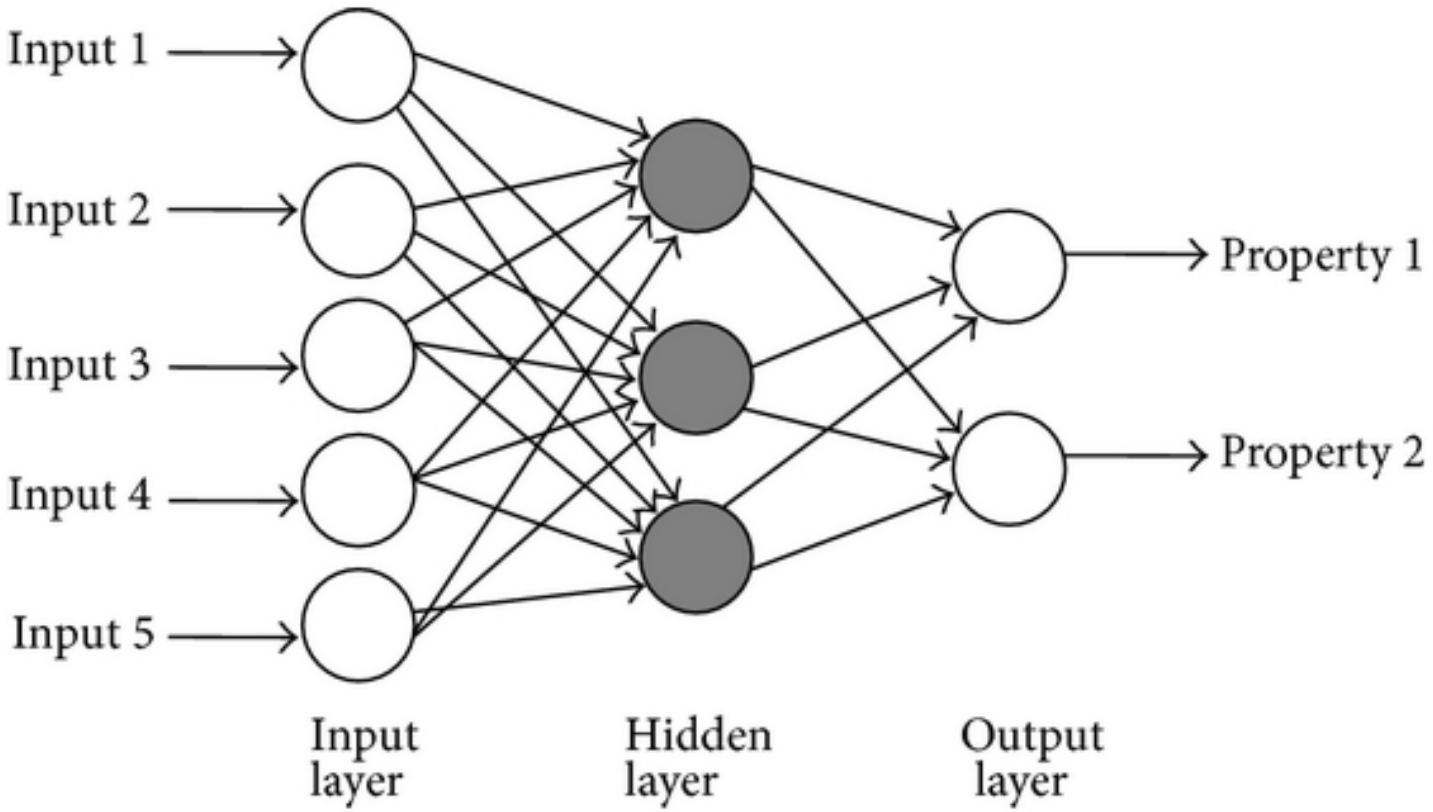
Réseaux de Neurones

Stéphane Ayache, Cécile Capponi, François Denis, Hachem Kadri
cecile.capponi@lif.univ-mrs.fr

Université Aix-Marseille

M1 Informatique

Plan


Input 1
Input 2
Input 3
Input 4
Input 5
Property 1
Property 2
Input layer
Hidden layer
Output layer

Du perceptron à l'apprentissage profond

1957 (Rosenblatt) Perceptron

1960 (Widrow, Hoff) ADALINE

1969 (Minsky, Papert) Problème XOR

1986 (Rumelhart *et.* al) MLP et backpropagation

1992 (Vapnik *et.* al) SVM

1998 (LeCun *et.* al) LeNet

2010 (Hinton *et.* al) Deep Neural Networks

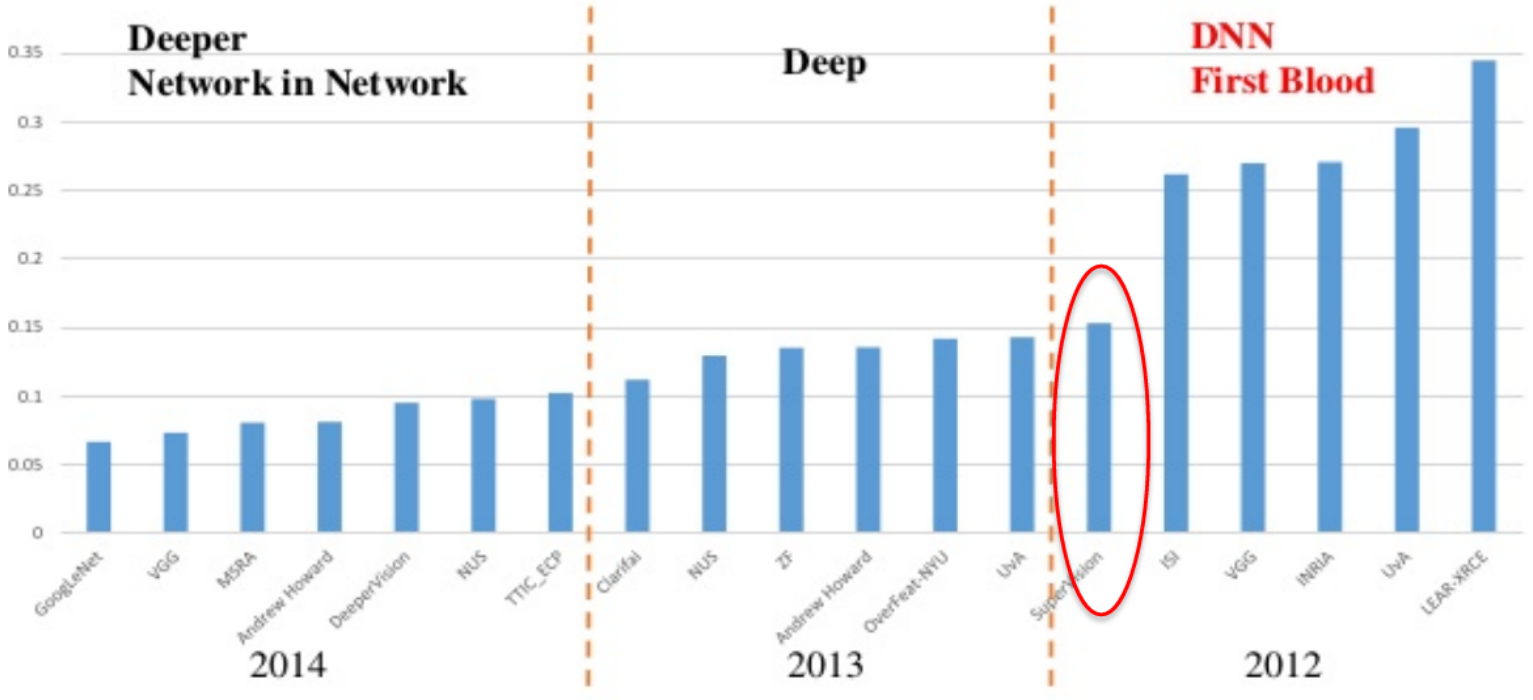
2012 (Krizhevsky, Hinton *et.* al) AlexNet, ILSVRC'2012, GPU – 8 couches

2014 GoogleNet – 22 couches

2015 Inception (Google) – Deep Dream

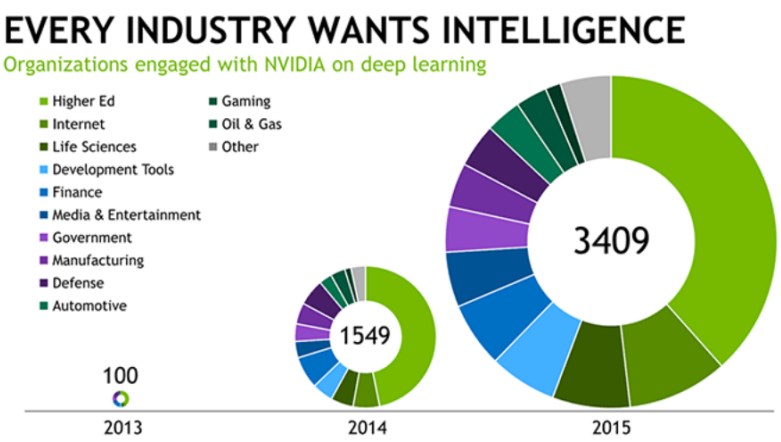
2016 ResidualNet (Microsoft/Facebook) – 152 couches

La rupture dans les performances (ImageNet)



Li Fei-Fei: ImageNet Large Scale Visual Recognition Challenge, 2014 http://image-net.org/

Performances dans autres domaines !

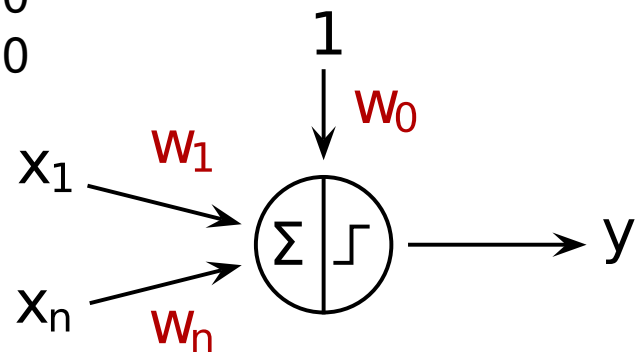


- Sciences de la vie
- Robotique
- Traduction automatique
- Reconnaissance de la parole (Siri)
- Recommandation (Spotify)
- etc.

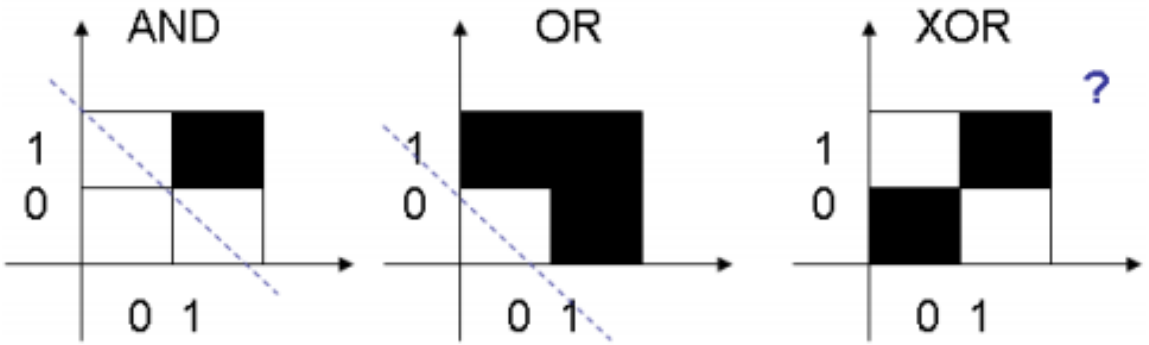
Rappel Perceptron

- $(x_1, \ldots, x_n) \longmapsto y = f_{step}(w_0 + \sum_{i=1}^{n}(w_i x_i)) = f_{step}(w_0 + \langle \vec{w}, \vec{x} \rangle)$ avec

