Machine Intelligence:: Deep Learning
Week 1

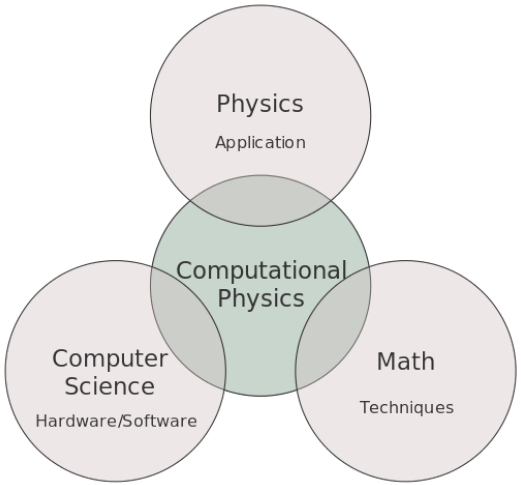
Oliver Dürr

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

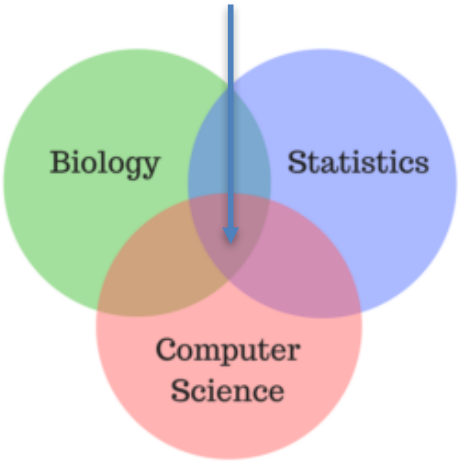
Winterthur, 19. Feb. 2019

Me: From Computational Physics → Data Science

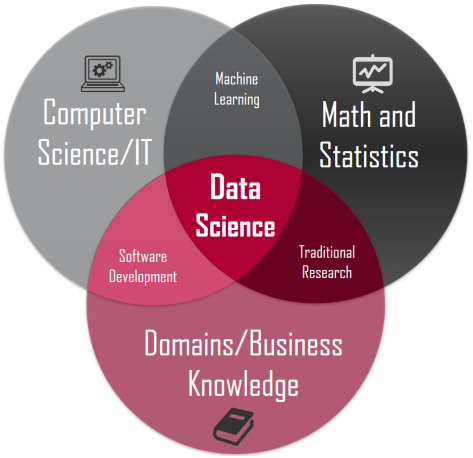
Computational Physics



Bioinformatics



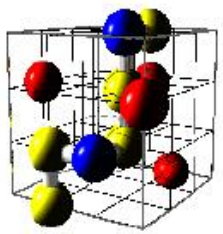
Data Science

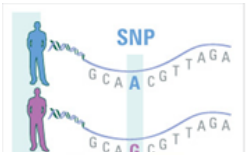


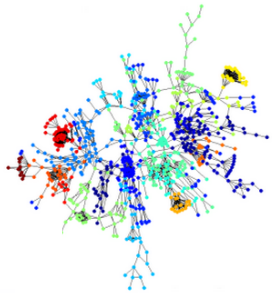
1990's
Uni-Konstanz

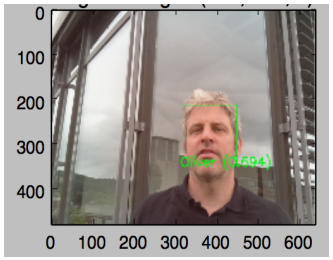
2000's
Genedata Basel

2010's
ZHAW Winterthur HTWG Konstanz









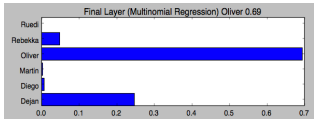


Image credits: Wikipedia, http://omgenomics.com/what-is-bioinformatics/ https://www.fox.temple.edu/institutes-and-centers/data-science/

Tensorchiefs

$$F^{\mu\nu}$$

- Jointly organize
  - The CAS Deep Learning Course
    - https://tensorchiefs.github.io/dl_course_2018/
  - The Deep Learning Workshops “Deep Learning Day”
    - https://tensorchiefs.github.io/dlday2018/
    - https://tensorchiefs.github.io/dlday2017/
    - https://sites.google.com/site/sdsdlday2016/
- Currently write a book about deep learning (from a “statistical” viewpoint)
  - https://tensorchiefs.github.io/dl_book/ (Notebooks for the book)

Before we start (check technical details)

- Website (same url but not content)
  - 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
    - ...

Organizational Issues: Times

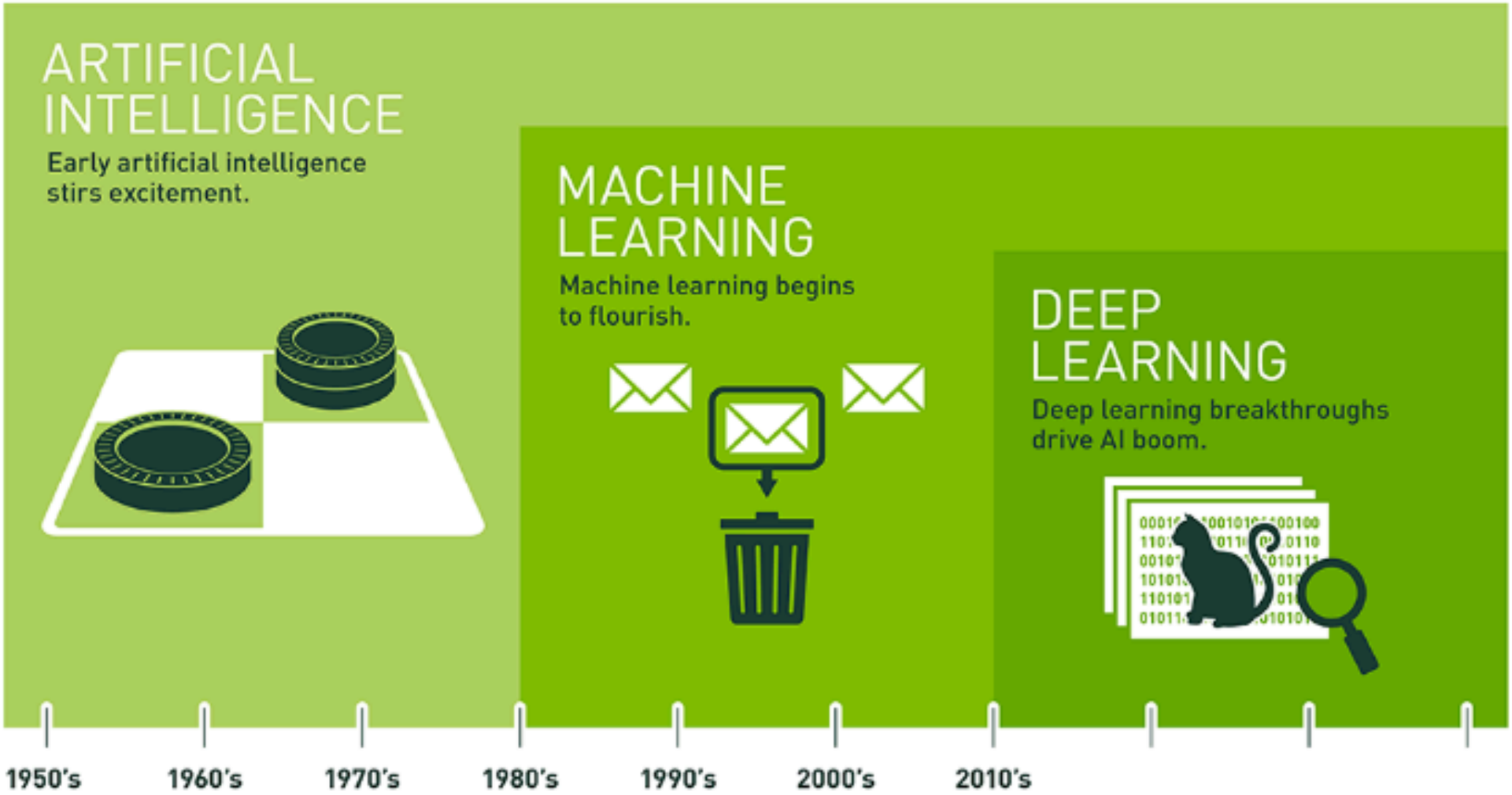
- Dates and times: see our webpage
- First 3 times
  - 13:30 – 15:00
  - 15:30 – 17:00
- Theory and exercises will be mixed
  - Could be 10 minutes theory 30 minutes exercises
  - Could be vice versa
- **Please interrupt us if something is unclear! The less we talk the better!**

Learning Objectives for today

- Get a rough idea what the DL is about
- Understand the computational graph
- Using Tensorflow:
  - Able to build simple computational graphs
  - Feeding and Fetching of computational graphs
- Possible
  - 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

KI (AI), Machine Learning, Deep Learning

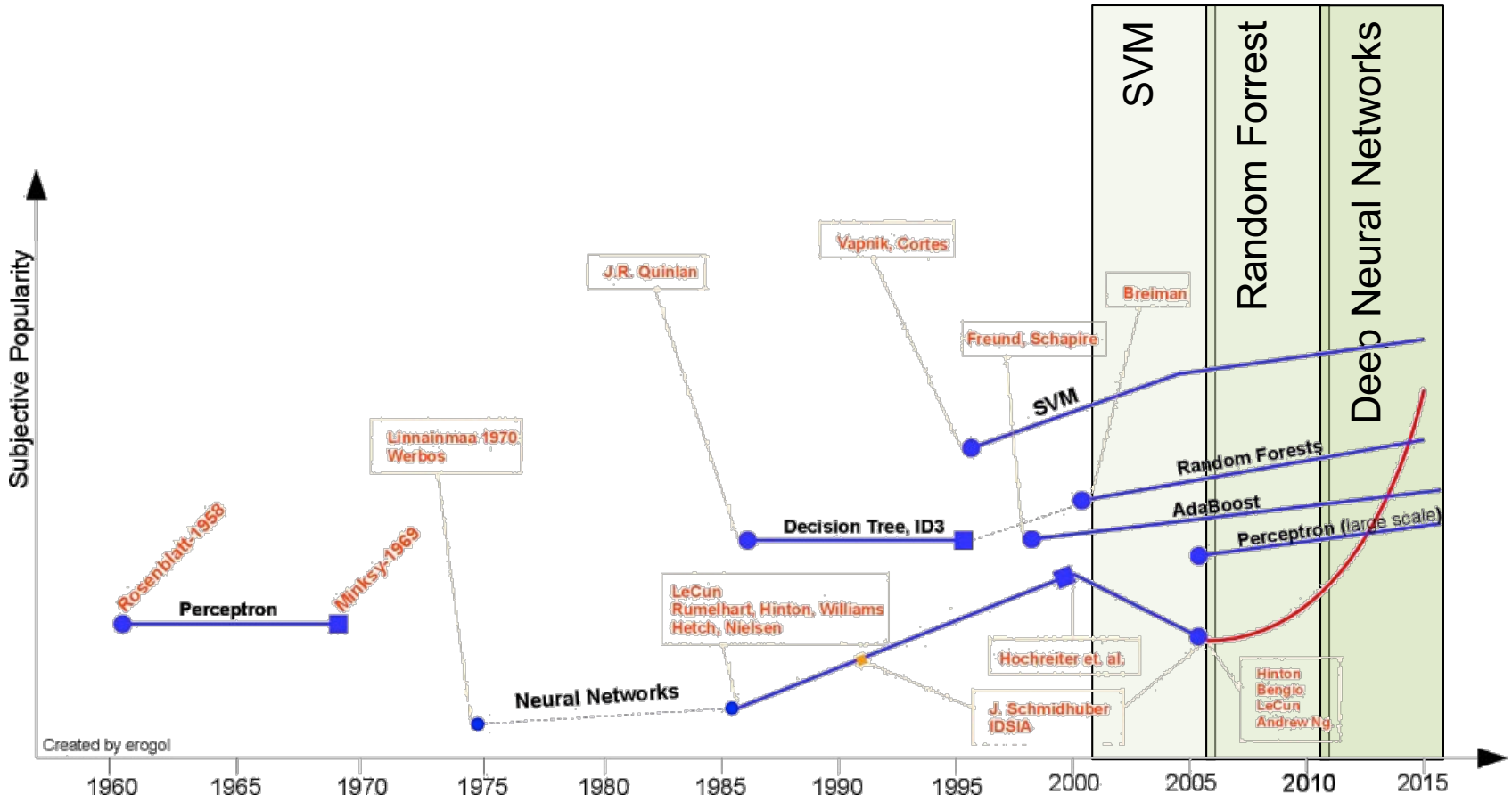


Slide credit: https://www.datasciencecentral.com/profiles/blogs/artificial-intelligence-vs-machine-learning-vs-deep-learning

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?

