Chiem Romain et Desmaret Mathéo #Rapport LO21 « Réseaux de neurones »

Table des matières

1. Choix de conception et d'implémentation

Liste chaînée pour les poids :

- Chaque neurone possède une liste chaînée de poids. Ce choix permet une gestion dynamique et extensible des poids sans besoin de donner leurs nombres à l'avance.
- La liste chaînée est idéale car les poids peuvent être ajoutés ou supprimés dynamiquement.

Structure pour le neurone :

- Un neurone contient une liste de poids et un biais, regroupant ainsi toutes les informations pour le calcul de sa sortie.

Liste dynamique pour les neurones dans une couche :

- Les couches sont représentées par une liste de neurones. Ce choix permet de gérer facilement un nombre variable de neurones dans chaque couche.

Structure pour le réseau :

- Le réseau est organisé en une liste chaînée de couches, ce qui permet une construction simple et pratique des couches du réseau.

Démarche adoptée

- Chaque composant (poids, neurone, couche, réseau) est conçu séparément pour respecter le principe de modularité.
- Les fonctions sont conçues pour être réutilisables et flexibles, par exemple en utilisant des paramètres tels que le nombre d’entrées ou les biais.
- Nous avons choisi une implémentation où le réseau est demandé à l'utilisateur pour le programme, ce n'est par exemple pas très adapté pour des réseaux plus complexes.
- Également, nous avons choisi de considérer que le nombre de poids d'un neurone correspond au nombre d'entrée qu'il reçoit, nous ne considérons donc pas des neurones avec des poids initialisé mais non utilisé.
- Ainsi il suffit de demander le nombre d'entrée des neurones de la première couche pour avoir le nombre de poids de toute la couche, puis le nombre de neurone de la couche indiquera le nombre de sorti et donc le nombre d'entrée de la couche suivante donc le nombre d'entrée des neurones. Le nombre de poids/entrée des neurones d'une couche dépend du nombre de neurone de la couche précedente.

2. Les Algorithmes

Fonction CreerListePoids (nombreEntrees: entier strict positif):liste
Début

```
listePoids <- créerListeVide()
courant <- CreerListeVide()

Pour i allant de 1 à nombreEntrees faire :
    poids<- (UserEntry)
    nouveauPoids<-CréerPoidsVide()
    nouveauPoids.poids<-poids
    Si listePoids == NULL faire
        listePoids <- nouveauPoids
    Sinon
        courant.suivant <- nouveauPoids
    courant <- nouveauPoids
    Fin Si
Fin Pour


CreerListePoids <- listePoids
```

Fin

Fonction InitNeur (nombreEntrees: entier strict positif): Neurone
Début

```
biais <- (UserEntry)
listePoids <- CreerListePoids(nombreEntrees)

neurone <- CréerNeuroneVide()
neurone.biais <- biais
neurone.listePoids <- listePoids
neurone.suivant <- NULL

InitNeur <- neurone
```

Fin

Fonction OutNeurone (neurone: Neurone, entrees: Liste<entrée:entier>): entier
Début

```
Si neurone == NULL ou entrees == NULL faire
    Afficher("Erreur : Neurone ou liste d'entrées invalide.")
    OutNeurone<- ERREUR
Fin Si

somme <- 0
courantPoids <- neurone.listePoids
courantEntree <- entrees

Tant que courantPoids != NULL et courantEntree != NULL faire
    somme <- somme + courantPoids.poids * courantEntree.valeur
    courantPoids <- courantPoids.suivant
    courantEntree <- courantEntree.suivant
Fin Tant Que

Si somme >= neurone.seuil faire
    OutNeurone<-1
Sinon
    OutNeurone<-0
Fin Si
```

Fin

Fonction InitCouche (nombreNeurones: entier strict positif, nombreEntrees: entier strict positif): Couche
Début

```
nouvelleCouche <- CréerCoucheVide()
nouvelleCouche.listeNeurones <- NULL
nouvelleCouche.suivant <- NULL
```

```
courant <- NULL

Pour i allant de 1 à nombreNeurones faire
    Afficher "Initialisation du neurone " + i + " de la couche :"

    nouveauNeurone <- InitNeur(nombreEntrees)

    Si nouvelleCouche.listeNeurones == NULL alors
        nouvelleCouche.listeNeurones <- nouveauNeurone
    Sinon
        courant.suivant <- nouveauNeurone
    Fin Si

    courant <- nouveauNeurone
Fin Pour

InitCouche <- nouvelleCouche
```

