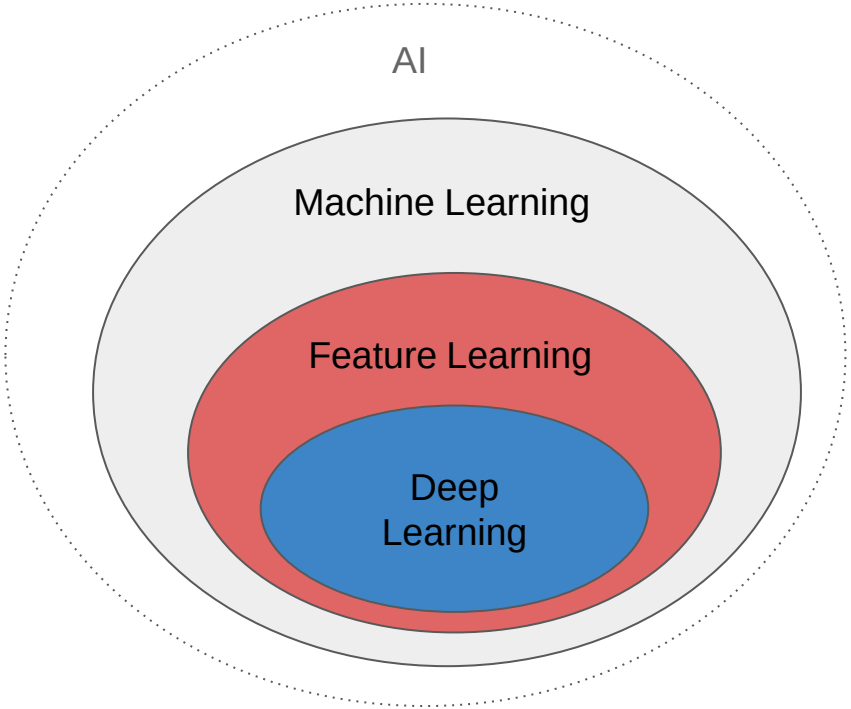
Deep Neural Networks

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



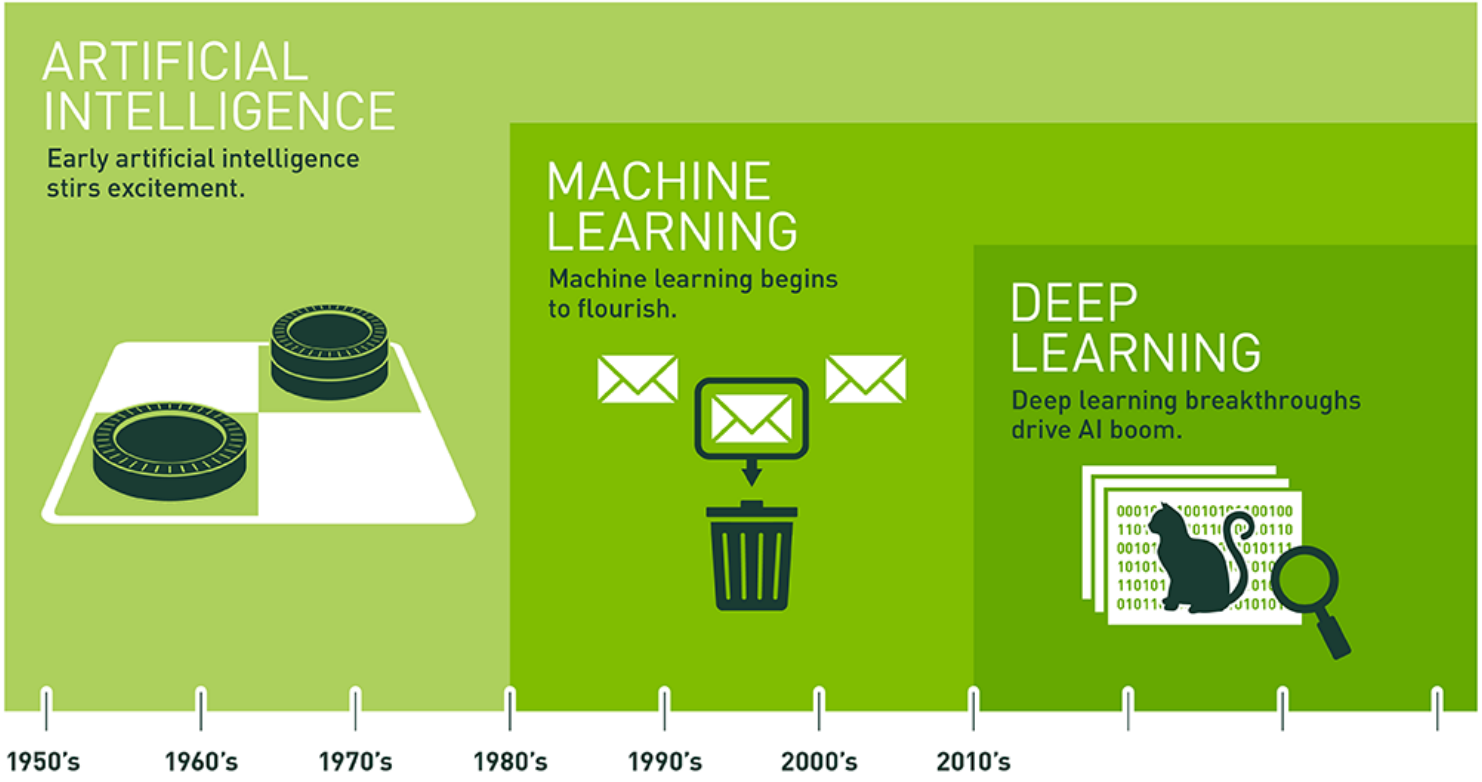
Deep Learning (DL) has emerged around the ‘10 as a general tool to solve recognition problems in:

- 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 = $\{c_1, c_1, ..., c_n\}$:= set of configurations, each one of them representing a possible solution to P

f:C→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* $x_1,x_2,...,x_N$ *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!)

