Université d'Avignon
M1 Intelligence Artificielle – Apprentissage supervisé – TP 1
Juan-Manuel Torres - 2025 - PERCEPTRONS

Travaillez individuellement ou en binôme. Langages acceptés : C/C++ ; python ; awk, rust, perl ; ruby – Exclus : R, java, javascript
Interdit d'utiliser des fonctions/librairies des RN natives (R, python, matlab, etc).
Interdit d'utiliser IA pour générer vos codes, par contre IN à volonté

Théorie

L'algorithme du perceptron peut trouver un hyperplan qui sépare 2 classes d'exemples si l'ensemble d'apprentissage est linéairement séparable (LS). Or, créer un ensemble de P points de dimension N (tiré au hasard, dont la classe est tirée au hasard aussi) et garantir qu'il est LS s'avére difficile, car on ne saurait pas dire s'il est ou pas LS avant de le classer par un perceptron... Ceci revient au problème de l'œuf et de la poule.

Une façon de garantir la séparabilité linéaire d'un tel ensemble de données artificielles, consiste à tirer au hasard les poids **W***(i); i=0,1,...,N d'un perceptron appelé **perceptron professeur**, puis d'affecter la classe **TAU**(mu); mu=1,...,P ; des P points de **X** par :

TAU(mu) = signe [**W***(i) · **X**(mu, i)] ; où **signe(a)** = -1 si a < 0, +1 autrement

Nous allons étudier certaines propriétés de ces ensembles aléatoires

Travail pratique

0/ Perceptron. Programmer l'algorithme du perceptron a) version batch et b) version online. Pour chaque version faire apprendre un perceptron W sur les ensembles d'apprentissage suivants :

i/ fonction ET entrées binaires (N=2) ;
ii/ fonction OU entrées binaires (N=2) ;
iii/ Un des ensemble au choix des exemples vus au cours.

Fixer le taux d'apprentissage **eta** à une valeur adéquate 0 < **eta** < 1. Vérifiez que l'algorithme marche bien.

1/ Données LS aléatoires. Construire un ensemble LS de P exemples en N+1 dimensions avec un **perceptron professeur** **W*.** *Attention : le poids* **W***(0) *étant le biais, donc* **X**(mu,0)=1

2/ Apprentissage. Apprendre avec a) la **version batch** de l'algorithme du perceptron et b) la **version online**. Pour chaque version faire apprendre un ensemble de 20 **perceptrons élèves** **W** sur l'ensemble LS obtenu lors de la génération de données. Calculez :

a) Les N+1 poids **W** du perceptron élève ;
b) Le nombre d'itérations IT nécessaires pour converger ;
c) Le recouvrement R entre le perceptron professeur **W*** et les élèves **W** :

R = cos [(**W*** · **W**) / | **W*** |·| **W** |] ; | **a** | = norme de **a** = $\sqrt{\sum}$ a(i)^2 ; i=0,1,…,N

Utilisez **threads** pour l'apprentissage des elèves.

3/ Tests. Lancer le programme avec les valeurs: N=2,10,100,500,1000,5000; P=10,100,200,500,1000. Donner vos résultats sous la forme de 2 tableaux (ou graphiques en couleurs = bonus) (batch et online) où chaque case affiche la moyenne du nombre d'itérations **<IT>** et la moyenne du rapport **<R>**, sur 100 tirages aléatoires.
Construisez trois tableaux en fonction de eta : d'abord l'eta0 de l'exercice 0, puis avec eta=eta0/2 et finalement eta=eta0/10
Bonus : utilisez threads pour vos tests

eta

	P=10	P=100	P=500	P=1000
N = 2	<IT>;<R>			
N = 10				
N = 100				
…		…		
N = 5000				

eta/2

	P=10	P=100	P=500	P=1000
N = 2	<IT>;<R>			
N = 10				
N = 100				
…		…		
N = 5000				

eta/10

	P=10	P=100	P=500	P=1000
N = 2	<IT>;<R>			
N = 10				
N = 100				
…		…		
N = 5000				

4/ Questions. Que pouvez vous dire des moyennes de **<IT>** et de **<R>** par rapport à N et P ? Que pouvez dire du temps d'exécution en fonction d'eta pour la version batch et celle online ?
5/ Rapport. Tableaux de moyennes + réponse aux questions. Rendu PDF et codes sources compressés ZIP/GZIP. dépôt sur ENT.

