esiea
INGÉNIEUR·E·S D'UN NUMÉRIQUE UTILE

Machine Learning

PART 3

Bonnes pratiques d'apprentissage

1) Il est **indispensable de pré-traiter** les données (entrées, sorties désirées)

2) L'algorithme de rétro-propagation est séquentiel mais il peut être modifié en calculant le **gradient total** (somme des gradients de tous les exemples)

3) Le **pas d'apprentissage λ peut être facilement adapté** par l'algorithme lui-même

4) Il faut mesurer **l'impact du nombre de poids sur les performances**

Pré-traitements : amplitude

Entrées :

1) normaliser (centrer et réduire)

→ $x_i' = (x_i - \mu_i)/\sigma_i$

2) décorréler (analyse en composantes principales)

Sorties désirées :

Éviter de saturer les sorties (-1,+1)

→ choisir les sorties désirées dans la zone linéaire de la sigmoïde

Pré-traitements : équilibrage

Exemple : 2 classes (not spam/spam),1000 observations

→ taux de reconnaissance = 95% → **OK ??**

Décision / Label	Not spam	Spam
Not spam	940	10
Spam	45	5

→ indicateurs : Précision et Rappel

Cas très fréquent (fraudes, pathologies …)

Que faire ?

1) essayer de trouver d'autres données

2) sous échantillonner la classe majoritaire (pour les grandes bases de données)

3) sur-échantillonner la classe minoritaire (tirage aléatoire AVEC remise)

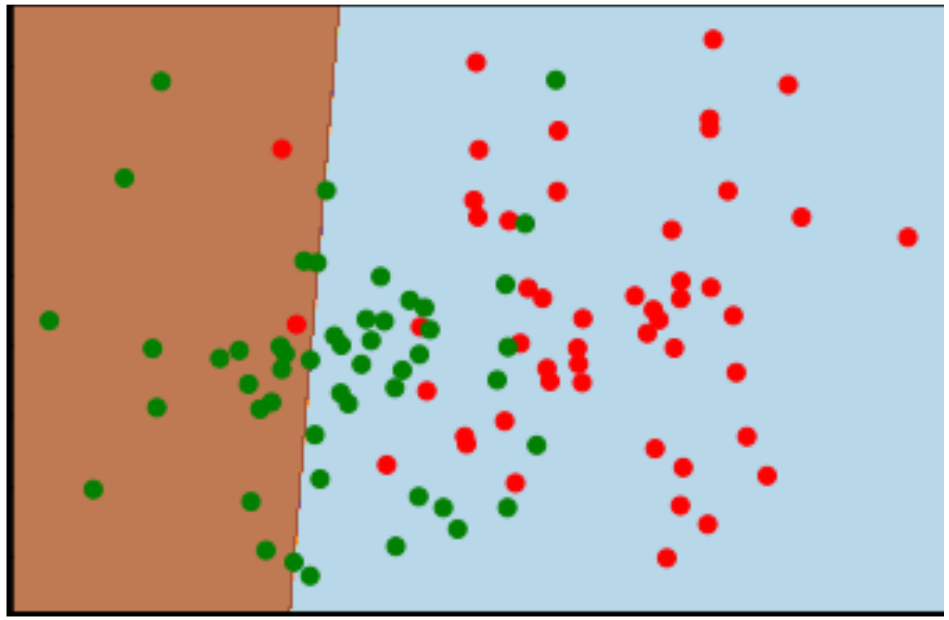
(…)

4) pénaliser les erreurs/coûts : $E = \sum \mathbf{c_i} (y_i - y_{di})^2$

Exemple :

$\mathbf{c_1 = 1}$
$\mathbf{c_2 = 0.1}$

(*class_weight*)



Data augmentation

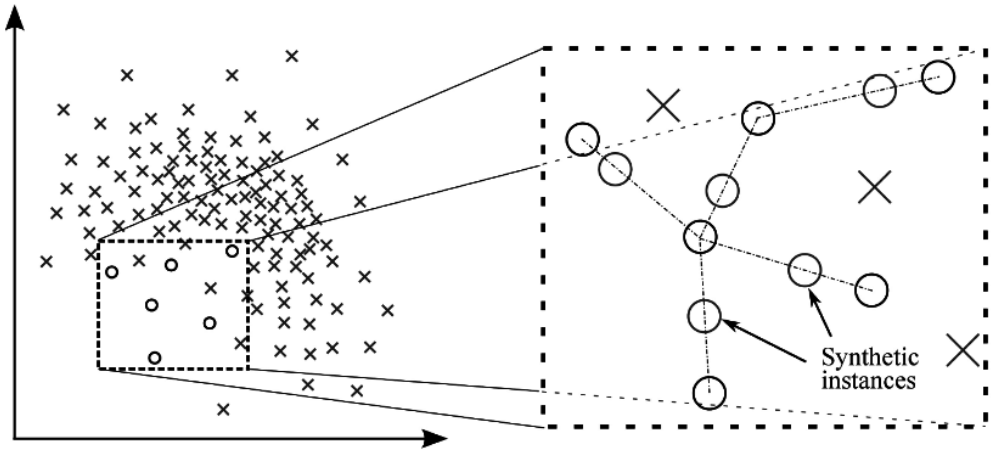
5) générer des données synthétiques

- Ajouter du bruit

- Problem-driven :

(transformations géométriques : translation/rotation…)

- Data-driven:
  - SMOTE*
  - Modèles génératifs



(*) Synthetic Minority Over-Sampling TEchnics

Gradient stochastique

(*Stochastic Gradient Descent : SGD*)

Répéter pour tous les exemple de la base d'apprentissage

propagation (calcul des sorties)

rétro-propagation (calcul du gradient)

modification des poids

→ **Les poids sont mis à jour après chaque présentation d'un exemple**

Gradient total (*batch*)

Répéter

Pour tous les exemple de la base d'apprentissage

propagation (calcul des sorties)

accumulation du gradient

rétro-propagation

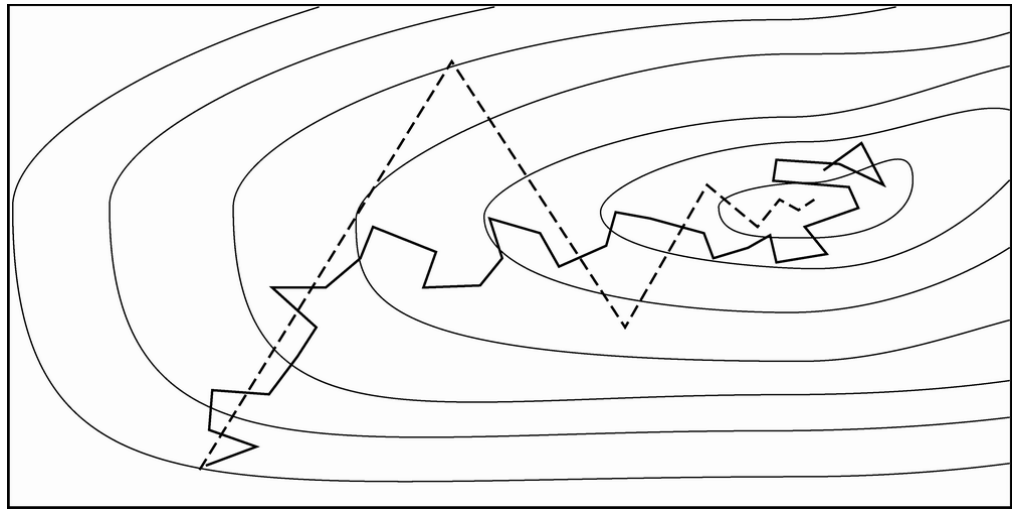
modification des poids

→ Les gradients sont accumulés sur toute la base d'apprentissage avant de mettre à jour les poids

Comparaison

Évolution des poids d'un réseau pendant l'apprentissage :

- gradient stochastique (trait continu) : changements de direction fréquents
- gradient total (pointillé)



SGD :

☹ Non parallélisable, bruité

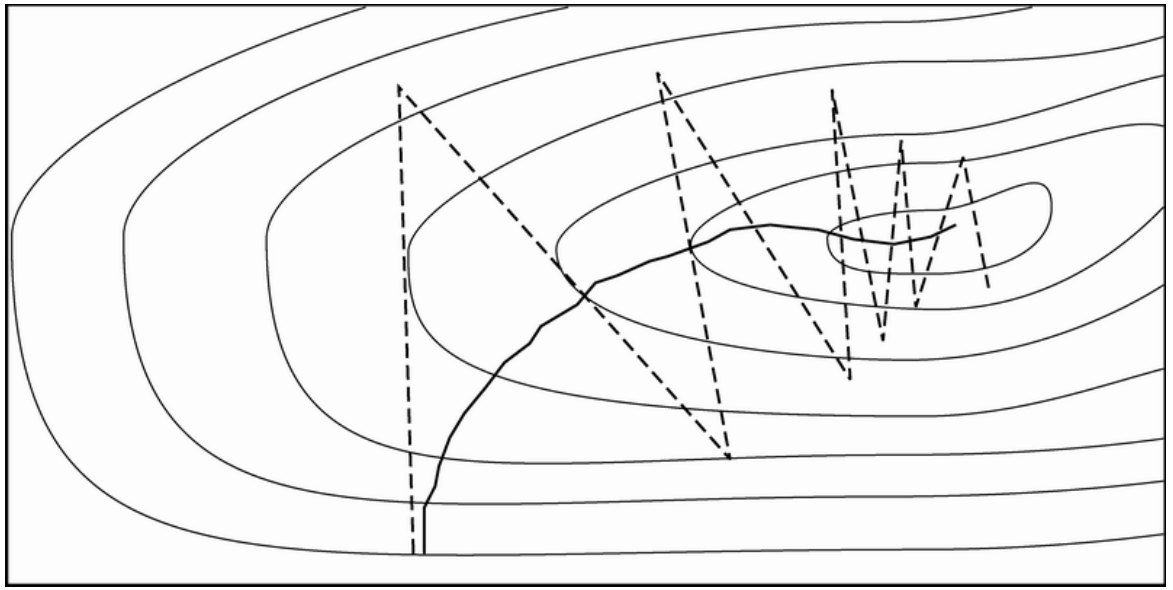
☺ Meilleure généralisation
(évite les minimas locaux)

Compromis → **« mini batch »** : entrainement séquentiel sur des sous-parties indépendantes (tirage aléatoire sans remise, stratifié) de la base d'apprentissage

Pas d’apprentissage

évolution des poids d’un réseau pendant l’apprentissage :

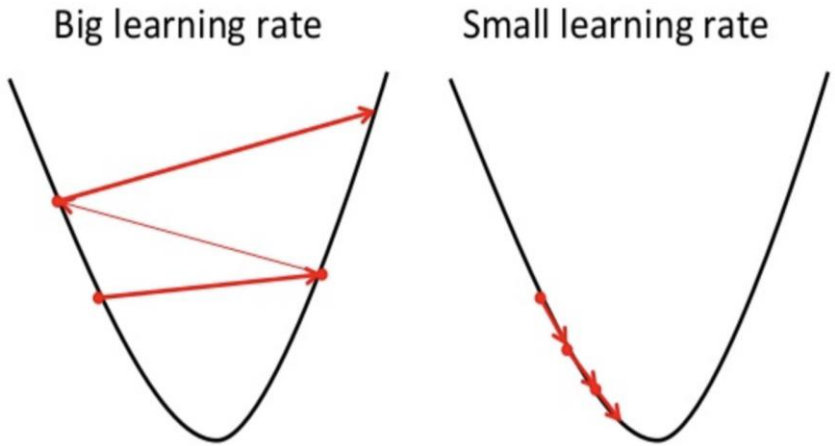
- pas d’apprentissage trop faible (trait continu) : nécessite beaucoup d’itérations
- pas d’apprentissage trop important (pointillé) : risque de rater le minimum



Règles d'adaptation

- Solution 1 (triviale : *invscaling*) :

$$\omega_{ij}(t+1) = \omega_{ij}(t) - \lambda_t \frac{\partial E}{\partial \omega_{ij}} \qquad \text{avec } \lambda_t = \lambda_0 / t$$



- Solution 2 (*adaptive*) :

Si E croissant : $\lambda_{t+1} = \lambda_t * C_{dec}$ (ralentir → $C_{dec} = 0.9$)

Si E décroissant : $\lambda_{t+1} = \lambda_t * C_{inc}$ (accélérer → $C_{inc} = 1.1$)

