4. NEURAL NETWORKS TRAINING

Stephan Robert-Nicoud
HEIG-VD/HES-SO

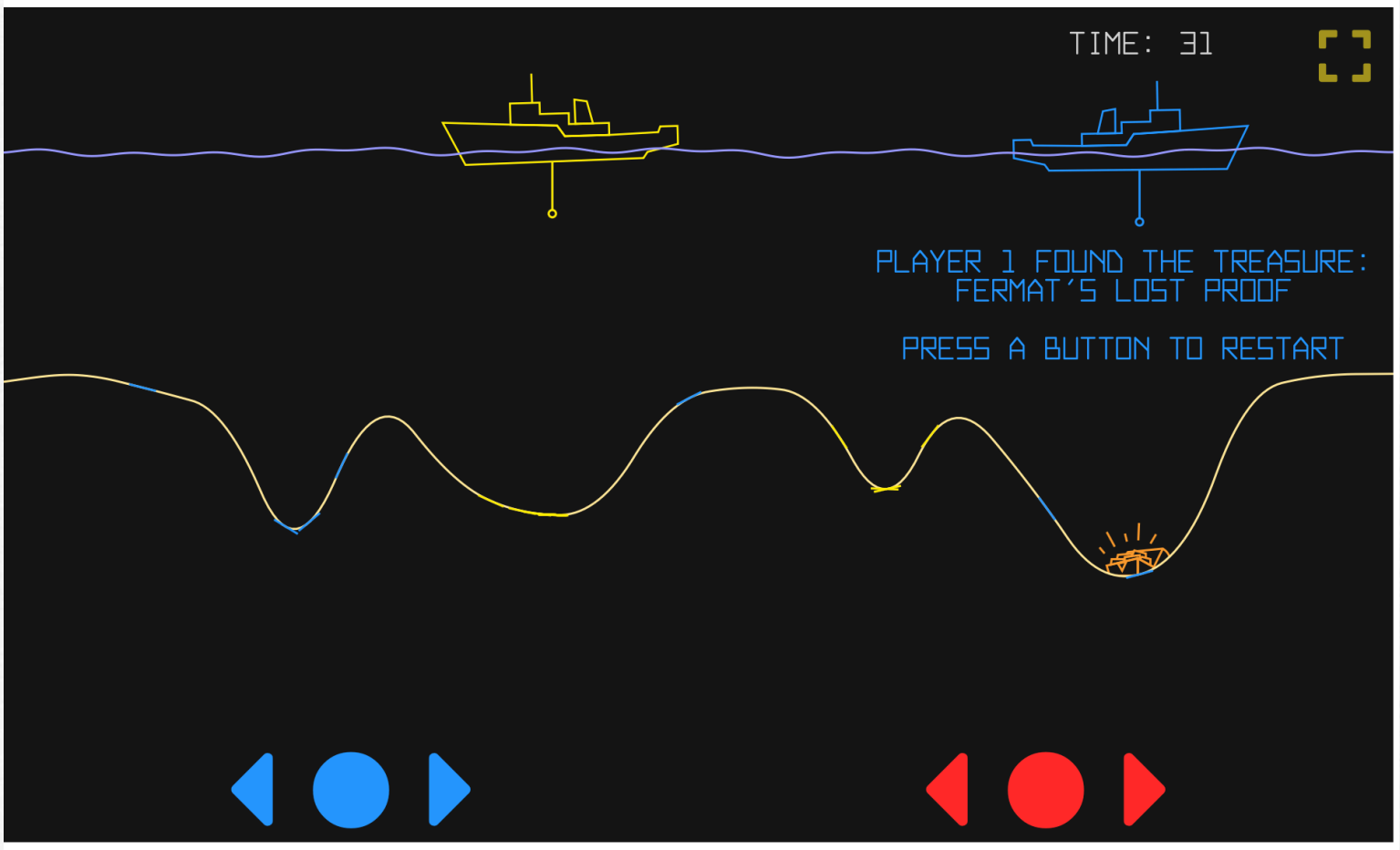
Credit: Andres Perez-Uribe

APPRENTISSAGE PAR RESEAUX DE NEURONES ARTIFICIELS

Objectives

- Understand the problem of minimizing a complex loss function
- Understand the consequences of overfitting a neural network
- Understand the mechanisms that can help us avoid overfitting
- Apply the "generic" Machine-Learning model selection methodology to neural networks using training, validation, and test datasets

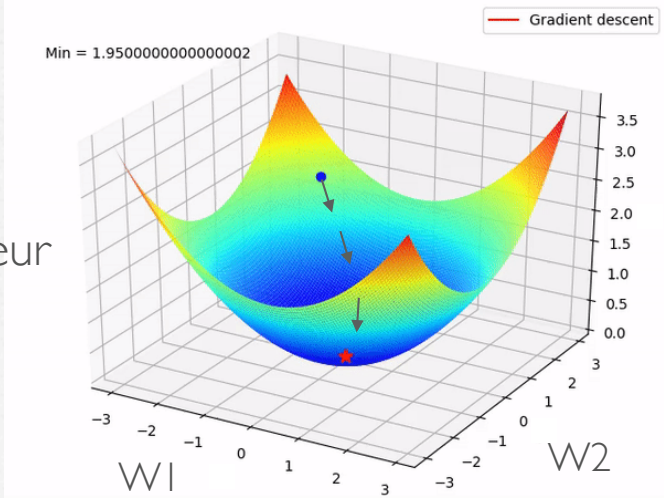
The Gradient descent game

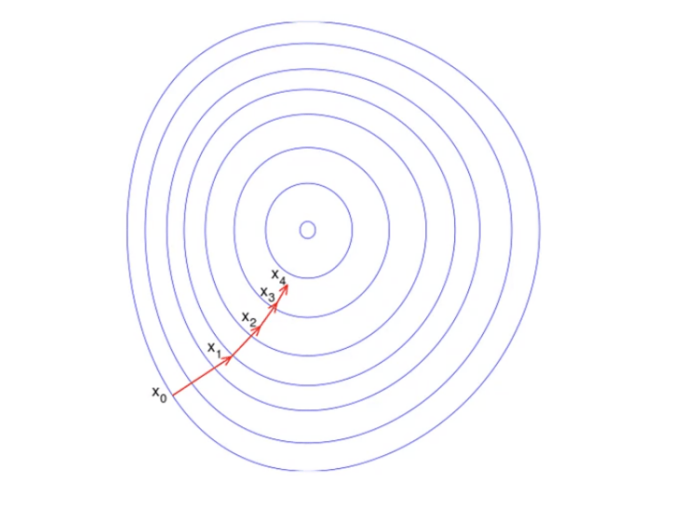


https://www.i-am.ai/gradient-descent.html

The path towards the minimum

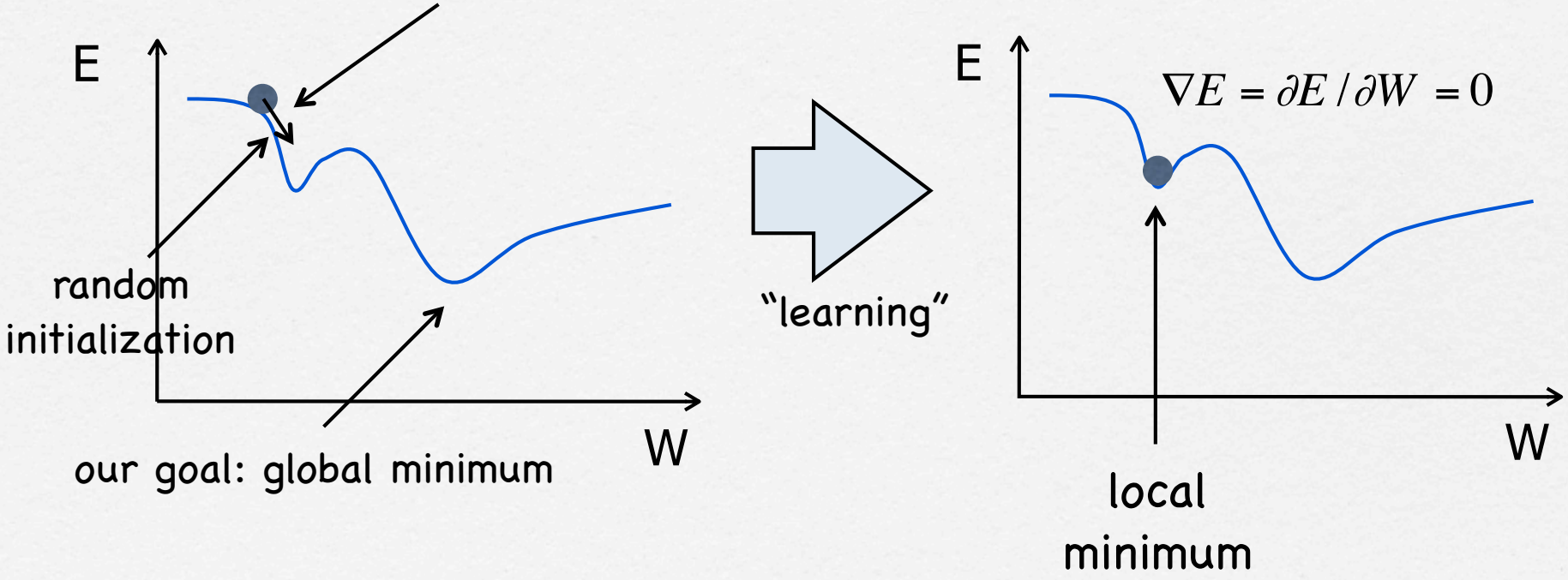
The gradient descent towards the minimum is often illustrated in 3D space [left] or using level curves (contour plots) [right] as follows:





Gradient descent by Backpropagation

$$\Delta w_{ij}^{(k)} = -\eta \partial E / \partial w_{ij}^{(k)}$$

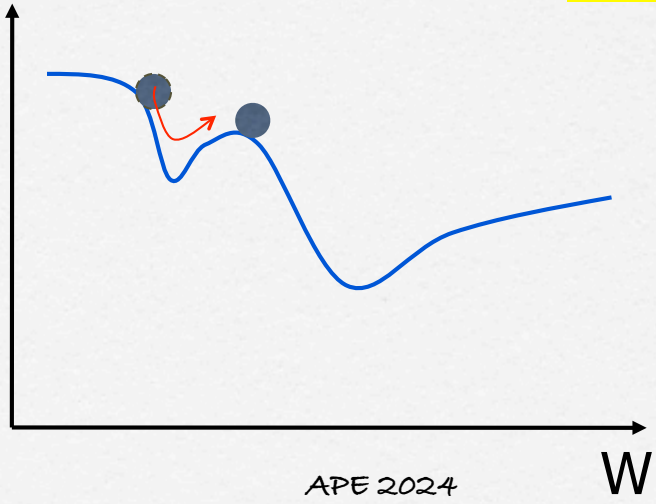


"Learning" stops when the derivative equals zero

Backpropagation with momentum

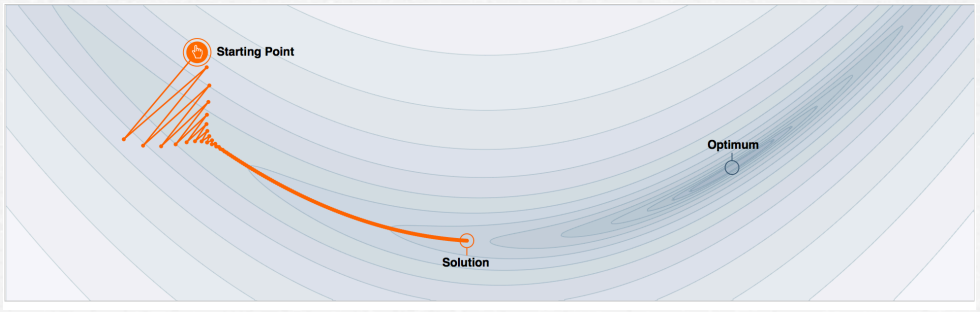
Momentum is a physical property that enables a particular object with mass to continue in it's trajectory even when an external opposing force is applied. In the context of neural networks, the idea behind this "trick" is to add a momentum to the weight adaptation.

$$\Delta w_{ij}^{(k)}(t) = -\eta \partial E / \partial w_{ij}^{(k)}(t) + \mu \Delta w_{ij}^{(k)}(t-1)$$

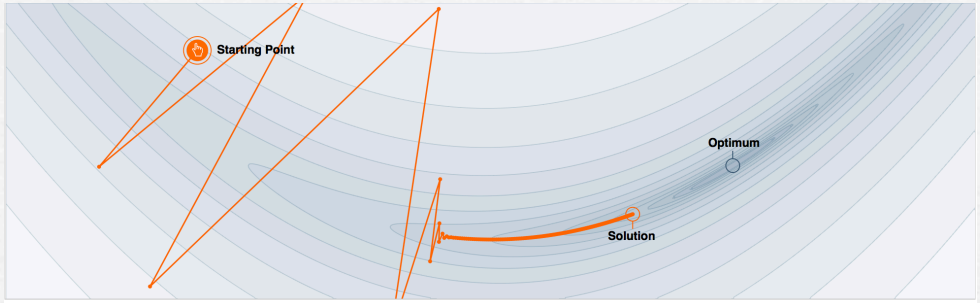


momentum term based on the previous weight modification (e.g., μ=0.9)

