https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf

Introduction to Machine Learning

Marco Cristani

These slides were assembled by Marco Cristani, with grateful acknowledgement of the many others who made their course materials freely available online. Feel free to reuse or adapt these slides for your own academic purposes, provided that you include proper attribution.

School-leaving (scientific) 1995

Bachelor+Master@ UNIVR 1995-2002

Erasmus for the MD thesis in Lisboa 2001-02

PhD@ UNIVR 2003-2006

Los Angeles 2004-05

Researcher in CS@ UNIVR 2007-2014

Team Leader @ IIT Genova

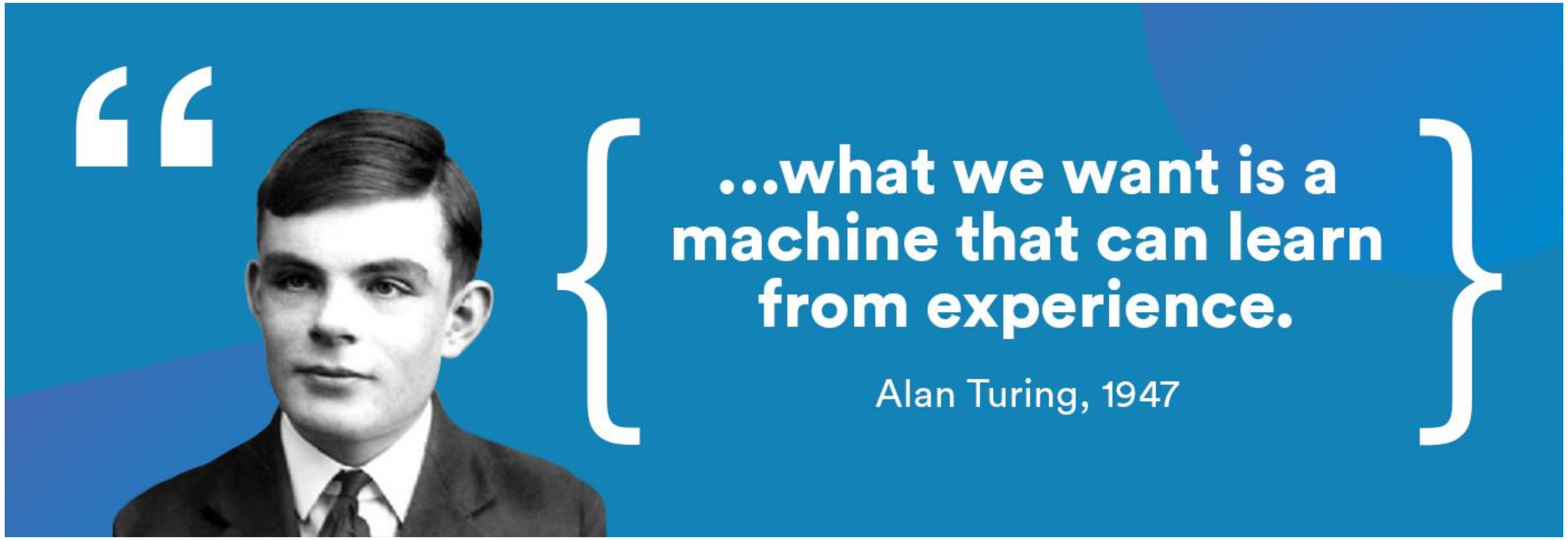
Associate Professor in CS @ UNIVR 2014-

Associate member National Research Council of Italy

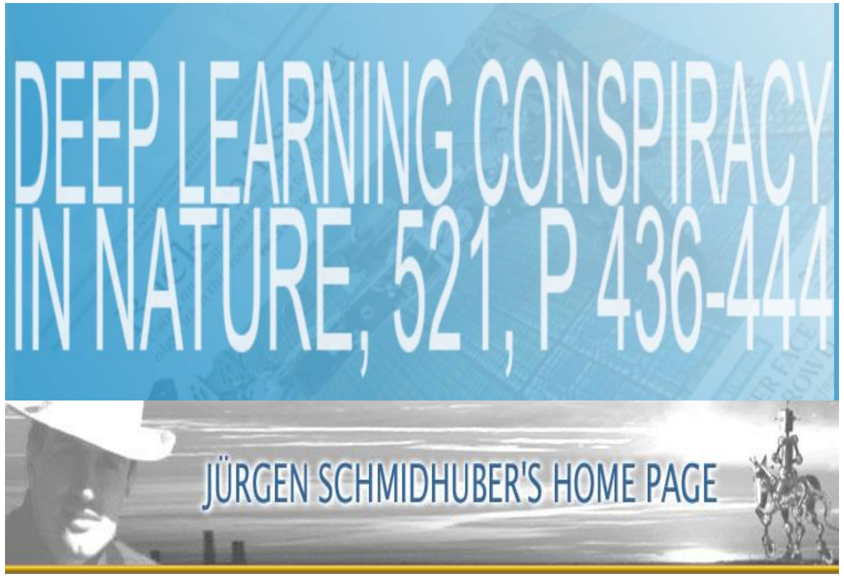
Director of the *Computer Science Park* 2019-

Founder and CRO HUMATICS 2016

CTO MakarenaLabs 2018



…what we want is a machine that can learn from experience.
Alan Turing, 1947



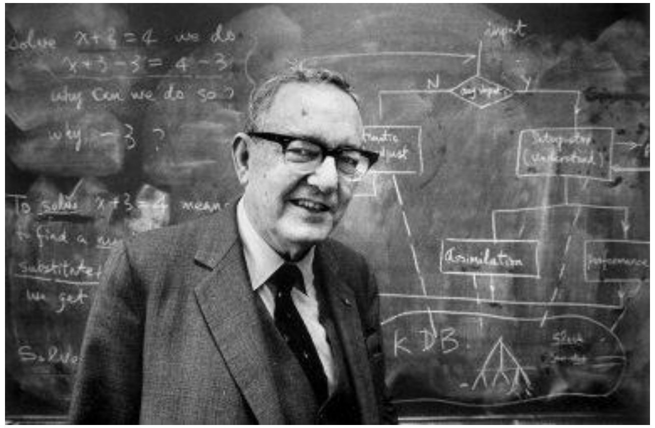
DEEP LEARNING CONSPIRACY IN NATURE, 521, P 436-444
JÜRGEN SCHMIDHUBER'S HOME PAGE

ACM A.M. TURING AWARD
Yoshua Bengio
ACM A.M. TURING AWARD
Geoffrey Hinton
ACM A.M. TURING AWARD
Yann LeCun

What is machine learning?

- “Learning is any process by which a system improves performance from experience”

  - Herbert Simon (*Turing Award* 1975, *Nobel Prize* in Economics 1978)



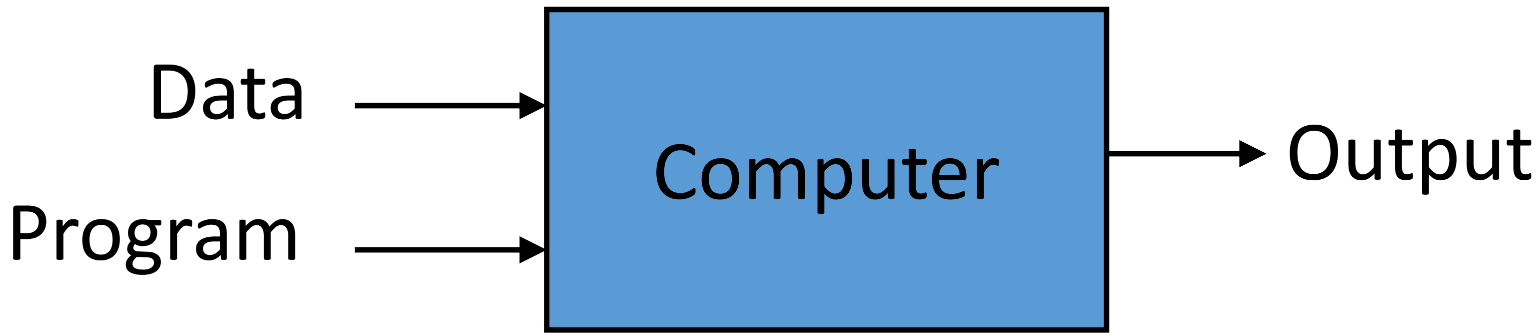
- Definition by Tom Mitchell (1998):

  Machine Learning is the study of algorithms that
  - improve their performance *P*
  - at some task *T*
  - with experience *E*

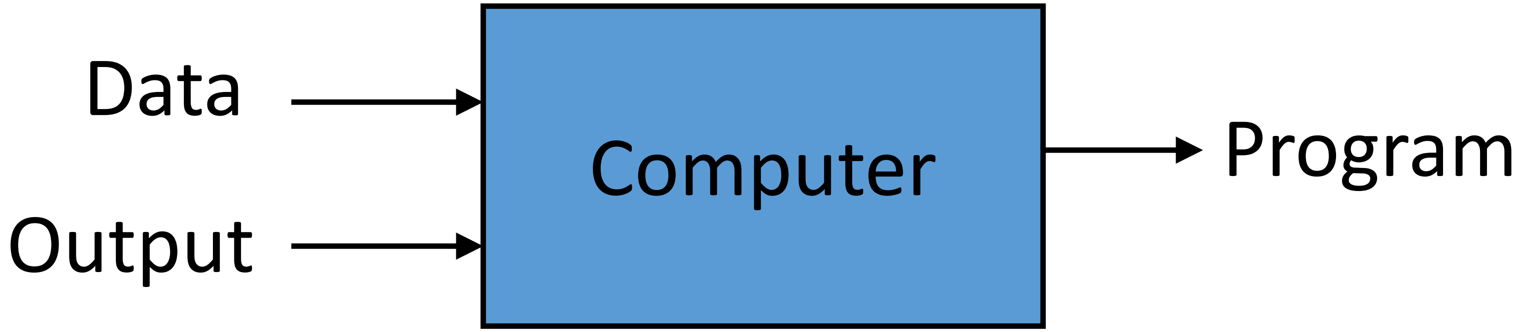
  A well-defined learning task is given by *<P, T, E>*

Difference w.r.t. traditional programming

Traditional Programming



Machine Learning



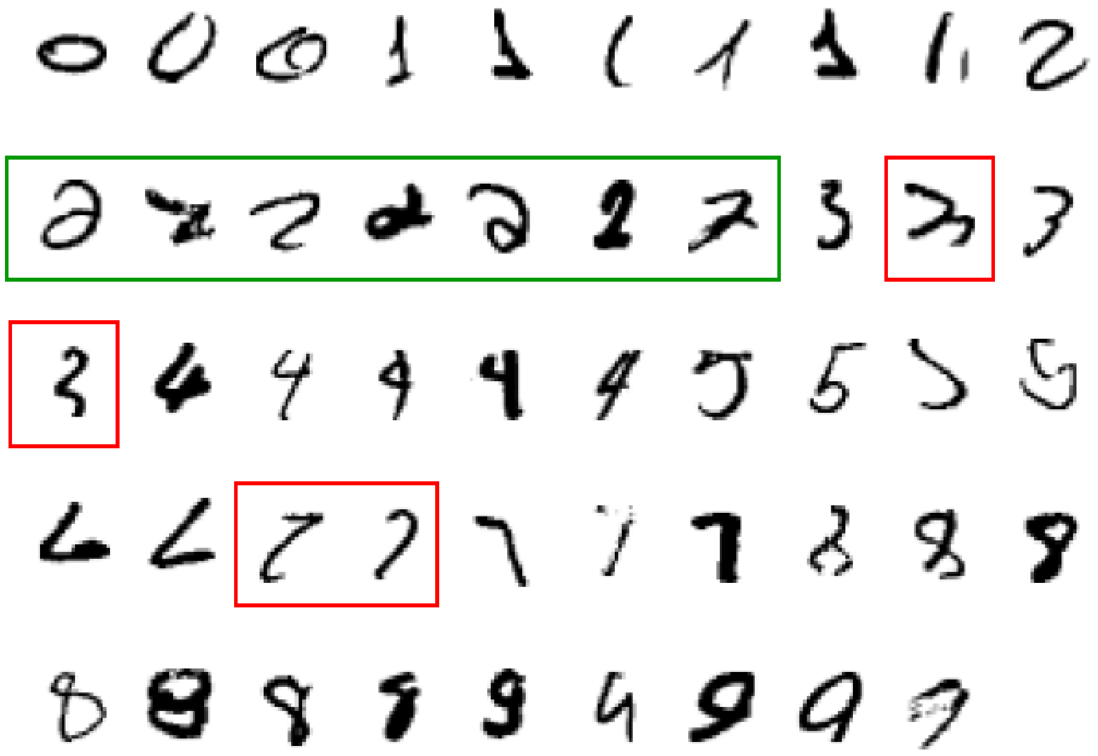
Magic?

No, more like gardening

- **Seeds** = Algorithms
- **Nutrients** = Data
- **Gardener** = You
- **Plants** = Programs

When Do We Use Machine Learning?

- ML is used when:
  - Human expertise does not exist (navigating on Mars, forecasting)
  - Humans can't explain their expertise (recognition)
  - Models must be customized (personalized medicine)
  - Models are based on huge amounts of data (genomics)



A case of handwritten recognition: what makes a «2»?

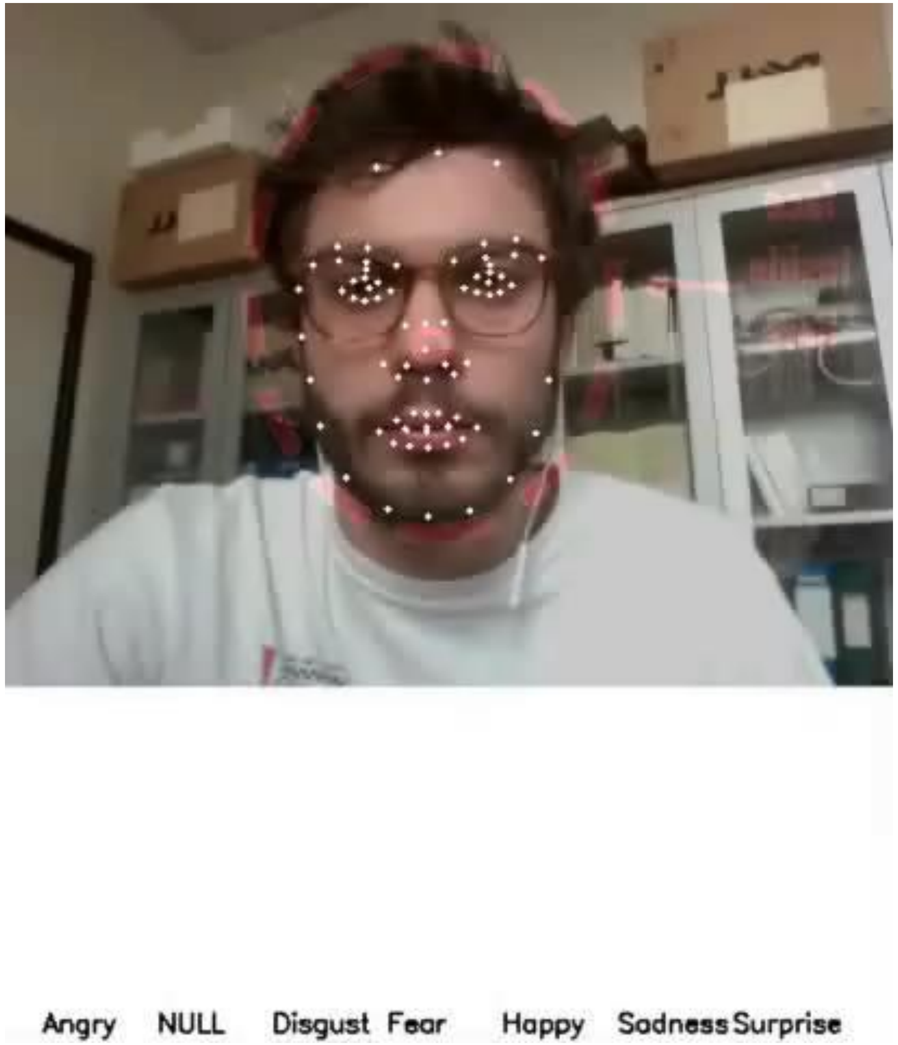
Some more examples of tasks that are best solved by using a learning algorithm

- Recognizing patterns:
  - Facial identities or facial expressions
  - Handwritten or spoken words
  - Medical images
- Generating patterns:
  - Generating images or motion sequences
- Recognizing anomalies:
  - Unusual credit card transactions
  - Unusual patterns of sensor readings in a nuclear power plant
- Forecasting:
  - Future stock prices or currency exchange rates

Some *bachelor* projects in machine learning

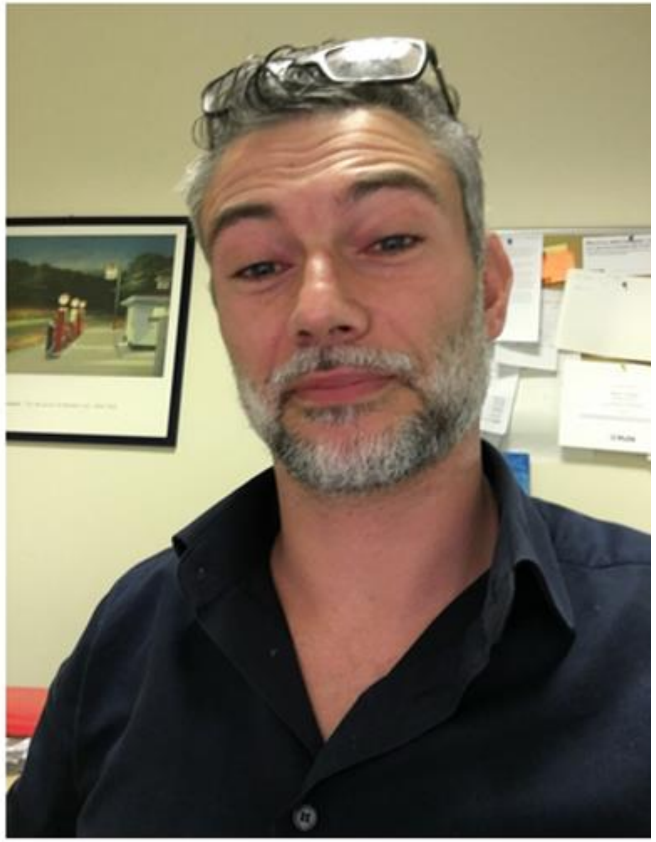
- Expression recognition
  - (Dr. *Luca Brunelli*, now in Statwolf, data science, https://www.statwolf.com/)

