Hiver 2018

Techniques d’apprentissage

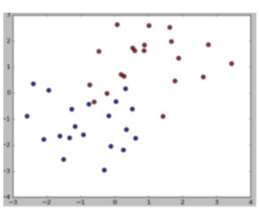
IFT 603

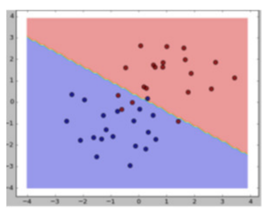
Classification linéaire

Par
Pierre-Marc Jodoin
/
Hugo Larochelle

Classification supervisée (illustrée)

Entraînement





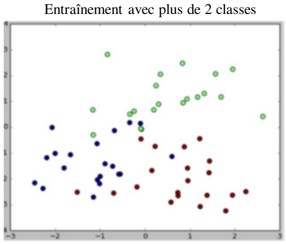
Soient des données de 2 classes ● et ●
(ici dans un espace 2D)

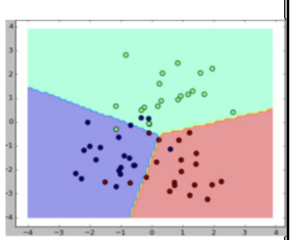
Le but est de trouver une function $y(\vec{x})$ telle que

$y(●) =$ class 1
$y(●) =$ class 2

Classification supervisée (illustrée)

Entraînement avec plus de 2 classes





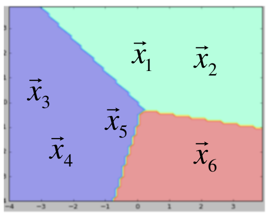
Soient des données de 3 classes ●, ○ et ●
(ici dans un espace 2D)

Le but est de trouver une function $y(\vec{x})$ telle que

$y(●) =$ classe 1
$y(●) =$ classe 2
$y(○) =$ classe 3

Classification supervisée (illustrée)

Une fois l'entraînement terminé, nous disposons d'une function $y(\vec{x})$ pouvant convertir n'importe quel point 2D $\vec{x}$ en une **étiquette de classe**.



$y(\vec{x}_1)$ => classe
$y(\vec{x}_2)$ => classe
$y(\vec{x}_3)$ => classe
$y(\vec{x}_4)$ => classe
$y(\vec{x}_5)$ => classe
$y(\vec{x}_6)$ => classe

Notation

Ensemble d'entraînement: $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), ..., (\vec{x}_N, t_N)\}$

$\vec{x}_n \in \Re^d$ vecteur de données du n-ème élement
$t_n \in \{c_1, c_2, ..., c_K\}$ etiquette de classe du i-ème élément

Fonctions: avec *D*, on doit *apprendre* une **fonction de classification**

$$y : \Re^d \rightarrow \{c_1, c_2, ..., c_k\}$$

qui nous informe à quelle classe appartient le vecteur $\vec{x}$.

Au menu : 5 méthodes

Régression
Modèles génératifs
Discriminant de Fisher

Émettent **l'hypothèse** que les données sont **gaussiennes**

Solution de type « ***closed form*** » (inversion de matrice)

Perceptron
Régression logistique

Aucune hypothèse quant à la distribution des données

Solution obtenue grâce à une **descente de gradient**.

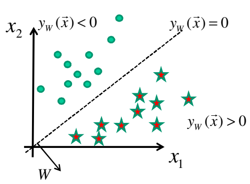
Introduction à la classification linéaire

Au tableau !!!

Séparation linéaire

(2D et 2 classes)

$$\underbrace{y_W(\vec{x}) = w_0}_{\text{biais}} + \underbrace{w_1 x_1 + w_2 x_1}_{\text{poids}}$$
$$= w_0 + W^{\mathrm{T}}\vec{x}$$
$$= W'^{\mathrm{T}}\vec{x}'$$



$y_W(\vec{x}) = W^{\mathrm{T}}\vec{x}$ Par simplicité

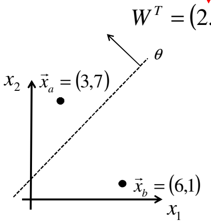
2 grands **advantages**. Une fois l'entraînement terminé,

1. Plus besoin de données d'entraînement
2. Classification est très rapide (**produit scalaire** entre 2 vecteurs)

Séparation linéaire

Exemple jouet

$$W^T = (\underbrace{2.2}_{\text{biais}}, \underbrace{-5.5, 4.4}_{\text{poids}})$$



$\vec{x}_a$ est en FACE du plan

$$y_W(\vec{x}_a) = W^T\vec{x}_a = (2.2, -5.5, 4.4)\begin{bmatrix}1\\3\\7\end{bmatrix} = 16.5$$

$$y_W(\vec{x}_b) = W^T\vec{x}_b = (2.2, -5.5, 4.4)\begin{bmatrix}1\\6\\1\end{bmatrix} = -26.4$$

$\vec{x}_b$ est DERRIÈRE le plan

Régression
Modèles génératifs
Discriminant de Fisher

Émettent l'hypothèque que les données sont **gaussiennes**

Solution de type « ***closed form*** » (inversion de matrice)

Perceptron
Régression logistique

Émettent **aucune hypothèse** quant à la distribution des données

Solution obtenue grâce à une **descente de gradient**.

Régression par les moindres carrés
(section 4.1.3, Bishop)

Régression par les moindres carrés
Cas 2 classes

On peut **classifier des données** en utilisant une approche de **régression** comme celle vue au chapitre précédent.

- On pourrait **prédire directement** la valeur de la cible (*t=1.0* vs *t=-1.0*)
- Si $y_W(\vec{x}) \geq 0$ on classifie dans *Classe1* sinon dans *Classe2*

Régression par moindres carrés

RAPPEL

On a vu qu'on peut utiliser une fonction d'erreur **par moindres carrés**

Maximum de vraisemblance

$$W = \arg\min_W \underbrace{\sum_{n=1}^{N} \frac{(t_n - y_W(\vec{x}_n))^2}{2}}_{E_D(W)}$$

Maximum *a posteriori*

$$W = \arg\min_W \underbrace{\sum_{n=1}^{N} \frac{(t_n - y_W(\vec{x}_n))^2}{2} + \lambda \frac{W^{\mathrm{T}}W}{2}}_{E_D(W)}$$

On peut prendre la même approche pour la classification

Régression par les moindres carrés

Cas 2 classes

RAPPEL

Maximum de vraisemblance

$$W = \arg\min_W \sum_{n=1}^{N} \frac{(t_n - y_W(\vec{x}_n))^2}{2}$$

$$\boxed{W_{\mathrm{MV}} = \left(X^{\mathrm{T}}X\right)^{-1} X^{\mathrm{T}}T}$$

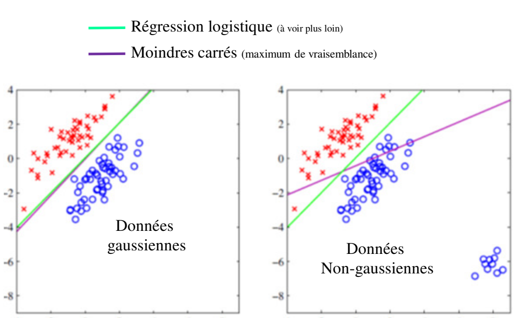
Maximum *a posteriori*

$$W = \arg\min_W \sum_{n=1}^{N} \frac{(t_n - y_W(\vec{x}_n))^2}{2} + \lambda \frac{W^{\mathrm{T}}W}{2}$$

$$\boxed{W_{\mathrm{MAP}} = \left(X^{\mathrm{T}}X + \lambda I\right)^{-1} X^{\mathrm{T}}T}$$

Ces fonctions de coût s'appuient sur l'hypothèse de données gaussiennes

Régression par moindres carrés



Régression par les moindres carrés

Cas K>2 classes

On va traiter le cas K classes comme une **régression multiple**

- **Cible :** vecteur à K dim. indiquant a quelle classe appartient l'entrée
- **Exemple** : Pour K=5 classes et un entrée associée à la classe 2

$$t_n = \begin{pmatrix} -1 & 1 & -1 & -1 & -1 \end{pmatrix}^{\mathbf{T}}$$

- **Classification**: On classifie dans la classe k une donnée dont la valeur de $y_{W,k}(\vec{x})$ est la plus élevée.

Régression par les moindres carrés

Cas K>2 classes

Le modèle doit maintenant **prédire un vecteur**

$$y_{\mathrm{W}}(\vec{x}) = \mathrm{W}^{\mathrm{T}}\vec{x}$$

où $\mathbf{W}^{\mathbf{T}}$ est une matrice ***K* x *d***

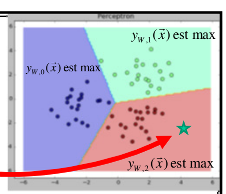
Chaque ligne de $\mathbf{W}^{\mathrm{T}}$ peut être vue comme un vecteur W_k du modèle $y_{W_k}(\vec{x}) = W_k^T\vec{x}$ pour la k^e cible

17

Cas K=3 classes

Example

★ (1.1, -2.0)



$$y_W(\vec{x}) = \mathrm{W}^{\mathrm{T}}\vec{x} = \begin{bmatrix} -.2 & .05 & -.36 \\ -.4 & .41 & .24 \\ -.6 & -.49 & .40 \end{bmatrix} \begin{bmatrix} 1 \\ -2.0 \\ 1.1 \end{bmatrix} = \begin{bmatrix} -.7 \\ -1 \\ .8 \end{bmatrix} \begin{array}{l} \text{Classe 0} \\ \text{Classe 1} \\ \text{Classe 2} \end{array}$$

Régression
Modèles génératifs
Discriminant de Fisher

- Émettent l'hypothèque que les données sont **gaussiennes**
- Solution de type « ***closed form*** » (inversion de matrice)

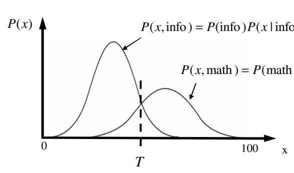
Perceptron
Régression logistique

- Émettent aucune hypothèse quant à la distribution des données
- Solution obtenue grâce à une **descente de gradient**.

