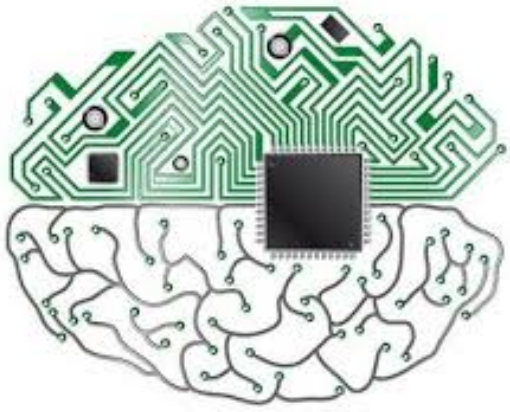
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Projet C++
GM4-2018/2019

Rapport de projet C++

Logiciel de création et d'entrainement de réseaux de neurones

Juin 2019



Etudiants : David Albert, Kevin Maouchi, Marion Schaeffer, Coralie Solaz, Adrien Vason et Samra Youssouf

A l'attention de : M.Kotowicz

Table des matières

Partie 1

Introduction

Pour notre projet C++, nous avons réalisé un logiciel de création et d'entraînement de réseaux de neurones. A partir des entrées spécifiées par l'utilisateur, le système crée un réseau de neurones et peut l'entraîner. Les entrées seront les images, le nombre de couches ainsi que leur type, la fonction de coût, l'algorithme d'optimisation, etc. La sortie se fera par un affichage sur le terminal avec les principaux résultats.

Tout au long de ce projet, nous avons pu manipuler les notions introduites durant le cours de C++. Nous avons souhaité utiliser au maximum les différents outils présentés pour ainsi produire des structures de données génériques et réutilisables ainsi qu'un code modulable et modifiable de façon très simple.

Devant la grande quantité de fonctionnalités que nous souhaitions développer, nous avons fixé un ordre de priorité afin de commencer par implémenter les fonctionnalités principales et essentielles du logiciel, puis ensuite intégrer celles utiles mais moins centrales, qui relevaient plus d'une amélioration que d'une véritable fonctionnalité. Voici donc les tâches que nous souhaitions accomplir dans l'ordre de leur priorité :

- Construire une structure de données pour les réseaux de neurones
- Construire une structure de données pour les calculs faits au sein du réseau, que l'on appellera Tenseur
- Créer les mécanismes nécessaires à la création d'un réseau de neurones
- Créer les mécanismes de mise à jour du réseau de neurones lors de modifications
- Créer les outils graphiques nécessaires à la création d'un réseau de neurones
- Créer les fonctions nécessaires à l'apprentissage du réseau de neurones
- Créer les fonctions nécessaires à la récupération des images et à leur prétraitement dans le but de les passer en entrée d'un réseau de neurones
- Créer les fonctions permettant la sauvegarde des réseaux de neurones créés/entraînés
- Construire un jeu de données et une structure de réseau de neurones pour concevoir un exemple de discrimination d'images avec le logiciel créé

Nous reviendrons par la suite sur chacune de ces étapes en détaillant celles effectuées et en expliquant pourquoi les autres n'ont pas été développées (et comment le faire dans le futur).

Dans ce rapport, nous commencerons par une partie théorique où nous présenterons les structures de données que nous avons créées pour implémenter notre logiciel. Nous reviendrons ensuite sur les spécifications et les conceptions préliminaire et détaillée pour mettre en évidence les modifications effectuées depuis le début de l'implémentation et en expliquer les causes. Nous détaillerons ensuite l'implémentation et les tests effectués. Enfin, nous vous présenterons un manuel d'utilisation avant de conclure en abordant les perspectives d'évolution de notre logiciel.

Partie 2

Notions théoriques

Nous allons expliquer quelques notions théoriques sur les réseaux de neurones et les mathématiques que nous avons utilisées pour notre logiciel.

2.1 Représentation d'un tenseur

Pour le stockage des données et des coefficients synaptiques, nous avons utilisé des tenseurs, qui sont la généralisation des vecteurs et des matrices. Les tenseurs sont caractérisés par un ordre et un ensemble de dimensions. L'ordre représente le nombre de dimensions. Par exemple, une matrice est un tenseur d'ordre 2. Si on considère que cette matrice est de dimension 15×10 alors ce sera un tenseur d'ordre 2 avec pour dimensions 15 et 10. On peut par la suite avoir des cubes matriciels (Tenseur d'ordre 3) et plus encore. Nous pourrons grâce à cette structure de données, utiliser n'importe quel type de données comme entrée d'un réseau de neurones.

Exemples de données possibles :

Donnée	Représentation discrète	Ordre du tenseur
Valeur numérique	$\mathbf{R}$	0
Son / Fonction	$\mathbf{R}^{freq}$	1
Musique	$\mathbf{R}^{freq} \times \mathbf{R}^{temps}$	2
Image 2D	$\mathbf{R}^{longueur} \times \mathbf{R}^{hauteur} \times \mathbf{R}^{couleur}$	3
Image 3D	$\mathbf{R}^{longueur} \times \mathbf{R}^{hauteur} \times \mathbf{R}^{profondeur} \times \mathbf{R}^{couleur}$	4
Film 2D	$\mathbf{R}^{longueur} \times \mathbf{R}^{hauteur} \times \mathbf{R}^{couleur} \times \mathbf{R}^{temps}$	4

2.2 Représentation d'un réseau de neurones

On introduit la structure de données "réseau de neurones" tel que suit : R = {S, A, I, F} avec :

S : les sommets du réseau de neurones (= des couches ou fonctions)
$A \subset S \times S$: les arcs orientés, ils représentent les données de sortie d'une couche $s_1 \in S$ qui fluctuent vers l'entrée d'une autre couche $s_2 \in S$
$I \subset S$: les couches initiales (ou d'entrée)
$F \subset S$: les couches finales (ou de sortie)

Pour représenter un réseau de neurones, on passe par l'intermédiaire d'un graphe. En effet, un réseau de neurones peut être vu comme un graphe particulier où chaque sommet est une fonction et chaque arc représente un flux de données. En tant que graphe on peut donc lui appliquer toutes les opérations classiques sur les graphes : ajout/suppression de noeuds, ajout/suppression d'arcs, vérification de la connectivité, de la présence d'un cycle, création de listes de sommets adjacents et de sommets antécédents,etc. Nous avons donc implémenté toutes ces fonctionnalités pour un graphe générique et utilisé la notion d'héritage pour en faire bénéficier un réseau de neurones tout en y ajoutant les fonctionnalités qui lui sont spécifiques.

