Pré-traitement

Lorsque les variables ont des plages de valeurs très différentes (par exemple VAR1 $\in [-2, 4]$ et VAR2 $\in [0.7, 1.3]$), la normalisation devient cruciale pour éviter qu'une variable ne domine excessivement l'analyse, en rééquilibrant leur influence relative.

Normalisation [0;1]

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Normalisation [−1;1]

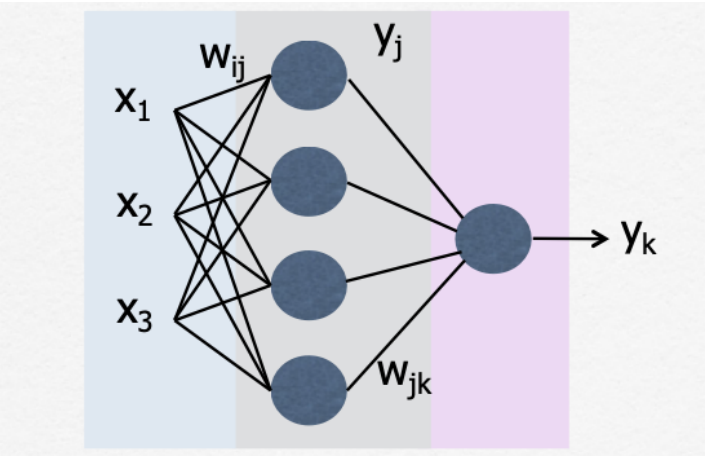
$$x' = 2\frac{x - x_{\min}}{(x_{\max} - x_{\min})} - 1$$

Cependant MinMax peut rencontrer des problèmes pour normaliser s'il y a des valeurs aberrantes. nous devons donc en premier lieu filtrer les valeurs abérantes avec la formule de normalisation z (z-score) est :

$$x' = \frac{x - \mu}{\sigma}$$

x est la valeur originale, **μ** (mu) est la moyenne des données, **σ** (sigma) est l'écart-type

Perceptron



- **Gauche**: nous avons les 3 **entrées** x_1, x_2 et x_3.
- **Milieu**: nous avons la **couche cachée** avec ses neurones chaqu'un ayant des entrées avec un poid w ainsi qu'un biais b.
- **Droite**: nous avons la **sortie** y qui est le résultat de la somme des entrées multipliées par leurs poids respectifs plus le biais b.

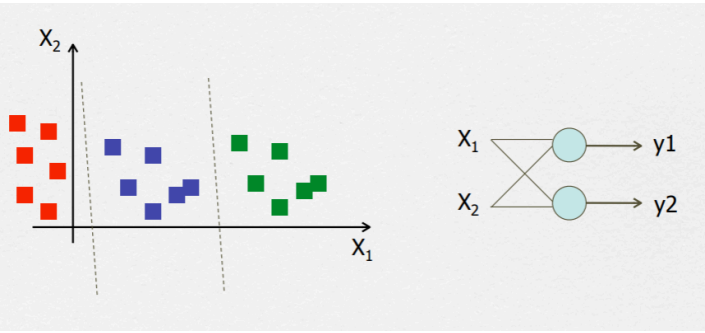
Total des poids

On somme:

- Poids d'entrée: $\text{nb entrée} * \text{nb neurones}$
- Poids de sortie: $\text{nb sortie} * \text{nb neurones}$
- **Ajouter le biais**

Strucutre lors de plusieurs classes

Le nombre d'entrées reflète la complexité du problème (2D x,y = 2 entrées). Le nombre de neurones cachés s'adapte à la séparabilité linéaire, nécessitant plus de neurones ou de couches pour les problèmes non linéairement séparables. Le nombre de sorties correspond aux classes à prédire.



Formule d'un neurone

$$y = f(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

Backpropagation

1. Initialiser aléatoirement les poids.

- Générer des valeurs aléatoires pour les matrices de poids
- Plage typique : intervalle $[-0.5, 0.5]$

2. Calculer les sorties yk **pour un vecteur d'entrée donné** X**.**

$$y_k = f\left(\sum_j w_{\text{jk}} y_j\right)$$

$$y_j = f\left(\sum_i w_{\text{ij}} x_i\right)$$

$$f(s) = \frac{1}{1 + e^{-s}}$$

3. Pour chaque neurone de sortie, calculer :

$$\delta_k = (y_k - t_k) f'(y_j)$$

$$f'(y_j) = y_{k(1-y_k)}$$

est la dérivée de la fonction sigmoïde, et t_k est la sortie désirée du neurone de sortie k

4. Pour chaque neurone de la couche cachée, calculer :

$$\delta_j = \sum_k w_{\text{jk}} f'(x_i) \delta_k$$

$$f'(x_i) = y_{j(1-y_j)}$$

5. Mettre à jour les poids du réseau comme suit :

$$w_{\text{jk}}(t+1) = w_{\text{jk}}(t) - \eta \delta_k y_j$$

$$w_{\text{ij}}(t+1) = w_{\text{ij}}(t) - \eta \delta_j x_i$$

où η est le taux d'apprentissage, $0 < \eta < 1$

6. Répéter les étapes 2 à 5 pendant un nombre donné d'itérations ou jusqu'à ce que l'erreur soit inférieure à un seuil donné.

