Apprentissage connexionniste

Nathalie Villa-Vialaneix
http://www.nathalievilla.org

Journées d'Études en Statistique

2-7 octobre 2016 - Fréjus

Apprentissage statistique et données massives

Sommaire

Sommaire

What are (artificial) neural networks?

Common properties

- (artificial) "Neural networks": general name for supervised and unsupervised methods developed in (vague) analogy to the brain;

What are (artificial) neural networks?

Common properties

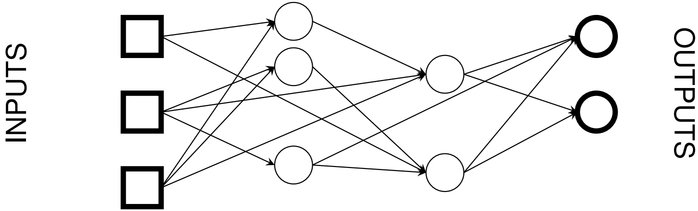
- (artificial) "Neural networks": general name for supervised and unsupervised methods developed in (vague) analogy to the brain;
- combination (network) of simple elements (neurons).

What are (artificial) neural networks?

Common properties

- (artificial) "Neural networks": general name for supervised and unsupervised methods developed in (vague) analogy to the brain;
- combination (network) of simple elements (neurons).

Example of graphical representation:



Different types of neural networks

A neural network is defined by:

1. the network structure;
2. the neuron type.

Different types of neural networks

A neural network is defined by:

1. the network structure;
2. the neuron type.

Standard examples

- Multilayer perceptrons (MLP) *Perceptron multi-couches*: dedicated to supervised problems (classification and regression);

Different types of neural networks

A neural network is defined by:

1. the network structure;
2. the neuron type.

Standard examples

- Multilayer perceptrons (MLP) *Perceptron multi-couches*: dedicated to supervised problems (classification and regression);
- Radial basis function networks (RBF): same purpose but based on local smoothing;

Different types of neural networks

A neural network is defined by:

1. the network structure;
2. the neuron type.

Standard examples

- Multilayer perceptrons (MLP) *Perceptron multi-couches*: dedicated to supervised problems (classification and regression);
- Radial basis function networks (RBF): same purpose but based on local smoothing;
- Self-organizing maps (SOM also sometimes called Kohonen's maps) or Topographic maps: dedicated to unsupervised problems (clustering), self-organized;
- ...

Different types of neural networks

A neural network is defined by:

1. the network structure;
2. the neuron type.

Standard examples

- Multilayer perceptrons (MLP) *Perceptron multi-couches*: dedicated to supervised problems (classification and regression);
- Radial basis function networks (RBF): same purpose but based on local smoothing;
- Self-organizing maps (SOM also sometimes called Kohonen's maps) or Topographic maps: dedicated to unsupervised problems (clustering), self-organized;
- ...

In this talk, focus on MLP.

MLP: Advantages/Drawbacks

Advantages

- classification OR regression (i.e., Y can be a numeric variable or a factor);
- non parametric method: flexible;
- good theoretical properties.

MLP: Advantages/Drawbacks

Advantages

- classification OR regression (i.e., Y can be a numeric variable or a factor);
- non parametric method: flexible;
- good theoretical properties.

Drawbacks

- hard to train (high computational cost, especially when d is large);
- overfit easily;
- "black box" models (hard to interpret).

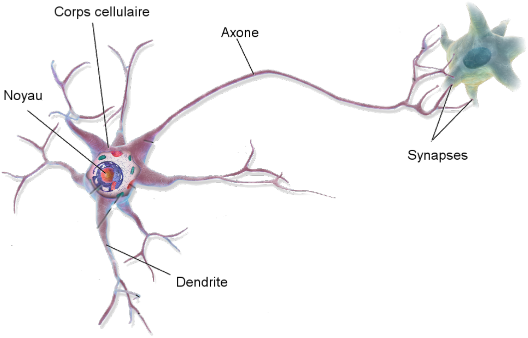
References

Advised references:

- [Bishop, 1995, Ripley, 1996] overview of the topic from a learning (more than statistical) perspective
- [Devroye et al., 1996, Györfi et al., 2002] in dedicated chapters present statistical properties of perceptrons

Sommaire

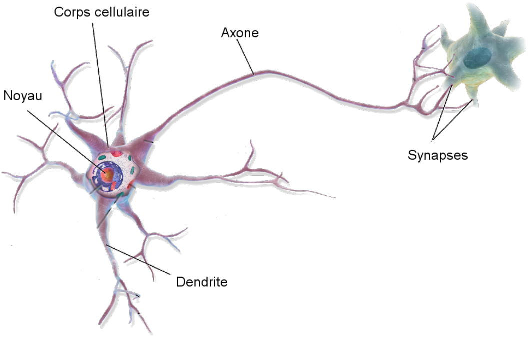
Analogy to the brain



1. a neuron collects signals from neighboring neurons through its dendrites

connexions which frequently lead to activating a neuron are enforced (tend to have an increasing impact on the destination neuron)

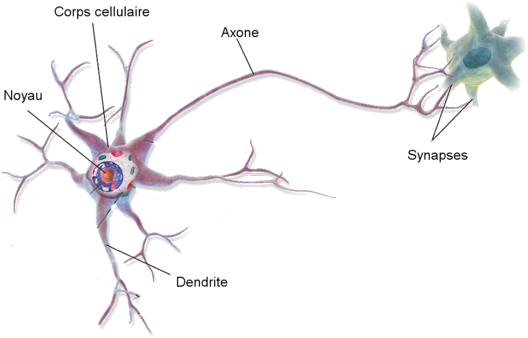
Analogy to the brain



1. a neuron collects signals from neighboring neurons through its dendrites
2. when total signal is above a given threshold, the neuron is activated

connexions which frequently lead to activating a neuron are enforced (tend to have an increasing impact on the destination neuron)

Analogy to the brain

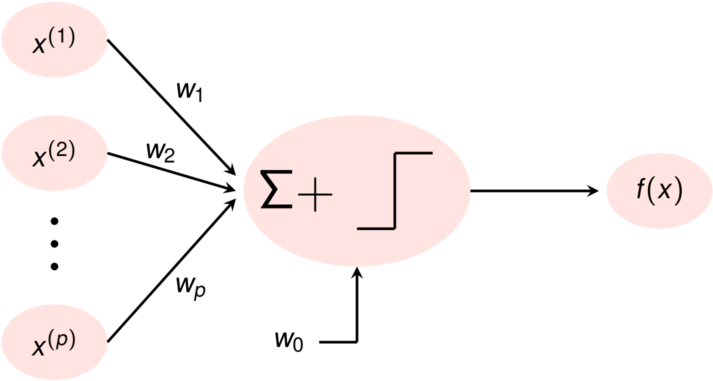


1. a neuron collects signals from neighboring neurons through its dendrites
2. when total signal is above a given threshold, the neuron is activated
3. ... and a signal is sent to other neurons through the axon

connexions which frequently lead to activating a neuron are enforced (tend to have an increasing impact on the destination neuron)

First model of artificial neuron

[Mc Culloch and Pitts, 1943, Rosenblatt, 1958, Rosenblatt, 1962]



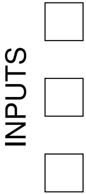
$$f: \; x \in \mathbb{R}^p \rightarrow \mathbb{1}_{\left\{\sum_{j=1}^p w_j x^{(j)} + w_0 \geq 0\right\}}$$

(artificial) Perceptron

Layers

- MLP have one input layer ($x \in \mathbb{R}^p$), one output layer ($y \in \mathbb{R}$ or $\in \{1, \ldots, K-1\}$ values) and several hidden layers;
- no connections within a layer;
- connections between two consecutive layers (feedforward).