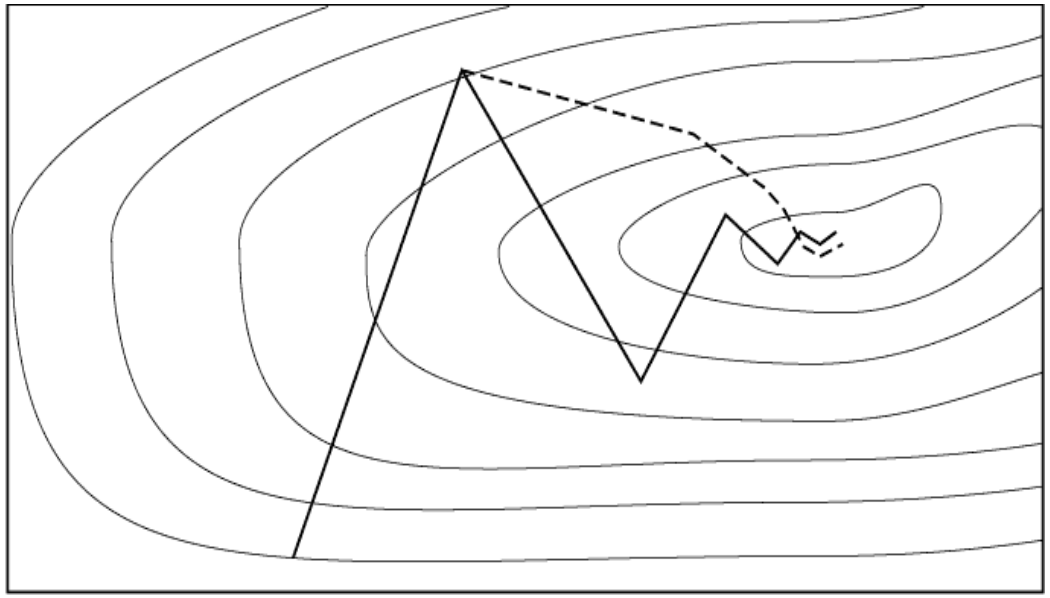
(…)

- Solution 3 (*gradient descent with momentum*) :

la modification d'un poids reprend en partie
la modification antérieure :

$$\omega_{ij}(t+1) = \omega_{ij}(t) - \lambda_t \frac{\partial E}{\partial \omega_{ij}} + \mu \Delta \omega_{ij}(t-1)$$

avec $\mu < 1$



(…)

• solution 4 : adagrad, rmsprop (→ tutoriel)

• solution 5 (*lbfgs*) :

Méthodes du second ordre : $\omega(t+1) = \omega(t) - H_t(E)^{-1}\nabla E$

H_t (E) est la matrice Hessienne (des dérivées secondes de E par rapport à ω_{ij})

PB : inversion d'une matrice (*qxq**) à chaque itération !

→ Approximation de H_t^{-1} (quasi-Newton)

→ méthode de Broyden-Fletcher-Goldfard-Shanno (BFGS)

(*) q : nombre de paramètres libres

Paramètres du réseau

Nombre de neurones d'entrée et de sortie: dépend du problème à résoudre

→ Nombre d'entrée = nombre de variables prédictives (*features*)

→ Nombre de sortie = nombre de classe

Nombre de couches cachées déterminé par recherche exhaustive mais:

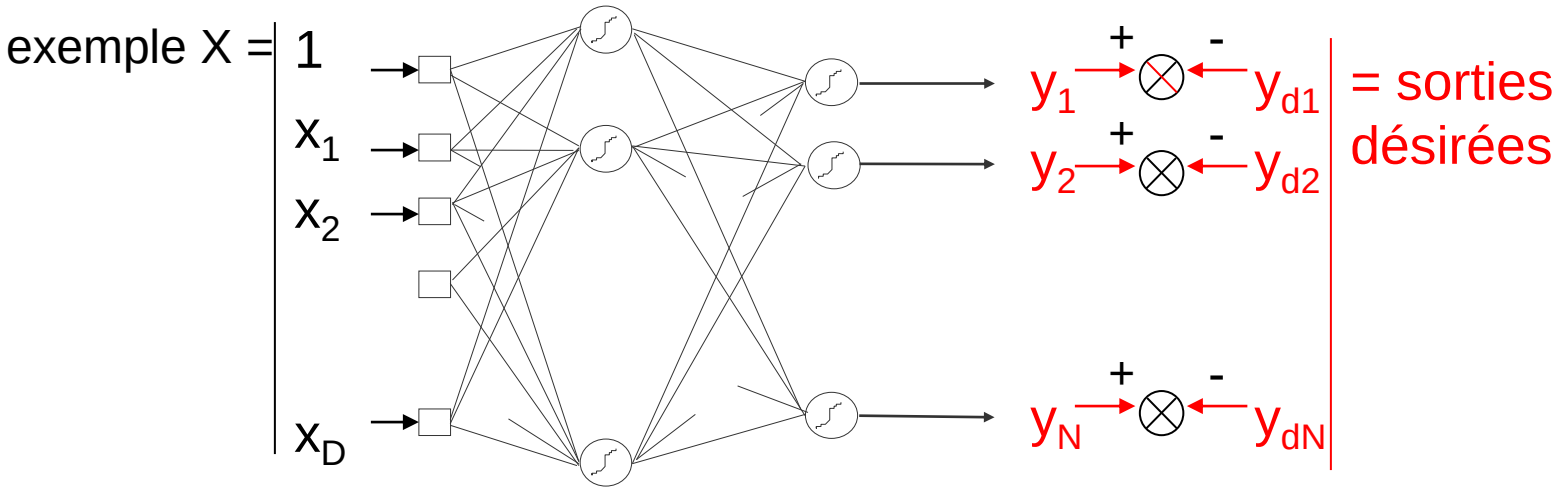
Pour des problèmes de complexité «raisonnable», une ou deux couches suffisent

Au-delà, phénomène de disparition du gradient (*vanishing gradient* → solution : cours n°4)

Initialisation des poids (*Xavier initialization*) : aléatoire entre $-1/\sqrt{M}$ et $1/\sqrt{M}$

où M est le nombre d'entrée du neurone (afin d'éviter la saturation)

(…)



1 exemple X → N équations non linéaires

BORNE : Nombre de paramètres libres ≤ Nb_appx N

Fun time

1) On souhaite reconnaître des images de fruits de résolution 50x50, réparties en trois classes : pommes, mandarines et prunes.
On utilise un réseau de neurones monocouche pour réaliser cette tâche.
Combien de paramètres libres (poids et biais) sont nécessaires ?

2) Même question avec une couche cachée de 10 neurones

3) Même question avec deux couches cachées de 100 et 20 neurones.

Fun time

3) On souhaite reconnaître des données en dimension 200, distribuées en 2 classes.
On utilise un réseau de neurones (sans biais) multicouche à 10 neurones cachés pour réaliser cette tâche.
Combien faut-il d'exemples d'apprentissage ?

4) On souhaite reconnaître des données en dimension 200, distribuées en 2 classes.
On utilise un réseau de neurones (sans biais) multicouche à C neurones cachés pour réaliser cette tâche.
On dispose de 2000 exemples d'apprentissage.
Quelle est la valeur maximum de C (arrondi à l'entier le plus proche) ?

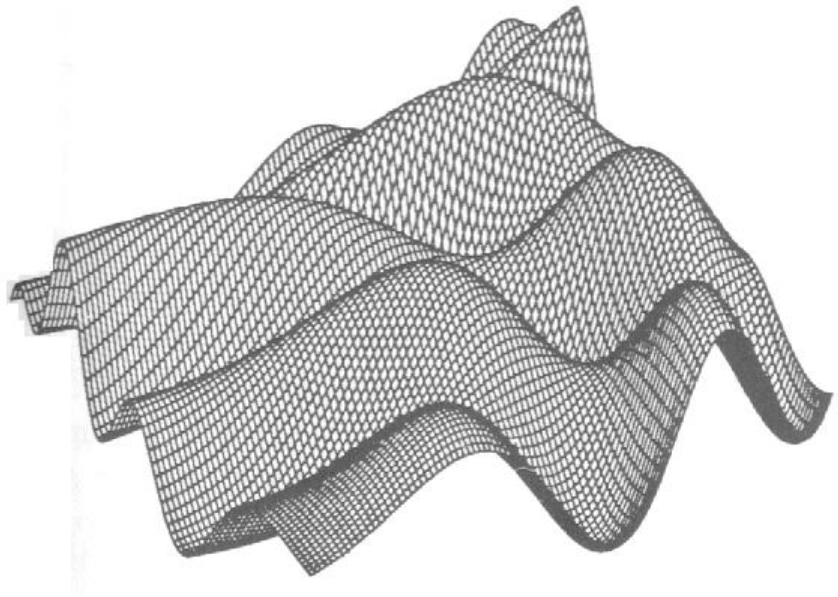
Influence de l'initialisation aléatoire

Ici : $E(\omega_1 , \omega_2)$

Fonction fortement non convexe

→ Nombreux minima

→ Présence de vallées où le gradient varie peu

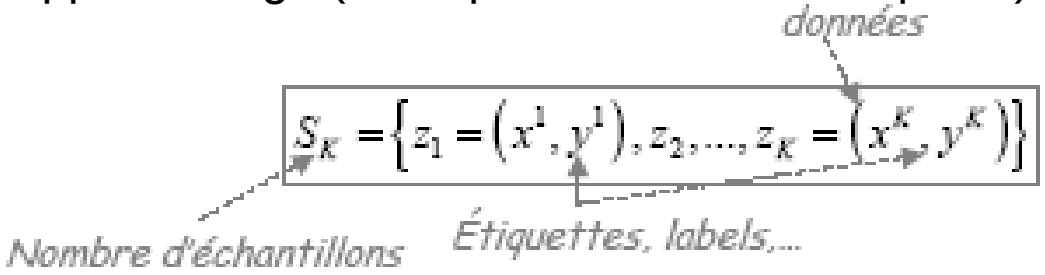


→ Tester plusieurs réalisations (initialisations) de l'algorithme

→ Peut nécessiter de nombreuses itérations pour converger … ou pas !

Influence de l'architecture du modèle

Échantillon d'apprentissage (K couples observation/étiquette) :



Il existe une fonction *f* (appartenant à une famille de fonctions *F*) réalisant l 'association entre les entrées x^k et les label y^k.

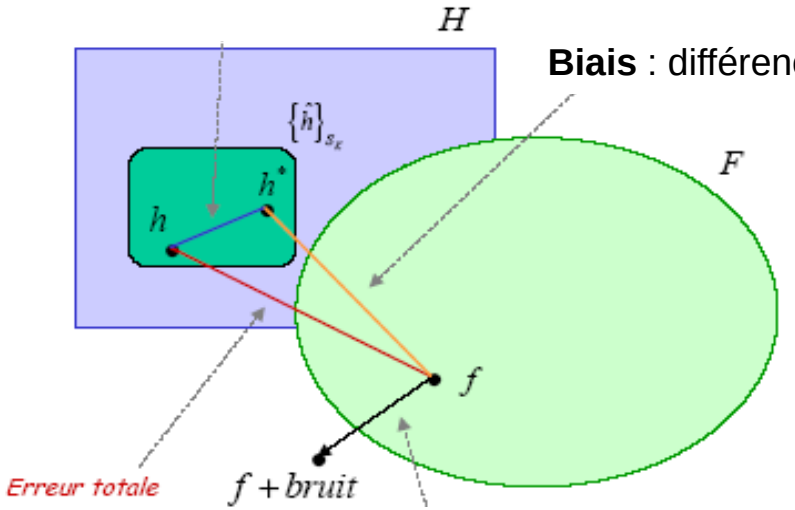
→ L'apprentissage vise à trouver une fonction hypothèse *h* (appartenant à une famille de fonctions *H*), **le plus proche possible** de *f* :

$\hat{y}^k = h(x^k)$ **minimisant une fonction de perte *L*(S_k, *h*) (exemple : EQM)**

(…)

- Les modèles « simples » (H très différent de F) présentent un biais fort, mais une faible variance
- Les modèles « complexes » (nombreux paramètres à estimer) présentent un biais faible, mais une forte variance

