

# Apprentissage Automatique

## Régression

S. Herbin, A. Chan Hon Tong

# Objectifs du cours

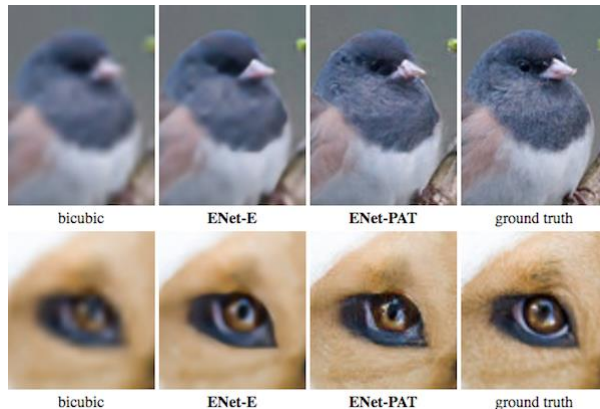
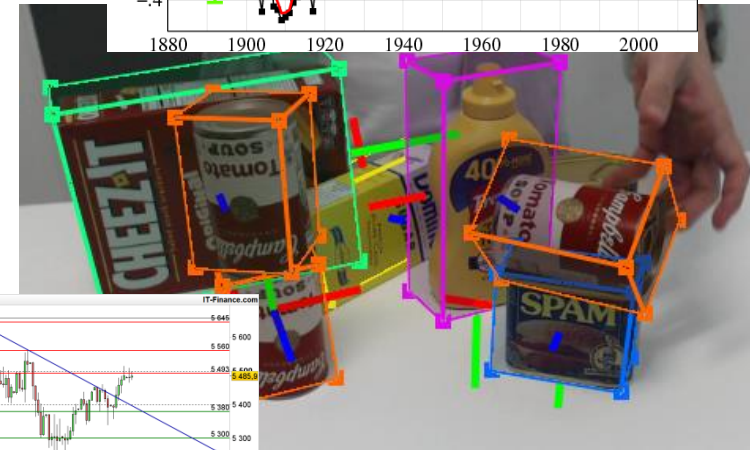
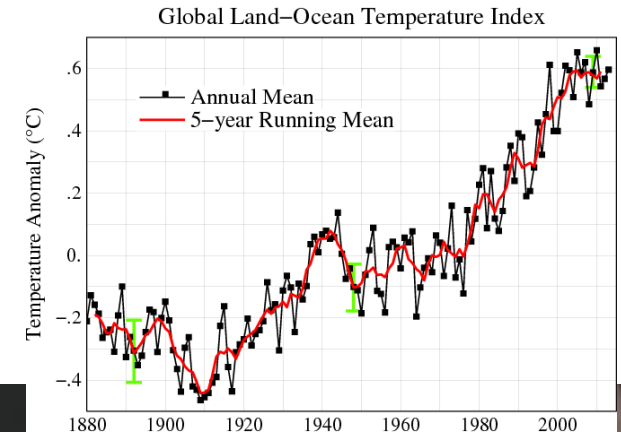
- Présentation des techniques classiques de régression
- Savoir les choisir et utiliser
- Régression et Deep Learning

# A quoi ça sert

- **Prédire** une valeur (interpolation, extrapolation)
- Expliquer/détecter/repérer des corrélations/tendances

- Exemples de prédiction:

- Prédiction de régime moteur
- Prédiction de durée de survie
- Estimation de prix
- Prévision météo ou climatique
- Prévision de cours de la bourse
- Super résolution
- Estimation de pose d'objet



# Formulation de la régression

- Prédicteur:  $W, x \mapsto y$
- Modèle :  $P(t | x, W)$  ou  $t = y(x, W) + \varepsilon$

Où

- entrée  $x \in \mathbb{R}^M$ , prédiction  $y \in \mathbb{R}^p$ , cible  $t \in \mathbb{R}^p$
  - $W$  paramètres du prédicteur estimé à partir d'un ensemble d'apprentissage:  $\mathcal{L} = \{x_i, t_i\}_{i=1..N}$
  - $\varepsilon$  variable aléatoire décrivant l'erreur de prédiction (souvent considérée gaussienne)
- 
- C'est du « Machine Learning » (Apprentissage supervisé)
  - C'est du « Data Mining » (Estimation de dépendances entre variables)

# Modèles linéaires

- La base de la régression
  - Fondements mathématiques solides
  - Calculs analytiques ou optimisation séquentielle
- Plusieurs manières de dépasser la linéarité
  - Modèles linéaires généralisés
  - Modèles à base de noyaux (« kernels »)
- Plusieurs manières de maîtriser la complexité
  - « Large margin » (cf. SVM)
  - Régularisateurs

# Modèles linéaires généralisés (cas scalaire $y \in \mathbb{R}$ )

biais

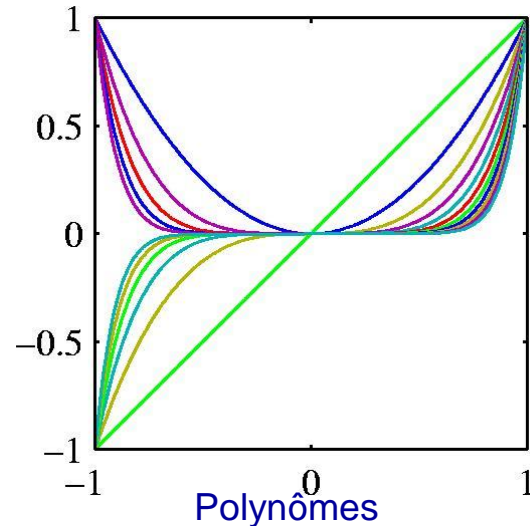
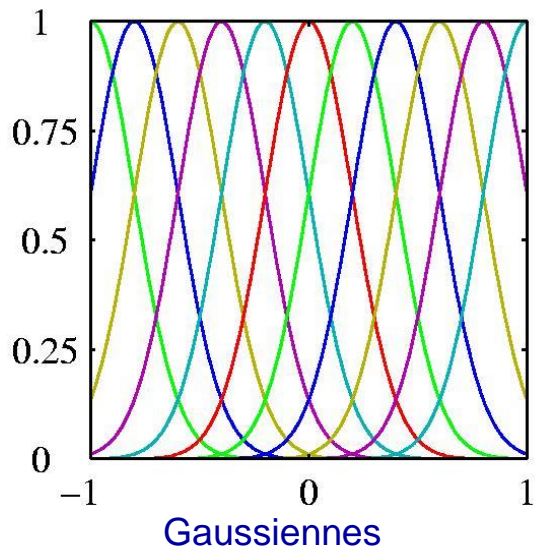
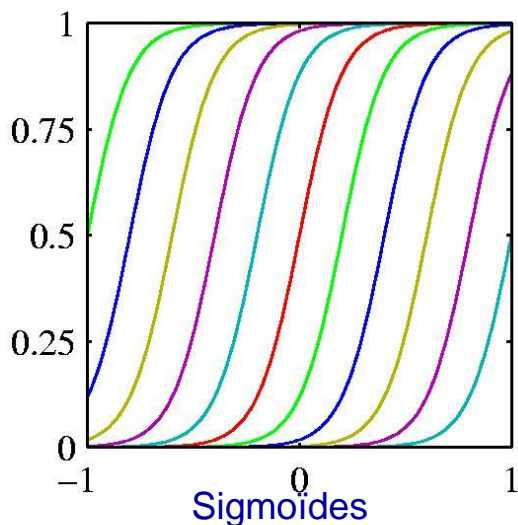


$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \dots = \mathbf{w}^T \mathbf{x}$$

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 \phi_1(\mathbf{x}) + w_2 \phi_2(\mathbf{x}) + \dots = \mathbf{w}^T \Phi(\mathbf{x})$$

- Principe simple: utiliser plusieurs **fonctions de base**  $\phi$  encodant les données source (« features »)
- Une fois définies les fonctions, le problème reste **linéaire!**
- Comment trouver ces fonctions?
  - Se les donner
  - Les apprendre

# Exemples classiques de fonctions de base 1D



Remarque: les sigmoïdes et gaussiennes sont des fonctions d'activation usuelles dans les réseaux de neurones

➔ les RN permettent d'apprendre les  $\phi$

# Apprentissage = trouver $W_{ML}$

- Forme du prédicteur

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

- Critère d'erreur (évaluation)

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

- Principe statistique: maximum de vraisemblance

$$W_{ML} = \underset{W}{\operatorname{argmax}} P(\mathbf{t} | \mathbf{x}, W)$$



# Maximum Likelihood and Least Squares (1)

---

Assume observations from a deterministic function with added Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \quad \text{where} \quad p(\epsilon|\beta) = \mathcal{N}(\epsilon|0, \beta^{-1})$$

which is the same as saying,

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

Given observed inputs,  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ , and targets,  $\mathbf{t} = [t_1, \dots, t_N]^T$ , we obtain the likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n|\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}).$$

# Maximum Likelihood and Least Squares (2)

---

Taking the logarithm, we get

$$\begin{aligned}\ln p(\mathbf{t}|\mathbf{w}, \beta) &= \sum_{n=1}^N \ln \mathcal{N}(t_n | \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) \\ &= \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(\mathbf{w})\end{aligned}$$

where

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

is the sum-of-squares error.