STATWOLF



Some *bachelor* projects in machine learning

- Scene recognition



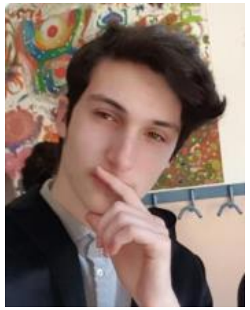
http://places.csail.mit.edu/m.html

Predictions:

- **Type of environment:** indoor
- **Semantic categories:** office:0.61, home_office:0.13,
- **SUN scene attributes:** enclosedarea, nohorizon, cloth, man-made, electricindoorlighting, matte, research, sterile

Some *bachelor* projects in machine learning

- creating 3D objects from 2D images

(Carlo Veronesi,
Nicholas Merci,
their ongoing
bachelor thesis)

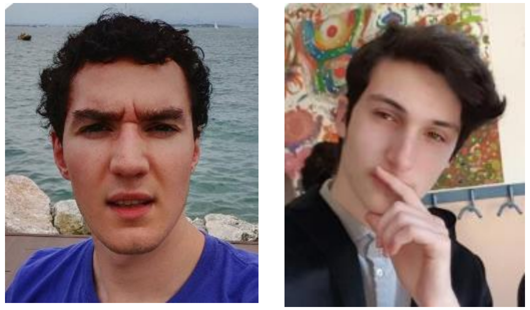
2D image


3D ***generated*** images

a particular of
the 3D body

Some *bachelor* projects in machine learning

- creating 3D objects from 2D images (2)



(Carlo Veronesi, Nicholas Merci, their ongoing bachelor thesis)

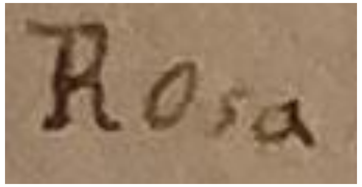
2D image

3D ***generated*** images

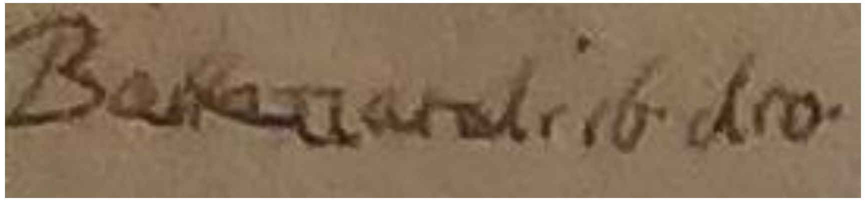
a particular of the 3D body

Some *bachelor* projects in machine learning

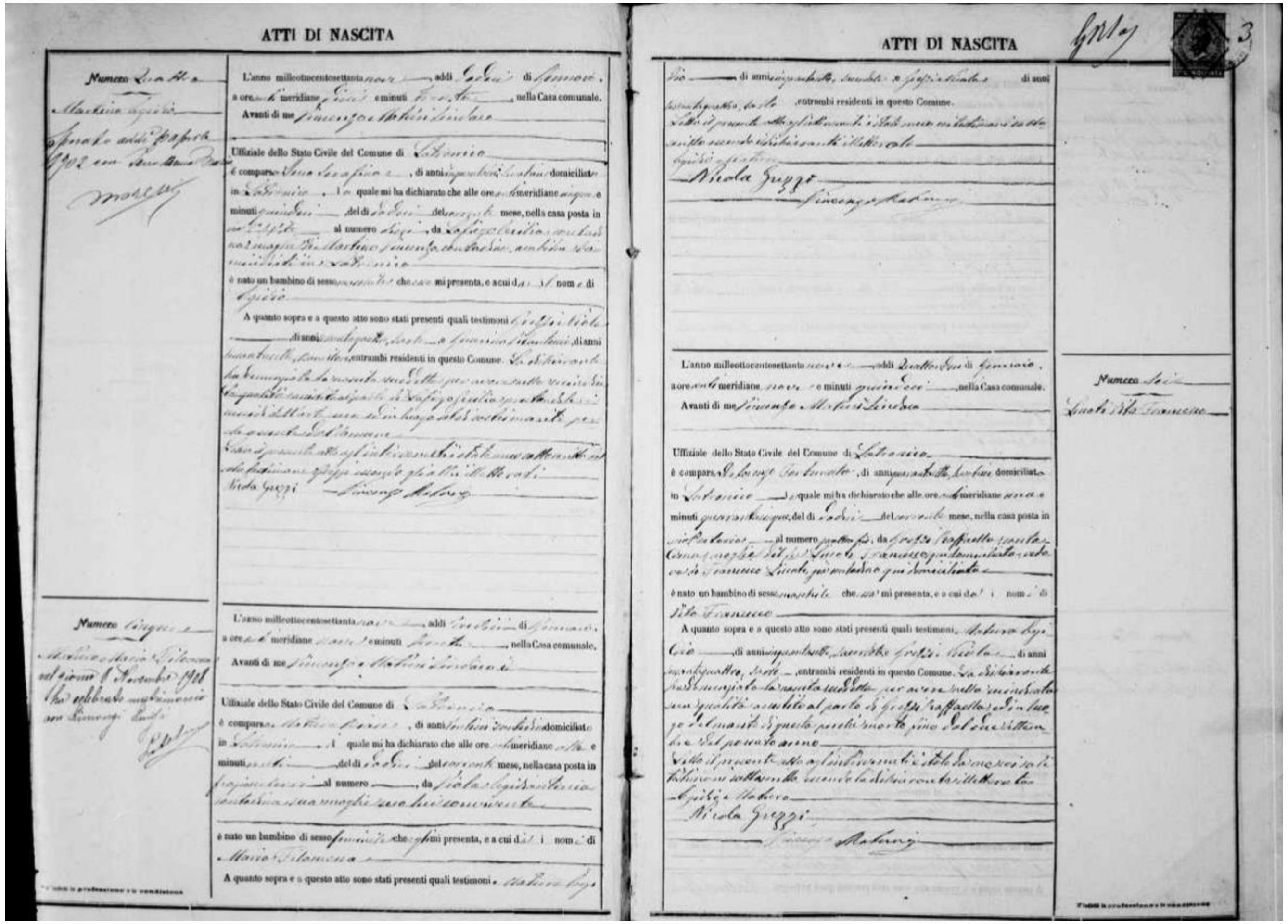
- Birth records digitalization towards family tree estimation



Rosa



Battezzardi ib tro (???)


ATTI DI NASCITA

ATTI DI NASCITA

Sample applications

- Web search
- Computational biology
- Finance
- E-commerce
- Space exploration
- Robotics
- Information extraction
- Social networks
- Debugging
- *[Your favorite area]*

Defining the learning task

Improve on task T, with respect to performance metric P, based on experience E

T: Playing checkers
P: Percentage of games won against an arbitrary opponent
E: Playing practice games against itself

T: Driving on four-lane highways using vision sensors
P: Average distance traveled before a human-judged error
E: A sequence of images and steering commands recorded while observing a human driver.

T: Recognizing hand-written words
P: Percentage of words correctly classified
E: Database of human-labeled images of handwritten words

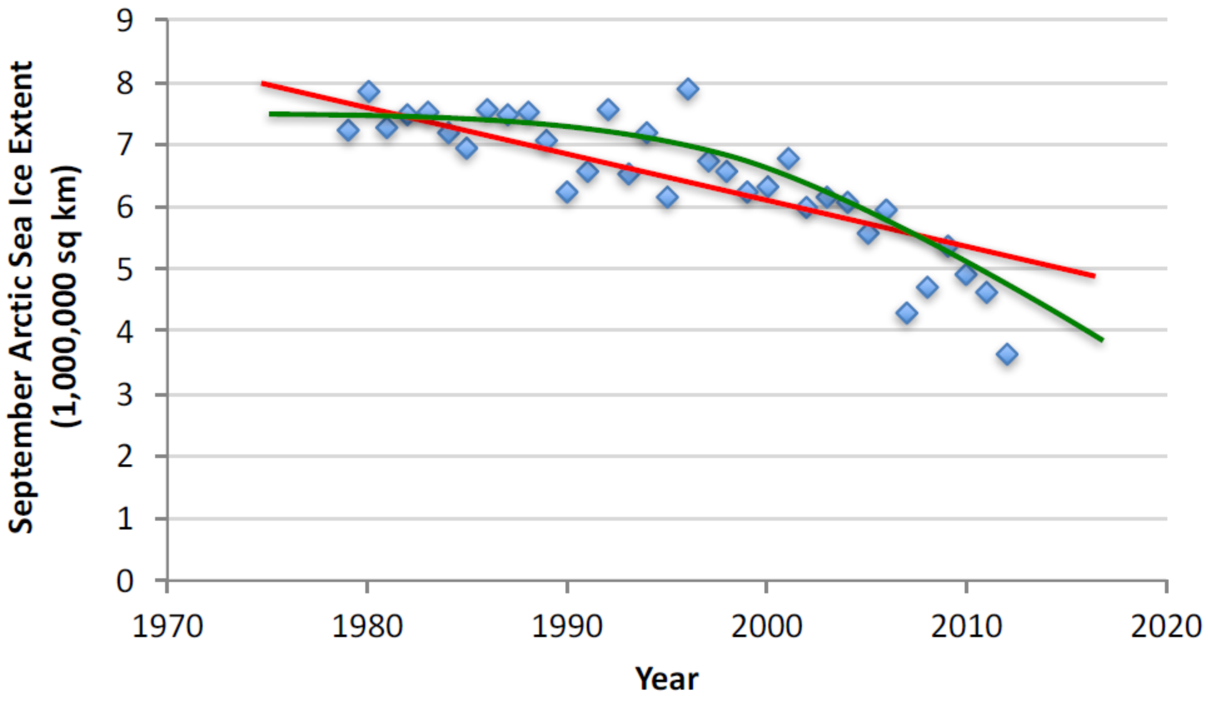
T: Categorize email messages as spam or legitimate.
P: Percentage of email messages correctly classified.
E: Database of emails, some with human-given labels

Types of learning

- **Supervised (inductive) learning (= *regression|sup.ed classification*)**
  - – Given: training data + desired outputs (labels)
- **Unsupervised learning**
  - – Given: training data (without desired outputs)
- **Semi-supervised learning**
  - – Given: training data + a few desired outputs
- **Reinforcement learning**
  - – Rewards from sequence of actions

Supervised Learning: Regression

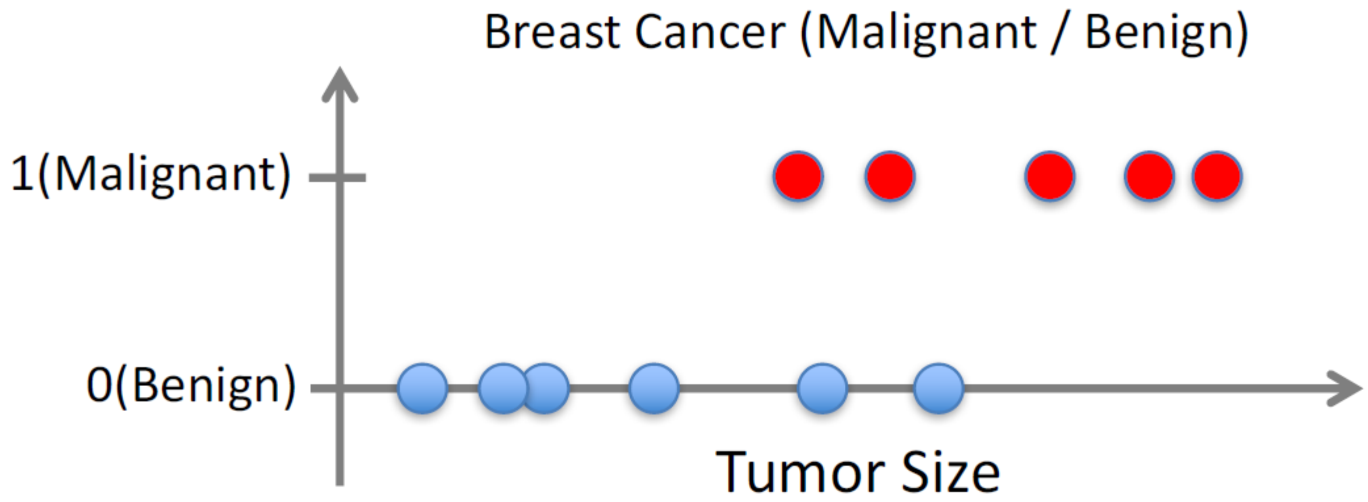
- Given (x1, y1), (x2, y2), ..., (xn, yn)
- Learn a function f(x) to predict y given x
- – y is real-valued == regression



Data from G. Witt. Journal of Statistics Education, Volume 21, Number 1 (2013)

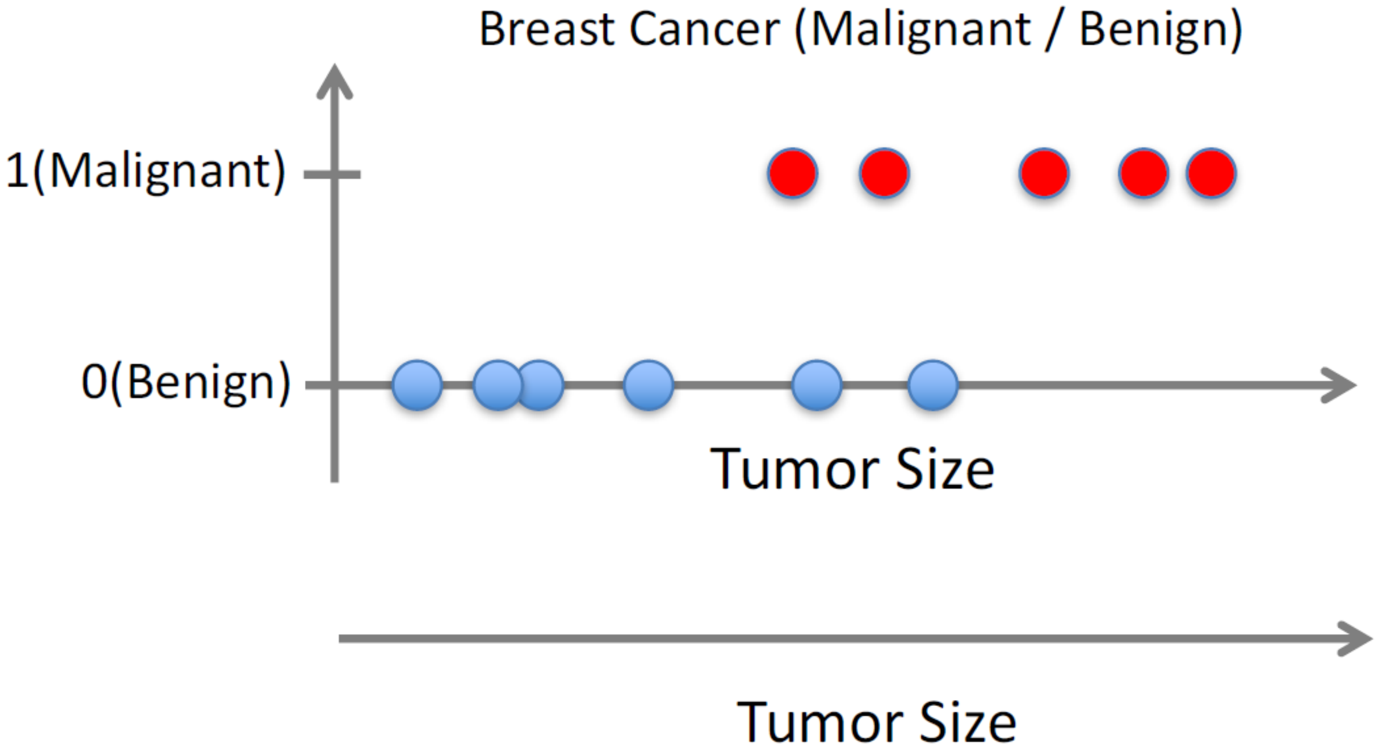
Supervised Learning: Classification

- Given (x1, y1), (x2, y2), ..., (xn, yn)
- Learn a function f(x) to predict y given x
- – y is categorical == (supervised) classification