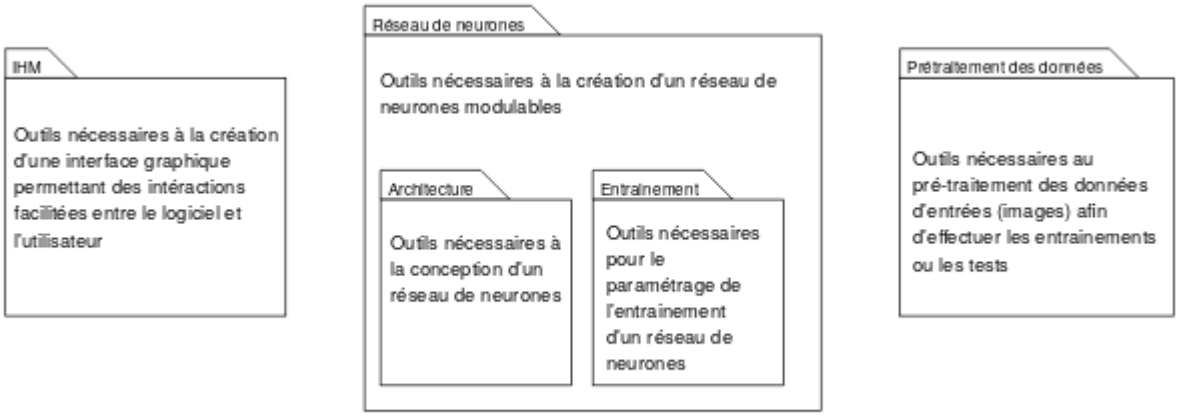
Une couche représente une fonction mathématique sur un tenseur (voir ci-dessus). Celle-ci peut être de deux types : soit de combinaison, soit d'activation. Les couches de combinaison sont des couches à paramètres. Ce sont elles qui vont porter les poids synaptiques. Les couches d'activation, quant à elles, appliquent de simples fonctions qui permettent d'envoyer un stimuli qui sera transmis vers les couches suivantes.

Du fait du manque de temps, nous avons choisi de ne traiter qu'une petite partie des architectures de réseaux de neurones possibles. Ainsi nous avons fait en sorte de ne pas pouvoir créer de réseaux de neurones récurrents. C'est-à-dire qu'il n'est pas possible de créer un réseau de neurones qui contient un cycle. De plus, nous n'avons pas pu implémenter autant de couches que nous le souhaitions. Dans cette première version, seuls des réseaux de neurones très basiques pourront être créés.

Partie 3

Spécifications

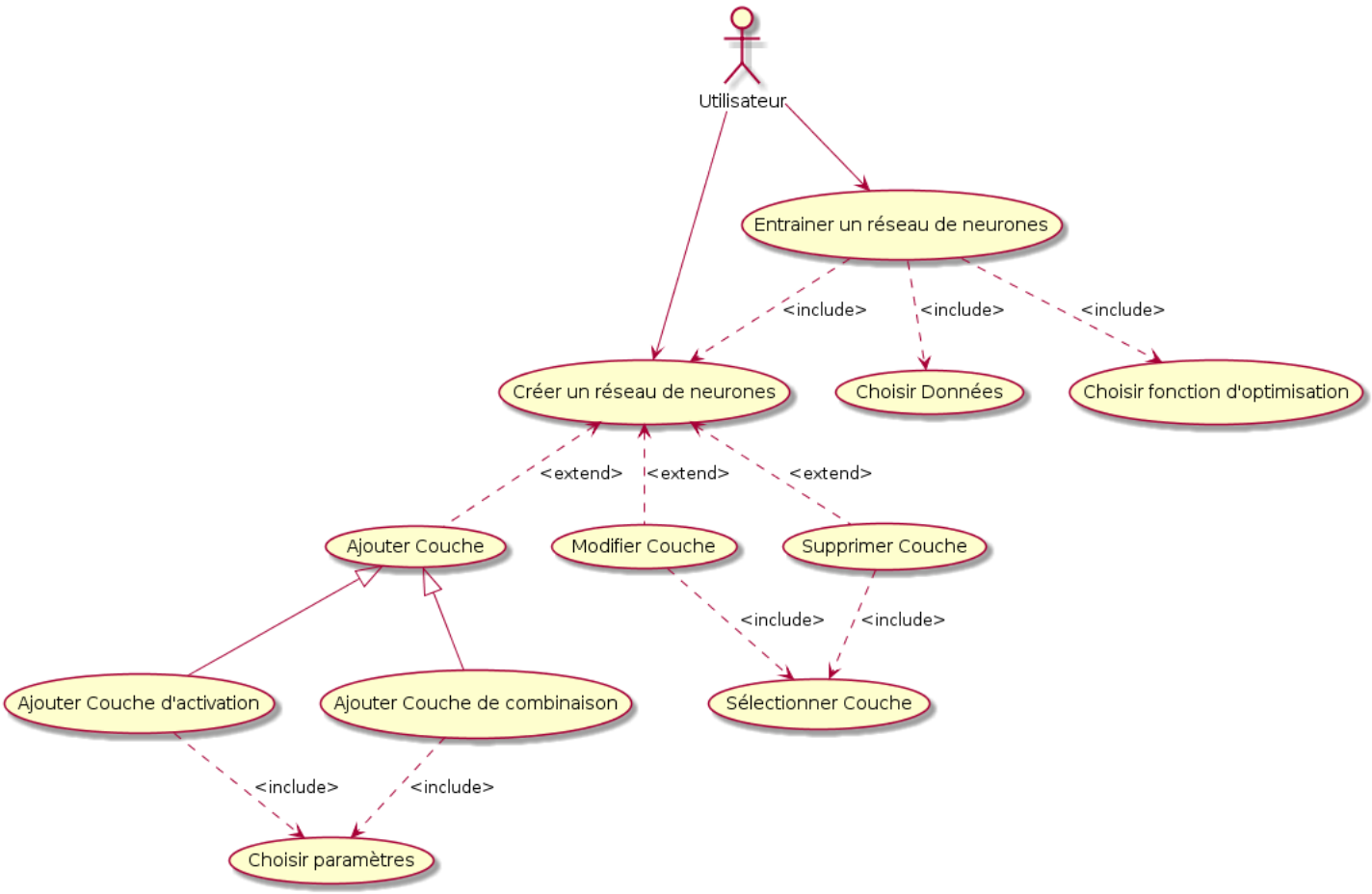
3.1 Modules



Pour la création de notre logiciel, nous sommes restés fidèles à la décomposition en 3 modules différents proposée avant l'implémentation.

- Un module pour l'interface graphique : il regroupe les fonctions permettant la création de manière graphique du réseau de neurones ainsi qu'une intéraction facilitée entre l'utilisateur et le logiciel
- Un module pour le choix des données qui engendrent leur prétraitement
- Un module pour le codage du réseau de neurones : il regroupe les fonctions permettant le développement d'un réseau de neurones personnalisé. Il comprend à son tour 2 modules différents :
  - Un module pour la création de l'architecture du réseau de neurones (ajout/suppression de noeuds/arcs)
  - Un module pour l'entraînement du réseau de neurones, c'est-à-dire le paramétrage de l'entraînement ainsi que son déroulement

3.2 Cas d'utilisation



3.2.1 Description des scénarios

Nous nous sommes concentrés sur 2 scénarios principaux :

— Créer un réseau de neurones : on pourra ajouter un type de couche en choisissant les paramètres associés, modifier ou supprimer une couche existante en la sélectionnant. On pourra ensuite entraîner le réseau créé.

— Entraîner un réseau : on devra d'abord créer un réseau de neurones puis choisir un jeu de données d'entraînement et les fonctions d'optimisation souhaitées.

3.2.2 Spécifications détaillées par cas d'utilisation

Création

Pour créer un réseau de neurones, on doit pouvoir ajouter une couche. Cette dernière peut être de deux types différents. Elle peut être de combinaison, c'est-à-dire qu'elle va permettre l'amélioration des paramètres du réseau dans la phase d'entraînement. Elle peut aussi être d'activation, c'est-à-dire qu'elle va réaliser un dernier traitement sur les données (par exemple les normaliser grâce à une fonction sigmoïde). Il existe des modèles classiques pour ces deux types de couche qui ont des paramètres prédéfinis relatifs à leur nombre d'entrées, leur nombre de sorties et comment leurs connexions avec les autres couches s'opèrent.

