EPSI l'École d'ingénierie informatique

Réseaux de Neurones

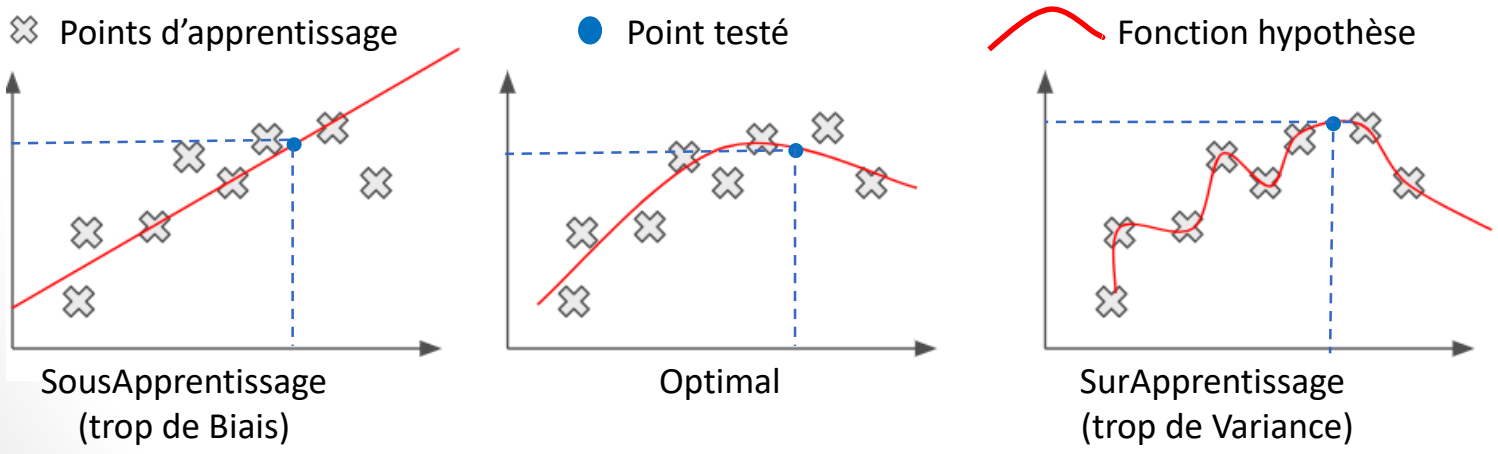
ANN : Artificial Neural Network

Jean-Claude Houbart

IV. LE SURAPPRENTISSAGE

Le surapprentissage

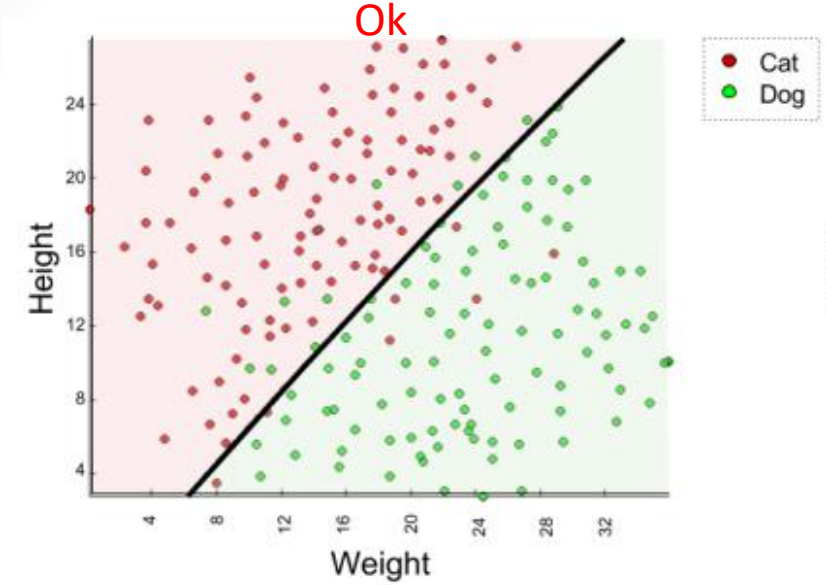
- C'est la principale difficulté à affronter lors du travail d'apprentissage.
- La puissance des réseaux de neurones fait qu'il va coller trop parfaitement au données d'apprentissage, perdant ainsi l'objectif de généralisation.
- Le plus souvent le terme overfitting est utilisé.
- On parle aussi de Dilemme Biais *-Variance



