AVIGNON UNIVERSITÉ

Rapport TP 2 Approches Neuronales

Abdou NIANG

15 mars 2023

Master Informatique
Intelligence Artificielle

UE APPRENTISSAGE SUPERVISE
ECUE Approches Neuronales

Responsable
Juan-Manuel Torres

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Présentation du TP 2

Ce TP numéro 2 est une suite logique du TP numéro 1 Approches Neuronales où on avait programmé deux types de perceptrons, version Batch et Online, et fait leurs comparaisons. L'objectif de ce TP 2 est d'abord d'apprendre et de tester sur l'un des perceptrons cités au dessus, avec des données des échos de sonar codées en 60 dimensions, et ensuite de calculer les erreurs d'apprentissage, de généralisation et les stabilités.

2 PARTIE I

2.1 Choix du Perceptron Batch ou Online

Nous avons choisit le perceptron **Version Batch** car d'aprés notre **TP 1**, cette version présente **moins d'itération** et le **temps de convergence** est **plus rapide** que celui de la version online

2.2 Apprentissage sur « train »

2.2.1 Calculer les erreurs d'apprentissage Ea et de généralisation Eg

Nous avons utilisé une fonction nommée erreur pour calculer directement le nombre de données mal classée.
Ainsi le calcule de l'Ea se fait sur les données d'entrainement et l'Eg sur les données de test.

```
def erreur(X, W, tau):
    ones = np.ones((len(X),1))
    X_new = np.append(ones,X,axis=1)
    y = np.sign(X_new@W)
    return np.sum(y != tau)
```

Apres Apprentissage, l'**erreur d'apprentissage** est égale à **0** et l'**erreur de généralisation** est égale à **21**.
Cela montre que 21 sur 104 valeurs (données) ont été mal classées.

2.2.2 Afficher les N+1 poids W du perceptron

Le poid W de convergence aprés apprentissage est :
W = [-97.59370889 83.30464029 210.82783467 38.33899339 110.29577166 72.7514072 143.85641112 -64.67613896 -158.14848582 8.56377583 107.44613065 39.40800534 13.52500482 2.50607866 4.19112908 58.80201602 -194.93536667 -55.61782494 202.58424617 -37.96450241 -10.16647537 -39.16076343 59.9656041 -44.44622684 107.910952 -11.86599739 -13.00557167 11.1672445 -20.50451042 -19.22759559 204.11702886 -305.38951699 232.01181951 -159.90950211 73.85068469 -40.24848693 -21.08138266 -38.52775193 -20.81968008 170.183523 -152.3339456 63.10641422 -13.86356271 -11.56570923 -65.76728597 119.45601557 104.03837512 -106.05093168 209.89522295 289.03608273 -128.81707905 52.98836151 51.56759829 29.77964848 82.43212574 -12.21430306 -23.21586375 -54.39944421 33.71965812 -14.12824508 -16.5430189]

2.2.3 Calculer les stabilités des P exemples de « test »

Ci-dessous nous avons les valeurs des stabilités.
Stabilités = [-6.21353647e-02 -4.92966037e-02 -2.30270025e-02 3.14309132e-02 -3.19755832e-02 5.62555025e-02 -4.86023185e-03 1.26925245e-01 4.45395576e-02 -5.97232762e-03 8.42783574e-02 7.97783454e-03 5.10329373e-02 2.95948065e-02 1.20192652e-01 -1.57389186e-02 4.09750941e-02 4.08783739e-02 1.46077748e-01 1.19935441e-01 -1.03636861e-01 6.78805279e-02 6.50503639e-02 4.63041025e-02 1.67234964e-01

