

# Statistics and learning

## Neural Networks

Emmanuel Rachelson, Matthieu Vignes and Nathalie Villa-Vialaneix

ISAE SupAero

12<sup>th</sup> December 2013

# Some intuition

“Artificial Neural Networks” ?

# Spoiler alert!

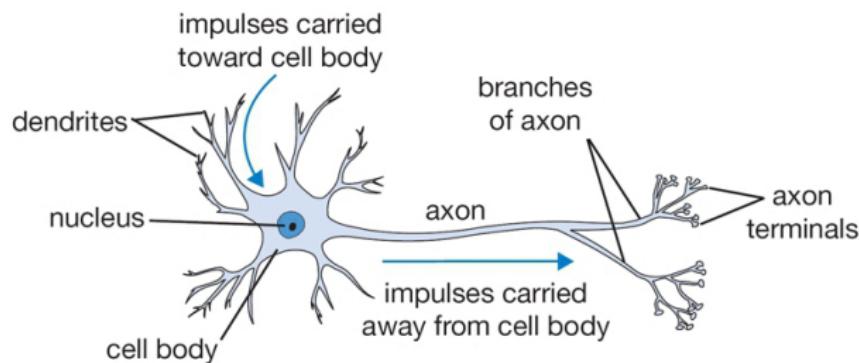
## Keywords

- ▶ Artificial Neurons and Artificial Neural Networks (and biological ones!).
- ▶ Hidden units/layers.
- ▶ Backpropagation, delta rule, NN batch/online training.
- ▶ Influence of the number of neurons/layers.
- ▶ Pros and cons of NN.

# A bit of history (and biology)

- ▶ *Early XXth cent.* The neuron (the biological one)!
- ▶ *40s.* McCulloch (neurophysiologist) and Pitts (logician), first formal neuron. Hebb's learning rule. Turing.
- ▶ *60s.* Rosenblatt's perceptron, XOR problem. Widrow and Hoff, backpropagation.
- ▶ *90s.* Computational power but new algorithms (SVM, ... )
- ▶ *Today.* Some great successes. Deep Learning.

# The biological neuron



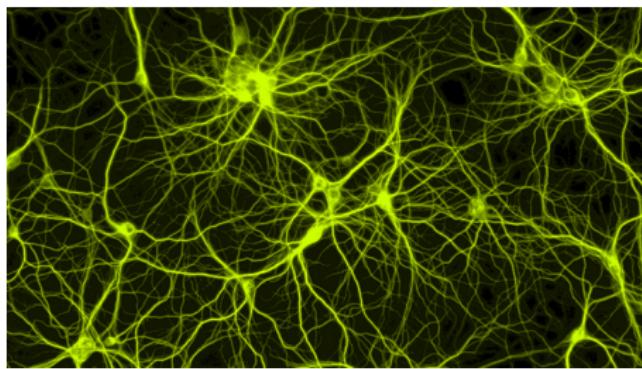
A neuron processes the info from its synapses and outputs it to the axon.

$$\text{Formal neuron : } z = \sigma (\alpha_0 + \alpha^T x)$$

## Activation function

$\sigma$  is the neuron's *activation function*.

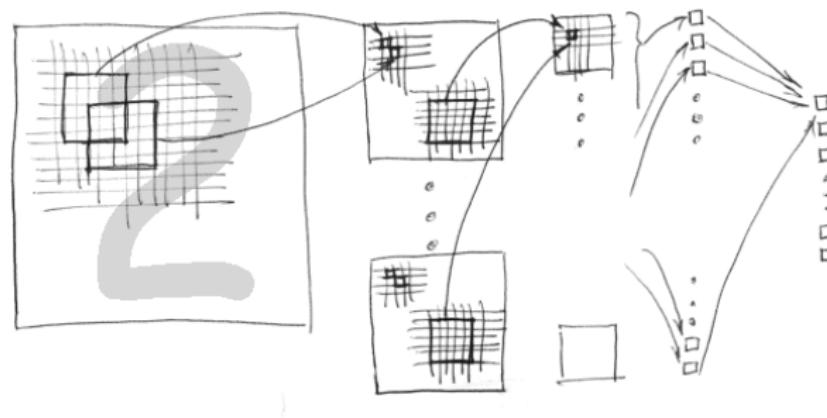
# The biological neural network



Each neuron processes a bit of info and passes it to its children.  
Overall the network processes raw information into general concepts.  
e.g. visual neurons.

