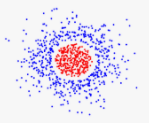
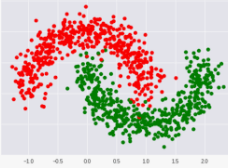
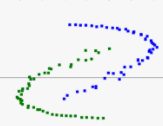
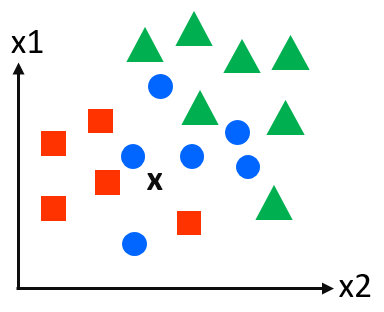
|  |  |  |  |
| --- | --- | --- | --- |
| **1.Neurone** | **2.Layers** | **3.Activation Functions** | **4.Optimizers** |

L’objectif de cet atelier est de connaitre comment créer un Multi Layer Neural Network et de découvrir ces différents éléments et traitements

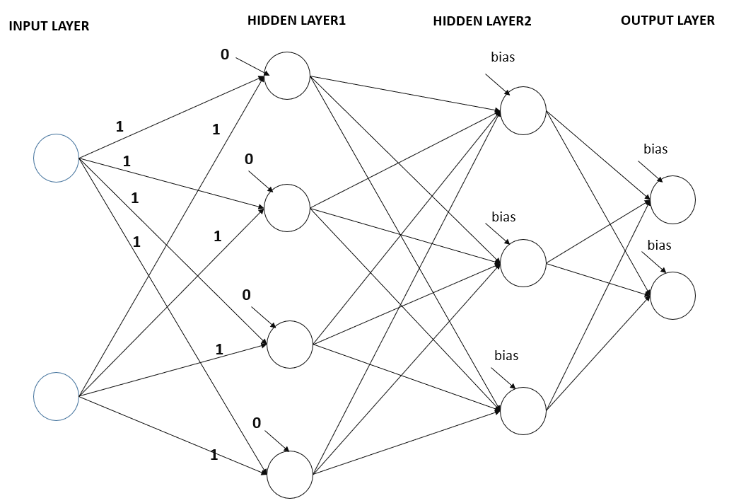
**Exercice 1 : non linear learning, concepts de base**

* Que représente un problème non linéairement séparable ? examiner les figures ci-dessous
* Pourquoi un réseau avec un seul neurone ne peut pas être adapté à ce genre de problème ?
* Comment adapter un Neural Network pour résoudre ce genre de problème ?

**Exercice 2 : Multi layer neural network, Keras, concepts de base**

1. Rappeler les éléments d’un neurone
2. Quel est le rôle d’une fonction d’activation.
3. Si nous avons une classification binaire, quelle fonction d’activation à choisir dans la couche de sortie ?
4. En général, quelle est la fonction d’activation à mettre dans les couches cachées ?
5. Rappeler les éléments d’un Neural Network multilayer
6. Sans exécuter le script, dessiner le réseau de neurones obtenu à l’aide du program1
7. Quel est le nombre de poids obtenus (weights) de chaque couche tout en donnant le nombre de bias de chaque couche ? Quel est le nombre total de poids ?
8. A l’aide de la fonction summary() afficher le détail du réseau de neurone. Vérifier la réponse de la question 5.
9. Ecrire le code source permettant de créer le réseau de neurones de la figure ci-dessous



**Exercice 3: Xor Problem**

Voir le code source du program2 en annexe.

Program pour séparer un problème XOR basé sur support vector machine classifier (SVC)

1. Analyser le code source.

Pour afficher le même résultat, essayer d’installer **mlxtend** à l’aide de la commande ci-dessous



L’objectif de cet exercice est de trouver un modèle basé sur **Neural Network** pour séparer des données dispersées sous format de XOR. (voir les deux figures ci-dessous qui sont obtenus à l’aide de modèles différents accompagnées de la performance)

1. Créer un réseau de neurone ayant trois couches (le nombre de neurones de la couche cachée est 2)
   * Définir les fonctions d’activation à utiliser
   * Préciser l’optimizer du modèle
   * Faire l’apprentissage en changeant le nombre d’epochs
   * Afficher les metrics Loss et accuracy
   * Dessiner le graphe (résultat1) (pas forcément d’avoir exactement le même résultat)
   * En observant l’exécution de la phase d’entrainement, donner une définition simplifiée de vanishing problem
2. Créer un réseau de neurones ayant trois couches (le nombre de neurones de la couche cachée est 10)
   * Définir les fonctions d’activation à utiliser
   * Préciser l’optimizer du modèle
   * Faire l’apprentissage en changeant le nombre d’epochs
   * Afficher les metrics Loss et accuracy
   * Dessiner le graphe (résultat2) (pas forcément d’avoir exactement le même résultat)
3. Faire les conclusions qu’il faut

**Résultat 1 :**

|  |  |
| --- | --- |
|  | ['loss', 'accuracy']  [0.578, 0.689] |

**Résultat 2 :**

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

Annexe :

**#Program1**

from keras.models import Sequential

from keras.layers import Activation, Dense,Flatten

model = Sequential()

a=Dense(5,input\_dim=3)

model.add(a)

b=Dense(4)

model.add(b)

c=Dense(6)

model.add(c)

d=Dense(2)

model.add(d)

print(model.summary())

**#Program2:**

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 :

