

# Machine Intelligence:: Deep Learning

## Week 1

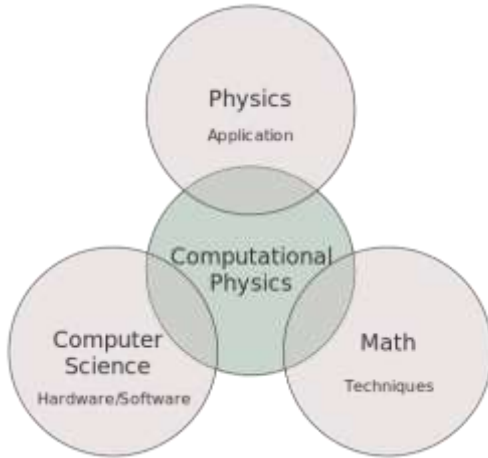
*Oliver Dürr*

Institut für Datenanalyse und Prozessdesign  
Zürcher Hochschule für Angewandte Wissenschaften

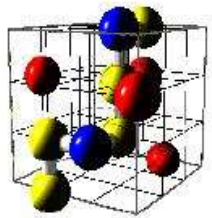
Winterthur, 20. Feb. 2018

# Me: From Computational Physics → Data Science

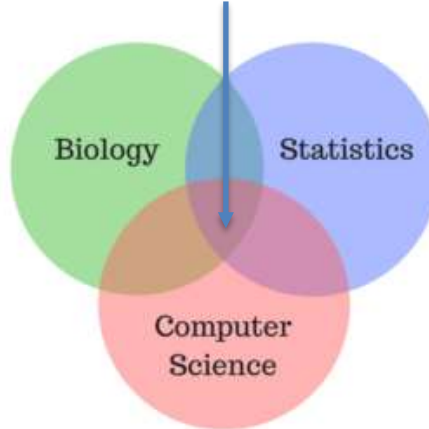
## Computational Physics



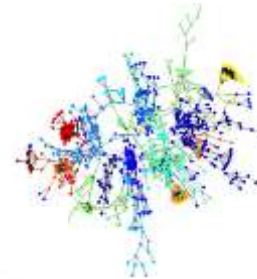
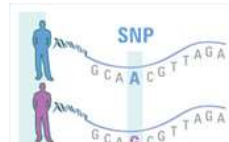
1990's  
Uni-Konstanz



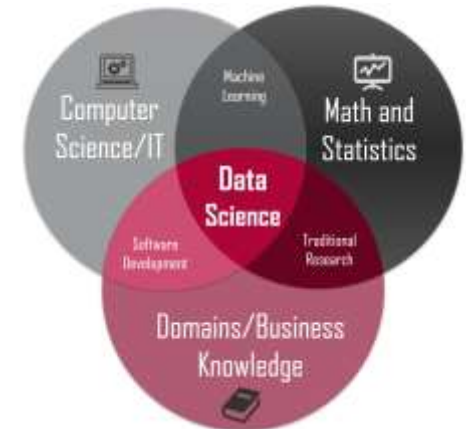
## Bioinformatics



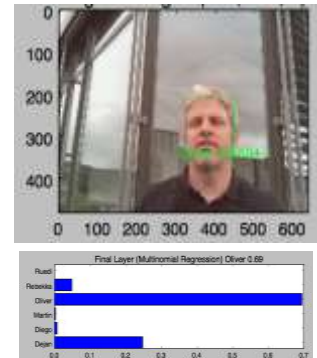
2000's  
Genedata Basel



## Data Science



2010's  
ZHAW Winterthur



# Before we start (check technical details)

- Website:
  - [https://tensorchiefs.github.io/dl\\_course\\_2018/](https://tensorchiefs.github.io/dl_course_2018/)
- Docker image: oduerr/tf\_docker:cpu\_r
  - Based on the official TF docker images with additional libraries needed for the course
  - How to use docker (see website)
  - As an alternative you can use the anaconda distribution
- Before we start make sure you have the latest image
  - `docker pull oduerr/tf_docker:cpu_r`

# Organizational Issues: ~~Test~~ Projects

- Projects (2-3 People)
- Presented on the last day
  - Spotlight talk (5 Minutes)
  - Poster
- Topics
  - You can / should choose a topic of your own (have to be discussed with us latest by week4)
  - Possible Topics
    - Take part in a Kaggle Competition (e.g. Leaf Classification / Dogs vs. Cats)
    - DeepFake
    - ...
- GPU power:
  - We have a google grant and can reimburse your costs to use the google cloud infrastructure (approx. 100-500 CHF)
  - Please talk to us until week 4

# Organizational Issues: Times

- Dates and times: see our webpage
- First 2 times
  - 13:30 – 15:00
  - 15:30 – 17:00
- Theory and exercises will be mixed
  - Could be 10 minutes theory 30 minutes exercises
  - Could be vice versa
- **Please interrupt us if something is unclear! The less we talk the better!**

# Learning Objectives for today

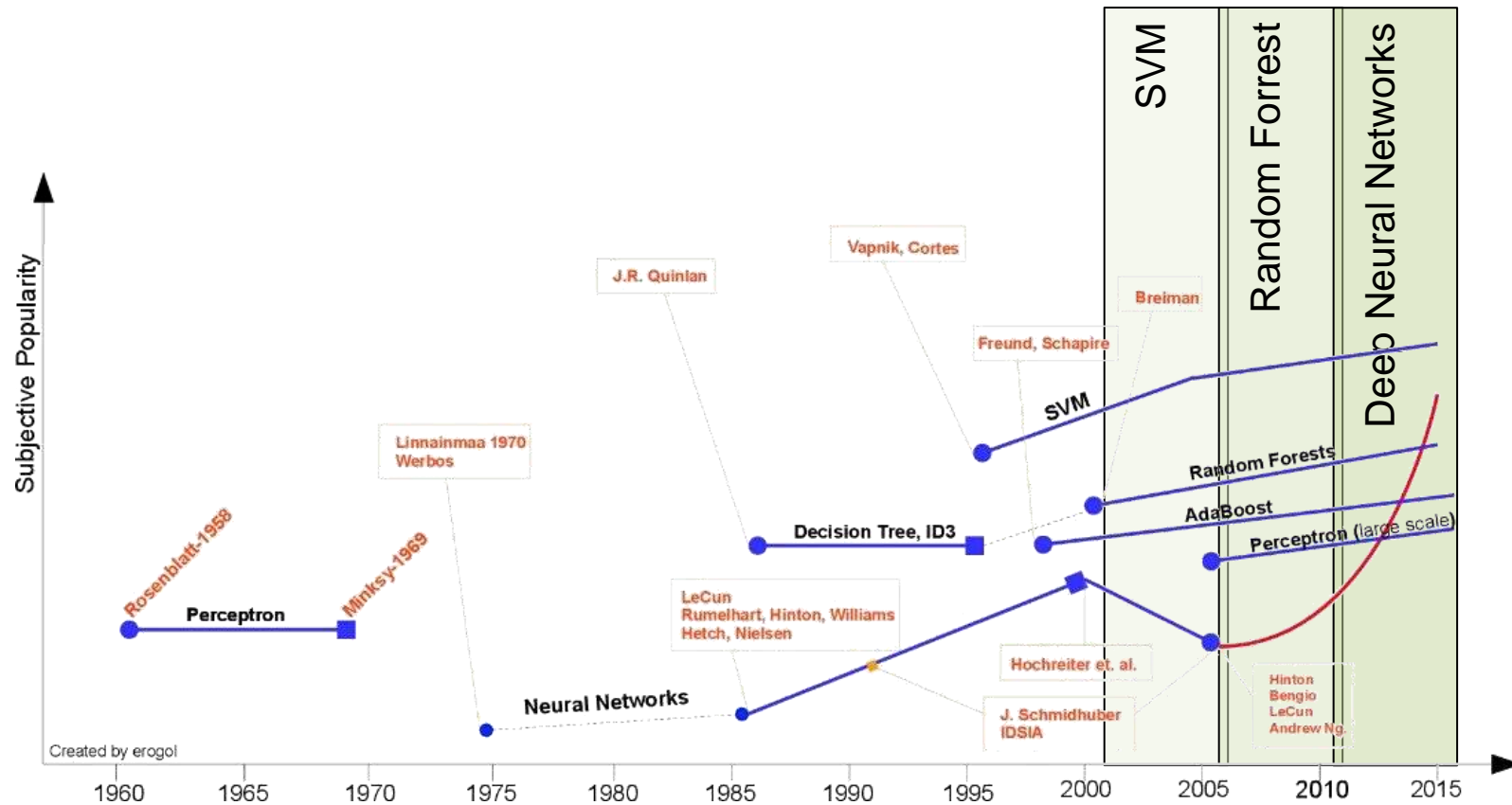
- Get a rough idea what the DL is about
- Understand the computation graph
- Using Tensorflow:
  - Able to build simple computational graphs
  - Feeding and Fetching of computational graphs
- Understand idea of loss functions and the need to optimize them
- Understand gradient decent and how to implement that in TF

# Introduction to Deep Learning

# A simplified history of Machine Learning

- See Thilo this morning

Trends in research-oriented practice  
(subjective view)



Original chart by Diego Marinho de Oliveira, <https://www.linkedin.com/pulse/20141024101110-52688293-brief-history-of-machine-learning>

DL especially successful in machine perception



# Machine Perception

- Computers have been quite bad in things which are easy for humans (images, text, sound)
- A Kaggle contest 2012
- In the following we explain why

[Kaggle dog vs cat competition](#)



Deep Blue beat Kasparov at chess in 1997.  
Watson beat the brightest trivia minds at Jeopardy in 2011.  
Can you tell Fido from Mittens in 2013?

