PROJET AU FIL DE L'ANNÉE

SPÉCIFICATIONS LOGICIELLES

Plateforme d'analyse de réseaux de neurones récurrents

21 novembre 2018

Jofrey Gaurivaud
Alexandre Heuillet
Julien Buré
Rémi Garcia
Jean-Louis Perche
Léo Tomas
Azeddine Harchaoui

Table des matières

Chapitre 1

Introduction

1.1 Objectif général

Ce présent document recense la totalité des spécifications et analyses permettant la conception d'une plateforme d'étude de réseaux de neurones récurrents commandée par les Drs. Frédéric Alexandre et Xavier Hinaut de l'Institut National de Recherche en Informatique et Automatique (INRIA).

Ce projet a pour but de réaliser un logiciel proposant de créer des réseaux neuronaux récurrents sur le modèle du Reservoir Computing, ainsi que d'en observer le comportement, via l'affichage de métriques. Il sera possible d'en modifier les caractéristiques de manière dynamique, le tout de manière aisée à l'aide d'une interface intuitive.

Il autorisera ainsi les chercheurs à effectuer des observations dynamiques pendant l'exécution des calculs par le réseau. En effet, ils n'effectuaient jusqu'alors des relevés que de manière statique, c'est-à-dire une fois l'exécution terminée.

1.2 Motivation

Nous sommes une équipe d'élèves-ingénieurs très intéressés par tout ce qui touche à l'intelligence artificielle, et particulièrement les réseaux de neurones, ainsi qu'au domaine du biomédical.

Ainsi, ce sujet était pour nous l'occasion de découvrir des technologies émergeantes et prometteuses telles que les réseaux de neurones récurrents et le modèle de Reservoir Computing tout en mettant en applications nos connaissances théoriques, à la fois scientifiques et managériales, acquises à l'ENSEIRB-MATMECA.

Chapitre 2

Exigences

2.1 Besoins fonctionnels

L'objectif de ce projet est de définir et de réaliser une plate-forme logicielle sous la forme d'une application web facilitant l'analyse expérimentale de certaines propriétés de réseaux de neurones, afin d'assister deux chercheurs de l'équipe Mnemosyne LaBRI-Inria spécialisée dans la modélisation de fonctions cérébrales par réseaux de neurones dans leur travail. Nous nous intéresserons plus spécifiquement ici aux réseaux de neurones récurrents sur le modèle du Reservoir Computing.

L'application devra alors permettre de générer des réseaux de neurones, à partir de caractéristiques souhaitées et données par l'utilisateur, de sorte à pouvoir traiter plusieurs types de tâches. Ensuite, l'application pourra simuler l'apprentissage de ces tâches par le réseau de neurones et la phase de test à l'aide de corpus de données ajoutées par l'utilisateur. Après vérification de la conformité du fichier de données (qui peut contenir des associations d'entrées et de sorties attendues pour la phase d'apprentissage ou bien seulement des entrées pour la phase de test), l'utilisateur pourra observer l'évolution de l'état du réseau. Toutes les requêtes seront données à travers une interface simple et ergonomique. Afin d'assurer un cloisonnement des données de chaque utilisateur, un système de login sera mis en place.

2.1.1 Définition du réseau

La génération du réseau de type réservoir doit pouvoir s'adapter à plusieurs types de structures. Ainsi l'utilisateur doit pouvoir configurer le masque de connexion donnant la matrice des connexions récurrentes du réservoir. La matrice est générée aléatoirement mais on veut pouvoir garder un certain contrôle en choisissant la densité des valeurs non nulles par exemple, ou en les regroupant en blocs dans la matrice pour faire apparaître des clusters. De plus, de nombreux paramètres pourront être spécifiés. En voici une liste qui peut être amenée à s'élargir au cours du projet :

— Présence ou non d'une connexion feedback depuis la couche de sortie vers le réservoir

— Fonction de transfert utilisée afin d'analyser l'impact sur le taux d'erreur
— Entraînement offline, online, supervisé ou non supervisé
— Rayon spectral de la matrice des connexions récurrentes

Dans un premier temps, la propagation des activations se fera en mode synchrone, c'est-à-dire que toutes les unités sont mises à jour simultanément. Si le temps le permet, un mode asynchrone pourra être envisagé (cf. le chapitre 2).

2.1.2 Tâches

Les réseaux générés doivent être capables de traiter différentes tâches. On pourra étudier la mémoire de travail avec les n-back task et delayed match-to-sample task ainsi que leur combinaison et généralisation. On voudra aussi assigner une tâche de prédiction à partir de séquences numériques ou symboliques pour être capable d'estimer l'évolution future d'une nouvelle séquence.

2.1.3 Observations

Une fonctionnalité importante de l'application sera la possibilité d'observer l'état du réseau à la fin de l'apprentissage à partir du corpus de données, mais surtout au cours de cet apprentissage. Ainsi, une fois le réseau créé et le corpus de données fourni, il sera possible d'étudier l'évolution temporelle et spatiale des paramètres en détail grâce à l'affichage de courbes et de représentation en niveaux de gris. Une panoplie d'outils de lecture sera à disposition de l'utilisateur pour simplifier sa tâche comme des fonctions *play*, *pause*, avance d'un certain nombre des pas, jusqu'à un certain événement, *replay* ainsi qu'un système de sauvegarde de l'état courant du réseau pour permettre de créer une "collection" de réseaux que l'on pourra choisir d'entraîner ou d'observer.

2.1.4 Potentielles fonctionnalités supplémentaires

Il pourrait être possible d'ajouter d'autres fonctionnalités dans la plate-forme, néanmoins, par manque de temps ou de par la complexité de la tâche, il n'est pas prévu d'implémenter les fonctionnalités suivantes, bien qu'elles aient leurs intérêts. Il sera néanmoins tenu compte dans la conception du code qu'elles devront pouvoir être implémentées sans refonte importante du code :

— La façon de calculer le prochain état du réseau est pour le moment effectuée en synchrone, par produit matriciel classique. Il pourrait être souhaitable de réaliser ce calcul d'une manière différente, par exemple en appliquant les produits coordonnée par coordonnée dans un certain ordre, en asynchrone.
— Le réservoir du réseau de neurones sera représenté par une matrice dans notre modèle. On pourrait envisager de le représenter par plusieurs matrices. Le calcul du prochain état serait alors impacté, mais aussi la visualisation potentielle du réservoir s'en trouverait modifiée.