Modèles probabilistes génératifs
(section 4.2, Bishop)

Prenons le cas 1D, 2 Classes

Ex: examen de mathématique avec étudiants en math et en informatique



T est le seuil qui **minimise l'erreur de classification**

$$P(\text{info})P(x=T\,|\,\text{info}) = P(\text{math})P(x=T\,|\,\text{math})$$

$$\boxed{P(\text{info})P(x\,|\,\text{info}) \underset{\text{math}}{\overset{\text{info}}{\gtrless}} P(\text{math})P(x\,|\,\text{math})}$$

21

$$P(\text{info})P(x \mid \text{info}) \underset{\text{math}}{\overset{\text{info}}{\gtrless}} P(\text{math})P(x \mid \text{math})$$

est équivalent à un **maximum a posteriori**

$$\begin{aligned} t &= \arg\max_t P(\underbrace{t}_{\text{Inconnue}} \mid \underbrace{x}_{\text{Connue}}) \quad \text{où } t \in \{\text{math}, \text{info}\} \\ &= (\cdots) \\ &= \arg\max_t P(t)P(x \mid t) \end{aligned}$$

Prenons le cas 1D, 2 Classes

Ex: examen de math avec étudiant en math et en informatique

$$P(\text{info})P(x \mid \text{info}) \underset{\text{math}}{\overset{\text{info}}{\gtrless}} P(\text{math})P(x \mid \text{math})$$

Si on suppose que la **vraisemblance** de chaque classe est **gaussienne**:

$$P(x \mid \text{info}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{info}}} \exp\left(-\frac{(x-\mu_{\text{info}})^2}{2\sigma^2_{\text{info}}}\right)$$

$$P(x \mid \text{math}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{math}}} \exp\left(-\frac{(x-\mu_{\text{math}})^2}{2\sigma^2_{\text{math}}}\right)$$

où

μ_{math} : moyenne des étudiants de math.

σ_{math} : écart-type des étudiants de math.

Modèle probabiliste génératif

Algorithme du seuil « optimal »

$$\mu_{\text{info}} = \frac{1}{N_{\text{info}}} \sum_{t_n=\text{info}} x_n, \quad \mu_{\text{math}} = \frac{1}{N_{\text{math}}} \sum_{t_n=\text{math}} x_n$$

$$\sigma^2_{\text{info}} = \frac{1}{N_{\text{info}}} \sum_{t_n=\text{info}} (x_n - \mu_{\text{info}})^2, \quad \sigma^2_{\text{math}} = \frac{1}{N_{\text{math}}} \sum_{t_n=\text{math}} (x_n - \mu_{\text{math}})^2$$

$$P(\text{math}) = \frac{N_{\text{math}}}{N_{\text{info}} + N_{\text{math}}}, \quad P(\text{info}) = \frac{N_{\text{info}}}{N_{\text{info}} + N_{\text{math}}}$$

POUR CHAQUE note x FAIRE

$$P_i = \frac{P(\text{info})}{\sqrt{2\pi}\sigma_{\text{info}}} \exp\left(-\frac{(x-\mu_{\text{info}})^2}{2\sigma^2_{\text{info}}}\right)$$

$$P_m = \frac{P(\text{math})}{\sqrt{2\pi}\sigma_{\text{math}}} \exp\left(-\frac{(x-\mu_{\text{math}})^2}{2\sigma^2_{\text{math}}}\right)$$

SI $P_i > P_m$ ALORS

$t = 1$ /* étudiant « info » */

SINON

$t = 0$ /* étudiant « math » */

Modèle probabiliste génératif

Classificateur quadratique, cas 1D, 2 Classes

$$P(\text{info})P(x \mid \text{info}) = P(\text{math})P(x \mid \text{math})$$

$$\frac{P(\text{info})}{\sqrt{2\pi}\sigma_{\text{info}}}\exp\left(-\frac{(x-\mu_{\text{info}})^2}{2\sigma^2_{\text{info}}}\right) = \frac{P(\text{math})}{\sqrt{2\pi}\sigma_{\text{math}}}\exp\left(-\frac{(x-\mu_{\text{math}})^2}{2\sigma^2_{\text{math}}}\right)$$

On peut facilement démontrer que

$$\boxed{y_W(x) = w_2 x^2 + w_1 x + w_0 = 0}$$

$$w_2 = \frac{\sigma^2_{\text{math}} - \sigma^2_{\text{info}}}{2}$$

$$w_1 = \mu_{\text{math}}\sigma^2_{\text{info}} - \mu_{\text{info}}\sigma^2_{\text{math}}$$

$$w_0 = \frac{\mu^2_{\text{info}}\sigma^2_{\text{math}}}{2} - \frac{\mu^2_{\text{math}}\sigma^2_{\text{info}}}{2} - \sigma^2_{\text{info}}\sigma^2_{\text{math}}\ln\left(\frac{\sigma_{\text{math}}P(\text{info})}{\sigma_{\text{info}}P(\text{math})}\right)$$

25

Pour un **maximum de vraisemblance**

Connue **Inconnue**

$$t = \arg\max_t P(x \mid t) \quad \text{où } t \in \{\text{math}, \text{info}\}$$

On obtient que

$$P(x \mid \text{info}) \underset{\text{math}}{\overset{\text{info}}{\gtrless}} P(x \mid \text{math})$$

Et que

$$\boxed{y_W(x) = w_2 x^2 + w_1 x + w_0 = 0}$$

$$w_2 = \frac{\sigma^2_{\text{math}} - \sigma^2_{\text{info}}}{2}$$

$$w_1 = \mu_{\text{math}}\sigma^2_{\text{info}} - \mu_{\text{info}}\sigma^2_{\text{math}}$$

$$w_0 = \frac{\mu^2_{\text{info}}\sigma^2_{\text{math}}}{2} - \frac{\mu^2_{\text{math}}\sigma^2_{\text{info}}}{2} - \sigma^2_{\text{info}}\sigma^2_{\text{math}}\ln\left(\frac{\sigma_{\text{math}}}{\sigma_{\text{info}}}\right)$$

Modèle probabiliste génératif

Classificateur linéaire, cas 1D, 2 Classes

Si on suppose que $\sigma_{\text{info}} = \sigma_{\text{math}} = \sigma$

$$P(\text{info})P(x \mid \text{info}) = P(\text{math})P(x \mid \text{math})$$

$$\frac{P(\text{info})}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu_{\text{info}})^2}{2\sigma^2}\right) = \frac{P(\text{math})}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu_{\text{math}})^2}{2\sigma^2}\right)$$

$$\boxed{y_W(x) = w_1 x + w_0 = 0}$$

$$w_1 = \frac{(\mu_{\text{math}} - \mu_{\text{info}})}{\sigma^2}$$

$$w_0 = \frac{\mu^2_{\text{info}}}{2\sigma^2} - \frac{\mu^2_{\text{math}}}{2\sigma^2} - \ln\left(\frac{P(\text{info})}{P(\text{math})}\right)$$

27

Modèle probabiliste génératif

Classificateur linéaire, **cas d-D, 2 Classes**

$$\boxed{y_W(\vec{x}) = \mathrm{W}^{\mathrm{T}}\vec{x} + w_0 = 0}$$

$$\mathrm{W} = \Sigma^{-1}(\vec{\mu}_1 - \vec{\mu}_2)$$

$$w_0 = \frac{\vec{\mu}_2^{\mathrm{T}}\Sigma^{-1}\vec{\mu}_2}{2} - \frac{\vec{\mu}_1^{\mathrm{T}}\Sigma^{-1}\vec{\mu}_1}{2} - \ln\left(\frac{P(\mathrm{C}_1)}{P(\mathrm{C}_2)}\right)$$

28

Note

Tel que mentionné au chapitre 4.2.2, lorsque les 2 classes n'ont pas la même variance-covariance, on peut utiliser le modèle linéaire mais avec la matrice

$$\Sigma = P(\mathrm{C}_1)\Sigma_1 + P(\mathrm{C}_2)\Sigma_2$$

Modèle probabiliste génératif

Classificateur linéaire, cas d-D, K Classes

On peut généraliser au cas à **plusieurs classes**

- Voir fin des sections 4.2 et 4.2.1

30

Régression
Modèles génératifs
Discriminant de Fisher

- Émettent l'hypothèque que les données sont **gaussiennes**
- Solution de type « ***closed form*** » (inversion de matrice)

Perceptron
Régression logistique

- Émettent **aucune hypothèse** quant à la distribution des données
- Solution obtenue grâce à une **descente de gradient**.