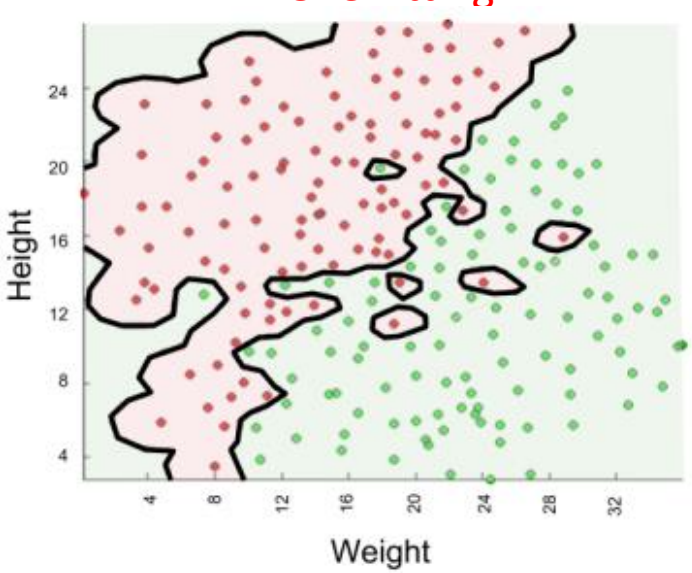
Exemple de surapprentissage

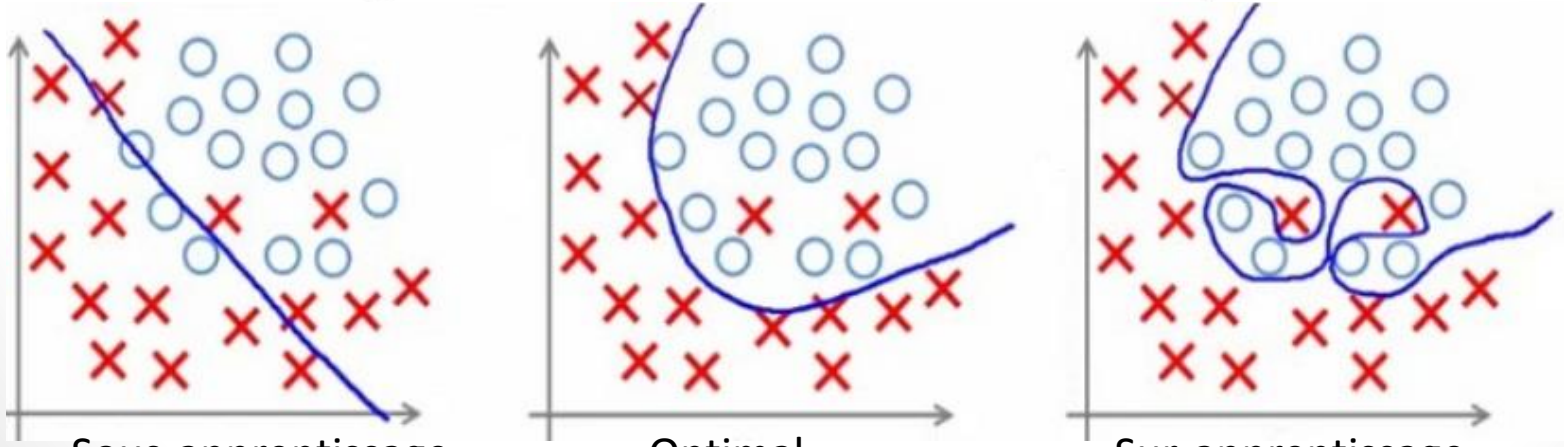
- Problème : Distinguer chat et chien en fonction de la taille et du poids de l'animal.

Ok



Overfitting

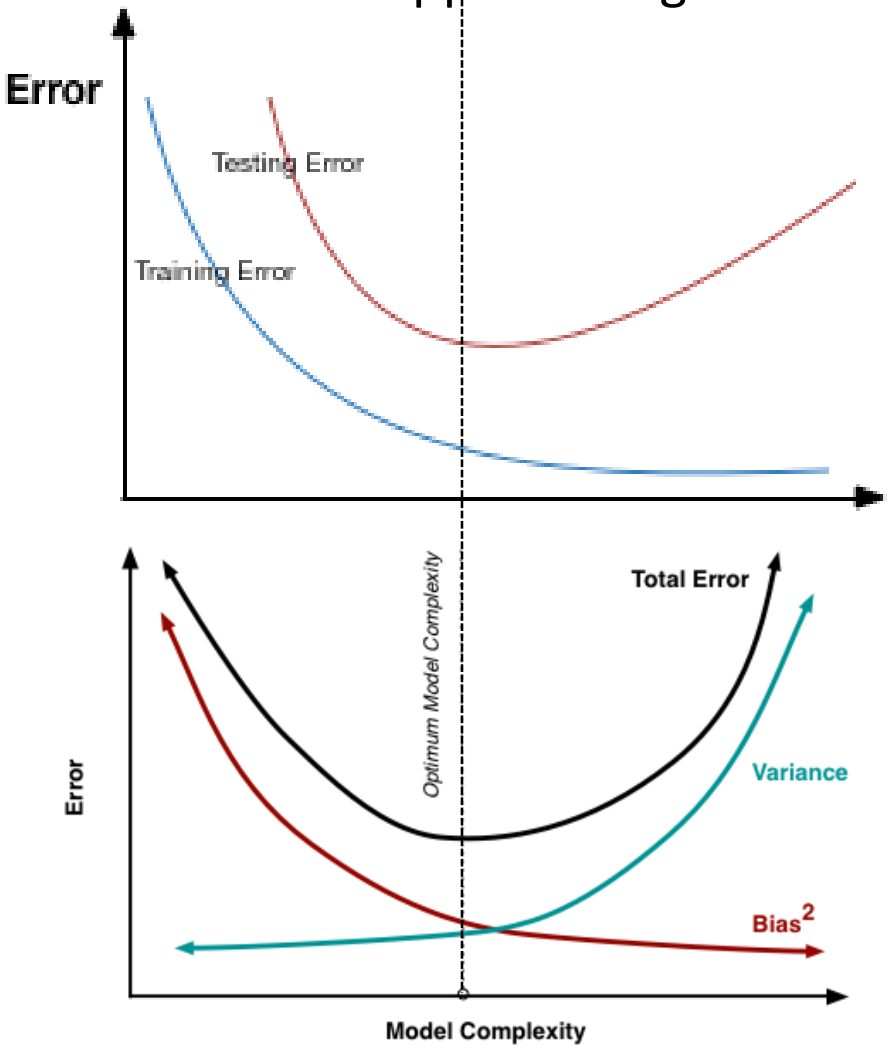




Sous-apprentissage | Optimal | Sur-apprentissage

Identification du surapprentissage

- Pour identifier le bon équilibre biais variance, il faut calculer et tracer les erreurs sur les données d'apprentissage et de de test:



Remèdes au surapprentissage

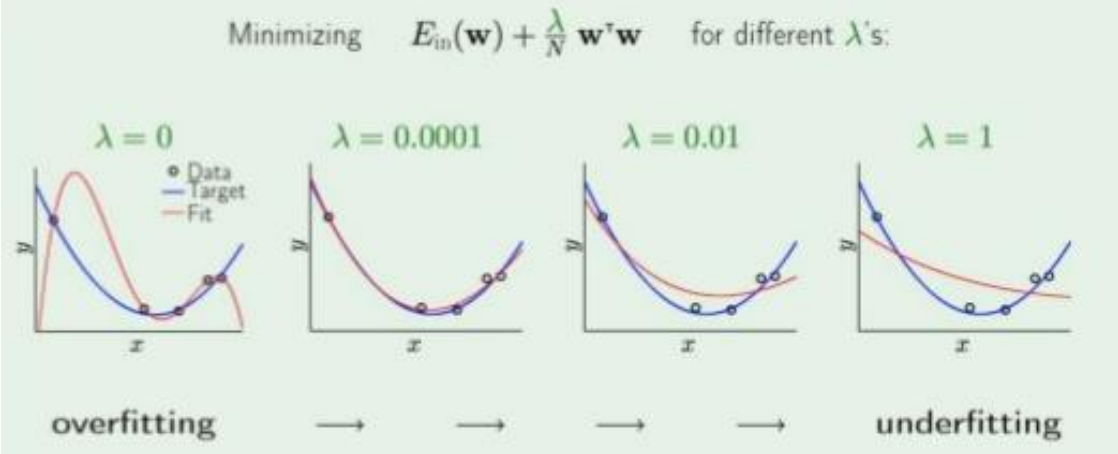
- Solutions « basiques » :
  - Simplifier le modèle ou augmenter le nombre d'exemple.
  - Limiter le nombre d'itération (« Early Stopping »), en utilisant les données de tests.
- Pourtant, en Deep Learning, Le nombre de paramètres peut être très grand (# 10^6) et parfois même dépasser le nombre d'exemple. On recourt alors à deux techniques qui vont permettre de limiter tout de même l'overfitting :
  - La régularisation va permettre de limiter la variation des poids du réseaux (paramètres). La variante la plus utilisée est le Weight Decay.
  - Le Drop out, inventé en 2014, consiste à optimiser chaque itération en ignorant aléatoirement une certaine proportion des nœuds du réseau.
  - Les deux techniques peuvent être utilisés conjointement ou séparément.
- En deep learning, un assez haut niveau d'overfitting, mesuré comme la différence entre erreur sur apprentissage et test, est souvent acceptable et donne les meilleurs résultats.

La régularisation

- La régularisation consiste à modifier la fonction de coût pour obliger le modèle à limiter la variation des poids. La plus connu est la méthode « weight decay », où on ajoute la norme L2 des paramètres:

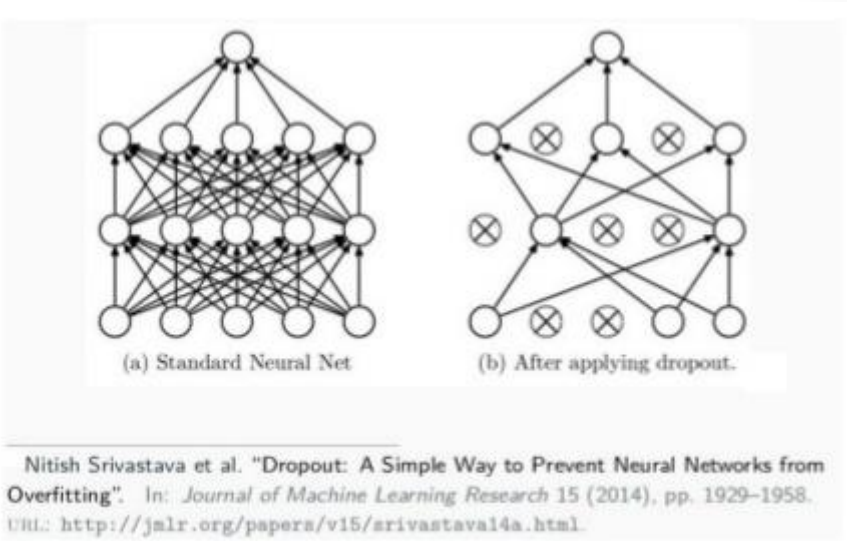
$$J(\theta) = \frac{1}{2m}\left[\sum_{i=0}^{m}(h_\theta(x^{(i)}) - y^i)^2 + \lambda \sum_{i=1}^{n} \theta_j{}^2\right]$$

- λ est le paramètre de régularisation. Par convention, θ_0 n'est pas pris en compte.
- On peut démontrer mathématiquement que l'algorithme de Gradient Descent continue à fonctionner sur cette fonction de coût modifiée.



