

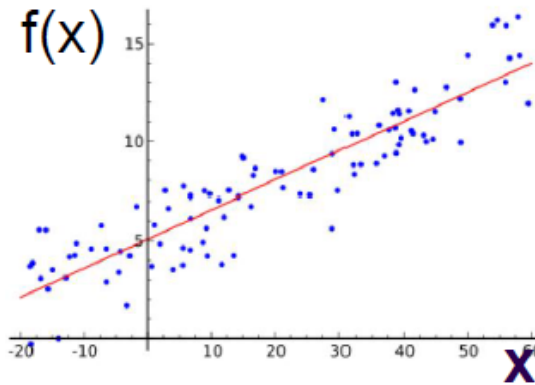
INTRODUCTION TO DATA SCIENCE

26/01/2021

WFAiS UJ, Informatyka Stosowana
I stopień studiów

What are Multivariate Techniques

→ Many things ... starting from “linear regression” ...

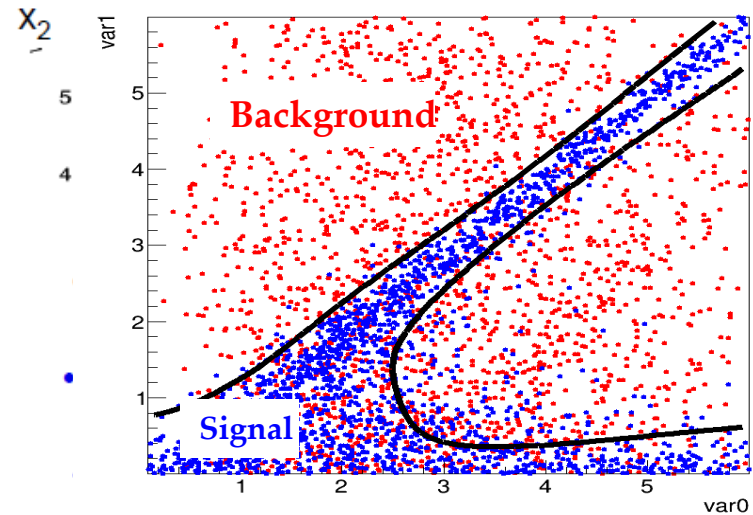


to multivariate event classification

→ or w/o prior ‘analytic’ model

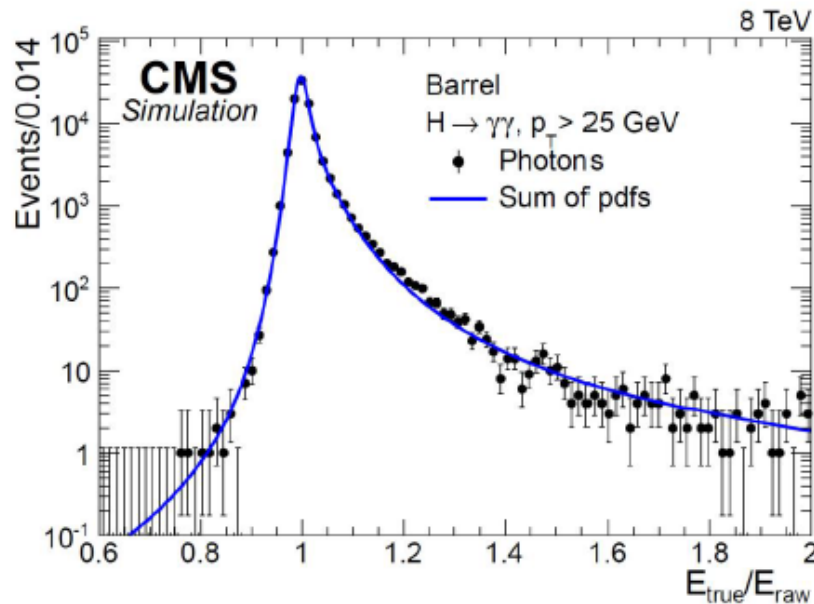
→ typically “multivariate”

- 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 \rightarrow 'regression'
 - (e.g. calorimeter calibration/correction function)

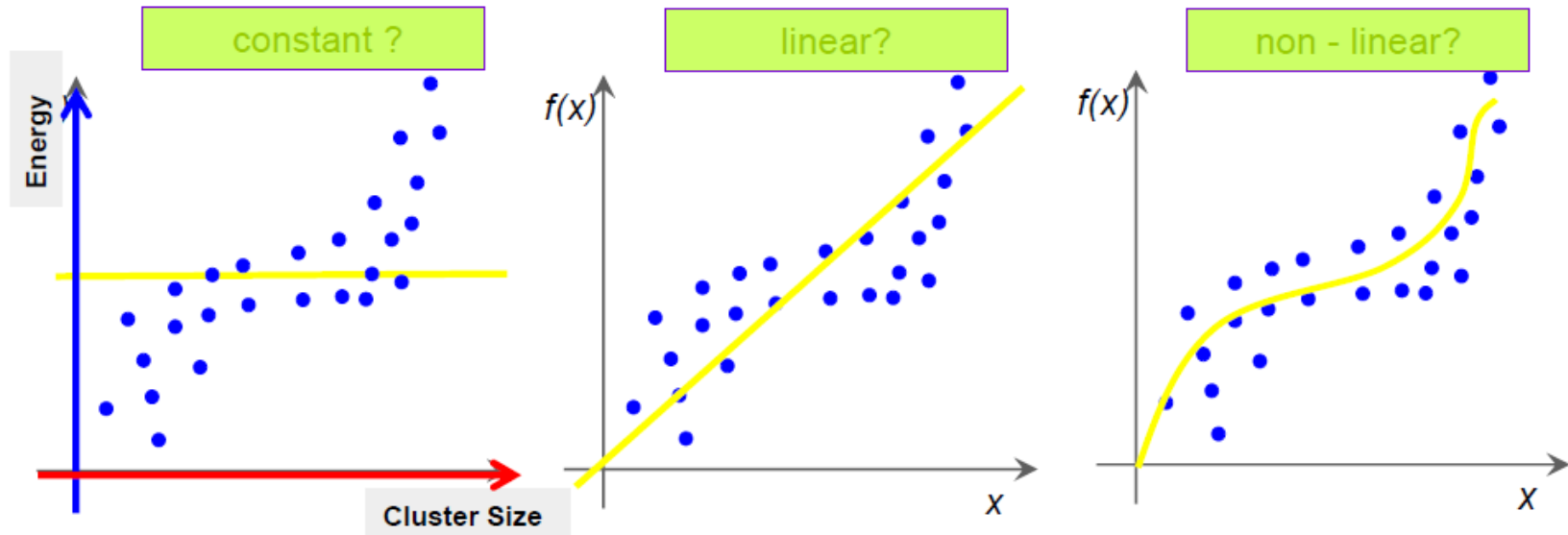


MC sample: γ +jets

- Raw energy in crystals, η , Φ
 - Cluster shape variables
 - Local cluster position variables
(energy leakage)
 - Pile-up estimators
- \rightarrow predict energy correction (i.e. parameters in crystal-ball: pdf for energy measurement)

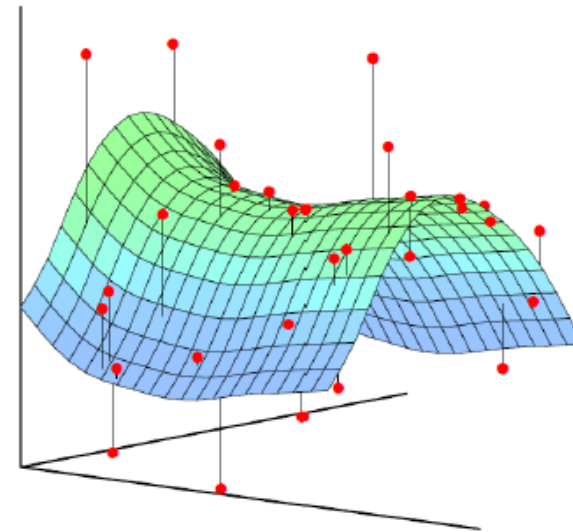
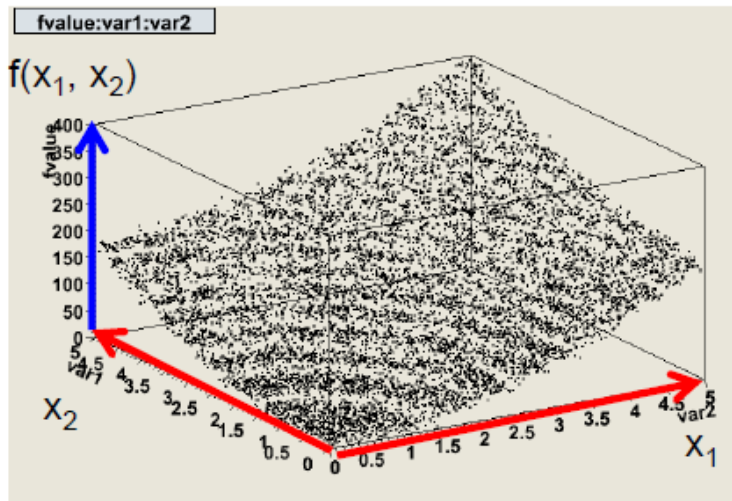
Regression

- “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. $f(x) = ax_1^2 + bx_2^2 + c$
- 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)’, it is not: “Newton deriving $F=m \cdot a$ ”
→ 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:

$$\vec{x} = (x_1, \dots, x_n) \quad \text{"feature vector"}$$

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

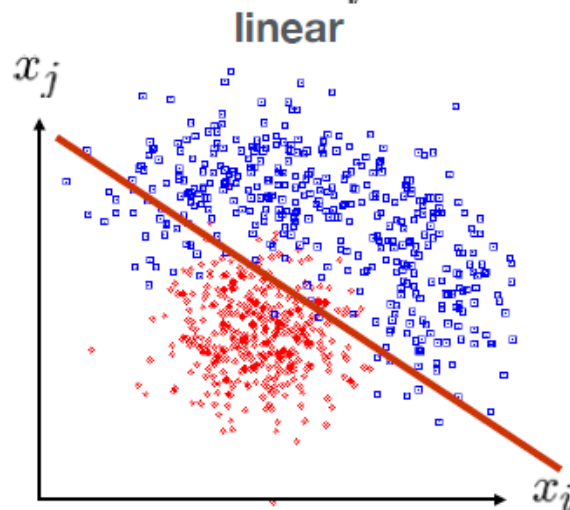
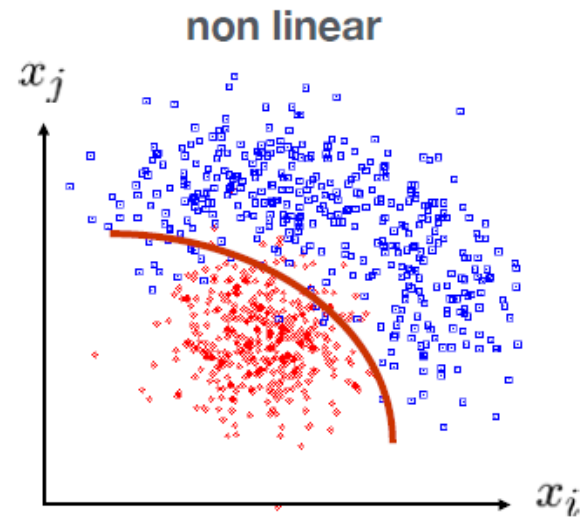
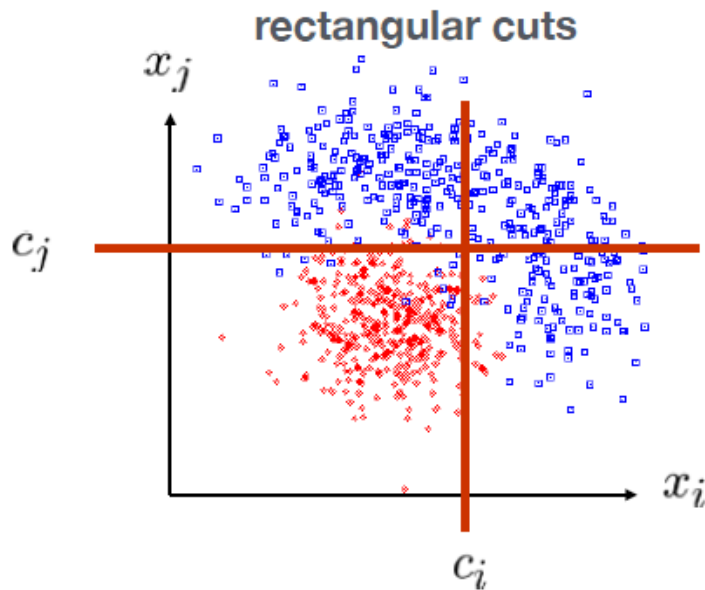
This is usually done by mapping the feature vector to a single variable, i.e., to scalar test statistic:

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

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

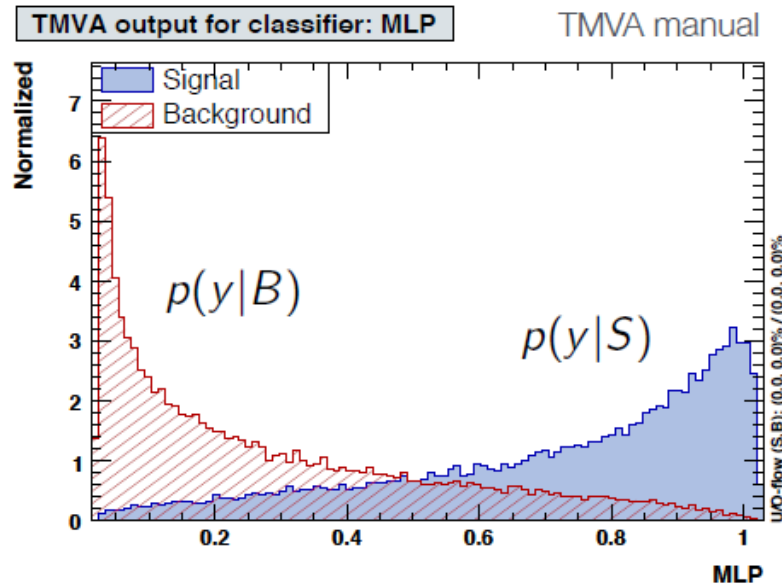


k-Nearest-Neighbor,
Boosted Decision Trees,
Multi-Layer Perceptrons,
Support Vector Machines

...

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:

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

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.: cartesian 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* ($x = \{\text{medium, small, large}\}$) or *categorical* ($x = \{\text{red, blue, green}\}$)

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/inappropriate (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

SUPERVISED

Predict a category

CLASSIFICATION

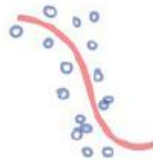
«Divide the socks by color»



Predict a number

REGRESSION

«Divide the ties by length»



OUR FOCUS

Data is not labeled in any way

UNSUPERVISED

Divide by similarity

CLUSTERING

«Split up similar clothing into stacks»



Identify sequences

Find hidden dependencies

ASSOCIATION

«Find what clothes I often wear together»



DIMENSION REDUCTION (generalization)

«Make the best outfits from the given clothes»



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

Where are the neural networks?

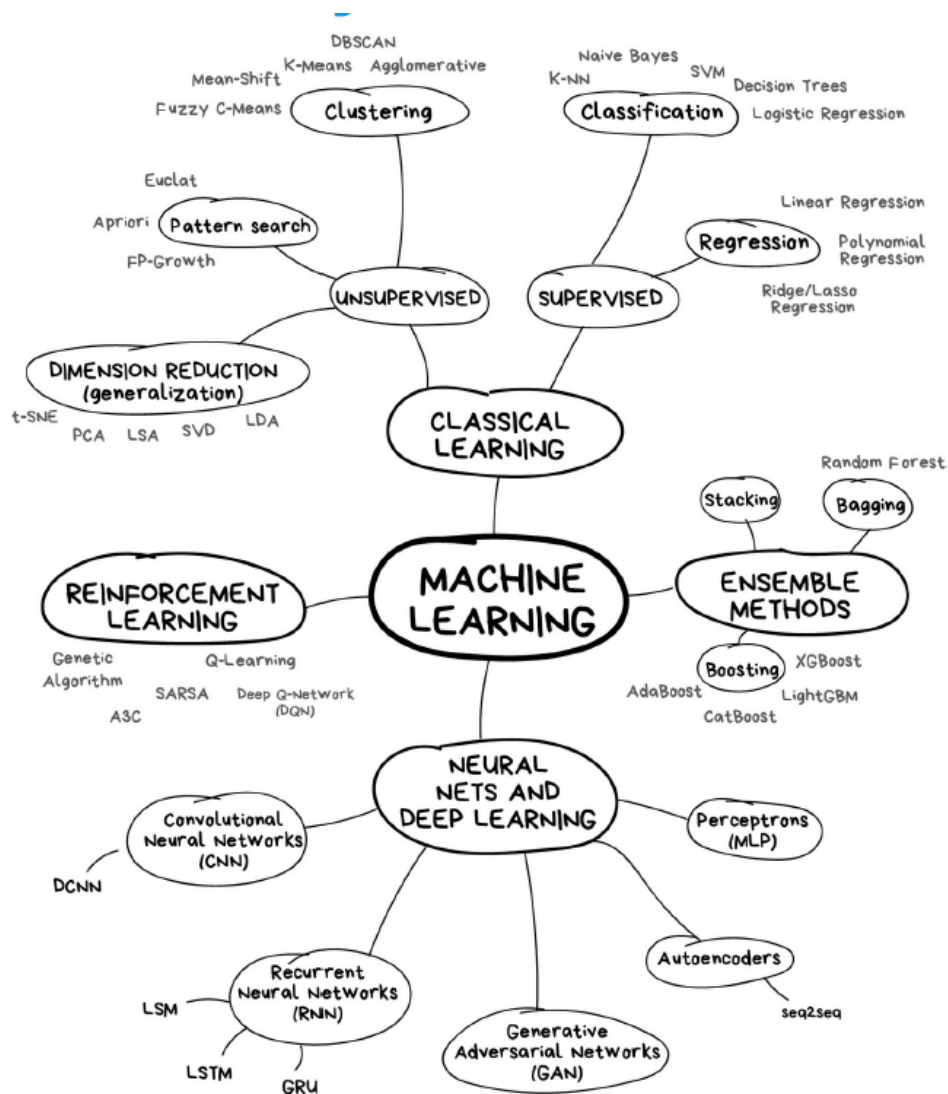


Image credit: https://vas3k.com/blog/machine_learning/

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*.

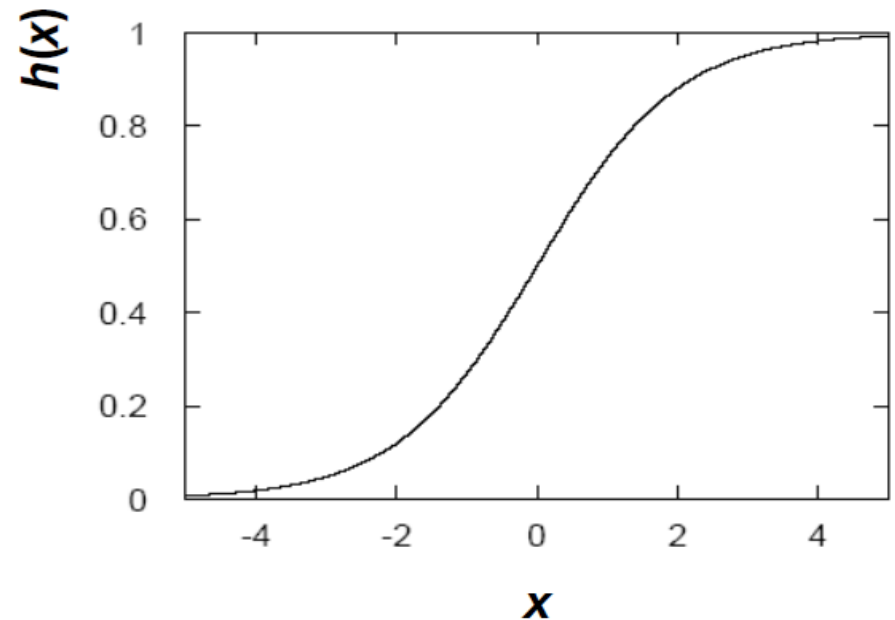
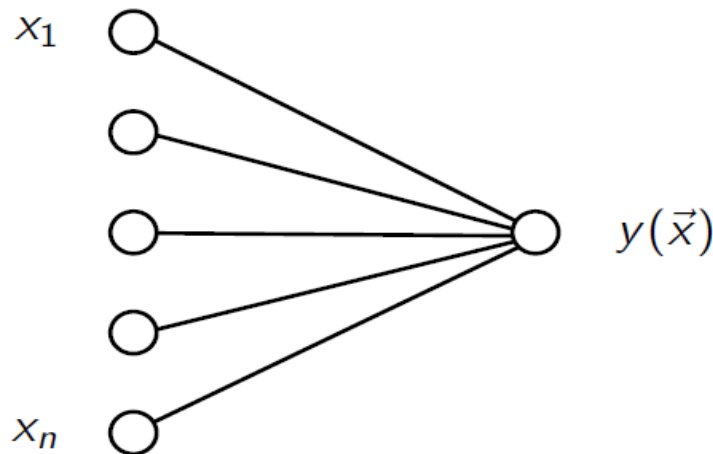
- These weights tell the neuron to respond more to one input and less to another. Weights are adjusted when training — that's how the network learns.