Example (regression, $y \in \mathbb{R}$):



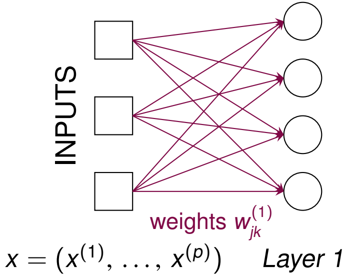
$$x = (x^{(1)}, \ldots, x^{(p)})$$

(artificial) Perceptron

Layers

- MLP have one input layer ($x \in \mathbb{R}^p$), one output layer ($y \in \mathbb{R}$ or $\in \{1, \ldots, K-1\}$ values) and several hidden layers;
- no connections within a layer;
- connections between two consecutive layers (feedforward).

Example (regression, $y \in \mathbb{R}$):

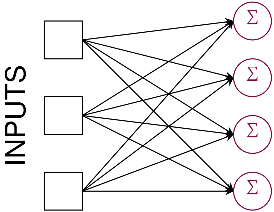


(artificial) Perceptron

Layers

- MLP have one input layer ($x \in \mathbb{R}^p$), one output layer ($y \in \mathbb{R}$ or $\in \{1, \ldots, K-1\}$ values) and several hidden layers;
- no connections within a layer;
- connections between two consecutive layers (feedforward).

Example (regression, $y \in \mathbb{R}$):



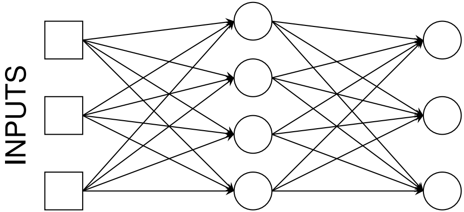
$x = (x^{(1)}, \ldots, x^{(p)})$ *Layer 1*

(artificial) Perceptron

Layers

- MLP have one input layer ($x \in \mathbb{R}^p$), one output layer ($y \in \mathbb{R}$ or $\in \{1, \ldots, K-1\}$ values) and several hidden layers;
- no connections within a layer;
- connections between two consecutive layers (feedforward).

Example (regression, $y \in \mathbb{R}$):



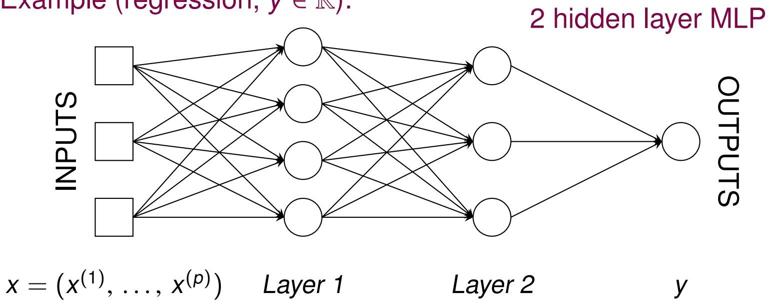
$x = (x^{(1)}, \ldots, x^{(p)})$ *Layer 1* *Layer 2*

(artificial) Perceptron

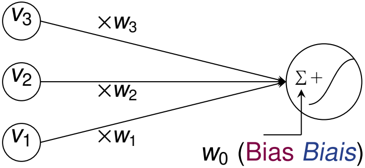
Layers

- MLP have one input layer ($x \in \mathbb{R}^p$), one output layer ($y \in \mathbb{R}$ or $\in \{1, \ldots, K-1\}$ values) and several hidden layers;
- no connections within a layer;
- connections between two consecutive layers (feedforward).

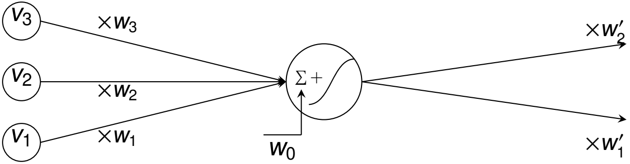
Example (regression, $y \in \mathbb{R}$):



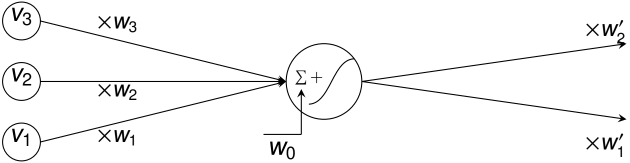
A neuron in MLP



A neuron in MLP

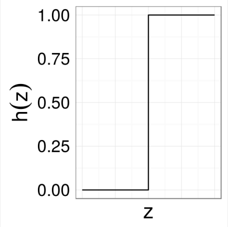


A neuron in MLP



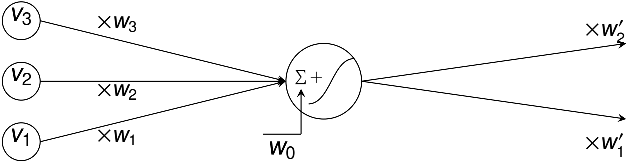
Standard activation functions *fonctions de transfert / d'activation*

Biologically inspired: Heaviside function



$$h(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{otherwise.} \end{cases}$$

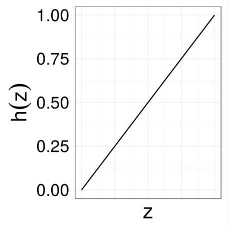
A neuron in MLP



Standard activation functions

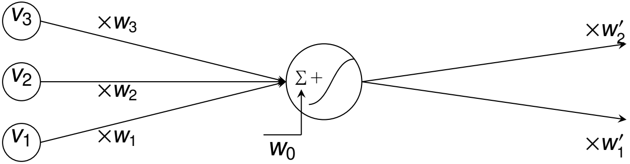
Main issue with the Heaviside function: not continuous!

