Cloud Computing & Big Data

PARALLEL & SCALABLE MACHINE LEARNING & DEEP LEARNING

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

LECTURE 6

Deep Learning driven by Big Data

October 11th, 2018
Room Stapi 108

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

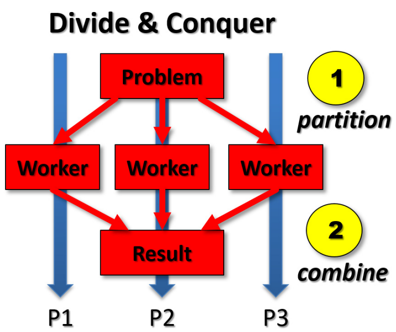
JÜLICH Forschungszentrum

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

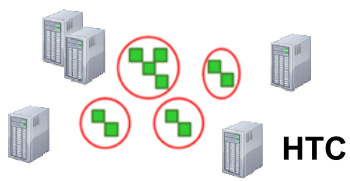
DEEP Projects

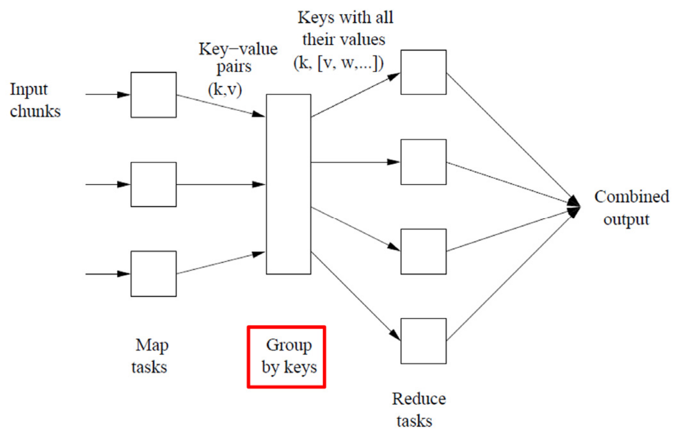
Review of Lecture 5 – Map Reduce Computing Paradigm

- Map Reduce



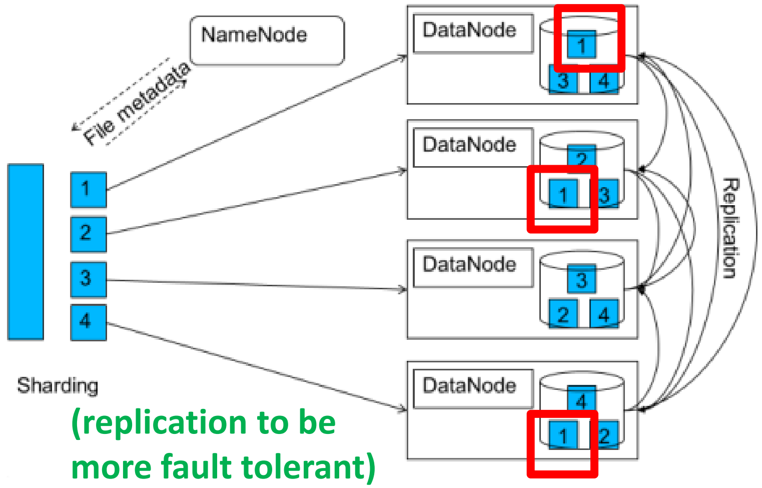
(origin in traditional computer & computational sciences)





(group/shuffle/sort done by the framework)

- Hadoop Distributed File System



(replication to be more fault tolerant)

- Selected Cloud Applications

Modified from [1] Mining of Massive Datasets *[2] Apache Hadoop* *[3] AWS Marketplace*

Outline of the Course

1. Cloud Computing & Big Data
2. Machine Learning Models in Clouds
3. Apache Spark for Cloud Applications
4. Virtualization & Data Center Design
5. Map-Reduce Computing Paradigm
6. Deep Learning driven by Big Data
7. Deep Learning Applications in Clouds
8. Infrastructure-As-A-Service (IAAS)
9. Platform-As-A-Service (PAAS)
10. Software-As-A-Service (SAAS)
11. Data Analytics & Cloud Data Mining
12. Docker & Container Management
13. OpenStack Cloud Operating System
14. Online Social Networking & Graphs
15. Data Streaming Tools & Applications
16. Epilogue

+ additional practical lectures for our hands-on exercises in context

- Practical Topics
- Theoretical / Conceptual Topics

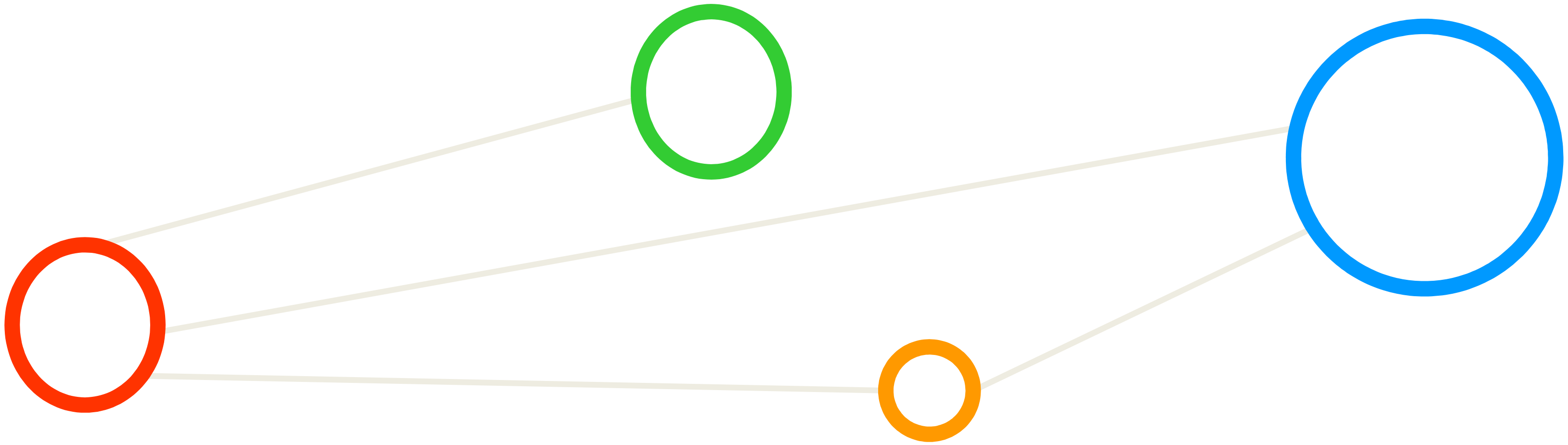
Outline

- Deep Learning Fundamentals
  - Supervised Learning Applications
  - Logistic Regression Algorithm
  - Stochastic Gradient Descent (SGD)
  - Derivatives & Backpropagation Algorithm
  - Artificial Neural Networks (ANNs)

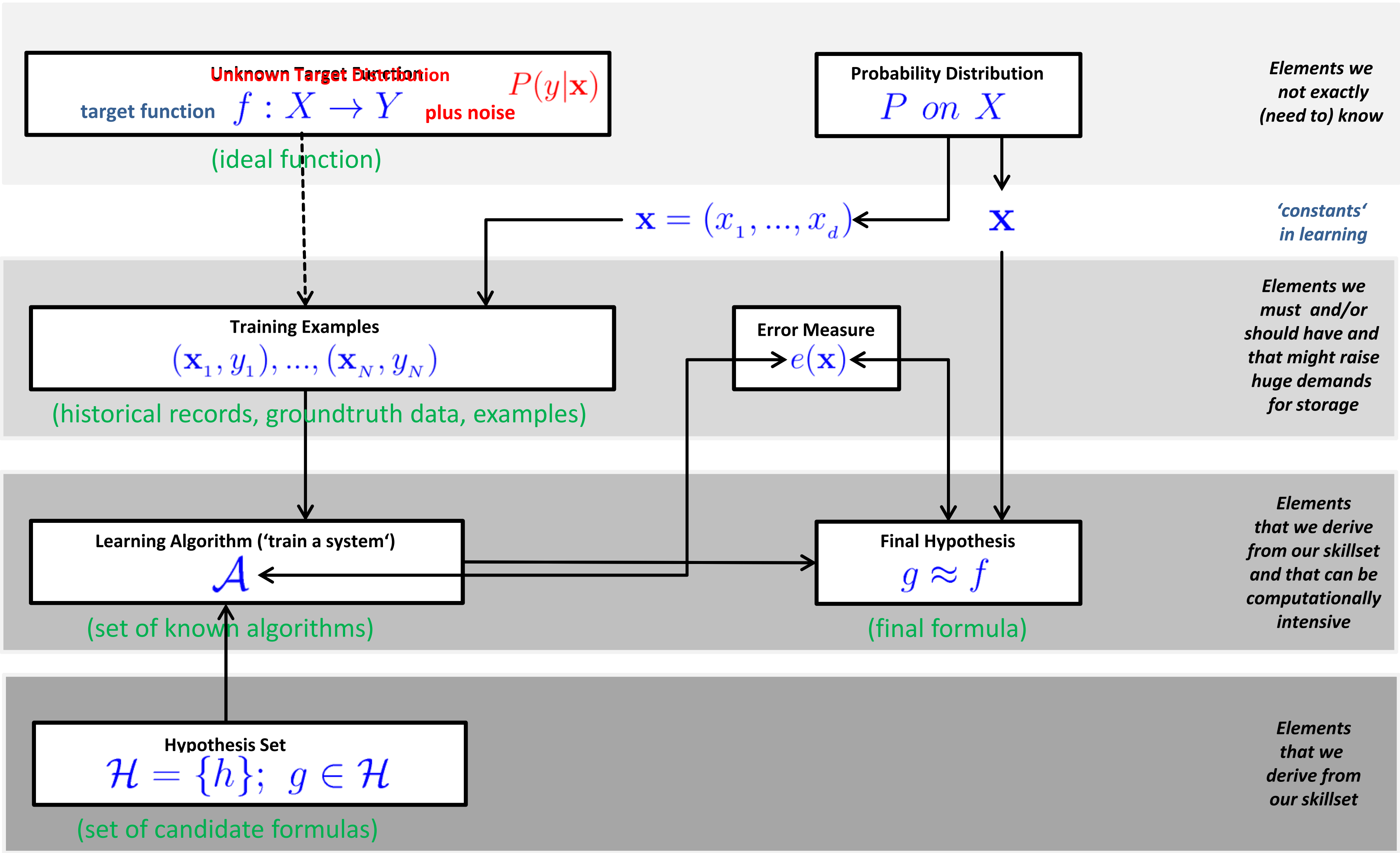
- Deep Learning Models & Big Data
  - Conceptual Idea of Deep Learning
  - Relationship Big Data & Deep Learning
  - Convolutional Neural Networks (CNNs)
  - Cloud Support & GPGPU Relevance
  - Other Deep Learning Models

- **Promises from previous lecture(s):**
- ***Lecture 0 & Practical Lecture 0.1 – Prologue:*** **Lecture 6 & 7 provide a short introduction to deep learning models and applications in clouds**
- ***Practical Lecture 0.1:*** **Lecture 6 & 7 provide a a more thorough introduction to tensors of various dimensions**
- ***Lecture 1:*** **Lecture 6 & 7 provide more details on how GPUs are used as key technology in deep learning**
- ***Lecture 2 & Practical Lecture 3.1:*** **Lecture 6 & 7 offer more details on feature selection concepts including spatial concepts in images**
- ***Lecture 2 & Practical Lecture 3.1:*** **Lecture 6 & 7 & 11 provide more data visualization examples in context of different applications**
- ***Lecture 2:*** **Lecture 3 & 6 provides more insights into Logistic Regression & Gradient Descent Optimization**

Deep Learning Fundamentals

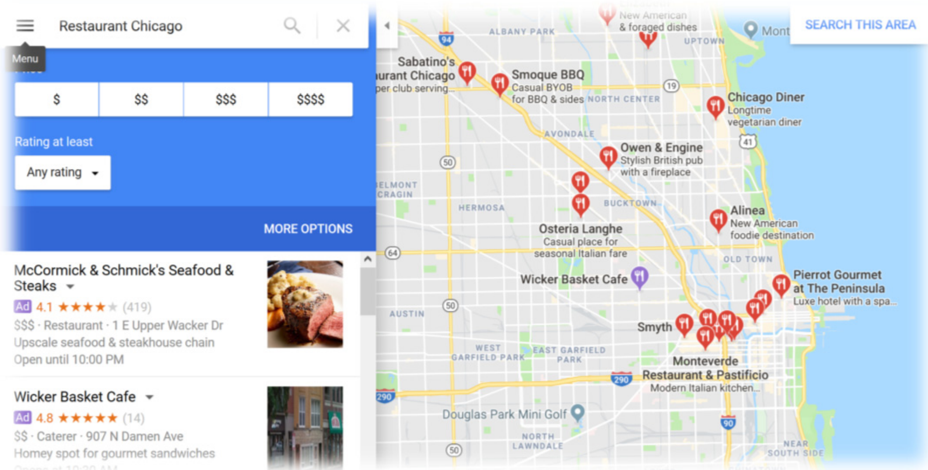


Supervised Learning Revisited – Overview & Summary



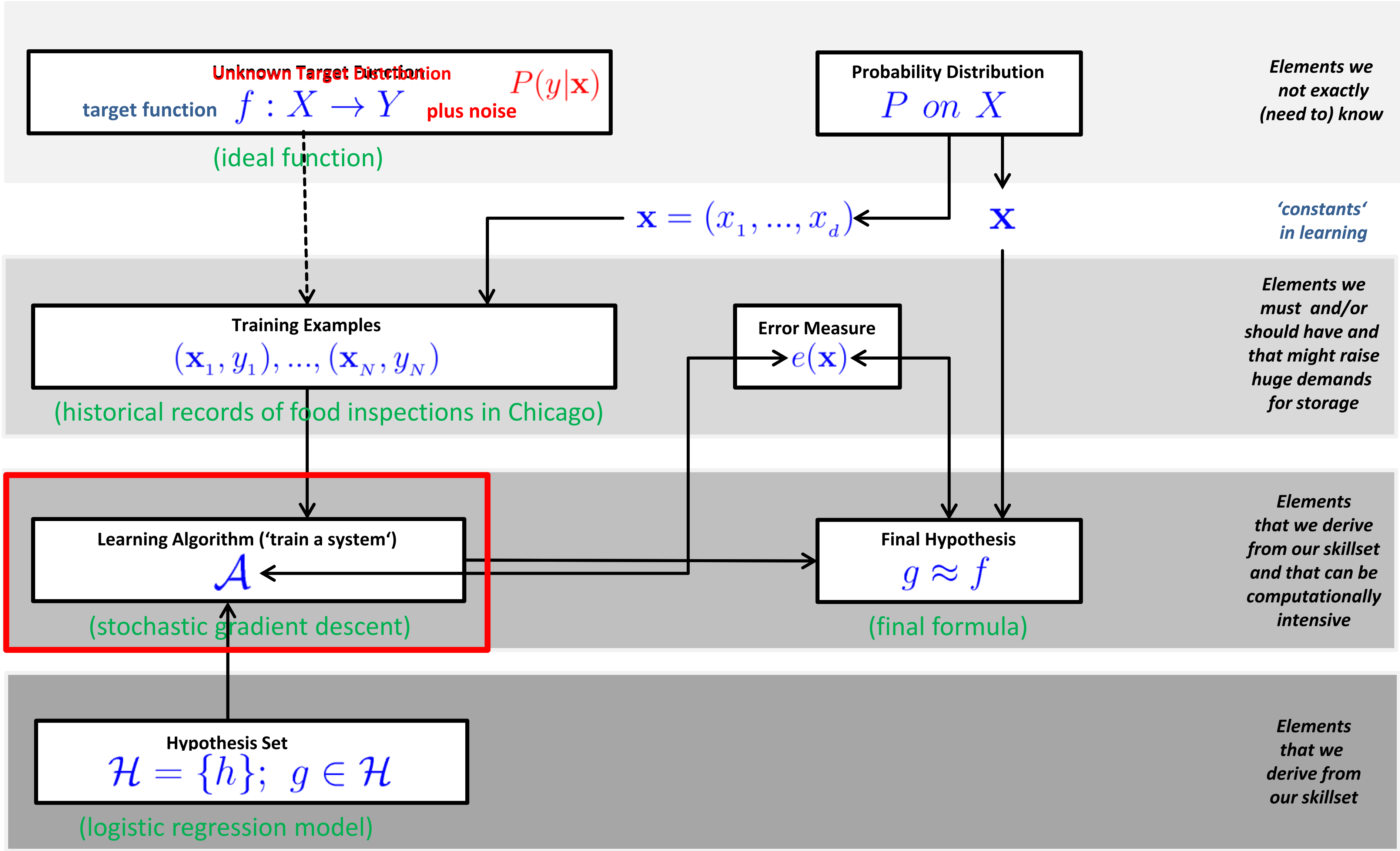
Food Inspection in Chicago Application – Revisited

1. Some pattern exists:
   - Believe in a pattern with 'quality violations in checking restaurants' will somehow influence if food inspection pass or fail (binary classification)
2. No exact mathematical formula
   - To the best of our knowledge there is no precise formula for this problem
3. Data exists
   - Data collection from City of Chicago



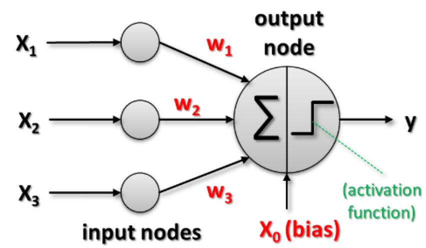
- **The goal of the advanced machine learning application with food inspection of restaurants in the City of Chicago is to predict the outcome of food inspection of new Chicago restaurants given some of existing violations of other restaurants already obtained in Chicago**

Supervised Learning – Food Inspection Chicago Application

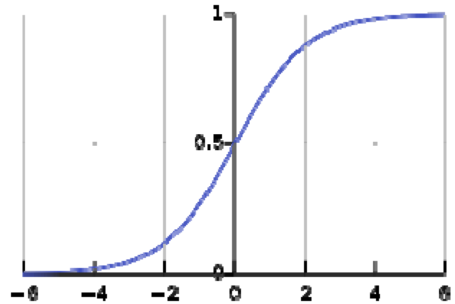


Logistic Regression Using Non-Linear Activation Function

- Linear Classification
  - Simple binary classification (linearly seperable)
  - Linear combination of the inputs x_i with weights w_i



- Linear Regression
  - Real value with the activiation being the identity function
  - E.g. how much sales given marketing money spend on TV advertising
- Logistic Regression
  - Different from above: model/error measure/learning algorithm is different
  - Captures non-linear data dependencies using the so-called Sigmoid function
  - Key idea is to bring values between 0 and 1 to estimate a probability



- **One of the optimization algorithms that can be used to train a logistic regression model is Stochastic Gradient Descent (SGD)**