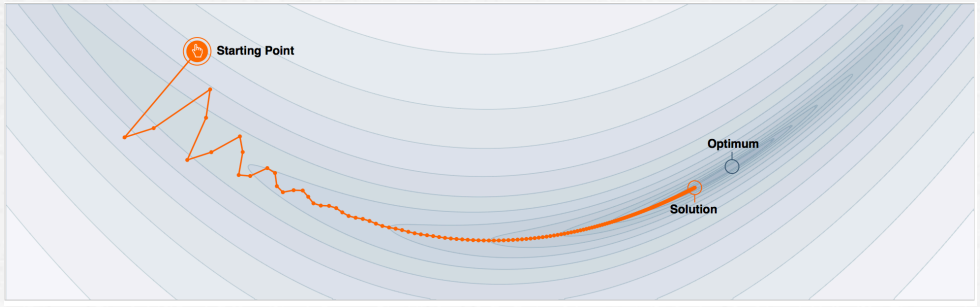
The momentum in action



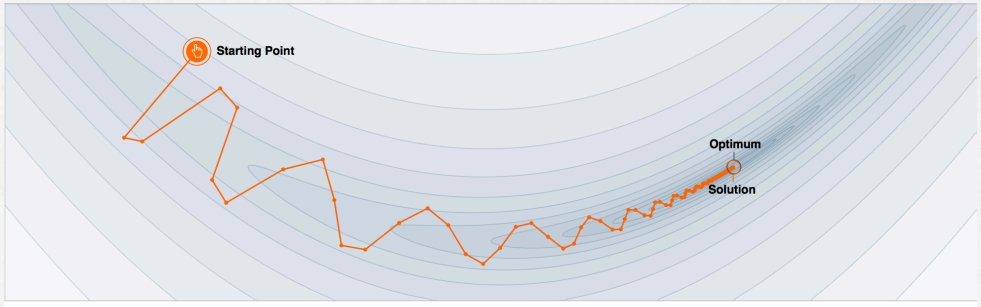
Learning rate = 0.003, Momentum = 0.0



Learning rate = 0.004, Momentum = 0.0



Learning rate = 0.003, Momentum = 0.7

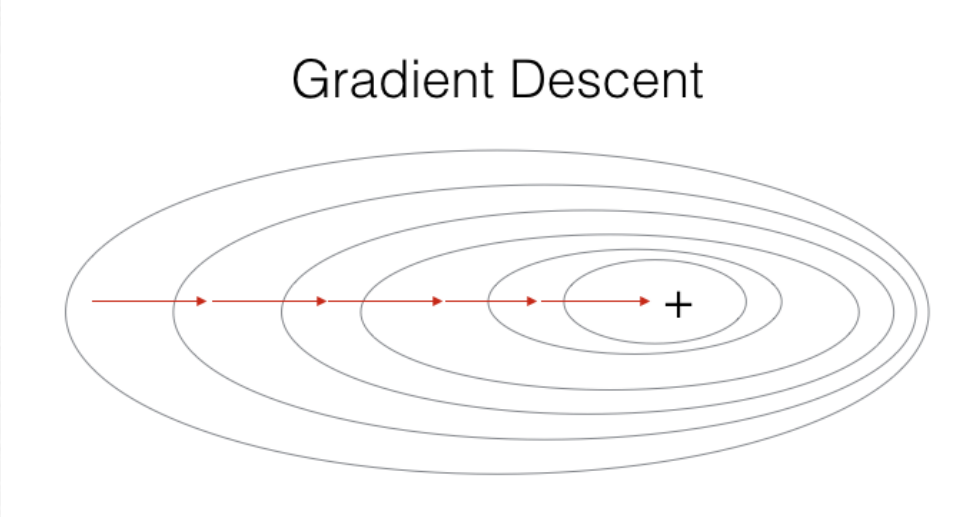


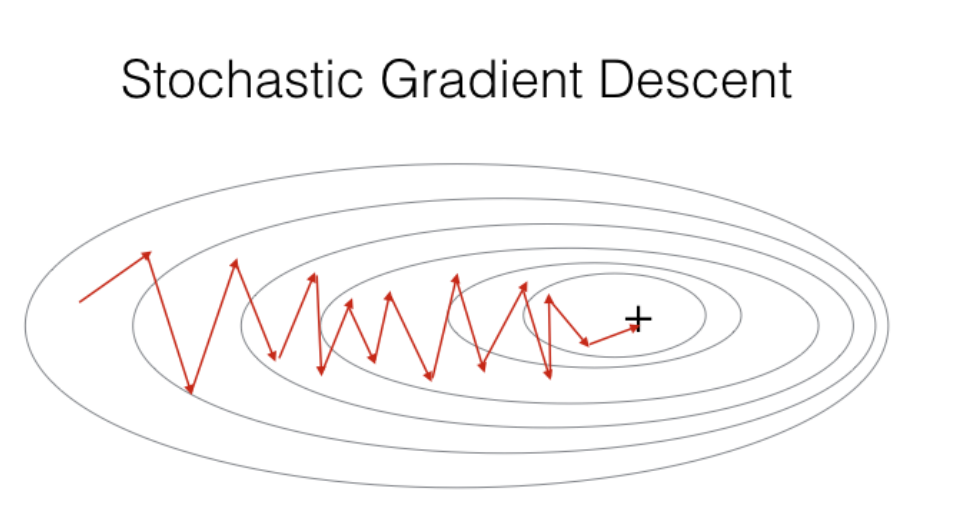
Learning rate = 0.003, Momentum = 0.85

https://distill.pub/2017/momentum/

Gradient computation flavors

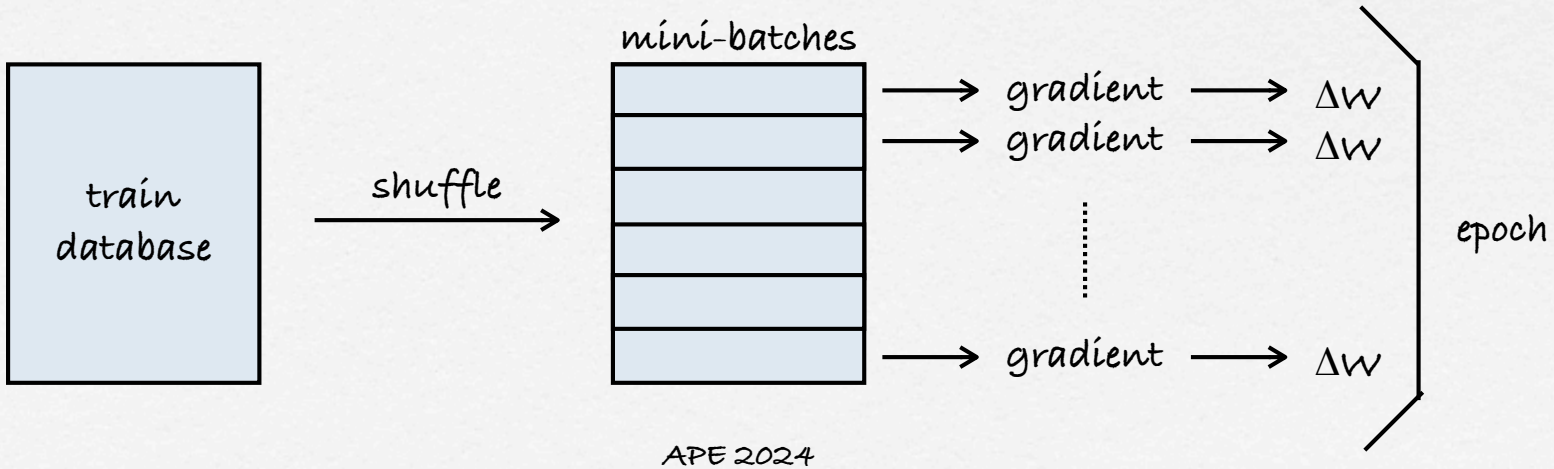
- Batch gradient descent: when the gradient is calculated from the entire data set
- Stochastic gradient descent: when the gradient is calculated from a single data sample.





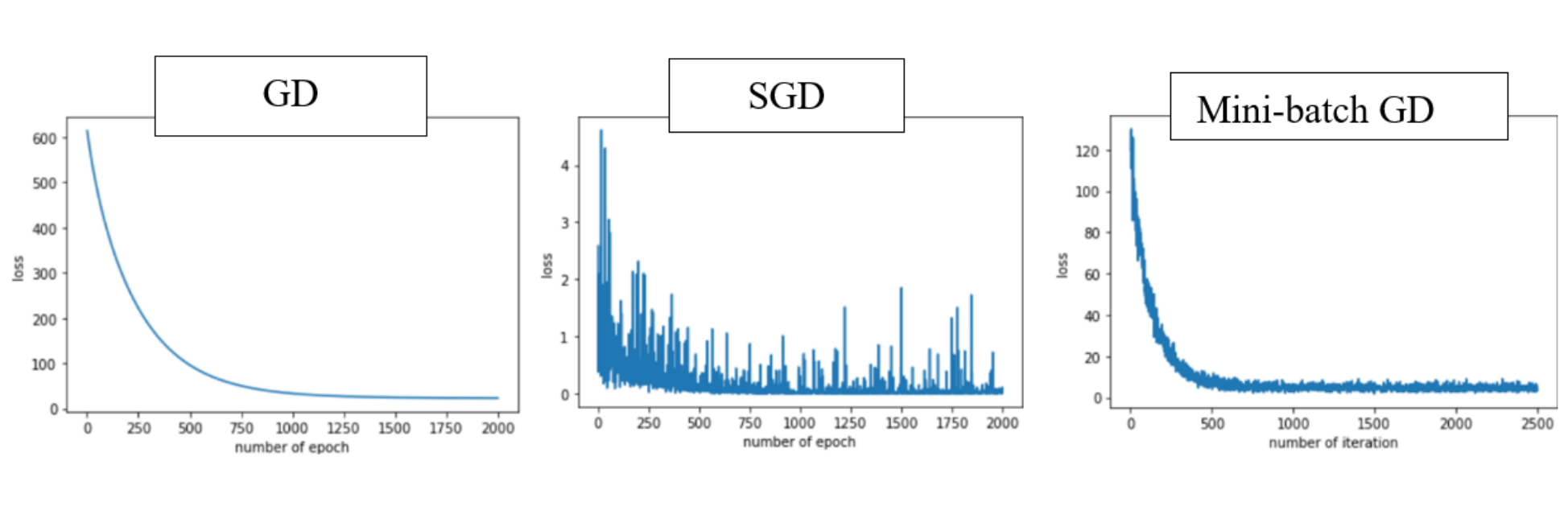
Mini-batch gradient descent

- In practice we train a neural network using a stochastic approximation of the exact gradient (calculated from the entire data set) by an estimate calculated from a randomly selected subset of the data, called a mini batch.
- Especially in high-dimensional optimization problems this reduces the very computational burden, achieving faster iterations in trade for a lower convergence rate.



Learning process: loss function minimization

- Evolution of the loss (e.g., mse) as a function of the number of epochs when using the exact gradient (GD), the Stochastic Gradient Descent (SGD) and the mini-batch* approach.



