HLIN601

Projet de Programmation

—

Simulateur Deep

—

Melvin Bardin
Maël Bonneaud
Malika Lin-Wee-Kuan

—

Encadré par M. Poncelet

—

2019-2020

Table des matières

1 Introduction

Dans le cadre du module HLIN601 de Projets de Programmation de L3, nous nous sommes orientés vers le sujet "Simulateur Deep". Le sujet de M. Poncelet sur le Deep Learning nous a intéressé car il s'agissait d'un sujet dont on ne connaissait que très peu de choses et dont nous étions curieux d'en apprendre d'avantage sur celui-ci.
En effet les réseaux de neurones sont souvent considérés comme des boîtes noires, elles prennent des données, modifient des poids, mais en aucun cas nous ne voyons comment tout cela est mis en place pour obtenir une classification. Dans notre projet, nous nous situons dans de l'apprentissage supervisé. En d'autres termes, nous considérons que les classes ont un label et l'objectif est justement de créer un modèle qui soit capable d'apprendre à partir de ces classes.
Notre objectif était donc de réaliser une application Web permettant de simuler le comportement d'un réseau de neurones, pour en expliciter le comportement, mieux le saisir et en découvrir les subtilités.

Le code ainsi qu'une vidéo de démonstration sont disponibles à l'adresse :
" https://github.com/Kyrial/Simulateur_Deep"

Le reste du rapport est organisé de la manière suivante. Afin de mieux comprendre l'application que nous avons réalisé, la section 2 présente un rappel sur les réseaux de neurones qui nous permet de mettre en évidence les principaux composants mais également la manière dont ceux-ci fonctionnent. La section 3 décrit les technologies utilisées. Dans la section 4, nous présentons le travail réalisé durant la conception de l'application.
Ensuite dans la section 5 nous verrons comment on peut interpréter les résultats obtenus par l'application. Nous continuerons ensuite sur comment le projet s'est déroulé et comment nous nous sommes organisés dans la section 6. Nous terminerons tout cela par une conclusion et par une ouverture sur les différents milieux d'application des réseaux de neurones dans la section 7.

2 Les Réseaux de neurones

2.1 L'histoire des réseaux de neurones

Les réseaux de neurones sont de plus en plus présents en informatique, que se soit pour la reconnaissance faciale à la classification des données en passant par la prédiction météorologique. Ils existent réellement que depuis le XX$^{\text{ème}}$ siècle. On pourrait alors se demander quand les réseaux de neurones ont commencés ? La première théorie avec le neurone formel est apparue en 1943 et il a fallu attendre les années 2000 pour voir les prémisses dans l'informatique.

2.2 Rappel sur les réseaux de neurones

Interrogeons nous sur le fonctionnement d'un réseau de neurones qui sera important pour la compréhension de notre projet. Un neurone, comme en biologie, est une cellule qui va prendre en entrée des signaux (ici une donnée) et qui va renvoyer en sortie des signaux différents (via ce qu'on peut appeler des synapses). Un réseau de neurones est une organisation de plusieurs neurones.

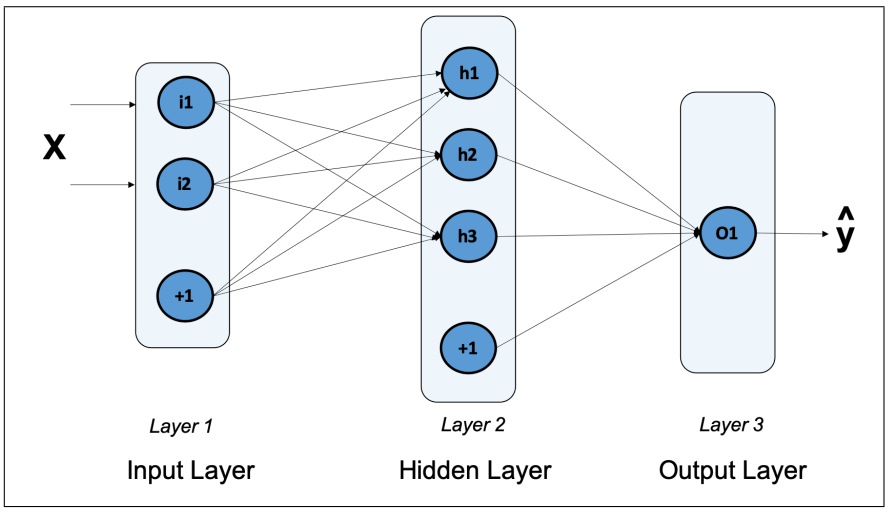


FIGURE 1 – Les différentes types de couches

La figure 1 illustre un réseau de neurones composé de 3 *layers*. Le premier *layer* (généralement noté *Layer 0 ou 1*) et appelé *input layer* correspond à la couche d'entrée du réseau et contiendra les variables prédictives. Le dernier (cf figure 1 Layer 3) correspond à l'*output layer* et donne le résultat de la classification. Au milieu nous avons les *hidden layer* (cf figure 1 Layer 2) qui eux vont permettre de modifier les valeurs récupérées et c'est là que l'apprentissage du réseau de neurones s'effectue.