Perceptron

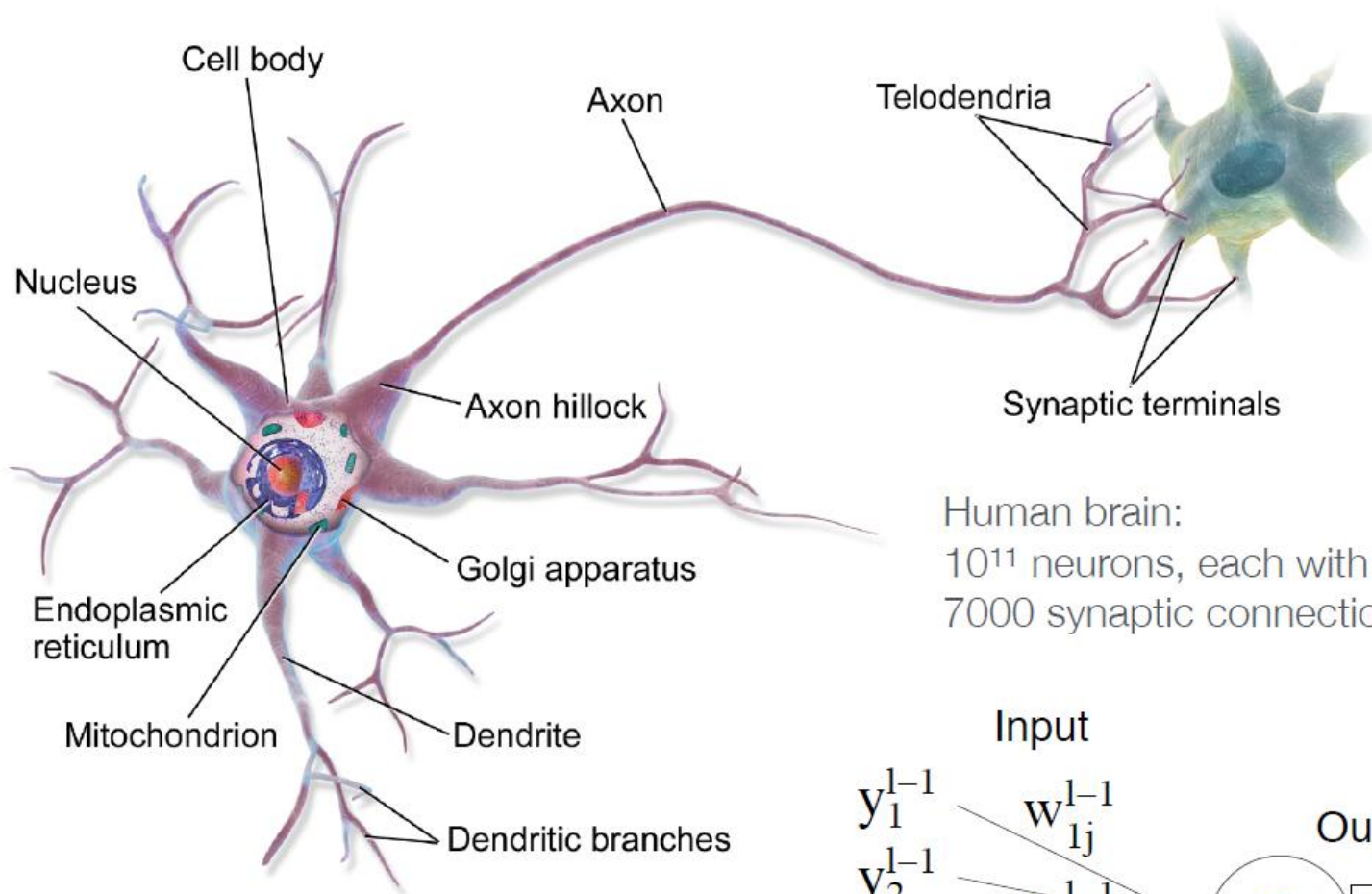
Discriminant:
$$y(\vec{x}) = h \left(w_0 + \sum_{i=1}^n w_i x_i \right)$$

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

Typical choices for h : $\frac{1}{1 + e^{-x}}$ ("sigmoid"), $\tanh x$

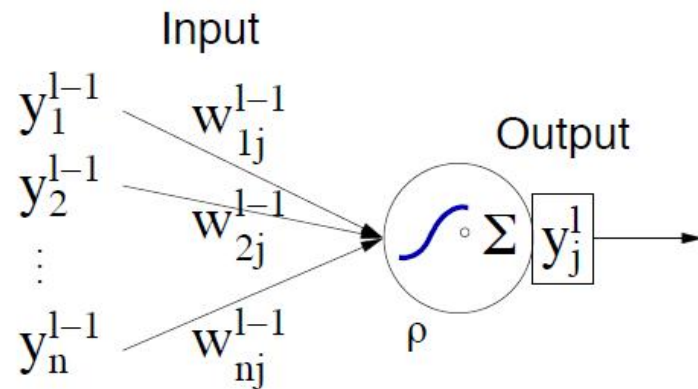


The Biological Inspiration: the Neuron

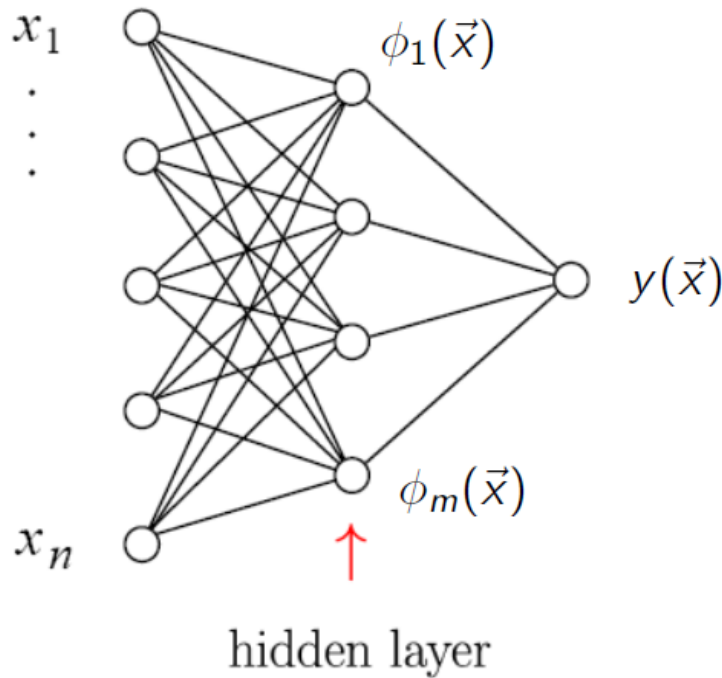


Human brain:
10¹¹ neurons, each with on average
7000 synaptic connections

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



Feedforward Neural Network with One Hidden Layer



superscripts indicates layer number

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

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

Straightforward to generalize to multiple hidden layers

Network Training

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

t_a : correct label for training event a

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

\vec{w} : 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 and then update weights after each training event:

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

└── learning rate

Let's write network output as follows:

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

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

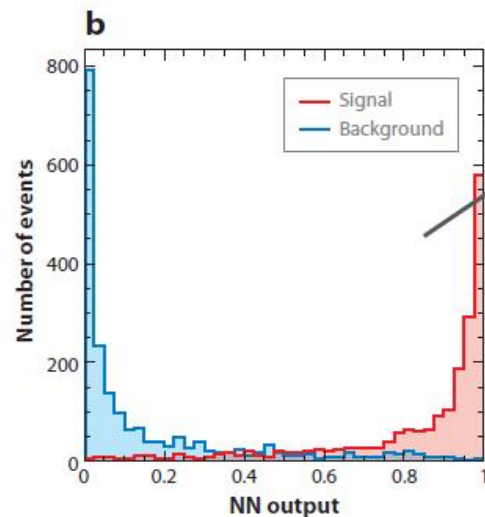
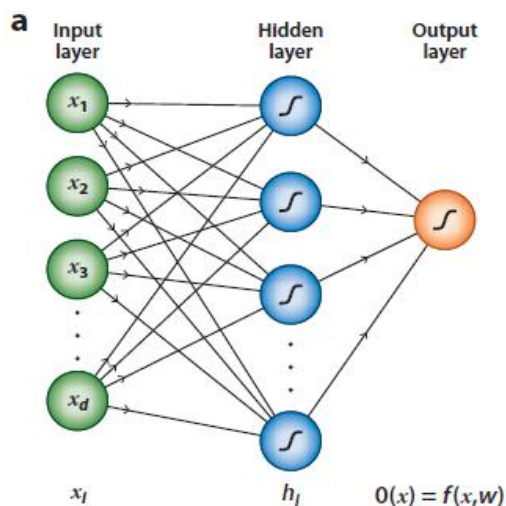
Weights from hidden layer to output:

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

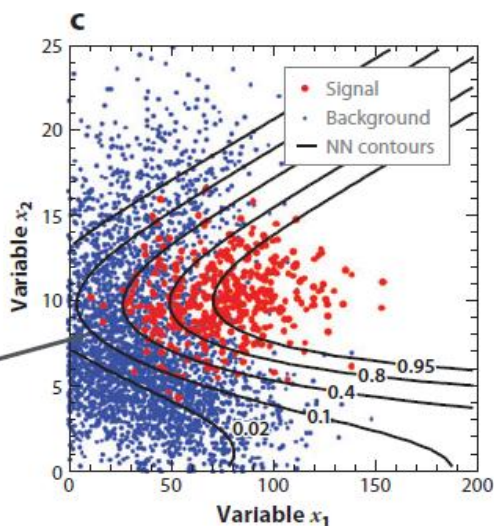
Weights from input layer to hidden layer (\rightarrow further application of chain rule):

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

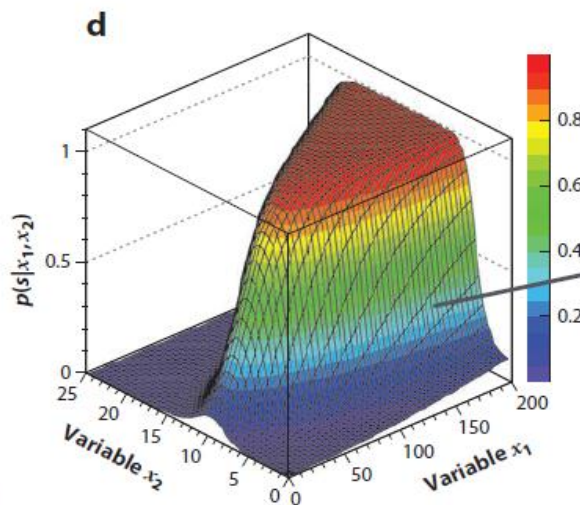
Neural Network Output and Decision Boundaries



output of neural network



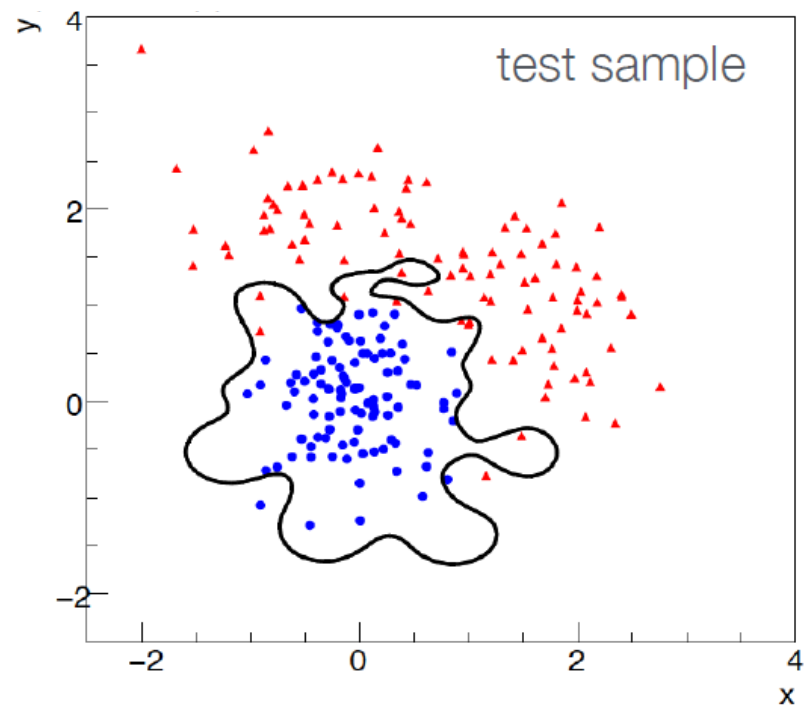
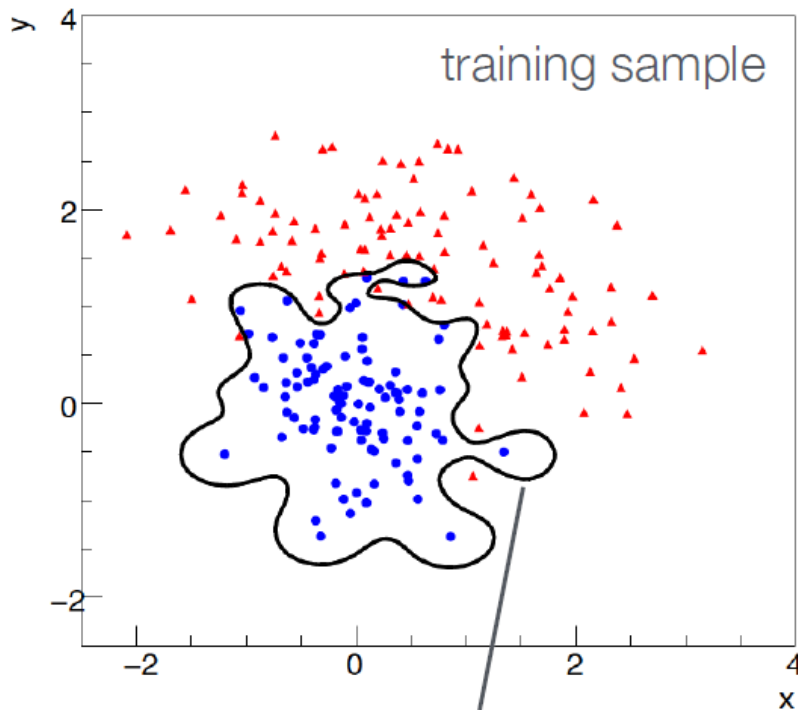
decision boundaries for different cuts on NN output



signal probability $p(s | x_1, x_2)$

Example of Overtraining

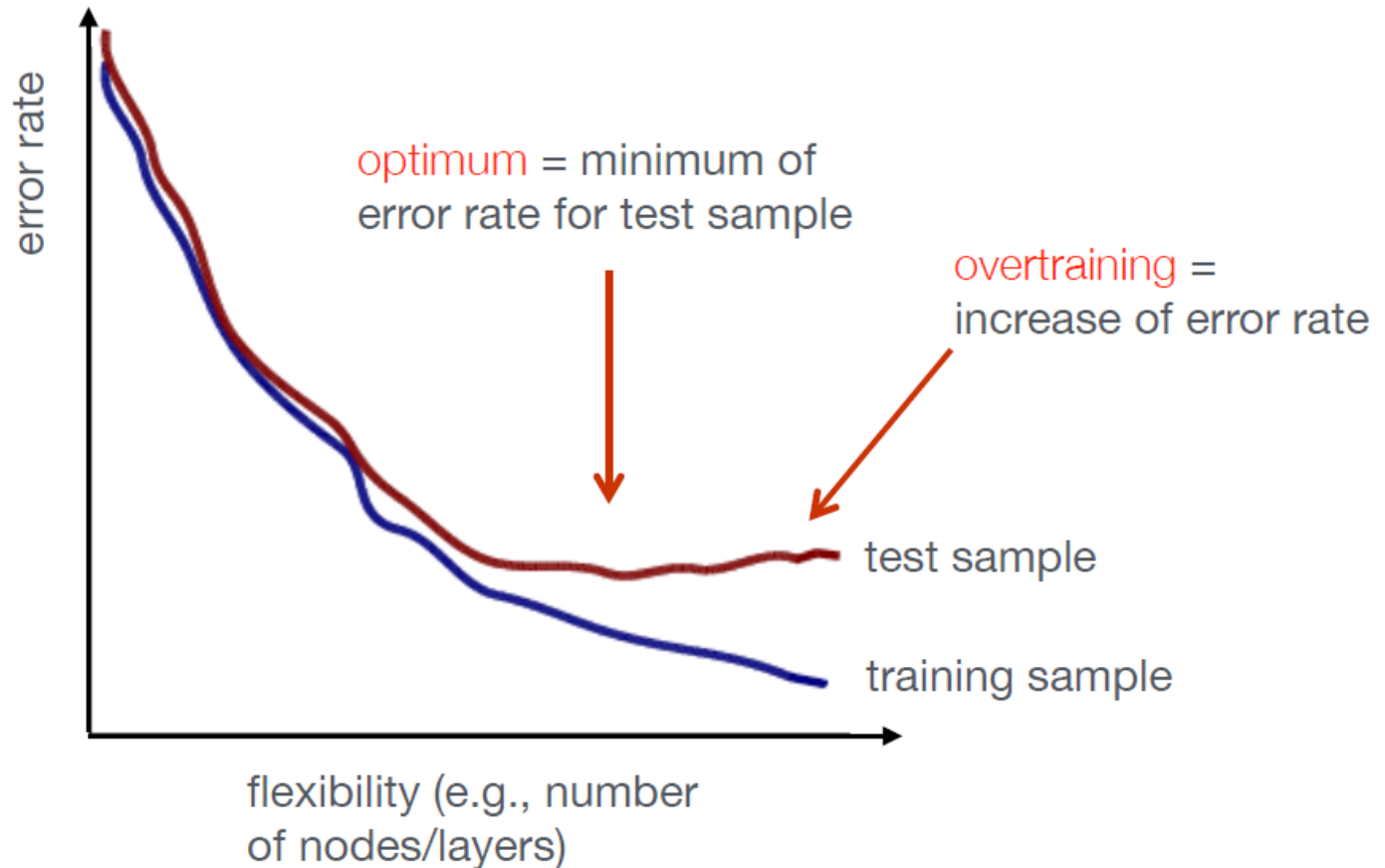
Too many neurons/layers make a neural network too flexible
→ overtraining



Network "learns" features that are merely statistical fluctuations in the training sample

Monitoring Overtraining

Monitor fraction of misclassified events (or error function:)



Deep Neural Networks

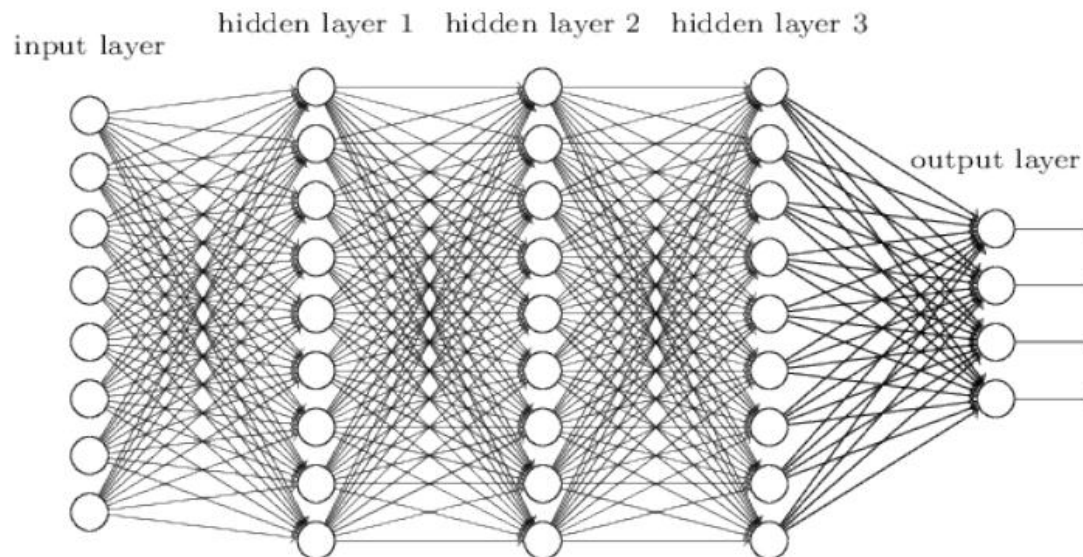
Deep networks: many hidden layers with large number of neurons

Challenges

- ▶ Hard to train ("vanishing gradient problem")
- ▶ Training slow
- ▶ Risk of overtraining

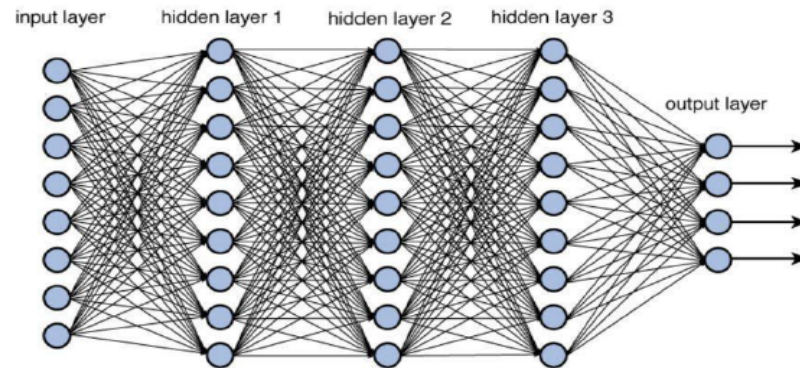
Big progress in recent years

- ▶ 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?



layer

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

activation function

weights

bias

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

How do NNs learn?

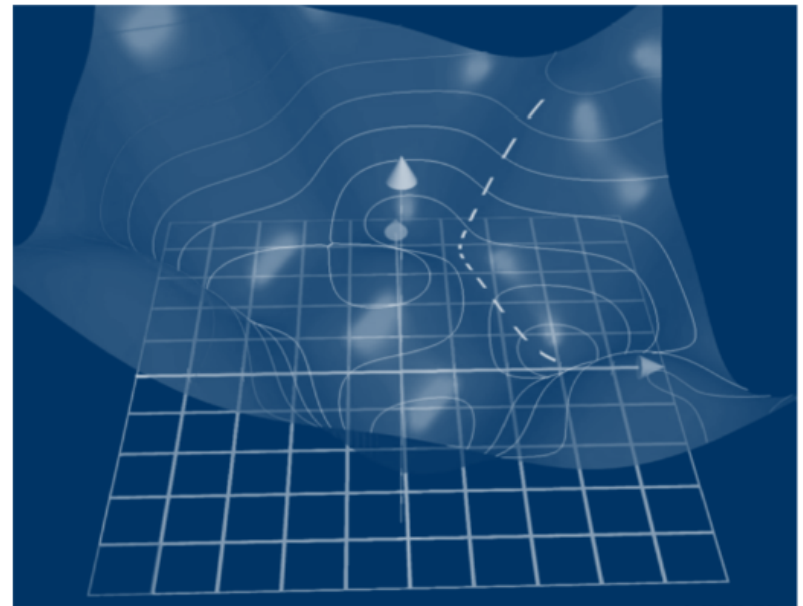
After we constructed a network, our task is to **assign proper weights** so neurons will react correctly to incoming signals.