* the batch size is another hyper-parameter that can be tuned

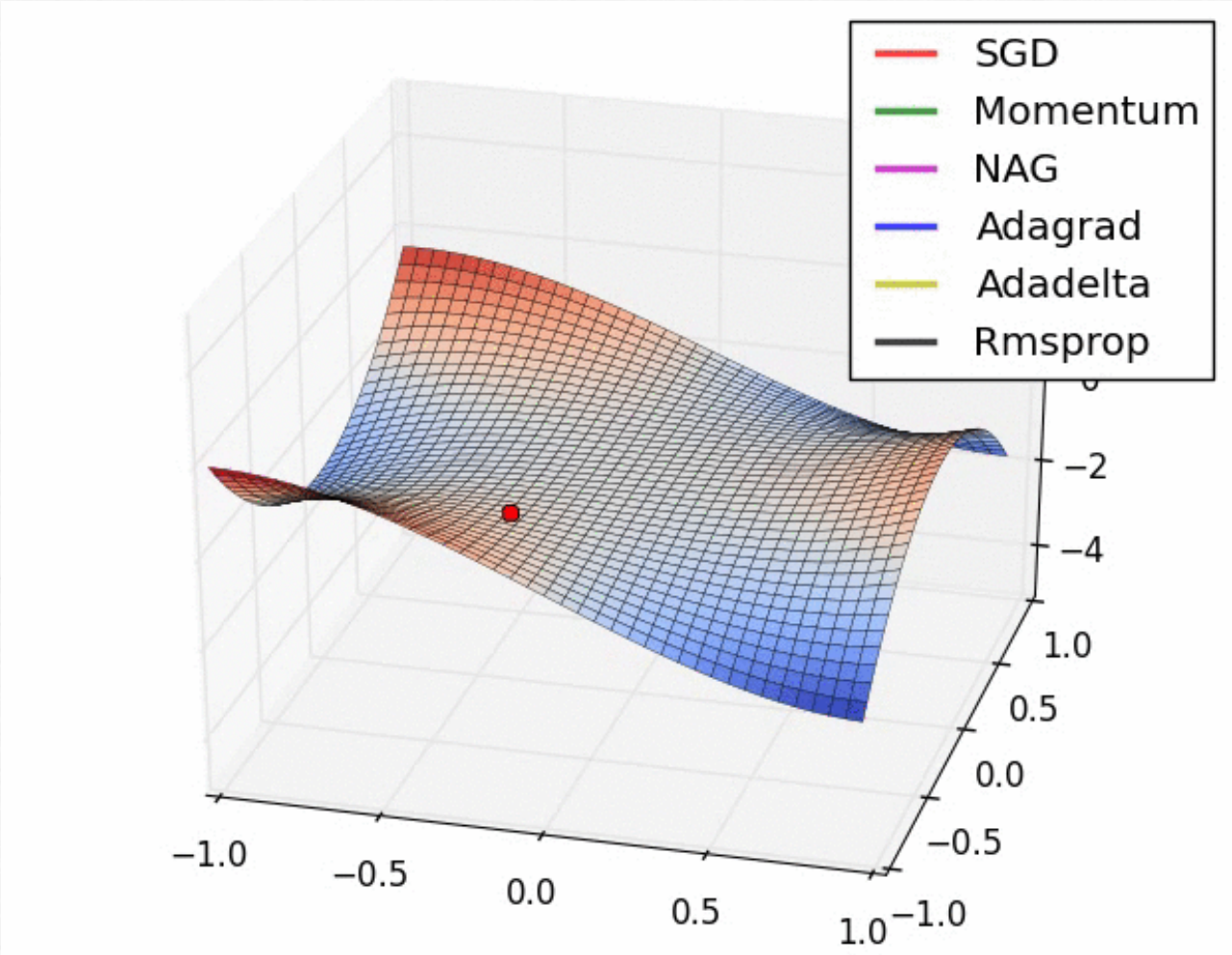
Gradient descent methods (1)

- Popular gradient descent methods are:
  - Stochastic Gradient Descent (SGD): it is basically mini-batch Backprop with momentum
  - Root Mean Square Propagation (RMSprop): I read somewhere that it is useful to establish a baseline of performance
  - Adam: Adaptive Moment Estimation:

$$
\begin{aligned}
m(w,t) &:= \gamma_1 * m(w,t-1) + (1-\gamma_1) * \nabla Q_i(w) \\
v(w,t) &:= \gamma_2 * v(w,t-1) + (1-\gamma_2) * (\nabla Q_i(w))^2 \\
\hat{m}(w,t) &:= \frac{m(w,t)}{(1-\gamma_1^t)} \\
\hat{v}(w,t) &:= \frac{v(w,t)}{(1-\gamma_2^t)} \\
w &:= w - \frac{\eta}{\sqrt{\hat{v}(w,t)} + \epsilon} * \hat{m}(w,t)
\end{aligned}
$$

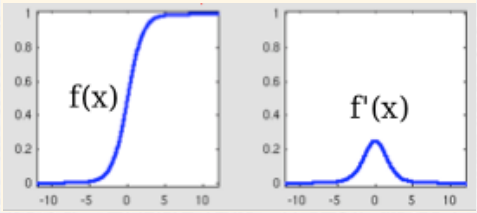
* ML libraires refer to these algorithms as optimizers

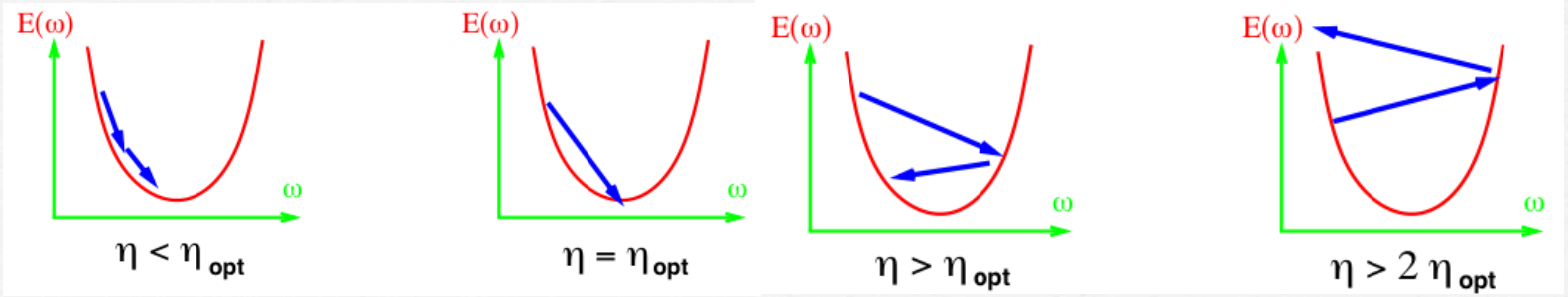
Gradient descent methods (2)



Practical considerations (1)

- Topology: number of layers, number of hidden neurons in each hidden layer (complexity of the model)
- Activity function: sigmoid, tanh, exp (Radial Basis Functions), ReLu
- Weight Initialisation: if too small, all units do the same, if too big, the sigmoid/tanh saturates
- Learning rate:





Y. LeCun, 2006

Practical considerations (2)

- Input normalization

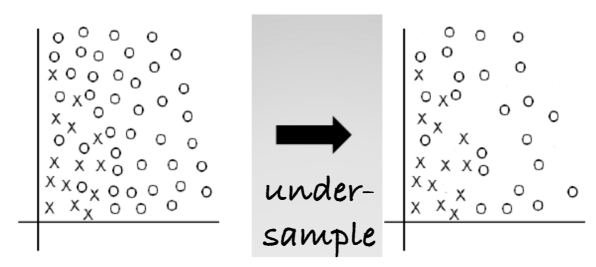
  Neural networks might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.

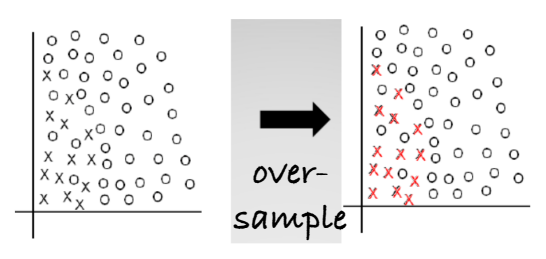
- Encoding categorical data

  e.g., given an input with two possible values high and low productivity
  -> use i-of-C coding: (1,0) and (0,1)

Practical considerations (3)

- Class imbalance problem: under-sample or over-sample to balance the set sizes for all classes.