Université d'Avignon – Master Intelligence Artificielle
Apprentissage supervisé – TP 2
Juan-Manuel Torres - 2025 Binômes ou monômes

Langages acceptés : C/C++ ; python ; awk, rust, perl ; ruby – Exclus : R, java, javascript
Interdit d'utiliser des fonctions/librairies des RN natives (R, python, matlab, etc).
IA interdite, IN à volonté

Théorie

Le serveur UCI (machine learning datasets) https://archive.ics.uci.edu contient beaucoup des données pour tester les algorithmes d'apprentissage supervisé. Dans ce TP nous allons utiliser l'ensemble de données des échos de sonar codées en plusieurs dimensions, proposé par Tierry Sejnowsky. Ceci est un ensemble classique dans le domaine des Réseaux de neurones. Les données et leur description se trouvent ici :
https://archive.ics.uci.edu/dataset/151/connectionist+bench+sonar+mines+vs+rocks

Pour ce TP vous pouvez consulter le papier *Torres & Gordon, Characterization of the Sonar Signals Benchmark*, https://link.springer.com/article/10.1023/A:1009605531255 qui est disponible sur le site du cours dans le fichier SONAR_torres.pdf

Travail pratique

1 **Données**. Télécharger les données du Sonar depuis UCI. Construire 3 ensembles d'apprentissage : train (P=104 données marquées avec une *), test (P=104 données) et ensemble complet (208 données). Le nombre de dimensions est N=60. Formater un écho par ligne, séparation de colonnes par des tabs.

PARTIE I

2 **Apprentissage sur « train »**. Utiliser l'algorithme du perceptron (justifier le choix **version batch** vs **online** selon votre TP1**)** pour apprendre l'ensemble « train », puis tester sur l'ensemble de « test ».

a) Calculer les erreurs d'apprentissage Ea et de généralisation Eg ;
b) Afficher les N+1 poids **W** du perceptron ;
c) Calculer les stabilités des P exemples de « test » selon la formule de gamma (distance a l'hyperplan séparateur avec les poids normés)
d) Graphique des stabilités

3. **Apprentissage sur « test », cad inverser les ensembles :** Apprendre sur l'ensemble « test », puis généraliser sur l'ensemble « train ». Calculer a), b) et c) du point précédent 2.

PARTIE II

4 Programmer l'algorithme d'apprentissage Pocket : il garde le meilleur résultat de l'algorithme du perceptron en fonctions des itérations et en fonction d'un Nb d'erreurs prédéfini.
Apprendre sur l'ensemble « train » en stoppant l'erreur d'apprentissage Ea fixé à l'avance, puis tester la généralisation Eg sur l'ensemble de test.
Tester aussi l'initialisation aléatoire vs une initialisation de Hebb. Refaire les expériences en échangeant les données « train » et « test ». Que pouvez vous dire de Ea et Eg en fonction des ensembles? et en fonction de l'initialisation? En fonction du eta (pas d'apprentissage)? Montrer vos résultats sous forme de tables ou de graphiques (bonus).

PARTIE III

5 Apprentissage sur « train + test ». Utiliser l'algorithme du perceptron pour apprendre l'ensemble fusionné L = L(train)+ L(test) (avec leur classe tau)
L'ensemble L est-il LS ou pas LS ? Justifier votre réponse en fonction des calculs de vos programmes.

PARTIIE IV

6. Early Stopping. Utilisez l'ensemble L : Train+Test (208 patrons) pour créer 3 ensembles au hasard:
LA = Apprentissage 50 % des données
LV = Validation 20 % des données
LT = Test 30 % des données

Faites l'apprentissage sur LA, validez l'erreur sur LV et avec Early Stopping testez sur LT.
Répétez cette expérience plusieurs fois afin d'obtenir des statistiques sur la moyenne des erreurs Ea, Et, et Ev (validation).

6 Rapport. Pour chaque partie : Ea, Eg, (Ev en plus pour la partie IV), P stabilités, N+1 poids, réponses aux questions. Rendu PDF du Latex et codes sources compressés ZIP/GZIP su ENT.

Que la force soit avec vous !

