Artificial Intelligence Data Analysis (AIDA)

1st School for Heliophysicists

Prof. Dr. – Ing. Morris Riedel
Associated Professor
School of Engineering and Natural Sciences, University of Iceland, Reykjavik, Iceland
Research Group Leader, Juelich Supercomputing Centre, Forschungszentrum Juelich, Germany

@Morris Riedel @MorrisRiedel @MorrisRiedel

LECTURE 4

Supervised Learning – Multi-Class Classification & Generalization

January 20, 2020
CINECA, Bologna, Italy

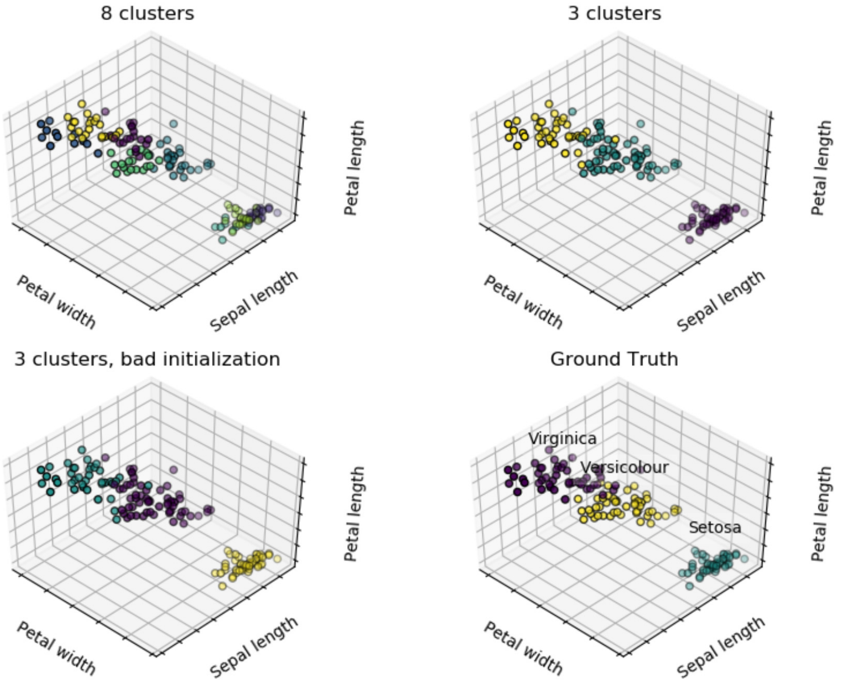
UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

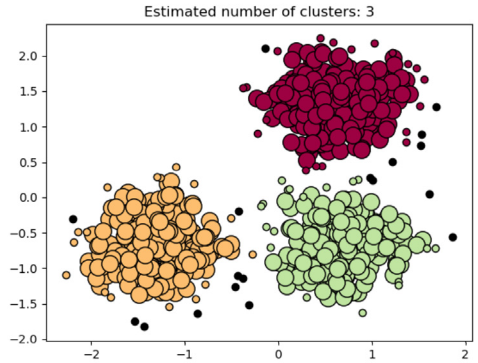
DEEP Projects

HELMHOLTZAI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

Review of Lecture 3 – Unsupervised Learning

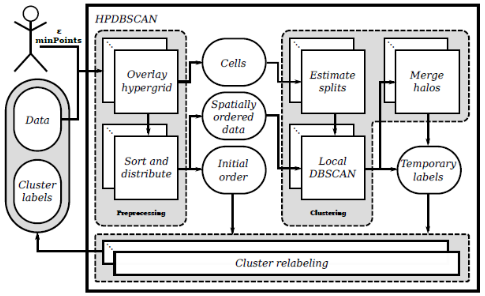


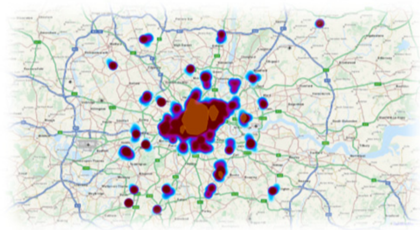
(IRIS dataset initially used for classification, cf. Lecture 1)



(using DBSCAN for outlier detection, e.g. to clean a dataset for data classification)

Out:
```
Estimated number of clusters: 3
Estimated number of noise points: 18
Homogeneity: 0.953
Completeness: 0.883
V-measure: 0.917
Adjusted Rand Index: 0.952
Adjusted Mutual Information: 0.916
Silhouette Coefficient: 0.626
```





(clustering example of tweets from Twitter)

[9] Scitkit-Learn

[10] M.Goetz, M. Riedel et al., 'HPDBSCAN – Highly Parallel DBSCAN', MLHPC Workshop at Supercomputing 2015

(parallelization of machine learning algorithms not trivial)

Outline of the School

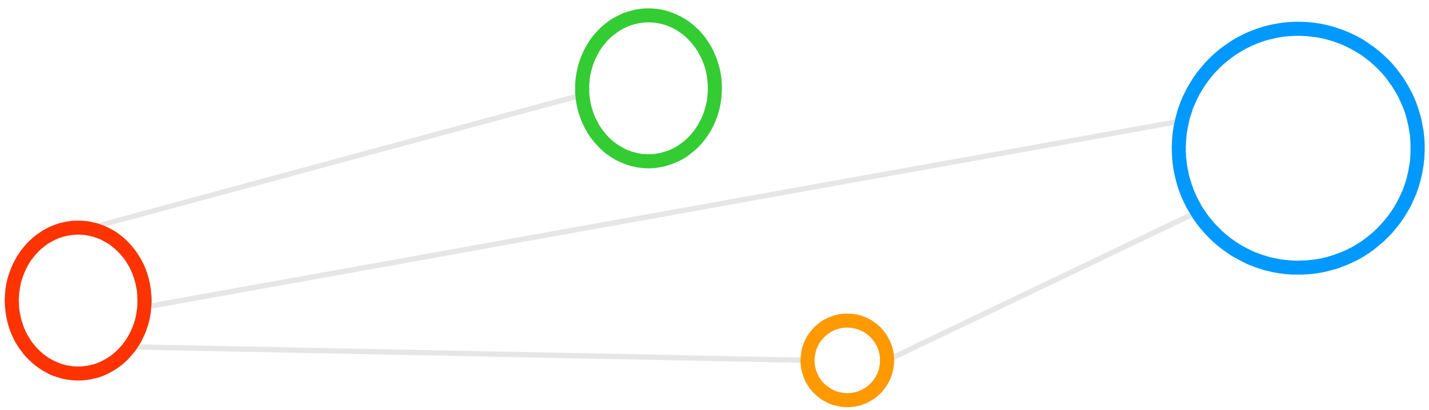
Time	**Day 1**	**Day 2**	**Day 3**
9 - 10	Welcome and intro to the school (Giovanni Lapenta, Jorge Amaya)	Space missions data acquisition (Hugo Breuillard)	Review of ML applied to heliophysics (Peter Wintoft)
10 - 11	Introduction and differences between AI, ML, NN and Big Data (Morris Riedel)	Data manipulation in python with pandas, xarray, and additional python tools (Geert Jan Bex)	Review of ML applied to heliophysics (Peter Wintoft)
	Coffee break	Coffee break	Coffee break
11:30 - 12:30	Unsupervised learning (Morris Riedel)	Feature engineering and data reduction (Geert Jan Bex)	Reinforcement learning (Morris Riedel)
	Lunch	Lunch	Lunch
14 - 15	Unsupervised learning (Morris Riedel)	Data reduction and visualization (Geert Jan Bex)	Physics informed ML (Romain Dupuis)
15 -16	Supervised learning (Morris Riedel)	CNN, DNN (Morris Riedel)	Explainable AI (Jorge Amaya)
	Coffee break	Coffee break	Coffee break
16:30 - 18:00	Supervised learning (Morris Riedel)	CNN, DNN (Morris Riedel)	Performance and tuning of ML (Morris Riedel)

Outline

- Supervised Learning & Multi-Class Classification Problems
  - Supervised Learning Revisited & Role of Deep Learning Frameworks
  - Formalization of Machine Learning Fundamentals & Perceptron Model
  - MNIST & Multi-Class Classification Problems
  - Relevance of Data Exploration, Data Preparation & Normalization
  - Multi-Output Perceptron Learning Model

- Supervised Learning & Theory of Generalization
  - Formalization of Supervised Learning & Mathematical Building Blocks
  - Feasibility of Learning & Understanding the Theory of Generalization
  - Role Learning Algorithms, and Final Hypothesis
  - Different Models in Hypothesis Set & Unlimited 'Degrees of Freedom'
  - Using Training Dataset as Training Dataset and as Testing Dataset

Supervised Learning and Multi-Class Classification Problems

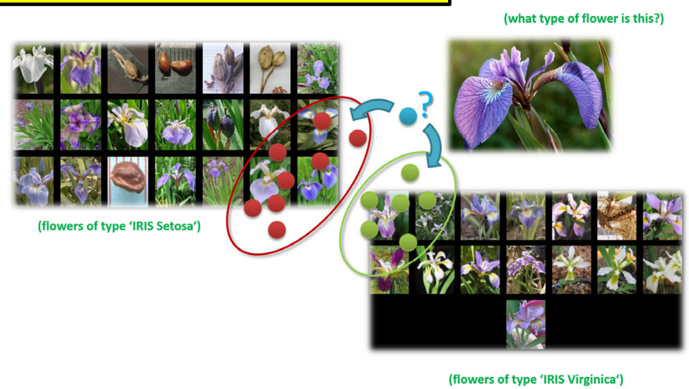


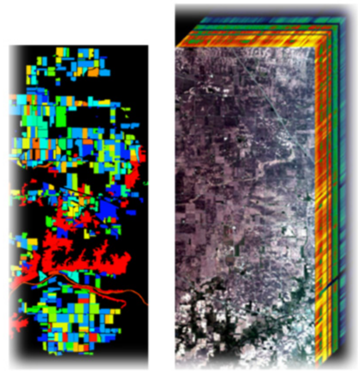
Learning Approaches – What means Learning from data – Revisited

- The basic meaning of learning is 'to use a set of observations to uncover an underlying process'
- The three different learning approaches are supervised, unsupervised, and reinforcement learning