2.3 Apprentissage du réseau de neurone

Pour apprendre, le réseau de neurone va avoir besoin de plusieurs tentatives et de nombreuses données. En effet les signaux (ou données) envoyés entre chaque synapses vont être modifiés avec ce qu'on appelle un poids, qui va selon sa valeur activer ou non le neurone en récupérant le signal. Ensuite le neurone va appliquer à ce signal un biais qui permettra de favoriser (ou pas) l'activation de la synapse dans certains cas. Après, chaque neurone a ce qu'on appelle une fonction d'activation qui va permettre de déterminer l'activation (ou pas) de la synapse.
Il existe de nombreuses fonctions d'activation comme la "sigmoïde" et la "ReLU" que nous utilisons dans notre projet. Ce signal est ensuite transmis au neurone suivant jusqu'à arriver au dernier layer (*"Output Layer"*) de sortie.

Dans un premier temps le jeu de données est divisé en deux parties, les données d'évaluation serviront de vérifications des performances du modèle tout au long des calculs et les données d'apprentissage permettrons l'apprentissage de ce modèle.
Pour apprendre, le réseau va donc en premier lieu utiliser des valeurs aléatoires car il ne saura pas quel poids appliquer pour arriver à un résultat correct. Lors du premier passage (*Forward propa-*

gation), la valeur de sortie est fausse par rapport au résultat attendu. Nous allons donc appliquer ce qu'on appelle une fonction de coût qui sera plus où moins élevée si on est éloigné ou non du résultat attendu. Cette fonction s'appliquera sur chaque neurone dans le sens inverse (*Backward propagation*). Chaque nouvelle propagation est appelée époque (*epoch*).
Ainsi nous cherchons à avoir un coût le plus faible possible pour s'approcher du résultat attendu. C'est pour cela que nous utilisons la technique de la descente de gradient qui utilise la dérivée partielle afin d'obtenir le plus petit coût possible pour chaque neurone. Cette fonction est associée à un taux d'apprentissage (*learning rate*) qui correspond à la vitesse de convergence de la descente de gradient vers le minimum de la fonction coût. Ainsi le réseau de neurone apprendra petit à petit. Notre objectif consiste à partir de tous les composants d'un réseau de neurones de mettre en évidence comment se propagent les informations (mises à jour des poids) pour parvenir à une classification.

3 Technologies utilisées

Durant notre projet nous avons utilisé de nombreux outils technologiques répondant à nos besoins. Nous allons voir l'utilisation de Jupyter, Python, *My Neural Network* et enfin HTML.

3.1 Jupyter notebook

Dans un premier temps nous avons utilisé Jupyter notebook. Le bloc-notes Jupyter [1] est une application Web open source qui permet de créer et de partager des documents contenant du code en direct, des équations, des visualisations et du texte narratif. Nous l'avons donc principalement utilisé pour la présentation des algorithmes en lien avec les réseaux de neurones.

3.2 Python (Flask)

Ensuite tous les algorithmes générant les réseaux de neurones utilisés lors de notre projet utilisent Python. Cette technologie est un langage de programmation contenant de nombreuses librairies qui nous ont été très utiles pour notre projet. Nous avons donc pour la conception de notre site utilisé une libraire Python dont le nom est Flask [2] , qui nous a permis de mettre en place un serveur Python afin d'exécuter nos algorithmes.
Après avoir testé différentes technologies pour exécuter nos scripts Python, nous nous sommes orientés vers Flask pour héberger notre serveur car c'est celui qui correspondait au mieux à nos besoins. En effet Flask est compatible avec les machines de la faculté et il est léger, ce qui est parfait pour notre projet qui était déjà un peu lourd de base.

1. https ://jupyter.org/[dernière consulation le 23/04/2020].
2. https ://flask.palletsprojects.com/ [dernière consulation le 23/04/2020].

3.3 My Neural Network

My Neural Network[4] est un groupement de fonctions Python utilisant les outils de calculs de réseaux de neurones (Keras, Tensorflow, etc..) fournit par notre référent permettant d'expliciter les étapes de calcul pour mieux cerner leurs fonctionnements et leurs utilités.

3.4 HTML / CSS / JAVASCRIPT

Concernant les technologies Web utilisées pour la conception de notre site nous avons utilisé HTML et CSS pour la partie *frontend* (visuel : conception / interface) du projet. Ensuite nous avons utilisé principalement Javascript pour modifier le DOM et utiliser les données renvoyées par le serveur, afin de construire le réseau de neurones et afficher les données, les courbes et la simulation du fonctionnement du réseau de neurones. Concernant Javascript nous avons également utilisé AJAX afin d'empêcher le rechargement de la page Web lors de l'exécution des algorithmes Python.

4 Travail réalisé

La conception de notre site Web s'est axée sur un serveur qui effectue les calculs et une page web qui récupère les résultats et les affiches graphiquement. Le serveur renvoie ces données à travers des tableaux et coté Web, des Json.