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

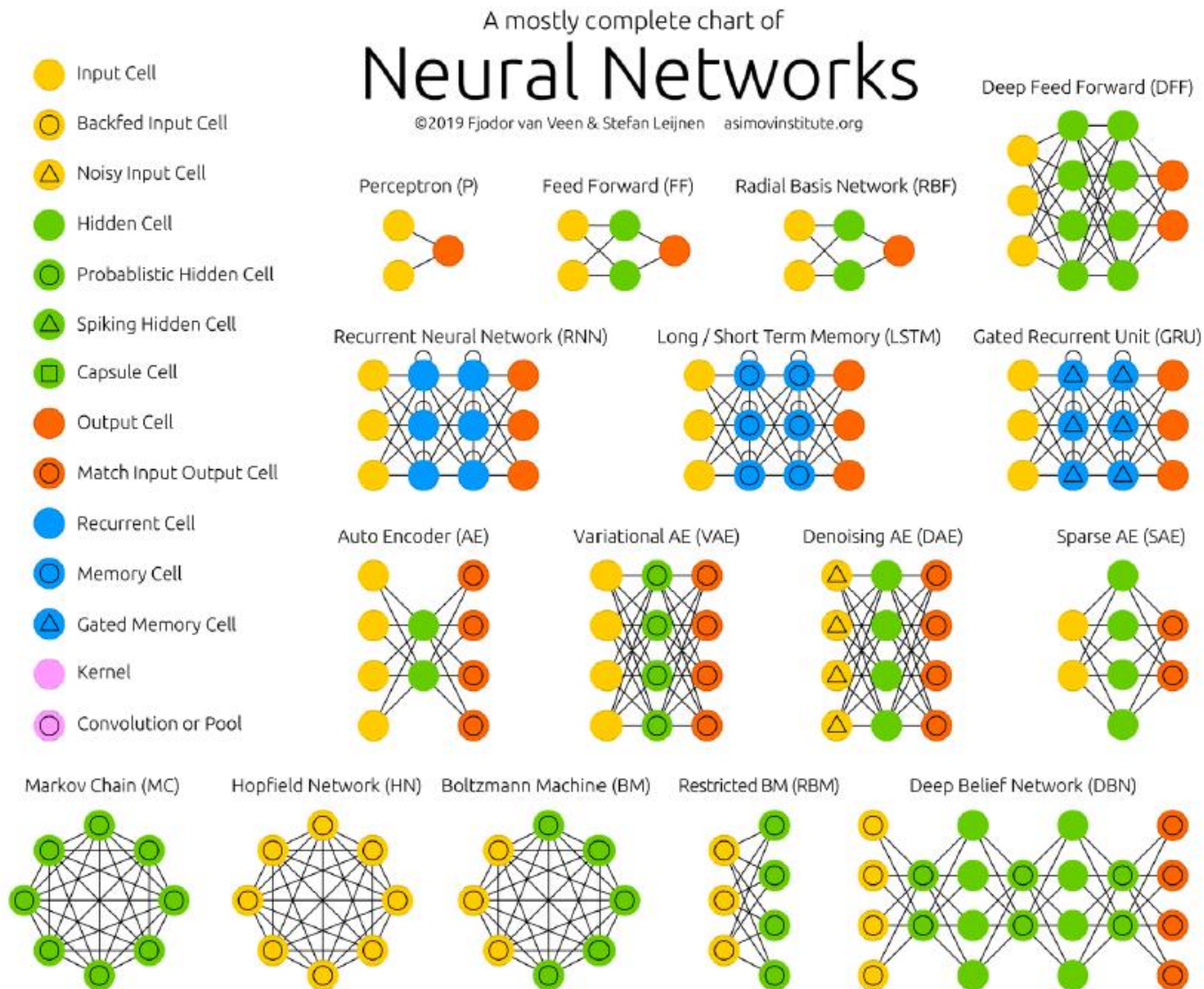
This function is a function of all the weights and biases in the NN (a priori a very large number), and the goal of training is to find its minimum.

- 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?



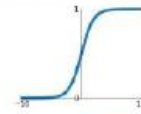
There are many more...

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 function (non linearity).**

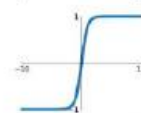
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$



Leaky ReLU

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

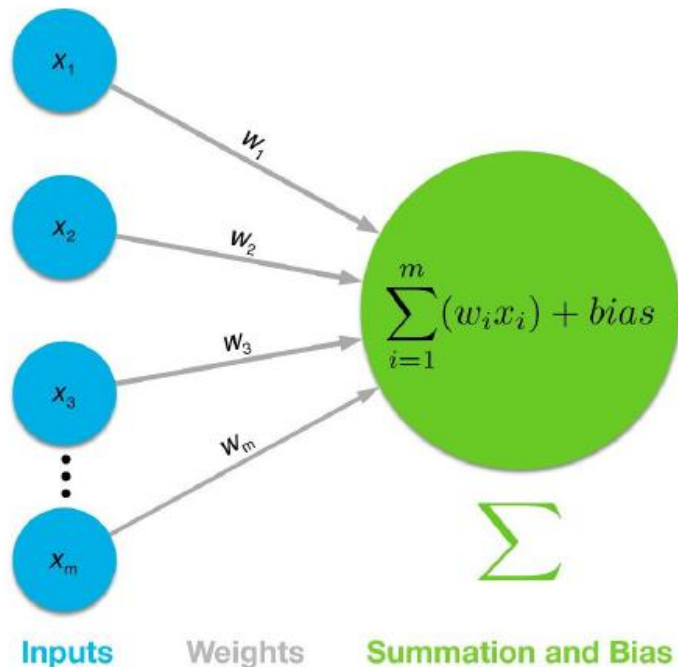
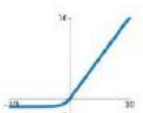


Maxout

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

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



$$\hat{y} = f \left(\sum_{i=1}^m (w_i x_i) + bias \right)$$

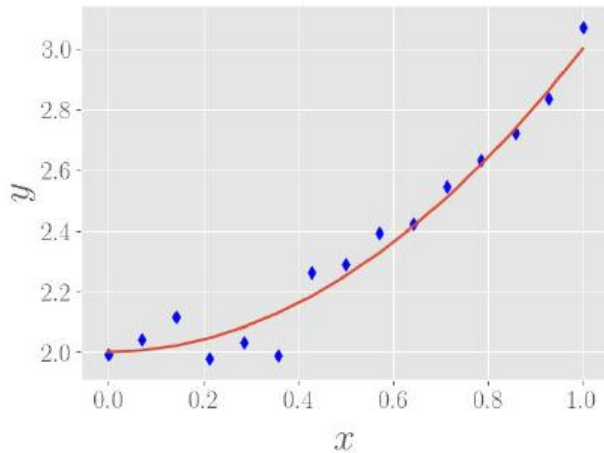
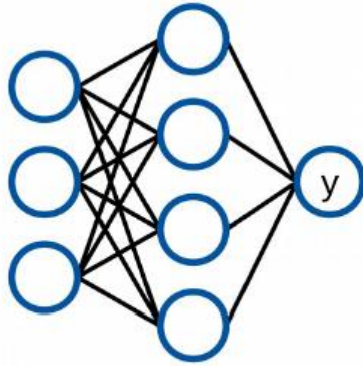
$$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$$

There's a colorful world inside black boxes!

Typical Applications

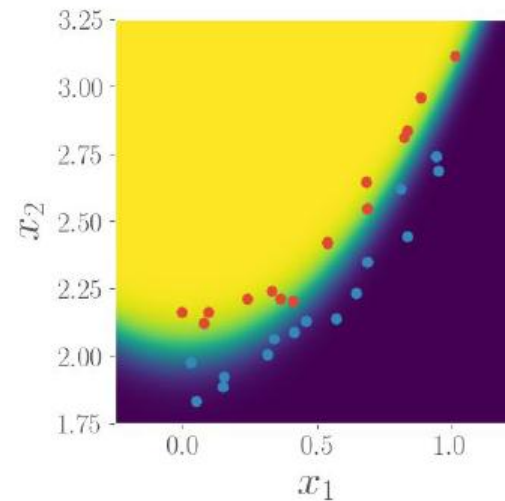
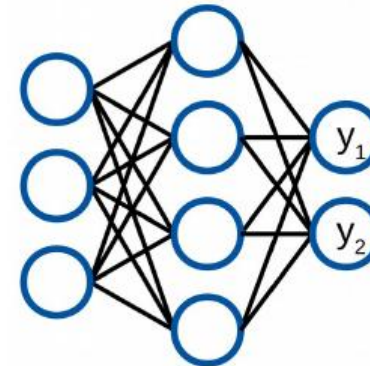
Regression:

Predict a continuous label.



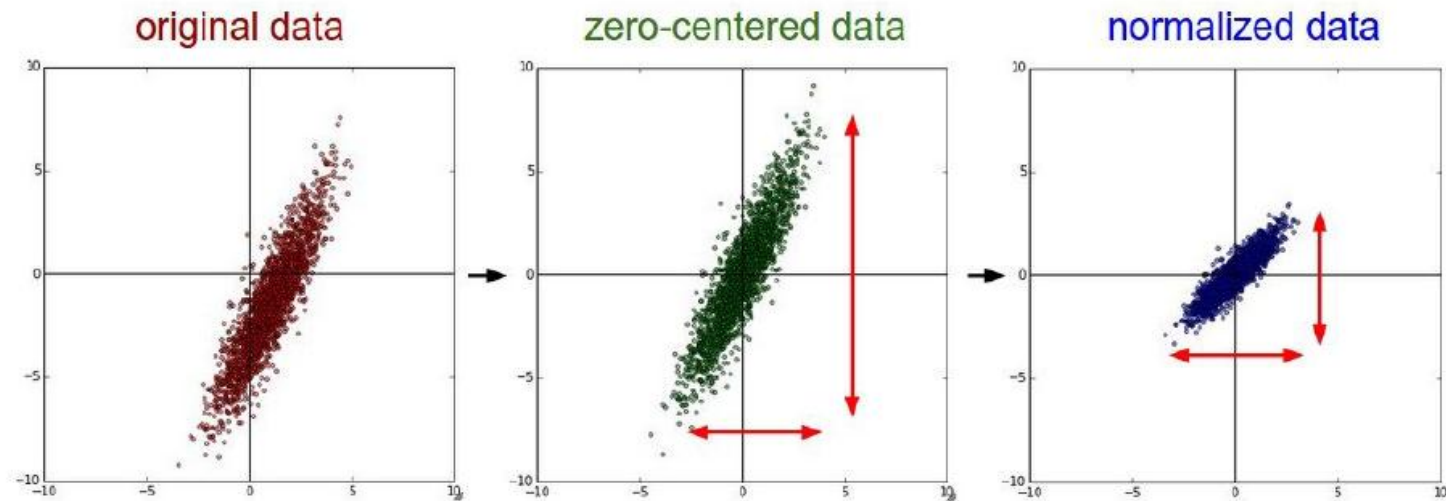
Classification:

Separate events into multiple categories.



Input Preprocessing

- ◆ 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.

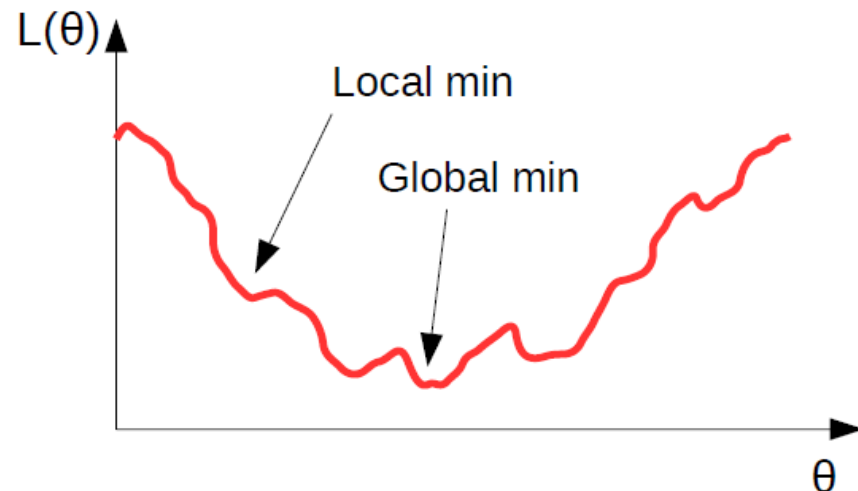


- ◆ Other options possible: decorrelation, non-linear transformation, ...

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 $L(\theta)$:
 - ◆ The loss function 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.



Training: (Stochastic) Gradient Descent

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

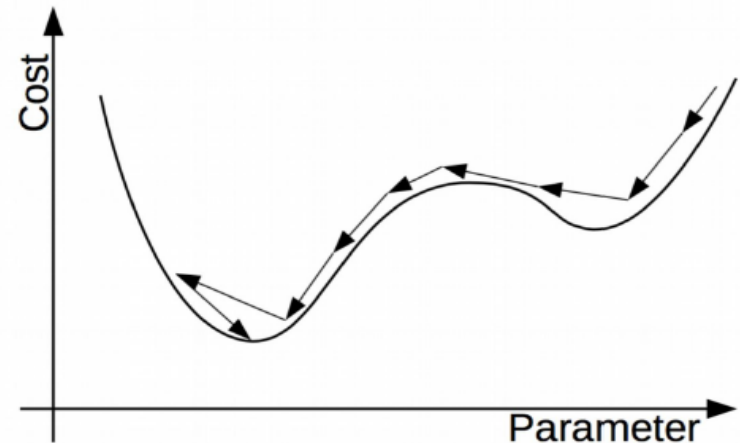
Iterative minimization by updating θ in opposite direction of gradient.

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

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

Stochastic gradient descent:

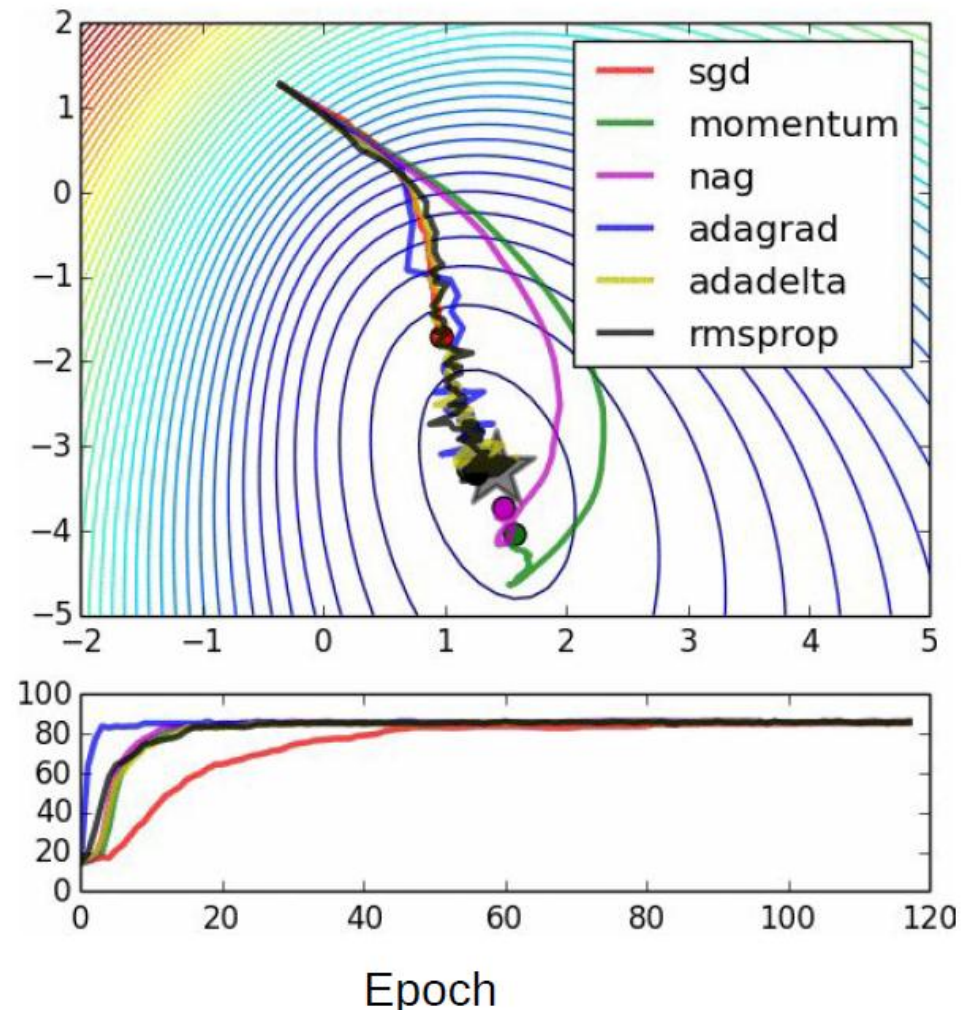
Calculate gradient for a small stochastic subset of the training sample (batch). → This also helps to avoid local minima!



One iteration over the full training dataset is called *epoch*.

Training: more optimisers

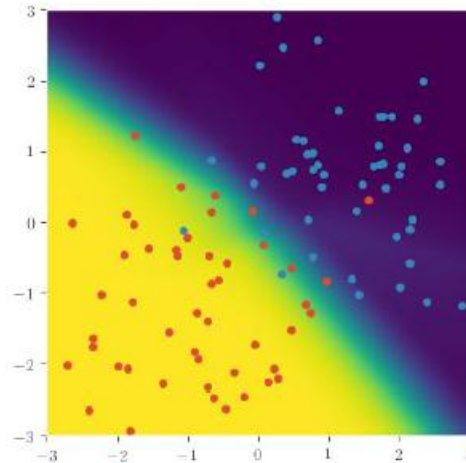
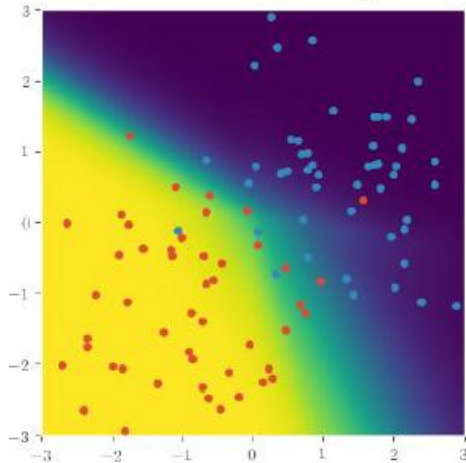
- ▶ More advanced options than fixed learning rate.
 - ◆ Momentum: past gradients used as « velocity »
 - ◆ Adaptive methods: different learning rates for each parameter and as a function of past gradients.



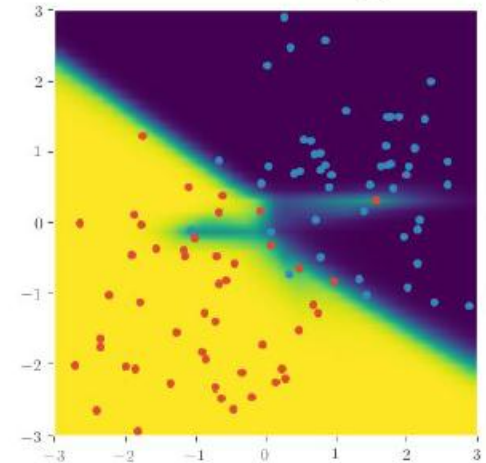
Underfitting and overtraining

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

Underfitting



Overfitting



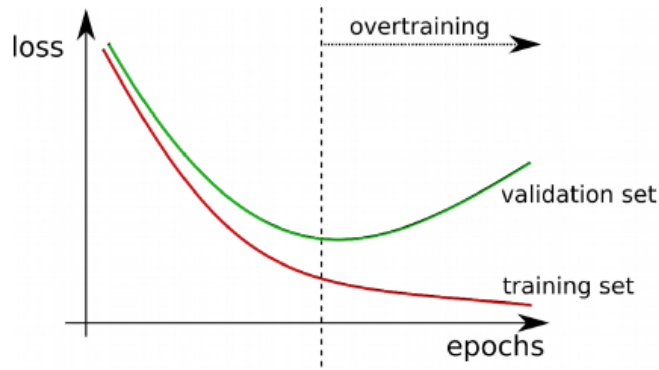
Overfitting: If model capacity is too high, network can « memorize » training samples
→ bad generalization.

Overtraining solutions

Early stop:

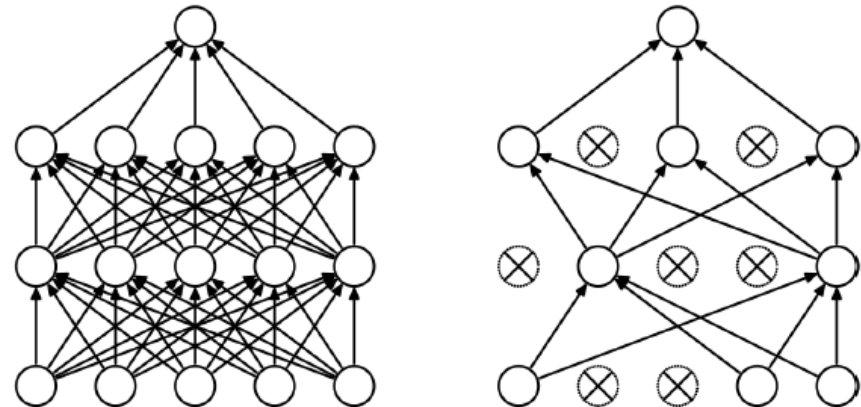
Evaluate the performance of the network on a validation dataset.

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.