$$f_{step}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

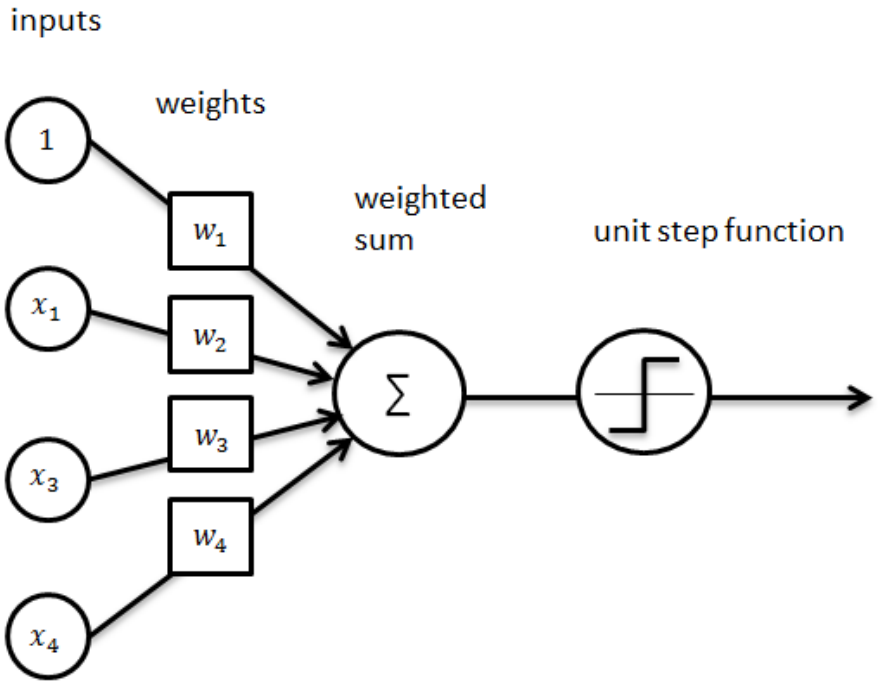


- Les perceptrons ne sont pas capables de résoudre des tâches complexe



- Associer plusieurs perceptrons : perceptron multi-couches ou réseaux de neurones

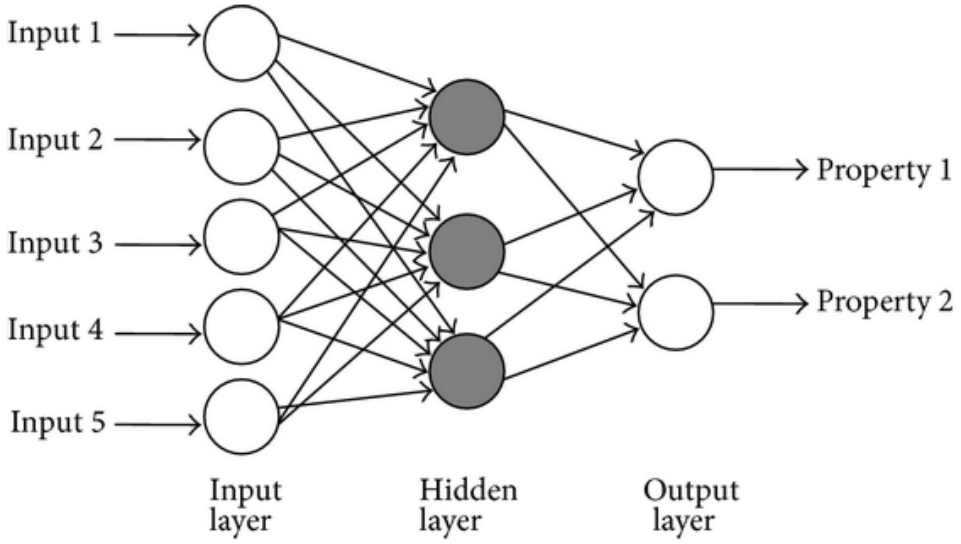
Le perceptron



Inadéquation face aux tâches complexes

- Restrictions à des calculs linéaires
- Fonction de décision discontinue : $x \mapsto A_{\text{step}}(w_0 + \langle w, x \rangle)$

Le perceptron multi-couches (PMC ou MLP)

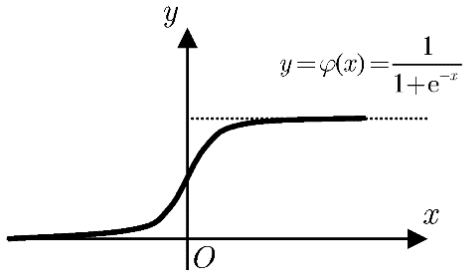


Définition

- Au moins une couche avec fonction d'activation non-linéaire
- Les neurones de la couche i servent d'entrées aux neurones de la couche $i + 1$
- Neurones d'entrées, neurones de sortie, neurones cachés
- Généralement : tous les neurones d'une couche sont connectés à tous les neurones des couches précédentes et suivantes
- Toutes les connections $i \rightarrow j$ sont pondérées par le poids w_{ji}

Réseaux de neurones : Propagation

Fonction d'activation couches internes: sigmoïde



Propagation vers l'avant, couche k

$$a^{(k)} = W^{(k)} h^{(k-1)} + b^{(k)}$$

$$h^{(k)} = f_{sig}^{(k)}(a^{(k)})$$

- $W^{(k)}$: matrice des poids faisant le lien entre la couche $k-1$ et la couche k
- $b^{(k)}$: vecteur des seuils de la couche k
- $h^{(k)}$: vecteur de sortie de la couche k
- $f_{sig}^{(k)}$ (element-wise)

Réseaux de neurones : Propagation

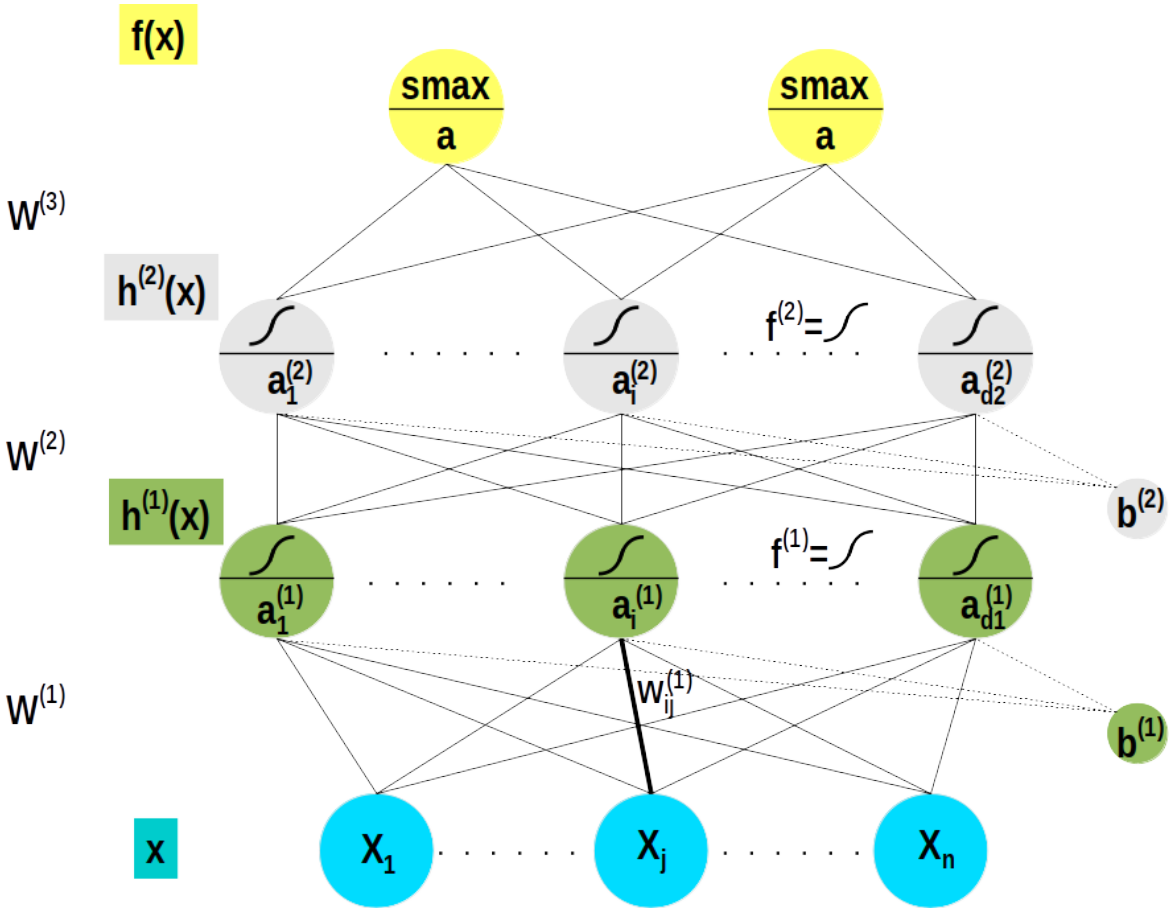
Illustration couche cachée 1

- *i*-ème neurone:

$$a_i^{(1)} = \sum_{j=1}^{n} w_{ij}^{(1)} x_j + b^{(1)}$$

- sortie du *i*-ème neurone :

