

Entraînement de Réseaux de Neurones Artificiels

Nathan Trouvain

Inria Mnemosyne

nathan.trouvain@inria.fr

Sommaire

- 1 Optimisation de modèles paramétriques (rappels)
- 2 Algorithme de rétropropagation
- 3 Implémentation et outils

Sommaire

1 Optimisation de modèles paramétriques (rappels)

2 Algorithme de rétropropagation

3 Implémentation et outils

Modèles statistiques

Rappels

Définition

Un modèle statistique est un modèle mathématique qui met en application un certain nombre d'hypothèses statistiques concernant la génération d'un jeu de données ; il permet d'expliquer, de manière idéalisée, le processus de génération de ces données.

Modèles statistiques

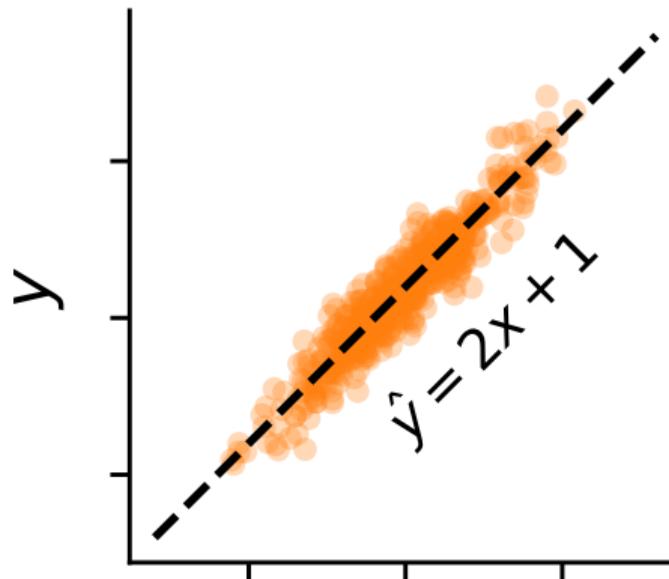
Notations

X	variables aléatoires observées
$\mathbf{x}_i = [x_{i,1}, x_{i,2}, \dots x_{i,p}]$	observation i (p variables)
$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_n]$	matrice de n observations ($n \times p$)
Idem pour $Y, \mathbf{y}_i, \mathbf{Y}$	variables/observations cibles
$\Theta = \{\theta_1, \theta_2, \dots \theta_m\}$	tous les paramètres d'un modèle

But : trouver f, Θ tels que $f_\Theta(\mathbf{x}) = \mathbf{y}$.
 f_Θ est notre modèle. Il implémente nos hypothèses (f, Θ) .

Modèles statistiques

Exemple



Hypothèse la plus probable pour expliquer Y à partir de X :

$$Y = w \cdot X + b$$

avec $\Theta = (w, b) = (2, 1)$ et f une fonction affine.

Apprentissage supervisé

Objectif général

Apprentissage supervisé

Processus permettant de déterminer automatiquement Θ à partir d'observations (\mathbf{X}, \mathbf{Y}), en supposant f .

Nécessaire

- **un modèle** → *quel type d'hypothèses je vais faire ?*
- **une fonction d'objectif** → *à quel point mon hypothèse est valide ?*
- **un algorithme permettant de minimiser/maximiser cette objectif**
→ *comment poser une meilleure hypothèse ?*

Apprentissage supervisé

Dans le cadre de ce cours

Apprentissage supervisé

Processus permettant de déterminer automatiquement Θ les plus probables à partir d'observations (\mathbf{X}, \mathbf{Y}) , en supposant f .

Nécessaire

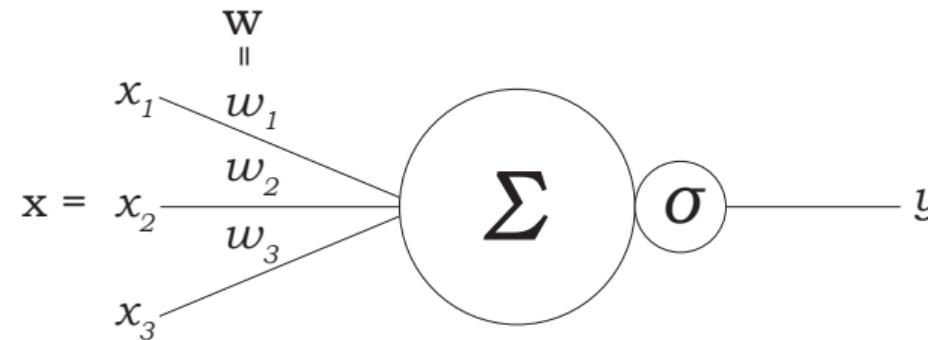
- **un modèle** → *Perceptron multicouche* (approximateurs universels)
- **une fonction d'objectif** → *Loss*
- **un algorithme permettant de minimiser/maximiser cette objectif**
→ *Descente de gradient*

Rappels : perceptron multicouche

Notations et définition

Perceptron simple avec $\theta = \{\mathbf{w}, b\}$ et fonction d'activation σ :

$$f_{\theta}(\mathbf{x}) = \sigma\left(\sum_i^p w_i x_i + b\right) = \sigma(\mathbf{w}^\top \cdot \mathbf{x} + b)$$

