Introduction to Machine Learning & Deep Learning - part 1

Sorbonne Université – Master DAC- Master M2A. Patrick Gallinari
patrick.gallinari@sorbonne-universite.fr,
https://pages.isir.upmc.fr/gallinari
2023-2024

Course Outline and Organization

- Introductory ML course with a focus on Neural Networks and Deep Learning
- Organization
  - Courses 14 x 2 h – P. Gallinari
  - Practice and exercises 14 x 2 h
- Outline
  - Introduction
    - Basic Concepts of Machine Learning
  - Neural Networks and Deep Learning
    - Introductory Concepts - Perceptron-Adaline
    - Linear Regression and Logistic Regression - Optimization Basics
    - Multilayer Perceptrons – Generalization Properties
    - Convolutional Neural Networks – Vision applications
    - Recurrent Neural Networks – Language applications
    - Transformers and attention models – Language applications
  - Kernel machines
    - Support Vector Machines
    - Gaussian processes
  - Neural processes and meta-learning

Ressources

- Mattermost Channel
  - To come
- Books
  - The following two books cover the course (more or less)
    - Deep Learning, An MIT Press book, I. Goodfellow, Y. Bengio and A. Courville, 2017
      - http://www.deeplearningbook.org/
    - Pattern recognition and Machine Learning, C. Bishop, Springer, 2006
      - Chapters3, 4, 5, 6, 7, 9,
  - Many other books can be profitable, e.g.
    - The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition, T. Hastie, R. Tibshirani, J. Friedman, Springer, 2009
      - Version pdf accessible : http://statweb.stanford.edu/~tibs/ElemStatLearn/
    - Bayesian Reasoning and Machine Learning, D. Barber, Cambridge University Press, 2012
      - Version pdf accessible : http://www.cs.ucl.ac.uk/staff/d.barber/brml/
- Courses
  - Several on line ressources, covering this topic and others
    - Course slides and material: Machine Learning, Deep Learning for Vision, Natural Language Processing, …
    - MOOCS: e.g. Andrew Ng ML course on Coursera
- Software Platforms
  - … introduced in the practice sessions

Machine Learning General Framework

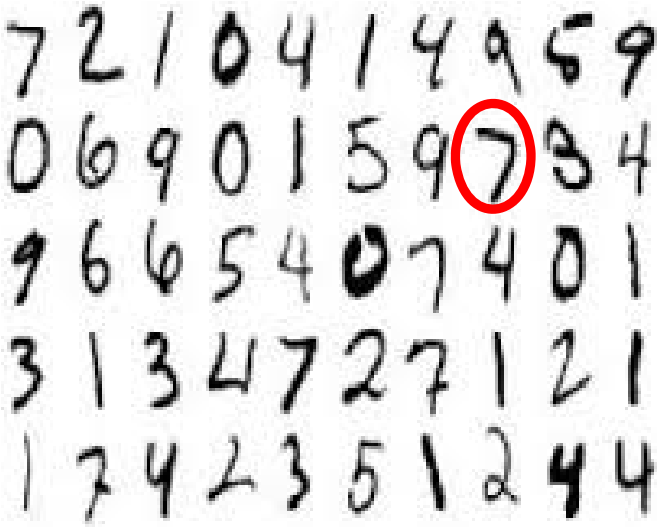
- 4 learning problems
- Risk, Empirical Risk

4 learning problems

- ML develops generic methods for solving different types of problems
- Typical classification of ML problems:
  - Supervised
  - Unsupervised
  - Semi-supervised
  - Reinforcement

4 learning problems
Supervised learning

- Training set: couples (inputs, target) $(\boldsymbol{x}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{x}^N, \boldsymbol{y}^N)$
- Objective : learn to associate inputs to outputs
  - With good generalization properties
- Classical problems: classification, regression, ranking



seven

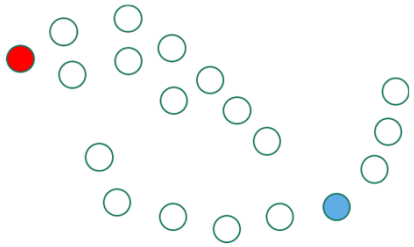
- Most applications today fall under the supervised learning paradigm

4 learning problems
Unsupervised learning

- Training set
  - Only input data $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N$, no target
- Objective
  - Extract some regularities from data
    - Similarities, relations between items, latent factors explaining data generation
- Use
  - Density estimation, clustering, latent factors identification, generative models

4 learning problems
Semi-supervised learning

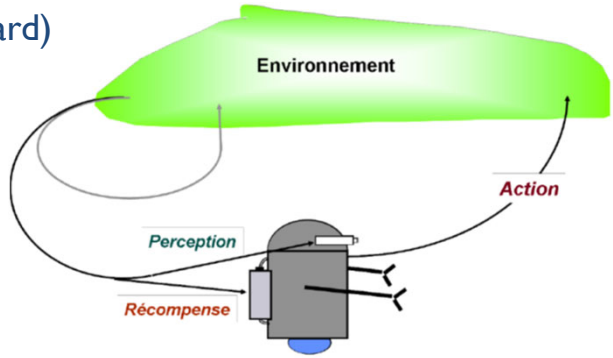
- Task
  - Similar to supervised learning
- Training set
  - Small number of labeled data $(\boldsymbol{x}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{x}^N, \boldsymbol{y}^N)$
  - Large number of unlabeled data $\boldsymbol{x}^{N+1}, \ldots, \boldsymbol{x}^{N+M}$
- Objective
  - Extract information from unlabeled data useful for labeling examples
    - e.g. structure
  - Joint learning from the two datasets



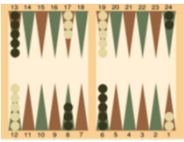
- Use
  - When large amounts of data are available and labeling is costly

4 learning problems
Reinforcement learning

- Training set
  - Couples (input, qualitative target)
  - $x^i s$ may be sequences (temporal credit assignment), y^i are qualitative targets (e.g. $0,1$), deterministic or stochastic
- Paradigm
  - Learning by exploring the environment, using reinforcement signals (reward)
  - Exploration/ exploitation paradigm