Le DropOut

- Le drop out consiste à forcer une part aléatoire des poids à 0 lors de chaque itération.

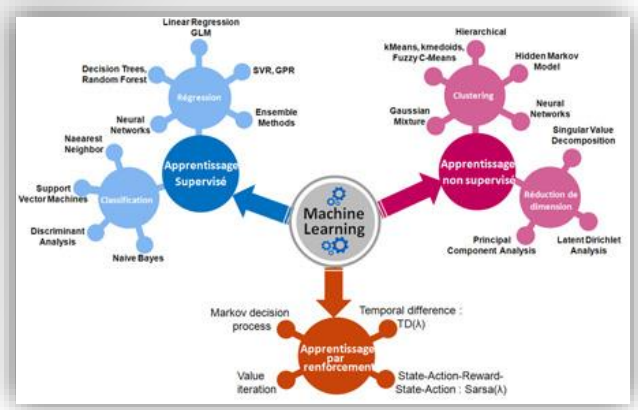


Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

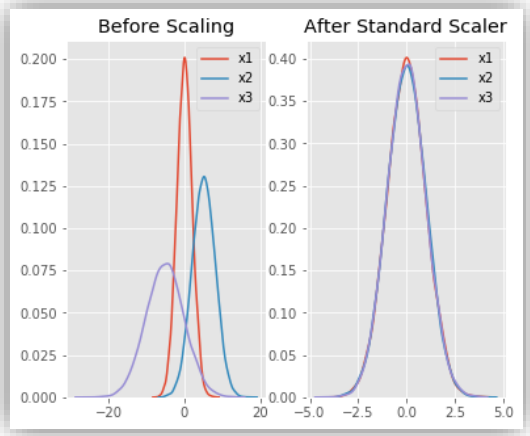
- De cette façon, on rend les poids plus indépendants les un des autres
- On peut voir le drop out comme une façon d'entrainer une multitude de réseau et d'en faire la moyenne.
- La proportion de neurones éteints est généralement appelée « drop_prob ».
- Attention : Il faut désactiver le drop out lors de l'inférence ! Heureusement, Pytorch (et Keras) le fait pour nous, mais il faut bien exécuter la fonction eval() sur le modèle.

V. ÉLÉMENTS DE MÉTHODOLOGIE

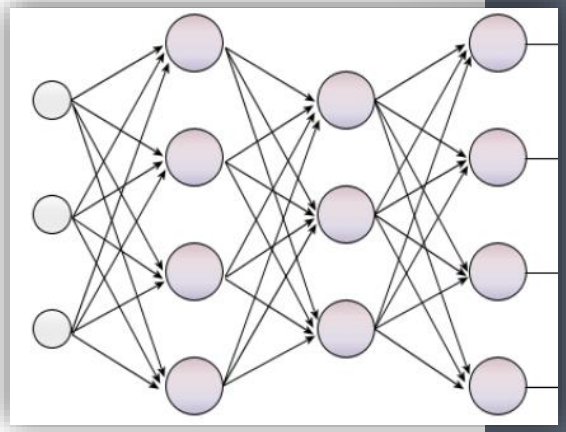
Process du Machine Learning



Sélection et combinaison de modèles



Choix / mise en forme des features



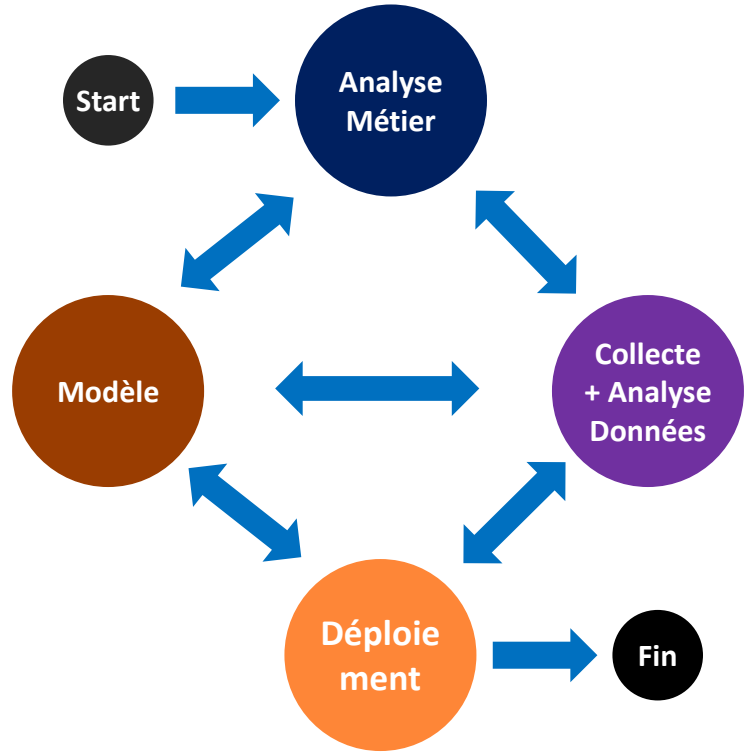
Architecture ; Nombre de couches, fonction de coût…

Choix des hyperparamètres : Learning rate ; paramètre de régularisation ; Drop Prob , Nombre d'itérations…

Mise en production / supervision des modèles.

