Introduction to Deep Learning
Part I - Basics

April 13, 2022

Auliya Fitri, Jakob Gawlikowski

Machine Learning Group
Institute of Data Science (DW)
Jena

Auliya Fitri

German Aerospace Center – Institute of Data Science
Data Management and Analysis
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

Jakob Gawlikowski

German Aerospace Center – Institute of Data Science
Data Management and Analysis
Machine Learning Group
PhD Student: Data Science in Earth Observation (TUM)

Research Interests:
- > Robust Machine Learning
- > Uncertainty in Neural Networks
- > Out-of-Distribution Detection
- > Data Fusion

Contact:
Phone: +49 3641 30960 143
Mail: jakob.gawlikowski@dlr.de

Curriculum

A: Theoretical introduction – Morning

I. Introduction and basics

II. Advanced concepts

III. Practical application

B. Hands-on seminar – Afternoon

Run prepared Jupyter Notebooks online on Binder, or locally on your own laptop.

Curriculum

I. Introduction and basics

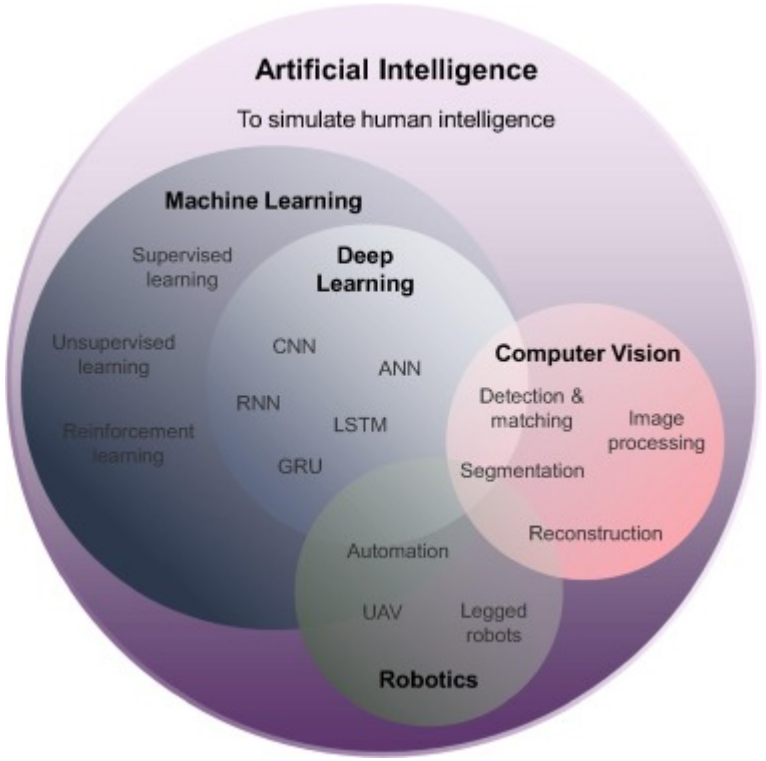
- Application examples
- Machine learning background
- Neural network concepts

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



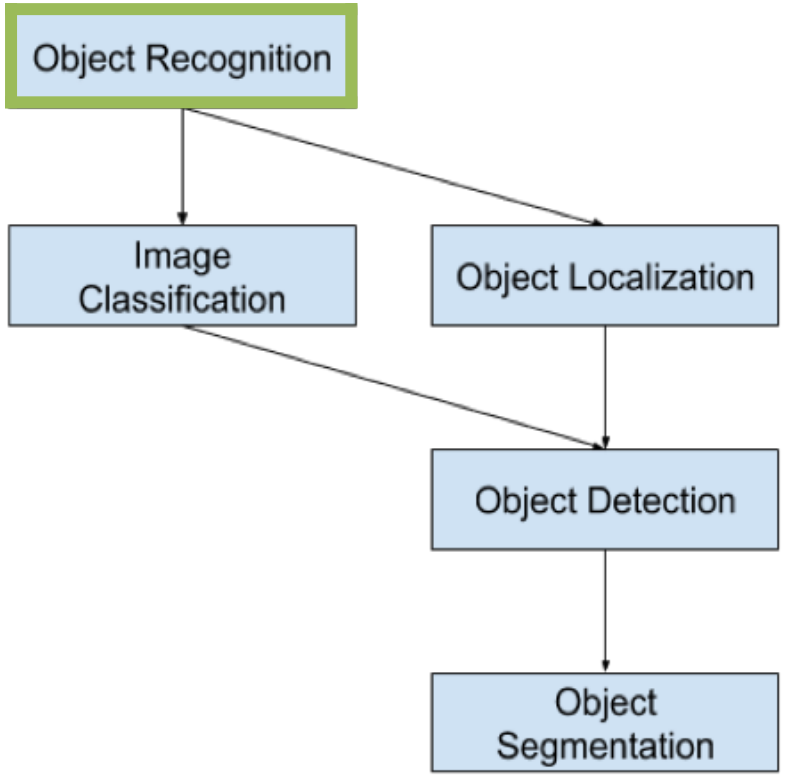
Computer Vision and Deep Learning – SevenShineStudios (wordpress.com)

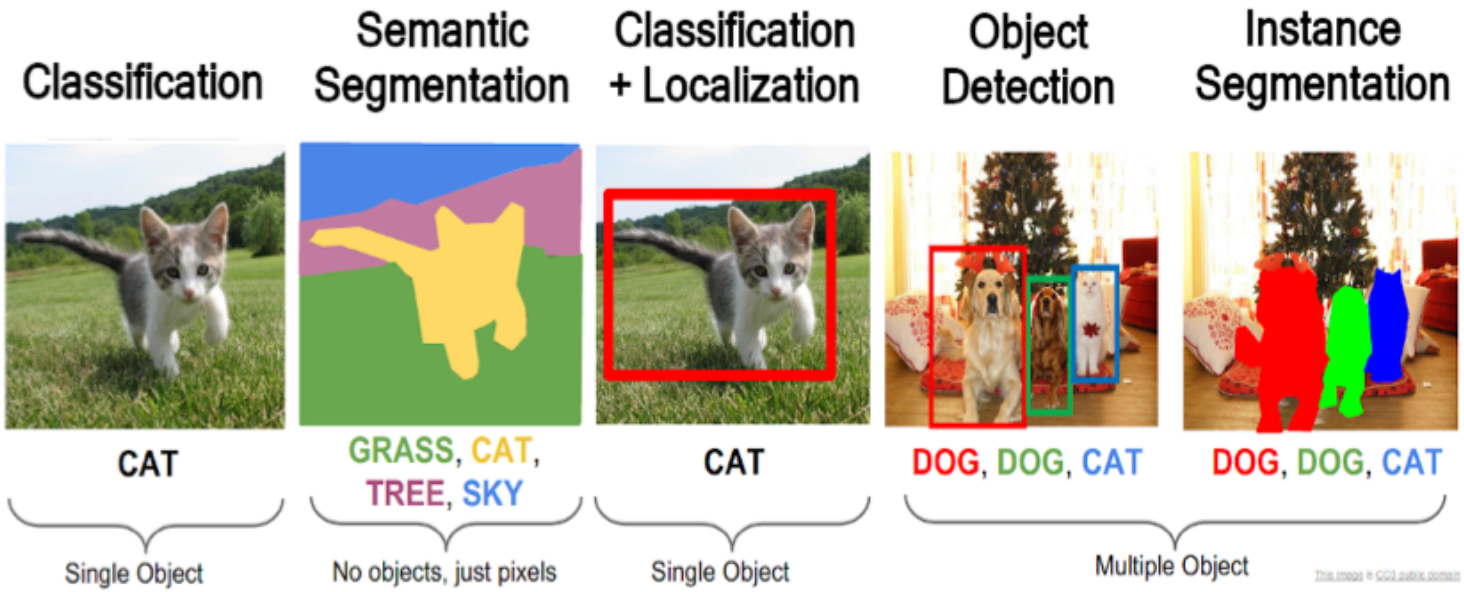
BIM, machine learning and computer vision techniques in underground construction: Current status and future perspectives - ScienceDirect

Application Examples

Computer Vision - Object Recognition / Image Processing

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





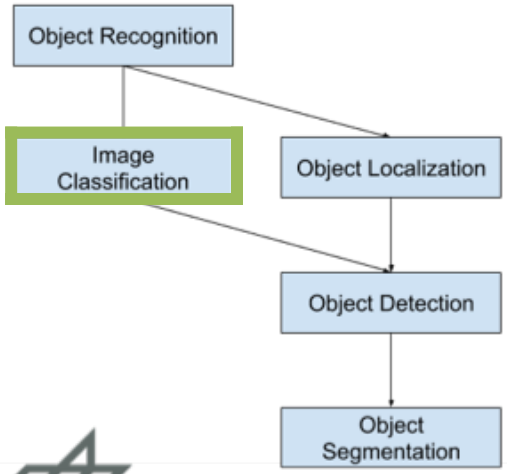
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

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.”

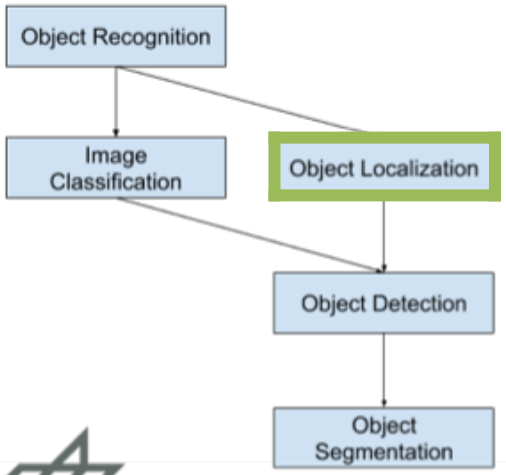


CAT

Application Examples

Computer Vision – Object Localization

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

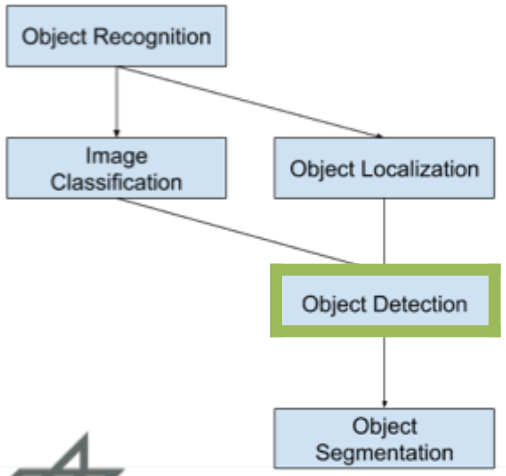


CAT

Application Examples

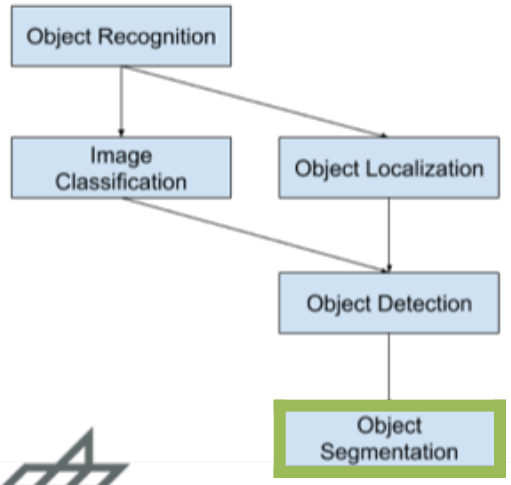
Computer Vision – Object Detection

"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

GRASS, CAT, TREE, SKY

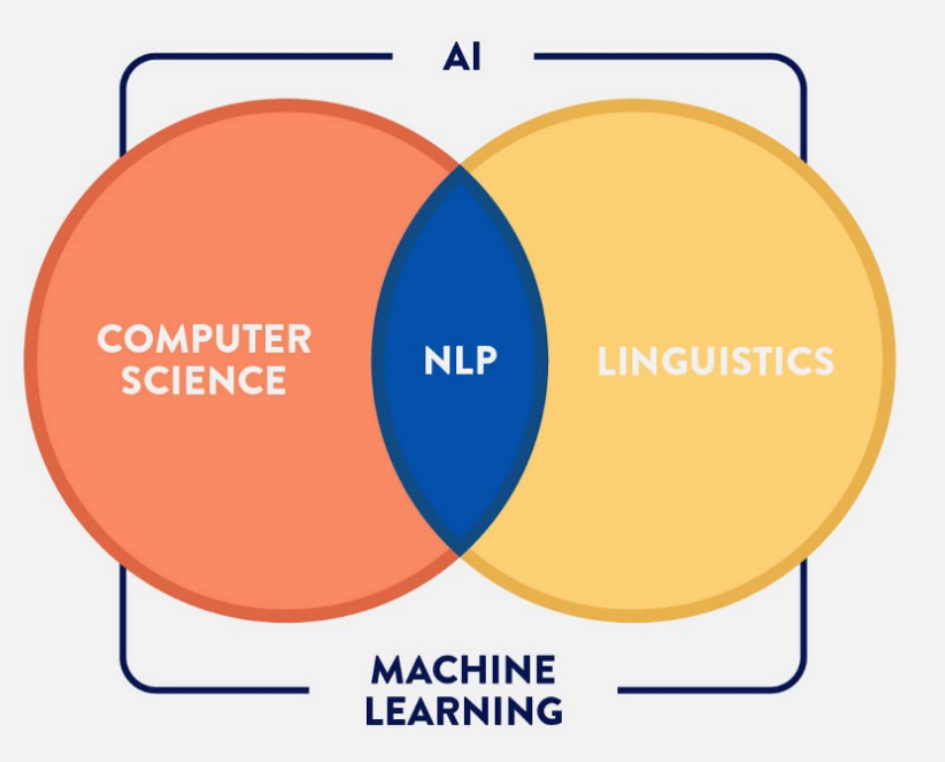
"Object Segmentation highlights the specific pixels of the object instead of a coarse bounding box."

DOG, DOG, CAT

Application Examples

Natural Language Processing (NLP)

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

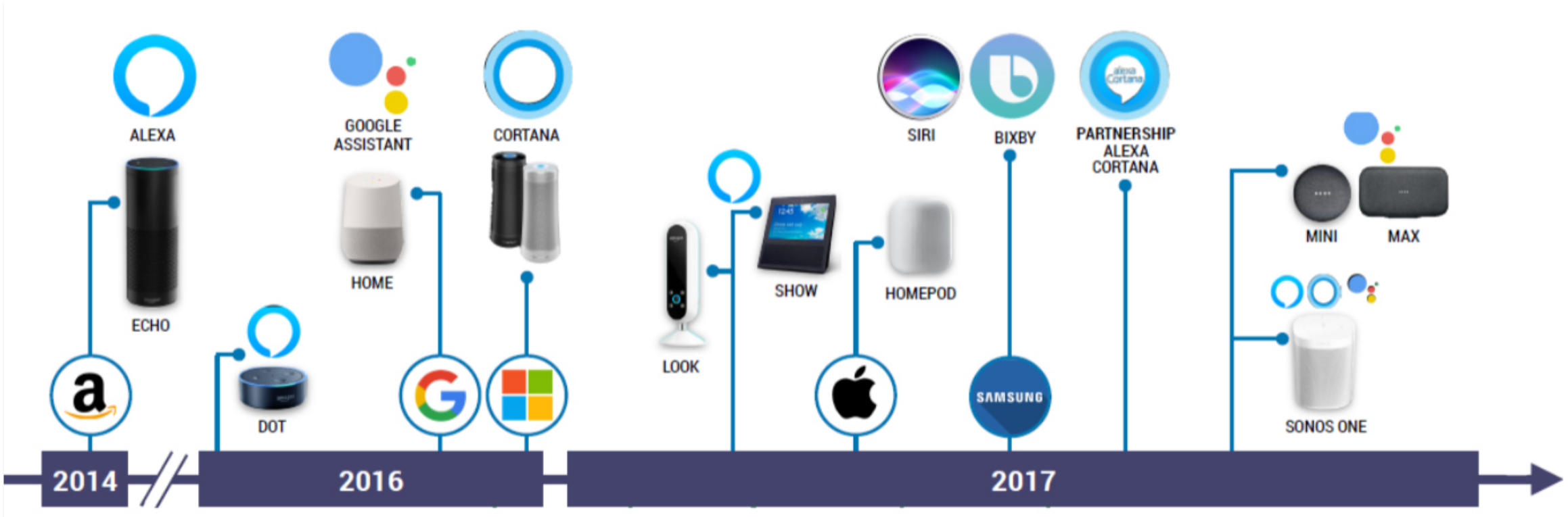


Introduction to Natural Language Processing (NLP) | by Ibrahim Sharaf ElDen | Towards Data Science

