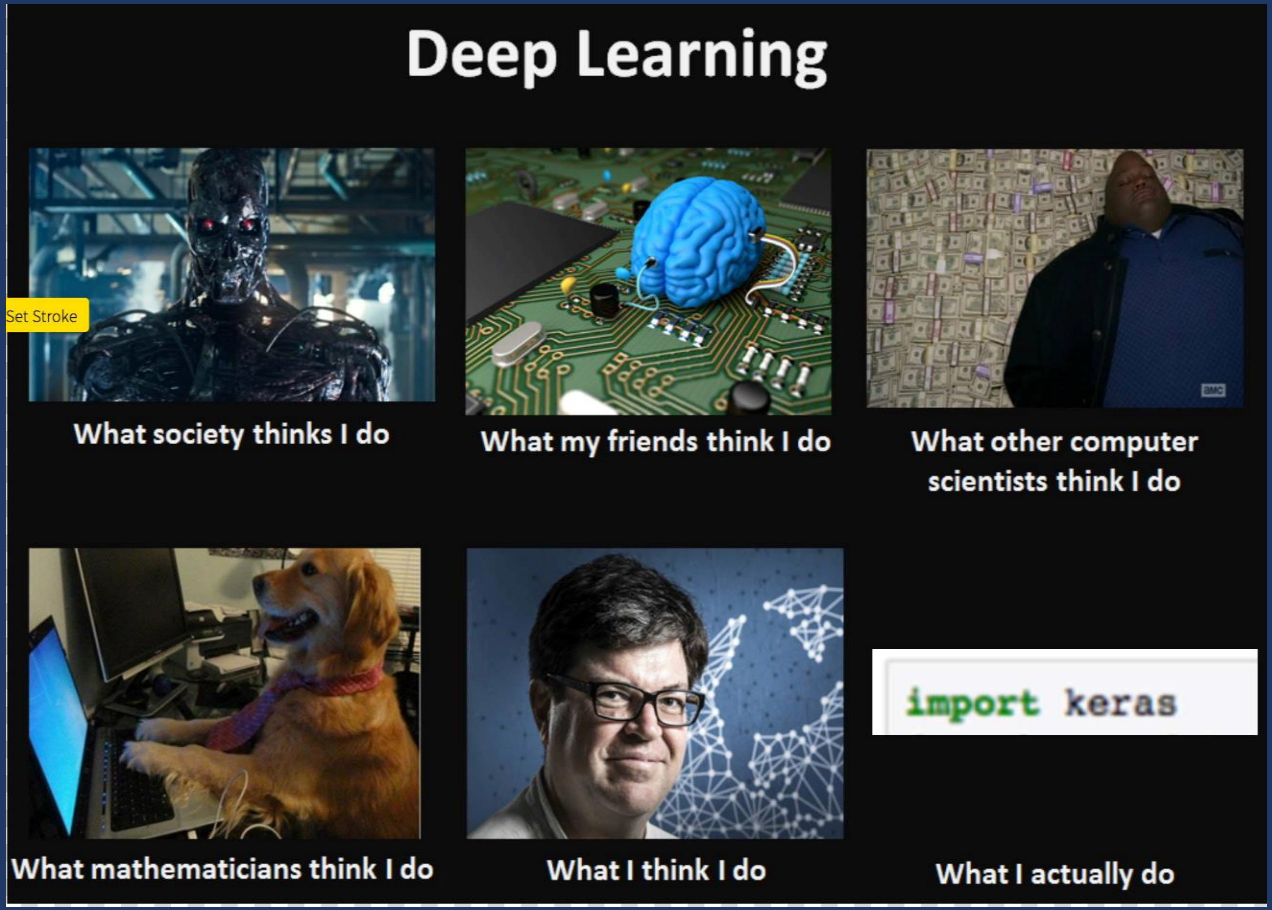
Deep Learning
Summer semester '24

1. Introduction to Deep Learning
(+ Course Organization)

What is Deep Learning?

- What is Machine Learning?
- What is the difference between ML and DL?
- What is the relation between ML/DL and AI?

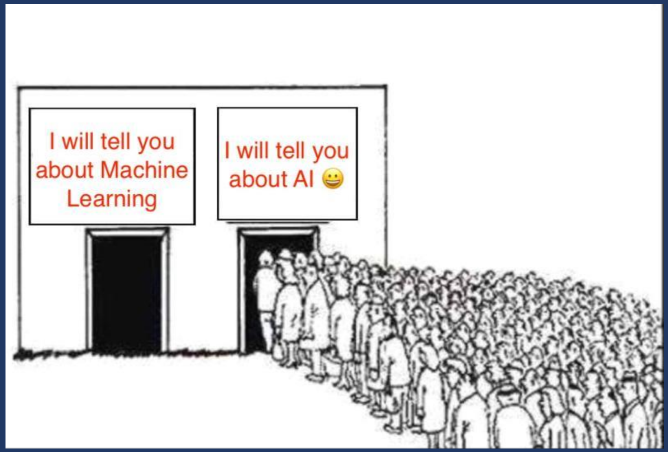


Fahrplan

- Machine Learning
- Deep Learning
- Course Organization

AI vs. Machine Learning

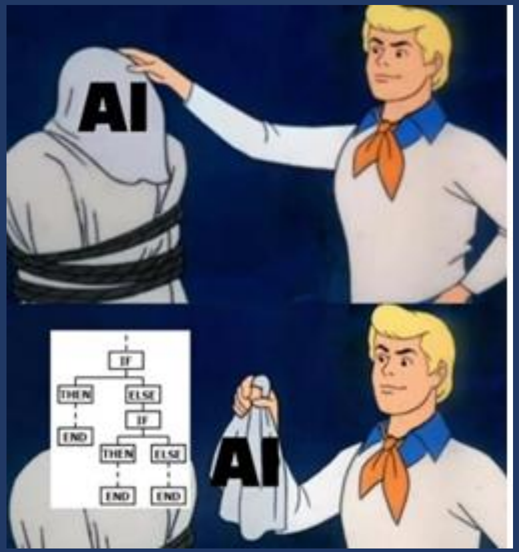
- **Machine learning** – or **learning from data** is the beating heart of modern **AI**
  - (Un)supervised learning
  - Reinforcement learning
  - Representation learning
  - Deep Learning
  - Bayesian Learning
  - Transfer Learning
  - …



Source: https://tinyurl.com/4c86ts2f

Machine Learning

- Successful AI that's not ML-based?
  - Rare, and effectively limited to **rules**
  - Not suited for tackling **complex problems** „in the wild" (any domain)
  - Example: expert systems
    - Popular in the 1980s

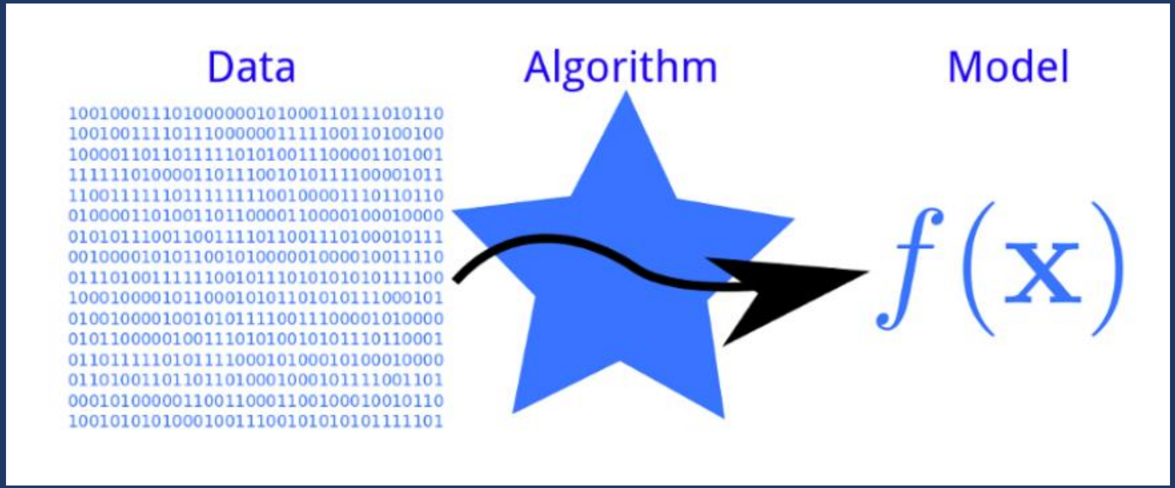


Source: https://tinyurl.com/4c86ts2f

Machine learning

Machine Learning

Machine learning denotes the multitude of algorithms for (semi-)automatic extraction of new and useful knowledge from arbitrary **collections of data** (aka **datasets**). This knowledge is typically captured in the form of rules, patterns, or **models**.



Source: https://tinyurl.com/mpd39647

Why Machine Learning?