Big Data: Python/NumPy & Vectorization Not Enough

- 'Small-scale example' of the power of 'parallelization'
  - Enables element-wise computations at the same time (aka in parallel)
  - 'small-scale' since we are still within one computer – but perform operations in parallel on different data

$h(\mathbf{x}) = s\,(\mathbf{w}^T\mathbf{x})$ **(logistic regression)**

(vector notation, using T = transpose)

$$\mathbf{w}_i = (w_{i1}, w_{i2}, \ldots, w_d)$$

$$\mathbf{w}_i^T = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \cdots \\ w_{id} \end{bmatrix}$$

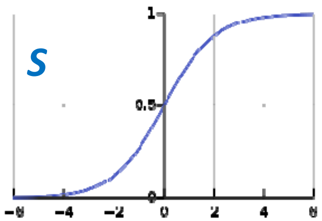
$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_d)$$

$$h(\mathbf{x}) = s\,(\mathbf{w} \cdot \mathbf{x})$$

(equivalent dotproduct notation)

```
In [ ]: import numpy as np
        ...
        print(np.dot(w.T,x))
```

(np.dot() is a vectorized function that is fast, but still not fast enough when facing big data sets)

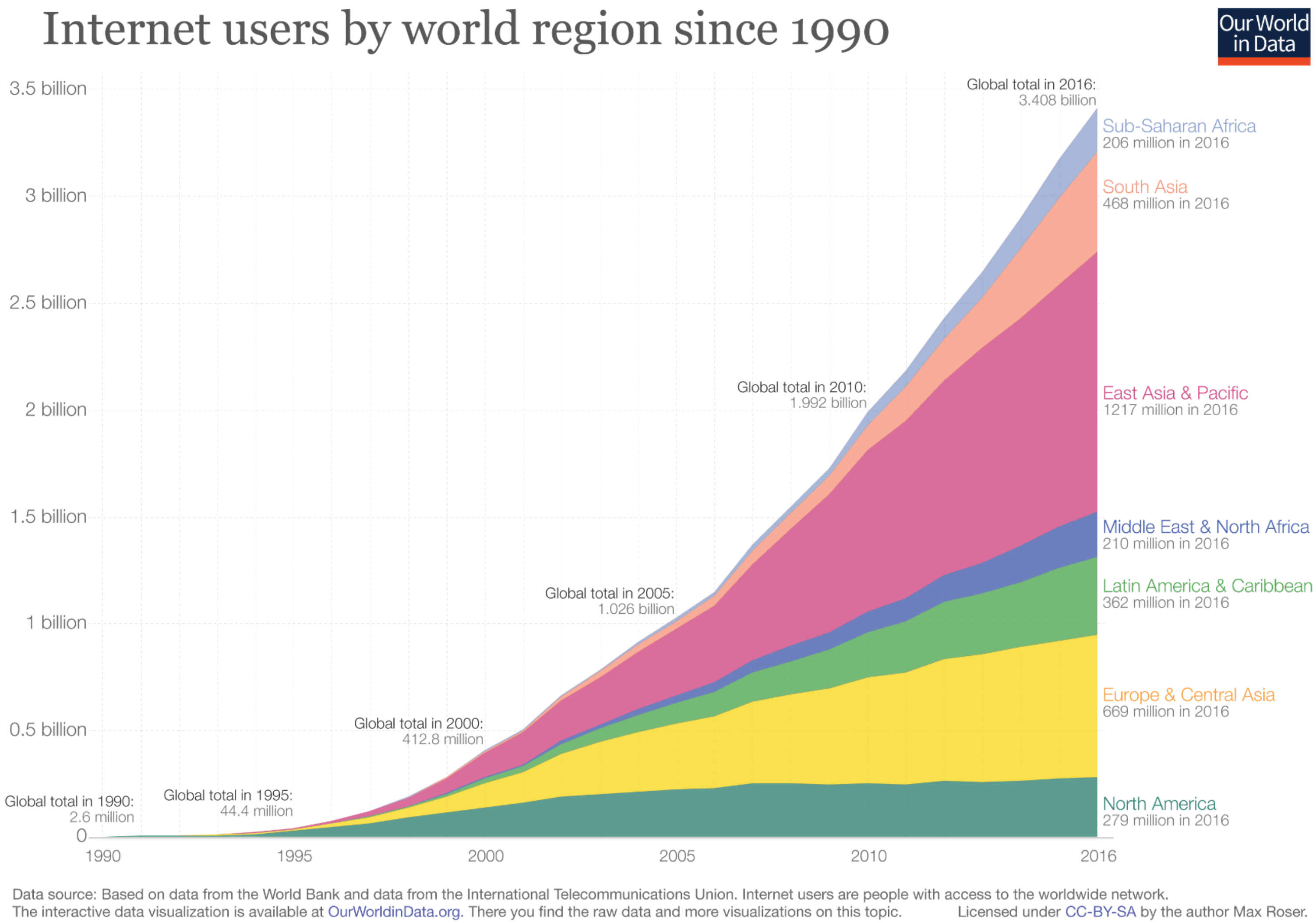


(sigmoid function to bring all values towards probability between 0 and 1)

- **Challenges for Big Data in real life scenarios require a large-scale & elastic Cloud infrastructure**
- **Vectorization matter in small-scale per CPU/node, but large-scale parallelization is also important**

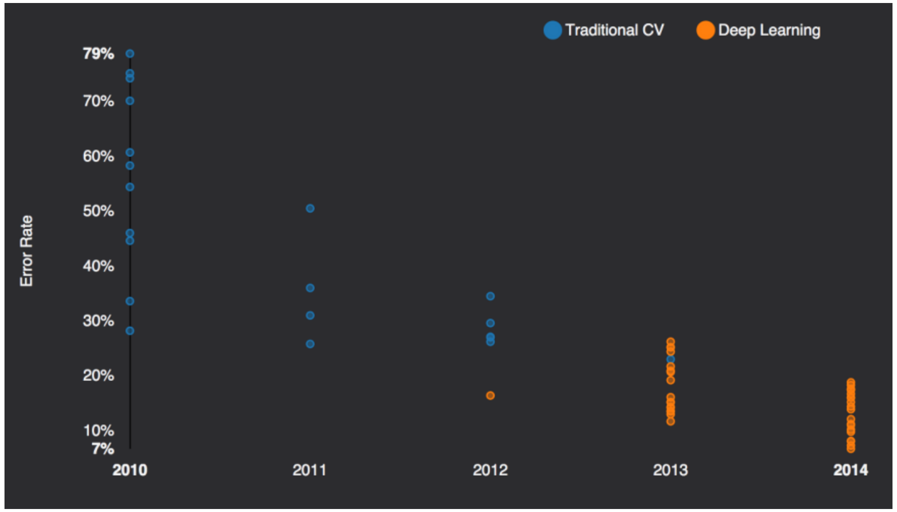
➢ **Lecture 7 provides more details on SGD in context of Logistic Regression & Neural Networks**

Big Data Challenges: Growth of Users → Growth of Data



Motivation Deep Learning – ImageNet 'Big Data' Example

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



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

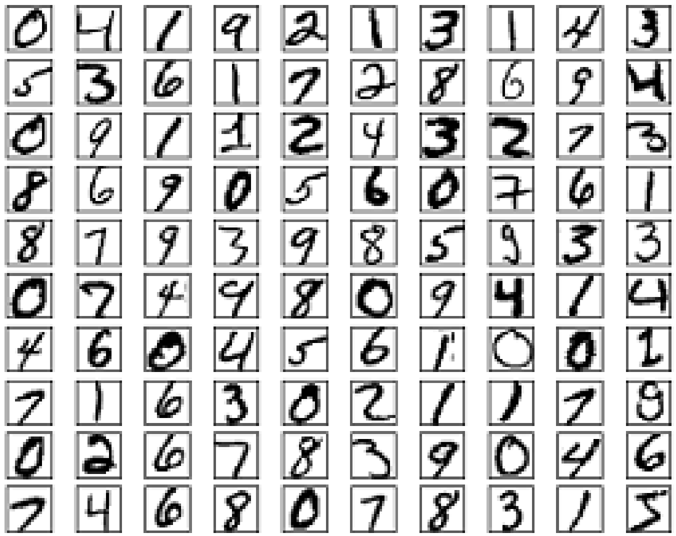
High level category	# synset (subcategories)	Avg # images per synset	Total # images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	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

[22] ImageNet Web page

Handwritten Character Recognition MNIST Application

- Metadata
  - Subset of a larger dataset from US National Institute of Standards (NIST)
  - Handwritten digits including corresponding labels with values 0 to 9
  - All digits have been size-normalized to 28 * 28 pixels
    and are centered in a fixed-size image for direct processing
  - Not very challenging dataset, but good for experiments / tutorials

- MNIST Dataset Samples
  - Labelled data (10 classes)
  - Two separate files
    for training and test
  - 60000 training samples (~47 MB)
  - 10000 test samples (~7.8 MB)



MNIST Dataset – Data Exploration (cf. Practical Lecture 0.1)

- When working with the dataset
  - Dataset is not in any standard image format like jpg, bmp, or gif
  - One needs to write typically a small program to read and work for them
  - Data samples are stored in a simple file format that is designed for storing vectors and multidimensional matrices (here numpy binary files)
  - 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.

```
/homea/hpclab/train001/data/mnist
[train001@jrl09 mnist]$ pwd
/homea/hpclab/train001/data/mnist
[train001@jrl09 mnist]$ ls -al
total 53728
drwxr-xr-x  2 train001 hpclab      512 Jun  6 12:17 .
drwxr-xr-x 10 train001 hpclab      512 Jun  6 12:17 ..
-rw-r-----  1 train001 hpclab  7840080 Jun  6 12:17 x_test.npy
-rw-r-----  1 train001 hpclab 47040080 Jun  6 12:17 x_train.npy
-rw-r-----  1 train001 hpclab    10080 Jun  6 12:17 y_test.npy
-rw-r-----  1 train001 hpclab    60080 Jun  6 12:17 y_train.npy
```

MNIST Dataset – Python Script Training Data Exploration

```
import numpy as np

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")
```

```
print("Samples of 28 x 28 pixel matrices reserved for training")

# function for showing a character
def character_show(character):
    for y in character:
         row = ""
         for x in y:
             row += '{0: <4}'.format(x)
         print row
```

```
# view first 10 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)**

- **Loop of the training dataset and the testing 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 – Data Visualization & Exploration

```
[train001@jrl09 mnist]$ python explore-mnist-training.py
Samples of 28 x 28 pixel matrices reserved for training
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   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 – Exploration Script Testing

```
import numpy as np

# n x 28 x 28 pixel testing data
X_test = np.load("/homea/hpclab/train001/data/mnist/x_test.npy")

# n x 1 testing labels
Y_test = np.load("/homea/hpclab/train001/data/mnist/y_test.npy")

print("Samples of 28 x 28 pixel matrices reserved for testing")

# function for showing a character
def character_show(character):
    for y in character:
        row = ""
        for x in y:
            row += '{0: <4}'.format(x)
        print row

# view first 10 characters
for i in range (0,9):
  character_show(X_test[i])
  print("\n")
  print("Label:")
  print(Y_test[i])
  print("\n")
```

MNIST Dataset – Exploration – Selected Testing 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	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

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51	159	253	159	50	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	238	252	252	252	237	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	54	227	253	252	239	233	252	57	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	10	60	224	252	253	252	202	84	252	253	122	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	163	252	252	252	253	252	252	96	189	253	167	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	51	238	253	253	190	114	253	228	47	79	255	168	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	48	238	252	252	179	12	75	121	21	0	0	253	243	50	0	0	0	0	0
0	0	0	0	0	0	0	0	38	165	253	233	208	84	0	0	0	0	0	0	253	252	165	0	0	0	0	0
0	0	0	0	0	0	0	7	178	252	240	71	19	28	0	0	0	0	0	0	253	252	195	0	0	0	0	0
0	0	0	0	0	0	0	57	252	252	63	0	0	0	0	0	0	0	0	0	253	252	195	0	0	0	0	0
0	0	0	0	0	0	0	198	253	190	0	0	0	0	0	0	0	0	0	0	255	253	196	0	0	0	0	0
0	0	0	0	0	0	76	246	252	112	0	0	0	0	0	0	0	0	0	0	253	252	148	0	0	0	0	0
0	0	0	0	0	0	85	252	230	25	0	0	0	0	0	0	0	0	7	135	253	186	12	0	0	0	0	0
0	0	0	0	0	0	85	252	223	0	0	0	0	0	0	0	0	7	131	252	225	71	0	0	0	0	0	0
0	0	0	0	0	0	85	252	145	0	0	0	0	0	0	0	48	165	252	173	0	0	0	0	0	0	0	0
0	0	0	0	0	0	86	253	225	0	0	0	0	0	0	114	238	253	162	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	85	252	249	146	48	29	85	178	225	253	223	167	56	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	85	252	252	252	229	215	252	252	252	196	130	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	28	199	252	252	253	252	252	233	145	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	25	128	252	253	252	141	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	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:
0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	67	232	39	0	0	0	0	0
0	0	0	0	62	81	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	180	39	0	0	0	0	0
0	0	0	0	126	163	0	0	0	0	0	0	0	0	0	0	0	0	0	2	153	210	40	0	0	0	0	0
0	0	0	0	220	163	0	0	0	0	0	0	0	0	0	0	0	0	0	27	254	162	0	0	0	0	0	0
0	0	0	0	222	163	0	0	0	0	0	0	0	0	0	0	0	0	0	183	254	125	0	0	0	0	0	0
0	0	0	46	245	163	0	0	0	0	0	0	0	0	0	0	0	0	0	198	254	56	0	0	0	0	0	0
0	0	0	120	254	163	0	0	0	0	0	0	0	0	0	0	0	0	23	231	254	29	0	0	0	0	0	0
0	0	0	159	254	120	0	0	0	0	0	0	0	0	0	0	0	0	163	254	216	16	0	0	0	0	0	0
0	0	0	159	254	67	0	0	0	0	0	0	0	0	0	14	86	178	248	254	91	0	0	0	0	0	0	0
0	0	0	159	254	85	0	0	0	47	49	116	144	150	241	243	234	179	241	252	40	0	0	0	0	0	0	0
0	0	0	150	253	237	207	207	207	253	254	250	240	198	143	91	28	5	233	250	0	0	0	0	0	0	0	0
0	0	0	0	119	177	177	177	177	177	98	56	0	0	0	0	0	102	254	220	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	254	137	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	254	57	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	254	57	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	255	94	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	254	96	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	254	153	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	255	153	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96	254	153	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	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:
4

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	25	100	122	7	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	33	151	208	252	252	252	146	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	40	152	244	252	253	224	211	252	232	40	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	15	152	239	252	252	252	216	31	37	252	252	60	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	96	252	252	252	252	217	29	0	37	252	252	60	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	181	252	252	220	167	30	0	0	77	252	252	60	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	26	128	58	22	0	0	0	0	100	252	252	60	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	157	252	252	60	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	110	121	122	121	202	252	194	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	53	179	253	253	255	253	253	228	35	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	5	54	227	252	243	228	170	242	252	252	231	117	6	0	0	0	0	0	0	0
0	0	0	0	0	0	0	6	78	252	252	125	59	0	18	208	252	252	252	252	87	7	0	0	0	0	0	0
0	0	0	0	0	0	5	135	252	252	180	16	0	21	203	253	247	129	173	252	252	184	66	49	49	0	0	0
0	0	0	0	0	3	136	252	241	106	17	0	53	200	252	216	65	0	14	72	163	241	252	252	223	0	0	0
0	0	0	0	0	105	252	242	88	18	73	170	244	252	126	29	0	0	0	0	0	89	180	180	37	0	0	0
0	0	0	0	0	231	252	245	205	216	252	252	252	124	3	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	207	252	252	252	252	178	116	36	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	13	93	143	121	23	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	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:
2

MNIST Dataset – Reshape & Normalization

```
import numpy as np

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")

# n x 28 x 28 pixel testing data
X_test = np.load("/homea/hpclab/train001/data/mnist/x_test.npy")

# n x 1 testing labels
Y_test = np.load("/homea/hpclab/train001/data/mnist/y_test.npy")

# reshape for the neural network 28 x 28 = 784
RESHAPED= 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
# specify type
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

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

# data output: number of values / sample
print(X_train.shape[1], 'input pixel values per train samples')
print(X_test.shape[1], 'input pixel values per test samples')

# data output: character
print(X_train[0])
```

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

- **Reshape from 28 x 28 matrix of pixels to 784 pixel values considered to be the input for the neural networks later (unroll in one large vector, also called 'vectorization')**
- **Normalization is added for mathematical convenience since the computing with numbers get easier (not too large)**