Notons que cette phase de création peut également être vue comme une phase de modification d'un réseau de neurones. En effet, l'utilisateur peut tout à fait décider de modifier et/ou supprimer une couche. Pour ce faire, il devra bien entendu la sélectionner au préalable. Cette possibilité implique de devoir vérifier que les paramètres des couches adjacentes ne risquent pas de rentrer en conflit après l'action de l'utilisateur.

Entraînement

La phase d'entraînement d'un réseau de neurones doit pouvoir être réalisable directement après sa création sans pour autant l'obliger. Une fois le choix du réseau de neurones effectué, il suffit de définir la fonction d'optimisation et le jeu de données d'apprentissage à utiliser pour l'entraînement. Les fonctions d'optimisation disponibles correspondent à celles qui sont le plus utilisées dans le domaine du machine learning. Le jeu de données implique un prétraitement pour s'assurer qu'elles ont le même format.

3.2.3 Principales modifications

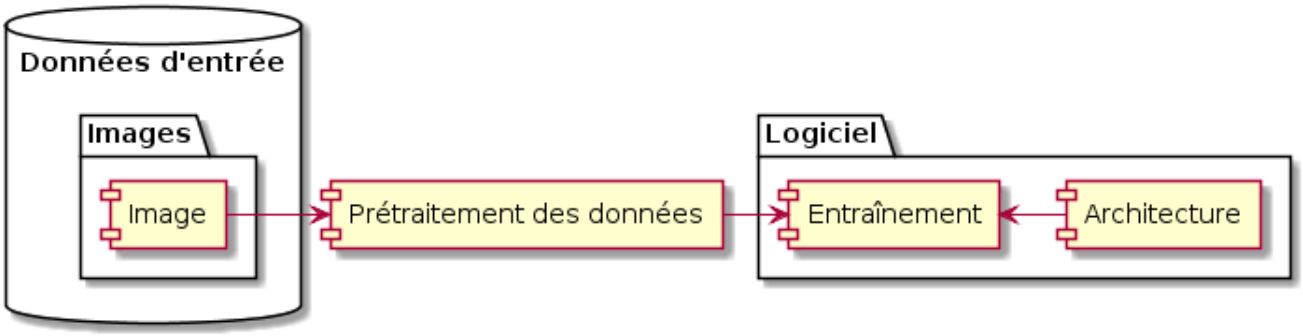
Les modifications que nous avons faites pour les cas d'utilisation reflètent les fonctionnalités que nous n'avons pas eu le temps d'implémenter. En effet, nous n'avons pas eu le temps d'implémenter les fonctionnalités permettant la sauvegarde d'une architecture de réseau de neurones. Il est donc impossible de sauvegarder un réseau après sa création, de charger un réseau déjà crée pour l'entraîner,.. Il faut donc créer son réseau et l'entrainer directement après si l'objectif de l'utilisateur est d'entraîner son réseau car dès que l'on quitte le logiciel tout le travail est effacé. Nous n'avons pas non plus développé la partie test d'un réseau de neurones car nous avons mis beaucoup de temps à implémenter la partie entraînement et nous avons fait le choix de nous concentrer sur cette dernière plutôt que sur celle concernant le test d'un réseau.

Pour les parties concernant la création d'un réseau de neurones, nos cas d'utilisations n'ont pas changé et l'implémentation faite respecte bien ce que nous avions représenté lors des premiers rendus.

Partie 4

Conception préliminaire

4.1 Intéractions entre les composants



Notre base de données représente les données d'entrées utilisées pour l'entraînement. Ces données sont d'abord prétraitées par un module externe. Pour le module de prétraitement, les entrées sont un fichier d'images et la sortie est stockée dans une structure de données appelée **Tenseur** qui sera utilisée par le réseau de neurones.

Concernant notre logiciel, le module d'architecture ne prendra pas d'entrée extérieure mais produira en sortie un réseau de neurones avec l'architecture et les paramètres aléatoires du réseau. A l'inverse, le module d'entraînement prendra en entrée un réseau de neurones non entrainé (paramètres aléatoires) ainsi que les données d'entrée prétraitées et produira un réseau de neurones avec ses paramètres mis à jour.

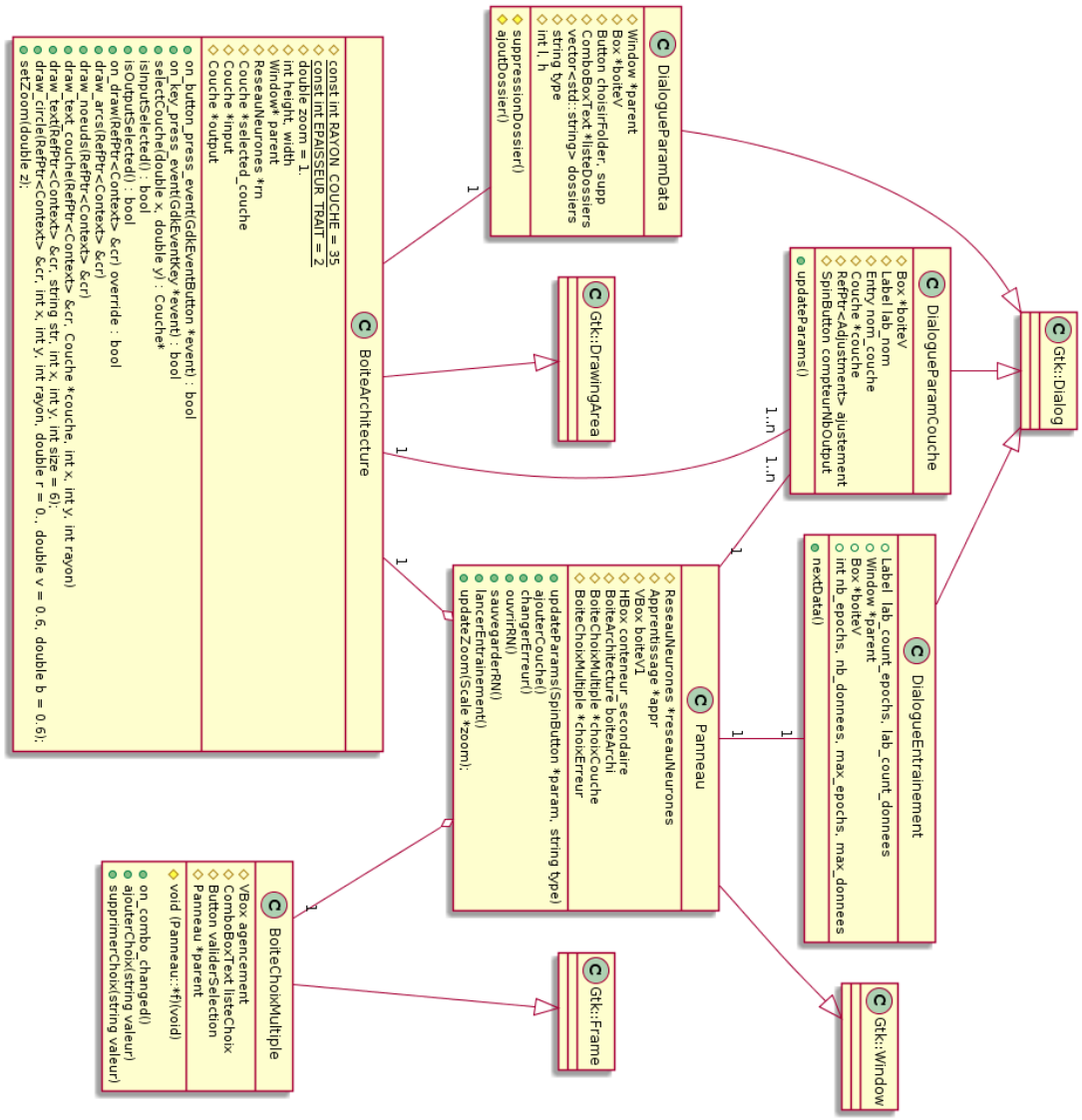
4.2 Principales modifications

Les modifications sur le diagramme d'intéraction entre les composants sont dûes à l'absence de sauvegarde dans des fichiers. En effet, nous souhaitions pouvoir enregistrer un réseau de neurones créé pour pouvoir à nouveau l'utiliser pour créer un autre réseau ou alors pour l'entraîner ou le tester. Malheureusement cela n'a pas été possible. On ne peut donc pas charger d'architecture existante ni diriger la sortie de l'entraînement vers un fichier. Les mêmes modifications sont visibles à cause de la disparition du module de test, tous les fichiers qui s'y référaient ne sont donc plus utiles.