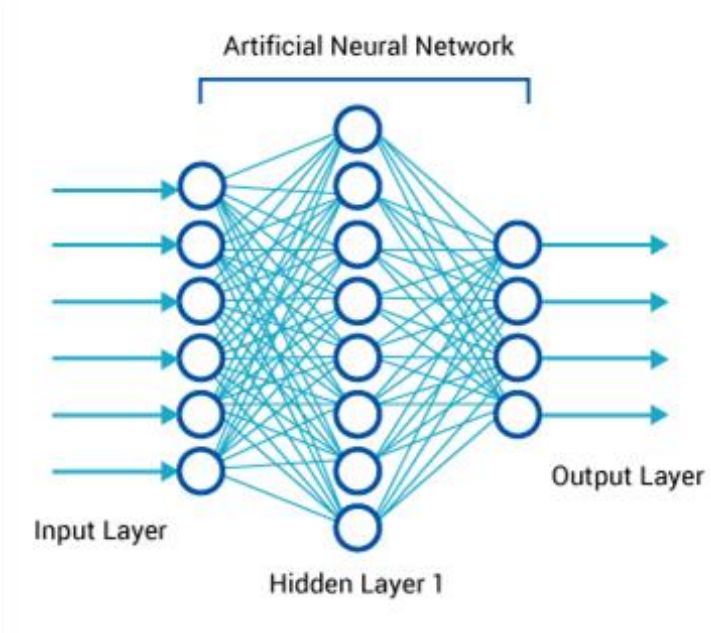
The set of configurations C give a space to explore (very often, a manifold)

Optimizing means to explore the manifold by iterative approaches (e.g., the gradient descent family of strategies)

The more the manifold is complex (non concave, 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*

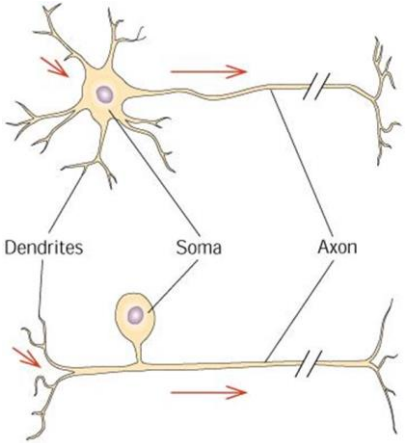


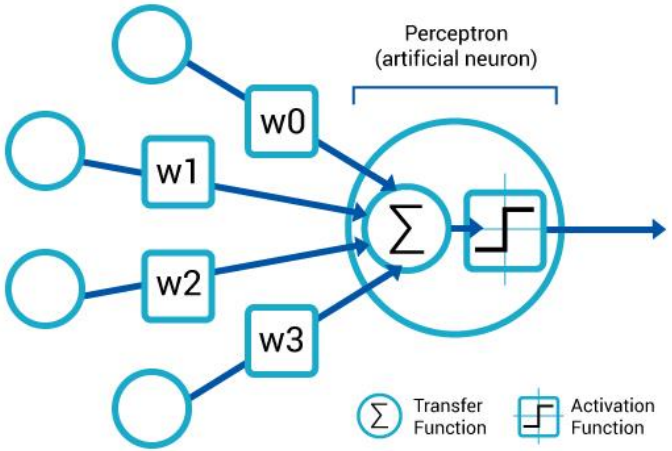
Neural Networks - Neurons

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

Each neuron has:

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

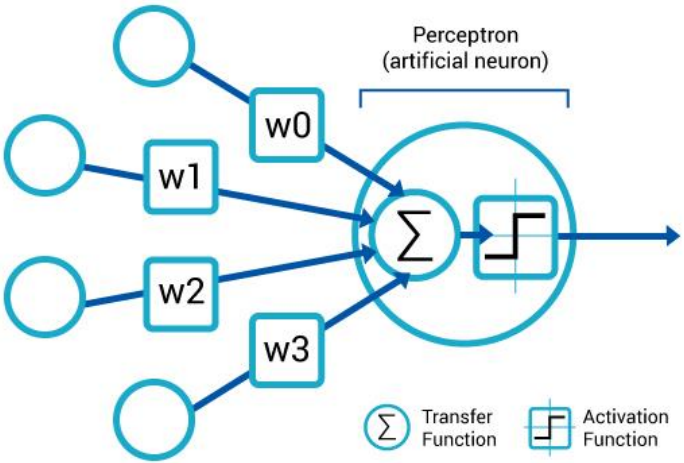


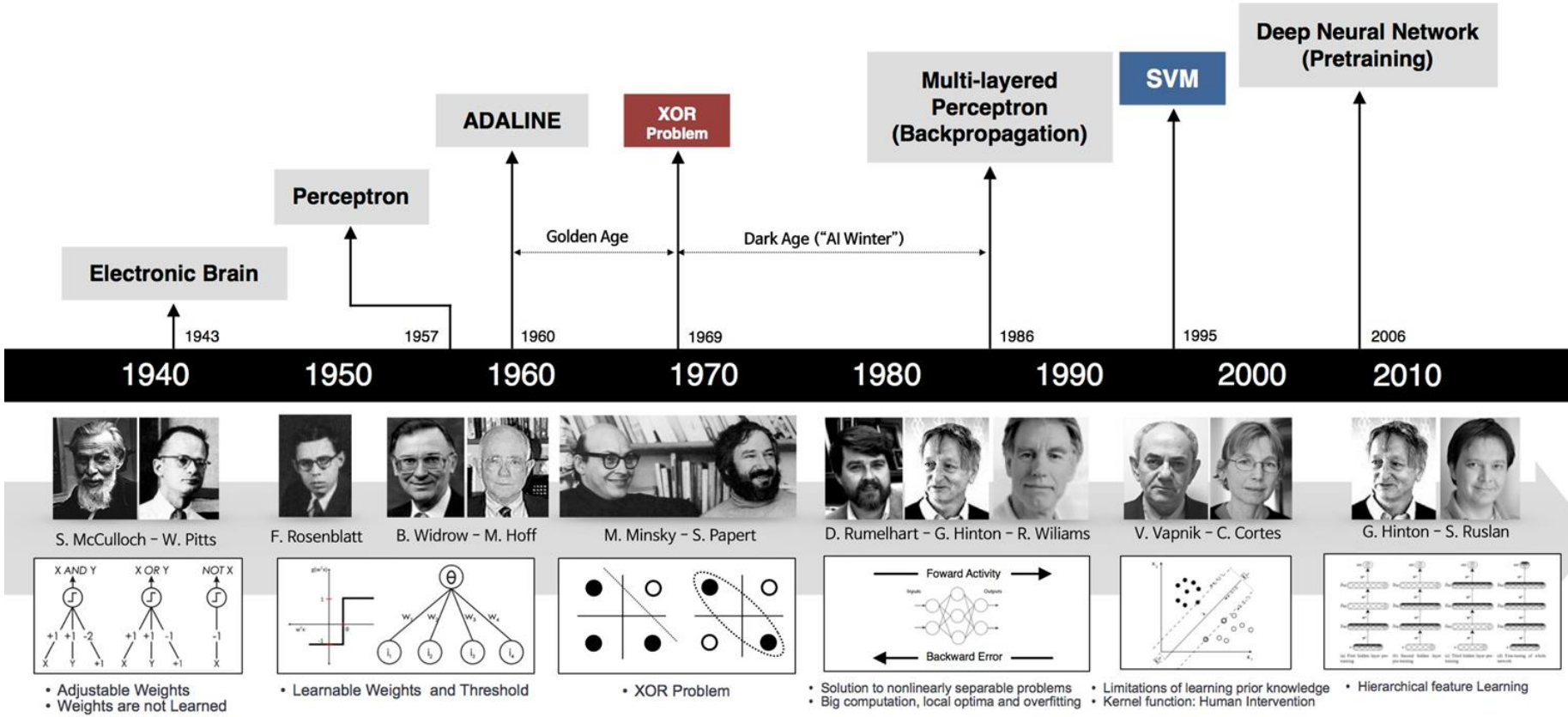


Neural Networks - Neurons (2)

The information flow is **unidirectional**:

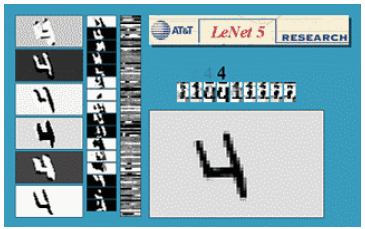
- The neuron get **inputs** (electric potentials) from the *dendrites*, that weight them (w_i's)
- In the *nucleus*, the weighted inputs are summed together (the **transfer** Σ of the whole information coming from the dendrites)
- In the *nucleus*, the summation flows into an **activation function**,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




Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
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

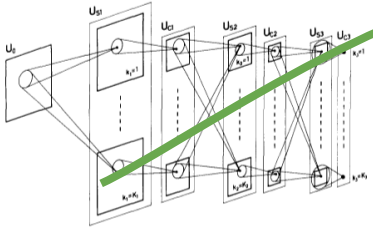
Neural Networks - the renaissance

Le Cun's
LeNet-5



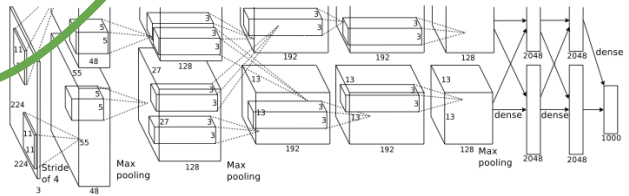
What happened?

Fukushima's
Neocognitron



...

