Table of Contents

Learning (supervised and unsupervised)

- From artificial neuron to deep learning
- Unsupervised learning: k-means and bandit algorithms
- See next devoted sections

Table of Contents

Table of Contents

Linear artificial neuron

- Mc Cullogh & Pitts in 1943 : linear "neuron". Emulate some logical functions ($\{0,1\}$-valued inputs and outputs).
- The first perceptron: 1957 by Rosenblatt ([9]) as a binary classifier.
- Learning by Hebb's rule also known as reinforcement rule.
- Only able to classify linearly separable classes of data.

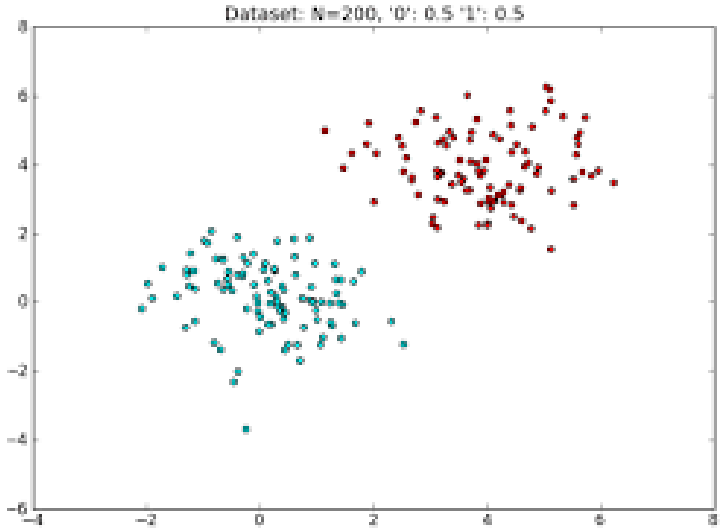


Figure: The Rosenblatt neural network performances.

[9] Rosenblatt, Frank (1957), The Perceptron: a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

No hidden layer: ADALINE (Adaptive Linear Neuron (B. Widrow & T. Hoff, 1960))

- Input data $x_1, \ldots, x_N$ of the form $x_k = (1, x_k^1, \ldots, x_k^d) \in \mathbb{R}^{d+1}$.
- Output (true) data are real numbers $y_1, \ldots, y_N$.
- Data set: $z_k = (x_k, y_k)$, $k = 1 : N$ (supervised learning).
- Let $\Theta = \mathbb{R}^{d+1}$ the parameter set.
- The answer of the ADALINE neuron/network for $\theta \in \Theta$ "fed" with an imput datum $x = (1, x^1, \ldots, x^d)$ is

$$\theta^\top x = \sum_{i=0}^{d} \theta^i x^i = \theta^0 + \sum_{i=1}^{d} \theta^i x^i$$

  where $\theta^\top$ denote the transpose of θ.
- Local (convex) prediction/loss function: $v\big(\theta, (x, y)\big) = \frac{1}{2}(y - \theta^\top x)^2$.

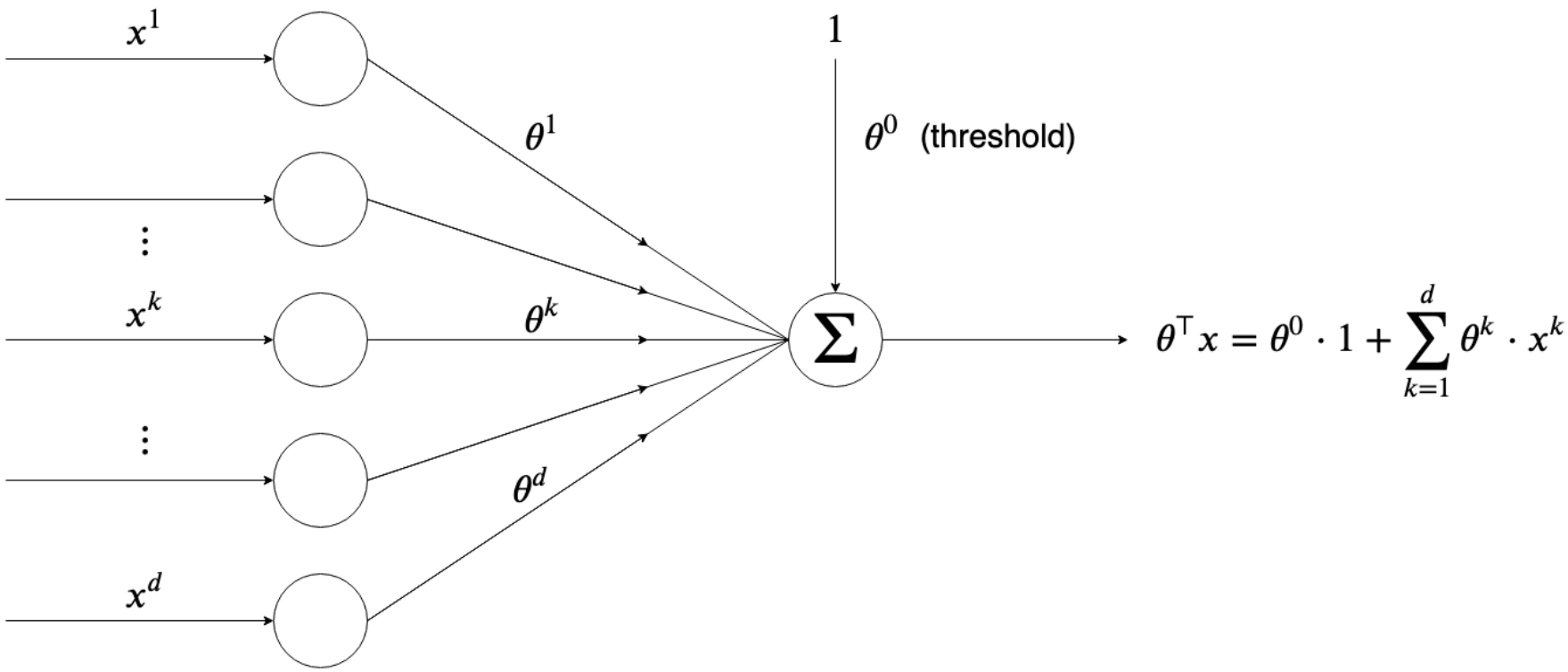


Figure: The ADALINE neural network.

Remark. When $\sum$ is replaced by the indicator function $\mathbf{1}_{\mathbb{R}_+}$ i.e. the output is $\mathbf{1}_{\theta^\top x \geq 0}$ this network becomes Rosenblatt's perceptron.

- $V(\theta) = \frac{1}{2}\int (\theta^\top x - y)^2 d\mu_N(x,y) = \mathbb{E}\,(\theta^\top x_I - y_I)^2$ (convex!) (where $I \overset{d}{=} \mathcal{U}(\{1,\dots,N\})$)
- i.e. Global prediction/loss function = Ordinary Least squares !
- $\nabla V(\theta) = \int (\theta^\top x - y)x\, d\mu_N(x,y)) = \mathbb{E}\,(\theta^\top x_I - y_I)x_I$.
- The target is

$$\nabla V(\theta) = 0 \Longleftrightarrow \int \underbrace{(\theta^\top x)x}_{=(xx^\top)\theta} d\mu_N(x,y)) = \int y\, x\, d\mu_N(x,y))$$

$$\Longleftrightarrow \theta_{opt} = \left(\int x\, x^\top d\mu_N(x,y)\right)^{-1}\left(\int y\, x\, d\mu_N(x,y)\right).$$

- $\left[\textit{Exercise:}\ \text{Prove that}\ (\theta^\top x)x = (xx^\top)\theta\right]$.
- It is linear regression (as expected).