6.33628871e-02 9.61051298e-02 1.18828995e-01 -1.22682485e-02 3.71999831e-02 7.06213265e-03 -8.10850686e-03 5.41145164e-02 4.93263226e-02 2.18186170e-02 1.02723003e-01 -1.42250746e-02 -2.47051105e-03 4.78446252e-02 7.30053185e-02 7.92521359e-02 1.29746675e-01 3.77542438e-02 2.01961821e-01 1.23880986e-01 1.63501778e-01 1.27997115e-01 9.78092417e-02 3.04321254e-02 4.00104654e-02 4.63163476e-02 8.03160730e-03 -2.89009856e-02 5.92644884e-02 6.10070572e-02 8.19711507e-02 1.31994333e-01 5.85436912e-02 6.02041888e-02 5.75927269e-02 5.95990558e-02 7.89290958e-03 5.79093257e-02 7.11173726e-02 1.09751992e-01 -7.47470157e-02 -9.10942193e-02 -1.08799663e-01 3.26849322e-03 -5.17431339e-05 5.68900183e-02 6.76740777e-02 4.51877403e-02 1.40277523e-02 3.92941569e-02 3.06648161e-02 5.21905180e-02 8.98627063e-02 2.81914015e-02 2.47280814e-02 3.39278387e-05 -9.18494334e-03 2.51981333e-02 3.64980023e-02 2.15586346e-02 7.54218931e-03 9.45551331e-02 1.74259519e-01 1.47378521e-01 5.38024241e-02 5.05044836e-02 4.68000283e-02 -4.29066009e-02 1.35895146e-02 1.07681982e-01 8.57065389e-02 -5.80053116e-02 -9.73083727e-03 6.55667850e-03 6.53287793e-02 3.27571739e-02 1.10573683e-01 2.81647661e-02 5.18223001e-02]

2.2.4 Graphique des stabilités

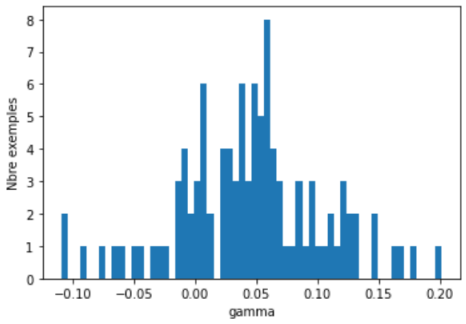

Figure 1. graph des stabilités

Nous voyons que la **somme de la valeur des sommets** des **gamma négatifs** c'est à dire compris entre **-0.20** et **0.00**, donne la valeur de l'erreur de généralisation **Eg** qui est égale à **21**.
Nous constatons que la majorité des valeurs sont positives c'est à dire comprise entre **0.00** et **0.21**, ce qui prouve la **robustesse** de la reponse de notre perceptron.

2.3 Apprentissage sur « test »

2.3.1 Calculer les erreurs d'apprentissage Ea et de généralisation Eg

Apres Apprentissage sur le test, l'**erreur d'apprentissage** est égale à **0** et l'**erreur de généralisation** est égale à **25**.
On note que l'Ea reste toujours à **0**, mais l'Eg dans l'apprentissage des données test est **plus grand** que l'Eg sur l'apprentissage des données train

2.3.2 Afficher les N+1 poids W du perceptron

Ci-dessous, les poid W de convergence apres apprentissage sur le test.
W = [-33.65112672 7.19725893 -11.74397874 -8.3035814 11.78705969 6.01989328 -32.93488584 -60.70618729 -3.05916717 44.35681454 17.49730736 71.49114453 50.10710941 3.96544384 -12.19231866 -35.44336526 19.82225472 2.12148628 -19.31942211 -5.93216991 10.99157877 17.16659572 6.07018521 17.82353756 19.06983772 -26.74867365 -27.70352363 3.41511336 22.7109556 1.06339953

1.8664158 -21.91364095 9.61357557 -7.49326281 -27.48332131 14.82642768 -17.30023808 -62.64277029 -5.95571954 35.88156807 -25.95409777 12.19657435 15.86401694 48.62400464 63.41334212 34.84509099 4.43575139 9.1024473 24.72505213 0.22881202 -5.27736675 9.6045295 7.15220938 1.92156251 2.48839652 -0.46399257 0.82518071 3.22636088 1.8421621 6.86427327 0.5105958]