Application Examples

NLP – Virtual & Voice Assistants

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

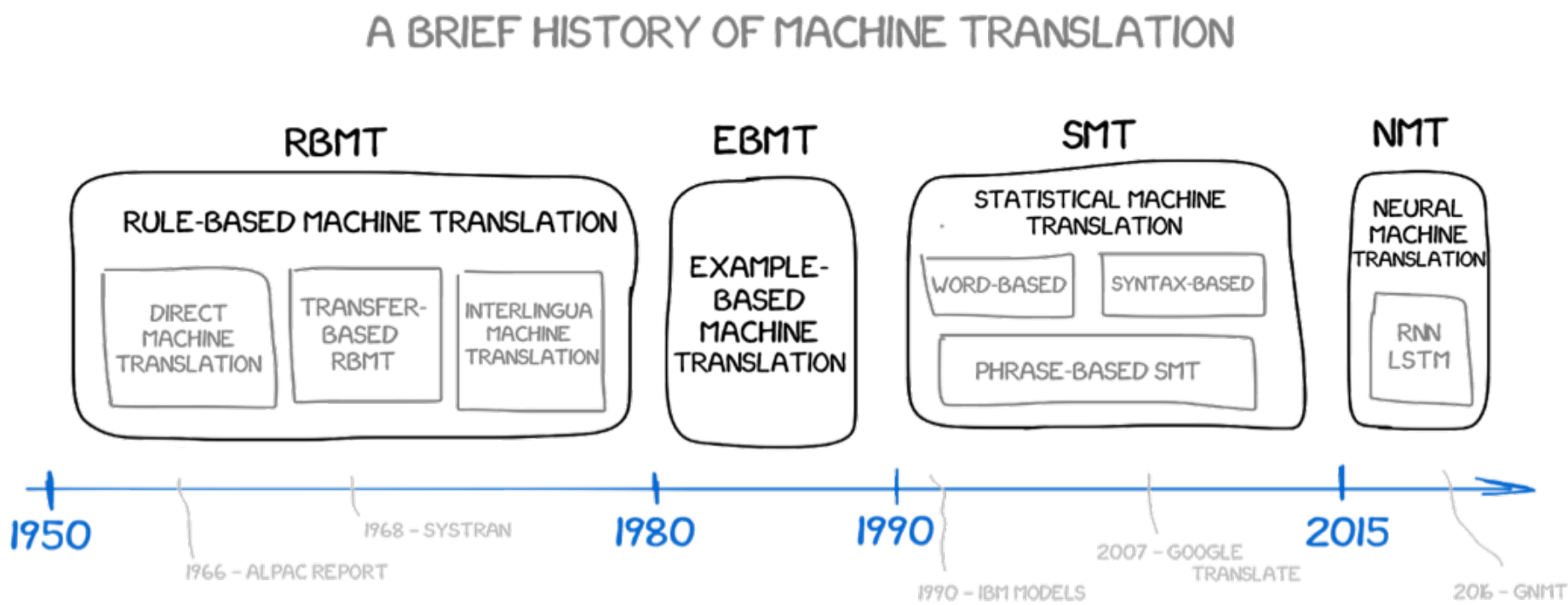


Haben Sie schon Ihren persönlichen Voice Assistant? (pidas.com)

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."

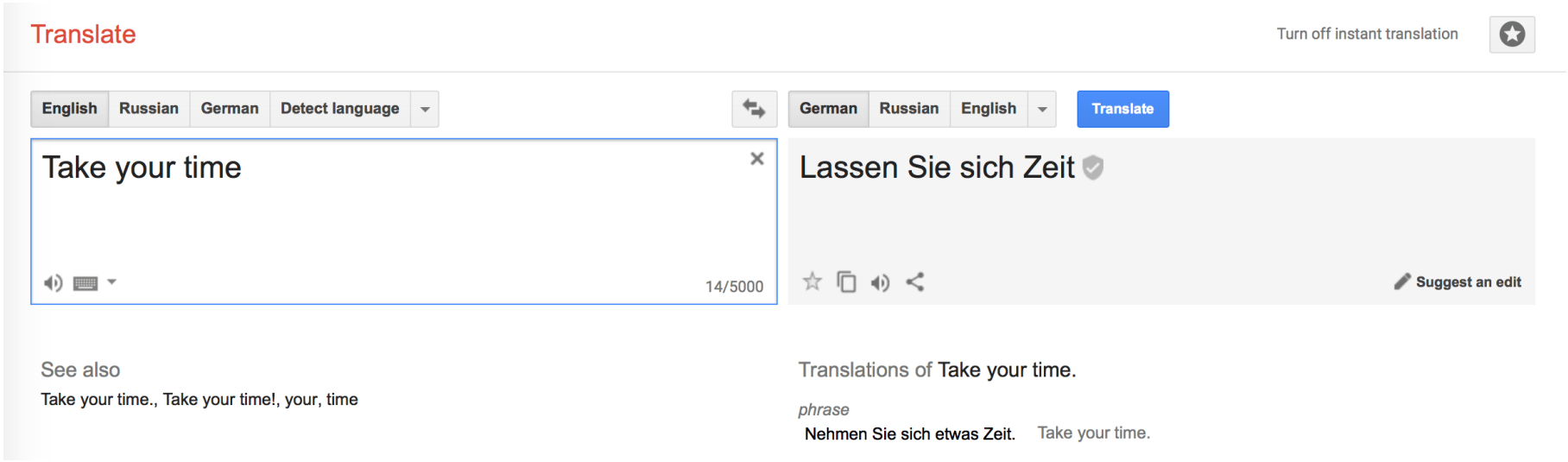


Machine Translation :: From the Cold War to Deep Learning :: vas3k.com

Application Examples

NLP – Language Translation / Machine Translation

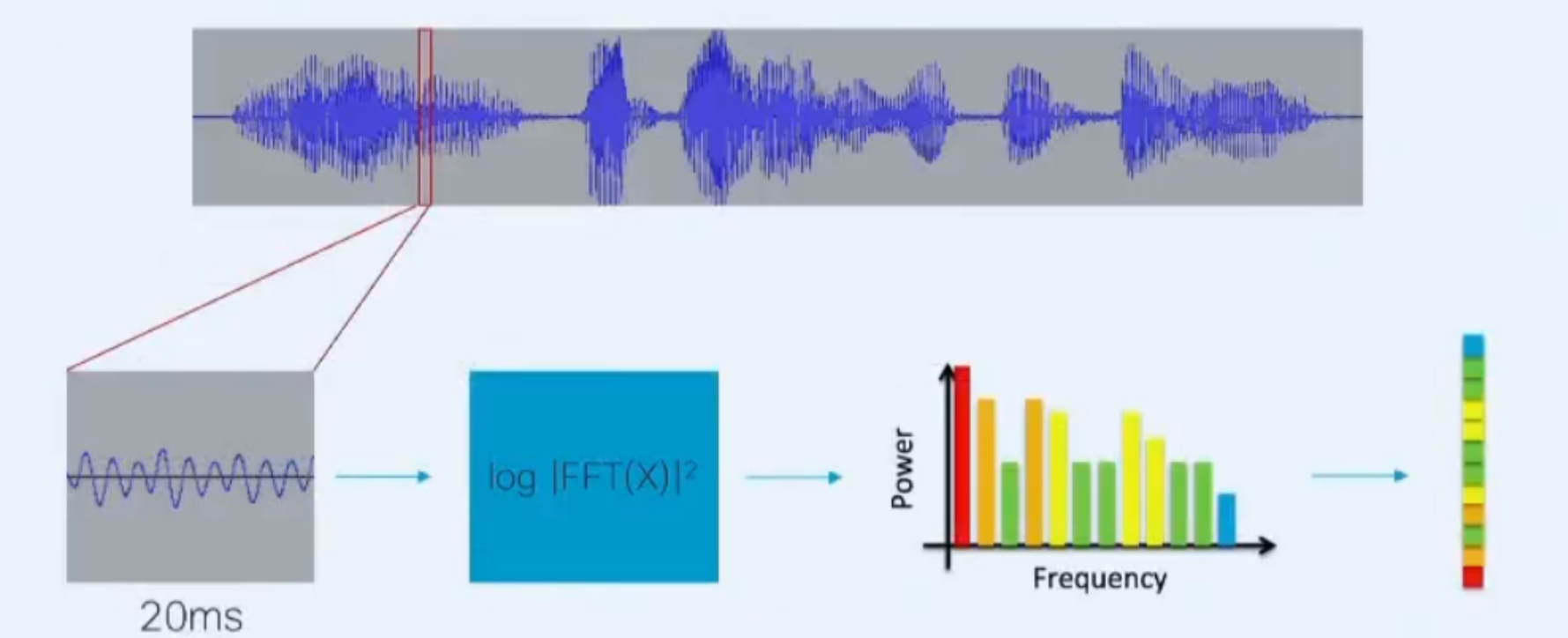
Google Translate



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

Application Examples
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

Entertainment – games & bots

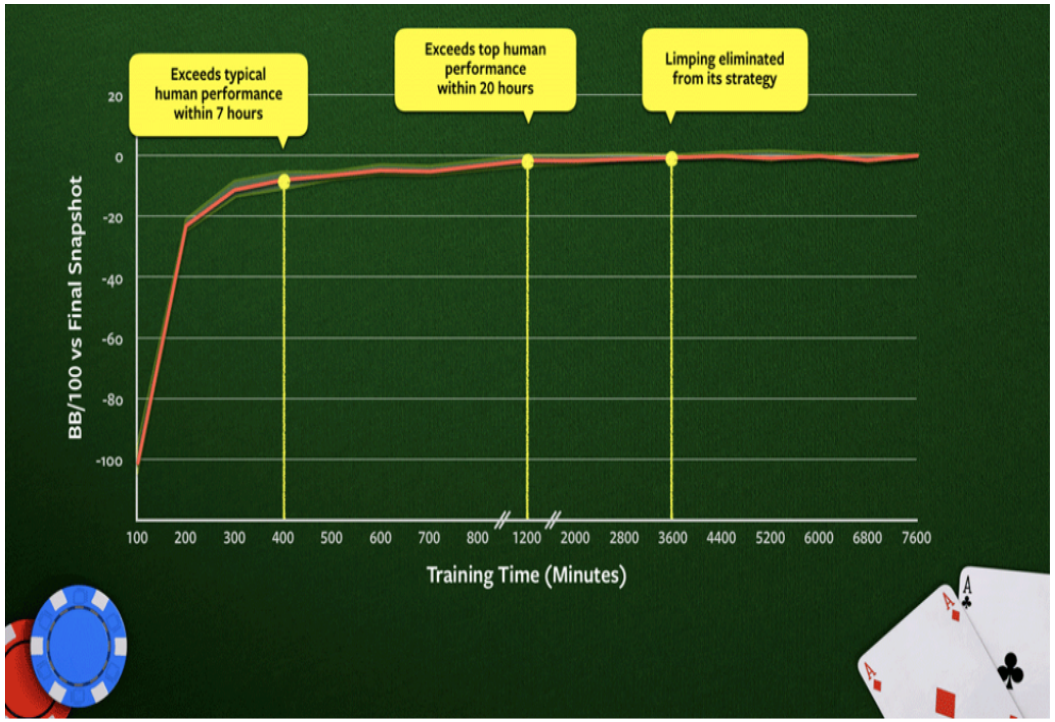
AlphaZero (deepmind) is a self-taught computer program for very high complexities. It learns with a complete game information approach solely based on game rules

Go (game) - Wikipedia

Application Examples

Entertainment – games & bots

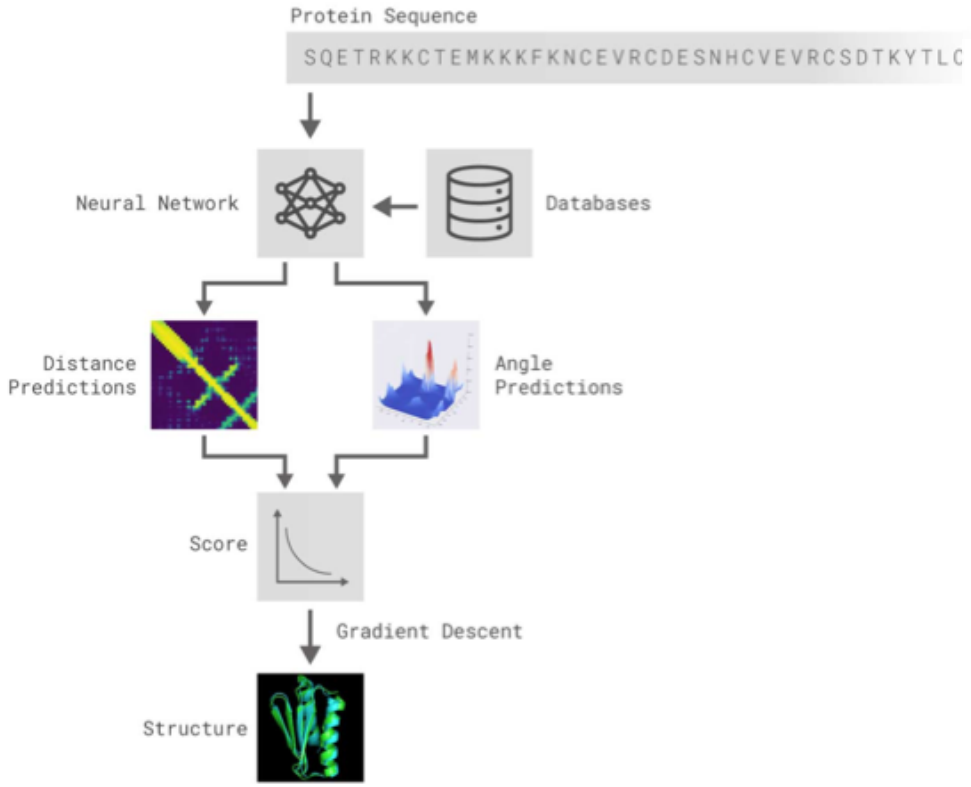
Pluribus (facebook) is an AI for multiplayer poker that beats 6 pros at once.
It reaches top performances after 20h training and learns with an incomplete game information approach.



