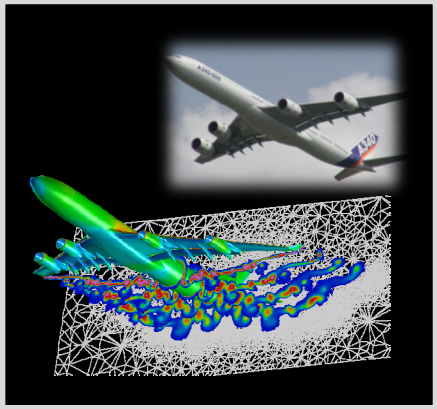


High Performance Computing

ADVANCED SCIENTIFIC COMPUTING

Prof. Dr. – Ing. Morris Riedel
Adjunct 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 8

Parallel & Scalable Machine & Deep Learning

November 04, 2019
Room V02-156

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

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

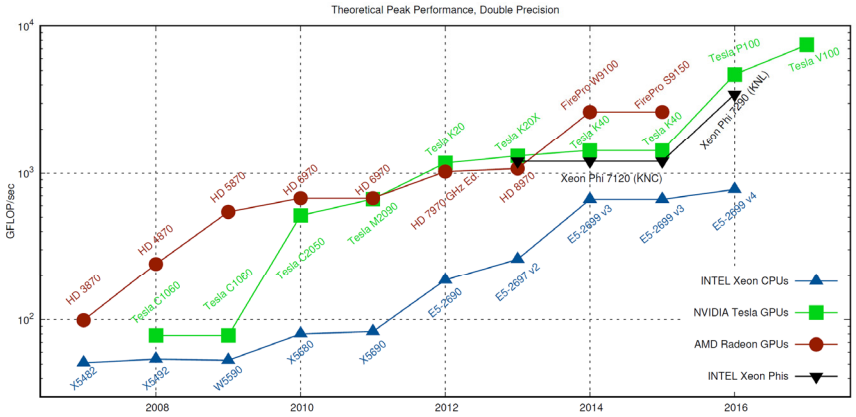
DEEP Projects

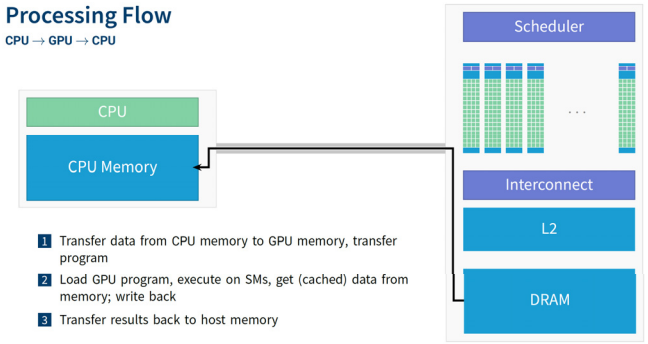
HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

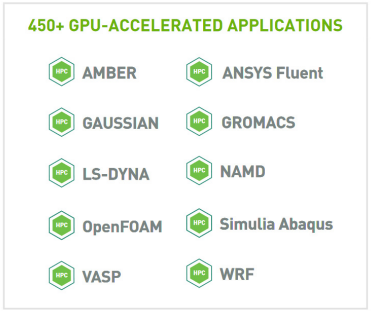
HAICU HELMHOLTZ ARTIFICIAL INTELLIGENCE COOPERATION UNIT

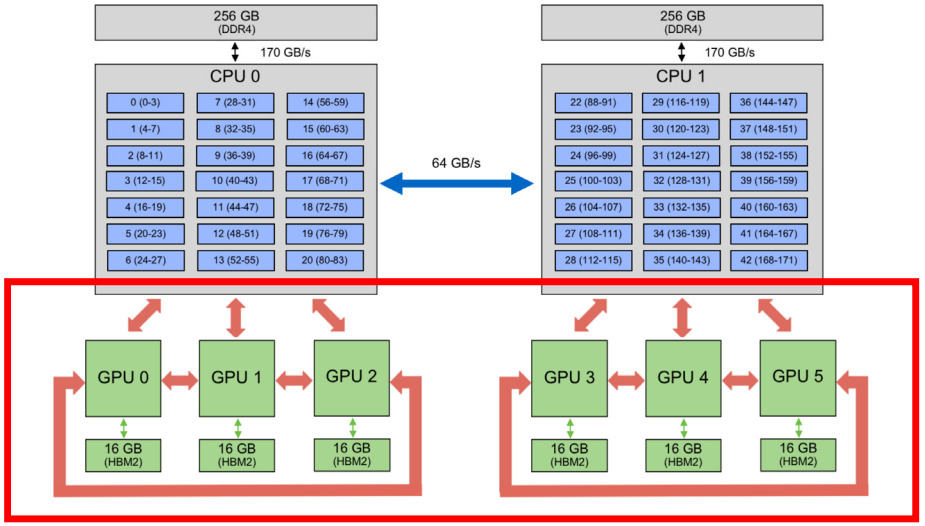
Review of Lecture 7 – Graphical Processing Units (GPUs)

- General Purpose Graphical Processing Units (GPGPUs) aka ‘GPUs‘

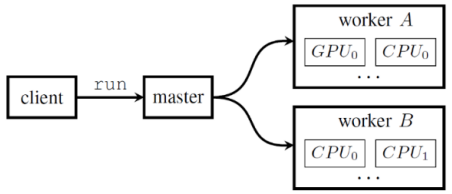








NVLink (v2) GPU Interconnect ~50 GB/s



TensorFlow

Keras

[8] CPU/GPU Comparison *[9] Summit Architecture Overview* *[3] Tensorflow Web page* *[4] Keras Web page* *[10] JSC GPU Course* *[12] NVIDIA Training*

Outline of the Course

1. High Performance Computing
2. Parallel Programming with MPI
3. Parallelization Fundamentals
4. Advanced MPI Techniques
5. Parallel Algorithms & Data Structures
6. Parallel Programming with OpenMP
7. Graphical Processing Units (GPUs)
8. Parallel & Scalable Machine & Deep Learning
9. Debugging & Profiling & Performance Toolsets
10. Hybrid Programming & Patterns
11. Scientific Visualization & Scalable Infrastructures
12. Terrestrial Systems & Climate
13. Systems Biology & Bioinformatics
14. Molecular Systems & Libraries
15. Computational Fluid Dynamics & Finite Elements
16. Epilogue

+ additional practical lectures & Webinars for our hands-on assignments in context

- Practical Topics
- Theoretical / Conceptual Topics

Outline

- Parallel & Scalable Machine Learning Techniques
  - Short Introduction to Machine Learning Approaches
  - HPDBSCAN MPI/OpenMP Implementation & Clustering
  - piSVM MPI Implementation & Land Cover Classification
  - Handwritten Character Recognition MNIST Dataset
  - Artificial Neural Networks with TensorFlow & Keras
- Parallel & Scalable Deep Learning Techniques
  - Convolutional Neural Networks via TensorFlow & Keras
  - Distributed Training via multiple GPUs with Horovod
  - Long Short-Term Memory & Autoencoder Networks
  - Neural Architecture Search via Reinforcement Learning
  - Modular Supercomputing & Data Analytics Module

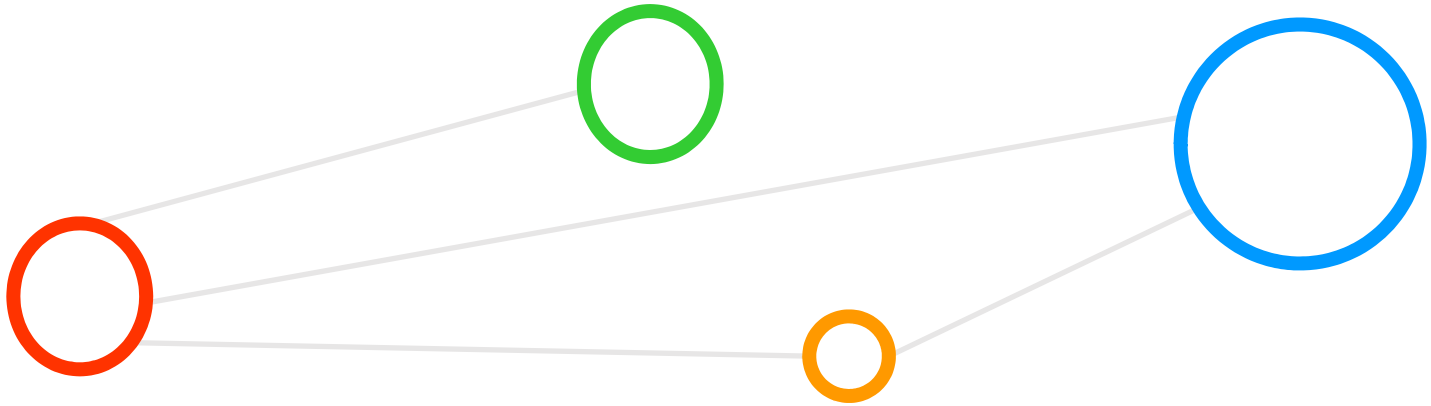
- **Promises from previous lecture(s):**
- ***Practical Lecture 0.2:*** **Lecture 8 will provide an overview of performing unsupervised learning with clustering using the parallel HPDBSCAN module**
- ***Lecture 1 & 7:*** **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms and how many-core HPC is used**
- ***Lecture 1:*** **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms and remote sensing applications**
- ***Lecture 2 & 3:*** **Lecture 8 will provide more details on MPI application examples with a particular focus on parallel and scalable machine learning**
- ***Lecture 5 & Practical Lecture 5:*** **Lecture 8 provides more details about using MPI and OpenMP for data science algorithms used in clustering and classification of data**
- ***Lecture 7:*** **Lecture 8 will provide more details about using Tensorflow & Keras in Deep Learning via Python for a wide variety of data science tasks**
- ***Lecture 7:*** **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms are used with remote sensing datasets**
- **Lecture 7: Lecture 8 will provide more details about using distributed training with Horovod & more examples of speed-ups with multi GPU usage**

Selected Learning Outcomes

- Students understand…
  - Latest developments in parallel processing & high performance computing (HPC)
  - How to create and use high-performance clusters
  - What are scalable networks & data-intensive workloads
  - The importance of domain decomposition
  - Complex aspects of parallel programming
  - HPC environment tools that support programming or analyze behaviour
  - Different abstractions of parallel computing on various levels
  - Foundations and approaches of scientific domain-specific applications

- Students are able to …
  - Programm and use HPC programming paradigms
  - Take advantage of innovative scientific computing simulations & technology
  - Work with technologies and tools to handle parallelism complexity

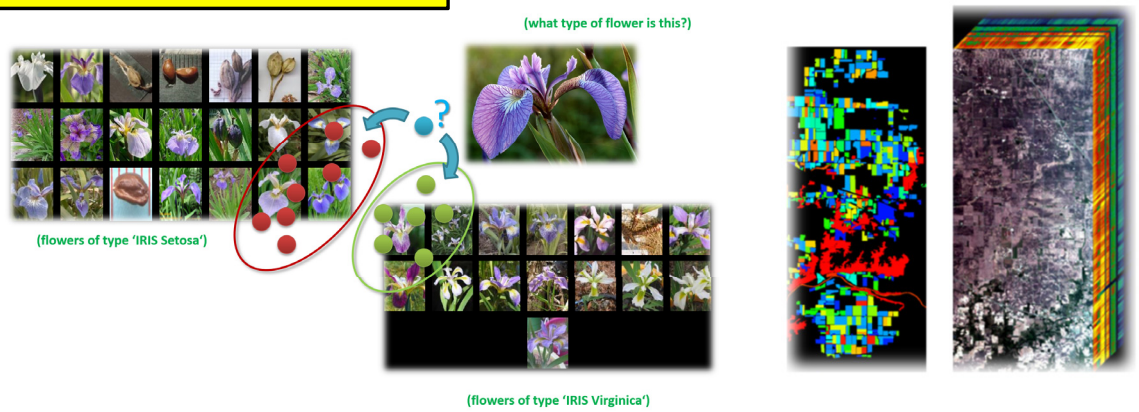
Parallel & Scalable Machine Learning Techniques



Learning Approaches – What means Learning from data?

- **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**

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

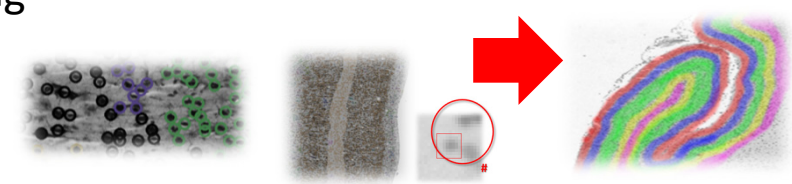


Supervised Learning

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

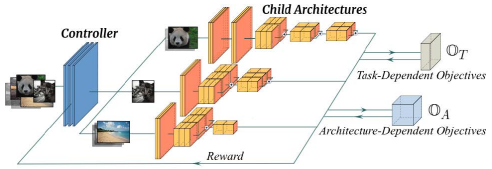
Unsupervised Learning

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



Reinforcement Learning

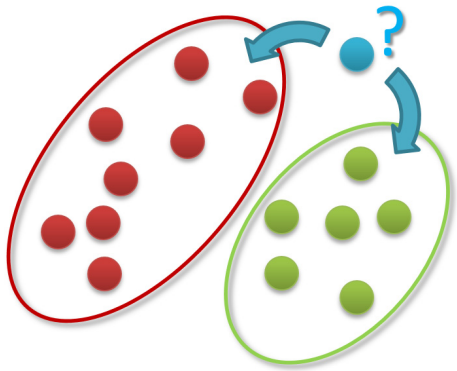
- Typical 'human way' of learning
- Example: Toddler tries to touch a hot cup of tea (again and again)



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

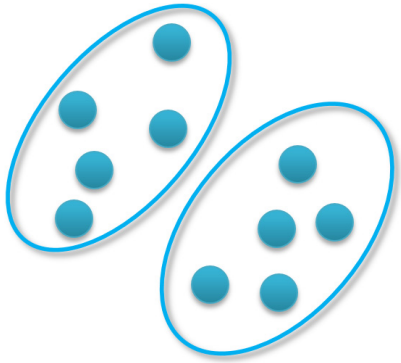
Machine Learning Models – Short Overview

Classification



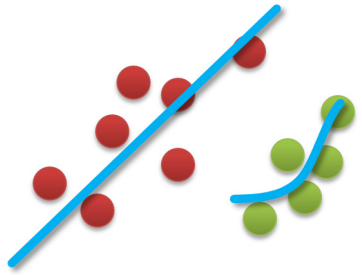
- Groups of data exist
- New data classified to existing groups