[1] Image sources: Species Iris Group of North America Database, www.signa.org

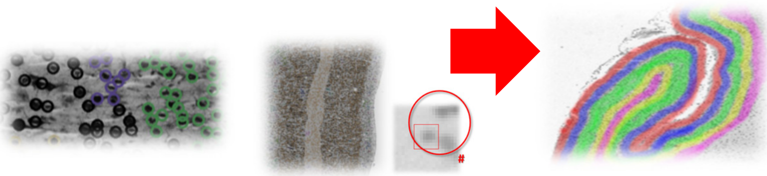
- Supervised Learning
  - Majority of methods follow this approach in this course
  - Example: credit card approval based on previous customer applications

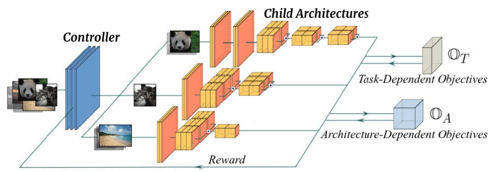




- Unsupervised Learning
  - Often applied before other learning → higher level data representation
  - Example: Coin recognition in vending machine based on weight and size

[2] A.C. Cheng et al., 'InstaNAS: Instance-aware Neural Architecture Search', 2018





- Reinforcement Learning
  - Typical 'human way' of learning
  - Example: Toddler tries to touch a hot cup of tea (again and again)

➢ **Day 1 offers details about unsupervised & supervised learning with examples & Day 3 offers an introduction to reinforcement learning**

Learning Approaches – Supervised Learning – Revisited

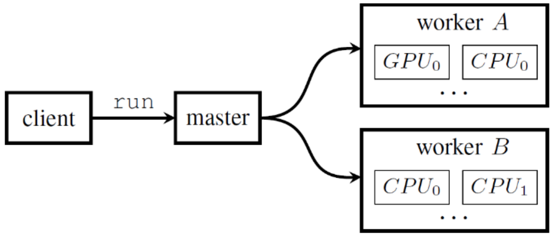
- Each observation of the predictor measurement(s) has an associated response measurement:
  - Input $\mathbf{x} = x_1, ..., x_d$
  - Output $y_i, i = 1, .., n$
  - Data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
  - (the output guides the learning process as a ‘supervisor‘)

- Goal: Fit a model that relates the response to the predictors
  - Prediction: Aims of accurately predicting the response for future observations
  - Inference: Aims to better understanding the relationship between the response and the predictors

- **Supervised learning approaches fits a model that related the response to the predictors**
- **Supervised learning approaches are used in classification algorithms such as SVMs**
- **Supervised learning works with data = [input, correct output]**

Deep Learning Frameworks using GPUs also good for Artificial Neural Networks

- TensorFlow
  - One of the most popular deep learning frameworks available today
  - Execution on multi-core CPUs or many-core GPUs



[3] Tensorflow Web page

- **Tensorflow is an open source library for deep learning models using a flow graph approach**
- **Tensorflow nodes model mathematical operations and graph edges between the nodes are so-called tensors (also known as multi-dimensional arrays)**
- **The Tensorflow tool supports the use of CPUs and GPUs (much more faster than CPUs)**
- **Tensorflow work with the high-level deep learning tool Keras in order to create models fast**
- **New versions of Tensorflow have Keras shipped with it as well & many further tools**

[4] Keras Web page

- Keras
  - Often used in combination with low-level frameworks like Tensorflow

- **Keras is a high-level deep learning library implemented in Python that works on top of existing other rather low-level deep learning frameworks like Tensorflow, CNTK, or Theano**
- **Created deep learning models with Keras run seamlessly on CPU and GPU via low-level deep learning frameworks**
- **The key idea behind the Keras tool is to enable faster experimentation with deep networks**

➢ **Day 2 offers more details on how these frameworks and tools are used with GPUs and for selected Deep Learning Techniques**

Exercises – Preparing & Installing the Keras Framework

DEEP Cluster: Preparing & Installing the Keras Framework

```
[riedel1@deepv ~]$ module load Python/3.6.8
```

```
[riedel1@deepv DeepLearning]$ pip install --user keras
```

Keras

[4] Keras Web page

```
[riedel1@deepv ~]$ pip install --user keras
Requirement already satisfied: keras in ./.local/lib/python3.6/site-packages (2.2.4)
Collecting numpy>=1.9.1 (from keras)
  Downloading https://files.pythonhosted.org/packages/0e/46/ae6773894f7eacf53308086287897ec568eac9768918d913d5b9d366c5db/numpy-1.17.3-cp36-cp36m-manylinux1_x86_64.whl (20.0MB)
    100% |████████████████████████████████| 20.0MB 534kB/s
Collecting h5py (from keras)
  Downloading https://files.pythonhosted.org/packages/60/06/cafdd44889200e5438b897388f3075b52a8ef01f28a17366d91de0fa2d05/h5py-2.10.0-cp36-cp36m-manylinux1_x86_64.whl (2.9MB)
    100% |████████████████████████████████| 2.9MB 918kB/s
Requirement already satisfied: six>=1.9.0 in /direct/usr_local/software/skylake/Stages/2019a/software/Python/3.6.8-GCCcore-8.3.0/lib/python3.6/site-packages/six-1.12.0-py3.6.egg (from keras) (1.12.0)
Collecting keras-applications>=1.0.6 (from keras)
  Downloading https://files.pythonhosted.org/packages/71/e3/19762fdfc62877ae9102edf6342d71b28fbfd9dea3d2f96a882ce099b03f/Keras_Applications-1.0.8-py3-none-any.whl (50kB)
    100% |████████████████████████████████| 51kB 1.4MB/s
Collecting scipy>=0.14 (from keras)
  Downloading https://files.pythonhosted.org/packages/29/50/a552a5aff252ae915f522e44642bb49a7b7b31677f9580cfd11bcc869976/scipy-1.3.1-cp36-cp36m-manylinux1_x86_64.whl (25.2MB)
    100% |████████████████████████████████| 25.2MB 463kB/s
Collecting keras-preprocessing>=1.0.5 (from keras)
  Using cached https://files.pythonhosted.org/packages/28/6a/8c1f62c37212d9fc441a7e26736df51ce6f0e38455816445471f10da4f0a/Keras_Preprocessing-1.1.0-py2.py3-none-any.whl
Requirement already satisfied: pyyaml in /direct/usr_local/software/skylake/Stages/2019a/software/Python/3.6.8-GCCcore-8.3.0/lib/python3.6/site-packages (from keras) (5.1)
Installing collected packages: numpy, h5py, keras-applications, scipy, keras-preprocessing
  The scripts f2py, f2py3 and f2py3.6 are installed in '/p/home/jusers/riedel1/deep/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed h5py-2.10.0 keras-applications-1.0.8 keras-preprocessing-1.1.0 numpy-1.17.3 scipy-1.3.1
```

Perceptron Model – Mathematical Notation for one Neuron

non-linear activation function

linear combination of input data

$$\hat{y} = g\left(1 * w_0 + \sum_{i=1}^{m} x_i * w_i\right) \Rightarrow \hat{y} = g\left(w_0 + X^T \mathbf{w}\right)$$

Output

Bias

Sum

Constants

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

Input Data

Trainable Weights

- Simplify the perceptron learning model formula with techniques from linear algebra for mathematical convenience

Handwritten Character Recognition MNIST Dataset

- Metadata
  - Not very challenging dataset, but good for benchmarks & tutorials
- When working with the dataset
  - Dataset is not in any standard image format like jpg, bmp, or gif (i.e. file format not known to a graphics viewer)
  - Data samples are stored in a simple file format that is designed for storing vectors and multidimensional matrices (i.e. numpy arrays)
  - The pixels of the handwritten digit images are organized row-wise with pixel values ranging from 0 (white background) to 255 (black foreground)
  - Images contain grey levels as a result of an anti-aliasing technique used by the normalization algorithm that generated this dataset

- **Handwritten Character Recognition MNIST dataset is a subset of a larger dataset from US National Institute of Standards (NIST)**
- **MNIST handwritten digits includes corresponding labels with values 0-9 and is therefore a labeled dataset**
- **MNIST digits have been size-normalized to 28 * 28 pixels & are centered in a fixed-size image for direct processing**
- **Two separate files for training & test: 60000 training samples (~47 MB) & 10000 test samples (~7.8 MB)**

(10 class classification problem)

```python
import numpy as np
from keras.datasets import mnist
```

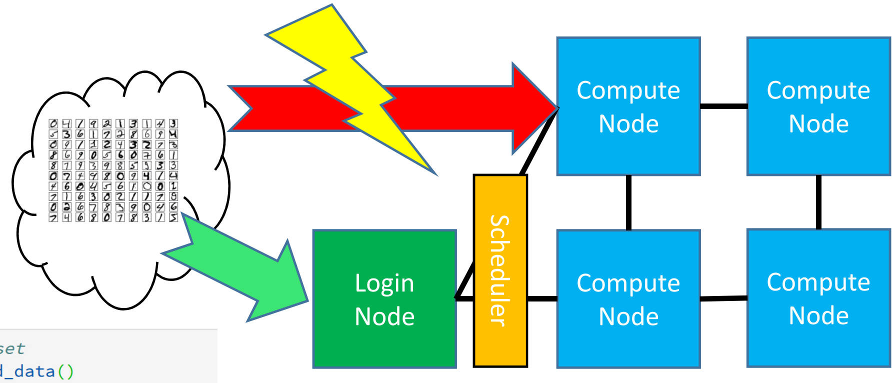
```python
# download and shuffled as training and testing set
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

(downloads data into ~home/.keras/datasets as NPZ file format of numpy that provides storage of array data using gzip compression)

MNIST Dataset – Data Access in Python & HPC Download Challenges

- Warning for very secure HPC environments
  - Note that HPC batch nodes often do not allow for download of remote files

> - **A useful workaround for download remotely stored datasets and files is to start the Keras script on the login node and after data download stop the script for a proper execution on batch nodes for training & inference**




```
# download and shuffled as training and testing set
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 6s 1us/step
```

```
[riedel1@juron1-adm datasets]$ pwd
/p/home/jusers/riedel1/juron/.keras/datasets
[riedel1@juron1-adm datasets]$ ls -al
total 11234
drwxr-xr-x 2 riedel1 jusers     4096 Jan 20 22:05 .
drwxr-xr-x 3 riedel1 jusers     4096 Jan 20 22:03 ..
-rw-r--r-- 1 riedel1 jusers 11490434 Jan 20 22:05 mnist.npz
```