Deep Learning vs. Machine Learning

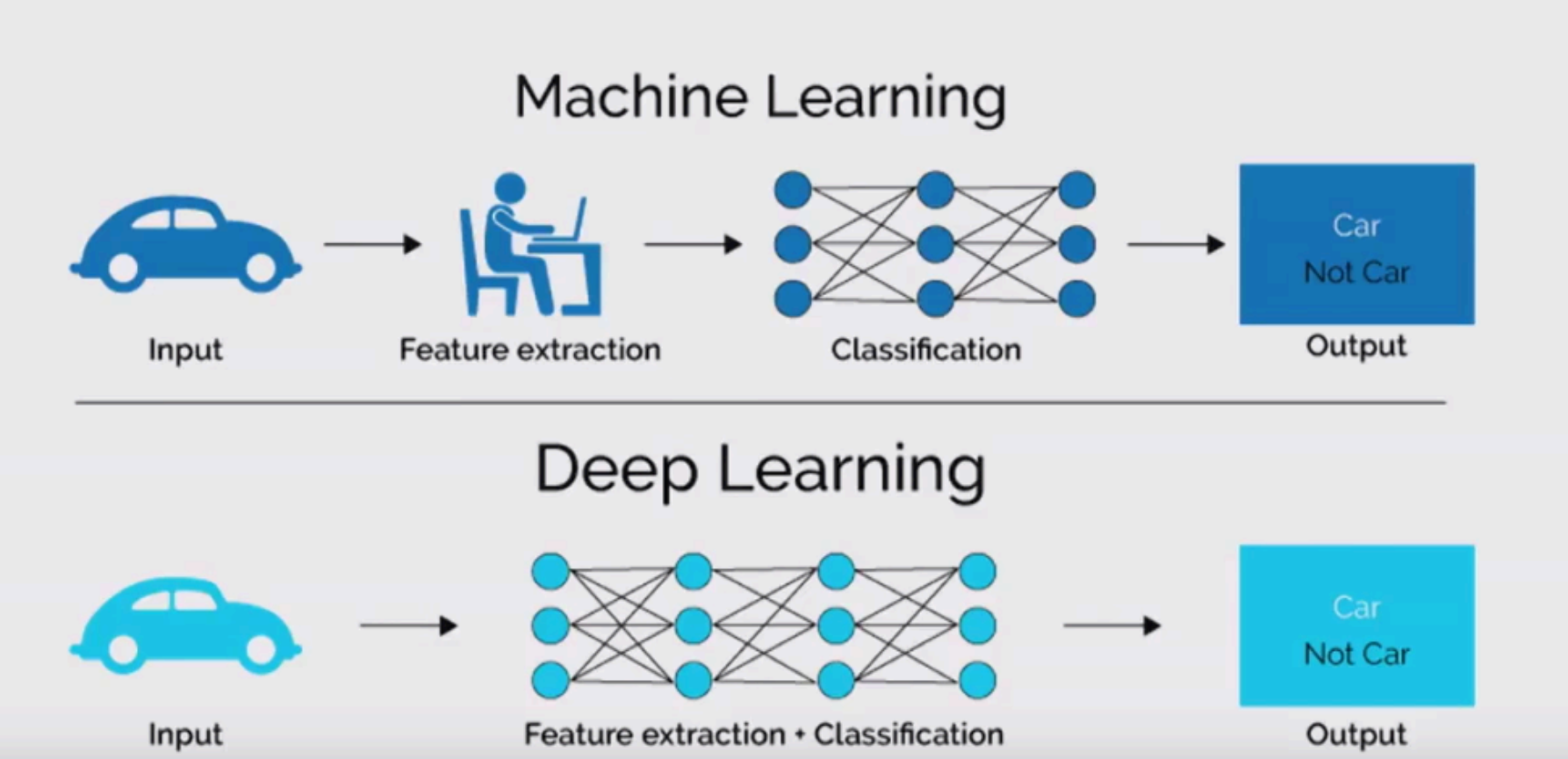


image credit: Siraj Raval

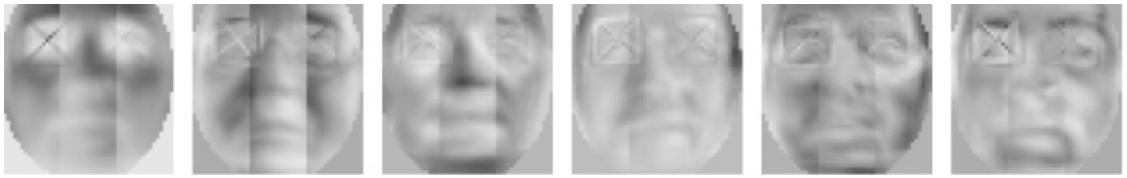
Traditional Feature Engineering: on images

- Simplest version: Just unroll the image

1.1	2	3
4	5	6
7	8	9

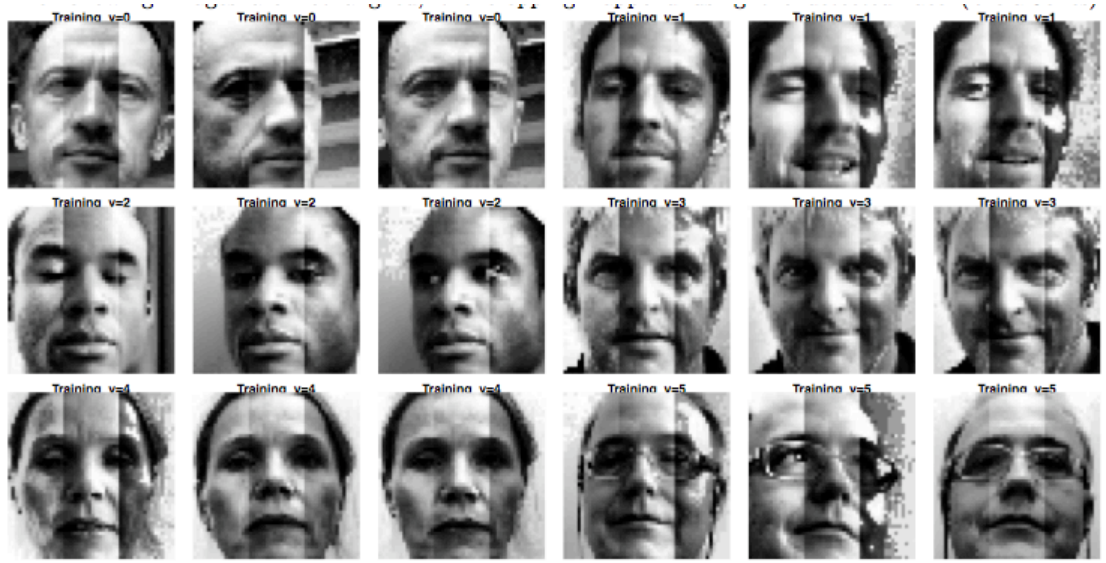
→ (1.1,2,3,4,5,6,7,8,9)

- Used for example in Eigenfaces / Fisher-Faces



- What's wrong with this approach?

How stupid is the machine I



Unaligned

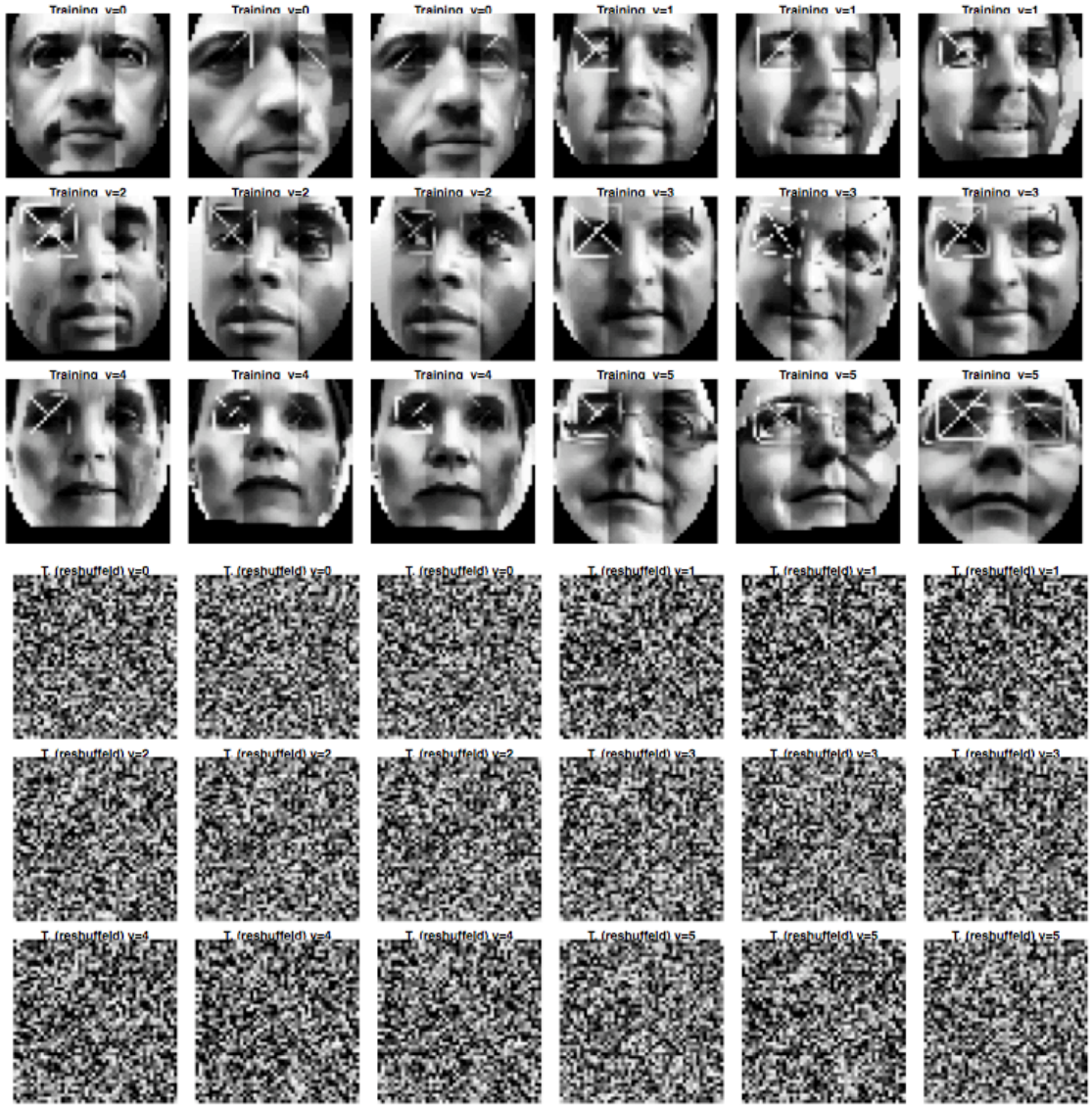
Machine Performance: quite bad

Aligned (fixed eyes)

Machine: OK

Humans don‘t care about the alignment

How stupid is the machine II



Aligned

Aligned & Shuffeld

Guess the performance?

.	Eigenfaces	Fisher	SVM
Reshuffeled	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 hierachy is learnt

- In the lectures on CNN (starting week 4) we see better approaches

Traditional feature engineering on text

aardvark

- The famous bag of words

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

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

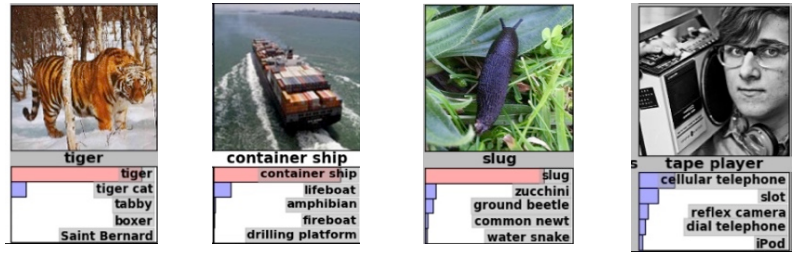
- **n-grams**
  - counts co-occurrence of a sequence of words

- Do you believe that with this features you can learn to understand language?

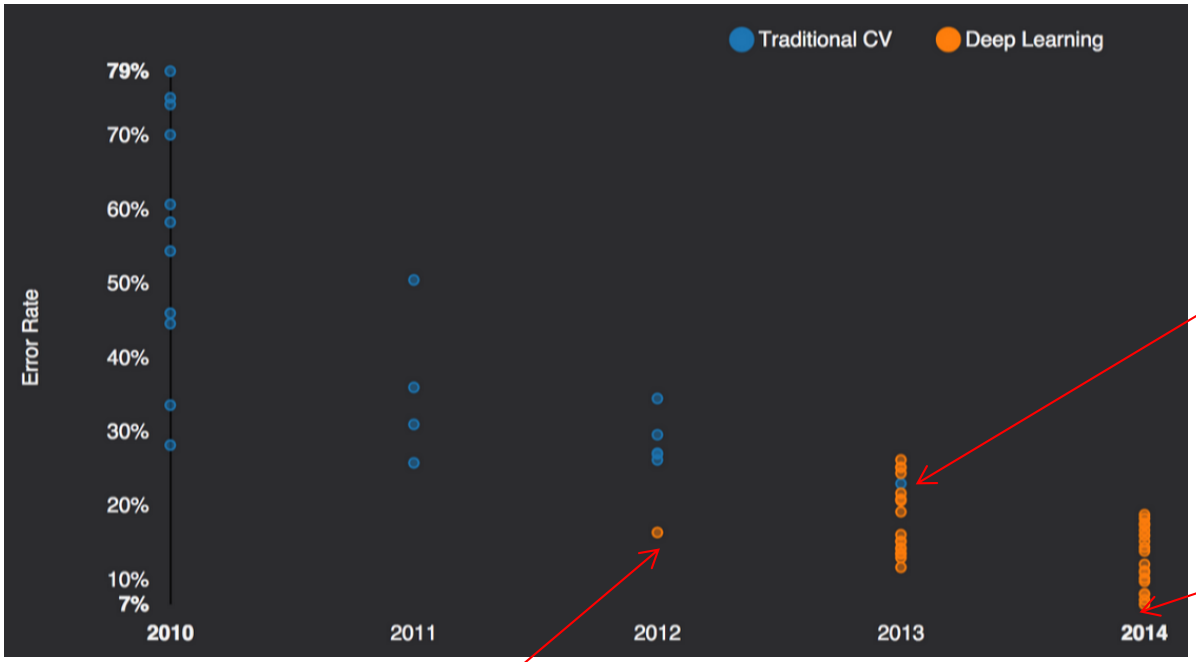
The most convincing case for DL (subjective view)

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)

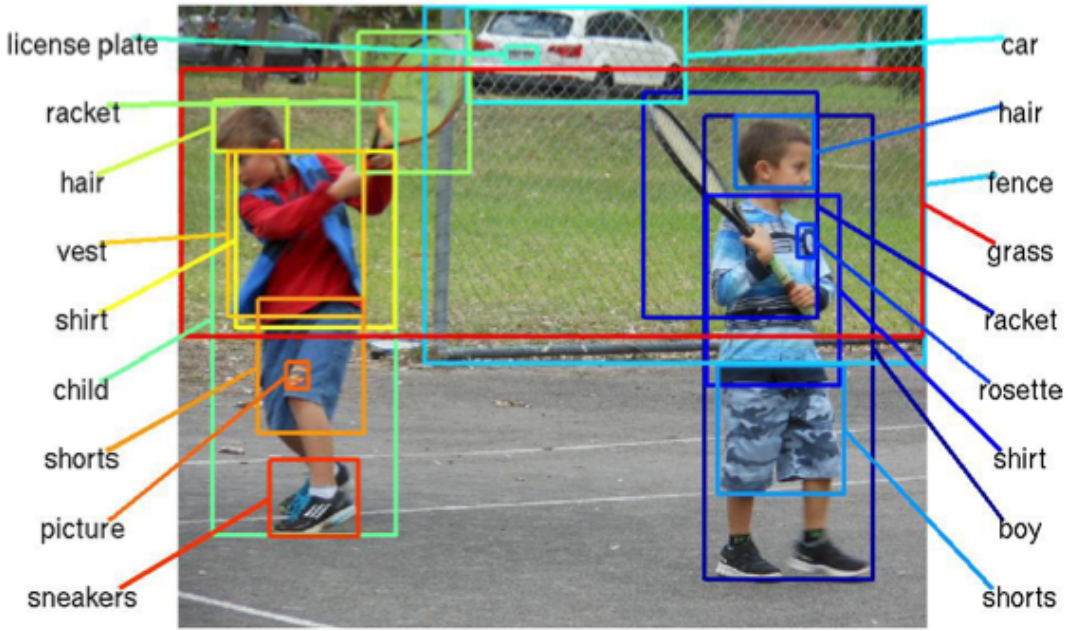
Figure: https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519

20

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."

