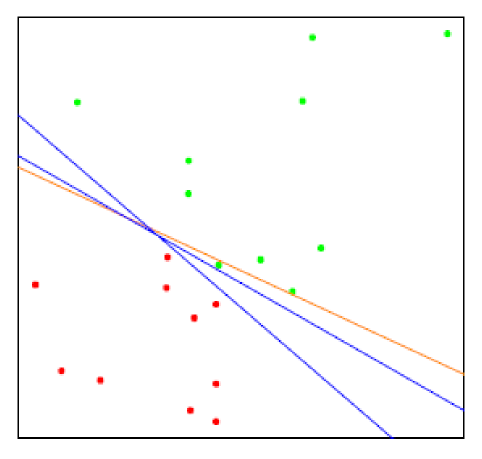


Deep Learning

Using a Convolutional Neural Network

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

LECTURE 1

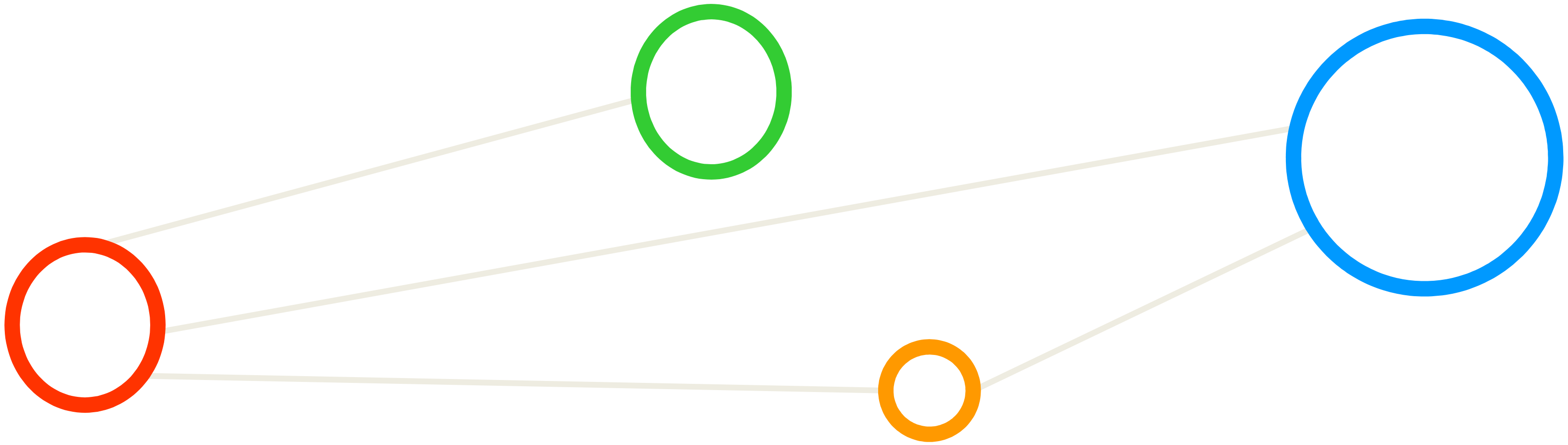
Deep Learning Fundamentals & GPGPUs

November 30th, 2017
Ghent, Belgium

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

JÜLICH
FORSCHUNGSZENTRUM

Outline



Outline of the Course

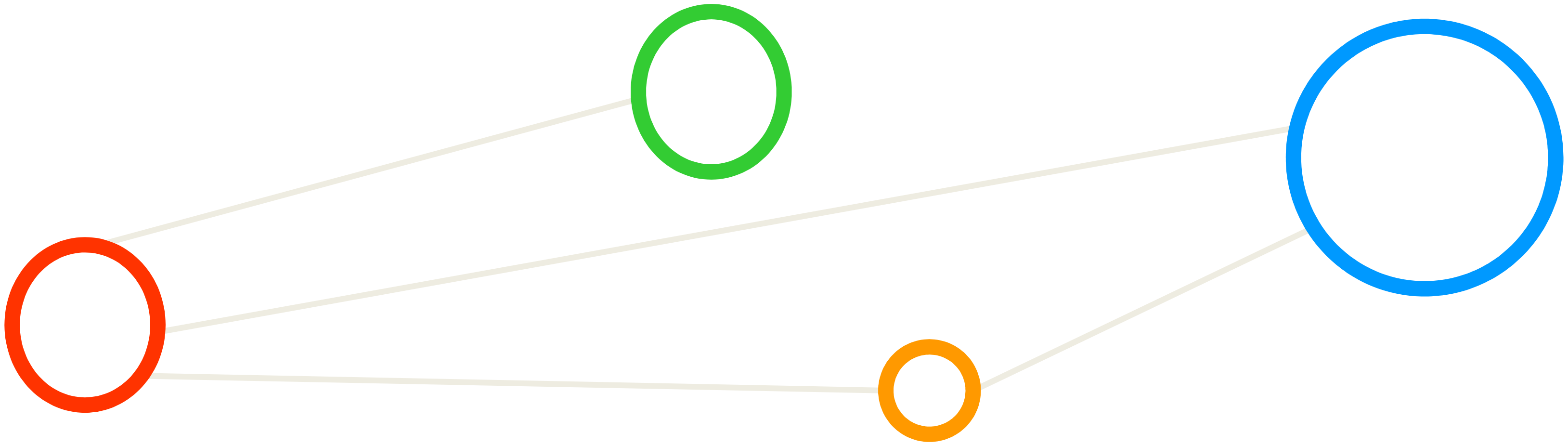
1. Deep Learning Fundamentals & GPGPUs
2. Convolutional Neural Networks & Tools
3. Convolutional Neural Network Applications
4. Convolutional Neural Network Challenges
5. Transfer Learning Technique
6. Other Deep Learning Models & Summary

Outline

- Deep Learning Foundations
  - Biological Inspiration & Perceptron Limits
  - Artificial Neural Networks & Backpropagation
  - Application Examples in Science & Industry
  - Deep Learning Properties & Feature Learning
  - Parallel Computing Methods & Architectures

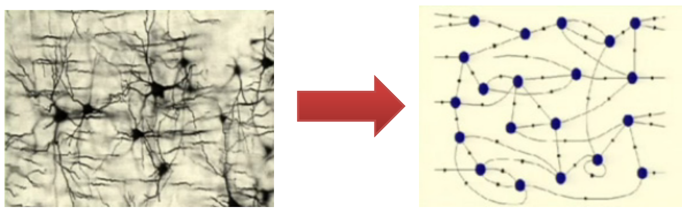
- GPGPUs & Tools
  - Terminology & Many-core Architecture
  - GPU Acceleration
  - NVidia & CUDA Examples
  - OpenCL Programming Models
  - Usage Models & Applications

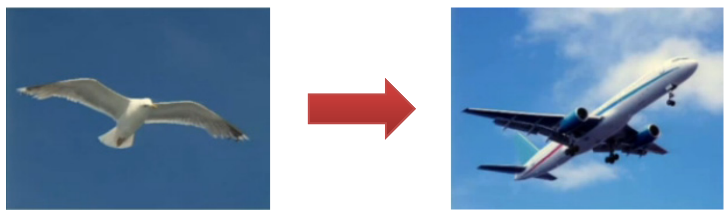
Deep Learning Foundations



Learning Models derived from Biological Inspiration

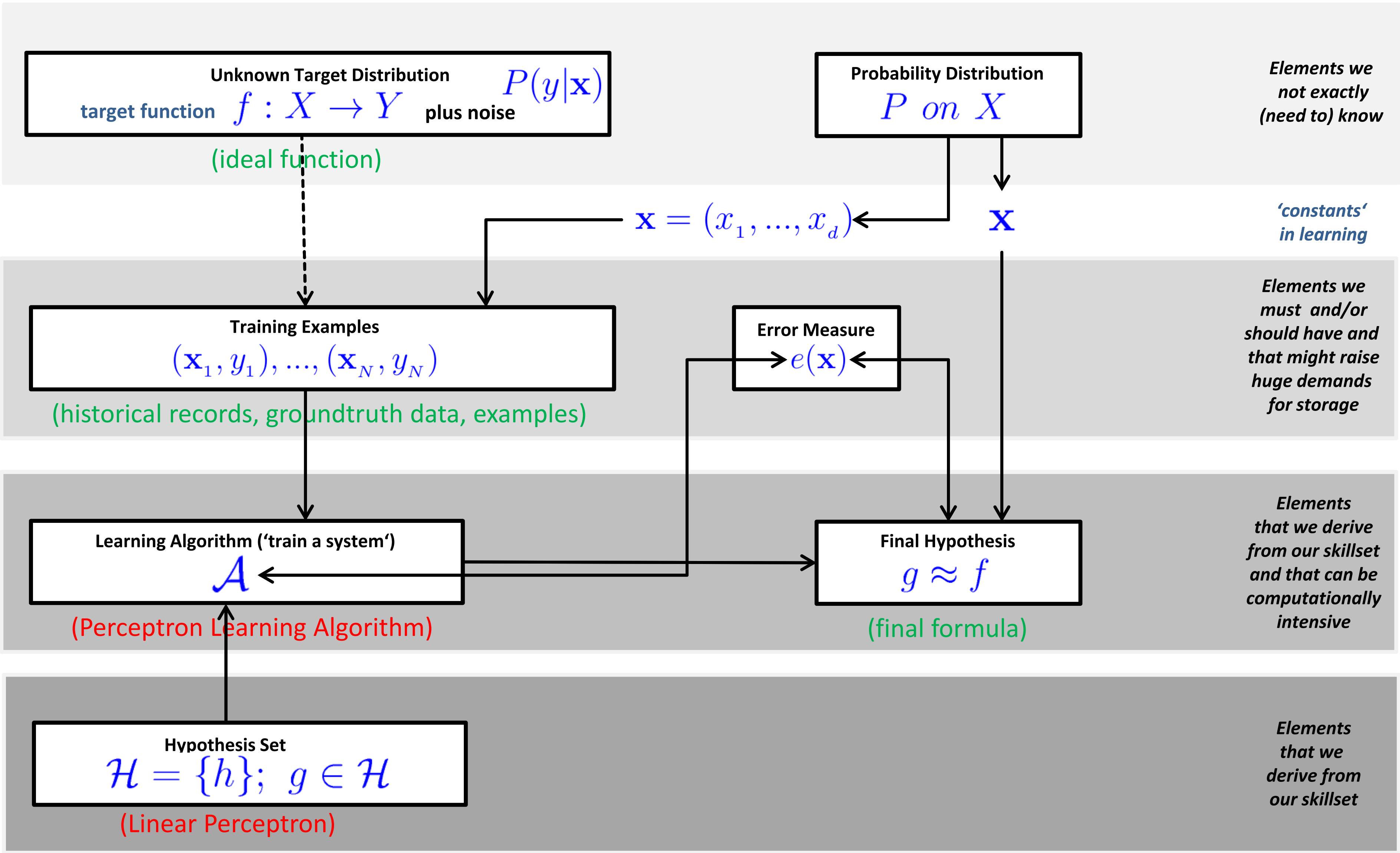
- Biological Inspiration (cf. Machine Learning Tutorial last week)
  - Humans learn (a biological function) → machines can learn
  - Means we are interested in ‘replicating’ the ‘biological function’
- Approach: Replicating the ‘biological structure’
  - Neurons connected to synapses (large number)
  - 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‘





[1] Neural Networks

Solution Tools: Artificial Neural Networks Learning Model



Perceptron Learning Algorithm – Revisited

- When: If we believe there is a linear pattern to be detected
  - Assumption: Linearly seperable data (lets the algorithm converge)
  - (cf. Machine learning tutorial last week)

Training Examples

$(x_1, y_1), \ldots, (x_N, y_N)$

(existing dataset already being labelled as +1/-1)

Learning Algorithm ('train a system')

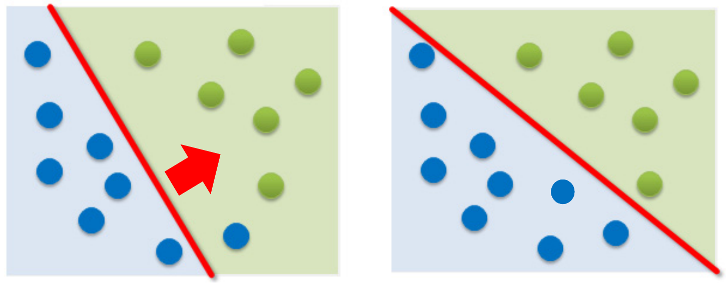
$\mathcal{A}$

(Perceptron Learning Algorithm)

Hypothesis Set

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

(Perceptron model)



[4] Rosenblatt, 1958

- $$h(\mathbf{x}) = sign\left(\left(\sum_{i=0}^{d} w_i x_i\right)\right);\ x_0 = 1$$

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right);\ w_0 = -threshold$$

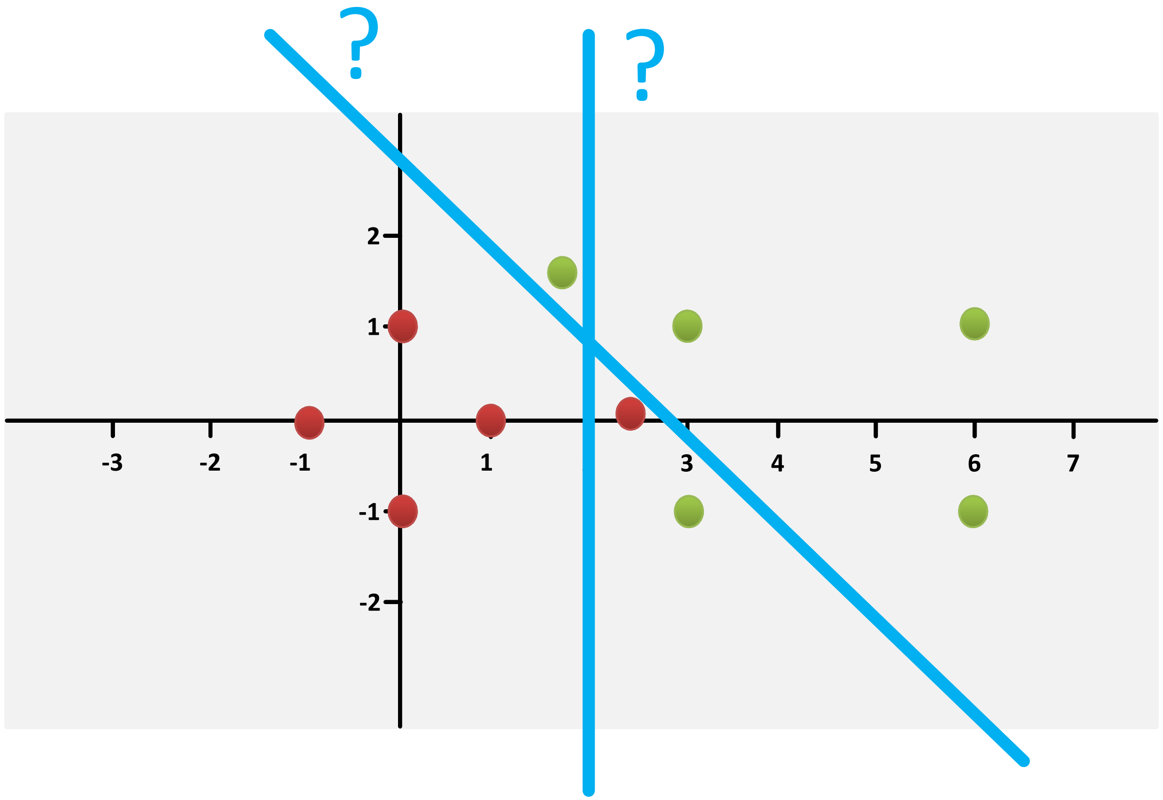
$$h(\mathbf{x}) = sign(\mathbf{w}^T \mathbf{x})$$ **(vector notation, using transpose)**

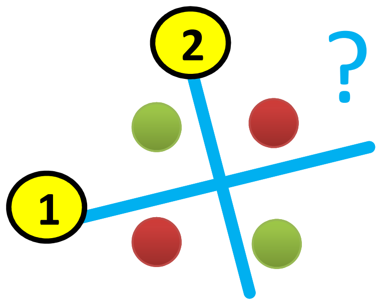
(transpose = reflecting elements along main diagonal)

Exercises

Practice: Non-linearly Seperable Data

- More often in practice, requires a 'soft threshold'
  - 'soft-threshold' means allowing 'some errors' being 'overall' better

