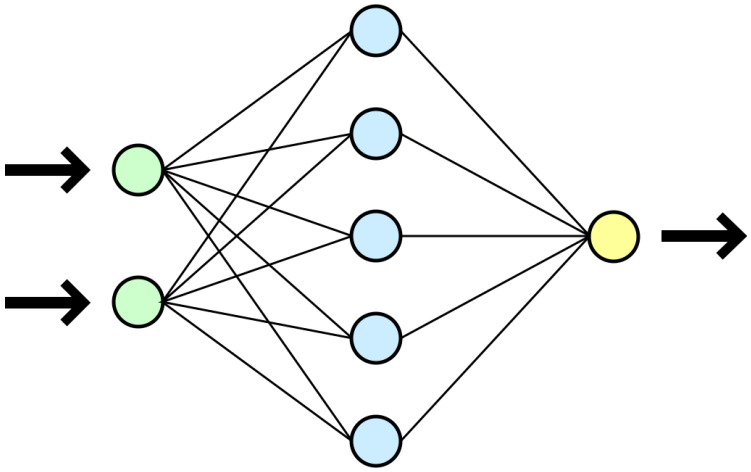
TP : réseaux de neurones

Nous allons, dans ce TP, créer, entraîner puis utiliser un petit réseau de neurones (quelques couches). Nous allons créer des couches de type *Fully Connected*, c'est à dire que chaque neurone d'une couche est connecté à tous les neurones de la couche suivante.

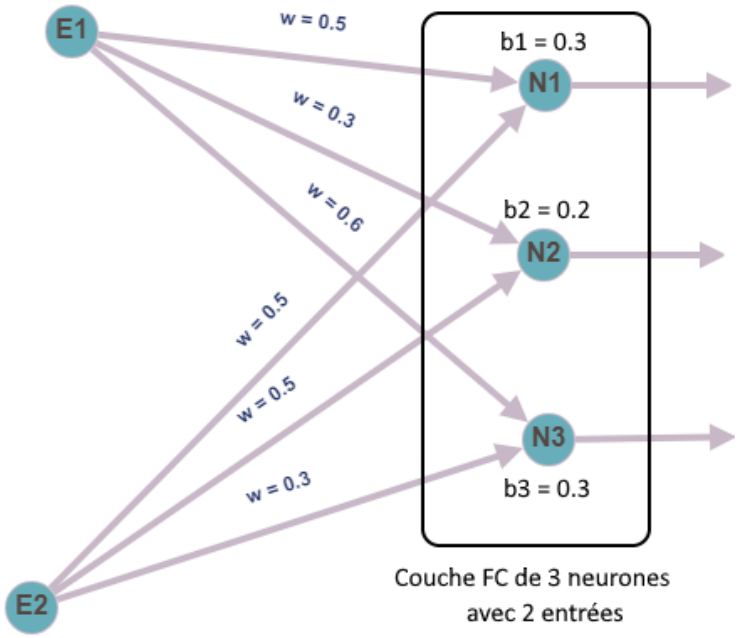


1 Choix d'une structure de couche

Nous n'allons pas créer les neurones un par un mais travailler par couche. Une couche sera un objet Python ayant comme attributs :

- input : list → les valeurs d'entrées (tableau de 0 par défaut)
- output : list → les sorties (tableau de 0 par défaut), il y en a une par neurone de la couche
- bias : list[float] → les biais de chaque neurone
- weight : list[list[float]] → les poids de chaque entrée pour chaque neurone. J'ai choisi de mettre une ligne par neurone (ce n'est pas le plus standard), chaque valeur de la ligne correspondant au poids d'une entrée pour ce neurone

Pour simplifier la gestion de la rétro-propagation l'activation des neurones se fera dans une couche séparée.
Considérons la couche suivante :



1. Écrire en Python le tableau des poids et celui des biais de cette couche
2. Quel calcul doit-on faire pour trouver la sortie du neurone 1 ? (on prendra E1 = 1 et E2 = 2)
3. Compléter le tableau des sorties, on s'en servira pour tester notre implémentation.