Clustering



- No groups of data exist
- Create groups from data close to each other

Regression



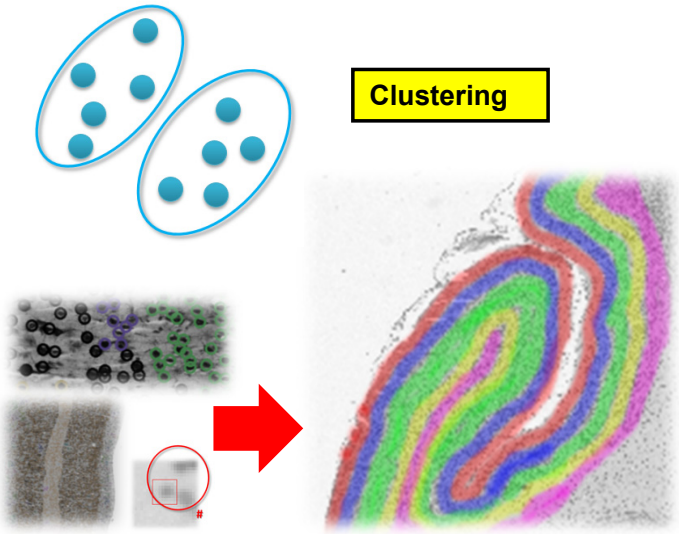
- Identify a line with a certain slope describing the data

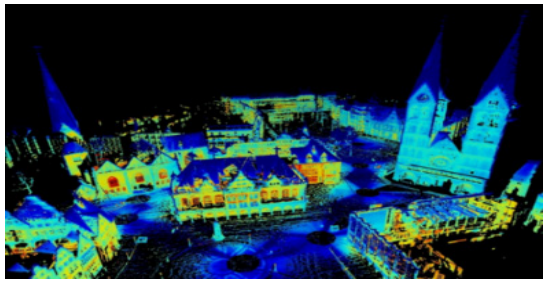
- **Machine learning methods can be roughly categorized in classification, clustering, or regression augmented with various techniques for data exploration, selection, or reduction – despite the momentum of deep learning, traditional machine learning algorithms are still widely relevant today**

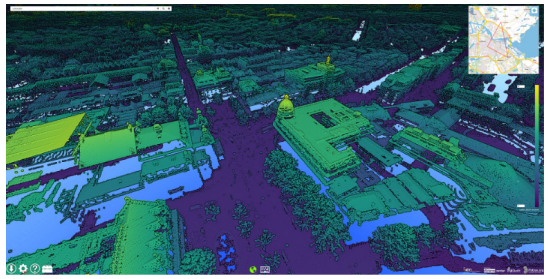
➢ **This course focus on supervised classification techniques and unsupervised clustering methods; more in complementary cloud course**

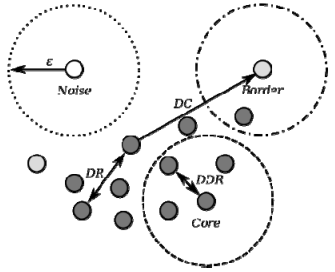
Parallel Programming with MPI & OpenMP – Data Science Applications for HPC

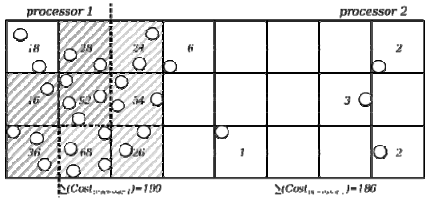
- Machine Learning Algorithms
  - Example: Highly Parallel Density-based spatial clustering of applications with noise (DBSCAN)
  - Selected Applications: Clustering different cortical layers in brain tissue & point cloud data analysis

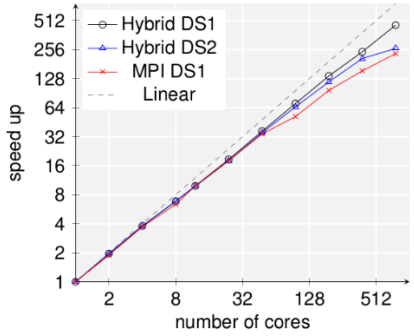








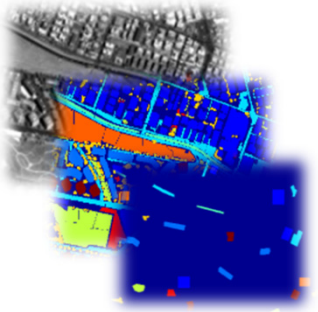




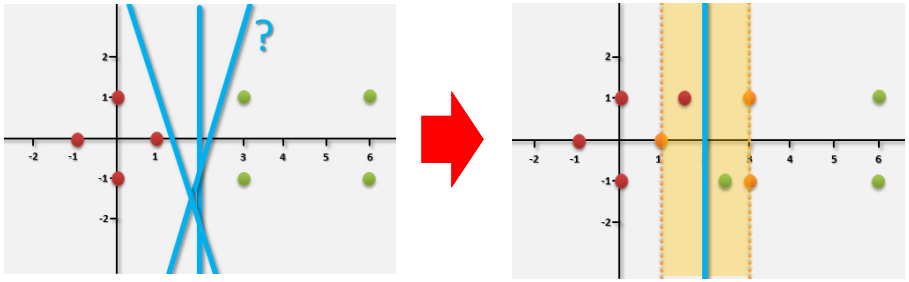
[13] M. Goetz and M. Riedel et al, Proceedings IEEE Supercomputing Conference, 2015

Parallel and Scalable Machine Learning – Parallel Support Vector Machine (SVM)

- ‘Different kind‘ of parallel algorithms
  - ‘learn from data‘ instead of modelling/approximate reality with physics
  - Parallel algorithms often useful to reduce ‘overall time for data analysis‘
- E.g. Parallel Support Vector Machines (SVMs) Technique
  - Data classification algorithm PiSVM using MPI to reduce ‘training time‘
  - Example: classification of land cover masses from satellite image data



Class	Training	Test
Buildings	18126	163129
Blocks	10982	98834
Roads	16353	147176
Light Train	1606	14454
Vegetation	6962	62655
Trees	9088	81792
Bare Soil	8127	73144
Soil	1506	13551
Tower	4792	43124
Total	77542	697859



[16] C. Cortes & V. Vapnik, ‘Support Vector Networks’, Machine Learning, 1995

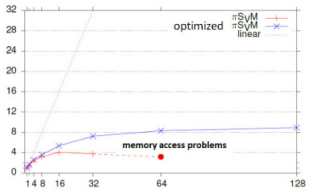
```
#!/bin/bash -x
#SBATCH--nodes=4
#SBATCH--ntasks=96
#SBATCH--ntasks-per-node=24
#SBATCH--output=mpi-out.%j
#SBATCH--error=mpi-err.%j
#SBATCH--time=04:00:00
#SBATCH--partition=batch
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=train-indianpines-4-96-24
#SBATCH--reservation=ml-hpc-2

### location executable
PISVM=/homea/hpclab/train001/tools/pisvm-1.2.1/pisvm-train

### location data
TRAINDATA=/homea/hpclab/train001/data/indianpines/indian_raw_training.el

### submit
srun $PISVM -D -o 1024 -q 512 -c 100 -g 8 -t 2 -m 1024 -s 0 $TRAINDATA
```

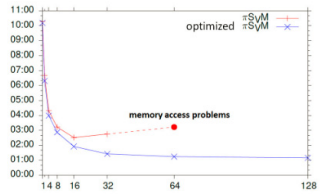
```
#!/bin/bash -x
#SBATCH--nodes=4
#SBATCH--ntasks=96
#SBATCH--ntasks-per-node=24
#SBATCH--output=mpi-out.%j
#SBATCH--error=mpi-err.%j
#SBATCH--time=04:00:00
#SBATCH--partition=batch
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=pred-indianpines-4-96-24
#SBATCH--reservation=ml-hpc-2

### location executable
PISVMPRED=/homea/hpclab/train001/tools/pisvm-1.2.1/pisvm-predict

### location data
TESTDATA=/homea/hpclab/train001/data/indianpines/indian_raw_test.el

### trained model data
MODELDATA=/homea/hpclab/train001/tools/pisvm-1.2.1/indian_raw_training.el.model

### submit
srun $PISVMPRED $TESTDATA $MODELDATA results.txt
```

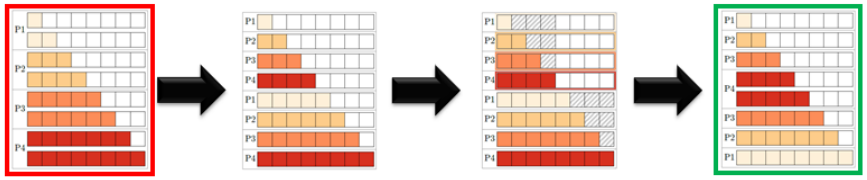
[15] G. Cavallaro & M. Riedel & J.A. Benediktsson et al., ‘On Understanding Big Data Impacts in Remotely Sensed Image Classification Using Support Vector Machine Methods’, Journal of Applied Earth Observations and Remote Sensing, 2015

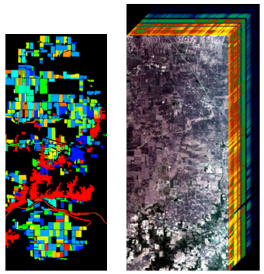
Parallel Support Vector Machine (SVM) – piSVM MPI Implementation & Impact

- Original piSVM 1.2 version (2011)
  - Open-source and based on libSVM library, C
  - Message Passing Interface (MPI)
  - New version appeared 2014-10 v. 1.3 (no major improvements)
  - Lack of ‘big data‘ support (e.g. memory)
- Tuned scalable parallel piSVM tool 1.2.1
  - Highly scalable version maintained by Juelich
  - Based on original piSVM 1.2 tool
  - Optimizations: load balancing; MPI collectives









```
//Jeder Prozess hat l Werte an die Stelle Rang * l in p_cache_status geschrieben
for(int k = 0; k < p; ++k)
{
    //Jeder Prozess broadcastet sein Ergebnis zu allen anderen Prozessen
    MPI_Bcast(&p_cache_status[k * l], l, MPI_CHAR, k, comm);
}
```

Using MPI Allgather() instead

```
for(int i = 0; i < p; ++i) {
    if(rank == i) { //Der sendende Prozess kopiert in den Sendebereich
        for(int j = 0; j < lmn; ++j)
            G_buf[j] = G_n[j];
    }
    //Alle anderen Prozesse erhalten die Daten
    MPI_Bcast(G_buf, lmn, MPI_DOUBLE, i, comm);
    //Und addieren sie auf
    for(int j = 0; j < lmn; ++j)
        G[not_work_set[j]] += G_buf[j];
}
```

Using MPI_Allreduce() instead

Scenario ‘pre-processed data‘, 10xCV **serial**: accuracy (min)

γ/C	1	10	100	1000	10 000
2	48.90 (18.81)	65.01 (19.57)	73.21 (20.11)	75.55 (22.53)	74.42 (21.21)
4	57.53 (16.82)	70.74 (13.94)	75.94 (13.53)	76.04 (14.04)	74.06 (15.55)
8	64.18 (18.30)	74.45 (15.04)	**77.00** (14.41)	75.78 (14.65)	74.58 (14.92)
16	68.37 (23.21)	76.20 (21.88)	76.51 (20.69)	75.32 (19.60)	74.72 (19.66)
32	70.17 (34.45)	75.48 (34.76)	74.88 (34.05)	74.08 (34.03)	73.84 (38.78)

Scenario ‘pre-processed data‘, 10xCV **parallel**: accuracy (min)

γ/C	1	10	100	1000	10 000
2	75.26 (1.02)	65.12 (1.03)	73.18 (1.33)	75.76 (2.35)	74.53 (4.40)
4	57.60 (1.03)	70.88 (1.02)	75.87 (1.03)	76.01 (1.33)	74.06 (2.35)
8	64.17 (1.02)	74.52 (1.03)	**77.02 (1.02)**	75.79 (1.04)	74.42 (1.34)
16	68.57 (1.33)	76.07 (1.33)	76.40 (1.34)	75.26 (1.05)	74.53 (1.34)
32	70.21 (1.33)	75.38 (1.34)	74.69 (1.34)	73.91 (1.47)	73.73 (1.33)

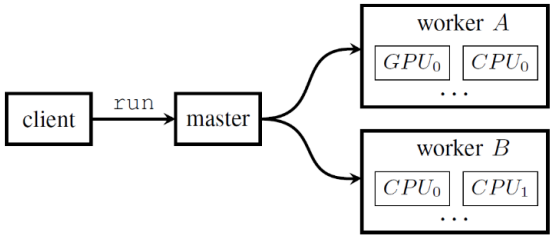
First Result: best parameter set from 14.41 min to 1.02 min
Second Result: all parameter sets from ~9 hours to ~35 min

[15] G. Cavallaro & M. Riedel & J.A. Benediktsson et al., ‘On Understanding Big Data Impacts in Remotely Sensed Image Classification Using Support Vector Machine Methods’, Journal of Applied Earth Observations and Remote Sensing

[17] piSVM on SourceForge, 2008

Deep Learning Frameworks using GPUs also good for Artificial Neural Networks

- TensorFlow (cf. Lecture 7)
  - One of the most popular deep learning frameworks available today
  - Execution on multi-core CPUs or many-core GPUs



- **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**

TensorFlow

[3] Tensorflow Web page

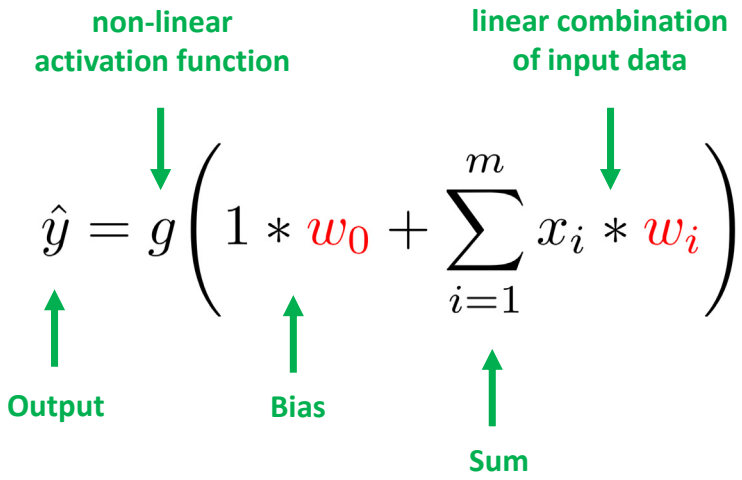
- Keras (cf. Lecture 7)
  - Often used in combination with low-level frameworks like Tensorflow