2.2 BESOINS NON FONCTIONNELS

Au delà des services rendus par l'application, un certain nombre de besoins auxiliaires sont à satisfaire. Premièrement, le choix des technologies : dans le souci de proposer une application multi-plateforme et au déploiement le plus simple possible, l'association d'un framework web, ici Django, et du langage Python est toute indiquée compte tenu de la facilité d'interfaçage des scripts (voir le chapitre 3).

L'application devra néanmoins consommer des ressources en temps raisonnables, malgré la quantité de données à traiter.

En outre, l'application a vocation à être utilisée au clavier et à la souris. L'interface graphique en est donc une composante majeure.

L'utilisateur final devra pouvoir entrer les caractéristiques du réseau via des interactions simples.

Il devra pouvoir ajouter des corpus dans différents formats, en les précisant à la plateforme. L'application devra vérifier qu'un corpus est correctement formaté avant de l'accepter.

L'utilisateur devra aussi pouvoir voir l'ensemble des actions effectuées (entraînement et test) sur le réseau étudié. De plus, il devra avoir un affichage clair des actions effectuées quand le serveur fait tourner le réseau.

L'interface sera fournie en anglais, les noms des menus seront donc en anglais eux aussi.

L'utilisateur final devra pouvoir entrer les caractéristiques du réseau via des interactions simples.

2.2.1 Ajout de paramètres

Ce programme est à destination de chercheurs, il est par nature impossible de prévoir tous les besoins d'expérimentations qu'ils auront dans le futur. Néanmoins la plate-forme d'analyse devra toujours être utilisable pour tester des cas de figure non pris en compte à la base, et ce avec le moins de modifications possibles à apporter au code :

— Il faudra assurer que des paramètres de génération, ou même des méthodes de génération, pourront être ajoutés facilement, sans modifier le code, ou seulement très peu.
— De plus, il faudra aussi pouvoir ajouter de nouvelles fonctions de modification des réseaux qui s'appliqueront durant des entraînements offline.

— Enfin, il en va de même pour les observables des réseaux, le programme doit pouvoir prendre en compte l'ajout de nouveaux observables aussi simplement pour l'utilisateur que possible.

La meilleure solution envisageable pour répondre à ces problématiques serait d'avoir des fichiers éditables de déclaration des paramètres (en JavaScript Object Notation par exemple), des méthodes de génération, des observables, et des fonctions d'édition. Il s'agit ensuite de pouvoir ajouter leur code associé dans des fichiers de code traitant de la génération des réseaux, du calcul des observables, et des modifications du réseau. Le logiciel devrait ensuite charger les fichiers de déclaration et proposer toutes les options qui y figurent, puis se référer au code fourni pour lancer les calculs.

