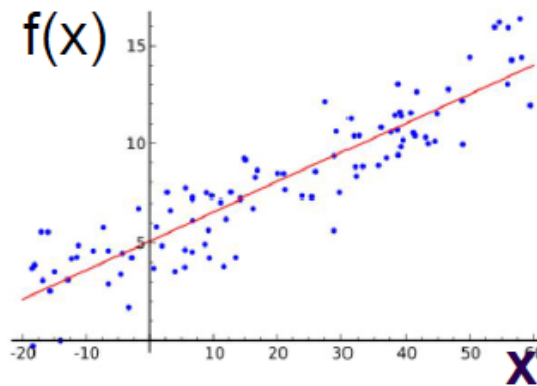# INTRODUCTION TO DATA SCIENCE

25/01/2023

WFAiS UJ, Informatyka Stosowana

I stopień studiów

What are Multivariate Techniques

→ Many things … starting from "linear regression" …
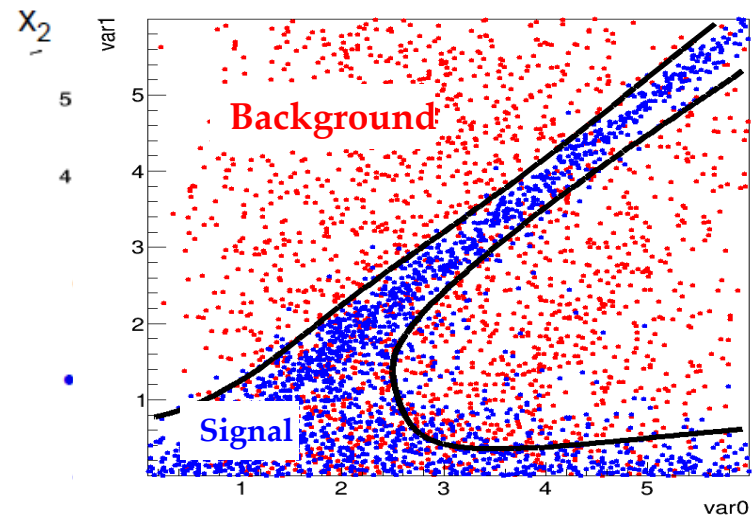

to multivariate event classification
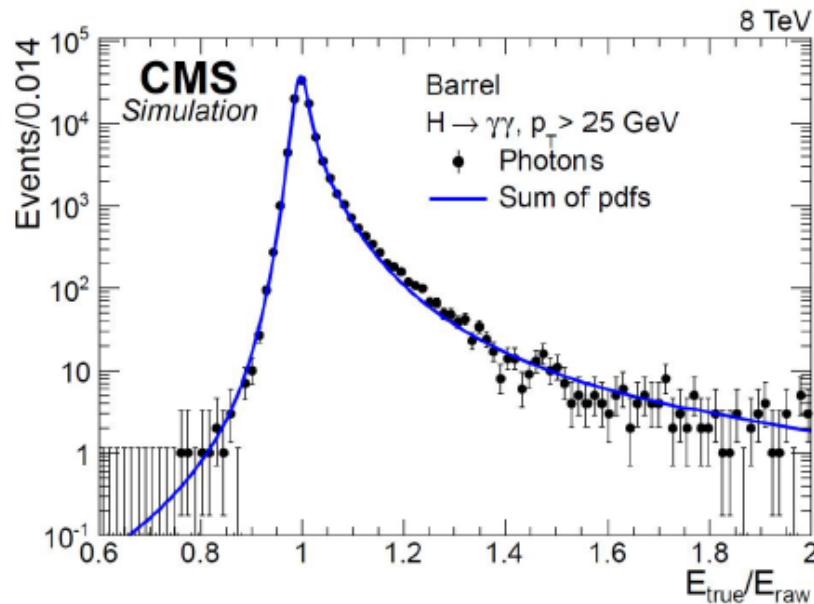

→ or w/o prior 'analytic' model

→ typically "multivariate"

- Parameters depend on the 'joint distribution' $f(x_1, x_2)$

- 'learning from experience' → known data points

Machine Learning - Multivariate Techniques

- ▪ **fitted (non-)analytic function may approximate:**

  - ▪ **target value → 'regression'**

    (e.g. calorimeter calibration/correction function)

MC sample: γ +jets

- ▪ **Raw energy in crystals, η, Φ**

- ▪ **Cluster shape variables**

- ▪ **Local cluster position variables**

  (energy leakage)

- ▪ **Pile-up estimators**

→ **predict energy correction (i.e. parameters in crystal-ball: pdf for energy measurement)**

Regression

- 'known measurements" → model "functional behaviour"

- e.g. : photon energy as function "D"-variables: ECAL shower parameters + …
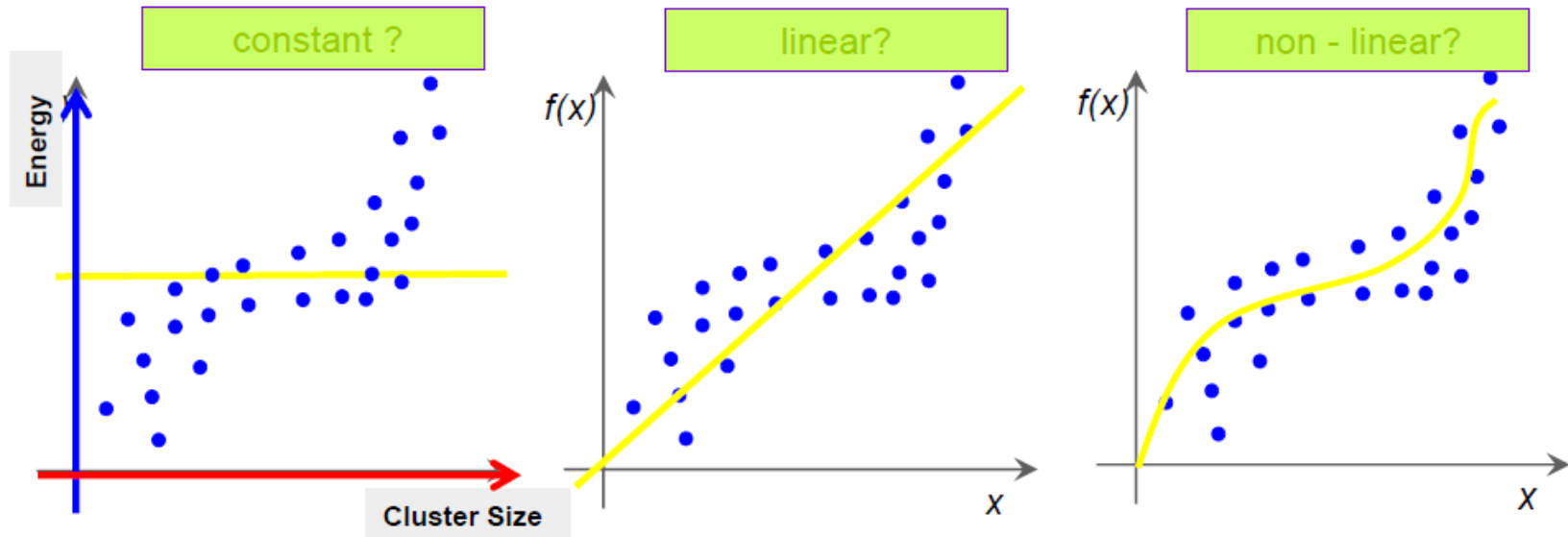
- known analytic model (i.e. nth -order polynomial) → Maximum Likelihood Fit)
- no model ?
  - → "draw any kind of curve" and parameterize it?

- **seems trivial ?** → human brain has very good pattern recognition capabilities!

- what if you have **many** input variables?

Regression -> model functional behaviour
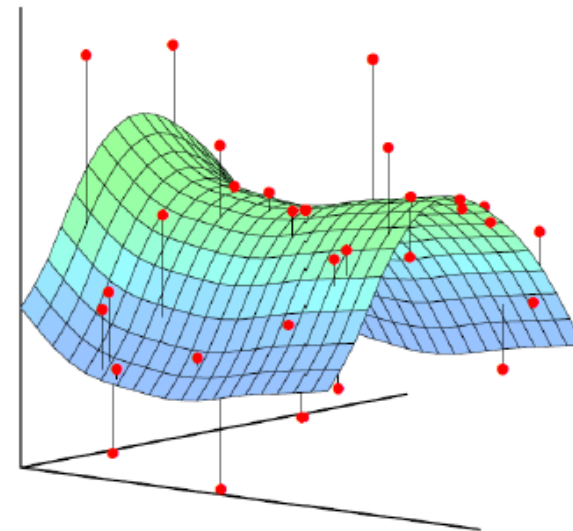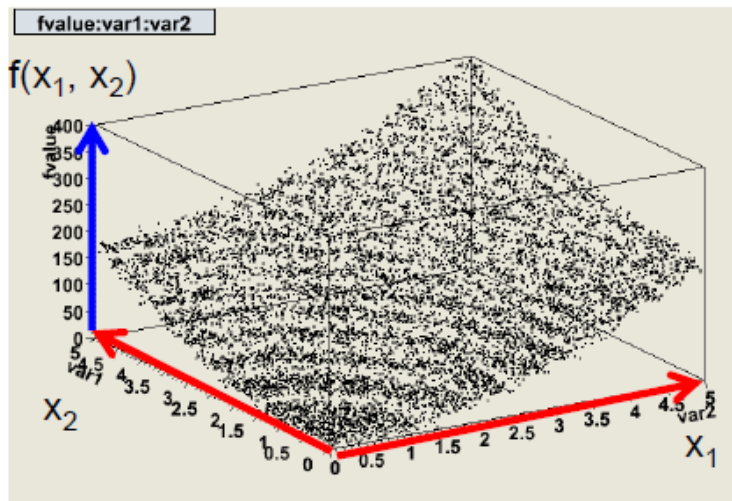
- "standard" regression → fit a known analytic function

    - *e.g.* $f(\mathbf{x}) = ax_1^2 + bx_2^2 + c$

- BUT most times: don't have a reasonable "model" ?   → need something more general:

    - *e.g.* piecewise defined splines, kernel estimators, decision trees to approximate $f(x)$

Note: we are not interested in the 'fitted parameter(s)', <u>it is not:</u> "Newton deriving F=m·a"
→ just provide prediction of function values $f(x)$ for new measurements x

Multi-Variate Classification

Consider events which can be either signal or background events.

Each event is characterized by n observables:

$$\vec{x} = (x_1, ..., x_n)$$ "feature vector"

Goal: classify events as signal or background in an optimal way.

This is usually done by mapping the feature vector to a single variable, i.e., to scalar test statistic:

$$\mathbb{R}^n \to \mathbb{R} : \quad y(\vec{x})$$

A cut $y > c$ to classify events as signal corresponds to selecting a potentially complicated hyper-surface in feature space. In general superior to classical "rectangular" cuts on the x_i.

Problem closely related to *machine learning* (*pattern recognition*, *data mining*, …)

Classification: Different Approaches
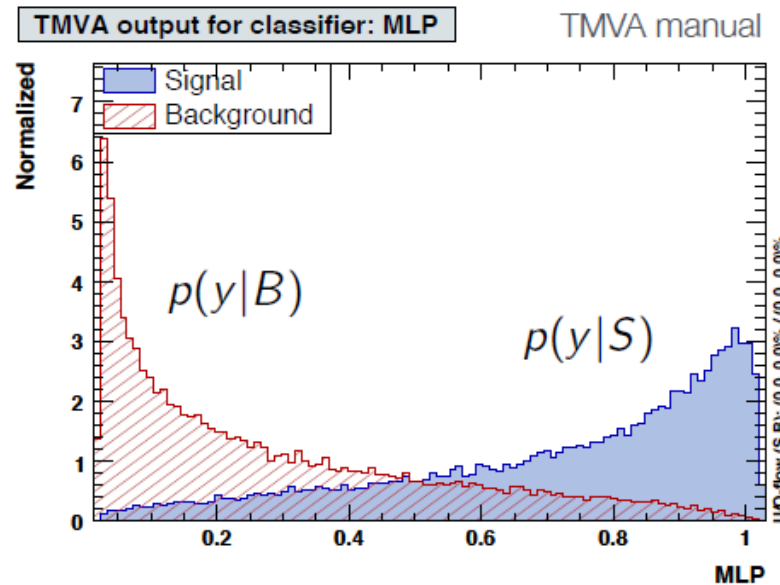

rectangular cuts

non linear

linear
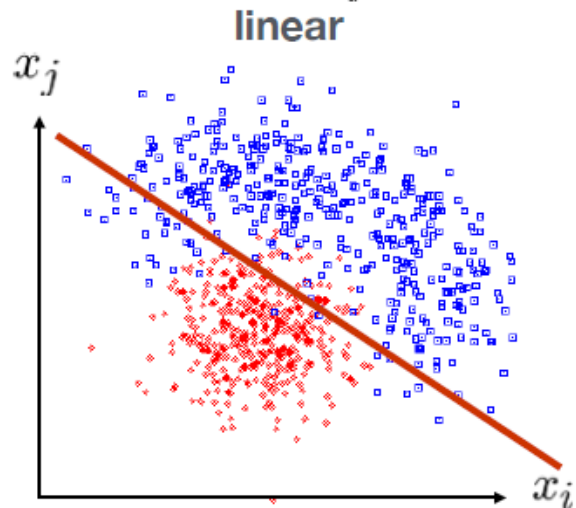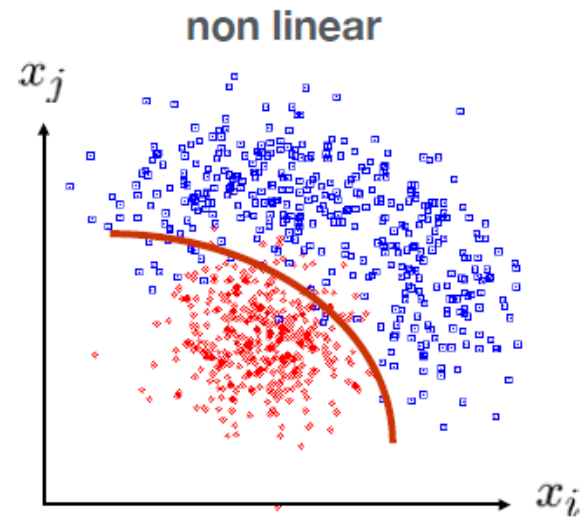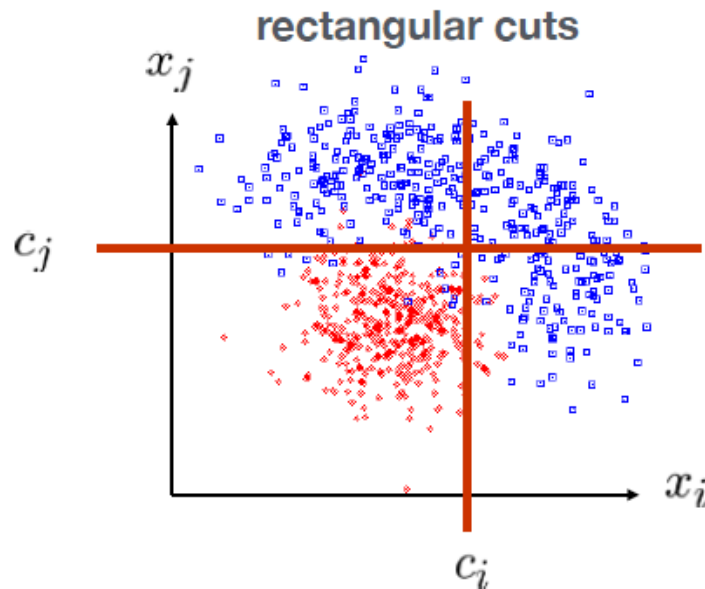
k-Nearest-Neighbor,
Boosted Decision Trees,
Multi-Layer Perceptrons,
Support Vector Machines

…

Signal Probability Instead of Hard Decisions

Example: test statistic y for signal and background from a Multi-Layer Perceptron (MLP):

Instead of a hard yes/no decision one can also define the probability of an event to be a signal event:

$$P_s(y) \equiv P(S|y) = \frac{p(y|S) \cdot f_s}{p(y|S) \cdot f_s + p(y|B) \cdot (1 - f_s)}, \qquad f_s = \frac{n_s}{n_s + n_b}$$

Machine learning: Basic terminology

The goal of machine learning is to predict results based on incoming data.

Features (also parameters, or variables): these are the factors for a machine to look at. E.g.: carthesian coordinates, pixel colors, a car mileage, user's gender, stock price, word frequency in the text.

- Quantitative (x = {1.02, 0.21, 0.12, 2})

- Qualitative *discrete* (x = {medium, small, large}) or *categorical* (x={red, blue, green})

Algorithms (also models): Any problem can be solved in different ways. The method you choose affects the precision, performance, and size of the final model.

- If the data is insufficient/inapproriate (e.g. statistically limited or missing important features), even the best algorithm won't help. Pay attention to the accuracy of your results only when you have a good enough dataset.

CLASSICAL MACHINE LEARNING

Data is pre-categorized or numerical

Data is not labeled in any way

SUPERVISED

UNSUPERVISED

Predict a category

Predict a number

Divide by similarity

Identify sequences

CLASSIFICATION
«Divide the socks by color»

CLUSTERING
«Split up similar clothing into stacks»

Find hidden dependencies

ASSOCIATION
«Find what clothes I often wear together»

REGRESSION
«Divide the ties by length»

DIMENSION REDUCTION (generalization)
«Make the best outfits from the given clothes»

OUR FOCUS

Image credit: https://vas3k.com/blog/machine_learning/

Where are the neural networks?
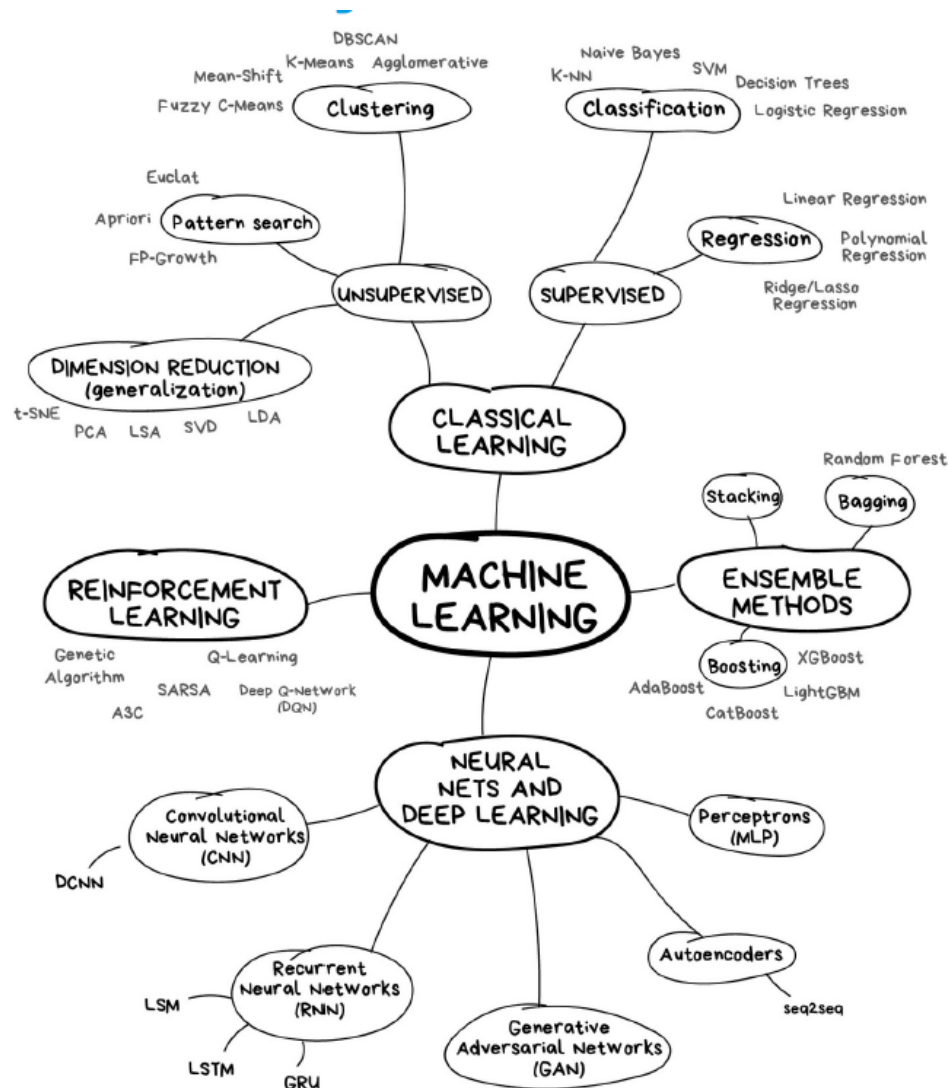
Image credit: https://vas3k.com/blog/machine_learning/

Neural Networks

Any neural network is a collection of **neurons** and **connections** between them.

Neuron is a function with a set of inputs and one output. Its task is to take all numbers from its input, apply a function on them and send the result to the output.

- Example: sum up all numbers from the inputs and if that sum is bigger than N give 1 as a result. Otherwise return zero.

Connections are like channels between neurons. They connect outputs of one neuron with the inputs of another so they can send digits to each other. Each connection has only one parameter the *weight*.

- These weights tell the neuron to respond more to one input and less to another. Weights are adjusted when training — that's how the network learns.

Perceptron

Discriminant:
$$y(\vec{x}) = h\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

The nonlinear, monotonic function h is called *activation function*.