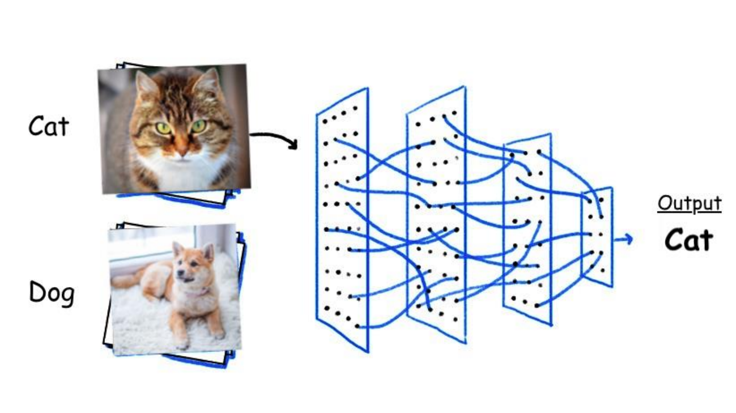
- Write an algorithm (in pseudocode) for the following problems...

Image Classification

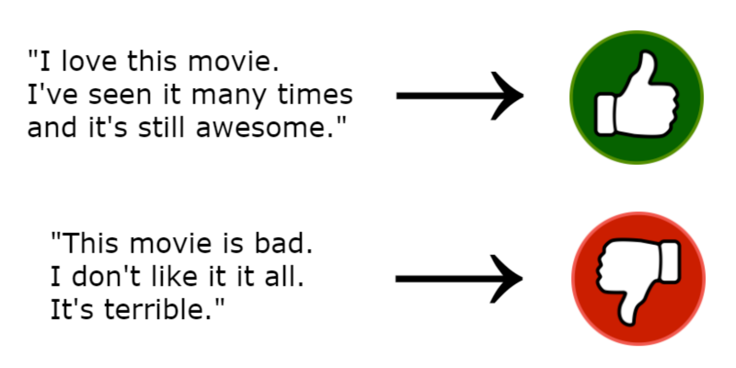
Given an **arbitrary image**, determine which object, from a set of objects C of interest (e.g., C = {*cat*, *dog*, *chicken*}) is on the image.

Sentiment Analysis

Given an **arbitrary product review** (text in natural language), determine whether it expresses positive or negative sentiment towards the product.



Source: https://tinyurl.com/yhtnxm3x



Source: https://cfml.se/blog/sentiment_classification

AI-Complete Problems

- **AI-Complete Problems:** problems that seem to require „human-like” intelligence, not solvable in classic „algorithmic” way

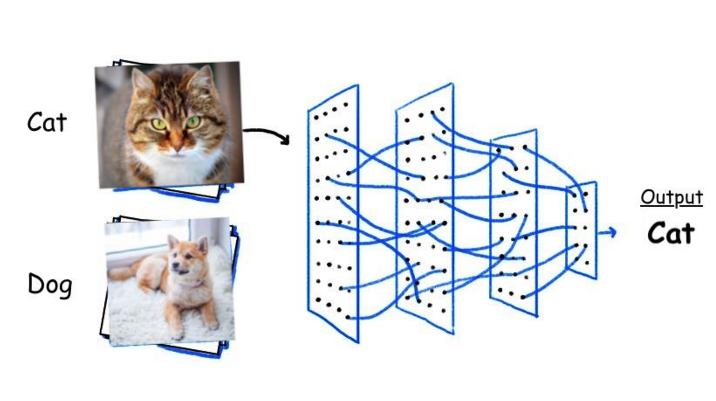
- Classic/Traditional AI: **Search**
  - Humans know how to define and tackle the problem
  - This knowledge is **„codifiable”** into a set of instructions
  - Machines solve the problems more efficiently

- Modern AI Approach: **Learning**
  - There is no codifiable human knowledge on how to reach a solution
  - Humans don’t know how to explain the solution to the problem (e.g., speech recognition)
  - **Humans typically solve these problems with ease!**

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Source: https://en.wikipedia.org/wiki/Sudoku_solving_algorithms

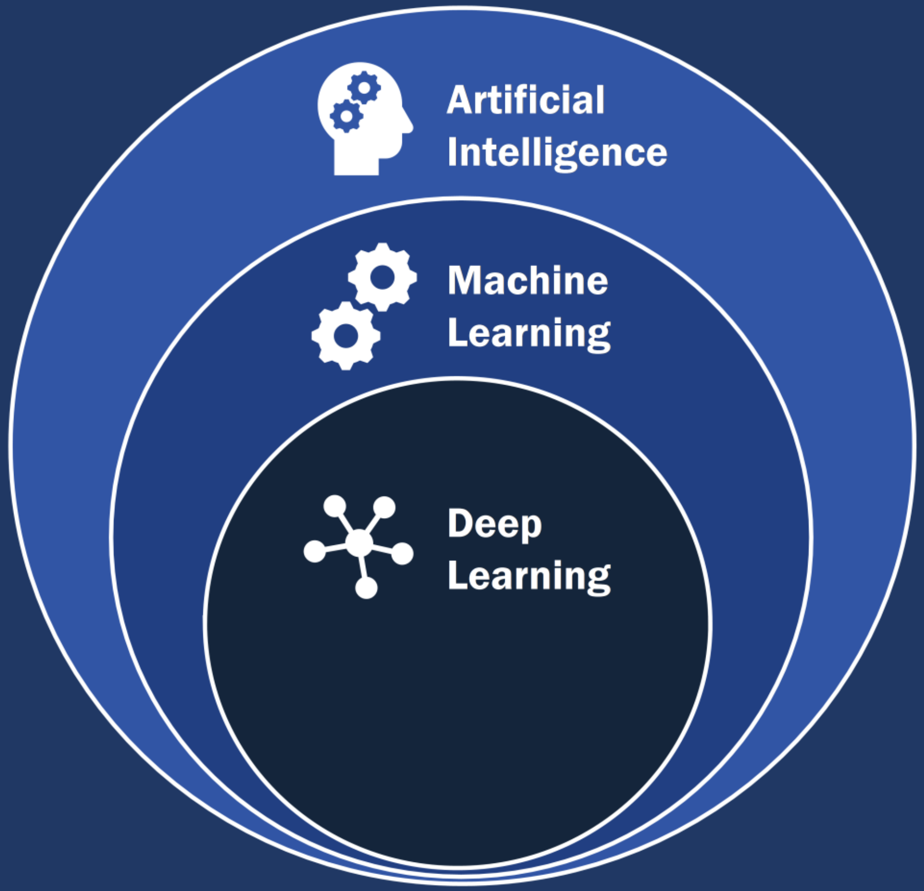
VS.



Source: https://tinyurl.com/yhtnxm3x

AI vs. ML vs. DL

- AI is broader than just ML
- DL is a special type of ML
- 100% of today's **AI hype** is caused by DL models

