# Machine Listening for Music and Sound Analysis

# Lecture 2 – Machine Learning/Deep Learning

Dr.-Ing. Jakob Abeßer
Fraunhofer IDMT

jakob.abesser@idmt.fraunhofer.de

https://machinelistening.github.io

# Learning Objectives

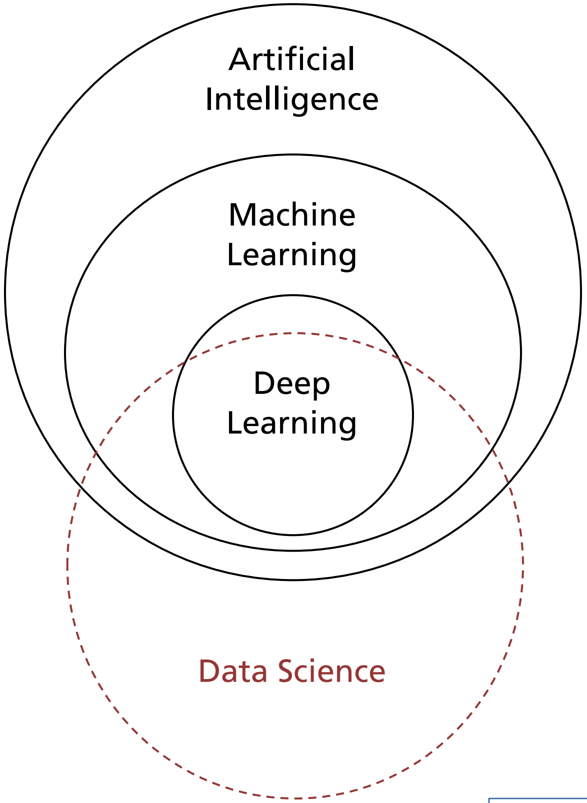
- Introduction
- Learning paradigms
- Machine learning (ML) project pipeline
- Deep learning

# Introduction

- Goals
  - "...give computers the ability to learn without being explicitly programmed" [Samuels, 1959]
  - Learning structures in given (un)labeled data to make predictions on new / unseen data
- Paradigm change
  - <u>Before</u>: manually designed / general-purpose features
  - <u>Now</u>: joint representation learning (features) & data modeling (classification)
- Related disciplines
  - Statistics, data science, optimization

# Introduction
## Terminology

- Artificial Intelligence (AI)
  - “an agent‘s ability to achieve goals in a wide range of environments“ [Legg & Hutter, 2007]
- Machine Learning (ML)
  - Pattern recognition, data modeling, learning, prediction
- Deep Learning (DL)
  - (Brain-inspired) artificial neural networks (ANN)
- Data Science
  - Knowledge extraction from data




Own

# Introduction
## Application Scenarios

- Computational finance (credit scoring, algorithmic trading)
- Computer vision (face & object recognition, motion detection)
- Computational biology (tumor detection, drug discovery, DNA sequencing)

# Introduction
## Application Scenarios

- Computational finance (credit scoring, algorithmic trading)
- Computer vision (face & object recognition, motion detection)
- Computational biology (tumor detection, drug discovery, DNA sequencing)
- Energy (price & load forecasting)
- Predictive maintenance (automotive, aerospace, manufacturing)

# Introduction
## Application Scenarios

- Computational finance (credit scoring, algorithmic trading)
- Computer vision (face & object recognition, motion detection)
- Computational biology (tumor detection, drug discovery, DNA sequencing)
- Energy (price & load forecasting)
- Predictive maintenance (automotive, aerospace, manufacturing)
- Natural language processing (sentiment classification, text search, translation)
- Machine listening (music transcription, instrument recognition, sound event detection, acoustic scene classification)

# Learning Paradigms

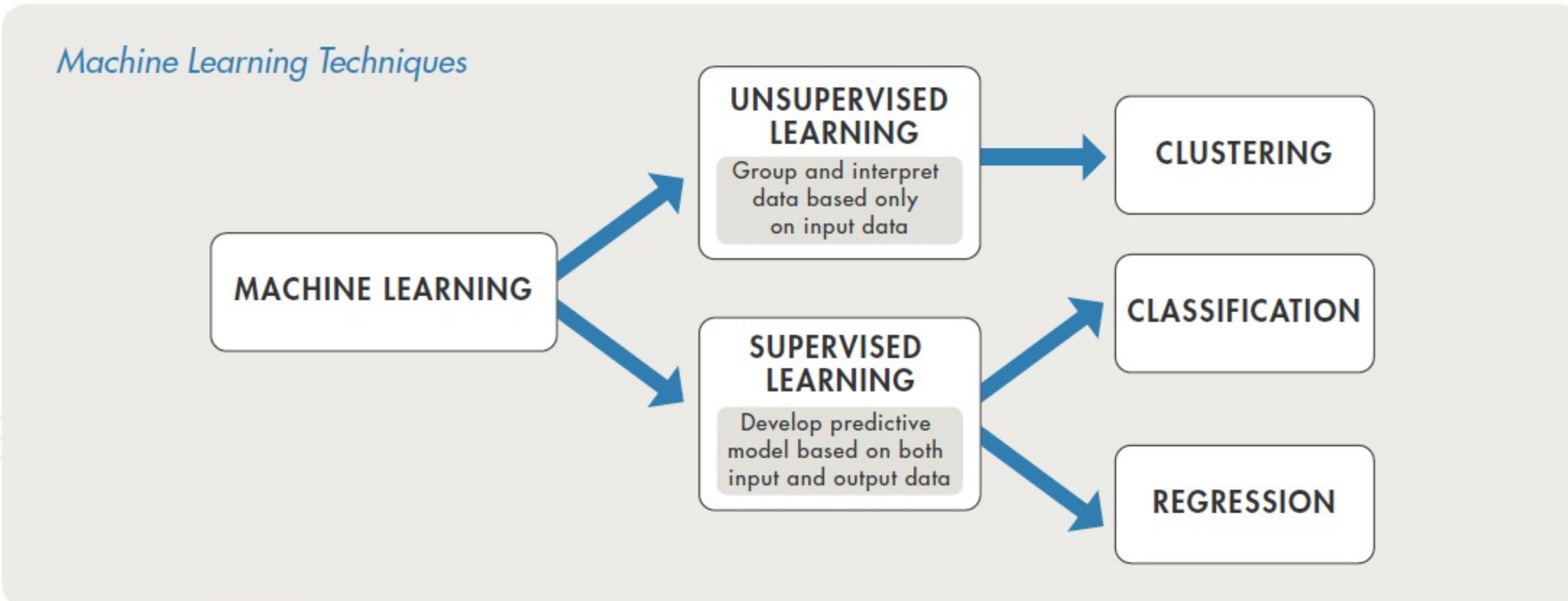



Fig. 1

# Learning Paradigms
## Unsupervised Learning

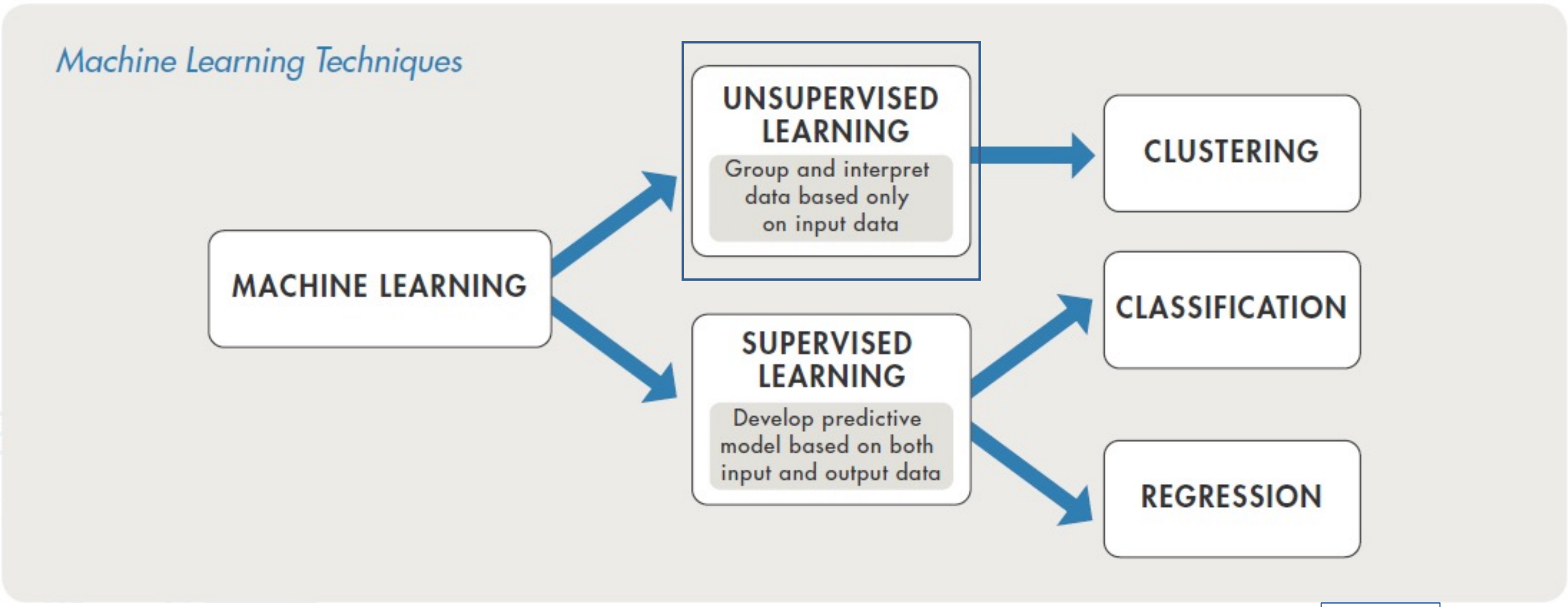


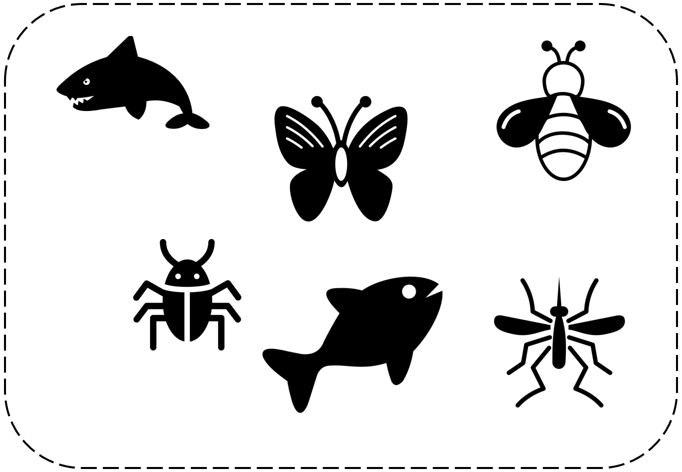

Fig. 1

# Learning Paradigms
## Unsupervised Learning

- Goal
  - Find hidden **structure** and **patterns** in data
  - **No annotations** available

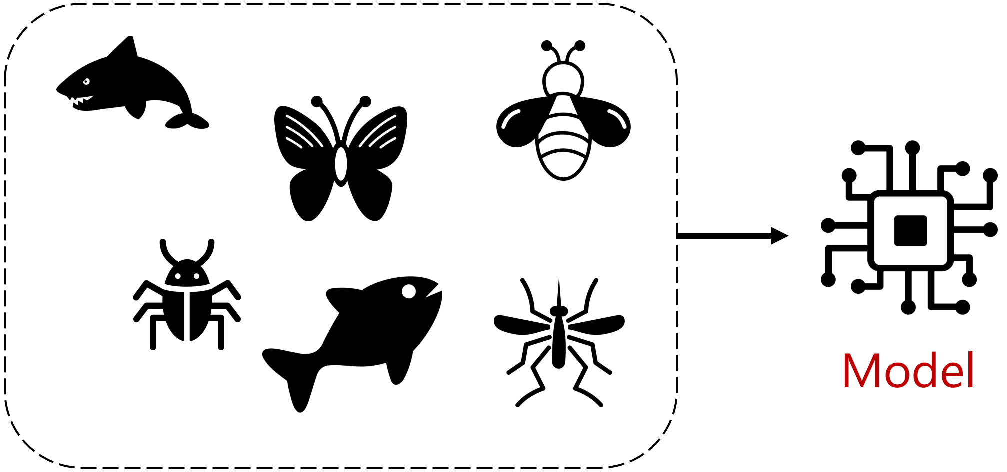
# Learning Paradigms
## Unsupervised Learning

- Goal
  - Find hidden **structure** and **patterns** in data
  - **No annotations** available
- Clustering
  - **Grouping** of **similar** data instances



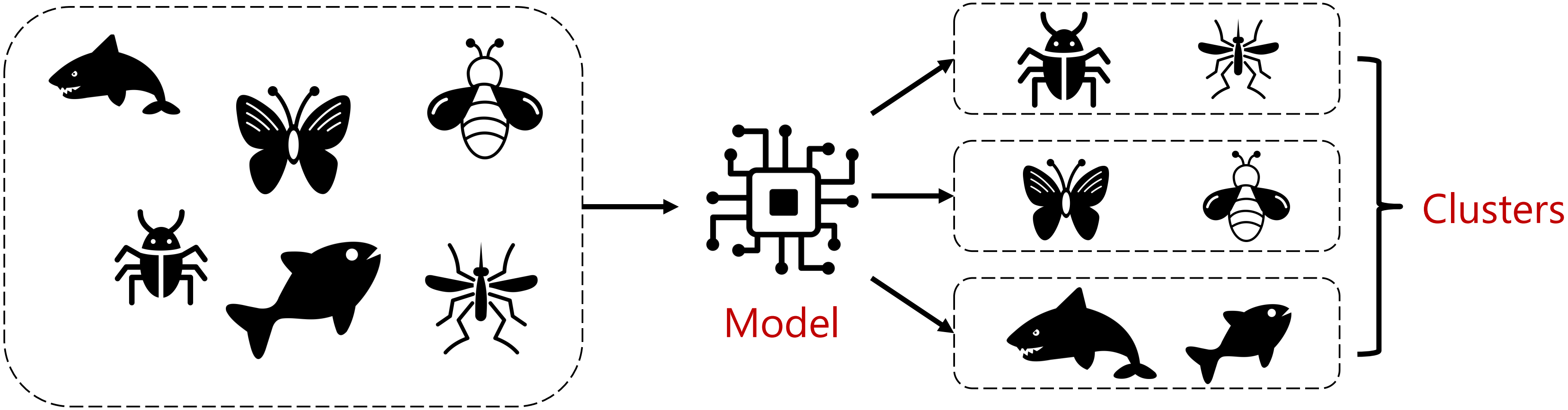
# Learning Paradigms
## Unsupervised Learning

- Goal
    - Find hidden **structure** and **patterns** in data
    - **No annotations** available
- Clustering
    - **Grouping** of **similar** data instances



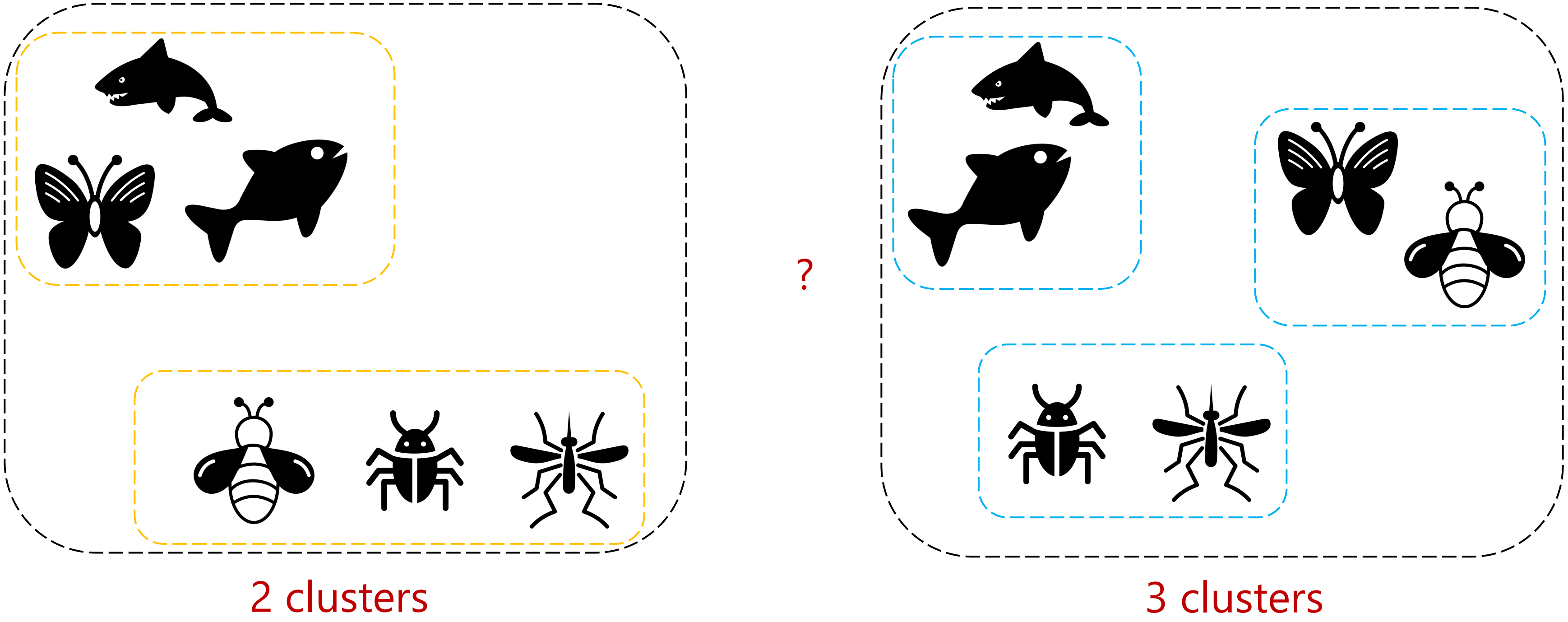
# Learning Paradigms
## Unsupervised Learning

- Goal
  - Find hidden **structure** and **patterns** in data
  - **No annotations** available
- Clustering
  - **Grouping** of **similar** data instances



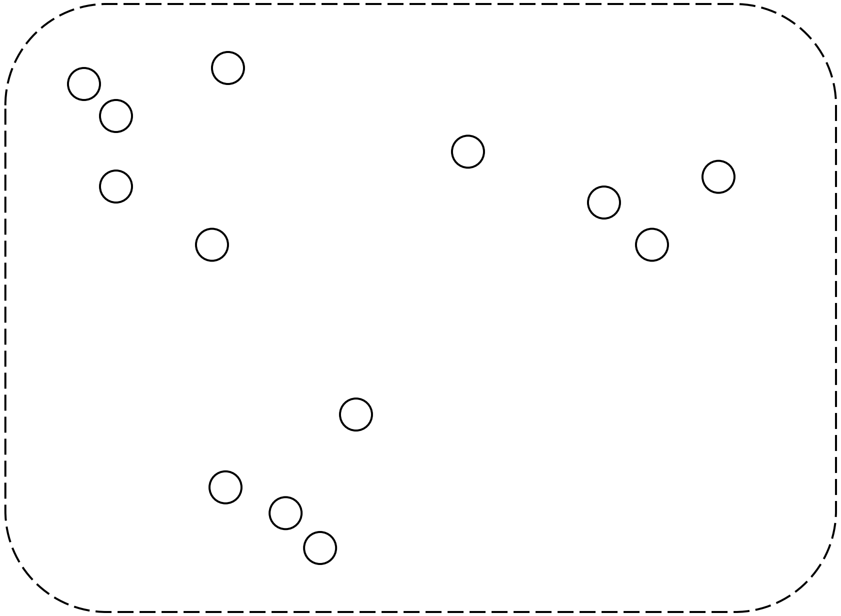
# Learning Paradigms
## Unsupervised Learning

- Challenges
  - What is the **optimal number of clusters**?