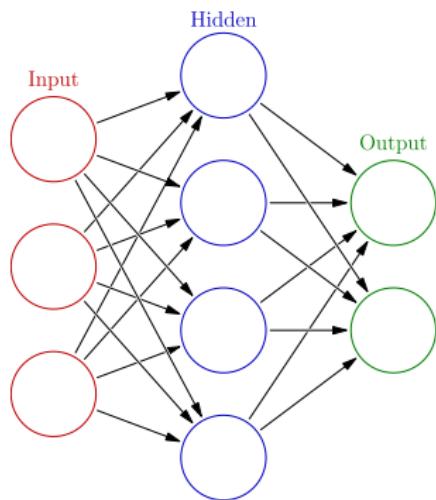
Our focus today: can we mimic this hierarchy of neurons into a learning system that adapts to data?

# From biological to artificial



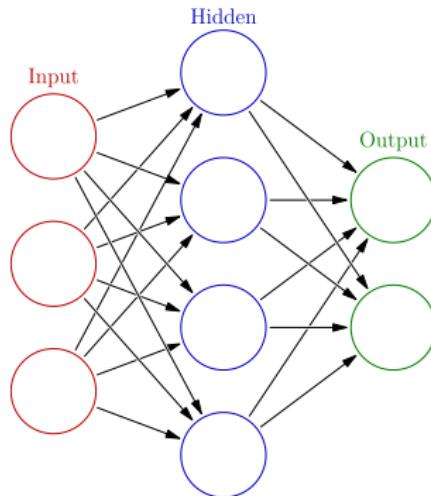
**Gradient-based learning applied to document recognition**, Le Cun et al., *IEEE*, 1998.

# The artificial neural network

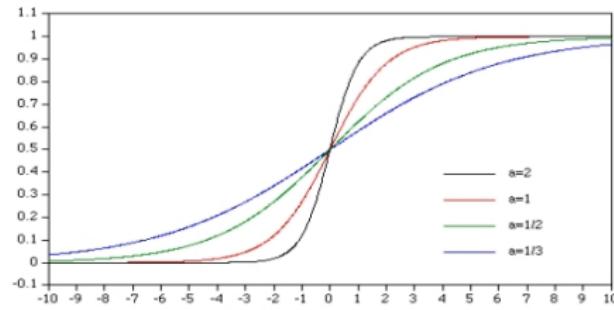


- ▶ Network diagram
- ▶ Layer = set of unconnected similar neurons
- ▶ Neuron = processing unit
- ▶ Parameters = edges weights
- ▶ Our case: single layer (easily generalizable)
- ▶ Input layer:  $X$
- ▶ Hidden layer:  $Z_m = \sigma(\alpha_{0m} + \alpha_m^T X)$
- ▶ Output layer:  $T_k = \beta_{0k} + \beta^T Z$  and  
 $Y_k = g_k(T) = f_k(X)$

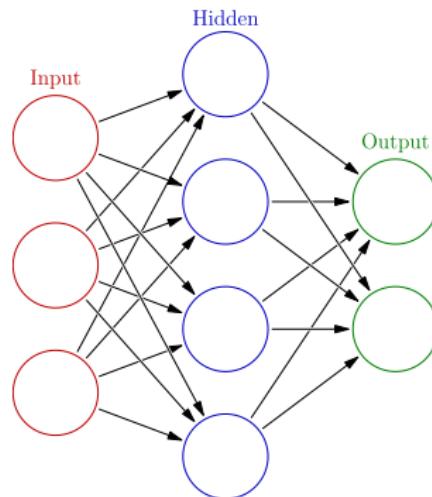
# Activation functions



- ▶ Sigmoid  $\sigma(v) = \frac{1}{1-e^{-v}}$ , mostly used in supervised learning.
- ▶ Linear  $\sigma(v) = v$ , results in linear model.
- ▶ Heaviside  $\sigma(v) = 1$  if  $v \geq 0$ , 0 otherwise, biological inspiration.
- ▶ RBF  $\sigma(v) = e^{-v^2}$ , used in unsupervised learning (SOM).



# Output functions



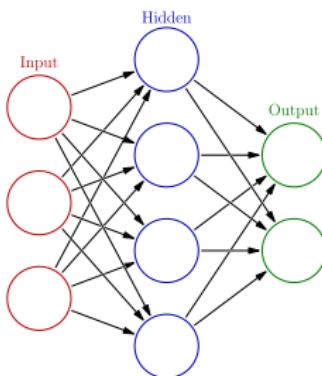
$$\text{Output: } T_k = \beta_{0k} + \beta^T Z$$

- $Z = (\text{hidden})$  basis expansion of  $X$ .

$$\text{Output: } Y_k = g_k(T)$$

- Regression  $g_k(T) = T_k$
- Classification  $g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^K e^{T_k}}$  (softmax)

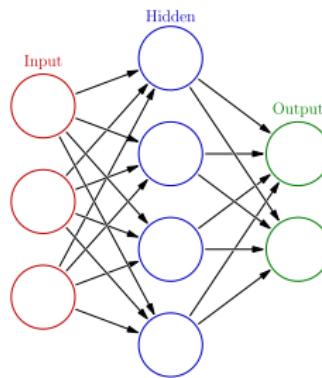
# Model parameters



Parameters vector  $\theta$ : 
$$\left\{ \begin{array}{l} \{\alpha_{0m}, \alpha_m; m = 1..M\} \rightarrow M(p+1) \text{ weights,} \\ \{\beta_{0k}, \beta_k; k = 1..K\} \rightarrow K(M+1) \text{ weights.} \end{array} \right.$$

Trick: get rid of  $\alpha_{0m}$  and  $\beta_{0k}$  by introducing a constant “1” input neuron.