MNIST Dataset – Reshape & Normalization – Example

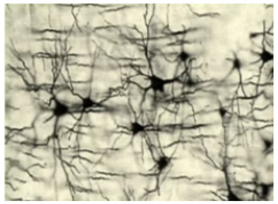
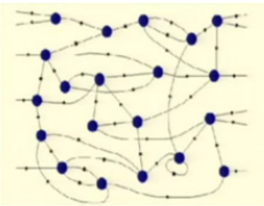
(one large vector instead of 28 * 28 pixel matrix)

```
[train001@jrl09 mnist]$ python mnist-reshape.py
(60000, 'train samples')
(10000, 'test samples')
(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
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.3137255  0.6117647  0.41960785 0.99215686
 0.99215686 0.8039216  0.04313726 0.         0.16862746 0.6039216
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

(numbers are between 0 and 1 for mathematical convenience)

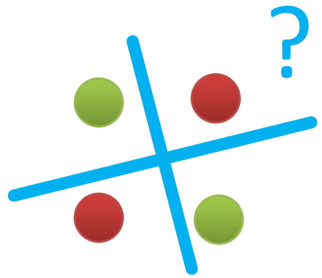
Learning Models from Biological Inspiration

- Biological Inspiration
  - Humans learn (a biological function) → machines can learn
  - Means we are interested in ‘replicating’ the ‘biological function’
- Approach: Replicating the ‘biological structure’
  - Neurons connected to synapses (large number)
  - Action of neurons depends on ‘stimula of different synapses’
  - Synapses have ‘weights’
  - Principle: neurons are in the following like a ‘single perceptron’
  - Neural network: put together a ‘bunch of perceptrons’ in layers
  - Deep learning network: create many layers with ‘smart functionalites‘

Perceptron Model vs. ANN

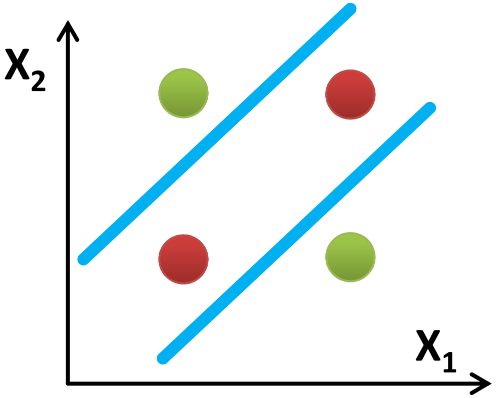
- Simple perceptrons fail: ‘not linearly seperable’



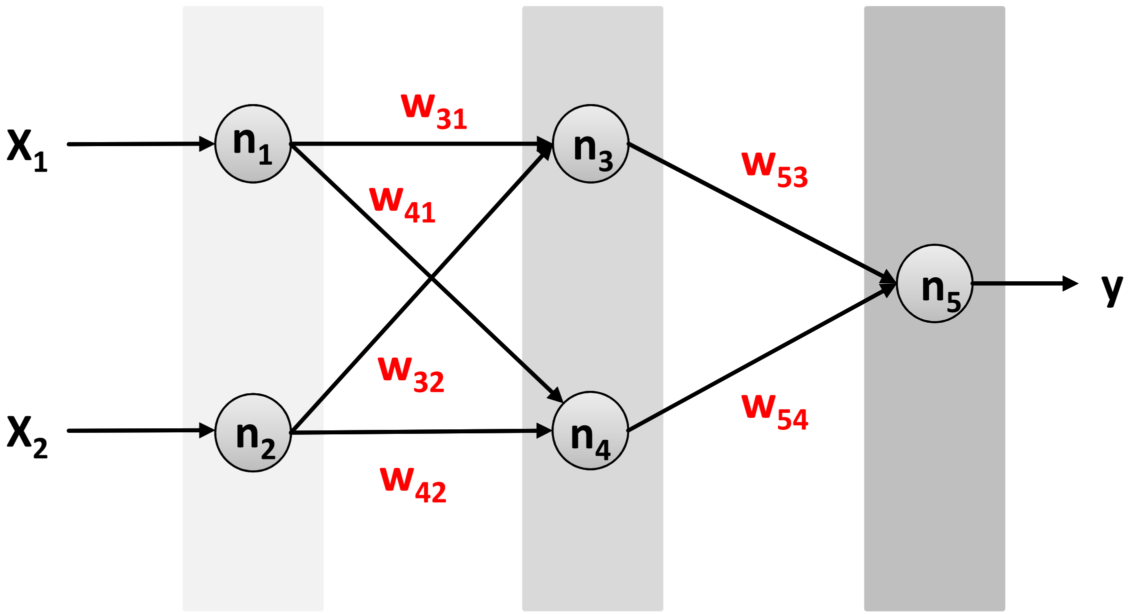
X_1	X_2	Y
0	*0*	*-1*
1	*0*	*1*
0	*1*	*1*
1	*1*	*-1*

Labelled Data Table

(Idea: instances can be classified using two lines at once to model XOR)

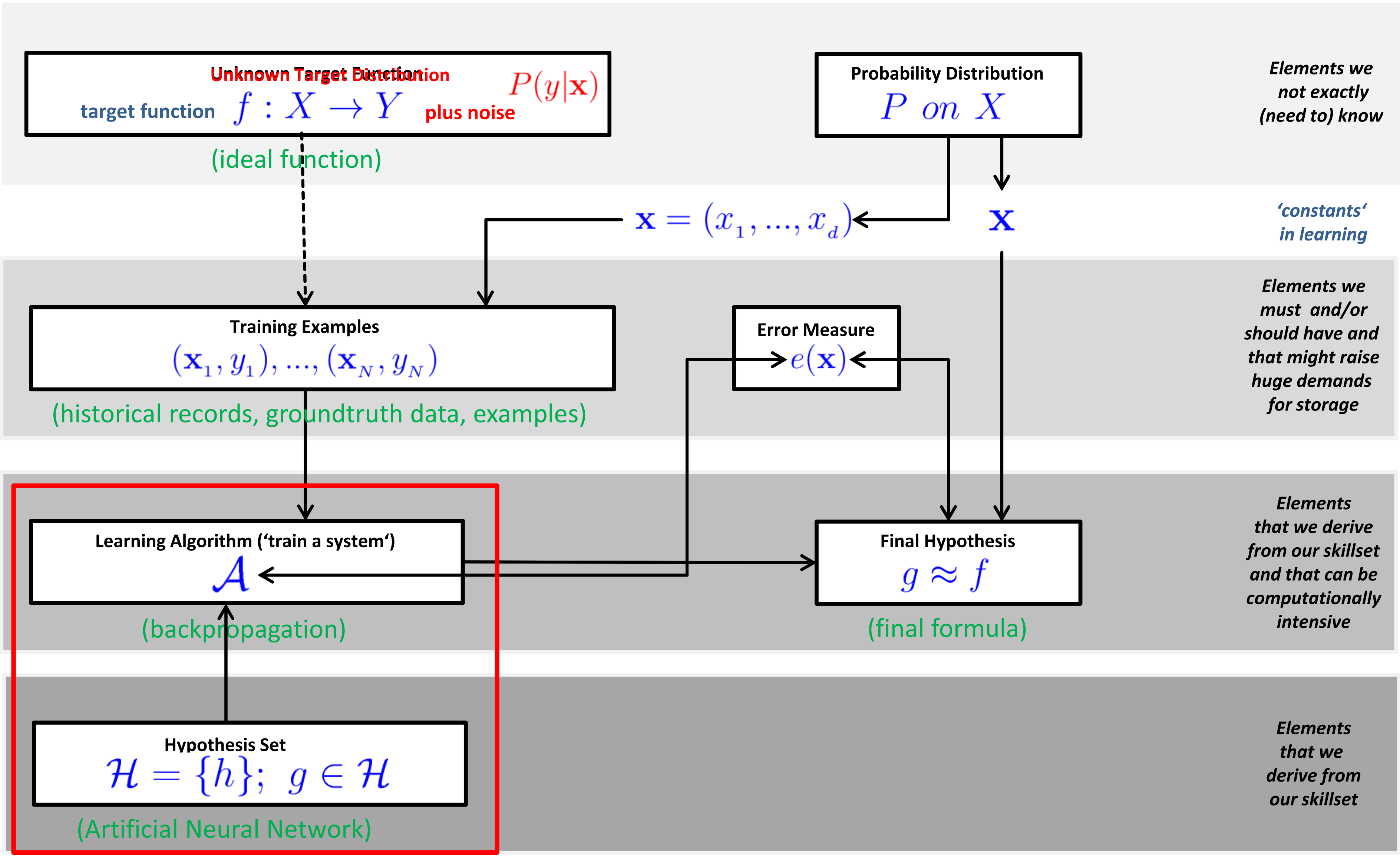


Decision Boundary



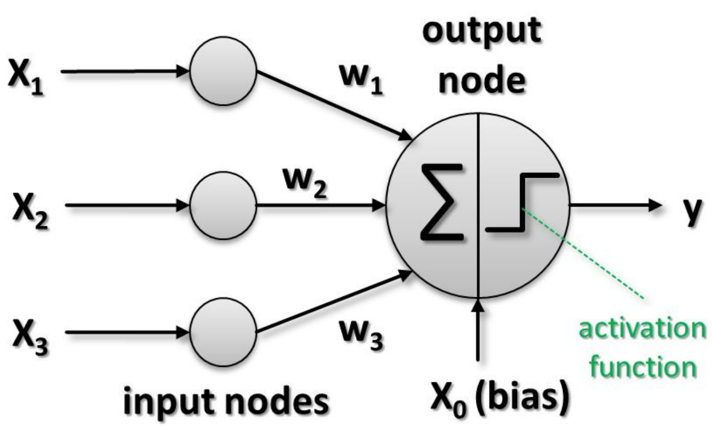
Two-Layer, feed-forward Artificial Neural Network topology

Supervised Learning – Artificial Neural Network (ANN)

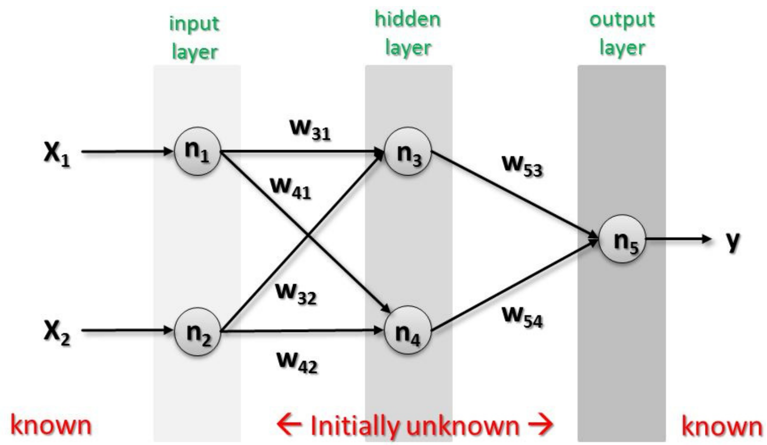


Artificial Neural Network – Feature Engineering & Layers

- Approach: Prepare data before
  - Classical Machine Learning
  - Feature engineering
  - Dimensionality reduction techniques
  - Low number of layers (many layers computationally infeasible in the past)
  - Very succesful for speech recognitition (‘state-of-the-art in your phone‘)

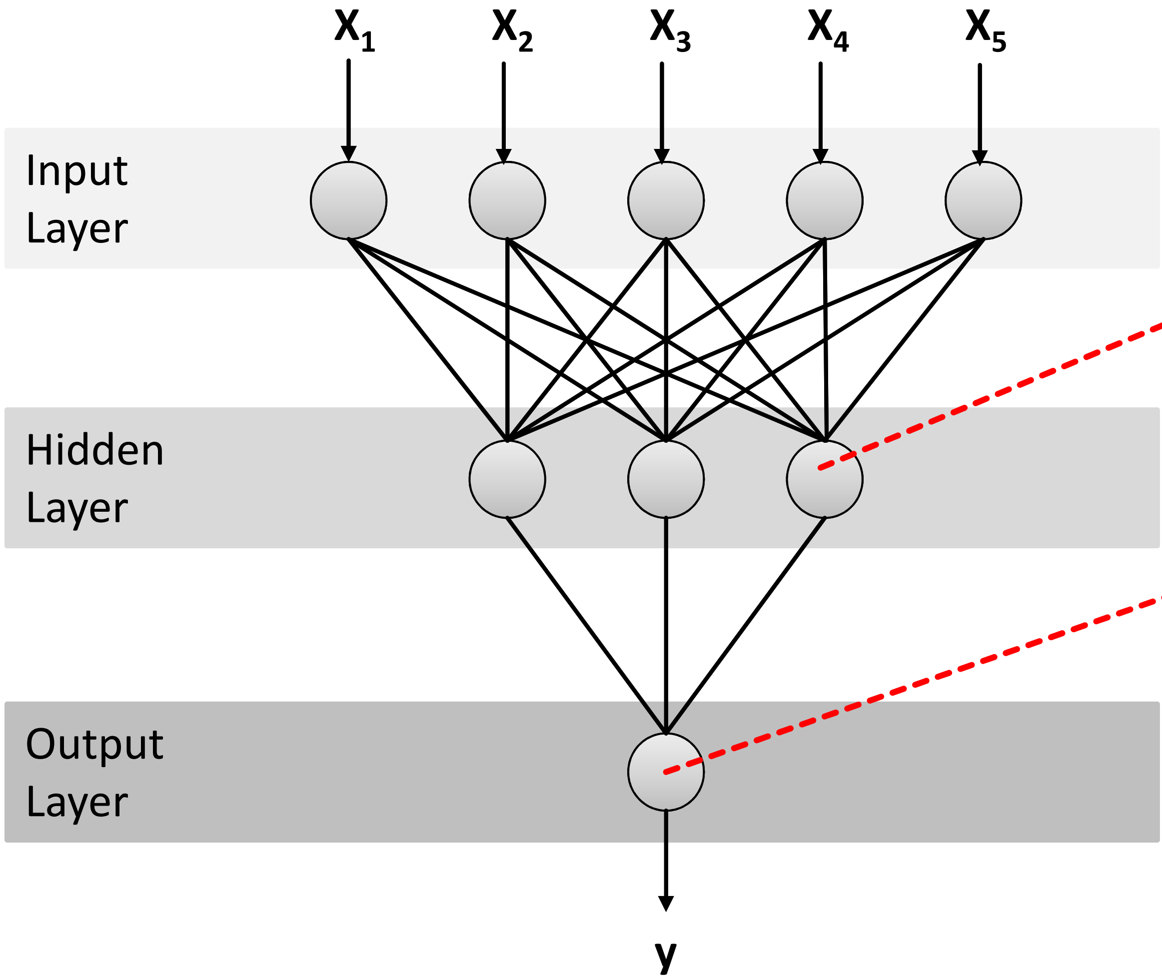


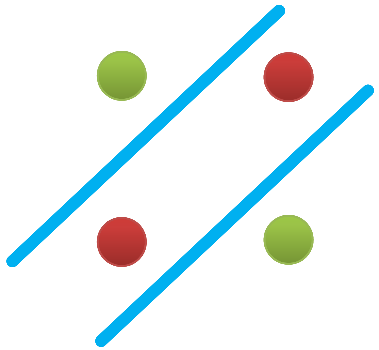
(Perceptron model: designed after human brain neuron)



(Artificial neural network two layer feed – forward)

Artificial Neural Networks (ANN) – Layers & Nodes





- Think each hidden node as a 'simple perceptron' that each creates one hyperplane
- Think the output node simply combines the results of all the perceptrons to yield the 'decision boundary' above
- Feed-forward neural network: nodes in one layer are connected only to the nodes in the next layer ('a constraint of network construction')

ANN - Learning Algorithm & Optimization

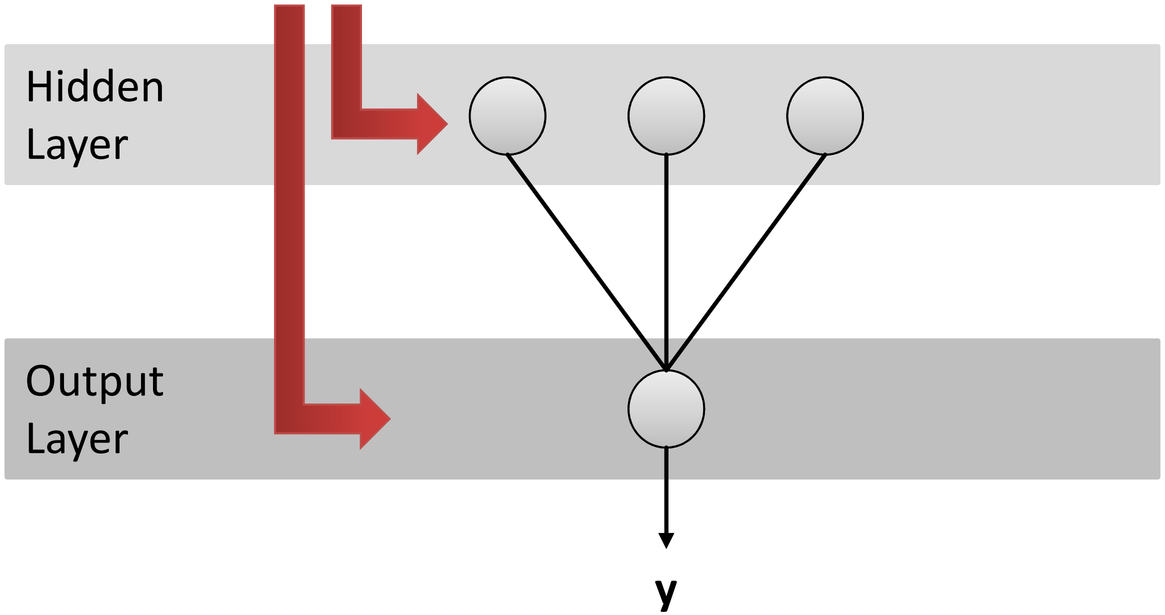
- Determine a set of weights **w** that
  'minimize the total sum of squared errors':

y = sign (w · x)

Linear perceptron

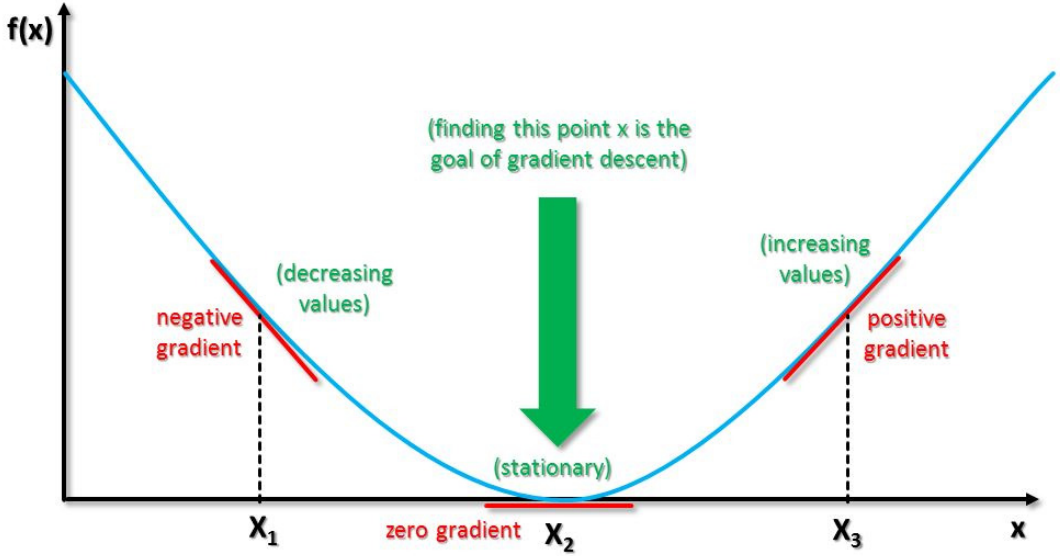
Sum of squared errors depend on w, because predicted class y is a 'function of the weights' assigned to the hidden and output nodes

$$E(w) = \frac{1}{2} \sum_{i=1}^{N} \boxed{(y_i - \acute{y}_i)}^2$$



- **Cost function is quadratic in its parameters and a global minimum can be easily found**
- **Other loss functions possible, e.g. categorical cross-entropy**
- **Loss functions used for one single training sample**
- **Cost functions used over the entire training dataset**

Gradient Descent Method (1)



(minimization: substract gradient term because we move towards local minima)

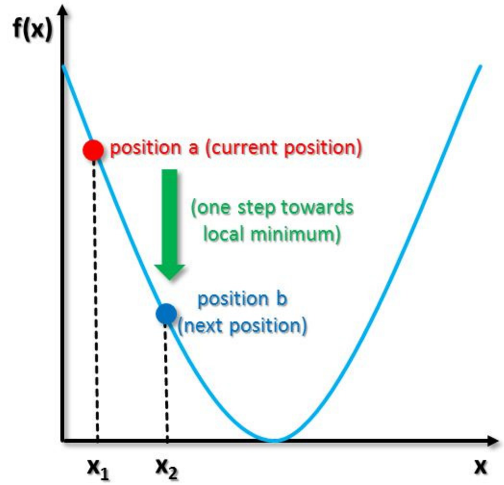
(the derivative of f with respect to a)

$$b = a - \gamma \; \underbrace{\nabla \, f(a)}_{\text{(gradient term is steepest ascent)}}$$

(old position before the step)

(new position after the step)

(weighting factor known as step-size, can change at every iteration, also called learning rate)



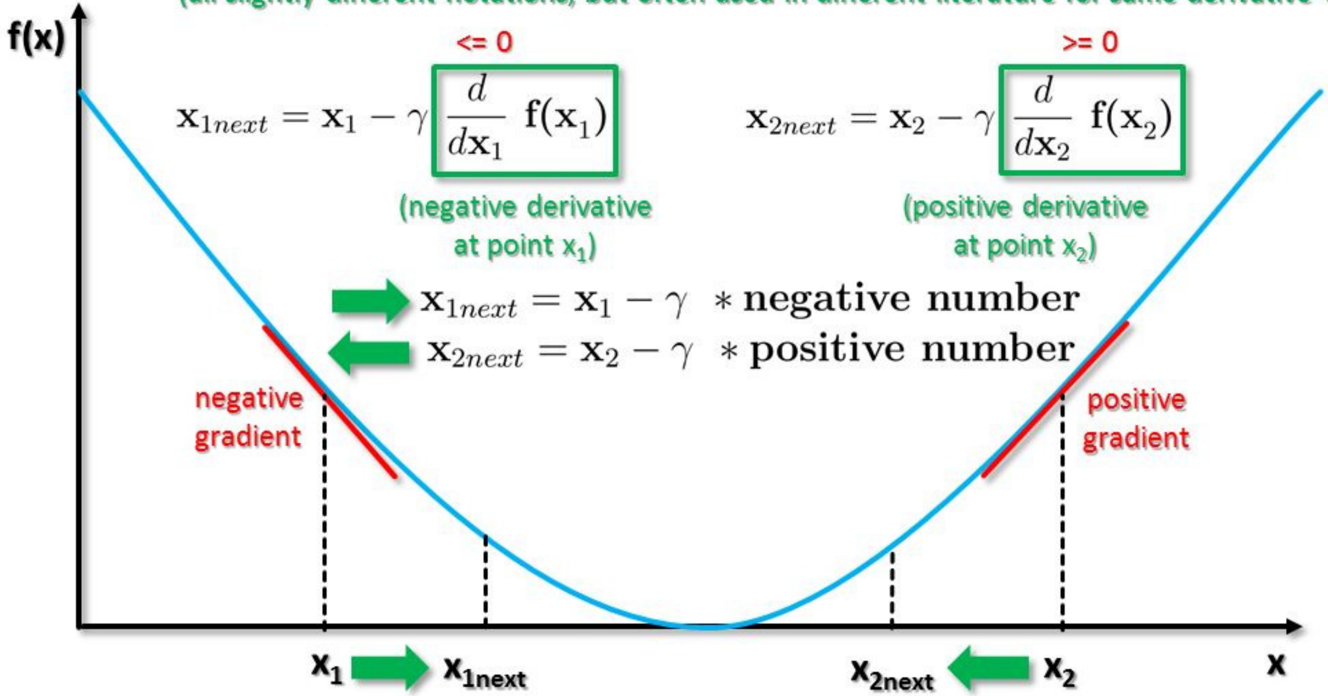
[19] Big Data Tips, Gradient Descent

Gradient Descent Method (2)

- **Gradient Descent (GD) uses all the training samples available for a step within a iteration**
- **Stochastic Gradient Descent (SGD) converges faster: only one training samples used per iteration**

$$\mathrm{b} = \mathbf{a} - \gamma \, \nabla \, \mathrm{f}(\mathbf{a}) \qquad \mathrm{b} = \mathbf{a} - \gamma \, \frac{\partial}{\partial \mathbf{a}} \, \mathrm{f}(\mathbf{a}) \qquad \mathrm{b} = \mathbf{a} - \gamma \, \frac{d}{d\mathbf{a}} \, \mathrm{f}(\mathbf{a})$$

(all slightly different notations, but often used in different literature for same derivative term)

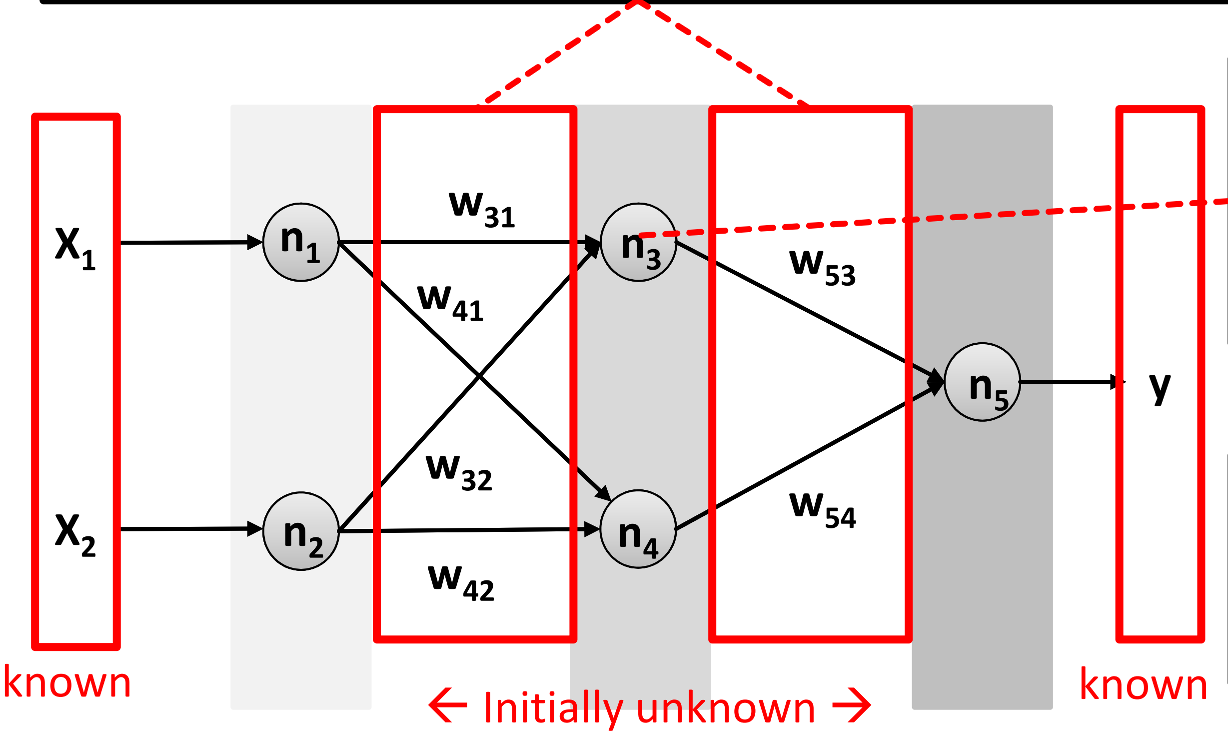


[19] Big Data Tips, Gradient Descent

ANN – Backpropagation Algorithm (BP) Basics

- One of the most widely used algorithms for supervised learning
  - Applicable in multi-layered feed-forward neural networks

- **'Gradient descent method' can be used to learn the weights of the output and hidden nodes of a artificial neural network**

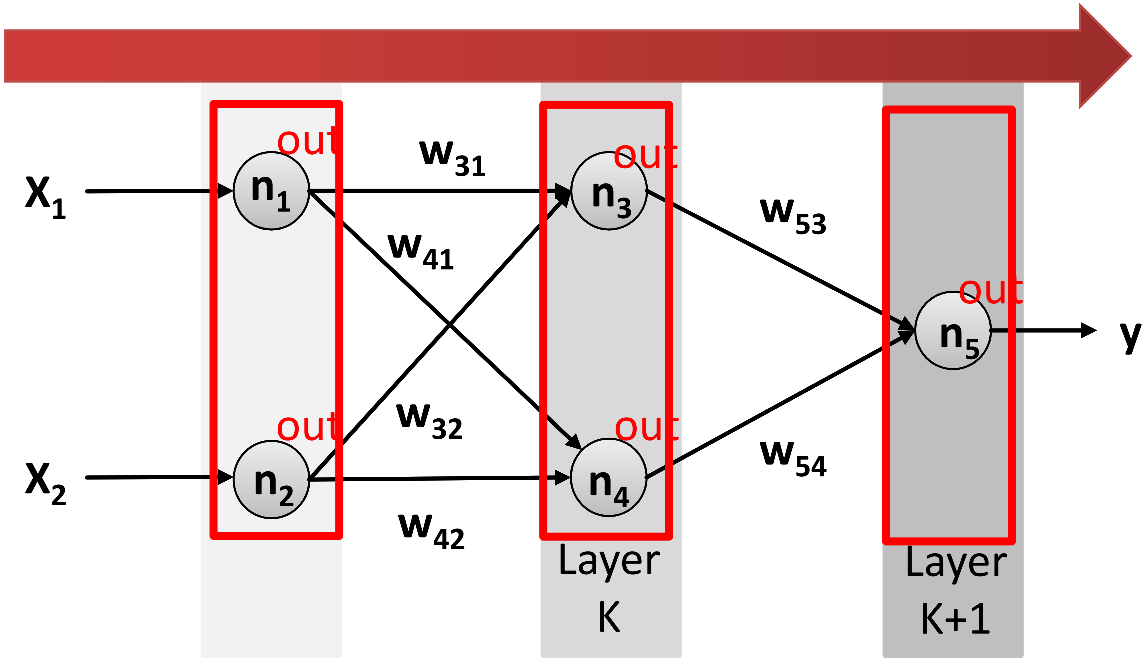


- **Hidden nodes problem: computing error term hard: ∂ E / ∂ wj**
- **Their Output values are unknown to us (here)…**

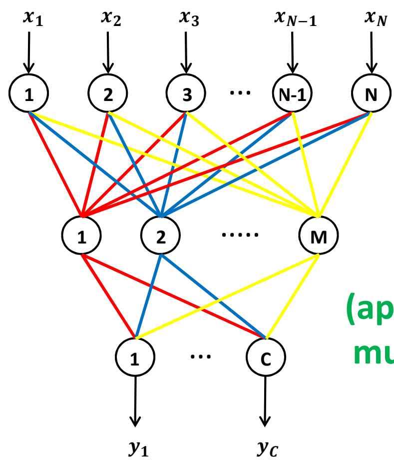
- **The backpropagation algorithm solves exactly this problem with two phases per iteration(!)**

ANN – Backpropagation Algorithm Forward Phase

1. 'Forward phase (does not change weights, re-use old weights)':
   - Weights obtained from the previous iteration are used to compute the output value of each neuron in the network ('initialize weights randomly')
   - Computation progresses in the 'forward direction', i.e. outputs 'out' of the neurons at level k are computed prior to level k+1



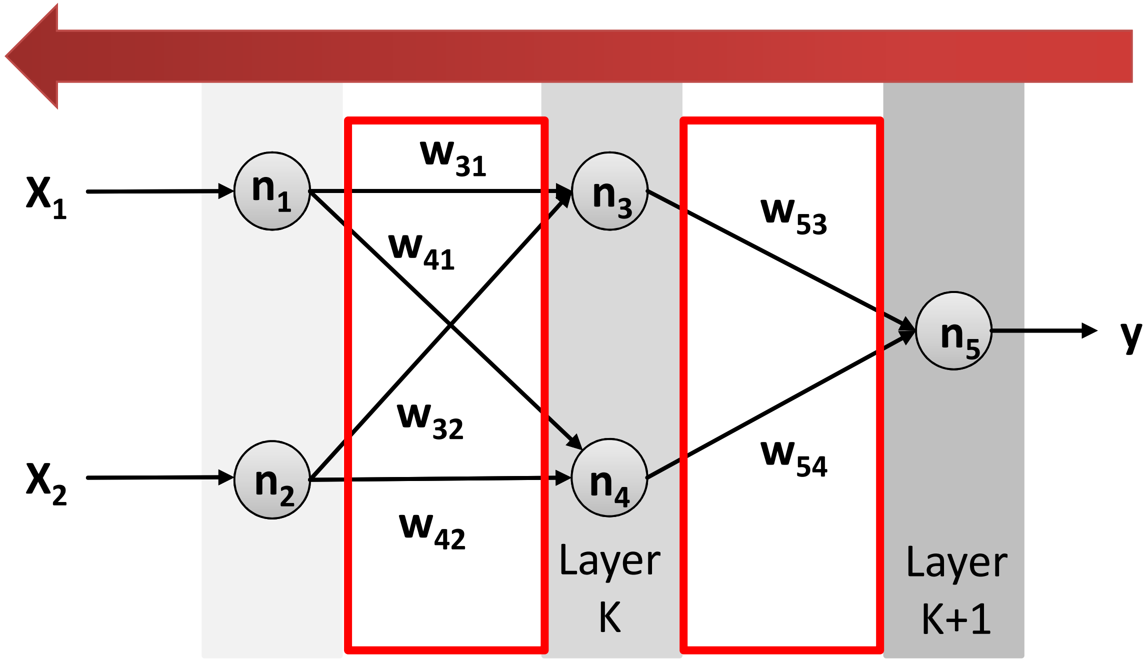
- **Use corresponding 'activation function' but with 'old weights'**



(applicable for multiclass to)

ANN – Backpropagation Algorithm Backward Phase

2. 'Backward phase ('learning' → change the weights in the ANN)':
   - Weight update formula is applied in the 'reverse direction'
   - Weights at level K + 1 are updated before the weights at level k
   - Idea: use the errors for neurons at layer k + 1 to estimate errors for neurons at layer k



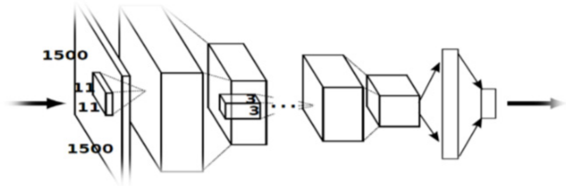
$$w_j <- wj - \lambda \frac{\partial E(w)}{\partial w_j}$$

weight update formula
of the 'gradient descent method'

Now that can compute
the error one-by-one

Deep Learning in Clouds – Using Python & Keras

- Cloud Computing
  - Enables large-scale deep learning from 'big data'
  - Deep learning library Keras on top of TensorFlow, MXNet, CNTK, and many others
  - (cf. Practical Lecture 0.1)