Identity



$$h(z) = z$$

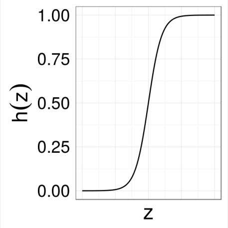
A neuron in MLP



Standard activation functions

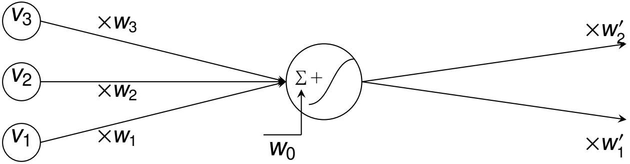
But identity activation function gives linear model if used with one hidden layer: not flexible enough

Logistic function



$$h(z) = \frac{1}{1+\exp(-z)}$$

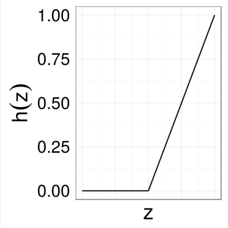
A neuron in MLP



Standard activation functions

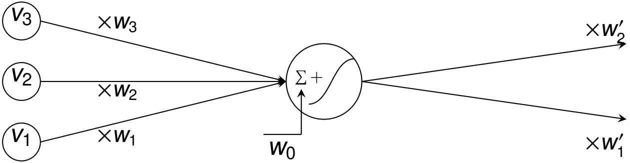
Another popular activation function (useful to model positive real numbers)

Rectified linear (ReLU)



$$h(z) = \max(0, z)$$

A neuron in MLP



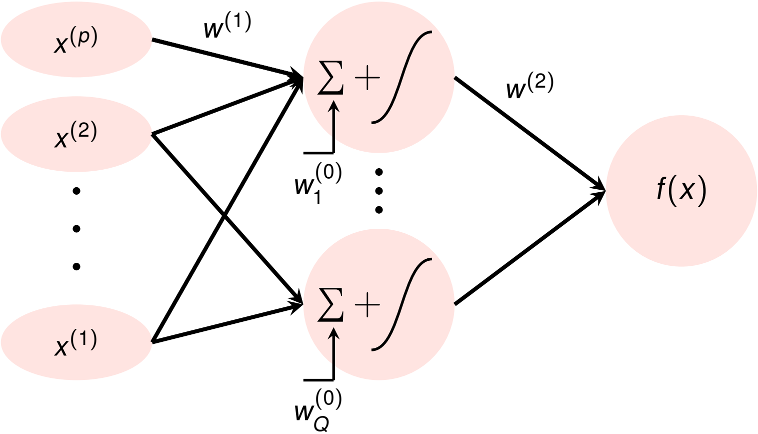
General sigmoid

sigmoid: nondecreasing function $h : \mathbb{R} \to \mathbb{R}$ such that

$$\lim_{z \to +\infty} h(z) = 1 \qquad \lim_{z \to -\infty} h(z) = 0$$

Focus on one-hidden-layer perceptrons

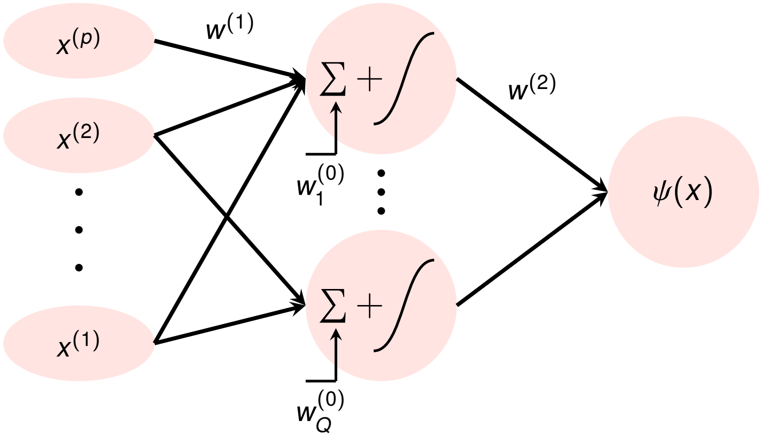
Regression case



$$f(x) = \sum_{k=1}^{Q} w_k^{(2)} h_k \left(x^\top w_k^{(1)} + w_k^{(0)} \right) + w_0^{(2)},$$ with h_k a (logistic) sigmoid

Focus on one-hidden-layer perceptrons

Binary classification case

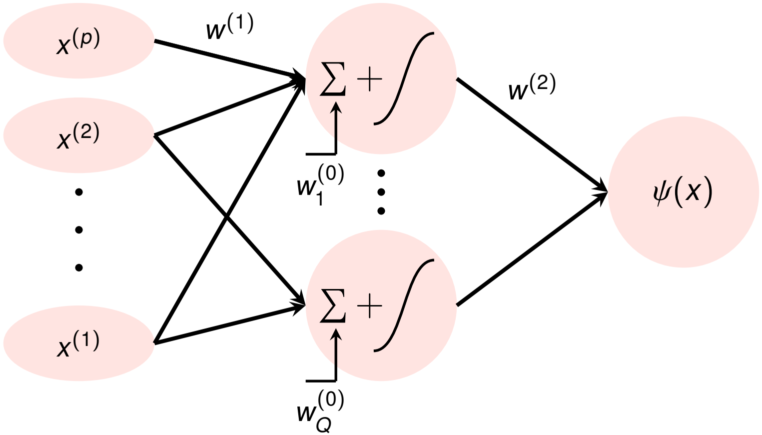


$$\psi(x) = h_0\left(\sum_{k=1}^{Q} w_k^{(2)} h_k\left(x^\top w_k^{(1)} + w_k^{(0)}\right) + w_0^{(2)}\right)$$

with h_0 logistic sigmoid or identity.

Focus on one-hidden-layer perceptrons

Binary classification case

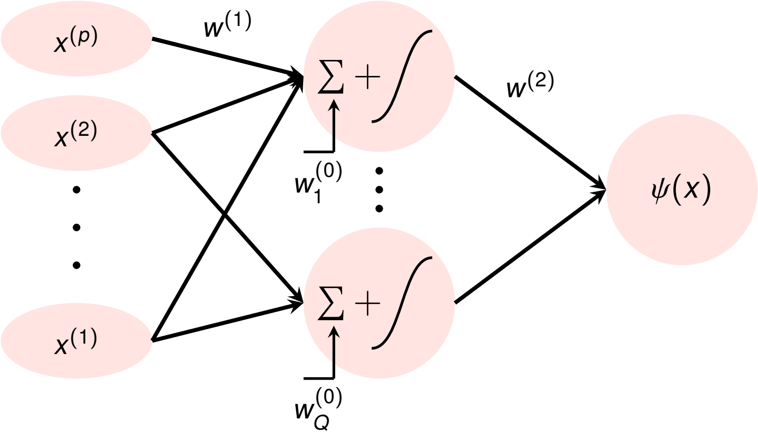


decision with:

$$f(x) = \begin{cases} 0 & \text{if } \psi(x) < 1/2 \\ 1 & \text{otherwise} \end{cases}$$

Focus on one-hidden-layer perceptrons

Extension to any classification problem in $\{1, \ldots, K-1\}$



Straightforward extension to multiple classes with a multiple output perceptron (number of output units equal to K) and a maximum probability rule for the decision.

Theoretical properties of perceptrons

This section answers two questions:

1. can we approximate any function $g : [0,1]^p \rightarrow \mathbb{R}$ arbitrary well with a perceptron?