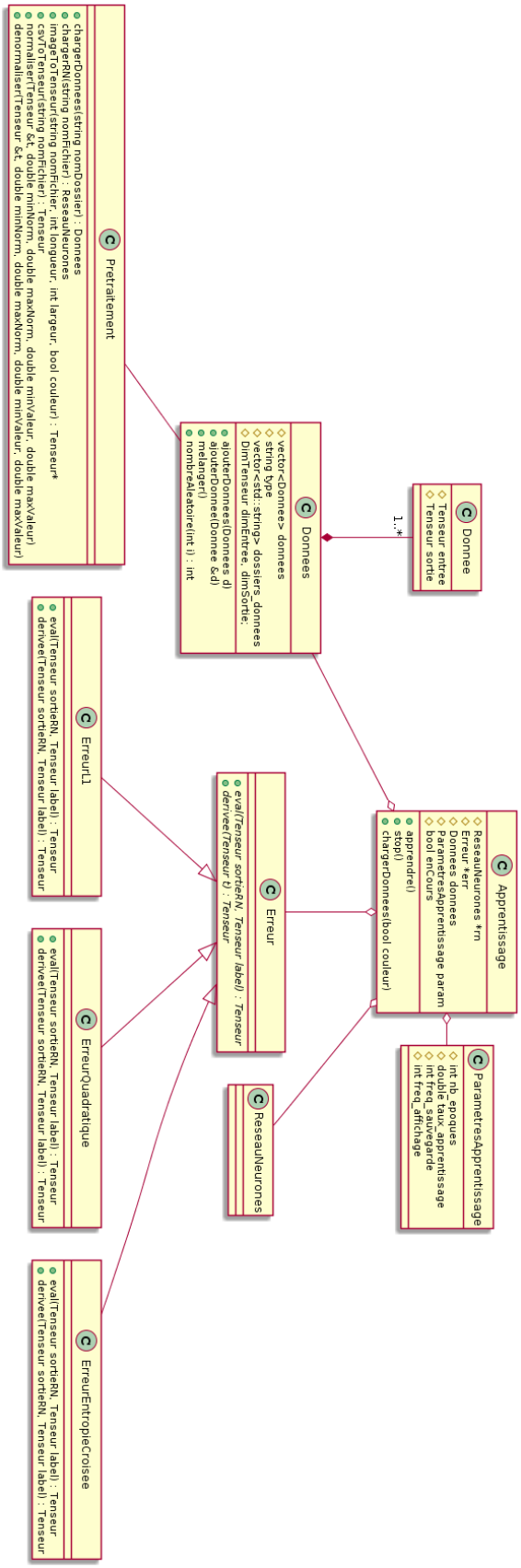
Partie 5

Conception détaillée

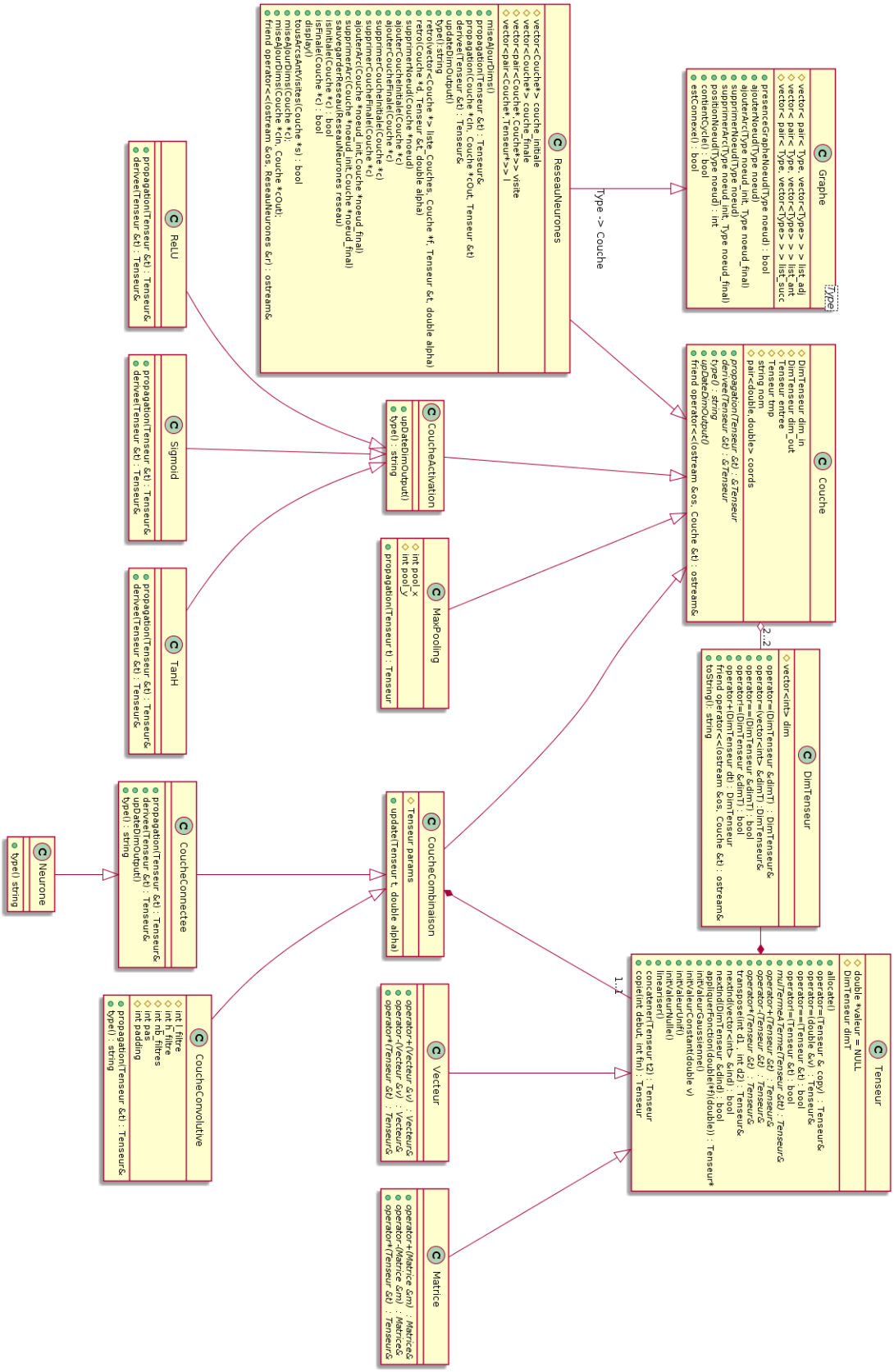
5.1 Diagramme de Classe pour l'interface Homme-Machine



5.2 Diagramme de Classe pour l'entraînement



5.3 Diagramme de Classe pour l'architecture d'un réseau de neurones



Partie 6

Implémentation et tests

6.1 Choix d'implémentation

6.1.1 Réseau de neurones

Architecture

Pour implémenter l'architecture d'un réseau de neurones, nous avons utilisé les propriétés des graphes que nous avons implémenté. Pour représenter les noeuds, nous avons utilisé des "template" pour rendre le code réutilisable, en particulier pour les algorithmes de parcours de graphe.