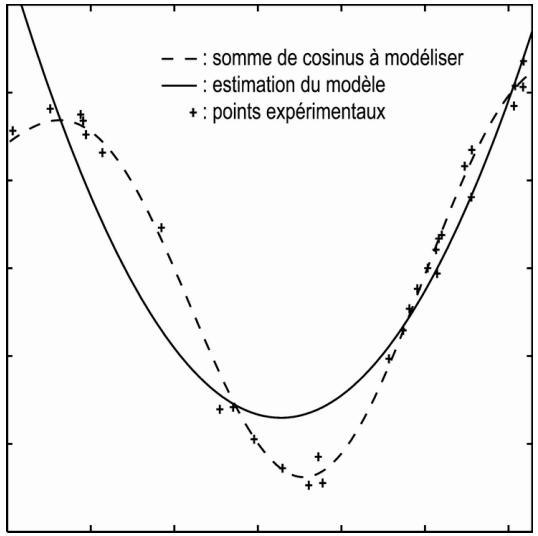
Variance augmente avec *H*



Biais : différence entre *F* et *H*

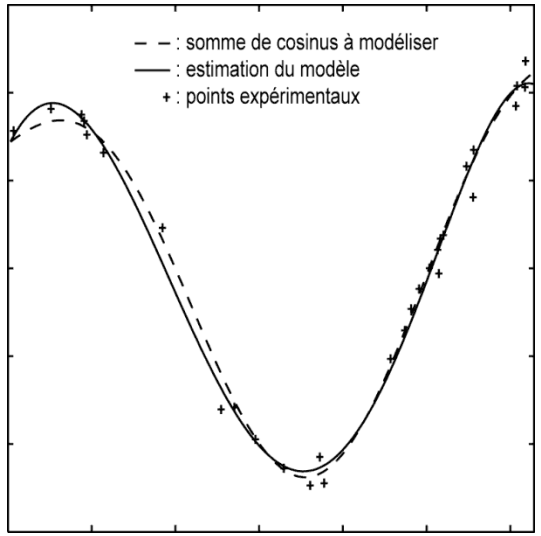
complique la recherche de *h**

Exemple en régression



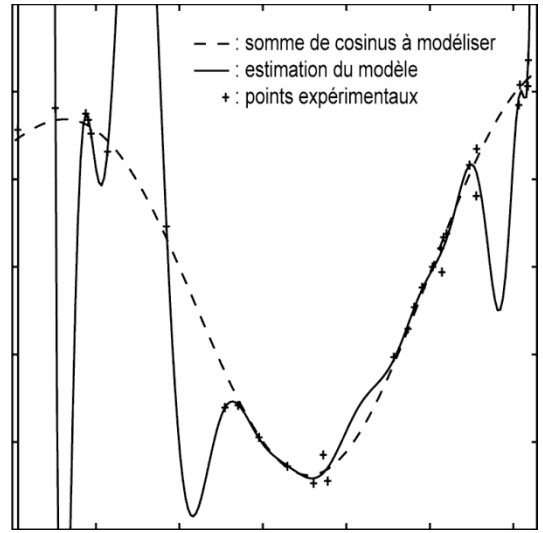
A

Modèle sous dimensionné (2 cellules, biais élevé mais variance faible).



B

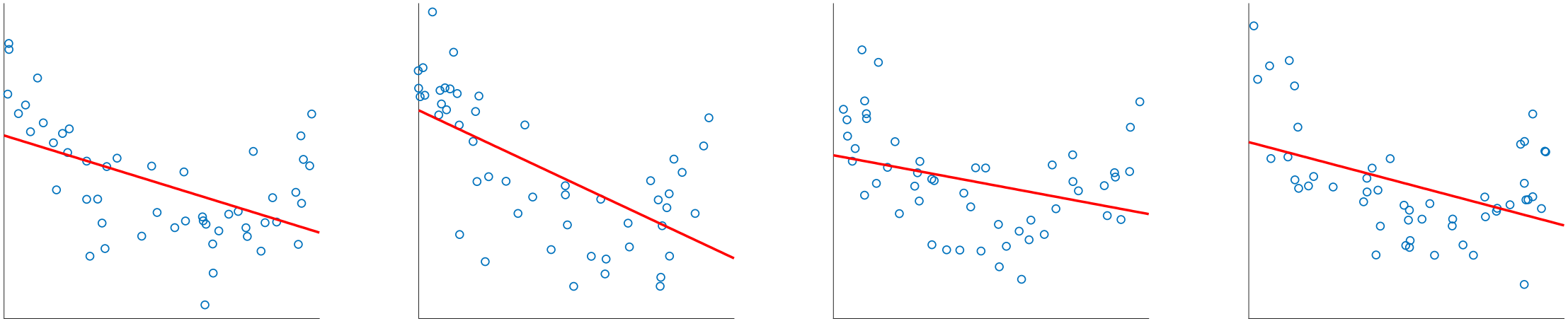
Modèle bien dimensionné (5 cellules).



C

Modèle sur dimensionné (20 cellules, biais faible mais variance élevée).

Focus : le biais



Sur les données d'apprentissage :

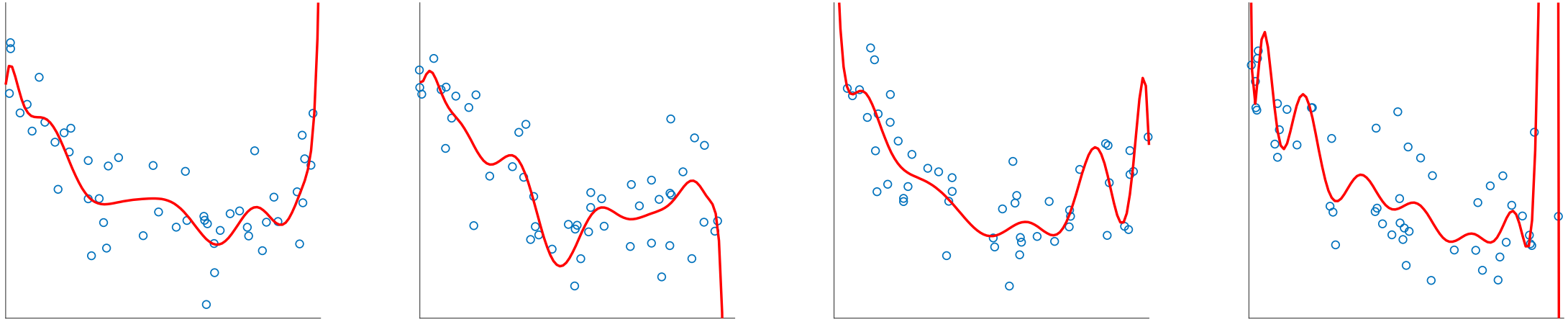
$$\text{Biais}\left[\hat{f}(x)\right] = \text{E}\left[\hat{f}(x) - f(x)\right]$$

Indépendamment des données d'apprentissage, le prédicteur est :

- Trop « simple » (biais fort)
- Varie peu avec les données d'apprentissage (variance faible)

→ Sous-apprentissage (under-fitting)

Focus : la variance

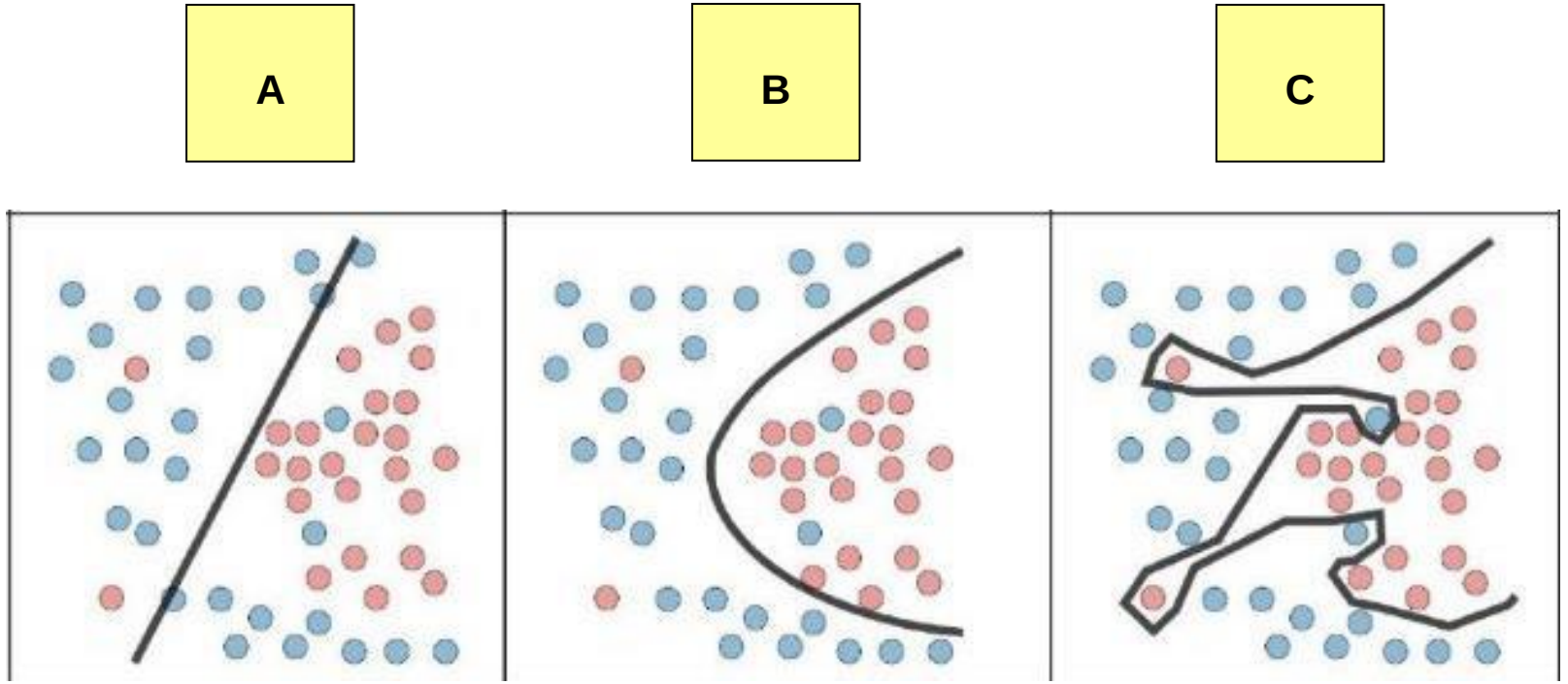


$$\mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{E}\left[\left(\hat{f}(x) - \mathrm{E}[\hat{f}(x)]\right)^2\right]$$

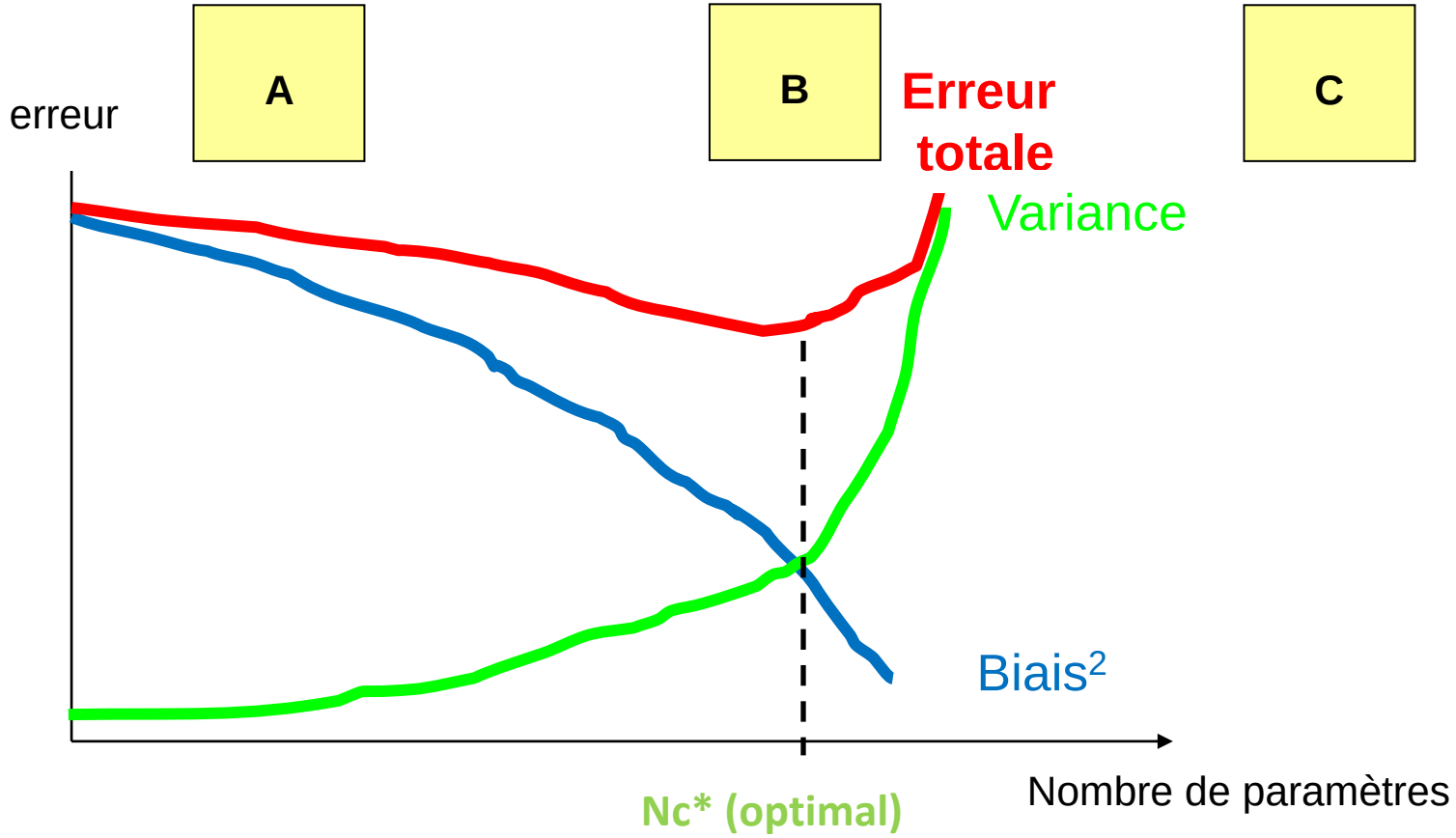
Lorsque l'ensemble d'apprentissage change
- Le prédicteur change fortement (variance forte)
- Le prédicteur apprend « par cœur » les données

→ Sur-apprentissage (over-fitting)

Exemple en classification



Le compromise biais/variance



(…)

L'objectif de l'apprentissage est de minimiser simultanément le biais et la variance du modèle

(possible statistiquement si la taille de la base d'apprentissage tend vers l'infini)

Dans la réalité, base de taille finie !