- Use
  - command, sequential decision, robotis, two players game, dynamic programming, …
  - RL for games
    - Backgammon (TD Gammon Thesauro 1992)
    - Trained on 1.5 M plays
    - Plays against itself
  - Deep RL
    - AlphaGo (2015), AlphaGo Zero (2017)
    - Alphazero (2017)



Risk – Empirical Risk
Probabilistic formalism

- Data
  - Random vectors $(\boldsymbol{z})$ generated from distribution $p(\boldsymbol{z})$
- Learning model
  - $\boldsymbol{F} = \{F_\theta\}_\theta$ with θ the model parameters, usually real parameters
- Loss
  - $c_\theta(\boldsymbol{z})$ for model F_θ and example $\boldsymbol{z}$
- Risk
  - $R_\theta = E_{\boldsymbol{z}}[c_\theta(\boldsymbol{z})] = \int_z c_\theta(\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$
- Optimal solution
  - $F_{\theta^*} = argmin_\theta R_\theta$

Risk – Empirical Risk
Learning from examples

- Data
  - $D = \{\boldsymbol{z}^i\}_{i=1..N}$
- Empirical risk
  - $C = \frac{1}{N}\sum_{i=1}^{N} c_\theta(\boldsymbol{z}^i)$
- Empirical risk minimization principle
  - F_{θ^*} minimizing the theoretical risk is approximated by $F_{\hat{\theta}}$ mimizing the empirical risk
  - Is that sufficient ? Answer is No
- Inductive framework
  - We will consider the following ML framework
    - The model learns on an available training set
    - Once trained parameters are fixed and the model can be used for inference and/or evaluated on a test set

Example of generic ML problems

- Classification
  - $\boldsymbol{z} = (\boldsymbol{x}, y), y \in \{0,1\}$
  - F_θ threshold functions
  - R : probability of incorrect classification
  - C : error frequency

$$c_\theta(\boldsymbol{z}) = \begin{cases} 0 \text{ if } y = F_\theta(\boldsymbol{x}) \\ 1 \text{ otherwise} \end{cases}$$

- Regression
  - $\boldsymbol{z} = (\boldsymbol{x}, y), y \in R$
  - F_θ real functions (e.g. linear NNs)
  - R : expectation of quadratic error
  - C : sum of quadratic errors

$$c_\theta(\boldsymbol{z}) = \|y - F_\theta(\boldsymbol{x})\|^2$$

- Density estimation
  - $\boldsymbol{z} = \boldsymbol{x}$
  - F_θ real functions
  - R : likelihood (expectation)
  - C : empirical estimator of likelihood (sum)

$$c_\theta(\boldsymbol{z}) = -ln p_\theta(\boldsymbol{x})$$

Neural Networks and Deep Learning

Context

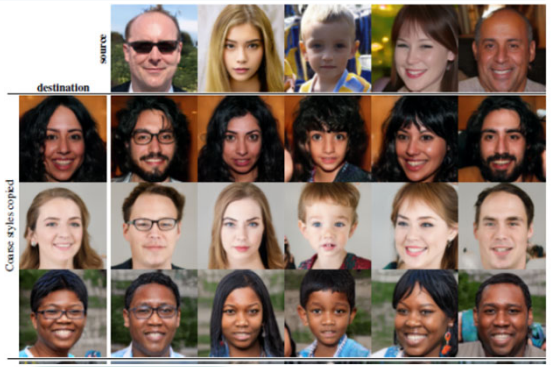
Context
Deep Learning today

- Deep Learning is today the most popular paradigm in data science
- Popularized since 2006, first by some academic actors and then by big players (GAFAs, BATs, etc)
- It has initiated a « paradigm shift » in the field of data science / AI and definitely changed the way one will exploit data
  - e.g. key players have made available development platforms (initiated e.g.with TensorFlow, PyTorch)
    - Allowing the development in a « short time » of complex processing chains
    - Making complex DL methods available for a large community
- Today DL is developing at a much larger scale, including
  - Software development platforms and environments
  - Services in multiple domains: biotech, health, finance, client management, etc

Machine Learning successes

- Mainly concern the numerical world and GAFAs/BATs applications
  - Semantic data analysis: vision, speech, language, traces;
  - Virtual worlds, e.g. games

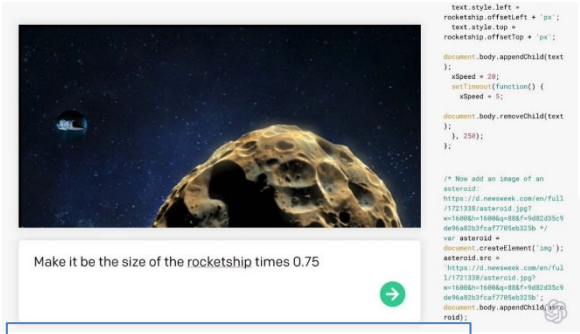
Generative models - (Karras et al. 2019) – Style GAN - NVIDIA



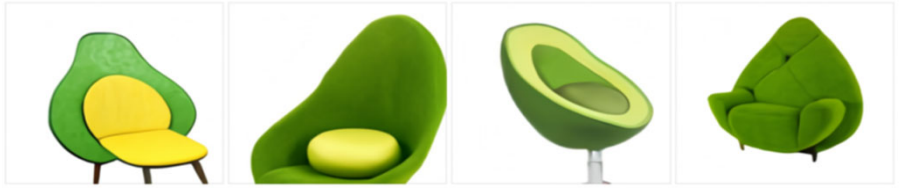
Generative models 2022 Stable-diffusion https://stablediffusionweb.com/

Alphastar, Vinyals et al. 2019 (Starcraft) - Deepmind

ChatGPT 2022



OpenAI-Codex 2021
natural language to code



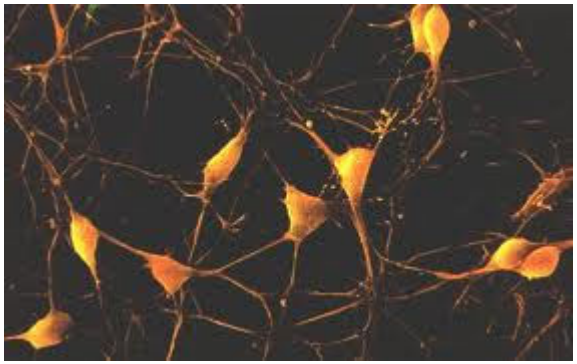
DALL.E - 2021 https://openai.com/blog/dall-e/
Text: an armchair in the shape of an avocado. . . .