- Des IA commence à faire ce travail (par exemple AutoML sur Google Cloud Platform)

Cycle de vie Data Science



Sélection des données, normalisation, compléments
PréTraitement

Choix et architecture modèles, Hyperparamétrage, Ensemble
Apprentissage

Tests de permutation, Validation croisée, Valeurs SHap
Evaluation

Stockage
SQL ou non
Indexation
Flux ou batch

Préparation
Structuration
Nettoyage
Enrichissement
Validation

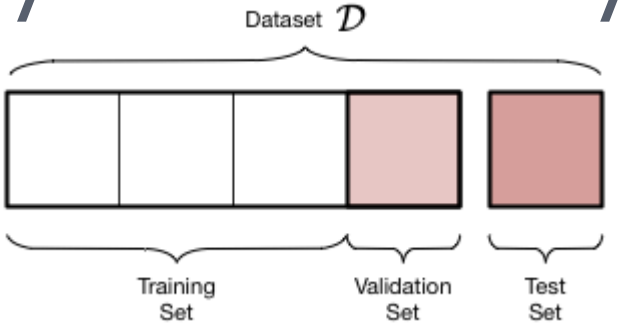
Visualisation
Dataviz
Reporting

Score
Post-traitements
Métriques
Analyse de cas

Performance
Temps de réponses

Supervision
Suivi
Alertes

Training / Validation / Test

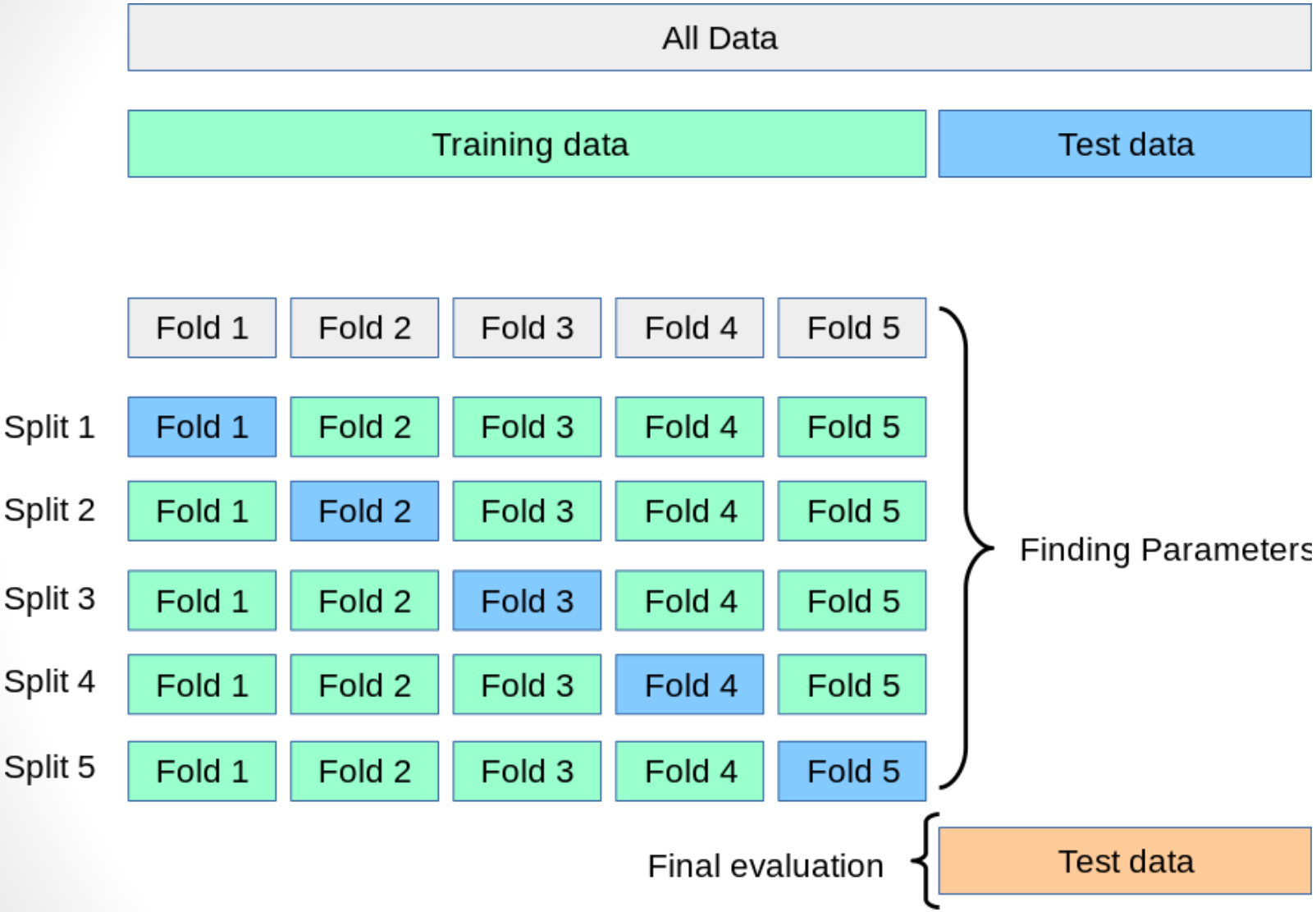


- On a vu que le Data Scientist peut diviser ses données en 3 paquets:
  - Les données d'apprentissage (Training Set) pour entraîner son modèle.
  - Les données de Validation (Validation Set) pour éviter le surapprentissage.
  - Les données de test (Test Set) pour évaluer le score final. Si la différence entre le score sur le Validation Set et le Test Set est trop importante, on a peut-être « sur-appris » sur les donnés de validation.
- Classiquement, les proportions recommandées sont 70/15/15.
- Mais avec des très grands Dataset, on peut arriver à des proportion de type 99,5/0,25/0,25.

