Techniques d'Apprentissage Artificiel

Apprentissage par minimisation de l'erreur

Introduction

L'apprentissage par minimisation de l'erreur est un type d'apprentissage qui permet à un réseaux de neurones (RNN) d'être performant dans la réalisation d'une tâche bien précise en minimisant l'erreur entre sa sortie et la valeur désirée.

Fonction objective

- Concrètement, l'apprentissage consiste à mettre à jour un ensemble de paramètres (poids + bias) de sorte à minimiser l'erreur entre la sortie du modèle et la sortie désirée (cible).
- Une fonction dite* **fonction de perte / fonction coût / fonction erreur / fonction objective,** est utilisée pour mesurer la performance d'un modèle (bon ou mauvais).

- * Loss function= cost function = error function = objective function

Fonction objective

- Une faible valeur de la fonction de perte indique une bonne performance alors qu'une haute valeur indique une mauvaise performance

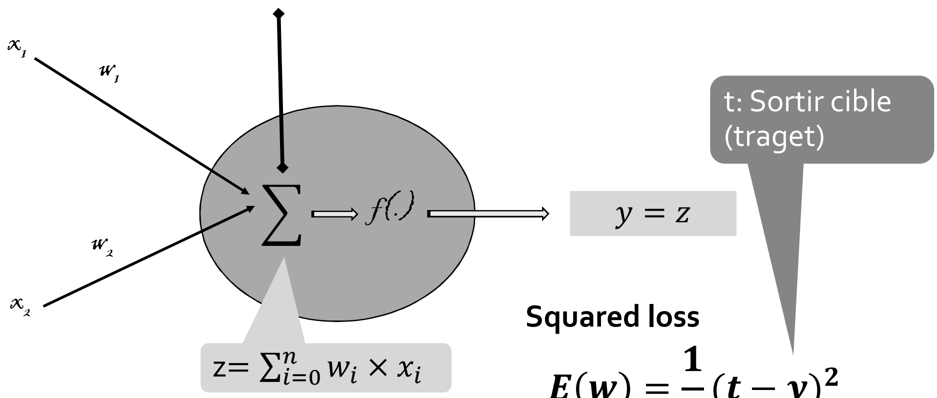
Fonction objective pour la classification

- Cross entropy/ Log loss
- Hinge loss
- Focal loss
- …

Fonction objective pour la régression

- Mean squared error (MSE) / quadratic loss /L2 loss
- Squared loss
- Sum Squared Error (SSE)
- …

Fonction objective:
Exemple de régression linéaire



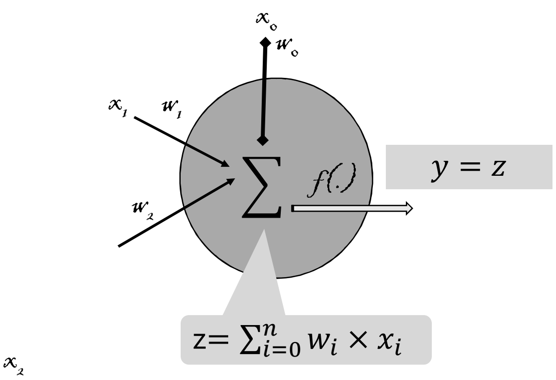
Squared loss

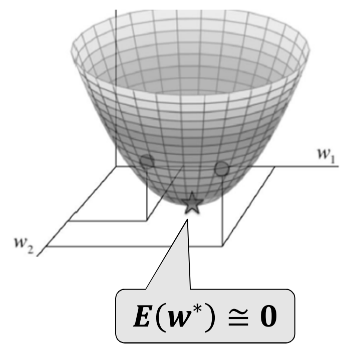
$$\boldsymbol{E(w) = \frac{1}{2}(t - y)^2}$$

$$E(w\ ) \cong 0$$

Fonction objective:
Exemple de régression linéaire

$$\boldsymbol{E(w) = \frac{1}{2}(t - y)^2}$$





Problème d'optimisation

$E(w^*) \cong 0 \Rightarrow w^* \ telque \ E(w^*) < E(w) \ \forall \ w \neq w^*$

- déterminer la valeurs des paramètres (poids + seuils) qui minimise la fonction de perte.

