

Réalisation d'un réseau neuronal

Rapport de projet

2019 - 2020

Préface

Notre équipe est constituée des personnes suivantes :

- Louis Chaillou
- Hugo Sinprasith
- Lucca Anthoine (chef de projet)
- Sylvain Thor

Nous avons été accompagnés par notre tuteur Oleg Loukianov que nous tenions absolument à remercier.

Table des matières

Réalisation d'un réseau neuronal.....	1
2019 - 2020	1
.....	1
Préface	2
1- Introduction	4
2- Réalisation du projet	4
2-1- Stratégie adoptée.....	4
2-1-1- Division des tâches	4
2-1-2- Travail de recherche.....	5
2-1-3- Le Python	5
2-1-4- Pair-programming	6
2-1-5- L'interface graphique	7
2-1-6- Déroulement étape par étape	16
2-2- Le réseau neuronal	17
2-2-1- Qu'est-ce qu'un réseau neuronal ?.....	17
2-2-2- Apprentissage	21
2-2-3- Sortie.....	23
3- Conclusion	25
Annexes	27
TITRE.....	19

1- Introduction

Dans le cadre de notre DUT, nous avons choisi de réaliser un réseau neuronal faisant l'objet de notre projet tutoré. Aimant tous les 4 la programmation et tout ce qui touche à l'intelligence artificielle, ce sujet s'est révélé être plus qu'intéressant pour nous.

Ayant précédemment programmés un jeu de Memory en C et un algorithme d'Ariane en Java au cours de notre cursus, nous nous sommes orientés vers le Python afin d'apprendre un nouveau langage.

Pour expliciter brièvement, un réseau de neurones possède une structure en couches qui est similaire à la structure en réseau des neurones du cerveau, avec des couches de nœuds connectés. Un réseau de neurones peut apprendre à partir de données, il peut ainsi être entraîné à reconnaître des tendances, classer des données et prévoir des événements à venir.

Notre principal objectif était de pouvoir constater notre réseau neuronal être capable de s'entraîner sur des données, et plus particulièrement des images, et d'interpréter certains schémas/patterns (nous y reviendrons plus tard).

2- Réalisation du projet

2-1- Stratégie adoptée

2-1-1- Division des tâches

Face à un projet d'une telle ampleur, nous devons nous répartir les tâches de sorte qu'on puisse tous avancer en même temps. Pour commencer, nous avons eu l'idée de diviser le groupe en 2 :

- Lucca et Sylvain sur la partie mathématique du réseau neuronal avec le calcul des poids, matrices, biais, rétropropagation de l'erreur, etc.
- Louis et Hugo sur la partie interface graphique ainsi que sur la partie portant sur l'exploitation des données par python et la structure du code.

Nous avons représenté sous forme d'un diagramme de Gantt (cf Annexe 1) comment s'est déroulé le projet et les différentes étapes qu'il impliquait. Le premier aspect qui nous a marqué est le fait que la phase de documentation et de compréhension a été plus longue que

l'on ne l'imaginait. Effectivement, on ne pensait pas que cette dernière allait durer plus d'un mois.

De plus, nous n'avions pas anticipé le fait qu'il puisse y avoir une phase d'optimisation consacrée aux ajustements et à l'augmentation de la précision du réseau neuronal.

Cependant, la période de développement a bien été aussi longue que ce que nous avions prévu.

2-1-2- Travail de recherche

N'ayant jamais approchés les réseaux neuronaux (ou deep-learning), puis n'ayant jamais programmés à l'aide de Python, nous avons dû nous renseigner sur ces aspects ainsi que sur les notions qu'ils impliquent.



Effectivement, la phase de documentation et celle de l'acquisition de connaissances a été fondamentale pour le bon déroulement du projet. Ces phases nous ont occupé les premières semaines, le temps de pouvoir comprendre et apprendre sur les réseaux neuronaux, il en a été de même pour le langage Python.

Bien que nous nous sommes principalement concentrés sur la documentation au début du projet, nous avons continué de découvrir et de prendre connaissance de nouveaux concepts tout au long de notre projet, et ce, à travers des articles, des études ou encore des vidéos explicatives.

2-1-3- Le Python

Tout d'abord, il a été question d'apprendre les bases du Python (3.7). En effet, après avoir travaillé sur du Java, nous n'avons rencontré que très peu de problèmes face au Python.

Nous avons donc appris à déclarer/manipuler/traiter des tableaux/chaînes/listes à travers différents scénarios.



Nous nous sommes particulièrement aidés de ce site, qui plus est la bibliothèque standard de Python 3.7 :

<https://docs.python.org/fr/3.7/library/index.html>

Avec le recul, nous ne regrettons absolument pas d'avoir choisi ce langage car en plus d'élargir notre panel de langage maîtrisé, il est à la fois complet et intuitif.

Pour la partie algorithmique, il nous a été d'une grande efficacité. Cependant pour la partie graphique, nous avons dû nous familiariser avec la syntaxe et les nouvelles méthodes rencontrées.

2-1-4- Pair-programming

Après les premières semaines de documentation et de renseignement sur le langage Python et les réseaux neuronaux, il n'a presque été incontournable que d'évoluer à 2 sur le code. En effet, chacun ayant appris et s'étant documenté individuellement, nous avons pu tous développer une base de connaissances propres à chacun. Ce qui permet de penser différemment et de pouvoir proposer des solutions lorsque quelqu'un n'arrive plus à avancer.

De ce fait, nous avons développé une complémentarité au sein de l'équipe, qui aurait pu nous faire défaut si nous avions décidé de travailler chacun de notre côté.



Ainsi, le pair-programming a été primordial pour le développement de la partie mathématique du réseau neuronal. Effectivement, cela nous a permis un gain de temps phénoménal par rapport à toutes les erreurs d'inattention ou autres problèmes rencontrés lors de la programmation.

De la même manière mais moins fréquemment, du côté interface graphique, le pair-programming a permis la confrontation de différents points de vue quant à l'organisation des composants graphiques de la fenêtre afin de la rendre la plus ergonomique et la plus efficace possible.

2-1-5- L'interface graphique

La première étape pour la conception de l'interface graphique a été le choix du module graphique. En effet, le langage Python ne possède pas de bibliothèque graphique native. Nous avons donc décidé de choisir le module "Tkinter" pour notre application car celui-ci est un module répandu et efficace pour toute sorte d'interface utilisateur graphique.

Pour cela, nous avons appris et assimilé les différents outils du module ainsi que leur mode de fonctionnement tout en gardant la structure et l'architecture MVC de l'application globale.

Ensuite, nous avons réparti le travail graphique en deux étapes différentes qui sont le menu de navigation utilisateur puis l'affichage graphique des réseaux neuronaux.

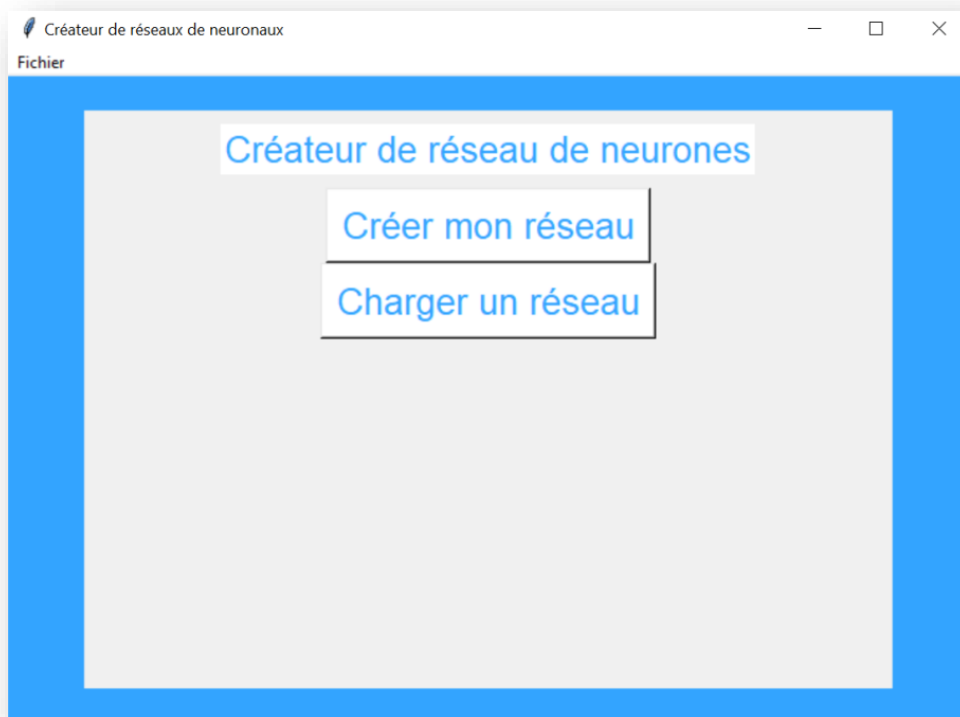
L'application est donc composée de deux vues c'est-à-dire une pour le menu et une seconde pour l'affichage des réseaux respectivement Nav.py et Dessin.py.

a) Menu de navigation

La première vue est le menu de navigation. Il se base sur 3 "Containers" qui sont des boîtes contenant des composants graphiques (Boutons, Labels, Champs de texte, ...) à travers lesquels on peut naviguer en toute simplicité.