```
keras.layers.Dense(units,
                   activation=None,
                   use_bias=True,
                   kernel_initializer='glorot_uniform',
                   bias_initializer='zeros',
                   kernel_regularizer=None,
                   bias_regularizer=None,
                   activity_regularizer=None,
                   kernel_constraint=None,
                   bias_constraint=None)
```

Keras

[6] Keras Python Deep Learning Library

```
keras.optimizers.SGD(lr=0.01,
                     momentum=0.0,
                     decay=0.0,
                     nesterov=False)
```

- **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, MXNet, CNTK, or Theano**
- **The key idea behind the Keras tool is to enable faster experimentation with deep networks**
- **Created deep learning models run seamlessly on CPU and GPU via low-level frameworks**

> **Lecture 7 will provide more examples of using backpropagation and loss functions in Clouds**

ANN – MNIST Dataset – Create ANN Blueprint

✓ Data Preprocessing done (i.e. data normalization, reshape, etc.)

1. Define a neural network topology
   - Which layers are required?
   - Think about input layer need to match the data – what data we had?
   - Maybe hidden layers?
   - Think Dense layer – Keras?
   - Think about final Activation as Softmay (cf. Day One) → output probability
2. Compile the model → model representation for Tensorflow et al.
   - Think about what loss function you want to use in your problem?
   - What is your optimizer strategy, e.g. SGD
3. Fit the model → the model learning takes place
   - How long you want to train (e.g. NB_EPOCHS)
   - How much samples are involved (e.g. BATCH_SIZE)

ANN – MNIST Dataset – Parameters & Data Normalization

```python
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils

# ANN parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 128
OPTIMIZER = 'SGD'

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")

# n x 28 x 28 pixel testing data
X_test = np.load("/homea/hpclab/train001/data/mnist/x_test.npy")

# n x 1 testing labels
Y_test = np.load("/homea/hpclab/train001/data/mnist/y_test.npy")

# reshape for the neural network 28 x 28 = 784
RESHAPED= 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
# specify type
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

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

# data output: number of values / sample
print(X_train.shape[1], 'input pixel values per train samples')
print(X_test.shape[1], 'input pixel values per test samples')
```

- **NB_CLASSES: 10 Class Problem**
- **NB_EPOCH: number of times the model is exposed to the 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**
- **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]**

[11] A. Gulli et al.

ANN – MNIST Dataset – A Simple Model & Softmax

- 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‘ 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

```
# convert label vectors to binary matrices of classes
Y_train = np_utils.to_categorical(Y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(Y_test, NB_CLASSES)

# simple ANN model
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
model.add(Activation('softmax'))
model.summary()

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

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

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

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

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

- Train the model (‘fit‘)

ANN – Train MNIST Dataset – Check Output

```
[train001@jrl09 scripts]$ more mnist_out.5522445
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
_________________________________________________________________
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
_________________________________________________________________
Epoch 1/200

  128/60000 [..............................] - ETA: 5:56 - loss: 2.5313 - acc: 0.0625
 2816/60000 [>.............................] - ETA: 16s - loss: 2.3530 - acc: 0.1225
 5888/60000 [=>............................] - ETA: 7s - loss: 2.2351 - acc: 0.2040
 8960/60000 [===>..........................] - ETA: 5s - loss: 2.1290 - acc: 0.2907
```

Model Evaluation – Testing Phase & Performance Metrics

Counting per sample		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

(100% accuracy in learning often points to problems using machine learning methos in practice)

- Accuracy (usually in %)

$$Accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions}$$

- Error rate

$$Error\ rate = \frac{number\ of\ wrong\ predictions}{total\ number\ of\ predictions}$$

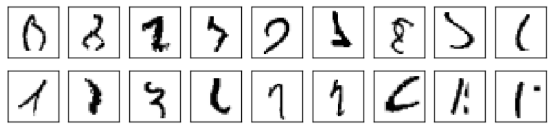
ANN – MNIST Dataset – A Simple Model – Output

```
[train001@jrl09 scripts]$ tail mnist_out.5522445
   32/10000 [..............................] - ETA: 3s
 1504/10000 [===>..........................] - ETA: 0s
 3008/10000 [========>.....................] - ETA: 0s
 4544/10000 [============>.................] - ETA: 0s
 5952/10000 [================>.............] - ETA: 0s
 7392/10000 [=====================>........] - ETA: 0s
 8768/10000 [=========================>....] - ETA: 0s
10000/10000 [==============================] - 0s 36us/step
('Test score: ', 0.2727356147527695)
('Test accuracy: ', 0.9228)
```

(92,2% accuracy
is already quite good)