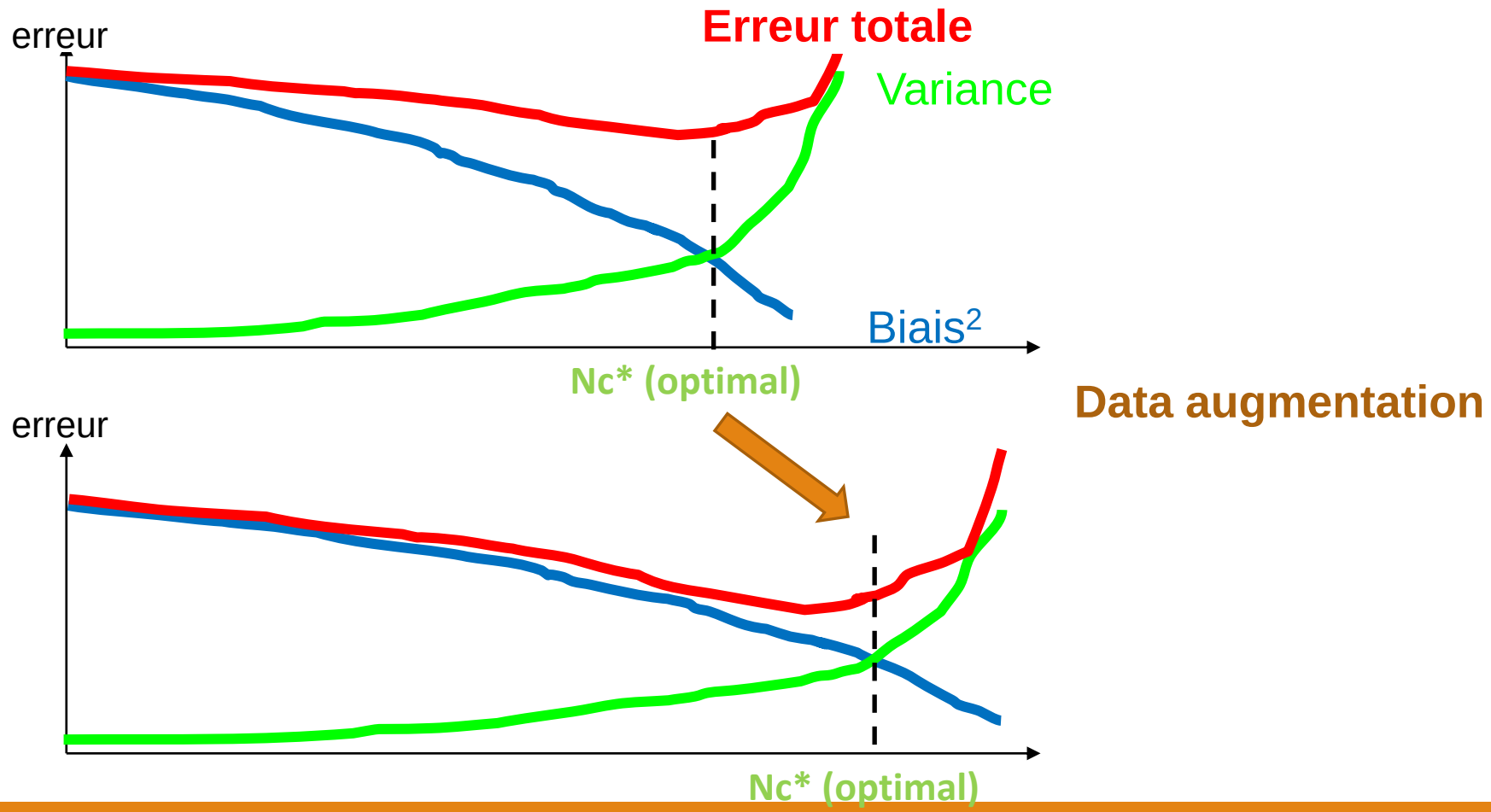
→ **réseau sous-dimensionné : biais élevé** ☹

→ **réseau surdimensionné : variance élevée** ☹

→ **Data augmentation**

→ **Rechercher des solutions parcimonieuses (*rasoir d'Ockham*)**

Augmentation de données



Data augmentation

Solutions parcimonieuses

Stabilisation structurelle :

Contrôler la complexité du réseau par élagage de cellules

- *optimal brain damage*
- *drop out*

Régularisation :

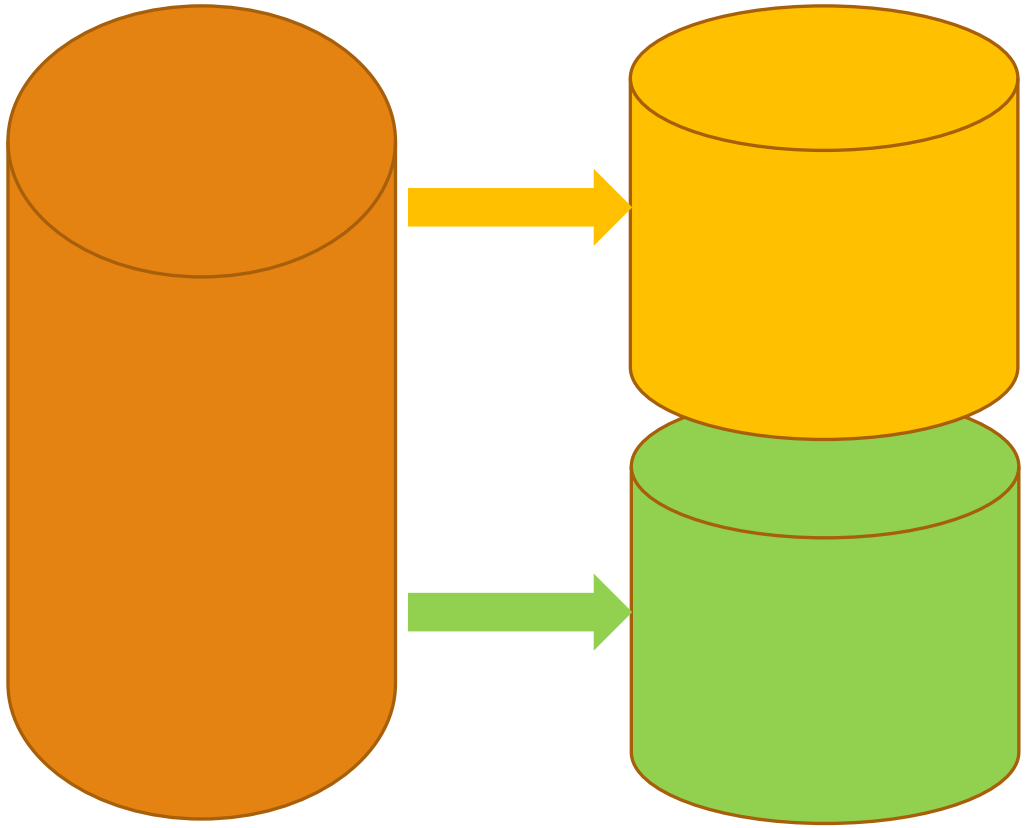
Pénaliser les poids : des poids plus faibles réduisent l'impact des neurones cachés. Dans ce cas, ces neurones deviennent négligeables et la complexité globale du réseau est réduite.

- *L1 regularization*
- *L2 regularization*

→ Arrêt **précoce** de l'apprentissage : *early stopping*)

→ Contrôler l'apprentissage par adjonction à la fonction de coût d'un critère d'arrêt

Sur quelle base ?



Apprentissage = estimation des poids du réseau

→ **Risque de sur-apprentissage**

Généralis pacité du réseau à es données inconnues

→ **Perform es c tionnelles**

Ensemble de validation

Pour estimer les performances, il suffit de disposer d'une base **indépendante** de celle d'apprentissage.

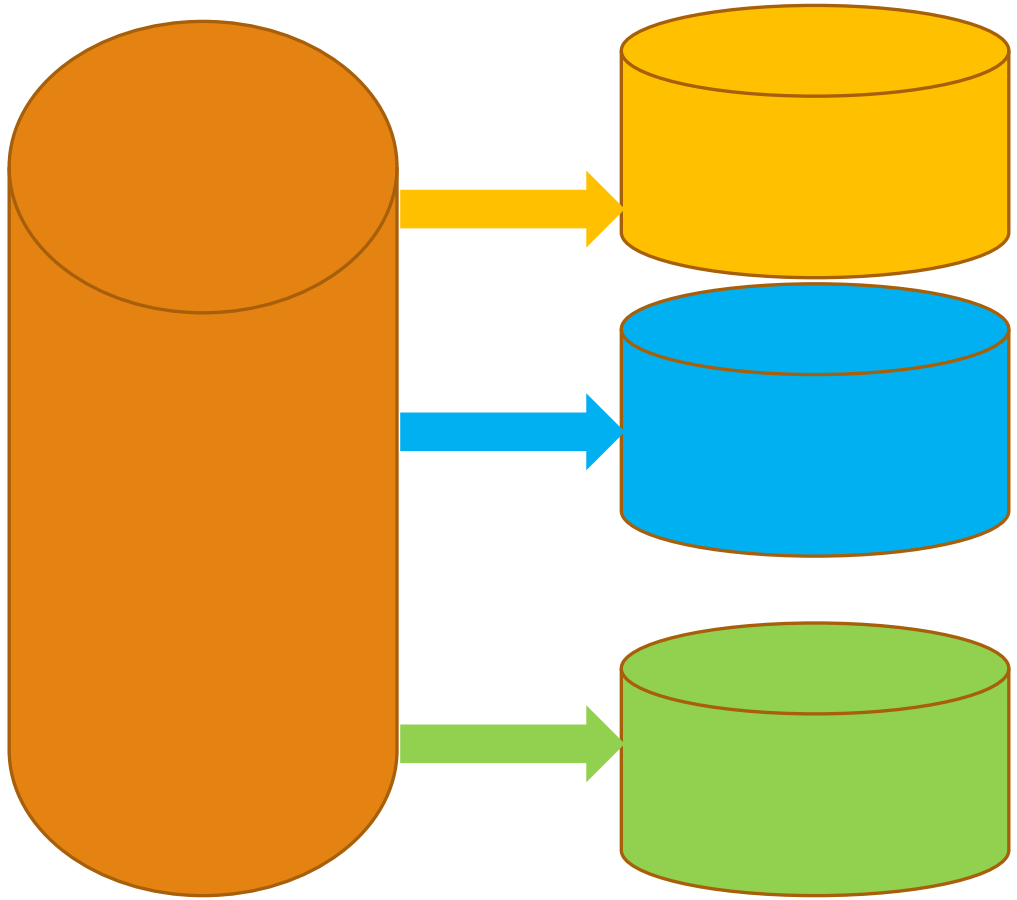
Principe :

(1) Découper la base d'apprentissage en deux parties

- une partie utilisée pour l'apprentissage
- une partie pour le réglage des paramètres appelée **ensemble de validation** *(validation_fraction)*

(2) Estimer régulièrement, pendant l'apprentissage, les performances sir l'ensemble de validation.