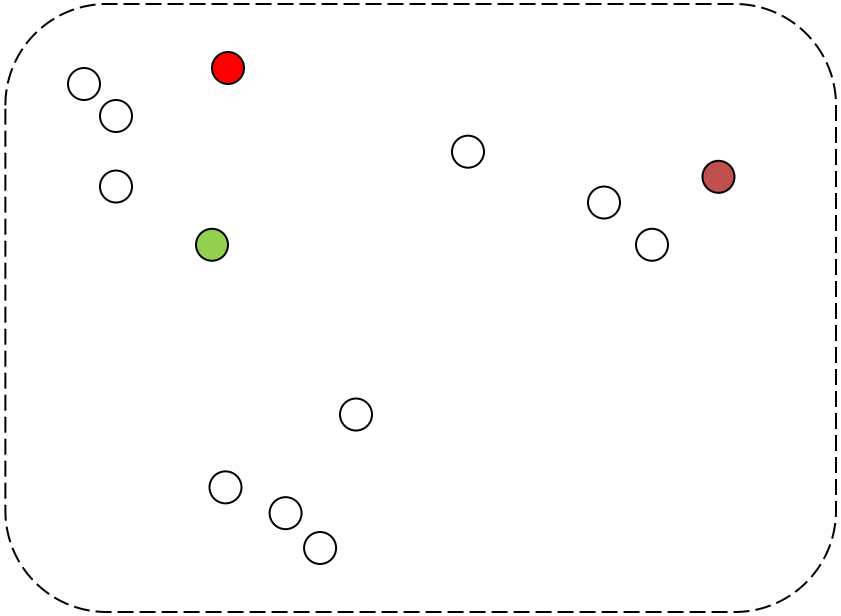
# Learning Paradigms
## Unsupervised Learning

- *K*-means clustering
    - Initialize *K* “means” randomly (=cluster centroids)



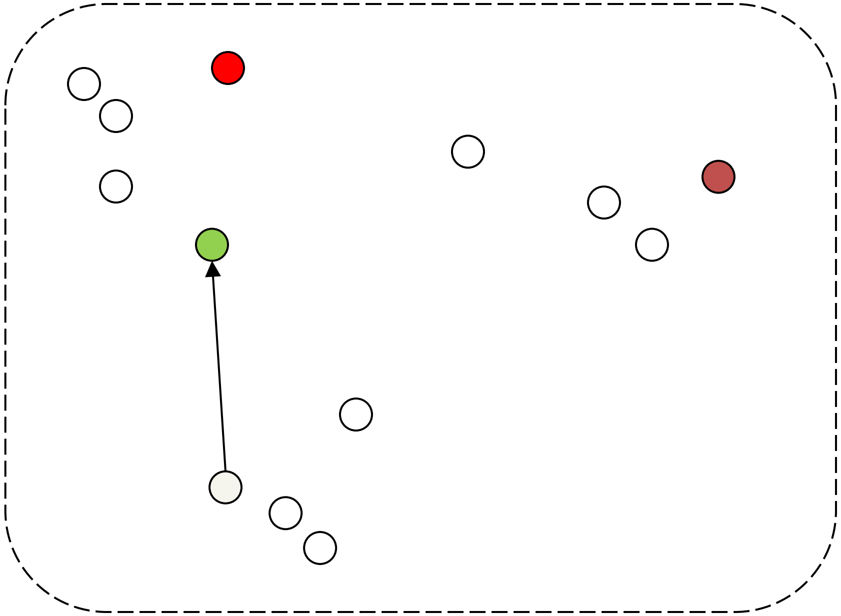
# Learning Paradigms
## Unsupervised Learning

- *K*-means clustering
  - $K$=3



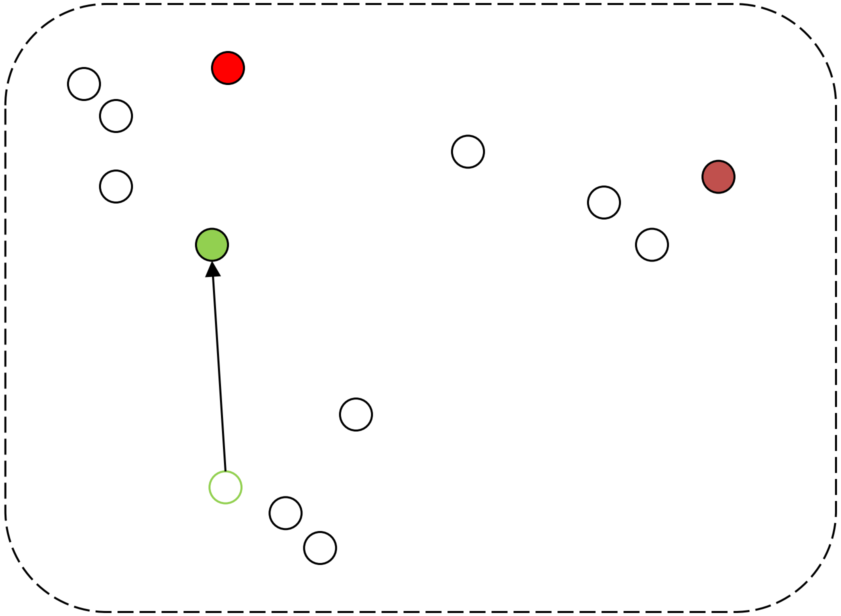
# Learning Paradigms
## Unsupervised Learning

- *K*-means clustering
  - Assignment: assign each data point to its closest mean



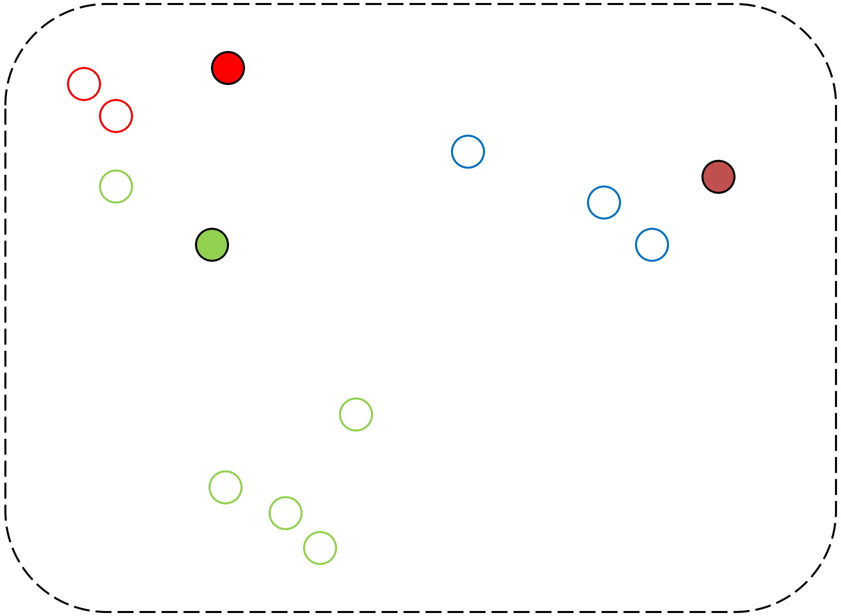
# Learning Paradigms
## Unsupervised Learning

- *K*-means clustering
  - Assignment: assign each data point to its closest mean



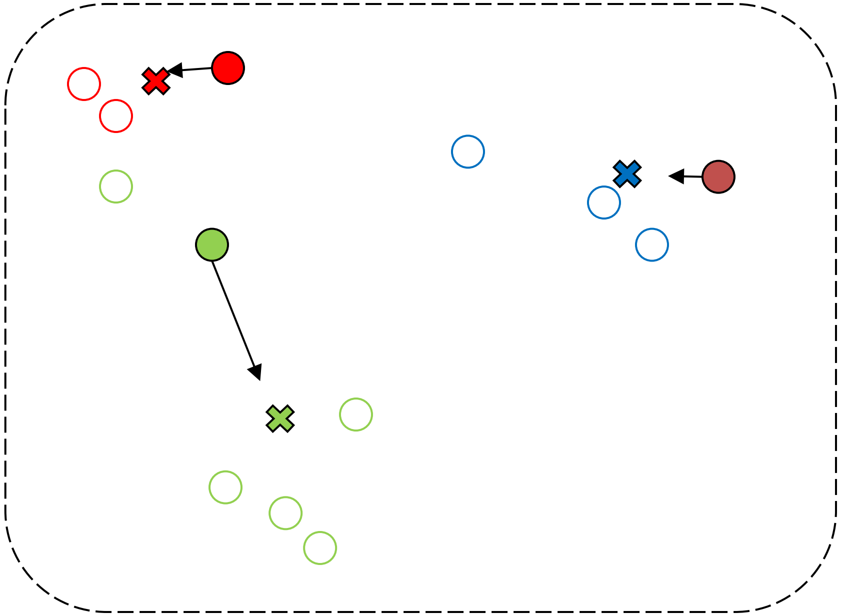
# Learning Paradigms
## Unsupervised Learning

- *K*-means clustering
  - Assignment: assign each data point to its closest mean



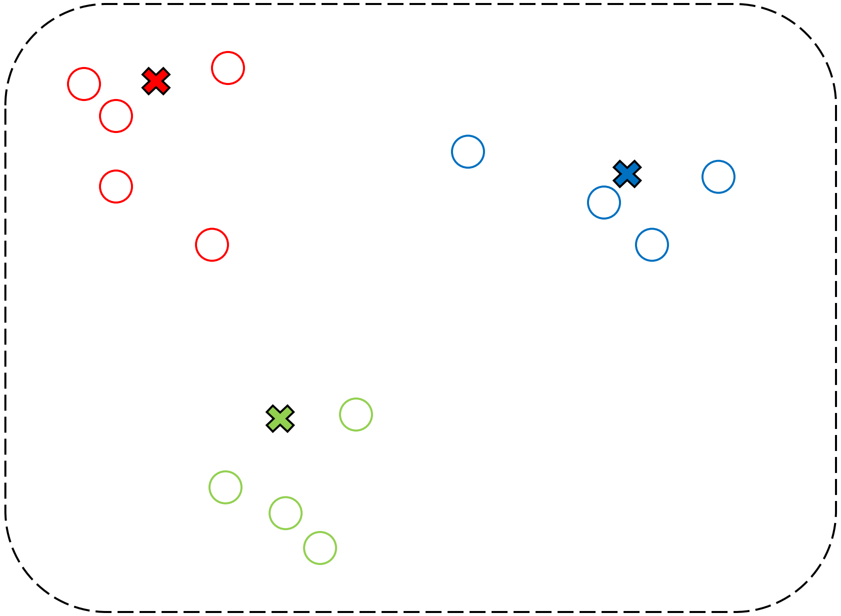
# Learning Paradigms
## Unsupervised Learning

- *K*-means clustering
    - Update: update mean by average over all assigned data points



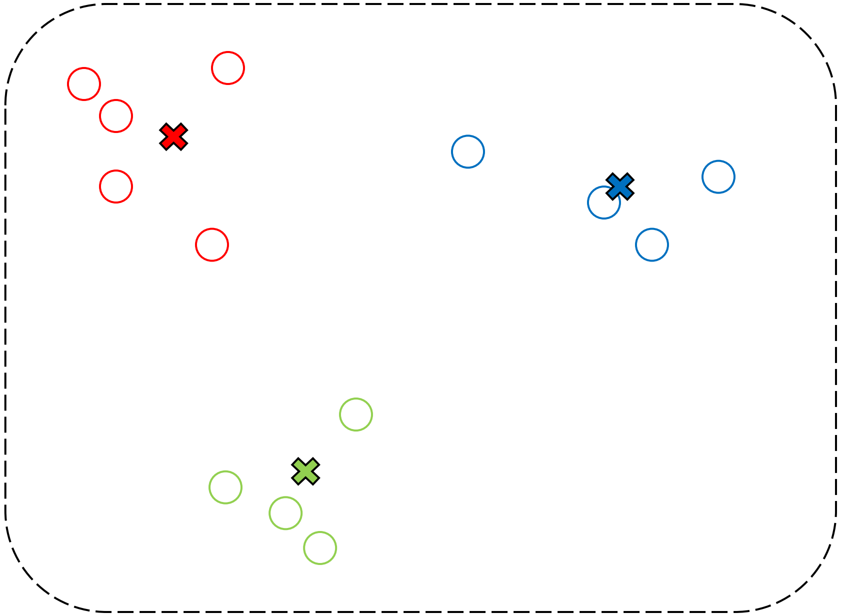
# Learning Paradigms
## Unsupervised Learning

- *K*-means clustering
  - Assignment: re-assign data points to closest mean



# Learning Paradigms
## Unsupervised Learning

- *K*-means clustering
  - Update: re-assign data points to closest mean (repeat until convergence)



# Learning Paradigms
## Supervised Learning

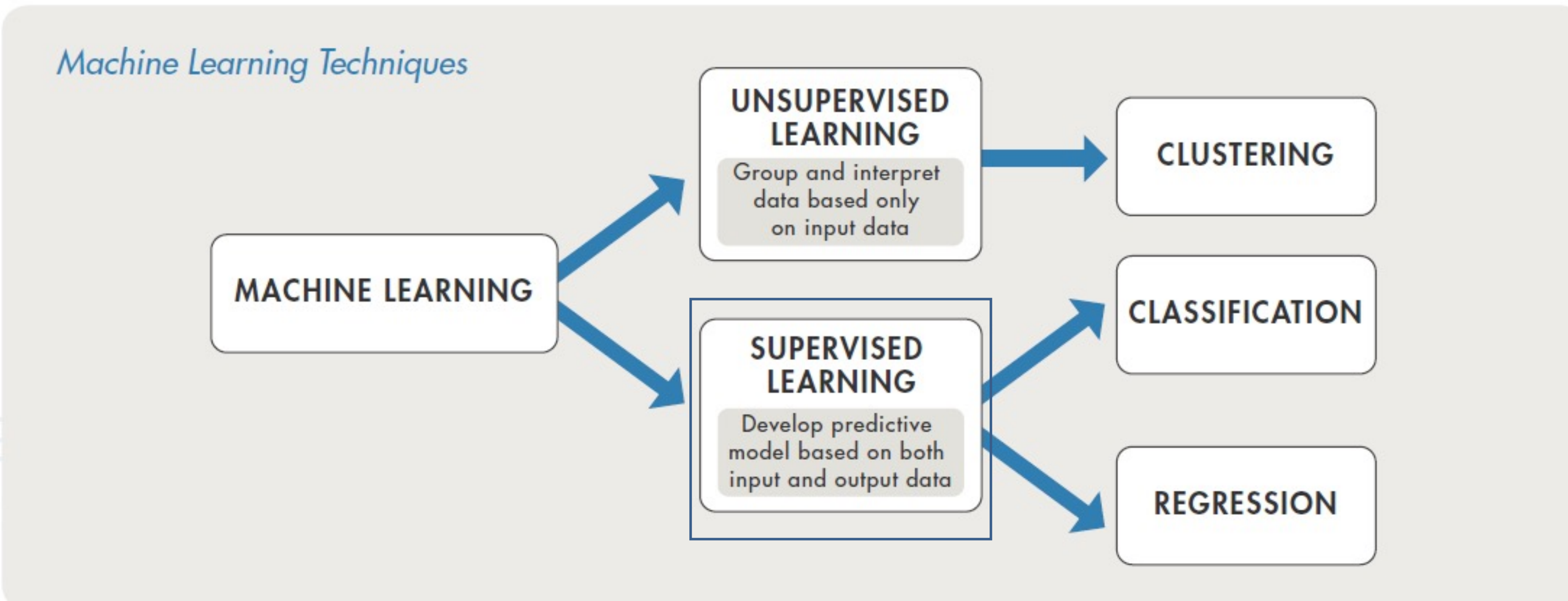



Fig. 1

[2]

