HLIN601

Projet de Programmation

—

Simulateur Deep

—

Melvin Bardin
Maël Bonneaud
Malika Lin-Wee-Kuan

—

Encadré par M. Poncelet

—

2019-2020

Table des matières

Première partie

Présentation du projet

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 nous étions curieux d'en apprendre d'avantage sur celui-ci.

1.1 L'histoire des Réseaux de neurones

En effet les réseaux de neurones sont de plus en plus présent 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 est ce que les réseaux de neurones ont commencés ? La première théorie avec le neurone formel est apparu en 1943 et il a fallu attendre les années 2000 pour voir les prémisse dans l'informatique.

1.2 Fonctionnement d'un réseau de neurone

Interrogeons nous sur le fonctionnement d'un réseau de neurone 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 neurone c'est une organisation de plusieurs neurones.
En effet nous allons avoir trois catégories de neurones dans un réseaux de neurones. Chaque catégorie sera appelée *"layer"* (ou couche en français). Nous aurons d'abord les *"input layer"* qui prendrons des données qui seront ensuite transmise au layer suivant.
À la fin, le dernier layer c'est celui qui en sorti va nous permettre de voir avec son résultat si il a identifié correctement la donnée (reconnaissance correct ou non d'un chat par exemple). Entre ces deux *"layers"* il y a ce qu'on appelle les *"Hidden Layers"* qui eux vont permettre de modifier les valeurs récupérées et c'est là que l'apprentissage du réseau de neurone s'effectue.

1.3 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 va être modifié 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 neurones 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 fonction 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 de sortie.

Dans un premier temps le jeu de donnée est divisé en deux parties, les données d'évaluation servirons de vérifications des performance du modèle tout au long des calculs et les donnée 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 correcte. Lors du premier passage (Forward propagation), 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.

1.4 Un cas concret

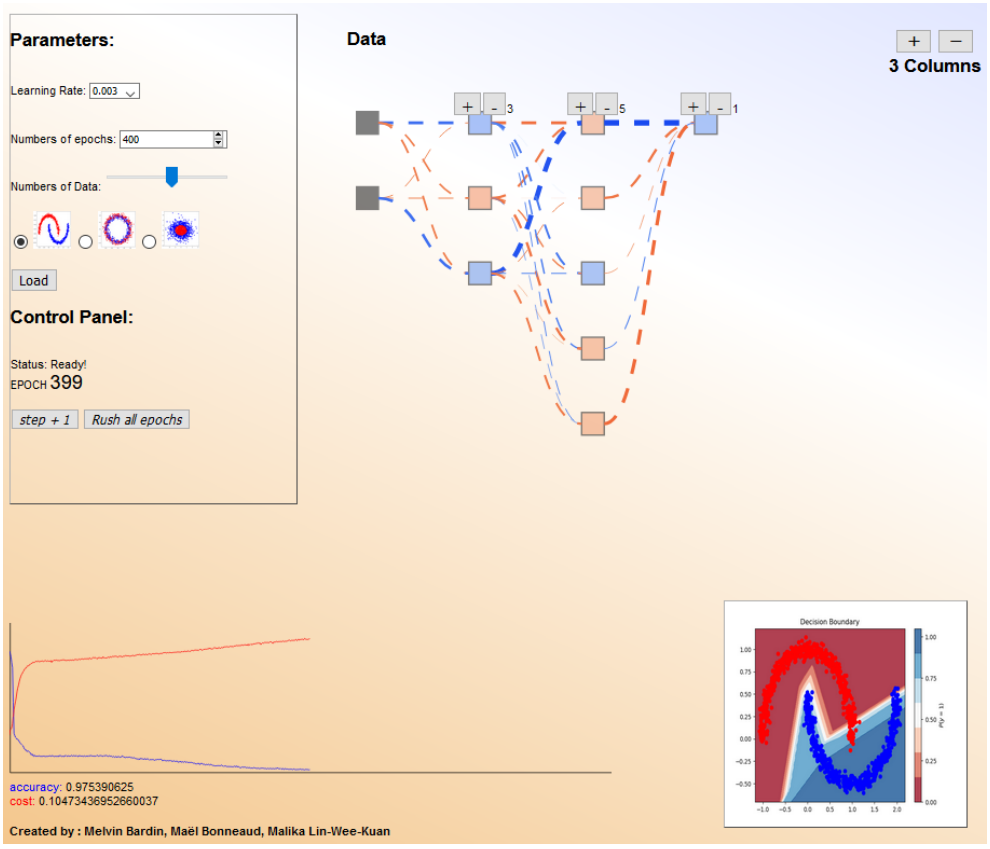

FIGURE 1 – Exemple concret issu de notre site Web

Visualisons concrètement les précédentes informations avec un exemple pour mieux comprendre. Le réseau de neurones est composé de deux neurones d'entrée, deux couches de neurones cachées,