# Error function



$$\text{Regression: } R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2$$

$$\text{Classification: } R(\theta) = - \sum_{k=1}^K \sum_{i=1}^N y_{ik} \log f_k(x_i)$$

What about noise? Overfitting?

## Fitting the NN to the data

$$R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2$$

Given  $\mathcal{T} = \{(x_i, y_i)\}$ , how do you suggest we proceed to find  $\theta$ ?

# Fitting the NN to the data

$$R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2$$

(Stochastic) gradient descent :

$$\min_{\theta} R(\theta)$$

⇒ compute  $\frac{\partial R}{\partial \theta}$

then update  $\theta^{(r+1)} \leftarrow \theta^{(r)} + \gamma_r \frac{\partial R}{\partial \theta}$

So let's see what  $\frac{\partial R}{\partial \theta}$  looks like!

## Gradients on $\beta$

Lets write  $R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2 = \sum_{i=1}^N R_i.$

## Gradients on $\beta$

Lets write  $R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2 = \sum_{i=1}^N R_i.$

$$\frac{\partial R_i}{\partial \beta_{km}} = \frac{\partial (y_i - f_k(x_i))^2}{\partial \beta_{km}}$$

## Gradients on $\beta$

Lets write  $R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2 = \sum_{i=1}^N R_i.$

$$\begin{aligned}\frac{\partial R_i}{\partial \beta_{km}} &= \frac{\partial (y_i - f_k(x_i))^2}{\partial \beta_{km}} \\ &= -2(y_{ik} - f_k(x_i)) \frac{\partial f_k(x_i)}{\partial \beta_{km}}\end{aligned}$$

## Gradients on $\beta$

Lets write  $R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2 = \sum_{i=1}^N R_i.$

$$\begin{aligned}\frac{\partial R_i}{\partial \beta_{km}} &= \frac{\partial (y_i - f_k(x_i))^2}{\partial \beta_{km}} \\ &= -2(y_{ik} - f_k(x_i)) \frac{\partial f_k(x_i)}{\partial \beta_{km}}, \text{ but } f_k(x_i) = g_k(\beta_k^T z_i) \\ &= -2(y_{ik} - f_k(x_i)) g'_k(\beta_k^T z_i) \frac{\partial \beta_k^T z_i}{\partial \beta_{km}}\end{aligned}$$

## Gradients on $\beta$

Lets write  $R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2 = \sum_{i=1}^N R_i.$

$$\begin{aligned}\frac{\partial R_i}{\partial \beta_{km}} &= \frac{\partial (y_i - f_k(x_i))^2}{\partial \beta_{km}} \\ &= -2(y_{ik} - f_k(x_i)) \frac{\partial f_k(x_i)}{\partial \beta_{km}}, \text{ but } f_k(x_i) = g_k(\beta_k^T z_i) \\ &= -2(y_{ik} - f_k(x_i)) g'_k(\beta_k^T z_i) \frac{\partial \beta_k^T z_i}{\partial \beta_{km}} \\ &= -2(y_{ik} - f_k(x_i)) g'_k(\beta_k^T z_i) z_{mi}\end{aligned}$$

## Gradients on $\beta$

Lets write  $R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2 = \sum_{i=1}^N R_i.$

$$\begin{aligned}\frac{\partial R_i}{\partial \beta_{km}} &= \frac{\partial (y_i - f_k(x_i))^2}{\partial \beta_{km}} \\ &= -2(y_{ik} - f_k(x_i)) \frac{\partial f_k(x_i)}{\partial \beta_{km}}, \text{ but } f_k(x_i) = g_k(\beta_k^T z_i) \\ &= -2(y_{ik} - f_k(x_i)) g'_k(\beta_k^T z_i) \frac{\partial \beta_k^T z_i}{\partial \beta_{km}} \\ &= -2(y_{ik} - f_k(x_i)) g'_k(\beta_k^T z_i) z_{mi}\end{aligned}$$

So, as  $x_i$  goes through the network, one can compute this gradient! Let's write:

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}$$

## Gradients on $\alpha$

Left as exercice:

$$\frac{\partial R_i}{\partial \alpha_{ml}} = - \sum_{k=1}^K 2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{il}$$