Introductory NN concepts

Intuitive introduction via 2 simple –historical- models

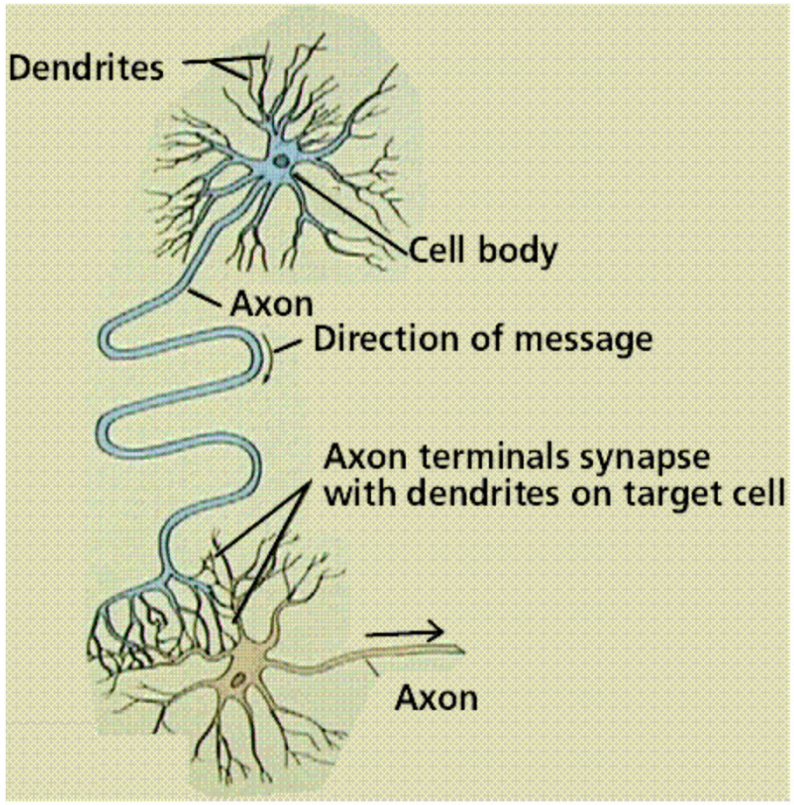
Perceptrons and Adalines

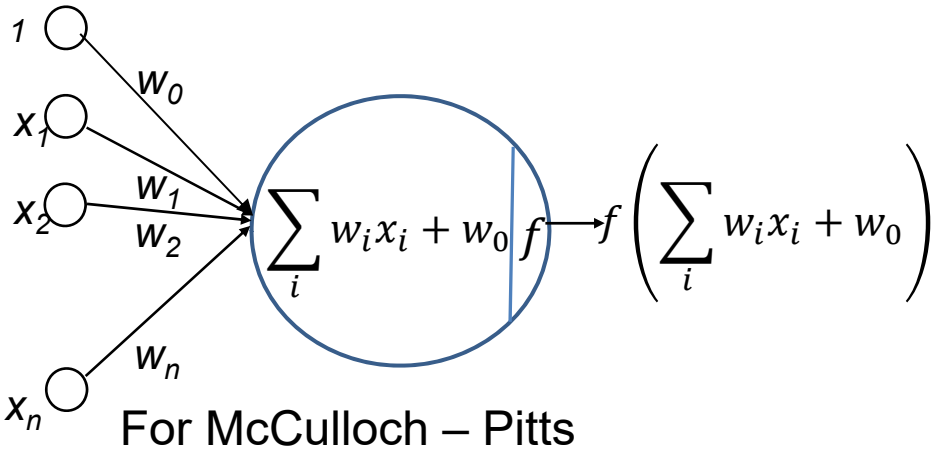
Neural Networks inspired Machine Learning
Brain metaphor



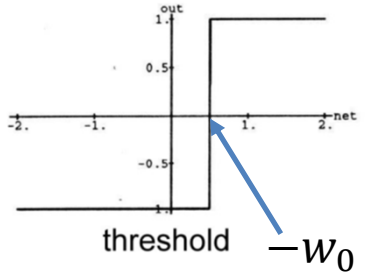
- Artificial Network Networks are an important paradigm in Statistical Machine learning and Artificial Intelligence
- Human brain is used as a source of inspiration and as a **metaphor** for developing Artificial NN
  - Human brain is a dense network 10^{11} of simple computing units, the neurons. Each neuron is connected – in mean- to 10^{4} neurons.
  - Brain as a computation model
    - Distributed computations by simple processing units
    - Information and control are distributed
    - Learning is performed by observing/ analyzing huge quantities of data and also by trials and errors

Formal Model of the Neuron
McCulloch – Pitts 1943





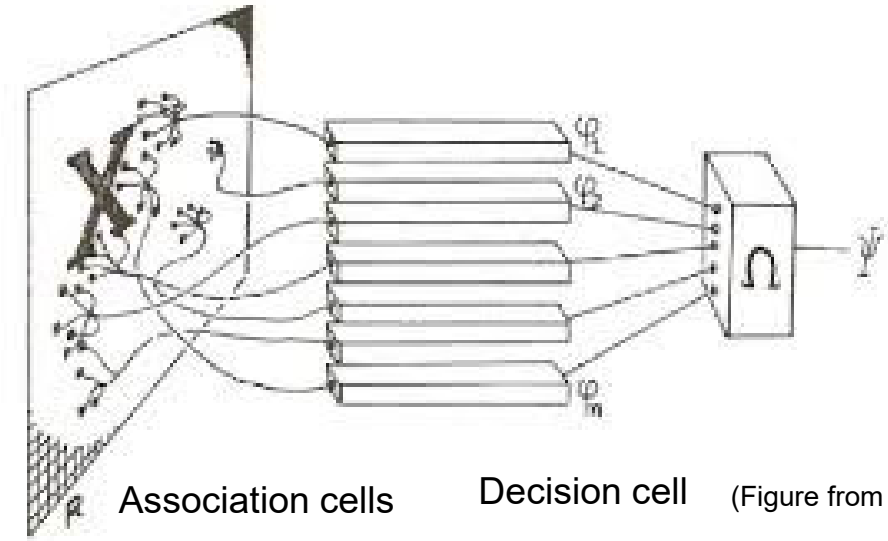
For McCulloch – Pitts neuron, f is a threshold (sign) function