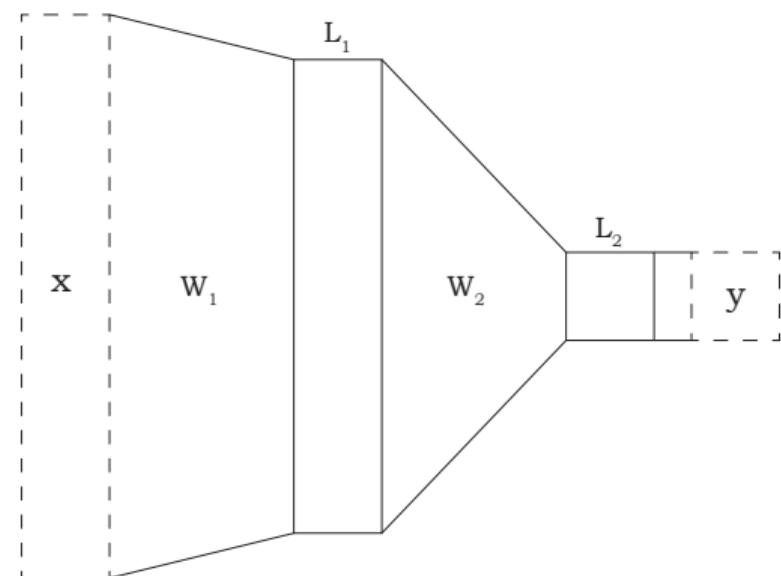


Rappels : perceptron multicouche

Notations et définition

Perceptron multicouche avec L couches
de perceptrons $(f_{\theta_i}^{(i)})_{i=0 \dots L}$ et
 $\Theta = \{\theta_1, \dots, \theta_L\}$:

$$\begin{aligned} F_{\Theta}(\mathbf{x}) &= f_{\theta_l}^{(l)}(f_{\theta_{l-1}}^{(l-1)}(\dots f_{\theta_1}^{(1)}(\mathbf{x}))) \\ &= f_{\theta_l}^{(l)} \circ f_{\theta_{l-1}}^{(l-1)} \circ \dots \circ f_{\theta_1}^{(1)}(\mathbf{x}) \end{aligned}$$



Rappels : *Loss*

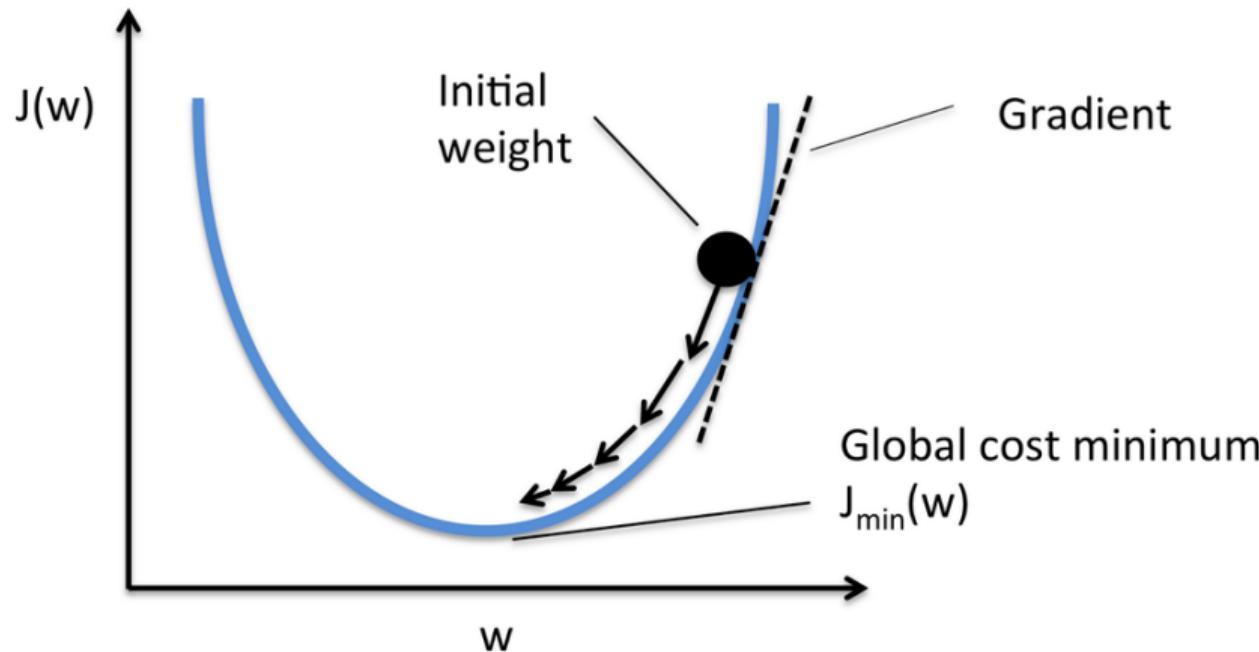
C'est quoi, en fait ?

But de l'optimisation

Minimiser **une fonction d'objectif (ou *loss*)** $\ell(\Theta)$;
 $\ell(\Theta)$ est **convexe** (a un minimum) et est **continue et différentiable** (on peut glisser jusqu'à ce minimum sans problème). **Le minimum de $\ell(\Theta)$ nous indique quels Θ répondent le mieux à la tâche fixée sur les données disponibles.**

Rappels : Loss

C'est quoi, en fait ?



Rappels : Loss

C'est quoi, en fait ?

Remarque (hors cadre du cours)

Souvent, en pratique, $\ell(\Theta) = E(\Theta) + R(\Theta)$

E est une **fonction d'erreur** (à quel distance je suis des prédictions souhaitées ?) et R est une **fonction de régularisation** (quel est le modèle le plus simple pour y parvenir ?).