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

Application Examples
Other Sciences

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



AlphaFold – Wikipedia

Machine learning background

Machine learning background

Artificial Intelligence

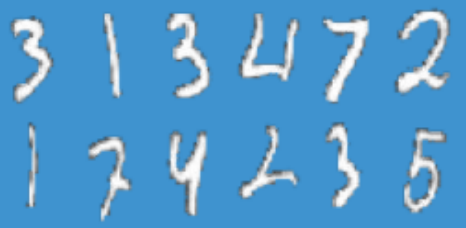
Any technique that enables computers to mimic human behavior

Machine Learning

Ability to learn without explicitly being programmed

Deep Learning

Extract patterns from data using neural networks

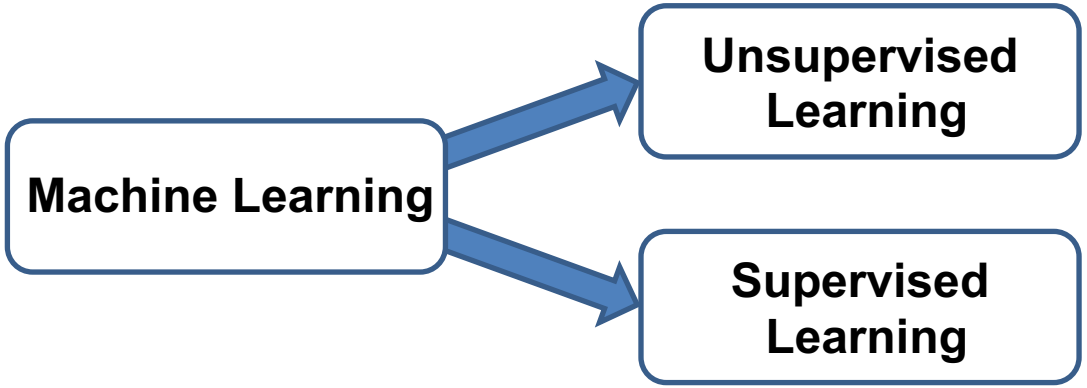


Machine learning background
Machine Learning Technics

Machine Learning

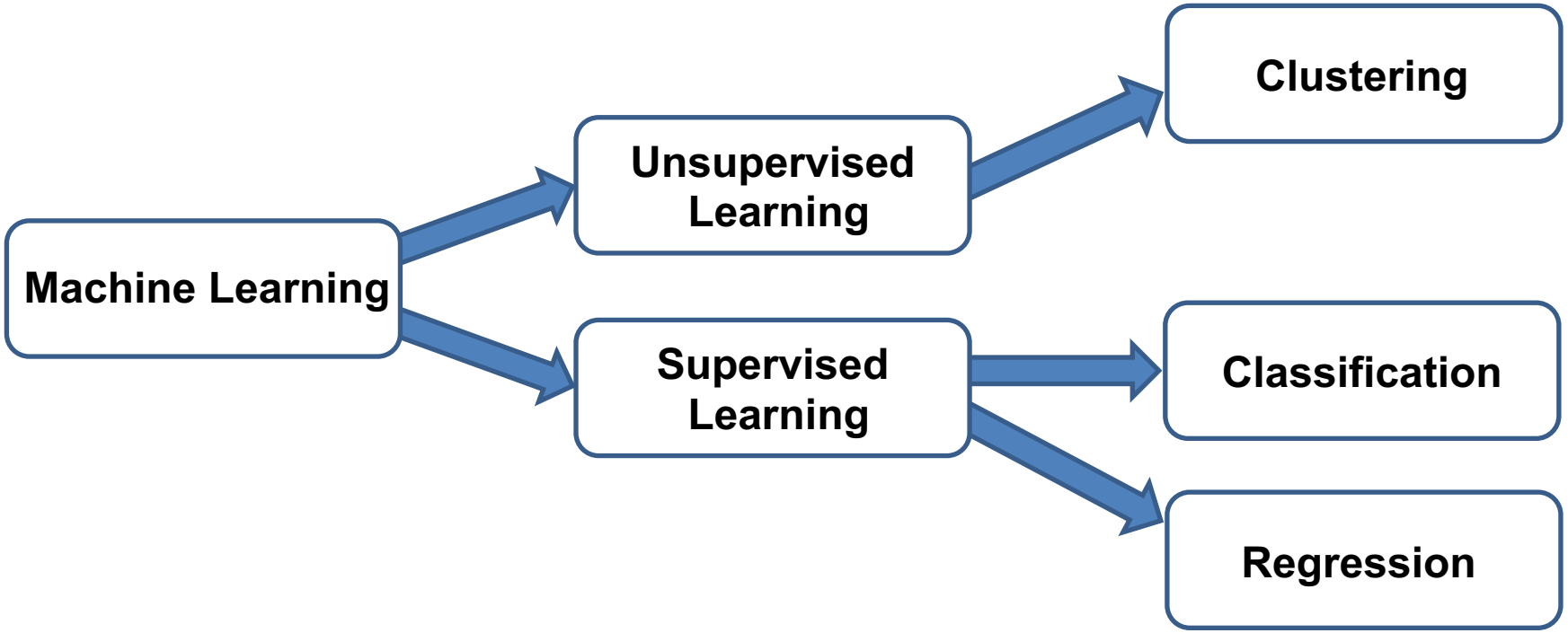
Machine learning background

Machine Learning Technics



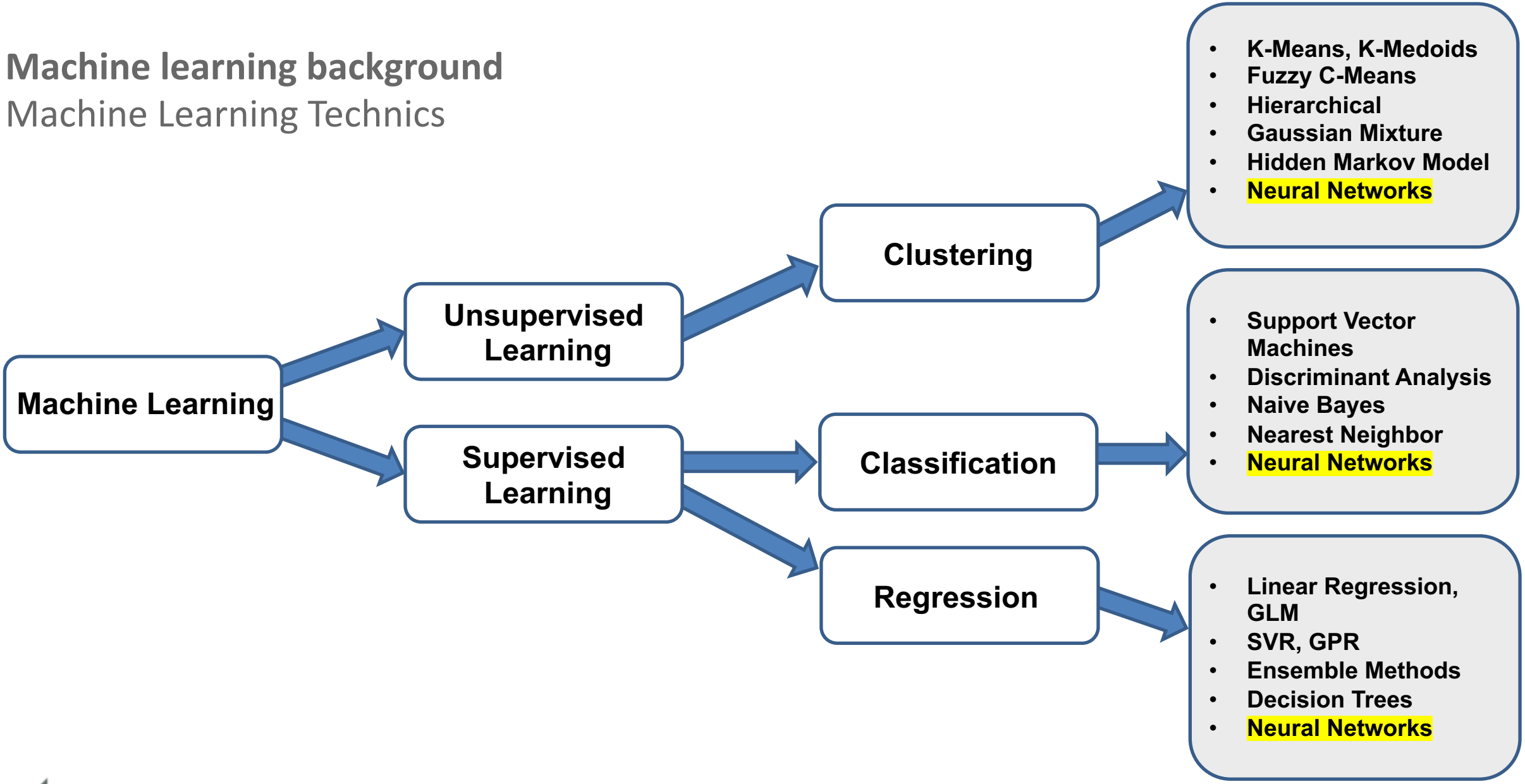
Machine learning background

Machine Learning Technics



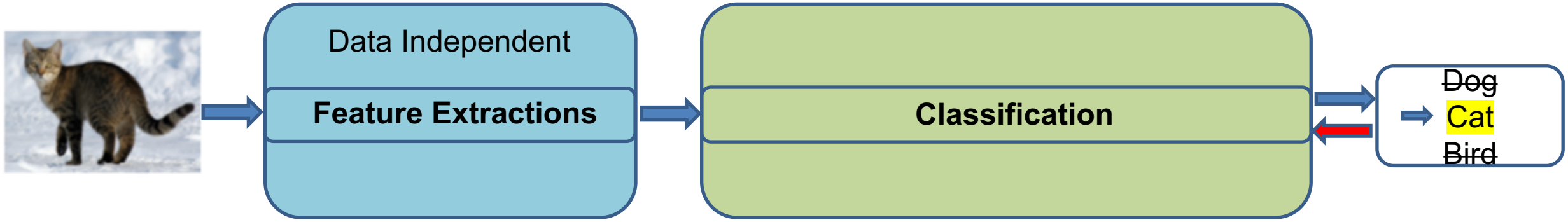
Machine learning background

Machine Learning Technics

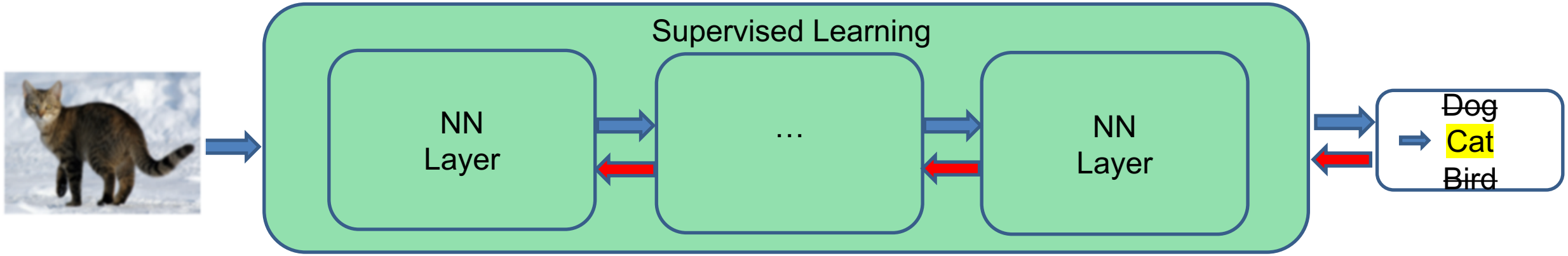


Machine learning background

“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

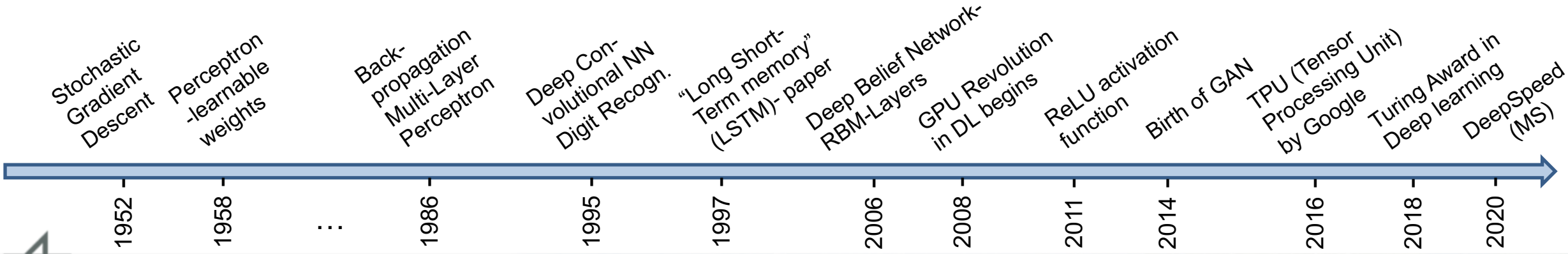
3. Software

+improved techniques from growing research

+optimization SW&HW

+new models and toolboxes

PyTorch TensorFlow



DLR

Neural network concepts

Neural network concepts

Building the network

- Input layer
- Hidden layers
  - Initialize weights and biases
- Output layer

Forward propagate input

- Calculating weighted sum from input
- Add non-linearity using activation function
- Make prediction from input

Evaluate performance

- Compute the error and its gradient

Back propagate

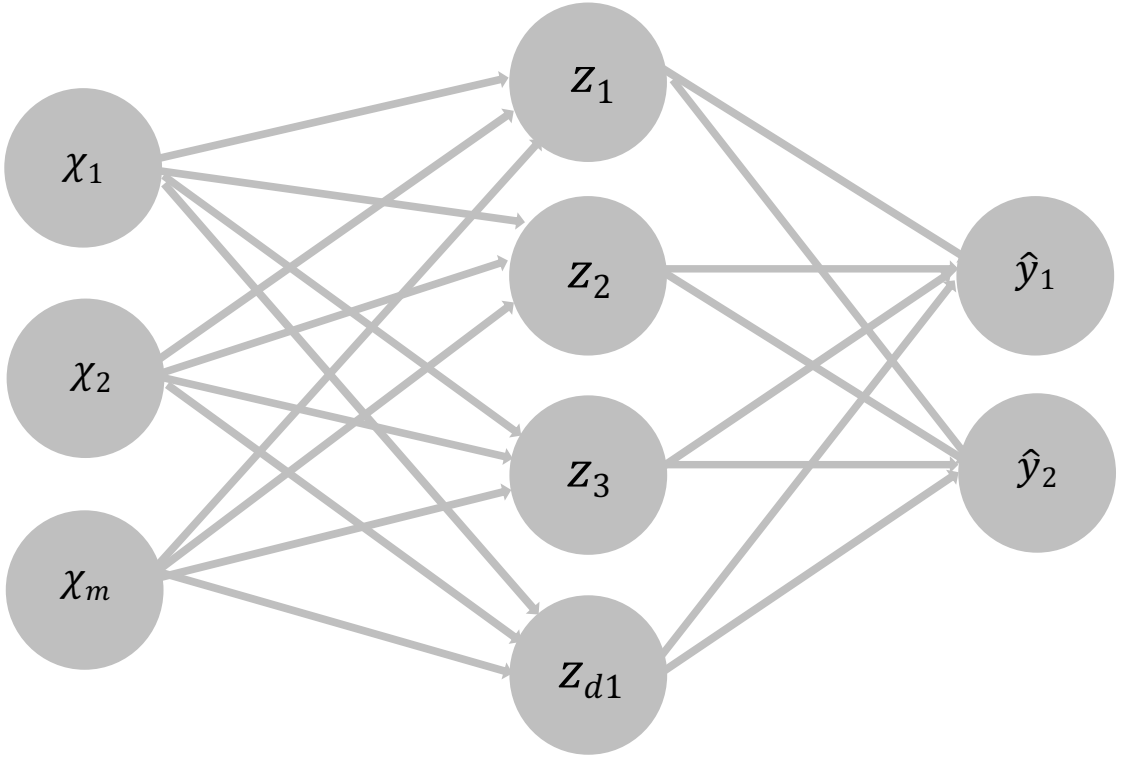
- Update weights according to the optimization algorithm

Improving the network

- Regularization,
- Data preprocessing,
- Fine-tuning parameters, etc.