- Resulting (SGD):

$$\theta_{n+1} = \theta_n - \gamma_{n+1}\big(\theta_n^\top x_{I_{n+1}} - y_{I_{n+1}}\big)x_{I_{n+1}}$$

where $(I_n)_{n\geq 1}$ are i.i.d. with unifrom distribution on $\{1,\ldots,N\}$.
All assumptions are satisfied $\Longrightarrow \theta_n \to \theta_{opt}$ $a.s.$

- Conclusion : the ADALINE network performs linear regression without matrix inversion.

- Extension to q-dimensional (true) output data y_k with $\Theta = \mathbb{M}(d,q)$ and

$$v\big(\Theta,(x,y)\big) = \tfrac{1}{2}|\Theta^\top x - y|^2.$$

"Non-linear" ADALINE

- Assume the output data y_k are e.g. $(0,1)$-valued.
- Let $\Psi \in C\big(\mathbb{R},(0,1)\big)$, Ψ increasing C^2-diffeomorphism from $\mathbb{R}$ onto $(0,1)$.
- $V(\theta) = \frac{1}{2}\mathbb{E}\left(\Psi(\theta^\top X_I) - Y_I\right)^2$ with $I \sim \mathcal{U}\big(\{1,\ldots,N\}\big)$.
- $\nabla V(\theta) = \mathbb{E}\left(\big(\Psi(\theta^\top X_I) - Y_I\big)\Psi'(\theta^\top X_I)X_I\right)$.
- Loss of convexity since

$$\nabla^2 V(\theta) = \int \Big[\Psi'(\theta^\top x)^2 + \underbrace{\big(\Psi(\theta^\top x) - y\big)}_{\text{varying sign}}\underbrace{\Psi''(\theta^\top x)}_{\not\geq 0}\Big] xx^\top d\mu_N(x,y)$$

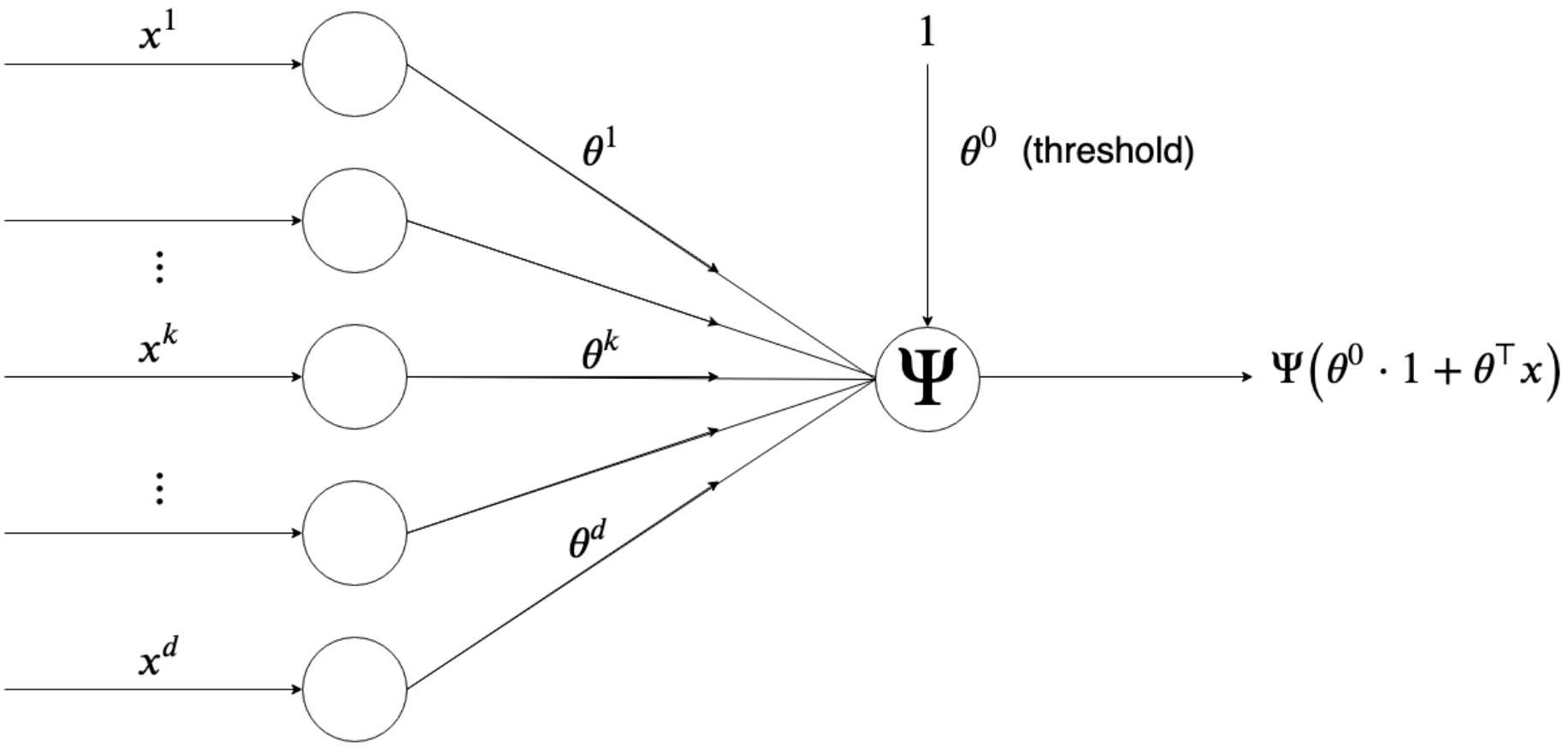


Figure: The non-linear ADALINE neural network.

- Historical Rosenblatt's "linear perceptron"' (1957): $\Psi(u) = \mathbf{1}_{u \geq 0}$ is a linear classifier.