# Traditional Machine Perception

Raw data



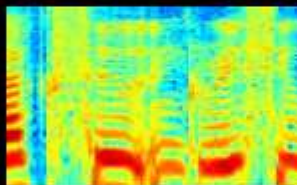
Feature extraction



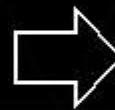
Classifier/  
detector

SVM,  
shallow neural net,  
...

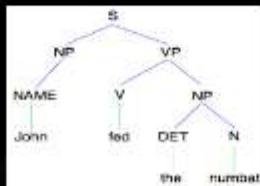
Result



HMM,  
shallow neural net,  
...



Speaker ID,  
speech transcription, ...



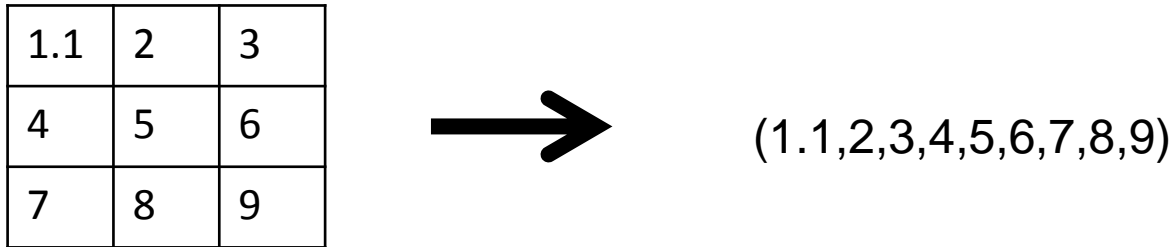
Clustering, HMM,  
LDA, LSA  
...



Topic classification,  
machine translation,  
sentiment analysis...

# Traditional Feature Engineering: on images

- Simplest version: Just unroll the image



- Used for example in Eigenfaces / Fisher-Faces



- What's wrong with this approach?

# How stupid is the machine I



Unaligned

Machine Performance: quite bad



Aligned (fixed eyes)

Machine: OK

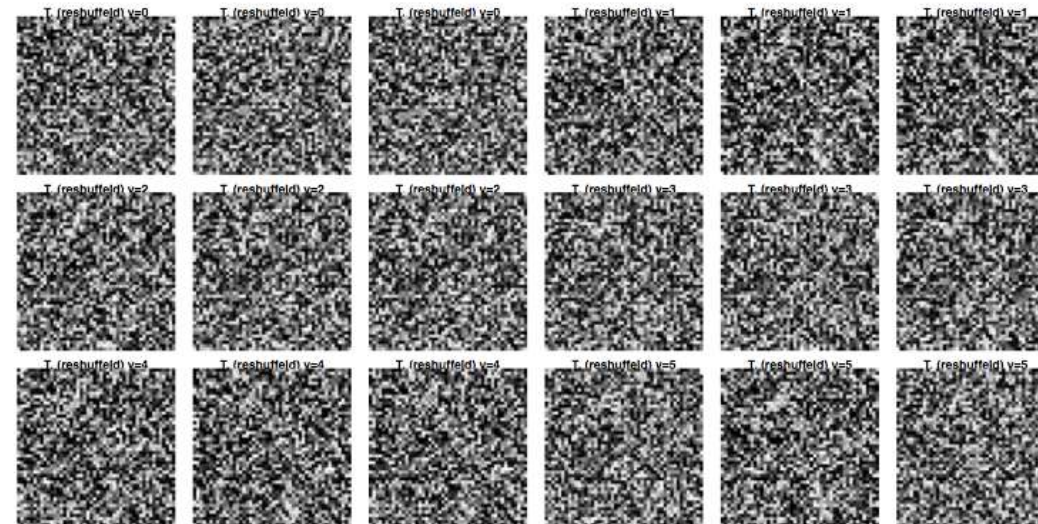
Humans don't care about the alignment



# How stupid is the machine II



Aligned



Aligned & Shuffled

Guess the performance?

	Eigenfaces	Fisher	SVM
Reshuffled	0.7349	0.9163	0.9023
Original	0.7349	0.9163	0.9023

The previous algorithms are not robust against translations and don't care about the locality of the pixels!

# Handcrafted Features for Computer Vision

- Better features need quite some engineering
- Basically success in Computer Vision has been driven by better features
- Hogg, SWIFT, ...
- No hierarchy is learnt
- In the lectures on CNN (starting week 4) we see better approaches

# Traditional feature engineering on text

- The famous bag of words

	aardvark	a	...	Zyxt
doc1	2	4	...	3
doc2	-	4	...	1



[aardvark](#)

Zyxt (obsolete, Kentish)  
second-person singular past  
tense of to see. Notable  
because it's the last word in  
the Oxford English Dictionary.

- **n-grams**
  - counts co-occurrence of a sequence of words
- Do you believe that with this features you can learn to understand language?

The most convincing case for  
DL (subjective view)

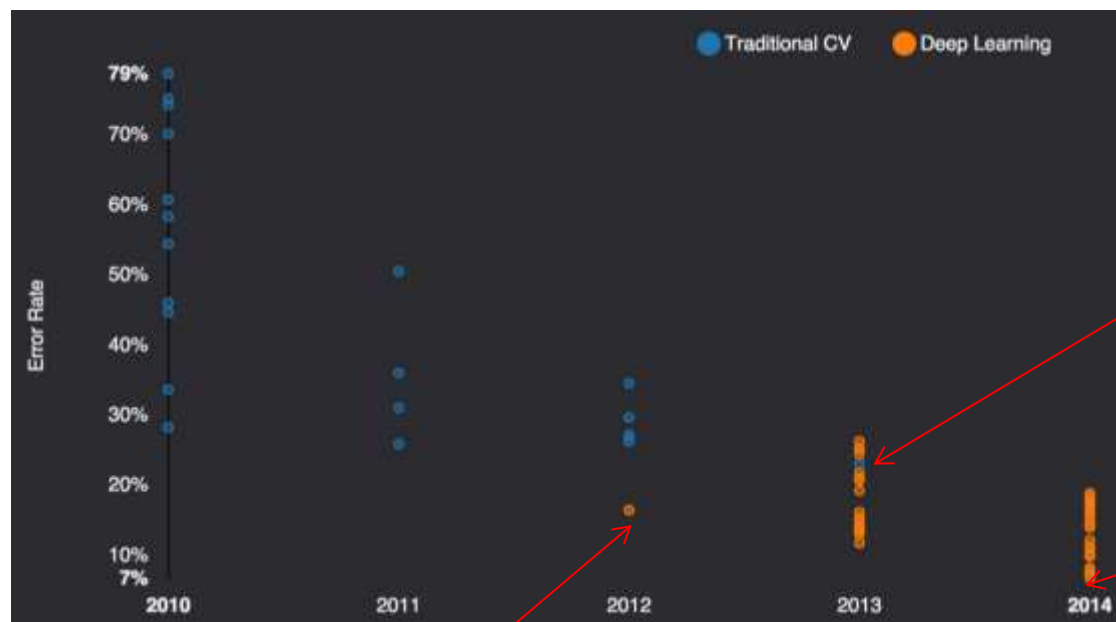


# Why DL: Imagenet 2012, 2013, 2014, 2015

1000 classes  
1 Mio samples



...



Human: 5% misclassification

Only one non-CNN approach in 2013

GoogLeNet 6.7%

A. Krizhevsky  
first CNN in 2012  
**Und es hat zoom gemacht**

2015: It gets tougher

4.95% Microsoft ([Feb 6](#) surpassing human performance 5.1%)

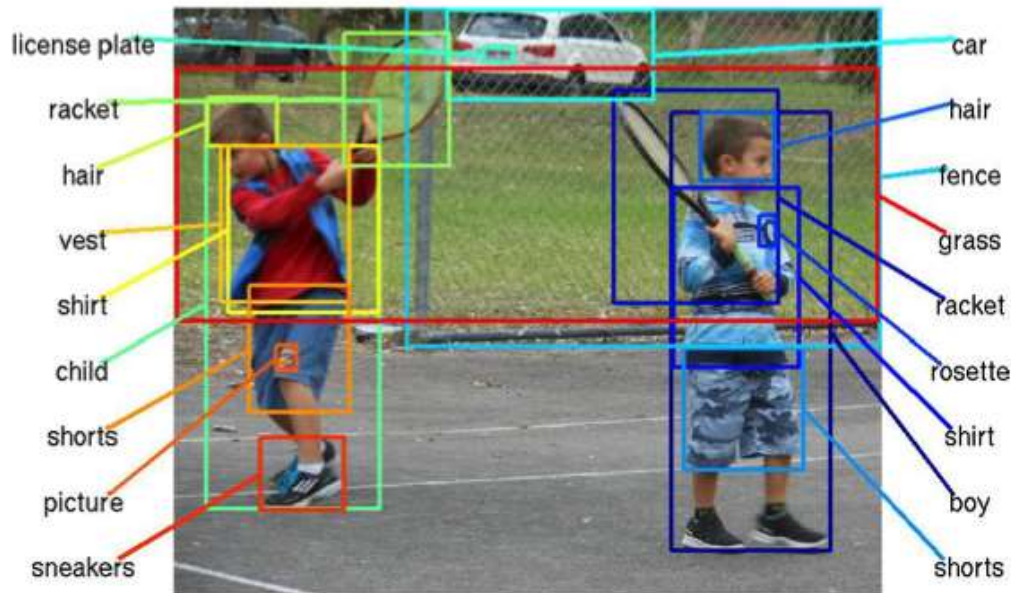
4.8% Google ([Feb 11](#)) -> further improved to 3.6 (Dec)?

