



DEUXIÈME ANNÉE-FILIÈRE INFORMATIQUE
Bordeaux INP - Enseirb-Matmeca

NEURON IT ! - PFA

Rapport de projet

AUTEURS :

Thomas Caroff, Remicia Di Franco
Xavier Dussieux, Thomas Saux
Adnane Khattabi, Ahcene Mahtout
Nathan Nguyen

RESPONSABLES DE PROJETS :

Responsable des PFA : Antoine Rollet
Professeure encadrante : Myriam Desainte-Catherine
Client (INRIA) : Xavier Hinaut

Table des matières

I Étude du projet	4
1 Contexte et problématique	4
2 Choix des outils et méthodes de développement	4
2.1 Environnements de développement	4
2.2 Outils de communication	4
2.3 Méthode agile : Scrum	5
3 Élaboration du document de spécification et application de la méthode agile	5
4 Objectifs du projet	5
5 Répartition du travail	6
II Réalisation et état courant du <i>backend</i>	7
6 Les réseaux de neurones	7
6.1 Neurones, propagation et rétro-propagation	7
6.2 Le perceptron simple	7
6.3 Le perceptron multicouches	8
6.4 Des réseaux récurrents : Elman et Jordan	9
6.5 Prise en main des réseaux de neurones par la création d'un auto-encodeur	10
6.6 Les réseaux opérateurs	10
7 Résolution d'un jeu de l'environnement OpenAi Gym à l'aide d'un réseau de neurones	12
7.1 CartPole-v0	12
7.2 Algorithme de résolution	13
8 Recherche des jeux de paramètres	14
8.1 Recherche manuelle	14
8.2 Optimisations avec HyperOpt	15
8.3 Deux formes d'apprentissage	15
8.4 Résultats	15
9 Fonctionnalités finales	19
9.1 Graphe de réseaux	19
9.1.1 Motivation pour la construction d'un graphe et propriétés du-dit graphe	19
9.1.2 Grille de jeu	19
9.2 Initialisation et propagation dans le graphe	20
9.3 L'apprentissage dans le graphe	21
9.4 Autres fonctionnalités	21
9.4.1 Création de logs	21
9.5 Sauvegarder et charger une partie	21
9.5.1 Gestion des erreurs	21
10 Tests	21
11 Phase d'intégration	22
12 Répartition du travail	22
III Réalisations du <i>frontend</i>	23

13 Les principales applications	23
13.1 Page d'accueil	23
13.2 "À propos"	24
13.3 "En savoir plus"	25
13.4 "Contactez-nous"	25
13.5 Classements	26
14 Gestion de la base de données - SQLite	27
14.1 Pourquoi SQLite?	27
14.2 Fonctionnement de la base de données	27
14.2.1 Description de la base de données	28
14.3 Limites de SQLite	28
15 L'interface administrateur	29
16 Le forum avec Spirit	30
17 Le jeu / Tutoriel en javascript	31
17.1 Paramètres	31
17.2 Le jeu	31
17.2.1 La graphe de jeu	31
17.2.2 Palette d'actions	32
17.3 Lancement du jeu	34
18 Les tests du site	34
IV Conclusion : Résultats et perspectives	35
19 Réussites du projet	35
20 Erreurs commises	36
21 Améliorations possibles	36
22 Bibliographie	37

Première partie

Étude du projet

1 Contexte et problématique

Contexte

Ce projet s'inscrit dans le cadre du PFA (projet au fil de l'année) proposé en 2ème année de la filière Informatique de l'école d'ingénieur ENSEIRB-MATMECA, entre octobre 2016 et avril 2017. Notre équipe est composée de sept membres (Thomas Caroff, Remicia Di Franco, Xavier Dussieux, Adnane Khattabi, Ahcène Mahtout, Nathan Nguyen et Thomas Saux) et est encadrée par un responsable pédagogique : Myriam Desainte-Catherine, enseignante à l'ENSEIRB-MATMECA et chercheuse au LaBRI.

Notre équipe est en charge du projet *Neuron It!*, proposé par Xavier Hinaut, chercheur à l'INRIA et membre de l'équipe Mnemosyne dont le sujet de recherche concerne les neurosciences computationnelles et l'apprentissage artificiel.

Problématique

Le cerveau humain possède des capacités d'apprentissage et de mémorisation très performantes mais aussi très complexes. La modélisation virtuelle du cerveau (on parle de réseaux de neurones artificiels) est aujourd'hui le sujet de beaucoup de recherches à travers le monde. Pour réussir à comprendre comment le cerveau apprend et mémorise, nous allons avoir recours à une approche appelée le *crowdsourcing*.

Le *crowdsourcing* est un concept de production participative par l'utilisation de la créativité, de l'intelligence et du savoir-faire d'un grand nombre de personnes, en sous-traitance. Dans ce projet, nous espérons que le *crowdsourcing* permettra d'obtenir des solutions diverses et originales à un problème difficile malgré les possibilités de résolution numériques, tel que l'a illustré le jeu *Foldit* ; les résultats pourront alors être interprétés par des chercheurs et permettront de mieux comprendre comment fonctionne le cerveau.

2 Choix des outils et méthodes de développement

2.1 Environnements de développement

En ce qui concerne le partage du code, nous avons choisi d'utiliser le logiciel de gestion de versions Git entre membres du groupe. Un autre dépôt sur Github nous a permis de transmettre le contenu de notre travail à Mr Hinaut.

Nous avons choisi de pouvoir développer la totalité du projet sur les machines disponibles à l'ENSEIRB. En effet, cela permet dans un premier temps d'assurer que chacun puisse participer au développement, qu'il dispose ou non d'une machine personnelle, mais aussi de garder un même contexte de développement. Ce choix s'est avéré problématique sur quelques points. En effet, ne disposant pas des droits d'installation sur les machines, il fallait parfois trouver un moyen pour détourner l'installation de certains logiciels, ce qui peut être long et hasardeux. Finalement, une fois tous les outils mis en place, aucun autre problème du genre n'est venu troubler l'équipe.

2.2 Outils de communication

Afin de communiquer de façon efficace avec le client et l'encadrant, nous avons jugé pertinent de créer une adresse mail commune à tous les membres de l'équipe, ce qui permet d'éviter de devoir faire de nombreux transferts. Par ailleurs, ce compte email nous a aussi donné l'accès à un service de stockage de document collaboratif en ligne (Google Drive), qui a été utilisé lors de l'édition des documents de l'équipe. Il était ainsi facile de partager, d'accéder ou d'éditer les documents tels que les comptes-rendus de réunion ou les liens de documentation utiles, les configurations techniques importantes (par exemple les astuces d'installation d'un outil particulier pour lequel nous n'avions pas les droits d'installation classiques) et les documents de

spécifications qui étaient alors directement accessibles et commentables par le client depuis un lien.

Les membres de l'équipe communiquaient entre eux grâce à Telegram, un outil de messagerie instantanée disponible sur tout type d'appareil, ce qui permettait à chacun d'être réactif aux demandes des autres membres du groupe, et d'échanger efficacement entre développeurs travaillant sur des fonctionnalités voisines.

2.3 Méthode agile : Scrum

Nous avons choisi de travailler en méthode agile pour ce projet. Une méthode agile est une méthode de développement informatique qui repose sur un cycle de développement itératif, incrémental et adaptatif permettant de concevoir des logiciels en impliquant au maximum le client, ce qui permet une meilleure compréhension de ses besoins au fil du projet. Puisque de nombreux aspects du projet n'étaient alors pas clairement définis au début et que ce projet proposait une forte composante de recherche, la méthode agile, ou les fonctionnalités et les objectifs peuvent être revues au cours du projet, semblait bien plus convenante qu'une méthode impliquant un cycle en V.

Au début du projet, les membres de l'équipe ont suivi une initiation aux méthodes agiles, et plus particulièrement la méthode *Scrum* dans le cadre de la scolarité à l'*ENSEIRB-MATMECA*. Nous avons donc naturellement choisi d'appliquer cette méthode qui était adaptée à notre projet, ce qui s'est avéré délicat au début, quand aucun membre n'avait encore fini la formation.

3 Élaboration du document de spécification et application de la méthode agile

Cette étape du projet, consistant à bien comprendre et à mettre en forme les attentes du client, a occupé une grande partie du projet. Nous avons d'abord dû nous approprier le sujet, ainsi que nous documenter sur les réseaux de neurones et le framework proposé par le client, pour avoir une idée plus précise du travail demandé. Il a également fallu diviser ce travail en tâches et évaluer l'effort, le risque et le retour d'investissement pour chacune d'elles, conformément à la méthode *Scrum*. Le projet a été découpé en plusieurs phases de développement, chacune définie par ses objectifs et tâches propres, précisées lors de la phase correspondante.

Cette méthode s'est avérée délicate à respecter. Premièrement, aucun membre de l'équipe n'était encore formé aux méthodes agiles au début du projet. Ensuite, le projet n'étant pas réalisé à temps plein, les rendez-vous avec le client n'ont pas eu lieu à une fréquence suffisamment élevée. Nous avons néanmoins essayé de garder le côté collaboration caractéristique à la méthode en communiquant régulièrement par mail avec le client pour l'informer de l'avancée de notre travail et recueillir ses impressions et ses demandes. Le caractère même du projet et notre manque d'expérience pour identifier les tâches et les évaluer en terme de complexité, de temps ou de risque, ont mené à de mauvaises prédictions quant aux tâches qui pourraient être réalisées pendant les phases de réalisation. En effet, de nombreuses tâches ont pris bien plus de temps que prévu, surtout lorsqu'il s'agissait de découvrir une nouvelle technologie par exemple. Mais certaines autres ont été réalisées bien plus tôt que prévu. Finalement, lorsque l'équipe a commencé à réellement saisir le projet et que les techniques étaient maîtrisées, le travail s'est poursuivi de façon fluide mais sans nécessairement prévoir à l'avance l'ordre et la durée des tâches.

C'est finalement aujourd'hui à la fin du projet qu'il apparaît plus clairement quelles étaient les tâches à considérer, et la façon de les découper. À l'avenir il faudra s'assurer de bien comprendre le sujet, la demande du client, afin de pouvoir déterminer les tâches à effectuer avec précision par rapport aux besoins réels du client. Il convient ensuite de bien connaître les différents domaines que touche le projet, sans quoi les estimations de difficulté, risques et temps mèneront à de mauvaises prédictions.

4 Objectifs du projet

L'objectif de ce projet est de créer une interface web de *crowdsourcing* sur laquelle les internautes pourront manipuler des réseaux de neurones afin de résoudre des jeux proposés parmi ceux de la plate-forme *OpenAI Gym*. Cette interface devra être intuitive de manière à ce qu'un maximum de personnes puisse résoudre les

problèmes proposés sans avoir de connaissance particulière des réseaux de neurones.

Plus particulièrement, le site doit contenir certaines fonctionnalités :

- Au moins un jeu d'*OpenAI Gym* doit être fonctionnel.
- Il doit contenir des tutoriels indiquant comment manipuler les réseaux de neurones pour résoudre les jeux.
- Les réseaux de neurones doivent être représentés de façon simple pour que la résolution soit intuitive.
- Le site doit disposer d'un onglet pour contacter les administrateurs en cas de dysfonctions, problèmes ou questions.
- Les joueurs peuvent accéder à un tableau des meilleurs scores pour comparer leurs résultats.
- Il doit être attaché au site un forum de discussion.
- Des informations concernant les réseaux de neurones pour comprendre l'enjeu du site
- Un système permettant aux joueurs de s'authentifier et de sauvegarder leurs parties.

Outre les objectifs techniques proposés par le projet, le *PFA* a un but pédagogique. Il s'agit d'un travail sur le long terme, où l'équipe travaille avec une grande autonomie. Elle lui revient donc, tout en étant encadrée par son responsable pédagogique, de gérer l'organisation (prise de contacts avec le client, prise de rendez-vous, gestion des ressources), la répartition du travail entre les sous-équipes et les membres des équipes ou bien la gestion des conflits et des problèmes.

5 Répartition du travail

Au début du projet, durant la phase de spécification, chacun des membres avait le même rôle. En effet, il s'agissait alors de se documenter sur les possibilités offertes par les réseaux de neurones, le framework utilisé, les sites de *crowdsourcing* similaires, afin de définir les contours et la portée du projet. Régulièrement la recherche s'arrêtait pour effectuer une mise en commun des informations et une réflexion commune, pour aboutir aux meilleures idées possibles.

Puis, rapidement, deux parties se sont dégagées pour la réalisation du projet :

- *backend* : cette partie a pour rôle l'implémentation des réseaux de neurones (fournis ou non par le client), des intermédiaires de calcul, ainsi que les algorithmes d'apprentissage pour ces réseaux. De plus, le *backend* est responsable de la création des scripts *Python* qui doivent être lancés derrière l'ensemble des fonctionnalités du site afin d'effectuer tout le calcul de l'apprentissage des réseaux. Une étude des réseaux de neurones doit aussi être effectuée afin de dégager les paramètres envisageables à fournir aux utilisateurs du site.
- *frontend* : création du site-web contenant l'ensemble de l'interface graphique qui traduit les fonctionnalités des réseaux de neurones réalisées par le *backend*. Cette interface doit avoir un caractère intuitif pour faciliter la navigation à travers le site pour des utilisateurs novices dans le domaine des réseaux de neurones. De nombreux aspects supplémentaires sociaux et ludiques doivent être implémenter pour agrémenter le site-web (leader board, forum, tutoriels, et autres).

La répartition du travail durant ces phases sera précisée dans les parties correspondantes de ce rapport.

Deuxième partie

Réalisation et état courant du *backend*

Glossaire

Avant d'entamer les descriptions formelles des différents types de réseaux de neurones, il est nécessaire de préciser toute la terminologie technique qui y figure par la suite. Voici alors une liste de termes qui pourraient mener à une incompréhension ou confusion de sens :

- **Époque**: représente l'intégralité d'une partie de jeu menant à une victoire ou une défaite.
- **Simulation**: processus d'apprentissage du réseau de neurone et qui contient plusieurs époques.
- **Décision**: au fil d'une époque, les décisions sont les choix (souvent binaires) effectués par le réseau lors d'un tour de jeu.

6 Les réseaux de neurones

La réalisation des objectifs du *backend* se base sur une compréhension minimum du fonctionnement des réseaux de neurones. Ainsi nous avons eu l'occasion d'étudier un ensemble de réseaux de neurones, fournis par le client pour la plupart. Dans cette partie seront décrits les différents réseaux utilisés au cours du projet. Nous commencerons par évoquer les composants et les propriétés des réseaux de neurones.

6.1 Neurones, propagation et rétro-propagation

Le neurone

Le neurone est la brique élémentaire constituant les réseaux de neurones. À celui-ci sont associées plusieurs entrées et une sortie, la sortie étant liée aux entrées par la relation 1.

$$S = f\left(\sum_{i=1}^n (x_i * w_i)\right) \quad (1)$$

Dans la relation 1 : n est le nombre d'entrées du neurone, x_i la valeur de l'entrée i et w_i sa pondération. La fonction f utilisée est généralement une fonction seuil permettant de définir une sortie du neurone. La fonction seuil implémentée au sein de nos différents réseaux est la fonction tangente hyperbolique et donc $S \in [-1, 1]$.

Propagation

La propagation consiste à transmettre les données de l'entrée du réseau à travers l'ensemble des neurones qui le constituent et ceci en utilisant la fonction de somme et la fonction d'activation explicitées dans la formule précédente de S .

Rétro-propagation

La rétropropagation consiste à rétropager l'erreur commise par un neurone à ses synapses et aux neurones qui y sont reliés. Pour les réseaux de neurones, on utilise habituellement la rétropropagation du gradient de l'erreur, qui consiste à corriger les erreurs selon l'importance des éléments qui ont justement participé à la réalisation de ces erreurs : les poids synaptiques qui contribuent à engendrer une erreur importante se verront modifiés de manière plus significative que les poids qui ont engendré une erreur marginale.

6.2 Le perceptron simple

Le perceptron simple est un réseau constitué d'un unique neurone, dont le schéma est présenté en figure 1 ci-dessous.

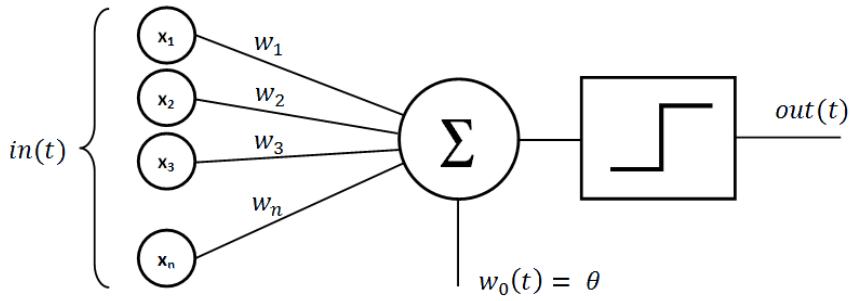


FIGURE 1 – Schéma du perceptron simple.

- **La propagation** : consiste globalement à calculer S .
- **La rétropropagation** : consiste à comparer la sortie du réseau au résultat prévu et de récompenser ou punir le réseau en fonction de sa marge d'erreur. Ceci est assuré par une modification des poids des liaisons des neurones.

Puisque le perceptron simple n'est constitué que d'un seul neurone, il ne permet pas de résoudre de problèmes complexes.

6.3 Le perceptron multicouches

Le perceptron multicouches est une amélioration du perceptron simple. Il est caractérisé par son architecture composée de plusieurs couches qui elles contiennent de un à plusieurs neurones. Dans le perceptron multicouche, un neurone d'une couche donnée est relié à l'ensemble des neurones de la couche suivante. Par ailleurs, la fonction d'activation des neurones reste inchangée de celle du perceptron simple. Le schéma du perceptron multicouche est visible en figure 2.

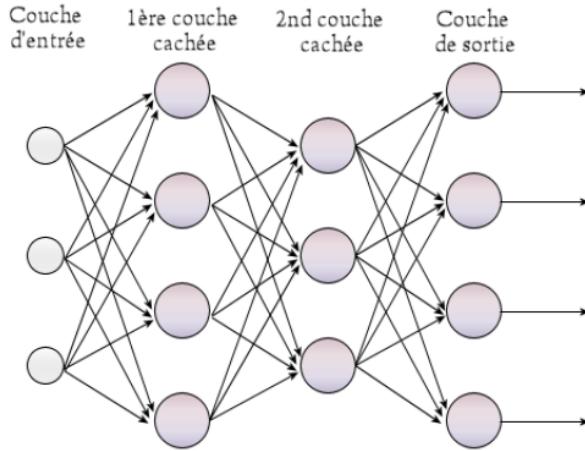


FIGURE 2 – Schéma du perceptron multicouche.