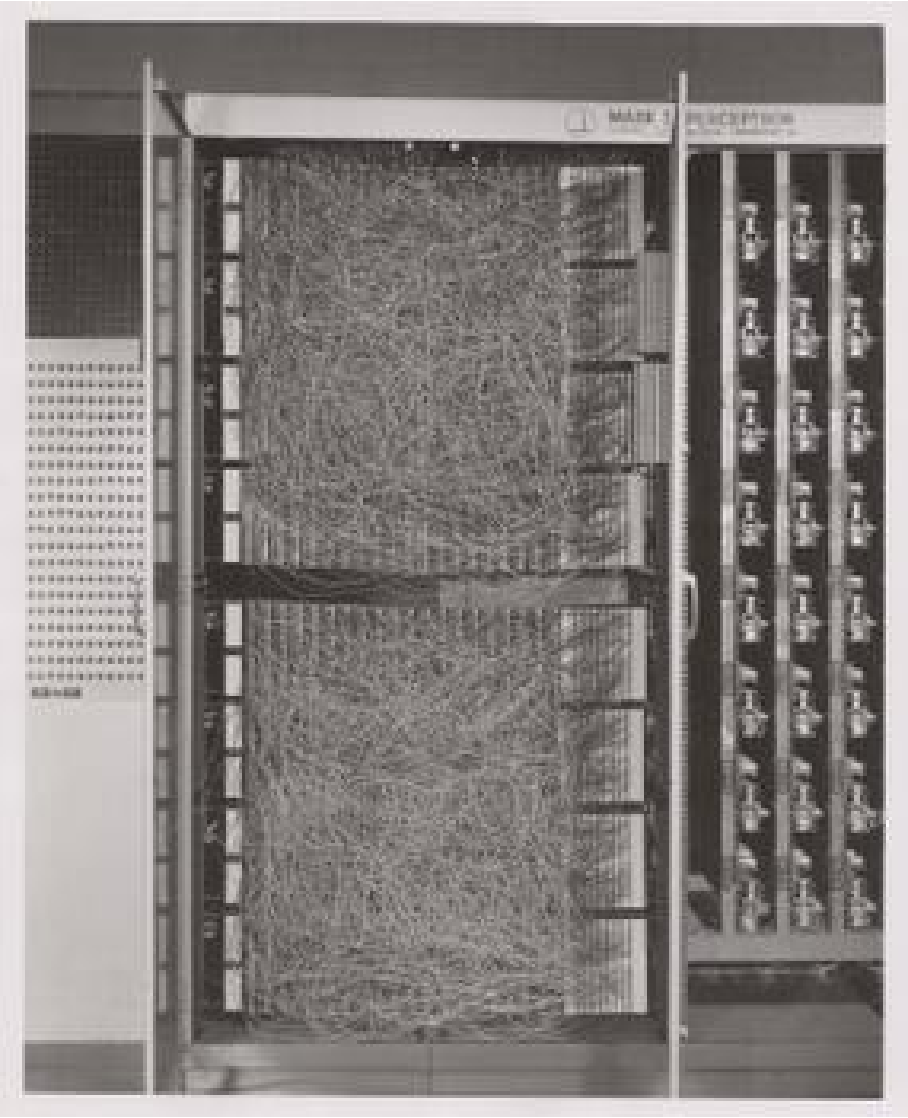


Figure: Historical Rosenblatt's perceptron: the hardware

Table of Contents

One hidden layer: universal approximation property

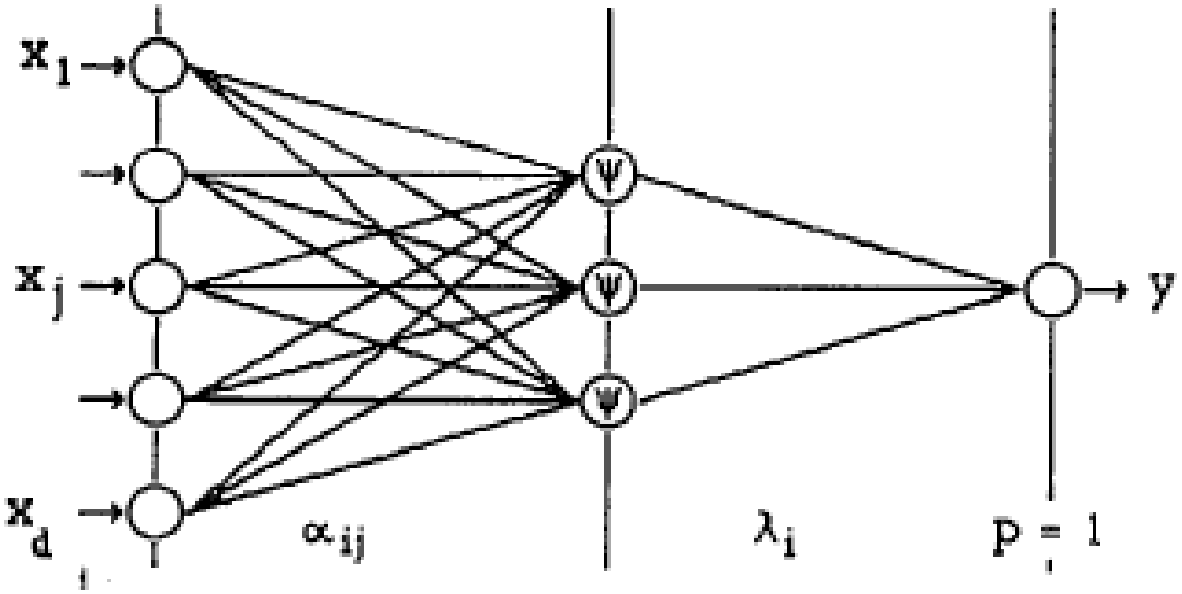


Figure: *One hidden layer MLP* *(below: convention $\alpha_{ij} = w_{ij}$).*

- Introduced by P. Werbos en 1974 (including backpropagation of errors) ([10]).

[10] Werbos, Paul J. (1994). The Roots of Backpropagation : From Ordered Derivatives to Neural Networks and Political Forecasting. New York: John Wiley & Sons.

One hidden layer: universal approximation property

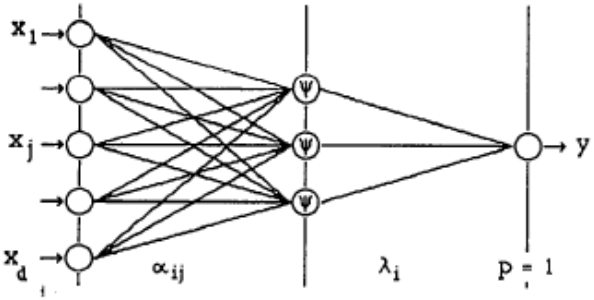


Figure: *One hidden layer MLP* *(below: convention $\alpha_{ij} = w_{ij}$).*

- $d = d_x$-dimensional inputs. Switch to $x \rightsquigarrow \begin{pmatrix} 1 \\ x \end{pmatrix}$.
- L units $i = 1 : L$ on the hidden layer, with an *activation function*

$$\Psi \in C(\mathbb{R}, \mathbb{R}).$$

- Unit i of the hidden layer receives $x = (x^j)_{j=1:d}$ and emits

$$\Psi\big((w_{i\cdot}|x)\big) = \Psi\Big(w_{i0} + \sum_{1 \le j \le d} w_{ij}\, x^j\Big).$$

- The output layer receives $\big[\Psi\big((w_{i\cdot}|x)\big)\big]_{i=1:L}$ and emits

$$\sum_{1 \le i \le L} \lambda_i \Psi\big((w_{i\cdot}|x)\big).$$

Table of Contents

Universal approximation property

Theorem (Cybenko, 1989)

([a]) If the activation function Ψ satisfies

$$(CL_\Psi) \equiv \Psi \in C_b(\mathbb{R},\mathbb{R}), \quad \lim_{\xi\to-\infty}\Psi(\xi)=0, \quad \lim_{\xi\to+\infty}\Psi(\xi)=1,$$

then

$$F=\left\{x\longmapsto\sum_{i=1}^{L}\lambda_i\Psi\big((w_i.|x)+w_{i0}\big),\ L\in\mathbb{N}^*,\ \lambda\in\mathbb{R}^L,\ w\in\mathbb{R}^{L\times(d+1)}\right\}$$

is $\|\cdot\|_{\sup}$-dense in $C([0,1]^d,\mathbb{R})$.

[a] G. Cybenko, Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems, 2:303-314, 1989.

- (CL_Ψ) can be slightly relaxed into Hornik et al's condition ([11], 1989)

$$(CS_\Psi) \equiv \Psi \in C_b(\mathbb{R},\mathbb{R}), \quad \exists\, u_1,\ u_2\in\mathbb{R} \text{ such that } \Psi(u_1)\neq\Psi(u_2).$$

- Extension to vector-valued outputs is straightforward.

[11] Hornik K., Stinchcombe M. & White H., Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, 359–366

Proof I

Assume $d = 1$ (for simplicity).

- Step 1. If $\overline{F}^{\|\cdot\|_{\sup}} \subsetneq C([0,1],\mathbb{R})$, $\exists$ a linear form $\Lambda \in C([0,1],\mathbb{R})^*$ s.t.

$$\Lambda \not\equiv 0 \quad \text{and} \quad \Lambda_{|\overline{F}} \equiv 0 \quad \text{(by Hahn-Banach's Theorem)}.$$

By Riesz's representation Theorem, there exists a signed measure μ s.t.

$$\Lambda g = \int_{\mathbb{R}} g \, d\mu.$$

Hence, with for every weight $w = (\alpha, \beta)$,

$$\forall\, w = (\alpha,\beta) \in \mathbb{R}^2, \quad \int_{\mathbb{R}} \Psi(\alpha x + \beta)\mu(dx) = 0.$$

- Step 2. Let $\mu = \mu^+ - \mu^-$, denote the/a decomposition of μ.
  - Let $\alpha = 0$, $\beta \to +\infty$.

$$\int \mathbf{1}\mu(dx) = 0 \quad \text{so that} \quad \mu^+(\mathbb{R}) = \mu^-(\mathbb{R}).$$