# Learning Paradigms
## Supervised Learning

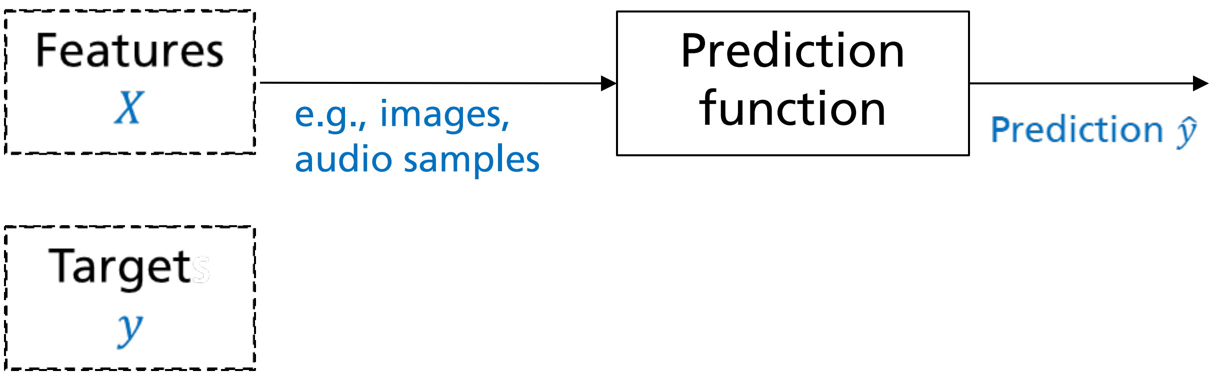
Features
$X$

e.g., images,
audio samples

Targets
$y$

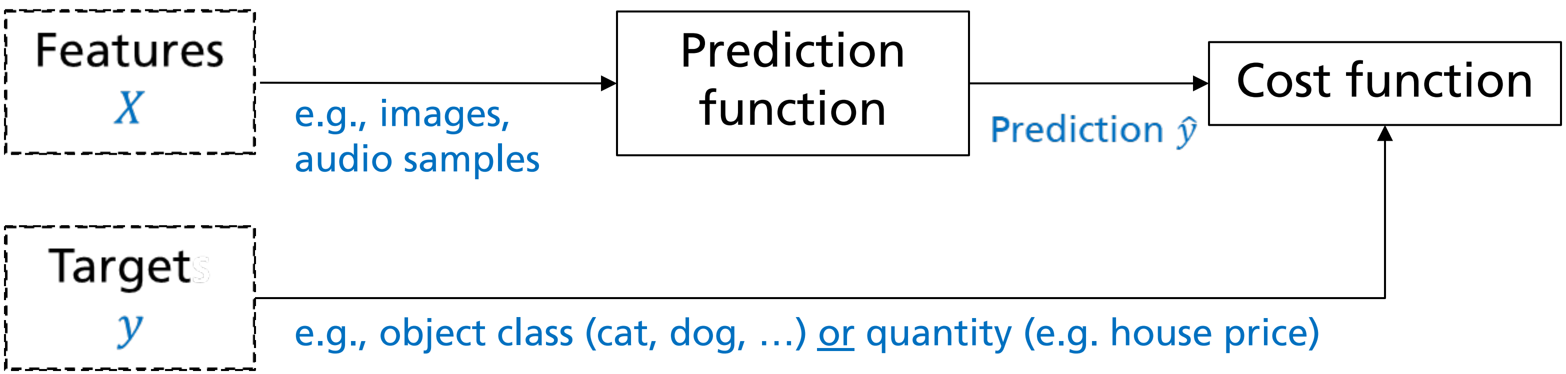
Own

# Learning Paradigms
## Supervised Learning



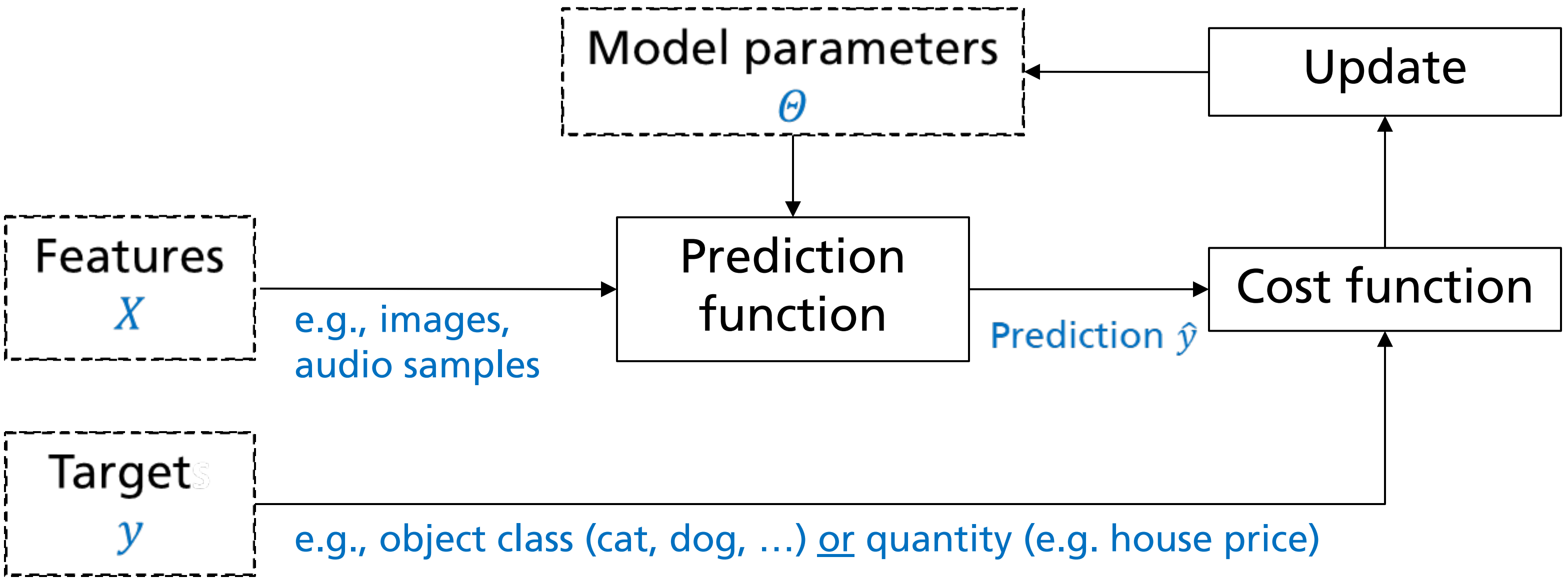

Own

# Learning Paradigms
## Supervised Learning




Own

# Learning Paradigms
## Supervised Learning




Own

# Learning Paradigms
## Supervised Learning - Classification

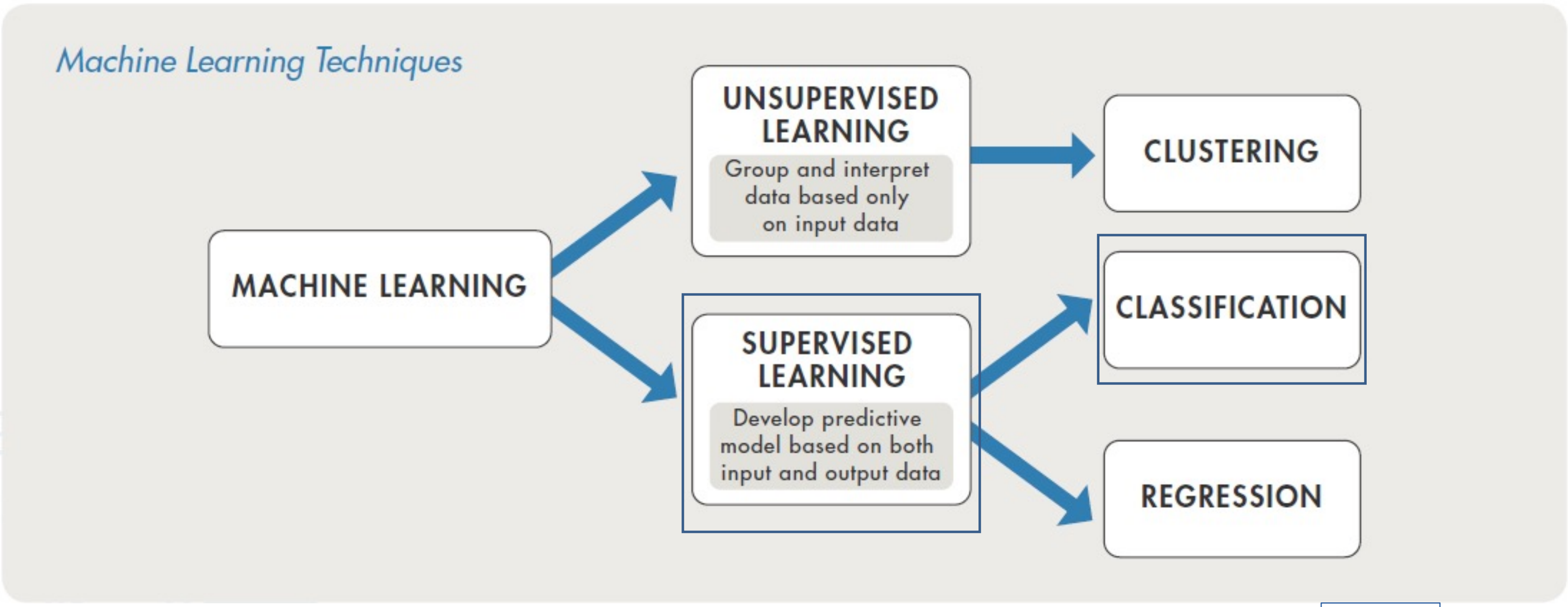



Fig. 1

[2]

# Learning Paradigms
## Supervised Learning - Classification

- Predict one or multiple categorical labels from features
  - Examples → music genre, instrument(s), key

# Learning Paradigms
## Supervised Learning - Classification

- Predict one or multiple categorical labels from features
    - Examples → music genre, instrument(s), key
- Feature space modeling (Example: 2 classes)

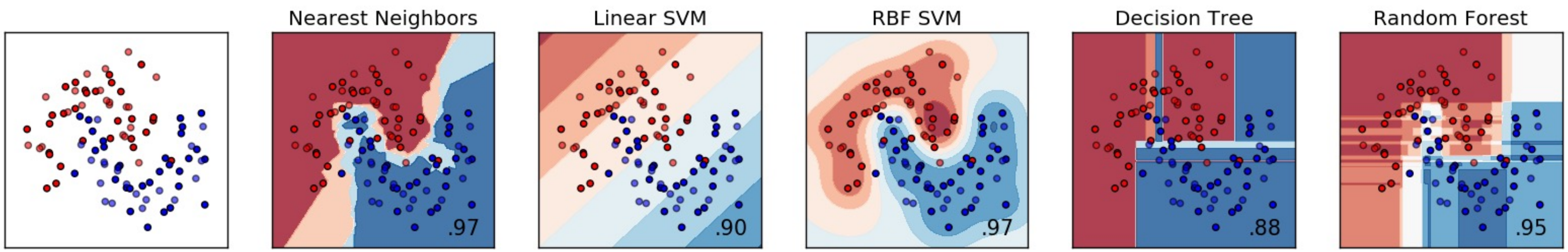



Fig. 3

# Learning Paradigms
## Supervised Learning - Classification

- Predict one or multiple categorical labels from features
    - Examples → music genre, instrument(s), key
- Feature space modeling (Example: 2 classes)

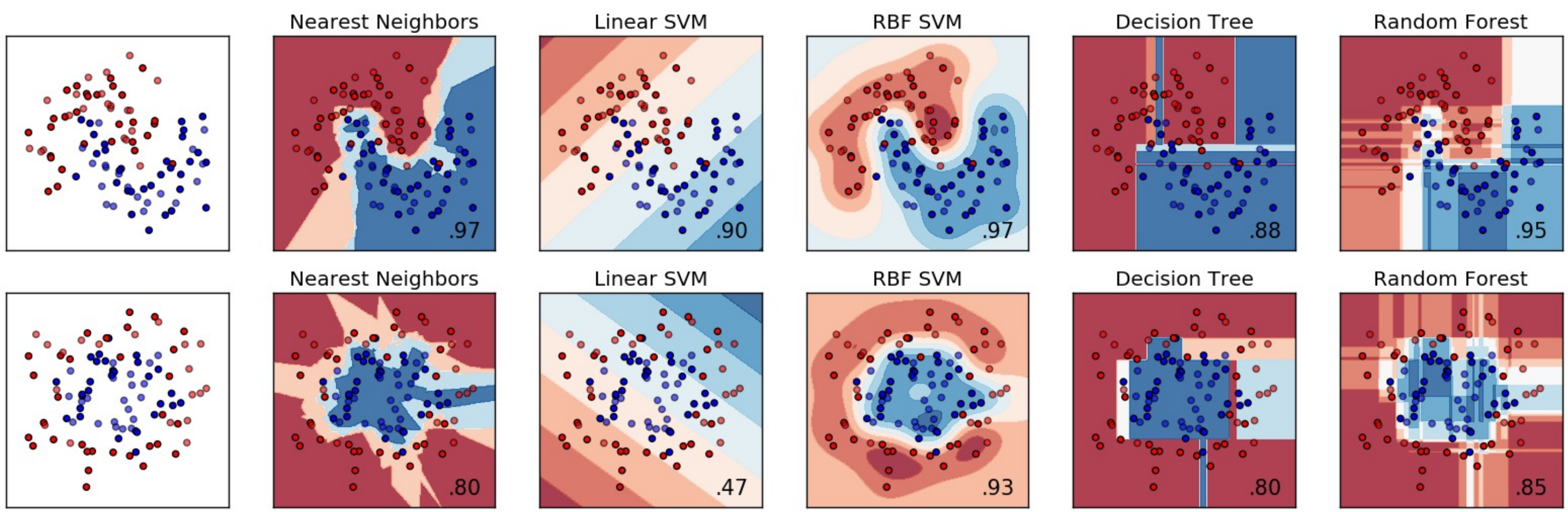



Fig. 3

# Learning Paradigms
## Supervised Learning - Classification

- Example: *k*-Nearest Neighbors
  - Training → Store all examples

# Learning Paradigms
## Supervised Learning - Classification

- Example: *k*-Nearest Neighbors
    - Training → Store all examples

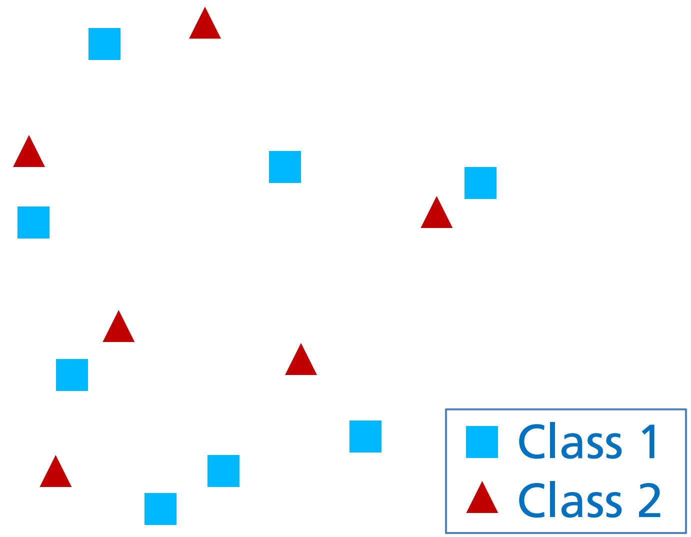



Fig. 4

# Learning Paradigms
## Supervised Learning - Classification

- Example: *k*-Nearest Neighbors
  - Training → Store all examples
  - Test → Assign test item to dominant class label of the *k* clostest training data items

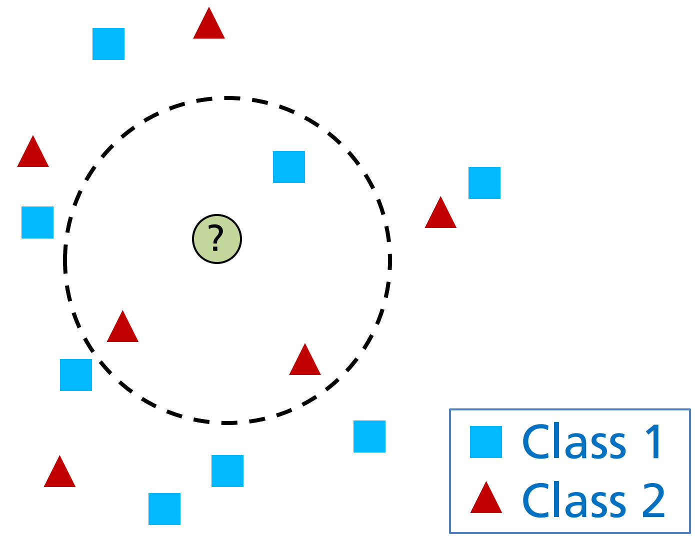


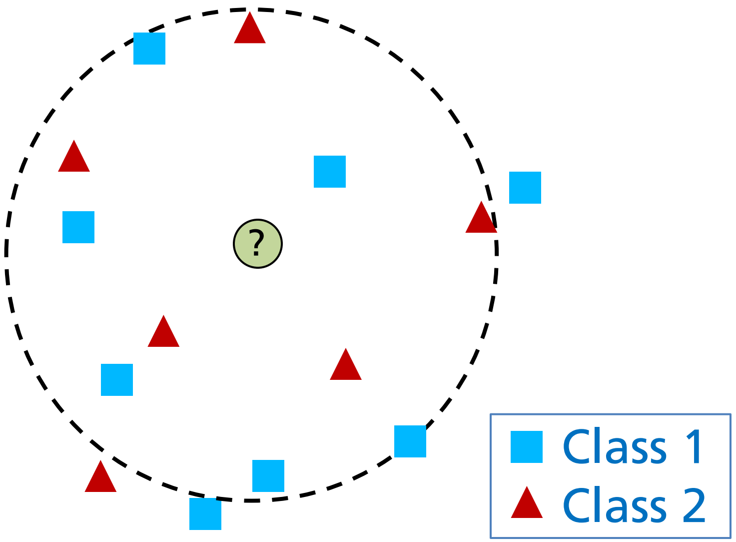

Fig. 4

$k = 3 \rightarrow$ ? = ▲

# Learning Paradigms

## Supervised Learning - Classification

- Example: *k*-Nearest Neighbors
  - Training → Store all examples
  - Test → Assign test item to dominant class label of the *k* clostest training data items




$k = 11 \rightarrow$ ? = ■

Fig. 4

# Learning Paradigms
## Supervised Learning - Classification

- Example: *k*-Nearest Neighbors
  - Training → Store all examples
  - Test → Assign test item to dominant class label of the *k* clostest training data items
- Distance measures
  - Euclidean distance, Manhatten distance, cosine distance, …