# Maximum Likelihood and Least Squares (3)

---

Computing the gradient and setting it to zero yields

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\} \phi(\mathbf{x}_n)^T = \mathbf{0}.$$

Solving for  $\mathbf{w}$ , we get

$$\mathbf{w}_{\text{ML}} = \left( \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}$$

The Moore-Penrose  
pseudo-inverse,  $\Phi^\dagger$ .

where

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

# Maximum Likelihood and Least Squares (4)

---

Maximizing with respect to the bias,  $w_0$ , alone, we see that

$$\begin{aligned} w_0 &= \bar{t} - \sum_{j=1}^{M-1} w_j \bar{\phi}_j \\ &= \underbrace{\frac{1}{N} \sum_{n=1}^N t_n}_{\text{mean of } t} - \sum_{j=1}^{M-1} w_j \underbrace{\frac{1}{N} \sum_{n=1}^N \phi_j(\mathbf{x}_n)}_{\text{mean of } \phi_j} \end{aligned}$$

We can also maximize with respect to  $\beta$ , giving

$$\frac{1}{\beta_{\text{ML}}} = \frac{1}{N} \sum_{n=1}^N \{t_n - \mathbf{w}_{\text{ML}}^T \phi(\mathbf{x}_n)\}^2$$

# Geometry of Least Squares

---

Consider

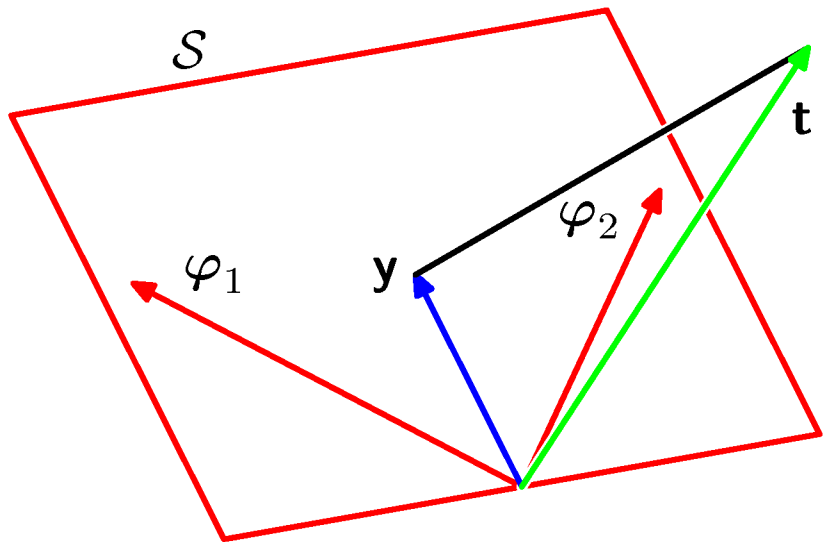
$$\mathbf{y} = \Phi \mathbf{w}_{\text{ML}} = [\varphi_1, \dots, \varphi_M] \mathbf{w}_{\text{ML}}.$$

$$\mathbf{y} \in \mathcal{S} \subseteq \mathcal{T} \quad \mathbf{t} \in \mathcal{T}$$

↑                    ↑  
M-dimensional    N-dimensional

$\mathcal{S}$  is spanned by  $\varphi_1, \dots, \varphi_M$ .

$\mathbf{w}_{\text{ML}}$  minimizes the distance between  $\mathbf{t}$  and its orthogonal projection on  $\mathcal{S}$ , i.e.  $\mathbf{y}$ .



## Sorties vectorielles ( $y \in \mathbb{R}^P$ )

On peut étendre le résultat précédent à:

$$\mathbf{W}_{\text{ML}} = \left( \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{T}.$$

où  $\mathbf{T} = [\mathbf{t}_1, \dots, \mathbf{t}_N]^T$  est la matrice des cibles ( $N \times p$ )

Pour une cible unique  $\mathbf{t}_k = [t_{1k}, \dots, t_{Nk}]^T$  on a

$$\mathbf{w}_k = \left( \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}_k = \Phi^\dagger \mathbf{t}_k$$

où  $\Phi^\dagger$  ne dépend que des données d'entrée.

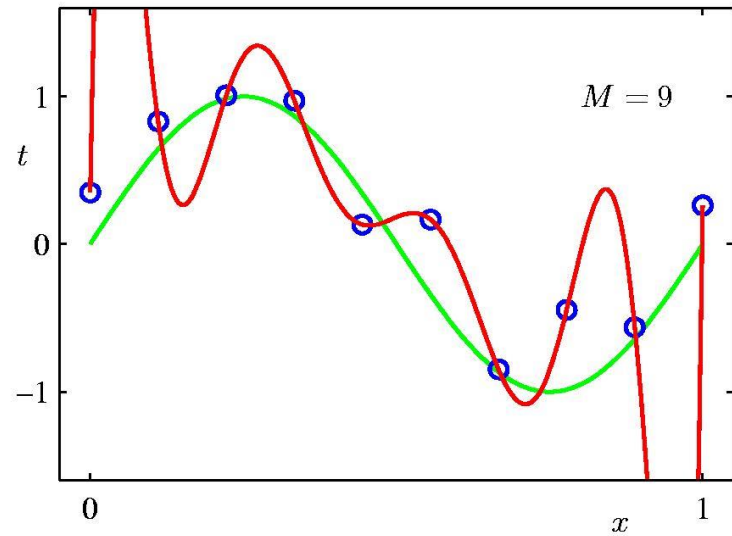
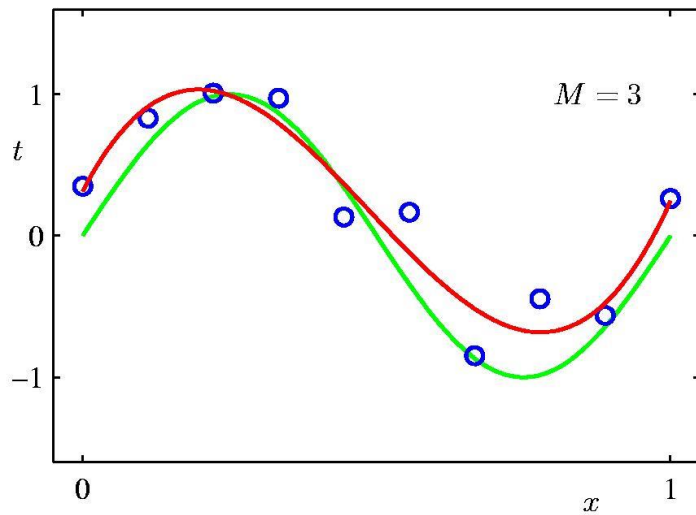
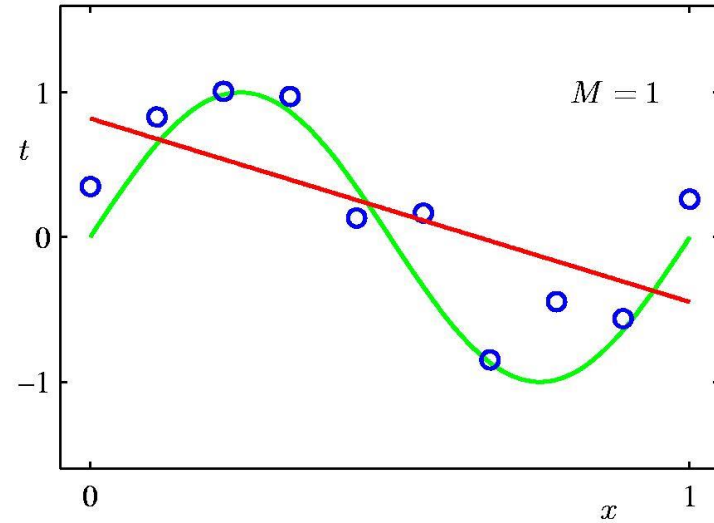
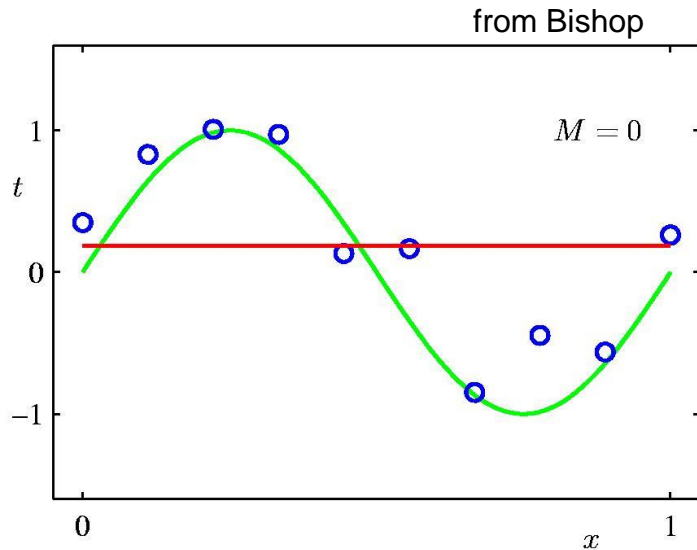
# Alternative: apprentissage séquentiel