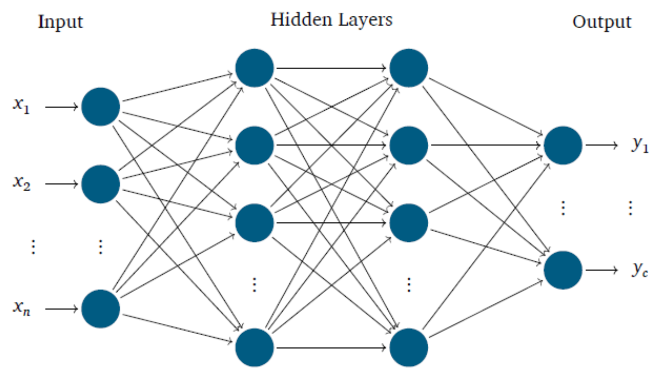
ANN – MNIST Dataset – Extend ANN Blueprint

- ✓ Data Preprocessing done (i.e. data normalization, reshape, etc.)
- ✓ Initial ANN topology existing
- ✓ Initial setup of model works → not 100% (create, compile, fit)



(some dataset samples just hard to learn from)

- Extend 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?
  - Think Dense layer – Keras?
  - Think about final Activation as Softmax → output probability



(two hidden layer usually show good performance in classification tasks)

ANN – MNIST Dataset – Add Two Hidden Layers

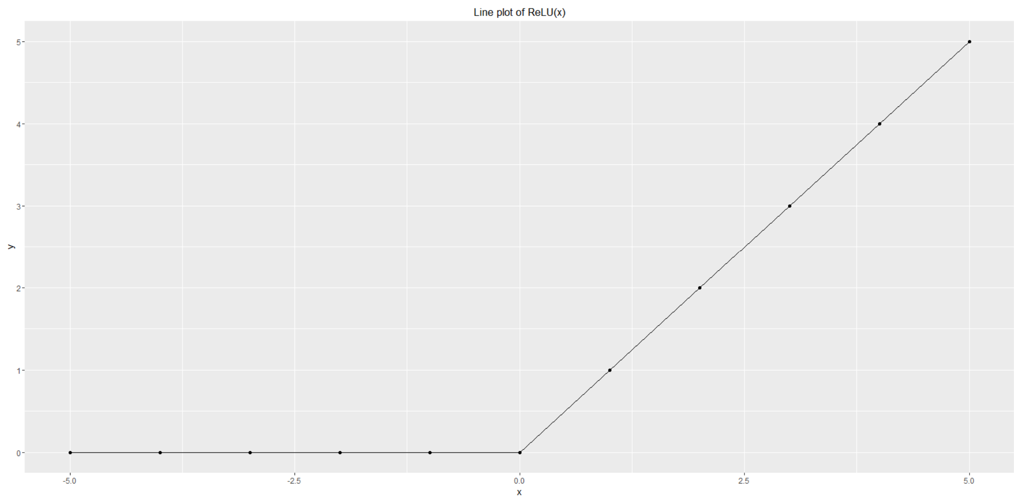
- A hidden layer in an ANN can be represented by a fully connected Dense layer in Keras by just specifying the number of hidden neurons in the hidden layer

- 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

```
# simple ANN model
model = Sequential()
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))
model.add(Activation('softmax'))
model.summary()

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

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

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

Activation Function – Rectified Linear Unit (ReLU)

- Definition
  - Simple, yet effective
  - f(x) = max(0, x)
- Selected Facts
  - Provides good results in training neural networks
  - Experts refer to this also as a 'ramp function'




```
> f = function(x){pmax(0,x)}
> x = -5:5
> y = f(x)
> dataset = data.frame(x,y)
```

(simple R script implementing the ReLU function)

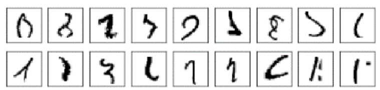
[16] Big Data Tips, ReLU Neural Network

- **The activation function Rectified Linear Unit (ReLU) is defined as f(x) = max(0,x)**
- **ReLU is more recently used in the training of neural networks and deep learning networks**
- **ReLU just returns 0 for negative values and grows linearly for only positive values**

ANN – MNIST Dataset – Two Hidden Layers & Check Output

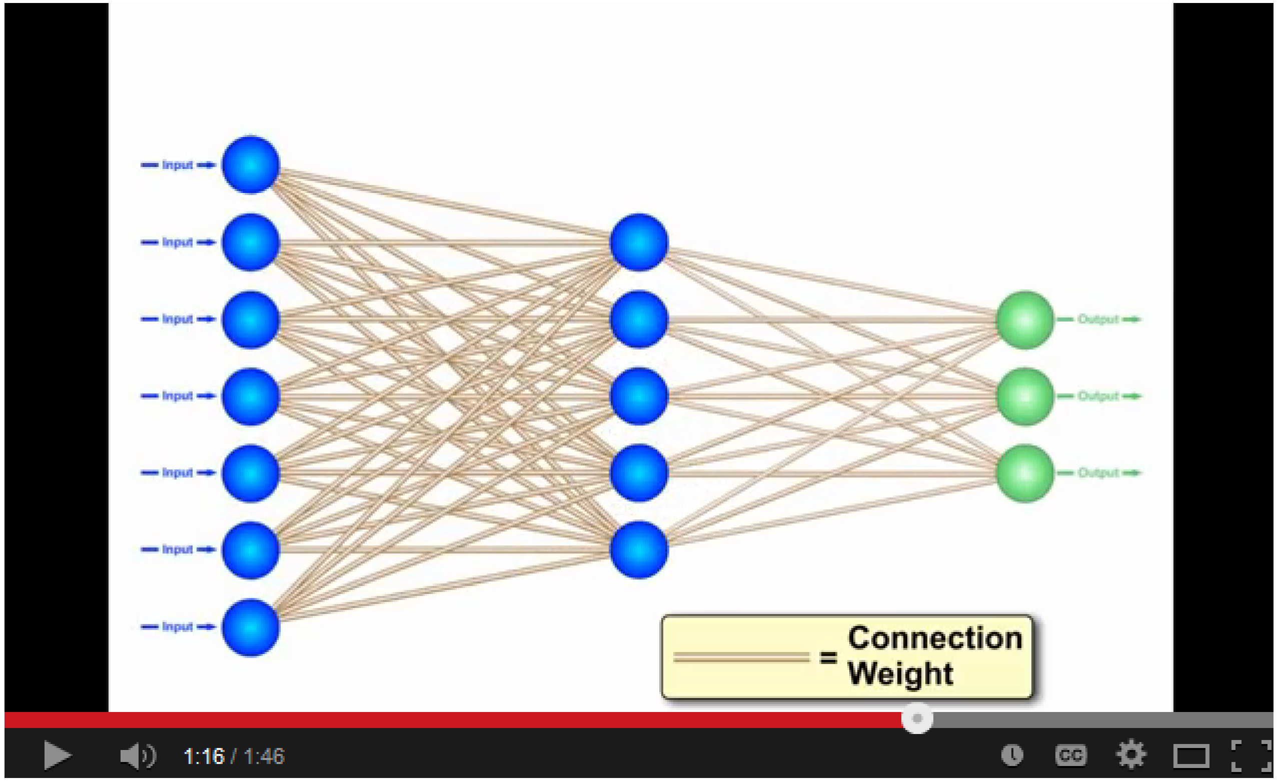
```
[train001@jrl06 scripts]$ more ann-2hidden-mnist_out.5522545
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
_________________________________________________________________
Layer (type)                 Output 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
_________________________________________________________________
Epoch 1/200

  128/60000 [..............................] - ETA: 6:02 - loss: 2.3471 - acc: 0.0781
 2560/60000 [>.............................] - ETA: 18s - loss: 2.3044 - acc: 0.0902
 4992/60000 [=>............................] - ETA: 9s - loss: 2.2580 - acc: 0.1332
 7552/60000 [==>...........................] - ETA: 6s - loss: 2.2119 - acc: 0.2080
```