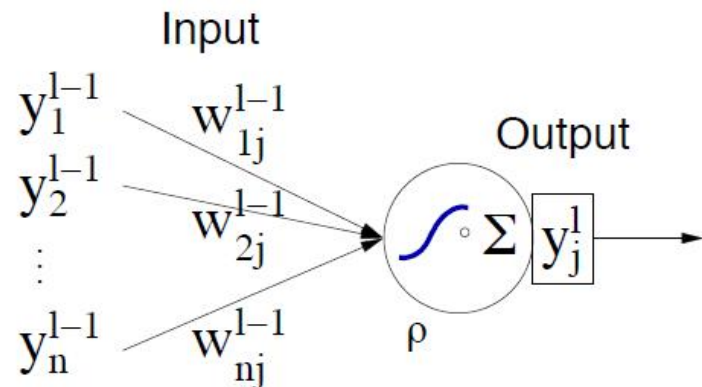Typical choices for h: $\dfrac{1}{1+e^{-x}}$ ("sigmoid"), $\tanh x$
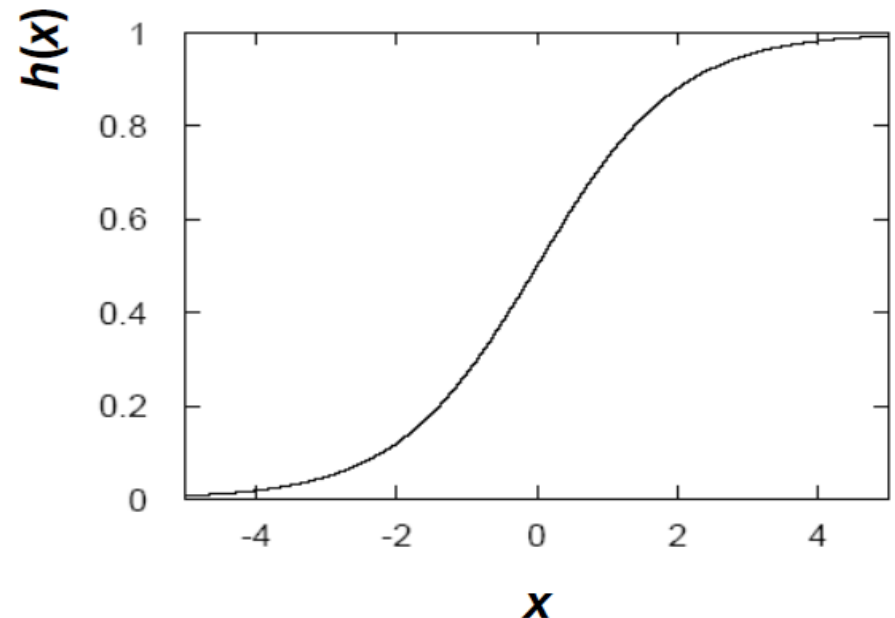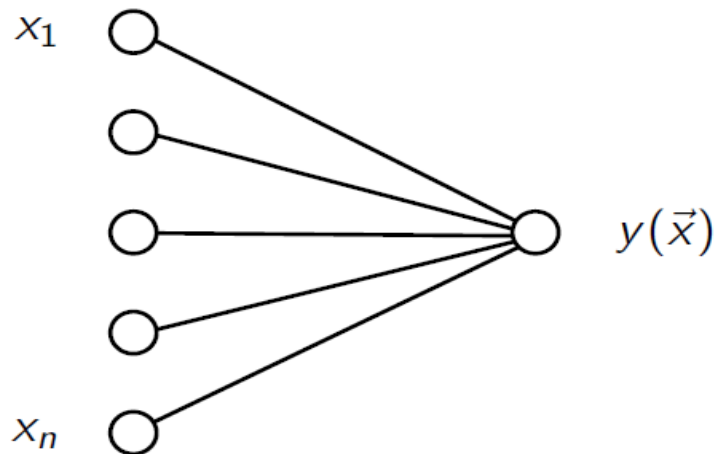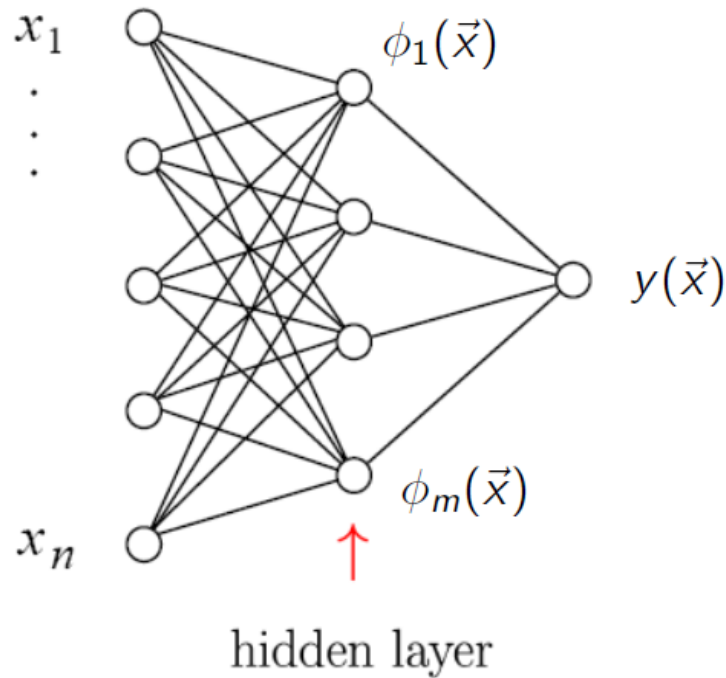
The Biological Inspiration: the Neuron

Cell body

Nucleus

Axon

Telodendria

Axon hillock

Synaptic terminals

Endoplasmic reticulum

Golgi apparatus

Mitochondrion

Dendrite

Dendritic branches
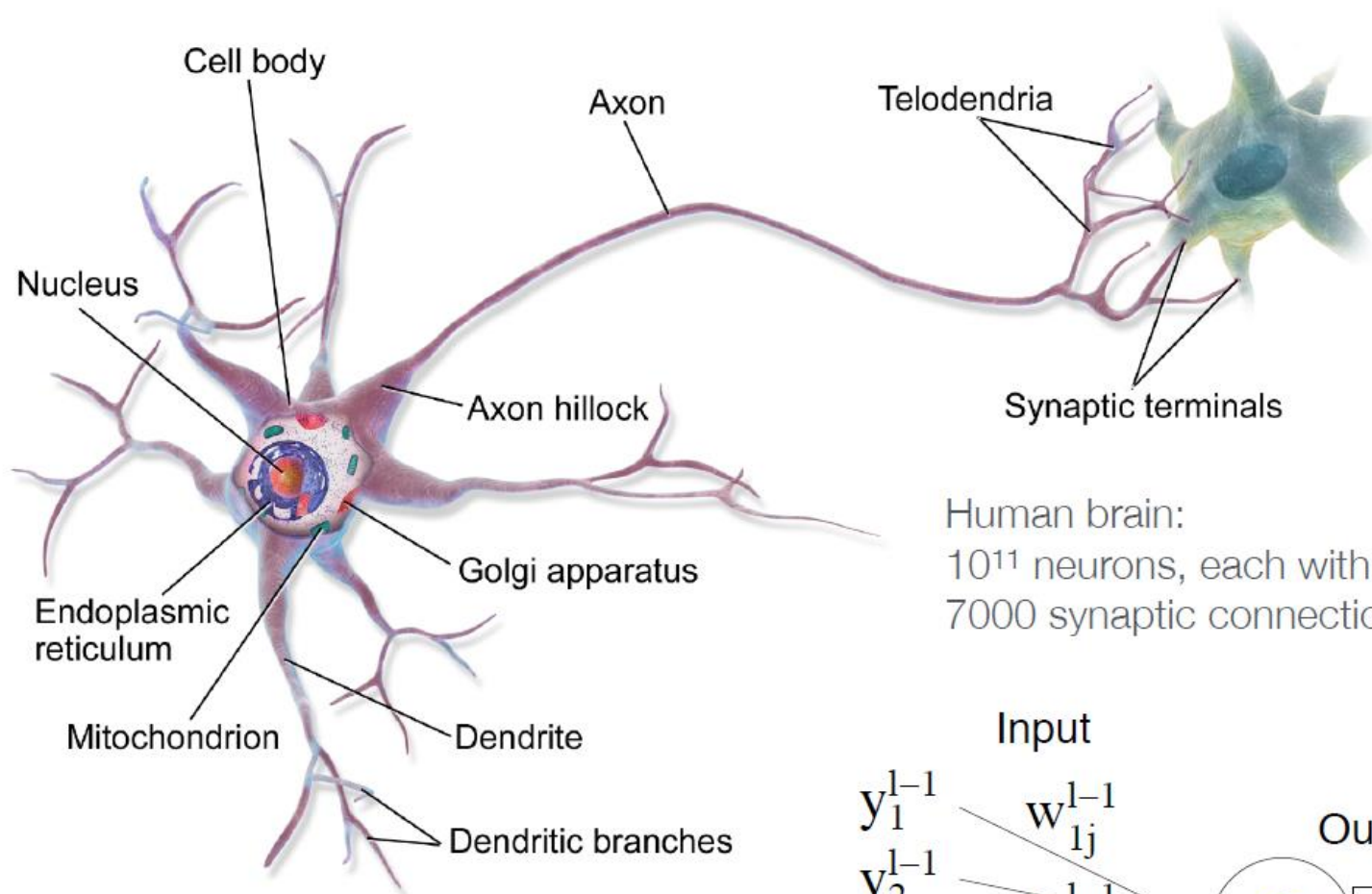
Human brain:
10^{11} neurons, each with on average 7000 synaptic connections

https://en.wikipedia.org/wiki/Neuron

Input

y_1^{l-1} w_{1j}^{l-1}

y_2^{l-1} w_{2j}^{l-1}

$\vdots$

y_n^{l-1} w_{nj}^{l-1}

Output

$\int \circ \Sigma$ y_j^l

ρ

Feedforward Neural Network with One Hiden Layer

$$\phi_i(\vec{x}) = h\left(w_{i0}^{(1)} + \sum_{j=1}^{n} w_{ij}^{(1)} x_j \right)$$

superscripts indicates layer number

$$y(\vec{x}) = h\left(w_{10}^{(2)} + \sum_{j=1}^{m} w_{1j}^{(2)} \phi_j(\vec{x}) \right)$$

hidden layer

Straightforward to generalize to multiple hidden layers

Network Training

$$\vec{x}_a : \text{ training event, } a = 1, ..., N$$

$$t_a : \text{ correct label for training event } a$$

e.g., $t_a = 1, 0$ for signal and background, respectively

$$\vec{w} : \text{ vector containing all weights}$$

Error function:

$$E(\vec{w}) = \frac{1}{2} \sum_{a=1}^{N} (y(\vec{x}_a, \vec{w}) - t_a)^2 = \sum_{a=1}^{N} E_a(\vec{w})$$

Weights are determined by minimizing the error function.

Backpropagation

Start with an initial guess $\vec{w}^{(0)}$ for the weights an then update weights after each training event:

$$\vec{w}^{(\tau+1)} = \vec{w}^{(\tau)} - \eta \nabla E_a(\vec{w}^{(\tau)})$$

$\underline{\qquad}$ learning rate

Let's write network output as follows:

$$y(\vec{x}) = h(u(\vec{x})) \;\; \text{with} \;\; u(\vec{x}) = \sum_{j=0}^{m} w_{1j}^{(2)} \phi_j(\vec{x}), \;\; \phi_j(\vec{x}) = h\left(\sum_{k=0}^{n} w_{jk}^{(1)} x_k\right) \equiv h(v_j(\vec{x}))$$

Here we defined $\phi_0 = x_0 = 1$ and the sums start from 0 to include the offsets.

Weights from hidden layer to output:

$$E_a = \frac{1}{2}(y_a - t_a)^2 \;\rightarrow\; \frac{\partial E_a}{\partial w_{1j}^{(2)}} = (y_a - t_a)h'(u(\vec{x}_a))\frac{\partial u}{\partial w_{1j}^{(2)}} = (y_a - t_a)h'(u(\vec{x}_a))\phi_j(\vec{x}_a)$$

Weights from input layer to hidden layer ($\rightarrow$ further application of chain rule):

$$\frac{\partial E_a}{\partial w_{jk}^{(1)}} = (y_a - t_a)h'(u(\vec{x}_a))w_{1j}^{(2)}h'(v_j(\vec{x}_a))x_{a,k} \qquad\qquad \vec{x}_a \equiv (x_{a,1}, \ldots, x_{a,n})$$

Neural Network Output and Decision Boundaries

a Input layer — Hidden layer — Output layer

x_1, x_2, x_3, x_d

x_l h_j $0(x) = f(x,w)$

b Signal / Background — output of neural network

Number of events — NN output

c Signal / Background / NN contours
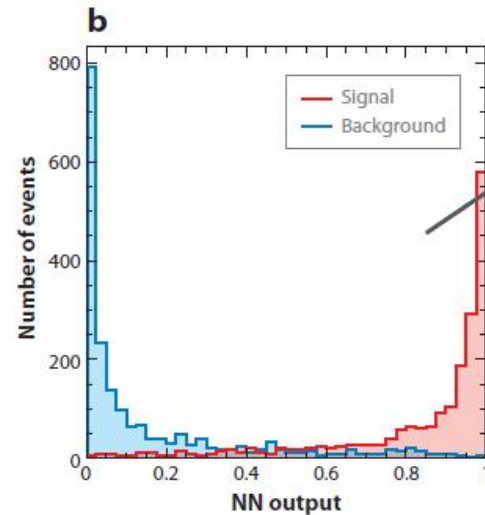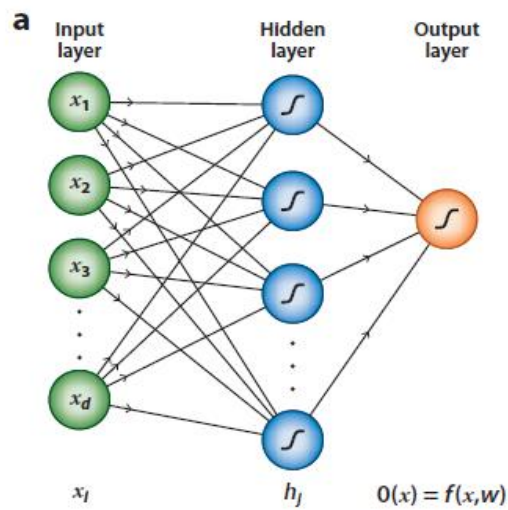
decision boundaries for different cuts on NN output

Variable x_2 — Variable x_1

d signal probability $p(s \mid x_1, x_2)$

$p(s|x_1, x_2)$ — Variable x_2 — Variable x_1
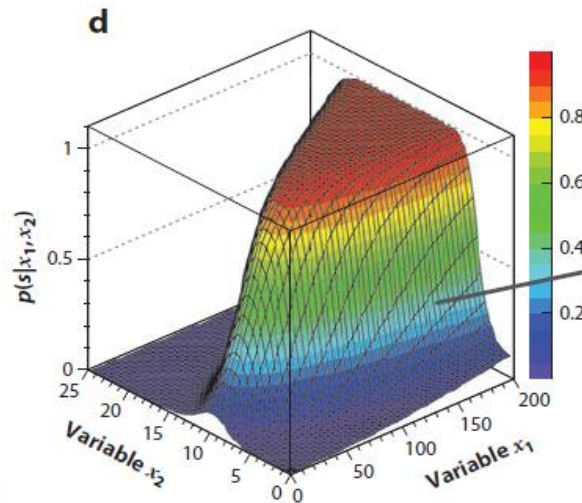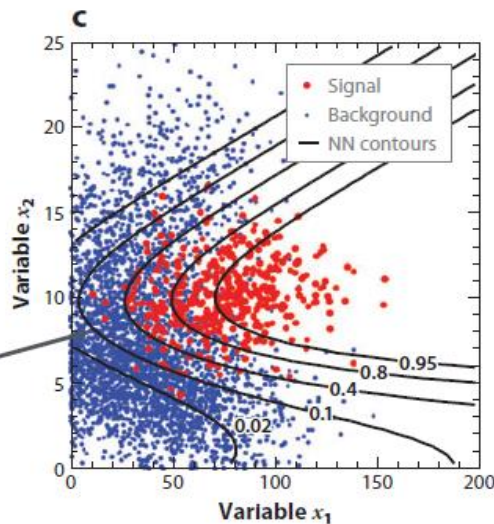
Example of Overtraining

Too many neurons/layers make a neural network too flexible
→ overtraining
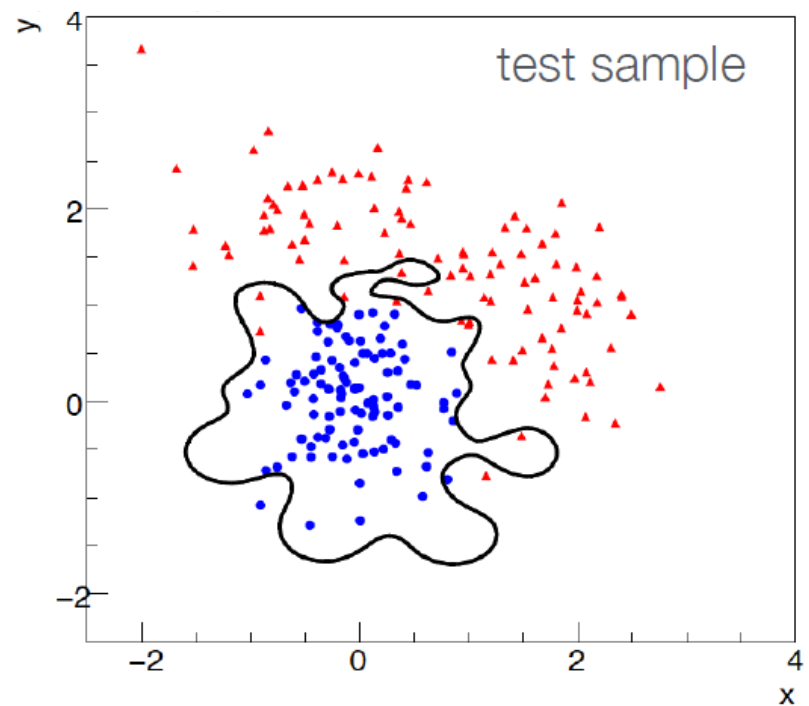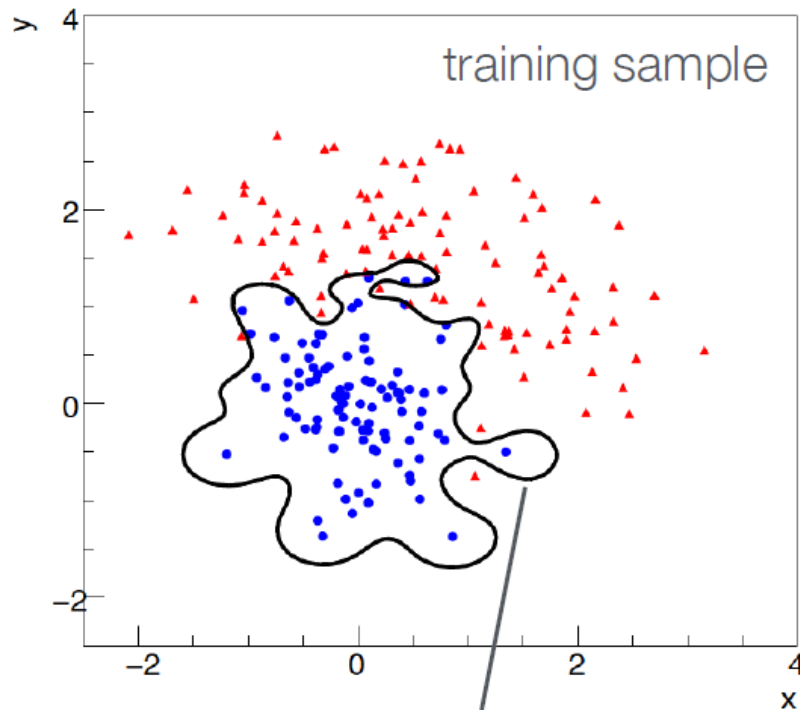
Network "learns" features that are merely
statistical fluctuations in the training sample

Monitoring Overtraining

Monitor fraction of misclassified events (or error function:)

optimum = minimum of error rate for test sample

overtraining = increase of error rate

test sample

training sample

flexibility (e.g., number of nodes/layers)
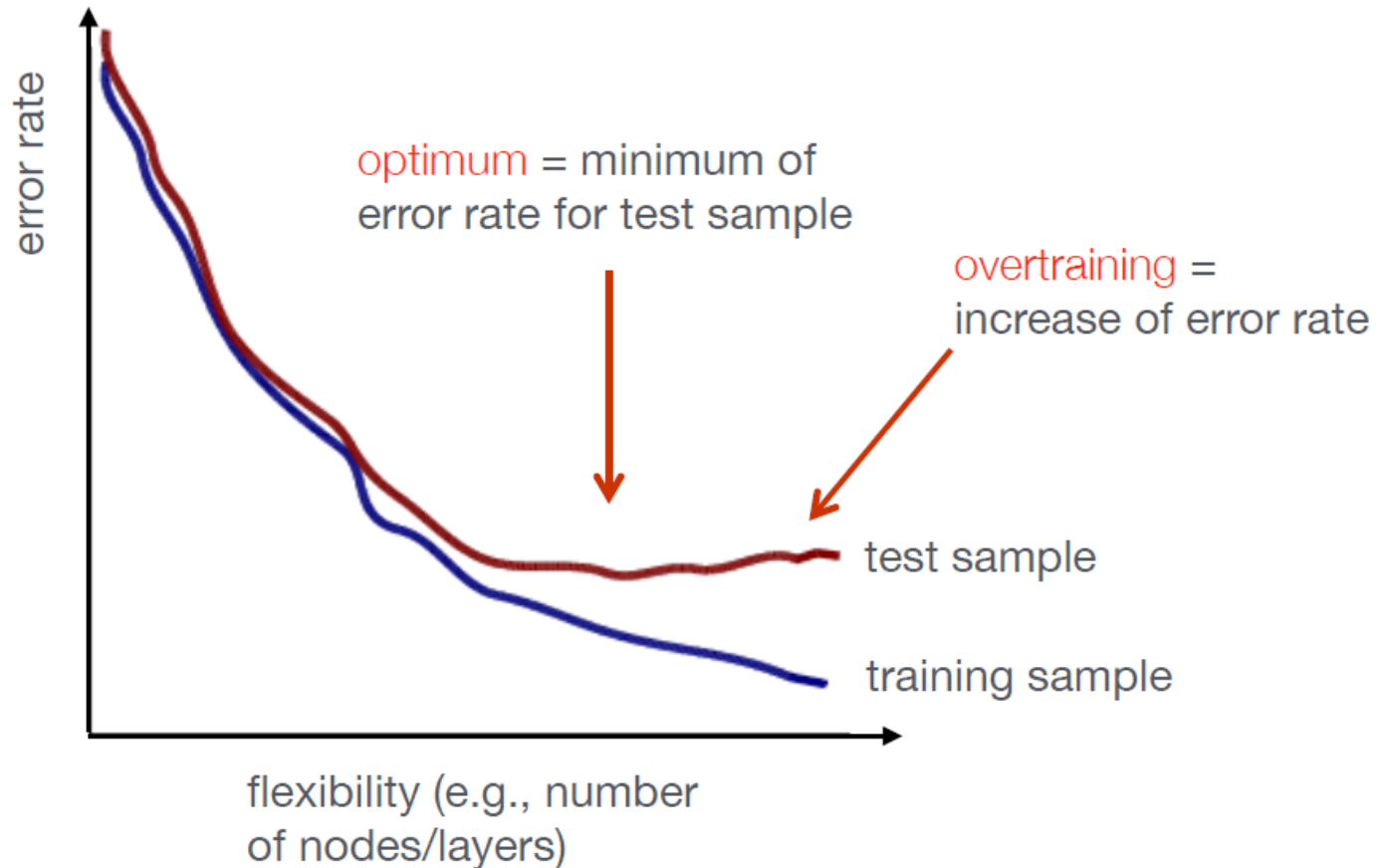
Deep Neural Networks

Deep networks: many hidden layers with large number of neurons

Challenges

- ▸ Hard too train ("vanishing gradient problem")
- ▸ Training slow
- ▸ Risk of overtraining

Big progress in recent years

- ▸ Interest in NN waned before ca. 2006
- ▸ Milestone: paper by G. Hinton (2006): "learning for deep belief nets"
- ▸ Image recognition, AlphaGo, …
- ▸ Soon: self-driving cars, …
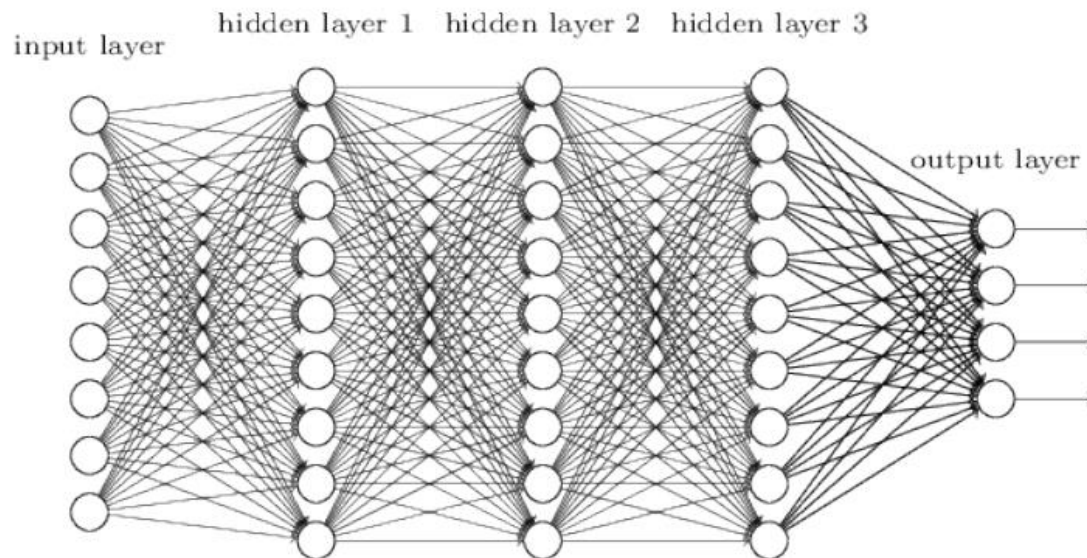
How do NNs work?

How do NNs work?

$$a_0^{(1)} = f(w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} + \ldots + w_{0,n}\, a_n^{(0)} + b_0)$$

layer

activation function

weights

bias
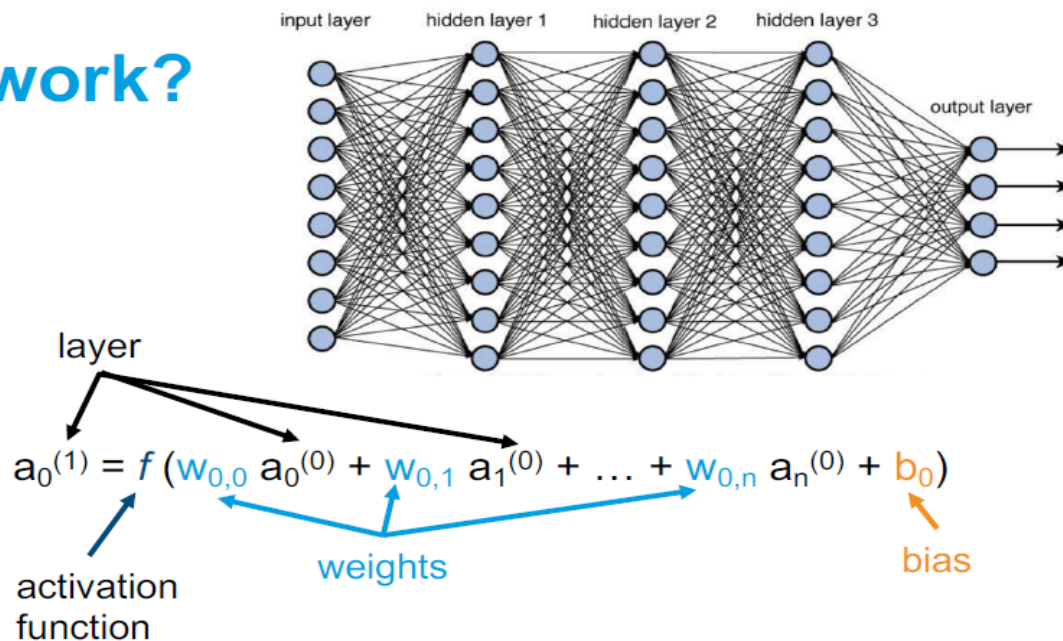
$$\begin{bmatrix} a_0^{(1)} \\ \ldots \\ a_n^{(1)} \end{bmatrix} = f\left(\begin{bmatrix} w_{0,0} & \cdots & w_{0,n} \\ \vdots & \ddots & \vdots \\ w_{k,0} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ \ldots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ \ldots \\ b_n \end{bmatrix} \right)$$

How do NNs learn?