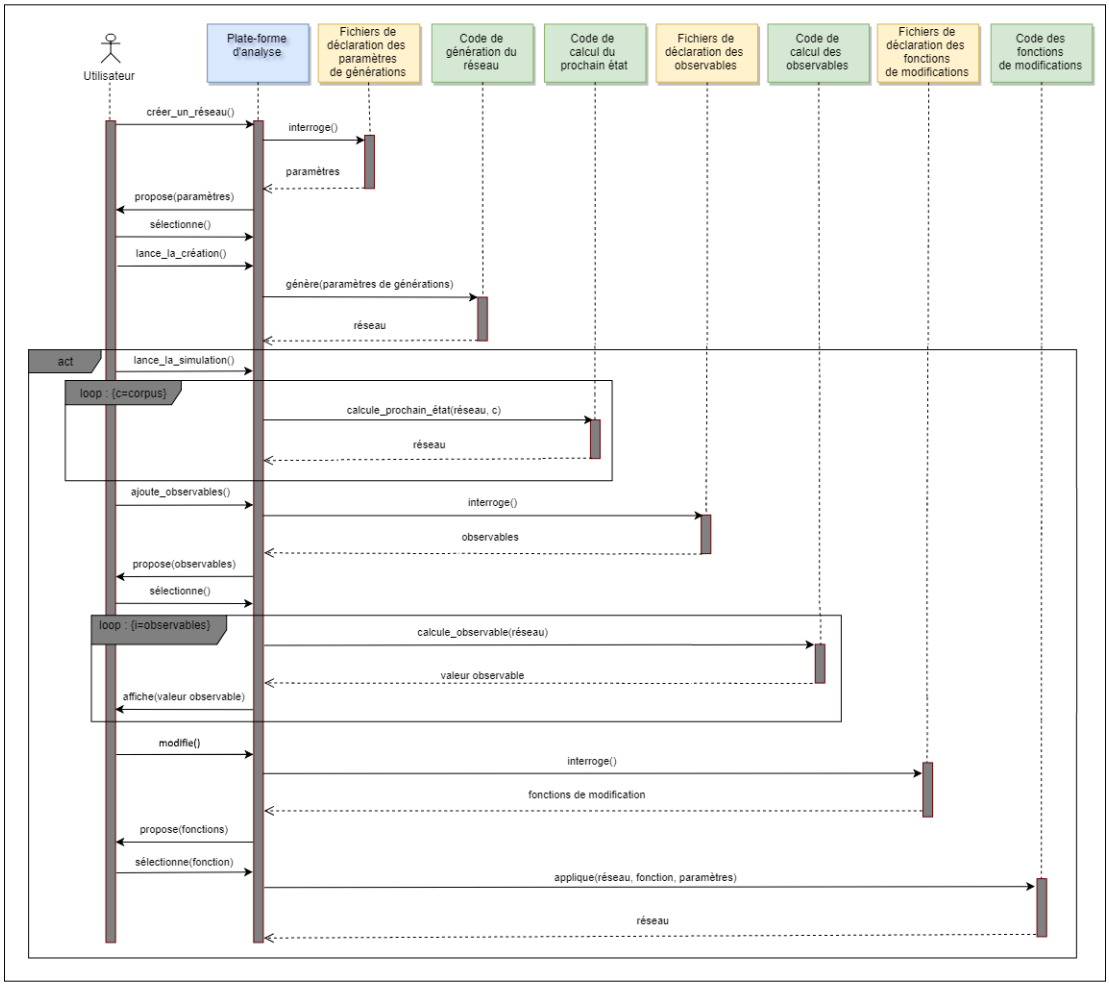

FIGURE 2.1 – Diagramme de séquence illustrant l'évolutivité du code

La figure suivante illustre d'avantage ce découpage du code. On notera ici que les entités sont purement conceptuelles. Dans la réalisation finale, les entités "code de génération du réseau" et "code de calcul du prochain état" seront situées dans le même fichier Python.

2.2.2 L'interface graphique de la plate-forme

Ici, le lancement des interactions entre neurones doit se faire par le clic sur un seul bouton, typiquement un bouton "Run". Des menus seront disponibles pour les options de lancement concernant le nombre de pas à effectuer. Un bouton "Pause" sera aussi intégré pour dégager les caractéristiques du réseau à un instant t.

Dans la même optique d'observation statique, des menus serviront à l'importation/exportation de l'état courant du réseau.

Voici l'interface telle qu'elle se présentera à l'utilisateur (avec les menus traduits en français) :

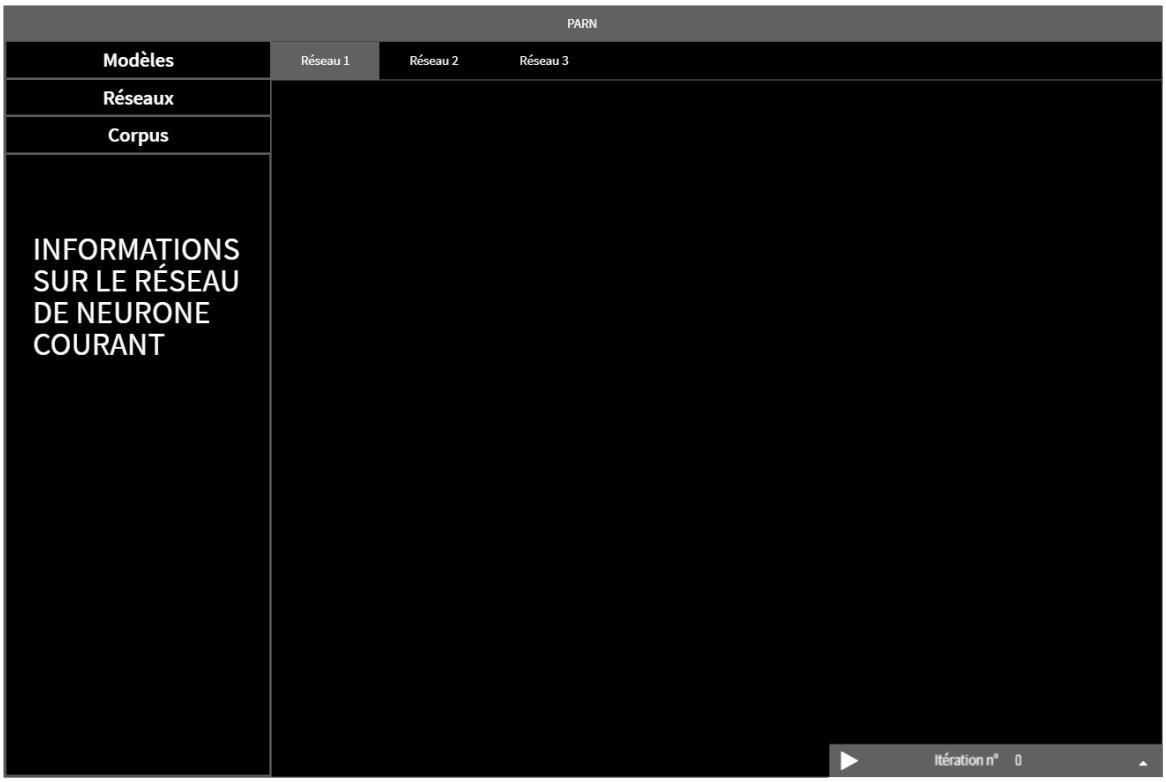

FIGURE 2.2 – Interface principale de la plate-forme

Pour chaque réseau chargé par le programme, un nouvel onglet apparaît avec le nom du réseau dessus. Un clic gauche utilisateur permet de passer d'un onglet à l'autre et donc d'un réseau chargé par le programme à un autre. Le réseau dernièrement sélectionné est appelé courant.

Les informations sur le réseau courant comprennent ses paramètres de génération, son nom, les derniers entraînements qu'il a reçus, sa date de création, et sa configuration actuelle.

Les menus Modèles, Réseaux et Corpus sont des menus déroulants qui permettent de gérer les différentes données sauvegardées sur le serveur via différentes actions.

Le menu Modèle va permettre de gérer les fichiers de configurations des réseaux. Ces configurations de réseaux sont des pré-sets de génération des réseaux de neurones qui permettent de générer directement un nouveau réseau de neurones sans rentrer tous les paramètres à la main.

Voici les options correspondant à ce menu déroulant :

FIGURE 2.3 – Menu déroulant des Modèles

Ici, tous les boutons ouvrent des pop-ups permettant d'effectuer les actions décrites par le nom de la case.

Pour l'option "Créer", on ouvre directement un éditeur permettant de rentrer les paramètres désirés. Celui-ci propose tous les paramètres possibles avec leurs valeurs par défauts, et laisse l'utilisateur les rentrer dans le cadre de leurs valeurs possibles. Une fois terminé, on peut entrer le nom de notre modèle et l'enregistrer sur le serveur.