Ces containers sont la page de menu lors du démarrage de l'application, la page de création de réseaux neuronaux et enfin la page de chargement de réseaux.

En se basant sur le synopsis, l'utilisateur arrive sur la page de menu lorsqu'il démarre l'application. Celle-ci est composée de deux boutons distincts "Créer un réseau" et "Charger un réseau" et mène tous les deux vers les pages de leur intitulé.



La page de création est un formulaire à remplir pour les différents paramètres du nouveau réseau. Il y a tout d'abord des champs de texte pour son nom ainsi que pour le nombre de neurones d'entrée et de sortie.

Ensuite, on y trouve un champ avec une syntaxe rigoureuse. Il s'agit d'un champ où il faut renseigner le nombre de couches de neurones ainsi que le nombre de neurones sur chacune de ses couches.

Pour cela, il faut écrire le nombre de neurone par couches et la longueur de cette chaîne est le nombre de couche. Chaque nombre de neurone doit être séparé par un espace.

Par exemple, si on souhaite un réseau de 3 couches avec 2 neurones sur la première couche, 3 neurones sur la deuxième et 1 neurone sur la troisième, il faut renseigner le champ tel que : 2 3 1.

Créateur de réseaux de neuronaux

Fichier

Nouveau réseau

Nom du réseau

Nombre de neurones d'entrée

0

Nombre de neurone pour chaque couches
Exemple : 3 couches, 1 neurone pour la première, 3 pour la deuxième et 2 pour la dernière == 1 3 2

Nombre de neurone en sortie

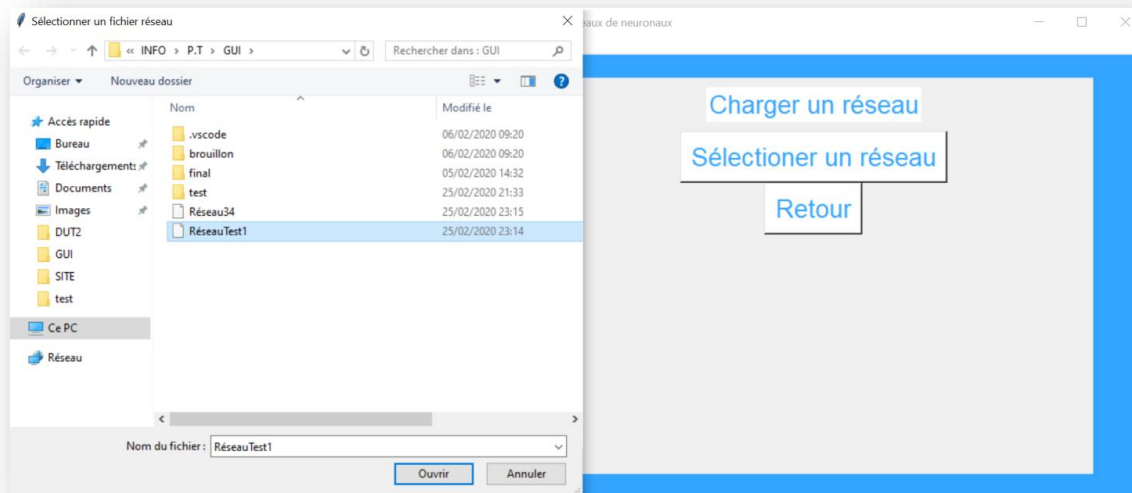
0

Retour

Valider

Une fois le réseau validé, un premier pop-up s'affiche attendant la confirmation de l'utilisateur pour créer le réseau et ses paramètres. Si l'utilisateur confirme, un fichier exploitable contenant les paramètres du réseau est alors créé dans le répertoire courant. Un second pop-up confirmant la création du réseau s'affiche donc et indique qu'il est désormais chargeable.

L'utilisateur est donc ramené à la page de menu où il peut maintenant aller dans la page de chargement de réseau et sélectionner un réseau qu'il a créé.



Lorsqu'il choisit un fichier de réseau, il est directement amené sur l'affichage graphique du réseau où il pourra choisir différentes options et ensuite lancer un entraînement.

b) Affichage graphique des réseaux neuronaux

Dans cette partie, le véritable enjeu est de comprendre la mise en place et la représentation du système neuronal dans notre interface graphique.

Le fichier concernant cette partie est Dessin.py

1- Représentation générale

Le système neuronal ainsi que sa barre latérale droite de fonctionnalités sont représentés sur une seule et unique fenêtre. A la différence du menu, aucun frame n'a été utilisé, tout a été fait à partir d'un Canvas du célèbre l'outil "tkinter". Les frames étant donc inexistantes, les containers n'étaient alors plus une possibilité pour nous. La seule option était de se servir des dimensions de l'écran hôte. La réelle explication de cette non-utilisation est l'évolution de nos idées au fur et à mesure du projet. Nous avons commencé par vouloir simplement représenter le réseau neuronal, ce qui n'impliquait pas une grande utilisation de frame ou containers, puis ensuite ajouter une colonne d'outils et enfin un "terminal" où l'utilisateur pouvait suivre son avancement, ses erreurs ainsi qu'obtenir des informations sur le système.

En commençant au début par dessiner le réseau, aucun frame n'a été utilisé et ayant accompli une part assez conséquente de la représentation, nous ne voulions pas abandonner nos avancées. Ainsi nous avons trouvé intéressant le fait de faire une interface graphique sans certains outils qui nous simplifierais la vie mais en contrepartie, l'investissement devait être bien plus important.

2- Représentation du réseau neuronal

Le réseau neuronal occupe la majeure partie de la fenêtre et bien sur la taille du réseau pouvant être dessiné n'est pas infinie, nous pouvons réaliser un réseau contenant seulement 10 couches cachées ainsi que 10 neurones par couches maximum. Il s'agit donc d'une représentation maximale de 12 par 10 (12 car 10 couches cachées au maximum ainsi que l'entrée et la sortie). Ces choix ont été fait lors de nos différentes réunions et notamment celle ou on s'interrogeait sur l'esthétisme et la lisibilité de notre interface.

Pour essayer de rendre notre code plus accessible à des modifications, nous avons organisé notre dessin en plusieurs étapes. Pour cela nous avons créé une méthode d'organisation (orgaDessin). Cette dernière comporte trois différentes parties : le dessin des cercles, des poids ainsi que ce que l'on appellera des pointillés qui sont les informations lorsque le réseau dépasse la taille maximum dessinaable.

Le dessin des cercles est basé sur le réseau neuronal fournit par le constructeur. Il dessine colonne par colonne le réseau. Pour dessiner justement le réseau en fonction de l'écran, nous divisons la fenêtre par le nombre de couches de neurones (pour la largeur) ainsi que par le nombre de neurones par couches (pour la hauteur). Le seul problème est de dessiné le bon nombre de neurones par couches sachant que chaque couche est différente et soumise à une taille maximum. Une simple vérification au début de la méthode suffit mais comme pour tout le dessin, il faut savoir se munir uniquement des informations qui vont nous servir dans la représentation du réseau. Nous utilisons alors le tableau d'informations créé lors de la naissance du réseau pour réussir une exploitation correcte.

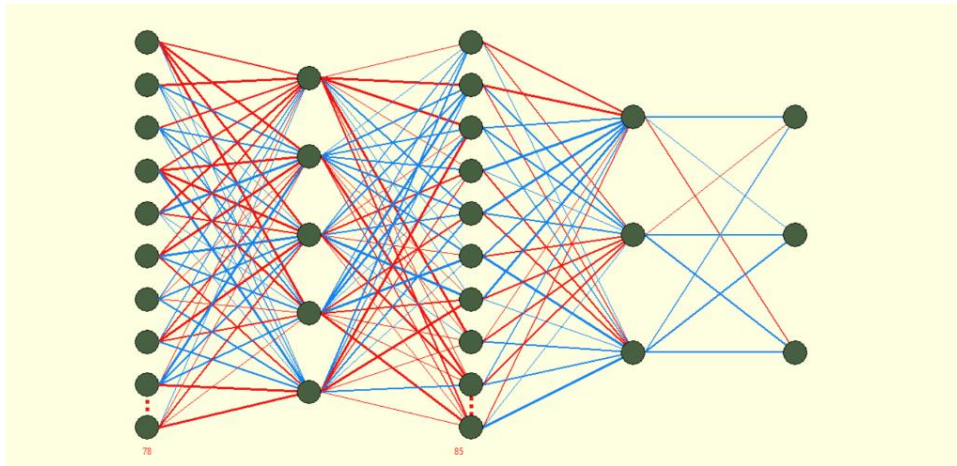
La partie la plus difficile à gérer a été les lignes reliant les neurones. Autrement appelé poids, plusieurs versions de leur représentation ont vu le jour. Nous avons commencé en représentant seulement le poids par une ligne créée à partir de la méthode `create_line` de `tkinter`. Pour dessiner uniquement les lignes nous avons besoin de la couche actuelle sur laquelle nous nous trouvons ainsi que celle d'après. A cela venait ensuite s'ajouter le neurone précis sur lequel nous nous trouvons et les neurones vers lesquels nous voulions aller. Toutes ces informations ont été calculées par des boucles et des dimensions.

Mais lorsque nous avons finis cette partie, nous avons voulu discerner à l'œil nu les poids négatifs et positifs. Nous avons donc dû lire et exploiter la matrice créée par le réseau neuronal grâce à la méthode `getMatrice()` qui nous renvoyait une matrice de neurones dont nous pouvions exploiter une matrice de poids par neurones. Pour symboliser la positivité nous avons choisi le bleu et pour la négativité le rouge.