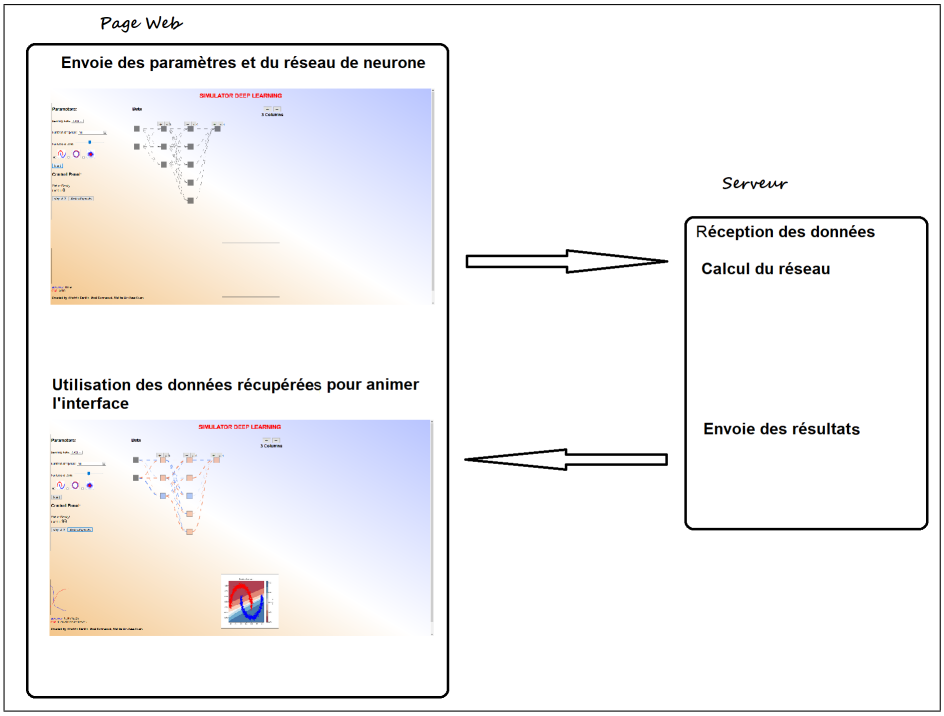


FIGURE 2 – Interaction entre page Web et Serveur

La figure 2 illustre l'architecture mise en place. Elle repose sur un serveur, une application qui calcule le réseau de neurones et une page web qui permet de gérer les paramètres et de simuler le

réseau. Les différentes étapes sont décrites dans les sous sections suivantes.

4.1 Entrées et paramètres de l'application

À l'exécution, aucune entrée n'est nécessaire au bon fonctionnement du site Web. Lorsqu'une requête est envoyée au serveur pour effectuer des calculs, tous les paramètres ainsi que la configuration du réseau de neurones sont passés en paramètre grâce à un formulaire qui vérifie l'intégrité des données.

```
Number of epoch:  100
Learning rate:  0.001
Valid_size : 0.5
Config neural network:  ['2', '3', '5', '1']
```

FIGURE 3 – Réceptions des données côté serveur

La figure 3 illustre un exemple de passage des paramètres qui sont envoyés au modèle. Comme nous pouvons le constater, tous les paramètres (nombre de noeuds par couche, nombre de couches) sont présents pour créer dans un premier temps le modèle et les autres (*learning rate, epochs, validation_size*) sont alors utilités pour lancer l'exécution du modèle.

Après le traitement, le serveur retourne les résultats de l'algorithme sous forme de trois tableaux. Ces derniers sont réceptionnés au format Json et correspondent à l'historique des calculs permettant d'effectuer l'animation des neurones ainsi que de dessiner les courbes de précision et de coût. L'image s'affichant en fin d'animation n'est pas retournée avec la réponse du serveur, elle est sauvegardée sur la machine puis est récupérée par la page Web.

```
▸ Array(50) [ {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, … ]
▸ Array(50) [ 0.167578125, 0.190234375, 0.196484375, 0.198046875, 0.21015625, 0.2109375, 0.209375, 0.2171875, 0.229296875, 0.23828125, … ]
▸ Array(50) [ 0.7818785399251145, 0.7348829616695993, 0.7145613315189627, 0.7117079550877129, 0.7093771538417855, 0.708475920498522, 0.706807067061
```

FIGURE 4 – Réponse du serveur au site Web

La figure 4 illustre un exemple de données retournées par le serveur. Le premier tableau est un tableau de dictionnaire contenant tous les poids de chaque arrête du réseau de neurones de toutes les époques (ici 50 époques). Ces données pouvant être très volumineuses, peuvent prendre un certain temps a être transférées (dans certains cas, elles peuvent dépasser le mégaoctets). Les deux autres tableaux correspondent respectivement à la précision *(accuracy)* et le coût *(cost)* pour chaque époque.