Les boutons "Supprimer", "Copier", "Renommer", "Modifier" et "Télécharger" ouvriront la même fenêtre, un explorateur de fichier simplifié permettant de rentrer le nom d'un modèle et de le sélectionner. Une fois ceci fait, l'action indiquée sur le bouton s'applique sur le fichier sélectionné.

Voici cette fenêtre :

Dans le cas de la suppression, une confirmation sera demandée avant d'effacer le fichier.

Dans le cas du renommage, un nouveau nom sera demandé à l'utilisateur dans une nouvelle fenêtre, puis après confirmation, le fichier sera renommé.

Dans le cas de la modification, après avoir sélectionné le fichier, celui-ci s'ouvre dans la même fenêtre que pour la création de modèle, mais au lieu des paramètres par défaut, ce sont les paramètres du fichier sauvegardé qui sont sélectionnés.

Pour ce qui est du téléchargement, la sélection du fichier suffit à le lancer.

Enfin dans le cas de la copie, le nom de la copie du fichier sera le nom du fichier suivi de

FIGURE 2.4 – Fenêtre de sélection

"-copie".

Le menu Corpus va permettre de gérer les fichiers d'exemples d'entraînement pour les réseaux. Ceux-ci devront être générés par l'utilisateur et uploadés vers le serveur pour être utilisés.
Voici les options correspondantes à ce menu déroulant :

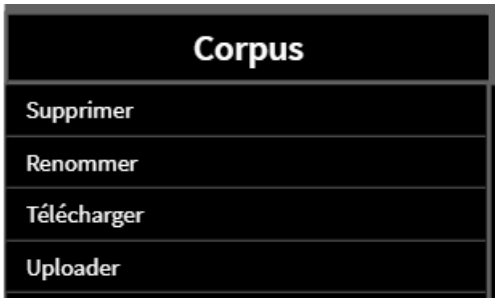

FIGURE 2.5 – Menu déroulant des Corpus

Ici, tous les boutons présents auront le même comportement que ceux du menu "Modèles".

Le menu "Réseaux" va permettre de gérer les réseaux. Il doit permettre tous les traitements d'archivage, de génération, mais aussi de chargement pour avoir accès à l'analyse des réseaux.

Voici les options correspondantes à ce menu déroulant :

FIGURE 2.6 – Menu déroulant des Réseaux

Les boutons "Supprimer", "Copier", "Renommer", "Télécharger", et "Uploader" fonctionnent tous sous le même principe que leurs équivalents dans le menu "Modèle".
Le bouton "Créer" va ouvrir une première pop-up demandant à l'utilisateur s'il veut charger un modèle pré-existant ou non.

S'il répond oui, il sera amené sur le même écran de sélection des modèles que, par exemple, lorsque on cherche à supprimer un modèle. L'utilisateur peut ensuite sélectionner le modèle. La génération est alors lancée, et le réseau est chargé dans un nouvel onglet.

S'il répond non, l'onglet de création de modèle s'ouvre et le laisse entrer tous les paramètres qu'il souhaite. Une fois ceci fait, il dispose de deux boutons, "Sauvegarder le modèle" et "Générer le réseau". Le premier se comporte comme décrit plus haut, mis à part qu'il ne ferme pas toutes les pop-ups, puisque l'utilisateur doit pouvoir cliquer sur "Générer le réseau". Le bouton "Générer le réseau" a le même comportement que décrit au dessus.
Le bouton "Charger" permet d'ouvrir l'écran de sélection listant les réseaux déjà présents sur le serveur. De là, on peut sélectionner un réseau et le charger. Il s'ouvre alors dans un nouvel onglet.
Le bouton "Sauver" permet de sauvegarder le réseau courant sur le serveur. Il ouvre la même pop-up que pour l'enregistrement d'un modèle lors du nommage de celui-ci. Après confirmation, l'état du réseau courant est enregistré sur le serveur.

Le menu en bas à droite remonte sur l'écran lorsque l'utilisateur clique sur l'icône de flèche vers le haut. Il ressemble alors à cela :

FIGURE 2.7 – Menu de manipulation du réseau courant

La flèche de droite permet de réduire à nouveau le menu.
Ici le triangle blanc est un bouton start, il permet de lancer les calculs avec les paramètres déclarés en dessous.

Juste à sa droite se trouve un compteur du nombre d'itérations menées avec ces paramètres.

Il est possible de définir un nombre d'itérations après lesquelles l'apprentissage va s'arrêter. Cela se fait en entrant une valeur dans la case "Arrêter après ... itérations". Par défaut, celle ci est vide. Si cette case est vide, l'itération ne s'arrête que si l'utilisateur stoppe manuellement le processus ou que le corpus est arrivé à sa fin.

Lorsque l'apprentissage est stoppé, le bouton "Start" est remplacé par d'autres boutons, on obtient alors :

FIGURE 2.8 – En-tête du menu de manipulation des réseaux

Les deux boutons supplémentaires servent assez intuitivement à parcourir les états enregistrés du réseau, en respectant le pas défini dans "faire des bons de".

Le menu "Séquence programmées" est un menu déroulant qui permet de configurer l'ordre dans lequel les séquences apparaissent, et de les supprimer. Le voici :

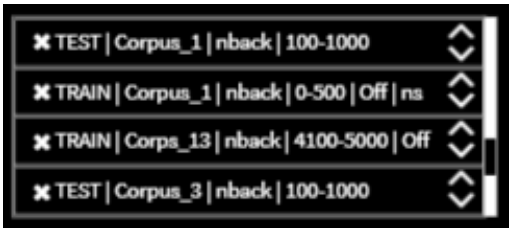

FIGURE 2.9 – Menu déroulant des séquences a effectuer

Les séquences sont affichées dans l'ordre dans lequel elles s'exécutent. Les flèches à droite permettent de monter ou descendre les séquences dans la liste et donc de changer l'ordre d'exécution de celle-ci. La croix permet quant à elle de supprimer une séquence.