Keras

[4] Keras Web page

- **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**

Perceptron Model – Mathematical Notation for one Neuron



$$\hat{y} = g\left(w_0 + X^T \mathbf{w}\right)$$

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)

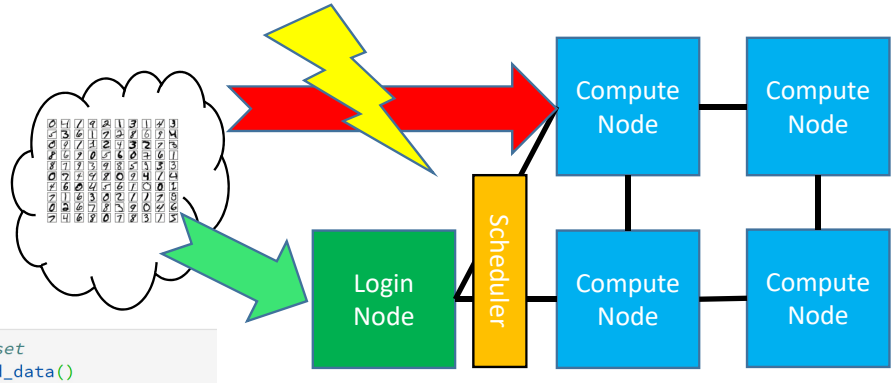
```python
import numpy as np
from keras.datasets import mnist

# 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
```

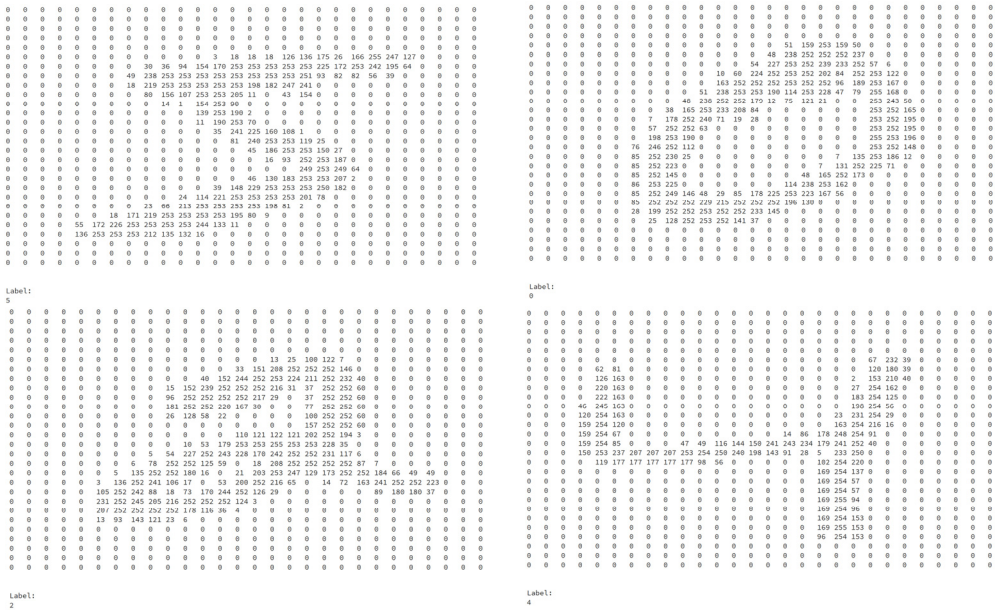
```
import numpy as np
from keras.datasets import mnist
```

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

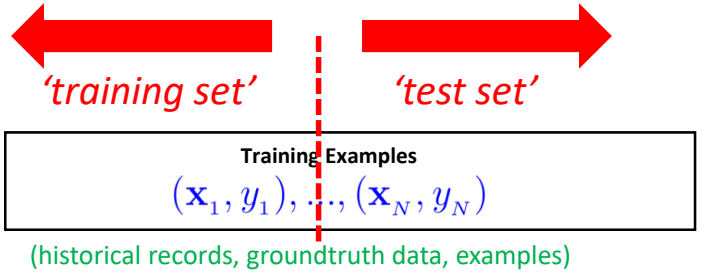
(downloads data into ~home/.keras/datasets as NPZ file format of numpy that provides storage of array data using gzip compression)

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()
```

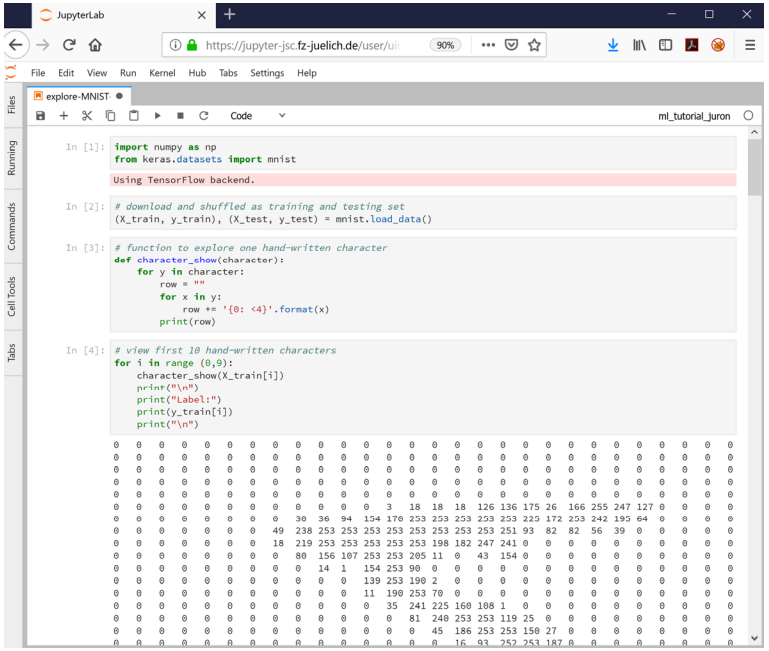
- Loading MNIST training datasets (X) with labels (Y) stored in a binary numpy format
- Format is 28 x 28 pixel values with grey level from 0 (white background) to 255 (black foreground)

```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)
```

- 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)

```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")
```

- 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)



[2] Jupyter Web Page

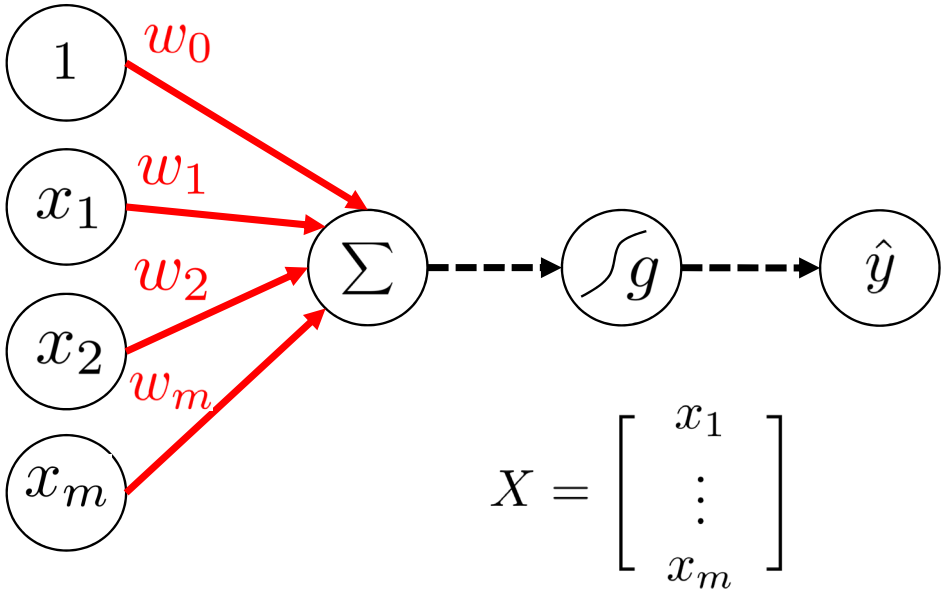
MNIST Dataset with Perceptron Learning Model – Need for Reshape

- 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

```
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   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  238 253 253 253 253 253 253 253 253 253 251 93  82  82  56  39  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

- **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**



MNIST Dataset – Reshape & Normalization – Example

(one long input vector with length 784)

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

(numbers are between 0 and 1)

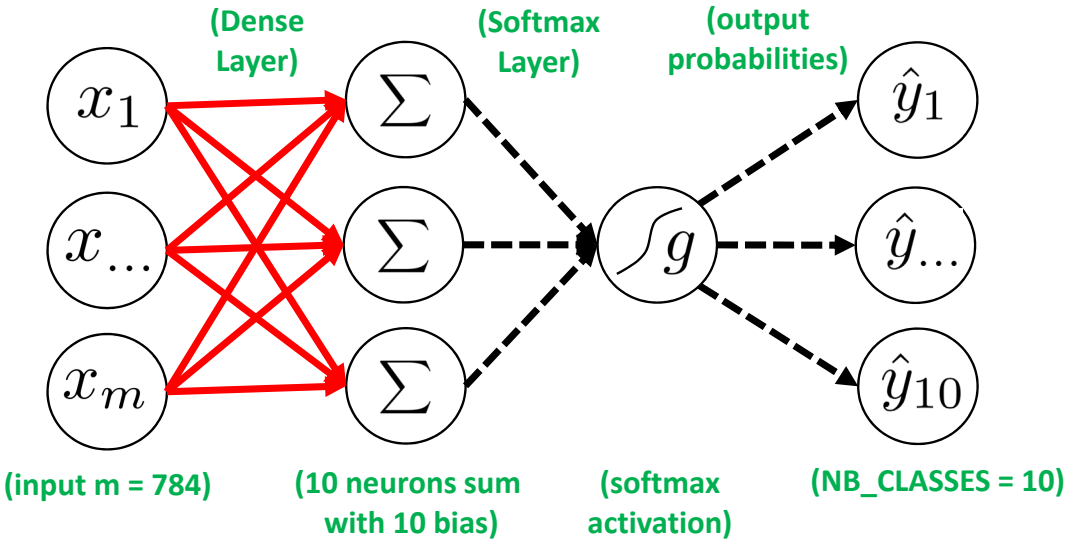
```
784 input pixel values per train samples
784 input pixel values per test samples
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         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
```

(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
```

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

# 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()
```

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

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

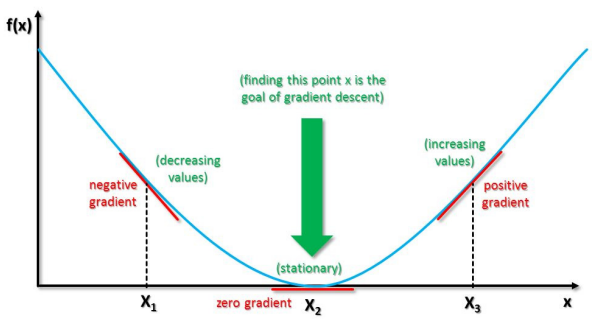
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0

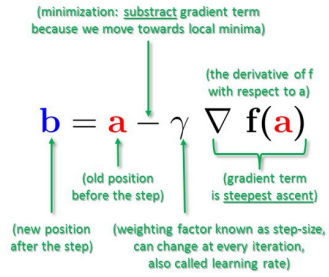
- **Note that the output units are independent among each other in contrast to neural networks with one hidden layer**
- **The output of softmax gives class probabilities**
- **The non-linear Activation function 'softmax' represents a generalization of the sigmoid function – it squashes an n-dimensional 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**

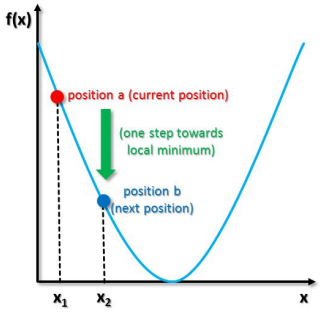
MNIST Dataset & Compile Multi Output Perceptron Model

- 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

```
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)**

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

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

[5] 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
```

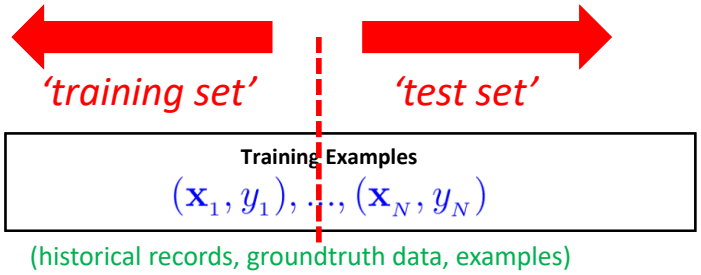
```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimization technique
VERBOSE = 1
```

```
# 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]**



(historical records, groundtruth data, examples)

➢ **Assignment #2 will explore the change of parameters in context of changes in running time when training models on GPUs vs. CPUs**

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

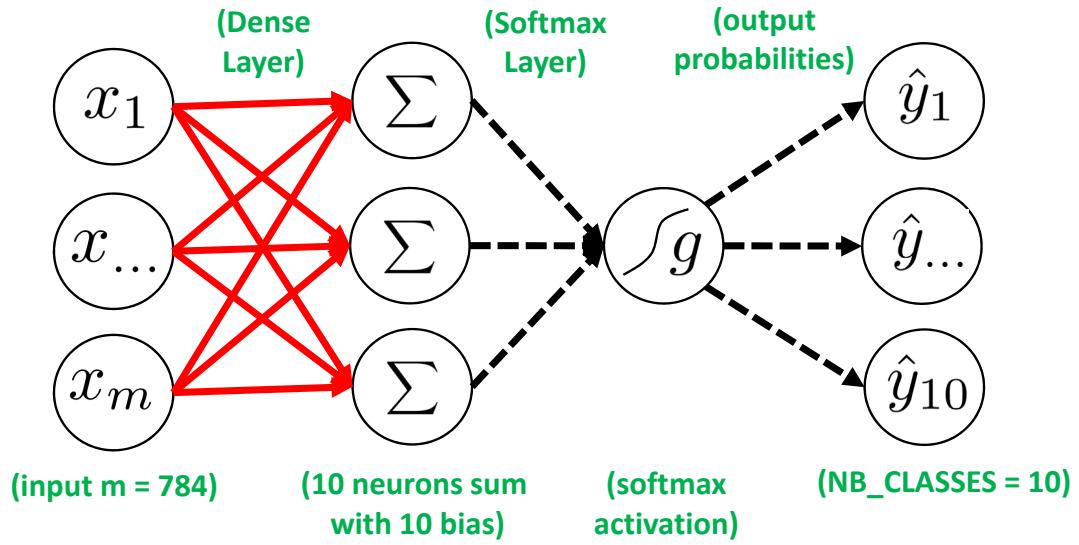
➢ **Assignment #2 will explore the change of parameters in context of changes in running time when training models on GPUs vs. CPUs**

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 Softmay (cf. Day One) → output probability

MNIST Dataset – Add Two Hidden Layers for Artificial Neural Network (ANN)

- All parameter value remain the same as before
- We add N_HIDDEN as parameter in order to set 128 neurons in one hidden layer – this number is a hyperparameter that is not directly defined and needs to be find with parameter search

