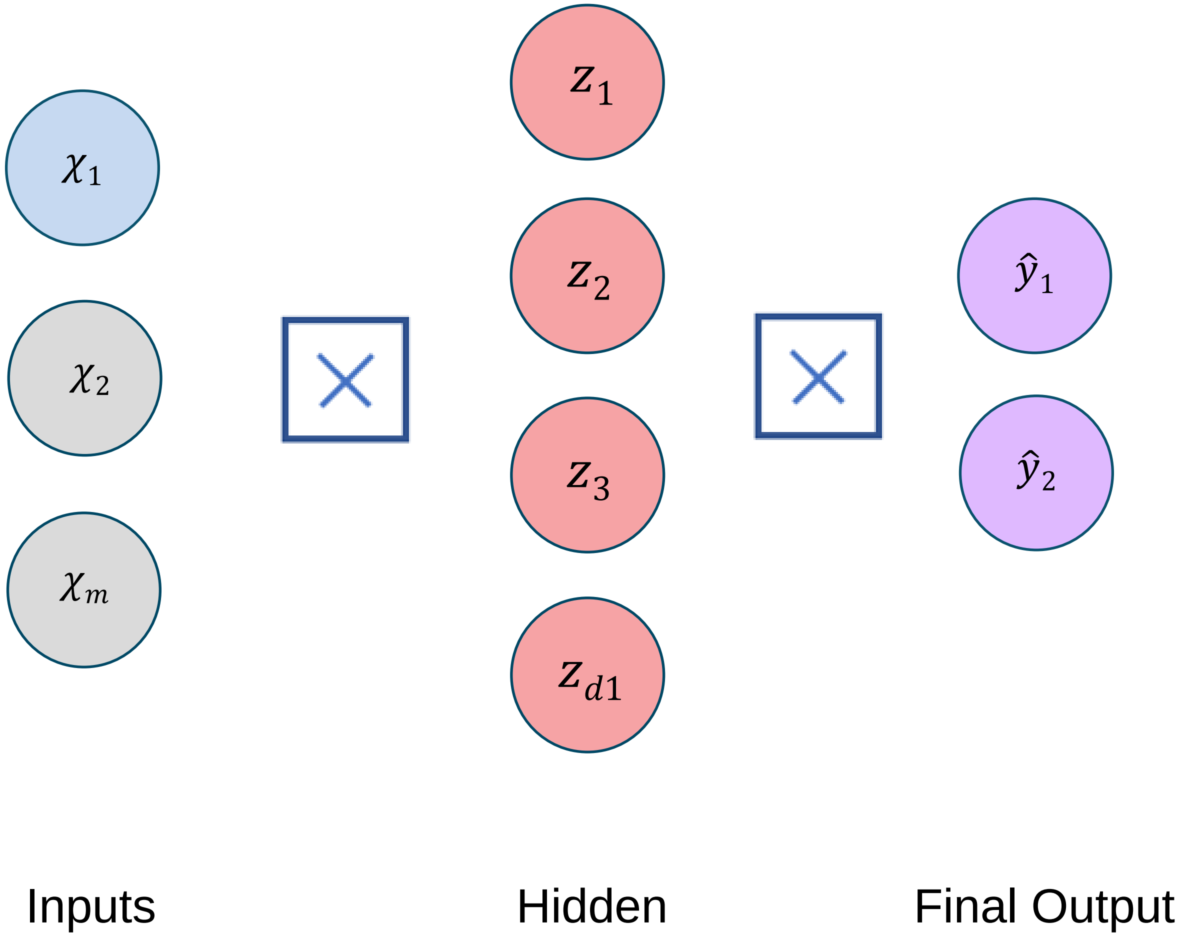
A single-layer neural network

$$z_i = \left(w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \right)$$



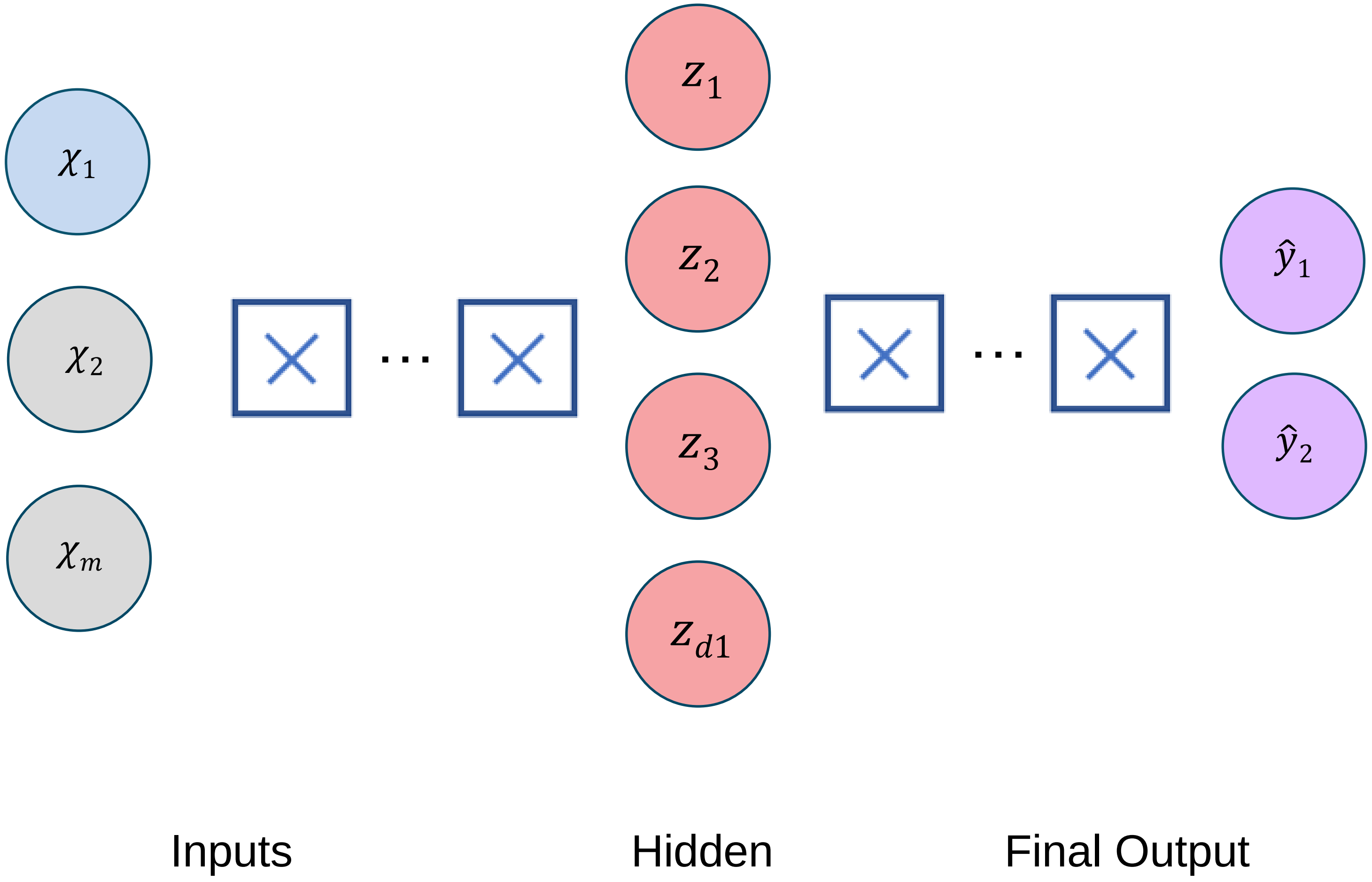
$$\hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)} \right)$$