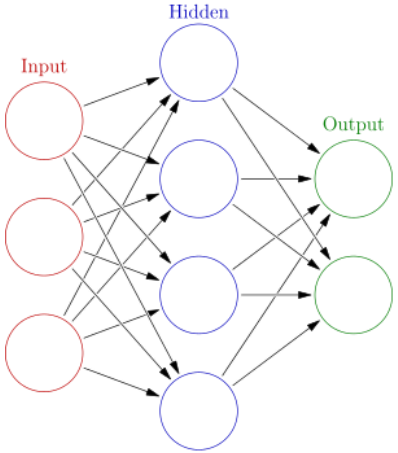
Krizhevsky's
AlexNet

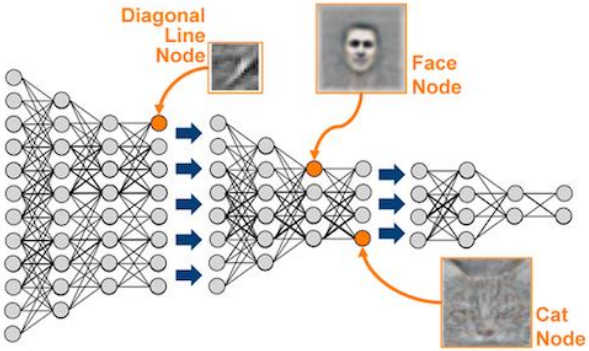


1980 1990 2000 2010 2020

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

Neural Networks and Deep Learning





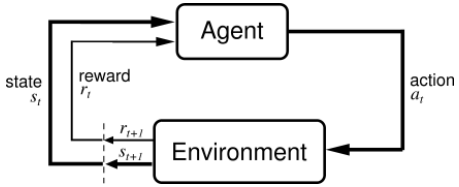
Neural Networks

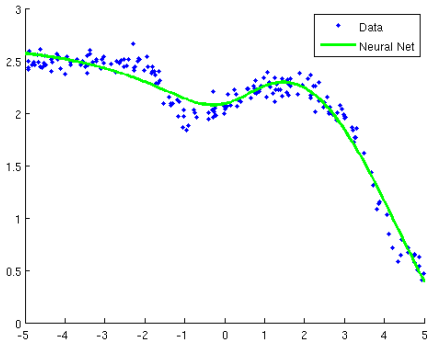
GPUs + Data

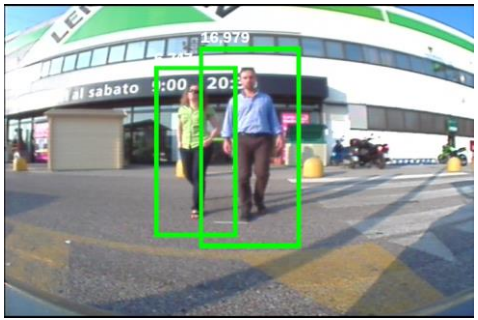
Deep Neural Networks

Supervised Classification

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





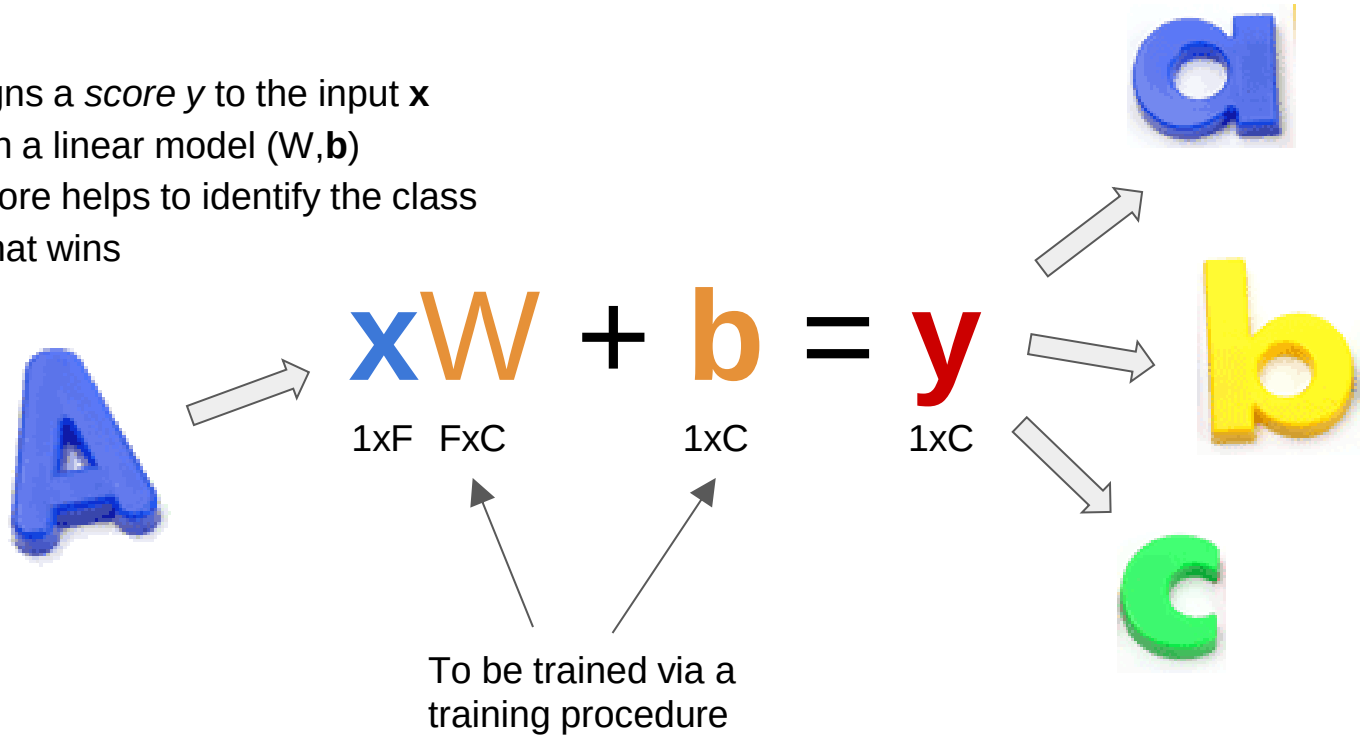


Labels {‘a’, ‘b’, ‘c’, ‘d’, ‘e’}

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



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

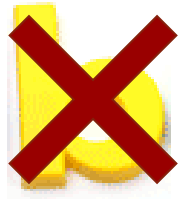
$$xW + b = y \Rightarrow s(y)$$

$$\begin{pmatrix} 2.0 \\ 1.0 \\ 0.1 \end{pmatrix}$$

scores

$$\begin{pmatrix} 0.7 \\ 0.2 \\ 0.1 \end{pmatrix}$$

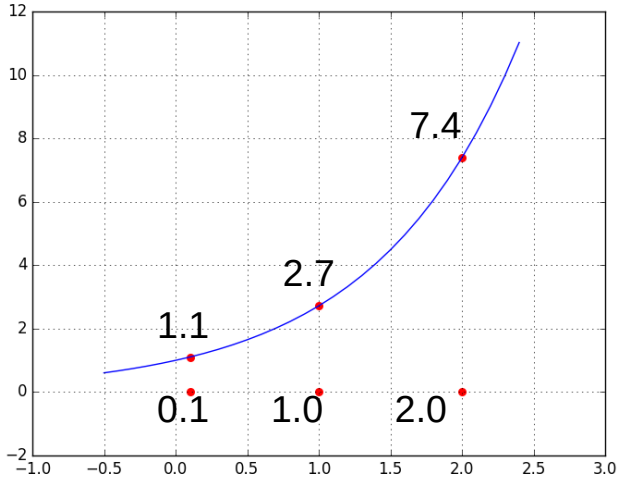
probabilities



$s(y)$ is the **SOFTMAX** function

Softmax function

- Converts scores into probability distributions
  - $\mathbb{R} \rightarrow (0,1)$
  - *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