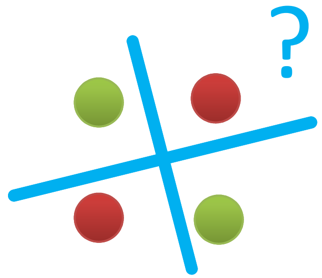




(known also as XOR problem)

Simple Application Example: Limitations of Perceptrons

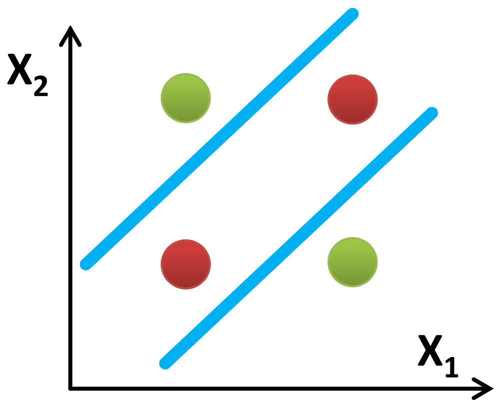
- Simple perceptrons fail: 'not linearly seperable'



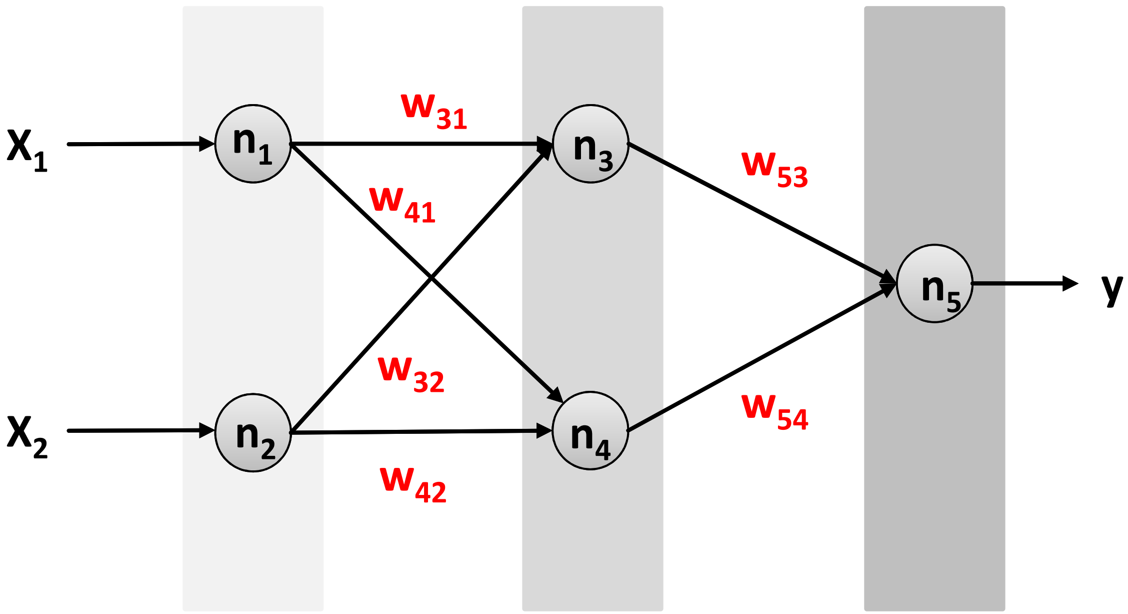
(Idea: instances can be classified using two lines at once to model XOR)

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

Labelled Data Table



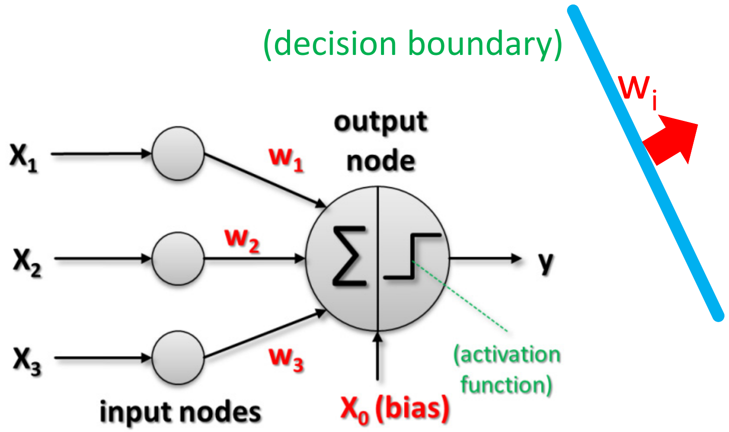
Decision Boundary



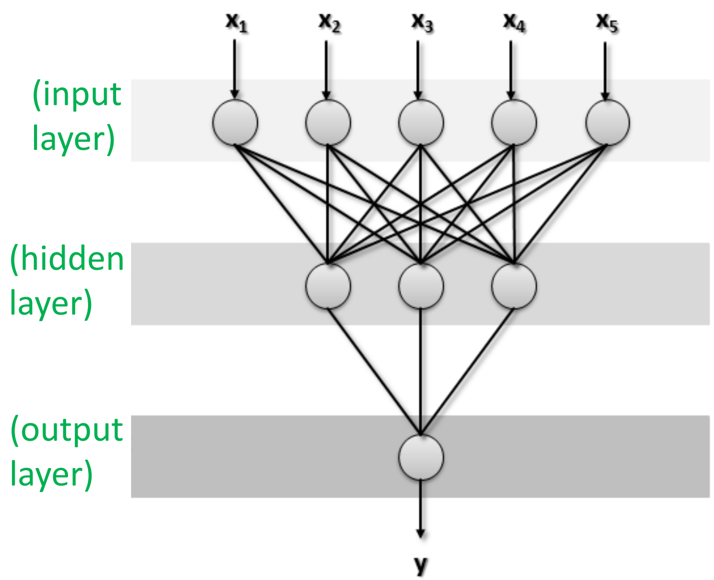
Two-Layer, feed-forward Artificial Neural Network topology

Multi Layer Perceptrons – Artificial Neural Networks

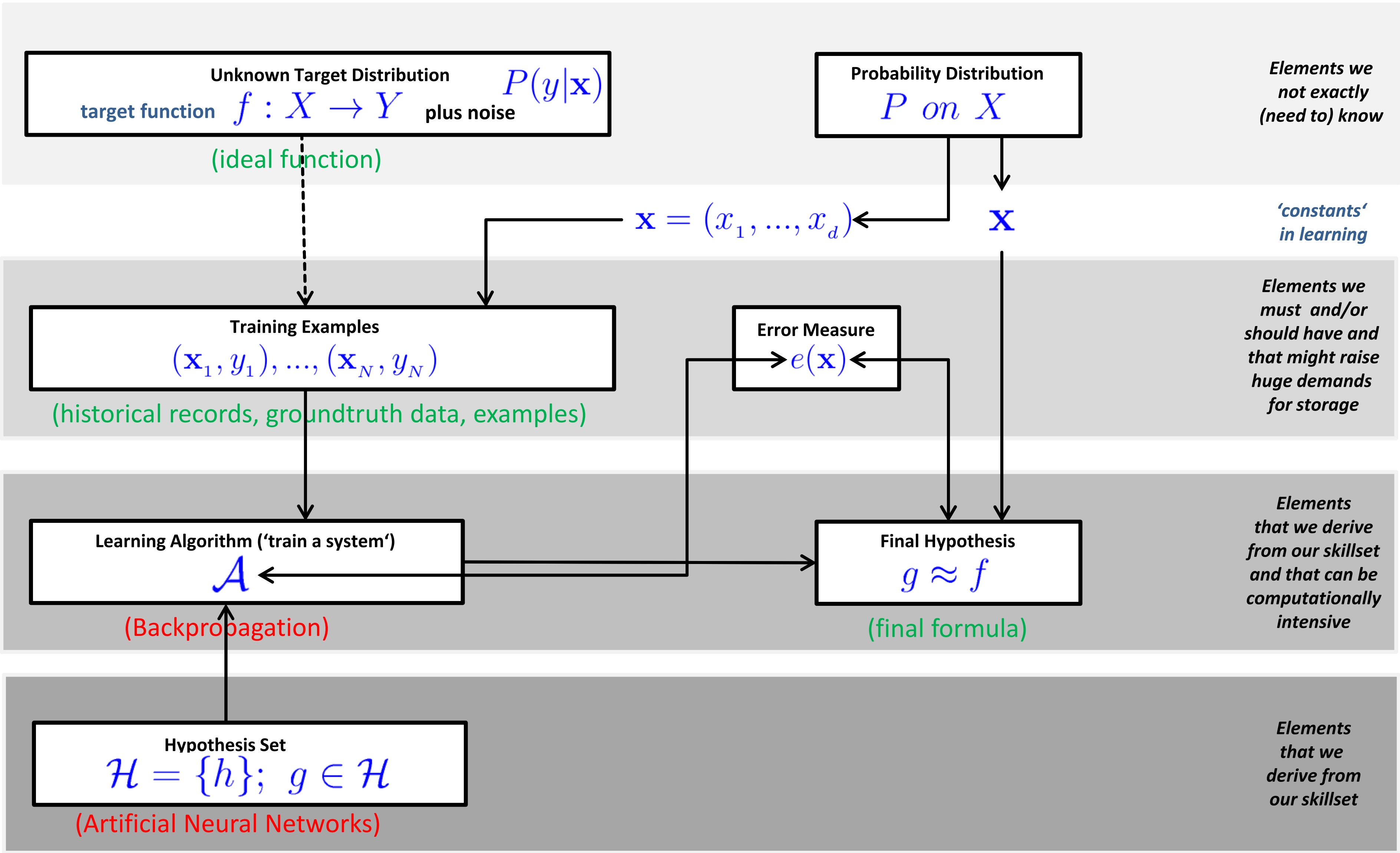
- Key Building Block
  - Perceptron learning model
  - Simplest linear learning model
  - Linearity in learned weights w_i
  - One decision boundary
- Artificial Neural Networks (ANNs)
  - Creating more complex structures
  - Enable the modelling of more complex relationships in the datasets
  - May contain several intermediary layers
  - E.g. 2-4 hidden layers with hidden nodes
  - Use of activation function that can produce output values that are nonlinear in their input parameters



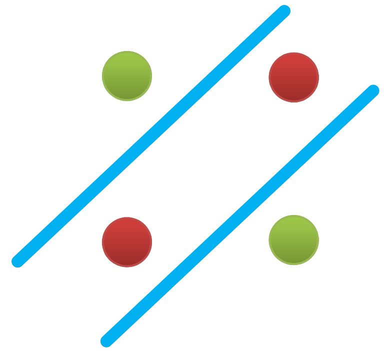
(perceptron model)

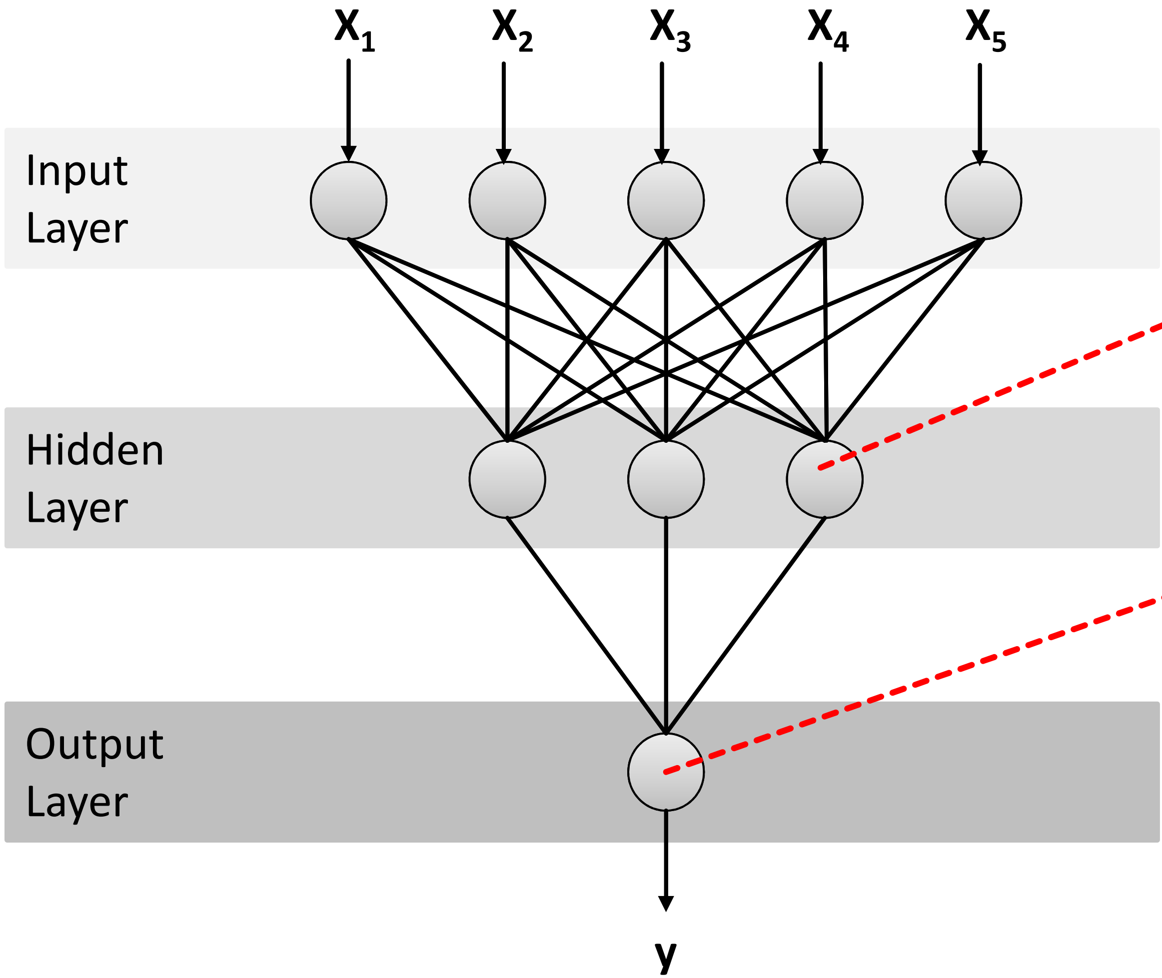


Solution Tools: Artificial Neural Networks Learning Model



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

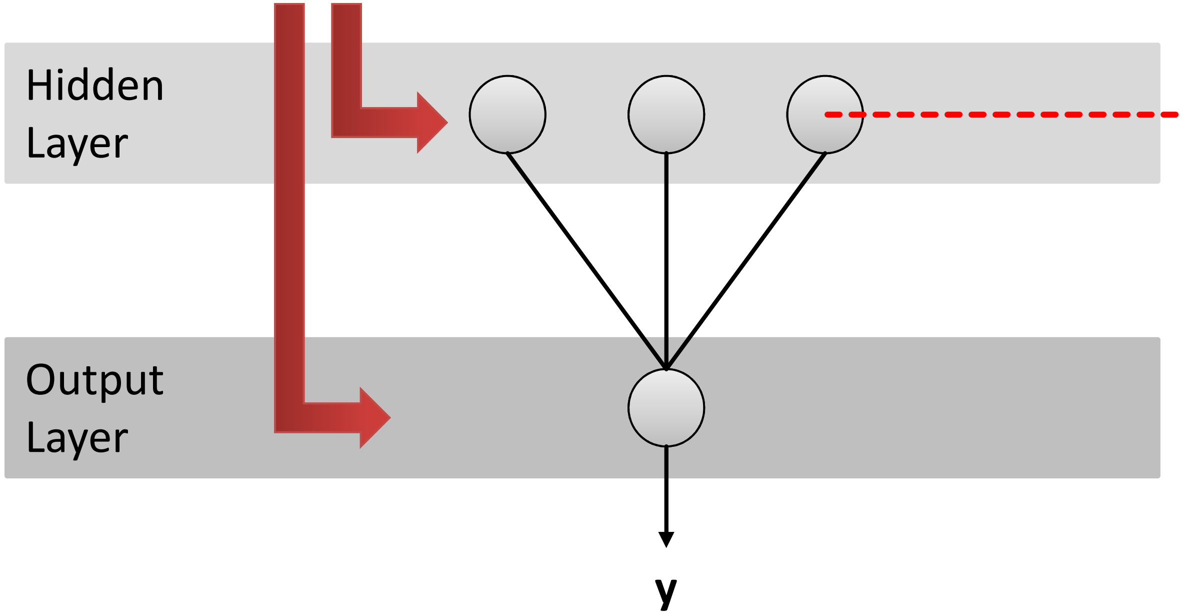
- Determine a set of weights **w** that
  'minimize the total sum of squared errors':