Nous avons implémenté la structure de données **Tenseur** ainsi que les tenseurs particuliers **Vecteur** et **Matrice**. Nous avons utilisé la surcharge d'opérateurs afin de permettre l'utilisation des opérations usuelles sur les tenseurs. Le logiciel peut donc très bien être utilisé pour faire du calcul tensoriel.

Nous avons développé différents types de couches parmi celles les plus utilisées en machine learning :
- couche d'activation
  - ReLu
  - TanH
  - Sigmoïd
- couche de combinaison
  - couche convolutive
  - couche connectée

Les méthodes d'héritage et de polymorphisme disponibles en programmation objet ont été nécessaires à leur implémentation. En effet, toutes ces couches héritent de la classe astraite **Couche** qui est un moule générique regroupant les principales fonctionnalités d'une couche. Un réseau de neurones n'est autre qu'une fonction, donc en plus de faire hériter de Graphe la classe réseau de neurones, nous avons choisi de la faire également hériter de Couche. Ainsi un réseau de neurones peut être propagé, il a une dimension d'entrée et de sortie et un réseau de neurones peut être constitué de plusieurs sous-réseaux de neurones, eux mêmes constitués de couches, etc.

Enfin, nous avons utilisé de nombreuses exceptions liées aux graphes (noeud inexistant, arc déjà créé, graphe non connexe, graphe comprenant un cycle) ainsi que des exceptions liées aux opérations sur les tenseurs (dimensions incompatibles).

Entraînement

Le module d'entraînement comprant les erreurs. La classe abstraite **Erreur** est héritée par les différents types d'erreur :
- erreur L1
- erreur d'entropie croisée
- erreur quadratique

Nous avons représenté les images avec la classe **Donnee**. On décompose une donnée en une entrée qui est l'image traitée et en une sortie qui est la caractérisation de l'image. L'ensemble de données d'apprentissage est regroupé dans une classe **Donnees**.

Pour le déroulement de l'entraînement, nous avons besoin de nombreux paramètres comme le nombre d'époques (nombre de passages sur toutes les données lors de l'apprentissage), le taux d'apprentissage qui représente la vitesse à laquelle le réseau de neurone apprend, ainsi que la fréquence d'affichage du résultat de la propagation dans le réseau. Tout ceci est récupéré et géré dans la classe **ParametresApprentissage**.

Pour finir, l'apprentissage a lieu dans une classe du même nom ayant pour paramètre un réseau de neurones et appliquant à ce dernier successivement la propagation des données puis la rétropropagation de l'erreur engendrée. Une erreur de dimension apparait lors de la rétropropagation et nous n'avons pas réussi à en identifier la source. Nous avons donc laissé uniquement la propagation et l'évaluation de l'erreur pour l'apprentissage. Une perspective d'évolution prioritaire serait la résolution de ce problème car le réseau n'est pour l'instant pas entraîné si on ne rétropropage pas son erreur.

6.1.2 Module IHM

Pour l'IHM, nous avons utilisé la librairie GTKMM qui est une librairie graphique pour C++. En effet c'est en quelque sorte la version C++ de la librairie GTK.

L'IHM a été simplifié par rapport à ce qui était prévu dans la première conception du projet. Nous avons choisi, par soucis de simplicité, de rassembler la partie architecture et entrainement. Ainsi, il n'y a maintenant plus qu'un composant graphique général qui permet de créer le réseau de neurones et de paramétrer l'apprentissage en même temps. Nous avons de plus fait le choix de n'avoir à utiliser que la souris pour l'utilisation du logiciel. Le manuel d'utilisation est disponible plus loin (voir 7). L'interaction avec des boites de dialogue pour faciliter le paramétrage est grandement présent dans le logiciel (à l'image des grands logiciels notamment de CAO tels que Catia ou FreeCad).

6.1.3 Module Prétraitement

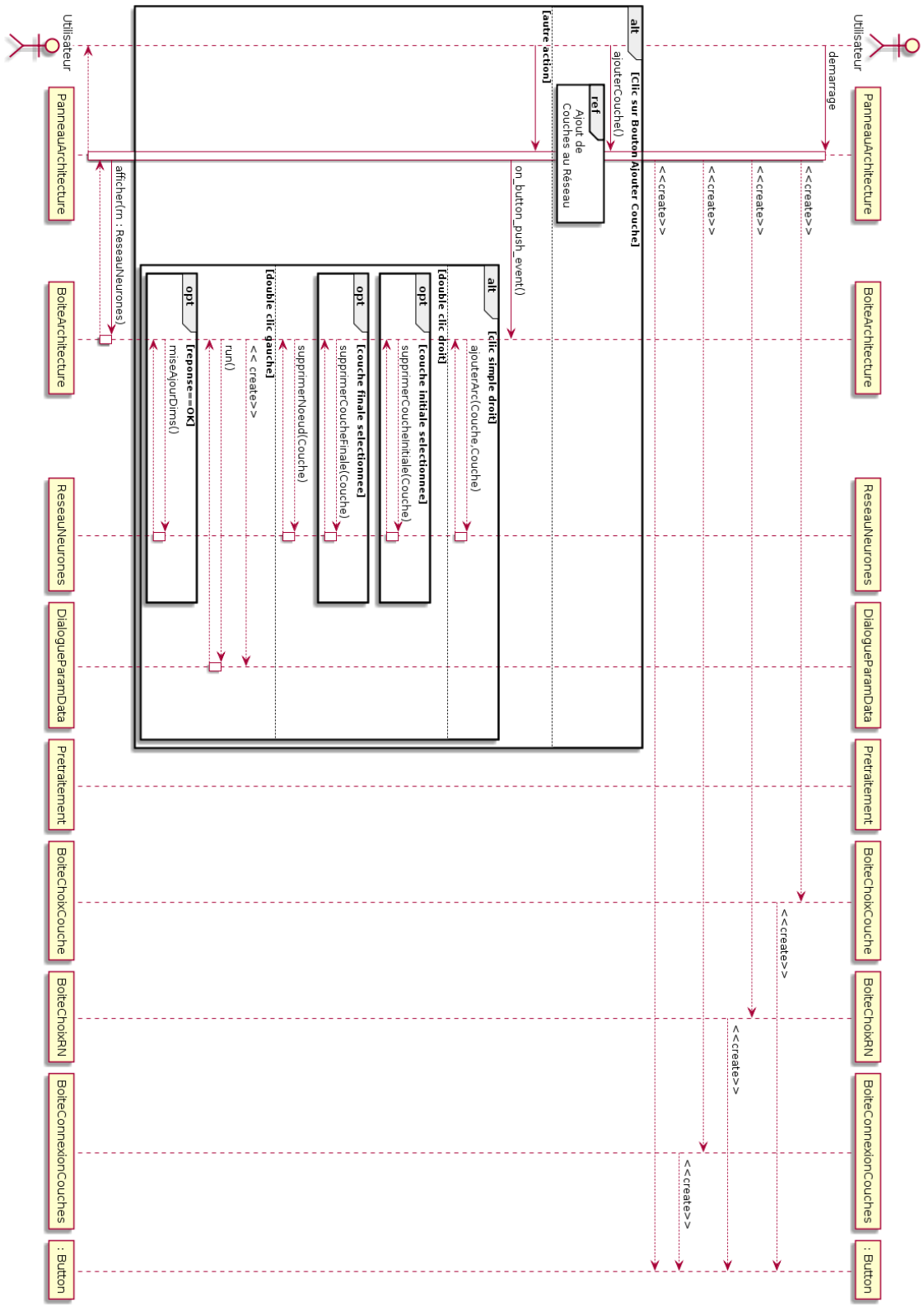
Nous avons utilisé la librairie Magick++ pour prétraiter des images de manière simple. Magick++ est à l'image de la célèbre librairie OpenCV et permet de faire du traitement d'images simplement. Contrairement à OpenCV, la version C++ est beaucoup plus simple à installer. C'est la raison pour laquelle nous l'avons choisie. Les fonctionnalités de Magick++ que nous avons utilisées sont la lecture d'une image, le redimensionnement de celle-ci, la transformation en noir et blanc et la récupération des valeurs des pixels. En effet, lors du prétraitement des images, on les redimensionne toutes avec la taille choisie par l'utilisateur. Ensuite, on applique ou non la fonction noir et blanc en fonction du choix de l'utilisateur puis on copie chaque pixel de l'image $Magick++$ dans un Tenseur pour pouvoir utiliser les images comme entrées du réseau de neurones. Ainsi pour les images en couleur, on crée un tenseur à 3 dimensions. Les 3 dimensions représentent la largeur en pixel, la hauteur et la couleur. Pour coder la couleur, on stocke le niveau de vert, de rouge et de bleu du pixel sur une échelle allant de 0 à 1. Lorsque l'image est en noir et blanc, le tenseur associé a uniquement deux dimensions représentant sa largeur et sa hauteur, où l'on stocke son niveau de gris sur une échelle allant également de 0 à 1. Des erreurs surviennent parfois lors du chargement des données de façon inexplicable. D'un jeu de données à l'autre, l'erreur apparaît ou non, de même pour les nouvelles dimensions choisies. Si ce problème se présente, il est conseillé de faire plusieurs essais. Une nouvelle fois, cette difficulté apparaît être une perspective d'amélioration prioritaire.