Ayant envie de rendre le réseau plus intuitif, une dernière idée a été suggérée, celle de faire d'augmenter la largeur des lignes en fonction du poids, ainsi plus le poids est gros plus la ligne

est grosse. Pour cela aucune difficulté, nous avons décidé d'ajouter une largeur de base (1) au poids entre les deux neurones fournis par notre matrice de poids.

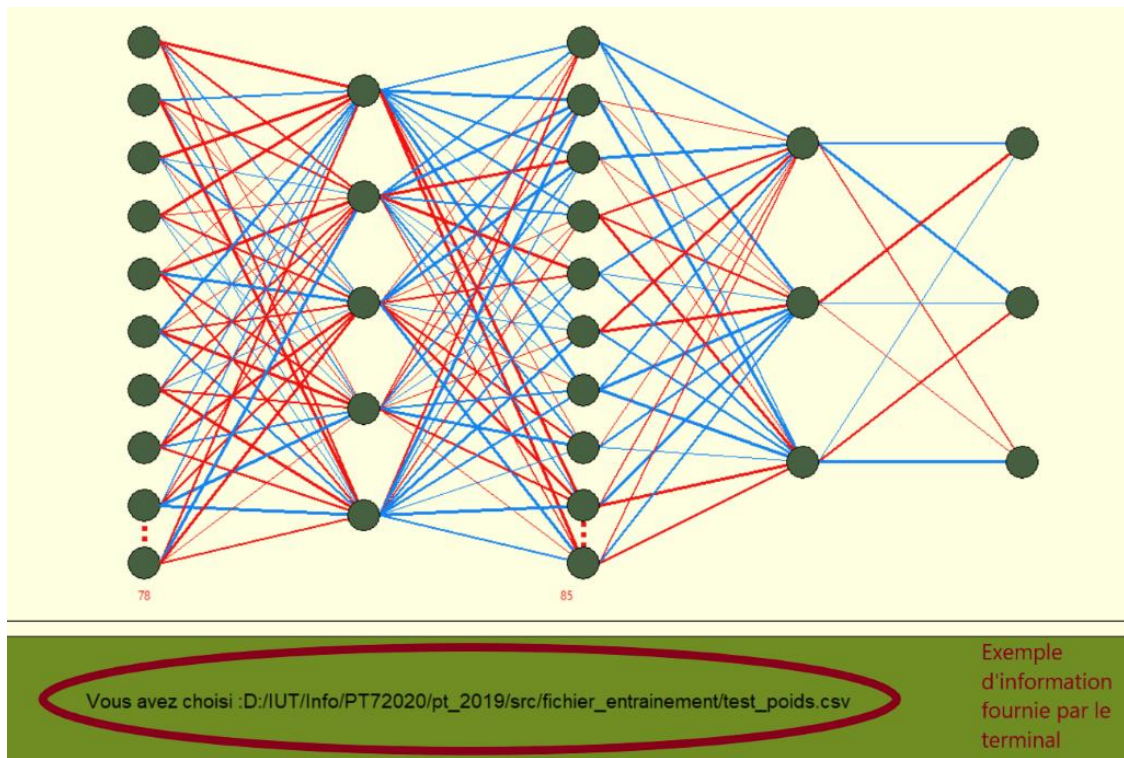
La troisième partie de notre organisation était de symboliser le nombre de neurone total non représentable en mode graphique. Nous avons donc opté pour un système de pointillé suivi par le nombre de total de neurone pour informer l'utilisateur. Cette option est remarquable pour chaque couche de neurones strictement supérieur à 10. Pour informer l'utilisateur du grand nombre de couches cachées, nous avons opté pour un message à l'écran plus tard déplacé dans le terminal ("Nombre de couches cachées au total : " + ...+ " (car > 10)"). Malheureusement, lorsque nous devons employer une réduction du réseau lors d'un nombre trop élevé de couches, nous perdons la représentation des poids entre la dernière couche cachée et la couche de sortie car il n'existe pas de relation concrète entre ces deux-là.



Pour terminer sur cette partie, il faut savoir que le nombre maximal de neurone représentable peut varier très simplement en modifiant quelques lignes de codes. Nous avons choisi 10 pour respecter les cours d'interface homme machine enseignés au début du 3ème semestre. La représentation la plus exacte était l'objectif mais passait bien après la bonne lisibilité du dessin par le lecteur.

3– Terminal et barre latérale

Le terminal est un espace situé sous le réseau neuronal, il permet d'afficher les différentes étapes de l'avancement du projet. Ce dernier va notamment servir à afficher les différentes sélections afin que l'utilisateur puisse être sûr qu'il s'agit bien du bon fichier. Son atout principal est lors de l'entraînement coup par coup. Nous allons afficher le pourcentage de lignes du fichier d'entraînement lu, pour une représentation plus intuitive pour l'utilisateur. Le terminal n'est en aucun cas un domaine d'interaction avec l'utilisateur. Son placement en bas de page est inspiré des IDE comme IntelliJ ou VisualStudio Code.



La partie la plus importante autre que le réseau neuronal dessiné est la barre de sélection placée sur le côté droit.

Elle contient :

- Un bouton ouvrant un gestionnaire de fichier permettant de sélectionner un fichier d'entraînement ;
- 2 boutons radios représentant le mode d'entraînement que nous voulons effectuer ;
- Un bouton valider pour lancer l'entraînement ;
- 2 boutons gestionnaires de fichiers pour l'entrée et la sortie du test sur le réseau neuronal effectué ;
- Un bouton pour lancer le test.

Le bouton de choix d'entraînement est le fichier que le réseau va lire pour s'entraîner, la majeure partie de la visualisation se fera sur cette épreuve.

Les deux boutons radios représentent comment nous voulons visualiser l'avancement de l'entraînement. Dans le cas où nous choisissons "Coup par coup", le réseau neuronal va lire 100 lignes par 100 lignes le fichier choisi précédemment et mettre à jour les poids. C'est donc chaque 100 lignes que nous allons rafraichir la représentation du réseau pour changer la largeur des traits afin qu'elle s'adapte à la valeur des poids. Dans le cas du choix de l'option "En un coup", l'utilisateur va assister à seulement un changement qui se passera à la fin de l'entraînement et qui rafraichira le dessin afin de le mettre à jour avec les poids finaux des neurones.

Les trois autres boutons seront relatifs aux tests et seront respectivement, le choix du fichier sur lequel les tests doivent être effectué, le choix du fichier qui sera rempli au fur par le résultat de chaque test et enfin le bouton permettant de valider le lancement d'un test.

Options

Choisir un entraînement

Lancer entraînement :