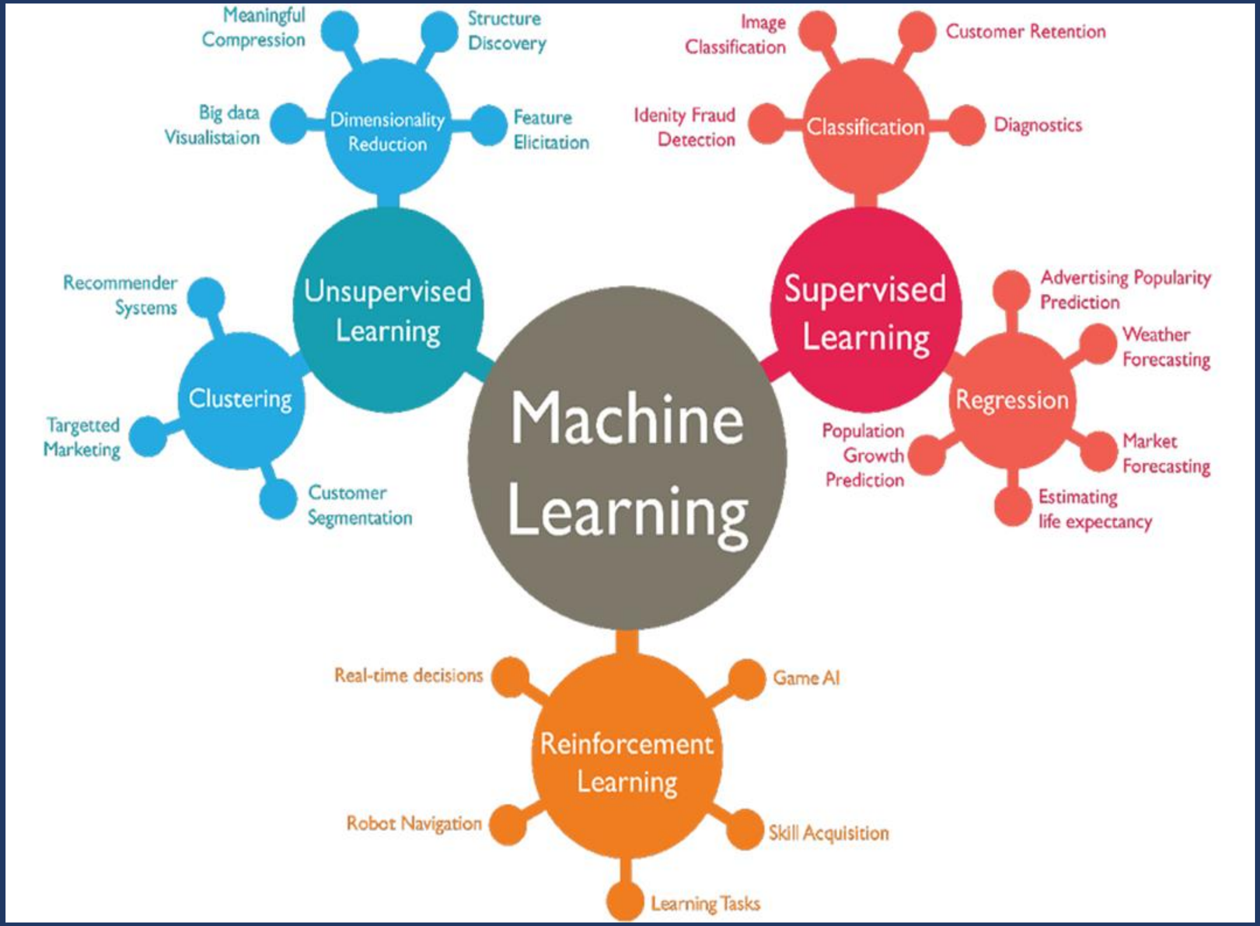


Source: https://tinyurl.com/2yy97tu3

ML Paradigms

- Three main paradigms
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

- In each of the three paradigms there are
  - DL models
  - Non-DL (traditional ML) models



Source: https://vitalflux.com/great-mind-maps-for-learning-machine-learning/

Supervised Learning

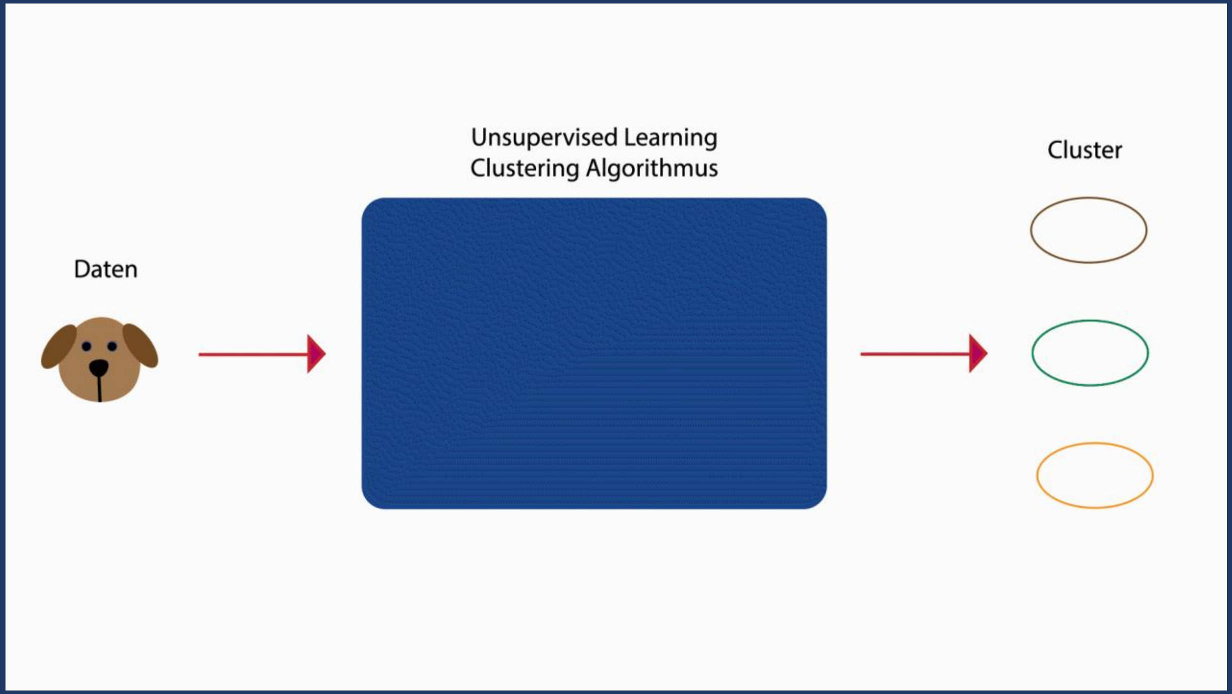
- We have labeled data
  - Inputs with correct labels
  - Labeled data used to „train" the ML model

- Classification
  - Label is discrete (class)
- Regression
  - Label is continuous (score)

Source:https://www.tecislava.com/blog/supervised-unsupervised-reinforcement

Unsupervised Learning

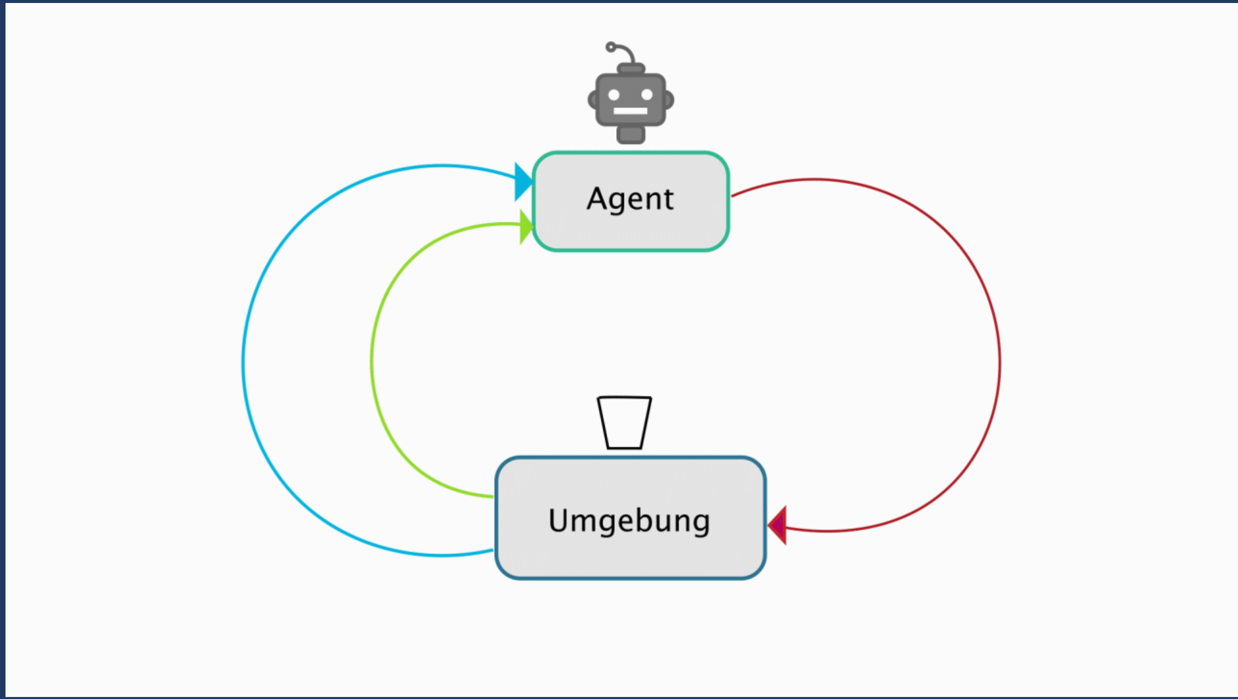
- We have only input data instances, no labels
  - E.g., only images of objects, no indication which objects
- Clustering
  - Grouping similar inputs
- Outlier detection
  - Finding instances very dissimilar from most other
- Dimensionality reduction
  - Finding regularities in data in lower-dimensional spaces



Source:https://www.tecislava.com/blog/supervised-unsupervised-reinforcement

Reinforcement Learning

- An agent interacts with an environment to achieve a goal
  - The agent takes actions that change the state of the environment
- Agent typically makes several actions to achieve the goal
  - Policy decides which action to take at each step
- Reward: an indirect label, specifies whether the goal was achieved
  - Learning = adjusting the policy based on the reward



Source:https://www.tecislava.com/blog/supervised-unsupervised-reinforcement

Space of Examples

- We typically operate in (vector) **spaces** of examples in which **individual examples** (aka **instances**) are concrete points