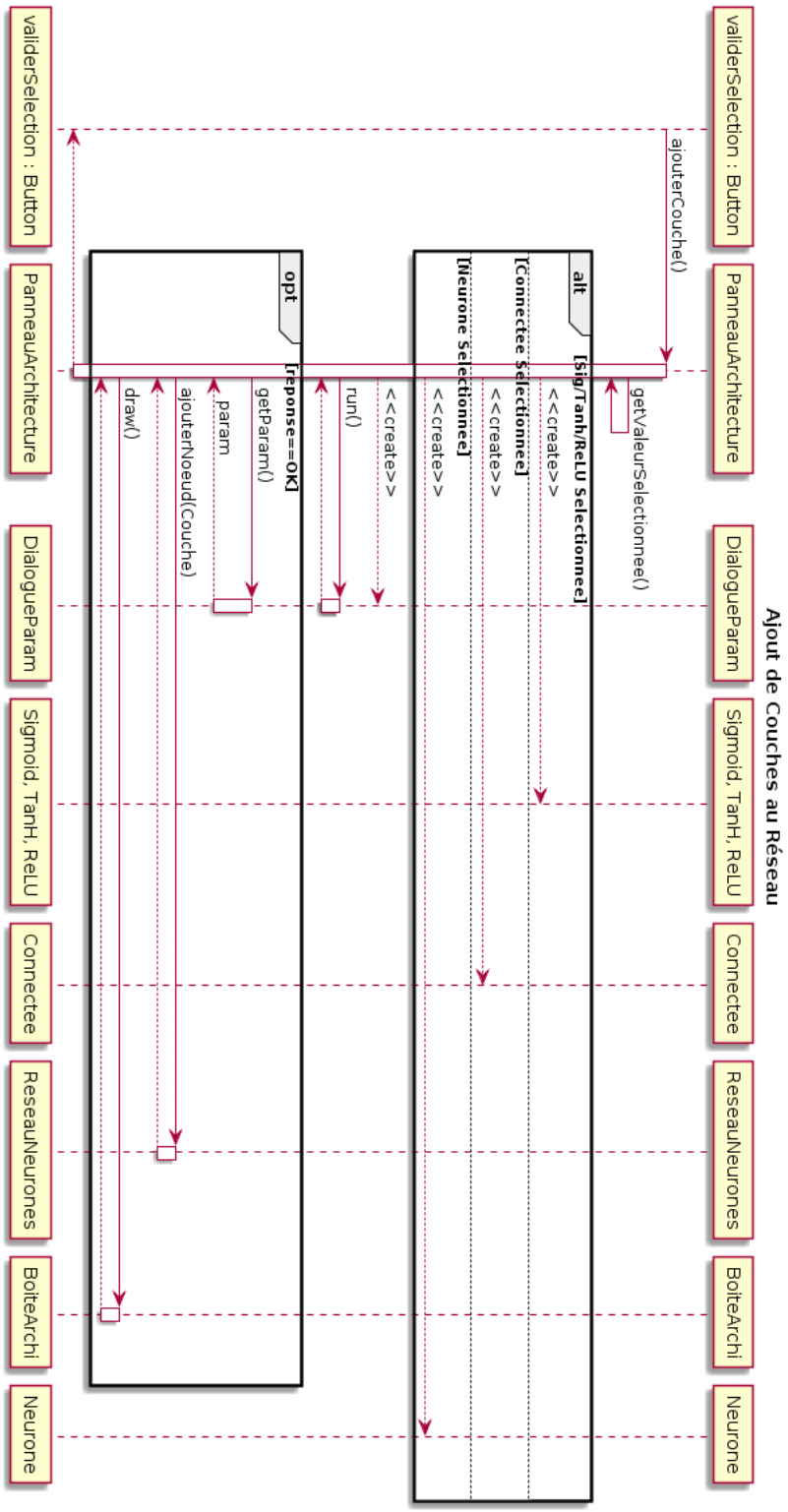
6.2 Tests unitaires

Les tests unitaires sont les mêmes que ceux que nous avions déjà implémentés. Ils sont disponibles avec notre code source.