Par exemple, $R(\Theta) = \|\Theta\|^2$ (régularisation ℓ_2 ou *ridge*).

Rappels : *Loss*

C'est quoi, en fait ?

Tâche	<i>Loss canonique</i>
Régression	<i>Mean Squared Error</i>
Classification (2 classes)	<i>Binary crossentropy</i>
Classification (k classes)	<i>Crossentropy</i>
...	...

Remarque

En pratique, la fonction d'objectif (ou *loss*, ou encore *cost*) utilisée pour l'apprentissage dans les réseaux de neurones est souvent issue de la *negative log-likelihood* (moins le log de la vraisemblance).

(Plus vraiment des) Rappels : *Likelihood*

C'est quoi, en fait ?

Exemple : objectif pour la classification

Soit X un ensemble d'images de chats ou de chiens, et Y un ensemble de labels valant 1 pour les chats et 0 pour les chiens. On souhaite estimer $p = P(Y = 1|X)$, la probabilité qu'une image soit un chat. On définit un modèle tel que $p = f_{\Theta}(x)$.

Objectif : avoir la meilleure estimation de p par f_{Θ} sachant n'importe quel x .

(Plus vraiment des) Rappels : *Likelihood*

C'est quoi, en fait ?

Exemple : objectif pour la classification

Objectif : avoir la meilleure estimation de p par f_{Θ} sachant n'importe quel \mathbf{x} .

$$P(Y = 1 \text{ (chat)} | X = \mathbf{x}) = p = f_{\Theta}(\mathbf{x})$$

$$P(Y = 0 \text{ (chien)} | X = \mathbf{x}) = 1 - p = 1 - f_{\Theta}(\mathbf{x})$$

$$\implies P(Y = k | X) = p^k(1 - p)^{k'} \text{ (loi de Bernoulli)}$$

(Plus vraiment des) Rappels : *Likelihood*

C'est quoi, en fait ?

On définit la **likelihood (vraisemblance)** comme la probabilité que les données observées soient issues du modèle f_Θ :

$$\begin{aligned}\mathcal{L}(\Theta) &= P(Y_1 = \mathbf{y}_1, \dots, Y_n = \mathbf{y}_n | X = \mathbf{x}_1, \dots, X = \mathbf{x}_n) \\ &\stackrel{iid}{=} \prod_i^n \underbrace{P(Y_i = \mathbf{y}_i | X = \mathbf{x}_i)}_{\text{dépend de } \Theta} \\ &= \prod_i^n f_\Theta(\mathbf{x}_i)^{\mathbf{y}_i} (1 - f_\Theta(\mathbf{x}_i))^{\mathbf{y}_i}\end{aligned}$$

(Plus vraiment des) Rappels : *Likelihood*

C'est quoi, en fait ?

Exemple : objectif pour la classification

On cherche donc **des valeurs de Θ qui maximisent la vraisemblance** :

$$\operatorname{argmax}_{\Theta} \mathcal{L}(\Theta)$$

Pour simplifier les calculs (et éviter de multiplier des probabilités très faibles ensembles), on définit **une loss équivalente à minimiser, la negative log-likelihood** $\ell(\Theta)$:

$$\operatorname{argmin}_{\Theta} \ell(\Theta) = \mathbb{E}[-\ln \mathcal{L}(\Theta)]$$

(Plus vraiment des) Rappels : *Likelihood*

C'est quoi, en fait ?

Exemple : objectif pour la classification

$$\begin{aligned}\ell(\Theta) &= \mathbb{E}[-\ln \mathcal{L}(\Theta)] \\ &= -\frac{1}{n} \sum_i^n \ln P(Y_i = \mathbf{y}_i | X = \mathbf{x}_i) \\ &= -\frac{1}{n} \sum_i^n \mathbf{y}_i \ln(f_\Theta(\mathbf{x}_i)) + (1 - \mathbf{y}_i) \ln(1 - f_\Theta(\mathbf{x}_i))\end{aligned}$$

On appelle cette *loss* une **entropie croisée binaire** (*binary crossentropy*).

(Plus vraiment des) Rappels : *Likelihood*

C'est quoi, en fait ?

Autres objectifs — Exercices

De la même manière, on peut démontrer que :

- la *loss* issue d'un objectif de maximisation de la vraisemblance sur une tâche de régression (avec cette fois-ci $P(Y|X) \sim \mathcal{N}(f_\Theta(X), \sigma^2)$) est **l'erreur quadratique moyenne** (*mean squared error* a.k.a. MSE).
- la *loss* issue du même objectif sur une tâche de classification multiclasse (plus de 2 classes) donne **l'entropie croisée** (*crossentropy*).

Rappels : descente de gradient

It's all coming together

Trouver le meilleur modèle f_{Θ} qui prédit la valeur de Y sachant X

⇒ Trouver le minimum de la *loss* $\ell(\Theta)$

⇒ Trouver un point Θ^* tel que $\frac{d\ell}{d\Theta}(\Theta^*) = 0$

Solution : descendre la pente de ℓ , donc sa dérivée $\frac{d\ell}{d\Theta}$, jusqu'à s'approcher suffisamment d'un Θ^* intéressant.

«video descente de gradient»

Problématique : *credit assignment problem*

Marvin Minsky, Seymour Papert, 1969 : prémisses de l'hiver de l'IA

(Nous sommes en 1969, la descente de gradient est encore un art mathématique obscur.)