```
import numpy as np
from keras.datasets import mnist
```

```
# download and shuffled as training and testing set
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

(downloads data into ~home/.keras/datasets as NPZ file format of numpy that provides storage of array data using gzip compression)

Exercises – Download MNIST Data

DEEP Cluster: Download MNIST Data

- Execute on Login-Node the Data Exploration Script to Download MNIST Data
  - *Load the following module environment:*

```
[riedel1@deepv 2019-HPC-Course-MLDL-Parts]$ module load Python/3.6.8
[riedel1@deepv 2019-HPC-Course-MLDL-Parts]$ module load scikit/2019a-Python-3.6.8
[riedel1@deepv 2019-HPC-Course-MLDL-Parts]$ module load TensorFlow/1.13.1-GPU-Python-3.6.8
```

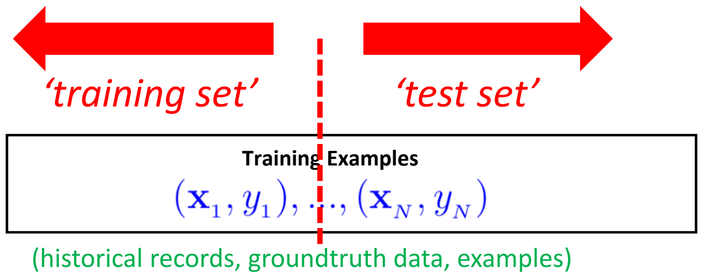
  - *Python explore-MNIST-training.py*

```
[riedel1@deepv 2019-HPC-Course-MLDL-Parts]$ python explore-MNIST-training.py
Using TensorFlow backend.
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 84 185 159 151 60 36 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 222 254 254 254 254 241 198 198 198 198 198 198 198 198 170 52 0 0 0 0 0 0
0 0 0 0 0 0 67 114 72 114 163 227 254 225 254 254 254 250 229 254 254 140 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 17 66 14 67 67 67 59 21 236 254 106 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 83 253 209 18 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 22 233 255 83 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 129 254 238 44 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 59 249 254 62 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 133 254 187 5 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 205 248 58 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 126 254 182 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 75 251 240 57 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 19 221 254 166 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 203 254 219 35 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 38 254 254 77 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 31 224 254 115 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 133 254 254 52 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 61 242 254 254 52 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 121 254 254 219 40 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 121 254 207 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


Label:
7
```

MNIST Dataset – Training/Testing Datasets & One Character Encoding

- **Different phases in machine learning**
- **Training phases is a hypothesis search**
- **Testing phase checks if we are on the right track once the hypothesis is clear**
- **Validation phane for model selection (set fixed parameters and set model types)**

- Work on two disjoint datasets
  - One for training only (i.e. training set)
  - One for testing only (i.e. test set)
  - Exact seperation is rule of thumb per use case (e.g. 10 % training, 90% test)
  - Practice: If you get a dataset take immediately test data away ('throw it into the corner and forget about it during modelling')
  - Once we learned from training data it has an 'optimistic bias'
  - Usually start by exploring the dataset and its format & labels



MNIST Dataset – Data Exploration Script Training Data & JupyterLab Example

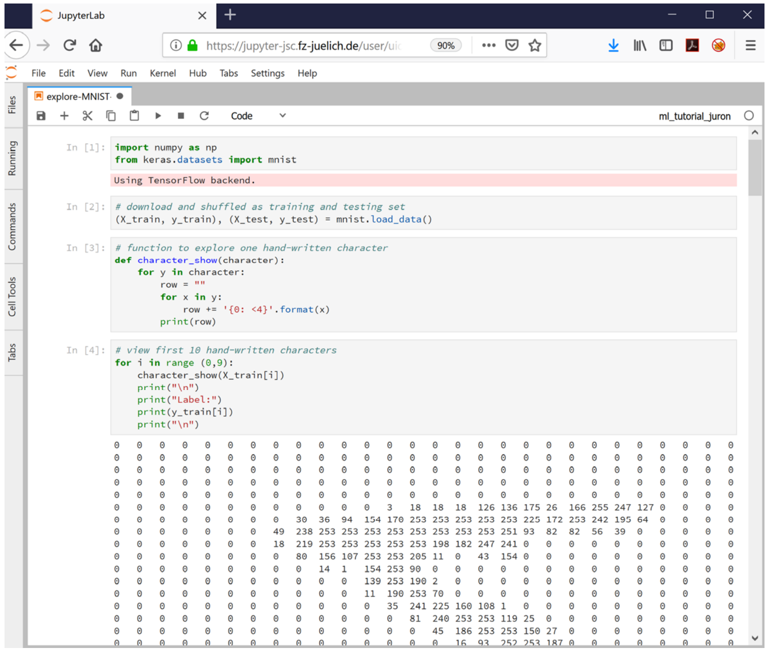
```python
import numpy as np
from keras.datasets import mnist

# download and shuffled as training and testing set
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```python
# function to explore one hand-written character
def character_show(character):
    for y in character:
        row = ""
        for x in y:
            row += '{0: <4}'.format(x)
        print(row)
```

```python
# view first 10 hand-written characters
for i in range (0,9):
    character_show(X_train[i])
    print("\n")
    print("Label:")
    print(y_train[i])
    print("\n")
```

- Loading MNIST training datasets (X) with labels (Y) stored in a binary numpy format
- Format is 28 x 28 pixel values with grey level from 0 (white background) to 255 (black foreground)

- Small helper function that prints row-wise one ‘hand-written‘ character with the grey levels stored in training dataset
- Should reveal the nature of the number (aka label)



[5] Jupyter Web Page

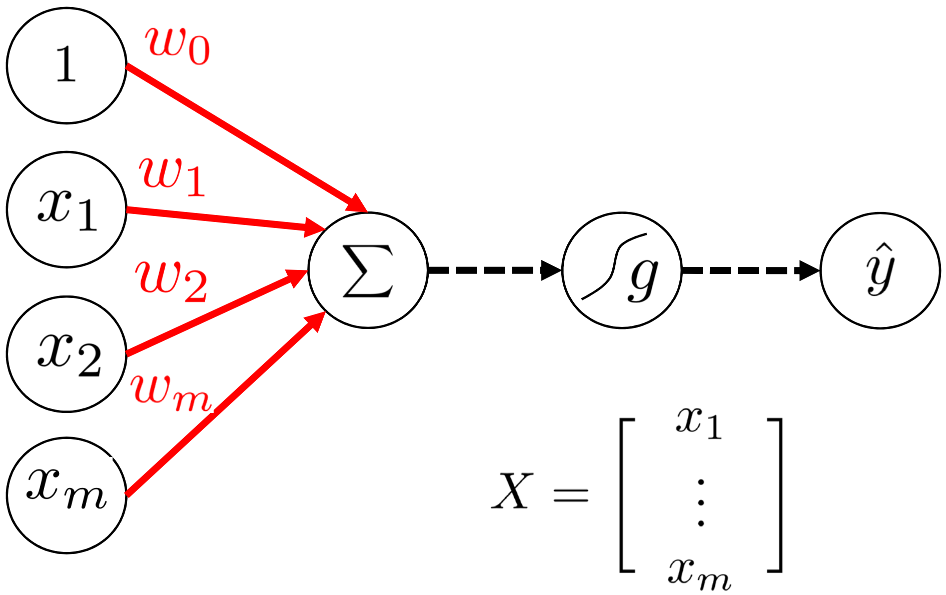
- Example: loop of the training dataset (e.g. first 10 characters as shown here)
- At each loop interval the ‘hand-written‘ character (X) is printed in ‘matrix notation‘ & label (Y)

MNIST Dataset with Perceptron Learning Model – Need for Reshape

- Two dimensional dataset (28 x 28)
  - Does not fit well with input to Perceptron Model
  - Need to prepare the data even more
  - Reshape data → we need one long vector

- Note that the reshape from two dimensional MNIST data to one long vector means that we loose the surrounding context
- Loosing the surrounding context is one factor why later in this lecture deep learning networks achieving essentially better performance by, e.g., keeping the surrounding context

```
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   3   18  18  18  126 136 175 26  166 255 247 127 0   0   0   0
0   0   0   0   0   0   0   0   30  36  94  154 170 253 253 253 253 253 225 172 253 242 195 64  0   0   0   0
0   0   0   0   0   0   0   49  238 253 253 253 253 253 253 253 253 251 93  82  82  56  39  0   0   0   0   0
0   0   0   0   0   0   0   18  219 253 253 253 253 253 198 182 247 241 0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   80  156 107 253 253 205 11  0   43  154 0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   14  1   154 253 90  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   139 253 190 2   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   11  190 253 70  0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   35  241 225 160 108 1   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   81  240 253 253 119 25  0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   45  186 253 253 150 27  0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   16  93  252 253 187 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   249 253 249 64  0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   46  130 183 253 253 207 2   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   39  148 229 253 253 253 250 182 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   24  114 221 253 253 253 253 201 78  0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   23  66  213 253 253 253 253 198 81  2   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   18  171 219 253 253 253 253 195 80  9   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   55  172 226 253 253 253 253 244 133 11  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   136 253 253 253 212 135 132 16  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Label:
5
```



MNIST Dataset – Reshape & Normalization – Example

(one long input vector with length 784)

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

```
784 input pixel values per train samples
784 input pixel values per test samples
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333336 0.6862745  0.10196079 0.6509804  1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117648 0.36862746 0.6039216
 0.6666667  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235295 0.6745098  0.99215686 0.9490196  0.7647059  0.2509804
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215687
 0.93333334 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.9843137  0.3647059  0.32156864
 0.32156864 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882354 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.7764706  0.7137255
```

(numbers are between 0 and 1)