Discriminant linéaire de Fisher

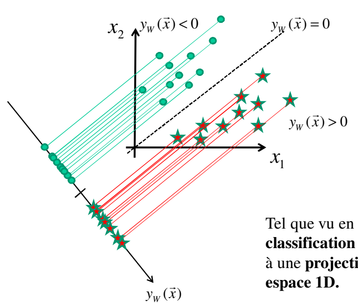
(section 4.1.4, Bishop)

Classification linéaire = Projection 1D

(2D et 2 classes)

$$y_W(\vec{x}) = w_0 + W^T \vec{x}$$

biais **poids**

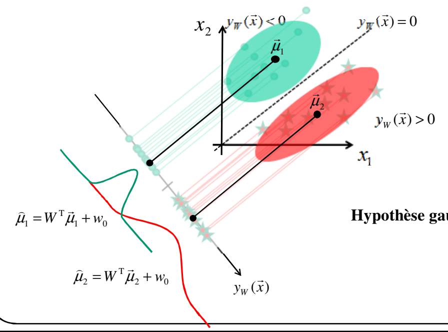


Tel que vu en classe, une **classification linéaire** revient à une **projection vers un espace 1D.**

Classification linéaire = Projection 1D

(2D et 2 classes)

$y_W(\vec{x}) = w_0 + W^{\mathrm{T}}\vec{x}$

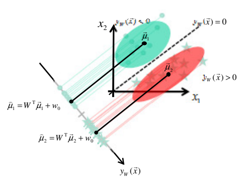


Hypothèse gaussienne…

Classification linéaire = Projection 1D

(2D et 2 classes)

$y_W(\vec{x}) = w_0 + W^{\mathrm{T}}\vec{x}$



Intuitivement, une bonne solution $y_W(\vec{x})$ en est une pour laquelle la **distance entre les moyennes projetées est grande.**

$$
\begin{aligned}
W &= \arg\max_W (\tilde{\mu}_1 - \tilde{\mu}_2)^2 \\
&= \arg\max_W \left(W^{\mathrm{T}}\vec{\mu}_1 + w_0 - W^{\mathrm{T}}\vec{\mu}_2 - w_0\right)^2 \\
&= \arg\max_W \left(W^{\mathrm{T}}(\vec{\mu}_1 - \vec{\mu}_2)\right)^2
\end{aligned}
$$

Classification linéaire = Projection 1D

(2D et 2 classes)

$y_W(\vec{x}) = W^{\mathrm{T}}\vec{x} + w_0$

$$W = \arg\max_W \left(W^{\mathrm{T}}(\vec{\mu}_1 - \vec{\mu}_2)\right)^2$$

Ce **problème est mal posé** car il suffit d'augmenter **W** infiniment pour maximiser cette fonction.

Classification linéaire = Projection 1D

(2D et 2 classes)

$y_W(\vec{x}) = W^{\mathrm{T}}\vec{x} + w_0$

$$W = \arg\max_W \left(W^{\mathrm{T}}(\vec{\mu}_1 - \vec{\mu}_2)\right)^2$$

Par contre si on impose que la **norme de W = 1**

on obtient que

$$W \propto (\vec{\mu}_1 - \vec{\mu}_2)$$

Discriminant linéaire

Une fois **W** calculé, il faut trouver le biais w_0

- Un choix fréquent lorsque les classes sont balancées

$$w_0 = -\frac{W^{\mathrm{T}}\vec{\mu}_1 + W^{\mathrm{T}}\vec{\mu}_2}{2}$$

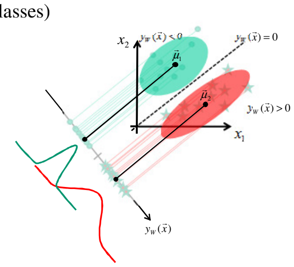
- Sinon

$$w_0 = -W^{\mathrm{T}}\left(\frac{N_1}{N_1+N_2}\vec{\mu}_1 + \frac{N_1}{N_1+N_2}\vec{\mu}_2\right)$$

où N1 et N2 sont le nombre d'éléments dans chaque classe.

Discriminant linéaire

(2D et 2 classes)



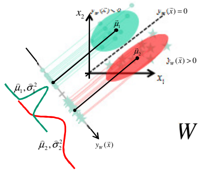
Les 2 gaussiennes projetées:

$$\hat{\mu}_1 = W^{\mathrm{T}}\vec{\mu}_1 + w_0 \qquad \hat{\mu}_2 = W^{\mathrm{T}}\vec{\mu}_2 + w_0$$

$$\hat{\sigma}_1^2 = W^{\mathrm{T}}\Sigma_1 W \qquad \hat{\sigma}_2^2 = W^{\mathrm{T}}\Sigma_2 W$$

Discriminant linéaire de Fisher

(2D et 2 classes)



Critère de Fisher

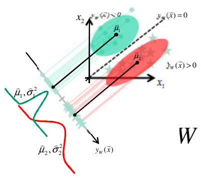
$$W = \arg\max_{W} \frac{(\hat{\mu}_1 - \hat{\mu}_2)^2}{\hat{\sigma}_1^2 + \hat{\sigma}_2^2}$$

On obtient le meilleur W en forçant à 0

$$\frac{d}{dW}\left(\frac{(\hat{\mu}_1 - \hat{\mu}_2)^2}{\hat{\sigma}_1^2 + \hat{\sigma}_2^2}\right) = 0$$

Discriminant linéaire de Fisher

(2D et 2 classes)



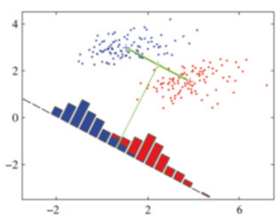
Critère de Fisher

$$W = \arg\max_{W} \frac{(\hat{\mu}_1 - \hat{\mu}_2)^2}{\hat{\sigma}_1^2 + \hat{\sigma}_2^2}$$

$$\Rightarrow W \propto \Sigma_W^{-1}(\vec{\mu}_1 - \vec{\mu}_2)$$
$$\Rightarrow \Sigma_W = \sum_{t_n = C_1} (\vec{x}_n - \vec{\mu}_1)(\vec{x}_n - \vec{\mu}_1)^{\mathrm{T}} + \sum_{t_n = C_2} (\vec{x}_n - \vec{\mu}_2)(\vec{x}_n - \vec{\mu}_2)^{\mathrm{T}}$$

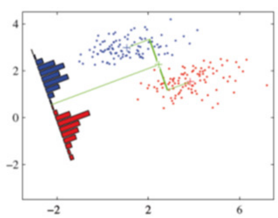
Discriminant linéaire de Fisher

(2D et 2 classes)



Sans minimisation intra-classe

$$W \propto (\vec{\mu}_1 - \vec{\mu}_2)$$



Avec minimisation intra-classe

$$W \propto \Sigma_W^{-1} (\vec{\mu}_1 - \vec{\mu}_2)$$

Discriminant linéaire de Fisher

Algorithme 2-Classes, entraînement

Calculer $\vec{\mu}_1, \vec{\mu}_2$

$$\Sigma_W = \sum_{t_n = C_1} (\vec{x}_n - \vec{\mu}_1)(\vec{x}_n - \vec{\mu}_1)^{\mathrm{T}} + \sum_{t_n = C_2} (\vec{x}_n - \vec{\mu}_2)(\vec{x}_n - \vec{\mu}_2)^{\mathrm{T}}$$

$$W = \Sigma_W^{-1} (\vec{\mu}_1 - \vec{\mu}_2)$$

$$w_0 = -\frac{\left(W^{\mathrm{T}}\vec{\mu}_1 + W^{\mathrm{T}}\vec{\mu}_2\right)}{2} \quad \left(\text{ou } w_0 = -W^{\mathrm{T}}\left(\frac{N_1}{N_1+N_2}\vec{\mu}_1 + \frac{N_1}{N_1+N_2}\vec{\mu}_2\right)\right)$$

Algorithme 2-Classes, généralisation

POUR CHAQUE donnée test $\vec{x}$ FAIRE

$t = y_W(\vec{x}) = W^{\mathrm{T}}\vec{x} + w_0$

SI $t < 0$ ALORS

t=1

SINON

t=2

Discriminant linéaire de Fisher

- On peut voir l'analyse discriminante linéaire comme un cas particulier des **moindres carrés**
  - voir section 4.1.5
- Il est possible de généraliser au cas à **plus de 2 classes**
  - voir section 4.1.6

Régression
Modèles génératifs
Discriminant de Fisher

- Émettent l'hypothèque que les données sont **gaussiennes**
- Solution de type « ***closed form*** » (inversion de matrice)

Perceptron
Régression logistique

- Émettent **aucune hypothèse** quant à la distribution des données
- Solution obtenue grâce à une **descente de gradient**.