Space of Examples in ML

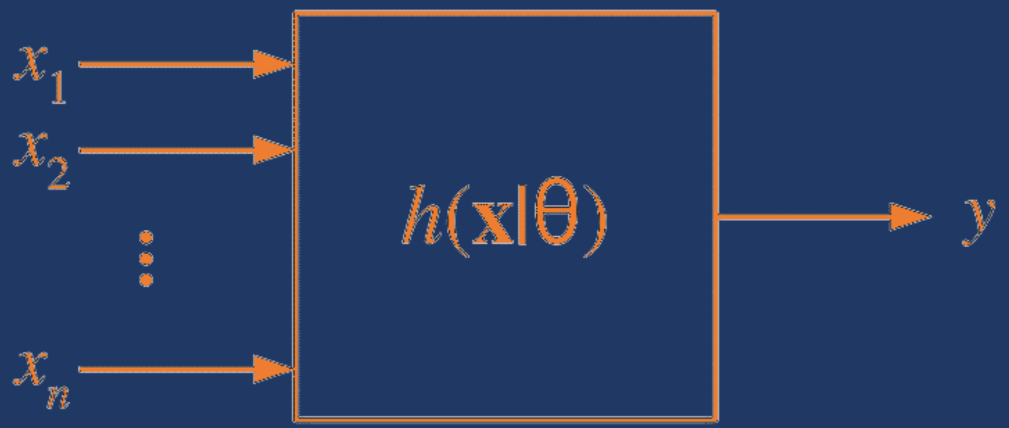
In **machine learning**, individual examples (or instances) $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ are **points** in a space $\mathbf{X}$, consisting of values for **features** $x_1, x_2, \ldots, x_n$. The space $\mathbf{X}$ is the determined (i.e., spanned) by the domains of the features: $\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_n$. The domains of different features can be **discrete** (the so-called categorical or multinomial features) or **continuous**.

- $[x_1, x_2, \ldots, x_n]$ is a **feature vector** of the example/instance

Let's start with basics of ML…

- **Input**: example represented by the **feature vector**: $\mathbf{x} = [x_1, x_2, ..., x_n]$
- **Output** (in supervised learning): the **label** y assigned to the example
  - y is a **discrete class** (in classification problems) or a **score** (in regression problems)

- A machine learning **model** *h* maps an input $[x_1, x_2, ..., x_n]$ to a label y
- The model has a set of *k* **parameters** $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_k]$: $y = h(\mathbf{x}|\boldsymbol{\theta})$



Supervised ML: Toy Example

- You want to **learn** a **classifier** that can differentiate between an apple and a banana

- **Instance/example**: some *concrete* apple or some *concrete* banana.
  - Feature vector $\mathbf{x} = [x_1, x_2, x_3, x_4, ...]$

    x_1: length of the fruit
    x_2: circumference
    x_3: weight
    x_4: color
    ...

- **Label**: $y \in \{ c_1 = \text{apple}, c_2 = \text{banana}\}$

Machine Learning Components

- Any ML algorithm/approach has to have the following three components:
  - **Model**
  - **Objective**
  - **Optimization algorithm**

Machine Learning Components

- Any ML algorithm/approach has three components:

1. Model

- A set of functions among which we're looking for the „best" one

$$H = \{h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$$

- **Hypothesis** h = a concrete function obtained for some concrete values of $\boldsymbol{\theta}$
- Model = set of hypotheses

Machine Learning Components

- Any ML algorithm/approach has three components:

2. Objective

- We're looking from the best hypothesis h in the model $H = \{h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$
  - Q: But „best" according to what?
- **Objective** J is a function that quantifies how good/bad a hypothesis h is
  - Usually J is a „loss function" that we're minimizing
- We're looking for h (that is, values of parameters $\boldsymbol{\theta}$) that maximize or minimize the objective J

$$h^* = \mathrm{argmin}_{h \in H} J(h(\mathbf{x}|\boldsymbol{\theta}))$$
$$\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}} J(h(\mathbf{x}|\boldsymbol{\theta}))$$

- ML thus amounts to solving optimization problems

Machine Learning Components

- Any ML algorithm/approach has three components:

3. Optimization algorithm

- An exact algorithm that we use to solve the optimization problem

$$\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}}\, J(h(\mathbf{x}|\boldsymbol{\theta}))$$

- Selection/type of the optimization algorithm depends on the two functions – the model H and the objective J

Example: Linear Regression

- **Linear Regression** is one of the simplest (supervised) ML model
  - **Model**: output is a linear combination of input features
    - Parameters $\boldsymbol{\theta}$: „weights" that define how much to scale each input feature

$$h(\mathbf{x} = [x_1, x_2, \ldots, x_n] \mid \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

  - **Objective** (loss) function: mean square error
    - $D = \{(\mathbf{x}, y)\}_i$ is the training set – pairs of inputs $\mathbf{x}$ with corresponding outputs y

$$L(y, h(\mathbf{x} \mid \boldsymbol{\theta})) = (y - h(\mathbf{x} \mid \boldsymbol{\theta}))^2$$

$$J(h \mid D) = \tfrac{1}{2} \sum_{i=1}^{N} (y_i - h(\mathbf{x}_i \mid \boldsymbol{\theta}))^2$$

  - **Optimization algorithm**:

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \ldots \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,n} \\ & \vdots & \ddots & \vdots \\ 1 & xN,_1 & \cdots & x_{N,n} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_N \end{bmatrix}$$

Solution is then computed as:

$$\boldsymbol{\theta}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Example: Logistic Regression

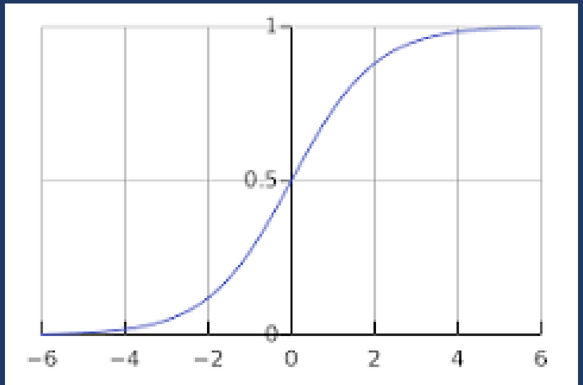
- **Logistic Regression** is one of the most widely used ML models
  - **Model**: logistic function (non-linearity) applied on linear comb. of inputs
    - Parameters $\boldsymbol{\theta}$: „weights” that define how much to scale each input feature

$$h(\mathbf{x}|\boldsymbol{\theta}) = \sigma(\mathbf{x}^T\boldsymbol{\theta})$$

$$= \frac{1}{1+\exp(-\mathbf{x}^T\boldsymbol{\theta})}$$

$$= \frac{1}{1+\exp(-(\theta_0+\theta_1 * x_1 + \ldots + \theta_n * x_n))}$$

$\sigma(x) = 1/(1+e^{-x})$



- **Objective** (loss) function: cross entropy error

$$L_{CE}(h(\mathbf{x}_i|\boldsymbol{\theta}), y_i) = -[\, y_i * \ln h(\mathbf{x}_i|\boldsymbol{\theta}) + (1 - y_i) * \ln(1 - h(\mathbf{x}_i|\boldsymbol{\theta}))\,]$$

$$J(h|D) = \frac{1}{N}\sum_{i=1}^{N} L(h(\boldsymbol{x}_i|\boldsymbol{\theta}), y_i)$$

Logistic Regression

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} J$$

$$\text{Minimize per } \boldsymbol{\theta}: -\frac{1}{N}\sum_{i=1}^{N} [\, y_i * \mathit{ln}\, h(\mathbf{x}_i | \boldsymbol{\theta}) + (1 - y_i) * \mathit{ln}\, (1 - h(\mathbf{x}_i | \boldsymbol{\theta}))\,]$$

- **Q:** How do we find the minimum of a continuous function?
  - We compute the gradient and solve the equation „gradient = 0"

$$\nabla_{\boldsymbol{\theta}} J = 0$$

$$\nabla_{\boldsymbol{\theta}} \left[-\frac{1}{N}\sum_{i=1}^{N} [\, y_i * \mathit{ln}\, h(\mathbf{x}_i | \boldsymbol{\theta}) + (1 - y_i) * \mathit{ln}\, (1 - h(\mathbf{x}_i | \boldsymbol{\theta}))\,] \right] = 0$$

  - Unlike for linear regression, this equation has no closed form solution.
  - **Q:** What do we do then?
    - Numerical optimization: gradient descent & co.