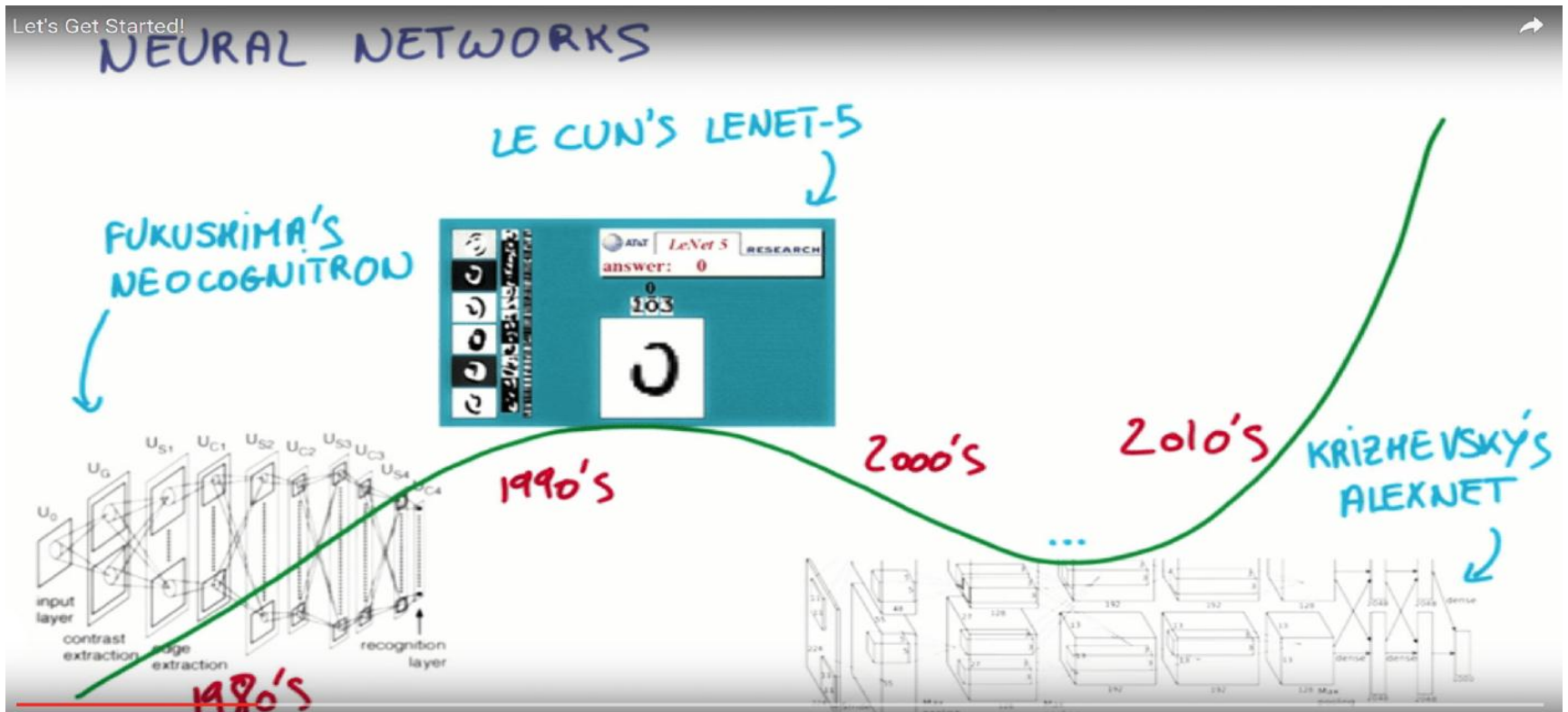
Deep-learning Neural Network

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

NEURAL NETWORKS → DEEP LEARNING

- 2009: SPEECH RECOGNITION
- 2012: COMPUTER VISION
- 2014: MACHINE TRANSLATION

① DATA



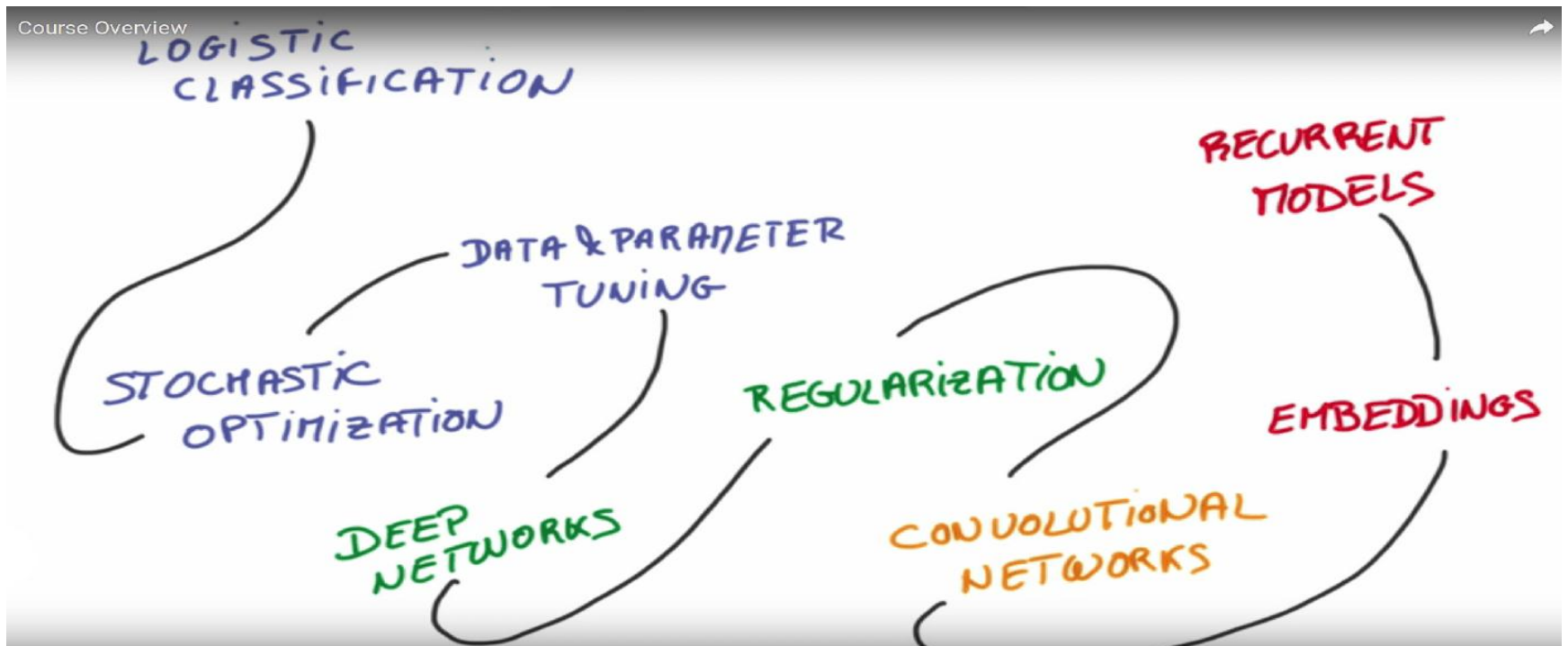
② GPUS



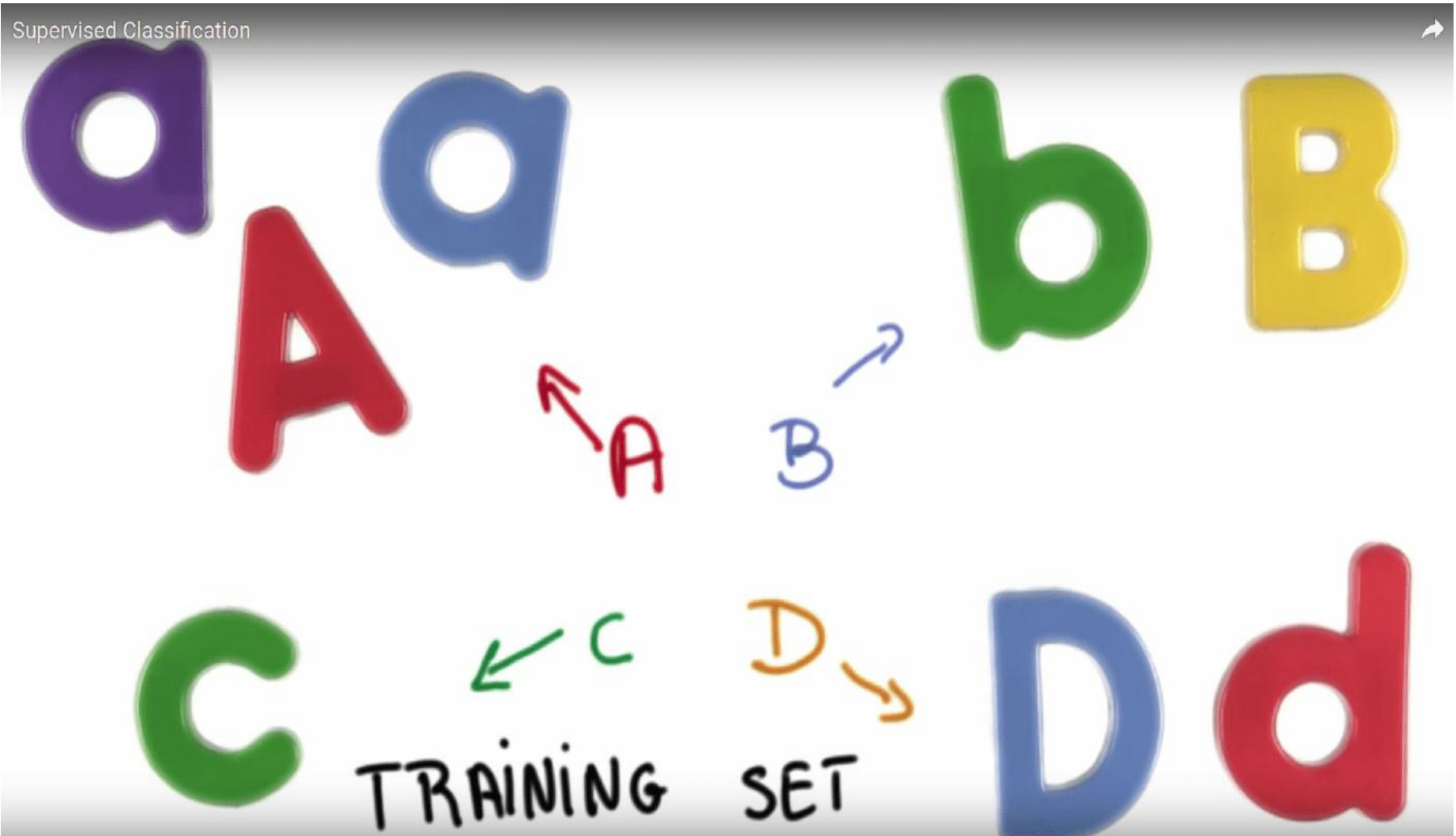
THANK YOU
GAMERS!!!

Deep-Learning tutorial @ udacity

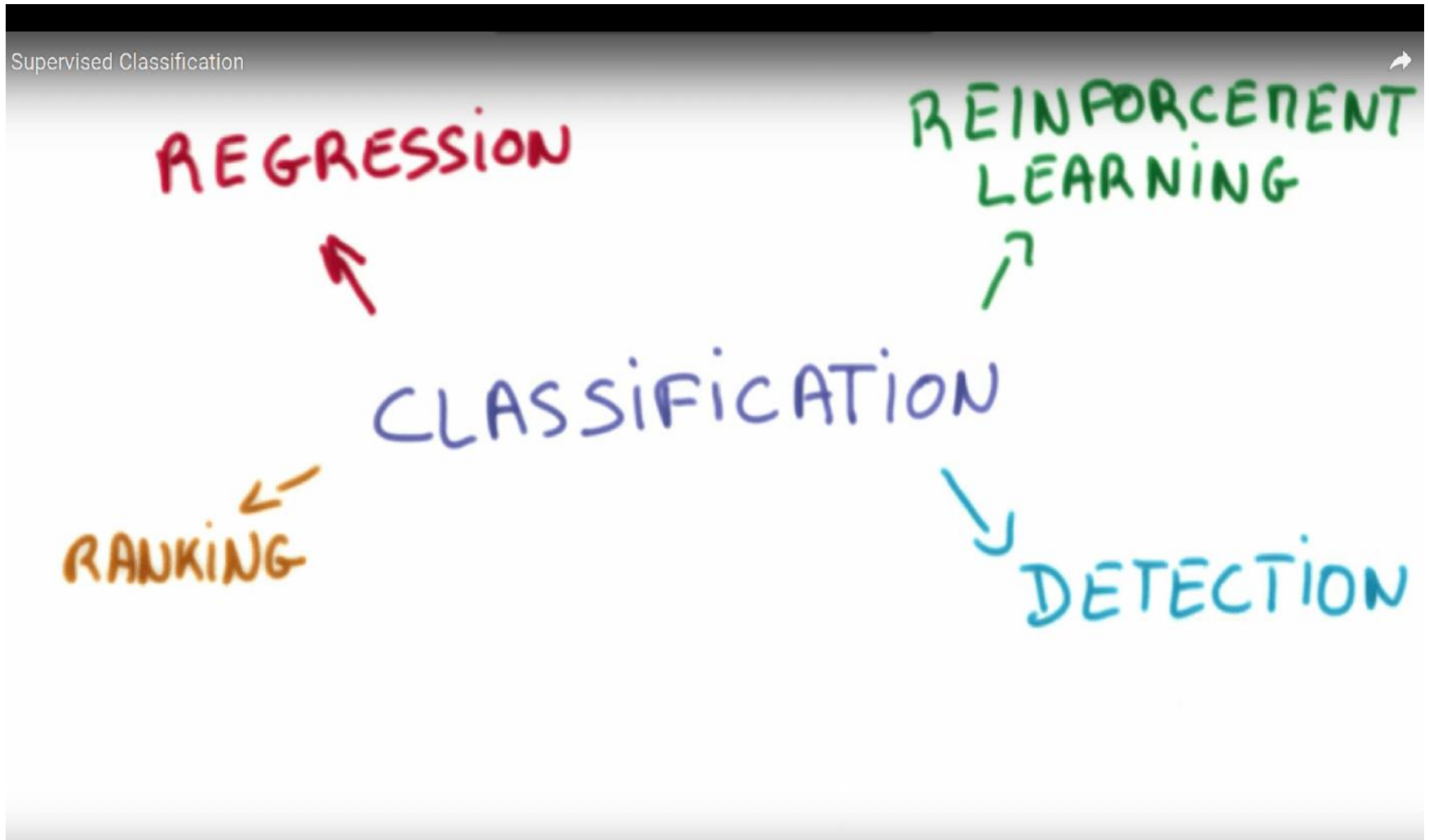
<https://www.udacity.com/course/deep-learning--ud730>



Supervised Classifications



Supervised Classifications

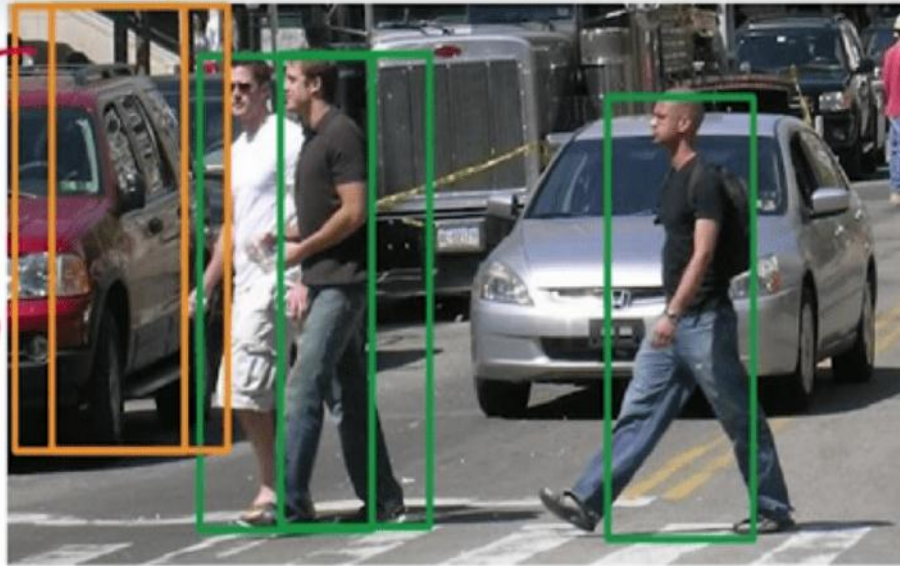


Classifications for Detection

Classification For Detection Solution

DETECTION

PEDESTRIAN
VS.
NO PEDESTRIAN



Use a binary pedestrian / no pedestrian classifier. Slide it over all possible locations in the image.

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 Alpaydin, 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

Training Your Logistic Classifier

LOGISTIC CLASSIFIER

A

$W X + b = Y$

WEIGHTS

BIAS

a

b

c

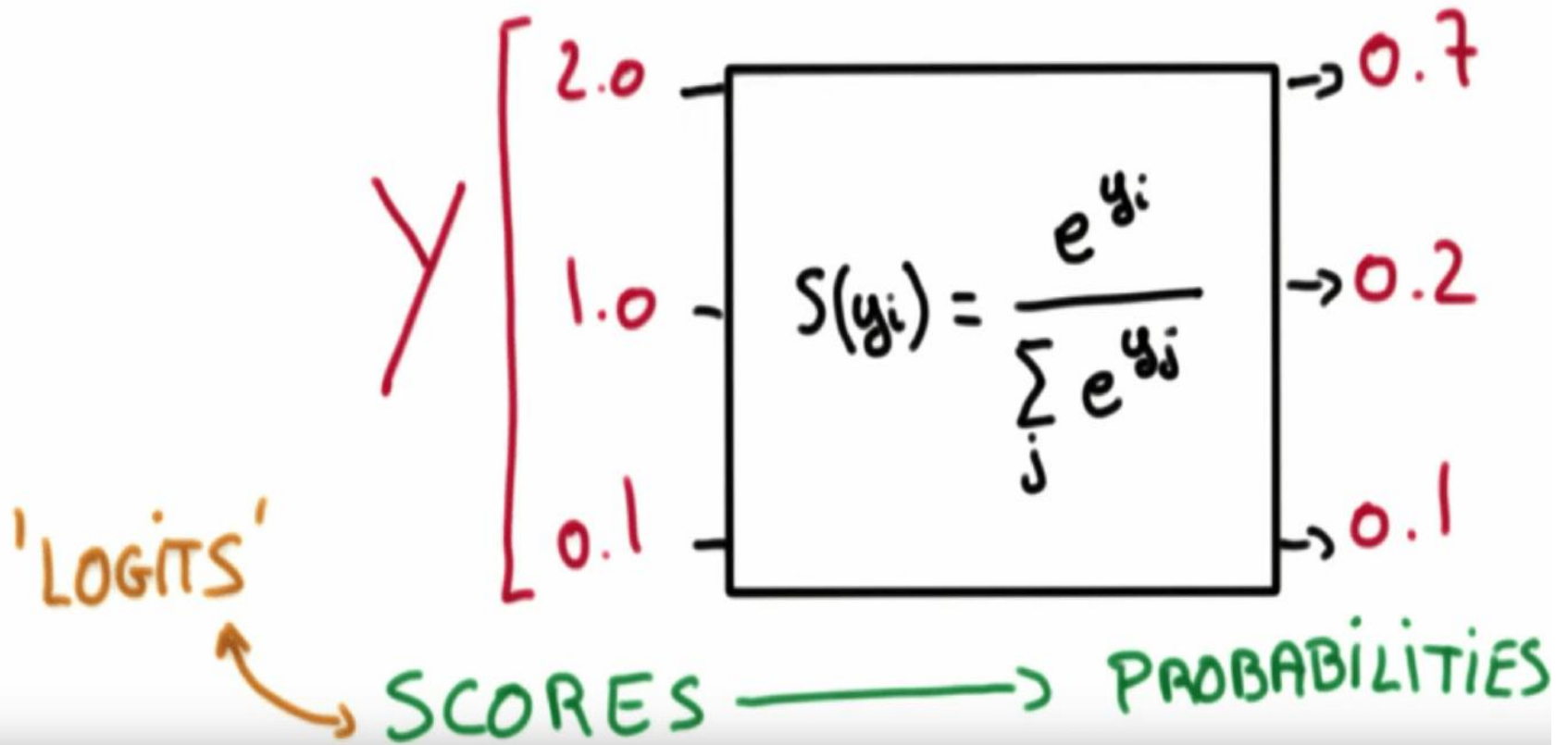
TRAINED

The diagram illustrates the process of a linear model in a logistic classifier. At the top left, the text "LOGISTIC CLASSIFIER" is written in blue. Below it, a large purple letter "A" represents the input. An arrow points down from "A" to the equation $W X + b = Y$. In this equation, "W" is orange, "X" is blue, "+" is green, "b" is orange, "=" is green, and "Y" is red. Below "W" is the word "WEIGHTS" with an arrow pointing up to "W". Below "b" is the word "BIAS" with an arrow pointing up to "b". To the right of the equation, three arrows point to the letters "a", "b", and "c", which are colored yellow, blue, and green respectively. At the bottom, the word "TRAINED" is written in blue with an arrow pointing to the equation.