$$h_i^{(1)} = f^{(1)}(a_i^{(1)})$$



Réseaux de neurones : Propagation

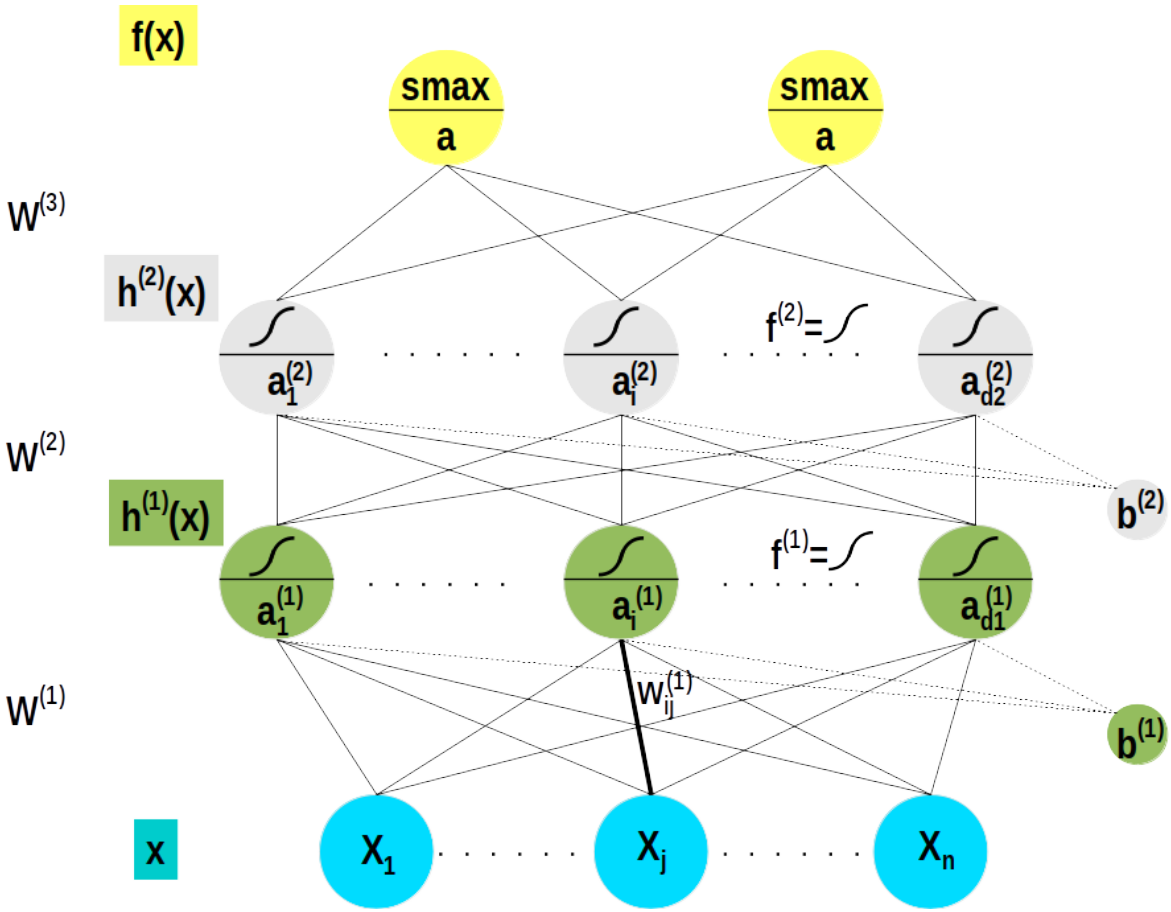
Illustration couche cachée 1

- Activation

$$a^{(1)} = W^{(1)}x + b^{(1)}$$

- Sortie

$$h^{(1)}(x) = f^{(1)}(a^{(1)})$$



Réseaux de neurones : Propagation

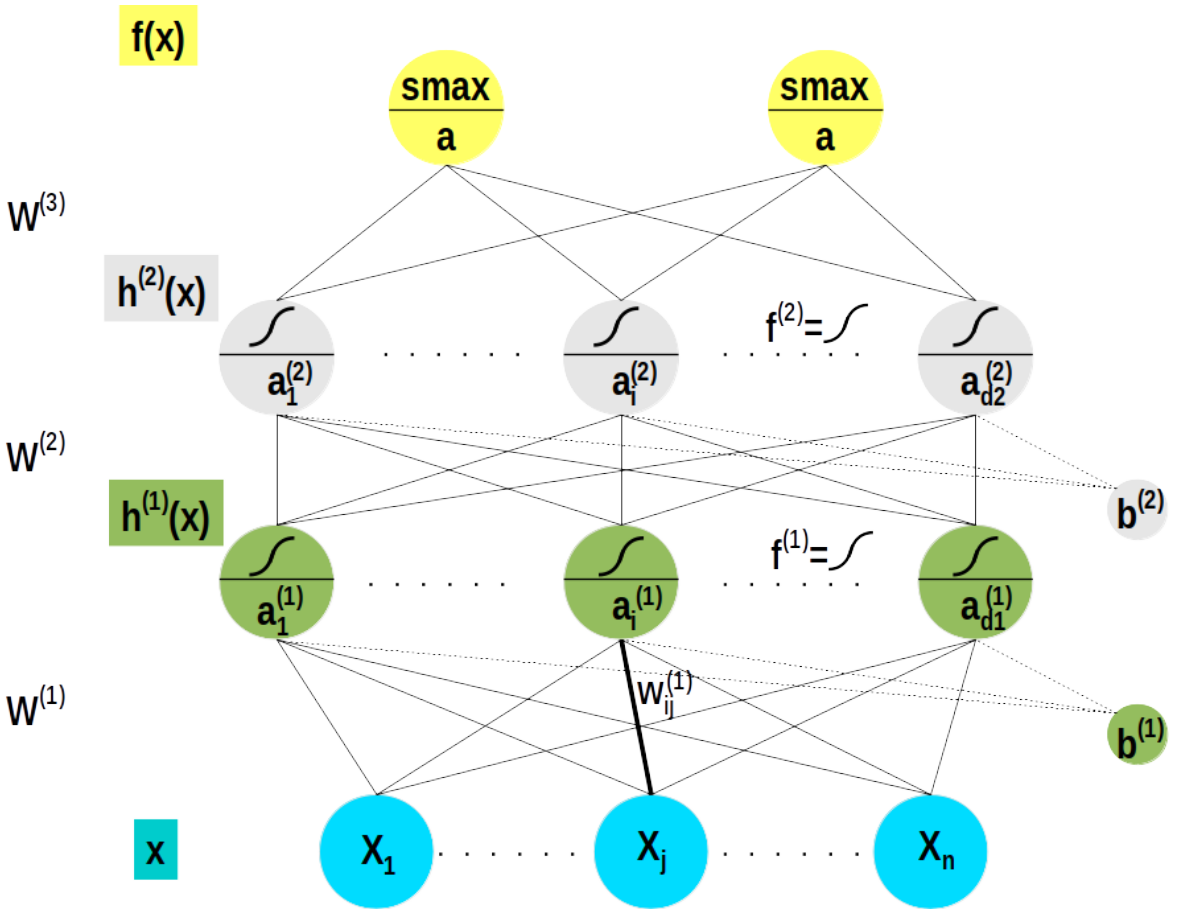
Illustration couche cachée 2

- Activation

$$a^{(2)} = W^{(2)} h^{(1)}(x) + b^{(2)}$$

- Sortie :

$$h^{(2)}(x) = f^{(2)}(a^{(2)})$$



Réseaux de neurones : Propagation

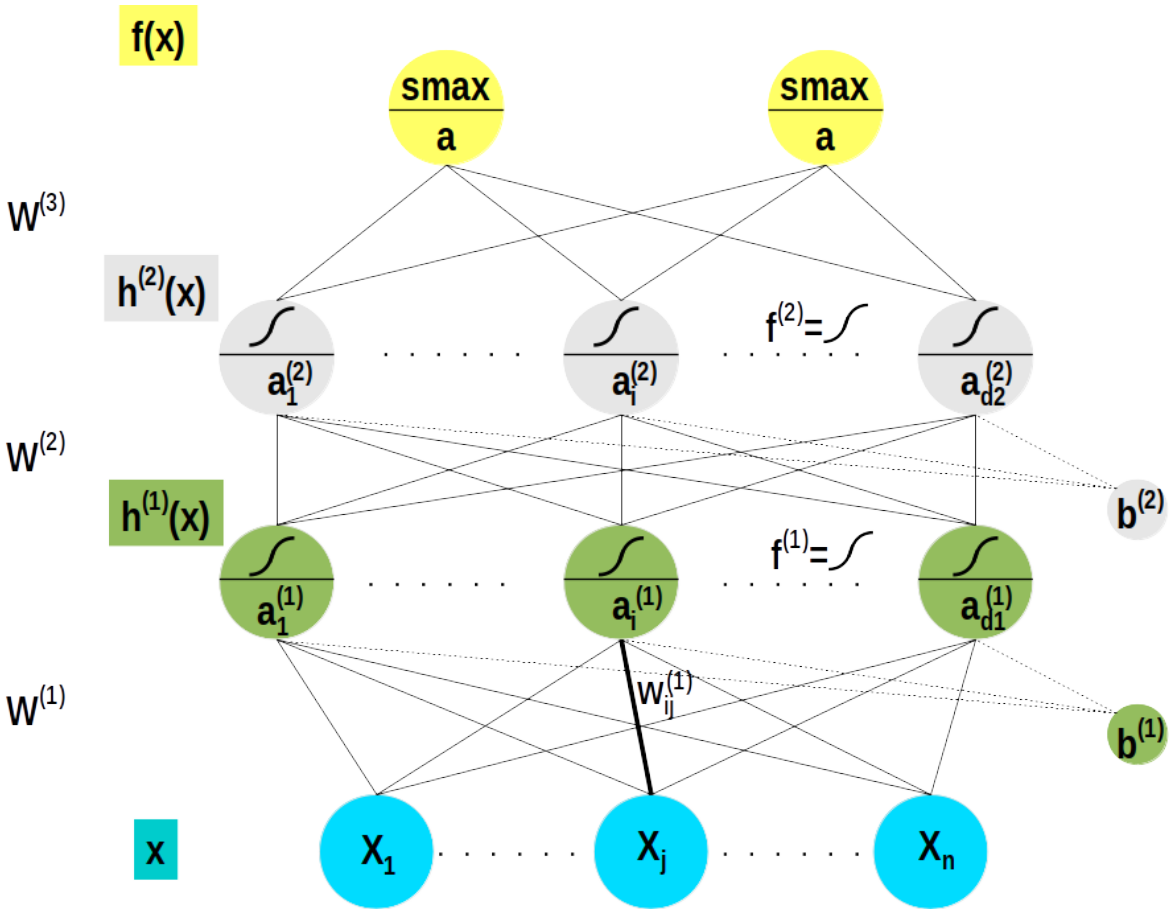
Illustration couche de sortie

- Activation :

$$a^{(3)} = W^{(3)} h^{(2)}(x) + b^{(3)}$$

- Sortie :

$$f(x) = f^{(3)}(a^{(3)})$$



Propriété importante si une seule couche cachée

Toute fonction booléenne peut être calculée par un PMC linéaire à seuil comprenant une seule couche cachée.

Pourquoi ?

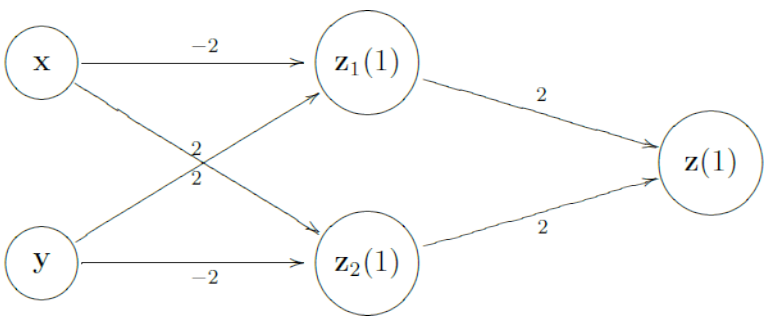
- Chaque neurone est un perceptron à seuil
- n variables binaires
- OU/ET binaires se calculent par un perceptron à seuil
- Fonction booléenne peut être mise sous forme normale disjonctive
- Couches cachée = conjonctions, couche de sortie = disjonction !

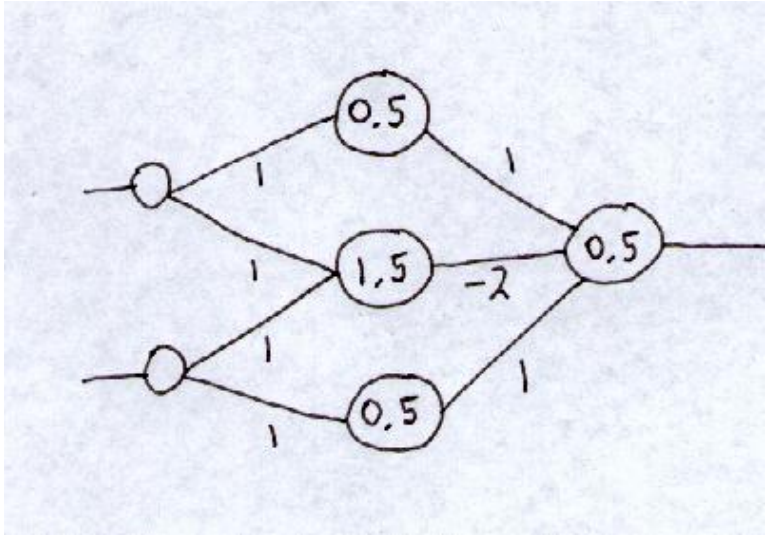
Exemple : le PMC du XOR à une seule couche cachée

On a :

$$a \oplus b = (a \vee b) \wedge \neg(a \wedge b) = a\hat{b} \vee \hat{a}b$$

Dans les deux exemples de PMC pour le XOR :





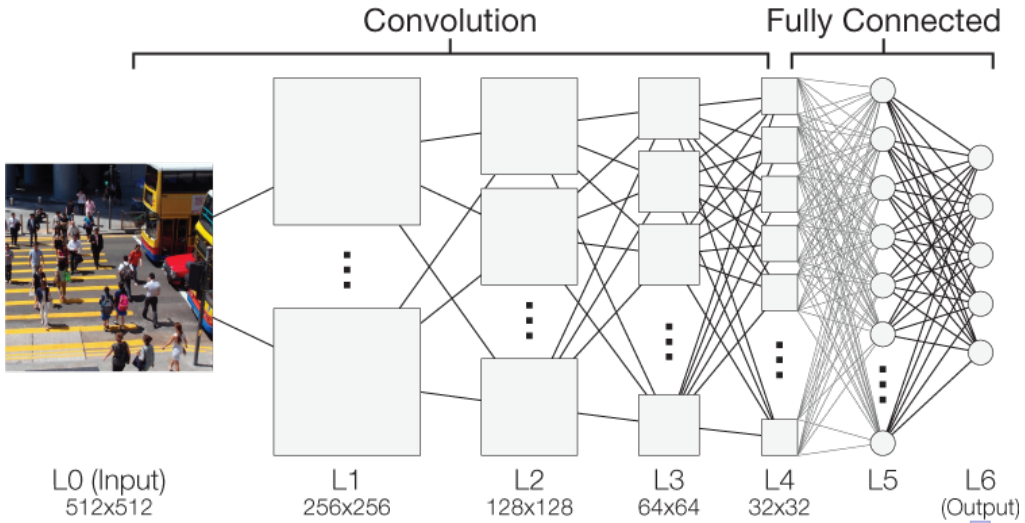
- vérifier la sortie pour les points (0,0) à (1,1)
- donner le PMC équivalent avec les biais plutôt que les seuils