After we constructed a network, our task is to **assign proper weights** so neurons will react correctly to incoming signals.

- define a loss function to measure how far the response is from the truth

This function is a function of all the weights and biases in the NN (a priori a very large number), and the goal of training is to find its minimum.

- To start with, all weights are assigned randomly.

- After evaluating the NN on the training dataset, we can compute all the per-neuron differences with respect to the correct result.

- Computing the gradient of the loss, gives us a direction in which to tune the weights towards a local minimum

The process of correcting the weights is called backpropagation an error.

How do NNs learn?

A mostly complete chart of
Neural Networks
©2019 Fjodor van Veen & Stefan Leijnen   asimovinstitute.org

25/01/2023
There are many more…
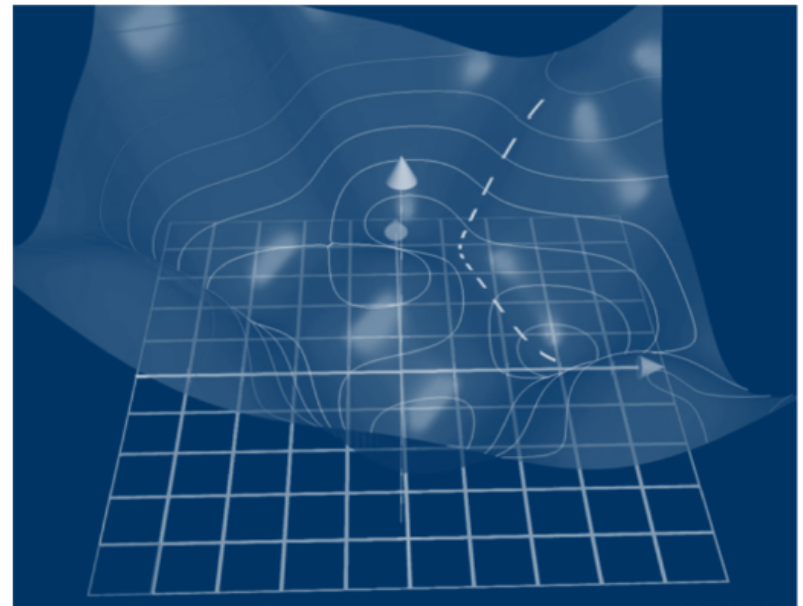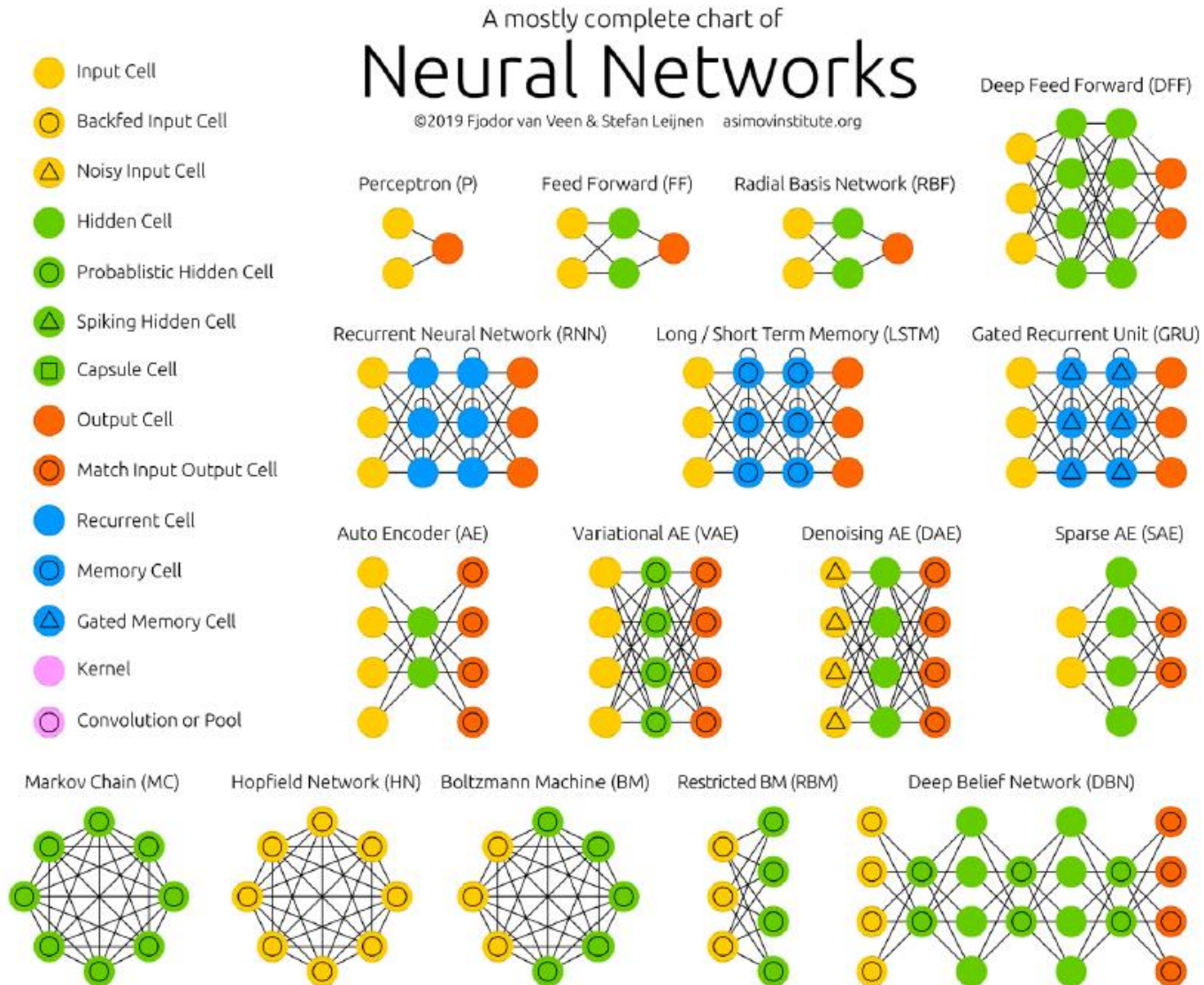
How do NNs learn?

- Each input is multiplied by a weight.
- The weighted values are summed and a bias is added
- **The result is passed to an activation funciton (non liniarity).**

Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
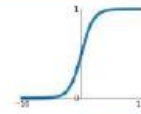
tanh
$\tanh(x)$
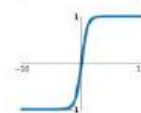
ReLU
$\max(0, x)$

Leaky ReLU
$\max(0.1x, x)$

Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$
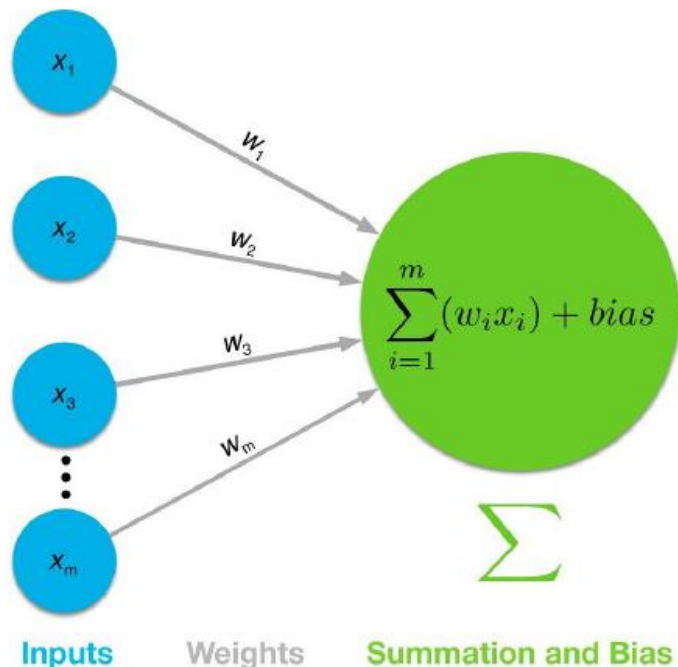
$$\hat{y} = f\left(\sum_{i=1}^{m}(w_i x_i) + bias\right)$$

$$\sum_{i=1}^{m}(w_i x_i) + bias$$

$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$

$\hat{y}$

Thers's a colorful world inside black boxes!

Inputs   **Weights**   **Summation and Bias**   **Activation**   **Output**

Typical Applications

Regression:
Predict a continuous label.
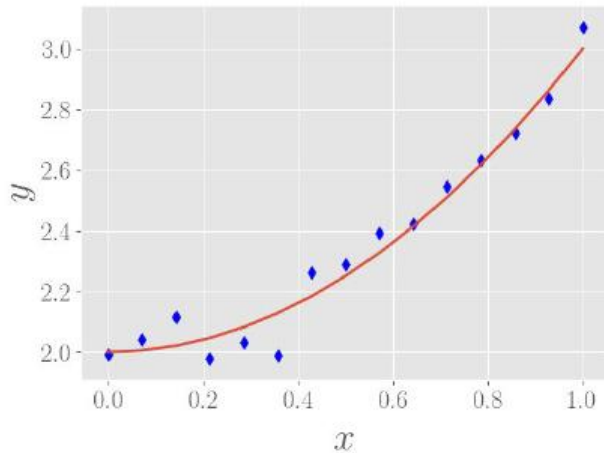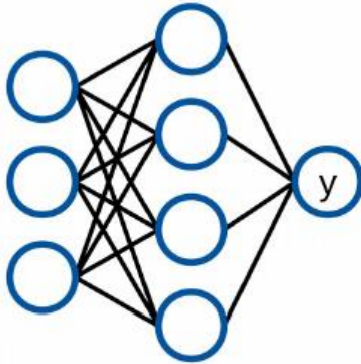
Classification:
Separate events into multiple categories.
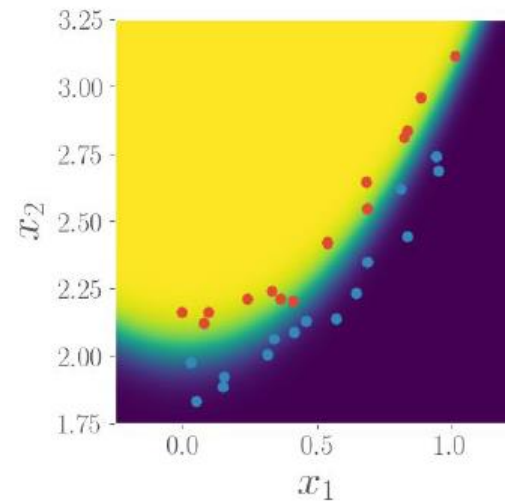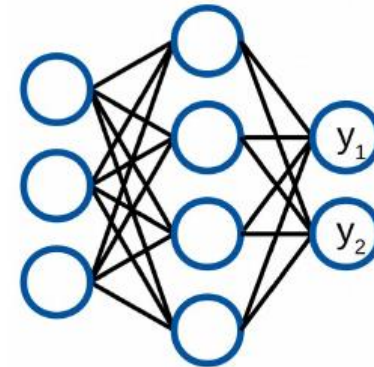
Input Preprocesing

- Input features could have vastly different scales (e.g. p_T vs η).

    - Difficult to find optimal values.

- Basic strategy: Normalize to mean=0 and variance=1.
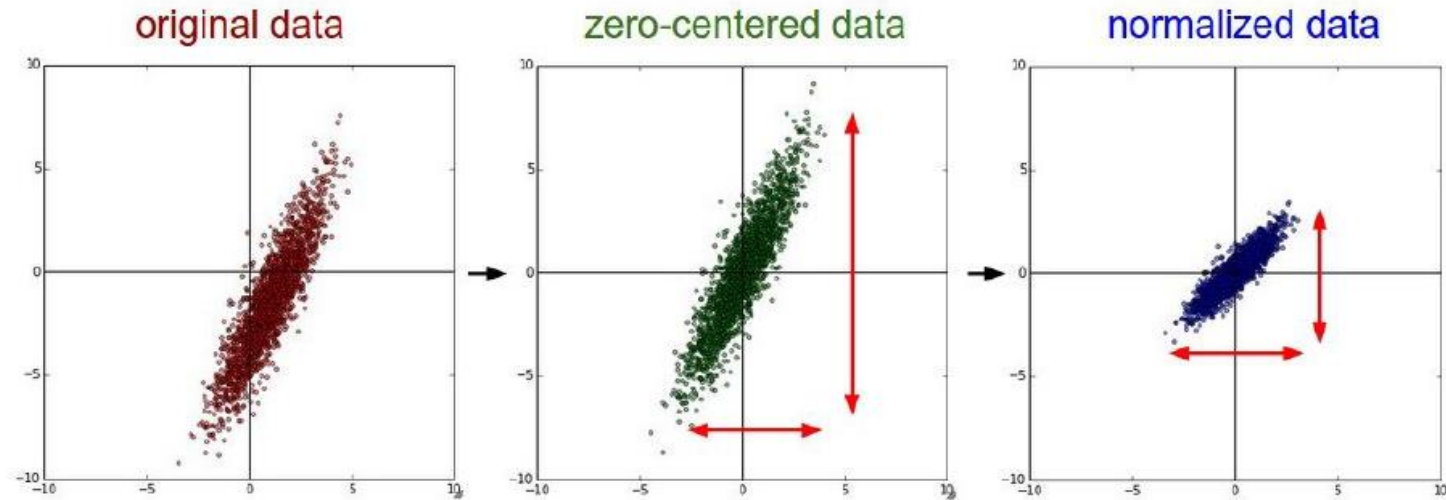
- Other options possible: decorrelation, non-linear transformation, ...

Training

- Training starts specifying an input and a target dataset.

  - For each input set, the target is what the network should learn for that input.

- A loss function is required L(θ):

  - The loss funtion quantifies the mistakes the NN makes. E.g. *mean squared error* can be used for regression.

Training is the minimization of the loss function w.r.t. the NN parameters.

L(θ)

Local min

Global min

θ
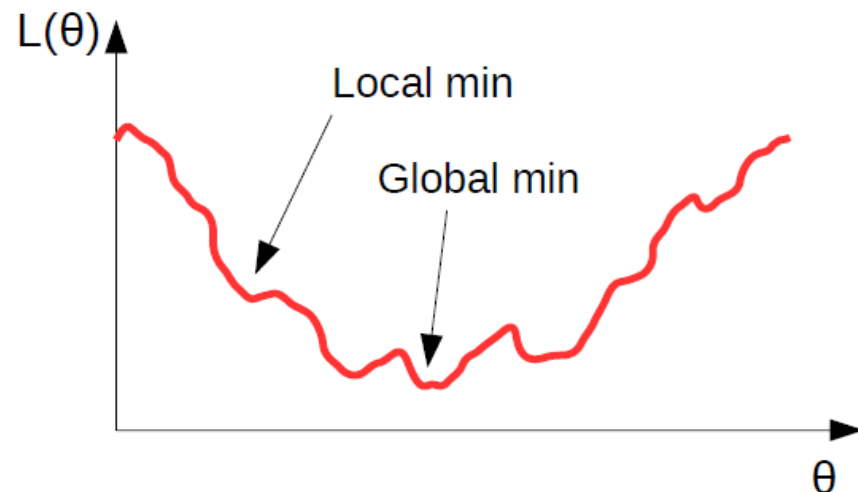
Training: (Stochastic) Gradient Descent

- Given the increasing size of datasets and parameters, it is no more possible to directly minimize the loss function.

Iterative minimization by updating θ in opposite direction of gradient.

$$\theta_i = \theta_{i-1} - \alpha \frac{\partial L}{\partial \theta}$$, where α is the so-called « learning rate ».

- Evaluation and derivation of the loss funciton for the full dataset is costly:

Stochastic gradient descent:
Calculate gradient for a small stochastic subset of the training sample (batch). → This also helps to avoid local minima!
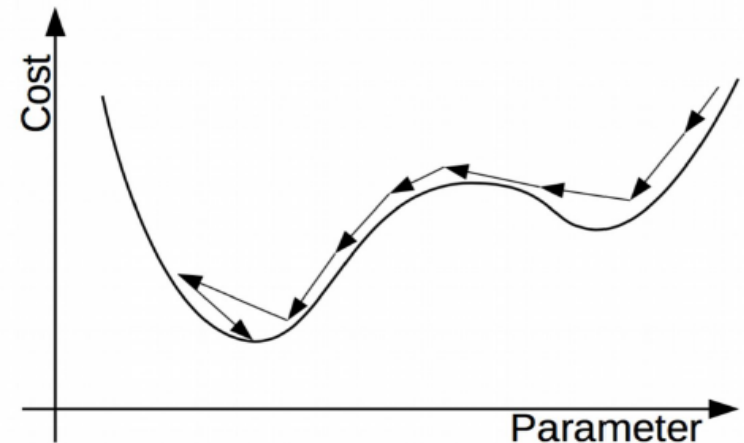
One iteration over the full training dataset is called *epoch*.

Training: more optimisers

- More advanced options than fixed learning rate.

  - ◆ Momentum:
    past gradients used as
    « velocity »

  - ◆ Adaptive methods:
    different learning rates
    for each parameter and
    as a function of past
    gradients.
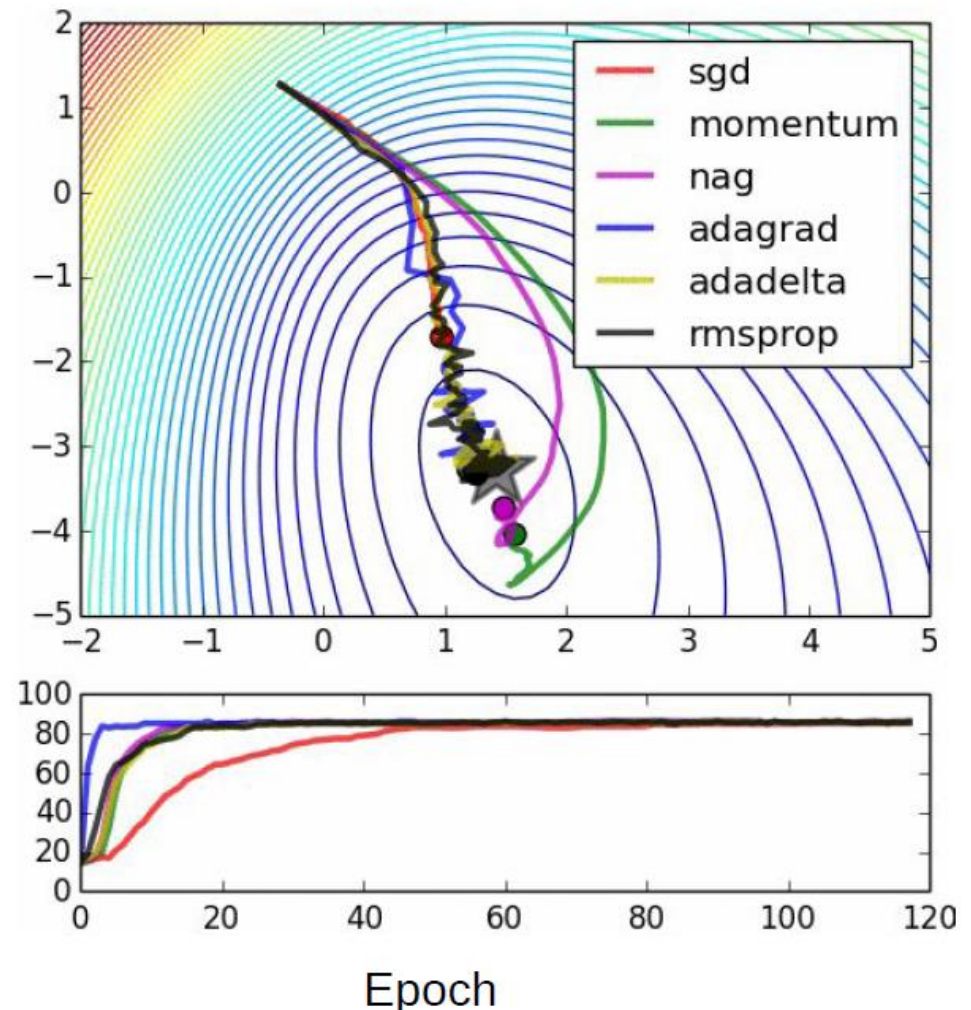
Underfitting and overtraining

Underfitting: If model capacity is too low
or if training is not enough
→ bad performance.

Underfitting
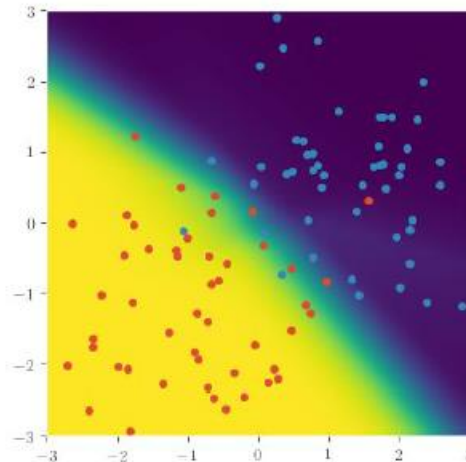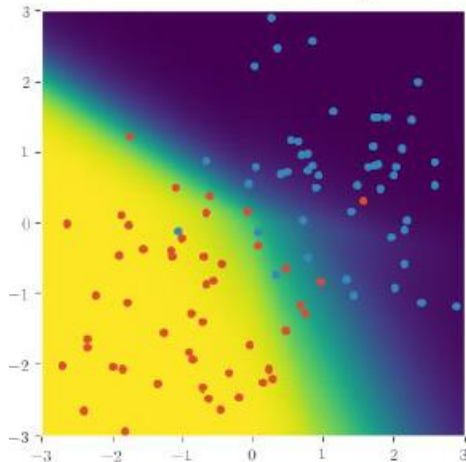
Overfitting
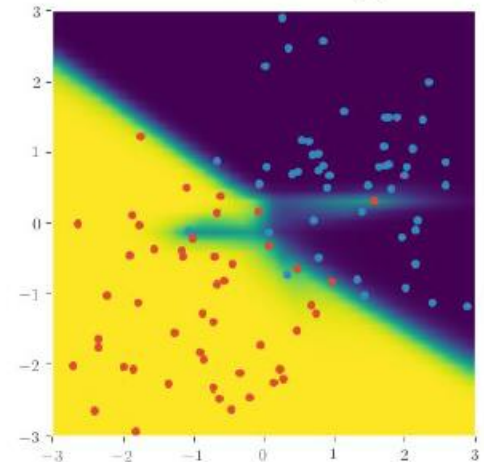
Overfitting: If model capacity is too high,
network can « memorize » training samples
→ bad generalization.