L'apprentissage de ce type de réseau se fait en deux étapes distinctes :

- **La propagation** : similaire à celle du perceptron sauf qu'elle parcourt l'ensemble des couches du perceptron multicouche.
- **La rétropropagation** : basée sur l'algorithme de la rétropropagation du gradient suivant :

pour une sortie du réseau donnée y de taille n on définit l'erreur e par $e_j = d_j - y_j$ où d est la sortie souhaitée du réseau. Ensuite on définit l'erreur globale par $\epsilon = \frac{1}{2} \sum_{j=1}^n e_j^2$. La modification du poids d'une liaison entre deux noeuds i et j est définie par la dérivée partielle de l'erreur globale par rapport à sa $j - i$ eme composante multipliée par y_i et un facteur μ représentant le taux d'apprentissage du réseau. La matrice de poids est alors mise à jour selon l'équation 2.

$$\Delta w_{ij} = -\mu \frac{\partial \epsilon}{\partial v_j} y_i \quad (2)$$

Cet algorithme d'apprentissage est effectué pour chaque itération de la simulation et un apprentissage efficace du perceptron multicouches nécessite un très grand nombre d'itérations (la grandeur de ce nombre dépend de la complexité du problème à résoudre et de la matrice des poids initialisée aléatoirement au début de l'apprentissage).

6.4 Des réseaux récurrents : Elman et Jordan

Les réseaux récurrents sont à leur tour une modification des réseaux de type perceptron multicouches car ils introduisent des neurones appelés **neurones de contexte** (entouré par un cadre bleu dans la figure 3). Ces neurones de contexte permettent de sauvegarder l'état du réseaux à un instant donné t pour ensuite utiliser cet état pour la prise de décision durant l'époque suivante.

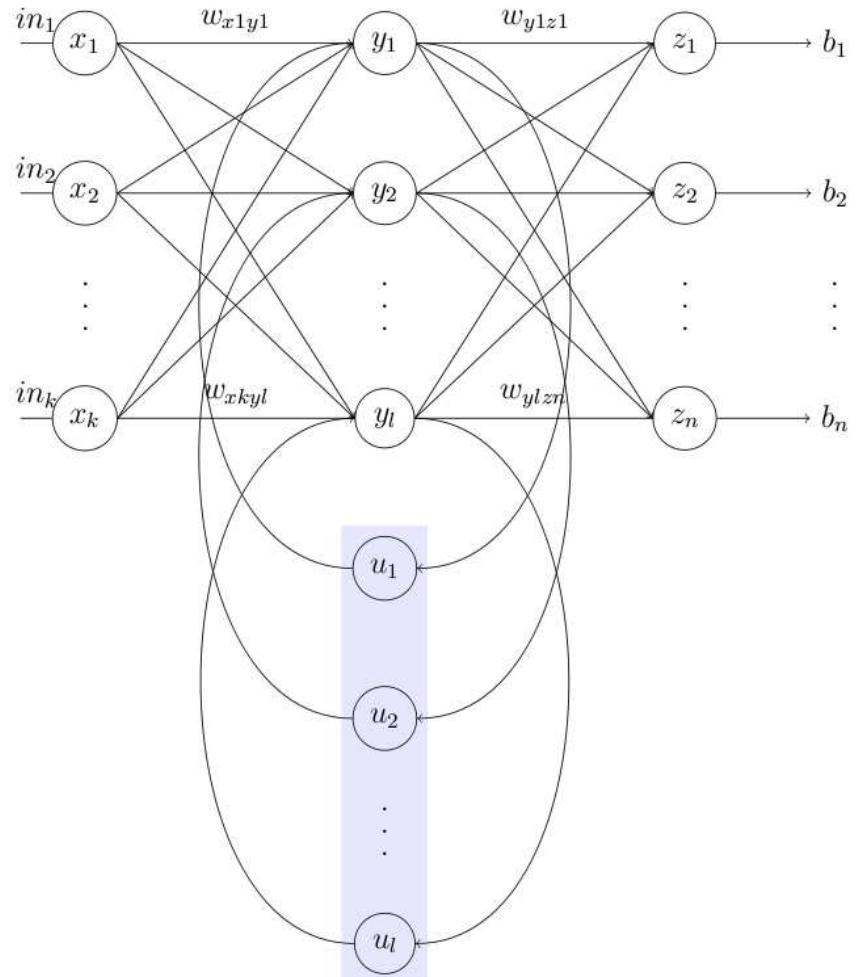


FIGURE 3 – Schéma d'un réseau de type Elman.

La façon par laquelle le réseau alimente ses neurones de contexte diffère entre les réseaux de type **Jordan** et ceux de type **Elman** :

- **Elman** : Chaque neurone de la couche cachée est relié à un neurone de contexte avec une pondération égale à 1 pour assurer la propagation de l'état vers ces neurones de contexte. Les liaisons réciproques permettent quant à eux d'alimenter la couche cachée durant l'époque suivante par une pondération fixée.
- **Jordan** : La différence avec le type de réseaux **Elman** figure dans la source d'alimentation des neurones de contexte, en effet les neurones de contexte dans les réseaux de type **Jordan** s'alimentent à partir de la sortie du réseau et propagent l'information vers la couche cachée du réseau.

L'algorithme d'apprentissage devient alors une **rétropropagation dans le temps (BackPropagation Through Time)** qui est similaire à l'algorithme de rétropropagation normal, mais qui prend en compte pour chaque époque, le contexte du réseau durant l'époque précédente.

6.5 Prise en main des réseaux de neurones par la création d'un auto-encodeur

Afin de compléter le jeu de réseau fourni par le client, un réseau de type auto-encodeur a été implémenté. Ce réseau, composé de trois couches, permet de réduire la taille d'une entrée en lui apprenant à reproduire celle-ci, d'où le terme auto-encodeur.

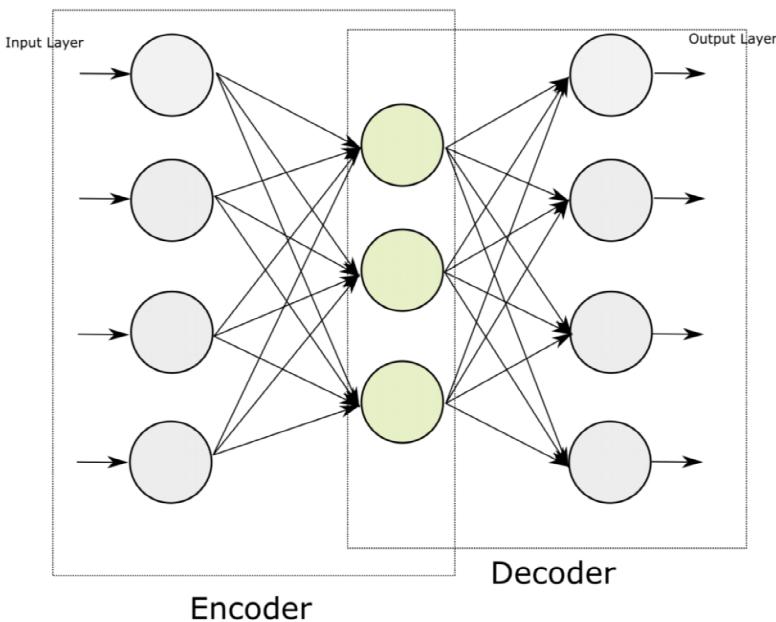


FIGURE 4 – Schéma de l'auto-encodeur.

Sur cet exemple, l'information de départ, de dimension 4, est approchée par un vecteur de taille 3 (représenté par la couche intermédiaire composée de neurones en jaune dans la figure 4 ci-dessus).

6.6 Les réseaux opérateurs

Pour les besoins initiaux qui consistent en le développement d'une plate-forme de jeux où il est possible de combiner les réseaux avec de résoudre des problèmes, il était nécessaire de décider de la manière dont les réseaux peuvent être combinés et de développer les outils nécessaires à ces combinaisons. Les réseaux

opérateurs, qui sont la solution à cette problématique, ne sont pas des réseaux de neurones à proprement parler. Ils permettent de réaliser des opérations arithmétiques basiques sur des vecteurs. Ils peuvent donc se placer avant ou après les réseaux. Ces opérateurs sont implémentés selon le même schéma que les réseaux de neurones et possède les méthodes utilisées pour les réseaux de neurones, ce qui permet au code d'exécution d'être très générique. C'est pourquoi nous conservons cette appellation de réseau opérateurs.

- L'opérateur d'addition permet de sommer un nombre de vecteurs supérieur ou égal à deux.

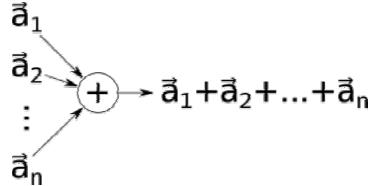


FIGURE 5 – Opérateur d'addition.

- L'opérateur de soustraction permet de faire la différence de deux vecteurs.

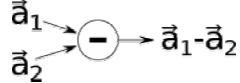


FIGURE 6 – Opérateur de soustraction.

- L'opérateur de multiplication permet de réaliser le produit terme à terme d'un nombre de vecteurs supérieur ou égal à deux.

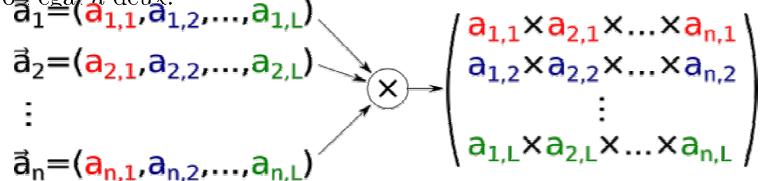


FIGURE 7 – Opérateur de multiplication.

- L'opérateur de division permet de déterminer le quotient de deux vecteurs.

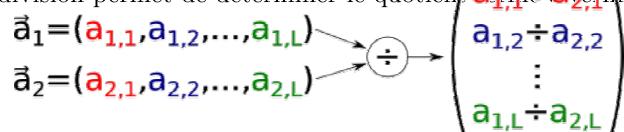


FIGURE 8 – Opérateur de division.

- L'opérateur moyenne calcule la moyenne terme à terme d'un nombre de vecteurs supérieur ou égal à deux.

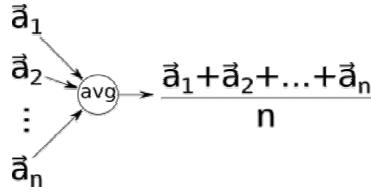


FIGURE 9 – Opérateur de valeur moyenne.

- L'opérateur de recomposition permet de concaténer un nombre de vecteurs supérieur ou égal à deux.

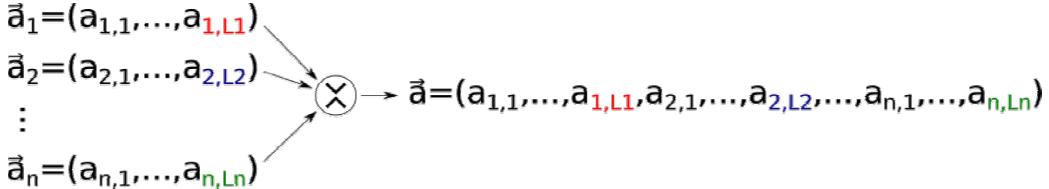


FIGURE 10 – Opérateur de recomposition.

- L'opérateur de décomposition permet de séparer un vecteur d'entrée en plusieurs vecteurs de taille déterminée dont la somme des tailles est égale à la taille de l'entrée. C'est le seul opérateur qui nécessite un paramètre, à savoir le schéma de découpage, représenté comme une liste rattachée à l'opérateur sur le schéma.

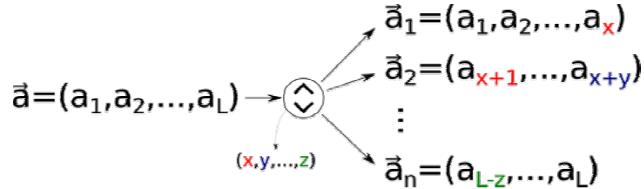


FIGURE 11 – Opérateur de décomposition

- L'opérateur de temporisation sert à étendre la portée de la réponse d'un réseau. Son utilité sera détaillée ultérieurement.



FIGURE 12 – Opérateur de temporisation.

7 Résolution d'un jeu de l'environnement OpenAi Gym à l'aide d'un réseau de neurones

7.1 CartPole-v0

Le jeu du *cart pole*, présenté en figure 13 consiste à maintenir un barreau à la verticale sur un chariot. Ce jeu est issu des bibliothèques d'environnement de la compagnie OpenAI.

Entrée Le réseau prend en entrée un quadruplet $(x, \dot{x}, \theta, \dot{\theta})$ où :

- $x \in [-2.4, 2.4]$ est la position du chariot,
- \dot{x} est la vitesse du chariot,
- $\theta \in [-5^\circ, 5^\circ]$ est l'angle du barreau par rapport à la verticale,
- $\dot{\theta}$ est la vitesse angulaire du barreau.

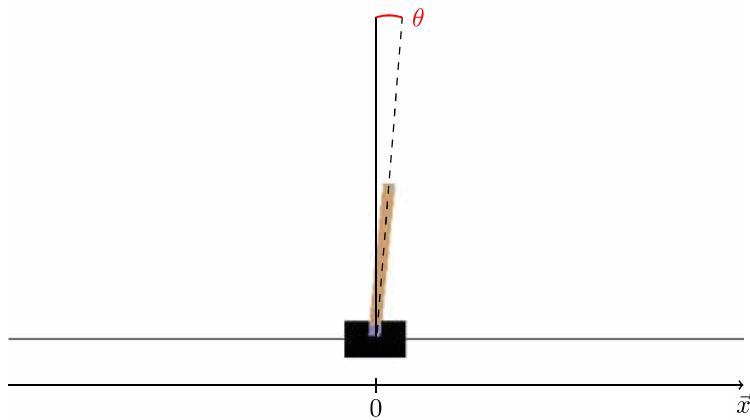


FIGURE 13 – Jeu du *cart pole*.

Sortie Le réseau, sur l'entrée $(x, \dot{x}, \theta, \dot{\theta})$, renvoie un réel $c \in [-1, 1]$ qui correspond à son choix : lorsque $c > 0$, le chariot part vers la droite, sinon, il part vers la gauche.

L'objectif est d'atteindre un score d'au moins 200, de façon la plus régulière possible¹, par conséquent le jeu est arrêté lorsque le réseau atteint un score de 200, et la prochaine itération est lancée. Les graphes présentés dans ce rapport ne prennent donc pas compte des scores supérieurs à 200.

7.2 Algorithme de résolution

Afin de faire interagir les réseaux de neurones avec le jeu CartPole, il fallu choisir quand et comment réaliser l'apprentissage du réseau.

La première stratégie d'apprentissage était la suivante : tant que l'époque n'est pas finie, alors le réseau est récompensé, et il est puni uniquement lors de l'arrêt du jeu. Cela signifie que les connexions reliant une entrée $(x, \dot{x}, \theta, \dot{\theta})$ à une sortie c sont renforcées dès que la décision ne fait pas perdre la partie. Ce qui se passe alors, c'est que le réseau renforce ses connexions qui favorisent le premier mouvement qu'il fait, puis les suivants, jusqu'à l'arrêt du jeu. On observe alors une convergence très rapide vers un choix systématique à gauche ou à droite, en fonction du premier mouvement qui lui s'effectue de manière aléatoire, car le réseau n'a pas encore appris.

Nous avons d'abord tenté de modifier le comportement lorsque une époque est perdue : une décision perdante est alors largement punie. Il est alors apparu que le réseau se comportait de la même façon au sein d'une époque, mais changeait de direction à chaque époque, puisque la dernière décision, dans le sens opposé, était très mauvaise. Pour éviter ce comportement de convergence, nous avions alors deux possibilités : créer un critère qui permet de décider si une décision était bonne ou mauvaise (par exemple *récompense* $= -(\theta/10 + |x|)$) ce qui revenait finalement à résoudre le problème sans réseau, ou décider de ne pas récompenser les actions sans savoir si elles sont bonnes.

Nous avons ensuite abandonné l'idée de faire apprendre le réseau lors d'une époque pour simplement se concentrer sur la défaite ou de la victoire de celle-ci. En effet, maintenir l'apprentissage au cours d'une époque jeu relève de l'apprentissage supervisé, que nous souhaitons éviter.

Effectivement, une fois que l'apprentissage durant les époques a été abandonné, on a remarqué un gain important dans la stabilité de l'apprentissage du réseau : le réseau se trouvait coincé fréquemment dans un minimum local ce qui nuisait gravement au résultat final de la simulation et ceci indifféremment du nombre

1. Selon les critères d'OpenAI Gym, il faut atteindre un score supérieur à 195 en moyenne sur 100 essais consécutifs pour résoudre le problème.

d'époques effectuées lors d'une simulation.

Cette phase du projet s'est donc accompagnée de la mise en place de l'apprentissage par renforcement. Celui-ci consiste à garder en mémoire la liste des décisions effectuées afin de faire apprendre au réseau non pas la dernière décision mais la liste de celles ayant conduites à la victoire (ou à la défaite).

On définit donc un facteur $\gamma < 1$ qui décroît en fonction de l'ancienneté a de la réponse, ainsi la réponse r qui sera propagée pour une série e d'événements triés du plus récent au plus ancien est :

$$\forall a \text{ tel que } \gamma^a > \epsilon, \text{ on a : } r = e[a] * \gamma^a$$

L'implémentation de l'apprentissage renforcé a aussi permis d'améliorer la vitesse de convergence de l'apprentissage des réseaux. En effet, en supposant que l'ensemble des décisions jouées durant l'intégralité de l'époque influencent le résultat (victoire ou défaite) et non seulement la dernière décision avant la fin de l'époque, on remarque un apprentissage du réseau bien plus cohérent.

Il a aussi fallu déterminer les paramètres auxquels pourraient accéder les utilisateurs du site. D'une manière générale, ces paramètres se sont imposés d'eux mêmes : se sont les variables que nous avons modifiées lors des simulations réalisées dans le but de vérifier le bon fonctionnement du code. On distingue donc les paramètres suivants :

- **La récompense de victoire** détermine l'intensité avec laquelle il faut récompenser le réseau lors d'une victoire.
- **La punition de défaite** détermine l'intensité avec laquelle il faut punir le réseau lorsqu'il perd.
- **La taille du réseau** détermine la forme d'un réseau. Bien que la taille de la couche d'entrée soit fixé par l'observation et la taille de la couche de sortie par l'action qu'il est possible de prendre sur le réseau, le choix du nombre et de la taille des couches intermédiaires est laissé libre à l'utilisateur.
- **Le multiplicateur de poids** intervient lors de l'initialisation du réseau. Lors d'une initialisation ou réinitialisation du réseau, tous les poids sont tirés aléatoirement dans un intervalle borné par ce multiplicateur de poids.

Ces paramètres sont ceux utilisées aujourd'hui dans notre implémentation. Lors de la mise en place de l'algorithme, d'autres paramètres ont semblé intéressant avant d'être par la suite retirés. Se distinguent en particulier la récompense passive, qui était utilisée lors de l'apprentissage au cours du jeu et qui fonctionnait suivant le même principe que la récompense de victoire, et l'option d'arrêt d'apprentissage, qui permettait d'arrêter de modifier le réseau une fois que celui-ci avait atteint un bon résultat.