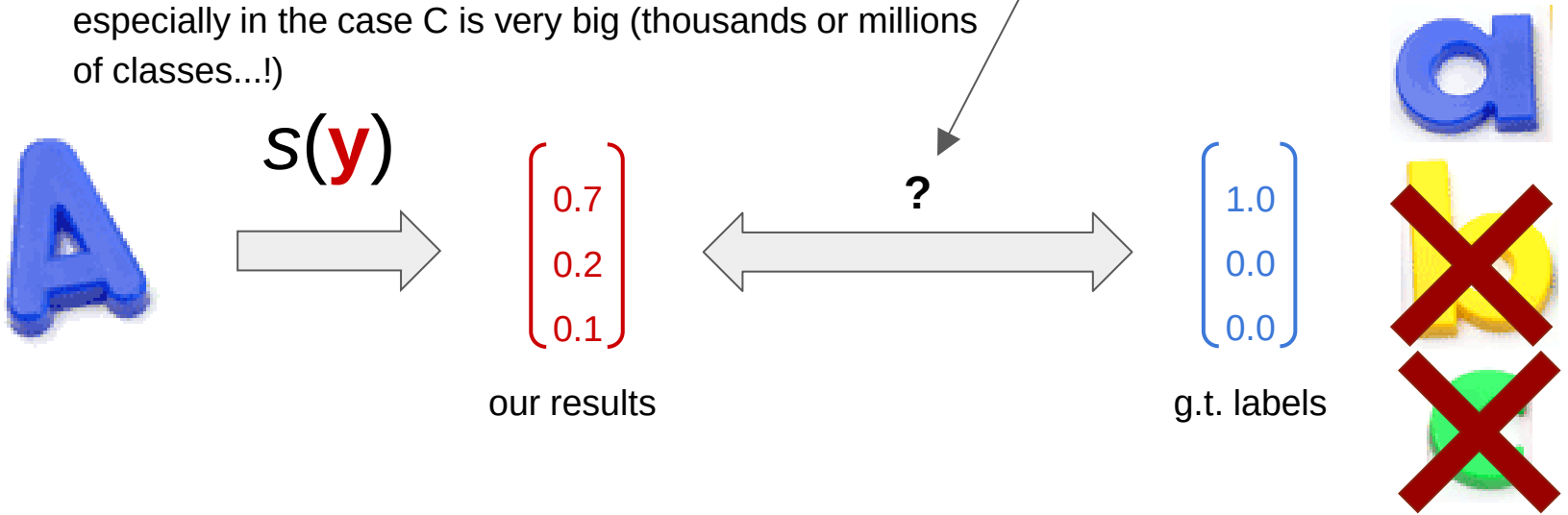
One-Hot Encoding (OHE)

- → *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 with 0's
- OHE needs sparse representation which is inefficient especially in the case C is very big (thousands or millions of classes...!)

Which distance measure?

MSE

Cross-Entropy



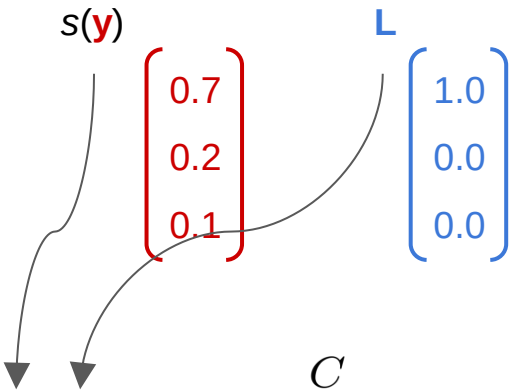
Cross-Entropy

GOAL: computes the distance between two probability vectors

- Non symmetric function

$$D(\mathbf{s}, \mathbf{L}) \neq D(\mathbf{L}, \mathbf{s})$$

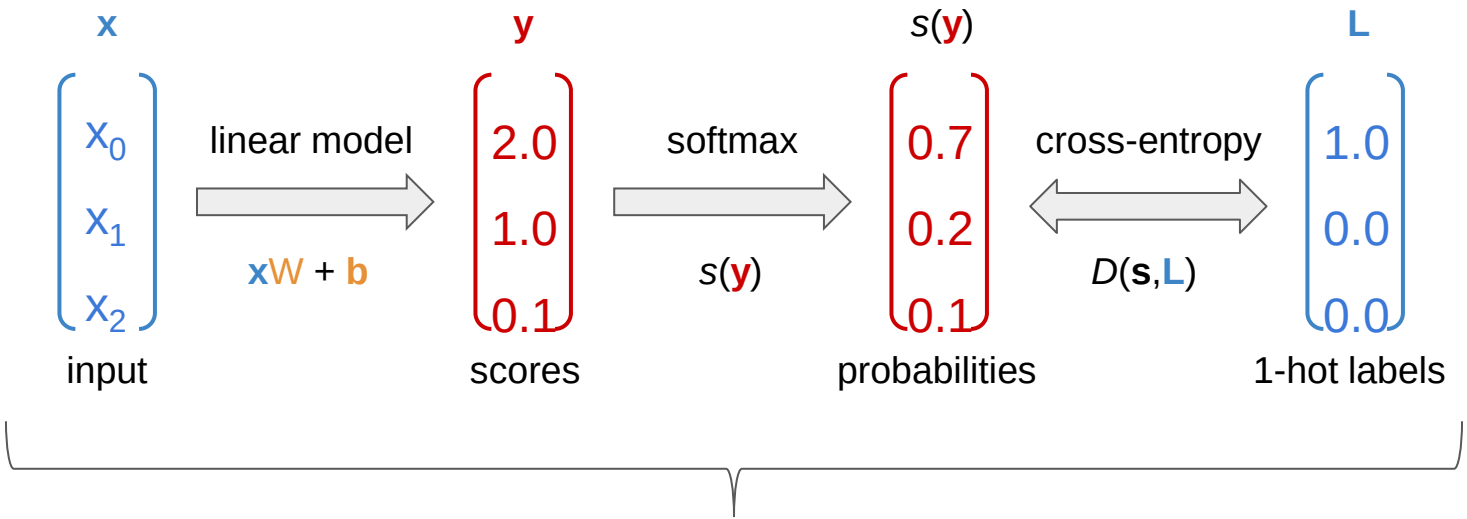
$D(\mathbf{s},\mathbf{L})$ $\begin{cases} \to 0 & \text{high similarity} \\ \to \infty & \text{low similarity} \end{cases}$