Forward Propagate Input
A single-layer neural network



Forward Propagate Input
Deep neural network

$$z_{k,i} = \left(w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g\left(z_{k-1,j}\right) w_{j,i}^{(k)} \right)$$

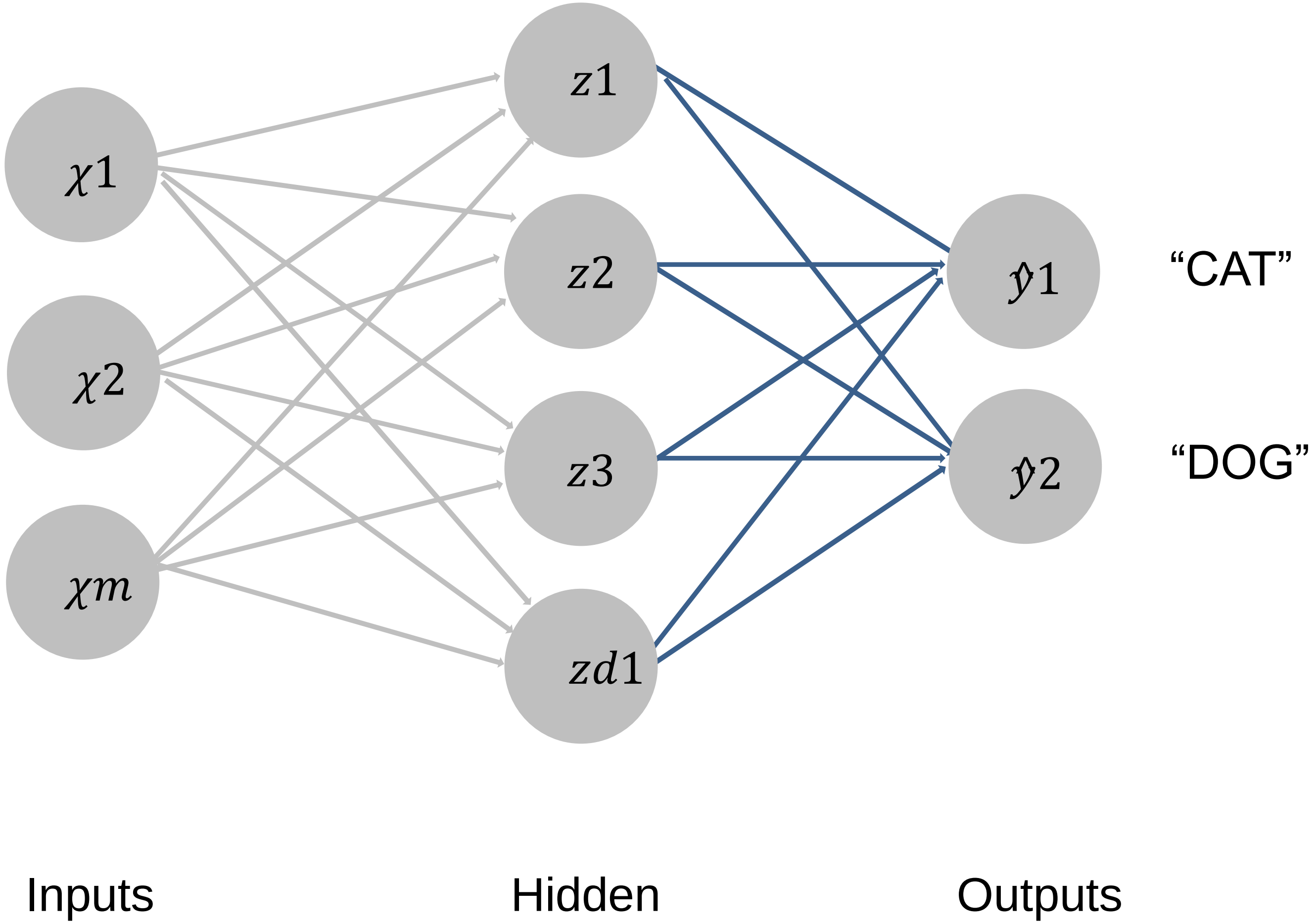


Forward Propagate Input

CAT

$$\begin{bmatrix} \chi 1 \\ \chi 2 \\ \chi m \end{bmatrix}$$

Calculating weighted sum
Add non-linearity
Make prediction



Forward Propagate Input

Softmax activation
(= softargmax = normalized exponential function)

$$S\left(f_{y_i}\right) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

$$S\left(\mathrm{x_i}\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Input pixels, $\mathbf{x}$