Certains paramètres pourraient être ajoutés à cette liste. Par exemple, le paramètre γ a été fixé par la suite pour des raisons liées à la réalisation, mais il serait sans doute utile dans une version future de rajouter ce paramètres à ceux modifiables par l'utilisateur.

8 Recherche des jeux de paramètres

8.1 Recherche manuelle

Il a été accordé beaucoup de temps à la compréhension des résultats obtenus avec les différents neurones, à la recherche des paramètres à fournir à l'utilisateur et à la vérification de notre code. Pour toutes ces phases, il nous fallait connaître un certains nombre de jeux de paramètres sur lesquels il était possible travailler. Tous ces paramètres ont été recherchés manuellement à l'aide des connaissances acquises par l'équipe.

Dans un second temps, afin de pouvoir éclaircir l'utilisation des réseaux, et la détermination des paramètres pour l'utilisateur, nous avons cherché des jeux des paramètres suffisamment performants pour résoudre le jeu. Cette recherche manuelle de jeux de paramètres efficaces n'a cependant pas du tout été payante. En effet, les paramètres par défaut ne permettait pas du tout d'obtenir des résultats concluant et de résoudre

les jeux. Aujourd’hui, avec le recul nécessaire il apparaît que nous avons passé énormément de temps, sans doute trop, à effectuer cette recherche, temps que nous aurions pu dédier à des tâches plus importantes.

8.2 Optimisations avec HyperOpt

HyperOpt est une librairie spécialisée dans la recherche de paramètres. Son fonctionnement repose sur la recherche d’un jeu de paramètres permettant de minimiser une fonction : HyperOpt détermine un ensemble de valeur qui permette de minimiser le retour d’une fonction, avec les paramètres appartenant à un espace défini. Dans notre cas cette fonction est l’écart entre la moyenne des résultats obtenus et la réponse parfaite : un apprentissage complet et instantané qui permet d’atteindre le score parfait à chaque époque.

HyperOpt calcule la fonction d’écart mentionnée en plusieurs reprises pour un paramètre donné dans un intervalle précis (un intervalle pour chaque paramètre), et ceci selon un algorithme probabiliste pour la valeur du paramètre (toutes les valeurs sont équiprobables). Une fois le calcul effectué, HyperOpt renvoie la valeur du paramètre qui correspond à la valeur minimale de la fonction d’entrée : ce paramètre représente alors, le cas où la simulation se rapproche le plus du résultat idéal dans l’intervalle donné, et dans la mesure du nombre de calculs effectués.

Afin d’obtenir les résultats les plus performants possible nous avons cherché à moyenner les erreurs dans un premier temps. Pour chaque jeu de paramètres qu’HyperOpt test, au lieu de ne lancer qu’une unique fois la fonction à minimiser (il est à préciser que la fonction à minimiser à des comportements différents à chaque nouvelle époque, en fonction de l’initialisation aléatoire des réseaux et des environnements d’OpenAI Gym), qui correspond à une unique simulation, mais un grand nombre de fois et les résultats sont moyennés avant d’être rendus à HyperOpt. Il s’agit ensuite de déclarer les variables à faire varier, les intervalles auxquels les variables appartiennent, la méthode de recherche et le nombre d’évaluations à HyperOpt.

8.3 Deux formes d’apprentissage

Comme signalé en partie 7.2, nous avions constaté qu’un réseau qui arrête d’apprendre une fois la partie gagnée au moins une fois obtient de meilleurs résultats. En effet, si un réseau à appris suffisamment pour gagner, il est censé pouvoir reproduire ce résultat de façon régulière. De plus, associer une grande récompense à la victoire déstabilise fortement les poids, et le réseau revient subitement à des scores largement inférieurs.

Nous avons alors décidé de mettre en place deux formes d’apprentissage différentes : l’une avec une récompense à la victoire, et l’autre sans. La principale motivation pour le développement de ces deux formes d’apprentissage est le fait que lors des phases de recherches, nos observations montraient que la récompense déstabilisait le réseau, alors qu’il venait de réussir le test. Nous avons donc choisi de garder l’implémentation sans récompense à la victoire par défaut, mais les deux versions sont disponibles dans les sources.

8.4 Résultats

La première chose à signaler est l’écart entre les meilleures valeurs supposées des paramètres et celles calculées par HyperOpt. Un exemple qui nous a fortement marqué : la taille de la couche interne. Nous supposâmes au cours de nos recherches qu’une couche cachée de trop grande taille ne bénéficierait pas à l’apprentissage compte tenu des comportements observés. Or, les résultats d’HyperOpt ont été dans le sens opposé à cela, puisque les paramètres optimaux proposés faisaient état de 45 et 34 neurones dans la couche interne, respectivement pour les apprentissages sans et avec récompense à la victoire. Comme prévu, puisque les erreurs avaient été moyennées sur de nombreuses simulations lors de la recherche des paramètres optimaux, les résultats finaux sont très largement reproductibles et fiables.

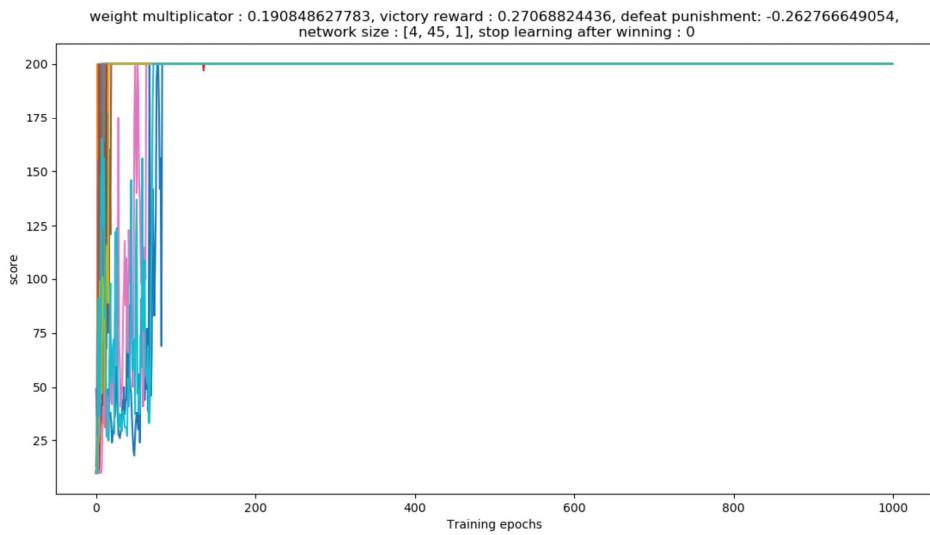
Nous avons implémenté un système de log qui sera évoqué dans la suite de ce document. Ce système permet d’enregistrer les scores moyens produits par différents réseaux en fonction de ses paramètres. Nous n’avons pas eu l’occasion d’explorer ces fichiers afin de mettre en valeur qu’elles sont les grandeurs pertinentes pour chacun des paramètres, ce qui aurait été très enrichissant concernant notre compréhension du fonctionnement des réseaux.

Nous avons pu tirer de nos observations concernant nos tests d'apprentissage les conclusions suivantes :

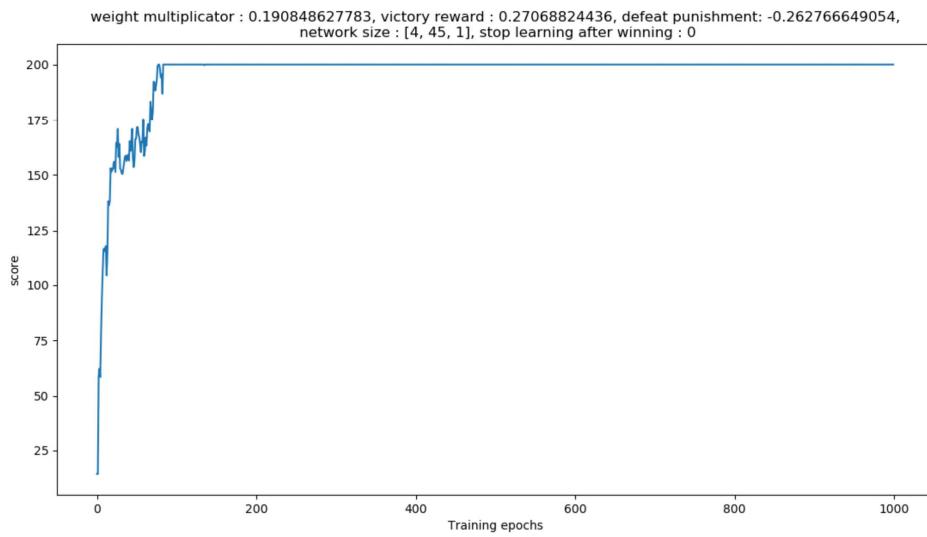
- Les réseaux n'apprennent pas plus vite, quelque soit leur méthode d'apprentissage. En effet, puisque les algorithmes d'apprentissage reposent sur le fait d'opérer, ou non, une rétropropagation avec une récompense lors de la victoire, il n'y a aucune différence de comportement jusqu'à la réussite.
- Les réseaux qui ne sont pas récompensé sont plus stables une fois la victoire atteinte, car leurs matrices des poids ne sont justement pas modifiées.

L'optimisation avec HyperOpt sur ces deux versions de l'apprentissage à fourni les résultats visibles dans les figures 14 et 15. La figure 14 illustre d'excellents résultats obtenus avec l'apprentissage sans récompense à la victoire. Les réseaux les plus rapides apprennent en une poignée d'époques², les plus lents en quelques centaines, mais tous atteignent une maîtrise parfaite du jeu. La figure 15 illustre quant à elle l'instabilité que peuvent avoir les réseaux suivant l'apprentissage avec récompense à la victoire, qui a déjà été évoquée précédemment.

2. L'exploit le plus impressionnant enregistré est un apprentissage parfait en 3 époques, et un score parfait sur les 997 époques suivantes.

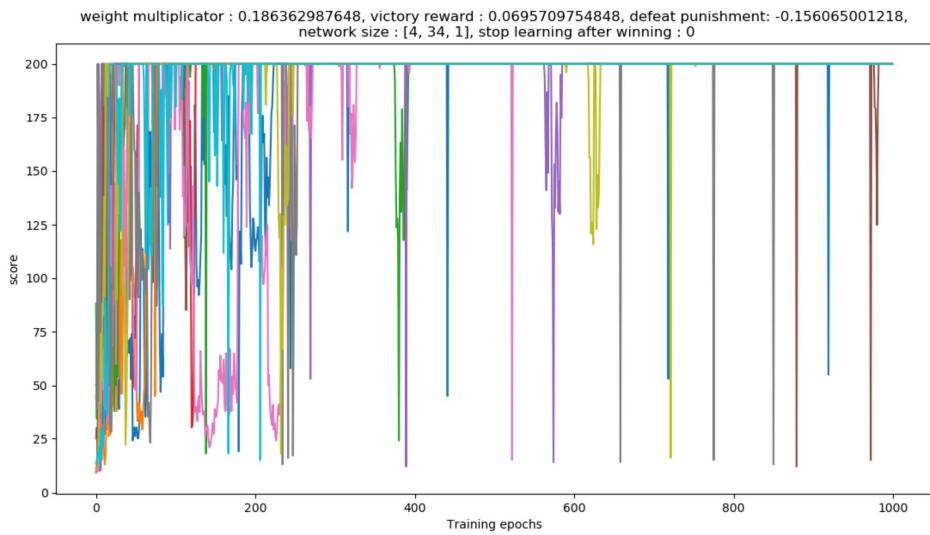


(a) Apprentissage sans victoire de chaque réseau

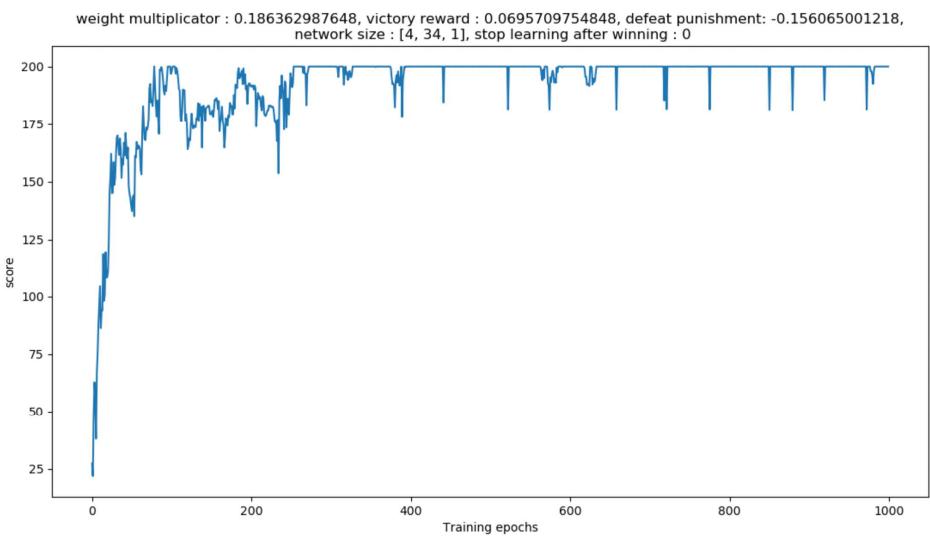


(b) Score moyen des apprentissages sans victoire.

FIGURE 14 – Résultats de l’optimisation pour l’apprentissage sans victoire.



(a) Apprentissage avec victoire de chaque réseau



(b) Score moyen des apprentissages avec victoire.

FIGURE 15 – Résultats de l'optimisation pour l'apprentissage avec victoire.

9 Fonctionnalités finales

Cette section traite du développement des systèmes actuellement utilisés par le frontend pour mener les calculs que les utilisateurs demandent.

9.1 Graphe de réseaux

Le besoin initial quant à la résolution des jeux peut, d'après les spécifications, se formuler ainsi :

- Pouvoir faire fonctionner un jeu avec l'environnement d'OpenAI Gym.
- Pouvoir résoudre ce jeu avec les réseaux de neurones.
- Pouvoir réaliser des réseaux de neurones.
- Ajout d'opérateurs élémentaires sous forme de réseaux de neurones à disposer dans les réseaux de neurones.

9.1.1 Motivation pour la construction d'un graphe et propriétés du-dit graphe

Supposons que nous disposons d'un tel réseau de réseaux de neurones, qui dispose d'une entrée unique, et d'une sortie unique (qui peuvent être multidimensionnelles cependant). Puisque les réseaux sont reliés d'une façon quelconque, qui restera à déterminer, il est possible de définir une notion de profondeur dans ce réseau, égale à la distance en terme de « liens (ou réseaux intermédiaire) à traverser » depuis l'entrée. De plus, nos réseaux de neurones ont deux propriétés particulières. La première est que les liens sont orientés, ne peuvent aller que d'une profondeur donnée vers la profondeur voisine qui est plus élevée. La seconde est que la propagation dans un réseau de réseau est par conséquent linéaire, il y a un pas de calcul qui correspond à la propagation dans les réseaux de la même profondeur, puis un pas de propagation à l'échelle du grand réseau qui correspond à déterminer les réseaux de la profondeur suivante, afin de pouvoir leur faire propager les résultats obtenus à la profondeur actuelle.

On retrouve dans ces deux caractéristiques des propriétés d'une structure en arborescence, dans laquelle la propagation d'un résultat s'effectue d'une façon similaire à une recherche en largeur d'abord, mais aussi la structure générale d'un graphe orienté, puisque un noeud d'une profondeur donnée peut joindre n'importe quel noeud de la profondeur suivante, ce qui n'est pas le cas dans un arbre. Notre réseau de réseaux de neurones est donc désormais un graphe de réseaux de neurones, qui est architecturé en plusieurs niveaux de profondeur, et on parlera désormais du **graphe de réseaux**.

9.1.2 Grille de jeu

Il est alors pertinent de proposer la chose suivante à l'utilisateur : l'utilisateur dispose d'une grille où sont placés les réseaux d'entrée et de sortie (soient respectivement le noeud qui donne l'observation fournie par l'environnement OpenAI Gym, et celui qui injecte le résultat calculé par le graphe dans l'environnement). Il peut alors placer les réseaux de son choix sur les points de la grille. Dans ce cas, la notion de profondeur apparaît directement à l'utilisateur : deux réseaux sur la même colonne sont à la même profondeur, leur résultats seront donc calculés en même temps, ou tout du moins de façon parallèle, dans le même pas de calcul. Puis il peut créer un arc entre deux noeuds de son choix, mais uniquement si'ils sont sur deux colonnes distinctes et consécutives, d'une profondeur donnée vers la suivante. Cette grille et la notion de profondeur sont illustrées en figure 16 ci-dessous.

Dans le cas présent, il n'y a plus lieu de se poser la question de propagation depuis différentes profondeurs concernant les opérateurs. En effet, un opérateur doit recevoir plusieurs résultats, mais si l'opérateur est disposé sur la grille, les résultats proviennent à coup sûr de la profondeur juste précédente, et donc arrivent au même pas de propagation à l'opérateur, et le calcul de l'opérateur pourra être effectué au pas de calcul suivant. La propagation d'un résultat s'effectue à l'aide d'un algorithme de parcours en largeur dans un graphe, par conséquent, à chaque profondeur on crée l'ensemble des calculs à propager sur la profondeur suivante. Dans le cas des opérateurs, il faudra bien veiller à récupérer tous les éléments de cet ensemble qui le

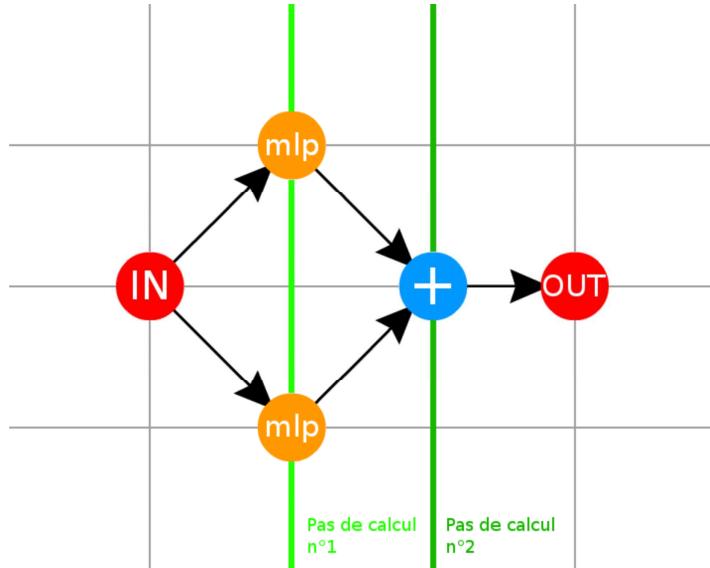


FIGURE 16 – Illustration du graphe de réseaux de neurones sur la grille et des pas de calcul.

concerne ; contrairement aux réseaux classique qui ne peuvent recevoir qu'un résultat en entrée. C'est aussi ici que l'opérateur de temporisation prend son sens : on peut vouloir propager les résultats à des profondeurs lointaines, il suffit alors de placer des opérateur de temporisation jusqu'à destination.