Building the network

$$\begin{bmatrix} x_1 \\ x_2 \\ x_m \end{bmatrix}$$

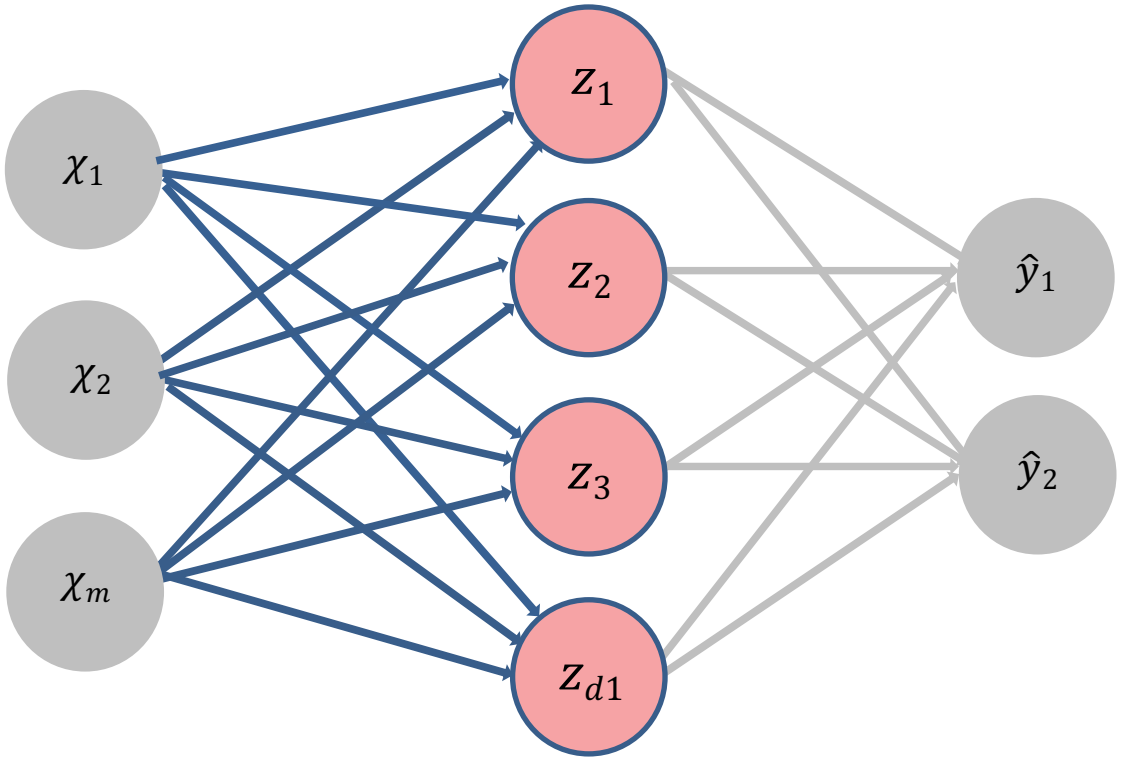


Forward Propagate Input

CAT

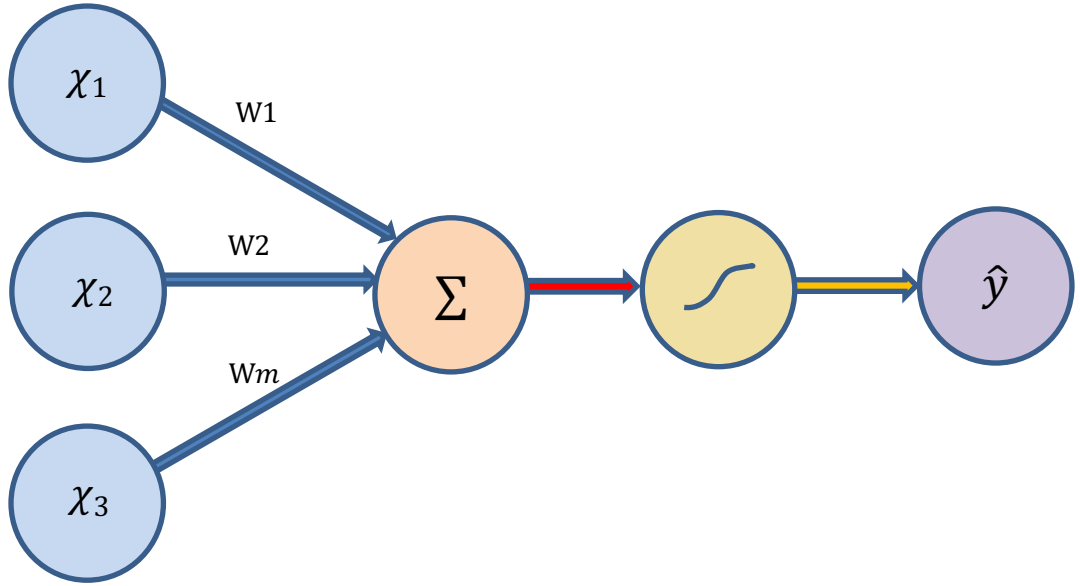
$$\begin{bmatrix} \chi_1 \\ \chi_2 \\ \chi_m \end{bmatrix}$$

Calculating weighted sum
Add non-linearity
Make prediction



Neural network concepts

A single neuron (perceptron)



Inputs Weights Sum Non-Linearity Output

Output

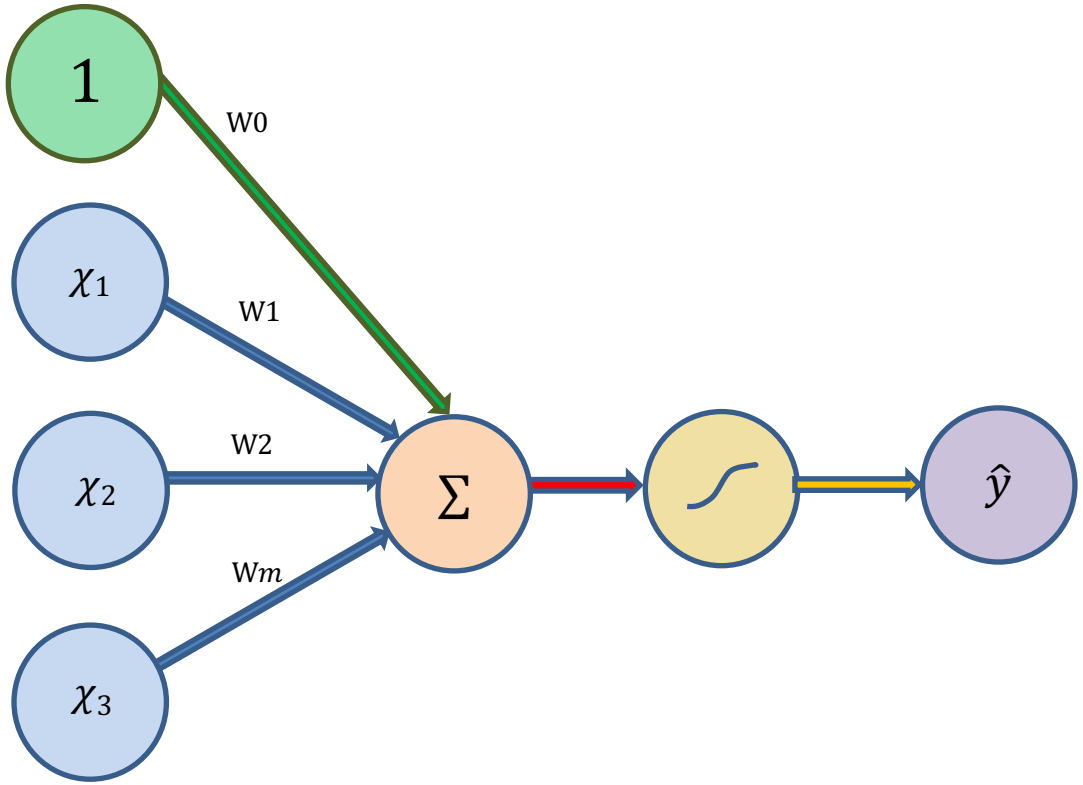
Linear combination of inputs

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

Non-linear activation function

Neural network concepts

A single neuron (perceptron)



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

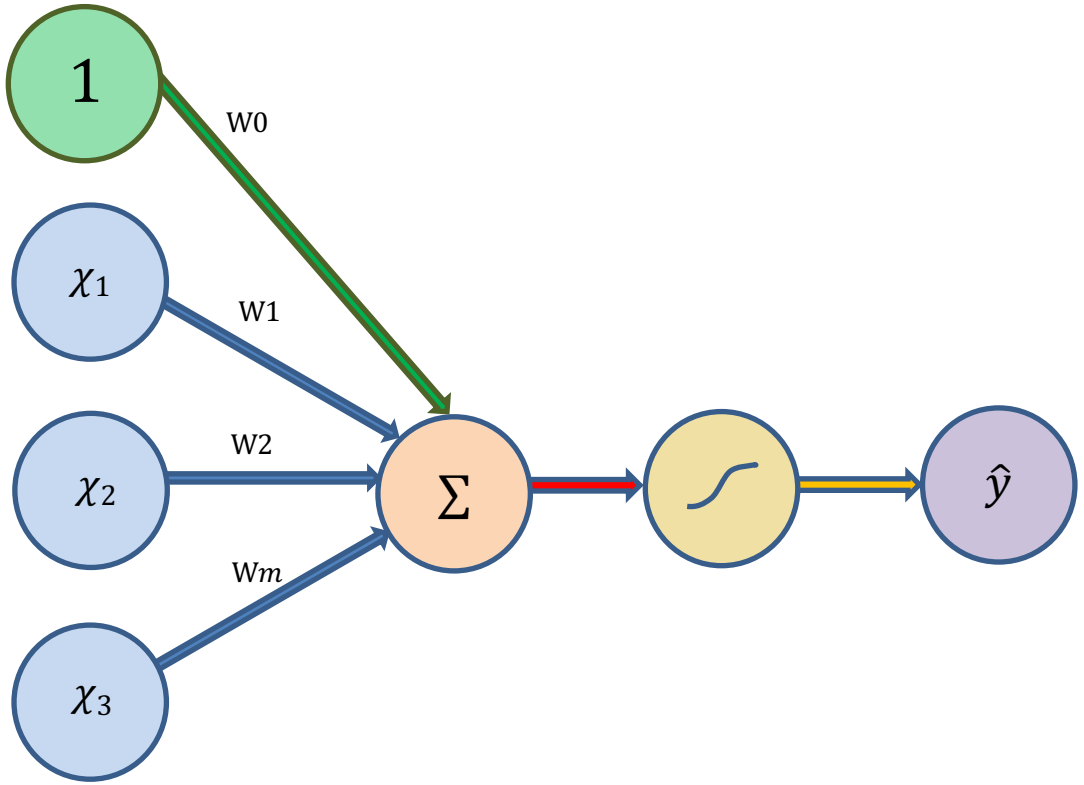
Bias

Inputs Weights Sum Non-Linearity Output

Neural network concepts

A single neuron (perceptron)

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



Inputs Weights Sum Non-Linearity Output

Vector/ Matrix operations

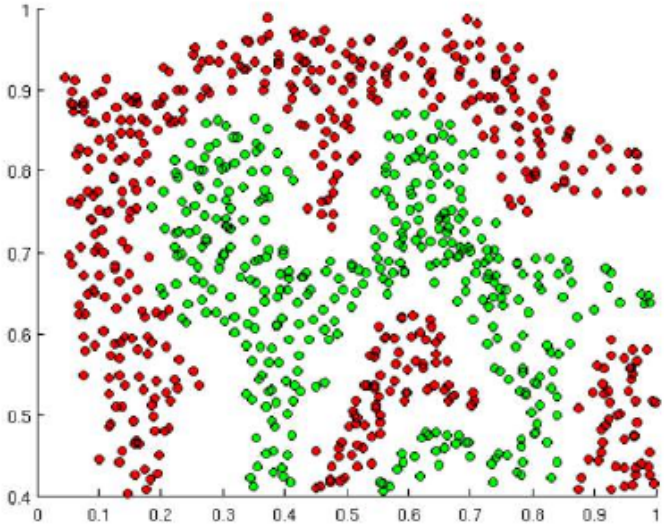
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

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

Neural network concepts

Non-linear Activation functions

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

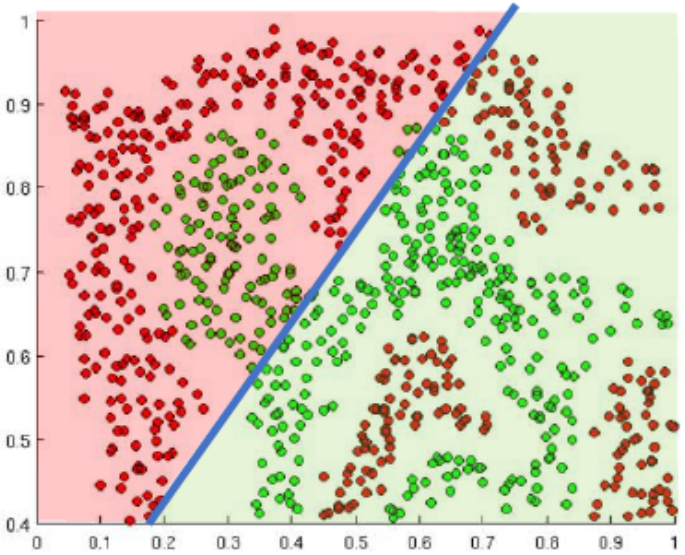


What if we wanted to build a Neural Network to distinguish green vs red points?

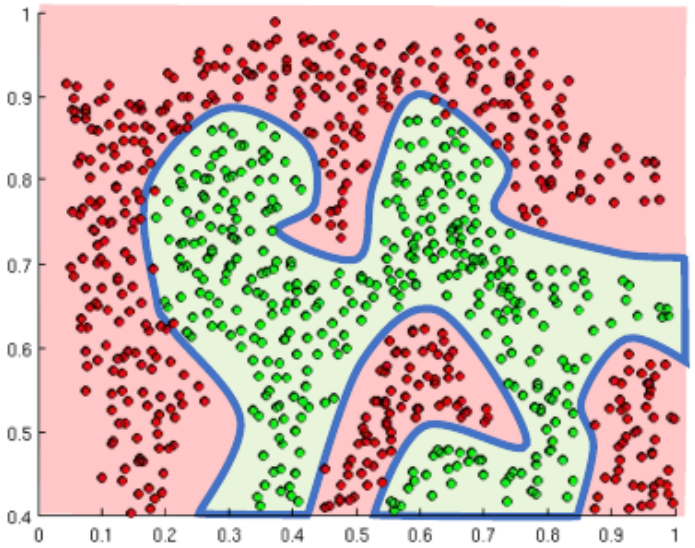
Neural network concepts

Non-linear 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

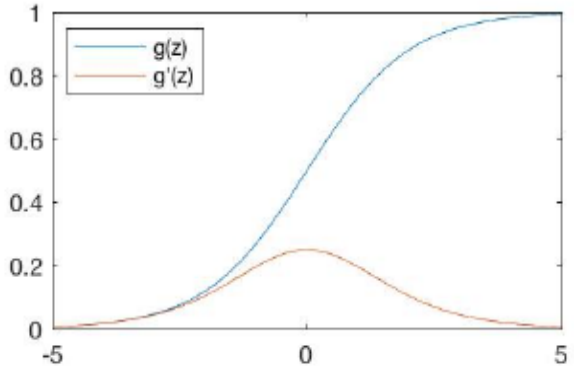


Non-linearities allow us to approximate arbitrarily complex functions

Neural network concepts

Non-linear Activation functions

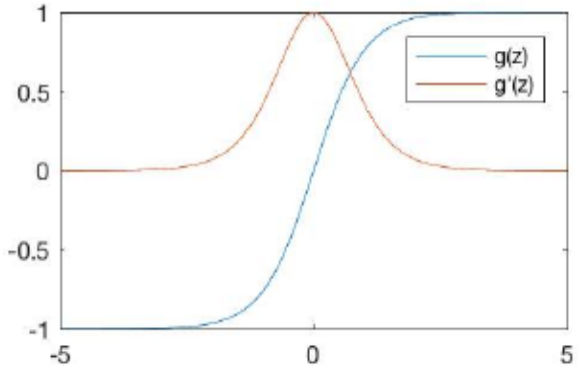
Sigmoid Function



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

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

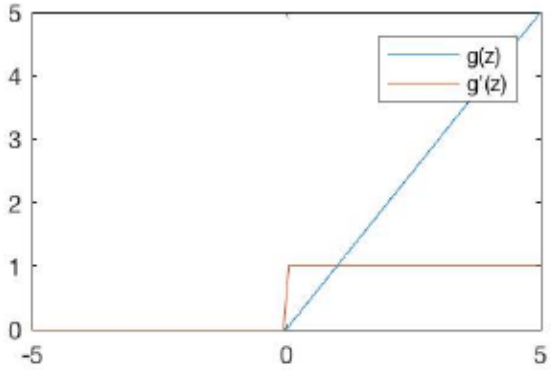
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)