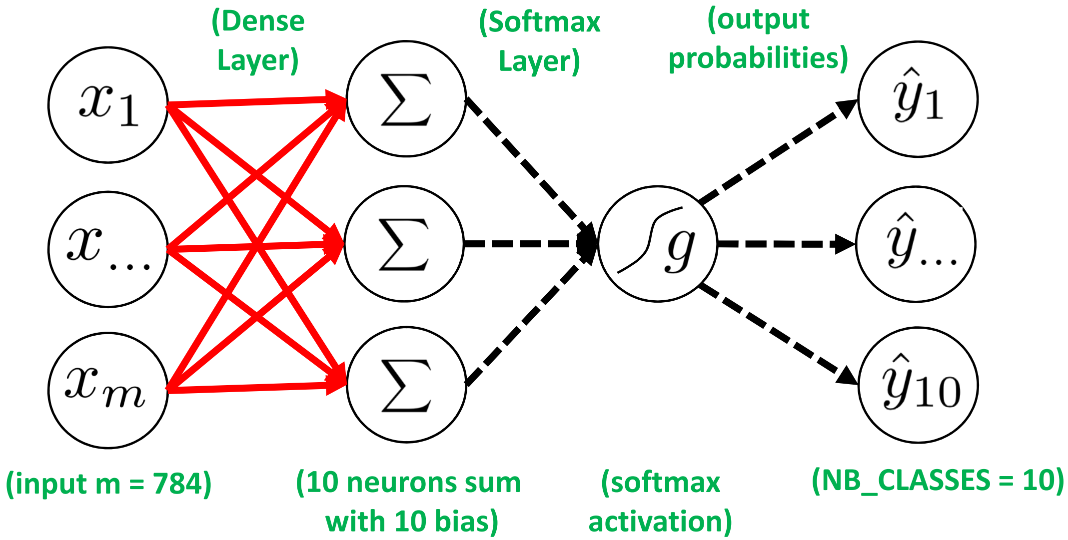
(two dimensional original input)

```
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   3   18  18  18  126 136 175 26  166 255 247 127 0   0   0   0
0   0   0   0   0   0   0   0   30  36  94  154 170 253 253 253 253 253 225 172 253 242 195 64  0   0   0   0
0   0   0   0   0   0   0   49  238 253 253 253 253 253 253 253 253 251 93  82  82  56  39  0   0   0   0   0
0   0   0   0   0   0   0   18  219 253 253 253 253 253 198 182 247 241 0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   80  156 107 253 253 205 11  0   43  154 0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   14  1   154 253 90  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   139 253 190 2   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   11  190 253 70  0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   35  241 225 160 108 1   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   81  240 253 253 119 25  0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   45  186 253 253 150 27  0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   16  93  252 253 187 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   249 253 249 64  0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   46  130 183 253 253 207 2   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   39  148 229 253 253 253 250 182 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   24  114 221 253 253 253 253 201 78  0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   23  66  213 253 253 253 253 198 81  2   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   18  171 219 253 253 253 253 195 80  9   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   55  172 226 253 253 253 253 244 133 11  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   136 253 253 253 212 135 132 16  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Label:
5
```

Exercises – Perform Reshape & Normalization on Different Training / Testing Data

MNIST Dataset & Multi Output Perceptron Model

- 10 Class Classification Problem
  - Use 10 Perceptrons for 10 outputs with softmax activation function (enables probabilities for 10 classes)




```python
from keras.models import Sequential
from keras.layers.core import Dense, Activation
```

```python
# model Keras sequential
model = Sequential()
```

```python
# add fully connected layer - input with output
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
```

```python
# add activation function layer to get class probabilities
model.add(Activation('softmax'))
```

```python
# printout a summary of the model to understand model complexity
model.summary()
```

(parameters = 784 * 10 + 10 bias = 7850)

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 10)	7850
activation_1 (Activation)	(None, 10)	0

Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0

- **Note that the output units are independent among each other in contrast to neural networks with one hidden layer**
- **The output of softmax gives class probabilities**
- **The non-linear Activation function 'softmax' represents a generalization of the sigmoid function – it squashes an n-dimensional vector of arbitrary real values into a n-dimenensional vector of real values in the range of 0 and 1 – here it aggregates 10 answers provided by the Dense layer with 10 neurons**

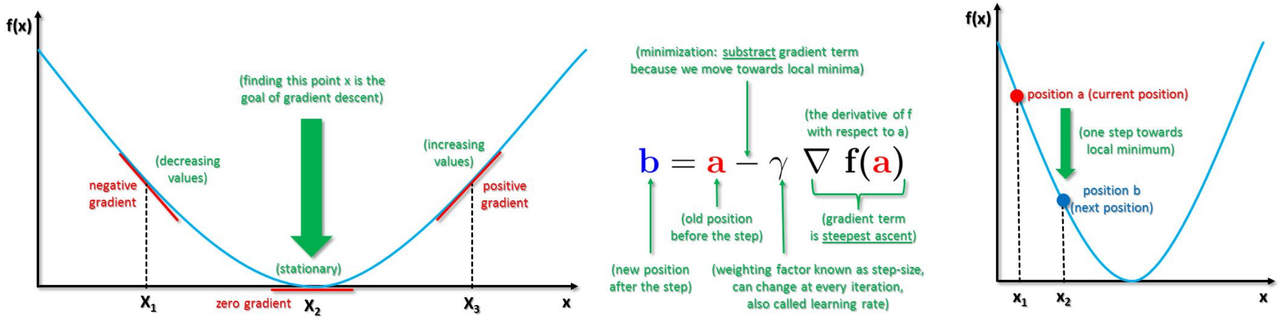
MNIST Dataset & Compile Multi Output Perceptron Model

- Compile the model
  - Optimizer as algorithm used to update weights while training the model
  - Specify loss function (i.e. objective function) that is used by the optimizer to navigate the space of weights
  - (note: process of optimization is also called loss minimization, cf. Invited lecture Gabriele Cavallaro)
  - Indicate metric for model evaluation (e.g., accuracy)
- Specify loss function
  - Compare prediction vs. given class label
  - E.g. categorical crossentropy

```
# specify loss, optimizer and metric
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])
```

```
from keras.optimizers import SGD
```

```
OPTIMIZER = SGD() # optimization technique
```



- **Compile the model to be executed by the Keras backend (e.g. TensorFlow)**
- **Optimizer Gradient Descent (GD) uses all the training samples available for a step within a iteration**
- **Optimizer Stochastic Gradient Descent (SGD) converges faster: only one training samples used per iteration**
- **Loss function is a multi-class logarithmic loss: target is *ti,j* and prediction is *pi,j***
- **Categorical crossentropy is suitable for multiclass label predictions (default with softmax)**

$$L_i = -\Sigma_j t_{i,j} \log(p_{i,j})$$

[6] Big Data Tips, Gradient Descent

Full Script: MNIST Dataset – Model Parameters & Data Normalization

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils
```

```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimization technique
VERBOSE = 1
```

```
# download and shuffled as training and testing set
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
# X_train is 60000 rows of 28x28 values --> reshaped in 60000 x 784
RESHAPED = 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

```
# normalize
X_train /= 255
X_test /= 255
```

```
# output number of samples
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

- **NB_CLASSES: 10 Class Problem**
- **NB_EPOCH: number of times the model is exposed to the overall training set – at each iteration the optimizer adjusts the weights so that the objective function is minimized**
- **BATCH_SIZE: number of training instances taken into account before the optimizer performs a weight update to the model**
- **OPTIMIZER: Stochastic Gradient Descent ('SGD') – only one training sample/iteration**

- **Data load shuffled between training and testing set in files**
- **Data preparation, e.g. X_train is 60000 samples / rows of 28 x 28 pixel values that are reshaped in 60000 x 784 including type specification (i.e. float32)**
- **Data normalization: divide by 255 – the max intensity value to obtain values in range [0,1]**

'training set' *'test set'*

Training Examples

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$$

(historical records, groundtruth data, examples)

Full Script: MNIST Dataset – Fitting a Multi Output Perceptron Model

(full script continued from previous slide)

```
# convert class label vectors using one hot encoding
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# model Keras sequential
model = Sequential()

# add fully connected layer - input with output
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))

# add activation function layer to get class probabilities
model.add(Activation('softmax'))

# printout a summary of the model to understand model complexity
model.summary()

# specify loss, optimizer and metric
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# model training
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE)

# model evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

- **The Sequential() Keras model is a linear pipeline (aka 'a stack') of various neural network layers including Activation functions of different types (e.g. softmax)**
- **Dense() represents a fully connected layer used in ANNs that means that each neuron in a layer is connected to all neurons located in the previous layer**

- **The non-linear activation function 'softmax' is a generalization of the sigmoid function – it squashes an n-dimensional vector of arbitrary real values into a n-dimenensional vector of real values in the range of 0 and 1 – here it aggregates 10 answers provided by the Dense layer with 10 neurons**

- **Loss function is a multi-class logarithmic loss: target is *ti,j* and the prediction is *pi,j***

$$L_i = -\Sigma_j t_{i,j} \log(p_{i,j})$$

- **Train the model ('fit') using selected batch & epoch sizes on training & test data**

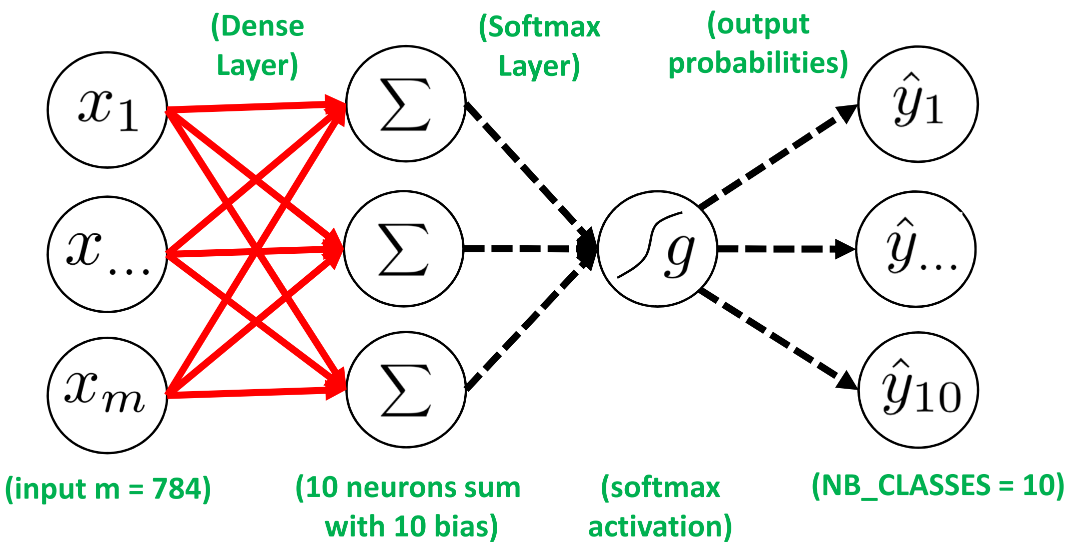
Exercises – Execute Multi Output Perceptron Model