Enfin, il ne peut pas y avoir de cycle qui causerait théoriquement un calcul infini. Il est à noté que selon l'implémentation de la fonction de propagation dans le graphe, un cycle n'impacterai pas le bon déroulement du programme, puisque dès qu'un résultat est propagé dans le noeud de sortie, le calcul s'arrête. La véritable raison ici est que cette représentation sans cycle empêche l'utilisateur d'avoir une idée faussée de ce qui se passe dans le graphe qu'il a devant lui. En effet, il s'agit ici d'un problème de décision binaire la plupart du temps, c'est à dire que le graphe doit renvoyer une sortie réelle à sa sortie (qui sera interprétée par la suite, par exemple une sortie positive signifie droite, et une sortie négative signifie gauche), ce qui n'aurait pas de sens avec un cycle puisque la sortie obtiendrait deux décision à des pas de calcul différents, ce qui n'a pas de sens.

9.2 Initialisation et propagation dans le graphe

Initialisation Lors de la conception du graphe de réseaux, il a été fait le choix que l'utilisateur ne pourrait choisir que les tailles des couches internes et de sortie de chaque réseau. En effet, la taille de la première couche d'un réseau peut être déterminée par la taille de la dernière couche de son prédecesseur. Et puisque la taille de l'observation (i.e. la taille de l'entrée dans le graphe) est déterminée par le jeu, il est assuré que cette propriété peut se vérifier de façon récurrente sur l'ensemble du graphe, depuis l'entrée. Par ailleurs, une vérification est faite au niveau de l'interface utilisateur pour s'assurer qu'un réseau non-opérateur ne puisse pas recevoir de multiples entrées, ce qui implique l'unicité de la taille de la première couche.

L'initialisation du graphe se fait à l'aide d'un algorithme de parcours en largeur d'abord. Conformément à l'algorithme classique de parcours en largeur, lorsque qu'un noeud est atteint, les sommets voisins non encore marqués sont marqués et rajoutés à la liste des noeuds à visiter. L'algorithme a été modifié de façon à rajouter à cette liste la taille de l'entrée comme étant égale à la taille de sortie de son prédecesseur, lors du remplissage de la liste des voisins à visiter. Ainsi il n'y a pas besoin de se préoccuper de la provenance de l'information, puisqu'elle est unique et transmise avec l'indice du noeud. De plus, lors de cette phase d'initialisation les réseaux sont instanciés depuis les classes correspondantes à leurs types, leurs facteurs de poids leurs sont assignés, puis ils sont réinitialisés.

Propagation Au même titre que l'initialisation, la propagation dans un graphe se fait avec un algorithme de parcours en largeur. Les deux différences majeures ici sont que premièrement, il n'est pas nécessaire de passer cette fois la taille de la couche, mais le résultats calculé par le prédecesseur. Ensuite il faut s'assurer que pour les opérateurs qui accepte plus de deux entrées, toutes les entrées sont bien prise en compte pour le calcul. Il faudra alors constituer une liste des résultats à propager en parcourant la file des noeuds à visiter : si l'un de ces noeuds est celui qui est en train d'être traité, alors il faut prendre en compte le résultat qu'il était censé propager, et le retirer de la file.

9.3 L'apprentissage dans le graphe

Pour pouvoir faire apprendre le réseau, il est nécessaire de connaître la réponse qu'il était censé donner en sortie. Or, pour les réseaux autres que le réseau de sortie d'un graphe de réseaux, il est impossible de déterminer quelle était la sortie idéale. Afin de pouvoir malgré tout adapter le réseau à chaque époque, il est considéré que l'erreur commise par le réseau de sortie est l'erreur de tous les réseaux. Ces derniers appliquent donc la mise à jour de leurs poids avec cette erreur commune.

9.4 Autres fonctionnalités

Dans cette partie sont abordées les fonctionnalités annexes qui seront utilisées par le client dans sont besoin de recherche, et par les utilisateurs.

9.4.1 Création de logs

L'objectif du client étant de dispenser d'un site de *crowdsourcing* permettant la découverte de nouvelles façon de penser l'utilisation des réseaux de neurones, il est essentiel de fournir un outil d'agrégation des données utiles, qui pourront être analysées par la suite. L'une des tâches du développement du *backend* à donc consisté en le développement d'un système de log, où toutes les informations utiles pour l'analyse de la partie sont stockée.

À chaque nouvelle partie, un log est donc créé contenant toutes les informations nécessaires à propos du jeu, et qui contiendra par la suite (au cours du déroulement de l'exécution de la partie) les résultats obtenus par le graphe de l'utilisateur. Afin de pourvoir identifier ces logs et de garantir leur unicité, le nom du log est formé par le nom d'utilisateur, le nom du jeu et la date précise à la seconde. Il est alors fortement improbable que deux logs rentrent en collision et fusionnent.

9.5 Sauvegarder et charger une partie

Le code du backend embarque un système de sauvegarde et de chargement qui permet au joueurs de reprendre d'anciennes parties avec la configuration du graphe, et le stockage des poids des réseaux en attendant l'exécution.

La sauvegarde est, au même titre que le log, unique. Ainsi le système de nommage est identique à celui des logs. Le système de sauvegarde est appelée régulièrement, modifie le fichier de sauvegarde à une fréquence précisée dans l'appel de fonction. Si le fichier de sauvegarde n'existe pas, alors il est créé, et sinon il est chargé.

9.5.1 Gestion des erreurs

Afin de pouvoir faire comprendre au joueur les erreurs qu'il commet et qui empêchent la bonne exécution du programme, les erreurs lèvent des exceptions et renvoient des messages d'erreur explicites pour le joueur. Par exemple, lorsqu'aucun chemin ne relie l'entrée à la sortie, une erreur `Execution never reach "out"` est levée.

10 Tests

Toutes les fonctionnalités du *backend* ont été testées fonctionnellement et avec des tests unitaires représentatifs de cas dont le résultat attendu était connu, et des cas d'erreur. Ces tests ont permis de s'assurer de

la non régression du code lors du développement des nouvelles fonctionnalités.

11 Phase d'intégration

L'objectif final est toujours resté à l'esprit des membres de l'équipe, et les avancements respectifs du *backend* et du *frontend* permettaient de jeter la lumière sur des problématiques et des possibilités à l'échelle du projet. Par conséquent, en échangeant régulièrement sur les aboutissements espérés, les équipes semblaient se diriger vers une idée commune.

Cependant quelques changements ont dû être effectués sur le code. Premièrement les fonctionnalités ont été découpées et séparées dans des fichiers différents, ce qui permet plus facilement d'accéder à certaines fonctionnalités précises depuis l'interface utilisateur. Ces fonctions découpées ont aussi vu leur résultats modifiés : avant de lier *backend* et *frontend*, de nombreux résultats s'enregistraient dans variables partagées, mais qui ne sont pas accessibles depuis le *frontend*, par conséquent les fonctions retournent un certains nombre de résultat supplémentaires pour pouvoir accéder aux résultats, et les interprètent.

Enfin, la mise en place de la communication avec les technologies utilisées au niveau du *frontend* a impliqué la mise en place de tests et de transtypage, selon les arguments envoyés par l'interface dans les fonctions du *backend*.

12 Répartition du travail

La réalisation du *backend* peut se séparer en deux grandes phases. La première consistait à comprendre le fonctionnement des réseaux, à les interfaçer avec les environnements fournis par OpenAI Gym, et à mettre en place la stratégie d'apprentissage. La seconde phase consistait en l'élaboration du système de gestion multi-réseaux (c'est à dire faire fonctionner un graphe de réseaux comme ils ont été définis dans la partie précédente), et des fonctionnalités nécessaires au bon fonctionnement du site, à savoir : les réseaux opérateurs, le système de sauvegarde et de chargement, la récupération des données depuis le frontend.

Les membres chargés du *backend* sont restés en étroite collaboration durant toute la première phase de la réalisation. Il s'agissait de saisir et retranscrire la théorie liée aux réseaux de neurones, ce qui représentait une unique macro-tâche. C'est pourquoi nous travaillions alors en binôme ou trinôme sur les mêmes fonctionnalités, afin d'en appréhender la complexité le plus finement possible.

La réalisation de la seconde phase a été bien différente. En effet, la méthode agile, et donc la discussion avec le client, la nouvelle compréhension du fonctionnement des réseaux de neurones, et l'avancement du frontend ont permis de définir avec précisions quels seraient les besoins du serveur, il s'agissait ici de répondre à un cahier des charges alors déjà bien plus précis. Ainsi il a pu être déterminé un découpage des tâches afin de répartir la charge de travail entre tous les membres du groupe *backend*, comme la réalisation des opérateurs, ou du système de sauvegarde par exemple qui étaient deux sous-tâches.

Troisième partie

Réalisations du *frontend*

Le but de ce projet est de réaliser un site web de crowdsourcing sur les réseaux de neurones capables de jouer à des jeux de la plateforme *OpenAI Gym*. Le rôle du *frontend* correspond donc à la réalisation d'un tel site à partir de jeux de réseaux de neurones prédéfinis (voir backend).

En premier lieu, il est nécessaire de choisir avec quel langage implémenter le site. En cours, nous avons étudié les applications web à travers les langages HTML et php. Mais ici nous avions une contrainte supplémentaire : le site doit pouvoir être compatible avec les jeux de la plateforme *OpenAI Gym*. Comme les jeux et les réseaux de neurones de cette plateforme sont implémentés en langage Python, nous avons décidé que le plus simple pour nous serait de choisir un langage similaire pour réaliser le site web. C'est pourquoi nous avons choisi d'utiliser un framework Python appelé *Django*. Voilà pourquoi nous avons effectué ce choix dans le cadre très particulier de notre projet.

De plus, le framework *Django* a été publié en 2005 sous licence BSD (c'est-à-dire que c'est un framework open-source). Ainsi, ce framework a bénéficié de la force de développement d'une communauté et est donc aujourd'hui un outil fiable (le système d'authentification et la gestion des exceptions sont assez performants) et très bien documenté ce qui est un grand point positif pour notre projet. En effet, nous n'avions à priori aucune connaissance en terme de programmation web avec *Django* mais grâce à la documentation officielle très détaillée (voir *Django documentation*) et au grand nombre de sites web *Django* open source, nous avons pu comprendre les mécanismes de ce framework et réaliser le site web *Neuron It !*.

13 Les principales applications

Dans cette partie nous allons parler des principales applications (au sens du framework *Django*) qui constituent le site web *Neuron It !*. Il ne sera pas question de l'application *Jeu* ni de l'application *Tutoriel* qui seront abordés dans une prochaine partie (voir section 17.3). Il en est de même de l'application forum qui sera également abordée dans une partie ultérieure (voir section 16).

Chacune des applications qui vont être présentées ci-dessous intègre (par un mécanisme d'héritage) la barre des onglets et le pied de page. Cela permet dans un premier temps de pouvoir aisément naviguer à travers le site grâce aux onglets mais aussi d'uniformiser les applications en un ensemble commun. En effet, ce mécanisme d'héritage permet de donner à toutes les pages Web du site des éléments communs ce qui donne l'impression de naviguer à l'intérieur d'un site Web plutôt qu'à l'intérieur d'un ensemble de pages sans lien.

13.1 Page d'accueil

La page d'accueil du site est la première page à laquelle on accède lorsque l'on visite le site. Il est donc très important qu'elle soit esthétique et ergonomique. Comme la finalité du site *Neuron It !* est le crowdsourcing, il ne faut pas que ce soit une page trop technique qui fasse fuir les internautes. La page d'accueil doit introduire le site de la manière la plus simple possible.

Nous avons donc décidé que la page d'accueil serait constituée d'un carrousel³ géant occupant tout l'espace de la page. De plus ce carrousel est entièrement personnalisable depuis la console d'administration du site (voir section 15). Cela permet au client de personnaliser sa page d'accueil comme il le souhaite et de la faire évoluer au cours du temps.

Voici un aperçu de la page d'accueil :

³. En terme d'application web, un carrousel est un objet contenant plusieurs images défilant automatiquement ou par un clique de l'utilisateur.

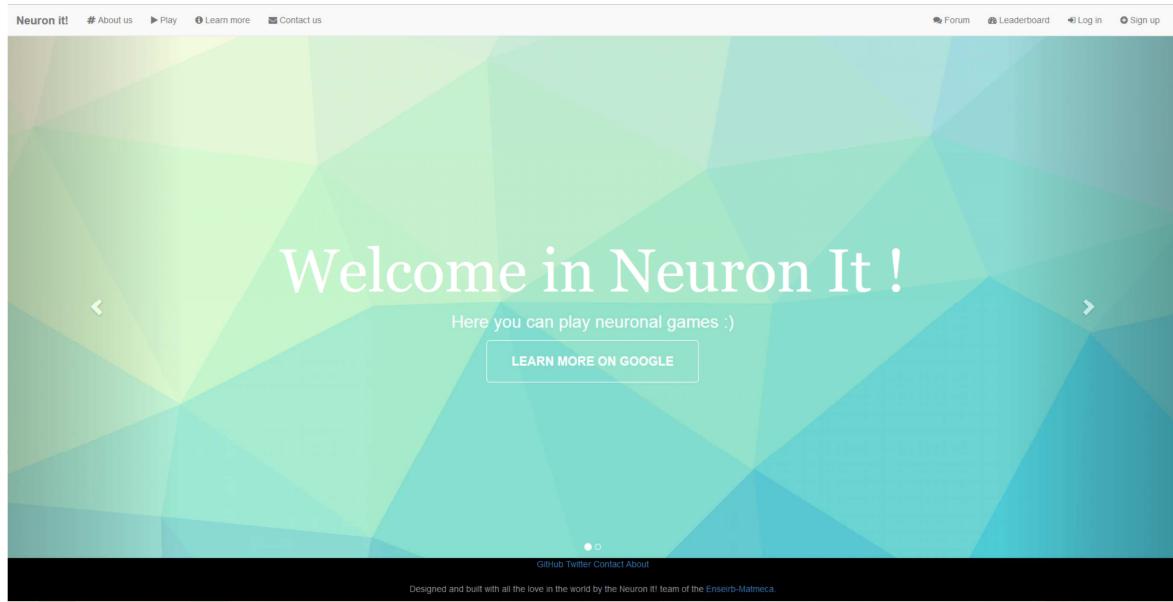


FIGURE 17 – Capture d'écran de la page d'accueil

13.2 "À propos"

La page "à propos" a pour but de présenter le projet et l'équipe de recherche qui travaille derrière. Cela permet aux joueurs de réaliser que ce projet est plus qu'un simple jeu, c'est un investissement pour la recherche en neurosciences. Il semble essentiel que chacun ait le droit de connaître la portée de ses actions.

Cette page est également entièrement modifiable depuis la console d'administration. Elle est constituée de deux parties :

- Une partie de présentation du projet et de contexte de recherche. Cette partie est composée de paragraphes que l'on peut créer à partir de la console d'administration. De plus, nous avons rendu possible l'intégration de balises HTML depuis l'interface administrateur de sorte à ce que le rendu visuel soit le plus personnalisable possible.
- Une partie présentation de l'équipe. Il est possible d'ajouter un membre de l'équipe depuis la console d'administration avec un nom, un rôle, une description et une photo. Nous avons rendu optionnel le renseignement de la description mais aussi et surtout celui de la photo (car tous les membres de l'équipe ne souhaitent pas forcément avoir leur photo en ligne). Par défaut, une image de profil neutre est générée.

Voici un aperçu de la page "À propos" :

The screenshot shows the 'About Us' page of a website. At the top, there's a navigation bar with links for 'Neuron it!', 'About us', 'Play', 'Learn more', 'Contact us', 'admin', 'Forum', 'Leaderboard', 'Settings', 'Log out', and 'Admin access'. Below the navigation is a large section titled 'ABOUT US' in bold. Underneath it, a sub-section titled 'Présentation du projet' contains a block of Latin text. Further down is a section titled 'OUR TEAM' with three small profile pictures of men. At the bottom of the page is a dark footer bar containing links for 'GitHub', 'Twitter', 'Contact', and 'About', along with a note that the site was 'Designed and built with all the love in the world by the Neuron it! team of the Eiseiit-Matmeca'.

FIGURE 18 – Capture d'écran de la page "À propos"

13.3 "En savoir plus"

La page "En savoir plus" sert à donner des éléments de réponse aux joueurs curieux qui, en jouant, se poseraient des questions techniques sur le fonctionnement des réseaux de neurones de manière générale. C'est d'ailleurs une méthode très ergonomique de générer de la connaissance chez un individu ; car si au lieu de simplement apprendre, il apprend à aimer ce qu'il fait dans un premier temps pour pouvoir ensuite apprendre et acquérir des connaissances dans un second temps, le savoir s'imprègne plus facilement et plus rapidement.

Cette page se découpe en deux sections qui sont encore une fois éditables depuis l'interface administrateur :

- Une partie résumé sur le fonctionnement des réseaux de neurones. Cela permet de répondre à des questions d'ordre général que se poseraient les joueurs. Il est intéressant que cette page soit évolutive puisqu'elle va pouvoir être adaptée en fonction des questions les plus fréquentes chez les joueurs (questions recueillies soit dans le forum soit par envoi de mail depuis la page "Contactez-nous").
- Une partie liens pertinents qui va permettre de répondre à des questions très spécifiques que pourraient se poser les joueurs. Il est nécessaire que cette partie soit modifiable depuis la console d'administration car les liens peuvent évoluer : de nouveaux liens intéressants pourraient se créer ou alors à l'inverse des liens partagés sur cette page pourraient correspondre à des sites qui ont cessé d'exister.

13.4 "Contactez-nous"

La page "contactez-nous" offre l'opportunité à n'importe quel visiteur de contacter l'équipe de recherche. Cela peut être fortement apprécié dans le cas de la remontée d'un bug découvert sur le site ou alors cela peut correspondre à une demande de renseignement quelconque. Les adresses e-mail auxquelles vont être envoyés les messages rédigés sur cette page sont modifiables depuis la console d'administration. Ainsi, il est possible de supprimer une adresse qui n'est plus valide ou d'en ajouter une nouvelle pour partager les e-mails entre les membres de l'équipe de recherche.

Voici un aperçu de la page "contactez-nous" :

Contact us

Any comments, issues, or suggestions will be examined

Enter Full name
Enter Email
Subject
Your message

[Send message](#)

[GitHub](#) [Twitter](#) [Contact](#) [About](#)

Designed and built with all the love in the world by the Neuron It! team of the [Enseirb-Matmeca](#).

FIGURE 19 – Capture d'écran de la page de contact

13.5 Classements

La page des classements permet aux joueurs de pouvoir se comparer aux meilleurs joueurs. Cela peut les inciter à essayer de faire encore mieux et à l'inverse, être affiché parmi les meilleurs joueurs est une récompense pour avoir réussi à construire un réseau de neurone performant. Le classement est aussi visible pour les visiteurs non inscrits qui peuvent ainsi voir les joueurs ayant le mieux réussi et regarder les commentaires qu'ils ont pu écrire dans le forum.