Perceptron
(section 4.1.7, Bishop)

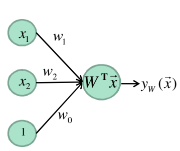
Perceptron (2 classes)

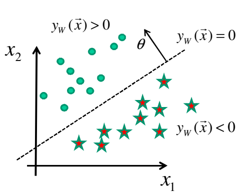
Contrairement aux approches précédentes, le Perceptron **n'émet pas** l'hypothèse que les données sont **gaussiennes**

Le Perceptron part de la definition brute de la classification binaire par **hyperplan**

$$\begin{aligned} y_W(\vec{x}) &= sign(W^T\vec{x}) \\ &= sign(\underbrace{w_0}_{\text{biais}} + \underbrace{w_1x_1 + w_2x_2 + \ldots + w_dx_d}_{\text{poids}}) \end{aligned}$$

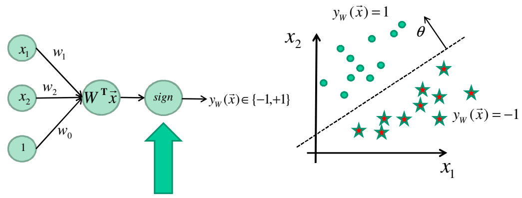
Perceptron
(2D et 2 classes)





$$\begin{aligned} y_W(\vec{x}) &= w_0 + w_1x_1 + w_2x_1 \\ &= W^{\mathrm{T}}\vec{x} \\ &= W'^{\mathrm{T}}\vec{x}' \\ &\Rightarrow W^{\mathrm{T}}\vec{x} \end{aligned}$$

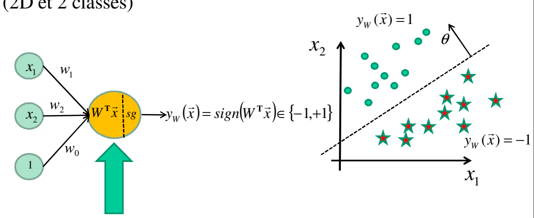
Perceptron
(2D et 2 classes)



Fonction d'activation

$$y_W(\vec{x}) = sign(W^{\mathrm{T}}\vec{x})$$

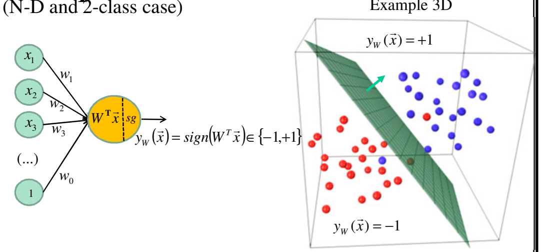
Perceptron
(2D et 2 classes)



Neurone
Produit scalaire + fonction d'activation

Perceptron
(N-D and 2-class case)

Example 3D



Nouvelle fonction de coût pour **<u>apprendre *W*</u>**

<u>Le but</u>: avec des données d'entraînement $D=\{(\vec{x}_1,t_1),(\vec{x}_2,t_2),\dots,(\vec{x}_N,t_N)\}$, estimer ***W*** afin que:

$$y_W(\vec{x}_n)=t_n \qquad \forall n$$

En d'autres mots, minimiser **l'erreur d'entraînement**

$$E_D(W)=\sum_{n=1}^{N} l(y_W(\vec{x}_n),t_n)$$

où *l(.,.)* est une **fonction de perte** (*loss function* en anglais).

Trouver la bonne fonction de perte et le bon algorithme **d'optimisation** et un sujet central en **apprentissage machine.**

Régression et classification

RAPPEL

Vous vous souvenez de la **régression**?

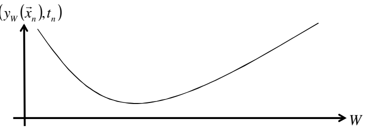
Maximum de vraisemblance $$W=\arg\min_W \underbrace{\sum_{n=1}^{N}\frac{(t_n-y_W(\vec{x}_n))^2}{2}}_{E_D(W)}$$

Maximum *a posteriori* $$W=\arg\min_W \underbrace{\sum_{n=1}^{N}\frac{(t_n-y_W(\vec{x}_n))^2}{2}+\lambda\frac{W^{\mathsf{T}}W}{2}}_{E_D(W)}$$

C'est un peu la même idée pour le Perceptron mais avec une <u>nouvelle fonction de coût</u>.

Nouvelle fonction de coût pour **<u>apprendre *W*</u>**

$E_D(W)=\sum_{n=1}^{N} l(y_W(\vec{x}_n),t_n)$



Comme nous l'avons vu auparavant, les algorithmes d'apprentissage sont des **problèmes d'optimisation** qu'on peut formuler ainsi:

$$W=\arg\min_W = \underbrace{\sum_{n=1}^{N} l(y_W(\vec{x}_n),t_n)}_{E_D(W)}$$

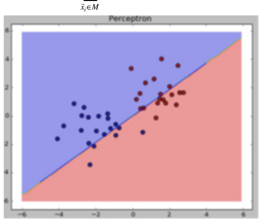
En général, on cherche une fonction de coût :

- qui a **un seul minima**
- qui est *smooth* et qui **est dérivable en tout point**
- solution **optimale** à $\frac{dE_D(W)}{dW}=0$

Nouvelle fonction de coût pour **apprendre *W***

Une function simple et indépendante de la distribution des données serait de compter 1 pour chaque donnée mal classée et 0 sinon

$$E_D(\mathrm{W}) = \sum_{\vec{x}_i \in M} 1 \qquad \text{où } M \text{ est l'ensemble des données mal classées}$$



Exemple:

$$E_D(\mathrm{w}) = 15$$

Ainsi, la meilleure solution serait celle pour laquelle on **aurait aucune donnée mal classée**.

Malheureusement, cette function n'est **pas dérivable** partout et $\frac{dE_D(\mathrm{W})}{d\mathrm{w}} = 0$ pour des **solutions non-optimales**

Critère du Perceptron

Observation

Une donnée est **mal classée** quand $W^{\mathrm{T}}\vec{x}_n > 0$ et $t_n = -1$ ou quand $W^{\mathrm{T}}\vec{x}_n < 0$ et $t_n = +1$.

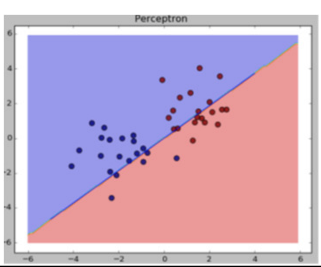
Par consequent

$-W^{\mathrm{T}}\vec{x}_n t_n$ est **toujours négatif pour les données mal classés**

Critère du Perceptron

Le **critère du Perception** est une function qui pénalise les données mal classées

$$E_D(\mathrm{W}) = \sum_{\vec{x}_n \in M} -W^{\mathrm{T}}\vec{x}_n t_n \qquad \text{où } M \text{ est l'ensemble des données mal classées}$$



$$E_D(\mathrm{w}) = 464.15$$

Perceptron

Question: comment trouver la meilleure solution W avec cette function de perte?

Réponse: une solution frequente est la **descente de gradient**.

$$W^{[k+1]} = W^{[k]} - \eta \nabla E_D\left(W^{[k]}\right)$$

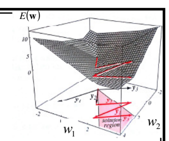
Gradient de la function de coût

Taux d'apprentissage (*learning rate*).

Descente de gradient de base

Initialiser W
k=0
FAIRE k=k+1
$W = W - \eta \nabla E_D(W)$
JUSQU'À ce que toutes les données soient bien classées

Perceptron



[Duda-Hart-Stork]

Pour le critère du Perceptron

$$\nabla E_D(\mathrm{w}) = \sum_{\vec{x}_n \in M} -t_n \vec{x}_n$$

Batch optimization

Initialiser W
k=0
DO k=k+1
$$W = W - \eta \left(\sum_{\vec{x}_n \in M} -t_n \vec{x}_n \right)$$
UNTIL toutes les données sont bien classées

NOTE importante sur le **taux d'apprentissage η**:

- **Trop faible** => convergence lente
- **Trop grand** => peut ne pas converger (et même diverger)
- Peut **décroître** à chaque itération (e.g.$\eta^{[k]} = cst / k$)

Perceptron

Une autre version de l'algorithme consiste à analyser **une donnée par itération**.

Descente de gradient stochastique

Initialiser W
k=0
DO k=k+1
FOR n = 1 to N
IF $W^{\mathrm{T}} \vec{x}_n t_n < 0$ THEN /* donnée mal classée */
$\mathbf{w} = \mathbf{w} + \eta t_n \vec{x}_n$
UNTIL toutes les données sont bien classées.

Critère du Perceptron

Fonctions d'énergie similaires au critère du Perceptron dont le **gradient est le même**

$$E_D(W) = \sum_{\vec{x}_n \in M} -W^{\mathrm{T}} \vec{x}_n t_n \qquad \text{où } M \text{ est l'ensemble des données mal classées}$$

$$E_D(W) = \sum_{n=1}^{N} \max\left(0, -t_n W^T \vec{x}_n\right)$$

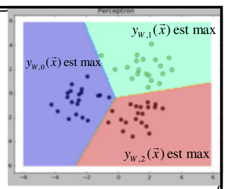
$$E_D(W) = \sum_{n=1}^{N} \max\left(0, 1 - t_n W^T \vec{x}_n\right)$$

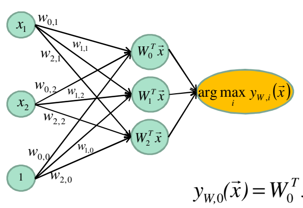
"Hinge Loss" or "SVM" Loss

Chapitre 6

Perceptron Multiclasse

(2D et 3 classes)

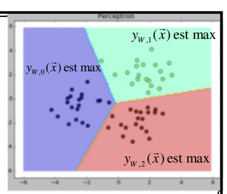


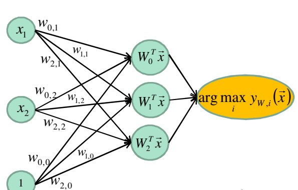


$$y_{W,0}(\vec{x}) = W_0^T \vec{x} = w_{0,0} + w_{0,1} x_1 + w_{0,2} x_2$$
$$y_{W,1}(\vec{x}) = W_1^T \vec{x} = w_{1,0} + w_{1,1} x_1 + w_{1,2} x_2$$
$$y_{W,2}(\vec{x}) = W_2^T \vec{x} = w_{2,0} + w_{2,1} x_1 + w_{2,2} x_2$$

Perceptron Multiclasse

(2D et 3 classes)





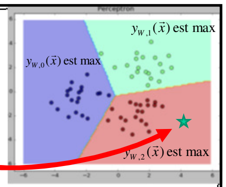
$$y_W(\vec{x}) = W^{\mathrm{T}} \vec{x}$$

$$y_W(\vec{x}) = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Perceptron Multiclasse