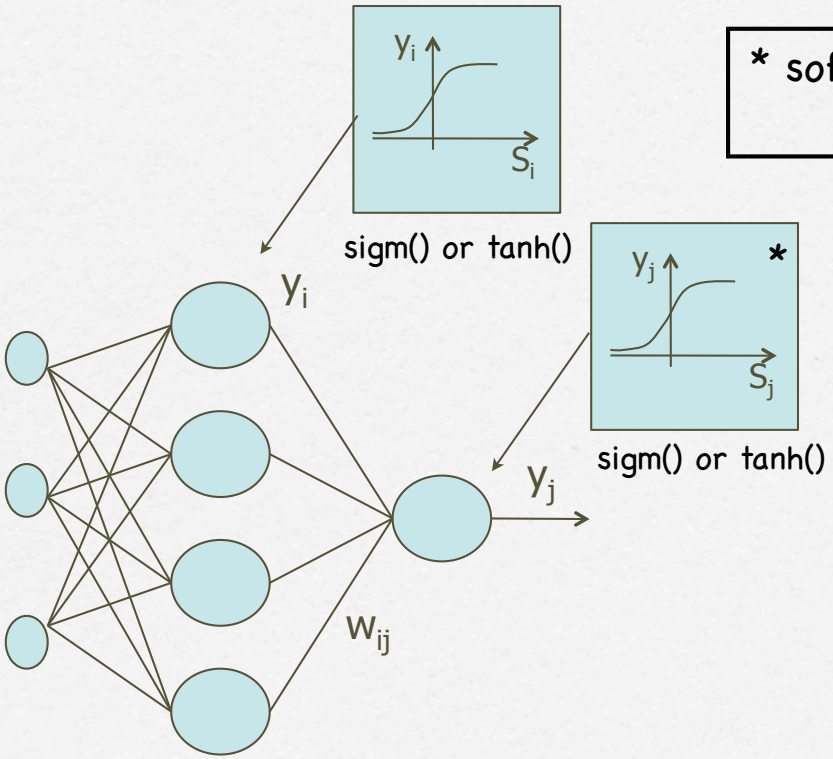




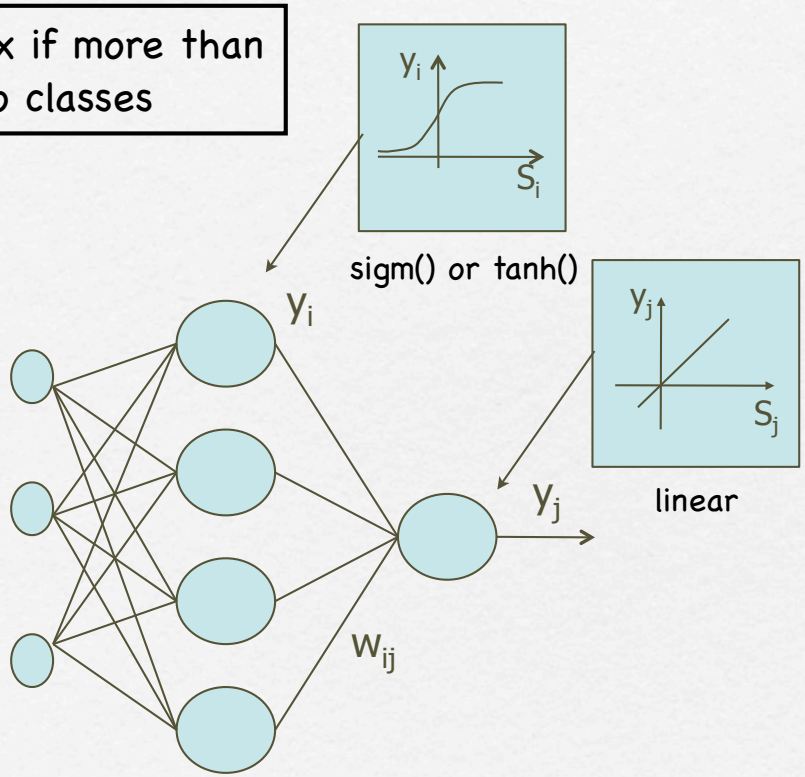
from Longadge et al, 2013

- weighted loss training: an alternative to deal with unbalanced classes is to give more credit to the minority class when computing the loss (implemented in the libraries)

Topologies for classification and regression



* softmax if more than two classes



classification network

loss: cross-entropy

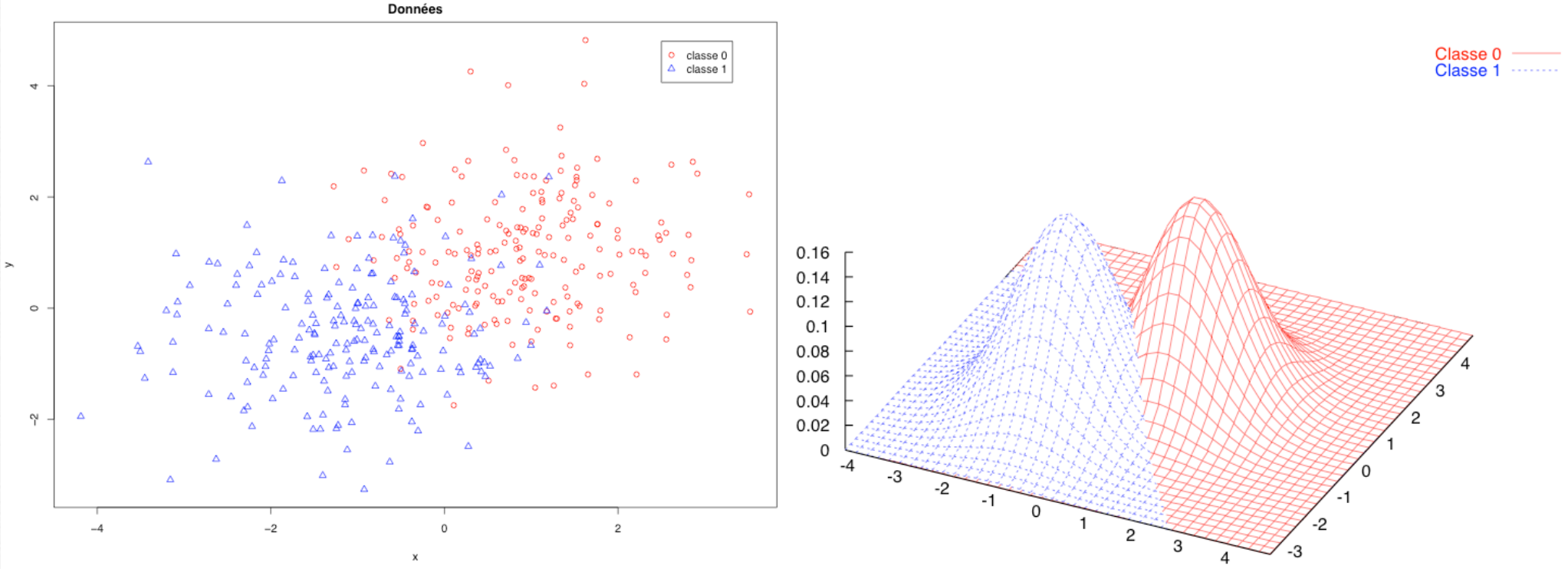
$CE = - \sum_p d_p \log(y_p)$

regression network

loss: mse

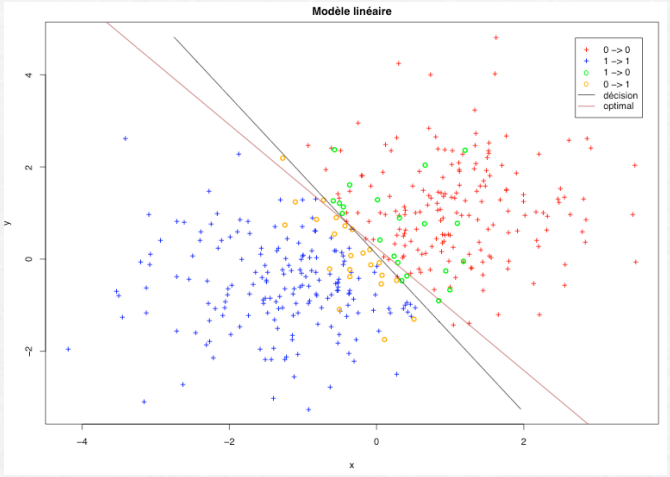
Effect of the number of hidden neurons (1)