Le bouton "Ajouter une séquence" permet d'ouvrir un menu de configuration qui permet d'ajouter une séquence supplémentaire. Le voici :

FIGURE 2.10 – Menu de configuration des séquences

Les options sur le type d'itération "test" et "entraînement" permettent de configurer si l'on veut que le réseau apprenne en supervisé (*teacher-forcing*) sur son read-out ou non.

Le bouton "Type d'entraînement" permet simplement de basculer entre Entraînement offline et online.

L'option "non supervisé" sert à configurer si le réseau va subir un entraînement non-supervisé sur son réservoir ou non.

L'icône à droite de la boite de corpus permet d'ouvrir dans une nouvelle fenêtre la sélection de fichier pour ainsi pouvoir sélectionner un corpus enregistré sur le serveur. Si le corpus ne correspond pas en taille à l'entrée du réseau, le programme refuse la saisie.

Enfin les 2 entrées numériques restantes permettent de configurer quel partie du fichier donnée en entrée on va lire.

Une fois la configuration terminée, la séquence est ajoutée dans la liste et sera visible dans le bandeau "dernière séquence" de la fenêtre principale lorsque le réseau sera exécuté dessus.

Le menu déroulant "Appliquer une modification" va afficher toutes les fonctions de modifications définies dans le programme. L'utilisateur peut cliquer sur l'une d'entre elles. Cela a pour effet d'ouvrir une pop-up qui va (si nécessaire) demander les paramètres de la fonction à l'utilisateur. Une fois tous les paramètres renseignés, l'utilisateur peut confirmer à l'application la modification, et l'opération est effectuée sur le réseau courant.

Enfin, le menu déroulant "Ajouter une observation" permet de sélectionner les observable du réseau à afficher. Il va donc lister tous les observables disponibles. Lorsque l'utilisateur clique sur l'un d'entre eux une fenêtre apparaît pour lui demander de renseigner toutes les caractéristiques d'affichage et les paramètres de calcul. Pour chaque observable calculé, un nouveau widget éponyme apparaît dans l'onglet. En voici un dans une vision globale du programme :

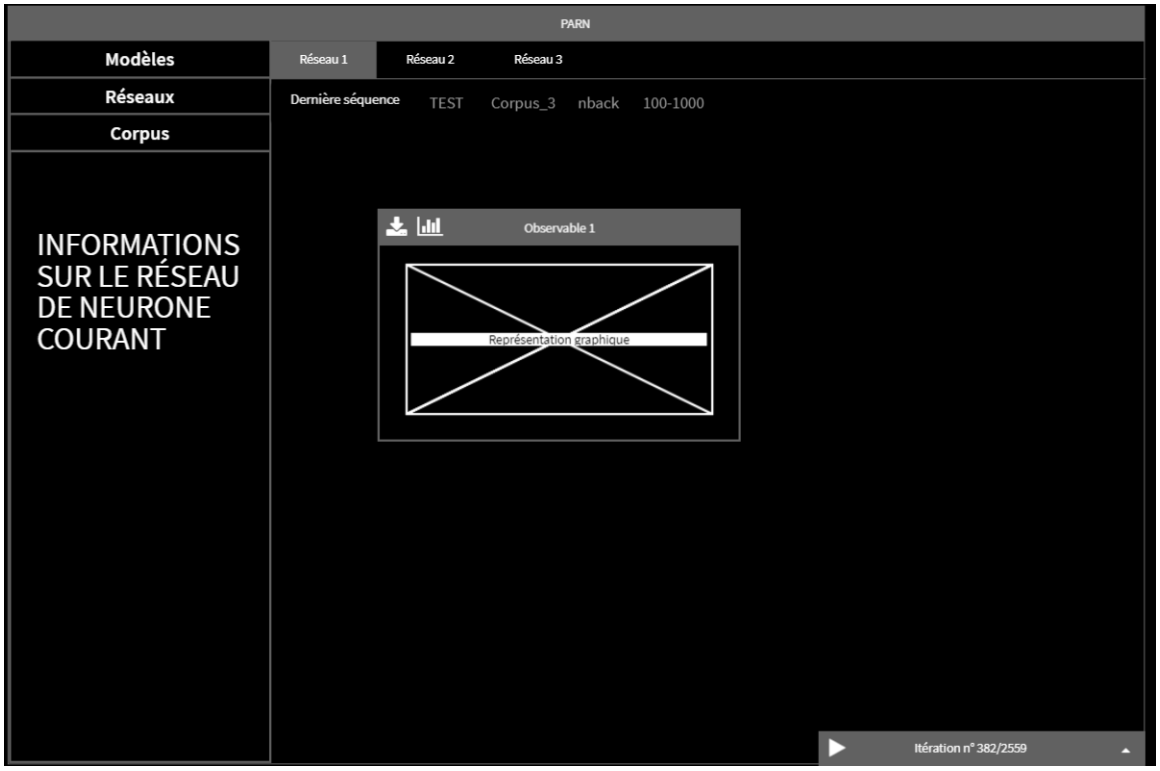

FIGURE 2.11 – Vision globale du programme avec un observable affiché

Ces widgets peuvent être redimensionnés en maintenant le bouton gauche de la souris et déplacés dans la fenêtre via un glisser-déposer.

Le bouton à l'extrême gauche de l'en-tête est cliquable : il a pour effet de télécharger directement les informations de l'observable actuellement disponibles sur la machine de l'utilisateur au format CSV.

L'affichage des observables se fait dans widgets sous le format que l'utilisateur a choisi d'utiliser. Pour sélectionner un mode d'affichage, l'utilisateur clique sur le deuxième bouton de l'en-tête. Celui-ci fait apparaître un menu déroulant qui affiche les options suivantes : "Graphique", "Numérique" et "Colorimétrique". L'option sur laquelle l'utilisateur clique permet de déterminer quel affichage on utilise.

L'affichage "Graphique" trace la courbe des points avec pour ordonnée la valeur de l'observable et pour abscisse l'itération.

L'affichage "Numérique" affiche simplement la dernière valeur numérique de l'observable calculée.