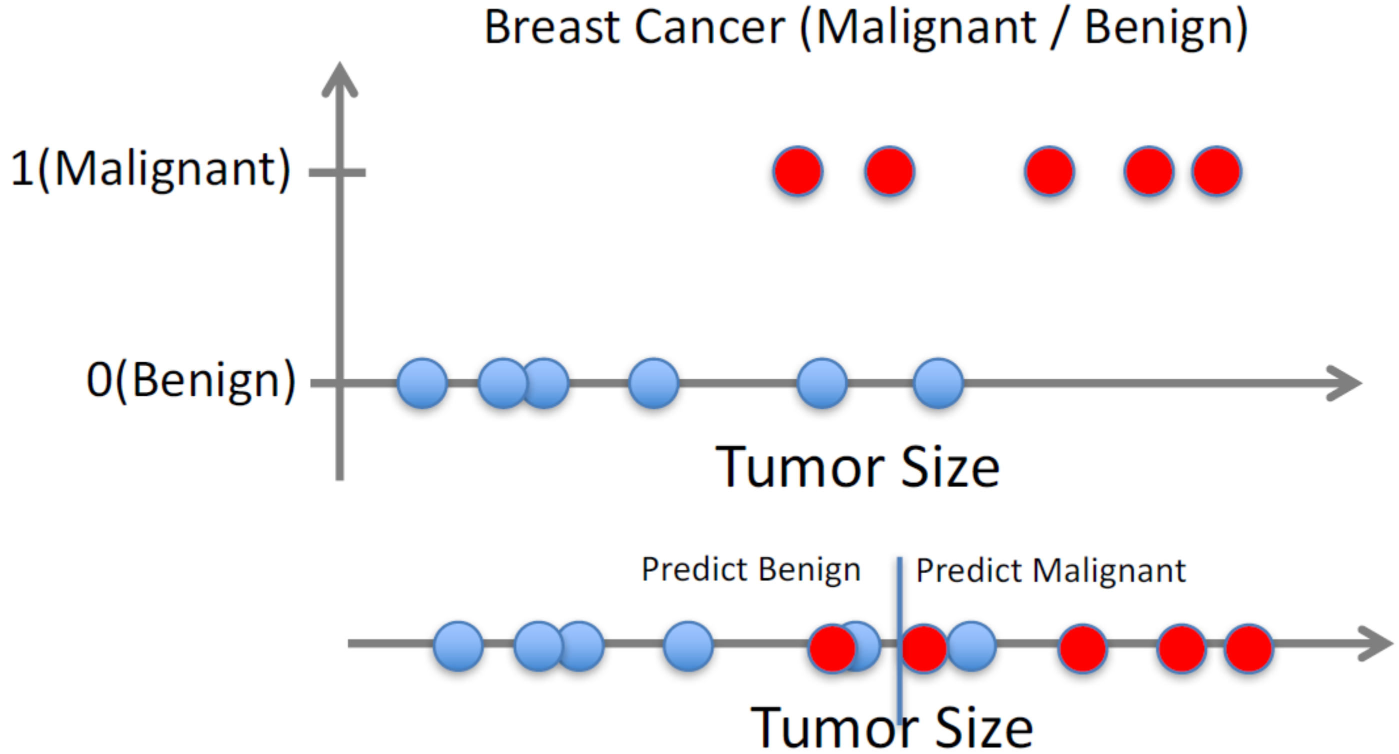
Supervised Learning: Classification

- Given (x1, y1), (x2, y2), ..., (xn, yn)
- Learn a function f(x) to predict y given x
- – y is categorical == (supervised) classification



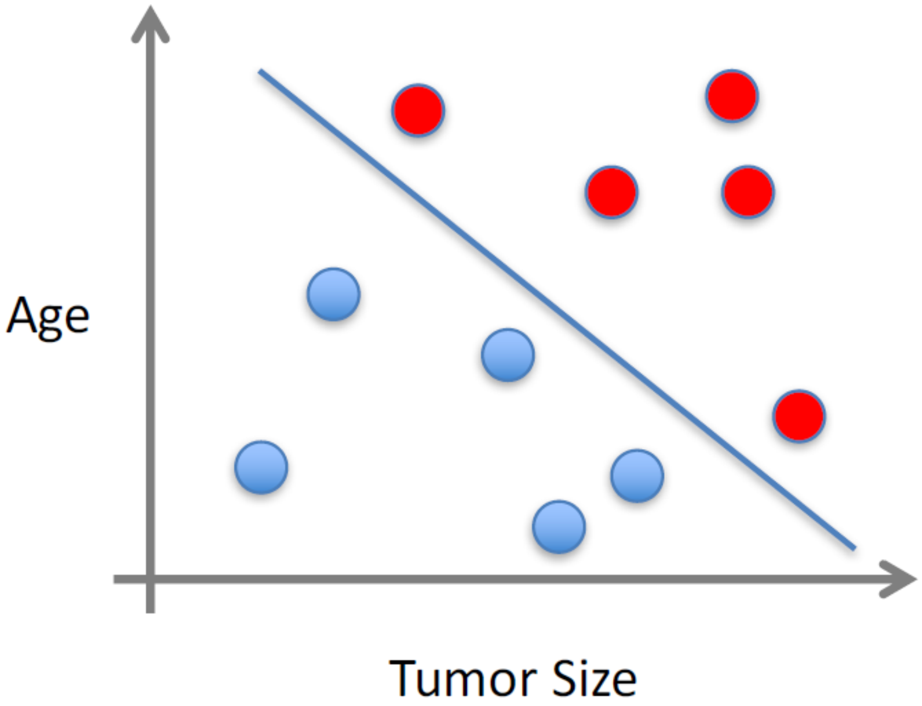
Supervised Learning: Classification

- Given (x1, y1), (x2, y2), ..., (xn, yn)
- Learn a function f(x) to predict y given x
- – y is categorical == (supervised) classification



Supervised Learning

- x can be multi-dimensional
  - Each dimension corresponds to an attribute



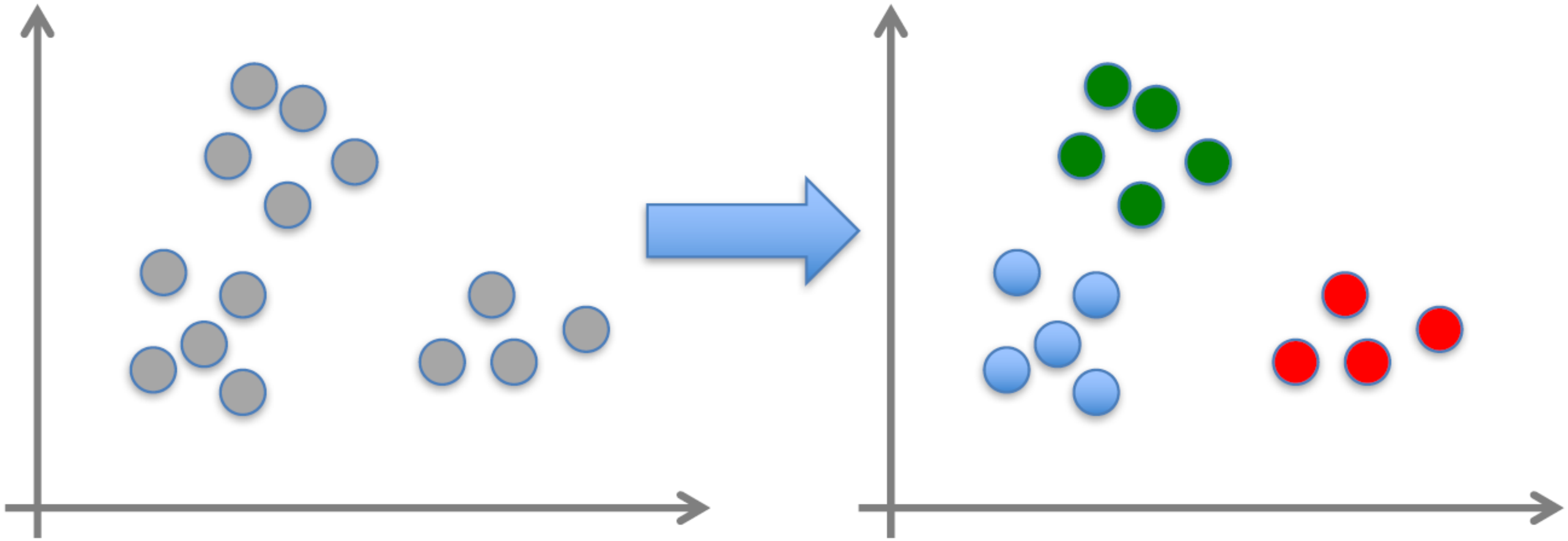
- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape

…

Unsupervised Learning

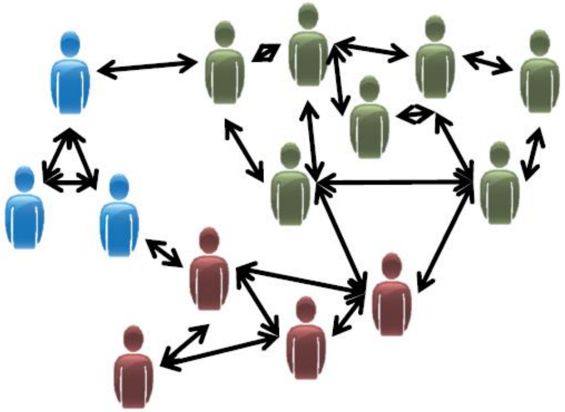
- Given x1, x2, ..., xn (without labels)
- Output hidden structure behind the x's

– e.g., clustering



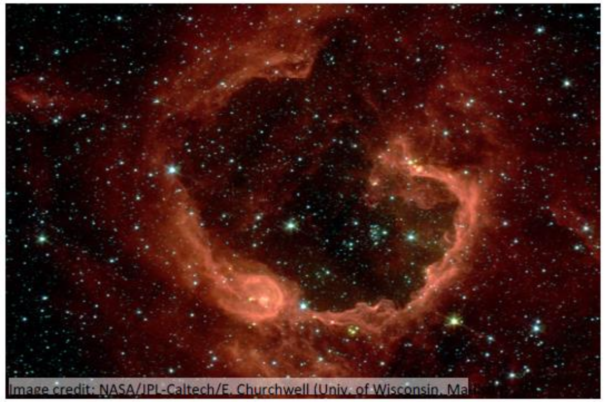
Unsupervised Learning

Organize computing clusters



Social network analysis

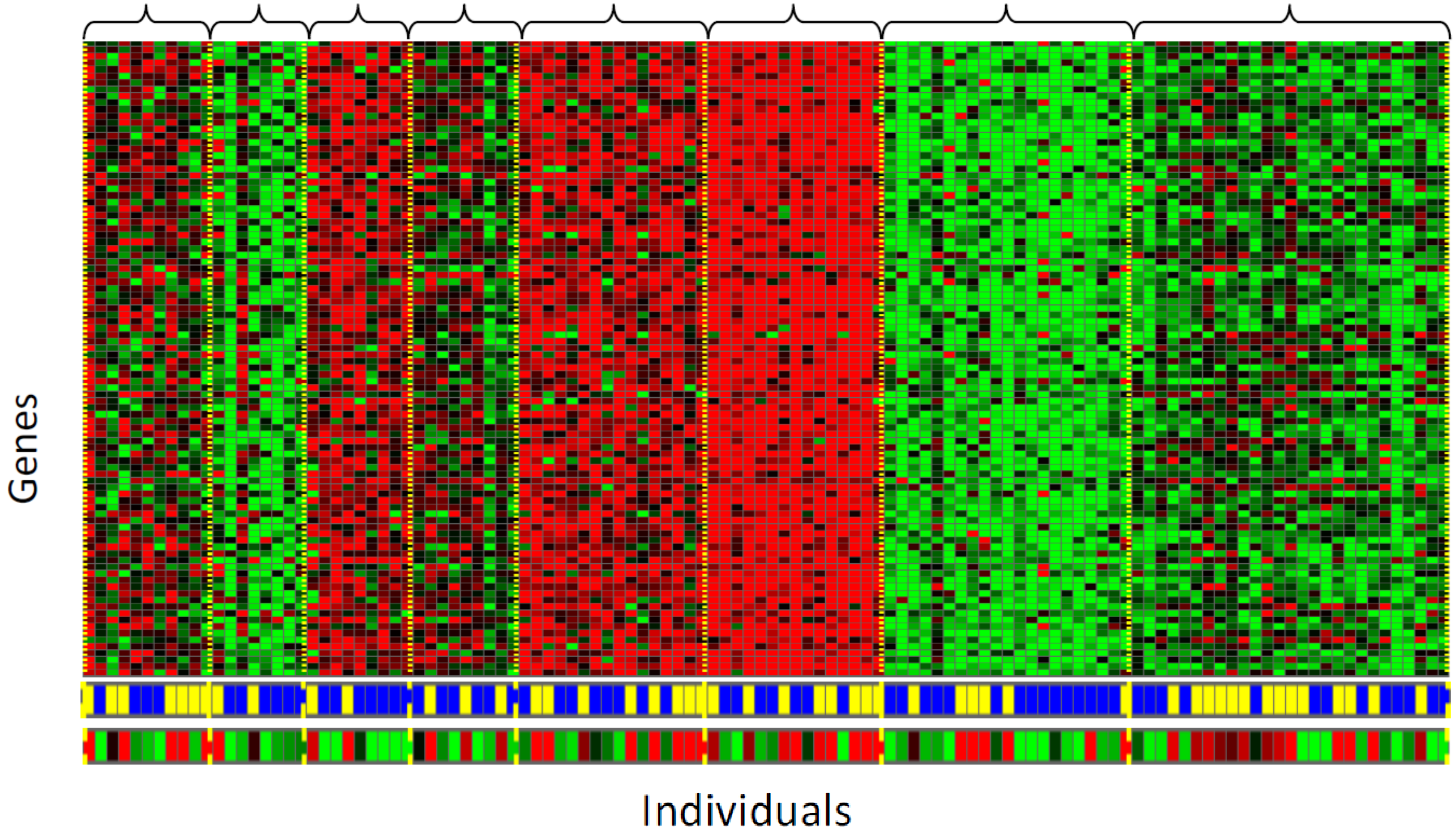
Market segmentation



Astronomical data analysis

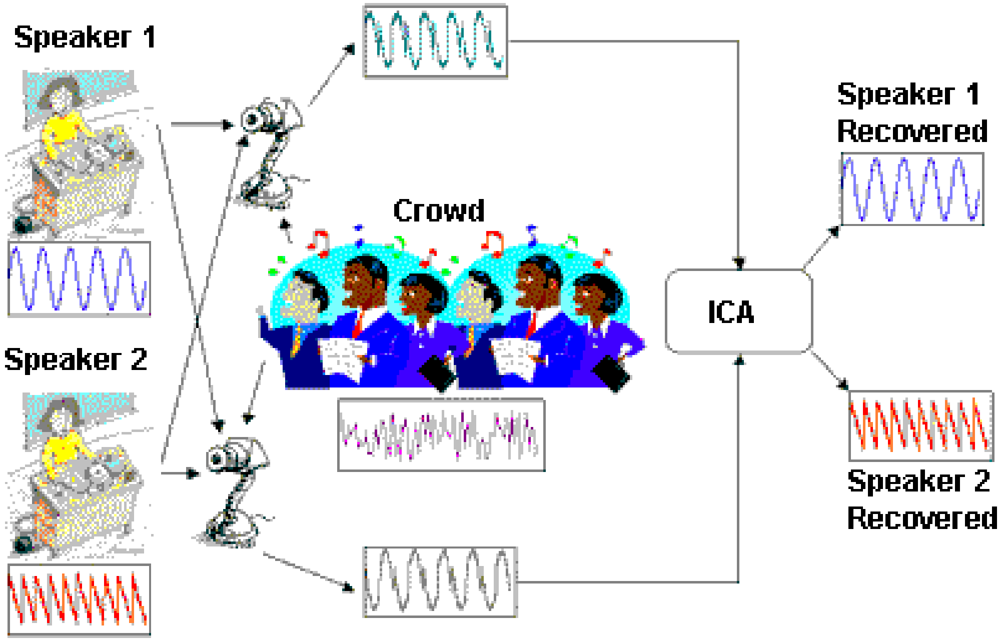
Unsupervised Learning

- Genomics application: group individuals by genetic similarity



Unsupervised Learning

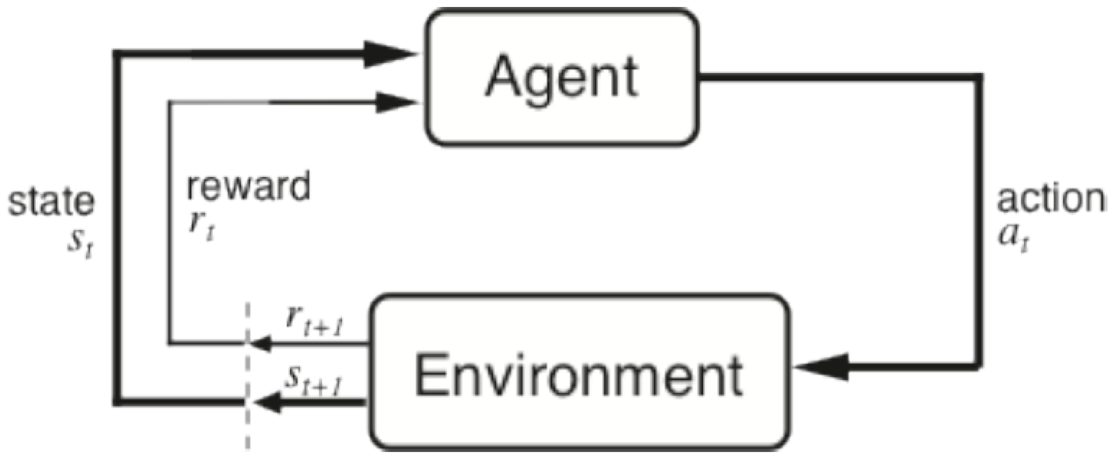
- Independent component analysis – separate a combined signal into its original sources



Reinforcement learning

- Given a sequence of states and actions with (delayed) rewards, output a policy
  - Policy is a mapping from states → actions that tells you what to do in a given state
- Examples:
  - – Credit assignment problem
  - – Game playing
  - – Robot in a maze
  - – Balance a pole on your hand

The Agent-Environment Interface



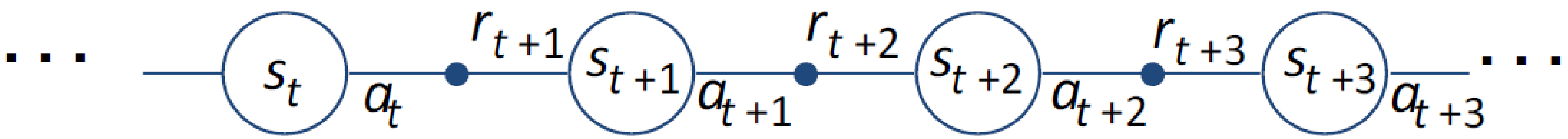
Agent and environment interact at discrete time steps : $t = 0, 1, 2, K$