☐ Coup par coup

☐ En un coup

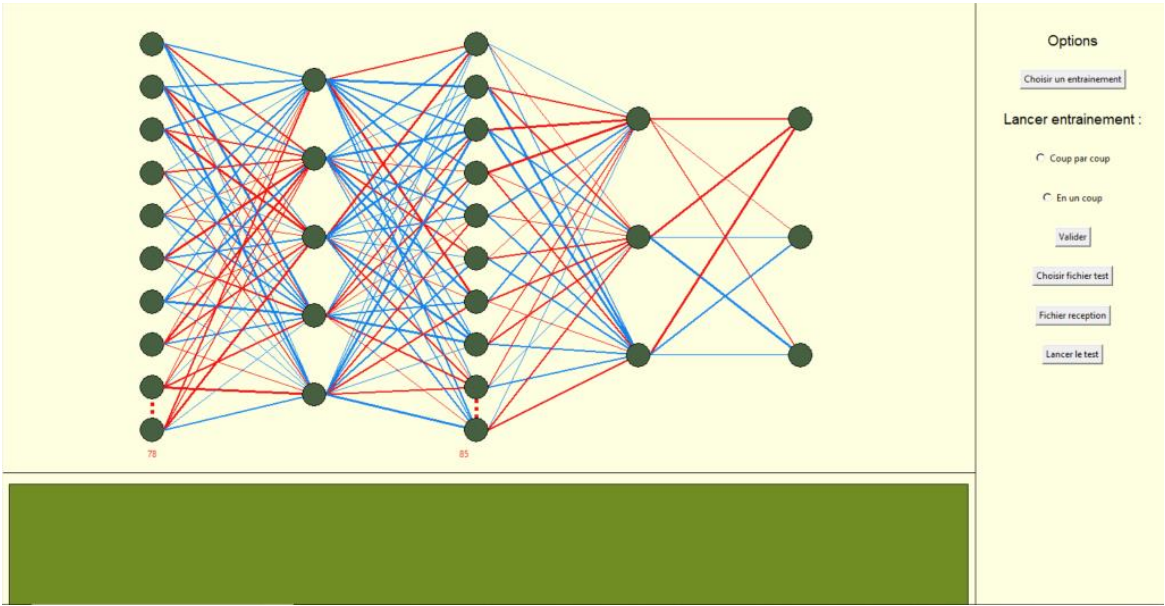
Valider

Choisir fichier test

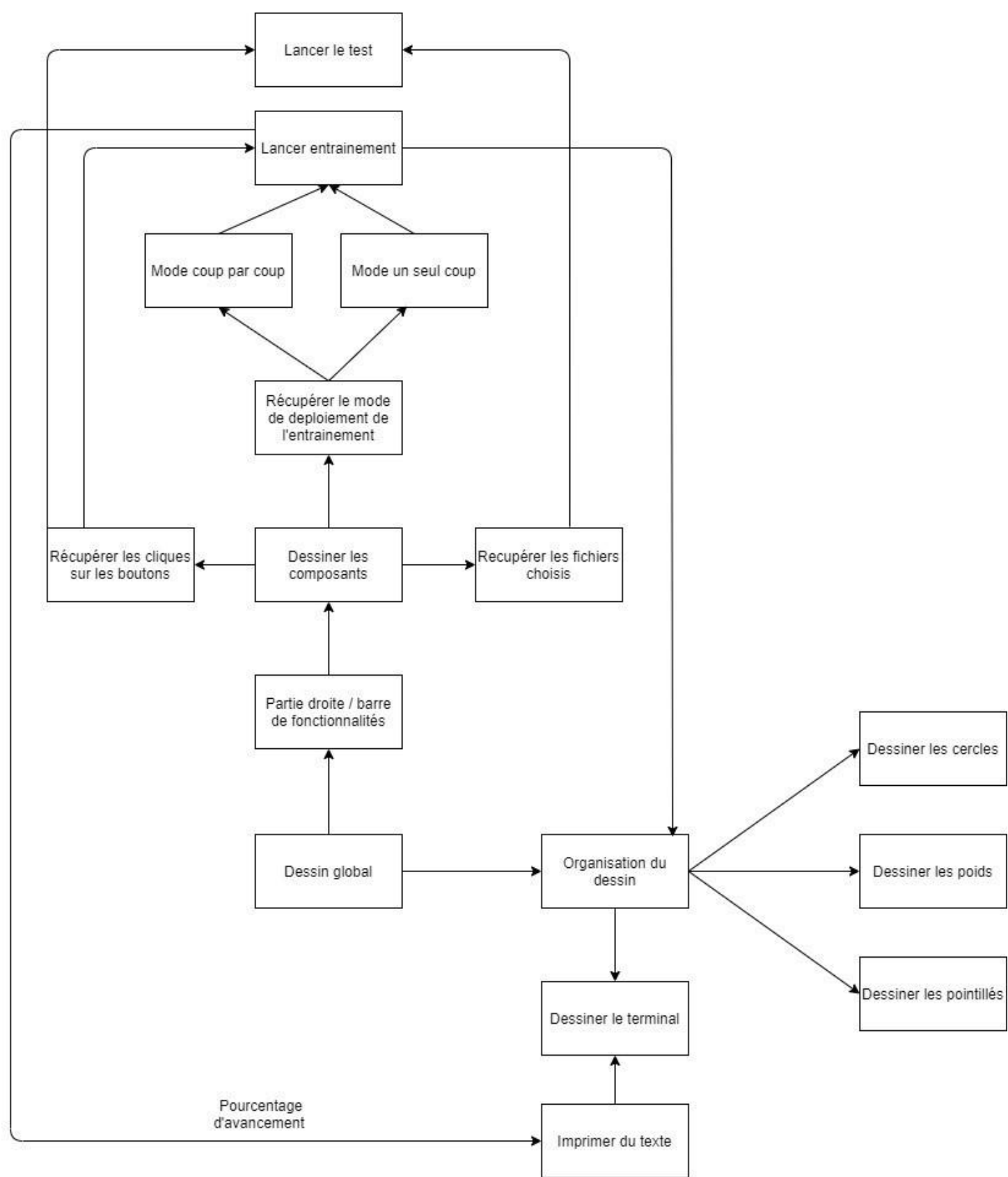
Fichier reception

Lancer le test

C'est lors du déclenchement de test ou d'entraînement que l'importance du terminal se fait ressentir, pour ne pas laisser l'utilisateur dans le flou. Ce concept respecte les cours d'ACDA sur l'interface graphique plus pour rendre l'interface "user friendly".



Pour résumer le fonctionnement de la partie graphique relative au fichier Dessin.py et plus particulièrement à son organisation, voici un schéma récapitulatif :



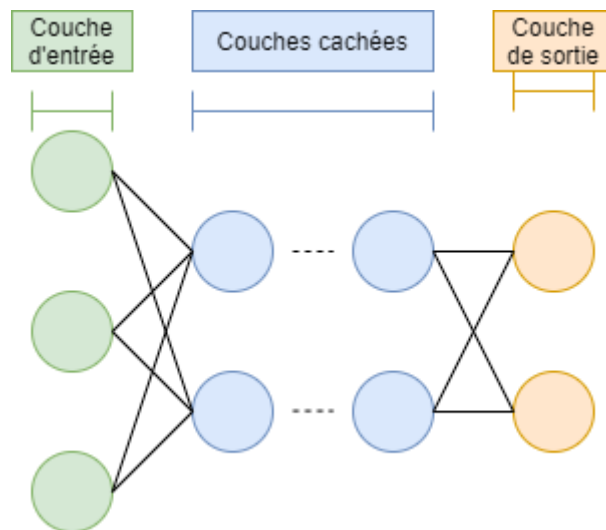
2-1-6- Déroulement étape par étape

Nous avons naturellement pris l'habitude d'élaborer chaque fonctionnalité étape par étape. Ce processus se présentait en général en **3 étapes**. On introduit une toute nouvelle idée qu'on va essayer de comprendre dans un premier temps afin d'élaborer un plan de travail, puis dans un second temps de réaliser en l'implémentant dans notre code.

a) Schémas, idées, concepts

Lorsqu'une nouvelle fonctionnalité veut voir le jour, elle passe par la case conceptualisation. Pour commencer on se met tous d'accord sur la nouvelle caractéristique à ajouter, qu'elle soit aussi bien fonctionnelle qu'esthétique.

Ensuite, on dessine et imagine les différents prototypes de la nouvelle fonctionnalité comme on peut le voir ci-dessous :



Ce schéma servait à représenter ce à quoi devait ressembler l'interface graphique grossièrement.

Comme la plupart de nos esquisses ont été réalisées sur papier, nous les avons redessinés au propre avec le site draw.io.

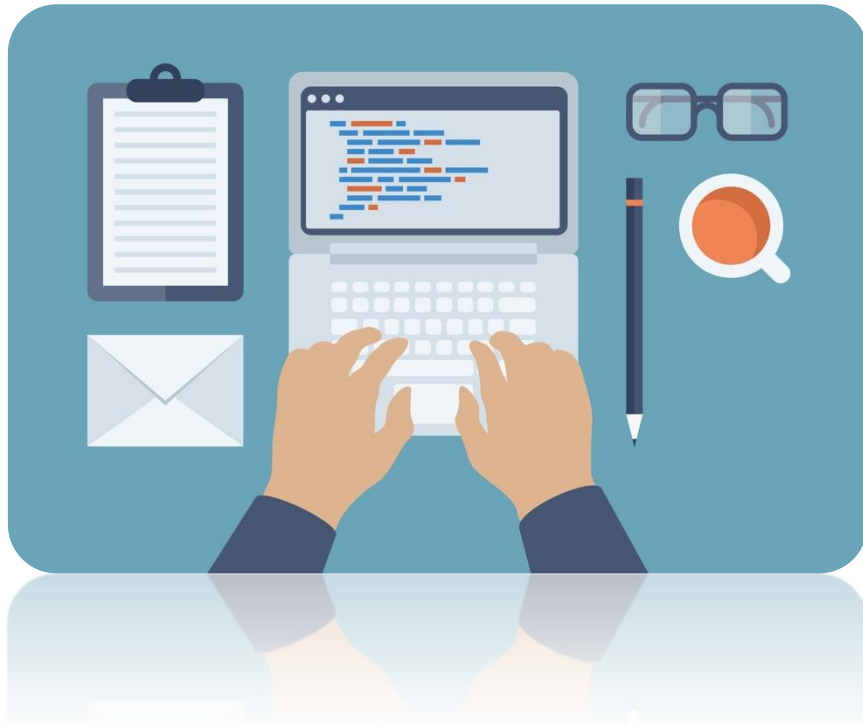
c) Compréhension et documentation

Après avoir pu approcher visuellement ce à quoi on s'attaquait, il fallait comprendre la nouvelle notion abordée et se préparer à la développer. Pour ce faire, on passait par de la documentation ou encore des mathématiques (comme nous allons le voir dans la partie 2-2- Le réseau neuronal).



b) Implémentation

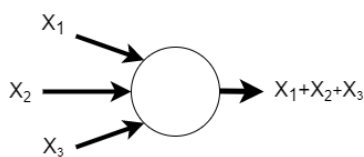
L'ultime phase concerne l'implémentation de l'idée à ajouter au code. Ce n'est pas la plus facile car il s'agit de traduire notre concept en lignes de code. Cette étape nous oblige à bien comprendre la notion qu'on étudie sous réserve de ne pas pouvoir l'implémenter.



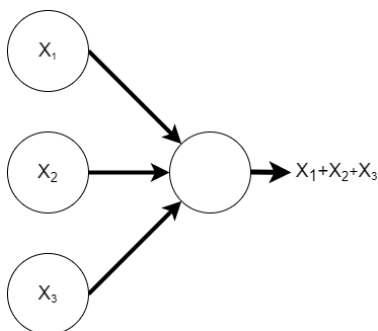
2-2- Le réseau neuronal

2-2-1- Qu'est-ce qu'un réseau neuronal ?