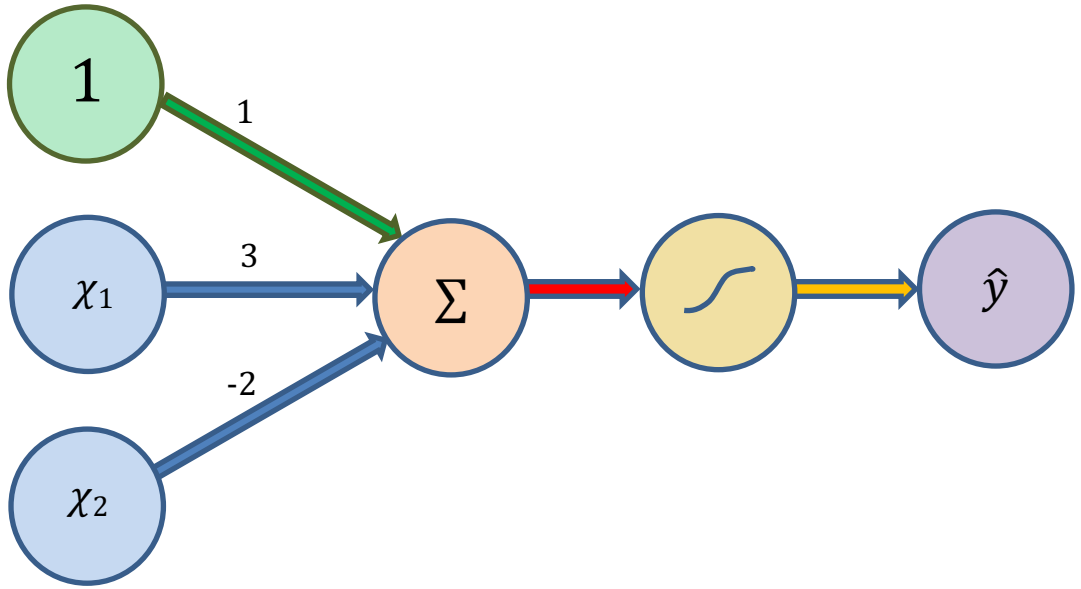


$$g(z) = \max(0, z)$$

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

Neural network concepts

A single neuron (perceptron) - Example



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

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

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

$w_0 = 1$

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

This is just a line in 2D

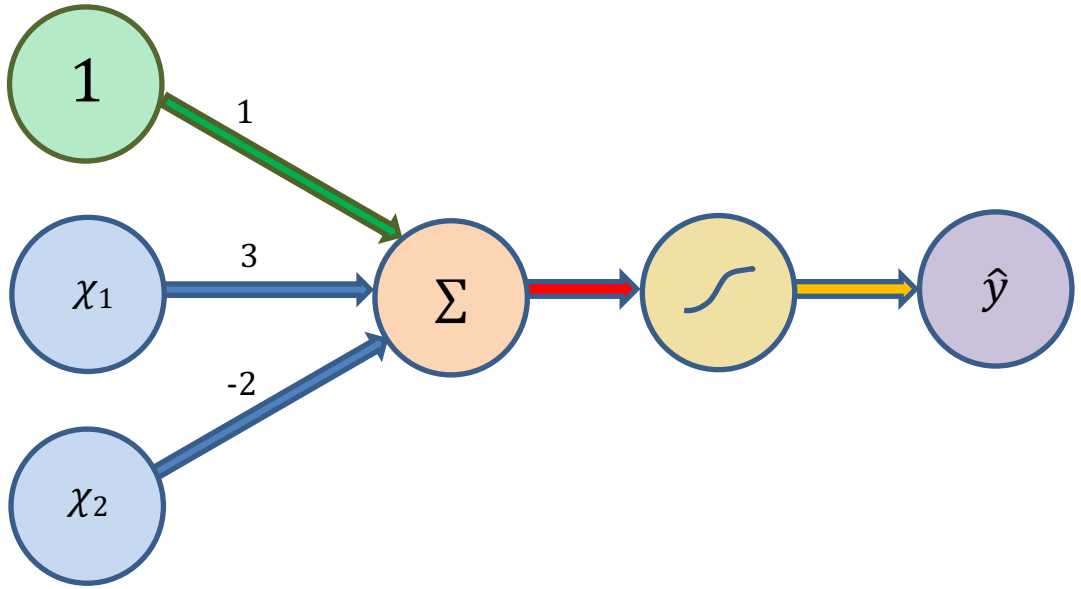
Neural network concepts

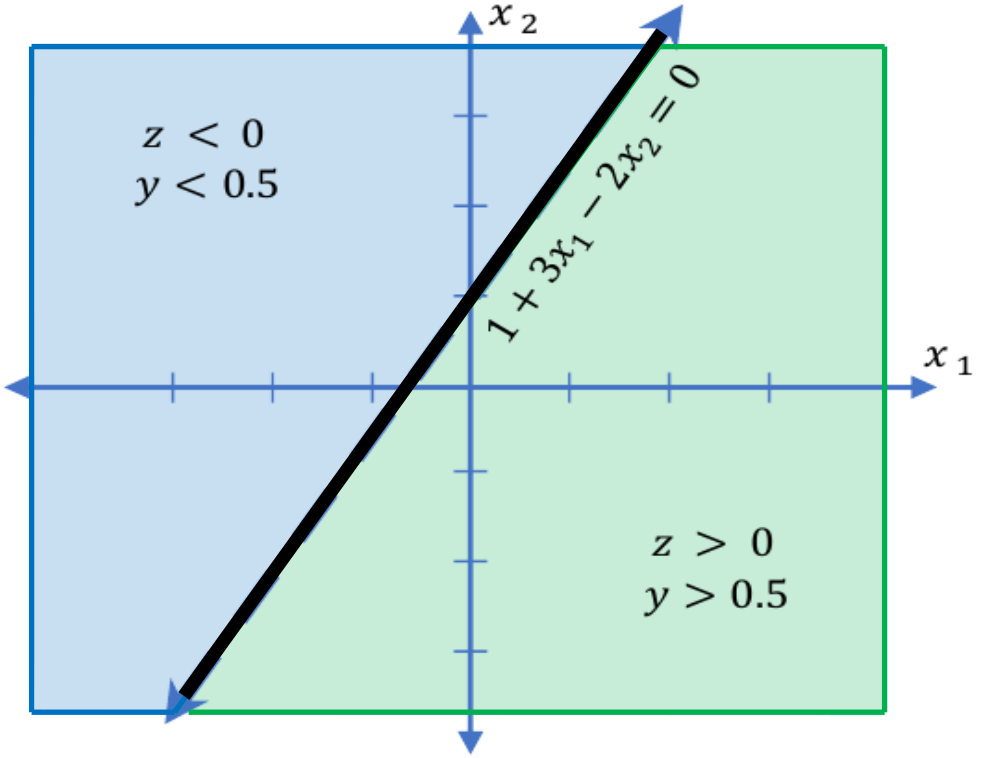
A single neuron (perceptron) - Example

$w_0 = 1$

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

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

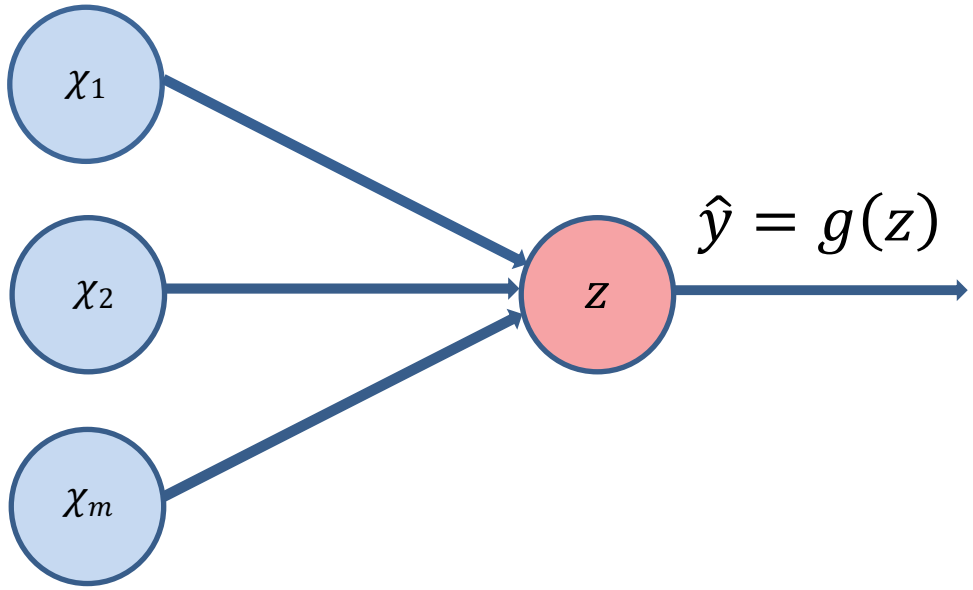




Neural network concepts

Simplified neuron

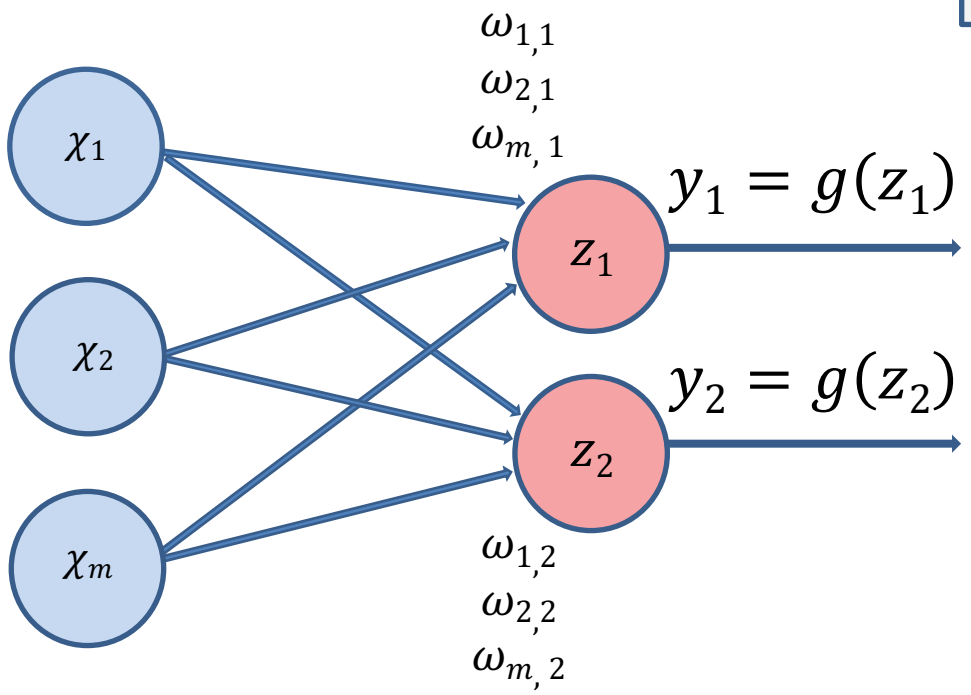
$$z = \left(w_0 + \sum_{j=1}^{m} x_j \, w_j \right)$$



Neural network concepts

Multi-output neuron

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

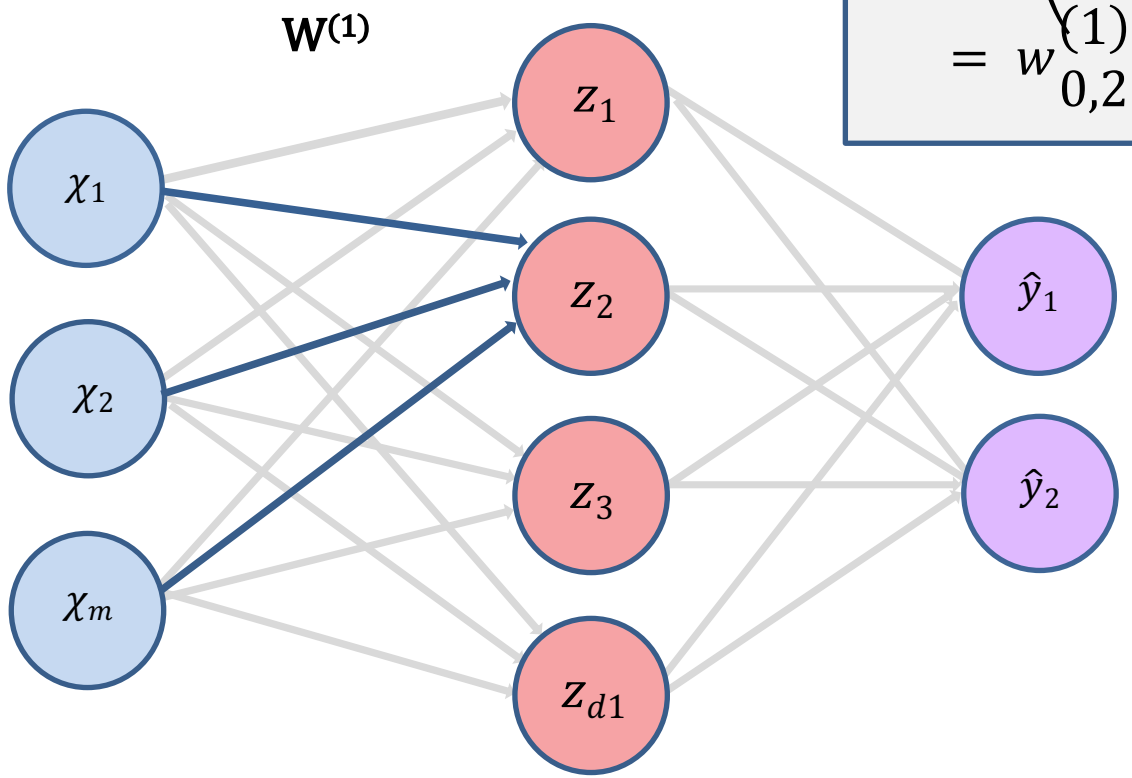


Neural network concepts

A single-layer neural network

$$z_2 = \left(w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \left(w_{j,2}^{(1)} \right) \right)$$

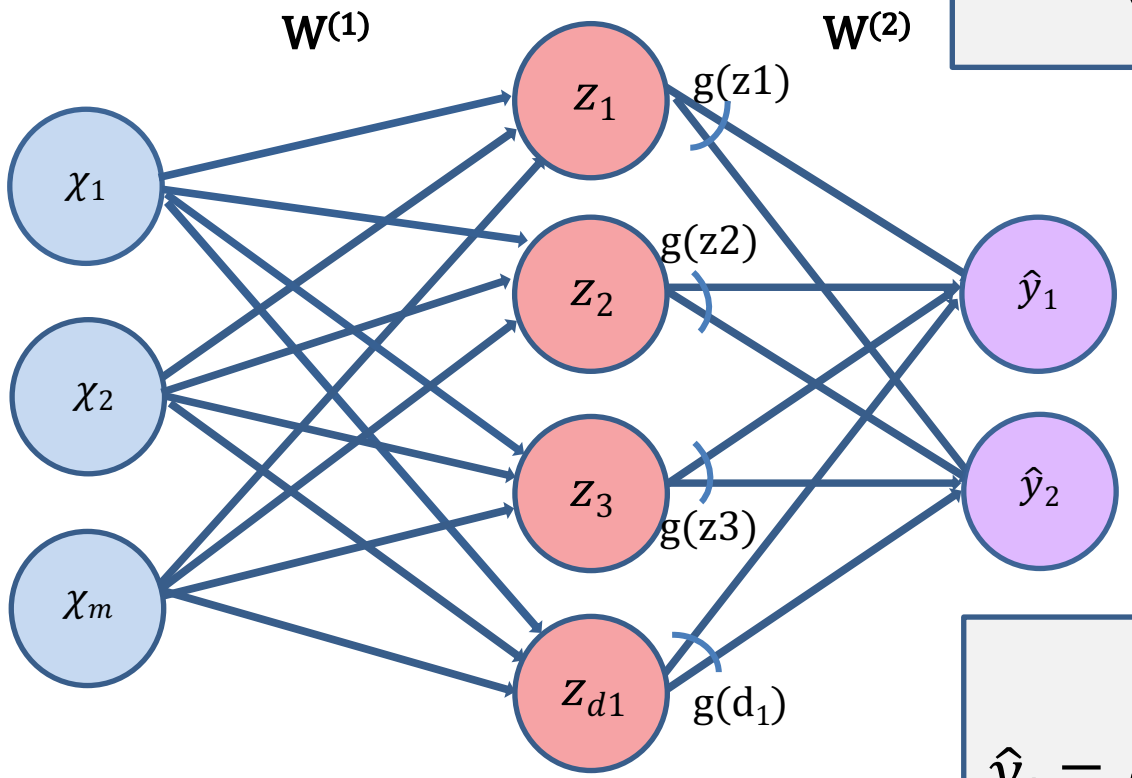
$$= w_{0,2}^{(1)} + \mathrm{x}_1 w_{1,2}^{(1)} + \mathrm{x}_2 w_{2,2}^{(1)} + \mathrm{x}_m w_{m,2}^{(1)}$$