Agent observes state at step t: $s_t \in S$

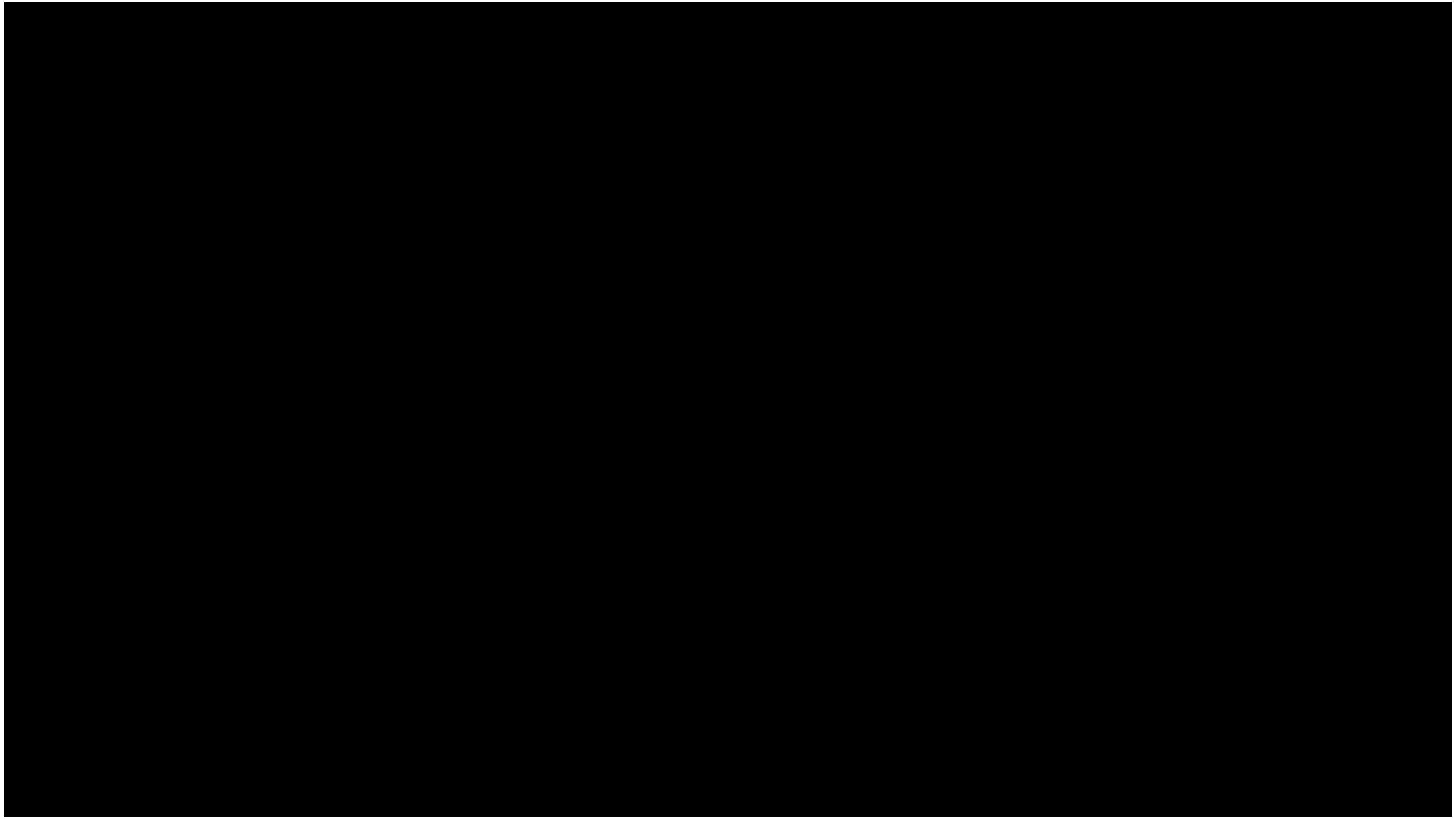
produces action at step t: $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}



Reinforcement learning



https://www.youtube.com/watch?v=4cgWya-wjgY

ML in a Nutshell

- Tens of thousands of machine learning algorithms
- Hundreds new every year
- Every machine learning algorithm has three components:
  - **Representation**
  - **Evaluation**
  - **Optimization**

Representation

- Decision trees
- Sets of rules / Logic programs
- Instances
- Graphical models (Bayes/Markov nets)
- Neural networks
- Support vector machines
- Model ensembles
- Etc.

Evaluation

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- Etc.

Optimization

- Combinatorial optimization
  - E.g.: Greedy search
- Convex optimization
  - E.g.: Gradient descent
- Constrained optimization
  - E.g.: Linear programming

Types of Learning

- **Supervised (inductive) learning**
  - Training data includes desired outputs
- **Unsupervised learning**
  - Training data does not include desired outputs
- **Semi-supervised learning**
  - Training data includes a few desired outputs
- **Reinforcement learning**
  - Rewards from sequence of actions

Inductive Learning

- **Given** examples of a function *(X, F(X))*
- **Predict** function *F(X)* for new examples *X*
  - Discrete *F(X)*: Classification
  - Continuous *F(X)*: Regression
  - *F(X)* = Probability(*X*): Probability estimation

ML in Practice

- Understanding domain, prior knowledge, and goals
- Data integration, selection, cleaning, pre-processing, etc.
- Learning models
- Interpreting results
- Consolidating and deploying discovered knowledge
- Loop

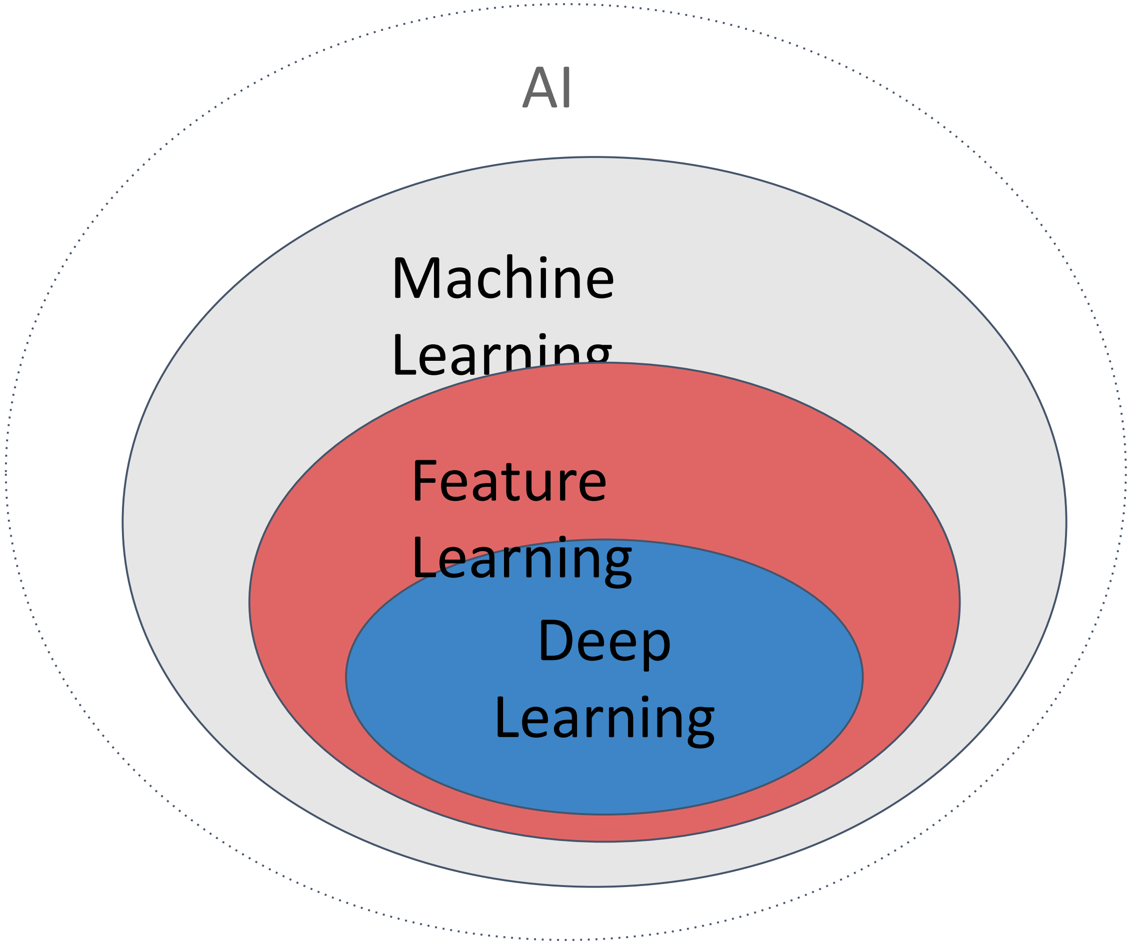

STILL WAITING
FOR MY NEURAL NETWORK TO TRAIN
memegenerator.net

Deep Neural Networks

Thanks to: Deep Learning by Google - Take machine learning to the next level

Example of *Inceptionism*
http://www.boredpanda.com/inceptionism-neural-network-deep-dream-art/

What is Deep Learning



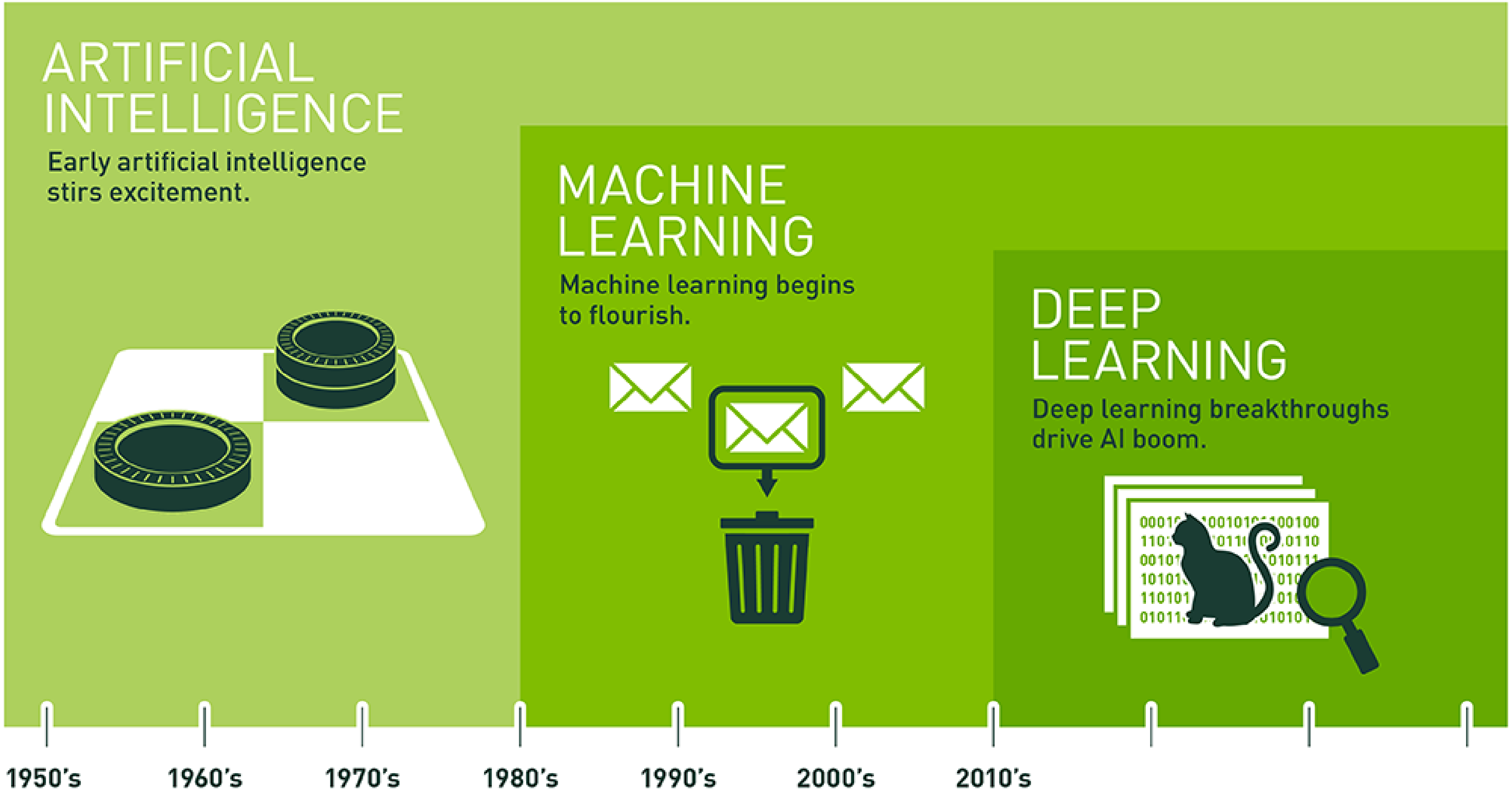
Deep Learning (DL) has emerged around the ‘10 as a general <u>tool to solve recognition problems</u> in:

- computer vision
- speech recognition
- robotics
- *discovering* new medicines
- *understanding* natural language
- *understanding* documents
- ranking
- … and many other applications!

Overview

- History
- Preliminaries: logistic classification
- Training
- Deep networks
- Regularization
- Architectures
  - Convolutional networks
  - Embeddings
  - Recurrent models

History



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Everything can be optimized in Computer Science

- Given a problem to solve P, it can be formalized as {P,C,F}
  - P := the problem formulation
  - $C = \{c_1, c_1, \ldots, c_n\}$:= set of configurations, each one of them representing a possible solution to P
  - $f: C \rightarrow R$:= function which provides a goodness measure of the configuration w.r.t. the problem to be solved
- Casting the problem via minimization means to maximize or minimize the function f in the C space, *independently on the implied meaning of P*

Minimization: to be used *always*?

- Problem P_1: *sort numbers* $x_1,x_2,\dots,x_N$ *in increasing order*
  - In this case, minimization could be left apart
  - In facts, there is at least one algorithm (e.g., quicksort) which brings directly to the best (in sense of the function f) configuration
- Problem P_2: foresee the stocks' trend
  - Much more difficult to formalize into an algorithm
  - Minimization comes to help [Yong et al. 2015]

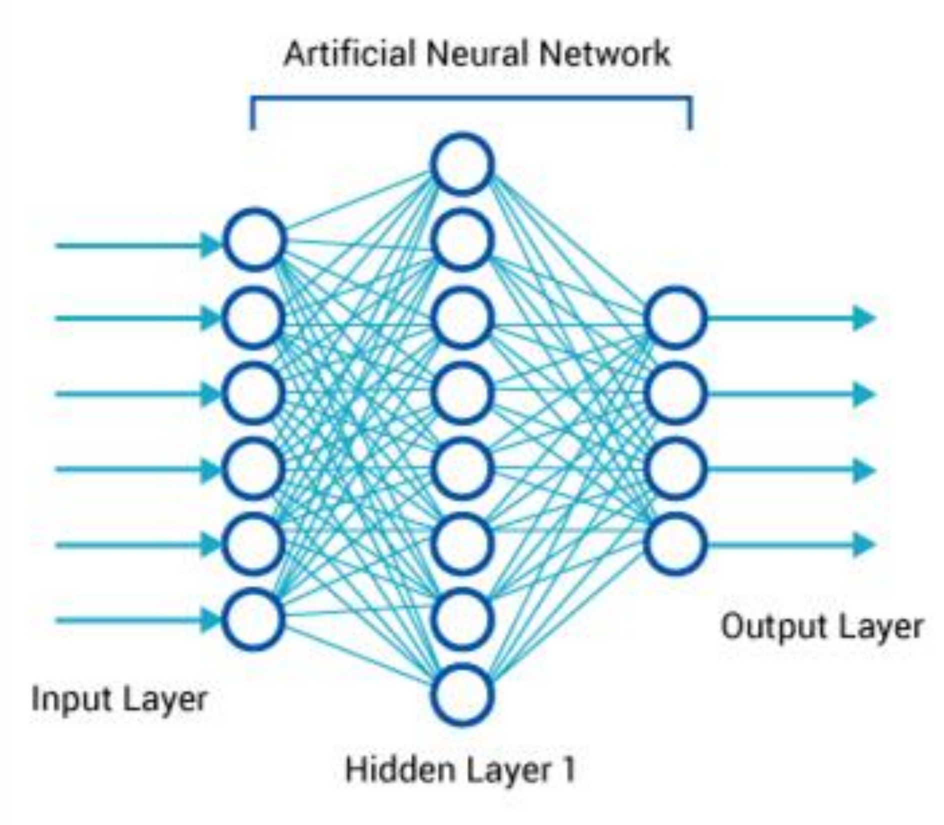
[Yong et al. 2015] Hu, Yong, et al. "Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review." *Applied Soft Computing* 36 (2015): 534-551.