2.3.3 Calculer les stabilités des P exemples de « test »

Ci-dessous les valeurs des stabilités :
Les stabilités = [-0.14128419 0.0578742 0.20087521 0.04358099 0.01936641 -0.07740469 0.01098802 -0.00192776 -0.00604403 0.04268051 0.06043549 0.14384973 0.13702551 0.15424662 0.21995571 0.09073517 0.12647446 -0.12481542 0.07509058 0.42422134 0.43117858 0.27173263 0.35490334 0.1594452 0.14977166 -0.04739078 0.03516267 0.07253381 0.03860439 0.17177444 0.21195931 0.16604681 0.03498308 0.02969741 0.10978808 0.03475159 0.32286294 0.02784832 -0.06070109 0.07846289 0.56711351 0.47339317 0.20385718 0.02698075 0.02115268 0.2514257 0.26936162 0.10152144 0.09674821 -0.02972553 -0.03104042 -0.30718827 -0.05162925 -0.19529549 0.08495525 -0.16204934 -0.04540812 -0.01027952 0.25348779 0.11286712 0.02574703 0.06980861 0.29034375 0.10312939 0.20293849 -0.32037597 -0.00599421 0.15566989 0.33632686 0.08774782 -0.00733532 0.00379961 -0.14277887 0.05571046 0.22737295 -0.13302262 -0.06112117 0.12595708 0.09160163 0.26509763 0.35527997 -0.22168176 0.03119935 -0.1789188 0.02778611 0.02917777 0.24644162 0.22823312 0.26662519 0.19795418 0.18666548 0.06052042 0.14928749 0.13122288 0.10585331 -0.06628169 0.03229103 -0.02505704 0.15688459 0.15646718 0.0946558 0.03467463 0.01766968 0.30106275]

2.3.4 Graphique des stabilités

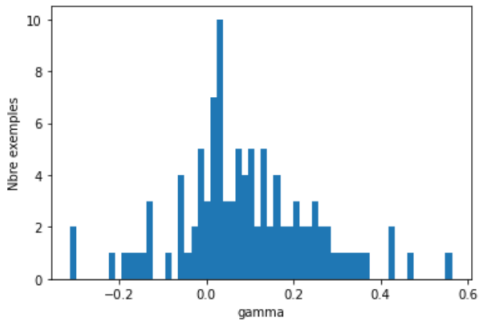

Figure 2. graph des stabilités

3 PARTIE II

3.1 Algorithme d'apprentissage Pocket

L'apprentissage Pocket est un apprentissage qui conserve le meilleur résultat pendant l'entrainement. Cela permet d'éviter de revenir en arrière si le modèle se trompe en choisissant un nouveau poids.
Ci-dessous, nous avons le code de l'algorithme d'apprentissage Pocket qui stop l'apprentissage pour une erreur égale à Ea-limit.

```
def apprentissage_pocket(P,N,X,tau,alpha,nb_epochs,Ea_limit,W_init):
    W = W_init #initialisation de poids
```

```
W_pocket= W.copy() # initialisation du vecteur de poids de poche
error_min = len(tau) #initialisation error à sa plus grande valeur
for k in range(nb_epochs) :
    dW = np.zeros((N+1,1))
    ones = np.ones((P,1))
    X_new = np.append(ones,X,axis=1)
    y= np.sign(X_new@W)
    if not np.array_equal(y,tau): # si la valeur predicte est different du
        tau(valeur relle)
        dW = X_new.T@(tau - y)  #alors on met a jour le poids
        W = W + dW
        error = erreur(X,W,tau) # et on calcule l'erreur
    if error < error_min : #on campare si l'erreur est plus petit que l'erreur min
        error_min = error # on met a jour l'erreur min à l'erreur trouver
        W_pocket = W.copy()       # et on copie aussi le poids sur W_pocket

    if error_min <= Ea_limit : #critere d'arret à un Ea fixé
        break
return W_pocket
```

3.2 initialisation aleatoir vs initialisation Hebb

Nous avons établi deux initialisations **aléatoire** et **hebb** distinctes appliquées à l'algorithme d'apprentissage Pocket avec **Ea fixé** à **10**, **8** et à **6**.
Nous avons calculé aussi les differentes valeurs à savoir Ea, Eg et stabilités en fixant Ea à des valeurs limites et nous avons fait varié l'hyperparamètre **alpha** entre **[0.1 0.7 1]** qui aura un impacte sur **delta**.
Ainsi nous pouvons voir en detaille les valeurs **Ea** et**Eg** en fonction des ensembles train et test, des initialisations aléatoire et hebb et en fonction de alpha.

3.3 Entrainement sur train et testé sur test

3.3.1 Initialisation aléatoire

Le résultat des differentes valeurs Ea et Eg sur les données train 1.

	Alpha = 0.1	Alpha = 0.7	Alpha = 1
	Ea ; Eg	Ea ; Eg	Ea ; Eg
Ea fixe = 10	10 ; 23	10 ; 24	10 ; 24
Ea fixe = 8	8 ; 25	8 ; 25	8 ; 24
Ea fixe = 6	6 ; 25	6 ; 24	6 ; 25

Table 1. initialisation Aléatoire : Tableau de Ea et Eg en fonction de Ea limite fixé et Alpha

3.3.2 Initialisation Hebb

Le résultat des differentes valeurs Ea et Eg sur les données train 2.