# Learning Paradigms
## Supervised Learning

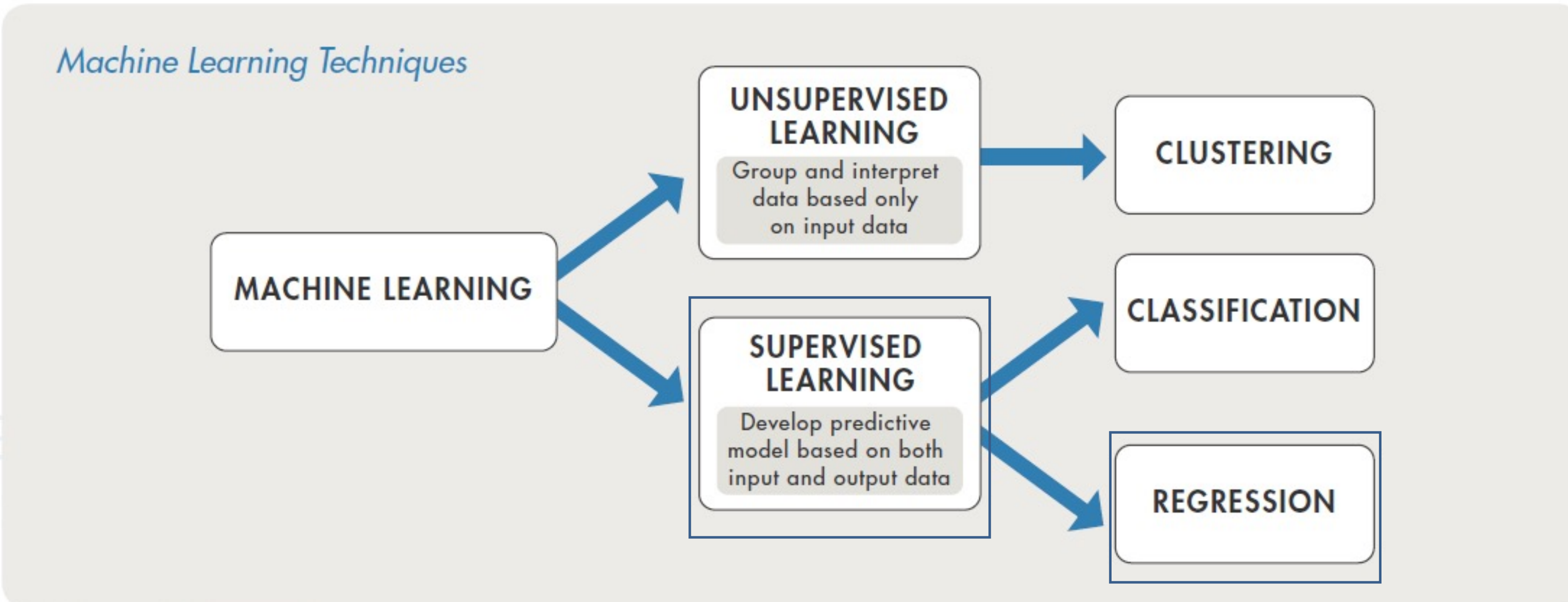


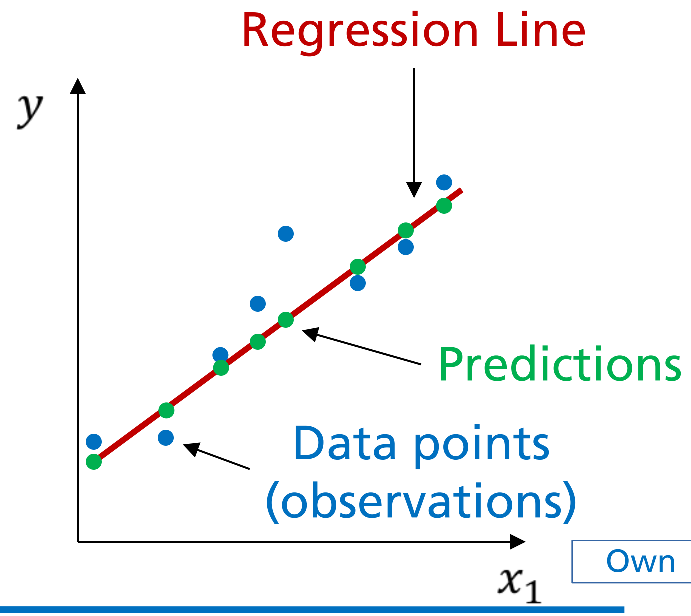

Fig. 1

# Learning Paradigms
## Supervised Learning - Regression

- Goal
  - Predict a dependent (response) variable given one or multiple independent variables (features)
  - Continuous quantities
- Examples
  - Univariate (linear) regression:
    - $y \approx \beta_0 + \beta_1 x_1$
      - $\beta_0 \rightarrow$ bias
      - $\beta_1 \rightarrow$ weight

# Learning Paradigms
## Supervised Learning - Regression

- Goal
    - Predict a dependent (response) variable given one or multiple independent variables (features)
    - Continuous quantities
- Examples
    - Univariate (linear) regression:
        - $y \approx \beta_0 + \beta_1 x_1$
            - $\beta_0 \rightarrow$ bias
            - $\beta_1 \rightarrow$ weight




Own

# ML Project Pipeline
## Overview

Data Collection & Pre-Processing

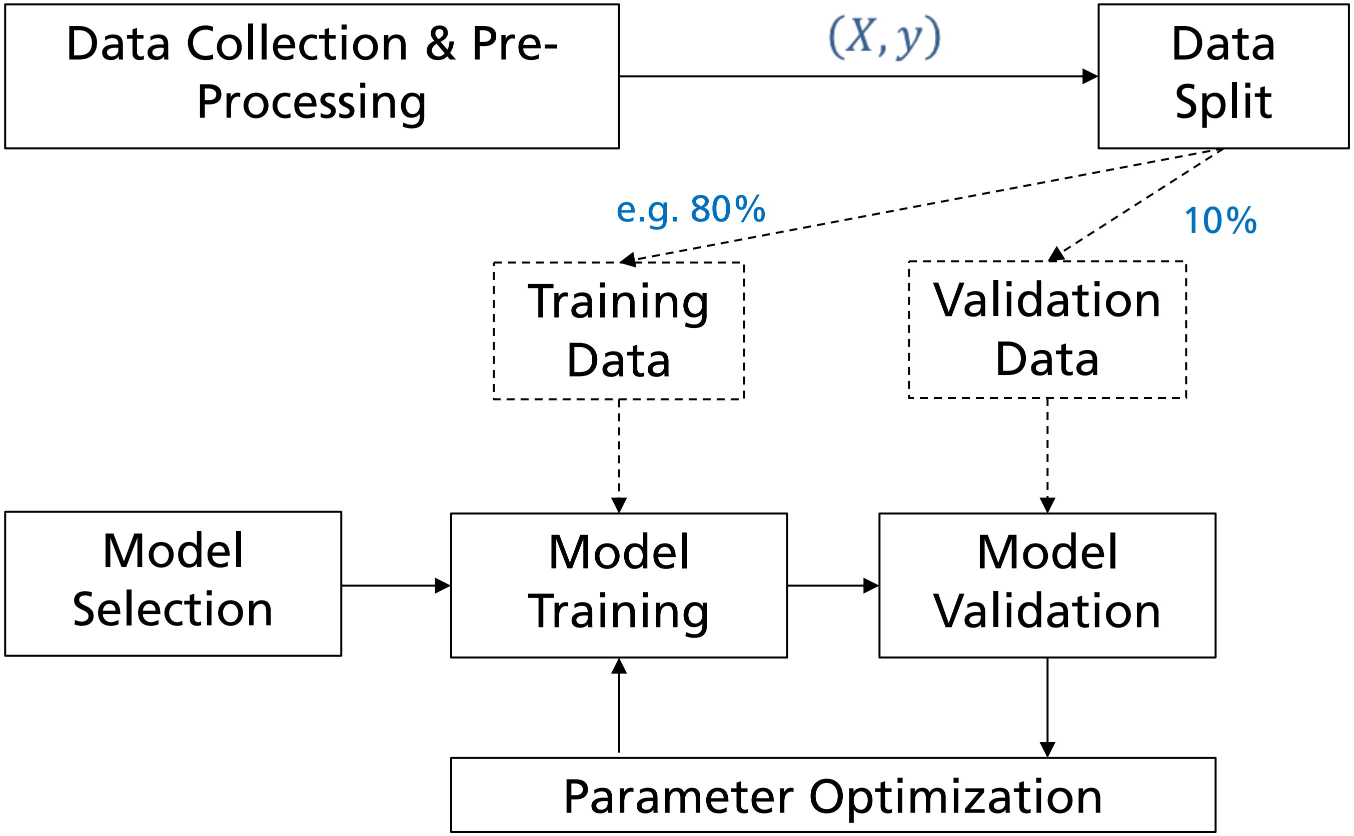
Own

# ML Project Pipeline
## Overview

Data Collection & Pre-Processing $\xrightarrow{(X, y)}$ Data Split

Own

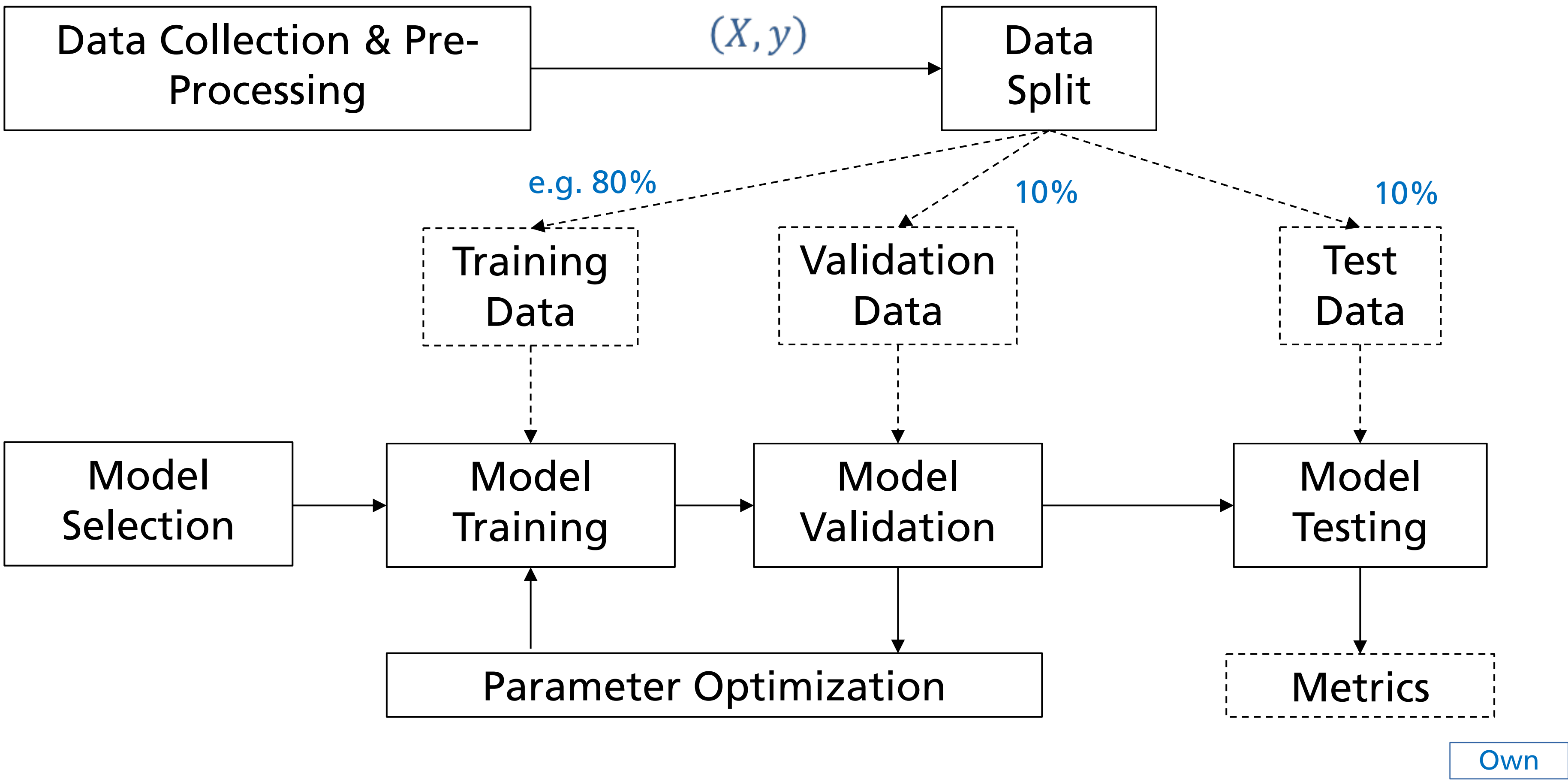
# ML Project Pipeline
## Overview




Own

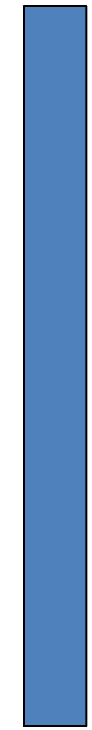
# ML Project Pipeline

## Overview



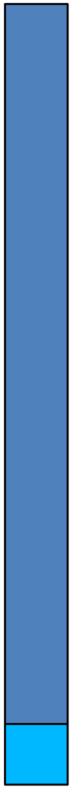
# ML Project Pipeline
## Data Split

- Training Set
  - Model learns from this data



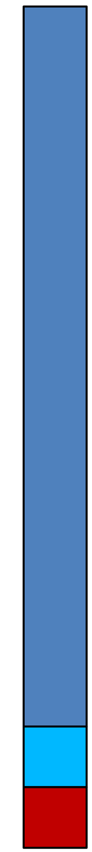
# ML Project Pipeline
## Data Split

- Training Set
  - Model learns from this data
- Validation / Development Set
  - Used to fine-tune the model (hyper)parameters
  - Model occasionally sees but does not learn from this data



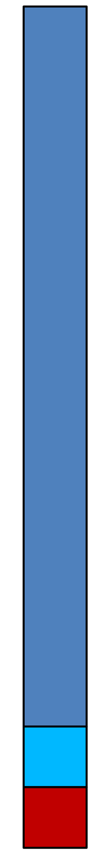
# ML Project Pipeline
## Data Split

- Training Set
  - Model learns from this data
- Validation / Development Set
  - Used to fine-tune the model (hyper)parameters
  - Model occasionally sees but does not learn from this data
- Test set
  - Only used once after the model training & tuning is completed
  - Should reflect the targeted real-world use case for the model



# ML Project Pipeline
## Data Split

- Training Set
  - Model learns from this data
- Validation / Development Set
  - Used to fine-tune the model (hyper)parameters
  - Model occasionally sees but does not learn from this data
- Test set
  - Only used once after the model training & tuning is completed
  - Should reflect the targeted real-world use case for the model
- Common split ratios
  - 80/10/10%  or even 98/1/1% (for large datasets)



# ML Project Pipeline
## Data Collection & Pre-Processing

- Data collection
  - Check for available data resources for given (or related) task
  - Collect / record / annotate new data (if necessary)
  - Ensure data variability
    - Example (from acoustic condition monitoring) → include different motor engine types & conditions, recording locations, microphones, …

# ML Project Pipeline
## Data Collection & Pre-Processing

- Data collection
  - Check for available data resources for given (or related) task
  - Collect / record / annotate new data (if necessary)
  - Ensure data variability
    - Example (from acoustic condition monitoring) → include different motor engine types & conditions, recording locations, microphones, …
- Data cleanup / pre-processing
  - Remove errors, silence, empty files, …
  - Balance dataset (proportions among class examples)
  - Normalize (depends on the model)

# ML Project Pipeline
## Model Selection

- Model Types (SVM, GMM, logistic regression, DNNs)
- Hyperparameters (SVM kernel functions, DNN layer types)

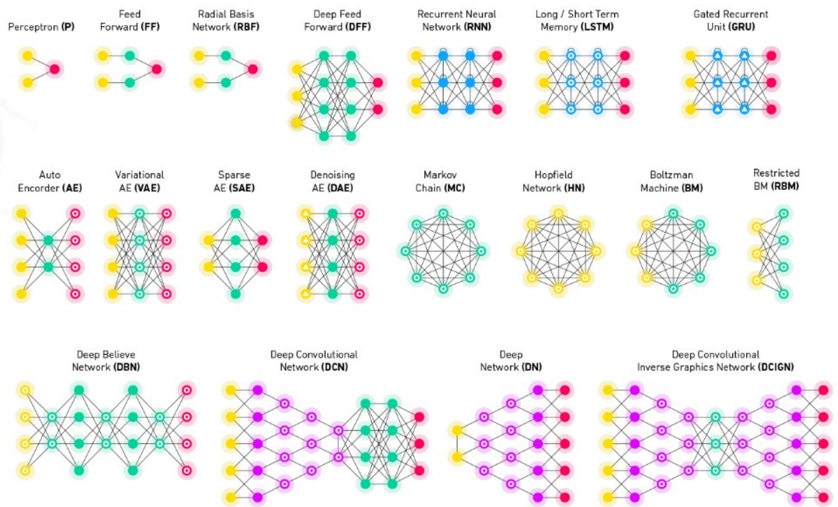



Fig. 6

# ML Project Pipeline
## Model Selection

- Model Types (SVM, GMM, logistic regression, DNNs)
- Hyperparameters (SVM kernel functions, DNN layer types)
- Constraints from application scenario
  - Model complexity (memory, training time, training data amount)

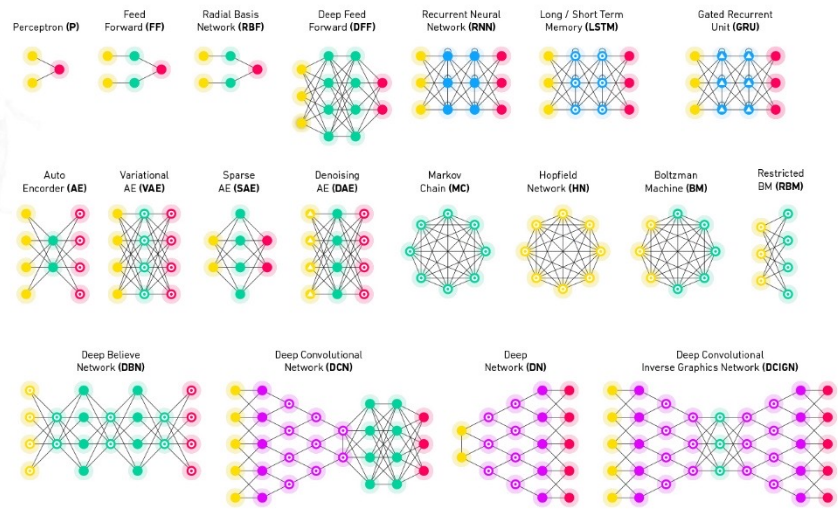



Fig. 6

# ML Project Pipeline
## Model Selection

- Model Types (SVM, GMM, logistic regression, DNNs)
- Hyperparameters (SVM kernel functions, DNN layer types)
- Constraints from application scenario
    - Model complexity (memory, training time, training data amount)
- Feature pre-processing depends on model type
- Use simple models for simple tasks