Inside the minimization approach

- The main goal of an optimization approach is that of exploring the configuration space C looking for the best configuration given the function f (obviously avoiding the brute force way!)
- The set of configurations C give a space to explore (very often, a manifold)
- Optimizing means to explore the manifold by iterative approaches (e.g., the gradient descent family of strategies)
- The more the manifold is complex (non convex, multimodal), the more often local minima are met

Neural Networks [1943 - McCulloch & Pitts]

- Optimization approaches which scale *very well* with data
- We are talking about *artificial neurons* and *layered computation*

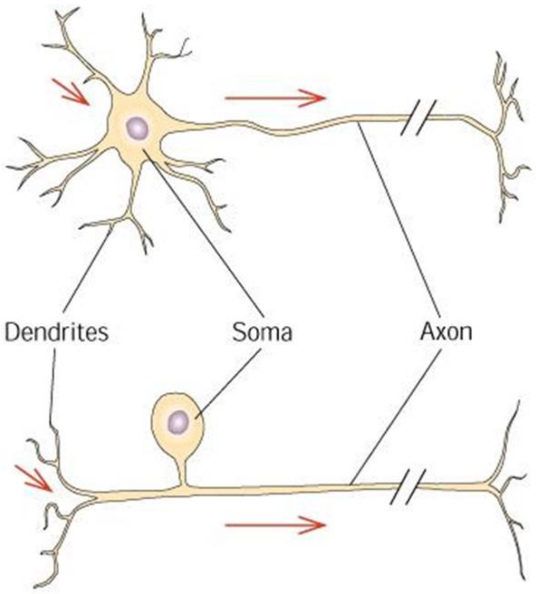


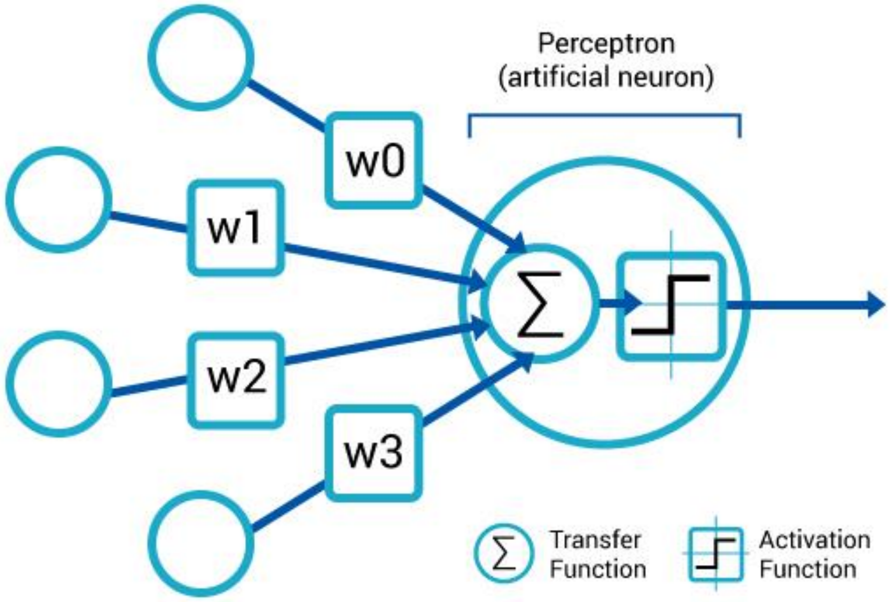
Neural Networks - Neurons

NN are composed by artificial neurons (1943 - McCulloch & Pitts).

Each neuron has:

- *dendrites* (**inputs**)
- a *nucleus/soma/perceptron* (**transfer fuction + activation function**)
- a

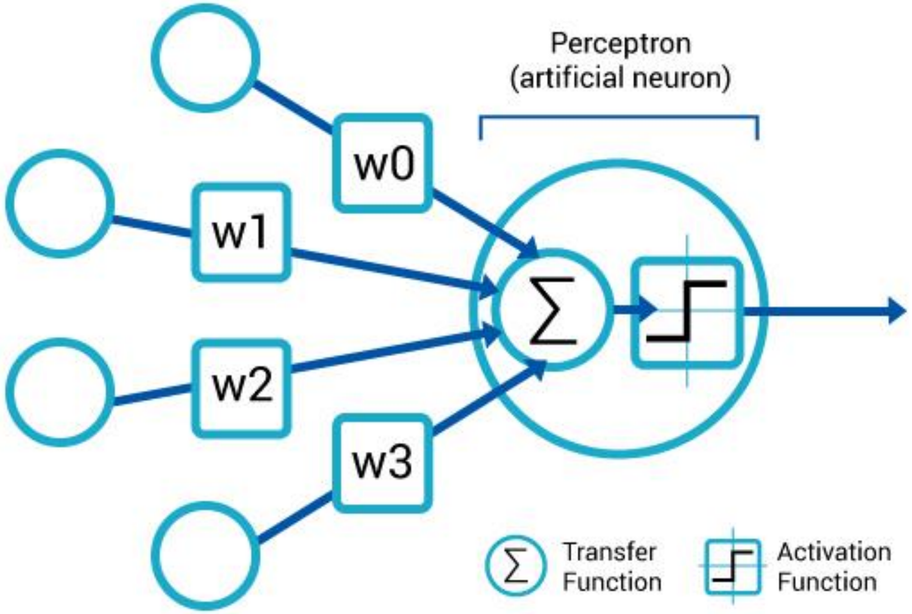


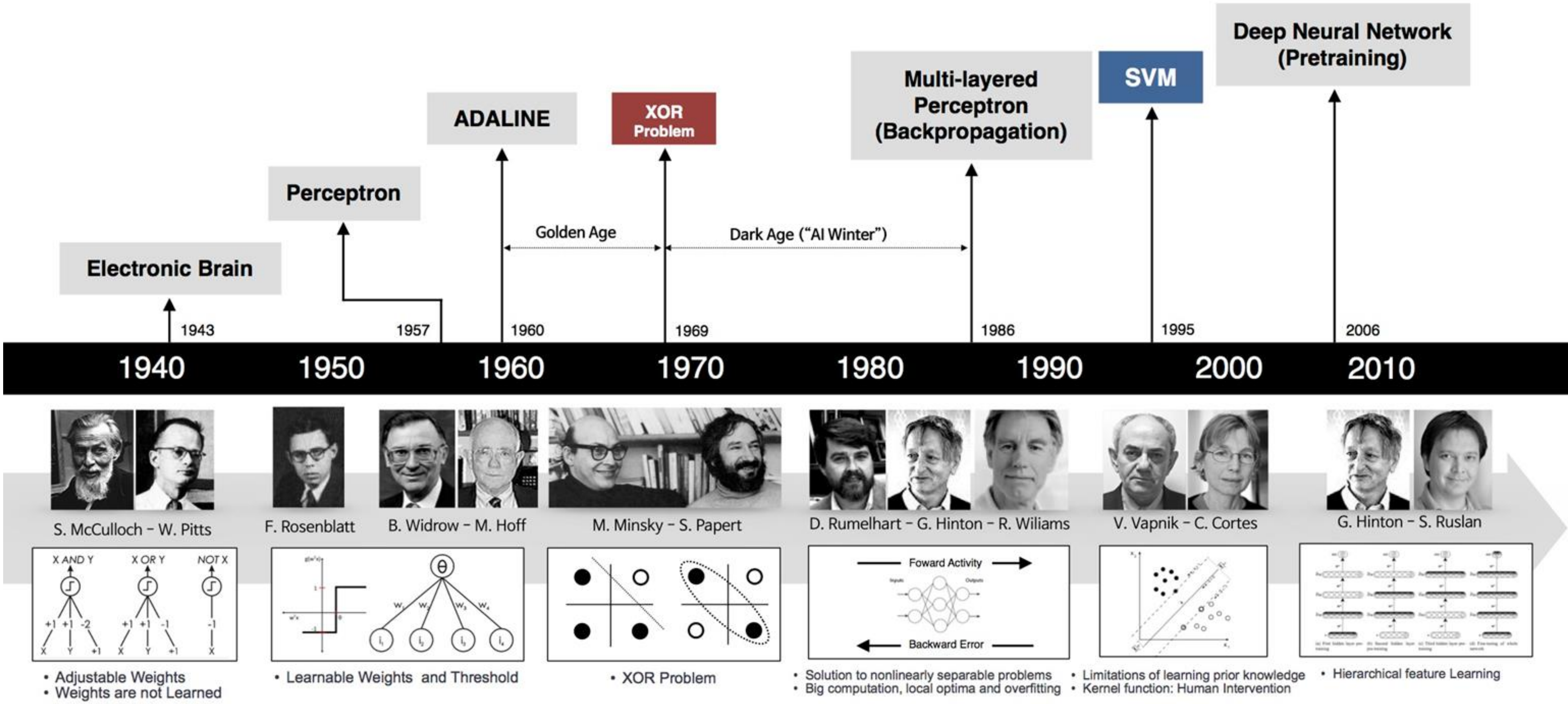


Neural Networks - Neurons (2)

The information flow is **unidirectional**:

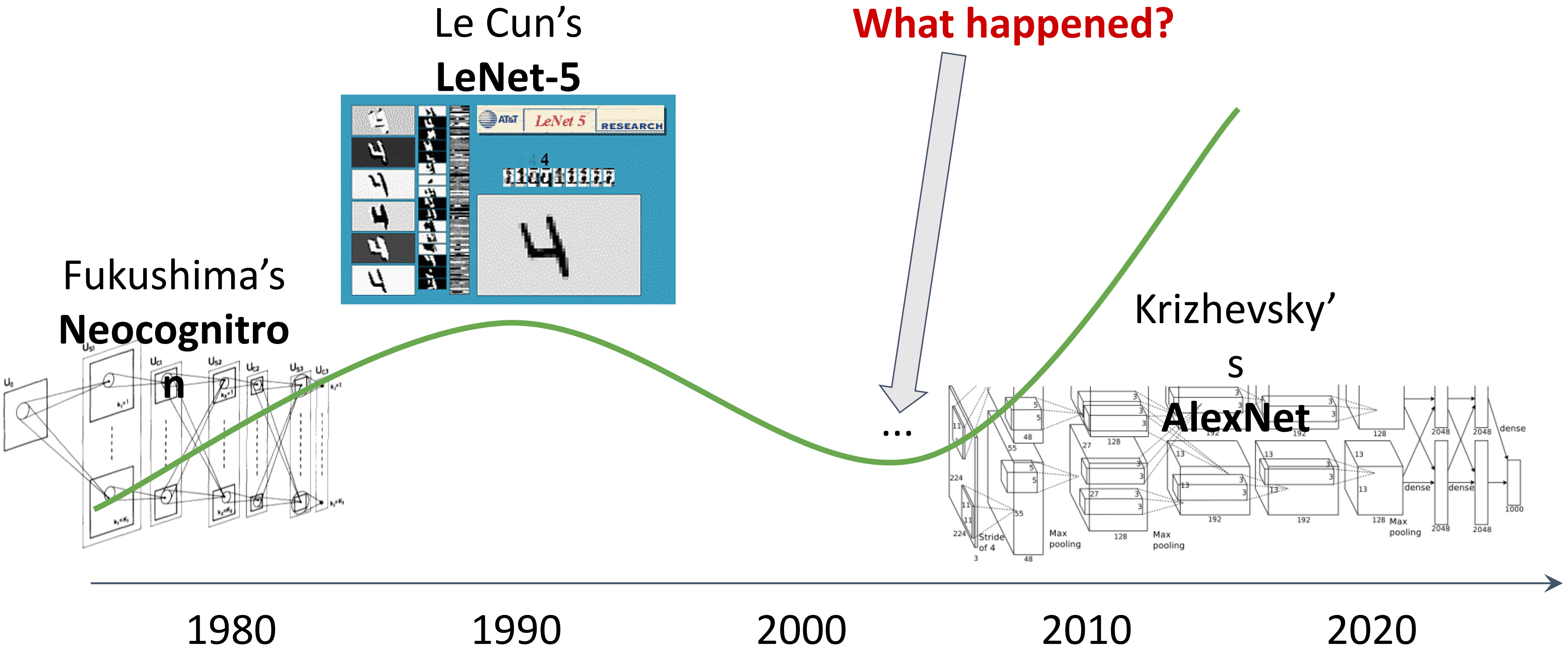
- The neuron get **inputs** (electric potentials) from the *dendrites*, that weight them (w_i's)
- In the *nucleus*, the weighted inputs are summed together (the **transfer** Σ of the whole information coming from the dendrites)
- In the *nucleus*, the summation flows into an **activation function**, which may inhibit, diminish or amplify it
- The **output** of the activation function





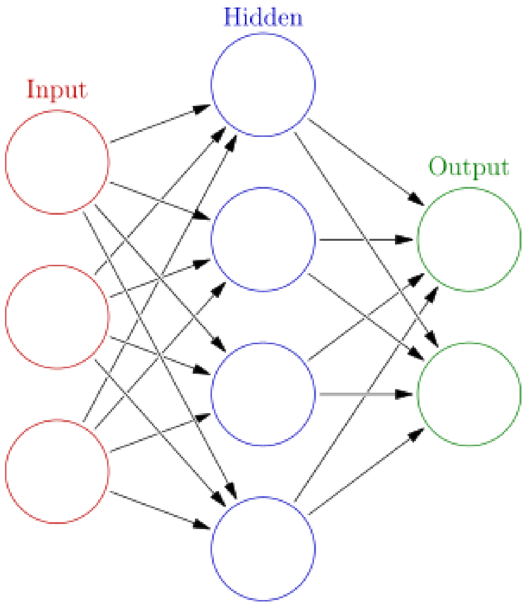
Electronic Brain
Perceptron
ADALINE
XOR Problem
Multi-layered Perceptron (Backpropagation)
SVM
Deep Neural Network (Pretraining)
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

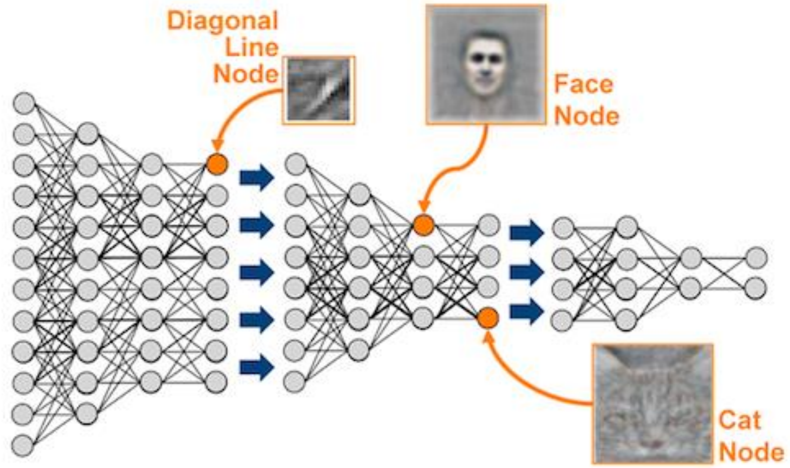
Neural Networks - the renaissance



http://people.csail.mit.edu/torralba/research/drawCNN/drawNet.html

Neural Networks and Deep Learning





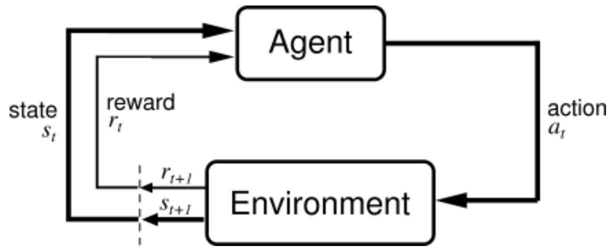
Neural Networks

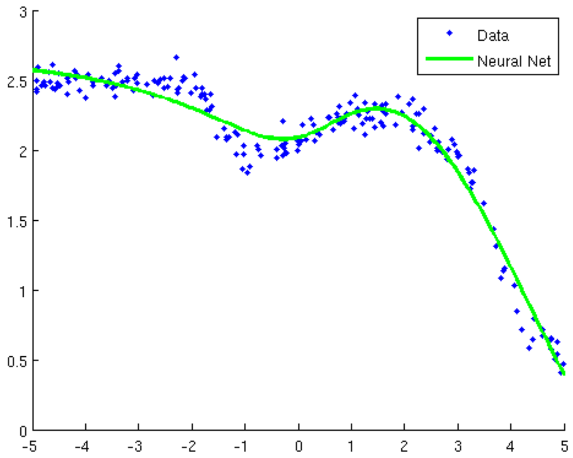
GPUs + Data

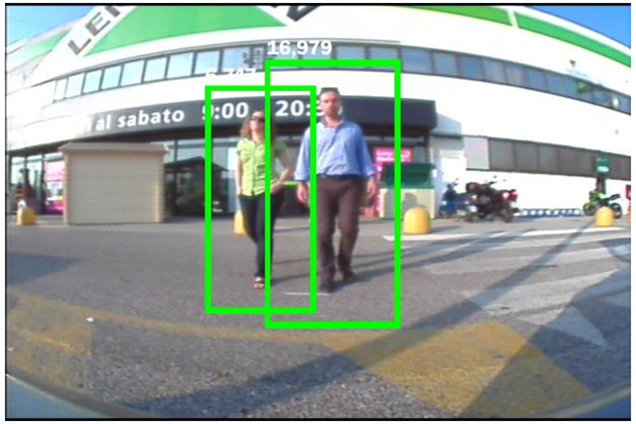
Deep Neural Networks

Supervised Classification

- Traditional kind of problem the NN do solve
  - Regression
  - Ranking
  - Reinforcement Lea





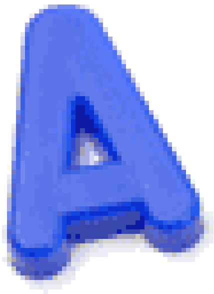


Labels {‘a’, ‘b’, ‘c’, ‘d’, ‘e’}

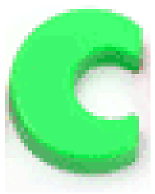
Preliminaries: logistic classification

Logistic Classifier

- It assigns a *score y* to the input **x** through a linear model (W,**b**)
- The score helps to identify the class label that wins



$$\underbrace{x}_{1xF}\underbrace{W}_{FxC} + \underbrace{b}_{1xC} = \underbrace{y}_{1xC}$$



To be trained via a training procedure

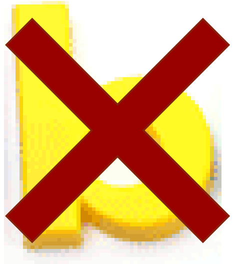
x *= input or feature vector;* ***F*** *= number of features;* ***W*** *= weights matrix;*
C *= number of classes* ***b*** *= biases;* ***y*** *= output or logits/scores vector*

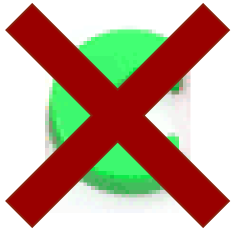
Logistic Classifier - the score is not enough

$$xW + b = y \Rightarrow s(y)$$

$$\begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix} \qquad \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$$

scores   probabilities

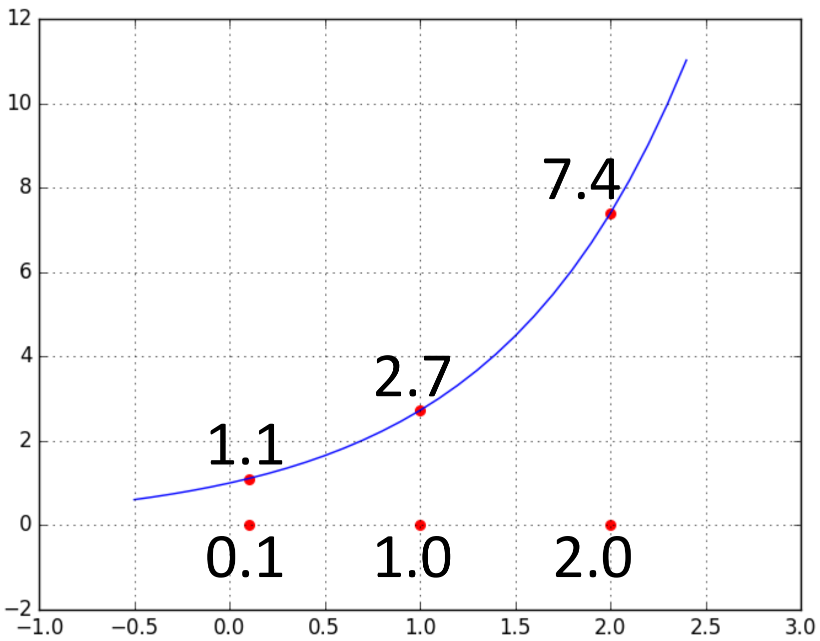




$s(\mathbf{y})$ is the **SOFTMAX** function

Softmax function

- Converts scores into probability distributions
  - ℝ → (0,1)
  - *Open* codomain!
- The softmax function highlights the largest values and suppress values which are significantly below the maximum value



$$\begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix} \Rightarrow s(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{C} e^{y_j}} \Rightarrow \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$$

scores

probabilities

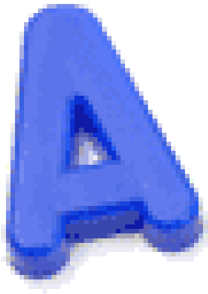
One-Hot Encoding (OHE)

- → *There is only one correct label for each input sample*
- → *There is the need to evaluate the classification result*
- OHE encodes labels for a C-class problem in R^C, indicating the c-th class label with 1, the rest 0's
  nee… representation w… …icient especially in the case C is very big (thousands or millions of classes...!)