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

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

Résumé



Multinomial Logistic Classification

$$D(s(xW+b), L)$$

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

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

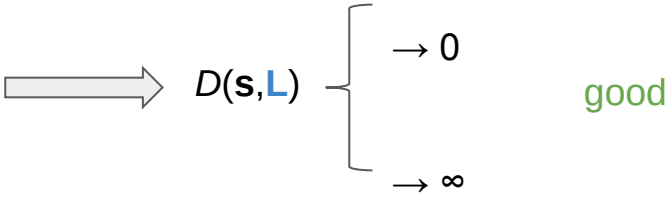
θ *= set of parameters*

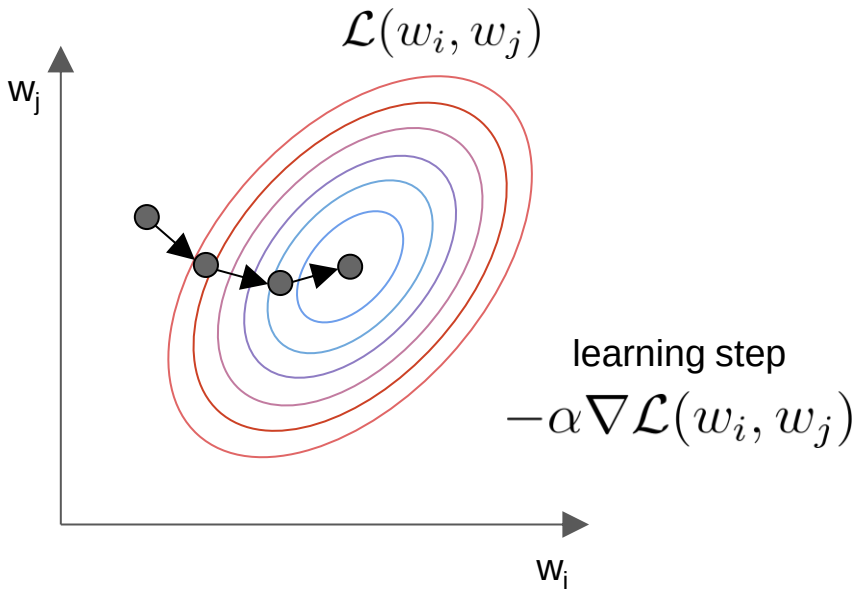
Gradient Descent

- GOAL: minimize a loss function
- 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





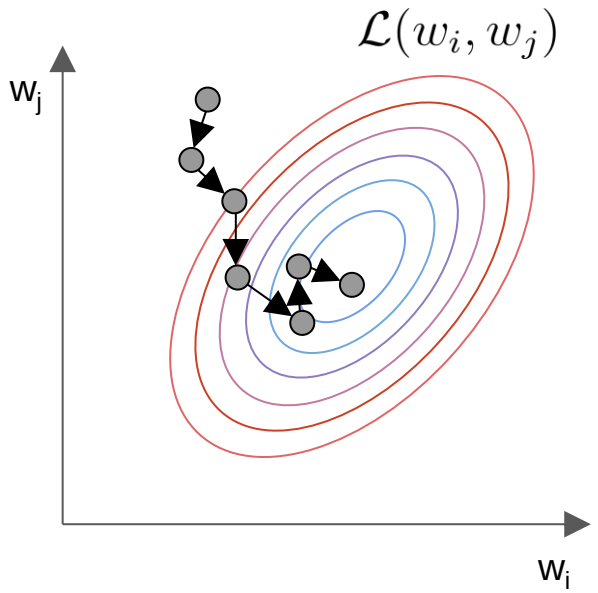
***N** = number of input vectors; **α**: learning rate value*

Stochastic Gradient Descent (SGD)

- IDEA: 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
    - bad approximation of the loss
    - suffers from local minima
    - requires data preprocessing to avoid numerical instability



… but tricks to ameliorate SGD are present!

SGD trick 1: momentum

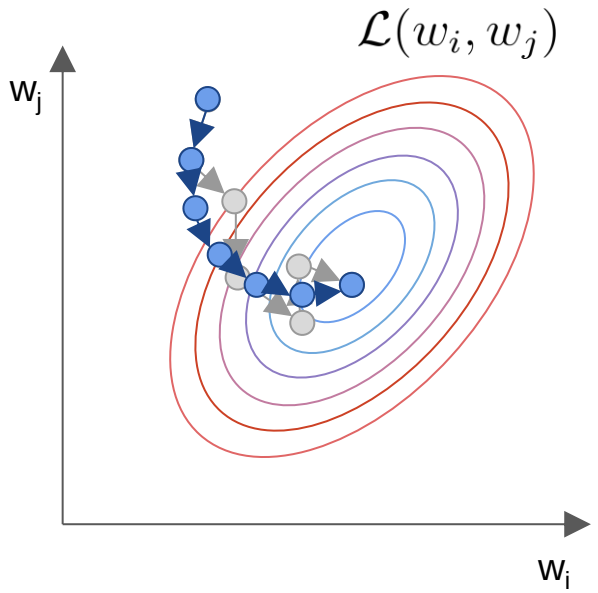
- GOAL: improve the convergence of the optimizer exploiting the accumulated knowledge from previous steps