Exemple

★ (1.1, -2.0)



$$y_W(\vec{x}) = \begin{bmatrix} -2 & -3.6 & 0.5 \\ -4 & 2.4 & 4.1 \\ -6 & 4 & -4.9 \end{bmatrix} \begin{bmatrix} 1 \\ 1.1 \\ -2 \end{bmatrix} = \begin{bmatrix} -6.9 \\ -9.6 \\ 8.2 \end{bmatrix} \begin{array}{l} \text{Classe 0} \\ \text{Classe 1} \\ \text{Classe 2} \end{array}$$

Perceptron Multiclasse

Fonction de coût

$$E_D(W) = \sum_{\vec{x}_n \in M} \left(W_j^T \vec{x}_n - W_{t_n}^T \vec{x}_n \right)$$

Somme sur l'ensemble des données mal classées

Score de la bonne classe

Score de la mauvaise classe

$$\nabla E_D(\mathrm{w}) = \sum_{\vec{x}_n \in M} \vec{x}_n$$

Perceptron Multiclasse

Descente de gradient stochastique

Initialiser **W**
k=0, i=0
DO k=k+1
  FOR n = 1 to N
    $j = \arg\max \mathrm{W}^T \vec{x}_n$
    IF $j \neq t_i$ THEN /* donnée mal classée*/
      $\mathrm{W}_j = \mathrm{W}_j - \eta \vec{x}_n$
      $\mathrm{W}_{t_n} = \mathrm{W}_{t_n} + \eta \vec{x}_n$
UNTIL toutes les données sont bien classées.

Perceptron Multiclasse

Exemple d'entraînement (η=*1*)

$$\vec{x}_n = (0.4, -1), t_n = 0$$

$$y_W(\vec{x}) = \begin{bmatrix} -2 & 3.6 & 0.5 \\ -4 & 2.4 & 4.1 \\ -6 & 4 & -4.9 \end{bmatrix} \begin{bmatrix} 1 \\ 0.4 \\ -1 \end{bmatrix} = \begin{array}{cl} \begin{bmatrix} -1.6 \\ -7.1 \\ 0.5 \end{bmatrix} & \begin{array}{l} \text{Classe 0} \\ \text{Classe 1} \\ \text{Classe 2} \end{array} \end{array}$$

FAUX!

Perceptron Multiclasse

Exemple d'entraînement (η=*1*)

$$\vec{x}_n = (0.4, -1.0), t_n = 0$$

$$W_0 \leftarrow W_0 + \vec{x}_n \qquad \begin{bmatrix} -2.0 \\ 3.6 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 1 \\ 0.4 \\ -1 \end{bmatrix} = \begin{bmatrix} -1.0 \\ 4.0 \\ -0.5 \end{bmatrix}$$

$$W_2 \leftarrow W_2 - \vec{x}_n \qquad \begin{bmatrix} -6.0 \\ 4.0 \\ -4.9 \end{bmatrix} - \begin{bmatrix} 1 \\ 0.4 \\ -1 \end{bmatrix} = \begin{bmatrix} -7.0 \\ 3.6 \\ -3.9 \end{bmatrix}$$

En résumé

2 classes

$$E_D(W) = \sum_{\vec{x}_n \in M} -t_n W^T \vec{x}_n \qquad \text{où } M \text{ est l'ensemble des données mal classées}$$

$$E_D(W) = \sum_{n=1}^{N} \max(0, -t_n W^T \vec{x}_n)$$

$$E_D(W) = \sum_{n=1}^{N} \max(0, 1 - t_n W^T \vec{x}_n) \qquad \text{“Hinge Loss” or “SVM” Loss}$$

K classes

$$E_D(W) = \sum_{\vec{x}_n \in M} (W_j^T \vec{x}_n - W_{t_n}^T \vec{x}_n) \qquad \text{où } M \text{ est l'ensemble des données mal classées}$$

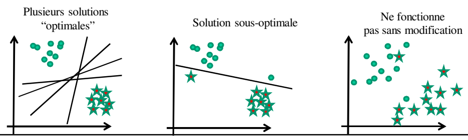
$$E_D(W) = \sum_{n=1}^{N} \sum_j \max(0, W_j^T \vec{x}_n - W_{t_n}^T \vec{x}_n)$$

$$E_D(W) = \sum_{n=1}^{N} \sum_j \max(0, 1 + W_j^T \vec{x}_n - W_{t_n}^T \vec{x}_n) \qquad \text{“Hinge Loss” or “SVM” Loss}$$

Perceptron

Advantages:
- Très simple
- Ne suppose pas que les données sont **gaussiennes**.
- Si les données sont linéairement séparables, le Perceptron est **<u>garantie</u>(!) de converger** en un nombre fini d'iterations (see Duda-Hart-Stork pour la preuve)

Limitations:
- Gradient nul pour plusieurs solutions => plusieurs solutions "parfaites"
- Les données doivent être **linéairement séparables**!

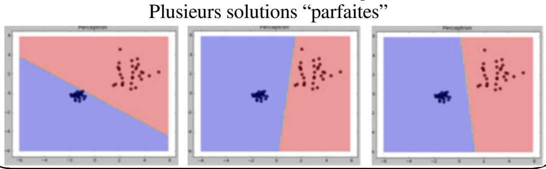


Perceptron

Advantages:
- Très simple
- Ne suppose pas que les données sont **gaussiennes**.
- Si les données sont linéairement séparables, le Perceptron est **<u>garantie</u>(!) de converger** en un nombre fini d'iterations (see Duda-Hart-Stork for proof)

Limitations:
- Gradient nul pour plusieurs solutions => plusieurs solutions "parfaites"
- Les données doivent être **linéairement séparables**!

Plusieurs solutions "parfaites"



Comment améliorer le Perceptron?

Trois façons d'améliorer le Perceptron

1. Nouvelle **fonction d'activation** + nouvelle **fonction de coût** → **Régression logistique**
2. Utiliser des **fonctions de base** → **Méthodes à noyau (chap. 5-6)**
3. **Nouveau réseau** → **Réseaux de neurones multicouches (chap. 7)**

Régression logistique

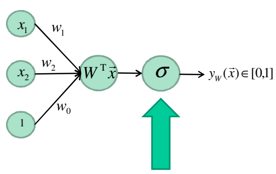
(Sections 4.2.0, 4.3.2, 5.2.0 –Bishop)

Amélioration du Perceptron

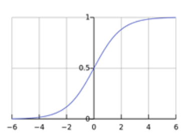
(2D, 2 classes)

Nouvelle fonction d'activation : **sigmoïde logistique**

$$\sigma(t) = \frac{1}{1+e^{-t}}$$



Fonction d'activation sigmoïdale



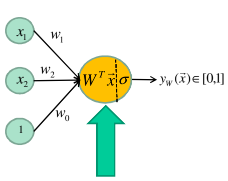
$$\boxed{y_W(\vec{x}) = \sigma\left(W^T\vec{x}\right) = \frac{1}{1+e^{-W^T\vec{x}}}}$$

Amélioration du Perceptron

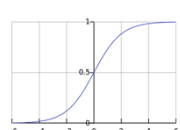
(2D, 2 classes)

Nouvelle fonction d'activation : **sigmoïde logistique**

$$\sigma(t) = \frac{1}{1+e^{-t}}$$



Neurone

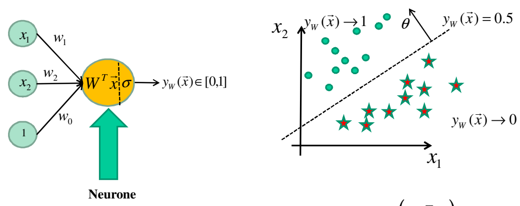


$$\boxed{y_W(\vec{x}) = \sigma\left(W^T\vec{x}\right) = \frac{1}{1+e^{-W^T\vec{x}}}}$$

Amélioration du Perceptron

(2D, 2 classes)

Nouvelle fonction d'activation : **sigmoïde logistique**



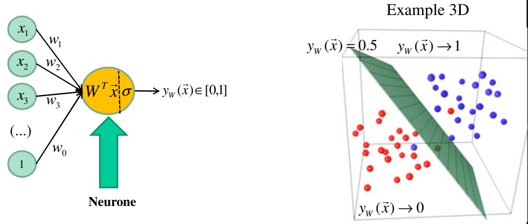
$$y_W(\vec{x}) = \sigma(W^T\vec{x})$$

Amélioration du Perceptron

(N-D, 2 classes)

Nouvelle fonction d'activation : **sigmoïde logistique**

Example 3D

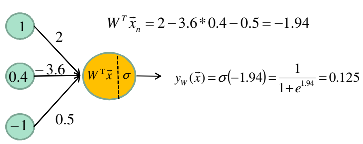


Amélioration du Perceptron

(N-D, 2 classes)

Exemple

$$\vec{x}_n = (0.4, -1.0), W = [2.0, -3.6, 0.5]$$



Puisque 0.125 est inférieur à 0.5, $\vec{x}_n$ est **derrière** le plan.