(…)



Apprentissage = estimation des poids du réseau

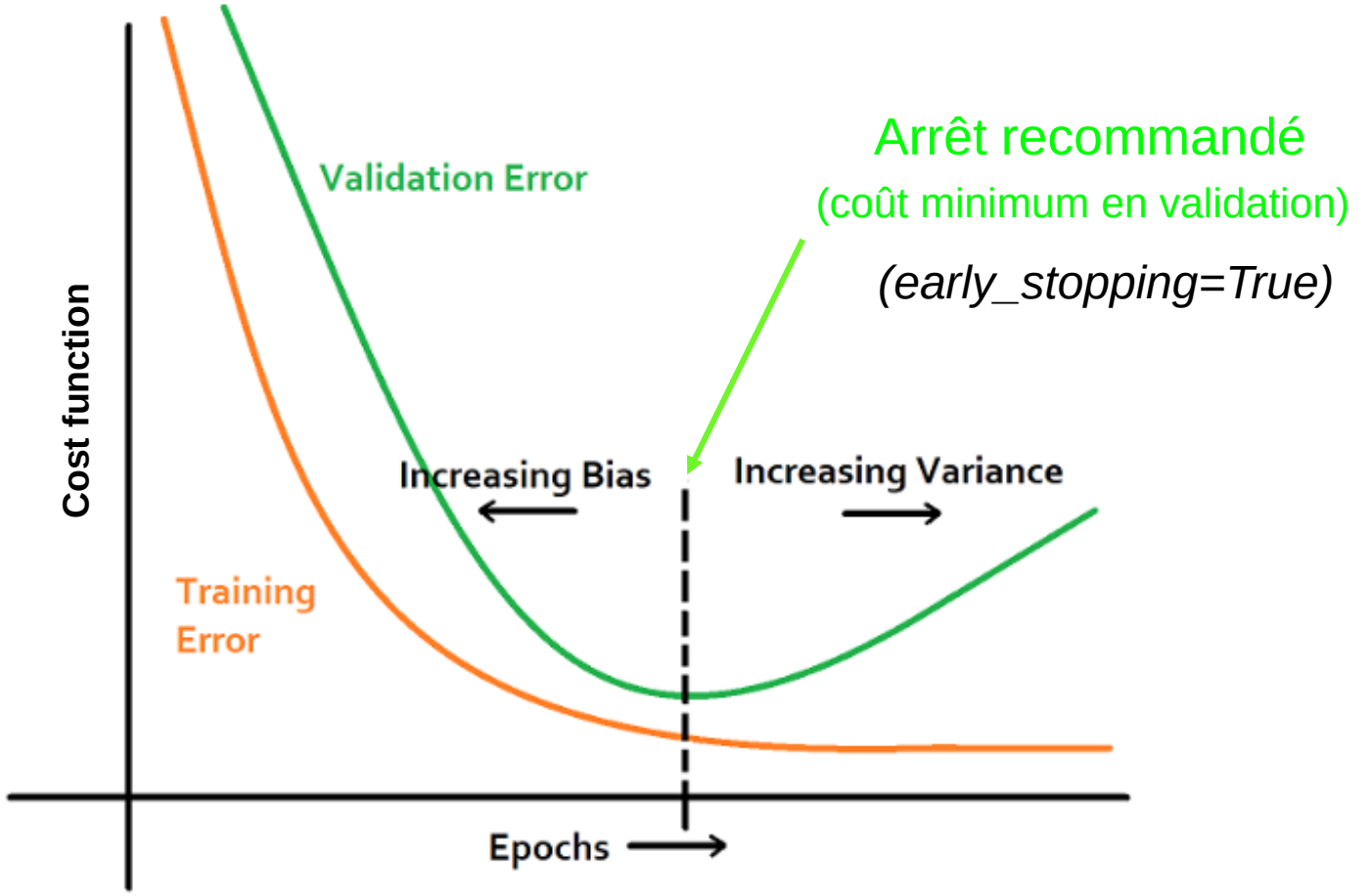
Validation : mesure des performances en généralisation pendant l'apprentissage

→ **Régularisation**

Généralisation = capacité du réseau à classer des données inconnues

→ **Performances opérationnelles**

(...)


Validation Error
Arrêt recommandé
(coût minimum en validation)
(early_stopping=True)
Cost function
Increasing Bias
Increasing Variance
Training
Error
Epochs

Bilan

Apprentissage supervisé : classe connue

- → cas binaire, linéairement séparable : perceptron (1 neurone)
- → cas multiclasse (N classes), linéairement séparable : réseau monocouche (N neurones)

- → cas non linéairement séparable : réseau multicouche (C neurones cachés)
  - → Préparation des données
  - → Apprentissage par rétro-propagation
  - → Décision : Winner Takes all ou variante

  - → **Estimer les performances (taux d'erreur, risque, courbe ROC …)**
  - → **Recherche exhaustive de l'architecture optimale par validation croisée**

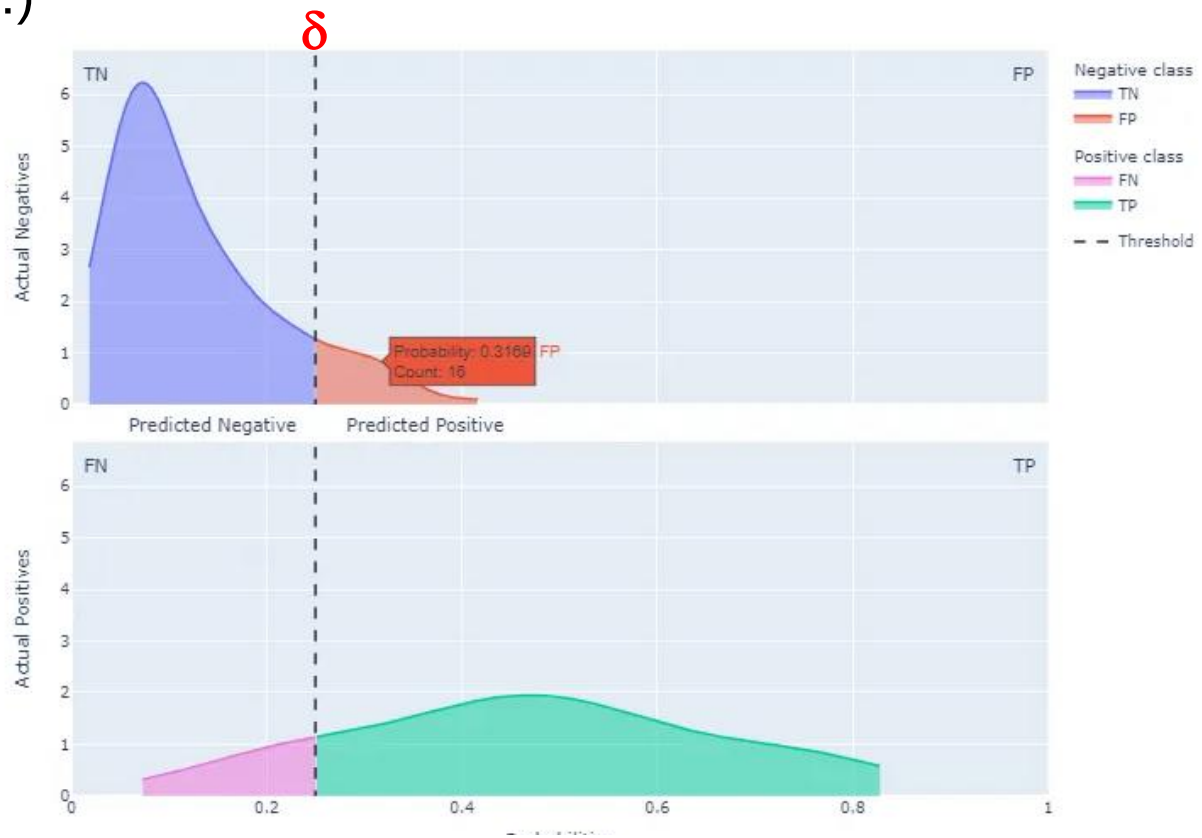
Performances : cas binaire

Problèmes de détection (visuelle, diagnostic …)

Décision :

- Si P(C|X) > δ *predict +*
- Sinon *predict -*

	+	-
+	TP	FN
-	FP	TN



https://towardsdatascience.com/roc-and-pr-curves-probabilities-distribution-and-density-plots-now-in-binclass-tools-python-9351681a3803

Courbe ROC

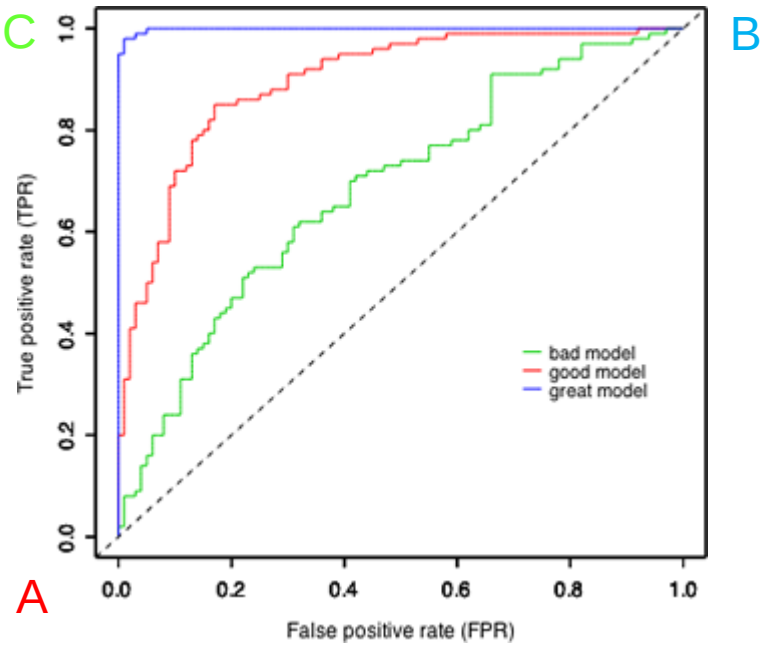
	+	-
+	TP	FN
-	FP	TN

True Positive Rate :

$$\text{TPR} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

False Positive Rate :

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$



→ **Courbe ROC {FPR(δ), TPR(δ)}**
(Receiving Operator Caracteristic)

→ **AUC : comparaison de classifieurs**
(Area Under the Curve)

A : prédit toujours négatif
B : prédit toujours positif
C : objectif (TPR=1, FPR=0)

Courbe Précision-Rappel

	+	-
+	TP	FN
-	FP	TN

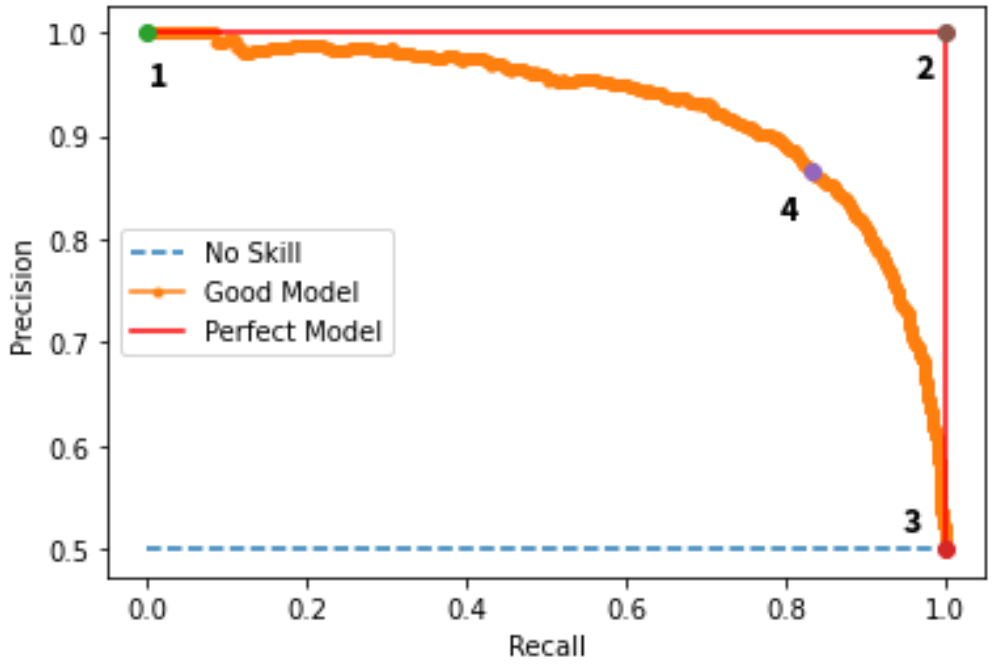
$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

→ **Courbe PR**

→ **AUPRC**

→ **F1 score :** $F1 = \frac{2 \times precision \times recall}{precision + recall}$



Fun time

Compute carefully TPR, FPR, Precision and Recall for classifier I & II.