Which distance measure?

MSE

Cross-Entropy

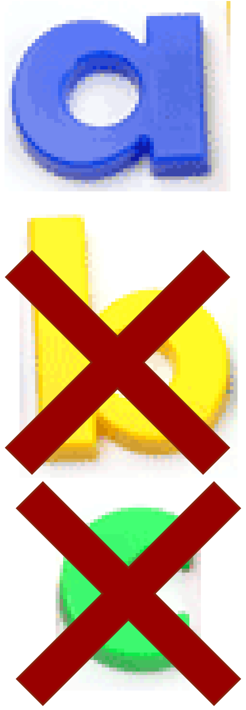


$s(\mathbf{y})$

$\begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$? $\begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$

our results

g.t. labels



Cross-Entropy

GOAL: computes the distance between two probability vectors

- Non symmetric function $D(\mathbf{s}, \mathbf{L}) \neq D(\mathbf{L}, \mathbf{s})$

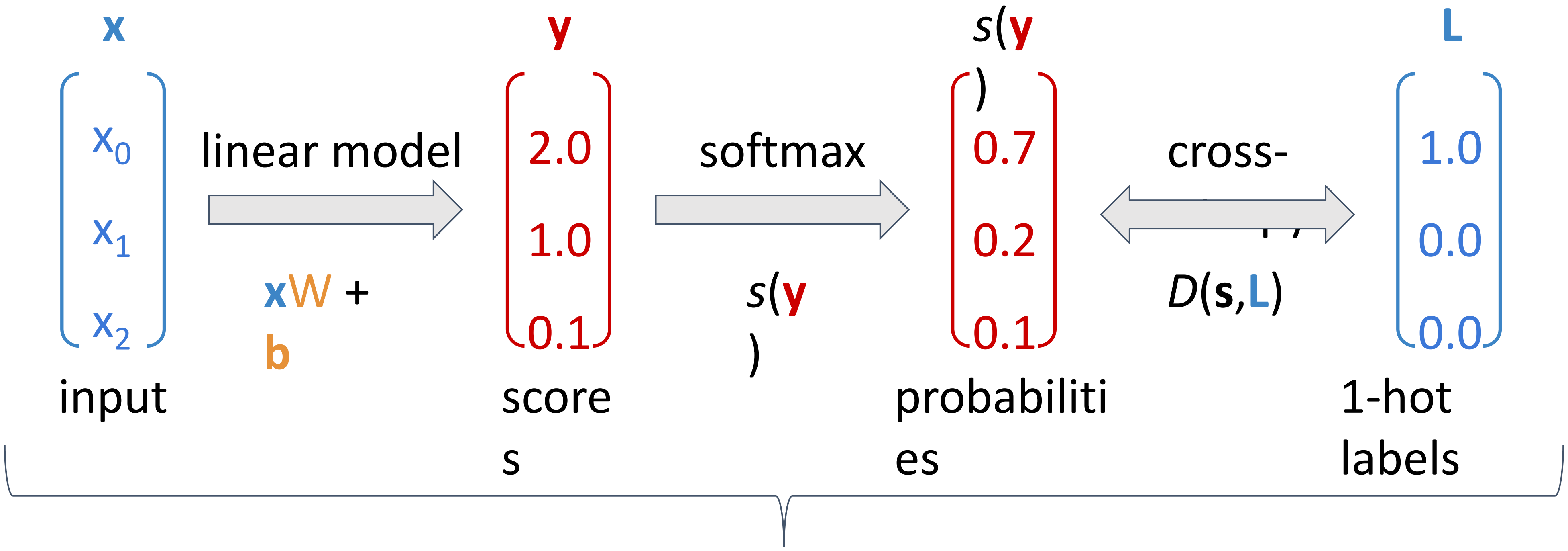
$$D(\mathbf{s},\mathbf{L}) \begin{cases} \to 0 & \text{high similarity} \\ \to \infty & \text{low similarity} \end{cases}$$

- The *log(·)* makes the training faster to converge than the other alternative MSE function $(\Sigma(s(y)-l)^2)$

$$s(\mathbf{y}) = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$D(\mathbf{s},\mathbf{L}) = -\sum_{j=1}^{C} L_i \log s_i = 0.36$$

Résumé



Multinomial Logistic Classification

$$D(s(\mathbf{x}W+\mathbf{b}), \mathbf{L})$$

Training

Gradient Descent

GOAL: search for the nearest local minimum of a function F

IDEA: iterate on the parameter set proportionally to the negative of the function gradient

$$\theta_{t+1} = \theta_t - \alpha \nabla F(\theta_t), t \geq 0$$

such that

$$F(\theta_0) \geq F(\theta_1) \geq F(\theta_2) \geq \ldots$$

ϑ = set of parameters

Gradient Descent

- GOAL: minimize a loss function

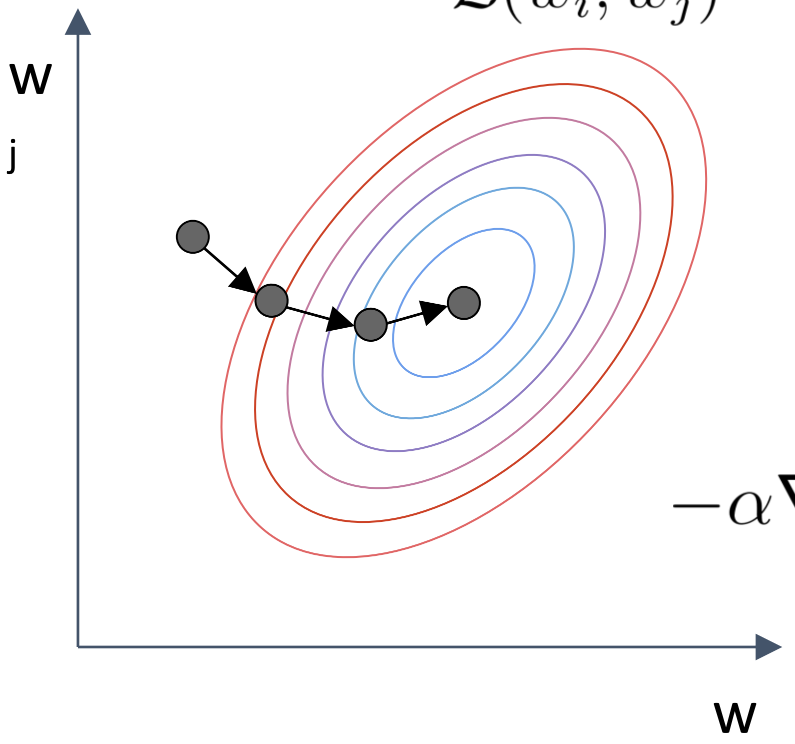
$$D(\mathbf{s},\mathbf{L}) \begin{cases} \to 0 & \text{good} \\ \to \infty & \text{bad} \end{cases}$$

- Needs to compute the entire training set performance of our linear model, that consists in N inputs (which is, in general, very big)
- Needs to minimize a loss function, which depends on W (big matrix) and **b**

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} D(s(\mathbf{x}_i W + \mathbf{b}), \mathbf{L}_i)$$

Loss = average cross-entropy

$\mathcal{L}(w_i, w_j)$



learning step

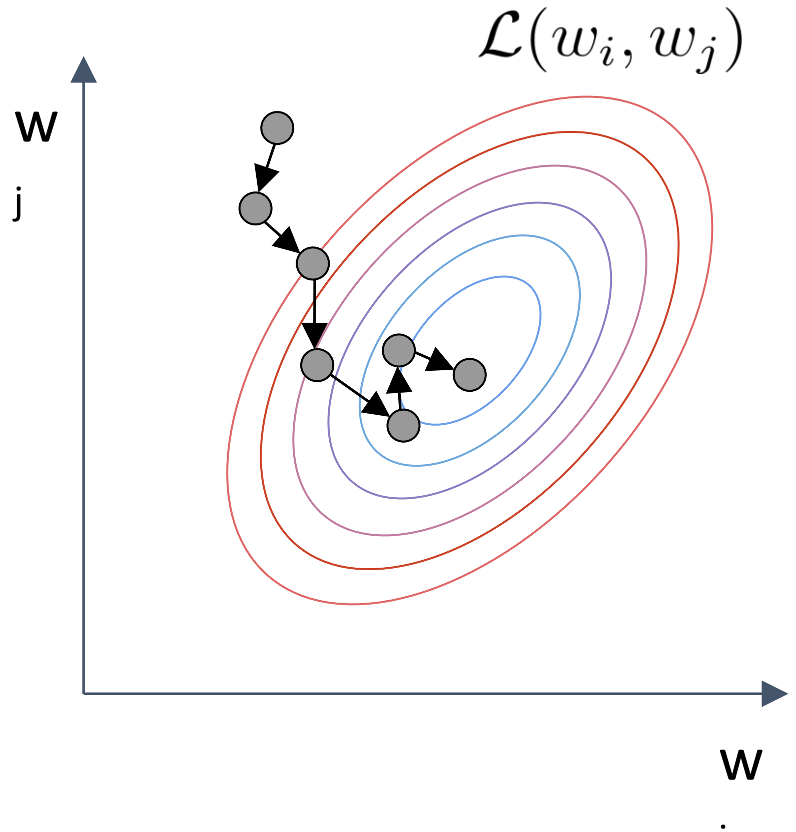
$-\alpha \nabla \mathcal{L}(w_i, w_j)$

***N** = number of input vectors; **α**: learning rate value*

Stochastic Gradient Descent (SGD)

- <u>IDEA</u>: use a random subset (*batch*) of the data (of a given *size*) to compute an approximation of the gradient of the loss function

Iterative implementation of the GD algorithm
At each step, a new batch is extracted



  - Pros:
    - simple but sufficiently effective
    - fast (depending on batch size)
    - scale the problem with data and model size
  - Cons:
    - needs more iterations to converge

… but tricks to ameliorate SGD are present!

SGD trick 1: momentum

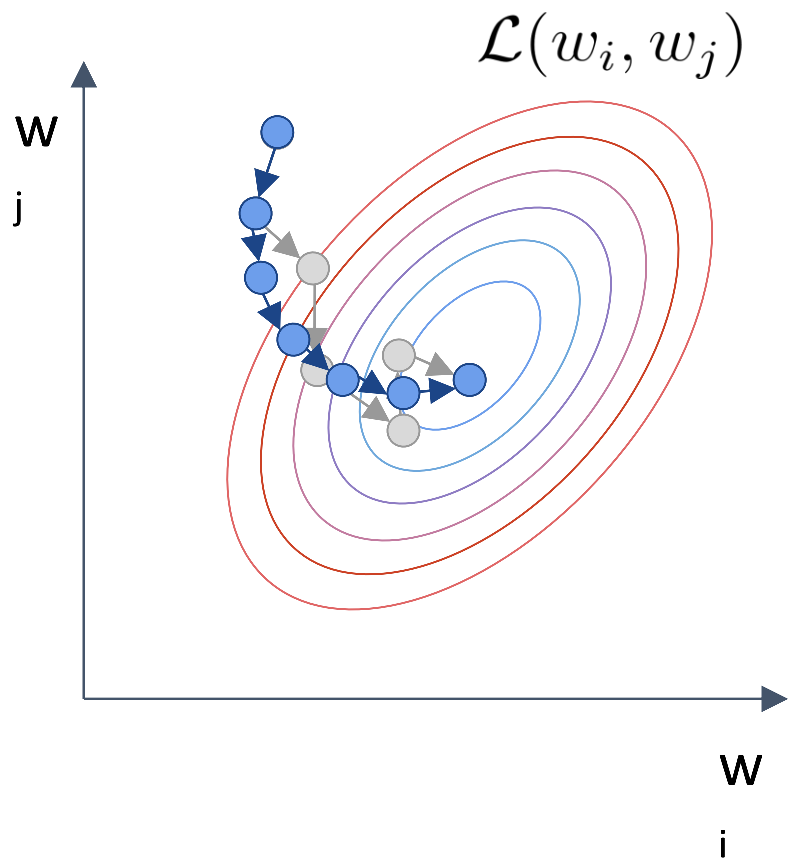
- GOAL: improve the convergence of the optimizer exploiting the accumulated knowledge from previous steps

- IDEA: add a fraction of the previous update vector to the current update vector

$$M_0 = \nabla \mathcal{L}_0(w_i, w_j) = 0$$

$$M_t = \alpha \nabla \mathcal{L}_t(w_i, w_j) + \beta M_{t-1}$$

$$(w_i, w_j)_t = (w_i, w_j)_t - M_t$$



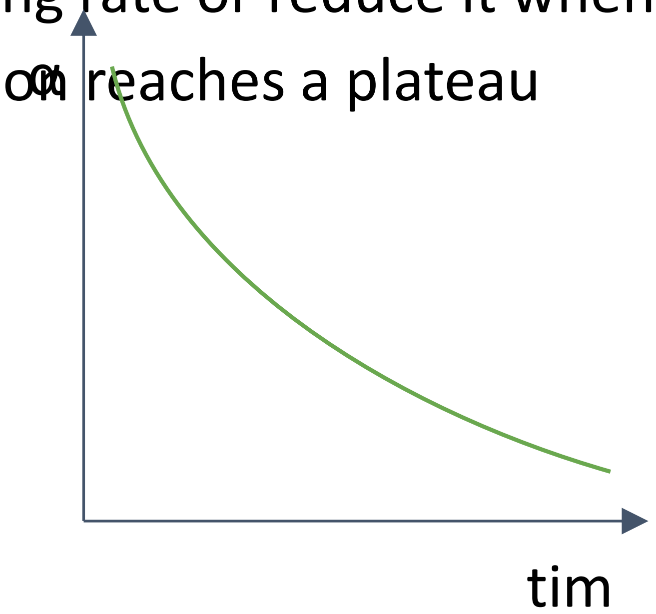
faster convergence and oscillation reduction

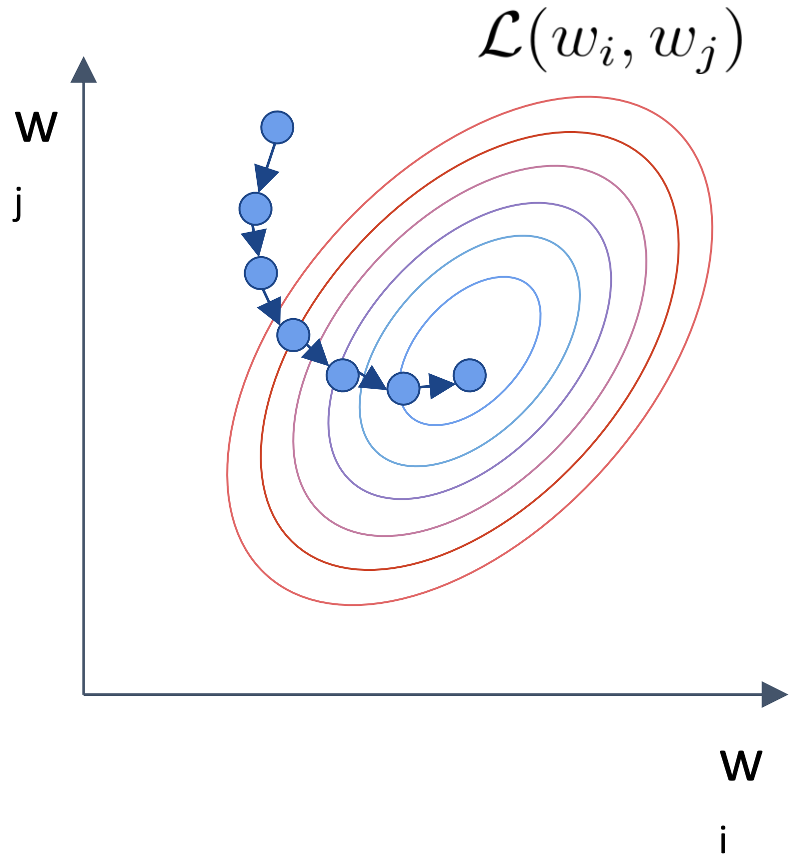
M *= momentum;* ***β****: friction value (usually 0.9)*