- Dans le cas de problèmes à grandes dimensions d'entrée (M) ou d'échantillons nombreux (N), le calcul de  $\Phi^\dagger$  peut être coûteux.
- On peut alors utiliser une descente de gradient stochastique, échantillon par échantillon:

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta \nabla E_n \\ &= \mathbf{w}^{(\tau)} + \eta (t_n - \mathbf{w}^{(\tau)\top} \phi(\mathbf{x}_n)) \phi(\mathbf{x}_n).\end{aligned}$$

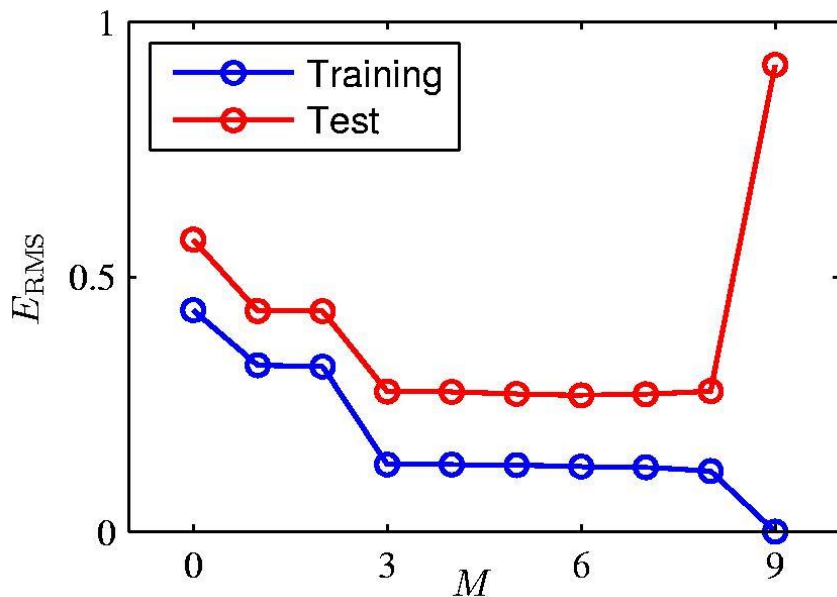
- On parle alors de *least-mean-squares (LMS) algorithm*.
- La question est alors de bien déterminer le pas du gradient...

# Retour sur régularisation





# Sur-apprentissage



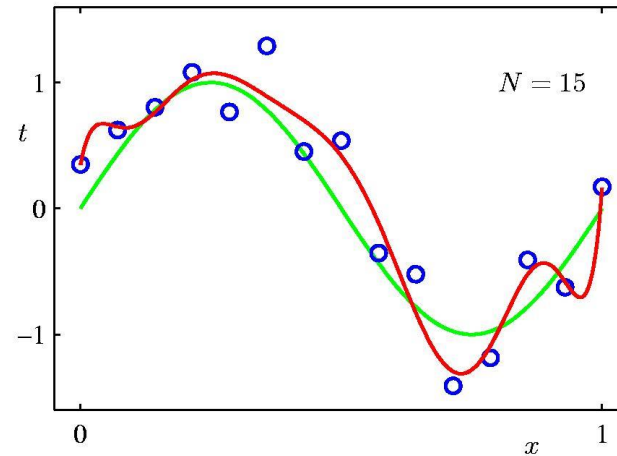
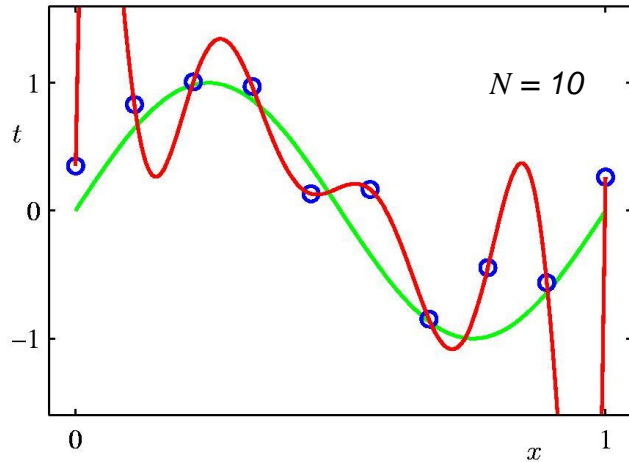
Comparaison des erreurs de test et d'apprentissage

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^*$	0.19	0.82	0.31	0.35
$w_1^*$		-1.27	7.99	232.37
$w_2^*$			-25.43	-5321.83
$w_3^*$			17.37	48568.31
$w_4^*$				-231639.30
$w_5^*$				640042.26
$w_6^*$				-1061800.52
$w_7^*$				1042400.18
$w_8^*$				-557682.99
$w_9^*$				125201.43

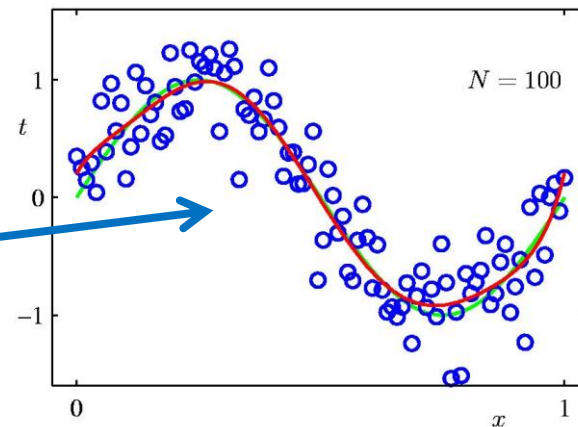
Coefficients des polynômes

# Influence de la quantité de données

## Polynôme d'ordre 9



C'est aussi un moyen de  
contrôler la régression



# Moindre carrés régularisés

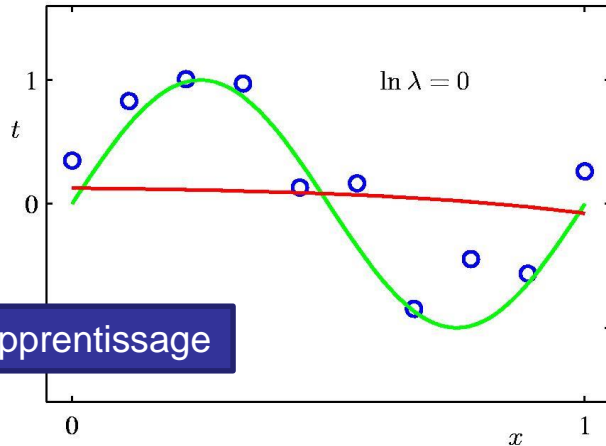
- Rappel: on peut rajouter une pénalisation à la fonction de coût:

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

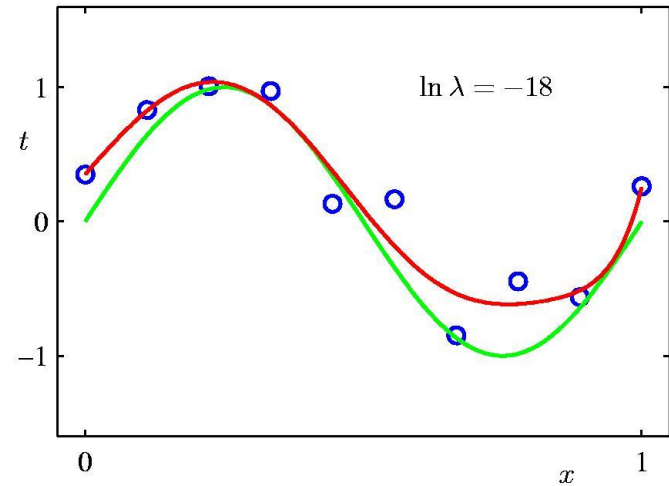
Dont l'optimum exact est alors:

$$\mathbf{w} = \left( \lambda \mathbf{I} + \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}.$$

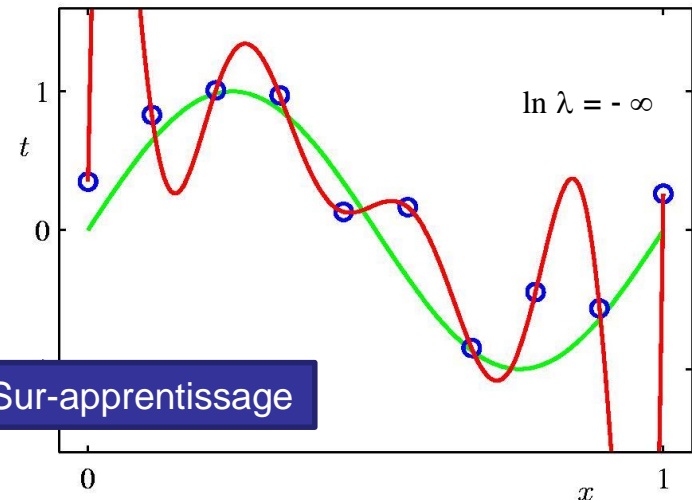
# Effet de la régularisation



Sous-apprentissage

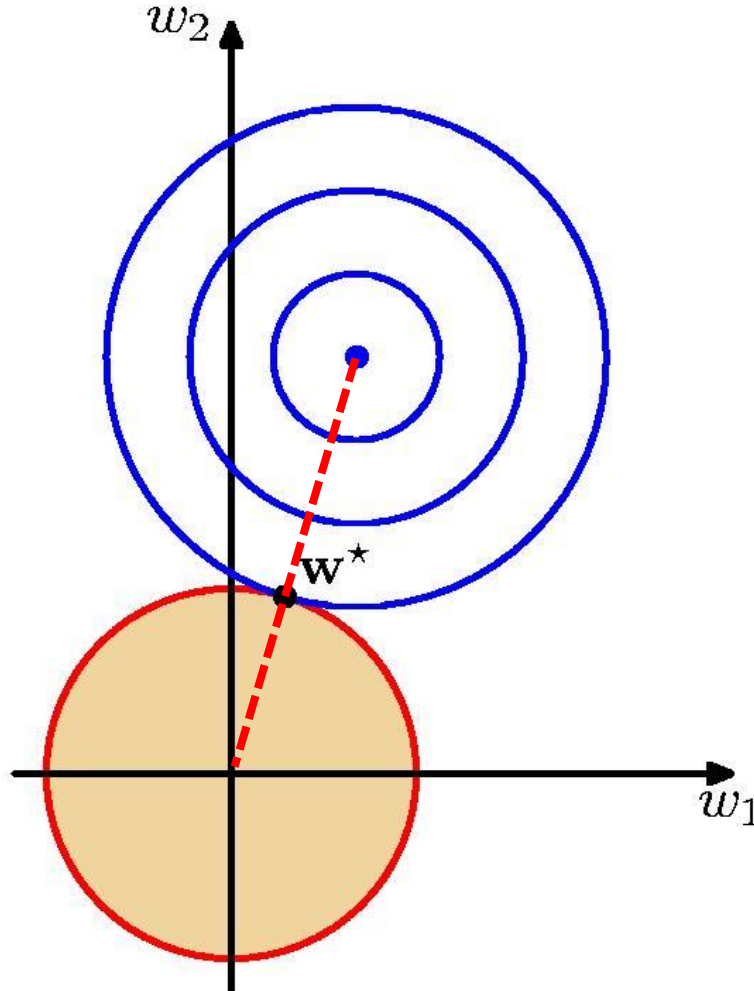


	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^*$	0.35	0.35	0.13
$w_1^*$	232.37	4.74	-0.05
$w_2^*$	-5321.83	-0.77	-0.06
$w_3^*$	48568.31	-31.97	-0.05
$w_4^*$	-231639.30	-3.89	-0.03
$w_5^*$	640042.26	55.28	-0.02
$w_6^*$	-1061800.52	41.32	-0.01
$w_7^*$	1042400.18	-45.95	-0.00
$w_8^*$	-557682.99	-91.53	0.00
$w_9^*$	125201.43	72.68	0.01



Sur-apprentissage

# Comment fonctionne la régularisation L2

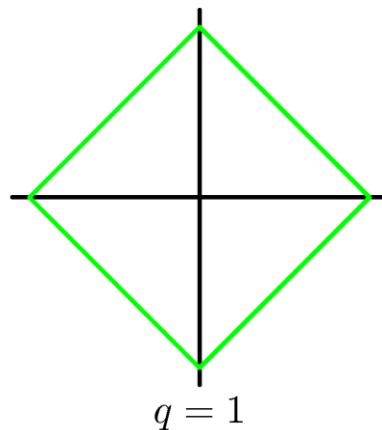
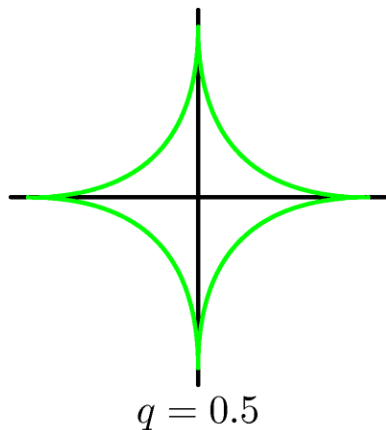


- Le coût global est la somme de deux « cuvettes ».
- La somme est aussi quadratique.
- Le minimum global est sur une ligne joignant l'origine et le minimum sans contrainte.
- La régularisation a pour effet de diminuer les poids.

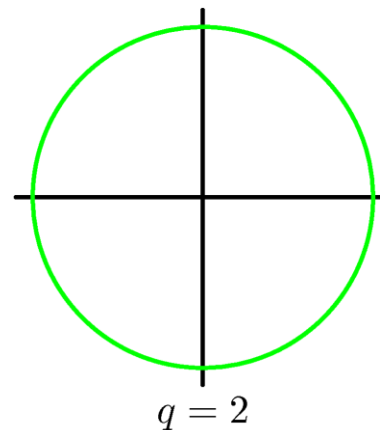
# Autres types de régularisation

Une manière simple: utiliser une autre norme

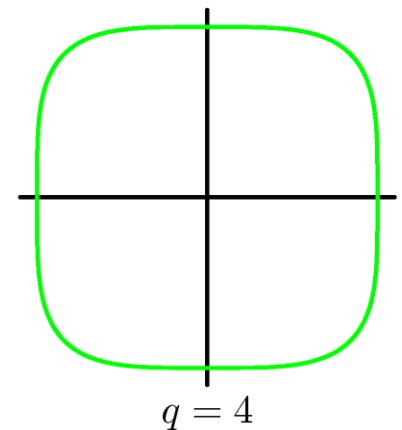
$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$