Theoretical properties of perceptrons

This section answers two questions:

1. can we approximate any function $g : [0,1]^p \rightarrow \mathbb{R}$ arbitrary well with a perceptron?
2. when a perceptron is trained with i.i.d. observations from an arbitrary random variable pair (X, Y), is it consistent? (*i.e.*, does it reach the minimum possible error asymptotically when the number of observations grows to infinity?)

Illustration of the universal approximation property

Simple examples

- a function to approximate: $g: \ [0,1] \to \sin\left(\frac{1}{x+0.1}\right)$

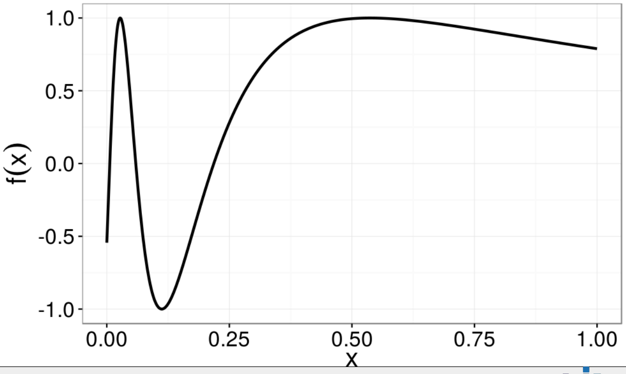
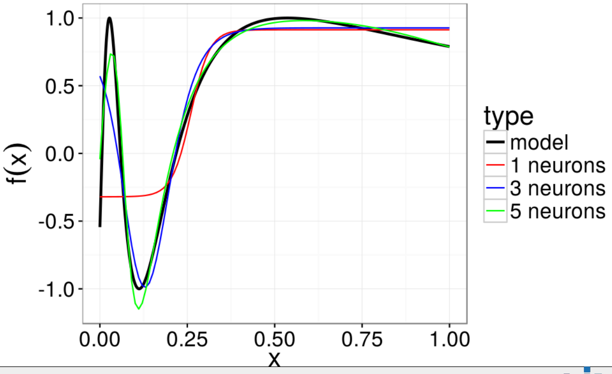


Illustration of the universal approximation property

Simple examples

- a function to approximate: $g: [0,1] \rightarrow \sin\left(\frac{1}{x+0.1}\right)$
- trying to approximate (how this is performed is explained later in this talk) this function with MLP having different numbers of neurons on their hidden layer



Universal property from a theoretical point of view

Set of MLPs with a given size:

$$\mathcal{P}^Q(h) = \left\{ x \in \mathbb{R}^p \rightarrow \sum_{k=1}^{Q} w_k^{(2)} h\left(x^\top w_k^{(1)} + w_k^{(0)}\right) + w_0^{(2)} \;:\; w_k^{(2)}, w_k^{(0)} \in \mathbb{R}, \; w_k^{(1)} \in \mathbb{R}^p \right\}$$

Universal property from a theoretical point of view

Set of MLPs with a given size:

$$\mathcal{P}^Q(h) = \left\{ x \in \mathbb{R}^p \rightarrow \sum_{k=1}^{Q} w_k^{(2)} h\left(x^\top w_k^{(1)} + w_k^{(0)}\right) + w_0^{(2)} \,:\; w_k^{(2)},\, w_k^{(0)} \in \mathbb{R},\, w_k^{(1)} \in \mathbb{R}^p \right\}$$

Set of all MLPs: $\mathcal{P}(h) = \cup_{Q \in \mathbb{N}} \mathcal{P}^Q(h)$

Universal property from a theoretical point of view

Set of MLPs with a given size:

$$\mathcal{P}^Q(h) = \left\{ x \in \mathbb{R}^p \rightarrow \sum_{k=1}^{Q} w_k^{(2)} h\left(x^\top w_k^{(1)} + w_k^{(0)}\right) + w_0^{(2)} : \; w_k^{(2)}, w_k^{(0)} \in \mathbb{R}, \; w_k^{(1)} \in \mathbb{R}^p \right\}$$

Set of all MLPs: $\mathcal{P}(h) = \cup_{Q \in \mathbb{N}} \mathcal{P}^Q(h)$

Universal approximation [Pinkus, 1999]

If h is a non polynomial continuous function, then, for any continuous function $g : [0,1]^p \rightarrow \mathbb{R}$ and any $\epsilon > 0$, $\exists f \in \mathcal{P}(h)$ such that:

$$\sup_{x \in [0,1]^p} |f(x) - g(x)| \leq \epsilon.$$

Remarks on universal approximation

- continuity of the activation function is not required (see [Devroye et al., 1996] for a result with arbitrary sigmoids)
- other versions of this property are given in [Hornik, 1991, Hornik, 1993, Stinchcombe, 1999] for different functional spaces for g
- none of the spaces $\mathcal{P}^Q(h)$, for a fixed Q, has this property

MLP from a statistical learning perspective

Set of MLPs with a given size:

$$\mathcal{P}^Q(h) = \left\{ x \in \mathbb{R}^p \to \sum_{k=1}^{Q} w_k^{(2)} h\left(x^\top w_k^{(1)} + w_k^{(0)}\right) + w_0^{(2)} :\; w_k^{(2)},\, w_k^{(0)} \in \mathbb{R},\, w_k^{(1)} \in \mathbb{R}^p \right\}$$

Set of all MLPs: $\mathcal{P}(h) = \cup_{Q\in\mathbb{N}} \mathcal{P}^Q(h)$

In a binary classification framework, set of decision functions:
$\mathcal{F}^Q(h) = \left\{ f :\; x \in \mathbb{R}^p \to \mathbb{R} : \exists \psi \in \mathcal{P}^Q(h) \text{ st } f(x) = \mathbb{1}_{\{\psi(x) > 1/2\}} \right\}$

MLP from a statistical learning perspective

Set of MLPs with a given size:

$$\mathcal{P}^Q(h) = \left\{ x \in \mathbb{R}^p \to \sum_{k=1}^{Q} w_k^{(2)} h\left(x^\top w_k^{(1)} + w_k^{(0)}\right) + w_0^{(2)} :\ w_k^{(2)}, w_k^{(0)} \in \mathbb{R}, w_k^{(1)} \in \mathbb{R}^p \right\}$$

Set of all MLPs: $\mathcal{P}(h) = \cup_{Q\in\mathbb{N}} \mathcal{P}^Q(h)$

In a binary classification framework, set of decision functions:
$\mathcal{F}^Q(h) = \left\{ f :\ x \in \mathbb{R}^p \to \mathbb{R} : \exists\, \psi \in \mathcal{P}^Q(h) \text{ st } f(x) = \mathbb{1}_{\{\psi(x) > 1/2\}} \right\}$

If (X, Y) are random variables in $\mathcal{R}^p \times \{0, 1\}$

- Risk associated with $f \in \mathcal{F}^Q(h)$: $\mathcal{R}_P(f) = \mathbb{P}(f(X) \neq Y)$
- Best achievable performance: $\mathcal{R}_P^* = \inf_{f:\ \mathbb{R}^p \to \{0,1\}} \mathbb{P}(f(X) \neq Y)$

How do these risks compare?