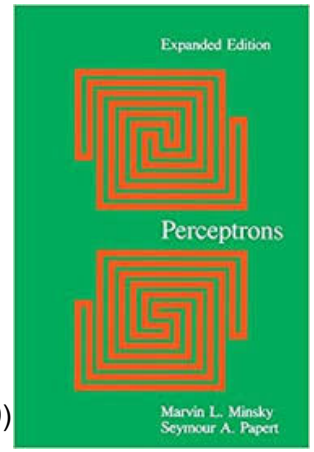
$$f(x) = \begin{cases} 1 \; if \sum_i (w_i x_i + w_0) > 0 \\ -1 \; otherwise \end{cases}$$

A synchronous assembly of neurons is capable of universal computations (aka equivalent to a Turing machine)

Perceptron (1958 Rosenblatt)

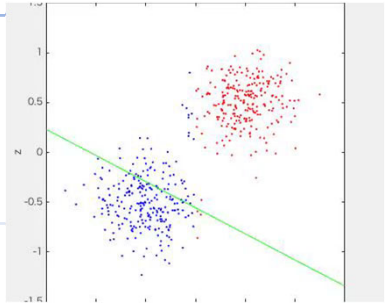


(Figure from Perceptrons, Minsky and Papert 1969)



- The decision cell is a threshold function (McCulloch – Pitts neuron)
  - $F(\boldsymbol{x}) = sgn(\sum_{i=1}^{n} w_i x_i + \mathrm{w}_0)$
- This simple perceptron can perform 2 classes classification

Perceptron Algorithm (2 classes)



Data

Labeled Dataset $\{(\boldsymbol{x}^i, y^i), i = 1..N, \boldsymbol{x} \in R^n, y \in \{-1,1\}\}$

Output

classifier $\boldsymbol{w} \in R^n$, decision $F(\boldsymbol{x}) = sgn(\sum_{i=0}^{n} w_i x_i)$

Training set
Classifier specificatio

Initialize $\boldsymbol{w}$ (0)

Repeat (t)

Choose an example $(\boldsymbol{x}(t), y(t))$

if $y(t)\boldsymbol{w}(t).\boldsymbol{x}(t) \leq 0$ then $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \epsilon y(t)\boldsymbol{x}(t)$

Stochastic Algorithm

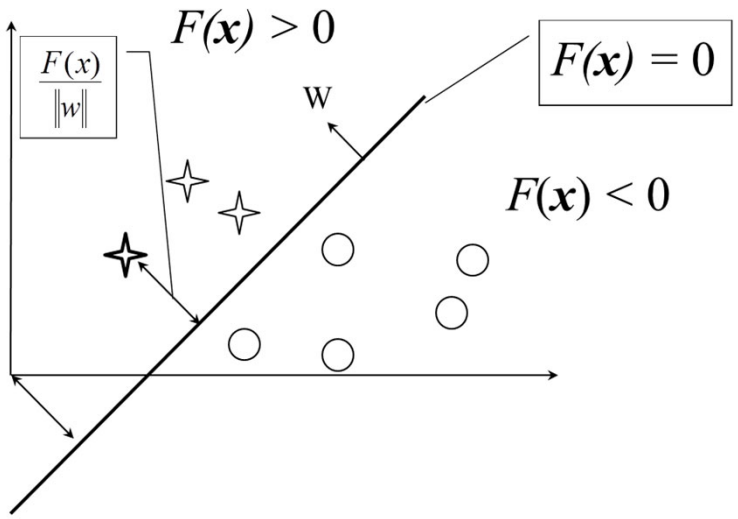
Until convergence

- The learning rule is a **stochastic gradient algorithm** for minimizing the number of wrongly predicted labels
- Multiple (p) classes: p perceptrons in parallel, 1 class versus all others!

Linear discriminant function

$$F(\boldsymbol{x}) = \boldsymbol{w}.\boldsymbol{x} + w_0 = \sum_{i=0}^{n} w_i x_i \text{ with } x_0 = 1$$

- Decision surface : hyperplane $F(\boldsymbol{x}) = 0$
- Properties
  - $\boldsymbol{w}$ is a normal vector to the hyperplane, it defines its orientation
  - distance from x to H : $r = F(\boldsymbol{x})/\|\boldsymbol{w}\|$
  - if $w_0 = 0$ H goes through the origin



Perceptron algorithm performs a stochastic gradient descent

- Loss function
  - $C = -\sum_{(x,y)\text{missclassified}} \boldsymbol{w}.\boldsymbol{x}y = -\sum_{(x,y)\text{miss-classified}} \text{c}(\boldsymbol{x}, \text{y})$
  - Objective : minimize C
- gradient
  - $grad_{\boldsymbol{w}} C = \left(\frac{\partial C}{\partial w_1}, \dots, \frac{\partial C}{\partial w_n}\right)^T$ with $\frac{\partial C}{\partial w_i} = -\sum_{(x,d)\text{missclassified}} \boldsymbol{x}y$
- Learning rule
  - Stochastic gradient descent for minimizing loss C
  - Repeat (t)
    - Choose an example $(\boldsymbol{x}(t), y(t))$
    - $\boldsymbol{w}(\text{t}) = \boldsymbol{w}(\text{t}-1) - \epsilon\, grad_{\boldsymbol{w}} \text{c}(\boldsymbol{x}, \text{y})$

Multi-class generalization

- Usual approach: one vs all
  - p classes = p " 2 class problems " : class C_i against the others
    - Learn p discriminant functions $F_i(\boldsymbol{x}), i = 1 \ldots p$
    - Decision rule: $\boldsymbol{x} \in Ci$ if $F_i(\boldsymbol{x}) > F_j(\boldsymbol{x})$ for $j \neq i$
    - This creates a partition of the input space
    - Each class is a polygon with at most $p - 1$ faces.
  - Convex regions: limits the expressive power of linear classifiers

Perceptron properties (1958 Rosenblatt)