Lasso

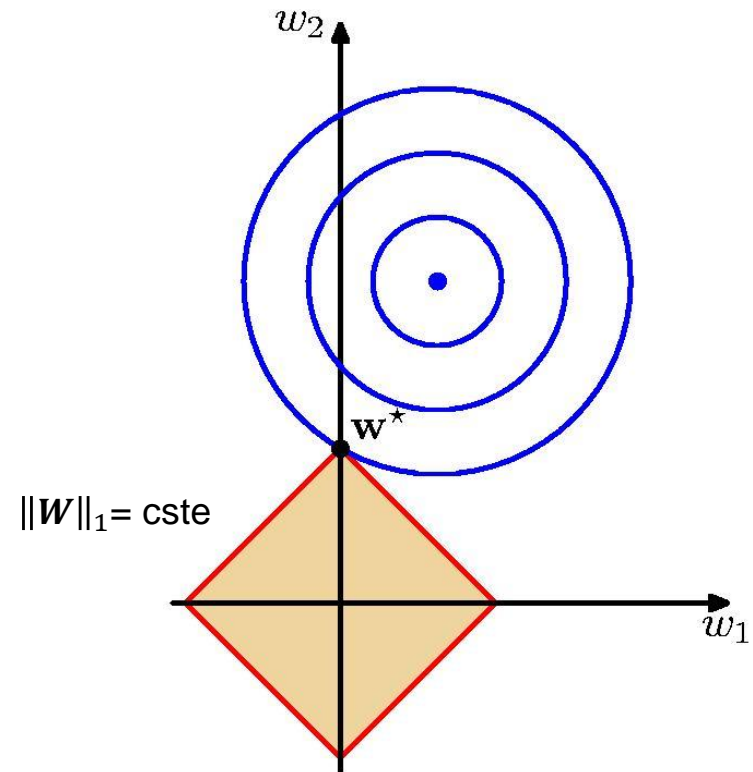
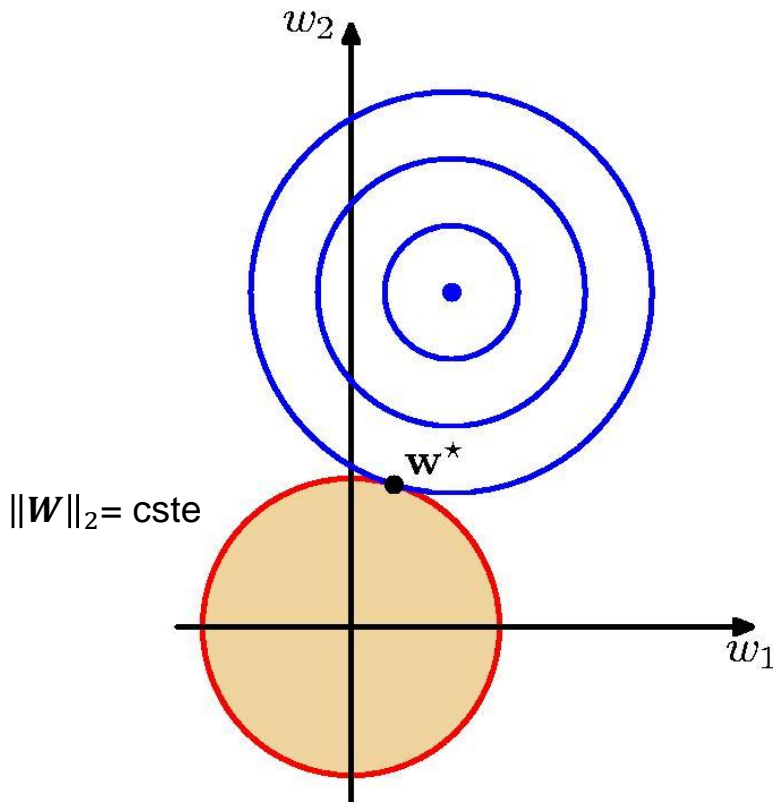


Quadratic



# Interprétation géométrique

Lasso



Remarque:  $w_1=0$  à l'optimum. **Sparsité** de la solution.

# D'autres pénalisations...

- **The Tikhonov regularization:**  $\psi(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2$ .
- **The  $\ell_1$ -norm:**  $\psi(\mathbf{w}) = \|\mathbf{w}\|_1$ .
- **The Elastic-Net:**  $\psi(\mathbf{w}) = \|\mathbf{w}\|_1 + \gamma \|\mathbf{w}\|_2^2$ .
- **The Fused-Lasso:**  $\psi(\mathbf{w}) = \|\mathbf{w}\|_1 + \gamma \|\mathbf{w}\|_2^2 + \gamma_2 \sum_{i=1}^{p-1} |\mathbf{w}_{i+1} - \mathbf{w}_i|$ .
- **The group Lasso:**  $\psi(\mathbf{w}) = \sum_{g \in G} \eta_g \|\mathbf{w}_g\|_2$ , where  $G$  are groups of variables.
- **The group Lasso with  $\ell_\infty$ -norm:**  $\psi(\mathbf{w}) = \sum_{g \in G} \eta_g \|\mathbf{w}_g\|_\infty$ , where  $G$  are groups of variables.
- **The sparse group Lasso:** same as above but with an additional  $\ell_1$  term.
- **The tree-structured sum of  $\ell_2$ -norms:**  $\psi(\mathbf{w}) = \sum_{g \in G} \eta_g \|\mathbf{w}_g\|_2$ , where  $G$  is a tree-structured set of groups [15], and the  $\eta_g$  are positive weights.
- **The tree-structured sum of  $\ell_\infty$ -norms:**  $\psi(\mathbf{w}) = \sum_{g \in G} \eta_g \|\mathbf{w}_g\|_\infty$ . See [15]
- **General sum of  $\ell_\infty$ -norms:**  $\psi(\mathbf{w}) = \sum_{g \in G} \eta_g \|\mathbf{w}_g\|_\infty$ , where no assumption are made on the groups  $G$ .
- **The path-coding penalties of [24].**
- **the  $\ell_1$ -constraint.**

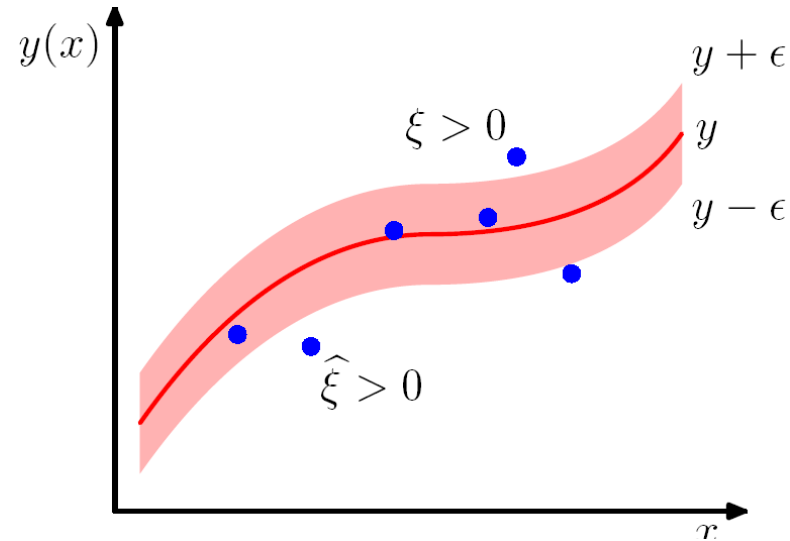
<http://spams-devel.gforge.inria.fr/documentation.html>



# Support Vector Regression

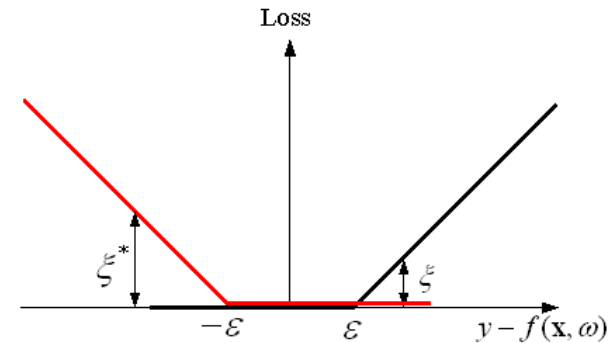
$$\text{Min } \frac{1}{2} ||w||^2 + C \sum_{i=1}^n (\xi_i + \xi_i^*)$$

$$\text{subject to } \begin{cases} u_i - \mathbf{w}^T \mathbf{x}_i - b \leq \epsilon + \xi_i \\ \mathbf{w}^T \mathbf{x}_i + b - u_i \leq \epsilon + \xi_i^* \\ \xi_i \geq 0, \xi_i^* \geq 0 \end{cases}$$