Dans un modèle profond, **calculer la contribution de chaque paramètre θ_i à la réponse finale du modèle est non trivial.**

Exemple

Par exemple, dans le modèle :

$$y = wx + b$$

il est évident que w définit la contribution de la variable x au résultat final. La *loss* peut-être minimisée analytiquement. Mais dans un modèle profond, cette contribution n'est plus évidente :

$$y = f_2(w_2 f_1(w_1 x + b_1) + b_2)$$

Problématique : *credit assignment problem*

Marvin Minsky, Seymour Papert, 1969 : prémisses de l'hiver de l'IA

"In playing a complex game such as chess or checkers, or in writing a computer program, one has a definite success criterion – the game is won or lost. But in the course of play, each ultimate success (or failure) is associated with a vast number of internal decisions. If the run is successful, how can we assign credit for the success among the multitude of decisions ?"

— Minsky, 1963

Solution : la rétropropagation du gradient d'erreur

Seppo I. Linnainmaa, 1970 ; Paul Werbos, 1974 ; David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams, 1986

Calculer, pour tous les $\Theta = \{\theta_1, \dots, \theta_m\}$, le **gradient** (vecteur de dérivées partielles) de $\ell(\Theta)$ \Rightarrow **calculer implicitement la contribution de chaque paramètre** :

$$\nabla_{\Theta} \ell = \begin{bmatrix} \frac{\partial \ell}{\partial \theta_1} \\ \frac{\partial \ell}{\partial \theta_2} \\ \vdots \\ \frac{\partial \ell}{\partial \theta_m} \end{bmatrix}$$

Appliquer chaque dérivée partielle au θ_i correspondant (descente de gradient) :

$$\theta_i := \theta_i - \eta \nabla_{\Theta} \ell$$

Sommaire

1 Optimisation de modèles paramétriques (rappels)

2 Algorithme de rétropropagation

3 Implémentation et outils

Préliminaires

Notations

Soit un perceptron multicouche F_{Θ} , composé de L couches, avec $\Theta = \{\mathbf{w}_1, \mathbf{w}_2 \dots \mathbf{w}_L, b_1, b_2, \dots b_L\}$, l'ensemble des paramètres. Soit \mathbf{h}_l la sortie d'une couche l **avant activation** et \mathbf{a}_l **après activation** :

$$\mathbf{h}_l = \mathbf{w} \cdot \mathbf{h}_{l-1} + b$$

$$\mathbf{a}_l = \sigma(\mathbf{h}_l)$$

Remarque : Si $l = 1$ (entrée du réseau), alors $\mathbf{h}_{l-1} = \mathbf{x}$.

Préliminaires

Notations

Soit une fonction objectif $\ell(\Theta)$ quelconque, qu'on suppose continue et différentiable. On souhaite ici qu'elle calcule l'erreur commise entre une valeur souhaitée \mathbf{y} et une valeur prédite par la dernière couche du réseau $\hat{\mathbf{y}} = \mathbf{a}_L$:

$$\ell(\Theta) = err(\hat{\mathbf{y}}, \mathbf{y}) = err(\mathbf{a}_L, \mathbf{y})$$

Règle de chaînage

LA DIAPO LA PLUS IMPORTANTE DU COURS

En considérant toutes les fonctions impliquées comme **differentiables**, on peut écrire les dérivées partielles de l'erreur par rapport à chaque paramètre i de chaque couche l ($\theta_{l,i}$) que l'on note $\frac{\partial \ell}{\partial \theta_{l,i}}$.

Cette valeur est directement calculable par la **règle de chaînage des dérivées** :

$$(f \circ g)'(x) = f'(g(x)) \cdot g'(x) = \frac{df}{dg} \cdot \frac{dg}{dx}$$

$$\frac{\partial \ell}{\partial \theta_{l,i}} = \frac{\partial \ell}{\partial \mathbf{a}_L} \cdot \frac{\partial \mathbf{a}_L}{\partial \mathbf{h}_L} \cdots \frac{\partial \mathbf{a}_{l+1}}{\partial \mathbf{h}_{l+1}} \cdot \frac{\partial \mathbf{h}_{l+1}}{\partial \mathbf{a}_l} \cdot \frac{\partial \mathbf{a}_l}{\partial \mathbf{h}_l} \cdot \frac{\partial \mathbf{h}_l}{\partial \theta_{l,i}}$$

→ Comme passer une donnée dans le réseau, mais à **l'envers**.

Application

Apprentissage en trois étapes

1. *Forward pass*

Calculer tous les h_l et a_l couche par couche, à partir d'une donnée x soit
émettre une prédition avec le réseau.

Calculer $\ell(\Theta)$, l'erreur commise avec les paramètres actuels.

Application

Apprentissage en trois étapes

2. Backward pass

En partant de la valeur $\ell(\Theta)$ calculée, passer dans le réseau à l'envers pour calculer la contribution de chaque paramètre à cette valeur, soit **calculer la dérivée de l'erreur par rapport à chaque paramètres.**

On applique pour ce faire la règle de chaînage.