Fin

Fonction Outcouche (couche: Couche, entrees: Liste<entré:entier>): Liste<entrée:entier>
Début

```
Si couche == NULL ou couche.listeNeurones == NULL ou entrees == NULL alors
    Afficher "Erreur : Couche ou liste d'entrées invalide."
    Outcouche<- ERROR
Fin Si

listeSorties <- CréerListeVide()
courantSorties <- NULL

courantNeurone <- couche.listeNeurones

Tant que courantNeurone != NULL faire
    sortie <- Outneurone(courantNeurone, entrees)

    Si listeSorties == NULL alors
        listeSorties <- nouvelleSortie
    Sinon
        courantSorties.suivant <- nouvelleSortie
    Fin Si

    courantSorties <- nouvelleSortie
    courantNeurone <- courantNeurone.suivant
```

```
Fin Tant Que

Outcouche<-listeSorties
```

Fin

Fonction CreerResNeur (nbCouches: entier strict positif, neuronesParCouche: tableau d'entiers, entreesPremiereCouche: entier strict positif): Reseau
Début

```
reseau <- CréerRéseauVide(Reseau)
reseau.premiereCouche <- NULL
reseau.nbCouches <- nbCouches

coucheCourante <- NULL
nbEntrees <- entreesPremiereCouche

Pour i allant de 0 à nbCouches - 1 faire

    Afficher "Création de la couche " + (i + 1) + " avec " +
neuronesParCouche[i] + " neurones..."

    nouvelleCouche <- InitCouche(neuronesParCouche[i], nbEntrees)

    Si reseau.premiereCouche == NULL alors
        reseau.premiereCouche <- nouvelleCouche
    Sinon
        coucheCourante.suivant <- nouvelleCouche
    Fin Si

    coucheCourante <- nouvelleCouche

    nbEntrees <- neuronesParCouche[i]

Fin Pour

CréerResNeur <- reseau
```

Fin

Fonction OutReseau (reseau: Reseau, entreesInitiales: Liste<Entrée:entier>): Liste<Entrée:entier>
Début

```
entreesActuelles <- entreesInitiales
coucheCourante <- reseau.premiereCouche
```

```
Tant que coucheCourante != NULL faire
    entreesActuelles <- Outcouche(coucheCourante, entreesActuelles)
    coucheCourante <- coucheCourante.suivant
Fin Tant que

OutReseau <- entreesActuelles
```

Fin

Fonction afficherListeEntrees (listeEntrees: Liste<Entrée>)
Début

```
courant <- listeEntrees
Tant que courant != NULL faire
    Afficher courant.valeur
    courant <- courant.suivant
Fin Tant que

Afficher "\n"
```

Fin

Fonction AfficherNeurone (neurone: Neurone)
Début

```
courant <- neurone.listePoids
Afficher "Poids : "

Tant que courant != NULL faire
    Afficher courant.poids
    courant <- courant.suivant
Fin Tant que

Afficher "Biais : " + neurone.biais
```

Fin

Fonction afficherCouche (couche: Couche)
Début

```
Si couche == NULL ou couche.listeNeurones == NULL alors
    Afficher "La couche est vide."
    afficheCouche<-- ERROR
Fin Si
```

```
courant <- couche.listeNeurones
index <- 1
Tant que courant != NULL faire
    Afficher "Neurone " + index + " :"
    afficherNeurone(courant)
    courant <- courant.suivant
    index <- index + 1
Fin Tant Que
```

Fin

Fonction afficherReseau (reseau: Reseau)
Début

```
coucheCourante <- reseau.premiereCouche
indexCouche <- 1

Tant que coucheCourante != NULL faire
    Afficher "Couche ", indexCouche, ":"
    afficherCouche(coucheCourante)
    coucheCourante <- coucheCourante.suivant
    indexCouche <- indexCouche + 1
Fin Tant que
```

