Mini Projet 1 : Problèmes non linéairement séparable

L'objectif de ce mini – projet est de connaitre comment créer un Multi Layer Neural Network pour mettre en place un modèle de classification ayant un accuracy élevé relativement à un problème non linéairement séparable. Trouver un modèle d'accuracy élevé ce n'est pas toujours une tâche facile et dépend de la nature du dataset. Si on considère que le dataset est bien préparé, l'accuracy reste sensible à plusieurs facteurs :

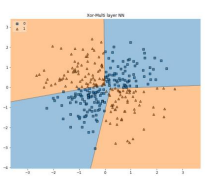
- Initialisation des poids du modèle avant l'entrainement
- Le learning rate choisi
- L'optimizer choisi (Batch Dradient Descent – mini batch GD, SGD, Adam
- La manière de mise à jour des poids : les éléments à considérer : gradient, learning rate,…

En résumé, l'objectif est de découvrir d'une manière profonde comment l'accuracy (ou autre métrique) évolue en fonction des itérations et comment assurer une mesure de performance élevée

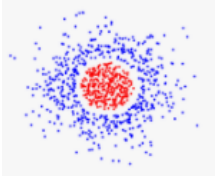
Mise en situation : non linear learning, concepts de base

1. Que représente un problème non linéairement séparable ? examiner les figures ci-dessous
2. Pourquoi un réseau avec un seul neurone ne peut pas être adapté à ce genre de problème ?
3. Comment adapter un Neural Network pour résoudre ce genre de problème en considérant les éléments suivants : Architecture choisie, Initialisation, Learning rate, optimizer

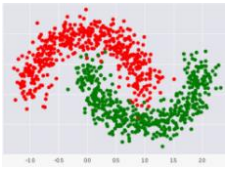
On considère les cinq problèmes non linéairement séparables suivants :



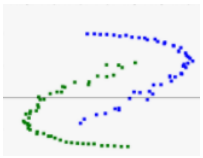
Problem1



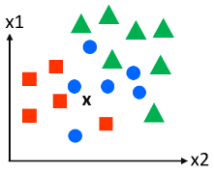
Problem2



Problem3



Problem4



Problem5

Etape 1 :

Pour le problème de **XOR (Problem1)**,

En se référant au résultat obtenu à l'aide de SVM (voir annexe-program1),

- Créer le dataset et le visualiser (voir le data set généré en annexe-program1) et voir annexe-installation
- Split data : X_train, X_test, y_train,y_test
- Créer un réseau de neurones en utilisant keras-tensorflow ayant les éléments suivants :
  - La couche d'entrée
  - Une couche cachée ayant deux neurones
  - La couche de sortie
  - Faire les initialisations qu'il faut
  - Lancer l'apprentissage
  - Evaluer le modèle obtenu. Quelle accuracy a été obtenu
- Améliorer la performance du modèle en précisant et justifiant les pratiques suivies en tenant en compte les éléments suivants :
  - On considère que le dataset est bien préparé (pas de transformations à faire relativement au dataset)
  - Garder la même architecture

- Vanishing problem, est ce qu'il a eu lieu, justifier (consulter l'évolution de l'apprentissage)? quelles pratiques à suivre pour l'éviter ?
- L'optimizer choisi, quel algorithme le mieux adapté
- Le learning rate, est ce qu'il a un effet sur l'apprentissage, comment le choisir,…
- Le nombre d'epochs
- Autres problèmes à soulever dépendant de problème
- Autres pratiques à appliquer
- Quel est le meilleur score obtenu, se comparer avec annexe-résultat1. Commenter le score obtenu.

Etape 2 :

- Créer un réseau de neurones ayant les éléments suivants :
  - La couche d'entrée
  - Une couche cachée ayant **quatre** neurones
  - La couche de sortie
  - Faire les initialisations qu'il faut
  - Lancer l'apprentissage
  - Evaluer le modèle obtenu. Quelle accuracy a été obtenu
- Améliorer la performance du modèle en précisant et justifiant les pratiques suivies en tenant en compte les éléments suivants :
  - On considère que le dataset est bien préparé (pas de transformations à faire relativement au dataset)
  - Garder la même architecture
  - Vanishing problem, est ce qu'il a eu lieu, justifier (consulter l'évolution de l'apprentissage)? quelles pratiques à suivre pour l'éviter ?
  - L'optimizer choisi, quel algorithme le mieux adapté
  - Le learning rate, est ce qu'il a un effet sur l'apprentissage, comment le choisir,…
  - Le nombre d'epochs
  - Autres problèmes à soulever dépendant de problème
  - Autres pratiques à appliquer
  - Quel est le meilleur score obtenu, se comparer avec annexe-résultat1. Commenter le score obtenu.