Application

Apprentissage en trois étapes

3. Descente de gradient

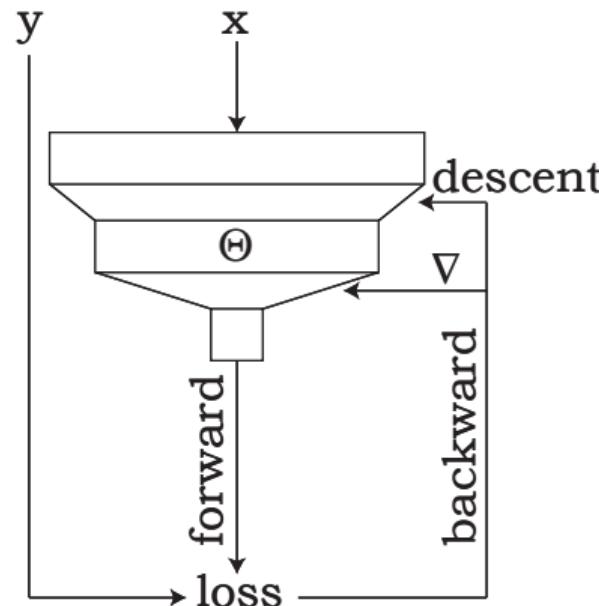
Pour terminer, on applique l'algorithme de descente de gradient pour mettre à jour les paramètres avec les gradients calculés pendant la *backward pass*.

Puis retour à 1.

Recommencer avec un nouveau point ou groupe de points de donnée x jusqu'à **convergence** → jusqu'à ce que $\ell(\Theta)$ ne se réduise plus à chaque mise à jour des Θ .

Application

Apprentissage en trois étapes



Exemple

Forward pass

On se propose d'effectuer une tâche de régression sur un jeu de données

$(\mathbf{X}, \mathbf{Y}) = (x_i, y_i)_{i=1\dots n}$ de taille n , avec $\dim x_i = 1 = \dim y_i$.

Soit un perceptron multicouche $\hat{y} = F(x)$ à 3 couches, chacune contenant un unique neurone, équipées d'une fonction d'activation sigmoïde $\sigma(x) = \frac{1}{1+e^{-x}}$, $\frac{d\sigma}{dx} = \sigma(x)(1 - \sigma(x))$ ou identité $Id(x) = x$ pour la dernière couche.

Couche $l = 1$

$$h_1 = w_1 x + b_1$$

$$a_1 = \sigma(h_1)$$

Couche $l = 2$

$$h_2 = w_2 a_1 + b_2$$

$$a_2 = \sigma(h_2)$$

Couche $l = 3$

$$h_3 = w_3 a_2 + b_3$$

$$\hat{y} = a_3 = Id(h_3)$$

Exemple

Forward pass

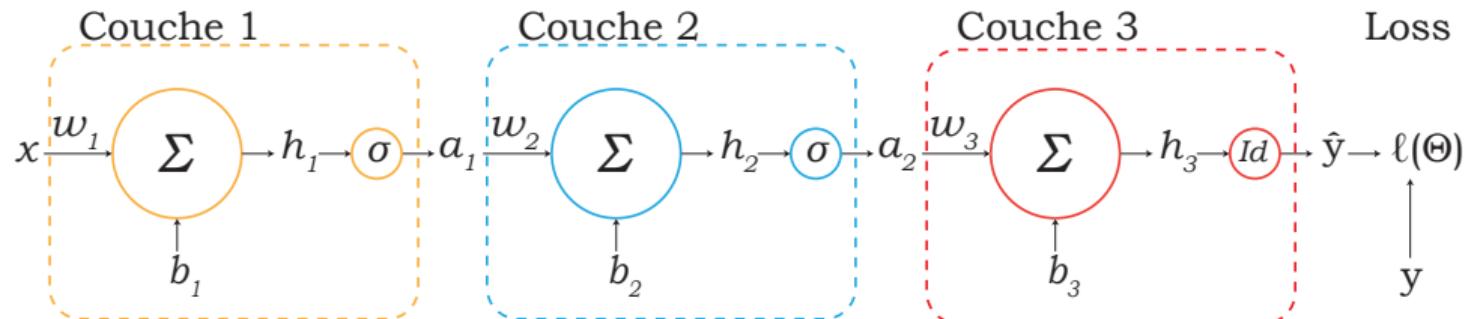
Pour un point (ou un groupe de points) de données x , on calcule les h_l et a_l .
A la fin de la *forward pass*, on calcule une dernière fonction, la *loss*, dépendante de la sortie du réseau et donc de ses paramètres Θ .

$$\ell(\Theta) = \frac{1}{2} \sum_i (\hat{y}_i - y_i)^2 \quad \text{pseudo-MSE}$$

Il s'agit d'une estimation de la distance entre \hat{y} et la valeur attendue y .

Exemple

Forward pass



Exemple

Backward pass

On commence par calculer la dérivée de la *loss* par rapport à l'activité du dernier neurone du réseau h_3 . La fonction d'activation est intégrée à ce calcul par simplicité.

$$\begin{aligned}\frac{\partial \ell}{\partial h_3} &= \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{d\hat{y}}{dh_3} \\ &= \frac{\partial 1/2(\hat{y} - y)^2}{\partial \hat{y}} \cdot \frac{dh_3}{dh_3} \\ &= (\hat{y} - y) \cdot 1 \\ &= \varepsilon_3\end{aligned}$$

Ce terme, ε_3 , représente l'erreur dans le contexte de la dernière couche, ici la couche 3.

Exemple

Backward pass – couche 3

On applique la règle de chaînage.