```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimization technique
VERBOSE = 1
N_HIDDEN = 128 # number of neurons in one hidden layer
```

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

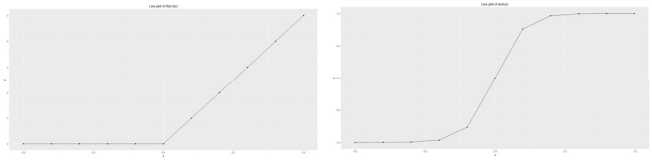
```
# modeling step
# 2 hidden layers each N_HIDDEN neurons
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dense(NB_CLASSES))
```

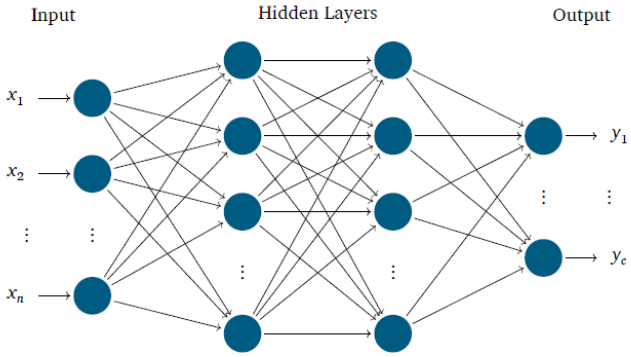
```
# add activation function layer to get class probabilities
model.add(Activation('softmax'))
```

[6] big-data.tips, 'Relu Neural Network'

[7] big-data.tips, 'tanh'

(activation functions ReLU & Tanh)






```
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
```

```
model.add(Dense(N_HIDDEN))
model.add(Activation('tanh'))
```

- The non-linear Activation function 'relu' represents a so-called Rectified Linear Unit (ReLU) that only recently became very popular because it generates good experimental results in ANNs and more recent deep learning models – it just returns 0 for negative values and grows linearly for only positive values
- A hidden layer in an ANN can be represented by a fully connected Dense layer in Keras by just specifying the number of hidden neurons in the hidden layer

➢ **Assignment #2 will explore the change of parameters in context of changes in running time when training models on GPUs vs. CPUs**

MNIST Dataset – ANN Model Parameters & Output Evaluation

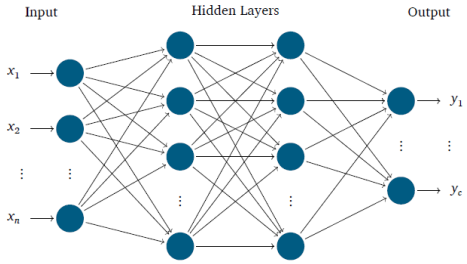
```
Epoch 7/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2743 - acc: 0.9223
Epoch 8/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2601 - acc: 0.9266
Epoch 9/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2477 - acc: 0.9301
Epoch 10/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2365 - acc: 0.9329
Epoch 11/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2264 - acc: 0.9356
Epoch 12/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2175 - acc: 0.9386
Epoch 13/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2092 - acc: 0.9412
Epoch 14/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2013 - acc: 0.9432
Epoch 15/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1942 - acc: 0.9454
Epoch 16/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1876 - acc: 0.9472
Epoch 17/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1813 - acc: 0.9487
Epoch 18/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1754 - acc: 0.9502
Epoch 19/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1700 - acc: 0.9522
Epoch 20/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1647 - acc: 0.9536
```

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

```
10000/10000 [==============================] - 0s 33us/step
Test score: 0.16286438911408185
Test accuracy: 0.9514
```

- ✓ **Multi Output Perceptron: ~91,01% (20 Epochs)**
- ✓ **ANN 2 Hidden Layers: ~95,14 % (20 Epochs)**

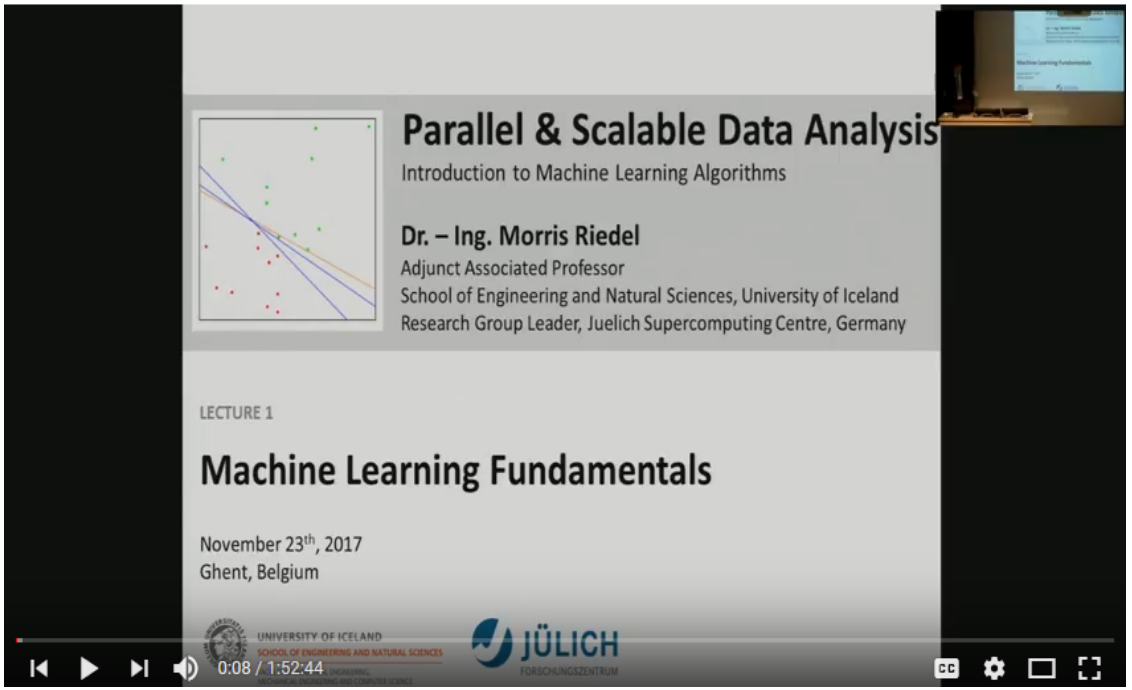
```
# printout a summary of the model to understand model complexity
model.summary()
```




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
```

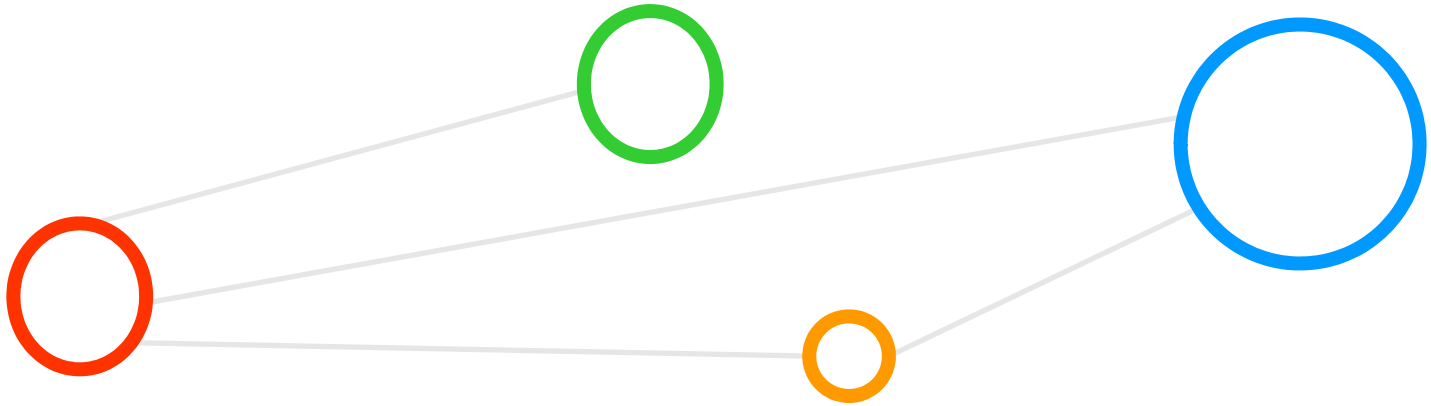
- **Dense Layer connects every neuron in this dense layer to the next dense layer with each of its neuron also called a fully connected network element with weights as trainiable parameters**
- **Choosing a model with different layers is a model selection that directly also influences the number of parameters (e.g. add Dense layer from Keras means new weights)**
- **Adding a layer with these new weights means much more computational complexity since each of the weights must be trained in each epoch (depending on #neurons in layer)**

[YouTube Lectures] More Details about Machine Learning Basics



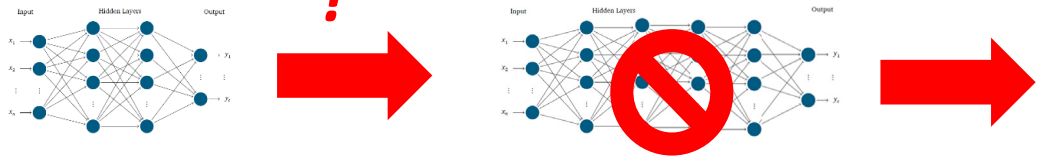
[1] Morris Riedel, 'Introduction to Machine Learning Algorithms', Invited YouTube Lecture, six lectures, University of Ghent, 2017

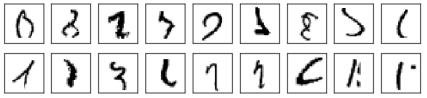
Parallel & Scalable Deep Learning Techniques



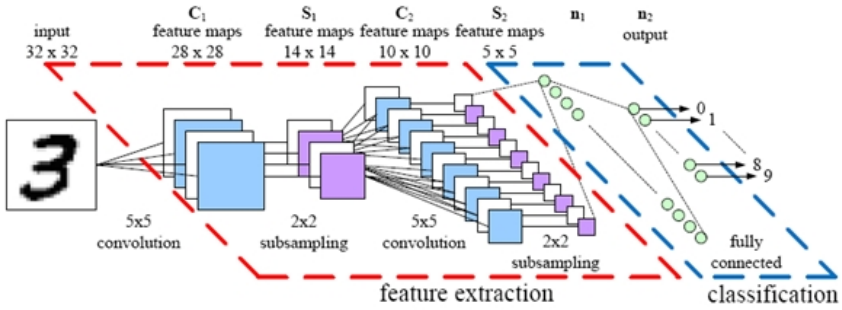
DEEP Learning takes advantage of Many-Core Technologies (cf. Lecture 1 & 7)

[4] Neural Network 3D Simulation





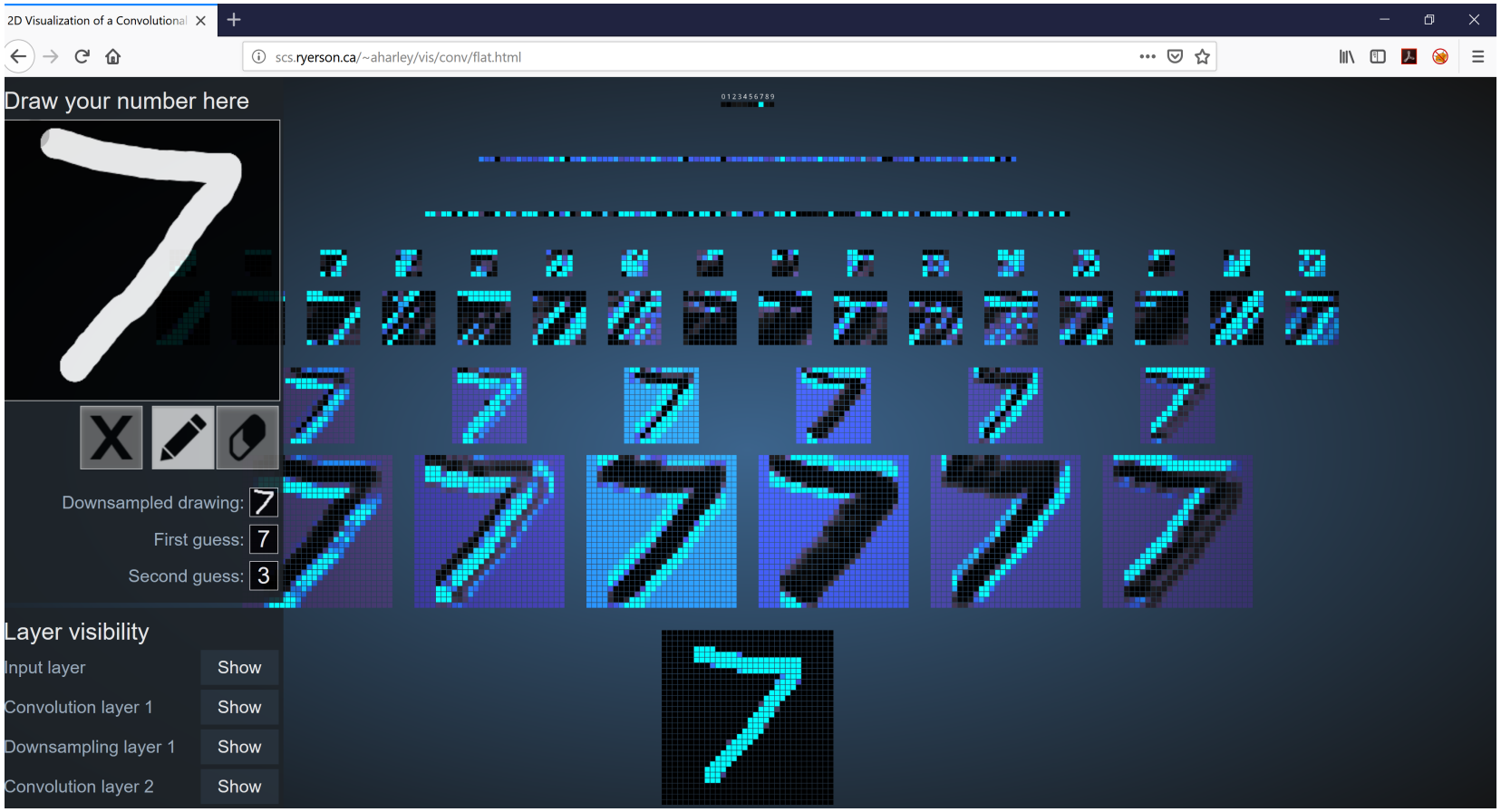
- **Innovation via specific layers and architecture types**