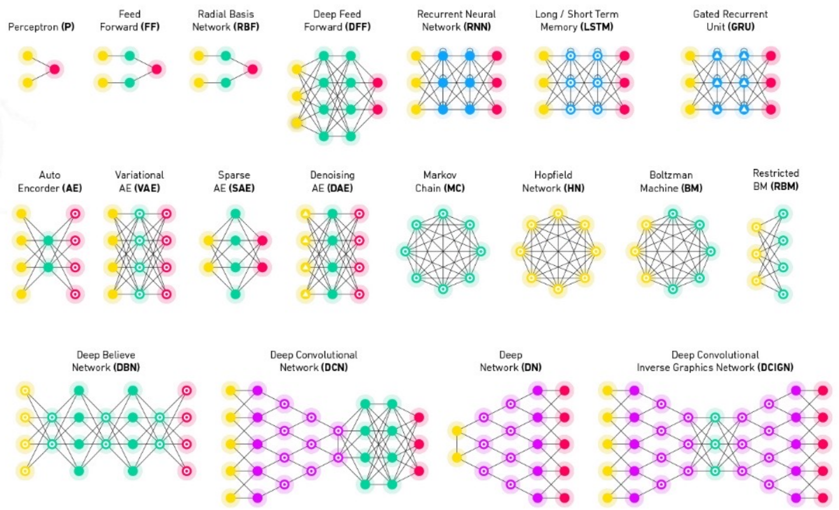



Fig. 6

# ML Project Pipeline
## Model Training

- Iterative process
  - Typically: start with random parameter initialization

# ML Project Pipeline
## Model Training

- Iterative process
  - Typically: start with random parameter initialization
  - Use (batches of) training data to iteratively improve model predictions (optimization)
    - Learn from examples

# ML Project Pipeline
## Model Training

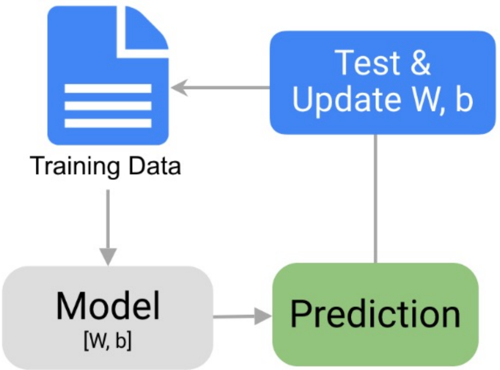
- Iterative process
    - Typically: start with random parameter initialization
    - Use (batches of) training data to iteratively improve model predictions (optimization)
        - Learn from examples
    - Update model parameters according to loss function

# ML Project Pipeline
## Model Training

- Example: linear regression

$$y \approx \beta_0 + \beta_1\, x_1$$

- Training loop




Own

# ML Project Pipeline
## Model Validation

- Regular model evaluation each or multiple training iteration

# ML Project Pipeline
## Model Validation

- Regular model evaluation each or multiple training iteration

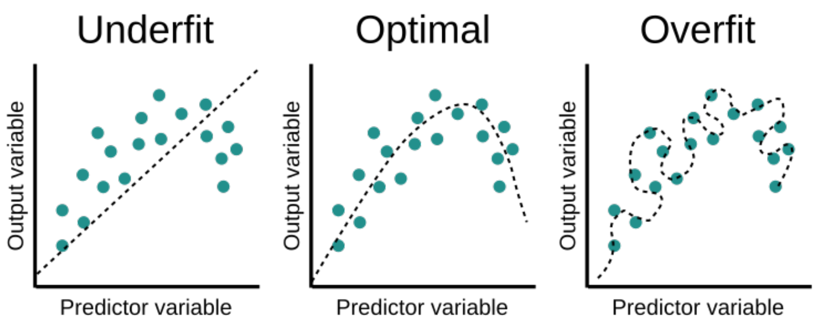



Fig. 7

# ML Project Pipeline
## Model Validation

- Regular model evaluation each or multiple training iteration
- Helps to
  - optimize model (hyper)parameters
  - detect overfitting on training data
  - stop the training

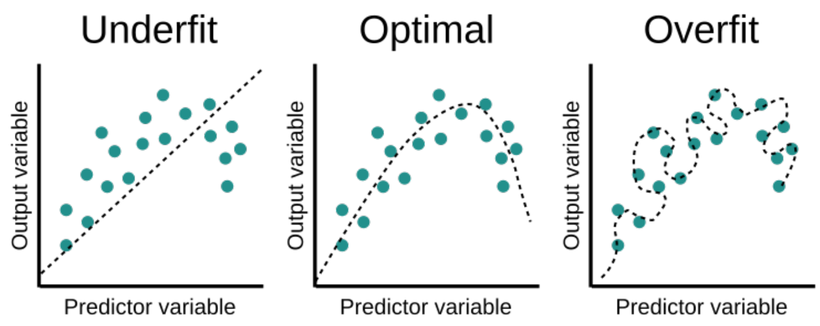
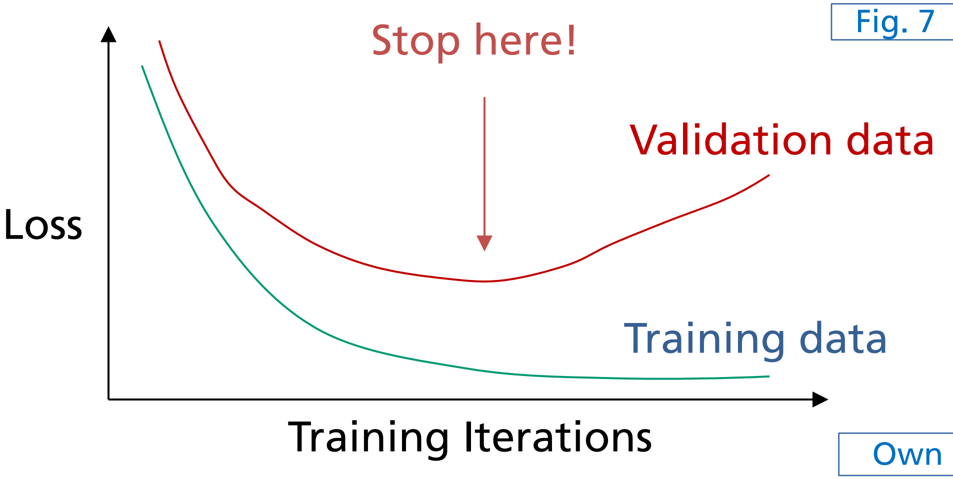



Fig. 7




Own

# ML Project Pipeline
## Model Testing

- Example: Binary classification evaluation
    - True/false positives (TP/FP)
    - True/false negatives (TN/FN)

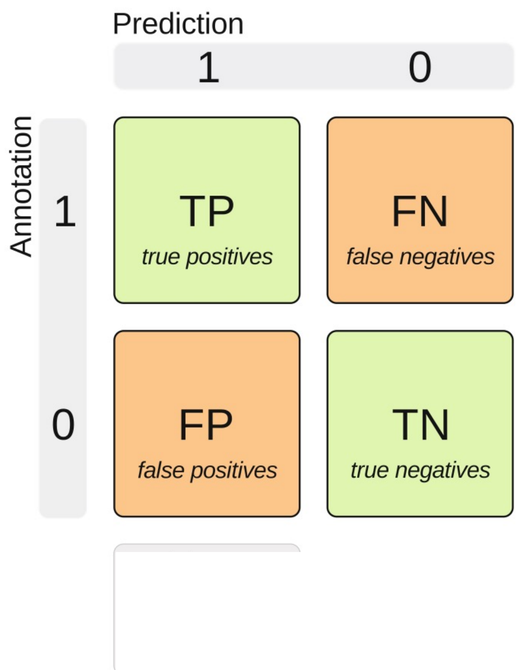



Fig. 8

# ML Project Pipeline
## Model Testing

- Example: Binary classification evaluation
  - True/false positives (TP/FP)
  - True/false negatives (TN/FN)
  - Metrics
    - Precision
    - Recall
    - Accuracy
    - F-score

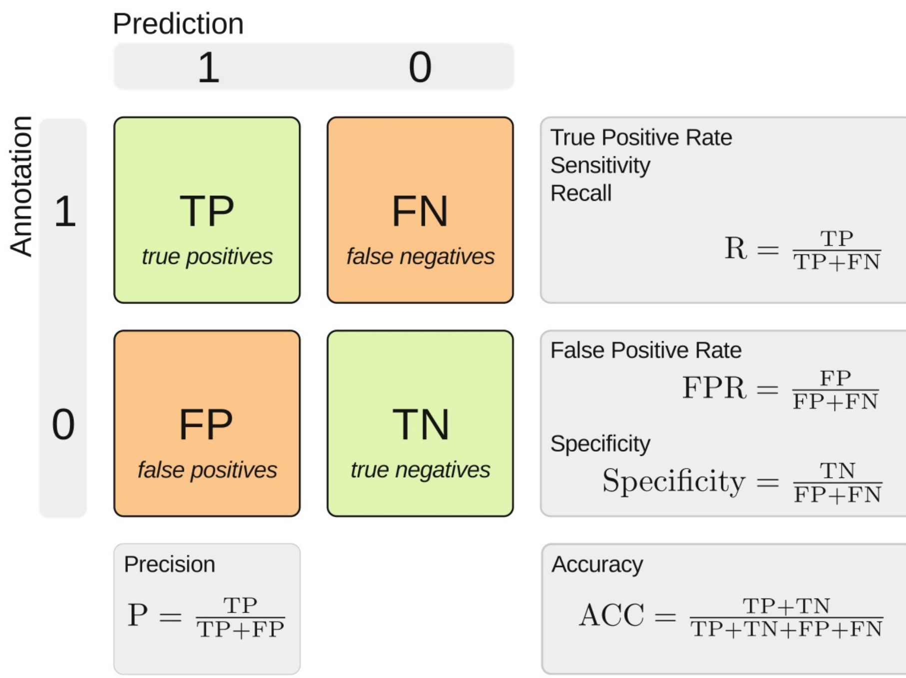



Fig. 8

# Deep Learning
## Introduction

- Artificial neural networks → mimic brain processing
  - Connected neurons
  - Weighted input summation
  - Non-linear processing

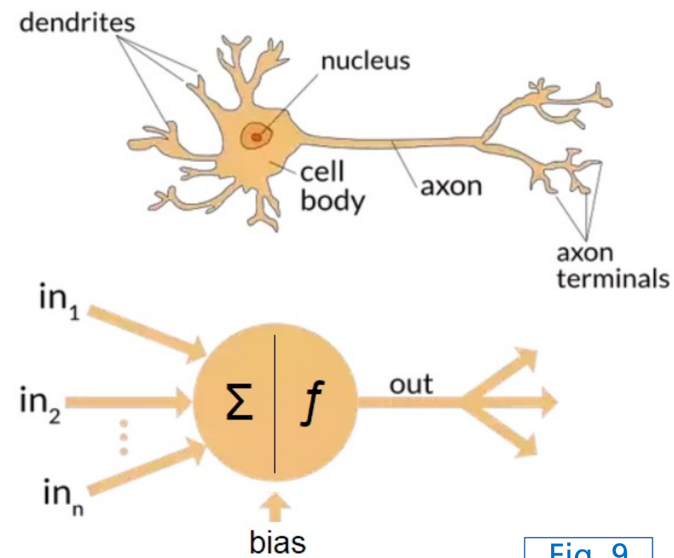



Fig. 9

# Deep Learning
## Introduction

- Artificial neural networks → mimic brain processing
  - Connected neurons
  - Weighted input summation
  - Non-linear processing
- Shallow networks

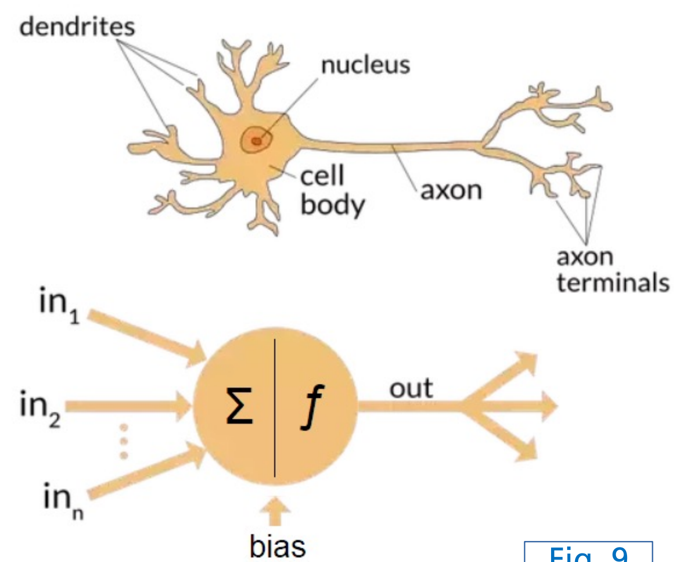



Fig. 9

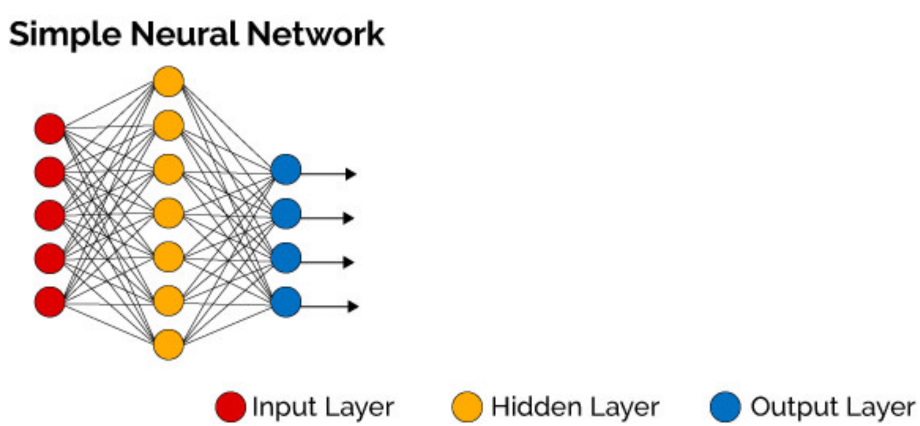



Fig. 10

# Deep Learning
## Introduction

- Artificial neural networks → mimic brain processing
  - Connected neurons
  - Weighted input summation
  - Non-linear processing
- Shallow networks → deep networks

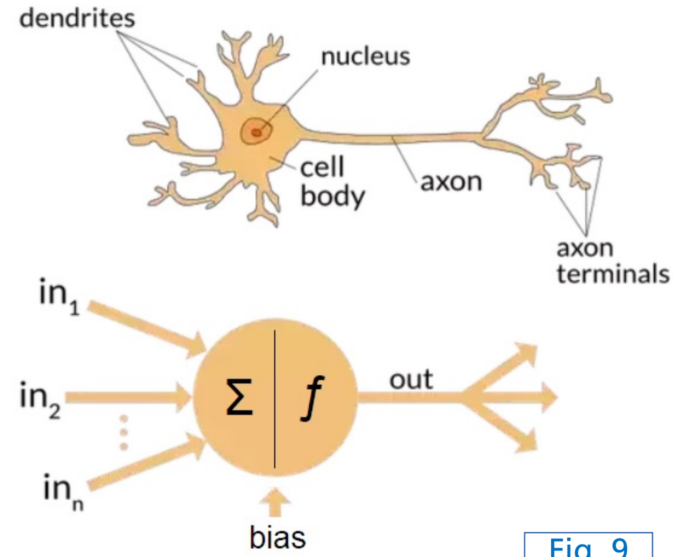



Fig. 9

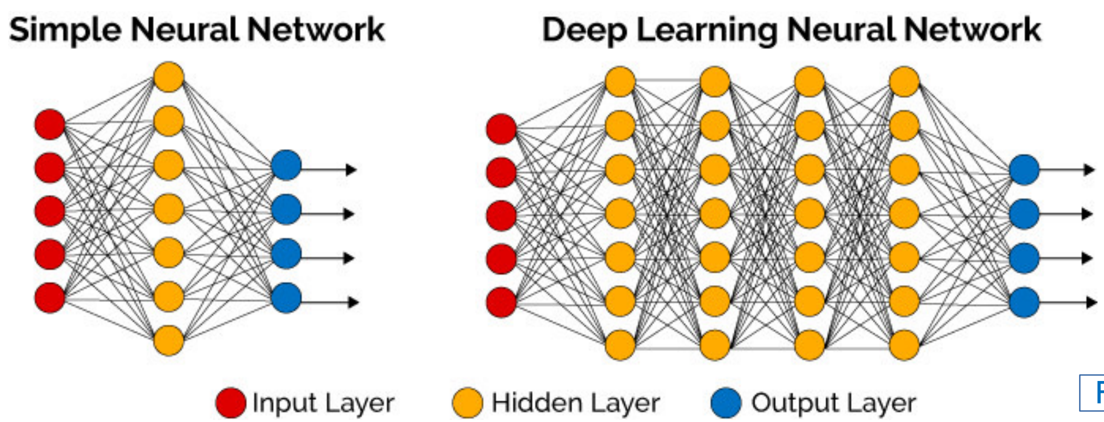
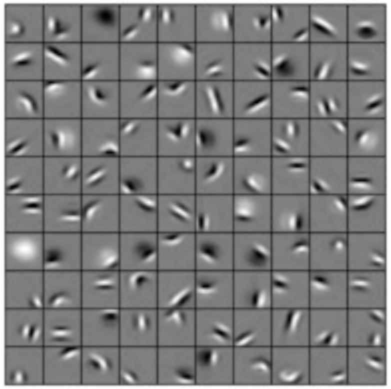



Fig. 10

# Deep Learning
## Introduction

- Hierarchical feature learning
  - Example (face recognition)



Edges, curves

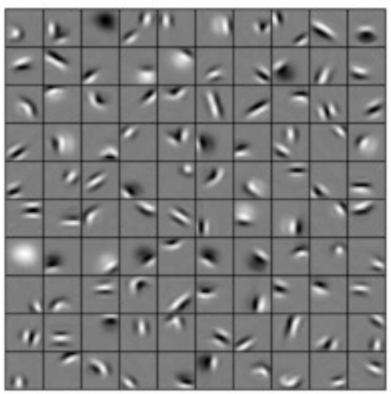
Fig. 11

First layers

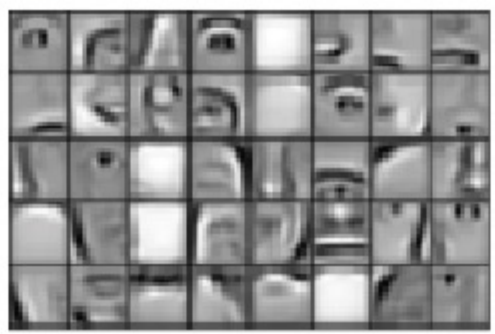
Final layers

# Deep Learning
## Introduction

- Hierarchical feature learning
  - Example (face recognition)