Proof II

- Now setting $\beta = -\alpha u$ yields

$$\forall\, \alpha,\ u \in \mathbb{R}, \quad \int \Psi(\alpha(x-u))\mu(dx) = 0.$$

Letting $\alpha \to +\infty$ implies, it follows from Lebesgue Dominated Cvgce theorem

$$\forall\, u \in \mathbb{R}, \quad \mu\big(\{x : x > u\}\big) + \Psi(0)\mu(\{u\}) = 0.$$

- The set $D = \big\{u : \mu^+(\{u\}) > 0 \text{ or } \mu^-(\{u\}) > 0\big\}$ is at most countable and, for every $u \in D^c$,

$$\forall\, u \in D^c, \quad \mu\big(\{x : x > u\}\big) = 0.$$

▶ Step 3. Combined with $\mu^+(\mathbb{R}) = \mu^-(\mathbb{R})$, one has

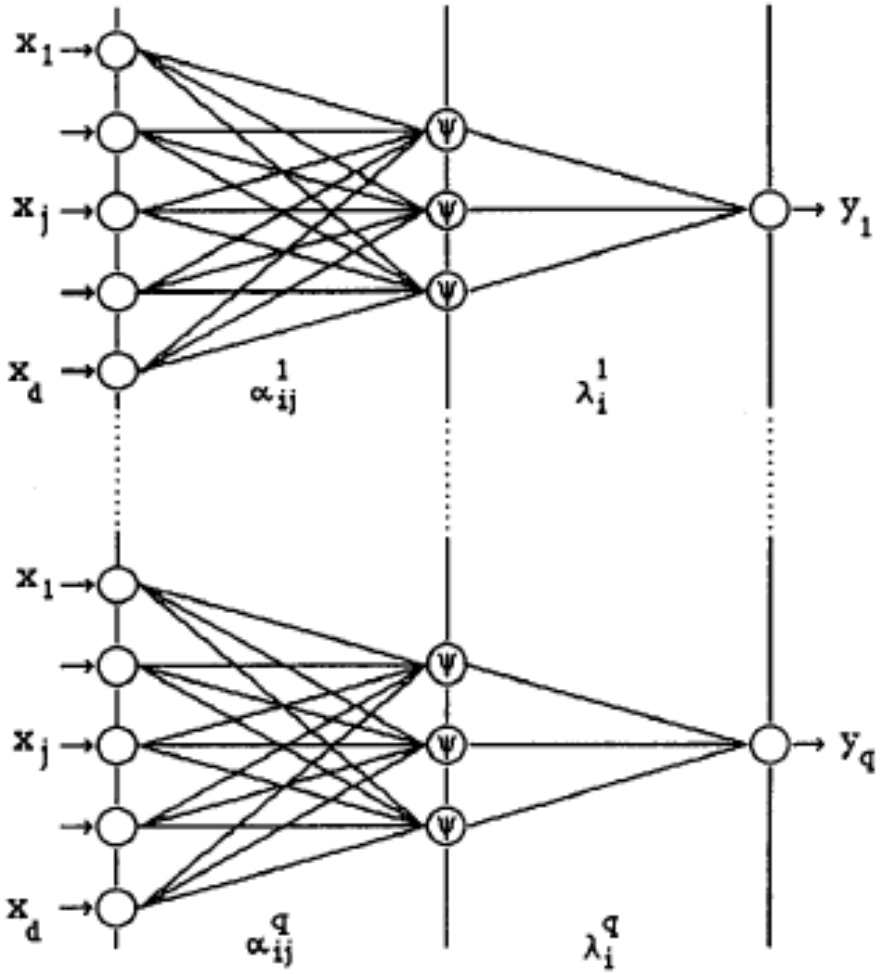
$$\forall\, u \in \mathbb{R}, \quad \mu^+\big(\{x : x > u\}\big) = \mu^-\big(\{x : x > u\}\big).$$

Hence $\mu^+ = \mu^-$ since D^c is everywhere dense in $\mathbb{R}$ i.e. $\mu \equiv 0$.

▶ Step 4. Contradiction! Since $\Lambda \not\equiv 0$. □

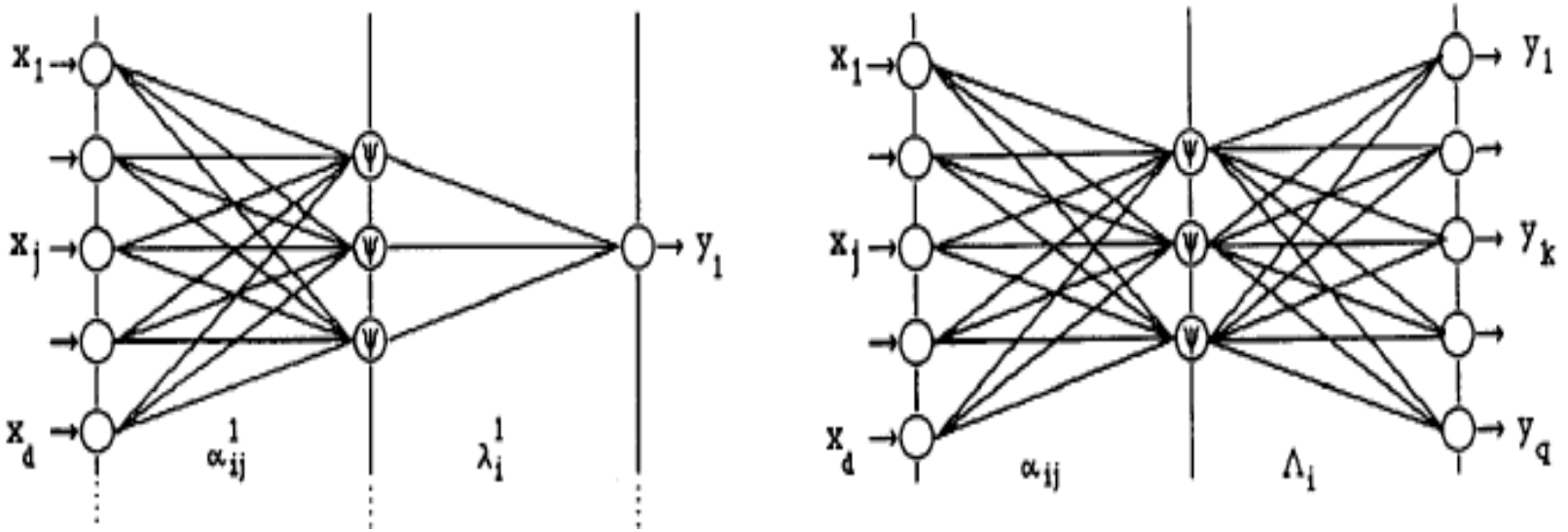
Higher dimensional outputs I

- Instant (theoretical) result by concatenating networks. . . (Warning! Unusual notations for indices)



Higher dimensional outputs II

- . . . Although this is better in practice (Warning! same unusual notations for indices)



Constructive proof/smoothness

- Both proofs are not constructive (Hornik's relies on a distribution based argument).
- Constructive approach are possible including rates for higher order derivatives like. Among others...

Theorem (Attali-P., *Neural Networks*, 1997)

([a]) Assume the activation function $\Psi \in C_b^r(\mathbb{R}, \mathbb{R})$ *and satisfies*

ψ *is non-polynomial on any open interval and* $\exists\, \xi_0$ *s.t.* $\psi^{(s)}(\xi_0) \neq 0$, $s = 0, \ldots, r$.

Let $f \in C^\infty(\mathbb{R}^d, \mathbb{R})$. *For every* $L \in \mathbb{N}$, *there exists weights* $(w_{ij}^{(L)})_{i=1:L, j=1:d}$, $\lambda_{i=1:L}^{(L)}$ *such that (with* $\|g\|_{[0,1]^d} = \sup_{\xi \in [0,1]^d} |g(\xi)|$*)*

$$\max_{s=0:r} \left\| \partial^s \Big(\sum_{i=1}^{L} \lambda_i^{(L)} \psi\big((w_{i\cdot}^{(L)} | x)\big) \Big) - \partial^s f \right\|_{[0,1]^d} \leq C_f \cdot L^{-1/d}$$

[a] J.-G. Attali, G. Pagès, Approximations of Functions by a Multilayer perceptron: a New Approach, *Neural Networks*, **10**(6):1069-10811,1997.