MLPs can have a low risk

From the universal approximation property, we can obtain:

[Devroye et al., 1996]

If h is a sigmoid then

$$\lim_{Q \to +\infty} \inf_{f \in \mathcal{F}^Q(h)} \mathcal{R}_P(f) - \mathcal{R}_P^* = 0.$$

Universal consistency

Learning set: $(X_i, Y_i)_{i=1,\ldots,n}$ are i.i.d. observations of (X, Y)

How to choose an accurate perceptron within $\mathcal{F}^Q(h)$?

Universal consistency

Learning set: $(X_i, Y_i)_{i=1,\ldots,n}$ are i.i.d. observations of (X, Y)

How to choose an accurate perceptron within $\mathcal{F}^Q(h)$? Suppose that we can minimize the empirical classification error:

$$\hat{f}_{\mathcal{F}^Q(h)} := \arg\min_{f \in \mathcal{F}^Q(h)} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{f(X_i) \neq Y_i\}}$$

Universal consistency

Learning set: $(X_i, Y_i)_{i=1,\ldots,n}$ are i.i.d. observations of (X, Y)

How to choose an accurate perceptron within $\mathcal{F}^Q(h)$? Suppose that we can minimize the empirical classification error:

$$\hat{f}_{\mathcal{F}^Q(h)} := \arg\min_{f\in\mathcal{F}^Q(h)} \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{f(X_i)\neq Y_i\}}$$

[Farago and Lugosi, 1993]

If h is the Heaviside activation function, then for $Q \rightarrow +\infty$ and $(Q\log n)/n \xrightarrow{n\rightarrow+\infty} 0$, we have that

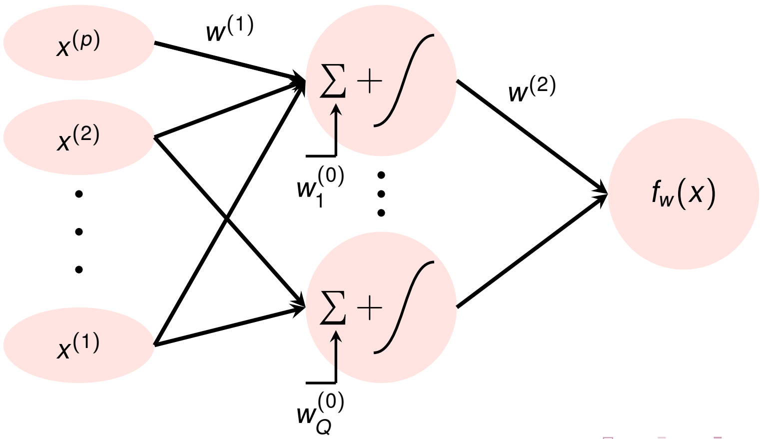
$$\lim_{n\rightarrow+\infty} \mathcal{R}_P\left(\hat{f}_{\mathcal{F}^Q(h)}\right) = \mathcal{R}_P^*.$$

Remarks on universal consistency

- Q (which controls the complexity of MLP) must increase with n
- other results (with more general sigmoids) have been proved (see [Devroye et al., 1996] for a review)
- similar consistency results have also been proved in the regression framework as well as rates of convergence (see [White, 1990, White, 1991, Barron, 1994, McCaffrey and Gallant, 1994])

Empirical error minimization

Given i.i.d. observations of (X, Y), (X_i, Y_i), how to choose the weights w?



Empirical error minimization

Given i.i.d. observations of (X, Y), (X_i, Y_i), how to choose the weights w?

Standard approach: minimize the empirical L_2 risk:

$$\widehat{\mathcal{R}}_n(w) = \sum_{i=1}^{n} [f_w(X_i) - Y_i]^2$$

with

- $Y_i \in \mathbb{R}$ for the regression case
- $Y_i \in \{0, 1\}$ for the classification case, with the associated decision rule $x \rightarrow \mathbb{1}_{\{f_w(x) \leq 1/2\}}$.

Empirical error minimization

Given i.i.d. observations of (X, Y), (X_i, Y_i), how to choose the weights w?

Standard approach: minimize the empirical L_2 risk:

$$\widehat{\mathcal{R}}_n(w) = \sum_{i=1}^{n} [f_w(X_i) - Y_i]^2$$

with

- $Y_i \in \mathbb{R}$ for the regression case
- $Y_i \in \{0, 1\}$ for the classification case, with the associated decision rule $x \to \mathbb{1}_{\{f_w(x) \leq 1/2\}}$.

But: $\hat{\mathcal{R}}_n(w)$ is not convex in w $\Rightarrow$ general optimization problem

Optimization with gradient descent

Method: initialize (randomly or with some prior knowledge) the weights $w(0) \in \mathbb{R}^{Qp+2Q+1}$

- Batch approach: for $t = 1, \ldots, T$

$$w(t+1) = w(t) - \mu(t)\nabla_w \hat{\mathcal{R}}_n(w(t));$$

Optimization with gradient descent

Method: initialize (randomly or with some prior knowledge) the weights $w(0) \in \mathbb{R}^{Qp+2Q+1}$

- Batch approach: for $t = 1, \ldots, T$

$$w(t+1) = w(t) - \mu(t)\nabla_w \hat{\mathcal{R}}_n(w(t));$$

- online (or stochastic) approach: write

$$\hat{\mathcal{R}}_n(w) = \sum_{i=1}^{n} \underbrace{[f_w(X_i) - Y_i]^2}_{=E_i}$$

and for $t = 1, \ldots, T$, randomly pick $i \in \{1, \ldots, n\}$ and update:

$$w(t+1) = w(t) - \mu(t)\nabla_w E_i(w(t)).$$

Discussion about practical choices for this approach

- batch version converges (in an optimization point of view) to a local minimum of the error for a good choice of $\mu(t)$ but convergence can be slow
- stochastic version is usually very inefficient but is useful for large datasets (*n* large)
- more efficient algorithms exist to solve the optimization task. The one implemented in the R package **nnet** uses higher order derivatives (BFGS algorithm)
- in all cases, solutions returned are, at best, local minima which strongly depends on the initialization: using more than one initialization state is advised

Gradient backpropagation method

[Rumelhart and Mc Clelland, 1986]

The gradient backpropagation *rétropropagation du gradient* principle is used to easily calculate gradients in perceptrons (or in other types of neural network):