SGD trick 2: learning-rate decay

- GOAL: make the optimization more robust and accurate over time

- IDEA: apply a decay function to the learning rate or reduce it when the loss function reaches a plateau

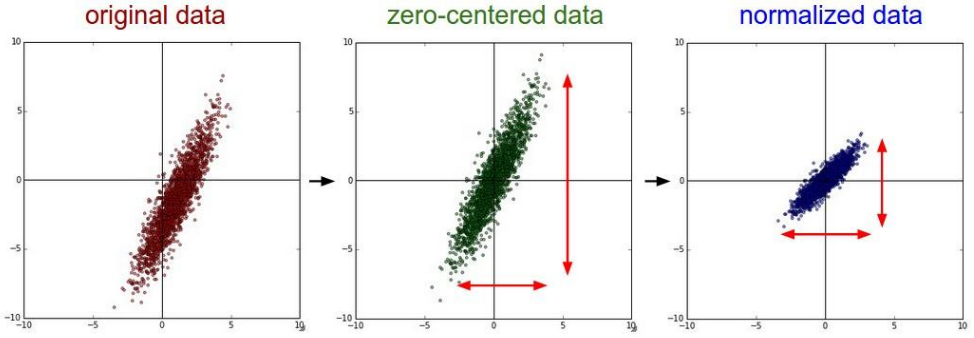




SGD trick 3: z-normalization

- <u>GOAL</u>: avoid numerical instability

The values involved in the calculation of the gradient descent never get too big or too small



- <u>IDEA</u>: remove the mean and normalize over the variance of the *i*-th feature of the input vector **x**

$$\mathbf{n} = \frac{\mathbf{x} - E[\mathbf{x}]}{Var[\mathbf{x}]}$$

z-normalization

$$E[\mathbf{x}] = 0$$

$$\forall(i,j), Var[\mathbf{x}_i] = Var[\mathbf{x}_j]$$

http://cs231n.github.io/

SGD trick 4: initialization

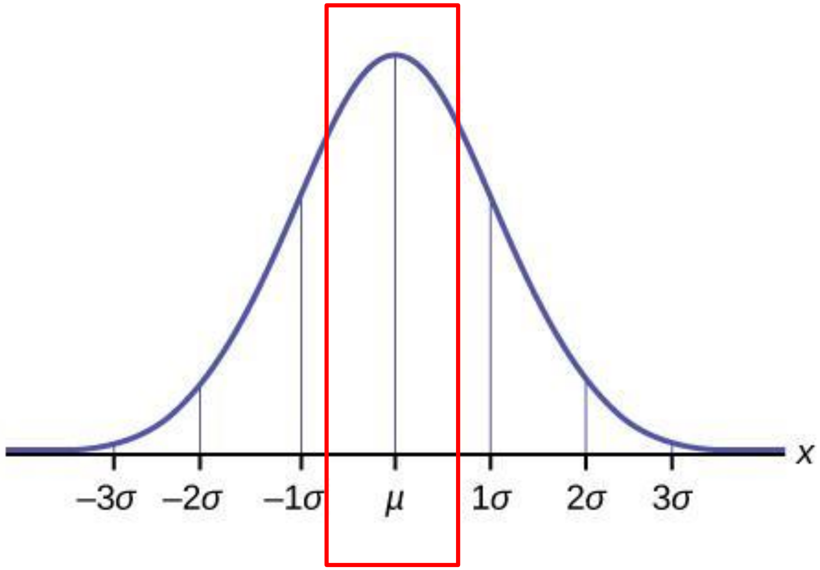
A random initialization of the weights and the zero-init of the biases is critical to get a good starting point for the training phase and the convergence of the SGD algorithm.

$$w_i = \mathcal{N}(\mu = 0, \sigma \to 0)$$

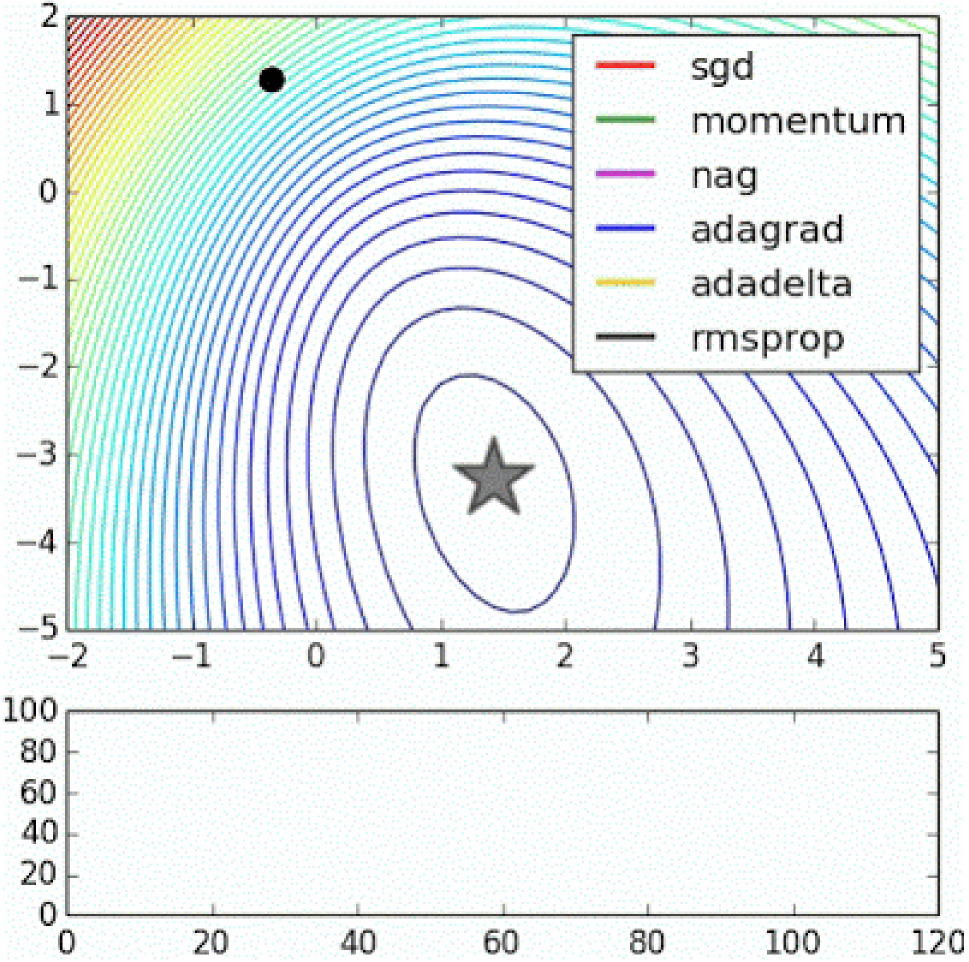
$$\mathbf{b} = \begin{bmatrix} 0_1 & 0_2 & \dots & 0_C \end{bmatrix}$$

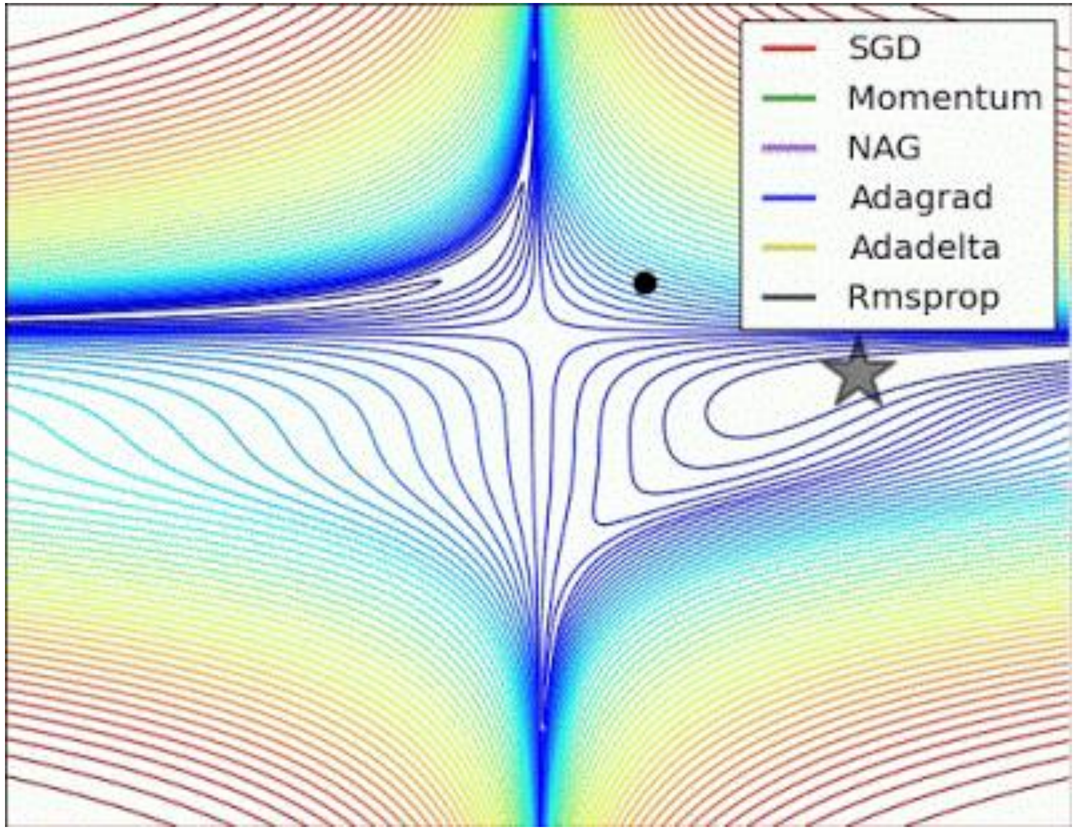
$$\mu = 0 \land \sigma \to 0 \implies \forall (i,j), w_i = w_j \pm \epsilon$$

equal probability
of the weights
(no prior)



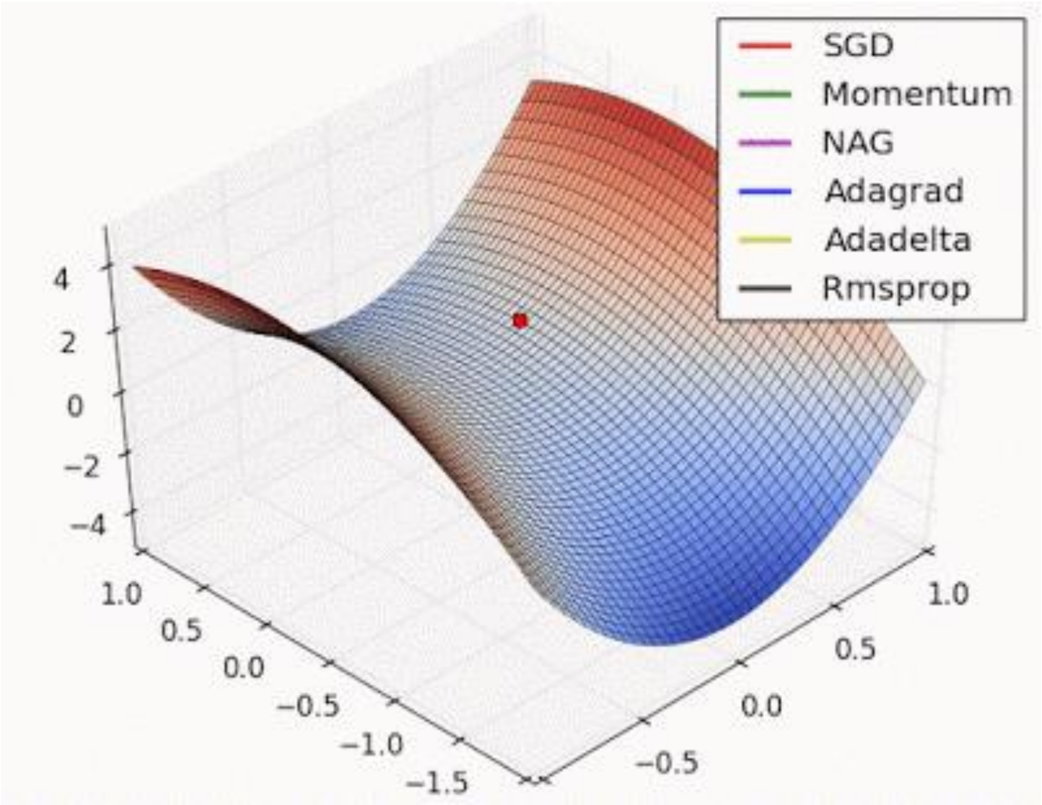
Gradient Descent: graphical representation (2D)

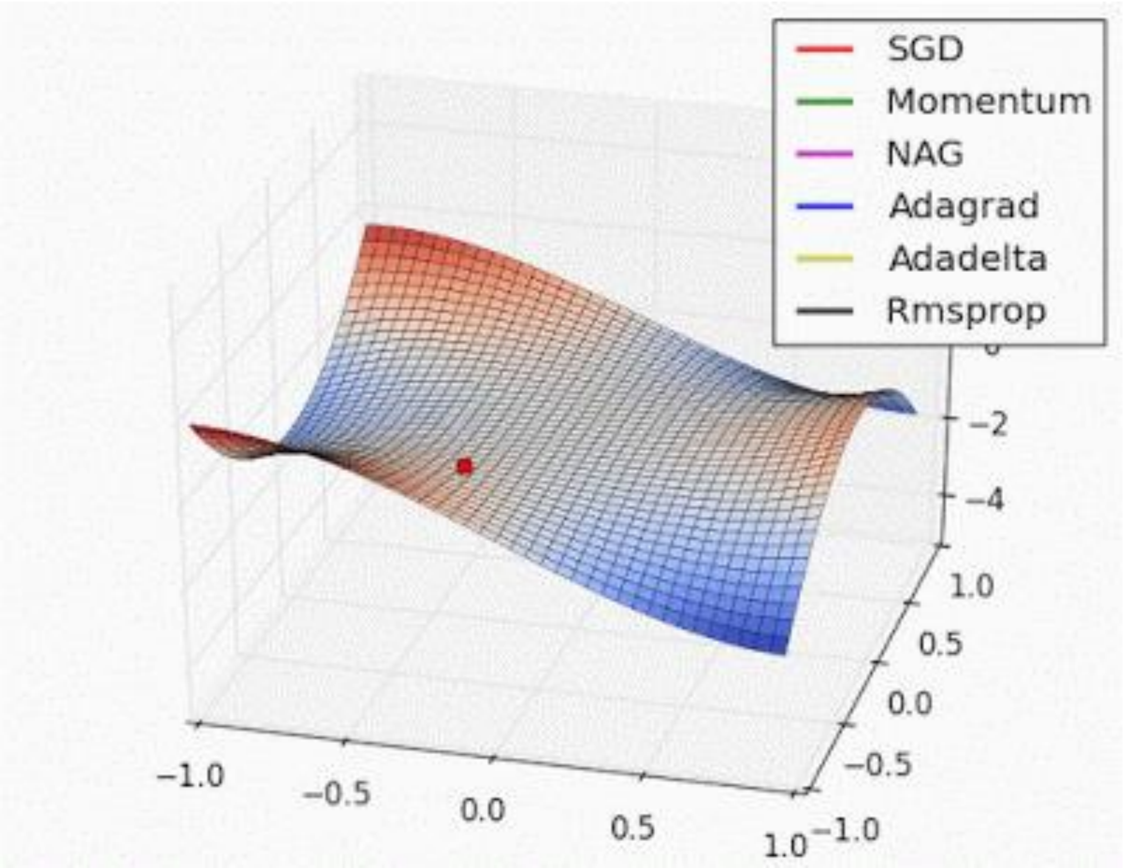




http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

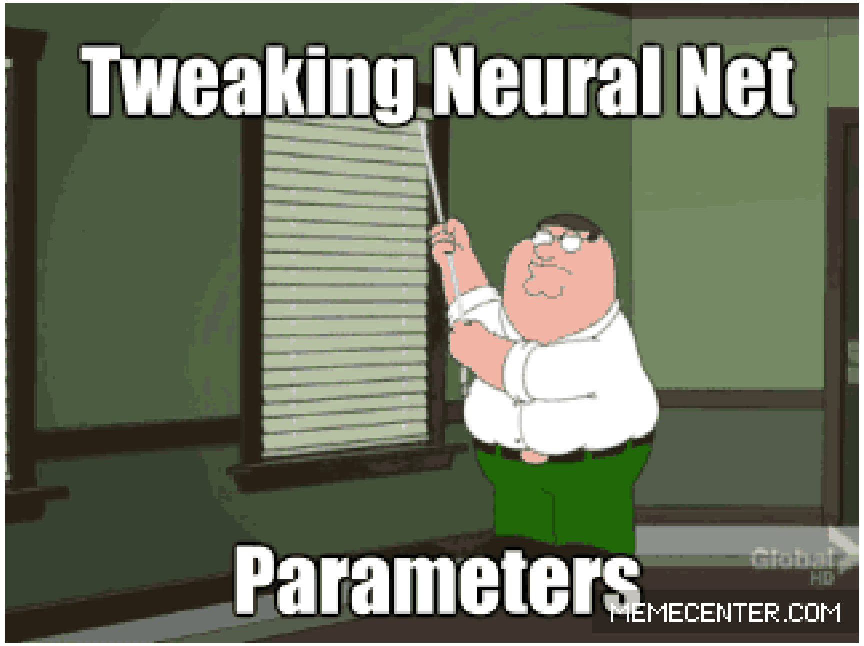
Gradient Descent: graphical representation (3D)





http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

SGD: tuning



SGD

Many hyperparameters:

- initial learning rate
- learning rate decay
- momentum
- batch size
- weight initialization

AdaGrad

SGD modification which implicitly applies momentum and learning rate decay. It uses fewer parameters:

SGD: H. Robinds and S. Monro, “A stochastic approximation method,” Annals of Mathematical Statistics, vol. 22, pp. 400–407, 1951.