Curse of dimensionality

- The proof heavily relies on multi-variable Bernstein polynomials and Vandermonde determinants.
- Obviously suffers from the curse of dimensionality through this $L^{-1/d}$ or even $L^{-(r+1-s)/d}$ for $\partial_s f$.
- A celebrated paper by A. Barron (seems to) break(s) this curse using Maurey's Lemma with a rate $O(L^{-1/2})$ for a function $f : [0,1]^d \to \mathbb{R}$, C^1, with an additional constraint on the Fourier coefficients (a series is converging) ([12]).
- Unfortunately this condition is more and more difficult to check as d grows: the space of such functions becomes sparse in $C^1([0,1]^d, \mathbb{R})$.
- These papers came too late: Vapnik's *SVM* were coming at the front owing to a nice mathematical framework.
- Third winter of Neural networks begins.

[12] Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Transactions on Information Theory*, Vol.39, Issue: 3, 930-945, May 1993.

By the way, what about ReLu ?

- What about using ReLu function ?
- Have in mind $\mathrm{ReLu}(x) = x^+ = \max(x, 0)$.
- In fact

$$\psi(x) = \tfrac{1}{2}\big((x+1)^+ - (x-1)^+\big)$$

  satisfies the assumptions of Cybenko's (and a fortiori Hornik et al. 's theorem(s))
- Hence, their theorem remains true by considering

$$\psi(x) = \mathrm{Relu}(x) = x^+.$$

Renewal !

- G. Hinton, Y. Le Cun and Y. Bengio stroke back in 2012's with outstanding performances classification results at the 2012 edition of ImageNet challenge, using deep learning...
- New networks with more layers, not fully connected and new type of units (convolutive units, recurrent units).
- and even more recently Generative Adversarial Networks (2014) seem to get rid of curse of dimensionality,
- but, so far, no theoretical evidence.

Table of Contents

From learning to deep learning

- Feedforward multilayer perceptron.
- **1998** (Y. Le Cun): MNIST database of handwritten figures.
  - Input: Image $= 28 \times 28$ pixels $\times$ 256 grey levels $d_x \simeq 784$.
  - Output: probability y_i for each 10 figures : $[0,1]^{10}$ i.e. $d_y \simeq 9$ or 10.
  - Predictor : Multilayer perceptron (MLP) with 1 hidden layers with 200 units $\Longrightarrow 784 \times 200 \times 10 = 1.6 \cdot 10^6$ parameters
  - Size of the database: $N = 50\,000$ for learning, 10 000 for testing.
- **2010**: Same MNIST database
  - Predictor: MLP with convolutional units on *GPU* with 5 hidden layers

$$[2\,500, 2\,000, 1\,500, 1\,000, 500] \simeq 7,84 \cdot 10^9 \text{ parameters}$$

- **2013** (Google) : ImageNet database of $N = 162 \times 10^6$ images of 100×100 pixels and 256 grey levels: $d_x \simeq 10^4$.
  - $d_y = 21$ (classifier across 21 classes).
  - Predictor: Convolutional MLP network $\simeq 10^4 \times 1.72 \cdot 10^9$ parameters.
  - Error rate $< 1/1\,000$.