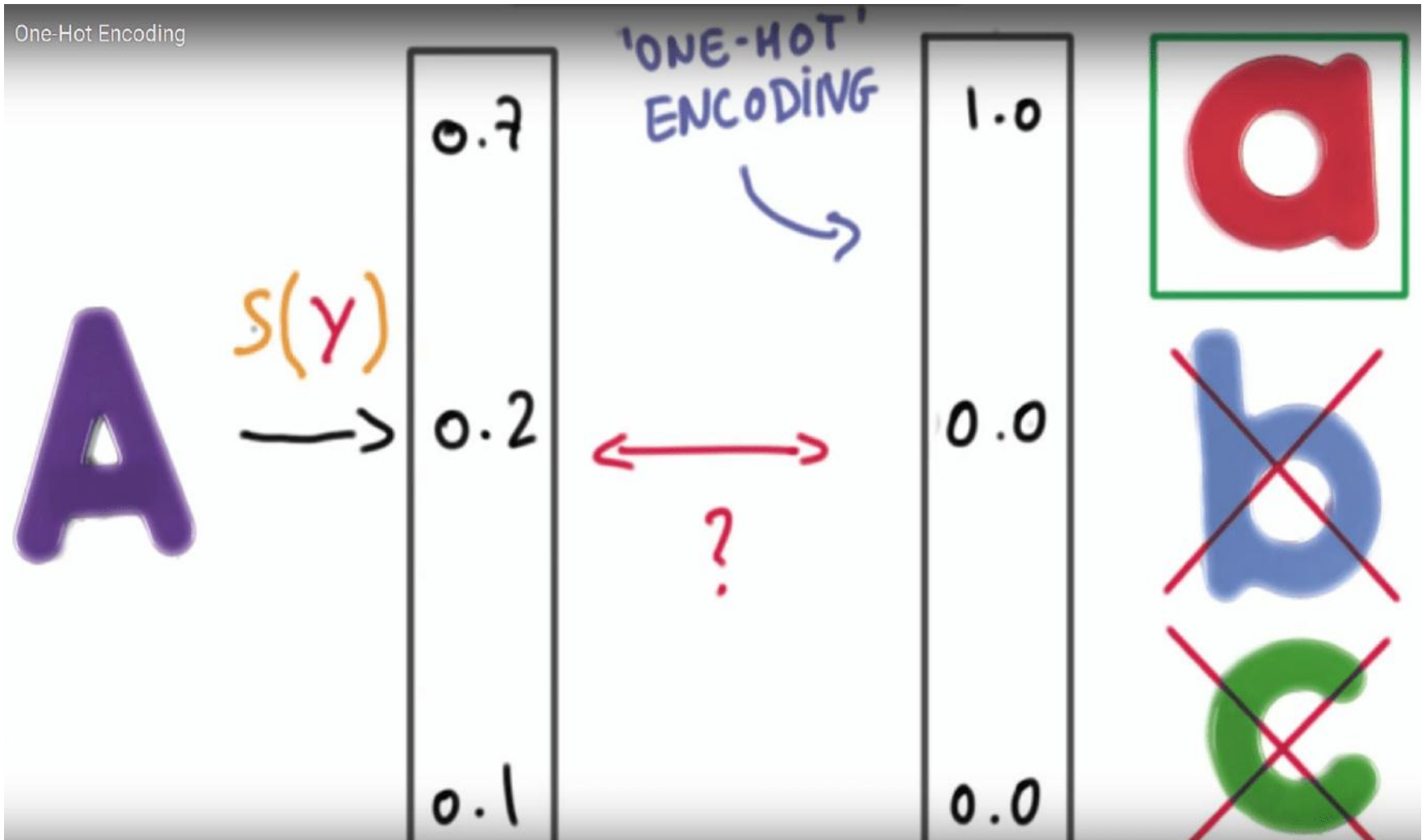
Softmax

Training Your Logistic Classifier

SOFTMAX



„One hot” encoding



„One hot” encoding

Cross Entropy

The diagram shows a grid of handwritten characters. On the left, a vertical list of characters is shown with arrows pointing to the corresponding rows in the grid:

- a →
- b →
- c →
- d →
- ...
- 6
- 0

The grid consists of several columns of characters. The character 'd' in the fourth column is highlighted with a green oval and a red '1', indicating its active state in the encoding vector.

0:17 / 1:39

cc YouTube

Optimisation: Cross-Entropy

Cross Entropy

CROSS-ENTROPY

$S(Y)$

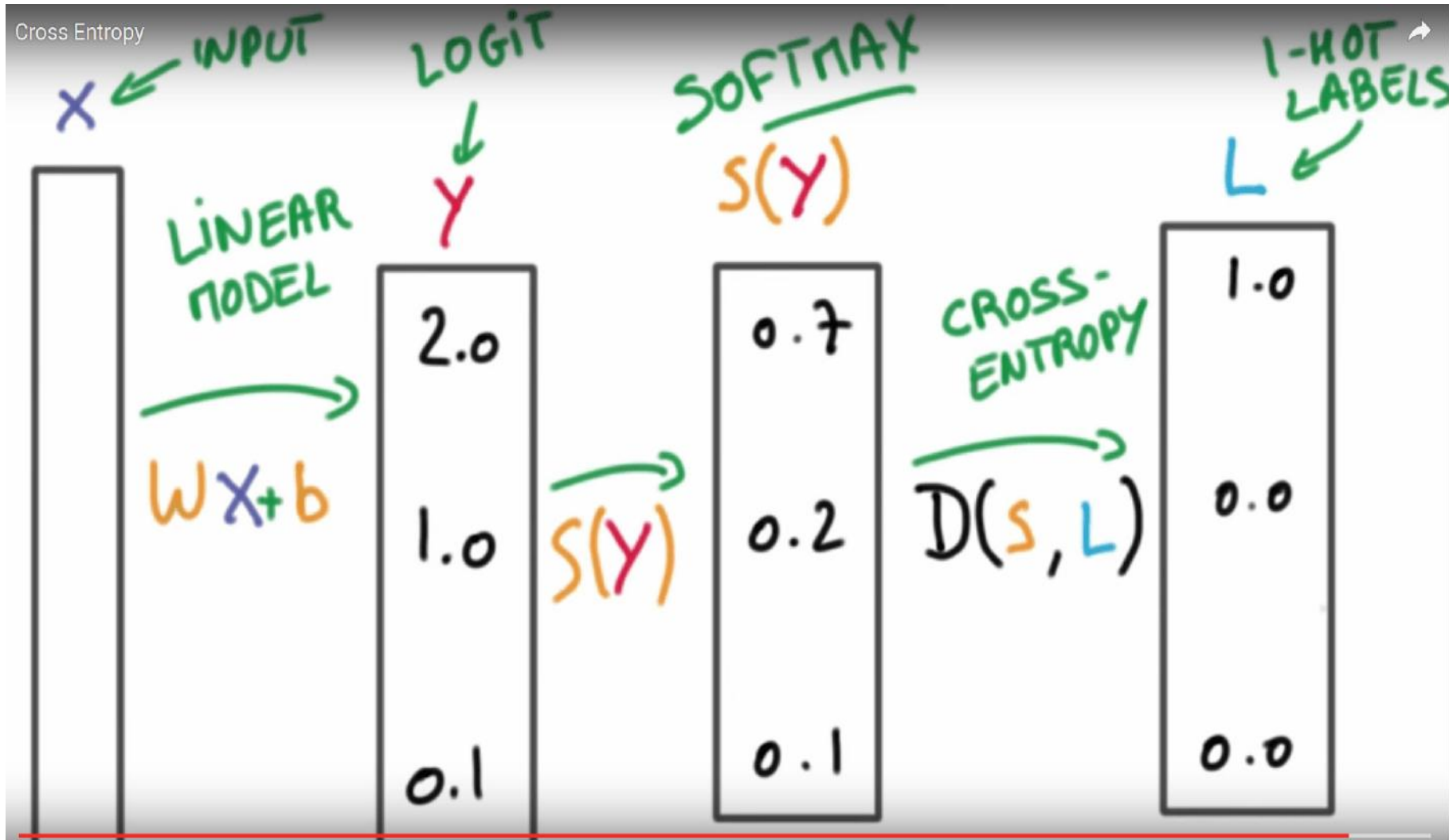
0.7
0.2
0.1

$$D(S, L) = - \sum_i L_i \log(S_i)$$

L

1.0
0.0
0.0

Multinomial logistic classification



Optimisation of average loss

Minimizing Cross Entropy

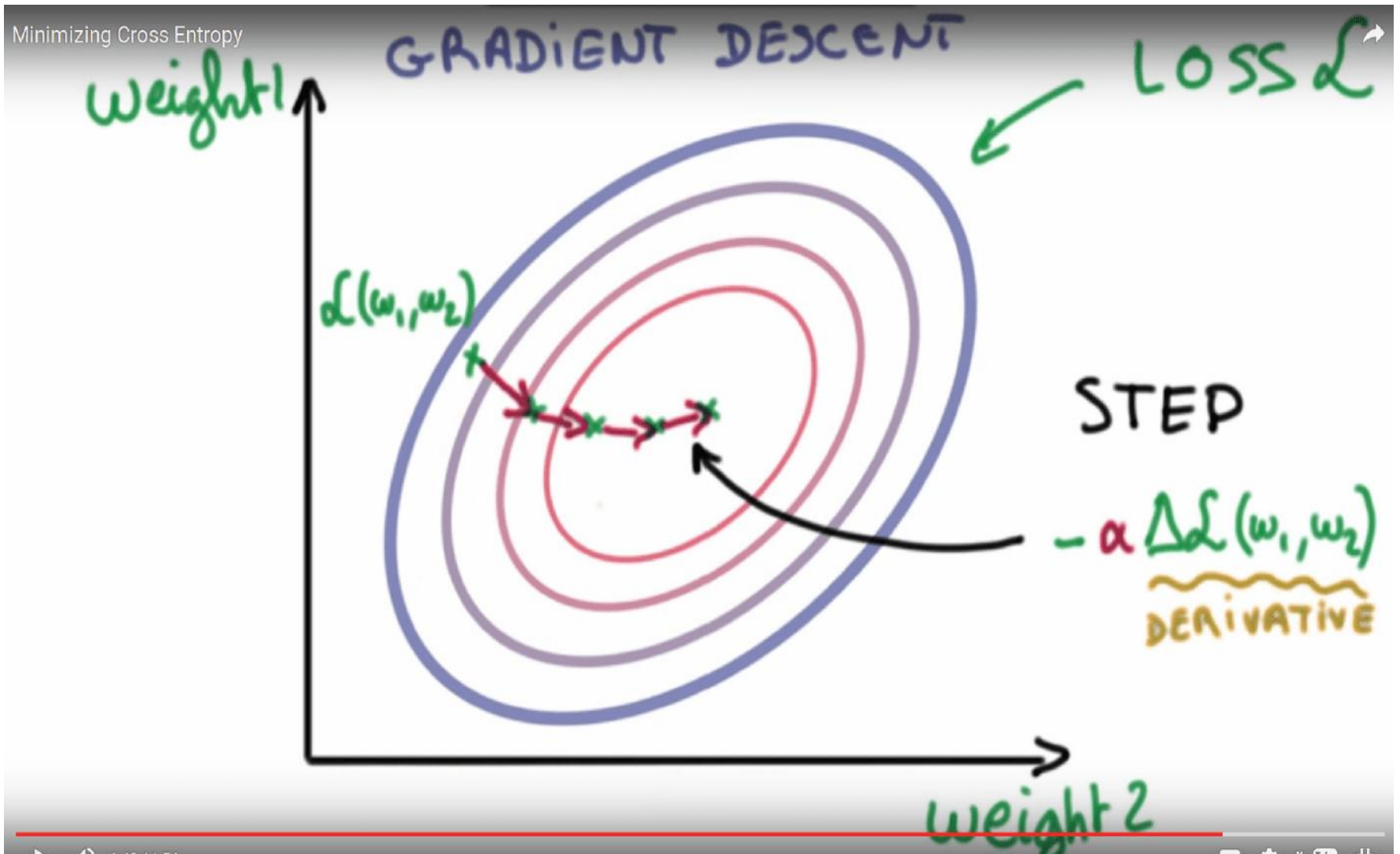
LOSS = AVERAGE CROSS-ENTROPY

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

BIG MATRIX!!

BIG SUM!!

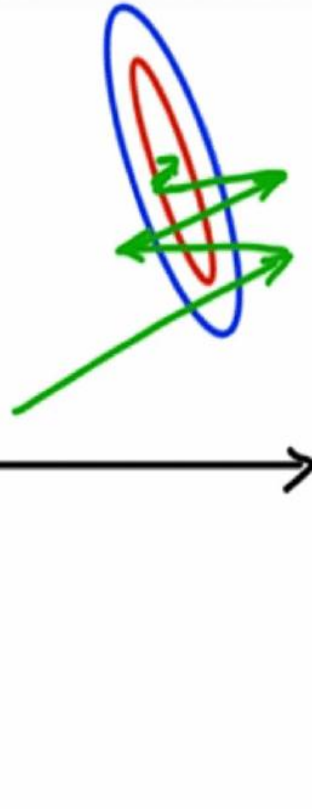
Gradient decent



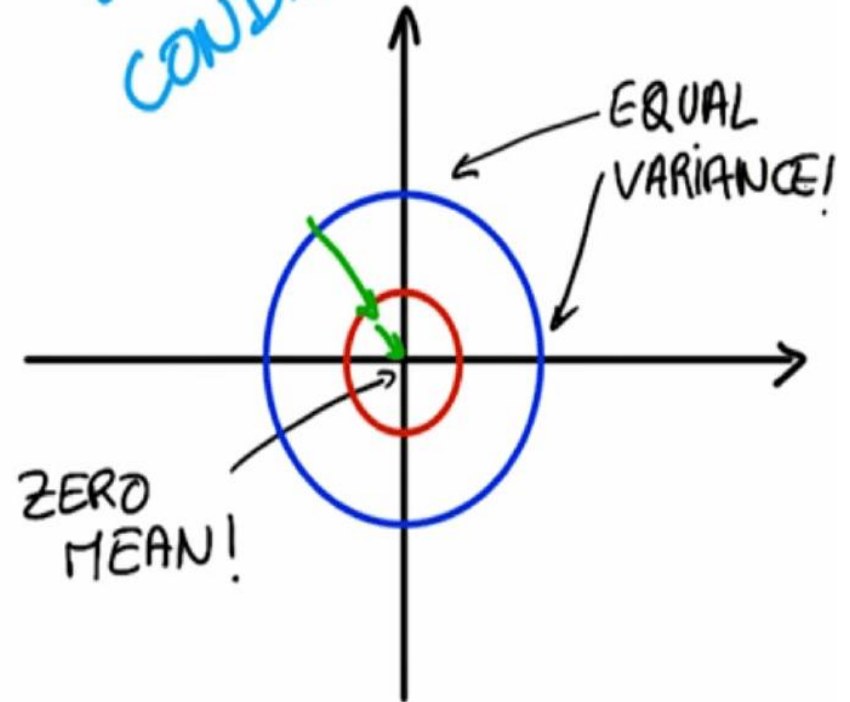
Normalised input and output

Normalized Inputs and Initial Weights

BADLY
CONDITIONED



WELL
CONDITIONED



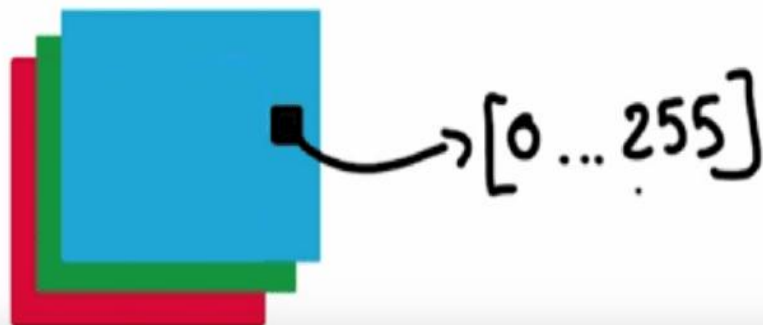
Normalised input and output

IMAGES

$$\frac{R - 128}{128}$$

$$\frac{G - 128}{128}$$

$$\frac{B - 128}{128}$$



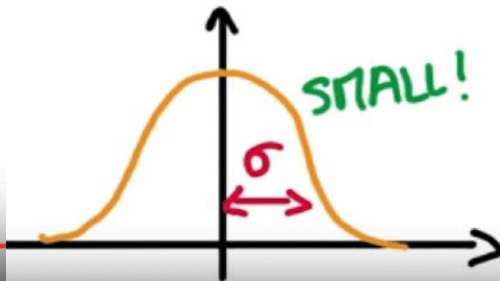
Initialisation

Normalized Inputs and Initial Weights

INITIALIZATION
OF THE LOGISTIC
CLASSIFIER

$$\frac{\text{pixels} - 128}{128}$$

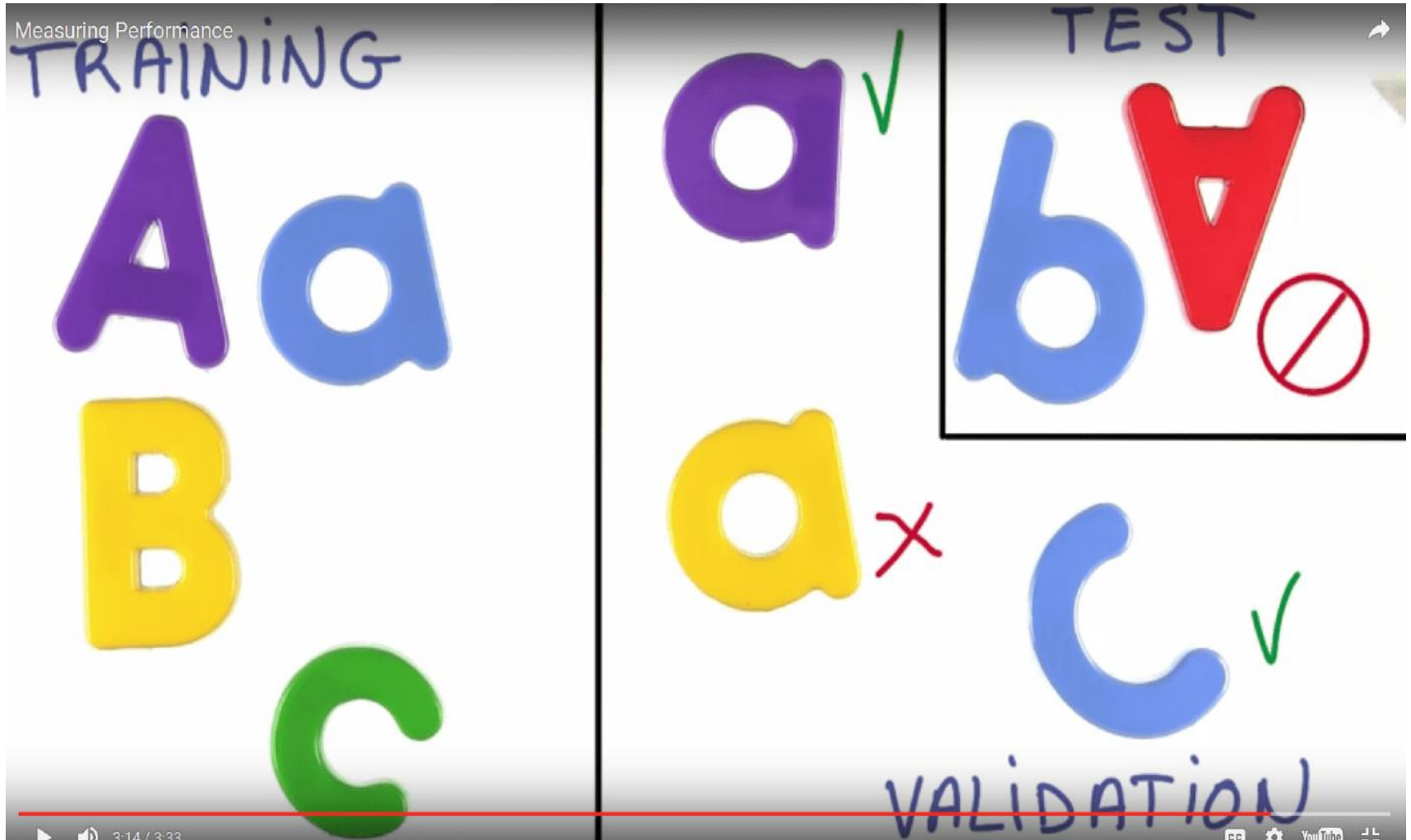
$$\mathcal{L} = \frac{1}{N} \sum_i \mathcal{D}(s(wX_i + b), L_i)$$