Enfin l'affichage "Colorimétrique" convertit la valeur de l'observable en couleur et l'affiche selon les mêmes modalités que l'affichage "Numérique".

On notera que si l'observable est un vecteur, on superposera juste les affichages de chacune de ses coordonnées.

2.3 PLAN DE TESTS DE VALIDATION

Pour nous assurer du bon fonctionnement et de la qualité de notre application, un certain nombre de tests seront effectués. L'objectif des tests sera uniquement la correction du programme. Les paramètres de performance et d'optimisation ne seront pas pris en compte car ils n'intéressent pas le client. Les tests réalisés seront de trois types : d'abord les tests unitaires, puis tests d'intégration et enfin, les tests de validation.

2.3.1 Tests unitaires

Les tests unitaires consisteront à tester chaque fonction de chaque module de manière ciblée (tests boite blanche sur les instructions). Ils permettront de détecter les erreurs de base dans notre module. On commencera par écrire la liste des modules développés. Une fois un module terminé, nous écrirons les tests unitaires associés. A chaque modification d'un module, nous repasserons ces tests pour vérifier la non-régression du code.

2.3.2 Tests d'intégration

Il faudra ensuite s'assurer de la compatibilité des modules, autrement dit on teste les interfaces entre modules (tests boîte blanche sur les interfaces, tests boîte noire sur les instructions). Nous vérifierons notamment que les réseaux de neurones générés ont bien les propriétés saisies par l'utilisateur, que les graphiques s'affichent bien, et que l'interface est correctement réalisée.

2.3.3 Test de validation

Une fois le logiciel assemblé, nous le testerons sur des cas concrets d'utilisation. L'application sera lancée sur des scénarios d'apprentissage caractéristiques pour que l'on vérifie son fonctionnement global. Les exigences sus-mentionnées seront alors vérifiées une par une. Quand toutes seront validées, l'application pourra être livrée.

2.3.4 Documentation

Deux types de documentation seront rédigées :

— **Un manuel utilisateur :** Il décrira les différents cas d'utilisation du logiciel et détaillera sa prise en main. Il sera rédigé en LaTeX.

— **Un manuel de maintenance :** Il comportera la description technique de l'application en détaillant le fonctionnement de chacun de ses modules, de façon à permettre une reprise sans encombre du développement par une autre équipe en cas de besoin. Il sera rédigé à l'aide de l'outil de génération de documentation Python http ://www.sphinx-doc.org/en/master/index.html.

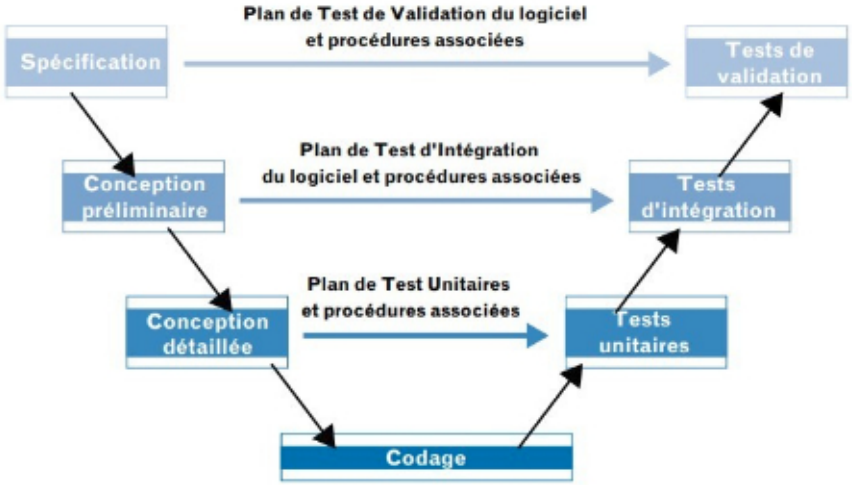

FIGURE 2.12 – Schéma des tests dans le cycle de « développement en V »

Chapitre 3

Architecture et conception logicielles

3.1 Architecture globale de l'application

L'architecture logicielle retenue est celle d'une application Web. Elle repose sur trois pierres angulaires :

- **Le navigateur Web :** Il affiche la *GUI* (General User Interface) à partir de code HTML envoyé par le serveur Web. Il interprète également les actions réalisées par l'utilisateur (clic sur un bouton par exemple) en requêtes HTTP qui seront traitées par le serveur Web.
- **Le serveur Web :** Il héberge les fichiers composant l'application et transmet le code HTML au navigateur. Il effectue également les calculs nécessaires au fonctionnement de l'application et au rendu graphique via le traitement de requêtes HTTP transmises par le navigateur.
- **La base de données :** Elle autorise la persistance des données de l'application en les stockant et les délivrant au serveur Web sur demande de ce dernier.

Le diagramme ci-dessous résume les différentes actions effectuées par les éléments de l'architecture Web lors d'une session :