Amélioration du Perceptron

(N-D, 2 classes)

Avec une sigmoid, on peut **simuler une probabilité conditionnelle** sur c_1 étant donné $\vec{x}$

$$y_W(\vec{x}) = \sigma\left(W^T\vec{x}\right) \Rightarrow P(c_1 \mid \vec{x})$$

Preuve:

$$P(c_1 \mid \vec{x}) = \frac{P(\vec{x} \mid c_1)P(c_1)}{P(\vec{x} \mid c_0)P(c_0) + P(\vec{x} \mid c_1)P(c_1)} \qquad \text{(Bayes)}$$

$$= \frac{1}{1 + \frac{P(\vec{x} \mid c_0)P(c_0)}{P(\vec{x} \mid c_1)P(c_1)}}$$

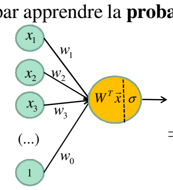
$$= \frac{1}{1+e^{-a}} \qquad \text{où } a = \ln\left[\frac{P(\vec{x} \mid c_0)P(c_0)}{P(\vec{x} \mid c_1)P(c_1)}\right]$$

$$= \sigma(a)$$

Amélioration du Perceptron

(N-D, 2 classes)

En d'autres mots, si on entraîne correctement un réseau logistique, on fini par apprendre la **probabilité conditionnelle de la classe c_1.**



$$\rightarrow y_W(\vec{x}) = \sigma\left(W^T\vec{x}\right) = P(c_1 \mid W, \vec{x})$$

$$\Rightarrow P(c_0 \mid W, \vec{x}) = 1 - P(c_1 \mid W, \vec{x}) = 1 - y_W(\vec{x})$$

Quelle est la function de coût d'un réseau logistique?

Rappel

Probabilité jointe
et
distribution de Bernouilli

RAPPEL

Soit 2 classes c_0 et c_1. Si on connaît $P(c_0)$ et $P(c_1)$ alors

la probabilité d'observer une séquence $T=\{t_1,t_2,\ldots,t_N\}$ où tous les t_n sont indépendants est donnée par

$$\begin{aligned} P(T) &= P(t_1,t_2,\ldots,t_N) \\ &= P(t_1)P(t_2)\ldots P(t_N) \\ &= \prod_{n=1}^{N} P(t_n) \end{aligned}$$

Exemple:

$$P(c_0)=0.7, P(c_1)=0.3 \text{ et } T=\{c_0,c_1,c_1,c_0,c_1\}$$

$$P(T)=0.7\times0.3\times0.3\times0.7\times0.3=0.013$$

RAPPEL

Puisqu'on a 2 classes et que $P(c_1)=1-P(c_0)$

$$P(T)=\prod_{n=1}^{N}P(t_n) \;=\prod_{n=1}^{N}P(c_1)^{t_n}(1-P(c_1))^{1-t_n}$$

où $t_i=0$ pour c_0, $t_i=1$ pour c_1

Exemple: $P(c_0)=0.7, P(c_1)=0.3$ et $T=\{c_0,c_1,c_1,c_0,c_1\}$

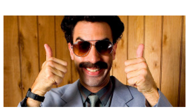
$$\begin{aligned} P(Y) &= \prod_{n=1}^{5}0.3^{t_n}(1-0.3)^{1-t_n} \\ &= 0.3^{t_1}(0.7)^{1-t_1}\times0.3^{t_2}(0.7)^{1-t_2}\times0.3^{t_3}(0.7)^{1-t_3}\times0.3^{t_4}(0.7)^{1-t_4}\times0.3^{t_5}(0.7)^{1-t_5} \\ &= 0.3^{0}(0.7)^{1-0}\times0.3^{1}(0.7)^{1-1}\times0.3^{1}(0.7)^{1-1}\times0.3^{0}(0.7)^{1-0}\times0.3^{1}(0.7)^{1-1} \\ &= 0.7\times0.3\times0.3\times0.7\times0.3 \\ &= 0.013 \end{aligned}$$

RAPPEL

Puisqu'on a 2 classe et que $P(c_1)=1-P(c_0)$

$$P(T)=\prod_{n=1}^{N}P(c_1)^{t_n}(1-P(c_1))^{1-t_n}$$

où $t_n=0$ pour c_0, $t_n=1$ pour c_1



Distribution de Bernoulli

Fin du Rappel

Fonction de coût d'un réseau logistique?

(2 classes)

Dans le cas d'un réseau logistique nous avons

Ensemble d'entraînement : $D=\{(\vec{x}_1,t_1),(\vec{x}_2,t_2),\ldots,(\vec{x}_n,t_n)\}$

Sortie du réseau: $y_W(\vec{x})=\sigma(\mathrm{W}^T\vec{x})=P(c_1 \mid W,\vec{x})$

$$P(D \mid W)=\prod_{n=1}^{N} P(c_1 \mid W,\vec{x}_n)^{t_n}(1-P(c_1 \mid W,\vec{x}_n))^{1-t_n}$$

$$=\prod_{n=1}^{N} y_W(\vec{x}_n)^{t_n}(1-y_W(\vec{x}_n))^{1-t_n}$$

Fonction de coût d'un réseau logistique?

(2 classes)

$$P(D \mid W)=\prod_{n=1}^{N} y_W(\vec{x}_n)^{t_n}(1-y_W(\vec{x}_n))^{1-t_n}$$

Solution : Maximum de vraisemblance

$$\begin{aligned} W &= \arg\max_W P(D \mid W) \\ &= \arg\max_W \prod_{n=1}^{N} y_W(\vec{x}_n)^{t_n}\left(1-y_W(\vec{x}_n)\right)^{1-t_n} \\ &= \arg\min_W \sum_{n=1}^{N} -\ln\left[y_W(\vec{x}_n)^{t_n}\left(1-y_W(\vec{x}_n)\right)^{1-t_n}\right] \\ &= \arg\min_W \underbrace{\sum_{n=1}^{N} t_n \ln\left(y_W(\vec{x}_n)\right)+(1-t_n)\ln\left(1-y_W(\vec{x}_n)\right)}_{E_D(W)} \end{aligned}$$

Fonction de coût d'un réseau logistique?

(2 classes)

$$P(D|W)=\prod_{n=1}^{N} y_W(\vec{x}_n)^{t_n}\left(1-y_W(\vec{x}_n)\right)^{1-t_n}$$

La fonction de coût est **–ln de la vraisemblance**

$$E_D(\mathrm{W})=-\sum_{n=1}^{N} t_n \ln\left(y_W(\vec{x}_n)\right)+(1-t_n)\ln\left(1-y_W(\vec{x}_n)\right)$$

Entropie croisée (*Cross entropy*)

On peut également démontrer que

$$\frac{dE_D(\mathrm{W})}{d\mathrm{W}}=\sum_{n=1}^{N}\left(y_W(\vec{x}_n)-t_n\right)\vec{x}_n$$

Contrairement au Perceptron
le gradient ne depend pas seulement
des données mal classées

Optimisation d'un réseau logistique

Optimisation par batch

```
Initialiser W
k=0, i=0
DO k=k+1
```
$$\frac{dE_D(\mathrm{W})}{d\mathrm{W}}=\sum_{n=1}^{N}\left(y_W(\vec{x}_n)-t_n\right)\vec{x}_n$$
$$\mathrm{W}=\mathrm{W}-\eta\frac{dE_D(\mathrm{W})}{d\mathrm{W}}$$
```
UNTIL K==K_MAX.
```

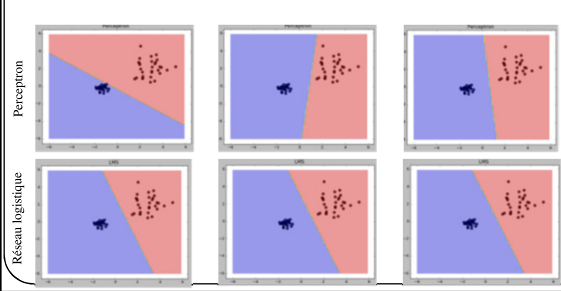
Descente de gradient stochastique

```
Initialiser W
k=0, i=0
DO k=k+1
   FOR n = 1 to N
```
$$\mathrm{W}=\mathrm{W}-\eta\left(y_W(\vec{x}_n)-t_n\right)\vec{x}_n$$
```
UNTIL K==K_MAX.
```

Réseau logistique

Advantages:

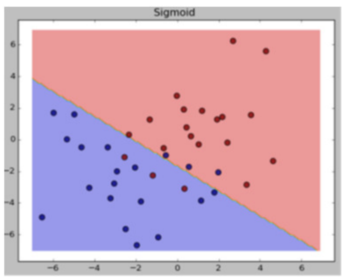
- **Plus stable que le Perceptron**
- Fonctionne mieux avec des **données non séparables**



Réseau logistique

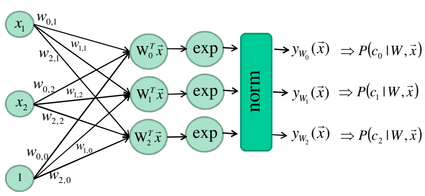
Advantages:

- **Plus stable que le Perceptron**
- Fonctionne mieux avec des **données non séparables**



Et pour K>2 classes?

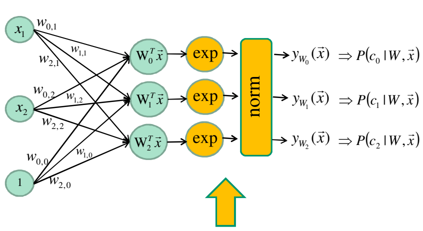
Nouvelle fonction d'activation : **Softmax**



$$y_{W_i}(\vec{x}) = \frac{e^{W_i^T \vec{x}}}{\sum_c e^{W_c^T \vec{x}}}$$

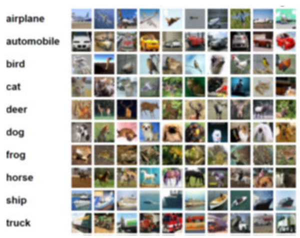
Et pour K>2 classes?