Gradient backpropagation method

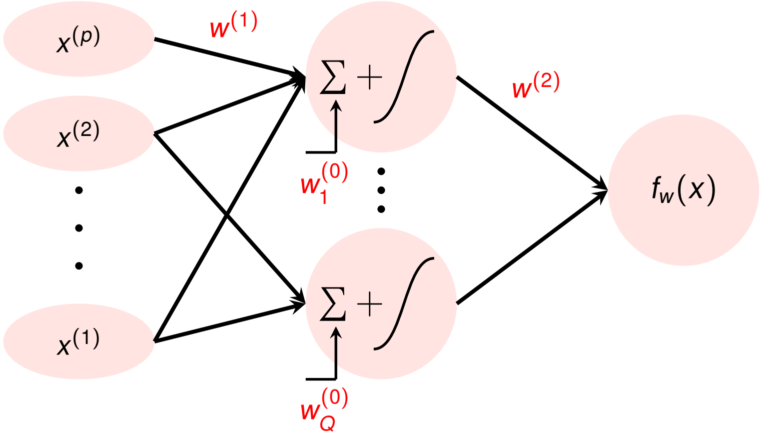
[Rumelhart and Mc Clelland, 1986]

The gradient backpropagation *rétropropagation du gradient* principle is used to easily calculate gradients in perceptrons (or in other types of neural network):

This way, stochastic gradient descent alternates:

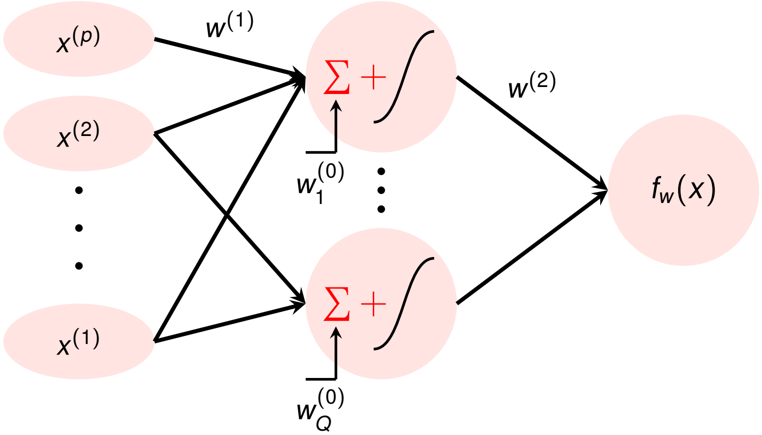
- a forward step which aims at calculating outputs from all observations X_i given a value of the weights w
- a backward step in which the gradient backpropagation principle is used to obtain the gradient for the current weights w

Backpropagation in practice



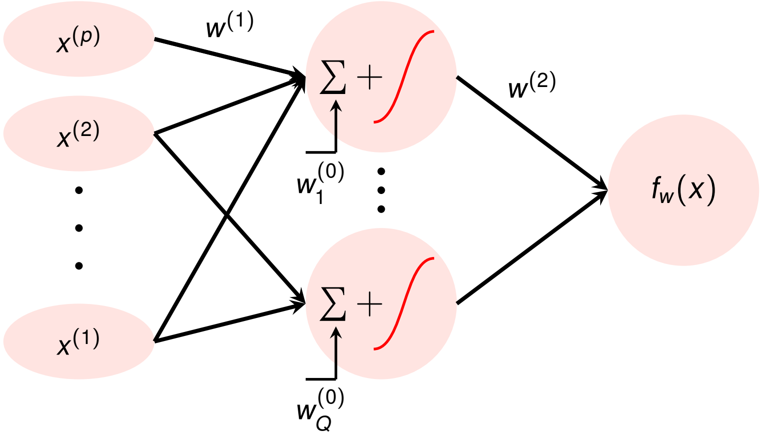
initialize weights

Backpropagation in practice



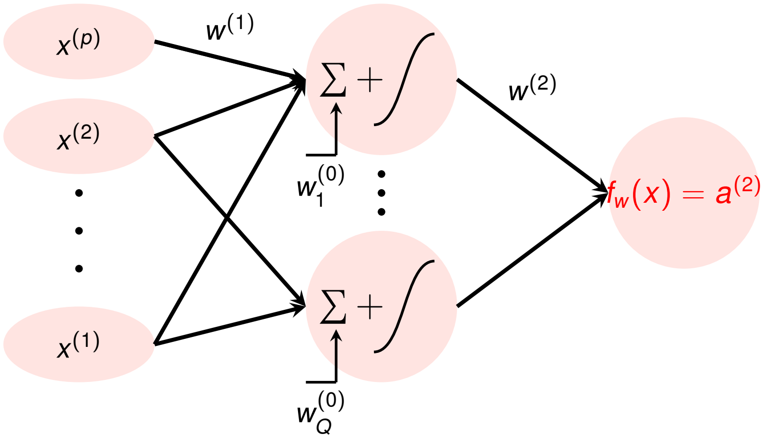
Forward step: for all k, calculate $a_k^{(1)} = X_i^\top w_k^{(1)} + w_k^{(0)}$

Backpropagation in practice



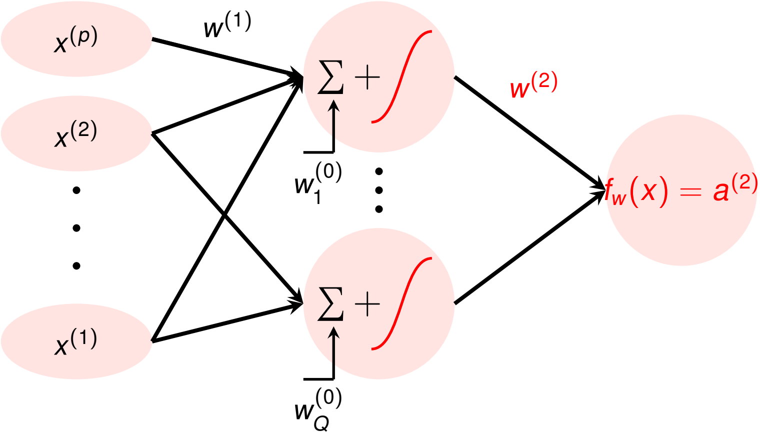
Forward step: for all k, calculate $z_k^{(1)} = h_k(a_k^{(1)})$

Backpropagation in practice



Forward step: calculate $a^{(2)} = \sum_{k=1}^{Q} w_k^{(2)} z_k^{(1)} + w_0^{(2)}$

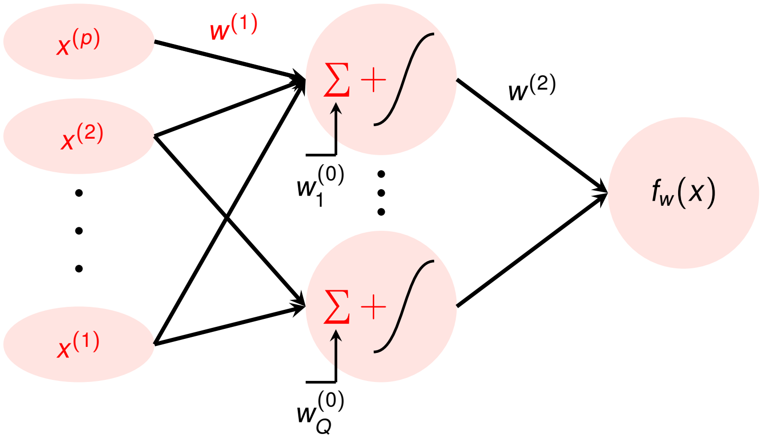
Backpropagation in practice



Backward step: calculate $\frac{\partial E_i}{\partial w_k^{(2)}} = \delta^{(2)} \times z_k$ with

$$\delta^{(2)} = \frac{\partial E_i}{\partial a^{(2)}} = \frac{[h_0(a^{(2)}) - Y_i]^2}{\partial a^{(2)}} = 2h_0'(a^{(2)}) \times [h_0(a^{(2)}) - Y_i]$$