Edges, curves



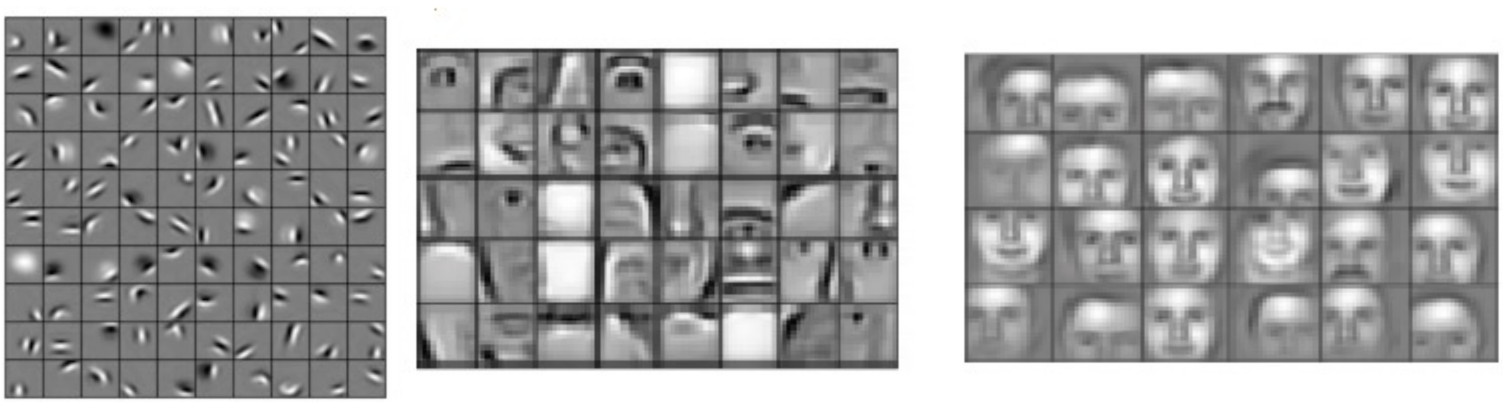
Shapes, object parts

Fig. 11

First layers → Final layers

# Deep Learning
## Introduction

- Hierarchical feature learning
    - Example (face recognition)

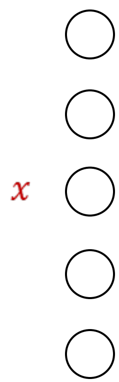


Edges, curves | Shapes, object parts | Objects (faces) | Fig. 11

First layers → Final layers

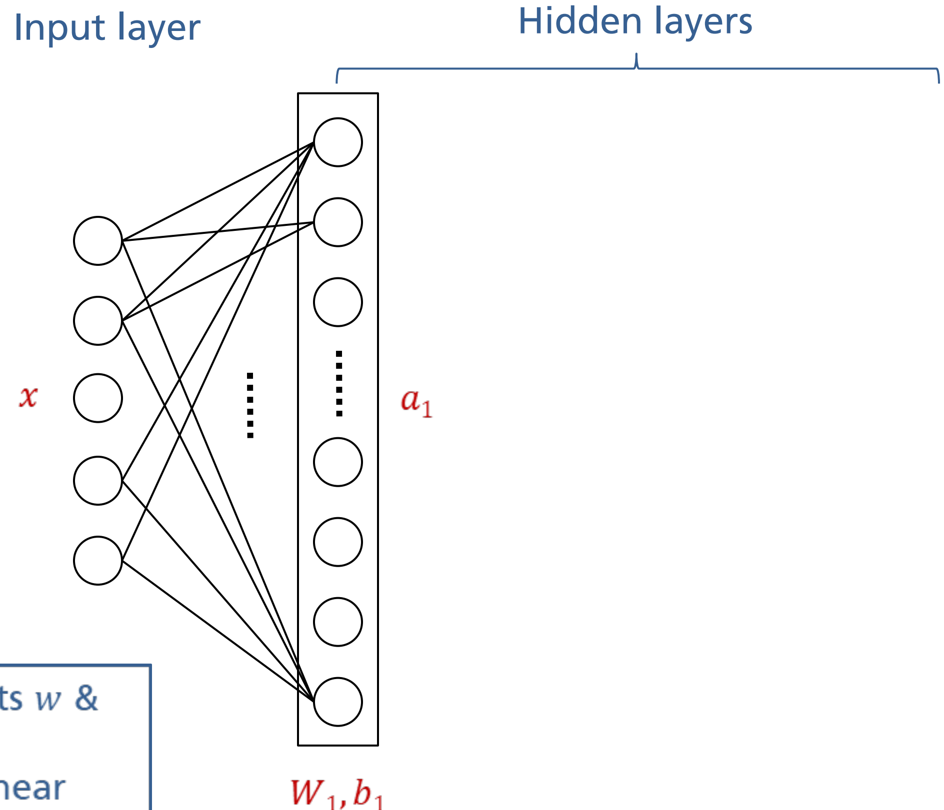
# Deep Learning
## Fully-connected (Deep) Neural Networks

Input layer




Own

# Deep Learning
## Fully-connected (Deep) Neural Networks



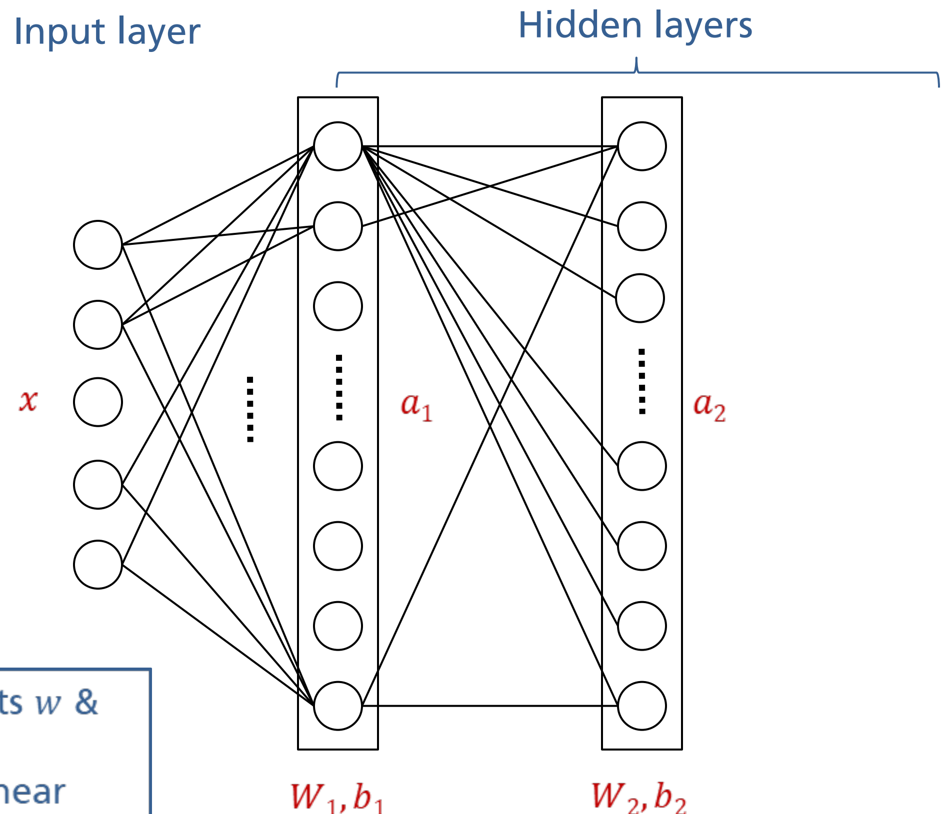

1) Weights $w$ & biases $b$
2) Non-linear activation function $g(z)$

$$z_1 = W_1 x + b_1$$
$$a_1 = g(z_1)$$

Own

# Deep Learning

## Fully-connected (Deep) Neural Networks



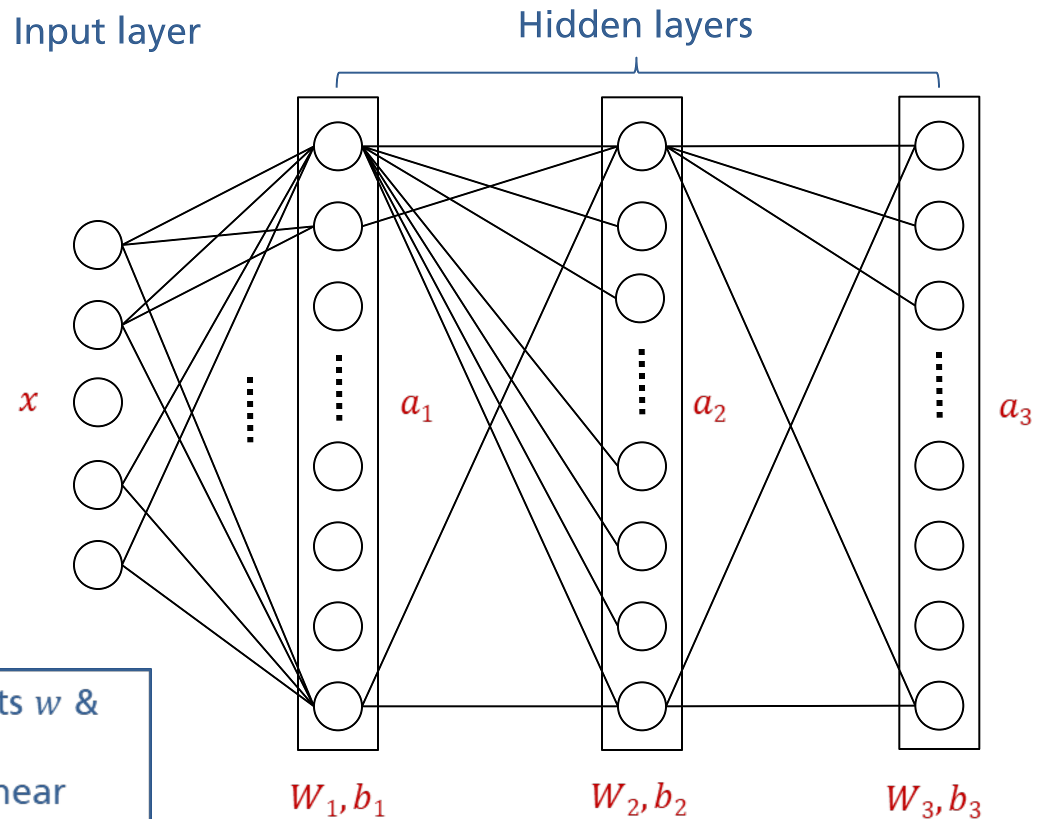

$z_1 = W_1 x + b_1$
$a_1 = g(z_1)$

$z_2 = W_2 a_1 + b_2$
$a_2 = g(z_2)$

1) Weights $w$ & biases $b$
2) Non-linear activation function $g(z)$

Own

# Deep Learning
## Fully-connected (Deep) Neural Networks




$z_1 = W_1 x + b_1$
$a_1 = g(z_1)$

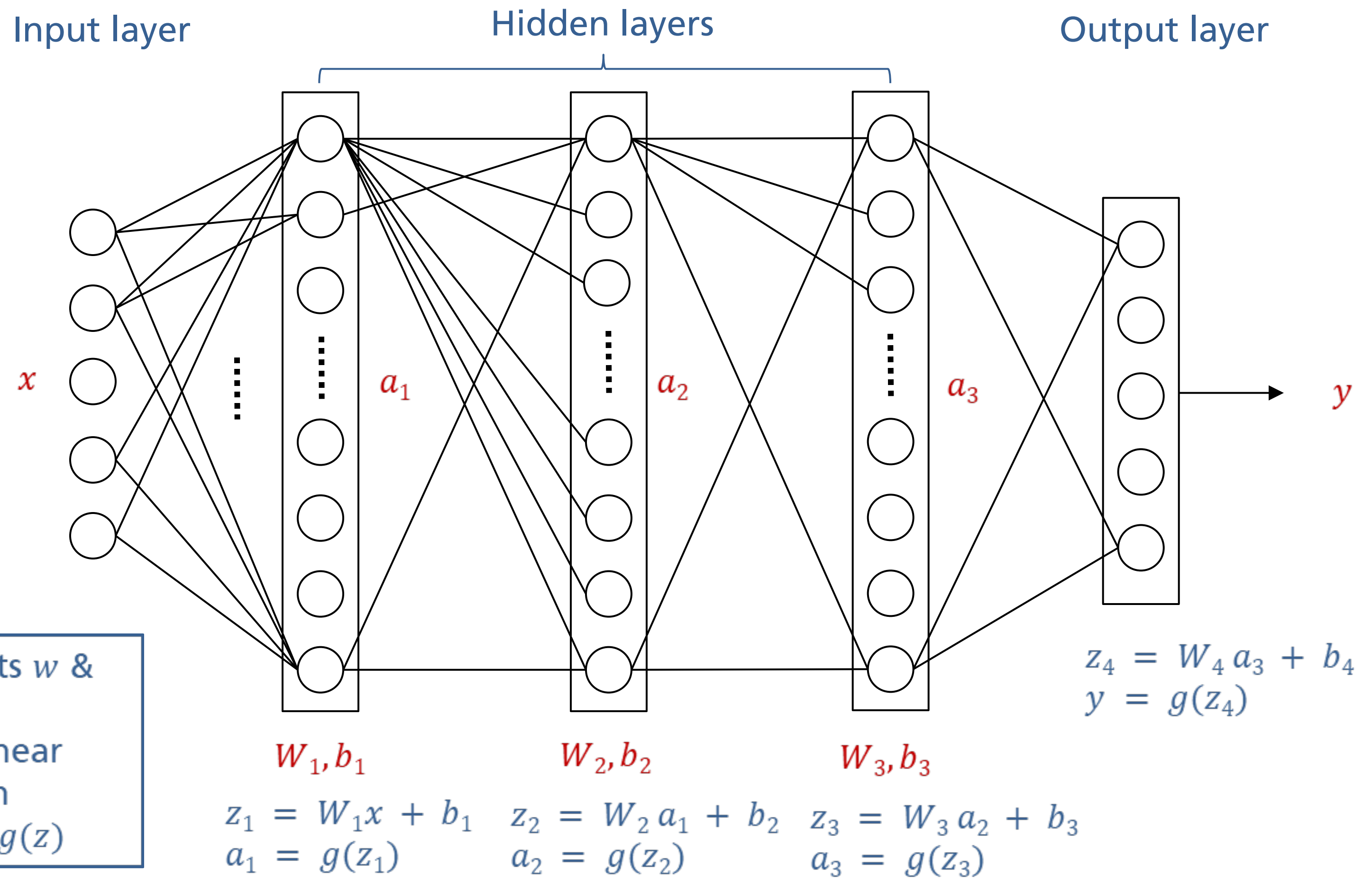
$z_2 = W_2 a_1 + b_2$
$a_2 = g(z_2)$

$z_3 = W_3 a_2 + b_3$
$a_3 = g(z_3)$

1) Weights $w$ & biases $b$
2) Non-linear activation function $g(z)$

Own

# Deep Learning
## Fully-connected (Deep) Neural Networks




1) Weights $w$ & biases $b$
2) Non-linear activation function $g(z)$

Own

# Deep Learning
## Activation Functions

- Activation functions add non-linearity
- Make networks more powerful in (complex) pattern recognition
- Examples:

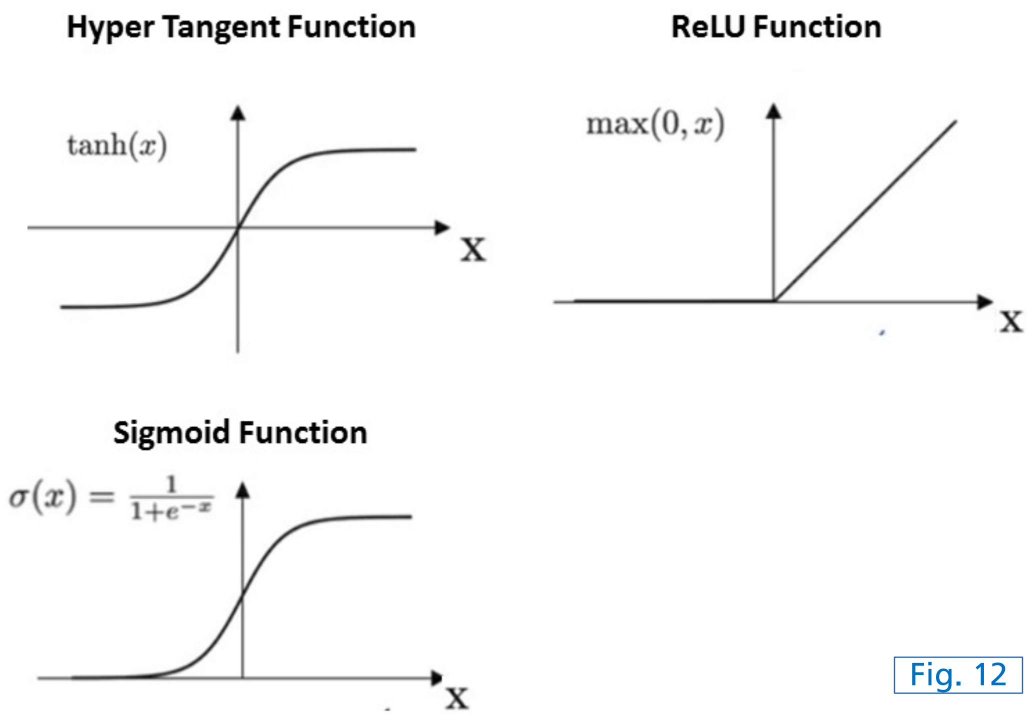



Fig. 12

# Deep Learning
## Training

- Overview

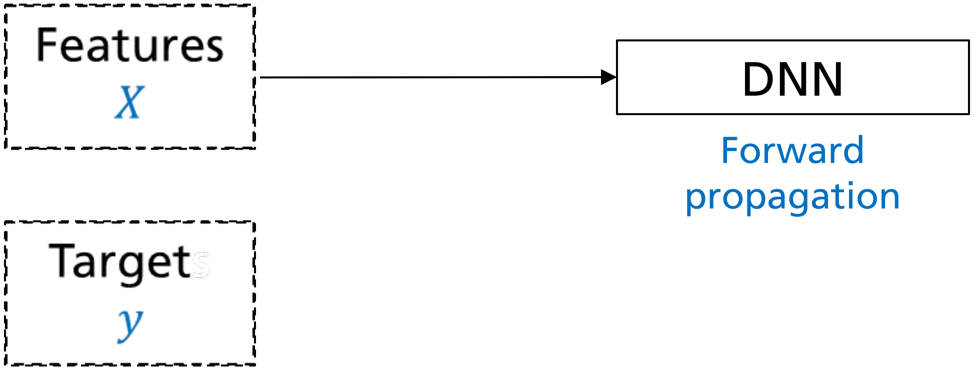
Features
$X$

Targets
$y$

Own

# Deep Learning
## Training

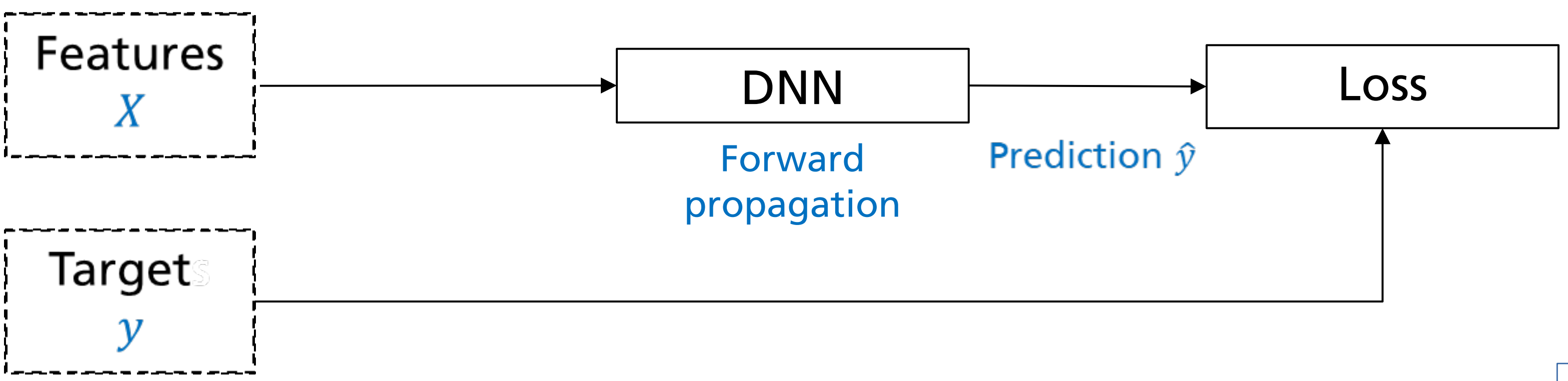
- Overview




Own

# Deep Learning
## Training

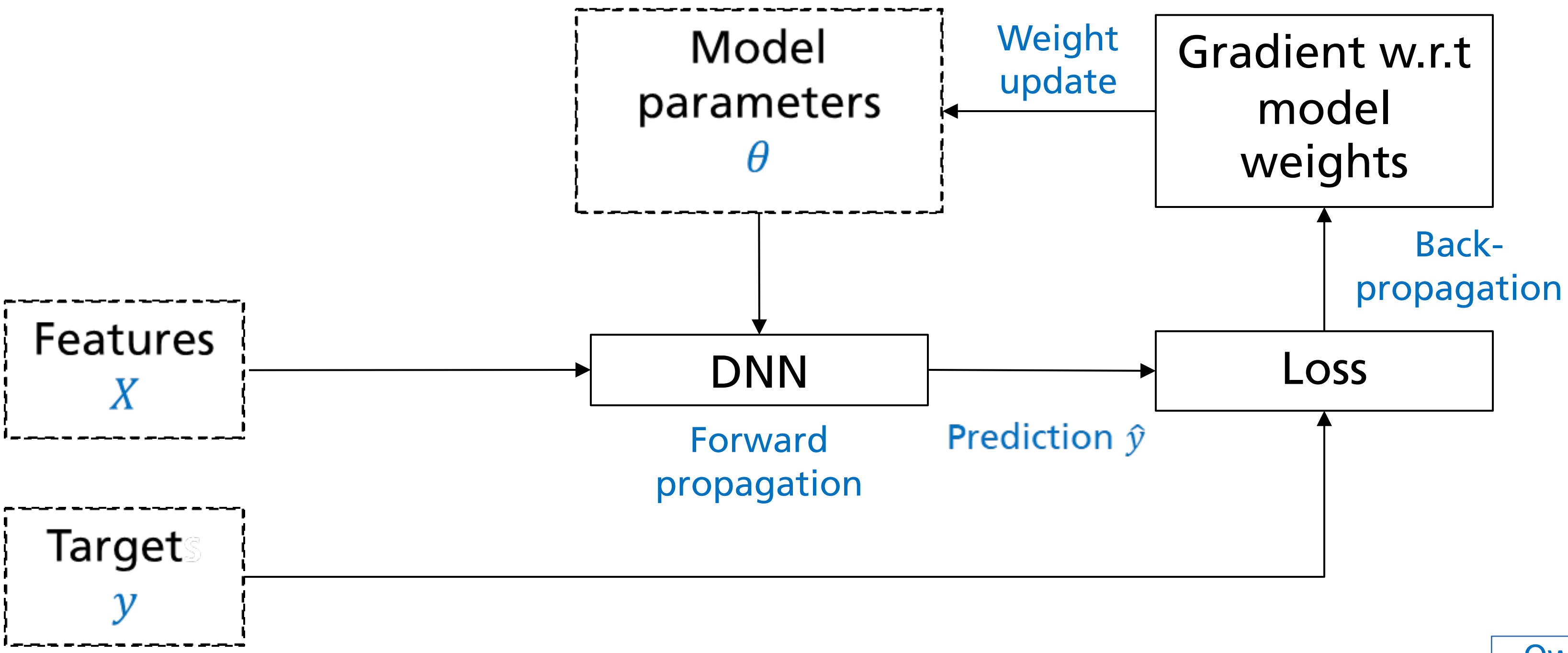
- Overview




Own

# Deep Learning
## Training

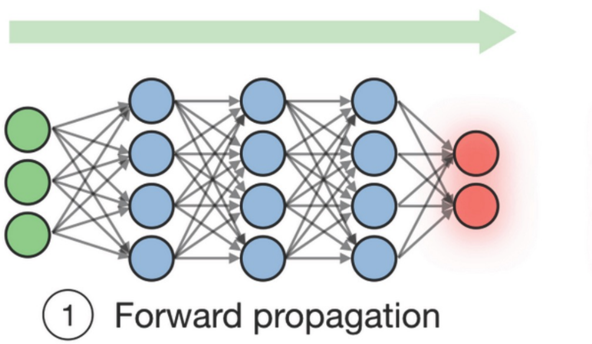
- Overview




Own

# Deep Learning
## Training

- Forward propagation → propagate batch of training data through the network → compute loss (compare to targets)

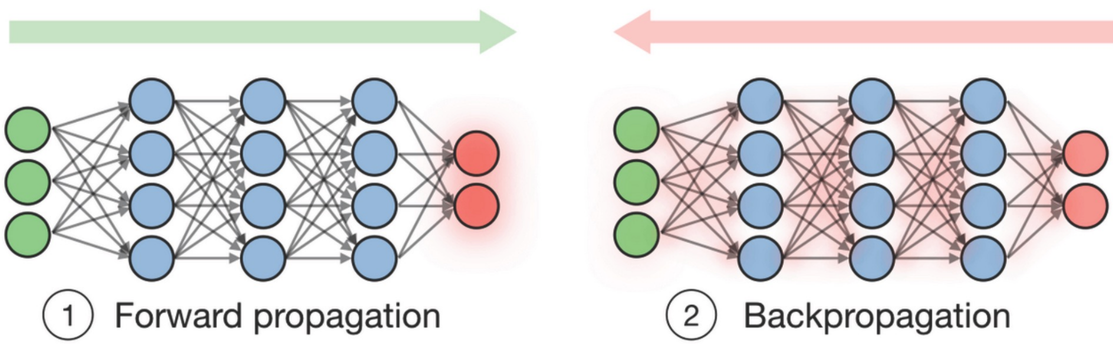



$$\boxed{L(z,y) = -\Big[y\log(z) + (1-y)\log(1-z)\Big]}$$

Fig. 20

# Deep Learning
## Training

- Forward propagation → propagate batch of training data through the network → compute loss (compare to targets)
- Backpropagation → backpropagate loss → compute gradients of loss w.r.t. weights



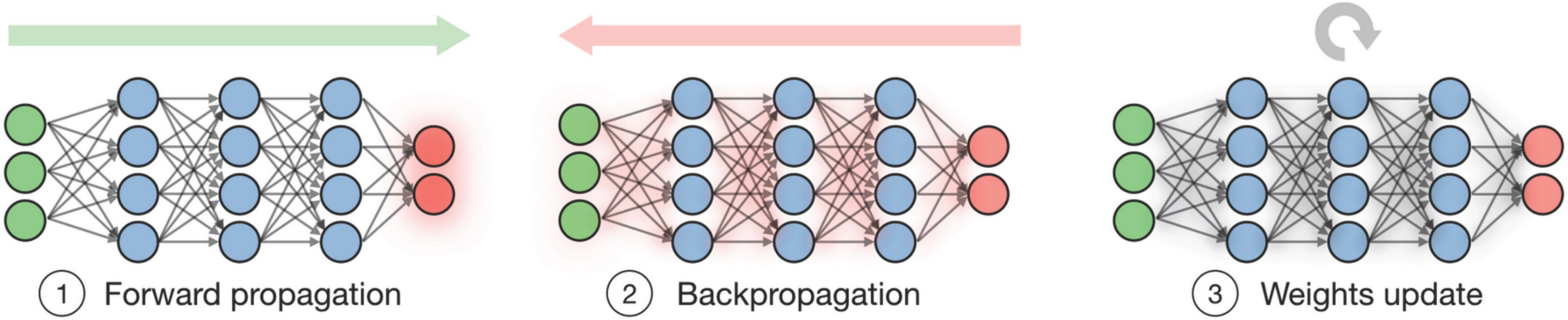

$$\boxed{L(z,y) = -\left[y\log(z) + (1-y)\log(1-z)\right]}$$

$$\boxed{\frac{\partial L(z,y)}{\partial w}}$$

Fig. 20

# Deep Learning
## Training

- Forward propagation → propagate batch of training data through the network → compute loss (compare to targets)
- Backpropagation → backpropagate loss → compute gradients of loss w.r.t. weights
- Weights update → use gradients & learning rate to update weights




$$\boxed{L(z,y) = -\Big[y\log(z) + (1-y)\log(1-z)\Big]}$$

$$\boxed{\frac{\partial L(z,y)}{\partial w}}$$

$$\boxed{w \longleftarrow w - \alpha\frac{\partial L(z,y)}{\partial w}}$$

Fig. 20

# Deep Learning
## Training

- Gradient descent
  - Move in opposite direction of gradient
  - Learning rate effects step size

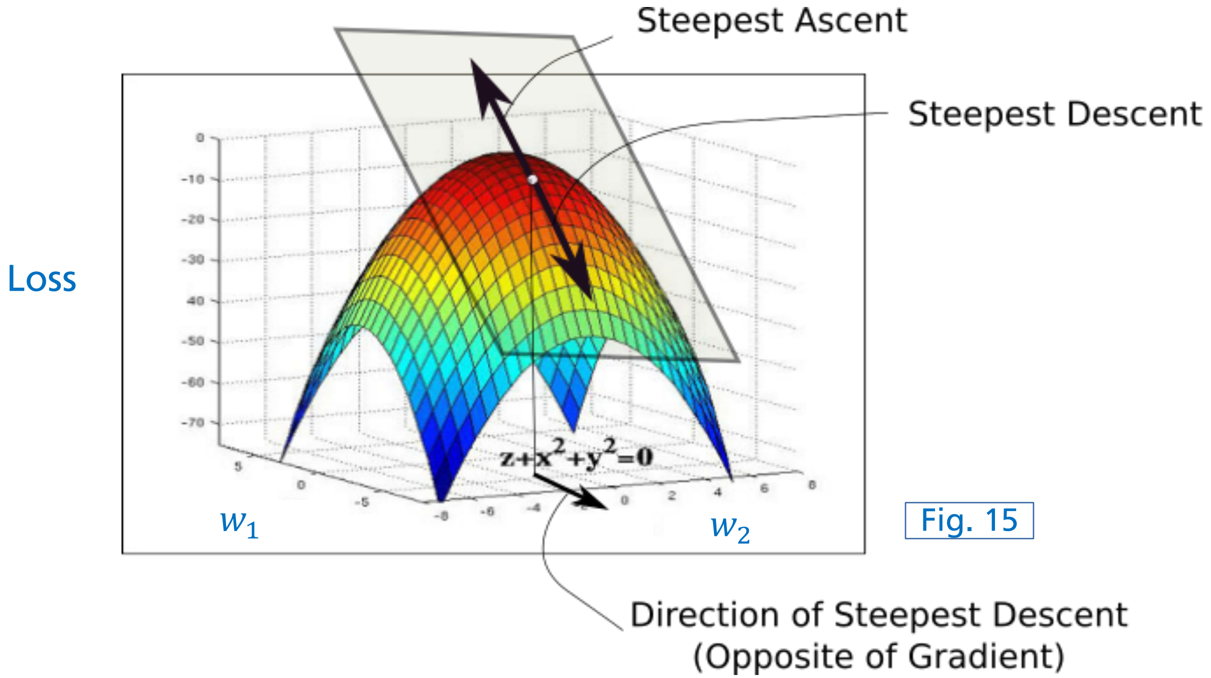



Fig. 15

# Deep Learning
## Training

- Loss contour
    - Goal → find global minima

Toy example

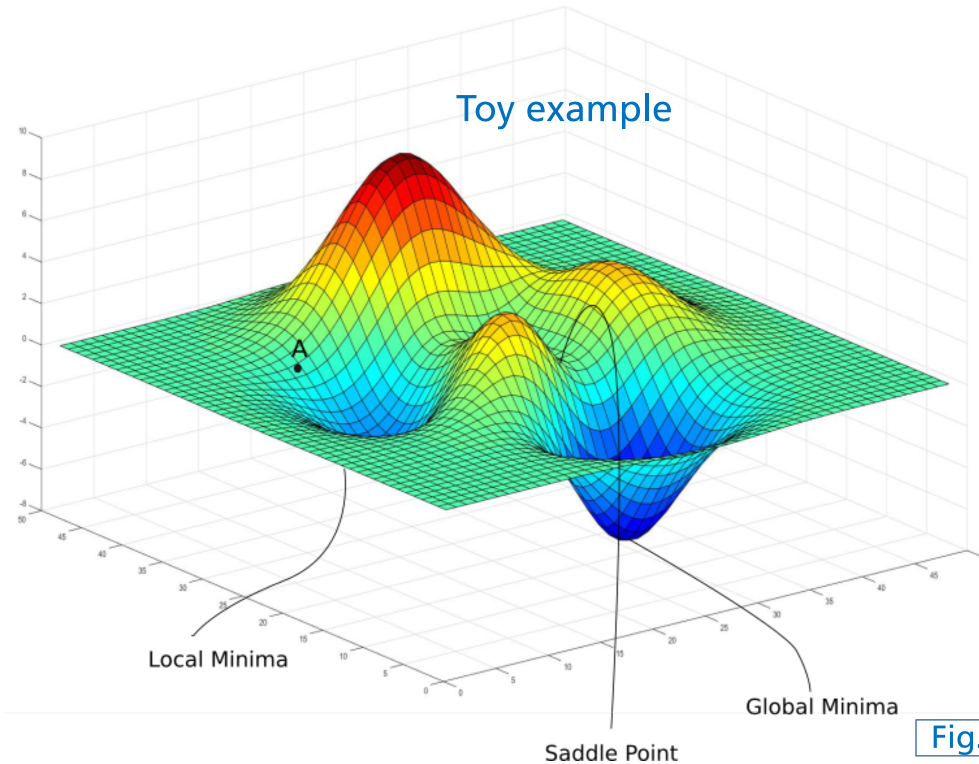



Fig. 13

Complex DNN

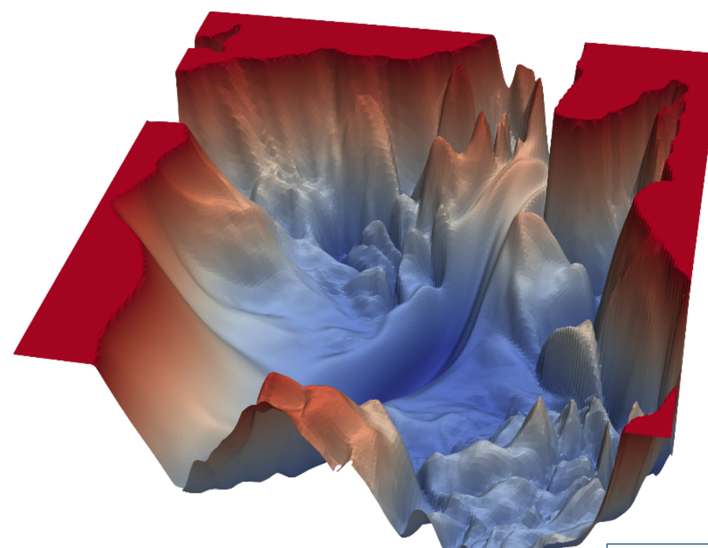


Fig. 14

# Deep Learning
## Playground

- A neural network playground!
  - https://playground.tensorflow.org

# Deep Learning
## Convolutional Neural Networks (CNN)

- Convolutional layers
    - "Convolution" → (local) dot-product between filter and input

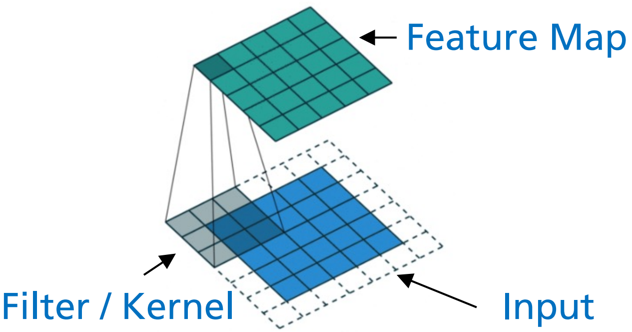



Fig. 16

# Deep Learning
## Convolutional Neural Networks (CNN)

- Convolutional layers
  - “Convolution” → (local) dot-product between filter and input
  - Shared weights (fewer parameters)

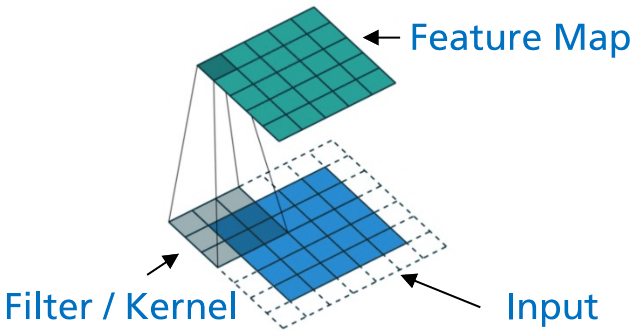



Fig. 16

# Deep Learning
## Convolutional Neural Networks (CNN)

- Convolutional layers
  - “Convolution” → (local) dot-product between filter and input
  - Shared weights (fewer parameters)
  - Translation of input → translation of activations (equivariance)