- **Convergence** theorem (Novikof, 1962)
  - Let $D = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$ a data sample. If
    - $R = \max_{1 \le i \le N} \|\boldsymbol{x}^i\|$
    - $\sup_{\boldsymbol{w}} \min_i y^i (\boldsymbol{w}.\boldsymbol{x}^i) > \rho$ (ρ is called a margin)
    - The training sequence is presented a sufficient number of time
  - The algorithm will converge after at most $\left[\frac{R^2}{\rho^2}\right]$ corrections
- **Generalization** bound (Aizerman, 1964)
  - If in addition we provide the following stopping rule:
    - Perceptron stops if after correction number k, the next $m_k = \frac{1+2\ln k - \ln \eta}{-\ln(1-\epsilon)}$ data are correctly recognized
  - Then
    - the perceptron will converge in at most $l \le \frac{1+4\ln R/\rho - \ln \eta}{-\ln(1-\epsilon)} [R^2/\rho^2]$ steps
    - with probability $1-\eta$, test error is less than ϵ

Link between training and generalization performance

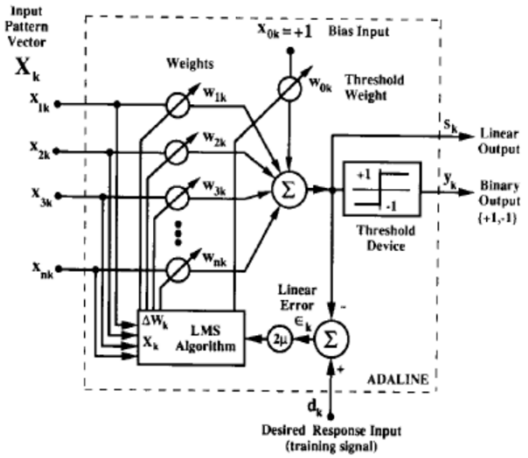
Convergence proof (Novikof)

- Hyp: lets take $w^* \;/\; \|w^*\| = 1$
  - $w_0 = 0, w_{t-1}$ is the weight vector before the t^{th} correction
  - $w_t = w_{t-1} + \epsilon y(t) x(t)$
  - $w_t . w^* = w_{t-1} . w^* + \epsilon y(t) x(t) . w^* \geq w_{t-1} . w^* + \epsilon \rho$
  - By induction $w_t . w^* \geq t \epsilon \rho$

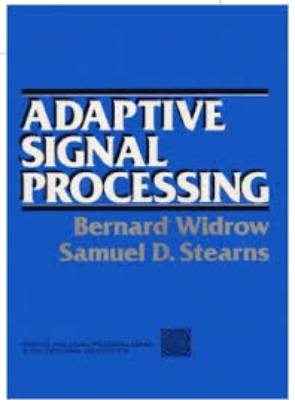
  - $\|w_t\|^2 = \|w_{t-1}\|^2 + 2\epsilon y(t) w_{t-1} . x(t) + \epsilon^2 \|x(t)\|^2$
  - $\|w_t\|^2 \leq \|w_{t-1}\|^2 + \epsilon^2 \|x(t)\|^2$ since $y(t) w_{t-1} . x(t) < 0$ (remember that $x(t)$ is incorrectly classified)
  - $\|w_t\|^2 \leq \|w_{t-1}\|^2 + \epsilon^2 \mathrm{R}^2$
  - By induction $\|w_t\|^2 \leq t \epsilon^2 R^2$

  - $t \epsilon \rho \leq w_t . w^* \leq \|w_t\| \|w^*\| \leq \sqrt{t} \epsilon R \|w^*\|$
  - $t \leq \frac{R^2}{\rho^2} \|w^*\|^2 = \frac{R^2}{\rho^2}$

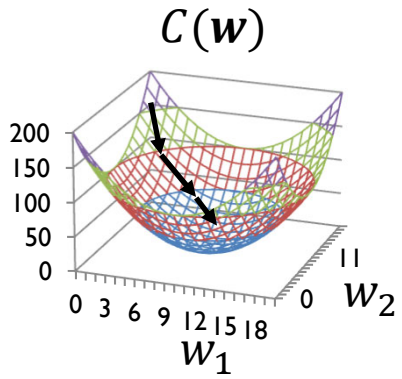
Adaline – Adaptive Linear Element (Widrow - Hoff 1959)



Linear unit: $F(x) = \sum_i \ w_i x_i + w_0$



- « Least Mean Square » LMS algorithm
  - Loss: $c(\boldsymbol{x}, \boldsymbol{y}) = \left|\left|\boldsymbol{y} - F(\boldsymbol{x})\right|\right|^2$
  - Algorithm: Stochastic Gradient Descent (Robbins – Monro (1951))
    - Initialize $w(0)$
    - Iterate
      - Choose an example $(\boldsymbol{x}(t), y(t))$
      - $w(t+1) = w(t) - \epsilon \nabla_w \, c(\boldsymbol{x}, \boldsymbol{y})$
  - Workhorse algorithm of adaptive signal processing: filtering, equalization, etc.



Adaline example motivating the need for adaptivity from an engineering perspective

- **Adaptive** noise cancelling

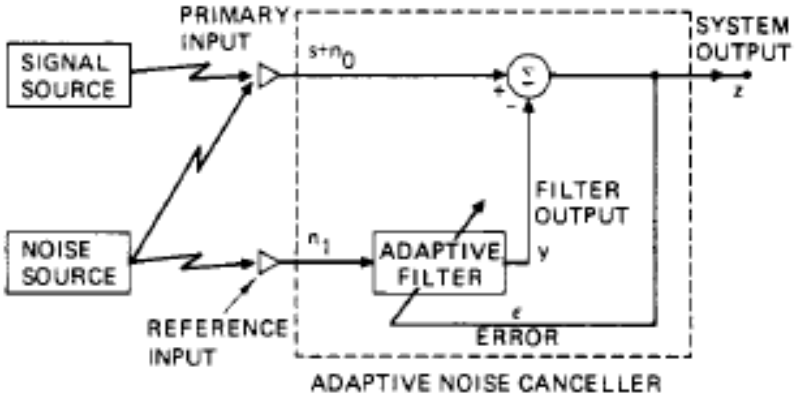


Fig. 1. The adaptive noise cancelling concept.