Forward propagation →

Feedforward output, $\mathbf{y}_i$

cat	dog	horse
5	4	2
4	2	8
4	4	1

Softmax function →

Softmax output, $\mathbf{S}(\mathbf{y}_i)$

cat	dog	horse
0.71	0.26	0.04
0.02	0.00	0.98
0.49	0.49	0.02

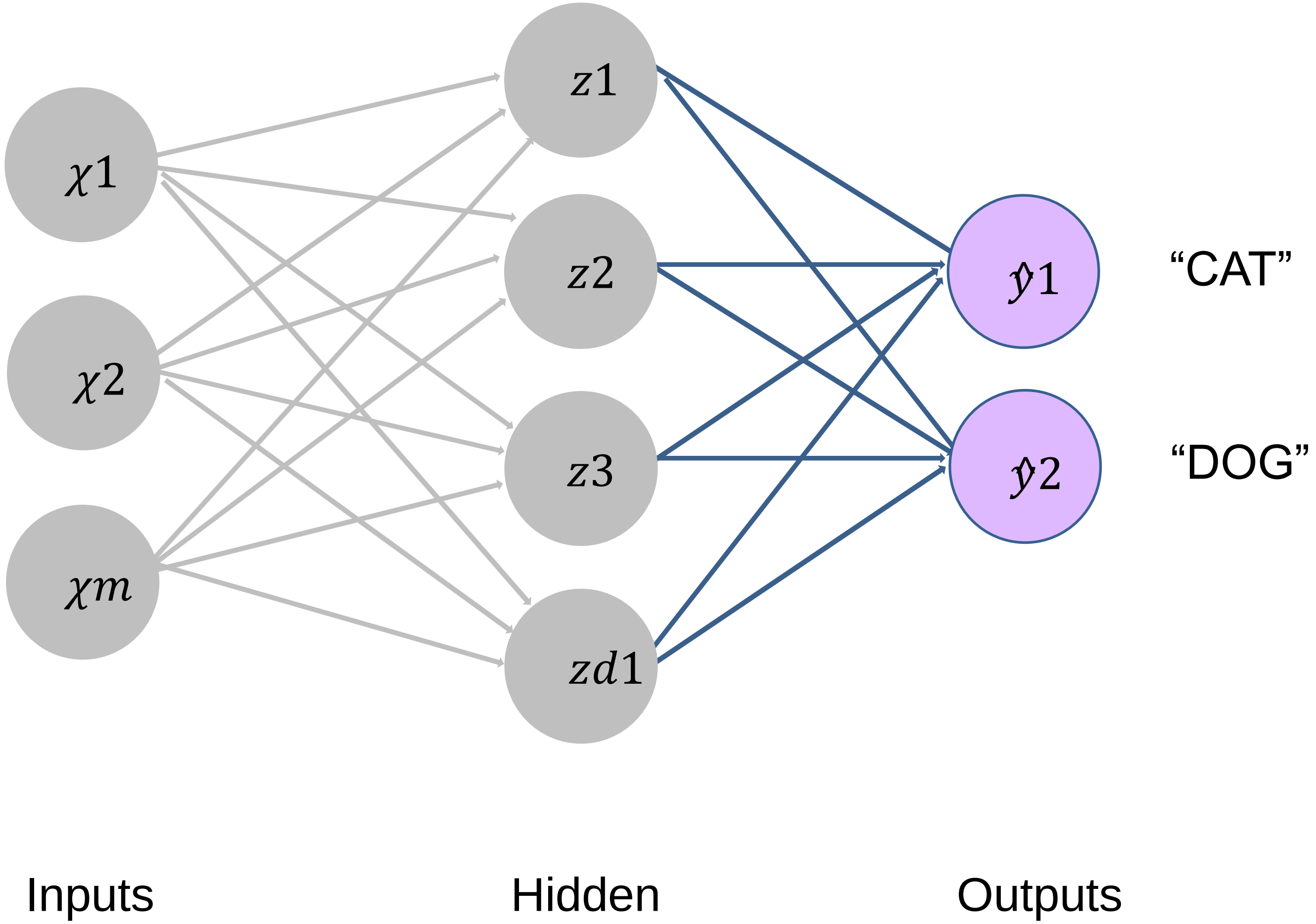
https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

Evaluate Prediction

Compute the error and its gradient

CAT

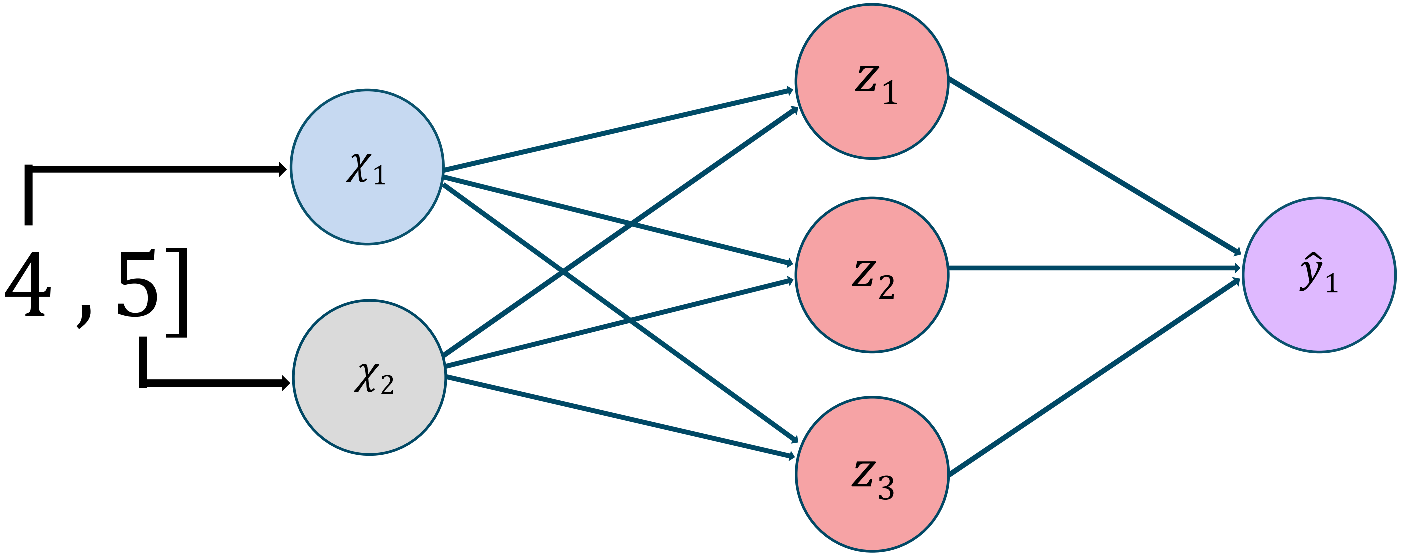
$$\begin{bmatrix} \chi 1 \\ \chi 2 \\ \chi m \end{bmatrix}$$



Evaluate Prediction
Loss function

The **loss** of our network measures the cost incurred from incorrect predictions

$$x^{(1)} = [4\,,5]$$



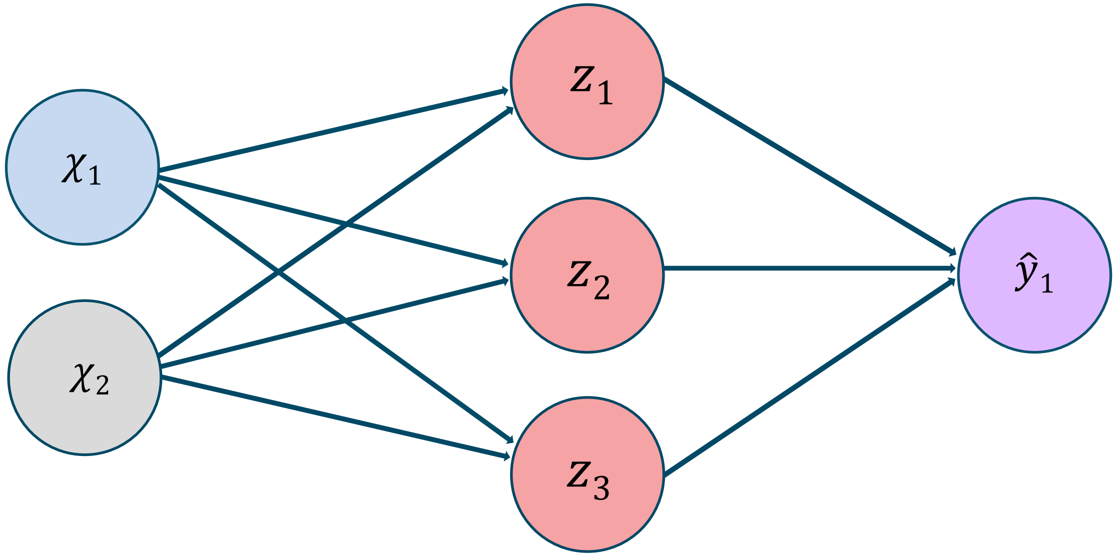
Predicted: **0.1**
Actual: **1**

$$\mathcal{L}(f(\underbrace{x^{(i)};\boldsymbol{W}}_{\text{Predicted}}),\underbrace{y^{(i)}}_{\text{Actual}})$$