```
 5632/10000 [===============>..............] - ETA: 0s
 7040/10000 [====================>.........] - ETA: 0s
 8448/10000 [========================>.....] - ETA: 0s
 9824/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - 0s 37us/step
('Test score: ', 0.07480177137681167)
('Test accuracy: ', 0.9777)
```



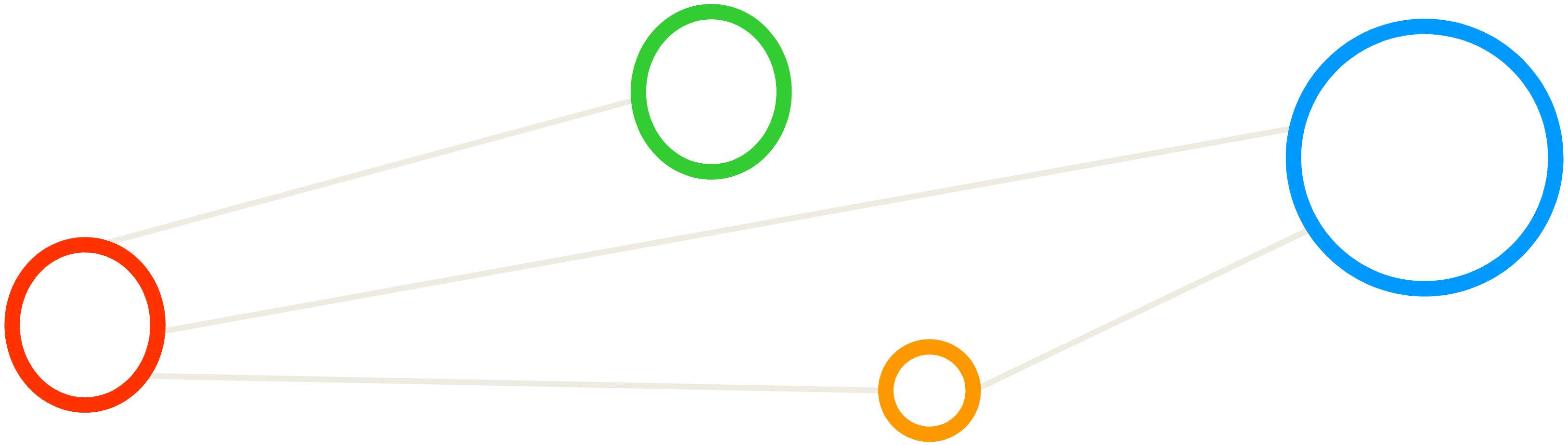
(97,7% accuracy
is a good improvement)

[Video] Towards Multi-Layer Perceptrons



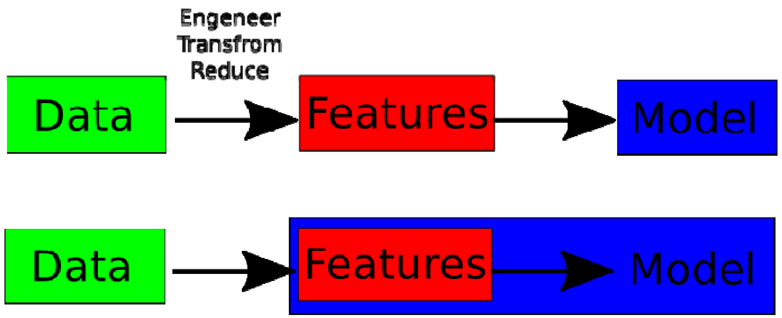
[5] YouTube Video, Neural Networks – A Simple Explanation

Deep Learning Models & Big Data

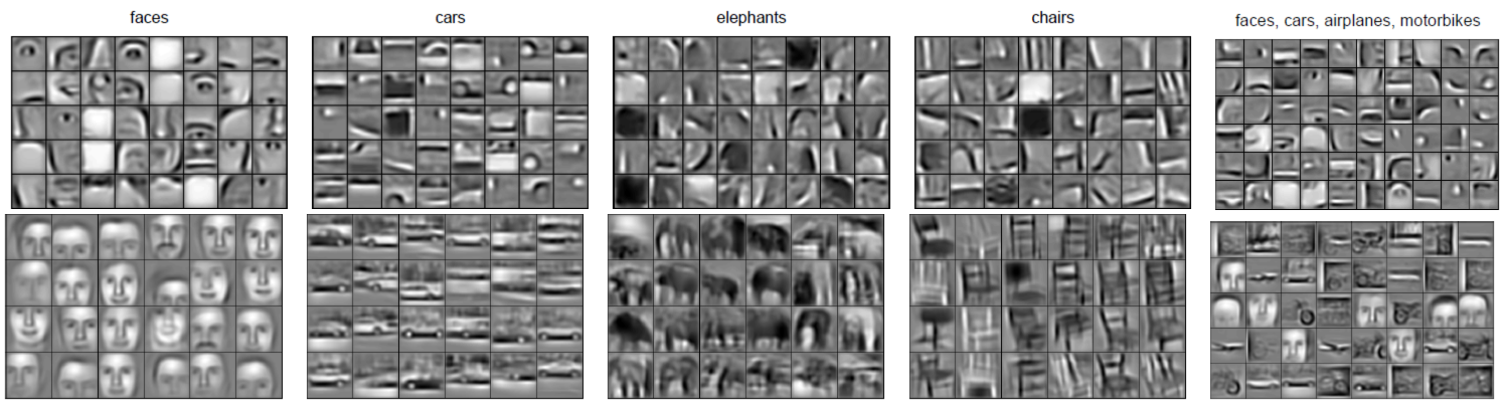


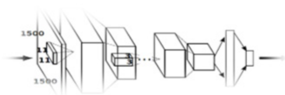
Conceptual Idea - ‘Big Data‘ drives Deep Learning Models

- Approach: Learn Features
  - Classical Machine Learning
  - (Powerful computing evolved)
  - Deep (Feature) Learning



  - Very succesful for image recognition and other emerging areas
  - Assumption: data was generated by the interactions of many different factors on different levels (i.e. form a hierarchical representation)

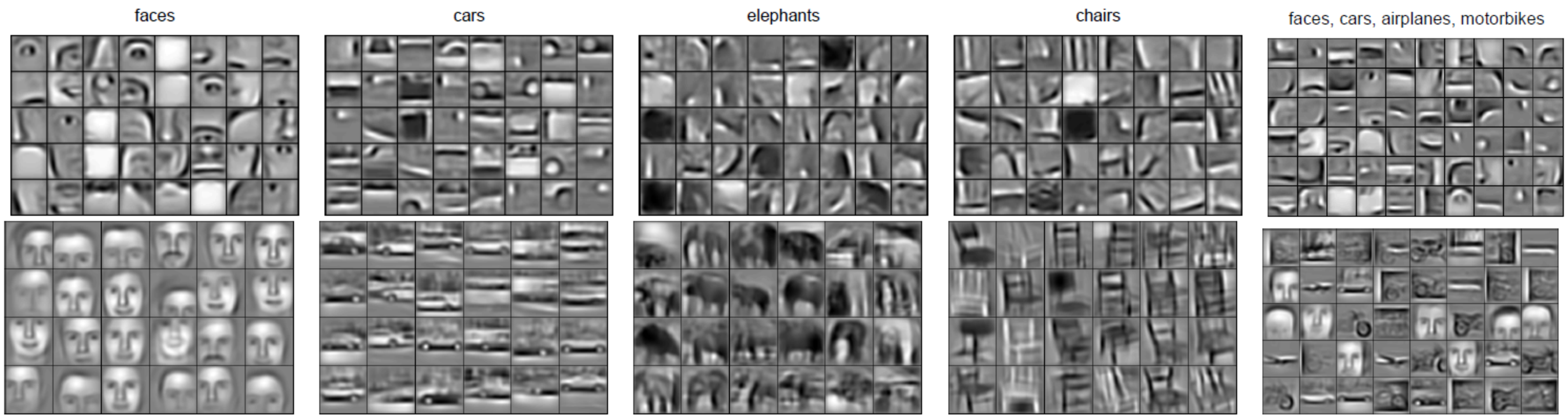


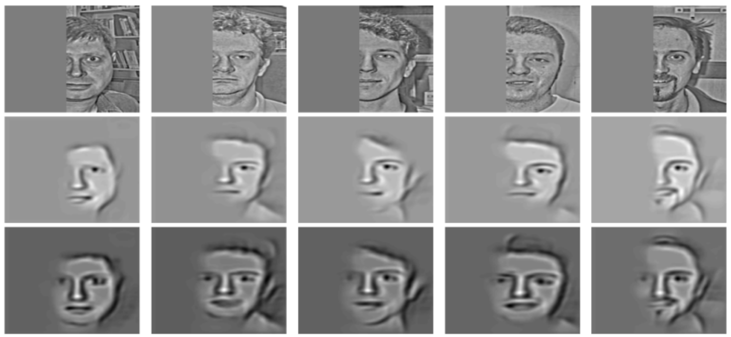


[7] H. Lee et al.

➢ **Practical Lecture 6.1 given by Dr. Gabriele Cavallaro provides examples from remote sensing**

Deep Learning – Feature Learning Benefits





[7] H. Lee et al.

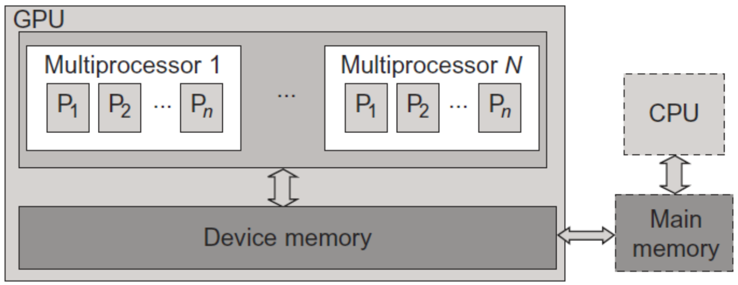
- **Traditional machine learning applied feature engineering before modeling**
- **Feature engineering requires expert knowledge, is time-consuming and a often long manual process, requires often 90% of the time in applications, and is sometimes even problem-specific**
- **Deep Learning enables feature learning promising a massive time advancement**

> ➢ **Practical Lecture 6.1 given by Dr. Cavallaro provides application examples in remote sensing**

Many-core GPUs further Drives Success of Deep Learning

- **Graphics Processing Unit (GPU) is great for data parallelism and task parallelism**
- **Compared to multi-core CPUs, GPUs consist of a many-core architecture with hundreds to even thousands of very simple cores executing threads rather slowly**

- Use of very many simple cores
  - High throughput computing-oriented architecture
  - Use massive parallelism by executing a lot of concurrent threads slowly
  - Handle an ever increasing amount of multiple instruction threads
  - CPUs instead typically execute a single long thread as fast as possible
- Many-core GPUs are used in large clusters and within massively parallel supercomputers today
  - Named General-Purpose Computing on GPUs (GPGPU)



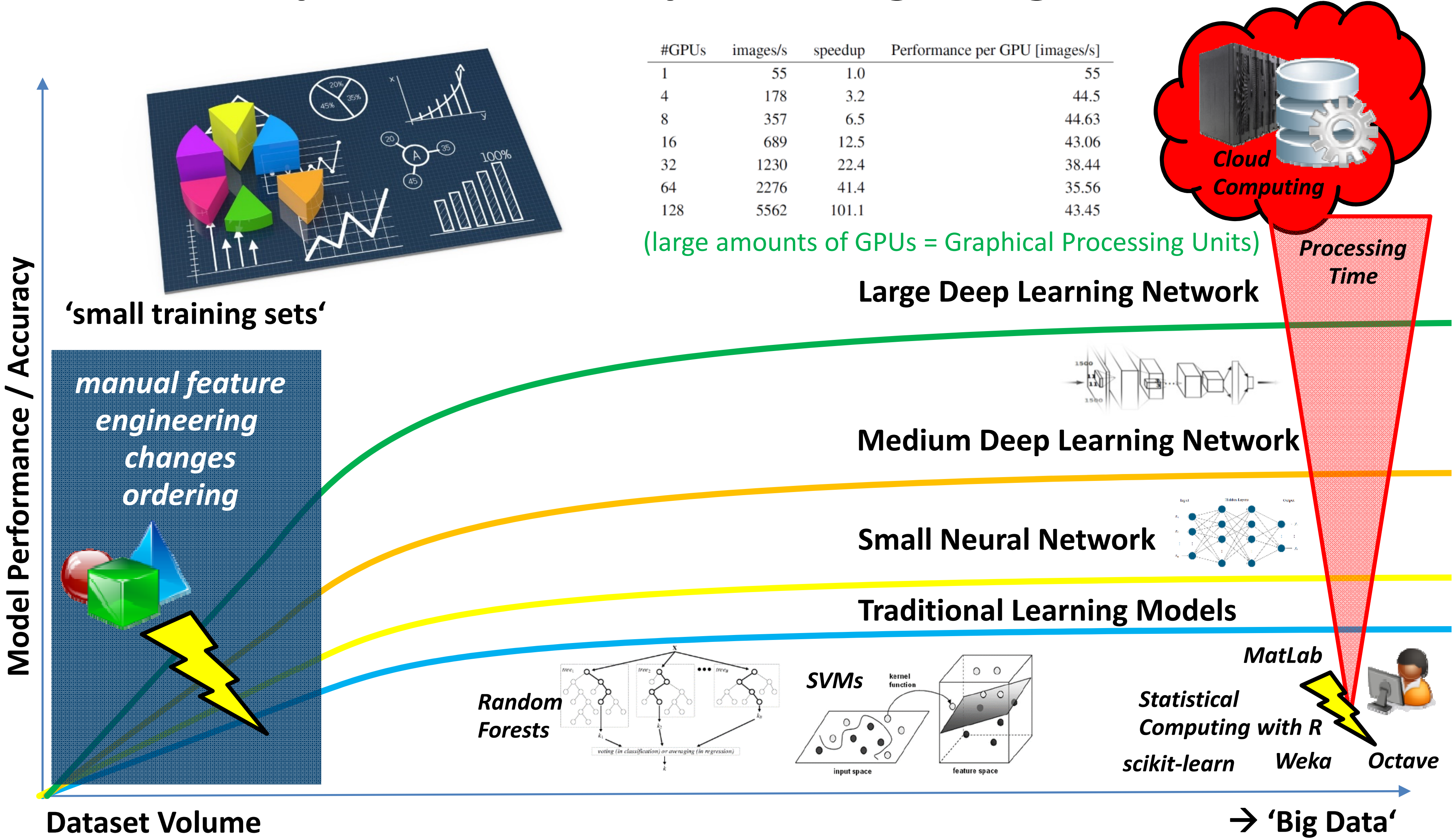
[8] Distributed & Cloud Computing Book

➢ **Practical Lecture 6.1 given by Dr. Cavallaro shows how GPGPUs are used in remote sensing**

Relationship Machine/Deep Learning & Big Data & Clouds

#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

(large amounts of GPUs = Graphical Processing Units)



Deep Learning – Key Properties & Application Areas

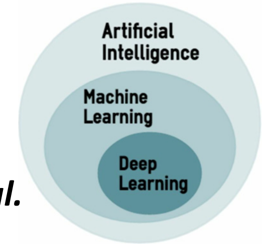
- **In Deep Learning networks are many layers between the input and output layers enabling multiple processing layers that are composed of multiple linear and non-linear transformations**
- **Layers are not (all) made of neurons (but it helps to think about this analogy to understand them)**
- **Deep Learning performs (unsupervised) learning of multiple levels of features whereby higher level features are derived from lower level features and thus form a hierarchical representation**

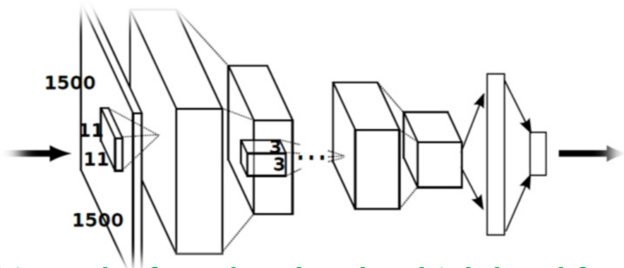
- Application before modeling data with other models
  - Create better data representations and create deep learning models to learn these data representations from large-scale unlabeled data
- Application areas
  - Automatic speech recognition
  - Natural language processing
  - Bioinformatics
  - Computer vision
  - ...

(Deep Learning is often characterized as 'buzzword')

(Deep Learning is often 'just' called rebranding of traditional neural networks)

[11] A. Gulli et al.

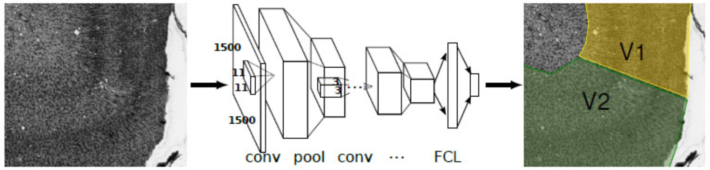




(hierarchy from low level to high level features)

Deep Learning Architectures

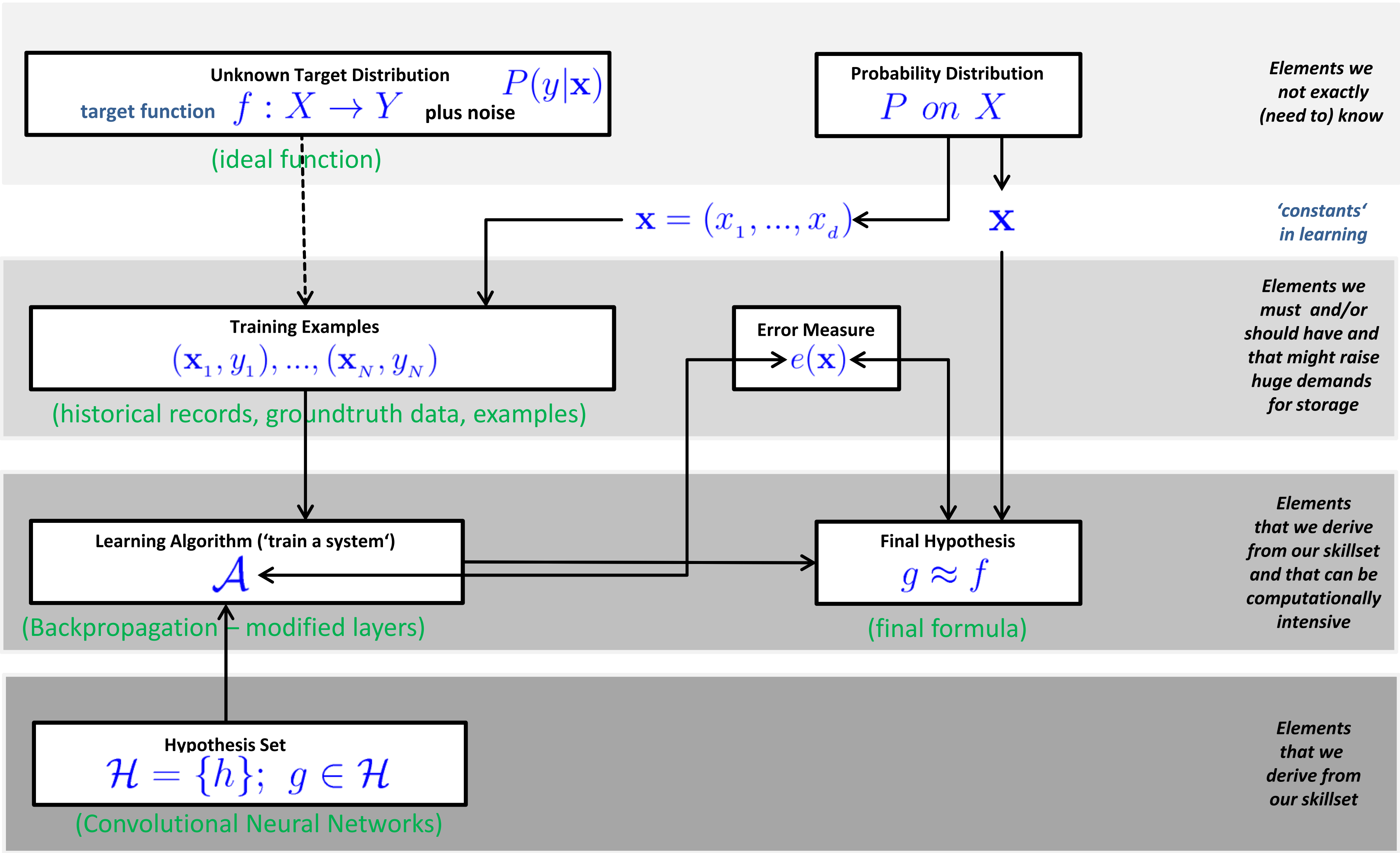
- Deep Neural Network (DNN)
  - ‘Shallow ANN‘ approach with many hidden layers between input/output
- Convolutional Neural Network (CNN, sometimes ConvNet)
  - Connectivity pattern between neurons is like animal visual cortex



- Deep Belief Network (DBN)
  - Composed of mult iple layers of variables; only connections between layers
- Recurrent Neural Network (RNN)
  - ‘ANN‘ but connections form a directed cycle; state and temporal behaviour

- **Deep Learning architectures can be classified into Deep Neural Networks, Convolutional Neural Networks, Deep Belief Networks, and Recurrent Neural Networks all with unique characteristica**
- **Deep Learning needs ‘big data‘ to work well & for high accuracy – works not well on sparse data**

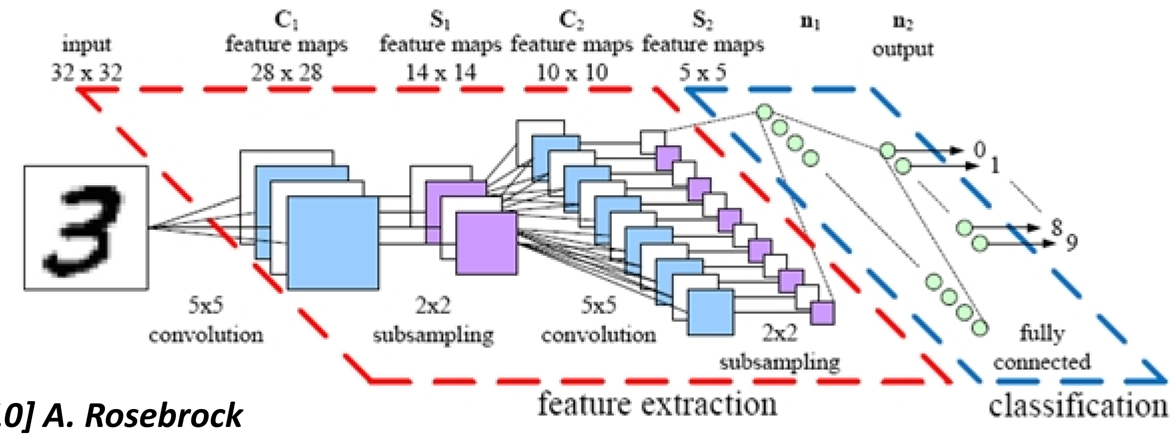
Convolutional Neural Networks (CNNs) Learning Model



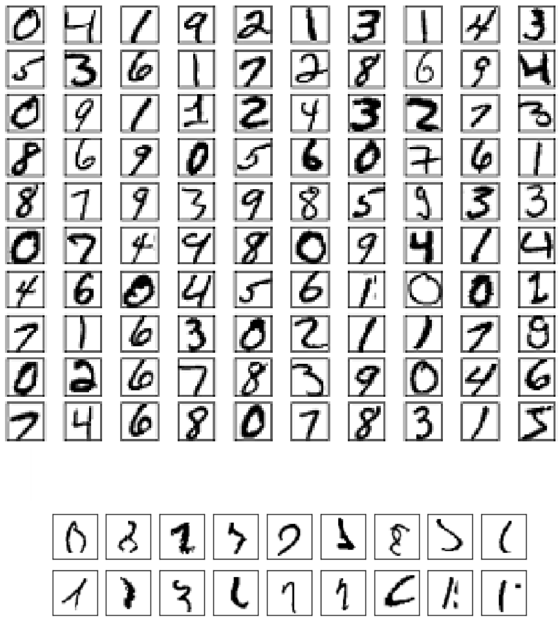
CNNs – Basic Principles

- **Convolutional Neural Networks (CNNs/ConvNets) implement a connectivity pattner between neurons inspired by the animal visual cortex and use several types of layers (convolution, pooling)**
- **CNN key principles are local receptive fields, shared weights, and pooling (or down/sub-sampling)**
- **CNNs are optimized to take advantage of the spatial structure of the data**

- Simple application example
  - MNIST database written characters
  - Use CNN architecture with different layers
  - Goal: automatic classification of characters



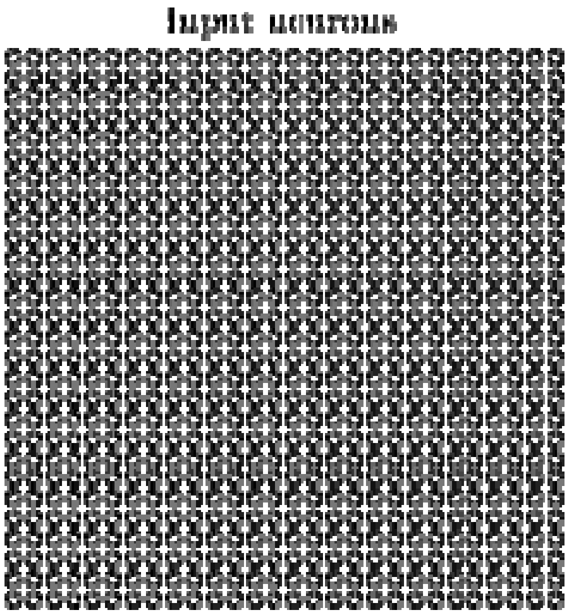
[10] A. Rosebrock



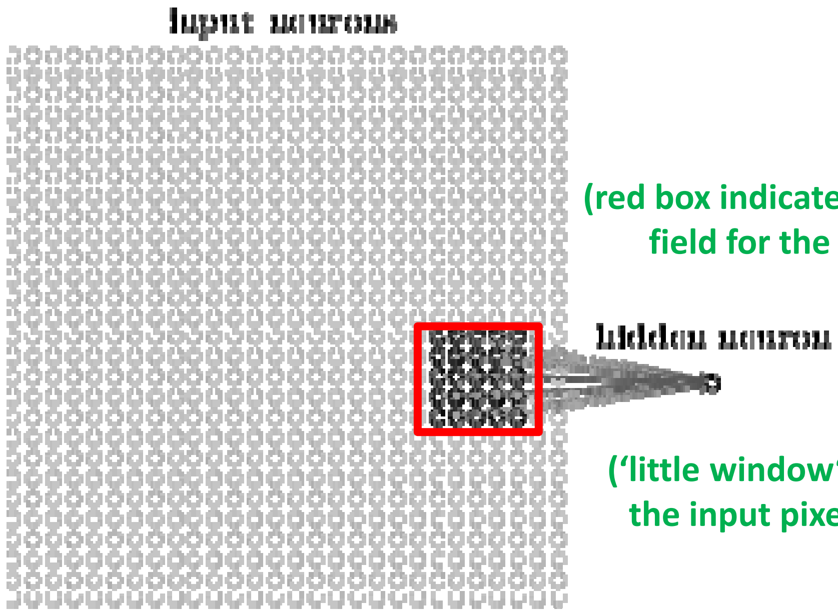
[9] M. Nielsen

CNNs – Principle Local Receptive Fields

- MNIST dataset example
  - 28 * 28 pixels modeled as square of neurons in a convolutional net
  - Values correspond to the 28 * 28 pixel intensities as inputs



(28 * 28 pixel image)



(red box indicate the local receptive field for the hidden neuron)

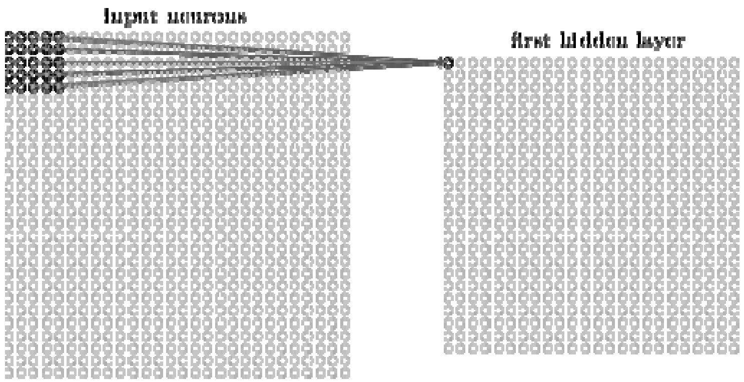
('little window' on the input pixels)

(5 * 5 local connectivity)

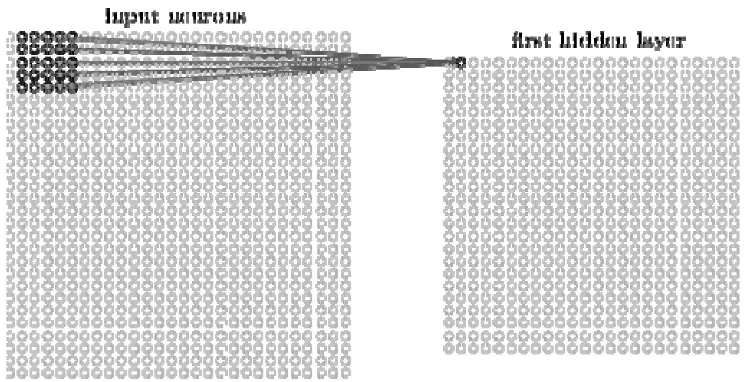
[9] M. Nielsen

CNNs – Principle Local Receptive Fields & Sliding

- MNIST database example
  - Apply stride length = 1
  - Different configurations possible and depends on application goals
  - Creates ‘feature map‘ of 24 * 24 neurons (hidden layer)



(28 * 28 pixel image) **(24 * 24 feature map)**



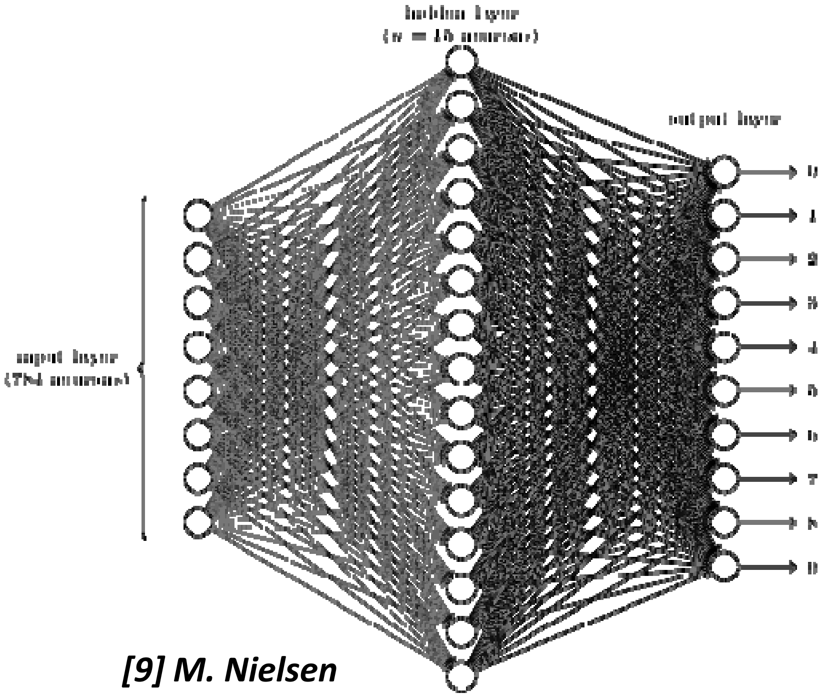
(28 * 28 pixel image) **(24 * 24 feature map)**

[9] M. Nielsen

CNNs –Example with an ANN with risk of Overfitting

- MNIST database example
  - CNN: e.g. 20 feature maps with 5 * 5 (+bias) = 520 weights to learn
  - Apply ANN that is fully connected between neurons
