

4A : RESEAUX DE NEURONES

Nom et Prénom : SHE Houyu, MOUDOUB Lila, GUIZANI Ahmed

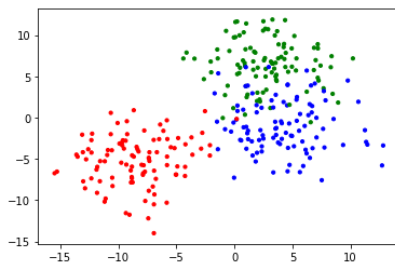
Classe: 4A41

TP2 (durée : 3h)

Apprentissage de réseaux de neurones pour la classification

1. Réseau mono-couche pour la classification

- a) Charger la distribution de données *dataset*. Préciser le nombre d'observations, leurs dimensions et le nombre de classes. Séparer les données en bases d'apprentissage et de test avec un ratio de 70/30. Afficher les données.



- Nombre de dimensions : 2
- Nombre de classes : 3 (1,2,3)
- Nombre d'observations : 300 (210 d'apprentissage, 90 de test)

- b) Définir un réseau de neurones classifieur **mono-couche** :

`clf1=SGDClassifier(loss='perceptron', eta0=A, max_iter=B, learning_rate='constant', verbose=1)`

A représente le pas d'apprentissage et **B** le nombre d'itérations, que vous saisissez avant de définir le réseau. Préciser le nombre de paramètres libres (poids et biais) du réseau.

- **Poids = $300 \times 3 = 900$**
- **Biais = 3**
- **Le nombre de paramètres libres = 903**

- c) Entraîner ce réseau (fonction *fit*) en utilisant successivement les paramètres suivants :

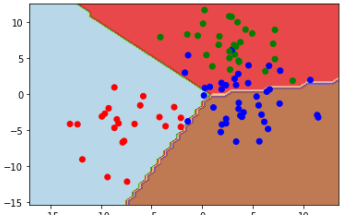
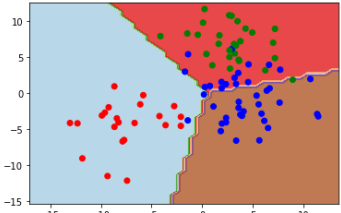
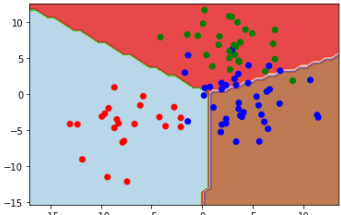
- A = 0.000001 B = 1
- A = 1 B = 1
- A = 0.000001 B = 10
- A = 1 B = 10

La comparaison pourra se faire sur plusieurs points.

- le taux de reconnaissance (fonction *score*) en apprentissage et en test, obtenu en utilisant la règle de décision WTA (*Winner Takes All*).

- la forme des frontières entre les classes.

<i>max_iter=B</i>	<i>PAS = A</i>	Taux de reconnaissance	Frontières de décision
1	0.000001	TR app : 0.91% TR test : 0.84%	

1	1	TR app : 0.9% TR test : 0.8%	
10	0.000001	TR app : 0.9% TR test : 0.8%	
10	1	TR app : 0.89% TR test : 0.8%	

Conclure sur l'impact de ces paramètres (ou tenter d'expliquer leur absence d'impact).

- Dans le premier cas, le nombre d'itération et que le pas d'apprentissage est faible, le réseau de neurones est peu sensible à chaque exemple qu'on lui présente. En effet, **0.000001** est un pas d'apprentissage faible et ne va pas trop changer le poids de l'itération.

De plus, avec seulement 1 itérations, l'entraînement va s'arrêter très rapidement et n'aura pas eu le temps de bouger significativement les frontières.

Lorsqu'on passe à un pas d'apprentissage (à 1), on remarque qu'il suffit seulement 10 itérations pour que les frontières se placent "correctement". Le pas d'apprentissage élevé a engendré un changement remarquable dans le poids du réseau durant les 10 itérations. De ce fait, cela a permis de se rapprocher d'un minimum local rapidement.

Pour le cas où on effectue 10 itérations avec un pas d'apprentissage de 1, le taux de reconnaissance ne semble pas s'améliorer, on s'approche de la limite.

Pour conclure, on peut dire que le pas d'apprentissage à une importance sur le changement des poids dans le réseau.

Lors de l'utilisation d'un pas d'apprentissage faible, on peut effectuer une recherche fine des paramètres, cependant, il nous faudra plus d'itération pour trouver un minimum. Au contraire, un pas d'apprentissage élevé permet de changer les poids du réseau de manière plus importante et donc moins précise : cela induit un risque de rater des paramètres optimaux et de diverger. Cependant, cela permet aussi d'aller plus rapidement lorsque l'on se trouve loin d'un minimum.