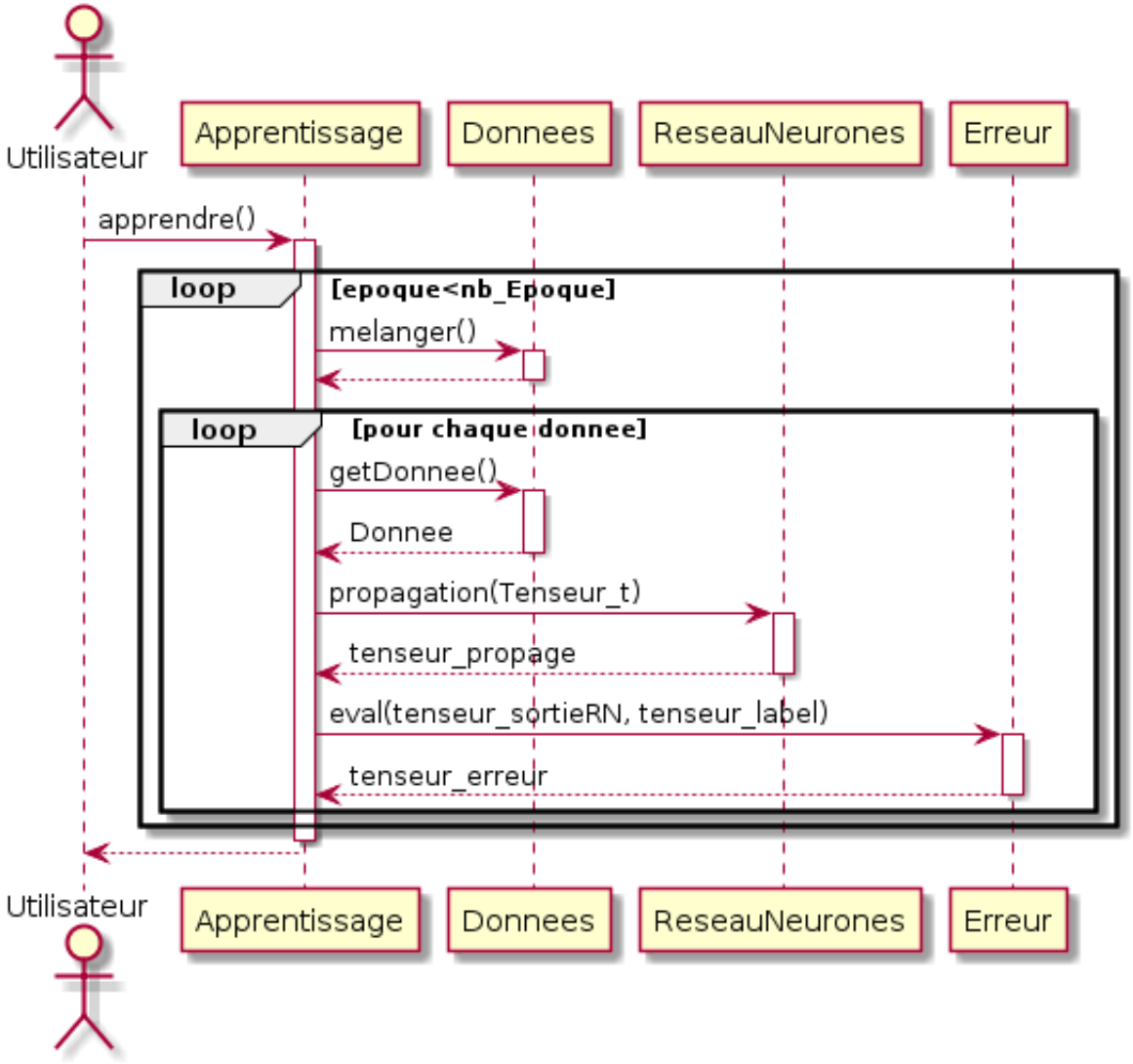
6.3 Tests d'intégration

6.3.1 Scénario pour la création




Ajout de Couches au Réseau
validerSelection : Button
PanneauArchitecture
DialogueParam
Sigmoid, TanH, ReLU
Connectee
ReseauNeurones
BoiteArchi
Neurone
ajouterCouche()
getValeurSelectionnee()
alt
[Sig/Tanh/ReLU Selectionnee]
<<create>>
[Connectee Selectionnee]
<<create>>
[Neurone Selectionnee]
<<create>>
opt
[reponse==OK]
getParam()
param
run()
ajouterNoeud(Couche)
draw()

6.3.2 Scénario pour l'apprentissage

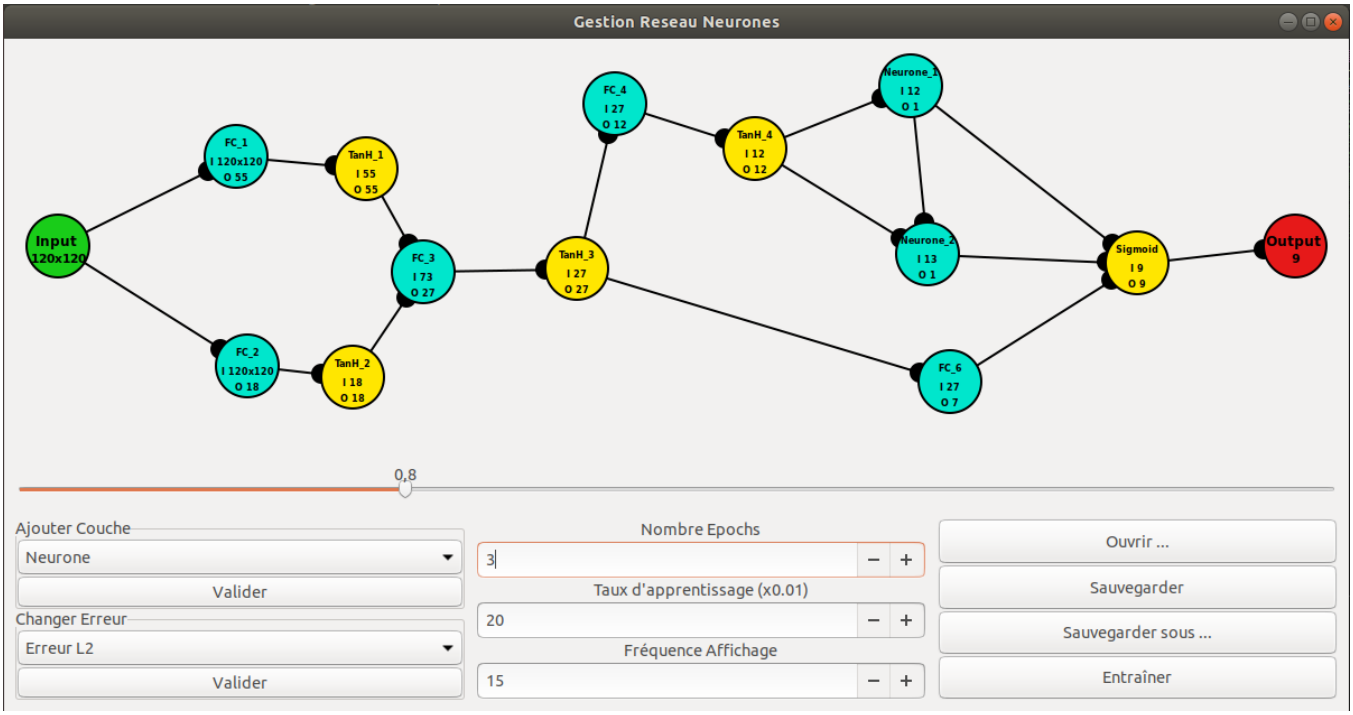


Partie 7

Manuel d'utilisation

7.1 Utilisation depuis l'interface graphique

Le logiciel se présente sous la forme d'une interface graphique divisée en deux parties. Une première partie, en haut, permet d'afficher le réseau de neurones actuel et de faire des modifications sur sa structure. Une seconde partie, en bas, permet le paramétrage de l'apprentissage.