Contribution de w_3 à l'erreur :

$$\begin{aligned}\frac{\partial \ell}{\partial w_3} &= \underbrace{\frac{\partial \ell}{\partial \hat{y}}}_{\varepsilon_3} \cdot \frac{\partial \hat{y}}{\partial h_3} \cdot \frac{\partial h_3}{\partial w_3} \\ &= \varepsilon_3 \cdot \frac{\partial h_3}{\partial w_3} = \varepsilon_3 \cdot \frac{\partial(w_3 a_2 + b_3)}{\partial w_3} \\ &= \varepsilon_3 \cdot a_2\end{aligned}$$

Contribution de b_3 à l'erreur :

$$\begin{aligned}\frac{\partial \ell}{\partial b_3} &= \underbrace{\frac{\partial \ell}{\partial \hat{y}}}_{\varepsilon_3} \cdot \frac{\partial \hat{y}}{\partial h_3} \cdot \frac{\partial h_3}{\partial b_3} \\ &= \varepsilon_3 \cdot \frac{\partial h_3}{\partial b_3} = \varepsilon_3 \cdot \frac{\partial(w_3 a_2 + b_3)}{\partial b_3} \\ &= \varepsilon_3 \cdot 1\end{aligned}$$

Exemple

Backward pass – couche 3 vers 2

On passe l'erreur dans la couche 3 pour la mettre à portée de la couche 2. On cherche donc à calculer la contribution des sorties de la couche 2 (h_2) à l'erreur actuelle (ε_2) :

$$\begin{aligned}\frac{\partial \ell}{\partial h_2} &= \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_2} \\ &= \varepsilon_3 \cdot \frac{\partial h_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_2} = \varepsilon_3 \cdot \frac{\partial(w_3 a_2 + b_3)}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_2} \\ &= \varepsilon_3 \cdot w_3 \cdot a_2(1 - a_2) \\ &= \varepsilon_2\end{aligned}$$

Exemple

Backward pass – couche 2

Contribution de w_2 à l'erreur :

$$\begin{aligned}\frac{\partial \ell}{\partial w_2} &= \underbrace{\frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_2}}_{\varepsilon_2} \\ &= \varepsilon_2 \cdot \frac{\partial h_2}{\partial w_2} \\ &= \varepsilon_2 \cdot a_1\end{aligned}$$

Contribution de b_2 à l'erreur :

$$\begin{aligned}\frac{\partial \ell}{\partial b_2} &= \underbrace{\frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial b_2}}_{\varepsilon_2} \\ &= \varepsilon_2 \cdot \frac{\partial h_2}{\partial b_2} \\ &= \varepsilon_2 \cdot 1\end{aligned}$$

Exemple

Backward pass – couche 2 vers 1

On passe l'erreur dans la couche 2 pour atteindre la couche 1 (contribution de h_1) :

$$\begin{aligned}\frac{\partial \ell}{\partial h_1} &= \underbrace{\frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_1} \cdot \frac{\partial a_i}{\partial h_1}}_{\varepsilon_2} \\ &= \varepsilon_2 \cdot \frac{\partial h_2}{\partial a_1} \cdot \frac{\partial a_i}{\partial h_1} \\ &= \varepsilon_2 \cdot w_1 \cdot a_1(1 - a_1) \\ &= \varepsilon_1\end{aligned}$$

Exemple

Backward pass – couche 1 et fin

Encore une (dernière) fois :

Contribution de w_1 à l'erreur :

$$\begin{aligned}\frac{\partial \ell}{\partial w_1} &= \varepsilon_1 \cdot \frac{\partial h_1}{\partial w_1} \\ &= \varepsilon_1 \cdot x\end{aligned}$$

Contribution de b_1 à l'erreur :

$$\begin{aligned}\frac{\partial \ell}{\partial b_1} &= \varepsilon_1 \cdot \frac{\partial h_1}{\partial b_1} \\ &= \varepsilon_1 \cdot 1\end{aligned}$$

Backward pass – formule générique

La **backward pass** consiste donc premièrement à recalculer le ε_l de la couche courante. Dans le cas où chaque couche possède m_l neurones (donc $\mathbf{w} \in \mathbb{R}^{m_l \times m_{l+1}}$ et $\dim \mathbf{a}_l = \dim \mathbf{h}_l = m_l$) :

$$\underbrace{\nabla_{\mathbf{h}_l} \ell}_{\text{gradient}} = \varepsilon_l^\top = \mathbf{w}_{l+1} \cdot \varepsilon_{l+1} \odot \left(\frac{d\mathbf{a}_l}{d\mathbf{h}_l} \right)^\top$$

On peut ensuite calculer la contribution des paramètres :

$$\frac{\partial \ell}{\partial \mathbf{w}_l} = \varepsilon_l \cdot \mathbf{a}_l^\top = \nabla_{\mathbf{w}_l} \ell$$

$$\frac{\partial \ell}{\partial b_l} = \varepsilon_l \cdot \mathbf{1} = \sum_i \varepsilon_{l,i}$$

Descente de gradient

Rappel

Les gradients sont ensuite utilisés dans le cadre de l'algorithme de descente de gradient :

$$\mathbf{w}_l := \mathbf{w}_l - \eta \nabla_{\mathbf{w}_l} \ell$$

$$b_l := b_l - \eta \frac{\partial \ell}{\partial b_l}$$

avec η un taux d'apprentissage fixé.

Remarque : il existe de nombreux autres algorithmes de descente de gradient, comme *Adam* ou *RMSProp*.