MNIST Dataset – A Multi Output Perceptron Model – Output & Evaluation

```
Epoch 7/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4419 - acc: 0.8838
Epoch 8/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4271 - acc: 0.8866
Epoch 9/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.4151 - acc: 0.8888
Epoch 10/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4052 - acc: 0.8910
Epoch 11/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3968 - acc: 0.8924
Epoch 12/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3896 - acc: 0.8944
Epoch 13/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3832 - acc: 0.8956
Epoch 14/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3777 - acc: 0.8969
Epoch 15/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3727 - acc: 0.8982
Epoch 16/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3682 - acc: 0.8989
Epoch 17/20
60000/60000 [==============================] - 1s 25us/step - loss: 0.3641 - acc: 0.9001
Epoch 18/20
60000/60000 [==============================] - 1s 25us/step - loss: 0.3604 - acc: 0.9007
Epoch 19/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3570 - acc: 0.9016
Epoch 20/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3538 - acc: 0.9023
```

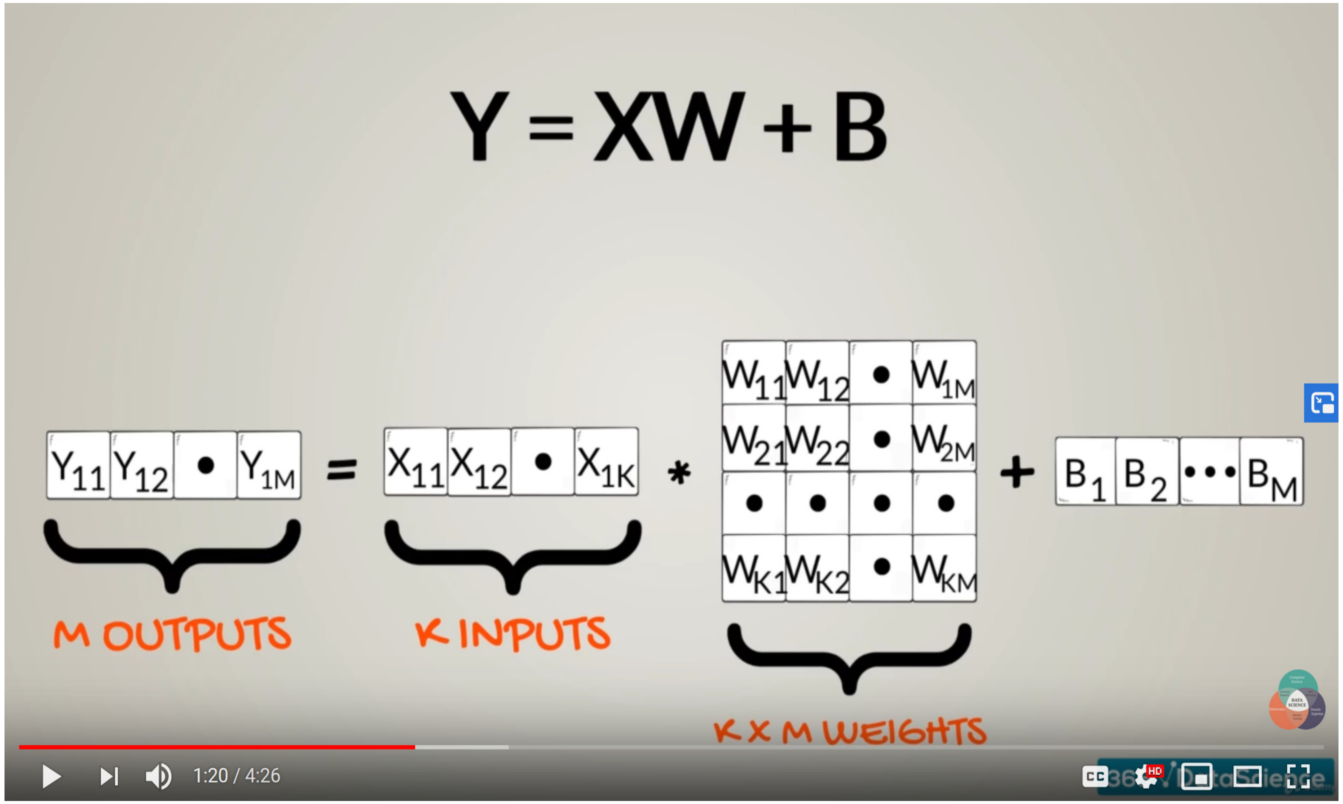
```python
# model evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 0s 41us/step
Test score: 0.33423959468007086
Test accuracy: 0.9101
```



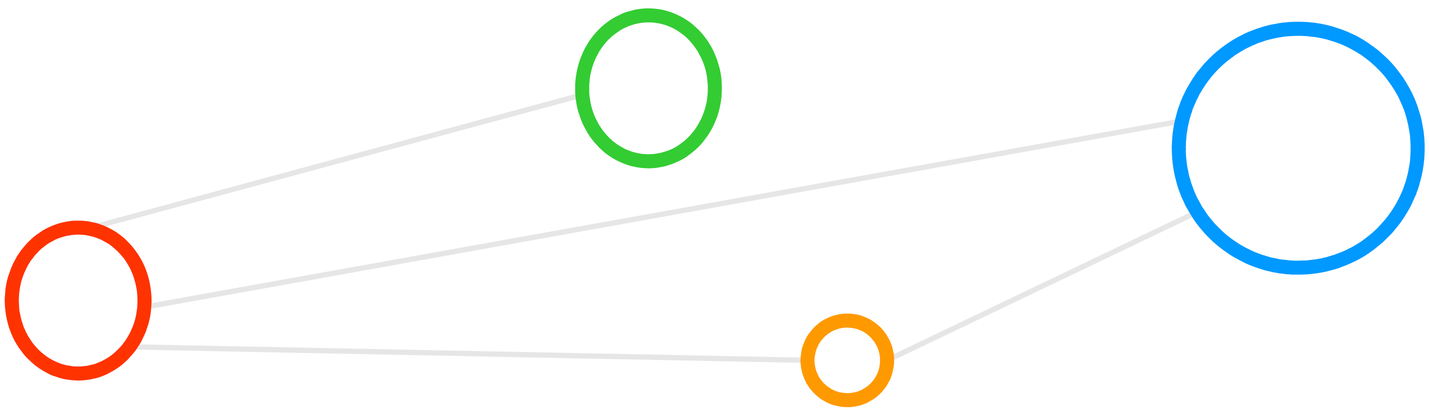
- **How to improve the model design by extending the neural network topology?**
- **Which layers are required?**
- **Think about input layer need to match the data – what data we had?**
- **Maybe hidden layers?**
- **How many hidden layers?**
- **What activation function for which layer (e.g. maybe ReLU)?**
- **Think Dense layer – Keras?**
- **Think about final Activation as Softmax → output probability**

[Video] Multi Output Perceptron – More Details



[8] YouTube Video, The Linear model with Multiple Inputs and Multiple Outputs Detail Explanation

Formalization of Supervised Learning & Theory of Generalization



Exercises – Execute Multi Output Perceptron Model and Test on Training Dataset

Learning Approaches – Supervised Learning – Formalization

- Each observation of the predictor measurement(s) has an associated response measurement:
  - Input $\mathbf{x} = x_1, ..., x_d$
  - Output $y_i, i = 1, .., n$
  - Data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
  - (the output guides the learning process as a 'supervisor')
- Goal: Fit a model that relates the response to the predictors
  - Prediction: Aims of accurately predicting the response for future observations
  - Inference: Aims to better understanding the relationship between the response and the predictors

> - **Supervised learning approaches fits a model that related the response to the predictors**
> - **Supervised learning approaches are used in classification algorithms such as SVMs**
> - **Supervised learning works with data = [input, correct output]**

Training Examples

$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$

(historical records, groundtruth data, examples)

Feasibility of Learning from Data – Formalization

- Theoretical framework underlying practical learning algorithms
  - E.g. Support Vector Machines (SVMs)
  - Best understood for ‘Supervised Learning‘
  - Valid for bascially all machine learning algorithms

- Theoretical background used to solve ‘A learning problem‘
  - Inferring one ‘target function‘ that maps between input and output
  - Learned function can be used to predict output from future input (fitting existing data is not enough)

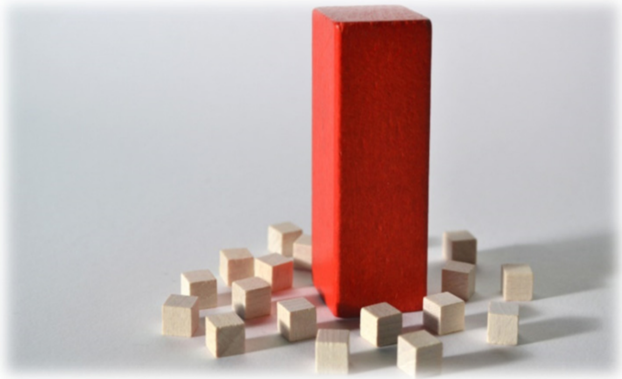
Unknown Target Function

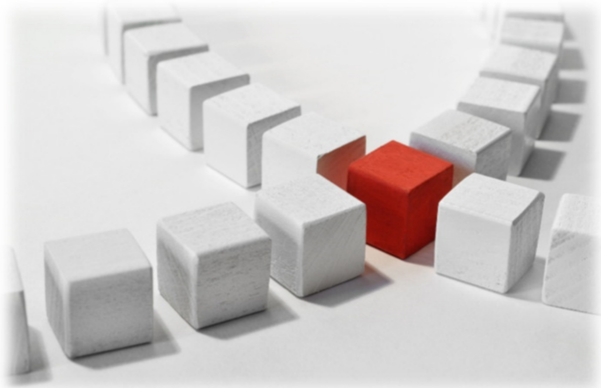
$f : X \rightarrow Y$

(ideal function)