Il faut trouver un équilibre entre le pas d'apprentissage et le nombre d'itération pour d'avancer rapidement dans la descente du gradient sans diverger.

Les résultats précédents varient-ils lorsque l'on relance plusieurs fois le même apprentissage ? Pourquoi ?

-Il n'y a pas de résultats significatifs en modifiant les valeurs de A et B car entre le taux de reconnaissance par apprentissage et par test, les résultats sont très proches. De plus, les données ne varient pas lorsque l'on relance plusieurs fois le même apprentissage.

Le taux de reconnaissance ne s'améliore pas comme les données ne sont pas linéairement séparables.

Que doit-on envisager pour améliorer les performances ? Justifier votre réponse.

-Il faut donc passer au réseau multi-couche pour améliorer les performances.

2. Réseau multi-couche pour la classification

On travaille sur les mêmes données.

- a) Définir un réseau de neurones classifieur **multi-couche** :

```
clf2 = MLPClassifier(hidden_layer_sizes=C, activation='tanh', solver='sgd', batch_size=1,
alpha=0, learning_rate='constant', max_iter=100, momentum=0)
```

- b) Sensibilité aux hyper-paramètres :

Pour C = 5 :

- Noter la valeur du coût (*loss*) en fin d'apprentissage et le nombre d'itérations effectuées ;
 - la valeur du coût = 0.22
 - le nombre d'itérations effectuées = 100
- Multiplier le pas d'apprentissage par 10 et noter les valeurs précédentes ;
 - la valeur du coût = 0.21
 - le nombre d'itérations effectuées = 55
- Changer le solveur (utiliser '*adam*' au lieu de '*sgd*') et noter les valeurs précédentes ;
 - la valeur du coût = 0.19
 - le nombre d'itérations effectuées = 53
- Conclure sur les meilleurs paramètres à utiliser.

```
clf2 = MLPClassifier(hidden_layer_sizes=C, activation='tanh', solver='adam', batch_size=1, alpha=0,
learning_rate='constant', learning_rate_init = 0.01, max_iter=100, momentum=0, verbose=True)
```

Avec ces paramètres, la valeur du coût et le nombre d'itération sont les plus petits.

- c) Trouver, par recherche exhaustive, le nombre optimal de neurones cachés C^* . pour ce faire, tester les valeurs de C suivantes : 1, 2, 5, 10, 20, 50, 100.
-le nombre optimal de neurones cachés $C^* = 20$

Pour chaque valeur de C, répéter 10 fois l'apprentissage et le test. Calculer les moyennes et écart-type des taux en apprentissage et en test.

Reporter ces résultats dans un tableau, en précisant pour chaque architecture, le nombre de paramètres libres du réseau. Analyser et conclure quant au biais et à la variance.

C	1	2	5	10	20	50	100
Moyenne des taux en apprentissage	0.78	0.91	0.93	0.94	0.94	0.93	0.93
Moyenne des taux en Test	0.7	0.87	0.83	0.82	0.84	0.86	0.84
Écart-type des taux en apprentissage	0.08	0.009	0.008	0.013	0.010	0.009	0.006
Écart-type des taux en et en test.	0.106	0.035	0.026	0.018	0.029	0.035	0.033

3. Conclusions

Comparer les performances obtenues dans les parties 1 et 2. Conclure quant à l'intérêt d'utiliser des modèles parcimonieux.

- Mono Couche :

Pour un modèle compliqué c'est difficile de trouver les bonnes valeurs, problème d'optimisation

- Multi Couche :

La capacité d'apprentissage du Multi Couche est plus élevée pour les modèles compliqués, parce qu'il permet de préciser la caractéristique unique des données selon les résultats obtenus et sa grâce aux couches qu'il contient.

Il faut trouver le nombre de couches optimale afin d'obtenir le meilleur résultat.

Quand on augmente le nombre de couche la capacité d'apprentissage devient meilleur.

Mais si on augmente trop le nombre de couche, on remarque le phénomène du perdu (phénomène de la disparition du gradient en machine learning).
Alors pour des raisons de complexité, une ou deux couches suffisent.

Comparer aux performances obtenues à l'aide de l'algorithme des plus-proches-voisins (TP1) : taux de reconnaissance et temps de calcul. Conclure quant à l'intérêt d'utiliser des réseaux de neurones pour la classification.

- Pour l'algorithme des plus-proches-voisins, c'est un algorithme simple. Cela fonctionne bien pour les modèles non-linéairement séparable. De plus, son temps de calcul est court et nous devons trouver la meilleure valeur de k pour effectuer cet algorithme.
- Pour réseaux de neurones, il a besoin plus de temps de calculs, mais le taux de reconnaissance est plus élevé que l'algorithme knn. Lorsque le nombre est très élevé, on tombe dans le phénomène de perte de gradient, ce qui engendre des résultats erronés.