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

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

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

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

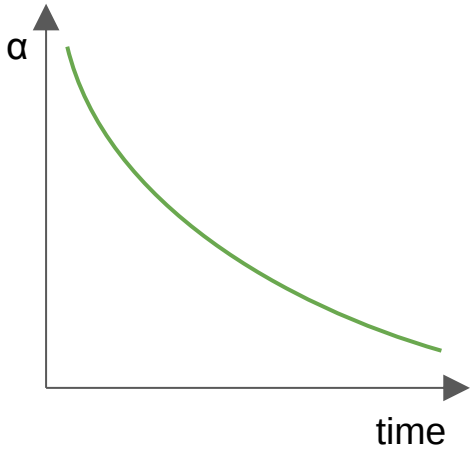


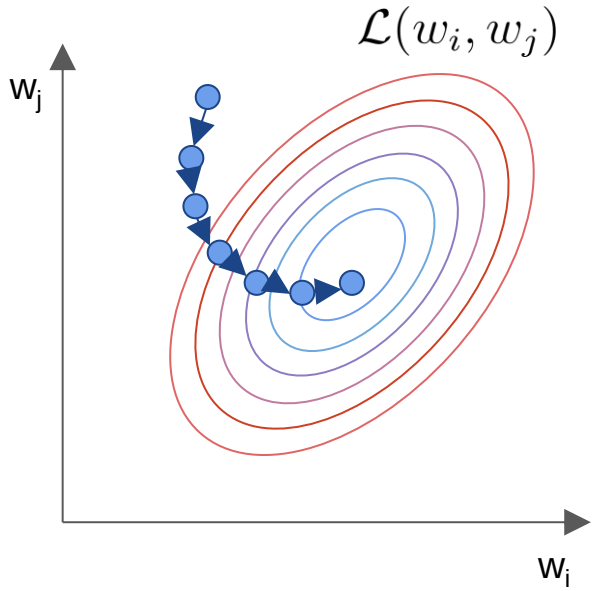
faster convergence and oscillation reduction

***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

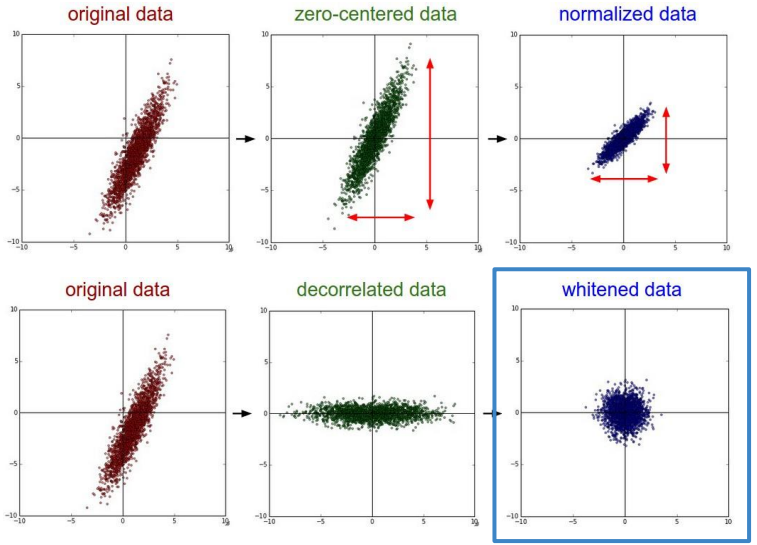
- GOAL: avoid numerical instability

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

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

z-normalization (*whitening*)

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



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

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

http://cs231n.github.io/

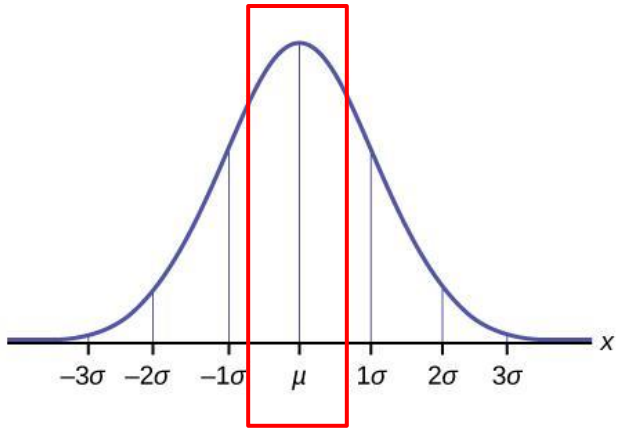
SGD trick 4: initialization

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.

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

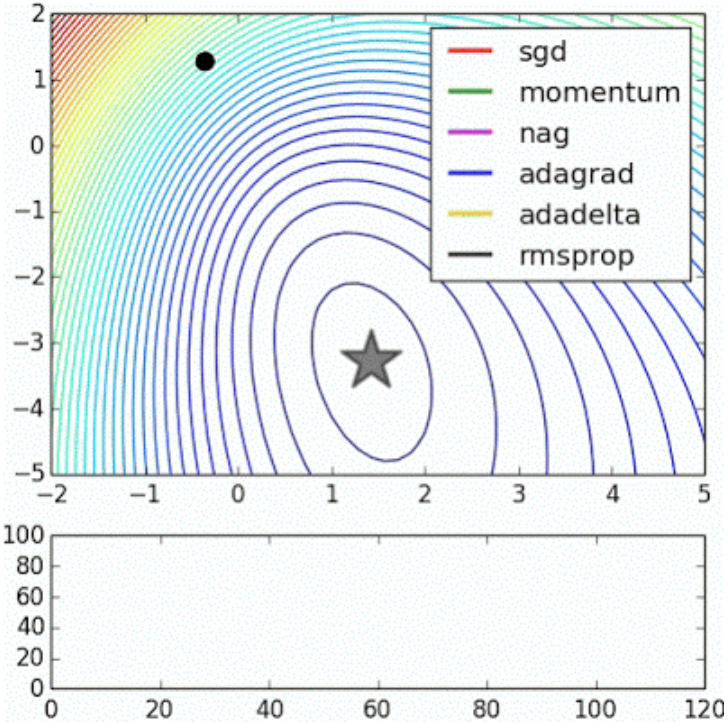
$$\mathbf{b} = \begin{bmatrix} 0_1 & 0_2 & \dots & 0_C \end{bmatrix}$$