Les réseaux de neurones profonds

DEEP LEARNING

Beaucoup de couches, complexité d'apprentissage

- Autres fonctions d'activation, possibilité de couches avec fonction d'aggrégation (moyenne, etc.), ou fonction classique d'activation
- Architectures particulières (convolution, auto-encodeurs, GAN, etc.)
- Cycles possibles
- Des architectures en pagaille
- Nécessité de GPU



Objectifs de ce cours

- Notion de réseaux de neurones : qu'est-ce que c'est ?
- Quel(s) apprentissage(s) avec des RN ?
- Optimisation non-linéaire, sensibilisation à la descente de gradient (en l'absence de solution analytique)
- Premiers insights sur retro-propagation du gradient, la base du *deep learning*
- Importance actuelle des réseaux de neurones

Plan

Une couche = un ensemble de perceptrons

Communication de couche à couche suivante

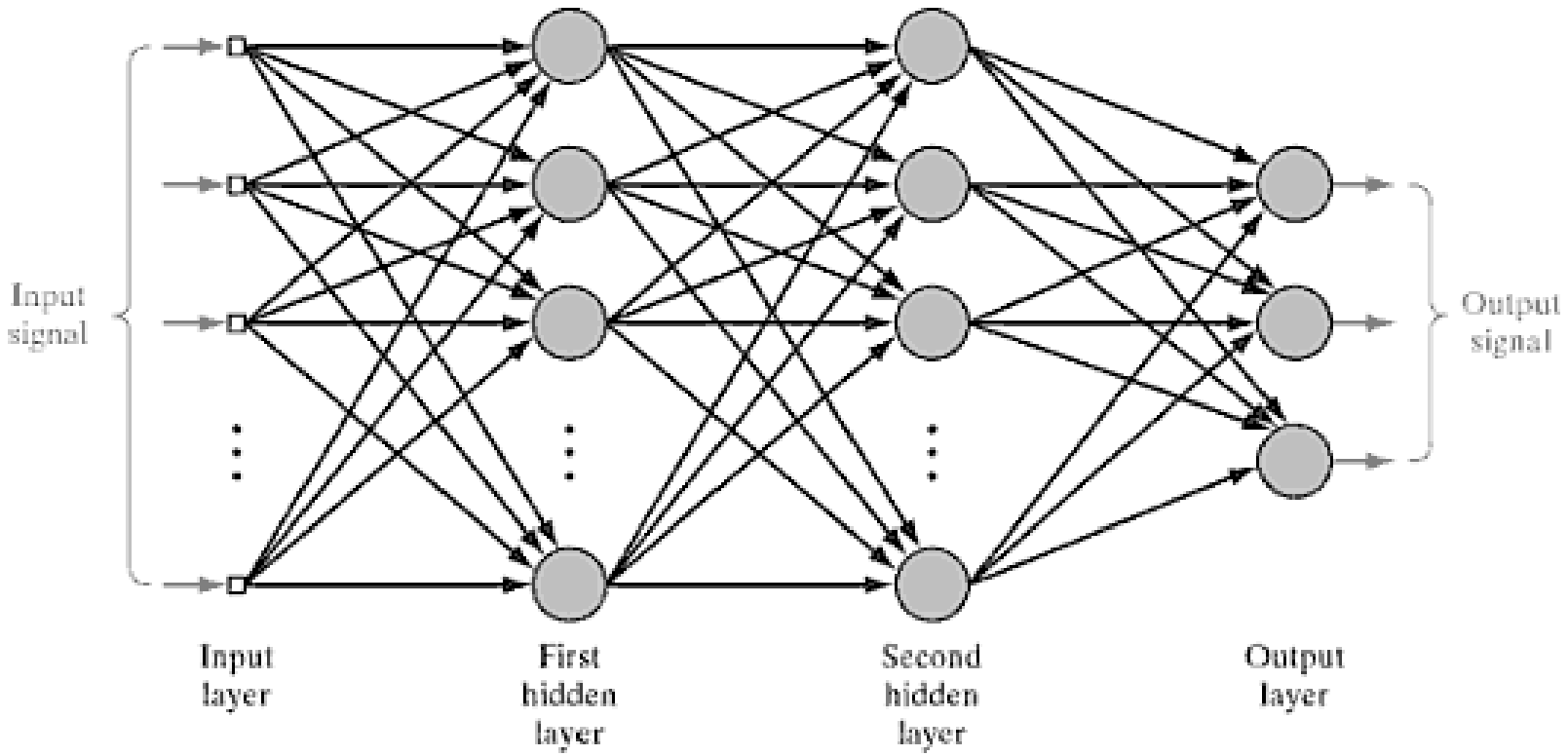


FIGURE 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

Apprendre avec risque le plus faible

Cas d'un PMC à une couche cachée n entrées, p sorties

- Soit un exemple $(x, y) \in S, x = (x_1, \cdots x_n), y = (y_1, \cdots, y_p)$, la fonction de décision à la sortie k est :

$$h_k(x) = A_o(W_o A(\langle w, x \rangle))$$

- avec A non linéaire, non polynomiale, continue, dérivable, approximant la fonction de Heaviside :
  - $A(x) = x tanh(x)$
  - $A(x) = \frac{1}{1+e^{-x}}$ fonction logistique (sigmoïde)
- et A_o continue dérivable, softmax en classification multi-classes

Quel que soit le nombre de couches

Erreur et risque : dépend de tous les w

- Apprentissage avec risque minimal = minimiser $\ell(h_k(x), y_k)$
- Fixons la fonction de risque $\ell(h_k(x), y) = (h_k(x) - y_k)^2$, continue dérivable si $h_k(x), y \in \mathbb{R}$ (régression).

Comment minimiser le risque

Expression de la fonction à minimiser

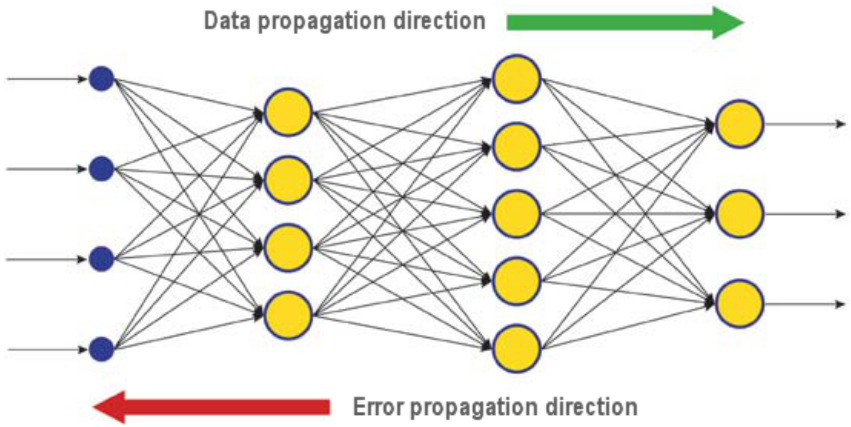
L'erreur (si fonction de perte = mean-square, h_k sortie du neurone k) :

$$E(w) = \frac{1}{2} \sum_{(x,y)\in S} \sum_{k=1}^{p} (h_k - y_k)^2$$

avec $h_k = A_0(w_{k0} + \sum_{j\in \text{Pred}(k)}(w_{kj} a_j))$

- Contribution de tous les neurones et de toutes les synapses
- Prise en compte de tous les exemples de l'échantillon

Activation dans un sens, propagation de l'erreur dans l'autre



Un problème d'optimisation

Optimisation non-linéaire

Fonctions continues, dérivables

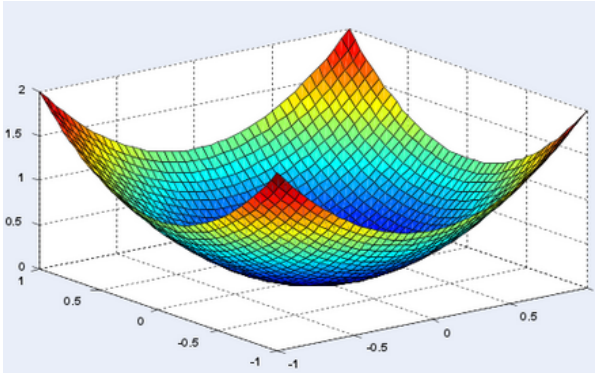
Similarité avec le perceptron

- Minimisation de l'erreur sur chaque présentation individuelle des exemples (règle de Widrow-Hoff)
- On doit minimiser :

$$E = E_{(x,y)}(w) = \frac{1}{2}\sum_{k=1}^{p}(h_k - y_k)^2$$

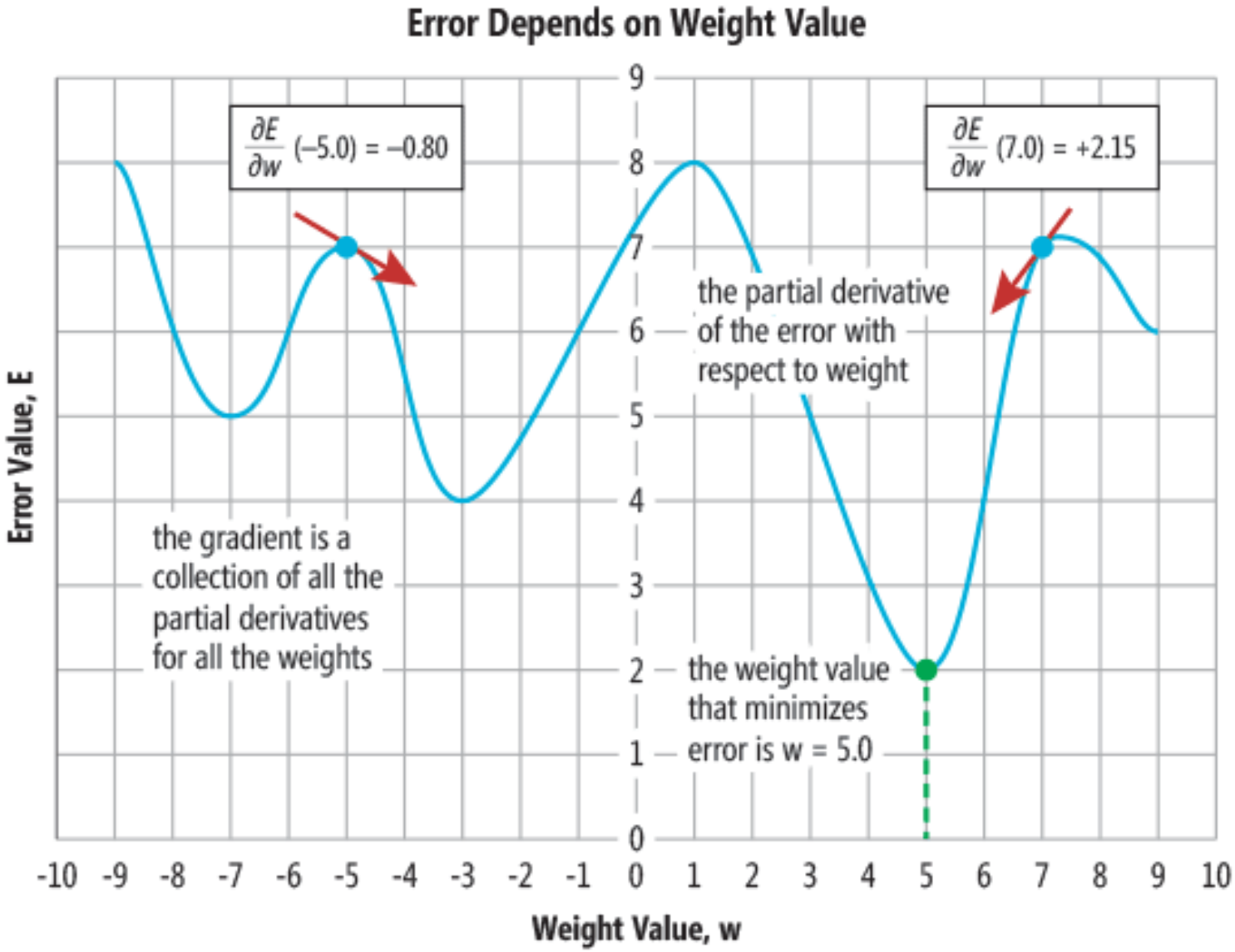
- Plus généralement $\arg\min_\theta L(f_\theta, S) = \arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x^i), y^i)$,