the risk of overfitting

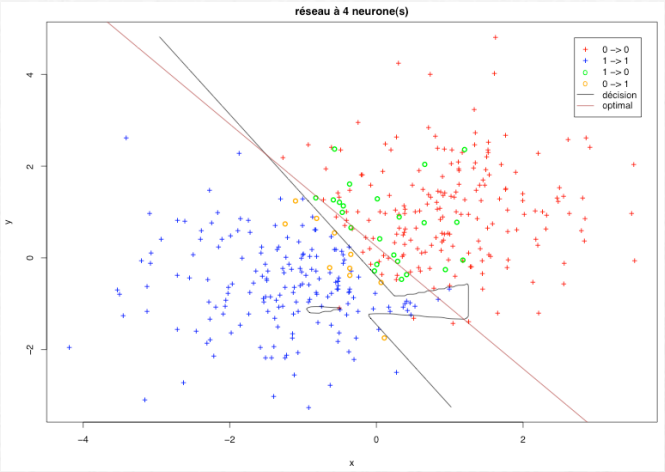


Two-class data points

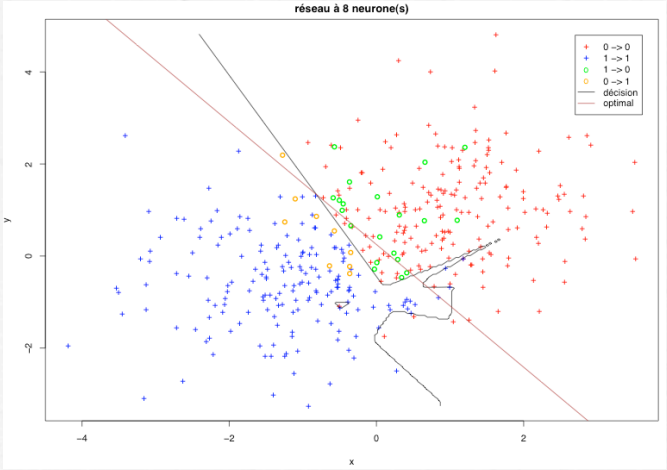
Effect of the number of hidden neurons (2)



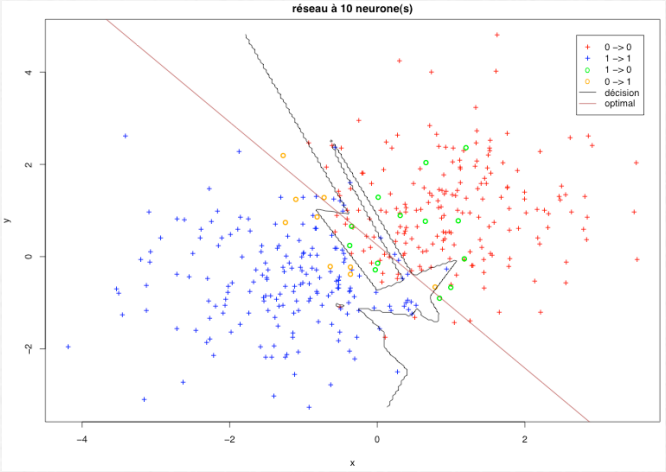
Linear classification



MLP 2-4-1

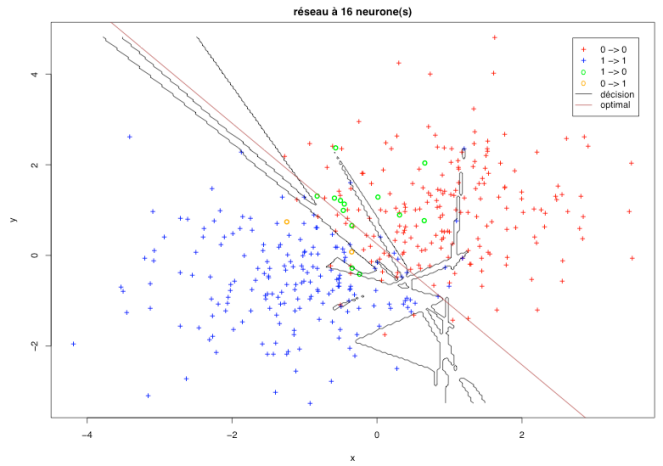


MLP 2-8-1

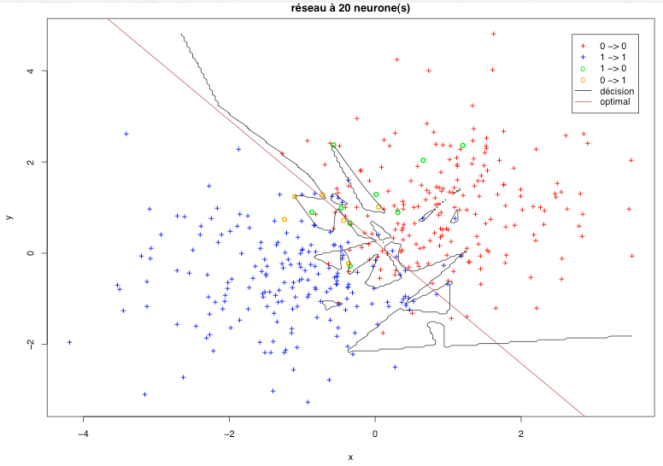


MLP 2-10-1

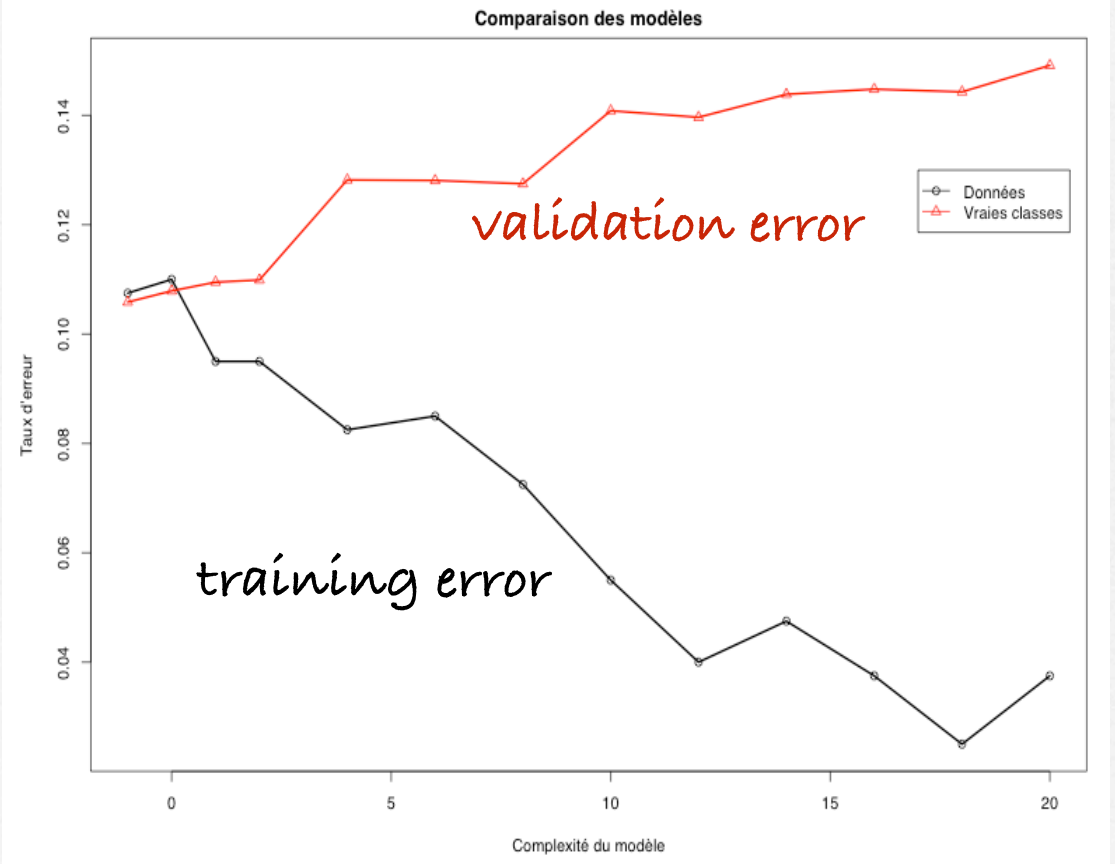
Effect of the number of hidden neurons (3)