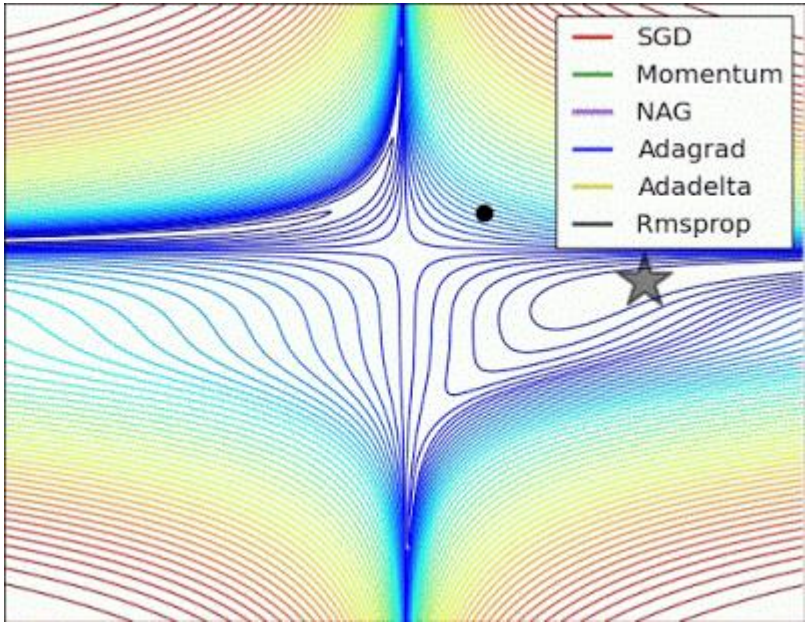
$$\mu = 0 \wedge \sigma \to 0 \implies \forall (i,j), w_i = w_j \pm \epsilon$$



equal probability
of the weights
(no prior)

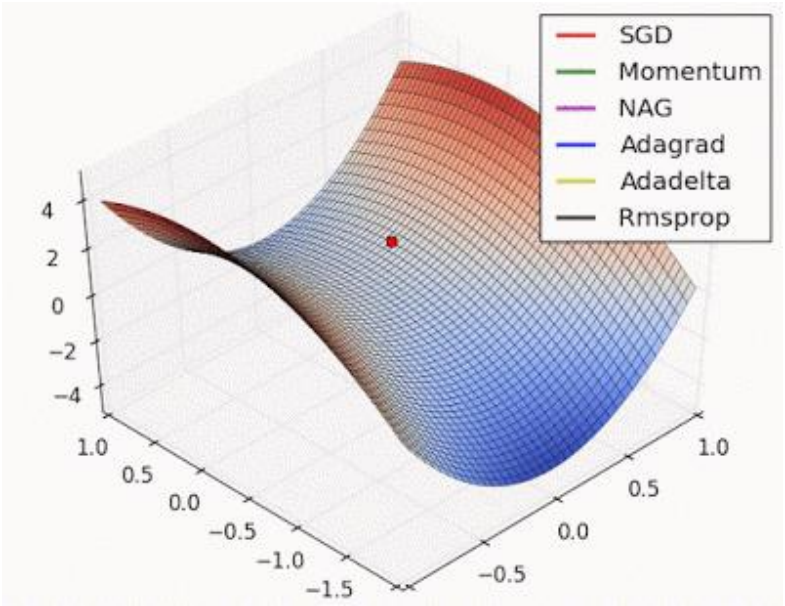
Gradient Descent: graphical representation (2D)

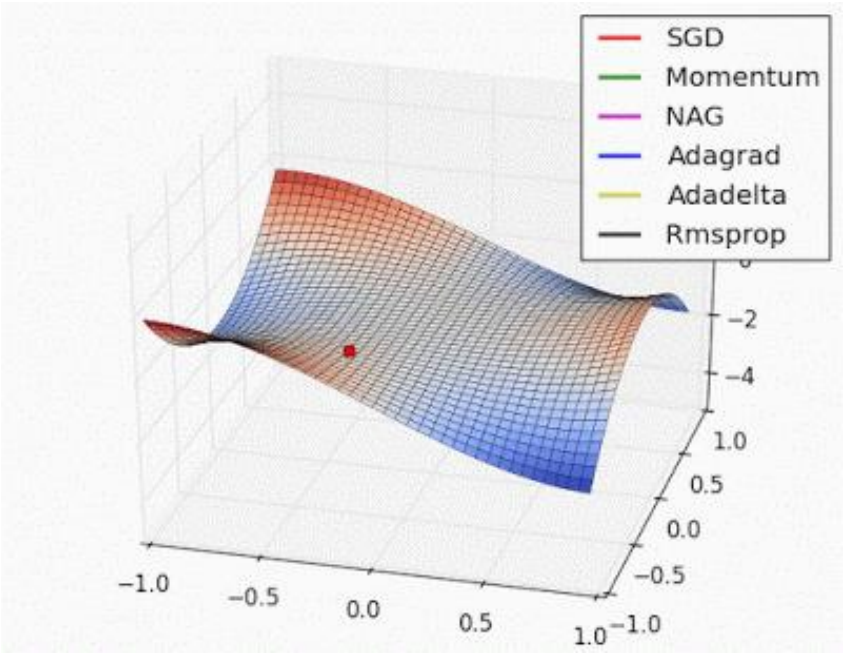




http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

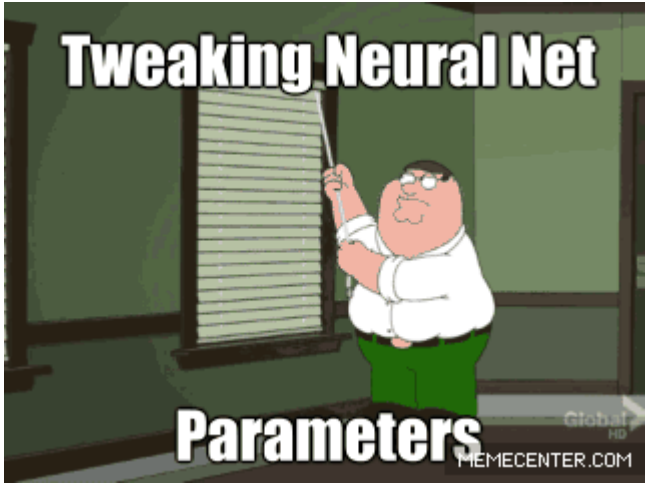
Gradient Descent: graphical representation (3D)





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:

- batch size
- weight initialization

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.