et un neurone de sortie.
Les calculs ont été effectué sur le jeu de donnée *"MakeMoons"*. On peut remarquer qu'après 400 époques, la précision (la courbe bleu *"accuracy"*) tend vers 1 et le coup (la courbe rouge *"cost"*) tend vers 0. Ce qui signifie que le résultat est concluant, de plus la frontière de décision (*decision boundary*) entre les deux classes (donnée rouge et donnée bleu) s'est plutôt bien dessinés. Nous avons donc une justification Numérique et graphique pur certifier le résultat obtenue.
Portons maintenant notre attention sur les deux courbes. Comme dit précédemment, les valeurs choisies à la première époque sont aléatoire, 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 suivante. Plus le résultat est proche de la solution, plus la vitesse de convergence du coût sera lente.

2 Technologies utilisées

Durant notre projet nous avons dû utiliser une multitude de technologies répondant à nos besoins. Nous allons voir l'utilisation de Jypyter, Python, My Neural Network et enfin HTML.

2.1 Jupyter notebook

Dans un premier temps nous avons utilisé Jupyter notebook. Le bloc-notes Jupyter 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.

2.2 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 étaient très utiles pour notre projet. Nous avons donc pour la conception de notre site utilisé une libraire Python dont le nom est Flask , qui nous a permit de mettre en place un serveur Python afin d'exécuter nos algorithmes.
Après avoir testé différentes technologies pour exécuter nos script Python, nous nous sommes orienté 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.

2.3 My Neural Network

My Neural Network 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.

2.4 HTML / CSS / JAVASCRIPT

Concernant les technologies Web utilisées pour la conception de notre site nous avons utilisés HTML et CSS pour la partie *frontend* (visuelle : 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 neurone et afficher les données, les courbes et la simulation du fonctionnement du réseau de neurone. Concernant Javascript nous avons également utilisé AJAX afin d'empêcher le rechargement de la page Web lors de l'exécution des algorithmes Python.

Deuxième partie
Développements Logiciel

3 Conception, Modélisation et Implémentations

3.1 Conception

La conception de notre site Web s'est axé sur un serveur qui effectue les calculs et une page web qui récupère les résultats et les affiches graphiquement.

3.2 L'interface graphique

Notre site Web est composé d'une seule page. Celle ci est composé d'une zone d'édition permettant de configurer le réseau de neurone que l'on souhaite, d'un cadre permettant de renseigner les paramètres de calcul et d'un panneau de contrôle pour visualiser l'avancement des époques et de lancer l'animation. Dans la partie inférieure, deux courbes se dessinent permettant de visualiser la précision des calculs et le résultat sous forme d'image des calculs effectués.

3.3 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 passées en paramètre grâce à un formulaire qui vérifie l'intégrité des données.
Après le traitement, le serveur retourne les résultats de l'algorithme sous forme de trois dictionnaires. Ces derniers 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.

3.4 Statistique

4 Algorithmes et Structures de Données

4.1 Les structures de données

Les principales structures de données sont du coté serveur des dictionnaires et coté Web, des Json. En effet lorsque le serveur effectue les calculs, il garde les valeurs comme trace dans trois dictionnaires pour l'envoyer sous forme de Json au site web. Ce dernier va les extraire pour effectuer le rendu graphique. Ces données pouvant être très volumineuse, dépassant le méga-Octet dans certains cas, les générer et les transférer sont les premières causes de ralentissement observé.

4.2 les Algorithmes

5 Analyse des Résultats

5.1 Performance et efficacité du logiciel

Les temps de calculs côté serveur pouvant être assez long, 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.

5.2 Analyse du réseau de Neurones

Il a été constaté que lorsque le nombre de couches de neurones est importants, un motif répétitif se perçoit. On peut donc supposer que les couches concerné par ce schéma n'apporte pas vraiment en efficacité ou précision. En effet cela montre que l'information entre ces mêmes motifs est plus ou moins la même et que sans ces répétition le réseau aurait obtenue la même conclusion.