Problème d'optimisation

$E(w^*) \cong 0 \Rightarrow w^* \ telque \ E(w^*) < E(w) \ \forall \ w \neq w^*$

$\boldsymbol{w^* = argmin \, E(w)}$

Problème d'optimisation

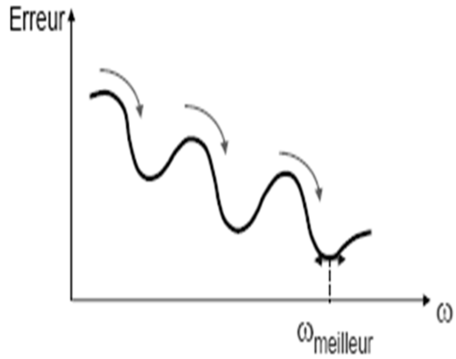
Solution :Descente du gradient

Descente du gradient

Gradient=dérivée partielle

$$df(x) = \frac{\partial f}{\partial x}$$

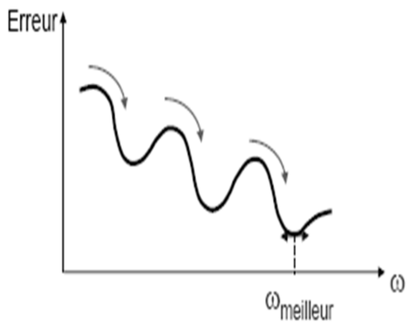
Permet de descendre pas à pas vers la valeur minimale



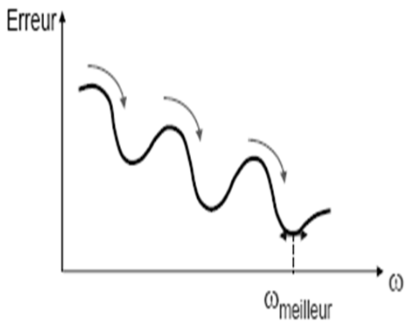
Descente du gradient

Ajuster les paramètres dans la direction du gradient pour descendre pas à pas vers les valeurs optimales qui minimise la fonction de perte

$$W^t = W^{t-1} + \Delta W$$



Descente du gradient



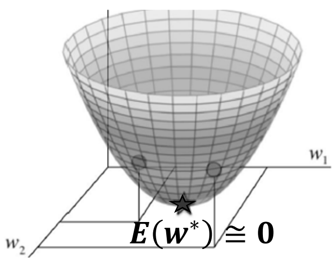
$$\Delta W = Wt - W^{t-1} = df(x) = \frac{\partial f}{\partial x}$$

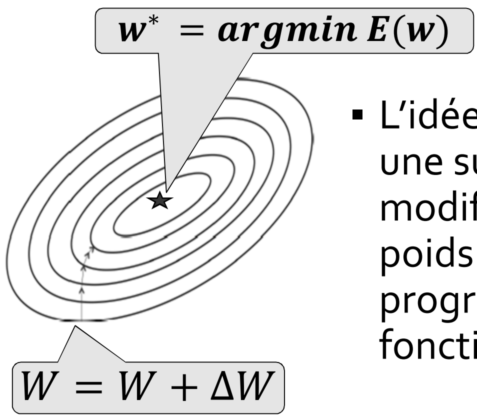
$$\frac{\partial f}{\partial x} > 0 \leftrightarrow f(x) \nearrow si\ x \nearrow \quad donc\ on\ doit\ x \searrow$$

$$\frac{\partial f}{\partial x} < 0 \leftrightarrow f(x) \searrow si\ x \nearrow \quad donc\ on\ doit\ x \nearrow$$

$$\frac{\partial f}{\partial x} = 0 \leftrightarrow f(x) est\ optimal\ donc\ \ x\ constant$$

Descente du gradient





- L'idée est d'appliquer une suite de petites modifications sur les poids de sorte à réduire progressivement la fonction de perte E.

Descente du gradient

$$\left.\begin{array}{l} \dfrac{\partial f}{\partial x} > 0 \leftrightarrow f(x) \nearrow si\ x \nearrow \quad donc\ on\ doit\ x \searrow \\[2ex] \dfrac{\partial f}{\partial x} < 0 \leftrightarrow f(x) \searrow si\ x \nearrow \quad donc\ on\ doit\ x \nearrow \\[2ex] \dfrac{\partial f}{\partial x} = 0 \leftrightarrow f(x) est\ optimal\ donc\ \ x\ constant \end{array}\right\} \Delta W = -\eta \nabla E(W)$$

$$\nabla E(W) = \frac{\partial E}{\partial w_i}$$

$$W = W + \Delta W$$

Descente du gradient

$$W = W + \Delta W$$

$$\Delta W = -\eta \nabla E(W)$$

$$\nabla E(W) = \frac{\partial E}{\partial w_i}$$

Règle d'apprentissage Delta
(Delta learning rule)

Widrow & Hoff 1960

Descente du gradient

$$ {}^{t}w^{c}_{i,j} = {}^{t-1}w^{c}_{i,j} + \Delta w^{c}_{i,j} $$

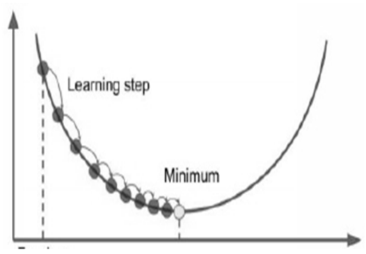
$$ {}^{t}w^{c}_{i,j} = {}^{t-1}w^{c}_{i,j} \overbrace{+ \; -\boxed{\eta} \times \frac{\partial \boxed{E}}{\partial w_{i,j}}}^{\Delta w^{c}_{i,j}} $$