Evaluate Prediction
Loss function

The **empirical loss** measures the total loss over our entire dataset

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

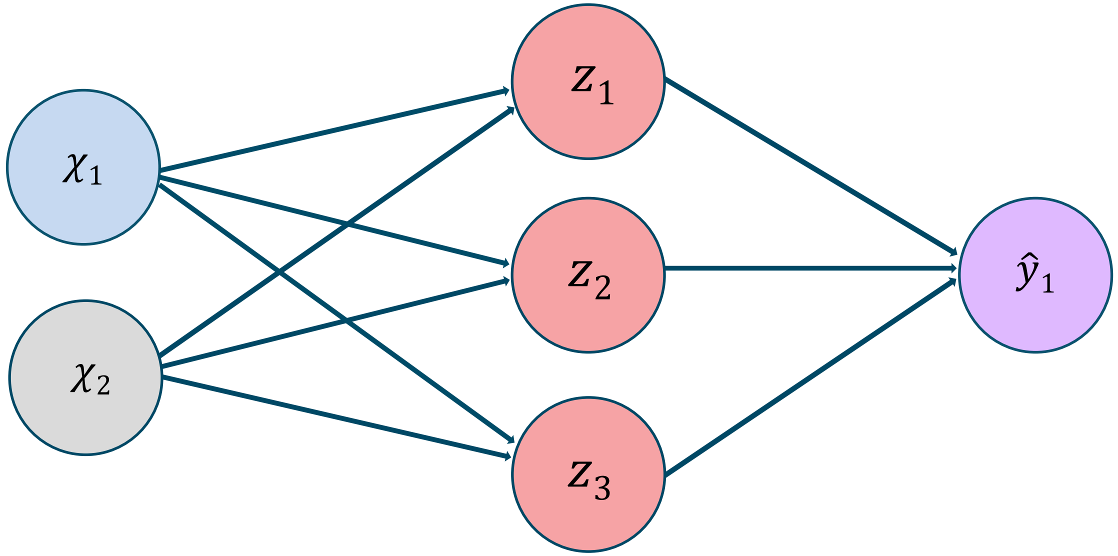
$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(\underbrace{x^{(i)};\boldsymbol{W}}_{\text{Predicted}}), \underbrace{y^{(i)}}_{\text{Actual}})$$

Also known as:
- Objective function
- Cost function
- Empirical Risk

Evaluate Prediction
Binary cross-entropy loss

The **Cross entropy loss** can be used with models that output a probability between 0 and 1

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



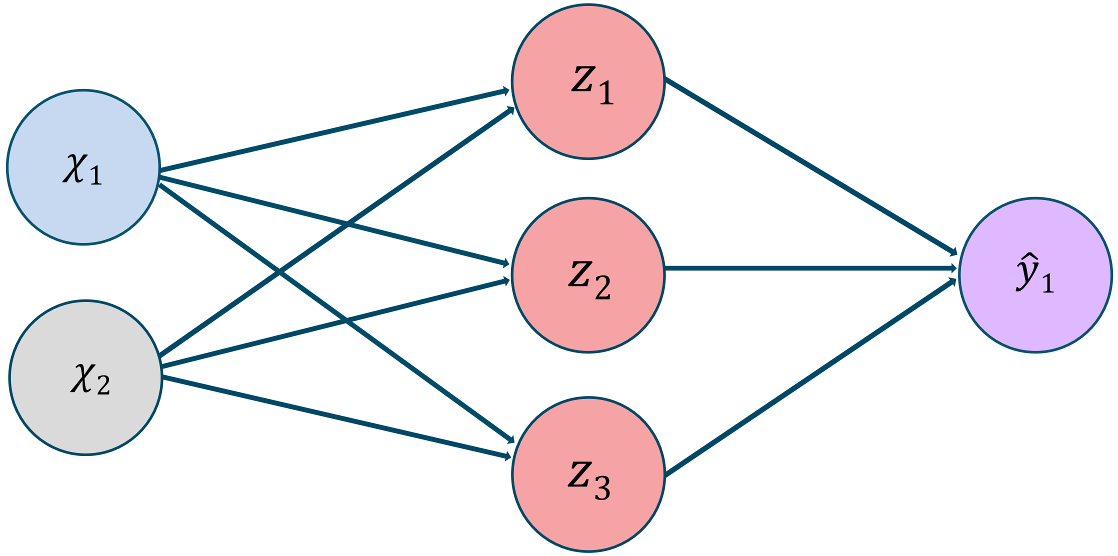
$$\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log(f(\underbrace{x^{(i)};\boldsymbol{W}}_{\text{Predicted}})) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log(1 - f(\underbrace{x^{(i)};\boldsymbol{W}}_{\text{Predicted}}))$$

Evaluate Prediction
Mean squared error loss

Mean squared error loss can be used with regression models that output continuous real numbers

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} & \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix} \end{array}$$

Final Grades (percentage)