4.2 L'interface graphique

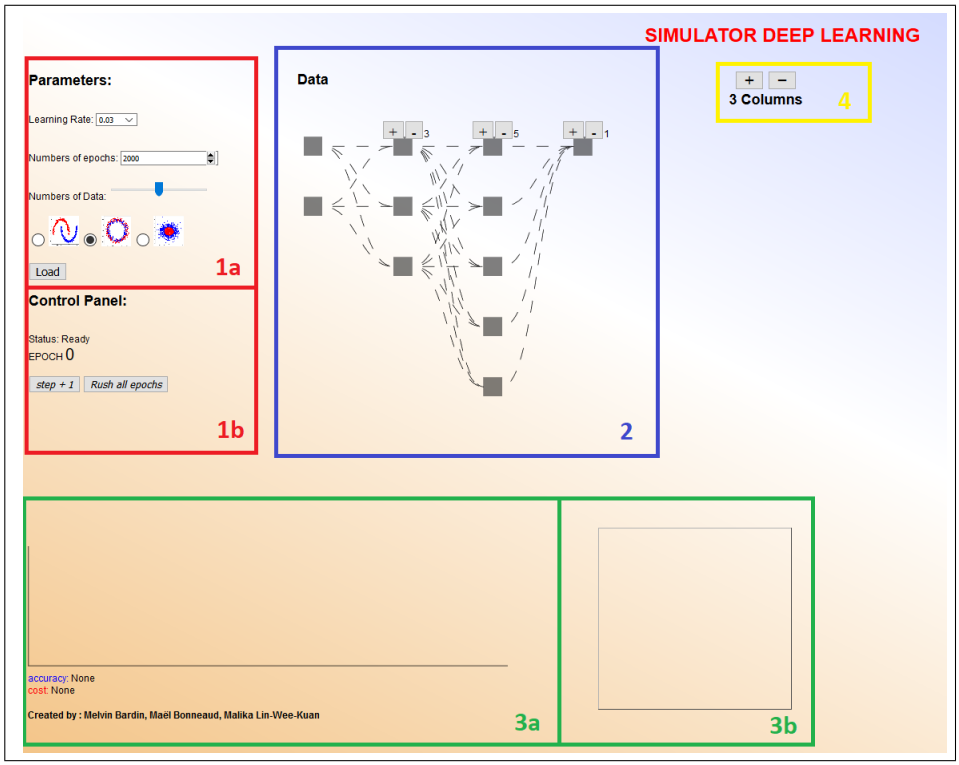


FIGURE 5 – L'interface de l'application

Notre site Web est composé d'une seule page. Celle ci est composée d'une zone d'édition (c.f. Figure 5.2) permettant de configurer le réseau de neurones que l'on souhaite.
On peut ajouter des neurones dans les *layers* avec les boutons "+" et "-", sachant qu'il faut que le dernier layer soit à 1. Il est aussi possible de rajouter des *layers* avec les boutons "+" et "-" (c.f. Figure 5.4).
Nous avons aussi un cadre permettant de renseigner les paramètres de calcul (c.f. Figure 5.1a). On peut y choisir son jeu de données, le nombre de données, le taux d'apprentissage ainsi que le nombre d'époques. Tous ces paramètres sont très importants pour faire fonctionner le réseau de neurones. Le bouton *"Load"* permet de charger tout ces paramètres une fois qu'ils ont été choisis.
En dessous, nous avons également un panneau de contrôle (c.f. Figure 5.1b) pour visualiser l'avancement des époques et lancer l'animation.
Dans la partie inférieure, deux courbes se dessinent (c.f. Figure 5.3a) permettant de visualiser la précision des calculs ainsi que le coût durant l'évolution des *Epochs*. Nous obtenons ainsi le résultat à la fin sous forme d'image des calculs effectués (c.f. Figure 5.3b).On pourra ainsi visualiser la frontière de décision.

4.2.1 Courbes

Lors de l'animation, la taille des courbes ainsi que la couleur ont un rapport avec le poids. L'épaisseur est proportionnelle à la valeur absolue du poids et sera bleue si négative et rouge si

positive. La couleur des neurones est basée sur la moyenne des poids des courbes entrantes.

4.3 Performance et efficacité du logiciel

Les temps de calculs côté serveur pouvant être assez longs, nous avons utilisé un formulaire asynchrone avec Ajax afin d'éviter d'avoir la page figée le temps de la requête et d'avoir le réseau de neurones qui se remette à sa version par défaut.

5 Analyse des Résultats

Dans cette partie, nous analysons les résultats obtenus via notre simulateur. En premier temps on va voir une analyse d'un réseau simple de neurones et ensuite nous analyserons plusieurs configurations du réseau de neurones.