Nouvelle fonction d'activation : **Softmax**



Softmax

Et pour K>2 classes?



'airplane' ⇒ t = [1000000000]
'automobile ' ⇒ t = [0100000000]
'bird' ⇒ t = [0010000000]
'cat' ⇒ t = [0001000000]
'deer' ⇒ t = [0000100000]
'dog' ⇒ t = [0000010000]
'frog' ⇒ t = [0000001000]
'horse' ⇒ t = [0000000100]
'ship' ⇒ t = [0000000010]
'truck' ⇒ t = [0000000001]

Étiquettes de classe **: <u>one-hot vector</u>**

Et pour K>2 classes?

$$P(D|W)=\prod_{n=1}^{N}\prod_{k=1}^{K}\left(P(t_n|W,\vec{x}_n)\right)^{t_{nk}}$$

Entropie croisée (*cross entropy*)

$$E_D(\mathrm{W})=-\ln\left(P(D|W)\right)=-\sum_{n=1}^{N}\sum_{k=1}^{K}t_{kn}\ln P(t_n|W,\vec{x}_n)$$

Puisqu'on veut que la sortie du réseau $y_W(\vec{x}_n)$ soit égale à $P(t_n|W,\vec{x}_n)$

$$\boxed{E_D(\mathrm{W})=-\sum_{n=1}^{N}\sum_{k=1}^{K}t_{kn}\ln y_{W_k}(\vec{x}_n)}$$

Et pour K>2 classes?

$$E_D(\mathrm{W})=-\sum_{n=1}^{N}\sum_{k=1}^{K}t_{kn}\ln y_W(\vec{x}_n)$$

On peut montrer que

$$\boxed{\frac{dE_D(\mathrm{W})}{d\mathrm{W}}=\frac{1}{N}\sum_{n=1}^{N}\vec{x}_n\left(y_W(\vec{x}_n)-t_{kn}\right)}$$

Optimisation d'un réseau logistique multiclasse

Optimisation par batch

Initialiser **W**
k=0, i=0
DO k=k+1

$$\frac{dE(\mathrm{W})}{d\mathrm{W}} = \frac{1}{N}\sum_{n=1}^{N}\left(y_W(\vec{x}_n)-t_n\right)\vec{x}_n$$

$$\mathrm{W} = \mathrm{W} - \eta\frac{dE(\mathrm{W})}{d\mathrm{W}}$$

UNTIL K==K_MAX.

Voir Ipython notebook « classification_lineaire.ipynb pour un exemple de code

Descente de gradient stochastique

Initialiser **W**
k=0, i=0
DO k=k+1
FOR n = 1 to N

$$\mathrm{W} = \mathrm{W} - \eta\left(y_W(\vec{x}_n)-t_n\right)\vec{x}_n$$

UNTIL K==K_MAX.

Régularisation

Différents poids peuvent donner le même score

$$\vec{x} = (1.0, 1.0, 1.0)$$
$$\vec{w}_1 = [1, 0, 0]$$
$$\vec{w}_2 = [1/3, 1/3, 1/3]$$

$$\vec{w}_1\vec{x} = \vec{w}_2\vec{x} = 1$$

Quels poids sont les meilleurs?

Solution: Maximum a posteriori

Maximum *a posteriori*

Régularisation

Maximum de vraisemblance

$$W = \arg\max_W P(D \mid W)$$
$$\ldots$$
$$= \arg\min_W \underbrace{\sum_{n=1}^{N} -\ln P(t_n \mid \vec{x}_n, W)}_{E_D(W)}$$

Maximum *a posteriori*

$$W = \arg\max_W P(W \mid D)$$
$$\ldots$$
$$= \arg\min_W \underbrace{\sum_{n=1}^{N} -\ln P(t_n \mid \vec{x}_n, W)}_{E_D(W)} + \frac{\lambda}{2}R(W)$$

Maximum *a posteriori*

Régularisation

Constante

$$\arg\min_{W} = E_D(W) \; + \frac{\lambda}{2} R(W)$$

Fonction de perte

Régularisation

En général L1 ou L2 $R(\theta) = \|\mathrm{W}\|_1$ ou $\|\mathrm{W}\|_2$

Note :
il est fréquent de combiner différentes **fonctions de coût** avec différentes **fonctions de régularisation**

Maximum *a posteriori*

Exemple : **entropie croisée** + **normalisation L2**

$$\arg\min_{W} -\ln\left(P(D \mid W)\right) + \frac{\lambda}{2}\|\mathrm{W}\|^2$$

$$\arg\min_{W} -\sum_{n=0}^{N} t_n \ln\left(y_W(\vec{x}_n)\right) + (1-t_i)\ln\left(1-y_W(\vec{x}_n)\right) + \frac{\lambda}{2}\sum_{d=1}^{D} w_d^{\,2}$$

$$\frac{dE(\mathbf{w})}{d\mathbf{w}} = \sum_{n=1}^{N}\left(y(\vec{x}_n)-t_n\right)\vec{x}_n + \lambda\sum_{d=0}^{D} w_d$$

Maximum *a posteriori*

Exemple : ***Hinge loss*** + **normalisation L2**

$$E(\mathbf{w}) = \sum_{n=1}^{N} \max\left(0, 1 - t_n \mathbf{w}^T \vec{x}_n\right) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

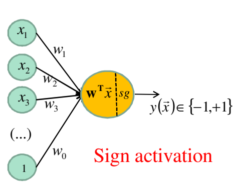
$$\frac{dE(\mathbf{w})}{d\mathbf{w}} = \sum_{\vec{x}_n \in M} -t_n \vec{x}_n + \lambda \sum_{d=0}^{D} w_d$$

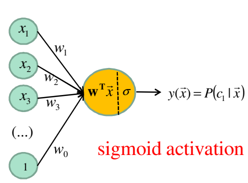
Wow! Beaucoup d'information…

Résumons…

Réseaux de neurones

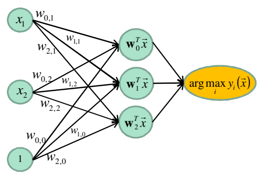
2 classes

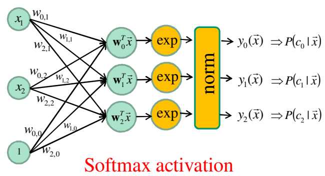




Réseaux de neurones

K classes





Softmax activation

Fonctions de coûts

2 classes

$$E_D(\mathrm{W}) = \sum_{\vec{x}_n \in M} -t_n \mathrm{W}^{\mathrm{T}} \vec{x}_n$$ où M est l'ensemble des données mal classées

$$E_D(\mathrm{W}) = \sum_{n=1}^{N} \max\left(0, -t_n \mathrm{W}^{\mathrm{T}} \vec{x}_n\right)$$

$$E_D(\mathrm{W}) = \sum_{n=1}^{N} \max\left(0, 1 - t_n \mathrm{W}^{\mathrm{T}} \vec{x}_n\right)$$ "Hinge Loss" ou "SVM" Loss

$$E_D(\mathrm{W}) = -\sum_{n=1}^{N} t_n \ln\left(y_W(\vec{x}_n)\right) + (1 - t_n)\ln\left(1 - y_W(\vec{x}_n)\right)$$ Entropie croisée (ou *cross entropy*)

Fonctions de coûts

K classes

$$E_D(\mathrm{W}) = \sum_{\vec{x}_n \in M} \left(\mathrm{W}_j^{\mathrm{T}} \vec{x}_n - \mathrm{W}_{t_n}^{\mathrm{T}} \vec{x}_n\right)$$ où M est l'ensemble des données mal classées

$$E_D(\mathrm{W}) = \sum_{n=1}^{M} \sum_{j} \max\left(0, \mathrm{W}_j^{\mathrm{T}} \vec{x}_n - \mathrm{W}_{t_n}^{\mathrm{T}} \vec{x}_n\right)$$

$$E_D(\mathrm{W}) = \sum_{n=1}^{N} \sum_{j} \max\left(0, 1 + \mathrm{W}_j^{\mathrm{T}} \vec{x}_n - \mathrm{W}_{t_n}^{\mathrm{T}} \vec{x}_n\right)$$ "Hinge Loss" ou "SVM" Loss

$$E_D(\mathrm{W}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_{W,k}(\vec{x}_n)$$ Entropie croisée avec « *one hot vector* » (ou *cross entropy*)

Maximum *a posteriori*

Constante

$$E(\mathbf{w}) = \sum_{n=1}^{N} l(y(\vec{x}_n), t_n) + \frac{\lambda}{2} R(\mathbf{w})$$

Fonction de perte

Regularisation

$$R(\theta) = \|W\|_1 \ ou\ \|W\|_2$$

Optimisation

Descente de gradient

$$\mathbf{w}^{[k+1]} = \mathbf{w}^{[k]} - \eta^{[k]} \nabla E$$

∇E : Gradient de la function de coût

$\eta^{[k]}$: Taux d'apprentissage ou "learning rate".