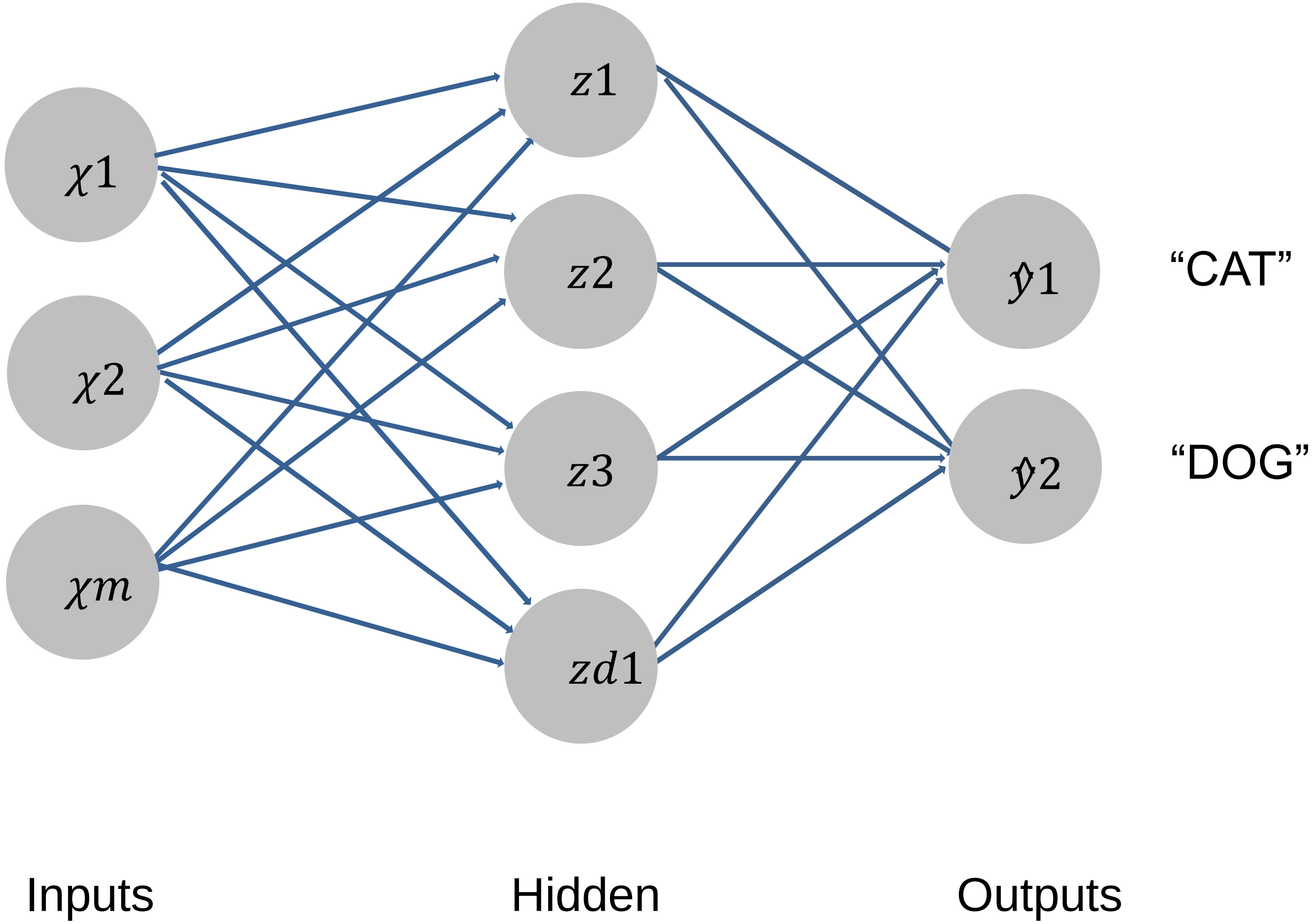
$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)^{\mathbf{2}}$$

Back propagate

CAT

$$\begin{bmatrix} \chi 1 \\ \chi 2 \\ \chi m \end{bmatrix}$$

Update weights according to the optimization algorithm



Back Propagate
Loss optimization

For **optimization** we want to find the network weights that **achieve the lowest loss**

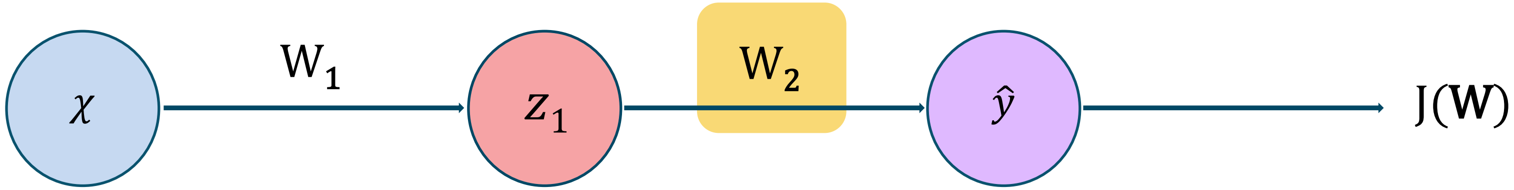
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{argmin} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}; \boldsymbol{W}), y^{(i)})$$

$\boldsymbol{w_0} = 1$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{argmin}\, \boldsymbol{J}(\mathbf{W})$$

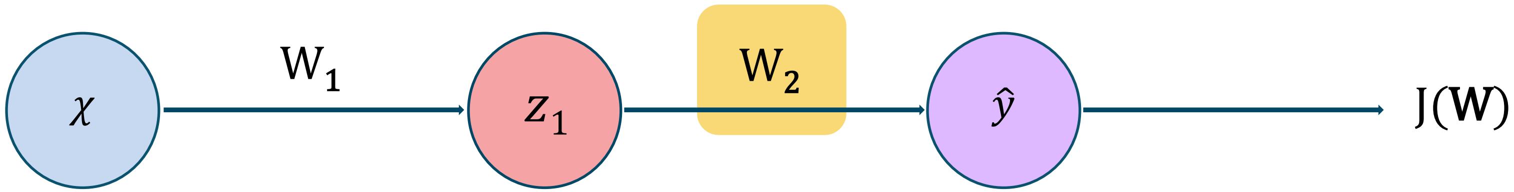
$$\mathbf{W} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots\}$$

Back Propagate



How does a small change in one weight (ex. w_2) affect the final loss $J(\mathbf{W})$

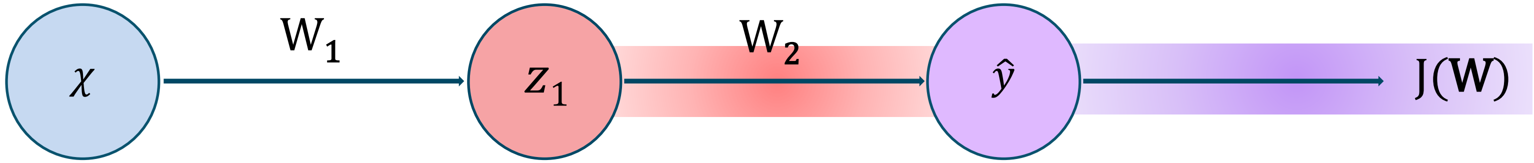
Back Propagate



$$\frac{\partial J(W\,)}{\partial w_2} =$$

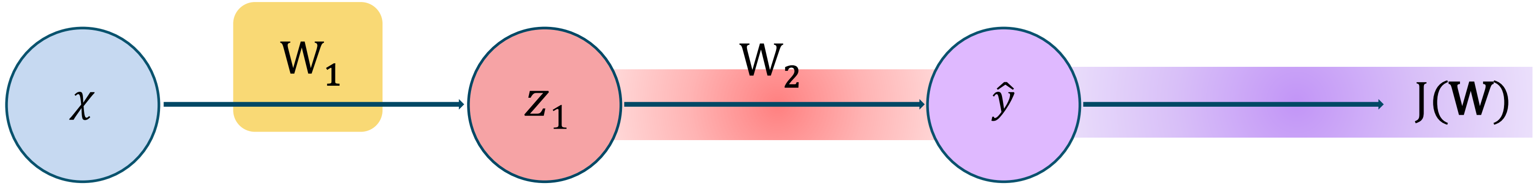
Let’s use the chain rule!