Validation Croisée

- Sur des problèmes concrets, Nous sommes souvent confrontés à un volume insuffisant de données.
- Dans ce cas, la division en 3 partie est handicapante.
- On peut alors utiliser une technique qui va permettre d'utiliser au maximum toutes les données, au prix d'une augmentation importante du temps d'apprentissage.
- La validation croisée (« cross validation ») consiste pour chaque jeu d'hyperparamètres testés à :
  - découper les données en N morceaux.
  - Tour à tour, prendre un des morceaux comme jeu de validation apprendre sur le reste.
  - La moyenne des N scores obtenus est alors utilisé pour évaluer le jeu de d'hyperparamètres
- Finalement, l'apprentissage final est réalisé sur toutes les données avec le meilleur jeu d'hyperparamètres.
- Par cette technique, toutes les données auront servi à l'apprentissage et à la validation. Néanmoins, On peut conserver un jeu de test.

Validation Croisée - Résumé



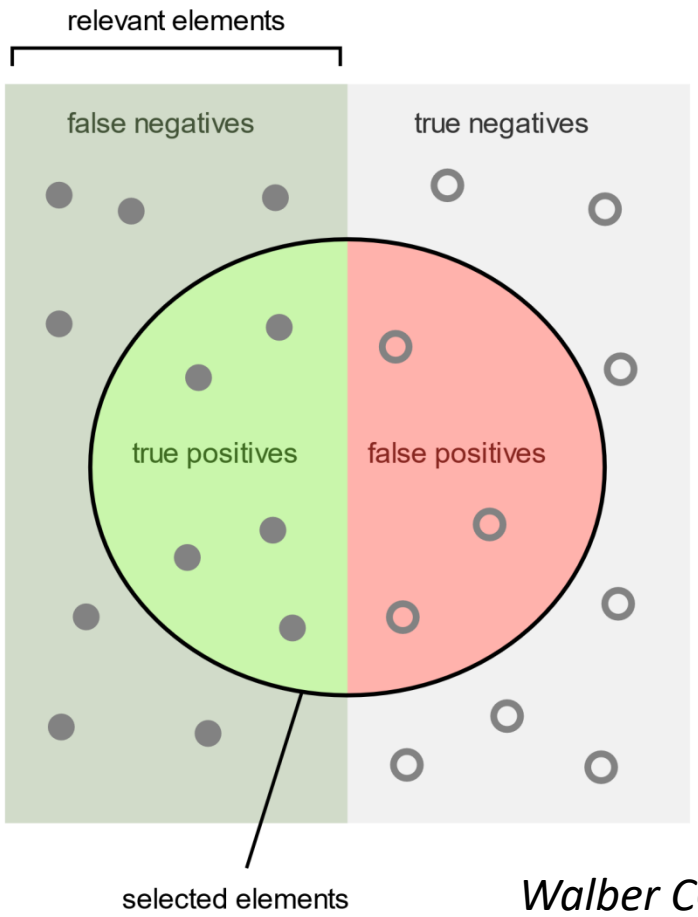
Métriques classe asymétrique

- une classe asymétrique est une classe de résultat ou une valeur est ultra-dominant. Par exemple : 99.5 % de tumeurs non cancéreuses. Par convention y=1 pour la classe "rare". Dans ce cas, prévoir systématiquement la valeur dominante donne le meilleure résultat.

=> On va changer la métrique en se focalisation sur la précision sur les cas rares. On va mesurer les "vrais positifs".

- Precision : Vrai positif /(Vrai positifs + Faux positifs). On mesure la proportion correcte dans les positifs donnés par le modèle.
- Recall : Vrai positif /(Vrai positif + Faux négatifs). On mesure la proportion de positifs trouvé par le modèle.

Précision et Recall



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{selected elements}}$$

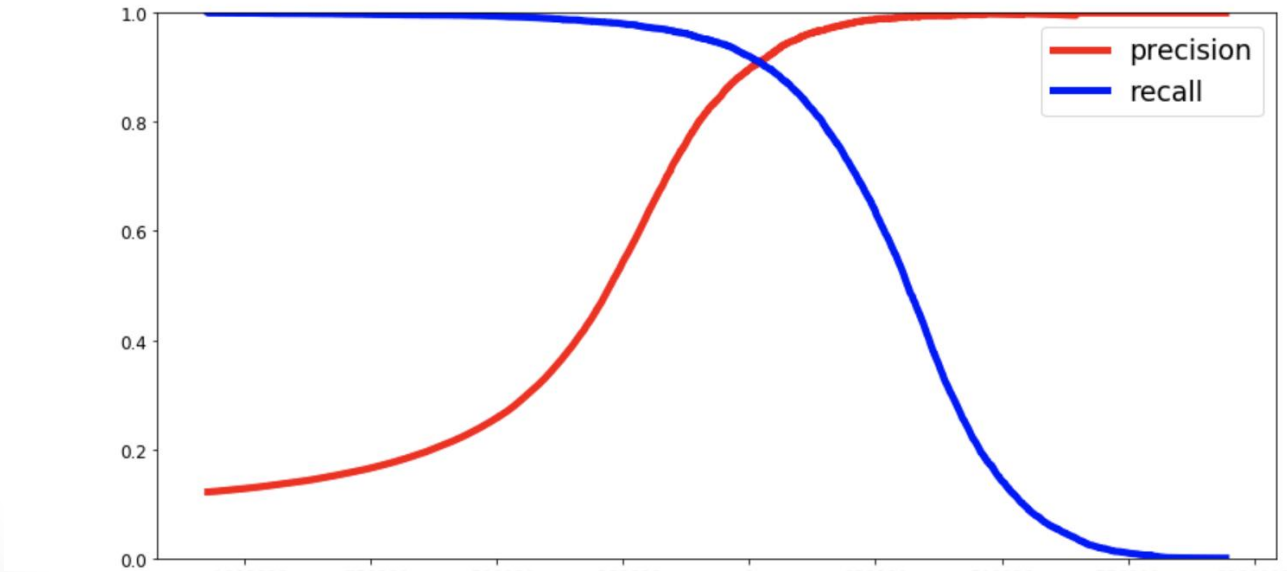
How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