Images form cs229n

The importance of seeing: the Cambrian explosion

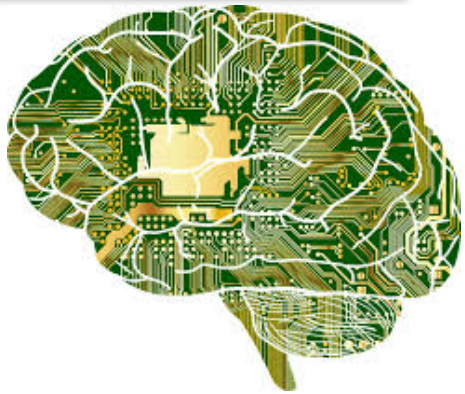
In a short period of time ~20 mio yrs (~540 mio yrs ago) most animal groups appeared.

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

A simpler explanation

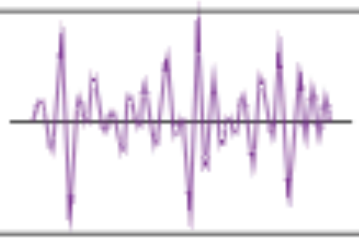
creatures learned to see

A more liable equivalence of deep learning than …



Idea: Stolen from CS231n, brain wikipedia

Use cases of deep learning

Input x to DL model	Output y of DL model	Application
Images 	Label "Tiger"	Image classification
Audio 	Sequence / Text "see you tomorrow"	Voice recognition
ASCII-Sequences "Hallo, wie gehts?"	Unicode-Sequences "你好，你好吗?"	Translation
ASCII-Sequence This movie was rather good	Label (Sentiment) positive	Sentiment analysis

Deep Learning öffnet Tür zu hören, sehen und Texten. (kein Verstehen aber statistische Zusammenhänge).

Source: Oliver Dürr, Beate Sick, Elvis Murina in Deep Learning in Depth, Manning forthcomming

Deep Learning vs. Machine Learning

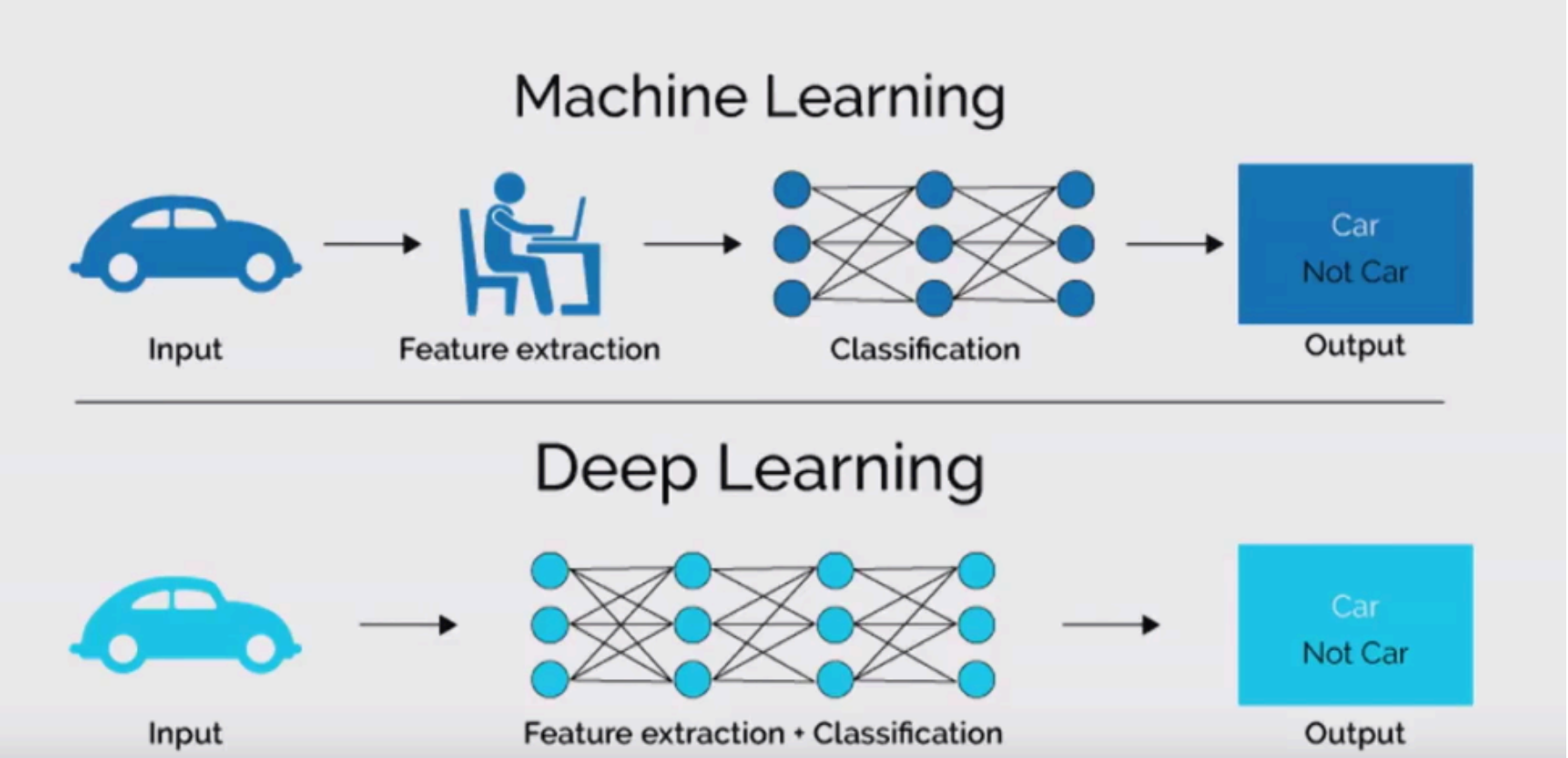
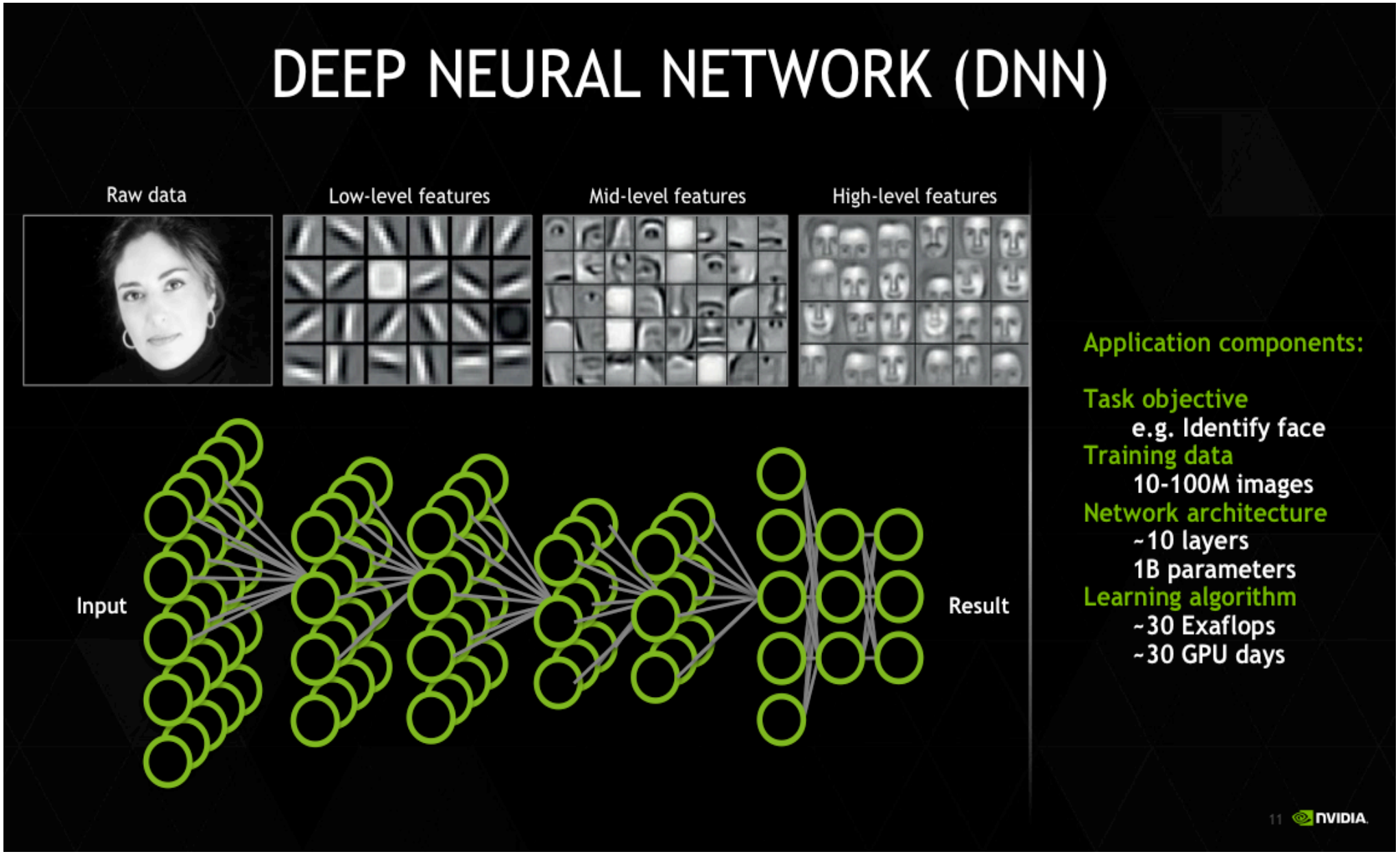


image credit: Siraj Raval

Main Idea in DL



Learn hierarchy of features

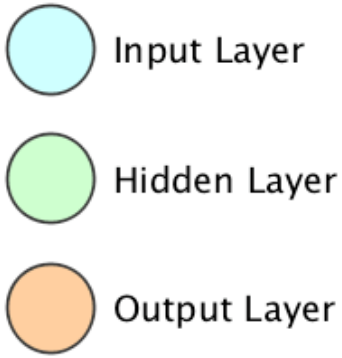
What's typical deep learning?

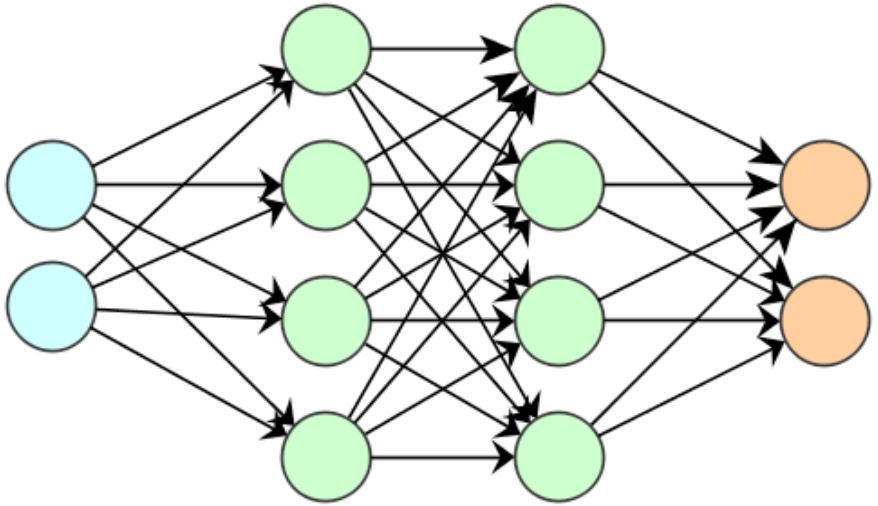
- Data (basically raw)
  - Little / no feature engineering
  - A hierarchy of increasingly complex features is learnt
- Architecture (deep)
  - Composition of simple building blocks "***layers***"
    - Fully Connected, Softmax, Convolutional, Max-Pooling, LSTM-Cells,…
- Training (long)
  - **A loss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

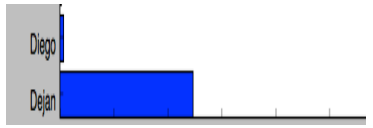
Recent success mainly due to **more data** and **more compute** power (GPU). The building blocks where there for years.

27

Preview: The first network



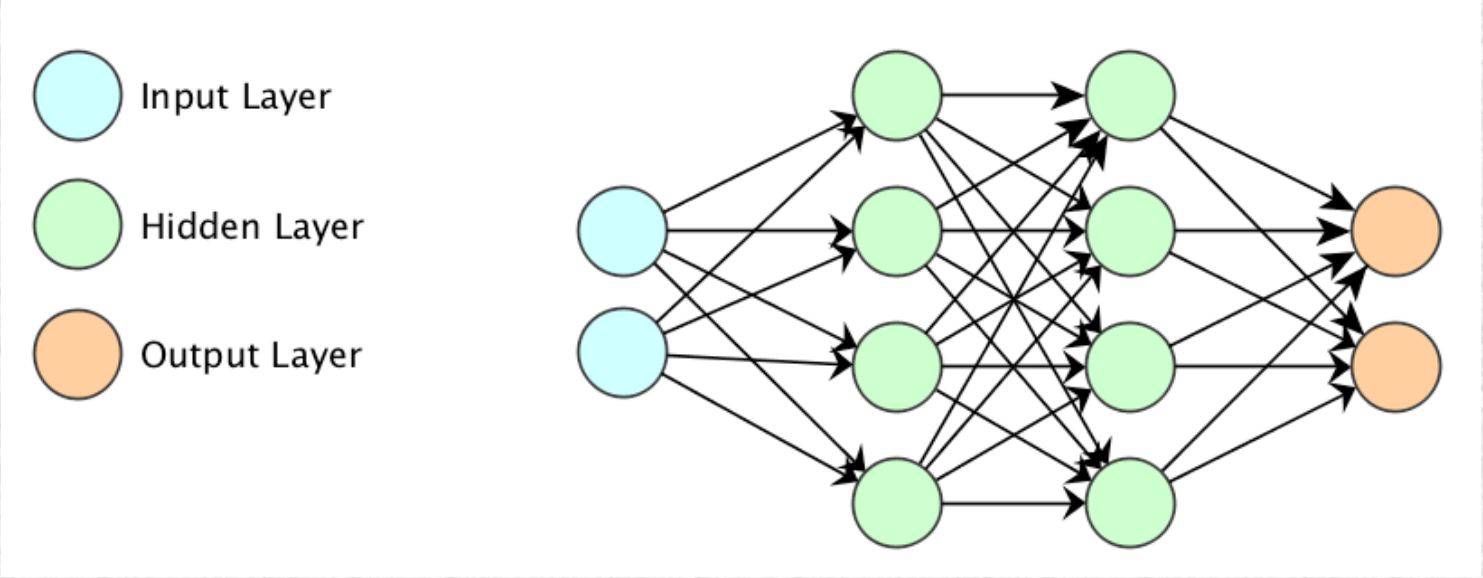




- The input: e.g. intensity values of pixels of an image
- Information is processed layer by layer
- Output: probability that image belongs to certain person