Descente de gradient stochastique

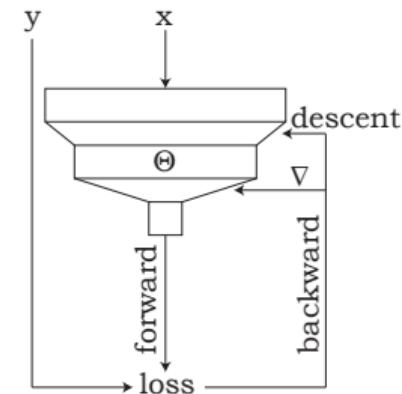
Definition

Boucle d'apprentissage (*Training loop*)

Pour que l'algorithme de descente du gradient converge vers des valeurs Θ qui minimisent la *loss*, on doit itérer *forward* et *backward pass* sur tout le jeu de données, plusieurs fois.

On appelle une ***epoch*** une itération complète du jeu de données.

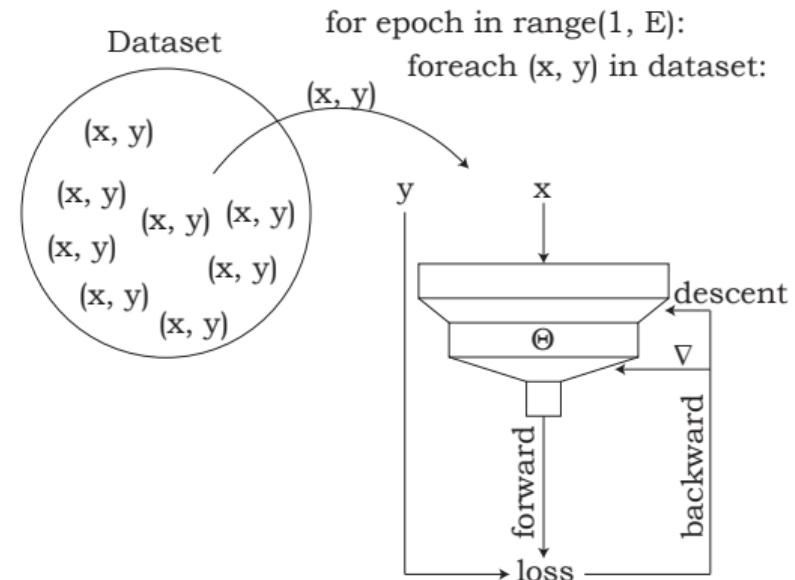
```
for epoch in range(1, E):  
    foreach (x, y) in dataset:
```



Descente de gradient stochastique

Définition

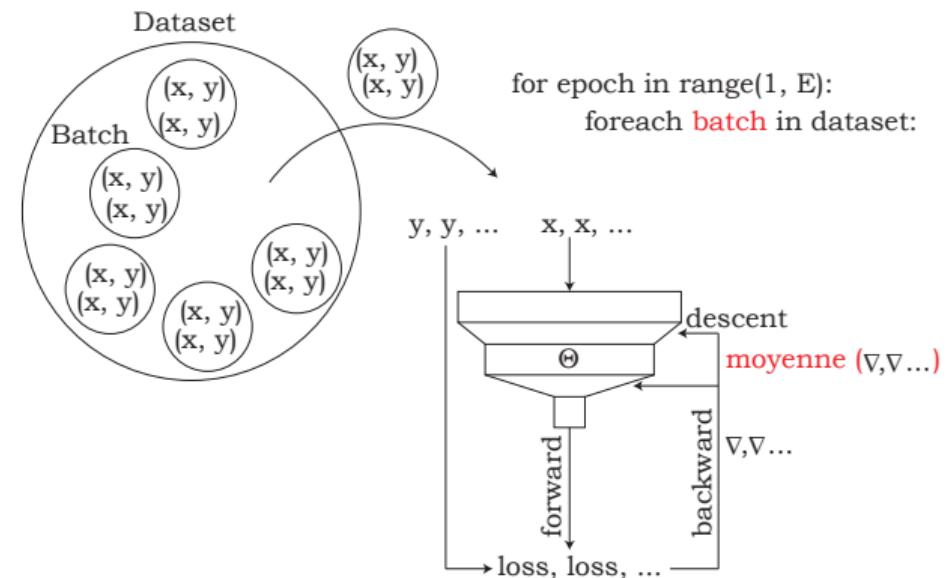
On parle de descente de gradient stochastique lorsque, à chaque itération, le gradient est évalué sur une donnée (x_i, y_i) tirée au hasard (sans replacement) dans le jeu de données.



Descente de gradient stochastique

Descente par *batch*

En pratique, pour accélérer les calculs et stabiliser l'apprentissage, on calcule la moyenne des gradients sur un lot (*batch*) aléatoire de points $(x_i, y_i)_{i=1 \dots k}$, avec k la taille d'un *batch*.



Procédure d'entraînement

Ne pas oublier ses bonnes manières

Évidemment, ces techniques doivent être appliquées dans le cadre de travail du *Machine Learning* étudié précédemment :

- on définit des jeux de données de tests, de validation et d'entraînement pour mettre à jour le modèle et évaluer ses performances ;
- les données sont formatées correctement (données numériques, *one-hot encoding*...) ;
- plusieurs hypothèses (architectures de modèles) sont envisagées et comparées ;
- etc.