5.1 Analyse d'un réseau simple de neurones

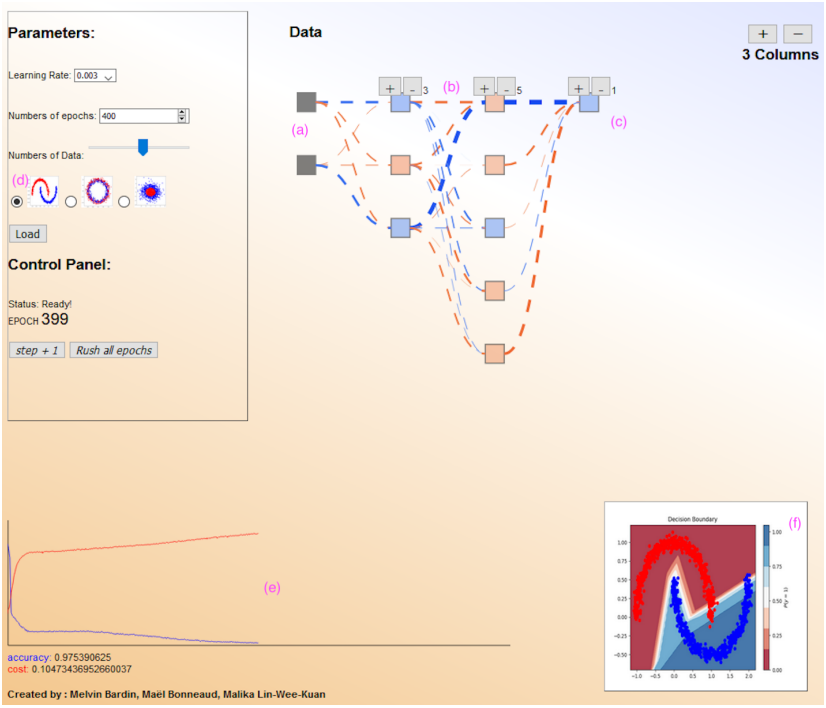


FIGURE 6 – Réseau de neurones de base

Ce réseau de neurones est composé de deux neurones d'entrées (c.f. figure 6a), deux couches de neurones cachées (c.f. figure 6 b), et un neurone de sortie (c.f. figure 6 c).
Les calculs ont été effectués sur le jeu de données *"MakeMoons"* (c.f. figure 6d). On peut remarquer qu'après 400 époques, la précision (la courbe bleu *"accuracy"*) tend vers 1 et le coût (la courbe rouge *"cost"*) tend vers 0 (c.f. figure 6e). Ce qui signifie que le résultat est concluant, de plus la frontière de décision (*decision boundary*) entre les deux classes (c.f. figure 6f : donnée rouge et donnée bleu) sont plutôt bien dessinées. Nous avons donc une justification numérique et graphique pour

certifier le résultat obtenu.
Portons maintenant notre attention sur les deux courbes (c.f. figure 6e). Pour rappel, les valeurs choisies à la première époque sont aléatoires. Ici la précision est faible, donc très loin de la solution, ce qui signifie que le coût sera élevé et fera converger rapidement vers la solution lors des époques suivantes.
Il est intéressant de constater que les courbes tendent très vite vers la solution puis soudainement la progression devient nettement plus lente. On peut donc en déduire que plus le résultat est proche de la solution, plus la vitesse de convergence du coût est lente.

5.2 Analyse des configurations du réseau de neurones

Nous pouvons remarquer que des couches semblent redondantes car les valeurs exprimées sur les différents arcs se répètent de manières symétriques et donc ces couches n'apportent pas vraiment de modification. Elles sont mises en évidence dans la figure 8. Cette information peut aider le concepteur du modèle à se poser des questions sur le fait qu'il y ait trop de *layers*.

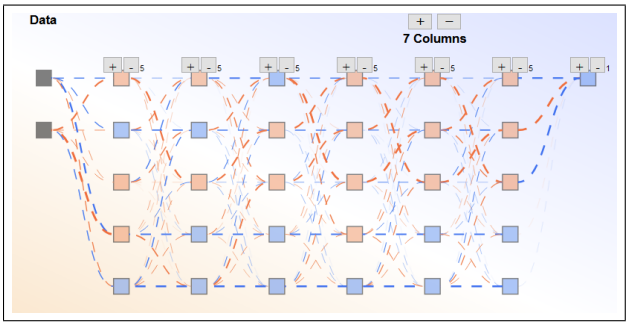


FIGURE 7 – Réseau avec des layers redondantes

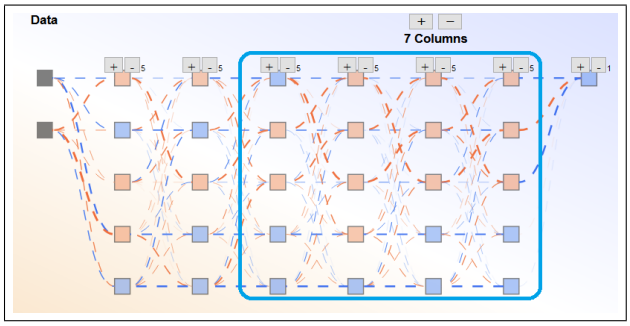


FIGURE 8 – Mise en évidence de layer redondantes

Si nous diminuons le nombre de *layers* comme l'illustre la figure 9, nous constatons que *l'accuracy* de la figure 11 n'a pas bougé et que justement *l'accuracy* reste tout à fait correcte par rapport au réseau avec 7 *layers*.

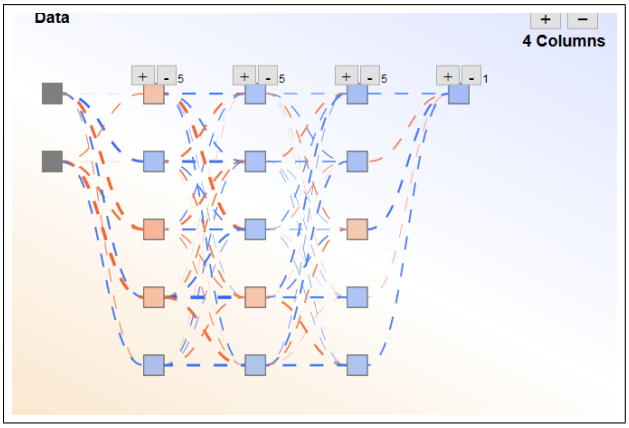


FIGURE 9 – Réseau avec layers non redondants

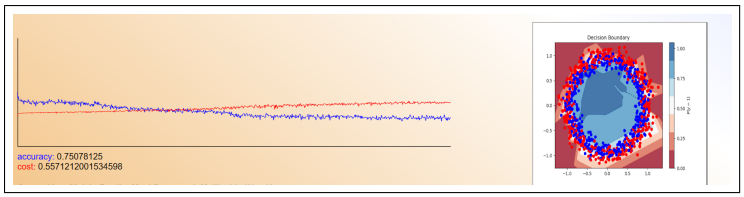


FIGURE 10 – Résultat avec layers redondantes

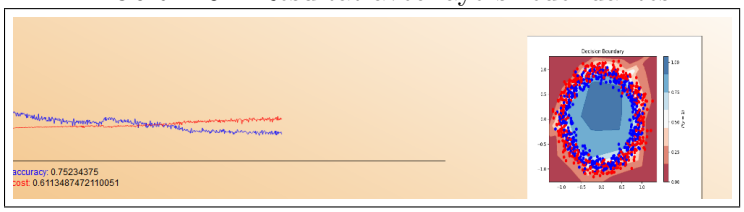


FIGURE 11 – Résultat avec layers non redondants

Concernant le nombre de neurones par couches, il est moins évident de constater un motif répétitif mais il semblerait que certains neurones soient moins actifs voir même pas activés (figure 13 cadre bleu), ils n'influenceraient pas la solution et peuvent être enlevés.