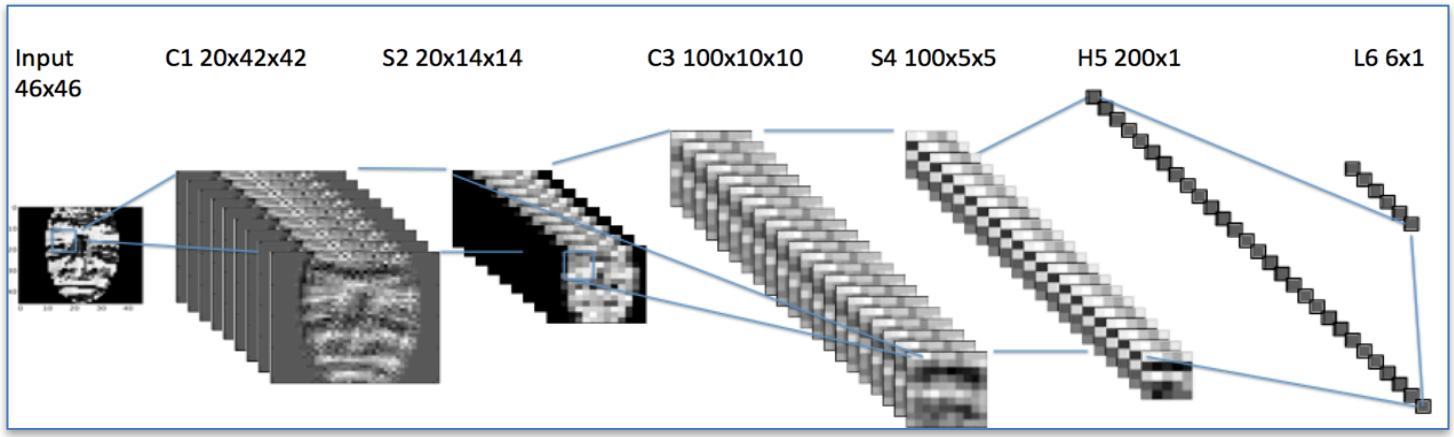
- Arrows are weights (these need to be learned)

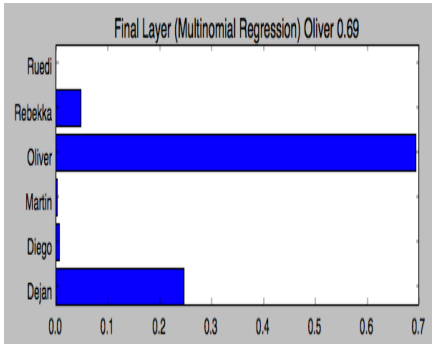
Preview: The first network



- 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, tensorflow eager)
  - 2 step procedure (1st build graph, 2nd 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
  - ...

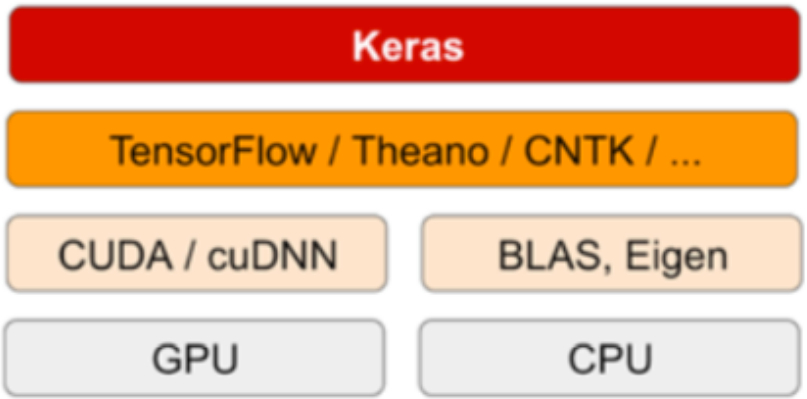


Figure: From Deep Learning with Python, Francois Chollett

Some Facts about TensorFlow

- Open sourced 9th 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

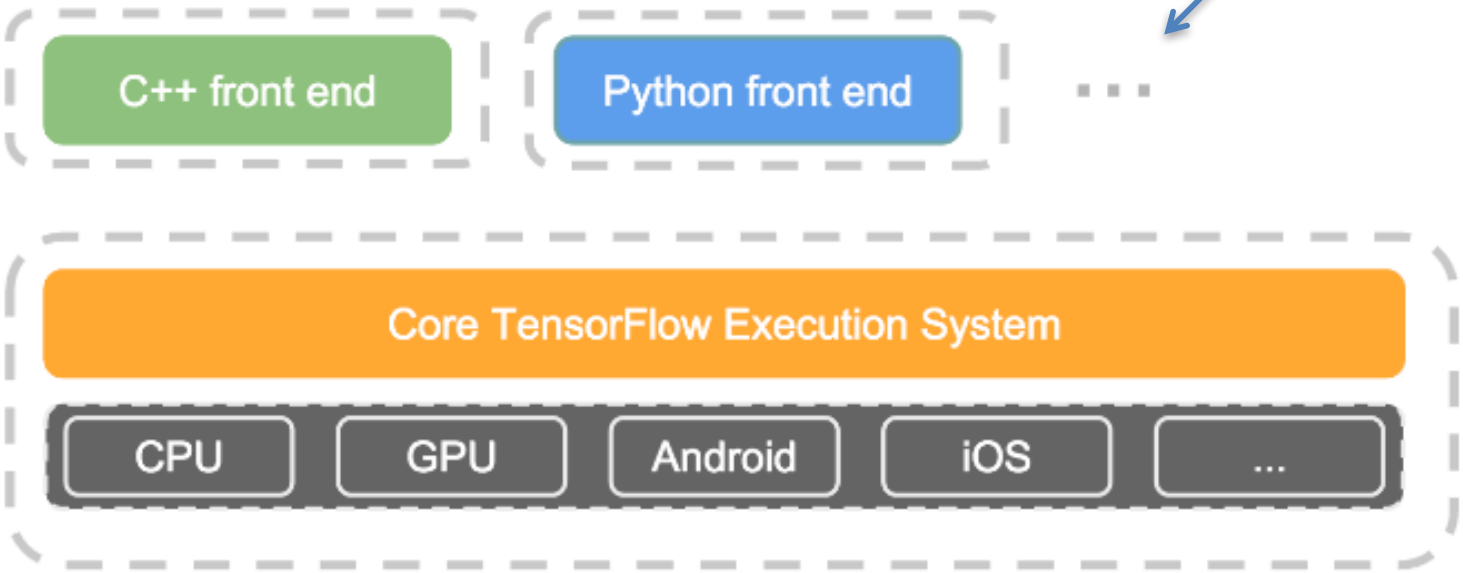
Slides from Deep Learning Day Tutorial

Some Facts about TensorFlow

Front Ends

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

and **R!**



Slides from Deep Learning Day Tutorial

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

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

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

$$\mathbf{f} \mapsto \mathbf{f} \cdot R = \left(\mathbf{e}_i R^i_1, \dots, \mathbf{e}_i R^i_n\right)$$

then the multidimensional array obeys the transformation law

$$T^{i'_1 \dots i'_p}_{j'_1 \dots j'_q}[\mathbf{f} \cdot R] = (R^{-1})^{i'_1}_{i_1} \cdots (R^{-1})^{i'_p}_{i_p} \; T^{i_1,\dots,i_p}_{j_1,\dots,j_q}[\mathbf{f}] \; R^{j_1}_{j'_1} \cdots R^{j_q}_{j'_q}.$$

Just kidding...

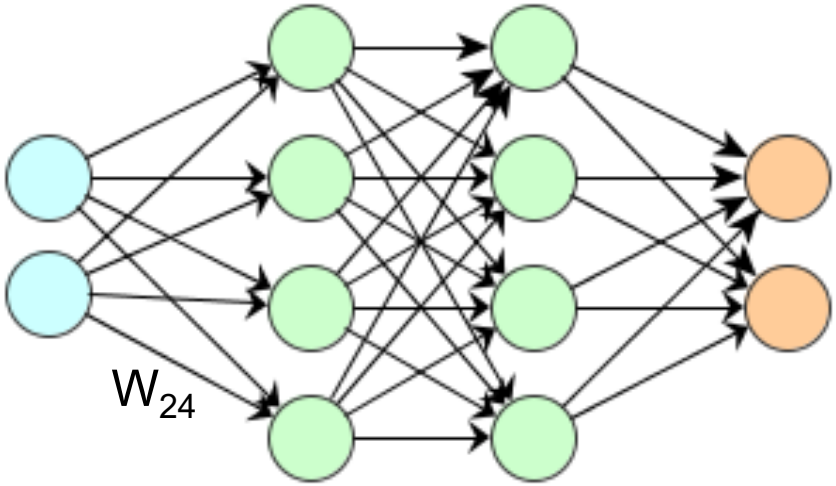
Sharpe, R. W. (1997). Differential Geometry: Cartan's Generalization of Klein's Erlangen Program. Berlin, New York: Springer-Verlag. p. 194. ISBN 978-0-387-94732-7.

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)
```

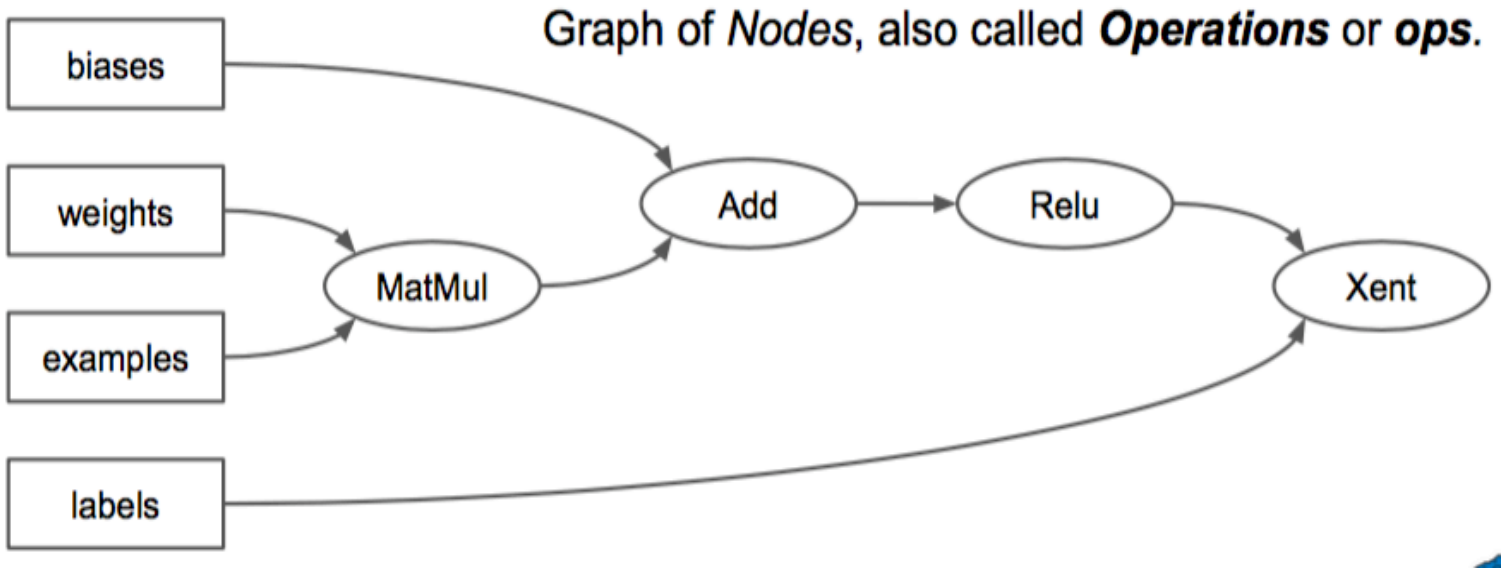
Typical Tensors in Deep Learning



- 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_j)
- A mini-batch of size 64 images with 256,256 pixels and 3 color-channels can be understood as a tensor of order 4.

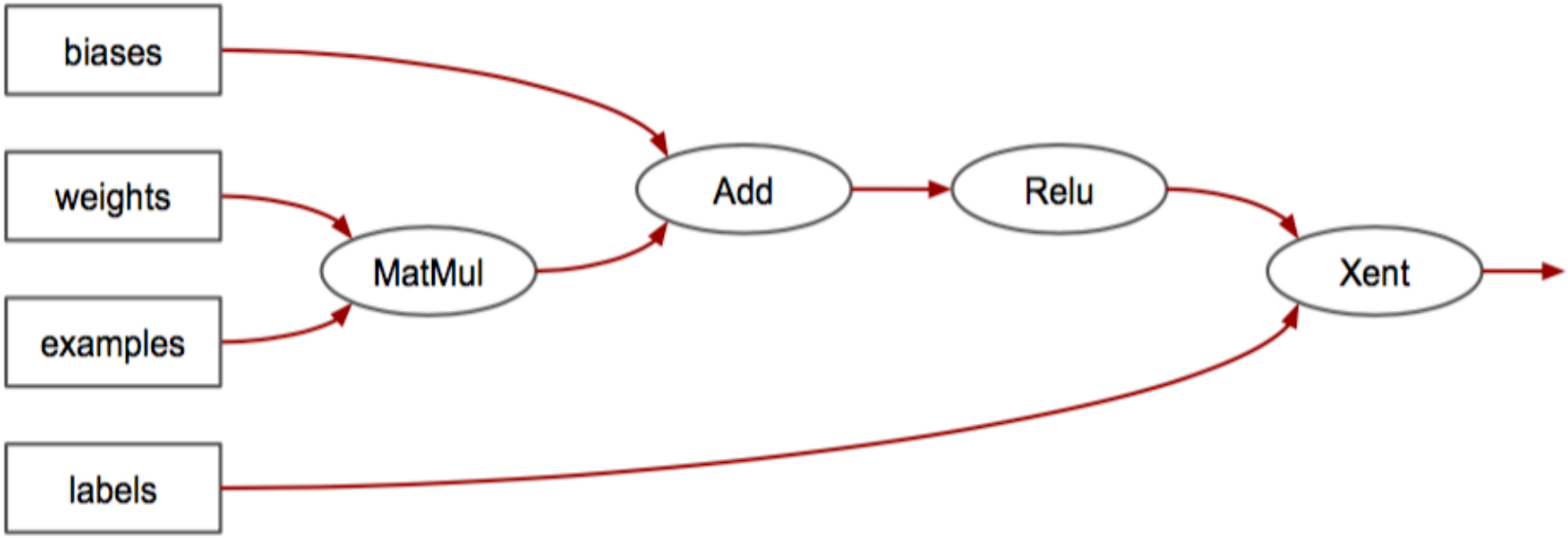
Computations in TensorFlow (and Theano)

- Computation is expressed as a dataflow graph



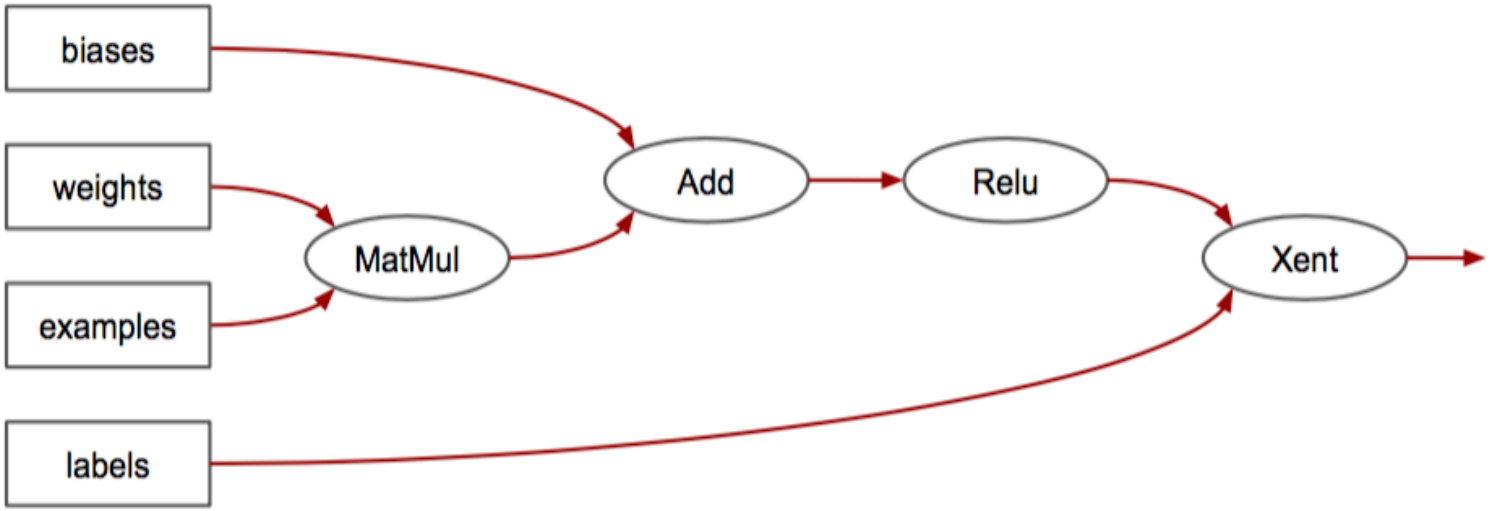
Computations in TensorFlow (and Theano)

- 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)

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

numpy

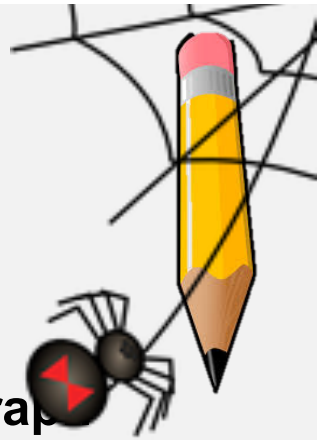
In [1]:

```
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

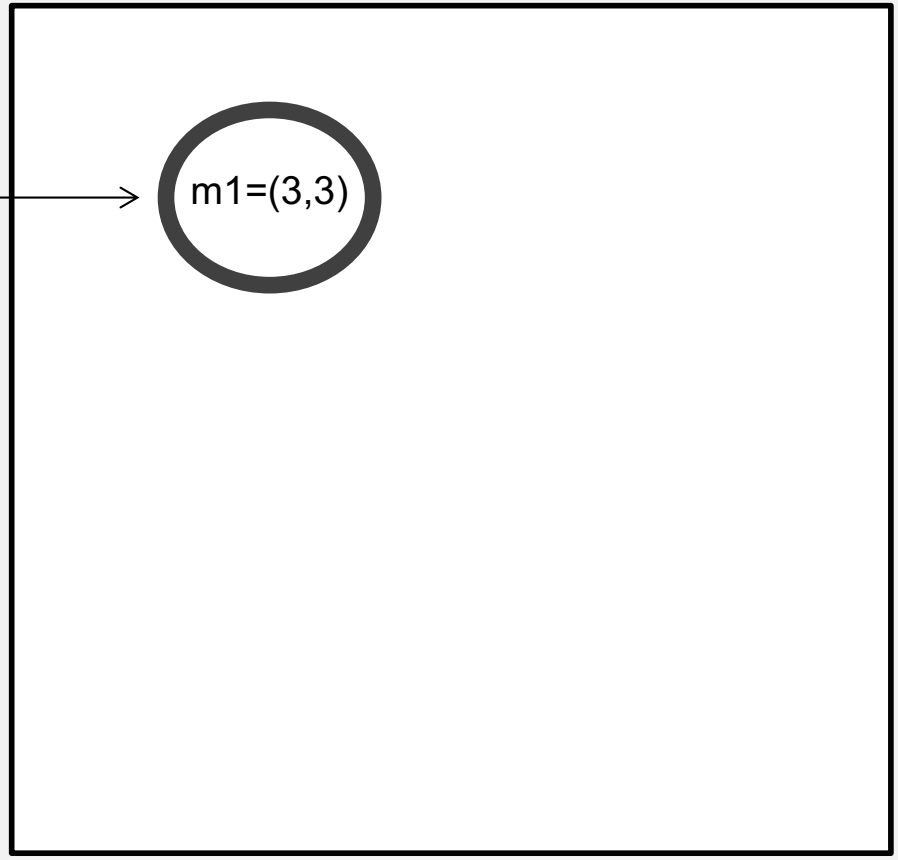
TensorFlow: Building the graph

```
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

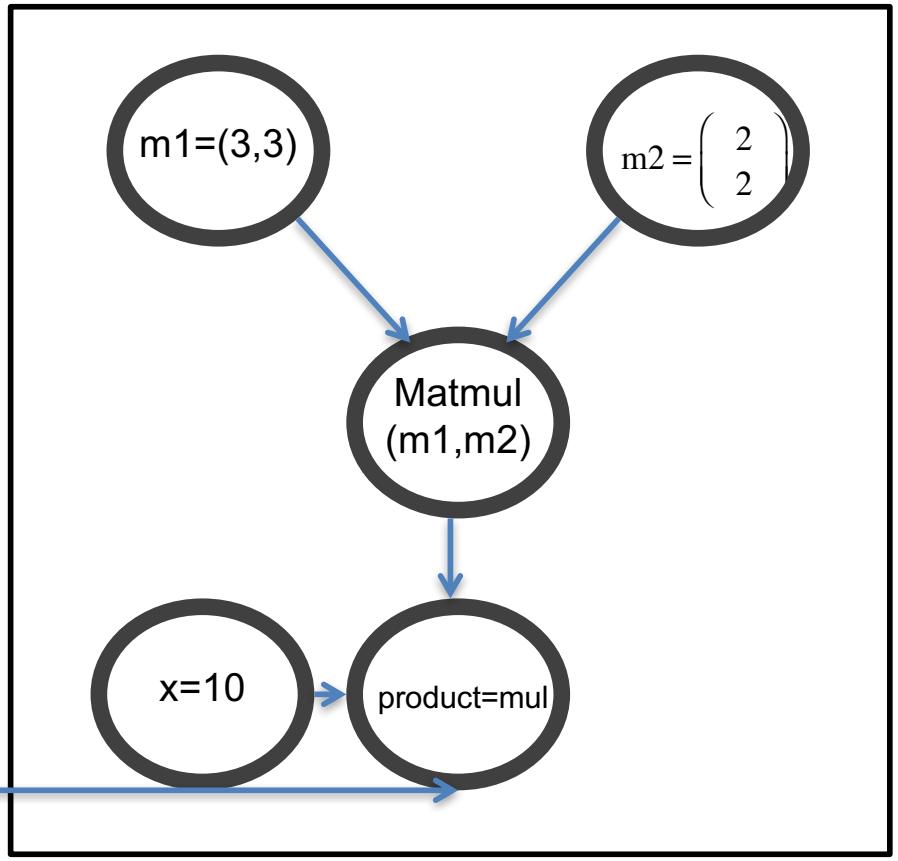
```
import tensorflow as tf
# We construct a graph (we write to the default graph)
# make first sure the default graph is empty
tf.reset_default_graph()
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

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

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

numpy

In [1]:

```
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

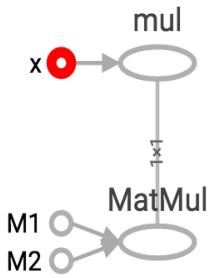
```
import tensorflow as tf
# We construct a graph (we write to the default graph)
# make first sure the default graph is empty
tf.reset_default_graph()
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

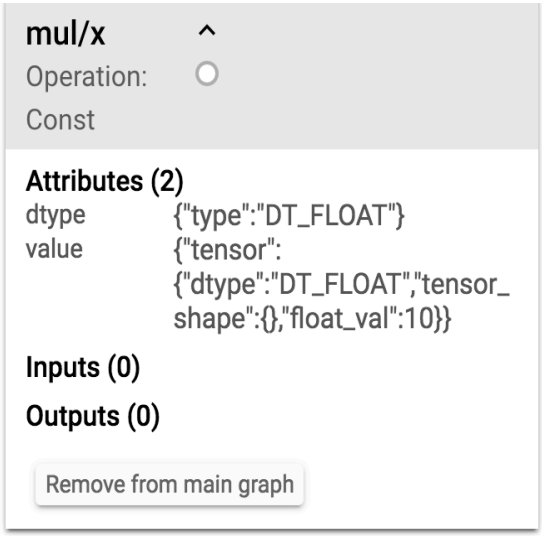
TensorFlow: Executing the graph

In [4]:

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

```
[[ 120.]]
```





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
- No installation? ➔ Google Colab

First Task with TensorFlow

Website: 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 restults. 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

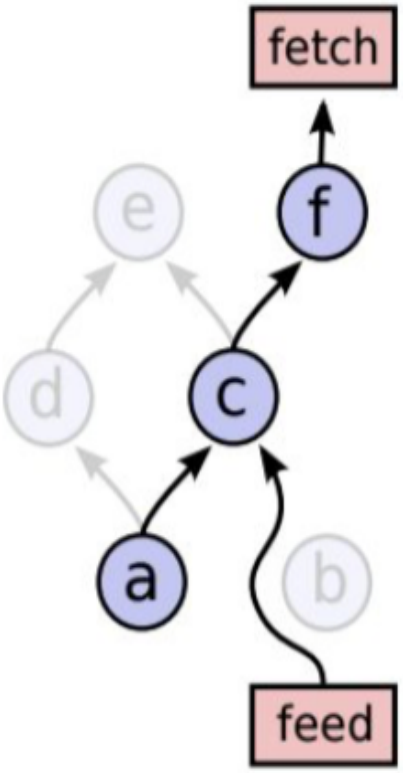
- A session **is a concrete realization**
  - It places the ops on physical devices such as GPUs
  - It initializes variables
  - We can feed and fetch a session (see next slides)

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

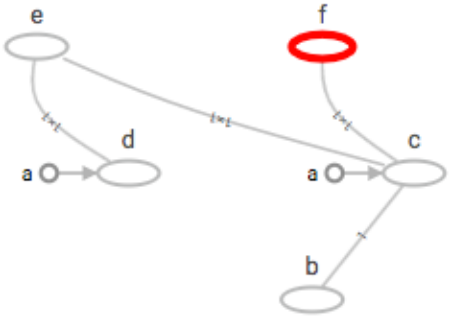
Alternatively use the with construct

```
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]})
```

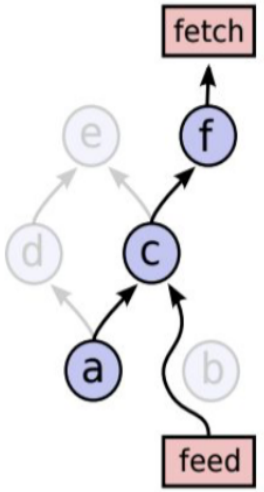
fetch
(the numeric value)

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)

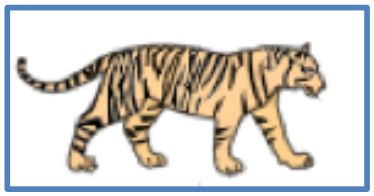
two inputs (feeds)

Prinzip von Deep Learning: Training

Prinzipielle Funktionsweise: Training Bild Klassifikation

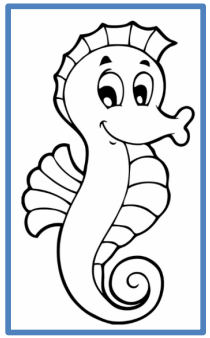
Wahre Klasse

Vorhergesagtes Label

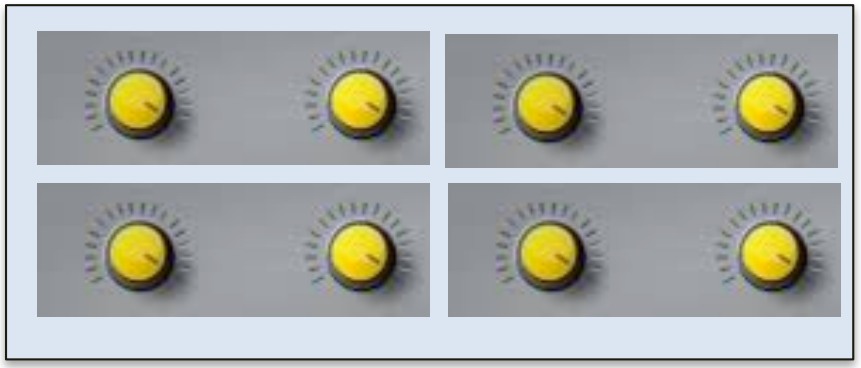


Tiger → Seehund 👎

Tiger → Tiger 👍



Seepferd → Seepferd 👍



...

Typisch 1 Mio. Trainingsdaten

Trainingsprinzip:
Parameter werden so eingestellt, dass möglichst wenige Fehler in den Trainingsdaten gemacht werden.