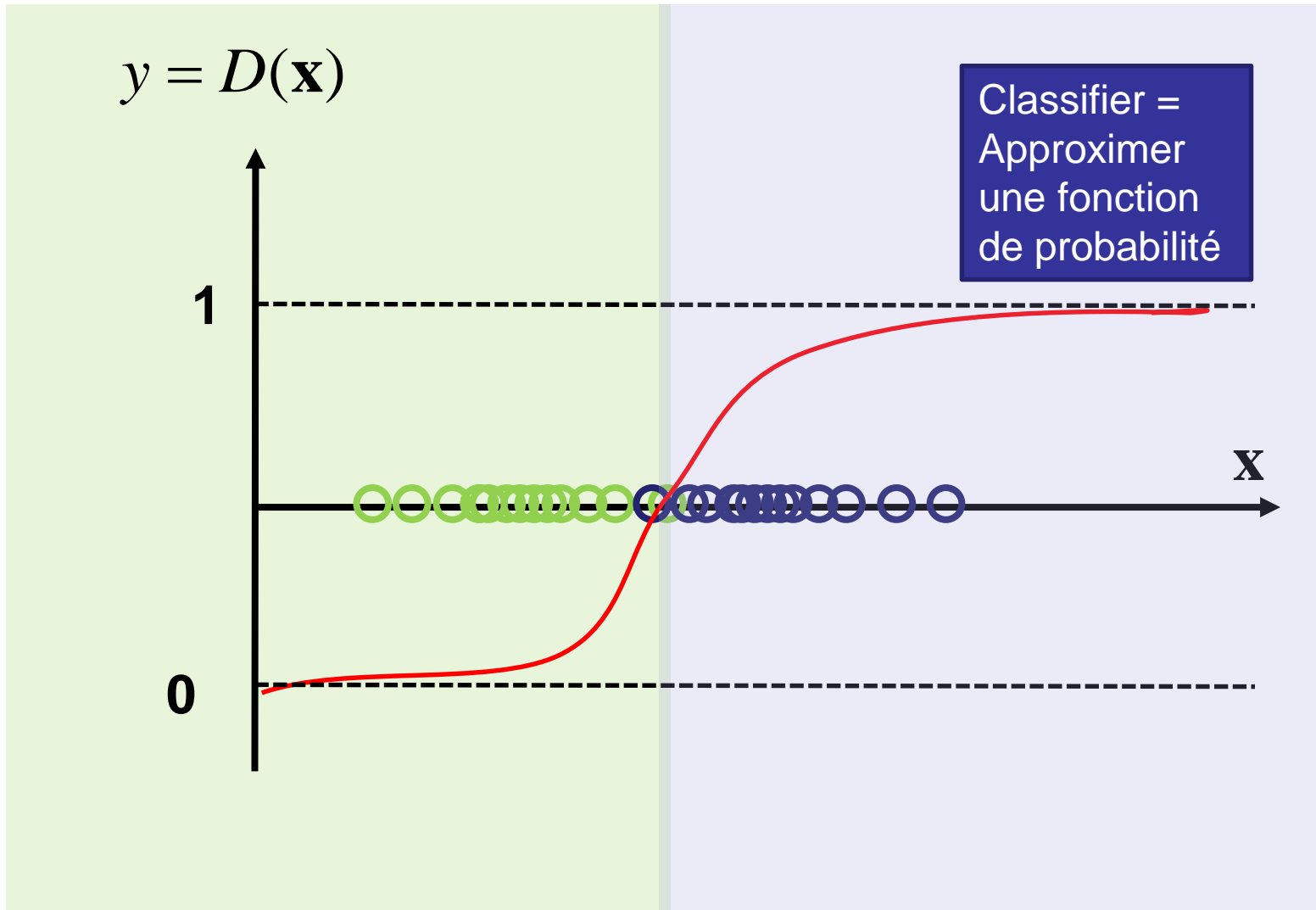
Formulation comparable à celle de la classification

- Fonction de coût différente (dépend d'un paramètre)
- Expression du « soft margin » symétrique
- « Kernel trick » applicable
- Sparsité de la solution



$$L_{\epsilon}(y, f(\mathbf{x}, \omega)) = \max(|y - f(\mathbf{x}, \omega)| - \epsilon, 0)$$

# Classification et Régression: : même combat



The graph shows the error function  $\text{erf}(x)$  plotted against  $x$ . The x-axis ranges from -3 to 3 with major ticks every 1 unit. The y-axis ranges from -1.00 to 1.00 with major ticks every 0.25 units. A red curve represents the function, which is an odd function passing through the origin (0,0). It approaches -1 as  $x \rightarrow -\infty$  and approaches 1 as  $x \rightarrow \infty$ .

- $$p(C_1 | \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0) \quad \text{où} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- $$\begin{aligned}
 E(\mathbf{w}) &= -\sum_{n=1}^N \ln p(t_n | y_n) \\
 &= -\sum_{n=1}^N t_n \ln y_n + (1-t_n) \ln (1-y_n)
 \end{aligned}$$

si  $t = 0$

# Optimisation de la régression logistique

- La minimisation du critère n'admet pas formulation analytique

➔ descente de gradient

- Le gradient se calcule facilement

$$\frac{\partial E_n}{\partial \mathbf{w}} = (y_n - t_n) \mathbf{x}_n$$

- On peut rajouter comme pour la régression des pénalités de régularisation

# Problème liés à la régularisation globale

- La pénalisation est isotrope: toutes les dimensions sont considérées simultanément, avec le même poids
  - on fait l'hypothèse que les dimensions sont comparables (en unité et signification)
  - Besoin de normaliser ou « blanchir » les données. Mais risque alors de louper les fortes corrélations entre données.
- Beaucoup de types de régularisation (et d'algorithmes d'optimisation...): comment choisir?
  - Validation croisée
  - Structure du problème (on veut forcer certaines dimensions ou certaines corrélations entre dimensions)
  - On veut obtenir une solution interprétable → sparsité. Mais il y a d'autres approches algorithmiques pour la rechercher directement .
- Remarque: la recherche sur les algorithmes « sparse » était très active avant le « deep learning era ». Maintenant moins...

# Evaluer une régression

- Deux objectifs: prédiction ou modélisation
- Prédiction: on cherche la fonction ayant l'erreur de généralisation la plus faible
  - ➔ Erreur estimée sur base de test
  - ➔ Recherche des paramètres sur base de validation (validation croisée)
- Modélisation: on cherche la fonction qui explique au mieux les corrélations entre données
  - ➔ Tests statistiques ( $R^2$  et p-values)

# Formulation bayésienne

- Principes
  - Considérer la distribution jointe des sorties  $t$  conditionnellement aux entrées  $x$  comme gaussiennes
  - considérer les poids  $W$  comme des variables aléatoires  
→ on évolue dans des espaces de distributions
- Loi a priori  $P(W)$
- Vraisemblance des entrées  $P(t | x, W)$
- Loi a posteriori  $P(W | t)$
- Plus hyper paramètres de modélisation des lois (en général prises gaussiennes)

$$p(\mathbf{t} | \mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n | \mathbf{w}^T \mathbf{x}_n, \beta^{-1})$$

Gaussienne
variance du bruit de sortie

← vraisemblance

$$p(\mathbf{w} | \alpha) = \mathcal{N}(\mathbf{w} | 0, \alpha^{-1} \mathbf{I})$$

← prior

$$-\ln p(\mathbf{w} | \mathbf{t}) = \frac{\beta}{2} \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} + \text{const}$$

Inverse de la variance du prior

On retrouve une formulation  
avec régularisation

$$\lambda = \frac{\alpha}{\beta}$$



# Exemple

- Modèle à deux paramètres:

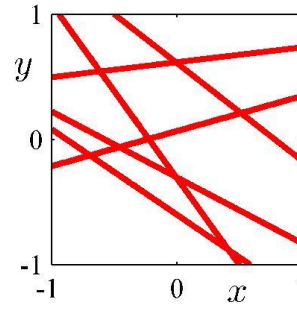
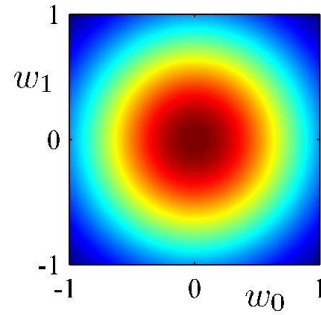
$$y(x, \mathbf{w}) = w_0 + w_1 x$$

- On peut imaginer la loi a posteriori sur les poids
- La vraisemblance est gaussienne  
→ loi a posteriori aussi gaussienne si la loi a priori l'est