Fahrplan

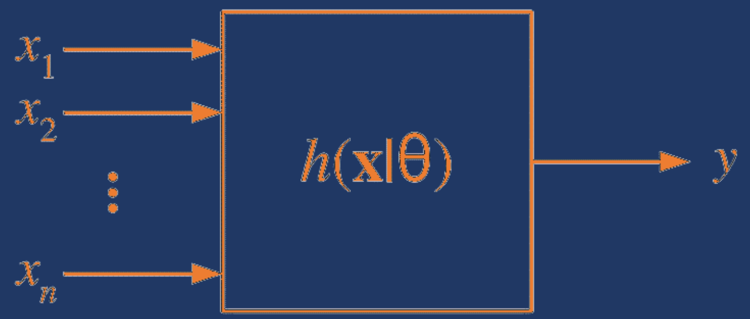
- Machine Learning
- **Deep Learning**
- Course Organization

DL vs. ML

- **Input**: example represented by the **feature vector**: $\mathbf{x} = [x_1, x_2, ..., x_n]$
- **Output** (in supervised learning): the **label** y assigned to the example
  - y is a **discrete class** (in classification problems) or a **score** (in regression problems)

- A machine learning **model** *h* maps an input $[x_1, x_2, ..., x_n]$ to output/label y:
- The model has a set of *k* **parameters** $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_k]$: $y = h(\mathbf{x}|\boldsymbol{\theta})$

Q: So, what is different in **Deep Learning?**



DL vs. ML: Representation Learning

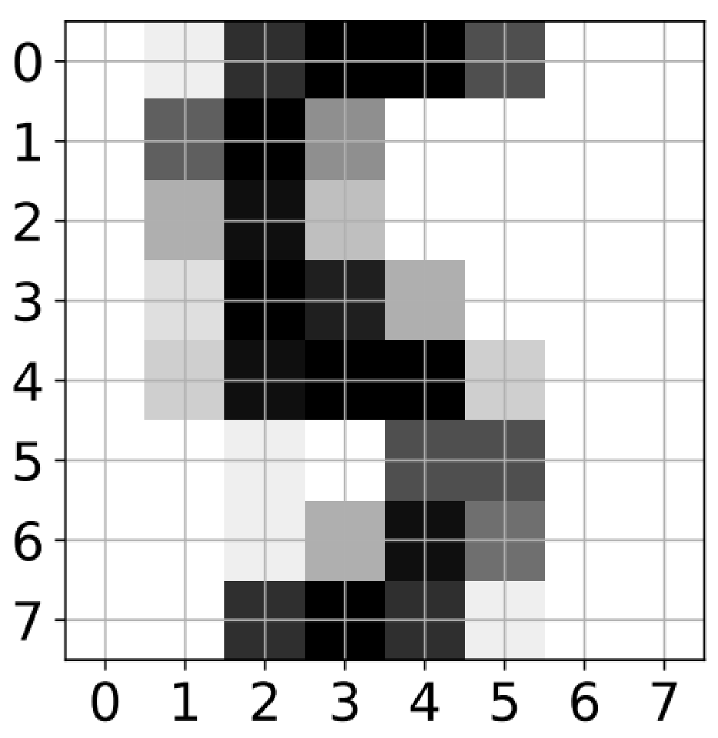
- Q: Anyone heard of „representation learning”?
- In ML that is not DL („traditional” ML), feature calculation (x_1, ..., x_n) is not really part of the ML model/algorithm itself
- „Manual feature design”
  - we need design and precompute good features for the problem

Example: Handwritten digit classification

- **Input**: 8x8 Pixel Images of handwritten digits
- **Output/Label**: the digit (in the image)

- Which features can we compute from the raw data (image = sequence of pixels) that would be predictive of the actual digit?

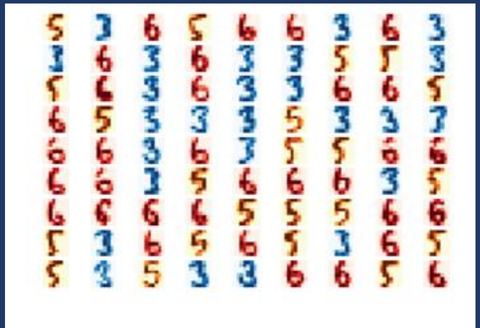
- Feature extraction
  - Option 1: each pixel one feature, 64 features?

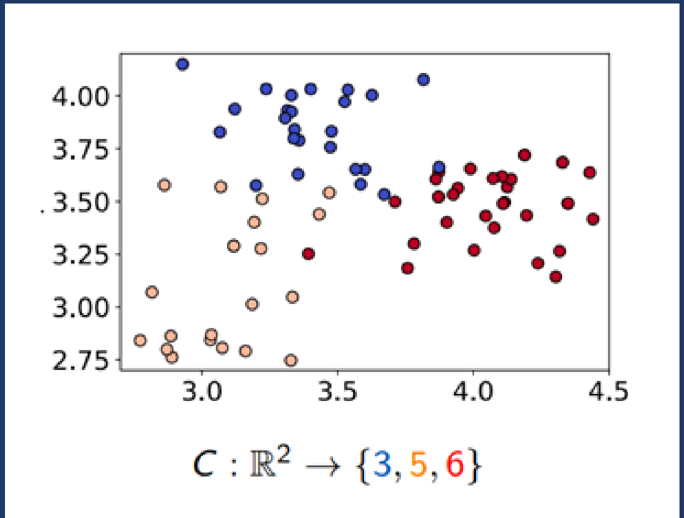


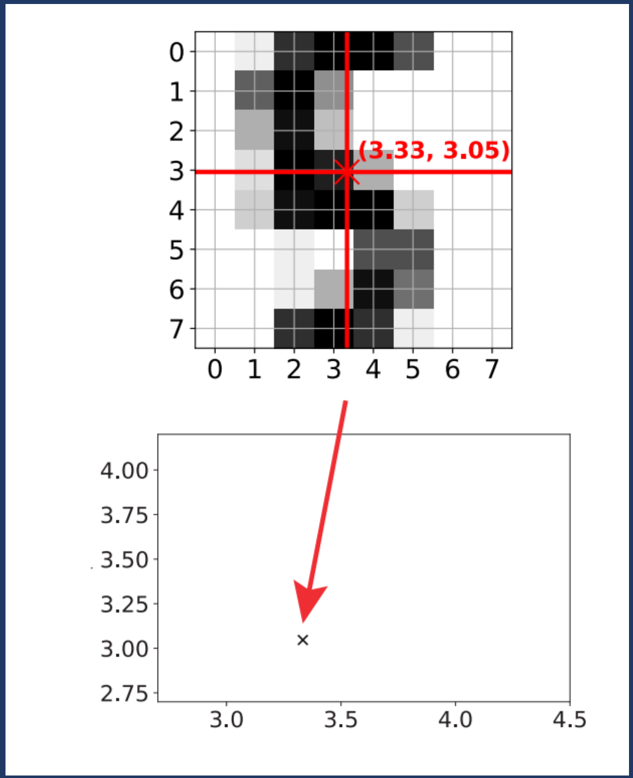
Source: Image/example from Ingo Scholtes

Example: Handwritten digit classification

- Feature extraction
  - Option 1: each pixel one feature, 64 features?
  - Option 2: precompute something indicative – e.g., center of mass (Schwerpunkt)







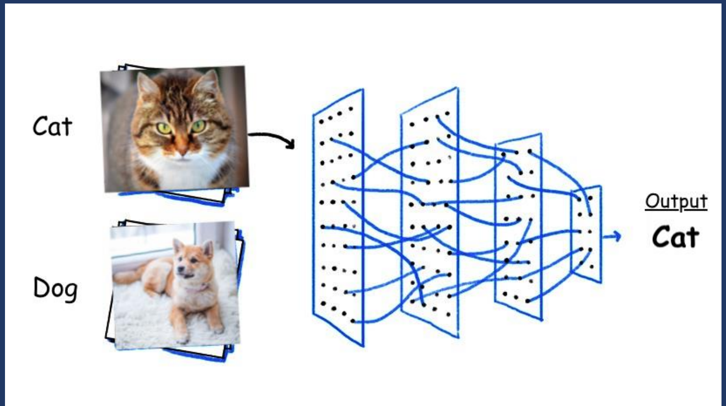
Source: Images/example from Ingo Scholtes

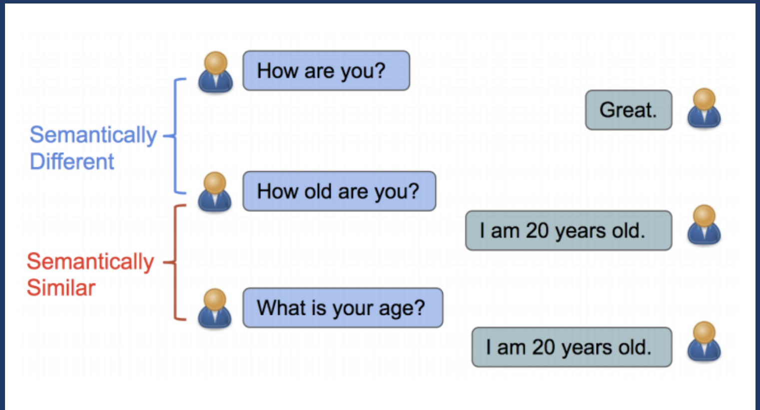
DL vs. ML: Representation Learning

- Two key shortcomings of manual feature design:

1. Difficult (not obvious how) to design good features

- Especially in domains with „unstructured data"
- Visual (Computer Vision) and language data (Natural Language Processing)
  - Q: Good features for image object classification?
  - Q: Good features for semantic text similarity?