Université d'Avignon – Master 1
Intelligence Artificielle – Apprentissage supervisé – TP 3
Juan-Manuel Torres - 2025

Travaillez individuellement ou en binôme. Langages acceptés : C/C++ ; python ; awk, rust, perl ; ruby – Exclus : R, java, javascript
Interdit d'utiliser des fonctions/librairies des RN natives (R, python, matlab, etc)
IA interdite, IN à volonté

Théorie

L'algorithme d'apprentissage Minimerror utilise une fonction de coût qui dépend d'un hyper-paramètre appelé température T>0; qui peut être écrit comme **β**=1/T. Il a les propriétés de trouver un hyperplan séparateur de stabilité maximale si l'ensemble d'apprentissage est LS ou s'il ne l'est pas, de trouver le nombre minimum de fautes ayant des stabilités maximales. Utilisez la fonction de coût, sa dérivée et une stratégie de recuit déterministe pour programmer l'algorithme minimerror.

Pour ce TP, en particulier pour les paramètres (initialisation, delta de températures, delta du recuit, etc) consulter thèse de M *Torres et les slides Minimerror,* disponibles sur ENT.
Il s'agit d'un algorithme difficile : Il n'y a pas UNE seule implantation de cette algorithme : impléméntez votre propre version de l'implémentation

Travail pratique. Données au choix : ET, XOR, sonar, machine UCI...

PARTIE I

Minimerror avec 1 température β=1/T. Programmer l'algorithme minimerror pour apprendre sur l'ensemble « train », puis tester sur l'ensemble de « test ». Les données à tester NE DOIVENT PAS CONTENIR LEUR CLASSE.

a) Calculer les erreurs d'apprentissage Ea et de généralisation Eg ;
b) Afficher les N+1 poids **W** du perceptron ;
c) Calculer les stabilités des P exemples de « test » (distance a l'hyperplan séparateur avec les poids normés)
d) Graphique des stabilités

Note : Il est possible que Ea>0 en fonction des paramètres

PARTIE II

Minimerror avec 2 températures β+ / β- = rapport de températures. Modifier l'algorithme minimerror pour apprendre avec 2 températures sur l'ensemble « train », puis tester sur l'ensemble de « test ». Les données à tester NE DOIVENT PAS CONTENIR LEUR CLASSE.

a) Calculer les erreurs d'apprentissage Ea et de généralisation Eg ;
b) Stocker les N+1 poids **W** du perceptron dans un fichier;
c) Calculer les stabilités des P exemples de « test » (distance a l'hyperplan séparateur avec les poids normés)
d) Graphique des stabilités en fonction de **β = 1,2,...10**

Rapport. Pour chaque partie : Vos programmes, Ea, Eg, P stabilités, N+1 poids. Rendu PDF et codes sources compressés sur ENT

Université d'Avignon – Master 1
Intelligence Artificielle – Apprentissage supervisé – TP 4 BONUS
Juan-Manuel Torres - 2025

Travaillez individuellement ou en binôme. Langages acceptés : C/C++ ; python ; awk, rust, perl ; ruby – Exclus : R, java, javascript
Interdit d'utiliser des fonctions/librairies des RN natives (R, python, matlab, etc)

Théorie

L'algorithme incrémental Monoplan ajoute des hyperplans afin de classer les exemples de l'ensemble d'apprentissage. Un premier hyperplan classe le mieux possible les exemples en diminuant le nb de fautes d'apprentissage. Un deuxième hyperplan classe les exemples mal appris par le 1er hyperplan, tout en modifiant la sortie attendue par tau(t+1) = tau(t)*sigma(t) ; t etant l'étape initiale de la construction du 1er hyperplan. Les perceptrons peuvent être entraînés avec n'importe quelle règle d'apprentissage.

Travail pratique

Monoplan. Programmer l'algorithme Monoplan pour apprendre un problème de classification exhaustif : la N-parité (voir papier sur le site du cours). L'apprentissage de chaque perceptron avec un algorithme au choix.

a) Ajouter des hyperplans au fur et a mesure de la construction du réseau
b) Garder les poids du réseau : matrice de (N+1) * (NH + 1)
c) Calculer les stabilités des P exemples d'apprentissage (distance a l'hyperplan séparateur avec les poids normés)
d) Graphique des stabilités pour N=2,3,…,10

Rendu. Rapport : Rapporter les poids du réseau. Fichiers : Codes Monoplan+Poids des réseaux. Rendu PDF Latex et codes sources compressés ZIP/GZIP. Le 16 janvier avant 23h59 : dépôt sur ENT.