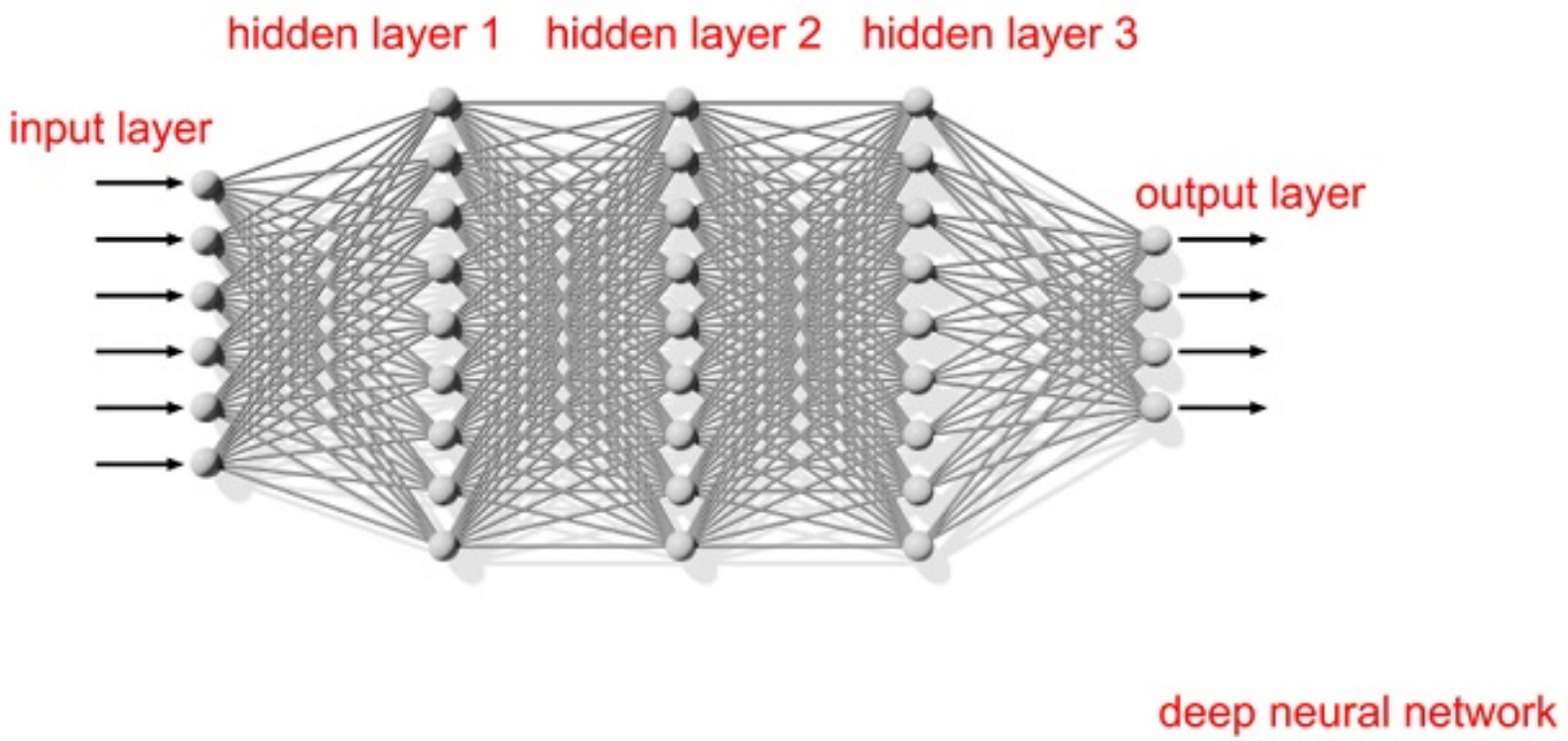


Figure: *A three-hidden layer feedforward perceptron.*

Table of Contents

Multilayer : toward Back propagation

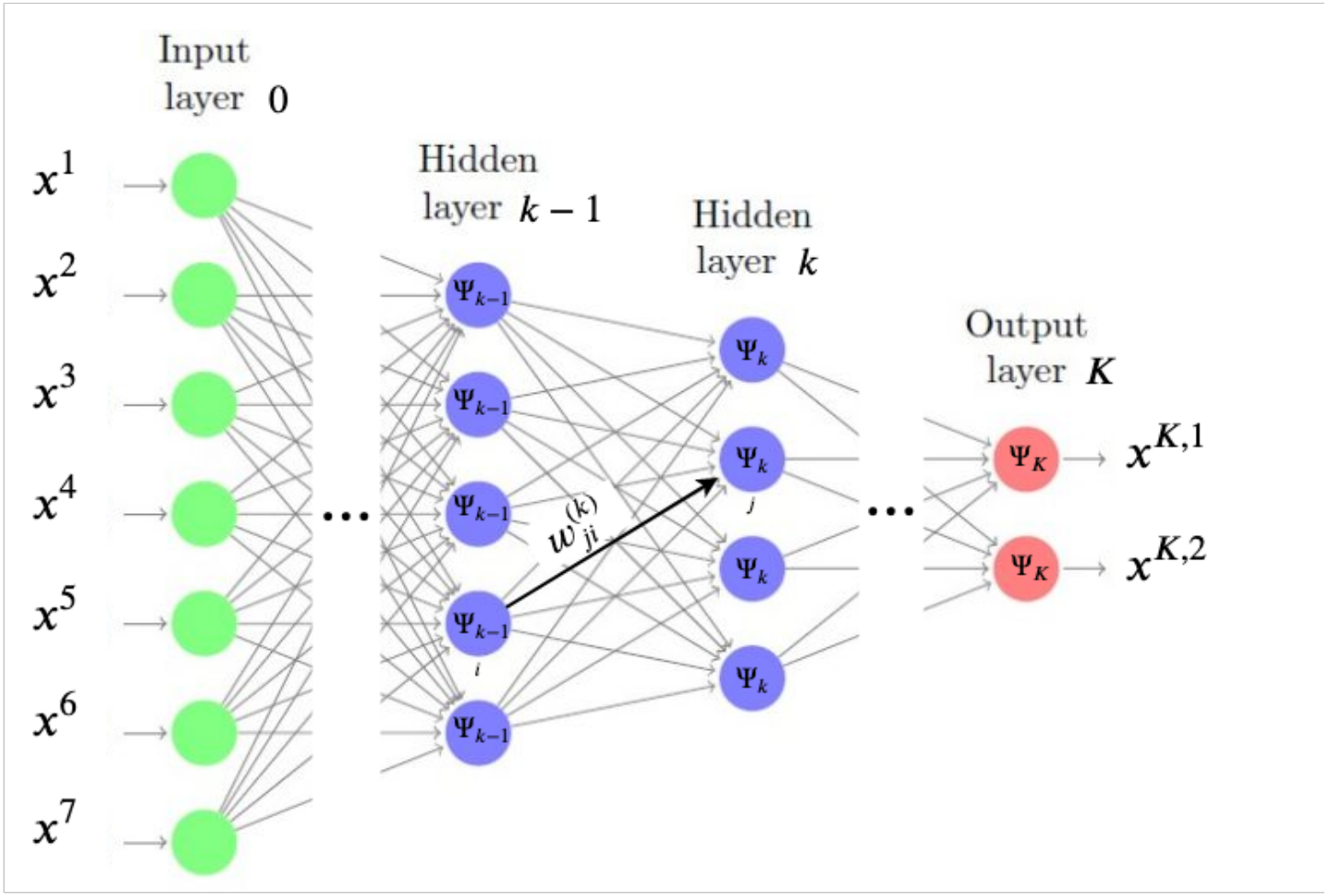


Figure: *A $K-1$ hidden layer feedforward perceptron.*

Multilayer and backpropagation in one slide

- Input data (layer 0): $x^0 = x$ i.e. $x_j^0 = x_j$, $j = 1 : d_0 = d + 1$,
- $K - 1$ hidden layers ($k = 1 : K - 1$) and $K + 1$ layers (0 and K)
- Output of unit i of layer $k - 1$: $x^{k-1,i}$.
- Input of unit j of layer k:

$$\sum_{1 \le i \le d_{k-1}} w_{ji}^{(k)} x^{k-1,i} = (w_{j\cdot}^{(k)} \mid x^{k-1,\cdot})$$

so that, after passing through the activation function

$$x^{k,j} = \Psi_k\big((w_{j\cdot}^{(k)} \mid x^{k-1,\cdot})\big).$$

- Typical activation functions

$$\Psi_k(\xi) = \frac{e^\xi}{e^\xi + 1}, \ \Psi_k(\xi) = \tanh(\xi), \ \Psi_k(\xi) = \xi_+ \ (\text{ReLu}),$$

and $\Psi_k(\xi) = \log(1 + e^\xi)$ or $\mathrm{swish}(\xi) = \xi \cdot \frac{e^\xi}{e^\xi + 1}$.

Intermission: connection with AAD

- Automatic differentiation : Robert E. Wengert 1964 in his PhD thesis ([13]).
- Reverse/Adjoint AD: Bryson (1962); Werbos (1974) but also Le Cun (1985); Rumelhart, Hinton (the three for Backprop!) and Speelpenning (1980) for true "automatic".

- AAD for Adjoint Automatic differentiation, proposes a model close to backpropagation but simpler.
- Let $g_k : \mathbb{R}^d \to \mathbb{R}^d$, $k = 1, \ldots, n$ be n differentiable vector fields.
- Set $J_x(g) = \left[\frac{\partial g^i}{\partial x^j}(x)\right]$ the Jacobian of $g = (g^1, \ldots, g^d)$ at $x = (x^1, \ldots, x^d)$.
- Then
$$J_x(g_n \circ \cdots \circ g_1)^\top = J_{y_0}(g_1)^\top \circ \cdots \circ J_{y_n}(g_n)^\top$$
- where $y_k = g_{k-1}(y_{k-1}), \ldots, y_0 = x$.

[13] Robert E. Wengert. A simple automatic derivative evaluation program. Communications of the ACM 7(8):463–4, Aug 1964

In fact two slides

- Parameter to be calibrated to perform the learning of the network:

$$\mathbf{w} = (w^{(0)}, w^{(1)}, \ldots, w^{(K)})$$

  with $w^{(0)} = Id$, $w^{(k)} \in \mathbb{M}_{d_k, d_{k+1}}(\mathbb{R})$.

- Dimension of $\mathbf{w} := D := d_0 \cdot d_1 + \cdots + d_{K-1} \cdot d_K$.
- Local Predictor: x^0 input datum, $x^K(\mathbf{w})$ output of network, y true "label"/target output.

$$v\big(\mathbf{w}, (x^0, y)\big) = \tfrac{1}{2}|y - x^K(\mathbf{w})|^2$$

- Global Predictor: $V(\mathbf{w}) = \mathbb{E}_{\mu_N}\, v\big(\mathbf{w}, (x^0, y)\big) = \mathbb{E}\, v\big(\mathbf{w}, (x_I^0, y_I)\big)$, $I \sim \mathcal{U}nif(\{1, \ldots, N\})$.
- Learning phase = Calibration

$$\min_{\mathbf{w} \in \mathbb{R}^D} V$$

- (SGD) ? Needs to differentiate V in $\mathbf{w}$ i.e. $v\big(\mathbf{w}, (x^0, y)\big)$!!