[5] A. Rosebrock

➢ **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms and how many-core HPC is used**

Understanding Feature Maps & Convolutions – Online Web Tool



[18] Harley, A.W., An Interactive Node-Link Visualization of Convolutional Neural Networks

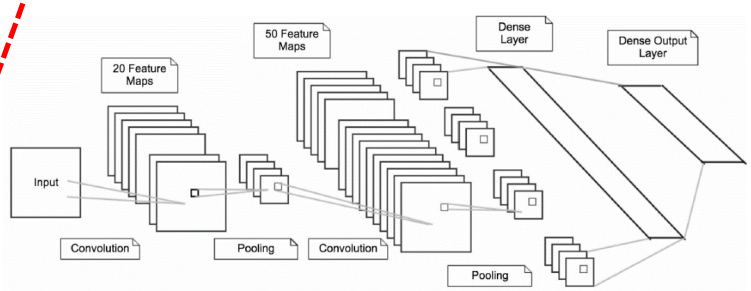
MNIST Dataset – Convolutional Neural Network (CNN) Model

```
from keras import backend as K
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.datasets import mnist
from keras.utils import np_utils
from keras.optimizers import SGD, RMSprop, Adam
import numpy as np
import matplotlib.pyplot as plt
```

```
#define the CNN model
class CNN:
  @staticmethod
  def build(input_shape, classes):
    model = Sequential()
    # CONV => RELU => POOL
    model.add(Conv2D(20, kernel_size=5, padding="same",
    input_shape=input_shape))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    # CONV => RELU => POOL
    model.add(Conv2D(50, kernel_size=5, border_mode="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    # Flatten => RELU layers
    model.add(Flatten())
    model.add(Dense(500))
    model.add(Activation("relu"))
    # a softmax classifier
    model.add(Dense(classes))
    model.add(Activation("softmax"))
    return model
```

- **Increasing the number of filters learned to 50 in the next layer from 20 in the first layer**
- **Increasing the number of filters in deeper layers is a common technique in deep learning architecture modeling**
- **Flattening the output as input for a Dense layer (fully connected layer)**
- **Fully connected / Dense layer responsible with softmay activation for classification based on learned filters and features**

[19] A. Gulli et al.




```
# initialize the optimizer and model
model = CNN.build(input_shape=INPUT_SHAPE, classes=NB_CLASSES)
model.compile(loss="categorical_crossentropy", optimizer=OPTIMIZER,
metrics=["accuracy"])
```

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

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 20, 28, 28)	520
activation_1 (Activation)	(None, 20, 28, 28)	0
max_pooling2d_1 (MaxPooling2	(None, 20, 14, 14)	0
conv2d_2 (Conv2D)	(None, 50, 14, 14)	25050
activation_2 (Activation)	(None, 50, 14, 14)	0
max_pooling2d_2 (MaxPooling2	(None, 50, 7, 7)	0
flatten_1 (Flatten)	(None, 2450)	0
dense_1 (Dense)	(None, 500)	1225500
activation_3 (Activation)	(None, 500)	0
dense_2 (Dense)	(None, 10)	5010
activation_4 (Activation)	(None, 10)	0

```
Total params: 1,256,080
Trainable params: 1,256,080
Non-trainable params: 0
```

MNIST Dataset – Model Parameters & 2D Input Data

```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
VERBOSE = 1
OPTIMIZER = Adam()
VALIDATION_SPLIT=0.2
IMG_ROWS, IMG_COLS = 28, 28 # input image dimensions
NB_CLASSES = 10 # number of outputs = number of digits
INPUT_SHAPE = (1, IMG_ROWS, IMG_COLS)
```

```
# data: shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
K.set_image_dim_ordering("th")
# consider them as float and normalize
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
# we need a 60K x [1 x 28 x 28] shape as input to the CONVNET
X_train = X_train[:, np.newaxis, :, :]
X_test = X_test[:, np.newaxis, :, :]
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
# convert class vectors to binary class matrices
y_train = np_utils.to_categorical(y_train, NB_CLASSES)
y_test = np_utils.to_categorical(y_test, NB_CLASSES)
```

- **OPTIMIZER: Adam - advanced optimization technique that includes the concept of a momentum (a certain velocity component) in addition to the acceleration component of Stochastic Gradient Descent (SGD)**
- **Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients**
- **Adam enables faster convergence at the cost of more computation and is currently recommended as the default algorithm to use (or SGD + Nesterov Momentum)**

[11] D. Kingma et al., 'Adam: A Method for Stochastic Optimization'

- **Compared to the Multi-Output Perceptron and Artificial Neural Networks (ANN) model, the input dataset remains as 2d matricew with 1 x 28 x 28 per image, including also the class vectors that are converted to binary class matrices**

➢ **Assignment #2 will explore the change of parameters in context of changes in running time when training models on GPUs vs. CPUs**

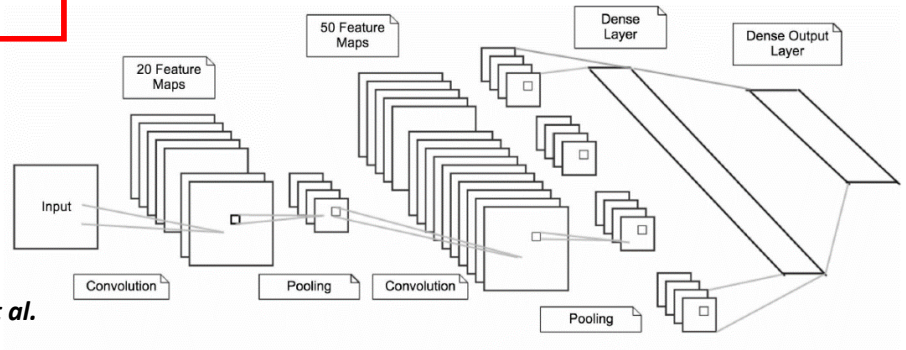
MNIST Dataset – CNN Model Output & Evaluation

```
Epoch 14/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0065 - acc: 0.9980 - val_loss: 0.0346 - val_acc: 0.9921
Epoch 15/20
48000/48000 [==============================] - 4s 89us/step - loss: 0.0030 - acc: 0.9990 - val_loss: 0.0418 - val_acc: 0.9903
Epoch 16/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0057 - acc: 0.9980 - val_loss: 0.0470 - val_acc: 0.9910
Epoch 17/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0043 - acc: 0.9985 - val_loss: 0.0440 - val_acc: 0.9906
Epoch 18/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0046 - acc: 0.9985 - val_loss: 0.0474 - val_acc: 0.9891
Epoch 19/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0047 - acc: 0.9986 - val_loss: 0.0353 - val_acc: 0.9928
Epoch 20/20
48000/48000 [==============================] - 4s 88us/step - loss: 3.4055e-04 - acc: 1.0000 - val_loss: 0.0374 - val_acc: 0.9927
```

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

```
10000/10000 [==============================] - 1s 70us/step
Test score: 0.0303058747581508
Test accuracy: 0.9936
```

- ✓ **Multi Output Perceptron: ~91,01% (20 Epochs)**
- ✓ **ANN 2 Hidden Layers: ~95,14 % (20 Epochs)**
- ✓ **CNN Deep Learning Model: ~99,36 % (20 Epochs)**



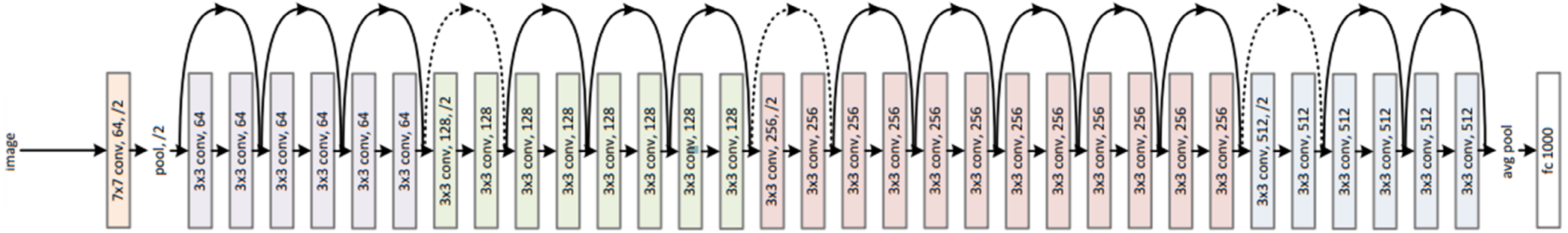
[19] A. Gulli et al.

Why not 100%

some samples even for a human unrecognizable

More Computation: Deep Learning via RESNET-50 Architecture (cf. Lecture 7)

- Classification of land cover in scenes (cf. Invited Talk G. Cavallaro)
  - Very suitable for parallelization via distributed training on multi GPUs

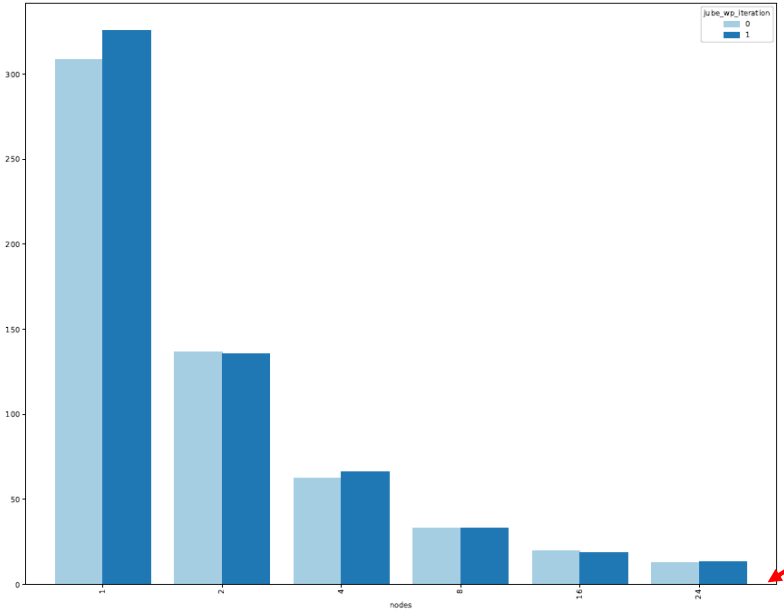


[21] RESNET

- **RESNET-50 is a known neural network architecture that has established a strong baseline in terms of accuracy**
- **The computational complexity of training the RESNET-50 architecture relies in the fact that is has ~ 25.6 millions of trainable parameters**
- **RESNET-50 still represents a good trade-off between accuracy, depth and number of parameters**
- **The setups of RESNET-50 makes it very suitable for parallelization via distributed training on multi GPUs**

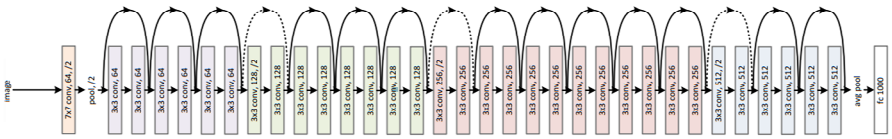
Distributed Training via Multi GPUs with Horovod – Remote Sensing Example

Time per epoch [sec]

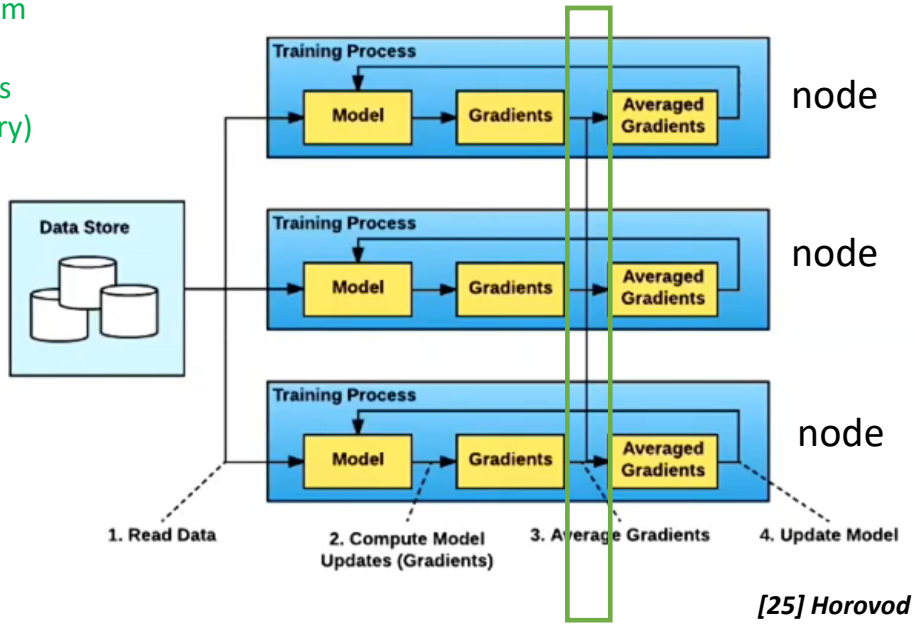


A partition of the JUWELS system has 56 compute nodes, each with 4 NVIDIA V100 GPUs (equipped with 16 GB of memory)

24 nodes x 4 GPUs = 96 GPUs



[21] RESNET



[25] Horovod

MPI_Allreduce()

- **Horovod is a distributed training framework used in combination with low-level deep learning frameworks like Tensorflow**
- **Horovod uses MPI for inter-process communication, e.g., MPI_Allreduce()**
- **Distributed training using data parallelism approach means: (1) Gradients for different batches of data are calculated separately on each node; (2) But averaged across nodes to apply consistent updated to the deep learning model in each node**

Distributed Training via Multi GPUs with Horovod – ImageNet Example

- Dataset: ImageNet
  - Total number of images: 14.197.122
  - Images with bounding box annotations: 1.034.908

(huge collection of images with high level categories)

High level category	# synset (subcategories)	Avg # images per synset	Total # images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K
geological formation	151	838	127K
invertebrate	728	573	417K
mammal	1138	821	934K
musical instrument	157	891	140K
plant	1666	600	999K
reptile	268	707	190K
sport	166	1207	200K
structure	1239	763	946K
tool	316	551	174K
tree	993	568	564K
utensil	86	912	78K
vegetable	176	764	135K
vehicle	481	778	374K
person	2035	468	952K