4.58% Baidu (May 11 [banned due too many submissions](#))

3.57% Microsoft (Resnet winner 2015)

# The computer vision success story

- With DL it took approx. 3 years to solve object detection and other computer vision task



Deep Blue beat Kasparov at chess in 1997.  
Watson beat the brightest trivia minds at Jeopardy in 2011.  
Can you tell Fido from Mittens in 2013?



"man in black shirt is playing  
guitar."

# The importance of seeing: the Cambrian explosion

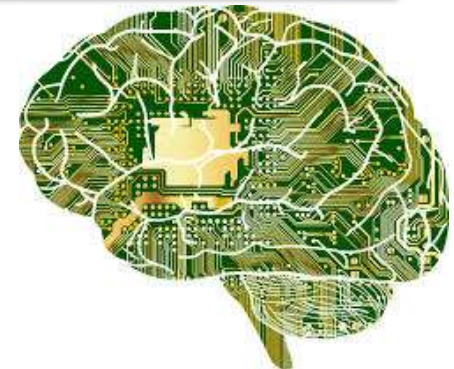


In a short period of time ~20 mio yrs (~540 mio yrs ago) most animal groups appeared.

So extraordinary fast that this is taken as an explanation intelligent design.


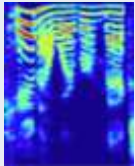
A simpler explanation  
**creatures learned to see**

A more liable equivalence of  
deep learning than ...





# Selection of tasks solved by deep learning

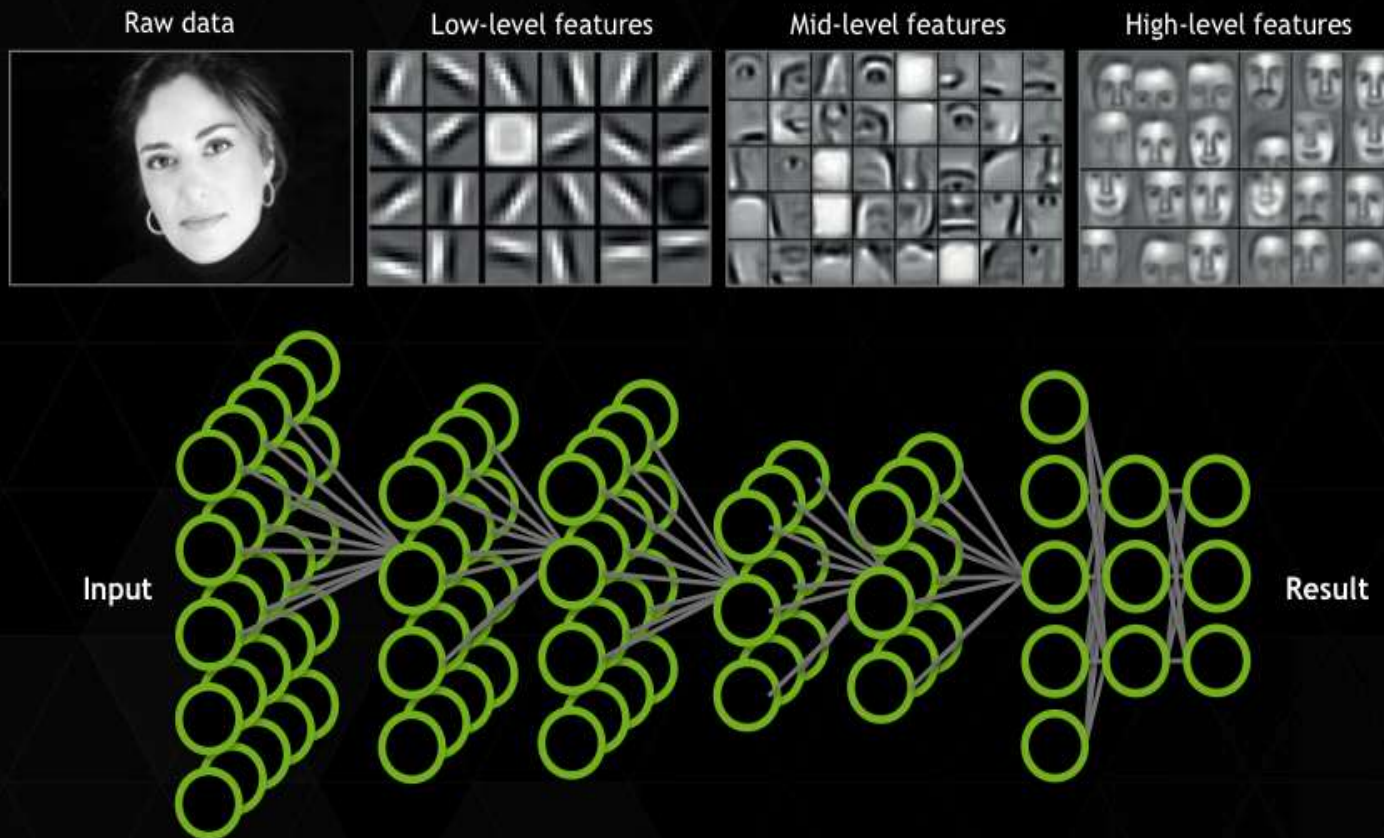
Input		Output	
Images		Label "Tiger"	Image classification
Audio		Sequence / Text "see you tomorrow"	Voice Recognition
Sequences Hallo, wie gehts?		Sequences Hello how are you?	Translation
Sequence This movie was rather good		Label (Sentiment) positive	Sentiment Analysis
Input context city='london', device='mobile'		P("user clicks on add")	Click prediction



Deep  
Reinforcement  
Learning



## DEEP NEURAL NETWORK (DNN)



**Application components:**

**Task objective**  
e.g. Identify face

**Training data**  
10-100M images

**Network architecture**  
~10 layers

1B parameters

**Learning algorithm**  
~30 Exaflops  
~30 GPU days

# Research Topics



Basic Building Blocks  
of modern DL-Architectures

Convolutional Architectures (CNNs)

CNN 1980  
Fukushima

LeNet 1998  
Yann LeCun

German Traffic Sign 2011  
Ciresan, Schmidhuber

ImageNet  
AlexNet  
Krizhevsky, Hinton

DeepFace

VGG16

DeepDream

Artstyle Transf

Inception

ResNet

2012 — 2013 — 2014 — 2015 — 2016

Other Breakthroughs / Architectures  
Subjective Selection

Reinforcement Learning:

DeepQ

AlphaGO

Partly CNN:

Auto. Captions

Draw

LSTM 1997  
Hochreiter, Schmidhuber

Unsupervised  
Pre-training  
DNN 2006

FC 1986  
Rumelhart,...

Adagrad

Dropout

BatchNorm

2012 — 2013 — 2014 — 2015 — 2016

Generative Models:

VAE

GAN

Hot  
in ML

Neural Networks

SVM

Bagging / Boosting

Deep Learning

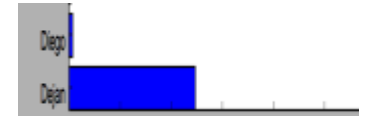
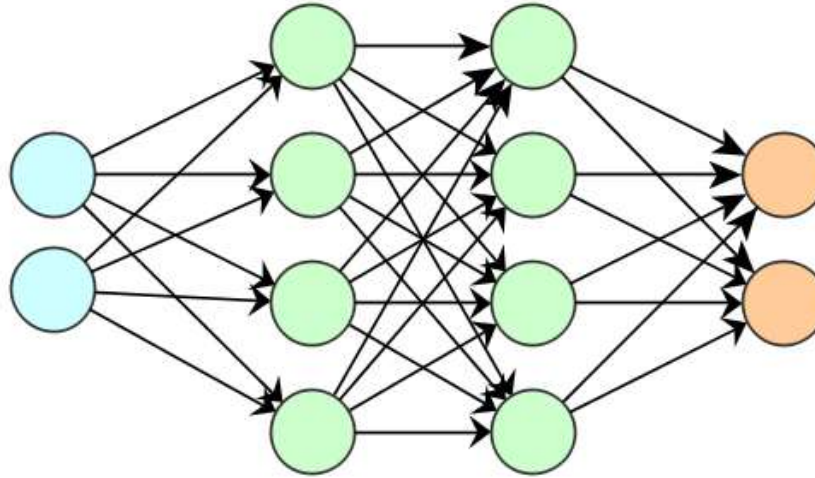
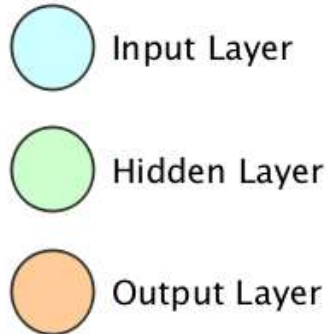
# What's typical deep learning?