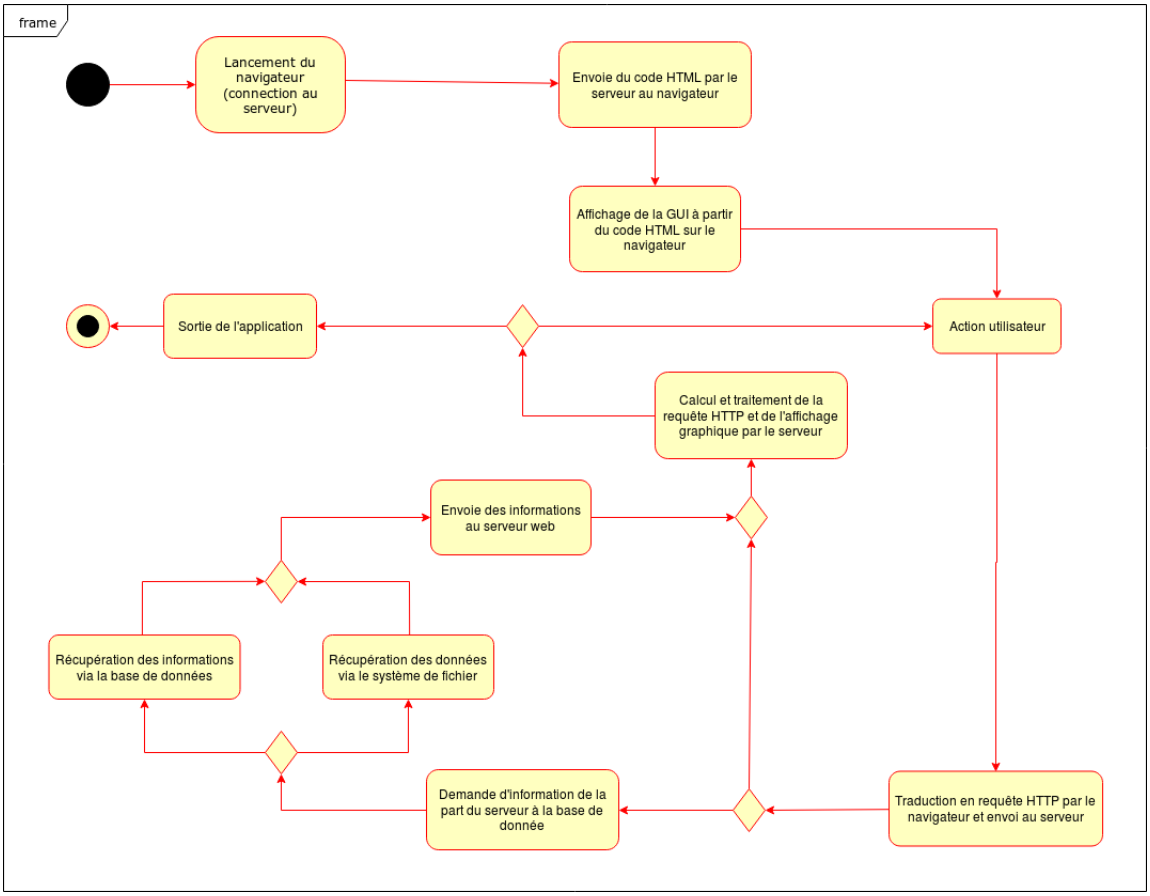

FIGURE 3.1 – Diagramme d'activité Web

En effet, cela permet de déployer le logiciel facilement, en le rendant accessible simultanément à un grand nombre de professeurs et d'élèves de part son installation sur un serveur Web public (de l'INRIA par exemple). De plus, on pourra toujours l'exécuter localement sur un ordinateur personnel, qui jouera donc à la fois le rôle de client et de serveur via la boucle locale, si le besoin s'en fait sentir.

3.2 FRAMEWORK

Le client dispose déjà d'un script Python permettant de construire un réseau de neurones récurrents suivant ses caractéristiques (détaillées dans le chapitre 2). Ainsi, il paraît judicieux de poursuivre le développement en langage Python dans le but de faciliter au mieux l'interfaçage avec l'existant. De cette façon s'est alors posée la question du framework à employer pour réaliser une interface moderne et correspondant à l'architecture Web choisie.

Finalement, le projet Django fut retenu de par sa facilité de prise en main ainsi que sa grande versatilité. En effet, il permet de rapidement créer de nouvelles vues à l'aide de modèles HTML et possède un serveur Web de développement intégré ce qui évite de passer

par des logiciels externes tels qu'Apache ou Ngnix (sauf pour la phase de déploiement bien évidemment).

En outre, il comporte de nombreux modules dont un *ORM* (Object Relation Mapper) qui permet de faciliter la communication avec la base de données et la gestion des relations entre les entités, ainsi qu'une gestion native des sessions utilisateurs. De plus, il permet d'utiliser la totalité des bibliothèques Python 3 existantes (telles que Numpy par exemple), ce qui permet de se soustraire à la réécriture de l'existant, et donc de perdre du temps tout en assurant un code plus facilement débogable et maintenable.

Concernant la base de données, nous avons opté pour MySQL de part sa très bonne compatibilité avec l'*ORM* de Django.

Enfin, on emploiera également Bootstrap, un framework HTML/CSS, en conjonction de Django afin d'améliorer l'aspect esthétique général de l'application.

3.3 CONCEPTION LOGICIELLE ET PARADIGME

Le framework employé, Django, implique l'adoption d'un paradigme objet. Pour rappel, la programmation orientée objet est une technique de programmation consistant à considérer des entités élémentaires virtuelles appelées *objets*, instances de modèles appelés *classes*, et reposant sur des principes de masquage des réalisations, de polymorphisme, d'héritage et d'encapsulation des objets les uns dans les autres.

De cette façon, l'application a subi le découpage en modules détaillé dans le diagramme associé.

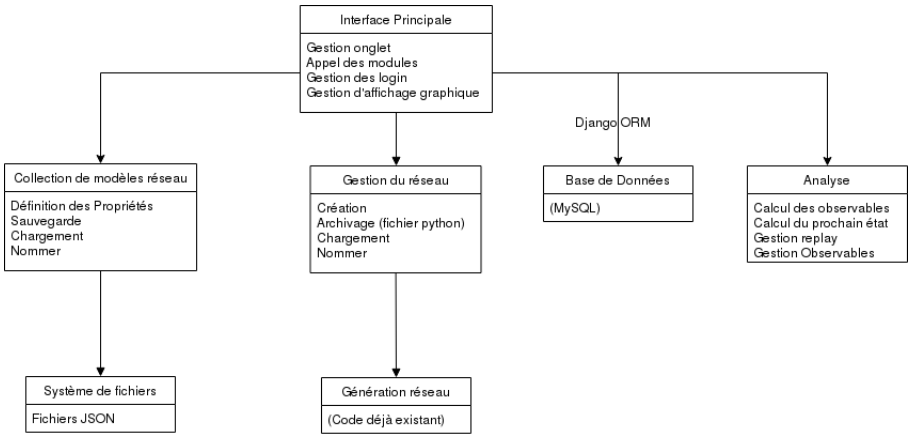

FIGURE 3.2 – Diagramme de l'architecture objet/modules

Il est important de noter que chacun des 4 modules décrits ci-dessus sera implémenté en utilisant un sous-découpage MVC (Modèle-Vue-Contrôleur) :