Overtraining solutions

Early stop:
Evaluate the performance of the network on a validation dataset.
Stop when performance on validation set decreases
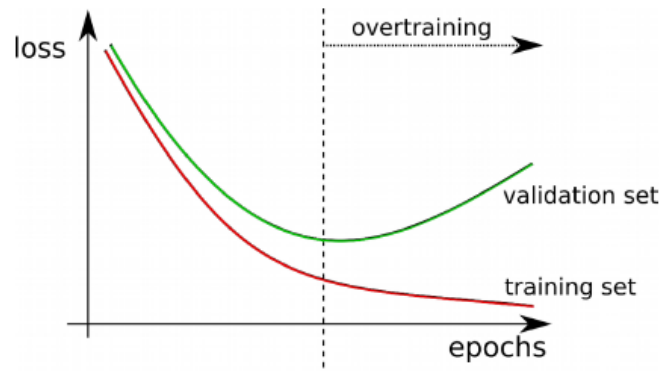
Dropout:
Randomly drop a percentage of nodes at each training step. Learn redundant representations, hence giving a more robust model.
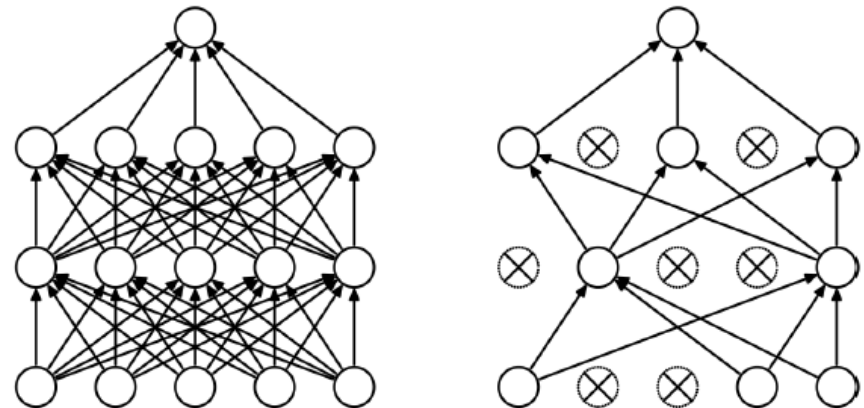
Deep-learning Neural Network

TensorFlow™

MNIST example

Neural network

Since 2010 new era in Machine Learning: rapidly increasing areas of applications
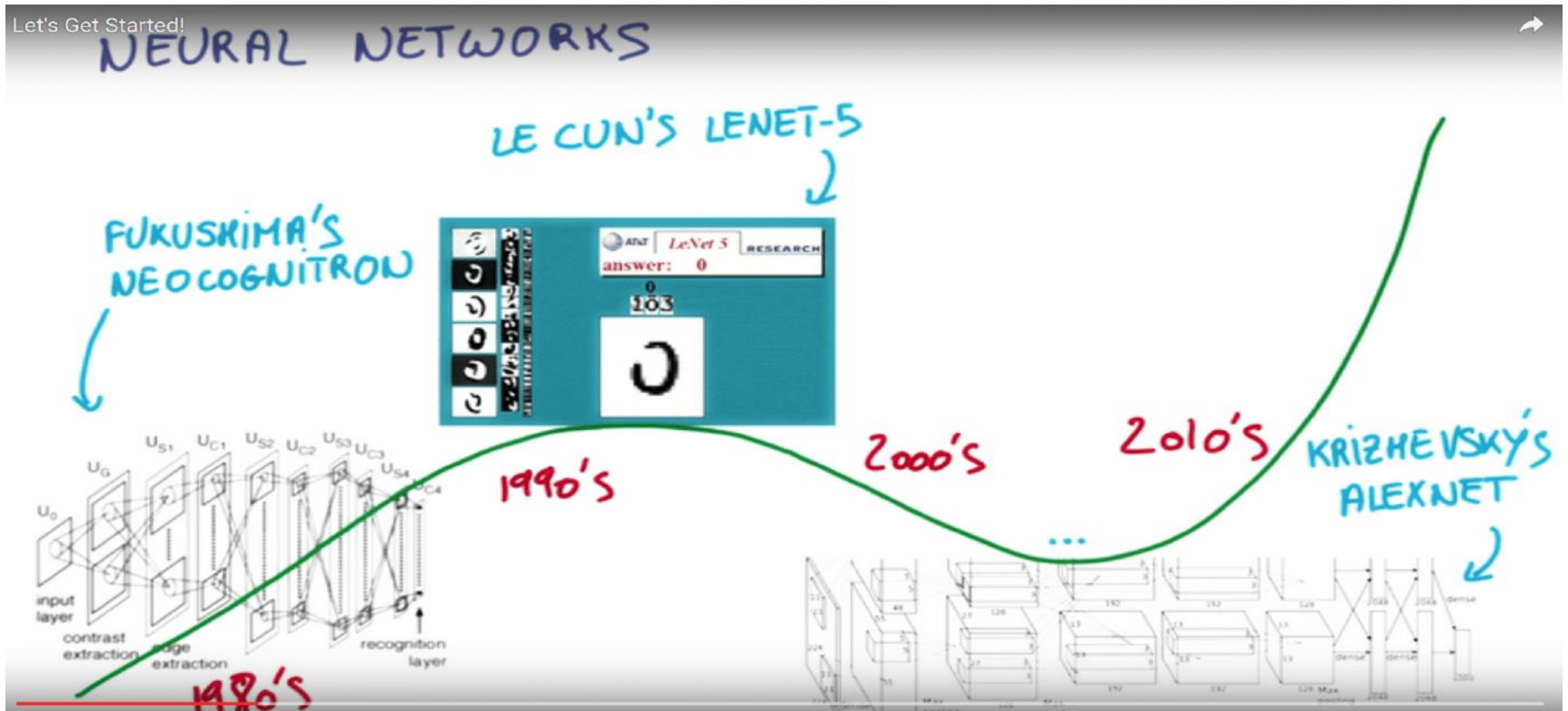
Neural network

Since 2010 new era: rapidly increasing areas of applications

Deep-Learning tutorial @ udacity

https://www.udacity.com/course/deep-learning--ud730
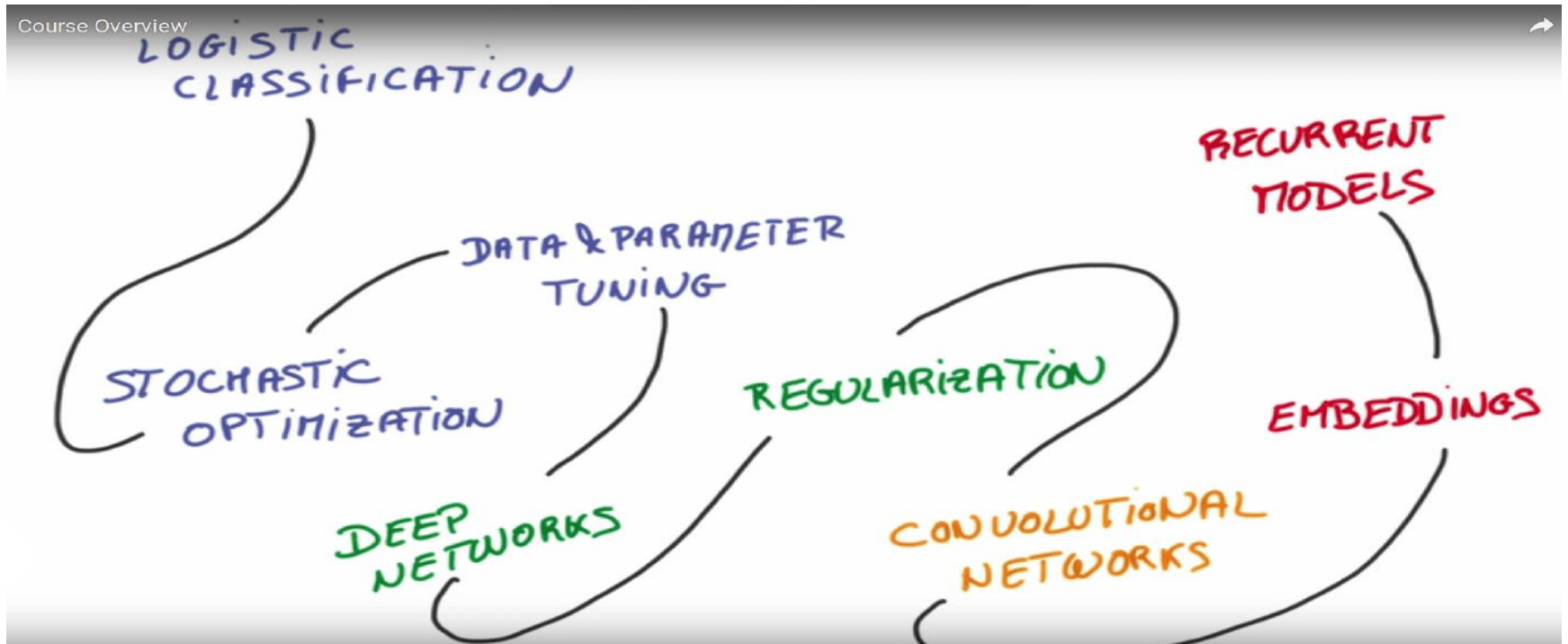
Supervised Classifications
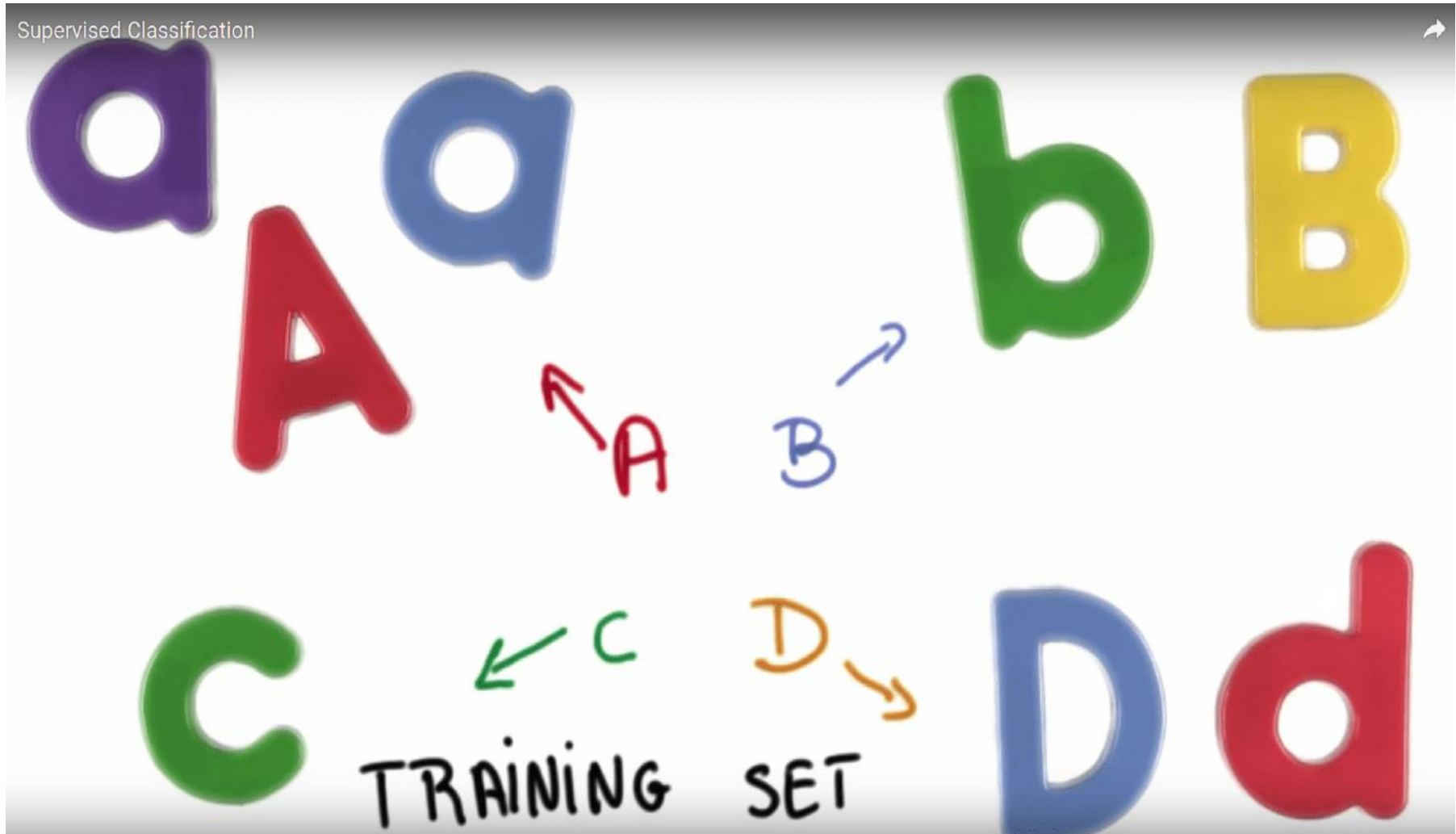
Supervised Classifications
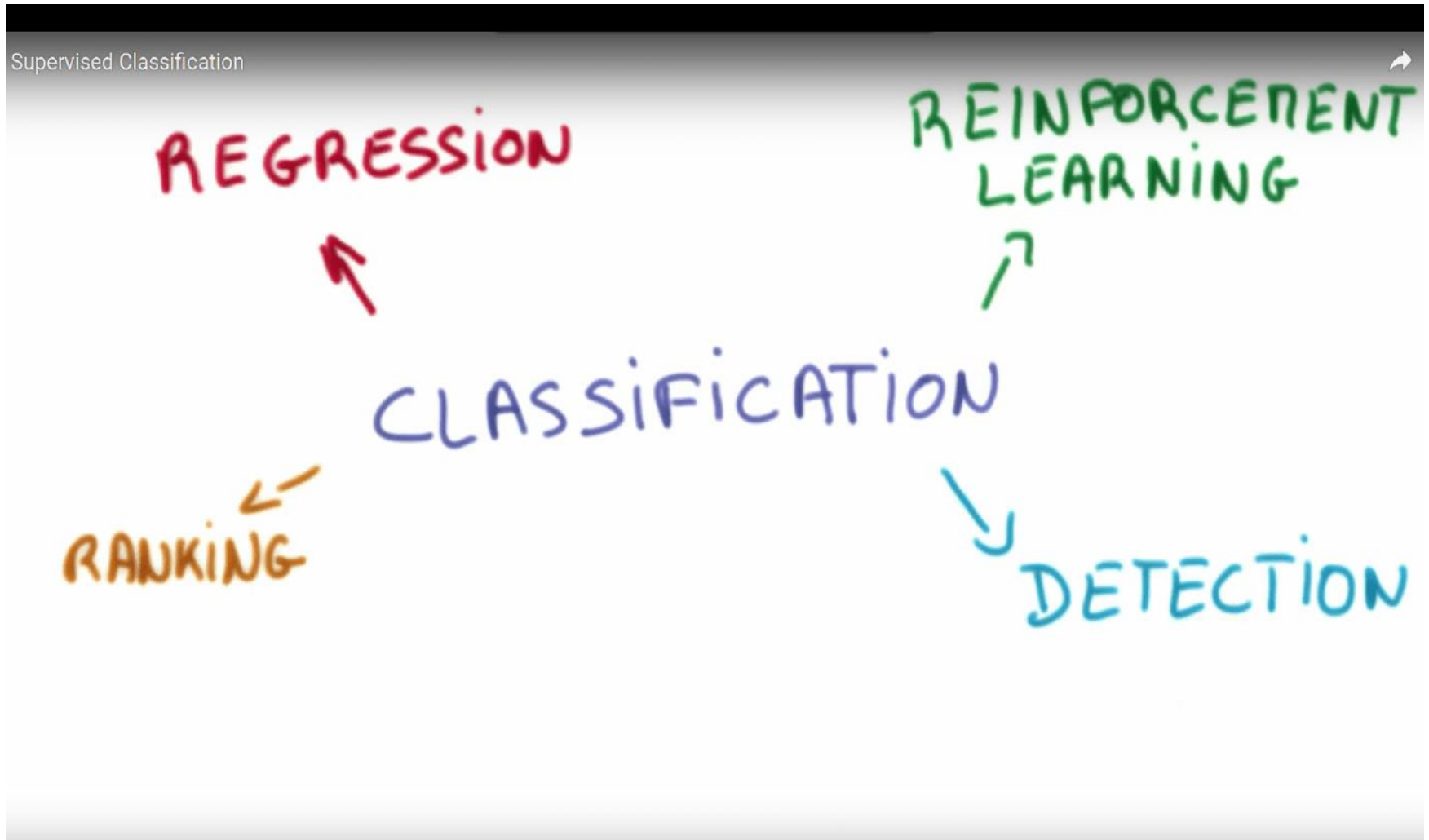
Classifications for Detection

Classifications for Ranking

Classification For Ranking Solution

RANKING

Udacity deep learning

Web   News   Videos   Images   Shopping   More ▾   Search tools

About 46,700 results (0.64 seconds)

Machine Learning: Supervised Learning - Udacity
https://www.udacity.com/wiki/ud675 ▾ Udacity ▾
Oct 3, 2014 - Ethem Alpaydın, Introduction to Machine Learning. Second Edition. ...

CLASSIFIER

RELEVANT !

Use a classifier that takes the pair
(query, web page) as an input, and
classifies the pair as relevant / not relevant

Logistic classifier: Linear model

Softmax
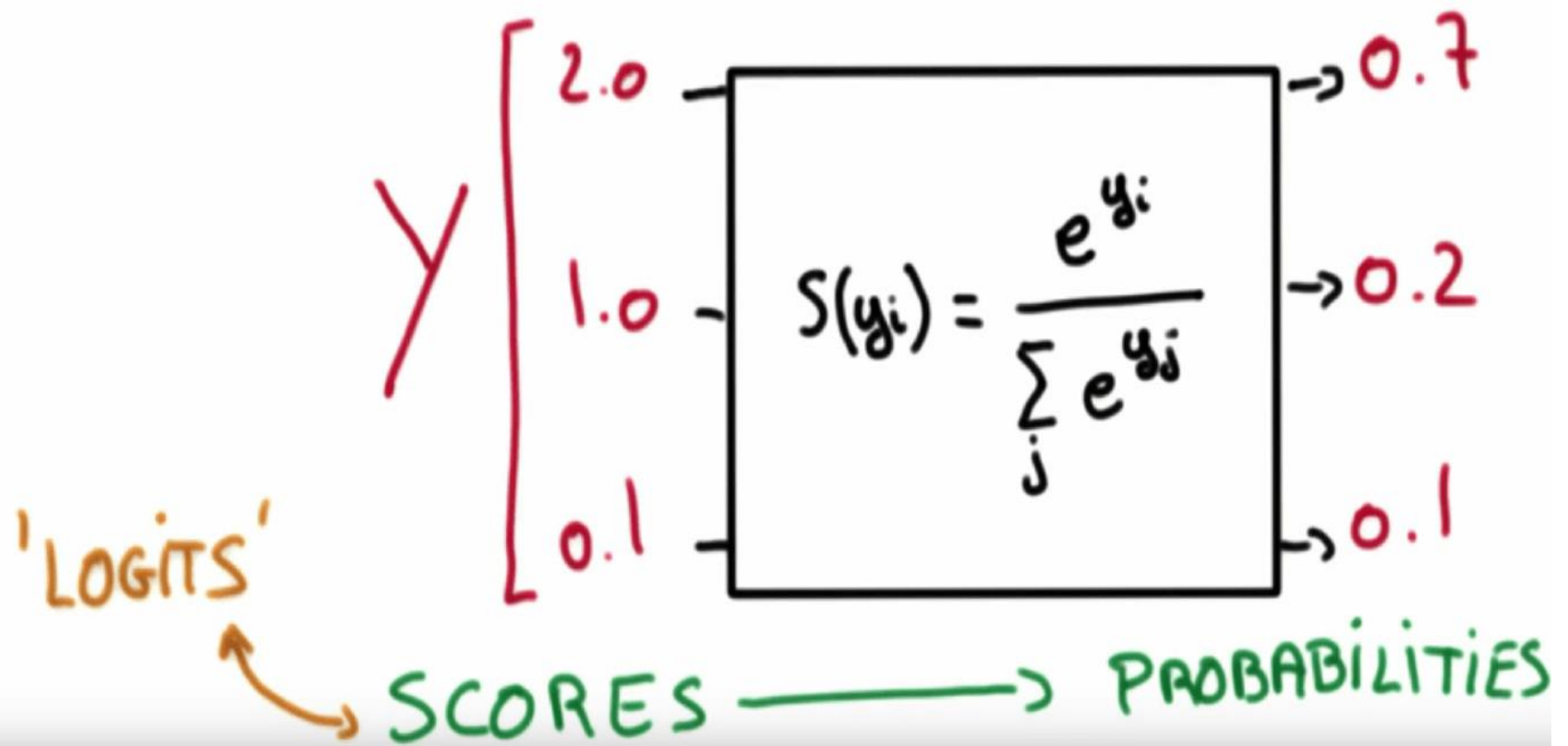
„One hot" encoding
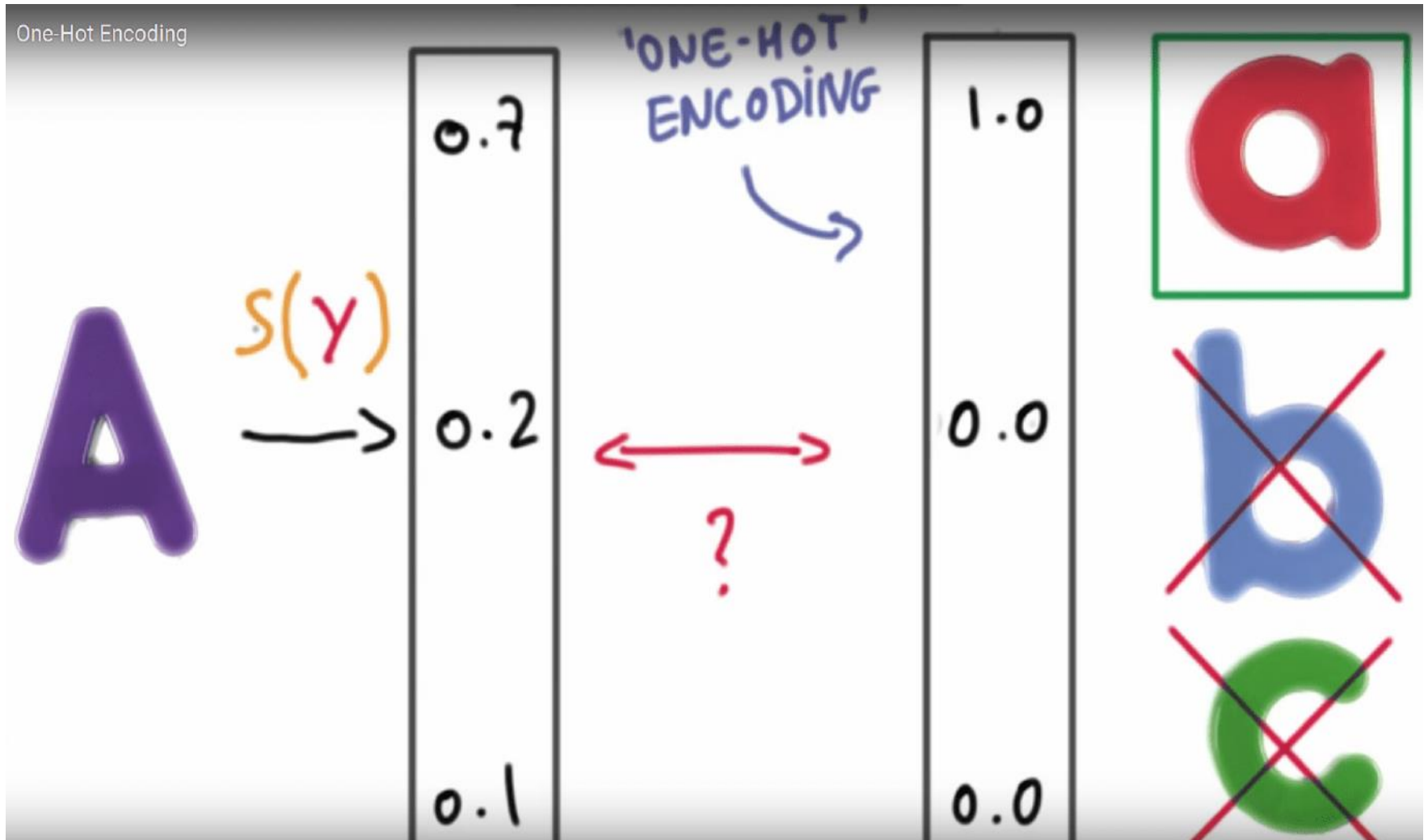
„One hot" encoding
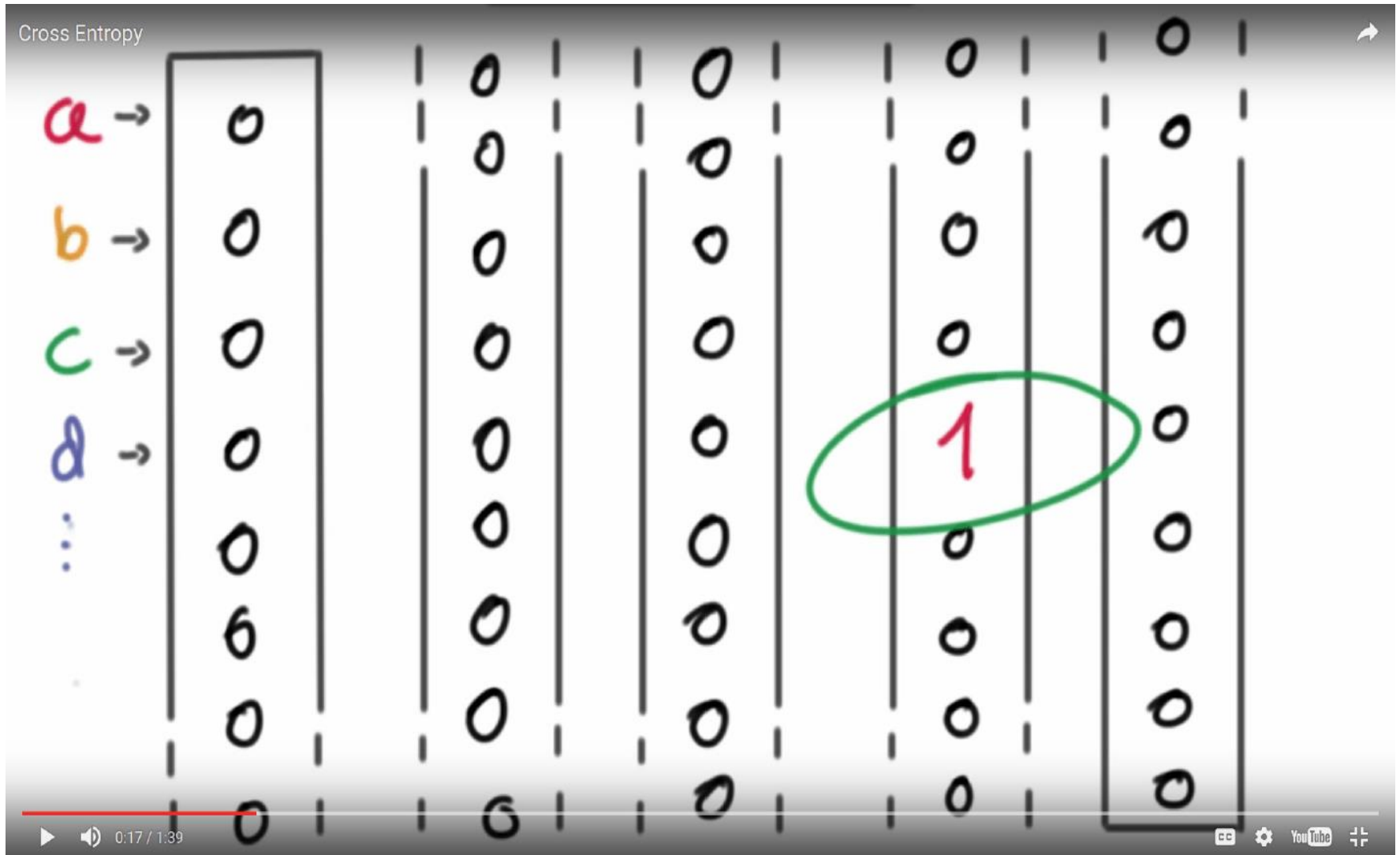
Optimisation: Cross-Entropy