- Data (basically raw)
  - Little / no feature engineering
  - A hierarchy of increasingly complex features is learnt
- Architecture (deep)
  - Composition of simple building blocks “*layers*”
    - Fully Connected, Softmax, Convolutional, Max-Pooling, LSTM-Cells,...
- Training (long)
  - **A loss function is minimized** (supervised, weakly supervised, unsupervised)



Recent success mainly due to **more data** and **more compute** power (GPU). The building blocks where there for years.

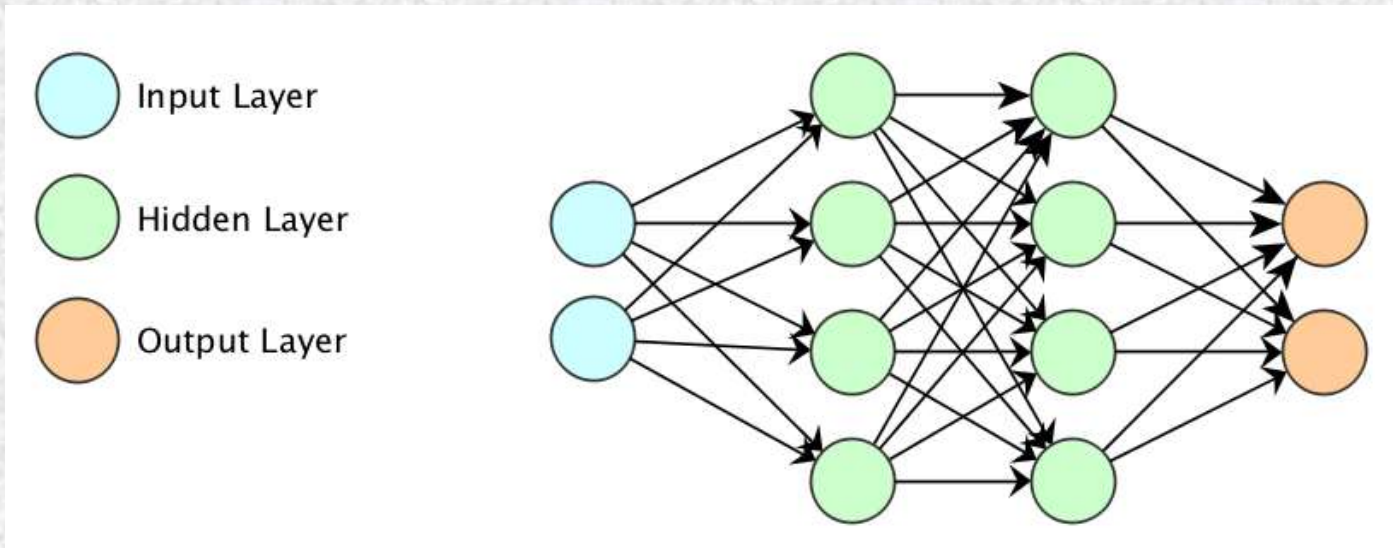
# Preview: The first network



- The input: e.g. intensity values of pixels of an image
- Information is processed layer by layer
- Output: probability that image belongs to certain person
- Arrows are weights (these need to be learned)

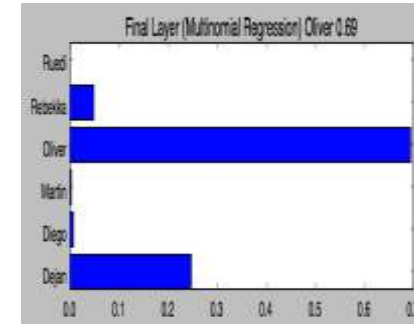
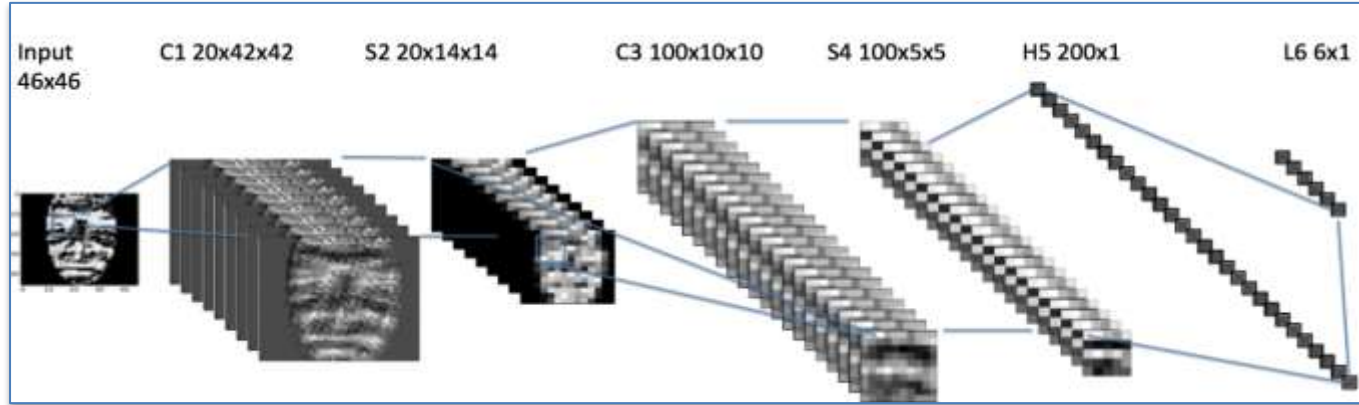


# Preview: The first network



- Start (1,2) →
- Information flow through the graph

# Preview: Convolutional Neural Network (CNN)



- The input: e.g. intensity values are arrays (x,y)
- Inner layers: (x,y,z)
- Output: probability that image belongs to certain person

# Deep Learning Frameworks

# Deep Learning Frameworks (common)

- Computation needs to be done on GPU or specialized hardware (compute performance)
- On GPU: almost exclusively on NVIDIA using the cuda library, cudnn
- Data Structure are multidimensional arrays (*tensors*) which are manipulated
- Learning require to calculate derivatives of the network w.r.t parameters.
- Calculation is done using a *computational graph*. To approaches:
  - Building graph on the fly (chainer, pytorch)
  - 2 step procedure (1<sup>st</sup> build graph, 2<sup>nd</sup> execute graph) TensorFlow
- Nearly all libraries use python, some support other languages
- There are (too) many frameworks, it's a bit like vim vs emacs

# Software for Deep Learning (low level libraries)

- Torch / pytorch
  - Facebook, quite flexible, lua (Jan 2017 also in python, **pytorch**)
- Chainer
  - Flexible, build graph on the fly
- MXNet
  - Can be used in many languages, build graph on the fly?
- Caffe
  - Inflexible, good of CV
  - Calculate gradients by hand
- Theano
  - Around since 2008,
  - Active development abandoned
  - Slow compiling of graph (due to optimization)
  - Some nice libraries on top: Lasagne and Keras
- TensorFlow
  - Open sourced by Google Dec 2015
  - Do symbolic calculations of gradients (autodiff)

For the this CAS we pick TensorFlow since currently it's mostly used.

# Libraries on top of TensorFlow

- There are lots of libraries on top of TensorFlow. Some of them are in the *tensorflow.contrib* package and are thus installed with TensorFlow
  - Keras
    - Keras is now part of TF core
  - TF-Slim
    - nice to build networks
  - ...