MLP 2-16-1

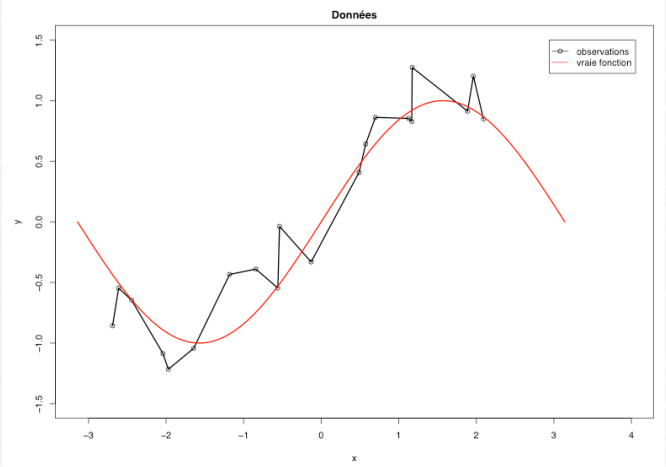


MLP 2-20-1

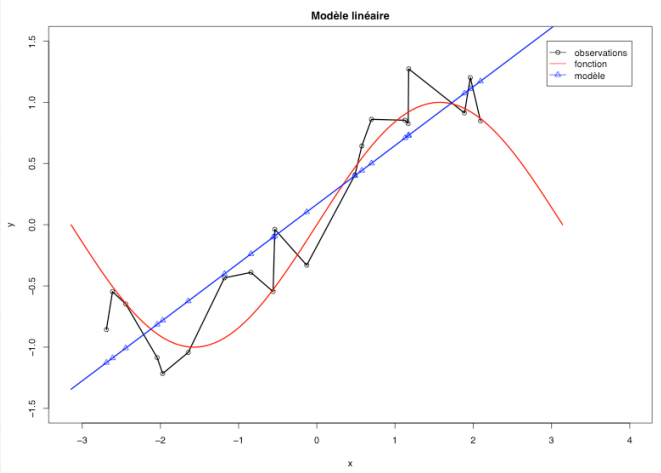


Effect of the number of hidden neurons (4)

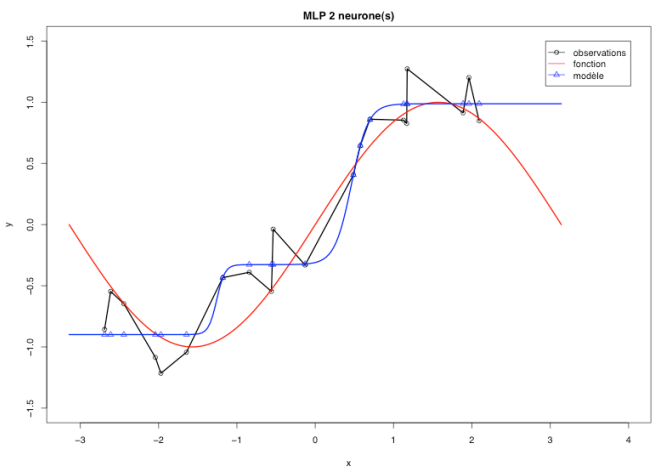
regression problem



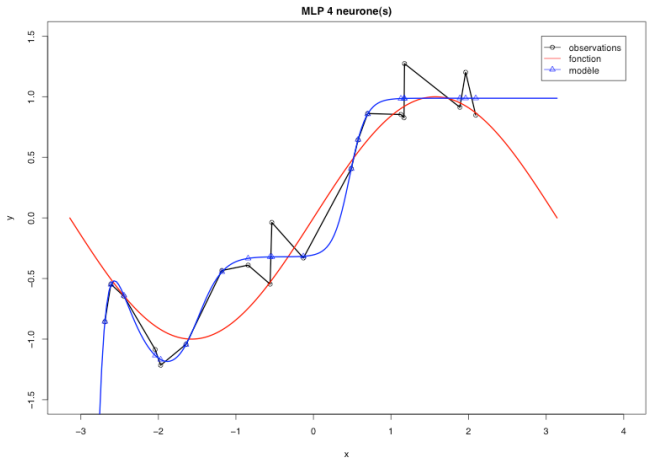
Data



Linear model

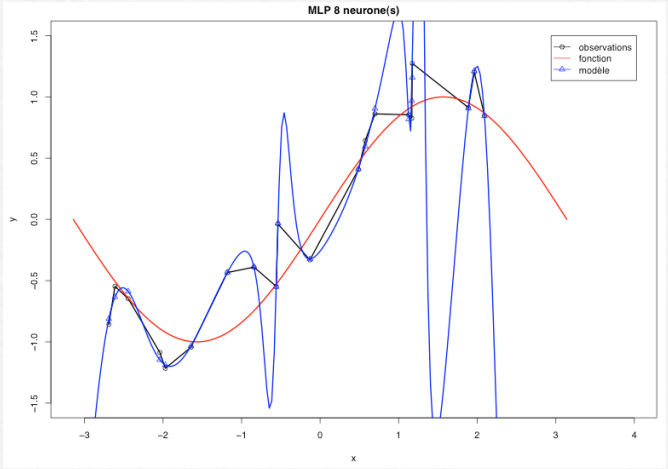


MLP 1-2-1

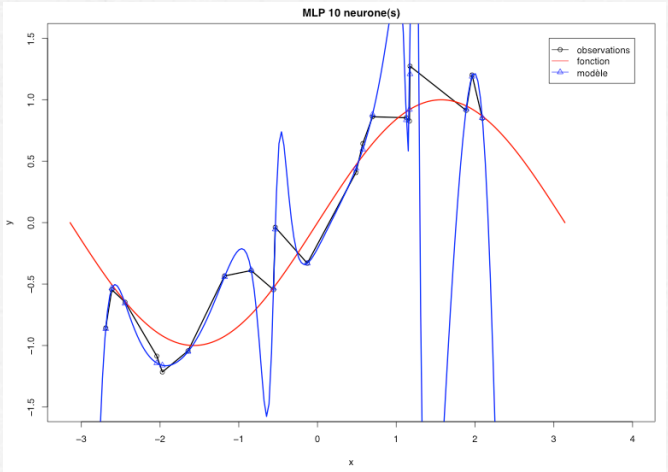


MLP 1-4-1

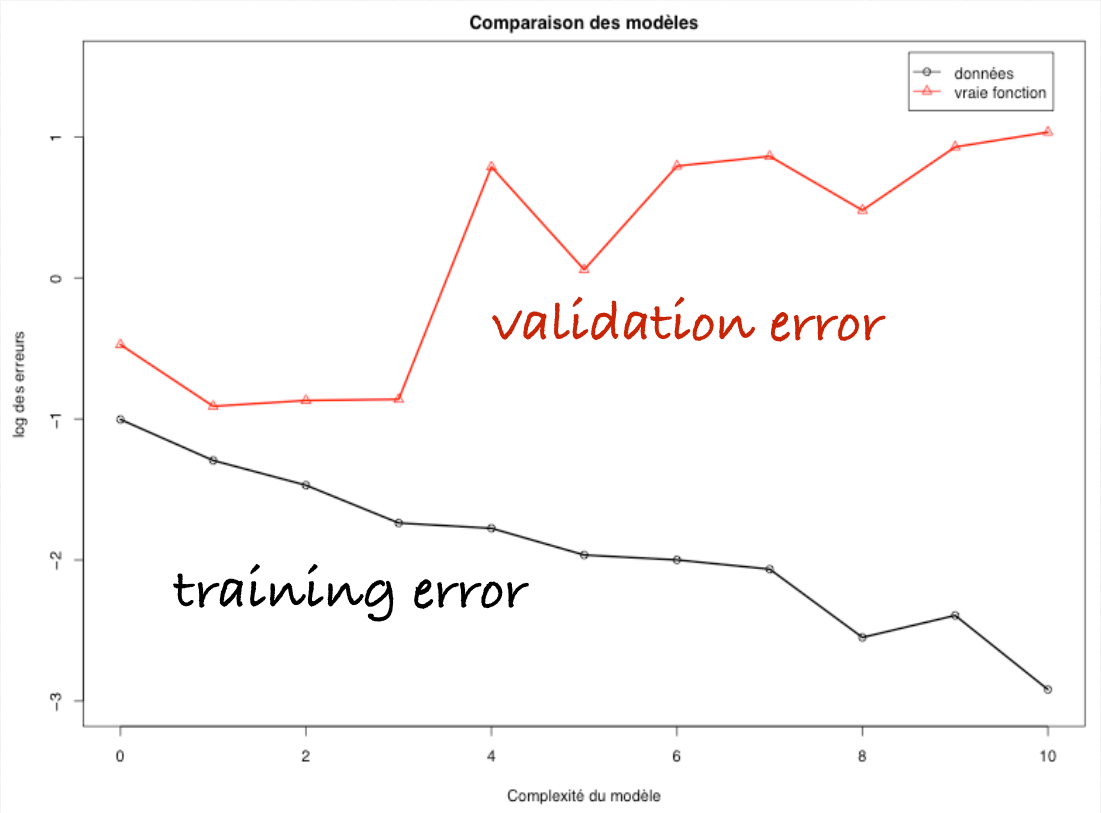
Effect of the number of hidden neurons (5)