Prinzipielle Funktionsweise: Training Lineare Regression

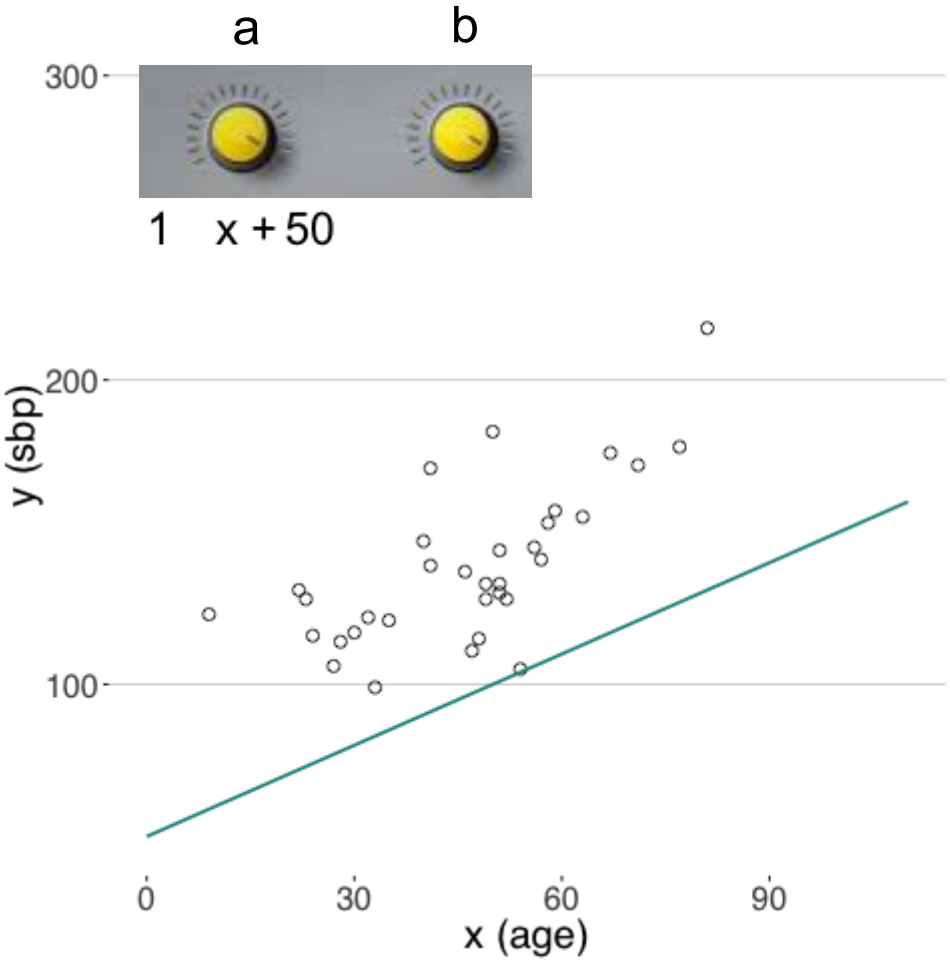
	Wahrer Blutdruck	Vorhergesagter Blutdruck
Alter: 55	152	130
Alter: 35	155	151
Alter: 55	140	131

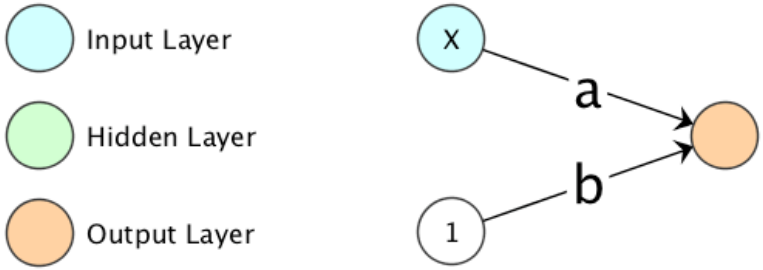
Das einfachste “Deep Learning“ Model hat 2 Parameter.

...

33 Trainingsdaten aus einer Studie von nordamerikanischen Frauen

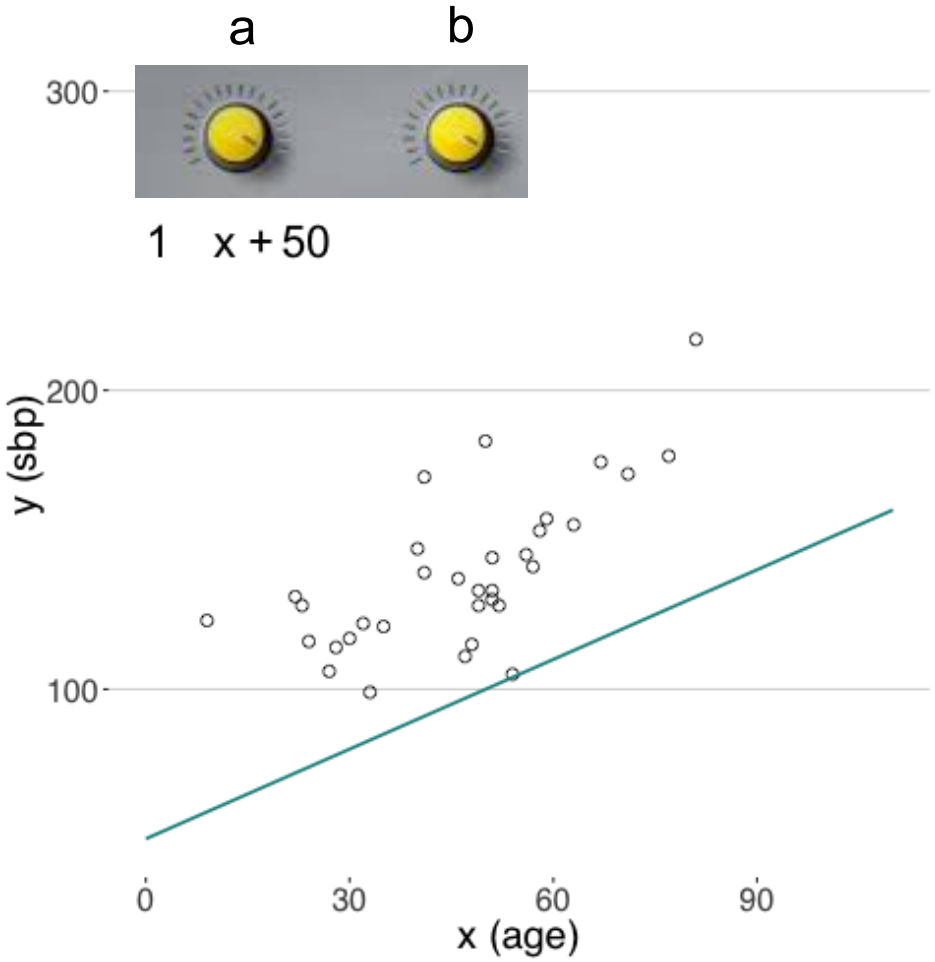
Linear Regression (nicht angepasstes Model)

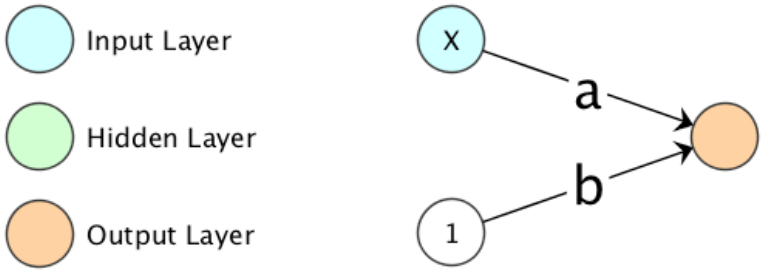




$$\hat{y}^{(i)} = ax^{(i)} + b$$

Linear Regression (Training)

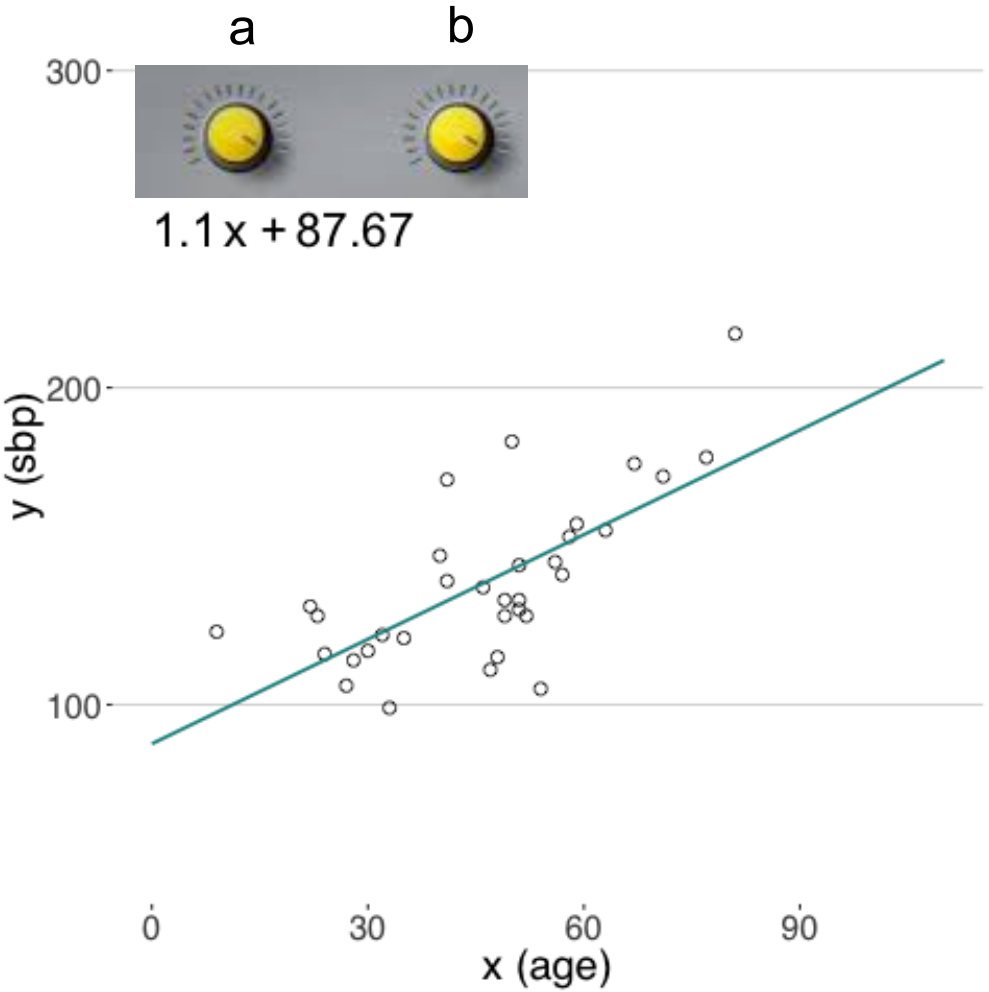


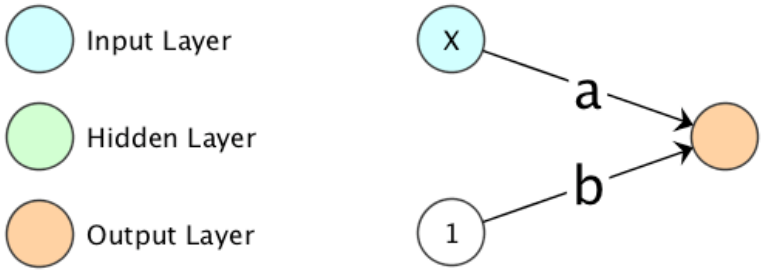


$$\hat{y}^{(i)} = ax^{(i)} + b$$

- Parameter a und b werden an die Daten angepasst.
  - Training oder Fitting

Linear Regression (Trainiertes Modell)





$$\hat{y}^{(i)} = ax^{(i)} + b$$

- Haben ein Modell und können dann vorhersagen.
  - Intrapolation vs. Extrapolation
- Passen die Trainingsdaten nicht, dann stimmt auch Vorhersage nicht

The loss function

Tuning a neural network

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

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

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

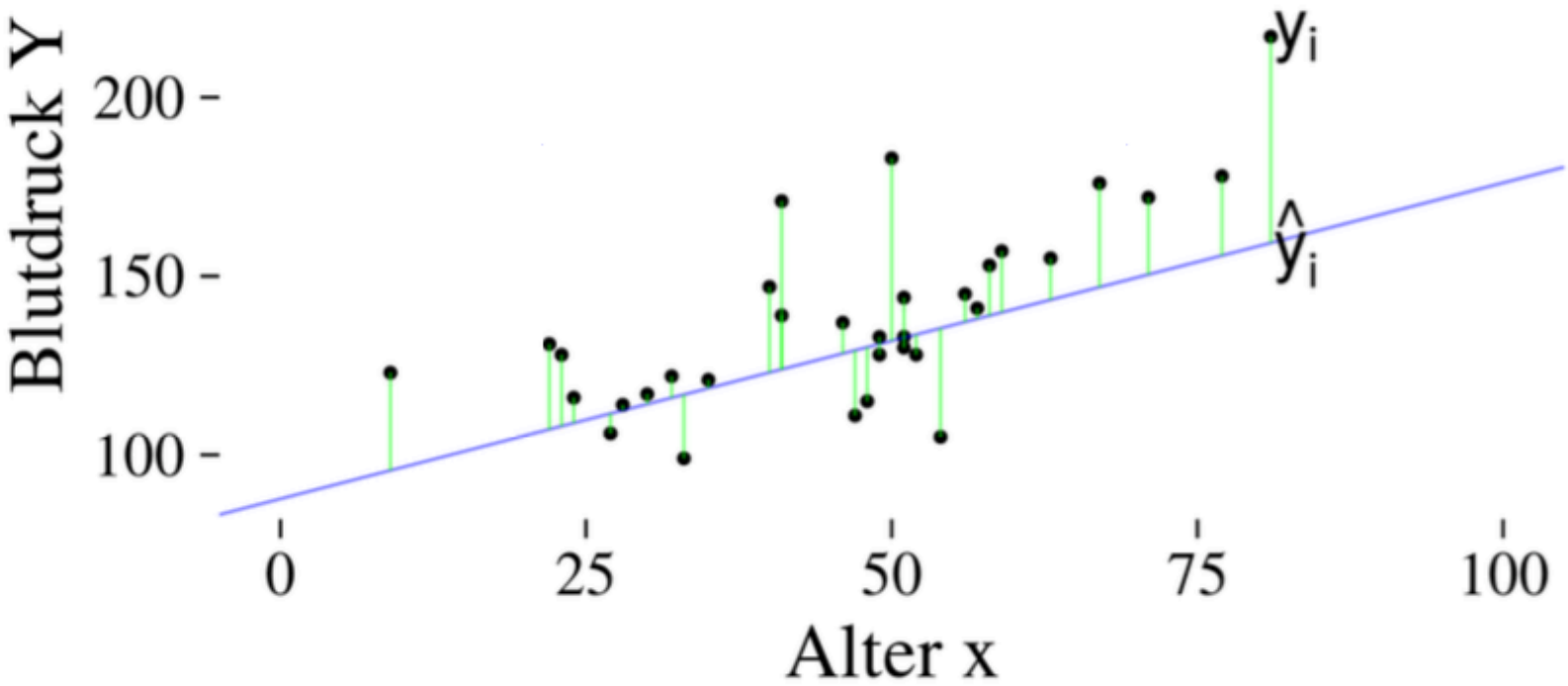
- Examples (your task what are the x's what are the y'?)
  - Facerec.: Faces and Names
  - Age Prediction: Faces and Age (numerical problem)