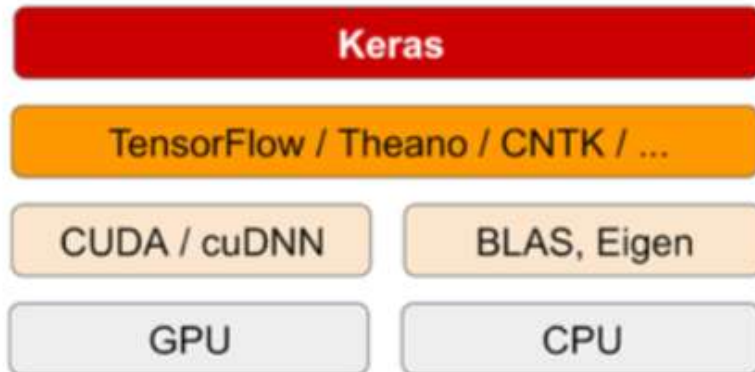


Figure: From Deep Learning with Python, Francois Chollett

# Popularity of Deep Learning Frameworks Oct, 2017

Library	Rank	Overall	Github	Stack Overflow	Google Results
tensorflow	1	10.87	4.25	4.37	2.24
keras	2	1.93	0.61	0.83	0.48
caffe	3	1.86	1.00	0.30	0.55
theano	4	0.76	-0.16	0.36	0.55
pytorch	5	0.48	-0.20	-0.30	0.98
sonnet	6	0.43	-0.33	-0.36	1.12
mxnet	7	0.10	0.12	-0.31	0.28
torch	8	0.01	-0.15	-0.01	0.17
cntk	9	-0.02	0.10	-0.28	0.17
dlib	10	-0.60	-0.40	-0.22	0.02
caffe2	11	-0.67	-0.27	-0.36	-0.04
chainer	12	-0.70	-0.40	-0.23	-0.07
paddlepaddle	13	-0.83	-0.27	-0.37	-0.20
deeplearning4j	14	-0.89	-0.06	-0.32	-0.51
lasagne	15	-1.11	-0.38	-0.29	-0.44
bigdl	16	-1.13	-0.46	-0.37	-0.30
dynet	17	-1.25	-0.47	-0.37	-0.42
apache singa	18	-1.34	-0.50	-0.37	-0.47
nvidia digits	19	-1.39	-0.41	-0.35	-0.64
matconvnet	20	-1.41	-0.49	-0.35	-0.58
tflearn	21	-1.45	-0.23	-0.28	-0.94
nervana neon	22	-1.65	-0.39	-0.37	-0.89
opennn	23	-1.97	-0.53	-0.37	-1.07

1 means one standard deviation above average

Github (stars and forks)

Stack Overflow (tags and questions)

Google Results (total and quarterly growth rate)

pytorch is only one year old!

In this course we will start with low level tensorflow and then will use keras.

# Some Facts about TensorFlow

- Open sourced 9<sup>th</sup> Nov. 2015
- Runs on a variety of platforms

Automatically runs models on range of platforms:

from **phones** ...



to **single machines** (CPU and/or GPUs) ...



to **distributed systems** of many 100s of GPU cards



**TPUs**

Custom machine learning ASIC



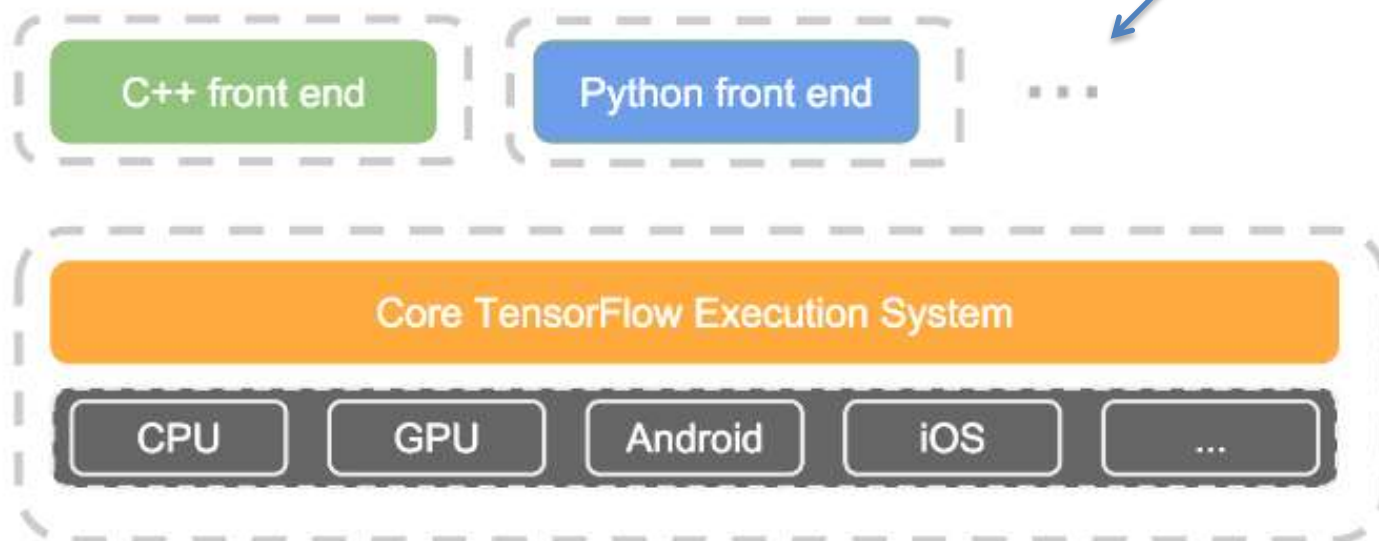


# Some Facts about TensorFlow

## Front Ends

- Different front ends for specifying/driving the computation
  - Python and C++ today, easy to add more

and **R!**



# Myth booster: Tensorflow is not (only) in python!

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow')
sess = tf.Session()
sess.run(hello)
```

```
library(tensorflow)
sess <- tf$Session()
hello <- tf$constant('Hello, TensorFlow')
sess$run(hello)
```

⬆️ [-] **PM\_ME\_ELLEN\_PAO** 15 points 1 month ago

```
library(tensorflow)
sess = tf$Session()
hello <- tf$constant('Hello, TensorFlow!')
sess$run(hello)
W <- tf$Variable(tf$zeros(shape(784L, 10L)))
b <- tf$Variable(tf$zeros(shape(10L)))
```

My eyes! It burns!

# Overview (next 2 days)

- For the first two days we use plain vanilla TensorFlow to get an idea.
- Theoretical / Intuitive Understanding of Deep Learning
  - **The loss function**
  - **Linear regression as the mother of all networks**
  - (Multinomial) logistic regression for classification
  - Fully Connected Network
  - Regularization
  - Backpropagation and gradient flow
- Practical Deep Learning (includes understanding of frameworks)
  - **Tensors and the compute graph**
  - **Inspecting the graph (loading frozen network)**

# What is TensorFlow

- It's API about **tensors**, which **flow** in a **computational graph**



<https://www.tensorflow.org/>

- What is a computational graph? But first..
- What are **tensors**?

# What is a tensor?

In this course we only need the simple and easy accessible definition of Ricci:

**Definition.** A tensor of type  $(p, q)$  is an assignment of a multidimensional array

$$T_{j_1 \dots j_q}^{i_1 \dots i_p} [\mathbf{f}]$$

to each basis  $\mathbf{f} = (\mathbf{e}_1, \dots, \mathbf{e}_n)$  of a fixed  $n$ -dimensional vector space such that, if we apply the change of basis

$$\mathbf{f} \mapsto \mathbf{f} \cdot R = (\mathbf{e}_i R_1^i, \dots, \mathbf{e}_i R_q^i)$$

**Just kidding...**

then the multidimensional array obeys the transformation law

$$T_{j'_1 \dots j'_q}^{i'_1 \dots i'_p} [\mathbf{f} \cdot R] = (R^{-1})_{i_1}^{i'_1} \dots (R^{-1})_{i_p}^{i'_p} T_{j_1 \dots j_q}^{i_1 \dots i_p} [\mathbf{f}] R_{j'_1}^{j_1} \dots R_{j'_q}^{j_q}.$$

Sharpe, R. W. (1997). Differential Geometry: Cartan's Generalization of Klein's Erlangen Program. Berlin, New York: Springer-Verlag. p. 194. ISBN 978-0-387-94732-7.

# What is a tensor?

For TensorFlow: A tensor is an array with several indices (like in numpy). Order (a.k.a rank) are number of indices and shape is the range.

```
In [1]: import numpy as np
```

```
In [2]: T1 = np.asarray([1,2,3]) #Tensor of order 1 aka Vector  
T1
```

```
Out[2]: array([1, 2, 3])
```

```
In [3]: T2 = np.asarray([[1,2,3],[4,5,6]]) #Tensor of order 2 aka Matrix  
T2
```

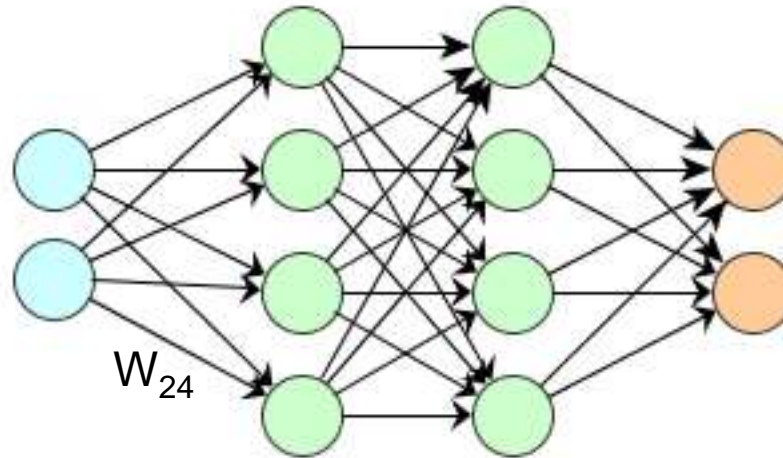
```
Out[3]: array([[1, 2, 3],  
              [4, 5, 6]])
```