Games

Select a game

- CartPole-v0
- MountainCar-v0**
- MsPacman-v0

Leaderboard

Best 20 players

Name	Score	Date
admin	163	March 17, 2017
toto	60	March 12, 2017
foo	13	March 12, 2017

Your score:

toto 60 March 12, 2017

[GitHub](#) [Twitter](#) [Contact](#) [About](#)

Designed and built with all the love in the world by the Neuron It! team of the [Enseirb-Matmeca](#).

FIGURE 20 – Capture d'écran de la page des classements

14 Gestion de la base de données - SQLite

Le projet **Neuron It !** consiste à la base à récolter des données sur les constructions de réseaux de neurones des joueurs. Pour cela, il est essentiel d'implémenter une base de données capable de stocker toutes ces informations. De plus il est aussi nécessaire d'être capable de les ordonner intelligemment de manière à pouvoir effectuer "rapidement" des requêtes sur ces jeux de données. Par exemple, il doit pouvoir être possible de répondre "efficacement" à la question : quel joueur a obtenu le meilleur score à un jeu donné et quelle est la construction de réseau de neurone qui lui a permis d'obtenir ce score ?

De plus, comme dans la majorité des projets web, il est nécessaire d'implémenter une base de donnée de manière à pouvoir implémenter un système d'authentification interne⁴. Cela permet d'enregistrer les utilisateurs inscrits et de faire de la gestion de droits pour limiter ou non leur action sur le site. Par exemple, il peut exister une classe "omnipotente" d'utilisateurs administrateurs, une seconde classe avec moins de permissions d'utilisateurs modérateurs et enfin une classe de simple utilisateurs comportant des droits relativement restreints.

Enfin, il est également important dans un projet qui se veut évolutif et modulable d'offrir à l'administrateur du site un moyen d'effectuer des modifications sur le contenu du site sans avoir à modifier le template HTML d'une page. À ce problème également, les bases de données peuvent apporter une solution.

14.1 Pourquoi SQLite ?

Maintenant que l'on a réalisé l'importance d'utiliser une base de données, la première question à ce poser est : quel système de gestion de base de données utiliser ? Il en existe aujourd'hui plusieurs dizaines.

Parmi tous ces systèmes de gestion de base de données, nous avons choisi d'utiliser **SQLite** pour les raisons suivantes :

- Une particularité de **SQLite** est qu'il n'est pas construit sur le schéma classique client-serveur mais est directement intégré au programme. Ceci constitue la raison principale de notre choix car en effet il n'est pas nécessaire d'installer une bibliothèque supplémentaire, **SQLite** est directement intégré dans la bibliothèque standard de **Python**. De plus, il est possible d'utiliser l'abstraction qu'offre **Python Django** sur le fonctionnement de la base de données avec **SQLite** ce qui permet d'arriver rapidement à des résultats convenables ce qui est plutôt important pour un projet à court terme comme le PFA. Nous verrons par la suite que cette "légèreté" qu'offre **SQLite** est aussi une faiblesse.
- **SQLite** est un système de gestion de base de données libre, gratuit et s'appuie sur le moteur de base de donnée le plus utilisé au monde. En effet, **SQLite** est utilisé dans de nombreux logiciels tels que **Firefox** ou bien **Skype**. Ceci permet de garantir un certain niveau de confiance en terme de fiabilité et de sécurité.

14.2 Fonctionnement de la base de données

Dans cette partie, nous allons parler du rôle de la base de données dans le projet. Mais tout d'abord, il faut rappeler que nous manipulons la base de données à travers une abstraction qui est celle de **Python Django**. Ainsi les principaux outils que nous avons utilisés sont les suivants :

- Le système de modèle **Python**. Un modèle est une classe **Python** qui correspond à une table dans la base de donnée. Les attributs de cette classe correspondent à des attributs dans la table. Le modèle se construit dans un fichier appelé **models.py**. Chaque application (au sens de **Python Django**) possède leur propre fichier de modèle. Il est ainsi possible de hiérarchiser simplement la base de données en inscrivant les modèles au sein des applications qui les utilisent.

4. Il est aussi possible d'implémenter un système d'authentification externe avec **Google Firebase** pour Android par exemple

- La commande `python manage.py makemigrations <application>` compare les modèles présents dans le fichier de modèle de l'application donnée avec les tables correspondantes dans la base de données. Si elles n'existent pas alors la commande va générer un script Python d'ajout de ces modèles dans la base de données. Sinon si des modifications ont été effectuées sur le modèle, la commande va vérifier que ces modifications ne sont pas incompatibles avec les éventuelles entrées sur les tables concernées avant de générer le script d'altération de table. Ces scripts sont appelés `migrations`.
- La commande `python manage.py migrate` permet d'appliquer à la base de données toutes les migrations qui ont été générées avec la commande `makemigrations`. Si cette commande n'est pas exécutée, alors il y a une divergence entre les modèles présents dans le code du site et les tables de la base de données. Il y aura ainsi de fortes chances qu'une exception Python soit levée au moment où l'on souhaitera accéder à l'application concernée.
- La commande `python manage.py dumpdata <application>` permet de créer un "dump" des entrées de tous les modèles de l'application donnée. À l'inverse, la commande `python manage.py loaddata <dump>` permet de charger en base de données les entrées présentes dans le dump entré en paramètre.
- Les fonctions de la librairie standard de Python permettant d'effectuer des requêtes sur la base de données.

14.2.1 Description de la base de données

La base de données est découpée en 4 grandes parties :

- Les données de base de Django. Ces tables correspondent aux modèles de bases du framework Django. Parmi ces modèles, il y a des modèles relatifs à l'authentification (contenant les comptes des joueurs, les e-mails, les hashs des mots de passe, les permissions, etc). On trouve aussi un modèle stockant des logs sur les activités du site ou encore un modèle contenant les paramètres principaux du site (nom et adresse du site).
- Les données du forum : ces données sont des tables contenant des informations sur les topics du forum, les postes, les catégories, les commentaires etc.
- Les données du jeu :
 - `Game` : modèle contenant le nom des jeux de la plateforme Open AI Gym intégré au site web Neuron It !.
 - `Reseau` : modèle contenant des informations sur les différentes briques de réseau de neurones
 - `Save` : modèle contenant les fichiers de sauvegarde des parties sauvegardées par les joueurs leur permettant ainsi de pouvoir reprendre leur partie là où ils l'avaient arrêtée.
 - `Score` : modèle contenant tous les scores obtenus par tous les joueurs sur les différents jeux
 - `BestScore` : modèle contenant seulement le meilleur score de chaque joueur sur chaque jeu permettant ainsi d'alléger les requêtes de génération de tableau de classement des joueurs.
- Les données de contenu des pages du site. Ces tables ont été créées de manière à pouvoir modifier dynamiquement le contenu du site depuis la console d'administration (voir section 15).

14.3 Limites de SQLite

Il est possible qu'il devienne nécessaire de changer de système de gestion de base de données à long terme pour améliorer les performances du site. Nous avons vu un peu plus tôt que SQLite est directement intégré au programme contrairement à la plupart des autres systèmes de gestion de base de données. Ceci offre en effet une certaine simplicité d'utilisation mais limite les performances lorsque l'on est confronté à d'importants volumes de données.

Puisque le projet Neuron It ! est un projet de crowdsourcing, il est possible qu'à terme les entrées dans la base de données deviennent de plus en plus importantes et que les performances qu'offre SQLite ne soit

plus suffisantes. À ce moment là, il sera alors nécessaire de migrer vers un autre système de gestion de base de données comme MySQL qui est un des systèmes de gestion de base de données les plus utilisé au monde. Grâce à la notoriété de ces systèmes de gestion de base de données, il est possible de trouver relativement facilement des programmes ou simplement des scripts de migration d'un système à l'autre⁵.

15 L'interface administrateur

L'interface administrateur est une fonctionnalité implémentée dans la bibliothèque standard de Django. Pour l'intégrer à un projet, il suffit d'ajouter l'application `django.contrib.admin`.

Elle permet principalement de pouvoir modifier des modèles (voir définition section 14.2) en temps réel sur une interface accessible depuis le site. Pour qu'un modèle soit modifiable depuis la console d'administration, il faut déclarer une nouvelle classe dans `models.py` qui va décrire comment le modèle sera accessible depuis l'interface administrateur (quels champs vont pouvoir être affichés, quels champs vont pouvoir être utilisés comme référence pour effectuer une recherche dans la table, etc). Il faut ensuite relier la classe d'administration du modèle à la classe du modèle grâce à la fonction `django.contrib.admin.site.register`.

L'interface administrateur n'est accessible seulement par les utilisateurs ayant les droits administrateur. Pour ces derniers, le code HTML du haut de page est différent⁶ et contient un lien vers la page d'administration du site. Les autres utilisateurs n'ont pas de tel lien. De manière à éviter que n'importe qui puisse tout de même accéder à la page administrateur en ajoutant `/admin` à la fin de l'URL du site, nous avons modifié l'URL de cette page et nous avons remplacé "admin" par une longue chaîne de caractères alphanumériques générée aléatoirement. Si un utilisateur parvient à trouver cette URL et à accéder à la page administrateur, il message s'affiche stipulant que son compte n'est pas administrateur et que le contenu de cette page ne lui est pas autorisé.

Voici un aperçu de l'interface administrateur :

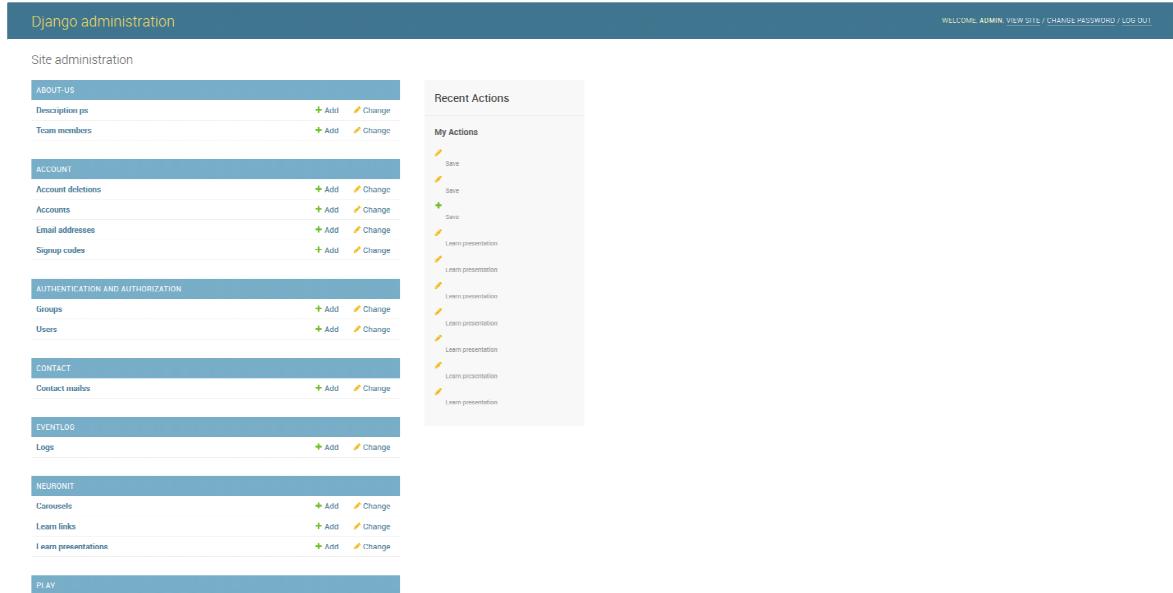


FIGURE 21 – Capture d'écran de l'interface administrateur

5. Cette migration va cependant nécessiter une mise en maintenance du site pendant plusieurs heures

6. Le code HTML est chargé dynamiquement en Python Django à partir d'un template HTML

16 Le forum avec Spirit

L'esprit de communauté prend de plus en plus d'ampleur sur internet de nos jours. Les internautes se sentent moins seul en discutant avec les autres, en partageant des expériences ou en s'affrontant sur des jeux vidéos. Il aurait été impensable de réaliser un projet de crowdsourcing sans penser à l'esprit communautaire.

Un forum permet au joueurs de s'exprimer, de se tenir informé, de communiquer avec d'autres joueurs, etc. C'est donc un bon moyen d'animer cet esprit de communauté.

Nous n'avions pas le temps nécessaire pour implémenter un forum à partir de zéro, nous avons donc décidé de regarder parmi les implémentations libres et open source. Nous avons trouvé plusieurs projet de forum en **Python Django** mais la plupart de ces derniers sont encore en version alpha ou beta. Le forum **Spirit** est l'un des seuls à posséder une version stable. Nous avons donc décidé de l'intégrer à notre site.

L'intégration n'a pas été simple, cela a créé beaucoup de conflits qu'il a fallu gérer. Il a aussi fallu raccomorder les tables de la base de données de manière à ce que les joueurs possèdent le même compte sur le site que sur le forum.

Enfin, nous avons souhaité de ne pas totalement intégrer le forum au site visuellement de manière à faire ressortir l'importance du forum par rapport aux autres applications.

Voici un aperçu du forum :

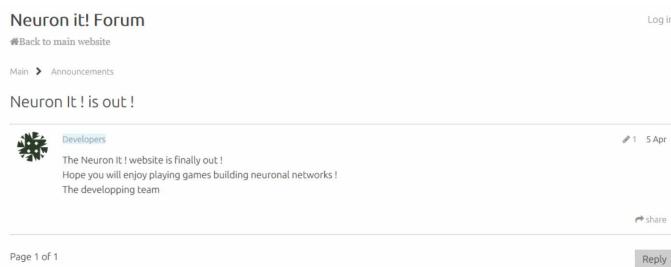


FIGURE 22 – Capture d'écran du forum avec Spirit

17 Le jeu / Tutoriel en javascript

Le cœur de ce projet était la réalisation d'une page de jeu permettant à l'utilisateur de créer des architectures basées sur les réseaux de neurones et de pouvoir faire "jouer" ces architectures à des jeux de la plateforme OpenAI Gym.

Nous avons alors découpé les tâches en plusieurs parties : une gestion des paramètres pour chaque noeud, une gestion de l'architecture en elle-même, et l'envoi au serveur de l'architecture créée par le client.

17.1 Paramètres

La réalisation des paramètres et de leur affichage s'est fait en collaboration avec le groupe *backend*. Ce dernier nous remontant les paramètres pertinents de l'architecture et des réseaux en particulier.

Après détermination de ces paramètres, nous nous sommes basés sur un modèle de formulaire pour la transmission des données et le stockage possible en base de donnée. Ce modèle est conçu en utilisant les outils Django et modèle Django permettant de générer des formulaires à partir de modèle de base de donnée.

L'avantage que nous avons de cette implémentation de formulaire est, comme dit précédemment, le stockage possible en base de donnée des paramètres données par un utilisateur ; et ceci, de façon transparente et possiblement automatique.

L'ajout de paramètres se fait par l'ajout d'attributs au modèle de formulaire. L'ensemble des formulaires liés à l'application play est stocké dans un unique fichier : `model.py`. Ce fichier permet entre autre de gérer les tables de la base données. À chaque paramètre est associé une bulle d'aide qui décrit à l'utilisateur l'utilité du paramètre. Cette information est ajoutée en même temps le paramètre.

Deux types de paramètres se distinguent dans le formulaire :

- des paramètres globaux relatifs à l'ensemble du jeu : `epoch`s et `simulations`
- des paramètres propres à chaque noeud, dépendant de leur type (`type` est un paramètre commun à tous les noeuds) : `Weight scaling`, `Victory reward`, `Defeat punishment` et `Hidden layers`

Les noeuds de type opérateurs n'ont pas accès au paramètre de noeud. La structure du formulaire nous permet de cacher les champs des paramètres non disponibles.

Une fois les paramètres données, le graphe du jeu va recevoir les informations à l'aide du bouton `Modify`. Ce bouton propre au formulaire fait à l'appel à une méthode écrite en Javascript qui modifiera la structure du graphe.

17.2 Le jeu

Nous avons séparés le jeu en deux éléments distincts pour faciliter l'expérience de jeu du joueur. Une première partie visuelle représente la construction de graphe réalisée. Ce graphe comme décrit plus haut est repéré par une grille qui limite les positions des noeuds et possède une signification particulière sur le réseau de neurones global. Une seconde partie désigne les actions à disposition du joueur qu'il peut effectuer sur le graphe.

17.2.1 La graphe de jeu

Pour la modélisation, nous nous sommes basés comme dit précédemment sur le graphe de réseau voulu par le back-end. Les noeuds peuvent être soit des noeuds symbolisant des réseaux de neurones soit des opérateurs. Pour respecter le choix de pas de calcul, les arêtes sont limités à une longueur de un. La construction de séries de réseaux de neurones étant de longueurs différentes, il était nécessaire de rajouter un noeud temporisateur ou identité dont le but était de retransmettre son entrée sans réaliser de calcul. À ce premier opérateur

s'ajoute sept autres, qui agissent sur les sorties de deux ou plusieurs noeuds. Parmi ces derniers, nous trouvons les opérateurs classiques mathématiques l'addition, la soustraction, la multiplication et la division. Nous pouvons faire appel à des opérateurs qui réunissent les sorties de noeuds ou et découpent la sortie d'un noeud afin faciliter les calculs. Il est également avantageux d'avoir la possibilité de réaliser des actions différentes sur les variables de sorties. Par ailleurs, les opérateurs n'ont pas le même nombre de paramètre en entrée. Quand l'opérateur addition, de multiplication ou de moyenne possèdent un nombre de paramètre "infini", les opérateurs soustraction et divide sont limités à deux. L'arité des opérateurs définit en conséquence le nombre d'arêtes entrantes d'un noeud-opérateur. Plus largement, voici une liste de l'ensemble des possibles arités des noeuds du graphe :

- Arité "infini" : opérateur de moyenne, de multiplication, de recomposition et d'addition
- Arité de 2 : opérateur de soustraction et de division
- Arité de 1 : réseaux de neurones (mlp, jordan et elman), noeud de sortie, temporisateur et opérateur de décomposition
- Arité de 0 : noeud entrant.



FIGURE 23 – Ensemble des noeuds présent dans le jeu - les noeuds en bleu sont les opérateurs, le noeud vert est le temporisateur, les noeuds rouges sont dans l'ordre le noeud entrant et le noeud sortant

Par défaut, le graphe de jeu est modélisé par deux noeuds de même nature un noeud entrant et un noeud sortant et d'un réseau de neurone, par défaut de type Multi-Layer Perceptron (MLP). Pour faciliter la compréhension du jeu au joueur une arête entre deux sommets de réseaux a été créer. Les noeuds peuvent à chaque croisement du quadrillage, excepté la première colonne ainsi que la dernière. En effet aucun noeud, réseau de neurone ne peut débuter avant le noeud d'entrée ou terminer après le noeud de sortie.

À l'aide de la souris il est possible de modifier la nature et les paramètres de chacun des noeuds. En cliquant sur un des noeuds du graphe, ses paramètres sont automatiquement chargés dans le formulaire. Nous pouvons obtenir très rapidement les informations d'un noeud donnée et les enregistrer en utilisant le bouton . Il est très important d'appuyer sur le bouton **Modify**. Dans le cas contraire, les paramètres modifiés ne seront pas pris en compte.