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

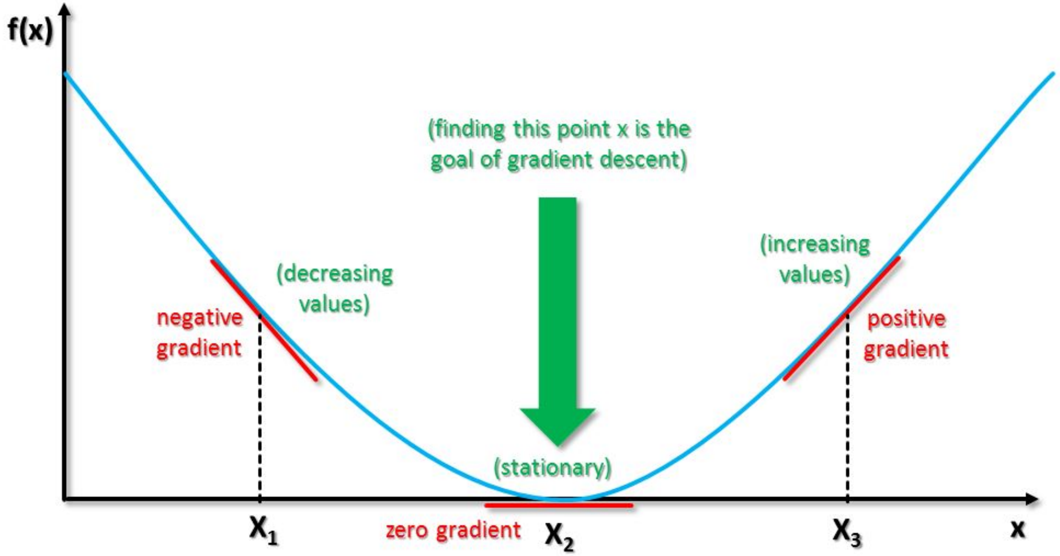
$$E\,(w) = \frac{1}{2}\sum_{i=1}^{N}\boxed{(y_i - \acute{y}_i)}^2$$



- Error term, associated with each hidden node

- Error function is quadratic in its parameters and a global minimum can be easily found
- Other objective / loss functions possible, e.g. categorical cross-entropy

Gradient Descent Method (1)



(minimization: <u>substract</u> gradient term because we move towards local minima)

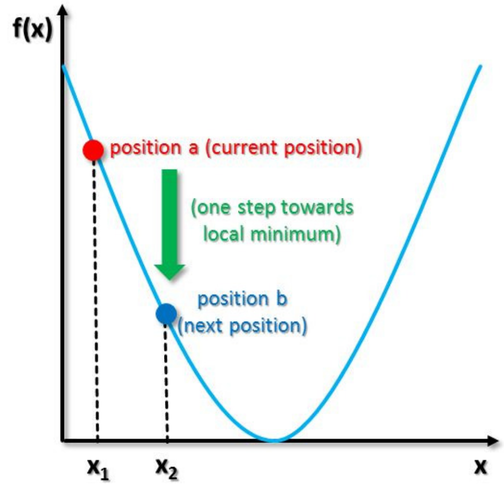
(the derivative of f with respect to a)

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

(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)



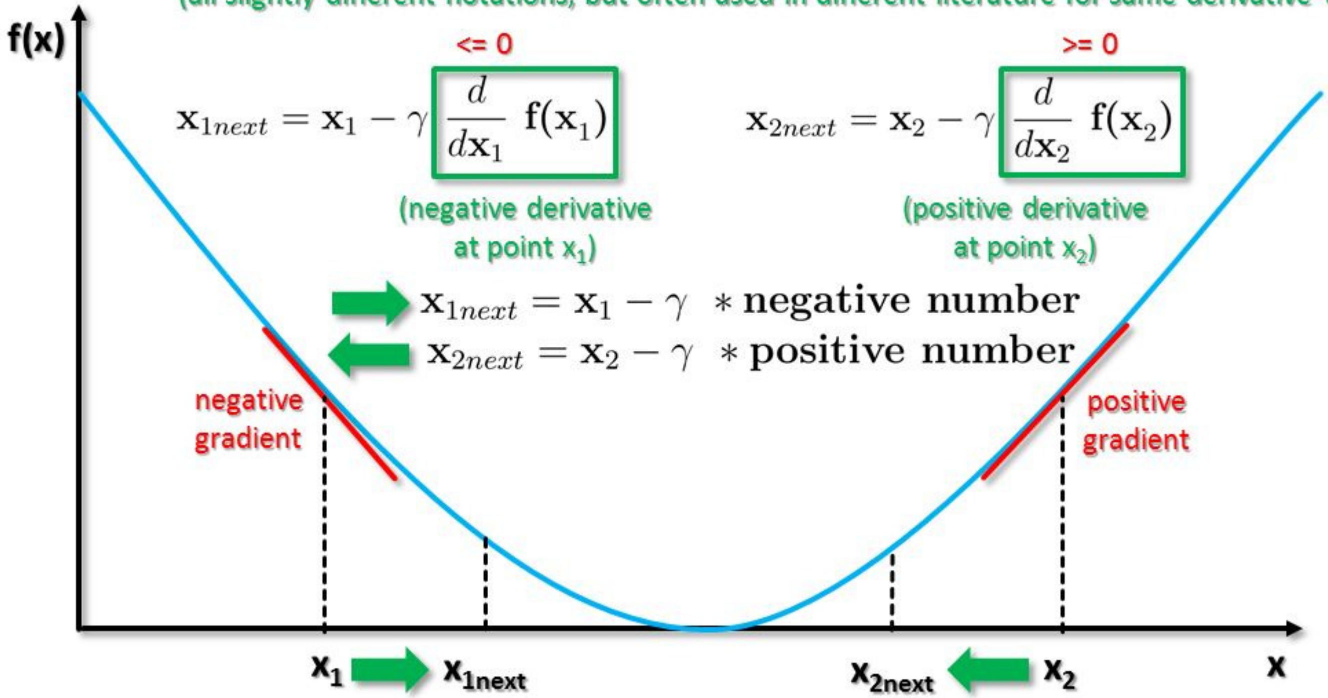
[6] Big Data Tips, Gradient Descent

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)



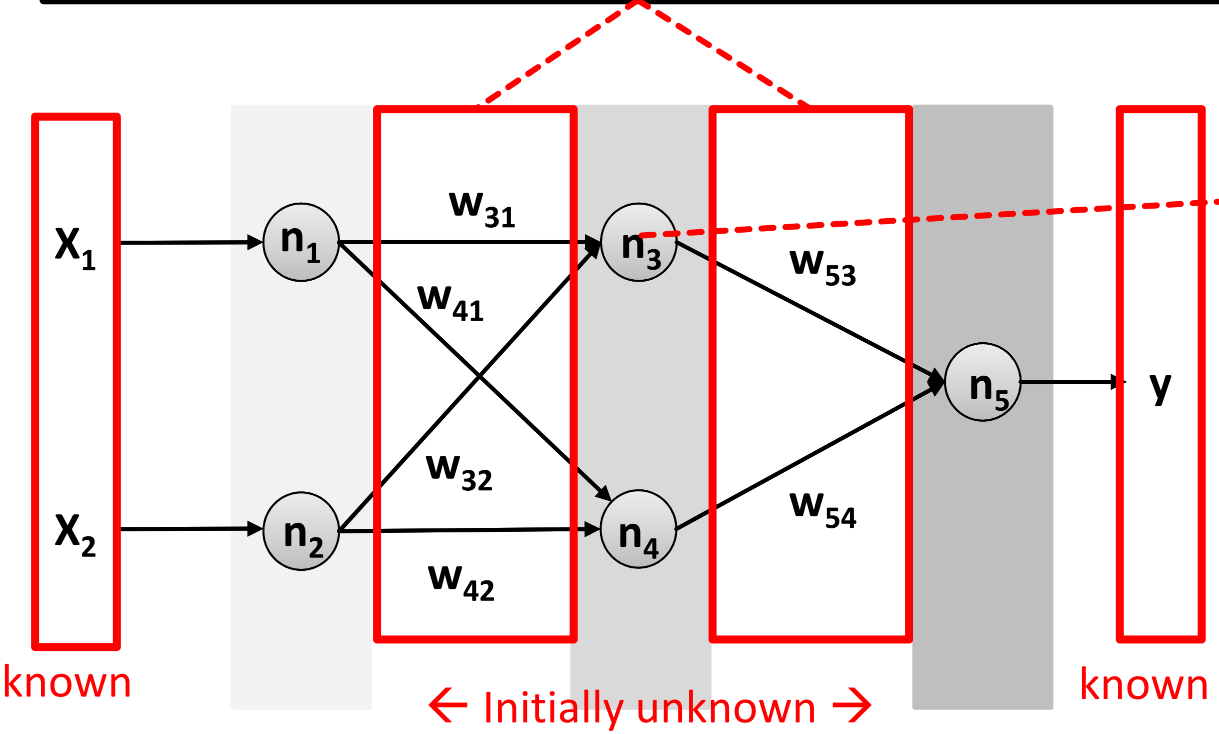
[6] Big Data Tips, Gradient Descent

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

[3] Introduction to Data Mining

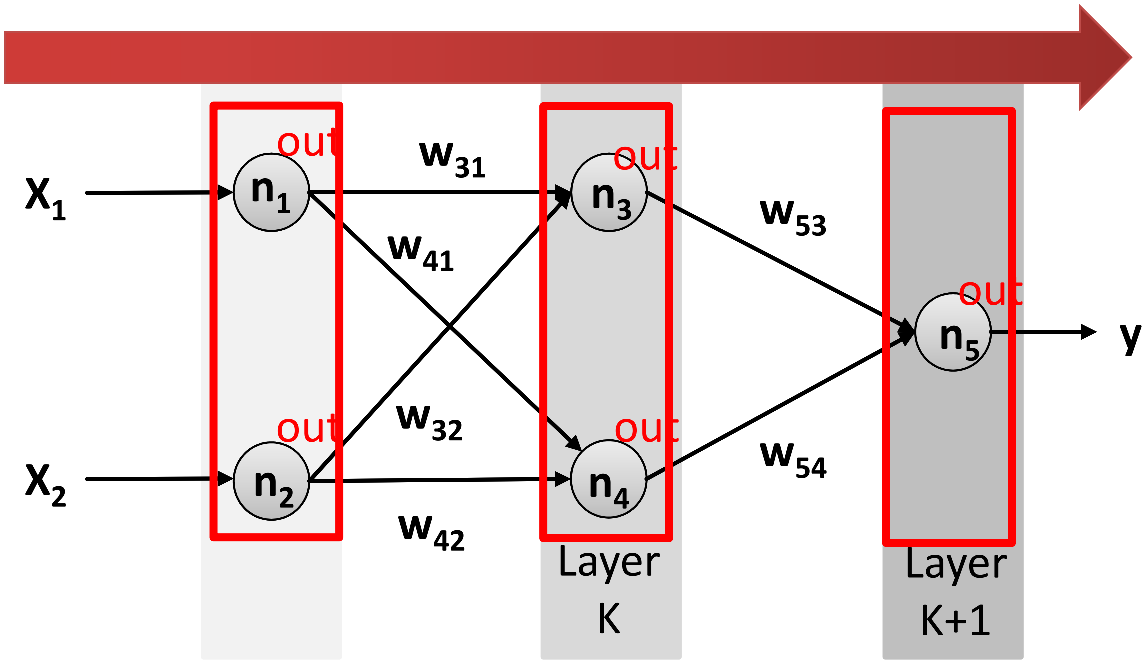


- **Hidden nodes problem: computing error term hard: $\partial E / \partial wj$**
- **Their Output values are unknown to us (here)…**

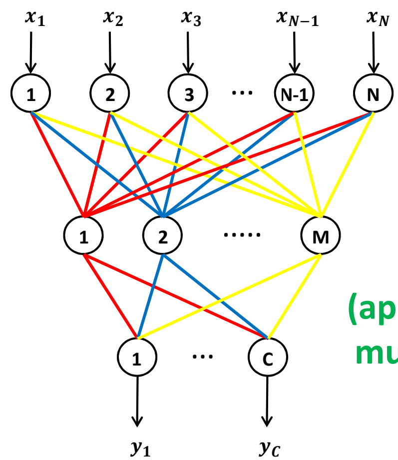
- **The backpropagation algorithm solves exactly this problem with two phases per iteration(!)**

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



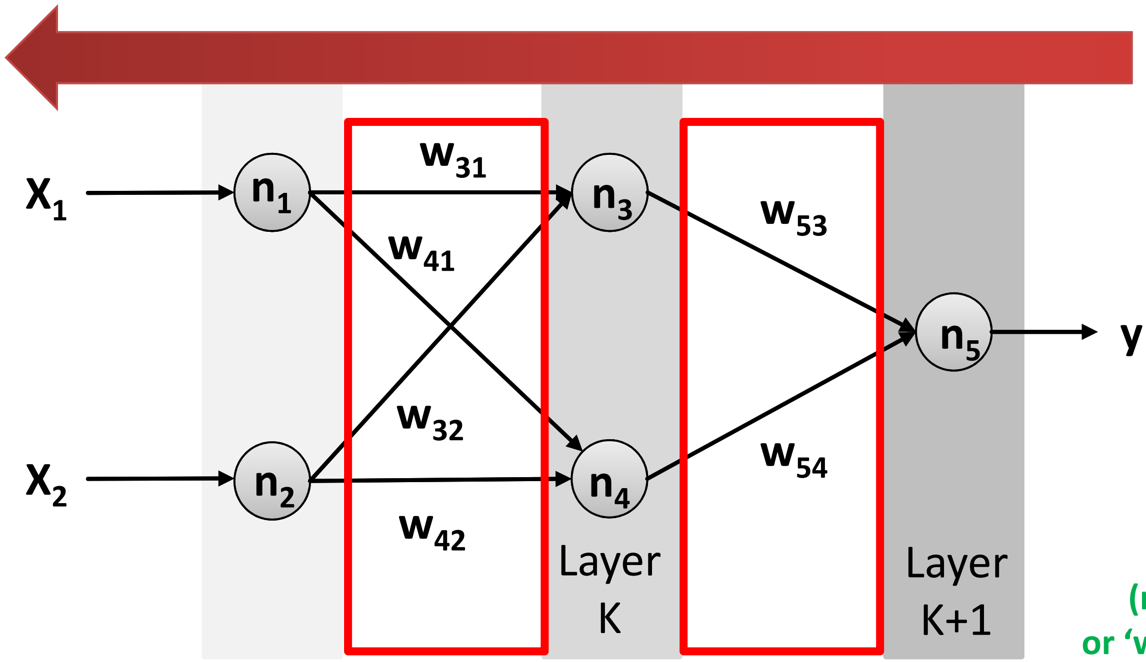
- **Use corresponding ‘activation function’ but with ‘old weights’**