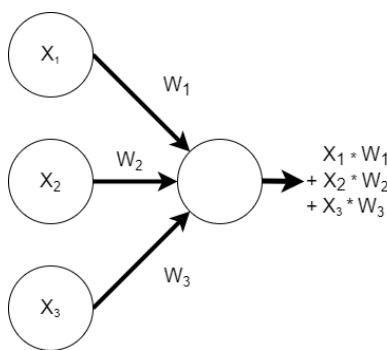
Tout d'abord, qu'est qu'un **neurone** ? Un neurone ici c'est très simple, il s'agit d'un élément qui prend des valeurs en entrées et les additionnent :



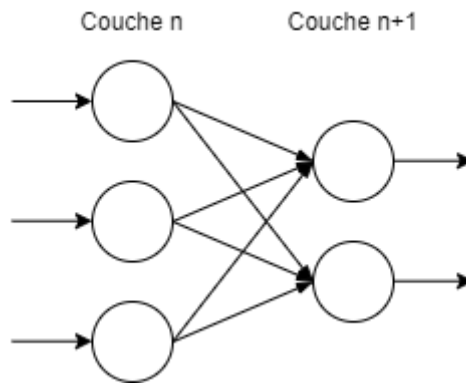
Dans un réseau neuronal, ses valeurs en entrées sont simplement les valeurs en sorties d'autres neurones :



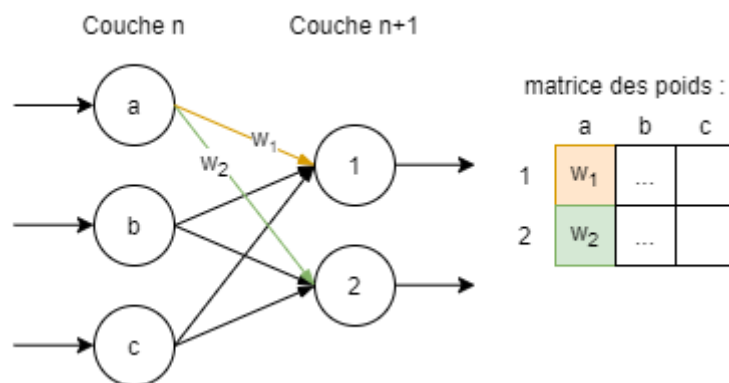
Mais il manque encore les éléments les plus importants du réseau neuronal : les **poids**. Les poids sont des valeurs affectées aux liaisons entre les neurones et qui déterminent à quel point un neurone va affecter la somme finale, la somme finale devient maintenant la somme des valeurs de sortie des neurones * le poids des liaisons :



Un réseau neuronal est composé de **couches de neurones** où chaque neurone de la nième couche sont les valeurs d'entrées de la couche $n+1$:

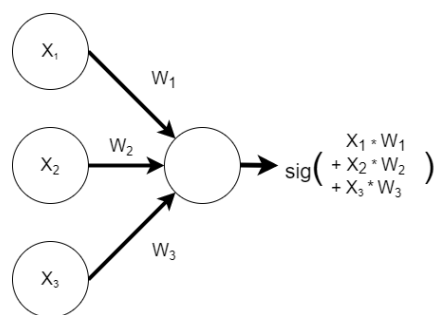


On peut constater que les poids peuvent être disposés dans une matrice de la manière suivante :

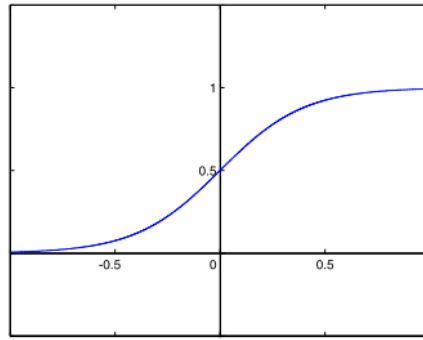


Enfin afin de garder les valeurs dans un ensemble réduit on passe la somme à ce qu'on appelle une **fonction d'activation**. De nombreuses fonction d'activation existe, ici nous avons

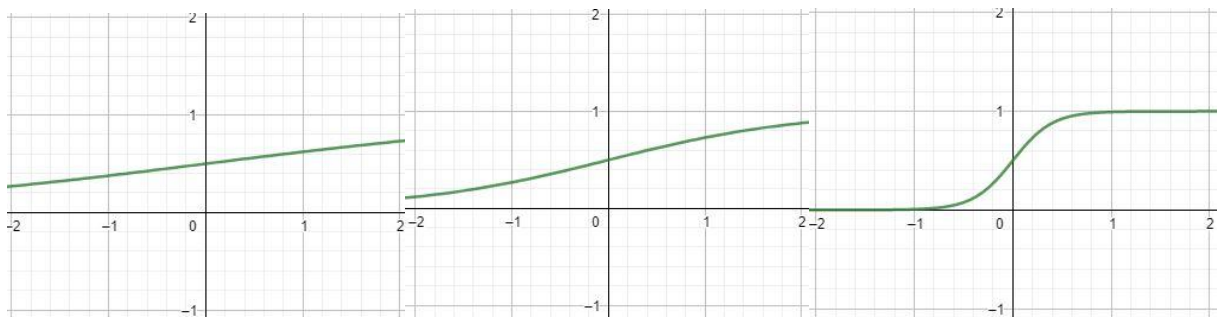
choisi une des plus utilisées, la fonction **sigmoïde** ($f(x) = \frac{1}{1 + e^{-x}}$) que l'on notera sig(x), ainsi les neurones ont maintenant pour sortie :



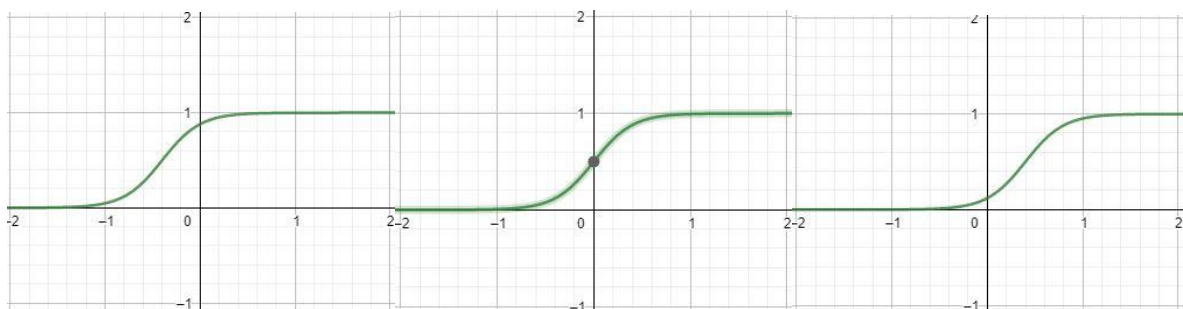
Qui est une valeur comprise entre -1 et 1 car la fonction sigmoïde a cette forme particulière :



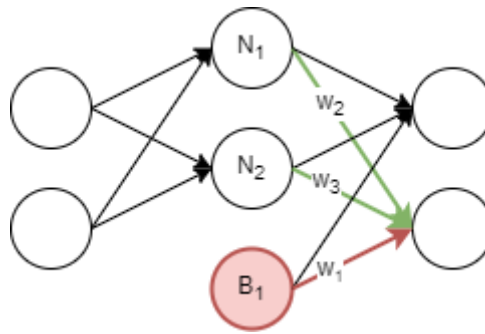
Dans le réseau neuronal au cours de son apprentissage, les éléments qui seront changé seront les poids car ils affectent la forme que prend la fonction sigmoïde dans un neurone et donc la sortie de celui-ci :



Dans ces exemple nous pouvons observer $\text{sig}(0.5x)$, $\text{sig}(x)$ et $\text{sig}(5x)$. Comme on peut le constater, la fonction est modifiée par le facteur, notons le w , devant x . Dans le reseau neuronal il s'agit du poids. Toutefois la fonction reste invariante sur certains points par exemple quel que soit le facteur $\text{sig}(0) = 0.5$. Pour changer ça, nous pouvons ajouter une valeur, notons la b , à x , ainsi nous avons $\text{sig}(wx+b)$:



Ici nous avons respectivement $\text{sig}(5x - 2)$, $\text{sig}(5x)$ et $\text{sig}(5x + 2)$ et cette fois si la fonction est déplacer sur l'axe des abscisses. Ce nouveau paramètre est-ce que l'on appelle un **biais**. Dans le réseau neuronal il apparait comme un neurone vers lequel aucun neurone ne pointe et dont la valeur de sortie est invariante, lui par contre pointe sur les neurones de la couche suivante et est lié par un poids qui va déterminer le " b " :



Nous avons maintenant un réseau neuronal fonctionnel (cf Annexe 2), pour l'utiliser il suffit de donner des valeurs aléatoires aux poids du réseau, fournir des valeurs aux neurones d'entrées (ceux au début de la chaîne), que chaque couche, l'une à la suite de l'autre, calcule la valeur de ses neurones et enfin de récupérer les valeurs en sortie des neurones de sortie (en fin de chaîne). Mais là notre réseau renverra une valeur aléatoire, il nous faut donc maintenant modifier chaque poids afin de faire faire à notre réseau ce que l'on veut, pour ça nous allons lui faire **apprendre**.

2-2-2- Apprentissage

L'apprentissage se présente sous la forme d'un algorithme qui, à partir de l'erreur faite par le neurone, va modifier les poids en fonction de leur participation à l'erreur.