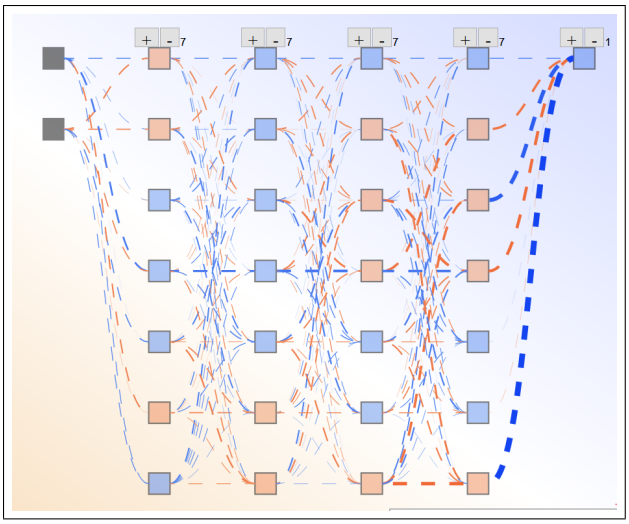


FIGURE 12 – Réseau avec des neurones inactifs

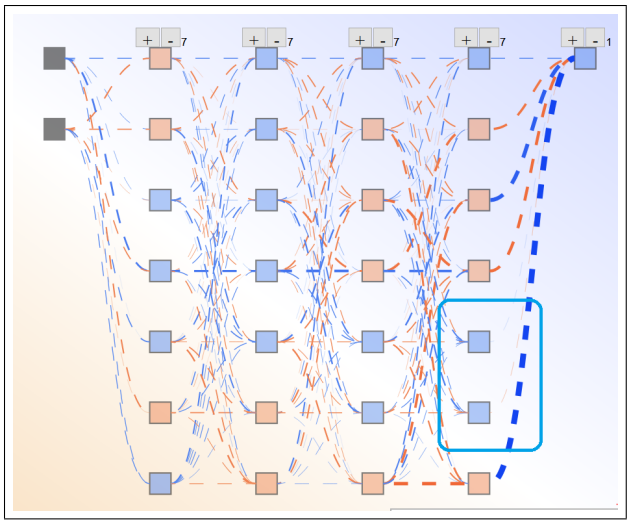


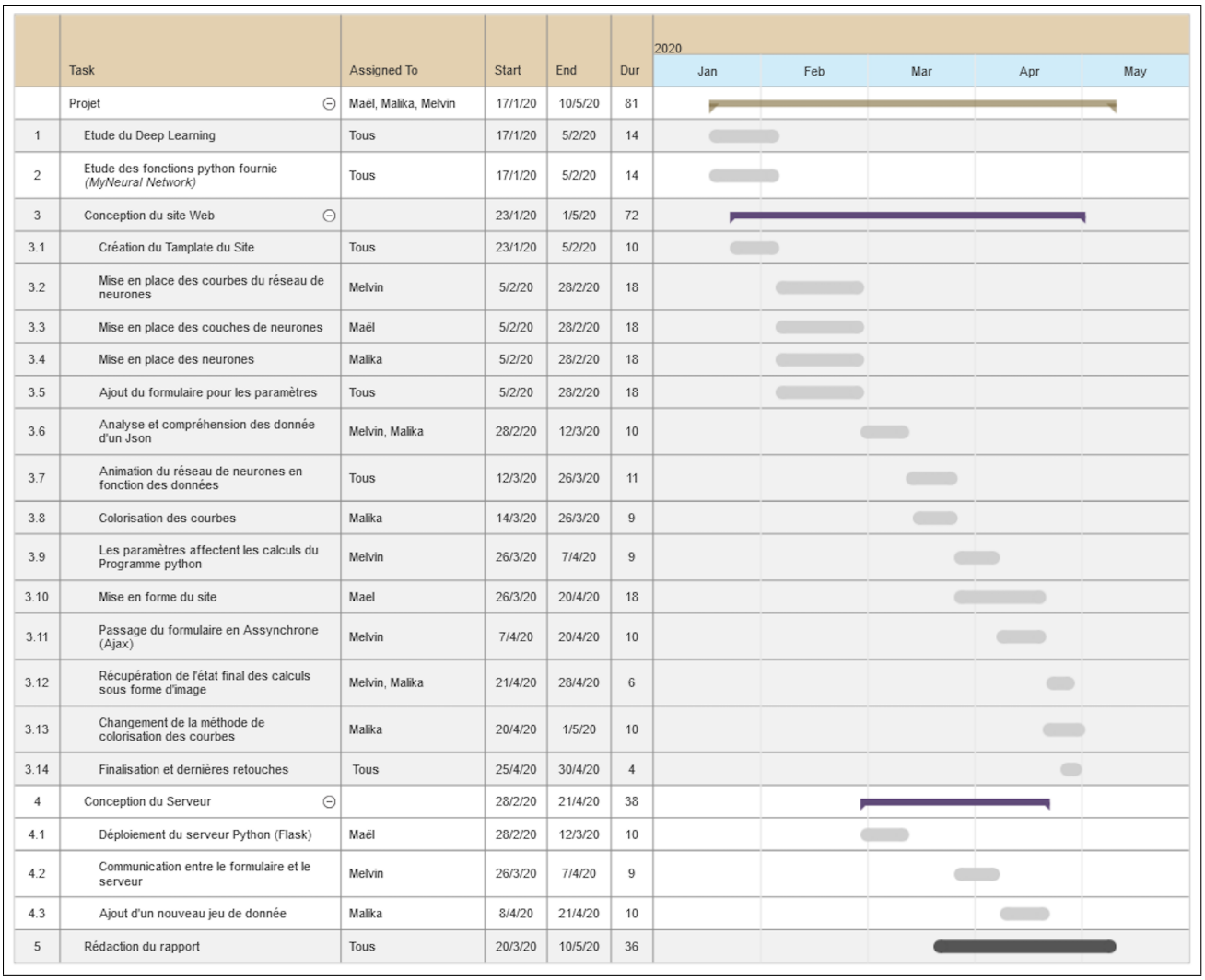
FIGURE 13 – Mise en évidence de neurones inactifs