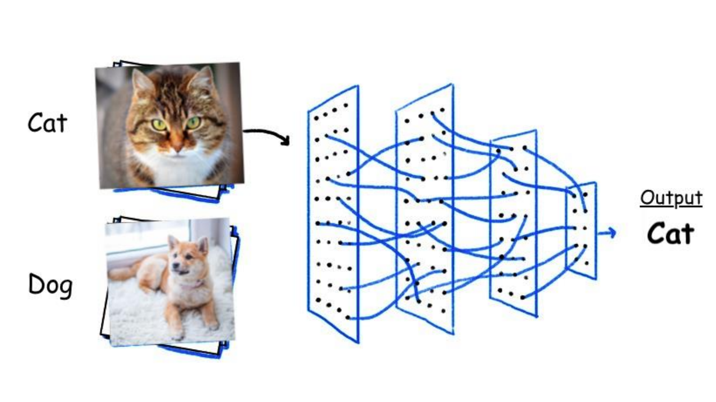


DL vs. ML: Representation Learning

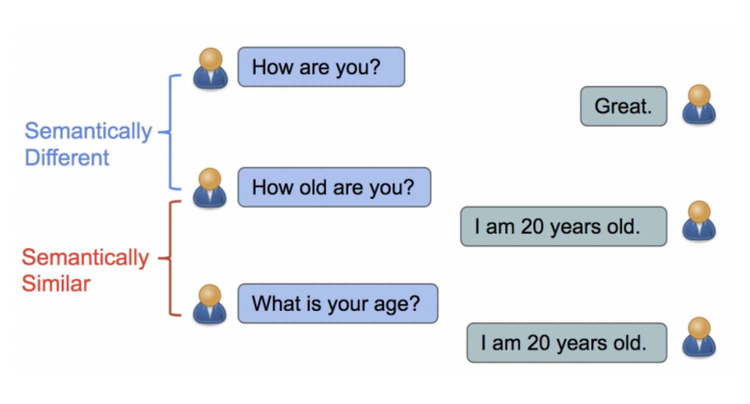
- Two key shortcomings of manual feature design:

2. Loss of information

  - Features compute something from the raw data
  - The classifier in the end sees only the computed features – a lossy representation of the original (whole) data



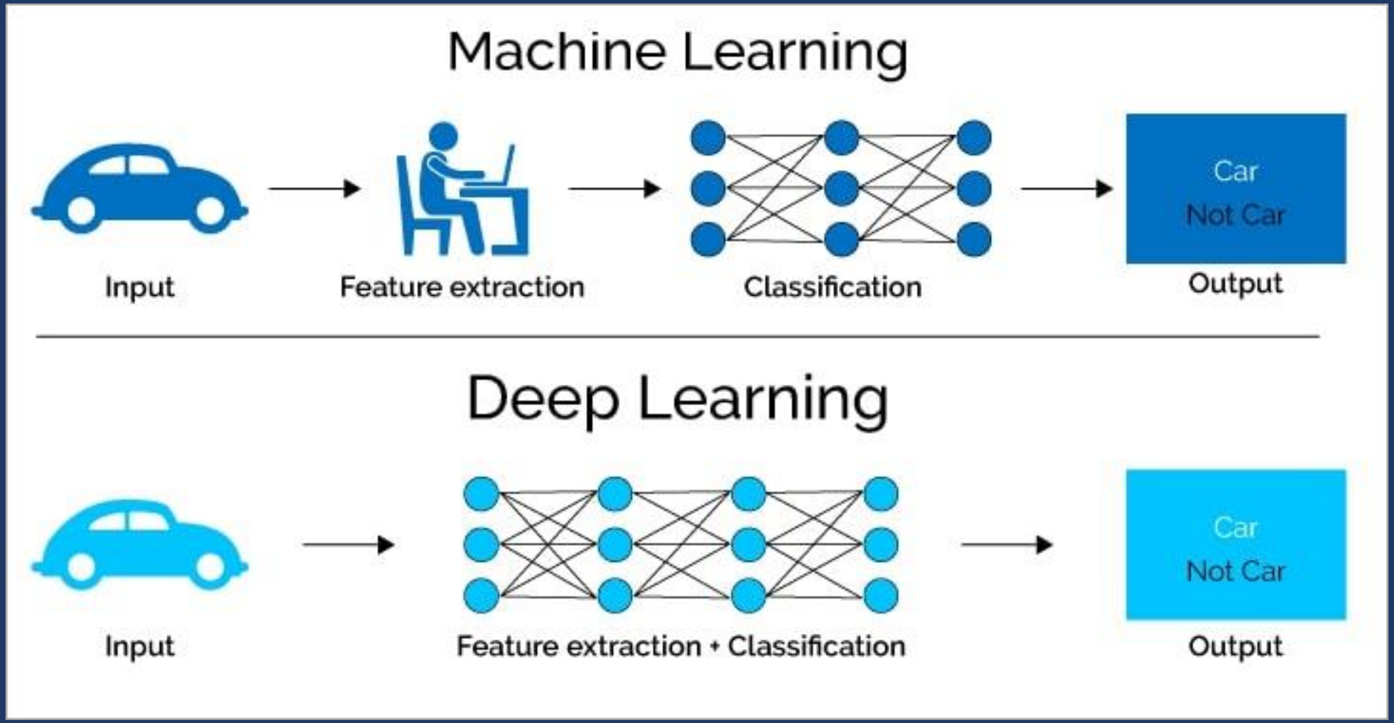
Source: https://tinyurl.com/yhtnxm3x



Source: https://tinyurl.com/3sknhyp6

DL vs. ML: Representation Learning

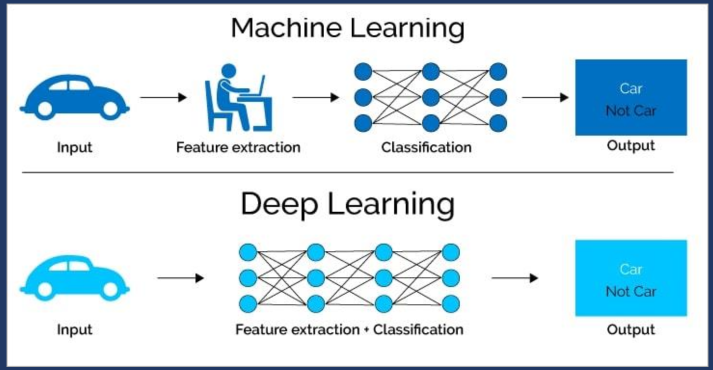
- The key principle of deep learning is representation learning
  - Instead of precomputing features according to human intuition, **let's learn features from the raw data**



Source: https://levity.ai/blog/difference-machine-learning-deep-learning

DL vs. ML: Representation Learning

- **ML**: Feature extraction separate from the model
- **DL**: Feature extraction part of the model



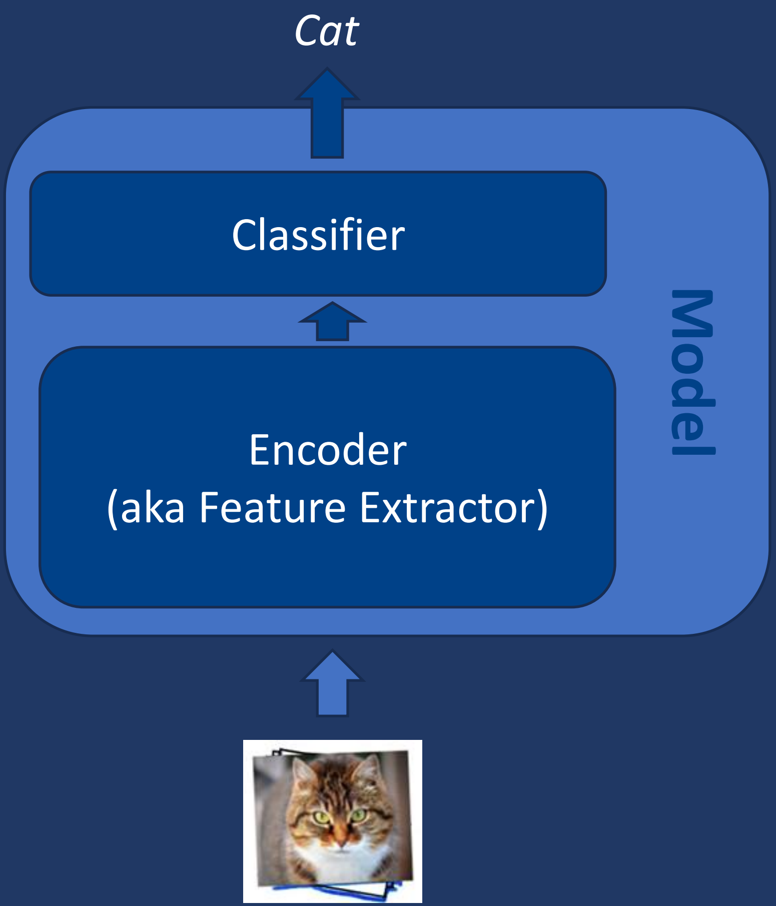
Source: https://levity.ai/blog/difference-machine-learning-deep-learning