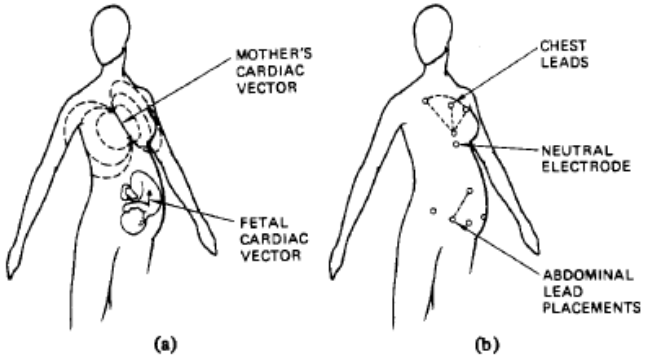


Fig. 14. Cancelling maternal heartbeat in fetal electrocardiography. (a) Cardiac electric field vectors of mother and fetus. (b) Placement of leads.

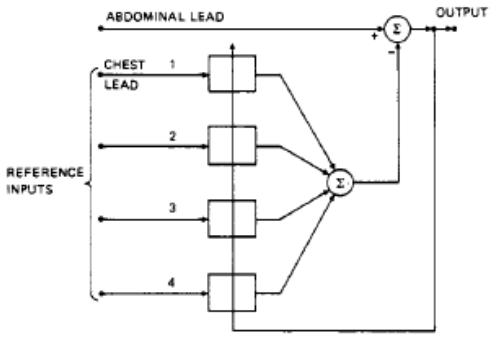


Fig. 15. Multiple-reference noise canceller used in fetal ECG experiment.

Fig. from Adaptive Signal Processing, Widrow, Stearn

Heartbeat cancelling

Objective: get z as close as possible to the baby signal s

Adaline – heartbeat cancelling detailed

- With the notations of the Figure
- Hyp.:
  - s, n_0, n_1, y are stationary with zero means
  - s is uncorrelated with n_0, n_1 and then y
- Filtering scheme
  - output $z = s + n_0 - y$
  - Loss function to be minimized $E[z^2]$
- Then
  - $z^2 = s^2 + (n_0 - y)^2 + 2s(n_0 - y)$
  - $E[z^2] = E[s^2] + E[(n_0 - y)^2] + 2E[s(n_0 - y)]$
  - $E[z^2] = E[s^2] + E[(n_0 - y)^2]$ since s and $(n_0 - y)$ are not correlated
- So that
  - $Min\ E[z^2] = E[s^2] + Min\ E[(n_0 - y)^2]$
- When the filter is trained to minimize $E[z^2]$, it also minimizes $E[(n_0 - y)^2]$
- Then y is the best LMS estimate of n_0, and z is the best LMS estimate of signal s (since $z - s = n_0 - y$)

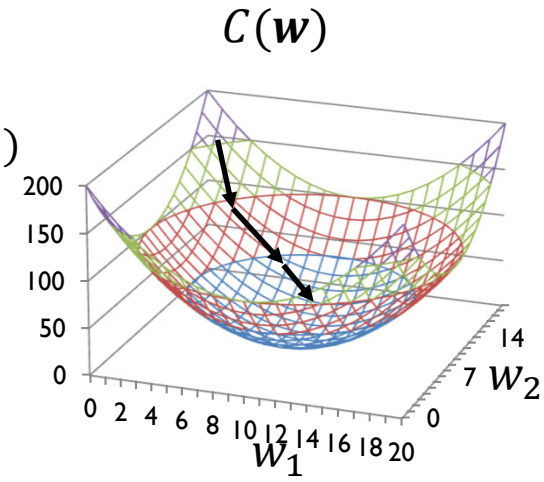
Introductory concepts
Summary of key ideas

- Learning from examples
  - Perceptron and Adaline are supervised learning algorithm
  - Training and test set concepts
    - Parameters are learned from a training set, performance is evaluated on a test set
    - Supervised means each example is a couple $(\boldsymbol{x}, \boldsymbol{y})$
- Stochastic optimization algorithms
  - Training requires exploring the parameter space of the model (the weights)
  - For NNs, most optimization methods are based on stochastic gradient descent
- Generalization properties
  - Learning ≠ Optimization
  - One wants to learn functions that generalize well

Optimisation : gradient methods – introduction

Optimization
Batch gradient algorithms

- Batch gradient general scheme
  - Training Data Set
    - $D = \{(\boldsymbol{x}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{x}^N, \boldsymbol{y}^N)\}$
  - Objective
    - Optimize a loss function $C(\boldsymbol{w}) = \sum_{i=1}^{N} c_{\mathrm{w}}(\boldsymbol{x}^i, \boldsymbol{y}^i)$
      - Sum of invidual losses $c_{\mathrm{w}}(.,.)$ on each example $(\boldsymbol{x}^i, \boldsymbol{y}^i)$
  - Principle
    - Initialize $\boldsymbol{w} = \boldsymbol{w}(0)$
    - Iterate until convergence
      - $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \epsilon(t)\boldsymbol{\Delta}_{\mathbf{w}}(t)$



    - $\boldsymbol{\Delta}_{\mathbf{w}}(t)$ is the descent direction, $\epsilon(t)$ is the gradient step
  - Both are determined via local information computed from $\boldsymbol{C}(\boldsymbol{w})$, using approximations of the 1st or 2nd order of $\boldsymbol{C}(\boldsymbol{w})$
    - e.g. steepest descent, is a 1^{st} order gradient with : $\boldsymbol{\Delta}_{\mathbf{w}}(t) = -\nabla_{\boldsymbol{w}} C(\mathrm{t}), \epsilon(t) = \epsilon$

Optimization
Batch second order gradients

- Consider a quadratic approximation of the loss function
  - C is approximated via a parabola
    - $C(w) = \mathrm{C}\big(\mathrm{w(t)}\big) + \big(\mathrm{w} - \mathrm{w(t)}\big)^{\mathrm{T}} \nabla C\big(w(t)\big) + \frac{1}{2}\big(\mathrm{w} - \mathrm{w(t)}\big)^{T} H\big(\mathrm{w} - \mathrm{w(t)}\big)$
    - where $\mathrm{w(t)}$ is the parameter vector at time t
      - H is the Hessian of $C(.) : H_{ij} = \frac{\partial^2 C}{\partial w_i \partial w_j}$
  - Differentiating w.r.t. w
    - $\nabla C(w) = \nabla C\big(w(t)\big) + H(w - w(t))$
  - The minimum of C is obtained for
    - $\nabla C(w) = 0$
  - Several iterative methods could be used
    - E.g. Newton
      - $w(t+1) = w(\mathrm{t}) - \mathrm{H}^{-1} \nabla C(w(t))$
      - Complexity $O(n^3)$ for the inverse + partial derivatives
      - In practice on makes use of quasi-Newton methods : H^{-1} is approximated iteratively