Pas d'apprentissage (Learning rate)

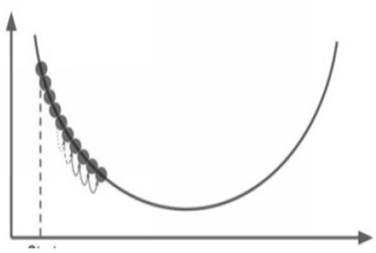
Erreur

Règle d'apprentissage Delta
(Delta learning rule)

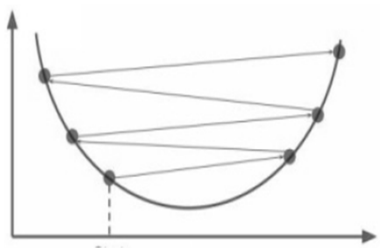
Descente du gradient



Le pas d'apprentissage permet de faire un saut vers le prochain point où on doit calculer la dérivée.



Pas d'apprentissage faible ➔ temps de convergence élevé



Pas d'apprentissage élevé ➔ Risque de non convergence

Descente du gradient

$$ {}^{t}w_{i,j}^{c} = {}^{t-1}w_{i,j}^{c} + \Delta w_{i,j}^{c} $$

$$ {}^{t}w_{i,j}^{c} = {}^{t-1}w_{i,j}^{c} + \ -\eta \times \boxed{\frac{\partial E}{\partial w_{i,j}}} $$

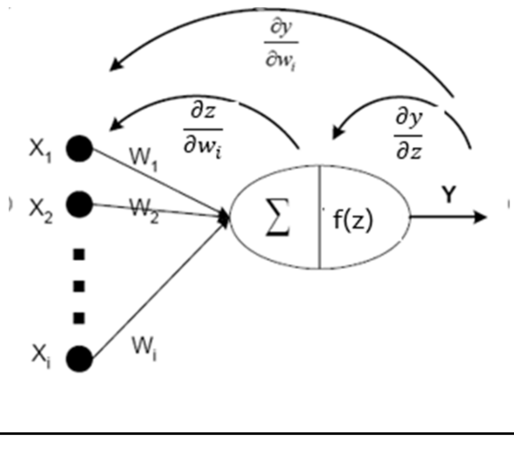
Comment le calculer

Rétro-propagation

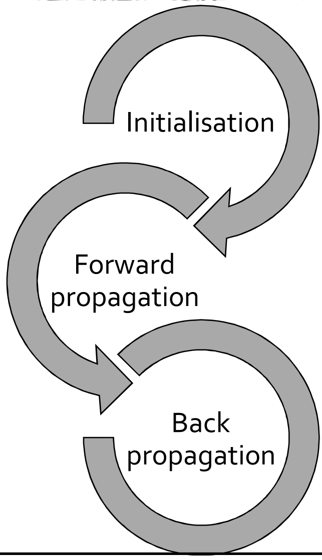
Backpropagation = *backward propagation of errors*

Solution possible grâce
à la dérivée en chaine

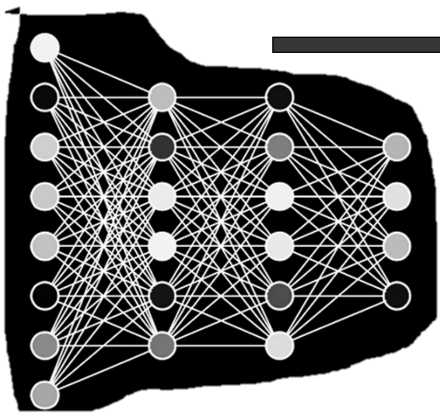
$$ \frac{\partial f(x)}{\partial x} = \frac{\partial g(h(x))}{\partial h(x)} \cdot \frac{\partial h(x)}{\partial x} $$



Rétro-propagation



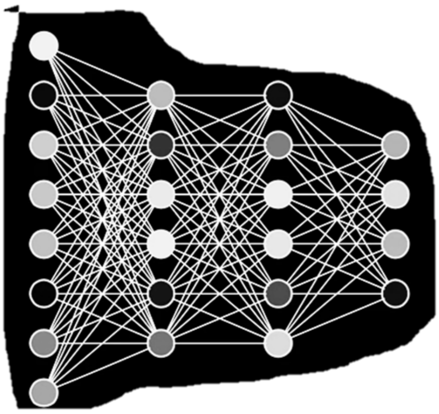
Problèmes non linéaires : Notion de couches