(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



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

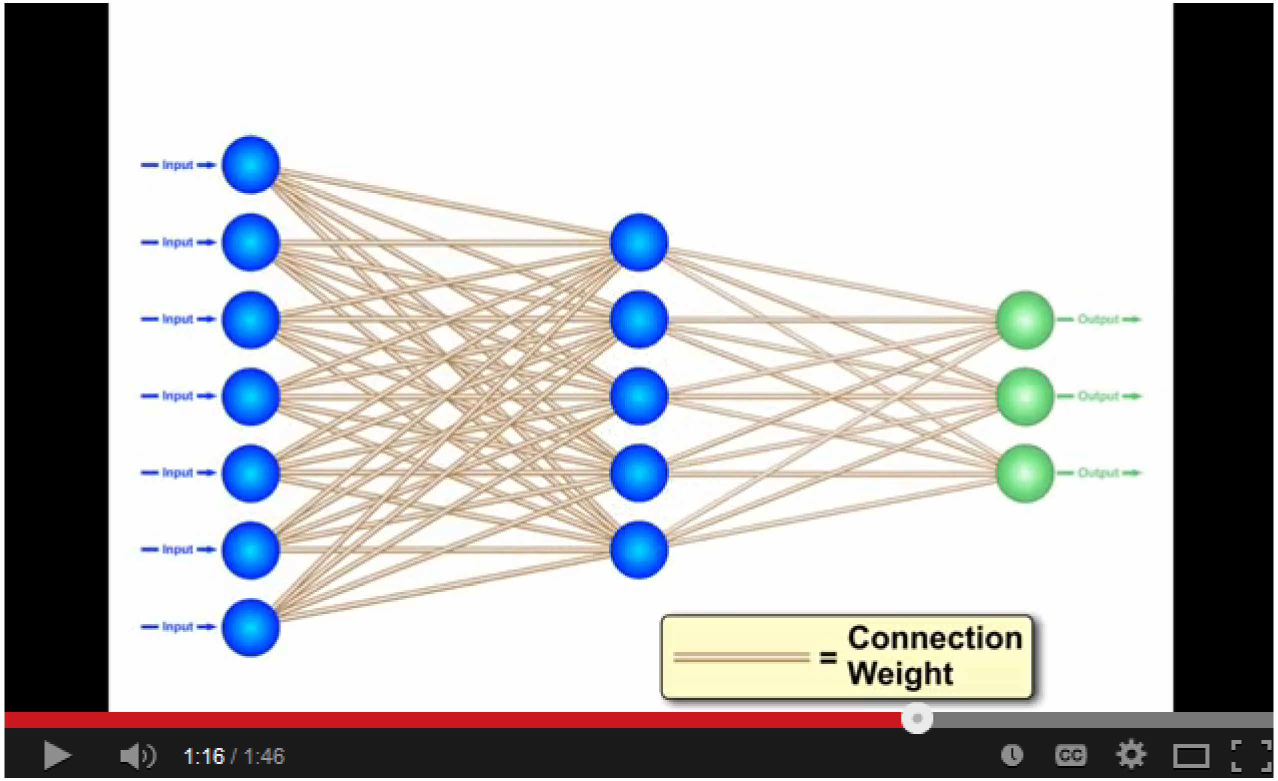
weight update formula of the 'gradient descent method'

Now that can compute the error one-by-one

$$E_{in}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$$

(regularization method 'weight decay' or 'weight drop' is used in neural networks')

[Video] Towards Multi-Layer Perceptrons



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

High-level Tools – Keras

- **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**

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

- **Tool Keras supports inherently the creation of artificial neural networks using Dense layers and optimizers (e.g. SGD)**
- **Includes regularization (e.g. weight decay) or momentum**

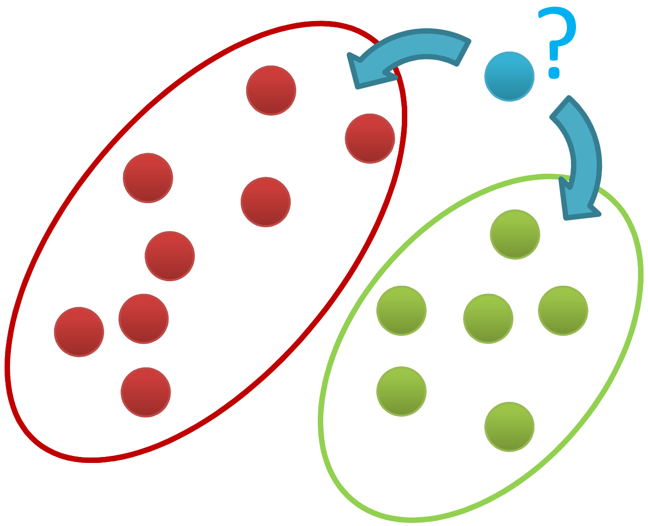
Keras

[19] Keras Python Deep Learning Library

Methods Overview – Focus in this Lecture

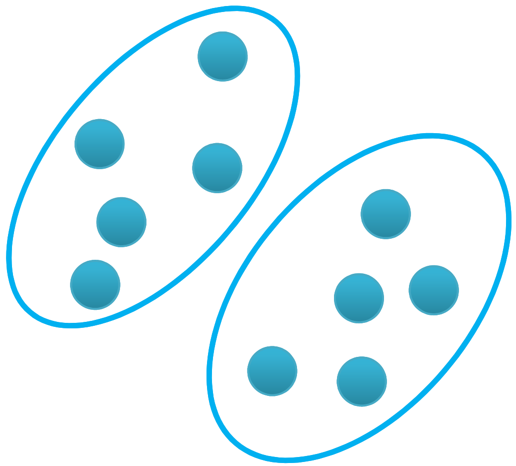
- **Statistical data mining methods can be roughly categorized in classification, clustering, or regression augmented with various techniques for data exploration, selection, or reduction**

Classification



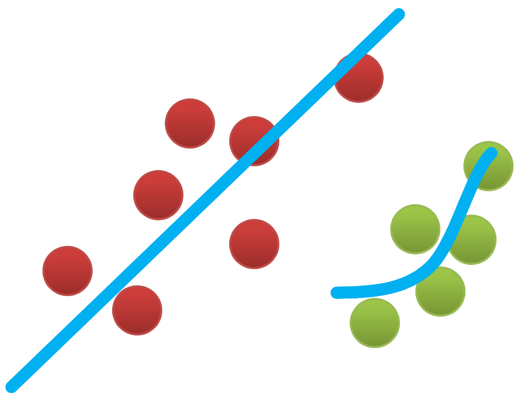
- Groups of data exist
- New data classified to existing groups

Clustering



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

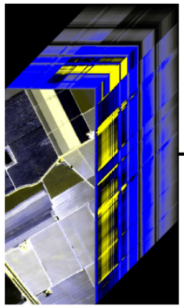
Regression



- Identify a line with a certain slope describing the data

ANN – Application Example Remote Sensing 'SALINAS'

- Hyperspectral data (AVIRIS sensor)
  - 'Salinas' Valley, California
  - Spectral resolution: 224 bands
  - Spatial resolution: 3.7 meter pixels

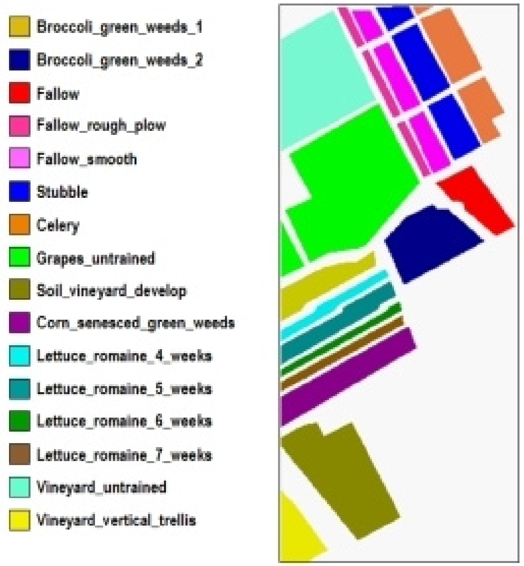


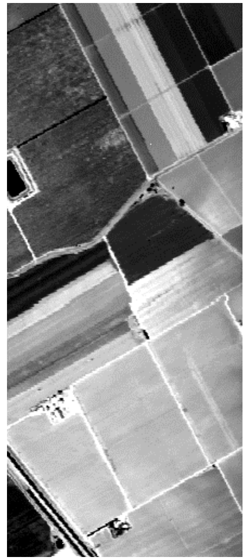
224 bands

(DAFE = Discriminant Analysis Feature Extraction;
SDAP = Self Dual Attribute Profile)

DAFE (14 Feat.) → **Feature reduction**
↓
SDAP (243 Feat.) → **Spatial analysis**
↓
DAFE (14 Feat.) → **Feature reduction**
↓
ANN LEARNING ← TRAINING SAMPLES
↓
ANN PREDICTION ← TEST SAMPLES
↓
Classification

GROUNDTRUTH → TRAINING SAMPLES, TEST SAMPLES



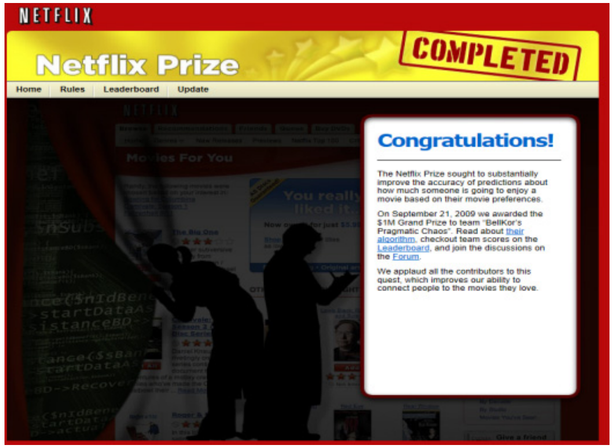


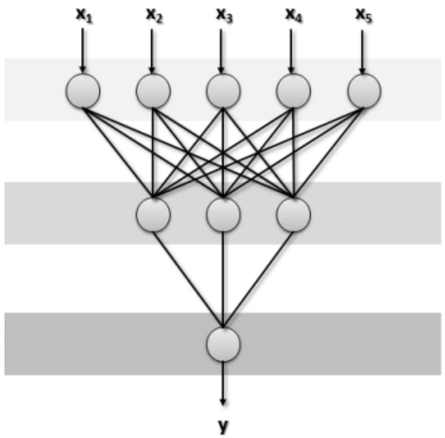
(OA = Overall Accuracy; AA = Average Accuracy;
K = Kappa coefficient obtained by classifiers)

	Original (204 Feat.)			DAFE (14 Feat.)			DAFE_SDAP_DAFE (14 Feat.)		
	RF	SVM	ANN	RF	SVM	ANN	RF	SVM	ANN
AA	91.46	**93,11**	84,73	**94,38**	94,23	92,92	97,68	**98,02**	95,84
OA	87.75	89,12	**92,27**	89,89	88,22	**94,91**	96,02	96,77	**97,16**
K	86.34	87,87	**91,37**	88,72	86,89	**94,32**	95,57	96,4	**96,84**

ANN – Application Example in Industry

- ~2009 - Netflix Prize Challenge 2009
  - Data: Netflix company provided data to learn from previous movie rentals
  - Challenge: Improve Netflix in-house movie recommender system
  - Prize: 1.000.000 US $ for team with 10% improvements
  - Approaches: Machine learning algorithms and collaborative filterings
  - Winner: Prize received by working with Artificial Neural Network (ANNs)





[5] A. Töscher and M. Jahrer, 'The BigChaos Solution to the Netflix Grand Prize', 2009

ANN – Handwritten Character Recognition MNIST Dataset

- 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

- 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 for the Tutorial

- When working with the dataset
  - Dataset is not in any standard image format like jpg, bmp, or gif
  - File format not known to a graphics viewer
  - 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
  - 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.
- Available already for the tutorial
  - Part of the Tensorflow tutorial package and Keras tutorial package

```
# download & unpack MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Exercises

Ugent Tier-2 Clusters

- Using Parallel Computing
  - Compiled from open source
  - Requires MPI library
  - Intended to be used by High Performance Computing system (i.e. good interconnects)

- Job runs
  - Use of job scripts
  - Depend on scheduler

- **Use our ssh keys to get an access and use reservation**
- **Put the private key into your ./ssh directory (UNIX)**
- **Use the private key with your putty tool (Windows)**

Vlaams Supercomputer Centrum

[2] UGent Tier-2 Clusters

UGent Tier-2 Clusters – GOLETT in the Tutorial

	#nodes	CPU	Mem/node	Diskspace/node	Network
Raichu	64	2 x 8-core Intel E5-2670 (Sandy Bridge @ 2.6 GHz)	32 GB	400 GB	GbE
Delcatty	160	2 x 8-core Intel E5-2670 (Sandy Bridge @ 2.6 GHz)	64 GB	400 GB	FDR InfiniBand
Phanpy	16	2 x 12-core Intel E5-2680v3 (Haswell-EP @ 2.5 GHz)	512 GB	3x 400 GB (SSD, striped)	FDR InfiniBand
Golett	200	2 x 12-core Intel E5-2680v3 (Haswell-EP @ 2.5 GHz)	64 GB	500 GB	FDR-10 InfiniBand
Swalot	128	2 x 10-core Intel E5-2660v3 (Haswell-EP @ 2.6 GHz)	128 GB	1 TB	FDR InfiniBand

Vlaams Supercomputer Centrum

[2] UGent Tier-2 Clusters

UGent Tier-2 Clusters – Login & Module Swap Cluster/golett

```
adminuser@linux-8djg:~> ssh vsc42544@login.hpc.ugent.be
Last login: Wed Nov 22 17:15:00 2017 from pool-216-7-zam606.vpn.kfa-juelich.de

STEVIN HPC-UGent infrastructure status on Wed, 22 Nov 2017 22:15:01

   cluster - full - free -  part - total - running - queued
             nodes  nodes   free   nodes     jobs      jobs
------------------------------------------------------------------
  delcatty   153      0       4     159      N/A       N/A
    golett   102     35      57     196      N/A       N/A
    phanpy    11      0       5      16      N/A       N/A
    raichu    56      0       0      56      N/A       N/A
    swalot   107      0      21     128      N/A       N/A

For a full view of the current loads and queues see:
 http://hpc.ugent.be/clusterstate/
Updates on maintenance and unscheduled downtime can be found on
 https://www.vscentrum.be/en/user-portal/system-status
```

```
[vsc42544@gligar01 ~]$ module swap cluster/golett

The following have been reloaded with a version change:
  1) cluster/delcatty => cluster/golett
```

[2] UGent Tier-2 Clusters

Copy Files to your Home Directory

```
[vsc42544@gligar03 deeplearning]$ pwd
/user/home/gent/vsc425/vsc42544/deeplearning
[vsc42544@gligar03 deeplearning]$ ls -al
total 1152
drwxrwxr-x 2 vsc42544 vsc42544   4096 Nov 29 22:28 .
drwx------ 6 vsc42544 vsc42544   4096 Nov 29 22:57 ..
-rwxrw-r-- 1 vsc42544 vsc42544    581 Nov 29 19:27 job_ann_hidden.sh
-rwxrw-r-- 1 vsc42544 vsc42544    567 Nov 29 19:17 job_ann.sh
-rw-r--r-- 1 vsc42544 vsc42544    394 Nov 28 18:12 job_basic.sh
-rwxrw-r-- 1 vsc42544 vsc42544    594 Nov 29 22:27 job_cnn.sh
-rwxrw-r-- 1 vsc42544 vsc42544    550 Nov 29 16:07 job_data.sh
-rw------- 1 vsc42544 vsc42544     26 Nov 29 21:21 KERAS_MNIST_ANN.e1179465
-rw------- 1 vsc42544 vsc42544     26 Nov 29 21:32 KERAS_MNIST_ANN_HIDDEN.e1179466
-rw------- 1 vsc42544 vsc42544 348173 Nov 29 21:32 KERAS_MNIST_ANN_HIDDEN.o1179466
-rw-r--r-- 1 vsc42544 vsc42544   1571 Nov 29 21:20 KERAS_MNIST_ANN_HIDDEN.py
-rw------- 1 vsc42544 vsc42544 216975 Nov 29 21:21 KERAS_MNIST_ANN.o1179465
-rw-r--r-- 1 vsc42544 vsc42544   1396 Nov 29 21:10 KERAS_MNIST_ANN.py
-rw-r--r-- 1 vsc42544 vsc42544   2170 Nov 29 22:28 KERAS_MNIST_CNN.py
-rw-r--r-- 1 vsc42544 vsc42544    739 Nov 29 16:07 KERAS_MNIST_DATA.py
-rw-r--r-- 1 vsc42544 vsc42544   1418 Nov 29 16:41 TF_MNIST_basic.py
```

ANN –MNIST Dataset – 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.utils import np_utils
```

```
# parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 128
OPTIMIZER = 'SGD'
VALIDATION_SPLIT = 0.2
```

- 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

```
# dataset 28 x 28 pixels = 784 reshaped
(X_train, y_train), (X_test, y_test) = mnist.load_data()
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')

# normalization
X_train /= 255
X_test /= 255

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

- Data load shuffled between training and testing set
- 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]

ANN – MNIST Dataset – A Simple Model

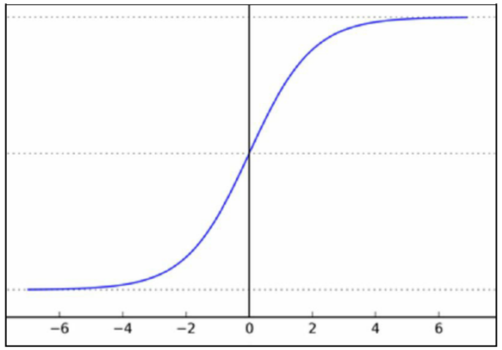
- The Sequential() Keras model is a linear pipeline (aka 'a stack') of various neural network layers including Activation functions of different types (e.g. softmax)

- 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 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, validation_split=VALIDATION_SPLIT)

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

$$\mathrm{softmax}(\mathbf{x})_i = \frac{\exp(\mathbf{x}_i)}{\sum_j \exp(\mathbf{x}_j)}$$



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

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

ANN – MNIST Dataset – Job Script

```
#!/bin/bash
#PBS -l nodes=1:ppn=all
#PBS -l walltime=1:0:0
#PBS -N KERAS_MNIST_ANN

module load TensorFlow/1.4.0-intel-2017b-Python-3.6.3
module load Keras/2.1.1-intel-2017b-Python-3.6.3

# make sure Keras is using TensorFlow as backend
export KERAS_BACKEND=tensorflow

export WORKDIR=$VSC_SCRATCH/${PBS_JOBNAME}_${PBS_JOBID}
mkdir -p $WORKDIR
cd $WORKDIR

export OMP_NUM_THREADS=1
python $PBS_O_WORKDIR/KERAS_MNIST_ANN.py

echo "Working directory was $WORKDIR"
```

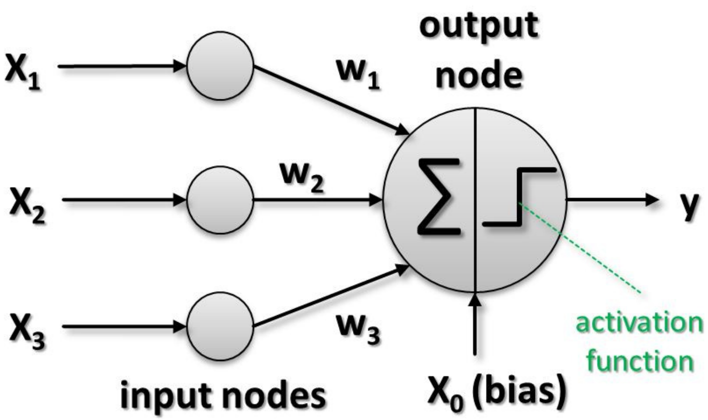
ANN –MNIST Dataset – A Simple Model – Output

```
[vsc42544@gligar03 deeplearning]$ more KERAS_MNIST_ANN.e1179465
Using TensorFlow backend.