Nous avons ajouté également des retours d'informations visuels (feedbacks) afin d'améliorer l'expérience de jeu du joueur. Ces feedbacks se réalisent à l'aide de la souris et exclusivement dans notre jeu sur les noeuds et les arrêtes. Lorsque un noeud est sélectionné, ce dernier est entouré non plus d'un trait plein mais d'un trait en pointillé. De la même façon pour distinguer le graphe du quadrillage, le pointeur par défaut est un réticule lorsque nous survolons le quadrillage mais possède un autre symbole quant nous nous retrouvons au dessus d'une arrête ou d'un noeud.

17.2.2 Palette d'actions

Parlons désormais des actions modifiant la structure du graphe. En effet, nous pouvons ajouter des noeuds, les supprimer, ajouter des arêtes, les supprimer ou bien modifier le quadrillage. Toutes ces actions sont réalisables grâce à la palette de boutons se trouvant au dessus de la grille de jeu. La réalisation d'une action ne désactive pas le bouton préalablement sélectionné, au contraire nous pouvons répéter une action autant que l'on veut.

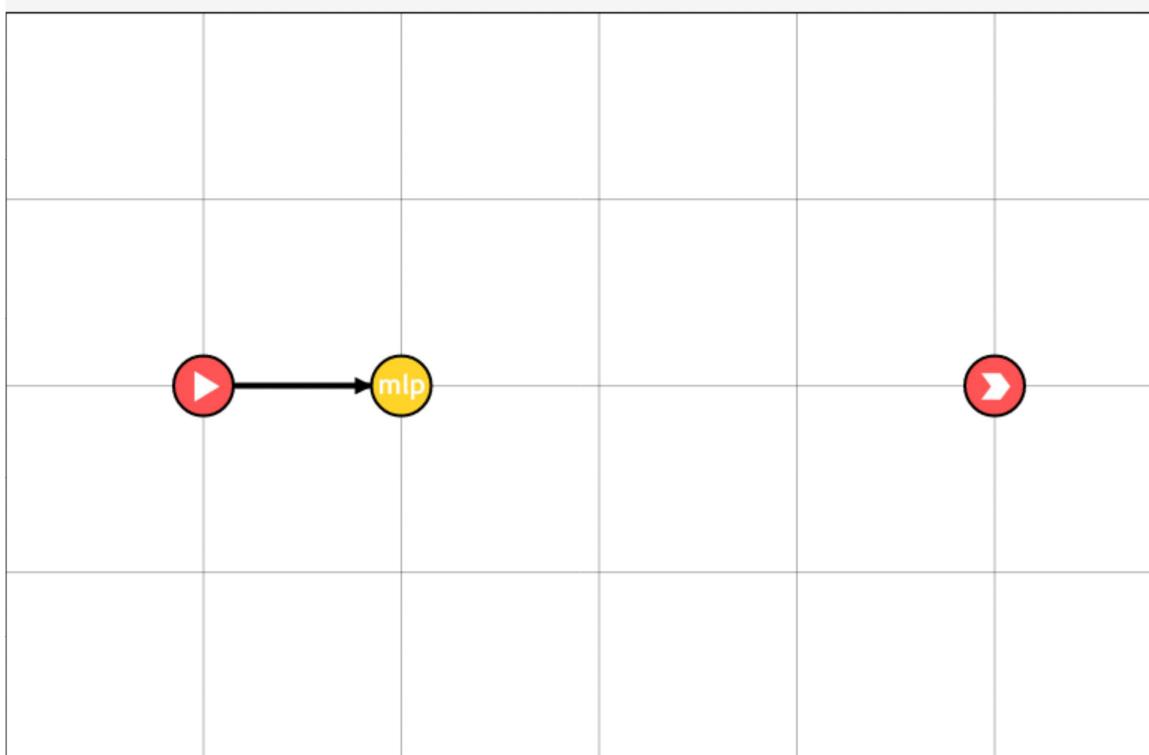


FIGURE 24 – Graphe du jeu par défaut



FIGURE 25 – Palette de boutons

Il y a 4 actions réalisables sur la structure du graphe. Avant chaque action il est nécessaire d'appuyer sur le bouton correspondant afin de déclencher sa procédure.

- **Add edge** : pour ajouter une arête, il nous suffit simplement de sélectionner conséutivement deux noeuds : un premier noeud source et un second destination. L'ajout d'une arête n'est pas fonctionnelle entre deux noeuds d'une même colonne ou bien deux noeuds distants d'une longueur de plus de un. Par ailleurs, la création d'une arête est limitée par le nombre d'arêtes entrant dans le noeud destination.
- **Add vertex** : une fois la position du nouveau noeud déterminé par le joueur, il suffit de cliquer sur quadrillage pour que le nouveau noeud se place automatiquement à l'intersection des lignes et des colonnes la plus proche. Il est évidemment interdit d'ajouter des noeuds sur l'emplacement de noeuds déjà existant.
- **Delete edge/Delete vertex** : en sélectionnant l'arrête ou le noeud, nous le supprimons. Néanmoins lorsque un noeud est supprimé l'ensemble des arêtes reliés à ce dernier sont supprimées en conséquence.

Les actions sur le quadrillage sont utiles afin d'agrandir ou réduire la taille de la surface de jeu.

- **Add line/Add Column** : l'ajout de colonne se fait par la droite tandis que l'ajout de ligne se fait par le bas. Il est possible que des arrêtes avant l'ajout soit relié au noeud sortant. A l'ajout l'ensemble des arrêtes sont supprimé pour faire place à la nouvelle colonne.
- **Delete line/Delete column** : la suppression se fait par ordre d'arrivée ou d'insertion. Ainsi quand nous demandons à supprimer une ligne, celle qui est supprimée est la ligne la plus basse dans le quadrillage et respectivement la colonne la plus à droite pour les colonnes. Lors de la suppression d'une ligne ou d'une colonne l'ensemble des noeuds présents sur cette dernière sont supprimés en conséquence ainsi que les arrêtes reliés à ces noeuds. Il est important d'être sûr de son choix lorsque nous désirons réaliser une suppression.

17.3 Lancement du jeu

Une fois que le graphe de jeu est défini par l'utilisateur, celui-ci peut modifier l'ensemble des paramètres des réseaux de neurones et réseaux opérateurs constituant ce graphe pour ensuite, une fois satisfait de la configuration globale, lancer la simulation grâce au bouton **Play**.

18 Les tests du site

Les tests qui ont été réalisés au niveau du site sont principalement visuels.

L'avantage de la console offert par les navigateurs et les rendus principalement visuels des tâches au niveau du *frontend* nous ont permis de ne pas faire beaucoup de fonctions de tests.

De plus, l'optimisation des fonctions javascripts, local à chaque page et réutilisable dans peu de conditions sur d'autres pages permet de ne se soucier que de fonctions locales à la page et non fonctionnelles ailleurs.

Enfin, les tests visuels permettent de mettre en lumière des cas particuliers, des erreurs que l'on ne soupçonne pas à première vue, étant donné la situation qui comporte beaucoup d'interactions avec un joueur sur le graphe de jeu.

Quatrième partie

Conclusion : Résultats et perspectives

19 Réussites du projet

En ce qui concerne la partie authentification et sauvegarde du **backlog**, l'ensemble des tâches de priorité haute a été réalisé. Nous avons donc un site sur lequel l'utilisateur peut se connecter et sauvegarder ses performances ce qui constituait l'essentiel de la demande du client concernant cet aspect.

Les tâches de priorités moyenne et basse suivantes n'ont, quant à elles, pas été implémentées par manque de temps :

- Permettre à l'utilisateur de s'identifier à l'aide de son compte facebook ou google+.
- Rappeler à l'utilisateur de se connecter pour conserver ses résultats avant de quitter.

Au niveau du retour des joueurs, les tâches les plus importantes étaient :

- Le client doit pouvoir avoir un retour des performances des joueurs.
- Un agencement de réseaux qui résolve tous les jeux soit trouvé.

La première tâche a été réalisée, l'ensemble des résultats étant conservé dans les logs et les sauvegardes. Le client peut donc y accéder afin de pouvoir avancer dans ses recherches en cas de résultats satisfaisants. La seconde tâche ne pourra se concrétiser qu'à la mise en ligne du site, dans le cas où les joueurs parviennent à obtenir de bons scores aux jeux proposés. Cette partie n'est donc pas de notre ressort. Nous avons néanmoins utilisé HyperOpt pour tenter d'obtenir des résultats concluants. Dans cette catégorie, nous avions deux tâches de priorité moindre, la possibilité pour le joueur de noter le jeu ainsi que celle de contacter le développeur en cas de problèmes ou d'améliorations à proposer. Cette dernière a été implémentée, l'utilisateur peut à travers l'onglet "Contact us", décrit précédemment, envoyer un message au gestionnaire du site. Le joueur n'a cependant pas l'opportunité de noter le jeu, nous avons considéré que cette tâche était secondaire et à réaliser si le temps nous le permettait ce qui n'a pas été le cas.

L'aspect communautaire du site faisait partie des tâches de priorité moyenne. Nous avons néanmoins achevé une partie du travail prévu dans ce domaine. Les joueurs peuvent donc communiquer entre eux et être informés d'éventuels évolutions du site par les développeurs grâce à un forum dédié. Ils peuvent ainsi comparer leurs résultats pour une meilleure réussite. Celle-ci est représentée par un score obtenu par le joueur qu'il peut comparer à ceux des autres grâce à l'onglet "Leaderboard". Sur celui-ci sont affichés les scores des 20 meilleurs joueurs ainsi que celui du joueur connecté et son classement. Cela permet à l'utilisateur d'augmenter sa motivation en tentant de battre les scores déjà établis. Le site permet également aux "visiteurs" de s'informer sur l'aboutissant d'un tel site de crowdsourcing et de comprendre la correspondance entre le jeu et les réseaux de neurones sur la page "Learn more". Un joueur ne peut cependant pas accéder aux solutions des autres joueurs, il peut néanmoins en discuter grâce au forum. Le système d'"amis" n'a pas été mis en place, l'utilisateur ne peut pas se "connecter" à d'autres joueurs pour avoir accès à ses résultats ou communiquer plus facilement avec eux.

A propos de l'interface, l'objectif d'obtenir un site fonctionnel permettant de jouer à au moins un jeu est accompli. La réalisation des tâches suivantes est assez subjective :

- « En tant que client, je veux que les réseaux de neurones puissent être paramétrés facilement et visuellement afin de donner de la liberté à l'utilisateur quel qu'il soit. »
- « En tant que client, je veux que différents réseaux de neurones puissent être liés entre eux facilement et visuellement par l'utilisateur quel qu'il soit afin qu'il puisse résoudre un problème. »
- « En tant qu'utilisateur, je veux que l'interface de jeu soit intuitive afin que je m'y retrouve facilement et que mon expérience de joueur soit moins désagréable. »

Nous avons essayé de rendre l'interface la plus ludique et simple possible. Pour cela, l'utilisateur peut paramétrier les réseaux de neurones à l'aide de menus déroulants contenant différentes propositions. Pour les paramètres à valeur numérique, il est possible de les incrémenter ou décrémenter à l'aide de flèches ou

bien d'écrire directement la valeur voulue. En cas de valeurs impossibles, une bulle indique à l'utilisateur l'intervalle dans lequel doit être compris le paramètre. A côté de chaque paramètre, un point d'interrogation permet de préciser la nature du paramètre en question. Les réseaux de neurones sont, quant à eux, représentés par des noeuds et peuvent être reliés par des arêtes comme décrit précédemment. Il est donc simple de lier les différents réseaux de neurones.

En ce qui concerne la fluidité du site, elle est satisfaisante pour l'utilisation du site et pour jouer à un jeu mais elle est nettement améliorable (voir les améliorations possibles).

Par rapport aux modes de jeux, plusieurs jeux sont disponibles pour le joueur. De plus, l'utilisateur peut jouer sans que la finalité du jeu apparaisse nécessairement et le client a bien accès aux scores de joueurs ce qui constituait les deux tâches de priorité moyenne. L'objectif de priorité basse qui était de pouvoir changer la langue du site n'a pas été réalisé.

20 Erreurs commises

La principale erreur commise du côté du *frontend* a été la précipitation. La volonté d'avancer rapidement la construction du site a finalement été un frein. En effet, nous avons voulu obtenir des résultats rapidement sans prendre le temps de se familiariser avec *Django* et *javascript*. Nous avons essayé de récupérer du code sur internet et de construire le site à partir de ceux-ci. Cette méthode n'a pas été concluante et nous avons dû repartir de zéro en apprenant les bases de *Django* et *javascript*. De plus, la familiarisation avec les outils techniques a été plus longue que ce que nous avions prévu, ce qui nous a fait douter sur la possibilité d'accomplissement du projet dans le temps imparti. Même si cela nous a donné l'impression de piétiner au début puisque nous ne produisions rien de concret, cette phase d'apprentissage a été nécessaire et nous a permis d'avancer plus rapidement par la suite. Nous aurions pu faire cela directement ce qui nous aurait fait gagner du temps.

De plus certaines fonctionnalités à implémenter ont été revu pour une meilleure optimisation, et même si ce n'est pas une erreur en soi, une approche plus réfléchie et moins précipitée aurait souvent pu nous faire gagner du temps.

Au niveau du *backend*, nous avons essayé d'obtenir de bons résultats d'apprentissage en cherchant de manière empirique les paramètres optimaux. Cette méthode a été longue et fastidieuse mais très peu concluante. Nous avons ensuite utilisé HyperOpt qui a été plus efficace et moins pénible à mettre en place. Essayer d'optimiser l'apprentissage du réseau a été une étape clé pour dégager les paramètres utilisateurs. Ceci dit, une grande période de temps a été consacrée à la recherche des intervalles délimitant ces paramètres ainsi que leurs valeurs optimales ce qui constitue en réalité, l'objectif des utilisateurs du site-web et non des développeurs. Il est arrivé parfois qu'on essayait de reproduire des résultats d'apprentissage du réseau avec une certaine configuration des paramètres sans prendre en considération l'aspect élusif du mécanisme d'apprentissage des réseaux. En effet, un approfondissement dans la théorie des réseaux de neurones vers la fin du projet nous a permis de comprendre la difficulté non seulement de contrôler mais aussi de saisir en détails la façon par laquelle apprenne les réseaux de neurones.

21 Améliorations possibles

L'ensemble des tâches décrites dans le *backlog* et non réalisées constitue des améliorations possibles. La fluidité du site notamment constitue un progrès important sur lequel il faudrait se pencher. D'un point de vue organisationnel, nous aurions peut-être pu suivre de façon plus méthodique la méthode agile et le planning des sprints même si le cadre de travail du projet n'était pas nécessairement adapté à celle-ci.

Au niveau du lancement du jeu, il est à noter qu'il ne tourne pas en background et qu'avec un peu plus de temps, nous aurions pu implémenter un package de gestion de background (notamment *Celery*) qui aurait permis la gestion multiple de joueurs ce qui aurait donner un aboutissement très satisfaisant du projet.

22 Bibliographie

- Interface openAI gym, <https://gym.openai.com>
- Andrej Karpathy, Mai 2016, <https://karpathy.github.io/2016/05/31/r1/>
- Documentation Django, <https://www.djangoproject.com/>
- Packages Django, <https://djangopackages.org/>
- Enseignement "deep learning" par David Silver à l'université UCL :
<http://www.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>
- *Neural Networks and Deep Learning* de Michael Nilsen, Janvier 2017,
<http://neuralnetworksanddeeplearning.com/chap1.html>
- Thèse de Carina Garber, *Crowdsourced Data Generation - How to Design a Website for Citizen Scientists ?*
- Ipython notebook avec openAigym, <http://playground.tensorflow.org>
- Greg Brockman et John Schulman, Avril 2016, <https://openai.com/blog/openai-gym-beta/>
- Discussions sur openAI gym, <https://gitter.im/openai/gym>
- Mark Harris, Avril 2016,
<https://devblogs.nvidia.com/parallelforall/train-reinforcement-learning-agents-openai-gym/>
- Patrick Mineault, Pour faire des animations JS de openAigym dans ipython notebook
<http://nbviewer.jupyter.org/github/patrickmineault/xcorr-notebooks/blob/master/Render%20openAI%20gym.ipynb>
- Nicolas Rougier, Août 2015, Tutoriels Python, numpy, matplotlib
<https://github.com/rougier/neural-networks>
<http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html>
<http://www.labri.fr/perso/nrougier/teaching/matplotlib/>
<http://www.labri.fr/perso/nrougier/teaching/numpy.100/index.html>
- Adam Geitgey, Mai 2014, Fonctionnement Machine Learning
<https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471>
<https://medium.com/@ageitgey/machine-learning-is-fun-part-2-a26a10b68df3>
- MOOC d'Andrew Ng, Avril 2003, <https://www.coursera.org/learn/machine-learning>
- Michael Nielsen, Janvier 2017, <http://neuralnetworksanddeeplearning.com/chap1.html>
- Thèse de Xavier Hinaut, Janvier 2013, <https://sites.google.com/site/xavierhinaut/phd-thesis>
- Romain Pastureau, 2016, <https://github.com/RomainPastureau/Reservoir-Jupyter>
- Skymind, <http://deeplearning4j.org/lstm.html>
- Reinforcement learning https://en.wikipedia.org/wiki/Temporal_difference_learning
<https://fr.wikipedia.org/wiki/Q-learning>
- Documentation Ajax, <http://api.jquery.com/jquery.ajax/>
- HyperOpt, <https://hyperopt.github.io/hyperopt/>

- Article sur les réseaux récurrents, http://www.wikiwand.com/fr/R%C3%A9seau_de_neurones_r%C3%A9current

ANNEXES



DEUXIÈME ANNÉE-FILIÈRE INFORMATIQUE
Bordeaux INP - Enseirb-Matmeca

NEURON IT ! - PFA

Manuel d'utilisation

AUTEURS :

Thomas Caroff, Remicia Di Franco
Xavier Dussieux, Thomas Saux
Adnane Khattabi, Ahcene Mahtout
Nathan Nguyen

RESPONSABLES DE PROJETS :

Responsable des PFA : Antoine Rollet
Professeure encadrante : Myriam Desainte-Catherine
Client (INRIA) : Xavier Hinaut

Installation du site

Prérequis

Voici la liste des outils nécessaire à la manipulation du site :

- Une machine avec un noyau linux ou une autre machine avec un environnement virtuel compatible avec Python Django
- La machine doit posséder un minimum d'espace disque disponible afin de pouvoir installer toutes les dépendances
- Python 3.0
- Installateur de Python : pip

Mise en place du site

Pour mettre en place le site, il est nécessaire de suivre les étapes suivantes (les étapes suivantes sont aussi rappelées dans le fichier `README.md` présent à la racine du site) :

- **Installation des dépendances** : à la racine du répertoire contenant le site, installer les dépendance grâce à la commande : `pip install -r requirements.txt`
- **Création de la base de données** : à la racine du répertoire contenant le site, instancier la base de données avec la commande : `python manage.py migrate`
- **Peuplement de la base de données** : à la racine du répertoire contenant le site, charger les dumps de base de donnée avec les commandes suivantes :
 - `python manage.py loaddata sites` Chargement du nom du site
 - `python manage.py loaddata base` Chargement des comptes de base et d'un poste d'exemple dans le forum
 - `python manage.py loaddata game` (*) Chargement des jeux Open AI Gym gérés par l'application play du site
 - `python manage.py loaddata types` (*) Chargement des types de réseau de neurones
 - `python manage.py loaddata neuronit` Chargement d'une diapo de base du carrousel sur la page de présentation et d'un remplissage par défaut de la page learn more
 - `python manage.py loaddata about-us` Chargement de données de base das la page about us
 - `python manage.py loaddata leaderboard` Chargement de scores d'exemple qui apparaissent sur le tableau des meilleurs score dans la page leaderboard

Le chargement des données contenant le signe (*) est nécessaire au bon fonctionnement du site. Les autres chargements ne sont pas nécessaires au bon fonctionnement du site, ils permettent seulement au site d'être entièrement complet en terme de remplissage des pages. Pour faciliter la prise en main rapide du site, le chargement du dump nommé `base` installe 2 comptes :

- admin -> mdp=admin : compte administrateur
- developers -> mdp=toto : compte modérateur

Pour des raisons de sécurité évidentes, il est vivement conseillé de ne pas charger ce dump dans la version finale du site.