- How to tune the nobs 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

Loss for linear regression: sums of squared error

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



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 fist 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)

Ende FS 2018 @17:00

Optimization

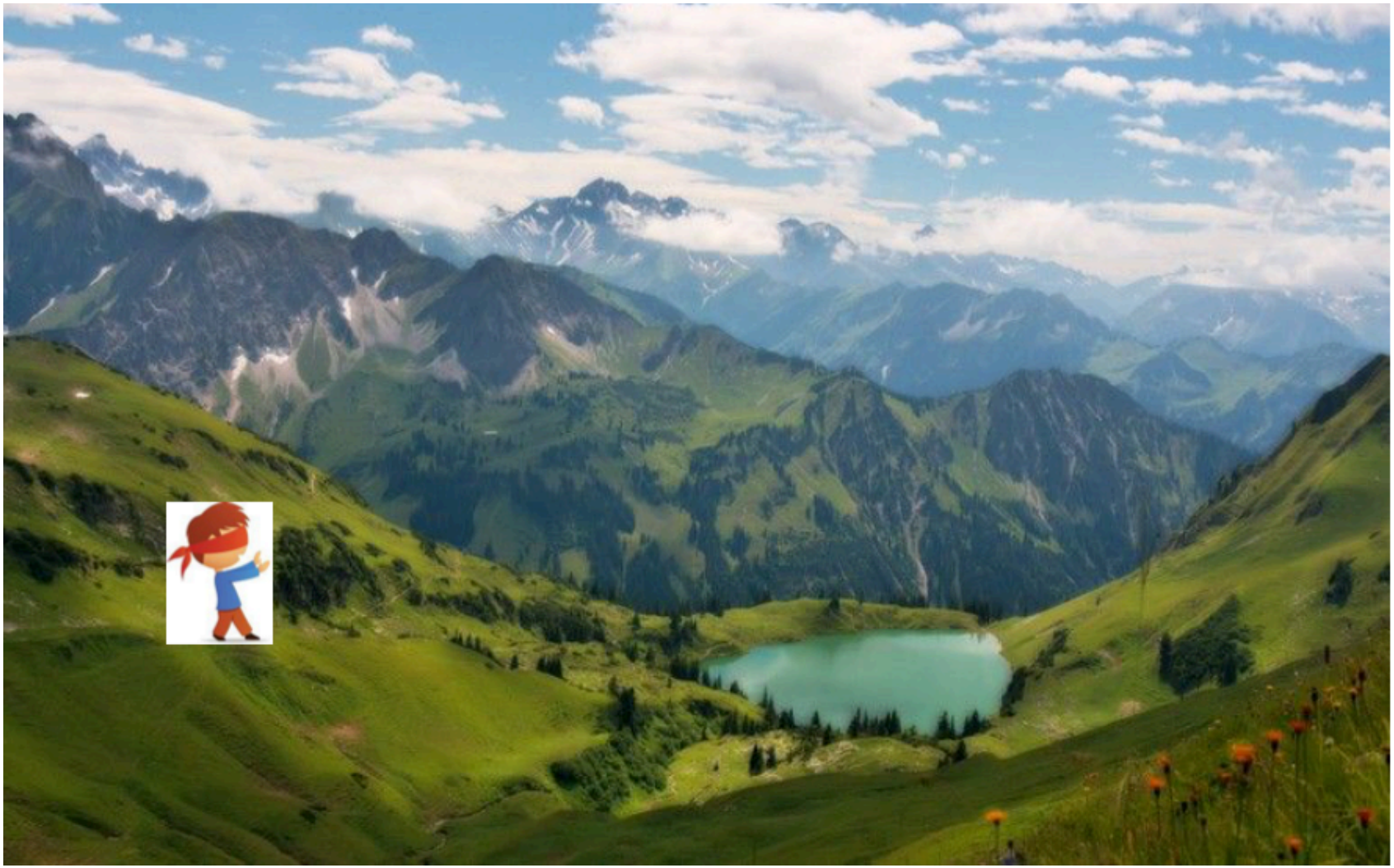
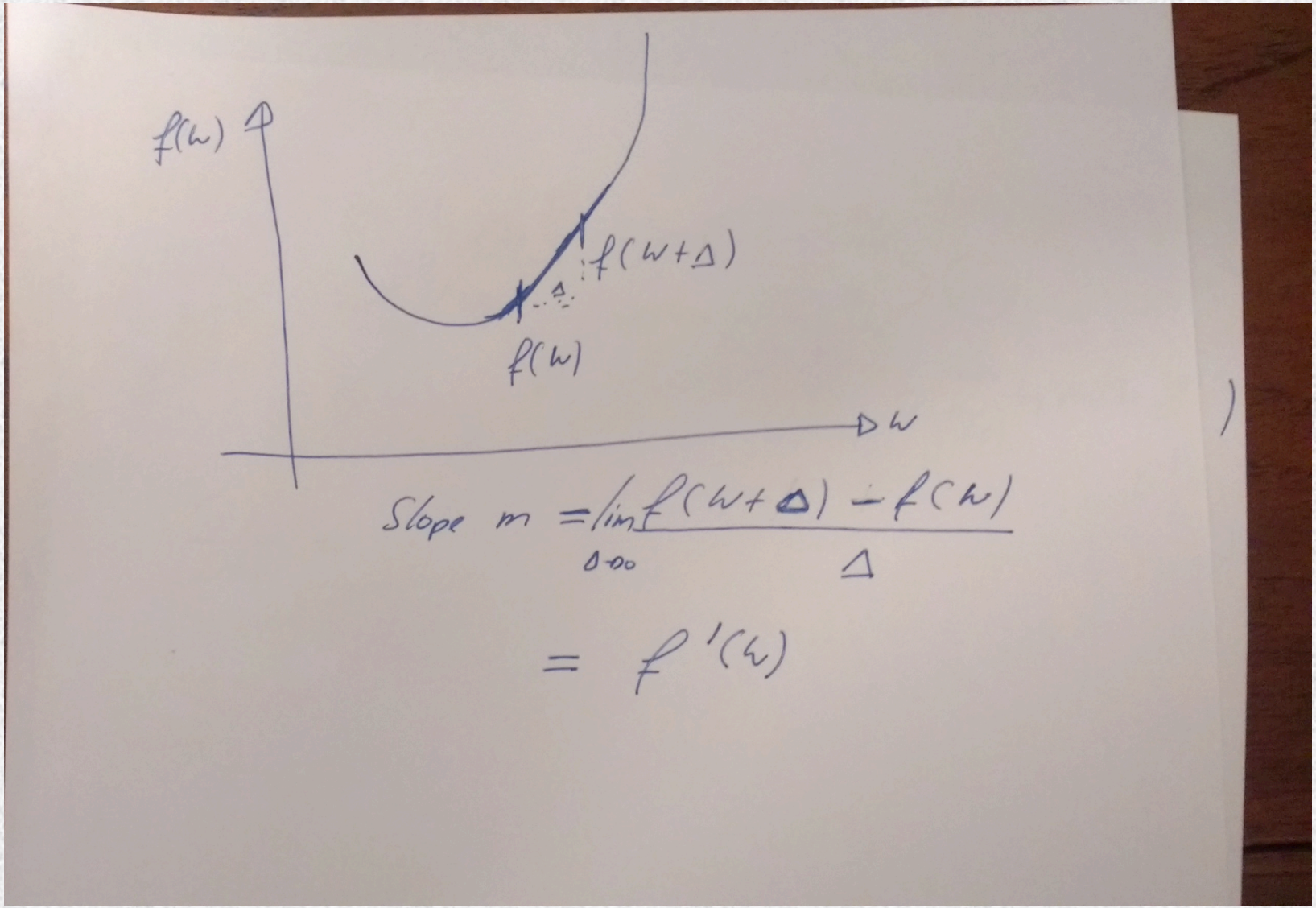


Figure shows a 2 dimensional loss function. In DL Millions!
We just know the current value (blind)

Slide from cs229

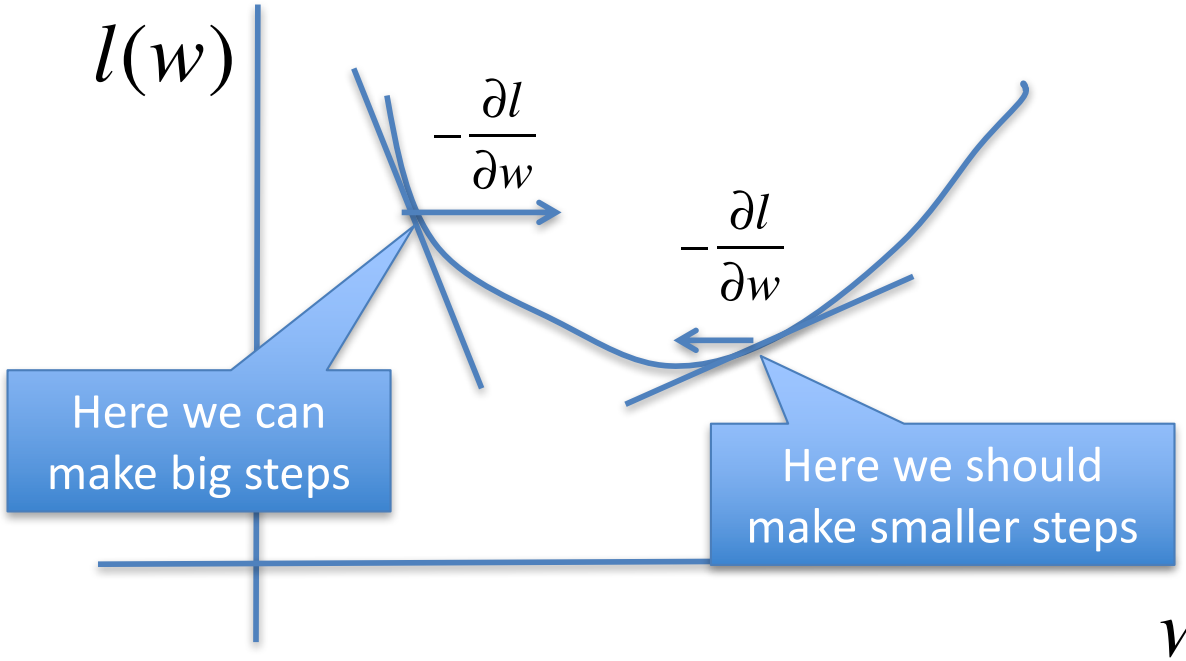
Gradient Descent: Gradient of 1-D function

- **Draw**



Optimization

- Intuition of Gradient Descent
  - We just know the value of the current loss function
  - The gradient gives us the direction and slope of the steepest descent
  - We just take a single step in direction of the steepest descent
  - The steeper the curve the larger the possible step
- Gradient in 1d (derivative)



$-\frac{\partial l}{\partial w}$

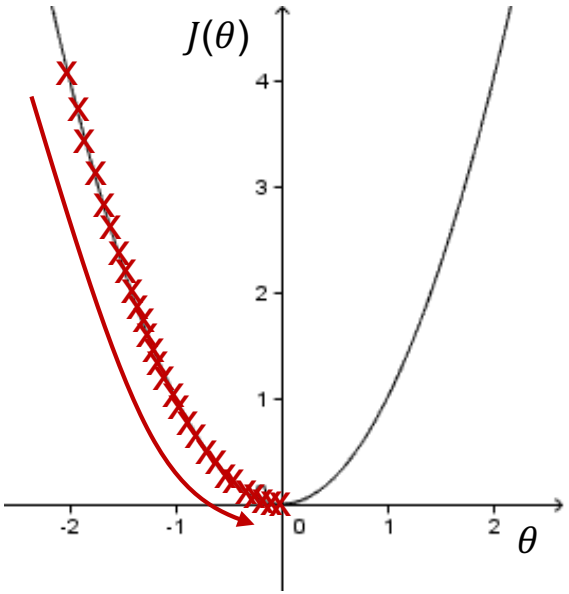
- Points to the downward direction.
- The magnitude is proportional to the slope

Iterative update

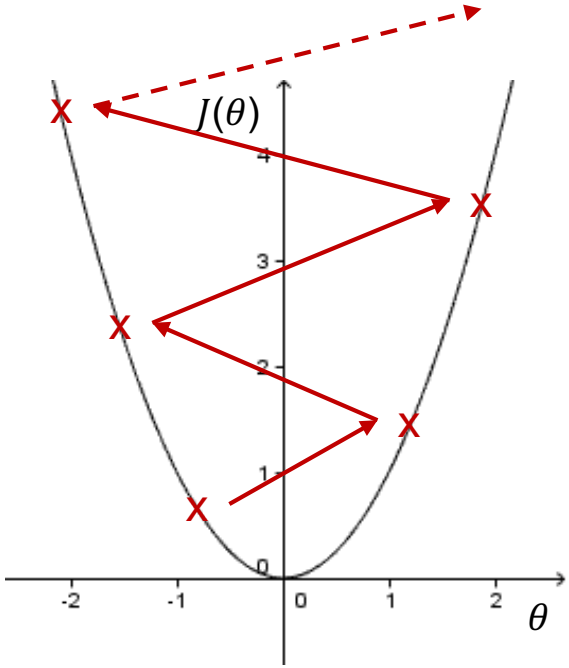
$$w^{t+1} = w^t - \varepsilon \left.\frac{\partial l}{\partial w}\right|_{w^t}$$

Problem with step size ε

- too small
  ➔ gradient descent is slow



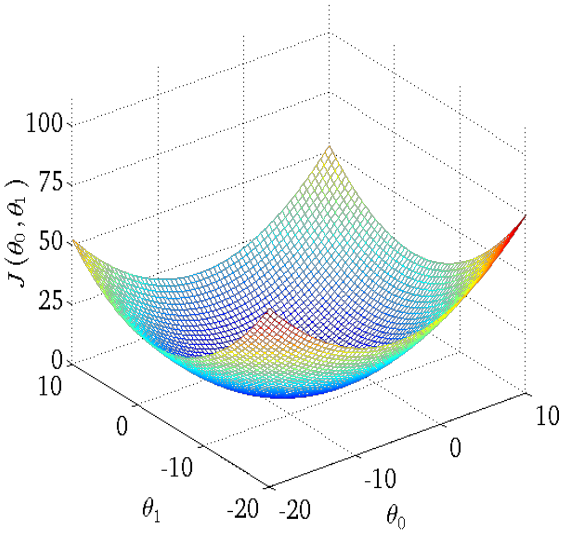
- too large
  ➔ gradient descent overshoots minimum
  ➔ no convergence or divergence!

