# TP N°11 – INTELLIGENCE ARTIFICIELLE

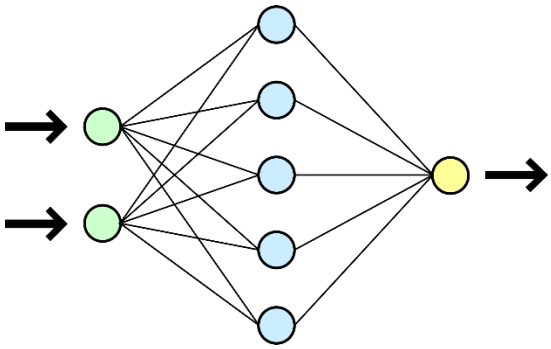
## Objectifs du TP

- Dans ce TP, nous allons mettre en œuvre une librairie d'intelligence artificielle appelée NEUROPH.

## Reprenez le projet approche-objet

## Etape 1 : Théorie

***Un peu de théorie sur les réseaux de neurones.***



Le réseau de neurone est une simulation du fonctionnement du cerveau humain, mais il ne prétend pas fonctionner comme le cerveau humain.

***« le réseau de neurones est au cerveau, ce que l'aile d'avion est à l'aile d'un oiseau ».***

Un réseau de neurones élémentaire de type perceptron (perceptron avec 1 couche intermédiaire) est constitué de 3 couches :

- les neurones d'entrée (en vert)
- les neurones cachés (en bleu)
- les neurones de sortie (en jaune).

Les neurones d'entrée, comme leur nom l'indique, servent à fournir en entrée des valeurs numériques (double en java).

Les neurones de sortie fournissent des valeurs en sortie (double en java). Il peut y avoir plusieurs sorties si besoin.

La couche de neurones cachés est gérée par le réseau de neurones.

**Les poids entre neurones**

Chaque **liaison** entre neurones porte ce qu'on appelle un **poids** qui est une valeur comprise entre 0 et 1.

En fonction des valeurs d'entrées, et des valeurs des poids, le réseau de neurones va être capable de fournir une ou plusieurs valeurs de sortie.

Toute la difficulté va être d'ajuster ces poids (par l'apprentissage du réseau) afin que le réseau fournisse des valeurs de sortie correctes pour des valeurs d'entrée données.

**Le réseau de neurones au départ**

Au départ lorsqu'on créé un réseau de neurones, ce dernier est comme un cerveau qui n'a pas d'expérience. Les poids sont initialisés à une valeur aléatoire comprise entre 0 et 1.

Si on lui fournit des valeurs d'entrée, il va fournir un résultat calculé sur la base de ces poids, et qui ne sera pas du tout le résultat attendu.

⇨ Avant de pouvoir utiliser un réseau de neurones, il faut donc faire son **apprentissage**.

**L'apprentissage**

Pour faire **l'apprentissage** d'un réseau de neurones il faut lui fournir des valeurs d'entrée et lui indiquer quelles doivent être dans ce cas précis les valeurs de sortie attendues.

Lors de la phase d'apprentissage :

- le réseau commence par calculer les sorties qu'il obtient avec les poids actuels.
- il calcule alors l'écart (ou erreur) entre les valeurs de sortie qu'il obtient et les valeurs qu'il aurait dû obtenir et qui lui sont fournies.
- il recalcule alors les poids de ses liaisons. C'est ce qu'on appelle la **rétro-propagation**.
- Après cette première rétro-propagation, il ne faut pas s'attendre à ce que le réseau de neurones fournisse des valeurs justes pour les valeurs d'entrée précédentes. En effet, l'apprentissage procède par petites variations des poids et ne fais pas de modification brutale.
- L'apprentissage peut nécessiter un grand nombre d'itérations, tout dépend de ce qu'on veut enseigner au réseau de neurones.