```
In [4]: T3 = np.zeros((10,2,3)) #Tensor of order 3 (Volume like objects)
```

```
In [6]: print(T1.shape)  
print(T2.shape)  
print(T3.shape)
```

```
(3,)  
(2, 3)  
(10, 2, 3)
```

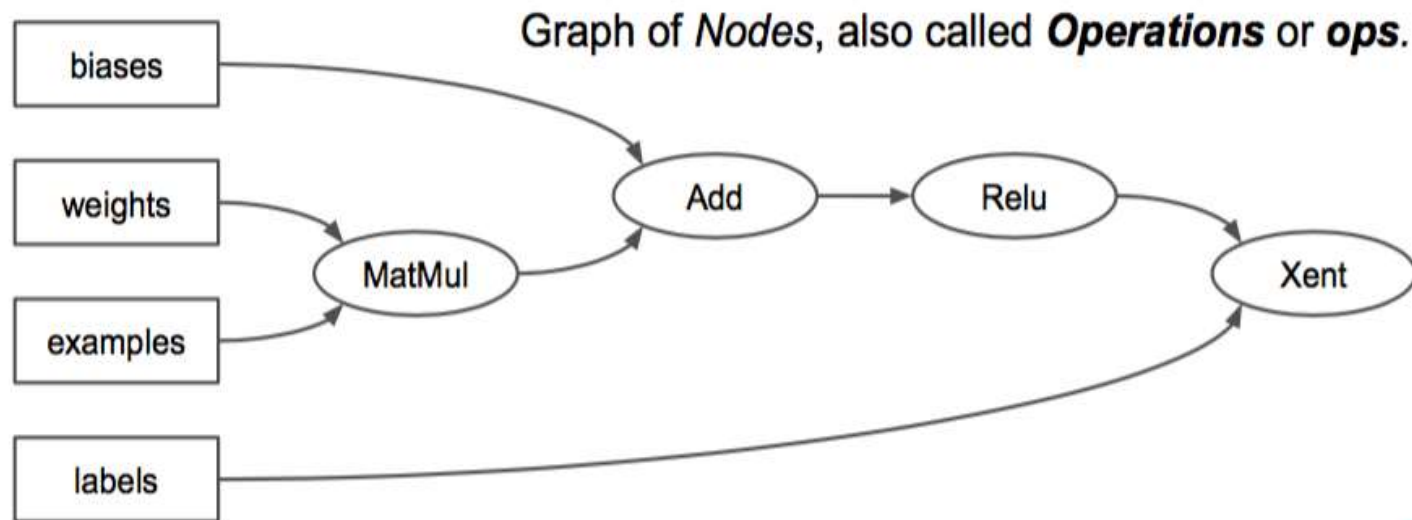
# Typical Tensors in Deep Learning



- The input can be understood as a vector
- The weights going from e.g. Layer  $L_1$  to Layer  $L_2$  can be written as a matrix (often called  $W$ )
- A mini-batch of size 64 of input vectors can be understood as tensor of order 2
  - (index in batch,  $x_i$ )
- A mini-batch of size 64 images with 256,256 pixels and 3 color-channels can be understood as a tensor of order 4.

# Computations in TensorFlow (and Theano)

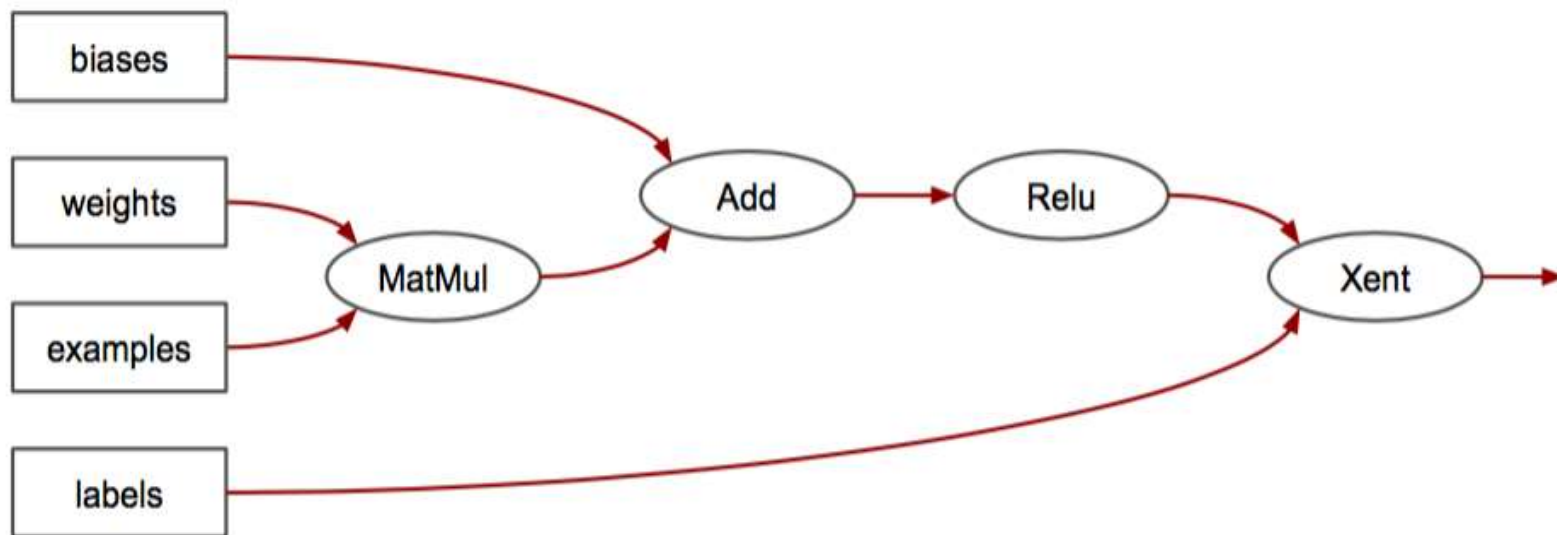
- Computation is expressed as a dataflow graph





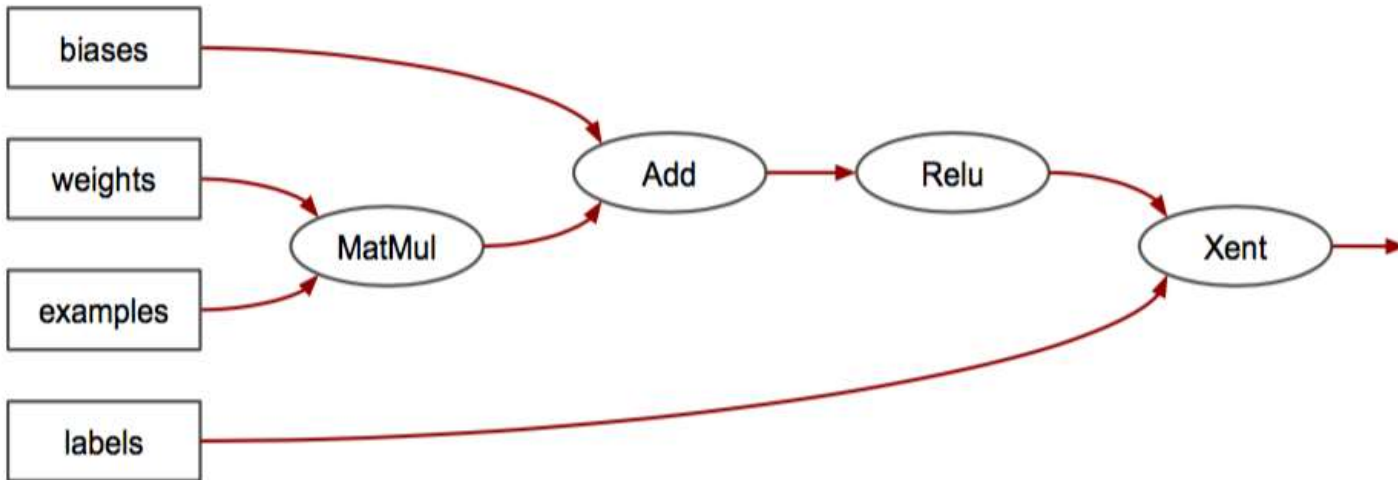
# Computations in TensorFlow (and Theano)

- Edges are N-dimensional Arrays: Tensors



# Summary

- The computation in TF is done via a computational graph



- The nodes are ops
- The edges are the flowing tensors

# TensorFlow: Computation in 2 steps

- Computations are **done in 2 steps**
  - **First:** Build the graph
  - **Second:** Execute the graph
- Both steps can be done in many languages (python, C++, R, Java)
- Best supported so far is python

## Recap Matrix Multiplication (scalar and with vector)

$$10 \begin{pmatrix} 3 & 3 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \end{pmatrix} = 120$$

What happens, if we add a second colon to matrix with values (1,4)?

# Building the graph (python)

$$10 \begin{pmatrix} 3 & 3 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \end{pmatrix} = 120$$

In [1]: numpy

```
import numpy as np
m1 = np.array([[3., 3.]])
m2 = np.array([[2.],[2.]])
10 * np.dot(m1,m2)
```

Out[1]:

```
array([[ 120.]])
```

# Be the spider who knits a computational graph



Translate the following TF code in a graph

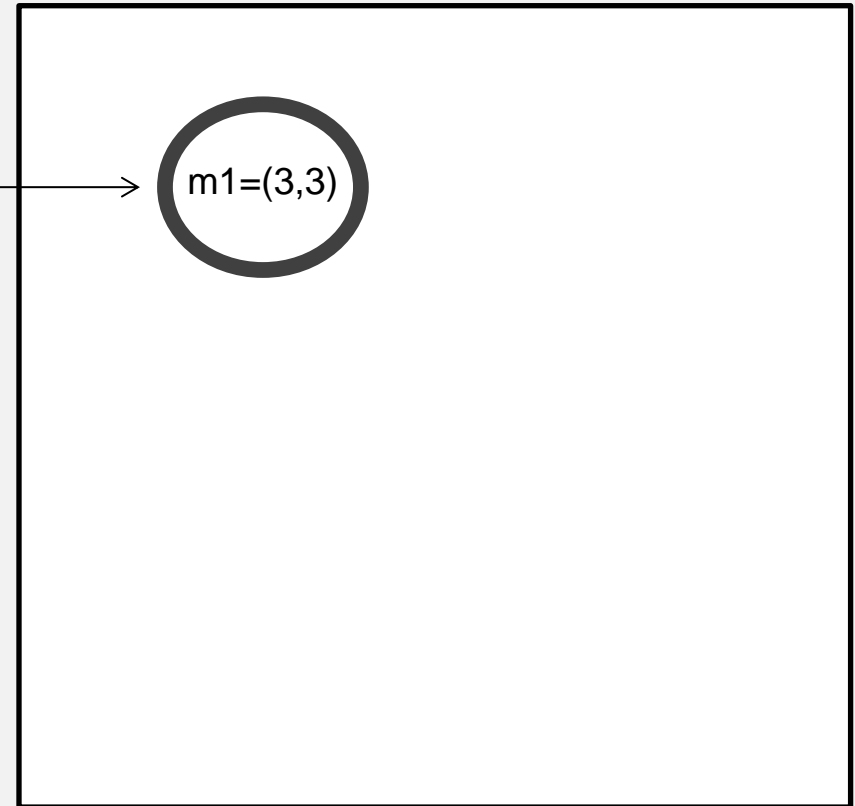
TensorFlow: Building the graph

```
import tensorflow as tf
# We construct a graph (we write to the default graph)
# make first sure the default graph is empty
tf.reset_default_graph()
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

wipes the graph

Quite much happen in here!

Finish the computation graph



# Be the spider who knits the computational graph

Translate the following TF code in a graph

TensorFlow: Building the graph

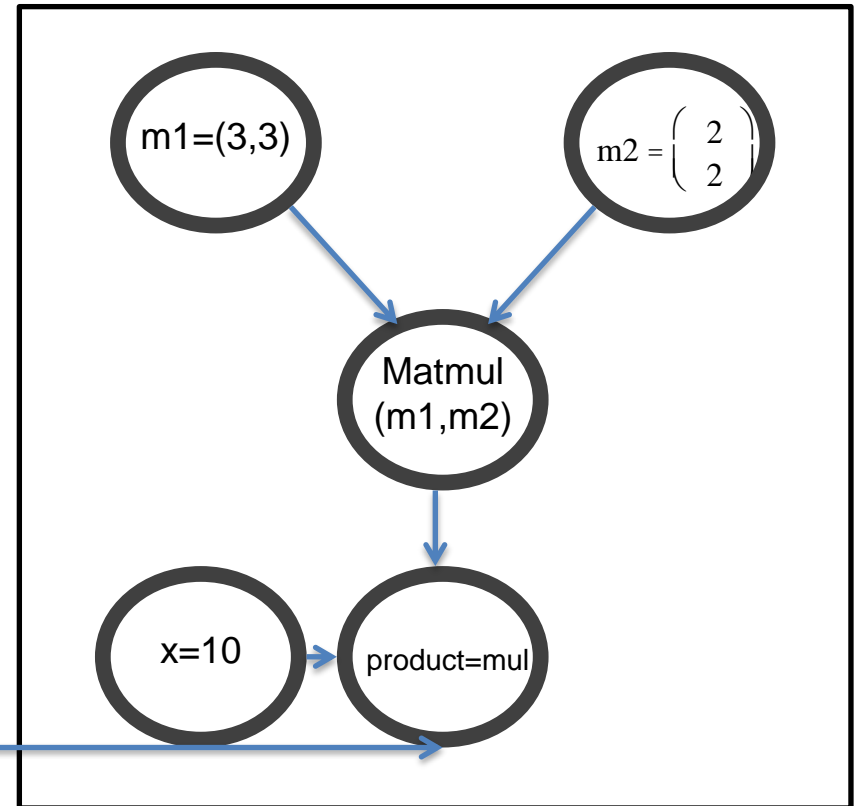
```
import tensorflow as tf
# We construct a graph (we write to the default graph)
# make first sure the default graph is empty
tf.reset_default_graph()
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

TensorFlow: Executing the graph

```
In [4]:
sess = tf.Session()
res = sess.run(product)
print(res)
sess.close()

[[ 120.]]
```

Finish the computation graph





# Building the graph:

## How it looks in Tensorflow

$$10 \begin{pmatrix} 3 & 3 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \end{pmatrix} = 120$$

In [1]: numpy

```
import numpy as np
m1 = np.array([[3., 3.]])
m2 = np.array([[2.],[2.]])
10 * np.dot(m1,m2)
```

Out[1]:  
array([[ 120.]])

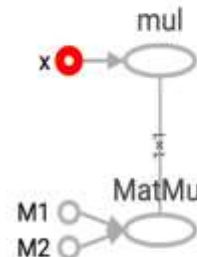
## TensorFlow: Building the graph

```
import tensorflow as tf
# We construct a graph (we write to the default graph)
# make first sure the default graph is empty
tf.reset_default_graph()
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

In [4]: TensorFlow: Executing the graph

```
sess = tf.Session()
res = sess.run(product)
print(res)
sess.close()
```

[[ 120.]]



**mul/x** ^

Operation: ☐

Const

**Attributes (2)**

dtype {"type":"DT\_FLOAT"}

value {"tensor": {"dtype":"DT\_FLOAT","tensor\_shape":{"shape":{},"float\_val":10}}}

**Inputs (0)**

**Outputs (0)**

Remove from main graph

# Demo of docker and jupyter notebooks

- Demo of docker

- `docker run -p 8888:8888 -p 6006:6006 -it oduerr/tf_docker:cpu_r`

- Running docker with access to local directory

- `docker run -p 8888:8888 -p 6006:6006 -v /Users/oli/Documents/workspace/dl_course/:/notebooks/ -it oduerr/tf_docker:tf1_py3`

- Do you know what ipython/jupyter notebooks are?

- Small demo?

- Simple commands
    - Up and downloading notebooks

# First Task with TensorFlow



Website: [https://github.com/tensorchiefs/dl\\_course\\_2018/](https://github.com/tensorchiefs/dl_course_2018/)

## Matrix Multiplication in TensorFlow

- a) Open the notebook MatrixMultiplication and visualize the computational graph in TensorBoard

Hint: run all cells in the jupyter notebook and then open a terminal and type:  
`tensorboard --logdir=path/to/log-directory` or run `tensorboard` from the notebook with: `!tensorboard --logdir=path/to/log-directory`

- b) Change the code in the notebook that it divides the matrix multiplication by 10 instead of multiplying it with 10. Compare the numpy and tensorflow results. What do you observe?

Additional Task [If time] do a matrix addition

Do not do the part c): Matrix Multiplication Feeding yet

# Session vs Graph

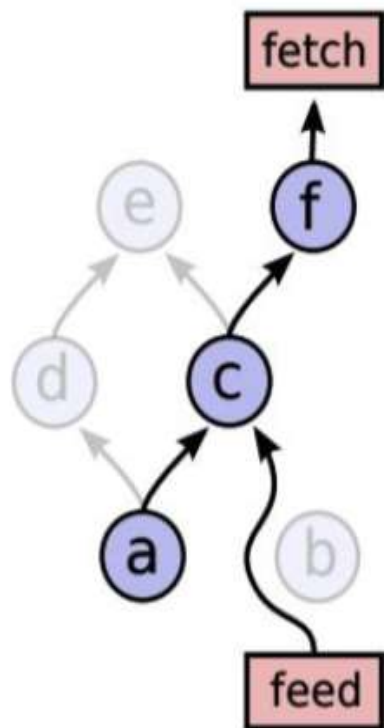
- A graph is the abstract definition of the calculation
- A session **is a concrete realization**
  - It places the ops on physical devices such as GPUs
  - It initializes variables
  - We can feed and fetch a session (see next slides)

```
sess = tf.Session()  
... #do stuff  
sess.close() #Free the resources (TF eats all mem on GPU!)
```