$Y^k = f^k(W^k * Y^{k-1} + b^k)$

Problèmes non linéaires : Notion de couches

$$^{t}w_{i,j}^{c} = {}^{t-1}w_{i,j}^{c} + \Delta w_{i,j}^{c}$$



Rétro-propagation

- Nombre de neurones de la couche d'entrée (nbr-in)
- Nombre de neurones de la couche cachée (nbr-c)
- Le nombre de neurones de la couche de sortie (nbr-out)
- Pas d'apprentissage (learning rate) (η)
- Le nombre d'itérations (epoch)
- Erreur de tolérance (seuil pour l'arrêt de l'apprentissage)

Rétro-propagation

$$\Delta w_{i,k}^{c} = \eta \times e_k^c \times y_i^{c-1}$$

$$e_k^c = f_k^{c\,\prime}(y_k^c) \times e_k^{c+1}$$

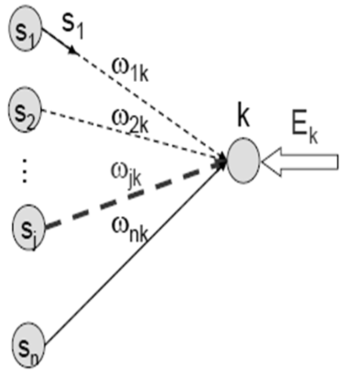
–η: pas d'apprentissage

–y_i^{c-1} : sortie du neurone i, en entrée de k, la correction est proportion. à son importance

–e_k^{c+1} : erreur pondérée du neurone k

–$f_k^{c\,\prime}(y_k^c)$: dérivée de la fonction du neurone

–e_k^c: erreur commise par le neurone k, la correction sera proportionnelle à son importance



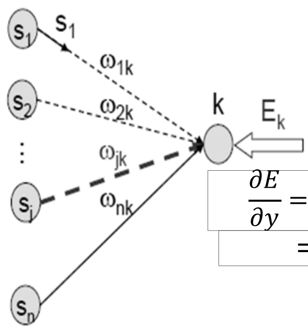
Rétro-propagation

Couche de sortie

Pour la couche de sortie: on sait exactement ce que l'on veut obtenir en sortie donc on peut corriger en conséquence

$$e_k^s = f_k^{s\,\prime}(y_k^s) \times (t_k - y_k^s)$$

$$\Delta w_{i,k}^{s} = \eta \times e_k^s \times y_i^{s-1}$$



$$E(w) = \frac{1}{2}(t-y)^2$$

$$\frac{\partial E}{\partial y} = 2 \times \frac{1}{2} \times (t-y) \times f'(y)$$

$$= (t-y) \times f'(y)$$

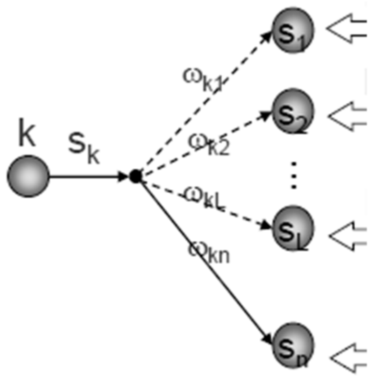
Rétro-propagation

Couche cachée

Problème : car pas de contact direct avec la solution (sortie)
•Solution: On fait une estimation de l'erreur effectuée pour chaque neurone de la couche cachée

$$e_k^c = {f_k^c}'(y_k^c) \times \sum_{j=1}^{n^{c+1}} w_{k,j}^{c+1} \times e_j^{c+1}$$

$$\Delta w_{i,k}^c = \eta \times e_k^c \times y_i^{c-1}$$



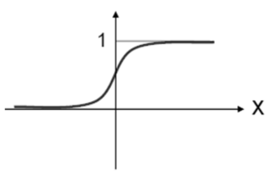
Exercice d'application

- Soit un réseau de neurones 3-3-1; ayant la base de données du tableau1 et les poids initiaux du tableau 2.
- Sachant que toutes les fonctions d'activation sont sigmoïde et que la fonction de perte est squared loss appliquer la rétro-propagation pour une epoch.

N	x1	x2	x3	t
1	1	0	1	0
2	0	0	1	1
3	0	1	0	1
4	1	1	1	1
5	1	0	0	0
6	1	1	1	1
7	0	1	1	0
8	1	1	0	0

$w_{i;j}^1$	1	2	3
0	0.1	0.1	0.1
1	0.2	0.35	0.1
2	0.3	0.27	0.25
3	0.1	0.4	0.35

$w_{i;1}^2$	1
0	0.1
1	0.1
2	0.2
3	0.25



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

$$f'(x) = f(x).(1-f(x))$$