AdaGrad: J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online leaning and stochastic optimization,” in COLT, 2010.

From MLC to NN to Deep-NN

- **Multinomial Logistic Classification** (MLC)
  - *fast*: efficient computation due to the linear model
  - *stable*: small input variations generates small output variations and the derivative of the model is constant

$$D(s(\mathbf{x}W+\mathbf{b}), \mathbf{L})$$

but...

  - trains only a "small" parameter set

$$\underbrace{\mathbf{x}\,\overbrace{W_1 + \mathbf{b}_1}\, = \mathbf{y}_1\,\overbrace{W_2 + \mathbf{b}_2}\, = \mathbf{y}_2}_{\text{still a linear model...!}} \Rightarrow \mathbf{x}\,W_3 + \mathbf{b}_3 = \mathbf{y}_3$$

- What if I concatenate multiple MLCs?

From MLC to NN to Deep-NN

- GOAL: build a bigger and non linear model
- IDEA: concatenate many linear systems (MLC) and insert a ***non linear function*** between two consecutive MLCs, that is, the activation function

Which function?

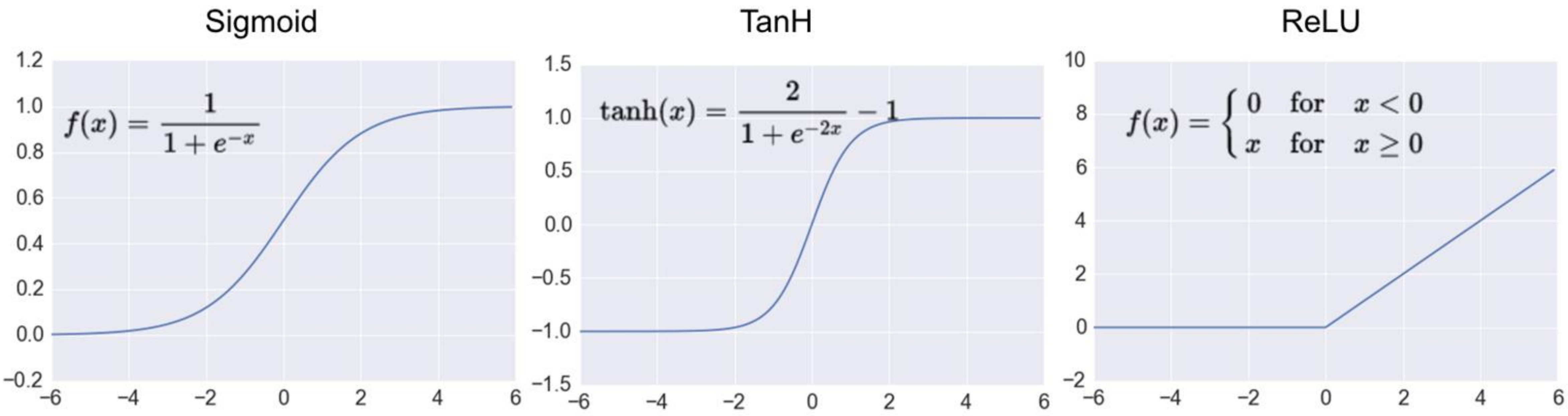
ReLU

sigmoid

tanh

$$\mathbf{x} \quad W_1 + \mathbf{b}_1 = \mathbf{y}_1 \rightarrow \; ? \; \rightarrow W_2 + \mathbf{b}_2 = \mathbf{y}_2 \rightarrow s(\mathbf{y}_2)$$

Choose the (non linear) activation function



- A perceptron classic

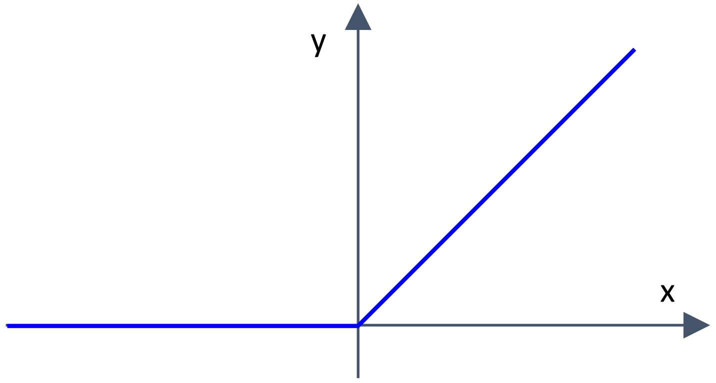
- similar to sigmoid

- simplest function
- constant derivative

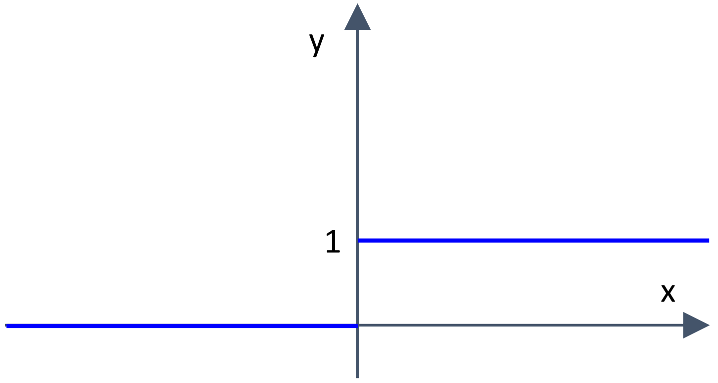
http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

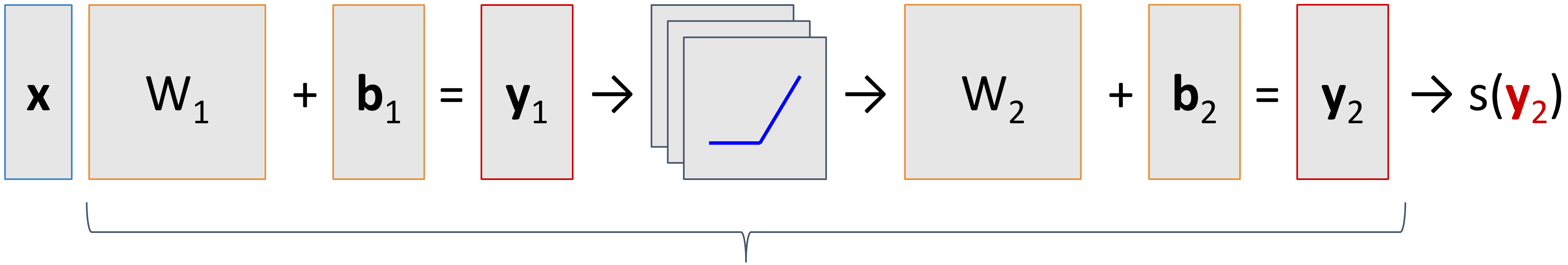
ReLU: Rectified Linear Unit

$$ReLU(x) = max(0, x)$$



derivative

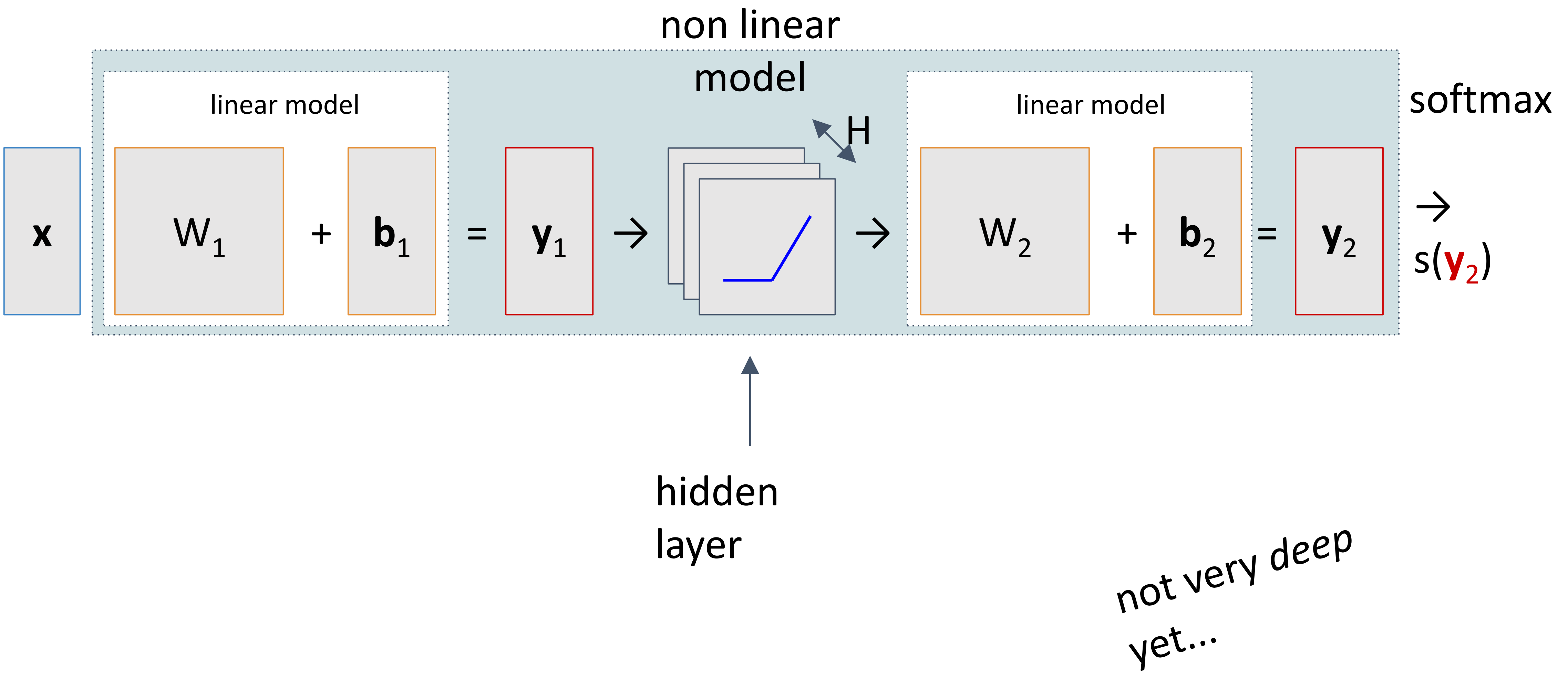




no more linear...!

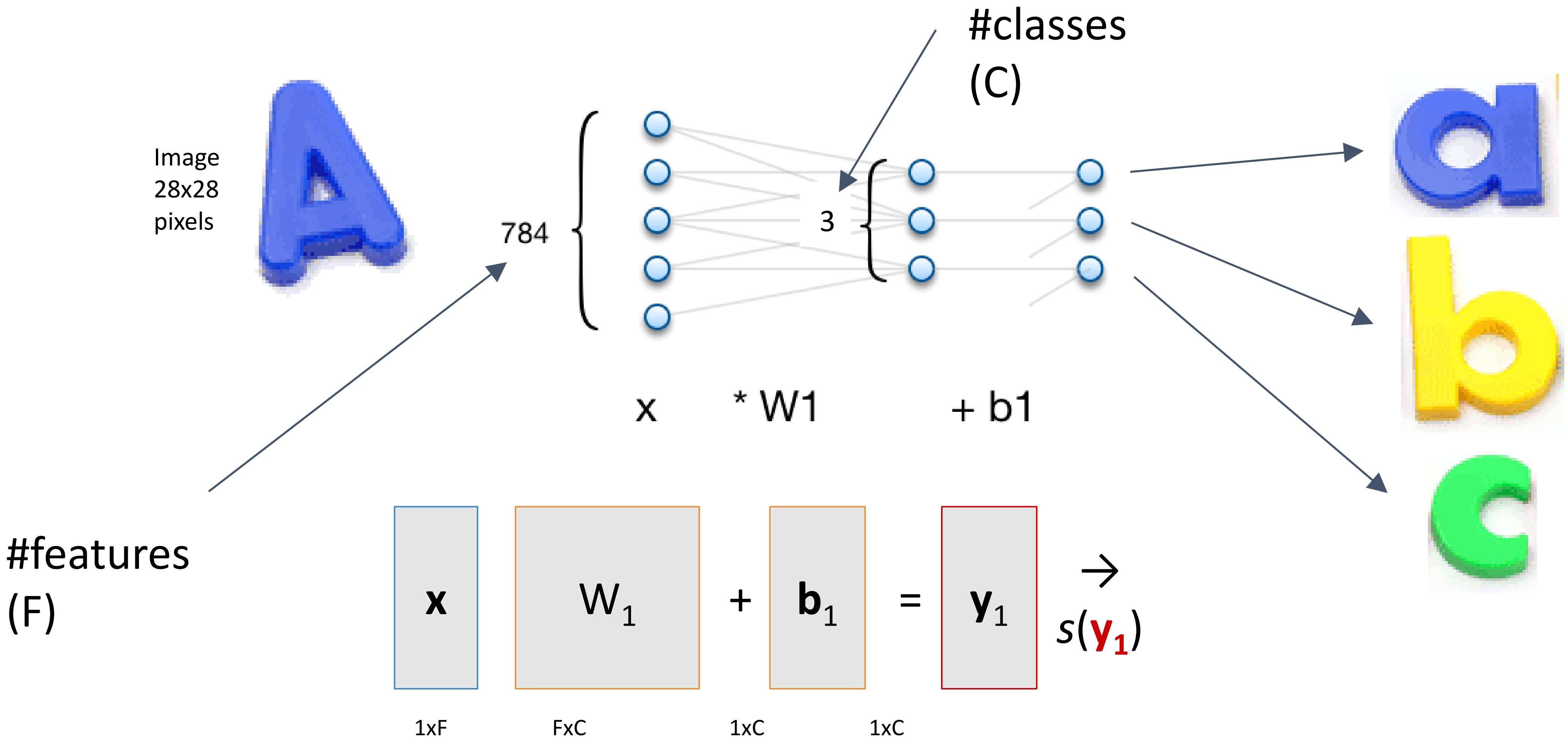
http://www.kdnuggets.com/2016/03/must-know-tips-deep-learning-part-2.html

Résumé

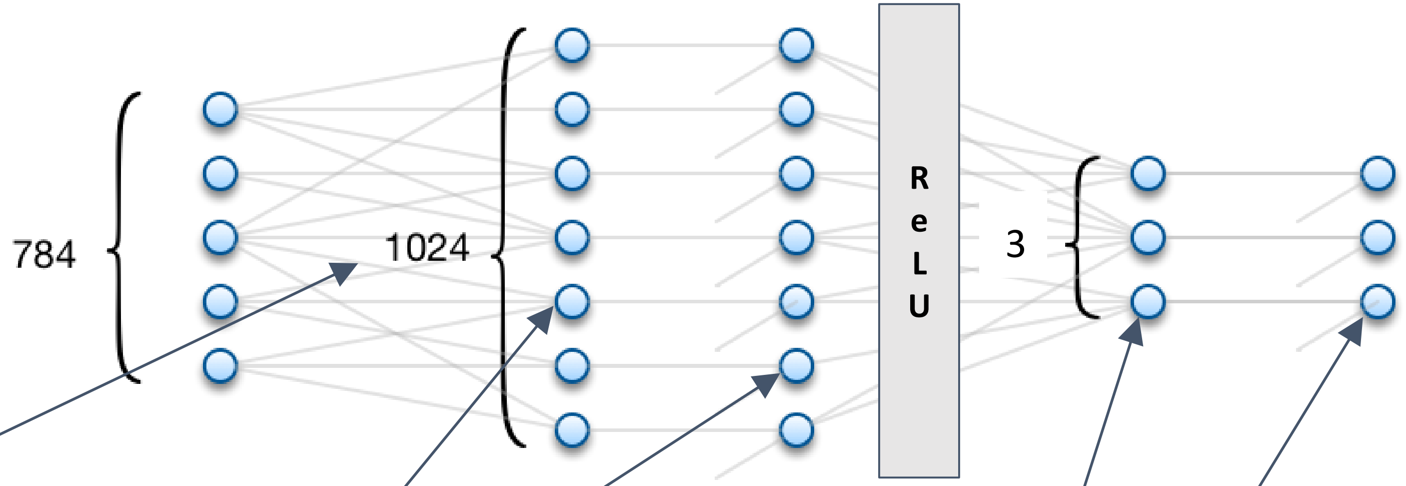


***H** = number of "neurons" in the hidden layer*

Practical example (1/2)

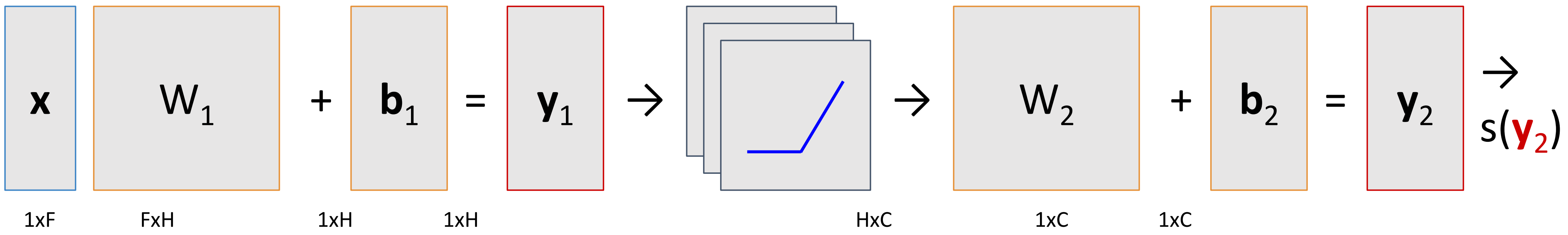


Practical example (2/2)



#hidden_neurons
(H)

$$\mathbf{x} * W_1 + \mathbf{b}_1 = \mathbf{y}_1 \rightarrow \text{ReLU}(\mathbf{y}_1) * W_2 + \mathbf{b}_2 = \mathbf{y}_2$$



Examples of deep networks - conclusions