  - ANN: fully connected first layer with 28 * 28 = 784 input neurons
  - ANN: e.g. 15 hidden neurons with 784 * 15 = 11760 weights to learn

(eventually lead to overfitting and much computing time)

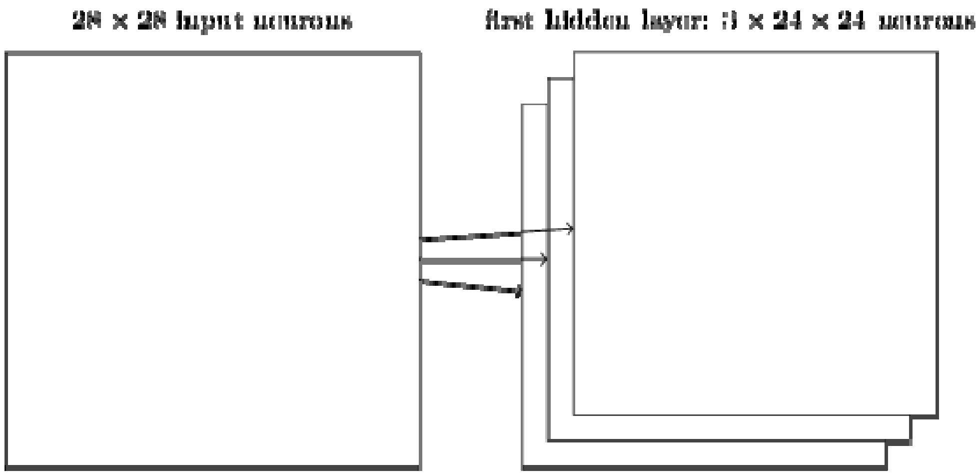


[9] M. Nielsen

- **Overfitting refers to fit the data too well – more than is warranted – thus may misguide the learning – rather memorizing than realy learning**
- **A good model must have low training error (E_{in}) and low generalization error (E_{out})**
- **Model overfitting is if a model fits the data too well (E_{in}) with a poorer generalization error (E_{out}) than another model with a higher training error (E_{in})**
- **The two general approaches to prevent overfitting are (1) regularization and (2) validation**

CNNs – Principle Shared Weights & Feature Maps

- Approach
  - CNNs use same shared weights for each of the 24 * 24 hidden neurons
  - Goals: significant reduction of number of parameters (prevent overfitting)
  - Example: 5 * 5 receptive field → 25 shared weights + shared bias
- Feature Map
  - Detects one local feature
  - E.g. 3: each feature map is defined by a set of 5 * 5 shared weights and a single shared bias leading to 24 * 24
  - Goal: The network can now detect 3 different kind of features **(many more in practice)**

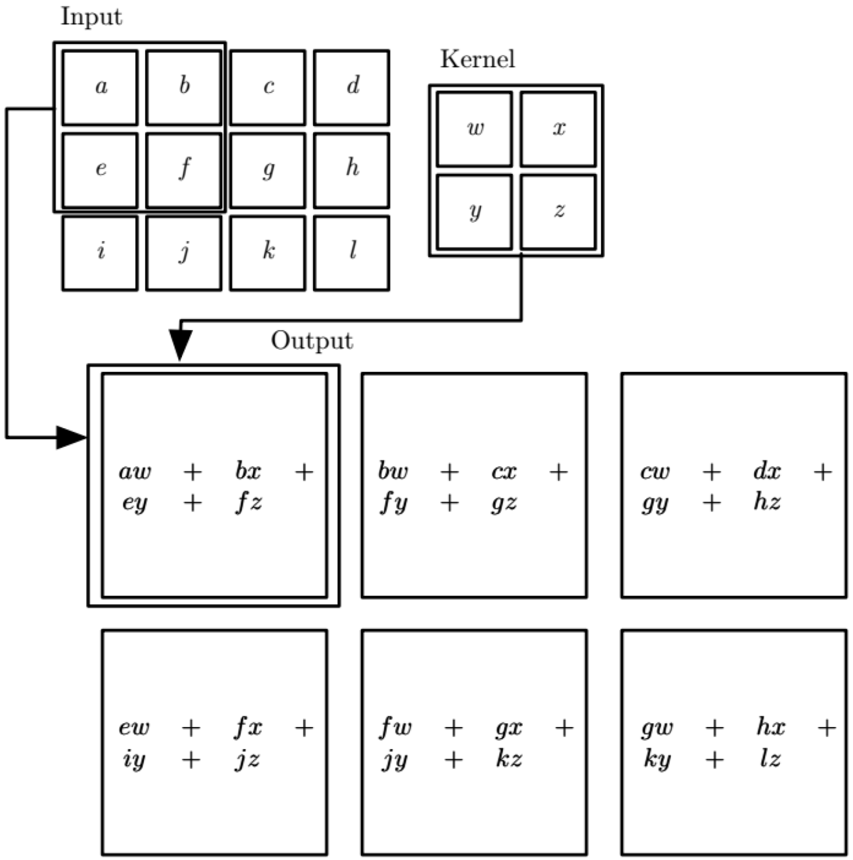


(shared weights are also known to define a kernel or filter)

  - Benefit: learned feature being detectable across the entire image

[9] M. Nielsen

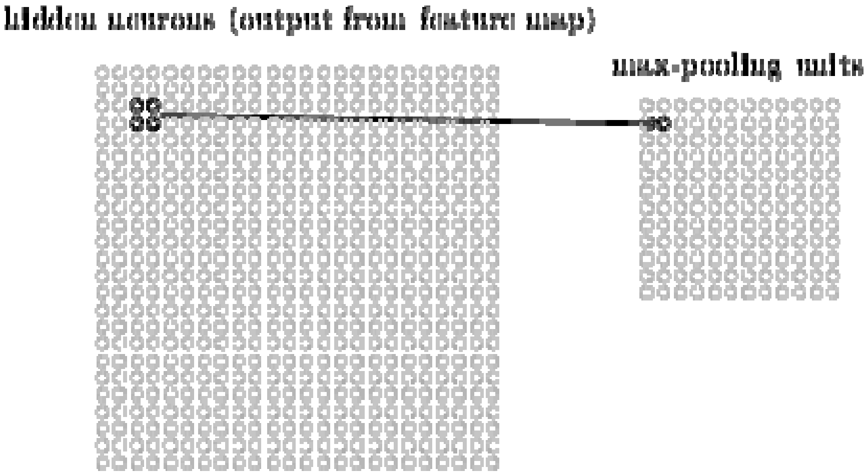
CNNs – Principle of Convolution & Example

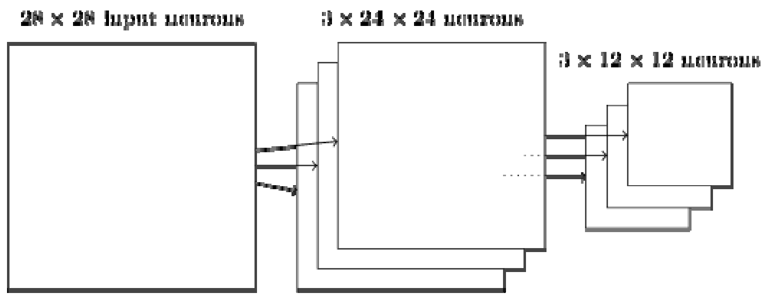


- Example: 3x4 input matrix processed by a 2x2 kernel with stride=1 that calculates the sum of its content
- Valid convolution does not exceed the input‘s boundary
- Same convolution adds a so called ‘padding‘ to maintain the input‘s dimension for each convolutional layer

CNNs – Principle of Pooling

- ‘Downsampling‘ Approach
  - Usually applied directly after convolutional layers
  - Idea is to simplify the information in the output from the convolution
  - Take each feature map output from the convolutional layer and generate a condensed feature map
  - E.g. Pooling with 2 * 2 neurons using ‘max-pooling‘
  - Max-Pooling outputs the maximum activation in the 2 * 2 region

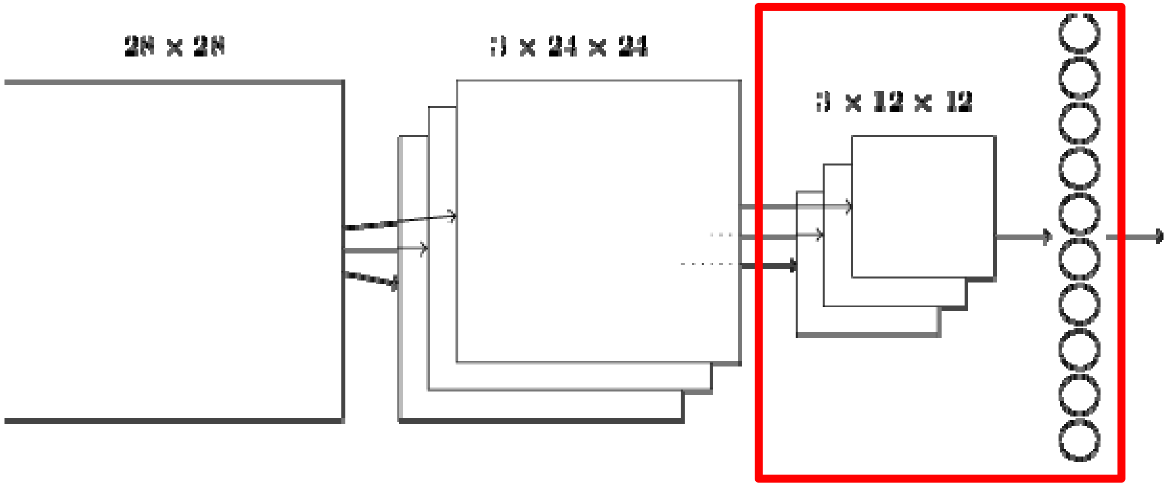


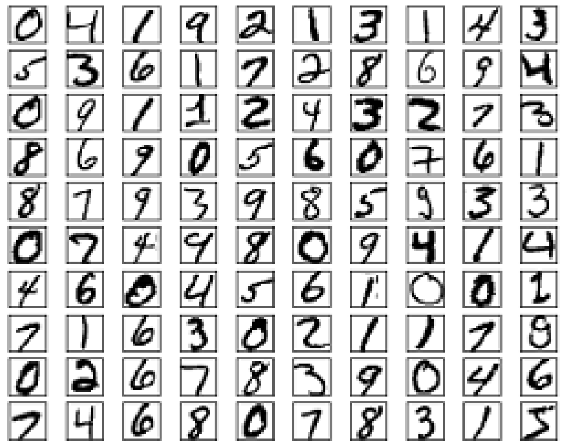


[9] M. Nielsen

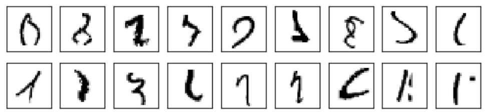
CNN – Application Example MNIST

- MNIST database example
  - Full CNN with the addition of output neurons per class of digits
  - Apply 'fully connected layer': layer connects every neuron from the max-pooling outcome layer to every neuron of the 10 out neurons
  - Train with backpropagation algorithm (gradient descent), only small modifications for new layers





  - Approach works, except for some bad training and test examples



(another indicator that even with cutting edge technology machine learning never achieves 100% performance)

[9] M. Nielsen

CNN in Comparison with other Previous Learning Models

- Application Example
  - MNIST Dataset
- Perceptron
  - Simple model
- Multilayer Perceptron
  - ANN model
    with backpropagation
- Deep Learning
  - CNN model
    learning features

[17] Neural Network 3D Simulation

> ➢ **Lecture 7 will provide more CNN application examples in cloud computing environments**

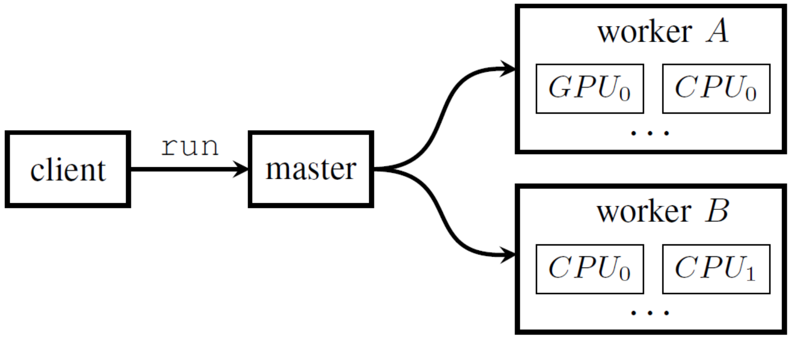
Keras with Tensorflow Backend – GPU Support

- **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**
- **The key idea behind the Keras tool is to enable faster experimentation with deep networks**
- **Created deep learning models run seamlessly on CPU and GPU via low-level frameworks**

[6] Keras Python Deep Learning Library

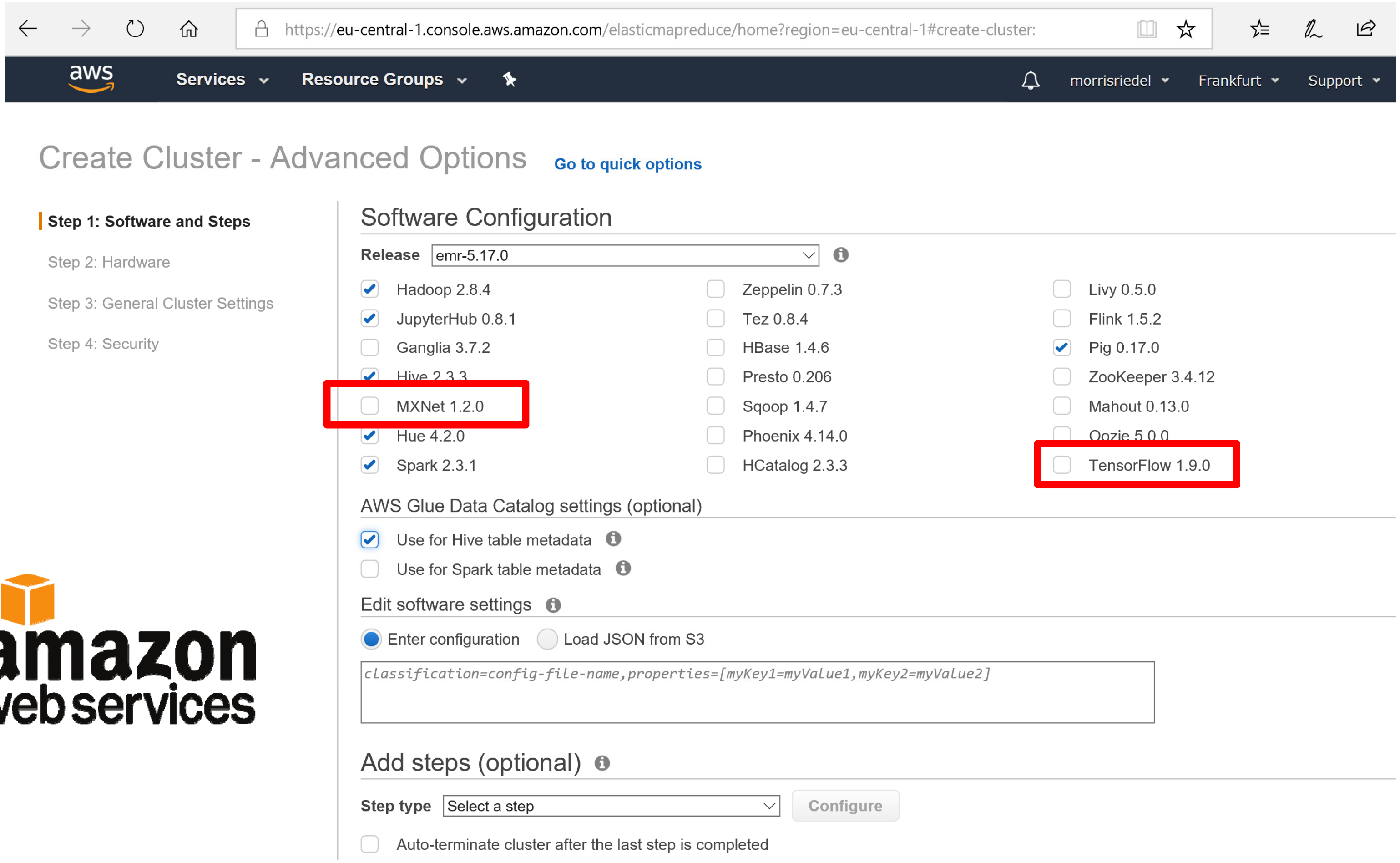
- **Tensorflow is an open source library for deep learning models using a flow graph approach**
- **Tensorflow nodes model mathematical operations and graph edges between the nodes are so-called tensors (also known as multi-dimensional arrays)**
- **The Tensorflow tool supports the use of CPUs and GPUs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

[14] Tensorflow Deep Learning Framework



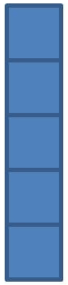
[15] A Tour of Tensorflow

AWS EMR Example – Cloud Support for Deep Learning



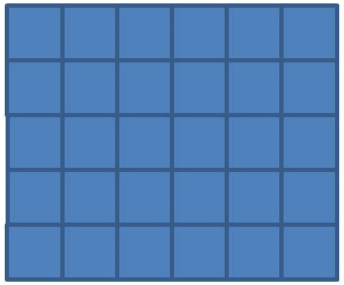
➤ **Lecture 7 will provide more CNN application examples in cloud computing environments**

Deep Learning in Clouds – What are Tensors?



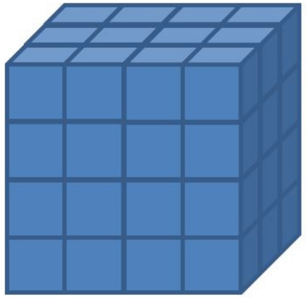
(one dimensional tensor)

(vector of dimension [5])



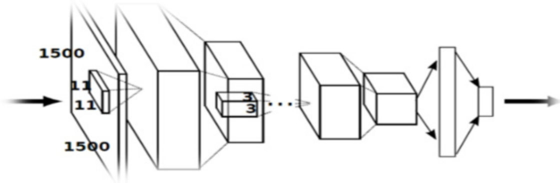
(two dimensional tensor)

(matrix of dimensions [5,6])



(three dimensional tensor)

(tensor of dimension [4,4,3])



(deep learning network
use tensors much and
NumPy is good to work
in Python scripts with these)

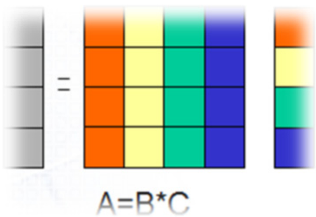
[13] Big Data Tips, What is a Tensor?

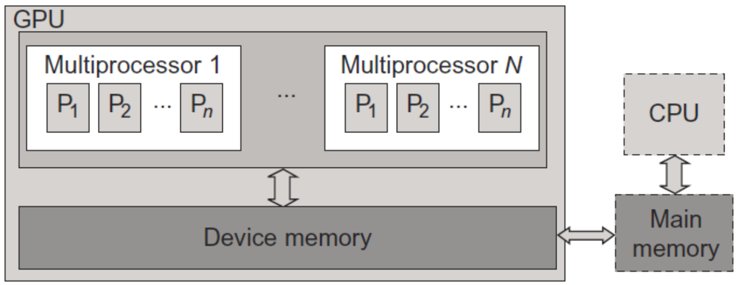
- A Tensor is nothing else than a multi-dimensional array often used in scientific & engineering environments
- Tensors are best understood when comparing it with vectors or matrices and their dimensions
- Those tensors ‘flow‘ through the deep learning network during the optimization / learning & inference process

GPU Acceleration

- **CPU acceleration means that GPUs accelerate computing due to a massive parallelism with thousands of threads compared to only a few threads used by conventional CPUs**
- **GPUs are designed to compute large numbers of floating point operations in parallel**

- GPU accelerator architecture example (e.g. NVIDEA card)
  - GPUs can have 128 cores on one single GPU chip
  - Each core can work with eight threads of instructions
  - GPU is able to concurrently execute 128 * 8 = 1024 threads
  - Interaction and thus major (bandwidth) bottleneck between CPU and GPU is via memory interactions
  - E.g. applications that use matrix – vector multiplication

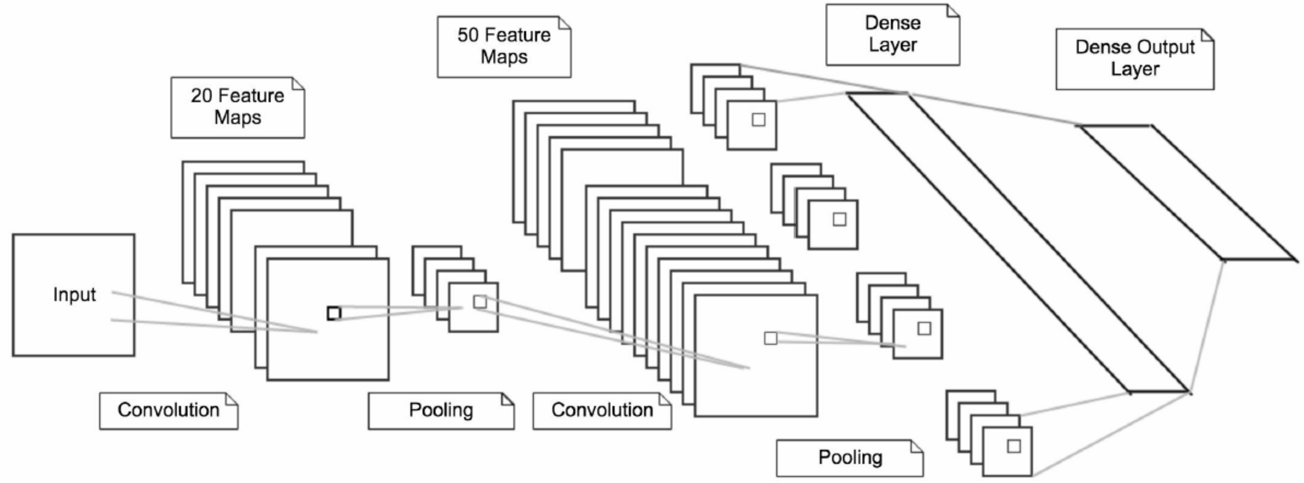




[8] Distributed & Cloud Computing Book

MNIST Dataset – CNN Model

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Flatten
from keras.utils import np_utils
from keras import backend as K
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.optimizers import SGD, RMSprop, Adam

# model
class CNN:
  @staticmethod
  def build(input_shape, classes):
    model = Sequential()
    model.add(Convolution2D(20, kernel_size=5, padding="same", input_shape=input_shape))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
    model.add(Convolution2D(50, kernel_size=5, border_mode="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
    model.add(Flatten())
    model.add(Dense(500))
    model.add(Activation("relu"))
    model.add(Dense(classes))
    model.add(Activation("softmax"
    return model
```