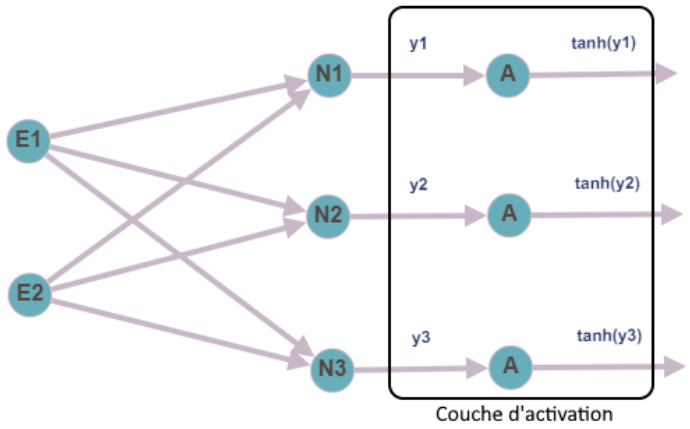
Chaque couche possèdera les 3 méthodes suivantes :

- le constructeur de classe, cette méthode crée les tableaux de 0 pour les entrées et les sorties et elle crée aussi les poids et les biais en prenant des valeurs aléatoires entre -0.5 et +0.5.
- forward_propagation(input) : calcul des sorties en fonction des entrées données
- backward_propagation(output_error, learning_rate) : ajustement des poids / biais de la couche en fonction de l'erreur calculée et du taux d'apprentissage (méthode donnée complètement)

2 Implémentation des couches FCLayer et ActLayer en Python

Vous allez dans cette partie implémenter une couche de type Fully Connected Layer (sans activation) puis une couche d'activation. Les méthodes de rétro-propagation pour l'ajustement des biais et des poids sont données (je les ai commentées si leur fonctionnement vous intéresse et que vous êtes à l'aise avec le produit matriciel et la dérivation).

1. Récupérer les fichiers FCLayer_Squelette.py et ActLayer_Squelette.py et les enregistrer sur votre clef sans la partie _Squelette dans le nom.
2. Compléter la méthode __init__ de la couche FC.
3. Compléter la méthode forward_propagation de la couche FC. La tester avec les valeurs de l'exemple précédent (vous écrirez la matrice des poids et le tableau des biais à la main).
4. Faire de même pour la couche d'activation. L'activation consiste simplement à écrire en sortie tanh(x) pour chaque entrée x.



Les couches sont maintenant prêtes, le plus dur est fait. Il n'y a plus qu'à les utiliser dans un réseau.

3 Le réseau de neurones

Nous allons créer une classe NeuralNetwork. Cette classe n'aura qu'un seul attribut : *layers* qui est un tableau contenant les différentes couches de neurones.
Elle possède par contre 3 méthodes (en plus du constructeur) :

- addLayer(nbInput,nbOuput) : méthode qui permet d'ajouter une couche de neurones de type FCLayer au réseau. Pour chaque couche FCLayer insérée on ajoute une couche ActLayer.
- training(training_inputs,training_outputs,epochs,learning_rate) : méthode pour entraîner le réseau à partir d'un jeu de données comprenant les entrées et les sorties attendues. Cette méthode est donnée complètement.
- predict(inputs) : méthode permettant de calculer les valeurs de sorties du réseau à partir des valeurs d'entrées fournies par propagation vers l'avant à travers toutes les couches.

1. Récupérer le fichier NeuralNetwork_Squelette.py et l'enregistrer sur la clef.
2. Compléter les méthodes __init__, addLayer et predict.

4 Utilisation du réseau

Il n'y a plus qu'à utiliser notre réseau.

4.1 XOR

1. Dans un fichier séparé, importer le réseau de neurones
2. Créer un réseau de 2-3 couches, ayant 2 entrées et 1 seule sortie
3. On va maintenant entraîner le réseau avec un XOR. Pour cela on va créer une liste d'entrées possibles et une autre contenant les sorties associées.

```
entrees = [[0,0],[0,1],[1,0],[1,1]]
sorties = [[0],[...],[...],[...]]
```

4. Entraîner le réseau un millier de fois sur ces données.
5. Tester pour voir si le réseau à appris correctement!!!

4.2 Les Iris!!

Refaire le travail précédent pour créer un réseau capable de classer des iris.
Vous pourrez récupérer le fichier iris.csv pour créer vos jeux d'entrées / sorties.