Fin

Fonction freeListePoids(listePoids: Liste)
Début

```
courant <- listePoids
Tant que courant != NULL faire
    suivant <- courant.suivant
    free(courant)
    courant <- suivant
Fin Tant que
```

Fin

Fonction freeListeNeurones(listeNeurones: Liste)
Début

```
courant <- listeNeurones
Tant que courant != NULL faire
    suivant <- courant.suivant
```

```
    freeListePoids(courant.listePoids)
    free(courant)
    courant <- suivant
Fin Tant que
```

Fin

Fonction freeListeCouches(listeCouches: Liste)

Début

```
courant <- listeCouches
Tant que courant != NULL faire
    suivant <- courant.suivant
    freeListeNeurones(courant.listeNeurones)
    Libérer courant
    courant <- suivant
Fin Tant que
```

Fin

Fonction freeReseau(reseau: Reseau)

Début

```
Si reseau = NULL alors
    Retourner
Fin Si
freeListeCouches(reseau.premiereCouche)
free(reseau)
```

Fin

Explications des algorithmes

1. Fonction `CreerListePoids(nombreEntrees: entier strict positif): liste<Poids>`

- **Entrée :**
  - `nombreEntrees` : Un entier strictement positif représentant le nombre d'entrées pour le neurone.
- **Sortie :**
  - Une liste chaînée de poids (`liste<Poids>`), où chaque élément contient un poids associé à une entrée du neurone.

- **Lexique :**
  - `listePoids` : La liste qui contiendra les poids des entrées.
  - `nouveauPoids` : Un nœud de la liste, contenant un poids.
  - `courant` : Un pointeur permettant de naviguer dans la liste des poids.

---

2. Fonction `InitNeur(nombreEntrees: entier strict positif): Neurone`

- **Entrée :**
  - `nombreEntrees` : Un entier strictement positif représentant le nombre d'entrées pour le neurone.
- **Sortie :**
  - Un neurone initialisé (`Neurone`), avec un biais et une liste de poids associée.
- **Lexique :**
  - `biais` : La valeur du biais du neurone, saisie par l'utilisateur.
  - `listePoids` : Liste des poids des entrées, créée par la fonction `CreerListePoids`.
  - `neurone` : Le neurone initialisé, contenant un biais et une liste de poids.

---

3. Fonction `OutNeurone(neurone: Neurone, entrees: Liste<entrée:entier>): entier`

- **Entrée :**
  - `neurone` : Le neurone dont on veut calculer la sortie.
  - `entrees` : La liste d'entrées à traiter.
- **Sortie :**
  - Un entier représentant la sortie du neurone, soit `0` ou `1`.
- **Lexique :**
  - `somme` : La somme des produits des poids et des entrées.
  - `courantPoids` : Un pointeur sur les poids du neurone.
  - `courantEntree` : Un pointeur sur les entrées.
  - `seuil` : La valeur seuil qui détermine la sortie (si la somme est supérieure ou égale au seuil, la sortie est `1`).

---

4. Fonction `InitCouche(nombreNeurones: entier strict positif, nombreEntrees: entier strict positif): Couche`

- **Entrée :**
  - `nombreNeurones` : Un entier strictement positif représentant le nombre de neurones dans la couche.
  - `nombreEntrees` : Un entier représentant le nombre d'entrées pour chaque neurone.
- **Sortie :**
  - Une couche de neurones (`Couche`), qui contient les neurones initialisés.
- **Lexique :**
  - `nouvelleCouche` : La couche créée.
  - `courant` : Un pointeur qui permet de relier les neurones entre eux dans la couche.
  - `nouveauNeurone` : Un neurone créé pour la couche, initialisé avec les entrées spécifiées.

---

5. Fonction `Outcouche(couche: Couche, entrees: Liste<entrée:entier>): Liste<entrée:entier>`

- **Entrée :**
  - `couche` : La couche de neurones dont on veut calculer la sortie.
  - `entrees` : La liste des entrées pour cette couche.
- **Sortie :**
  - Une liste d'entiers représentant les sorties des neurones dans la couche.
- **Lexique :**
  - `listeSorties` : Liste des sorties des neurones dans la couche.
  - `courantSorties` : Pointeur pour naviguer et relier les sorties.
  - `courantNeurone` : Pointeur pour naviguer dans la liste des neurones de la couche.