MLP 1-8-1

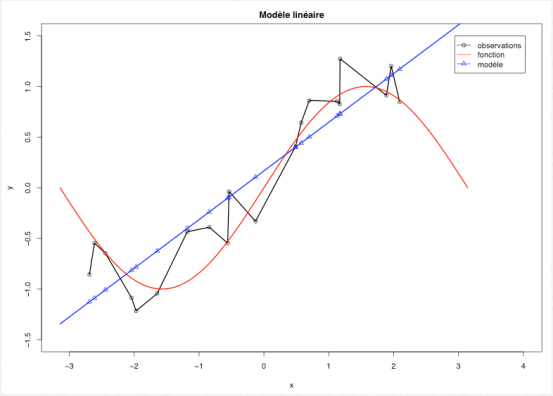


MLP 1-10-1

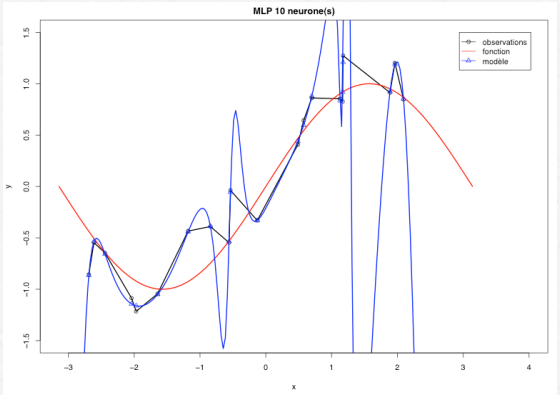


Bias-variance tradeoff

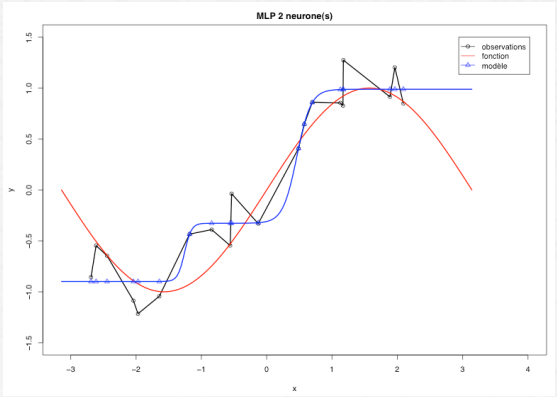
- bias: systematic error of the model
- variance: sensibility of the model



high bias, low variance



low bias, high variance

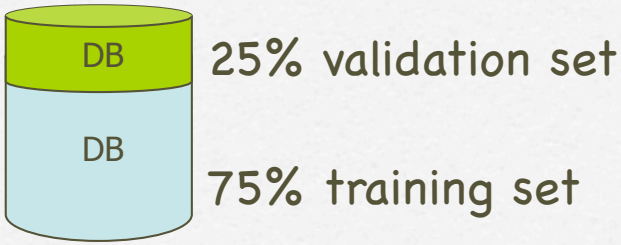


good bias-variance trade-off

Avoiding overfitting

- The more complex (flexible) the model is, the less bias it has (e.g., it can better learn the training data).
- However, the more complex the model, the higher variance (sensibility) it exhibits
- Some tricks to avoid overfitting:
  - Early stopping
  - Regularization
  - Data augmentation

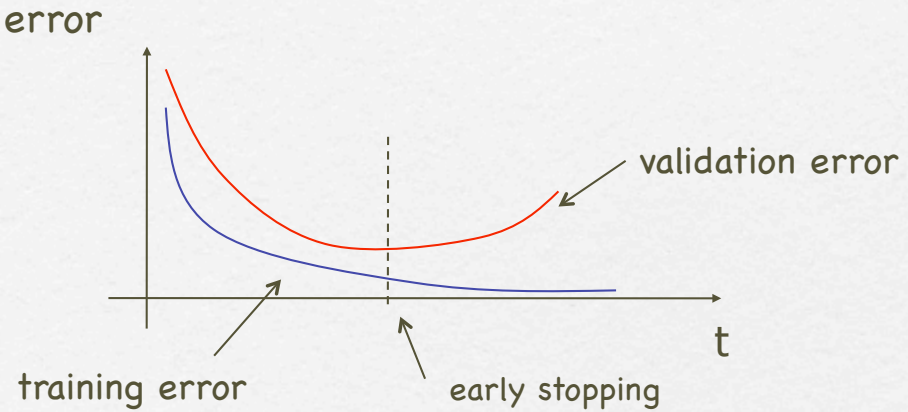
Early stopping



25% validation set

75% training set

"data splitting"



- We setup a large number of iterations, but monitor the evolution of learning and validation errors to stop the training whenever the validation error starts to increase too much (early stopping patience parameter)

Regularization

Regularization refers to providing constraints to limit the values of the parameters we are learning (e.g., the weights).

We modify the loss or error function that Backpropagation is trying to minimize by adding a penalty term.

Weight decay (or L2): avoid large weight values

$$pénalité = 1/2 \sum_{ijk} \left[w_{ij}^{(k)} \right]^2$$

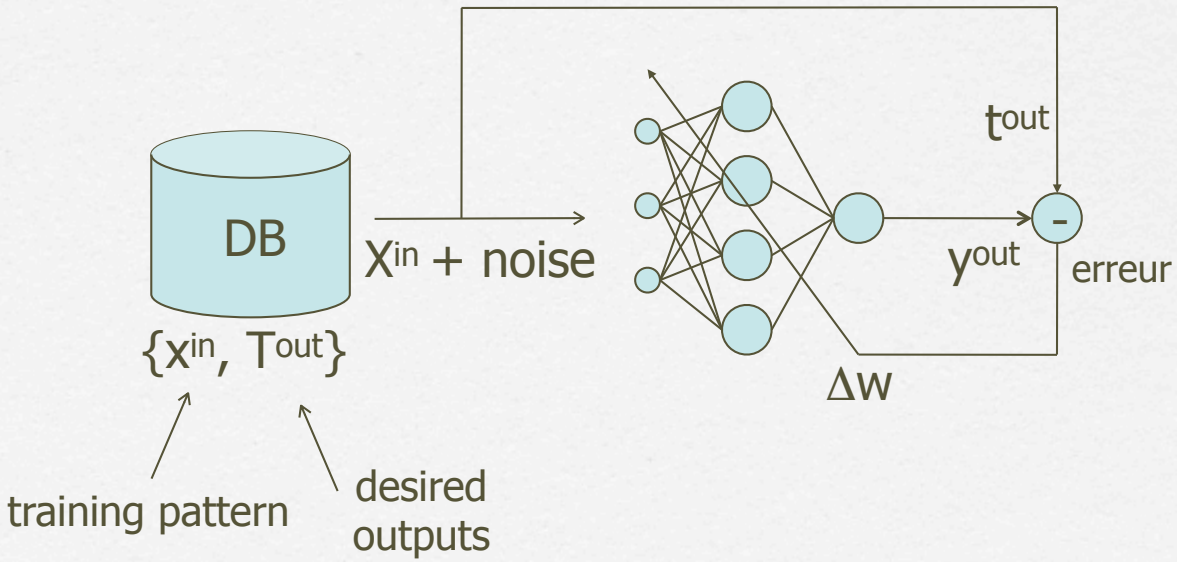
another hyper-parameter : (

Modified objective function: $E = E + \lambda pénalité$

$$\Delta w_{ij}^{(k)}(t) = \left\{ terme\ standard \right\} - \eta \lambda w_{ij}^{(k)}$$

Data augmentation

- A simple way to avoid overfitting consists on adding some noise to the input data to avoid the learning "by heart" of the available examples

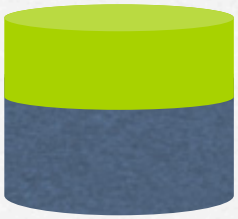


Model selection

hyper-parameter optimization

set of model "configs"
- number of layers
- neurons per layer
- learning rate
- momentum term
- number of epochs
- etc...

for each config :
perform cross-validation



e.g., 4-fold cross-validation

performance

selected model

final performance