[vsc42544@gligar03 deeplearning]$ more KERAS_MNIST_ANN.o1179465
60000 train samples
10000 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
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
```

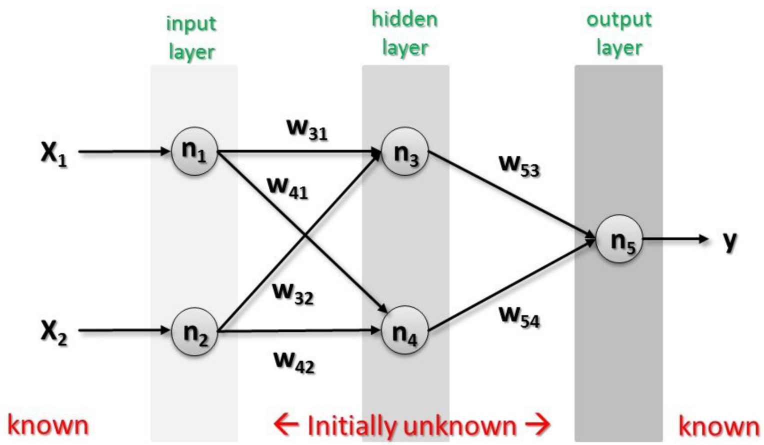
```
[vsc42544@gligar03 deeplearning]$ tail KERAS_MNIST_ANN.o1179465
48000/48000 [==============================] - 1s 12us/step - loss: 0.2760 - acc: 0.9227 - val_loss: 0.2747 - val_acc: 0.9234

   32/10000 [..............................] - ETA: 0s
 3104/10000 [========>.....................] - ETA: 0s
 6208/10000 [=================>............] - ETA: 0s
 9344/10000 [===========================>..] - ETA: 0s
10000/10000 [==============================] - 0s 16us/step
Test score: 0.277443544486
Test accuracy: 0.9221
Working directory was /user/scratch/gent/vsc425/vsc42544/KERAS_MNIST_ANN_1179465.master19.golett.gent.vsc
```

Artificial Neural Network – Feature Engineering & Layers

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



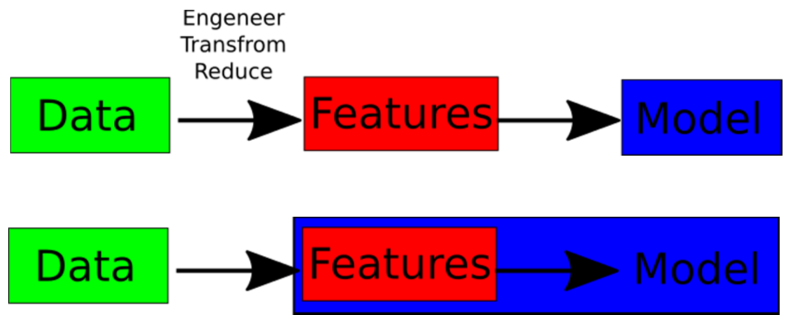
(Perceptron model: designed after human brain neuron)



(Artificial neural network two layer feed – forward)

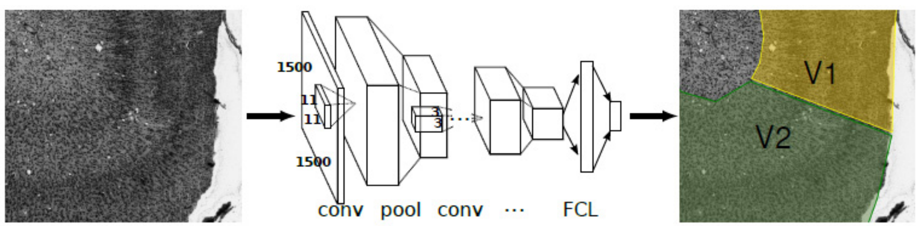
Deep Learning – Feature Learning & More Smart Layers

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

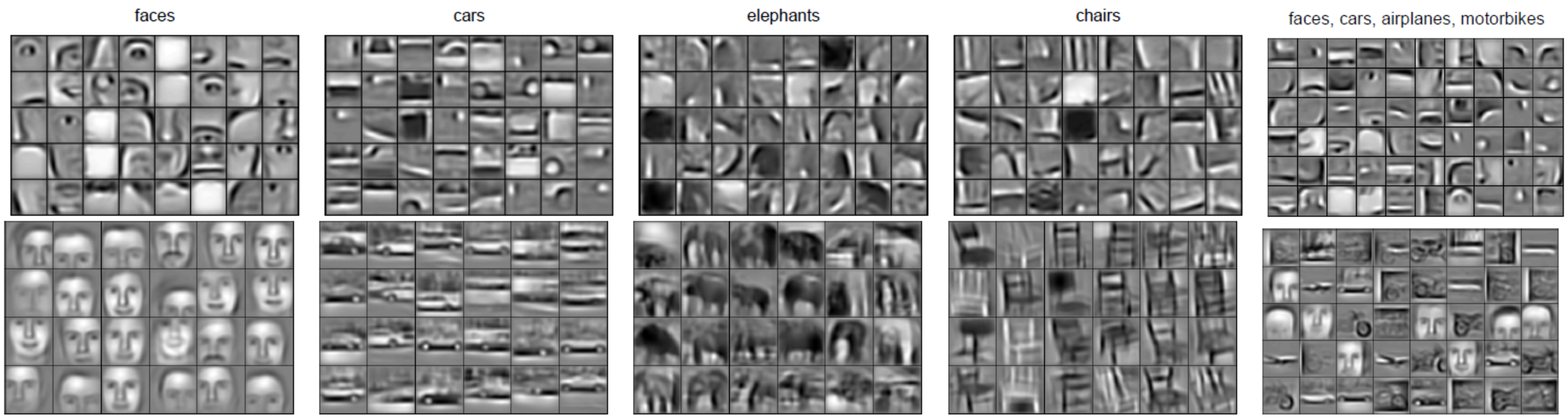


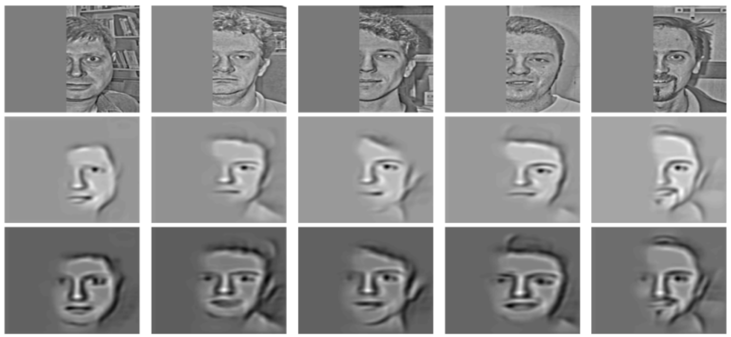
  - Very succesful for image recognition and other emerging areas
  - Assumption: data was generated by the interactions of many different factors on different levels (i.e. form a hierarchical representation)
  - Organize factors into multiple levels, corresponding to different levels of abstraction or composition(i.e. first layers do some kind of filtering)
  - Challenge: Different learning architectures: varying numbers of layers, layer sizes & types used to provide different amounts of abstraction

(Example: Parcellation of cytoarchitectonic cortical regions in the human brain)



Deep Learning – Feature Learning Benefits





[3] H. Lee et al., 'Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations'

- **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**

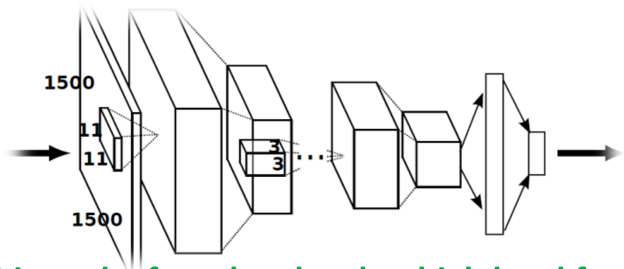
Deep Learning – Key Properties & Application Areas

- **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 (e.g. SVM)
  - Create better data representations and create deep learning models to learn these data representations from large-scale unlabeled data
- Application areas
  - Computer vision
  - Automatic speech recognition
  - Natural language processing
  - Bioinformatics
  - …

(Deep Learning is often characterized as 'buzzword')

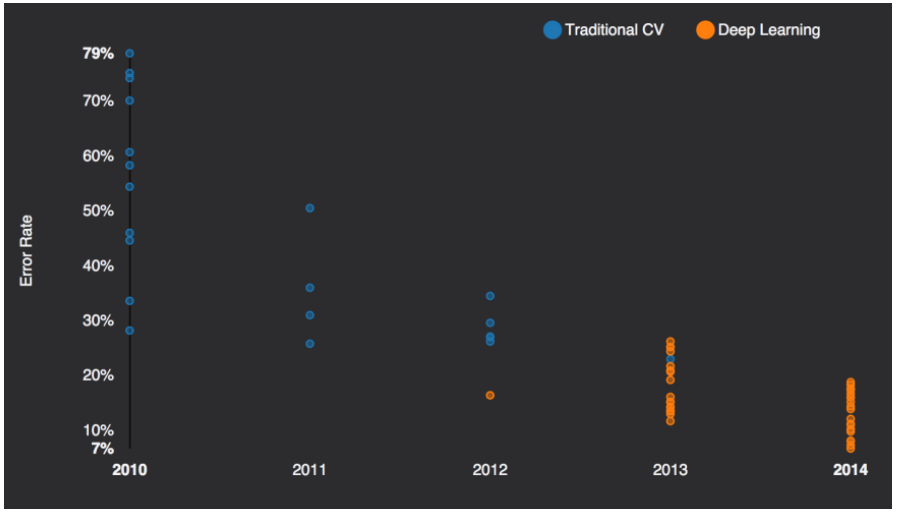
(Deep Learning is often 'just' called rebranding of traditional neural networks)



(hierarchy from low level to high level features)

Basic ImageNet Dataset as Base for Learning

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

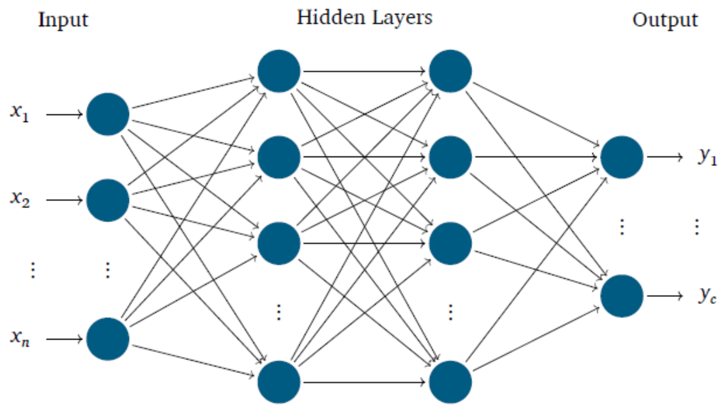


[7] J. Dean et al., 'Large-Scale Deep Learning'

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

[8] ImageNet Web page

Exercises – Add Hidden Layers



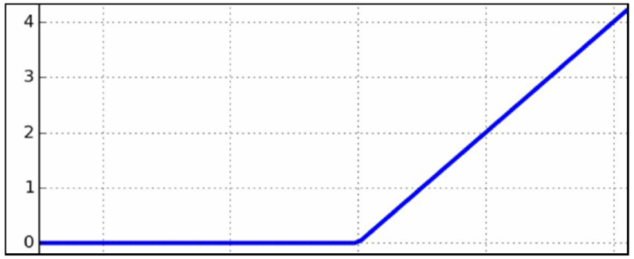
ANN – MNIST Dataset – Add Two Hidden Layers

- 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 – it just returns 0 for negative values and grows linearly for only positive values

```
# 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)

# ANN model with hidden layers
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, validation_split=VALIDATION_SPLIT)

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

$$f(x) = \max(0, x)$$



ANN – MNIST Dataset – Add Hidden Layers – JobScript

```
#!/bin/bash
#PBS -l nodes=1:ppn=all
#PBS -l walltime=1:0:0
#PBS -N KERAS_MNIST_ANN_HIDDEN

module load TensorFlow/1.4.0-intel-2017b-Python-3.6.3
module load Keras/2.1.1-intel-2017b-Python-3.6.3

# make sure Keras is using TensorFlow as backend
export KERAS_BACKEND=tensorflow

export WORKDIR=$VSC_SCRATCH/${PBS_JOBNAME}_${PBS_JOBID}
mkdir -p $WORKDIR
cd $WORKDIR

export OMP_NUM_THREADS=1
python $PBS_O_WORKDIR/KERAS_MNIST_ANN_HIDDEN.py

echo "Working directory was $WORKDIR"
```

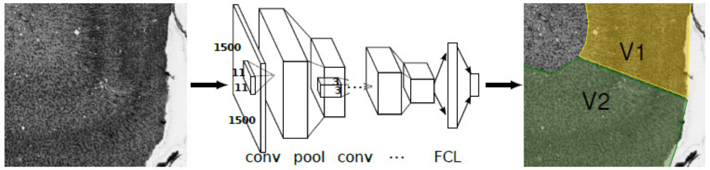
ANN – MNIST Dataset – Add Hidden Layers - Output

```
[vsc42544@gligar03 deeplearning]$ more KERAS_MNIST_ANN_HIDDEN.o1179466
60000 train samples
10000 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
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/200

  128/48000 [..............................] - ETA: 4:29 - loss: 2.3122 - acc: 0.1094
 2176/48000 [>.............................] - ETA: 16s - loss: 2.2732 - acc: 0.1085
 4864/48000 [==>...........................] - ETA: 7s - loss: 2.2178 - acc: 0.1721
 7424/48000 [===>..........................] - ETA: 4s - loss: 2.1676 - acc: 0.2515
```

```
[vsc42544@gligar03 deeplearning]$ tail KERAS_MNIST_ANN_HIDDEN.o1179466

   32/10000 [..............................] - ETA: 0s
 2272/10000 [=====>........................] - ETA: 0s
 4544/10000 [============>.................] - ETA: 0s
 6784/10000 [===================>..........] - ETA: 0s
 9088/10000 [==========================>...] - ETA: 0s
10000/10000 [==============================] - 0s 22us/step
Test score: 0.0772481116249
Test accuracy: 0.9773
Working directory was /user/scratch/gent/vsc425/vsc42544/KERAS_MNIST_ANN_HIDDEN_1179466.master19.golett.gent.vsc
```