- Advantages of FE being part of the model:
  - Removes the need for manual feature extraction
  - No loss of information

- Disadvantage:
  - Features based on which the prediction is made are typically no longer interpretable – just vectors of numbers
  - In manual feature extraction, we know exactly what each feature is and how we computed it from raw data

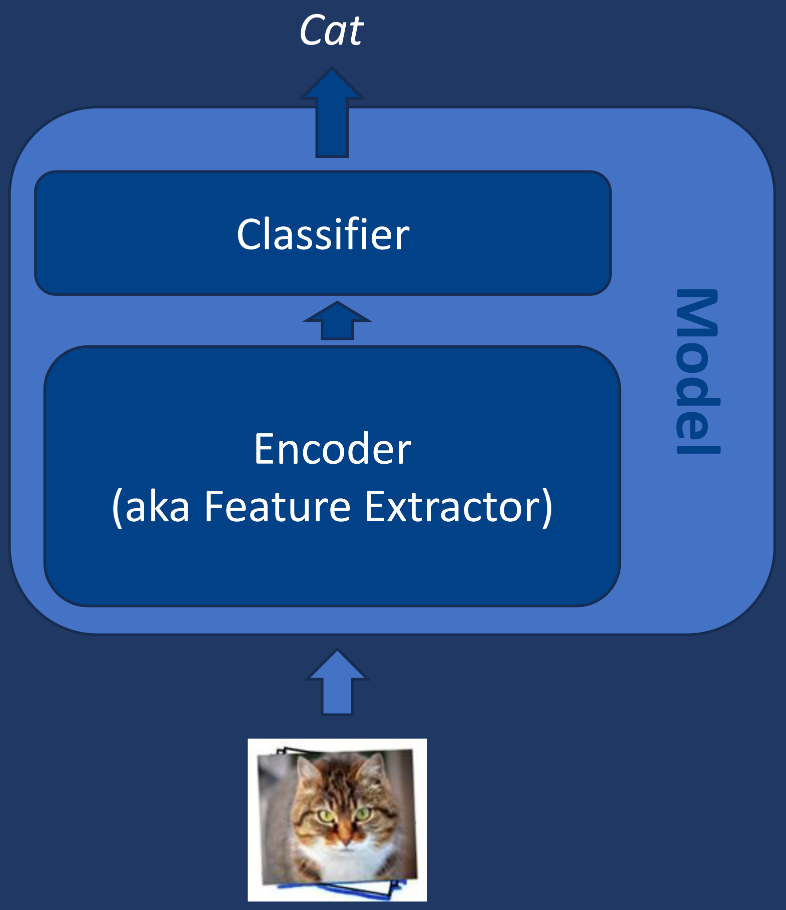
Deep Learning Models

- DL models couple feature extraction with prediction making (classification/regression), thus have two components
- **Encoder**
  - Does the feature extraction
  - In other words, converts the „raw input” into the (latent) feature vector $\mathbf{x}' \in \mathbb{R}^d$
  - „Body” of the model
- **Classifier (or regressor)**
  - Gets the feature vector $\mathbf{x}' \in \mathbb{R}^d$ from the encoder and converts it into a prediction (scalar y or vector $\mathbf{y}$)
  - In traditional ML, the feature vector is precomputed
  - „Head” of the model



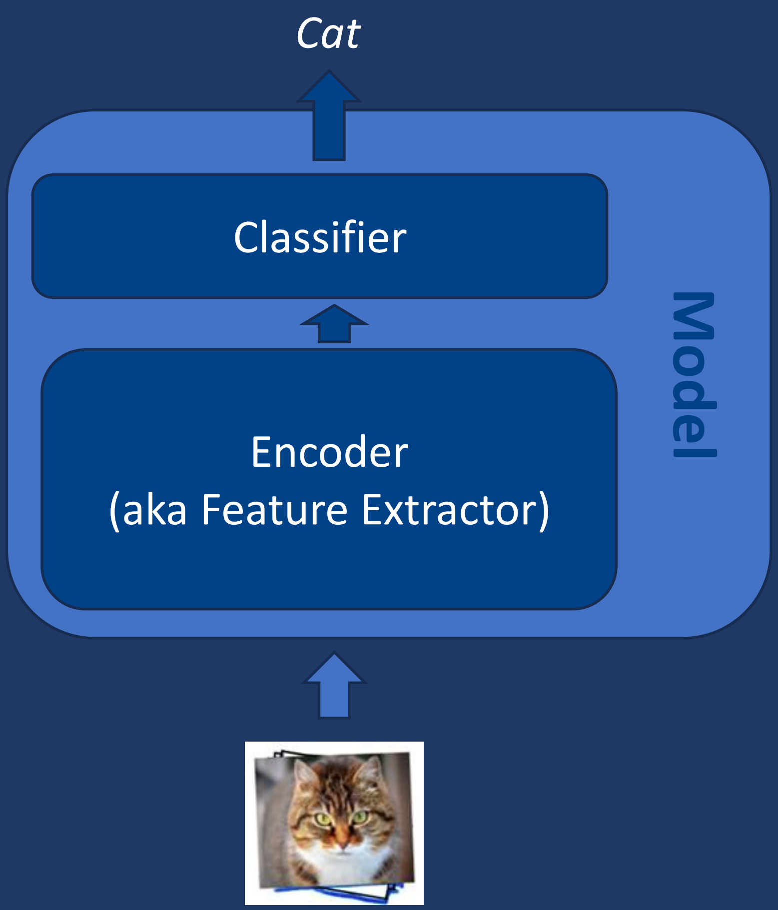
Deep Learning Models

- **Encoder is (usually a complex) parameterized function**
  - $\mathbf{x'} = \mathrm{enc}(\mathbf{x} | \boldsymbol{\theta}_{\mathbf{enc}})$
- **Classifier is (usually a simpler) parameterized function**
  - $\mathbf{y} = \mathrm{cl}(\mathbf{x'} | \boldsymbol{\theta}_{\mathbf{cl}})$
- Encoder and classifier are trained together
  - $\mathrm{model}(\mathbf{x} | \boldsymbol{\theta}_{\mathbf{enc}}, \boldsymbol{\theta}_{\mathbf{cl}})$
  - „end-to-end” training



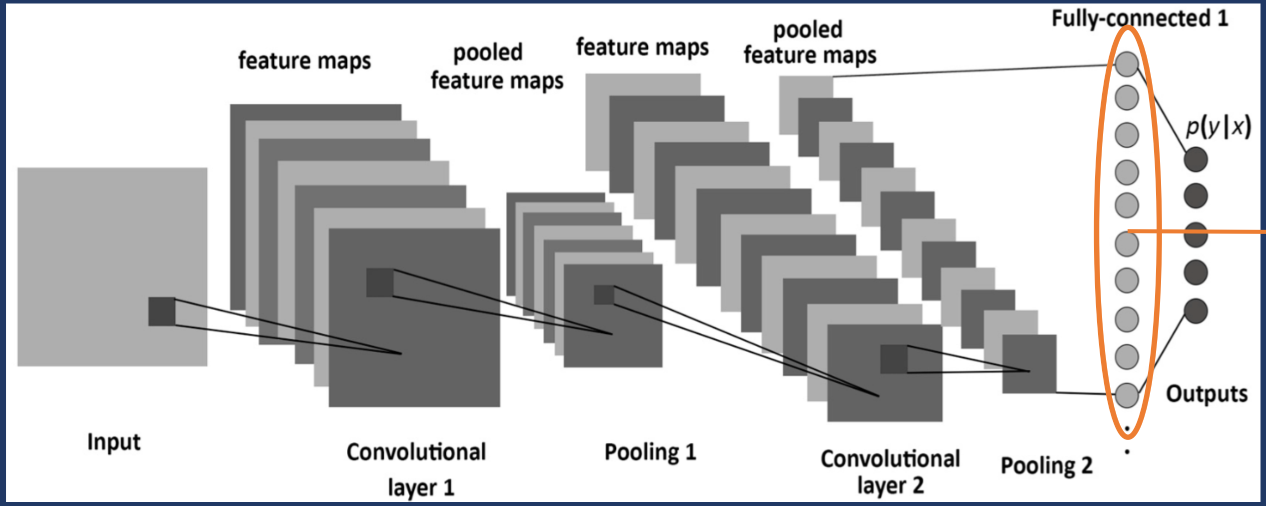
Why „Deep” Learning?