---

6. Fonction `CreerResNeur(nbCouches: entier strict positif, neuronesParCouche: tableau d'entiers, entreesPremiereCouche: entier strict positif): Reseau`

- **Entrée :**
  - `nbCouches` : Le nombre de couches dans le réseau.
  - `neuronesParCouche` : Un tableau d'entiers indiquant le nombre de neurones pour chaque couche.
  - `entreesPremiereCouche` : Le nombre d'entrées pour la première couche du réseau.
- **Sortie :**
  - Un réseau de neurones (`Reseau`), composé de plusieurs couches de neurones.

- **Lexique :**
  - `reseau` : Le réseau de neurones créé.
  - `nouvelleCouche` : Une couche de neurones créée pour le réseau.
  - `coucheCourante` : Un pointeur pour naviguer à travers les couches du réseau.
  - `nbEntrees` : Le nombre d'entrées pour chaque couche, mis à jour à chaque itération.

---

7. Fonction `OutReseau(reseau: Reseau, entreesInitiales: Liste<Entrée:entier>): Liste<Entrée:entier>`

- **Entrée :**
  - `reseau` : Le réseau de neurones à traiter.
  - `entreesInitiales` : La liste des entrées initiales pour le réseau.
- **Sortie :**
  - La liste des sorties finales du réseau.
- **Lexique :**
  - `entreesActuelles` : La liste d'entrées à chaque étape (met à jour les sorties des couches successives).
  - `coucheCourante` : Un pointeur permettant de naviguer à travers les couches du réseau.

---

8. Fonction `afficherListeEntrees(listeEntrees: Liste<Entrée>)`

- **Entrée :**
  - `listeEntrees` : Une liste d'entrées à afficher.
- **Sortie :**
  - Aucune. Cette fonction affiche les valeurs des entrées à l'écran.
- **Lexique :**
  - `courant` : Pointeur permettant de naviguer à travers la liste d'entrées.

---

9. Fonction `AfficherNeurone(neurone: Neurone)`

- **Entrée :**
  - `neurone` : Le neurone dont on veut afficher les poids et le biais.
- **Sortie :**
  - Aucune. Cette fonction affiche les poids et le biais du neurone.
- **Lexique :**

- `courant` : Pointeur permettant de naviguer dans la liste des poids du neurone.

---

10. Fonction `afficherCouche(couche: Couche)`

- **Entrée :**
  - `couche` : La couche de neurones à afficher.
- **Sortie :**
  - Aucune. Cette fonction affiche tous les neurones de la couche.
- **Lexique :**
  - `courant` : Pointeur permettant de naviguer à travers les neurones de la couche.
  - `index` : Un compteur pour indiquer l'index du neurone dans la couche.

---

11. Fonction `afficherReseau(reseau: Reseau)`

- **Entrée :**
  - `reseau` : Le réseau de neurones à afficher.
- **Sortie :**
  - Aucune. Cette fonction affiche toutes les couches et neurones du réseau.
- **Lexique :**
  - `coucheCourante` : Pointeur permettant de naviguer à travers les couches du réseau.
  - `indexCouche` : Un compteur pour indiquer l'index de la couche dans le réseau.

12.Fonction freeListePoids(listePoids: Liste)

- **Entrée :**
  - `listePoids` : Une liste chaînée de poids à libérer de la mémoire.
- **Sortie :**
  - Aucune. La fonction libère chaque nœud de la liste.
- **Lexique :**
  - `courant` : Un pointeur qui parcourt les nœuds de la liste des poids.
  - `suivant` : Un pointeur temporaire pour stocker le nœud suivant pendant la libération.

---

13. Fonction freeListeNeurones(listeNeurones: Liste)

- **Entrée :**
  - `listeNeurones` : Une liste chaînée de neurones à libérer de la mémoire.

- **Sortie :**
  - Aucune. La fonction libère chaque neurone, y compris ses poids.
- **Lexique :**
  - `courant` : Un pointeur qui parcourt les nœuds de la liste des neurones.
  - `suivant` : Un pointeur temporaire pour stocker le neurone suivant pendant la libération.
  - `courant.listePoids` : La liste de poids associée à un neurone, qui est libérée avant le neurone lui-même.