[11] A. Gulli et al.

MNIST Dataset – CNN Python Script

```
# parameters
NB_CLASSES = 10
NB_EPOCH = 20
BATCH_SIZE = 128
VERBOSE = 1
OPTIMIZER = 'Adam'
VALIDATION_SPLIT = 0.2
IMG_ROWS, IMG_COLS = 28, 28
INPUT_SHAPE = (1, IMG_ROWS, IMG_COLS)

# dataset 28 x 28 pixels
(X_train, y_train), (X_test, y_test) = mnist.load_data()
K.set_image_dim_ordering("th")
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

# input convnet
X_train = X_train[:, np.newaxis, :, :]
X_test = X_test[:, np.newaxis, :, :]

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

# convert vectors to binary matrices of classes
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# Simple CNN model
model = CNN.build(input_shape=INPUT_SHAPE, classes=NB_CLASSES)

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

# Fit the model
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE, validation_split=VALIDATION_SPLIT)

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

- **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)**

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

[11] A. Gulli et al.

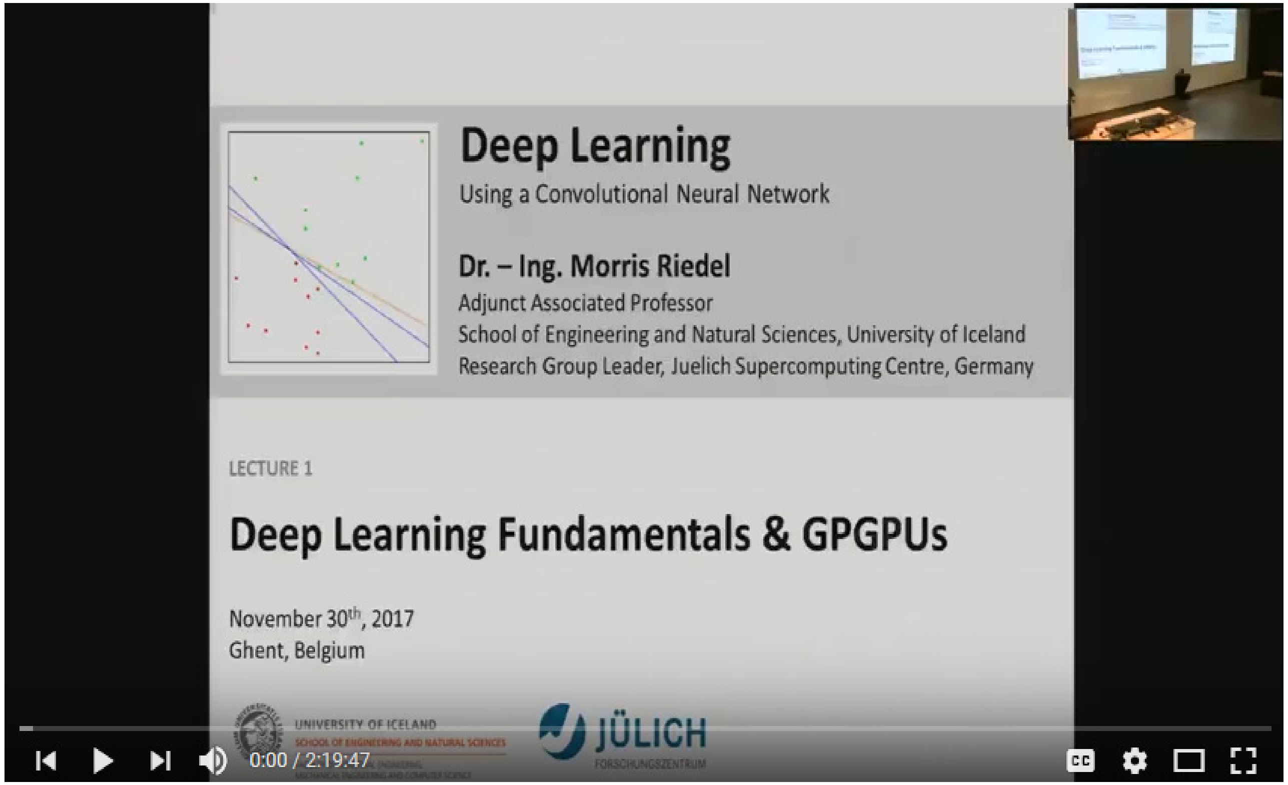
MNIST Dataset – CNN Model – Output

```
[vsc42544@gligar01 deeplearning]$ head KERAS_MNIST_CNN.o1179880
60000 train samples
10000 test samples
Train on 48000 samples, validate on 12000 samples
Epoch 1/20

  128/48000 [..............................] - ETA: 10:06 - loss: 2.2997 - acc: 0.1250
  256/48000 [..............................] - ETA: 7:46 - loss: 2.2578 - acc: 0.1992
  384/48000 [..............................] - ETA: 6:58 - loss: 2.2127 - acc: 0.2083
  512/48000 [..............................] - ETA: 6:35 - loss: 2.1632 - acc: 0.2598
  640/48000 [..............................] - ETA: 6:20 - loss: 2.0934 - acc: 0.3234
```

```
[vsc42544@gligar01 deeplearning]$ tail KERAS_MNIST_CNN.o1179880
 9824/10000 [============================>.] - ETA: 0s
 9856/10000 [============================>.] - ETA: 0s
 9888/10000 [============================>.] - ETA: 0s
 9920/10000 [============================>.] - ETA: 0s
 9952/10000 [============================>.] - ETA: 0s
 9984/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - 41s 4ms/step
Test score: 0.0483192791523
Test accuracy: 0.99
```

(97,7% accuracy was best for ANNs,
this is another improvement by using different type of layers)

[YouTube Lectures] More Deep Learning Fundamentals

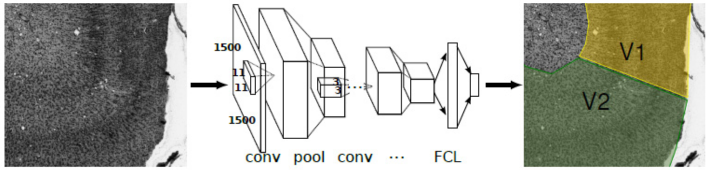


[4] Morris Riedel, 'Deep Learning - Using a Convolutional Neural Network', Invited YouTube Lecture, six lectures, University of Ghent, 2017

- **Note that this course is not a full deep learning course but rather focusses on Big Data & Clouds**

Deep Learning Architectures – Revisited

- Deep Neural Network (DNN)
  - ‘Shallow ANN‘ approach with many hidden layers between input/output
- Convolutional Neural Network (CNN, sometimes ConvNet)
  - Connectivity pattern between neurons is like animal visual cortex



- Deep Belief Network (DBN)
  - Composed of mult iple layers of variables; only connections between layers
- Recurrent Neural Network (RNN)
  - ‘ANN‘ but connections form a directed cycle; state and temporal behaviour

- **Deep Learning architectures can be classified into Deep Neural Networks, Convolutional Neural Networks, Deep Belief Networks, and Recurrent Neural Networks all with unique characteristica**
- **Deep Learning needs ‘big data‘ to work well & for high accuracy – works not well on sparse data**

RNNs for Sequence Models

- **Sequence models enable various sequence predictions that are inherent different to other more traditional predictive modeling techniques or supervised learning approaches**
- **In contrast to mathematical sets often used, the ‘sequence‘ model imposes an explicit order on the input/output data that needs to be preserved in training and/or inference**
- **Sequence models are driven by application goals and include sequence prediction, sequence classification, sequence generation, and sequence-to-sequence prediction**

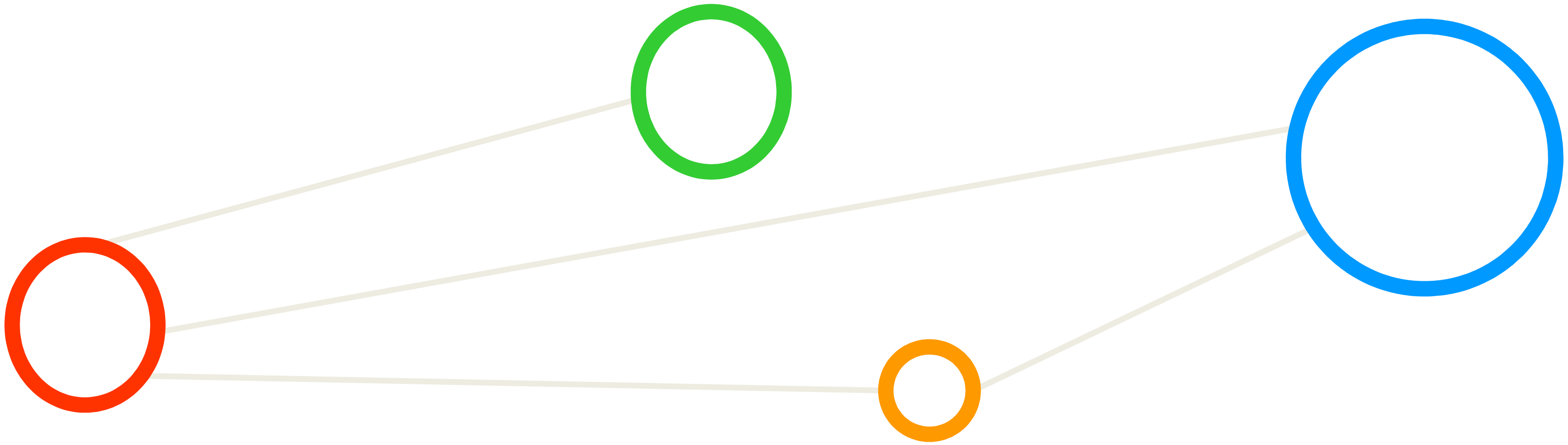
- Model Categorization
  - Based on different inputs/outputs to/from the sequence models
- Practical ‘standard dataset‘ perspective
  - Often the order of samples is not important
  - Training/testing datasets and their samples have often no explicit order (i.e. ‘sets‘)
- Practical ‘sequence dataset‘ perspective
  - Order of samples is important: sequence learning/inference needs order

> **Lecture 7 will provide selected application examples of sequence models using RNNs in Clouds**

[Video] Deep Learning in Gaming Applications

[18] YouTube Video, DQN Breakout

Lecture Bibliography



Lecture Bibliography (1)

- [1] Mining of Massive Datasets,
  Online: http://infolab.stanford.edu/~ullman/mmds/book.pdf
- [2] Apache Hadoop Web page,
  Online: http://hadoop.apache.org/
- [3] AWS Marketplace,
  Online: https://aws.amazon.com/marketplace/
- [4] 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
- [5] YouTube Video, 'Neural Networks, A Simple Explanation',
  Online: http://www.youtube.com/watch?v=gcK_5x2KsLA
- [6] Keras Python Deep Learning Library,
  Online: https://keras.io/
- [7] H. Lee et al., 'Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations', Proceedings of the 26th annual International Conference on Machine Learning (ICML), ACM, 2009
- [8] K. Hwang, G. C. Fox, J. J. Dongarra, 'Distributed and Cloud Computing', Book,
  Online: http://store.elsevier.com/product.jsp?locale=en_EU&isbn=9780128002049
- [9] M. Nielsen, 'Neural Networks and Deep Learning',
  Online: http://neuralnetworksanddeeplearning.com/
- [10] A. Rosebrock, 'Get off the deep learning bandwagon and get some perspective',
  Online: http://www.pyimagesearch.com/2014/06/09/get-deep-learning-bandwagon-get-perspective/

Lecture Bibliography (2)

- [11] 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
- [12] D. Kingma and Jimmy Ba, ‘Adam: A Method for Stochastic Optimization‘, Online: https://arxiv.org/abs/1412.6980
- [13] Big Data Tips, ‘What is a Tensor?‘, Online: http://www.big-data.tips/what-is-a-tensor
- [14] Tensorflow Deep Learning Framework, Online: https://www.tensorflow.org/
- [15] A Tour of Tensorflow, Online: https://arxiv.org/pdf/1610.01178.pdf
- [16] Big Data Tips, ‘ReLU Neural Network‘, Online: http://www.big-data.tips/relu-neural-network
- [17] YouTube Video, ‘Neural Network 3D Simulation‘, Online: https://www.youtube.com/watch?v=3JQ3hYko51Y
- [18] YouTube Video, ‘DQN Breakout‘, Online: https://www.youtube.com/watch?v=TmPfTpjtdgg
- [19] Big Data Tips, ‘Gradient Descent‘, Online: http://www.big-data.tips/gradient-descent
- [20] Our World in Data Web Page, Online: https://ourworldindata.org/internet

Lecture Bibliography (3)

- [21] J. Dean et al., ‘Large scale deep learning’, Keynote GPU Technical Conference, 2015
- [22] ImageNet Web page,
  Online: http://image-net.org


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