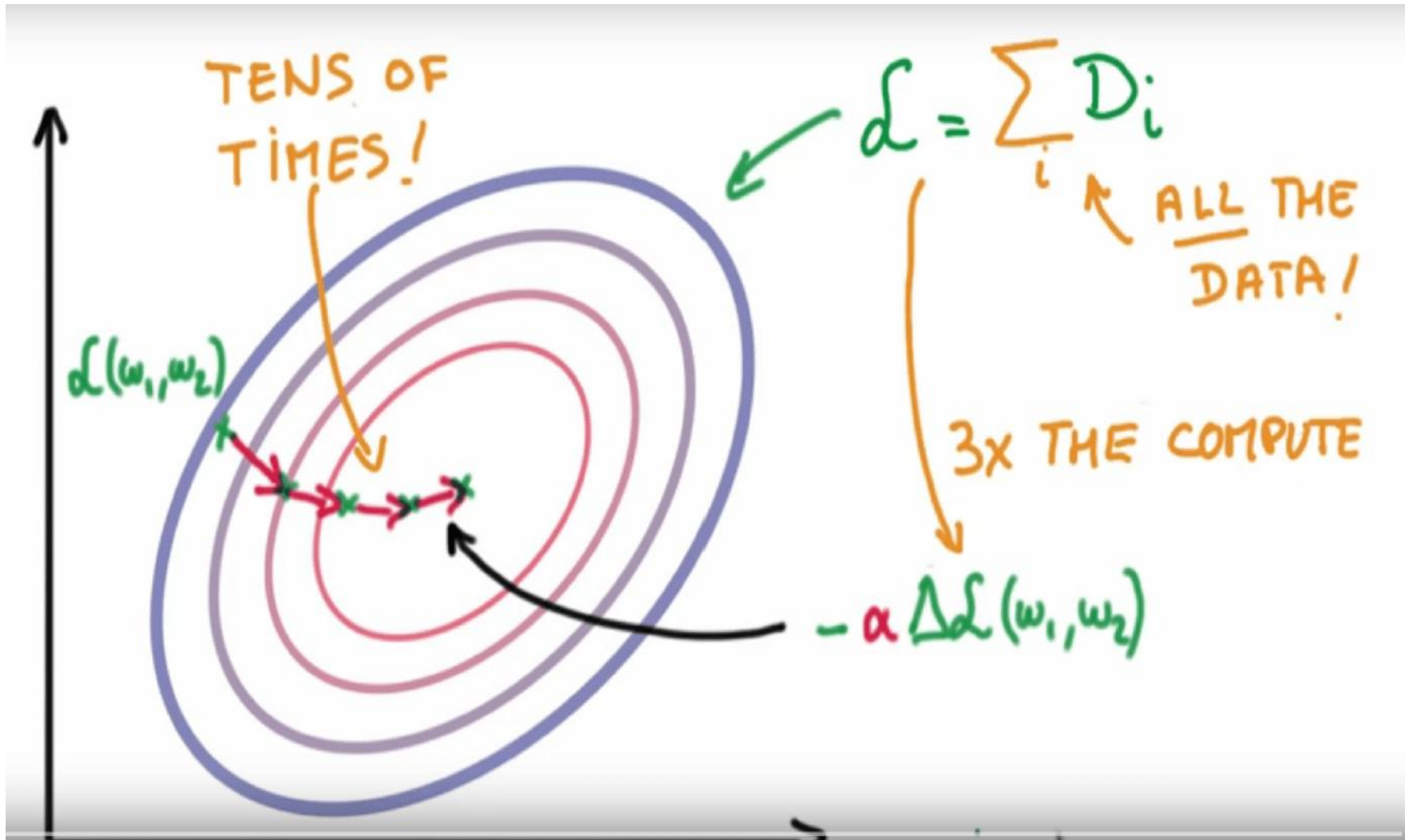
2:07 / 2:45

CC BY-NC-SA YouTube

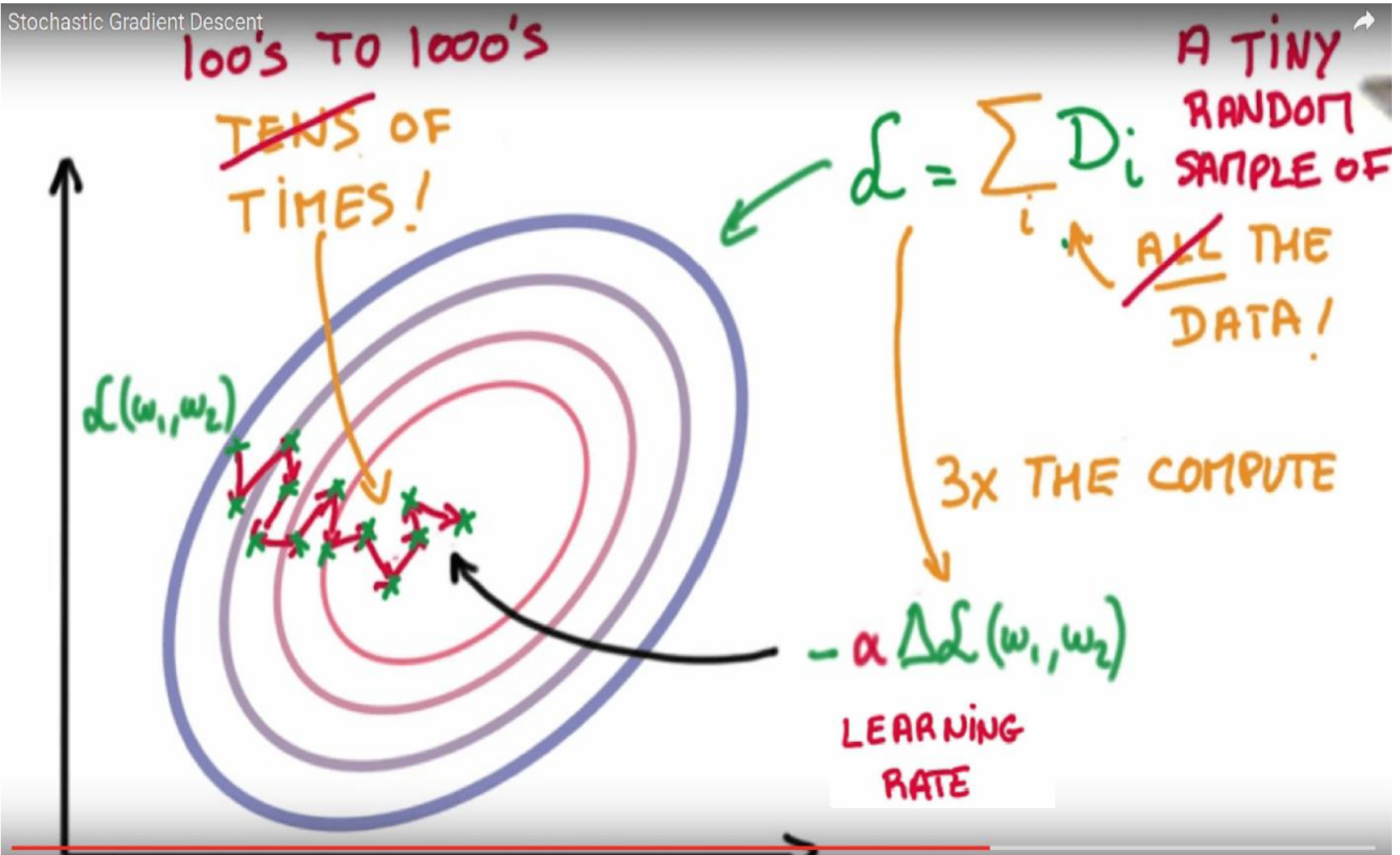
Training, validation, testing



Gradient Descent



Stochastic Gradient Descent



SDG: optimising with momentum

Momentum and Learning Rate Decay

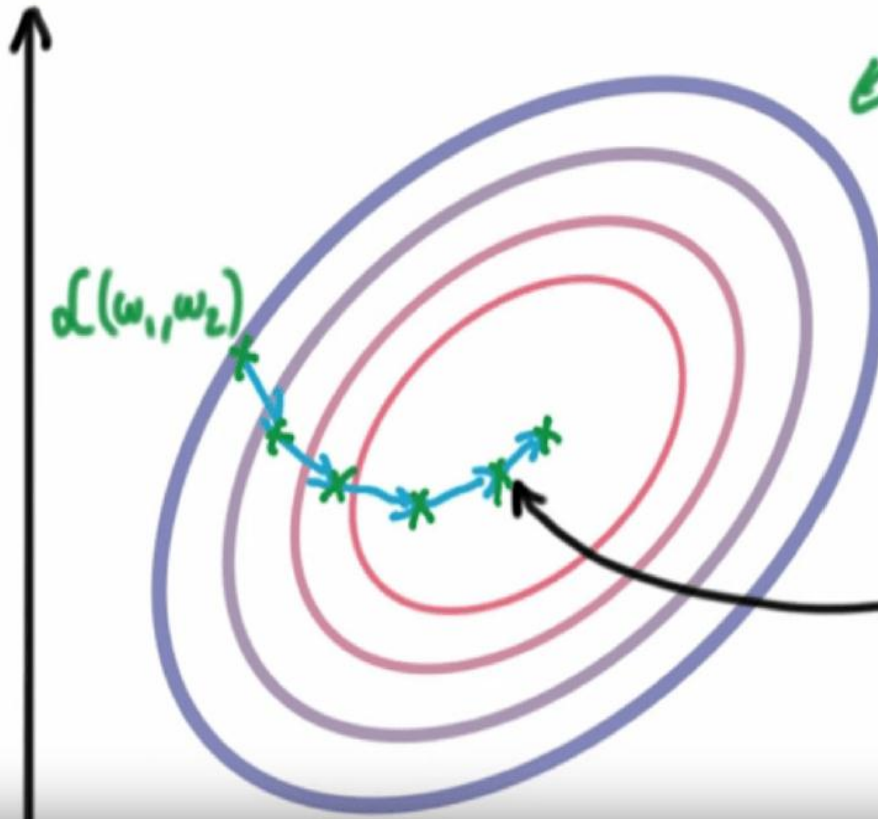
MOMENTUM

$$\mathcal{L} = \sum_i D_i$$

RUNNING AVERAGE

$$M \leftarrow 0.9M + \Delta \mathcal{L}$$

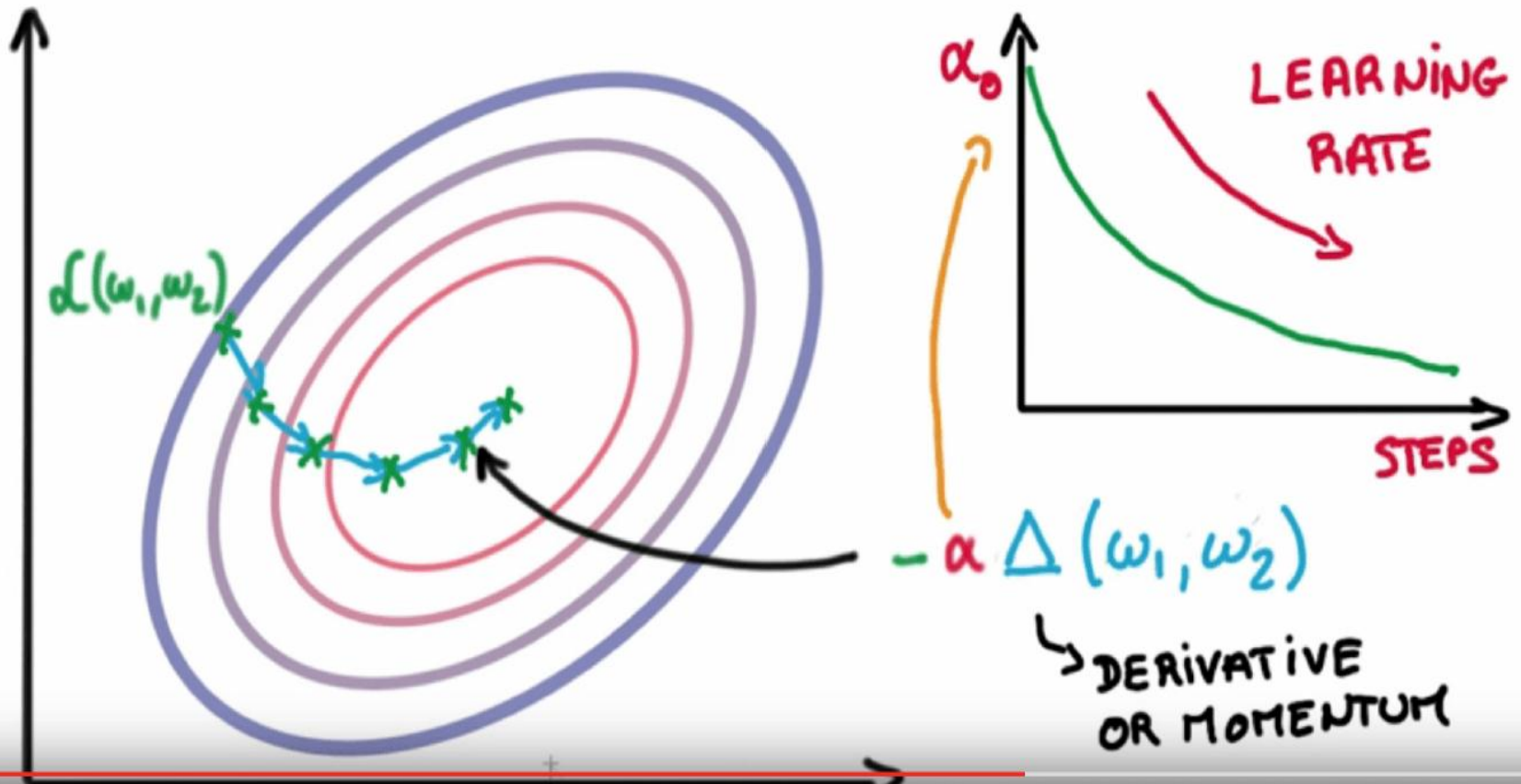
$$-\alpha \cancel{\Delta \mathcal{L}}(\omega_1, \omega_2) M(\omega_1, \omega_2)$$



SDG: learning rate

Momentum and Learning Rate Decay

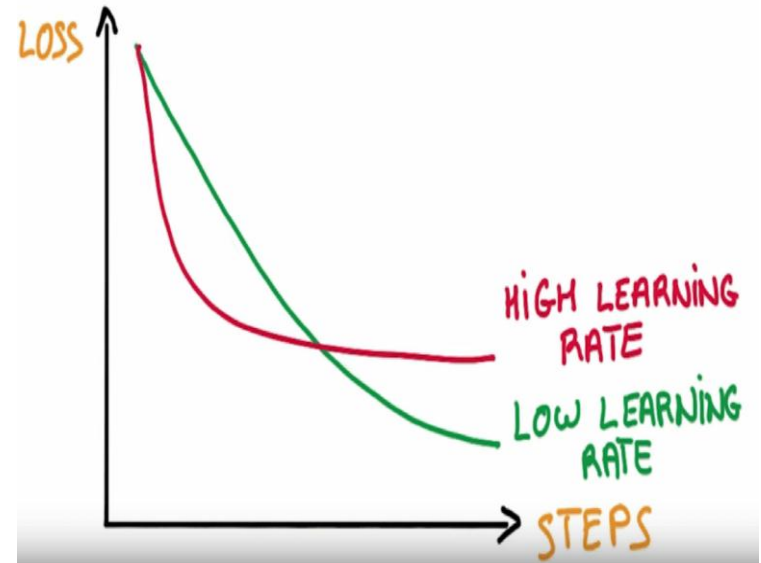
LEARNING RATE DECAY



SDG: „black magic”

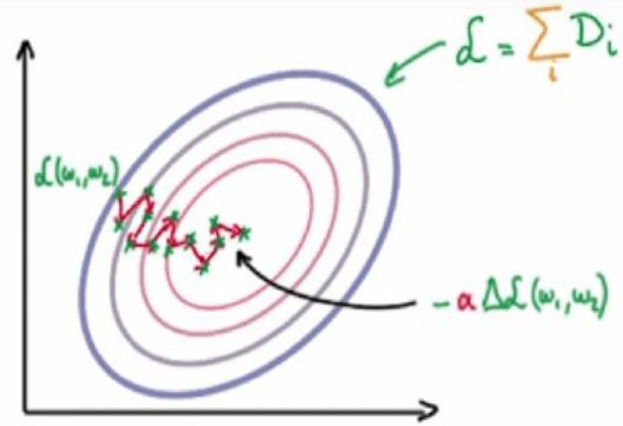
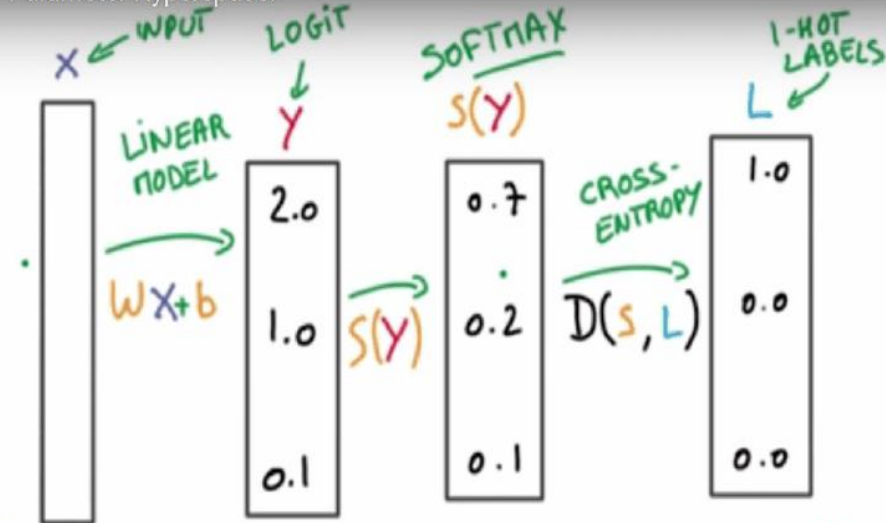
MANY HYPER-PARAMETERS

- INITIAL LEARNING RATE
- LEARNING RATE DECAY
- MOMENTUM
- BATCH SIZE
- WEIGHT INITIALIZATION



Input – linear - output

Parameter Hyperspace!



'SHALLOW' MODEL

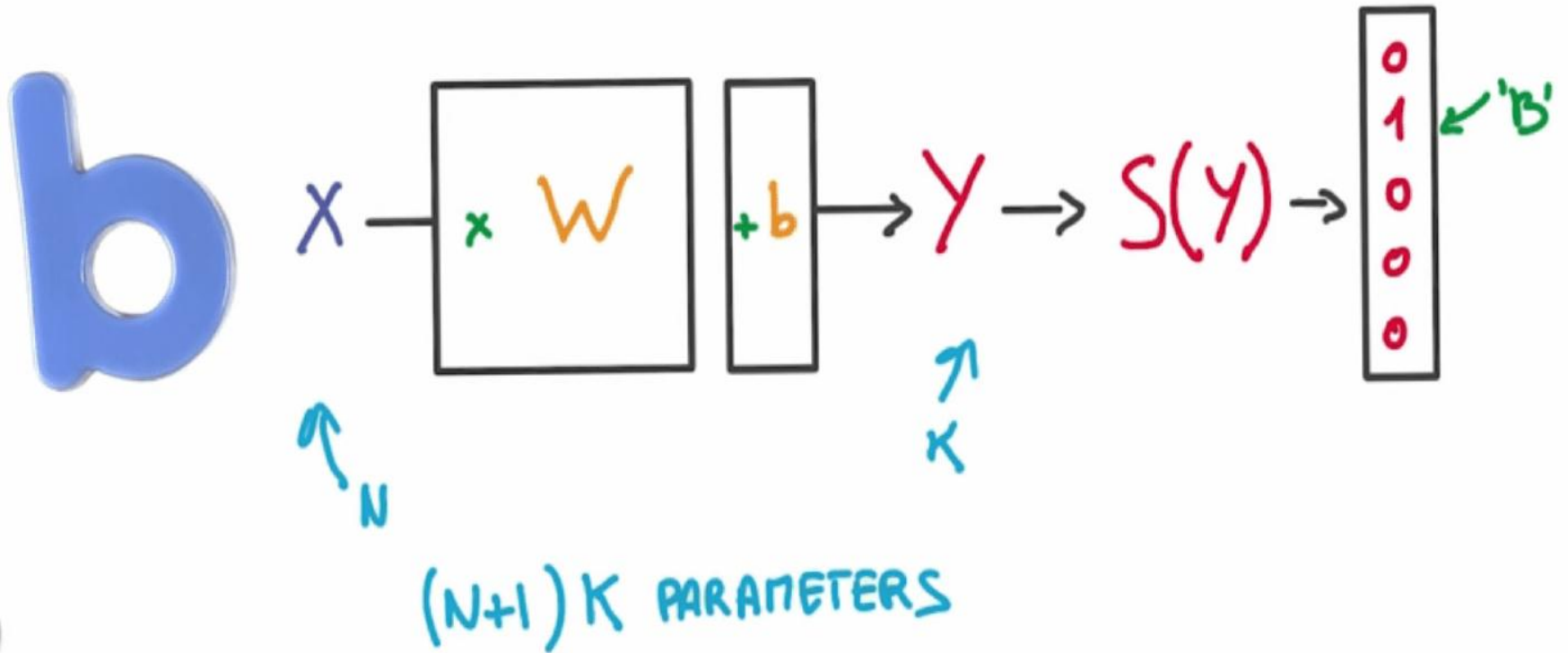
INPUT \rightarrow LINEAR \rightarrow OUTPUT

NO DEEP LEARNING YET!

Linear models are linear

Linear Models are Limited

LINEAR MODEL COMPLEXITY



Linear models are stable

$$Y = W X \rightarrow \Delta Y \sim |W| \Delta X$$

ΔY is SMALL, $|W|$ is BOUNDED, ΔX is SMALL

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

W^T is CONSTANTS

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

Linear models are here to stay

- **This is still linear**

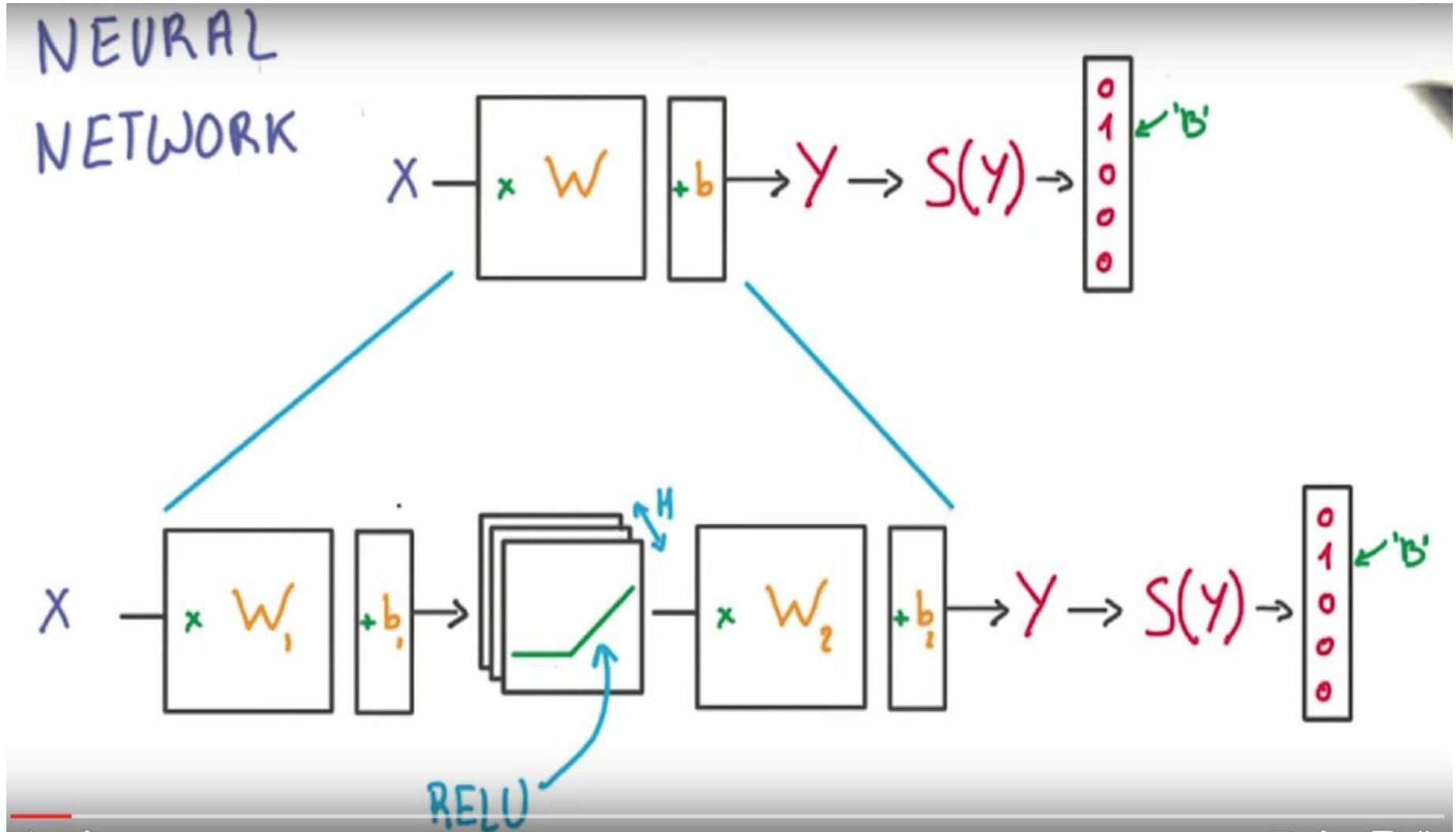
$$Y = \omega_1 \omega_2 \omega_3 X = W X$$

- **Lets introduce non-linearity**

$$Y = \omega_1 \omega_2 \omega_3 X = W X$$

NON - LINEARITIES

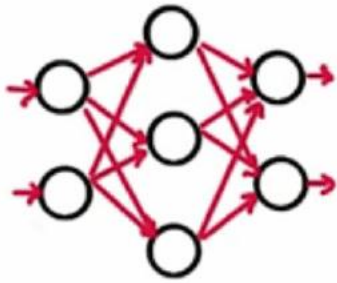
RELU: Rectified Linear Unit



Networks of ReLU

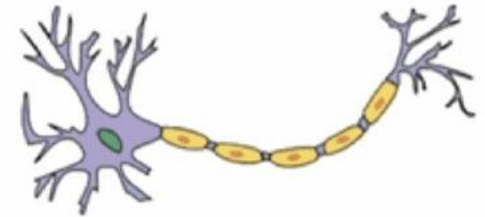
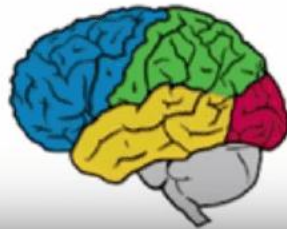
Network of ReLUs

NEURAL NETWORK



NEURONS?