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



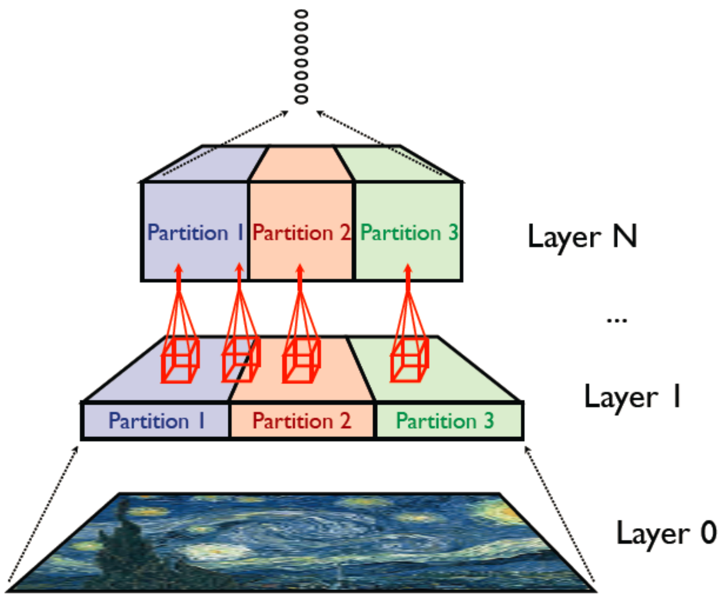
(focus in this course)

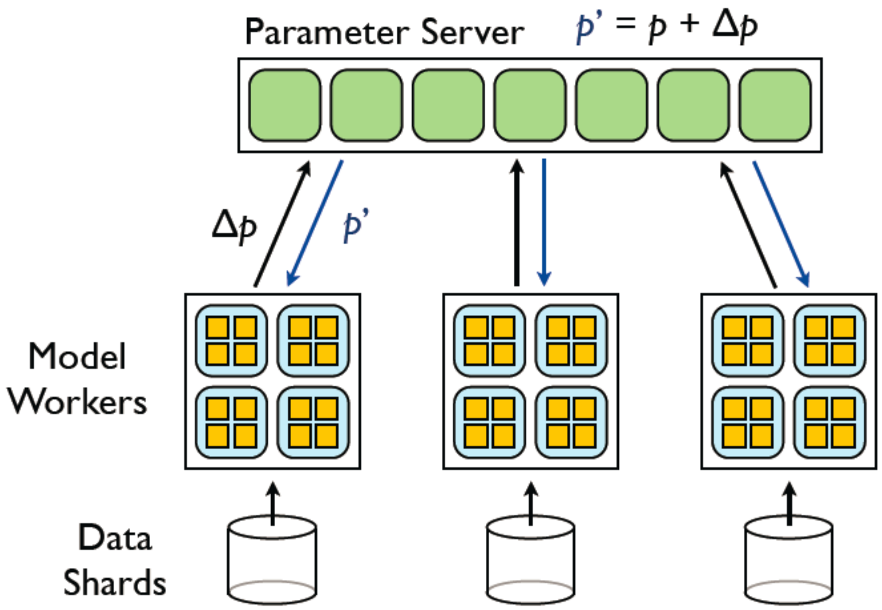
- Deep Belief Network (DBN)
  - Composed of mult iple layers of variables; only connections between layers
- Recurrent Neural Network (RNN)
  - ‘ANN‘ but connections form a directed cycle; state and temporal behaviour

- **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**

Deep Learning – Parallel Computing Methods

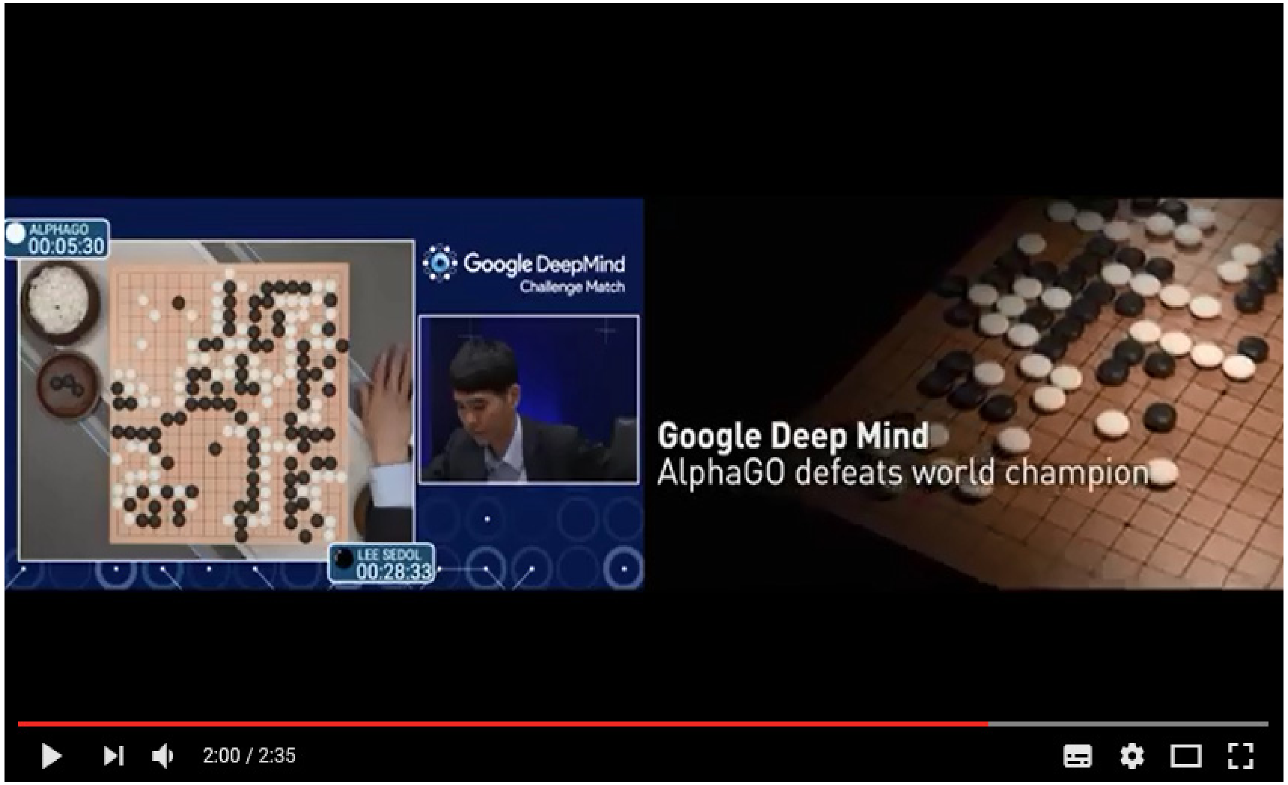
- Exploiting two kinds of parallelism
  - Model and data parallelism ('hierarchical domain decomposition')
  - Challenge: distributed asynchronous stochastic gradient descent algorithm
  - Minimal network cost: most densely connected areas are on one partition





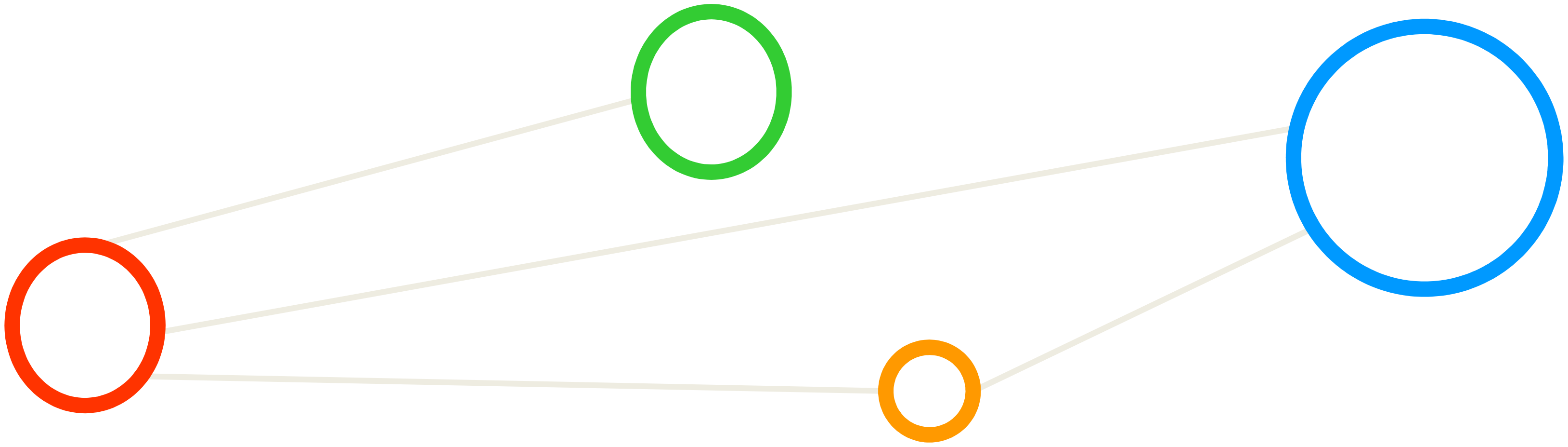
[7] J. Dean et al., 'Large-Scale Deep Learning'

[Video] Deep Learning 'Revolution'



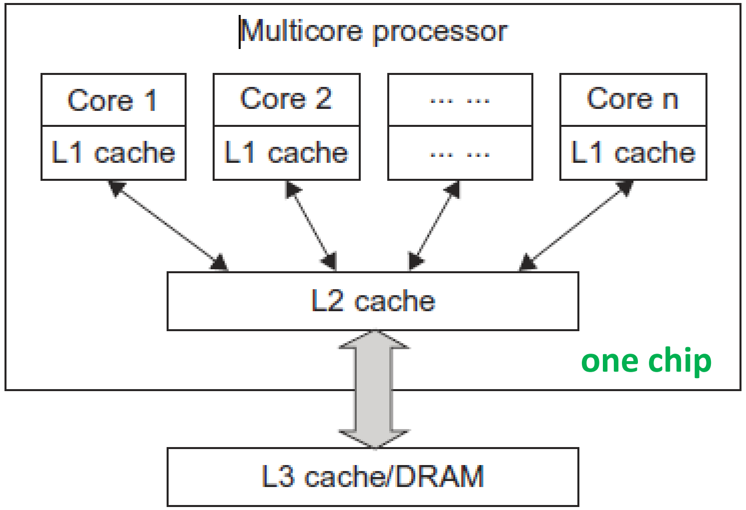
[9] The Deep Learning Revolution, YouTube

GPGPUs



Multi-core CPU Processors

- Significant advances in CPU (or microprocessor chips)
  - Multi-core architecture with dual, quad, six, or n processing cores
  - Processing cores are all on one chip
- Multi-core CPU chip architecture
  - Hierarchy of caches (on/off chip)
  - L1 cache is private to each core; on-chip
  - L2 cache is shared; on-chip
  - L3 cache or Dynamic random access memory (DRAM); off-chip



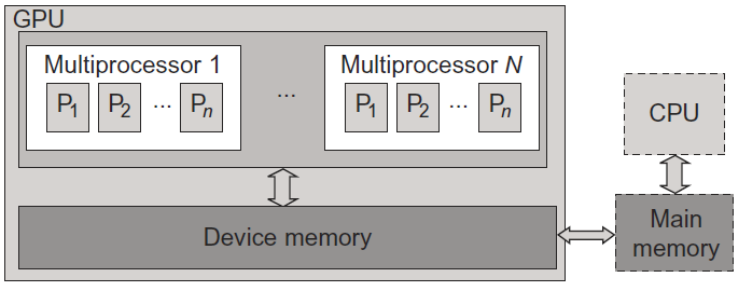
[10] Distributed & Cloud Computing Book

- **Clock-rate for single processors increased from 10 MHz (Intel 286) to 4 GHz (Pentium 4) in 30 years**
- **Clock rate increase with higher 5 GHz unfortunately reached a limit due to power limitations / heat**
- **Multi-core CPU chips have quad, six, or n processing cores on one chip and use cache hierarchies**

Many-core GPUs

- **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)

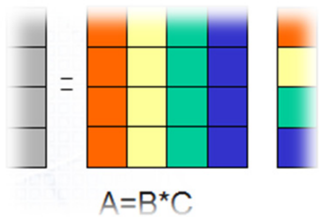


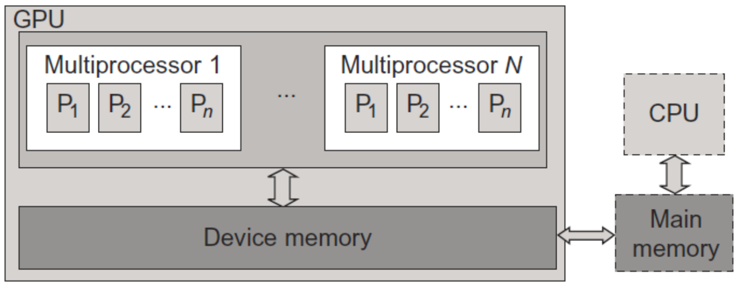
[10] Distributed & Cloud Computing Book

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. NVIDIA 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





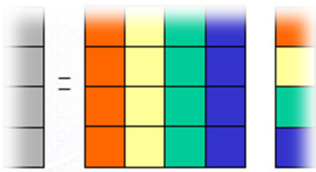
[10] Distributed & Cloud Computing Book

(other well known accelerators & many-core processors are e.g. Intel Xeon Phi → run 'CPU' applications easier)

Exercises

GPU Application Example – Matrix-Vector Multiplication

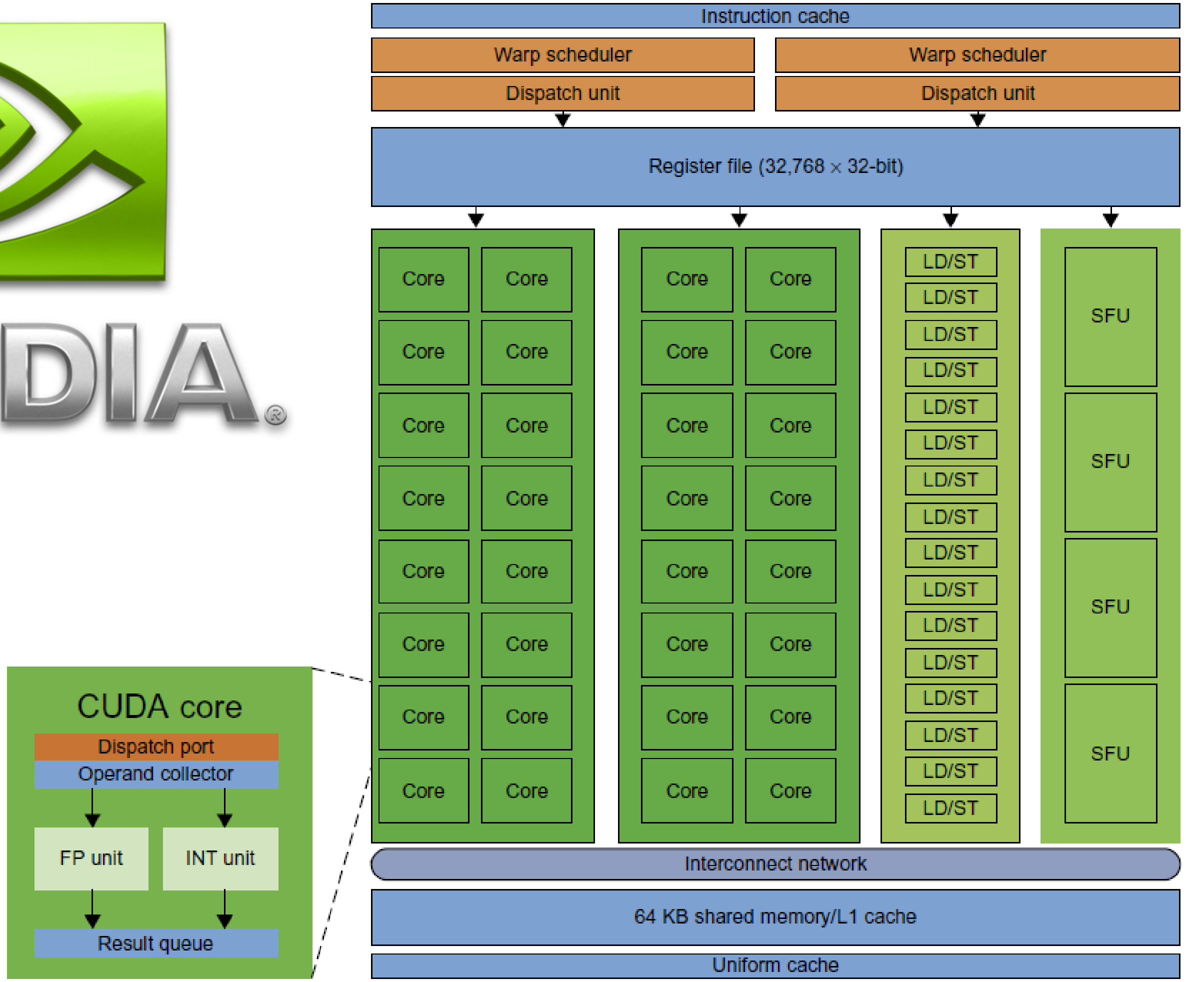
- What are the benefits of using GPUs in this application?