— **Le modèle :** Il permet de formaliser la représentation des entités (objets) employées par le module et est donc responsable de la gestion des données (sauvegarde et chargement vers ou depuis la base de données). Ainsi, le contrôleur peut connaître avec exactitude la nature de celles-ci (leurs attributs par exemple) lorsqu'il les manipule.
— **La vue :** Elle contient l'agencement de l'interface graphique et permet donc à l'utilisateur d'effectuer des actions. Ces dernières peuvent entraîner un accès ou une modification des données du modèle et sont traitées par le contrôleur.
— **Le contrôleur :** Il effectue les traitements nécessaires à la prise en compte des actions effectuées dans la vue et il peut ajouter, modifier ou supprimer les données du modèle.

Concernant le stockage des données relatives aux réseaux et à leurs états, on aura recours à des fichiers JSON (*Javascript Object Notation*) et Python (pour du stockage de matrices à l'aide de la bibliothèque Numpy). L'avantage principal du JSON est sa capacité, une fois parsé par Django, à définir des objets (donc aisément manipulables).
De fait, on note que le rôle de la base de données se limitera à la persistance des données de gestion de session utilisateur.

Chapitre 4

Répartition des tâches

4.1 Répartition des tâches

Nous sommes un groupe de 6 développeurs pour réaliser l'application. L'organisation de notre travail se fera sur le modèle du diagramme de Gantt suivant.

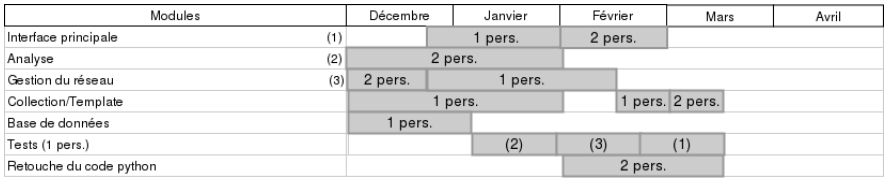

Figure 4.1 – Diagramme de Gantt de la réalisation du projet

Plusieurs remarques sont à faire. Tout d'abord, l'organisation des tests n'est qu'à titre indicatif car les tests de plusieurs modules peuvent être effectués en parallèle, elle ne sera pas forcément linéaire comme indiqué sur le diagramme (par exemple, les tests de la gestion du réseau peuvent être commencés avant que les tests du module analyse ne soient terminés). Tous les tests ne seront pas effectués par la même personne, nous nous organiserons de manière à faire tourner la répartition de cette tâche pendant la durée du projet. De plus, il est à noter que la personne affectée aux tests sera différente de la personne ayant implémenté les fonctions testées. De cette manière, nous garantissons l'objectivité et l'efficacité des tests fournis.
Le diagramme de Gantt a été réalisé afin qu'une personne terminant une tâche soit affectée à une nouvelle tâche commençante en priorité, de manière à minimiser les refontes de code déjà effectué par une autre personne. Cette réalisation permet une économie de temps et d'énergie afin d'assurer le meilleur rendu final que nous puissions fournir.
Enfin, la répartition des tâches est modulable et est susceptible d'être modifiée dans le temps en fonction de l'avancement du projet. Nous avons également laissé 3 semaines de délai avant la date prévue pour le rendu final afin de prévoir un éventuel retard.

Glossaire

Delayed match-to-sample task Un stimulus est présenté au sujet (souvent une couleur ou un symbole) et il va devoir le retenir. La tâche consiste, après un certain délai, a identifier parmi plusieurs stimuli lequel correspond à celui mémorisé précédemment. 4

Entraînement non supervisé Il s'agit de trouver des structures sous-jacentes à partir de données non étiquetées. Puisque les données ne sont pas étiquetées, il n'est pas possible d'affecter au résultat de l'algorithme utilisé un score d'adéquation. Cette absence d'étiquetage (ou d'annotation) est ce qui distingue les tâches d'apprentissage non-supervisé des tâches d'apprentissage supervisé. 4

Entraînement offline Entraînement effectué sur toutes les données d'un corpus d'apprentissage qui sont traitées simultanément. 4, 5, 13

Entraînement online Entraînement effectué sur des exemples présentés les uns après les autres au fur et à mesure de leur disponibilité. 4, 13

Entraînement supervisé Les exemples présentés durant l'entraînement sont fournis avec la réponse attendue. Le but des méthodes d'apprentissage supervisé est de bien généraliser, c'est-à-dire d'apprendre une fonction qui fasse des prédictions correctes sur des données non présentes dans l'ensemble d'apprentissage. 4

Feedback (ici) Information provenant de la sortie du réseau (*output*) et qui est réinjectée dans ce même réseau afin d'en améliorer les performances, notamment en permettant un recalcul des poids des connexions. 3

Fonction de transfert (ou de régularisation) Il s'agit d'une modification de la fonction d'erreur visant à éviter les problèmes de surajustement ou de sous-ajustement des données d'apprentissage, en construisant un facteur de pénalité en fonction de la complexité du réseau. 4

Framework Ensemble d'outils et de composants logiciels à la base d'un logiciel ou d'une application. Il établit les fondations d'un logiciel ou son squelette applicatif. 5, 18, 19

N-back task Une séquence de stimuli est présentée au sujet, et la tâche consiste à indiquer si le stimulus actuel correspond au stimulus présenté n itérations plus tôt. 4

Observable Toute donnée ou information du réseau directement accessible ou calculable. 6

Reservoir Computing Une technique de calcul offrant une architecture d'apprentissage supervisé pour un grand réseau de neurones récurrents à poids de connexions aléatoires mais fixés (le "réservoir"). L'idée principale étant d'induire une réponse non-linéaire de la part de chaque neurone du réseau à partir du signal d'entrée. 2