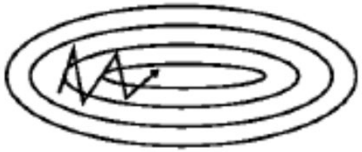
Optimization
Stochastic Gradient algorithms

- Objectives
  - Training NNs involves finding the parameters $\boldsymbol{w}$ by optimizing a loss
- Difficulties
  - Deep NN have a large number of parameters and meta-parameters, the loss is most often a non linear function of these parameters: the optimization problem is non convex
  - Optimization for Deep NN is often difficult:
    - Multiple local minima with high loss, …. might not be a problem in high dimensional spaces
    - Flat regions: plateaus -> 0 gradients, saddle points -> pb for 2nd order methods
    - Sharp regions: gradients may explode
    - Deep architectures: large number of gradient multiplications may often cause gradient vanishing or gradient exploding
- Solutions
  - There is no unique answer to all these challenges
  - The most common family of optimization methods for Deep NN is based on **stochastic gradient algorithms**
    - **Exploit the redundency in the data, at the cost of high variance in gradient estimates**
  - Deep Learning has developed several heuristic training methods
  - They are provided in the different toolboxes (Pytorch etc)
  - Some examples follow

Optimization
Stochastic gradient algorithms (From Ruder 2016)

- Data + Loss
  - Training Data Set
    - $D = \{(\boldsymbol{x}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{x}^N, \boldsymbol{y}^N)\}$
  - Loss function
    - $C(\boldsymbol{w}) = \sum_{i=1}^{N} c_{\mathrm{w}}(\boldsymbol{x}^i, \boldsymbol{y}^i)$
  - All the algorithms are given in vector form
- Basic Stochastic Gradient Descent
  - Initialise $\boldsymbol{w}(0)$
  - Iterate until stop criterion
  - sample un exemple $(\boldsymbol{x}(t), \boldsymbol{y}(t))$
  - $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \epsilon \nabla_{\boldsymbol{w}} c(\boldsymbol{x}(t), \boldsymbol{y}(t))$
  - Rq: might produce a lot of oscillations
- Momentum
  - Dampens oscillations
    - $\boldsymbol{m}(\boldsymbol{t}) = \boldsymbol{\gamma m}(\boldsymbol{t}-\mathbf{1}) + \epsilon \nabla_{\boldsymbol{w}} c(\boldsymbol{x}(t), \mathbf{y}(t))$
    - $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \boldsymbol{m}(\boldsymbol{t})$

(a) SGD without momentum



(b) SGD with momentum

Figures from (Ruder 2016)

Optimization
SGD algorithms with Adaptive learning rate

- Adagrad
  - One learning rate for each parameter w_i at each time step t
  - Iteration t
    - Compute gradient $\boldsymbol{g}(t) = \nabla_{\boldsymbol{w}} c(\boldsymbol{x}(t), \boldsymbol{y}(t))$ — Vector
    - Accumulate squared gradients for each component $r_i(t) = r_i(t-1) + (g_i(t))^2$ — Scalar
      - kind of gradient variance
      - Sum of the squared gradients up to step t
    - Componentwise:
    - $w_i(t+1) = w_i(t) - \frac{\epsilon}{\sqrt{\boldsymbol{r_i}(t)+\epsilon'}} \nabla_{w_i} c(\boldsymbol{x}(t), \boldsymbol{y}(t))$ — Scalar
  - In vector form
    - $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \frac{\epsilon}{\sqrt{\boldsymbol{r}(t)+\epsilon'}} \odot \nabla_{\boldsymbol{w}} c(\boldsymbol{x}(t), \boldsymbol{y}(t))$ — Vector
    - $\odot$ elementwise multiplication, ϵ' ($\approx 10^{-8}$) avoids dividing by 0, $\frac{\epsilon}{\sqrt{\boldsymbol{r}(t)+\epsilon'}}$ is a vector with components $\frac{\epsilon}{\sqrt{\boldsymbol{r_i}(t)+\epsilon'}}$
  - Default : learning rate shrinks too fast
- RMS prop
  - Replace $r(t)$ in Adagrad by an exponentially decaying average of past gradients
    - $\boldsymbol{r}(t) = \gamma r(t-1) + (1-\gamma)\boldsymbol{g}(t) \odot \boldsymbol{g}(t), \quad 0 < \gamma < 1$
    - $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \frac{\epsilon}{\sqrt{\boldsymbol{r}(t)+\epsilon'}} \odot \nabla_{\boldsymbol{w}} c(\boldsymbol{x}(t), \boldsymbol{y}(t))$ — Vector