Minimiser une fonction : recherche descendante du point de dérivée nulle de la fonction



L'erreur dépend des poids

Trouver les *w* qui mènent à la plus petite erreur, si possible

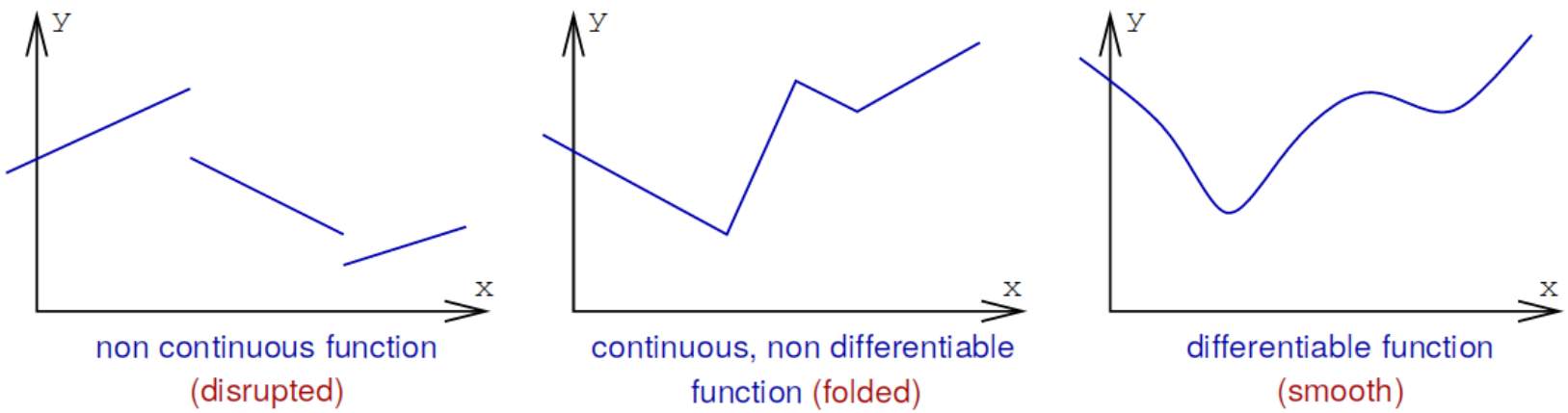


Rappels : fonctions, dérivées, tangentes, etc.

Simplifions les définitions

Soit $x \in \mathcal{I} \subseteq \mathbb{R}$, et la fonction $f(x) \in \mathbb{R}$

- f continue, dérivable sur $\mathcal{I}$
- f' est une fonction indiquant l'orientation de f sur $\mathcal{I}$ (croissance, décroissance, stagnation)
- $f'(x)$ indique la pente de f au point x, définit l'équation de la tangente à f en x



Rappels : fonctions, dérivées, tangentes, etc.

Simplifions les définitions

Soit $x \in \mathcal{I} \subseteq \mathbb{R}$, et la fonction $f(x) \in \mathbb{R}$

- f continue, dérivable sur $\mathcal{I}$
- f' est une fonction indiquant l'orientation de f sur $\mathcal{I}$ (croissance, décroissance, stagnation)
- $f'(x)$ indique la pente de f au point x, définit l'équation de la tangente à f en x

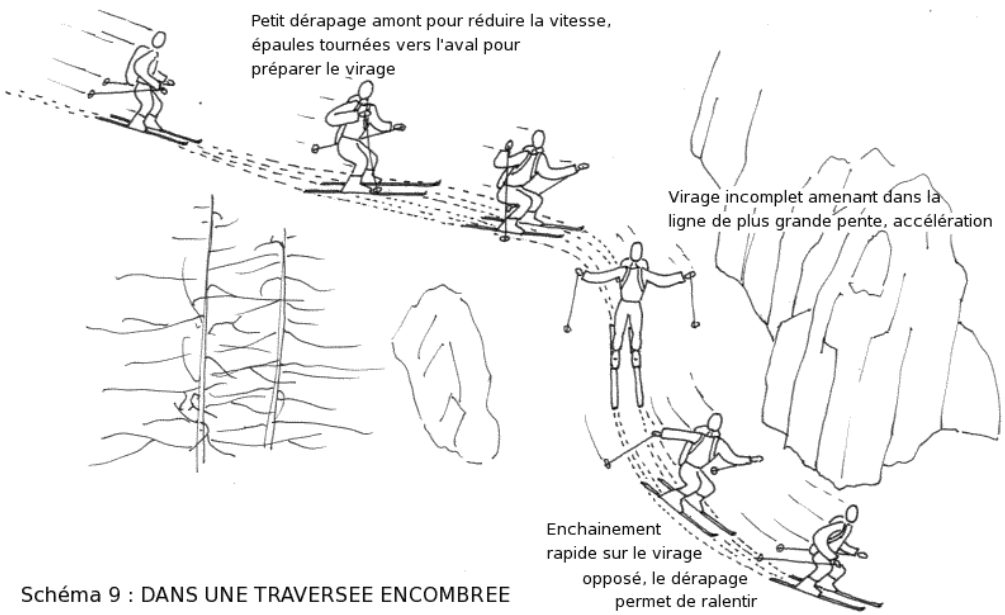


Schéma 9 : DANS UNE TRAVERSEE ENCOMBREE

Rappels : fonctions, dérivées, tangentes, etc.

Simplifions les définitions

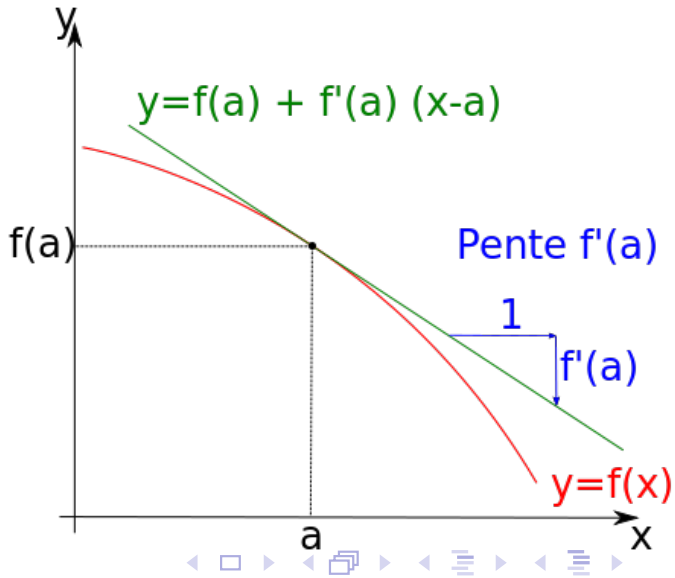
Soit $x \in \mathcal{I} \subseteq \mathbb{R}$, et la fonction $f(x) \in \mathbb{R}$

- f continue, dérivable sur $\mathcal{I}$
- f' est une fonction indiquant l'orientation de f sur $\mathcal{I}$ (croissance, décroissance, stagnation)
- $f'(x)$ indique la pente de f au point x, définit l'équation de la tangente à f en x

Exemple

Cas observés :

- $f'(x) < 0$
- $f'(x) = 0$
- $f'(x) > 0$



Fonctions de plusieurs variables, dérivées partielles

Cas de la fonction $y = f(x_1, x_2)$, $f : \mathcal{I}^2 \mapsto \mathbb{R}$

- Etude de l'orientation de f lorsqu'une seule composante d'un point varie
- $\frac{\partial f(x_1,x_2)}{\partial x_1}$: une fonction $f'(x_1, x_2)$ qui indique comment varie f lorsque seule x_1 varie un peu (autres variables considérées comme constantes)
- Exemple :

$$f(x_1, x_2) = 3x_1^2x_2^5 - 2x_1x_2^3 + 2x_1^4x_2^2 - 4x_1 - 2x_2 + 5$$

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 6x_1x_2^5 - 2x_2^3 + 8x_1^3x_2^2 - 4$$

- Les valeur et signe de $\frac{\partial f(x_1,x_2)}{\partial x_1}$ indiquent l'orientation de la courbe de f sur l'axe x_2 lorsque seule x_1 varie (un peu)

Généralisation à $y = f(X), X \in \mathcal{I}^d$: gradient

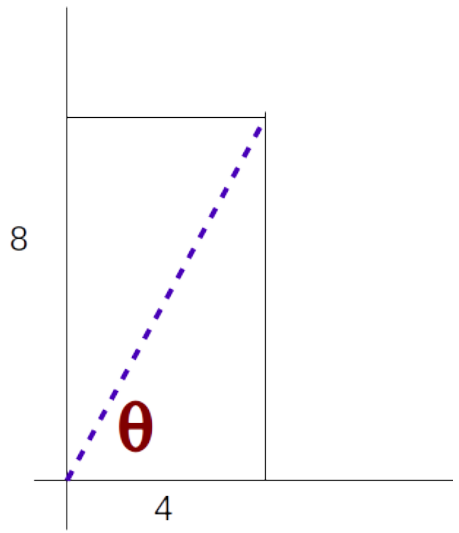
Tout est fixé sauf x_i : où et à quelle vitesse se dirige f ?

$$\frac{\partial f(x_1, \cdots, x_d)}{\partial x_i}$$

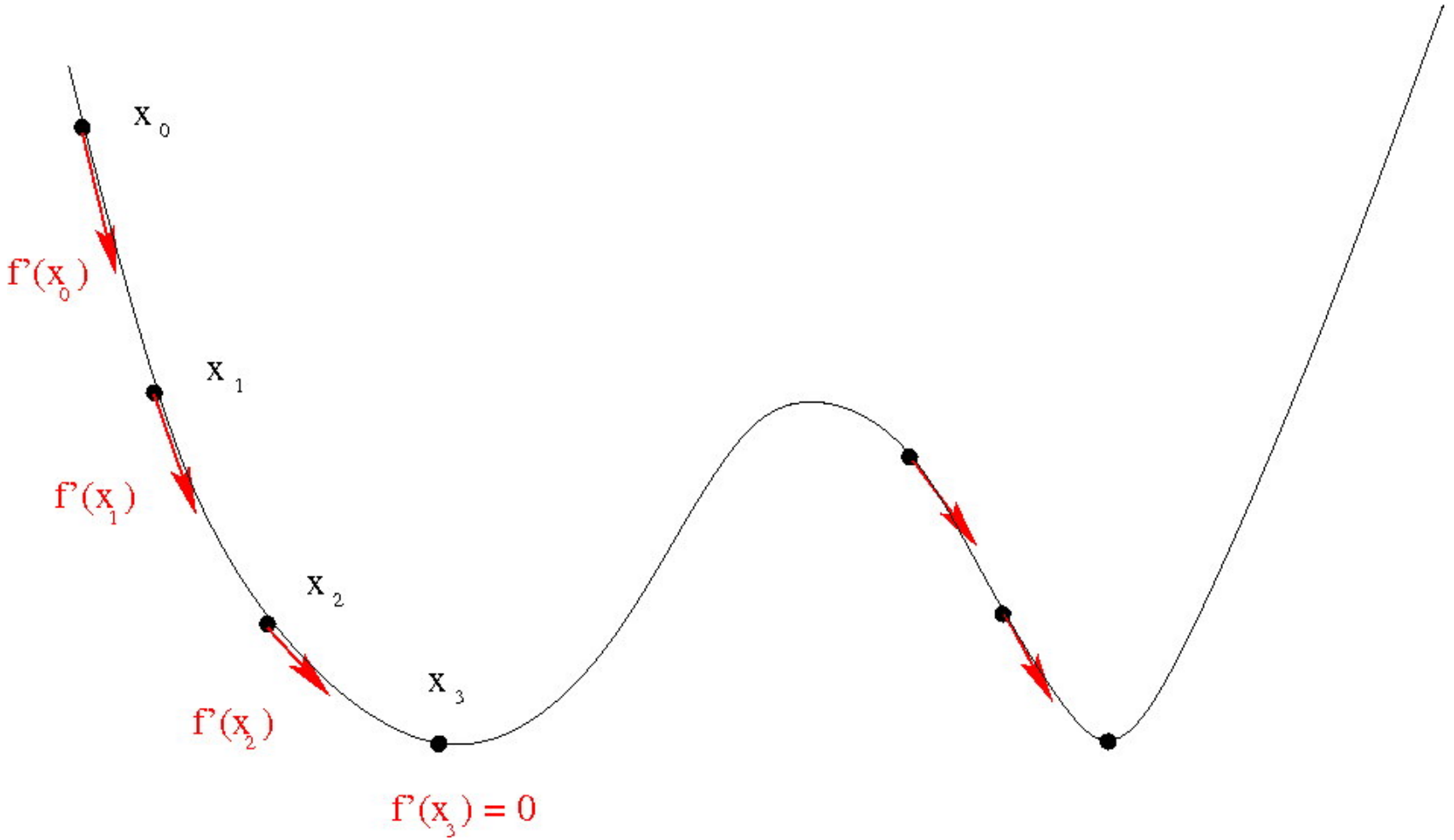
Exercice

- Utiliser le gradient pour évaluer la direction asscendante la plus rapide de la fonction $f(x_1, x_2) = x_1 x_2^2$ au point $P = (x_1 = 2, x_2 = 2)$.
- Indice : calculer dérivées partielles selon x_1 puis x_2 au point $P = (2, 2)$
- Solution :