- **Création d'un compte administrateur** : à la racine du répertoire contenant le site, créer un compte administrateur avec la commande : `python manage.py createsuperuser`
- **Démarrage du site** : à la racine du répertoire contenant le site, démarrer le site avec la commande : `python manage.py runserver`

Manuel de l'administrateur

L'administrateur peut modifier dynamiquement la plupart des composants du site. Il lui suffit de se connecter (par défaut il est possible de s'y connecter avec l'identifiant et le mot de passe `admin`), puis d'accéder à l'espace administrateur :



FIGURE 1 – Accès administrateur.

Django administration

Site administration

ABOUT-US

- Description ps [+ Add](#) [Change](#)
- Team members [+ Add](#) [Change](#)

ACCOUNT

- Account deletions [+ Add](#) [Change](#)
- Accounts [+ Add](#) [Change](#)
- Email addresses [+ Add](#) [Change](#)
- Signup codes [+ Add](#) [Change](#)

AUTHENTICATION AND AUTHORIZATION

- Groups [+ Add](#) [Change](#)
- Users [+ Add](#) [Change](#)

CONTACT

- Contact mails [+ Add](#) [Change](#)

Recent Actions
My Actions
None available

FIGURE 2 – Espace administrateur.

Une fois sur cet espace, l'administrateur peut modifier dynamiquement tous les composants suivants :

- **About-Us**
 - Description
 - Team members
- **Account**
 - Account deletions
 - Accounts
 - Email addresses
 - Signup codes
- **Authentication and Authorization**
 - Groups
 - Users
- **Contact**
 - Contact mails
- **Eventlog**
 - Logs
- **Neuronit**
 - Carousels
 - Learn links
 - Learn presentations
- **Play**
 - Best scores
 - Saves

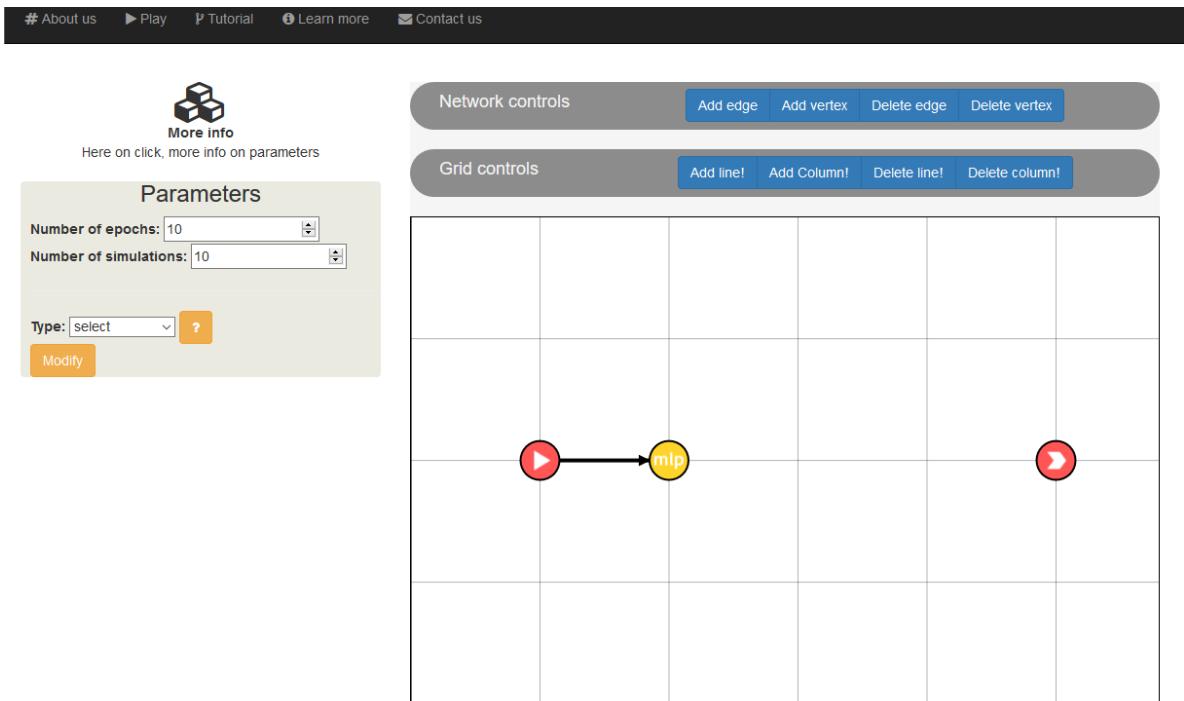
- Scores
- **Sites**
- Sites

1 Page Tutorial et Play

1.1 Tutorial

La page Tutorial a vocation à initier l'utilisateur à la création d'un réseau trivial et la modification de paramètres en lui apportant quelques informations. Elle est actuellement très basique et est vouée à amélioration.

1.2 Play



Sur la page "Play" se trouve :

- un formulaire permettant de changer les noeuds du graphe ;
- une barre d'outils permettant l'ajout, suppression de noeuds et colonnes, lignes de la grille ;
- le graphe résultant avec un noeud "in" et un noeud "out" pour les entrées des informations du jeu et sorties des actions du réseau sur le jeu.

Le jeu étant intuitif, il est facile d'y jouer sans instructions. Cependant quelques informations à retenir :

- il n'est pas possible de lier deux réseaux qui sont séparés par une colonne : un réseau de type "temporize" permet de faire la liaison sans altérations des données envoyées ;
- il n'est pas possible de lier deux réseaux sur la même colonne, l'ajout de colonnes permet de rajouter le nombre de réseaux désirées. Ainsi une colonne représente un "pas de calcul".



DEUXIÈME ANNÉE-FILIÈRE INFORMATIQUE
Bordeaux INP - Enseirb-Matmeca

NEURON IT ! - PFA

Manuel de maintenance

AUTEURS :

Thomas Caroff, Remicia Di Franco
Xavier Dussieux, Thomas Saux
Adnane Khattabi, Ahcene Mahtout
Nathan Nguyen

RESPONSABLES DE PROJETS :

Responsable des PFA : Antoine Rollet
Professeure encadrante : Myriam Desainte-Catherine
Client (INRIA) : Xavier Hinaut

Table des matières

I Frontend	3
1 Introduction	3
2 Packages	3
3 Structure du projet	4
4 Déploiement et débuguage	4
5 Ajout d'applications	4
6 Modèles jeux et réseaux	4
7 Page Play	5
7.1 Les jeux	5
7.2 Formulaire	5
7.3 Graphe	6
7.4 Lancement du jeu	7
8 Page Tutorial	7
9 Améliorations envisageables	7
9.1 Gestion dynamique du formulaire	7
9.2 Gestion en background du jeu	7
II Backend	7
10 Arborescence	8
11 Rajout d'un type de réseau	9

Première partie

Frontend

1 Introduction

Ce manuel a pour but de décrire succinctement l'architecture du logiciel et les méthodes de modification des différentes fonctionnalités possibles. Vous trouverez ici les différents packages utilisés, les moyens de changement de certaines fonctionnalités et les méthodes d'extension.

2 Packages

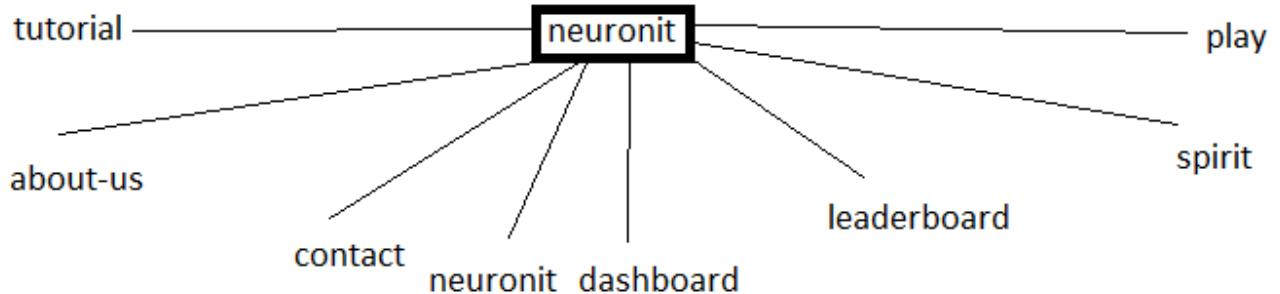
Ce projet se base sur différents packages :

-  django 1.9.8 par soucis de stabilité avec pinax (cf : ci-dessous)
-  pinax incluant Bootstrap 3.3.5, Font Awesome 4.4.0, jQuery 3.2.0. Ce package permet de déployer rapidement un site web incluant certaines fonctionnalités de base. Le site est basé sur *Pinax-Project-Account* lancé avec la commande : pinax start account "nomdusite", avec des modifications.

Informations : <http://pinaxproject.com/>.
- jQuery 3.2.0, pinax inclut jQuery 2.1.4 mais certaines pages de prenaient pas en compte jQuery 2.1.4, donc nous avons réimporté ce package.
- Spirit, une application de forum basé sur Django. Elle a été modifiée pour s'adapter à notre projet car elle n'a pas été conçue pour être une application réutilisable facilement.
Plus d'informations :<http://spirit-project.com/>
- tether.min.js qui se base sur bootstrap permettant l'affichage des pop-ups à coté du formulaire.
- d3.js version 3 permet de rendre la manipulation d'un svg souple et de gérer les données du graphe en utilisant notamment les fonctions pré-écrites par D3.
- Ainsi que divers autres dépendances dont on peut retrouver les informations à la racine du projet dans REQUIREMENTS.TXT

3 Structure du projet

Le projet est structuré de façon la plus classique par rapport aux projets basés sur Django bien que pinax ait une structure un peu différente. Ainsi on retrouve différentes applications granulaires présentes dans un dossier principal *neuronit* :



Chaque application est explicite sur sa fonctionnalité et contient les informations nécessaires aux modifications liées à sa fonctionnalité. On peut noter l'application *play* contenant le code de la page principale du jeu et l'application *neuronit* contenant la page "home" du site et les entêtes et pied de page ainsi que divers éléments js et css.

Il est à noter que contrairement aux dossiers standards, il y a un fichier *static* global à la racine du projet contenant notamment font-awesome et bootstrap. Ceci a été laissé tel quel avec le déploiement de pinax.

4 Déploiement et débuguage

la page Django <https://docs.djangoproject.com/fr/1.10/ref/settings/> constitue une bonne documentation pour le déploiement du site. En effet, beaucoup de cas sont traités et pour une bonne optimisation et sécurisation du projet, il est encouragé de regarder cette document avant déploiement. Il faut notamment veiller à mettre Debug à "False".

5 Ajout d'applications

Il est facile de rajouter des applications ou fonctionnalités comme le prévoit le framework Django :<https://docs.djangoproject.com/fr/1.10/intro/tutorial01/> Il ainsi encouragé de suivre la documentation Django dans l'optique d'ajouter des applications. Cette documentation nous indique de :

- lancer la commande `:python manage.py startapp nameapp` pour créer l'application name ;
- créer une vue dans `nameapp/views.py` et rajouter l'url dans `nameapp/urls.py`
- rajouter dans `neuronit/settings.py` dans `INSTALLED_APPS` "nameapp",

Via ces commandes, nous avons rajouté une application à notre projet, il suffit maintenant d'implémenter nos fonctionnalités.

Lors d'ajout d'applications, il est possible que Django demande de faire des migrations via : `./manage.py migrate`. Encore une fois, la documentation Django est bien fourni : <https://docs.djangoproject.com/fr/1.10/topics/migrations/>

6 Modèles jeux et réseaux

Il est possible de rajouter des jeux et réseaux dans la base de donnée qui seront alors modifier dynamiquement sur les pages du site : les "fixtures" Django permettent cela, elles permettent le pré-chargement de données en base de donnée avant déploiement.

Des dossiers fixtures sont présents dans les applications ayant besoin de préchargement.

Ainsi pour rajouter des éléments en base de donnée il faut les rajouter avec les informations nécessaires dans le fichier `play/fixtures/games.json` ou `play/fixtures/types.json` respectivement pour rajouter un jeu et rajouter un type de réseau.

Ensuite la commande `./manage.py loaddata games` permet de charger en base de données les informations contenues dans `play/fixtures/games.json` par exemple.

7 Page Play

7.1 Les jeux

Sur cette page il y a la liste des jeux auxquels on peut jouer qui sont donc modifiables à partir de `play/fixtures/games.json` :

The screenshot shows a list of games under a header "Games". The games listed are "CartPole-v0", "MountainCar-v0", and "MsPacman-v0", each preceded by a small icon.

7.2 Formulaire

Dans le formulaire se trouve différentes informations : la zone rouge entourée indique les zones d'information modifiables qu'on peut rajouter/modifier dans `play/fixtures/types.json` avec l'attribut "info" du modèle RESEAU.INFO.

The screenshot shows a configuration form titled "Parameters". It includes several input fields with dropdown menus and orange "?" buttons. A red box highlights the "More info" button above the parameters section.

Parameter	Value
Number of epochs:	10
Number of simulations:	10
Type:	Elman
Weight scaling:	
Victory reward:	
Defeat punishment:	
Hidden layers:	

More info

Parameters

Number of epochs: 10

Number of simulations: 10

Type: Elman

Weight scaling:

Victory reward:

Defeat punishment:

Hidden layers:

Modify

Chaque paramètre est chargée par rapport à la base de donnée et au modèle RESEAU contenant les paramètres à charger et leurs propriétés comme la valeur maximale autorisée, les textes des bulles d'aides, etc ...

Sur le clic de "Modify", le réseau correspondant est modifiée selon les champs remplis : des variables globales sont modifiées et réutilisées par la partie gérant le graphe du réseau, ainsi le graphe est modifiée en accordance au formulaire.

Ces champs sont hautement modifiables à convenance dans les parties :

- play/models.py à la classe *Reseau* et *ReseauForm* (la documentation des formulaires basés sur les modèles est disponible sur Django)
- play/forms.py à la class *GlobalParametersForm* pour modifier les paramètres globaux (epochs et simulations)

Enfin, le formulaire est dynamiquement traité par les fichiers "js" présents dans play/static/play/js/ et notamment par "main.js" qui fait notamment la gestion de l'affichage des champs selon le type sélectionné et la gestion des entrées de "Hidden layers" selon un format.

7.3 Graphe

Le graphe de jeu est écrit en Javascript (js) et est dépendant du package D3 (Data Driven Documents). L'application se trouvant au centre de la page play repose sur un unique fichier app.js. Au chargement de la page, une fonction écrite en js est appelée (start()). Son but est d'initialiser les variables de jeu et de lancer la fonction de routine (restart()). Cette dernière est appelée à chaque évènement ayant un impact sur le jeu. Les événements peuvent être soit des cliques sur noeuds (mousedown) ou des arêtes soit des passages de pointeur de souris sur les éléments (mouseover). Les évènements contraires ont été également ajouté afin d'annuler si besoin les actions.

La structure du graphe est contenu dans deux variables distinctes : **circles** pour les noeuds et **links** pour les arrêtes. Visuellement le jeu est représenté par un Scalable Vectors Graphics (svg). Ce svg est créé au préalable dans le code HTML. Une fois la boucle de routine lancée, le svg est automatiquement mis à jour et ajoute les éléments ou les supprime en fonction de ce que les variables links et circles contiennent. D3.js traduit les instructions js en HTML et représente les éléments de circles par cercles : balise <circle></circle> et les arrêtes par des chemins <path></path>. Au svg est associé un évènement permettant, si un des boutons de modifications du graphe est activé, de modifier le contenu des variables circles ou links. Les noeuds possèdent une structure différente dans circles et dans le svg. Dans la variable circle se trouve les informations de jeu : son identifiant, le numéro de colonne, le numéro et les paramètres définis dans le formulaire. Tandis que dans le svg, les attributs des noeuds sont la circonference du cercle, son type et sa position. Chaque cercle du svg possède également un attribut de style correspondant à son type. Tous les types de noeuds donnés (mlp, jordan, opérateur d'addition ...) ont été préalablement chargé dans le fichier play.html. Avec les fonctions de selections js il nous suffit d'associer la bonne image au bon noeud.

L'application possède également une feuille de style app.css et un fichier annexe utils.js qui regroupe



FIGURE 1 – Palette de bouton

Le quadrillage du jeu est crée à l'aide des fonctions fournis par D3.js. Nous pouvons avec ces fonctions créer des axes analogues à celle que l'on peut trouver dans le tracé d'une courbe. Les boutons sont des balises HTML dont sont rattachés des évènements onclick. La transcription des paramètres se fait à l'aide de la fonction *update_f*. Elle va chercher les éléments du DOM et récupère leur valeur. Inversement, une fonction refresh présent dans le fichier refresh.js s'occupe de charger les valeurs du

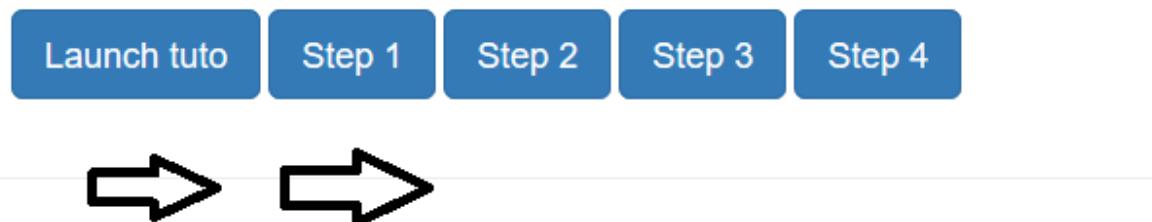
noeuds dans le formulaire. La fonction `send_to_backend` présente dans le fichier s'attelle à parser les éléments du graphe pour les envoyer au `back_end`.