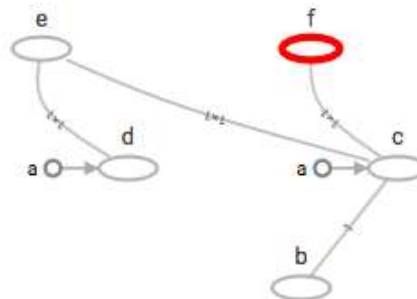
Alternatively use the with construct

```
with tf.Session() as sess:  
    ... #do stuff  
#Free the resources when leaving the scope of with
```

# Computations using feeding and fetching



```
a = tf.constant([[1]], name='a')
b = tf.placeholder(dtype='int32', shape=[1], name='b')
d = tf.identity(a, name='d')
c = tf.multiply(a, b, name='c')
e = tf.multiply(d, c, name='e')
f = tf.identity(c, name='f')
```



```
res = sess.run(f, feed_dict={b:[2]})
```

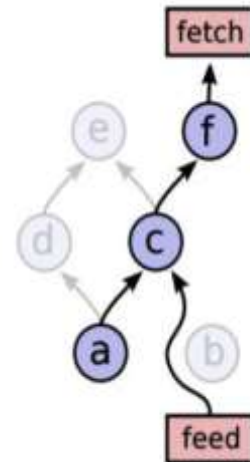
↑  
fetch  
(the numeric value)

↑  
Fetch  
f (symbolic)

↑  
symbolic

↑  
values

# Feed and Fetch



- Fetches can be a list of tensors
- Feed (from TF docu)
  - A feed temporarily replaces the **output of an operation** with a tensor value. You supply feed data as an argument to a `run()` call. The feed is only used for the run call to which it is passed. The most common use case involves designating specific operations to be “feed” operations by using `tf.placeholder()` to create them.

```
b = tf.placeholder(tf.float32, shape=(10, 10))
...
res = sess.run(f, feed_dict={b:data[0:10,0:10]})
```

## A more general example

```
x = tf.placeholder(tf.float32, shape=(1024, 1024))
res1, res2 = sess.run([loss, loss2], feed_dict={x:data[:,0], y:data[:,1]})
```

↑  
fetches  
(the numeric values)

↑  
fetches  
(symbolic)

↑    ↑  
two inputs (feeds)

# The loss function



# Tuning a neural network

- Neural networks are models which have parameters
- We have (training  $i = 1 \dots N_{\text{training}}$ ) data in pairs  $x^{(i)}$  and  $y^{(i)}$

$x^{(i)} \rightarrow \text{model parametrized with weights} \rightarrow \hat{y}^{(i)}$

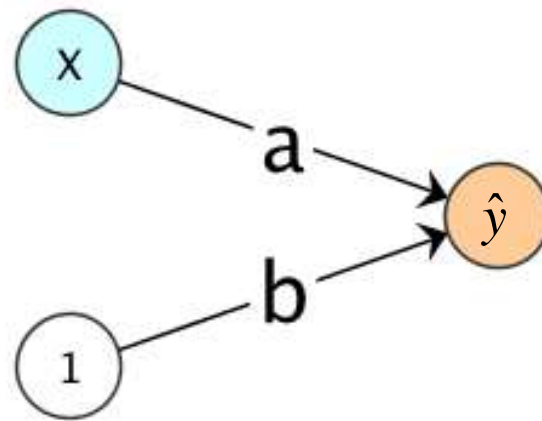
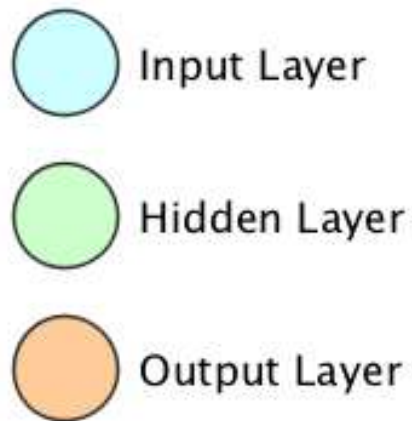


Depending on the weight the model produces a different  $\hat{y}^{(i)}$

- Examples (your task what are the x's what are the y'?)
  - Facerec.: Faces and Names
  - Age Prediction: Faces and Age (numerical problem)
- How to tune the knobs that the output of the model  $\hat{y}^{(i)}$  matches the “true” value  $y^{(i)}$ ? We optimize a loss.
- To understand the principle, we start with something dead simple: good old linear regression

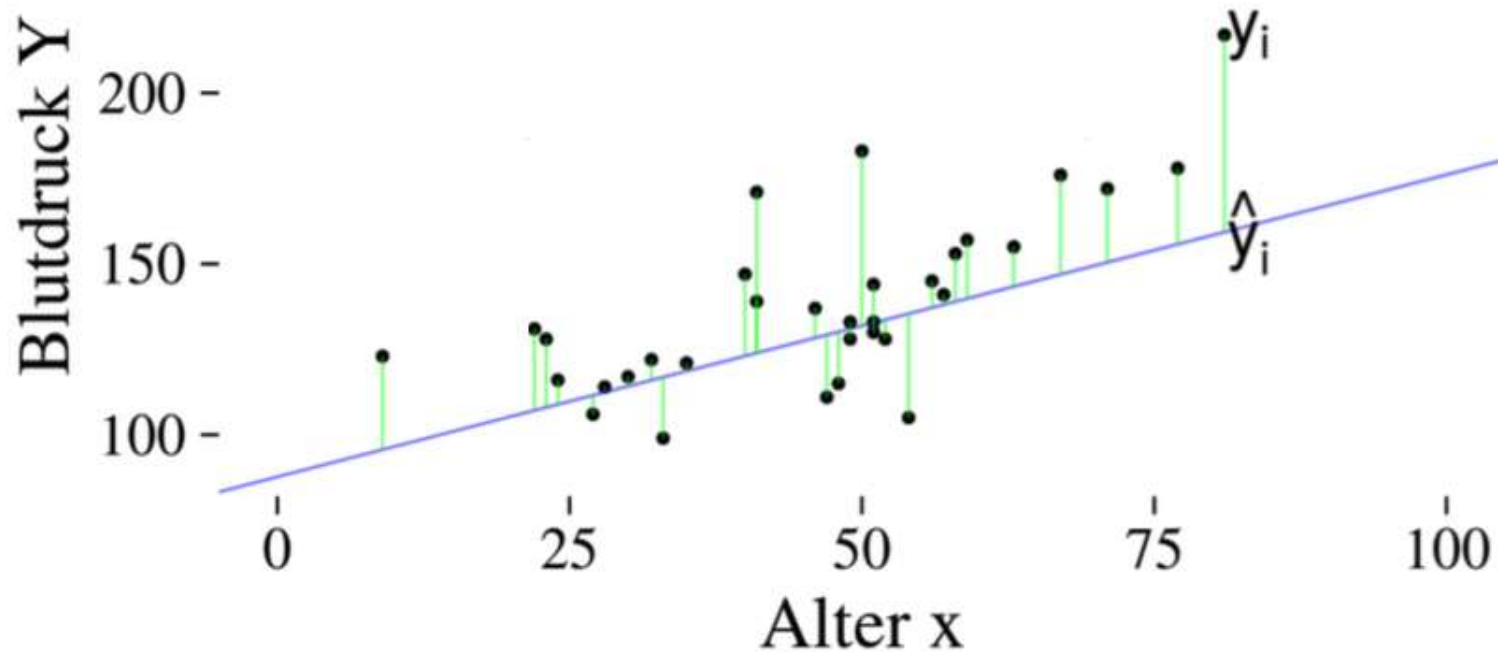
# Linear Regression

- Data  $(x^{(i)}, y^{(i)})$
- Model  $\hat{y}^{(i)} = ax^{(i)} + b$
- Parameters  $a, b$
- In neural network speak



# Loss for linear regression: sums of squared error

$$RSS(a, b) = \sum_{i=1}^n (y_i - \underbrace{(a \cdot x_i + b)}_{=: \hat{y}_i})^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$



a

b



Tune a,b until RSS or 1/N RSS is minimal.

# TF: Variables, Constants, Placeholders

The name appears in the graph

```
# Constant
```

```
pi = tf.constant(3.141, tf.float32, name='crude_pi')
```

```
# Allows to feed in external input. In DL typically x,y
```

```
x = tf.placeholder('float32', [N], name='x_data')
```

```
# Variable
```

```
a = tf.Variable(1.0, name = 'a')
```

```
# Is initialized with value 1, but this value can change
```

```
# Could also be a random initialization of 5,5 Matrix
```

```
w = tf.Variable(tf.random_normal([5, 5]),name="w")
```

```
...
```

```
# Variables need to be initialized
```

```
init_op = tf.global_variables_initializer() #Adds init-stuff
```

```
#Initialized the variables (e.g. draw rnd numbers)
```

```
sess.run(init_op)
```

# Feeding and Fetching the graph



## Matrix Multiplication in TensorFlow (Rest)

c) Now use a placeholder for m2 to feed-in values. You must specify the shape of the m2 matrix (rows, columns).

## Linear regression in TensorFlow

a) Open the notebook Linreg\_with\_slider and run the first 4 cells and try to minimize the loss by adjusting the parameters a and b.

b) Run the next two cells and feed your adjusted parameters through the graph. You have to modify cell 6 a bit.

Do not do c, d)