Walber CC BY-SA 4.0, commons.wikimedia.org

Métriques classe asymétrique

- Si on augmente le seuil de h(x) pour mettre la valeur à 1, par exemple h(x)>0.7 au lieu de 0.5, on va améliorer la précision, mais empirer le recall. Inversement si on abaisse le seuil en dessous de 0.5.
- On peut tracer une courbe recall/precision pour les différentes valeurs de seuil.



F1 Score

- Comment revenir à un seul métrique ? La moyenne n'est pas une solution par exemple si précision =0.02 et recall = 1.
- On va utiliser le F (ou F1) Score :

$$F_1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision + Recall}}$$

- Valeur entre 0 et 1. Optimale pour 1.

Matrice et table de confusion

- La matrice de confusion permet d'identifier les cas mal gérés par le modèle.

Prévision

Réalité	Pluie	Beau Temps	Neige
Pluie	31	1	9
Beau Temps	6	23	8
Neige	5	6	32

- La table de confusion permet d'analyser les résultats par classe.

Réalité

Prévision	Pluie	NonPluie
Test Positif	31	11
Test Negative	10	69

	Neige	Non(Neige)
Test Positif	32	17
Test Negative	11	61

- Calculer Précision, Recall et F1-Score pour ces deux classes.

Pluie : P = 73,8% ; R = 75,6% ; F1=74,7%
Neige : P = 64,6% ; R=73,8% ; F1=68,9%

VI. QUELQUES PROCÉDÉS TECHNIQUES

Minibatch et SGC

- On a vu qu'on cherchait à chaque itération à minimiser la fonction de coût, par exemple :

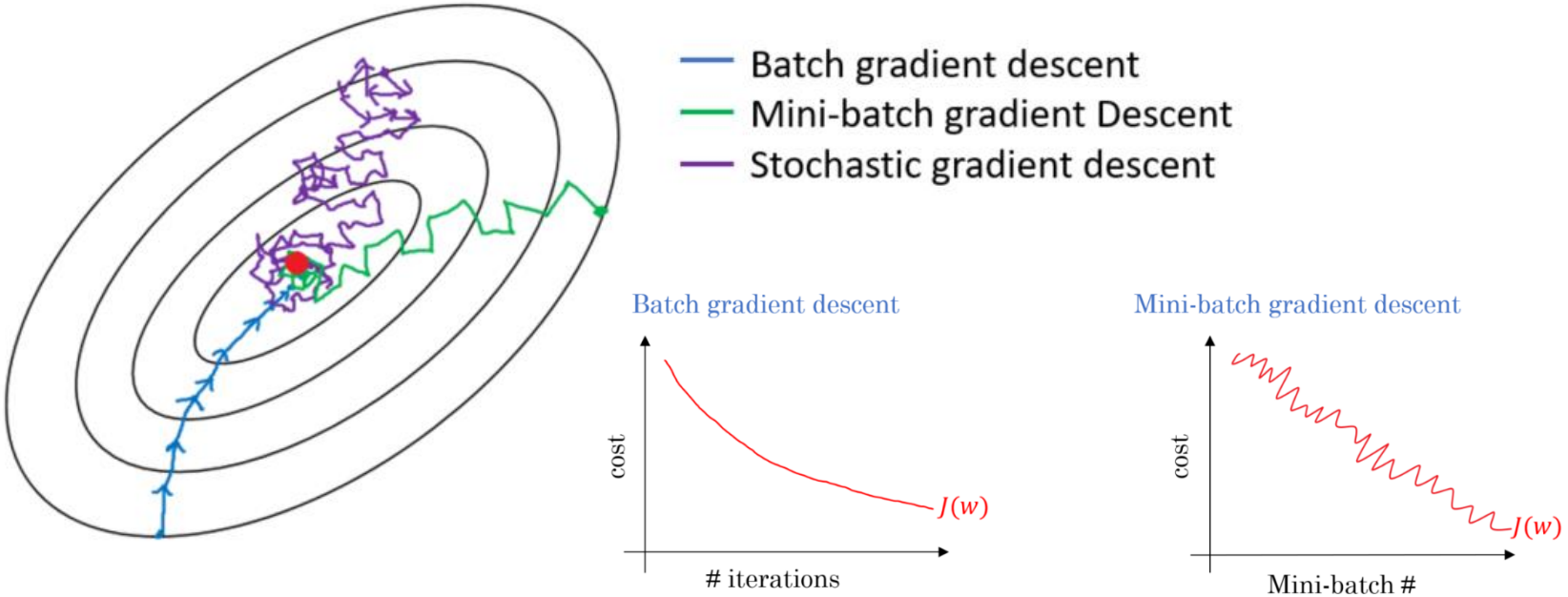
$$J(\theta_1, \dots, \theta_k) = \sum (dj - y_j)^2$$

… la somme des carrés des différences entre les valeurs obtenus et celles attendues, pour tous les exemples.