A=B*C

$$\begin{array}{cc} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = & \begin{bmatrix} b_{0,0}c_0 & + & b_{0,1}c_1 & + & b_{0,2}c_2 & + & b_{0,3}c_3 \\ b_{1,0}c_0 & + & b_{1,1}c_1 & + & b_{1,2}c_2 & + & b_{1,3}c_3 \\ b_{2,0}c_0 & + & b_{2,1}c_1 & + & b_{2,2}c_2 & + & b_{2,3}c_3 \\ b_{3,0}c_0 & + & b_{3,1}c_1 & + & b_{3,2}c_2 & + & b_{3,3}c_3 \end{bmatrix} \\ & \begin{matrix} \text{P0} & \quad & \text{P1} & \quad & \text{P2} & \quad & \text{P3} \end{matrix} \end{array}$$

NVIDIA Fermi GPU Example



[10] Distributed & Cloud Computing Book

GPGPUs – Terminology

- **General-Purpose Computing On Graphics Processing Units (GPGPUs)**
- **GPUs have been traditionally used to perform computing for computer graphics (e.g games)**
- **GPGPUs use GPUs to perform application computation instead or in addition to normal CPUs**

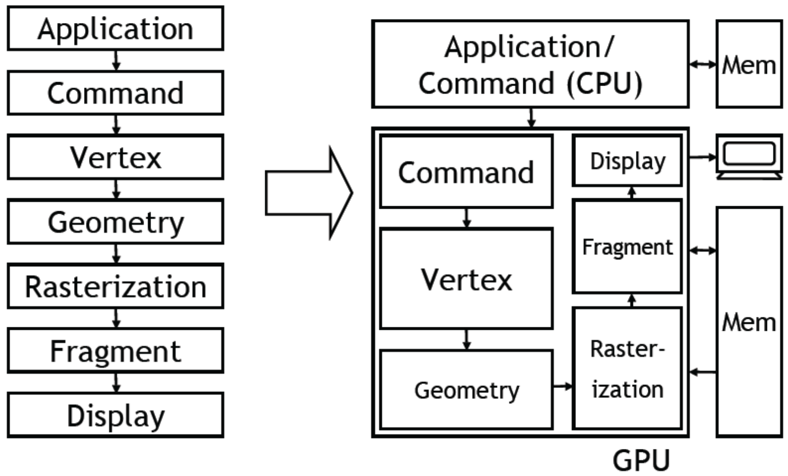
- Origin & HPC relationships
  - Starting ~2001 with reformulating computational problems in terms of graphics primitives (e.g. matrix multiplications)
- Programming Models
  - OpenCL as open general-purpose GPU programming model
  - NVidia Compute Unified Device Architecture (CUDA) as dominant propriety framework

[11] NVidia Tesla

- Selected Application Fields
  - GPU-accelerated scientific computing applications increasing
  - Increasing machine learning & statistical data mining implementations

GPGPUs – Architecture

- Parallelizes the already 'parallel nature of graphics processing'
  - Use of multiple graphics cards on one computer
  - Use of large numbers of graphics chips
- Terminology
  - GPU 'device'
  - CPU 'host'
  - Function 'kernel' (runs on device)
  - Vertices & fragments are elements in processing 'streams'

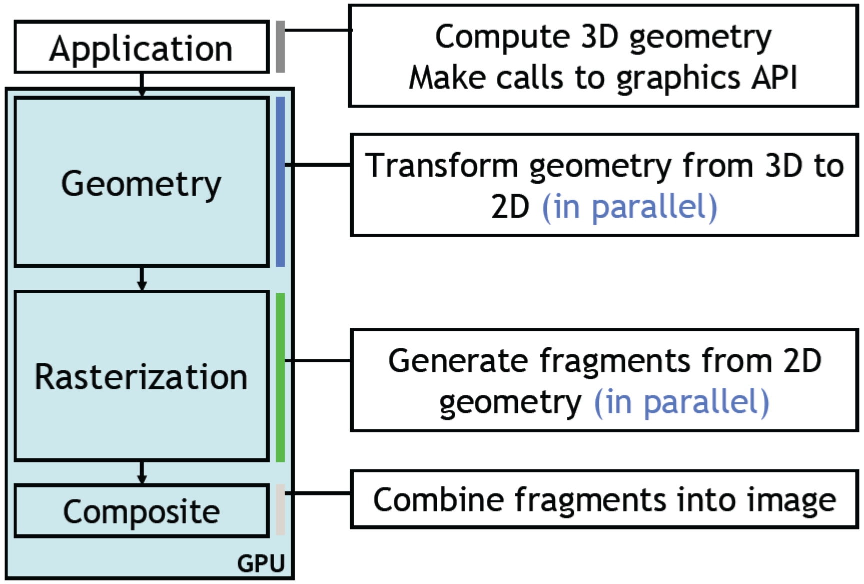


[11] J. Owens, GPGPU Architecture Overview

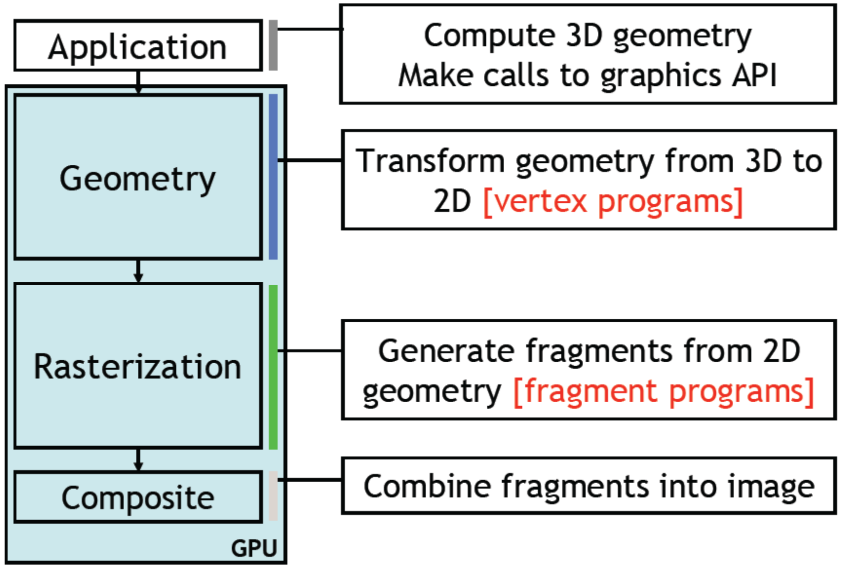
- **GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly**
- **In the context of GPUs, the Kernel is a function that runs on the GPU device**

GPGPUs – From Pure Graphics to General Processing

- Rendering Pipeline



- Programmable Pipeline

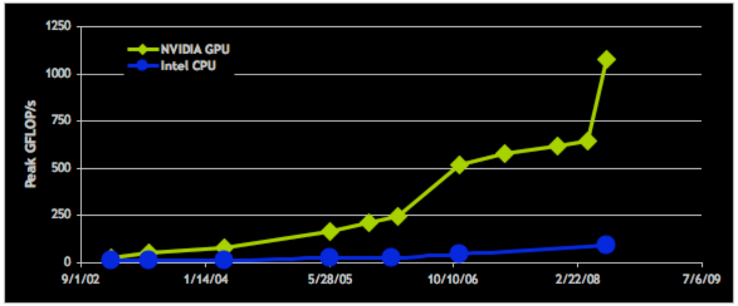


[11] J. Owens, GPGPU Architecture Overview

- **Rendering pipeline designed for massively parallelism and independent operations**
- **General processing in science and engineering partly rely on independent operations & data**

GPGPUs – Performance & Programming Approach

- GPUs are 'massively multithreaded' many-core chips
  - Hundreds of cores & thousands of concurrent threads
  - Aggressive performance growth



[31] NVidia Training Introduction

- Different to plain 'multi-core'

  (multi-core – heavy weight fast threads)

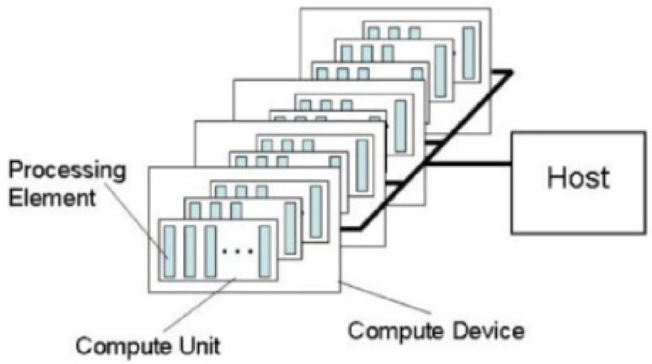
  (GPUs – fine light-weight slow threads)

- 'Stream' / data parallel programming approach
  - 'Set of records' that require similar computation (less communication)
  - Kernel functions are applied to each element of the stream

- **GPGPUs are very restrictive in operations and programming, but ideal for data parallel tasks**
- **GPGPUs are very effective for a set of records that require similar computation named as streams**

GPGPUs – Programming Model OpenCL

- Open Computing Language (OpenCL) *[12] Khronos Group, OpenCL*
  - The open standard for parallel programming of heterogeneous systems
  - Enable algorithms & programming patterns to be easily 'accelerated'



[13] Rastergrid, OpenCL platform model

- Practice
  - Hard to compete with NVidia CUDA & emerge as standard (e.g. MPI took > 10 years to position itself as the programming standard)

- **OpenCL is the open general-purpose GPU programming model approach that is vendor neutral**
- **Despite of the open standard OpenCL the de-facto standard in GPU programming is CUDA today**

GPGPUs – Programming Model CUDA

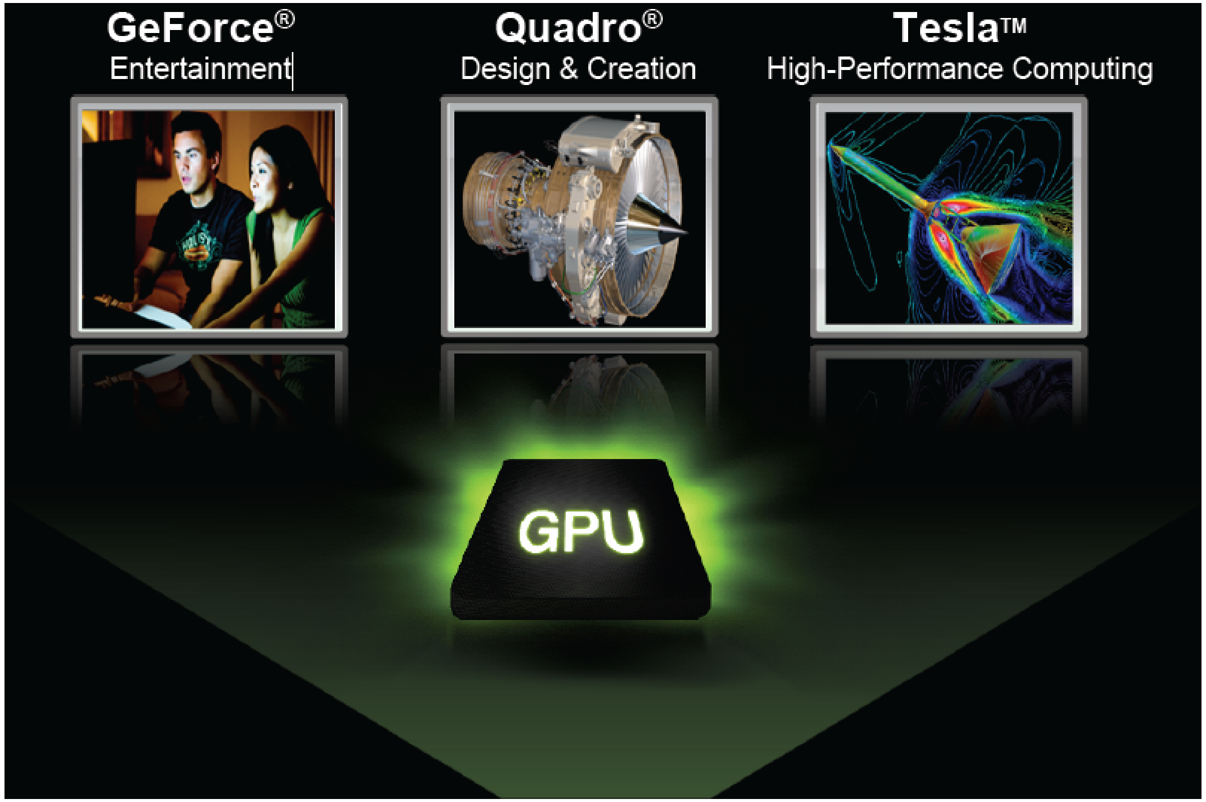
- Compute Unified Device Architecture (CUDA)
  - Industry standard programming model
  - Dominant since NVidia is major producer of GPGPUs in the market
  - Subset of programming language C
  - Defines a programming model and a memory model

- (Unlimited) Scalability
  - Parallel portions of application executed on the GPU device as kernels
  - Program for one thread can be instantiated on many parallel threads
  - Program runs on any number of processors without recompiling

- **CUDA is the dominant propriety general-purpose GPU programming model that is vendor-specific**

GPGPUs – NVidia Usage Models

- Example: Three 'different types of NVidia GPUs'
  - Designed for different levels of performance requirements



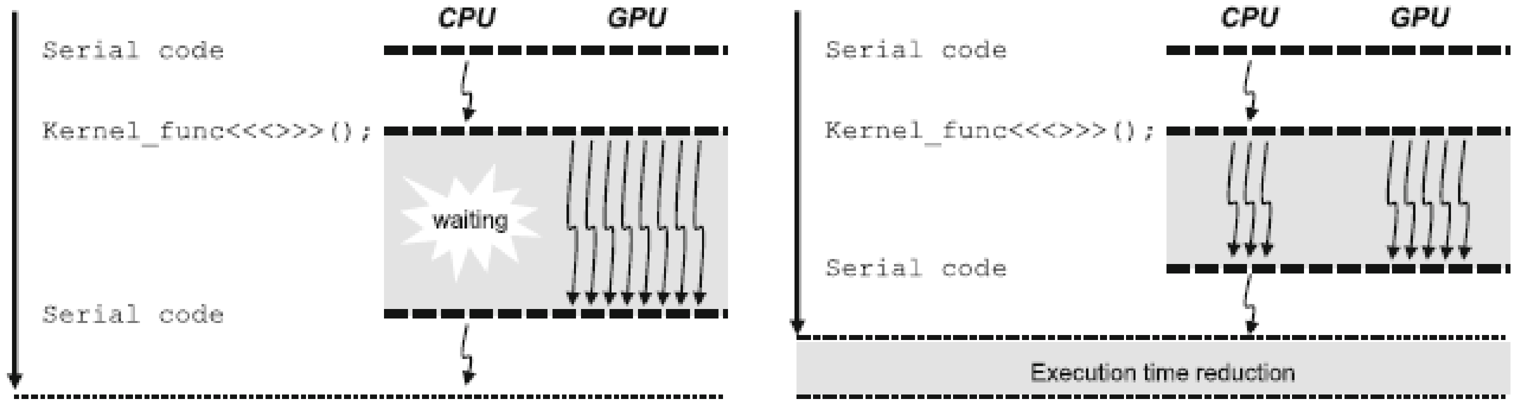
(constant evolution: NVidia Fermi as successor of NVidia Tesla)

(constant innovation: NVidia Kepler) as successor of NVidia Fermi)

[14] NVidia Training Introduction

Hybrid Programming: CPUs & GPGPUs Revisited

- Emerging 'hybrid programming model'
  - Using General-purpose computing on graphics processing units (GPGPUs)
  - Combine with traditional CPUs to accelerate elements of processing
  - Idea: exploit parallelism across host CPU cores in addition to the GPU cores