$$D(S, L) = -\sum_i L_i \log(S_i)$$
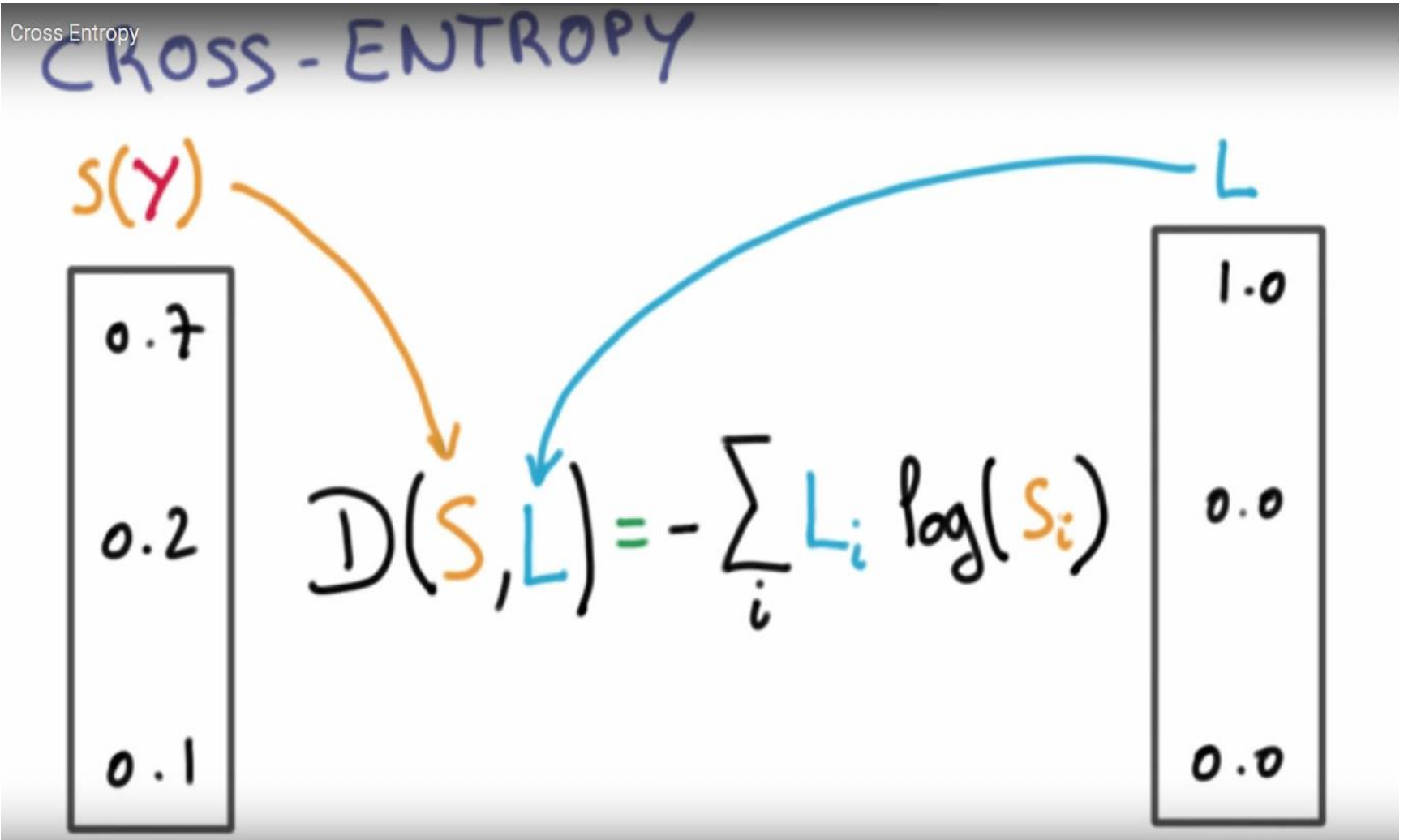
Multinomial logistic classification
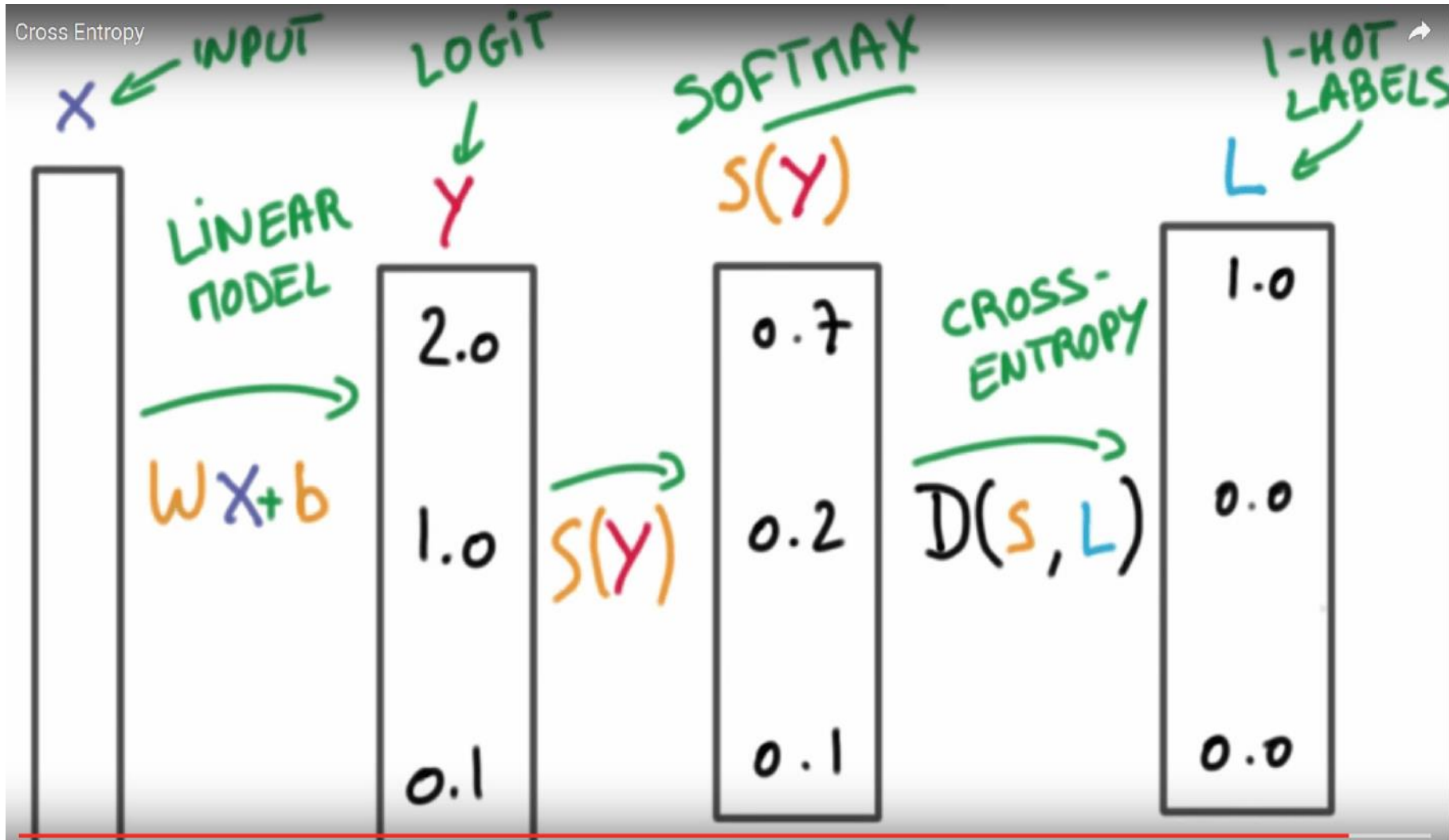
Optimisation of average loss

LOSS = AVERAGE CROSS-ENTROPY

BIG MATRIX!!

$$\mathcal{L} = \frac{1}{N} \sum_i D(S(Wx_i + b), L_i)$$

BIG SUM!!

Gradient decent
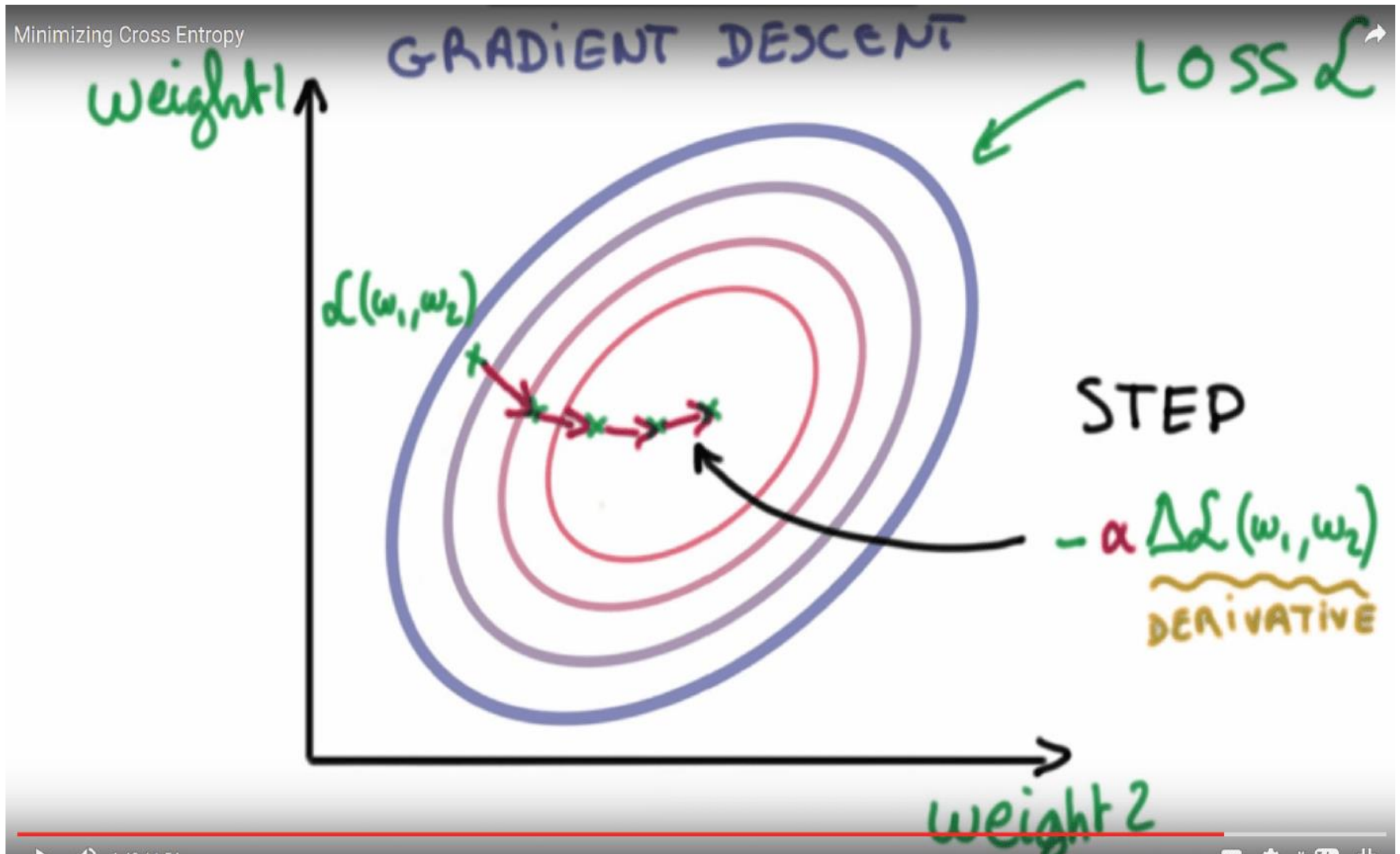
Normalised input and output
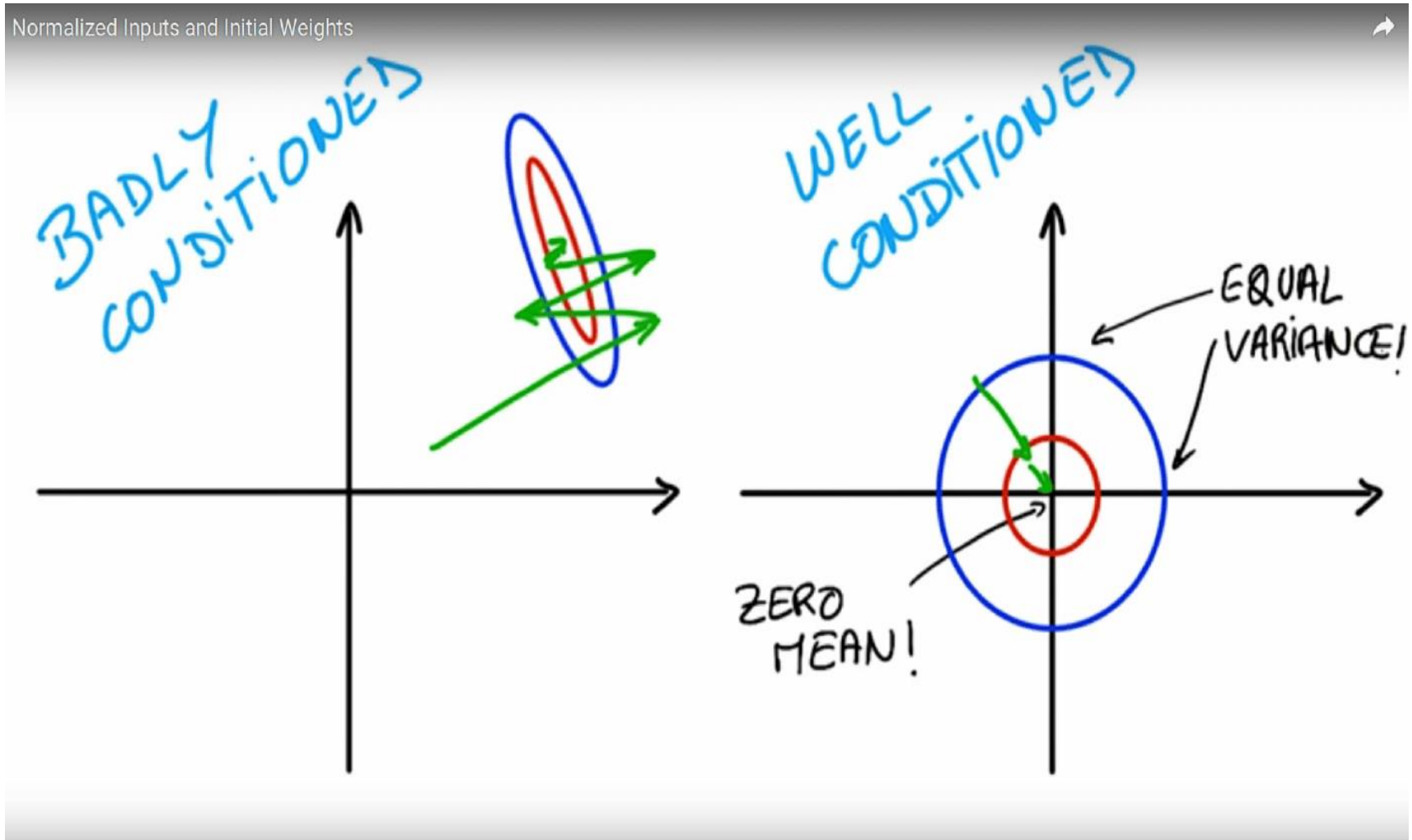
Normalised input and output