Comme nous l'avons vu précédemment, le fait de pouvoir analyser les évolutions permet de mieux comprendre comment les résultats finaux sont obtenus. Cependant, cette application permet également d'avoir des informations importantes sur la composition même du réseau.

6 Gestion du Projet

6.1 Le diagramme de Gantt

Nous avons utilisé un diagramme de Gantt pour nous organiser, ce dernier répertorie toutes les tâches à accomplir pour mener le projet à bien, et indique la date à laquelle ces tâches doivent être effectuées (le planning). La colonne de gauche du diagramme énumère toutes les tâches à effectuer, tandis que la ligne d'en-tête représente les unités de temps les plus adaptées au projet (jours, semaines, mois etc). Chaque tâche est matérialisée par une barre horizontale, dont la position et la longueur représentent la date de début, la durée et la date de fin.
Voici notre diagramme de Gantt du projet :



	Task	Assigned To	Start	End	Dur
	Projet	Maël, Malika, Melvin	17/1/20	10/5/20	81
1	Etude du Deep Learning	Tous	17/1/20	5/2/20	14
2	Etude des fonctions python fournie *(MyNeural Network)*	Tous	17/1/20	5/2/20	14
3	Conception du site Web		23/1/20	1/5/20	72
3.1	Création du Tamplate du Site	Tous	23/1/20	5/2/20	10
3.2	Mise en place des courbes du réseau de neurones	Melvin	5/2/20	28/2/20	18
3.3	Mise en place des couches de neurones	Maël	5/2/20	28/2/20	18
3.4	Mise en place des neurones	Malika	5/2/20	28/2/20	18
3.5	Ajout du formulaire pour les paramètres	Tous	5/2/20	28/2/20	18
3.6	Analyse et compréhension des donnée d'un Json	Melvin, Malika	28/2/20	12/3/20	10
3.7	Animation du réseau de neurones en fonction des données	Tous	12/3/20	26/3/20	11
3.8	Colorisation des courbes	Malika	14/3/20	26/3/20	9
3.9	Les paramètres affectent les calculs du Programme python	Melvin	26/3/20	7/4/20	9
3.10	Mise en forme du site	Mael	26/3/20	20/4/20	18
3.11	Passage du formulaire en Assynchrone (Ajax)	Melvin	7/4/20	20/4/20	10
3.12	Récupération de l'état final des calculs sous forme d'image	Melvin, Malika	21/4/20	28/4/20	6
3.13	Changement de la méthode de colorisation des courbes	Malika	20/4/20	1/5/20	10
3.14	Finalisation et dernières retouches	Tous	25/4/20	30/4/20	4
4	Conception du Serveur		28/2/20	21/4/20	38
4.1	Déploiement du serveur Python (Flask)	Maël	28/2/20	12/3/20	10
4.2	Communication entre le formulaire et le serveur	Melvin	26/3/20	7/4/20	9
4.3	Ajout d'un nouveau jeu de donnée	Malika	8/4/20	21/4/20	10
5	Rédaction du rapport	Tous	20/3/20	10/5/20	36

FIGURE 14 – Diagramme de Gantt du Projet

6.2 Trello

Trello est une application web permettant d'organiser l'avancement d'un projet très simplement. En effet Trello permet de gérer les différentes étapes de développement , ou d'avancement du projet sous formes de tableaux, de cartes etc... L'avancement du projet devient donc beaucoup plus clair et simple, il est facile pour les développeurs de savoir ce qu'il y a à faire et ce qui a été fait par les autres.

6.3 Exemple d'échanges lors du projet

mael bonneaud <mael190399@hotmail.fr>
À pascal.poncelet@lirmm.fr, melvin.bardin@etu.umontpellier.fr, malika.lin-wee-kuan@etu.umontpellier.fr

Compte Rendu de la réunion du 28 Février.
Lieu: département informatique.
Présent: Melvin Bardin, Maël Bonneaud, Malika Lin-Wee-Kuan.