Classifier I

	Tr = 0	Tr = 1
Pr = 0	598500	200
Pr = 1	400500	800

Classifier II

	Tr = 0	Tr = 1
Pr = 0	547000	150
Pr = 1	452000	850

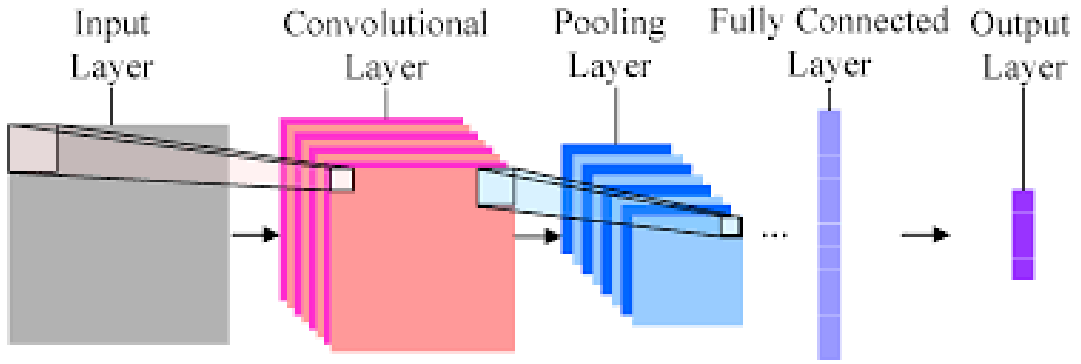
Going deeper : https://towardsdatascience.com/demystifying-roc-and-precision-recall-curves-d30f3fad2cbf

1990 : Réseaux à convolution

Objectifs :

→ Réduction du nombre de poids (→ meilleure généralisation)

→ Apprentissage de représentation



Principes :

→ connexions locales, poids partagés

→ couches **profondes** : extraction d'information (contours …)

→ couches de sortie : complètement connectées (denses) pour la décision

→ **deep neural networks**

Connexions locales

Les données ont très souvent des propriétés locales qui se traduisent par des dépendances

- Temporelles (signal de parole)
- Spatiales (images)

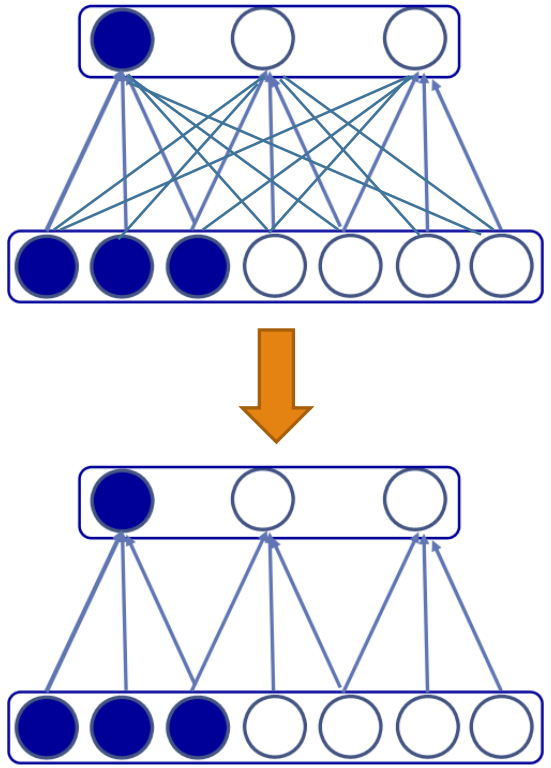
entre des attributs voisins (ex : pixels de l'image)

Les connexions **totales** ne sont pas adaptées !

il vaut mieux utiliser des connexions **locales**

(phénomène observé dans le cortex visuel)

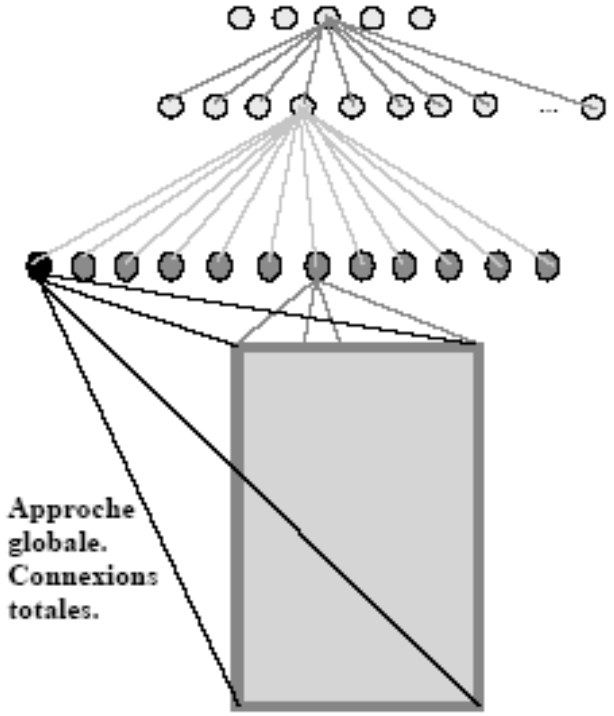
→ **Réduction du nombre de paramètres libres**

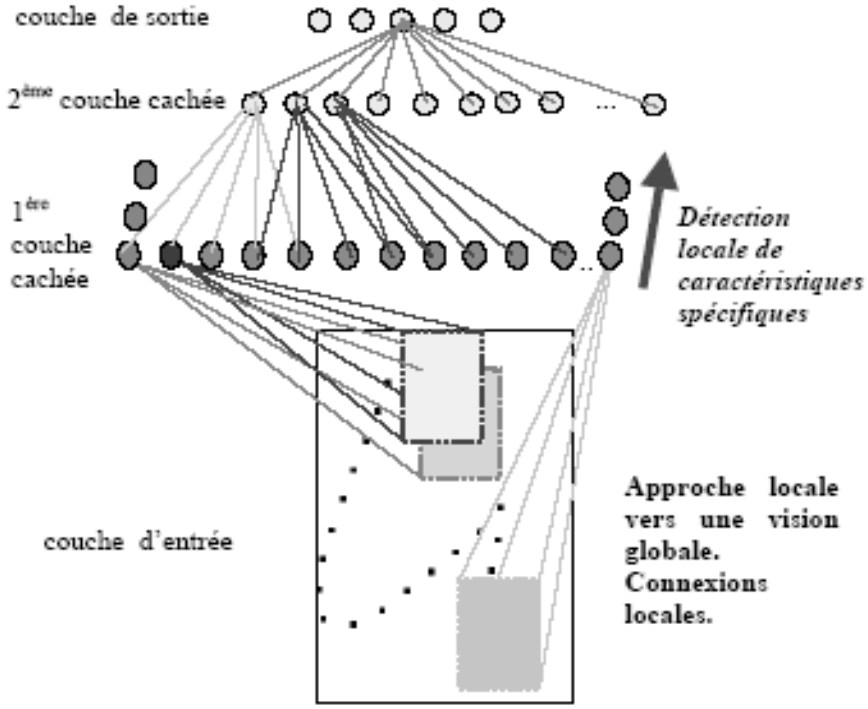


7*3 ☹

3*3 ☺

(…)


Approche globale. Connexions totales.

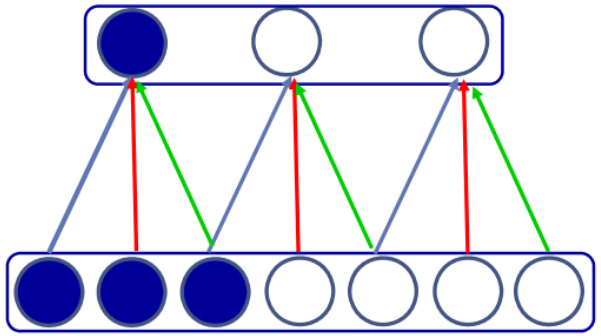

couche de sortie
2ème couche cachée
1ère couche cachée
Détection locale de caractéristiques spécifiques
couche d'entrée
Approche locale vers une vision globale. Connexions locales.

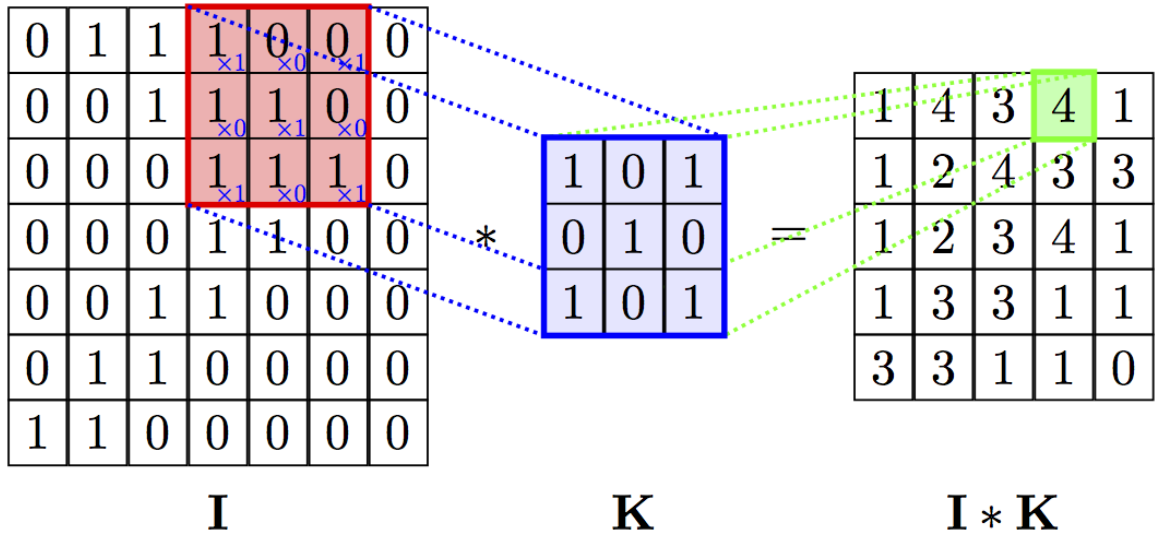
Poids partagés

Les connexions locales peuvent être contraintes à partager le même vecteur de poids

→ Poids partagés

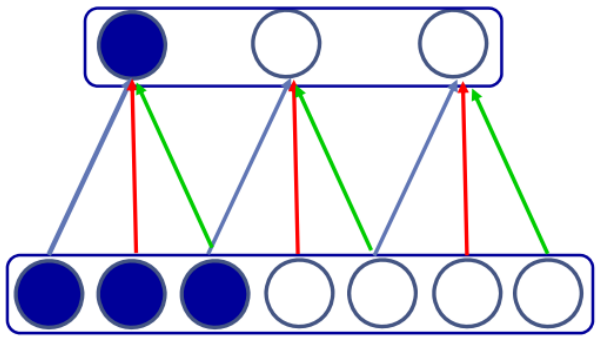
→ revient à convoluer le signal d'entrée à l'aide d'un masque de convolution unique

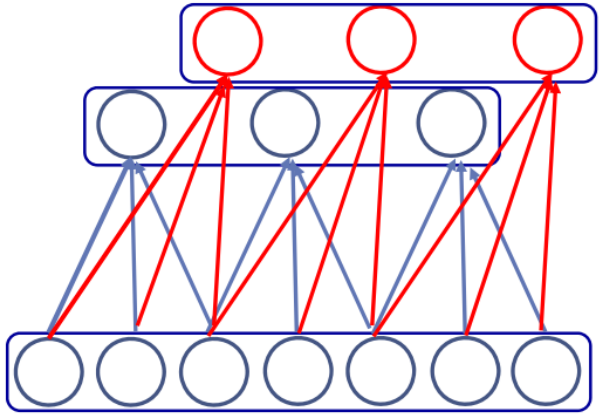




(…)