HOW THE
BRAIN WORKS?



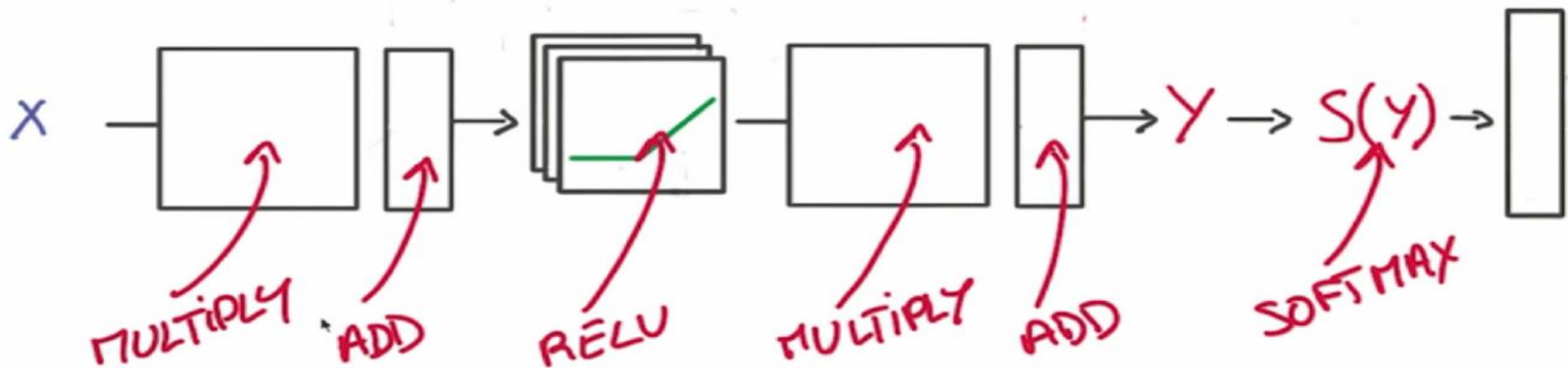
NEUROMORPHIC
ENGINEERING?

The Chain Rule

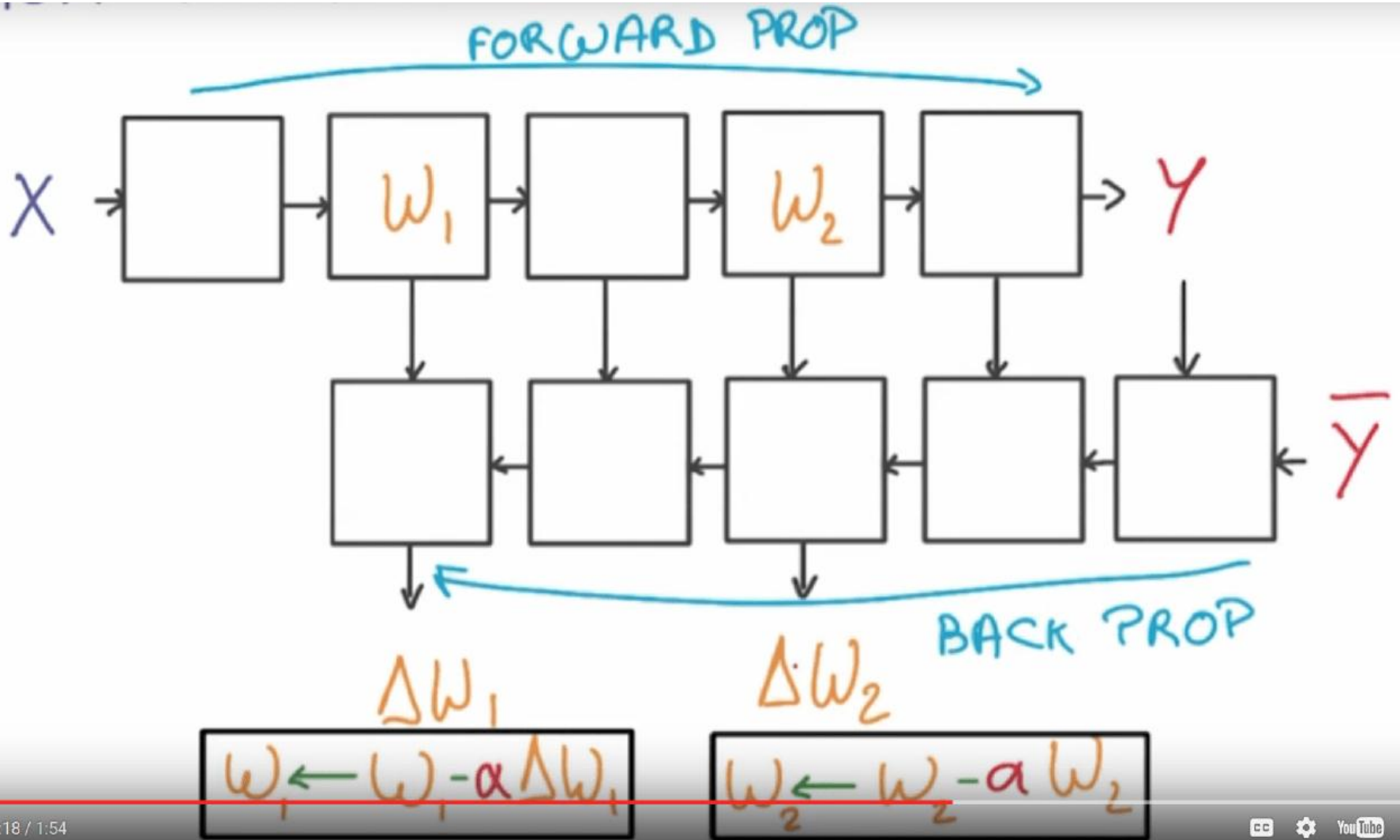
$$[g(f(x))]' = g'(f(x)) \cdot f'(x)$$

DERIVATIVE PRODUCT

STACKING UP SIMPLE OPERATIONS



Back - propagation



Optimisation tricks

REGULARIZATION



L_2 REGULARIZATION

NEW LOSS

$$\mathcal{L}' = \mathcal{L} + \beta \frac{1}{2} \|W\|_2^2$$

LOSS

A graph showing a green parabola representing the loss function. The horizontal axis is labeled W . The vertical axis is labeled $\frac{1}{2} \|W\|_2^2$. A red arrow points to the minimum of the parabola at $W=0$.

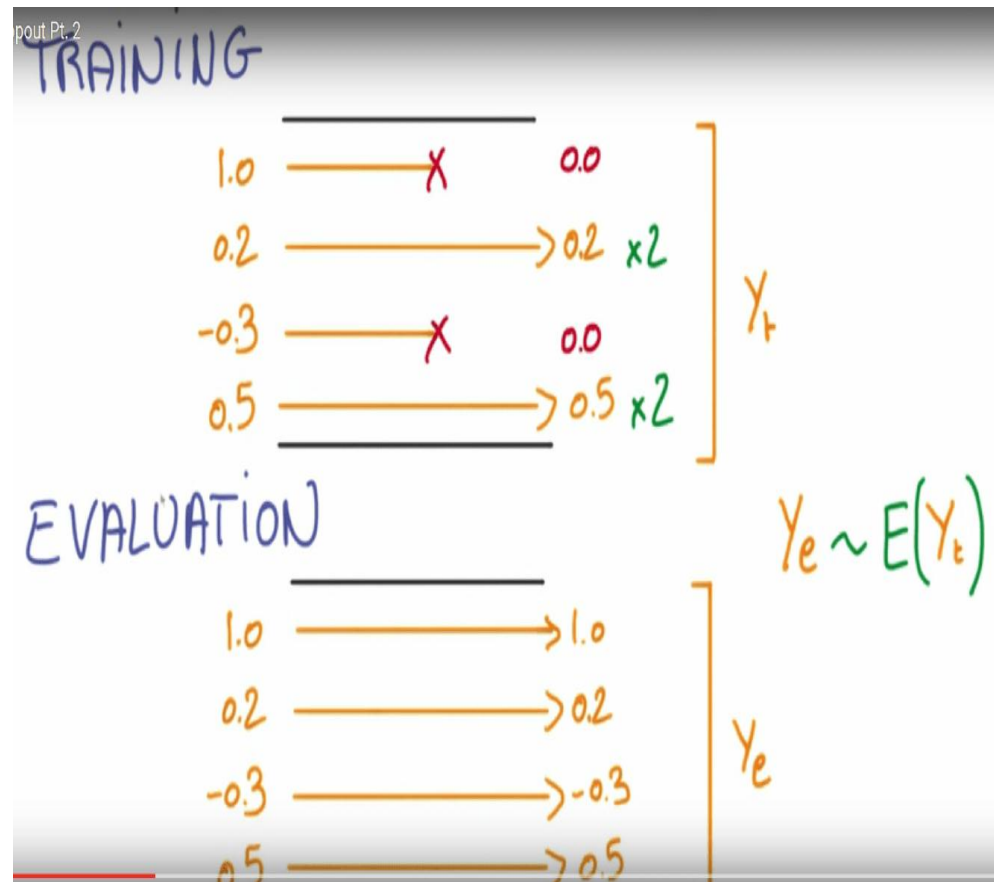
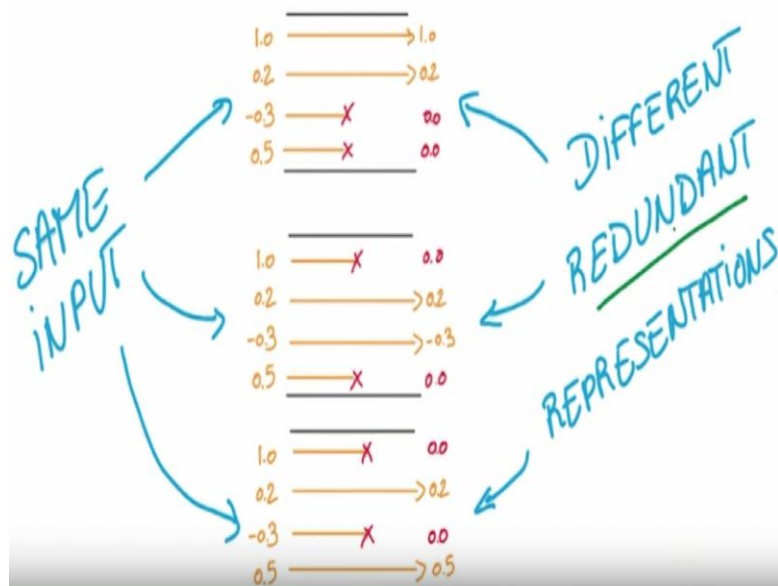
EARLY TERMINATION



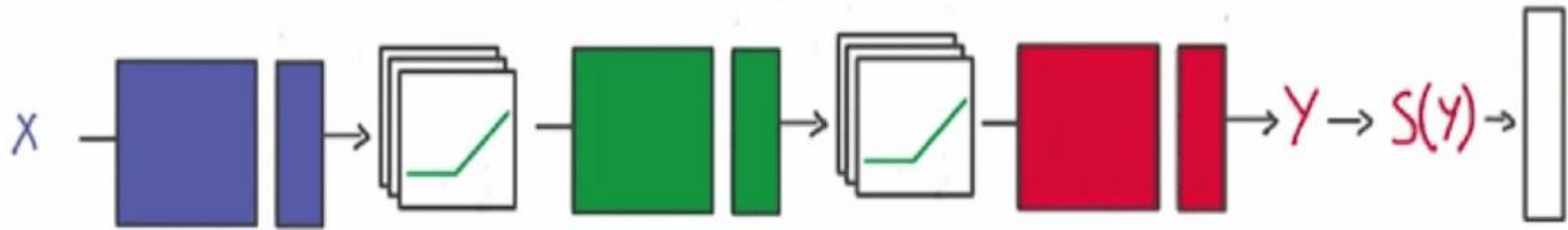
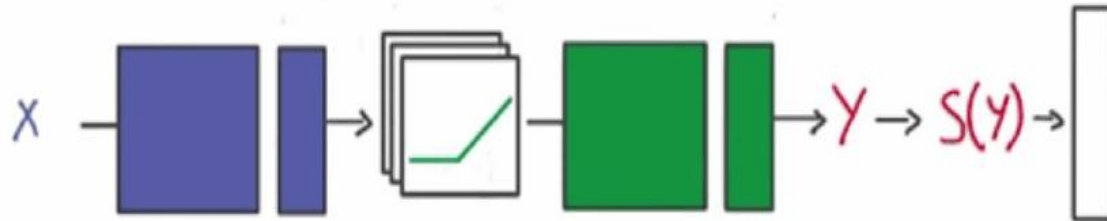
ACTIVATIONS

1.0	→	X	→	0.0
0.2	→		→	0.2
-0.3	→	X	→	0.0
0.5	→		→	0.5

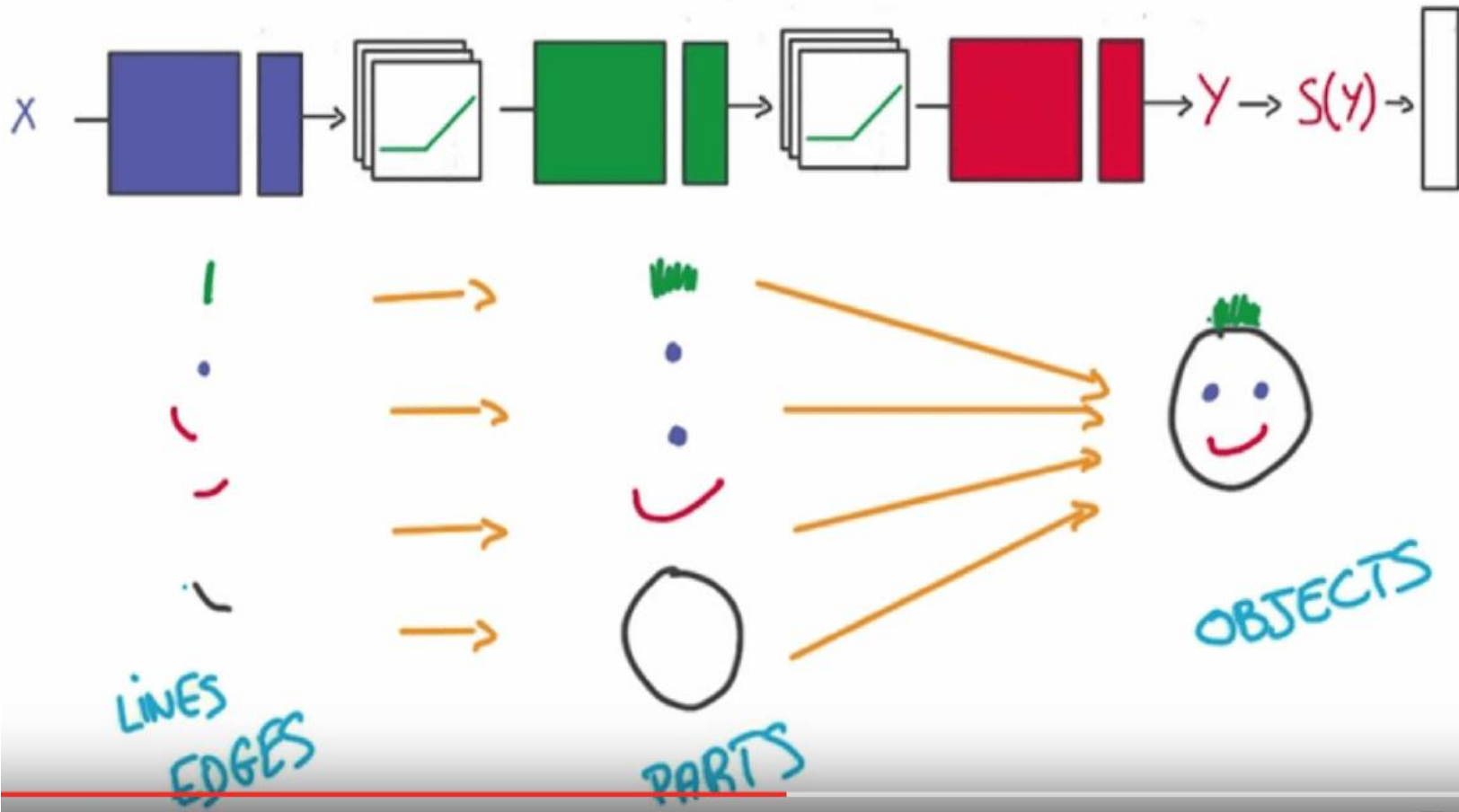
Optimisation trick: dropout



Deep networks



Deep networks



tensorflow.org/paper/whitepaper2015.pdf

TensorFlow:

Large-Scale Machine Learning on Heterogeneous Distributed Systems

(Preliminary White Paper, November 9, 2015)

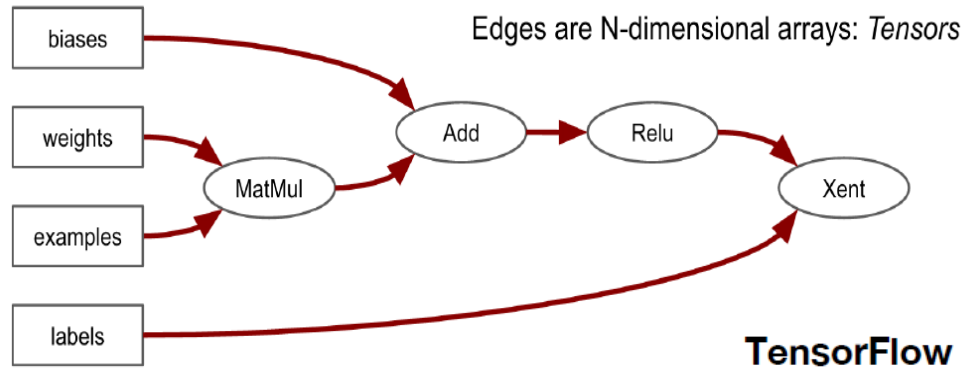
Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, 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 Zhen^σ
Google Research*

Abstract

TensorFlow [1] is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. A computation expressed using TensorFlow can be executed with little or no change on a wide variety of heterogeneous 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 learning systems into production across more than a dozen areas of computer science and other fields, including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery. This paper describes the TensorFlow 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.