3.4 Entrainement sur test et testé sur train

3.4.1 Initialisation aléatoire

Le résultat des differentes valeurs Ea et Eg sur les données test 3.

	Alpha = 0.1	Alpha = 0.7	Alpha = 1
	Ea ; Eg	Ea ; Eg	Ea ; Eg
Ea fixe = 10	10 ; 25	10 ; 25	10 ; 25
Ea fixe = 8	8 ; 24	8 ; 24	8 ; 24
Ea fixe = 6	6 ; 24	6 ; 24	6 ; 24

Table 2. initialisation Hebb : Tableau de Ea et Eg en fonction de Ea limite fixé et Alpha

	Alpha = 0.1	Alpha = 0.7	Alpha = 1
	Ea ; Eg	Ea ; Eg	Ea ; Eg
Ea fixe = 10	10 ; 28	10 ; 28	10 ; 29
Ea fixe = 8	7 ; 27	8 ; 29	6 ; 27
Ea fixe = 6	4 ; 27	6 ; 27	6 ; 26

Table 3. initialisation Aléatoire : Tableau de Ea et Eg en fonction de Ea limite fixé et Alpha

3.4.2 Initialisation Hebb

Le résultat des differentes valeurs Ea et Eg sur les données test 4.

	Alpha = 0.1	Alpha = 0.7	Alpha = 1
	Ea ; Eg	Ea ; Eg	Ea ; Eg
Ea fixe = 10	10 ; 30	10 ; 30	10 ; 30
Ea fixe = 8	8 ; 28	8 ; 28	8 ; 28
Ea fixe = 6	6 ; 27	6 ; 27	6 ; 27

Table 4. initialisation Hebb : Tableau de Ea et Eg en fonction de Ea limite fixé et Alpha

3.5 Interprétation

Pour Alpha fixe, on constate que l'erreur **Eg** de généralisation **diminue** quand l'erreur d'apprentissage **Ea fixé diminue**.
Certes dans la majorité des cas notre algorithme atteint l'erreur d'apprentissage fixé mais cela n'empêche pas une diminution de l'erreur de généralisation.

4 PARTIE III

4.1 L'ensemble L= Train + Test est t-il LS ?

On peux constaté que l'ensemble L = Train + Test est lineairement séparable car l'erreur d'apprentissage **Ea** est égale à **0**. Il y'a aussi une itération fini et le poids de perceptron converge vers le poid égale à
W = [-3.26617026e+02 8.73799784e+02 4.34999214e+01 -1.29237298e+03 5.99333311e+02 -2.55629489e+02 3.55299880e+02 -4.69368360e+02 -3.54630910e+02 4.91876317e+02 -1.96076216e+02 1.98801620e+02 4.50038493e+02 -2.20796268e+02 2.20946649e+01 1.45469708e+02 -2.20618793e+02 -2.29072847e+02 2.61878091e+02 -2.01613131e+02 4.84909111e+02 -5.39340345e+02 6.37373605e+02 -4.51362472e+02 5.09268290e+02 -2.30375047e+02 -7.60818924e+01 1.94063142e+02 -1.01441103e+02 -7.88455277e+01 4.93302117e+02 -6.30123365e+02 2.21920919e+02 1.84195705e+02 -3.27011611e+02 2.46116898e+02 -3.15145047e+01 -3.33971590e+02 8.11280096e+01 2.79331935e+02 -3.81805634e+02 8.60313891e+01 1.19951266e+02 5.10347252e+01 1.06497384e+02 -1.05982993e+02 2.51584773e+02 2.56409602e+00 7.37796260e+02 1.07976023e+03

-4.52563978e+03 1.53339753e+03 2.66057341e+03 1.79698385e+03 3.73503750e+02 1.74188812e+02 -1.18806436e+03 -8.97523448e+02 1.38332544e+03 8.82232181e+02 1.90667325e+02].

5 Conclusion

Ce projet nous a permis d'apprendre notre algorithme de perceptron codé dans TP 1 sur des données d'entrainement et de la tester sur des données de test.
Nous avons aussi fait connaissance avec les differents types d'initialisation à savoir l'initialisation aléatoire et hebb avant de faire leures comparaisons.