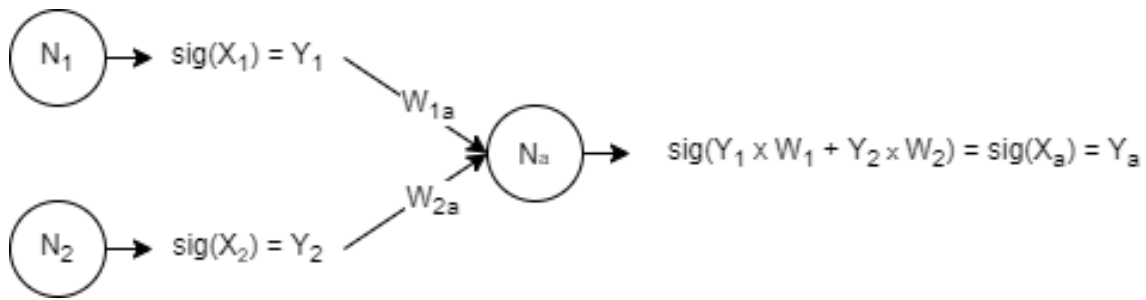
Tout commence par le choix d'une **fonction d'erreur** qui détermine le "montant" de l'erreur commise par le réseau pendant un entraînement. Nous avons décidé de choisir une fonction souvent utilisée : $E(y_{output}, y_{target}) = \frac{1}{2}(y_{output} - y_{target})^2$

y_{output} est ce que le réseau répond et y_{target} ce que l'on aura souhaité que le réseau réponde. Nous allons donc calculer comment l'erreur évolue en fonction d'un poids donné et mettre à jour le poids d'après cette fonction :

$$w_{ij} = w_{ij} - \alpha \frac{dE}{dw_{ij}}$$

α étant la vitesse d'apprentissage qui est à choisir, si elle est trop élevée ou trop basse le réseau neuronal n'apprendra pas correctement, trouver la valeur correcte se fait en essayant tout simplement.

Pour simplifier la compréhension du reste des calculs nous allons nous placer dans un contexte concret, celui-ci :



D'après le théorème de dérivation des fonctions composées (TDFC) :

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

Donc on peut dire :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial x_j}{\partial w_{ij}} \frac{\partial E}{\partial x_j} = y_i \frac{\partial E}{\partial x_j}$$

Si on prend W_{1a} comme exemple pour W_{ij} alors on a :

$$\frac{\partial E}{\partial W_{1a}} = \frac{\partial X_a}{\partial W_{1a}} \frac{\partial E}{\partial X_a}$$

Or, $X_a = Y_1 W_1 + Y_2 W_2$. Donc si on dérive en prenant W_1 comme variable alors, $X_a = Y_1$

$$\text{Et donc } \frac{\partial E}{\partial W_{1a}} = Y_1 \frac{\partial E}{\partial X_a}.$$

Maintenant si nous appliquons de nouveau le TDFC nous pouvons dire :

$$\frac{\partial E}{\partial X_a} = \frac{\partial Y_a}{\partial X_a} \frac{\partial E}{\partial Y_a}$$

Pour $\frac{\partial Y_a}{\partial X_a}$, il s'agit tout simplement de la dérivée de $f(x)$ tel que $f(X_a) = Y_a$ ici c'est la fonction d'activation que nous avons choisie (sigmoïde) dont la dérivée vaut $f(x)(1-f(x))$ donc on a :

$$\frac{\partial E}{\partial X_a} = f(X_a)(1 - f(X_a)) \frac{\partial E}{\partial Y_a}$$

Pour $\frac{\partial E}{\partial Y_a}$, étant donné que la fonction d'erreur que nous avons choisie est

$E(y_{\text{output}}, y_{\text{target}}) = \frac{1}{2}(y_{\text{output}} - y_{\text{target}})^2$, sa dérivée partielle par rapport à Y_a est tout simplement $Y_a - Y_{\text{target}}$.

Ainsi les derniers poids sont modifiable grâce à l'équation :

$$W_{ij} = W_{ij} - \alpha Y_{1j} f(X_a) (1 - f(X_a)) (Y_a - Y_{target})$$

Mais pour les poids suivants (ou précédant) on ne peut pas obtenir Y_{target} étant donné que l'on ne sait pas vraiment ce que l'on souhaite obtenir en sortie de la nième couche, il faut

donc obtenir $\frac{\partial E}{\partial Y_a}$ d'une autre manière, pour faire cela nous utilisons de nouveau le TDFC et nous obtenons :

$$\frac{\partial E}{\partial Y_i} = \sum_j^l \frac{\partial X_j}{\partial Y_j} \frac{\partial E}{\partial X_j} = \sum_j^l W_{ij} \frac{\partial E}{\partial X_j}$$

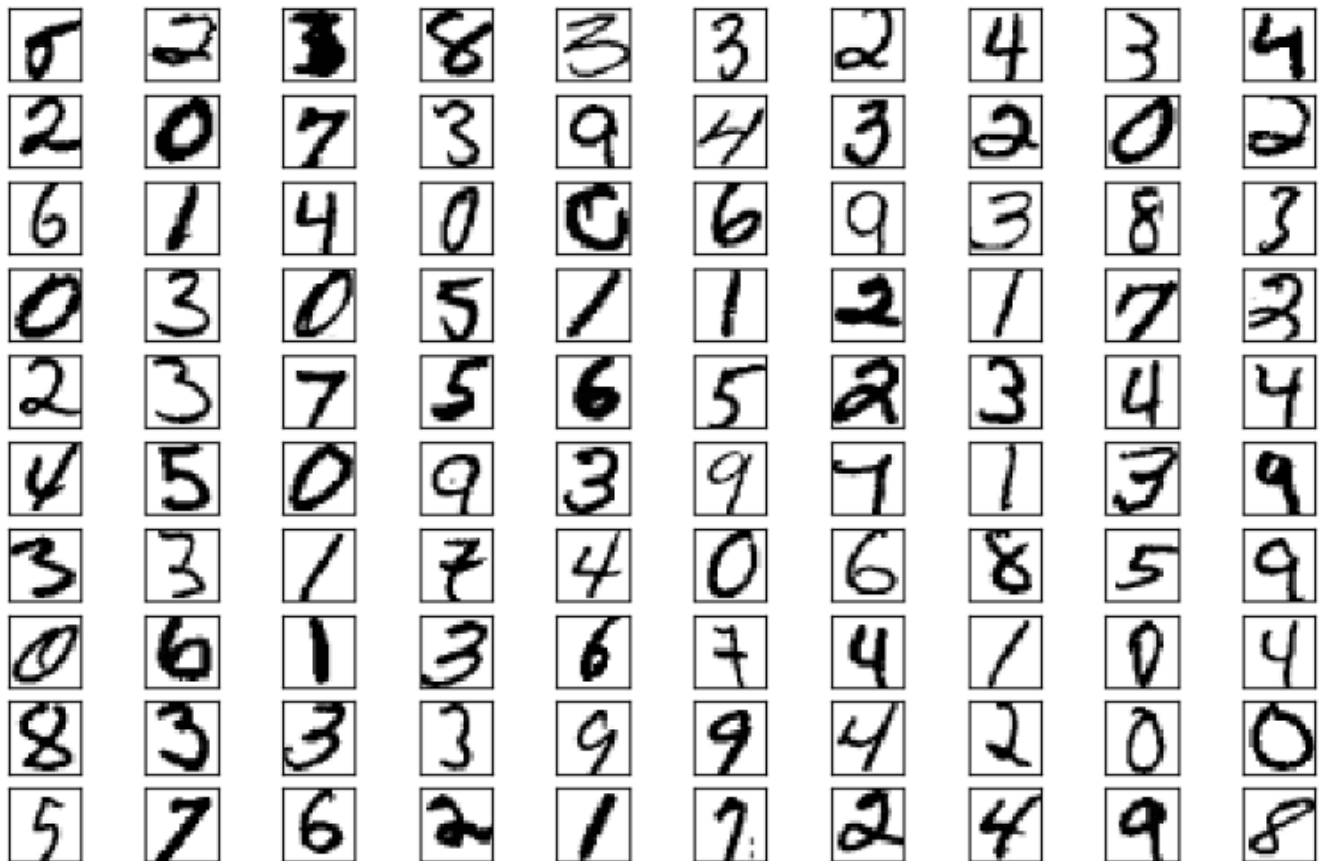
Nous avons donc désormais tout ce qu'il nous faut à notre disposition pour réaliser un algorithme partant de la fin du réseau neuronal vers le début en changeant chaque poids.

2-2-3- Sortie

Après que les entrées ont été transformées en sorties selon des règles précises, à savoir le modèle ou schéma du réseau neuronal adopté (qui dépend du nombre de couches cachées, du nombre de neurones par couche cachées, du nombre de neurones dans la couche d'entrée et de sortie, etc.).

Comme dit précédemment, l'interprétation des données dépend du modèle de notre réseau neuronal et des données rentrées préalablement dans la couche d'entrée.

Par exemple le modèle que nous avons le plus entraîné et le plus utilisé est celui où le réseau neuronal parvient à 91.6% de réussite à reconnaître sur une image de 28*28 pixels de quel chiffre il s'agit.