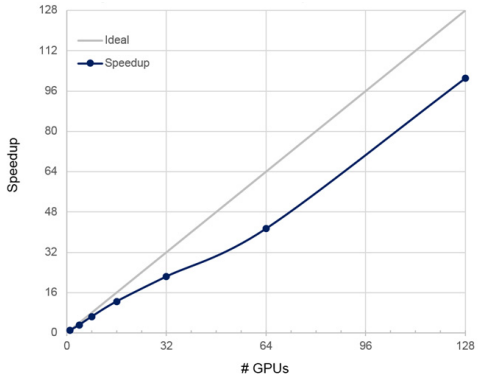
- **Open source tool Horovod enables distributed deep learning with TensorFlow / Keras**
- **Machine & Deep Learning: speed-up is just secondary goal after 1st goal accuracy**
- **Speed-up & parallelization good for faster hyperparameter tuning, training, inference**
- **Third goal is to avoid much feature engineering through 'feature learning'**

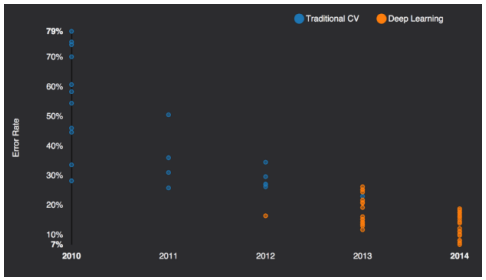
(ImageNet as a benchmark in deep learning community)

(setup 1.2 Mio Images 224x224 pixels: TensorFlow 1.4, Python 2.7, CUDA 8, cuDNN 6, Horovod 0.11.2, MVAPICH-2.2-GDR on JURECA K80 GPUs)

#GPUs	images/s	speedup	Performance per GPU [images/s]
1	55	1.0	55
4	178	3.2	44.5
8	357	6.5	44.63
16	689	12.5	43.06
32	1230	22.4	38.44
64	2276	41.4	35.56
128	5562	101.1	43.45

[25] Horovod

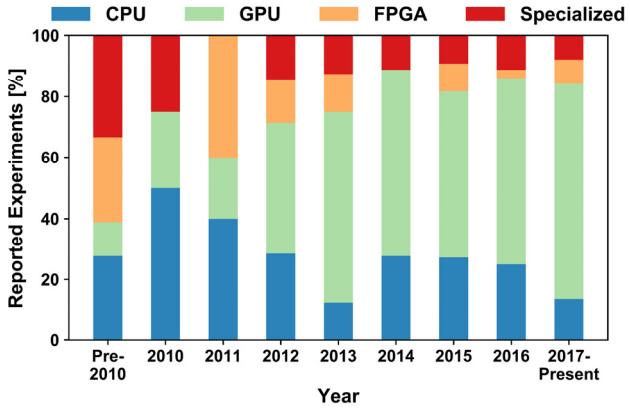


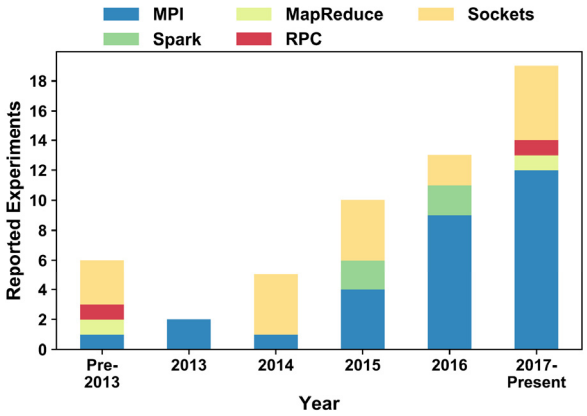


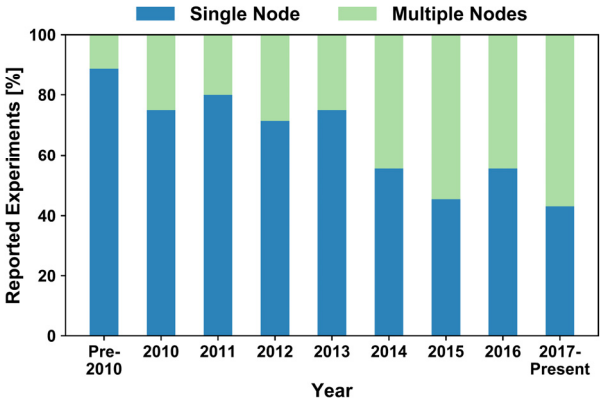
[22] J. Dean et al., 'Large-Scale Deep Learning'

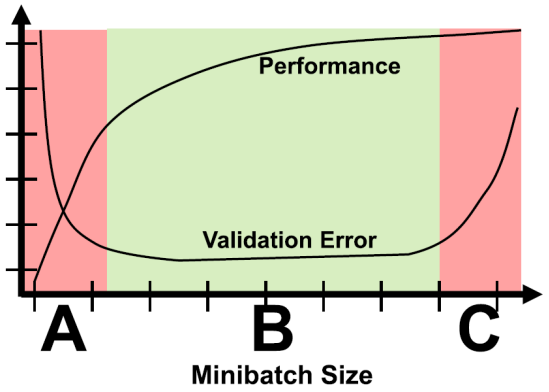
[23] ImageNet Web page

Parallel Computing & HPC using GPUs for Deep Learning – Selected Impacts

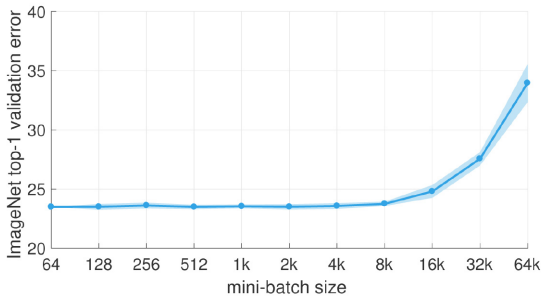








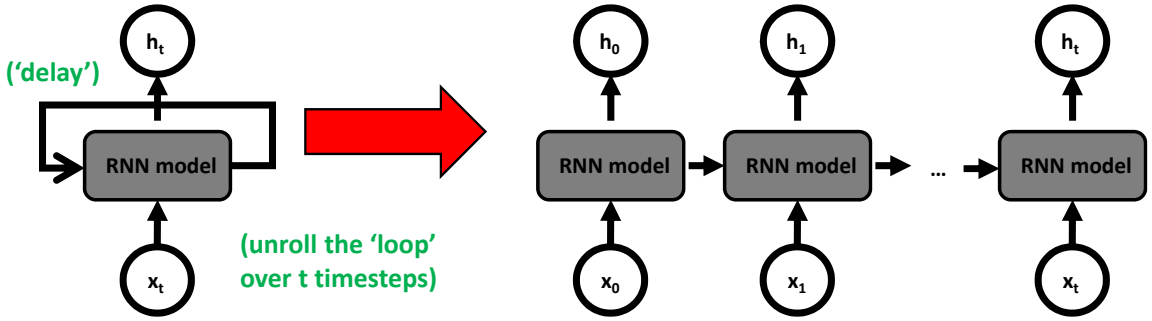
- **Facts: GPUs are mostly used today for deep learning compared to CPUs, FPGA, and specialized hardware**
- **Facts: ~55% of all users that use deep learning use it with multiple nodes instead of just a single node**
- **Facts: The communication layer MPI is mostly used as communication layer for distributed training compared to Spark, Remote Procedure Calls, MapReduce, or traditional Sockets**
- **Most users use deep learning today with minibatches that are selected numbers of samples for performing the optimization (e.g. SGD on minibatches)**
- **Minibatches should be not too small to increate performance, but also not too large to increase validation error**



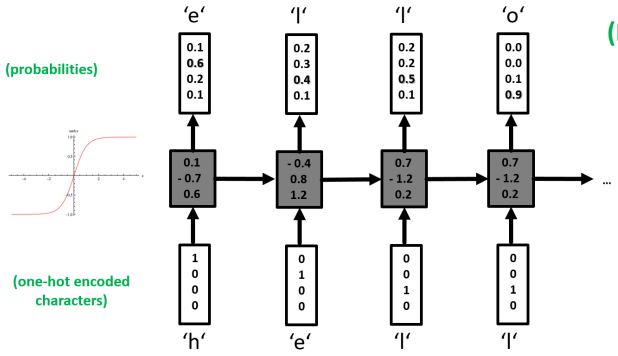
[33] T. Ben-Nun & T. Hoefler

➢ **Complementary Cloud Computing & Big Data Course offers more parallel programming models such as map-reduce & Apache Spark**

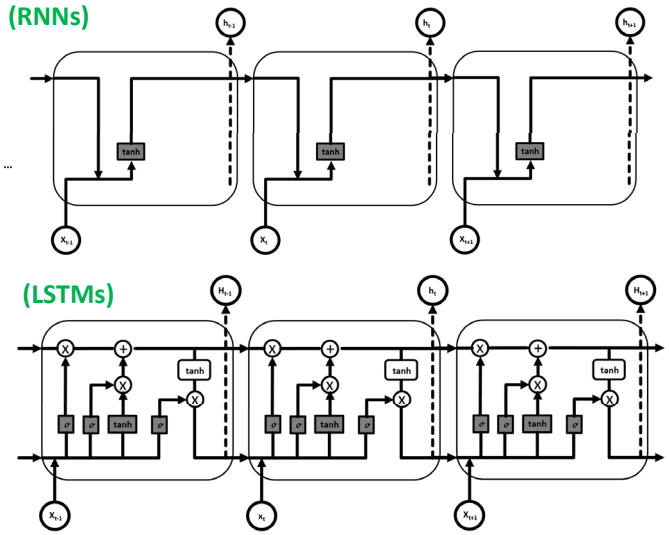
More Complex Deep Learning Model Example: Long Short-Term Memory (LSTM)



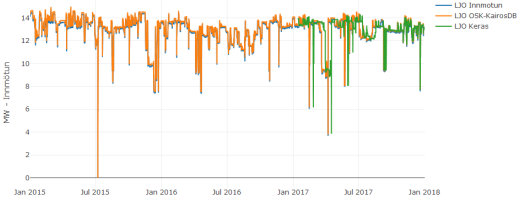
- A Recurrent Neural Network (RNN) consists of cyclic connections that enable the neural network to better model sequence data compared to a traditional feed forward artificial neural network (ANN)
- RNNs consists of 'loops' (i.e. cyclic connections) that allow for information to persist while training
- The repeating RNN model structure is very simple whereby each has only a single layer (e.g. tanh)



[32] MIT Course



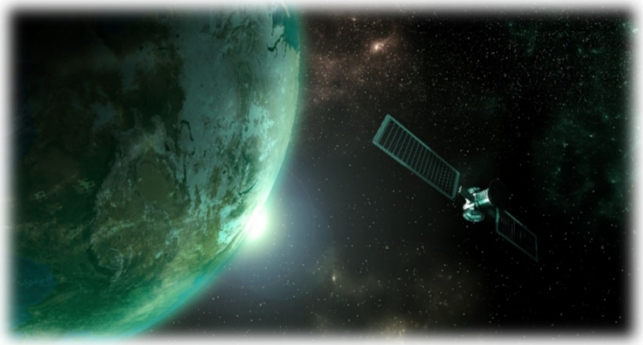
(Key challenge: find the right parameters)

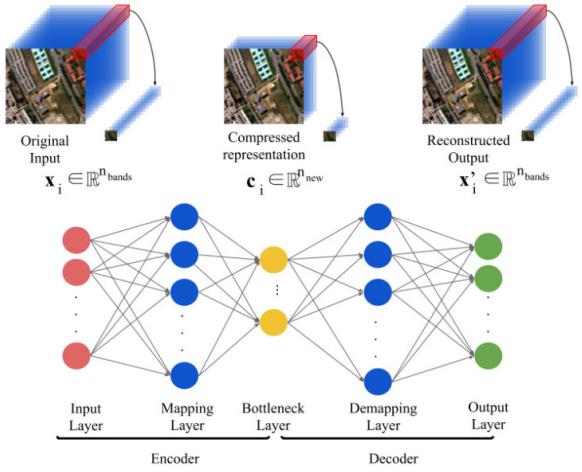


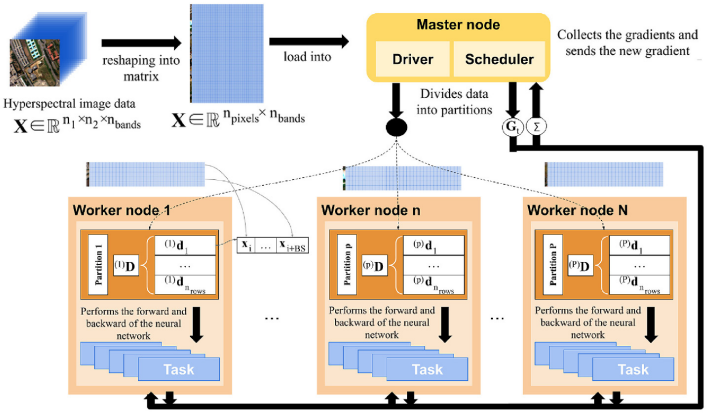
Landsvirkjun
National Power Company of Iceland

- Long Short Term Memory (LSTM) networks are a special kind of Recurrent Neural Networks (RNNs)
- LSTMs learn long-term dependencies in data by remembering information for long periods of time
- The LSTM chain structure consists of four neural network layers interacting in a specific way

More Complex Deep Learning Model Example: Autoencoder Networks



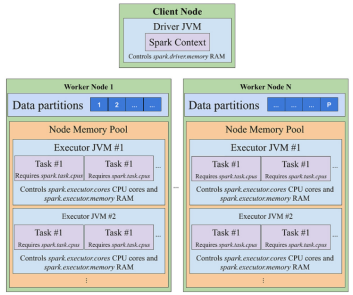




[28] J. Haut, G. Cavallaro and M. Riedel et al., IEEE Transactions on Geoscience and Remote Sensing, 2019

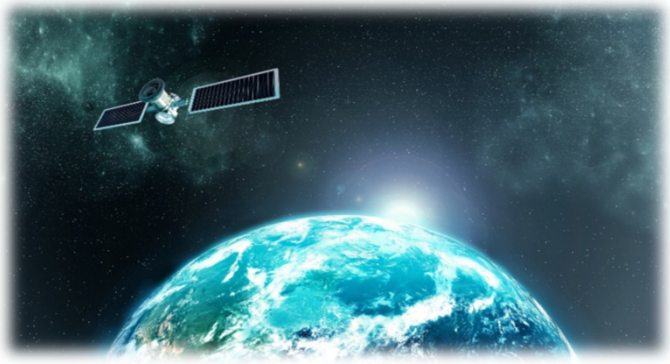
[29] Apache Spark Web page

- **Find right set of hyper-parameters and the right neural network architecture for autoencoder is a manual time-consuming and error-prone process**
- **Needs urgently HPC, but a systematic and automated way is required as trying out all options of hyper-parameters and architectures is computationally infeasible**
- **As resolutions of sensors becomes better and more data is available it is likely that the learning model will be increasingly complex in the future that in turn raises demands for automated architecture search and meta-learning approaches**