Fonction d'activation

Sigmoïde

La fonction sigmoide est bornée entre $[0; 1]$

Tanh

La fonction tangeante hyperbolique est bornée entre $[-1; 1]$

ReLU

La fonction ReLU est bornée entre $[0; +\infty]$

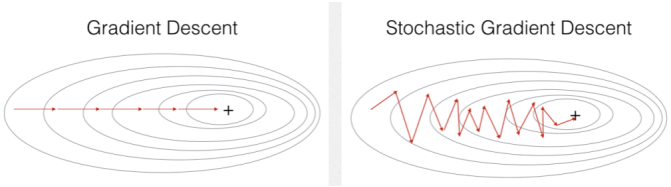
> **Note**
> Si plus de deux classes alors il faut utiliser la fonction de sortie **softmax**!

Concept de Momentum

Le momentum est une propriété physique qui permet à un objet ayant une masse de continuer sa trajectoire même lorsqu'une force externe opposée est appliquée.
Dans le contexte des réseaux de neurones, l'idée derrière ce "truc" est d'ajouter un terme de momentum à l'adaptation des poids. Cela permet d'accélérer la convergence du modèle et d'éviter les oscillations lorsque le gradient fluctue de manière importante d'une itération à l'autre.

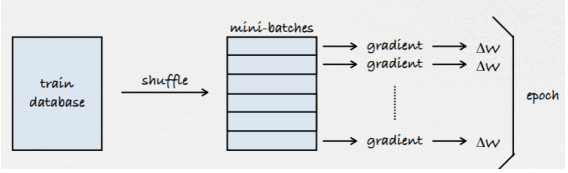
Variantes du calcul du gradient

1. **Descente de gradient par lot** : Le gradient est calculé à partir de l'ensemble de données complet, offrant des mises à jour plus précises mais plus lentes.
2. **Descente de gradient stochastique** : Le gradient est calculé à partir d'un seul échantillon, ce qui accélère les mises à jour mais peut rendre la convergence moins stable.



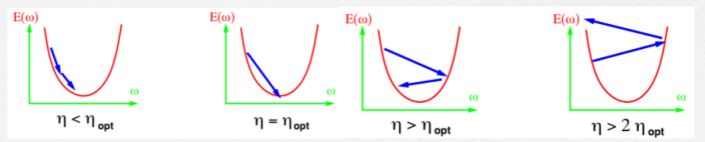
Descente de gradient par mini-batch

La descente de gradient par mini-batch utilise un sous-ensemble aléatoire des données (mini-batch) pour calculer une approximation stochastique du gradient exact. Cela réduit la charge computationnelle et accélère les itérations, mais avec un taux de convergence plus faible comparé à la descente de gradient par batch.



Méthodes de descente de gradient

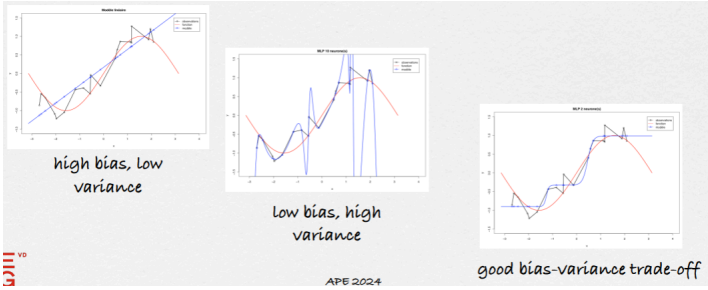
1. **SGD** : Mise à jour des poids après chaque échantillon ou mini-lot, accélère l'apprentissage mais avec une convergence moins stable.
2. **RMSprop** : Ajuste le taux d'apprentissage en fonction des gradients passés, stabilise les mises à jour pour de meilleures performances.
3. **Adam** : Combine SGD et moment, adapte les taux d'apprentissage pour chaque paramètre, accélère la convergence et améliore l'efficacité.
4. Learning Rate :



Compromis biais-variance

- **Biais** : Erreur systématique du modèle, souvent due à un modèle trop simple.
- **Variance** : Sensibilité du modèle aux variations des données d'entraînement, souvent liée à un modèle trop complexe.

L'objectif est de trouver un équilibre entre biais et variance pour minimiser l'erreur globale et améliorer la généralisation.



Techniques pour éviter le surapprentissage :

- **Early stopping**: Cette technique consiste à arrêter l'entraînement du modèle dès que la performance sur un ensemble de validation cesse de s'améliorer. Cela permet d'éviter que le modèle apprenne des détails non pertinents des données d'entraînement.
- **Régularisation** : L'ajout d'une pénalité sur la complexité du modèle (comme la régularisation L1 ou L2) aide à limiter les valeurs des poids, réduisant ainsi le risque de surajustement et améliorant la généralisation.
- **Augmentation des données** : En générant artificiellement plus de données d'entraînement par des transformations telles que la rotation, le redimensionnement ou le bruit, on permet au modèle de mieux généraliser en apprenant à partir d'une plus grande variété d'exemples.