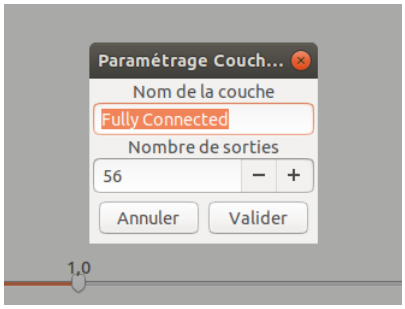
7.1.1 Affichage du réseau

Pour modifier le réseau de neurones, les 4 boutons suivants sont nécessaires :

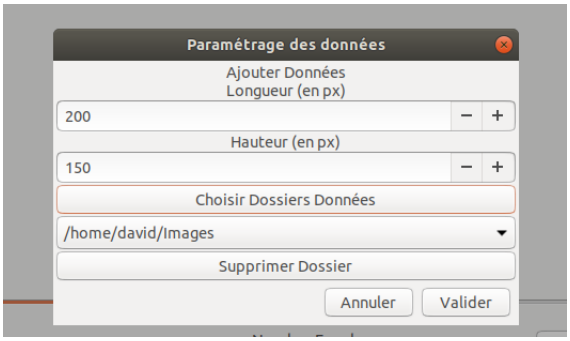
- Simple Clic Gauche : Permet de sélectionner une couche.

- Simple Clic Droit : Dans le cas où une couche est sélectionnée, permet de déplacer la couche sélectionnée à l'endroit souhaité si la zone pointée est vide. Sinon on va créer un arc entre la couche sélectionnée et la couche pointée.

- Double Clic Gauche : Permet de modifier les paramètres de la couche (nom, nombre de sorties pour une couche connectée par exemple).



Un double clic gauche sur la couche Input permet de choisir les données.



- Double Clic Droit : Permet de supprimer une couche.

7.1.2 Paramétrage du réseau

Dans la partie du bas, on a différents choix cliquables pour la création du réseau et la gestion des paramètres d'apprentissage :

— **Ajouter Couche** qui permet d'ajouter une couche avec les paramètres souhaités
— **Changer Erreur** qui permet de choisir le type d'erreur (L1, L2, entropie croisée) désiré
— **Nombre Epoques**, **Taux d'apprentissage** et **Frequence Affichage** qui permettent d'ajuster ces paramètres

Puis enfin, il y a quatre boutons permettant de faire des sauvegardes, d'ouvrir un réseau de neurones et surtout de pouvoir l'entraîner.
Pour l'entraîner, la seule de ces fonctionnalités déjà implémentée, il suffit de cliquer sur **Entrainer**.

7.2 Utilisation en tant que développeur

Nous avons vu l'utilisation en tant que concepteur de réseaux de neurones en logiciel graphique. Nous pouvons cependant imaginer que nos librairies pourraient être utilisées par d'autres développeurs. Ceux ci pourraient ainsi réutiliser le travail déjà fait et également créer de nouvelles couches pour la création de réseaux de neurones plus sophistiqués ou plus spécialisés.

L'utilisation du module tenseur permet d'instancier, d'initialiser et faire des opérations sur les tenseurs. A titre d'exemple, nous pouvons : appliquer des fonctions, additionner, soustraire, concaténer ou encore multiplier des tenseurs entre eux.

```
Tenseur t1({5,3}), t2({5,5}), tPlus = {3,5}, tMoins = tPlus, tRes = {3,5};
t1 = 5.1;
t2 = 3.5;
tPlus = 40.05;
tMoins = 29.3;
tRes = 100;

Tenseur tm;

tm = t1 * t2 + tPlus - tMoins;

cout << "Resultat : " << tm << endl;
```

L'utilisateur développeur pourra également créer un réseau de neurones simplement et faire la propagation dessus. Evidemment, pour utiliser son réseau de neurones dans la vie de tous les jours, il devra auparavant l'avoir entrainer. Comme le réseau créé n'est pas entrainé, l'exemple ci-dessous donnera un résultat de sortie aléatoire.

```
// -------------------------------------------------
// EXEMPLE CREATION RESEAU DE NEURONE ET PROPAGATION

Tenseur *tout = new Tenseur({5}), *res = new Tenseur({5, 5});
res->initValeurConstant(5.5);
tout->initValeurConstant(3.);

ReseauNeurones *r2 = new ReseauNeurones("MonRN");
r2->setDimInput(tout->getDim());

Couche *c1 = new CoucheConnectee(10, "FC_1");
Couche *c2 = new CoucheConnectee(10, "FC_2");
Couche *c3 = new CoucheConnectee(10, "FC_3");
Couche *c4 = new Sigmoid("Sigmoid2");
Couche *c5 = new TanH("Tanh3");
Couche *c6 = new Sigmoid("Sigmoid3");

r2->ajouterNoeud(c1);
r2->ajouterNoeud(c2);
r2->ajouterNoeud(c3);
r2->ajouterNoeud(c4);
r2->ajouterNoeud(c5);
r2->ajouterNoeud(c6);

r2->ajouterCoucheInitiale(c1);
r2->ajouterCoucheInitiale(c2);
r2->ajouterCoucheInitiale(c3);
r2->ajouterCoucheFinale(c5);
r2->ajouterCoucheFinale(c6);

r2->ajouterArc(c1, c5);
r2->ajouterArc(c1, c4);
r2->ajouterArc(c2, c4);
r2->ajouterArc(c3, c4);
r2->ajouterArc(c4, c5);
r2->ajouterArc(c4, c6);

*res = r2->propagation(*tout);

cout << "RESULTAT FINAL !!! \n\n" << *res << endl;
```

Partie 8

Conclusion et perspectives

Tout au long de ce projet, nous avons pu mettre en pratique nos connaissances en C++ et surtout découvrir les nombreuses possibilités (et ennuis) offerts par ce langage. Nous avons choisi un projet ambitieux, qui nous a permis de découvir la collaboration en grand groupe car nous n'avions jamais travaillé à 6. Nous avons également réutilisé de nombreux points vus en cours qui nous ont permis de largement progresser en utilisant des techniques de programmation plus complexes que ce à quoi nous étions habitués. Pour la majorité d'entre nous, nous avons également découvert le travail collaboratif à l'aide de la plateforme GitLab, qui a permis à chacun d'évoluer en parallèle des autres.

Devant l'envergure des tâches que nous souhaitions accomplir, nous n'avons malheureusement pas pu tout faire, et nous avons été contraints de rendre un projet qui n'est pas terminé. Nous sommes parfois restés bloqués sur des méthodes qui ne fonctionnaient pas et qui nous empêchaient d'avancer, comme celles liées à l'apprentissage (la partie concernant la rétropropagation) et celles liées au chargement et au prétraitement des données (erreurs changeantes d'un essai à l'autre). Même en se répartissant les tâches, il était alors difficile voire impossible de progresser plus rapidement.

Pour rendre le projet réellement utile et abouti, il semble essentiel de parvenir à sauvegarder un réseau de neurones dans un fichier de façon à pouvoir réutiliser une architecture entrainée dans un logiciel ou programme externe au notre. Il parait également très important de pouvoir l'entraîner et le réentrainer en changeant ou non les paramètres quand on le souhaite et non pas forcément après sa création. On pourrait également sauvegarder les paramètres des réseaux entrainés pour procéder à plusieurs entrainements successifs. Le développement d'une partie s'occupant de tester les performances du réseau de neurones sur un lot de données serait enfin l'aboutissement du projet. Cela nous permettrait de présenter une première version d'un logiciel de gestion de réseau de neurones. On pourrait également imaginer ajouter plus de choix pour les couches et les erreurs.

Pour finir, ceci reste une version miniature de l'ampleur que pourrait avoir un tel logiciel. De nombreuses choses pourraient être ajoutées pour aboutir à un logiciel réellement performant et utile. En effet, le très grande nombre de possibilités pour créer des réseaux de neurones et le nombre croissant de façons de les entraîner pourrait aboutir à un logiciel très complexe.