Optimization
SGD algorithm with momentum and Adaptive learning rate

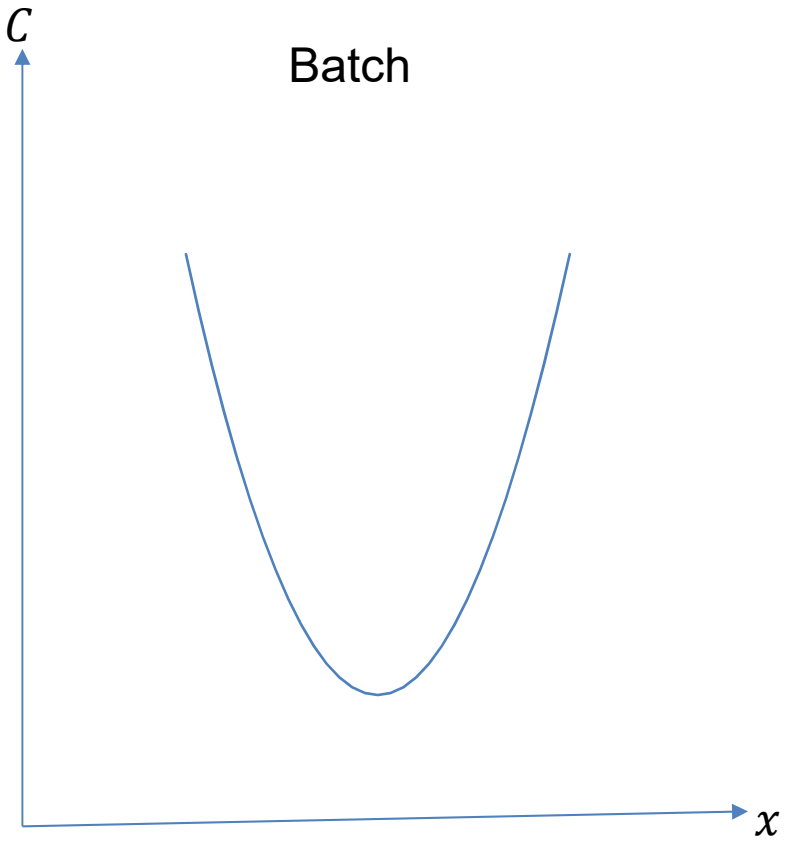
- Adam (adaptive moment estimation)
  - Computes
    - Adaptive learning rates for each parameter
    - An exponentially decaying avarage of past gradients (momentum)
    - An exponentially decaying average of past squared gradients (like RMSprop)
  - Iteration t
    - Momentum term : $\boldsymbol{m}(\boldsymbol{t}) = \gamma_1 \boldsymbol{m}(t-1) + \epsilon(1-\gamma_1)\boldsymbol{g}(t)$
    - Gradient variance term: $\mathrm{r}(t) = \gamma_2 \boldsymbol{r}(t-1) + \epsilon(1-\gamma_2)\boldsymbol{g}(t) \odot \boldsymbol{g}(t)$
    - $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \frac{\epsilon}{\sqrt{\boldsymbol{r}(t)}+\epsilon\prime} \odot \boldsymbol{m}(t)$
    - Bias correction
      - The 2 moments are initialized at 0, they tend to be biased towards 0, the following correction terms reduce this effect
        - Correct bias of $\boldsymbol{m}$: $\boldsymbol{m}(\boldsymbol{t}) = \frac{\boldsymbol{m}(\boldsymbol{t})}{1-\gamma_1^t}$
        - Correct bias of $\boldsymbol{r}$: $\mathbf{r}(t) = \frac{\boldsymbol{r}(\boldsymbol{t})}{1-\gamma_2^t}$

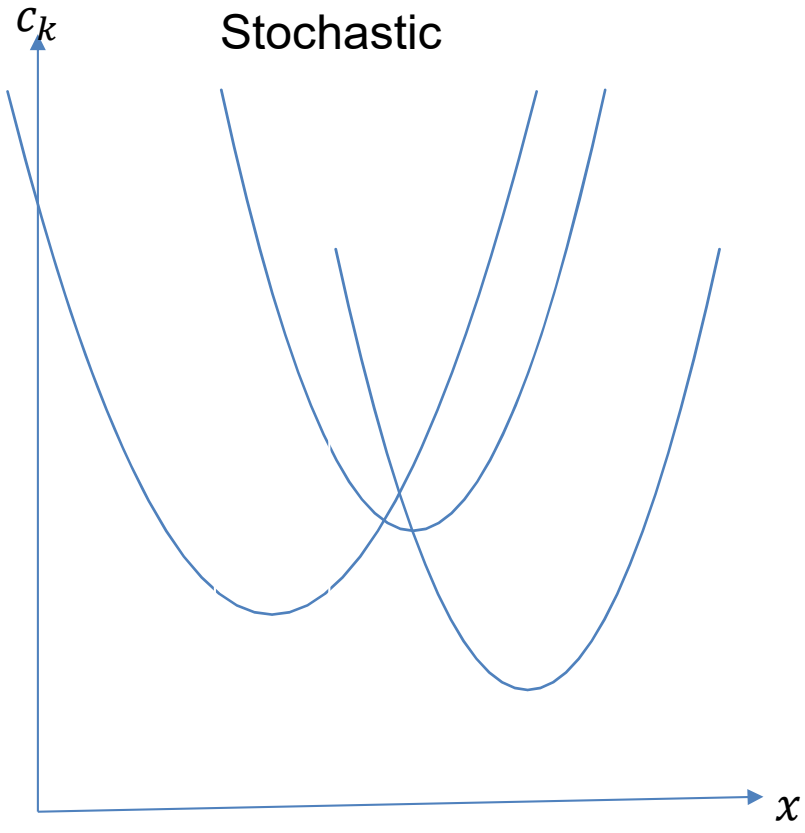
Batch vs stochastic gradient

$$C = \frac{1}{N}\sum_k c_k$$

C: global loss

c_k: individual (pattern k) loss





Gradient methods as numerical integration of ordinary differential equations (ODE)

- Let $l: R^d \rightarrow R$ a function we seek to minimize
  - We make the assumption that l is « well behaved »
- Consider the following gradient flow equation
  - $\frac{dw(t)}{dt} = -\nabla l(w(t))$
    $w(0) = w_0$
- Taylor expansion around $w(t)$ is:
  - $w(t+h) = w(t) + h\frac{\mathrm{dw(t)}}{\mathrm{dt}} + O(h^2)$
- Lets take $\mathrm{t} = kh$, by neglegting the second order terms, we get the explicit Euler method for integrating ODEs
  - $w_{k+1} = w_k - h\nabla l(w(t))$
  - Wich is the steepest descent algorithm
- Message
  - This interpretation of Gradient Descent as a numerical integration method for the gradient flow equation allows us to use the results from numerical analysis to characterize useful properties e. g. stability / consistence of the method
  - This is used for analyzing more sophisticated GD algorithms

Optimization
Summary

- Which method to use?
  - No « one solution for all problems »
  - For large scale applications, Adam is often used today as a default choice together with minibatches
  - But… simple SGD with heuristic learning rate decay can sometimes be competitive …
- Batch, mini batch, pure SGD
  - Stochastic methods exploit data redundancy
  - Mini batch well suited for GPU
  -