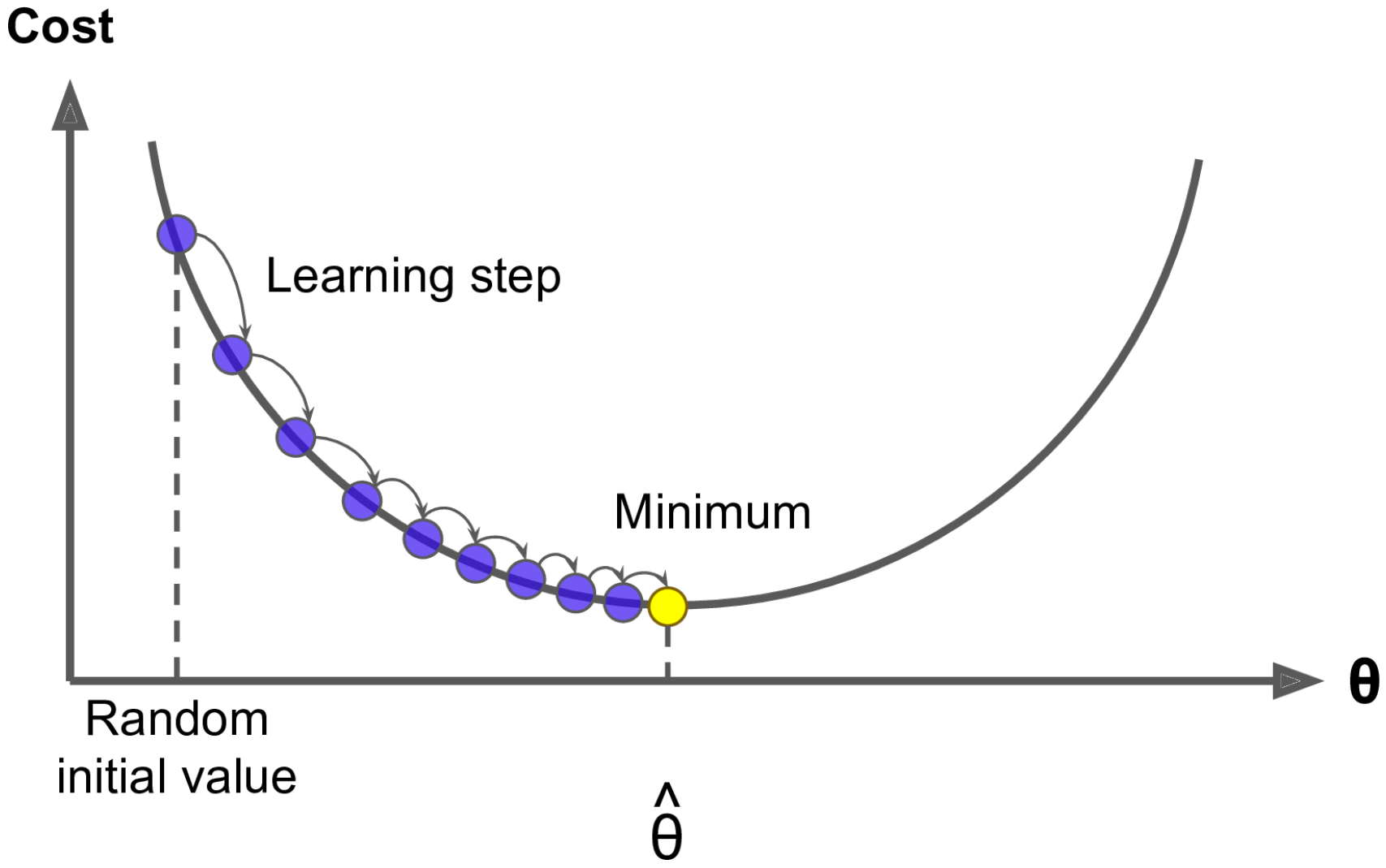
- $\frac{\partial f}{\partial x_1} = x_2^2$ et $\frac{\partial f}{\partial x_2} = 2x_1 x_2$
- Sur x_1 : 4, Sur x_2 : 8
- Direction de l'orientation en ce point $\theta = \tan^{-1}(\frac{8}{4}) = 63.4$
- Magnitude de l'orientation en ce point $\Delta f(P) = \sqrt{(\frac{\partial f}{\partial x_1})^2 + (\frac{\partial f}{\partial x_2})^2} = 8.944$ en P



Descente de gradient : intuitivement



Descente de gradient : intuitivement – cont'd



Un algorithme itératif

Cas en une dimension

Soit $f : \mathbb{R} \to \mathbb{R}$ la fonction à minimiser :

- Fixer un point de départ x_0
- Construire itérativement une suite de valeurs x_i :

$$x_{i+1} = x_i - \alpha f'(x_i)$$

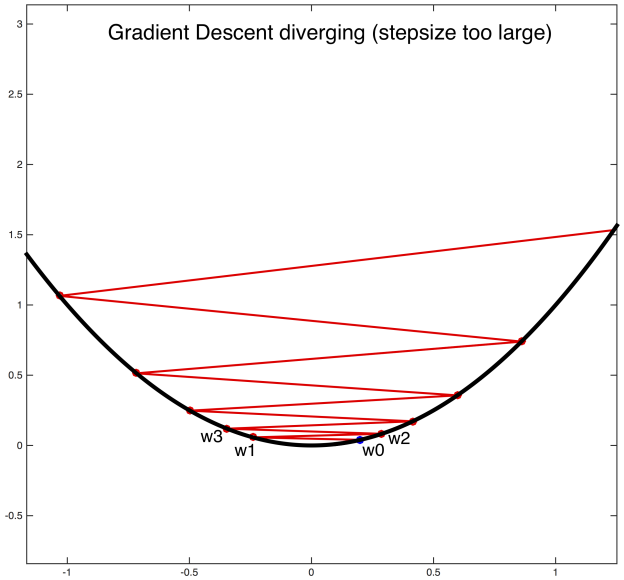
- de façon équivalent, on écrit

$$\Delta x = x_{i+1} - x_i = -\alpha f'(x)$$

- α est le pas d'apprentissage
- Arrêt lorsque $\Delta x \leq \epsilon$, ou lorsque $i > M$ (valeurs seuils)

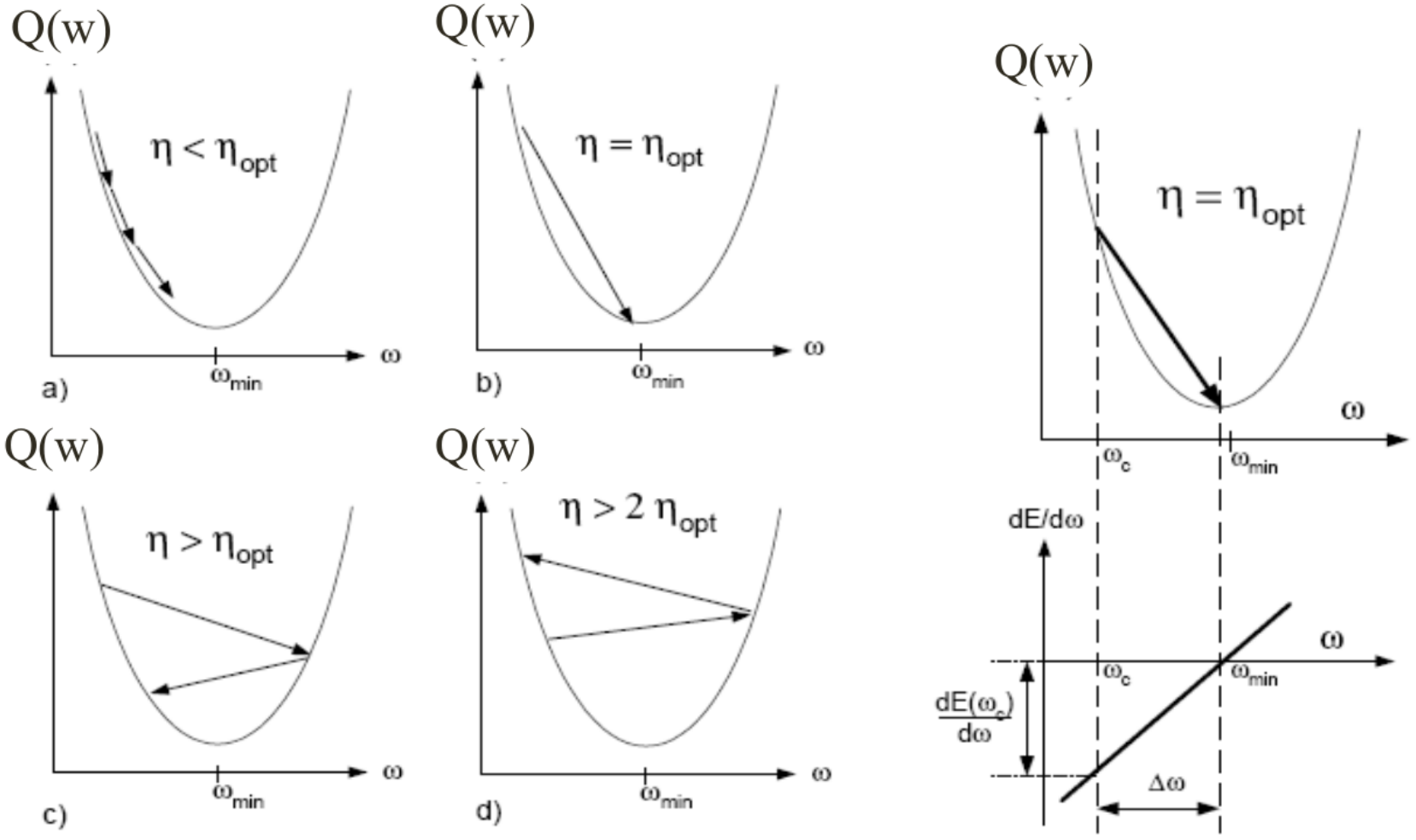
Quelques écueils

- Convergence assurée ?
- Rapidité de convergence
- Convexité de la fonction, minima locaux