IMAGES

$$\frac{R-128}{128} \qquad \frac{G-128}{128} \qquad \frac{B-128}{128}$$

$\rightarrow [0 \dots 255]$
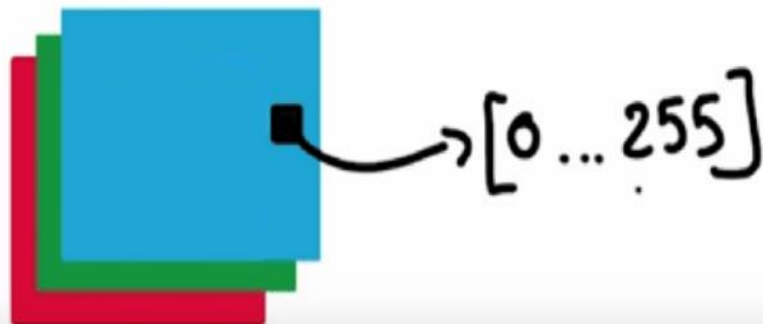
Initialisation
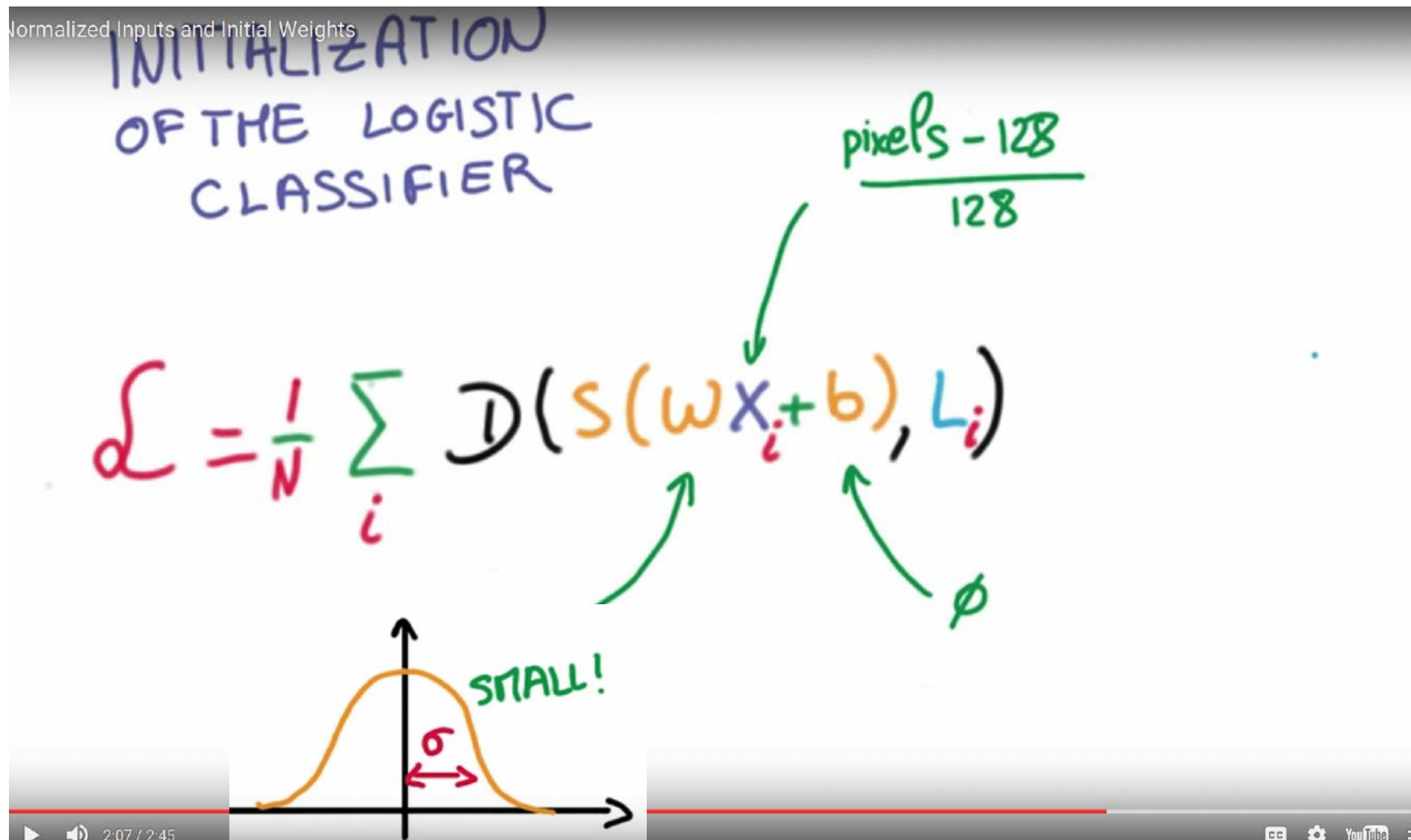
Training, validation, testing
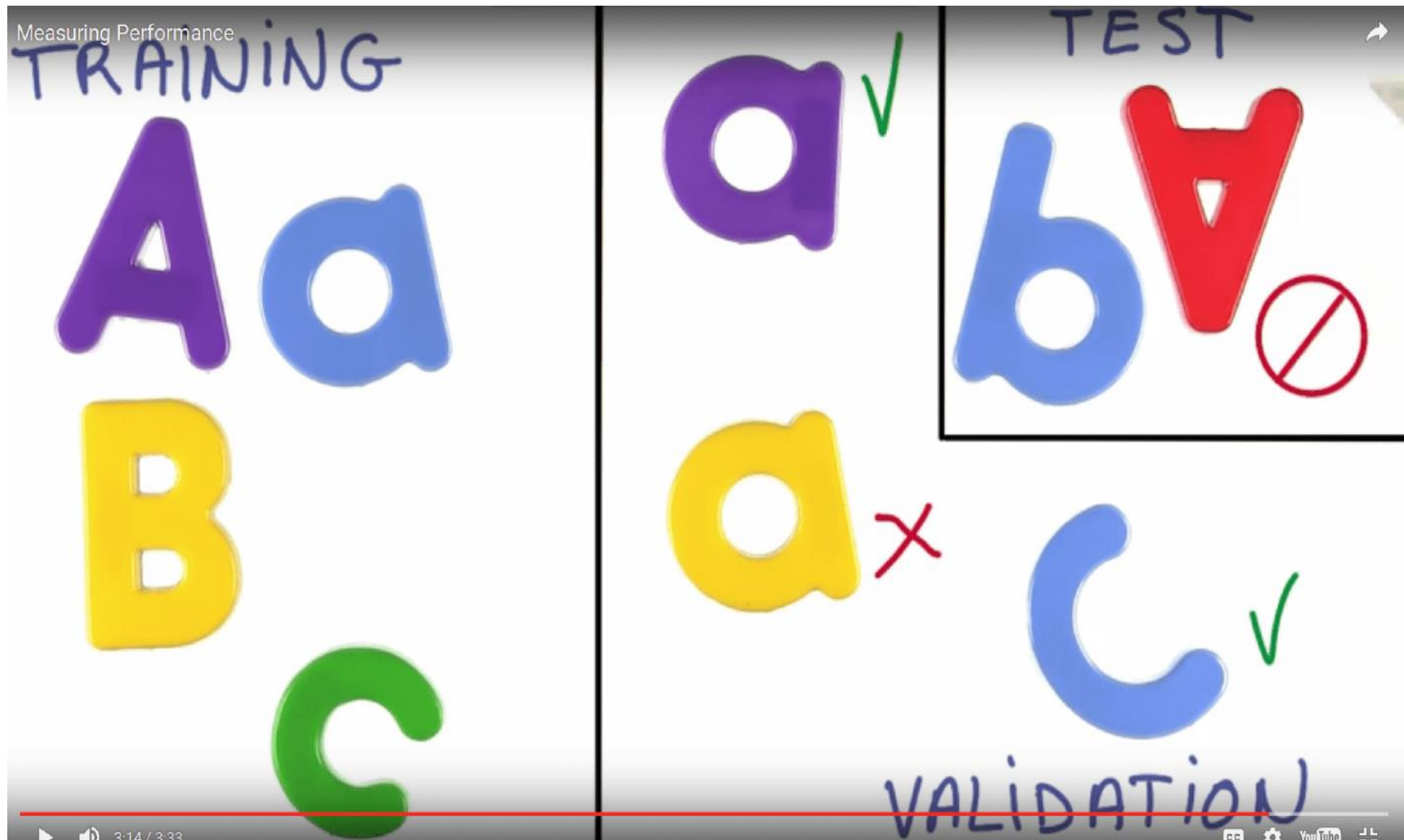
Gradient Descent
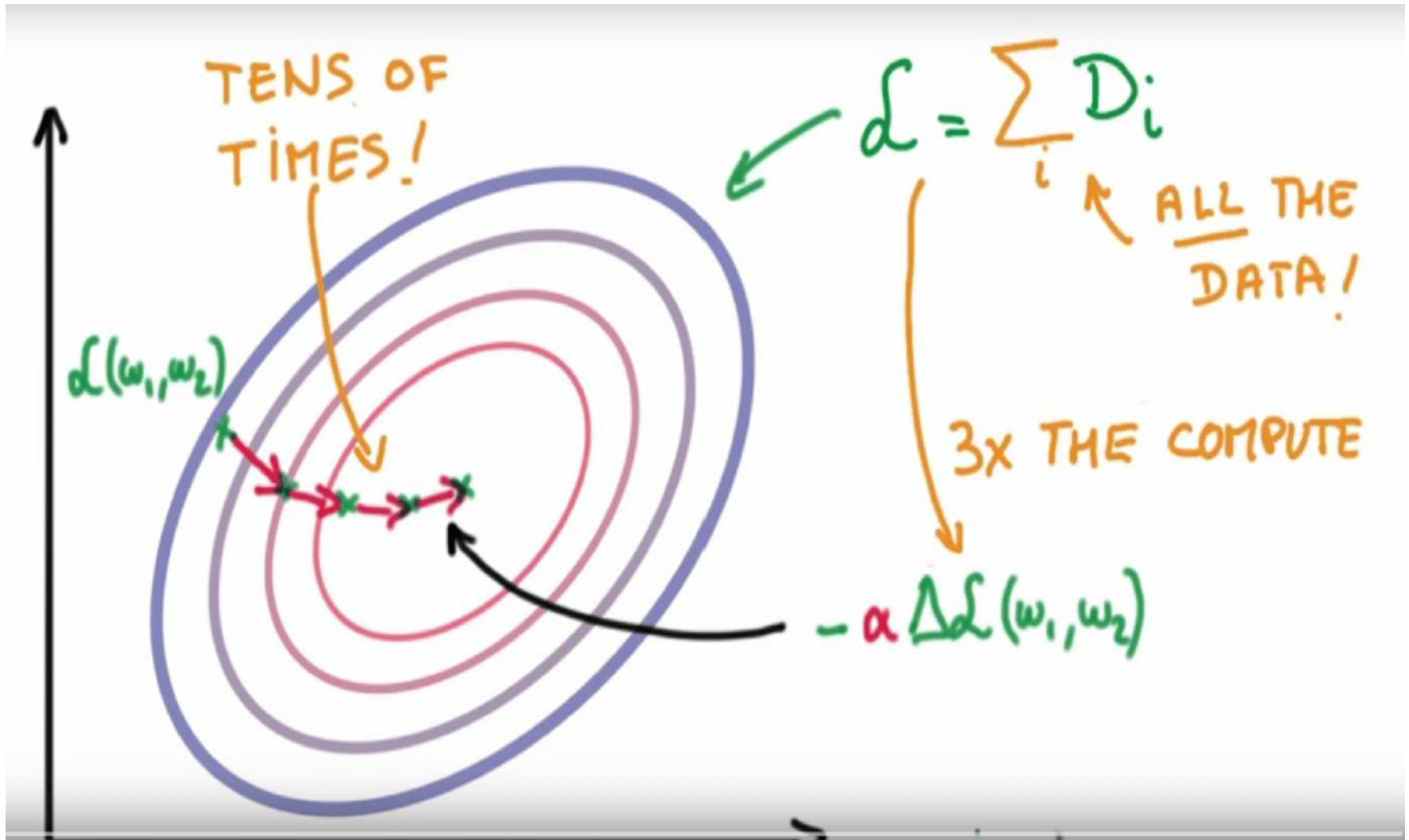
Stochastic Gradient Descent
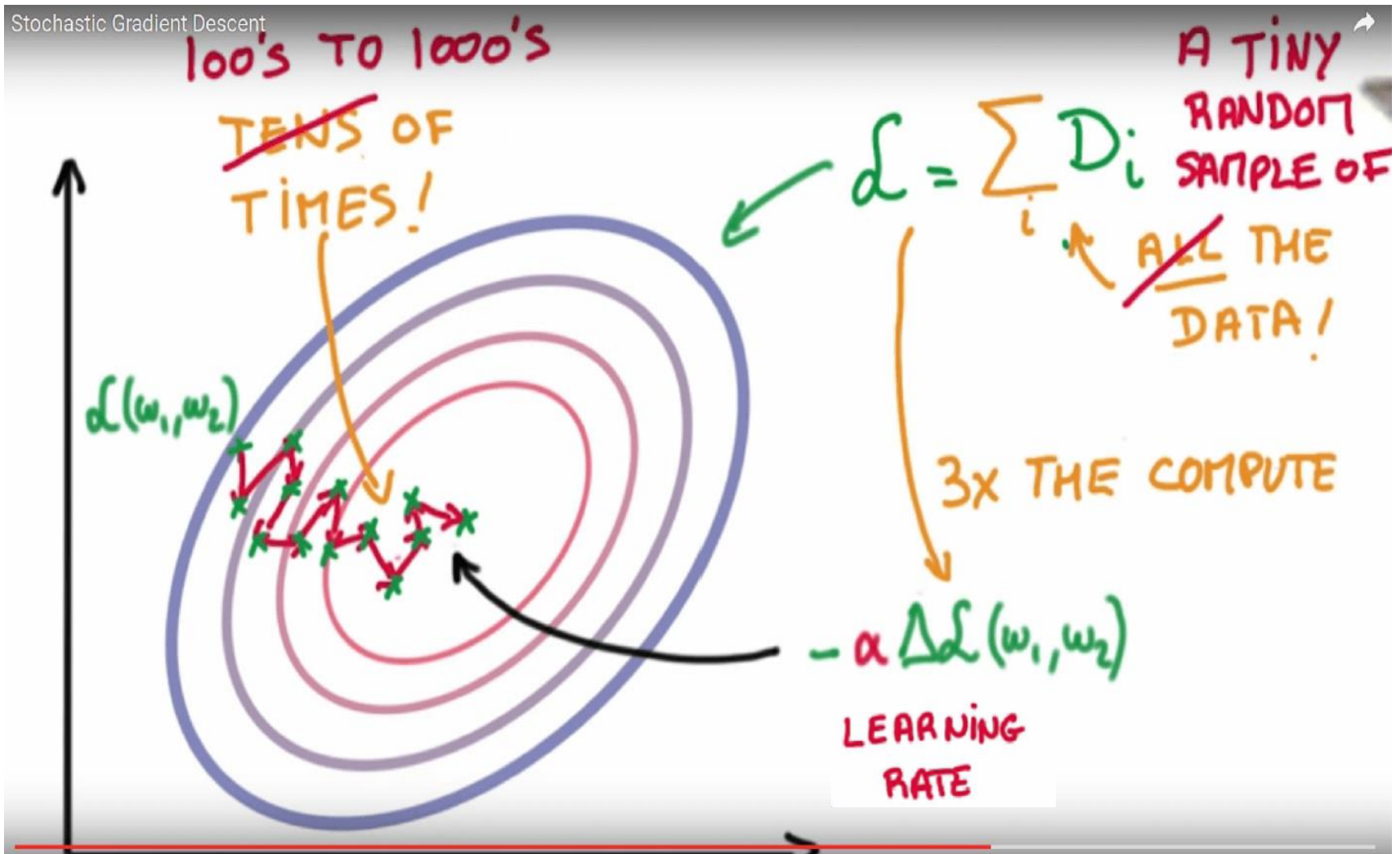
SDG: optimising with momentum
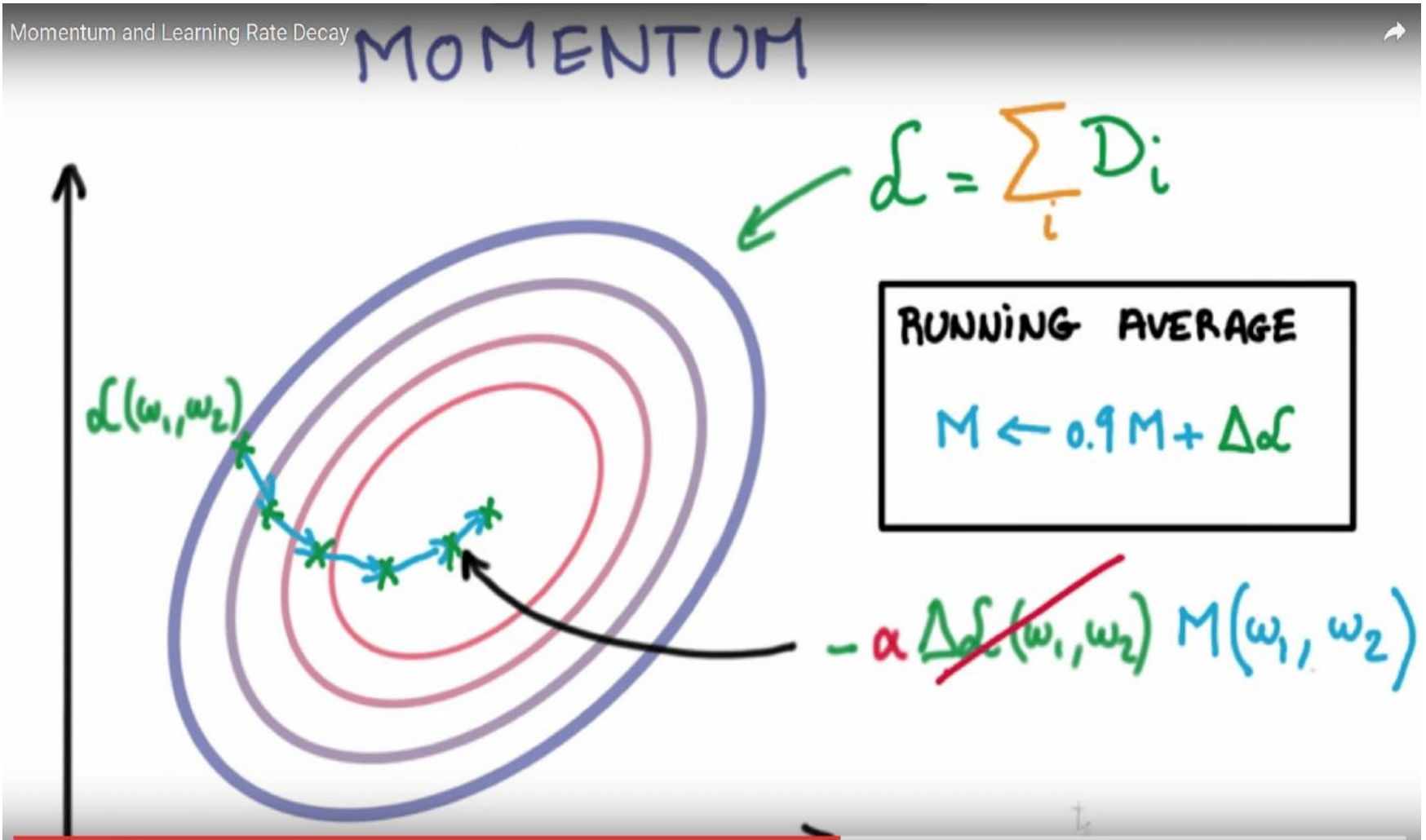
SDG: learning rate

SDG: „black magic"

MANY HYPER-PARAMETERS
- INITIAL LEARNING RATE
- LEARNING RATE DECAY
- MOMENTUM
- BATCH SIZE
- WEIGHT INITIALIZATION
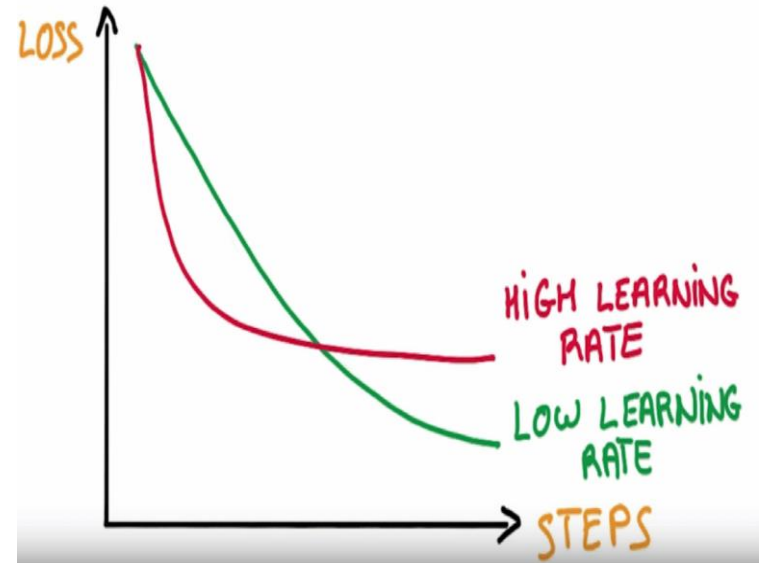
Input – linear - output
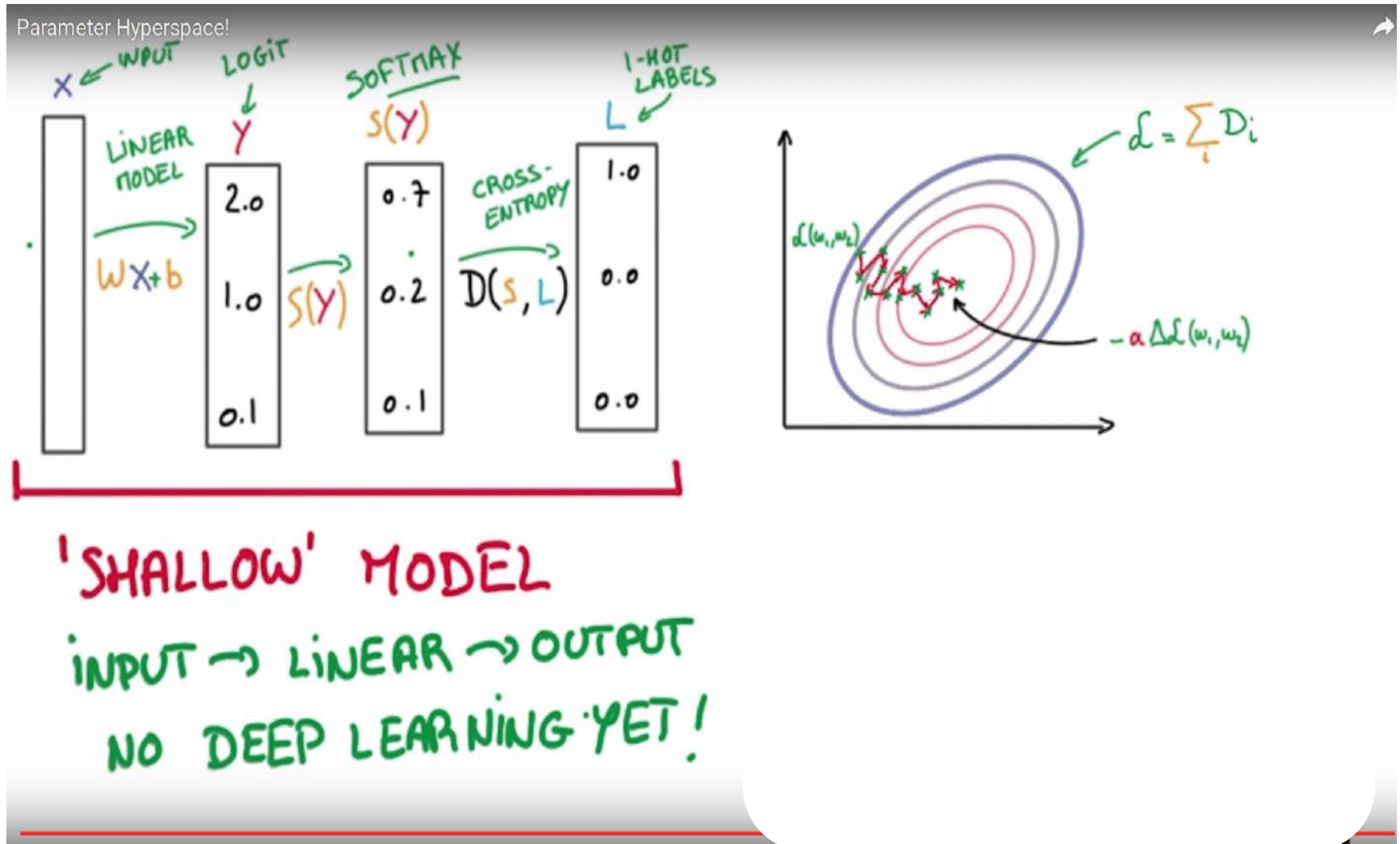
Linear models are linear
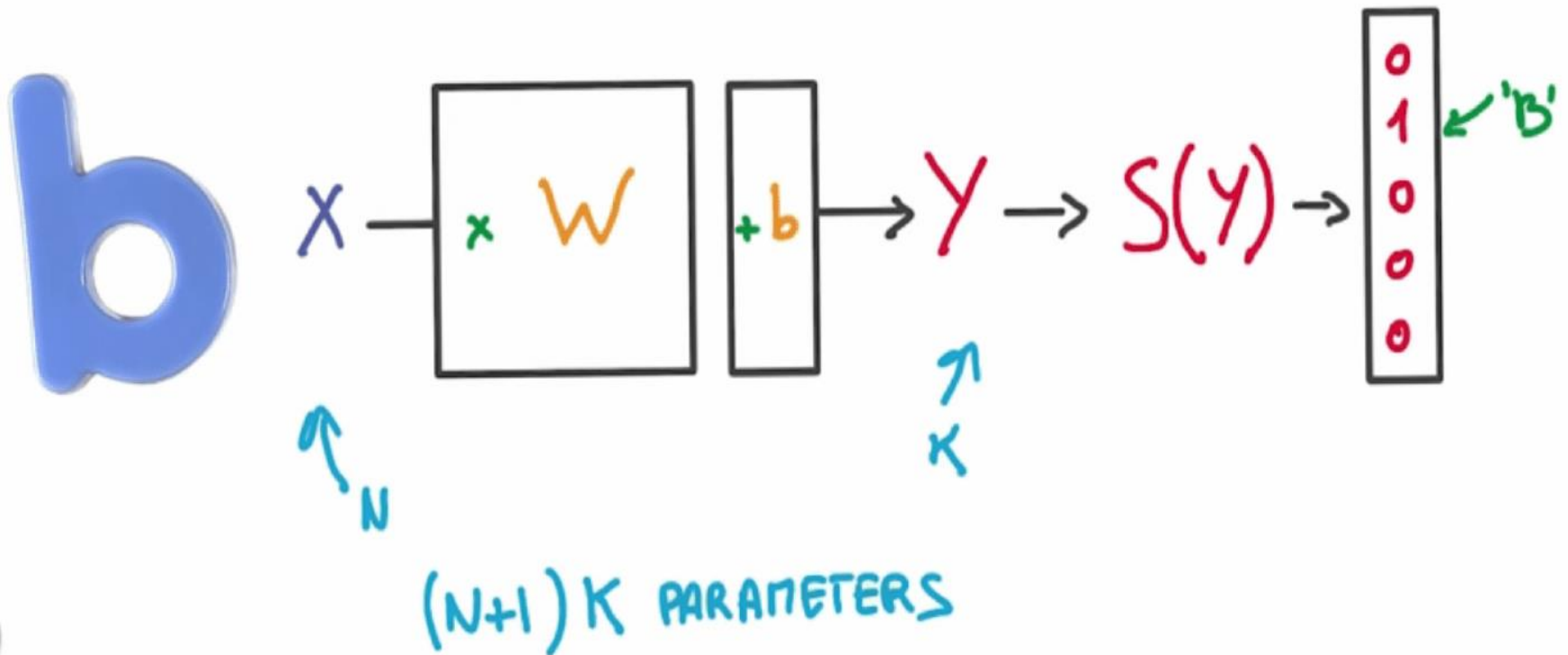
Linear models are stable

$$Y = WX \longrightarrow \Delta Y \sim |W| \Delta X$$

SMALL   BOUNDED   SMALL

$$Y = WX \longrightarrow$$

$$\frac{\Delta Y}{\Delta X} = W^T$$

$$\frac{\Delta Y}{\Delta W} = X^T$$

CONSTANTS

Linear models are here to stay

- ## This is still linear

$$Y = W_1 W_2 W_3 X = W X$$

- ## Lets introduce non-linearity

$$Y = W_1 W_2 W_3 X = W_2 X$$

NON - LINEARITIES

RELU: Rectified Linear Unit
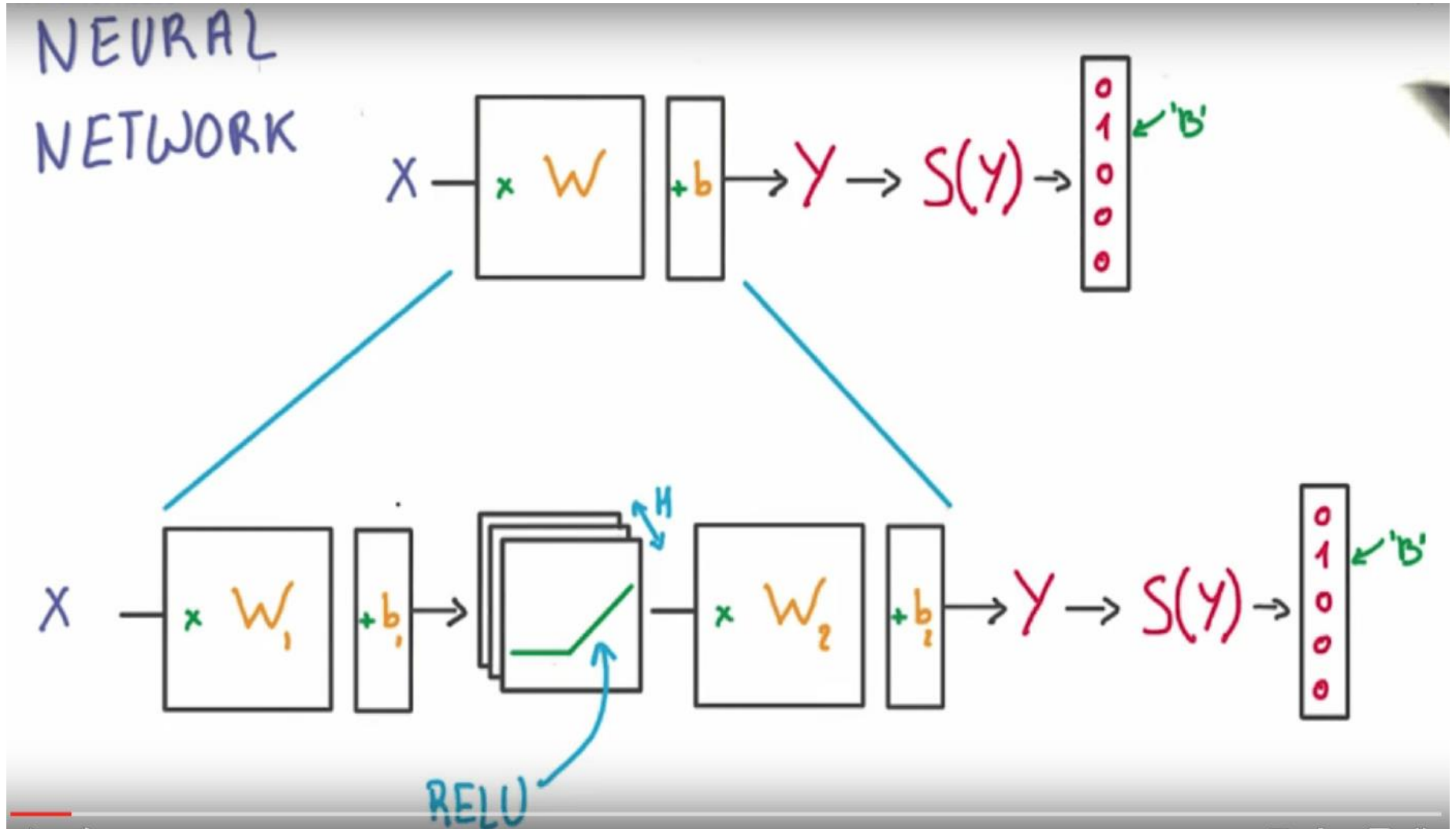
Networks of RELU
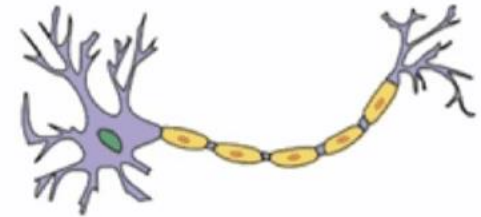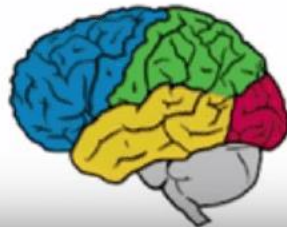
The Chain Rule