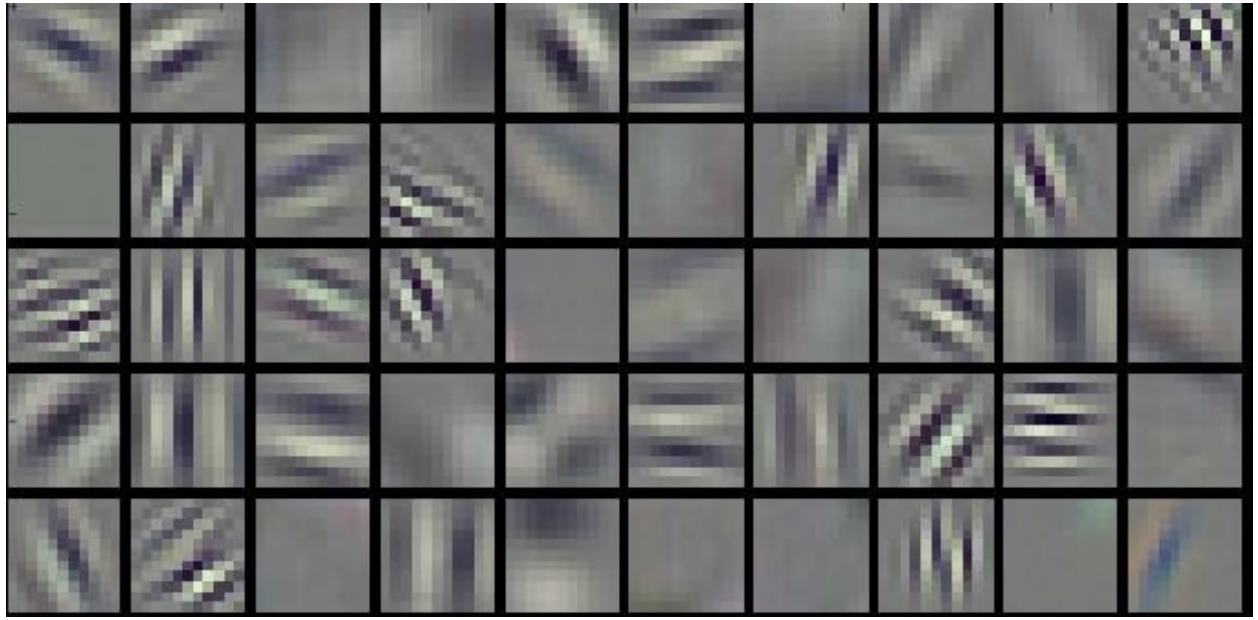
il est possible d’apprendre simultanément plusieurs masques (banc de filtres détecteur de contour en imagerie)





Interprétation cognitive

Sorties de la première couche



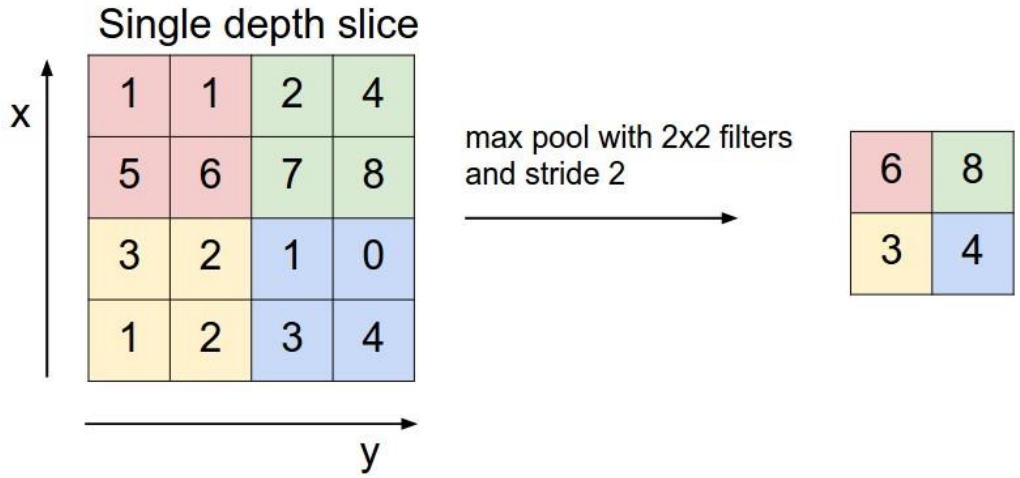
Aggrégation (*pooling*)

Si de nombreux masques dont utilisés, le nombre de caractéristiques extrait peut être très grand

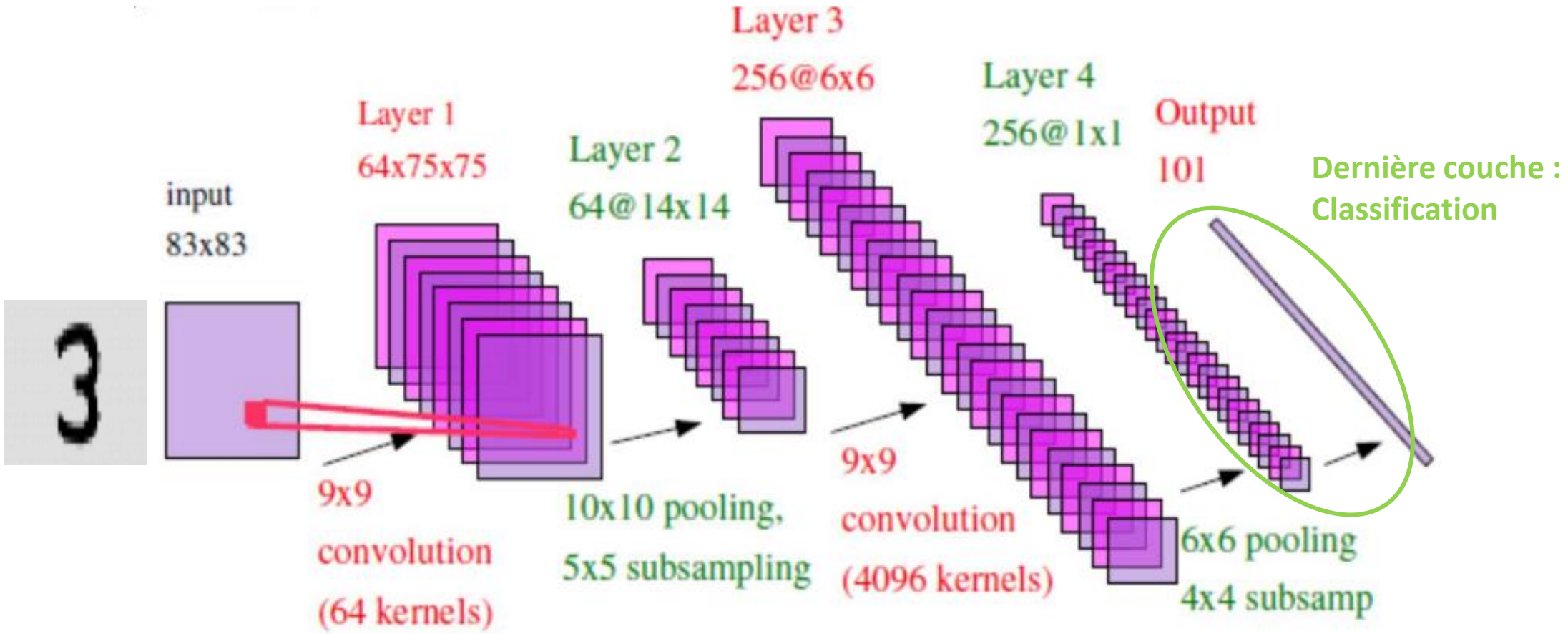
→ pour le réduire on va agréger les caractéristiques localement

→ opérateur d'agrégation : moyenne ou maximum

→ produit une invariance à la translation (des données dans l'image)



Architecture complete (LeNet90)



Limitations

☹ Algorithmiques : impossible d'apprendre des réseaux à plus de 3 ou 4 couches (phénomène de « disparition du gradient »)

☹ Mathématiques : le nombre de poids explose … il faut beaucoup de données

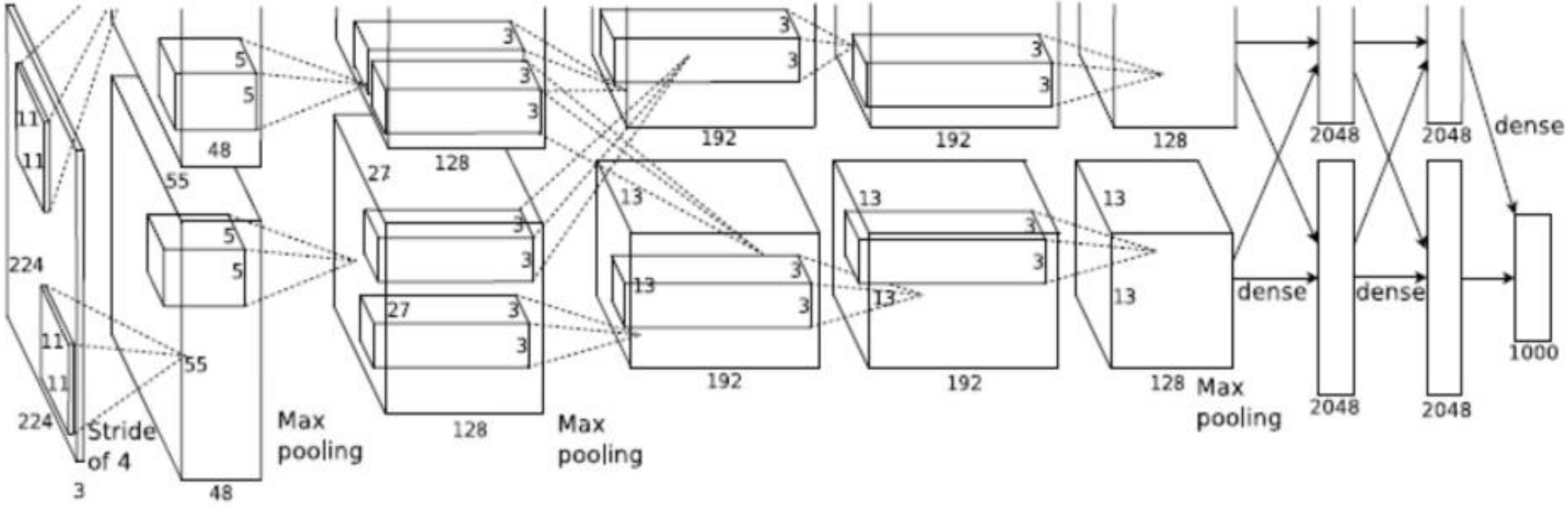
2ème « hiver » du connexionnisme

→ SVM, Bagging, Random Forest, Boosting

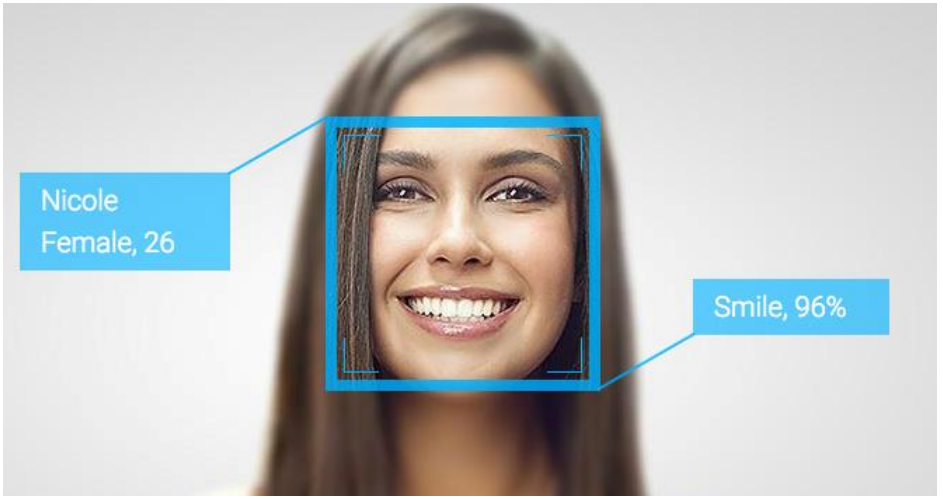
Deep ConvNet (ImageNet 2012)

14,197,122 images

- 650K neurons, 630M synapses, 60M parameters

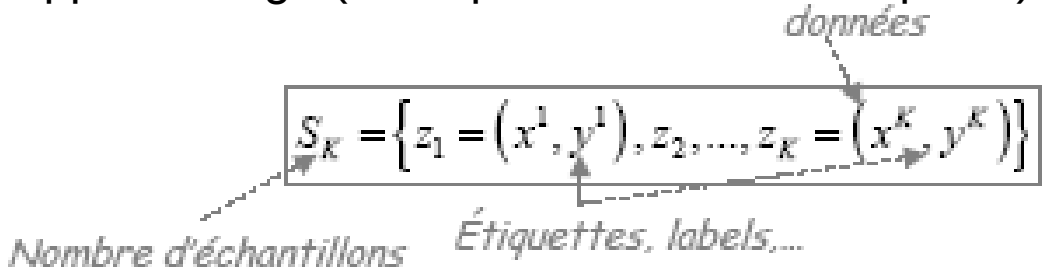


Applications



Pour approfondir :

Échantillon d'apprentissage (K couples observation/étiquette) :



→ Il existe une fonction *f* (appartenant à une famille de fonctions *F*) réalisant l 'association entre les entrées x^k et les étiquettes y^k.