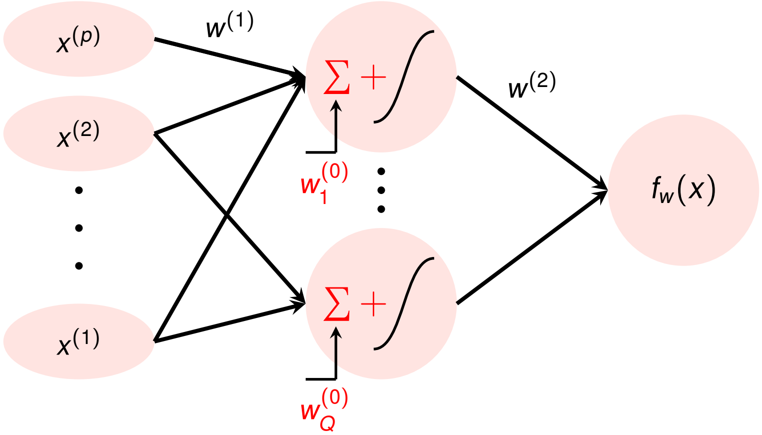
Backpropagation in practice



Backward step: $\frac{\partial E_i}{\partial w_{kj}^{(1)}} = \delta_k^{(1)} \times X_i^{(j)}$ with

$$\delta_k^{(1)} = \frac{\partial E_i}{\partial a_k^{(1)}} = \frac{\partial E_i}{\partial a^{(2)}} \times \frac{\partial a^{(2)}}{\partial a_k^{(1)}} = \delta^{(2)} \times w_k^{(2)} h_k'(a_k^{(1)})$$

Backpropagation in practice



Backward step: $\frac{\partial E_i}{\partial w_k^{(0)}} = \delta_k^{(1)}$

Initialization and stopping of the training algorithm

1. How to initialize weights? Standard choices $w_{jk}^{(1)} \sim \mathcal{N}(0, 1/\sqrt{p})$ and $w_k^{(2)} \sim \mathcal{N}(0, 1/\sqrt{Q})$

Initialization and stopping of the training algorithm

1. How to initialize weights? Standard choices $w_{jk}^{(1)} \sim \mathcal{N}(0, 1/\sqrt{p})$ and $w_k^{(2)} \sim \mathcal{N}(0, 1/\sqrt{Q})$

2. When to stop the algorithm? (gradient descent or alike) Standard choices:
   - bounded T
   - target value of the error $\hat{\mathcal{R}}_n(w)$
   - target value of the evolution $\hat{\mathcal{R}}_n(w(t)) - \hat{\mathcal{R}}_n(w(t+1))$

Initialization and stopping of the training algorithm

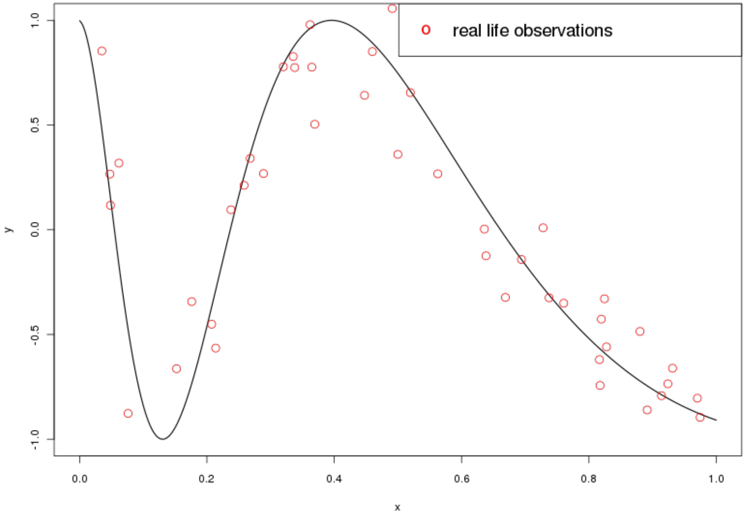
1. How to initialize weights? Standard choices $w_{jk}^{(1)} \sim \mathcal{N}(0, 1/\sqrt{p})$ and $w_k^{(2)} \sim \mathcal{N}(0, 1/\sqrt{Q})$
   In the R package **nnet**, weights are sampled uniformly between $[-0.5, 0.5]$ or between $\left[-\frac{1}{\max_i X_i^{(j)}}, \frac{1}{\max_i X_i^{(j)}}\right]$ if $X^{(j)}$ is large.

2. When to stop the algorithm? (gradient descent or alike) Standard choices:
   - bounded T
   - target value of the error $\hat{\mathcal{R}}_n(w)$
   - target value of the evolution $\hat{\mathcal{R}}_n(w(t)) - \hat{\mathcal{R}}_n(w(t+1))$

   In the R package **nnet**, a combination of the three criteria is used and tunable.

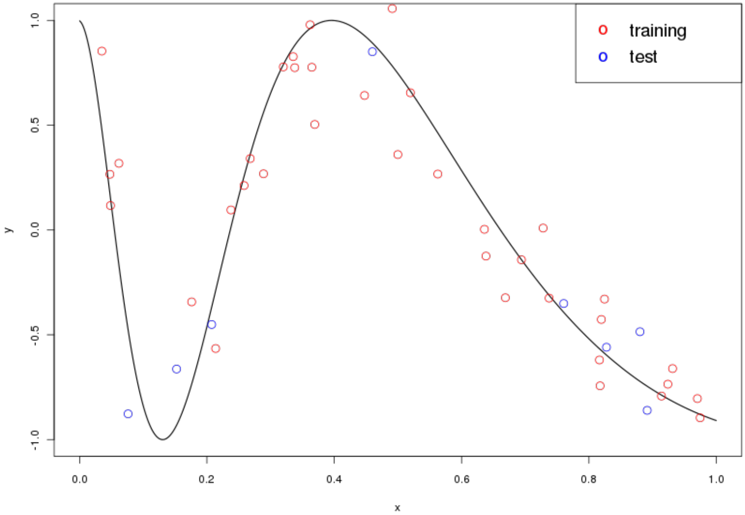
Avoid overfitting: do not trust empirical risk minimization