Le fichier que nous avons passé au réseau neuronal afin qu'il s'entraîne à reconnaître des chiffres contient exactement 40000 lignes, où chacune des lignes référence les 784 pixels correspondant à un chiffre entre 0 et 9.

Chaque pixel à une valeur entre 0 et 1 pour les nuances de gris, par exemple une ligne du fichier de données pourrait ressembler à cela :

5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.152625412,0.1559494,0.5984954,0.944421,[...],0,0,0,0,0,0

Ensuite, en interprétant ce pattern régulier, nous pouvons non difficilement l'implémenter à un réseau neuronal qui prendrait 784 neurones d'entrée (correspondant aux 784 pixels).

A l'aide d'un modèle de réseau neuronal adapté à ce genre d'épreuve, nous arrivons à un taux de reconnaissance de chiffre s'élevant à plus de 90%, et ce grâce à un réseau comportant :

- 784 neurones d'entrée
- 15 neurones dans la 1ere couche cachée
- 15 neurones dans la 2eme couche cachée
- 10 neurones de sortie

Nous n'avons pas trouvé ce modèle de réseau neuronal du premier coup, nous y sommes allés par tâtonnements en commençant avec un peu plus de couches cachées et beaucoup plus de neurones par couche cachée. Ce qui était une erreur de notre part que de penser que plus il y a de neurones, plus notre réseau neuronal sera précis.

3- Conclusion

En conclusion, le projet a été très intéressant dans la mesure où il nous a permis d'approcher pour la première fois une forme d'intelligence artificielle. De plus, nous avons pu apprendre le python, langage que nous n'avions jamais abordé auparavant. En outre, chacun d'entre nous a pu tirer une expérience de ce projet.

Les 4 mots d'ordre que nous avons choisis pour définir ce projet sont les suivants :

- **Rigueur** : La première des choses qui nous a été la plus importante est la rigueur, effectivement, la ponctualité, la régularité et l'exigence ont été clé au sein de ce projet pour mener à bien le bon déroulement de ce dernier.
- **Patience** : Il a été fondamental de se montrer persévérant face aux différents problèmes rencontrés lors de la compilation du code ou de l'incompréhension de certaines notions.
- **Curiosité** : De manière générale, nous devons essentiellement rester curieux et ce dans toutes les situations. En effet, lorsqu'on essaye d'acquérir une toute nouvelle connaissance, on se doit de s'intéresser sans cesse et de se questionner quant aux limites, à la compatibilité ou encore à la difficulté de réalisation.
- **Esprit d'équipe** : Une des choses qui nous ont sauvés dans maintes de situations est la collectivité. Développer un esprit d'équipe au sein d'un groupe de projet semble être pour nous inévitable. En réalité, le fait de toujours avoir quelqu'un à nos côtés nous a bien aidés tout au long du projet, notamment lors des phases de pair-programming.

Synthèses personnelles

Louis Chaillou

Cette expérience a été pour moi très positive. D'une part grâce à elle nous avons pu mettre en œuvre un projet de plusieurs mois au sein d'un véritable équipe, et donc diviser les tâches, être à l'écoute et s'entraider. Cet aspect du projet m'a conduit à avancer d'une façon beaucoup plus professionnelle et proche du fonctionnement des équipes projets en entreprise. D'autre part parce qu'elle a été totalement innovante pour chaque personne du groupe et moi-même, grâce au sujet qui portait sur de l'intelligence artificielle ou encore grâce au langage employé qu'était le python. Je finirais par dire que ce projet tutoré a rempli entièrement sa tâche qui était de nous former au travail d'équipe, d'aller plus loin que les cours de l'iut et surtout d'explorer un domaine différent qui pourrait peut-être nous intéresser

Lucca Anthoine

J'ai beaucoup aimé ce projet pour plusieurs raisons : tout d'abord, il m'a permis de m'essayer à la création d'intelligence artificielle, il m'a permis d'apprendre et de m'investir dans des notions mathématiques complexes, et il m'a permis de réaliser une application utile et concrète. Je pense d'ailleurs poursuivre ses recherches dans l'IA de manière personnelle.

Sylvain Thor

L'idée de pouvoir créer ma première intelligence artificielle, et ce avec une équipe à mes côtés, a été une des meilleures expériences (si ce n'est la meilleure) à ce jour en cette 2ème année de DUT informatique. En effet, la programmation étant la matière que j'apprécie le plus, le fait de créer un réseau neuronal étant capable de reconnaître des chiffres (à environ 90% de réussite) a été un projet très passionnant qui a su éveiller ma curiosité et susciter mon intérêt.

Hugo Sinprasih

Ce projet tutoré a été un gain d'expérience énorme que ce soit pour le côté conception ou bien le côté technique. En effet, une intelligence artificielle est beaucoup plus compliquée qu'elle n'en a l'air. De plus, il m'a permis d'apprendre énormément de choses notamment un nouveau langage de programmation ainsi que des outils de gestion de projets. Cependant, il s'agit d'un sujet auquel je me suis intéressé par curiosité et faire ce projet m'a permis de découvrir du concret et d'appliquer quelque connaissance mais surtout en découvrir de nouvelles.

Annexes

Annexe 1

RéseauNeuronal

26 févr. 2020

<http://>

Chef de projet	Luccas Anthoine
Dates du projet	16 oct. 2019 - 25 févr. 2020
Avancée	100%
Tâches	25
Ressources	4

Diagramme de GANTT du Projet Tuteuré "Réseau Neuronal"

Tâches

2

Nom	Date de début	Date de fin
Répartition des tâches/rôles	16/10/19	16/10/19
Phase de documentation/apprentissage	21/10/19	11/11/19
Réunion n°1	21/10/19	21/10/19
Apprentissage du langage Python	22/10/19	11/11/19
Documentation sur les réseaux neuronaux	22/10/19	11/11/19
Réunion n°2	11/11/19	11/11/19
Phase de conception logique	12/11/19	03/12/19
Réunion n°3	12/11/19	12/11/19
Renforcement des connaissances sur les réseaux neuronaux	13/11/19	02/12/19
Conception de diagramme représentant le réseau neuronal	13/11/19	02/12/19
Prise en main du module graphique Tkinter	13/11/19	02/12/19
Réunion n°4	03/12/19	03/12/19
Phase de développement	04/12/19	06/02/20
Réunion n°5	04/12/19	04/12/19
Développement de l'interface graphique	05/12/19	05/02/20
Exploitation des algorithmes	05/12/19	05/02/20
Réunion n°6	06/01/20	06/01/20
Développement de l'interprétation graphique du réseau	07/01/20	05/02/20
Mise en place du réseau (modèle logique de l'application)	07/01/20	05/02/20
Réunion n°7	06/02/20	06/02/20
Phase d'optimisation	07/02/20	24/02/20
Réunion n°8	07/02/20	07/02/20
Mise en commun du code	10/02/20	21/02/20
Réduction de l'erreur et amélioration de l'algorithme de reconnaissance	10/02/20	21/02/20
Réunion n°9	24/02/20	24/02/20


Ressources

3

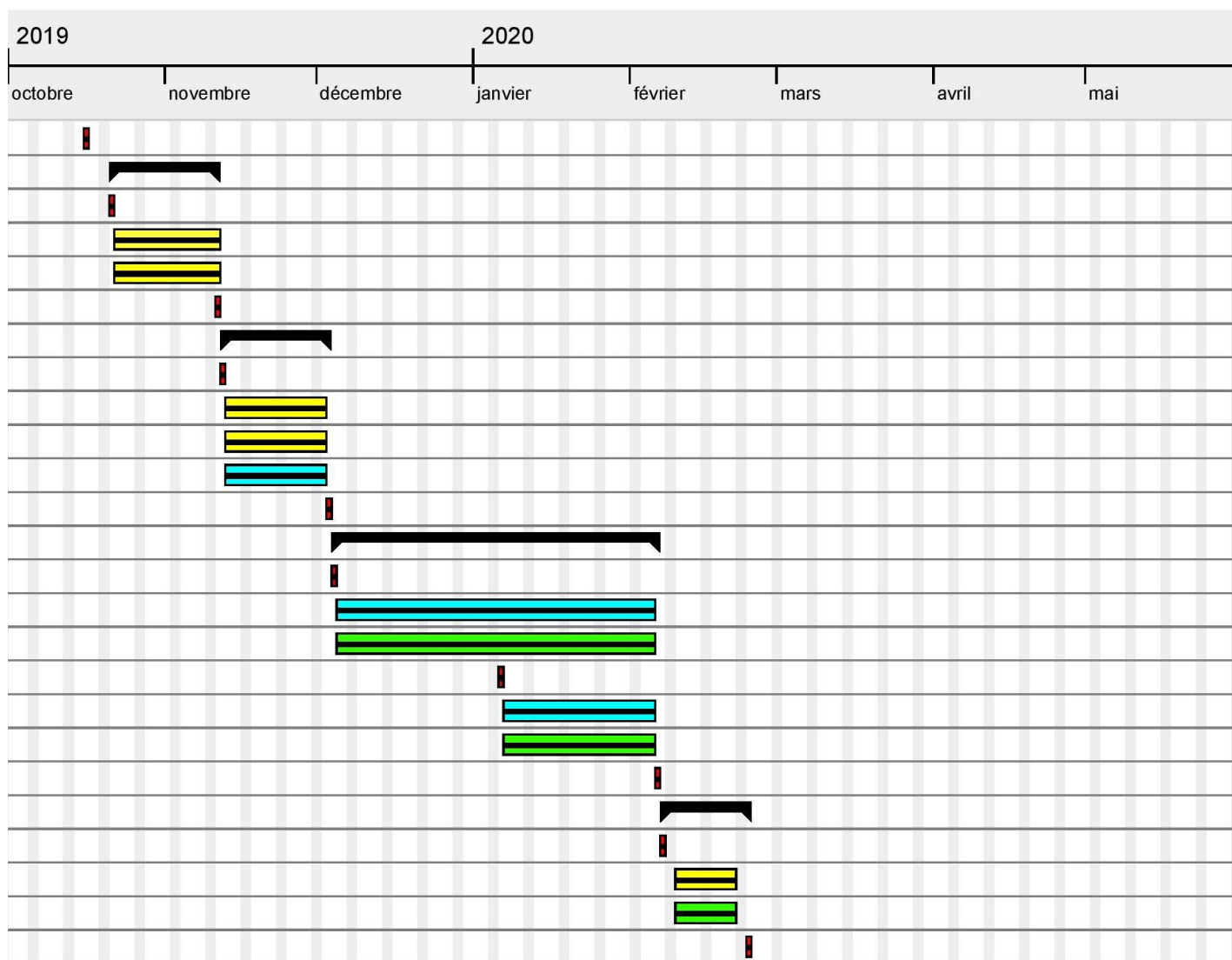
Nom	Rôle par défaut
Luccas Anthoine	Chef de projet
Louis Chaillou	Développeur graphique
Sylvain Thor	Développeur
Hugo Sinprasith	Développeur graphique