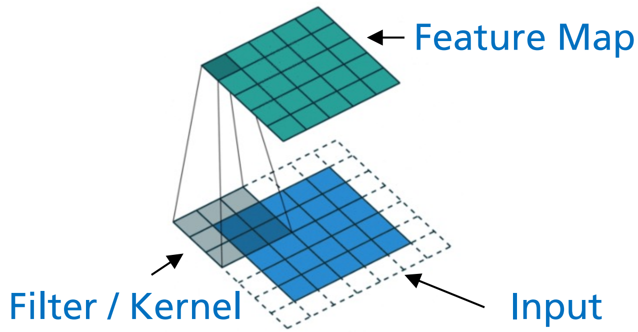



Fig. 16

# Deep Learning
## Convolutional Neural Networks (CNN)

- Convolutional layers
  - "Convolution" → (local) dot-product between filter and input
  - Shared weights (fewer parameters)
  - Translation of input → translation of activations (equivariance)
- Pooling → local aggregation / down-sampling

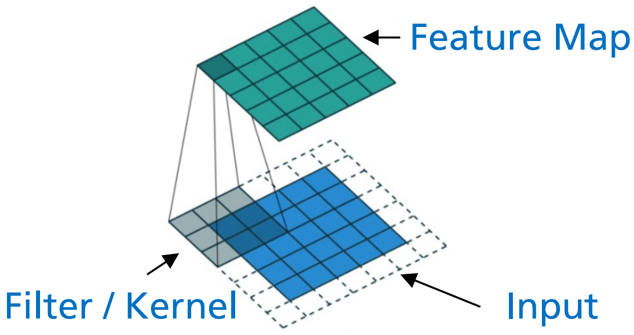



Fig. 16

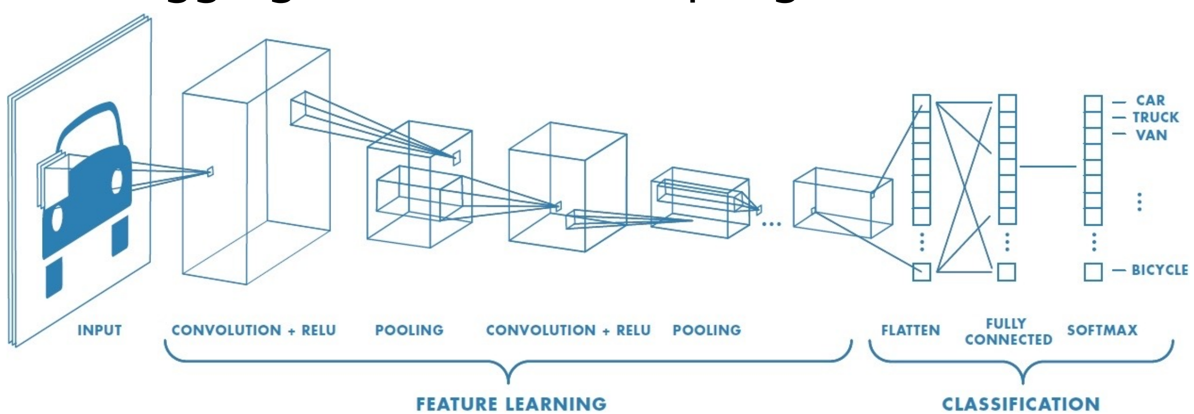



Fig. 17

# Deep Learning
## Recurrent Neural Networks (RNN)

- Recurrent layers
  - Model sequential data → model dynamic temporal behaviour
  - Internal memory state(s) → memorize previous data for future predictions

# Deep Learning
## Recurrent Neural Networks (RNN)

- Recurrent layers
    - Model sequential data → model dynamic temporal behaviour
    - Internal memory state(s) → memorize previous data for future predictions
- Vanishing gradient problem
    - Gating mechanisms (Gated Recurrent Units (GRU), Long Short-term Memory (LSTM)

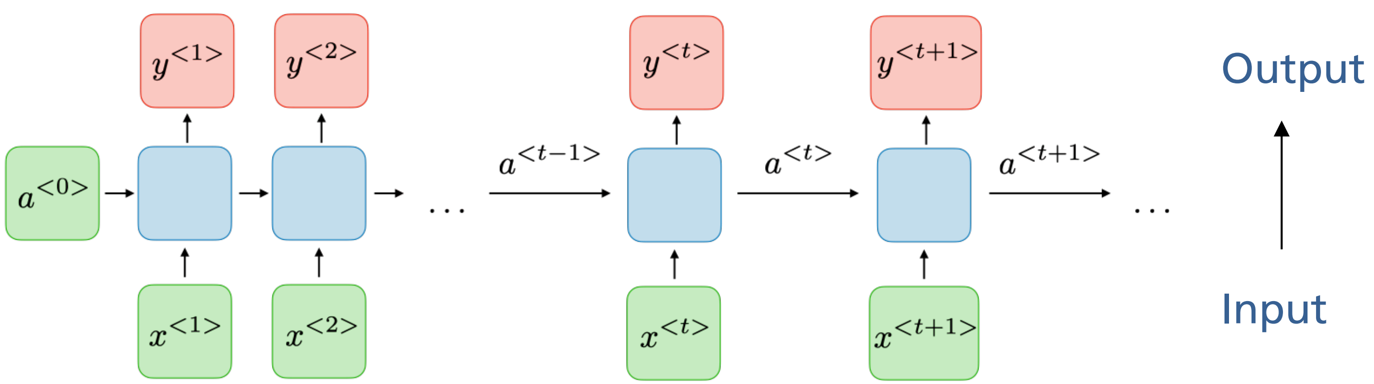



Fig. 18

# Deep Learning
## Recurrent Neural Networks (RNN)

- Application Examples
  - One-to-many: sequential music generation (given a starting note)


Fig. 19

# Deep Learning
## Recurrent Neural Networks (RNN)

- Application Examples
  - One-to-many: sequential music generation (given a starting note)
  - Many-to-one: sentiment classification (positive vs. negative)

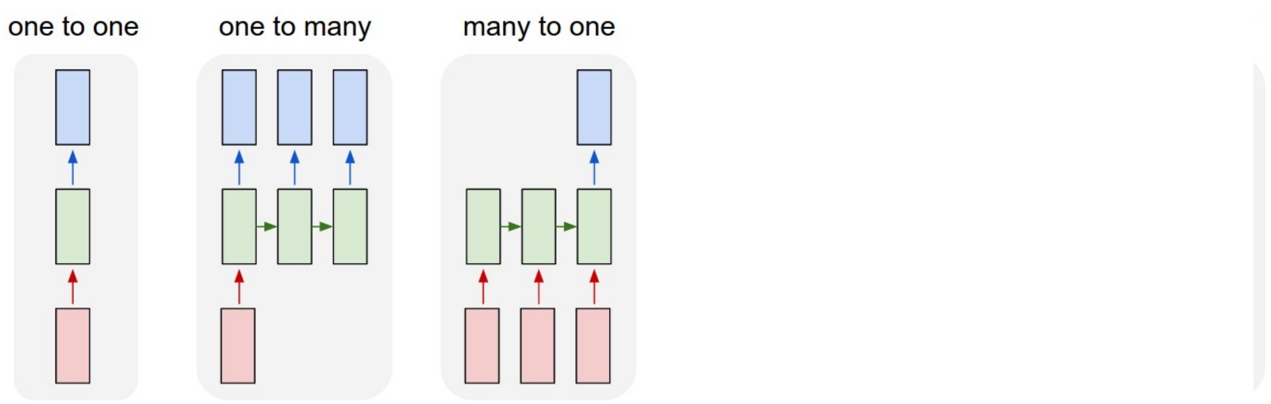



Fig. 19

# Deep Learning
## Recurrent Neural Networks (RNN)

- Application Examples
  - One-to-many: sequential music generation (given a starting note)
  - Many-to-one: sentiment classification (positive vs. negative)
  - Many-to-many: machine translation (e.g., Spanish to German)

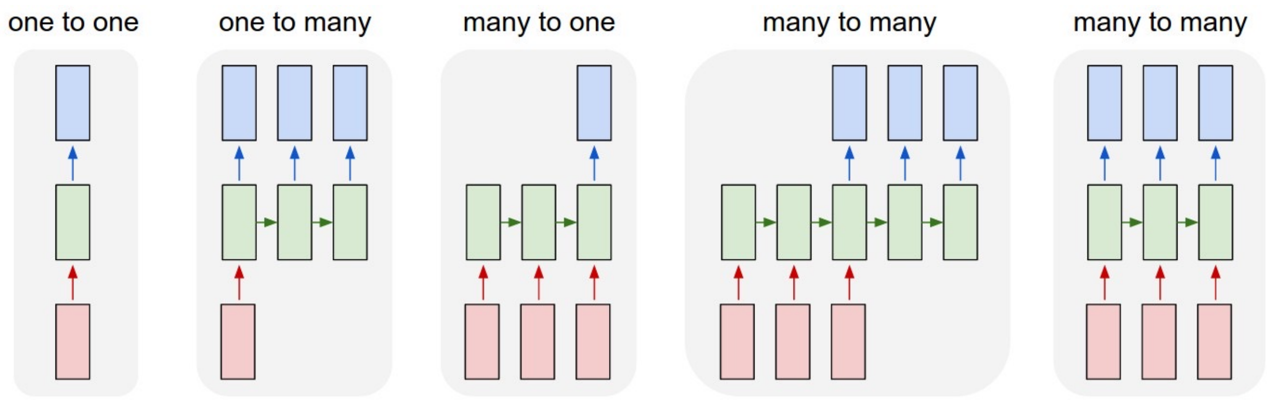
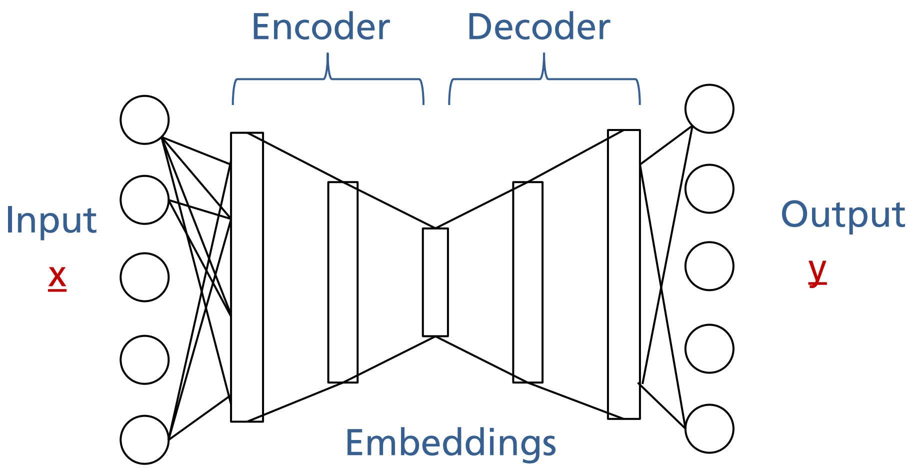



Fig. 19

# Deep Learning
## Autoencoders

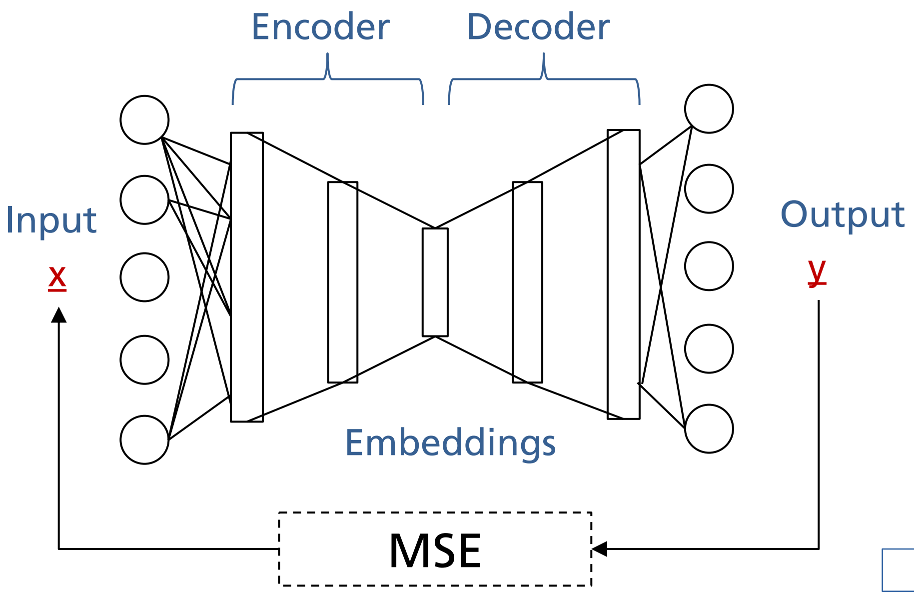
- Symmetric architecture (decoder & encoder)




Own

# Deep Learning
## Autoencoders

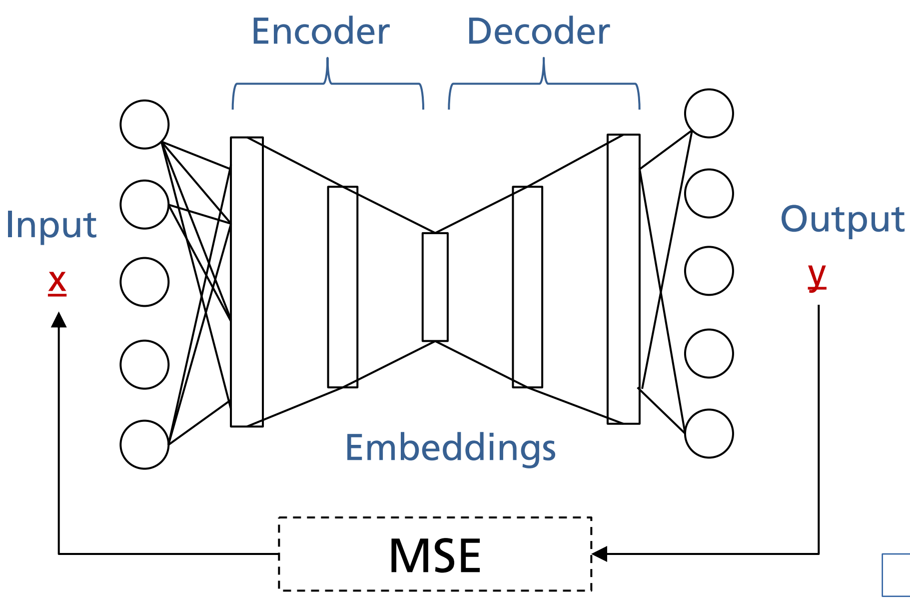
- Symmetric architecture (decoder & encoder)
- Objective: minimize reconstruction error (e.g., mean squared error, MSE)




Own

# Deep Learning
## Autoencoders

- Symmetric architecture (decoder & encoder)
- Objective: minimize reconstruction error (e.g., mean squared error, MSE)
- Compression of input (embedding)
- Prioritize important information → learn useful representations




Own

# Summary

- Introduction
  - Terminology, application scenarios
- Learning Paradigms
  - Unsupervised, supervised, self-supervised learning
- ML project pipeline
  - Data collection, pre-processing, split
  - Model selection, training, validation, testing
- Deep Learning
  - DNN, CNN, RNN, Autoencoders

# References

*Introducing Machine Learning*. (2016). Retrieved from https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/i/88174_92991v00_machine_learning_section1_ebook.pdf

S. Legg, M. Hutter (2007). Universal Intelligence: A Definition of Machine Intelligence. Minds & Machines. 17 (4): 391-444.

L. Samuel (1959). Some studies in machine learning using the game of checkers. IBM Journal of research and development. 3(3), 210-229

Srihari, S. N. (2020). *Forward Propagation and Backward Propagation (Deep Learning Lecture)*. Retrieved from https://cedar.buffalo.edu/~srihari/CSE676/6.5.0 Forward Backward.pdf

Virtanen, T., Plumbley, M. D., & Ellis, D. (Eds.). (2018). *Computational Analysis of Sound Scenes and Events*. Cham, Switzerland: Springer International Publishing.

# Images

Fig. 1: [Machine Learning, 2016], p. 4, Fig. 2

Fig. 2: https://i0.wp.com/www.sthda.com/sthda/RDoc/figure/clustering/ partitioning-cluster-analysis-k-means-plot-4-groups-1.png

Fig. 3: https://i.stack.imgur.com/hsilO.png (https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html)

Fig. 4: https://miro.medium.com/max/975/1*OyYyr9qY-w8RkaRh2TKo0w.png (reproduced)

Fig. 5: https://lilianweng.github.io/lil-log/assets/images/self-sup-lecun.png

Fig. 6: https://www.asimovinstitute.org/wp-content/uploads/2019/04/NeuralNetworkZoo20042019.png

Fig. 7: https://www.educative.io/api/edpresso/shot/6668977167138816/image/5033807687188480

Fig. 8: [Virtanen, 2018], p. 170, Fig. 6.7

Fig. 9: https://miro.medium.com/max/915/1*SJPacPhP4KDEB1AdhOFy_Q.png

Fig. 10: https://www.skampakis.com/wp-content/uploads/2018/03/simple_neural_network_vs_deep_learning.jpg

Fig. 11: https://pic4.zhimg.com/80/v2-057b248288a8af2f01272a956f862873_1440w.png

Fig. 12: https://blog.e-kursy.it/deeplearning4j-workshop/video/html/presentation_specific/img/4_activation_functions.png

# Images

Fig. 13: https://blog.paperspace.com/content/images/2018/05/challenges-1.png

Fig. 14: https://www.cs.umd.edu/~tomg/img/landscapes/noshort.png

Fig. 15: https://blog.paperspace.com/content/images/2018/05/grad.png

Fig. 16: https://www.wandb.com/articles/intro-to-cnns-with-wandb

Fig. 17: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

Fig. 18: https://wiki.tum.de/download/attachments/22578349/RNN1.png

Fig. 19: https://stanford.edu/~shervine/teaching/cs-230/illustrations/architecture-rnn-ltr.png

Fig. 20: [Srihari, 2020], p.8, (Fig. 1)

# Thank you!

- Any questions?

Dr.-Ing. Jakob Abeßer

Fraunhofer IDMT

Jakob.abesser@idmt.fraunhofer.de

https://www.machinelistening.de