- Dans la pratique, le temps de calcul est rédhibitoire quand le nombre d'exemples est grand.
- Solution : On tire aléatoirement à chaque itération un nombre réduit d'exemples (quelques centaines typiquement, et une puissance de 2, par exemple 256), le « mini-lot » ou minibatch, pour lequel on calcule l'optimisation.
- On compense cette imprécision en réalisant un grand nombre d'itérations.
- Si on utilise un seul exemple à la fois (taille du lot = 1), on parle de Stochastic Gradient Descent (SGC).
- En pratique, SGC est souvent utilisé pour désigner le minibatch aussi.

Différences de Convergence

- En minibatch ou SGC, nous ne sommes pas sûr d'avancer vers l'optimum à chaque itération :



- Pour accélérer la convergence, on peut utiliser le momentum.

Cas général : $\theta=\theta-\eta\cdot\nabla_\theta J(\theta)$. H est le learning rate.

Avec Momentum : $v_t=\Upsilon v_{t-1}+\eta\nabla_\theta J(\theta)$. ($\nabla_\theta$ opérateur différentiel sur les poids)

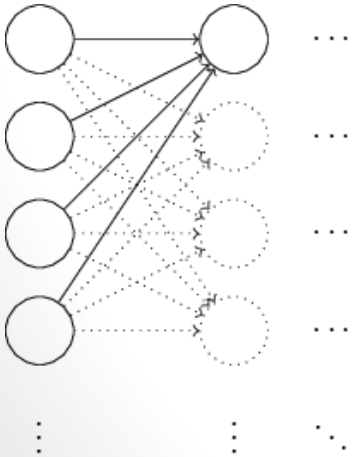
$\theta=\theta-v_t$

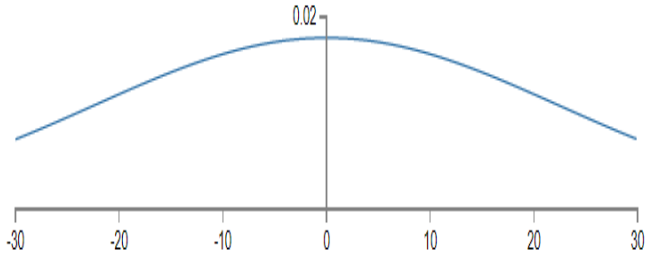
- On ajoute une proportion Υ de la variation précédentes. Revient à donner « de l'élan » à notre trajectoire.
- La méthode Nesterov Advanced Gradient (NAG) améliore encore cette idée.

Initialisation des poids

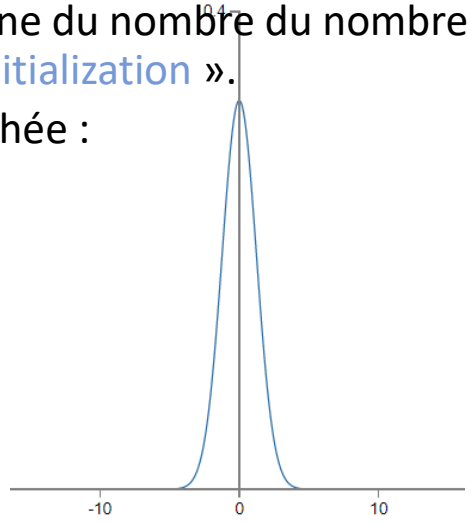
- Question : comment initialiser les poids au départ de la première itération.
- Problème important qui a été un des progrès menant au deep learning. La rapidité de convergence en dépend.
- A l'origine, les poids étaient initialisé à 0. Du temps était perdu pour « activer » les neurones.
- Ensuite, les poids étaient initialisés aléatoirement entre -1 et 1 (en fait avec une moyenne à 0, et un écart type de 1). C'est correct pour des petits réseaux. Mais des grands réseaux, on aboutit à des valeurs d'entrées, trop importantes pour la première couche cachée. Toutes sorties des neurones sont très proches de 0 ou 1. On dit qu'ils sont saturées.
- Pour éviter cela, on va diviser les valeurs aléatoires par la racine du nombre du nombre de neurones d'entrée. On appelle cette pratique « he-et-al initialization ».

Distribution de la valeur d'entrée d'un neurone de la couche cachée :





Initialisation aléatoire écart type 1



Initialisation aléatoire écart type $\frac{1}{\sqrt{N_{in}}}$

Décroissance du learning rate

- Le learning rate permet de faire varier les poids plus ou moins vite, lors de la descente de gradient.
- Pour les premières itérations, on a intérêt à faire des « grands pas » vers l'optimal, puis des « petits pas » jusqu'à l'optimal. Pensez au Golf.
- Pour cela on peut faire décroître le Learning Rate au fil des itérations.