In fact three slides

- Big Question: How to differentiate (and compute!) $v\big(\mathbf{w},(x^0,y)\big)$?
- Easy part: from $v\big(\mathbf{w},(x^0,y)\big)=\frac{1}{2}|y-x^K(\mathbf{w})|^2$

$$(^\top \text{ for transpose}) \Downarrow (J \text{ for Jacobian matrix})$$

$$J_{\mathbf{w}} v\big(\mathbf{w},(x^0,y)\big)=\big(J_{\mathbf{w}} x^K(\mathbf{w})\big)^\top (x^K(\mathbf{w})-y).$$

- Use x^k as an auxiliary variable and the induction

$$x^k=\varphi_k(w^{(k)},x^{k-1}):=[\Psi_k\big((w^{(k)}_{j\cdot}\,|\,x^{k-1})\big)]_{1\le j\le d_k},\ k=1:K.$$

- Recursive formula: note that $J_{\mathbf{w}} x^k(\mathbf{w})^\top = J_{\mathbf{w}^{(1:k)}} x^k(\mathbf{w}^{(1:k)})^\top$ and

$$\begin{aligned} J_{\mathbf{w}} x^k(\mathbf{w})^\top &= J_{w^{(k)}}\varphi_K(w^{(k)},x^{k-1}(\mathbf{w}))^\top + J_{\mathbf{w}} x^{k-1}(\mathbf{w})^\top\, J_x\varphi_k(w^{(k)},x^{k-1}(\mathbf{w}))^\top \\ &= J_{w^{(k)}}\varphi_K(w^{(k)},x^{k-1}(\mathbf{w}))^\top \\ &\quad + J_{\mathbf{w}^{(1:k-1)}} x^{k-1}(\mathbf{w}^{(1:k-1)})^\top\, J_x\varphi_k(w^{(k)},x^{k-1}(\mathbf{w}))^\top. \end{aligned}$$

Finally . . . in four slides

Lemma

In a ring $(A, +, \cdot)$ with regular multiplication, if (b_k, c_k being known) the sequence $(a_k)_k$ recursively defined

$$a_k = b_k + a_{k-1}c_k, \ k = 1 : K$$

reads $$a_K = b_K + b_{K-1}c_K + b_{K-2}c_{K-1}c_K + \cdots + b_1 c_2 \cdots c_K + a_0 c_1 \cdots c_K$$

which can be computed recursively in a backward way.

The backpropagation algorithm ([14]) is two-fold.

- A forward step: Compute the (auxiliary!) values x_j^k at each unit j of each layer k using the weights $w^{(k-1)}$.
- A backward step: Apply the lemma to the recursion to compute

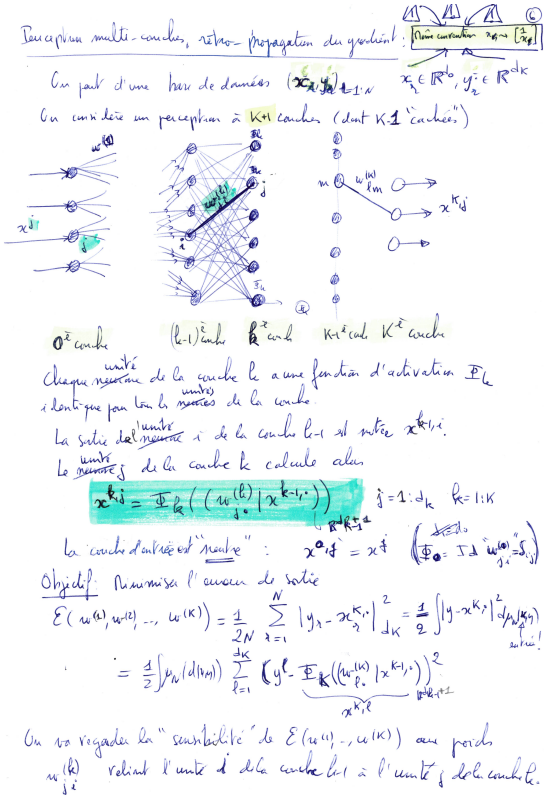
$$J_{w^{(k)}}\varphi_k(w^{(k)}, x^{k-1})^\top \quad \text{and} \quad J_{\mathbf{w}^{(1:k-1)}}x^{k-1}(\mathbf{w}^{(1:k-1)})^\top$$

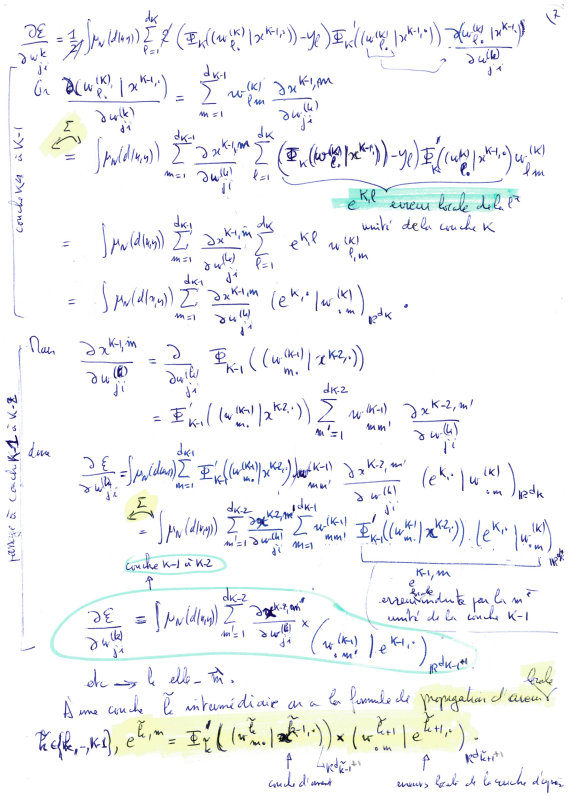
- And that's (almost) it!! (see further on).

[14] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323, 533–536. Le Cun's PhD thesis: Modèles Connexionnistes de l'Apprentissage (Univ. Paris 6, 1987). But goes back to Paul

The truth. . . about Backpropagation of gradient

- Backpropagation of gradient is similar to the reverse mode of automatic differentiation developed independently in the seventies!!
- True formulas: very involved but some "invariant" local errors terms appear (in yellow) to alleviate computations (cf. Le Cun).





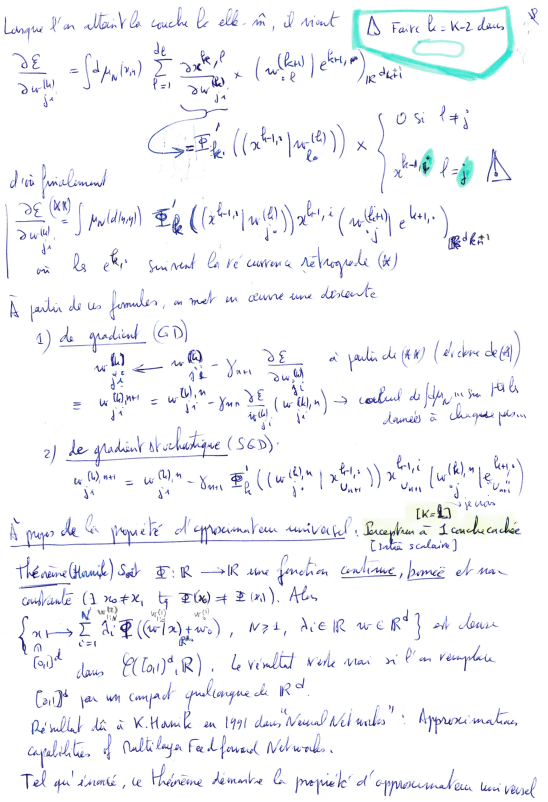


Table of Contents

Back to paradigm of Learning

- Database $(z_k)_{k=1:N}$, parameters $\theta \in \Theta \subset \mathbb{R}^K$ and (local) loss function/predictor $v(\theta, z)$.
- Let $(I_k)_{k\geq 1}$ be an i.i.d. sequence $\mathcal{U}\big(\{1,\ldots,N\}\big)$-distributed.
- The stochastic gradient descent reads

$$\theta_{n+1} = \theta_n - \gamma_{n+1}\nabla_\theta v\big(\theta_n, z_{I_{n+1}}\big)$$

  where $z_{I_{n+1}}$ means that a datum has been picked up at random in the database uniformly in $\{1,\ldots,N\}$.
- Check that

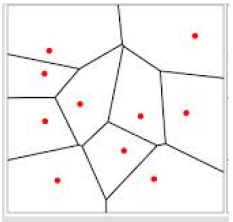
$$\begin{aligned}\mathbb{E}\,\nabla_\theta v\big(\theta_n, z_I\big) &= \frac{1}{N}\sum_{k=1}^{N}\nabla_\theta v\big(\theta_n, z_k\big)\\ &= \int \nabla_\theta v\big(\theta_n, z\big)\mu_N(dz) = \nabla V(\theta_n)\end{aligned}$$

Table of Contents

Unsupervised learning (example of clustering)

- Only input $z_k = x_k \in \mathbb{R}^d$, $k = 1 : N$.
- Prototype parameter set: $\theta := (\theta^1, \ldots, \theta^r) \in \Theta = (\mathbb{R}^d)^r$, $r \in \mathbb{N}$.
- (An example of) Local loss function: nearest neighbor among the prototypes: $x \in \mathbb{R}^d$, $\theta \in \Theta$.

$$v(\theta, x) = \frac{1}{2} \min_{i=1:r} |\theta^i - x|^2 = \frac{1}{2} \mathrm{dist}\big(x, \{\theta^1, \ldots, \theta^r\}\big)^2$$



(minimal distance to prototypes).