Back Propagate



$$\boxed{\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}}$$

Back Propagate

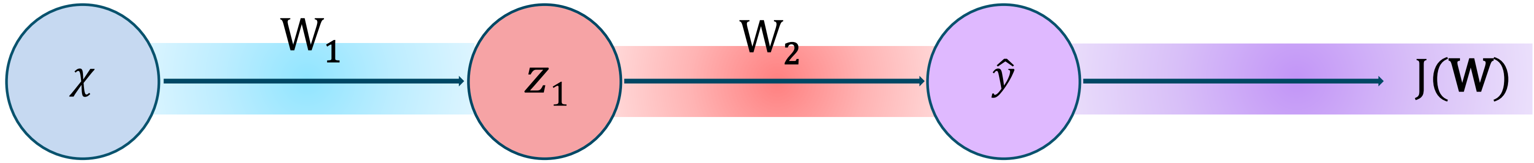


$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

Back Propagate



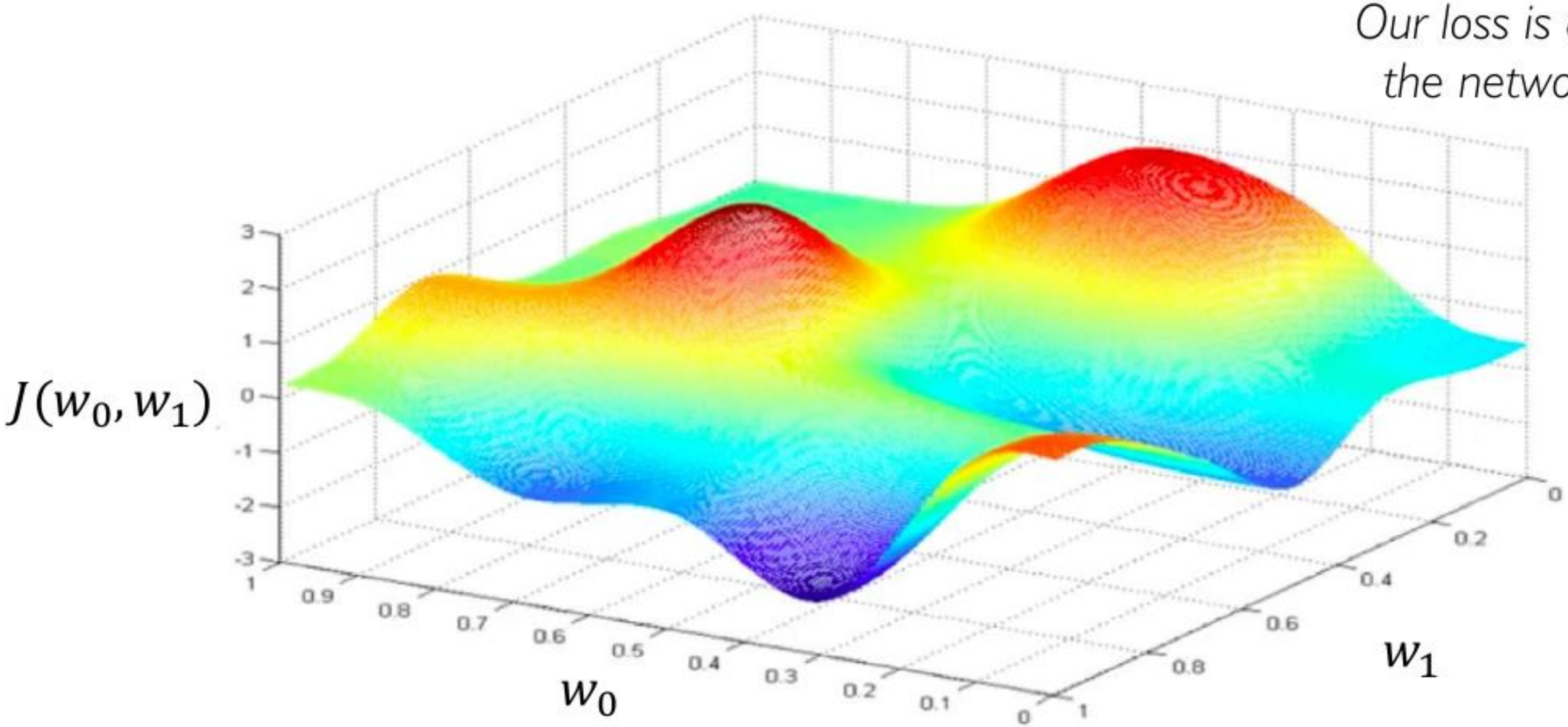
$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for **every weight in the network** using gradients from layers