Optimisation par Batch

Initialiser $\mathbf{w}$
k=0
FAIRE k=k+1
$\mathbf{w} = \mathbf{w} - \eta^{[k]} \sum_i \nabla E(\vec{x}_i)$
JUSQU'À ce que toutes les données sont bien classées ou k==MAX_ITER

Descente de gradient stochastique

Initialiser $\mathbf{w}$
k=0
FAIRE k=k+1
FOR n = 1 to N
$\mathbf{w} = \mathbf{w} - \eta^{[k]} \nabla E(\vec{x}_n)$
JUSQU'À ce que toutes les données sont bien classées ou k== MAX_ITER

Parfois $\eta^{[k]} = cst / k$

Exemples

```
from sklearn.linear_model import SGDClassifier

C = SGDClassifier(loss='perceptron',learning_rate='constant',
                  eta0=1, n_iter=1000)


C = SGDClassifier(loss='log', learning_rate ='constant',
                  eta0=1, n_iter=1000)


C = SGDClassifier(loss='hinge',penalty='l2',alpha=0.01,
                  learning_rate ='invscaling', eta0=1, n_iter=1000)
```

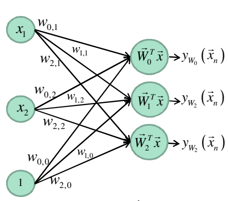
Mieux comprendre

Entropie croisée vs *Hinge loss*

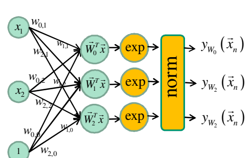
Entropie croisée vs *Hinge Loss*

Dépendamment de la *loss* utilisée, la **sortie du réseau** sera différente

- *Hinge loss* : sortie multiplication matrice-vecteur
- Entropie croisée : sortie softmax



$$y_{W_i}(\vec{x}_n) = \vec{W}_i^{\mathrm{T}} \vec{x}$$



$$y_{W_i}(\vec{x}_n) = \frac{e^{\vec{W}_i^{\mathrm{T}} \vec{x}}}{\sum_j e^{\vec{W}_j^{\mathrm{T}} \vec{x}}}$$

$$\vec{x} = \begin{bmatrix}-15\\22\\-44\\56\end{bmatrix}, t = 2$$

$$\underbrace{\begin{bmatrix}0.0 & 0.01 & -0.05 & 0.1 & 0.05\\ 0.2 & 0.7 & 0.2 & 0.05 & 0.16\\ -0.3 & 0.0 & -0.45 & -0.2 & 0.03\end{bmatrix}}_{W} \underbrace{\begin{bmatrix}1\\-15\\22\\-44\\56\end{bmatrix}}_{\vec{x}}$$

Hinge loss

Score

$$\begin{bmatrix}-2.85\\0.86\\0.28\end{bmatrix}$$

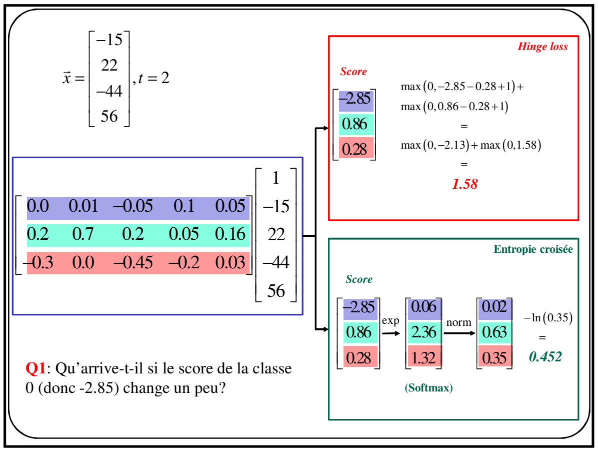
$$\max(0, -2.85 - 0.28 + 1) + \max(0, 0.86 - 0.28 + 1) = \max(0, -2.13) + \max(0, 1.58) = \mathbf{1.58}$$

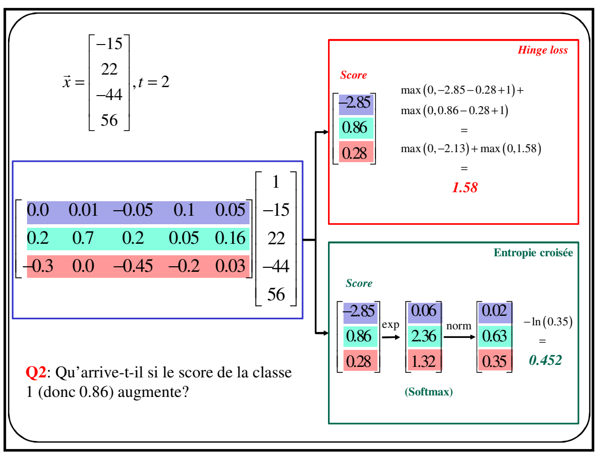
Entropie croisée

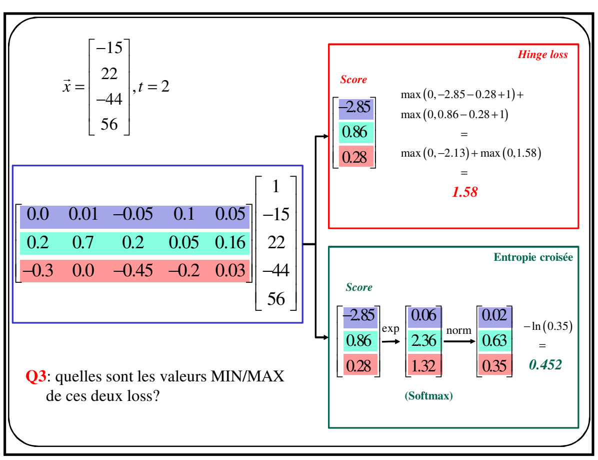
Score

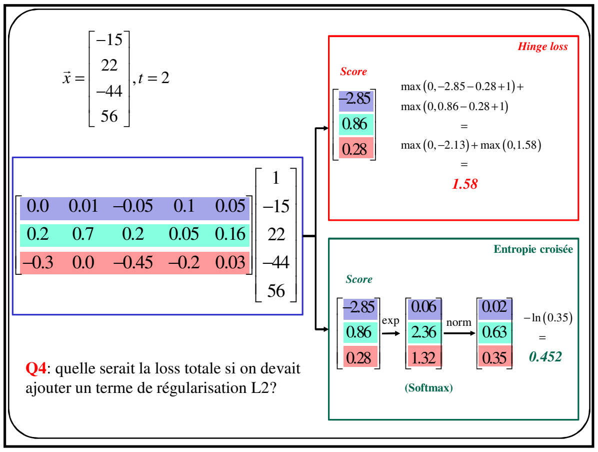
$$\begin{bmatrix}-2.85\\0.86\\0.28\end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix}0.06\\2.36\\1.32\end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix}0.02\\0.63\\0.35\end{bmatrix} \quad -\ln(0.35) = \mathbf{0.452}$$

(Softmax)


x = [−15, 22, −44, 56], t = 2
0.0 0.01 −0.05 0.1 0.05
0.2 0.7 0.2 0.05 0.16
−0.3 0.0 −0.45 −0.2 0.03
[1, −15, 22, −44, 56]
Hinge loss
Score
−2.85
0.86
0.28
max(0, −2.85 − 0.28 + 1) +
max(0, 0.86 − 0.28 + 1)
=
max(0, −2.13) + max(0, 1.58)
=
1.58
Entropie croisée
Score
−2.85 0.86 0.28
exp
0.06 2.36 1.32
norm
0.02 0.63 0.35
−ln(0.35)
=
0.452
(Softmax)
Q1: Qu'arrive-t-il si le score de la classe 0 (donc -2.85) change un peu?


x = [−15, 22, −44, 56], t = 2
0.0 0.01 −0.05 0.1 0.05
0.2 0.7 0.2 0.05 0.16
−0.3 0.0 −0.45 −0.2 0.03
[1, −15, 22, −44, 56]
Hinge loss
Score
−2.85
0.86
0.28
max(0, −2.85 − 0.28 + 1) +
max(0, 0.86 − 0.28 + 1)
=
max(0, −2.13) + max(0, 1.58)
=
1.58
Entropie croisée
Score
−2.85 0.86 0.28
exp
0.06 2.36 1.32
norm
0.02 0.63 0.35
−ln(0.35)
=
0.452
(Softmax)
Q2: Qu'arrive-t-il si le score de la classe 1 (donc 0.86) augmente?


x = [−15, 22, −44, 56], t = 2
0.0 0.01 −0.05 0.1 0.05
0.2 0.7 0.2 0.05 0.16
−0.3 0.0 −0.45 −0.2 0.03
[1, −15, 22, −44, 56]
Hinge loss
Score
−2.85
0.86
0.28
max(0, −2.85 − 0.28 + 1) +
max(0, 0.86 − 0.28 + 1)
=
max(0, −2.13) + max(0, 1.58)
=
1.58
Entropie croisée
Score
−2.85 0.86 0.28
exp
0.06 2.36 1.32
norm
0.02 0.63 0.35
−ln(0.35)
=
0.452
(Softmax)
Q3: quelles sont les valeurs MIN/MAX de ces deux loss?



$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$
$\begin{bmatrix} 0.0 & 0.01 & -0.05 & 0.1 & 0.05 \\ 0.2 & 0.7 & 0.2 & 0.05 & 0.16 \\ -0.3 & 0.0 & -0.45 & -0.2 & 0.03 \end{bmatrix} \begin{bmatrix} 1 \\ -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}$
Hinge loss
Score
$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$
$\max(0, -2.85 - 0.28 + 1) +$
$\max(0, 0.86 - 0.28 + 1)$
$=$
$\max(0, -2.13) + \max(0, 1.58)$
$=$
1.58
Entropie croisée
Score
$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} \xrightarrow{\exp} \begin{bmatrix} 0.06 \\ 2.36 \\ 1.32 \end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix} 0.02 \\ 0.63 \\ 0.35 \end{bmatrix}$
$-\ln(0.35)$
$=$
0.452
(Softmax)
Q4: quelle serait la loss totale si on devait ajouter un terme de régularisation L2?