Observations



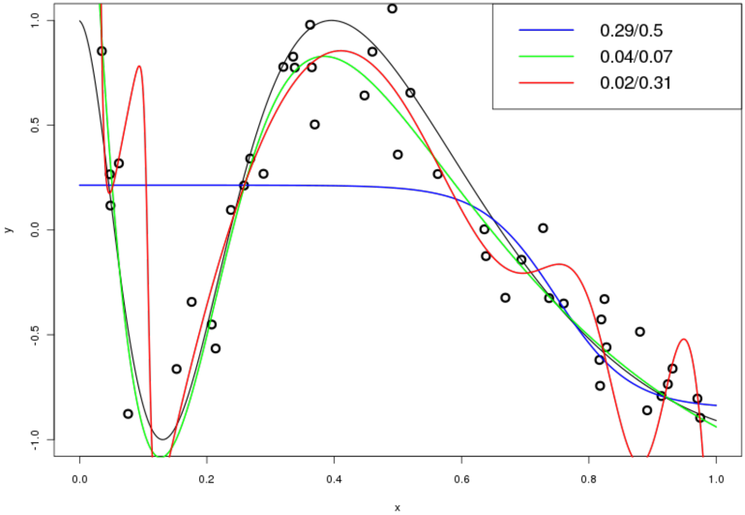
Avoid overfitting: do not trust empirical risk minimization

Training/Test datasets



Avoid overfitting: do not trust empirical risk minimization

Training/Test errors



Strategies to avoid overfitting

- Properly tune Q with a CV or a bootstrap estimation of the generalization ability of the method

Strategies to avoid overfitting

- Properly tune Q with a CV or a bootstrap estimation of the generalization ability of the method
- Early stopping: for Q large enough, use a part of the data as a validation set and stops the training (gradient descent) when the empirical error calculated on this dataset starts to increase

Strategies to avoid overfitting

- Properly tune Q with a CV or a bootstrap estimation of the generalization ability of the method

- Early stopping: for Q large enough, use a part of the data as a validation set and stops the training (gradient descent) when the empirical error calculated on this dataset starts to increase

- Weight decay: for Q large enough, penalize the empirical risk with a function of the weights, *e.g.*,

$$\hat{\mathcal{R}}_n(w) + \lambda w^\top w$$

Strategies to avoid overfitting

- Properly tune Q with a CV or a bootstrap estimation of the generalization ability of the method

- Early stopping: for Q large enough, use a part of the data as a validation set and stops the training (gradient descent) when the empirical error calculated on this dataset starts to increase

- Weight decay: for Q large enough, penalize the empirical risk with a function of the weights, *e.g.*,

$$\hat{\mathcal{R}}_n(w) + \lambda w^\top w$$

- Noise injection: modify the input data with a random noise during the training

Sommaire

Software description

Use cases (simulated and real data) are illustrated with the R package **nnet** [Ripley, 1996]. Different types of single layer neural networks are implemented in this package.

Software description

Use cases (simulated and real data) are illustrated with the R package **nnet** [Ripley, 1996]. Different types of single layer neural networks are implemented in this package.

1 layer MLP (function `nnet`) have the following options (among others):

- number of neurons on the hidden layer `size`;
- initial values of the weights `Wts`. If not provided, weights are initialized randomly with a uniform distribution in $[-0.5, 0.5]$ or $\left[-\frac{1}{\max_i |x_i^{(j)}|}, \frac{1}{\max_i |x_i^{(j)}|}\right]$ if $\max_i |x_i^{(j)}|$ is "large". Argument `rang` can be used to initialized weights between [−`rang`, `rang`];
- maximum number of iterations, objective error and objective evolution of the error `maxit` (default to 100), `abstol` and `reltol`;
- is the output activation function a logistic sigmoid (default) or the identity (`linout = TRUE`);
- a regularization parameter for the weight decay `decay` (default to 0).

Software description

Use cases (simulated and real data) are illustrated with the R package **nnet** [Ripley, 1996]. Different types of single layer neural networks are implemented in this package.

1 layer MLP (function `nnet`) have the following options (among others):

- number of neurons on the hidden layer `size`;
- initial values of the weights `Wts`. If not provided, weights are initialized randomly with a uniform distribution in $[-0.5, 0.5]$ or $\left[-\frac{1}{\max_i |x_i^{(j)}|}, \frac{1}{\max_i |x_i^{(j)}|}\right]$ if $\max_i |x_i^{(j)}|$ is "large". Argument `rang` can be used to initialized weights between [−`rang`, `rang`];
- maximum number of iterations, objective error and objective evolution of the error `maxit` (default to 100), `abstol` and `reltol`;
- is the output activation function a logistic sigmoid (default) or the identity (`linout = TRUE`);
- a regularization parameter for the weight decay `decay` (default to 0).

e1071 has a convenient wrapper of the function `nnet` to tune hyperparameters: `tune.nnet`.

Take your laptop and start R!

References

Barron, A. (1994).
Approximation and estimation bounds for artificial neural networks.
Machine Learning, 14:115–133.

Bishop, C. (1995).
Neural Networks for Pattern Recognition.
Oxford University Press, New York, USA.

Devroye, L., Györfi, L., and Lugosi, G. (1996).
A Probabilistic Theory for Pattern Recognition.
Springer-Verlag, New York, NY, USA.

Farago, A. and Lugosi, G. (1993).
Strong universal consistency of neural network classifiers.
IEEE Transactions on Information Theory, 39(4):1146–1151.

Györfi, L., Kohler, M., Krzyżak, A., and Walk, H. (2002).
A Distribution-Free Theory of Nonparametric Regression.
Springer-Verlag, New York, NY, USA.

Hornik, K. (1991).
Approximation capabilities of multilayer feedfoward networks.
Neural Networks, 4(2):251–257.

Hornik, K. (1993).
Some new results on neural network approximation.
Neural Networks, 6(8):1069–1072.

Mc Culloch, W. and Pitts, W. (1943).
A logical calculus of ideas immanent in nervous activity.
Bulletin of Mathematical Biophysics, 5(4):115–133.

McCaffrey, D. and Gallant, A. (1994).

Convergence rates for single hidden layer feedforward networks.
Neural Networks, 7(1):115–133.

Pinkus, A. (1999).
Approximation theory of the MLP model in neural networks.
Acta Numerica, 8:143–195.

Ripley, B. (1996).
Pattern Recognition and Neural Networks.
Cambridge University Press.

Rosenblatt, F. (1958).
The perceptron: a probabilistic model for information storage and organization in the brain.
Psychological Review, 65:386–408.

Rosenblatt, F. (1962).
Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.
Spartan Books, Washington, DC, USA.

Rumelhart, D. and Mc Clelland, J. (1986).
Parallel Distributed Processing: Exploration in the MicroStructure of Cognition.
MIT Press, Cambridge, MA, USA.

Stinchcombe, M. (1999).
Neural network approximation of continuous functionals and continuous functions on compactifications.
Neural Network, 12(3):467–477.

White, H. (1990).
Connectionist nonparametric regression: multilayer feedforward networks can learn arbitraty mappings.
Neural Networks, 3:535–549.

White, H. (1991).
Nonparametric estimation of conditional quantiles using neural networks.

In *Proceedings of the 23rd Symposium of the Interface: Computing Science and Statistics*, pages 190–199, Alexandria, VA, USA. American Statistical Association.