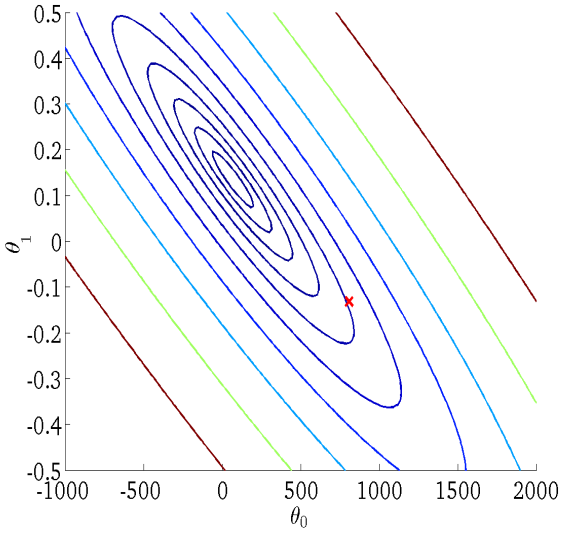


Slide credit: Thilo

Optimization

- 2 equivalent representations





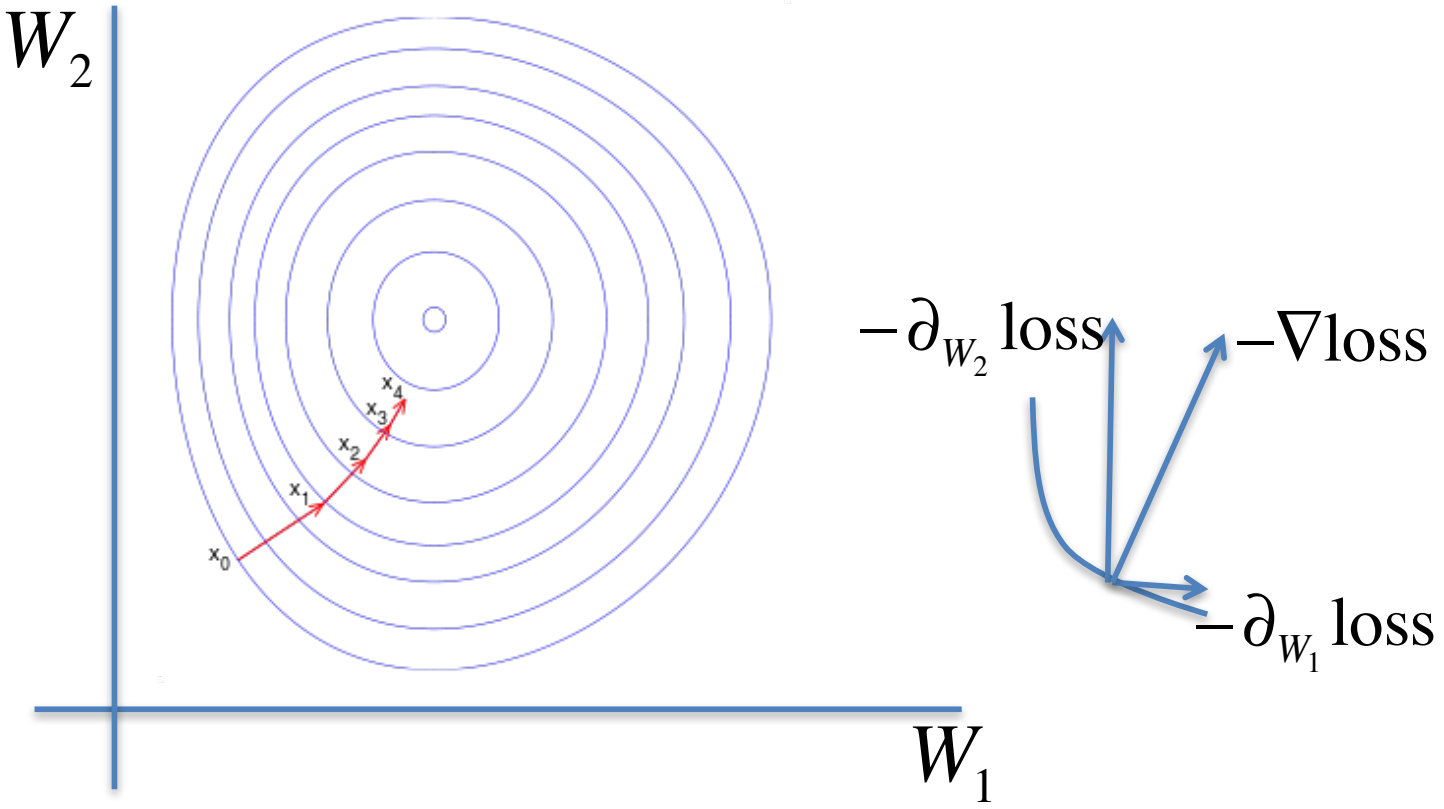
Slide credit: Thilo, wikipedia

Optimization

- Gradient Descent

  Gradient is perpendicular to levels

$$W_i^{t+1} = W_i^t - \varepsilon \partial_{W_i} \text{loss}$$

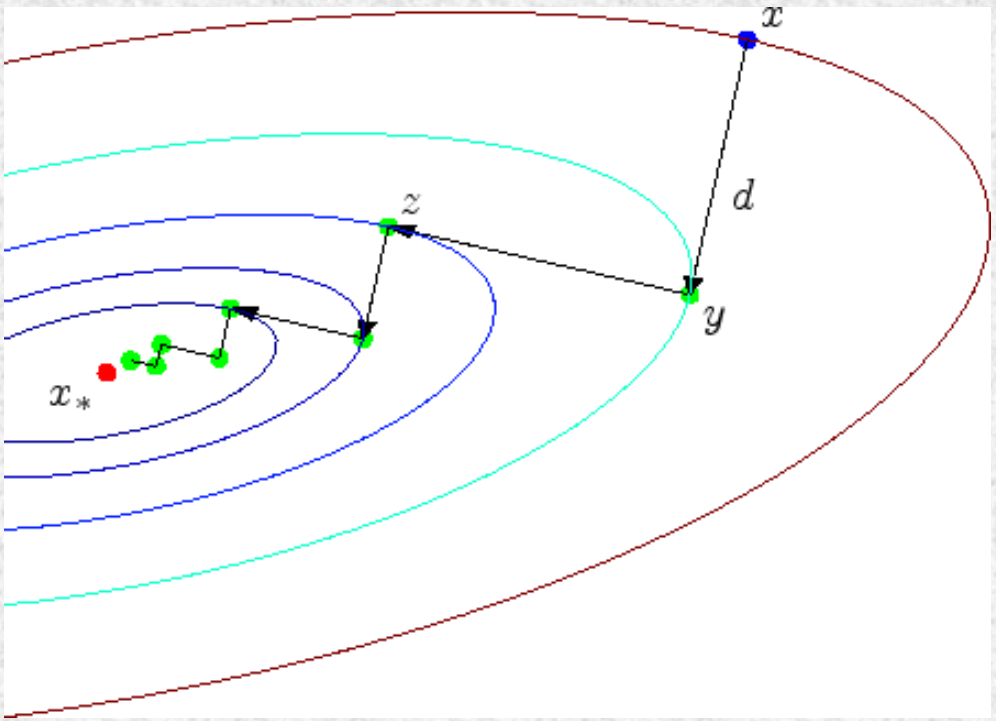


Slide credit: wikipedia

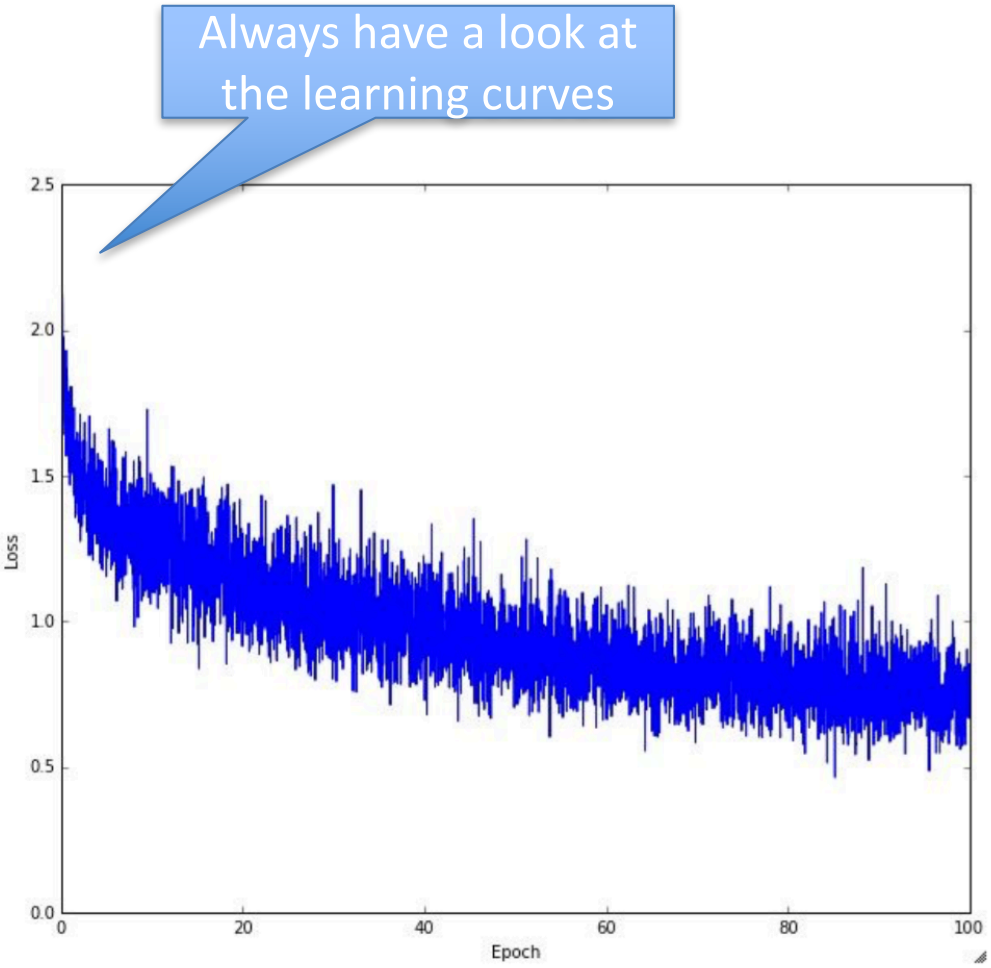
Tafel

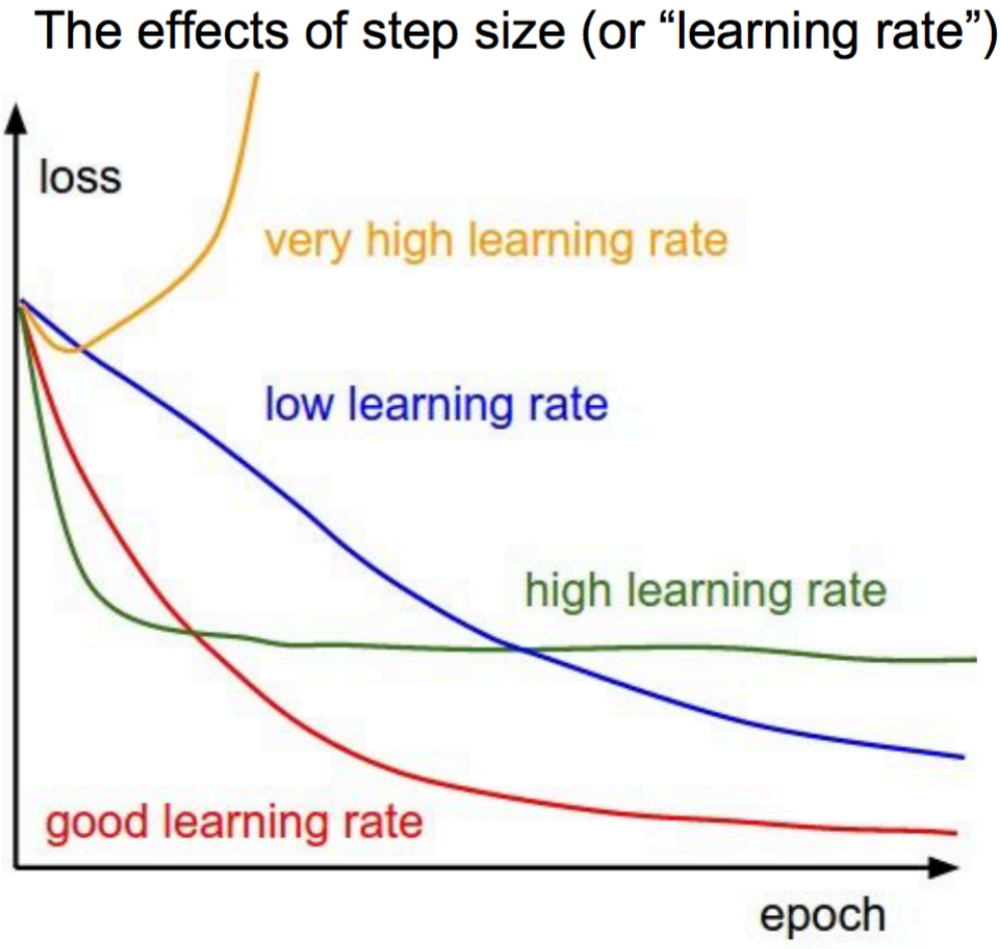
- **Höhenlinien einzeichenen und runterhüpfen**
  - Gradient senkrecht zu Höhenlinien

- **Make clear that this is an iterative**



Gradient Descent in DL





Gradient Descent in TensorFlow

- In Theano and TensorFlow the Framework does the calculation of the gradient for you (autodiff)
- You just have to provide a graph

```
# loss has to be defined symbolically
train_op = tf.train.GradientDescentOptimizer(0.0001).minimize(loss)

…
for e in range(epochs): #Fitting the data for some epochs
  res = sess.run([train_op, loss], feed_dict={x:x_data, y:y_data})
```

Example: linear regression with Tensorflow

Finish: Linear regression with TensorFlow, optimization

c) Now let TensorFlow optimize the parameters in cell 7. Modify cell 7 with the right feeding data.

Hint: Look at the learning rate

d) Draw the linreg graph and compare your graph with the Tensorboard graph.

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