Summary Terminologies & Importance of Theory of Generalization

- Target Function
  - Ideal function that 'explains' the data we want to learn
- Labelled Dataset (samples)
  - 'in-sample' data given to us:
- Learning vs. Memorizing
  - The goal is to create a system that works well 'out of sample'
  - In other words we want to classify 'future data' (ouf of sample) correct
- Dataset Part One: Training set
  - Used for training a machine learning algorithms
  - Result after using a training set: a trained system
- Dataset Part Two: Test set
  - Used for testing whether the trained system might work well
  - Result after using a test set: accuracy of the trained model





Unknown Target Function

$f : X \to Y$

(ideal function)

Elements we not exactly (need to) know

Training Examples

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(historical records, groundtruth data, examples)

Elements we must and/or should have and that might raise huge demands for storage

Elements that we derive from our skillset and that can be computationally intensive

Elements that we derive from our skillset

Mathematical Building Blocks (1) – Our Linear Example

Unknown Target Function
$f: X \rightarrow Y$

(ideal function)

↓

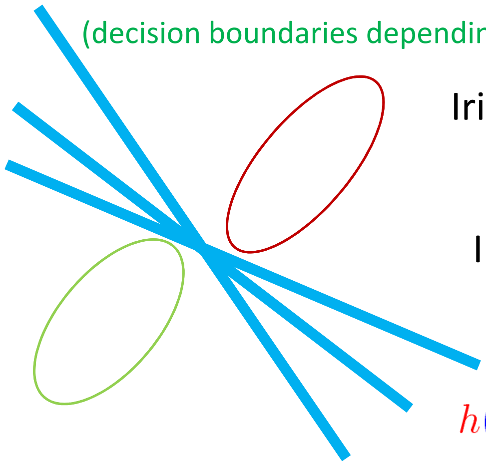
Training Examples
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(historical records, groundtruth data, examples)

1. Some pattern exists
2. No exact mathematical formula (i.e. target function)
3. Data exists

(if we would know the exact target function we dont need machine learning, it would not make sense)

(decision boundaries depending on f)



Iris-virginica if $\sum_{i=1}^{d} w_i x_i > threshold$

Iris-setosa if $\sum_{i=1}^{d} w_i x_i < threshold$

(w$_i$ and threshold are still unknown to us)

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right); h \in \mathcal{H}$$

(we search a function similiar like a target function)

Feasibility of Learning – Hypothesis Set & Final Hypothesis

- The 'ideal function' will remain unknown in learning
  - Impossible to know and learn from data
  - If known a straightforward implementation would be better than learning
  - E.g. hidden features/attributes of data not known or not part of data

Unknown Target Function
$f : X \rightarrow Y$

- But '(function) approximation' of the target function is possible
  - Use training examples to learn and approximate it
  - Hypothesis set $\mathcal{H}$ consists of m different hypothesis (candidate functions)

$$\mathcal{H} = \{h_1, ..., h_m\};$$

'select one function' that best approximates

$$g : X \rightarrow Y$$

Hypothesis Set
$\mathcal{H} = \{h\}; \; g \in \mathcal{H}$

Final Hypothesis
$g \approx f$

Unknown Target Function

$f: X \to Y$

(ideal function)

Elements we not exactly (need to) know

Training Examples

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(historical records, groundtruth data, examples)

Elements we must and/or should have and that might raise huge demands for storage

Final Hypothesis

$g \approx f$

Elements that we derive from our skillset and that can be computationally intensive

Hypothesis Set

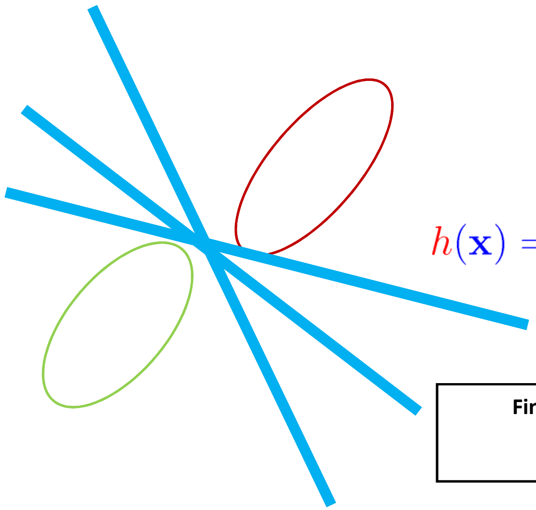
$\mathcal{H} = \{h\}; \; g \in \mathcal{H}$

(set of candidate formulas)

Elements that we derive from our skillset

Mathematical Building Blocks (2) – Our Linear Example

(decision boundaries depending on f)



$$\mathcal{H} = \{h_1, ..., h_m\};$$

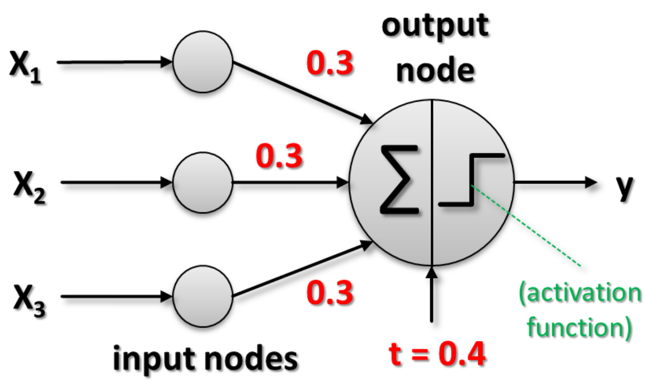
(we search a function similiar like a target function)

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right); h \in \mathcal{H}$$

Final Hypothesis

$$g \approx f$$

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right); h \in \mathcal{H}$$



Hypothesis Set

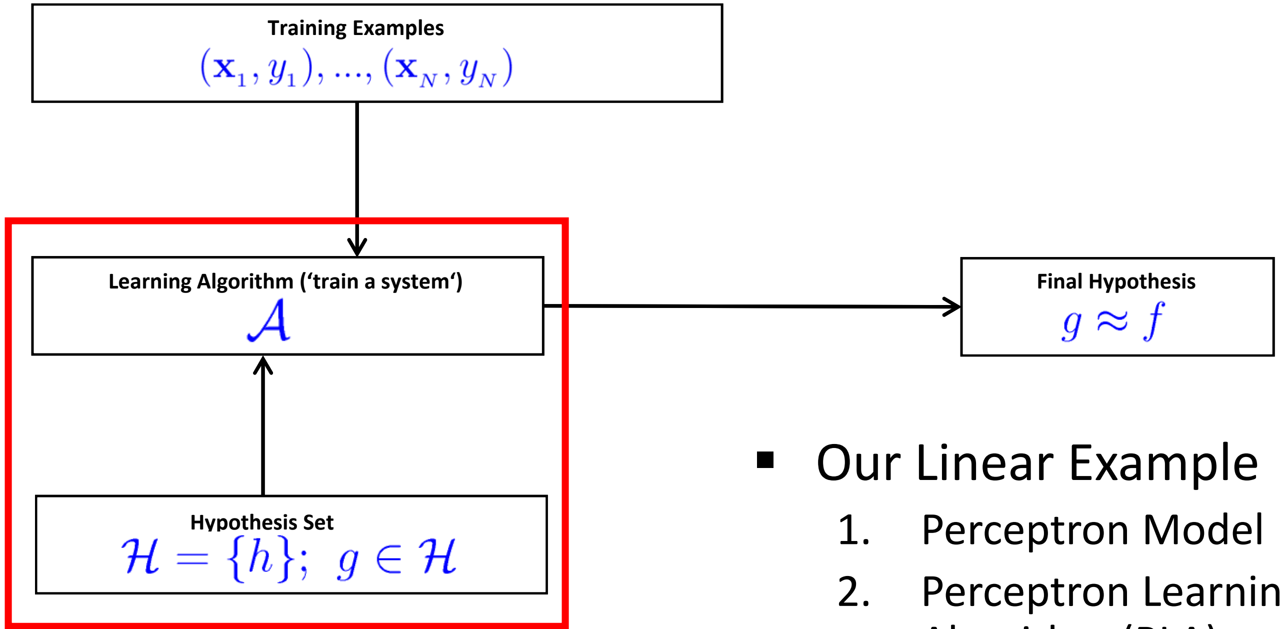
$$\mathcal{H} = \{h\};\ g \in \mathcal{H}$$

(Perceptron model – linear model)

(trained perceptron model and our selected final hypothesis)