Etape 3 :

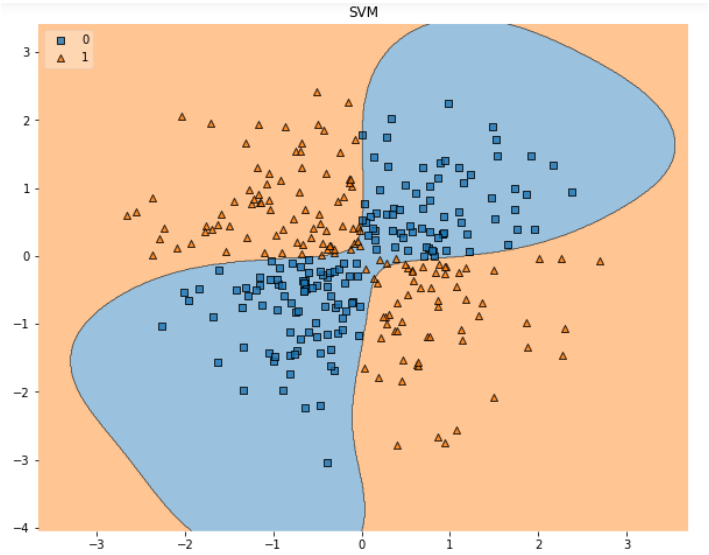
Trouver le modèle adéquat qui concerne Problem 2 et un modèle qui concerne Problem3 tout en suivant des bonnes pratiques

- Iitialiser le dataset
- Créer le modèle
- Entrainer le modèle
- Analyser la performance
- Soulever les problèmes associés à chaque cas
- Justifier les choix effectués : architecture, hyperparamètres, optimizers…

Annexe :

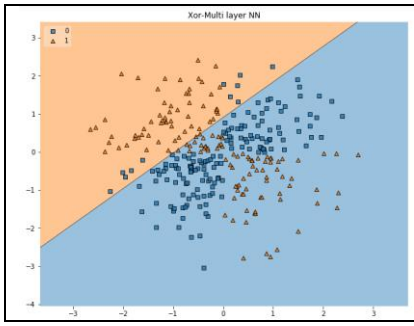
```
#Program1:
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
import numpy as np

rng = np.random.RandomState(0)
X = rng.randn(300, 2)
y = np.array(np.logical_xor(X[:, 0] > 0, X[:, 1] > 0),
          dtype=int)

svm_model = SVC(gamma='auto')
svm_model.fit(X, y)

fig = plt.figure(figsize=(10,8))
fig = plot_decision_regions(X=X, y=y, clf=svm_model, legend=2)
plt.title("SVM")
plt.show()
```

Le programme ci-dessus donne le résultat suivant :

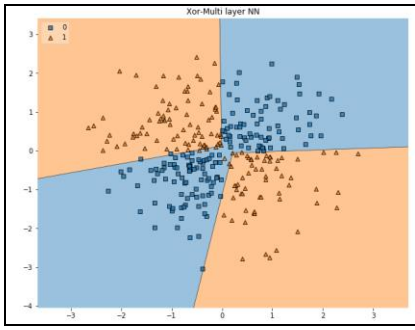


Résultats :

Résultat 1 – réseau de neurones avec une couche cachée avec deux neurones

	`['loss', 'accuracy']` `[0.578, 0.689]`

Résultat 2 – réseau de neurones avec une couche cachée avec quatre neurones

	['loss', 'accuracy'] [0.1009, 0.970]

Annexe - installation

Pour afficher le graphique de Xor essayer d'installer **mlxtend** à l'aide de la commande ci-dessous

```
(base) C:\Users\NAJI>conda install -c conda-forge mlxtend
```