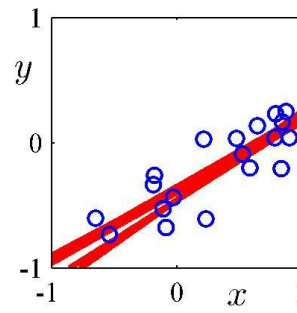
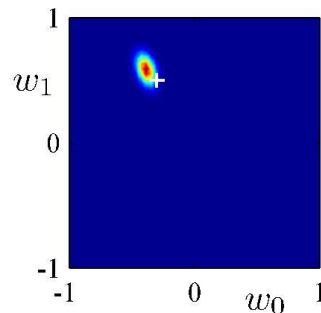
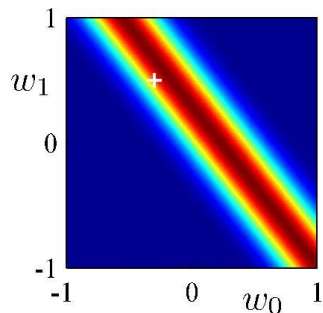
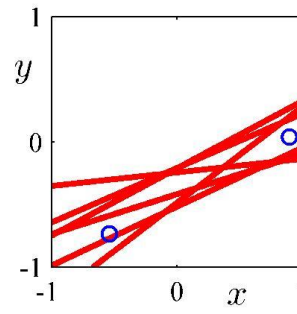
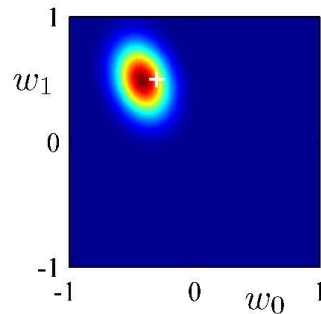
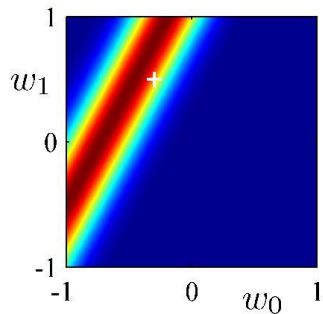
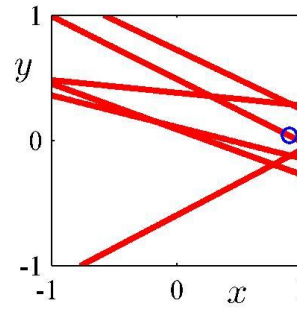
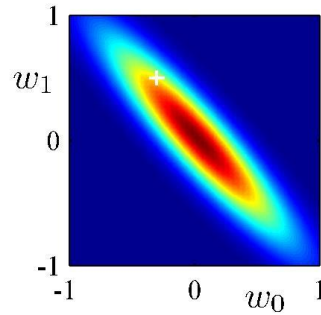
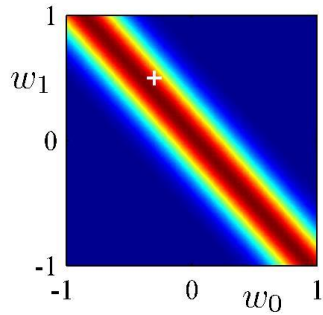
likelihood

prior/posterior

data space



- Sans données, on échantillonne la loi a priori

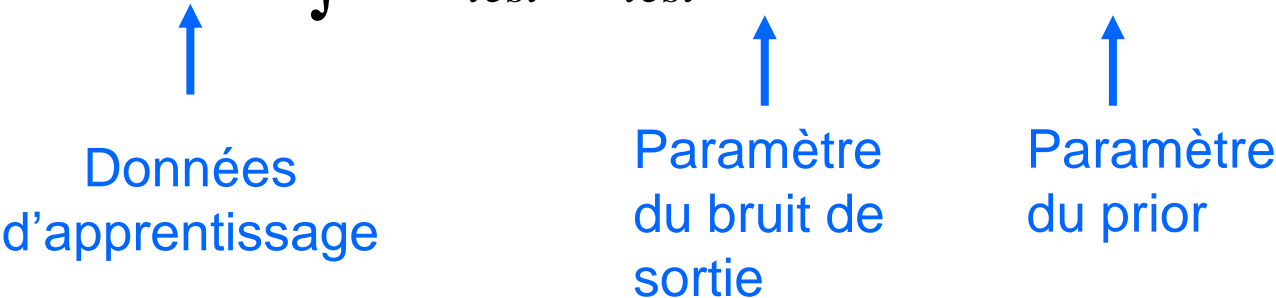


- La loi a priori a peu d'impact si 20 points

# Utilisation de la loi a posteriori

- Elle permet de faire des prédictions, des tirages aléatoires, des estimations d'intervalle de confiance...
- Dans le cas gaussien, on peut calculer explicitement en intégrant sur les paramètres

$$p(t_{test} | x_{test}, \alpha, \beta, D) = \int p(t_{test} | x_{test}, \beta, \mathbf{w}) p(\mathbf{w} | \alpha, \beta, D) d\mathbf{w}$$



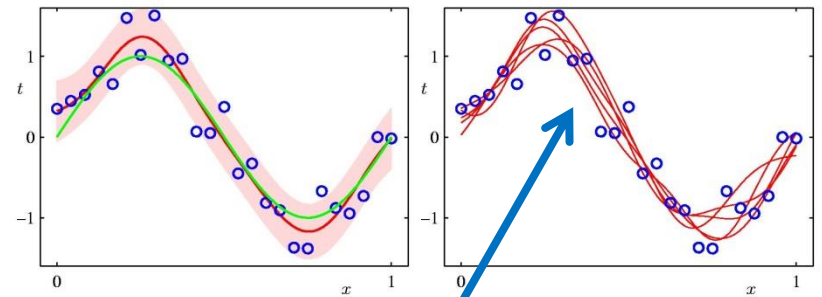
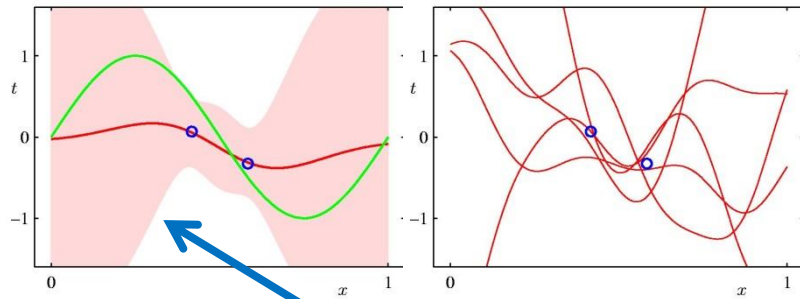
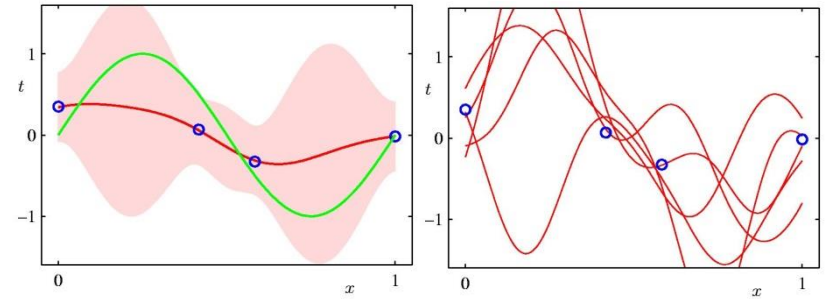
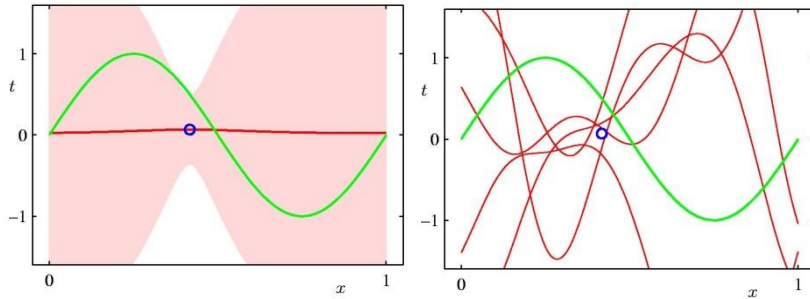
Données  
d'apprentissage

Paramètre  
du bruit de  
sortie

Paramètre  
du prior

- On peut aussi introduire directement les corrélations dans les modèles et utiliser le « kernel trick »
- ➔ Processus Gaussiens (« kriging »)

# Exemple de prédiction à partir d'une base de fonctions sinusoidales



Intervalle de confiance  
à un écart-type +  
moyenne

Echantillonnage de  
la loi a posteriori

# Régression et Deep Learning

- Les RN sont des fonctions paramétriques
- On dispose d'un algorithme « générique » d'optimisation: gradient stochastique (et variantes)
- Fonction de coût: erreur quadratique, cosinus...
- Difficulté: comment introduire la régularisation?
  - « Weight decay » (pénalisation L2)
  - « Drop-out »
  - « Early stopping »
  - Ajout de bruit
  - Multi-tâche
  - Lasso ?
  - Discussion générale ici:  
<https://www.deeplearningbook.org/contents/regularization.html>

The diagram illustrates the proposed 3D pose estimation framework, which is divided into three main stages: Feature Extraction, Embedding, and Classification / Regression.