Neural network concepts

A single-layer neural network

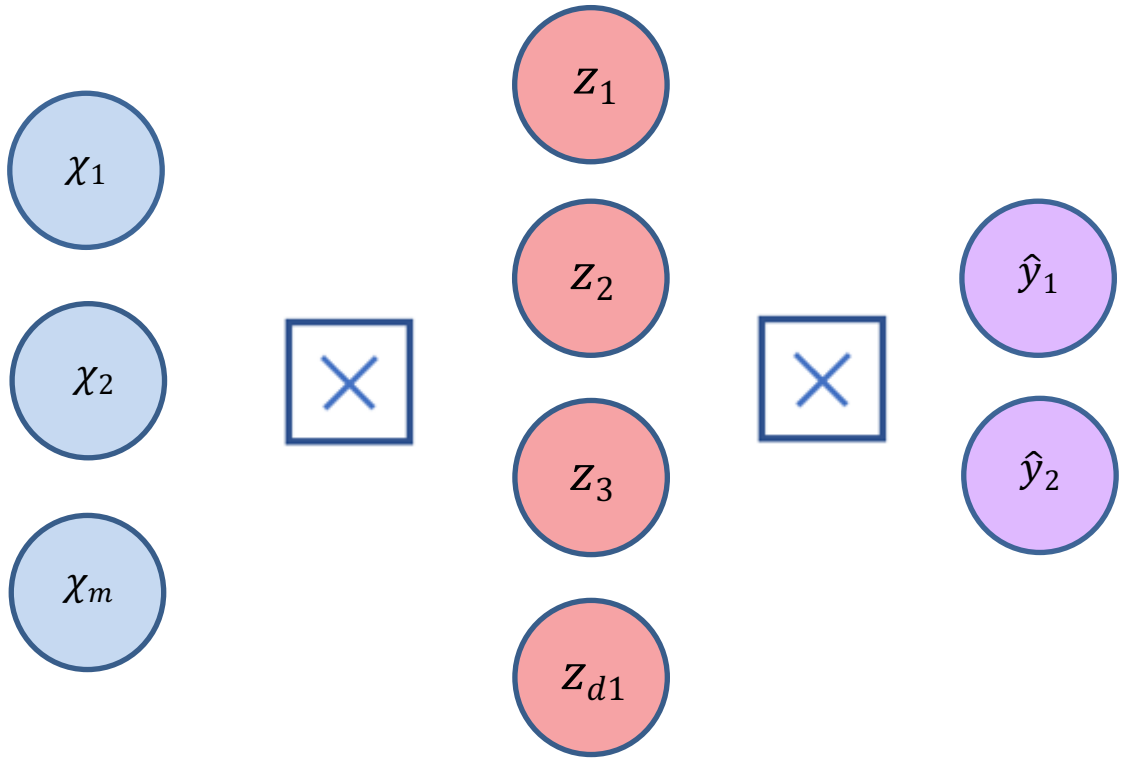
$$z_i = \left(w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \right)$$



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

Neural network concepts

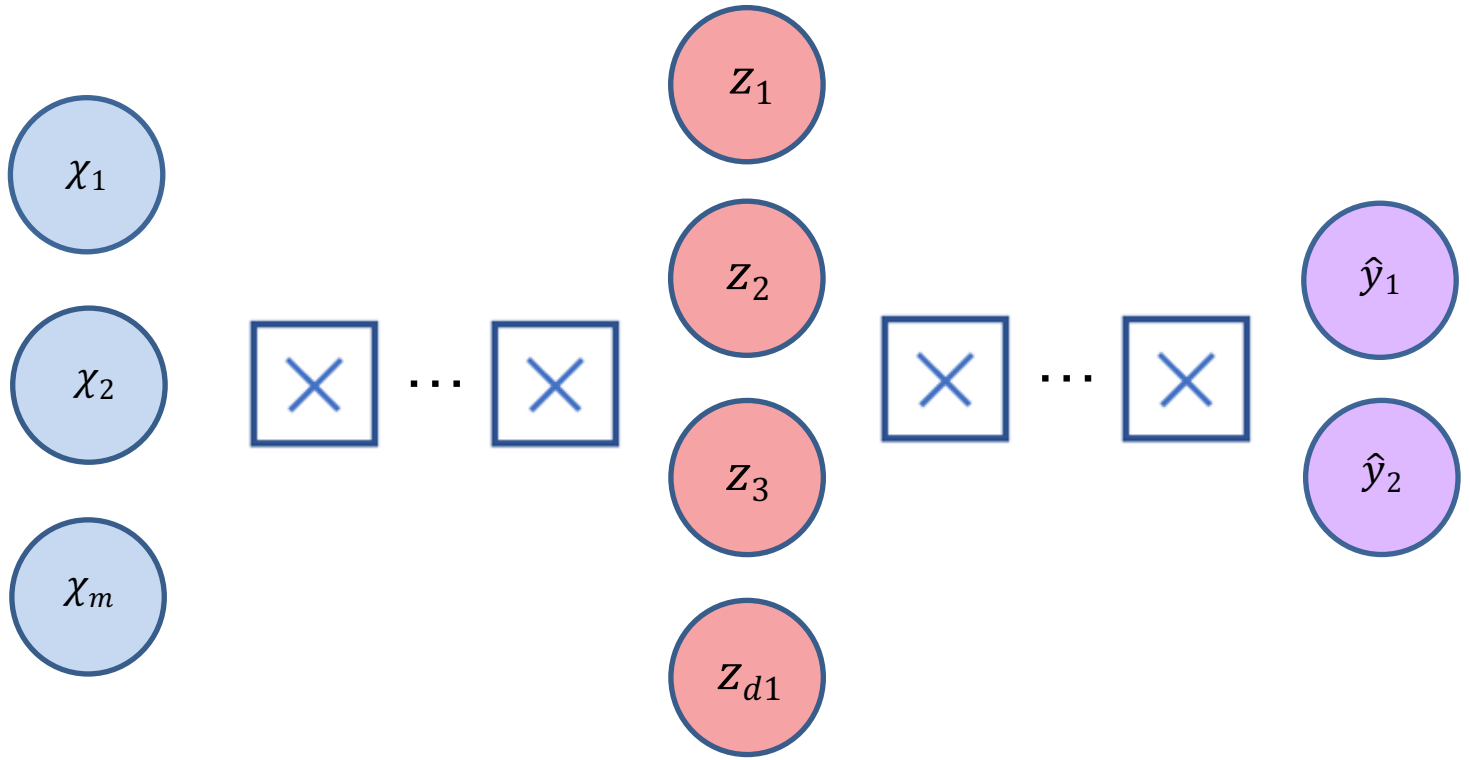
A single-layer neural network



Neural network concepts

Deep neural network

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

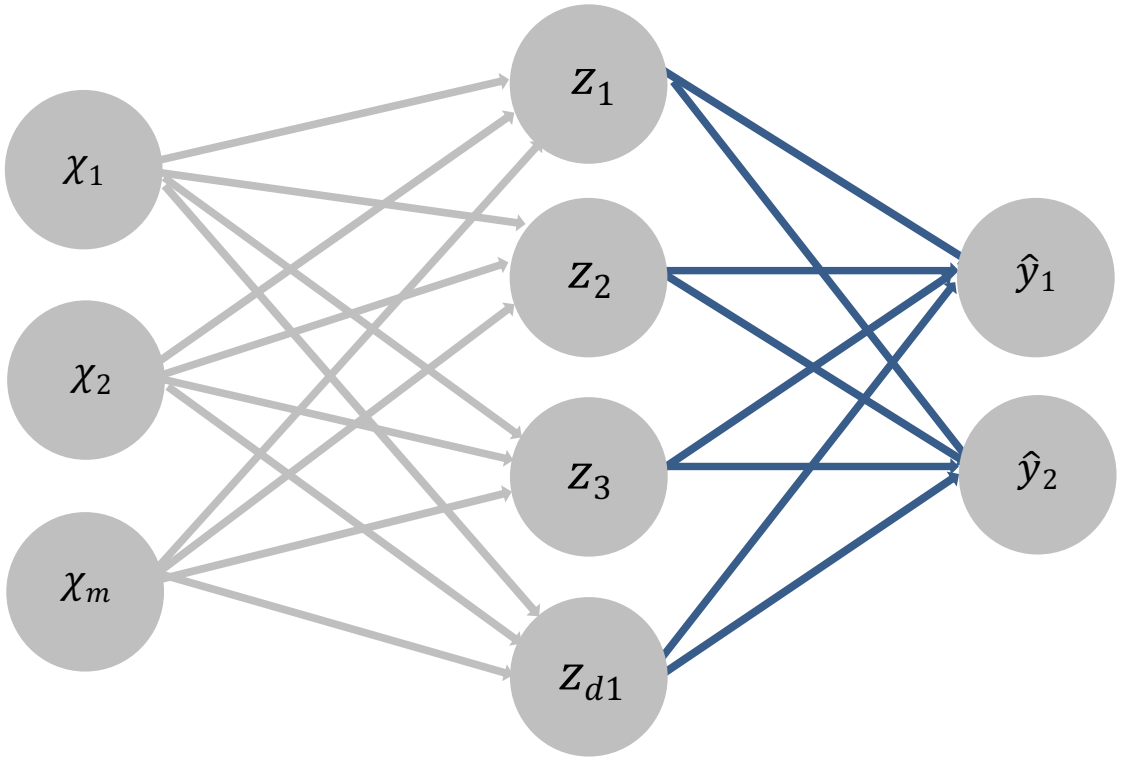


Forward Propagate Input

CAT

$$\begin{bmatrix} \chi_1 \\ \chi_2 \\ \chi_m \end{bmatrix}$$

Calculating weighted sum
Add non-linearity
Make prediction



Forward Propagate Input

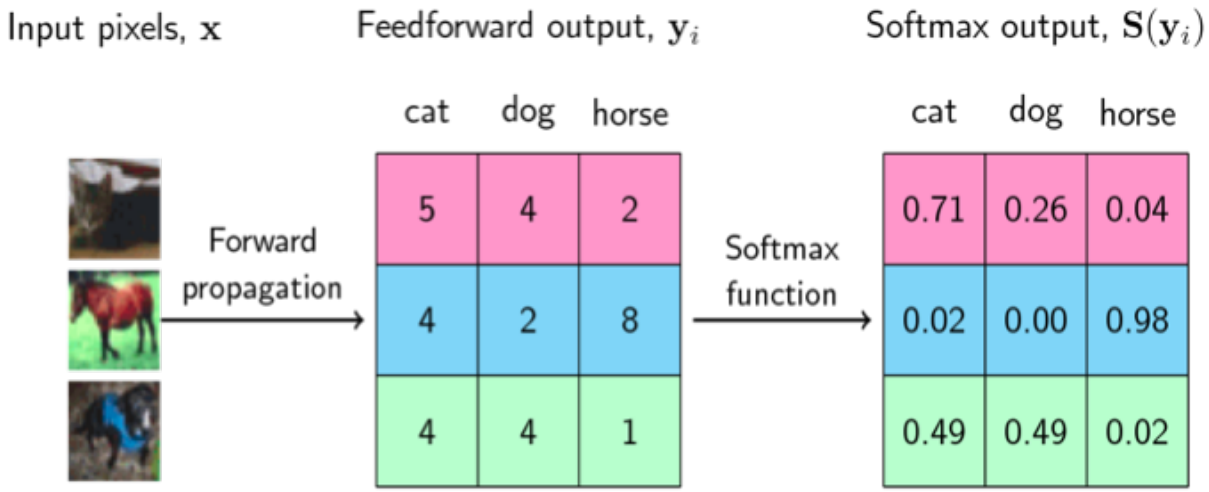
Softmax activation

(= softargmax = normalized exponential function)

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

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

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



Using own continually example ?

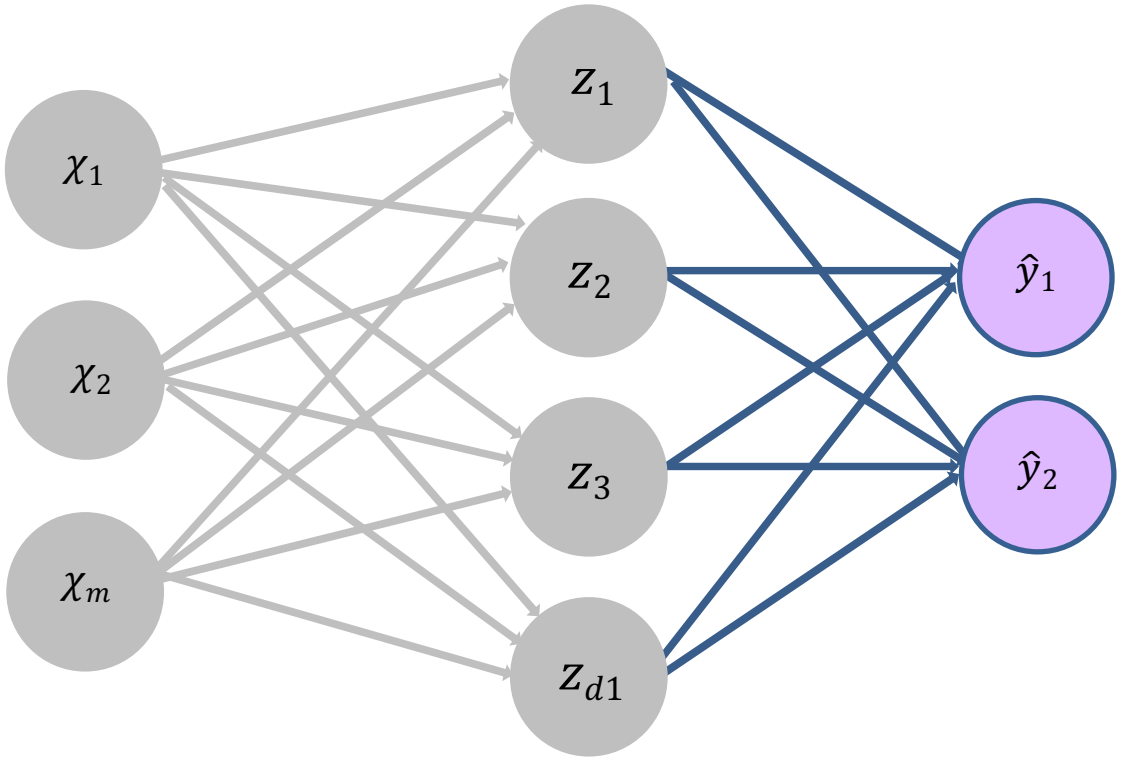
https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

Evaluate Prediction

CAT

$$\begin{bmatrix} \chi_1 \\ \chi_2 \\ \chi_m \end{bmatrix}$$

Compute the error and its gradient

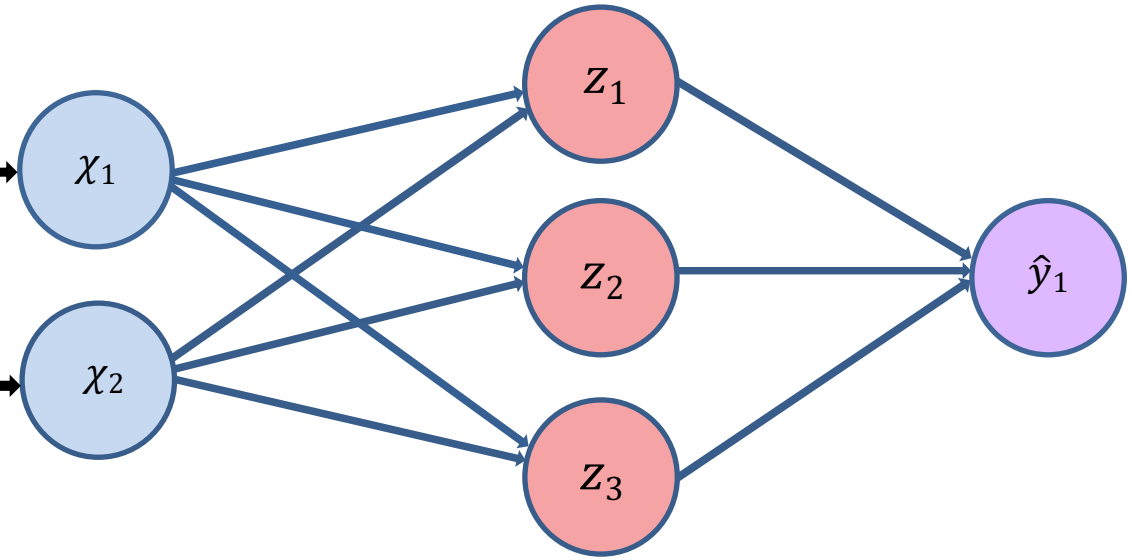


Evaluate Prediction

Loss function

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

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



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

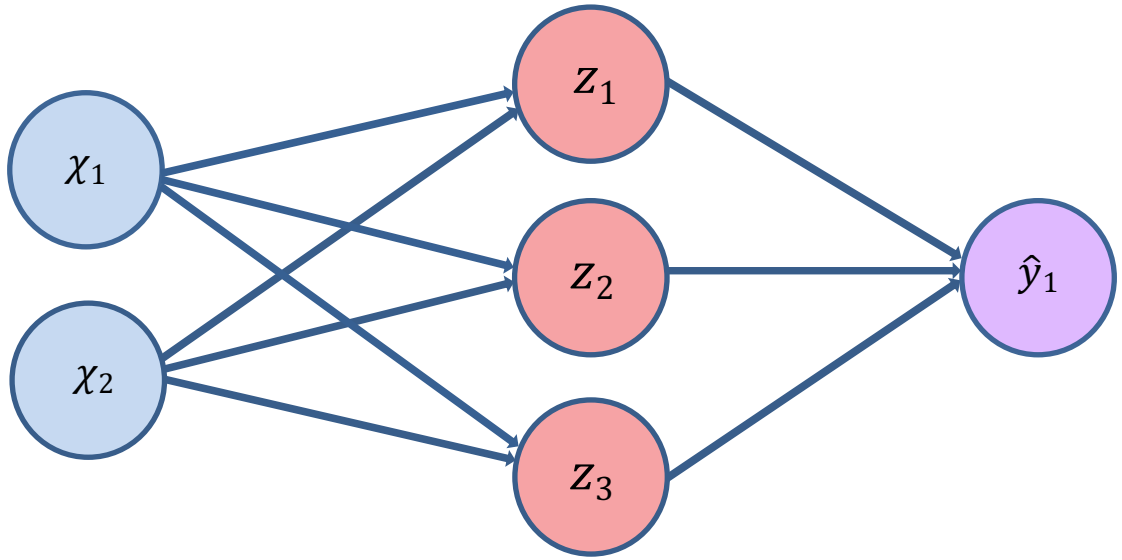
$$\mathcal{L}(f(\underbrace{x^{(i)}; \boldsymbol{W}}_{\text{Predicted}}), \underbrace{y^{(i)}}_{\text{Actual}})$$