Point abordés:
- Réponses aux questions sur le serveur python
- Explication de la communication entre les programmes python et le site web HTML
- Explication des ajouts à apportés et de comment les mettre en place (variables renvoyé par les programmes python à utiliser)

Décisions:
Construction du site Web avec :
- Backward Propagation
- couleur des courbes de Béziers
- Mise en place de la communication des programmes python avec le site Web
- Améliorer les fonctionnalités du site (ajout des paramètres , ajout des jeux de données ...)

Documents/code échangés:
Fichier historique envoyé par mail après la réunion.

Date et heure prochaine réunion : Jeudi 12 Mars 2020 à 11h30
Lieu de la prochaine réunion : département informatique.

Cordialement,
Bonneaud Maël.

FIGURE 15 – Compte rendu durant le projet

7 Conclusions

7.1 Problèmes rencontrés / Améliorations possibles

Durant la réalisation de notre application web, nous avons rencontrés diverses difficultés. En effet nous avons du choisir entre plusieurs technologies possibles lors de la conception du site. Nous devions faire des choix et ce n'était pas toujours évident de savoir quelle technologie serait la meilleure. Par exemple pour faire notre serveur nous avons hésité entre *FileZilla* ou *Wamp*, mais notre choix s'est finalement porté sur Flask qui répondait davantage à nos besoins.

Actuellement nous avons également un soucis au niveau de SVG. En effet les courbes semblent être inversées car pour les canevas SVG l'axe des Y augmente vers le bas donc cela entraîner une confusion.

En ce qui concerne les améliorations possibles, nous pourrions toujours ajouter davantage de jeux de données (dont des données non binaires), plus de paramètres, et des améliorations graphiques. Il serait tout à fait possible de permettre des configurations beaucoup plus avancées comme les réseaux de neurones récurrents. Cela consiste à avoir des boucles dans le réseau, ainsi les *"Forward Propagation"* s'effectuent dans les deux sens. Ce genre de configuration sont utiles lorsque les données ne sont pas indépendantes les unes des autres. Toutefois il serait intéressant de pouvoir optimiser la méthode de transmission des données pour minimiser les temps d'attente et permettre de charger des modèles plus conséquents.

7.2 Bilan

Dans ce rapport, nous avons présenté une application qui simule le fonctionnement d'un réseau de neurone et permet de montrer ce qu'il se passe à l'intérieur de celui-ci. Notre application utilise *MyNeuralNetwork* pour générer le réseau de neurones ainsi que les résultats de celui-ci. Elle utilise également différentes librairies présentes dans le *Read me*[8] du projet pour davantage d'informations sur les librairies et technologies utilisées. Avec ce type d'application, on peut visualiser ce qu'il se passe dans un réseau de neurones et ainsi éviter de se fier au réseau de neurones sans savoir pourquoi il aboutit à un résultat (problème de la boîte noire).

Malgré les problèmes et les difficultés que nous avons rencontrés (et vu précédemment), nous sommes satisfaits de notre travail.

Pour réaliser ce projet nous avons utilisé divers moyens de communications et de partage de documents, entre autre "Discord", Gantt, Trello et le dépot "Git".

Le *Deep Learning* est utilisé dans de nombreux domaines :

Robotique Ce qui a permis aux robots de détecter et de réagir à leur environnement.

Agriculture Permet de déployer des équipements capables de repérer et de différencier les plantes cultivées et les mauvaises herbes.

Étude d'images Ce qui ramène à différents points :

- Classification d'image/donnée (par exemple : savoir si la photo représente un chat)
- Reconnaissance faciale en Asie.
- Conduire une voiture
- **Imagerie Médicale**

  Permet la reconnaissance de cancer sur une radio, et permet d'assister dans les décisions médicales

  Dans la situation actuelle, à l'aide d'une photo permet de savoir si une personne à de la fièvre ou non

8 Bibliographie

Références

[1] The SciPy COMMUNITY. *numpy.ndarray.* URL : `https://www.miximum.fr/blog/introduction-au-deep-learning-2/` (visité le 07/05/2020).

[2] DEVELOPERS.GOOGLE. *Optimiser le taux d'apprentissage.* URL : `https://developers.google.com/machine-learning/crash-course/fitter/graph` (visité le 08/05/2020).

[3] Thibault JOUANNIC. *Deep learning : la rétropropagation du gradient.* URL : `https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html` (visité le 07/05/2020).

[4] Pascal PONCELET. *My Neural Network (Jupyter NoteBook).* (Visité le 08/05/2020).

[5] *Réseau de neurones artificiels : qu'est-ce que c'est et à quoi ça sert ?* URL : `https://www.lebigdata.fr/reseau-de-neurones-artificiels-definition` (visité le 10/05/2020).

[6] Florent SIMON. *Deep Learning, les fonctions d'activations.* URL : `https://www.supinfo.com/articles/single/7923-deep-learning-fonctions-activation` (visité le 08/05/2020).

[7] Daniel SMILKOV et Shan CARTER. *Simulateur Deep Learning.* URL : `https://playground.tensorflow.org/` (visité le 07/05/2020).

[8] Fonctionnement du site WEB. *Voir le 'Read me' du git.* URL : `https://github.com/Kyrial/Simulateur_Deep` (visité le 10/05/2020).

Table des figures