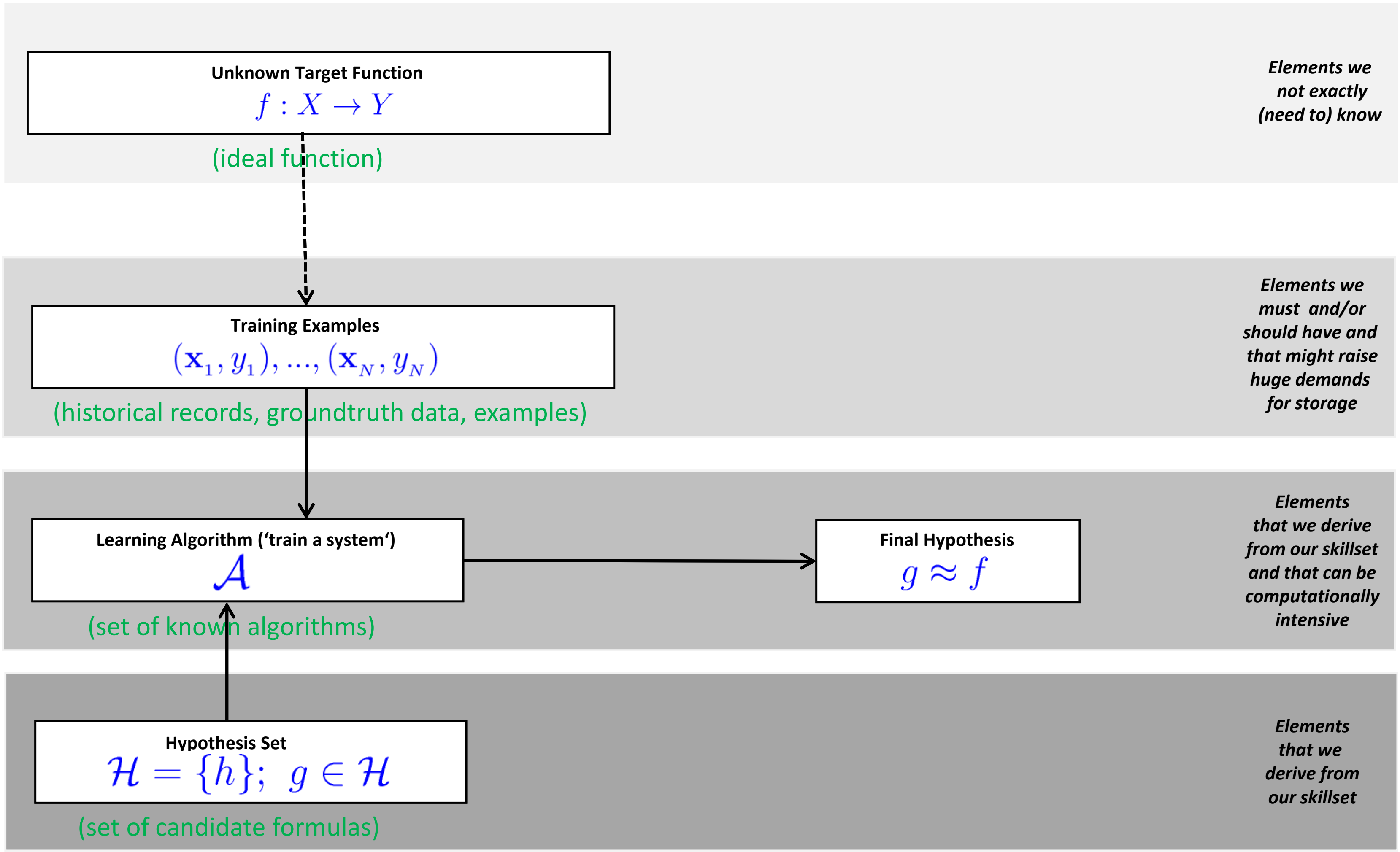
The Learning Model: Hypothesis Set & Learning Algorithm

- The solution tools – the learning model:
  1. Hypothesis set $\mathcal{H}$ - a set of candidate formulas /models
  2. Learning Algorithm $\mathcal{A}$ - ‘train a system‘ with known algorithms



‘solution tools‘

- Our Linear Example
  1. Perceptron Model
  2. Perceptron Learning Algorithm (PLA)


Unknown Target Function
$f: X \rightarrow Y$
(ideal function)
Elements we not exactly (need to) know
Training Examples
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
(historical records, groundtruth data, examples)
Elements we must and/or should have and that might raise huge demands for storage
Learning Algorithm ('train a system')
$\mathcal{A}$
(set of known algorithms)
Final Hypothesis
$g \approx f$
Elements that we derive from our skillset and that can be computationally intensive
Hypothesis Set
$\mathcal{H} = \{h\}; \; g \in \mathcal{H}$
(set of candidate formulas)
Elements that we derive from our skillset

Mathematical Building Blocks (3) – Our Linear Example

Unknown Target Function
$f : X \rightarrow Y$

(ideal function)

Training Examples
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(historical records, groundtruth data, examples)

Learning Algorithm ('train a system')
$\mathcal{A}$

(Perceptron Learning Algorithm)

Final Hypothesis
$g \approx f$

Hypothesis Set
$\mathcal{H} = \{h\}; \; g \in \mathcal{H}$

(Perceptron model – linear model)

	x1	x2	x3	y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1
5	0	0	1	-1
6	0	1	0	-1
7	0	1	1	1
8	0	0	0	-1

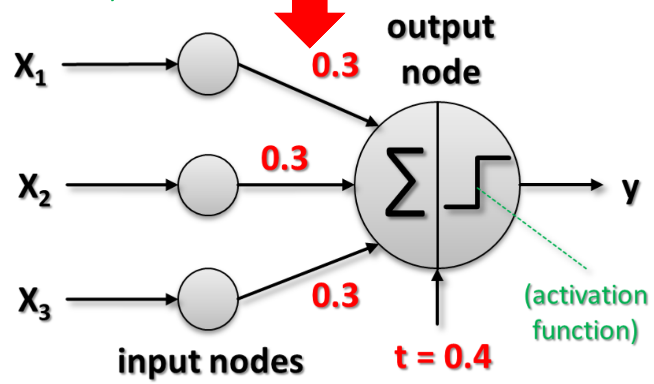
(algorithm uses training dataset)

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(training data)

$$sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right)$$

(training phase;
Find w_i and threshold
that fit the data)



(trained perceptron model
and our selected final hypothesis)

Different Models – Hypothesis Set & Unlimited 'Degrees of Freedom'

Hypothesis Set
$\mathcal{H} = \{h\};\ g \in \mathcal{H}$

$$\mathcal{H} = \{h_1, ..., h_m\};$$

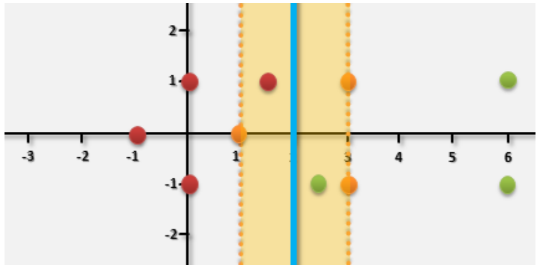
(all candidate functions
derived from models
and their parameters)

- **Choosing from various model approaches h_1, …, h_m is a different hypothesis**
- **Additionally a change in model parameters of h_1, …, h_m means a different hypothesis too**

'select one function'
that best approximates

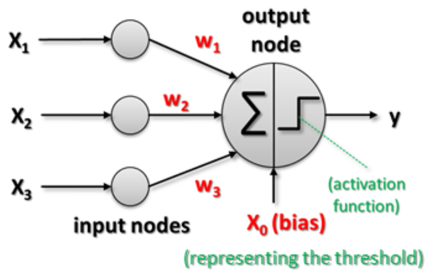
Final Hypothesis
$g \approx f$

h_1



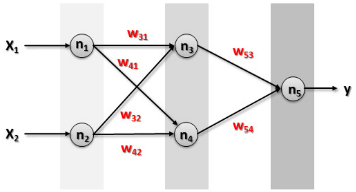
(e.g. support vector machine model)

h_2



(e.g. linear perceptron model)

h_m



(e.g. artificial neural network model)

MNIST Data – Testing on Training Dataset – Results & Discussion

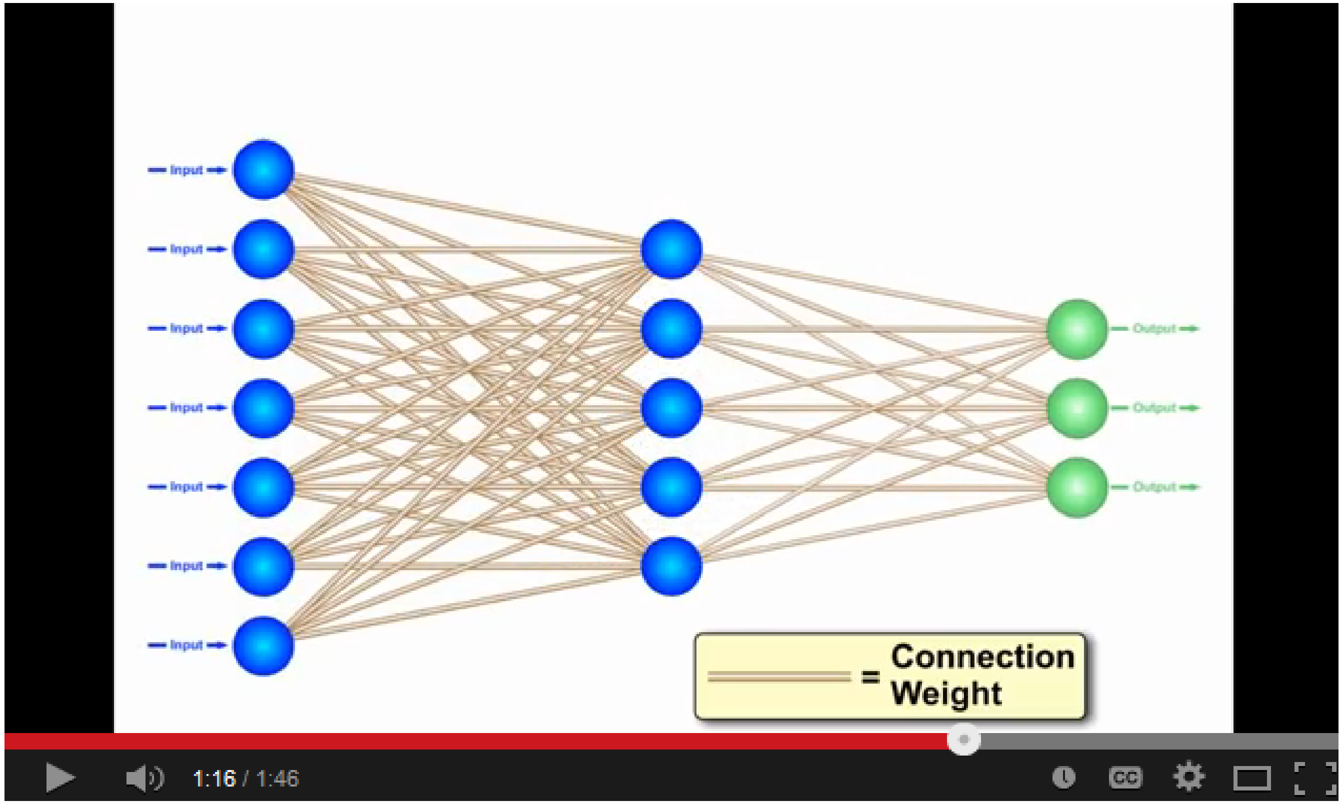
- Memorizing vs. Generalization
  → How much we got?
  - We memorize the data and created a model from it
  - No unseen data used → this no generalization!

```
10112/60000 [====>.........................] - ETA: 1s
11552/60000 [====>.........................] - ETA: 1s
12960/60000 [=====>........................] - ETA: 1s
14400/60000 [======>.......................] - ETA: 1s
15840/60000 [======>.......................] - ETA: 1s
17280/60000 [=======>......................] - ETA: 1s
18752/60000 [========>.....................] - ETA: 1s
20192/60000 [=========>....................] - ETA: 1s
21632/60000 [=========>....................] - ETA: 1s
23072/60000 [==========>...................] - ETA: 1s
24512/60000 [===========>..................] - ETA: 1s
25952/60000 [===========>..................] - ETA: 1s
27392/60000 [============>.................] - ETA: 1s
28832/60000 [=============>................] - ETA: 1s
30272/60000 [==============>...............] - ETA: 1s
31712/60000 [==============>...............] - ETA: 0s
33152/60000 [===============>..............] - ETA: 0s
34592/60000 [================>.............] - ETA: 0s
36032/60000 [=================>............] - ETA: 0s
37472/60000 [=================>............] - ETA: 0s
38912/60000 [==================>...........] - ETA: 0s
40352/60000 [===================>..........] - ETA: 0s
41792/60000 [===================>..........] - ETA: 0s
43232/60000 [====================>.........] - ETA: 0s
44672/60000 [=====================>........] - ETA: 0s
46080/60000 [======================>.......] - ETA: 0s
47520/60000 [======================>.......] - ETA: 0s
48768/60000 [=======================>......] - ETA: 0s
50208/60000 [========================>.....] - ETA: 0s
51648/60000 [========================>.....] - ETA: 0s
53088/60000 [=========================>....] - ETA: 0s
54528/60000 [==========================>...] - ETA: 0s
55968/60000 [==========================>...] - ETA: 0s
57408/60000 [===========================>..] - ETA: 0s
58848/60000 [============================>.] - ETA: 0s
60000/60000 [==============================] - 2s 35us/step
Using TensorFlow backend.
Test score: 0.035654142725043254
Test accuracy: 0.9916
```