But remember:  $\delta_{ki} = -2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)$ , so:

$$\frac{\partial R_i}{\partial \alpha_{ml}} = \left[ \sigma'(\alpha_m^T x_i) \sum_{k=1}^K \beta_{km} \delta_{ki} \right] x_{il} = s_{mi} x_{il}$$

# Back-propagation, delta rule, Widrow & Hoff 1960

Forward pass, compute (and keep):

- ▶  $\alpha_m^T x_i, z_{mi}$  (activation of neuron  $m$  by input  $x_i$ )
- ▶  $\beta_k^T z_i, f_k(x_i)$  (activation of output  $k$  by input  $x_i$ )

Backward pass, compute:

- ▶  $\delta_{ki} = -2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)$  (when  $x_i$ 's signal reaches output  $k$ )
- ▶  $s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^K \beta_{km} \delta_{ki}$  (error back-propagation)

Update rule:

- ▶  $\beta_{km}^{(r+1)} \leftarrow \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \beta_{km}^{(r)}} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^N \delta_{ki} z_{mi}$
- ▶  $\alpha_{ml}^{(r+1)} \leftarrow \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^N s_{mi} x_{il}$

## Remark 1/3: distributed computing

$$\alpha_m^T x_i, z_{mi}, \beta_k^T z_i, f_k(x_i), \delta_{ki}, s_{mi}$$

Compute only neuron-based local quantities!

With limited connectivity, parallel computing.

## Remark 2/3: online vs. batch

When updating  $\theta$

- ▶ Online : apply delta rule for each  $(x_i, y_i)$  independently.
- ▶ Batch: cycle through the cases.

## Remark 2/3: online vs. batch

When updating  $\theta$

- ▶ Online : apply delta rule for each  $(x_i, y_i)$  independently.
- ▶ Batch: cycle through the cases.

Learning rate  $\gamma_r$ :  $\theta^{(r+1)} \leftarrow \theta^{(r)} + \gamma_r \frac{\partial R}{\partial \theta}$

- ▶ Batch: line search in gradient descent.
- ▶ Online: stochastic approximation procedure

(Robbins-Monro, 51) CV if  $\sum_{r=1}^{\infty} \gamma_r = \infty$ ,  $\sum_{r=1}^{\infty} \gamma_r^2 < \infty$

## Remark 3/3: other optimization procedures

$$\min_{\theta} R(\theta)$$

In practice, back-propagation is slow.

- ▶ 2nd order methods too complex (size of Hessian matrix)
- ▶ Conjugate gradients, Levenberg-Marquadt algorithm.

## ANNs in practice: initializing weights

- ▶ Good practice: initialize randomly close to zero but  $\neq 0$ .
- ▶ Reason: close to zero, the sigmoid is almost linear. Training brings the differentiation. But zero weights would yield zero gradients.
- ▶ In practice: too large initial weights perform poorly.
- ▶ Good range:  $[-0.7, 0.7]$  if normalized inputs.

# ANNs in practice: avoiding overfitting

What do you think?

# ANNs in practice: avoiding overfitting

- ▶ early stopping rule (using validation set).
- ▶ cross validation.
- ▶ regularization:  $R(\theta) + \lambda J(\theta) = R(\theta) + \lambda \left[ \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2 \right]$

Find the good  $\lambda$  by cross-validation.

$J(\theta)$  is differentiable: change the delta rule accordingly.

# ANNs in practice: scaling the inputs

Always scale the inputs!

It makes uniform random weights relevant.

## ANNs in practice: number of neurons/layers

- ▶ Too few = bad, not expressive enough.
- ▶ Too many = risk overfitting  
Use regularization (too many + regularization = generally good).  
Slower convergence.

Good practice in many cases:

- ▶ Single layer
- ▶ [5, 100] neurons
- ▶ Then refine the activation functions (specialized neurons) and the network's architecture.

## ANNs in practice: convexity of $R(\theta)$

$R(\theta)$  has no reason to be convex!

- ▶ Try random initializations and compare.
- ▶ Mixtures of expert ANNs (see next class on Boosting).

# Why should you use ANNs?

- ▶ Artificial Neurons and Artificial Neural Networks.
- ▶ Hidden units/layers.
- ▶ Backpropagation, delta rule, NN batch/online training.
- ▶ Good practices.

Pros:

- ▶ Intuitive, explainable process.
- ▶ Can approximate any function with any precision.
- ▶ Wide range of implementations available.

Cons:

- ▶ Non explainable results (or weights, except in specific cases like fuzzy NN).
- ▶ Slow training.
- ▶ No margin guarantees (further reading: Bayesian NN, regularization in NN).
- ▶ Sensitivity to noise and overfitting.

Yet widely used in control, identification, finance, etc.