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

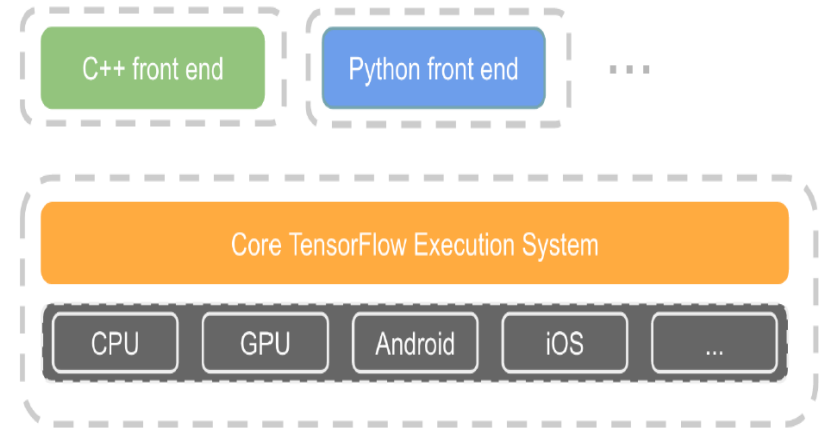


Parallel Execution

- 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

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

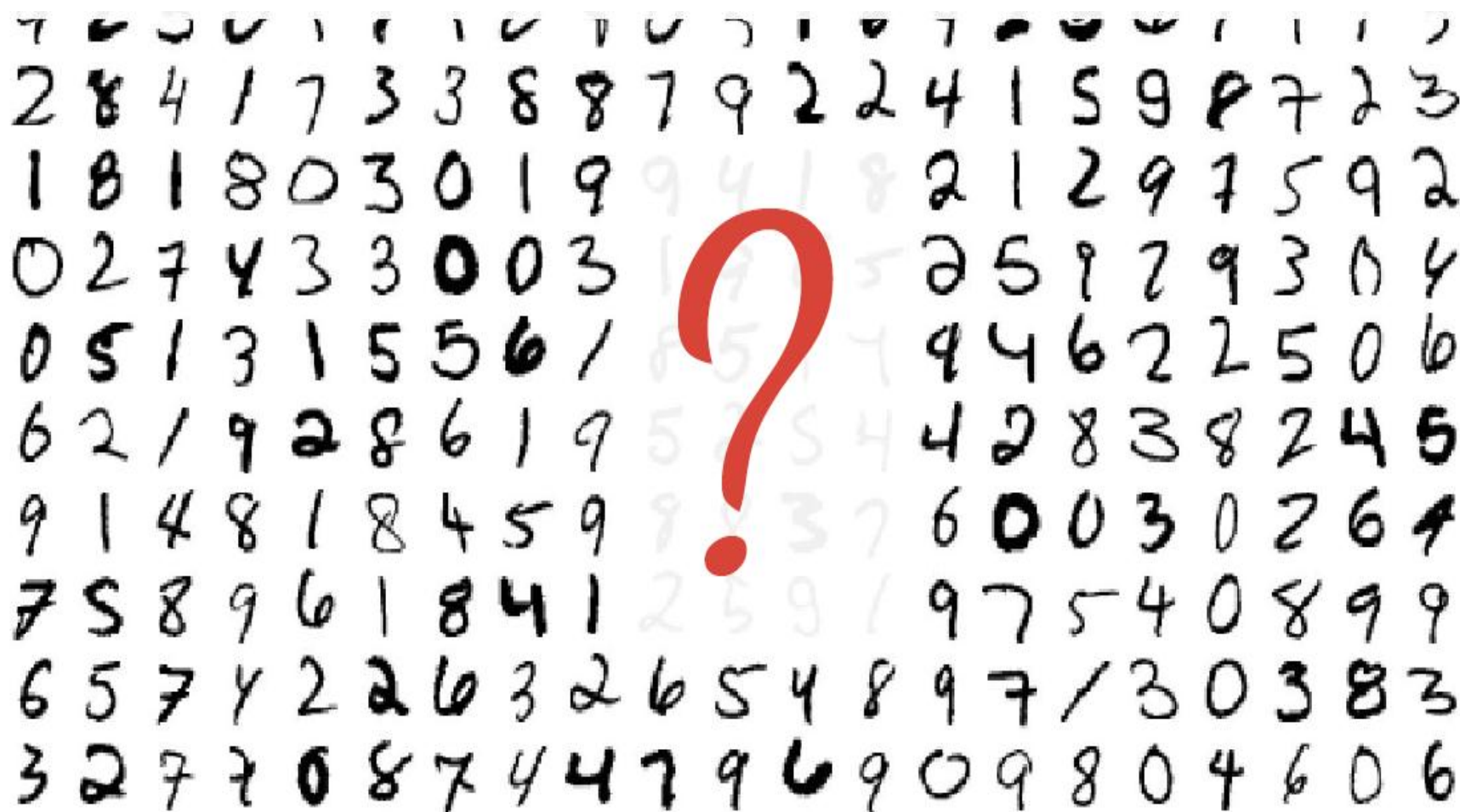
- Basics: constant, random, placeholder, cast, shape
- Variables: assign, assign_sub, assign_add
- Queues: enqueue, enqueue_batch, dequeue, blocking or not.
- Logical: equal, greater, less, where, min, max, argmin, argmax.
- Tensor computations: all math ops, matmul, determinant, inverse, cholesky.
-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



And many neural net specific Ops

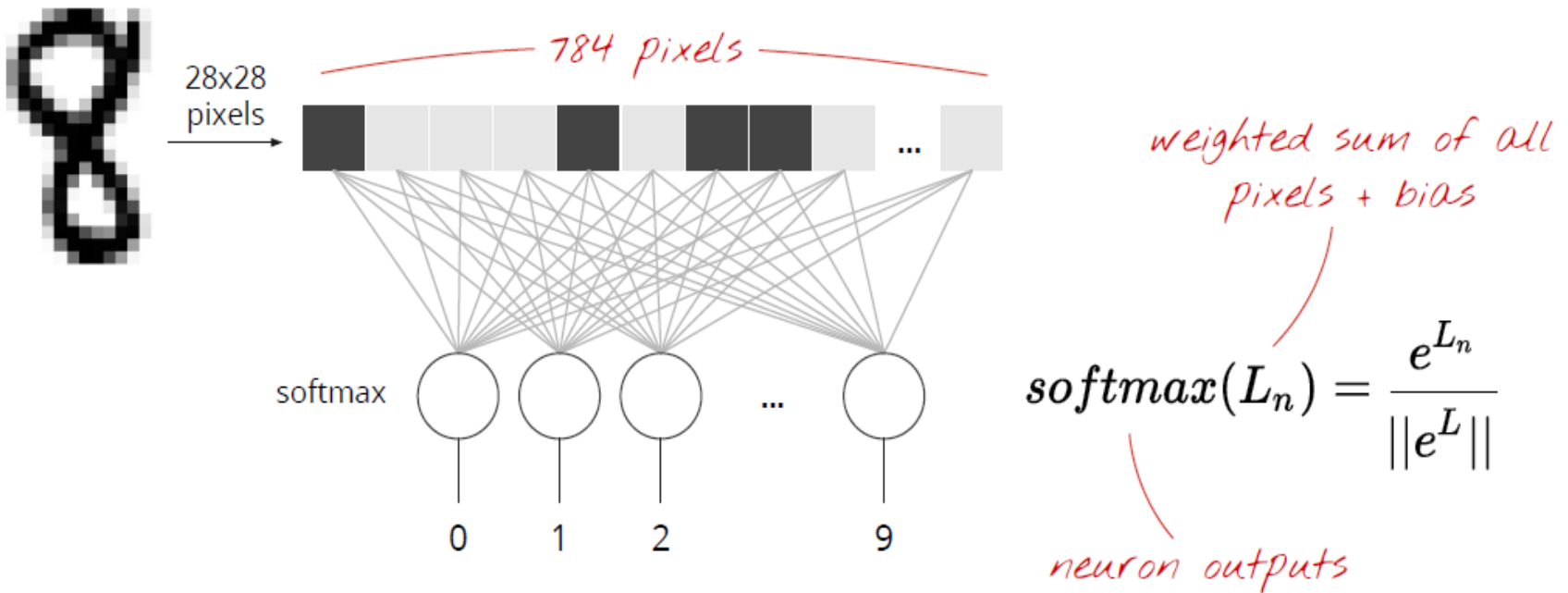
- Activations: sigmoid, tanh, relu, dropout, ...
- Pooling: avg, max.
- Convolutions: with many options.
- Normalization: local, batch, moving averages.
- Classification: softmax, softmax loss, cross entropy loss, topk.
- Embeddings: lookups/gather, scatter/updates.
- Sampling: candidate sampler (various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

Hand-written digits: MNIST



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

Simple linear model



Slides from M. Gerner tutorial

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

TensorFlow full python code

```
import tensorflow as tf
```

```
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
```

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

```
# placeholder for correct answers
```

```
Y_ = tf.placeholder(tf.float32, [None, 10])
```

```
# 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))
```

```
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)
```

```
# 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)
```

initialisation

model

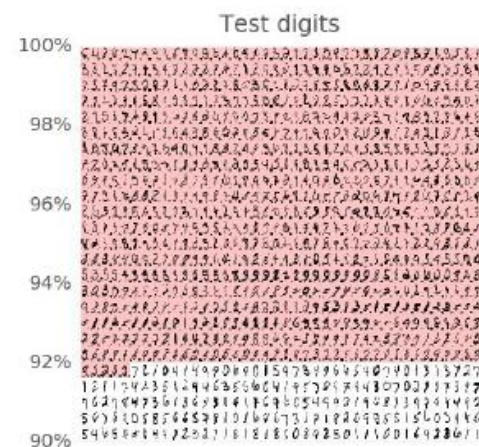
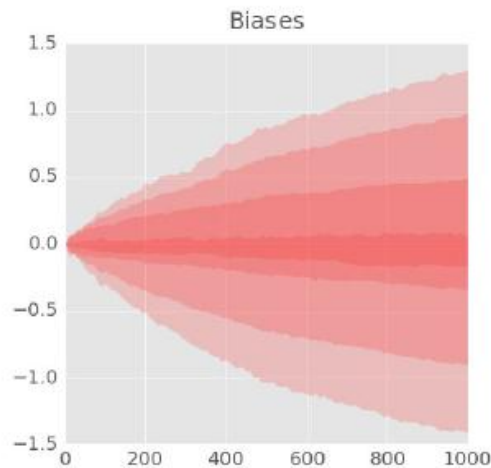
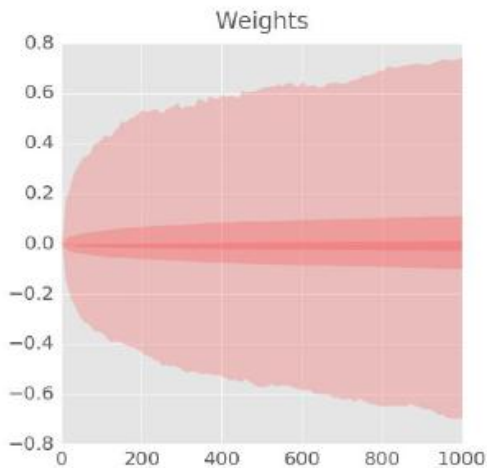
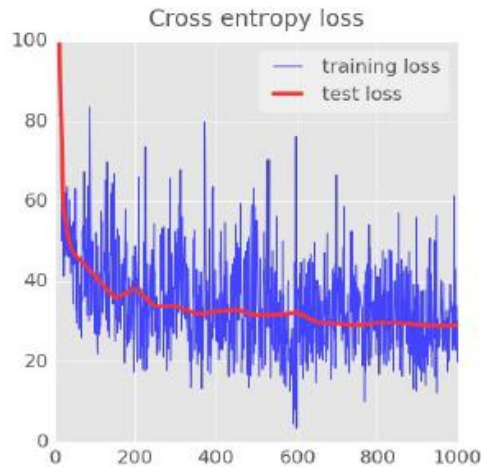
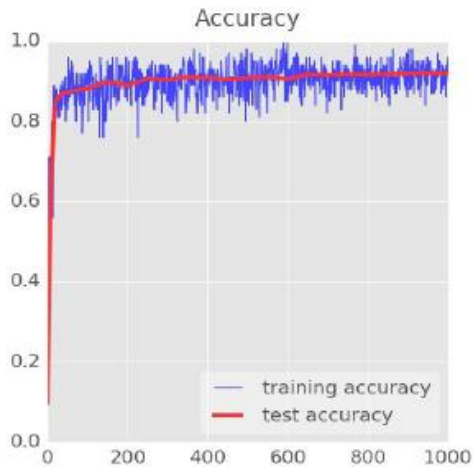
success metrics

training step

Run

Slides from M. Gorner@youtube

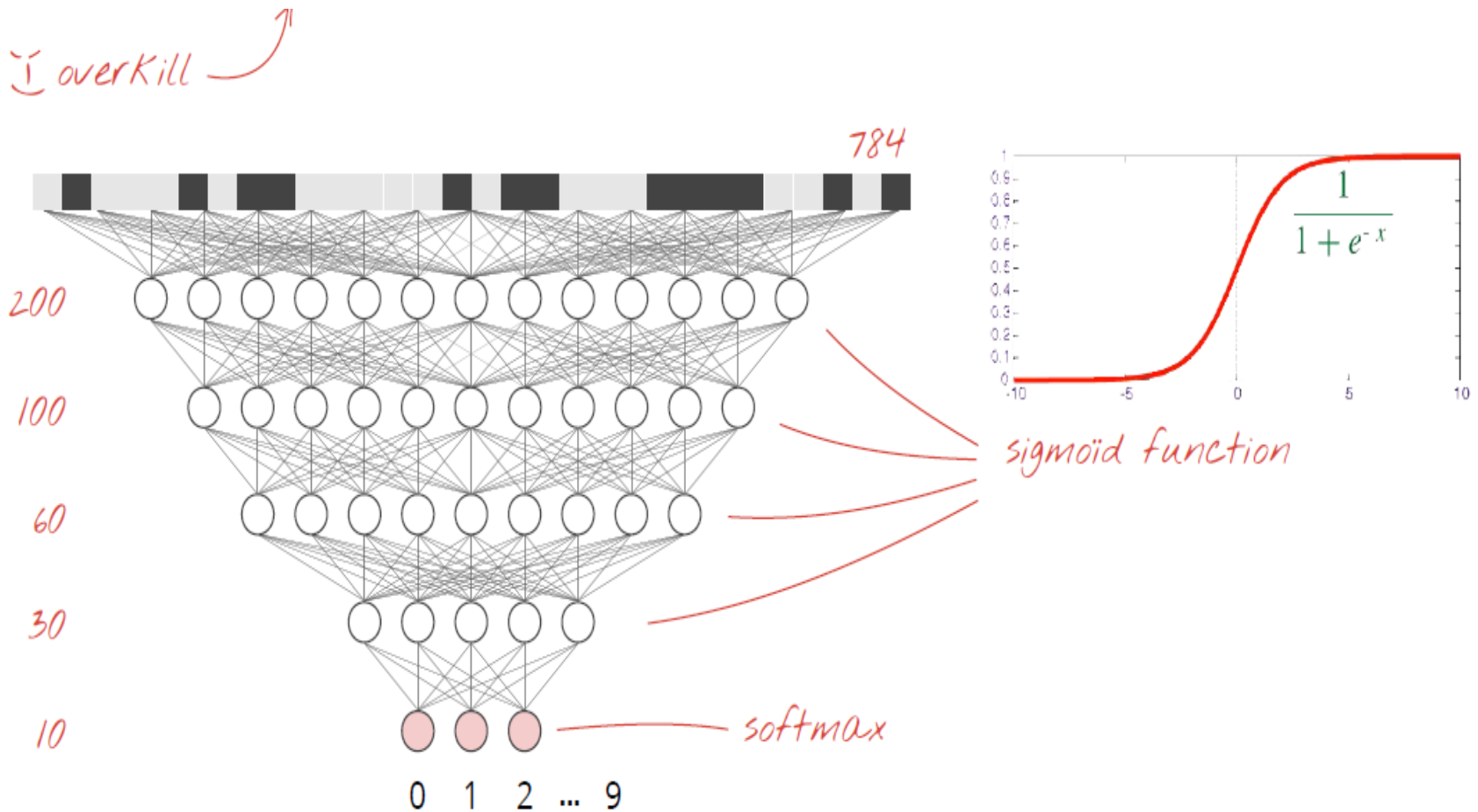
Simple linear model



92%

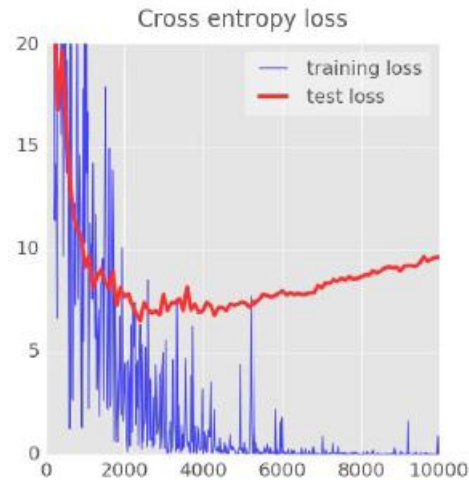
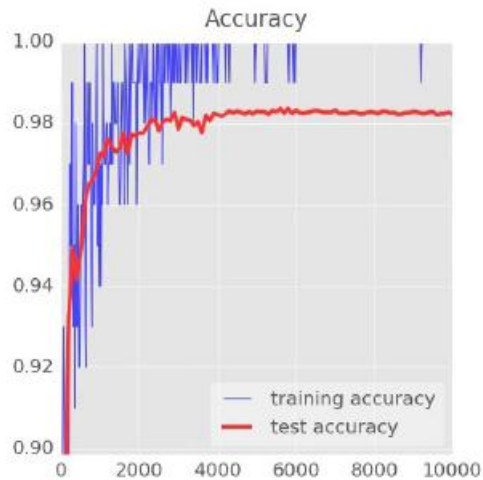
Slides from M. Gorner@youtube

Multi-layer connected network



Slides from M. Gorner@youtube

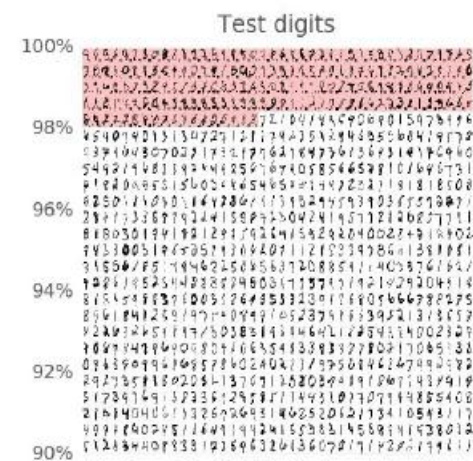
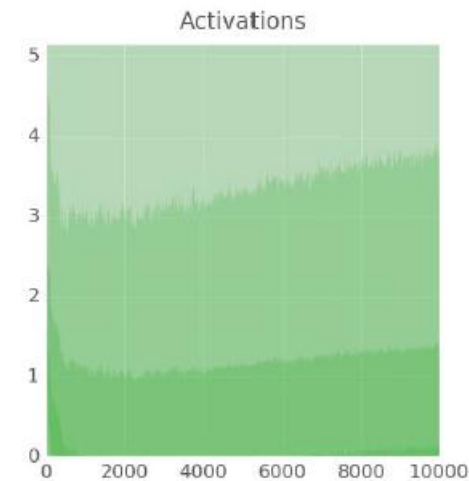
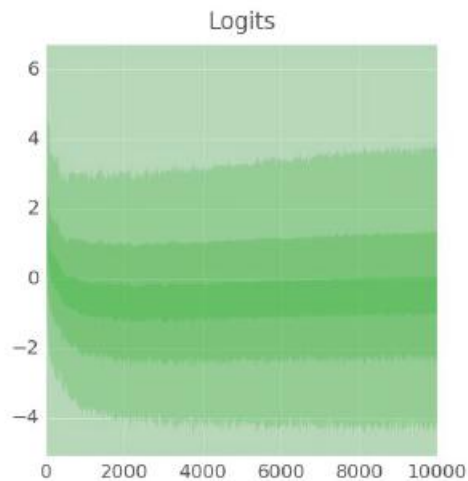
Multi-layer connected network



Training digits

```

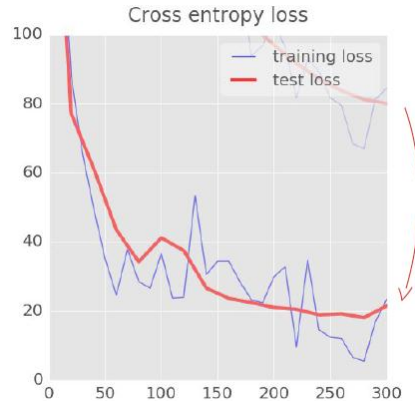
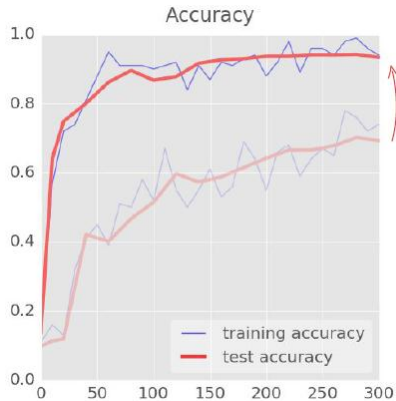
4 0 7 1 0 3 4 1 8 0
5 0 0 8 3 6 0 1 8 3
6 0 0 3 1 8 5 3 4 9
1 9 2 5 8 0 2 0 1 0
8 4 8 3 9 8 4 2 5 8
0 6 1 4 5 3 8 5 5 1
3 2 7 6 0 1 5 8 5 4
8 6 6 2 0 9 1 0 4 5
9 9 6 4 8 2 5 1 5 1
0 5 8 1 1 2 7 9 9 0
    
```



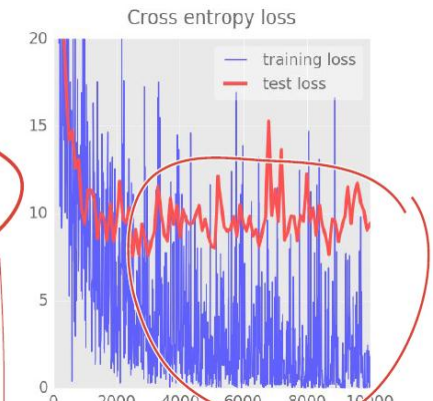
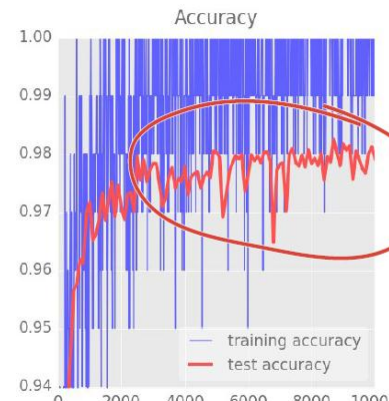
98%

All tricks count

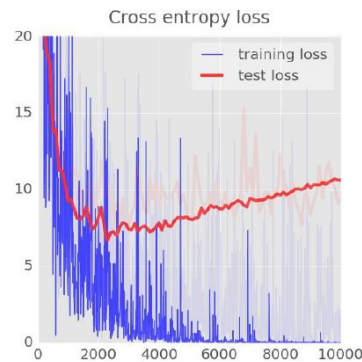
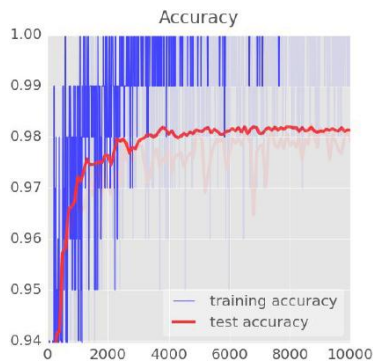
Use RELU



But noisy accuracy

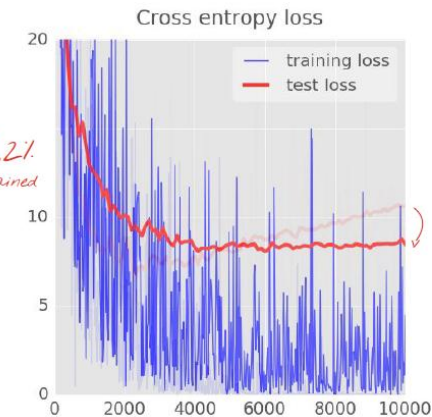
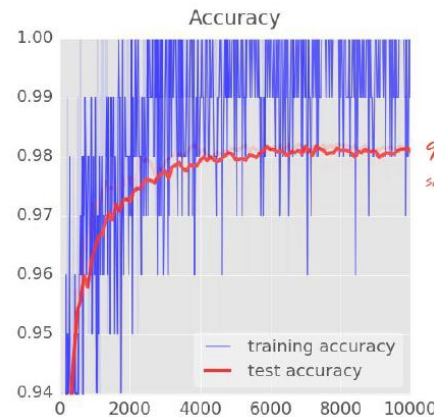


Exponentially reduce learning rates



Learning rate 0.003 at start then dropping exponentially to 0.0001

Add drop-out



RELU, decaying learning rate 0.003 \rightarrow 0.0001 and dropout 0.75

Slides from M. Gorner@youtube

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>