**Comment savoir lorsque l'apprentissage est terminé ou non ?**

Il faut faire des essais et tant que les valeurs de sortie ne sont pas celles attendues, on continue. Si vous n'arrivez jamais à obtenir les résultats attendus cela peut signifier :

- soit que votre jeu d'essais est insuffisant
- soit que la couche cachée ne possède pas assez de neurones cachés pour tenir compte de la complexité.
- soit que le réseau a besoin de plusieurs couches de neurones cachés pour multiplier le nombre de liaisons et donc de poids.

Il n'existe pas de calcul mathématique savant permettant de vous aider à prévoir à l'avance le nombre de cycles d'apprentissage nécessaire au bon apprentissage d'un réseau. Cette étape est purement empirique.

**Combien de neurones ai-je besoin pour une problématique donnée ?**

Là encore c'est à vous de réaliser des tests pour trouver le nombre de neurones optimal, c'est-à-dire qui ne nécessite pas un trop grand cycle d'apprentissages et qui va être capable de retourner des résultats très précis.

**Exemple du deep learning**

Jusqu'en 2012, on pensait qu'il était impossible de réaliser l'apprentissage de réseaux de neurones possédant trop de couches de neurones cachés, ou réseaux de neurones profonds.

Tout a changé en 2012, lorsqu'un français, appelé Yann le Cun, a gagné un concours de reconnaissance d'images avec une IA basée sur un réseau de neurones profond, ou deep-learning.

Non seulement il a gagné ce concours mais en plus il l'a fait en explosant littéralement ses concurrents.

**Passons au TP page suivante…**

## Etape 2 : Pratique

***Commençons par nous familiariser avec les classes de la librairie Neuroph.***

- Pour créer un réseau de neurones avec **Neuroph**, nous allons utiliser la classe **MultiLayerPerceptron** qui s'instancie avec 3 paramètres :
  - le nombre de neurones d'entrée
  - le nombre de neurones de la couche cachée
  - le nombre de neurones de sortie
- Pour réaliser l'apprentissage du réseau de neurones, il faut créer des jeux d'essais avec la classe **DataSet**. Un DataSet s'instancie avec 2 valeurs :
  - le nombre de valeurs d'entrée
  - le nombre de valeurs de sortie
  - Par défaut un DataSet est vide, il faut lui ajouter des jeux d'essais. Chaque jeu d'essai est représentée par la classe **DataSetRow**
- Une fois la classe DataSet complétée, on invoque la **méthode learn** sur le perceptron comme ci-dessous

**Exemple complet d'apprentissage :**

```
MultiLayerPerceptron neuralNetwork = new MultiLayerPerceptron(a, b, c);

DataSet dataSet = new DataSet(a, c);
DataSetRow rOne = new DataSetRow(new double[] {e1, e2}, new double[]{s1});
DataSetRow rTwo = new DataSetRow(new double[] {e3, e4}, new double[]{s2});
DataSetRow rThree = new DataSetRow(new double[] {e5, e6}, new double[]{s3});
DataSetRow rFour = new DataSetRow(new double[] {e7, e8}, new double[]{s4});
dataSet.addRow(rOne);
dataSet.addRow(rTwo);
dataSet.addRow(rThree);
dataSet.addRow(rFour);
neuralNetwork.learn(dataSet) ;
```

**Vérifiez les résultats :**

- La méthode **setInput(double… entrees)** permet de définir les valeurs d'entrée.
- La méthode **void calculate()** permet de lancer un calcul.
- la méthode **double[] getOutput()** permet de récupérer l'ensemble des valeurs de sortie sous forme d'un tableau de double.
- **Exemple complet d'affichage de résultats :**

```
neuralNetwork.setInput(e3,e4);
neuralNetwork.calculate();
for (double output: neuralNetwork.getOutput()){
        System.out.println(output);
}
```

## Etape 3

***Insiprez-vous du code ci-dessus pour réaliser votre propre réseau de neurones simulant la porte logique OU.***

**Commitez vos développements sur GitHub**