→ L'apprentissage vise à trouver une fonction hypothèse *h* (appartenant à une famille de fonctions *H*), le plus proche possible de *f* :

$\hat{y}^k = h(x^k)$ minimisant une fonction de perte *L* : $L(z_k, h)$

Principe MRE

Objectif : trouver la fonction hypothèse *h* qui minimise le **risque réel** *R* mesure statistique de perte définie sur un espace de données probabilisé (distance entre *h* et *f*)

$$R(h) = \int L(z, h)\, p(z) dz$$

p(z) est inconnue → *R* n'est pas mesurable !

Principe de Minimisation du Risque Empirique : on mesure le risque empirique (perte, coût moyen, taux d'erreur) d'une hypothèse particulière sur un échantillon d'exemples donné S_K :

$$\hat{R}(h, S_K) = \frac{1}{K} \sum_{k=1}^{K} L(z_k, h)$$

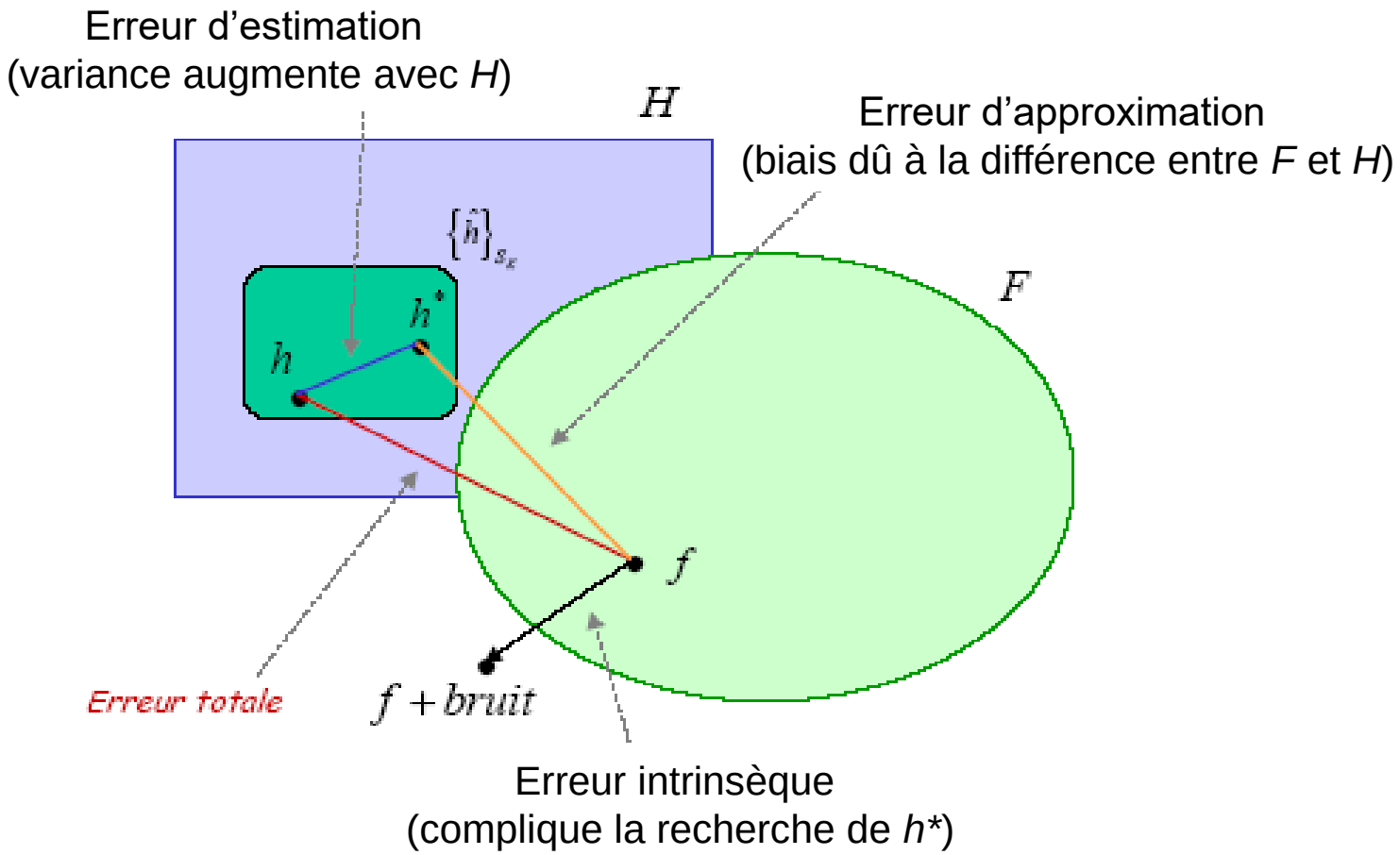
→ **On induit que l'hypothèse qui minimise le risque empirique, minimise également le risque réel**

Erreur d'apprentissage

On utilise le même échantillon S_K

(==base d'apprentissage) pour :

- **Trouver les paramètres de *h***
- **Estimer le risque**

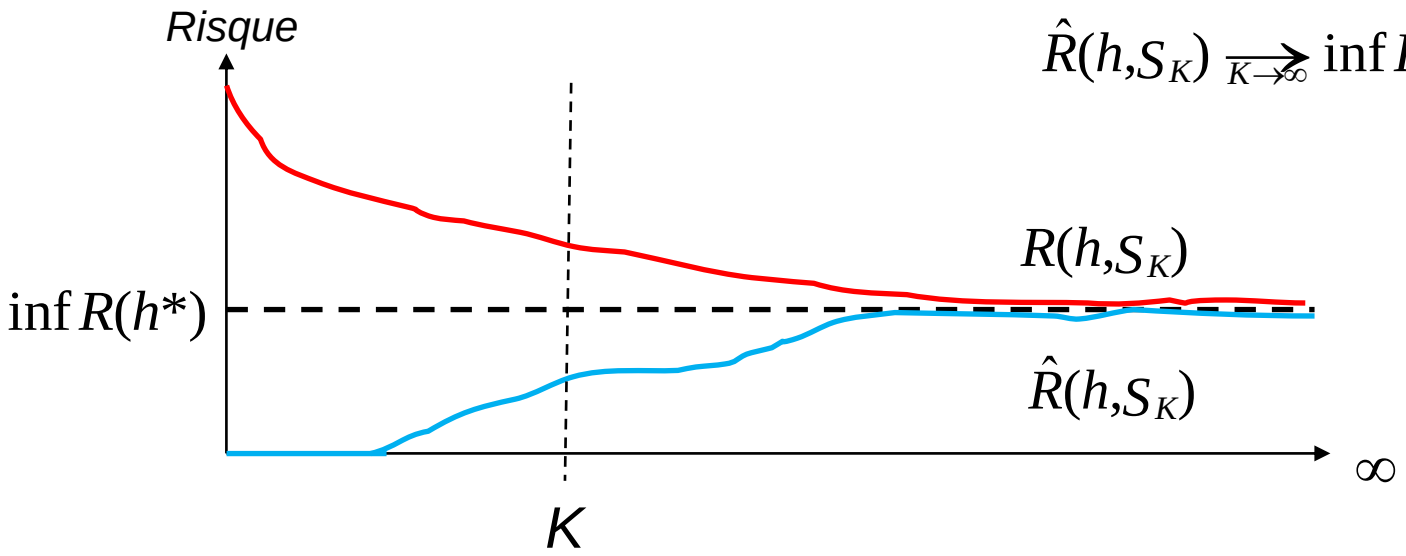


Consistance

Cadre PAC (*Probablement Approximativement Correct*) : l'apprentissage est consistant si l'apprenant fait (très probablement) au mieux de son possible quand la taille de l'échantillon tend vers l'infini

$$R(h,S_K) \xrightarrow[K\to\infty]{} \inf R(h^*)$$

$$\hat{R}(h,S_K) \xrightarrow[K\to\infty]{} \inf R(h^*)$$



Ensemble de validation

Pour estimer le risque, il faut disposer d'un échantillon **indépendant** de celui d'apprentissage.

Principe : découpage de l'échantillon en deux parties

- une partie utilisée pour l'apprentissage
- une partie pour le réglage des paramètres appelée **ensemble de validation croisée**

Le risque empirique sur l'ensemble de validation est un estimateur du risque réel.

Sélection de modèles

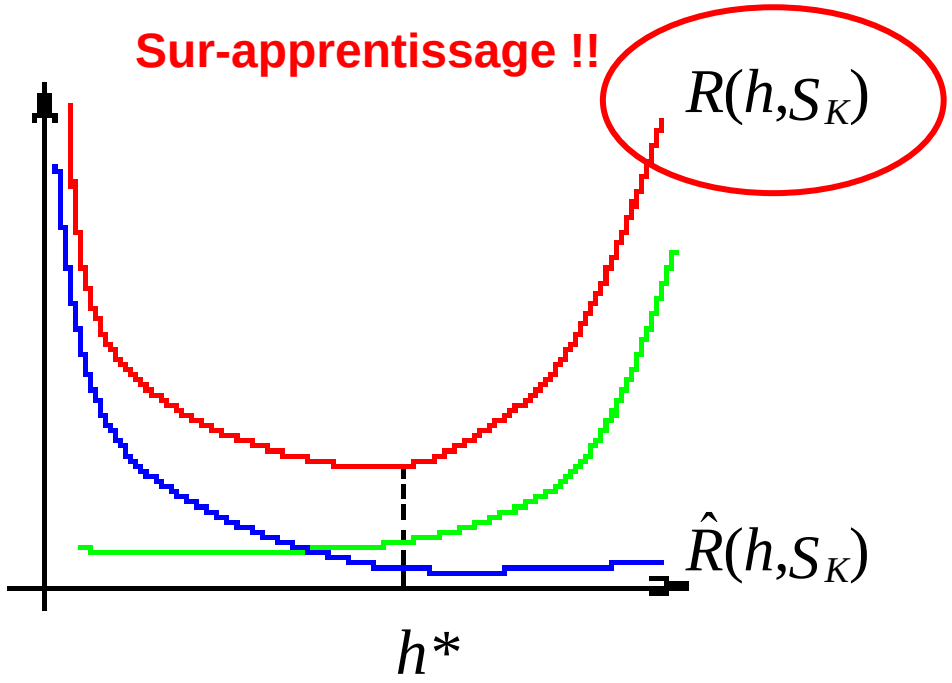
Le problème de la sélection de modèle consiste à choisir, sur la base d'un échantillon d'apprentissage *S* de taille *K*, la classe de fonctions hypothèses *H* et l'hypothèse $h^* \subset H$ de risque réel minimal

Principe

(1) Former des structures emboîtées
$H_1 \subset H_2 \subset \ldots \subset H_N$

(2) Minimiser sur chaque structure

(3) Sélectionner la structure de risque minimum



Ré-échantillonnage et sélection

Pour estimer *R(h*)*, il suffit de disposer d'un échantillon de test indépendant de celui d'apprentissage.
Le risque empirique sur l'ensemble de test est un estimateur du risque réel.

Principe : découpage de l'échantillon en deux parties

- une partie utilisée pour l'apprentissage
- une partie validation pour la sélection

Ces deux parties doivent être **indépendantes** (*hold-out method*) et l'échantillon est de taille limitée

Leave one out

Pour les bases de (très) petites tailles : Comment estimer R(h*) en conservant un échantillon d'apprentissage de taille (presque égale à) K ?

Principe :
Découpage de l'échantillon en deux parties
- N = K-1 couples pour l'apprentissage
- 1 couple pour la sélection

Répéter l'apprentissage N fois en faisant « tourner » la partie inutilisée

Avantage :
Très utilisé lorsque les données sont peu nombreuses

Inconvénient :
estimateur très variable

Leave many out

Même principe, mais au lieu d'écarter un couple, on retire une partie de l'échantillon (N = K/5 ou N = K/10)

Principe :

• découper l'échantillon en P parties de même taille et faire « tourner » la partie inutilisée pour le test
• estimation = moyenne des erreurs mesurées

Avantages :

• estimateur peu biaisé et moins variable que *leave-one-out*
• une variance importante est généralement synonyme d'une inadéquation de la fonction hypothèse h

Pb : quelle hypothèse doit-on finalement utiliser ?

→ Combinaison de modèles