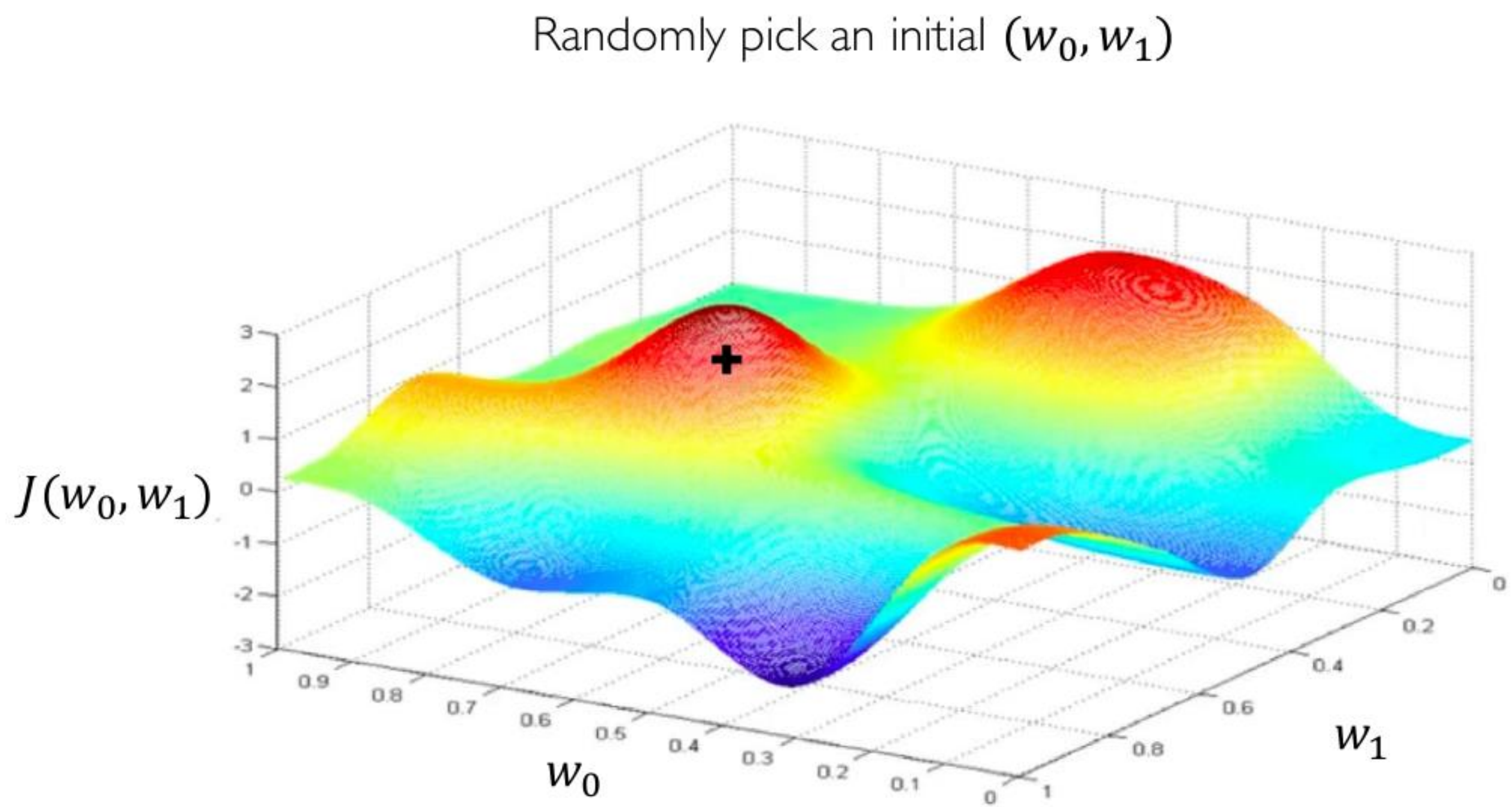
Back Propagate
Loss optimization

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$

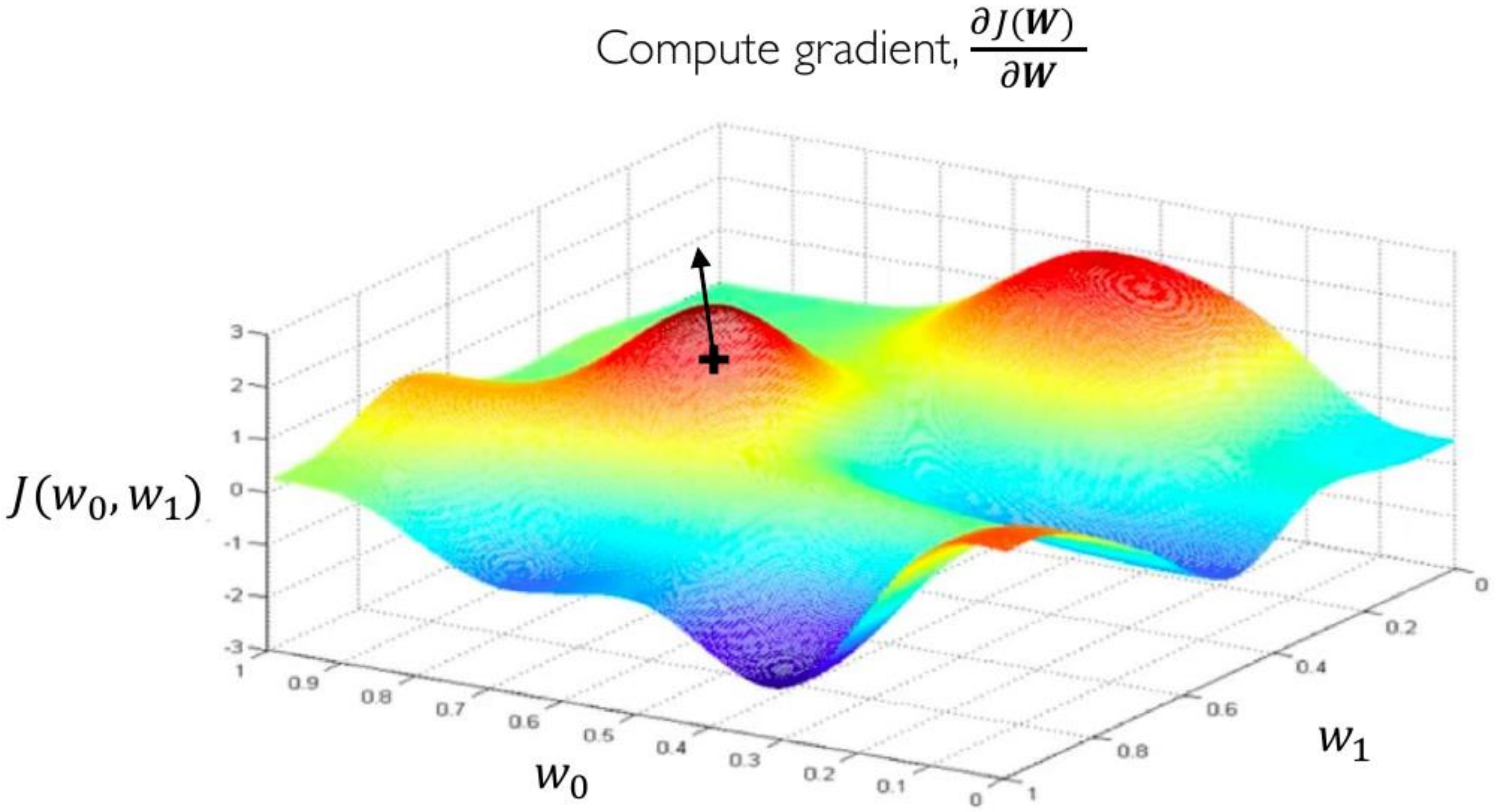
Remember:
Our loss is a function of the network weights!