RéseauNeuronal

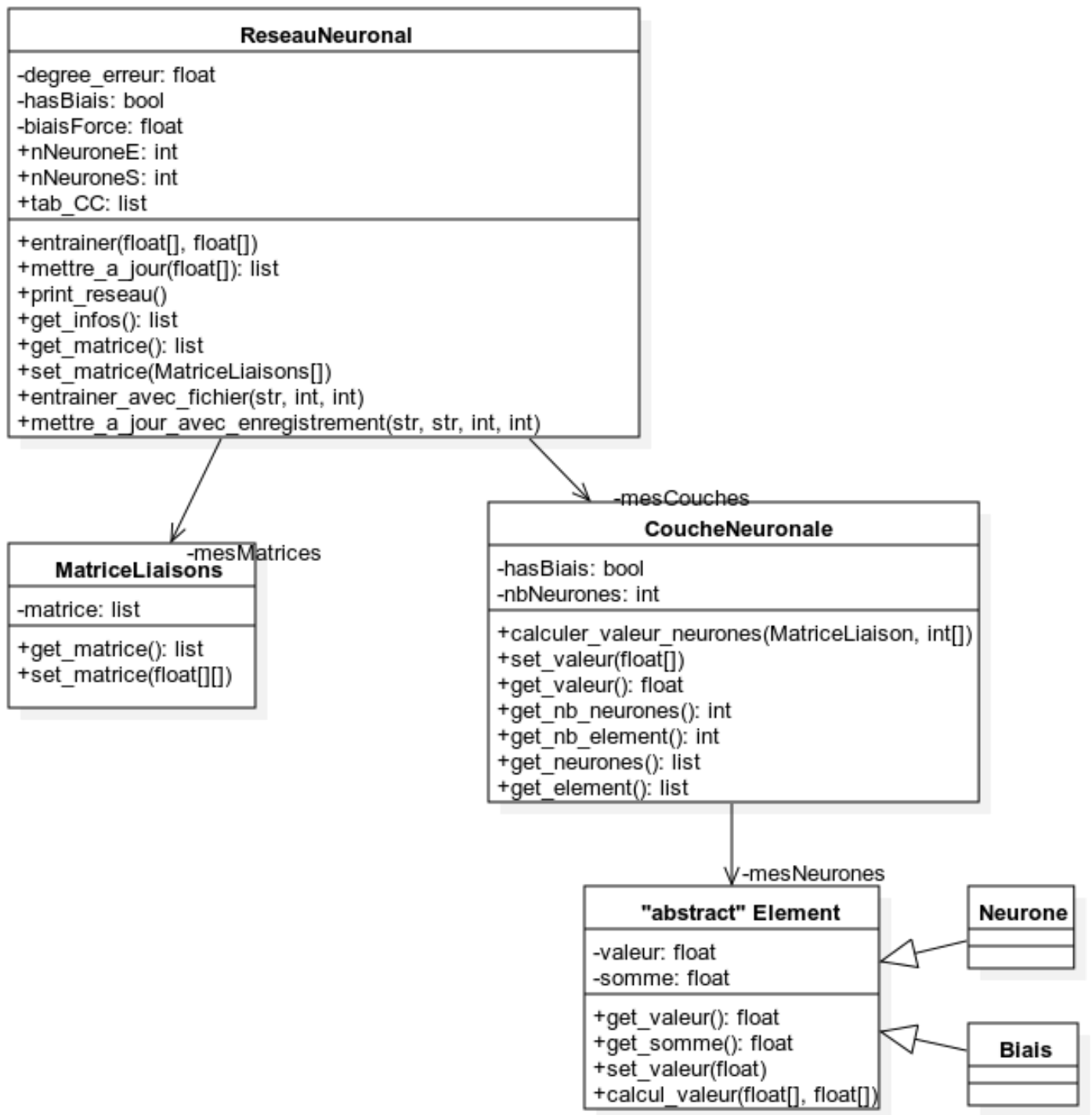
Diagramme de Gantt



Nom	Date de début	Date de fin
• Répartition des tâches/rôles	16/10/19	16/10/19
☐ • Phase de documentation/apprentissage	21/10/19	11/11/19
• Réunion n°1	21/10/19	21/10/19
• Apprentissage du langage Python	22/10/19	11/11/19
• Documentation sur les réseaux neuronaux	22/10/19	11/11/19
• Réunion n°2	11/11/19	11/11/19
☐ • Phase de conception logique	12/11/19	03/12/19
• Réunion n°3	12/11/19	12/11/19
• Renforcement des connaissances sur les réseau neuronaux	13/11/19	02/12/19
• Conception de diagramme représentant le réseau neuronal	13/11/19	02/12/19
• Prise en main du module graphique Tkinter	13/11/19	02/12/19
• Réunion n°4	03/12/19	03/12/19
☐ • Phase de développement	04/12/19	06/02/20
• Réunion n°5	04/12/19	04/12/19
• Développement de l'interface graphique	05/12/19	05/02/20
• Exploitation des algorithmes	05/12/19	05/02/20
• Réunion n°6	06/01/20	06/01/20
• Développement de l'interprétation graphique du réseau	07/01/20	05/02/20
• Mise en place du réseau (modèle logique de l'application)	07/01/20	05/02/20
• Réunion n°7	06/02/20	06/02/20
☐ • Phase d'optimisation	07/02/20	24/02/20
• Réunion n°8	07/02/20	07/02/20
• Mise en commun du code	10/02/20	21/02/20
• Réduction de l'erreur et amélioration de l'algorithme de rec...	10/02/20	21/02/20
• Réunion n°9	24/02/20	24/02/20



Annexe 2



Annexe 3

		Louis	Lucca	Sylvain	Hugo
APL					
Langage C	Affichage en console	x	x	x	x
	Gestion de versions	x	x	x	x
	Saisie en console	x	x	x	x
	Conditions	x	x	x	x
	Boucles	x	x	x	x
	Types	x	x	x	x

	Réels	x	x	x	x
	Tableaux	x	x	x	x
	Tableaux multidimensionnels	x	x	x	x
	Adresses	x	x	x	x
	Chaînes de caractères	x	x	x	x
	Fonctions	x	x	x	x
	Débogueur	x	x	x	x
	Organisation du code	x	x	x	x
	Allocation dynamique				
	Structures				
	Fichiers	x	x	x	x
	Listes chaînées	x			x
	Récursivité				
	Piles		x		
	Files		x		
Langage JAVA	Classes et objets	x	x	x	x
	Composants graphiques	x			x
	Mise en page	x			x
	Héritage	x	x	x	x
	Dessin	x		x	x
	Polymorphisme				
	Évènements	x			x
	Exceptions	x	x	x	x
	Flux d'octets				
	Flux de caractères	x	x	x	x
	Généricité				
	Récursivité				
	Listes	x	x	x	x
	Dictionnaires				
	Arbres				
		Louis	Lucca	Sylvain	Hugo
ACDA					
Usage	Synopsis	x	x	x	x
	Diagramme de cas d'usage	x	x	x	x
Structure	Diagramme de classe	x	x	x	x
	Diagramme objet				
Comportement	Diagramme de séquence système	x	x	x	x
	Diagramme de classe	x	x	x	x
Abstraction	Classe abstraite	x	x	x	x
	Interface				
	API		x		
	Couplage faible				
Patron de conception					
Gestion de projet	SCRUM	x	x	x	x
	MoSCoW	x	x	x	x
	Qualité	x	x	x	x
		Louis	Lucca	Sylvain	Hugo

Mathématiques (Liste exhaustive car trop de notions)				
Matrice	x	x	x	
Dérivée partielle		x	x	
Dérivée		x	x	
Théorème de dérivation des fonctions composées		x	x	
Fonction sigmoïde		x	x	
Perceptron		x	x	
Erreur quadratique		x	x	
Perceptron multicouche		x	x	
Rétropropagation de l'erreur		x	x	