Exercises – Train Perceptron on Testing Dataset and Test on Training Dataset

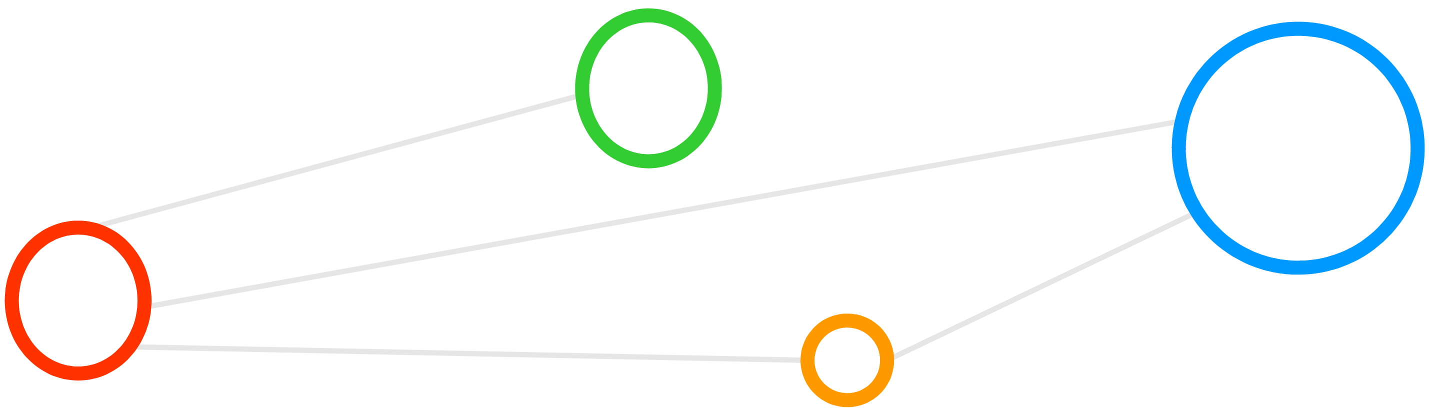
Exercises – Train on Testing Dataset & Test on Training Dataset & Increase Epochs

[Video] Neural Networks Summary



[7] YouTube Video, Neural Networks – A Simple Explanation

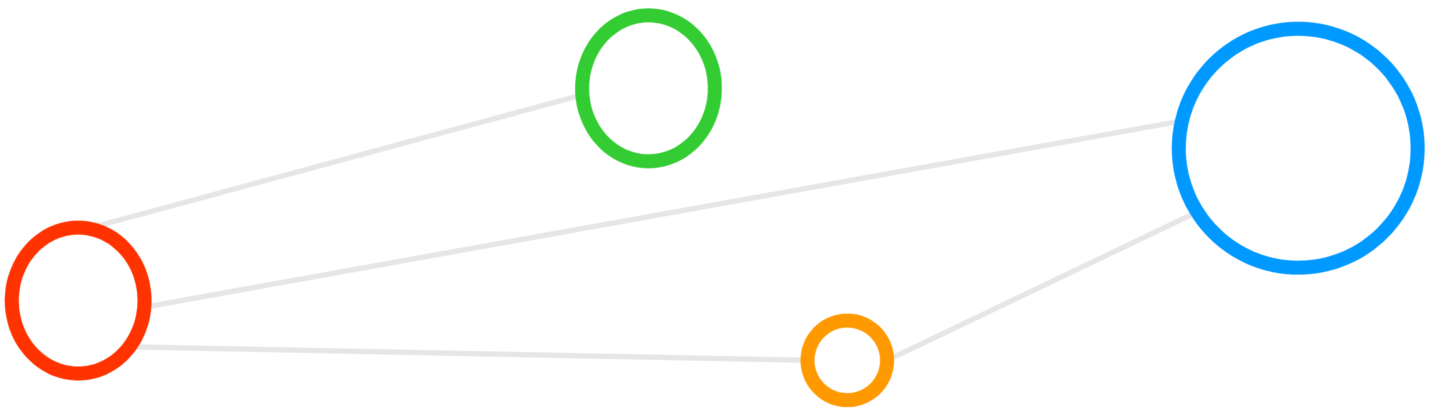
Lecture Bibliography



Lecture Bibliography

- [1] Species Iris Group of North America Database, Online: http://www.signa.org
- [2] Cheng, A.C, Lin, C.H., Juan, D.C., InstaNAS: Instance-aware Neural Architecture Search, Online: https://arxiv.org/abs/1811.10201
- [3] Tensorflow, Online: https://www.tensorflow.org/
- [4] Keras Python Deep Learning Library, Online: https://keras.io/
- [5] Jupyter Web Page, Online: https://jupyter.org/
- [6] Big Data Tips, 'Gradient Descent', Online: http://www.big-data.tips/gradient-descent
- [7] YouTube Video, 'Neural Networks, A Simple Explanation', Online: http://www.youtube.com/watch?v=gcK_5x2KsLA
- [8] YouTube Video, The Linear model with Multiple Inputs and Multiple Outputs Detail Explanation', Online: https://www.youtube.com/watch?v=zZg59p0sGVY
- [9] Scikit-Learn, Online: https://scikit-learn.org
- [10] M.Goetz, M. Riedel et al.,'HPDBSCAN – Highly Parallel DBSCAN', Proceedings of MLHPC Workshop at Supercomputing 2015, Online: https://www.researchgate.net/publication/301463871_HPDBSCAN_highly_parallel_DBSCAN

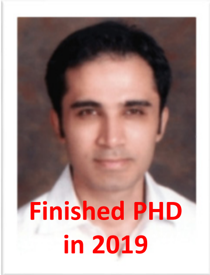
Acknowledgements



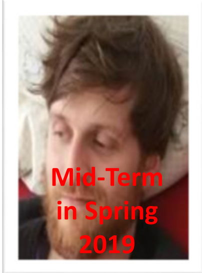
Acknowledgements – High Productivity Data Processing Research Group

PD Dr.
G. Cavallaro

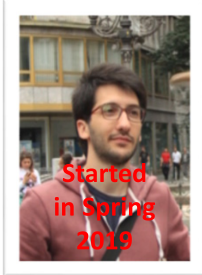
Senior PhD
Student A.S. Memon



Senior PhD
Student M.S. Memon



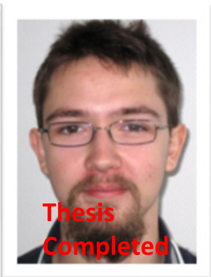
PhD Student
E. Erlingsson



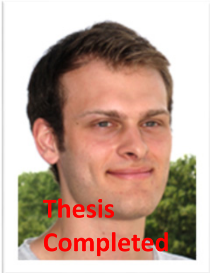
PhD Student
S. Bakarat

PhD Student
R. Sedona

Dr. M. Goetz
(now KIT)

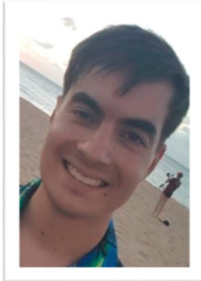


MSc M.
Richerzhagen
(now other division)



MSc
P. Glock
(now INM-1)

MSc
C. Bodenstein
(now
Soccerwatch.tv)



MSc Student
G.S. Guðmundsson
(Landsverkjun)

This research group has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 763558 (DEEP-EST EU Project)


Data
Big
Science
Research
science
analysis
methods
big
simulations
processing
computing
technologies
TB
European
computational
Computational
HPC
Juelich
Scientific
research
datasets
performance
Supercomputing
Climate
modelling
structures
databases
Centre
climate
Computing
nodes
centers
storage
ENES
EUDAT
community
Service
data-intensive
management
distributed
large-scale
access
Hardware
System
DLCLs
DLCL
services
resources
E.g
e.g
new
funding
concepts
measurement
Policy-based
cross-disciplinary
Cross-Disciplinary
forms
year
like
EU
brain
increasing
via
images
often
Enable
Modelling
Environment
scientific
important
Infrastructure
device
Analysis
Computer
expertise
disciplines
results
Key
Common
IPCC
Methods
Using
Work
Project
JSC
use
acquisition
Understanding
Germany
Provide
NASA
Energy
Simulation
systems
day
Health
directory
Earth
Technologies
AR5
project
General
Structure
compute
hundreds
Landscape
Services
network
system
stored
Resources
analyze
external
Concrete
Matters
Summary
Large
current
cores
Processing
physical
basis
files
different
DKRZ
Use
etc
PB
High
Test
fusion
within
Fusion
construction
Approaches
international
SMAQ
model
engineering
References
formats
codes
MapReduce
create
GB
Level
Nutshell
levels