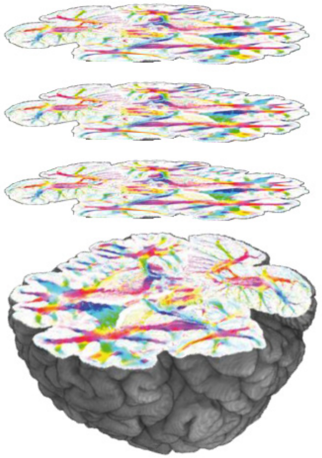
[15] 'Boosting CUDA Applications with CPU-GPU Hybrid Computing'

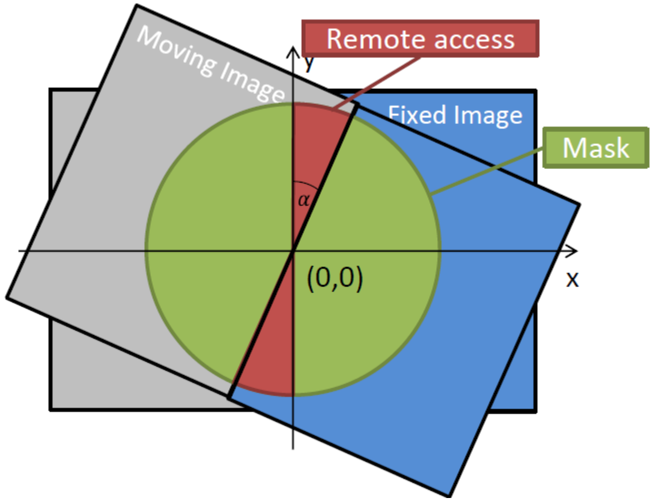
(constant innovation: new generation Intel Knights Landing many integrated cores (MIC) → run directly as host CPUs)

- **One drawback of a typical GPU is that it requires a host CPU in order to be used in a HPC system**

GPGPUs – Selected Applications

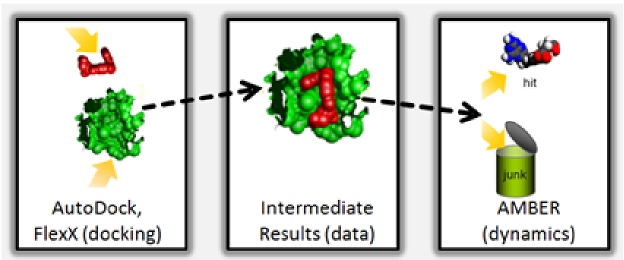
- Human brain research
  - Example: Registration of high resolution brain images





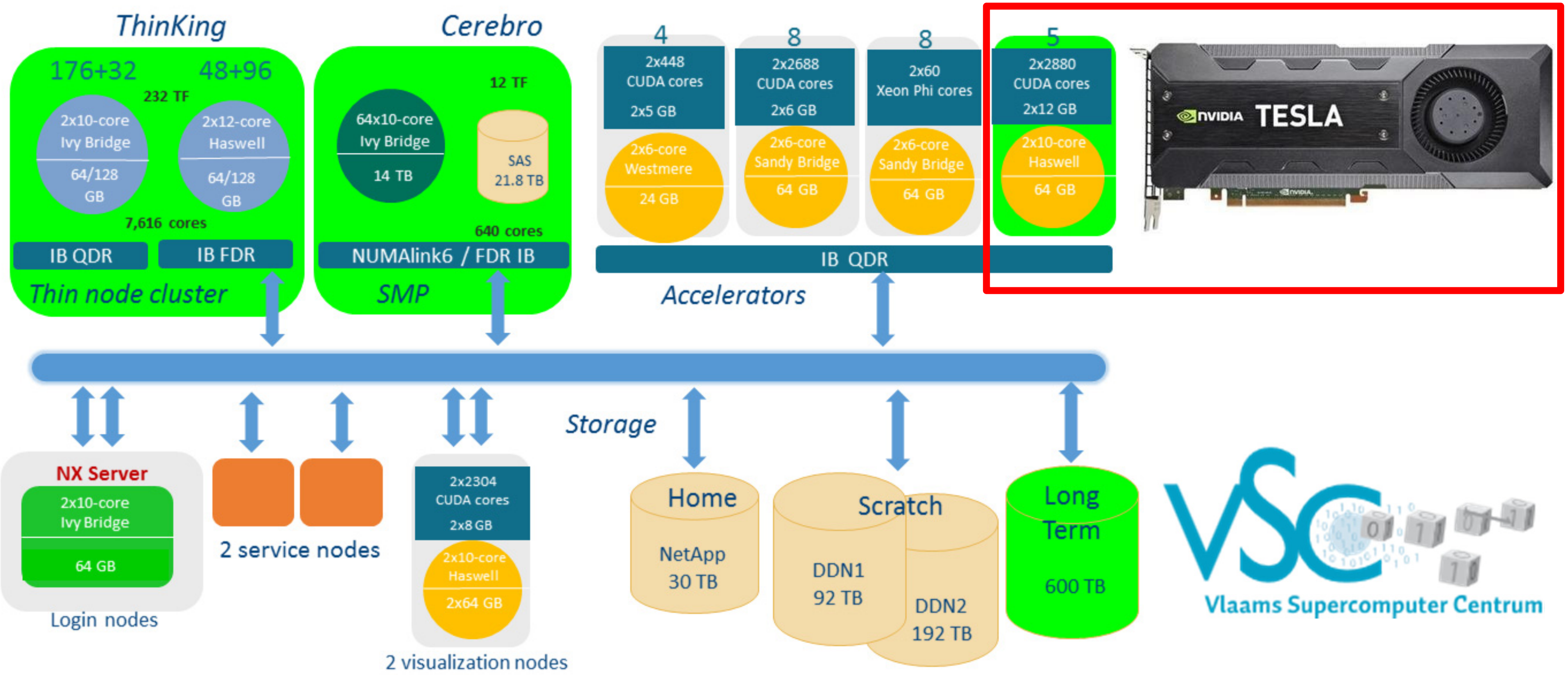
[16] NVidia Application Lab Juelich

- AMBER / CHARMM applications
  - Traditional HPC Applications
  - Molecular dynamics package to simulate molecular dynamics on biomolecules
  - Emerging support for GPGPU processing



HPC System KU Leuven – GPUs

- Accelerators
  - Nodes with two 10-core "Haswell" Xeon E5-2650v3 2.3GHz CPUs, 64 GB of RAM and 2 GPUs Tesla K40



modified from [18] HPC System KU Leuven

Exercises – Check Login into the KU Leuven System

```
[vsc42544@gligar03 ~]$ ssh login.hpc.kuleuven.be
```

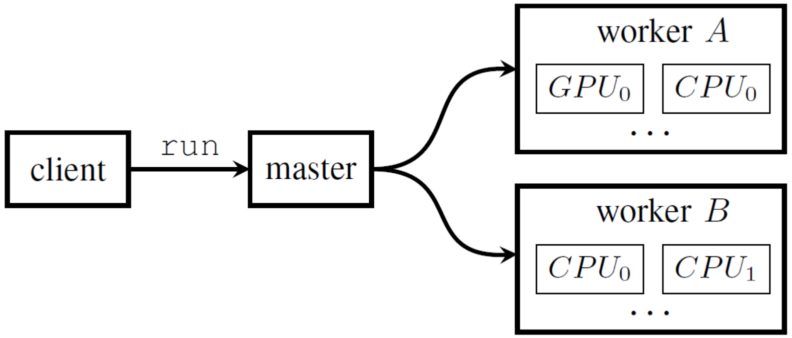
Keras with Tensorflow Backend – GPU Support

- **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**

Keras *[19] 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**

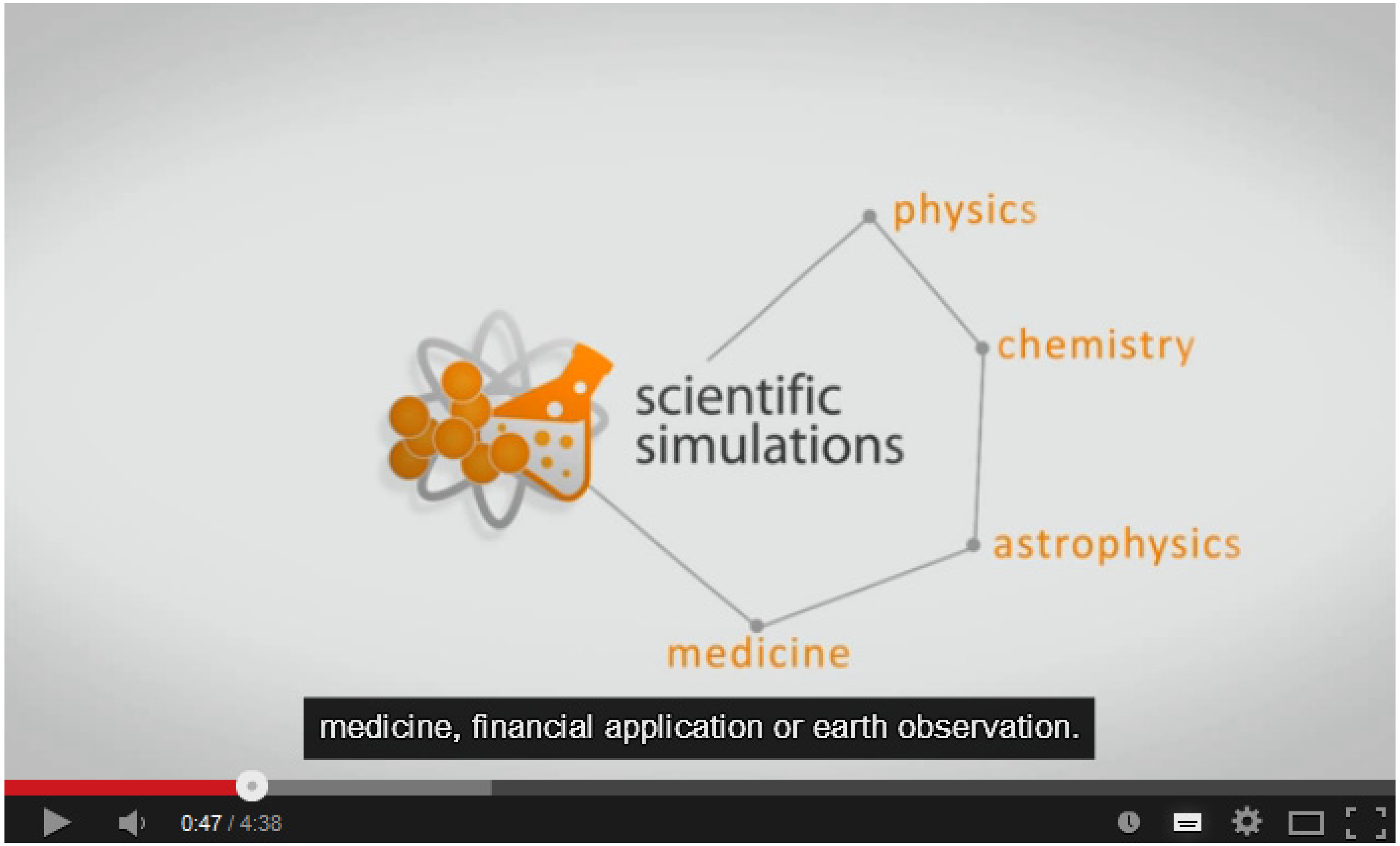
[20] Tensorflow Deep Learning Framework



[21] A Tour of Tensorflow

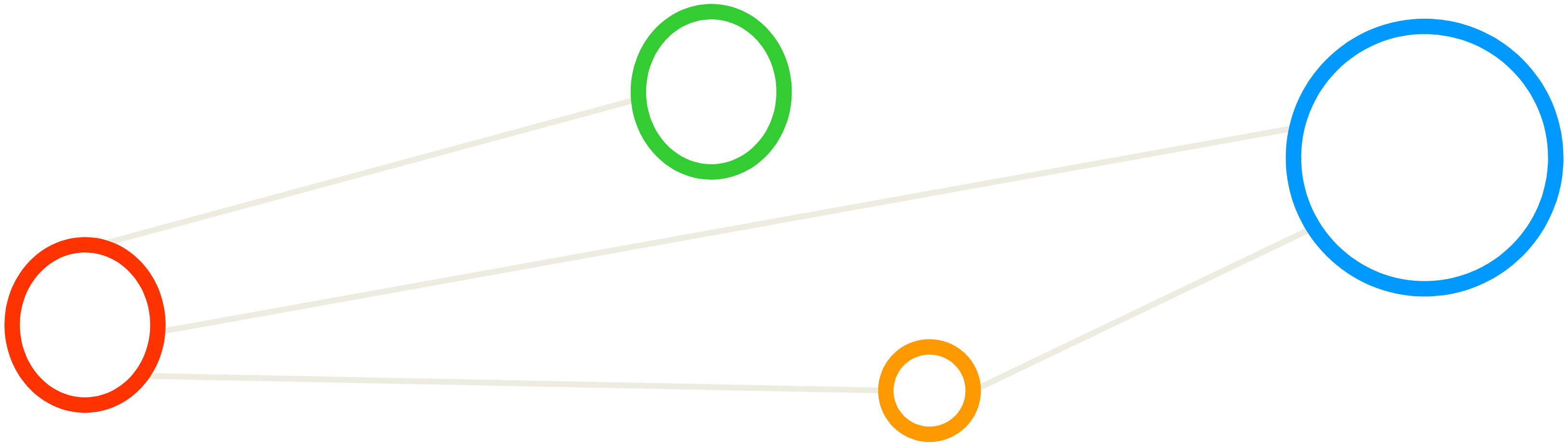
TensorFlow

[Video] GPGPUs & Applications



[17] 'HPC – GPGPUs', YouTube Video

Lecture Bibliography



Lecture Bibliography (1)

- [1] YouTube Video, ‘The Deep Learning Revolution’,
  Online: https://www.youtube.com/watch?v=Dy0hJWltsyE
- [2] YouTube Video, ’Neural Networks, A Simple Explanation’,
  Online: http://www.youtube.com/watch?v=gcK_5x2KsLA
- [3] 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
- [4] Rosenblatt,’The Perceptron: A probabilistic model for information storage and orgainzation in the brain’, Psychological Review 65(6), pp. 386-408, 1958
- [5] Andreas Töscher and Michael Jahrer, The BigChaos Solution to the Netflix Grand Prize, 2009
- [6] Big Data Tips, ‘Gradient Descent‘,
  Online: http://www.big-data.tips/gradient-descent
- [7] J. Dean et al., ‘Large scale deep learning’, Keynote GPU Technical Conference, 2015
- [8] ImageNet Web page,
  Online: http://image-net.org
- [9] YouTube Video, ‘The Deep Learning Revolution’,
  Online: https://www.youtube.com/watch?v=Dy0hJWltsyE
- [10] 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
- [11] NVidia Tesla,
  Online: http://www.nvidia.de/object/tesla-high-performance-computing-de.html

Lecture Bibliography (2)

- [11] J.Owens, 'GPGPU Architecture Overview',
  Online: http://gpgpu.org/static/s2007/slides/02-gpu-architecture-overview-s07.pdf
- [12] Khronos Group, 'OpenCL',
  Online: https://www.khronos.org/opencl/
- [13] Rastergrid Blog Figure,
  Online: http://rastergrid.com/blog/2010/11/texture-and-buffer-access-performance/
- [14] SDSC, Nvidia Training – Introduction,
  Online: http://www.sdsc.edu/us/training/assets/docs/NVIDIA-01-Intro.pdf
- [15] Changmin Lee, Won Woo Ro, Jean-Luc Gaudiot, 'Boosting CUDA Applications with CPU–GPU Hybrid Computing', Int J Parallel Prog (2014) 42:384–404, DOI 10.1007/s10766-013-0252-y
- [16] Juelich NVidia Application Lab @SC2013,
  Online: http://on-demand.gputechconf.com/supercomputing/2013/presentation/SC3134-GPU-Accelerated-Applications-Julich.pdf
- [17] 'HPC – Get a low-cost supercomputer by unleashing the power of GPUs',
  Online: https://www.youtube.com/watch?v=HYlWxPeL9-k
- [18] HPC System KU Leuven,
  Online: https://www.vscentrum.be/infrastructure/hardware/hardware-kul
- [19] Keras Python Deep Learning Library,
  Online: https://keras.io/
- [20] Tensorflow Deep Learning Framework,
  Online: https://www.tensorflow.org/
- [21] A Tour of Tensorflow,
  Online: https://arxiv.org/pdf/1610.01178.pdf


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