- **Model** is a complex function, a **composition** of a number of **non-linear** parametrized functions
- Encoder and classifier are trained together
  - $\text{model}(\mathbf{x} | \boldsymbol{\theta}_{enc}, \boldsymbol{\theta}_{cl}) = \text{cl}(\text{enc}(\mathbf{x}, \boldsymbol{\theta}_{enc}), \boldsymbol{\theta}_{cl})$
- But encoders/classifiers are also compositions of „subfunctions”
  - Called **layers** in deep learning
  - $\text{enc}(\mathbf{x}, \boldsymbol{\theta}_{enc}) = \text{lay}_n(\text{lay}_{n-1}(\ldots(\text{lay}_1(\mathbf{x} | \boldsymbol{\theta}_1) | \boldsymbol{\theta}_2)\ldots) | \boldsymbol{\theta}_n)$
  - $\boldsymbol{\theta}_{enc} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{n-1}, \boldsymbol{\theta}_n\}$



Example: Deep Convolutional Networks

- Different „layer” functions result in different model **architectures**
  - $\text{enc}(\mathbf{x}, \boldsymbol{\theta}_{enc}) = \text{lay}_n(\text{lay}_{n-1}(\ldots(\text{lay}_1(\mathbf{x}|\boldsymbol{\theta}_1)|\boldsymbol{\theta}_2)\ldots)|\boldsymbol{\theta}_n)$
  - $\boldsymbol{\theta}_{enc} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{n-1}, \boldsymbol{\theta}_n\}$
- Example: CNNs – two types of layers, „convolutional” and „pooling”



features (output of the convolutional encoder)

convolutional encoder **classifier**

Deep Learning: Parameters

- What are actually these **„parameters”** $\boldsymbol{\theta}$?
- Only **vectors** and/or **matrices** of real numbers!
  - Randomly initialized
  - „Learned” in training, via optimization algorithm

- Example: feed-forward networks (aka fully-connected networks aka multi-layer perceptron)
  - $lay_n(lay_{n-1}(\ldots(lay_1(\mathbf{x}|\boldsymbol{\theta}_1)|\boldsymbol{\theta}_2)\ldots)|\boldsymbol{\theta}_n)$
  - $lay_k(\mathbf{x}_{k-1}|\boldsymbol{\theta}_k) = g(\mathbf{x}_{k-1}\mathbf{W}_k + \mathbf{b}_k)$
    - $\mathbf{x}_{k-1} \in \mathbb{R}^m$ – output of the previous, (k-1)-th layer, input for the k-th layer
    - $\mathbf{W}_k \in \mathbb{R}^{m \times n}$ $\mathbf{b}_k \in \mathbb{R}^n$ – (trainable) **parameters** of the k-th layer, $\boldsymbol{\theta}_k = \{\mathbf{W}_k, \mathbf{b}_k\}$
    - g – a non-linear function, for example logistic function: $\sigma(a) = 1/(1+e^{-a})$

Deep Learning in Practice

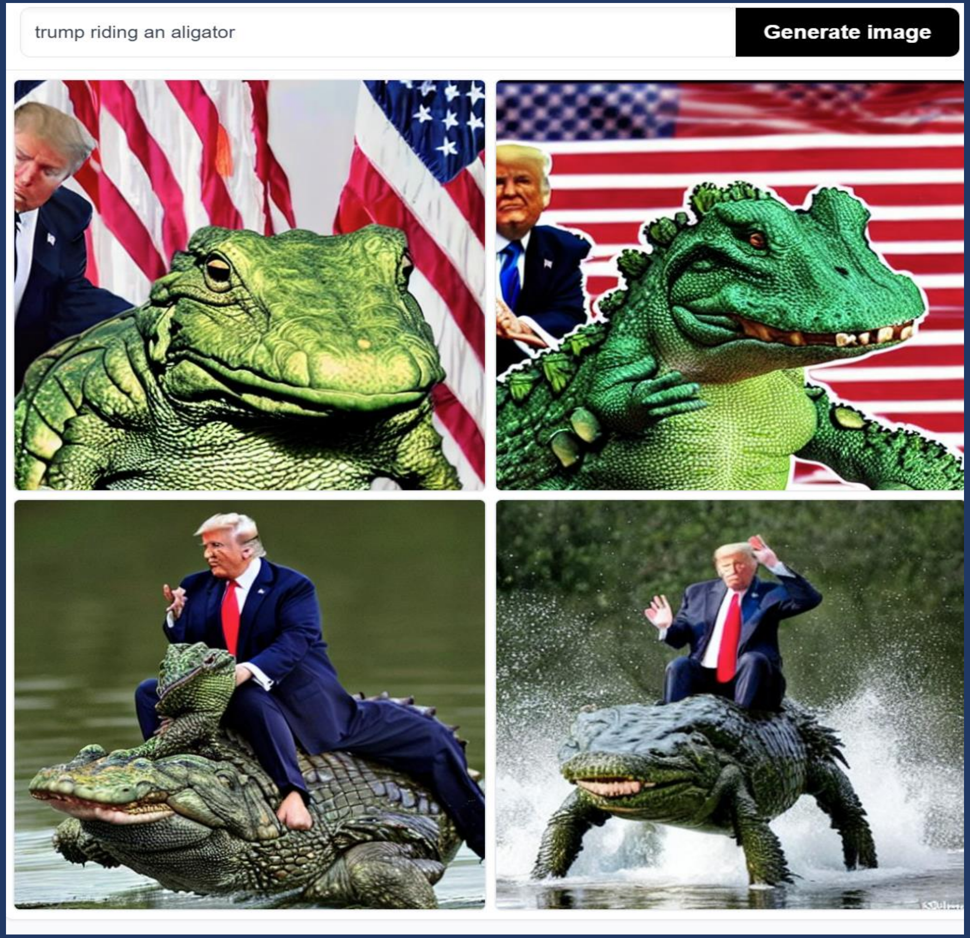
- **All hyped AI** today is based on **DL models**

- **Large Language Models (LLMs)**
  - ChatGPT / GPT-4
  - Gemini/Gemma
  - Llama
  - Vicuna
  - Mistral/Mixtral
  - Command R
  - …

Deep Learning in Practice

- **All hyped AI** today is based on **DL models**

- **(Text-Based)**
  Image Generation
  - Open AI's DALL-E
  - Bing/MS Image Creator
  - DreamStudio (Stability AI)
    - aka StableDiffusion
  - …



Deep Learning in Practice

- **All hyped AI** today is based on **DL models**
- **(Text-Based) Video Generation**

Fahrplan

- **Machine Learning**
- **Deep Learning**
- **Course Organization**

Content & Schedule I

- **L1**: Intro to DL & Organization (April 15; Glavaš)
- **L2**: Building blocks of DL & Feed-forward Nets (April 22; Breininger)
- **L3**: Optimization & Training (April 29; Glavaš)

- **L4**: Convolutional Networks (May 13; Breininger)
- **L5**: Autoencoders & GANs (May 27; Timofte)
- **L6**: Recurrent Networks (June 3; Hotho)

Content & Schedule II

- **L7**: Attention & Transformer (June 10; Hotho)
- **L8**: Introduction to Reinforcement Learning (June 17; D'Eramo)
- **L9**: Deep Reainforcement Learning (June 24; D'Eramo)

- **L10**: Graph Representation Learning (July 1; Scholtes)
- **L11**: Graph Neural Networks (July 8; Scholtes)

- Lectures: Monday, 12-14
- Exercise sessions: Tuesday, 14-16 (same location)

Lecturers

Katharina Breininger

Pattern Recognition

Carlo D’Eramo

Reinforcement Learning

Radu Timofte

Computer Vision

Ingo Scholtes

ML 4 Complex Networks

Andreas Hotho

Data Science

Goran Glavaš

Natural Language Processing

CAIDAS

Exercises

- Practically oriented
  - Though there may be also theoretical/conceptual questions

- Goal: **learn** PyTorch
  - **#1** DL library globally

- You'll be provided with code skeleton (as Jupyter notebooks)
  - Students have to implement key parts of the code (model, training/validation loop, evaluation, etc.)

Exercises

- 10 exercise sheets in total
  - One after each lecture (except in this first week)
  - Sheets published on Tuesdays (after the exercise session)
  - Sheets submitted (to WueCampus) before the next exercise session

- Each sheet evaluated with 0, 1, or 2 points
- In total, max. 20pts, if $\geq$ 17 pts, you get exam bonus
- Exam bonus = if you pass the exam, you get one grade up
- In teams of two students (also possible individually, if preferred)

Exam

- Written exam (most likely)
- Sometime in the second half of July

- Re-exam (Nachholklausur) before the start of the winter semester
    - Early October

Deep Learning
Summer semester ‘24

1. Introduction to Deep Learning
(+ Course Organization)