Sommaire

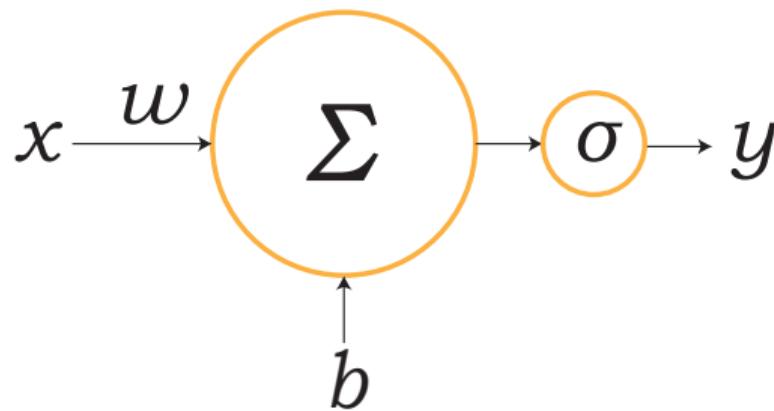
1 Optimisation de modèles paramétriques (rappels)

2 Algorithme de rétropropagation

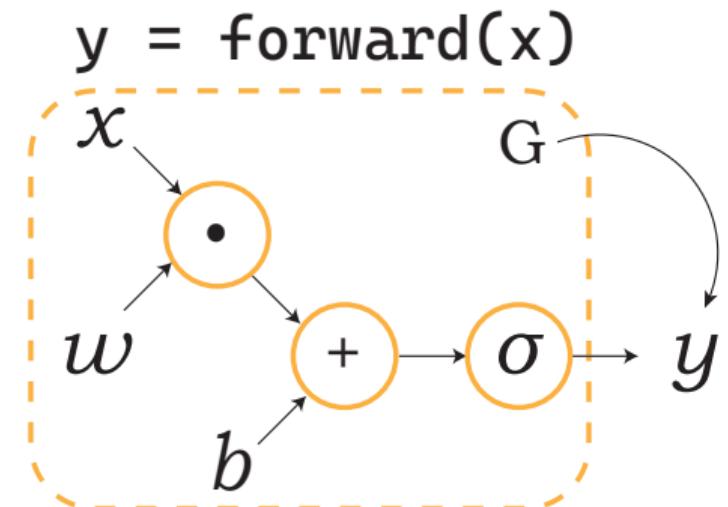
3 Implémentation et outils

Graphes computationnels

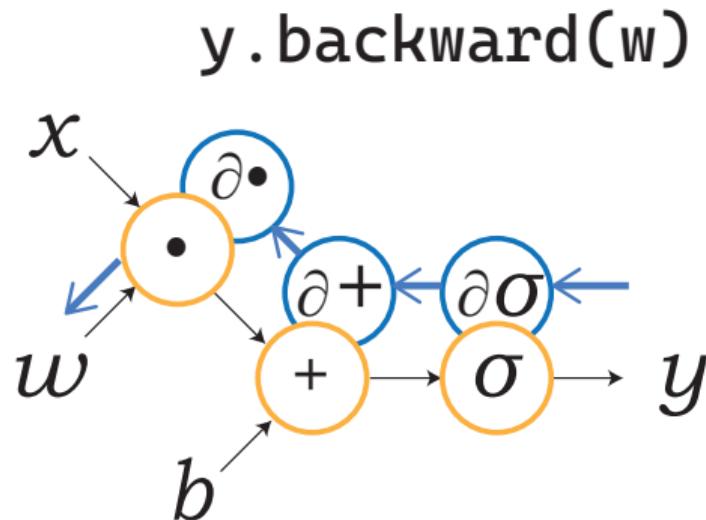
Schéma du circuit d'un perceptron



Graphe computationnel correspondant



Différentiation automatique (*Autodiff*)



Permet de propager le gradient grâce à la règle de chaînage **pour n'importe quelle opération** (opérateurs usuels, fonctions d'activation, et même du code Python grâce à certains outils). A chaque opération est attachée une opération *backward* qui calcule le gradient par chaînage.

Différentiation automatique (*Autodiff*)

Implémentation (*pytorch*)

```
class LinearFunction(Function):
    @staticmethod
    # ctx is the first argument to forward
    def forward(ctx, input, weight, bias=None):
        # The forward pass can use ctx.
        ctx.save_for_backward(input, weight, bias)
        output = input.mm(weight.t())
        if bias is not None:
            output += bias.unsqueeze(0).expand_as(output)
        return output

    @staticmethod
    def backward(ctx, grad_output):
        input, weight, bias = ctx.saved_tensors
        grad_input = grad_weight = grad_bias = None

        if ctx.needs_input_grad[0]:
            grad_input = grad_output.mm(weight)
        if ctx.needs_input_grad[1]:
            grad_weight = grad_output.t().mm(input)
        if bias is not None and ctx.needs_input_grad[2]:
            grad_bias = grad_output.sum(0)

        return grad_input, grad_weight, grad_bias
```

Logiciels de *deep learning*

Une question de goût



PyTorch



mxnet



PaddlePaddle



Logiciels de *deep learning*

Celui qui ne nous laissera jamais tomber



(Petite démonstration?)

Au prochain épisode

- TD6 – Entraînement de réseaux de neurones *from scratch*

Lectures complémentaires

- Minsky, M. (1961). Steps Toward Artificial Intelligence. *Proc. IRE*
- Rumelhart, D., Hinton, G. Williams, R. (1986). Learning representations by back-propagating errors. *Nature*
- Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. *MIT press.* (<https://www.deeplearningbook.org/>)
- Bishop, C. (2006). Pattern recognition and machine learning. *Springer* (online)
- Olah, C., colah's blog, <https://colah.github.io/>