**Feature Extraction:** An RGB image is processed through a series of convolutional layers (64, 128, 256, 512, 512) and max pooling layers to extract features.

**Embedding:** The extracted features are processed through three parallel branches:

- Classification / Regression:** The features are processed through a series of convolutional layers (64, 64, 64) to produce 64 #classes. This branch is used for classification and regression tasks.
- Center direction X, Y, and distance:** The features are processed through a series of convolutional layers (128, 128, 128) to produce 128 3 x #classes. This branch is used for estimating the center direction and distance.
- Pose Estimation:** The features are processed through a series of convolutional layers (512, 512, 512) to produce 512 512. This branch is used for estimating the 6D poses.

**Classification / Regression:** The output of the classification branch is used to produce the final 6D poses.

**Legend:**

- Convolution + ReLU (Blue block)
- Max Pooling (Green block)
- Deconvolution (Purple block)
- Addition (Grey block)
- Hough Voting (Red block)
- RoI Pooling (Yellow block)
- Fully Connected (Blue circles)

Automatique -- Régression-- 38

# « Single image super-resolution » en deep learning

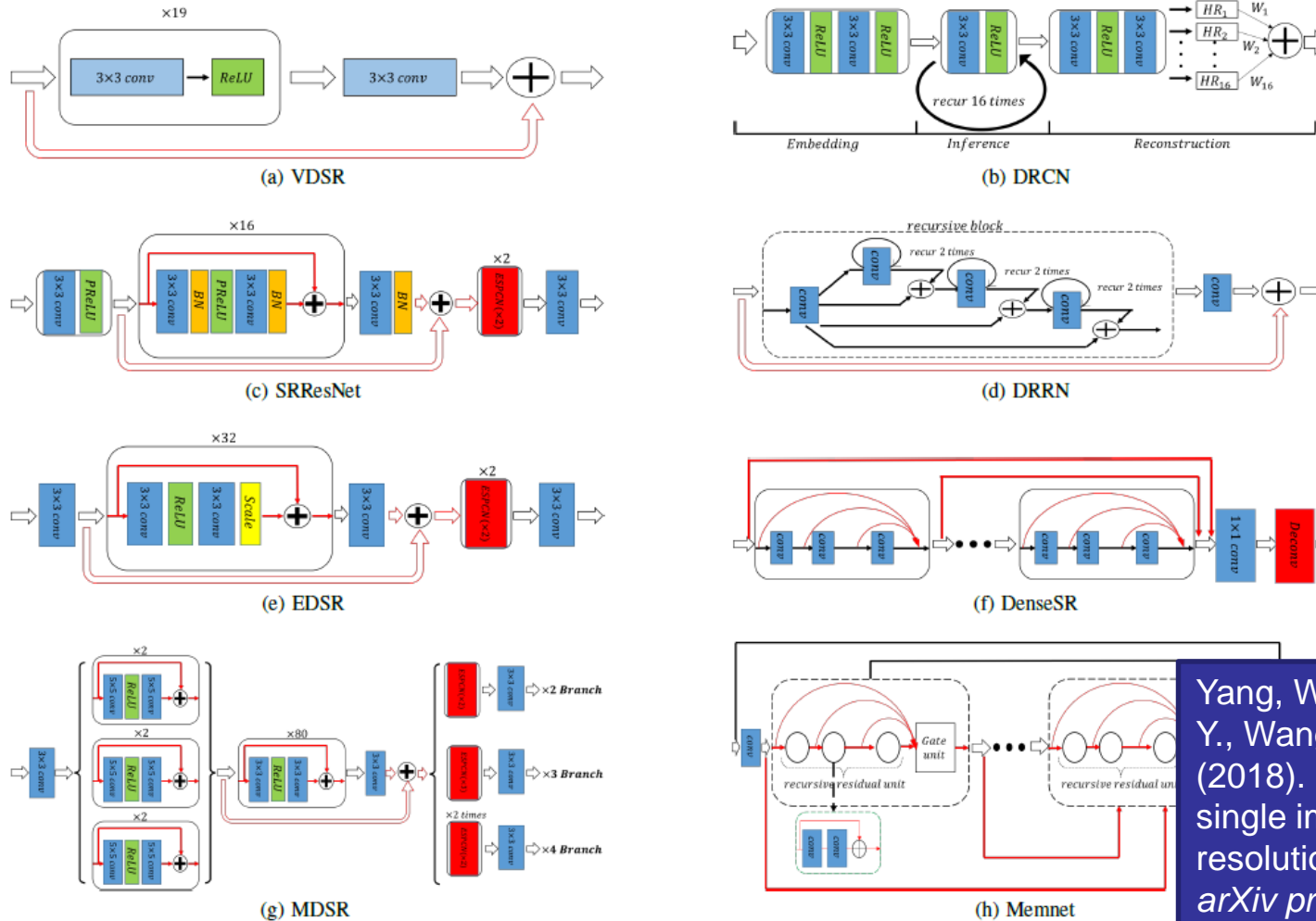


Figure 5: Sketch of several deep architectures for SISR.

Yang, W., Zhang, X., Tian, Y., Wang, W., & Xue, J. H. (2018). Deep learning for single image super-resolution: A brief review. *arXiv preprint arXiv:1808.03344*.

# Régression: les questions à se poser

- Quels sont les modèles & algorithmes?
  - Modèles linéaires généralisés
  - Processus gaussiens (krigeage)
  - Ensembles de prédicteurs (random forest, boosting, bagging...)
  - Réseaux de neurones
- Comment valider les modèles?
  - Validation croisée
  - Tests statistiques
- Comment maîtriser les grandes dimensions?
  - Projection / Construction de caractéristiques (« feature construction »)
  - Sélection de caractéristiques (« feature selection »)
  - Sparsité
- Comment contrôler les données aberrantes (« outliers »)?
  - Régularisation
  - Estimateurs robustes
  - RANSAC



# Implémentations logicielles

## SPAMS

- Bibliothèque orientée sparsité
- <http://spams-devel.gforge.inria.fr/documentation.html>

## Scikit-learn

- La plupart des modèles classiques implémentés
- Réseaux de neurones (mais pas profonds)
- [https://scikit-learn.org/stable/supervised\\_learning.html#supervised-learning](https://scikit-learn.org/stable/supervised_learning.html#supervised-learning)

## Pytorch

- Quelques exemples sur le site (<https://github.com/pytorch/examples>)
  - Régression polynomiale, Super-resolution
- Quelques tutoriels sur le web

## Tensorflow

- Quelques modules élémentaires
  - LinearRegressor, DNNRegressor
- Un tutoriel [https://www.tensorflow.org/tutorials/keras/basic\\_regression](https://www.tensorflow.org/tutorials/keras/basic_regression)

## Kaggle

- Plein d'exemples dans les « kernels »

# Références et sources

- Présentations et livre de C. Bishop (<https://www.microsoft.com/en-us/research/people/cmbishop/#!prml-book>)
- Cours de G. Hinton (<http://www.cs.toronto.edu/~hinton/csc2515/lectures.html>)
- Autres livres (en ligne):
  - Elements of Statistical Learning  
<https://web.stanford.edu/~hastie/Papers/ESLII.pdf>
  - An Introduction to Statistical Learning  
<http://www-bcf.usc.edu/~gareth/ISL/>
  - Gaussian processes  
<http://www.gaussianprocess.org/gpml/>
  - Machine Learning: a Probabilistic Perspective  
<https://www.cs.ubc.ca/~murphyk/MLbook/index.html>