---

14. Fonction freeListeCouches(listeCouches: Liste)

- **Entrée :**
  - `listeCouches` : Une liste chaînée de couches à libérer de la mémoire.
- **Sortie :**
  - Aucune. La fonction libère chaque couche, y compris les neurones qu'elle contient.
- **Lexique :**
  - `courant` : Un pointeur qui parcourt les nœuds de la liste des couches.
  - `suivant` : Un pointeur temporaire pour stocker la couche suivante pendant la libération.
  - `courant.listeNeurones` : La liste de neurones associée à une couche, qui est libérée avant la couche elle-même.

---

15. Fonction freeReseau(reseau: Reseau)

- **Entrée :**
  - `reseau` : Un réseau de neurones complet à libérer de la mémoire.
- **Sortie :**
  - Aucune. La fonction libère toutes les couches, neurones et poids du réseau.
- **Lexique :**
  - `reseau.premiereCouche` : La première couche du réseau, qui est libérée avec toutes les couches suivantes.
  - `reseau` : L'objet réseau lui-même, libéré après ses composants.

Résumé du Lexique :

- **Poids** : Valeur qui détermine l'importance d'une entrée pour un neurone.
- **Biais** : Valeur qui ajuste la sortie d'un neurone avant l'application de la fonction d'activation, dans le projet on considère que le biais est un seuil auquel on compare la somme pondérée.
- **Couche** : Un groupe de neurones dans le réseau.

- **Reseau** : Un ensemble de couches de neurones connectées entre elles.

Ces fonctions forment un réseau de neurones où chaque neurone reçoit des entrées, les multiplie par des poids et ajoute un biais, puis applique une fonction de seuil pour produire une sortie. Le réseau peut être constitué de plusieurs couches de neurones et est utilisé pour effectuer des prédictions ou classifications basées sur des entrées données.

3. Jeux d'essais

Une seule couche avec deux neurones

- **Entrées** : `4, 0, 1`
- **Réseau** :
  - Couche 1 :
    - Neurone 1 : Poids `2, 6, 2` Biais `12`
    - Neurone 2 : Poids `1, 3, 4` Biais `6`
- **Résultat attendu** :
  - Neurone 1 : `(4*2) + (0*6) + (1*2) = 10 < 12 ⇒ Sortie : 0`
  - Neurone 2 : `(4*1) + (0*3) + (1*4) = 8 ≥ 6 ⇒ Sortie : 1`
  - Sortie de la couche : `0, 1`