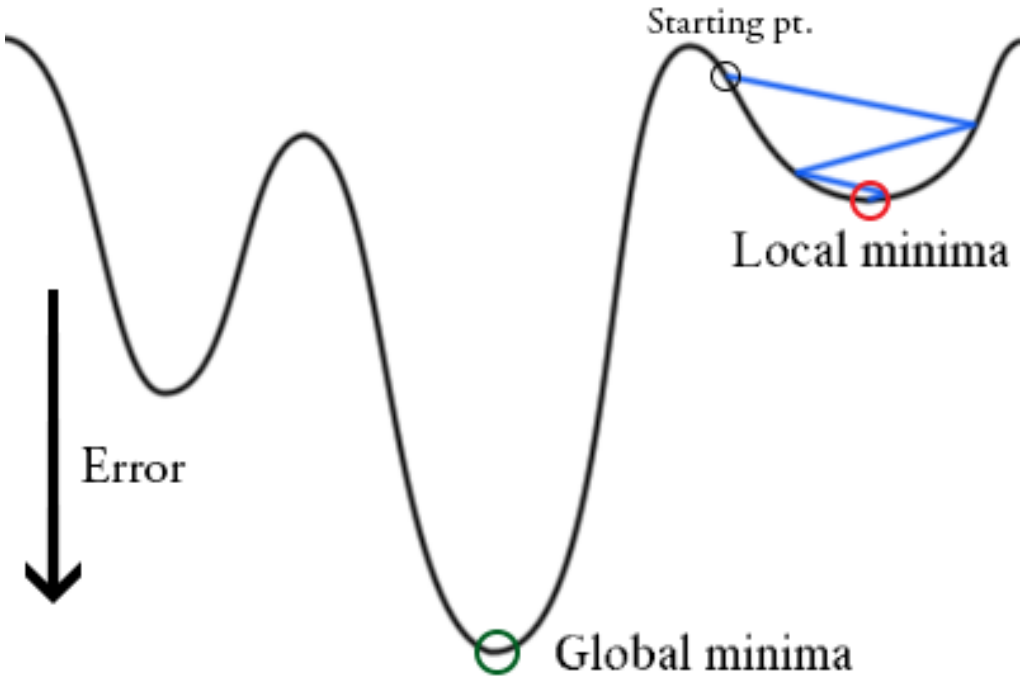
Le rôle du pas d'apprentissage

Petit, grand, adaptatif ?



Le rôle du point initial

Convergence vers l'idéal : minima locaux



Un algorithme itératif

En plusieurs dimensions

Dérivées partielles, soit $E : \mathbb{R}^d \to \mathbb{R}$ la fonction à minimiser

- Formule de mise à jour :

$$\Delta X = -\alpha \vec{\nabla} E(X)$$

- $\vec{\nabla} E$ est la fonction gradient de E, et $\vec{\nabla} E(X)$ est un vecteur de d coordonnées !
- Mise à jour de la coordonnées x_j de X :

$$\Delta x_j = -\alpha \frac{\partial E}{\partial x_j}(X)$$

- Critères d'arrêt

Descente de gradient : l'algorithme

Problème général

Trouver les valeurs du vecteur θ qui minimise le coût d'une fonction f_θ

$$\arg\min_\theta L(f_\theta, S) = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x^i), y^i),$$

Algorithme générique

$\theta^0 \leftarrow$ initialisé aléatoirement

Répéter

$$\theta^{t+1} \leftarrow \theta^t - \alpha \frac{\partial L(f_\theta, S)}{\partial \theta}$$

Tant que $\left\| \frac{\partial L(f_\theta, S)}{\partial \theta} \right\| > \epsilon$ (n'est pas proche de 0)

Difficultés

- Trouver le bon α ?
- Calcul de ∂L coûteux quand S est grand

Exercice

Soit

$$E(x_1, x_2) = (x_1 - 1)(x_1 - 2) + (x_2 + 3)(x_2 + 4)$$

- Calculer $\frac{\partial E}{\partial x_1}(x_1, x_2)$
- Calculer $\frac{\partial E}{\partial x_2}(x_1, x_2)$
- Appliquer l'algorithme de descente du gradient avec $x0 = (2, -4)$ et $\alpha = 0.1$
- Tous les calculs sont-ils nécessaires ?
- Quid d'une solution analytique ?
- Y at'il des minima locaux ?

Perceptron Multi-couches et backpropagation

Utilisation de la descente de gradient

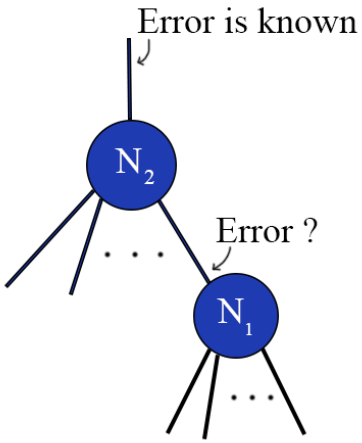
Possible car fonctions dérivables (fonctions d'activations et fonction de coût)

Comment calculer $\partial L/\partial w_i$ lors de la mise à jour $w_i \leftarrow w_i - \alpha \partial L(f_w)/\partial w_i$?

Exemple après exemple :

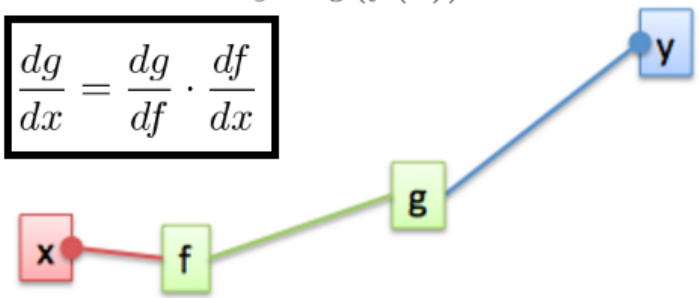
1. Calculer l'activation
2. Calculer le gradient en sortie
3. (retro-)Propager le gradient vers les entrées en déroulant une chaîne de propagation de la sortie vers la première couche

Observation

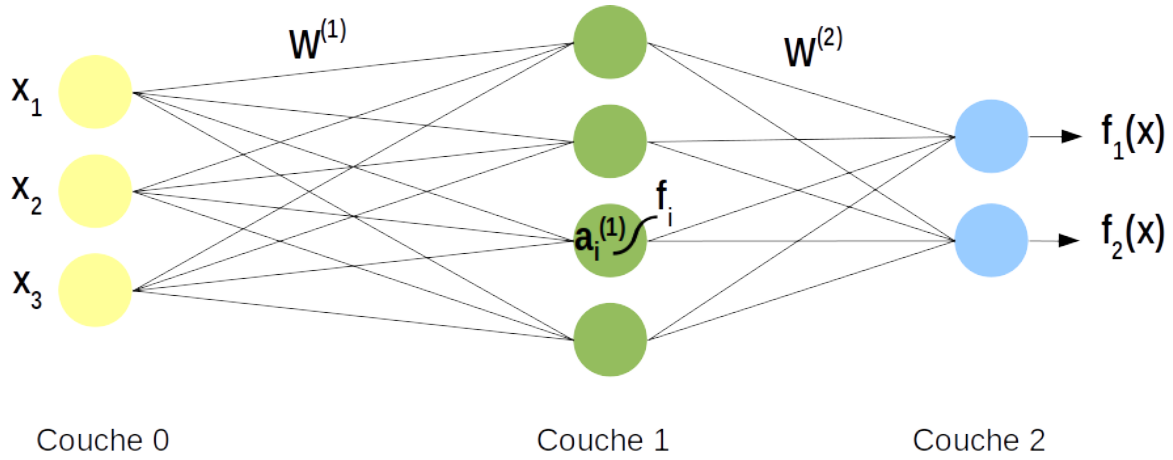


$$y = g(f(x))$$

$$\frac{dg}{dx} = \frac{dg}{df} \cdot \frac{df}{dx}$$



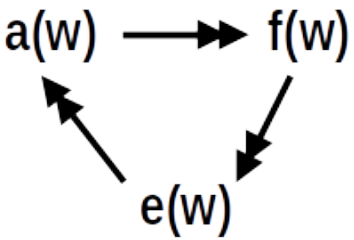
Algorithme dans le cas d'une seule couche cachée



1. Initialisation de tous les poids du réseau (petites valeurs)
2. Pour chaque exemple (x, y) de S faire :
   - 2.1 prediction = $f_k(x)$ = activation-du-MLP(x)
   - 2.2 calculer l'erreur $e_k = \ell(f_k - y_k)$ pour chaque sortie, et $E = \sum_k e_k$
   - 2.3 calculer Δw_{jk} pour tous les poids entre la couche cachée et la couche de sortie
   - 2.4 calculer Δw_{ij} pour tous les poids entre la couche d'entrée et la couche cachée
   - 2.5 Mettre à jour tous les poids