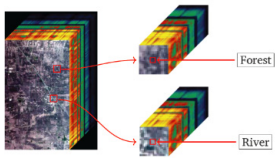
➢ **Complementary Cloud Computing & Big Data Course offers more details on using Apache Hadoop/Spark for Machine/Deep Learning**

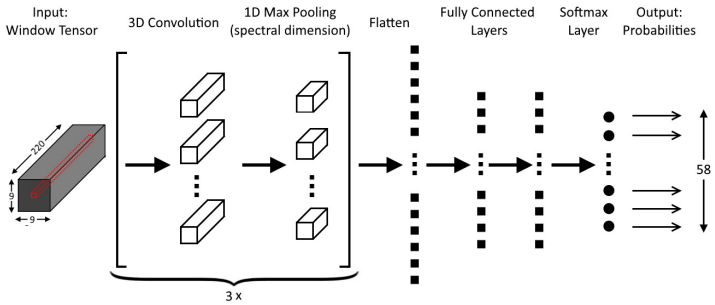
Deep Learning Application Examples – Key Challenge: Find the Right Parameters



- **Using Convolutional Neural Networks (CNNs) with hyperspectral remote sensing image data**

[26] J. Lange and M. Riedel et al., IGARSS Conference, 2018

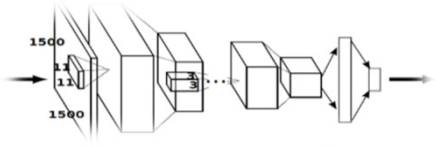




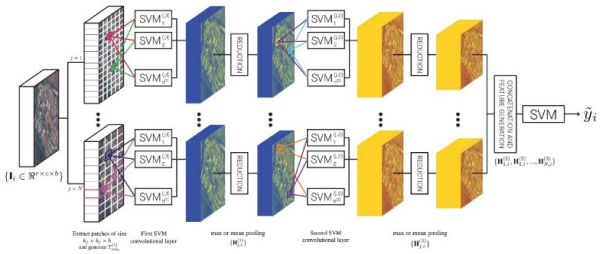
- **Find right set of hyper-parameters and the right neural network architecture is a manual time-consuming and error-prone process**
- **Needs urgently HPC, but a systematic and automated way is required as trying out all options of hyper-parameters and architectures is computationally infeasible**

- What is the right optimization method?
- How many convolutional layers we need?
- How many neurons in dense layers?
- What is the right filter size?
- How do we train best?

Feature	Representation / Value
Conv. Layer Filters	48, 32, 32
Conv. Layer Filter size	(3, 3, 5), (3, 3, 5), (3, 3, 5)
Dense Layer Neurons	128, 128
Optimizer	SGD
Loss Function	mean squared error
Activation Functions	ReLU
Training Epochs	600
Batch Size	50
Learning Rate	1
Learning Rate Decay	5×10^{-6}



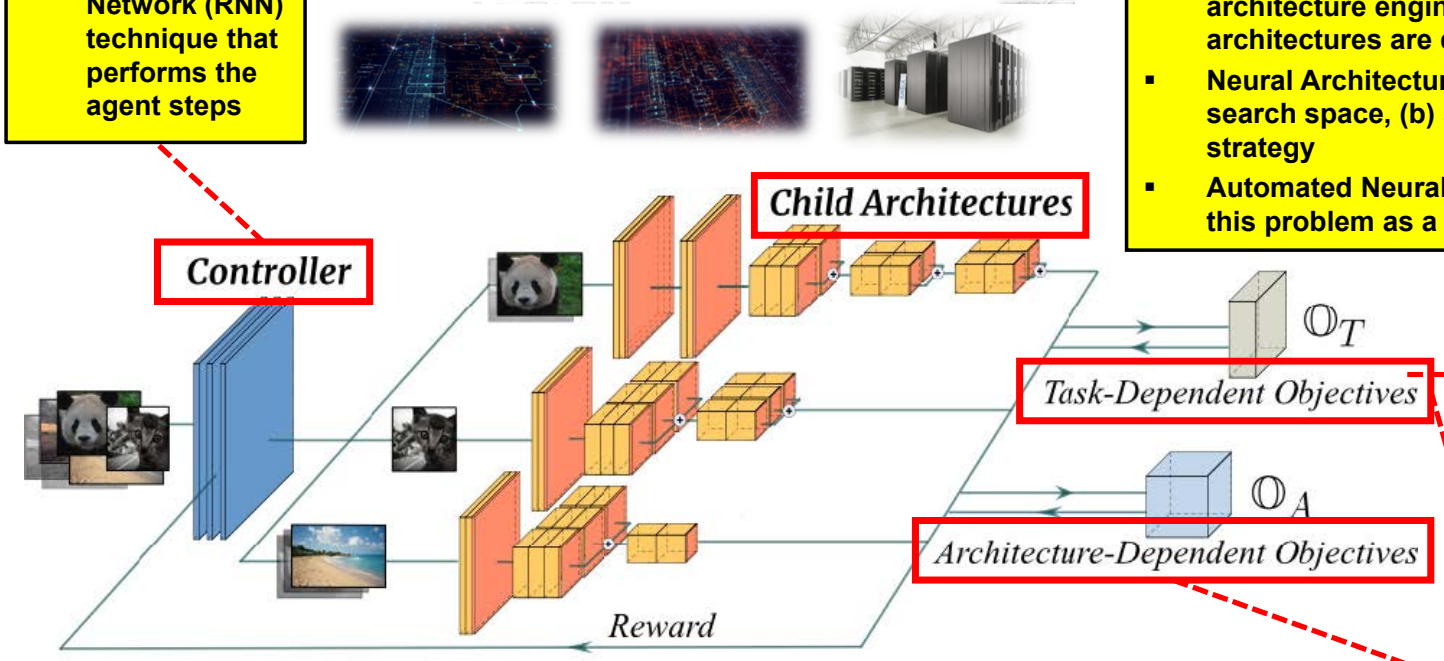
- **Find Hyperparameters & joint 'new-old' modeling & transfer learning given rare labeled/annotated data in science (e.g. 36,000 vs. 14,197,122 images ImageNet)**



[27] G. Cavallaro, M. Riedel et al., IGARSS 2019

Massive Requirement for HPC Resources: Neural Architecture Search (NAS)

- Often a Recurrent Neural Network (RNN) technique that performs the agent steps



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

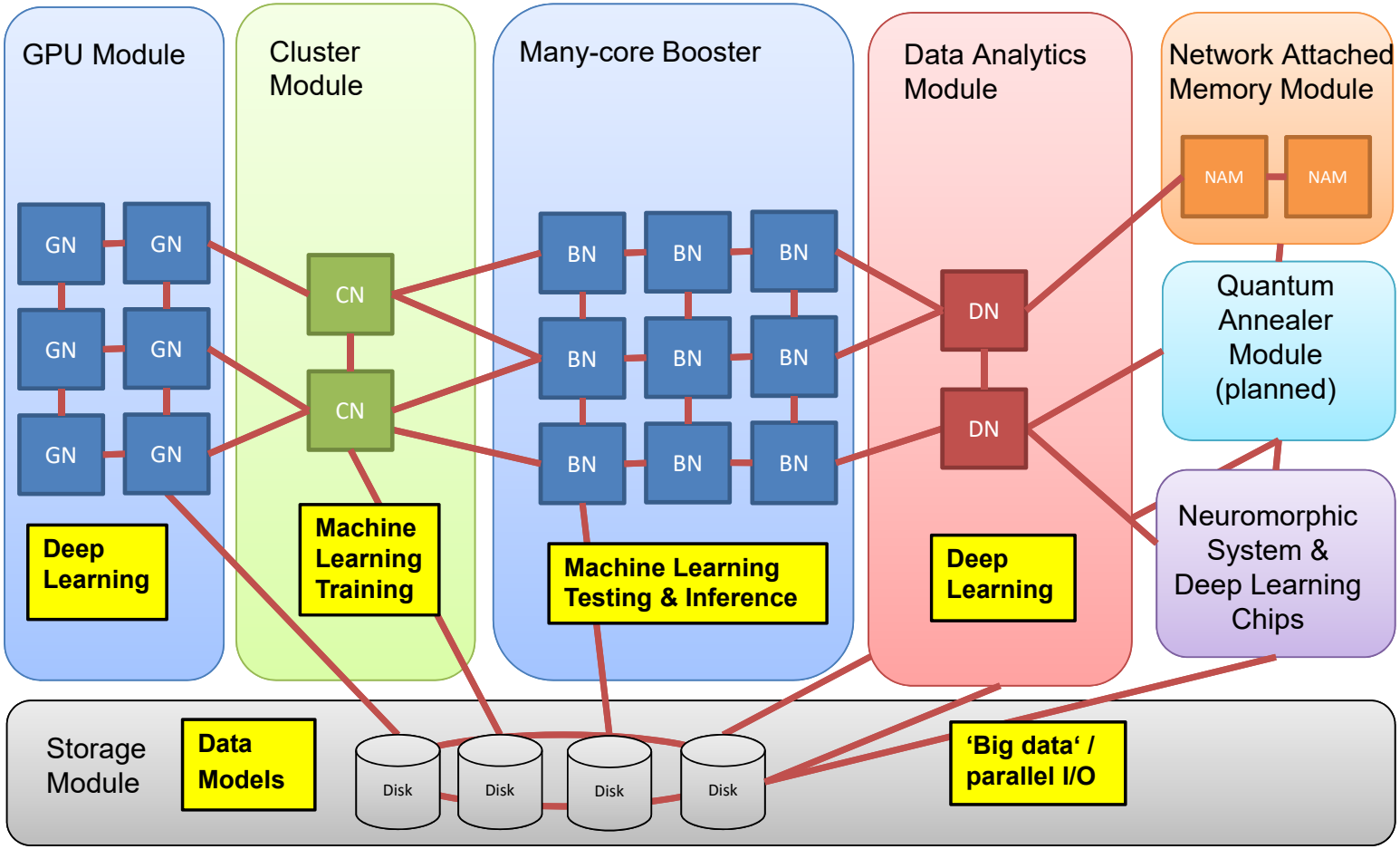
- Employed neural networks architectures are often developed manually by human experts that is time-consuming and error-prone
- Deep learning success has been accompanied by a rising demand for architecture engineering, where increasingly more complex neural architectures are designed manually
- Neural Architecture Search (NAS) methods can be categorized in (a) search space, (b) search strategy, and (c) performance estimation strategy
- Automated Neural Architecture (NAS) search methods aim to solve this problem as a process of automating Architecture engineering

[31] M. Riedel, 'NAS with Reinforcement Learning'

- Derived specific architectures that perform good for specific dataset samples
- E.g. what is the accuracy or error rate we obtain as metric to guide the search for specific architectures for specific dataset samples
- E.g. what is the latency of the network for a given dataset sample to guide the search for specific architectures that offer better latency by keeping accuracy(!)

Modular Supercomputing – Mapping of Machine/Deep Learning Processes

[24] DEEP Projects Web Page

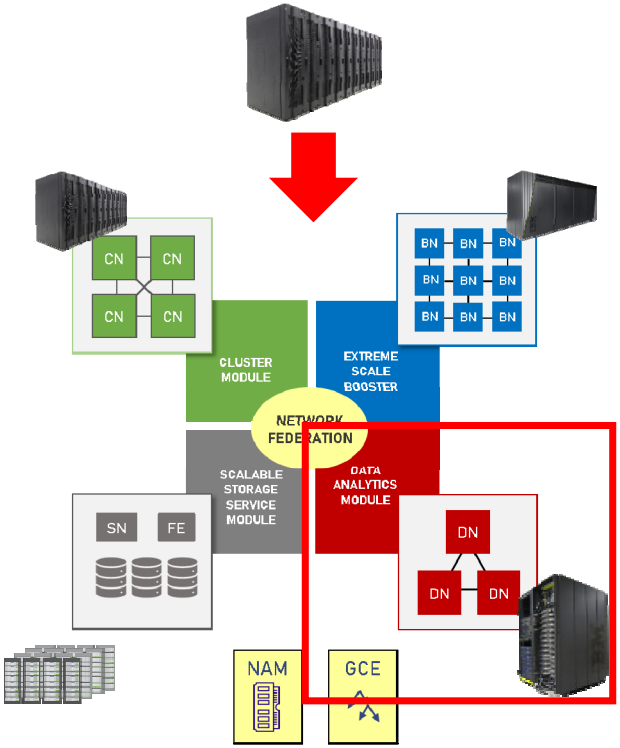


Innovative Ideas, e.g. trained models in memory, put/get store for data, non-volatile memory, etc.