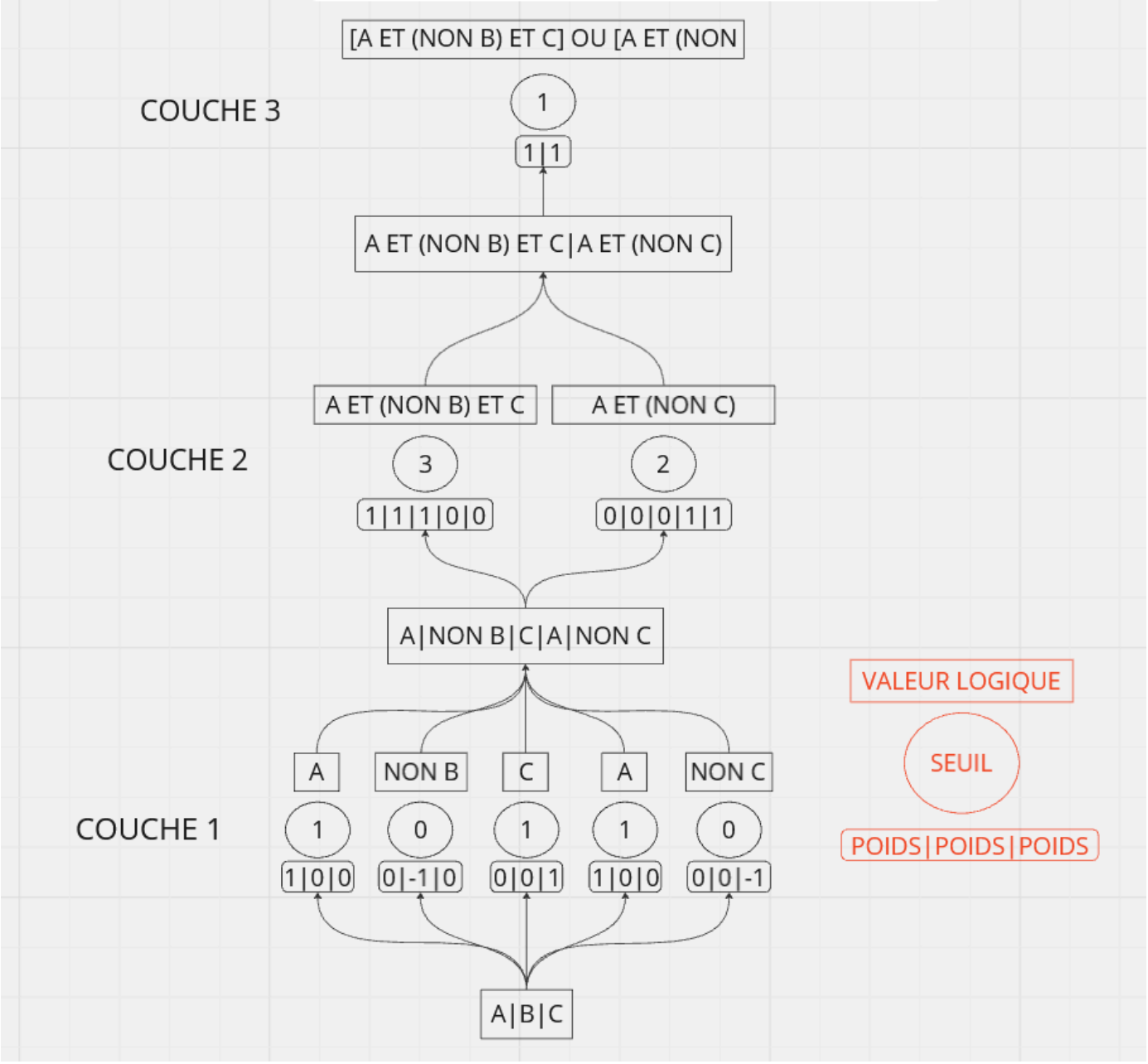
Deux couches connectées

- **Entrées** : `2, 3`
- **Réseau** :
  - Couche 1 :
    - Neurone 1 : Poids `2, 6` Biais `3`
    - Neurone 2 : Poids `2, 4` Biais `20`
  - Couche 2 :
    - Neurone 1 : Poids `3, 6` Biais `2`
    - Neurone 2 : Poids `2, 1` Biais `6`
    - Neurone 2 : Poids `1, 4` Biais `1`
- **Résultat attendu** : La sortie de la première couche alimente les entrées de la deuxième couche, permettant une propagation avant complète.
  - Couche 1 :
    - Neurone 1 : `(2*2) + (3*6) = 22 ≥ 3 ⇒ Sortie : 1`
    - Neurone 2 : `(2*2) + (3*4) = 16 < 20 ⇒ Sortie : 0`
    - Sortie de la couche : `1, 0`

- Couche 2 :
  - Neurone 1 : `(1*3) + (0*6) = 3 ≥ 2 ⇒ Sortie : 1`
  - Neurone 2 : `(1*2) + (0*1) = 2 < 6 ⇒ Sortie : 0`
  - Neurone 2 : `(1*1) + (0*4) = 1 ≥ 1 ⇒ Sortie : 1`
  - Sortie de la couche : `1, 0, 1`

Schéma pour le réseau multi-couche

Voici un schéma explicatif pour le réseau multi-couche



4. Commentaires sur les résultats

- La propagation avant du réseau de neurone s'effectue correctement dans les sous-couches cachées.
- La fonction d'activation choisi est une somme pondérée que l'on compare au seuil/biais du neurone.
- La fonction d'activation du neurone est facilement modifiable, le code est modulaire.

- Comme dit précedemment l'implémentation choisi est adapté pour des réseaux simples car le réseau est à initialiser à la main par l'utilisateur.
- Le système de réseau peut facilement être utilisé pour réaliser des opérations logiques complexes.