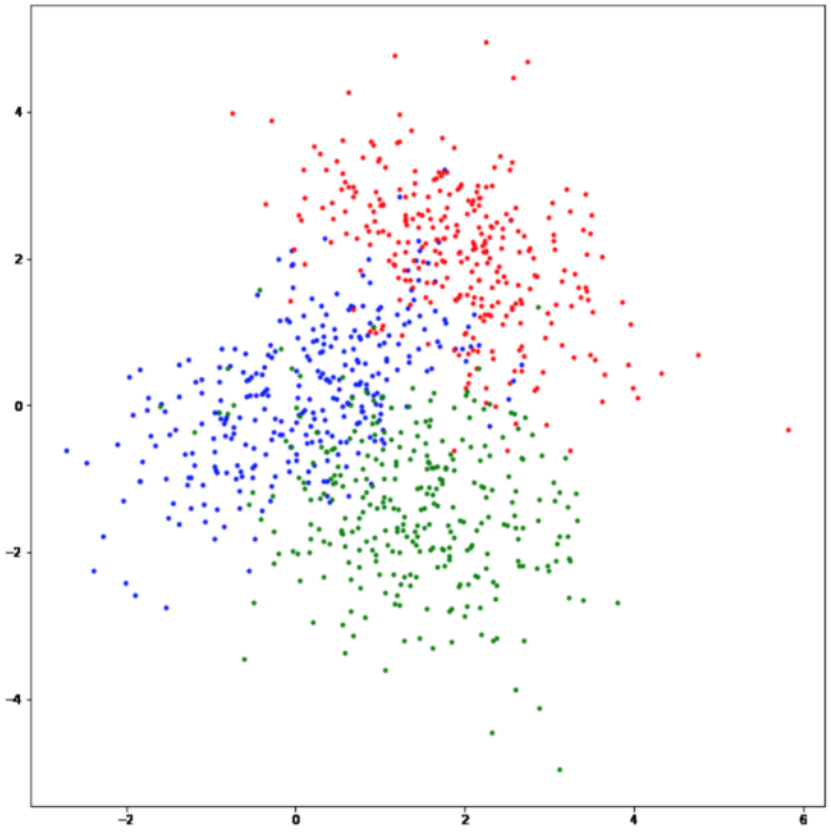
- $v(\theta, x)$ is not convex in θ!
- Global loss function (Distortion):

$$V(\theta) = \frac{1}{2N} \sum_{k=1}^{N} \min_{i=1:r} |\theta^i - x_k|^2 \quad \text{(mean minimal distance to prototypes).}$$

- Searching for the best prototypes: $\min_{\theta \in (\mathbb{R}^d)^r} V(\theta)$

A data set

The true data set is ...



Batch k-means/Forgy's algorithm

- Gradient at θ s.t. $\theta^i \neq \theta^j$: $\quad \nabla V(\theta) = \frac{1}{2N} \sum_{k=1}^{N} \nabla_\theta v(\theta, x_k)$

  with,

$$\forall\, i = 1 : r, \quad \partial_{\theta^i} v(\theta, x_k) = \left(\theta^i - x_k\right) \mathbf{1}_{\{|x_k - \theta^i| < \min_{j \neq i} |x_k - \theta^j|\}} \in \mathbb{R}^d.$$

- Compute the vector of $(\mathbb{R}^d)^r$: $\mathbf{1}_{\{|x_k - \theta^i| < \min_{j \neq i} |x_k - \theta^j|\}}$ = nearest neighbour search.
- Compute $\nabla V(\theta) = \frac{1}{2N} \sum_{k=1}^{N} \nabla_\theta v(\theta, x_k)$.
- $\Longrightarrow N\times$ nearest neighbour searches among r prototypes of dim d!
- Forgy's algorithm = GD algorithm (or batch GD algorithm):

$$\theta_{n+1} = \theta_n - \gamma_{n+1} \nabla V(\theta_n).$$

CLVQ/k-means (unsupervised learning)

- Aim:

$$\min_{(\theta^j)_{j=1:r}} \left[V(\theta) = \frac{1}{2} \sum_{k=1}^{N} \min_{i=1:r} |\theta^i - x_k|^2 \right]$$

  (mean minimal distance to prototypes).

- Competitive Learning Vector Quantization: let $\gamma_n \in (0,1)$, $n \geq 1$, then

$$\theta^i_{n+1} = \begin{cases} \theta^i_n - \gamma_{n+1}(\theta^i_n - x_{n+1}) & \text{if } |x_{n+1} - \theta^i_n| < \min_{j \neq i} |x_{n+1} - \theta^j_n| \\ \theta^i_n & \text{otherwise} \end{cases}$$

- In other words: $\rightarrow n+1$ reads
  - Nearest neighbour search to the datum among r prototypes of dimension d .
  - Moving the winner by a dilatation centered at the datum with ratio $1 - \gamma_{n+1} > 0$.

CLVQ

The true data set is ...

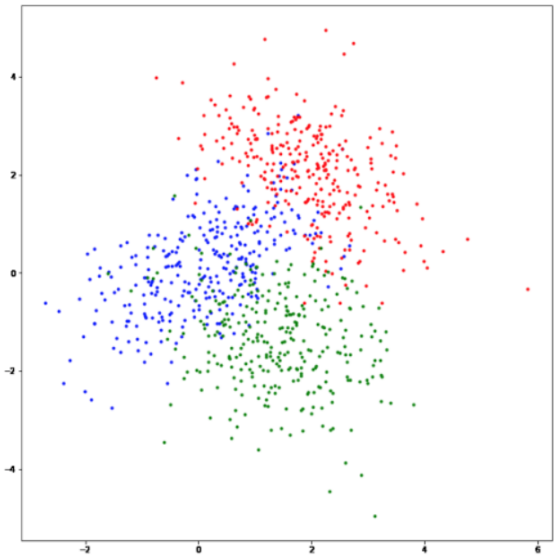


Figure: A mixture of 3 normal distributions.

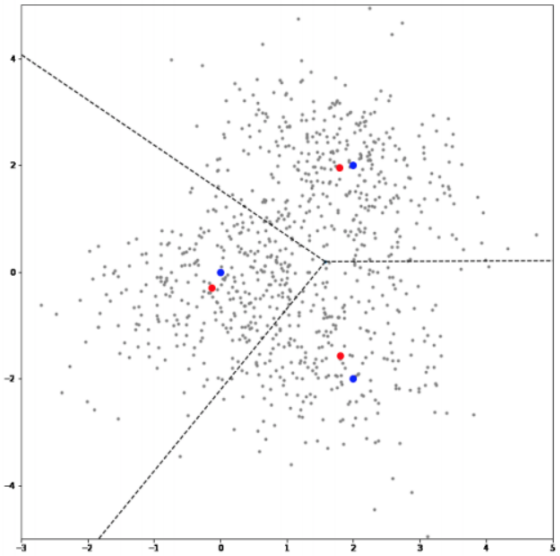


Figure: The red points are an optimal quantizer of size 3 of this data set. The blue points are the three centers of the normal distributions.

Figure: The CLVQ algorithm