$$[g(f(x))]' = g'(f(x)) \times f'(x)$$

DERIVATIVE

PRODUCT

STACKING UP SIMPLE OPERATIONS
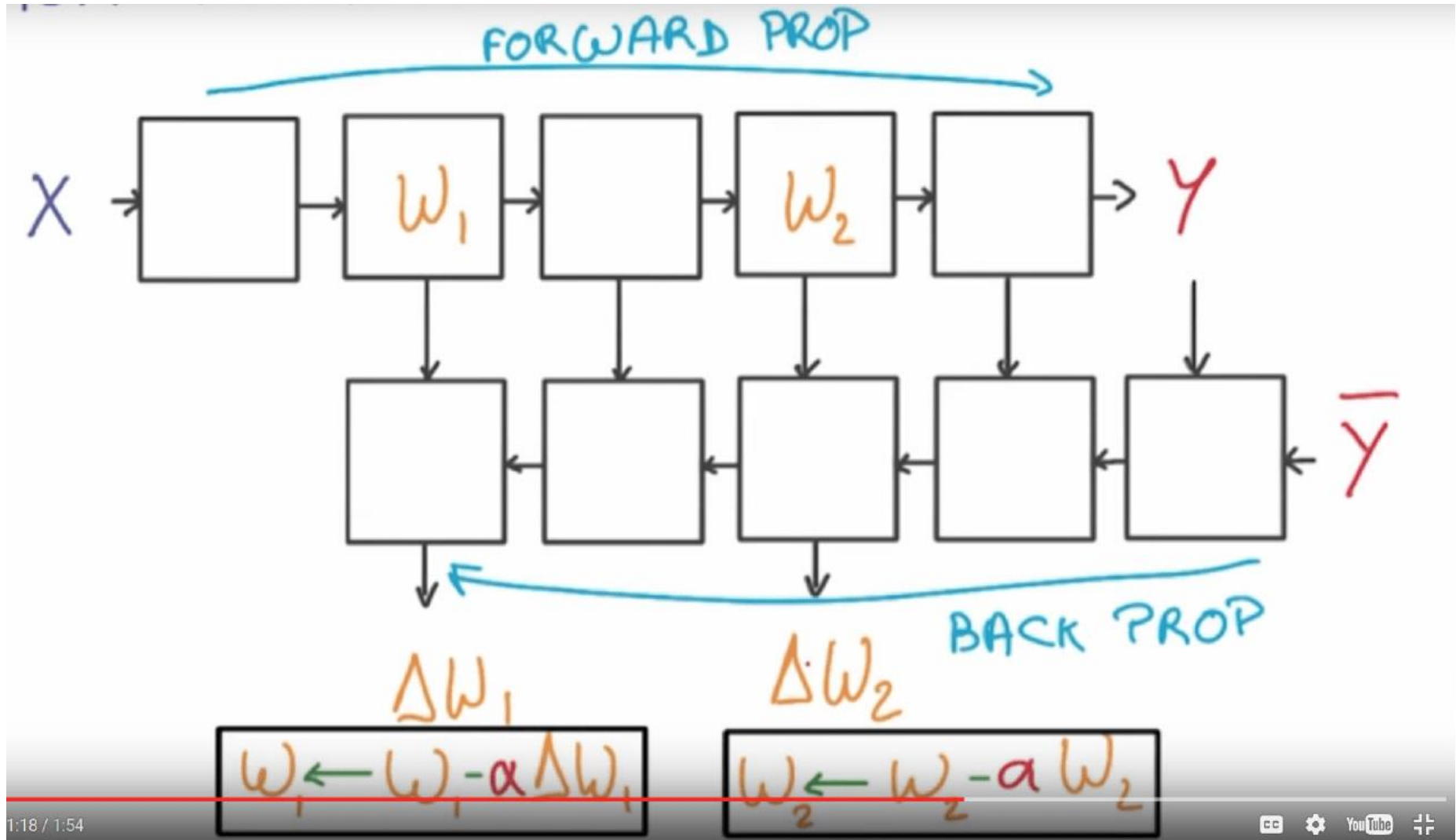
$X \rightarrow$ □ □ $\rightarrow$ [RELU] □ □ $\rightarrow Y \rightarrow S(Y) \rightarrow$ □

MULTIPLY ADD RELU MULTIPLY ADD SOFTMAX

Back - propagation
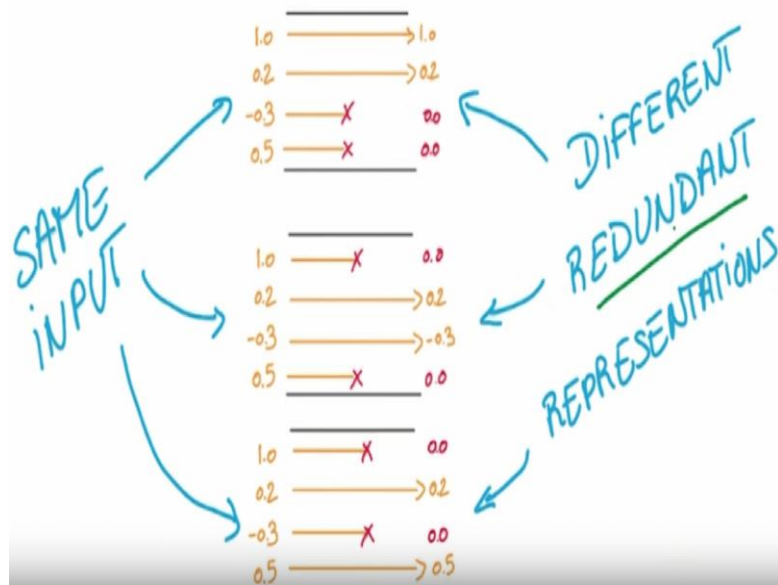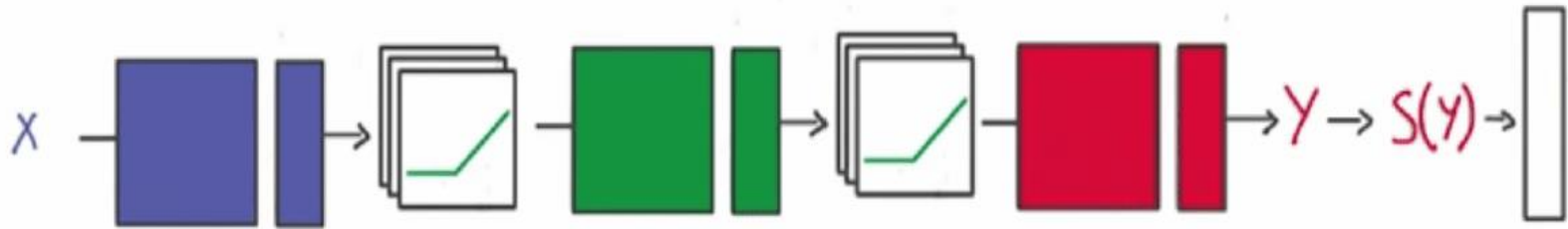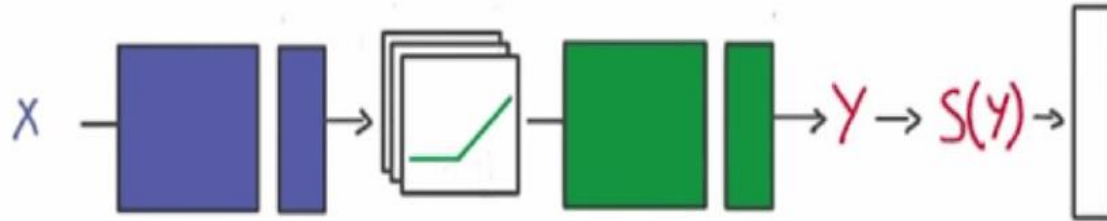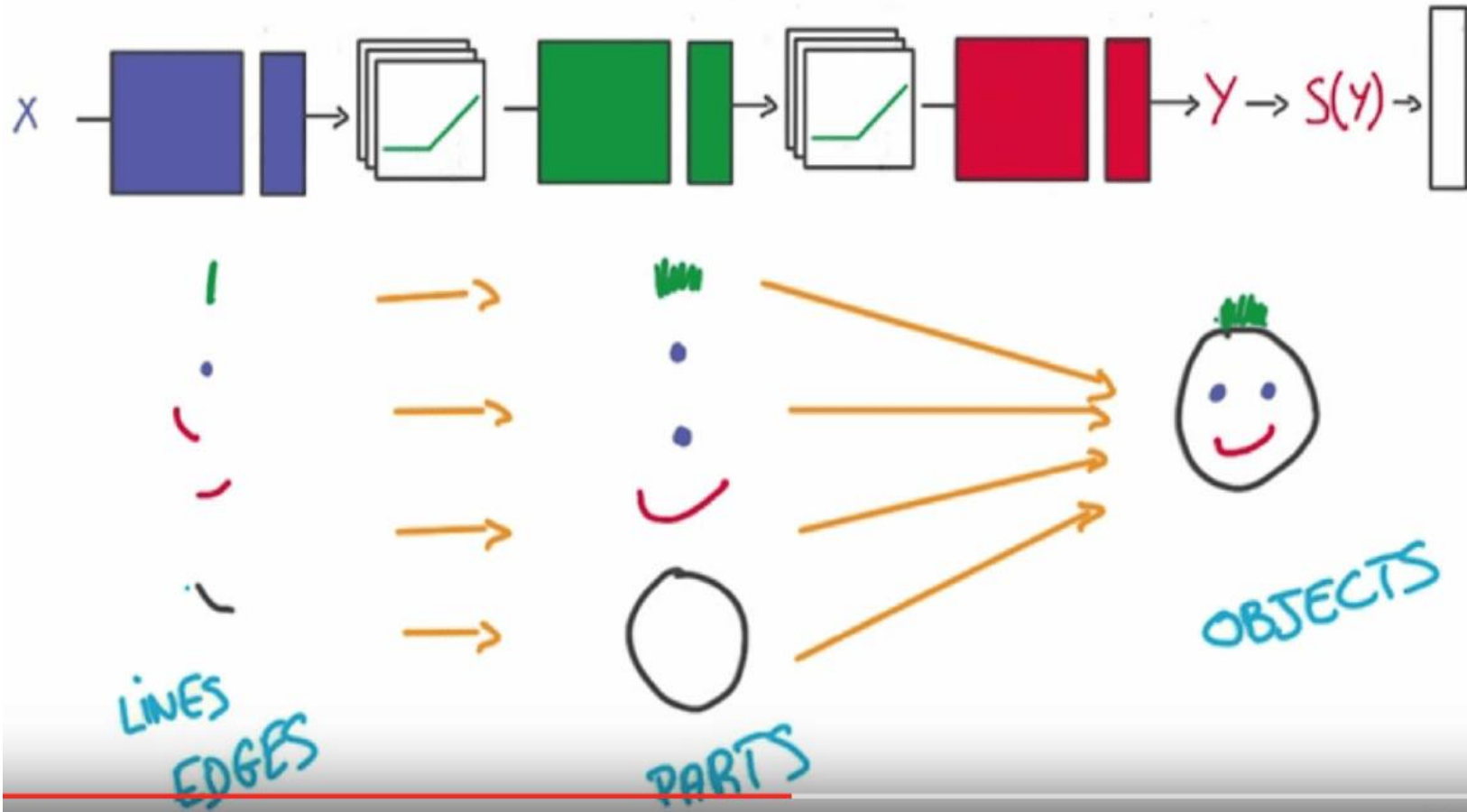
Optimisation tricks
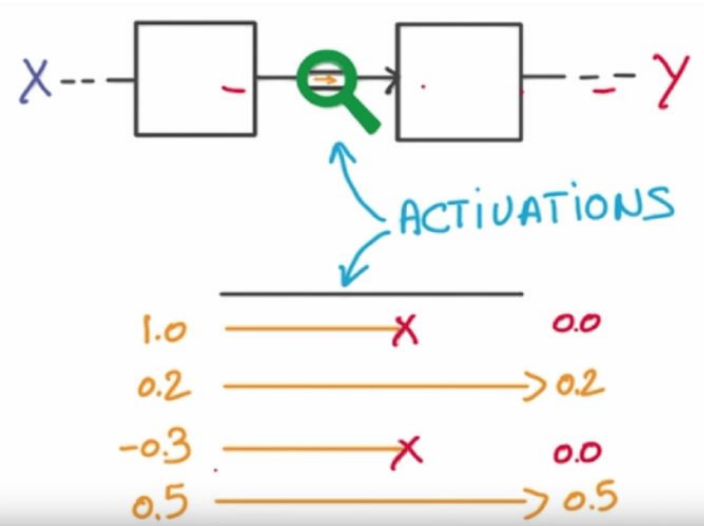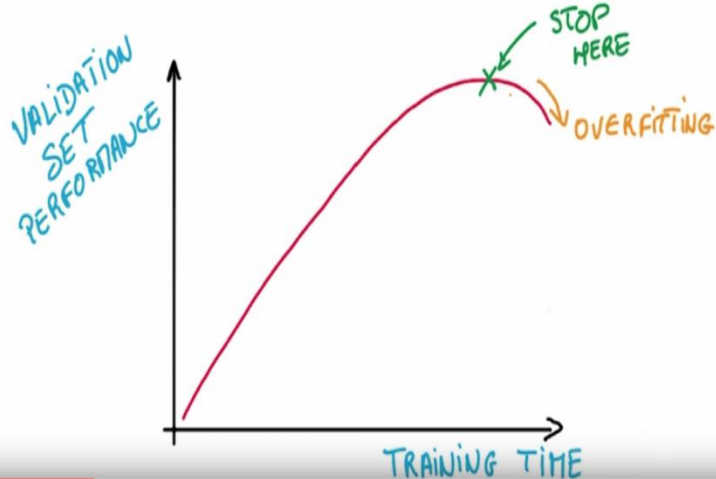
Optimisation trick: dropout
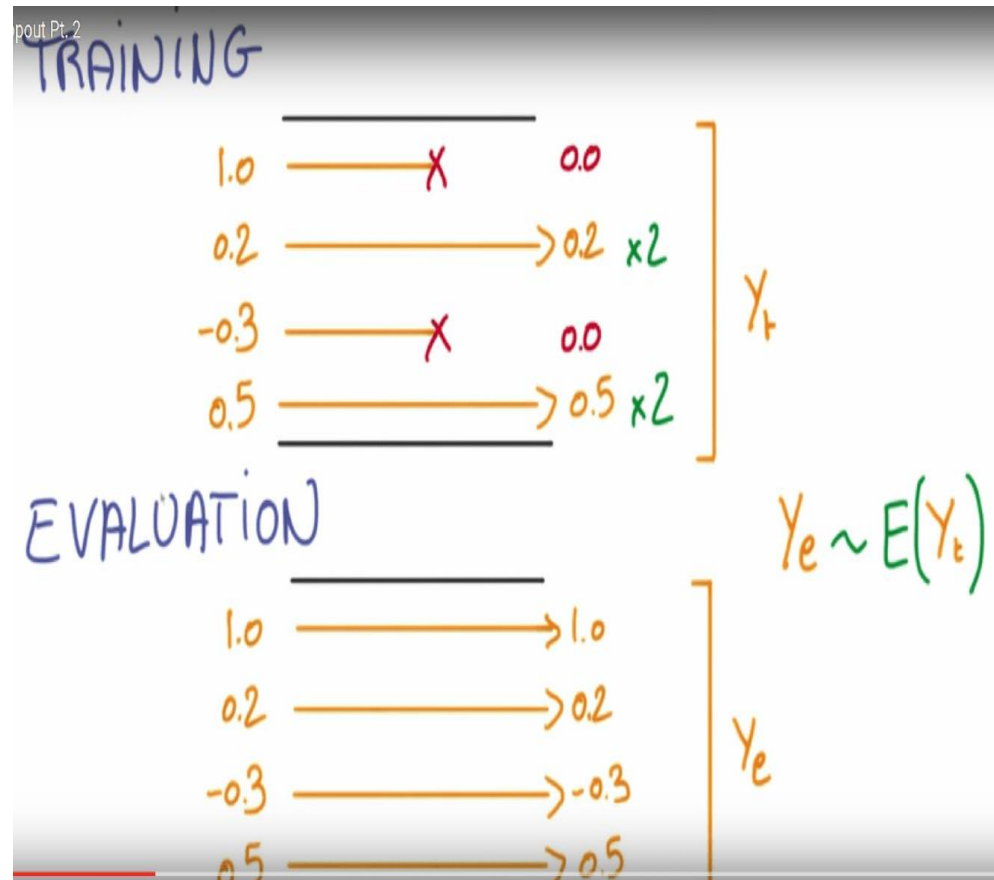
Deep networks

Deep networks

tensorflow.org/paper/whitepaper2015.pdf

TensorFlow:
Large-Scale Machine Learning on Heterogeneous Distributed Systems
(Preliminary White Paper, November 9, 2015)

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro,
Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,
Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser,
Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray,
Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar,
Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals,
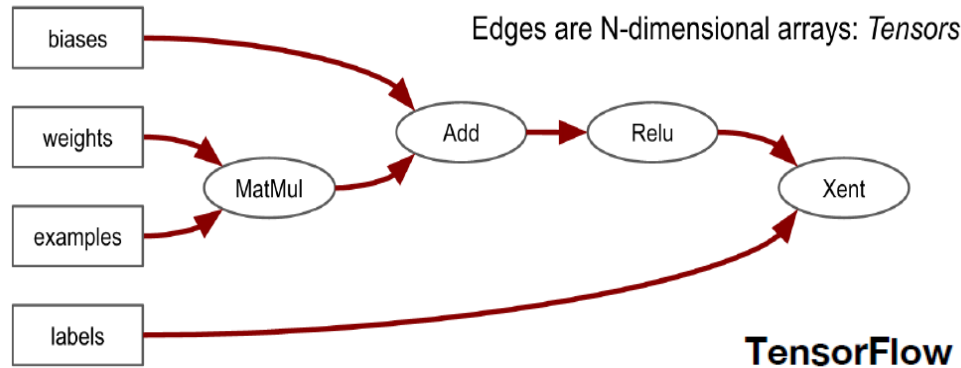Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng
Google Research*

Abstract

TensorFlow [1] is an interface for expressing machine learn-
ing algorithms, and an implementation for executing such al-
gorithms. A computation expressed using TensorFlow can be
executed with little or no change on a wide variety of hetero-
geneous systems, ranging from mobile devices such as phones
and tablets up to large-scale distributed systems of hundreds
of machines and thousands of computational devices such as
GPU cards. The system is flexible and can be used to express
a wide variety of algorithms, including training and inference
algorithms for deep neural network models, and it has been
used for conducting research and for deploying machine learn-
ing systems into production across more than a dozen areas of
computer science and other fields, including speech recogni-
tion, computer vision, robotics, information retrieval, natural
language processing, geographic information extraction, and
computational drug discovery. This paper describes the Ten-
sorFlow interface and an implementation of that interface that
we have built at Google. The TensorFlow API and a reference
implementation were released as an open-source package under
the Apache 2.0 license in November, 2015 and are available at
www.tensorflow.org.

TensorFlow is an open source software library for numerical computation using data flow graphs.

Edges are N-dimensional arrays: *Tensors*

biases → Add
weights → MatMul
examples → MatMul
MatMul → Add
Add → Relu
Relu → Xent
labels → Xent

TensorFlow - lazy evaluation

What is a Data Flow Graph?

Data flow graphs describe mathematical computation with a directed graph of nodes & edges. Nodes typically implement mathematical operations, but can also represent endpoints to feed in data, push out results, or read/write persistent variables. Edges describe the input/output relationships between nodes. These data edges carry dynamically-sized multidimensional data arrays, or tensors. The flow of tensors through the graph is where TensorFlow gets its name. Nodes are assigned to computational devices and execute asynchronously and in parallel once all the tensors on their incoming edges becomes available.

Key Features

- Deep Flexibility
- True Portability
- Connect Research and Production
- Auto-Differentiation
- Language Options
- Maximize Performance

Parallel Execution

- Launch graph in a Session
- Request output of some Ops with Run API
- TensorFlow computes set of Ops that must run to compute the requested outputs
- Ops execute, in parallel, as soon as their inputs are available

| C++ front end | Python front end | ... |

| Core TensorFlow Execution System |

| CPU | GPU | Android | iOS | ... |

There are hundreds of predefined Ops (and easy to add more)

- Basics: constant, random, placeholder, cast, shape
- Variables: assign, assign_sub, assign_add
- Queues: enqueue, enqueue_batch, dequeue, blocking or not.
- Logical: equal, greater, less, where, min, max, argmin, argmax.
- Tensor computations: all math ops, matmul, determinant, inverse, cholesky.
- Images: encode, decode, crop, pad, resize, color spaces, random perturbations.
- Sparse tensors: represented as 3 tensors.
- ...
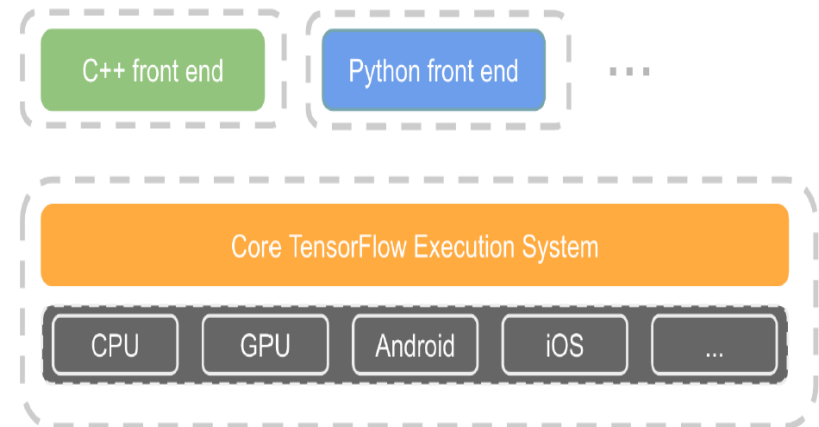
And many neural net specific Ops

- Activations: sigmoid, tanh, relu, dropout, ...
- Pooling: avg, max.
- Convolutions: with many options.
- Normalization: local, batch, moving averages.
- Classification: softmax, softmax loss, cross entropy loss, topk.
- Embeddings: lookups/gather, scatter/updates.
- Sampling: candidate sampler (various options), sampling softmax.
- Updates: "fused ops" to speed-up optimizer updates (Adagrad, Momentum.)
- Summaries: Capture information for visualization.