   Jusqu'à satisfaction d'un critère d'arrêt
3. Retourner le réseau

Principe de la *back propagation*



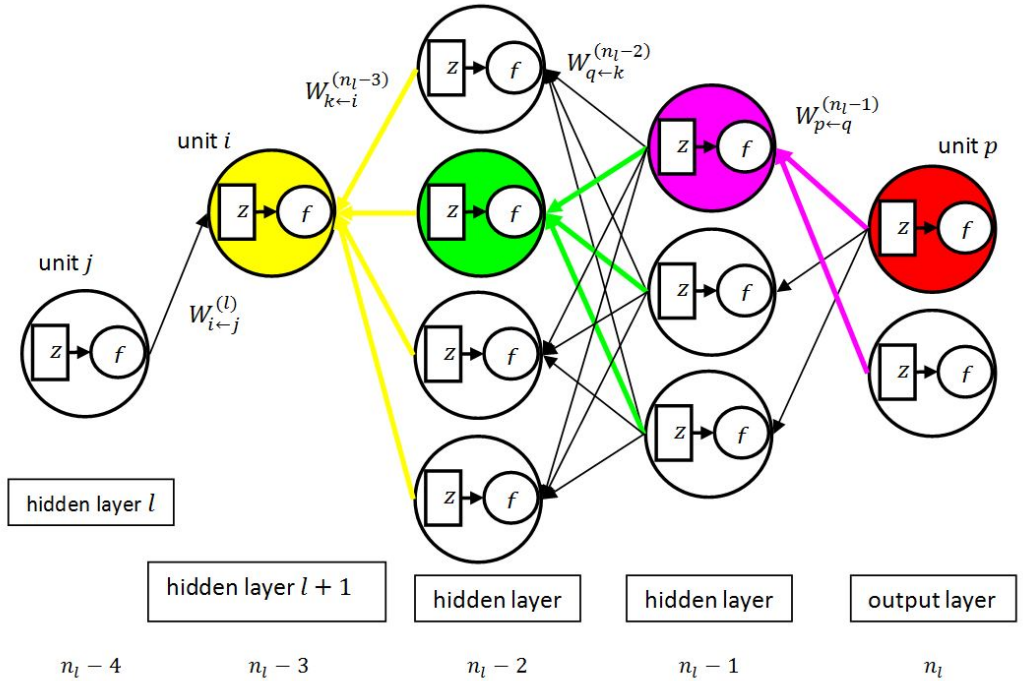


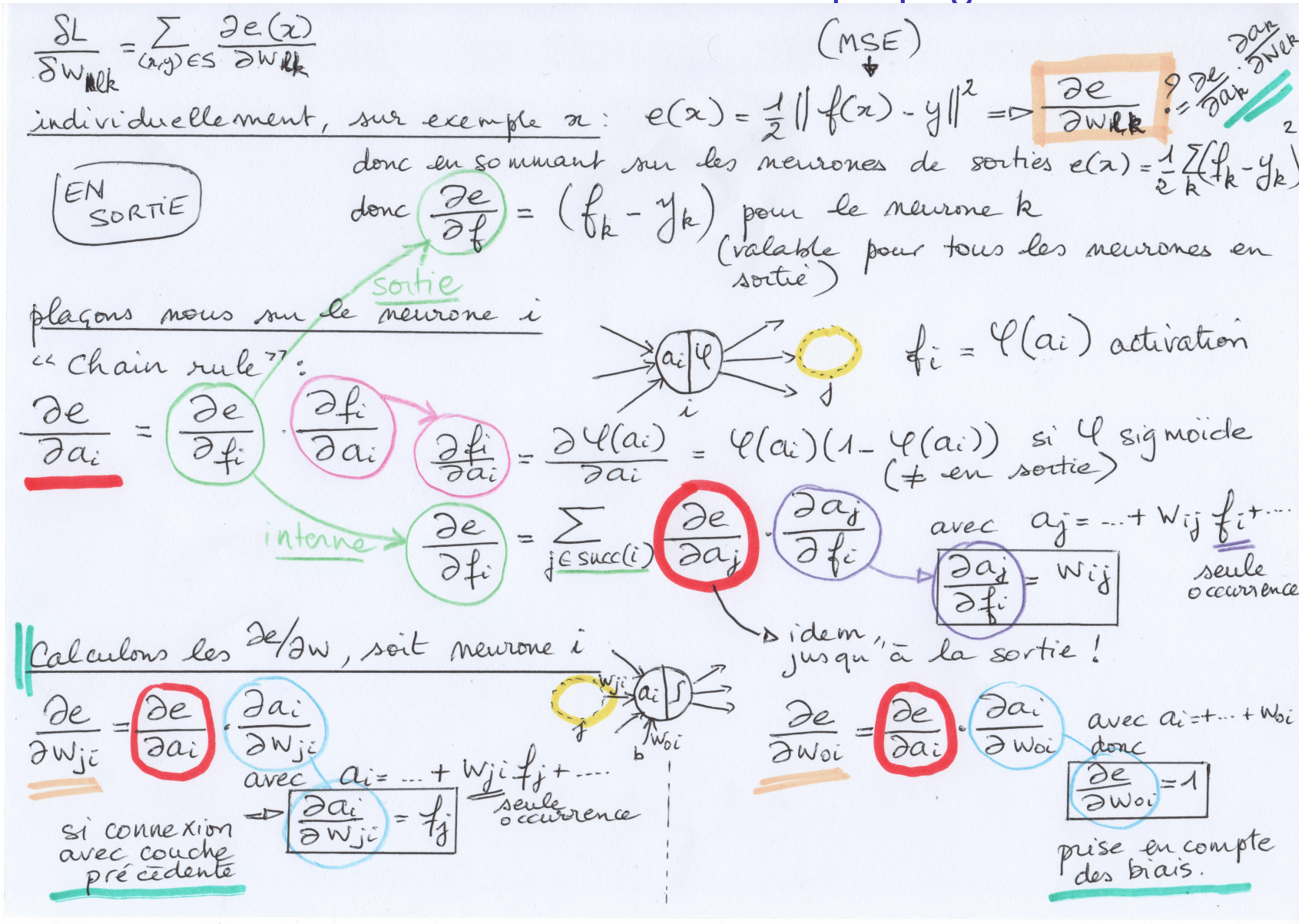
Figure 2 On each layer, the error is propagated to every unit there, but only one of them is shown using a coloured path.

$$\arg\min_W L(f(W), S)$$

$$L(f,S) = \frac{1}{2}\sum_{(x,y)\in S} ||f(x)-y||^2 = \sum_{(x,y)\in S} e(x)$$

$$\frac{\partial L}{\partial W_{kl}} = \sum_s \frac{\partial e(x)}{\partial W_{kl}}$$

Réseaux de neurones : Rétropropagation



Réseaux de neurones : Récapitulatif

Les étapes principales

Simplification de notations, mais dépendances
$e, a, f = e(W), a(W), f(W), e(a), a(f), e(f)$

1. Calcul du gradient en sortie (dépend de y)
2. Calcul du gradient entre chaque couple de neurones $j \to i$ $(\partial e/\partial w_{ji})$
3. Calcul du gradient à la sortie d'un neurone i $(\partial e/\partial a_i)$ (back propagation)
4. Calcul du gradient à la sortie d'un neurone i vers j $(\partial e/\partial f_i)$

La *chain rule*

- Erreur en entrée du neurone i via $j = e(a_i(w_{ji}))$ donc : $\frac{\partial e}{\partial w_{ji}} = \frac{\partial e}{\partial a_i}\frac{\partial a_i}{\partial w_{ji}}$
- Erreur en sortie du neurone $i = e(f_i(a_i))$ donc : $\frac{\partial e}{\partial a_i} = \frac{\partial e}{\partial f_i}\frac{\partial f_i}{\partial a_i}$
- Erreur en sortie du neurone i vers le neurone $j = e(a_j(f_i))$ donc :

$$\frac{\partial e}{\partial f_i} = \frac{\partial e}{\partial a_j}\frac{\partial a_j}{\partial f_i}$$

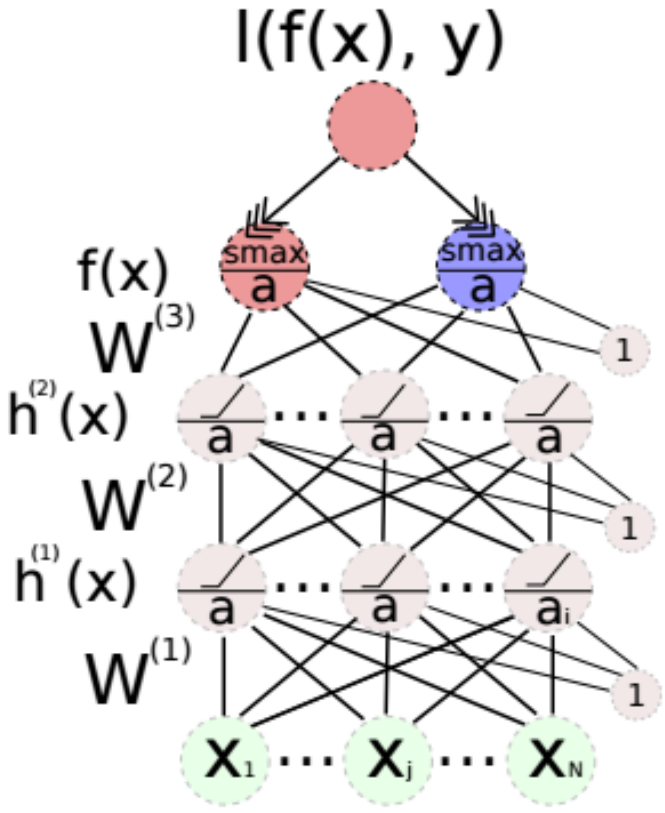
Réseaux de neurones : Illustration

- Calcul du gradient pour tout le réseau
  - Gradient sur les pondérations - couche 3 (de sortie):

$$\frac{\partial \ell(f(x), y)}{\partial W^{(3)}} = \frac{\partial \ell(f(x), y)}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial W^{(3)}}$$

  - $\frac{\partial \ell(f(x), y)}{\partial a^{(3)}}$ dépend de la fonction coût ℓ
  - $\frac{\partial a^{(3)}}{\partial W^{(3)}} = (h^{(2)}(x))^\top$
  - Gradient sur la couche précédente

$$\begin{aligned}\frac{\partial \ell(f(x), y)}{\partial h^{(2)}} &= \frac{\partial \ell(f(x), y)}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial h^{(2)}} \\ &= \frac{\partial \ell(f(x), y)}{\partial a^{(3)}} (W^{(3)})^\top\end{aligned}$$



Rappel - Propagation:

$$a^{(3)} = W^{(3)} h^{(2)}(x) + b^{(3)}$$

Réseaux de neurones : Illustration

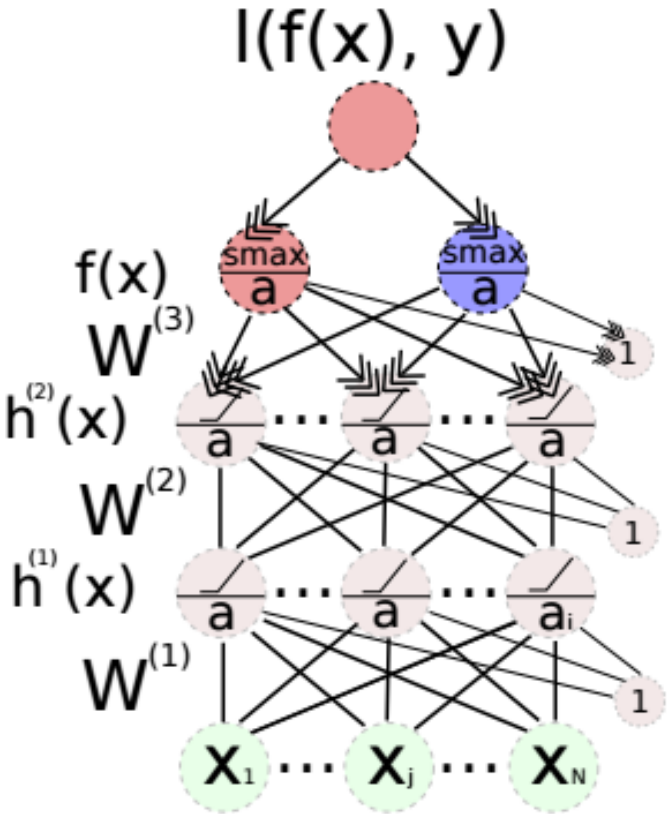
- Calcul du gradient pour tout le réseau
  - Gradient sur les pondérations - couche 2 :

$$\frac{\partial \ell(f(x), y)}{\partial W^{(2)}} = \frac{\partial \ell(f(x), y)}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W^{(2)}}$$

  - $\frac{\partial h^{(3)}}{\partial a^{(2)}}$ dépend de la fonction d'activation
  - fonction sigmoide $\sigma(\cdot)$

$$\frac{\partial h^{(3)}}{\partial a^{(2)}} = \sigma(a_i^{(2)})(1 - a_i^{(2)})$$

  - même procédure comme précédemment
  - rétropropager sur la couche 1



Rappel - Propagation:

$$a^{(2)} = W^{(2)} h^{(1)}(x) + b^{(1)}$$

Réseaux de neurones : Algorithme d'apprentissage

- Perceptron multicouche - Algorithme générique

1. initialiser le vecteur de paramètres w
2. initialiser les paramètres - méthode de minimisation
3. **Répéter**
4. **Pour tout** $(x, y) \in D$ **faire**
5. appliquer x au réseau et calculer la sortie correspondante
6. calculer $\frac{\partial \ell(f(x), y)}{\partial W_{ij}}$ pour toutes les pondérations
7. **Fin pour**
8. calculer $\frac{\partial L(f, D)}{\partial W_{ij}}$ en sommant sur toutes les données d'entrée
9. appliquer une mise à jour du vecteur de pondération - méthode de minimization
10. **Tant que** l'erreur n'a pas convergé

Rapport nombre de w / nombre de données

Profondeur du réseau

= nombre de couches $\times$ nombres de noeuds par couche

- taille moyenne = μ^p, si μ est le nombre moyen de neurones par couches
- autant de paramètres à apprendre
- nécessite beaucoup de données pour que le problème ne soit pas sous-défini

120 paramètres et seulement 15 exemples ?

Risque de sur-apprentissage

Problème sur-défini

- Régularisation (sparsité dans la fonction de minimisation)
- Drop-out

Interprétabilité

Variantes

- Drop-out
- Architectures pré-apprises (apprentissage de représentations)
- La RELU (convnet)
- Face au bruit (dans données) : ajouter des couches
- Autres descentes de gradient (conjugué, momentum – pas adaptatif -, stochastique, etc.)
- Quelle architecture, quels hyper-paramètres (activation, nb de couches, nb de neurones/couche, linéarité/aggrégation des activations, etc.), rapport $N * d$ (bonne définition du problème), etc.

Exercices (1)

Fonction booléenne

Soit la fonction booléenne $f(x, y, z) = x\bar{y} + xyz + \bar{x}\bar{y}\bar{z}$. Déterminer un perceptron linéaire à seuil pour chacun des trois monômes de f, puis déterminer un PMC calculant f.

Algorithme de retropropagation

En utilisant la fonction sigmoïde, on souhaite pénaliser les poids élevés. On modifie donc la fonction erreur en introduisant un terme de pénalisation :

$$E(w) = \frac{1}{2} \sum_{(x,y)\in S} \sum_{k=1}^{p} (y_k - f_k)^2 + \gamma \sum_{i,j} w_{ij}^2$$

Modifier l'algorithme de rétropropagation du gradient en conséquence.

Exercices (2)

Fonction parité

On veut calculer avec un PMC la fonction parité pour n variables booléenne : il s'agit de renvoyer 1 si le nombre d'entrée à 1 est pair, 0 sinon.
Indice : faire en sorte que le i-ème neurone de la couche cachée retourne 1 si au moins i cellules de la rétine sont à 1.