Evaluate Prediction

Loss function

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

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



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

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

Also known as:

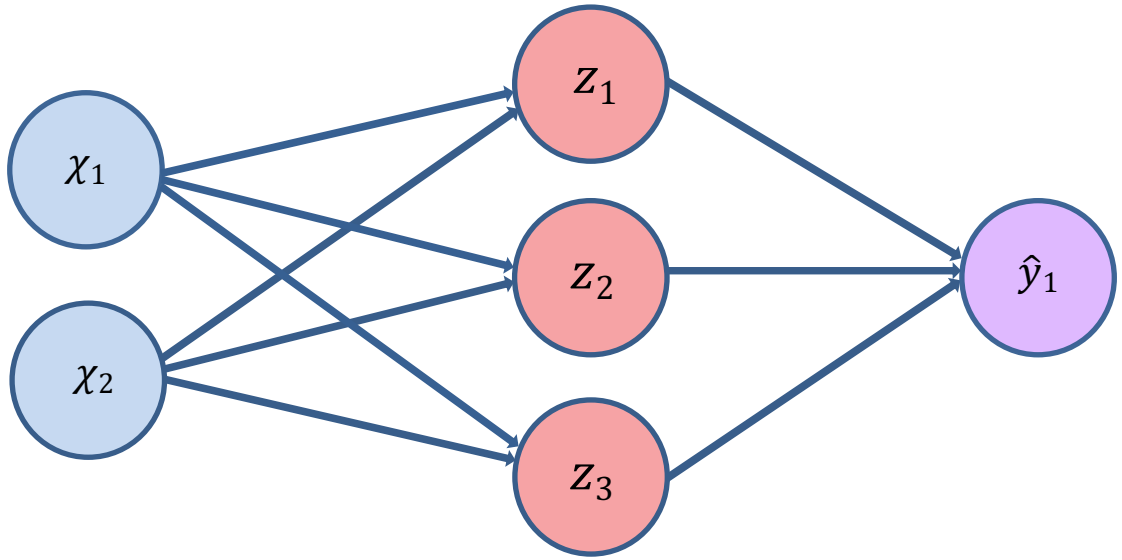
- Objective function
- Cost function
- Empirical Risk

Evaluate Prediction

Binary cross-entropy loss

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

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



$$\begin{matrix} f(x) & y \end{matrix}$$

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

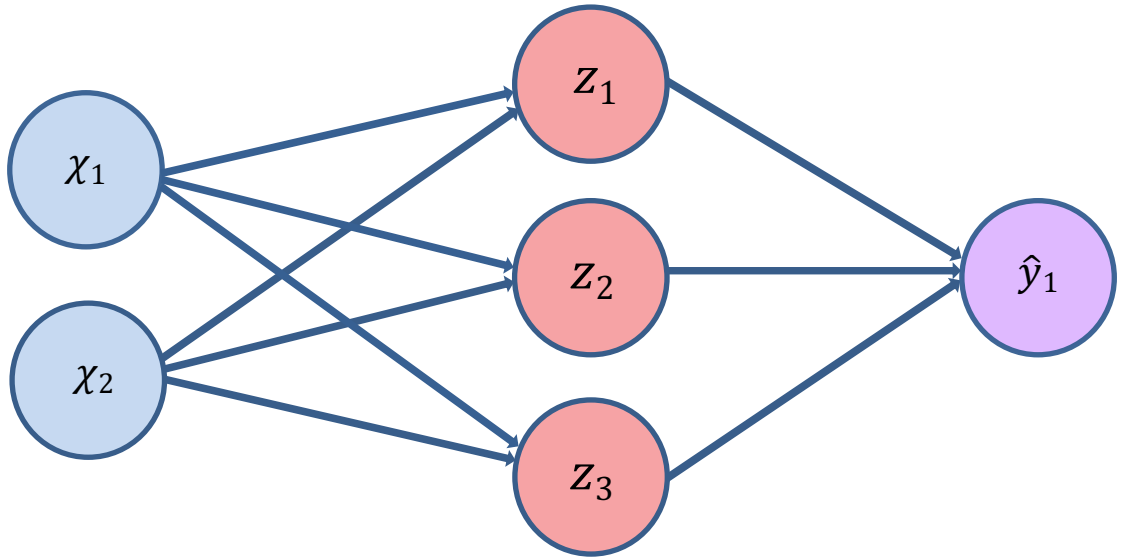
$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log(\underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}}) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log(1 - \underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}})$$

Evaluate Prediction

Mean squared error loss

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

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



$$\begin{matrix} f(x) & y \end{matrix}$$

$$\begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix}$$

Final Grades (percentage)

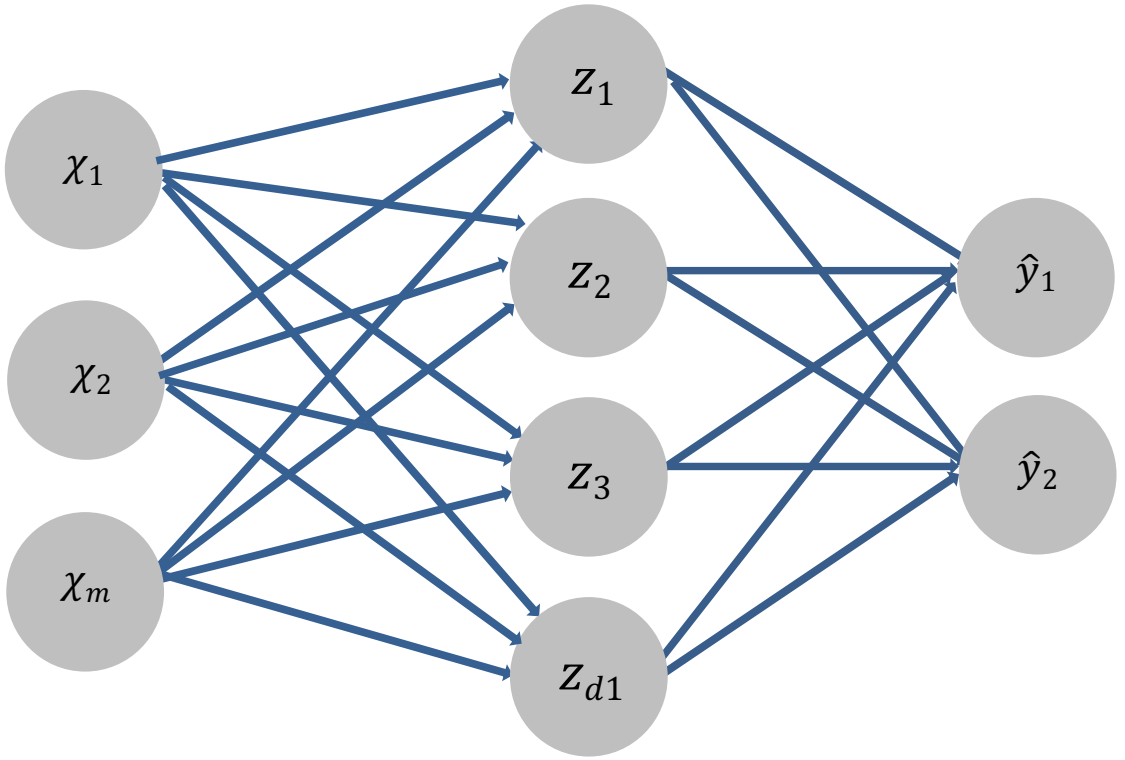
$$\boldsymbol{J(W)} = \frac{1}{n}\sum_{i=1}^{n} (\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}})^{\mathbf{2}}$$

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**

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

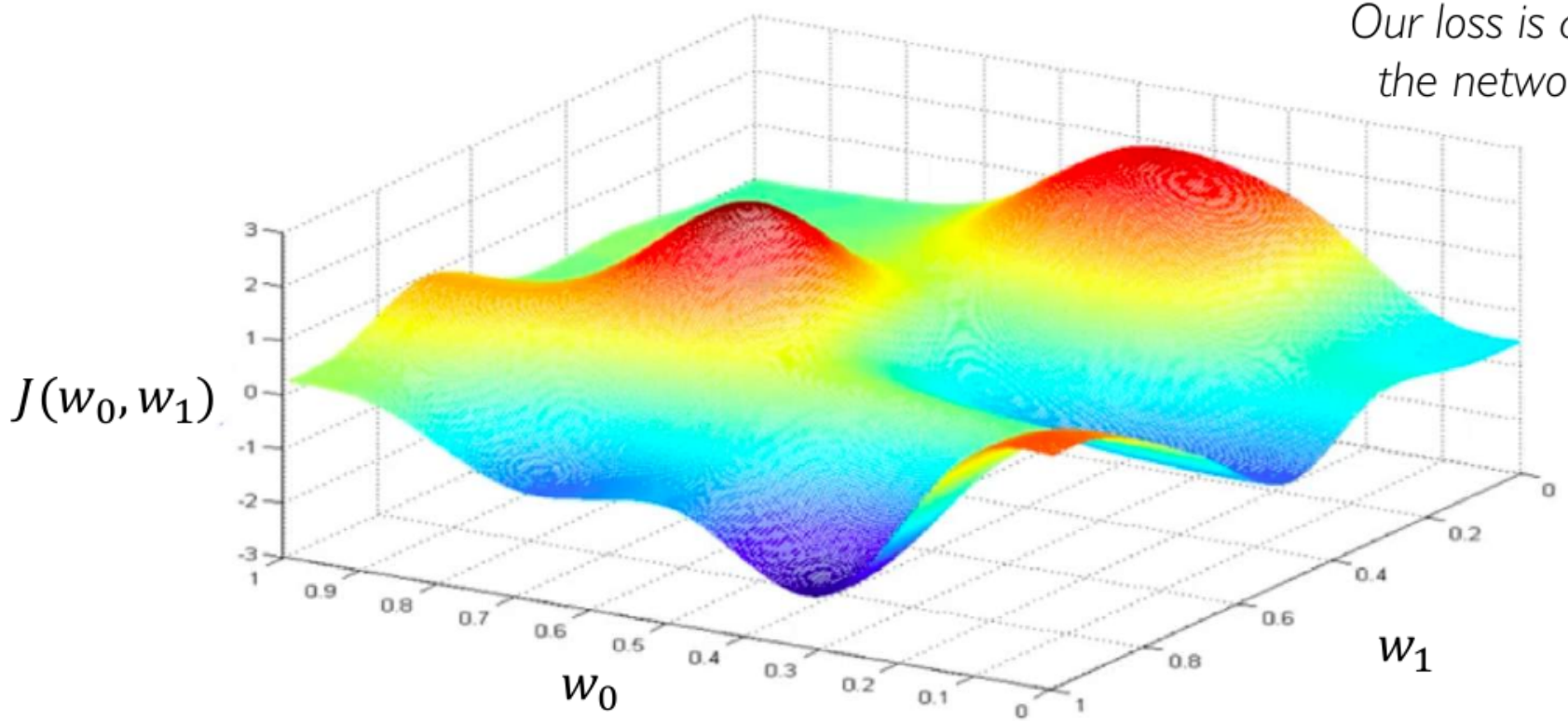
$$\boldsymbol{w_0} = 1$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{argmin}\, \boldsymbol{J}(\mathbf{W}) \quad \Big| \quad \mathbf{W} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots\}$$

Back Propagate
Loss optimization

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

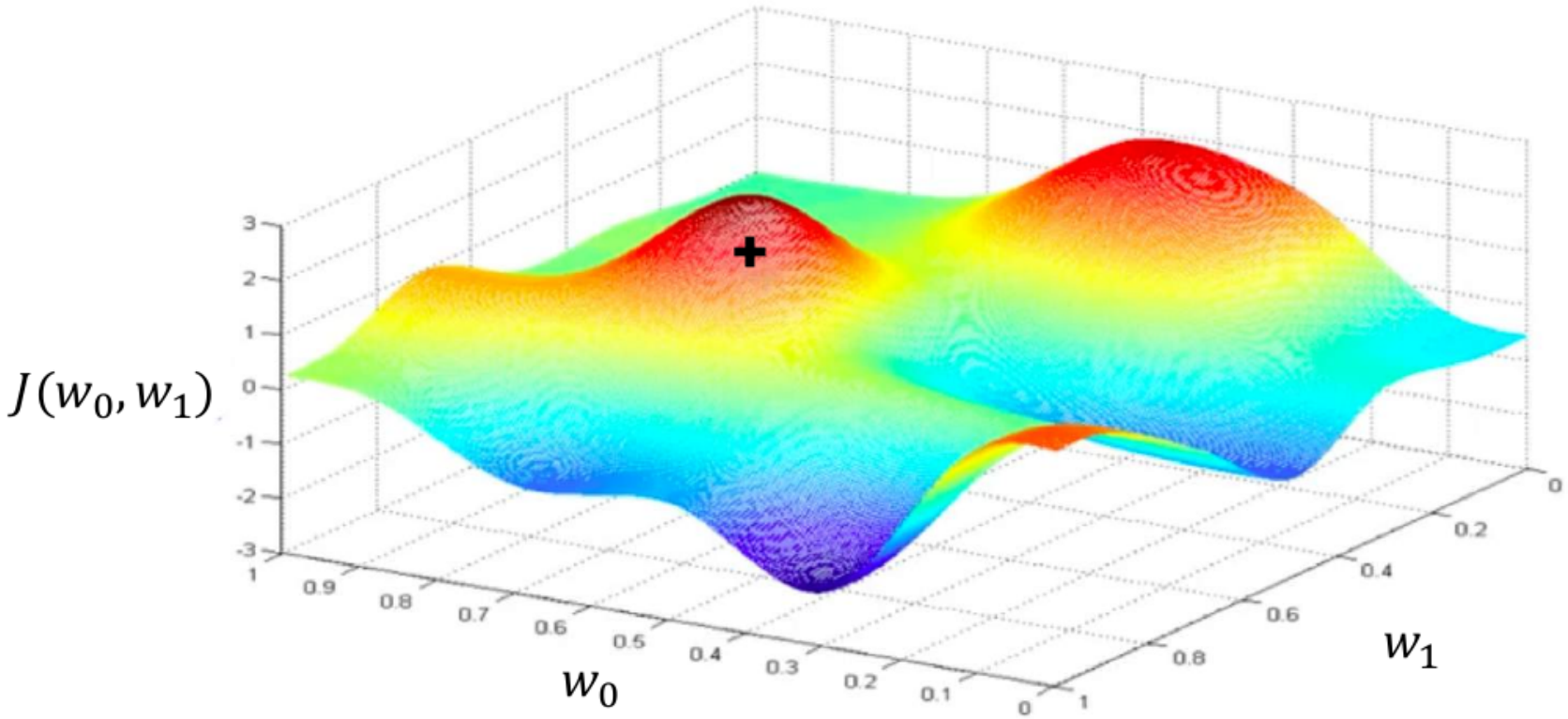
Remember:
Our loss is a function of the network weights!