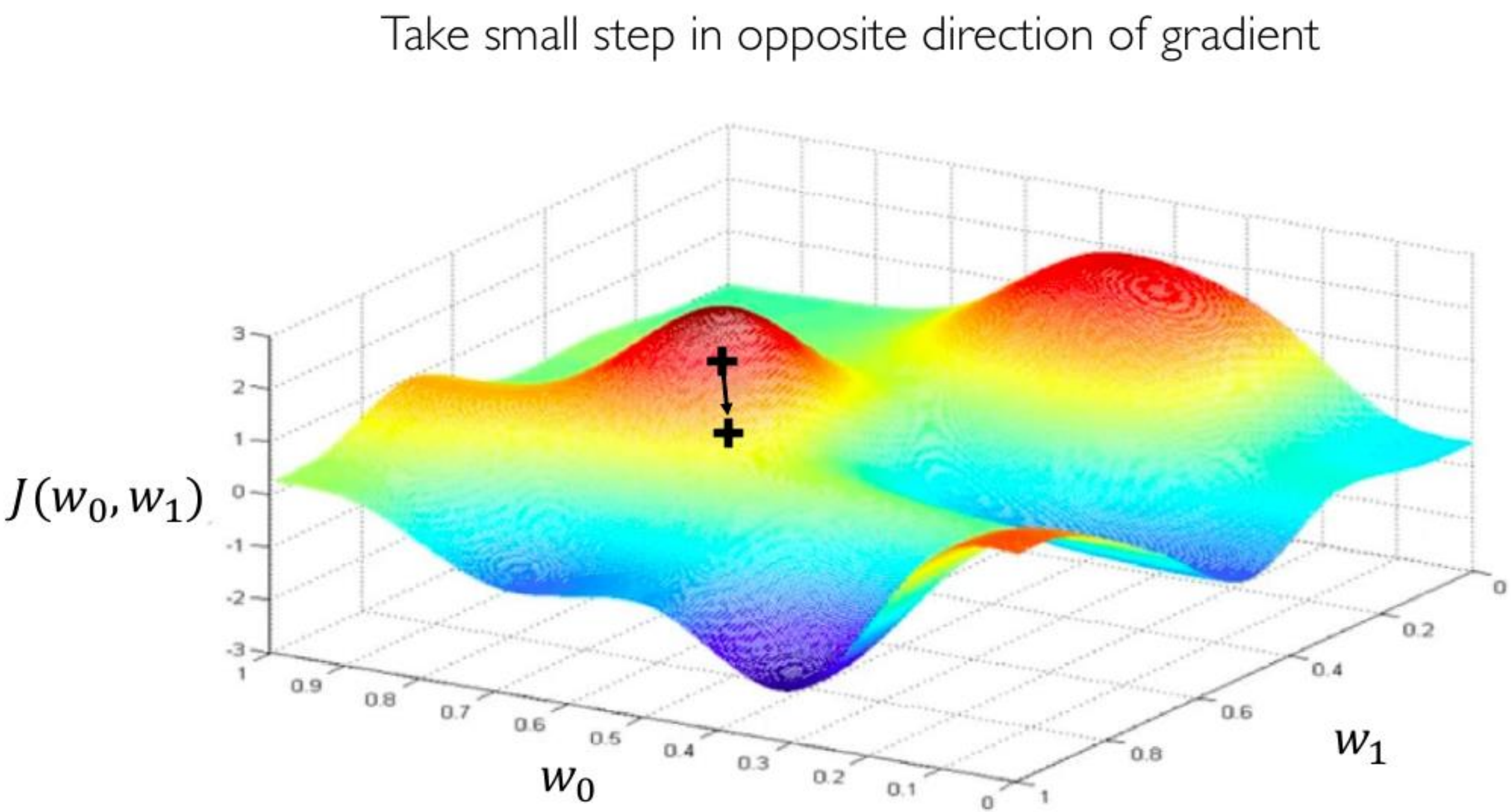
Back Propagate
Gradient descent



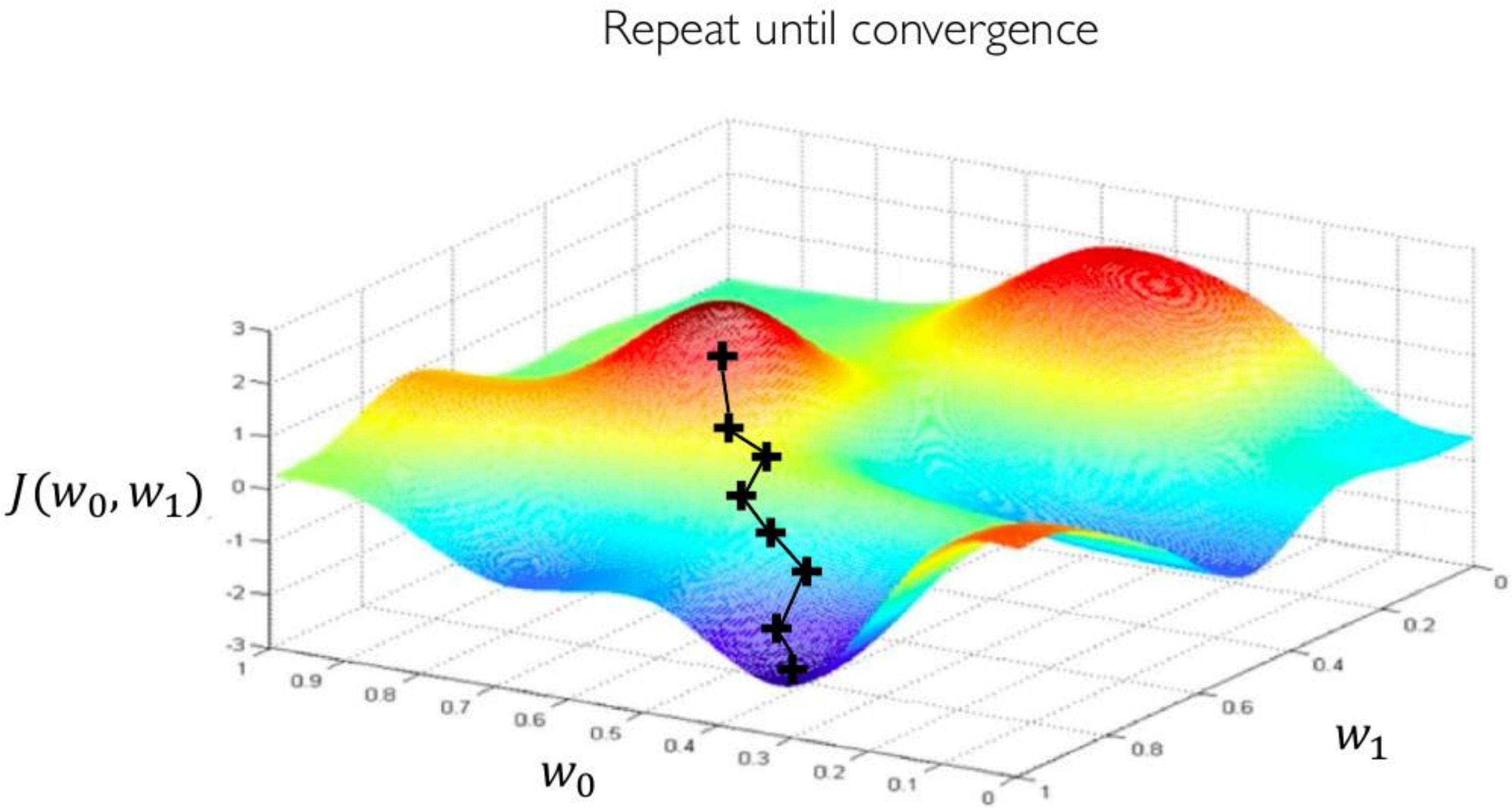
Back Propagate
Gradient descent



Back Propagate
Gradient descent



Back Propagate
Gradient descent



Back Propagate
Gradient descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial w}$
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} \leftarrow \eta \frac{\partial J(W)}{\partial w}$
5. Return weights

Curriculum

Building the network
- Input layer
- Hidden layers
- Output layer

Forward propagate input
- Propagate input through neurons
- Apply non-linearities

Evaluate performance
- Compute the loss of the prediction

Back propagate
- Update weights according to the optimization algorithm

Improving the network
- Regularization,
- Advanced architectures

Next:

- Hands-on I
- Theory Part II:
  - Advanced concept: Regularization
  - Convolutional Neural Networks

Imprint

Topic: **Introduction to Deep Learning**
Part I – Introduction and Basics

Date: 2025-11-13

Author: Auliya Fitri, Sai Vemuri, Sreerag Naveenachandran

Institute: Data Science