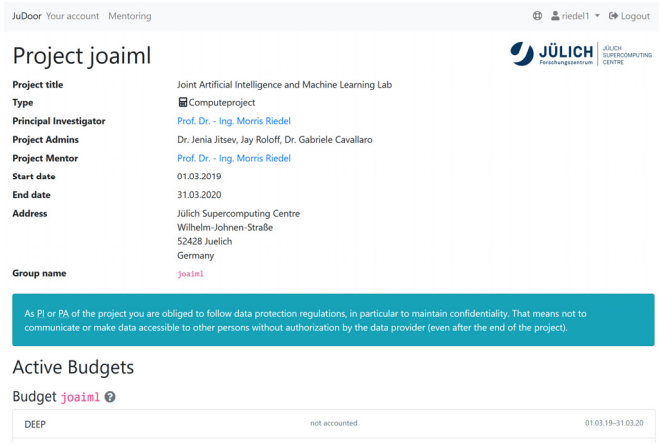
Innovative computing paradigms for specific tasks, e.g. solving optimization tasks in machine learning algorithms

Innovative chips, e.g. use of deep learning optimized chip designs

Modular Supercomputing Architecture – Data Analytics Module (DAM)



- Data Analytics Module (DAM)
  - Specific requirements for data science & analytics frameworks
  - 16 nodes with 2x Intel Xeon Cascade Lake; 24 cores
  - 1x NVIDIA V100 GPU / node
  - 1x Intel STRATIX10 FPGA PCIe3 / node
  - 384 GB DDR4 memory / node
  - 2 TB non-volatile memore / node
- DAM Prototype for teaching
  - 3 x 4 GPUs Tesla Volta V100
  - Slurm scheduling system



Every group need to register in JUSER Project JOAIML

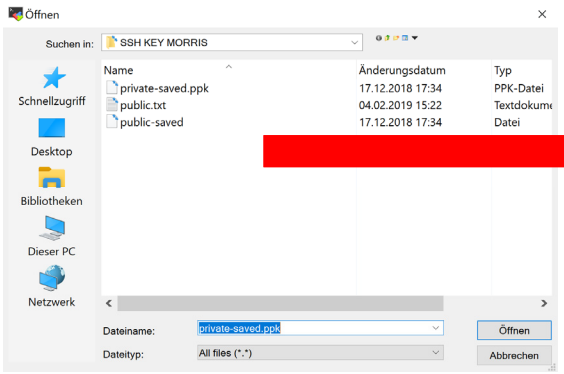
DEEP Projects

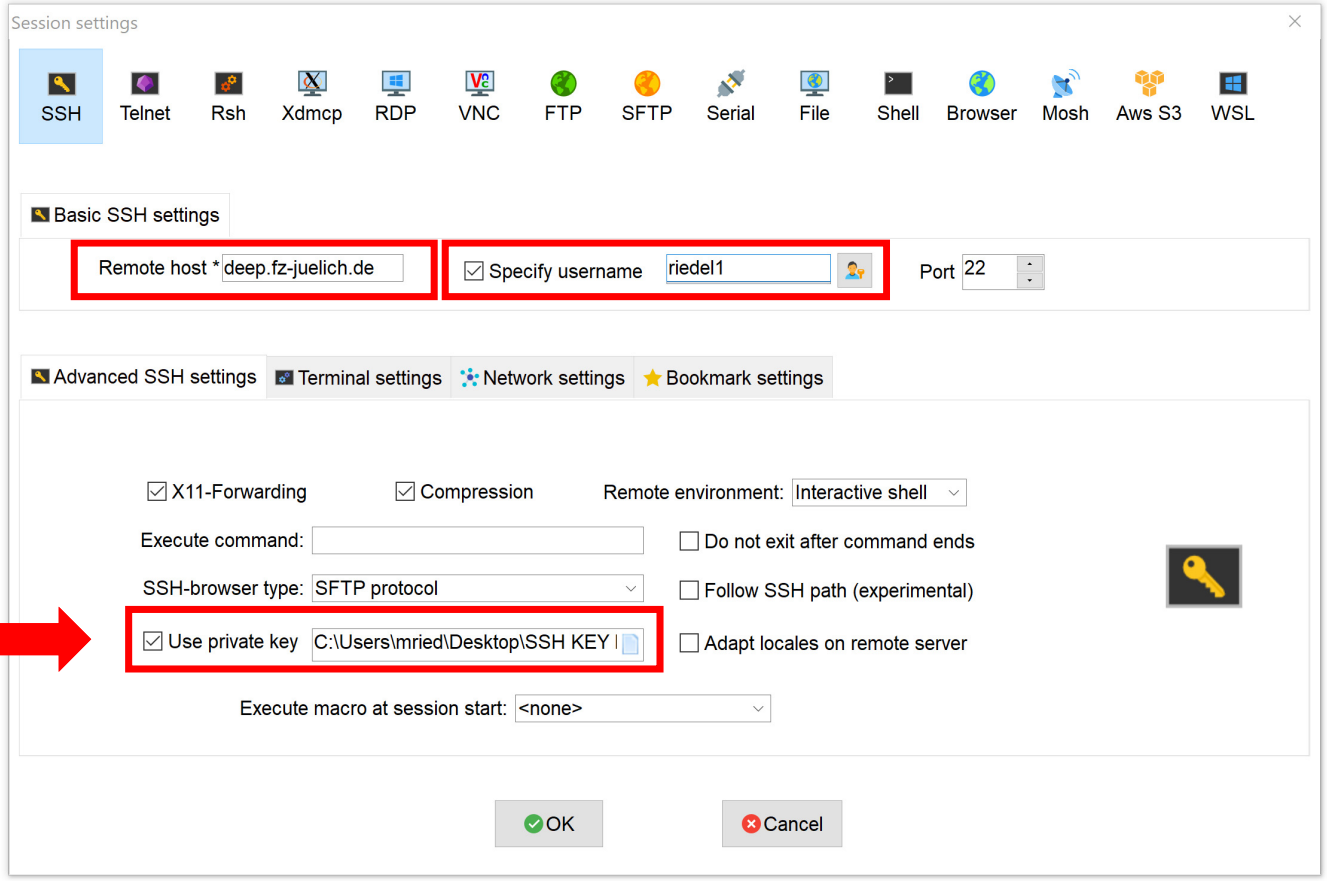
[24] DEEP Projects Web Page

➢ **The DAM prototype machine as part of the modular supercomputing architecture will be used in Assignment #2 for deep learning**

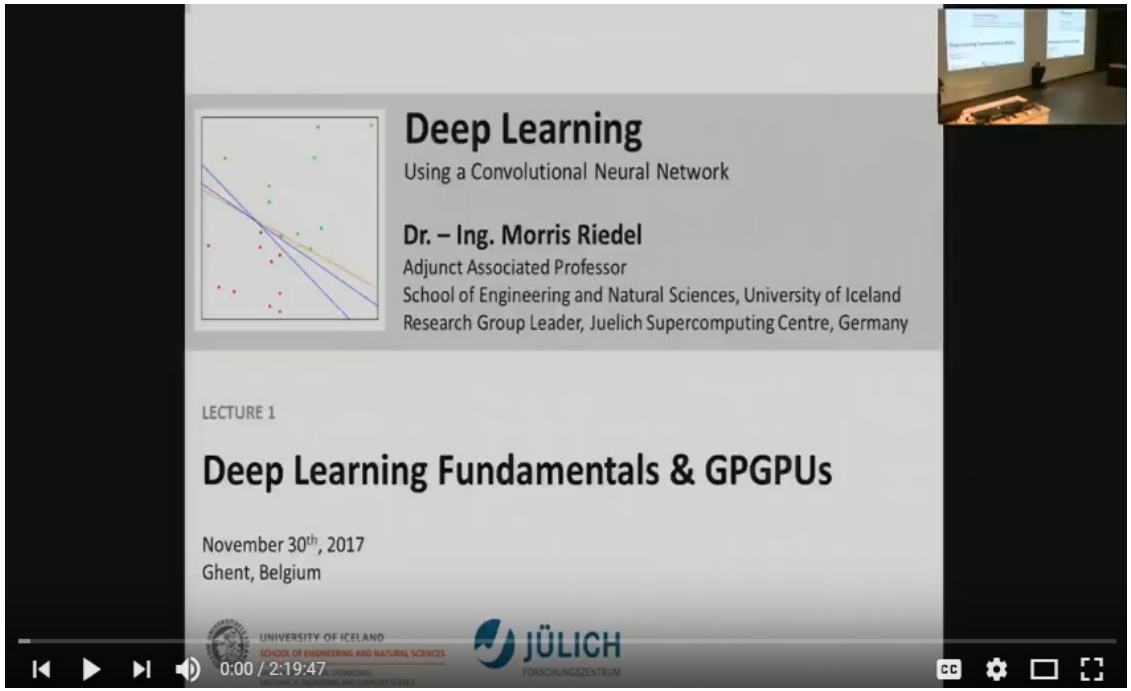
SSH Keys – Use Private/Public Key Pair to Access DEEP HPC System

- Remember to use your Private SSH Key to connect to the DEEP system
  - Corresponding Public SSH key is already uploaded on the HPC System (remote host) per username(!)
  - (cf. Practical Lecture 0.1)



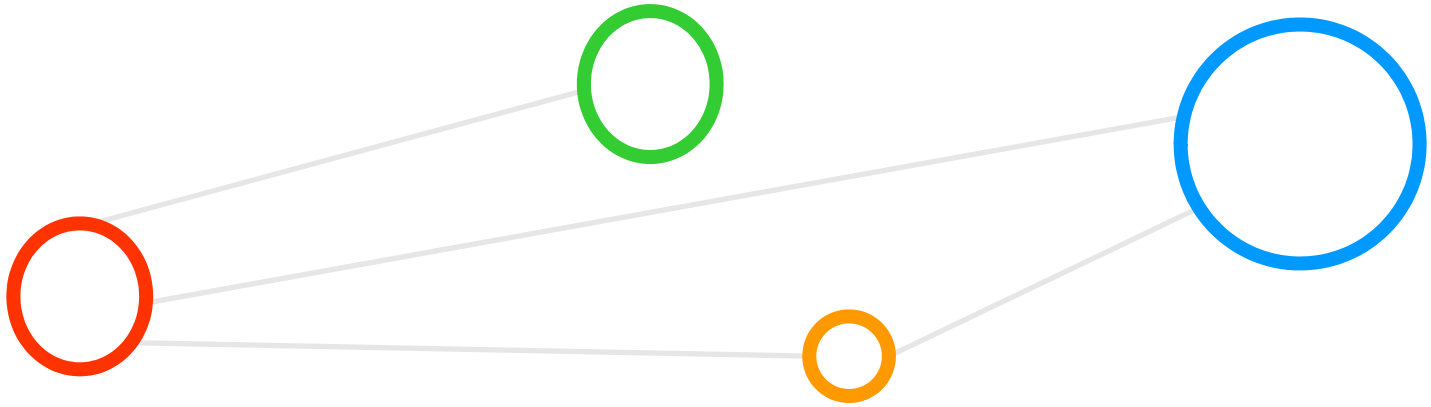


[YouTube Lectures] More Details about Deep Learning Basics



[20] Morris Riedel, 'Deep Learning - Using a Convolutional Neural Network', Invited YouTube Lecture, six lectures, University of Ghent, 2017

Lecture Bibliography



Lecture Bibliography (1)

- [1] Morris Riedel, 'Introduction to Machine Learning Algorithms', Invited YouTube Lecture, six lectures University of Ghent, 2017, Online: https://www.youtube.com/watch?v=KgiuUZ3WeP8&list=PLrmNhuZo9sgbcWtMGN0i6G9HEvh08JG0J
- [2] Jupyter Web Page, Online: https://jupyter.org/
- [3] Tensorflow, Online: https://www.tensorflow.org/
- [4] Keras Python Deep Learning Library, Online: https://keras.io/
- [5] Big Data Tips, 'Gradient Descent', Online: http://www.big-data.tips/gradient-descent
- [6] www.big-data.tips, 'Relu Neural Network', Online: http://www.big-data.tips/relu-neural-network
- [7] www.big-data.tips, 'tanh', Online: http://www.big-data.tips/tanh
- [8] CPU/GPU Performance Comparison, Online: https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardwarecharacteristics-over-time/
- [9] Summit Supercomputer Architecture Overview, Online: https://www.olcf.ornl.gov/wp-content/uploads/2019/05/Summit_System_Overview_20190520.pdf
- [10] A. Herten et al., Introduction to GPU Programming JSC Course
- [11] D. Kingma and Jimmy Ba, 'Adam: A Method for Stochastic Optimization', Online: https://arxiv.org/abs/1412.6980
- [12] SDSC, Nvidia Training – Introduction, Online: http://www.sdsc.edu/us/training/assets/docs/NVIDIA-01-Intro.pdf

Lecture Bibliography (2)

- [13] M. Goetz, C. Bodenstein, M. Riedel, 'HPDBSCAN – Highly Parallel DBSCAN', in proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC2015), Machine Learning in HPC Environments (MLHPC) Workshop, 2015, Online: https://www.researchgate.net/publication/301463871_HPDBSCAN_highly_parallel_DBSCAN
- [14] Species Iris Group of North America Database, Online: http://www.signa.org
- [15] G. Cavallaro, M. Riedel, M. Richerzhagen, J. A. Benediktsson and A. Plaza, "On Understanding Big Data Impacts in Remotely Sensed Image Classification Using Support Vector Machine Methods," *in the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4634-4646, Oct. 2015, Online: https://www.researchgate.net/publication/282524415_On_Understanding_Big_Data_Impacts_in_Remotely_Sensed_Image_Classification_Using_Support_Vector_Machine_Methods
- [16] C. Cortes & V. Vapnik (1995). Support-vector networks. Machine learning, 20(3), 273-297, Online: https://doi.org/10.1007/BF00994018
- [17] Original piSVM tool, Online: http://pisvm.sourceforge.net/
- [18] Harley, A.W., An Interactive Node-Link Visualization of Convolutional Neural Networks, Online: http://scs.ryerson.ca/~aharley/vis/conv/flat.html
- [19] A. Gulli and S. Pal, 'Deep Learning with Keras' Book, ISBN-13 9781787128422, 318 pages, Online: https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras
- [20] Morris Riedel, 'Deep Learning - Using a Convolutional Neural Network', Invited YouTube Lecture, six lectures, University of Ghent, 2017, Online: https://www.youtube.com/watch?v=gOL1_YIosYk&list=PLrmNhuZo9sgZUdaZ-f6OHK2yFW1kTS2qF
- [21] Kaiming He et al., 'Deep Residual Learning for Image Recognition', Online: https://arxiv.org/pdf/1512.03385.pdf
- [22] J. Dean et al., 'Large scale deep learning', Keynote GPU Technical Conference, 2015

Lecture Bibliography (3)

- [23] ImageNet Web page, Online: http://image-net.org
- [24] DEEP Projects Web page, Online: http://www.deep-projects.eu/
- [25] Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow, Online: https://www.slideshare.net/databricks/horovod-ubers-open-source-distributed-deep-learning-framework-for-tensorflow
- [26] J. Lange, G. Cavallaro, M. Goetz, E. Erlingsson, M. Riedel, 'The Influence of Sampling Methods on Pixel-Wise Hyperspectral Image Classification with 3D Convolutional Neural Networks', Proceedings of the IGARSS 2018 Conference
- [27] Cavallaro, G., Bazi, Y., Melgani, F., Riedel, M.: Multi-Scale Convolutional SVM Networks for Multi-Class Classification Problems of Remote Sensing Images, in conference proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2019), July 28 – August 2nd, 2019, Yokohama, Japan
- [28] Haut, J.M., Gallardo, J.A., Paoletti, M.E., Cavallaro, G., Plaza, J., Plaza, A., Riedel, M.: Cloud Deep Networks for Hyperspectral Image Analysis, IEEE Transactions on Geoscience and Remote Sensing, PP(99):1-17, 2019, Online: https://www.researchgate.net/publication/335181248_Cloud_Deep_Networks_for_Hyperspectral_Image_Analysis
- [29] Apache Spark, Online: https://spark.apache.org/
- [30] Cheng, A.C, Lin, C.H., Juan, D.C., InstaNAS: Instance-aware Neural Architecture Search, Online: https://arxiv.org/abs/1811.10201
- [31] M. Riedel, 'Neural Architecture Search with Reinforcement Learning', Online: http://www.morrisriedel.de/neural-architecture-search-with-reinforcement-learning
- [32] S.191 MIT Intro to Deep Learning, 'Sequence Modeling with Neural Networks' Online: https://www.youtube.com/watch?v=CznICCPa63Q&t=29s
- [33] T. Ben-Nun & T. Hoefler, 'Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis', Online: http://doi.acm.org/10.1145/3320060


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