Back Propagate

Gradient descent

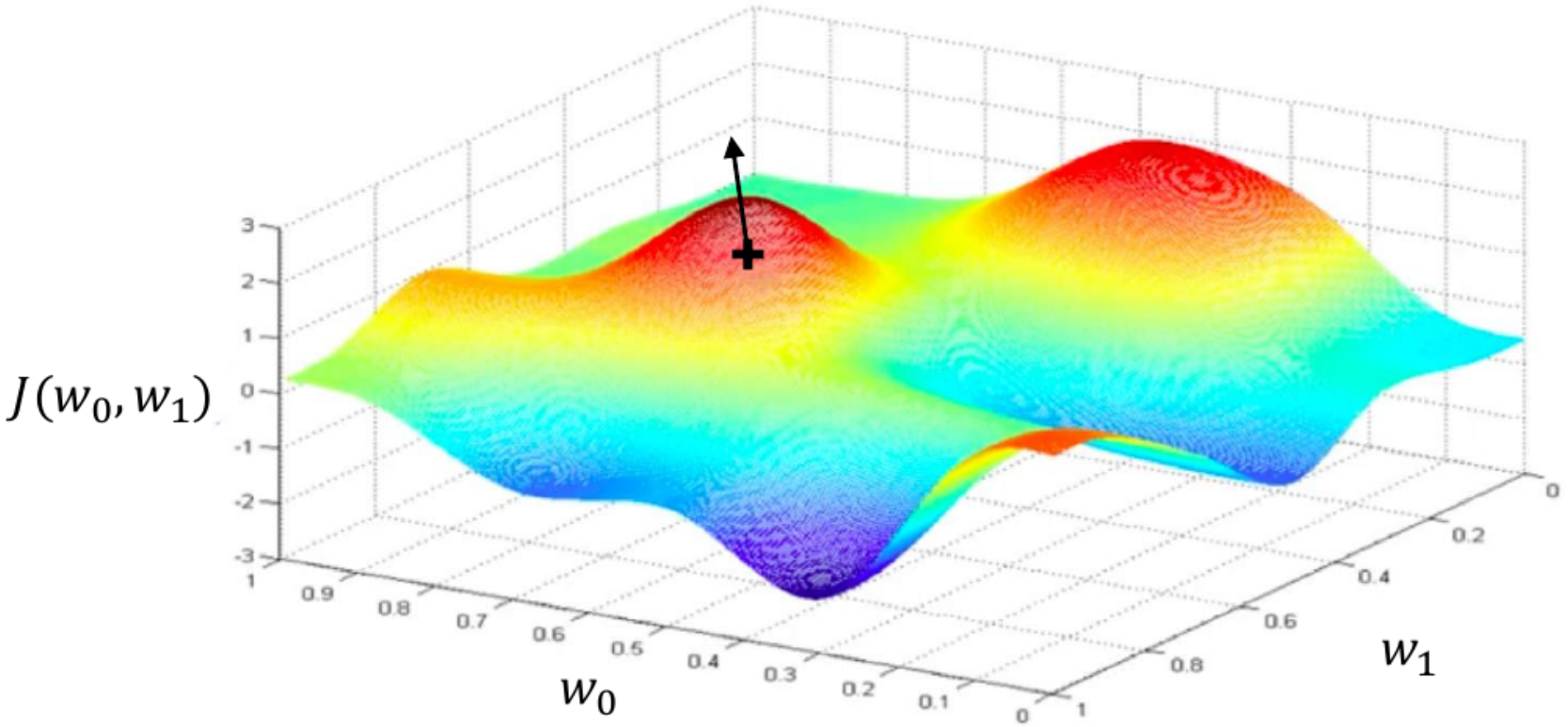
Randomly pick an initial (w_0, w_1)



Back Propagate

Gradient descent

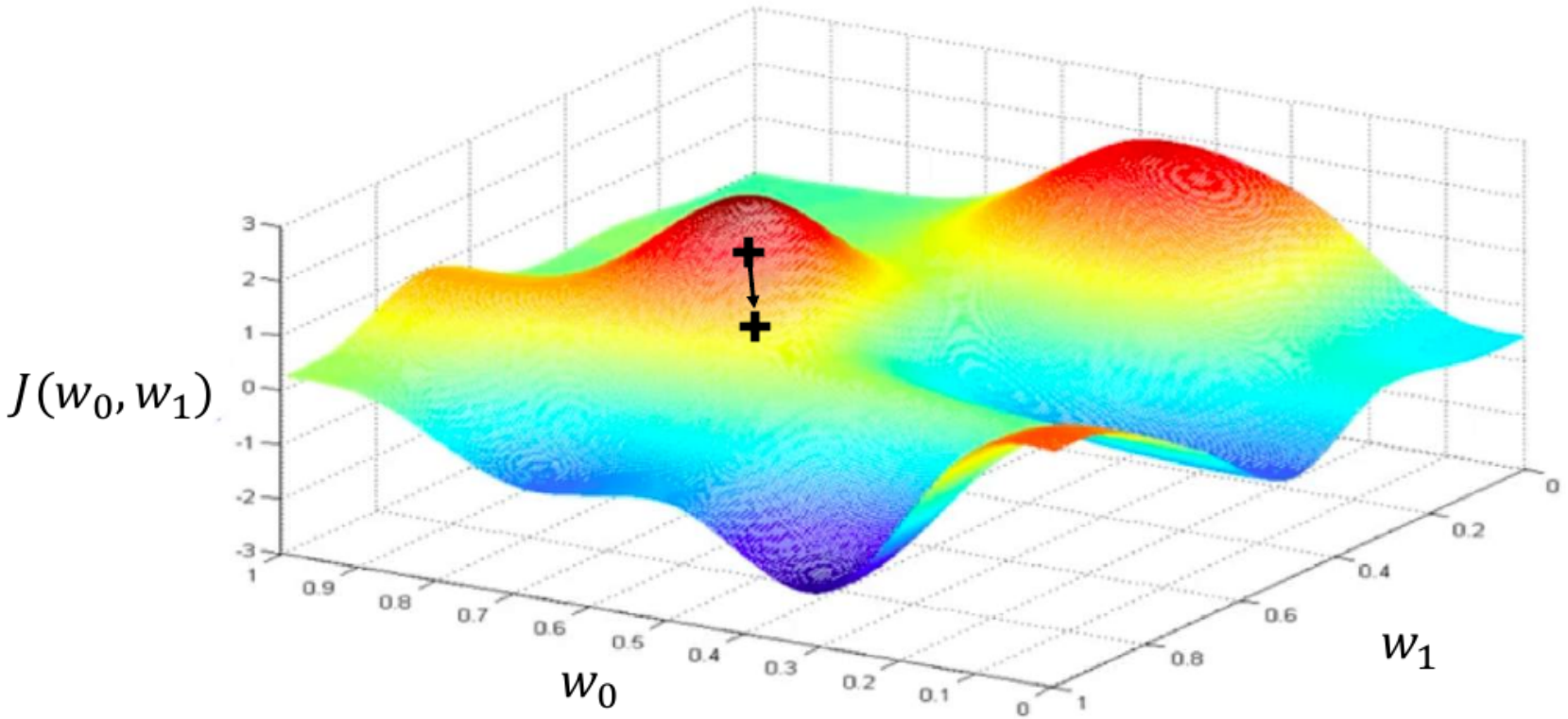
Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$



Back Propagate

Gradient descent

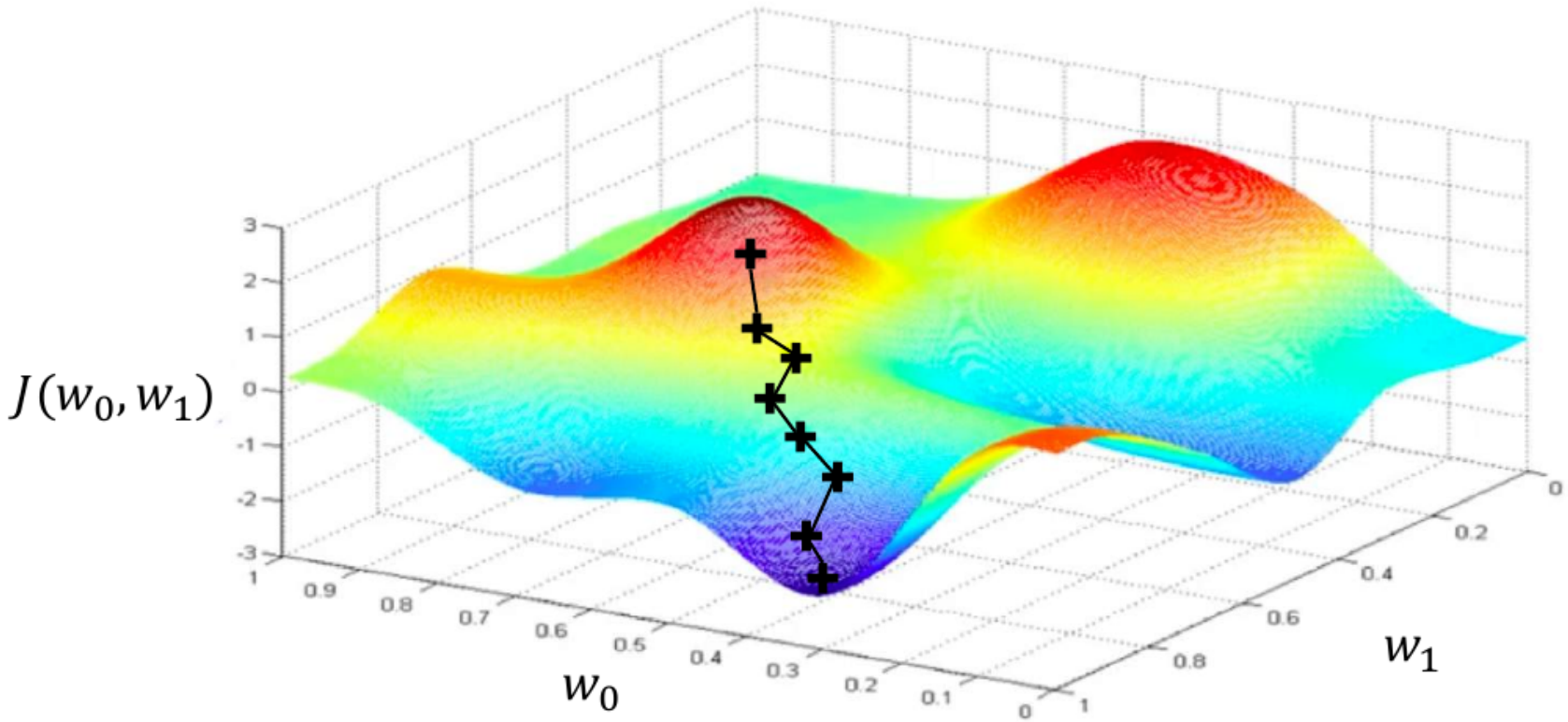
Take small step in opposite direction of gradient



Back Propagate

Gradient descent

Repeat until convergence



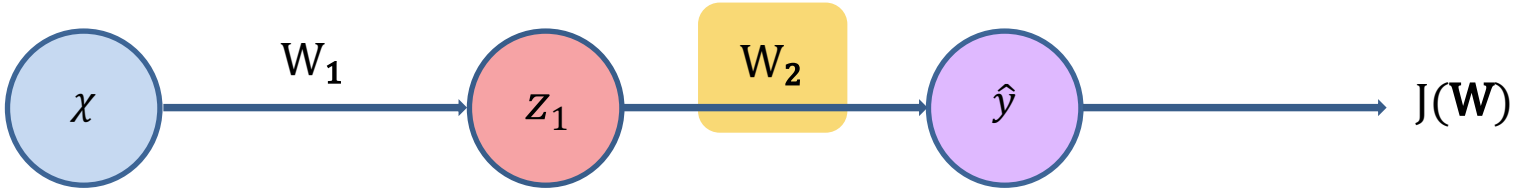
Back Propagate

Gradient descent

Algorithm

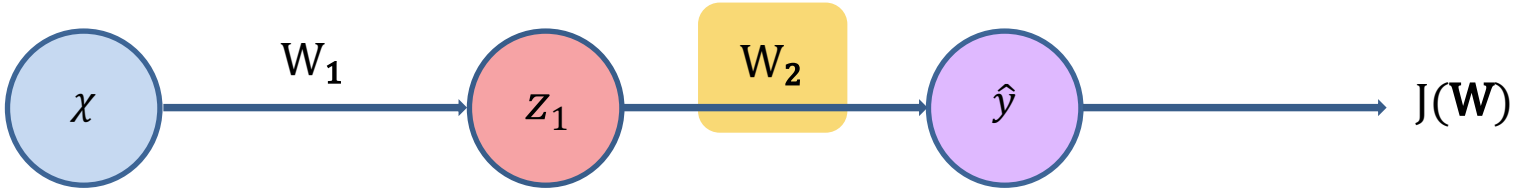
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} \leftarrow \eta \frac{\partial J(W)}{\partial W}$
5. Return weights

Backpropagation



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

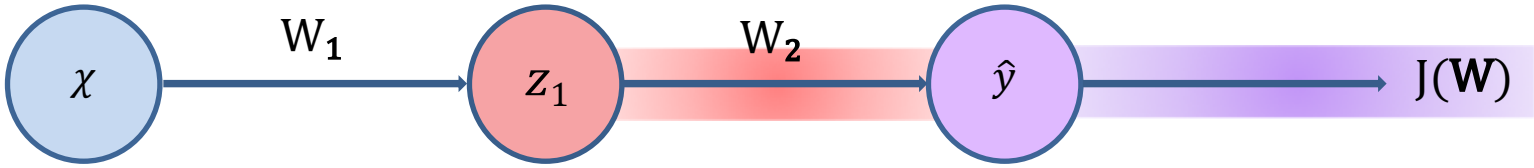
Backpropagation



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

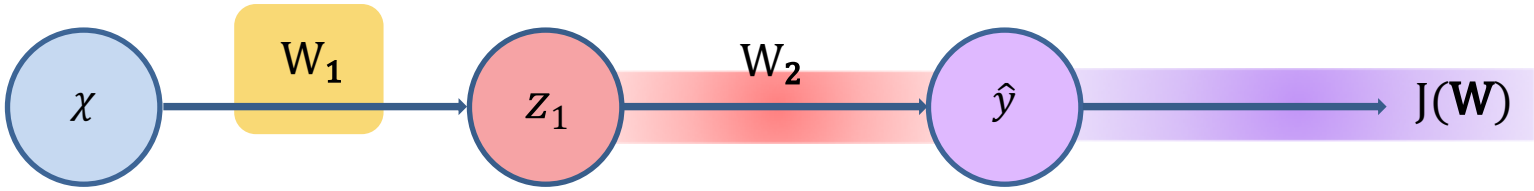
Let's use the chain rule!

Backpropagation



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

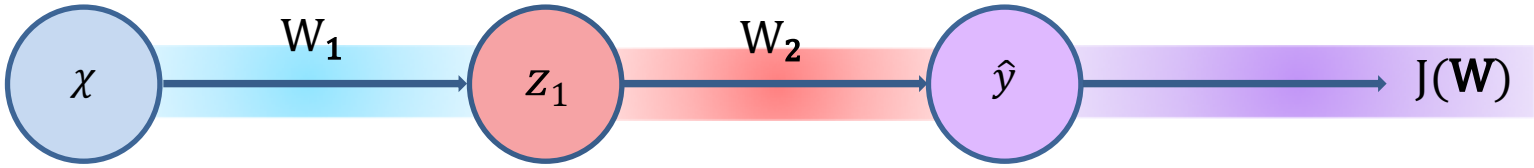
Backpropagation



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

Apply chain rule! Apply chain rule!

Backpropagation



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

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

Curriculum

Next: II. Advanced concepts

- Regularization
- Convolutional Neural Networks