7.4 Lancement du jeu

Un bouton play relie les données envoyées par le graphe à la vue dans `play/views.py` via *Ajax*, ces données sont traitées par un script python qui fait la liaison avec le "back end".

8 Page Tutorial

La page tutorial repose sur des clics dynamique de bouton "modals" qui viennent de bootstrap : à la chargement de la page un bouton est cliqué automatiquement et la suite des actions déclenchent les boutons suivants. Ainsi on a une succession de d'étapes.



Les fichiers "js" dans `tutorial/static/tutorial/js` gèrent le dynamisme et le codage est actuellement fait en dur dans la partie `templates/tutorial.html`.

A la fin du tutorial, un bouton "play" renvoie un graphe créé par l'utilisateur au même script python que dans la partie play et qui devrait renvoyé sur la page principale de jeu.

9 Améliorations envisageables

9.1 Gestion dynamique du formulaire

Bien que la gestion du formulaire a été pensée de façon optimale, certaines corrections de bugs influencées par le temps imparti du projet nous ont conduit à des solutions existantes mais peu élégantes pour certaines : notamment la gestion du champ `hidden layers/decomposite` qui est le même champ en base de donnée pour un traitement facile au niveau du back end. Ce champ change seulement de nom et de bulle d'aide selon le type sélectionné. Il faudrait plutôt créer des champs séparées et faire la liaison plus loin, par exemple au niveau du script python ou du traitement par le graphe. On peut retrouver cette gestion dans "main.js" cité précédemment.

9.2 Gestion en background du jeu

Après lancement du jeu par un utilisateur, le traitement tourne actuellement en pleine tâche du côté du serveur, le traitement de plusieurs utilisateurs n'a pas pu être testé sans le déploiement du serveur. Il est fort probable que l'on ait besoin d'une gestion en background avec possiblement le package CELERY : <http://www.celeryproject.org/>

Deuxième partie

Backend

10 Arborescence

La totalité de l'implémentation se trouve dans `back_end/`, à la racine du projet. L'arborescence de ce dossier se présente sous la forme suivante :

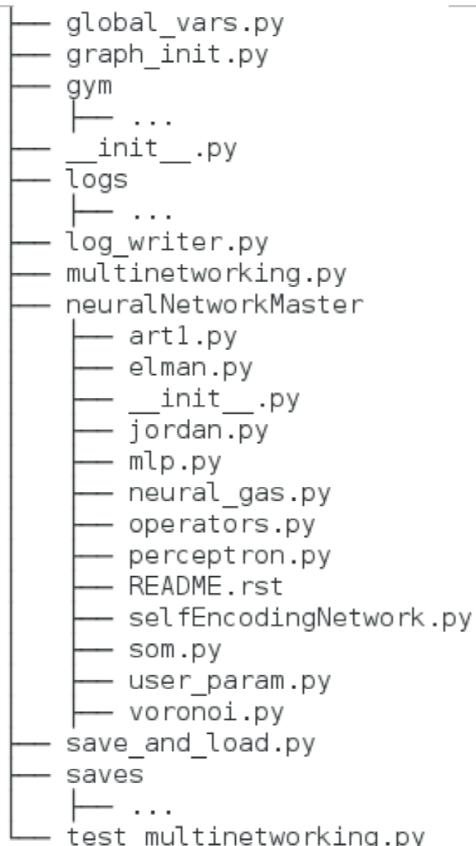


FIGURE 2 – Arborescence du dossier `back_end`.

Les fichiers présentés correspondent à :

- `global_vars.py` contient toutes les variables globales nécessaires au bon fonctionnement.
- `graph_init.py` contient les fonctions d'initialisation du graphe, à savoir l'initialisation en largeur, la création de nouveaux réseaux, et une fonction de recherche de cycle, qui n'est plus utilisée, dû à l'implémentation de la grille de jeu. Les justifications à cette décision se trouvent dans le rapport de projet.
- `gym/` contient les dépendances OpenAI Gym.
- `logs/` est le dossier voué à contenir les logs des tests si ceux-ci sont lancé depuis `back_end/`.
- `log_writer.py` permet la création de logs et l'écriture dans ces logs.
- `multinetworking.py` contient la fonction principale qui permet de lancer le jeu, la fonction de propagation dans le graphe, et la fonction d'apprentissage.
- `neuralNetworkMaster/` est le dossier contenant tous les réseaux de neurones et les opérateurs.
- `save_and_load.py` contient les fonctions de sauvegarde et de chargement.

— **saves/** est le dossier voué à contenir les logs des tests si ceux-ci sont lancé depuis **back_end/**.

Il est à noté que deux dossiers **logs/** et **saves** sont rajoutés à la racine du site, afin que les sauvegardes et les logs des utilisateurs y soient déposés.

11 Rajout d'un type de réseau

L'ajout d'un type de réseau peut se faire très facilement, il suffit en effet de déposer le fichier implémentant le réseau dans le dossier **back_end/neuralNetworksMaster**, de l'importer dans **graph_init.py** en suivant le modèle donné, et de rajouter les lignes nécessaires à l'intanciation dans la fonction **create_network()** en suivant ici aussi les instructions données en commentaires.



DEUXIÈME ANNÉE-FILIÈRE INFORMATIQUE
Bordeaux INP - Enseirb-Matmeca

NEURON IT ! - PFA

Backlog de produit

AUTEURS :

Thomas Caroff, Remicia Di Franco
Xavier Dussieux, Thomas Saux
Adnane Khattabi, Ahcene Mahtout
Nathan Nguyen

RESPONSABLES DE PROJETS

Antoine Rollet : Responsable des PFAs
Myriam Desainte-Catherine : Professeure encadrante
Xavier Hinault : Client (INRIA)

1 Acteurs du projet

Ce projet s'inscrit dans le cadre du PFA (projet au fil de l'année) proposé en 2ème année de la filière Informatique de l'école d'ingénieur ENSEIRB-MATMECA, entre octobre 2016 et avril 2017. Notre équipe, en charge du projet *Neuron It!* est composée de sept membres (Thomas Caroff, Remicia Di Franco, Xavier Dussieux, Adnane Khattabi, Ahcène Mahtout, Nathan Nguyen et Thomas Saux) et est encadrée par un responsable pédagogique : Myriam Desainte-Catherine, enseignante à l'ENSEIRB-MATMECA et chercheuse au LaBRI.

Le client, Xavier Hinaut, est chercheur à l'INRIA et membre de l'équipe Mnemosyne dont le sujet de recherche concerne les neurosciences computationnelles et l'apprentissage artificiel.

2 Contexte et définition du problème

Le cerveau humain possède des capacités d'apprentissage et de mémorisation très performantes mais aussi très complexes. La modélisation virtuelle du cerveau (on parle de réseaux de neurones artificiels) est aujourd'hui le sujet de beaucoup de recherches à travers le monde. Pour réussir à comprendre comment le cerveau apprend et mémorise, nous allons avoir recours à une approche appelée le *crowdsourcing*.

L'objectif de ce projet est de créer une interface web de crowdsourcing sur laquelle les internautes pourront manipuler des réseaux de neurones afin de résoudre des jeux proposés parmi ceux de la plate-forme Open AI Gym. Cette interface devra être intuitive de manière à ce que tout le monde puisse manipuler ces réseaux et résoudre les problèmes proposés sans avoir de connaissance particulière des réseaux de neurones.

Le crowdsourcing est un concept de production participative par l'utilisation de la créativité, de l'intelligence et du savoir-faire d'un grand nombre de personnes, en sous-traitance. Dans ce projet, nous espérons que le crowdsourcing permettra d'obtenir des solutions diverses et originales à un problème difficile malgré les possibilités de résolution numériques, tel que l'a illustré le jeu Foldit ; les résultats pourront alors être interprétés par des chercheurs et permettront de mieux comprendre comment fonctionne le cerveau. Par conséquent, le site sera disponible avant tout en anglais, pour atteindre le plus large public possible.

Si ce projet a un but professionnel, il n'en reste pas moins un projet d'étude pour notre équipe et est donc mené en parallèle des cours et autres projets. Un espace horaire a été dédié au projet de PFA (à savoir le mercredi de 10h à 16h), ce qui représente : entre le mercredi 23/11 et le mercredi 7/04, soit 6h par semaine et en considérant qu'aucun créneau ne soit réduit par l'ajout d'un cours ou d'une manifestation obligatoire, pour l'ensemble de l'équipe, environ 800h de travail. En fonction de la charge de travail demandé par le reste de l'occupation scolaire, la charge de travail réalisable est estimée entre 1000h et 1500h cumulées par l'équipe.

3 Liste des tâches

Organisation de la liste des tâches

La liste des tâches se présente sous la forme d'un tableau de tâches priorisé comportant quatre colonnes : scénario, priorité, effort et risque.

« Scénario » (ou « *User Story* »)

Un scénario est une exigence du système à développer formulée en une ou deux phrases dans le langage des utilisateurs pour servir un but. Sa granularité doit permettre à l'équipe de réalisation d'estimer son coût et de la réaliser entièrement à l'intérieur d'un sprint, c'est pour cela que les tâches doivent être indépendantes, négociables, évaluables, estimables, petites et testables.

Priorité

- ▶ **M** pour Must Have : DOIT être fait.
L'exigence est essentielle. Si elle n'est pas faite le projet échoue. On peut dire également priorité haute.
- ▶ **S** pour Should Have : DEVRAIT être fait.
Il s'agit d'une exigence essentielle, qu'il faut faire dans la mesure du possible. Mais si elle n'est pas faite, on peut la contourner et la livrer plus tard.
- ▶ **C** pour Could Have : POURRAIT être fait.
Il s'agit d'une exigence souhaitable. Elle pourrait être faite dans la mesure où elle n'a pas d'impact sur les autres tâches.
- ▶ **W** pour Won't Have : NE SERA (sans doutes) PAS fait.
Il s'agit d'une exigence luxueuse qui ne sera sans doute pas réalisable cette fois, mais intéressante et à garder pour la prochaine version.

Effort

Charge *estimée* de la tâche en homme.heure (charge de travail d'une personne pendant une heure). Une charge de travail peut à priori aussi bien être attribué à une seule personne comme à plusieurs, sans distinction supplémentaire.

Risque

Difficulté de la tâche (et donc incertitude sur la prédiction) sur une échelle de 1 à 3. Une difficulté de 1 correspond à une faible probabilité de rencontrer des complications, alors qu'une difficulté de 3 impliquera sûrement un dépassement de la durée estimée.

ROI

Le ROI, ou retour sur investissement, permet de déterminer l'ordre dans lequel seront effectués les tâches. Une tâche avec un ROI élevé sera effectuée avant une tâche avec un ROI faible. Celui-ci est déterminé à l'aide de la formule :

$$\frac{\text{Priorité}}{\text{Effort} * \left(1 + \frac{\text{Risque}-1}{5}\right)}$$

où la priorité d'une tâche M est associé à 50, S à 5, C à 1 et W à 0 afin d'obtenir des valeurs de ROI comprises entre 0 et 10.

Tâches

Authentification et sauvegarde

Scénario	Priorité	Effort	Risque	ROI
En tant qu'utilisateur, je veux pouvoir m'enregistrer/m'authentifier sur le site afin que je sois reconnu.	M	10	1	5
En tant qu'utilisateur, je veux que mes performances soient sauvegardées.	M	5	1	10
En tant qu'utilisateur, je veux pouvoir sauvegarder et charger mes constructions afin de pouvoir reprendre là où j'en étais.	M	30	2	1.19
En tant qu'utilisateur, je veux pouvoir m'identifier à partir de mon compte facebook/google+ afin de ne pas avoir à créer de compte.	C	5	1	0.2
En tant qu'utilisateur, je veux que l'on me rappelle de me connecter pour sauvegarder mes résultats avant que je quitte afin d'éviter que je perde mes données.	S	2	1	2.5

Phase d'apprentissage

Scénario	Priorité	Effort	Risque	ROI
En tant qu'utilisateur débutant, je veux pouvoir faire un tutoriel à difficulté progressive afin d'apprendre en jouant.	M	40	3	0.89
En tant qu'utilisateur débutant, je veux qu'un bouton tutoriel me propose lorsque je suis en jeu de revenir vers un tutoriel adapté au problème posé afin que je puisse mieux comprendre comment jouer.	S	5	2	0.83
En tant qu'utilisateur, je veux avoir une page m'expliquant le but du projet.	S	5	1	1
En tant qu'utilisateur, je veux avoir accès une section qui rassemble les connaissances utiles de base des tutoriels afin de jouer sans revoir les tutoriels.	S	10	1	0.5
En tant qu'utilisateur débutant, je veux que des conseils et suggestions me soient proposés afin que je comprenne plus rapidement comment jouer.	C	15	2	0.06

Retour des joueurs

Scénario	Priorité	Effort	Risque	ROI
En tant que client, je veux pouvoir obtenir un retour sur les activités des utilisateurs afin d'avancer dans mes recherches.	M	30	2	1.39
En tant que client, je veux trouver un agencement de réseaux qui résolve tous les jeux afin d'avancer dans mes recherches.	M	50	3	0.71
En tant qu'utilisateur, je veux pouvoir noter le jeu afin d'informer les autres joueurs de la qualité du jeu.	S	1	1	5
En tant qu'utilisateur, je veux pouvoir faire remonter mon avis au développeur à travers un onglet "Contactez-nous" afin qu'ils puissent améliorer tel ou tel aspect du jeu (feedback).	S	1	1	5

Communauté

Scénario	Priorité	Effort	Risque	ROI
En tant qu'utilisateur, je veux communiquer avec les autres joueurs à travers un chat/forum.	S	15	2	0.28
En tant qu'utilisateur, je veux pouvoir ajouter d'autres joueurs en ami afin de partager mes performances avec eux et de communiquer avec eux plus facilement.	S	1	1	5
En tant qu'utilisateur curieux, je veux pouvoir me renseigner plus amplement sur les réseaux de neurones afin de comprendre les mécanismes du jeu.	S	5	2	0.83
En tant qu'utilisateur, je veux pouvoir discuter des solutions des autres joueurs sur un espace personnel leur étant dédié.	S	20	3	0.18
En tant qu'utilisateur, je veux que mes performances soient traduites sous la forme d'un score afin de le comparer à ceux des autres joueurs.	S	10	2	0.42
En tant qu'utilisateur, je veux avoir accès aux solutions des autres joueurs afin de m'en inspirer pour améliorer mes performances.	S	10	2	0.42
En tant qu'utilisateur, je veux pouvoir partager le jeu sur les réseaux sociaux.	C	1	1	1

Interface

Scénario	Priorité	Effort	Risque	ROI
En tant que client, je veux avoir un site fonctionnel permettant à l'utilisateur de jouer au moins à un jeu.	M	80	1	0.625
En tant que client, je veux que les réseaux de neurones puissent être paramétrés facilement et visuellement afin de donner de la liberté à l'utilisateur quel qu'il soit.	M	50	2	0.83
En tant que client, je veux que différents réseaux de neurones puissent être liés entre eux facilement et visuellement par l'utilisateur quel qu'il soit afin qu'il puisse résoudre un problème.	M	50	2	0.83
En tant qu'utilisateur, je veux que l'interface de jeu soit intuitive afin que je m'y retrouve facilement et que mon expérience de joueur soit moins désagréable (ne pas avoir à chercher pour trouver comment je dois faire ceci ou cela)	M	30	3	1.19
En tant qu'utilisateur, je veux que certaines tâches répétitives soient automatisées afin d'éviter que je perde mon temps.	S	50	2	0.08
En tant que client, je veux que l'interface soit agréable afin que le site soit utilisé par plus de personnes.	S	30	3	0.12
En tant qu'utilisateur confirmé, je veux pouvoir décocher une case "bulles d'aide" afin que mon expérience de jeu ne soit pas perturbée.	C	1	1	1
En tant qu'utilisateur, je veux que certaines tâches redondantes soient allégées (si elles ne peuvent pas être automatisées) afin de me faire gagner du temps.	C	5	2	0.17
En tant qu'utilisateur, je veux que les raccourcis usuels se retrouvent dans l'interface de jeu.	C	5	2	0.17

Fluidité du site

Scénario	Priorité	Effort	Risque	ROI
En tant que client, je veux que mon programme s'exécute dans des temps raisonnables (quelques heures) afin de pouvoir utiliser correctement le produit.	M	20	2	2.08
En tant que client, je veux que le serveur soit stable et fluide même lorsque plusieurs utilisateurs sont en ligne, afin qu'ils puissent continuer à jouer.	S	30	3	0.12
En tant qu'utilisateur, je veux que le programme s'exécute rapidement (quelques minutes) afin de pouvoir utiliser correctement le produit.	C	30	3	0.02

Modes de jeux/type d'utilisateurs

Scénario	Priorité	Effort	Risque	ROI
En tant qu'utilisateur, je veux jouer à plusieurs jeux différents afin de diversifier les défis.	M	40	3	0.89
En tant qu'utilisateur non curieux, je veux pouvoir jouer sans être "embêté" par la finalité des jeux.	S	30	1	0.17
En tant que client, je veux introduire un système de gratification afin de motiver davantage les utilisateurs.	S	40	3	0.09
En tant qu'utilisateur non-anglophone, je veux pouvoir utiliser des langues autres que l'anglais afin d'améliorer mon expérience de jeu.	C	10	1	0.1

4 Planning des sprints et rendu de projet

Le planning des sprints représente une vision anticipée du travail de développement à réaliser durant le projet. La charge de travail étant divisée en des périodes de travail accélérées d'une durée variant entre une et quatre semaines appelées **Sprint**.

Sprint	Date début	Date fin	But	livrable
Sprint1	1 Décembre	17 Décembre	Implémentation d'une première interface du site simplifiée avec un seul exemple de jeu.	Maquette du site, Notebook fonctionnel.
Sprint2	22 Janvier	5 Février	Amélioration de l'interface du site en ce qui concerne le paramétrage des réseaux de neurones et introduction des tâches d'authentification.	Interface v1/primaire interactive, Notebook avec interactivité.
Sprint3	6 Février	20 Février	Résolution du problème de sauvegarde de l'avancement des joueurs et implémentation d'un système de récupération de données des retours des joueurs.	Interface avec système de sauvegarde, Notebook amélioré.
Sprint4	21 Février	7 Mars	Gamification de l'interface du site et implémentation de tutoriel à difficulté progressive.	Interface v2 et tutoriel.
Sprint5	12 Mars	26 Mars	Implémentation des fonctionnalités secondaires et amélioration des fonctionnalités du site en fonction des retours d'utilisateurs extérieurs.	Fonctionnalités supplémentaires.

Un intervalle de temps séparant le premier et deuxième **Sprint** a été prévu à cause de la charge de travail de cette période en dehors du projet PFA : Nombreux rendu de projet et partiels sont prévus durant cette période.

La date de rendu du rapport du projet est fixée au 7 avril. A ce propos, on a envisagé la fin du mois de mars comme date limite du projet afin de prévoir assez de temps pour la finalisation du rapport ainsi que l'ensemble des délivrables.