Hand-written diggits: MNIST
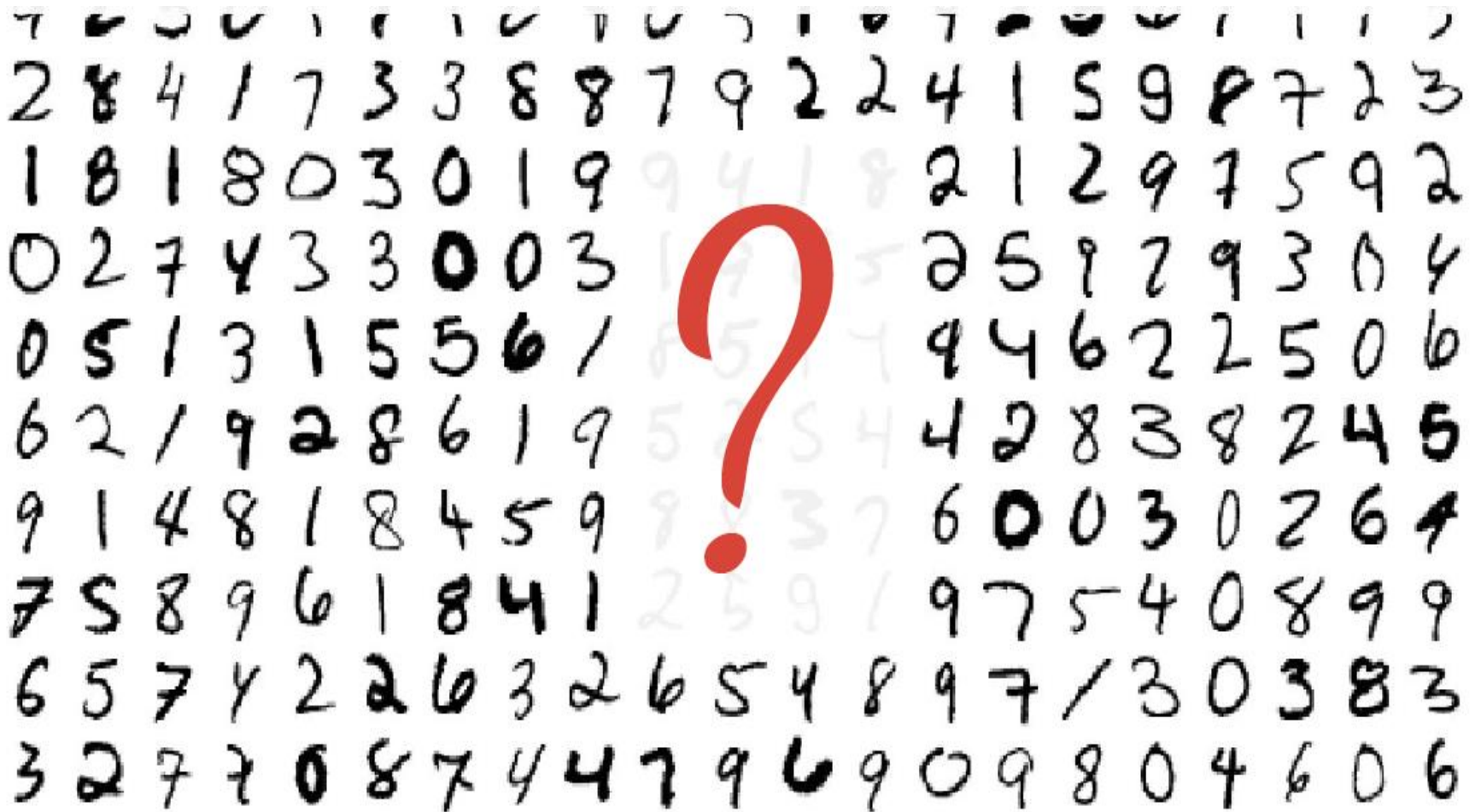
MNIST = Mixed National Institute of Standards and Technology - Download the dataset at http://yann.lecun.com/exdb/mnist/

Simple linear model

weighted sum of all pixels + bias

$$softmax(L_n) = \frac{e^{L_n}}{\|e^L\|}$$

neuron outputs
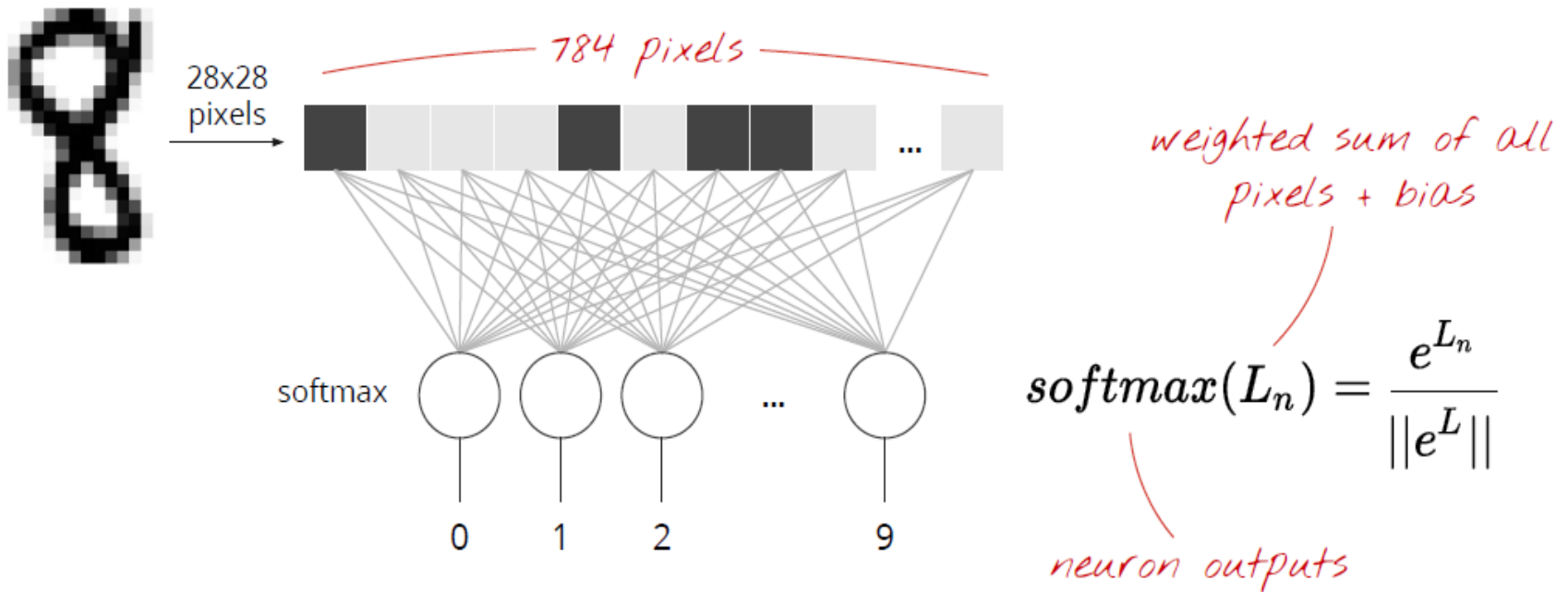
TensorFlow full python code

```python
import tensorflow as tf
```

initialisation

```python
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
init = tf.initialize_all_variables()
```

model

```python
# model
Y=tf.nn.softmax(tf.matmul(tf.reshape(X,[-1, 784]), W) + b)

# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])
```

success metrics

```python
# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct,tf.float32))
```

training step

```python
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)


sess = tf.Session()
sess.run(init)

for i in range(10000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)
```

Run

```python
    # success ? add code to print it
    a,c = sess.run([accuracy, cross_entropy], feed=train_data)

    # success on test data ?
    test_data={X:mnist.test.images, Y_:mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy], feed=test_data)
```

Slides from M. Gorner@youtube

Simple linear model

Multi-layer connected network
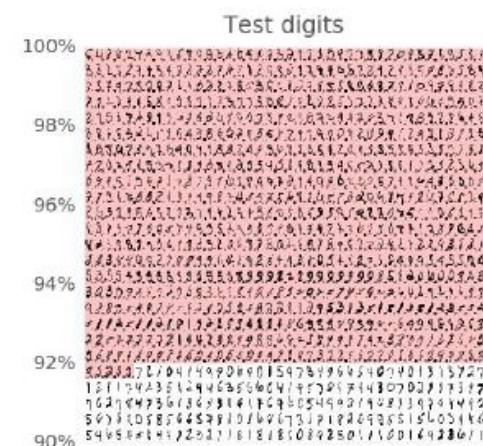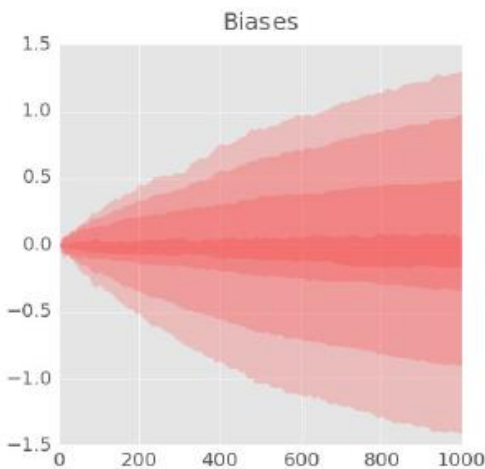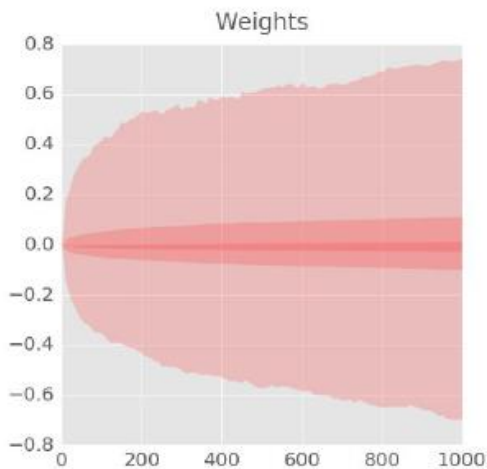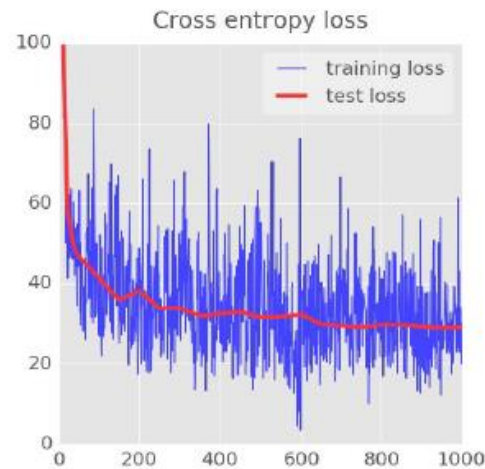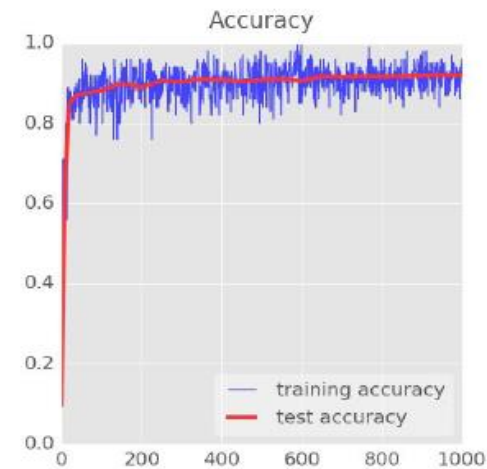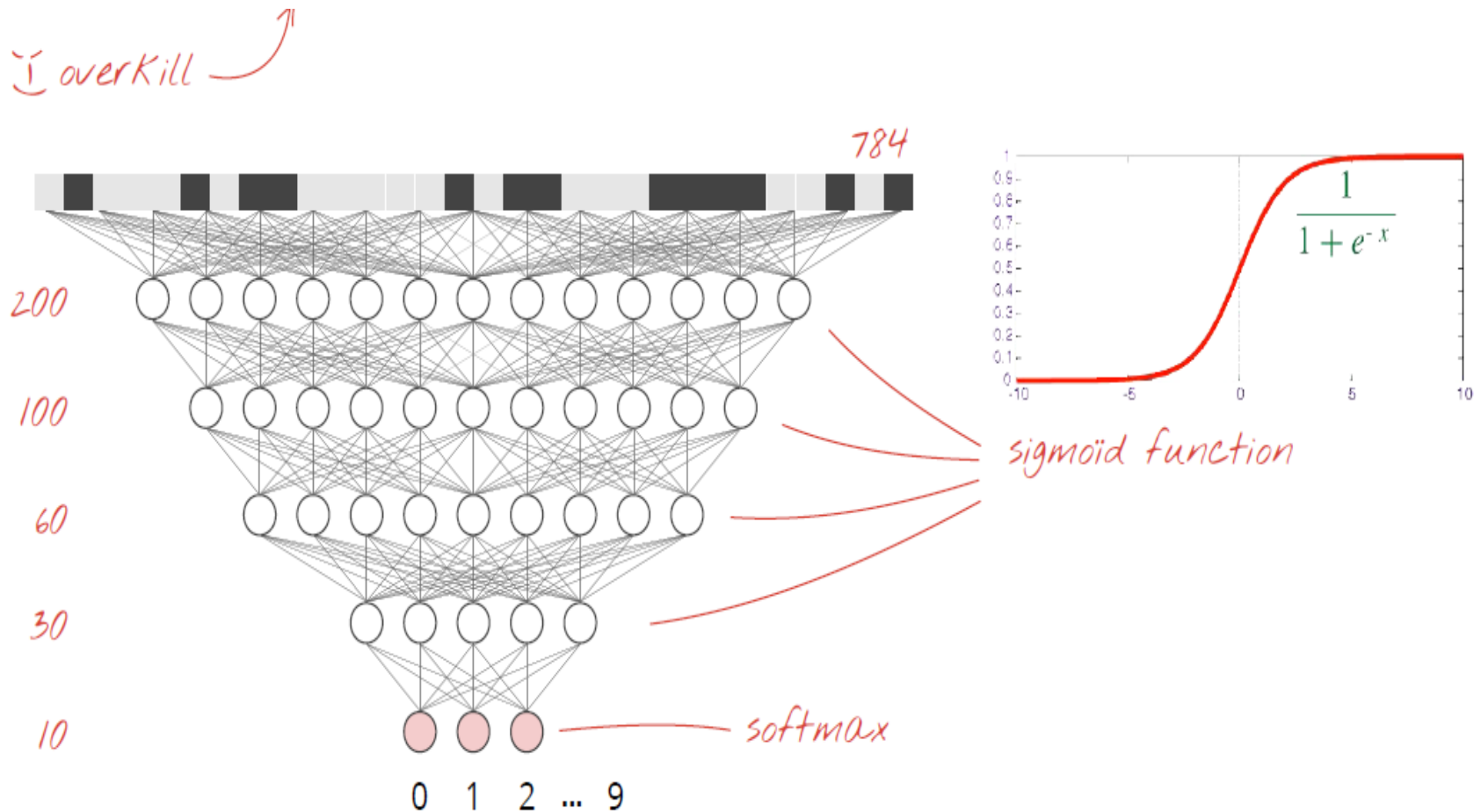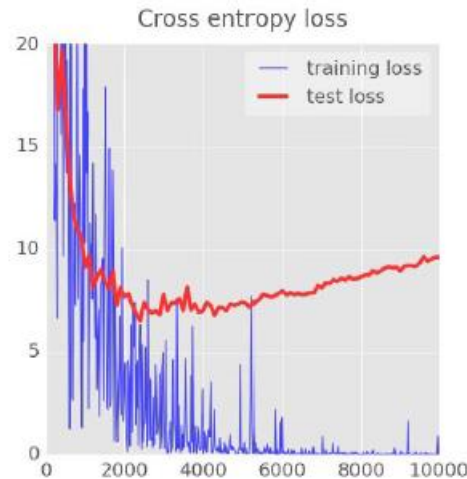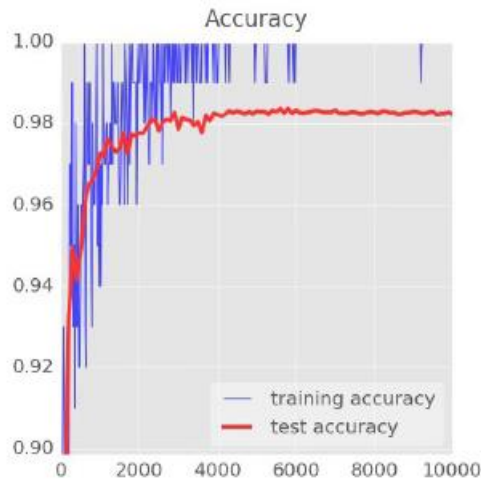

Slides from M. Gorner@youtube

Multi-layer connected network
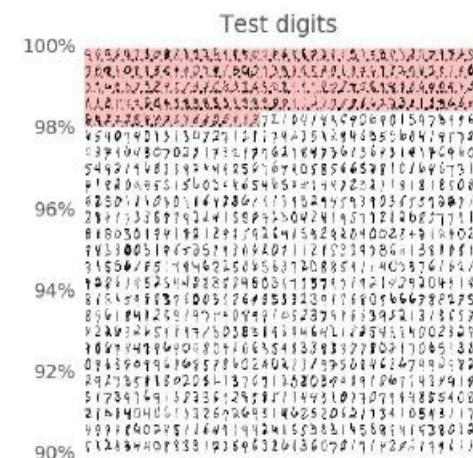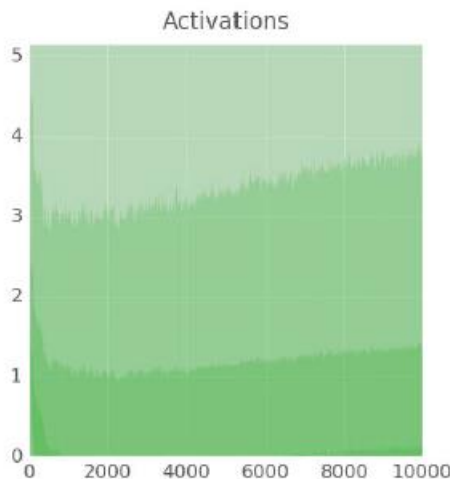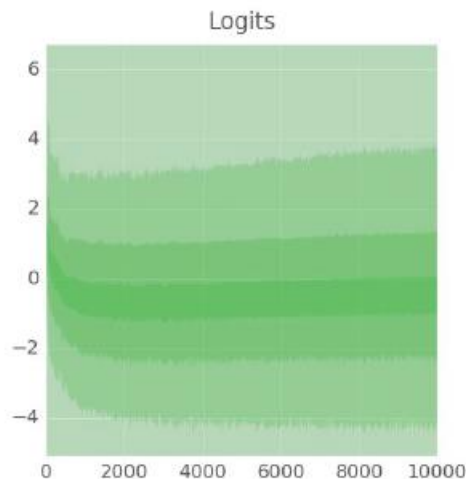
All tricks count
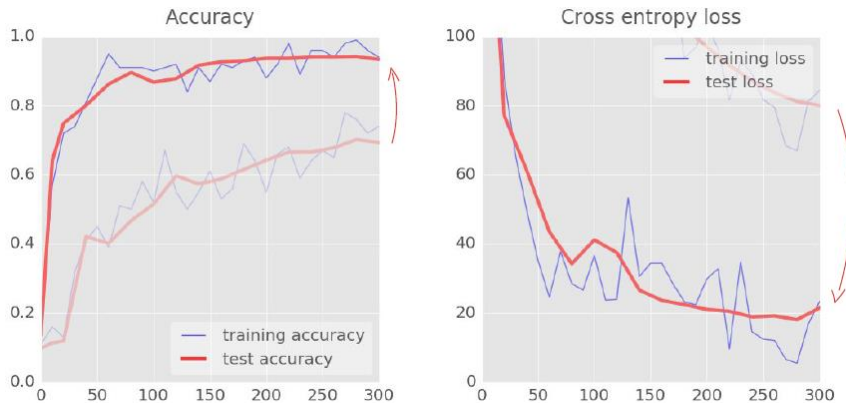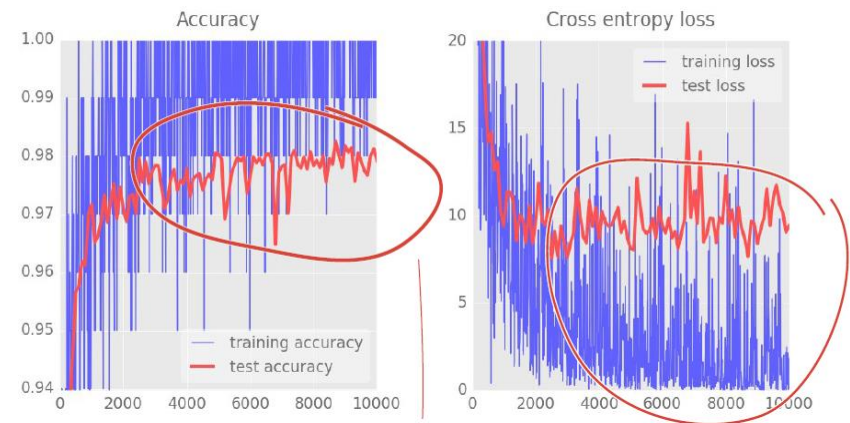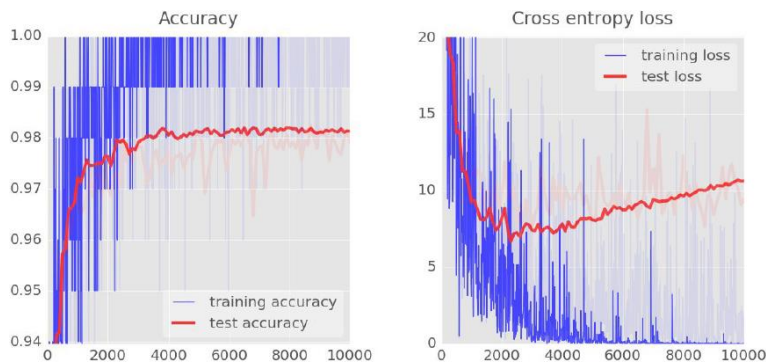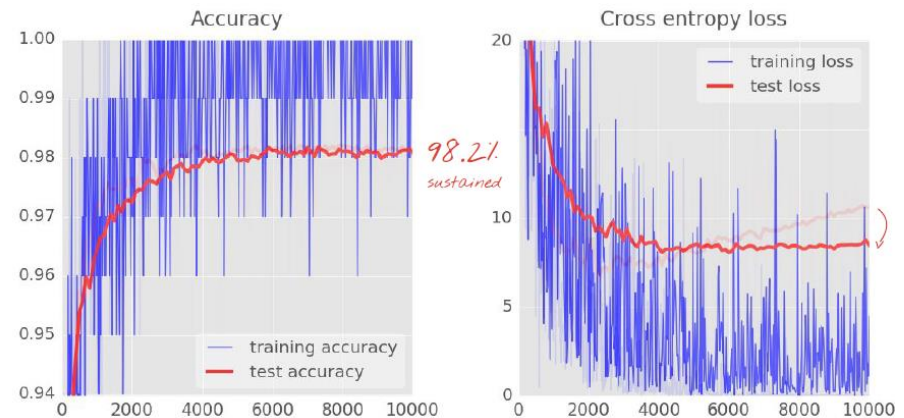
Exponentialy reduce learning rates

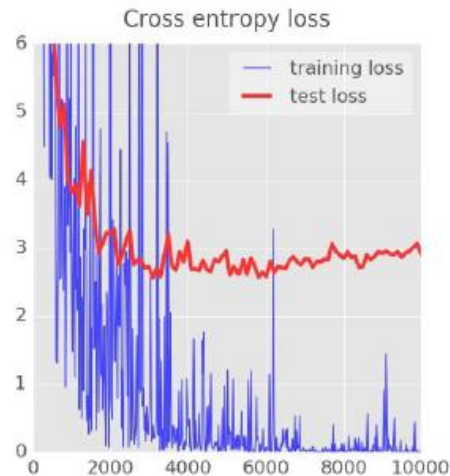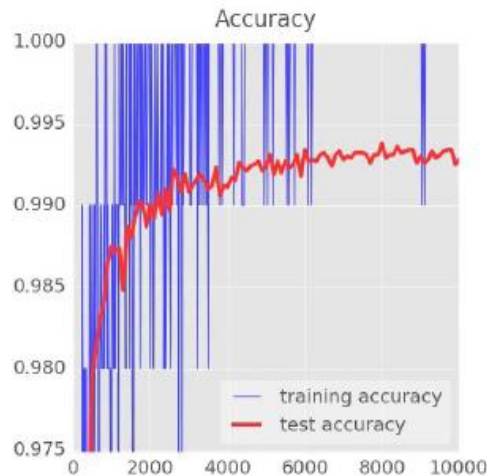Learning rate 0.003 at start then dropping exponentially to 0.0001

Add drop-out

RELU, decaying learning rate 0.003 -> 0.0001 and dropout 0.75
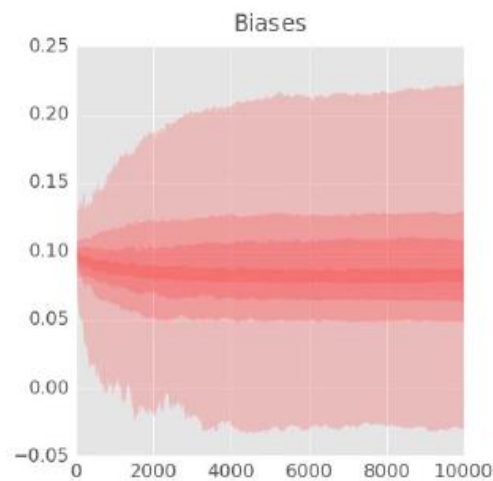
Slides from M. Gorner@youtube

Can do better with conv network

References

http://www.deeplearning.book.org

http://download.tensorflow.org/paper/whitepaper2015.pdf

https://www.tensorflow.org/

http://www.youtube.com/watch?v=vq2nnJ4g6NO

https://www.udacity.com/course/deep-learning--ud730