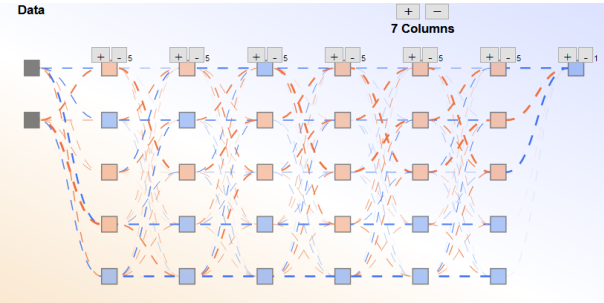

FIGURE 2 – Reseau avec des layers redondantes

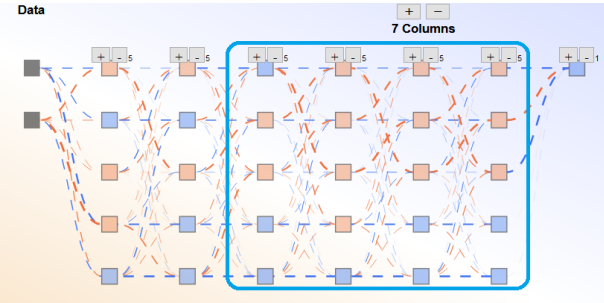

FIGURE 3 – Mise en évidence de layer redondantes

Concernant le nombre de neurones par couche, il est moins évident de constater un motif répétitif mais il semblerait que certains neurones soit moins actives voir même pas activées, ils n'influenceraient pas la solution et peuvent être enlevés.

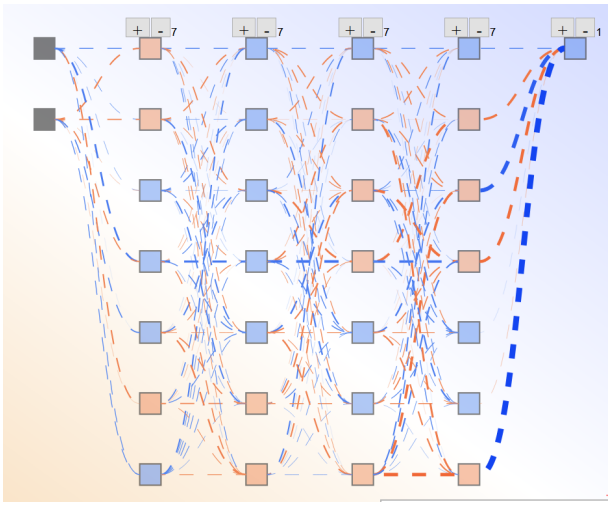

FIGURE 4 – Reseau avec des neurones inactifs

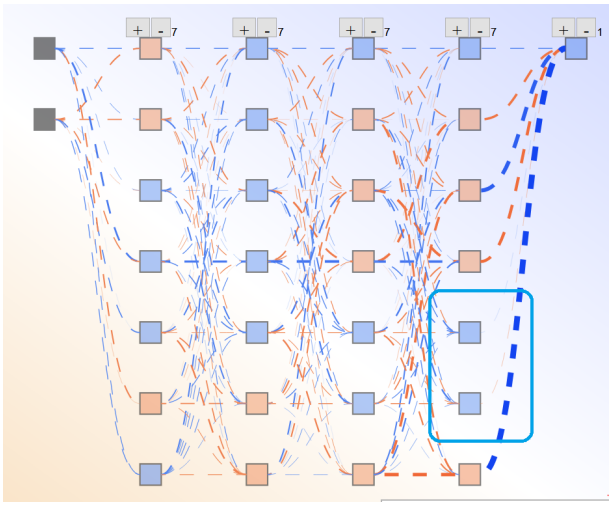

FIGURE 5 – Mise en évidence de neurones inactifs

Il a été remarqué aussi qu'un nombre trop faible de neurones ou de couches retourne une solution peu précise voir carrément fausse malgré un nombre d'époque beaucoup plus élevé.
On peux donc supposer qu'il existe un nombre de couche et de neurones par couche optimale permettant d'arriver à la conclusion sans qu'il n'y ai de motif répétitif ni de neurone inactif en un nombre d'époque le plus petit possible.

Troisième partie

Gestion du Projet

6 Le diagramme de Gantt

En résumé, un diagramme de Gantt 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.

	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 6 – Diagramme de Gantt du Projet

7 Trello

Trello est une application web permettant d'organiser l'avancement d'un projet très simplement. En effet Trello permet de gérer les différents 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 était fais par les autres.

Quatrième partie

Bilan

8 Conclusions

9 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] 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).

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

10 Annexes